ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADO EN INGENIERÍA DEL SOFTWARE

SISTEMA DE COMUNICACION PADRES-PROFESORES

PARENTS-TEACHERS COMMUNICATION SYSTEM

Realizado por
Naoual Amasri
Tutorizado por
José María Álvarez Palomo
Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, Diciembre de 2016

Fecha defensa:
El Secretario del Tribunal

Resumen:

Una buena comunicación entre padres y profesores es vital para garantizar una buena evolución y unos buenos resultados académicos de nuestros hijos. De ahí surge la idea de nuestro Sistema de comunicación padres-profesores, un sistema web cuyos usuarios habituales serán los tutores académicos y los tutores legales de los alumnos, para que la comunicación entre ambos sea ágil, fluida y eficiente.

Para llevar a cabo el desarrollo de éste software hemos seguido un desarrollo iterativo, una iteración inicial para un estudio global del problema y las tecnologías a utilizar, seguida de cinco iteraciones donde al inicio de cada una, seleccionamos las funcionalidades que más valor aportan a nuestro sistema, de manera que al final de cada iteración obtenemos una versión funcional y demostrable.

En cuanto a aspectos más técnicos, nuestro sistema cuenta con una base de datos relacional MySQL y un contenedor Tomcat. Para la implementación, hemos creído conveniente usar Spring Framework por su modularidad, flexibilidad y facilidad a la hora de implementar aplicaciones empresariales. Además, hoy en día se encuentra entre los frameworks más usados en el ámbito laboral.

Palabras clave: educación, comunicación, Spring Framework, software, aplicación web, tutor, profesores, padres

Abstract:

A good and fluent communication between parents and teachers is basic to guarantee for our children to get good progress and academic results. This the why behind our parents-teachers communication system. A system whose usual users will be students' academic tutors and their legal guardians, so that the communication between both is agile, fluent and efficient.

To carry out the development of this software, we have followed an iterative and incremental approach. The process was divided in several iterations, an initial one where we did a global study to have an overview about the system, followed by five more iterations, where we have evolved the product until the final result has been finished. At the beginning of each iteration, we select the requirements most valued, so at the end of each one, we get a functional and deliverable version.

With respect to the technical aspects, our system has a MySQL relational database and it will be deployed to a Tomcat container. The implementation has been done using the Spring Framework due to its modularity, flexibility and the facility that it provides in developing enterprise applications, besides being one of the frameworks most used professionally.

Keywords: education, communication, Spring Framework, software, Web Application, tutor, teachers, parents

Contenido

Capítulo 1. Introducción

El presente documento pertenece al trabajo fin de grado titulado “Sistema de comunicación padres-profesores”, de aquí en adelante abreviado como SCPP. En él expondremos los objetivos principales del trabajo, las principales tecnologías y herramientas que han sido utilizadas, así como un recorrido por todas las fases por las que hemos pasado para llevar a cabo este sistema software.

1.1. Motivación

Para garantizar un buen desarrollo académico de nuestros hijos es importante tener un seguimiento continuo de su rendimiento, de sus resultados y de la impresión que tienen sus profesores de ellos. De esta manera, se podrán prever situaciones de bloqueo o deterioro que puedan obstaculizar el proceso de su aprendizaje.

El sistema tradicional impide que ese seguimiento se haga de manera efectiva, ya que en ocasiones, tanto por las circunstancias del personal académico como por las circunstancias de los familiares, puede llegar a ser muy lento.

Todo lo anteriormente expuesto, ha sido la motivación principal para comenzar el presente trabajo e intentar dar una solución que mejore la situación actual.

1.2. Objetivos

Los objetivos principales del trabajo son:

- Implementar una herramienta informática que facilite y agilice la comunicación entre padres y profesores.
- Comprensión y aprendizaje de las metodologías y tecnologías escogidas.

1.3. La aplicación

El sistema que pretendemos diseñar e implementar es un sistema web cuyo público objetivo son los profesores y los padres de los alumnos, así que el escenario principal de este software va a ser en los institutos de educación secundaria, aunque también tiene cabida en colegios y otros centros similares.

El objetivo principal del aplicativo es crear un canal de comunicación entre los tutores legales y académicos, un canal ágil, fluido y eficiente frente al sistema tradicional. Esa comunicación se va a realizar mediante el intercambio de mensajes, además de facilitar la forma de concertar reuniones entre ambas partes.

Otro aspecto importante de la aplicación es que va permitir a los profesores registrar las diferentes tareas propuestas, pudiendo indicar detalles como la fecha de vencimiento, etc. Este registro podrá ser consultado tanto por parte de los padres como por parte de los alumnos.

1.4. Tecnologías y herramientas

Como se ha mencionado anteriormente, uno de los objetivos de este trabajo es familiarizarnos con las tecnologías y las herramientas que se van a utilizar así como adquirir unos conocimientos prácticos de cada una de ellas. A continuación, mencionamos las más relevantes:

1.4.1. Spring Core

Para el desarrollo de nuestra aplicación hemos escogido Spring Framework, una infraestructura de código abierto que se compone de múltiples módulos que abarcan una gran variedad de servicios que facilitan la implementación de una aplicación empresarial. Además, a día de hoy, es uno de los frameworks más utilizados en el mundo laboral.

1.4.2. Spring Web MVC

Spring Web MVC es uno de los submódulos con los que cuenta Spring Framework que nos proporciona una arquitectura basada en el conocido patrón MVC. Además, nos ofrece componentes listos para ser usados en el desarrollo de aplicaciones web flexibles y débilmente acopladas.

1.4.3. Spring Security

Para la gestión de roles y la autenticación de usuarios, hemos escogido Spring Security que es un módulo de Spring que ofrece medidas para la autenticación y el control de acceso a la aplicación.

1.4.4. Spring Data JPA

Spring Data es un módulo de Spring que a su vez contiene una colección de submódulos asociados a tecnologías de acceso a datos. Sus módulos dan acceso a tecnologías nuevas como bases de datos no relacionales, servicios cloud y framework map-reduce, pero algunos submódulos también ofrecen un soporte robusto sobre las tradicionales bases de datos relacionales.

En concreto, Spring Data JPA es un submódulo que permite implementar repositorios JPA de forma simple. Java Persistence API (JPA) es una API de persistencia desarrollada para Java que maneja datos relacionales siguiendo el patrón ORM. Spring Data JPA permite implementar la capa ORM de una forma simple y reduce las líneas de código necesarias implementando automáticamente la capa de acceso a los datos en base a interfaces de repositorio escritas por el desarrollador.

1.4.5. JSP

Java Server Pages (JSP) es una de las tecnologías que se pueden utilizar para renderizar una vista a partir de modelos de Spring. JSP permite generar páginas web dinámicas basadas en documentos estándar de páginas web como HTML o XML. Al ser basado en Java, se trata de código compilado, por lo tanto ofrece ventajas de estabilidad y rapidez frente a alternativas de código interpretado. Se ejecuta en el lado del servidor, por lo tanto ofrece facilidad y seguridad a la hora de acceder a la base de datos o hacer otros procesos complejos.

1.4.6. JSTL

JSP Standard Tag Library (JSTL) es una agrupación de etiquetas JSP estándares y comúnmente utilizadas en aplicaciones web. JSTL abarca tareas estructurales como iteraciones y condiciones y tareas de manipulación de información como documentos XML, $i_1$8n y extracción de datos a través de SQL. La colección de etiquetas se puede dividir en una serie de grupos, cada uno de los cuales compone una librería. Para utilizar cualquiera de estas librerías, se debe añadir mediante una directiva ‘taglib’.

1.4.7. Log4j

En un sistema informático, una buena gestión de los logs es primordial, ya que es la manera que tenemos para monitorizar el sistema. Log4j es una buena opción para realizar éste tipo de tarea.

1.4.8. Maven

Maven es una herramienta excelente para la construcción de proyectos, ya que simplifica las tareas de compilar y gestionar las dependencias entre otras cosas.

1.4.9. Apache Tomcat 8

Nuestro sistema es un sistema web construido con Spring MVC Web, por lo que va a precisar de un contenedor de servlets para el despliegue. Para ello, hemos optado por Tomcat, un contenedor de servlets capaz de recibir peticiones de páginas web y redireccionarlas a un objeto servlet. La facilidad de instalación y configuración que brinda, hace de él una buena opción.

1.4.10. MySQL

Para la persistencia de datos, MySQL es un sistema de gestión de bases de datos relacionales muy popular de código abierto.

1.4.11. Eclipse Luna

Eclipse es un entorno de desarrollo muy completo, robusto y adaptable, que permite configurar el ambiente de desarrollo según nuestras necesidades. La versión utilizada, es la última disponible en el momento de empezar el presente trabajo.

1.4.12. Subversion

En un proceso de desarrollo software, es importante contar con una herramienta de control de versiones, para ello, hemos escogido Subversion, una herramienta de código libre, eficiente y fácil de utilizar.

Capítulo 2. Funcionalidad requerida

En este capítulo, vamos a exponer la funcionalidad que hemos creído que el sistema debe tener para cumplir con el objetivo anteriormente marcado, teniendo en cuenta el tiempo limitado del que disponemos para la realización del trabajo.

2.1. Actores

Antes de pasar a analizar la funcionalidad requerida, es necesario identificar los actores principales de nuestro sistema. Principalmente tenemos cuatro tipos de usuarios:

- Profesor.
- Padre.
- Alumno.
- Administrador.

2.2. Requisitos funcionales

A continuación, detallamos los requisitos funcionales que han sido seleccionados.

2.2.1. RF01 – Módulo de comunicación padres-profesores

El primer requisito funcional detectado, es crear un módulo que permita la comunicación entre padres y profesores. Esta funcionalidad permitirá que ambas partes intercambien mensajes entre sí, esto incluye crear nuevos mensajes, responder a mensajes recibidos así como consultar el historial de mensajes enviados y recibidos.

RF01.01 – Enviar mensaje

El usuario podrá crear un nuevo mensaje y enviarlo a un destinatario previamente seleccionado. El sistema deberá facilitar al usuario un listado de posibles destinatarios. De esta forma evitamos comunicaciones innecesarias.

Restricciones:

- Un profesor sólo puede enviar mensajes a los padres de sus alumnos.
- Un padre sólo puede enviar mensajes al correspondiente tutor de su hijo.
- Esta funcionalidad no permite el envío de mensajes entre usuarios del mismo perfil.

RF01.02 – Leer un mensaje recibido

El usuario podrá leer un mensaje que se le haya enviado.

RF01.03 – Responder a un mensaje recibido

El usuario podrá responder a un mensaje que haya recibido.

RF01.04 – Visualizar mensajes

El usuario podrá visualizar el listado de mensajes enviado y recibidos.

2.2.2. RF02 – Módulo tutorías Online

Otro aspecto importante para facilitar la comunicación entre los tutores legales y los tutores académicos es facilitar la forma de concertar reuniones.

RF02.01 – Habilitar/deshabilitar franjas horarias para tutorías

La aplicación permitirá a los tutores académicos habilitar/deshabilitar franjas horarias que darán lugar a posibles reuniones. Estas franjas habilitadas serán visibles únicamente para los padres.

