



UNIVERSIDAD DE MÁLAGA



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA NFORMÁTICA

GRADUADO EN INGENIERÍA INFORMÁTICA

Evaluación del uso de LLMs en el proceso de desarrollo  
software

Evaluation of the use of LLMs in the software  
development process

Realizado por  
Salvador Ortega Vargas

Tutorizado por  
Carlos Manuel Rossi Jiménez

Departamento  
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA

MÁLAGA, septiembre de 2025



UNIVERSIDAD  
DE MÁLAGA



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADUADO EN INGENIERÍA INFORMÁTICA

## **Evaluación del uso de LLMs en el proceso de desarrollo software**

### **Evaluation of the use of LLMs in the software development process**

Realizado por  
**Salvador Ortega Vargas**

Tutorizado por  
**Carlos Manuel Rossi Jiménez**

Departamento  
**Lenguajes y Ciencias de la Computación**

UNIVERSIDAD DE MÁLAGA  
MÁLAGA, SEPTIEMBRE DE 2025

Fecha defensa: septiembre de  
2025

# Resumen

La inteligencia artificial generativa revoluciona de manera constante el sector de la ingeniería del software al automatizar tareas, resolver problemas de programación cada vez más complejos, asistir en tareas de documentación y pruebas de código, adquiriendo un papel cada vez más relevante en fases del desarrollo de aplicaciones software debido a su capacidad para proporcionar soluciones acertadas y de alta calidad. Ante el auge de su popularidad, han surgido LLMs gratuitos y de código abierto que buscan igualar su rendimiento con aquellos modelos de inteligencia artificial comerciales.

Se pretende realizar una evaluación de distintos modelos gratuitos para analizar su rendimiento y eficacia en diferentes fases del desarrollo de aplicaciones. La forma de evaluar estos modelos será a través de un caso práctico propuesto que también servirá para comparar el desempeño de los LLMs con respecto al desarrollo que haría una persona. Para ello, se utilizarán tres métricas de evaluación: completitud, corrección y rendimiento. El caso práctico que servirá como base para evaluar a los modelos consistirá en el desarrollo de una aplicación web para una empresa ficticia del sector turístico. Este será desarrollado de forma dual, es decir, primero por el autor y luego por los LLMs seleccionados para el estudio. Este enfoque permitirá poner en práctica distintas fases del proceso de desarrollo de software y servirá como base para realizar un análisis comparativo del rendimiento entre el desarrollo por un humano y el asistido por LLMs.

**Palabras clave:** Inteligencia Artificial Generativa, Código Abierto, LLMs, Desarrollo de software

# Abstract

Generative artificial intelligence is continuously revolutionizing the software engineering sector by automating tasks, solving complex programming problems, and assisting with documentation and code testing tasks. It is taking on an increasingly relevant role in various phases of software application development due to its ability to provide accurate and high-quality solutions. In response to its growing popularity, free and open-source LLMs have emerged, aiming to match the performance of commercial artificial intelligence models.

Therefore, the motivation behind this Final Degree Project arises from the appearance of free LLMs that aim to perform similar functions and offer performance comparable to commercial options. The objective is to evaluate different free models to analyze their performance and effectiveness across various phases of application development. These models will be evaluated through a proposed practical case, which will also serve to compare the performance of LLMs with that of a human developer. For this purpose, three evaluation metrics will be used: completeness, correctness, and performance. The practical case used as the basis for evaluating the models will involve the development of a web application for a fictional company in the tourism sector. This application will be developed in a dual manner, that is, first by the author of this project and then by the selected LLMs for the study. This approach will allow the practical application of different phases of the software development process and will serve as the foundation for a comparative analysis of the performance between human development and LLM-assisted development.

**Keywords:** Generative Artificial Intelligence, Open Source, LLMs, Software development



# Índice

<b>Índice de figuras</b>	<b>9</b>
<b>Índice de cuadros</b>	<b>15</b>
<b>1. Introducción</b>	<b>23</b>
1.1. Motivación . . . . .	23
1.2. Objetivos . . . . .	23
1.3. Metodología y tecnologías . . . . .	24
1.3.1. Metodología Scrum . . . . .	24
1.3.2. Tecnologías y herramientas utilizadas . . . . .	25
1.4. Estructura del documento . . . . .	25
<b>2. Marco metodológico y estado del arte</b>	<b>29</b>
2.1. Contexto . . . . .	29
2.2. Proceso de desarrollo software . . . . .	31
2.2.1. Identificación y especificación de requisitos . . . . .	31
2.2.2. Análisis y modelado del sistema . . . . .	33
2.2.3. Diseño y prototipado . . . . .	34
2.2.4. Implementación . . . . .	36
2.2.5. Pruebas de software . . . . .	38
2.3. Métricas de evaluación de LLMs . . . . .	40
2.4. Retos del uso de LLMs . . . . .	42
<b>3. Análisis y Diseño</b>	<b>45</b>
3.1. Caso práctico . . . . .	45
3.1.1. Funcionalidades de la aplicación . . . . .	45
3.1.2. Enunciado . . . . .	47
3.2. Catálogo de requisitos . . . . .	48
3.3. Casos de uso . . . . .	51

3.4.	Modelos . . . . .	55
3.5.	Diagramas de secuencia . . . . .	59
3.6.	Formatos de pantalla . . . . .	61
3.7.	Uso de LLMs . . . . .	63
3.7.1.	Llama 3.1 . . . . .	66
3.7.2.	Mistral . . . . .	78
3.8.	Evaluación de resultados . . . . .	90
3.8.1.	Compleitud . . . . .	90
3.8.2.	Corrección . . . . .	95
3.8.3.	Rendimiento . . . . .	106
3.8.4.	Evaluación de resultados . . . . .	107
<b>4.</b>	<b>Implementación</b>	<b>113</b>
4.1.	Arquitectura software . . . . .	113
4.1.1.	Frontend . . . . .	116
4.1.2.	Backend . . . . .	117
4.1.3.	Base de datos . . . . .	118
4.2.	Implementación software . . . . .	121
4.2.1.	API REST . . . . .	121
4.2.2.	Seguridad . . . . .	122
4.2.3.	Procesos de negocio implementados . . . . .	123
4.3.	Uso de LLMs . . . . .	127
4.3.1.	Llama 3.1 . . . . .	131
4.3.2.	Mistral . . . . .	135
4.4.	Evaluación de resultados . . . . .	140
4.4.1.	Compleitud . . . . .	140
4.4.2.	Corrección . . . . .	146
4.4.3.	Rendimiento . . . . .	152
4.4.4.	Evaluación de resultados . . . . .	152
<b>5.</b>	<b>Pruebas de software</b>	<b>157</b>
5.1.	Pruebas realizadas . . . . .	158

5.1.1. Pruebas manuales . . . . .	158
5.1.2. Pruebas E2E . . . . .	161
5.2. Uso de LLMs . . . . .	165
5.3. Evaluación de resultados . . . . .	168
5.3.1. Completitud . . . . .	168
5.3.2. Corrección . . . . .	169
5.3.3. Rendimiento . . . . .	171
5.3.4. Evaluación de resultados . . . . .	171
<b>6. Evaluación global</b>	<b>173</b>
6.1. Resultados por fase . . . . .	173
6.1.1. Análisis y Diseño . . . . .	174
6.1.2. Implementación . . . . .	176
6.1.3. Pruebas de software . . . . .	177
6.2. Evaluación global . . . . .	178
<b>7. Conclusiones y Líneas Futuras</b>	<b>187</b>
7.1. Conclusiones . . . . .	187
7.2. Líneas Futuras . . . . .	189
<b>Bibliografía</b>	<b>191</b>
<b>Apéndice A. Prompts y salidas</b>	<b>195</b>
A.1. Análisis y diseño . . . . .	195
A.1.1. Llama 3.1 . . . . .	195
A.1.2. Mistral . . . . .	248
A.2. Implementación software . . . . .	283
A.2.1. Llama 3.1 . . . . .	283
A.2.2. Mistral . . . . .	287
A.3. Pruebas software . . . . .	290
<b>Apéndice B. Desarrollo manual</b>	<b>301</b>
B.1. Análisis y Diseño . . . . .	301

- B.1.1. Catálogo de requisitos . . . . . 301
- B.1.2. Casos de uso . . . . . 310
- B.1.3. Diagramas de secuencia . . . . . 316
- B.1.4. Formatos de pantalla . . . . . 322
- B.2. Implementación . . . . . 328
  - B.2.1. Registro de usuario y Login . . . . . 328
  - B.2.2. Gestión de empleados . . . . . 329

# Índice de figuras

1.	Diagrama del contenido adicional del documento. Fuente: elaboración propia.	27
2.	Diagrama de la estructura principal del documento. Fuente: elaboración propia.	28
3.	Muestra de requisitos funcionales obtenidos tras el análisis del caso práctico. Fuente: elaboración propia. . . . .	49
4.	Muestra de requisitos no funcionales obtenidos tras el análisis del caso práctico. Fuente: elaboración propia. . . . .	50
5.	Requisitos de información obtenidos tras el análisis del caso práctico. Fuente: elaboración propia. . . . .	50
6.	Diagrama de casos de uso: Responsable y Empleado. Fuente: elaboración propia.	51
7.	Diagrama de casos de uso: Monitor y Cliente. Fuente: elaboración propia. . . .	52
8.	Diagrama de casos de uso: Usuario Anónimo (no registrado). Fuente: elaboración propia. . . . .	52
9.	Diagrama de modelo de clases. Fuente: elaboración propia. . . . .	56
10.	Diagrama de modelo de clases de control. Fuente: elaboración propia. . . . .	57
11.	Diagrama de modelo de interfaces. Fuente: elaboración propia. . . . .	57
12.	Diagrama de modelo físico de datos. Fuente: elaboración propia. . . . .	59
13.	Secuencia de registro de nuevo usuario en el sistema. Fuente: elaboración propia.	60
14.	Maqueta de pantalla de página de actividades disponibles. Fuente: elaboración propia. . . . .	61
15.	Maqueta de información de actividad seleccionada. Fuente: elaboración propia.	62
16.	Diagrama de estructura del uso de LLMs en las fases 1, 2 y 3. Fuente: elaboración propia. . . . .	63
17.	Comparación visual de resultados en la completitud de requisitos, casos de uso y su especificación respecto al desarrollo manual. Fuente: elaboración propia.	107
18.	Comparación visual de resultados en la completitud de diagramas de secuencia respecto al desarrollo manual. Fuente: elaboración propia. . . . .	108
19.	Comparación visual de resultados en la completitud de maquetas de pantalla respecto al desarrollo manual. Fuente: elaboración propia. . . . .	108

20.	Distribución de categorías de corrección en el análisis de requisitos funcionales obtenidos por cada modelo. Fuente: elaboración propia. . . . .	109
21.	Comparación visual de completitud de requisitos no funcionales por cada modelo. Fuente: elaboración propia. . . . .	109
22.	Distribución de categorías de corrección de casos de uso y actores obtenidos por cada modelo. Fuente: elaboración propia. . . . .	110
23.	Distribución de categorías de corrección de especificación de casos de uso obtenidos por cada modelo. Fuente: elaboración propia. . . . .	110
24.	Distribución de categorías de corrección de diagramas de secuencia obtenidos por cada modelo. Fuente: elaboración propia. . . . .	111
25.	Distribución de categorías de corrección de formatos de pantalla obtenidos por cada modelo. Fuente: elaboración propia. . . . .	112
26.	Flujo general de proceso típico del sistema. Fuente: elaboración propia. . . . .	115
27.	Flujo completo del sistema realizado con Mermaid.live[31]. Fuente: elaboración propia. . . . .	115
28.	Flujo típico de procesamiento de solicitud HTTP por parte de un cliente hacia el sistema realizado en draw.io [32]. Fuente: elaboración propia. . . . .	118
29.	Diagrama E/R de la base de datos. Fuente: elaboración propia . . . . .	119
30.	Ejemplo de token de acceso JWT decodificado. Fuente: elaboración propia. . . . .	123
31.	Implementación <i>Guards</i> en <i>backend</i> . Fuente: elaboración propia. . . . .	125
32.	Diagrama de estructura del uso de LLMs en las fase 4. Fuente: elaboración propia. . . . .	128
33.	Ponderación total de los elementos generados por LLMs frente a la implementación de referencia. Fuente: elaboración propia. . . . .	153
34.	Completitud en la tarea de bases de datos obtenida por cada LLM. Fuente: elaboración propia. . . . .	153
35.	Completitud en la tarea de generación de código obtenida por cada LLM. Fuente: elaboración propia. . . . .	154
36.	Recuento de categorías en la corrección de tablas generadas por Llama 3.1 y Mistral. Fuente: elaboración propia. . . . .	154
37.	Recuento de categorías en la corrección de relaciones generadas por Llama 3.1 y Mistral. Fuente: elaboración propia. . . . .	155

38.	Recuento de categorías en la corrección de atributos generados por Llama 3.1 y Mistral. Fuente: elaboración propia. . . . .	155
39.	Distribución total de categorías de corrección de elementos de bases de datos obtenidos por cada modelo. Fuente: elaboración propia. . . . .	156
40.	Distribución total de categorías de corrección en la generación de código por cada modelo. Fuente: elaboración propia. . . . .	156
41.	Colección de solicitudes HTTP utilizadas en <i>Insomnia</i> para validar el funcionamiento de la API REST desarrollada. Fuente: elaboración propia. . . . .	159
42.	Ejemplo de solicitud HTTP correcta en <i>Insomnia</i> . Fuente: elaboración propia. .	160
43.	Ejemplo de solicitud HTTP correcta en <i>Insomnia</i> . Fuente: elaboración propia. .	160
44.	Configuración previa de PactumJS para pruebas E2E en <i>backend</i> . Fuente: elaboración propia. . . . .	162
45.	Ejemplo de solicitud HTTP de prueba E2E con PactumJS. Fuente: elaboración propia. . . . .	163
46.	Diagrama de estructura del uso de LLMs en las fase 5. Fuente: elaboración propia.	165
47.	Comparación visual de resultados en la completitud de diagramas de secuencia respecto al desarrollo manual. Fuente: elaboración propia. . . . .	171
48.	Distribución de categorías de corrección de pruebas E2E obtenidas por cada modelo. Fuente: elaboración propia. . . . .	172
49.	Comparación de la completitud obtenida por los LLMs en cada fase. Fuente: elaboración propia. . . . .	182
50.	Comparación de la corrección obtenida por los LLMs en cada fase. Fuente: elaboración propia. . . . .	183
51.	Comparación de la calificación obtenida por los LLMs en cada fase. Fuente: elaboración propia. . . . .	184
52.	Comparación de la calificación obtenida por los LLMs en las tareas de cada fase del proceso de desarrollo software. Fuente: elaboración propia. . . . .	185
53.	Registro de nuevo usuario en el sistema. Fuente: Llama 3.1 . . . . .	229
54.	Crear nuevo empleado. Fuente: Llama 3.1 . . . . .	233
55.	Ver lista de empleados. Fuente: Llama 3.1 . . . . .	238
56.	Ver información de empleado. Fuente: Llama 3.1 . . . . .	238

57.	Ver información adicional de empleado. Fuente: Llama 3.1 . . . . .	239
58.	Lista de actividades y selección de actividad. Fuente: Llama 3.1 . . . . .	244
59.	Introducir datos de reserva y pago. Fuente: Llama 3.1 . . . . .	245
60.	Realizar pago de reserva de actividad turística. Fuente: Llama 3.1 . . . . .	247
61.	Registro de nuevo usuario. Fuente: Mistral . . . . .	274
62.	Eliminar empleado existente. Fuente: Mistral . . . . .	276
63.	Realizar pago de reserva de actividad turística. Fuente: Mistral . . . . .	283
64.	Inicio de sesión en el sistema por parte de un cliente. Fuente: elaboración propia.	319
65.	Gestión de empleados: Crear nuevo empleado. Fuente: elaboración propia. . .	319
66.	Gestión de empleados: Eliminar empleado. Fuente: elaboración propia. . . . .	320
67.	Gestión de empleados: Leer información de empleado. Fuente: elaboración propia. . . . .	320
68.	Gestión de empleados: Actualizar información de empleado. Fuente: elabora- ción propia. . . . .	321
69.	Realizar reserva de actividad turística. Fuente: elaboración propia. . . . .	321
70.	Realizar pago de reserva de actividad turística. Fuente: elaboración propia. . .	322
71.	Página principal de la aplicación web. Fuente: elaboración propia. . . . .	322
72.	Maqueta de pantalla de registro de usuario. Fuente: elaboración propia. . . . .	323
73.	Maqueta de interfaz de inicio de sesión de cliente ya registrado. Fuente: elabo- ración propia. . . . .	323
74.	Maqueta de pantalla de administración: Gestionar empleados. Fuente: elabo- ración propia. . . . .	324
75.	Maqueta de gestión de empleados: Crear nuevo empleado. Fuente: elaboración propia. . . . .	324
76.	Maqueta de gestión de empleados: Eliminar empleado. Fuente: elaboración propia. . . . .	325
77.	Maqueta de gestión de empleados: Visualizar empleado. Fuente: elaboración propia. . . . .	325
78.	Maqueta de gestión de empleados: Editar empleado. Fuente: elaboración propia.	326
79.	Maqueta de pantalla de reserva de actividad turística. Fuente: elaboración propia.	326
80.	Maqueta de pantalla de pago de reserva de actividad. Fuente: elaboración propia.	327

81.	Ejemplo de formulario reactivo en <i>frontend</i> . Fuente: elaboración propia. . . . .	328
82.	Código fuente para solicitud HTTP desde <i>frontend</i> hacia <i>backend</i> . Fuente: elaboración propia. . . . .	328
83.	Código fuente para <i>login</i> y firma de <i>token</i> en <i>backend</i> . Fuente: elaboración propia.	329
84.	Implementación <i>JwtAuthGuard</i> en <i>backend</i> . Fuente: elaboración propia. . . . .	329
85.	Implementación <i>RolesGuard</i> en <i>backend</i> . Fuente: elaboración propia. . . . .	330
86.	Implementación de Roles en <i>backend</i> . Fuente: elaboración propia. . . . .	330



# Índice de cuadros

1.	Cantidad de estudios de la fase de implementación que han aplicado LLMs a estas tareas. Fuente adaptada de Xinyi Hou et al. . . . . .	37
2.	Especificación de casos de uso: Registro de nuevo usuario. Fuente: elaboración propia. . . . .	54
3.	Catálogo de requisitos funcionales coincidentes respecto al desarrollo manual. Fuente: Llama 3.1 . . . . .	66
4.	Relación entre requisitos funcionales del análisis manual y coincidencias y equivalencias con requisitos funcionales obtenidos por Llama 3.1. Fuente: Llama 3.1 . . . . .	67
5.	Catálogo de requisitos no funcionales coincidentes respecto al desarrollo manual. Fuente: Llama 3.1 . . . . .	68
6.	Lista de actores coincidentes respecto al desarrollo manual. Fuente: Llama 3.1	68
7.	Tabla de elementos de la especificación de cada caso de uso generado por Llama 3.1 y coincidencias con el modelo de referencias. Fuente: elaboración propia. .	69
8.	Coincidencia total de elementos de especificación de cada caso de uso generado por Llama 3.1. Fuente: elaboración propia. . . . .	69
9.	Actor y modelos de clase del diagrama de <b>Registro de nuevo usuario</b> presentes en el diagrama de referencia generado por Llama 3.1. Fuente: elaboración propia. . . . .	70
10.	Lista de mensajes (solicitud y respuesta) del diagrama de referencia <b>Registro de nuevo usuario</b> presentes en el diagrama generado por Llama 3.1. Fuente: elaboración propia. . . . .	70
11.	Actor y modelos de clase del diagrama <b>Gestión de empleados</b> presentes en el diagrama generado por Llama 3.1. Fuente: elaboración propia. . . . .	71
12.	Lista de mensajes (solicitud y respuesta) del diagrama de referencia <b>Gestión de empleados</b> presentes en el diagrama generado por Llama 3.1. Fuente: elaboración propia. . . . .	71

13.	Actor y modelos de clase del diagrama de <b>Reserva de actividad turística</b> presentes en el diagrama de referencia generado por Llama 3.1. Fuente: elaboración propia. . . . .	72
14.	Lista de mensajes (solicitud y respuesta) del diagrama de referencia <b>Reserva de actividad turística</b> presentes en el diagrama generado por Llama 3.1. Fuente: elaboración propia. . . . .	73
15.	Actor y modelos de clase del diagrama de <b>Pago de reserva de actividad turística</b> presentes en el diagrama de referencia generado por Llama 3.1. Fuente: elaboración propia. . . . .	73
16.	Lista de mensajes (solicitud y respuesta) del diagrama de referencia <b>Pago de reserva de actividad turística</b> presentes en el diagrama generado por Llama 3.1. Fuente: elaboración propia. . . . .	74
17.	Elementos del formulario de registro de nuevo de usuario generados por Llama 3.1. Fuente: Llama 3.1. . . . .	74
18.	Elementos de pantalla del formulario de registro de nuevo empleado. Fuente: Llama 3.1. . . . .	75
19.	Elementos de pantalla eliminación de empleado. Fuente: Llama 3.1. . . . .	75
20.	Elementos de pantalla de visualización de empleado. Fuente: Llama 3.1. . . . .	75
21.	Elementos de la pantalla de edición de empleado. Fuente: Llama 3.1. . . . .	76
22.	Elementos de la pantalla de actividades. Fuente: Llama 3.1. . . . .	76
23.	Elementos de la pantalla de detalles de la actividad. Fuente: Llama 3.1. . . . .	76
24.	Elementos de la pantalla de reserva de actividad turística. Fuente: Llama 3.1. . . . .	77
25.	Elementos de la pantalla de pago de reserva de actividad. Fuente: Llama 3.1. . . . .	77
26.	Catálogo de requisitos funcionales coincidentes respecto al desarrollo manual. Fuente: Mistral . . . . .	78
27.	Relación entre requisitos funcionales del análisis manual con coincidencias y equivalencias con requisitos funcionales obtenidos por Mistral. Fuente: Mistral . . . . .	79
28.	Lista de actores coincidentes respecto al desarrollo manual. Fuente: Mistral . . . . .	80
29.	Elementos de la especificación de cada caso de uso generado por Mistral y coincidencias con el modelo de referencias. Fuente: elaboración propia. . . . .	80

30.	Coincidencia total de elementos de especificación de cada caso de uso generado por Mistral. Fuente: elaboración propia. . . . .	81
31.	Actor y modelos de clase del diagrama <b>Registro de nuevo usuario</b> presentes en el diagrama generado por Mistral. Fuente: elaboración propia. . . . .	81
32.	Lista de mensajes (solicitud y respuesta) del diagrama de referencia <b>Registro de nuevo usuario</b> presentes en el diagrama generado por Mistral. Fuente: elaboración propia. . . . .	82
33.	Actor y modelos de clase presentes en el diagrama <b>Gestión de empleados</b> generado por Mistral. Fuente: elaboración propia. . . . .	82
34.	Lista de mensajes (solicitud y respuesta) presentes en el diagrama <b>Gestión de empleados</b> generado por Mistral. Fuente: elaboración propia. . . . .	83
35.	Actor y modelos de clase presentes en el diagrama de <b>Reserva de actividad turística</b> generado por Mistral. Fuente: elaboración propia. . . . .	84
36.	Lista de mensajes (solicitud y respuesta) presentes en el diagrama de <b>Reserva de actividad turística</b> generado por Mistral. Fuente: elaboración propia. . . . .	84
37.	Actor y modelos de clase presentes en el diagrama <b>Pago de reserva de actividad turística</b> generado por Mistral. Fuente: elaboración propia. . . . .	85
38.	Lista de mensajes (solicitud y respuesta) presentes en el diagrama <b>Pago de reserva de actividad turística</b> generado por Mistral. Fuente: elaboración propia. . . . .	85
39.	Elementos del formulario de registro de nuevo de usuario generados por Mistral. Fuente: Mistral. . . . .	86
40.	Elementos de pantalla del formulario de registro de nuevo empleado. Fuente: Mistral . . . . .	86
41.	Elementos de pantalla eliminación de empleado. Fuente: Mistral. . . . .	87
42.	Elementos de la pantalla de edición de empleado. Fuente: Mistral. . . . .	87
43.	Elementos de la pantalla de visualización de empleado. Fuente: Mistral. . . . .	88
44.	Elementos del panel de actividades. Fuente: Mistral. . . . .	88
45.	Elementos del detalle de la actividad. Fuente: Mistral. . . . .	89
46.	Elementos de la pantalla de reserva de actividad. Fuente: Mistral. . . . .	89
47.	Elementos de la pantalla de pago de reserva. Fuente: Mistral. . . . .	90

48.	Comparación de de elementos coincidentes en la especificación de casos de uso entre Desarrollo Manual, Llama 3.1 y Mistral. Fuente: elaboración propia.	92
49.	Resumen de procesos, actores, clases de análisis y acciones/mensajes de los diagramas de referencia. Fuente: elaboración propia. . . . .	93
50.	Recuento total de elementos por formato de pantalla del desarrollo manual. Fuente: elaboración propia. . . . .	94
52.	Métrica de corrección de requisitos generados por los LLMs respecto al desarrollo manual. Fuente: elaboración propia. . . . .	98
53.	Métrica de corrección de actores y sus casos de uso generados por los LLMs respecto al desarrollo manual. Fuente: elaboración propia. . . . .	100
54.	Corrección de la especificación del caso de uso <b>Registro de nuevo usuario</b> respecto al modelo de referencia. Fuente: elaboración propia. . . . .	101
55.	Corrección de la especificación del caso de uso <b>Gestionar empleados</b> respecto al modelo de referencia. Fuente: elaboración propia. . . . .	101
56.	Corrección de la especificación del caso de uso <b>Realizar reserva de actividad turística</b> respecto al modelo de referencia. Fuente: elaboración propia. . . . .	102
57.	Corrección de la especificación del caso de uso <b>Pago de reserva de actividad turística</b> respecto al modelo de referencia. Fuente: elaboración propia. . . . .	102
58.	Corrección diagrama <b>Registro de nuevo usuario</b> , Llama 3.1. . . . .	103
59.	Corrección diagrama <b>Registro de nuevo usuario</b> , Mistral . . . . .	103
60.	Resumen y recuento total de la corrección de diagramas de secuencia generados por Llama 3.1. Fuente: elaboración propia. . . . .	104
61.	Resumen y recuento total de la corrección de diagramas de secuencia generados por Mistral. Fuente: elaboración propia. . . . .	104
62.	Corrección formato de pantalla de <b>Registro de nuevo usuario</b> , Llama 3.1. . . . .	105
63.	Corrección formato de pantalla de <b>Registro de nuevo usuario</b> , Mistral . . . . .	105
64.	Resumen de <i>endpoints</i> implementados en el <i>backend</i> . Fuente: elaboración propia.	122
68.	Resumen de código generado por Llama 3.1 para el registro de nuevos usuarios (clientes) y login de usuarios existentes en el sistema. Fuente: Llama 3.1.. . . .	133
69.	Resumen de código generado por Llama 3.1 para la gestión de empleados como responsable. Fuente: Llama 3.1. . . . .	134

73.	Resumen de código generado por Mistral para el registro de nuevos usuarios (clientes) y login de usuarios existentes en el sistema. Fuente: Mistral. . . . .	138
74.	Resumen de código generado por Mistral para la gestión de empleados como responsable. Fuente: Mistral. . . . .	139
75.	Resumen de elementos coincidentes en el código generado por Llama 3.1 para cada funcionalidad respecto al total de elementos de referencia. Fuente: elaboración propia. . . . .	142
76.	Resumen de elementos coincidentes en el código generado por Mistral para cada funcionalidad respecto al total de elementos de referencia. Fuente: elaboración propia. . . . .	142
77.	Elementos coincidentes de registro y login generado por los LLMs respecto al desarrollo de referencia ( <i>backend</i> ). Fuente: elaboración propia. . . . .	143
78.	Elementos coincidentes de registro y login generado por los LLMs respecto al desarrollo de referencia ( <i>frontend</i> ). Fuente: elaboración propia. . . . .	143
79.	Elementos coincidentes de la gestión de empleados generado por los LLMs respecto al desarrollo de referencia ( <i>backend</i> ). Fuente: elaboración propia. . . .	144
80.	Elementos coincidentes de la gestión de empleados generado por los LLMs respecto al desarrollo de referencia ( <i>frontend</i> ). Fuente: elaboración propia. . .	144
81.	Elementos coincidentes de reservas y pagos generado por los LLMs respecto al desarrollo de referencia ( <i>backend</i> ). Fuente: elaboración propia. . . . .	145
82.	Rúbrica para la evaluación de corrección de elementos en la implementación de la base de datos. Fuente: elaboración propia. . . . .	147
83.	Corrección de las tablas generadas por Llama 3.1. Fuente: elaboración propia. .	148
84.	Corrección de las relaciones generadas por Llama 3.1. Fuente: elaboración propia.	148
85.	Corrección de las tablas generadas por Mistral. Fuente: elaboración propia. . .	149
86.	Corrección de las relaciones generadas por Mistral. Solo relaciones directas. Fuente: elaboración propia. . . . .	149
87.	Corrección de las atributos generados por Llama 3.1 y Mistral. Fuente: elaboración propia. . . . .	150
88.	Número de elementos generados por cada modelo para el diseño e implementación de la base de datos. Fuente: elaboración propia. . . . .	151

89.	Recuento total de la distribución de categorías de corrección de Llama 3.1 y Mistral en la tarea de implementación software. Fuente: elaboración propia. . .	151
90.	Resumen de endpoints, estados esperados y requisitos realizados en el desarrollo manual para pruebas E2E. Fuente: elaboración propia. . . . .	164
91.	Resumen de elementos de la generación de pruebas E2E de <i>endpoints</i> por Llama 3.1. Fuente. Llama 3.1. . . . .	167
92.	Resumen de elementos de la generación de pruebas E2E de <i>endpoints</i> por Mistral. Fuente. Mistral. . . . .	167
93.	Criterios de corrección para los LLMs en la fase de pruebas. Fuente: elaboración propia. . . . .	169
94.	Resumen de la corrección de las pruebas E2E generadas por Llama 3.1. Fuente: elaboración propia. . . . .	170
95.	Resumen de la corrección de las pruebas E2E generadas por Mistral. Fuente: elaboración propia. . . . .	170
96.	Tabla de <b>ponderación</b> de categorías de <b>corrección</b> para el cálculo de la calificación por tareas en cada fase para cada LLM. Fuente: elaboración propia. . .	178
97.	Valores de las tareas por cada LLM en la fase 1. Fuente: elaboración propia. . .	179
98.	Valores de las tareas por cada LLM en la fase 2 y 3. Fuente: elaboración propia.	180
99.	Valores de las tareas por cada LLM en la fase 4 y 5. Fuente: elaboración propia.	180
100.	Notas por tarea obtenidas por Llama 3.1 y Mistral. Fuente: elaboración propia.	180
101.	Calificación por fase para Llama 3.1 y Mistral. Fuente: elaboración propia. . . .	181
102.	Calificación global de cada LLM. Fuente: elaboración propia. . . . .	181
103.	Media de la métrica de completitud obtenida por cada LLM en las distintas fases. Fuente: elaboración propia. . . . .	182
104.	Media de la métrica de corrección obtenida por cada LLM en las distintas fases. Fuente: elaboración propia. . . . .	183
117.	Catálogo de requisitos funcionales. Fuente: elaboración propia. . . . .	304
118.	Catálogo de requisitos no funcionales. Fuente: elaboración propia. . . . .	308
119.	Catálogo de requisitos de información. Fuente: elaboración propia. . . . .	309
120.	Especificación de casos de uso: Gestionar empleados. Fuente: elaboración propia.	312
121.	Especificación de casos de uso: Inicio de sesión. Fuente: elaboración propia. . .	313

122. Especificación de casos de uso: Reserva de una actividad turística. Fuente: elaboración propia. . . . .	314
123. Especificación de casos de uso: Pago de reserva de una actividad turística. Fuente: elaboración propia. . . . .	316



# 1

# Introducción

## 1.1. Motivación

En los últimos años, la inteligencia artificial generativa ha tomado un papel protagonista en el sector tecnológico-científico. La aparición de LLMs<sup>1</sup> comerciales como los de la familia GPT<sup>2</sup> [1] ha generado un amplio interés en la comunidad tecnológica debido a su capacidad para generar respuestas de alta calidad en poco tiempo, contribuyendo a la mejora de la productividad y obteniendo resultados generalmente acertados. Sin embargo, estos modelos suelen ser de carácter propietario y cerrados por lo que esta situación ha motivado la aparición de nuevos modelos de inteligencia artificial gratuitos que tienen como objetivo ofrecer capacidades y un rendimiento similar a grandes modelos de inteligencia artificial como GPT-4.

Ante la aparición de LLMs gratuitos suficientemente preentrenados y cada vez más robustos, este Trabajo Fin de Grado tiene como motivación seleccionar y estudiar modelos de inteligencia artificial generativa en distintas fases del proceso de desarrollo software a través de un caso práctico que sirva para evaluar su rendimiento y además, realizar un análisis comparativo entre el desarrollo que haría un humano y el resultado obtenido del desarrollo asistido por LLMs.

## 1.2. Objetivos

El objetivo principal del proyecto consiste en evaluar el rendimiento de distintos LLMs en ciertas fases del proceso de desarrollo software. Para ello, se desarrollará un caso práctico de una aplicación web para una empresa ficticia del sector turístico. Este desarrollo será realizado por dos actores principales, humano y LLMs.

---

<sup>1</sup>Large Language Models

<sup>2</sup>Generative Pre-trained Transformer

La dinámica que se seguirá durante el proyecto consiste en que por cada fase validada del desarrollo software, se pida mediante instrucciones textuales formuladas a los distintos LLMs que realicen el análisis o desarrollo que se les indique. Una vez se obtengan los dos resultados, es decir, el desarrollado por el humano y el obtenido por la inteligencia artificial, se realizará un análisis comparativo y evaluación de los modelos.

Este proyecto no tiene como objetivo entrenar ningún modelo de los seleccionados, sino de evaluar las capacidades de estos modelos ya entrenados.

En conclusión, los objetivos principales son el desarrollo de una aplicación web que servirá como base de referencia para la evaluación de modelos inteligencia artificial generativa gratuitos en fases típicas del desarrollo de aplicaciones software, que también se utilizará para comparar el desempeño de los modelos respecto al trabajo obtenido por un humano.

### 1.3. Metodología y tecnologías

En esta sección se recogen los apartados de metodología de trabajo que se ha seguido durante el desarrollo del proyecto y también las tecnologías y herramientas empleadas para la realización del mismo.

#### 1.3.1. Metodología Scrum

Este proyecto ha requerido el desarrollo de un caso práctico de forma manual. Este desarrollo se ha utilizado como base en el análisis comparativo por lo que ha sido necesario que se valide cada fase del proyecto que se complete. Por ello, para el proceso de desarrollo de software por parte del autor se ha optado por seguir una metodología con enfoque ágil, en concreto, Scrum [2].

En Scrum, se trabaja en iteraciones, conocidas como *sprints* [3]. Para este proyecto, normalmente las iteraciones consistieron de una duración estimada entre 10 y 14 días tras los cuales se realizaron reuniones con el tutor en los que se mostró el trabajo desarrollado y, de esta forma, manteníamos un seguimiento y desarrollo continuo. Las iteraciones reflejaron dos partes. La primera parte consistió en el desarrollo manual de cada fase del proceso de desarrollo software por parte del autor, que, tras su validación con el tutor del TFG, se dio paso a realizar la segunda parte del *sprint*, basada en la obtención del desarrollo del proyecto por

parte de los modelos de inteligencia artificial. Después de cada *sprint* se realizó una evaluación para llevar a cabo un estudio con el fin de obtener un análisis comparativo en función de las métricas propuestas.

### 1.3.2. Tecnologías y herramientas utilizadas

Para el desarrollo del proyecto se han empleado diversas herramientas. **Hugging Face** [4] proporcionó los modelos de lenguaje de código abierto utilizados mediante su librería *transformers* [5]. **Visual Paradigm** [6] fue utilizado para la elaboración de modelos y diagramas del sistema. **GitHub** [7] y **Git** [8] se emplearon para el control de versiones y almacenamiento en la nube. La gestión ágil del proyecto, basada en Scrum, se realizó mediante **Taiga** [9], que permitió organizar tareas por *sprints*, registrar tiempos dedicados y facilitar la comunicación con el tutor académico. Para el diseño de interfaces se utilizó **MyBalsamiq** [10], y **Insomnia** [11] sirvió para la verificación y prueba de las APIs desarrolladas.

En cuanto a las tecnologías empleadas en el desarrollo del caso práctico, **Angular** [12] fue la herramienta seleccionada para construir el *frontend*, siguiendo una arquitectura basada en componentes y el patrón *Model-View-Controller* [13]. Para el *backend*, se optó por **NestJS** [14], por su arquitectura modular en TypeScript y su integración con Node.js. En la gestión de datos se utilizó **MariaDB** [15], administrada con **HeidiSQL** [16], ambos de código abierto. Todo el desarrollo fue llevado a cabo desde el entorno de desarrollo **Visual Studio Code** [17], elegido por su versatilidad, compatibilidad con múltiples tecnologías y herramientas integradas de depuración.

## 1.4. Estructura del documento

La **estructura** que compone la memoria de este **proyecto**<sup>3</sup> es la que se muestra a continuación.

- **Capítulo 1: Introducción**

Este apartado del documento contiene la estructura del mismo, objetivos y motivación del proyecto. Asimismo, recoge la metodología usada para el proyecto y las tecnologías y herramientas empleadas.

---

<sup>3</sup>Desde este punto se usará el término proyecto para referirse al Trabajo Fin de Grado

■ **Capítulo 2: Estado del arte**

Este capítulo presenta un análisis detallado de distintas fases del proceso de desarrollo de software, con el objetivo de identificar tareas clave en las que evaluar la viabilidad del uso de LLMs de código abierto y gratuitos. El análisis se basa en el enfoque y uso que un desarrollador o ingeniero de software podría hacer de la inteligencia artificial generativa en las tareas típicas de fases del desarrollo software. Asimismo, se han estudiado trabajos previos e investigaciones acerca del uso de LLMs para realizar distintas tareas del proceso de desarrollo software. Por último, se han seleccionado y descrito ciertos LLMs para la evaluación de este proyecto y así como métricas que se han aplicado.

■ **Capítulo 3, 4 y 5: Evaluación del uso de LLMs**

La evaluación del uso de LLMs recoge las tareas principales del análisis, diseño y desarrollo realizado a lo largo de este TFG por lo que se divide en distintos capítulos, del 3 al 5. Muestra el desarrollo manual que realizaría una persona con conocimientos en desarrollo e ingeniería del software frente a un caso práctico de desarrollo. Dicho desarrollo manual se ha utilizado como base para evaluar a los LLMs seleccionados en el capítulo 2. Por ello, este capítulo recopila el desarrollo manual realizado, desarrollo obtenido y generado por los distintos modelos en cada tarea de las distintas fases y por último la evaluación tras aplicar las métricas pertinentes. En esta sección de la memoria, también se introduce las funcionalidades del caso práctico utilizado para el Trabajo Fin de Grado.

■ **Capítulo 6: Evaluación final**

En este capítulo se recoge principalmente dos apartados. En primer lugar, se explican los resultados obtenidos mediante el uso de los LLMs en las distintas fases del proceso de desarrollos software. En segundo lugar, se finaliza la evaluación a través de aplicar una serie de fórmulas que posteriormente permiten obtener una calificación del uso de LLMs en varios aspectos; por tareas, fases y uso global. Asimismo, en esta segunda parte se incluye unos diagramas que muestran los resultados obtenidos por los modelos en las métricas principales del proyecto por cada fase.

■ **Capítulo 7: Conclusión y líneas futuras**

Este capítulo muestra una conclusión general del trabajo fin de grado tras la realización del mismo. Asimismo, se comentan las líneas futuras que podría seguir este proyecto.

### ■ Apéndices

Esta sección se encarga de recoger y mostrar información complementaria al documento. Asimismo, recopila todos los *prompts* utilizados en el proyecto y las salidas obtenidas por cada uno.

Dada la extensión de la memoria y la gran cantidad de información que contiene, se ha incluido un diagrama que representa la estructura del documento y su contenido. Para facilitar la lectura, se han realizado dos diagramas con estructura de árbol. Dicha estructura se compone de un nodo principal, *Trabajo Fin de Grado*, el cual ramifica en nodos hijos, cada uno haciendo referencia a una sección del documento. Cada sección se interpreta como un capítulo y, dentro de cada uno, se encuentran apartados que contienen sus apartados.

Se presentan dos diagramas, el principal mostrado en la figura 2 y, el segundo en la figura 1, el cual contiene el contenido adicional al cuerpo principal de la memoria.

En primer lugar, se muestra el **diagrama con el contenido adicional** de la memoria.

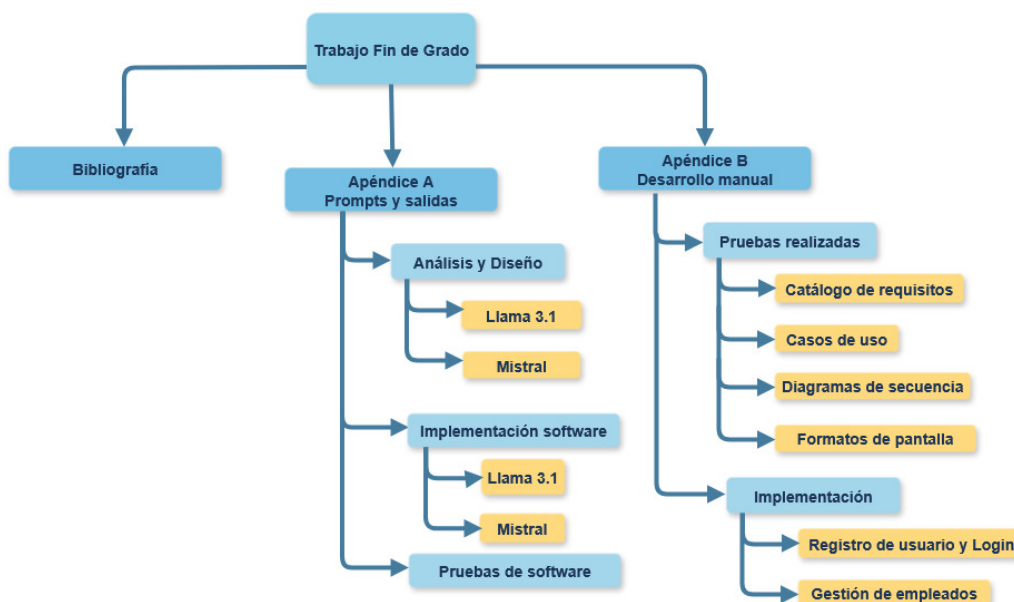


Figura 1: Diagrama del contenido adicional del documento. Fuente: elaboración propia.

Finalmente, en la siguiente página se muestra el **diagrama del cuerpo principal** de este documento.

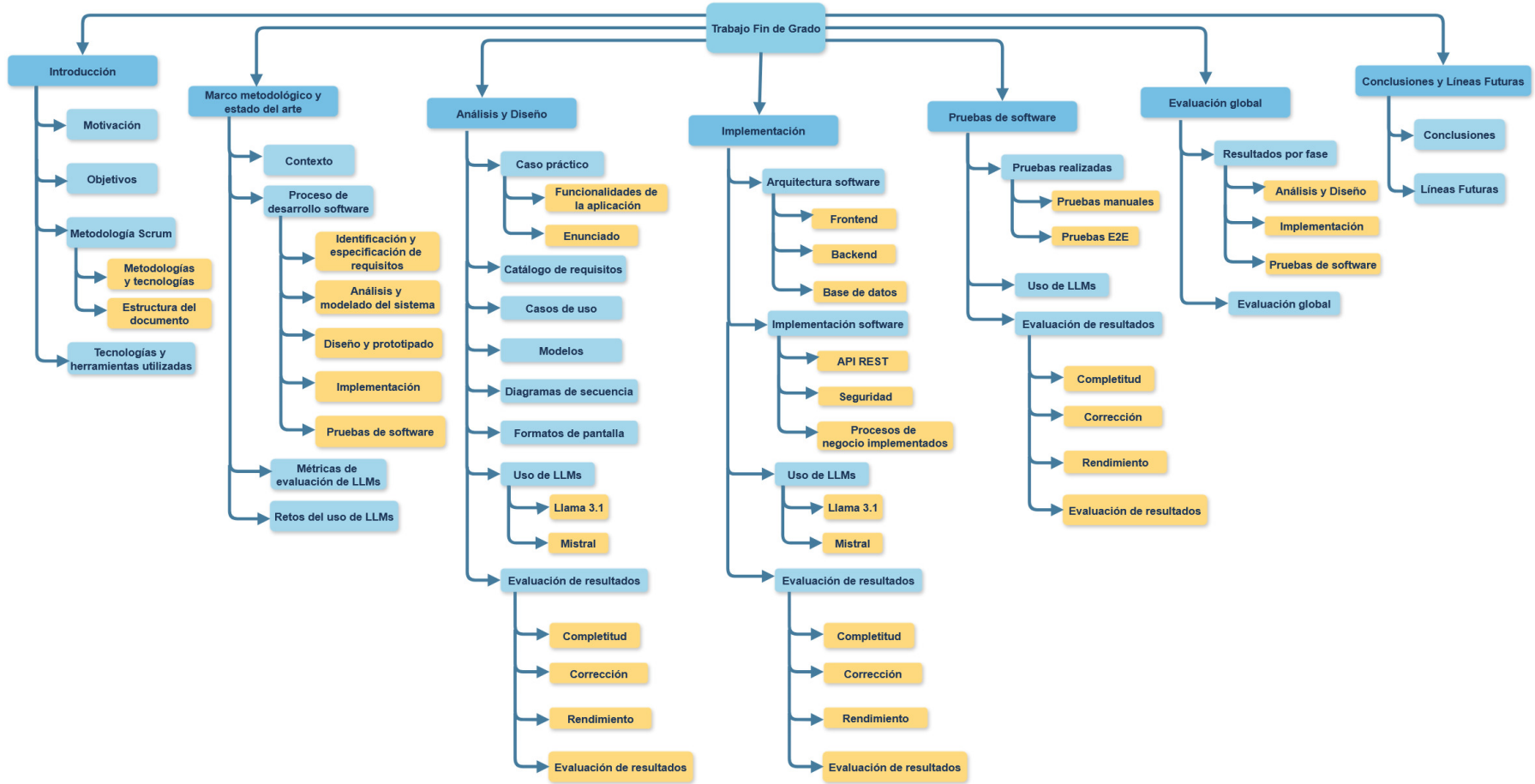


Figura 2: Diagrama de la estructura principal del documento. Fuente: elaboración propia.

# 2

## Marco metodológico y estado del arte

Este capítulo recoge una revisión estructurada de las distintas fases del proceso de desarrollo software y sus principales tareas. Asimismo, se ha realizado un estudio del uso de LLMs por cada fase, analizando su viabilidad a través del análisis de estudios e investigaciones las cuales recopilan resultados del uso de LLMs. Además, se presentan las distintas métricas utilizadas para la evaluación de los LLMs en las diferentes tareas de cada fase y, por último, se estudia las limitaciones del uso de LLMs en el proceso de desarrollo software.

### 2.1. Contexto

En la actualidad, los modelos de inteligencia artificial generativa, mejor conocidos como *Large Language Models* (LLMs) han empezado a ser utilizados con mayor frecuencia en fases típicas de la ingeniería del software debido a su potencial para mejorar la productividad y por su capacidad para generar respuestas rápidas de alta calidad. Estos modelos ayudan en tareas de diferentes fases, como identificar requisitos durante la fase de análisis, desarrollar diseños en la fase de modelado, generar código y solucionar errores en la fase de implementación, y crear casos de prueba en la fase de pruebas correspondiente [18]. La capacidad única de la inteligencia artificial generativa ha ofrecido nuevas formas de optimizar los procesos del desarrollo software. Sin embargo, la integración en los procesos actuales también presenta desafíos que exigen un análisis técnico para aprovechar de manera efectiva el potencial de la inteligencia artificial generativa [19].

Los modelos de inteligencia artificial generativa están transformando áreas de la propia ingeniería del software. Estos modelos de IAG<sup>4</sup> han comenzado a construir nuevos flujos de

---

<sup>4</sup>Inteligencia Artificial Generativa

trabajo y redefinir otros, cambiando roles en la ingeniería del software y abriendo una frontera entre la capacidad humana y la aptitud para automatizar por parte de los modelos.

En el ecosistema actual de los modelos de inteligencia artificial generativa se distinguen dos categorías de modelos: comerciales, desarrollados por compañías como OpenAI que restringen su uso y distribución, y modelos open source, como los publicados en la comunidad de Hugging Face, los cuales están disponibles de manera pública y son usados de forma gratuita bajo ciertas licencias. La elección entre modelos comerciales y gratuitos radica en decisiones estratégicas y técnicas. Los modelos comerciales se distinguen por su facilidad de integración y uso inmediato, ya que admiten procesos de adaptación ligeros, incluso con volúmenes reducidos de datos lo que implica un coste computacional bajo para el usuario. No obstante, su carácter propietario restringe el acceso al modelo subyacente y limita la posibilidad a la personalización. Por otro lado, los modelos open source, es decir, gratuitos, ofrecen una mayor flexibilidad, ya que permiten tanto el acceso al código fuente como la posibilidad de ajustar el entrenamiento del modelo para obtener respuestas más precisas dentro de un ámbito. Sin embargo, este nivel de personalización implica una inversión considerable en términos de recursos computacionales, tiempo y conocimientos técnicos avanzados.

Modelos comerciales como ChatGPT o GitHub Copilot han demostrado ser especialmente eficaces en tareas como la generación de código, la documentación automática o el análisis de errores, lo que ha motivado una gran cantidad de investigaciones centradas en su evaluación y aplicación práctica. Sin embargo, una revisión exhaustiva de la literatura revela una tendencia recurrente: muchos estudios existentes se limitan a analizar modelos comerciales o propietarios, dejando en segundo plano el potencial de los LLMs gratuitos y de código abierto. Esta falta de atención resulta relevante si se considera que el uso de modelos accesibles puede ser clave para democratizar el acceso a esta tecnología, especialmente en determinados contextos. Asimismo, si bien existen numerosos trabajos que recopilan y clasifican aplicaciones de LLMs en la ingeniería del software, pocos realizan una evaluación sistemática de modelos específicos en cada una de las fases del proceso de desarrollo software. Esta laguna en la literatura justifica la necesidad de realizar una evaluación centrada en analizar cómo el uso de LLMs gratuitos puede ser empleado eficazmente en dichas fases. En este contexto, en este trabajo se plantea una evaluación que busca cubrir ese vacío, mediante la evaluación práctica de varios LLMs accesibles y no comerciales, aplicados a tareas concretas mediante el desarrollo de un

caso práctico.

## **2.2. Proceso de desarrollo software**

El proceso de desarrollo software se entiende como un proceso sistemático y estructurado para crear aplicaciones informáticas que implica varias fases. Las fases típicas del proceso de desarrollo software incluyen etapas como planificación del proyecto, análisis y especificación, modelado del sistema, diseño y prototipado del software, implementación del software, pruebas software, despliegue, operaciones y mantenimiento. Debido al alcance de este proyecto y la capacidad actual de los modelos de IAG se ha optado por seleccionar fases específicas para este TFG, pertinentes a etapas que se pueden encontrar en cualquier proyecto de desarrollo software, independientemente de la complejidad del proyecto. De este modo, se ha evaluado a los modelos de IAG en distintas fases del proceso de desarrollo software y en tareas comunes dentro de cada una, representativas de aquellas que un desarrollador o ingeniero de software suele abordar habitualmente.

Como ya se ha indicado, el proceso de desarrollo software abarca fases desde la planificación hasta el mantenimiento de lo desarrollado. A pesar de esto, debido a las limitaciones de tiempo propias de un TFG, fases como planificación de recursos, objetivos, gestión del proyecto de desarrollo y mantenimiento del software han sido excluidas del alcance y objetivos del estudio.

En este capítulo se muestran qué LLMs pueden ayudar a realizar dichas tareas o mejorar sus procesos, seleccionando de esta manera a los LLMs evaluados en el desarrollo del caso práctico.

Las principales fases seleccionadas para la evaluación del uso de LLMs en el proceso de desarrollo software son la identificación y especificación de requisitos, análisis y modelado del sistema, diseño y prototipado, implementación y pruebas de software.

### **2.2.1. Identificación y especificación de requisitos**

La primera fase de todo proyecto de desarrollo software consiste en identificar y especificar requisitos. Esta fase es importante puesto que permite definir la base del proyecto a través de la recogida y documentación de necesidades del cliente o del usuario final, facilitando el

posterior desarrollo. Esta etapa responde a la pregunta de quién usará el sistema y qué hará, recogiendo y detallando las necesidades del software a implementar, a través de tareas típicas de esta fase entre las que se encuentran elaborar un catálogo de requisitos y detallar los casos de uso del sistema con los principales actores. En esta fase se define una estrategia de análisis y se recopilan los requisitos necesarios para diseñar el nuevo sistema [20].

Un requisito se define como una declaración de qué debe hacer el sistema. Se identifican distintos tipos de requisitos, funcionales, no funcionales y de información. Los requisitos funcionales recogen las funcionalidades que un sistema debe de implementar y ser capaz de ejecutar, mientras que los no funcionales, recogen cómo debe ser el comportamiento de un sistema, ya sea una característica, condición o capacidad. [21]. Finalmente los requisitos de información especifican la información que gestiona y almacena un sistema.

En cuanto a los casos de uso, son una manera formal de representar requisitos funcionales, en concreto, cómo se relaciona el sistema con su entorno. Sirven para documentar el sistema y muestra los distintos usuarios que interactúan con el mismo. Los elementos que incluyen los diagramas de casos de uso son actores, casos de uso y relaciones entre estos. [22]. Como parte de los casos de uso, se elaboran las especificaciones de casos de uso con el objetivo de describir de forma clara y estructurada las interacciones entre los usuarios y el sistema. Estas especificaciones permiten identificar los distintos escenarios de uso principales y escenarios alternativos, definir el comportamiento esperado del sistema ante cada acción del usuario y establecer las clases de estructura y comportamiento asociadas.

Como en cualquier fase de desarrollo de proceso software, se lleva a cabo una serie de tareas típicas y fundamentales. Esta fase de análisis dispone de tareas que pueden ser asistidas por distintos LLMs gratuitos y de código abierto. Entre estas tareas se distinguen el análisis de necesidades del cliente y usuario final a través de diferentes reuniones con el cliente para dejar claro qué necesita el sistema mediante la recopilación de requisitos que posteriormente son validados para asegurar la viabilidad técnica y evitar malentendidos. Para estas tareas, se hará uso del enunciado del caso práctico.

Según el artículo *A Survey on Large Language Models for Software Engineering* [23], el cual recopila, organiza y analiza estudios previos del uso de LLMs en la ingeniería del software entre los cuales se aborda un total de 112 tareas distintas de diferentes fases del proceso de desarrollo software, modelos como Llama 3.1 y Mistral aparecen evaluados, lo que destaca

su uso en tareas de identificación y especificación de requisitos, gracias a la capacidad de cada modelo de comprensión semántica y de traducir lenguaje natural a representaciones técnicas, facilitando tareas como la generación de especificaciones de requisitos a partir de lenguaje natural.

Finalmente, para la evaluación de LLMs en tareas típicas de esta fase se ha decidido optar por dos modelos, principalmente Llama 3.1 y Mistral debido a su rendimiento para abordar tareas de generación y comprensión de lenguaje natural. Si bien otros modelos como BERT o T5 han presentado buenos resultados en tareas de este tipo, Llama 3.1 y Mistral destacan por su arquitectura generativa ya que también pueden crear especificaciones completas, redactar descripciones o proponer requisitos alternativos, lo que los convierte en opciones especialmente adecuadas para ser seleccionadas para apoyar el análisis y especificación de requisitos.

### 2.2.2. Análisis y modelado del sistema

La fase de análisis y modelado del sistema es una etapa clave en el proceso de desarrollo software tras la recopilación de requisitos. Esta etapa se puede interpretar como una capa intermedia entre el análisis e identificación de requisitos y el diseño y prototipado de un sistema. En esta etapa se estudia el comportamiento que debe tener el sistema, estructurando los requisitos validados en la primera fase .

Durante esta fase, se estudian inconsistencias o fallos y requisitos que pueden no haber sido identificados durante la supervisión del catálogo de requisitos obtenido en la primera etapa, lo cual ayuda a prevenir errores en etapas posteriores. Asimismo, la etapa de modelado del sistema proporciona una visión común, evitando interpretaciones ambiguas.

Finalmente, en cuanto a las tareas típicas de esta fase se encuentran el modelado de estructura a través de diagramas de modelos de entidad, control e interfaz. A su vez, se define el comportamiento del sistema mediante el diseño lógico a través de diagramas de secuencia. Los diagramas de secuencia muestran cómo interactúan los actores del sistema y funcionalidades entre sí a lo largo del tiempo, facilitando la comprensión del flujo de mensajes y operaciones. El modelado de clases representa las clases, atributos, métodos y relaciones del sistema, con un diseño orientado a objetos. [22]

Como se indica en el artículo revisado en la primera fase del proceso de desarrollo software, *A Survey on Large Language Models for Software Engineering* [23], se sugiere la aplicabilidad de

LLMs en análisis y modelado de sistemas software. Por ello, estudios como *Model Generation with LLMs: From Requirements to UML Sequence Diagrams* [24] muestra cómo los LLMs pueden generar diagramas de secuencia UML directamente desde descripciones en lenguaje natural, automatizando así la transición desde requisitos hasta modelos que estructuren y definan el comportamiento del sistema. Asimismo, se ha estudiado el artículo *Towards using Few-Shot Prompt Learning for Automating Model Completion* [25] el cual propone un enfoque basado en *few-shot prompt learning* con modelos como GPT-3 para completar automáticamente modelos en desarrollo, es decir, completar de forma automática diagramas que han sido iniciados pero no están finalizados, incluyendo tanto diagramas de clases como diagramas de modelo físico de datos, permitiendo generar nuevas clases, atributos y asociaciones a partir de un modelo sin finalizar mediante una transformación semántica entre los elementos de modelado y las secuencias de texto que entienden los LLMs.

Para este proyecto, se ha decidido escoger y evaluar en esta fase el uso de los LLMs gratuitos comentados anteriormente, Llama y Mistral, por su capacidad para comprender y generar lenguaje natural de forma estructurada, son capaces de producir salidas técnicas como especificaciones estructuradas, similares a diagramas de secuencia, diagramas de clases o modelos físicos de datos.

### 2.2.3. Diseño y prototipado

La fase de diseño y prototipado en el proceso de desarrollo software es fundamental porque da lugar al paso entre el análisis realizado y la construcción a nivel técnico del sistema. En esta fase se representa la arquitectura software en la que se incluye el modelo físico de datos y se modelan prototipos de las interfaces de usuario, lo cual permite establecer cómo se organizará internamente el sistema, definiendo la estructura de componentes, la lógica de funcionamiento y las bases para una implementación coherente. Estos prototipos de modelado de interfaces de usuario permiten obtener una mejor visión del proyecto a través de la interacción con el diseño visual.

Principalmente, el objetivo de esta fase consiste en diseñar las interfaces de usuario y la experiencia de usuario necesaria, asegurando que se haya adaptado a las necesidades del cliente o usuario final y elaborar el modelo físico de datos. Este modelo describe cómo se almacenará la información en una base de datos real, con tablas, claves y tipos de datos definidos.

La estructura del proyecto está basada en el patrón de arquitectura software Modelo-Vista-Controlador [13]. Esta estructura ha servido como patrón para el código en tres componentes principales:

- **Modelo:** gestiona los datos de la aplicación, la validación y las interacciones con la base de datos. Notifica a otros componentes cuando los datos cambian.
- **Vista:** se encarga de la interfaz de usuario, mostrando datos, manejando la interacción del usuario, y actualizándose según los cambios en los datos.
- **Controlador:** actúa como intermediario entre el modelo y la vista, gestionando las entradas del usuario, estableciendo la lógica de negocio.

Entre las tareas típicas de esta fase se encuentran el diseño de la arquitectura del software, en este caso utilizando el patrón Modelo-Vista-Controlador. También se encuentran el diseño visual de interfaces a través de bocetos y maquetas en el que se aplican colores, fuentes y estilos visuales. Estos diseños no son funcionales, pero sí que sirven para simular su comportamiento.

En esta fase, el uso de LLMs está emergiendo como una herramienta de apoyo valiosa. Según ha explorado el estudio *Large Language Models for Software Engineering: A Systematic Literature Review* [26] el cual recopila más de 300 estudios de investigación sobre el uso de LLMs en tareas típicas del proceso de desarrollo software, muestra estudios en los cuales se ha utilizado modelos de inteligencia artificial generativa como BERT para la reutilización de interfaces de usuario a partir de búsquedas de preguntas o descripciones en lenguaje natural, permitiendo encontrar y reutilizar componentes visuales. De esta forma, los LLMs pueden ayudar a encontrar diseños útiles a partir de descripciones textuales, optimizando tiempos a los desarrolladores e ingenieros de software, sin necesidad de hacer desde cero una interfaz funcional. Por otro lado, este mismo estudio muestra cómo se utilizan los LLMs para el prototipado. Según el estudio, los LLMs permiten generar y adaptar prototipos de forma iterativa, lo que facilita la validación temprana de ideas de diseño. Además, el alcance de los LLMs se extiende al diseño de la arquitectura de software, asistiendo en decisiones de estructuración del sistema y qué módulos lo compondrán. Asimismo, los LLMs pueden sugerir patrones de diseño adecuados para cada situación, aplicando soluciones reutilizadas a problemas comunes.

Finalmente, para el diseño de interfaces de usuario, modelo físico de datos y prototipado del sistema, se ha hecho uso de los modelos mencionados hasta el momento, es decir, Llama 3.1 y Mistral.

### 2.2.4. Implementación

La fase de implementación es una de las más cruciales en el proceso de desarrollo software porque es cuando el análisis y diseño se convierte en un sistema funcional, es decir, todo lo que se ha analizado, especificado, modelado y estructurado en fases anteriores se utiliza para escribir código funcional que da lugar a componentes operativos. Durante esta fase escribir código no es exclusivo, también hay que interpretar correctamente el análisis y diseño que se haya realizado para asegurarse de que cada componente cumpla con los requisitos establecidos.

Durante esta etapa, el software comienza a tomar forma real y tangible. Es aquí donde se valida si las decisiones de diseño fueron acertadas, si la arquitectura es adecuada y si el sistema responde correctamente bajo condiciones reales. También es una fase crítica para detectar errores o carencias que no se tuvieron en cuenta durante el diseño o el modelado, y para realizar ajustes que aseguren la estabilidad y el rendimiento del software por lo que se requiere precisión, seguimiento de buenas prácticas de programación, control de versiones y pruebas constantes, teniendo esto último una fase entera dedicada dentro del contexto del proceso de desarrollo software.

Dentro de esta fase se identifican una gran variedad de tareas, aunque las típicas en esta etapa son la interpretación de los requisitos y traducción del diseño a código, estructuración de componentes internos del sistema, desarrollo frontend y backend, creación e implementación de bases de datos, integración entre componentes, control de versiones, configuración de entornos de desarrollo, documentación técnica, entre otros.

Numerosos estudios han tenido como objetivo recopilar la evaluación de publicaciones sobre la aplicación de LLMs en el proceso de desarrollo software, en concreto en tareas de generación de código, auto completado de código, etcétera. Según el estudio recopilatorio ya mencionado, *Large Language Models for Software Engineering: A Systematic Literature Review*, destaca que de los 395 estudios recopilados sobre el uso de LLMs en la ingeniería del software, 247 son relacionados a tareas de desarrollo e implementación de software.

Cuadro 1: Cantidad de estudios de la fase de implementación que han aplicado LLMs a estas tareas. Fuente adaptada de Xinyi Hou et al.

Fase	Tareas
Implementación y desarrollo software	Generación de código (118), Auto completado de código (22), Resumen de código (21), Búsqueda de código (12), Traducción de código (12), Comprensión de código (8), Síntesis de programas (6), Inferencia de API (5), Recomendación de API (5), Edición de código (5), Representación de código (3), Generación de comentarios de código (2), Recomendación de código (2), Generación de nombres de métodos (2), Estimación de puntos de historia ágil (1), Aumento de documentación de API (1), Deficiencias en documentación de API (1), Análisis de datos (1), Extracción de entidades y relaciones de API (1), Generación de controladores de prueba (1), Generación de grafo de flujo de control (1), Normalización de identificadores (1), Inferencia de tipos (1), Generación de instrucciones (1), Otros (14)

En este cuadro se observa que más del 60 % de estudios recopilados, están relacionados a

tareas de implementación de software, siendo un total de 247. Esta proporción resulta interesante ya que refleja la implicación y uso actual de los LLMs en esta fase concreta.

Debido a la popularidad de la capacidad de los LLMs para producir soluciones coherentes y precisas en tareas de codificación, se pueden utilizar en tareas anteriormente mencionadas o mostradas en el cuadro anterior, pero especialmente pueden ser utilizados especialmente en tareas de generación automática de código a partir de descripciones en lenguaje natural, completar automáticamente código a partir de fragmentos proporcionados, generar APIs<sup>5</sup>, refactorizar código e inferir tipos de datos.

Por ello, para este proyecto se ha decidido evaluar el uso de ciertos LLMs en esta fase, en concreto se ha hecho uso de Llama y Mistral, dada su eficacia en tareas de generación y auto-completado de código. Estos modelos destacan no solo por su capacidad para producir sintaxis correcta en diversos lenguajes de programación, sino también por su habilidad para comprender el contexto funcional del código, mantener coherencia semántica en bloques extensos y adaptar sus salidas a distintos estilos, características y habilidades necesarias en el desarrollo software asistido por IA.

### 2.2.5. Pruebas de software

Las fase de pruebas de software es un proceso crucial en el proceso de desarrollo software que implica la evaluación del software desarrollado. El objetivo principal de estas pruebas es identificar defectos o errores en el sistema de software que podrían conducir a un comportamiento incorrecto o inesperado, asegurando si el software cumple los requisitos establecidos o no. La fase de pruebas es iterativa y puede involucrar múltiples repeticiones de las mismas.

Durante esta fase, se identifican distintos tipos de pruebas, cada una orientada a validar aspectos específicos del desarrollo. Entre las pruebas más comunes se encuentran las pruebas unitarias, que consisten en evaluar de forma aislada cada unidad o componente del software para asegurar que funcione correctamente de acuerdo con los requisitos. A continuación, las pruebas de integración se encargan de verificar que los distintos módulos o componentes interactúan adecuadamente entre sí cuando se integran en conjunto. Posteriormente, se realizan las pruebas del sistema, en las cuales se evalúa el comportamiento del software como un único sistema, incluyendo todos los componentes internos y las posibles dependencias externas.

---

<sup>5</sup>*Application Programming Interface*

Finalmente, las pruebas de aceptación las cuales son habituales realizar en entornos de pre-producción (entornos del cliente), permiten validar que el sistema desarrollado cumple con los requisitos definidos. Adicionalmente, existen otro tipos de pruebas tales como las pruebas funcionales, las pruebas de rendimiento, las pruebas de seguridad o las pruebas de accesibilidad, entre otras.

Las tareas típicas de esta fase incluyen diseñar los escenarios de pruebas para realizar la preparación de los casos de prueba, que consiste en implementar y configurar técnicamente los escenarios de prueba de acuerdo con los diseños establecidos. Tras esto, se inicia la fase de ejecución de pruebas, utilizando los casos preparados para obtener los resultados finales de las pruebas.

Crear manualmente las pruebas unitarias puede resultar un proceso lento y exigente para muchos, lo que motiva la necesidad de automatización. Los LLMs se han aplicado recientemente para la generación automatizada de pruebas unitarias. Según el artículo *Software Testing With Large Language Models: Survey, Landscape, and Vision* [27] el cual revisa el uso de LLMs en tareas de pruebas software través de un análisis detallado de 102 estudios relevantes de pruebas software mediante LLMs, donde se exploran los enfoques más comunes en los que se han aplicado estos modelos de IAG, destacando especialmente tareas como la preparación de casos de prueba y la reparación automática de programas. Asimismo, se analizan los tipos de LLMs utilizados y los tipos de prompts utilizados en los estudios. En general, según el artículo, los resultados obtenidos muestran que aproximadamente un tercio de los estudios utiliza los LLMs mediante esquemas de preentrenamiento o *fine-tuning*, mientras que los demás emplean técnicas de ingeniería de *prompts* para interactuar con los modelos y orientar su comportamiento hacia resultados deseados. En cuanto a la ingeniería de prompts, las estrategias de *zero-shot learning* y *few-shot learning* son las más utilizadas, mientras que otros avances como cadena de pensamiento y la autoconsistencia se emplean con poca frecuencia [27].

Dentro del uso de LLMs para pruebas software se utilizan aquellos que siguen las arquitecturas *encoder-decoder* y *decoder-only* ya que soportan tareas generativas. No se considera la arquitectura *encoder-only* porque no es capaz de manejar tareas generativas.

Dado que los LLMs han mostrado resultados prometedores en tareas como la generación de código, y considerando que tanto la generación de código como la generación de casos de

pruebas unitarios implican la creación de código fuente, investigaciones recientes han ampliado el ámbito de la generación de código para incluir también la generación de pruebas unitarias. Por ello, entre las tareas típicas de esta fase, los LLMs se han utilizado en tareas de generación de casos de prueba, generación de pruebas unitarias, corrección de *bugs* dentro del código y aserción de pruebas.

Finalmente, los LLMs que se han utilizado para esta fase de pruebas del software son principalmente Mistral, ya que sigue una arquitectura de tipo *decoder-only*, lo que le permite manejar tareas generativas de manera eficiente. Esto lo hace adecuado para una amplia variedad de tareas como generación de código de casos de prueba unitarios, preparación de escenarios de prueba y reparación de código, [27], [28], [29]. Además, LLMs como Llama también han sido revisados en la evaluación [27], por lo que, debido a su arquitectura *decoder-only*, se ha utilizado para tareas de generación de código, depuración o reparación de código, y generación de pruebas unitarias.

### 2.3. Métricas de evaluación de LLMs

Para la evaluación de los LLMs se va a emplear una serie de métricas y aspectos relevantes, entre ellos el registro del tiempo de ejecución de cada tarea, tanto por parte del autor de este proyecto como por parte de los LLMs.

Las métricas que se han empleado para medir el desempeño y calidad de los modelos de inteligencia artificial generativa en el proceso del desarrollo software en comparación al desarrollo del caso práctico por parte del autor son las siguientes:

- **Completitud:** esta medida evalúa en qué grado el contenido generado por los modelos se ajusta a los elementos presentes en una solución de referencia. Estos elementos pueden variar según la tarea y fase: fragmentos de texto, entidades, atributos, fragmentos de código, entre otros. La completitud permite verificar si el resultado producido por el modelo cubre adecuadamente los componentes esperados según el análisis de referencia. En este trabajo, se utilizará como base el desarrollo realizado por el autor, considerándose como la implementación de referencia. A partir de esta, se solicitará mediante el uso de *prompts* a los modelos que proporcionen sus soluciones y, en función de eso, se realizará un análisis que determine el porcentaje de completitud, es decir, qué propor-

ción de los elementos relevantes de la solución de referencia están presentes en la salida generada por los LLMs. Los elementos generados por los LLMs que no se hayan tenido en cuenta en el análisis y desarrollo realizado por el humano, no se contabilizan para la aplicación de esta métrica. Según la naturaleza de cada tarea o fase del proceso de desarrollo software, podrán aplicarse versiones más específicas o variantes ponderadas de la fórmula.

$$\text{Compleitud} = \frac{\text{n}^\circ \text{ de elementos coincidentes}}{\text{n}^\circ \text{ total de elementos}} \times 100$$

- **Corrección:** esta métrica mide la capacidad del software para comprobar la precisión en la ejecución de las funcionalidades u otros elementos de los modelos tal y como fueron definidos. Para evaluarla, se categorizarán las salidas generadas por el LLM en comparación con el desarrollo del autor. Para esta métrica, se tiene en cuenta todos los requisitos y funcionalidades generadas aunque no coincidan con las utilizadas como referencia. Las categorías que se utilizarán para clasificar las respuestas generadas por los modelos de IAG son:
  1. **Incorrecto:** la funcionalidad está ausente o es totalmente errónea.
  2. **Parcialmente correcto:** la funcionalidad está presente, pero presenta errores o defectos que afectan parcialmente su comportamiento y ejecución.
  3. **Correcto:** la funcionalidad está bien implementada y cumple con los resultados esperados.
  4. **Excelente:** la funcionalidad está correctamente implementada y, además, presenta mejoras o consideraciones adicionales no contempladas originalmente.
- **Rendimiento:** esta medida comprueba la eficiencia y velocidad de la ejecución en las distintas fases del proceso de desarrollo de software.