RF02.02 – Solicitar tutoría

Los padres podrán solicitar reuniones, previamente habilitadas por un tutor.

RF02.03 – Visualizar horas de tutoría

Los profesores podrán consultar las reuniones que hayan sido concertadas.

2.2.3. RF03 – Gestión de tareas

Este módulo permitirá registrar las tareas encargadas por parte de los profesores para su posterior consulta, tanto por parte de los padres para tener una idea del esfuerzo exigido a su hijo, como por parte de los alumnos, que de esta forma no se perderán ninguna tarea por despiste o ausencia.

RF03.01 – Consultar tareas

Los usuarios podrán consultar las tareas correspondientes a su perfil, es decir:

- Los profesores podrán consultar las tareas creadas por ellos.
- Los alumnos y padres podrán consultar las tareas de las asignaturas en las que el alumno se encuentra matriculado.

RF03.02 – Crear tarea

Un profesor podrá crear una tarea nueva e indicar, si lo desea, su fecha de vencimiento.

RF03.03 – Detalle tarea

Los usuarios podrán consultar el detalle de una tarea.

RF03.04 – Editar tarea

Un profesor puede editar una tarea que haya creado anteriormente.

RF03.05 – Eliminar tarea

Un profesor podrá eliminar una tarea previamente creada.

2.2.4. RF04 – Gestión de cursos

Este módulo permitirá la gestión de cursos académicos, esto es necesario para poder relacionar las asignaturas y los grupos de alumnos. Este módulo sólo será disponible para el administrador del sistema.

RF04.01 – Consultar cursos

El administrador podrá consultar el listado de cursos disponibles.

RF04.02 – Crear curso

El administrador del sistema podrá crear los distintos cursos necesarios.

RF04.03 – Detalle curso

El administrador del sistema podrá consultar el detalle de un determinado curso.

RF04.04 – Editar curso

El administrador del sistema podrá editar la información de un curso.

RF04.05 – Eliminar curso

El administrador del sistema podrá eliminar un determinado curso.

2.2.5. RF06 – Gestión de grupos

Este módulo permitirá agrupar a los alumnos en grupos únicos. La creación, edición y eliminación de estos grupos sólo será disponible para el administrador del sistema.

RF06.01 – Consultar grupos

- Un administrador podrá consultar el listado de todos los grupos del sistema.
- Un tutor académico podrá consultar los grupos tutorizados por él.
- Un padre y un alumno podrán consultar el grupo al que pertenece el alumno.

RF06.02 – Crear grupo

El administrador del sistema podrá crear un grupo.

RF06.03 – Detalle grupo

Todos los usuarios podrán consultar el detalle de un grupo previamente seleccionado.

RF06.04 – Editar grupo

El administrador del sistema podrá editar la información de un grupo.

RF06.05 – Eliminar grupo

El administrador del sistema podrá eliminar un grupo determinado.

RF06.06 – Asignar tutor a un grupo

El administrador del sistema podrá asignar un tutor a un grupo determinado.

2.2.6. RF07 – Gestión de asignaturas

Este módulo permitirá crear las diferentes asignaturas necesarias.

RF07.01 – Consultar asignaturas

- El administrador del sistema podrá consultar el conjunto de todas las asignaturas.
- Un usuario profesor podrá consultar las asignaturas que tiene asignadas.
- Un usuario padre podrá consultar el listado de asignaturas a las que su perfil se encuentra vinculado.

- Un usuario alumno podrá consultar el listado de asignaturas en las que se encuentra matriculado.

RF07.02 – Crear asignatura

El administrador del sistema podrá crear las asignaturas necesarias.

RF07.03 – Detalle asignatura

Los usuarios podrán consultar el detalle de una determinada asignatura. La información mostrada irá en función del rol del usuario conectado.

RF07.04 – Editar asignatura

El administrador del sistema podrá editar la información de una determinada asignatura.

RF07.05 – Eliminar asignatura

El administrador del sistema podrá eliminar una determinada asignatura.

2.2.7. RF08 – Gestión de usuarios

Para la gestión de usuarios, partimos de la suposición de que los centros educativos disponen de sus propios sistemas para la gestión de alumnos, profesorado, etc. Por lo que para nuestro sistema, lo único que nos hace falta es consultar éstos usuarios, y dar de alta el usuario padre.

RF08.01 – Consultar listado de usuarios

- El administrador del sistema podrá consultar el listado de usuarios del sistema.
- Un profesor podrá consultar el listado de una asignatura que imparte, o de un grupo que tutoriza.

RF08.02 – Dar de alta usuario padre

Un usuario padre podrá darse de alta en el sistema, indicando su id y el de su hijo al que será vinculado.

RF08.03 – Ver perfil de usuario

- Los usuarios podrán consultar la información relativa a su perfil de usuario.
- Un padre, además de poder consultar la información de su perfil, podrá consultar la información relativa a los perfiles que tenga vinculado.

2.3. Requisitos no funcionales

A continuación se detallan los requisitos no funcionales que hemos creído necesarios:

RNF01 – Control de acceso

El acceso al sistema será controla mediante id y contraseña.

RNF02 – Funcionalidades visibles según el rol

Los usuarios dispondrán de barras de navegación para acceder a las funcionalidades del sistema. Estas barras irán personalizadas en función del perfil del usuario conectado.

RNF03 – Acceso simultáneo

El sistema permitirá el acceso simultáneo.

RNF04 – Usabilidad

El sistema tendrá un interfaz gráfico amigable, intuitivo y fácil de usar.

RNF05 – Cifrado de claves

Todas las claves se almacenarán de forma cifrada.

RNF06 – Manual de usuario

La aplicación dispondrá de un manual de usuario en línea, para ayudar al usuario a cómo usar el aplicativo.

Capítulo 3. Diseño de la aplicación

Una vez hemos definido la funcionalidad del sistema, en este capítulo, vamos a tratar aspectos de diseño.

3.1. Actores

Tal y como se ha mencionado en el capítulo anterior, tenemos cuatro actores principales. Estos actores son: Alumno, Padre, Profesor y Administrador. Cada actor puede realizar determinadas acciones. Esto quiere decir, que no toda la funcionalidad del sistema va a estar disponible para todos los usuarios. Para reflejar esta separación entre usuarios, hemos establecido la jerarquía que podemos contemplar en la Ilustración 3.1**.**

Un usuario Alumno es el que menos funcionalidades va a tener, ya que nuestro sistema se centra más en los padres y profesores. Un usuario Padre, además de poder realizar las mismas acciones que un usuario Alumno, va a tener funcionalidades propias de su perfil, como por ejemplo intercambiar mensajes con un profesor, solicitar tutoría, etc. Un usuario Profesor también va a tener funcionalidades propias de su perfil, como pueden ser habilitar tutorías, crear tareas, etc. Y por último, el usuario administrador es el que va a tener accesible toda la funcionalidad del sistema y con todos los privilegios.

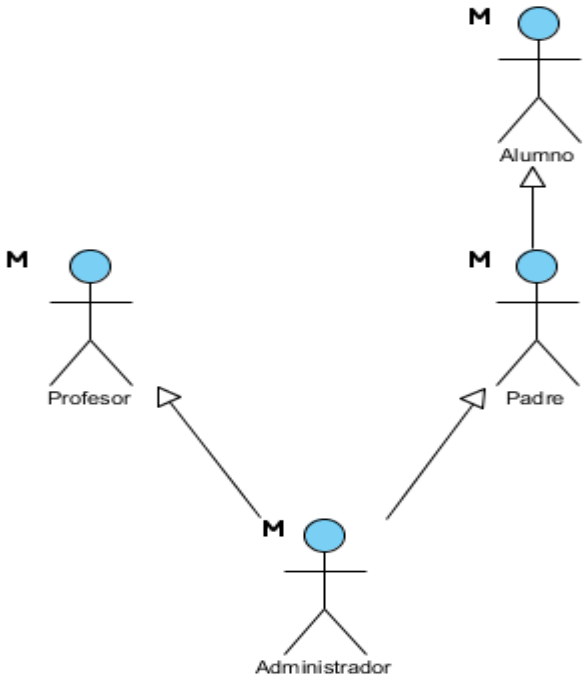


Ilustración 3.1

3.2. Diagramas de Casos de uso

En este apartado vamos a exponer los diferentes casos de uso que se dan en el sistema, asociados a los actores que los puedan realizar.

3.2.1. Módulo de comunicación padres-profesores

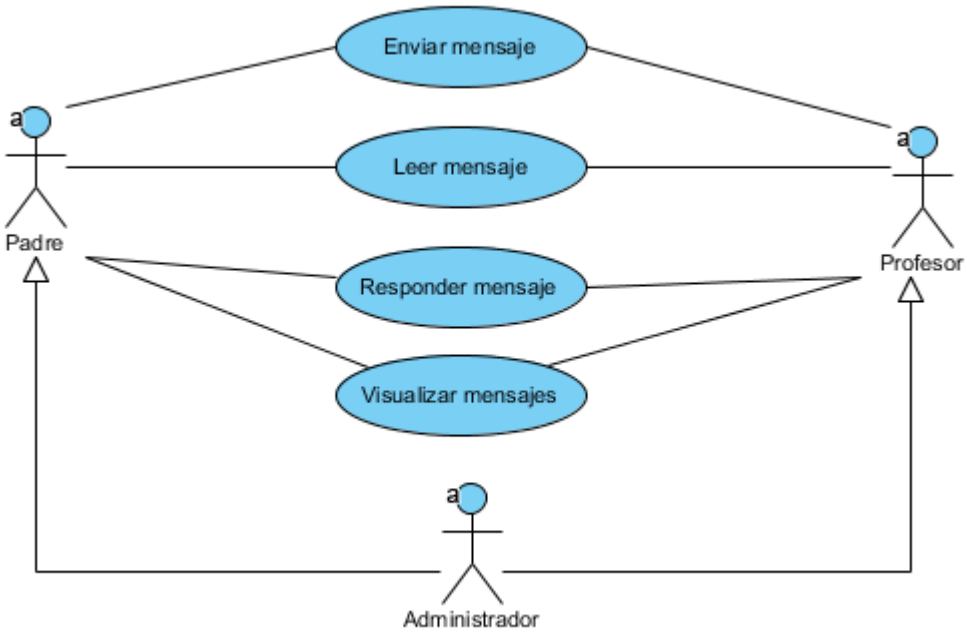


Ilustración 3.2. Módulo de comunicación padres-profesores

3.2.2. Módulo de tutorías Online

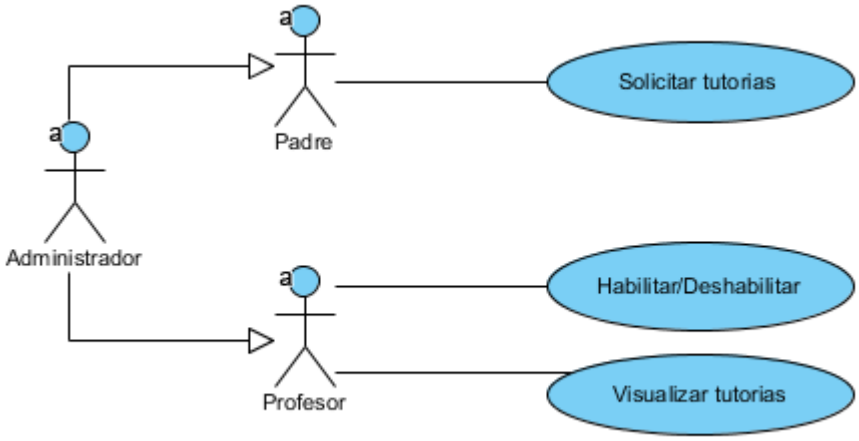


Ilustración 3.3. Módulo de tutorías Online.

3.2.3. Gestión de tareas

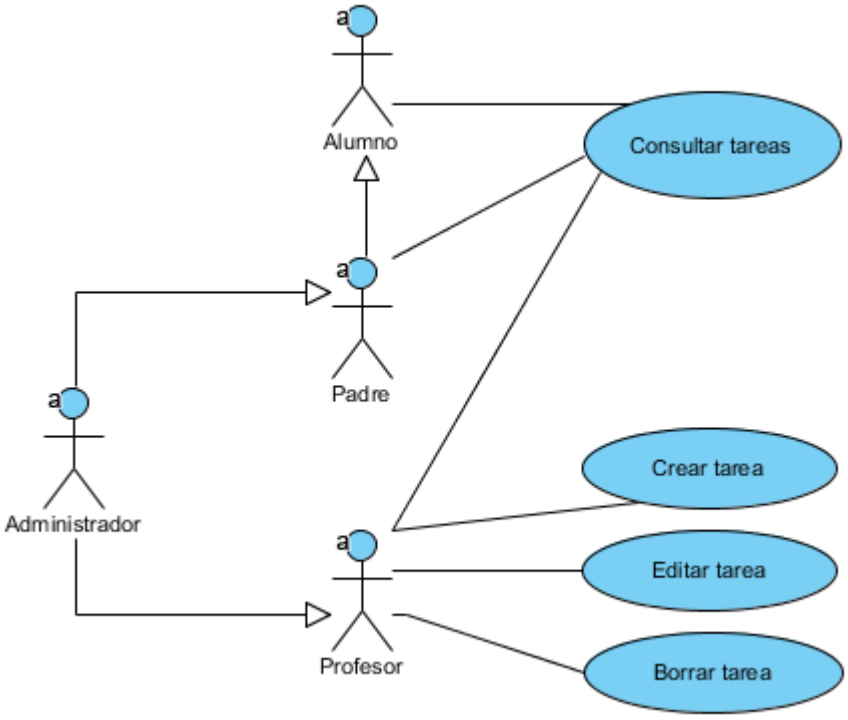


Ilustración 3.4. Gestión de tareas.

3.2.4. Gestión de cursos

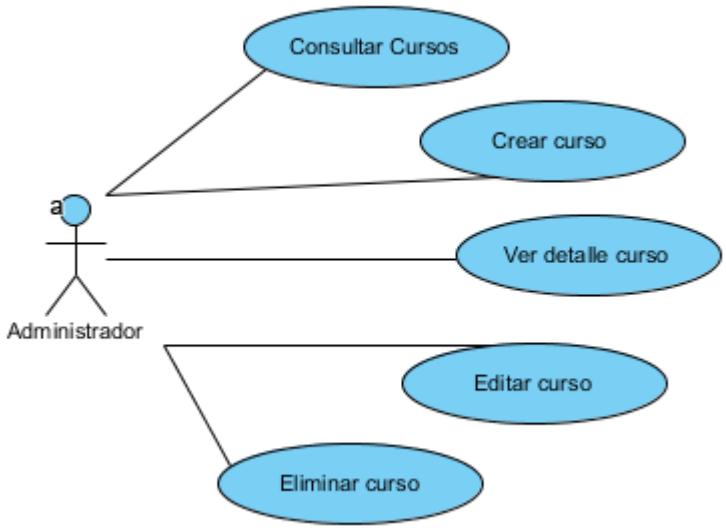


Ilustración 3.5. Gestión de cursos.

3.2.5. Gestión de grupos

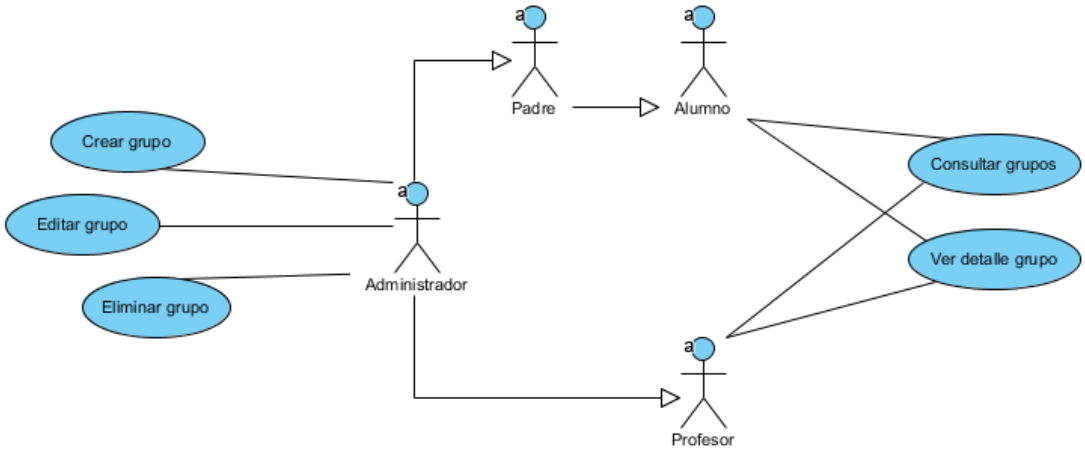


Ilustración 3.6. Gestión de grupos.

3.2.6. Gestión de asignaturas

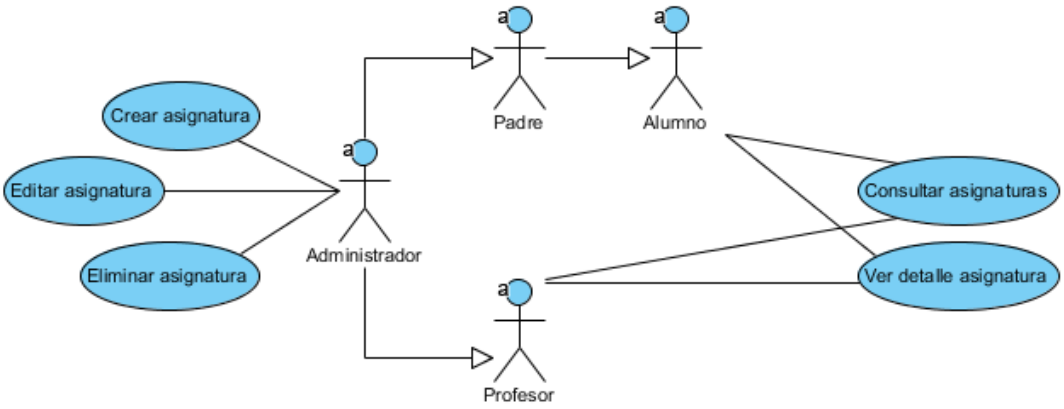


Ilustración 3.7. Gestión de asignaturas.

3.2.7. Gestión de usuarios

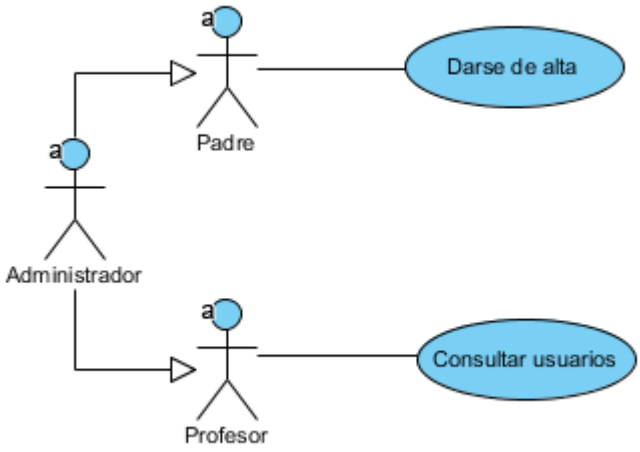
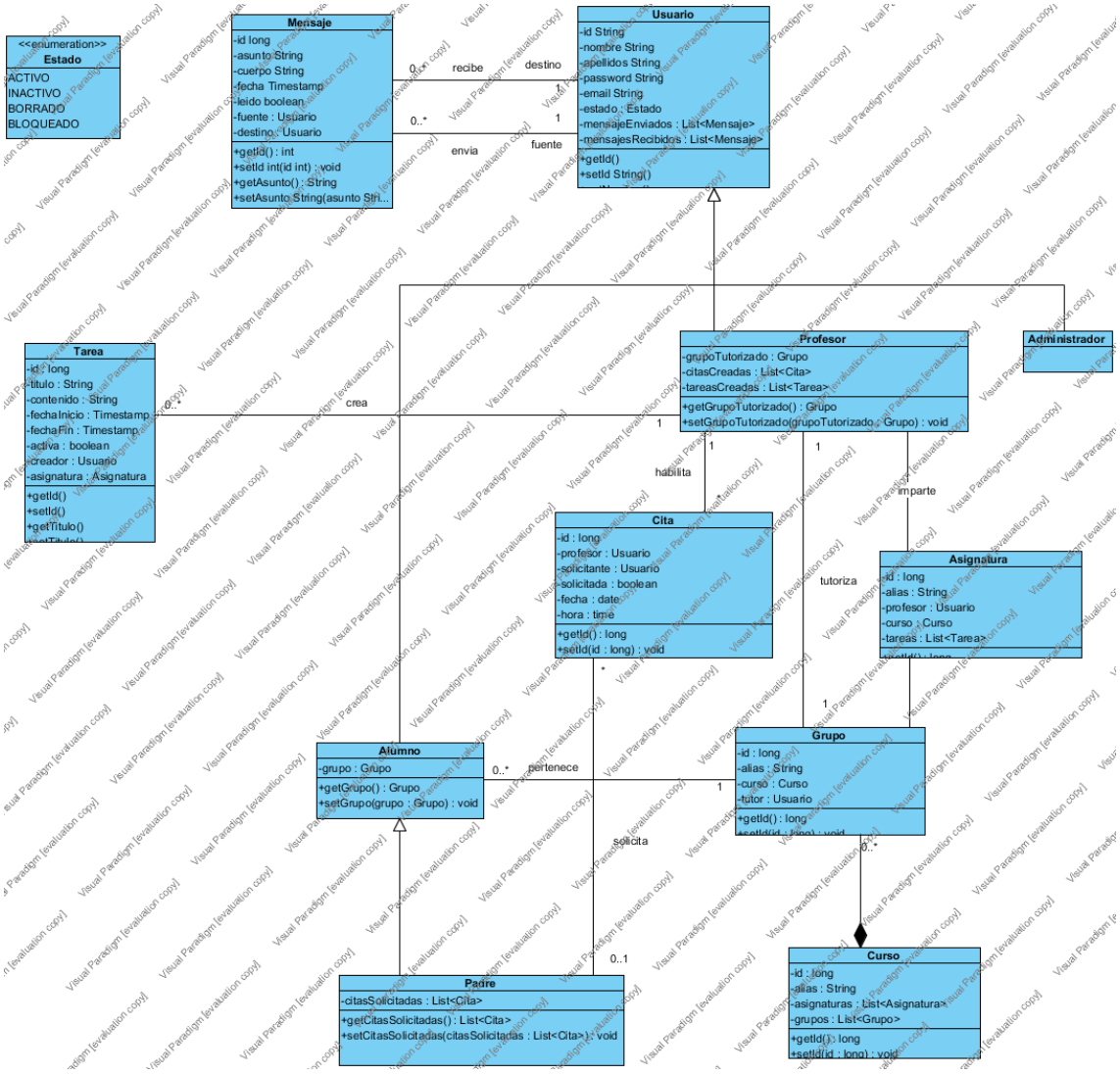


Ilustración 3.8. Gestión de usuarios.