En los apartados correspondientes a la evaluación del uso de los LLMs en las distintas tareas de cada fase, se han explicado la manera de evaluar cada contenido generado por los modelos a partir de estas métricas, considerando qué se ha tenido en cuenta para la evaluación.

## 2.4. Retos del uso de LLMs

Los modelos de IAG están emergiendo como herramientas clave en el ámbito de la ingeniería del software, ofreciendo nuevas posibilidades para automatizar tareas, asistir en distintas tareas y optimizar flujos de trabajo. Sin embargo, esta integración y uso no está carente de desafíos. La literatura revisada sobre la aplicación de LLMs en fases del proceso de desarrollo software ponen de manifiesto una serie de limitaciones y barreras técnicas que deben ser abordadas para lograr una integración efectiva y responsable uso de estas tecnologías.

Uno de los principales retos está relacionado con el tamaño de los modelos y su despliegue. A medida de que los LLMs han evolucionado, sus requisitos computacionales también se han incrementado. El entrenamiento de estos modelos exige grandes cantidades de recursos, requiriendo miles de horas de uso de GPU intenso durante semanas, lo cual representa un coste muy elevado para muchos. Otro reto es la ambigüedad inherente a la generación de código ya que muchas tareas pertinentes a la fase de implementación del software permiten múltiples soluciones válidas, lo que puede confundir a los modelos y derivar en código sintácticamente correcto, pero funcionalmente incorrecto, afectando así la efectividad de generar código mediante modelos de inteligencia artificial generativa. [26]

Otro reto es la necesidad de desarrollar LLMs orientados a tareas concretas y dominios específicos. La mayoría de los modelos actuales se entrenan con estrategias generalistas, lo que les permite abarcar una amplia variedad de lenguajes y tareas, pero también los sobrecarga con información irrelevante para contextos específicos. Esto puede provocar inconsistencias entre los objetivos del entrenamiento previo y las tareas de *fine-tuning*. Por ejemplo, en tareas de edición de código, los modelos suelen estar entrenados para generar fragmentos completos a partir de una descripción, lo que dificulta su adaptación a tareas que implican únicamente cambios parciales en el código existente. Por ello, se recomienda explorar el desarrollo de LLMs diseñados específicamente para tareas como la generación de código o actualización de pruebas unitarias, con objetivos de entrenamiento más enfocados. En los estudios citados, se observa una proporción desigual en la aplicación de LLMs en diversas fases. Los estudios mencionados recopilan gran cantidad de investigaciones que hacen mayor énfasis en tareas relacionadas a fases de implementación como generación de código, traducción de código, completado de código automatizado, generación de casos de prueba unitarios, entre otros, mientras que su

aplicación en otras tareas complejas como análisis y especificación de requisitos, generación de casos de uso y diseño de sistemas no están lo suficientemente estudiadas para el uso de LLMs. [23] [26]

Además, es importante señalar un reto práctico específico en el contexto de este TFG. Aunque el objetivo principal del proyecto no es desarrollar modelos, sino evaluarlos, el uso de LLMs gratuitos y de código abierto implica una serie de consideraciones adicionales previas a su aplicación. En particular, muchos de estos modelos requieren pruebas preliminares de funcionamiento y aporte de contexto o ejemplos para iniciar su evaluación en tareas concretas de las distintas fases del proceso de desarrollo software.



# 3

## Análisis y Diseño

Esta sección recoge principalmente dos partes. Se muestra el análisis, diseño y definición manual de distintas tareas de fases del proceso de desarrollo software, entre las que se encuentran el catálogo de requisitos, identificación y especificación de casos de uso, modelado de clases de análisis, diagramas de secuencia, modelo físico de datos y diseño de formatos de pantalla. La segunda parte recopila el resultado obtenido en cada *prompt* de los LLMs seleccionados, es decir, primero se muestra el desarrollo manual sobre el que se validan los resultados obtenidos y generados por la IA en las tareas de cada fase y, por último, se muestran los reportes de la evaluación teniendo en cuenta las métricas de este estudio. Cabe mencionar que el mismo *prompt* se utiliza en los dos modelos para que no haya diferencia en la entrada. Asimismo, del desarrollo manual se muestra solo una parte de cada tarea realizada y no todo el desarrollo. Esta decisión se tomó para no saturar el contenido del capítulo. El desarrollo manual completo realizado al respecto de las tres fases de este capítulo se encuentran el **Apéndice B**.

### 3.1. Caso práctico

Para evaluar a los LLMs en cada fase, se ha hecho uso del siguiente caso práctico.

#### 3.1.1. Funcionalidades de la aplicación

El caso práctico que se ha desarrollado es una aplicación web para una empresa ficticia dedicada al servicio de diversas actividades turísticas. Dentro de la aplicación se distinguen distintos roles. El catálogo de usuarios es:

- **Responsable:** Es el responsable de la empresa y de la aplicación web, dispondrá de todos los permisos y será el principal encargado de gestionar los empleados administrativos. Puede realizar cualquier funcionalidad que se haya implementado.

- **Empleados:** Los empleados son administrativos dentro de la empresa. En la aplicación web están encargados de gestionar a los monitores de las actividades, los calendarios de estos, gestionar la información de actividades turísticas, supervisión de comentarios y gestión de reservas. Además, pueden utilizar herramientas de análisis de datos implementadas en la web.
- **Monitores:** Dentro de la empresa son los encargados de realizar las actividades turísticas en función de su especialización. En la aplicación web, visualizan el calendario de actividades que se le ha asignado y los clientes de cada una.
- **Clientes:** Son los usuarios que deciden hacer uso de la aplicación y pueden agregar comentarios y visualizar todas las reservas que haya realizado.
- **Usuario Anónimo:** Este es el usuario final que no se registra el cual puede realizar funciones de reserva, pago y búsqueda.

Las principales funcionalidades de esta aplicación web son:

- **Gestión de empleados administrativos:** El responsable de la empresa tiene la capacidad para supervisar la información de los empleados a través de operaciones de creación, actualización, eliminación y lectura.
- **Gestión de monitores:** Los empleados realizan las mismas operaciones de gestión que el responsable con los empleados, pero aplicado a los monitores. Además, estos empleados son encargados de asignar las actividades y visualizar la información de ellos.
- **Gestión de usuarios:** Los empleados supervisan mediante operaciones los usuarios registrados en la aplicación.
- **Análisis de datos:** Tanto el responsable de la empresa y los empleados pueden realizar análisis de datos sobre los datos almacenados de la empresa.
- **Gestión de actividades:** El objetivo principal de la aplicación web es ofrecer actividades turísticas, mostrando tareas, su disponibilidad según el día y las tarifas por persona. Los empleados de la empresa realizan actividades de gestión sobre las actividades, además, son capaces de modificar y actualizar el precio de las tarifas y la disponibilidad si fuera necesario.

- Reservas online de actividades turísticas: La aplicación web implementa la búsqueda de actividades turísticas a través de un buscador integrado. Asimismo, los usuarios son capaces de reservar las actividades y gestionar el pago.
- Gestión de pago: El pago de las reservas de las actividades es de manera online y no reembolsable, con posibilidad de generar factura.
- Registro de usuarios: Los clientes pueden registrarse en la aplicación web para guardar su historial de reservas y añadir comentarios.
- Comentarios en las actividades: Las actividades turísticas en la web cuentan con un sistema de comentarios donde los clientes podrán añadir nuevos comentarios o responder a existentes. Estos comentarios son supervisados por los empleados.
- Historial de reservas: Los clientes registrados que hayan reservado actividades disponen de un historial de reservas pasadas y futuras.

### 3.1.2. Enunciado

El caso práctico que se ha utilizado para evaluar distintos modelos de inteligencia artificial generativa consiste en el desarrollo de una aplicación web para una empresa ficticia del sector turístico. Por ello, se han recogido las principales funcionalidades de la aplicación y formulado un enunciado el cuál sirve para introducir el ejercicio y como entrada textual para los modelos.

A continuación, se muestra el enunciado al caso práctico:

TurismoLLM<sup>6</sup>, empresa dedicada al servicio de diversas actividades turísticas, decide reformatizar sus servicios y encarga desarrollar una aplicación Web para realizar de manera más sencilla la gestión de recursos, empleados, horarios, actividades y reservas turísticas.

Las actividades se reservarán a través de un sistema de reservas online desde la aplicación web. Para reservar no hace falta registrar una cuenta de usuario de cliente, un usuario anónimo puede realizar las reservas que quiera. Los clientes tendrán un historial de reservas, pasadas y futuras. El pago de las reservas es no reembolsable y se dispondrá de dos métodos de pago; pago mediante transferencia bancaria y Paypal de los cuales se generará una factura de pago.

---

<sup>6</sup>Nombre que se ha creado para la empresa ficticia

Para cada actividad se almacenará la hora, descripción de la actividad, tarifa, imágenes, duración y día de la semana en la que se realiza dicha actividad. Adicionalmente, se administrarán las actividades por categorías, diferenciándose entre sí. Las actividades disponen de precios y duraciones fijas, aunque en ocasiones justificadas, se podrán actualizar estos campos.

Los empleados de la empresa son los encargados de la gestión de monitores, actividades y reservas online. El empleado estará encargado de gestionar las actividades y calendario de actividades que se asigna a un monitor. En general, se encarga de realizar la gestión de monitores y en todo momento dispondrá de una tabla con los monitores de la empresa.

Un empleado puede realizar reservas para clientes en el momento. También, se encarga de gestionar las reservas en el sistema y de revisar los comentarios de las actividades existentes, pudiendo borrar aquellos no apropiados.

Los usuarios pueden dejar comentarios en las actividades. Por cada cuenta de usuario, independientemente de si es cliente o trabajador de la empresa, se almacenará nombre, apellidos, DNI o NIE, correo electrónico y teléfono móvil. Para la gestión de monitores se almacena la misma información, pero con el añadido de que se requieren los estudios para gestionar la asignación de actividades en función de si está cualificado o no para llevar a cabo dicha actividad.

La empresa mantendrá un histórico de las reservas que se hayan realizado en la empresa para las tareas de análisis de datos e historial de reservas de cada cliente. De estas reservas se almacenará la actividad reservada, tipo de usuario que realizó la reserva, día de la semana y fecha.

Se distingue cliente de usuario anónimo aquel que se registra en la aplicación.

El responsable de la organización puede realizar todas las gestiones descritas anteriormente y, gestionar sus empleados.

Finalmente los empleados y responsable de la empresa pueden utilizar herramientas de análisis de datos incorporadas en la plataforma web para visualizar datos en gráficas y tablas.

### **3.2. Catálogo de requisitos**

Los requisitos que se han definido tras el análisis del caso práctico (sección 3.1.2) son los siguientes. Se muestra una versión reducida de cada tipo de requisito, excepto de los de información. Sin embargo, para visualizar el catálogo completo de cada uno de los tipos de requisitos,

se recomienda visitar la sección correspondiente en el apéndice B.

### Requisitos funcionales

Parte<sup>7</sup> de los requisitos funcionales obtenidos después del estudio y análisis del caso práctico son los que se muestran la siguiente imagen.

<p style="text-align: center;">&lt;&lt;requirement&gt;&gt; <b>Registro de nuevo usuario</b></p> <p>Text = "Un usuario anónimo debe poder introducir su correo y contraseña para que se pueda dar de alta en el sistema." ID = "RF-17" source = "" kind = "Functional" verifyMethod = "" risk = "" status = "Proposed"</p>	<p style="text-align: center;">&lt;&lt;requirement&gt;&gt; <b>Gestión de empleados</b></p> <p>Text = "El responsable de la aplicación Web debe ser capaz de gestionar los empleados de la empresa. En la gestión se incluyen operaciones de creación de empleados, lectura de su información, actualizar información de los empleados y eliminar a los mismos." ID = "RF-01" source = "" kind = "Functional" verifyMethod = "" risk = "" status = "Proposed"</p>
<p style="text-align: center;">&lt;&lt;requirement&gt;&gt; <b>Gestión de reservas online</b></p> <p>Text = "Permite a los empleados leer las reservas que se hayan realizado" ID = "RF-03" source = "" kind = "Functional" verifyMethod = "" risk = "" status = "Proposed"</p>	<p style="text-align: center;">&lt;&lt;requirement&gt;&gt; <b>Realizar reservas</b></p> <p>Text = "Los usuarios, registrados o no, podrán acceder a la aplicación Web para realizar reservas de las actividades turísticas disponibles en cada día." ID = "RF-21" source = "" kind = "Functional" verifyMethod = "" risk = "" status = "Proposed"</p>
<p style="text-align: center;">&lt;&lt;requirement&gt;&gt; <b>Gestión de pagos online</b></p> <p>Text = "Los usuarios, registrados o no, podrán realizar pagos de las reservas de actividades turísticas que quieran participar. La plataforma proporcionará dos métodos de pago los cuales solo se podrá elegir uno.  Los pagos son NO reembolsables." ID = "RF-10" source = "" kind = "Functional" verifyMethod = "" risk = "" status = "Proposed"</p>	<p style="text-align: center;">&lt;&lt;requirement&gt;&gt; <b>Pago mediante Paypal</b></p> <p>Text = "Los usuarios que escojan la opción de pago con Paypal serán redirigidos a un área de pago mediante el servicio Paypal." ID = "RF-11" source = "" kind = "Functional" verifyMethod = "" risk = "" status = "Proposed"</p>

Figura 3: Muestra de requisitos funcionales obtenidos tras el análisis del caso práctico. Fuente: elaboración propia.

### Requisitos no funcionales

Algunos<sup>8</sup> de los requisitos no funcionales obtenidos son:

<sup>7</sup>Muestra de requisitos. Visitar apéndice B.

<sup>8</sup>Visitar apéndice B para visión completa



Figura 4: Muestra de requisitos no funcionales obtenidos tras el análisis del caso práctico. Fuente: elaboración propia.

### Requisitos de información

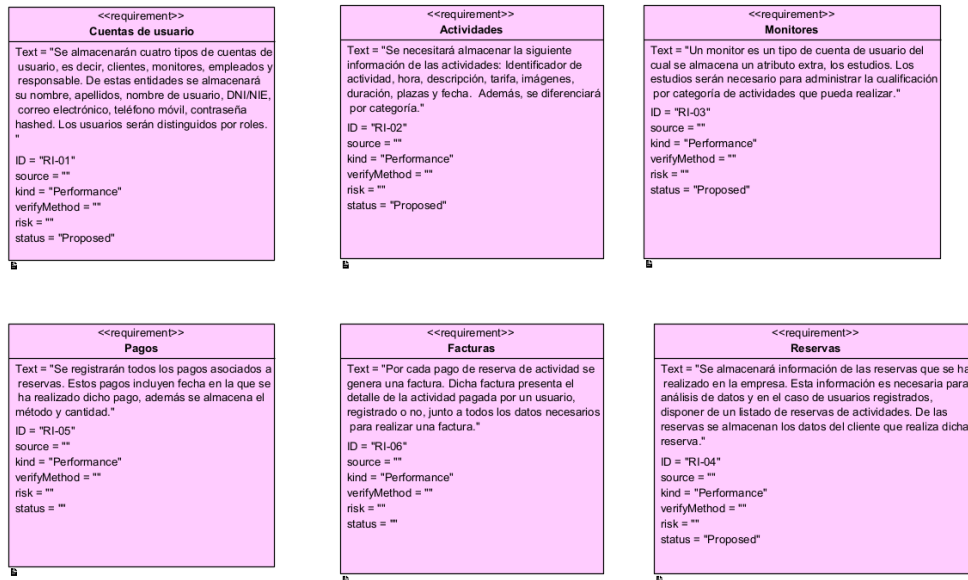


Figura 5: Requisitos de información obtenidos tras el análisis del caso práctico. Fuente: elaboración propia.

### 3.3. Casos de uso

Un caso de uso es una manera formal de representar requisitos funcionales, en concreto, cómo se relaciona el sistema con su entorno. Sirven para documentar el sistema y muestra los distintos usuarios que interactúan con el mismo. Los elementos que incluyen los diagramas de casos de uso son actores, casos de uso y relaciones entre estos. [22].

En la figura 6 se muestran los casos de uso de dos actores, Responsable de la organización y Empleado de la empresa turística. Un responsable puede realizar todas las funcionalidades de todos los actores del sistema. Como se muestra en la figura 7 se puede distinguir los siguientes actores, Monitor y Cliente. Un monitor es un empleado de la empresa encargado de realizar las actividades turísticas, están gestionados por los empleados. Un cliente no es más que un usuario anónimo que ha decidido registrarse en el sistema para visualizar un historial de reservas que haya realizado y dejar comentarios en actividades. Por último, en la figura 8 se visualiza el actor enfocado a servir como el usuario final de la aplicación web, y puede realizar distintos casos de uso. Todos los usuarios de la aplicación pueden realizar sus funciones.

Finalmente, después de definir los requisitos funcionales, se han identificado los siguientes casos de uso:

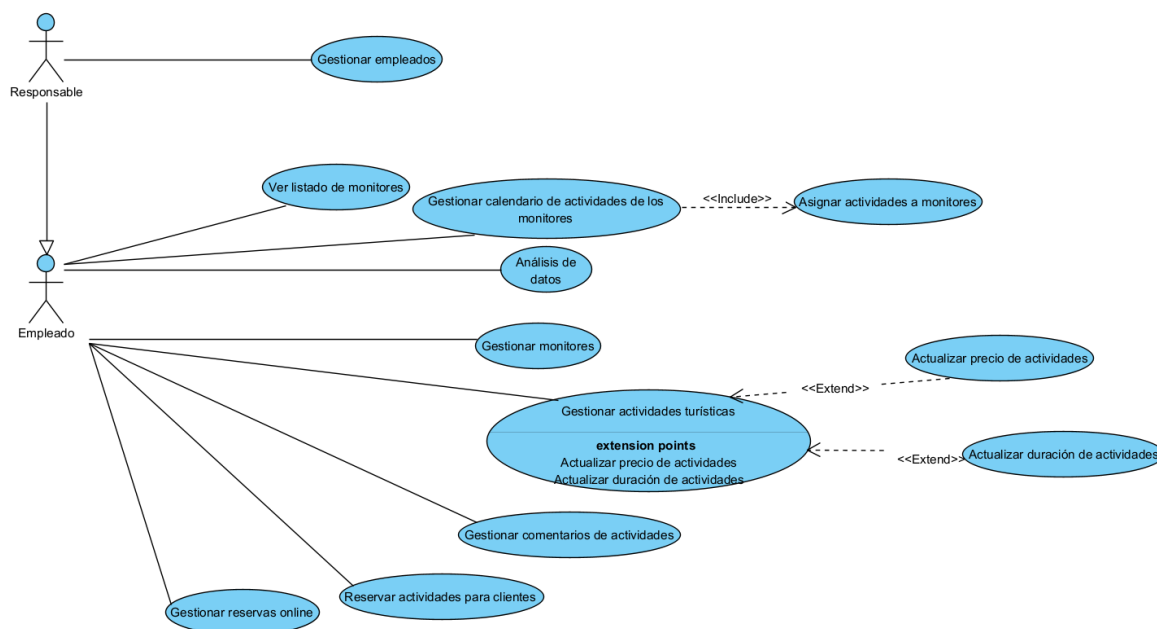


Figura 6: Diagrama de casos de uso: Responsable y Empleado. Fuente: elaboración propia.

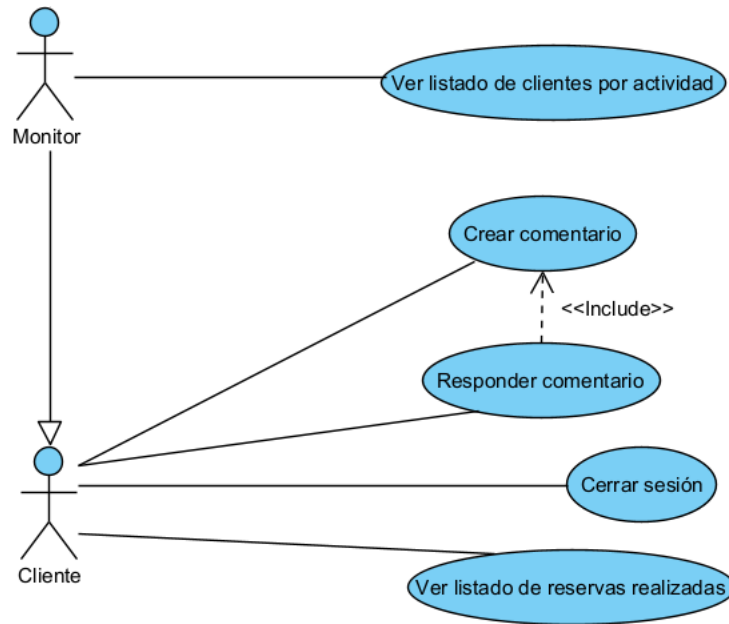


Figura 7: Diagrama de casos de uso: Monitor y Cliente. Fuente: elaboración propia.



Figura 8: Diagrama de casos de uso: Usuario Anónimo (no registrado). Fuente: elaboración propia.

### **Especificación casos de uso**

Para la especificación de casos de uso se sigue el mismo procedimiento, en esta sección se muestra una versión resumida, en concreto de una de las especificaciones para no saturar el cuerpo de este capítulo. Para visualizar la especificación de los casos de uso más relevantes, es necesario acudir al apéndice **B**.

La especificación de casos de uso cuenta con la estructura mostrada a continuación con la especificación del caso de *Registro de nuevo usuario*:

<b>Título</b>	Registro de nuevo usuario
<b>Descripción</b>	Un usuario anónimo decide registrarse en la plataforma web para mantener un historial de sus reservas.
<b>Pre-condición</b>	No estar registrado en el sistema.
<b>Post-condición</b>	Se crea su cuenta con las credenciales aportadas durante el registro y cuando vuelva a iniciar sesión, habrá entrado como cliente en el sistema.
<b>Prioridad</b>	Alta
<b>Escenario principal</b>	
<b>Escenario de registro de nuevo usuario</b>	
<ol style="list-style-type: none"> <li>1. El usuario en la página web hace clic en 'Registrarse'</li> <li>2. El sistema carga y muestra un formulario de registro</li> <li>3. El usuario introduce sus datos en los campos correspondientes</li> <li>4. El usuario hace clic en 'Crear cuenta'</li> <li>5. El sistema valida los datos</li> <li>6. El sistema crea la cuenta y muestra mensaje de confirmación al usuario</li> <li>7. El sistema redirige al usuario a la ventana de inicio de sesión</li> </ol>	
<b>Escenarios alternativos</b>	
<b>Escenario alternativo de registro de usuario ya existente</b>	
<ol style="list-style-type: none"> <li>6a. El sistema verifica que los datos introducidos como correo electrónico o nombre de usuario ya se encuentran en la base de datos del sistema</li> <li>6b. El sistema muestra un mensaje de error de cuenta ya existente</li> </ol>	
<b>Escenario alternativo datos incorrectos</b>	
<ol style="list-style-type: none"> <li>6a. El sistema comprueba que los datos introducidos no se ajustan al formato válido de los campos.</li> <li>6b. El sistema muestra un mensaje de error de datos inválidos</li> </ol>	
<b>Clases de análisis</b>	
<b>A. Clases de entidad</b>	Usuarios, Clientes
<b>B. Clases de control</b>	ControllerUsuarios
<b>C. Clases de interfaz</b>	ViewPaginaPrincipal, ViewRegistroUsuario, ViewIniciarSesión

Cuadro 2: Especificación de casos de uso: Registro de nuevo usuario. Fuente: elaboración propia.

### 3.4. Modelos

Este apartado recoge los distintos modelos que se han elaborado tras la especificación de casos de uso. Se han realizado diagramas de clases de análisis, entidad y control los cuales están orientados a los procesos de negocios más relevantes del caso práctico propuesto. Por último, se muestra el diagrama del modelo físico de datos, propio de la fase de diseño de sistemas.

#### **Modelo de clases**

El modelo de clases es fundamental en el diseño de software orientado a objetos, ya que permite representar de forma estructurada los elementos del sistema. A través de él, se identifican clases, atributos pertinentes a cada una, funciones y métodos de cada clase y relaciones entre las distintas clases. Su propósito es ofrecer una visión clara y ordenada de la arquitectura del software, lo que facilita su desarrollo, mantenimiento y escalabilidad. Mantiene una gran relación con el modelo físico de datos, se diferencian en que el modelado de clases se utiliza para estructurar el código del sistema desde un enfoque de objetos, mientras que el modelo físico de datos se usa para definir cómo se guardarán esos datos de forma eficiente en una base de datos real. Ambos son complementarios, pero trabajan en diferentes capas del proceso de desarrollo software. En la figura 9 se muestra el diagrama de clases.

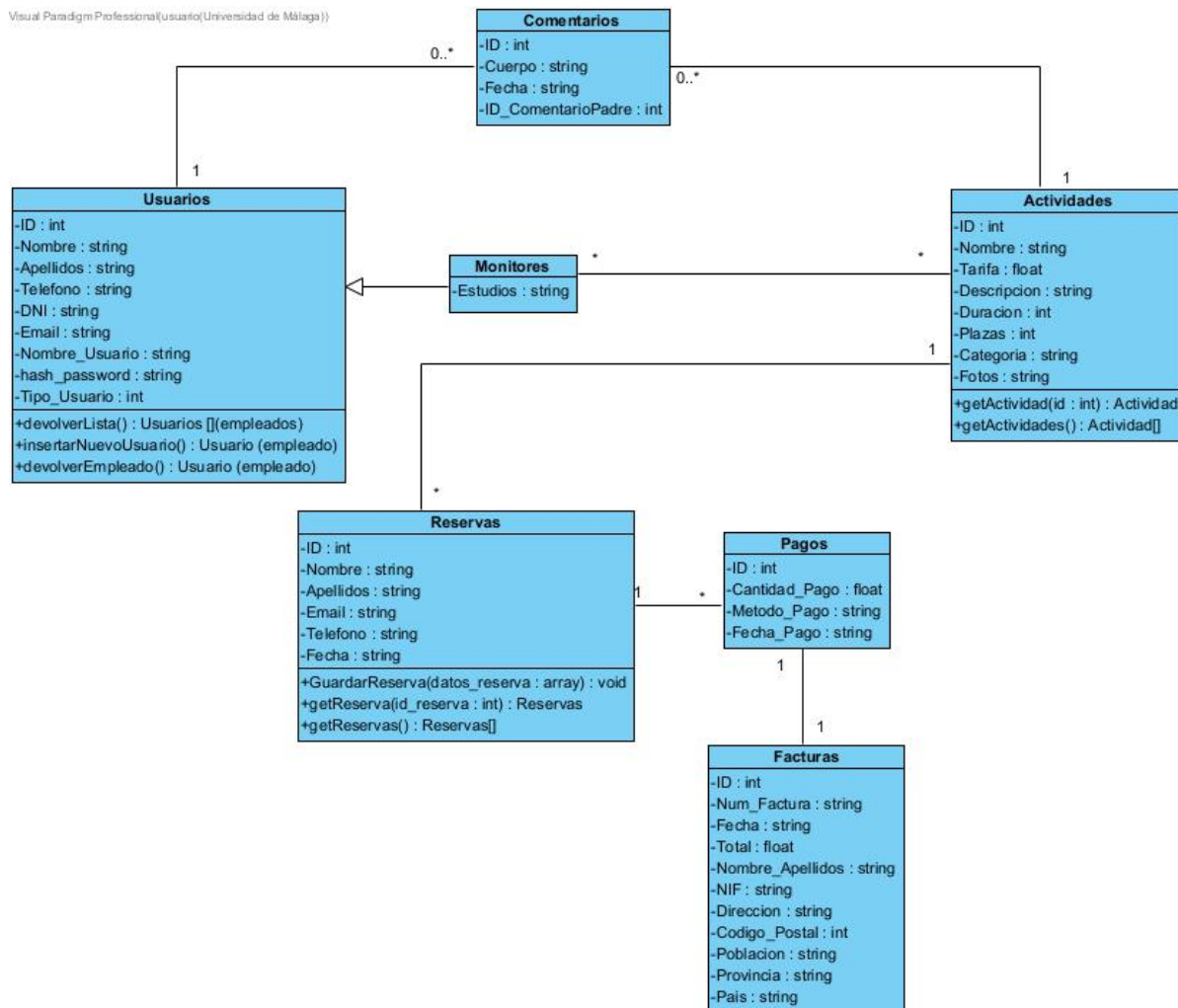


Figura 9: Diagrama de modelo de clases. Fuente: elaboración propia.

### Modelo de clases de control e interfaz

La capa de control es clave en la arquitectura del sistema, ya que coordina la lógica del negocio y actúa como enlace entre la presentación y las clases del sistema. Su modelado permite identificar claramente cómo se organizan las operaciones en módulos específicos, lo que favorece tanto la escalabilidad como el mantenimiento del software. Al estructurar la lógica en controladores separados, se consigue un desarrollo más ordenado y coherente, asegurando que cada funcionalidad se implemente de forma reutilizable y consistente. En la figura 10 se puede observar este modelo.

El modelado de clases de interfaz permite representar de forma clara cómo se conectan las diferentes pantallas o módulos de una aplicación. Este tipo de diagrama ayuda al equipo a definir los flujos de navegación, los puntos de contacto con el usuario y las funciones de cada

parte del sistema. Este diagrama se observa en la figura 11.

Para este proyecto, se ha hecho énfasis en clases, controladores e interfaces relacionados a procesos de negocio relevantes como los mencionados anteriormente para la especificación de casos de uso. Por ello, a continuación solo se muestran los modelados para respectivos procesos de negocio:



Figura 10: Diagrama de modelo de clases de control. Fuente: elaboración propia.

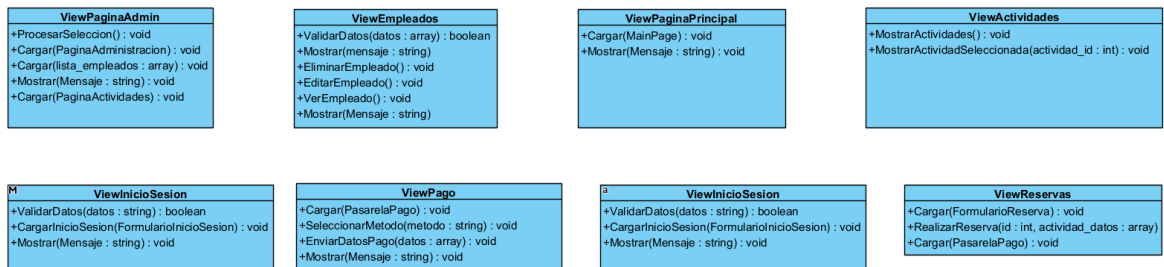


Figura 11: Diagrama de modelo de interfaces. Fuente: elaboración propia.

## Modelo físico de datos

El modelo físico de datos es esencial para estructurar y representar bases de datos, ya que define relaciones, claves y restricciones. Facilita la normalización y optimizan la gestión de la información. También guía a desarrolladores en la creación de objetos del sistema y consultas, además mejoran el rendimiento, seguridad y escalabilidad del sistema. Por ello, el modelo físico de datos se entiende como la representación lógica del sistema.

En el diagrama de la figura 12 se representa un sistema de reservas de actividades turísticas con gestión de usuarios, pagos y facturación.

La entidad **Usuarios** representa la información de los usuarios que interactúan con el sistema a través de la realización de reservas de actividades, pagos, comentarios y gestión de la página web. Entre los usuarios se distinguen cuatro tipos; **cliente**, **monitores**, **empleados**, **responsable**, los cuales almacenan la misma información, excepto los monitores que se requiere almacenar los estudios que tienen para poder impartir ciertas actividades. La gestión de usuarios y control de acceso se realiza a través del tipo de rol que tenga cada usuario del sistema. Por otra parte, **Actividades** registra los detalles de cada actividad ofrecida, como la descripción, duración y plazas disponibles. Las reservas realizadas por los usuarios se registran en **Reservas**, y estas pueden estar asociadas a pagos registrados en **Pagos**. A su vez, estas reservas generan facturas, cuyos datos se almacenan en **Facturas** con la información fiscal del cliente. En cuanto a las relaciones, un usuario puede hacer múltiples reservas, cada una vinculada a una actividad y un pago. Las actividades pueden ser asignadas a fechas específicas con monitores responsables, y el sistema permite que los comentarios entre usuarios estén relacionados mediante un campo de comentario padre, estableciendo así una estructura jerárquica.

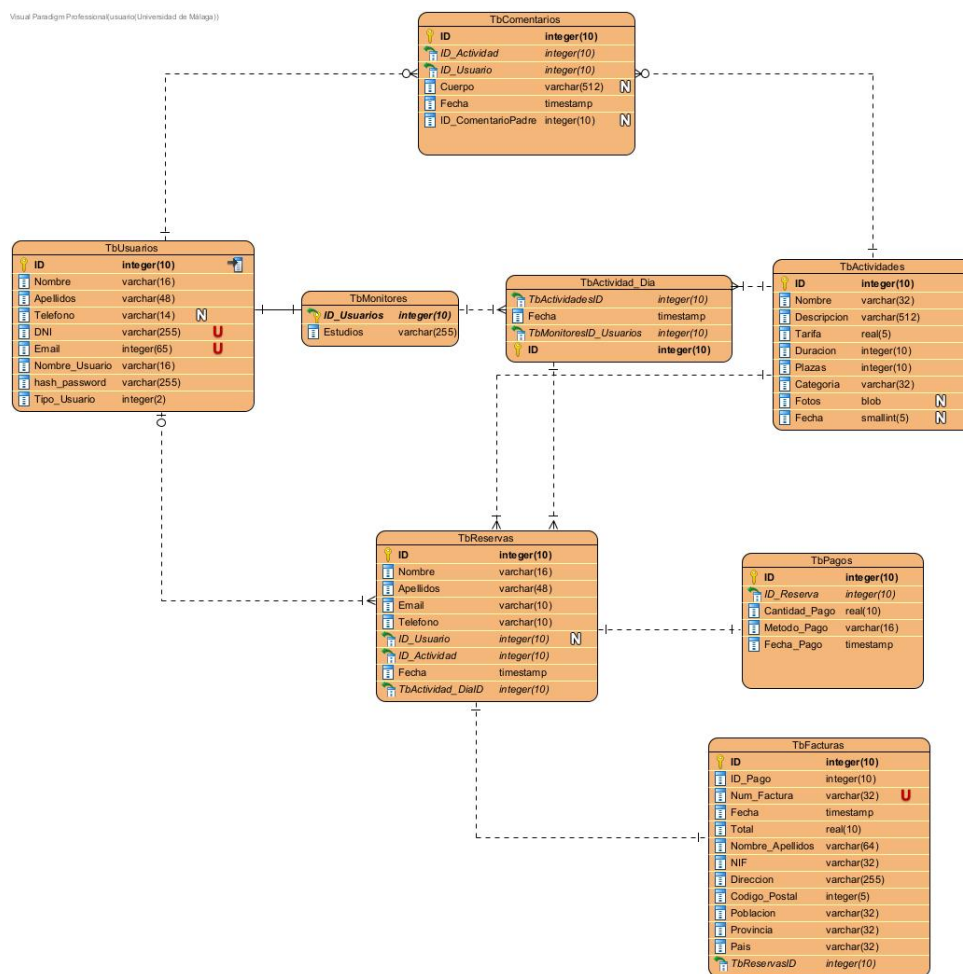


Figura 12: Diagrama de modelo físico de datos. Fuente: elaboración propia.

### 3.5. Diagramas de secuencia

El diagrama de secuencia es un modelo de interacción que tiene como finalidad representar el comportamiento dinámico de un sistema de información, destacando el orden en que los objetos intercambian mensajes [30]. Los diagramas de secuencia son esenciales durante el diseño de sistemas de información, ya que muestran gráficamente cómo interactúan los distintos elementos del sistema, como actores, controladores y bases de datos. Estos diagramas permiten entender de forma clara cómo se coordinan los mensajes y qué tareas realiza cada componente. Esto ayuda a comprender el funcionamiento general del sistema y a detectar posibles errores o mejoras desde etapas tempranas.

De nuevo, los diagramas que se han realizado han sido orientados a procesos de negocio interesantes a nivel de funcionalidad y relevantes en el caso práctico y, para reducir contenido

en este capítulo, se encuentran en el apéndice B. A continuación, se muestra un diagrama de secuencia acorde con uno de los procesos de negocio relevantes establecidos para el proyecto, además de su explicación.

En la figura 13 se describe el proceso que sigue un usuario anónimo (cliente) para registrarse en el sistema. Todo comienza cuando el usuario accede a la plataforma web y decide iniciar el proceso de registro y, como resultado, el sistema carga el formulario correspondiente. Una vez presentado el formulario, el usuario introduce los datos requeridos para su registro. Estos datos son enviados desde la vista de registro hacia el controlador de usuarios, que se encarga de gestionarlos. En caso de que los datos proporcionados sean correctos, el sistema devuelve una confirmación positiva al controlador. Acto seguido, el sistema procede a cargar el formulario de inicio de sesión, permitiendo que el nuevo usuario acceda directamente con sus credenciales recién creadas. Si los datos introducidos no hubieran sido válidos, el controlador lo comunica a la vista de registro y esta mostraría un mensaje de error.

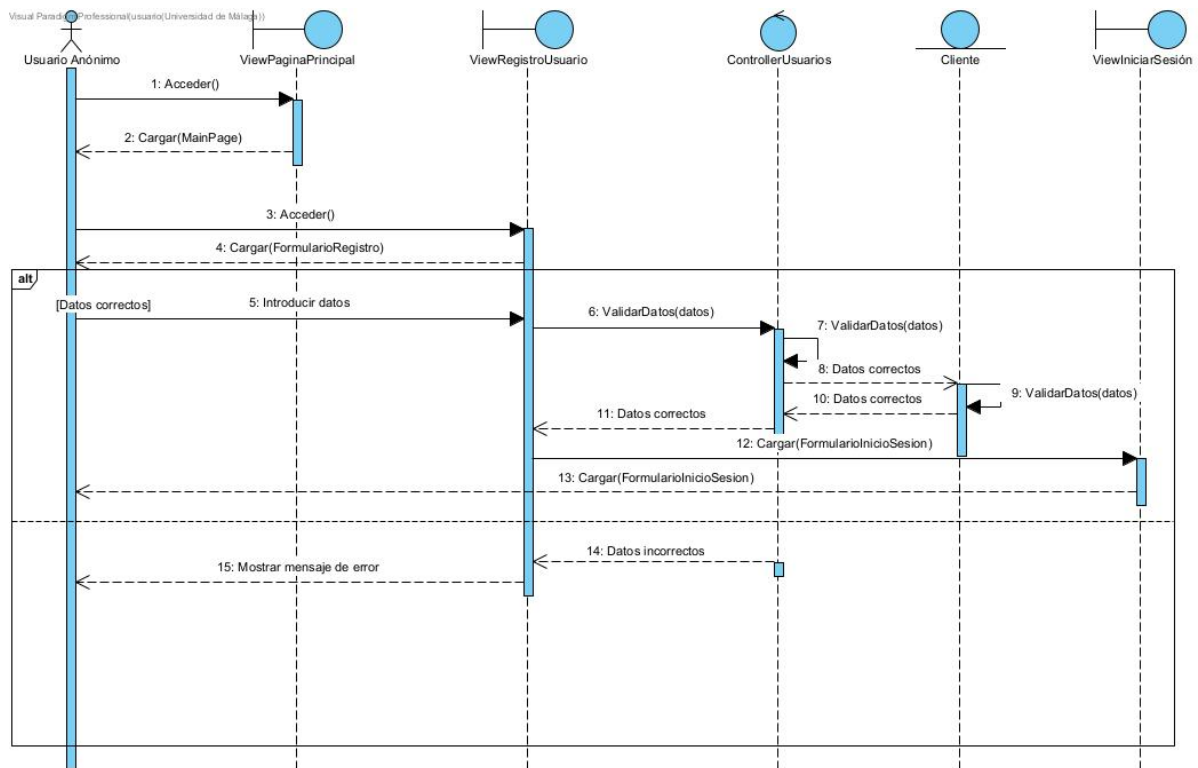


Figura 13: Secuencia de registro de nuevo usuario en el sistema. Fuente: elaboración propia.

### 3.6. Formatos de pantalla

El maquetado de interfaces de usuario es una parte esencial de todo proceso de desarrollo software ya que afecta directamente cómo los usuarios perciben y utilizan la plataforma.

A continuación, se presentan **algunos**<sup>9</sup> de los distintos formatos de pantalla desarrollados, los cuales fueron diseñados considerando los procesos de negocio relevantes mencionados con anterioridad. Algunas maquetas de interfaces de usuario desarrolladas son las siguientes:

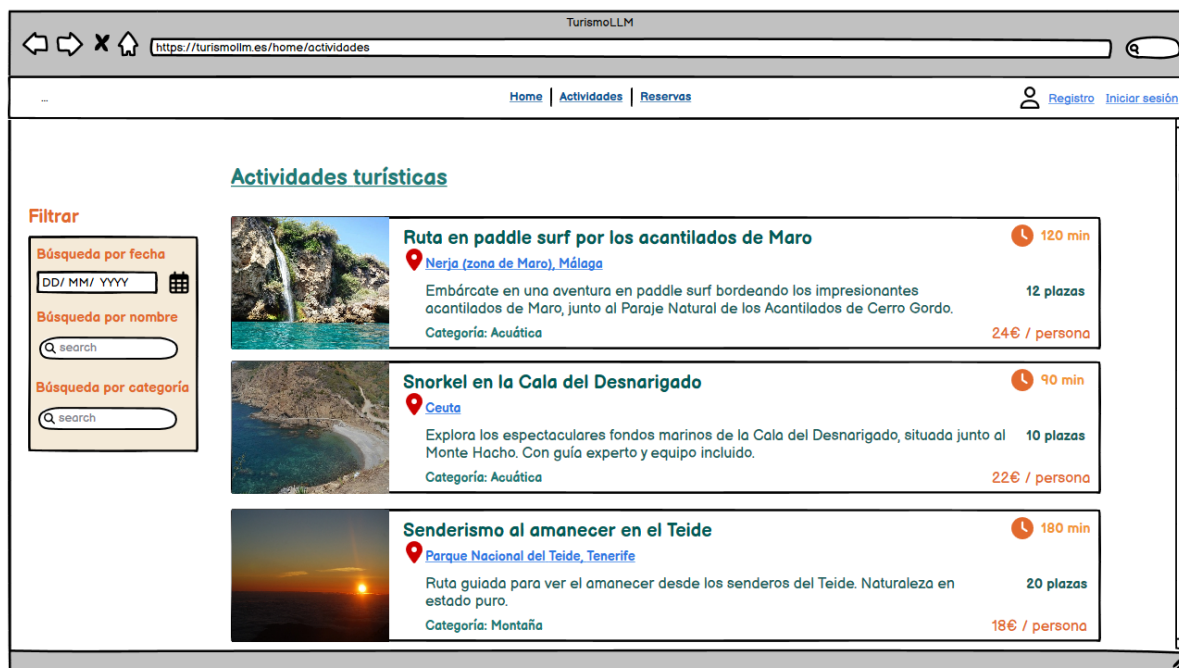


Figura 14: Maqueta de pantalla de página de actividades disponibles. Fuente: elaboración propia.

<sup>9</sup>Sección correspondiente en apéndice B.

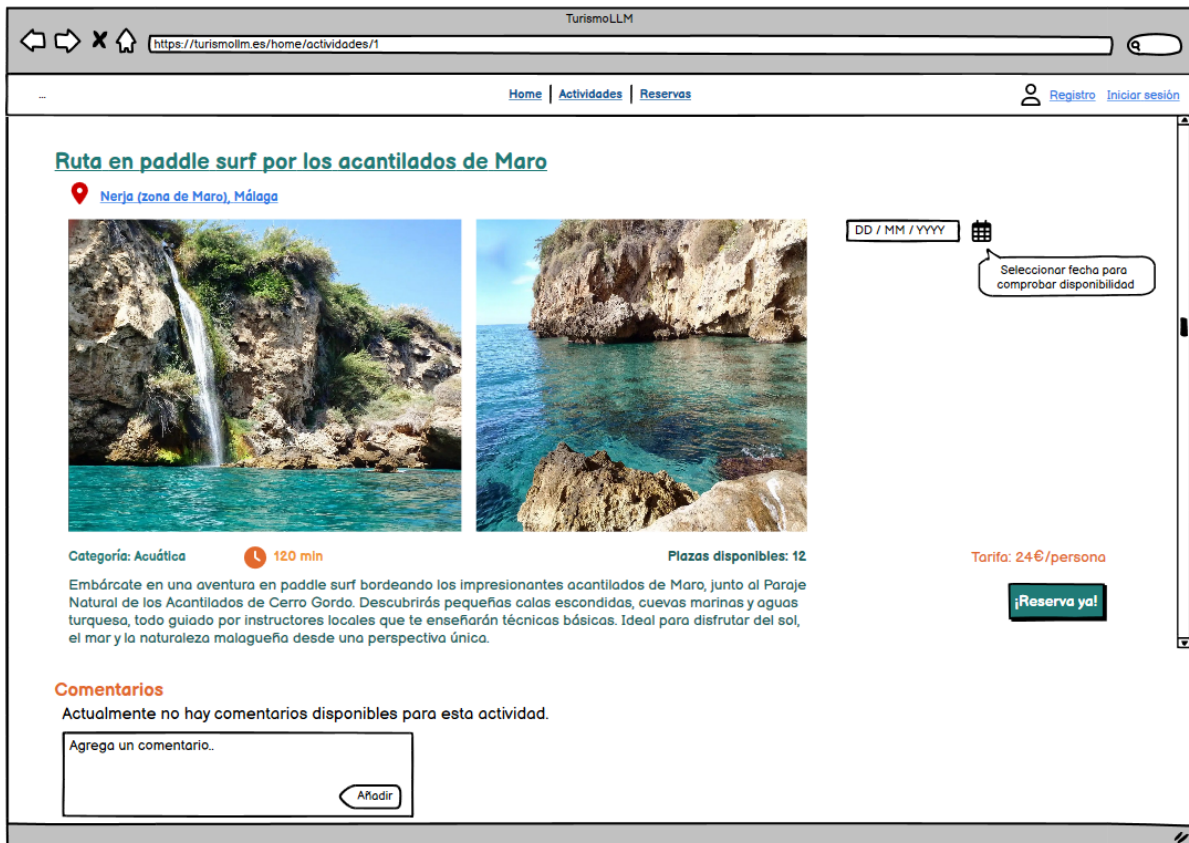


Figura 15: Maqueta de información de actividad seleccionada. Fuente: elaboración propia.

### 3.7. Uso de LLMs

En esta sección se recogen los resultados generados por los LLMs utilizando el mismo prompt en cada tarea de cada fase. La salida recogida es un resumen de aquellas más interesantes y relevantes. Todos los prompts y sus salidas se pueden encontrar en el Apéndice A. De esta forma, se ha aprovechado para mostrar el contenido generado por los modelos y a su vez, recoger los resultados en un apartado de evaluación del uso. La estructura de la sección es la siguiente.

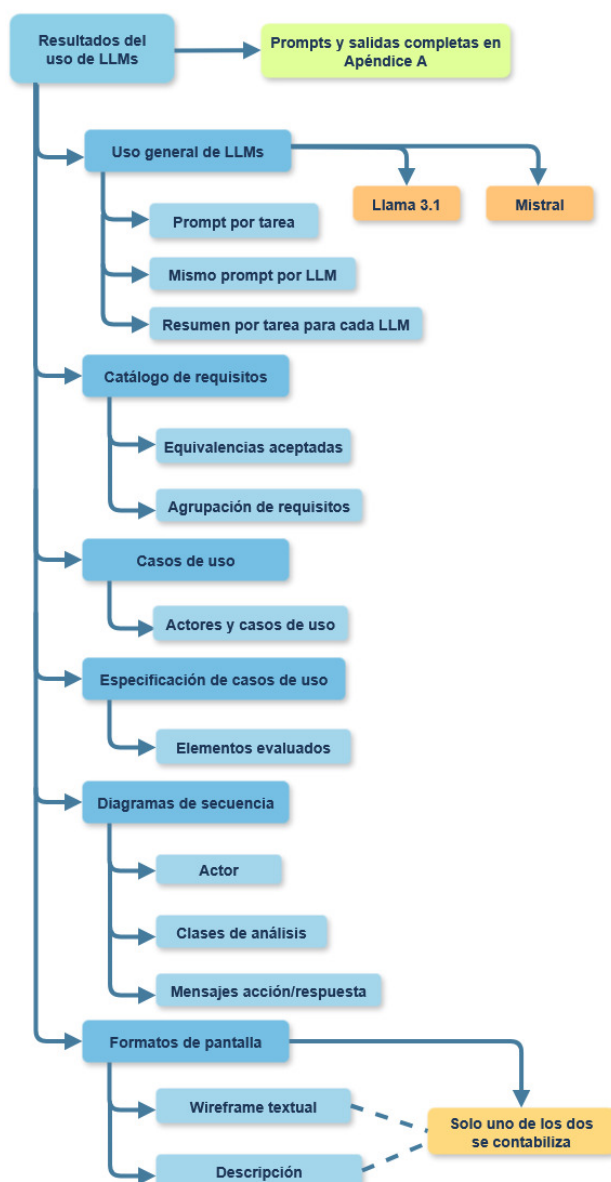


Figura 16: Diagrama de estructura del uso de LLMs en las fases 1, 2 y 3. Fuente: elaboración propia.

En la figura 16 se muestra la estructura de la sección *Uso de LLMs*, la cual contiene de manera gráfica la estructura que se ha seguido en este apartado. A rasgos generales, por cada LLM se utiliza el mismo *prompt* y por cada salida, se muestra resumida en los apartados *Llama 3.1* y *Mistral* de esta sección del capítulo. Por cada tarea, se muestra qué detalles y elementos se han tenido en cuenta para contabilizar en la evaluación del uso de LLMs en cada fase y, consiguiendo mostrar de manera resumida en cada apartado de LLM, el contenido relevante de manera detallada.

A continuación, se procede a explicar de manera detallada en los siguientes párrafos, cómo se ha generado el contenido para cada tarea y qué se ha valorado para su evaluación.

Comenzando con la **fase 1**, para el análisis de requisitos se ha tenido en cuenta que si un requisito validado de los proporcionados en el desarrollo manual es cubierto por la combinación de dos o más requisitos generados por los LLMs, se contabiliza como más de un acierto en la equivalencia, pero no en las métricas debido a la agrupación de requisitos. A su vez, no es necesario que los requisitos generados tengan una descripción exacta e idéntica para ser aptos para la aplicación positiva de las métricas, se han aceptado requisitos por el nombre correcto con una descripción parcial de la deseada.

En el caso de los casos de uso se ha dividido en dos. En primer lugar, se ha evaluado la generación de actores y casos de uso y, por otro lado, la especificación de los casos de uso. En el caso de los actores y sus casos de uso, se ha evaluado el número correcto respecto al desarrollo manual y, para la especificación de casos de uso, se ha decidido desglosar en elementos y subcomponentes para evaluar de forma objetiva el contenido. Los elementos son aquellos que aparecen en las tablas de especificación de casos de usos, en concreto, título, descripción, pre y post condiciones, pasos del escenario principal y aquellos de los escenarios alternativos. No se ha tenido en cuenta la prioridad y tampoco las clases de análisis para los casos de uso.

En la **fase 2**, para los diagramas de secuencia se ha decidido fijar los elementos a tener en cuenta. Por ello, los elementos considerados relevantes para la evaluación respecto a los diagramas de secuencia son el actor implicado y las clases de análisis involucradas. Asimismo, en el paso de mensajes, primero se ha tenido en cuenta contabilizar como uno aquellos mensajes que sean acción/respuesta y, también se ha tenido en cuenta aquellos no repetidos en los diagramas de secuencia de referencia. Por último, como el contenido generado por los modelos de IAG no es en formato UML ni similar, se han aceptado equivalencias en el texto,

admitiendo alias que fueran claros y objetivos.

Finalmente, para el caso de los formatos de pantalla, pertenecientes a la **fase 3**, en ocasiones, aunque se especificara en los *prompts* que se generara el *wireframe* textual de la interfaz deseada, no siempre se obtenía. Sin embargo, todas las salidas obtenidas, contenían una explicación o descripción general de lo que debería tener la interfaz. Por ello, para el caso de la evaluación, se ha decidido que si un LLM ha generado un *wireframe* textual y, a su vez una descripción general de la misma, si hubiera diferencias entre lo que se describe y se ha generado como pantalla, se utiliza el formato de pantalla textual (*wireframe*) generado y no la descripción. En el caso que no hubiera generación de pantalla en la salida, se tiene en cuenta únicamente la descripción.

Una vez explicados los detalles del uso de LLMs, se procede a mostrar en los siguientes apartados de manera resumida, el contenido generado por los LLMs para facilitar su posterior evaluación:

### 3.7.1. Llama 3.1

Comenzando con el catálogo de requisitos funcionales y no funcionales, el modelo generó un total de 15 requisitos funcionales y 10 no funcionales. Los requisitos coincidentes son:

#### Requisitos funcionales

Identificador	Título
RF-001	Reservas sin registro
RF-002	Historial de reserva
RF-003	Pago de las reservas
RF-005	Categorización de actividades
RF-006	Actualización de la tarifa y duración de las actividades
RF-007	Gestionar monitor
RF-008	Calificación necesaria para realizar la actividad
RF-009	Comentario sobre la actividad
RF-011	Asignar actividad
RF-014	Gestionar empleados
RF-015	Herramienta para visualizar los datos

Cuadro 3: Catálogo de requisitos funcionales coincidentes respecto al desarrollo manual. Fuente: Llama 3.1

Requisitos como responder comentarios se ha tenido en cuenta en otros como crear comentarios. Varios requisitos generados se agrupan en uno solo, pero se han tenido en cuenta para las métricas.

Las coincidencias de requisitos generados por la IA con los obtenidos en el análisis manual se presenta en la siguiente tabla, donde las filas representan los requisitos del estudio manual y las columnas hacen referencia a los identificadores de los requisitos funcionales generados por el LLM.

Debido al volumen de requisitos, se ha decidido que para no exceder el contenido del capítulo y páginas del mismo, solo se muestra este tipo de tabla para los requisitos funcionales debido a su importancia con las siguientes tareas de la primera fase del proceso de desarrollo software y las siguientes:

Requisito	RF-01	RF-02	RF-03	RF-04	RF-05	RF-06	RF-07	RF-08	RF-09	RF-10	RF-11	RF-12	RF-13	RF-14	RF-15
Gestión de Empleados														X	
Gestión de monitores															
Gestión de reservas online															
Reservar por parte de empleados hacia clientes															
Gestión de Actividades Turísticas															
Gestión de Actividades de Monitores							X				X				
Asignar monitor por actividad								X							
Ver listado de monitores															
Análisis de datos															X
Gestión de pagos online			X												
Pago mediante Paypal			X												
Pago mediante transferencia bancaria			X												
Generar factura de pago			X												
Gestión de comentarios de actividades															
Actualizar precios de actividades						X									
Ver listado de clientes por actividad turística															
Registro de nuevo usuario															
Iniciar sesión															
Cerrar sesión															
Buscar actividades					X										
Realizar reservas	X														
Crear comentarios									X						
Responder comentarios									X						
Ver listado de reservas pasadas		X													
Actualizar duración de las actividades						X									

Cuadro 4: Relación entre requisitos funcionales del análisis manual y coincidencias y equivalencias con requisitos funcionales obtenidos por Llama 3.1. Fuente: Llama 3.1

### Requisitos no funcionales

Identificador	Título
RNF-01	Seguridad.
RNF-04	Usabilidad.
RNF-05	Privacidad.
RNF-06	Soporte a diferentes dispositivos.
RNF-07	Almacenamiento adecuado de datos.

Cuadro 5: Catálogo de requisitos no funcionales coincidentes respecto al desarrollo manual.

Fuente: Llama 3.1

Los requisitos no funcionales de seguridad, privacidad y almacenamiento adecuado no se han decidido contabilizar como uno aunque su tengan un alto carácter de similitud en seguridad de la información.

Los casos de uso provienen de los requisitos funcionales por lo que el modelo ha identificado los siguientes actores y casos de uso:

#### Actores del sistema

Responsable de la aplicación web
Empleados
Monitores
Clientes registrados en el sistema
Usuarios anónimos (no registrados)

Cuadro 6: Lista de actores coincidentes respecto al desarrollo manual. Fuente: Llama 3.1

Llama identifica 6 actores de los cuales cinco coinciden con los obtenidos en el análisis manual.

#### Casos de uso y actores

Este modelo de IAG identifica correctamente los casos de uso y sus actores. Ha generado un total de 23 casos de uso, cada uno asociado a un actor. En la salida del *prompt* del apéndice A se observan los casos de uso generados, el actor que lo realiza y una breve descripción de la

funcionalidad a nivel operativo. No ha tenido en cuenta requisitos como la gestión de reservas y la gestión de monitores según la actividad.

Para la especificación, primero se han identificado los elementos generados y, por cada especificación de caso de uso, se ha contabilizado aquellos elementos que coinciden respecto al desarrollo manual.

<b>Casos de uso</b>	<b>Elementos generados</b>	<b>Coincidencias</b>
Registro de nuevo usuario	14	8
Gestionar empleados	15	10
Reservar actividad turística	20	10
Pago de reserva de actividad turística	18	9

Cuadro 7: Tabla de elementos de la especificación de cada caso de uso generado por Llama 3.1 y coincidencias con el modelo de referencias. Fuente: elaboración propia.

A continuación, se muestra un resumen de las coincidencias por cada caso de uso donde caso de uso 1 hace referencia a registro de nuevo usuario, caso de uso 2 a gestión de empleados y así en adelante.

<b>Elemento</b>	<b>CU1</b>	<b>CU2</b>	<b>CU3</b>	<b>C4</b>
Título	1	1	1	1
Descripción	1	1	1	1
Pre-condición	0	1	2	1
Post-condición	1	2	1	2
Escenario principal	3	5	3	3
Escenarios alternativos	2	0	2	1
<b>Total</b>	8	10	10	9

Cuadro 8: Coincidencia total de elementos de especificación de cada caso de uso generado por Llama 3.1. Fuente: elaboración propia.

En el reporte de evaluación de esta tarea se muestra el resultado del uso de Llama 3.1 en estas fases.

Ahora, se procede a mostrar de manera resumida lo obtenido en la generación de **diagramas de secuencia** por Llama 3.1. En la introducción de este capítulo se explicó lo que se ha

tenido en cuenta. Por ello, se muestra un resumen de cada uno de los diagramas generados por este modelo y su coincidencia con los diagramas de referencia.

La dinámica consiste primero en presentar por cada funcionalidad los elementos del diagrama de referencia presentes o equivalentes en el diagrama textual generado por el LLM. Posteriormente, se muestran los mensajes o acciones de los diagramas de secuencia de referencia, presentes o equivalentes en el diagrama generado. Aquellos elementos no mencionados en las siguientes tablas, se tienen en cuenta a la hora de la evaluación, concretamente con la corrección, estas tablas se presentan a modo de resumen.

<b>Actor y Modelos de Clase</b>
Usuario Anónimo
ViewPaginaPrincipal
ViewRegistroUsuario
ControllerUsuarios
Cliente
ViewIniciarSesión

Cuadro 9: Actor y modelos de clase del diagrama de **Registro de nuevo usuario** presentes en el diagrama de referencia generado por Llama 3.1. Fuente: elaboración propia.

<b>Mensajes</b>
Ir a registro y Cargar formulario
Introducir datos
Validar datos
Datos correctos → cargar inicio sesión
Datos incorrectos → mostrar error

Cuadro 10: Lista de mensajes (solicitud y respuesta) del diagrama de referencia **Registro de nuevo usuario** presentes en el diagrama generado por Llama 3.1. Fuente: elaboración propia.

Para este diagrama de secuencia, Llama 3.1 ha generado correctamente el actor requerido y ha utilizado las cinco clases de análisis indicadas en el *prompt*. Asimismo, de los mensajes

no repetidos del diagrama de referencia, la salida proporcionada por el LLM tiene presente las acciones indicadas, referentes a su secuencia 1 y 2, secuencia 3, secuencia 5, secuencia 7 y secuencia alternativa 2a.

<b>Actor y Modelos de Clase</b>
Responsable
ViewPaginaAdmin
ControllerUsuarios
Empleados
ViewEmpleados
Usuarios

Cuadro 11: Actor y modelos de clase del diagrama **Gestión de empleados** presentes en el diagrama generado por Llama 3.1. Fuente: elaboración propia.

<b>Mensajes</b>
Lectura (accede al listado en ViewPaginaAdmin)
Lectura (carga/muestra lista)
Creación
Creación / Actualización
Creación / Actualización (el sistema comprueba.)
Creación
Eliminación (muestra la lista de empleados actualizada)
Creación / Actualización
Eliminación
Eliminación
Eliminación (cancelar)
Lectura
Lectura

Cuadro 12: Lista de mensajes (solicitud y respuesta) del diagrama de referencia **Gestión de empleados** presentes en el diagrama generado por Llama 3.1. Fuente: elaboración propia.

Para este diagrama de secuencia, Llama 3.1 ha generado el actor y todas las clases de análisis presentes en el de referencia. Asimismo, para cada operación CRUD ha incluido las secuen-

cias no repetidas: secuencia 1, secuencia 2–3, secuencia 1–4, secuencia 5-6, secuencia 7 o 5a, secuencia 8, secuencia 3a, secuencia 4a o 5a, secuencia 1, secuencia 5, secuencia 3a, secuencia 4, secuencia 5.

<b>Actor y Modelos de Clase</b>
Usuario Anónimo
ViewPaginaPrincipal
ViewActividades
ControllerActividades
ViewReservas
ControllerReservas
ViewPago
ControllerFactura
Reserva

Cuadro 13: Actor y modelos de clase del diagrama de **Reserva de actividad turística** presentes en el diagrama de referencia generado por Llama 3.1. Fuente: elaboración propia.

<b>Mensajes</b>
Seleccionar actividad en página principal
Cargar página de actividades
Seleccionar actividad concreta
Comprobar disponibilidad
Introducir datos de reserva
Pulsar pagar y cargar pasarela
Seleccionar método e introducir datos pago
Validar pago y guardar reserva
Generar factura
Mostrar confirmación reserva

Cuadro 14: Lista de mensajes (solicitud y respuesta) del diagrama de referencia **Reserva de actividad turística** presentes en el diagrama generado por Llama 3.1. Fuente: elaboración propia.

El modelo para este diagrama de secuencia, respecto al diagrama original ha generado correctamente el actor y 9 clases de análisis a excepción de *Actividades*. En cuanto a las acciones, las secuencias coincidentes del contenido generado respecto al de referencia son los números 1, 2, 3, 4-5-6, 9, 10-11, 13 y 14.

<b>Actor y Modelos de Clase</b>
Usuario Anónimo
ViewPago
Factura

Cuadro 15: Actor y modelos de clase del diagrama de **Pago de reserva de actividad turística** presentes en el diagrama de referencia generado por Llama 3.1. Fuente: elaboración propia.

<b>Mensajes</b>
Seleccionar método de pago
Generar factura
Mostrar confirmación con factura

Cuadro 16: Lista de mensajes (solicitud y respuesta) del diagrama de referencia **Pago de reserva de actividad turística** presentes en el diagrama generado por Llama 3.1. Fuente: elaboración propia.

Llama 3.1 ha generado correctamente el actor y tres clases de análisis. Asimismo, solo ha generado correctamente tres mensajes equivalentes en su diagrama de secuencia respecto al de referencia.

Finalmente, los **formatos de pantalla**. En primer lugar, para el registro de nuevos usuarios (clientes), Llama 3.1 generó un *wireframe* textual. De esta maqueta se obtienen los siguientes elementos:

<b>Elementos</b>
Nombre de usuario
Correo electrónico
Número de teléfono
Contraseña
Repetir contraseña
DNI
Botón 'Crear cuenta'
Botón 'Ir a inicio de sesión'

Cuadro 17: Elementos del formulario de registro de nuevo de usuario generados por Llama 3.1. Fuente: Llama 3.1.

Ahora, para la gestión de empleados se han generado bastantes elementos debido a que incluye distintas maquetas de pantalla.

<b>Elemento</b>
Nombre de Usuario
Nombre y Apellidos
Email
Teléfono
Contraseña
Número de DNI
Rol: Empleado
Rol: Monitor
Botón Registrar Empleado

Cuadro 18: Elementos de pantalla del formulario de registro de nuevo empleado. Fuente: Llama 3.1.

<b>Elemento</b>
Contenido: Confirmación de eliminación
Botón Aceptar
Botón Cancelar
Contenido: Confirmación de eliminación exitosa

Cuadro 19: Elementos de pantalla eliminación de empleado. Fuente: Llama 3.1.

<b>Elemento</b>
Nombre y Apellidos
Correo Electrónico
Número de Teléfono
Dirección de Residencia (Adicional)
Fecha de Ingreso (Adicional)
Otros campos relevantes (Adicional)

Cuadro 20: Elementos de pantalla de visualización de empleado. Fuente: Llama 3.1.

<b>Elemento</b>
Campo nombre completo
Campo correo electrónico
Campo número de teléfono
Valor rol
Enlace cambiar contraseña
Botón Confirmar cambios

Cuadro 21: Elementos de la pantalla de edición de empleado. Fuente: Llama 3.1.

Continuando con el proceso de reservas, se tiene en cuenta la lista de actividades y el detalle por actividad.

<b>Elemento</b>
Filtro por categorías (lista desplegable)
Fecha
Nombre
Categoría

Cuadro 22: Elementos de la pantalla de actividades. Fuente: Llama 3.1.

<b>Elemento</b>
Fotos de la actividad
Título y descripción de la actividad
Duración y categoría de la actividad
Botón Reservar actividad

Cuadro 23: Elementos de la pantalla de detalles de la actividad. Fuente: Llama 3.1.

<b>Elemento</b>
Nombre y apellidos del usuario
Correo electrónico del usuario
Número de teléfono del usuario
Información adicional sobre la actividad
Botón: Realizar pago

Cuadro 24: Elementos de la pantalla de reserva de actividad turística. Fuente: Llama 3.1.

<b>Elementos</b>
Método de pago (PayPal / Transferencia bancaria)
Nombre completo
CIF
Dirección
Código postal
Población
País
Provincia
Botón Cancelar
Botón Realizar pago

Cuadro 25: Elementos de la pantalla de pago de reserva de actividad. Fuente: Llama 3.1.

### 3.7.2. Mistral

El modelo de inteligencia artificial generativa, Mistral, generó un total de 15 requisitos funcionales y 12 requisitos no funcionales.

Los requisitos coincidentes total o parcialmente son:

#### Requisitos funcionales

Identificador	Título
RQ001	Reserva Online
RQ003	Historial de Reservas
RQ004	Métodos de Pago
RQ005	Factura de Pago
RQ006	Actividades por Categorías
RQ007	Gestión de Actividades
RQ008	Gestión de Monitores
RQ009	Comentarios en las Actividades
RQ010	Registro de Usuario
RQ011	Gestión de Empleados
RQ013	Análisis de Datos - Reservas
RQ014	Análisis de Datos - Tabla y Gráficas
RQ015	Actualización de Precios y Duraciones

Cuadro 26: Catálogo de requisitos funcionales coincidentes respecto al desarrollo manual.

Fuente: Mistral

A continuación, se muestra la tabla de análisis de coincidencias y equivalencias entre los requisitos generados por el LLM Mistral y el análisis obtenido de forma manual:

Requisito	RQ-01	RQ-02	RQ-03	RQ-04	RQ-05	RQ-06	RQ-07	RQ-08	RQ-09	RQ-10	RQ-11	RQ-12	RQ-13	RQ-14	RQ-15
Gestión de Empleados											X				
Gestión de monitores											X				
Gestión de reservas online															
Reservar por parte de empleados hacia clientes															
Gestión de Actividades Turísticas															
Gestión de Actividades de Monitores								X							
Asignar monitor por actividad						X		X							
Ver listado de monitores															
Análisis de datos													X	X	
Gestión de pagos online				X											
Pago mediante Paypal				X											
Pago mediante transferencia bancaria				X											
Generar factura de pago					X										
Gestión de comentarios de actividades															
Actualizar precios de actividades															X
Ver listado de clientes por actividad turística															
Registro de nuevo usuario										X					
Iniciar sesión															
Cerrar sesión															
Buscar actividades															
Realizar reservas	X														
Crear comentarios									X						
Responder comentarios									X						
Ver listado de reservas pasadas			X												
Actualizar duración de las actividades															X

Cuadro 27: Relación entre requisitos funcionales del análisis manual con coincidencias y equivalencias con requisitos funcionales obtenidos por Mistral. Fuente: Mistral

No ha habido ninguna similitud con los **requisitos no funcionales** en Mistral.

### Casos de uso y actores

Los casos de uso y actores coincidentes generados por Mistral a partir de los requisitos funcionales obtenidos mediante el desarrollo manual son:

#### Actores del sistema

Responsable
Empleados
Monitores
Usuarios registrados
Usuarios anónimos

Cuadro 28: Lista de actores coincidentes respecto al desarrollo manual. Fuente: Mistral

El modelo Mistral ha identificado correctamente todos los actores involucrados en el sistema, sin generar más de los necesarios. Ahora bien, en los casos de uso ha generado un total de 19 casos de uso, los cuales coinciden con la mayor parte de los obtenidos en el proceso de análisis manual.

Casos de uso	Elementos generados	Coincidencias
Registro de nuevo usuario	15	10
Gestionar empleados	24	15
Reservar actividad turística	22	9
Pago de reserva de actividad turística	21	9

Cuadro 29: Elementos de la especificación de cada caso de uso generado por Mistral y coincidencias con el modelo de referencias. Fuente: elaboración propia.

<b>Elemento</b>	<b>CU1</b>	<b>CU2</b>	<b>CU3</b>	<b>C4</b>
Título	1	1	1	1
Descripción	1	1	1	1
Pre-condición	1	1	1	1
Post-condición	1	1	1	2
Escenario principal	5	11	2	3
Escenarios alternativos	1	0	3	1
<b>Total</b>	<b>10</b>	<b>15</b>	<b>9</b>	<b>9</b>

Cuadro 30: Coincidencia total de elementos de especificación de cada caso de uso generado por Mistral. Fuente: elaboración propia.

Para los **diagramas de secuencia**, el modelo Mistral muestra los siguientes elementos coincidentes con los diagramas de referencia:

<b>Actor y Clases de Análisis</b>
Usuario Anónimo
ViewPaginaPrincipal
ViewRegistroUsuario
ControllerUsuarios
Cliente
ViewIniciarSesión

Cuadro 31: Actor y modelos de clase del diagrama **Registro de nuevo usuario** presentes en el diagrama generado por Mistral. Fuente: elaboración propia.

<b>Mensajes</b>
Acceder a la web y cargar página principal
Ir a registro y Cargar formulario
Introducir datos
Validar datos
Datos correctos → cargar inicio sesión
Datos incorrectos → mostrar error

Cuadro 32: Lista de mensajes (solicitud y respuesta) del diagrama de referencia **Registro de nuevo usuario** presentes en el diagrama generado por Mistral. Fuente: elaboración propia.

Los elementos generados por Mistral para el primer diagrama muestra un alto grado de coincidencia con los diagramas de referencia. Para el actor y clases de análisis, genera todos. Para los mensajes no repetidos, también genera todos en las secuencias, en concreto, secuencias 1–2, secuencias 3–5, secuencia 6, secuencias 9–10, secuencia 11, secuencias 12–13 del diagrama textual generado.

<b>Actor y Clases de Análisis</b>
Responsable
ViewPaginaAdmin
ControllerUsuarios
Empleados (modelo)
ViewEmpleados

Cuadro 33: Actor y modelos de clase presentes en el diagrama **Gestión de empleados** generado por Mistral. Fuente: elaboración propia.

<b>Mensajes</b>
Nuevo usuario y Cargar formulario
Introducir datos de empleado
Validar datos
Crear empleado
Mostrar lista actualizada
Error por datos inválidos
Eliminar empleado
Confirmación de eliminación
Eliminar y refrescar lista
Cancelar eliminación
Ver empleado
Obtener empleado y mostrar ficha
Editar empleado y Cargar formulario
Actualizar emplead

Cuadro 34: Lista de mensajes (solicitud y respuesta) presentes en el diagrama **Gestión de empleados** generado por Mistral. Fuente: elaboración propia.

Ha incluido correctamente el actor y cinco clases de análisis y trece acciones del diagrama de referencia en su diagrama textual generado.

<b>Actor y Clases de Análisis</b>
Usuario Anónimo
ViewPaginaPrincipal
ViewActividades
ControllerActividades
Actividades
ViewReservas
ControllerReservas
ViewPago
ControllerFactura

Cuadro 35: Actor y modelos de clase presentes en el diagrama de **Reserva de actividad turística** generado por Mistral. Fuente: elaboración propia.

<b>Mensajes</b>
Acceder + cargar página principal
Seleccionar actividades + cargar página actividades
Seleccionar actividad concreta
Comprobar disponibilidad
Pulsar reservar + cargar formulario
Introducir datos de reserva
Validar datos de reserva
Pulsar pagar + cargar pasarela (parcial)
Validar pago + guardar reserva
Generar factura
Mostrar confirmación de reserva

Cuadro 36: Lista de mensajes (solicitud y respuesta) presentes en el diagrama de **Reserva de actividad turística** generado por Mistral. Fuente: elaboración propia.

En este diagrama de secuencia, Mistral ha generado correctamente el actor y las clases de análisis (a excepción de uno), y de las acciones del diagrama de referencia, referentes en su

diagrama textual a las secuencia 1, secuencia 2–5, secuencia 6–10, secuencia 11–15, secuencia 19, secuencia 20, secuencia 22–23 mensaje de error si inválidos, secuencia 20, secuencia 24, secuencia 25–26, secuencia 26.

<b>Actor y Clases de Análisis</b>
Usuario Anónimo
ViewPago
ControllerPago
Factura

Cuadro 37: Actor y modelos de clase presentes en el diagrama **Pago de reserva de actividad turística** generado por Mistral. Fuente: elaboración propia.

<b>Mensajes</b>
Acceder pasarela
Seleccionar método
Introducir datos de pago
Pulsar botón pagar
Enviar datos y validar pago
Generar factura
Confirmación con factura

Cuadro 38: Lista de mensajes (solicitud y respuesta) presentes en el diagrama **Pago de reserva de actividad turística** generado por Mistral. Fuente: elaboración propia.

Mistral para este último diagrama de secuencia ha incluido en su generación el actor, clases de análisis y todas las secuencias.

Finalizando con los **formatos de pantalla**:

Elementos de **registro de nuevos usuarios**:

<b>Elemento</b>
Name
Surname
Username
Email
Phone number
Password
DNI
Create Account
Login Now

Cuadro 39: Elementos del formulario de registro de nuevo de usuario generados por Mistral.

Fuente: Mistral.

Ahora, **CRUD empleados**:

<b>Elemento</b>	<b>Validación</b>
Username	Validación
Name and Surname	-
Email	Validación
Phone number	Validación
Password (hidden field)	-
DNI	Validación
Role (Employee, Monitor)	-
Register employee (button)	-
Confirmation message	-

Cuadro 40: Elementos de pantalla del formulario de registro de nuevo empleado. Fuente: Mistral

<b>Elemento</b>
Eliminar empleado botón
Confirm Delete Employee
Aceptar botón
Cancelar botón
Mensaje de confirmación
Botón de confirmación
Botón de cancelación
Employee has been deleted successfully.

Cuadro 41: Elementos de pantalla eliminación de empleado. Fuente: Mistral.

### **Editar empleado:**

<b>Elemento</b>
Botón "Atrás"
Nombre de usuario
Nombre
Apellidos
Correo electrónico
Teléfono móvil
Estado (Activo/Inactivo)
Botón "Editar información"
Botón "Eliminar empleado"

Cuadro 42: Elementos de la pantalla de edición de empleado. Fuente: Mistral.

<b>Elemento</b>
Nombre de usuario (editable)
Nombre completo (editable)
Correo electrónico (editable)
Número de teléfono (editable)
Botón “Editar información” con icono identificador
Campo oculto: contraseña (no editable)
Lista desplegable: rol del empleado (no editable)
Botón “Confirmar cambios”
Mensaje de éxito o error según validación del sistema

Cuadro 43: Elementos de la pantalla de visualización de empleado. Fuente: Mistral.

<b>Elemento</b>
Botones para filtrar por fecha, nombre o categoría
Imagen (panel de actividades)
Nombre (panel de actividades)
Categoría (panel de actividades)
Botón “Ver detalles”

Cuadro 44: Elementos del panel de actividades. Fuente: Mistral.

<b>Elemento</b>
Imagen (detalle de la actividad)
Nombre (detalle de la actividad)
Categoría (detalle de la actividad)
Duración
Descripción
Comentarios
Fechas disponibles
Precio por persona
Botón “Reservar actividad”

Cuadro 45: Elementos del detalle de la actividad. Fuente: Mistral.

<b>Elemento</b>
Nombre y apellidos
Correo electrónico
Número de teléfono
Información de la actividad
Botón “Realizar pago”

Cuadro 46: Elementos de la pantalla de reserva de actividad. Fuente: Mistral.

<b>Elemento</b>
Método de pago
Botón “Cancelar”
Importe, comercio, fecha
Descripción
Botón “Realizar pago”
Nombre completo (campo de texto)
CIF (campo de texto)
Dirección (campo de texto)
Código postal (campo de texto)
Población (campo de texto)
País (campo de texto)
Provincia (campo de texto)
Mensaje de confirmación

Cuadro 47: Elementos de la pantalla de pago de reserva. Fuente: Mistral.

### 3.8. Evaluación de resultados

En este apartado se ha recogido los resultados del uso de LLMs en el desarrollo de las distintas tareas de cada fase en los procesos de negocio más relevantes.

#### 3.8.1. Completitud

Para el caso de **LLama**, en la tarea de análisis y especificación de requisitos de la primera fase del proceso de desarrollo software se ha obtenido el siguiente resultado:

$$\text{Completitud REQ_FUNC} = \frac{15}{25} \times 100 = 60 \%$$

$$\text{Completitud REQ_NONFUNC} = \frac{5}{9} \times 100 = 56 \%$$

Para la completitud de los casos de uso se ha tenido en cuenta obtener correctamente el número de casos de uso por lo que el total del denominador hace referencia a la suma de

número de actores más número de casos de uso:

$$\text{Compleitud CASOS_USO} = \frac{28}{30} \times 100 = 93 \%$$

Llama 3.1 generó correctamente cinco actores del sistema más 23 casos de uso.

La evaluación del uso de **Mistral** en tareas típicas de esta fase (obtención de requisitos) es la siguiente, basándose en el contenido:

$$\text{Compleitud REQ_FUNC} = \frac{16}{25} \times 100 = 64 \%$$

$$\text{Compleitud REQ_NONFUNC} = \frac{0}{9} \times 100 = 0 \%$$

Para los actores y casos de uso en la completitud se ha obtenido lo siguiente:

$$\text{Compleitud CASOS_USO} = \frac{27}{30} \times 100 = 90 \%$$

Mistral generó correctamente cinco actores del sistema más 22 casos de uso.

La completitud de la especificación de casos de uso se ha evaluado de una forma ligeramente distinta. Para esta tarea, se va a realizar la evaluación de la siguiente forma. En primer lugar, a partir de la tabla de especificación de casos de uso utilizada, se ha decidido tener en cuenta elementos y contenido de cada uno. Los elementos se entienden como título, descripción, pre y post condición, los escenarios principales y alternativos. Dentro de cada elemento puede aparecer más de un punto o paso, por lo que estos en caso de coincidir, se contabilizarán como más de uno. El procedimiento consiste en comparar cada elemento y subcomponente generado por los modelos con los obtenidos en el desarrollo manual, y se considera coincidencia cuando el modelo reproduce el elemento esperado, aunque lo exprese con distinta redacción o un nivel de detalle diferente. En cambio, aquellos elementos o subcomponentes adicionales no contemplados en la referencia, no son contabilizados en el cálculo.

En primer lugar, se muestran en el siguiente cuadro los casos de uso y la cantidad de elementos en cada uno, correspondientes al desarrollo manual frente a los coincidentes en aquellos generados por los LLMs:

Casos de uso	Desarrollo manual	Llama 3.1	Mistral
Registro de nuevo usuario	16	8	10
Gestionar empleados	30	10	15
Reservar actividad turística	21	10	9
Pago de reserva de actividad turística	17	9	9

Cuadro 48: Comparación de de elementos coincidentes en la especificación de casos de uso entre Desarrollo Manual, Llama 3.1 y Mistral. Fuente: elaboración propia.

Cada celda representa la cantidad de elementos totales coincidentes respecto al desarrollo manual, siendo los valores de la primera fila la cantidad total de elementos por cada caso de uso.

Teniendo estos datos recopilados, la completitud de **la especificación de casos de uso** se obtiene de la siguiente forma:

Se calculó la media considerando el número total de elementos de referencia (16 + 30 + 21 + 17 = 84). El cálculo de la completitud de cada LLM se realizó de la siguiente forma:

Para el caso de **Llama 3.1** se ha obtenido:

$$\text{Completitud ESP\_CASOS\_USO} = \frac{(8 + 10 + 10 + 9)}{84} = \frac{37}{84} = 44,04\%$$

Para el caso de **Mistral** se ha obtenido:

$$\text{Completitud ESP\_CASOS\_USO} = \frac{(10 + 15 + 9 + 9)}{84} = \frac{43}{84} = 51,2\%$$

Continuando con los diagramas de secuencia, como se indicó en la sección de este capítulo referente al uso de LLMs, para los diagramas de secuencia de referencia se ha tenido en cuenta la cantidad de actores implicados, clases de análisis involucradas y mensajes (acciones) no repetidos.

Por ello, primero se muestra un resumen de los principales elementos de cada uno de los diagramas de secuencia de referencia más relevantes para posteriormente realizar la evaluación de cada modelo en la completitud:

Proceso	Actores	Clases de análisis	Acciones/Mensajes
Registro de nuevo usuario	1	5	6
Gestión de empleados	1	6	16
Reserva de actividad turística	1	9	13
Pago de reserva de actividad turística	1	3	7

Cuadro 49: Resumen de procesos, actores, clases de análisis y acciones/mensajes de los diagramas de referencia. Fuente: elaboración propia.

Para el caso de gestión de empleados como en realidad involucra cuatro procesos distintos (operaciones CRUD), se ha decidido mostrar el recuento total, no teniendo en cuenta clases de análisis repetidos o acciones duplicadas.

Una vez mostrado el resumen, se va a proceder a sumar las filas y calcular un recuento total para obtener la completitud general de cada LLM en esta tarea de su correspondiente fase. Por ello, el denominador consistirá en el recuento total de elementos en los diagramas de referencia y, el denominador, el recuento total de elementos coincidentes de cada LLM.

Para el denominador se obtiene:  $(1 + 5 + 6) + (1 + 6 + 16) + (1 + 9 + 13) + (1 + 3 + 7) = 69$ .

Ahora, se procede a calcular la completitud total de cada LLM para los diagramas de secuencia generados respecto a los de referencia.

**Llama 3.1** ha obtenido el siguiente resultado:

$$\text{Completitud DIG_SEQ} = \frac{(1 + 5 + 5) + (1 + 5 + 13) + (1 + 8 + 10) + (1 + 2 + 3)}{69} \times 100$$

$$\text{Completitud DIG_SEQ} = \frac{55}{69} \times 100 = 79,71 \%$$

**Mistral** muestra el siguiente resultado en la completitud de **diagramas de secuencia**:

$$\text{Completitud DIG_SEQ} = \frac{(1 + 5 + 6) + (1 + 4 + 14) + (1 + 8 + 11) + (1 + 3 + 7)}{69} \times 100$$

$$\text{Completitud DIG_SEQ} = \frac{62}{69} \times 100 = 89,85 \%$$

Finalizando la completitud de esta fase con los **formatos de pantalla**, se ha establecido un recuento total del universo de elementos de referencia:

<b>Formato de pantalla</b>	<b>Recuento total de elementos</b>
Registro de nuevo usuario	10
Gestión de empleados	26
Reserva de actividad turística	24
Pago de reserva de actividad	13

Cuadro 50: Recuento total de elementos por formato de pantalla del desarrollo manual. Fuente: elaboración propia.

Para el primer formato de pantalla, figura 72, no se ha tenido en cuenta el logo, pero sí todos los *inputs* de texto y tres botones, cancelar y los de crear cuenta o iniciar sesión.

Para el caso de la gestión de empleados, maqueta de la figura 74, no se ha tenido en cuenta esta tabla si no las maquetas referentes a las acciones CRUD de la gestión, figuras 75, 76, 77, 78. En estas maquetas se tienen en cuenta todos los botones e *inputs* de las referencias. Sin embargo, el panel lateral izquierdo no se ha contabilizado para esta evaluación.

Para las maquetas de reserva de actividad, sí que se ha tenido en cuenta algún formato adicional, como puede ser la vista de actividades a la que acceder para poder comprobar la disponibilidad de una actividad y reservarla, figuras 14, 15, 79. Por último, se ha tenido en cuenta toda la pantalla referente a la maqueta de la figura 80. Por cada filtro presente en la maqueta, se ha contabilizado como uno por separado y no como un conjunto. Asimismo, para la lista de actividades no se ha tenido en cuenta la cantidad de actividades, solo se evalúa una.

En general, no se tiene en cuenta ninguna imagen de las maquetas de pantalla de referencia, tampoco colores, aunque para el caso de las actividades se evalúa si mantiene un espacio reservado para las imágenes. En total, se va a tener en cuenta un recuento de 73 elementos.

Comenzando con la completitud de los **formatos de pantalla**, para el caso de Llama 3.1 se obtiene el siguiente resultado:

$$\text{Completitud FORM\_PNTLL} = \frac{(7 + (8 + 3 + 3 + 5) + (6 + 4 + 4) + 10)}{73} \times 100$$

$$\text{Compleitud FORM\_PNTLL} = \frac{50}{73} \times 100 = 68,49\%$$

Finalmente, para el caso de los formatos de pantalla de **Mistral** se obtiene:

$$\text{Compleitud FORM\_PNTLL} = \frac{(9 + (8 + 4 + 6 + 7) + (5 + 9 + 5) + 13)}{73} \times 100$$

$$\text{Compleitud FORM\_PNTLL} = \frac{66}{73} \times 100 = 90,41\%$$

### 3.8.2. Corrección

Las categorías de corrección que se utilizan al aplicar dicha métrica, son las mostradas en la sección 2.3. Para la corrección, se tienen en cuenta todos los requisitos y funcionalidades generadas aunque no coincidan con las utilizadas como referencia. Además, se va a hacer uso de las categorías establecidas en la sección 2.3.

Comenzando con la identificación y especificación de requisitos, se ha utilizado como base de referencia los catálogos de requisitos funcionales y no funcionales de la sección 3.2 de este capítulo.

Esta tabla es para ambos LLMs. Muestra el identificador del requisito, su nombre y dos columnas para cada modelo, la primera columna indica si está presente o no y la segunda su corrección (categoría de corrección).

ID	Nombre requisito	Presente: Llama	Corrección: Llama	Presente: Mistral	Corrección: Mistral
RF-01	Gestión de Empleados	Sí	Correcto - 3	Sí	Correcto - 3
RF-02	Gestión de Monitores	No	Incorrecto - 1	No	Incorrecto - 1
RF-03	Gestión de reservas online	No	Incorrecto - 1	No	Incorrecto - 1

RF-04	Reservar por parte de empleados hacia clientes	No	Incorrecto - 1	No	Incorrecto - 1
RF-05	Gestión de Actividades Turísticas	No	Incorrecto - 1	No	Incorrecto - 1
RF-06	Gestión de Actividades de Monitores	Sí	Correcto - 3	Sí	Correcto - 3
RF-07	Asignar monitor por actividad	Sí	Correcto - 3	Sí	Correcto - 3
RF-08	Ver listado de monitores	No	Incorrecto - 1	No	Incorrecto - 1
RF-09	Análisis de datos	Sí	Correcto - 3	Sí	Parcialmente correcto - 2
RF-10	Gestión de pagos online	Sí	Parcialmente correcto - 2	Sí	Correcto - 3
RF-11	Pago mediante Paypal	Sí	Parcialmente correcto - 2	Sí	Correcto - 3
RF-12	Pago mediante transferencia bancaria	Sí	Parcialmente correcto - 2	Sí	Correcto - 3
RF-13	Generar factura de pago	Sí	Parcialmente correcto - 2	Sí	Correcto - 3
RF-14	Gestión de comentarios de actividades	No	Incorrecto - 1	No	Incorrecto - 1
RF-15	Actualizar precios de actividades	Sí	Correcto - 3	Sí	Correcto - 3

RF-16	Ver listado de clientes por actividad turística	No	Incorrecto - 1	No	Incorrecto - 1
RF-17	Registro de nuevo usuario	No	Incorrecto - 1	Sí	Parcialmente correcto - 2
RF-18	Iniciar sesión	No	Incorrecto - 1	No	Incorrecto - 1
RF-19	Cerrar sesión	No	Incorrecto - 1	No	Incorrecto - 1
RF-20	Buscar actividades	Sí	Correcto - 3	No	Incorrecto - 1
RF-21	Realizar reservas	Sí	Correcto - 3	Sí	Correcto - 3
RF-22	Crear comentarios	Sí	Excelente - 4	Sí	Correcto - 3
RF-23	Responder comentarios	Sí	Excelente - 4	Sí	Correcto - 3
RF-24	Ver listado de reservas pasadas	Sí	Parcialmente correcto - 2	Sí	Correcto - 3
RF-25	Actualizar duración de las actividades	Sí	Correcto - 3	Sí	Correcto - 3

Para el análisis de requisitos no funcionales se ha obtenido la siguiente corrección:

ID	Nombre requisito	Presente: Llama	Corrección: Llama
RNF-01	Base de Datos del sistema	No	Incorrecto - 1
RNF-02	Web Responsive	Sí	Excelente - 4
RNF-03	Rendimiento de la aplicación Web	No	Incorrecto - 1

RNF-04	Seguridad de la aplicación Web	Sí	Correcto - 3
RNF-05	Seguridad en el pago	Sí	Correcto - 3
RNF-06	Usabilidad	Sí	Correcto - 3
RNF-07	Informar de errores	No	Incorrecto - 1
RNF-08	Roles de usuario	No	Incorrecto - 1
RNF-09	Control de acceso a los recursos	Sí	Parcialmente correcto - 2

Cuadro 52: Métrica de corrección de requisitos generados por los LLMs respecto al desarrollo manual. Fuente: elaboración propia.

Debido a que el modelo Mistral no fue capaz de identificar correctamente los requisitos no funcionales, no se ha tenido en cuenta para la realización de la tabla de corrección.

Teniendo en cuenta la corrección de casos de uso, en la siguiente tabla se muestran los casos de uso y actores identificados por cada LLM. La columna 'Nombre' representa el nombre del caso de uso utilizado como base para la evaluación. Asimismo, un caso de uso se considera parcialmente correcto si aunque el caso de uso esté bien identificado, su actor no es el correcto. Tampoco se ha tenido en cuenta la posibilidad de generalizar actores para la generación del contenido.

Aunque no se haya tenido en cuenta la generalización y herencia de funcionalidades de casos de uso para los actores, el modelo Mistral destaca en incluir más de un actor por caso de uso si fuera necesario, se puede ver en el apéndice A, apartado casos de uso de este LLM.

A continuación, en las siguientes páginas, se presentan las tabla de corrección de casos de uso y la corrección de la especificación de cada caso de uso por cada modelo, asignando cada elemento entre las distintas categorías. Seguido de esto, se muestra la evaluación de la corrección de especificación de casos de uso.

Asimismo, se muestra la evaluación de la corrección de los diagramas de secuencia y de los formatos de pantalla. Para los diagramas de secuencia se muestra en cada página horizontal,

en la medida de lo posible, la comparación de cada LLM en un diagrama específico. Para la corrección de esta tarea, se ha decidido eliminar la columna **Presente** debido a que no aportaba ningún valor adicional, para aprovechar el espacio en una página horizontal se ha decidido eliminar dicha columna.

El resto de la evaluación de la corrección de diagramas de secuencia se ha decidido no mostrar ya que la idea era la misma, es decir, dos tablas por página horizontal, una haciendo referencia a cada modelo en un diagrama. Sin embargo, por motivos de longitud de la memoria de este proyecto, se ha decidido que para no hacer tan largo este capítulo, mostrar directamente el resultado de la corrección de diagramas de secuencia en el apartado de evaluación de resultados. La misma lógica se ha seguido para los formatos de pantalla i.e. para evitar saturar la extensión del capítulo y por lo tanto de la memoria, se ha decidido mostrar únicamente un ejemplo de corrección y el resultado completo en su sección correspondiente.

Nombre del caso de uso	Actor	Presente: Llama	Categoría: Llama	Presente: Mistral	Categoría: Mistral
Gestionar empleados	Responsable	Sí	Correcto - 3	Sí	Correcto - 3
Ver listado de monitores	Empleado	Sí	Correcto - 3	No	Incorrecto - 1
Gestionar calendario de actividades de los monitores	Empleado	No	Incorrecto - 1	Sí	Parcialmente correcto - 2
Asignar actividades a monitores	Empleado	Sí	Correcto - 3	Sí	Correcto - 3
Análisis de datos	Empleado	Sí	Correcto - 3	Sí	Correcto - 3
Gestionar monitores	Empleado	Sí	Correcto - 3	Sí	Correcto - 3
Gestionar actividades turísticas	Empleado	Sí	Correcto - 3	Sí	Correcto - 3
Actualizar precio de actividades	Empleado	Sí	Correcto - 3	Sí	Correcto - 3
Actualizar duración de actividades	Empleado	Sí	Correcto - 3	No	Incorrecto - 1
Gestionar reservas online	Empleado	No	Incorrecto - 1	Sí	Correcto - 3
Reservar actividades para clientes	Empleado	Sí	Correcto - 3	Sí	Correcto - 3
Gestionar comentarios de actividades	Empleado	Sí	Correcto - 3	Sí	Correcto - 3
Ver listado de clientes por actividad	Monitor	Sí	Correcto - 3	No	Incorrecto - 1
Crear comentario	Cliente	Sí	Correcto - 3	Sí	Correcto - 3
Responder comentario	Cliente	Sí	Correcto - 3	Sí	Correcto - 3
Cerrar sesión	Cliente	Sí	Correcto - 3	Sí	Correcto - 3
Ver listado de reservas realizadas	Cliente	Sí	Correcto - 3	Sí	Excelente - 4
Buscar actividades	Usuario Anónimo	Sí	Correcto - 3	Sí	Correcto - 3
Realizar reserva	Usuario Anónimo	Sí	Correcto - 3	Sí	Correcto - 3
Realizar pago de reserva	Usuario Anónimo	Sí	Correcto - 3	Sí	Correcto - 3
Generar factura	Usuario Anónimo	Sí	Correcto - 3	Sí	Parcialmente correcto - 2
Pago mediante Paypal	Usuario Anónimo	Sí	Correcto - 3	Sí	Correcto - 3
Pago mediante transferencia bancaria	Usuario Anónimo	Sí	Correcto - 3	Sí	Correcto - 3
Registro de nuevo usuario	Usuario Anónimo	Sí	Correcto - 3	Sí	Correcto - 3
Iniciar sesión	Usuario Anónimo	Sí	Correcto - 3	Sí	Correcto - 3

Cuadro 53: Métrica de corrección de actores y sus casos de uso generados por los LLMs respecto al desarrollo manual. Fuente: elaboración propia.

Elemento CU	Presente: Llama	Corrección: Llama	Presente: Mistral	Corrección: Mistral
Título	Sí	Correcto - 3	Sí	Correcto - 3
Descripción	Sí	Correcto - 3	Sí	Correcto - 3
Precondiciones	Sí	Incorrecto - 1	Sí	Correcto - 3
Postcondiciones	Sí	Correcto - 3	Sí	Correcto - 3
Escenario principal	Sí	Correcto - 3	Sí	Correcto - 3
Escenarios alternativos	Sí	Correcto - 3	Sí	Correcto - 3

Cuadro 54: Corrección de la especificación del caso de uso **Registro de nuevo usuario** respecto al modelo de referencia. Fuente: elaboración propia.

Elemento CU	Presente: Llama	Corrección: Llama	Presente: Mistral	Corrección: Mistral
Título	Sí	Correcto - 3	Sí	Correcto - 3
Descripción	Sí	Correcto - 3	Sí	Correcto - 3
Precondiciones	Sí	Parc. correcto - 2	Sí	Incorrecto - 1
Postcondiciones	Sí	Correcto - 3	Sí	Correcto - 3
Escenario principal	Sí	Correcto - 3	Sí	Parc. correcto - 2
Escenarios alternativos	Sí	Incorrecto - 1	Sí	Incorrecto - 1

Cuadro 55: Corrección de la especificación del caso de uso **Gestionar empleados** respecto al modelo de referencia. Fuente: elaboración propia.

Elemento CU	Presente: Llama	Corrección: Llama	Presente: Mistral	Corrección: Mistral
Título	Sí	Correcto - 3	Sí	Correcto - 3
Descripción	Sí	Correcto - 3	Sí	Correcto - 3
Precondiciones	Sí	Excelente - 4	Sí	Parc. correcto - 2
Postcondiciones	Sí	Parc. correcto - 2	Sí	Parc. correcto - 2
Escenario principal	Sí	Parc. correcto - 2	Sí	Parc. correcto - 2
Escenarios alternativos	Sí	Excelente - 4	Sí	Excelente - 4

Cuadro 56: Corrección de la especificación del caso de uso **Realizar reserva de actividad turística** respecto al modelo de referencia.

Fuente: elaboración propia.

Elemento CU	Presente: Llama	Corrección: Llama	Presente: Mistral	Corrección: Mistral
Título	Sí	Correcto - 3	Sí	Correcto - 3
Descripción	Sí	Correcto - 3	Sí	Correcto - 3
Precondiciones	Sí	Correcto - 3	Sí	Parc. correcto - 2
Postcondiciones	Sí	Parc. correcto - 2	Sí	Parc. correcto - 2
Escenario principal	Sí	Correcto - 3	Sí	Correcto - 3
Escenarios alternativos	Sí	Correcto - 3	Sí	Excelente - 4

Cuadro 57: Corrección de la especificación del caso de uso **Pago de reserva de actividad turística** respecto al modelo de referencia.

Fuente: elaboración propia.

Cuadro 58: Corrección diagrama **Registro de nuevo usuario**, Llama 3.1.

Elementos	Corrección
Actor principal	Correcto - 3
Clases de Entidad	Parc. correcto - 2
Clases de Control	Correcto - 3
Clases de Interfaz	Parc. correcto - 2
Sec. principal 1: Clic registro	Parc. correcto - 2
Sec. principal 2: Mostrar formulario	Correcto - 3
Sec. principal 3: Introducir datos	Correcto - 3
Sec. principal 4: Clic crear	Correcto - 3
Sec. principal 5: Validar datos	Correcto - 3
Sec. principal 6a: Crear cuenta	Correcto - 3
Sec. principal 6b: Error datos	Correcto - 3
Sec. principal 7: Redirigir login	Correcto - 3
Sec. alternativa 1: Usuario existente	Correcto - 3
Sec. alternativa 2: Formato inválido	Correcto - 3

Cuadro 59: Corrección diagrama **Registro de nuevo usuario**, Mistral

Elementos	Corrección
Actor principal	Correcto - 3
Clases de Entidad	Parc. correcto - 2
Clases de Control	Correcto - 3
Clases de Interfaz	Correcto - 3
Sec. principal 1: Navegar web	Correcto - 3
Sec. principal 2: Mostrar inicio	Correcto - 3
Sec. principal 3: Registrarse	Correcto - 3
Sec. principal 4: Evento registrarse	Correcto - 3
Sec. principal 5: Mostrar formulario	Correcto - 3
Sec. principal 6: Rellenar campos	Correcto - 3
Sec. principal 7: Crear cuenta	Correcto - 3
Sec. principal 8: Evento crear cuenta	Correcto - 3
Sec. principal 9: Validar datos	Correcto - 3
Sec. principal 10: Respuesta validación	Correcto - 3
Sec. principal 11: Crear cuenta válida	Correcto - 3
Sec. alternativa 12: Usuario existente	Correcto - 3
Sec. alternativa 13: Formato inválido	Correcto - 3

Se presenta un resumen de la corrección de los diagramas de secuencia en la siguiente tabla:

Diagrama de secuencia	Incorrecto	Parc. correcto	Correcto	Excelente	Total
Registro de nuevo usuario	0	3	11	0	14
Gestión de empleados	0	1	15	2	18
Reserva de actividad	2	4	8	1	15
Pago de reserva	6	4	2	0	12

Cuadro 60: Resumen y recuento total de la corrección de diagramas de secuencia generados por Llama 3.1. Fuente: elaboración propia.

Diagrama de secuencia	Incorrecto	Parc. correcto	Correcto	Excelente	Total
Registro de nuevo usuario	0	1	16	0	17
Gestión de empleados	1	1	19	0	21
Reserva de actividad	2	2	23	0	27
Pago de reserva	2	2	9	1	14

Cuadro 61: Resumen y recuento total de la corrección de diagramas de secuencia generados por Mistral. Fuente: elaboración propia.

Finalmente, se presenta la evaluación de la corrección de los **formatos de pantalla** de cada LLM con un ejemplo de la primera pantalla y, por último, un resumen total de la distribución de categorías por cada formato de pantalla, donde cada columna intermedia hace referencia al recuento de la categoría correspondiente:

Cuadro 62: Corrección formato de pantalla de **Registro de nuevo usuario**, Llama 3.1.

Elementos	Corrección
Nombre de usuario	Correcto - 3
Correo electrónico	Correcto - 3
Número de teléfono	Correcto - 3
Contraseña	Correcto - 3
Repetir contraseña	Excelente - 4
DNI	Correcto - 3
Botón 'Crear cuenta'	Correcto - 3
Botón 'Ir a inicio de sesión'	Correcto - 3

Resumen Maquetas	I	P.C	C	E	Total
Registro de nuevo usuario	0	0	7	1	8
Gestión de empleados	3	5	16	1	25
Reserva de actividad	1	2	10	0	13
Pago de reserva	0	0	10	0	10

Cuadro 63: Corrección formato de pantalla de **Registro de nuevo usuario**, Mistral

Elementos	Corrección
Name	Correcto - 3
Surname	Correcto - 3
Username	Correcto - 3
Email	Correcto - 3
Phone number	Correcto - 3
Password	Correcto - 3
DNI	Correcto - 3
Create Account	Correcto - 3
Login Now	Correcto - 3

Resumen Maquetas	I	P.C	C	E	Total
Registro de nuevo usuario	0	0	9	0	9
Gestión de empleados	3	0	27	5	35
Reserva de actividad	0	0	19	0	19
Pago de reserva	0	0	11	2	13

### 3.8.3. Rendimiento

En este apartado se muestra la métrica de rendimiento que recoge los tiempos dedicados en cada fase a cada una de las distintas tareas de manera general.

FASE 1			
Actividad	Tiempo	Tiempo Llama	Tiempo Mistral
<b>Catálogo de requisitos</b>	109 min 49 s	21.73 s	29.08 s
<b>Casos de uso (incluida especificación)</b>	61 min 2 s	50.27 s	47.74 s
FASE 2			
Actividad	Tiempo	Tiempo Llama	Tiempo Mistral
<b>Modelo de clases (clase, control e interfaz)</b>	66 minutos	X	X
<b>Diagramas de secuencia</b>	129 min 57 s	40.75 segundos	34.21 segundos
FASE 3			
Actividad	Tiempo	Tiempo Llama	Tiempo Mistral
<b>Diseño de interfaces de usuario</b>	252 min	73 segundos	40.06 segundos
<b>Modelo físico de datos</b>	71 min 2 s	X	X

En la tabla se muestran las actividades de cada fase y los tiempos que se ha tardado en realizar cada una. Las columnas de tiempo hacen referencia al tiempo manual en ejecutar una tarea y tiempo de cada LLM en generar la salida. Aquellas celdas marcadas con una 'X' no han sido realizadas por los modelos. La razón de esto se ha explicado en la sección correspondiente.

El modelo físico de datos se ha decidido realizar en la siguiente fase de implementación del software, concretamente en el diseño de la base de datos. Asimismo, el modelado de clases de análisis se ha obtenido en la especificación de casos de uso generada por los LLMs. Para ampliar esta información, leer el apéndice [A](#).

### 3.8.4. Evaluación de resultados

En este apartado se recoge de manera visual y agrupada un reporte de las métricas aplicadas para facilitar la visualización de los resultados. Comenzando con la **completitud**, se han obtenido los siguientes resultados:

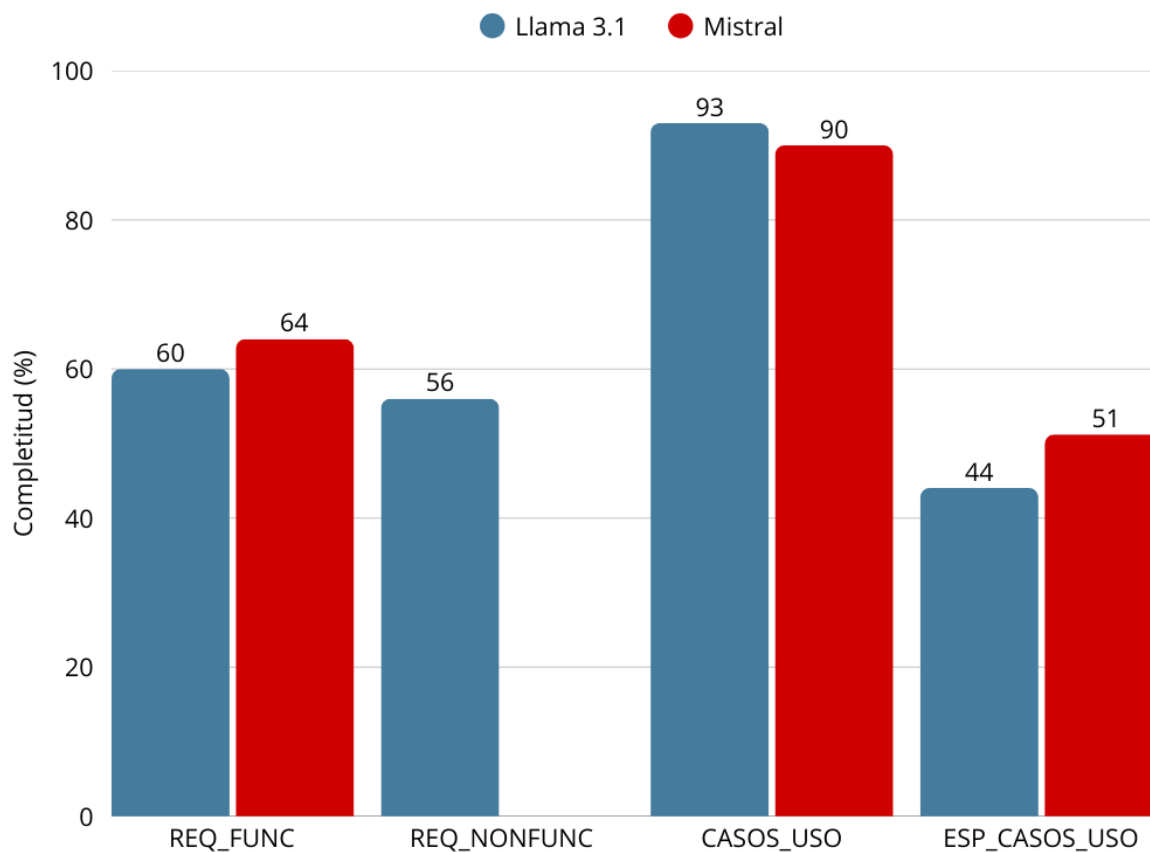


Figura 17: Comparación visual de resultados en la completitud de requisitos, casos de uso y su especificación respecto al desarrollo manual. Fuente: elaboración propia.

Estos últimos resultados hacen referencia a la **primera fase** del proceso de desarrollo software.

Continuando con la **segunda fase** en materia de completitud:

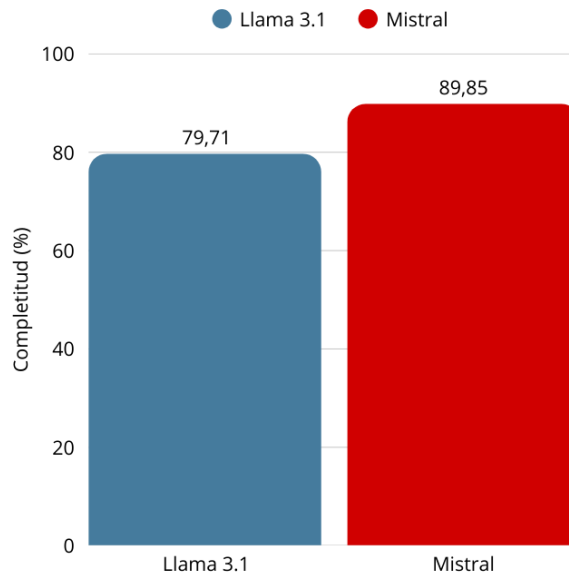


Figura 18: Comparación visual de resultados en la completitud de diagramas de secuencia respecto al desarrollo manual. Fuente: elaboración propia.

Finalizando con la **tercera fase** en materia de completitud, los resultados de las **maquetas de pantalla**:

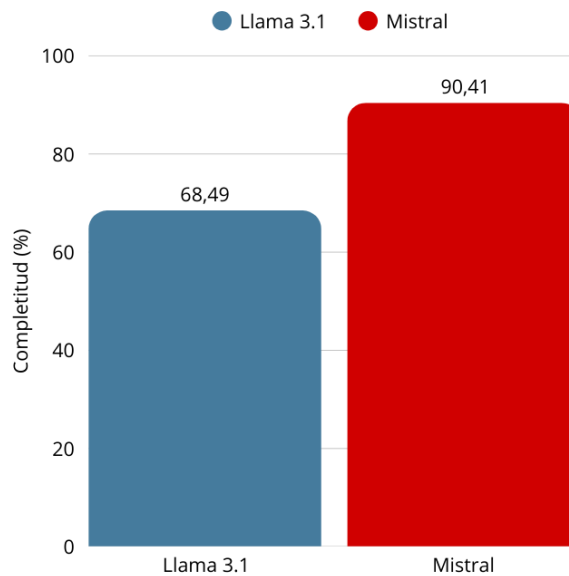


Figura 19: Comparación visual de resultados en la completitud de maquetas de pantalla respecto al desarrollo manual. Fuente: elaboración propia.

A partir de la **corrección** aplicada a la identificación de catálogo de **requisitos funcionales** se ha obtenido lo siguiente:

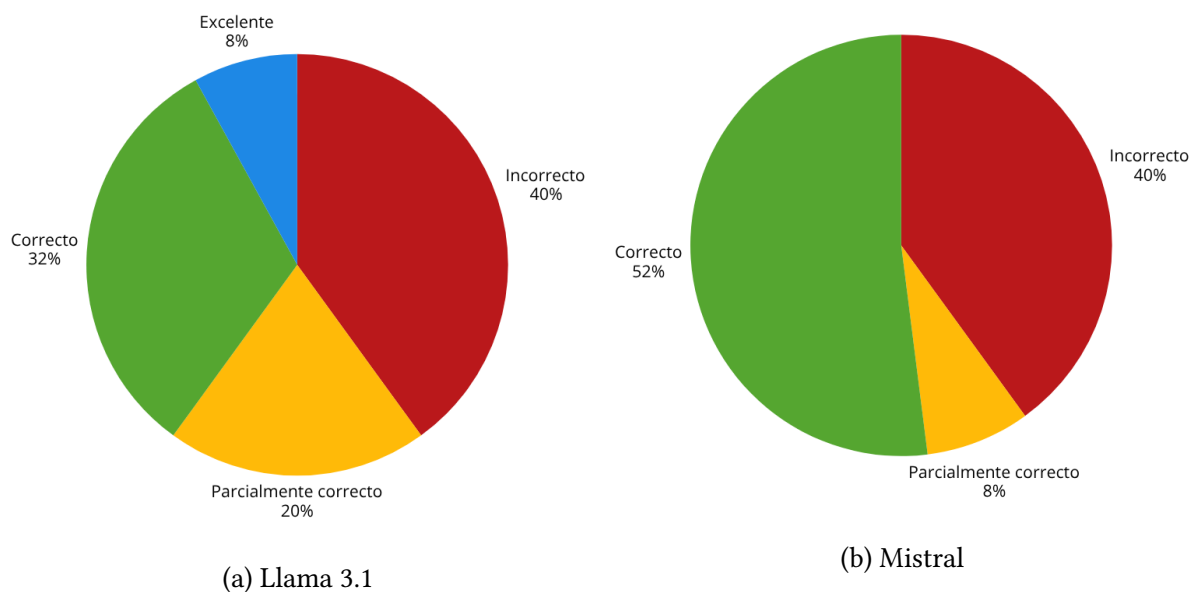


Figura 20: Distribución de categorías de corrección en el análisis de requisitos funcionales obtenidos por cada modelo. Fuente: elaboración propia.

En el caso de los **requisitos no funcionales**, el modelo Mistral no lo ha hecho correctamente por lo que solo se muestra la información de Llama:

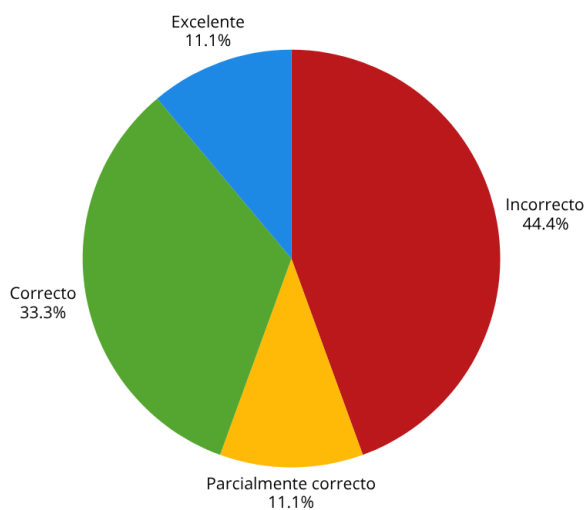


Figura 21: Comparación visual de completitud de requisitos no funcionales por cada modelo. Fuente: elaboración propia.

La **corrección** aplicada a los **actores y casos de uso** generados a partir del uso de los LLMs muestra los siguientes resultados. Asimismo, también se muestran los resultados de la **especificación de casos de uso**:

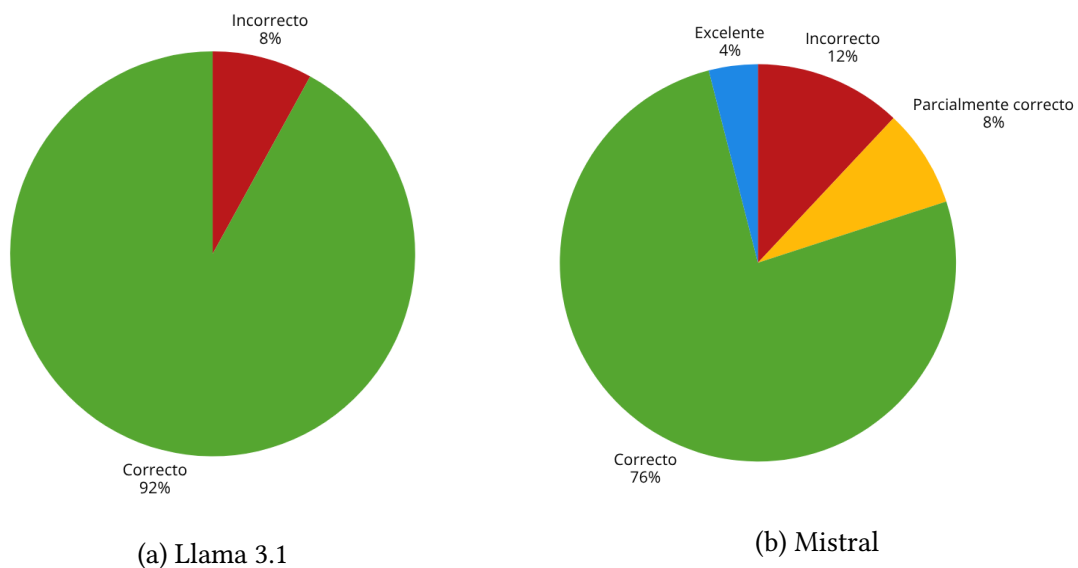


Figura 22: Distribución de categorías de corrección de casos de uso y actores obtenidos por cada modelo. Fuente: elaboración propia.

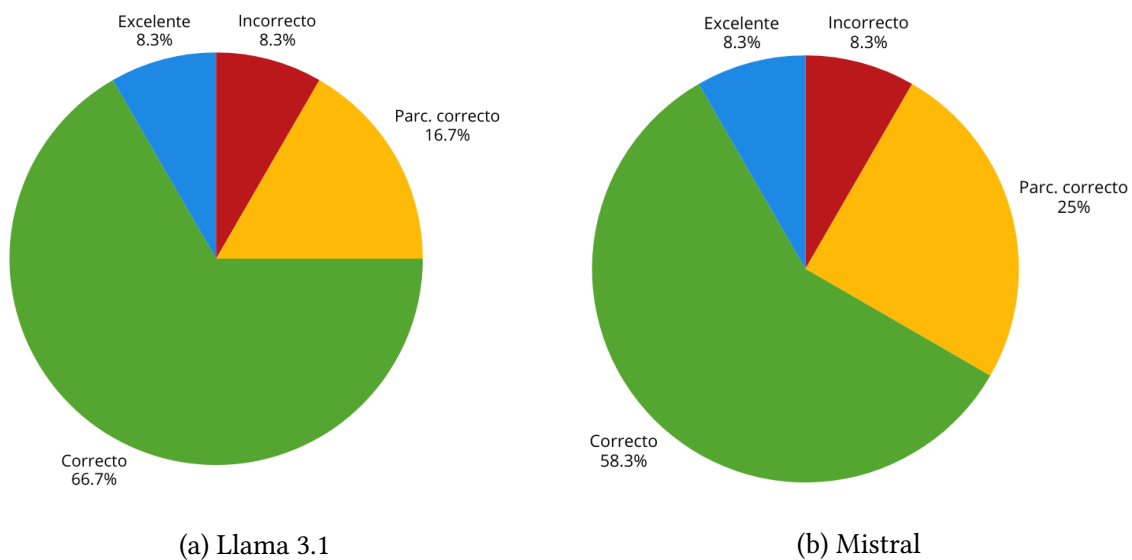


Figura 23: Distribución de categorías de corrección de especificación de casos de uso obtenidos por cada modelo. Fuente: elaboración propia.

Para la **especificación de casos de uso** se han asignado las categorías en función del contenido de cada elemento. Para mostrar el resultado de manera visual, el total de elementos por modelo se ha fijado a 24, siendo este el recuento total de elementos por cada LLM, es decir, no se ha tenido en cuenta todo el contenido generado para mostrar los resultados de manera visual si no que se ha tenido en cuenta únicamente la categoría en cada elemento i.e. categoría asignada a los títulos, a los escenarios alternativos, etcétera. Por ello, los resultados tras aplicar las categorías de corrección a los principales elementos de la especificación de cada caso de uso son las mostradas en la figura 23.

La corrección de los **diagramas de secuencia** en formato textual generado por los LLMs muestra el siguiente resultado:

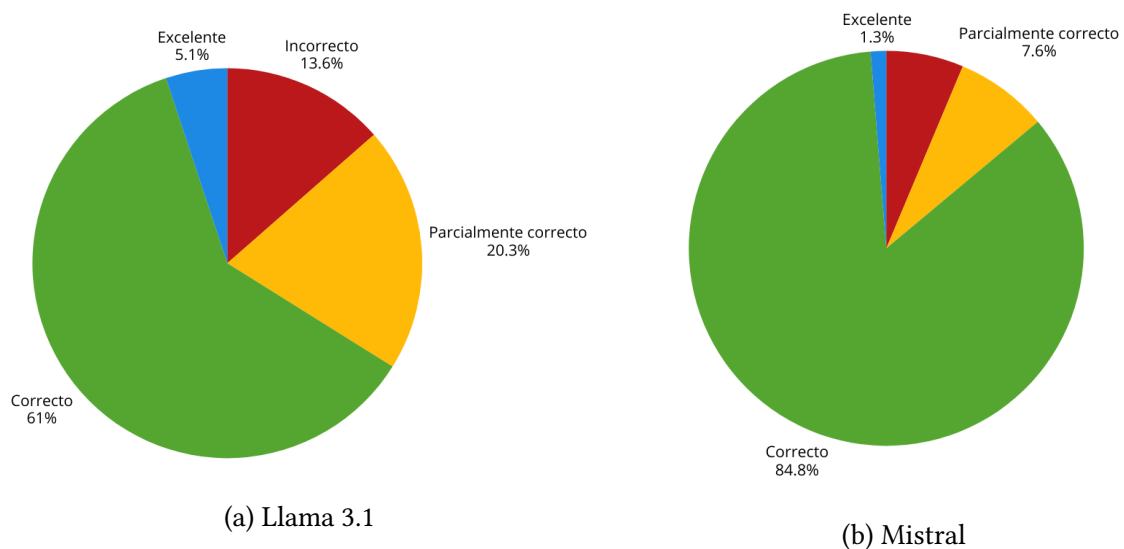


Figura 24: Distribución de categorías de corrección de diagramas de secuencia obtenidos por cada modelo. Fuente: elaboración propia.

Finalmente, la corrección de la última fase de este capítulo finaliza con los resultados de las **maquetas de pantalla**:

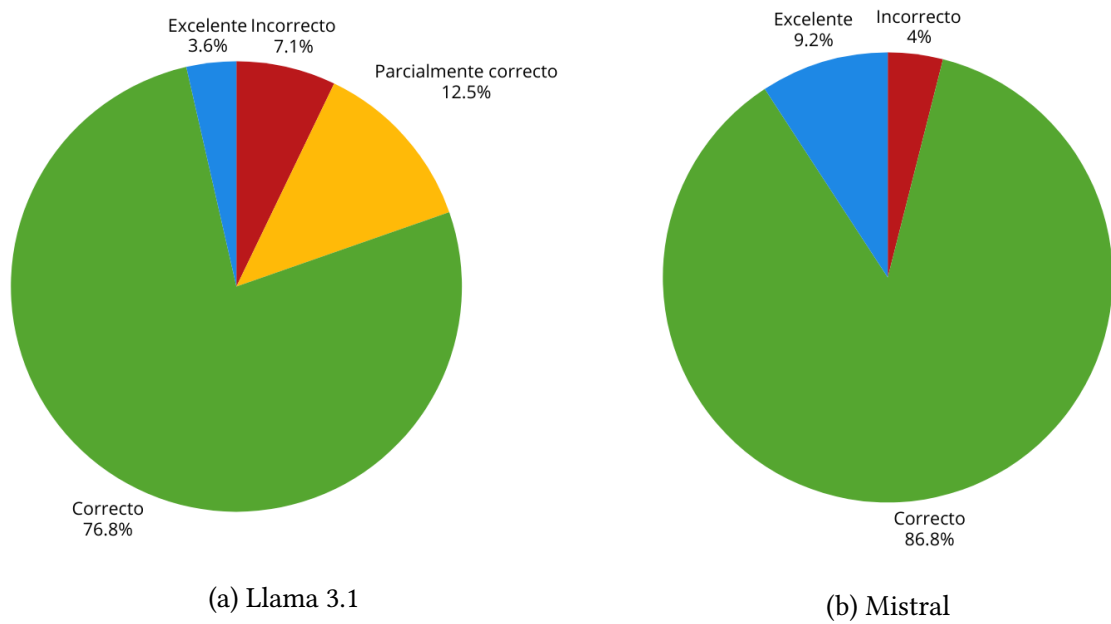


Figura 25: Distribución de categorías de corrección de formatos de pantalla obtenidos por cada modelo. Fuente: elaboración propia.

# 4

## Implementación

Este capítulo recopila apartados relevantes a la implementación del sistema, desde la arquitectura software, pasando por el diseño e implementación de bases de datos, hasta la realización del código para desarrollar las funcionalidades necesarias, priorizando las relacionadas a procesos de negocios más relevantes, ya comentados en los capítulos anteriores. Por último, se hace uso de los LLMs seleccionados para este proyecto para generar soluciones con las que realizar una evaluación de su empleo en esta fase, mostrando finalmente un resumen de las métricas que se han aplicado y qué se ha tenido en cuenta para cada una de las tareas para la aplicación de las métricas.

### 4.1. Arquitectura software

Esta sección recoge las diferentes partes en las que se divide el sistema desde un contexto de arquitectura del software. En esta sección se muestran las jerarquías de organización en cada parte, en concreto del *frontend* y *backend* de la aplicación web del caso práctico planteado. Además, se muestra finalmente cómo se realiza la comunicación entre distintas partes, incluyendo la base de datos.

Asimismo, se muestra el enfoque y arquitectura utilizado para el proyecto, es decir, se ha optado por utilizar un patrón de arquitectura **Modelo-Vista-Controlador**. También se detalla la implementación y uso de una API REST<sup>10</sup>. Asimismo, la arquitectura se distingue en tres capas:

- **Capa de presentación**
- **Capa de negocio**
- **Capa de datos**

---

<sup>10</sup> *Representational State Transfer*

La arquitectura de comunicación entre el cliente y el servidor en el sistema propuesto en el caso práctico se basa en una API REST que interactúa con una base de datos relacional. En las siguientes secciones se muestran de manera detallada las distintas capas y las principales partes de cada una en el desarrollo manual realizado.

En primer lugar, el cliente está implementado como una SPA<sup>11</sup> en Angular, que constituye la interfaz de usuario. Desde esta aplicación, el usuario puede realizar operaciones como registrarse, realizar *login* con sus credenciales de acceso, búsqueda de actividades, consulta de disponibilidad y realizar reservas y pagos. Cuando se requiere acceder a datos o ejecutar una acción en el servidor, Angular envía peticiones HTTP al backend. Estas solicitudes incluyen, en el encabezado Authorization de las peticiones mediante un token JSON Web Token que identifica al usuario y permite controlar su nivel de acceso según su rol. Asimismo, se ha configurado el mecanismo CORS<sup>12</sup> en el backend para permitir únicamente las solicitudes procedentes de orígenes autorizados, en este caso el dominio donde se aloja el frontend. El intercambio de información se realiza en formato JSON, que ofrece una estructura ligera y ampliamente compatible con aplicaciones web modernas.

En el lado del servidor, la API REST está desarrollada con NestJS. Al recibir una petición, la API aplica una serie de Guards que validan el JSON Web Token y verifican los permisos de acceso al recurso a través del rol del usuario. Posteriormente, la información de entrada pasa por Pipes y DTOs<sup>13</sup> en los controladores de los endpoints establecidos para la API en el servidor, los cuales se encargan de validar y transformar los datos antes de que lleguen a la capa de lógica de negocio. Dicha lógica se implementa en servicios, que contienen las instrucciones necesarias para la ejecución de las operaciones de los procesos de negocio y finalmente, acceder a la base de datos. Para acceder a la base de datos, se emplea Prisma ORM, que traduce los modelos y operaciones definidas en el backend a consultas SQL tipadas y seguras. La persistencia se realiza en MariaDB, donde se almacenan las entidades principales del dominio y las restricciones mediante claves primarias, foráneas y tipo CHECK.

Finalmente, la respuesta generada por la API se envía de vuelta al cliente en formato JSON a través de HTTPS. Esta respuesta puede contener los datos solicitados o información de error en caso de que la operación no pueda completarse. Una vez recibida, la aplicación Angular

---

<sup>11</sup> *Single Page Application*

<sup>12</sup> *Cross-Origin Resource Sharing*

<sup>13</sup> *Data Transfer Objects*

actualiza la interfaz de usuario para reflejar el resultado.

Un flujo típico y general (realizado en Mermaid.live[31]) de todo este proceso es el que se muestra en la siguiente imagen:

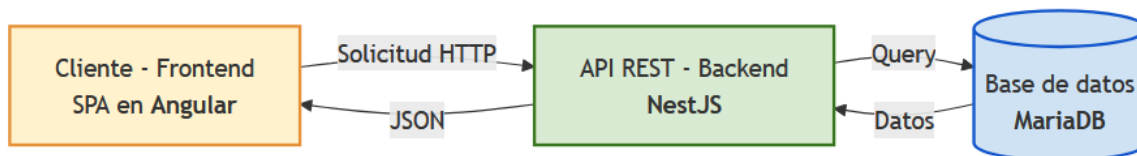


Figura 26: Flujo general de proceso típico del sistema. Fuente: elaboración propia.

En la figura 26 se muestra el flujo básico y general del paso de mensajes que se sigue dentro del sistema. Desde la capa de presentación (*frontend*), se realiza una solicitud HTTP a los *endpoints* de la capa de negocio y, esta capa aplica una serie de lógica y reglas que finalmente interviene con llamadas a la capa de datos, obteniendo la respuesta. De manera general, la representación de este flujo es válido, pero se ha realizado una representación más completa del sistema, que sirve para entender el paso de mensajes, acceso a recursos y respuestas en cada capa.

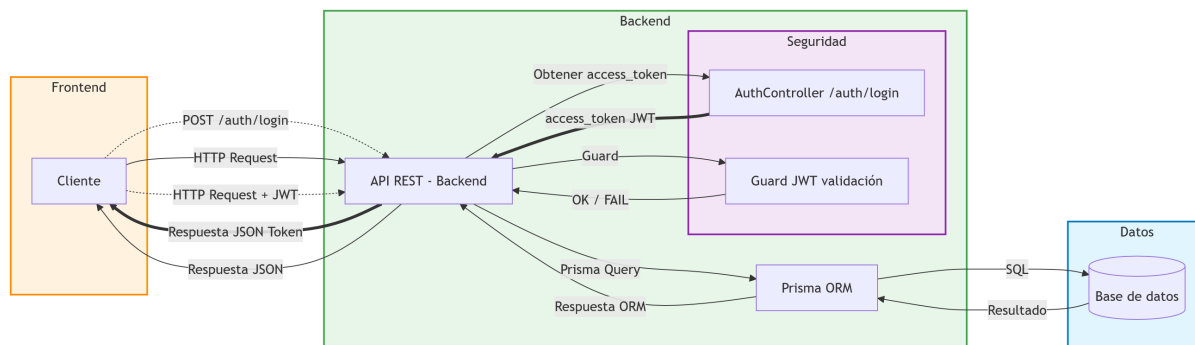


Figura 27: Flujo completo del sistema realizado con Mermaid.live[31]. Fuente: elaboración propia.

El diagrama de la figura 27 muestra la arquitectura lógica de la aplicación, donde intervienen el **cliente (frontend)**, el **servidor (backend con la API REST)** y la **base de datos**. El **frontend** representa la interfaz de usuario desde la que se envían peticiones HTTP, que pueden ser públicas o incluir un **token JWT** para acceder a recursos protegidos a través del tipo de usuario que tengan asignado en el sistema. El **backend** expone los *endpoints* de la API

REST y contiene un módulo de seguridad. Este módulo gestiona la autenticación mediante el endpoint `/auth/login`, que genera un `access_token JWT`, y valida los tokens a través de un **Guard** antes de permitir el acceso a las rutas restringidas. Para el acceso a los datos, la API se comunica con **Prisma ORM**, que transforma las consultas en instrucciones SQL y devuelve los resultados como objetos al servidor. Finalmente, la **base de datos** almacena la información y responde a las consultas enviadas por Prisma. En cuanto a los flujos principales:

- En el **login**, el usuario envía sus credenciales y recibe un `access_token JWT` en caso de ser válidas.
- En las **peticiones HTTP**, el *Guard* valida para ciertas peticiones el token antes de procesar la solicitud.
- En el acceso a datos, Prisma envía la consulta SQL, obtiene la respuesta y la API la devuelve al cliente en formato **JSON**.

#### 4.1.1. Frontend

En el **frontend**, es decir, capa de presentación, se ha utilizado **Angular** como *framework* principal, lo que ha permitido desarrollar una interfaz modular, escalable y dinámica. La aplicación se estructura en torno a un conjunto de **componentes reutilizables**, cada uno de los cuales encapsula la lógica y la presentación de una vista concreta. Estos componentes se comunican de manera eficiente mediante **servicios inyectables**.

El sistema de *routing* interno de Angular permite una navegación fluida entre las diferentes secciones de la aplicación, gestionando tanto rutas estáticas como rutas parametrizadas, y asegurando una experiencia de usuario coherente en la transición entre pantallas. Para la captura y gestión de datos, se emplean **formularios reactivos**, que proporcionan un control total sobre la validación y el modelado de la información antes de ser enviada al *backend*, integrando validaciones.

En lo que respecta a la comunicación con la capa de servidor, se hace uso de **HttpClient**, módulo nativo de Angular que facilita la interacción con la API expuesta en el *backend*, gestionando peticiones y respuestas bajo los estándares HTTP y asegurando un tratamiento estructurado de los datos recibidos en formato JSON. Esta integración permite el manejo de cabeceras

ras, interceptores y observables, lo que incrementa la robustez y seguridad en el intercambio de información entre cliente y servidor.

Finalmente, el uso de *pipes* personalizados y directivas de Angular contribuye a la transformación de datos y a la mejora de la experiencia de usuario, permitiendo formatear información, aplicar filtros dinámicos y extender el comportamiento de los elementos de la interfaz.

#### 4.1.2. Backend

El *backend* se entiende como la capa de negocio de la aplicación. Para la realización del *backend* del caso práctico se optó por utilizar NestJS debido a su arquitectura modular y escalable, soporte nativo para TypeScript, compatibilidad con Express y Fastify, *frameworks* de Node.js y por su similitud con Angular, el cual también se vio que se decidió utilizar para este proyecto. Asimismo, NestJS incorpora validación de datos mediante *Validation Pipes* e integra seguridad y autenticación mediante *Guards*.

Para el backend se ha hecho uso de distintos elementos. Por un lado se han utilizado **controladores** los cuales son los encargados de recibir solicitudes HTTP del cliente, es decir, actúan como puerta de entrada a la aplicación. Estos controladores se encargan de atender las peticiones de entrada y redirigirlas al servicio correspondiente, devolviendo cada respuesta. Sin embargo, antes de que un controlador ejecute su lógica, intervienen los **Guards**, que se encargan de decidir si la petición puede continuar o debe ser rechazada. En este punto se integra **PassportStrategy**, que facilita la autenticación mediante JWT u otros proveedores externos, aunque esto último no entra en el alcance del caso práctico. Cuando la petición contiene datos de entrada, entran en juego los DTOs y los **Validation Pipes**. Los DTOs establecen la forma que deben tener los datos, mientras que los pipes aplican las reglas de validación definidas antes de que lleguen al controlador. De esta forma se asegura que la aplicación solo procese información correcta y fiable, reduciendo así posibles errores y protegiendo la API frente a entradas maliciosas. Una vez validados los datos, el controlador delega la lógica de negocio a los **servicios**. Finalmente, cuando es necesario acceder a la base de datos, los servicios utilizan **Prisma ORM**.

Con el proceso explicado, en la siguiente figura se muestra el flujo típico que sigue una solicitud HTTP en el *backend*.

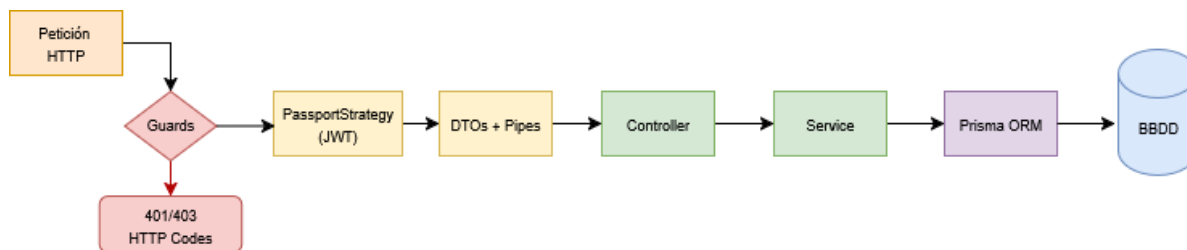


Figura 28: Flujo típico de procesamiento de solicitud HTTP por parte de un cliente hacia el sistema realizado en draw.io [32]. Fuente: elaboración propia.

Como aclaración, para el desarrollo del caso práctico, la mayoría de *endpoints* de la API no requieren de autenticación, es decir, token JWT con el que se compruebe su rol para acceder a los recursos, pero se ha decidido incluir en el flujo para mostrar el procedimiento completo.

#### 4.1.3. Base de datos

Finalmente, esta es la capa de datos de la aplicación web. En esta sección se ha recogido el diseño e implementación de base de datos a partir del caso práctico (sección 3.1.2).

Como se introdujo en el primer capítulo de esta memoria del proyecto (sección 1.3.2), se ha utilizado MariaDB como SGBD<sup>14</sup> por ser una solución robusta y de código abierto, con un amplio soporte comunitario. Para la ejecución de consultas SQL se ha utilizado HeidiSQL, software que permite administrar bases de datos de forma sencilla y con una interfaz intuitiva.

El diagrama entidad-relación ofrece una representación clara del modelo físico de datos desarrollado en la sección 3.4, presentándolo desde una perspectiva más familiar y convencional en el ámbito de las bases de datos. Por ello, tras realizar las sentencias SQL necesarias se ha obtenido el siguiente modelo, respetando las restricciones de claves, atributos y cardinalidad en las relaciones:

<sup>14</sup>Sistema Gestor de Bases de Datos

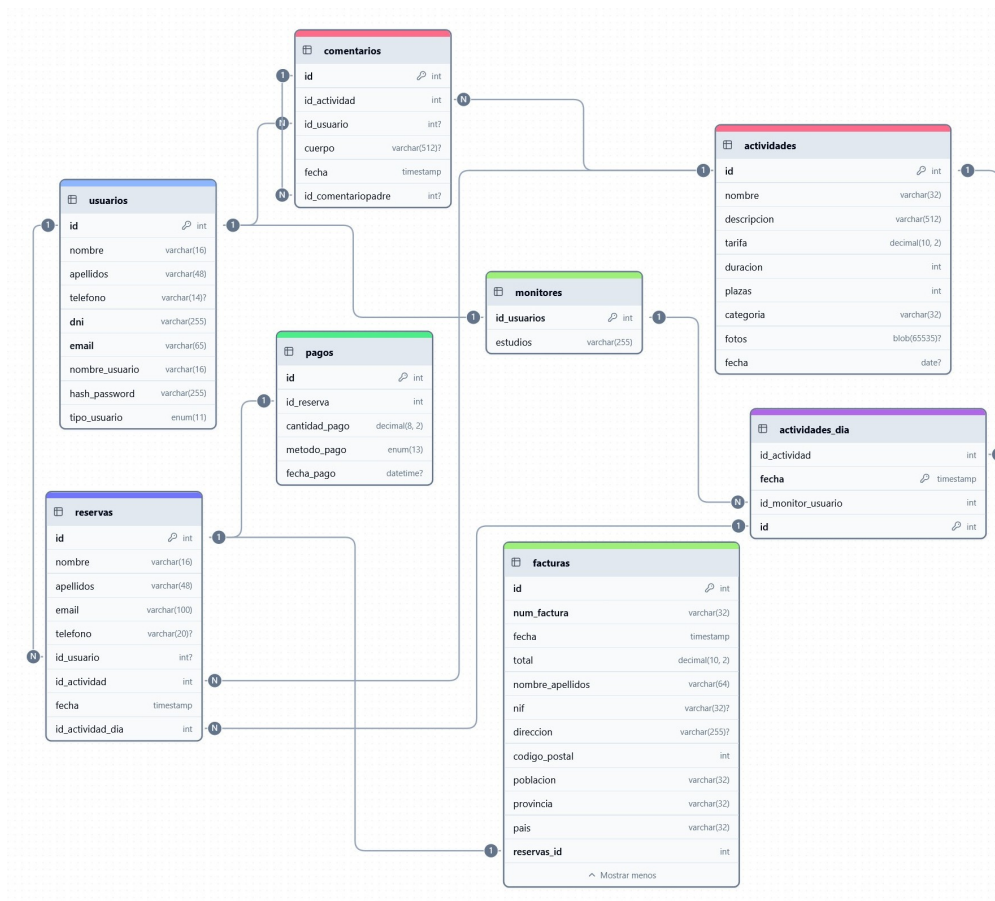


Figura 29: Diagrama E/R de la base de datos. Fuente: elaboración propia

Debido a que HeidiSQL no tiene la capacidad para generar un diagrama entidad-relación, se ha utilizado la plataforma web ChartDB [33].

El diagrama presentado en la figura 29 presenta las siguientes entidades y relaciones:

- **Usuarios:** Esta entidad representa todos los posibles tipos de usuarios registrados en el sistema. Todos los usuarios, excepto monitores, comparten la misma información y se diferencian a partir de un rol entre los que se distinguen responsable, empleado, monitor y cliente. Los usuarios guardan relación directa con las reservas que realicen y los comentarios que añaden en actividades. Un usuario puede realizar cero o múltiples reservas y añadir uno o más comentarios.
- **Monitores:** Los monitores son un tipo de usuario del cual se debe almacenar su cualificación profesional para tareas de asignación de actividades en las que es necesario comprobar si está cualificado para impartir dicha actividad o no. Es una entidad débil

que hereda su clave primaria de usuarios. Por ello, un identificador de monitor solo se puede asociar a un usuario.

- **Actividades:** Las actividades que ofrece la empresa ficticia se almacenan en esta tabla. A partir de esta tabla se asocian todos los comentarios que hayan en ella, pudiendo haber cero. Asimismo, mediante una relación M:N con monitores aparece la tabla puente de actividades por día en la que asocia todas las actividades que puede realizar un monitor en un mismo día.
- **Actividades\_Dia:** Esta tabla surge a partir de la relación M:N de Actividades con Monitores. Almacena la información en la que cada día un monitor realiza cierta actividad, las cuales están asociadas a reservas. La tabla presenta un identificador simple para reducir la complejidad de una clave compuesta.
- **Reservas:** Esta tabla almacena todas las reservas de actividades que se realicen en la aplicación web. Cada reserva tiene asociada una única factura y al menos un pago. Asimismo, un pago se asocia a una reserva. Muchas reservas de una actividad están asociadas a un día.
- **Facturas:** En esta tabla se almacenan toda la información asociada al cliente, registrado o no sobre las reservas de actividades turísticas realizadas.
- **Pagos:** La tabla pagos almacena información relevante al pago, pero no sobre el pago, es decir, solo almacena datos importantes de cada reserva como la cantidad, método de pago y fecha en la que se realizó. De esta forma se mantiene un histórico de datos útiles.
- **Comentarios:** Los comentarios asociados a cada actividad, agregados por los usuarios registrados se almacenan en esta tabla. Un usuario puede añadir o no comentarios por actividad y dicha actividad puede no tener comentarios.

## 4.2. Implementación software

En esta sección se recopila los aspectos más importantes y detallados de la implementación software desarrollada de forma manual. Se explica con mayor profundidad la implementación de los procesos de negocios más relevantes, útiles para la evaluación de los LLMs respecto a las tareas de esta fase. Además, contiene tanto partes de la capa de presentación, como de la capa de negocio.

### 4.2.1. API REST

Para el desarrollo de la API REST se han seguido buenas prácticas de de diseño de API para aplicaciones web, de acuerdo a la guía hecha por Microsoft [34].