3.3. Modelo conceptual

En este apartado mostramos el diagrama de clases que representa a modo conceptual el dominio de la aplicación. Se trata de un modelo abstracto diseñado con el objetivo de ayudar a comprender el contexto del problema.



3.4. Modelo relacional

La siguiente ilustración es el modelo relacional resultante tras haber analizado el modelo conceptual y adaptarlo a una base de datos.

Como podemos observar en ambos modelos, la relación de herencia representada en el modelo conceptual, tras estudiar las distintas alternativas para pasarla a nivel de base de datos, hemos optado por representarla mediante una única tabla, esto se explica mejor en el apartado 4.6.1 La herencia.

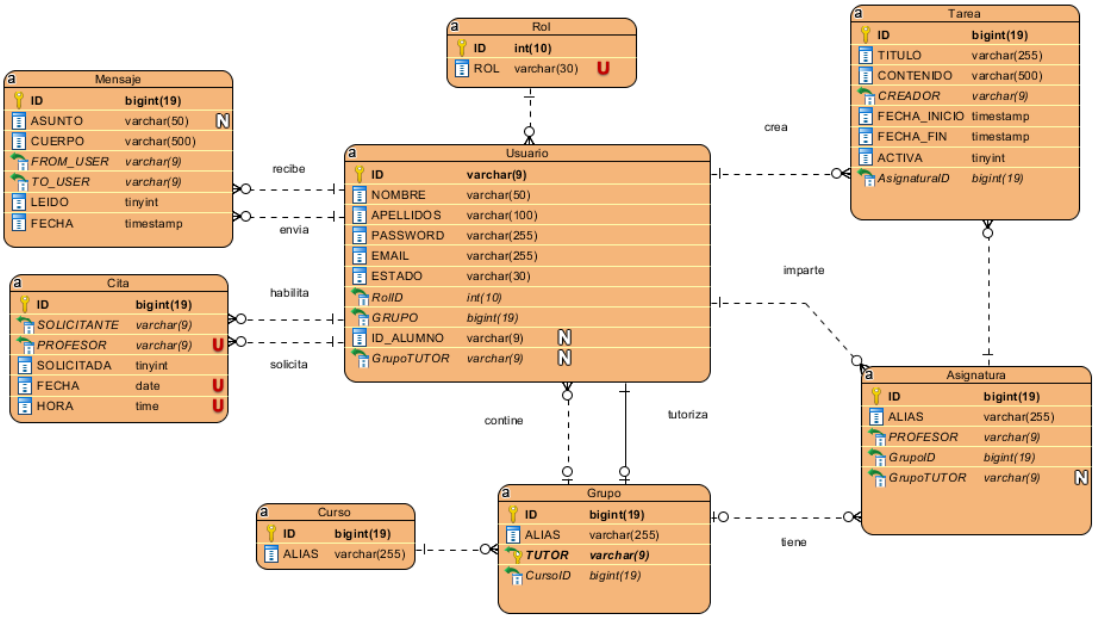


Ilustración 3.9. Diagrama relacional.

Capítulo 4. Implementación

El objetivo de este capítulo es destacar los aspectos más importantes de la implementación de la aplicación, la metodología que hemos usado, las fases por las que hemos pasado y los pasos seguidos para poder llevar a cabo todo el proceso.

4.1. Metodología de desarrollo

Para el desarrollo de nuestro sistema se ha escogido una metodología iterativa e incremental. Este tipo de metodologías se caracterizan por planificar los proyectos en bloques temporales más conocidos como iteraciones,. En cada iteración se evoluciona el producto a partir de los resultados completados en iteraciones anteriores, añadiendo nuevos requisitos, o mejorando los que ya fueron completados, de manera que al final de cada iteración se obtiene una versión parcial del producto final.

Las iteraciones suelen ser de una duración corta, aunque esto también depende de la naturaleza de cada proyecto. En cada iteración se repite un proceso de trabajo similar para alcanzar los objetivos marcados. En nuestro caso, este proceso lo hemos dividido en las siguientes tareas:

- Seleccionar los requisitos que implementaríamos durante la iteración.
- Transformar los requisitos seleccionados en casos de uso.
- Desarrollar los casos de usos.
- Crear casos de prueba correspondientes a los casos de uso desarrollados.
- Actualizar el modelo relacional.
- Implementar.
- Validar los casos de prueba.

En los siguientes apartados, vamos a exponer qué casos de uso se han desarrollado en cada iteración, y detallar los aspectos más importantes de cada uno de ellos. La especificación de cada caso de uso se encuentra en el Anexo 8.1 de este documento.

4.2. Iteración cero

Es muy frecuente en las metodologías iterativas el uso de la llamada iteración cero, cuyo objetivo son los preparativos previos a comenzar el desarrollo.

En nuestro caso hemos aprovechado esta iteración para realizar diversas tareas que detallamos a continuación.

4.2.1. Estudio inicial

En primer lugar hemos realizado un estudio inicial del sistema que nos ha ayudado a tener una idea más clara de la aplicación y su funcionalidad. Este estudio incluye identificar los requisitos que pensamos que el sistema debería incluir, esbozar algunas maquetas a partir de las cuales podríamos empezar el desarrollo, así como escoger las herramientas y tecnologías que se van a utilizar a lo largo del desarrollo.

4.2.2. Búsqueda de información

Puesto que uno de los objetivos del trabajo ha sido el aprendizaje y la familiarización con las tecnologías seleccionadas, hemos dedicado parte del tiempo de esta fase a leer y buscar información acerca de las tecnologías nuevas que vamos a utilizar. Aunque esto también se verá repetido en todas las iteraciones.

4.2.3. Instalar las herramientas

Una vez conocidas las herramientas que vamos a usar hemos procedido a su instalación y configuración.

4.2.4. Creación del repositorio

Tal y como hemos mencionado en el primer capítulo de esta memoria, vamos a contar con Subversion como herramienta para el control de versiones. En este paso vamos a proceder a la creación del repositorio y su estructura básica tal y como podemos observar en la Ilustración 4.1.

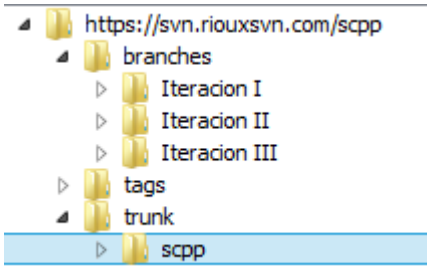


Ilustración 4.1

4.2.5. Creación de la estructura del proyecto

Maven es una excelente herramienta a la hora de construir proyectos software, ya que se encarga de realizar diferentes tareas que tendríamos que hacer manualmente o automatizarlas de alguna forma si no existiese esta herramienta.

A la hora de crear un proyecto Maven, esta herramienta nos da la posibilidad de elegir el tipo de proyecto que estamos creando. En nuestro caso es una aplicación web, esto es porque Maven tiene estructuras específicas para cada tipo de proyecto. Estas estructuras no pueden ser alteradas ya que eso podría alterar el correcto funcionamiento de la herramienta.

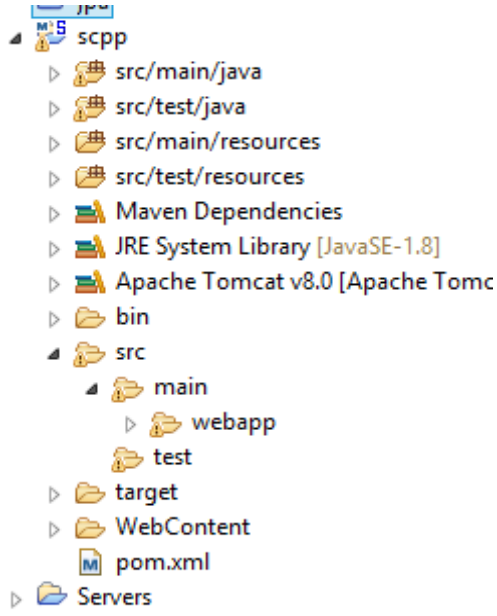


Ilustración 4.2

Para nuestro caso Maven ha creado la estructura que vemos en la Ilustración 4.2.

- **src/main/java**: un directorio para almacenar el código fuente de la aplicación.
- **src/test/java**: un directorio para almacenar los ficheros de tests como por ejemplo los test unitarios.
- **src/main/resources**: un directorio para almacenar los distintos recursos que nos puedan hacer falta (ficheros xml, ficheros de propiedades, etc.)
- **src/test/resources**: un directorio para los recursos que usa el código fuente de los tests.
- Además, al tratarse de una aplicación web, disponemos de un directorio **webapp**, donde alojaríamos las vistas y los recursos necesarios para estas, así como la configuración de la parte del frontal de la aplicación.

Una componente muy importante de cualquier proyecto Maven, es el POM (Project Object Model). Es un fichero xml que contiene información acerca de las dependencias, plugins utilizados, versiones, etc. Esto facilita mucho la labor de incluir librerías de terceros que vayamos a necesitar. En la Ilustración 4.3, podemos observar cómo hemos podido añadir fácilmente algunas de las librerías de Spring que vamos a necesitar.