Para el desarrollo del caso práctico, se optó por realizar una API de tipo REST por su simplicidad y eficiencia, además de ser la elección más común entre desarrollos de este mismo tipo.

La API REST implementada actúa como una capa de acceso entre el cliente y el servidor. Asimismo, la API desarrollada también integra *endpoints* de PayPal (en entorno *sandbox*) para la creación y validación de pagos. En la figura 27, se muestra un flujo completo entre el cliente y el servidor, el cual también contiene aspectos como la validación de *tokens JWT*. Sin embargo, en este diagrama, por motivos de diseño, no se ha decidido mostrar la integración con PayPal *sandbox*.

La API ha sido implementada con NestJS, utilizando TypeScript. La estructura principal recae en controladores y servicios y el acceso a datos se realiza mediante Prisma ORM.

En la siguiente tabla 64 se muestran los *endpoints* implementados en el *backend*, agrupados por módulos (Actividades, Registro, Empleados, Reservas, Pagos). Para mayor claridad, se resumen las operaciones principales, indicando el método HTTP, el endpoint y su función.

Módulo	Método	Endpoint	Descripción
Actividades	GET	/actividad/{id}	Obtener una actividad específica
Actividades	GET	/actividades	Listar todas las actividades
Auth	POST	/auth/registro	Registro de un nuevo cliente
Auth	POST	/auth/login	Inicio de sesión (cualquier usuario)
Empleados	GET	/empleados/{id}	Obtener datos de un empleado
Empleados	GET	/empleados	Obtener todos los empleados
Empleados	POST	/empleados/nuevo	Crear nuevo empleado
Empleados	PATCH	/empleados/{id}	Actualizar datos de un empleado
Empleados	DELETE	/empleados/{id}	Eliminar un empleado
PayPal	POST	/paypal/token	Crear token de acceso para PayPal
PayPal	POST	/paypal/orders	Crear orden de pago para PayPal
PayPal	POST	/paypal/orders/capture	Capturar orden de pago para PayPal
Reservas	POST	/reserva/checkout	Realizar una reserva en estado pendiente de pago
Pagos	POST	/pagos/orders	Registrar orden de pago de reserva mediante PayPal
Pagos	POST	/pagos/orders/capture	Capturar orden de pago de reserva mediante PayPal

Cuadro 64: Resumen de *endpoints* implementados en el *backend*. Fuente: elaboración propia.

#### 4.2.2. Seguridad

La mayor parte de seguridad implementada en el caso práctico se basa en **autenticación** mediante tokens JWT. Estos tokens incluyen en su *payload* el identificador del usuario, el nombre de usuario y el tipo de usuario, y se transmiten en cada petición protegida a través de la cabecera HTTP **Authorization: Bearer <access\_token>**. La duración de este token en el sistema es de 25 minutos. El campo *tipo de usuario* es imprescindible, ya que permite aplicar una estrategia de autorización mediante **Role-Based Access Control** (RBAC), restringiendo el acceso a ciertos recursos según el rol.

El *access\_token* necesario para acceder a ciertos recursos protegidos, como la gestión de empleados, se genera tras registrar un nuevo usuario cliente o después de iniciar sesión. El formato de un token consiste de tres partes:

- **Header:** Una cabecera que normalmente consiste en dos partes, el tipo del token, JWT en este caso, y el algoritmo de firma utilizado.
- **Payload:** Contiene la información necesaria para la transmisión, en el caso práctico hace

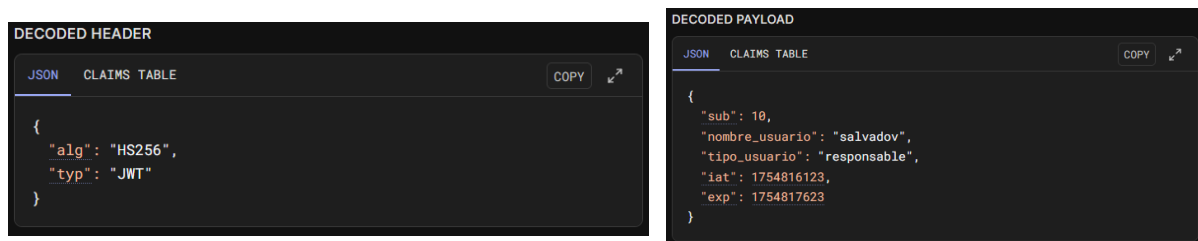
referencia a los tres elementos mencionados anteriormente; identificador de usuario, nombre de usuario y tipo de usuario.

- **Signature:** Firma digital generada que con una clave secreta para asegurar la identidad del emisor del mensaje, manteniendo integridad en los datos enviados en el mensaje.

Todo esto se codifica en un bloque en formato **Base64**.

Cuando un usuario cliente o un usuario con una cuenta registrada en el sistema inicia sesión, el servidor genera un JWT con cierto tiempo de expiración, y lo envía al cliente, el cual el servidor utilizará para verificar los privilegios e identidad de un usuario a la hora de acceder a ciertos recursos protegidos. Por último, la mayor parte de *endpoints* de la API REST no están protegidos debido a los requisitos del sistema, solo se protegen ciertos recursos, como la gestión de empleados.

Por ejemplo, obtenido un token de acceso mediante el login de un usuario y su información decodificada es la siguiente:



(a) Header

(b) Payload

Figura 30: Ejemplo de token de acceso JWT decodificado. Fuente: elaboración propia.

Por último, para garantizar la seguridad de las credenciales, las contraseñas se almacenan en la base de datos utilizando el **algoritmo de hashing**, **Argon2**, en su variante, *Argon2id*. Se ha hecho uso del paquete oficial de Nodejs, *node-argon2* [35], para utilizar dos funciones principales; *hash* y *verify*.

#### 4.2.3. Procesos de negocio implementados

Este apartado está dedicado a recoger una breve muestra de los procesos de negocio más relevantes desarrollados para el caso práctico y posterior evaluación de LLMs. Esta parte del capítulo se orienta a la implementación software de dichos procesos por lo que a diferencia del

anterior capítulo en el que se mostraba todo el desarrollo en el cuerpo principal de la memoria o en el apéndice B, se va a mostrar únicamente una descripción de qué hace cada funcionalidad, qué se ha cubierto, cómo se ha logrado y qué devuelve. Adicionalmente, en algunos de estos procesos, se mostrará código relevante. La mayor parte de las explicaciones están relacionadas con el flujo mostrado en la figura 27.

### **Registro de nuevos usuarios y Login**

Este proceso se incorpora a través de una interfaz y módulos que incorporan el acceso seguro a usuarios. Para ello, es necesario que los usuarios introduzcan una serie de datos, entre los que se incluyen email, dni y nombre de usuario únicos. Una vez registrados en el sistema, inician sesión en el sistema con las credenciales de acceso introducidas en el proceso de registro. La funcionalidad de registro está pensada para clientes del sistema, aunque la de inicio de sesión es para todos los usuarios ya registrados en el sistema.

Como ya se ha comentado, iniciar sesión proporciona un token de acceso, el cual incluye un campo de tipo de usuario que valida si un usuario tiene privilegios para acceder a ciertos recursos.

Durante el proceso de registro de nuevo usuario, se realizan una serie de validaciones, como email o dnis existentes, longitud de ciertos campos y presencia de campos obligatorios rellenados, entre otros.

El proceso de registro o inicio de sesión es a través de interfaces de usuario implementadas en el *frontend*. Estas interfaces incorporan varios aspectos de la lógica de la aplicación, entre las que se encuentran **formularios reactivos** que permiten incorporar una serie de atributos agrupados, con validadores integrados. Asimismo, estas interfaces también incorporan lógica de enrutamiento a distintas vistas. Una vez rellenado el formulario de registro o inicio de sesión, se almacena el *token* de acceso y se envía la petición al *backend* con los datos correspondientes.

En el *backend*, la petición llega desde la interfaz en un **DTO** el cual realiza una serie de validaciones y en caso de ser los datos correctos, se utiliza **Prisma** para interactuar con la capa de datos. Finalmente, si todo es correcto y no saltan excepciones, el proceso termina cuando se firma el *token* de acceso y se devuelve al cliente, almacenándolo en local.

Aunque no se indique explícitamente, este proceso incorpora más lógica, como acceso a rutas, enrutamiento, gestión de token de acceso, *guards*, *PassportStrategy*, entre otros.

Finalmente, en el apéndice B, sección correspondiente a la implementación software, se muestra capturas del código más relevante a estos procesos, tanto *frontend* como *backend*.

### Gestión de empleados

La gestión de empleados la realiza el responsable de la aplicación web. Esta gestión solo es accesible mediante autenticación con el *tipo de usuario responsable* y a través de una ruta únicamente conocida por dicho responsable, <https://turismollm.es/admin/empleados><sup>15</sup>.

El proceso de gestión de empleado consiste en acceso a la ruta de administración indicada por parte del responsable. En esta ruta, aparecerá una tabla con la lista de empleados, como en la mostrada en el apéndice B, sección *formatos de pantalla*. En esta lista, el responsable puede realizar operaciones CRUD. Para agregar, editar o ver un nuevo empleado, en el *frontend* se ha utilizado **Modals** de Bootstrap. Las operaciones son utilizadas a través de un servicio en la lógica de la interfaz el cual se encarga de mostrar *formularios reactivos*, interactuar con *Modals* y realizar peticiones HTTP al *backend*. Una vez llegada la solicitud, se utilizan *Guards* para comprobar los privilegios. Dichos *guards* son de dos tipos, **RolesGuard** y **JwtGuard**. Estos, establecen el tipo de usuario y token de acceso válidos para acceder a los controladores, es decir, comprueba la autenticación y autorización del usuario para acceder a los recursos. Estas guardias se integran en el comienzo del controlador de dichos *endpoints*. En caso de éxito, el funcionamiento ya es similar a otros *endpoints*, es decir, los *DTO* se encargan de validar los cuerpos de las peticiones y de ser válidos, los servicios se encargan de interactuar con la capa de datos, devolviendo la respuesta hasta el *frontend*.

En el apéndice B, implementación software, se muestra el código de los *Guards* para no tener que acceder al código.

```
@UseGuards(JwtAuthGuard, RolesGuard)
@Roles(usuarios_tipo_usuario.responsable)
@Controller('empleados')
export class EmpleadoController {}
```

Figura 31: Implementación *Guards* en *backend*. Fuente: elaboración propia.

---

<sup>15</sup>Dominio ficticio.

## Reservas y Pagos

Este último proceso recoge las funcionalidades de realizar reserva y pago de dicha reserva. La implementación que se ha realizado sigue un flujo continuo, es decir, cuando se reserva se realizan varias funciones a la vez.

El proceso de reserva y pago es público, no requiere de autenticación ni privilegios en el sistema. Para reservar, es necesario acceder a la interfaz de la lista de actividades y elegir una. Una vez elegida, se abrirá en la misma pestaña una ventana con la información de dicha actividad. Asimismo, aparecerá un botón de *Reservar*.

Cuando se reserva, aparece en la interfaz la información de la reserva y los campos que debe de rellenar el usuario. Una vez completos los campos, se hace clic sobre *Realizar pago*, el cual redirige a una interfaz en la que es necesario rellenar los datos para la factura y, se hace clic en *Pagar con PayPal*. Cuando se hace clic en el pago, en el *backend* se crea la reserva en estado *pendiente* y se crea la orden de pago en PayPal con los datos de la reserva. Dicha creación de orden de pago genera un enlace para que el usuario acceda a la interfaz de pago de PayPal y rellene los datos para aceptar el pago. Hasta que no se acepta el pago, el *backend* no cambia el estado de la reserva, ni almacena los datos del pago ni los de la factura. Una vez completado dicho proceso, se almacena la reserva en estado *confirmada*, los datos del cobro y los de la factura.

En resumen, primero se crea la reserva pendiente, se crea la orden de pago, luego si se realiza el pago, se crea y almacenan los datos de la reserva confirmada, los del pago y la factura.

Para todo el proceso de pago es necesario que persista el ***orderID*** que proporciona PayPal. Todo el proceso de reservas y pagos es similar al resto, desde las interfaces se hacen solicitudes con un *DTO* en el *body* de la petición y, en caso de ser correcto, los controladores y servicios se encargan de la comunicación con la capa de datos para su almacenamiento y gestión.

Esta parte de la implementación involucra varias clases por lo que al no tratarse de uno o dos métodos, no se muestra en el apéndice B.

### 4.3. Uso de LLMs

En esta sección se recogen resultados interesantes tras el uso de LLMs en esta fase, implementación de la base de datos y software. Para visualizar las salidas completas es necesario acceder al apéndice A, en la sección correspondiente a la implementación del software.

Debido a problemas con el compilador de *LaTeX*, se va a adjuntar cada *prompt* relacionado a la implementación software en distintos archivos de la entrega, junto a sus salidas. Por proceso a implementar, es probable que se haya utilizado más de un *prompt*.

Por cada proceso de negocio, se ha dado hasta 3 intentos a cada LLM para corregir su salida en un primer análisis.

Para el uso de LLMs de la tarea de generación de código en esta fase, se ha utilizado distintos *prompts*. No se parte desde cero, se proporciona siempre algo de contexto o código.

Para simplificar la lectura, se ha creado un diagrama de árbol que muestra la estructura de este apartado. Este diagrama se encuentra en la figura 32, en la siguiente página. El objetivo es guiar de manera simple y gráfica por los puntos tratados en la sección *Uso de LLMs* de este capítulo. La estructura se divide en dos partes principales, **Diseño e implementación de bases de datos e Implementación software**. Para la segunda parte, se ha diferenciado con distintos colores los apartados que se han evaluado, es decir, dentro de cada proceso implementado o generado, se muestran dos partes y, dentro de estas, el contenido que se ha solicitado a los LLMs y han generado. Dicho contenido es el que se ha tenido en cuenta para el uso de LLMs en esta parte de la **fase 4** y su posterior evaluación. Por último, en la primera parte, es decir, la relacionada a la implementación de base de datos, se tiene en cuenta todos los elementos de bases de datos generados por cada LLM y, a lo largo de la lectura, se explica en mayor detalle cómo se ha contabilizado, al igual que en la parte de implementación software.

El diagrama de la estructura del *uso de LLMs* de este capítulo es el mostrado en la siguiente página:

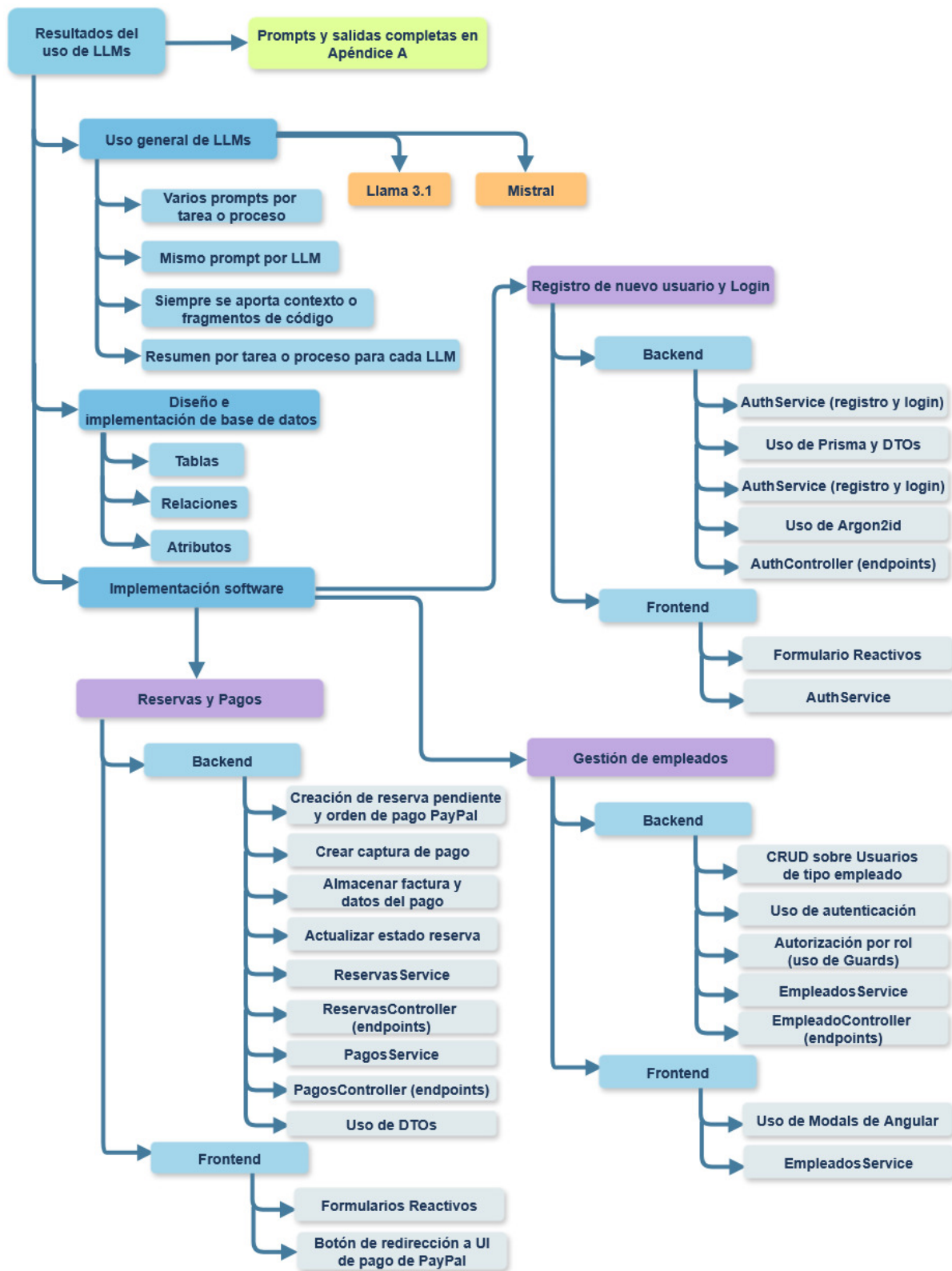


Figura 32: Diagrama de estructura del uso de LLMs en las fase 4. Fuente: elaboración propia.

A continuación, se va a mostrar un resumen del uso de LLMs por cada tarea:

### **Diseño e implementación de base de datos**

Entre las distintas tareas de esta fase, se ha decidido comenzar por la base de datos. Para ello, el *prompt* utilizado además de incluir el enunciado del caso práctico, incorpora una breve descripción del sistema y los requisitos de información (sección B.1.1) obtenidos mediante el desarrollo manual. De esta forma, se logra una salida más completa y precisa.

En ocasiones, el contenido de salida mostraba más de una relación entre las mismas dos tablas por lo que se ha decidido optar por elegir la primera generada, es decir, si en una tabla A aparecen dos relaciones de distinta cardinalidad hacia una tabla B que hacen referencia a la misma lógica, se ha optado por elegir la primera y, si cuando se llega a analizar el contenido generado para la tabla B se ha generado primero la relación que ha sido descartada en la tabla A, entonces se tiene en cuenta la segunda relación para mantener concordancia en el modelo entidad-relación generado por los LLMs.

### **Registro de nuevo usuario y Login**

Para la implementación de esta funcionalidad, se ha indicado en un primer *prompt* que se desarrolle un servicio *AuthService* con dos funciones principales, registro y login. Asimismo, se ha indicado la estructura del modelo de la tabla usuarios en el esquema de *Prisma* y los *DTOs* que reciben por parámetros las funciones. También, en un segundo *prompt* se indicó que se utilizara *Argon2id* para el *hashing* de la contraseña. Finalmente, se solicitó a los modelos que generaran el controlador con dos *endpoints* correspondientes a las funciones generadas.

Continuando con el *frontend*, se indicó a los modelos que generaran un *formulario reactivo* para el componente de estas dos funcionalidades y, por último, el desarrollo de un servicio de autenticación que se encargara de realizar peticiones al *backend*. Para este último servicio, es necesario un servicio de almacenamiento local para la gestión del *access\_token* del cliente por lo que se adjuntó junto a las entradas, para enfocar la generación a la lógica *TypeScript* del formulario y servicio de autenticación.

### **Gestión de empleados**

La implementación de esta gestión ha requerido primero indicar las operaciones que hay que desarrollar sobre la entidad **Usuarios** de tipo **empleado**. Estas operaciones, mencionadas a lo largo del proyecto son *CRUD*. Por otro lado, es necesario que tras la autenticación desarrollada, se implemente una autorización basada en roles, haciendo uso de *Guards*. Una vez solicitado esto, se continúa con el desarrollo de un servicio y un controlador con los *endpoints*

que implemente los *Guards* y servicio desarrollado.

Por último, para la implementación del *frontend*, se indicó el uso de *Modals* de *Bootstrap* para la lógica de las interfaces. Asimismo, se indicó la creación de un servicio que enviara solicitudes con la información correspondiente a la API REST.

### **Reservas y Pagos**

La implementación software realizada de estos dos procesos relevantes se hace de manera conjunta, es decir, cuando se crea una reserva en estado *pendiente*, se crea la orden de pago en PayPal y cuando el usuario acepta el pago, se almacenan los datos de dicho pago, se genera la factura correspondiente y se modifica el estado de la reserva. Para los LLMs se ha proporcionado los *DTOs* correspondientes, utilizados en los controladores y servicios. Asimismo, estos servicios ya incluyen llamadas a la API de PayPal. En resumen, primero se debió implementar las llamadas a la API de PayPal, luego implementar un servicio de reservas que las creara en estado pendiente y, tras esto, utilizar otro servicio, en concreto de pagos, encargado de utilizar el servicio de PayPal y de modificar para crear y capturas las órdenes de pago, gestionando a su vez el estado de las reservas, almacenando datos de pago y de facturas.

Para el *frontend* se necesitó de una interfaz que incluyera un *formulario reactivo* para introducir los datos y botones que se encargaran de la lógica para continuar con el pago mediante PayPal.

### 4.3.1. Llama 3.1

#### Implementación de la base de datos

Comenzando con la base de datos, Llama 3.1 tras introducir por entrada el enunciado del caso práctico, una breve descripción del sistema y los requisitos de información del mismo, ha identificado un total de 7 entidades y 6 relaciones **distintas** de clave foráneas. Entre las distintas relaciones, se podría destacar otra relación foránea implícita la cual es que monitor no es más que una generalización de usuario y, el contenido generado por el LLM destaca cuatro roles de usuario entre los que se encuentran el tipo de usuario "monitor".

Las entidades generadas a partir de la entrada por Llama 3.1 son las siguientes:

Tabla Usuario
Tabla Monitor
Tabla Actividad
Tabla Reserva
Tabla Pago
Tabla Factura
Tabla Comentario

En el caso de las distintas relaciones generadas, se han obtenido las siguientes relaciones de clave foránea:

Desde → Hacia	Cardinalidad	Descripción
<b>Reserva → Usuario</b>	N : 1	Un usuario puede realizar muchas reservas. Cada reserva la hace un solo usuario.
<b>Reserva → Actividad</b>	N : 1	Muchas reservas pueden referirse a una misma actividad.
<b>Pago → Reserva</b>	1 : 1	Cada reserva tiene un único pago. Cada pago pertenece a una sola reserva.
<b>Factura → Pago</b>	1 : 1	Una factura se genera por un único pago.
<b>Comentario → Usuario</b>	N : 1	Un usuario puede hacer muchos comentarios. Cada comentario lo hace un solo usuario.
<b>Comentario → Actividad</b>	N : 1	Una actividad puede recibir muchos comentarios. Cada comentario es sobre una actividad.

La tabla monitor no dispone de ninguna relación explícita tras analizar el diagrama propuesto por el LLM de manera textual ni en el código SQL proporcionado.

Finalmente, en el caso de los atributos generados por el modelo, se han obtenido los siguientes:

<b>Entidad</b>	Número de atributos
<b>Usuario</b>	8
<b>Monitor</b>	2
<b>Actividad</b>	9
<b>Reserva</b>	4
<b>Pago</b>	4
<b>Factura</b>	2
<b>Comentario</b>	3

Para los atributos se ha tenido en cuenta todos los generados en el código SQL o en el diagrama textual, incluyendo atributos de clave primaria y foránea, independiente de la cardinalidad de la relación.

Hasta este punto se muestra el resumen del diseño e implementación de bases de datos generado por Llama 3.1.

Finalmente, en cuanto a la **implementación software** se procede a mostrar un resumen de los módulos obtenidos mediante el uso de Llama 3.1. para cada proceso de negocio relevante:

**Registro de nuevo usuario y Login**

<b>Módulo</b>	<b>Resumen</b>
<b>Servicio (NestJS AuthService)</b>	<p><b>Registro:</b> comprueba duplicados (usuario/email/DNI) con Prisma; crea usuario; <i>hashing</i> de la contraseña con <i>argon2</i>; emite JWT, pero con secreto en claro.</p> <p><b>Login:</b> busca por <i>nombre_usuario</i>; valida <i>password</i> con <i>argon2.verify</i>; devuelve JWT con <i>id</i>, <i>nombre_usuario</i>, <i>tipo_usuario</i>.</p>
<b>Controlador (NestJS AuthController)</b>	Endpoints: <i>POST /auth/registro</i> y <i>POST /auth/login</i> , utilizan directamente el servicio.
<b>Frontend</b>	<p><b>AuthService:</b> métodos <i>registrar</i> y <i>login</i> que llaman a <i>/auth/registro</i> y <i>/auth/login</i> del <i>backend</i>, usa <i>HttpClient</i> y <i>toPromise()</i>.</p> <p><b>StorageService:</b> uso de <i>localStorage</i> para guardar token de acceso para la sesión en el lado del cliente.</p> <p><b>AuthComponent:</b> formularios reactivos de registro y login con validaciones.</p>

Cuadro 68: Resumen de código generado por Llama 3.1 para el registro de nuevos usuarios (clientes) y login de usuarios existentes en el sistema. Fuente: Llama 3.1..

Como observaciones, el código generado presenta un desajuste en campos entre *frontend* y *backend* que no fueron vistos en el primer análisis. Este desajuste de campos es que login utiliza nombre de usuario mientras que el *frontend* envía un dato de tipo correo electrónico. Además, el registro en el *backend* utiliza un dato de nombre de usuario que tampoco se muestra en el formulario de la lógica del *formulario reactivo* generado. Por último, no se indicó en la entrada del *prompt*, pero el secreto que ha generado viene en plano y no desde un archivo de configuración o entorno como *dotenv*. Finalmente, se hizo uso de cuatro *prompts* para obtener todo el código.

**Gestión de empleados**

<b>Módulo</b>	<b>Descripción</b>
<b>Servicio (NestJS EmpleadoService)</b>	Implementa operaciones CRUD sobre empleados: <i>crearEmpleado</i> inserta un nuevo usuario con tipo empleado; <i>obtenerEmpleados</i> devuelve la lista completa; <i>actualizarEmpleado</i> ; <i>eliminarEmpleado</i> elimina tras comprobar que existe. Realiza control de excepciones
<b>Controlador (NestJS EmpleadosController)</b>	Endpoints protegidos con JWT y <i>Guard</i> de roles ( <i>@Roles(Role.Responsable)</i> + <i>JwtAuthGuard</i> ): <i>POST /empleados/nuevo</i> crea un empleado; <i>GET /empleados</i> lista empleados; <i>PATCH /empleados/:id</i> actualiza un empleado; <i>DELETE /empleados/:id</i> elimina un empleado.
<b>Frontend</b>	Componente <i>standalone</i> con <i>formularios reactivos</i> y <i>HttpClient</i> . Funcionalidades: listar empleados, crear, editar y eliminar. Usa formularios reactivos con validaciones básicas y modales de Bootstrap 5 para creación y edición. Maneja errores y selección de empleado. Invoca la API REST del <i>backend</i> .

Cuadro 69: Resumen de código generado por Llama 3.1 para la gestión de empleados como responsable. Fuente: Llama 3.1.

Como observaciones, destaca que el servicio de empleados del *backend* mezcla Prisma ORM con anotaciones de TypeORM (*EntityRepository*), lo cual es inconsistente. El controlador requirió de dos *prompts* para corregir su salida. Finalmente, para generar este código con Llama 3.1., se requirió de cuatro *prompts*.

### 4.3.2. Mistral

#### Implementación de la base de datos

Para el uso de Mistral se ha tenido en cuenta el mismo prompt que con el anterior modelo, es por ello que se va a presentar de manera directa y resumida la información generada por este LLM. Este modelo ha generado un total de 7 tablas y 7 distintas relaciones de claves foráneas. Los atributos se presentan más tarde en esta misma sección, pero se ha dado el caso en el que el LLM ha optado por no proporcionar todos los atributos necesarios para algunas tablas, como puede ser el caso de la entidad factura.

Las entidades generadas por Mistral son las siguientes:

Tabla Usuario
Tabla Categoría
Tabla Actividad
Tabla Monitor
Tabla Reserva
Tabla Pago
Tabla Factura

Este modelo ha proporcionado siete tablas referentes a entidades entre las que se destaca la entidad categoría que más tarde se evaluará su contenido respecto al modelo entidad-relación obtenido de forma manual.

La generación de distintas relaciones de clave foránea muestra el siguiente contenido:

Desde → Hacia	Cardinalidad	Descripción
<b>actividad → categoría</b>	N:1	Cada actividad pertenece a una única categoría, pero una categoría puede tener muchas actividades.
<b>monitor → usuario</b>	1:1	Cada monitor es un usuario con formación adicional. Cada registro en monitor se asocia con exactamente un usuario.
<b>reserva → usuario</b>	N:1	Cada reserva es hecha por un cliente (usuario), y un usuario puede tener múltiples reservas.
<b>reserva → actividad</b>	N:1	Cada reserva se refiere a una única actividad, pero una actividad puede tener muchas reservas.
<b>pago → reserva</b>	1:1	Cada pago está asociado a una única reserva.
<b>pago → factura</b>	N:0..1	Cada pago puede estar vinculado a una factura (opcional), y una factura puede estar asociada a varios pagos.

En el caso del análisis y el diagrama proporcionado por el modelo, ocurre lo mismo que con el anterior. Se diferencian roles a través de una enumeración de usuarios, pero no se destacan en entidades distintas, salvo monitor. Además, en el contenido generado por los modelos para la tabla pago se han proporcionado dos posibilidades de relaciones para la misma tabla, reserva, por ello se ha seleccionado la primera. Por último, el modelo destaca una tabla categoría la cual se tiene en cuenta a la hora de aplicar las métricas.

Por último, los atributos generados por el modelo para cada entidad son los siguientes:

<b>Entidad</b>	Número de atributos
<b>Usuario</b>	8
<b>Categoría</b>	2
<b>Actividad</b>	8
<b>Monitor</b>	3
<b>Reserva</b>	4
<b>Pago</b>	6
<b>Factura</b>	3

Entre los atributos generados por Mistral, ciertos atributos como '*etcétera*' en la tabla **Factura** indican que el modelo podría haber generado al menos un atributo más por lo que se da por válido y contabiliza como uno.

Hasta este punto llega el resumen de lo generado por **Mistral** para el diseño e implementación de la base de datos.

A continuación, se muestra el resumen de la implementación software realizada por Mistral:

**Registro de nuevo usuario y Login**

<b>Módulo</b>	<b>Resumen</b>
<b>Servicio (NestJS AuthService)</b>	<p><b>Registro:</b> crea el usuario en la base de datos con Prisma, hashea la contraseña con <i>argon2</i> antes de guardarla y devuelve un JWT con <i>id</i>, <i>username</i> y <i>tipo_usuario</i>.</p> <p><b>Login:</b> busca al usuario por <i>nombre_usuario</i>, compara la contraseña con <i>argon2.verify</i>; si coincide, genera y devuelve el JWT; si no, lanza <i>UnauthorizedException</i>.</p>
<b>Controlador (NestJS Auth-Controller)</b>	Define dos endpoints: <i>POST /auth/register</i> → llama a <i>AuthService.register</i> y responde con el token. <i>POST /auth/login</i> → llama a <i>AuthService.login</i> y responde con el token. Usa <i>@Body()</i> para DTOs ( <i>CrearUsuarioDTO</i> , <i>AuthDTO</i> ) y <i>Express Response</i> para enviar JSON.
<b>Frontend (Angular)</b>	Ejemplo básico de <i>AuthService</i> (con <i>HttpClient</i> para <i>/login</i> y <i>/register</i> ). Ejemplo de <i>LoginComponent</i> con formularios reactivos ( <i>FormBuilder</i> , <i>Validators</i> ). No incluye integración con almacenamiento de sesión/token ni un flujo completo de registro.

Cuadro 73: Resumen de código generado por Mistral para el registro de nuevos usuarios (clientes) y login de usuarios existentes en el sistema. Fuente: Mistral.

Como observaciones, hay inconsistencias en los nombres de DTO (*CrearUsuarioDTO* y *CreateUsuarioDTO*), aunque no es un problema. El *frontend* no lo ha generado del todo completo ya que se limita a ejemplos básicos sin enlazar todo el flujo de autenticación.

Para generar todo el código de esta funcionalidad, se ha requerido el uso de seis *prompts*. Las salidas de los *prompts* cuatro y cinco, no mostraban código, si no respuestas en texto plano indicando los pasos a seguir para la implementación de cierto código, indicando el LLM que no era capaz de generar el código.

### Gestión de empleados

Módulo	Descripción
<b>Servicio (NestJS EmpleadosService)</b>	CRUD completo: <i>createEmpleado</i> crea un usuario con <i>tipo_usuario = empleado</i> y contraseña <i>hasheada</i> con <i>argon2id</i> ; <i>findAllEmpleados</i> devuelve la lista de empleados filtrando por tipo; <i>findOneEmpleado</i> obtiene un empleado por id y lanza <i>UnauthorizedException</i> si no existe; <i>updateEmpleado</i> actualiza los datos por id; <i>deleteEmpleado</i> elimina un empleado y lanza excepción si no existe. Usa <i>PrismaService</i> para la interacción con la base de datos.
<b>Controlador (NestJS EmpleadosController)</b>	Define los endpoints: <i>POST /empleados</i> crea empleado; <i>GET /empleados</i> lista empleados; <i>GET /empleados/:id</i> obtiene un empleado concreto; <i>PUT /empleados/:id</i> actualiza; <i>DELETE /empleados/:id</i> elimina. Seguridad: combina <i>JwtAuthGuard</i> y <i>RolesGuard('responsable')</i> para proteger endpoints críticos y validar rol.
<b>Frontend</b>	Componente <i>standalone</i> en Angular 17. Usa <i>HttpClient</i> , <i>formularios reactivos</i> , modales, entre otros. Métodos: crear, editar y eliminar empleados (este último está declarado, pero no implementado). Maneja selección de empleado y apertura/cierre de modales con <i>@ViewChild</i> .

Cuadro 74: Resumen de código generado por Mistral para la gestión de empleados como responsable. Fuente: Mistral.

Como observaciones, se podría mejorar el uso de excepciones para que sean más correctas según el escenario. El *frontend* no está del todo completo ya que no proporcionó toda la lógica de formularios ni servicios. El número total de *prompts* utilizados es de tres.

Finalmente, para la implementación software de **reservas y pagos** se ha decidido no mostrar de manera resumida ya que conlleva varias clases y métodos, mucho más que el resto. Por ello, se recomienda visualizar mejor en los archivos de salidas.

## 4.4. Evaluación de resultados

Esta sección recoge la aplicación de métricas en las distintas tareas de esta fase. A pesar de que para cada métrica se aplique el mismo concepto, la forma de evaluar cambia ligeramente con la tarea, pero los resultados son igual de válidos. Por ello, se presenta la aplicación de todas las métricas para cada LLM y un reporte final con todos los resultados por modelo:

### 4.4.1. Completitud

Comenzando con esta fase, la tarea de diseño e implementación de base de datos. Para esta métrica se ha decidido tener en cuenta todo lo generado, es decir, número de tablas, relaciones y atributos. Cada una de estas va a tener asociado una ponderación, dando más peso a las entidades y relaciones y menos a los atributos, no analizando el tipo de dato generado en este último elemento, al menos en esta métrica. Asimismo, sí que se ha tenido en cuenta la cardinalidad de las relaciones foráneas. La tabla de ponderaciones es la siguiente:

Elemento	Peso sugerido
Entidad	5
Relación	4
Atributo	1

Teniendo en cuenta estas ponderaciones y la fórmula de completitud de la sección 2.3, la forma de medir esta métrica respecto al desarrollo obtenido de forma manual es la siguiente:

$$\text{Completitud\_BBDD} = \frac{\text{Peso total de elementos coincidentes}}{\text{Peso total esperado}} \times 100$$

Por ello, para el caso práctico de referencia se obtiene los siguientes resultados tras aplicar los criterios de ponderación:

Elemento	Número de elementos	Resultado
Entidades (tablas)	8	40
Relaciones	10	40
Atributos	56	56

Estos datos se pueden verificar con la figura 29. Para las relaciones M:N se ha decidido contabilizar como dos debido a que en el modelo físico de datos se traducen como dos relaciones 1:N. Para el diagrama del modelo entidad-relación desarrollado de forma manual se ha obtenido un valor total de ponderación de 136.

Comenzando con **Llama**, se obtienen los siguientes elementos coincidentes; 7 tablas, 5 relaciones de clave foránea y 29 atributos. Por ello, se ha obtenido el siguiente resultado:

$$\text{Compleitud\_BBDD} = \frac{7 \times 5 + 5 \times 4 + 29 \times 1}{136} \times 100 = \frac{84}{136} \times 100 = 61,76 \%$$

La aplicación de esta métrica en **Mistral** ha proporcionado los siguientes resultados; 6 tablas, 4 relaciones de clave foránea y 28 atributos. El resultado de completitud en esta tarea del LLM Mistral es:

$$\text{Compleitud\_BBDD} = \frac{6 \times 5 + 4 \times 4 + 28 \times 1}{136} \times 100 = \frac{74}{136} \times 100 = 54,41 \%$$

Continuando con la **evaluación de código** generado por los LLMs, la completitud se va a realizar una cobertura de código, es decir, medir hasta qué punto el contenido generado por el LLM cubre los elementos esperados en la solución de referencia. Por ello, se identifican los elementos presentes en la solución de referencia como fragmentos de código, funcionalidades, inclusión de *DTOs*, entre otros. Luego, se contabilizan los elementos coincidentes en la solución generada por el LLM y se calcula el porcentaje de la completitud.

Las líneas de código generadas por los modelos de IAG no tienen que ser exactamente idénticas a las de la referencia, aunque sí parecidas o equivalentes. Una funcionalidad parcial se contabiliza como una coincidente siempre que esté más cerca de ser una solución total

debido a que en el proceso de desarrollo software asistido, salidas parciales implican utilizar más de un *prompt* para corregir la salida.

En resumen, la completitud de la generación de código se entiende como un todo, es decir, un conjunto en el que se valida si las clases están completas y bien construidas respecto al desarrollo realizado de forma manual.

Se va a mostrar el cálculo de la completitud por cada LLM, para el numerador se va a tener en cuenta la suma de elementos generados coincidentes por cada funcionalidad, mientras que el denominador será el número total de elementos presentes en el desarrollo de referencia. Se tiene en cuenta el total de elementos, tanto del *frontend* como del *backend*.

Para el caso de **Llama 3.1**, se han generado los siguientes elementos coincidentes:

Funcionalidad	Total elementos coincidentes	Elementos de referencia
Registro de nuevo usuario y Login	34	40
Gestión de empleados	32	51
Reservas y Pagos	25	30

Cuadro 75: Resumen de elementos coincidentes en el código generado por Llama 3.1 para cada funcionalidad respecto al total de elementos de referencia. Fuente: elaboración propia.

Para el caso de **Mistral** se han obtenido los siguientes elementos coincidentes tras el análisis respecto al desarrollo manual:

Funcionalidad	Total elementos coincidentes	Elementos de referencia
Registro de nuevo usuario y Login	32	40
Gestión de empleados	27	51
Reservas y Pagos	25	30

Cuadro 76: Resumen de elementos coincidentes en el código generado por Mistral para cada funcionalidad respecto al total de elementos de referencia. Fuente: elaboración propia.

Los datos se han obtenido de establecer primero los elementos de referencia en una tabla para cada proceso. Después, se ha hecho el análisis entre código generado por LLM y referencia para ver las coincidencias o equivalencias:

Elemento	Llama 3.1	Mistral
AuthService	X	X
Inyección PrismaService	X	X
Inyección JwtService	-	X
Función registro	X	X
DTO registro	X	X
Hash mediante Argon2id	X	X
Firma de JWT en registro	X	X
Excepciones Registro	X	-
Devolver JWT registro	X	X
Función login	X	X
DTO login	X	X
Verify de Argon2id en login	X	X
Firma de JWT en login	X	X
Excepciones login	X	X
Uso de PrismaService	X	X
Devolver JWT login	X	X
Ruta controlador	X	X
Endpoint registro	X	X
Endpoint login	X	X
Uso de AuthService	X	X
Body registro	X	X
Body login	X	X

Cuadro 77: Elementos coincidentes de registro y login generado por los LLMs respecto al desarrollo de referencia (*backend*). Fuente: elaboración propia.

Elemento	Llama 3.1	Mistral
AuthService	X	X
Inyección HttpClient	X	X
Inyección StorageService	-	-
HttpClient POST registro	X	X
Observable registro	-	X
Persistencia sesión registro	-	-
HttpClient POST login	X	X
Observable login	-	X
Persistencia sesión login	X	X
RegistroComponent	X	-
Inyección AuthService	X	-
Formulario Reactivo registro	X	-
Validaciones formulario	X	-
Envío form registro	X	-
Uso de AuthService	X	X
LoginComponent	-	X
Formulario Reactivo login	X	X
Envío form login	X	X

Cuadro 78: Elementos coincidentes de registro y login generado por los LLMs respecto al desarrollo de referencia (*frontend*). Fuente: elaboración propia.

Elemento	Llama 3.1	Mistral
EmpleadosService	X	X
Inyección PrismaService	X	X
crearEmpleado	X	X
DTO Empleado	X	X
Hash mediante Argon2id	-	X
Excepciones Creación	-	-
getEmpleados	X	X
Listar usuarios de rol empleado	-	X
Listar de empleados sin hash devuelto	-	-
Excepciones Lista de empleados	-	-
getEmpleado por ID	X	X
Empleado sin hash devuelto	-	-
Excepciones getEmpleado	X	X
actualizarEmpleado por ID	X	X
DTO UpdateEmpleado	X	X
Devolver empleado actualizado sin hash	X	-
Excepciones actualizarEmpleado	-	-
eliminarEmpleado por ID	X	X
Excepciones eliminarEmpleado	X	-
Uso de PrismaService	X	X
JwtAuthGuard	X	X
RolesGuard para responsable	X	X
Ruta controlador	X	-
Endpoint crear	X	X
Body crear	X	X
Endpoint lista	X	X
Endpoint get por ID	-	X
Endpoint patch por ID	X	-
Body UpdateEmpleado	X	X
Endpoint delete por ID	X	X
Uso de EmpleadosService	X	X

Cuadro 79: Elementos coincidentes de la gestión de empleados generado por los LLMs respecto al desarrollo de referencia (*backend*).

Fuente: elaboración propia.

Elemento	Llama 3.1	Mistral
AdminService	-	-
Inyección HttpClient	X	X
HttpClient POST crear	X	-
Observable crear	-	-
HttpClient GET lista	X	-
Observable lista	-	-
HttpClient PATCH actualizar	-	-
Observable actualizar	-	-
HttpClient DELETE	X	-
Observable delete	-	-
AdminComponent	X	X
Modal agregar	X	X
Modal lista	-	-
Modal eliminar	-	X
Modal actualizar	X	X
Formulario reactivo agregar	-	-
Formulario reactivo actualizar	-	-
Validaciones formularios	-	-
Uso de AdminService	-	-
Envío de formularios	X	-

Cuadro 80: Elementos coincidentes de la gestión de empleados generado por los LLMs respecto al desarrollo de referencia (*frontend*).

Fuente: elaboración propia.

Elementos	Llama 3.1	Mistral
ReservasService	X	X
Inyección PrismaService	X	X
Inyección PagosService	X	X
DTO CrearReservaDTO	-	X
Función CrearReserva pendiente	-	-
Excepciones de creación	-	-
Devolver reservaID	X	-
Función CrearReservaYPago	X	X
Devolver reservaID	X	X
Devolver orderID	X	X
Devolver approveLink (PayPal UI)	X	X
PayPalService	X	X
Inyección HttpService	X	X
Inyección ConfigService	X	X
Función getAccessToken	X	X
Función crearOrder	X	X
Función capturarOrder	X	X
PagosService	X	X
Inyección PrismaService	X	X
Inyección PayPalService	X	X
Función crearPago por reservaID	X	X
Devolver orden de pago	X	X
Crear Pago en BD	X	X
Función capturarOrderPago por orderID y Factura	X	X
Actualizar Reserva	X	X
Crear Factura en BD	X	X
Devolver reservaID pagoID y numFactura	-	-
Uso de PayPalService	X	X
Uso de PagosService	X	X
Uso de PrismaService	-	-

Cuadro 81: Elementos coincidentes de reservas y pagos generado por los LLMs respecto al desarrollo de referencia (*backend*). Fuente: elaboración propia.

Una vez mostrado el resumen de elementos coincidentes en las anteriores tablas, la completitud de la **generación de código** por los LLMs es la mostrada a continuación.

El número total de elementos de referencia es:  $40 + 51 + 30 = 121$

Para el caso de **Llama 3.1** se obtiene la siguiente completitud:

$$\text{Completitud GEN\_SW} = \frac{(34 + 32 + 25)}{121} \times 100 = \frac{91}{121} \times 100 = 75,21\%$$

Y, finalmente, para el caso de **Mistral** se obtiene el siguiente resultado en la completitud de esta fase:

$$\text{Completitud GEN\_SW} = \frac{(32 + 27 + 25)}{121} \times 100 = \frac{84}{121} \times 100 = 69,42\%$$

#### 4.4.2. Corrección

Entre las tareas de la fase de implementación software se distinguen varias y, aunque se apliquen las mismas métricas, dependiendo de la tarea se aplican ciertos criterios o ponderaciones distintas. Aún así, las categorías definidas en la sección 2.3 para el caso de la corrección, se siguen aplicando.

Par la aplicación de esta métrica, en tareas como la **implementación de la base de datos** se tiene en cuenta la validez del contenido generado por los modelos y su utilidad ante el caso práctico planteado. A su vez, elementos como presencia de entidades o tablas y relaciones de clave foránea también son contabilizadas. Por último, para el caso de los atributos de las entidades se mide lo correctos que sean para el sistema planteado. En definitiva, para aplicar esta métrica y aplicar las categorías de esta, definidas previamente, para el caso de la implementación de la base de datos se ha decidido evaluar los elementos principales por separado para después, recoger un resultado general del desempeño de cada modelo. Por ello, se ha tenido en cuenta las siguientes características a la hora de clasificar cada elemento en la respectiva corrección. Dichas características se resumen en viabilidad para las tablas, lógica en el caso de las relaciones entre las tablas generadas y por último, utilidad para el caso de los atributos generados. Para el caso de las relaciones, solo se han considerado relaciones directas, no las posibles a través de una tabla intermedia que no sea resultado directo de una relación M:N. Por último, en los atributos no se ha tenido en cuenta la presencia o ausencia de valores, es

decir, si se define como NULL o no. Además, para esto, también se tiene en cuenta si el tipo de dato es correcto según su nombre.

En la corrección, se tiene en cuenta todo el contenido generado, no solo el coincidente con modelo de referencia ya que para eso está métrica de completitud, pero sí que se tiene en cuenta cómo de correctos han sido los componentes generados respecto al desarrollo manual y cómo de viables, lógicos y útiles son para el sistema planteado.

Elemento	Categoría de corrección	Descripción de la categoría
<b>Tablas</b>	Incorrecto	Tabla irrelevante, inexistente o mal definida.
	Parcialmente correcto	Tabla conceptualmente útil pero general, ambigua o redundante.
	Correcto	Tabla representa una entidad clave, bien planteada.
	Excelente	Además de correcta, organiza bien la información o anticipa necesidades del dominio.
<b>Relaciones</b>	Incorrecto	No existe o conecta entidades de forma ilógica.
	Parcialmente correcto	Lógica general válida pero con errores de clave o dirección.
	Correcto	Relación coherente y bien construida.
	Excelente	Añade cardinalidad, integridad o relaciones implícitas adecuadas.
<b>Atributos</b>	Incorrecto	Atributo mal ubicado, sin sentido o irrelevante.
	Parcialmente correcto	Útil pero mal nombrado, ambiguo o poco preciso.
	Correcto	Útil, esperable y correctamente definido.
	Excelente	Aporta valor adicional.

Cuadro 82: Rúbrica para la evaluación de corrección de elementos en la implementación de la base de datos. Fuente: elaboración propia.

Para recoger los datos de la corrección de la base de datos, se ha realizado una tabla que distribuye las distintas categorías por cada elemento en ambos modelos. La columna **Presente** hace referencia a si está presente o no dicho elemento en el modelo de referencia utilizado, es decir, si la tabla A tiene relación directa con la tabla B en el modelo entidad-relación desarrollado manualmente.

Tabla	Presente	Categoría de corrección
Usuario	Sí	Correcto - 3
Monitor	Sí	Incorrecto - 1
Actividad	Sí	Correcto - 3
Reserva	Sí	Correcto - 3
Pago	Sí	Correcto - 3
Factura	Sí	Correcto - 3
Comentario	Sí	Correcto - 3

Cuadro 83: Corrección de las tablas generadas por Llama 3.1. Fuente: elaboración propia.

El cuadro 83 muestra la corrección de las tablas generadas por el modelo Llama 3.1. La tabla **Monitor**, a pesar de estar presente, es incorrecta ya que no dispone de relaciones ni atributos distintivos. El resto de tablas es correcta respecto al sistema planteado y desarrollado manualmente.

La siguiente tabla muestra la corrección de las relaciones generadas por los modelos.

Desde → Hacia	Cardinalidad	Presente	Categoría de corrección
Usuario → Reserva	1:N	Sí	Correcto - 3
Usuario → Comentario	1:N	Sí	Correcto - 3
Monitor	N/A	No	Incorrecto - 1
Actividad → Reserva	1:N	Sí	Correcto - 3
Actividad → Comentario	1:N	Sí	Correcto - 3
Reserva → Pago	1:1	Sí	Correcto - 3
Pago → Reserva	N:1	No	Incorrecto - 1
Pago → Factura	1:1	No	Parcialmente correcto - 2
Factura → Pago	N:1	No	Incorrecto - 1

Cuadro 84: Corrección de las relaciones generadas por Llama 3.1. Fuente: elaboración propia.

El cuadro 84 muestra la corrección de las relaciones entre tablas generadas por Llama 3.1. Para ello, se ha tenido en cuenta las características de presencia sobre el modelo de referencia y contenido para la lógica del modelo entidad-relación del sistema planteado en el caso práctico. La tabla **Monitor** se categoriza como incorrecta puesto que no posee relaciones. La tabla **Pago** mantiene una relación N:1 con Reserva, cuando primero se generó una 1:1 de Reserva a Pago.

También ocurre con Factura, cuando en Pago primero se definió una relación 1:1. Asimismo, la relación entre Pago y Factura con cardinalidad 1:1 se consideraba parcialmente correcta puesto que las facturas no se relacionan con los pagos si no con los servicios. El resto de relaciones son correctas.

Tabla	Presente	Categoría de corrección
Usuario	Sí	Correcto - 3
Categoría	No	Parcialmente correcto - 2
Monitor	Sí	Correcto - 3
Actividad	Sí	Correcto - 3
Reserva	Sí	Correcto - 3
Pago	Sí	Correcto - 3
Factura	Sí	Correcto - 3

Cuadro 85: Corrección de las tablas generadas por Mistral. Fuente: elaboración propia.

El cuadro 85 muestra la corrección de las tablas generadas por el modelo Mistral. Este modelo no contempló la presencia de la entidad **Comentarios**, pero incluyó otra, la tabla **Categoría**. Dicha tabla no se considera incorrecta por no estar presente en el modelo de referencia, se categoriza como parcialmente correcta por ser una tabla conceptualmente útil, aunque posiblemente redundante a un atributo de la tabla **Actividad**.

Desde → Hacia	Cardinalidad	Presente	Categoría de corrección
Usuario → Monitor	1:1	Sí	Correcta - 3
Usuario → Reserva	1:N	Sí	Correcta - 3
Categoría → Actividad	1:N	No	Correcta - 3
Actividad → Reserva	1:N	Sí	Correcta - 3
Reserva → Actividad	N:1	Sí	Correcta - 3
Reserva → Pago	1:1	Sí	Correcta - 3
Reserva → Pago	1:N	Sí	Incorrecta - 1
Pago → Factura	N:0	No	Incorrecta - 1

Cuadro 86: Corrección de las relaciones generadas por Mistral. Solo relaciones directas. Fuente: elaboración propia.

El cuadro 86 muestra la corrección de las relaciones generadas por el modelo Mistral. El

modelo en ocasiones generó más de una relación similar, pero de distinta cardinalidad entre las dos mismas tablas, como es el caso de **Reserva** con **Pago**. Asimismo, la relación entre **Pago** y **Factura** se considera incorrecta puesto que aunque una factura esté asociada a la compraventa de un bien o servicio, se puede emitir antes o durante el pago así que como la cardinalidad en el lado de **Factura** es opcional, indica que puede haber facturas no asociadas a pagos por lo que es erróneo, ya que en las facturas se detallan las cuantías a pagar.

Para el caso de los atributos se ha decidido hacer una tabla general para cada modelo, es decir, en las filas se muestran las categorías de corrección y en las columnas los modelos por lo que el valor de las celdas representa el recuento de categoría según el modelo. La columna **Presente** la cual aparece en tablas anteriores de esta métrica, no se va a tener en cuenta para los atributos puesto que se evalúa el contenido generado.

Categoría de corrección	Llama 3.1	Mistral
Incorrecto - 1	6	10
Parcialmente correcto - 2	4	2
Correcto - 3	22	21
Excelente - 4	0	1

Cuadro 87: Corrección de los atributos generados por Llama 3.1 y Mistral. Fuente: elaboración propia.

El cuadro 87 recoge la corrección de todos los atributos generados por los LLMs para cada tabla. Se ha tenido en cuenta el tipo de dato, el sentido de este tipo en función del nombre del atributo y el sentido de un atributo sea *null* o no. Entre los atributos categorizados como incorrectos, en Llama 3.1 destacan apellidos, rol, descripción, plazas, fecha de reserva y cantidad de pago debido a la posibilidad de ser nulos. Otros como categoría, teléfono móvil, etcétera, se han categorizado parcialmente correctos en función de la necesidad que se planteara en el caso práctico. Por último, Llama 3.1 no muestra ningún atributo excelente en su generación. El caso de Mistral muestra más atributos incorrectos por razones similares que el otro LLM, pero muestra un atributo excelente el cual es día de la semana por no ser nulo, tener un tipo de dato adecuado, es decir, enumerado y por proporcionar valor adicional.

Se ha hecho un recuento total de la cantidad de elementos generados por cada modelo para la implementación de bases de datos. Este recuento se ha obtenido de las tablas de corrección

de esta tarea de la fase por cada LLM. El recuento total de elementos generados por modelo para el diseño e implementación de la base de datos es:

LLM	Tablas	Relaciones	Atributos	Total
Llama 3.1	7	9	32	48
Mistral	7	8	34	49

Cuadro 88: Número de elementos generados por cada modelo para el diseño e implementación de la base de datos. Fuente: elaboración propia.

Para la corrección de la **implementación software** por parte de los LLMs se ha hecho lo siguiente. Se ha medido la precisión en la implementación y ejecución teórica de los elementos generados por cada LLM respecto a los resultados esperados, es decir, respecto al desarrollado realizado de forma manual. El objetivo es determinar si el código generado funcionaría correctamente en la práctica, comparando con el desarrollo de referencia.

Se ha utilizado las cuatro categorías: **Incorrecto**, cuando la funcionalidad está ausente o implementada de forma errónea; **Parcialmente correcto**, cuando la funcionalidad existe pero presenta errores o defectos que afectan parcialmente a su ejecución; **Correcto**, cuando la funcionalidad está bien implementada y cumple con lo esperado; y **Excelente**, cuando la funcionalidad no solo es correcta, sino que además incorpora mejoras o consideraciones adicionales que enriquecen su implementación.

Debido a la extensión del contenido generado por los LLMs en la tarea de implementación software, se va a mostrar directamente el recuento total de las categorías de corrección por LLM. Para la evaluación, se ha tenido en cuenta todo el contenido generado por los modelos.

LLM	Incorrecto	Parc. correcto	Correcto	Excelente
LLama 3.1	29	13	77	2
Mistral	35	16	69	1

Cuadro 89: Recuento total de la distribución de categorías de corrección de Llama 3.1 y Mistral en la tarea de implementación software. Fuente: elaboración propia.

#### 4.4.3. Rendimiento

Este apartado recoge todo el tiempo que se ha tardado en ejecutar las distintas tareas de esta fase, incluyendo el tiempo de generación de salidas de los LLMs.

FASE 4			
Actividad	Tiempo	Tiempo Llama	Tiempo Mistral
<b>Diseño e implementación BBDD</b>	84 minutos	13.89 segundos	13.84 segundos
<b>Registro de usuarios y login</b>	832 minutos	39.54 segundos	34.82 segundos
<b>Gestión de empleados</b>	679 minutos	35.59 segundos	25.41 segundos
<b>Realizar reserva, factura y pago mediante PayPal</b>	1258 minutos	38.67 segundos	27.94 segundos

Los tiempos relacionados a la implementación software como el desarrollo software de los procesos de negocio más relevantes, incluyen el tiempo invertido en desarrollar el *frontend* y *backend* de dicha parte.

#### 4.4.4. Evaluación de resultados

Comenzando con la métrica de **completitud** y con la tarea de **implementación de la base de datos** se han obtenido los siguientes resultados. En la figura 33 se muestra la ponderación total frente a la desarrollada de forma manual y, en la figura 34 se muestra la comparación de la completitud obtenida por cada LLM:

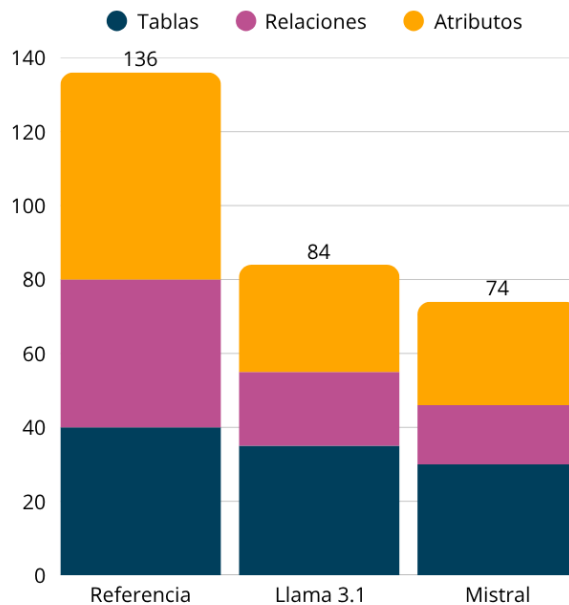


Figura 33: Ponderación total de los elementos generados por LLMs frente a la implementación de referencia. Fuente: elaboración propia.

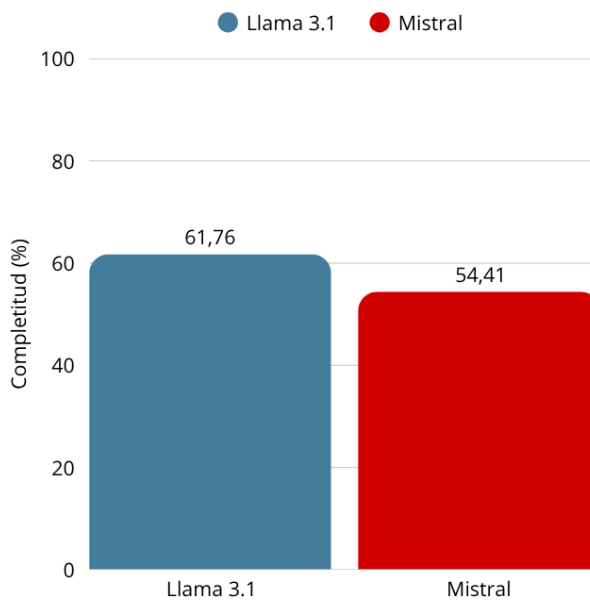


Figura 34: Complejitud en la tarea de bases de datos obtenida por cada LLM. Fuente: elaboración propia.

Continuando con la **complejitud**, ahora se procede a mostrar la obtenida por los LLMs en las tareas de **generación de código**.

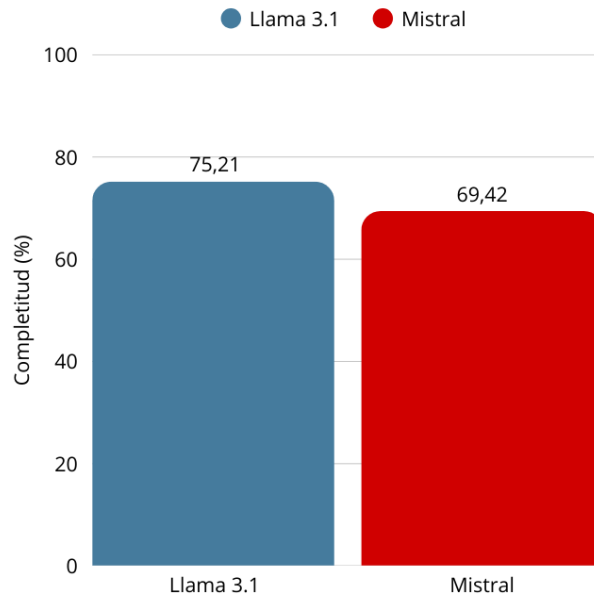


Figura 35: Complejitud en la tarea de generación de código obtenida por cada LLM. Fuente: elaboración propia.

A continuación, se procede a mostrar los resultados que se han obtenido tras aplicar la métrica de **corrección**. Empezando por la tarea de implementación de base de datos, se han recogido los datos de la evaluación de cada elemento generado por los modelos de manera separada y, finalmente, de manera general en cada uno.

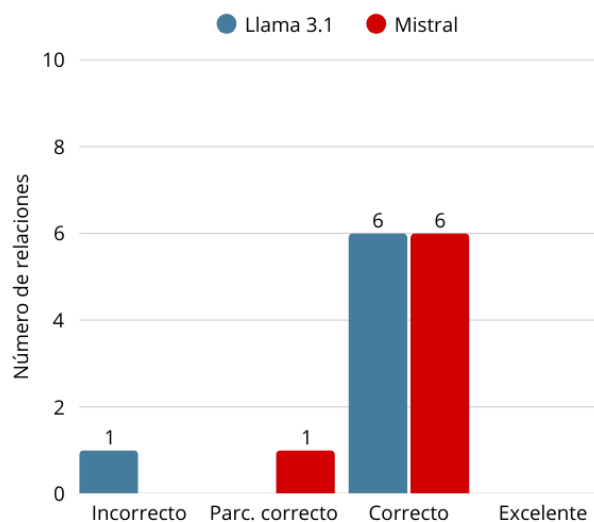


Figura 36: Recuento de categorías en la corrección de tablas generadas por Llama 3.1 y Mistral. Fuente: elaboración propia.

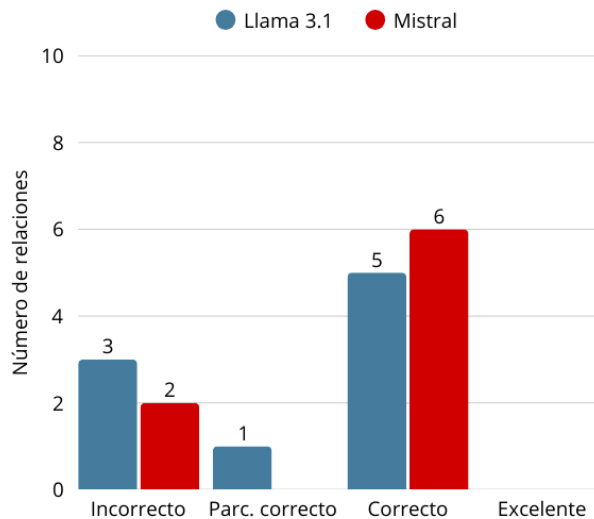


Figura 37: Recuento de categorías en la corrección de relaciones generadas por Llama 3.1 y Mistral. Fuente: elaboración propia.

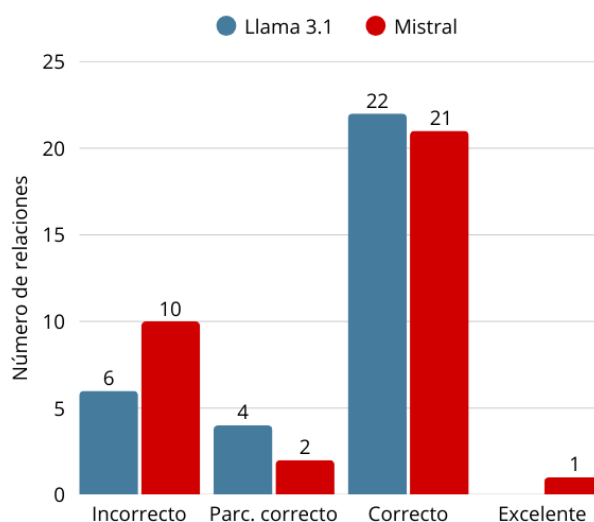


Figura 38: Recuento de categorías en la corrección de atributos generados por Llama 3.1 y Mistral. Fuente: elaboración propia.

Ahora se va a mostrar el resultado general de la corrección de la **implementación de la base de datos**.

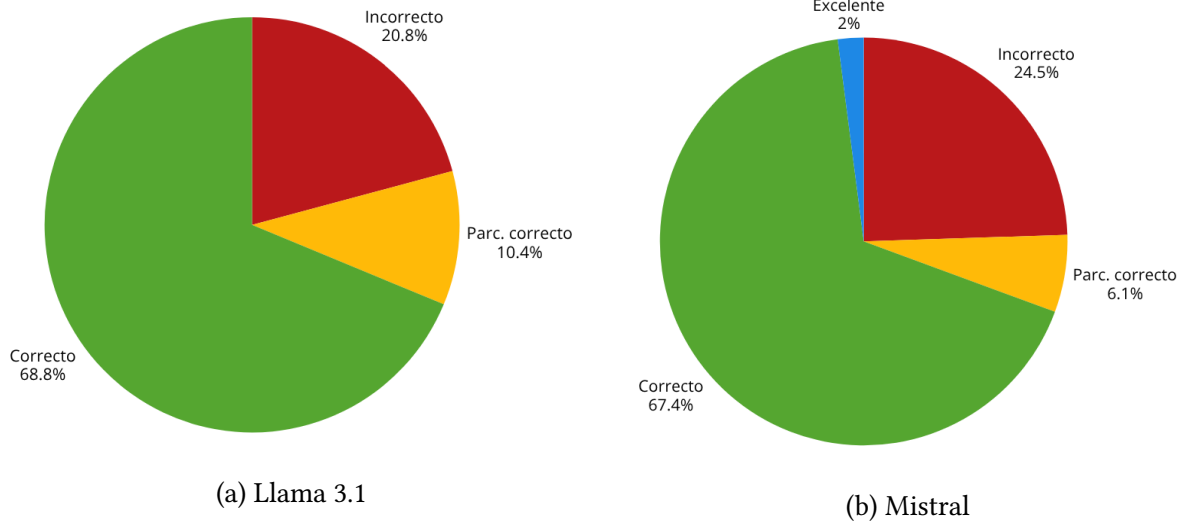


Figura 39: Distribución total de categorías de corrección de elementos de bases de datos obtenidos por cada modelo. Fuente: elaboración propia.

Por último, se procede a mostrar el resultado general de la corrección de la **generación de código**.

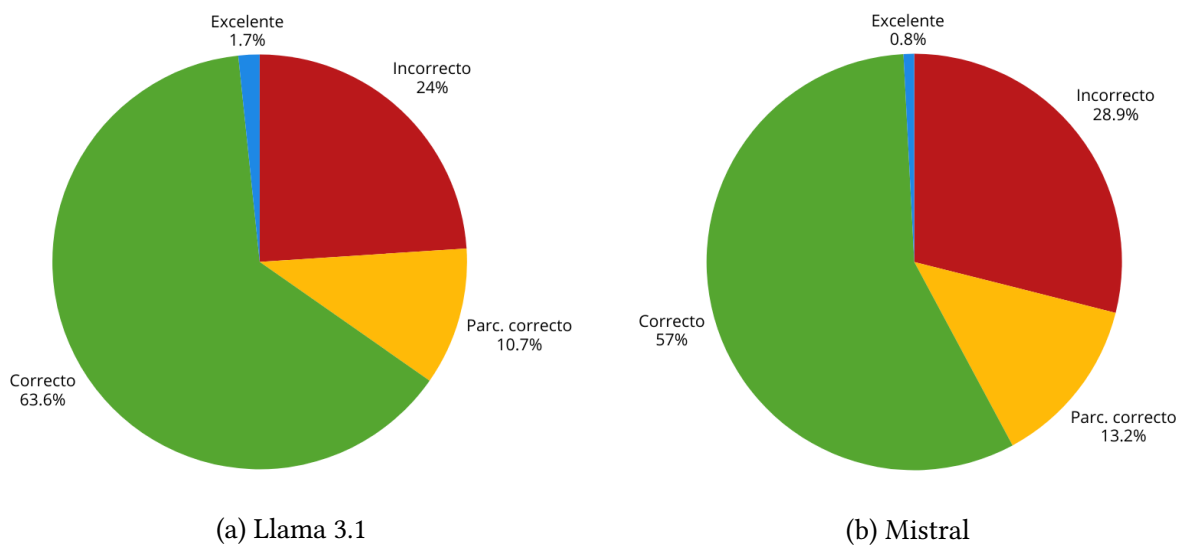


Figura 40: Distribución total de categorías de corrección en la generación de código por cada modelo. Fuente: elaboración propia.

# 5

## Pruebas de software

Este capítulo cubre las pruebas software realizadas en el caso práctico y las generadas por los LLMs para su evaluación y comparación con el desarrollo humano.

En el desarrollo del proyecto se ha seguido una metodología en la que, en primer lugar, se implementó el código del *backend* mediante una API REST que proporciona los distintos *endpoints* necesarios para la aplicación. Una vez completado este desarrollo, se procedió a la elaboración de **pruebas manuales** y *end-to-end*, abarcando tanto casos de prueba positivos como negativos. De este modo, se comprobó no solo el funcionamiento esperado de los flujos principales, sino también la correcta gestión de errores ante peticiones HTTP no válidas, asegurando la robustez de la aplicación.

Es importante señalar que, de acuerdo con las buenas prácticas en pruebas de software, lo recomendable es emplear una base de datos y una API REST específicas para *testing*, de forma que se mantenga aislado el entorno de desarrollo. No obstante, dado que este proyecto constituye un caso práctico ficticio, únicamente utilizado para evaluar el desempeño de LLMs gratuitos frente al desarrollo humano, se ha optado por realizar las pruebas directamente sobre la API REST y la base de datos de desarrollo. A pesar de esto, se reconoce la importancia de estas buenas prácticas.

Las pruebas implementadas no hacen uso de *mocks* ya que el objetivo de las pruebas ha sido validar el comportamiento global de la aplicación, incluyendo la interacción real con la base de datos y las validaciones implementadas en los distintos niveles de la arquitectura. De este modo, las pruebas desarrolladas han permitido garantizar que los flujos principales de la aplicación funcionan correctamente en condiciones similares a las que tendría un entorno real.

A lo largo de este capítulo se muestran las pruebas manuales realizadas, las pruebas *end-to-end* desarrolladas y el uso de LLMs para esta última fase seleccionada del proceso de desarrollo

software, finalizando con la aplicación de distintas métricas y recopilación de resultados obtenidos.