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>${org.springframework-version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>${org.springframework-version}</version>
</dependency>
<dependency>
  <groupId>org.springframework.data</groupId>
  <artifactId>spring-data-jpa</artifactId>
  <version>1.10.2.RELEASE</version>
</dependency>
```

Ilustración 4.3

Además de la estructura básica que nos impone Maven, hemos estructurado el proyecto en paquetes tal y como se puede ver en la Ilustración 4.4.

En cada paquete guardaremos los siguientes tipos de ficheros:

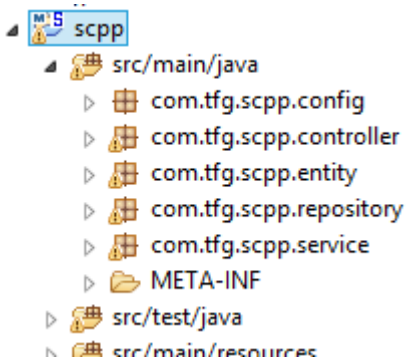


Ilustración 4.4

- En **com.tfg.scpp.config** guardaremos todas las clases relativas a la configuración del proyecto.
- En **com.tfg.scpp.controller** guardaremos los controladores.
- En **com.tfg.scpp.entity** guardaremos las entidades mapeadas desde la base de datos.
- En **com.tfg.scpp.repositorio** guardaremos las clases repositorio.
- En **com.tfg.scpp.service** guardaremos las clases service.

4.2.6. Creación de la base de datos

Otro punto que hemos considerado oportuno realizar en esta iteración es la creación de la base de datos pero solo lo que es el esquema, ya que las entidades y las tablas se irán creando de forma iterativa cuando corresponda.

4.2.7. Configuración del proyecto

Antes de empezar el desarrollo también es necesaria la configuración del proyecto, su conexión a la base de datos anteriormente creada, configurar el Log4J, etc.

Al estar utilizando la versión 4 de Spring Framework, toda la configuración relativa a este framework, es posible hacerla en clases java, lo cual nos hace

evitar el manejo de ficheros xml, que es como se hacía en la versión 2 y anteriores.

4.3. Iteración I

Una vez acabados los preparativos previos, empezamos la primera iteración.

Para esta iteración hemos seleccionado los casos de uso de la tabla de a continuación. Esta selección la realizamos en función de lo que más valor aporta a nuestro sistema.

Requisitos funcionales	Casos de uso derivados
RF01 – Módulo de comunicación padres-profesores	**CU01 – Módulo de comunicación padres profesores**
RF02 – Módulo de tutorías Online	**CU02 – Módulo de tutorías Online**
RF03 – Módulo de tareas	**CU03 – Gestión de tareas**

Puesto que el flujo de desarrollo es similar en todos los casos solo vamos a explicarlo para un caso, el CU01.

4.3.1. El acceso a datos

Para el módulo de comunicación padres-profesores, vamos necesitar crear 2 tablas en base de datos, las que corresponden en el modelo relacional a Mensaje y Usuario. Una vez creadas, vamos a proceder a su mapeo usando el estándar JPA.

Como podemos observar en las entidades obtenidas (Ver Anexo 8.2.1 – Entidad Usuario, Anexo 8.2.2 Entidad Mensaje), JPA también permite el uso de anotaciones, así que podemos indicar fácilmente especificaciones como pueden ser qué campo es la clave primaria (`@Id`), a qué columna de la tabla corresponde un determinado atributo (`@Column`), si un atributo no forma parte de la tabla física (`@Transient`), etc.

Puesto que la aplicación hará uso de Spring Data Repository Abstraction, el siguiente paso será crear un repositorio que enlace las entidades con la base de datos. Este paradigma se basa en interfaces parametrizadas que capturan el tipo del objeto que se va a gestionar junto al tipo del id del objeto. Las implementaciones de estos repositorios pueden incluir métodos de tipo CRUD (Create, Read, Update, Delete), como lo hace la interfaz CrudRepository<T, ID>, o incluso otro tipo de consultas más personalizadas. (Ver Ilustración 4.5)

Para el manejo de los objetos de la entidad `Mensaje` hemos creado una interfaz `MensajeRepository` que extiende de `CrudRepository` que es una interfaz

parametrizada. Los parámetros son la entidad y el tipo de dato de la clave primaria de la entidad en cuestión. Con sólo eso podremos realizar las operaciones habituales de crear, editar, eliminar, etc. Si queremos realizar consultas más específicas, hay distintas formas de hacerlo, una de ellas es escribiendo la consulta en la anotación `@Query` tal y como se ve en la Ilustración 4.5. Si la consulta en cuestión modifica el estado de la base de datos, tenemos que añadir la etiqueta `@Modifying`.

```
public interface MensajeRepository extends CrudRepository<Mensaje, Long> {

    @Query("select m from Mensaje m where m.destino = :to")
    List<Mensaje> findByOwner(@Param("to") Usuario to);

    @Query("select m from Mensaje m where m.fuente = :from")
    List<Mensaje> findBySender(@Param("from") Usuario from);

    @Query("select m from Mensaje m where m.destino = :user and leido = false")
    List<Mensaje> getMensajeNoLeidos(@Param("user") Usuario user);
}
```

Ilustración 4.5

Una vez definidas las entidades y su respectivo repositorio, es el momento de crear la capa de service, que son las que se encargan de realizar las acciones solicitadas por los controladores, procesando la información necesaria y hacerla llegar a los repositorios. Son clases anotadas con la anotación `@Service` en las que podemos declarar una interfaz repositorio y acceder a ella. (Ver Ilustración 4.6)

```
@Service
public class MensajeServiceImpl implements MensajeService {

    @Autowired
    private MensajeRepository mensajeRepo;

    @Override
    public void enviarMensaje(Mensaje mensaje) {
        this.mensajeRepo.save(mensaje);
    }
```

Ilustración 4.6

4.3.2. Spring MVC

El siguiente paso es crear la parte controladora correspondiente a este módulo. El controlador es el encargado de preparar el modelo y redirigirnos a la vista adecuada. Se trata de clases anotadas con la anotación `@Controller`, los métodos de esta clase suelen llevar la anotación `@RequestMapping(value = "/mensajes/nuevo", method = RequestMethod.GET)` con value estamos indicando la ruta de la cual procede la petición, y en method el tipo de petición

HTTP (GET, POST, PUT, etc.). En la Ilustración 4.7 podemos ver un ejemplo claro de la utilización de un controlador.

Es necesario declarar `mensajeService` y `usuarioService` para poder acceder a las funcionalidades que encapsulan.

La anotación `@Autowired` es para inyectar la instancia.

El objeto `model` es una interfaz que nos permite pasar los datos que vayamos a necesitar manejar en la vista.

```
 */
@Controller
public class MensajesController {

    private static final Logger logger = LoggerFactory
            .getLogger(MensajesController.class);

    @Autowired
    private MensajeService mensajeService;
    @Autowired
    private UsuarioService usuarioService;

    @RequestMapping(value = "/mensajes/nuevo", method = RequestMethod.GET)
    public String nuevoMensaje(Model model, HttpSession session) {
        logger.debug("mensaje nulo, nuevo mensaje");
        Usuario user = (Usuario) session.getAttribute("user");
        List<Usuario> destinatarios = obtenerDestinatariosRol(user);
        model.addAttribute("dest", destinatarios);
        Mensaje mensaje = new Mensaje();
        mensaje.setFuente(user);
        model.addAttribute("mensaje", mensaje);
        logger.info("Redirigiendo a la pantalla de nuevo mensaje");
        return "comunicacion/NuevoMensaje";
    }
```

Ilustración 4.7

Por ejemplo, en el caso de llamar a la vista de enviar un nuevo mensaje, necesitamos rellenar el desplegable de destinatarios y la fuente del mensaje (Ver Ilustración 4.8).

Ilustración 4.8

En cuanto a la parte de la vista, Spring nos da varias alternativas entre las cuales se encuentra Java Server Pages (JSP) que es la que hemos usado en combinación con JSTL para facilitar la implementación además de obtener un código mucho más limpio y fácil de mantener.

4.4. Iteración II

Una vez terminada y probada la versión resultante de la primera iteración es el momento de empezar la segunda. Basándonos en los resultados obtenidos, los requisitos que se van a desarrollar en esta iteración van a ser los siguientes:

RFN01 – Control de acceso

RFN02 – Funcionalidades visibles según el rol

RFN03 – Acceso simultáneo

4.4.1. Spring Security

Para la implementación del RFN01 y el RFN02, entra en juego Spring Security. Se trata de un subproyecto de Spring Framework que permite gestionar todos los aspectos de seguridad de nuestras aplicaciones.

Antes de nada, lo primero que hay que hacer es añadir las dependencias necesarias de Spring Security al POM de nuestro proyecto.

```
<!-- Spring Security -->
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-web</artifactId>
    <version>${org.spring-security-version}</version>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-config</artifactId>
    <version>${org.spring-security-version}</version>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-taglibs</artifactId>
    <version>${org.spring-security-version}</version>
</dependency>
```

Ilustración 4.9

Tal y como podemos contemplar en la anterior captura, la primera dependencia es para el control de la seguridad en la parte web, la segunda para temas de configuración que veremos a continuación, y la tercera es para manejar las taglibs de Spring Security de manera más fácil. Esto también lo veremos más adelante.

Una vez añadidas las dependencias hay que configurar el proyecto para que trabaje con este nuevo módulo. Para ello creamos una clase donde incluiremos todo lo relativo a la configuración de la seguridad.

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    @Qualifier("customUserDetailsService")
    UserDetailsService userDetailsService;
```

Ilustración 4.10

Con la anotación `@Configuration` estamos indicando que se trata de una clase de configuración y con `@EnableWebSecurity` estamos activando el uso de `spring-web-security.`

`UserDetailsService` es una interfaz que nos proporciona Spring Security a modo de DAO. Tenemos que implementarla para poder acceder a los usuarios que tenemos almacenados en nuestra base de datos. (Ver Ilustración 4.11)

```java
@Service("customUserDetailsService")
public class CustomUserDetailsService implements UserDetailsService{

    private static final Logger logger = LoggerFactory.getLogger(CustomUserDetailsService.class);

    @Autowired
    private UsuarioService usuarioService;

    @Transactional(readOnly=true)
    public UserDetails loadUserByUsername(String id) throws UsernameNotFoundException {
        logger.debug("id: " + id);
        Usuario user = usuarioService.getUserById(id);
        if (user == null) {
            logger.debug("user not found");
            throw new UsernameNotFoundException("User not found");
        }
        logger.debug("user: " + user);
        User u = new User(user.getId(), user.getPassword(), user.getEstado()
                .equals(Estado.ACTIVO), true, true, true,getGrantedAuthorities(user));

        return u;
    }
```

Ilustración 4.11

Con este método comprobamos si un determinado usuario existe en base de datos y si es activo y cargamos sus permisos. Estos permisos se pueden establecer de diferentes formas, según sea conveniente.

- En el caso de que quisiéramos restringir el acceso a una url completa a un determinado rol, lo haríamos en la clase de configuración, tal y como podemos ver en la Ilustración 4.12:

```java
@Override
protected void configure(HttpSecurity http) throws Exception {
  http.authorizeRequests()
    .antMatchers("/home").permitAll()
    .antMatchers("/admin/**").access("hasRole('ADMIN')")
    .antMatchers("/mensajes/**").access("hasAnyRole('ADMIN', 'PROFESOR', 'PADRE')")
    .antMatchers("/tareas/mistareas").access("hasAnyRole('ADMIN', 'PROFESOR', 'PADRE', 'ALUMNO'
```

Ilustración 4.12

  En la primera regla estamos dejando acceso a todo el mundo a la ruta /home, sin embargo, las rutas que tengan el patrón /admin/** sólo serán accesibles por un usuario con rol ADMIN. La tercera regla permite el acceso a la ruta /mensajes/** a usuarios ADMIN, PROFESOR, Y PADRE.