## 5.1. Pruebas realizadas

Las pruebas están orientadas a la parte *backend* del caso práctico. Para ello, se ha hecho uso de la API REST implementada en la fase anterior, implementación software. Asimismo, se ha decidido enfocar el desarrollo y generación de pruebas a esta parte de la aplicación web debido a que pruebas para el *frontend* del tipo *end-to-end* requieren del conocimiento de las interfaces y, como la mayor parte de la lógica de este caso práctico se agrupa en el *backend*, todas las pruebas son orientadas a esta parte del desarrollo.

Por último, en el apéndice A se encuentran los *prompts* utilizados para generar las pruebas haciendo uso de los LLMs. Solo se muestra la entrada, no las salidas y, es que estas últimas, se encuentran en archivos distintos a la memoria y se adjuntan con la entrega.

### 5.1.1. Pruebas manuales

El primer tipo de pruebas que se han realizado son manuales. Estas pruebas han consistido en el uso de la herramienta *Insomnia*, un cliente de peticiones HTTP que ha permitido realizar pruebas sobre los *endpoints* desarrollados.

Mediante *Insomnia* se creó una colección de peticiones que abarca los principales *endpoints* de la aplicación. Cada petición se configuró con los parámetros, cabeceras y cuerpos JSON necesarios, verificando la correcta respuesta del servidor y el código de estado devuelto por cada una. Esto ha permitido validar manualmente la API antes de la automatización de pruebas de otro tipo.

A continuación, en las imágenes de la figura 41 se muestran imágenes de la colección de solicitudes HTTP utilizadas en la herramienta. Asimismo, se muestra un ejemplo de solicitud completa hacia el servidor, con su configuración y su respuesta. Por último, se muestra un ejemplo de solicitud con un caso negativo para comprobar la validación de *pipes* implementados en el *backend*.

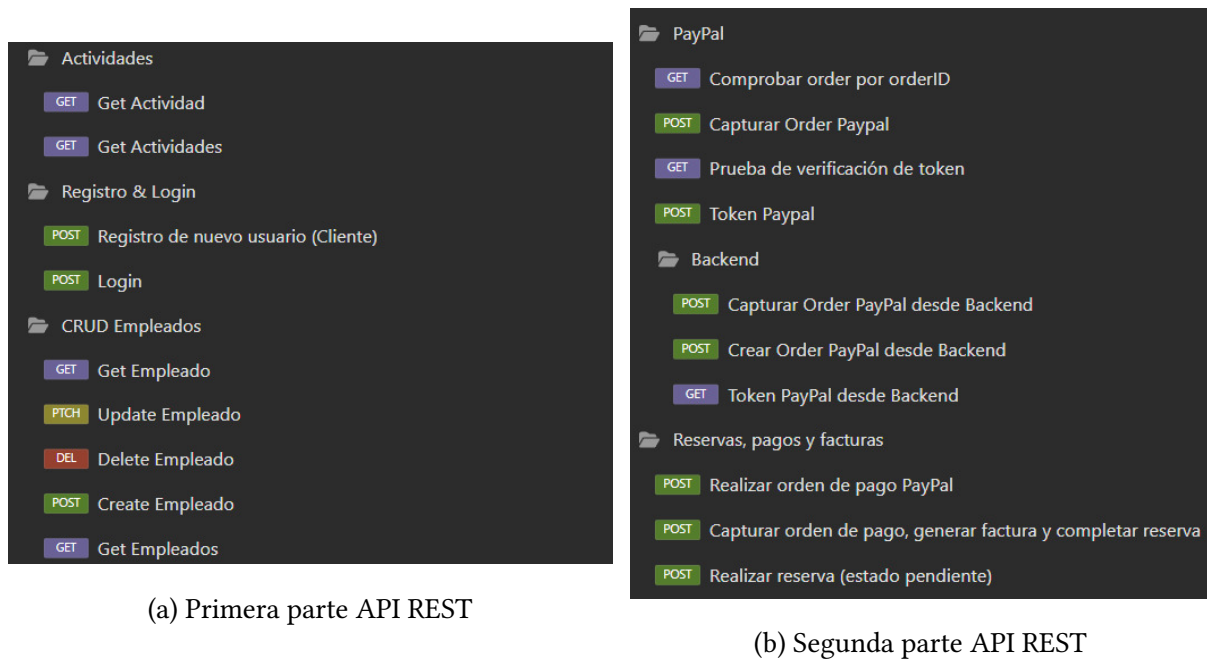


Figura 41: Colección de solicitudes HTTP utilizadas en *Insomnia* para validar el funcionamiento de la API REST desarrollada. Fuente: elaboración propia.

La mayor parte de solicitudes HTTP coinciden con los mostrados en la tabla 64 en el capítulo anterior. La principal diferencia es que en estas imágenes, se incluyen más *endpoints* debido a que para la implementación y desarrollo software de ciertas funcionalidades, hubo un proceso de aprendizaje en el que se incluían solicitudes a ciertos servicios y otro tipo de API, como la de PayPal *sandbox*.

Las principales llamadas HTTP utilizadas para los procesos de negocio más relevantes del caso práctico son todas las que aparecen en la primera imagen de la figura 41 y, la carpeta *Reservas, pagos y facturas* de la colección, presentes en la segunda imagen. El resto fueron parte del proceso de aprendizaje de implementación de PayPal. Otras peticiones como las de *Actividades* eran necesarias para el *frontend*.

Por último, se muestran dos peticiones de ejemplo en las figuras 42 y 43. La primera muestra una solicitud para obtener la lista de todos los empleados del sistema la cual requiere una cabecera de autorización con un *bearer token*, siendo este token el del responsable de la aplicación web. Esta solicitud devuelve la lista de empleados y un código de estado 200. En la segunda y última figura de este apartado, se muestra el intento de creación de un usuario cuyos datos únicos ya existen en el sistema y por lo tanto, es un conflicto, devolviendo la API un

código de estado 409.

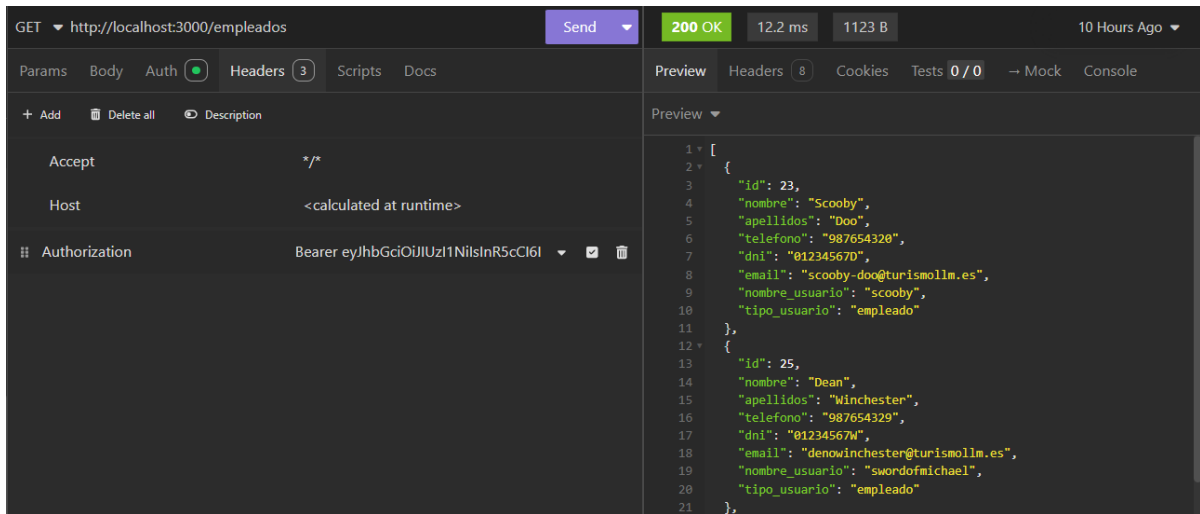


Figura 42: Ejemplo de solicitud HTTP correcta en *Insomnia*. Fuente: elaboración propia.

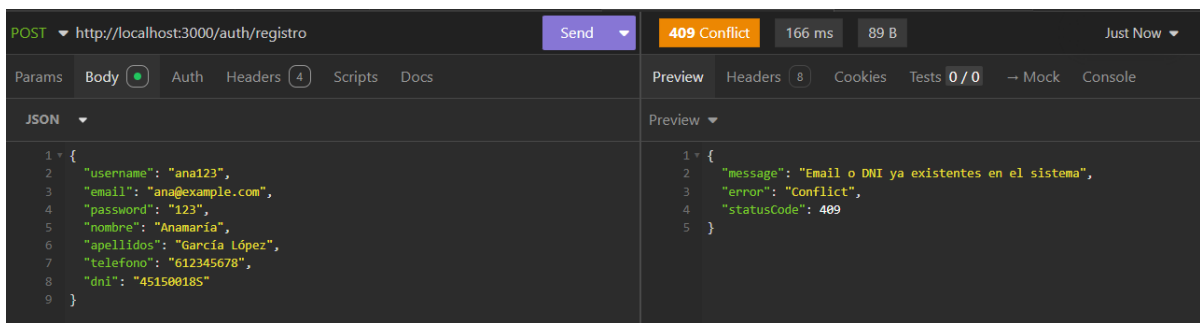


Figura 43: Ejemplo de solicitud HTTP correcta en *Insomnia*. Fuente: elaboración propia.

### 5.1.2. Pruebas E2E

Las pruebas *end-to-end* (E2E) son un tipo de pruebas software que verifican el flujo completo de una aplicación desde el punto de vista del usuario final, asegurando que todos los componentes de la aplicación web funcionen juntos como se espera. Prueban cómo responde la API y, por lo tanto, indirectamente comprueban la lógica implementada en el *backend*.

En concreto, validan que toda la aplicación funciona como un todo, simulando el comportamiento real de llamadas reales y de manera automática, comprobando que el sistema responde como debería.

Como se explicó en la introducción de este capítulo, se han realizado pruebas E2E en el backend de la aplicación. Las pruebas E2E realizadas han contemplado tanto casos positivos como negativos, de manera que no solo se ha validado el correcto funcionamiento de los flujos principales de la aplicación, sino también la gestión adecuada de errores y situaciones no válidas.

Finalmente, aunque las buenas prácticas aconsejan la utilización de una API REST y una base de datos distintas de la de desarrollo para pruebas, se ha optado por realizar la validación sobre el entorno de desarrollo, dado que se trata de un caso práctico ficticio.

Comenzando con las pruebas E2E desarrolladas, para su implementación se ha utilizado la herramienta **PactumJS** [36], que permite automatizar peticiones HTTP contra la API REST desarrollada en NestJS y comprobar que las respuestas y códigos de estado devueltos corresponden con lo esperado por la implementación del sistema. Estas pruebas se han diseñado con un enfoque de automatización, de manera que se puedan repetir y ejecutar de forma automática.

Por defecto, NestJS implementa **Jest** [37] para el desarrollo de pruebas unitarias y de integración, simulando módulos o levantando servicios, pero para lanzar peticiones HTTP reales contra el servidor, se decidió utilizar *PactumJS*.

Para comenzar a utilizar *PactumJS* es necesario realizar una configuración previa para implementar una simulación real del servidor, de manera que funcionen todos los módulos desarrollados e incluso sus validaciones de datos. En la siguiente imagen de la figura 44 se muestra la configuración realizada. Para *PactumJS* es necesario indicar la ruta del servidor y puertos a utilizar, es por ello que se extrae del archivo de entorno *dotenv* toda la URL de la

API.

```
dotenv.config();
const API_URL = process.env.API_URL || 'http://localhost:3000';
pactum.request.setBaseUrl(API_URL);

describe('TurismoLLM E2E', () => {
  let app: INestApplication;
  let prisma: PrismaService;

  beforeAll(async () => {
    const moduleRef = await Test.createTestingModule({
      imports: [AppModule],
    }).compile();

    app = moduleRef.createNestApplication();
    app.useGlobalPipes(
      new ValidationPipe({
        whitelist: true,
        transform: true,
      }),
    );

    await app.init();
    const port = Number(process.env.PORT) || 3000;
    await app.listen(port);
    prisma = app.get(PrismaService);
  });

  afterAll(() => {
    app.close();
  });
});
```

Figura 44: Configuración previa de PactumJS para pruebas E2E en *backend*. Fuente: elaboración propia.

Una vez configurado el servidor, se proceden a crear *suites* de *tests* en los que establecer una solicitud HTTP con cierta cabecera y cuerpo para comprobar el código de estado en la respuesta. Se muestra un ejemplo en la figura 45 en el que el responsable<sup>16</sup> de la aplicación web inicia sesión, almacenando su *access\_token*, útil para otras peticiones, como la gestión de empleados. Por último, métodos de PactumJS como *inspect* permiten ver en consola la salida de la llamada a la API, obteniendo un resultado en JSON.

---

<sup>16</sup>Todo dato sensible, como contraseñas, mostrados en este documento son datos ficticios

```
describe('login', () => {
  it('deberia hacer login de usuario responsable y devolver un código 200', () => {
    const dto: AuthDTO = {
      username: 'salvador',
      password: '123',
    };
    return pactum
      .spec()
      .post('/auth/login')
      .withHeaders('Content-Type', 'application/json')
      .withBody(dto)
      .expectStatus(200)
      .stores('token_priv', 'access_token')
      .inspect();
  });
});
```

Figura 45: Ejemplo de solicitud HTTP de prueba E2E con PactumJS. Fuente: elaboración propia.

El resto de *suites* y *tests* es similar, ajustando el código de estado de respuesta y las cabeceras y cuerpos.

Se ha desarrollado una *suite* que incluye un total de 22 *tests*. La *suite* principal contiene un *describe* por proceso de negocio más relevantes en los que se prueban ciertos escenarios para comprobar las respuestas de la API del *backend*.

A continuación, se procede a mostrar un resumen de las pruebas realizadas y los *endpoints* probados. Este resumen además de mostrar todos los escenarios probados, sirve para indicar lo que es necesario en cada petición HTTP realizada en las pruebas. Asimismo, se va a utilizar para la evaluación de esta fase. Por último, indicar que no se han tenido en cuenta todos los escenarios en el desarrollo manual y lo solicitado a los LLMs debido a que con mostrarlo al menos una vez, para peticiones similares y *endpoints* parecidos, no aportan mucho valor.

Endpoint	Método	Códigos esperados	Requisitos	Comentarios de respuestas
/auth/registro	POST	201, 400, 409	Cabecera y Body	201 si éxito, 400 si faltan datos, 409 si email o dni existentes en el sistema
/auth/login	POST	200, 400, 404	Cabecera y Body	200 si éxito en login y devuelve access_token, 400 si faltan datos, 404 no existe usuario en el sistema
/empleados	GET	200, 403	Cabecera y Token	200 si privilegios suficientes, 403 si autenticado aunque no autorizado
/empleados/:id	GET	200, 403, 404	Cabecera y Token	200 si privilegios suficientes, 403 si autenticado aunque no autorizado, 404 si empleado no existe en el sistema
/empleados/:id	PATCH	200, 403	Cabecera, Body y Token	200 si privilegios suficientes y actualizado empleado, 403 si no hay privilegios
/empleado/nuevo	POST	201, 403, 409	Cabecera, Body y Token	201 si privilegios suficientes y creado empleado, 409 si email o dni ya existen en el sistema, 403 si no tiene privilegios
/empleado/:id	DELETE	200, 404	Cabecera y Token	200 si privilegios suficientes y empleado eliminado, 404 si empleado no existe
/reservas/checkout	POST	201	Cabecera y Body	201 si reserva creada en estado pendiente y orden de pago PayPal creada y devuelve orderID
/pagos/orders/capture	POST	503	Cabecera, Body y orderID	503 si la orden de pago no se puede capturar a causa de pago no aceptado por comprador

Cuadro 90: Resumen de endpoints, estados esperados y requisitos realizados en el desarrollo manual para pruebas E2E. Fuente: elaboración propia.

## 5.2. Uso de LLMs

Para la generación de pruebas software de tipo E2E no es necesario utilizar en los *prompts* de entrada el código fuente completo de las funcionalidades software desarrolladas ya que para los LLMs no es necesario conocer la implementación para generar pruebas E2E debido a que estas tienen un carácter de **caja negra**. Lo único relevante para los LLMs es conocer por cada funcionalidad qué *endpoint* existe, qué recibe, qué devuelve, y qué códigos de estado de respuesta se esperan en cada caso. Por ello, es necesario indicar la **ruta** de cada *endpoint*, los **campos** de entrada **obligatorios** y sus **restricciones** (o hacerle conocer las validaciones que se realizan) y los **códigos** de **respuesta** esperados en cada escenario.

Al igual que en el resto de secciones de *Uso de LLMs* realizado en capítulos anteriores, se ha realizado un diagrama de estructura que contenga principalmente el uso que se ha dado a los LLMs y qué se ha tenido en cuenta en su uso para la evaluación de los mismos en esta fase.

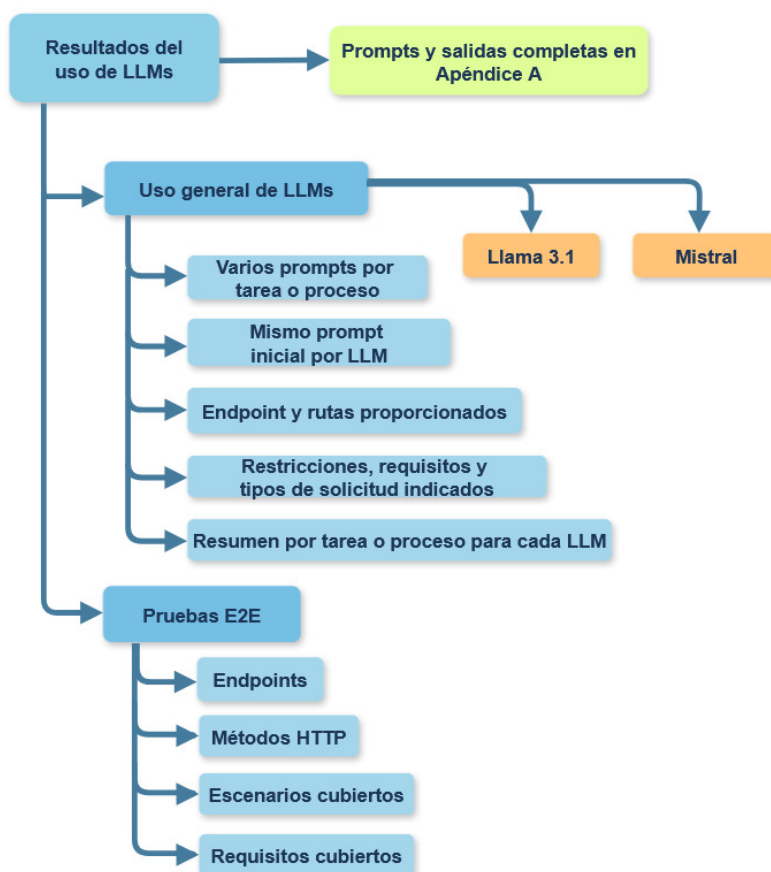


Figura 46: Diagrama de estructura del uso de LLMs en la fase 5. Fuente: elaboración propia.

La mayor parte de casos de prueba generados serán probados en el entorno de desarrollo utilizado por el actor humano. De esta forma se puede comprobar en tiempo real si lo generado por los LLMs sirve o requiere de corrección, necesitando otros aspectos como ingeniería de *prompts*. Sin embargo, en ocasiones, como para la implementación de software y pruebas de software se implementó contexto en las llamadas a la API del servidor donde están instalados los LLMs, hizo falta más de un *prompt* por lo que para algunas funcionalidades se corrigió la salida con al menos otro *prompt*. Las pruebas no fueron generadas todas en el mismo *prompt*, se fue solicitando a los LLMs ciertas pruebas dependiendo de la funcionalidad a probar.

Además, para esta fase, para la generación de pruebas software, se han proporcionado dos *prompts* adicionales y distintos de lo normal, el primero sirviendo para probar contexto y capacidad de corrección del modelo de IAG y, el segundo, para cambiar la forma en la que se solicitaba a los LLMs generar pruebas E2E, de forma que se evalúe la capacidad de comprensión del modelo.

Por último, en cada *prompt* fue necesario indicar que utilizara una configuración previa como la mostrada en la figura 44, ya que si no variaban mucho las salidas.

Para el uso de cada LLMs, se va a mostrar en una tabla resumen lo que ha generado cada modelo de IAG para las pruebas solicitadas. Las celdas vacías indican que el modelo no ha generado nada para esa columna. Asimismo, Mistral no genera todo el código para cada escenario.

A continuación, se muestra un resumen de los elementos principales generados por cada LLM:

Endpoint	Método	Estados cubiertos	Requisitos cubiertos
/auth/registro	-	201, 409 y 400	Cabecera y Body
/auth/login	-	200, 404 y 400	Cabecera, Body y Token almacenado
/empleados	GET	200 y 403	Cabecera y Token
/empleados/:id	GET	200, 403 y 404	Cabecera y Token
/empleados/:id	PATCH	200, 403 y 404	Cabecera, Body y Token
/empleado/nuevo	POST	201, 403 y 409	Cabecera, Body y Token
/empleado/:id	DELETE	200, 403 y 404	Cabecera y Token
/reservas/checkout	POST	201	Cabecera y Body, orderID no incluido
/pagos/orders/capture	POST	503	Cabecera y Body, orderID no incluido

Cuadro 91: Resumen de elementos de la generación de pruebas E2E de *endpoints* por Llama 3.1. Fuente. Llama 3.1.

Endpoint	Método	Estados cubiertos	Requisitos cubiertos
/auth/registro	POST	201, 409 y 400	Cabecera y Body
/auth/login	POST	200, 404 y 400	Cabecera, Body y Token almacenado
/empleados	GET	200, 403	Cabecera y Token
/empleados/:id	GET	200, 403 y 404	Cabecera y Token
/empleados/:id	PATCH	200, 403 y 404	Cabecera, Body y Token
/empleados/nuevo	POST	201, 403 y 409	Cabecera, Body y Token
/empleados/:id	DELETE	200, 403 y 404	Cabecera y Token
/reservas/checkout	POST	201	Cabecera, Body y orderID almacenado
/pagos/orders/capture	POST	503	Cabecera, Body y orderID

Cuadro 92: Resumen de elementos de la generación de pruebas E2E de *endpoints* por Mistral. Fuente. Mistral.

### 5.3. Evaluación de resultados

Esta sección final recoge la aplicación de las métricas utilizadas a lo largo del proyecto para evaluar el desempeño de los LLMs en el proceso de desarrollo software. En concreto, esta parte se centra en evaluar los LLMs en la generación de pruebas, concretamente de tipo E2E.

#### 5.3.1. Completitud

Para aplicar la completitud en las pruebas E2E generada por los modelos, se ha tenido en cuenta el desarrollo manual de referencia. Para esta métrica, se mide la cobertura, sin importar el número de elementos. Sin embargo, si el escenario no cubre algún estado de respuesta, utiliza incorrectamente un método HTTP o no lo incluye, la ruta del *endpoint* no es válida o no incluye todos los requisitos necesarios como cabeceras, cuerpos, entre otros, no se contabiliza para el cálculo. En resumen, no se evalúa la sintaxis respecto al desarrollo manual, sino los códigos de respuesta cubiertos en cada escenario de pruebas, habiendo hecho uso de los *requisitos* mínimos mostrados en las tablas anteriores.

Como referencia del desarrollo manual se va a utilizar la lista total de escenarios de prueba desarrollados el cual se va a utilizar como denominador. Para el numerador, se toma el conjunto de pruebas que el LLM generó y que cubre los escenarios de los *endpoints*. En otras palabras, no importa si el código generado es diferente, lo relevante es si cubre las condiciones de cada escenario. En caso de cubrirlas, se tienen en cuenta. Para esta métrica, no se tienen en cuenta elementos adicionales generados por los LLMs que no se hayan tenido en cuenta en la referencia.

Por ello, del desarrollo manual se extrae un total de **20** escenarios de prueba. No se han tenido en cuenta escenarios repetidos o muy similares dentro de la generación de pruebas para el mismo proceso de negocio relevante.

Para el caso de **Llama 3.1**, se obtiene la siguiente completitud en la generación de pruebas respecto al desarrollo manual:

$$\text{Completitud PR\_E2E} = \frac{12}{20} \times 100 = 60 \%$$

Para Llama, no se han contabilizado los dos primeros y dos últimos escenarios de los *endpoints* debido a que carecían de método HTTP o no incluían variables necesarias para las

peticiones.

Finalmente, para el caso de **Mistral** se ha obtenido el siguiente resultado:

$$\text{Compleitud PR\_E2E} = \frac{14}{20} \times 100 = 70\%$$

Mistral generó y cubrió correctamente 14 códigos de respuesta, repartidos entre distintos escenarios.

### 5.3.2. Corrección

Las categorías de corrección que se utilizan al aplicar dicha métrica, son las mostradas en la sección 2.3. Se ha analizado todo el contenido generado por los LLMs para esta fase, además, parte de la corrección se ha realizado haciendo uso de las tablas 5.1.2 y 5.2. Se van a utilizar los siguientes criterios de clasificación para la corrección de las pruebas generadas:

Categoría	Ruta	Escenarios	Aserciones	Variables
Incorrecto - 1	No se genera el método HTTP o se usa otro distinto al indicado.	No se cubre ningún escenario requerido.	No hay comprobaciones o son erróneas.	No se emplean variables.
Parc. correcto - 2	Método correcto pero ruta ligeramente distinta.	Se cubre alguno de los escenarios indicados, pero no todos.	Solo se valida alguna comprobación mínima (ej. status).	Se usan variables estáticas o almacenadas sin darles continuidad.
Correcto - 3	Ruta y método válidos como en el desarrollo manual.	Se cubren todos los escenarios requeridos.	Se valida cabeceras, cuerpo y otros requisitos principales.	Se obtienen y reutilizan tokens o identificadores en el flujo.
Excelente - 4	Además de ser correcto, aporta métodos distintivos o mejoras en las solicitudes.	Se cubren todos los escenarios requeridos y además escenarios extra válidos.	Se comprueban cabeceras y cuerpos en todos los escenarios, con aserciones extra.	Además de reutilizar variables, se generan datos parametrizados y aislados, con extras.

Cuadro 93: Criterios de corrección para los LLMs en la fase de pruebas. Fuente: elaboración propia.

Se va a mostrar el resultado total de la corrección obtenida por cada LLM respecto al desarrollo manual en esta fase.

Endpoint	Ruta	Escenarios cubiertos	Aserciones	Uso de variables
/auth/registro	Incorrecto - 1	Excelente - 4	Correcto - 3	-
/auth/login	Incorrecto - 1	Correcto - 3	Correcto - 3	Parc. correcto - 2
/empleados	Correcto - 3	Correcto - 3	Correcto - 3	Parc. correcto - 2
/empleados/:id	Correcto - 3	Correcto - 3	Correcto - 3	Parc. correcto - 2
/empleados/:id	Correcto - 3	Excelente - 4	Correcto - 3	Parc. correcto - 2
/empleados/nuevo	Correcto - 3	Correcto - 3	Correcto - 3	Incorrecto - 1
/empleados/:id	Correcto - 3	Excelente - 4	Correcto - 3	Incorrecto - 1
/reservas/checkout	Correcto - 3	Correcto - 3	Correcto - 3	Incorrecto - 1
/pagos/orders/capture	Correcto - 3	Correcto - 3	Parc. correcto - 2	Incorrecto - 1

Cuadro 94: Resumen de la corrección de las pruebas E2E generadas por Llama 3.1. Fuente: elaboración propia.

Endpoint	Ruta	Escenarios cubiertos	Aserciones	Uso de variables
/auth/registro	Correcto - 3	Excelente - 4	Correcto - 3	-
/auth/login	Correcto - 3	Correcto - 3	Correcto - 3	Correcto - 3
/empleados	Correcto - 3	Correcto - 3	Correcto - 3	Correcto - 3
/empleados/:id	Correcto - 3	Correcto - 3	Parc. correcto - 2	Correcto - 3
/empleados/:id	Correcto - 3	Correcto - 3	Correcto - 3	Correcto - 3
/empleados/nuevo	Correcto - 3	Correcto - 3	Correcto - 3	Correcto - 3
/empleados/:id	Correcto - 3	Correcto - 3	Correcto - 3	Correcto - 3
/reservas/checkout	Correcto - 3	Correcto - 3	Correcto - 3	Correcto - 3
/pagos/orders/capture	Correcto - 3	Correcto - 3	Correcto - 3	Correcto - 3

Cuadro 95: Resumen de la corrección de las pruebas E2E generadas por Mistral. Fuente: elaboración propia.

### 5.3.3. Rendimiento

Los tiempos obtenidos en el proceso de desarrollo y generación de pruebas software son los siguientes:

FASE 5			
Actividad	Tiempo	Tiempo Llama	Tiempo Mistral
Pruebas de software	289 minutos	71.21 segundos	43.28 segundos

### 5.3.4. Evaluación de resultados

Para el reporte de resultados de esta fase, se va a comenzar con la **completitud** obtenida por ambos modelos.

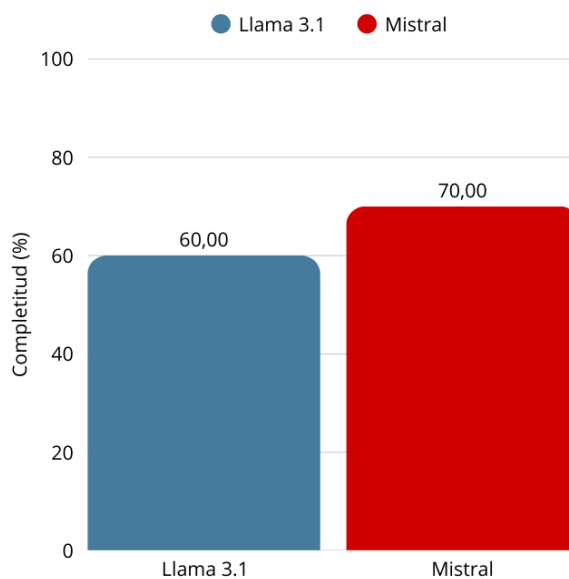


Figura 47: Comparación visual de resultados en la completitud de diagramas de secuencia respecto al desarrollo manual. Fuente: elaboración propia.

Continuando con la **corrección** de las pruebas realizadas por los LLMs, se obtiene el siguiente reparto de categorías:

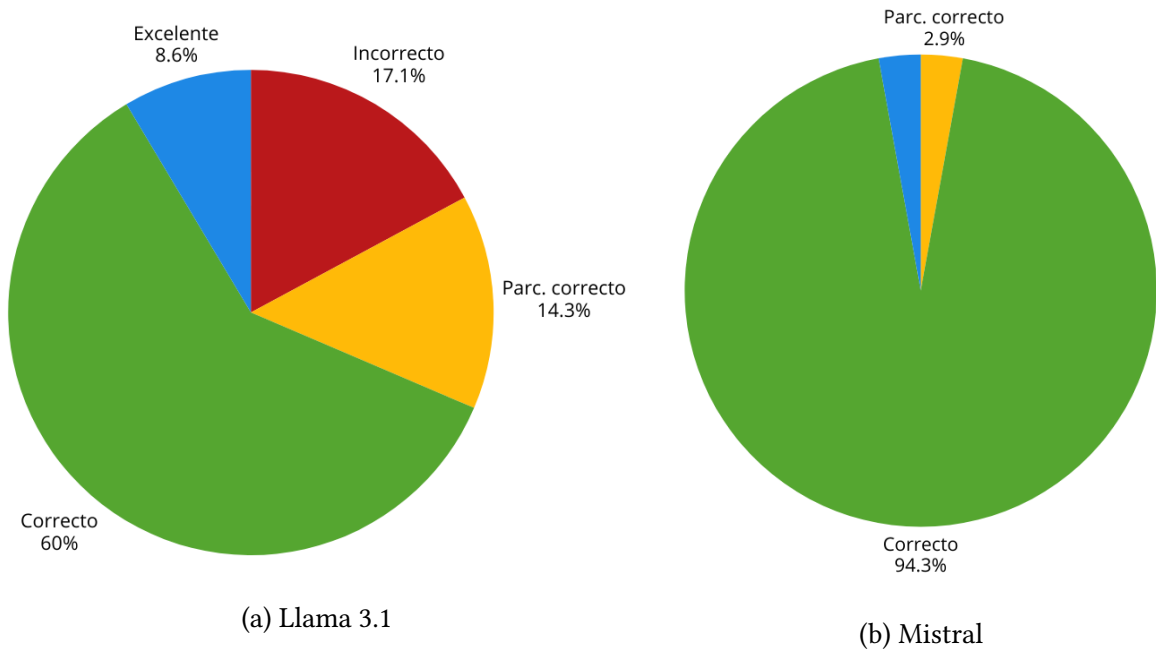


Figura 48: Distribución de categorías de corrección de pruebas E2E obtenidas por cada modelo.  
Fuente: elaboración propia.

# 6

## Evaluación global

En este capítulo se pretende finalizar la evaluación del uso de LLMs en el proceso de desarrollo software a través de la interpretación de los resultados obtenidos en cada fase.

Los datos son los mostrados en los apartados de *Evaluación de resultados* al final de los capítulos 3, 4 y 5. La información se presentó en distintas formas, en gráficos de barras, gráficos de barras agrupadas y por último, gráficos circulares. Estos gráficos, principalmente muestran los resultados obtenidos por cada modelo tras aplicar las métricas de **completitud** y **corrección**.

Este capítulo permitirá sacar conclusiones que ayuden a comprender mejor el trabajo desarrollado a lo largo del proyecto. Asimismo, se muestra que, por cada fase, se ha asignado una calificación a los distintos LLMs utilizados, con el objetivo de obtener una visión global de su desempeño en las diferentes tareas del proceso de desarrollo de software.

A continuación, se presenta un reporte de la evaluación final del uso de LLMs en el proceso de desarrollo software:

### 6.1. Resultados por fase

En esta sección se recogen los resultados de los gráficos mostrados al final del capítulo de cada fase.

Para las fases referentes a la identificación y especificación de requisitos, modelado del sistema y diseño y prototipado, se ha agrupado en la misma parte. Las fases de implementación y desarrollo software, y pruebas software, se han separado en dos secciones distintas.

Además de mostrar el desempeño de cada LLM en dichas fases, se señalan fortalezas, debilidades y diferencias relevantes entre las salidas generadas por los LLMs.

Antes de comenzar con las fases, se recuerda el objetivo de aplicar cada métrica, aunque la interpretación se ha orientado especialmente a la comparación del uso de distintos LLMs gratuitos. Para el caso de la **completitud**, se ha evaluado el grado de coincidencia del conte-

nido generado por los LLMs, con la solución desarrollada manualmente ya que de esta forma, se realiza una comparación entre el desarrollo de soluciones mediante el uso de IA generativa con el desarrollo realizado por una persona y, que además permite contrastar la capacidad de cada LLM seleccionado de proporcionar soluciones acertadas y de alta calidad. La **corrección** evalúa todo el contenido generado por los modelos, valorando si los elementos y fragmentos generados son funcionales, distribuyendo cada elemento generado en cuatro categorías.

### 6.1.1. Análisis y Diseño

#### Fase 1

Comenzando con la primera fase la cual involucra las tareas de análisis, especificación y obtención de los distintos tipos de requisitos y, casos de uso y su especificación.

Como se muestra en la figura 17, ambos LLMs obtienen unos valores muy similares en la generación de **requisitos funcionales**, alrededor del 60 %, no presentando una ventaja significativa el uso de un LLM u otro. Asimismo, la generación para este tipo de tareas no proporciona una cobertura completa, mostrando la necesidad de asistencia humana para lograr mejores resultados. Continuando con los **requisitos no funcionales**, Llama 3.1 muestra un mejor resultado en la identificación de este tipo de requisitos ya que Mistral no aportó resultados significativos, aunque en esta tarea, Llama 3.1 solo obtuvo un 56 % de cobertura respecto al desarrollo humano, presentando de nuevo la necesidad de asistencia humana para elaborar salidas más detalladas y cercanas a soluciones viables.

Los resultados obtenidos en la **corrección**, figura 20, del **catálogo de requisitos funcionales** generado por cada LLM es que ambos presentan el mismo porcentaje de contenido generado incorrecto, pero Llama 3.1 presenta un contenido irregular en su salida según la distribución de las categorías, mientras que Mistral muestra una salida más consistente. En resumen, los datos obtenidos permiten observar que Llama 3.1 tiene potencial para alcanzar casos de mayor calidad puntual debido a la presencia de elementos *excelentes* en su salida, Mistral ofrece un desempeño más estable y confiable. En el caso de los **catálogo de requisitos no funcionales**, Mistral no logró generar ninguna salida significativa por lo que no se aplicó esta métrica tras obtener un 0 % de resultados en la primera métrica. En este tipo de requisito, Llama 3.1 obtuvo en proporción un número elevado de elementos generados *incorrectos*, lo que destaca una necesidad de mejora en sus salidas.

Por otro lado, la figura 17 muestra unos valores muy elevados en la **completitud de casos de uso** para ambos modelos, presentando un desempeño elevado y similar. En este gráfico, se destaca la cobertura de al menos el 90 % de elementos generados en la tarea de generación de actores y casos de uso. Estos resultados evidencian que el desempeño de cada LLM depende de la fase del proceso de desarrollo software en la que se utilice y, posiblemente, de la entrada proporcionada. Sin embargo, la *completitud* de la **especificación de casos de uso** muestra resultados bastante bajos en ambos modelos respecto al desarrollo de referencia. En esta tarea, Llama 3.1 obtiene el peor resultado de los dos modelos, aunque obtienen un valor cercano.

Finalmente, en esta fase, para la **corrección de actores y casos de uso**, según lo mostrado en la figura 22, Llama 3.1 genera elementos consistentes en su salida y más acertados que Mistral, aunque este último genera algunos elementos de alta calidad, pero con mayor porcentaje de elementos incorrectos. Para la corrección de la **especificación de casos de uso**, ambos modelos obtienen una distribución de categorías muy similares, aunque Llama 3.1 obtiene más elementos *correctos*.

En resumen, la **completitud** recoge resultados cercanos y similares en la mayor parte de las tareas, mientras que la **corrección** al tener en cuenta todos los elementos generados, presenta para cada tarea una distribución de categorías consistente en *algunas* de las tareas, pero dependiendo de cual, destacan ciertos elementos *correctos* y *excelentes* de un LLM sobre el otro.

## Fase 2

En esta fase se ha evaluado el uso de LLMs para generar **diagramas de secuencia**, haciendo énfasis en el contenido.

En materia de **completitud**, según lo mostrado en la figura 18, a pesar de que se obtengan resultados muy cercanos, Mistral destaca en la cobertura de esta tarea respecto al desarrollo manual de referencia, con aproximadamente un 90 % obtenido.

Además, en cuanto a la **corrección**, como se muestra en la figura 24, Mistral obtiene mayor porcentaje de elementos *correctos* y en general, una generación de contenido consistente, a diferencia de Llama 3.1 el cual a pesar de generar un porcentaje mayor de elementos *excelentes* mayor que Mistral, también genera más contenido *incorrecto*.

## Fase 3

En esta fase se realizan las tareas de desarrollo de **maquetas de pantalla**.

La aplicación de las métricas en el contenido generado por los LLMs, muestra lo siguiente. Como se aprecia en la figura 19 sobre la **completitud**, Mistral cubre alrededor de un 22 % más que Llama 3.1, obteniendo un valor de casi el 91 % de cobertura total respecto al desarrollo de referencia. Esta diferencia destaca el uso de Mistral para la generación de formatos de pantalla y su acierto, mostrando no requerir asistencia humana para ello. Asimismo, en la **corrección de maquetas de pantalla**, Mistral vuelve a destacar en la generación de contenido consistente y además, presenta un mayor porcentaje de elementos *excelentes* en su salida en comparación a Llama 3.1 el cual también presenta cierto porcentaje de elementos **excelentes**, pero también muestra mayor porcentaje en su salida de elementos *parcialmente correctos e incorrectos*.

Respecto a todas las fases referentes al **Análisis y Diseño**, observando la recopilación de tiempo total dedicado por cada LLM para obtener cada tarea, muestra que Mistral ha requerido menos tiempo para generar las salidas, excepto en el catálogo de requisitos funcionales y no funcionales. A pesar de esta diferencia, si se compara con el tiempo que tarda una persona en desarrollar estas tareas, se obtiene una diferencia de tiempo muy grande, destacando la capacidad de la inteligencia artificial para generar soluciones válidas sin necesidad de asistencia humana hasta cierto punto. Además, esta diferencia entre el tiempo dedicado en el desarrollo que realiza una persona y el tiempo que tarda la IA, permite destacar la importancia de LLMs para aumentar la productividad en el proceso de desarrollo software mediante el uso de IA en las tareas de las distintas fases.

### 6.1.2. Implementación

En esta fase interviene dos tareas principales; diseño e implementación de bases de datos y desarrollo de software.

El uso de LLMs en la **implementación de bases de datos** se muestra en la Figura 33, donde se compara la cantidad total de elementos generados por Llama 3.1 y Mistral frente a la implementación de referencia. El gráfico muestra la ponderación aplicada a tablas, relaciones y atributos, lo que permite apreciar el nivel de detalle alcanzado por cada modelo en relación con la solución manual. Respecto a las salidas proporcionadas por ambos modelos, el contenido generado es cercano al desarrollo de referencia, excepto en la generación de atributos, donde no se logra detallar todos los elementos. Ahora, comenzando con la **completitud** en esta tarea, ambos LLMs muestran haber obtenido un valor cercano, aunque Llama 3.1 alcanza el valor más

alto, con aproximadamente un 62 %.

En las figuras 36, 37 y 38, se muestra la distribución y recuento de elementos generados por categoría de **corrección**, representado en la figura 39 los resultados obtenidos por cada modelo. En esta tarea, ambos LLMs generan contenido muy similar, aunque Llama 3.1 es algo más consistente en ello, ya que Mistral proporciona más elementos *incorrectos* y *excelentes*, mientras que Llama 3.1, no. Sin embargo, la única diferencia en la categoría de elementos *correctos* es aproximadamente de un 1.4 %.

Finalmente, la **completitud** de los modelos de IAG en la **generación de código** muestran resultados altos y muy cercanos entre sí. Respecto al desarrollo de referencia, proporcionaron salidas aceptables y viables, no requiriendo casi intervención humana para corregir sus salidas, aunque se proporcionó bastante contexto en las entradas. Por ello, destaca Llama 3.1. en la cobertura total de código respecto a la solución de referencia, con aproximadamente un 6 % más que Mistral. En cuanto a la **corrección**, ambos LLMs son muy similares en los resultados obtenidos por cada categoría, aunque Llama 3.1. vuelve a destacar por generar mayor número de elementos *excelentes*, *correctos* y menos elementos *incorrectos* que Mistral.

### 6.1.3. Pruebas de software

La **completitud** obtenida por cada LLM en la fase de **pruebas de software**, especialmente pruebas E2E o de *caja negra*, muestra en la figura 47 que ambos LLMs obtienen resultados notables en la generación de sus elementos, pero respecto al desarrollo de referencia, Mistral obtiene una mayor cobertura que Llama 3.1., en un 10 %. Asimismo, en cuanto a la **corrección**, las salidas proporcionadas por Mistral son muy consistentes en comparación a las de Llama 3.1, destacando Mistral en no haber presentado elementos *incorrectos*. A pesar de esto, Llama 3.1 obtiene un mayor porcentaje de elementos *excelentes*.

Los resultados de **rendimiento** en las fases 4 y 5, muestran que el tiempo requerido por un desarrollador humano para completar las tareas de dichas fases es sustancialmente mayor en comparación con los LLMs, que generan salidas en cuestión de segundos. Esta diferencia pone en ventaja la capacidad de los modelos para reducir drásticamente el tiempo de desarrollo de ciertas tareas. Entre los modelos evaluados, Mistral tiende a ser más rápido en la mayoría de las tareas, aunque la diferencia es de pocos segundos respecto a Llama 3.1. No obstante, la velocidad por sí sola no garantiza calidad, por lo que se ha mostrado en las otras métricas,

aunque los resultados sean aceptables, aún es necesaria la asistencia humana para desarrollo de las actividades de las tareas típicas.

## 6.2. Evaluación global

Finalmente, en este apartado del capítulo se muestra cómo se ha asignado una calificación a cada LLM en las distintas fases. De esta forma, se pretende mostrar en un **diagrama de radar** una comparación entre ambos LLMs en las diferentes fases.

Para asignar una calificación a cada LLM, se va a desglosar en dos partes. En primer lugar, se van a tener en cuenta los resultados obtenidos por cada LLM en la **completitud** y **corrección** de cada tarea. Por ello, debido a que la *corrección* se distribuye en cuatro categorías, se va a establecer una ponderación de pesos a cada tipo de categoría. Para la *completitud*, se va a coger directamente el valor obtenido por cada LLM, sin realizar cálculos adicionales. Una vez realizados los cálculos, en cada fase se va a establecer una ponderación a cada métrica entre las distintas tareas, ya que de esta forma, se obtendrá la calificación de cada LLM en dicha tarea. En segundo lugar, como una fase puede tener más de una tarea, se calcula la media de las notas y, finalmente, se obtiene la nota de cada LLM en dicha fase.

A continuación, se muestran las fórmulas y ponderaciones que se han utilizado para asignar una calificación a cada LLM en las distintas tareas y fases.

	<b>Incorrecto</b>	<b>Parc. correcto</b>	<b>Correcto</b>	<b>Excelente</b>
<b>Peso</b>	0	40	80	100

Cuadro 96: Tabla de **ponderación** de categorías de **corrección** para el cálculo de la calificación por tareas en cada fase para cada LLM. Fuente: elaboración propia.

Para el valor de la *corrección*, se va a dar en una escala decimal. Para los elementos *incorrectos* se ha decidido que no contabilizan en el cálculo, mientras que los parcialmente correctos, tienen menor peso asignado. El resto de pesos se puede observar en la tabla. A partir de esto, se obtiene la siguiente fórmula para calcular el valor de la corrección de cada LLM por tarea:

$$\text{Valor\_Correccion} = (\%E \times 100) + (\%C \times 80) + (\%P \times 40) + (\%I \times 0)$$

Continuando con la ponderación, los pesos que se van a asignar a cada métrica en cada tarea es equivalente, es decir, se consideran igual de importantes los resultados de la *completitud* y *corrección* por lo que el peso de cada una es 0.5.

Teniendo en cuenta el **valor de la corrección** y la **completitud** obtenida por un LLM en una tarea de una fase específica, se obtiene el siguiente cálculo:

$$\text{Nota\_Tarea} = 0,5 \times (\text{Completitud} + \text{Valor\_Correccion})$$

Esta última fórmula se aplica en cada LLM. Una vez obtenido esto, se calcula la media de las notas obtenidas en cada tarea por LLM ya que de esta forma se obtiene una nota de la **fase** y, por último, se realiza el cálculo de una nota global por LLM en la que se realiza la media de notas de fase.

$$\text{Nota\_Fase} = \frac{\sum \text{Nota\_Tarea}}{\text{n}^\circ \text{ de tareas}}$$

$$\text{Nota\_Global} = \frac{\sum \text{Nota\_Fase}}{\text{n}^\circ \text{ de fases}}$$

Teniendo estas fórmulas en cuenta, se procede a mostrar una serie de tablas recopilatorias de *completitud* y *valor de la corrección* por LLM en cada tarea de cada fase. Seguido de esto, se muestra la nota de cada fase y, finalmente, la nota global.

Tareas	Llama 3.1		Mistral	
	Completitud	V. Corrección	Completitud	V. Corrección
Requisitos funcionales	60	41.6	64	44.8
Requisitos no funcionales	56	42.18	0	0
Actores y casos de uso	93	73.6	90	68
Especificación de casos de uso	44	68.34	51	64.94

Cuadro 97: Valores de las tareas por cada LLM en la fase 1. Fuente: elaboración propia.

Ahora, con la fase 2 y 3 se obtienen las siguientes notas por tarea:

Tareas	Llama 3.1		Mistral	
	Compleitud	V. Corrección	Compleitud	V. Corrección
Diagramas de secuencia	79.71	62.02	89.95	72.18
Formatos de pantalla	68.49	70.04	90.41	78.64

Cuadro 98: Valores de las tareas por cada LLM en la fase 2 y 3. Fuente: elaboración propia.

Y, finalmente, en las fases 4 y 5 se obtienen los siguientes resultados en las tareas por cada LLM:

Tareas	Llama 3.1		Mistral	
	Compleitud	V. Corrección	Compleitud	V. Corrección
Implementación de BBDD	61.76	59.2	54.41	58.36
Implementación de software	75.21	58.86	69.42	51.68
Pruebas de software	60	62.32	70	79.5

Cuadro 99: Valores de las tareas por cada LLM en la fase 4 y 5. Fuente: elaboración propia.

Una vez calculado estos valores, se procede a utilizar el resto de fórmulas.

Tareas	Nota por cada tarea	
	Llama 3.1	Mistral
Requisitos funcionales	50.8	54.4
Requisitos no funcionales	49.09	0
Actores y casos de uso	83.3	79
Especificación de casos de uso	56.17	57.97
Diagramas de secuencia	70.86	81.06
Formatos de pantalla	69.27	84.52
Implementación de BBDD	60.48	56.38
Implementación de software	67.03	60.55
Pruebas de software	61.16	74.75

Cuadro 100: Notas por tarea obtenidas por Llama 3.1 y Mistral. Fuente: elaboración propia.

Ahora, se procede a calcular la nota por fase, teniendo en cuenta los resultados obtenidos en la tabla anterior.

Fase	Num. tareas	Nota_Fase	
		Llama 3.1	Mistral
Fase 1	4	59.84	47.84
Fase 2	1	70.86	81.06
Fase 3	1	69.27	84.52
Fase 4	2	63.76	58.46
Fase 5	1	61.16	74.75

Cuadro 101: Calificación por fase para Llama 3.1 y Mistral. Fuente: elaboración propia.

En las fases con solo una tarea, la nota de la tarea es igual a la nota de la fase. Por último, se calcula la nota global de cada LLM:

Nota global	Num. fases	Llama 3.1	Mistral
Calificación LLM	5	64.98	69.33

Cuadro 102: Calificación global de cada LLM. Fuente: elaboración propia.

Adicionalmente, para mostrar una representación de lo obtenido por cada modelo en materia de *completitud* y *corrección*, se va a aplicar una fórmula extra para realizar dos diagramas distintos. Los datos son los calculados previamente y, mostrados en los cuadros 97, 98 y 99. La fórmula adicional es:

$$\text{Media\_MetricaFase} = \frac{\sum_{i=1}^n x_i}{n}, \quad \text{donde } n \text{ es el número de tareas de una fase}$$

Esta fórmula sirve tanto la *completitud* como para la *corrección*.

Como se ha comentado, antes de mostrar la calificación global de los LLMs en las distintas fases, se representan en dos diagramas de radar distintos los resultados obtenidos por cada LLM tras aplicar las métricas de *completitud* y *corrección*. A continuación, se muestran los valores de la media de la **completitud** por cada LLM en las distintas fases. Posteriormente, se representa el diagrama de radar de la *completitud* obtenida por ambos modelos en dichas fases:

Media_MetricaFase	Fase 1	Fase 2	Fase 3	Fase 4	Fase 5
Llama 3.1	63.25	79.71	68.49	68.49	60
Mistral	51.25	89.95	90.41	61.92	70

Cuadro 103: Media de la métrica de completitud obtenida por cada LLM en las distintas fases. Fuente: elaboración propia.

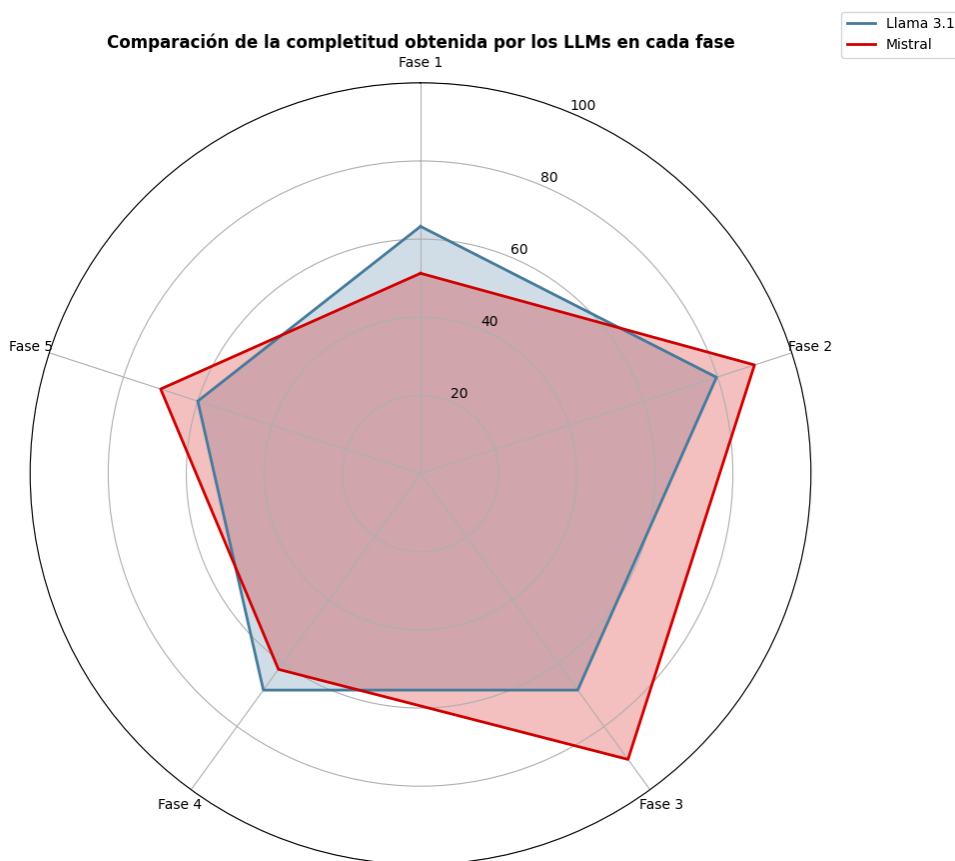


Figura 49: Comparación de la completitud obtenida por los LLMs en cada fase. Fuente: elaboración propia.

Asimismo, al igual que se ha mostrado un diagrama de radar que represente la *completitud* obtenida por cada LLM en las diferentes fases, se muestra a continuación la **corrección** obtenida por los LLMs en las fases del proceso de desarrollo software:

Media_MetricaFase	Fase 1	Fase 2	Fase 3	Fase 4	Fase 5
Llama 3.1	56.43	62.02	70.04	59.03	62.32
Mistral	44.44	72.18	78.64	55.02	79.5

Cuadro 104: Media de la métrica de corrección obtenida por cada LLM en las distintas fases.

Fuente: elaboración propia.

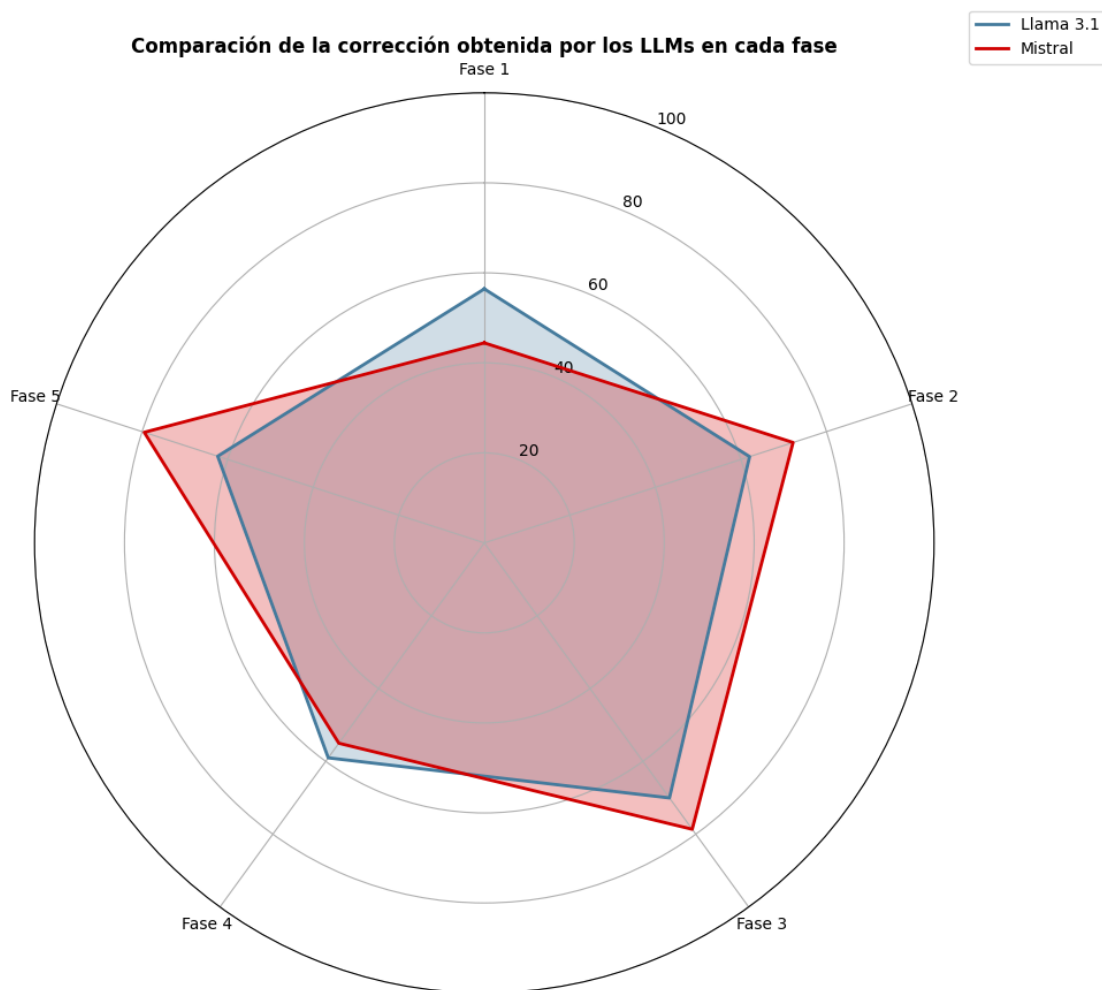


Figura 50: Comparación de la corrección obtenida por los LLMs en cada fase. Fuente: elaboración propia.

Una vez calculadas todas las calificaciones, se dispone a representar estos datos de notas globales por LLM en cada fase, en un diagrama de radar. Además, más adelante en este mismo apartado, se muestra la calificación obtenida por cada LLM en las distintas tareas de las fases del proceso de desarrollo software.

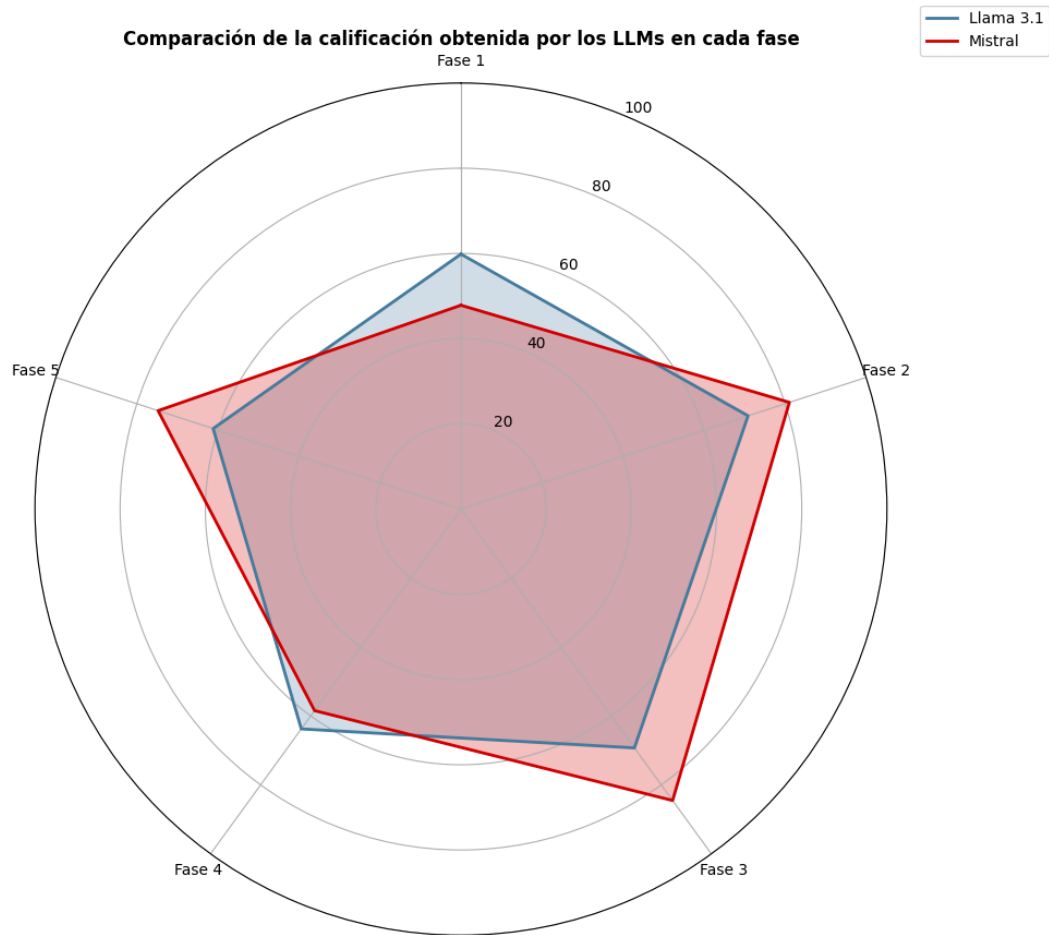


Figura 51: Comparación de la calificación obtenida por los LLMs en cada fase. Fuente: elaboración propia.

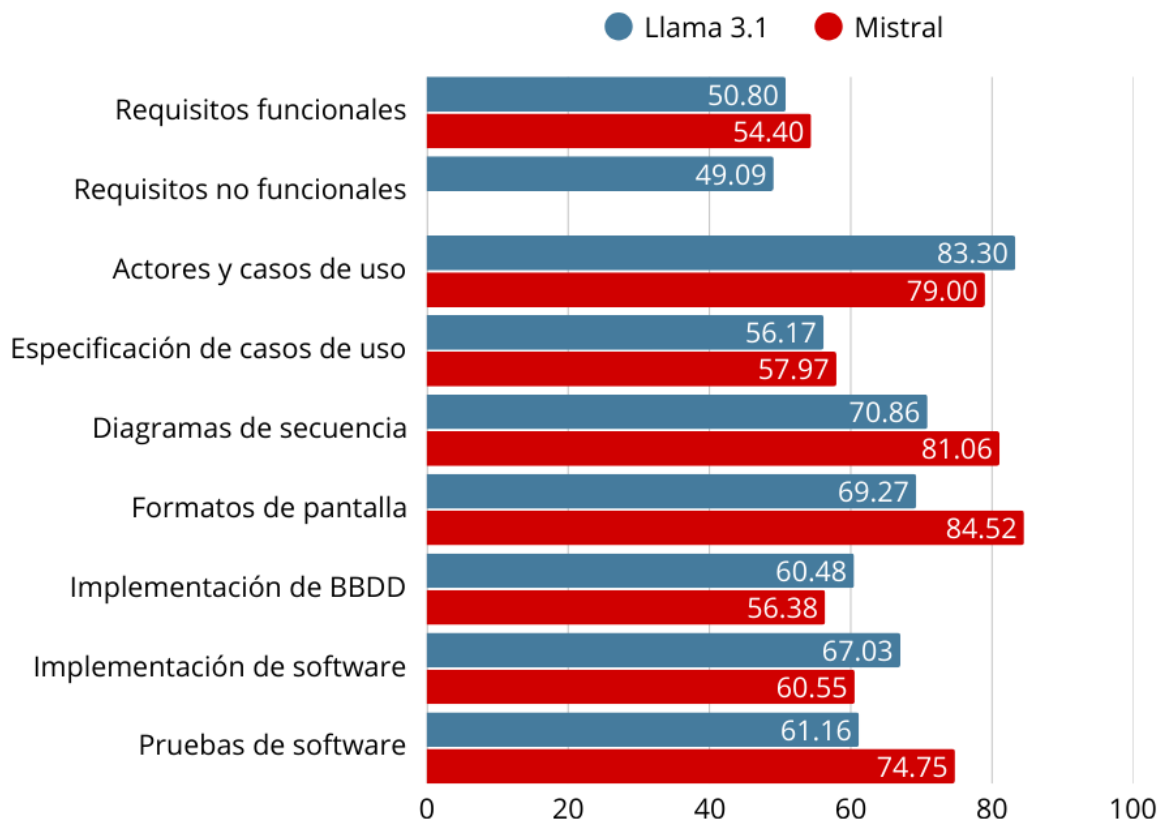


Figura 52: Comparación de la calificación obtenida por los LLMs en las tareas de cada fase del proceso de desarrollo software. Fuente: elaboración propia.



# 7

## Conclusiones y Líneas Futuras

En este capítulo se recogen las conclusiones del proyecto. Se abordan diferentes conclusiones desde distintas perspectivas. Asimismo, también se hace una recopilación de las líneas futuras que se identifican tras la realización de este trabajo.

### 7.1. Conclusiones

En primer lugar, para la realización de este proyecto se ha requerido gran parte de los conocimientos adquiridos durante el grado de Ingeniería Informática. La realización de este trabajo ha implicado el desarrollo de tareas correspondientes a distintas fases del proceso de desarrollo software, lo que ha favorecido una visión global e integrada de los contenidos aprendidos.

Este proyecto también ha permitido repasar conceptos transversales de la carrera y aplicarlos a un caso práctico, lo que ha supuesto una experiencia de repaso y aprendizaje que ha contribuido a la y consolidación de competencias y conceptos. Asimismo, ha facilitado un acercamiento a los LLMs, permitiendo comprender mejor qué son los LLMs, cómo funcionan y, sobre todo, cuál puede ser su utilidad práctica dentro del ámbito de la Ingeniería Informática.

Otro de los hallazgos más relevantes tras la realización de este proyecto es que el uso de inteligencia artificial generativa contribuye a un aumento significativo de la productividad. La capacidad de los LLMs para generar propuestas y respuestas en cuestión de segundos, en comparación con el tiempo que requiere el desarrollo humano, supone una mejora sustancial en la eficiencia de determinadas tareas.

De manera personal, se ha comprobado que la IA generativa muestra un rendimiento especialmente prometedor en fases como el Análisis y el Diseño, donde la generación de contenido

y elementos típicos de dichas tareas y fases, contenían un alto porcentaje de aciertos, lo que destaca su capacidad en la comprensión del lenguaje natural para generar soluciones acertadas y de alta calidad.

Finalmente, cabe destacar que este proyecto no solo ha aportado aprendizajes técnicos y prácticos, sino que también ha abierto nuevas perspectivas de investigación y aplicaciones. El trabajo realizado sienta las bases para futuras exploraciones sobre cómo aprovechar el uso de los LLMs en el proceso de desarrollo software, incorporando métricas más precisas, mejores prácticas, además de nuevos LLMs, ya que esta tecnología evoluciona de manera constante cada semana, obteniendo grandes avances en el sector de la informática.

Como aportación personal, desde un punto de vista de recién egresado de Ingeniería Informática y futuro profesional de la Informática, la realización de este proyecto me ha permitido identificar las ventajas y desventajas del uso de la inteligencia artificial generativa en equipos de desarrollo software. En la actualidad, la IA generativa contempla un ritmo de evolución constante y acelerado, lo que permite considerar su uso y aplicación en distintas fases del proceso de desarrollo software para mejorar aspectos como la productividad, la investigación de soluciones y la resolución de bloqueos durante el desarrollo.

Su integración en los equipos de desarrollo facilita la automatización de tareas rutinarias, que pueden llevarse a cabo de manera más eficiente bajo la supervisión de profesionales. No obstante, también existen inconvenientes relevantes como que muchas de las soluciones basadas en IA todavía presentan limitaciones en precisión y confiabilidad, lo que implica que el contenido generado requiere una validación exhaustiva, reduciendo su uso como ventaja para la ganancia en productividad. También, se suma la dependencia tecnológica y es que depender demasiado en estas herramientas y tecnologías puede llevar a la pérdida de habilidades críticas en los ingenieros informáticos.

Por estas razones, en el momento actual que se encuentran estas tecnologías, considero que la inteligencia artificial generativa es válida para su uso en equipos de desarrollo software siempre que haya una supervisión durante su uso. La IA generativa ayuda a la investigación y resolución de problemas, aunque es fundamental disponer de una base sólida de conocimientos en Ingeniería Informática para no ser dependiente de estas herramientas y siempre tener idea de lo que están generando los LLMs en sus salidas. Además, en muchas ocasiones, el uso excesivo de IA puede ser contraproducente, provocando la pérdida del tiempo, en lugar

de aprovechar métodos tradicionales y efectivos, como el uso de buscadores en Internet, la consulta en foros técnicos, o la colaboración con otros profesionales del sector. Por otro lado, es necesario conocer las limitaciones de los modelos actualmente, ya que como se ha visto a lo largo de la evaluación, se requiere de una gran cantidad de contexto en las entradas para generar respuestas aceptables y con cierto grado de calidad y, en la mayoría de ocasiones, un uso variado de *prompts* para corregir el contenido.

Como conclusión y opinión personal, considero que la integración de la inteligencia artificial generativa es, en la actualidad, una necesidad para mejorar la productividad en los equipos de desarrollo de software. No obstante, su papel debe entenderse como el de una herramienta de apoyo y no como un sustituto de los ingenieros informáticos y profesionales del sector. Dado que estas tecnologías aún requieren supervisión, su verdadero potencial radica en aprovecharlas como complemento, lo que permite obtener resultados significativos en plazos de tiempo reducidos.

## 7.2. Líneas Futuras

La situación actual<sup>17</sup> de la inteligencia artificial generativa en la realización de tareas del proceso de desarrollo software evoluciona de manera constante, por lo que este proyecto cuenta con un gran potencial de expansión y aplicaciones en distintas áreas de la ingeniería informática. A continuación, se presentan algunas líneas futuras de investigación y desarrollo derivadas de este trabajo:

### **Especialización de fases**

La principal línea de investigación que surge a partir de este proyecto consiste en profundizar en la evaluación del uso de IA generativa en tareas específicas de cada fase del proceso de software. Esto permitiría identificar en qué etapas su aportación es más significativa y qué limitaciones surgen en contextos concretos como análisis y diseño de sistemas, desarrollo software o pruebas de software.

### **Colaboración humana**

Este proyecto se puede realizar de distinta forma. Se podría realizar el desarrollo de todo un caso práctico haciendo uso de la inteligencia artificial generativa como asistente, teniendo en cuenta la experiencia del desarrollador.

---

<sup>17</sup>septiembre de 2025

### **Formalidad de métricas**

Otro aspecto clave consiste en definir y aplicar métricas más formales y estandarizadas, así como el uso de diferentes ponderaciones en función de cada tarea. De esta manera, sería posible obtener evaluaciones más objetivas y comparables entre distintos casos prácticos y modelos.

### **El uso de LLMs y buenas prácticas**

Una línea de investigación prometedora se centra en la combinación del uso de LLMs con la aplicación de buenas prácticas de ingeniería del software. Esto implica estudiar cómo los LLMs pueden no solo generar diseños, fragmentos de código o documentación, sino también fomentar estándares de calidad, patrones de diseño, pruebas automatizadas e integrarse con metodologías ágiles. De esta forma, los modelos no se limitarían a asistir en la producción de soluciones, sino que podrían actuar como agentes que promueven el cumplimiento de guías de estilo, la modularidad del sistema o la detección de posibles desviaciones respecto a las prácticas recomendadas en la industria. Este enfoque abriría la puerta a sistemas inteligentes capaces de no solo acelerar el desarrollo y la productividad humana, sino también garantizar que los resultados mantengan un nivel óptimo de calidad y coherencia con los principios fundamentales de la ingeniería informática. Además, este enfoque abre las puertas a introducir inteligencia artificial generativa en entornos de alta seguridad, para evaluar el uso de buenas prácticas y estándares, aunque siempre con asistencia humana y con experiencia.

### **Uso de distintos LLMs**

Finalmente, se plantea la ampliación del análisis a un mayor número de modelos de lenguaje, especialmente de código abierto. Esto permitiría establecer comparaciones más amplias, identificar fortalezas y debilidades particulares y explorar cómo la evolución de nuevas arquitecturas afecta a la calidad y eficiencia en la generación de software. Asimismo, también surge la necesidad de comparar modelos comerciales con modelos gratuitos y de código abierto.

Los distintos LLMs a utilizar pueden ser seleccionados a través del análisis y estudio de investigaciones académicas, como la mostrada en la sección [2.2](#).

# Bibliografía

- [1] OpenAI. *GPT-4 is OpenAI's most advanced system, producing safer and more useful responses*. <https://openai.com/index/gpt-4/>. 2023.
- [2] A. Álvarez, R. de las Heras y C. Lasa. *Métodos ágiles y Scrum*. Anaya Multimedia, 2012.
- [3] Ken Schwaber y Jeff Sutherland. *The Scrum Guide – The Definitive Guide to Scrum: The Rules of the Game*. <https://scrumguides.org/scrum-guide.html>. 2020.
- [4] Hugging Face. *The AI Community Building the Future*. <https://huggingface.co/>. s.f.
- [5] Thomas Wolf et al. *Transformers: State-of-the-Art Natural Language Processing*. <https://arxiv.org/abs/1910.03771>. 2023.
- [6] Visual Paradigm. *The First Development Tool Suite that drives your project to success*. <https://www.visual-paradigm.com/>. s.f.
- [7] GitHub Inc. *Build and ship software on a single, collaborative platform*. <https://github.com/>. s.f.
- [8] Git. *Linus Torvalds*. <https://git-scm.com/>. 2025.
- [9] LLC Taiga Agile. *Taiga: The free and open-source project management tool*. <https://taiga.io/>. 2025.
- [10] Balsamiq. *Wireframe your way to faster, better product decisions*. <https://balsamiq.com/>. 2024.
- [11] Kong Inc. *Insomnia*. Plataforma de desarrollo de APIs. 2025. URL: <https://insomnia.rest/>.
- [12] Angular Team. *Angular - Framework de desarrollo web*. <https://angular.dev/>. 2025.
- [13] Codecademy. *Model–View–Controller (MVC)*. <https://www.codecademy.com/article/mvc>. 2021.
- [14] NestJS. *A progressive Node.js framework for building efficient, reliable and scalable server-side applications*. <https://nestjs.com/>. 2025.
- [15] MariaDB. *MariaDB Server: the innovative open source database*. <https://mariadb.org/>. 2024.

- [16] HeidiSQL. *HeidiSQL - A lightweight client for MariaDB, MySQL and SQL Server*. <https://www.heidisql.com/>. 2024.
- [17] Microsoft. *Visual Studio Code - Code Editing*. <https://code.visualstudio.com/>. 2024.
- [18] Dae-Kyoo Kim y Hua Ming. “Assessing output reliability and similarity of large language models in software development: A comparative case study approach”. En: *Information and Software Technology* 185 (2025). <https://doi.org/10.1016/j.infsof.2025.107787>, pág. 107787. ISSN: 0950-5849.
- [19] Leonardo Banh, Florian Holldack y Gero Strobel. “Copiloting the future: How generative AI transforms Software Engineering”. En: *Information and Software Technology* 183 (2025). <https://www.sciencedirect.com/science/article/pii/S0950584925000904>, pág. 107751. ISSN: 0950-5849.
- [20] Alan Dennis, Barbara Haley Wixom y David Tegarden. “Systems Analysis and Design with UML”. En: 4th. Wiley, 2015. Cap. 1.
- [21] Alan Dennis, Barbara Haley Wixom y David Tegarden. “Systems Analysis and Design with UML”. En: 4th. Wiley, 2015. Cap. 3.
- [22] Alan Dennis, Barbara Haley Wixom y David Tegarden. “Systems Analysis and Design with UML”. En: 4th. Wiley, 2015. Cap. 4.
- [23] Qunjun Zhang et al. *A Survey on Large Language Models for Software Engineering*. arXiv preprint. 2024. arXiv: [2312.15223 \[cs.SE\]](https://arxiv.org/abs/2312.15223). URL: <https://arxiv.org/abs/2312.15223>.
- [24] Alessio Ferrari, Sallam Abualhaija y Chetan Arora. *Model Generation with LLMs: From Requirements to UML Sequence Diagrams*. 2024. arXiv: [2404.06371 \[cs.SE\]](https://arxiv.org/abs/2404.06371). URL: <https://arxiv.org/abs/2404.06371>.
- [25] Meriem Ben Chaaben, Lola Burgueño y Houari Sahraoui. *Towards using Few-Shot Prompt Learning for Automating Model Completion*. 2022. arXiv: [2212.03404 \[cs.SE\]](https://arxiv.org/abs/2212.03404). URL: <https://arxiv.org/abs/2212.03404>.
- [26] Xinyi Hou et al. *Large Language Models for Software Engineering: A Systematic Literature Review*. 2024. arXiv: [2308.10620 \[cs.SE\]](https://arxiv.org/abs/2308.10620). URL: <https://arxiv.org/abs/2308.10620>.

- [27] Junjie Wang et al. “Software Testing With Large Language Models: Survey, Landscape, and Vision”. En: *IEEE Transactions on Software Engineering* 50.4 (2024), págs. 911-936. DOI: [10.1109/TSE.2024.3368208](https://doi.org/10.1109/TSE.2024.3368208).
- [28] Nan Jiang et al. *Impact of Code Language Models on Automated Program Repair*. 2023. arXiv: [2302.05020 \[cs.SE\]](https://arxiv.org/abs/2302.05020). URL: <https://arxiv.org/abs/2302.05020>.
- [29] Matthew Jin et al. *InferFix: End-to-End Program Repair with LLMs*. 2023. arXiv: [2303.07263 \[cs.SE\]](https://arxiv.org/abs/2303.07263). URL: <https://arxiv.org/abs/2303.07263>.
- [30] Manuel Cillero. *Diagrama de secuencia: descripción y propósito*. <https://manuel.cillero.es/doc/metodologia/metrica-3/tecnicas/diagrama-de-interaccion/diagrama-de-secuencia/>.
- [31] Mermaid. *Mermaid Live Editor – Generate Diagrams from Text*. 2025. URL: <https://mermaid.live/>.
- [32] diagrams.net. *diagrams.net (anteriormente draw.io) – Online Diagramming Tool*. 2025. URL: <https://app.diagrams.net/>.
- [33] ChartDB. *ChartDB – Free Online Database Diagram Tool*. 2025. URL: <https://chartdb.io/>.
- [34] Microsoft. *Best practices for RESTful web API design*. <https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design>. 2025.
- [35] node-argon2 contributors. *argon2*. npm, Inc., 2025. URL: <https://www.npmjs.com/package/argon2>.
- [36] *PactumJS: A REST API Testing Tool for all levels in a Test Pyramid*. 2025. URL: [%5Curl%7Bhttps://pactumjs.github.io/%7D](https://pactumjs.github.io/).
- [37] *Jest: Delightful JavaScript Testing Framework*. <https://jestjs.io/>. 2025.
- [38] *Argon2 Hash Generator & Verifier*. <https://argon2.online/>.
- [39] Beyond Identity Glossary. *Rainbow Table Attack*. <https://www.beyondidentity.com/glossary/rainbow-table-attack>.



# Apéndice A

## Prompts y salidas

Este apéndice recoge todos los prompts utilizados por los modelos y las salidas de cada uno. Asimismo, recopila cada prompt y salida de cada fase. En ocasiones, se ha requerido traducir la salida generada por modelos, especialmente Mistral. De esta forma, se facilita la lectura e interpretación.

### A.1. Análisis y diseño

#### A.1.1. Llama 3.1

Después de realizar el análisis y especificación de cada requisito, se ha procedido a pedir a cada LLM que genere su respuesta. Para la definición de requisitos y catálogo, la entrada textual ha sido el enunciado del caso práctico (sección 3.1.2) y se ha especificado en cada entrada que para cada tipo de requisito (funcional, no funcional), genere un catálogo. Para el caso de requisitos de información no se ha tenido en cuenta para este estudio debido a la ambigüedad que puede generar con los requisitos funcionales y no funcionales.

Por ello, teniendo en cuenta el enunciado del caso práctico, la estructura de la especificación de requisitos (identificador, nombre, descripción), se ha obtenido los siguientes requisitos.

#### Catálogo de requisitos funcionales

##### Prompt

'Obtén de este enunciado caso práctico (sección 3.1.2) el análisis de requisitos FUNCIONALES. Proporciona la especificación de cada requisito teniendo en cuenta la siguiente estructura: IDENTIFICADOR, NOMBRE, DESCRIPCIÓN DE REQUISITO'

##### Salida del modelo

Catálogo inferior:

Identificador	Título	Descripción
---------------	--------	-------------

RF-001	Reservas sin registro	El sistema permite realizar reservas sin necesidad de registrarse previamente en la aplicación Web.
RF-002	Historial de reserva	Se almacena un historial de las reservas realizadas por cada usuario, incluyendo tanto reservas pasadas como futuras.
RF-003	Pago de las reservas	El sistema permite dos métodos de pago: transferencia bancaria y PayPal. Además, se generará una factura de pago para cada reserva realizada.
RF-004	Detalle de la actividad	Se almacena información como hora, descripción de la actividad, tarifa, imágenes, duración y día de la semana en el que se realiza cada actividad.
RF-005	Categorización de actividades	Las actividades están organizadas según sus categorías para facilitar su búsqueda y selección.
RF-006	Actualización de la tarifa y duración de las actividades	Los empleados pueden actualizar los campos de precio y duración de las actividades, siempre que estén justificadas adecuadamente.
RF-007	Gestionar monitor	Se asigna a cada monitor un calendario de actividades según su disponibilidad y calificaciones necesarias para realizar la actividad.
RF-008	Calificación necesaria para realizar la actividad	Las actividades están asociadas con ciertas calificaciones que los monitores deben tener para poder asignarlas y realizarlas.
RF-009	Comentario sobre la actividad	Los clientes pueden dejar comentarios después de realizar cada una de las actividades ofrecidas.
RF-010	Detalle del usuario	Se almacena información personal como nombre, apellidos, DNI o NIE, correo electrónico y teléfono móvil tanto para los clientes como para los empleados.

RF-011	Asignar actividad	Los empleados pueden actualizar la asignación de actividades a monitores en función de las necesidades y cambios en los horarios disponibles.
RF-012	Historial de reservas	Se mantiene un registro completo de todas las reservas realizadas en la empresa, que se utiliza para análisis de datos e historial de reservas.
RF-013	Diferenciar usuario anónimo	Se distingue claramente entre los usuarios que se registran en la aplicación (clientes) y aquellos que solo realizan reservas sin necesidad de registro.
RF-014	Gestionar	El responsable de la organización puede realizar todas las tareas descritas en el caso práctico, incluyendo gestionar a sus empleados.
RF-015	Herramienta para visualizar los datos	Los empleados y el responsable de la empresa pueden utilizar herramientas incorporadas en la plataforma Web para visualizar datos gráficamente.

### Catálogo de requisitos no funcionales

#### Prompt

'Obtén de este enunciado caso práctico (sección 3.1.2) el análisis de requisitos NO FUNCIONALES. Proporciona la especificación de cada requisito teniendo en cuenta la siguiente estructura: IDENTIFICADOR, NOMBRE, DESCRIPCIÓN DE REQUISITO'

#### Salida del modelo

Catálogo inferior:

Identificador	Título	Descripción
RNF-01	Seguridad	La aplicación debe garantizar la autenticación y autorización de los usuarios (clientes y empleados) a través de un sistema de sesiones seguras, protegiendo así el acceso a información sensible.

RNF-02	Confiabilidad	La aplicación debe ser capaz de manejar y recuperarse adecuadamente en caso de errores o fallas no previstas, asegurando la disponibilidad continua del servicio.
RNF-03	Escalabilidad	La aplicación debe poder acomodar un aumento significativo en el tráfico o número de usuarios sin una disminución en su rendimiento, asegurando que el servicio se mantenga disponible y eficiente.
RNF-04	Usabilidad	La interfaz gráfica de la aplicación debe ser intuitiva y fácil de usar para todos los usuarios, incluyendo clientes anónimos y empleados, con una estructura lógica y concisa en las secciones de registro e inicio de sesión.
RNF-05	Privacidad	La aplicación debe respetar la privacidad de los usuarios, protegiendo su información personal (nombre, apellidos, DNI o NIE, correo electrónico y teléfono móvil) de acceso no autorizado.
RNF-06	Soporte a diferentes dispositivos	La aplicación debe ser compatible con una variedad de navegadores web y plataformas (Windows, macOS, mobile) para garantizar su accesibilidad desde cualquier dispositivo.
RNF-07	Almacenamiento adecuado de datos	La aplicación debe almacenar los datos de las reservas y usuarios de manera segura, organizada y escalable para futuros análisis y necesidades.
RNF-08	Compatibilidad con métodos de pago	La aplicación debe admitir diferentes métodos de pago (transferencia bancaria, Paypal) de manera segura y eficiente para facilitar el proceso de pago al cliente.

RNF-09	Informes y análisis de datos	La aplicación debe proporcionar herramientas adecuadas para generar informes y realizar análisis de los datos almacenados, incluyendo gráficas y tablas para la fácil visualización.
RNF-10	Soporte técnico	La aplicación debe contar con un sistema adecuado de soporte técnico para responder a los problemas o inquietudes que puedan surgir, garantizando la disponibilidad del servicio.

### Casos de uso generados

Los casos de uso se basan en los requisitos funcionales, es por ello que para obtener los casos de uso a partir del uso de LLMs se ha introducido en la entrada los requisitos funcionales identificados por el humano.

#### Prompt

Identifica los actores y casos de uso dados los siguientes requisitos funcionales (apartado 3.2). Identifica los actores y casos de uso dados los siguientes requisitos funcionales. Agrupa los actores identificados con los correspondientes casos de uso que generes.

#### Salida del modelo

##### Actores

1. Responsable de la aplicación Web
2. Empleados de la empresa turística
3. Monitores de las actividades
4. Clientes registrados en el sistema
5. Usuarios anónimos (no registrados)
6. Entidad bancaria

##### Casos de Uso

1. **Gestión de empleados**

- Responsable de la aplicación Web
  - Crear nuevos empleados
  - Leer información de empleados
  - Actualizar información de empleados
  - Eliminar a los empleados

## 2. Gestión de monitores

- Empleados de la empresa turística
  - Crear nuevos monitores
  - Leer información de monitores
  - Actualizar información de monitores
  - Eliminar a los monitores

## 3. Reservar por parte de empleados hacia clientes

- Empleados de la empresa turística
  - Realizar reservas para clientes en el momento

## 4. Gestión de actividades turísticas

- Empleados de la empresa
  - Crear nuevas actividades
  - Actualizar información de actividades existentes
  - Acceder a información de actividades
  - Eliminar actividades

## 5. Asignar monitor por actividad

- Empleados de la empresa turística
  - Asignar actividades a cada monitor

## 6. Ver listado de monitores

- Empleados de la empresa turística
  - Acceder a tabla con todos los monitores de la empresa

#### **7. Análisis de datos**

- Responsable de la aplicación Web y empleados
  - Interactuar con datos mediante gráficas y tablas generadas

#### **8. Gestión de pagos online**

- Usuarios registrados o no
  - Realizar pagos de las reservas de actividades turísticas que deseen participar

#### **9. Pago mediante Paypal**

- Usuarios registrados o no (redirigidos a área de pago mediante el servicio Paypal)
  - Realizar pagos mediante la plataforma de PayPal

#### **10. Pago mediante transferencia bancaria**

- Usuarios registrados o no (redirigidos a plataforma de pago por parte de la entidad bancaria)
  - Realizar pagos mediante la plataforma de la entidad bancaria

#### **11. Generar factura de pago**

- Usuarios que realicen un pago
  - Pedir que se proporcione una factura para el pago realizado

#### **12. Gestión de comentarios de actividades**

- Empleados de la empresa turística
  - Leer y eliminar comentarios por actividad

**13. Actualizar precios de actividades**

- Empleados de la empresa
  - Actualizar los precios de las actividades turísticas

**14. Ver listado de clientes por actividad turística**

- Monitores
  - Acceder a listado de clientes por actividad turística que tienen asignada

**15. Registro de nuevo usuario**

- Usuarios anónimos (no registrados)
  - Dar de alta en el sistema proporcionando correo y contraseña

**16. Iniciar sesión**

- Clientes registrados en el sistema
  - Acceder al sistema mediante las credenciales que introdujeron en el registro de su cuenta

**17. Cerrar sesión**

- Clientes registrados con la sesión iniciada
  - Desconectarse como usuario registrado en el sistema

**18. Buscar actividades**

- Usuarios, registrados o no
  - Buscar actividades por categoría o nombre

**19. Realizar reservas**

- Usuarios, registrados o no
  - Acceder a la aplicación Web para realizar reservas de las actividades turísticas disponibles en cada día

#### 20. Crear comentarios

- Clientes registrados en el sistema
  - Dejar comentarios en las actividades turísticas

#### 21. Responder comentarios

- Clientes registrados, incluyendo empleados
  - Responder a comentarios existentes en las actividades turísticas

#### 22. Ver listado de reservas pasadas

- Usuarios registrados
  - Acceder a un listado de las reservas de actividades turísticas que realizaron con histórico

#### 23. Actualizar duración de las actividades

- Empleados o responsables de la aplicación Web (para situaciones excepcionales)
  - Actualizar la duración de la actividad en situaciones especiales

### Especificación de casos de uso

Para la especificación de casos de uso, la entrada al modelo se basa en utilizar el actor y caso de uso obtenido por el desarrollo manual para indicar al LLM que genere la especificación exacta, teniendo en cuenta una pre y post-condición y detallando un escenario principal y posibles escenarios alternativos, teniendo en cuenta las clases de análisis, útiles para los diagramas de secuencia.

#### Prompt

Dado el caso de uso de registro de usuario y el actor del caso de uso, usuario anónimo, genera la especificación del caso de uso teniendo en cuenta la siguiente estructura: Pre-Condición, Post-condición, Escenario principal del caso de uso, Escenarios alternativos, Clases de análisis involucradas en el caso de uso (clases de entidad, controladores e

interfaces).

### Salida del modelo

Salida en la tabla:

<b>Título</b>	Registro de Usuario
<b>Descripción</b>	El usuario anónimo puede crear una cuenta en la aplicación, proporcionando información personal y eligiendo un nombre de usuario y contraseña.
<b>Pre-condición</b>	<ul style="list-style-type: none"> <li>▪ El usuario debe tener acceso a la aplicación.</li> <li>▪ La aplicación debe estar disponible y conectada a la base de datos.</li> </ul>
<b>Post-condición</b>	<ul style="list-style-type: none"> <li>▪ El usuario se registra exitosamente en la aplicación con una cuenta válida.</li> <li>▪ Se envía un correo electrónico de confirmación para que el usuario active su cuenta.</li> </ul>
<b>Prioridad</b>	Alta
<b>Escenario principal</b>	

**Escenario de registro de nuevo usuario**

1. El usuario anónimo accede a la página de registro de la aplicación.
2. Proporciona su información personal solicitada (nombre, dirección, teléfono).
3. Elige un nombre de usuario y una contraseña.
4. Confirma la información proporcionada.
5. El sistema finaliza el registro y envía un correo de activación.

**Escenarios alternativos**

**Escenario alternativo**

- **A1:** Si la información es incorrecta o incompleta, la aplicación indicará los campos obligatorios.
- **A2:** Si el nombre de usuario ya existe, se mostrará un mensaje solicitando uno nuevo.
- **A3:** Si hay fallos técnicos con la base de datos o el servidor, se mostrará una página de error con instrucciones.

**Clases de análisis**

**A. Clases de entidad**

- Usuario: Representa a los usuarios que se registran.
- Rol: Define los roles de usuario y sus permisos.

**B. Clases de control**

- RegistroController: Gestiona el proceso de registro.
- AuthController: Gestiona la autenticación de usuarios.

<b>C. Clases de interfaz</b>	<ul style="list-style-type: none"> <li>▪ UsuarioService: Lógica de negocio para usuarios.</li> <li>▪ CorreoService: Envío de correos de confirmación.</li> </ul>
------------------------------	--

El inicio de sesión de un cliente es similar por lo que se ha obtenido la especificación de procesos más relevantes como la gestión de empleados, reservas de actividades y pagos de reservas.

Prompt
Dado el caso de uso de gestión de empleados (operaciones Create, Read, Update, Delete) y el actor del caso de uso, responsable de la aplicación Web, genera la especificación del caso de uso teniendo en cuenta la siguiente estructura: Pre-Condición, Post-condición, Escenario principal del caso de uso, Escenarios alternativos, Clases de análisis involucradas en el caso de uso (clases de entidad, controladores e interfaces).

Salida del modelo
Salida en la tabla:

<b>Título</b>	Gestión de Empleados
<b>Descripción</b>	Permite al responsable de la aplicación Web realizar operaciones de creación, consulta, actualización y eliminación de registros de empleados dentro del sistema.
<b>Pre-condición</b>	<ul style="list-style-type: none"> <li>▪ El usuario responsable de la aplicación Web debe tener permisos para acceder a la función de gestión de empleados.</li> <li>▪ Debe existir un registro en el sistema para cada empleado.</li> </ul>

<b>Post-condición</b>	<ul style="list-style-type: none"> <li>▪ Los datos del empleado deben estar actualizados en el sistema.</li> <li>▪ El empleado debe ser eliminado correctamente del sistema si se decide dar de baja o eliminar su cuenta.</li> </ul>
<b>Prioridad</b>	Alta
<b>Escenario principal</b>	
<ol style="list-style-type: none"> <li>1. El usuario responsable de la aplicación Web entra a la plataforma y busca la opción de "Gestión de Empleados".</li> <li>2. Se redirige al menú correspondiente donde puede crear, leer, actualizar o eliminar empleados.</li> <li>3. El usuario selecciona la acción deseada (Crear, Leer, Actualizar o Eliminar) y llena los campos correspondientes con la información del empleado.</li> <li>4. Si se crea un nuevo empleado, el sistema asigna automáticamente un ID único a ese registro y lo almacena en la base de datos.</li> <li>5. Si se actualiza o elimina un empleado, el sistema busca primero al empleado y realiza las modificaciones solicitadas.</li> </ol>	
<b>Escenarios alternativos</b>	

<ul style="list-style-type: none"> <li>▪ <b>Crear empleado:</b> El usuario puede crear un nuevo empleado con la función "Crear". El sistema solicita la información básica del empleado (nombre, apellido, correo electrónico, etc.) y crea un registro para ese empleado en la base de datos.</li> <li>▪ <b>Leer empleado:</b> El usuario selecciona la opción "Leer" el sistema muestra la lista de empleados registrados. Puede buscar por ID, nombre o correo electrónico a un empleado específico si así lo desea.</li> <li>▪ <b>Actualizar empleado:</b> Si se selecciona "Actualizar", el sistema busca al empleado cuyo registro deseamos modificar. Una vez encontrado, se pueden actualizar los campos correspondientes con las nuevas informaciones del empleado.</li> <li>▪ <b>Eliminar empleado:</b> El usuario selecciona "Eliminar" el sistema solicita confirmación para eliminar al empleado de la base de datos. Una vez eliminado, no podrá recuperarse la información.</li> </ul>	
<b>Clases de análisis</b>	
<b>A. Clases de entidad</b>	<ul style="list-style-type: none"> <li>▪ Empleados: Clase de entidad que representa a los empleados del sistema.</li> </ul>
<b>B. Clases de control</b>	<ul style="list-style-type: none"> <li>▪ ControladorGestionEmpleados: Controlador que maneja las operaciones básicas sobre la gestión de empleados (crear, leer, actualizar y eliminar).</li> <li>▪ Servicio de Empleado: Servicio que encapsula el comportamiento para manipular los registros de empleados en la base de datos.</li> </ul>

<p><b>C. Clases de interfaz</b></p>	<ul style="list-style-type: none"> <li>▪ Interfaz de Gestión de Empleados: Interfaz que define las operaciones para interactuar con la lógica del negocio relacionada a la gestión de empleados.</li> </ul>
-------------------------------------	---

<p><b>Prompt</b></p>
<p>Dado el caso de uso de reserva de actividad turística en la web y el actor del caso de uso, usuario anónimo, genera la especificación del caso de uso teniendo en cuenta la siguiente estructura: Pre-Condición, Post-condición, Escenario principal del caso de uso, Escenarios alternativos, Clases de análisis involucradas en el caso de uso (clases de entidad, controladores e interfaces).</p>

<p><b>Salida del modelo</b></p>
<p>Salida en la tabla:</p>

<p><b>Título</b></p>	<p>Reservar Actividad Turística</p>
<p><b>Descripción</b></p>	<p>El usuario anónimo puede reservar una actividad turística seleccionada, proporcionando sus datos personales y recibiendo confirmación por correo electrónico.</p>
<p><b>Pre-condición</b></p>	<ul style="list-style-type: none"> <li>▪ El usuario anónimo ha seleccionado la actividad turística que desea reservar.</li> <li>▪ La actividad tiene disponibilidad de fecha y hora.</li> <li>▪ El usuario cuenta con una cuenta en el sistema (no es necesario estar logueado).</li> </ul>

<b>Post-condición</b>	<ul style="list-style-type: none"> <li>▪ La reserva está registrada en el sistema.</li> <li>▪ El usuario anónimo ha recibido un correo electrónico con la confirmación de la reserva.</li> </ul>
<b>Prioridad</b>	Alta
<b>Escenario principal</b>	
<p><b>Escenario de reserva de actividad turística</b></p> <ol style="list-style-type: none"> <li>1. El usuario anónimo selecciona la actividad turística y da clic en Reservar”.</li> <li>2. Se abre un formulario donde introduce: <ul style="list-style-type: none"> <li>▪ Nombre y apellidos.</li> <li>▪ Correo electrónico.</li> <li>▪ Teléfono móvil.</li> <li>▪ Número de personas (si es necesario).</li> </ul> </li> <li>3. El usuario completa el formulario y da clic en Reservar”.</li> <li>4. El sistema comprueba la disponibilidad.</li> <li>5. Si es posible, se registra la reserva y se envía confirmación por correo electrónico.</li> <li>6. Se actualiza la disponibilidad de la actividad.</li> </ol>	
<b>Escenarios alternativos</b>	

<p><b>Escenario alternativo</b></p> <ul style="list-style-type: none"> <li>▪ <b>A1: Actividad no disponible</b> – Se informa al usuario que no hay plazas disponibles para la fecha y hora seleccionadas.</li> <li>▪ <b>A2: Formulario incompleto o incorrecto</b> – El sistema informa qué campos deben corregirse o completarse.</li> <li>▪ <b>A3: Error técnico</b> – Se muestra un mensaje explicando que hubo un problema al procesar la reserva.</li> </ul>	
<p><b>Clases de análisis</b></p>	
<p><b>A. Clases de entidad</b></p>	<ul style="list-style-type: none"> <li>▪ <b>ActividadTuristica:</b> Representa la actividad seleccionada.</li> <li>▪ <b>Reserva:</b> Representa la reserva realizada.</li> </ul>
<p><b>B. Clases de control</b></p>	<ul style="list-style-type: none"> <li>▪ <b>ControladorReservas:</b> Gestiona el proceso de reserva.</li> </ul>
<p><b>C. Clases de interfaz</b></p>	<ul style="list-style-type: none"> <li>▪ <b>ServicioCorreoElectronico:</b> Envía la confirmación por correo al usuario.</li> </ul>

**Prompt**

Dado el caso de uso de pago reserva de actividad turística en la web y el actor del caso de uso, usuario anónimo, genera la especificación del caso de uso teniendo en cuenta la siguiente estructura: Pre-Condición, Post-condición, Escenario principal del caso de uso, Escenarios alternativos, Clases de análisis involucradas en el caso de uso (clases de entidad, controladores e interfaces).

**Salida del modelo**

Salida en la tabla:

<b>Título</b>	Reservar Actividad Turística
<b>Descripción</b>	Permite a un usuario anónimo reservar una actividad turística disponible completando un formulario con sus datos personales.
<b>Pre-condición</b>	<ul style="list-style-type: none"> <li>▪ El usuario anónimo ha seleccionado la actividad turística que desea reservar.</li> <li>▪ La actividad tiene disponibilidad de fecha y hora.</li> <li>▪ El usuario cuenta con una cuenta en el sistema (no es necesario estar logueado).</li> </ul>
<b>Post-condición</b>	<ul style="list-style-type: none"> <li>▪ La reserva está registrada en el sistema.</li> <li>▪ El usuario anónimo ha recibido un correo electrónico con la confirmación de la reserva.</li> </ul>
<b>Prioridad</b>	Alta
<b>Escenario principal</b>	

1. El usuario anónimo selecciona la actividad turística que desea reservar y da clic en Reservar”.
2. Se abre un formulario para rellenar los datos necesarios para la reserva:
  - Nombre y apellidos.
  - Correo electrónico.
  - Teléfono móvil.
  - Número de personas que participarán (si es necesario).
3. El usuario anónimo rellena el formulario con los datos correctos y da clic en -eservar”.
4. El sistema comprueba la disponibilidad de la actividad y, si no hay problema alguno, envía una confirmación de reserva al usuario a través del correo electrónico.
5. El sistema también actualiza la disponibilidad de la actividad para evitar que se reserve con el mismo horario.

#### **Escenarios alternativos**

<ul style="list-style-type: none"> <li>▪ <b>Actividad no disponible:</b> Si la actividad no tiene disponibilidad de fecha y hora, se muestra un mensaje de error al usuario anónimo explicando que no hay plazas disponibles.</li> <li>▪ <b>Formulario incompleto:</b> Si el usuario anónimo no rellena todos los campos del formulario o introduce datos incorrectos, se muestra un mensaje de error con la descripción de qué campo está mal relleno y por qué.</li> <li>▪ <b>Error al procesar la reserva:</b> Si hay algún problema técnico al procesar la reserva (por ejemplo, problemas de conexión con la base de datos), se muestra un mensaje de error explicando el problema.</li> </ul>	
<b>Clases de análisis</b>	
<b>A. Clases de entidad</b>	<ul style="list-style-type: none"> <li>▪ <b>ActividadTuristica:</b> Representa la actividad turística seleccionada por el usuario anónimo.</li> <li>▪ <b>Reserva:</b> Representa la reserva realizada por el usuario anónimo.</li> </ul>
<b>B. Clases de control</b>	<ul style="list-style-type: none"> <li>▪ <b>ControladorReservas:</b> Gestiona las solicitudes de reserva y realiza los procesos necesarios para reservar una actividad.</li> </ul>
<b>C. Clases de interfaz</b>	<ul style="list-style-type: none"> <li>▪ <b>ServicioCorreoElectronico:</b> Envía correos electrónicos al usuario con la confirmación de la reserva.</li> </ul>

### Modelo de clases

Para el modelo de clases se ha tenido en cuenta lo generado en la especificación de casos

de uso.

### **Modelo físico de datos**

Se decidió que el uso de LLMs para obtener el modelo físico de datos no era relevante puesto que se pretende evaluar este modelado en la obtención de la base de datos del sistema, teniendo en cuenta relaciones y restricciones.

### **Diagramas de secuencia**

Para los diagramas de secuencia se ha proporcionado el actor del caso de uso, especificación del caso de uso que incluye escenario principal, alternativos y clases de análisis:

Comenzando con registro de nuevo usuario:

#### **Prompt**

Quiero que generes un diagrama de secuencia textual a partir de la siguiente especificación de caso de uso sobre registros de nuevos usuarios. No uses pseudo-lenguajes ni código de modelado como UML. En su lugar, describe toda la secuencia paso a paso, con actores, objetos, vistas, controladores y modelos, incluyendo mensajes, llamadas y retornos, así como condiciones para escenarios alternativos. El resultado debe reflejar un diagrama de secuencia UML completo, pero expresado en lenguaje natural.

Caso de uso: - Actor principal: [Usuario Anónimo] - Escenario principal: [ 1. El usuario en la página web hace clic en 'Registrarse' 2. El sistema carga y muestra un formulario de registro 3. El usuario introduce sus datos en los campos correspondientes 4. El usuario hace clic en 'Crear cuenta' 5. El sistema valida los datos 6. El sistema crea la cuenta y muestra mensaje de confirmación al usuario 7. El sistema redirige al usuario a la ventana de inicio de sesión ] - Escenarios alternativos: [ Escenario alternativo de registro de usuario ya existente 6a. El sistema verifica que los datos introducidos como correo electrónico o nombre de usuario ya se encuentran en la base de datos del sistema 6b. El sistema muestra un mensaje de error de cuenta ya existente Escenario alternativo datos incorrectos 6a. El sistema comprueba que los datos introducidos no se ajustan al formato válido de los campos. 6b. El sistema muestra un mensaje de error de datos inválidos ]

Clases de análisis implicadas: - Modelos: [Usuarios, Clientes] - Vistas: [ViewPaginaPrincipal, ViewRegistroUsuario, ViewIniciarSesión] - Controladores: [ControllerUsuarios]

Formato de salida esperado: - Lista ordenada de interacciones en secuencia. - Indicar

claramente qué actor/objeto inicia cada interacción. - Usar flechas o notación textual tipo: 'Actor → Vista: acción', 'Vista → Controlador: evento' 'Controlador → Modelo: consulta', 'Modelo → Controlador: respuesta'. - Incluir condiciones y ramificaciones de escenarios alternativos con frases tipo 'Si [condición], entonces...'.

### Salida del modelo

1. **Actor principal:** Usuario Anónimo.
2. **Escenario principal:**
  - **Paso 1:** El usuario hace clic en *Registrarse* en la página web.  
**Actor:** Usuario Anónimo  
**Interacción:** Hacer clic en botón *Registrarse*.
  - **Paso 2:** El sistema carga y muestra un formulario de registro. (Vista: *ViewRegistroUsuario*)  
**Actores/Objetos involucrados:** Sistema (Controlador)  
**Acción:** Cargar y mostrar formulario de registro.
  - **Paso 3:** El usuario introduce sus datos en los campos correspondientes.  
**Actor:** Usuario Anónimo  
**Interacción:** Introducir datos en el formulario.
  - **Paso 4:** El usuario hace clic en *Crear cuenta*.  
**Actor:** Usuario Anónimo  
**Interacción:** Hacer clic en botón *Crear cuenta*.
  - **Paso 5:** El sistema valida los datos.  
**Actores/Objetos involucrados:** Sistema (Controlador)  
**Acción:** Validar datos.
  - **Paso 6a:** Si los datos son válidos, el sistema crea la cuenta y muestra mensaje de confirmación al usuario. (Vista: *ViewIniciarSesión*)  
**Actores/Objetos involucrados:** Sistema (Controlador), Usuarios (Modelo)  
**Acción:** Crear cuenta y mostrar mensaje de confirmación.
  - **Paso 6b:** Si los datos no son válidos, el sistema muestra un mensaje de error

de datos inválidos. (Vista: *ViewRegistroUsuario*)

**Actores/Objetos involucrados:** Sistema (Controlador)

**Acción:** Mostrar mensaje de error.

- **Paso 7:** El sistema redirige al usuario a la ventana de inicio de sesión.

**Actores/Objetos involucrados:** Sistema (Controlador), Vista: *ViewIniciar-Sesión*

**Acción:** Redirigir a ventana de inicio de sesión.

### 3. Escenarios alternativos:

- **Alternativo 1: Registro de usuario ya existente**

El sistema verifica que los datos introducidos (correo electrónico o nombre de usuario) ya se encuentran en la base de datos del sistema.

**Actores/Objetos involucrados:** Sistema (Controlador), Usuarios (Modelo)

**Acción:** Verificar registro existente.

- **Alternativo 1a:** Si el registro existe, el sistema muestra un mensaje de error de cuenta ya existente. (Vista: *ViewRegistroUsuario*)

**Actores/Objetos involucrados:** Sistema (Controlador)

**Acción:** Mostrar mensaje de error.

- **Alternativo 2: Datos incorrectos**

El sistema comprueba que los datos introducidos no se ajustan al formato válido de los campos.

**Actores/Objetos involucrados:** Sistema (Controlador)

**Acción:** Comprobar formato de datos.

- **Alternativo 2a:** Si los datos son incorrectos, el sistema muestra un mensaje de error de datos inválidos. (Vista: *ViewRegistroUsuario*)

**Actores/Objetos involucrados:** Sistema (Controlador)

**Acción:** Mostrar mensaje de error.

**Prompt**

Quiero que generes un diagrama de secuencia textual a partir de la siguiente especificación de caso de uso sobre gestión de empleados (CRUD empleados). No uses pseudolenguajes ni código de modelado como UML. En su lugar, describe toda la secuencia paso a paso, con actores, objetos, vistas, controladores y modelos, incluyendo mensajes, llamadas y retornos, así como condiciones para escenarios alternativos. El resultado debe reflejar un diagrama de secuencia UML completo, pero expresado en lenguaje natural.

Caso de uso: - Actor principal: [Responsable] - Escenario principal: [ Escenario de creación de empleado 1. El responsable hace clic en 'Añadir empleado'. 2. El sistema genera un formulario para introducir los datos del nuevo empleado. 3. El responsable introduce los datos del nuevo empleado. 4. El sistema verifica y valida los datos del nuevo empleado que se quiere agregar. 5. El sistema guarda la información y confirma la creación del nuevo empleado en la página web de la empresa. Escenario de eliminación de empleado 1. El responsable busca el empleado y hace clic sobre el ícono de eliminar. 2. El sistema muestra un mensaje de aviso para confirmar eliminación de empleado. 3. El responsable confirma la acción de borrado. 4. El sistema elimina toda información de ese empleado y muestra un mensaje de confirmación del borrado. Escenario de lectura de información de empleado 1. El responsable accede al listado de empleados. 2. El responsable busca el empleado y hace clic sobre su cuenta. 3. El sistema carga y muestra el perfil del empleado. 4. El responsable, en el perfil de la cuenta del empleado ya tendría disponible toda la información sobre ese empleado. Escenario de actualización de información de empleado 1. El responsable busca el empleado y hace clic sobre el icono de editar. 2. El sistema carga y muestra el perfil del empleado, con permisos de edición sobre los campos de la información del empleado. 3. El responsable selecciona lo(s) campo(s) que quiere editar. 4. El responsable edita y hace clic en 'Confirmar cambios'. 5. El sistema valida la información y la actualiza. ] - Escenarios alternativos: [ Escenario alternativo de creación de empleado 4a. El sistema informa que los datos introducidos no son válidos. 4b. El responsable debe de volver a introducir los datos. Escenario alternativo de eliminación de empleado 3a. El responsable hace clic sobre el botón 'Cancelar'. 3b. El sistema deshace la transacción de borrado. Escenario alternativo de actualización de

información de empleado 5a. El sistema comprueba los nuevos datos introducidos en los campos. 5b. El sistema comprueba que ha introducido datos como DNI o email ya existentes en el sistema. 5c. El sistema muestra mensaje de error de datos ya existentes en el sistema. ]

Clases de análisis implicadas: - Modelos: [Empleados, Usuarios] - Vistas: [ViewPaginaAdmin, ViewEmpleados] - Controladores: [ControllerUsuarios]

Formato de salida esperado: - Lista ordenada de interacciones en secuencia. - Indicar claramente qué actor/objeto inicia cada interacción. - Usar flechas o notación textual tipo: 'Actor → Vista: acción', 'Vista → Controlador: evento' 'Controlador → Modelo: consulta', 'Modelo → Controlador: respuesta'. - Incluir condiciones y ramificaciones de escenarios alternativos con frases tipo 'Si [condición], entonces...'

## Salida del modelo

### Escenario principal: Creación de empleado

1. **Responsable** hace clic en *Añadir empleado* en la vista *ViewPaginaAdmin*.
2. La acción es recibida por el controlador *ControllerUsuarios*, que inicia un proceso para generar un formulario de registro de empleados.
3. El controlador *ControllerUsuarios* llama al modelo *Empleados* para obtener una instancia vacía de empleado.
4. El modelo *Empleados* devuelve la instancia vacía a la vista *ViewEmpleados*, que muestra el formulario de registro.
5. **Responsable** introduce los datos del nuevo empleado en el formulario.
6. La acción es recibida por la vista *ViewEmpleados*, que envía los datos al controlador *ControllerUsuarios*.
7. El controlador *ControllerUsuarios* llama al modelo *Empleados* para validar y verificar los datos introducidos.
8. Si los datos son válidos, el modelo *Empleados* guarda la información en la base de datos.

9. El controlador *ControllerUsuarios* envía una respuesta al responsable, mostrando un mensaje de confirmación de creación del empleado.

#### **Escenario alternativo: Creación de empleado (4a)**

Si los datos introducidos no son válidos, el modelo *Empleados* devuelve un error al controlador *ControllerUsuarios*. La acción es recibida por la vista *ViewEmpleados*, que muestra un mensaje de error y permite que el responsable vuelva a introducir los datos.

#### **Escenario principal: Eliminación de empleado**

1. **Responsable** busca el empleado en la lista de empleados y hace clic sobre el ícono de eliminar.
2. La acción es recibida por la vista *ViewEmpleados*, que envía un evento al controlador *ControllerUsuarios*.
3. El controlador *ControllerUsuarios* llama al modelo *Empleados* para obtener la información del empleado seleccionado.
4. El modelo *Empleados* devuelve la información del empleado al controlador *ControllerUsuarios*.
5. El controlador *ControllerUsuarios* envía un mensaje de confirmación a la vista *ViewEmpleados*, solicitando que el responsable confirme la eliminación.
6. **Responsable** confirma la acción de borrado.

#### **Escenario alternativo: Eliminación de empleado (3a)**

Si el responsable hace clic en el botón *Cancelar*, la acción es recibida por la vista *ViewEmpleados*, que deshace la transacción de borrado y muestra la lista de empleados actualizada.

#### **Escenario principal: Lectura de información de empleado**

1. **Responsable** accede al listado de empleados en la vista *ViewPaginaAdmin*.
2. La acción es recibida por el controlador *ControllerUsuarios*, que carga la lista de empleados.

3. El controlador *ControllerUsuarios* envía una respuesta al responsable, mostrando la lista de empleados.
4. **Responsable** busca el empleado y hace clic sobre su cuenta en la vista *ViewEmpleados*.
5. La acción es recibida por la vista *ViewEmpleados*, que carga y muestra el perfil del empleado.

#### **Escenario principal: Actualización de información de empleado**

1. **Responsable** busca el empleado y hace clic sobre el ícono de editar en la vista *ViewEmpleados*.
2. La acción es recibida por la vista *ViewEmpleados*, que envía un evento al controlador *ControllerUsuarios*.
3. El controlador *ControllerUsuarios* llama al modelo *Empleados* para obtener la información del empleado seleccionado.
4. El modelo *Empleados* devuelve la información del empleado al controlador *ControllerUsuarios*.
5. El controlador *ControllerUsuarios* envía un mensaje de confirmación a la vista *ViewEmpleados*, solicitando que el responsable confirme los cambios.
6. **Responsable** selecciona los campos que quiere editar y hace clic en *Confirmar cambios*.

#### **Escenario alternativo: Actualización de información de empleado (5a)**

Si el sistema comprueba que ha introducido datos como DNI o email ya existentes en el sistema, el modelo *Empleados* devuelve un error al controlador *ControllerUsuarios*. La acción es recibida por la vista *ViewEmpleados*, que muestra un mensaje de error y permite que el responsable vuelva a editar los campos.

#### **Modelos**

- **Empleados:** gestiona la información de los empleados, incluyendo su creación, eliminación y actualización.
- **Usuarios:** gestiona la información de los usuarios, incluyendo su acceso a la aplicación.

### Vistas

- **ViewPaginaAdmin:** muestra la página principal del administrador, donde se pueden realizar acciones como crear, eliminar y actualizar empleados.
- **ViewEmpleados:** muestra la lista de empleados y permite al responsable acceder a la información y realizar acciones sobre ellos.

### Controladores

- **ControllerUsuarios:** gestiona las acciones relacionadas con los empleados, incluyendo su creación, eliminación y actualización.

### Prompt

Quiero que generes un diagrama de secuencia textual a partir de la siguiente especificación de caso de uso de 'Reservar una actividad turística'. No uses pseudo-lenguajes ni código de modelado como UML. En su lugar, describe toda la secuencia paso a paso, con actores, objetos, vistas, controladores y modelos, incluyendo mensajes, llamadas y retornos, así como condiciones para escenarios alternativos. El resultado debe reflejar un diagrama de secuencia UML completo, pero expresado en lenguaje natural.

Caso de uso: - Actor principal: [Usuario Anónimo] - Escenario principal: [ Escenario de reserva de actividad turística 1. El usuario escoge la actividad que desea realizar 2. El sistema muestra los detalles de la actividad escogida 3. El usuario escoge la fecha de la actividad y comprueba su disponibilidad 4. El sistema verifica la disponibilidad y muestra que sí que está disponible la actividad para la fecha seleccionada 5. El usuario hace clic en 'Reservar' 6. El sistema redirige al usuario a rellenar los datos necesarios para la reserva 7. El usuario introduce los datos necesarios para la reserva 8. El usuario hace clic en 'Realizar pago' 9. El sistema redirige al usuario a una plataforma de pago 10. Finali-

zando el pago, el sistema guarda la reserva y muestra confirmación al usuario junto a la factura generada ] - Escenarios alternativos: [ Escenario alternativo actividad no disponible 4a. El sistema informa al usuario que no hay disponibilidad para esa actividad con la fecha indicada 4b. El usuario decide escoger otra fecha o salir de la actividad seleccionada Escenario alternativo campos no válidos 7a. El usuario deja campos obligatorios sin completar o introduce datos inválidos 7b. El sistema muestra un mensaje de error indicando los campos a corregir 7c. El usuario corrige los datos y continúa o decide salir del formulario de reserva]

Clases de análisis implicadas: - Modelos: [Actividades, Reservas, Factura] - Vistas: [ViewPaginaPrincipal, ViewActividades, ViewReservas, ViewPago] - Controladores: [ControllerActividades, ControllerReservas, ControllerFactura]

Formato de salida esperado: - Lista ordenada de interacciones en secuencia. - Indicar claramente qué actor/objeto inicia cada interacción. - Usar flechas o notación textual tipo: 'Actor → Vista: acción', 'Vista → Controlador: evento' 'Controlador → Modelo: consulta', 'Modelo → Controlador: respuesta'. - Incluir condiciones y ramificaciones de escenarios alternativos con frases tipo 'Si [condición], entonces...'

### Salida del modelo

1. **Usuario anónimo** interactúa con *ViewPáginaPrincipal*: Acción (seleccionar actividad).  
El usuario selecciona una actividad.
2. **ControllerActividades** recibe la solicitud y muestra los detalles de la actividad en *ViewActividades*.  
Controlador: evento.
3. **Usuario anónimo** interactúa con *ViewActividades*: Acción (seleccionar fecha y verificar disponibilidad).  
El usuario selecciona una fecha para la actividad.
4. **ControllerReservas** verifica la disponibilidad de la actividad en el modelo *Reserva*.  
Controlador: consulta.

5. Si la actividad está disponible, **ControllerReservas** muestra los detalles en *View-PáginaPrincipal*; si no está disponible:
6. **ControllerReservas** informa al usuario que no hay disponibilidad para esa actividad con la fecha indicada.  
Modelo: respuesta (no disponible).
7. El usuario decide escoger otra fecha o salir de la actividad seleccionada.
8. Si el usuario quiere elegir una fecha diferente, repite los pasos del 1 al 4.
9. **Usuario anónimo** interactúa con *ViewReservas*: Acción (rellenar datos para la reserva).  
El usuario introduce los datos necesarios para la reserva.
10. **ControllerFactura** recibe los datos y muestra un formulario de pago en *ViewPago*.  
Controlador: evento.
11. **Usuario anónimo** interactúa con *ViewPago*: Acción (realizar pago).  
El usuario selecciona una plataforma de pago para realizar el pago.
12. **ControllerFactura** redirige al usuario a la plataforma de pago para finalizar el pago.  
Controlador: evento.
13. Finalizado el pago, **ControllerReservas** guarda la reserva en el modelo *Reserva* y muestra confirmación al usuario junto con la factura generada.  
Modelo: respuesta (reserva guardada).
14. El sistema muestra una vista de confirmación con los detalles de la reserva.

### Escenarios alternativos

#### 4a. Escenario alternativo actividad no disponible

Si no hay disponibilidad para esa actividad con la fecha indicada, **ControllerReservas** informa al usuario que no hay disponibilidad.

Modelo: respuesta (no disponible).

El usuario decide escoger otra fecha o salir de la actividad seleccionada.

#### **7a. Escenario alternativo campos no válidos**

Si el usuario deja campos obligatorios sin completar o introduce datos inválidos, **ControllerReservas** muestra un mensaje de error indicando los campos a corregir.

Controlador: evento.

El usuario corrige los datos y continúa o decide salir del formulario de reserva.

#### **7b. Escenario alternativo campos no válidos (continuación)**

Si el usuario quiere continuar, repite los pasos del 8 al 14; si decide salir del formulario de reserva, termina la interacción con el sistema.

### **Prompt**

Quiero que generes un diagrama de secuencia textual a partir de la siguiente especificación de caso de uso de 'Pago de reserva de una actividad turística'. No uses pseudolenguajes ni código de modelado como UML. En su lugar, describe toda la secuencia paso a paso, con actores, objetos, vistas, controladores y modelos, incluyendo mensajes, llamadas y retornos, así como condiciones para escenarios alternativos. El resultado debe reflejar un diagrama de secuencia UML completo, pero expresado en lenguaje natural.

Caso de uso: - Actor principal: [Usuario Anónimo] - Escenario principal: [ Escenario de pago de reserva de actividad turística 1. El sistema carga la pasarela de pago al usuario 2. El usuario elige entre dos métodos de pago; Paypal o Transferencia Bancaria 3. El usuario selecciona un método de pago 4. El usuario introduce los datos para el pago 5. El sistema proporciona campos para los datos necesarios para generar la factura 6. El usuario introduce los datos en los campos necesarios para Realizar pago 7. El usuario hace clic en 'Realizar pago' 8. El sistema procesa el pago 9. El sistema verifica el pago y muestra un mensaje de confirmación de pago, devolviendo confirmación y factura de pago de reserva de actividad turística ] - Escenarios alternativos: [ Escenario alternativo cancelación de reserva antes del pago 4a. El usuario decide no continuar y cierra la página o hace clic en 'Cancelar'. 4b. El sistema descarta la reserva 4c. Se muestra un mensaje de confirmación de cancelación de reserva de actividad turística Escenario alternativo error en plataforma de pago 7a. Se produce un fallo durante el proceso de pago, que

puede ser por error técnico con la pasarela de pago o la tarjeta ha sido rechazada 7b. El sistema detecta que el pago no fue completado correctamente 7c. El sistema muestra un mensaje al usuario indicando el problema 7d. El usuario decide reintentar el proceso, seleccionar otro método de pago o cancelar la transacción ]

Clases de análisis implicadas: - Modelos: [Factura] - Vistas: [ViewPago] - Controladores: [ControllerPago]

Formato de salida esperado: - Lista ordenada de interacciones en secuencia. - Indicar claramente qué actor/objeto inicia cada interacción. - Usar flechas o notación textual tipo: 'Actor → Vista: acción', 'Vista → Controlador: evento' 'Controlador → Modelo: consulta', 'Modelo → Controlador: respuesta'. - Incluir condiciones y ramificaciones de escenarios alternativos con frases tipo 'Si [condición], entonces...'.

### Salida del modelo

1. **Usuario anónimo** interactúa con *ViewPáginaPrincipal*: Acción (seleccionar actividad).  
El usuario selecciona una actividad.
2. **ControllerActividades** recibe la solicitud y muestra los detalles de la actividad en *ViewActividades*.  
Controlador: evento.
3. **Usuario anónimo** interactúa con *ViewActividades*: Acción (seleccionar fecha y verificar disponibilidad).  
El usuario selecciona una fecha para la actividad.
4. **ControllerReservas** verifica la disponibilidad de la actividad en el modelo *Reserva*.  
Controlador: consulta.
5. Si la actividad está disponible, **ControllerReservas** muestra los detalles en *ViewPáginaPrincipal*; si no está disponible:
6. **ControllerReservas** informa al usuario que no hay disponibilidad para esa actividad con la fecha indicada.

Modelo: respuesta (no disponible).

7. El usuario decide escoger otra fecha o salir de la actividad seleccionada.
8. Si el usuario quiere elegir una fecha diferente, repite los pasos del 1 al 4.
9. **Usuario anónimo** interactúa con *ViewReservas*: Acción (rellenar datos para la reserva).  
El usuario introduce los datos necesarios para la reserva.
10. **ControllerFactura** recibe los datos y muestra un formulario de pago en *ViewPago*.  
Controlador: evento.
11. **Usuario anónimo** interactúa con *ViewPago*: Acción (realizar pago).  
El usuario selecciona una plataforma de pago para realizar el pago.
12. **ControllerFactura** redirige al usuario a la plataforma de pago para finalizar el pago.  
Controlador: evento.
13. Finalizado el pago, **ControllerReservas** guarda la reserva en el modelo *Reserva* y muestra confirmación al usuario junto con la factura generada.  
Modelo: respuesta (reserva guardada).
14. El sistema muestra una vista de confirmación con los detalles de la reserva.

### Escenarios alternativos

#### 4a. Escenario alternativo actividad no disponible

Si no hay disponibilidad para esa actividad con la fecha indicada, **ControllerReservas** informa al usuario que no hay disponibilidad.

Modelo: respuesta (no disponible).

El usuario decide escoger otra fecha o salir de la actividad seleccionada.

#### 7a. Escenario alternativo campos no válidos

Si el usuario deja campos obligatorios sin completar o introduce datos inválidos, **ControllerReservas** muestra un mensaje de error indicando los campos a corregir.

Controlador: evento.

El usuario corrige los datos y continúa o decide salir del formulario de reserva.

### **7b. Escenario alternativo campos no válidos (continuación)**

Si el usuario quiere continuar, repite los pasos del 8 al 14; si decide salir del formulario de reserva, termina la interacción con el sistema.

## **Formatos de pantalla**

### **Prompt**

A partir del siguiente escenario de caso de uso, una breve descripción acerca del caso de uso y un formato esperado, genera un formato de pantalla que responda a las necesidades descritas y aporta una explicación de lo que has generado. Además, si es posible, proporciona el wireframe textual.

Descripción caso de uso: Un usuario anónimo decide registrarse en la plataforma web de reservas de actividades turísticas. El nuevo usuario necesita aportar una serie de datos y credenciales tales como nombre de usuario, nombre, apellidos, correo electrónico, número de teléfono, contraseña y DNI.

Formato esperado: La interfaz debe incluir componentes principales (campos de entrada de datos, botones de 'Crear cuenta' y 'Ir a inicio de sesión', etcétera). La disposición de los componentes.

- Escenarios del caso de uso:
1. El usuario en la página web hace clic en 'Registrarse'
  2. El sistema carga y muestra un formulario de registro
  3. El usuario introduce sus datos en los campos correspondientes
  4. El usuario hace clic en 'Crear cuenta'
  5. El sistema valida los datos
  6. El sistema crea la cuenta y muestra mensaje de confirmación al usuario
  7. El sistema redirige al usuario a la ventana de inicio de sesión

### **Salida del modelo**

**Título: Registro de Cuenta**

**Componentes principales**

1. **Formulario de registro**



### Prompt

A partir del siguiente escenario de caso de uso, una breve descripción acerca del caso de uso y un formato esperado, genera un formato de pantalla que responda a las necesidades descritas y aporta una explicación de lo que has generado. Además, si es posible, proporciona el wireframe textual.

Descripción caso de uso: Un cliente decide entrar con su cuenta registrada en el sistema para realizar reservas. El usuario ya previamente registrado, debe de introducir su nombre de usuario y contraseña.

Formato esperado: La interfaz debe incluir componentes principales (campos de entrada de datos, botones de 'Ir a registro a' y 'Iniciar sesión', etcétera). La disposición de los componentes.

Escenarios del caso de uso: 1. El usuario accede a la página web

2. El usuario hace clic en 'Iniciar sesión'

3. El sistema carga y muestra el formulario de inicio de sesión

4. El usuario introduce sus datos correspondientes

5. El sistema verifica y valida los datos introducidos con los almacenados en el sistema

6. El sistema verifica correctamente el usuario, el usuario inicia sesión y carga la pantalla principal de la página web

### Prompt

A partir del siguiente escenario de caso de uso, una breve descripción acerca del caso de uso y un formato esperado, genera un formato de pantalla que responda a las necesidades descritas y aporta una explicación de lo que has generado. Además, si es posible, proporciona el wireframe textual.

Descripción caso de uso: El responsable (administrador) de una aplicación web de reservas de actividades turísticas realiza la gestión de sus empleados a través de un panel de administración. Este responsable decide añadir un nuevo empleado. Cuando hace clic sobre "añadir nuevo empleado", el sistema carga un formulario en el que debe de introducir un nombre de usuario, nombre y apellidos, email, teléfono, contraseña, número de teléfono, DNI y seleccionar el rol (empleado, monitor).

Formato esperado: La interfaz debe incluir componentes principales (campos de entrada

de datos, botón de Registrar empleado”, etcétera). La disposición de los componentes.

Escenarios del caso de uso: Escenario de creación de empleado

1. El responsable hace clic en 'Añadir empleado'.
2. El sistema genera un formulario para introducir los datos del nuevo empleado.
3. El responsable introduce los datos del nuevo empleado.
4. El sistema verifica y valida los datos del nuevo empleado que se quiere agregar.
5. El sistema guarda la información y confirma la creación del nuevo empleado en la página web de la empresa.

### Salida del modelo

Formato de pantalla para caso de uso: "Añadir Nuevo Empleado"

#### Campo 1: Nombre de Usuario

- **Tipo de campo:** Texto
- **Etiqueta:** Nombre de usuario (obligatorio)
- **Características:** El nombre de usuario debe ser único y no puede contener caracteres especiales.

#### Campo 2: Nombre y Apellidos

- **Tipo de campo:** Texto
- **Etiqueta:** Nombre y apellidos (obligatorio)
- **Características:** Solo permite texto sin caracteres especiales.

#### Campo 3: Email

- **Tipo de campo:** Correo electrónico
- **Etiqueta:** Email del empleado (obligatorio)
- **Características:** El correo electrónico debe ser válido y único para cada usuario. Se puede verificar su existencia en el servidor al momento de la creación.

#### **Campo 4: Teléfono**

- **Tipo de campo:** Número de teléfono
- **Etiqueta:** Número de teléfono (obligatorio)
- **Características:** Sólo permite números sin caracteres especiales. Se puede verificar su validez al momento de la creación.

#### **Campo 5: Contraseña**

- **Tipo de campo:** Texto con seguridad
- **Etiqueta:** Contraseña del empleado (obligatorio)
- **Características:** Requiere al menos un carácter en mayúscula, una minúscula y un número. El sistema debe guardar la contraseña encriptada.

#### **Campo 6: Número de DNI**

- **Tipo de campo:** Número
- **Etiqueta:** Número de DNI (obligatorio)
- **Características:** Sólo permite números sin caracteres especiales. Se puede verificar su validez al momento de la creación.

#### **Campo 7: Rol del empleado**

- **Tipo de campo:** Selección múltiple
- **Etiqueta:** Rol del empleado (obligatorio)
- **Características:** El sistema debe permitir seleccionar entre los siguientes roles:
  - Empleado
  - Monitor

#### **Botón Registrar Empleado”**

- **Etiqueta:** Registrar empleado (obligatorio)
- **Características:** Cuando se hace clic en este botón, el sistema verifica y valida todos los campos del formulario. Si todos están válidos, crea un nuevo usuario con la información proporcionada.

### Wireframe textual del formato de pantalla

```
Nombre de Usuario _____  
Campo 1  
-----  
  
Nombre y Apellidos _____  
Campo 2  
-----  
  
Email: _____  
Campo 3  
-----  
  
Teléfono: _____  
Campo 4  
-----  
  
Contraseña: _____  
Campo 5  
-----  
  
Número de DNI: _____  
Campo 6  
-----  
  
Rol del empleado:  
[ ] Empleado  
[ ] Monitor  
Campo 7  
-----  
  
Botón Registrar Empleado
```

Figura 54: Crear nuevo empleado. Fuente: Llama 3.1

### Explicación

El formato generado responde a las necesidades planteadas en el caso de uso. Incluye todos los campos requeridos para crear un nuevo empleado y permite seleccionar el rol correspondiente al empleado o monitor. La disposición de los componentes es clara y fácil de usar, facilitando la entrada de datos para el responsable del panel de administración. El botón “Registrar empleado” es el último paso en el que se verifica y valida todos los campos antes de guardar la información del nuevo empleado.

### Prompt

A partir del siguiente escenario de caso de uso, una breve descripción acerca del caso de uso y un formato esperado, genera un formato de pantalla que responda a las necesidades descritas y aporta una explicación de lo que has generado. Además, si es posible, proporciona el wireframe textual.

Descripción caso de uso: El responsable (administrador) de una aplicación web de reservas de actividades turísticas realiza la gestión de sus empleados a través de un panel de administración. Este responsable decide eliminar un empleado existente. Para eliminar un empleado, dispone de una lista con todos los empleados y a lado de cada fila un botón 'Eliminar empleado'. Cuando hace clic aparece un mensaje de confirmación en la pantalla con dos botones, 'Aceptar' y 'Cancelar'.

Formato esperado: La interfaz debe incluir componentes principales (campos de entrada de datos, botón de 'Aceptar' y 'Cancelar', etcétera). La disposición de los componentes.

Escenarios del caso de uso: Escenario de eliminación de empleado

1. El responsable busca el empleado y hace clic sobre el ícono de eliminar.
2. El sistema muestra un mensaje de aviso para confirmar eliminación de empleado.
3. El responsable confirma la acción de borrado.
4. El sistema elimina toda información de ese empleado y muestra un mensaje de confirmación del borrado.

### Salida del modelo

#### Formato de Pantalla

#### Componentes y descripción

- **Lista de empleados:** Mostrar todos los empleados registrados en la aplicación, con cada fila incluyendo el nombre del empleado, su rol o cargo y otros datos relevantes. Al lado de cada fila debe aparecer un botón "Eliminar empleado".
- **Botones de acción:** Un botón para eliminar a un empleado específico, que se activa cuando se hace clic sobre él. Cuando se presiona, se despliega el mensaje de confirmación en la pantalla.
- **Mensaje de confirmación:** Mostrar un mensaje de aviso y confirmación antes de

eliminar al empleado. Incluir dos botones: “Aceptar” y “Cancelar”. El mensaje debe informar al responsable sobre lo que está a punto de hacerse, es decir, eliminar el registro del empleado y todos sus datos asociados.

- **Mensaje de eliminación:** Después de eliminar el empleado, mostrar un mensaje de confirmación para notificar al administrador que la operación se ha completado con éxito.

### Wireframe textual

- **1. Página principal de panel de administración**
  - **Título:** Gestión de empleados
  - **Sección de lista de empleados:**
    - Encabezado: “Lista de empleados”
    - Fila de encabezado: “Nombre del empleado”, “Rol o cargo”
    - Lista de filas con datos de cada empleado, incluyendo botón “Eliminar empleado”
- **2. Mensaje de confirmación de eliminación**
  - **Título:** Confirmar eliminación de empleado
  - **Contenido:** “¿Está seguro que quiere eliminar al empleado [nombre del empleado]?”
  - **Botones:**
    - “Aceptar” (acción para eliminar el empleado)
    - “Cancelar” (acción para regresar a la lista de empleados sin hacer cambios)
- **3. Mensaje de eliminación exitosa**
  - **Título:** Empleado eliminado con éxito
  - **Contenido:** “El empleado [nombre del empleado] ha sido eliminado correctamente.”

### Explicación

Para generar este formato de pantalla y el wireframe textual, he seguido las instrucciones proporcionadas y he considerado los pasos del caso de uso presentado. El resultado es una interfaz que responde a las necesidades del administrador para eliminar empleados de la aplicación.

La disposición de los componentes permite una fácil navegación y comprensión por parte del usuario. La lista de empleados se muestra en primera instancia, permitiendo al administrador seleccionar el empleado deseado para su eliminación. Luego, cuando se presiona el botón “Eliminar empleado”, aparece el mensaje de confirmación con los botones “Aceptar” y “Cancelar”. Después de eliminar el empleado, se muestra un mensaje de confirmación del borrado.

El wireframe textual proporciona una representación detallada de la interfaz a través de un texto. Esto ayuda a visualizar la disposición de los elementos en la pantalla sin requerir herramientas gráficas específicas para su creación y modificación.

### Prompt

A partir del siguiente escenario de caso de uso, una breve descripción acerca del caso de uso y un formato esperado, genera un formato de pantalla que responda a las necesidades descritas y aporta una explicación de lo que has generado. Además, si es posible, proporciona el wireframe textual.

Descripción caso de uso: El responsable (administrador) de una aplicación web de reservas de actividades turísticas realiza la gestión de sus empleados a través de un panel de administración. Este responsable decide visualizar la información un empleado existente. Para ello, dispone de una lista con todos los empleados y a lado de cada fila un botón 'Ver empleado'. Cuando hace clic carga una pantalla con la información de dicho empleado en la que aparece nombre de usuario, nombre y apellidos, correo electrónico y número de teléfono.

Formato esperado: La interfaz debe incluir componentes principales (campos correspondientes). La disposición de los componentes.

Escenarios del caso de uso: Escenario de lectura de información de empleado

1. El responsable accede al listado de empleados.

2. El responsable busca el empleado y hace clic sobre su cuenta.
3. El sistema carga y muestra el perfil del empleado.
4. El responsable, en el perfil de la cuenta del empleado ya tendría disponible toda la información sobre ese empleado.

## Salida del modelo

### Caso de uso: Visualización de Información de Empleado

#### Breve descripción

El panel de administración permite a los responsables visualizar y gestionar la información de empleados existentes en la aplicación web de reservas de actividades turísticas. Para ello, el responsable puede acceder a un listado de empleados y seleccionar al empleado cuya información desee visualizar, lo que conduce a una pantalla de perfil donde se muestra la información completa del empleado.

#### Formato esperado

La interfaz debe incluir los siguientes componentes principales y características:

- **Listado de Empleados:** Un área donde se muestre la lista de todos los empleados existentes. Cada fila en la lista deberá contener al menos el nombre, apellidos o usuario del empleado y un botón para acceder a su perfil.
- **Perfil de Empleado:**
  - **Nombre de Usuario:** Campo de texto donde se muestra el nombre de usuario asignado al empleado.
  - **Nombre y Apellidos:** Campo de texto que incluye los nombres y apellidos del empleado.
  - **Correo Electrónico:** Campo de texto para la dirección de correo electrónico asociada al empleado.
  - **Número de Teléfono:** Campo de texto para el número telefónico del empleado.

#### Disposición de los componentes



### Wireframe textual para la pantalla del perfil del empleado con detalles adicionales (si corresponde)

```
+-----+
|                Perfil de Empleado                |
+-----+

Nombre y Apellidos: Juan Pérez
Correo Electrónico: juan.perez@email.com
Número de Teléfono: 123456789

Dirección de Residencia: Calle x, Dpto. y, Barrio z
Fecha de Ingreso: 2022-01-15
... (Otros campos relevantes)
```

Figura 57: Ver información adicional de empleado. Fuente: Llama 3.1

#### Explicación

Las imágenes de wireframe textual proporcionan una representación básica gráfica de cómo podría verse la interfaz en cada pantalla.

Este diseño responde a las necesidades de los responsables que gestionan empleados. La lista inicial permite navegar de forma sencilla por los usuarios registrados. El acceso al perfil muestra la información detallada y estructurada del empleado de forma clara y accesible. Esta interfaz permite una experiencia fluida de consulta, sin pasos adicionales innecesarios.

#### Prompt

A partir del siguiente escenario de caso de uso, una breve descripción acerca del caso de uso y un formato esperado, genera un formato de pantalla que responda a las necesidades descritas y aporta una explicación de lo que has generado. Además, si es posible, proporciona el wireframe textual.

Descripción caso de uso: El responsable (administrador) de una aplicación web de reservas de actividades turísticas realiza la gestión de sus empleados a través de un panel de administración. Este responsable decide editar la información un empleado existente. Para ello, dispone de una lista con todos los empleados y a lado de cada fila un botón 'Editar empleado'. Cuando hace clic carga una pantalla con la información de dicho empleado en la que aparece nombre de usuario, nombre y apellidos, correo electrónico, número de teléfono, contraseña y rol. Tras editar los cambios hace clic en 'Confirmar

cambios'. Todos los campos son editables, excepto contraseña y rol.

Formato esperado: La interfaz debe incluir componentes principales (campos correspondientes, botón 'Confirmar cambios', etcétera). La disposición de los componentes.

Escenarios del caso de uso: Escenario de actualización de información de empleado

1. El responsable busca el empleado y hace clic sobre el icono de editar.
2. El sistema carga y muestra el perfil del empleado, con permisos de edición sobre los campos de la información del empleado.
3. El responsable selecciona lo(s) campo(s) que quiere editar.
4. El responsable edita y hace clic en 'Confirmar cambios'.
5. El sistema valida la información y la actualiza.

## Salida del modelo

### Formato de Pantalla

La siguiente es una posible implementación gráfica de la gestión de empleados en el panel de administración, siguiendo las necesidades del caso de uso:

#### ▪ 1. Pantalla inicial

- **Encabezado:** El título "Gestión de Empleados" o similar.
- **Lista de empleados:** Tabla con información básica (usuario, nombre y apellidos) que permita al administrador buscar y seleccionar el empleado cuya información editará. Cada fila incluye un botón "Editar".

#### ▪ 2. Pantalla de detalles del empleado

- **Formulario de edición:**
  - **Información básica:** Campos para editar nombre completo, correo electrónico y número de teléfono.
  - **Rol y contraseña:** Campos con información estática (no editables) para rol y contraseña. Puede haber un enlace "Cambiar contraseña" que muestre una pantalla separada donde el administrador puede generar un nuevo código.

- **Botón “Confirmar cambios”:** En la parte inferior de la pantalla, para realizar los cambios editados.

### **Explicación**

La implementación anterior pretende cumplir las necesidades del caso de uso. El responsable (administrador) puede buscar y seleccionar al empleado que desea editar en una lista con información básica. Una vez seleccionado el empleado, se carga su perfil completo en la pantalla con campos editables para nombre completo, correo electrónico y número de teléfono. Los campos relacionados con rol y contraseña están bloqueados debido a seguridad. El responsable puede realizar cambios en los campos editables.

### **Wireframe textual**

#### **Pantalla inicial:**

**Encabezado:** Gestión de empleados

#### **Lista de empleados:**

- + Fila (por cada empleado):
  - Nombre completo
  - Botón Editar

#### **Pantalla de detalles del empleado:**

#### **Formulario de edición:**

- Información básica:
  - + Campo nombre completo
  - + Campo correo electrónico
  - + Campo número de teléfono
  - Rol y contraseña: (no editables)
    - + Valor rol
    - + Enlace cambiar contraseña
- Botón Confirmar cambios

### **Pantalla para cambiar contraseña:**

#### **Formulario de cambio de contraseña:**

- Campo nueva contraseña
  - Campo repetir contraseña
- Botón Guardar

#### **Comentario**

Esta representación gráfica textual busca mostrar cómo la interfaz debe estar compuesta a partir de los diferentes componentes funcionales que se necesitan para cumplir las exigencias del caso de uso.

### **Prompt**

A partir del siguiente escenario de caso de uso, una breve descripción acerca del caso de uso y un formato esperado, genera un formato de pantalla que responda a las necesidades descritas y aporta una explicación de lo que has generado. Además, si es posible, proporciona el wireframe textual.

Descripción caso de uso: Un usuario anónimo o cliente decide realizar una reserva de alguna actividad turística que le interese. Para ello, primero tiene que acceder a un panel de todas las actividades disponibles, pudiendo ser filtradas por fecha, nombre o categoría. Elige una actividad que le interese y el sistema muestra una pantalla con toda la información de la actividad como fotos, categoría, duración, descripción, comentarios, fechas, precio por persona y un botón de 'Reservar actividad'. Cuando hace clic en dicho botón, se carga un formulario en el que se deben introducir los datos de la reserva como nombre y apellidos, correo electrónico y un número de teléfono, además en el formulario se muestra la información de la actividad. En dicho panel aparece un botón de 'realizar pago'

Formato esperado: La interfaz debe incluir componentes principales (campos correspondientes y entradas de datos, botones, etcétera). La disposición de los componentes.