- Si queremos controlar que determinadas partes de una página sean visibles solo para un determinado rol, tenemos que usar spring-security-taglib que hemos añadido en las dependencias. Esto lo hemos usado por ejemplo para la barra de navegación, ya que no todos los usuarios van tener la misma visibilidad de las funcionalidades. Esto lo podemos ver mejor en la Ilustración 4.13:

```
<sec:authorize access="hasRole('ADMIN')">
<ul class="nav" id="side-menu">
        <li><a href="/scpp/principal"><i class="fa fa-home"></i>
                Inicio</a></li>
        <li><a href="/scpp/usuarios/${sessionScope.user.id}"><i cla
                Perfil de usuario</a></li>
        <li><a href="#"><i class="fa fa-exchange"></i>
                Mensajes<span class="fa arrow"></span></a>
```

Ilustración 4.13

Con el `tag <sec:authorize>` estamos indicando que todo el contenido que encierra, será visible únicamente por los perfiles indicados en el atributo `access`.

4.5. Iteración III

Para esta iteración hemos seleccionado los tres requisitos que listamos a continuación. Estos requisitos han sido necesarios para relacionar mejor las funcionalidades anteriormente desarrolladas.

Requisitos funcionales	Casos de uso derivados
RF04 – Gestión de cursos	**CU04 – Gestión de cursos**
RF06 – Gestión de grupos	**CU06 – Gestión de grupos**
RF07 – Gestión de asignaturas	**CU07 – Gestión de asignaturas**

4.5.1. Validación de datos

Estos tres requisitos son muy similares entre sí, ya que los tres son operaciones CRUD. Además, no presentan ninguna particularidad que no haya sido explicada anteriormente, así que aprovechamos este apartado para explicar cómo se ha realizado el tema de la validación de datos aprovechando uno de ellos.

Cuando disponemos de formularios que han de ser rellenados por el usuario, es importante realizar la validación de los datos introducidos por el usuario antes de proceder a su manipulación, ya que estos podrían no ser los adecuados y producirían problemas innecesarios. Para ello, vamos a necesitar las dos siguientes librerías:

```
<dependency>
    <groupId>javax.validation</groupId>
    <artifactId>validation-api</artifactId>
    <version>1.1.0.Final</version>
</dependency>
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>5.1.0.Final</version>
</dependency>
```

Ilustración 4.14

Supongamos el caso de gestión de cursos. Como podemos ver en el modelo relacional, la entidad curso solo dispone de dos atributos (id y alias). Por tanto, a la hora de mapear la tabla obtenemos la siguiente clase con los respectivos getters y setters de las propiedades. (La lista grupos surge de la relación 1:N existente entre Curso y Grupo)

```
@Entity
@NamedQuery(name="Curso.findAll", query="SELECT c FROM Curso c")
public class Curso implements Serializable {
    private static final long serialVersionUID = 1L;
    private Long id;
    private String alias;
    private List<Grupo> grupos;

    public Curso() {
    }
```

Ilustración 4.15

El formulario correspondiente es el siguiente:

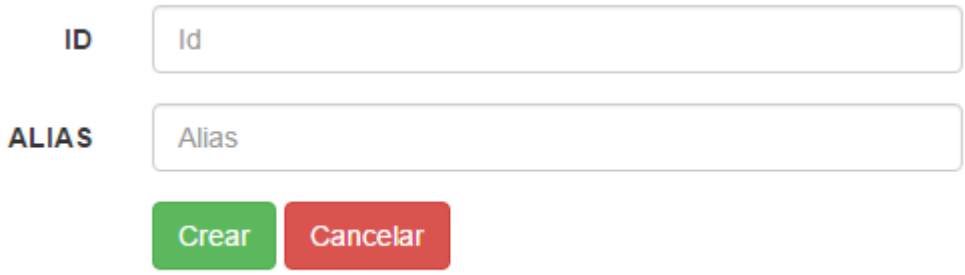


Ilustración 4.16

Y necesitamos especificar que el id no puede ser nulo, y el alias no puede ser vacío. Esto lo haremos haciendo uso de las anotaciones de las librerías que hemos añadido previamente.

```
@Id
@NotNull(message="Campo obligatorio",
@Valid
public Long getId() {
    return this.id;
}
```

Ilustración 4.17

```
@NotEmpty(message="Campo obligatorio",
@Valid
public String getAlias() {
    return this.alias;
}
```

Ilustración 4.18

La anotación `@Valid` obliga a que se valide el campo en cuestión cumpliendo con la anotación que tiene por encima.

En la jsp, necesitamos indicar que los campos del formulario necesitan ser validados. Esto lo indicamos con un tag adicional (<form:errors>), que nos mostrará el mensaje en caso de error.

```
<form:input class="form-control" path="id" placeholder="Id"/>
<form:errors path="id" cssClass="text-danger"/>
```

Ilustración 4.19

En el controlador, una vez recogidos los datos del formulario, indicamos que la entidad `Curso` recogida ha de ser validada. Esto lo indicamos con la anotación `@Valid`. Con el `BindingResult` comprobamos si se han recogido errores. En caso afirmativo, volvemos a recargar la página del formulario, y veremos los errores que se han dado.

```
@RequestMapping(value = "cursos/nuevo", method = RequestMethod.POST)
public String nuevaCurso(@ModelAttribute("curso") @Valid Curso curso,
        BindingResult result) throws ParseException{
    logger.info("Curso: " + curso.toString());
    if(result.hasErrors()){
        return "cursos/nuevoCurso";
    }
    cursoService.crearCurso(curso);
    return "redirect:/cursos/miscursos";
}
```

Ilustración 4.20

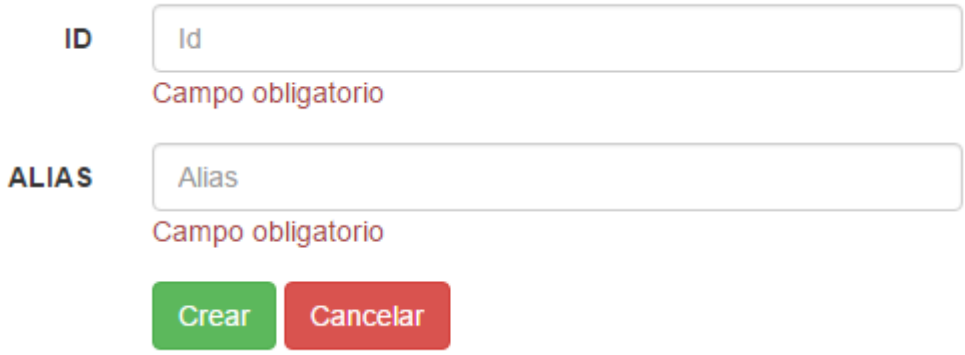


Ilustración 4.21

4.6. Iteración IV

Puesto que el objetivo del sistema no es llevar a cabo una gestión completa de los usuarios, hemos dejado el **RF08 – Gestión de usuarios** para esta cuarta iteración.

Requisitos funcionales	Casos de uso derivados
RF08 – Gestión de usuarios	**CU08 – Gestión de usuarios**

4.6.1. La herencia

Uno de los aspectos importantes de este módulo es la gestión de los distintos perfiles que pueden existir en el sistema. Dado que todos los usuarios, tengan el perfil que tengan, comparten bastantes propiedades pero tienen otras diferentes entre ellos, es evidente que lo más conveniente es implementar este caso mediante la herencia. Un usuario general con los atributos comunes y luego clases específicas para cada tipo de perfil, con los atributos necesarios (Ver diagrama conceptual).

El caso es que hay diferentes formas de llevar esta implementación a nivel de base de datos. Cada una con sus pros y sus contras.

- Utilizar una tabla única para todas las clases, dejando vacíos los campos que no son propios de la clase de la que se trata. Esto puede producir una gran cantidad de espacio reservado innecesario en muchos casos, pero es viable cuando los atributos de las clases prácticamente no varían. Se debe utilizar un discriminador para distinguir la clase de la que se trata en cada registro. El discriminador almacena el nombre de la clase para poder generar un objeto de la clase adecuada cuando se requiera.
- Utilizar una tabla para cada clase. En este caso, el espacio reservado siempre será adecuado y se puede distinguir la clase sin necesidad de un discriminador. Por otra parte, puede complicar peticiones como búsquedas conjuntas.
- Utilizar una tabla única para datos comunes y una tabla específica de cada clase para el resto. Es el caso en el que los atributos de la clase padre se comparten y los datos específicos en distintas tablas. También requiere discriminador.

Entre las opciones hemos optado por la primera, ya que aunque existen atributos que varían, no son muchos, además de facilitar las consultas comunes. Así que en la clase madre, Usuario, tenemos que indicar la columna que ejercerá de discriminador. (Ver Ilustración 4.24)

```
 */
@Entity
@DiscriminatorColumn(name="ROL")
@NamedQuery(name="Usuario.findAll", query="SELECT u FROM Usuario u")
public class Usuario implements Serializable {
    private static final long serialVersionUID = 1L;
    private String id;
    private String apellidos;
    private String email;
    private Estado estado;
    private String nombre;
    private String password;
    private String fullName;
```

Ilustración 4.22

En cada clase hija se indicará el valor de ese discriminador. (Ver Ilustración 4.25)

```
@Entity
@DiscriminatorValue("2")
@NamedQuery(name="Profesor.findAll", query="SELECT p FROM Profesor p")
public class Profesor extends Usuario implements Serializable {
    private static final long serialVersionUID = 1L;
    private Long grupoTutorizado;
    private List<Asignatura> asignaturasImpartidas;
    private List<Tarea> tareas;
    private List<Cita> citasProfesor;

    public Profesor() {
        super();
    }

    public Profesor(Usuario usuario){
        super(usuario);
    }
```

Ilustración 4.23

4.6.2. Cifrado de claves

Otro aspecto importante que se manifiesta a la hora de **CU08.02 – Dar de alta usuario padre**, es el **RFN05 – Cifrado de claves**. Una vez más, vamos a hacer uso de Spring Security, ya que nos proporciona distintos algoritmos de cifrado, sin tener que preocuparnos nosotros por ello.

Lo primero, en el archivo de configuración de security creamos el bean del tipo `BCryptPasswordEncoder`, que es el codificador que vamos a usar. Este bean

se lo pasamos al `AuthenticationManagerBuilder` cuando se configura.

```
@Bean
public PasswordEncoder passwordEncoder(){
    return new BCryptPasswordEncoder();
}

@Autowired
@Override
protected void configure(AuthenticationManagerBuilder authManagerBuilder) throws Exception{
    authManagerBuilder.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder());
}
```

Ilustración 4.24

Cuando recogemos los datos introducidos por el usuario a la hora de darse de alta, lo único que tenemos que hacer es llamar al método encode() para codificar la contraseña y almacenarla en base de datos.

```
@Override
@Transactional(propagation=Propagation.REQUIRED, readOnly = false)
public void alta(Usuario u){
    String encodedPass = passwordEncoder.encode(u.getPassword());
    u.setPassword(encodedPass);
    usuarioRepo.save(u);
}
```

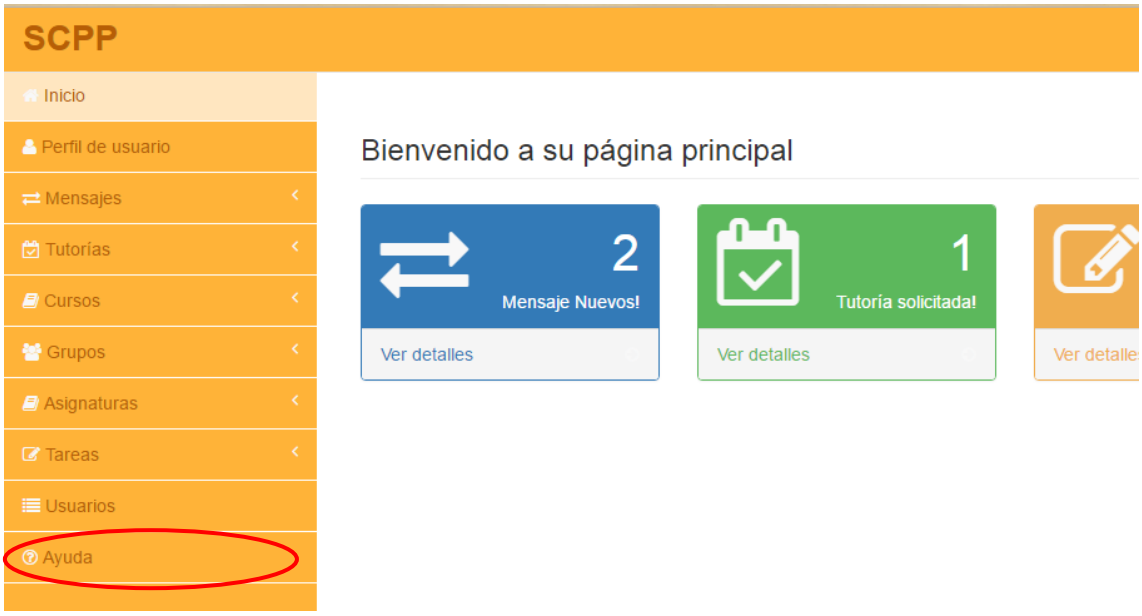
Ilustración 4.25

4.7. Iteración V

La ultima iteración la hemos dedicado a mejorar aspectos varios del sistema como por ejemplo el aspecto visual, la usabilidad, aspectos técnicos, así como elaborar un manual de usuario integrado en la propia aplicación.

RNF04 – Usabilidad

RNF06 – Manual de usuario


SCPP
Inicio
Perfil de usuario
Mensajes
Tutorías
Cursos
Grupos
Asignaturas
Tareas
Usuarios
Ayuda
Bienvenido a su página principal
2
Mensaje Nuevos!
Ver detalles
1
Tutoría solicitada!
Ver detalles

Capítulo 5. Aspecto visual

En este capítulo, abordaremos todas las cuestiones relativas al aspecto visual de la aplicación, framework, herramientas, estilo, etc.

Teniendo en cuenta las características del sistema así como los usuarios finales, pensamos que es importante contar con una interfaz gráfica agradable, intuitiva y fácil de usar, para contribuir a tener una buena experiencia de usuario.

5.1. Herramientas

Para conseguir el aspecto visual obtenido se han utilizado las siguientes herramientas:

- Bootstrap v3.3.7. Un framework que tiene como objetivo facilitar el diseño web. Además de ofrecer una gran variedad de componentes listos para ser usados, permite crear de forma sencilla webs de diseño adaptable.
- CSS. Hojas de estilo para centralizar los aspectos visuales y mantenerlos separados del contenido.
- Javascript.
- JQuery.
- Fontawesome. Una librería que cuenta con una gran variedad de iconos.

5.2. Componentes principales

Para diseñar la interfaz gráfica de usuario, hemos considerado un diseño minimalista y una paleta de colores homogénea. El diseño lo forman tres componentes principales:

- **Una barra superior**. Su única función es permitir al usuario desconectarse del sistema en cualquier momento, además de ofrecer un link directo a su página de perfil.

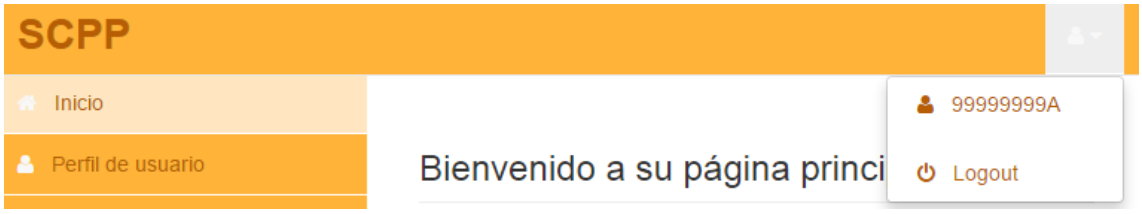


Ilustración 5.1

- **Una barra lateral**. Con todas las funcionalidades del sistema permitidas al usuario conectado.

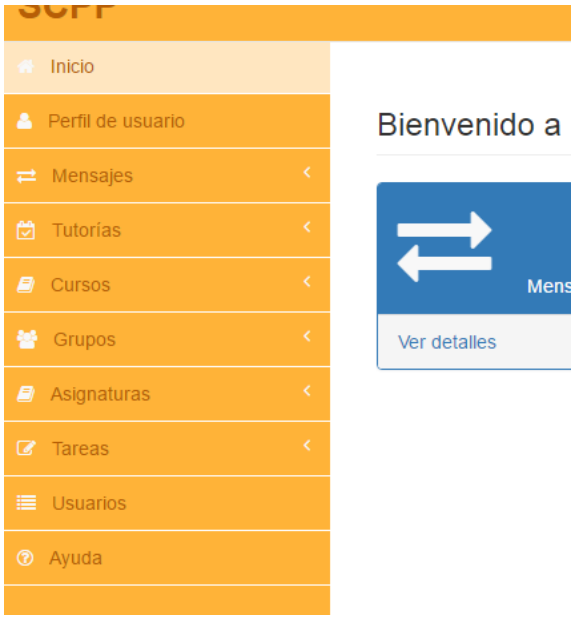


Ilustración 5.2

- **Un contenedor central**. Dónde colocamos el contenido de cada funcionalidad. (Ver siguientes ejemplos)

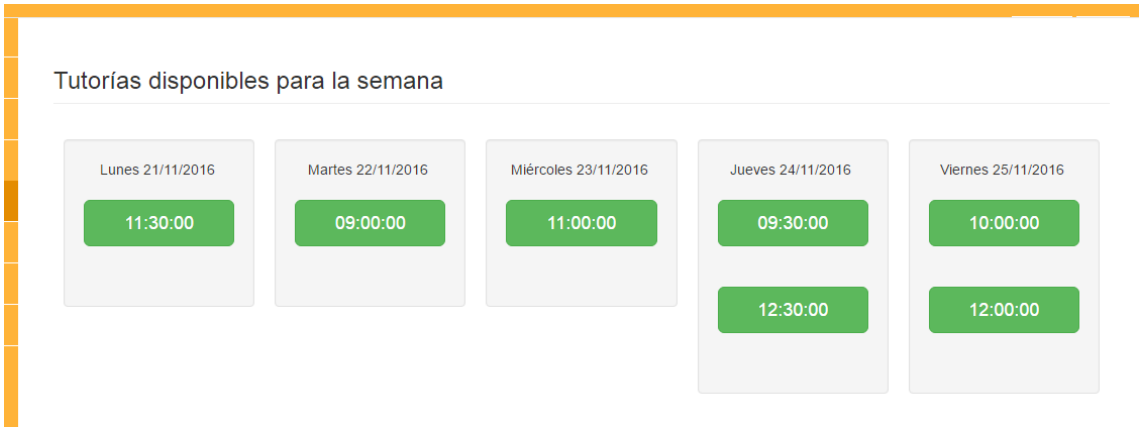


Ilustración 5.3

Ilustración 5.4

5.3. Aspectos tenidos en cuenta

A la hora de implementar una interfaz gráfica de usuario, hay muchos aspectos a tener en cuenta. Para nuestro caso, el principio que hemos tenido en mente en todo momento, es presentar al usuario una interfaz fácil de usar que apenas requiera aprendizaje. Lo que en el mundo de UX (User Experience) se denomina Usabilidad. Para conseguir este objetivo hemos considerado los siguientes detalles:

- Presentar un diseño minimalista sin recargar con información innecesaria.
- Toda la funcionalidad disponible visible en todo momento en la barra lateral.
- Accesos rápidos (shortcuts) en la página de inicio. (Ver Ilustración 5.5)
- El usuario sabe en todo momento en qué sección se encuentra. (Ver Ilustración 5.6)
- Usar iconos descriptivos. (Ver Ilustración 5.2)
- Interacción con el usuario mediante distintos tipos de mensajes. (Ver Ilustración 5.7)