Escenarios del caso de uso: Escenario de reserva de actividad turística

1. El usuario escoge la actividad que desea realizar

2. El sistema muestra los detalles de la actividad escogida
3. El usuario escoge la fecha de la actividad y comprueba su disponibilidad
4. El sistema verifica la disponibilidad y muestra que sí que está disponible la actividad para la fecha seleccionada
5. El usuario hace clic en 'Reservar'
6. El sistema redirige al usuario a rellenar los datos necesarios para la reserva
7. El usuario introduce los datos necesarios para la reserva
8. El usuario hace clic en 'Realizar pago'
9. El sistema redirige al usuario a una plataforma de pago
10. Finalizado el pago, el sistema guarda la reserva y muestra confirmación al usuario junto a la factura generada

## Salida del modelo

### Formato de Pantalla

- **1. Pantalla de actividades**
  - **Título:** "Actividades turísticas"
  - **Filtro por:**
    - Fecha
    - Nombre
    - Categoría
  - **Lista desplegable:** con todas las categorías disponibles.
- **2. Pantalla de detalles de la actividad**
  - Fotos de la actividad
  - Título y descripción de la actividad
  - Duración y categoría de la actividad
  - **Botón:** "Reservar actividad"
- **3. Pantalla de formulario de reserva**

- **Formulario con los siguientes campos:**
  - Nombre y apellidos del usuario
  - Correo electrónico del usuario
  - Número de teléfono del usuario
- Información adicional sobre la actividad (incluyendo el precio por persona)
- **Botón:** “Realizar pago”

#### ■ 4. Pantalla de pago

- Instrucciones para realizar el pago (dirección URL, etc.)
- Botones de cancelar o continuar con el pago

### Wireframe textual

#### 1. Pantalla de actividades y 2. Pantalla de detalles de la actividad:

```
1. Pantalla de actividades|
...
+-----+
| Actividades turísticas |
+-----+
| Fecha:                | Filtro por
| Nombre:               | Lista desplegable con categorías
| Categoría:            |
+-----+
...

2. Pantalla de detalles de la actividad
...
+-----+
| Fotos de la actividad |
| Título y descripción de la actividad |
| Duración y categoría de la actividad |
| Botón: "Reservar actividad" |
+-----+
```

Figura 58: Lista de actividades y selección de actividad. Fuente: Llama 3.1

#### 3. Pantalla de formulario de reserva y 4. Pantalla de pago:

```

3. Pantalla de formulario de reserva
...
-----+
| Formulario de reserva                |
| Nombre y apellidos del usuario      | Campo de texto
| Correo electrónico del usuario      | Campo de texto
| Número de teléfono del usuario      | Campo de texto
| Información adicional sobre la actividad | Botón: "Realizar pago"
-----+
...

4. Pantalla de pago
...
-----+
| Pago de reserva                      |
| Instrucciones para realizar el pago |
| Botones de cancelar o continuar con el |
| pago                                |
-----+
...

```

Figura 59: Introducir datos de reserva y pago. Fuente: Llama 3.1

### Explicación

El formato de pantalla que he propuesto incluye cuatro pantallas: la pantalla de actividades, la pantalla de detalles de la actividad, la pantalla de formulario de reserva y la pantalla de pago. Cada una de estas pantallas responde a los requisitos del escenario de caso de uso.

La pantalla de actividades permite al usuario seleccionar la actividad que desea realizar mediante un filtro por fecha, nombre o categoría. La lista desplegable con categorías proporciona una forma fácil de filtrar las actividades según su categoría.

La pantalla de detalles de la actividad muestra información adicional sobre la actividad seleccionada, como fotos, descripción y duración. El botón “Reservar actividad” permite al usuario avanzar al siguiente paso del proceso de reserva.

La pantalla de formulario de reserva requiere que el usuario ingrese sus datos personales (nombre y apellidos, correo electrónico y número de teléfono) antes de proceder con el pago.

Finalmente, la pantalla de pago proporciona instrucciones para realizar el pago y botones para cancelar o continuar con el proceso.

### Prompt

A partir del siguiente escenario de caso de uso, una breve descripción acerca del caso de uso y un formato esperado, genera un formato de pantalla que responda a las necesidades descritas y aporta una explicación de lo que has generado. Además, si es posible, proporciona el wireframe textual.

Descripción caso de uso: Un usuario anónimo o cliente decide realizar una reserva de alguna actividad turística que le interese. Para ello, ya ha seleccionado la actividad, introducido los datos necesarios en pantallas anteriores y está en la pasarela de pago. En esta pasarela aparece los datos de la operación como importe, comercio, fecha de operación y descripción de la compra. Además, aparece un formulario en el que el usuario elige entre dos métodos de pago, PayPal o Transferencia bancaria, y también tiene que introducir los datos correspondientes para la factura como nombre completo, CIF, dirección, código postal, población, país y provincia. Cuando elige el método de pago preferido y ha introducido los datos, tiene que hacer clic sobre dos posibles botones: 'Cancelar' y 'Realizar pago'. Este último botón lo redirige a una pasarela de pago del método que haya escogido.

Formato esperado: La interfaz debe incluir componentes principales (campos correspondientes y entradas de datos, botones, etcétera). La disposición de los componentes.

Escenarios del caso de uso: Escenario de pago de reserva de actividad turística

1. El sistema carga la pasarela de pago al usuario
2. El usuario elige entre dos métodos de pago; Paypal o Transferencia Bancaria
3. El usuario selecciona un método de pago
4. El usuario introduce los datos para el pago
5. El sistema proporciona campos para los datos necesarios para generar la factura
6. El usuario introduce los datos en los campos necesarios para Realizar pago
7. El usuario hace clic en 'Realizar pago'
8. El sistema procesa el pago
9. El sistema verifica el pago y muestra un mensaje de confirmación de pago, devolviendo confirmación y factura de pago de reserva de actividad turística

### Salida del modelo

#### Formato de pantalla esperado

**Título:** Realiza tu pago de la reserva de la actividad turística

**Imagen del fondo:** Imagen relacionada con el paisaje turístico o un símbolo que identifique a la empresa.

#### Componentes principales

## ■ 1. Formulario de pago

- Método de pago: PayPal / Transferencia bancaria
- Campos para los datos necesarios para realizar el pago:
  - Nombre completo
  - CIF
  - Dirección
  - Código postal
  - Población
  - País
  - Provincia

## ■ 2. Botones

- **Cancelar:** Volver a la pantalla anterior.
- **Realizar pago:** Continuar con el proceso de pago.

### Disposición de los componentes

En la parte superior del formulario se muestra la información relacionada con la reserva y el método de pago elegido. Los campos para introducir los datos necesarios para realizar el pago están organizados en una sección separada. A continuación, aparecen los botones “Cancelar” y “Realizar pago”.

### Imagen textual

```

**Imagen textual:**
...
+-----+
| Realiza tu pago de la reserva |
| de la actividad turística     |
+-----+
[Imagen del fondo]
Método de pago (PayPal/Transferencia bancaria)
Nombre completo          CIF
Dirección                Código postal
Población                País      Provincia
[Formulario para el pago]
... [Botón Cancelar]      [Botón Realizar pago]

```

Figura 60: Realizar pago de reserva de actividad turística. Fuente: Llama 3.1

### Wireframe textual

La pantalla se organiza en tres áreas principales:

- **Encabezado:** Título de la pantalla (“Realiza tu pago de la reserva de la actividad turística”) y un espacio para una imagen del fondo relacionada con el paisaje turístico o el símbolo identificador de la empresa.
- **Cuerpo:** Formulario de pago que incluye:
  - Método de pago: “PayPal” o “Transferencia bancaria”
  - Campos para datos del usuario: nombre completo, CIF, dirección, código postal, población, país y provincia
- **Pie:** Botones principales de acción del usuario: “Cancelar” y “Realizar pago”

### A.1.2. Mistral

Para evaluar el uso de este LLM se ha utilizado los mismos prompts de entrada de Llama 3.1. La estructura de salida es similar, para evitar diferencias.

#### Salida del modelo

Catálogo de requisitos funcionales:

Identificador	Título	Descripción
RQ001	Reserva Online	La aplicación web debe permitir la realización de reservas de actividades turísticas online sin necesidad de registrar una cuenta de usuario.
RQ002	Anónimo vs Registrado	La aplicación web debe distinguir entre usuarios anónimos y registrados.
RQ003	Historial de Reservas	La aplicación web debe permitir el acceso al historial de reservas tanto pasadas como futuras para cada cuenta de usuario.
RQ004	Métodos de Pago	La aplicación web debe ofrecer dos métodos de pago: transferencia bancaria y Paypal.

RQ005	Factura de Pago	La aplicación web debe generar una factura de pago al realizar una reserva.
RQ006	Actividades por Categorías	Las actividades se administrarán por categorías.
RQ007	Gestión de Actividades	La aplicación web debe permitir la gestión de las actividades y calendario de actividades que se asigna a un monitor.
RQ008	Gestión de Monitores	La aplicación web debe permitir la gestión de los monitores de la empresa y su calificación.
RQ009	Comentarios en las Actividades	Los usuarios pueden dejar comentarios en las actividades.
RQ010	Registro de Usuario	La aplicación web debe almacenar nombre, apellidos, DNI o NIE, correo electrónico y teléfono móvil para cada cuenta de usuario.
RQ011	Gestión de Empleados	La aplicación web debe permitir que los empleados gestionen las reservas online y realicen otras gestiones relacionadas con monitores y actividades.
RQ012	Gestión de Empleados - Monitores	La aplicación web debe almacenar la información adicional de los empleados que se encargan de la gestión de monitores, como estudios y capacitación.
RQ013	Análisis de Datos - Reservas	La aplicación web debe mantener un histórico de las reservas que se hayan realizado en la empresa para el análisis de datos.
RQ014	Análisis de Datos - Tabla y Gráficas	La aplicación web debe incorporar herramientas de análisis de datos que permitan visualizar datos en gráficas y tablas.
RQ015	Actualización de Precios y Duraciones	La aplicación web debe permitir actualizar los precios y duraciones de las actividades en casos justificados.

**Salida del modelo**

Catálogo de requisitos no funcionales:

<b>Identificador</b>	<b>Título</b>	<b>Descripción</b>
RNF-TPB	Tipo de Pago de la Actividad	La aplicación Web permitirá dos métodos de pago (transferencia bancaria y Paypal) para realizar las reservas turísticas.
RNF-IHC	Información Histórica de Cliente	La aplicación Web mantendrá un histórico de las reservas que se hayan realizado en la empresa para las tareas de análisis de datos e historial de reservas de cada cliente.
RNF-HID	Horario Inactividad Día	Para cada actividad se almacenará la hora y día de la semana en la que se realiza dicha actividad.
RNF-CCA	Categorías de Actividades	Las actividades dispondrán de precios y duraciones fijas, pero se administrarán las actividades por categorías.
RNF-RAC	Registro de Actividad Cliente	El sistema permitirá la verificación del historial de reservas para cada cliente.
RNF-UAS	Usuario Anónimo o Registrado	La aplicación Web permitirá la realización de reservas sin necesidad de registrarse como cliente, pero existirán usuarios registrados que podrán tener un historial de reservas.
RNF-GGE	Gestión General de Empleados	El responsable de la empresa podrá gestionar sus empleados en la plataforma Web.
RNF-GEM	Gestión Especial de Empleados	Un empleado podrá realizar reservas para clientes en el momento y gestionar las reservas en el sistema.
RNF-GAR	Gestión de Análisis de Datos	Los usuarios (empleados y responsable de la empresa) pueden utilizar herramientas de análisis de datos incorporadas en la plataforma Web para visualizar datos en gráficas y tablas.

RNF-RCA	Registro de Comentarios en Actividades	Los usuarios pueden dejar comentarios sobre las actividades y los empleados de la empresa serán capaces de revisar estos comentarios y borrarlos si se consideran inapropiados.
RNF-CQE	Cualificado o No para actividad	Para la gestión de monitores se almacena la misma información, pero con el añadido de que se requieren los estudios para gestionar la asignación de actividades en función de si está cualificado o no para llevar a cabo dicha actividad.
RNF-OIA	Organización Información Académica	Para la gestión de monitores se almacenará la información del nombre, apellidos, DNI o NIE, correo electrónico y teléfono móvil.

### Casos de uso

Para los casos de uso se ha hecho uso del mismo prompt. La salida es la siguiente:

**Salida del modelo**

**Actores**

- Responsable de la aplicación Web (Administrador)
- Empleados de la empresa turística
- Monitores
- Usuarios anónimos y registrados

\*Casos de uso

1. **Gestión de Empleados** (Admin, Empleados)
  - Creación de empleados
  - Lectura de información de empleados
  - Actualización de información de empleados
  - Eliminación de empleados

2. **Gestión de Monitores** (Empleados)
  - Creación de monitores
  - Lectura de información de monitores
  - Actualización de información de monitores
  - Eliminación de monitores
3. **Gestión de reservas online** (Empleados)
  - Lectura de reservas realizadas
4. **Reservar por parte de empleados hacia clientes** (Empleados)
  - Realización de reservas para clientes en el momento
5. **Gestión de Actividades Turísticas** (Empleados)
  - Creación de nuevas actividades turísticas
  - Lectura, actualización y eliminación de información de actividades turísticas
6. **Gestión de Actividades de Monitores** (Empleados)
  - Gestionar las actividades que cada monitor realiza por día
  - Asignar actividades a cada monitor
  - Ver listado de monitores
7. **Análisis de datos** (Admin, Empleados)
  - Interactuar con los datos mediante la lectura de gráficas y tablas generadas
8. **Gestión de pagos online** (Usuarios)
  - Realizar pagos de las reservas de actividades turísticas usando dos métodos:  
PayPal o transferencia bancaria
  - Generar factura de pago opcionalmente
9. **Gestión de comentarios de actividades** (Empleados)

- Lectura y eliminación de los comentarios que haya por actividad
10. **Actualizar precios de actividades** (Empleados)
    - Actualización de los precios de las actividades turísticas
  11. **Ver listado de reservas pasadas** (Usuarios registrados)
    - Acceso a un listado de las reservas de actividades turísticas que realizaron en el pasado y aún no se han realizado pero se reservaron para futuro
  12. **Crear comentarios** (Clientes registrados)
    - Dejar comentarios en las actividades turísticas
  13. **Responder comentarios** (Clientes registrados, Empleados)
    - Respuesta a los comentarios existentes en las actividades turísticas
  14. **Buscar actividades** (Usuarios, registrados o no)
    - Capacidad de buscar actividades por categoría o nombre
  15. **Realizar reservas** (Usuarios, registrados o no)
    - Acceso a la aplicación Web para realizar reservas de las actividades turísticas disponibles en cada día
  16. **Crear cuenta y registro** (Usuarios anónimos)
    - Registro como cliente en el sistema
  17. **Iniciar sesión** (Usuarios registrados)
    - Acceso al mismo sistema mediante las credenciales introducidas en el registro de su cuenta
  18. **Cerrar sesión** (Usuarios registrados)
    - Desconexión como usuario registrado en el sistema

## Especificación de casos de uso

### Salida del modelo

Salida en las tablas con la estructura:

<b>Título</b>	Registro de Usuario para Usuario Anónimo
<b>Descripción</b>	Permite que un usuario anónimo cree una cuenta en el sistema completando un formulario con sus datos básicos.
<b>Pre-condición</b>	<ul style="list-style-type: none"> <li>▪ El usuario no está registrado en el sistema.</li> <li>▪ El sistema permite la creación de cuentas a través del registro como usuario anónimo.</li> </ul>
<b>Post-condición</b>	<ul style="list-style-type: none"> <li>▪ El usuario ha completado el registro exitosamente.</li> <li>▪ La información básica del usuario ha sido guardada en la base de datos.</li> <li>▪ El usuario es redirigido a la página de inicio.</li> </ul>
<b>Prioridad</b>	Alta
<b>Escenario principal</b>	

### Escenario de registro de nuevo usuario

1. El usuario anónimo accede al sitio web y selecciona el botón Registrarse”.
2. El sistema redirige al formulario de registro.
3. El usuario ingresa nombre, correo electrónico, contraseña y confirma su edad.
4. El sistema valida que el correo electrónico no esté repetido.
5. Si todo es correcto, guarda los datos en la base de datos.
6. El usuario es redirigido a la página de inicio con un mensaje de éxito.

### Escenarios alternativos

#### Escenario alternativo

- **1. Correo ya registrado:** El sistema muestra un mensaje indicando que el correo ya está en uso y solicita ingresar otro.
- **2. Contraseña no coincide:** El sistema muestra un mensaje de error y solicita ingresar la misma contraseña en ambos campos.

### Clases de análisis

#### A. Clases de entidad

- Usuario
- Correo electrónico
- Contraseña
- Cuenta de usuario

<b>B. Clases de control</b>	<ul style="list-style-type: none"> <li>▪ RegistroController</li> </ul>
<b>C. Clases de interfaz</b>	<ul style="list-style-type: none"> <li>▪ Interfaz de registro (formulario)</li> </ul>

<b>Título</b>	Empleado Management Use Case
<b>Descripción</b>	Caso de uso que permite al administrador del sistema realizar operaciones de creación, consulta, actualización y eliminación (CRUD) sobre los empleados de la empresa mediante la aplicación web.
<b>Pre-condición</b>	<ul style="list-style-type: none"> <li>▪ El actor responsable de la aplicación web tiene acceso al sistema.</li> <li>▪ El sistema está funcionando correctamente y los datos necesarios están disponibles.</li> </ul>
<b>Post-condición</b>	<ul style="list-style-type: none"> <li>▪ La operación deseada sobre el empleado se ha completado exitosamente.</li> <li>▪ El sistema guarda los cambios realizados en la base de datos.</li> </ul>
<b>Prioridad</b>	Alta
<b>Escenario principal</b>	

### **Escenario de creación de empleado**

1. El actor accede a la aplicación web y selecciona "Crear empleado".
2. El sistema muestra un formulario con campos: nombre, apellido, correo electrónico, puesto y sueldo.
3. El actor completa el formulario y presiona "Guardar".
4. El sistema valida la información. Si hay errores, muestra un mensaje.
5. Si todo es válido, el sistema crea una nueva entidad Empleado y guarda el registro.
6. El sistema muestra un mensaje de éxito.

### **Escenarios alternativos**

**Escenario de consulta, actualización y eliminación de empleados**

**Lectura (Read):**

1. El actor selecciona "Ver empleados".
2. El sistema muestra la lista de empleados con sus datos.

**Actualización (Update):**

1. El actor selecciona "Actualizar empleado".
2. El sistema muestra los datos del empleado seleccionado.
3. El actor modifica los datos y pulsa "Guardar".
4. El sistema valida los datos. Si hay errores, muestra mensaje.
5. Si es válido, guarda los cambios y muestra mensaje de éxito.

**Eliminación (Delete):**

1. El actor selecciona "Eliminar empleado".
2. El sistema muestra la lista de empleados.
3. El actor elige al empleado a eliminar y pulsa "Eliminar".
4. El sistema valida si hay relaciones dependientes. Si las hay, muestra error.
5. Si no hay, elimina el registro y muestra mensaje de éxito.

**Clases de análisis**

**A. Clases de entidad**

- Empleado (ID, Nombre, Apellido, Correo electrónico, Puesto, Sueldo)

<b>B. Clases de control</b>	<ul style="list-style-type: none"> <li>▪ EmpleadoController (lógica de negocio CRUD para empleados)</li> </ul>
<b>C. Clases de interfaz</b>	<ul style="list-style-type: none"> <li>▪ WebEmployeeInterface (interfaz web con el sistema)</li> <li>▪ DatabaseEmployeeInterface (comunicación con la base de datos)</li> </ul>

<b>Título</b>	Reserva de Actividad Turística
<b>Descripción</b>	Permite a un usuario anónimo reservar una actividad turística desde la plataforma web, introduciendo sus datos personales y recibiendo una confirmación por correo electrónico.
<b>Pre-condición</b>	<ul style="list-style-type: none"> <li>▪ El sistema web debe estar disponible y funcionando correctamente.</li> <li>▪ El usuario anónimo debe tener acceso a una conexión a internet.</li> <li>▪ La actividad turística deseada debe estar disponible en el sistema.</li> </ul>
<b>Post-condición</b>	<ul style="list-style-type: none"> <li>▪ El usuario ha realizado la reserva exitosamente.</li> <li>▪ El sistema ha enviado un número de confirmación al usuario por correo electrónico.</li> </ul>
<b>Prioridad</b>	Alta

**Escenario principal**

**Escenario de reserva de actividad turística**

1. El usuario accede al sitio web y selecciona "Reservar actividad".
2. El sistema muestra la lista de actividades turísticas disponibles.
3. El usuario selecciona una actividad y hace clic en "Siguiente".
4. El sistema muestra un formulario para introducir: nombre, correo electrónico, teléfono, fecha de nacimiento y número de pasaporte.
5. El usuario confirma los datos ingresados.
6. Si los datos son válidos, el sistema:
  - Guarda la reserva en la base de datos.
  - Envía un correo con el número de confirmación y detalles de la actividad.
7. Si hay errores en los datos, se solicita corregirlos.

**Escenarios alternativos**

**Escenario alternativo**

- **1a.** El usuario no tiene conexión a internet → No puede realizar la reserva.
- **1b.** El sistema web no está disponible → No se puede acceder a las actividades.
- **2a.** La actividad no está disponible → Se solicita al usuario elegir otra.
- **3a.** El usuario no completa todos los campos del formulario → Se solicita rellenar los campos obligatorios.
- **4b.** Los datos del formulario son incorrectos → Se solicita corregirlos antes de continuar.
- **5a.** El usuario cierra la sesión antes de completar la reserva → No se guarda ningún registro.

**Clases de análisis**

**A. Clases de entidad**

- Usuario Anónimo
- Actividad Turística
- Reserva (fecha de reserva, número de confirmación, usuario)

**B. Clases de control**

- ReservarActividadController
- GestionarReservasController

<b>C. Clases de interfaz</b>	<ul style="list-style-type: none"> <li>▪ FormularioReserva</li> <li>▪ Confirmación de reserva</li> </ul>
------------------------------	--

<b>Título</b>	Pago de Reserva de Actividad Turística (Anónimo) Use Case
<b>Descripción</b>	Caso de uso que permite al usuario anónimo realizar el pago de una actividad turística reservada a través de un método válido de pago. El sistema gestiona la transacción, confirma la reserva y actualiza la información asociada al usuario.
<b>Pre-condición</b>	<ul style="list-style-type: none"> <li>▪ El usuario anónimo está conectado a la página web de la empresa turística.</li> <li>▪ La actividad turística deseada está disponible y con capacidad suficiente.</li> <li>▪ El usuario dispone de un método de pago válido.</li> <li>▪ La información del usuario en el sistema está actualizada (por ejemplo, idioma de preferencia).</li> </ul>
<b>Post-condición</b>	<ul style="list-style-type: none"> <li>▪ La reserva ha sido confirmada y asignada al usuario anónimo.</li> <li>▪ El usuario ha recibido una confirmación de la transacción exitosa.</li> <li>▪ La información del sistema ha sido actualizada para reflejar la reserva.</li> </ul>

<b>Prioridad</b>	Alta
<b>Escenario principal</b>	
<b>Escenario de pago de actividad turística</b>	
<ol style="list-style-type: none"> <li>1. El usuario anónimo accede a la página y selecciona la actividad deseada.</li> <li>2. El sistema muestra los detalles y solicita el pago.</li> <li>3. El usuario elige un método de pago válido y completa los datos.</li> <li>4. El sistema verifica la validez de los datos ingresados.</li> <li>5. Si el pago es válido, se genera la reserva y se envía una confirmación.</li> <li>6. El usuario accede a su cuenta (si existe) o se registra para ver los detalles.</li> <li>7. Si el pago es inválido, el sistema solicita nuevos datos.</li> <li>8. El proceso se repite hasta que la reserva sea completada o cancelada.</li> </ol>	
<b>Escenarios alternativos</b>	
<b>Escenario alternativo</b>	
<ul style="list-style-type: none"> <li>▪ 1. El usuario no tiene fondos suficientes → La plataforma notifica el error y permite reintentar más tarde.</li> <li>▪ 2. La actividad seleccionada ya no tiene plazas → El sistema informa al usuario y ofrece alternativas.</li> <li>▪ 3. El usuario introduce datos de pago incorrectos → El sistema solicita nuevos datos o permite reintento posterior.</li> </ul>	
<b>Clases de análisis</b>	

<p><b>A. Clases de entidad</b></p>	<ul style="list-style-type: none"> <li>▪ Usuario (ID, nombre, correo, método de pago, idioma)</li> <li>▪ Actividad turística (ID, nombre, descripción, capacidad, precio, fecha, horario)</li> <li>▪ Reserva (ID, usuario, actividad, fecha, hora, número de confirmación)</li> <li>▪ Pago (método, cuenta, tarjeta o número de teléfono)</li> </ul>
<p><b>B. Clases de control</b></p>	<ul style="list-style-type: none"> <li>▪ Controlador de reservas</li> </ul>
<p><b>C. Clases de interfaz</b></p>	<ul style="list-style-type: none"> <li>▪ Interface de pago</li> </ul>

### Diagramas de secuencia

#### Salida del modelo

1. **Usuario Anónimo** → *ViewPaginaPrincipal*: Acción de navegar a la página web.
2. *ViewPaginaPrincipal*: Acción de mostrar la página de inicio y el botón *Registrarse*.
3. **Usuario Anónimo** → *ViewPaginaPrincipal*: Hace clic en el botón *Registrarse*.
4. *ViewPaginaPrincipal* → *ControllerUsuarios*: Evento de clic en botón *Registrarse*.
5. *ControllerUsuarios* → *ViewRegistroUsuario*: Acción de cargar y mostrar el formulario de registro.
6. **Usuario Anónimo** → *ViewRegistroUsuario*: Acción de rellenar los campos del formulario de registro.
7. **Usuario Anónimo** → *ViewRegistroUsuario*: Hace clic en *Crear cuenta*.

8. *ViewRegistroUsuario* → *ControllerUsuarios*: Evento de clic en botón *Crear cuenta*.
9. *ControllerUsuarios* → *Modelos (Usuarios o Clientes)*: Consulta para validar los datos introducidos por el usuario.
10. *Modelos (Usuarios o Clientes)* → *ControllerUsuarios*: Respuesta de la consulta de validación.
11. Si los datos no se encuentran en la base de datos y se ajustan al formato válido:
  - *ControllerUsuarios* → *Modelos (Usuarios o Clientes)*: Consulta para crear una nueva cuenta con los datos introducidos por el usuario.
  - *Modelos (Usuarios o Clientes)* → *ControllerUsuarios*: Respuesta de la consulta de creación de cuenta.
  - *ControllerUsuarios* → *ViewIniciarSesión*: Acción de mostrar mensaje de confirmación al usuario y redirigirlo a la ventana de inicio de sesión.
12. Si los datos introducidos ya se encuentran en la base de datos:
  - *ControllerUsuarios* → *ViewRegistroUsuario*: Acción de mostrar mensaje de error de cuenta ya existente.
13. Si los datos introducidos no se ajustan al formato válido:
  - *ControllerUsuarios* → *ViewRegistroUsuario*: Acción de mostrar mensaje de error de datos inválidos.

### Salida del modelo

1. **Responsable (Actor)** → *ViewPaginaAdmin (Vista)*: acción (*Añadir empleado*).
  - **ControllerUsuarios (Controlador)**: evento (*Crear empleado*).
  - **Modelo de Empleados (Modelo)**: consulta (*Preparar formulario para introducir datos del nuevo empleado*).
2. **Responsable** → *Formulario del empleado (Vista dentro de ViewPaginaAdmin)*: acción (*Introducir datos del nuevo empleado*).

- **ControllerUsuarios:** evento (*Validar datos introducidos*).
  - **Modelo de Empleados:** consulta (*Verificar y validar los datos introducidos en el formulario*).
3. **Modelo de Empleados** → **ControllerUsuarios:** respuesta (*Los datos son válidos o no*).
- Si los datos introducidos no son válidos:
    - **ControllerUsuarios:** evento (*Mostrar mensaje de error y volver a mostrar el formulario del empleado*).
    - Repetir desde el paso 2.
  - Si los datos introducidos son válidos:
    - **ControllerUsuarios:** evento (*Guardar la información en la base de datos y redirigir a la página web de la empresa*).
4. **Responsable** → **ViewEmpleados (Vista):** acción (*Borrar empleado*).
- **ControllerUsuarios:** evento (*Eliminar empleado*).
  - **Modelo de Empleados:** consulta (*Obtener los datos del empleado a borrar*).
5. **Modelo de Empleados** → **ControllerUsuarios:** respuesta (*Los datos del empleado están disponibles para su eliminación o no*).
- Si los datos del empleado están disponibles:
    - **ControllerUsuarios:** evento (*Mostrar mensaje de aviso y confirmar la acción de borrado*).
    - **Responsable** → **ViewEmpleados (Vista):** acción (*Confirmar la acción de borrado o Cancelar*).
  - Si los datos del empleado no están disponibles:
    - **ControllerUsuarios:** evento (*Mostrar mensaje de error y volver a mostrar la lista de empleados*).
    - Repetir desde el paso 4.

6. Si el responsable confirma la acción de borrado:

- **Modelo de Empleados (Modelo):** consulta (*Eliminar los datos del empleado en la base de datos*).
- **ControllerUsuarios:** evento (*Mostrar mensaje de confirmación del borrado y volver a mostrar la lista de empleados*).

7. **Responsable** → *ViewEmpleados (Vista): acción (Leer información de empleado)*.

- **ControllerUsuarios:** evento (*Obtener los datos del empleado seleccionado*).
- **Modelo de Empleados:** consulta (*Buscar los datos del empleado seleccionado en la base de datos*).

8. **Modelo de Empleados** → **ControllerUsuarios:** respuesta (*Los datos del empleado están disponibles o no*).

- Si los datos del empleado están disponibles:
  - **ControllerUsuarios:** evento (*Cargar y mostrar el perfil del empleado en la vista de ViewEmpleados*).
- Si los datos del empleado no están disponibles:
  - **ControllerUsuarios:** evento (*Mostrar mensaje de error y volver a mostrar la lista de empleados*).
  - Repetir desde el paso 7.

9. **Responsable** → *ViewEmpleados (Vista): acción (Actualizar información del empleado)*.

- **ControllerUsuarios:** evento (*Editar el perfil del empleado*).
- **Modelo de Empleados:** consulta (*Obtener los datos actuales del empleado en la base de datos*).

10. **Modelo de Empleados** → **ControllerUsuarios:** respuesta (*Los datos del empleado están disponibles o no*).

- Si los datos del empleado están disponibles:
  - **ControllerUsuarios:** evento (*Cargar el formulario con los datos actuales del empleado en la vista de ViewEmpleados*).
  - **Responsable** → *Formulario del empleado (Vista dentro de ViewEmpleados): acción (Introducir los nuevos datos y confirmarlos)*.
- Si los datos del empleado no están disponibles:
  - **ControllerUsuarios:** evento (*Mostrar mensaje de error y volver a mostrar la lista de empleados*).
  - Repetir desde el paso 9.

11. Si el responsable confirma los nuevos datos del empleado:

- **Modelo de Empleados (Modelo):** consulta (*Actualizar los datos del empleado en la base de datos*).
- **ControllerUsuarios:** evento (*Mostrar mensaje de confirmación y volver a mostrar la lista de empleados*).

### Salida del modelo

1. **Actor (Usuario Anónimo)** → *ViewPaginaPrincipal (Vista): acciona la opción de Reservar Actividad Turística.*
2. *ViewActividades (Vista)* → **ControllerActividades (Controlador):** evento (*Recibe petición de mostrar actividades*).
3. **ControllerActividades (Controlador)** → **Modelo Actividades:** consulta (*Solicita los datos de las actividades disponibles*).
4. **Modelo Actividades** → **ControllerActividades:** respuesta (*Envía los datos de las actividades disponibles a la vista para su presentación*).
5. *ViewActividades (Vista)* → **Actor (Usuario Anónimo):** muestra las actividades disponibles.

6. **Actor (Usuario Anónimo)** → *ViewActividades (Vista)*: selecciona una actividad para reservar.
7. *ViewActividades (Vista)* → **ControllerActividades**: evento (*Recibe petición de más información sobre la actividad seleccionada*).
8. **ControllerActividades** → **Modelo Actividades**: consulta (*Solicita los detalles de la actividad seleccionada*).
9. **Modelo Actividades** → **ControllerActividades**: respuesta (*Envía los datos detallados de la actividad a la vista para su presentación*).
10. *ViewActividades (Vista)* → **Actor (Usuario Anónimo)**: muestra los detalles de la actividad seleccionada.
11. **Actor (Usuario Anónimo)** → *ViewActividades (Vista)*: selecciona una fecha para la actividad y comprueba su disponibilidad.
12. *ViewActividades (Vista)* → **ControllerActividades**: evento (*Recibe petición de comprobar disponibilidad*).
13. **ControllerActividades** → **Modelo Actividades**: consulta (*Solicita la disponibilidad de la actividad para la fecha seleccionada*).
14. **Modelo Actividades** → **ControllerActividades**: respuesta (*Envía la información de disponibilidad a la vista*).
15. *ViewActividades (Vista)* → **Actor (Usuario Anónimo)**: muestra que la actividad está disponible para la fecha seleccionada.
16. Si la actividad no está disponible:
  - *ViewActividades (Vista)* → **Actor (Usuario Anónimo)**: informa al usuario que no hay disponibilidad para esa actividad con la fecha indicada.
17. Si el usuario decide escoger otra fecha o salir de la actividad seleccionada:

- *ViewActividades (Vista) → ControllerActividades: evento (Recibe petición de mostrar actividades con otra fecha o ir a la vista principal).*
18. **Actor (Usuario Anónimo) → ViewActividades (Vista):** selecciona otra fecha y continúa o vuelve a la vista principal.
  19. Si el usuario hace clic en *Reservar*:
    - *ViewReservas (Vista) → ControllerReservas (Controlador): evento (Recibe petición de mostrar el formulario de reserva).*
  20. **Actor (Usuario Anónimo) → ViewReservas (Vista):** rellena los datos necesarios para la reserva y hace clic en *Realizar pago*.
  21. *ViewReservas (Vista) → ControllerReservas: evento (Recibe petición de redirigir al usuario a la vista de pago).*
  22. Si el usuario deja campos obligatorios sin completar o introduce datos inválidos:
    - *ViewReservas (Vista) → Actor (Usuario Anónimo):* muestra un mensaje de error indicando los campos a corregir.
  23. Si el usuario corrige los datos y continúa o decide salir del formulario de reserva:
    - *ViewReservas (Vista) → ControllerReservas: evento (Recibe petición de continuar o ir al inicio de la actividad).*
  24. Si finaliza el pago:
    - *ViewPago (Vista) → ControllerReservas: evento (Recibe petición de guardar la reserva y mostrar confirmación con la factura generada).*
  25. **ControllerFactura (Controlador) → Modelo Factura:** consulta (*Solicita los datos necesarios para crear la factura*).
  26. **Modelo Factura → ControllerFactura:** respuesta (*Envía los datos de la factura a la vista*).

27. *ViewPago (Vista)* → **Actor (Usuario Anónimo)**: muestra confirmación con la factura generada.

### Salida del modelo

1. **Usuario Anónimo (UA)** interacciona con **ViewPago (VP)**: Inicia sesión
  - **VistaPago**: Carga la pasarela de pago
2. Si el usuario no ha iniciado sesión, entra en un ciclo repetitivo de solicitudes para hacerlo hasta que lo hace o decide cancelar
  - **UA**: Intenta iniciar sesión
  - **VistaPago**: Verifica si el usuario ha iniciado sesión
    - Si el usuario no ha iniciado sesión, **VistaPago**: Solicita credenciales de inicio de sesión
    - **UA**: Intenta introducir las credenciales correctamente
      - Si la credencial es incorrecta, **VistaPago**: Muestra un mensaje de error y solicita nuevamente las credenciales
    - Si el usuario ha iniciado sesión, continúa con el siguiente paso
3. **UA** interacciona con **ViewPago**: Selecciona entre Paypal o Transferencia Bancaria
  - **VistaPago**: Muestra los dos métodos de pago y permite su selección
4. **UA** interacciona con **ViewPago**: Elige un método de pago
  - **VistaPago**: Guarda la elección del método de pago
5. **UA** interacciona con **ViewPago**: Introduce los datos necesarios para el pago (en caso de Transferencia Bancaria, los datos serían diferentes a los de Paypal)
  - **VistaPago**: Proporciona campos para introducir los datos del pago y generar la factura
6. **UA** interacciona con **ViewPago**: Realiza el pago

- **VistaPago:** Valida que se hayan completado todos los campos necesarios y envía la información al **ControladorPago (CP)** para procesamiento
7. Si no hay error en el proceso de pago, continúa con el siguiente paso; caso contrario, entra en un ciclo repetitivo de solicitudes hasta que se resuelva o el usuario cancela la transacción
- **ControladorPago:** Procesa el pago
    - Si hay error técnico con la pasarela de pago o la tarjeta es rechazada, **CP:** Muestra un mensaje al usuario indicando el problema y solicita una acción del usuario (reintentar, seleccionar otro método de pago o cancelar)
    - **UA** interacciona con **ViewPago:** Solicita una acción (reintentar, seleccionar otro método de pago o cancelar)
      - Si el usuario reintentar, continúa desde el paso 6, caso contrario entra en un ciclo repetitivo de solicitudes hasta que se resuelva o el usuario cancela la transacción
8. **ControladorPago:** Verifica el pago y muestra un mensaje de confirmación de pago
- **VistaPago:** Muestra un mensaje de confirmación de pago con la factura de pago de reserva de actividad turística
- 4a. **Escenario alternativo: cancelación de reserva antes del pago**
- **UA** interacciona con **ViewPago:** Cierra la página o hace clic en Cancelar
  - **VistaPago:** Se muestra un mensaje de confirmación de cancelación de reserva de actividad turística
- 4b. **Escenario alternativo: cancelación de reserva después del pago (cancelado por el sistema)**
- **ControladorPago:** Detecta que el pago no fue completado correctamente y muestra un mensaje al usuario indicando el problema

- **UA** interacciona con **ViewPago**: Solicita una acción del usuario (reintentar, seleccionar otro método de pago o cancelar)
  - Si el usuario reintenta, continúa desde el paso 6, caso contrario el sistema descarta la reserva y se muestra un mensaje de confirmación de cancelación de reserva de actividad turística

### **Modelos de clase**

No se ha tenido en cuenta para los LLMs.

### **Modelo físico de datos**

No se tiene en cuenta para esta fase.

### **Formatos de pantalla**

Se han utilizado los mismos prompts.

## **Salida del modelo**

### **Historia de Usuario e Interfaz de Usuario**

Basado en el escenario proporcionado, he creado una historia de usuario y un diseño de interfaz (wireframe textual) para abordar las necesidades descritas:

#### **Historia de Usuario**

**Título:** Registro de Usuario

Como nuevo usuario,

Quiero registrarme en la plataforma web de reservas de actividades turísticas,

Para poder hacer reservas de distintas actividades.

#### **Criterios de Aceptación**

1. Cuando el usuario hace clic en “Registrarse” en la página de inicio, el sistema debe mostrar el formulario de registro.
2. El formulario de registro debe incluir campos para Nombre, Apellidos, Nombre de usuario, Correo electrónico, Número de teléfono, Contraseña y DNI (Documento Nacional de Identidad).
3. Debe haber un botón con la etiqueta “Crear cuenta” y otro botón con la etiqueta “Iniciar sesión ahora”.

4. Cuando el usuario completa el formulario y hace clic en “Crear cuenta”, el sistema valida los datos introducidos.
5. Si los datos son válidos, el sistema crea la cuenta y muestra un mensaje de éxito al usuario.
6. El sistema debe redirigir al usuario a la página de inicio de sesión tras un registro exitoso.
7. Si hay errores en los datos introducidos, el sistema debe mostrar un mensaje de error y permitir que el usuario los corrija antes de volver a enviar el formulario.

```
Wireframe Textual:
---
|-----|
| Tour Activities Registration |
|-----|
|
| Name
| Surname
| Username
| Email
| Phone number
| Password
| DNI
|
| Create Account | Login Now
|-----|
---
```

Figura 61: Registro de nuevo usuario. Fuente: Mistral

### Salida del modelo

#### Diseño de Interfaz de Usuario (UI) para Añadir un Nuevo Empleado

Basado en el escenario proporcionado, he generado un diseño de interfaz de usuario para añadir un nuevo empleado al sistema. A continuación se muestra la descripción textual del wireframe:

**Título:** Añadir empleado

- **1. Botón “Añadir Empleado”:** Ubicado en el menú de navegación o en el panel principal (dashboard), este botón inicia el proceso para añadir un nuevo empleado.

- **2. Diseño del Formulario:** El formulario consiste en campos para las siguientes entradas de datos:
  - Nombre de usuario (Username)
  - Nombre y apellidos (Name and Surname)
  - Email
  - Teléfono (Phone number)
  - Contraseña (Password) (campo oculto)
  - DNI (Documento de identidad)
  - Rol (Role) con dos opciones: Empleado, Monitor
- **3. Validación de entradas:** La validación en tiempo real garantizará que los datos introducidos cumplan con ciertos requisitos de formato y longitud, como una dirección de correo válida o un formato correcto del DNI.
- **4. Botón “Registrar empleado”:** Después de completar todos los campos requeridos, este botón permite al usuario responsable enviar el formulario y añadir el nuevo empleado al sistema.
- **5. Mensaje de confirmación:** Tras un envío exitoso, el sistema muestra un mensaje de confirmación indicando que el empleado ha sido añadido a la base de datos. Este mensaje también puede incluir el nombre de usuario o el ID del nuevo empleado como referencia.

### Salida del modelo

#### **Título: Diseño de Pantalla para el Caso de Uso de Eliminación de Empleados**

El formato presentado incluye una interfaz con componentes principales que permitirán al responsable administrar los empleados a través del panel de administración de la aplicación web de reservas turísticas. El formato se basa en el escenario del caso de uso proporcionado, donde un empleado existente puede ser eliminado.

#### **Características del diseño de pantalla**

- **1. Lista de empleados:** Se muestra una tabla con todos los empleados disponibles en la interfaz (campo de datos).
- **2. Botón “Eliminar empleado”:** En cada fila de la lista hay un botón que permite al responsable seleccionar un empleado para eliminar (botón de acción).
- **3. Mensaje de confirmación:** Al hacer clic en “Eliminar empleado”, aparece un mensaje de confirmación en pantalla con dos botones: “Aceptar” y “Cancelar” (mensaje de alerta).
- **4. Eliminación y feedback:** Si se confirma la eliminación, el sistema borra la información del empleado seleccionado y muestra un mensaje de éxito confirmando el borrado (mensaje de retroalimentación).

### Wireframe textual

```

Wireframe textual:
...
Employee Management Panel - Admin

List of Employees
Employee Name | Eliminar empleado botón |
Employee Role | |
...
...
...
Confirm Delete Employee | Aceptar botón |
Are you sure you want to | Cancelar botón |
delete this employee?
Yes, I am sure
No, cancel delete

Employee has been deleted
successfully.
    
```

Figura 62: Eliminar empleado existente. Fuente: Mistral

### Salida del modelo

#### Formato de pantalla para el escenario de lectura de información de empleado

**Título:** Listado de empleados

#### Componentes principales

- **1. Filtro de búsqueda:** Campo de entrada para buscar por nombre de usuario, nombre o apellidos.
- **2. Tabla de empleados:** Incluye las siguientes columnas:

- Nombre de usuario
  - Nombre y Apellidos
  - Estado (Activo/Inactivo)
  - Acciones
- **3. Botón “Agregar empleado”:** Permite al responsable añadir un nuevo empleado a la lista.
  - **4. Botones “Ver empleado”:** Disponibles en cada fila de la tabla, junto a las columnas de Nombre y Apellidos.

**Título:** Perfil del empleado

#### **Componentes principales**

- **1. Botón “Atrás”:** Permite regresar a la pantalla de listado de empleados.
- **2. Título del perfil:** Contiene el nombre completo del empleado.
- **3. Campos informativos:** Se muestran con las siguientes etiquetas y datos:
  - Nombre de usuario
  - Nombre
  - Apellidos
  - Correo electrónico
  - Teléfono móvil
  - Estado (Activo/Inactivo)
- **4. Botón “Editar información”:** Permite modificar los datos del empleado.
- **5. Botón “Eliminar empleado”:** Permite quitar al empleado de la lista.

## Salida del modelo

### Formato de pantalla: Edición de información del empleado

Esta interfaz está diseñada para cumplir con los requisitos del caso de uso. Se compone de cuatro secciones: visualización superior, edición inferior, validación y confirmación de cambios.

#### 1. Sección Superior

- Nombre de usuario (editable)
- Nombre completo (editable)
- Correo electrónico (editable)
- Número de teléfono (editable)

#### 2. Sección Inferior

- Botón “Editar información” con un icono identificador
- Al pulsarlo, el sistema carga la página de edición del perfil del empleado con permisos de edición sobre los campos

#### 3. Sección de Edición (visible solo tras pulsar “Editar información”)

- Nombre de usuario (editable)
- Nombre completo (editable)
- Correo electrónico (editable)
- Número de teléfono (editable)
- Campo oculto: contraseña (no editable)
- Lista desplegable para seleccionar el rol del empleado (no editable)
- Botón “Confirmar cambios” para guardar las modificaciones

#### 4. Sección de Confirmación (visible solo tras pulsar “Confirmar cambios”)

- Mensaje de éxito o error según el resultado de la validación del sistema

### **Explicación del flujo**

- El responsable localiza al empleado y pulsa el botón o icono “Editar empleado”.
- Se carga la interfaz con los datos del empleado y se habilitan los campos editables.
- El responsable edita la información deseada.
- Al hacer clic en “Confirmar cambios”, el sistema valida los datos.
- Si la validación es exitosa, se actualiza la información y se muestra un mensaje de confirmación.

## **Salida del modelo**

### **Formato de pantalla**

**Título:** Actividades turísticas disponibles

### **Componentes principales**

- Botones para filtrar por fecha, nombre o categoría
- Panel de actividades con:
  - Imagen
  - Nombre
  - Categoría
  - Botón “Ver detalles”
- Detalle de la actividad:
  - Imagen
  - Nombre
  - Categoría
  - Duración

- Descripción
- Comentarios
- Fechas disponibles
- Precio por persona
- Botón “Reservar actividad”
- Formulario de reserva:
  - Nombre y apellidos
  - Correo electrónico
  - Número de teléfono
  - Información de la actividad
- Botón “Realizar pago”

#### **Disposición de los componentes**

- Header con logotipo y navegación (opcional)
- Filtros por fecha, nombre o categoría (botones)
- Panel de actividades
- Botón “Ver detalles” en el panel
- Detalle de la actividad
- Botón “Reservar actividad”
- Formulario de reserva
- Botón “Realizar pago”
- Inicio de sesión y registro (opcional)
- Footer con información adicional (opcional)

### **Explicación**

El formato generado contiene los componentes principales que permiten a un usuario realizar una reserva de actividades turísticas. La disposición de los elementos facilita la navegación y la interacción del usuario con el sistema.

Los pasos detallados describen la secuencia de eventos que ocurren durante el caso de uso de reserva de actividad turística. Este diseño puede ser ajustado según las necesidades específicas, siempre que se mantenga alineado con los objetivos del caso de uso y la experiencia de usuario.

### **Wireframe textual**

#### **Header {**

Logotipo

Navegación (opcional)

**}**

Filtrar por fecha, nombre o categoría (botones)

Panel de actividades

- Botón “Ver detalles”

Detalle de la actividad

Imagen

Nombre

Categoría

Duración

Descripción

Comentarios

Fechas disponibles

Precio por persona

Botón “Reservar actividad”

Formulario de reserva:

- Campo de nombre y apellidos
- Campo de correo electrónico
- Campo de número de teléfono
- Información de la actividad (repetida)

Botón “Realizar pago”

Footer (opcional)

### Salida del modelo

#### Formato de pantalla: Reserva de Actividad Turística - Pasarela de Pago

- **1. Título:** Muestra el nombre de la actividad turística seleccionada, su fecha y lugar, acompañado de un subtítulo indicando que se trata de la pasarela de pago.
- **2. Método de pago:** Botón para que el usuario elija entre dos métodos: PayPal o Transferencia Bancaria.
- **3. Datos de la operación:** Información sobre la transacción, incluyendo:
  - Importe
  - Comercio
  - Fecha de la operación
  - Descripción de la compra
- **4. Formulario de datos para la factura:** Campos que el usuario debe completar:
  - Nombre completo
  - CIF
  - Dirección
  - Código postal

- Población
  - País
  - Provincia
- **5. Botones “Cancelar” y “Realizar pago”:** Permiten cancelar la operación o continuar con el pago.
  - **6. Mensaje de confirmación de pago:** Se muestra tras completar el pago con éxito, incluyendo confirmación y factura.

### Descripción

Esta pantalla permite a un usuario realizar una reserva de actividad turística, introduciendo sus datos personales y eligiendo su método de pago dentro de la pasarela. Una vez confirmada la operación, se muestra un mensaje de confirmación junto con la factura correspondiente.

```

**Miframe textual:**
-----|
| Titulo: Nombre de la actividad |
| Subtitulo: Pasarela de pago |
|-----|
| Método de pago | Botón: Cancelar|
|-----|
| Importe, comercio, fecha | Botón: Realizar pago |
| Descripción |
|-----|
| Nombre completo | Campo de texto |
| CIF | Campo de texto |
| Dirección | Campo de texto (se muestra una línea de texto para cada campo) |
| Código postal | Campo de texto |
| Población | Campo de texto |
| País | Campo de texto |
| Provincia | Campo de texto |
|-----|
| Mensaje de confirmación |
|-----|

```

Figura 63: Realizar pago de reserva de actividad turística. Fuente: Mistral

## A.2. Implementación software

### A.2.1. Llama 3.1

#### Diseño e implementación base de datos

Comenzando con el diseño e implementación de la base de datos, se ha tenido en cuenta el siguiente prompt:

### Prompt

Tengo un conjunto de requisitos de información y un enunciado que describen sistema. Quiero que ayudes a diseñar una base de datos relacional. Genera el diagrama entidad-relación y el código SQL correspondiente para implementarlo. El diseño debe incluir claves primarias, claves foráneas, tipos de datos adecuados y relaciones entre entidades. Primero te voy a proporcionar una descripción resumida del sistema:

El sistema es una aplicación web de reservas de actividades turísticas. Existen usuarios que se diferencian por su rol, entre los que se distinguen responsable de la aplicación, empleados, monitores y clientes. Los usuarios como responsable y empleados se encargan de gestionar la aplicación web, pueden crear actividades turísticas. Los usuarios clientes pueden reservar las actividades. Un usuario puede tener múltiples reservas de actividades. A cada reserva se asocia una única factura. Los datos de las reservas de cada usuario se almacenan en el sistema. Por último, existe los usuarios tienen la posibilidad para añadir comentarios en actividades.

Los requisitos de información del sistema que debes de tener en cuenta son: sección [B.1.1](#)

Enunciado completo: sección [3.1.2](#)

Debido a problemas con el entorno LaTeX se ha optado por mostrar todo lo relacionado al código SQL de manera agrupada en la salida, mostrando por cada entidad los atributos, los cuales han sido extraídos del código SQL (CREATE).

### Salida del modelo

#### Usuario

- ID\_USER | INT | NOT NULL (PK)
- nombre | VARCHAR(50) | NOT NULL
- apellidos | VARCHAR(100) | NULL
- DNI\_NIE | VARCHAR(20) | NOT NULL
- correo\_electronico | VARCHAR(200) | NOT NULL
- telefono\_móvil | VARCHAR(20) | NULL

- contraseña\_hashed | VARCHAR(255) | NOT NULL
- rol | ENUM('cliente', 'monitor', 'empleado', 'responsable') | NULL

**Relaciones:**

- 1:N con *reserva* (usuario\_que\_realizo\_la\_reserva)
- 1:N con *comentario* (usuario\_que\_hizo\_el\_comentario)

**Monitor**

- ID\_MONITOR | INT | NOT NULL (PK)
- estudios | VARCHAR(100) | NOT NULL

**Relaciones:**

- No posee relaciones con el modelo proporcionado

**Actividad**

- ID\_ACTIVIDAD | INT | NOT NULL (PK)
- hora | TIME | NOT NULL
- descripcion | TEXT | NULL
- tarifa | DECIMAL(10,2) | NOT NULL
- imagenes | TEXT | NULL
- duracion | INT | NOT NULL
- plazas | INT | NULL
- fecha | DATE | NOT NULL
- categoria | ENUM('categorías definidas') | NULL

**Relaciones:**

- 1:N con *reserva* (actividad\_reservada)
- 1:N con *comentario* (actividad\_relacionada)

### **Reserva**

- ID\_RESERVA | INT | NOT NULL (PK)
- usuario\_que\_realizo\_la\_reserva | INT | NOT NULL (FK *usuario*)
- actividad\_reservada | INT | NOT NULL (FK *actividad*)
- fecha\_de\_reserva | DATE | NULL

### **Relaciones:**

- N:1 con *usuario*
- N:1 con *actividad*
- 1:1 con *pago* (reserva\_asociada)

### **Pago**

- ID\_PAGO | INT | NOT NULL (PK)
- reserva\_asociada | INT | NOT NULL (FK *reserva*)
- metodo\_de\_pago | TEXT | NOT NULL
- cantidad | DECIMAL(10,2) | NULL

### **Relaciones:**

- N:1 con *reserva*
- 1:1 con *factura* (pago\_asociado)

### **Factura**

- ID\_FACTURA | INT | NOT NULL (PK)

- pago\_asociado | INT | NOT NULL (FK *pago*)

**Relaciones:**

- N:1 con *pago*

**Comentario**

- ID\_COMENTARIO | INT | NOT NULL (PK)
- usuario\_que\_hizo\_el\_comentario | INT | NOT NULL (FK *usuario*)
- actividad\_relacionada | INT | NOT NULL (FK *actividad*)

**Relaciones:**

- N:1 con *usuario*
- N:1 con *actividad*

### A.2.2. Mistral

#### Diseño e implementación base de datos

Para el diseño e implementación de la base de datos, se ha tenido en cuenta el mismo *prompt* que se ha utilizado en el caso de Llama 3.1. La salida obtenida es:

#### Salida del modelo

##### Usuario

- ID\_usuario | INT | NOT NULL (PRIMARY KEY)
- Nombre | VARCHAR(255) | NOT NULL
- Apellidos | VARCHAR(255) | NULL
- DNI\_NIE | VARCHAR(9) | NOT NULL
- Correo\_electronico | VARCHAR(255) | NULL
- Telefono\_movil | VARCHAR(13) | NOT NULL

- Contraseña\_hash | VARCHAR(255) | NOT NULL
- Rol | ENUM ('Responsable', 'Empleado', 'Monitor', 'Cliente') | NOT NULL

**Relaciones:**

- Se relaciona con **monitor** (1:1) como clave foránea en monitor.Usuario\_ID\_usuario
- Se relaciona con **reserva** como 1:N

**Categoría**

- ID\_categoria | INT | NOT NULL (PRIMARY KEY)
- Nombre\_categoria | VARCHAR(255) | NULL

**Relaciones:**

- Se relaciona con **actividad** (1:N) como clave foránea en actividad.Categoría\_ID\_categoria

**Actividad**

- ID\_actividad | INT | NOT NULL (PRIMARY KEY)
- Hora | DATETIME | NOT NULL
- Descripción | TEXT | NULL
- Tarifa | DECIMAL(10,2) | NOT NULL
- Imagenes | BLOB | NULL
- Duración | INT | NOT NULL
- Día\_semana | ENUM('Lunes', ..., 'Domingo') | NOT NULL
- Categoría\_ID\_categoria | INT | NOT NULL (FOREIGN KEY)

**Relaciones:**

- Pertenece a una **categoría**
- Se relaciona con **reserva** (1:N)

#### **Monitor**

- ID\_monitor | INT | NOT NULL (PRIMARY KEY)
- Usuario\_ID\_usuario | INT | NOT NULL (FOREIGN KEY)
- Estudios | TEXT | NULL

#### **Relaciones:**

- Está vinculado a un **usuario** con rol = Monitor (1:1)

#### **Reserva**

- ID\_reserva | INT | NOT NULL (PRIMARY KEY)
- Cliente\_ID\_cliente | INT | NOT NULL (FOREIGN KEY)
- Actividad\_ID\_actividad | INT | NOT NULL (FOREIGN KEY)
- Fecha | DATETIME | NULL

#### **Relaciones:**

- Relaciona un **usuario (cliente)** con una **actividad**
- Se relaciona con **pago** (1:1 o 1:N)

#### **Pago**

- ID\_pago | INT | NOT NULL (PRIMARY KEY)
- Reserva\_ID\_reserva | INT | NOT NULL (FOREIGN KEY)
- Método | ENUM('Tarjeta', 'Transferencia') | NULL
- Cantidad | DECIMAL(10,2) | NOT NULL

- Fecha | DATETIME | NOT NULL
- Factura\_ID\_factura | INT | NULL (FOREIGN KEY)

**Relaciones:**

- Relacionado a una **reserva**
- Puede generar una **factura**

**Factura**

- ID\_factura | INT | NOT NULL (PRIMARY KEY)
- Detalle | TEXT | NULL
- etc.

**Relaciones:**

- Puede estar vinculada a uno o varios **pagos**

### A.3. Pruebas software

Para el caso de las pruebas software, debido a que no se va a mostrar código en esta parte de los apéndices si no que adjunto en la entrega en archivos específicos, se va a utilizar este apartado para indicar los *prompts* que se han utilizado.

Se han orientado las entradas de los LLMs a las funcionalidades, descritas por casos positivos y negativos.

#### Registro de nuevo usuario

**Prompt**

Genera tests E2E con PactumJS + Jest contra mi API real. Salida: solo código TypeScript, sin explicaciones.

Prohibido usar: `.send()`, `supertest` (`request(...)`, `.expect(...)`), `expect(respuesta.status).toBe(...)`, `axios`, `fetch`, `.inspect()`. Obligatorio usar (PactumJS): `.withHeaders(...)`, `.withJson(...)` (o `.withBody(...)`), `.expectStatus(...)`, y aserciones de

Pactum (expectHeader, expectJsonLike/expectJsonMatch).

Entorno:

Base URL: `BASE_URL.EnbeforeAll, llamaapactum.request.setBaseUrl('BASE_URL')`.

Sin mocks ni interceptores de Pactum.

Añade `withHeaders('Content-Type', 'application/json')` en cada petición que lo requiera.

Endpoint a probar: Registro

POST /auth/registro

Request DTO (JSON): `'nombre': 'NOMBRE', 'apellidos': 'APELLIDOS', 'telefono': 'TELEFONO', 'dni': 'DNI', 'email': 'EMAIL', 'username': 'USERNAME', 'password': 'PASSWORD'`

Reglas & estados esperados:

201 alta correcta

409 si email o dni ya existen.

400 si falta algún campo o falla una validación (p.ej., dni demasiado corto).

Estructura de pruebas:

Un `describe('auth/registro')` y un `it()` por funcionalidad:

Éxito 201 con datos únicos

Duplicado 409 repitiendo exactamente el mismo body del caso de éxito.

400 por campo faltante

400 por validación (ej.: dni con longitud inválida).

Aserciones por caso:

`expectStatus(...)`

`withHeaders('Content-Type', 'application/json')`

Utiliza las aserciones que creas necesarias.

## Login de usuario

### Prompt

Genera tests E2E con PactumJS + Jest contra mi API real. Salida: solo código TypeScript, sin explicaciones.

Prohibido usar: `.send()`, `supertest` (`request(...)`, `.expect(...)`), `ex-`

pect(respuesta.status).toBe(...), axios, fetch, .inspect(). Obligatorio usar (PactumJS):  
.withHeaders(...), .withJson(...) (o .withBody(...)), .expectStatus(...), y aserciones de  
Pactum (expectHeader, expectJsonLike/expectJsonMatch).

Entorno:

Base URL: `BASE_URL.EnbeforeAll, llamaapactum.request.setBaseUrl('BASE_URL')`.

Sin mocks ni interceptores de Pactum.

Añade `withHeaders('Content-Type', 'application/json')` en cada petición que lo requiera.

Endpoint a probar: Login de usuarios

POST /auth/login

Request DTO (JSON): `'username': 'USERNAME', 'password': 'PASSWORD'`

Reglas y estados esperados:

200 login correcto

404 si nombre de usuario no existente en el sistema

400 si falta algún campo o falla una validación (e.g., no se proporciona password).

Estructura de pruebas:

Un `describe('auth/login')` y un `it()` por funcionalidad:

Éxito 200 con datos únicos

404 si usuario no existente

400 por campo faltante (e.g. no se proporciona password)

Aserciones por caso:

`expectStatus(...)`

`withHeaders('Content-Type', 'application/json')`

Utiliza las aserciones que creas necesarias

Segundo *prompt* utilizado:

#### Prompt

Necesito que en las pruebas de auth/login/ en el caso de éxito, se almacene el `access_token` devuelto por el servidor en una variable `token`.

#### Gestión de empleados. Lista de empleados

### Prompt

Genera tests E2E con PactumJS + Jest contra mi API real. Salida: solo código TypeScript, sin explicaciones.

Prohibido usar: `.send()`, `supertest (request(...), .expect(...))`, `expect(respuesta.status).toBe(...)`, `axios`, `fetch`, `.inspect()`. Obligatorio usar (PactumJS): `.withHeaders(...)`, `.withJson(...)` (o `.withBody(...)`), `.expectStatus(...)`, y aserciones de Pactum (`expectHeader`, `expectJsonLike/expectJsonMatch`).

Entorno:

Base URL: `BASE_URL.EnbeforeAll, llamaapactum.request.setBaseUrl('BASE_URL')`.

Sin mocks ni interceptores de Pactum.

Añade `withHeaders('Content-Type','application/json')` en cada petición que lo requiera.

Endpoint a probar: Gestión de empleados. Lista de empleados.

GET /empleados

La lista de empleados solo es visible para el responsable de la aplicación. Es necesario el token almacenado previamente.

Reglas y estados esperados:

200 lista de empleados obtenida correctamente si se tienen permisos

403 si un usuario registrado y habiendo hecho login realiza la llamada a este recurso, pero no tiene permisos ya que su rol es distinto al de responsable

Estructura de pruebas:

Un `describe('lista de empleados')` y un `it()` por funcionalidad:

Éxito 200 con lista de empleados

403 si se está logueado, pero no se tienen permisos sobre el recurso

Aserciones por caso:

`expectStatus(...)`

`withHeaders('Content-Type', 'application/json')`

Utiliza las aserciones que creas necesarias

### Gestión de empleados. Obtener empleado

### Prompt

Genera tests E2E con PactumJS + Jest contra mi API real. Salida: solo código TypeScript, sin explicaciones.

Prohibido usar: `.send()`, `supertest (request(...), .expect(...))`, `expect(respuesta.status).toBe(...)`, `axios`, `fetch`, `.inspect()`.

Obligatorio usar (PactumJS): `.withHeaders(...)`, `.withJson(...)` (o `.withBody(...)`), `expectStatus(...)`, y aserciones de Pactum (`expectHeader`, `expectJsonLike/expectJsonMatch`).

Entorno: Base URL: `{{BASE_URL}}`. En `beforeAll`, llama a `pactum.request.setBaseUrl('{{BASE_URL}}')`.

Sin mocks ni interceptores de Pactum. Añade `withHeaders('Content-Type','application/json')` en cada petición que lo requiera.

Endpoint a probar: Gestión de empleados. Lista de empleados. GET `/empleados/:id`

Es necesario el token de responsable almacenado previamente.

Reglas estados esperados: 200 empleado existente obtenido correctamente si se tienen permisos;

403 si un usuario registrado y habiendo hecho login realiza la llamada a este recurso, pero no tiene permisos ya que su rol es distinto al de responsable;

404 si se tienen privilegios necesarios (token de responsable), pero no existe el empleado solicitado en el sistema.

Estructura de pruebas: Un `describe('lista de empleados')` y un `it()` por funcionalidad: Éxito 200 con empleado seleccionado;

403 si se está logueado, pero no se tienen permisos sobre el recurso; 404 si no existe el empleado.

Aserciones por caso: `expectStatus(...)`, `withHeaders('Content-Type', 'application/json')`.

Utiliza las aserciones que creas necesarias.

### Gestión de empleados. Actualizar empleado

### Prompt

Genera tests E2E con PactumJS + Jest contra mi API real. Salida: solo código TypeScript, sin explicaciones.

Prohibido usar: `.send()`, `supertest (request(...), .expect(...))`, `expect(respuesta.status).toBe(...)`, `axios`, `fetch`, `.inspect()`. Obligatorio usar (PactumJS): `.withHeaders(...)`, `.withJson(...)` (o `.withBody(...)`), `.expectStatus(...)`, y aserciones de Pactum (`expectHeader`, `expectJsonLike/expectJsonMatch`).

Entorno:

Base URL: `BASE_URL.EnbeforeAll, llamaapactum.request.setBaseUrl('BASE_URL')`.

Sin mocks ni interceptores de Pactum.

Añade `withHeaders('Content-Type','application/json')` en cada petición que lo requiera.

Endpoint a probar: Gestión de empleados. Actualizar empleado.

PATCH `/empleados/:id`

Request body: `'campo': 'CAMPO'`

Es necesario el token de responsable almacenado previamente.

Reglas estados esperados:

200 campo de empleado actualizado correctamente si se tienen permisos

403 si un usuario registrado y habiendo hecho login realiza la llamada a este recurso, pero no tiene permisos ya que su rol es distinto al de responsable

404 si se tienen privilegios necesarios (token de responsable), pero no existe el empleado solicitado en el sistema

Estructura de pruebas:

Un `describe('actualizar empleados')` y un `it()` por funcionalidad:

Éxito 200 con empleado actualizado

403 si se está logueado, pero no se tienen permisos sobre el recurso

404 si no existe el empleado

Aserciones por caso:

`expectStatus(...)`

`withHeaders('Content-Type', 'application/json')`

Utiliza las aserciones que creas necesarias

## Gestión de empleados. Crear empleado

**Prompt**

Genera tests E2E con PactumJS + Jest contra mi API real. Salida: solo código TypeScript, sin explicaciones.

Prohibido usar: `.send()`, `supertest (request(...), .expect(...))`, `expect(respuesta.status).toBe(...)`, `axios`, `fetch`, `.inspect()`. Obligatorio usar (PactumJS): `.withHeaders(...)`, `.withJson(...)` (o `.withBody(...)`), `.expectStatus(...)`, y aserciones de Pactum (`expectHeader`, `expectJsonLike/expectJsonMatch`).

Entorno:

Base URL: `BASE_URL.EnbeforeAll, llamaapactum.request.setBaseUrl('BASE_URL')`.

Sin mocks ni interceptores de Pactum.

Añade `withHeaders('Content-Type','application/json')` en cada petición que lo requiera.

Endpoint a probar: Gestión de empleados. Crear empleado.

POST `/empleados/nuevo`

Request DTO (JSON) `'nombre': 'NOMBRE', 'apellidos': 'APELLIDOS', 'telefono': 'TELEFONO', 'dni': 'DNI', 'email': 'EMAIL', 'username': 'USERNAME', 'password': 'PASSWORD'`

Es necesario el token de responsable almacenado previamente.

Reglas estados esperados:

201 empleado creado con éxito si se tienen permisos (token de responsable)

403 si un usuario registrado y habiendo hecho login realiza la llamada a este recurso, pero no tiene permisos ya que su rol es distinto al de responsable

409 si se tienen privilegios necesarios (token de responsable), pero ya existe el email o DNI utilizados en el proceso de creación

Estructura de pruebas:

Un `describe('crear empleado')` y un `it()` por funcionalidad:

Éxito 200 con empleado creado

403 si se está logueado, pero no se tienen permisos sobre el recurso

409 si datos únicos ya existen en el sistema

Aserciones por caso:

`expectStatus(...)`

```
withHeaders('Content-Type', 'application/json')
```

Utiliza las aserciones que creas necesarias

### Gestión de empleados. Eliminar empleado

#### Prompt

Genera tests E2E con PactumJS + Jest contra mi API real. Salida: solo código TypeScript, sin explicaciones.

Prohibido usar: `.send()`, `supertest` (`request(...)`, `.expect(...)`), `expect(respuesta.status).toBe(...)`, `axios`, `fetch`, `.inspect()`. Obligatorio usar (PactumJS): `.withHeaders(...)`, `.withJson(...)` (o `.withBody(...)`), `.expectStatus(...)`, y aserciones de Pactum (`expectHeader`, `expectJsonLike/expectJsonMatch`).

Entorno:

Base URL: `BASE_URL.EnbeforeAll, llamaapactum.request.setBaseUrl('BASE_URL')`.

Sin mocks ni interceptores de Pactum.

Añade `withHeaders('Content-Type','application/json')` en cada petición que lo requiera.

Endpoint a probar: Gestión de empleados. Eliminar empleado.

`DELETE /empleados/:id`

Es necesario el token de responsable almacenado previamente.

Reglas estados esperados:

200 empleado eliminado con éxito si se tienen permisos (token de responsable)

403 si un usuario registrado y habiendo hecho login realiza la llamada a este recurso, pero no tiene permisos ya que su rol es distinto al de responsable

404 si el empleado indicado no existe

Estructura de pruebas:

Un `describe('eliminar empleado')` y un `it()` por funcionalidad:

Éxito 200 con empleado eliminado

403 si se está logueado, pero no se tienen permisos sobre el recurso

404 si datos no existen en el sistema

Aserciones por caso:

```
expectStatus(...)
```

```
withHeaders('Content-Type', 'application/json')
```

Utiliza las aserciones que creas necesarias

## Reservar y pagar mediante PayPal

### Prompt

Genera tests E2E con PactumJS + Jest contra mi API real. Salida: solo código TypeScript, sin explicaciones.

Prohibido usar: `.send()`, `supertest (request(...), .expect(...))`, `expect(respuesta.status).toBe(...)`, `axios`, `fetch`, `.inspect()`. Obligatorio usar (PactumJS): `.withHeaders(...)`, `.withJson(...)` (o `.withBody(...)`), `.expectStatus(...)`, y aserciones de Pactum (`expectHeader`, `expectJsonLike/expectJsonMatch`).

Entorno:

Base URL: `BASE_URL.EnbeforeAll, llamaapactum.request.setBaseUrl('BASE_URL')`.

Sin mocks ni interceptores de Pactum.

Añade `withHeaders('Content-Type','application/json')` en cada petición que lo requiera.

Endpoint a probar: Realizar resera y pagar.

Crea un `describe('realizar reserva y pago de dicha reserva')` con dos `it()`

Crear reserva y crear orden de pago → 201

Endpoint: `POST /reservas/checkout`

Body JSON de ejemplo: `'nombre': 'Javier', 'apellidos': 'Ortega Molina', 'email': 'qa+reserva+Date.now()@turismollm.com', 'telefono': '600000002', 'id_actividad' : 5, 'id_actividad_dia' : 1, 'fecha' : '2025 - 08 - 25'`

Aserciones: `expectStatus(201)`

Guarda `orderId` de la respuesta

Capturar orden de pago, generar factura y completar reserva → 503

Endpoint: `POST /pagos/orders/capture`

Body JSON: `'orderId': 'SorderId', 'factura': 'nombre_apellidos' : 'JavierOrtegaMolina', 'nif' : '12345678Y', 'direccion' : 'C/Mayor11', 'codigo_postal' : 28001, 'poblacion' : 'Madrid', 'provincia' : 'Madrid', 'pais' : 'ES'`

Aserciones: `expectStatus(503)`

Notas

Usa `withJson(...)` y `expectStatus(...)` en todas las peticiones

No añadas comentarios ni texto fuera del código



# Apéndice B

## Desarrollo manual

En este apéndice se recoge todo el desarrollo manual que es necesario mostrar. Por ello, para evitar saturar ciertos capítulos del cuerpo principal, se ha decidido utilizar este apéndice para ello. Se divide en las distintas fases seleccionadas para el proyecto, aunque no aparecen todas.