Ilustración 5.5

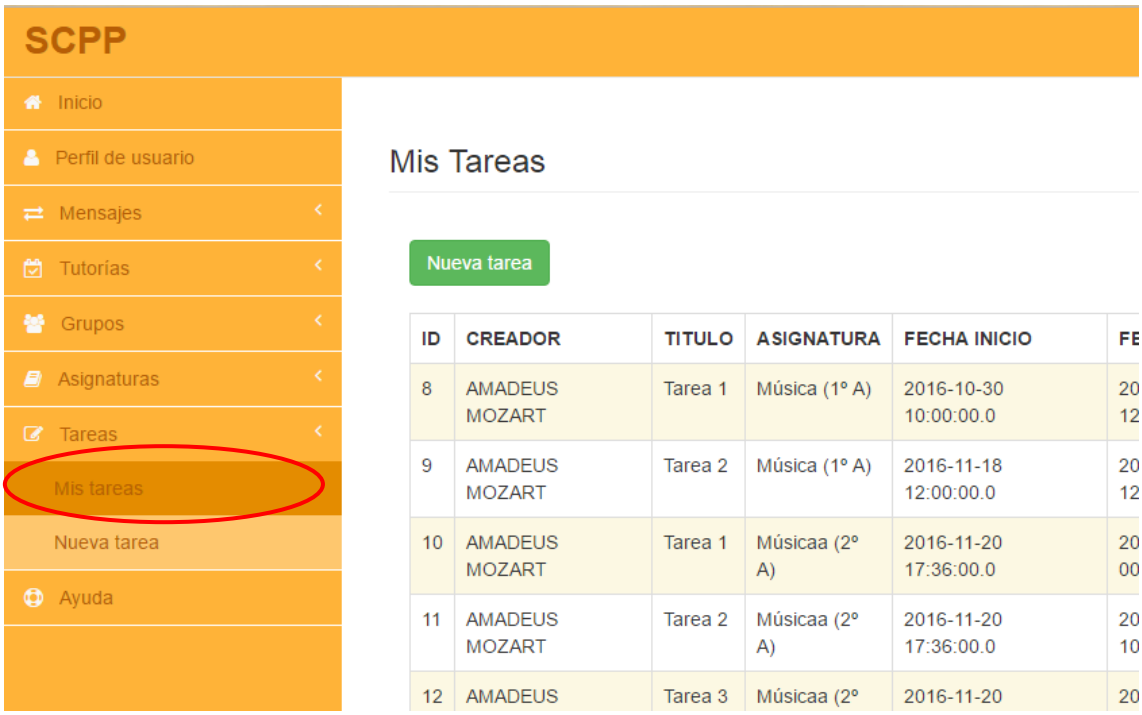


Ilustración 5.6

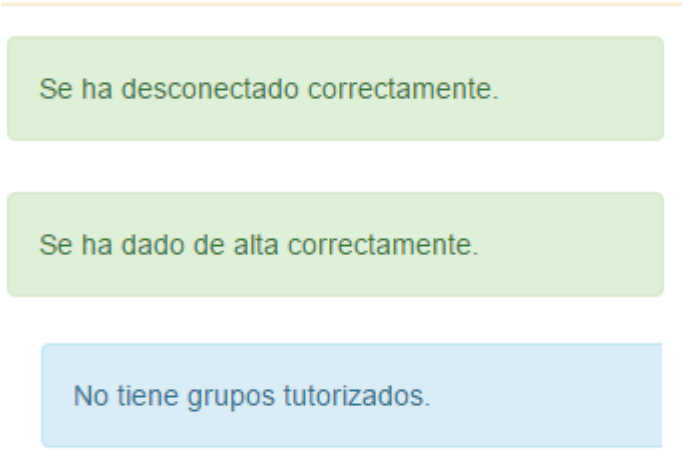


Ilustración 5.7

Capítulo 6. Pruebas

El proceso de pruebas y validación forma una parte fundamental en los procesos de desarrollo software. Pero por limitaciones de tiempo no se ha podido dedicar el tiempo suficiente a esta parte.

El tipo de pruebas que se han realizado, son pruebas de sistema. Se trata de pruebas que pretenden comprobar la corrección y completitud del sistema, teniendo como guía los requisitos y los casos de uso especificados. En el Anexo 8.3, se encuentran algunos de los casos de prueba que se han especificado para validar algunas de las funcionalidades.

La validación de los casos de prueba se ha realizado de forma manual.

Capítulo 7. Conclusiones y trabajo futuro

Llegados a este a apartado, finalizamos el proceso de trabajo fin de grado descrito en los apartados anteriores. Pero antes de concluirlo, en este apartado recopilamos todas las conclusiones obtenidas durante todo el proceso.

Además, incluimos una serie de propuestas que podrían implementarse en el futuro.

7.1. Tiempo consumido

En la siguiente tabla, podemos ver un resumen del tiempo consumido para llevar a cabo todo el proceso. Podemos ver que las primeras iteraciones en concreto la 0 y la I son las que más tiempo han llevado, debido al periodo de aprendizaje requerido. También podemos observar que la iteración III es la que menos tiempo ha necesitado, ya que no presentaba mucha complejidad respecto a otras, además, ya contábamos con cierta soltura adquirida de las iteraciones anteriores.

	Tareas realizadas	Número de horas
Iteración 0	Estudio Inicial	12
	Búsqueda de información	100
	Instalación de herramientas	6
	Creación del repositorio	1
	Creación de la base de datos	
	Configuración del proyecto	28
	Estructura del proyecto	2
Iteración I	CU01	56
	CU02	72
	CU03	28
Iteración II	Spring Security	12
	Control de la seguridad	25
	Interfaz gráfica	20
Iteración III	CU04	2
	CU06	4
	CU07	8

	Validación	8
Iteración IV	CU08	30
	Cifrado de claves	2
Iteración V	Mejorar aspectos varios	10
	Manual de usuario	12
	Memoria	60
	Total horas	498

7.2. Resultado obtenido

El primer objetivo de este trabajo, es intentar dar una solución al problema expuesto en el apartado 1.1 de este documento. Una vez finalizado todo el proceso, consideramos que se ha obtenido una solución, aunque mejorable, pero bastante aceptable para facilitar la comunicación entre los tutores académicos y los tutores legales de los alumnos, así como permitir a los padres llevar un seguimiento de una forma más fácil de la evolución de sus hijos.

7.3. Metodología utilizada

Tal y como se ha mencionado en apartados anteriores, para llevar a cabo el proyecto, se ha intentado seguir una metodología iterativa e incremental, una metodología conocida pero nunca utilizada, por lo que consideramos este trabajo el primer contacto con ella.

Es destacable mencionar, que ha sido todo un reto poner en práctica esta metodología, ´sobre todo a la hora de empezar, o en el momento de priorizar, o cuando en iteraciones avanzadas se requiere volver a tocar funcionalidades implementadas en iteraciones anteriores.

Si tuviéramos que corregir alguno de los aspectos relativos a la puesta en práctica de la metodología, creemos que el número de requisitos incluidos en cada iteración ha sido excesivo, sobretodo en la primera, descomponerla en tres iteraciones igual habría sido más acertado.

Pero a pesar de todo eso, el resultado final ha sido gratificante, ya que una vez arrancado el proceso y superadas las dos primeras iteraciones, el flujo de trabajo ha avanzado de una manera mucho más fluida, y el resultado de una iteración guiaba en gran medida el contenido de la siguiente.

7.4. Tecnologías utilizadas

El segundo objetivo principal del trabajo ha sido la familiarización con las tecnologías escogidas. Consideramos que se ha alcanzado dicho objetivo, ya

que se han adquirido los conocimientos suficientes como para conseguir el resultado obtenido, además, la experiencia ha sido, cuanto menos gratificante.

Puesto que la tecnología estrella del proyecto es Spring Framework, he aquí algunas conclusiones, fruto de la puesta en práctica de este framework:

El puntos más fuerte de este framework, desde mi punto de vista, es su modularidad. Esto hace que sea muy fácil de integrar con otras herramientas enfocadas, cada una en un área particular, como pueden ser la capa de la presentación, la persistencia, etc.

Otro aspecto destacable es que brinda muchas facilidades a la hora de escribir código, diseñar sistemas, etc. Proporcionando con ello una estructura sólida y consistente.

El punto negativo que le podría sacar, es que la curva de aprendizaje no es especialmente rápida, ya que se requiere invertir gran cantidad de tiempo para poder comprender y utilizar el framework correctamente.

7.5. Trabajo futuro

Debido al tiempo limitado disponible para la realización del trabajo, muchas funcionalidades importantes han quedado fuera de nuestro alcance. Por esta razón, las expondremos en este apartado como trabajo futuro, y pensamos que añadirían mucho valor para los usuarios finales del sistema. Entre estas funcionalidades mencionamos las siguientes:

- Calificaciones.
- Eventos.
- Faltas.
- Comunicación entre padres.
- Notificaciones.
- Etc.

Aunque el hecho de haber desarrollado el frontal usando Bootstrap Framework, hace que la aplicación sea adaptable a cualquier tipo de dispositivo, pensamos que sería de gran valor para el usuario final, disponer de una versión para dispositivos móviles.

Bibliografía

- Spring Web MVC: http://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html
- Spring Security: https://spring.io/guides/gs/securing-web/
- Spring Data JPA: http://docs.spring.io/springdata/jpa/docs/current/reference/html/
- Apache Tomcat: http://tomcat.apache.org/tomcat-8.0-doc/
- Log4J: http://logging.apache.org/log4j/2.x/manual/api.html
- JSTL: https://www.tutorialspoint.com/jsp/jsp_standard_tag_library.htm
- Bootstrap: http://getbootstrap.com/
- Fontawesome: http://fontawesome.io/icons/
- Además de consultas puntuales en otras páginas, manuales y foros de Internet.

Capítulo 8. Anexos técnicos

En esta capitulo se recogen todos los anexos técnicos que no tienen cabida en apartados anteriores.

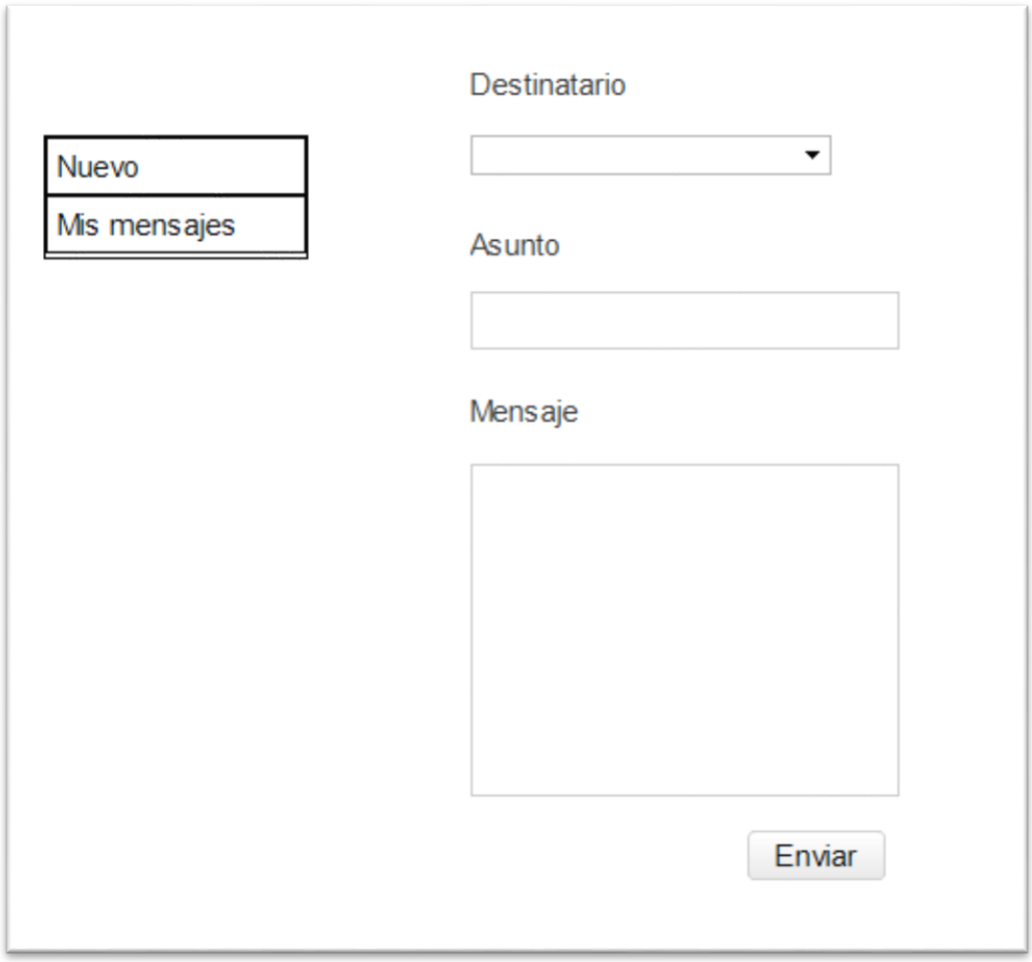
8.1. Casos de uso

A continuación, se exponen todos los casos de uso especificados.

8.1.1. CU01 – Módulo de comunicación padres-profesores