### B.1. Análisis y Diseño

Esta sección contiene de manera detallada todo el desarrollo manual realizado, necesario para comprender ciertas partes del desarrollo del caso práctico.

#### B.1.1. Catálogo de requisitos

Para los requisitos se han obtenido los siguientes catálogos

##### Catálogo de requisitos funcionales

Identificador	Título	Descripción
RF-01	Gestión de Empleados	El responsable de la aplicación web debe ser capaz de gestionar los empleados de la empresa. En la gestión se incluyen operaciones de creación de empleados, lectura de su información, actualizar información de los empleados y eliminar a los mismos.
RF-02	Gestión de Monitores	Los empleados de la empresa turística deben ser capaces de administrar a los monitores de las actividades. Esta gestión incluye operaciones de creación, lectura, actualización y eliminación de los monitores.
RF-03	Gestión de reservas online	Permite a los empleados leer las reservas que se hayan realizado.

RF-04	Reservar por parte de empleados hacia clientes	Los empleados de la empresa deben ser capaces de realizar reservas para clientes en el momento.
RF-05	Gestión de Actividades Turísticas	Los empleados de la empresa deben de ser capaces de crear nuevas actividades, actualizar la información de las existentes, acceder a su información y eliminar actividades.
RF-06	Gestión de Actividades de Monitores	Los empleados de la empresa son encargados de gestionar las actividades que cada monitor realiza por día. En esta gestión se tendrá en cuenta que un monitor no puede realizar más de una actividad distinta por día.
RF-07	Asignar monitor por actividad	Los empleados deben ser capaces de asignar actividades a cada monitor, teniendo en cuenta si está cualificado en función de la categoría de la actividad.
RF-08	Ver listado de monitores	Los empleados de la empresa tendrán a su disposición una tabla que contenga a todos los monitores de la empresa.
RF-09	Análisis de datos	El responsable de la aplicación Web y los empleados podrán acceder a un área de análisis de datos en el que podrán interactuar con los datos mediante la lectura de gráficas y tablas generadas, entre otros.
RF-10	Gestión de pagos online	Los usuarios, registrados o no, podrán realizar pagos de las reservas de actividades turísticas que quieran participar. La plataforma proporcionará dos métodos de pago los cuales solo se podrá elegir uno. Los pagos son NO reembolsables.
RF-11	Pago mediante Paypal	Los usuarios que escojan la opción de pago con Paypal serán redirigidos a un área de pago mediante el servicio Paypal.

RF-12	Pago mediante transferencia bancaria	Los usuarios que escojan la opción de pago mediante transferencia bancaria serán redirigidos a una plataforma de pago por parte de la entidad bancaria.
RF-13	Generar factura de pago	Los usuarios que realicen un pago mediante el método preferido podrán opcionalmente pedir que se proporcione una factura. Este requisito se encarga de generar la factura.
RF-14	Gestión de comentarios de actividades	Los empleados de la empresa podrán gestionar los comentarios que haya por actividad. Esta gestión de comentarios incluye la lectura y eliminación de los mismos.
RF-15	Actualizar precios de actividades	Los empleados de la empresa podrán actualizar los precios de las actividades turísticas.
RF-16	Ver listado de clientes por actividad turística	Los monitores deben de ser poder acceder a un listado de clientes por actividad turística que tienen asignada.
RF-17	Registro de nuevo usuario	Un usuario anónimo debe poder introducir su correo y contraseña para que se pueda dar de alta en el sistema. Cuando un usuario anónimo se registra, pasa a ser considerado como cliente en el sistema.
RF-18	Iniciar sesión	Los usuarios registrados en el sistema deben ser capaces de acceder al mismo sistema mediante las credenciales que introdujeron en el registro de su cuenta.
RF-19	Cerrar sesión	Los usuarios registrados y con la sesión iniciada deben ser capaces de cerrar su sesión para desconectarse como usuario registrado en el sistema.
RF-20	Buscar actividades	Los usuarios, registrados o no, deben ser capaces de buscar actividades por categoría o nombre.

RF-21	Realizar reservas	Los usuarios, registrados o no, podrán acceder a la aplicación web para realizar reservas de las actividades turísticas disponibles en cada día.
RF-22	Crear comentarios	Los clientes, registrados en el sistema, deben ser capaces de dejar comentarios en las actividades turísticas.
RF-23	Responder comentarios	Los clientes, registrados, incluyendo a los empleados, deben ser capaces de responder a comentarios existentes en las actividades turísticas.
RF-24	Ver listado de reservas	Los usuarios registrados deben ser capaces de acceder a un listado de las reservas de actividades turísticas que realizaron. En este listado se incluye un histórico de las que se reservaron en el pasado y ya pasaron como las que todavía no se han realizado, pero se reservaron para un futuro.
RF-25	Actualizar duración de las actividades	Aunque las actividades tengan una duración fija, en situaciones excepcionales se necesitará actualizar la duración de la actividad.

Cuadro 117: Catálogo de requisitos funcionales. Fuente: elaboración propia.

**Catálogo de requisitos no funcionales**

<b>Identificador</b>	<b>Título</b>	<b>Descripción</b>
RNF-01	Base de Datos del sistema	El sistema hará uso de una base de datos relacional de MariaDB con propiedades ACID para almacenar las cuentas de usuarios, monitores, empleados, responsables, reservas, comentarios, información de pagos, facturas y actividades turísticas. Tendrá una capacidad mínima para 5000 clientes, 100 actividades y 10,000 reservas. Asimismo, el servicio de la base de datos estará activo todo el tiempo. También almacenará la información que se considere necesario para, por ejemplo, uso de datos en el análisis de datos.
RNF-02	Web Responsive	La interfaz de la plataforma web deberá ser capaz de adaptarse automáticamente al tamaño de la pantalla del dispositivo desde el que se está accediendo y será compatible con las últimas versiones de navegadores como Chrome, Firefox, Safari y Edge.
RNF-03	Rendimiento de la aplicación web	La aplicación Web debe ser eficiente y disponer de un buen rendimiento, es decir, debe tener una carga rápida de los recursos, responder con un tiempo de respuesta adecuado a las acciones de los usuarios y no congelarse. Por ello, bajo una carga de 500 usuarios concurrentes, las peticiones a la API deberán de ser ejecutadas en un tiempo menor a 0,5 segundos.

RNF-04	Seguridad de la aplicación web	La aplicación Web debe ser segura, debe de usar protocolos de red seguros como HTTPS y uso de algoritmos de cifrado para transacciones seguras. La plataforma Web debe ser capaz de protegerse ante accesos no autorizados. Por ello, el tráfico en la web será sobre HTTPS y TLS versión 1.2 y superiores. Por otro lado, se utilizarán algoritmos como Argon2 [38] para implementar técnicas de hashing que aseguren los datos sensibles del sistema y, además, prevean ataques como <i>Rainbow Table</i> [39]. Finalmente, las sesiones expirarán tras 25 minutos.
RNF-05	Seguridad en el pago	Los pagos se van a realizar de forma online por lo que se hará uso de plataformas certificadas como Paypal que se encargarán de procesar la seguridad en el pago. En el caso de pago mediante transferencia bancaria también se hará uso de las herramientas de seguridad que proporcionan las mismas entidades bancarias. Asimismo, las pasarelas de pago estarán cifradas extremo a extremo para que no puedan ser interceptadas y sea susceptible a robos y ataques. Esta seguridad dependerá principalmente de las plataformas de pago implementadas.

RNF-06	Usabilidad	La aplicación web debe permitir que los usuarios comprendan y utilicen sus funcionalidades principales como registro de usuario, inicio de sesión, búsqueda de actividades turística, reservas de actividades y pagos de reservas, sin necesidad de capacitación previa ni asistencia externa. Tras un primer contacto con la interfaz, los usuarios deben ser capaces de completar cada una de estas tareas en un tiempo no mayor a 5 minutos, lo que supone una curva de aprendizaje baja y una interacción eficiente con el sistema.
RNF-07	Informar de errores	La aplicación web debe ser capaz de informar de errores que ocurran en la procesos de la plataforma o en general y que disponga de mensajes que los usuarios finales sean capaces de entender sin requerir un contexto técnico. Los mensajes deben de ser comprensibles, con códigos de referencia típicos (códigos 400 y 500) y acciones sugeridas.
RNF-08	Roles de usuario	Para hacer una gestión adecuada de los recursos y controlar lo que visualiza cada usuario del sistema o no, se implementará RBAC [ <b>cloudflare_bac</b> ] con roles de usuario como Responsable, Empleado, Monitor, Cliente y Usuario Anónimo. Cada acción del sistema requerirá permiso explícito del rol. Las pruebas de autorización cubrirán al menos el 100 % de rutas críticas, como la administración de la plataforma.

RNF-09	Control de acceso a los recursos	Se establecerán políticas de acceso y permisos a recursos mediante los roles de usuario previamente implementados. De esta forma, se realiza un control de acceso a los recursos que rechaza las peticiones y acciones de usuarios no identificados o con permisos insuficientes. Se seguirá la política de mínimo privilegio y se implementarán tokens de acceso como JSON Web Token que servirán para permitir acceso a ciertas rutas.
--------	----------------------------------	--

Cuadro 118: Catálogo de requisitos no funcionales. Fuente: elaboración propia.

### Catálogo de requisitos de información

Identificador	Título	Descripción
RI-01	Cuentas de usuario	Se almacenarán cuatro tipos de cuentas de usuario, es decir, clientes, monitores, empleados y responsable. De estas entidades se almacenará su nombre, apellidos, nombre de usuario, DNI/NIE, correo electrónico, teléfono móvil, contraseña <i>hashed</i> . Los usuarios serán distinguidos por roles.
RI-02	Actividades	Se necesitará almacenar la siguiente información de las actividades: Identificador de actividad, hora, descripción, tarifa, imágenes, duración, plazas y fecha. Además, se diferenciará por categoría.
RI-03	Monitores	Un monitor es un tipo de cuenta de usuario del cual se almacena un atributo extra, los estudios. Los estudios serán necesario para administrar la cualificación por categoría de actividades que pueda realizar.
RI-04	Reservas	Se almacenará información de las reservas que se han realizado en la empresa. Esta información es necesaria para análisis de datos y en el caso de usuarios registrados, disponer de un listado de reservas de actividades. De las reservas se almacenan los datos del cliente que realiza dicha reserva.
RI-05	Pagos	Se registrarán todos los pagos asociados a reservas. Estos pagos incluyen fecha en la que se ha realizado dicho pago, además se almacena el método y cantidad.
RI-06	Facturas	Por cada reserva de actividad se genera una factura. Dicha factura presenta el detalle de la actividad pagada por un usuario, registrado o no, junto a todos los datos necesarios para realizar una factura.

Cuadro 119: Catálogo de requisitos de información. Fuente: elaboración propia.

### B.1.2. Casos de uso

En esta sección se recoge la especificación de los casos de uso desarrollados manualmente.

#### Especificación de casos de uso

A continuación, se muestran la especificación de casos de uso de los procesos de negocios más relevantes, entre los que se encuentran la gestión de empleados que incluye operaciones de creación de nuevo empleado, eliminación de empleado, lectura de información de empleado y edición de la información de empleado. Se ha mostrado esta especificación de caso de uso puesto que el resto de procesos de gestión, usuarios, reservas, actividades, entre otros, sigue la misma lógica. Asimismo, entre los procesos de negocios relevantes a este caso práctico se han diferenciado registrar usuario e iniciar sesión, gestión de empleados, reserva de actividad turística y pago de reserva de actividad.

Por ello, la especificación de los casos de uso más relevantes son:

<b>Título</b>	Gestionar empleados
<b>Descripción</b>	El responsable de la plataforma web es el único que puede realizar acciones de creación, eliminación, actualización y lectura de la información de los empleados de la empresa.
<b>Pre-condición</b>	Usuario registrado y disponer de rol de responsable para acceder al panel de administración
<b>Post-condición</b>	El responsable habrá sido capaz de crear o eliminar perfiles de sus empleados, así como de actualizar o consultar su información.
<b>Prioridad</b>	Alta
<b>Escenario principal</b>	

### **Escenario de creación de empleado**

1. El responsable hace clic en 'Añadir empleado'.
2. El sistema genera un formulario para introducir los datos del nuevo empleado.
3. El responsable introduce los datos del nuevo empleado.
4. El sistema verifica y valida los datos del nuevo empleado que se quiere agregar.
5. El sistema guarda la información y confirma la creación del nuevo empleado en la página web de la empresa.

### **Escenario de eliminación de empleado**

1. El responsable busca el empleado y hace clic sobre el ícono de eliminar.
2. El sistema muestra un mensaje de aviso para confirmar eliminación de empleado.
3. El responsable confirma la acción de borrado.
4. El sistema elimina toda información de ese empleado y muestra un mensaje de confirmación del borrado.

### **Escenario de lectura de información de empleado**

1. El responsable accede al listado de empleados.
2. El responsable busca el empleado y hace clic sobre su cuenta.
3. El sistema carga y muestra el perfil del empleado.
4. El responsable, en el perfil de la cuenta del empleado ya tendría disponible toda la información sobre ese empleado.

### **Escenario de actualización de información de empleado**

1. El responsable busca el empleado y hace clic sobre el icono de editar.
2. El sistema carga y muestra el perfil del empleado, con permisos de edición sobre los campos de la información del empleado.
3. El responsable selecciona lo(s) campo(s) que quiere editar.
4. El responsable edita y hace clic en 'Confirmar cambios'.
5. El sistema valida la información y la actualiza.

### **Escenarios alternativos**

<b>Escenario alternativo de creación de empleado</b>	
4a. El sistema informa que los datos introducidos no son válidos.	
4b. El responsable debe de volver a introducir los datos.	
<b>Escenario alternativo de eliminación de empleado</b>	
3a. El responsable hace clic sobre el botón 'Cancelar'.	
3b. El sistema deshace la transacción de borrado.	
<b>Escenario alternativo de actualización de información de empleado</b>	
5a. El sistema comprueba los nuevos datos introducidos en los campos.	
5b. El sistema comprueba que ha introducido datos como DNI o email ya existentes en el sistema.	
5c. El sistema muestra mensaje de error de datos ya existentes en el sistema.	
<b>Clases de análisis</b>	
<b>A. Clases de entidad</b>	Empleados, Usuarios
<b>B. Clases de control</b>	ControllerUsuarios
<b>C. Clases de interfaz</b>	ViewPaginaAdmin, ViewEmpleados

Cuadro 120: Especificación de casos de uso: Gestionar empleados. Fuente: elaboración propia.

<b>Título</b>	Iniciar sesión
<b>Descripción</b>	Un cliente decide entrar con su cuenta registrada en el sistema para realizar reservas, agregar comentarios a actividades para dejar su opinión o ver su historial de reservas.
<b>Pre-condición</b>	Estar registrado en el sistema.
<b>Post-condición</b>	El cliente accede con su cuenta existente en el sistema y puede hacer uso de las funcionalidades de clientes en la aplicación web.
<b>Prioridad</b>	Alta
<b>Escenario principal</b>	

<b>Escenario de inicio de sesión</b>	
1. El usuario accede a la página web	
2. El usuario hace clic en 'Iniciar sesión'	
3. El sistema carga y muestra el formulario de inicio de sesión	
4. El usuario introduce sus datos correspondientes	
5. El sistema verifica y valida los datos introducidos con los almacenados en el sistema	
6. El sistema verifica correctamente el usuario, el usuario inicia sesión y carga la pantalla principal de la página web	
<b>Escenarios alternativos</b>	
<b>Escenario alternativo datos incorrectos</b>	
6a. El sistema comprueba que los datos introducidos son incorrectos o no existen	
6b. El sistema muestra mensaje de error de datos incorrectos	
<b>Clases de análisis</b>	
<b>A. Clases de entidad</b>	Usuarios, Clientes
<b>B. Clases de control</b>	ControllerLogin
<b>C. Clases de interfaz</b>	ViewPaginaPrincipal, ViewIniciarSesión, ViewPaginaPrincipal-Clientes

Cuadro 121: Especificación de casos de uso: Inicio de sesión. Fuente: elaboración propia.

<b>Título</b>	Realizar reserva de una actividad turística
<b>Descripción</b>	Un usuario decide realizar la reserva de una actividad turística seleccionada.
<b>Pre-condición</b>	Elegir actividad turística y que esté disponible.
<b>Post-condición</b>	El usuario habrá reservado una actividad turística tras haber introducido los datos de reserva requeridos y pagado la reserva.
<b>Prioridad</b>	Alta
<b>Escenario principal</b>	

<b>Escenario de reserva de actividad turística</b>	
1. El usuario escoge la actividad que desea realizar	
2. El sistema muestra los detalles de la actividad escogida	
3. El usuario escoge la fecha de la actividad y comprueba su disponibilidad	
4. El sistema verifica la disponibilidad y muestra que sí que está disponible la actividad para la fecha seleccionada	
5. El usuario hace clic en 'Reservar'	
6. El sistema redirige al usuario a rellenar los datos necesarios para la reserva	
7. El usuario introduce los datos necesarios para la reserva	
8. El usuario hace clic en 'Realizar pago'	
9. El sistema redirige al usuario a una plataforma de pago	
10. Finalizado el pago, el sistema guarda la reserva y muestra confirmación al usuario junto a la factura generada	
<b>Escenarios alternativos</b>	
<b>Escenario alternativo actividad no disponible</b>	
4a. El sistema informa al usuario que no hay disponibilidad para esa actividad con la fecha indicada	
4b. El usuario decide escoger otra fecha o salir de la actividad seleccionada	
<b>Escenario alternativo campos no válidos</b>	
7a. El usuario deja campos obligatorios sin completar o introduce datos inválidos	
7b. El sistema muestra un mensaje de error indicando los campos a corregir	
7c. El usuario corrige los datos y continúa o decide salir del formulario de reserva	
<b>Clases de análisis</b>	
<b>A. Clases de entidad</b>	Actividades, Reservas, Factura
<b>B. Clases de control</b>	ControllerActividades, ControllerReservas, ControllerFactura
<b>C. Clases de interfaz</b>	ViewPaginaPrincipal, ViewActividades, ViewReservas, ViewPago

Cuadro 122: Especificación de casos de uso: Reserva de una actividad turística. Fuente: elaboración propia.

<b>Título</b>	Realizar pago de reserva de actividad turística
<b>Descripción</b>	Un usuario realiza el pago de la reserva de una actividad turística disponible que haya seleccionado.
<b>Pre-condición</b>	Elegir actividad turística y haber introducido los datos de reserva
<b>Post-condición</b>	El usuario habrá reservado una actividad turística tras haber realizado el pago, obteniendo factura y confirmación de su reserva.
<b>Prioridad</b>	Alta
<b>Escenario principal</b>	
<b>Escenario de pago de reserva de actividad turística</b>	
<ol style="list-style-type: none"> <li>1. El sistema carga la pasarela de pago al usuario</li> <li>2. El usuario elige entre dos métodos de pago; Paypal o Transferencia Bancaria</li> <li>3. El usuario selecciona un método de pago</li> <li>4. El usuario introduce los datos para el pago</li> <li>5. El sistema proporciona campos para los datos necesarios para generar la factura</li> <li>6. El usuario introduce los datos en los campos necesarios para Realizar pago</li> <li>7. El usuario hace clic en 'Realizar pago'</li> <li>8. El sistema procesa el pago</li> <li>9. El sistema verifica el pago y muestra un mensaje de confirmación de pago, devolviendo confirmación y factura de pago de reserva de actividad turística</li> </ol>	
<b>Escenarios alternativos</b>	

<b>Escenario alternativo cancelación de reserva antes del pago</b>	
4a. El usuario decide no continuar y cierra la página o hace clic en 'Cancelar'.	
4b. El sistema descarta la reserva	
4c. Se muestra un mensaje de confirmación de cancelación de reserva de actividad turística	
<b>Escenario alternativo error en plataforma de pago</b>	
7a. Se produce un fallo durante el proceso de pago, que puede ser por error técnico con la pasarela de pago o la tarjeta ha sido rechazada	
7b. El sistema detecta que el pago no fue completado correctamente	
7c. El sistema muestra un mensaje al usuario indicando el problema	
7d. El usuario decide reintentar el proceso, seleccionar otro método de pago o cancelar la transacción	
<b>Clases de análisis</b>	
<b>A. Clases de entidad</b>	Factura
<b>B. Clases de control</b>	ControllerPago
<b>C. Clases de interfaz</b>	ViewPago

Cuadro 123: Especificación de casos de uso: Pago de reserva de una actividad turística. Fuente: elaboración propia.

### B.1.3. Diagramas de secuencia

Aquí se muestra el resto de diagramas de secuencia desarrollados, siendo los procesos de negocios más relevantes.

La figura 64 muestra un conjunto de secuencias similar, pero cambiando las vistas. En esta, el cliente introduce sus credenciales y si son validadas, el sistema continúa con la carga de la vista principal de clientes, permitiendo así el acceso del usuario a las funcionalidades y servicios destinados a los clientes registrados.

La gestión de empleados es un proceso de negocio destacado que muestra las distintas secuencias para operaciones típicas CRUD<sup>18</sup>. Dichas operaciones siguen la misma lógica para

<sup>18</sup>Create, Read, Update, Delete

la gestión de otras entidades por lo que solo se ha decidido mostrar solo la secuencia de la gestión de usuarios del tipo empleado.

El diagrama que aparece en la figura 65 muestra el proceso mediante el cual el responsable de la aplicación web (y, por lo tanto, de la empresa de reservas de actividades turísticas) accede al sistema y crea un nuevo empleado dentro de una interfaz de administración. Desde esta pantalla, el responsable selecciona la opción para gestionar empleados, lo cual es procesado por el controlador de usuarios. Este controlador valida el acceso del responsable y carga la lista actual de usuarios de tipo empleado. La lista es devuelta y cargada en la interfaz para que el responsable la visualice. Luego, el responsable pulsa el botón para crear un nuevo usuario, lo que activa la carga del formulario para un nuevo empleado. Una vez cargado, el responsable introduce los datos del nuevo trabajador. Si no ha habido errores en los datos, el sistema valida los datos y crea un nuevo usuario de tipo empleado con las credenciales introducidas por el responsable.

La primera secuencia de pasos es idéntica en toda la gestión de empleados, cuando se accede a este apartado de gestión dentro de la interfaz de administración, el sistema devuelve la lista de usuarios de tipo empleados que existen registrados en él.

Para la operación de borrado, la figura 66 representa el proceso mediante el cual el responsable de la web elimina a un empleado existente en el sistema desde la interfaz de administración, dentro de la misma sección de gestión de empleados. Dentro, el responsable pulsa el botón para eliminar un empleado específico. El sistema responde mostrando un mensaje de confirmación y el responsable puede cancelar el proceso o continuar y eliminar correctamente a un empleado.

Seguido de las operaciones básicas de gestión, se encuentra la lectura de información de la ficha de un empleado. Como se aprecia en la figura 67, el responsable hace clic en la opción para ver un empleado en particular y esto genera una solicitud al controlador de usuarios, que se encarga de obtener los datos del empleado a partir de su identificador. Una vez obtenida la información, se muestra en una ficha detallada del empleado, permitiendo al responsable consultar sus datos.

Finalmente, entre las operaciones de tipo CRUD se encuentra la edición de información de un empleado existente. El diagrama de la figura 68 representa el proceso mediante el cual un responsable accede al sistema y actualiza la información de un empleado. El responsable

hace clic en la opción para editar un empleado específico. Esto hace que el sistema cargue el formulario de edición, donde el responsable puede modificar ciertos datos y actualizar su información.

La actividad de negocio de la empresa se centra en las reservas de actividades turísticas. Por ello, se muestra en la figura 69 la secuencia de mensajes y clases de análisis involucradas. En primer lugar, este diagrama de secuencia está orientado a clientes y usuarios anónimos (no registrados en el sistema) que deciden reservar una actividad turística en caso de que esté disponible. Primero, el cliente o usuario anónimo decide acceder a la página principal, luego acceder a la sección de actividades. En esta sección, se carga una vista con la lista de actividades disponibles. Dentro, el usuario selecciona una actividad e introduce una fecha deseada, y el sistema consulta su disponibilidad en la fecha elegida. Si hay disponibilidad, se confirma al usuario. El usuario pulsa el botón para reservar, y el sistema carga un formulario con los datos de la reserva e introduce los datos requeridos para la misma. Tras esto, hace clic en realizar pago y es redirigido a una plataforma en la que elige entre dos métodos de pago, introduce los datos correspondientes de pago y los necesarios para generar la factura y finaliza la transacción. Una vez confirmado el pago, el sistema guarda la reserva y se le muestra confirmación de su reserva y se le proporciona la factura. Finalmente, el proceso de pago de reserva de actividad turística. Como se muestra en la figura 70, el proceso de pago de una reserva turística comienza cuando el usuario accede a la pasarela de pago. Allí selecciona el método deseado, introduce los datos correspondientes para el pago y para la factura y confirma la transacción. Una vez enviados, el sistema valida la información ingresada y, si es correcta, procede a generar automáticamente la factura. Finalmente, el sistema muestra al usuario un mensaje de confirmación de la reserva junto con la factura correspondiente.

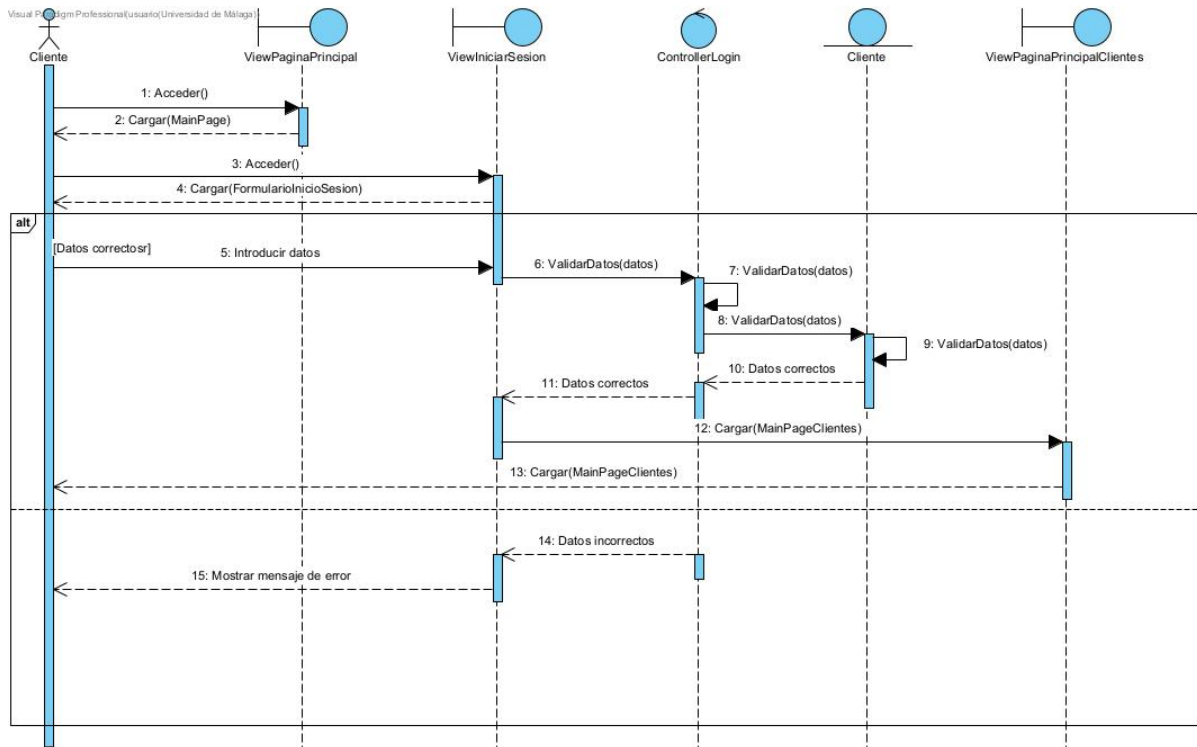


Figura 64: Inicio de sesión en el sistema por parte de un cliente. Fuente: elaboración propia.

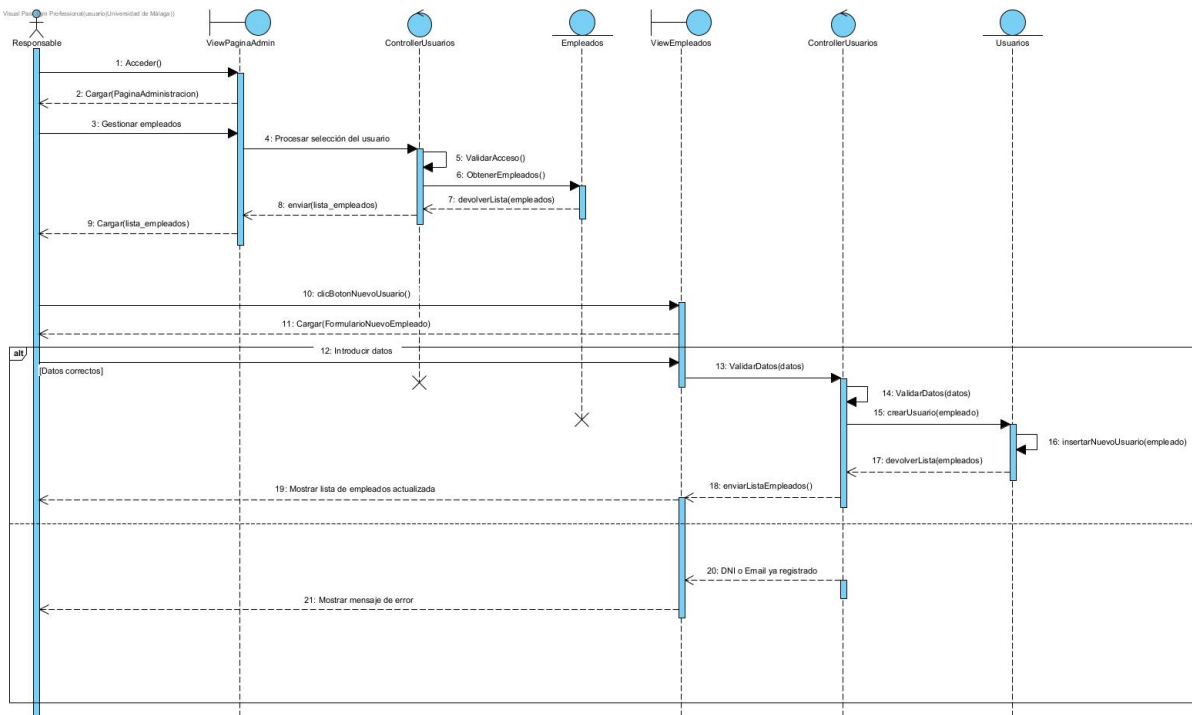


Figura 65: Gestión de empleados: Crear nuevo empleado. Fuente: elaboración propia.

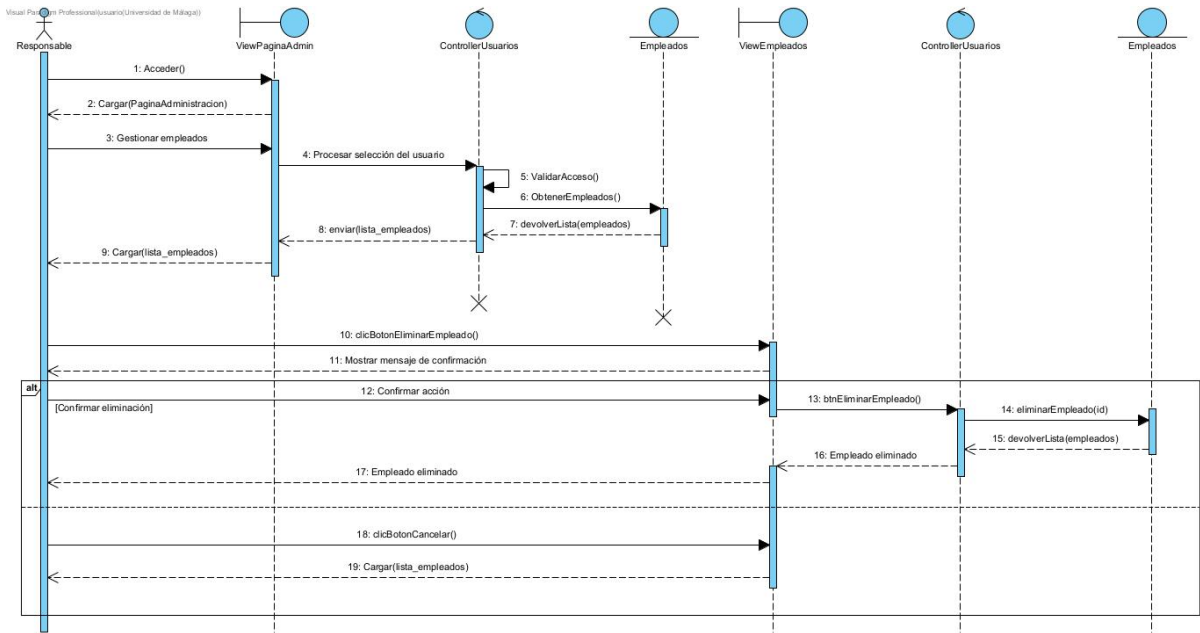


Figura 66: Gestión de empleados: Eliminar empleado. Fuente: elaboración propia.

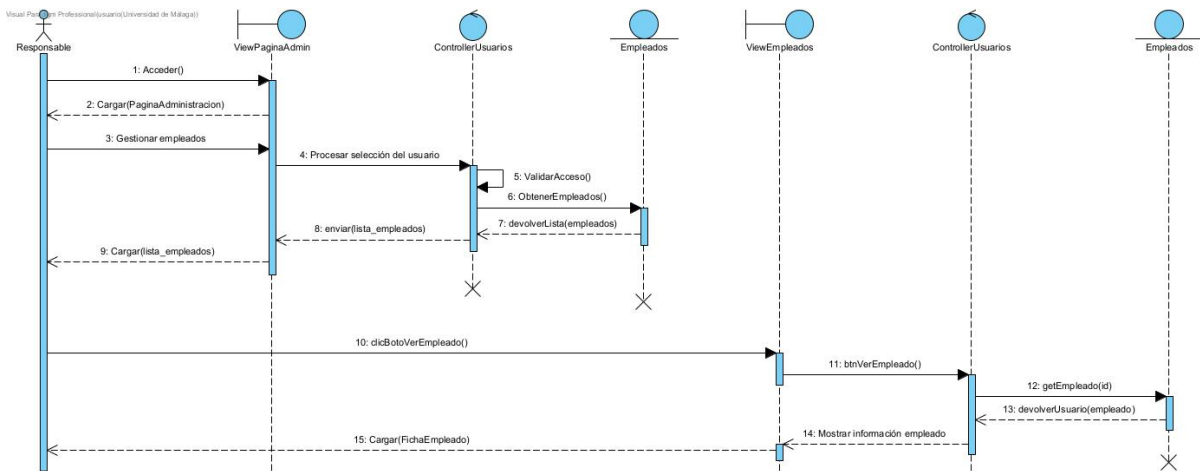


Figura 67: Gestión de empleados: Leer información de empleado. Fuente: elaboración propia.

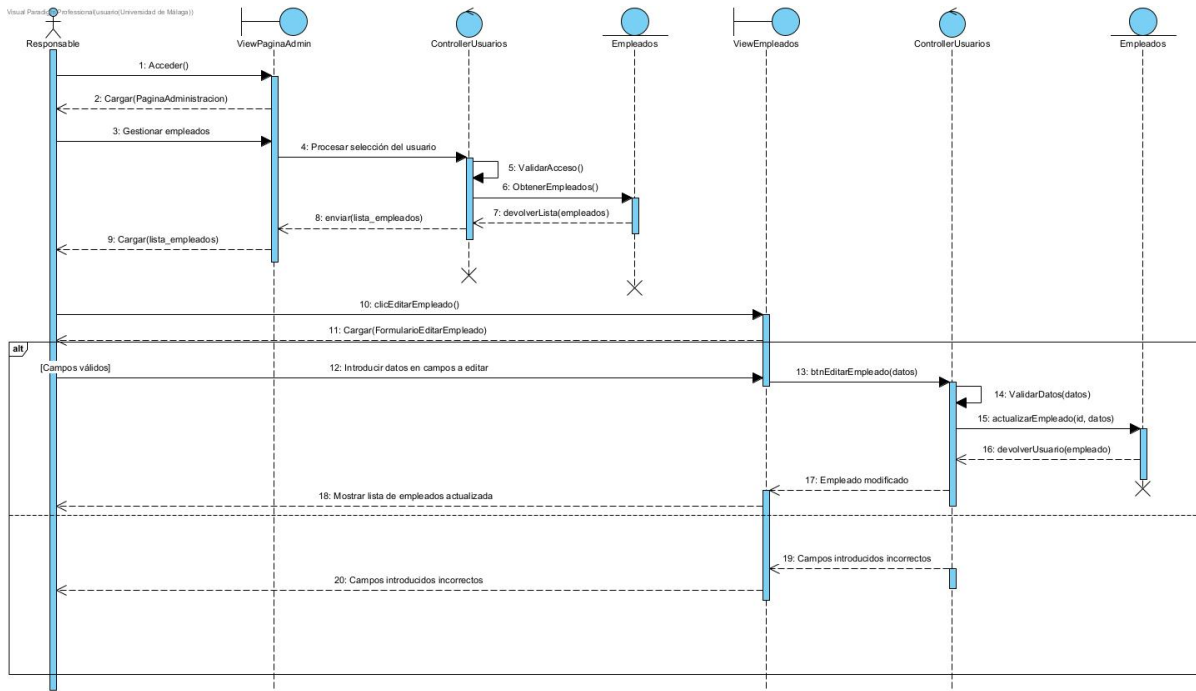


Figura 68: Gestión de empleados: Actualizar información de empleado. Fuente: elaboración propia.

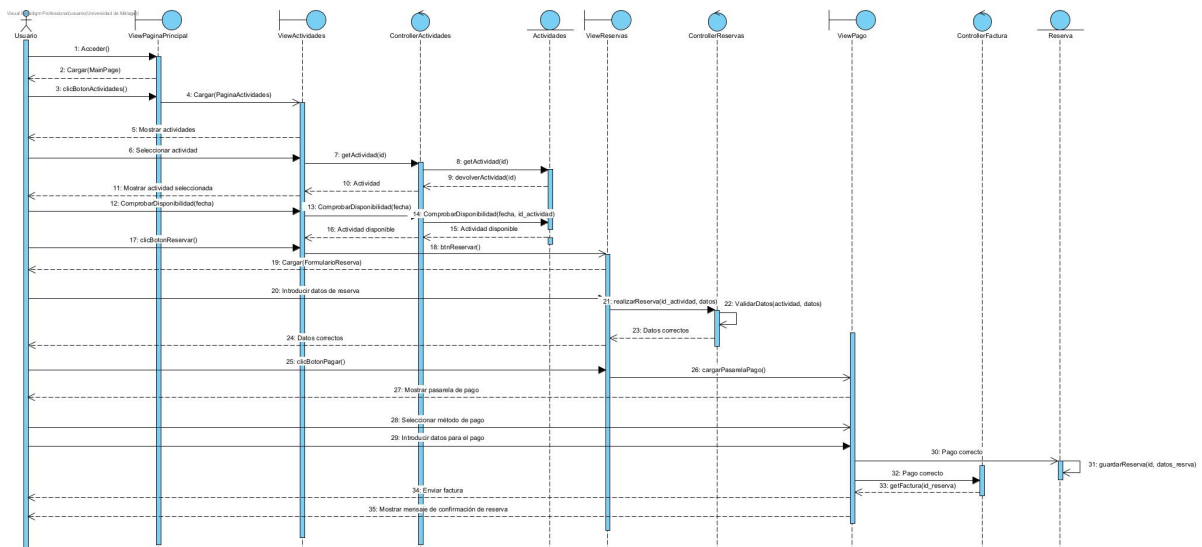


Figura 69: Realizar reserva de actividad turística. Fuente: elaboración propia.

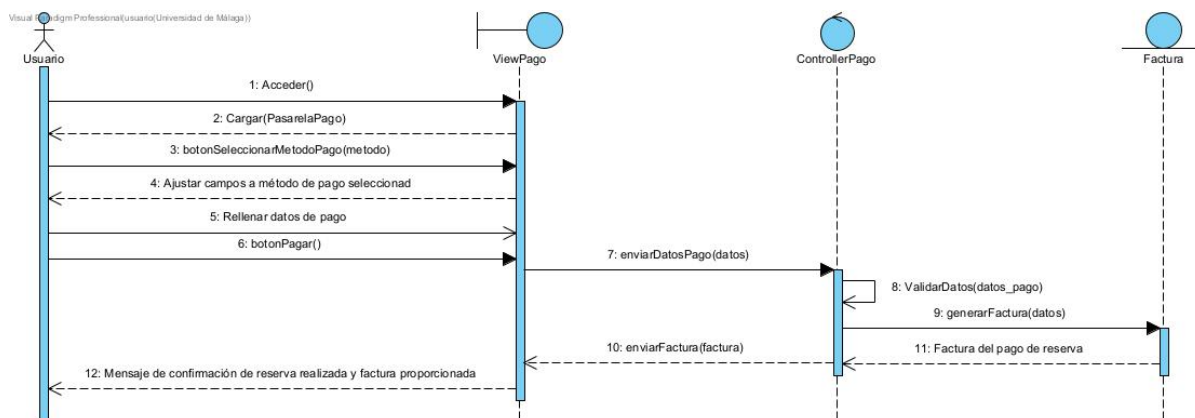


Figura 70: Realizar pago de reserva de actividad turística. Fuente: elaboración propia.

### B.1.4. Formatos de pantalla

A continuación, se presentan los distintos formatos de pantalla desarrollados, los cuales fueron diseñados considerando los procesos de negocio relevantes mencionados con anterioridad. Las maquetas de interfaces de usuario desarrolladas son las siguientes:

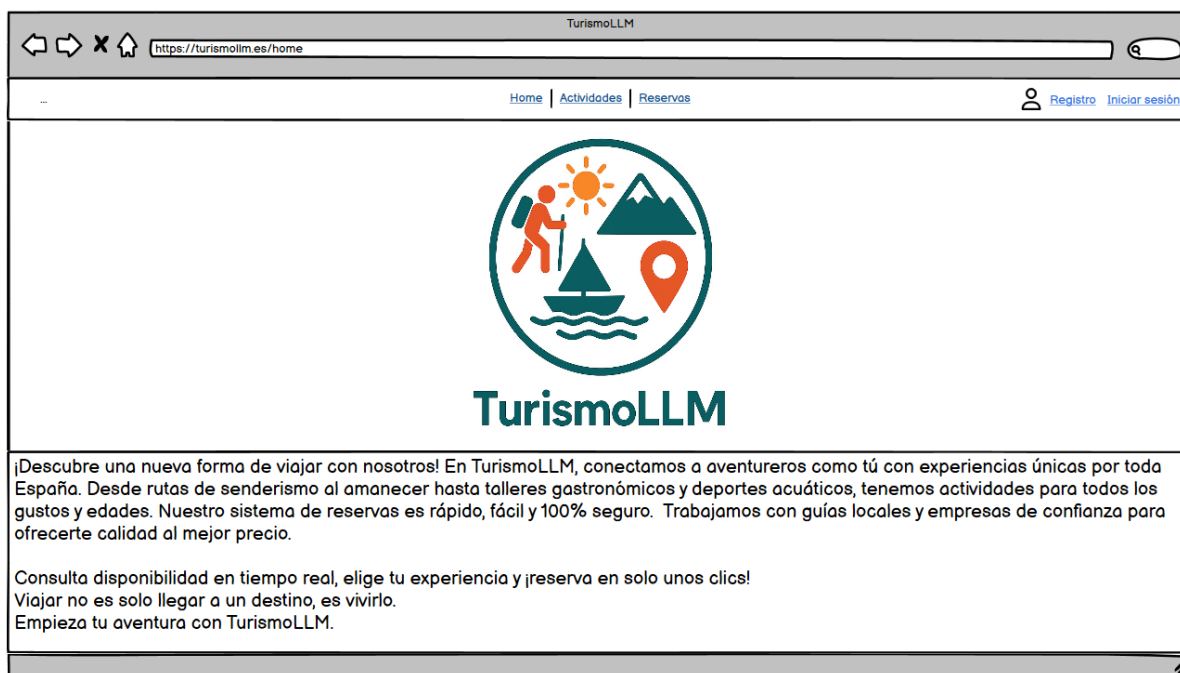


Figura 71: Página principal de la aplicación web. Fuente: elaboración propia.

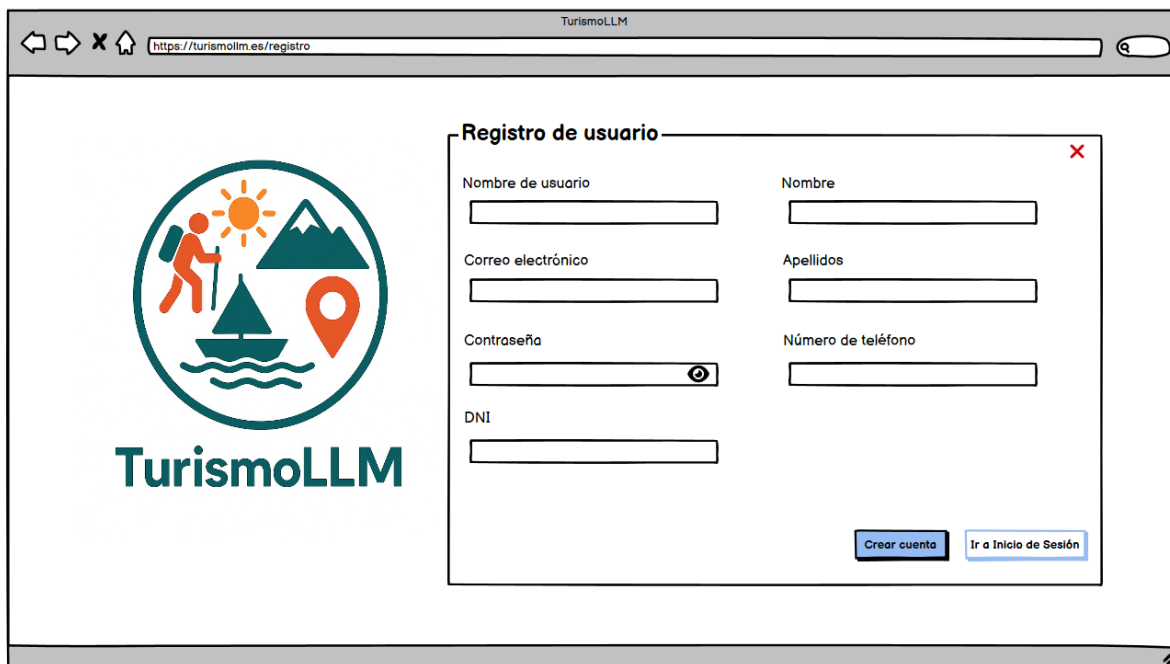


Figura 72: Maqueta de pantalla de registro de usuario. Fuente: elaboración propia.

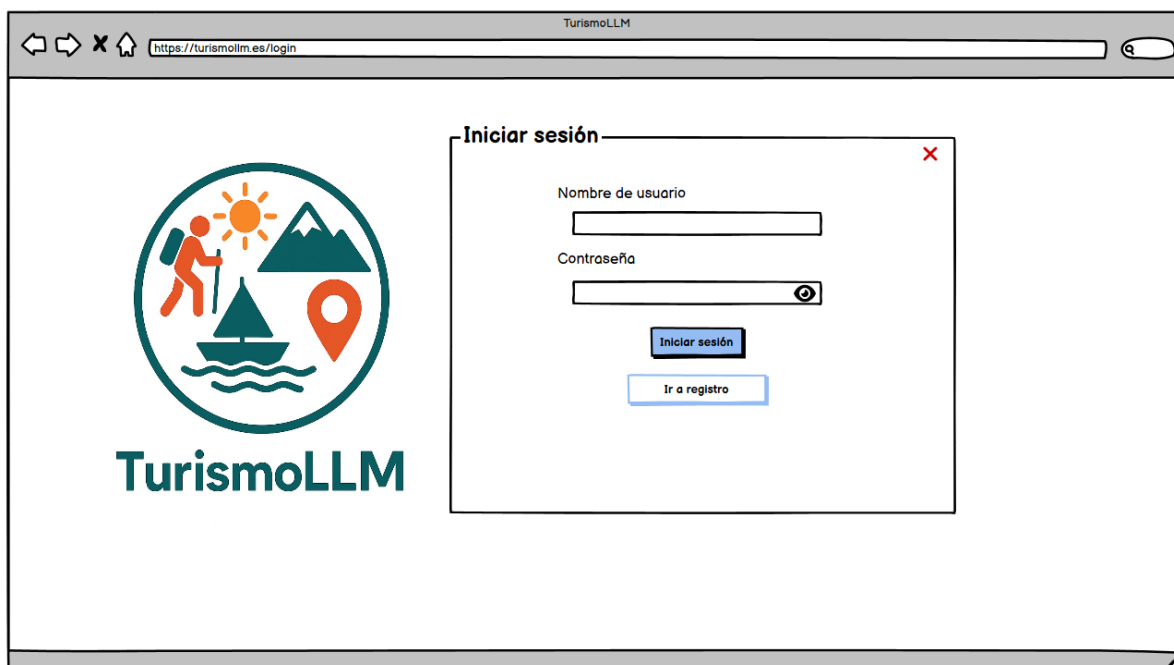


Figura 73: Maqueta de interfaz de inicio de sesión de cliente ya registrado. Fuente: elaboración propia.

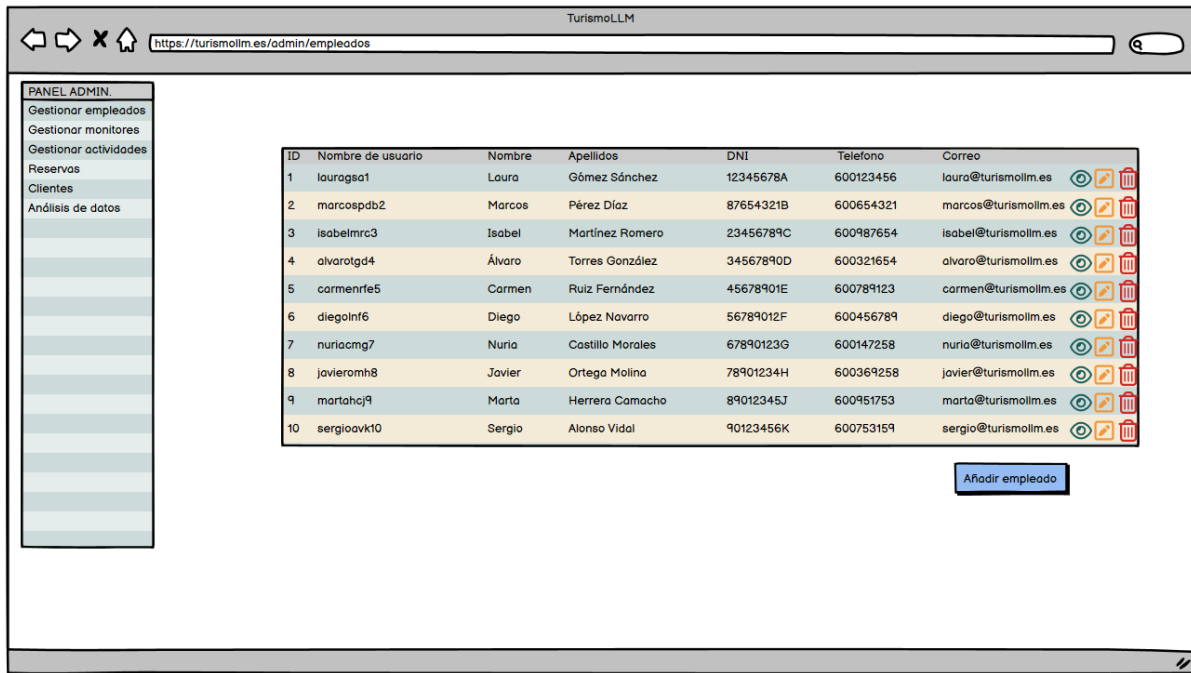


Figura 74: Maqueta de pantalla de administración: Gestionar empleados. Fuente: elaboración propia.

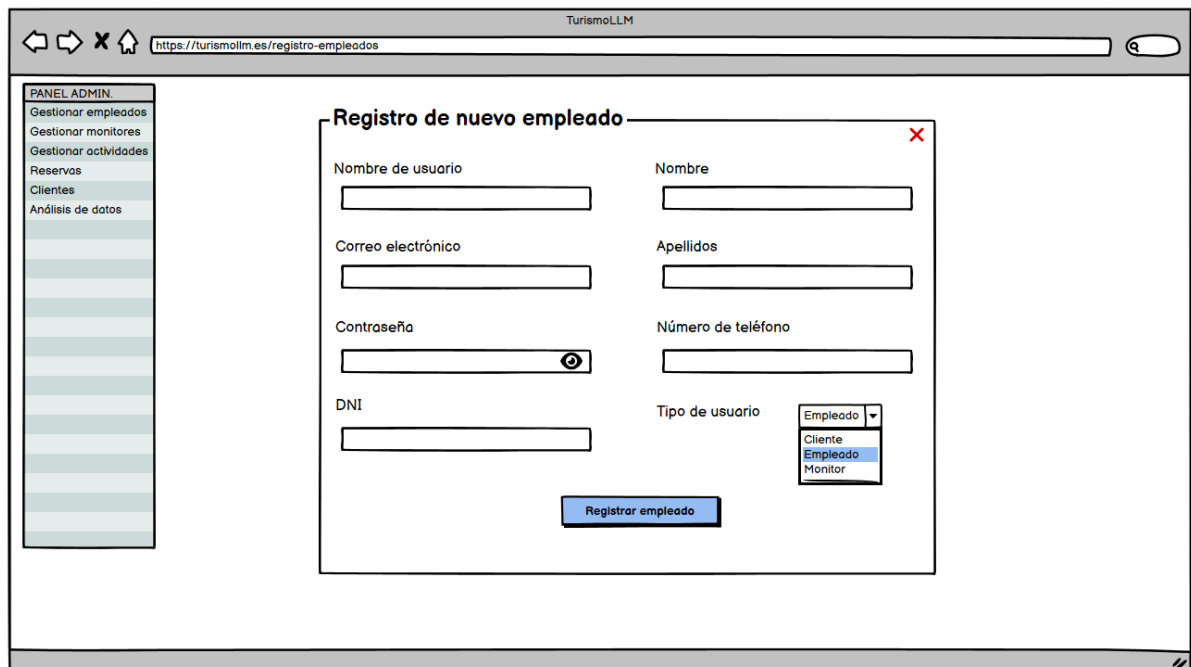


Figura 75: Maqueta de gestión de empleados: Crear nuevo empleado. Fuente: elaboración propia.

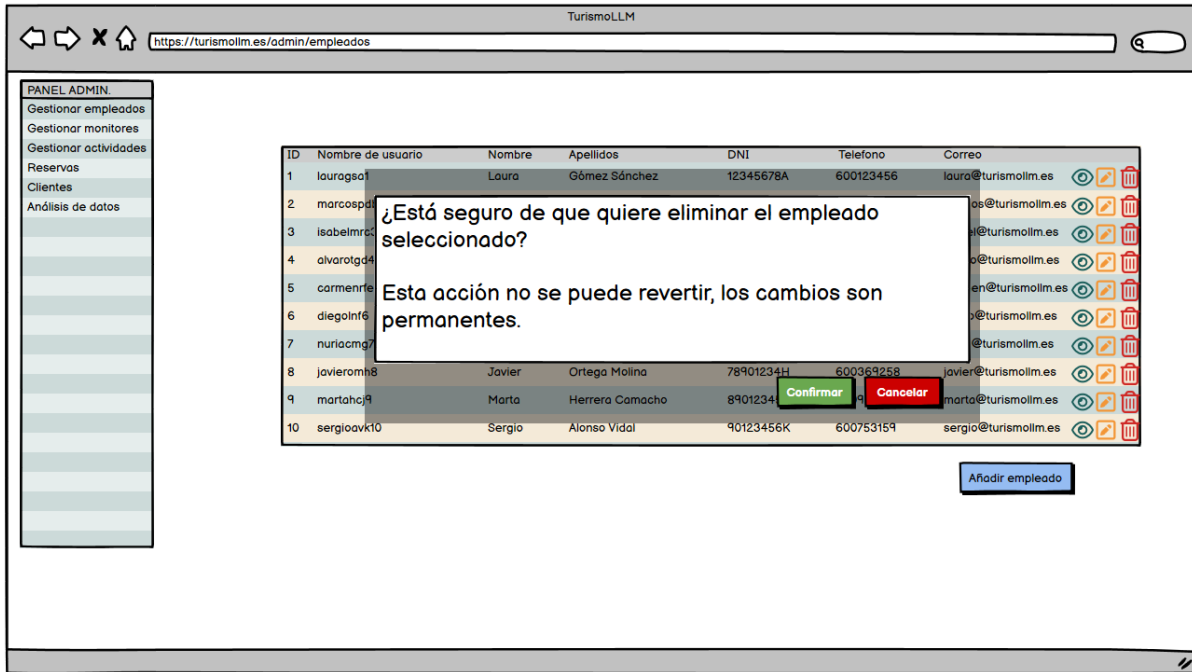


Figura 76: Maqueta de gestión de empleados: Eliminar empleado. Fuente: elaboración propia.

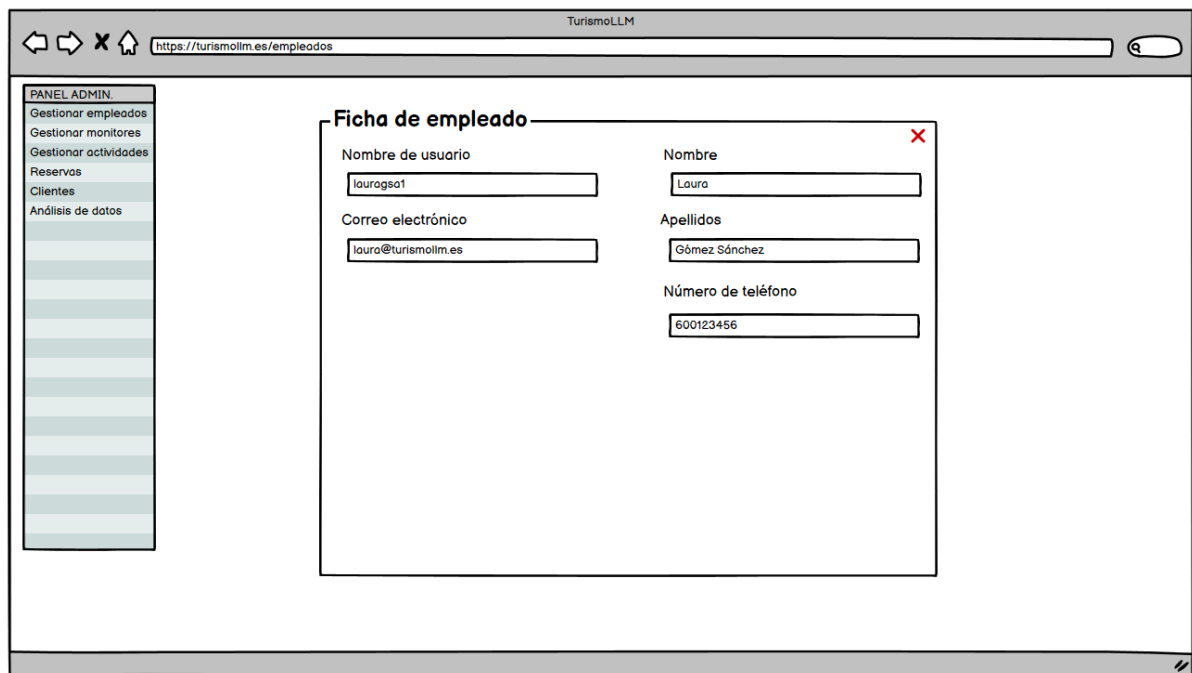


Figura 77: Maqueta de gestión de empleados: Visualizar empleado. Fuente: elaboración propia.

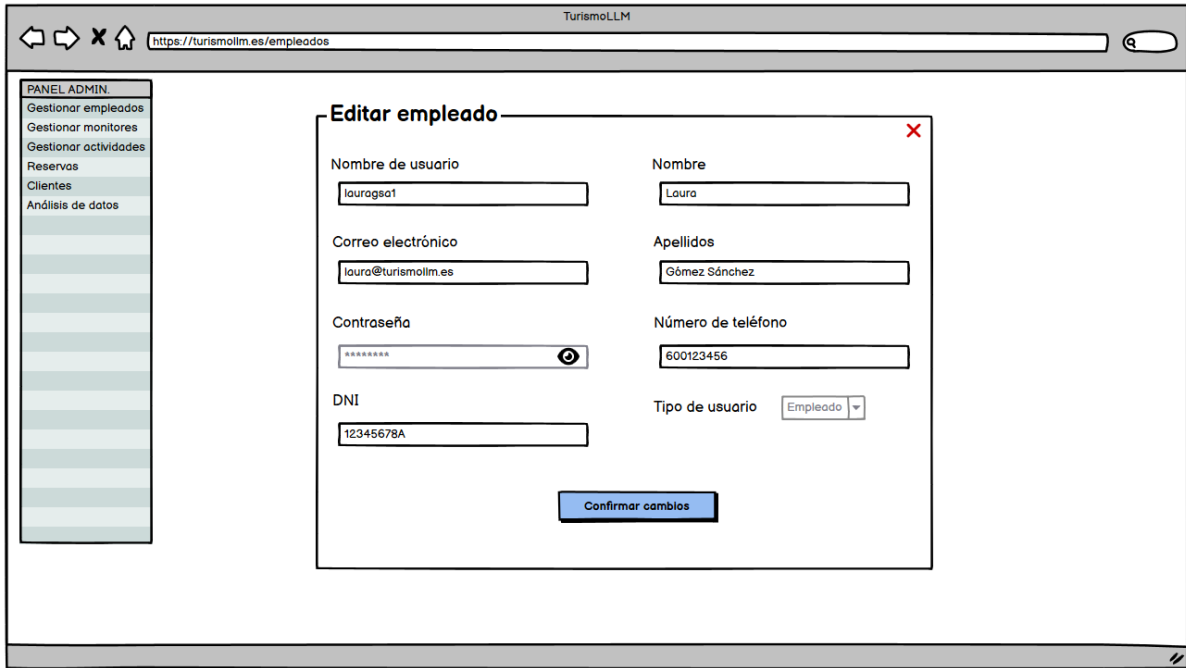


Figura 78: Maqueta de gestión de empleados: Editar empleado. Fuente: elaboración propia.

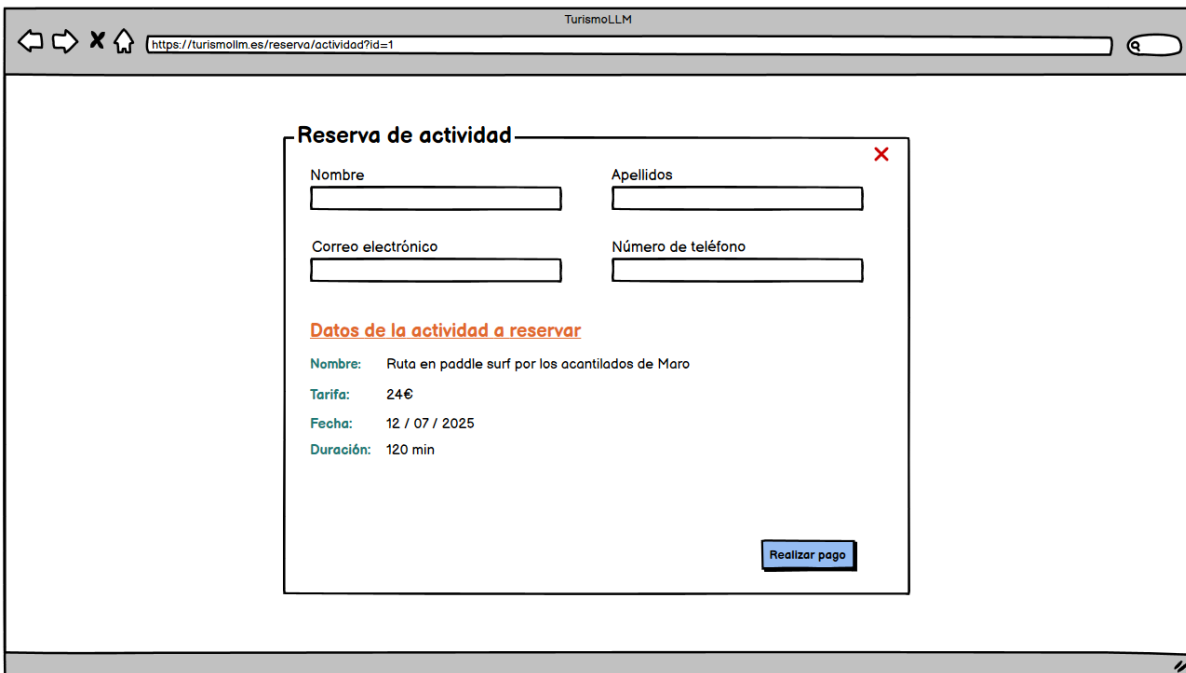


Figura 79: Maqueta de pantalla de reserva de actividad turística. Fuente: elaboración propia.

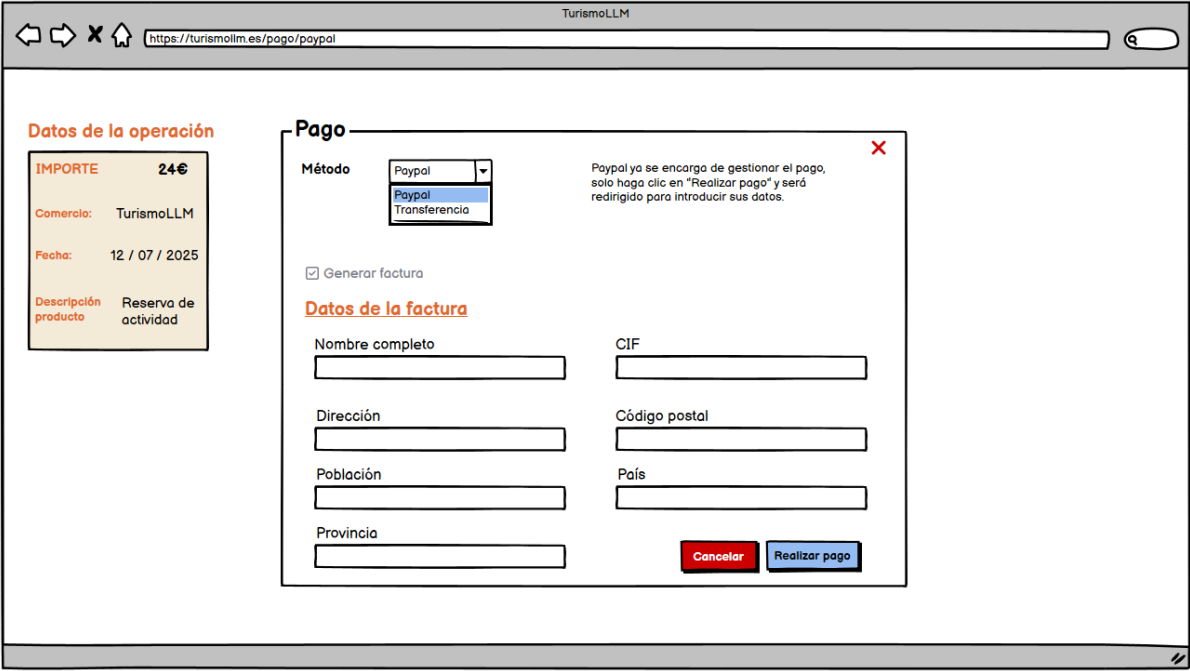


Figura 80: Maqueta de pantalla de pago de reserva de actividad. Fuente: elaboración propia.

## B.2. Implementación

Se recogen capturas de ciertas partes de código *frontend* y *backend*. De esta forma, se pretende proporcionar más contexto y una visión más completa de la implementación sin necesidad de tener que acceder al código fuente de la entrega.

### B.2.1. Registro de usuario y Login

```
constructor(private fb: FormBuilder) {
  this.registroForm = this.fb.group({
    username: ['', [Validators.required, Validators.maxLength(16)]],
    email: ['', [Validators.required, Validators.email]],
    password: ['', [Validators.required]],
    nombre: ['', [Validators.required, Validators.maxLength(16)]],
    apellidos: ['', [Validators.required, Validators.maxLength(48)]],
    telefono: ['', [Validators.nullValidator, Validators.maxLength(14)]],
    dni: ['', [Validators.required, Validators.minLength(9), Validators.maxLength(9)]],
  });
}
```

Figura 81: Ejemplo de formulario reactivo en *frontend*. Fuente: elaboración propia.

```
registro(usuario: any): Observable<any> {
  return this._http.post(`${environment.API_URL}/auth/registro`, usuario).pipe(tap(
    (response) => this._storage.set('session', JSON.stringify(response))
  ));
}

login(credenciales: any) {
  return this._http.post(`${environment.API_URL}/auth/login`, credenciales).pipe(tap(
    (response) => this._storage.set('session', JSON.stringify(response))
  ));
}
```

Figura 82: Código fuente para solicitud HTTP desde *frontend* hacia *backend*. Fuente: elaboración propia.

```
async login(dto: AuthDTO) {
  const usuario = await this.prisma.usuarios.findUnique({
    where: {
      nombre_usuario: dto.username,
    }
  });
  if (!usuario) throw new NotFoundException('Credenciales de acceso erróneas');

  const pwValida = await argon2.verify(usuario.hash_password, dto.password);
  if (!pwValida) throw new UnauthorizedException('Credenciales de acceso erróneas');

  return this.firmarToken(usuario.id, usuario.nombre_usuario, usuario.tipo_usuario);
}

async firmarToken (user_id: number, nombre_usuario: string, tipo_usuario: usuarios_tipo_usuario): Promise< {access_token: string} > {
  const payload = {
    sub: user_id,
    nombre_usuario,
    tipo_usuario
  };

  const token = await this.jwt.signAsync(payload);
  return { access_token: token };
}
```

Figura 83: Código fuente para *login* y firma de *token* en *backend*. Fuente: elaboración propia.

### B.2.2. Gestión de empleados

```
import { Injectable } from '@nestjs/common';
import { AuthGuard } from '@nestjs/passport';

@Injectable()
export class JwtAuthGuard extends AuthGuard('jwt') {}
```

Figura 84: Implementación *JwtAuthGuard* en *backend*. Fuente: elaboración propia.

```
@Injectable()
export class RolesGuard implements CanActivate {
  constructor(private reflector: Reflector) {}

  canActivate(ctx: ExecutionContext): boolean {
    const requiredRoles = this.reflector.getAllAndOverride<usuarios_tipo_usuario[]>(
      ROLES_KEY,
      [ctx.getHandler(), ctx.getClass()],
    );

    if (!requiredRoles || requiredRoles.length === 0) return true;

    const request = ctx.switchToHttp().getRequest();
    const user = request.user as { tipo_usuario?: usuarios_tipo_usuario };

    if (!user) throw new ForbiddenException('No autenticado');

    const allowed = requiredRoles.includes(user.tipo_usuario as usuarios_tipo_usuario);
    if (!allowed) throw new ForbiddenException('No tienes permisos para esta operación');

    return true;
  }
}
```

Figura 85: Implementación RolesGuard en *backend*. Fuente: elaboración propia.

```
export const ROLES_KEY = 'roles';
export const Roles = (...roles: usuarios_tipo_usuario[]) => SetMetadata(ROLES_KEY, roles);
```

Figura 86: Implementación de Roles en *backend*. Fuente: elaboración propia.



UNIVERSIDAD  
DE MÁLAGA

| [uma.es](http://uma.es)

E.T.S de Ingeniería Informática  
Bulevar Louis Pasteur, 35  
Campus de Teatinos  
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA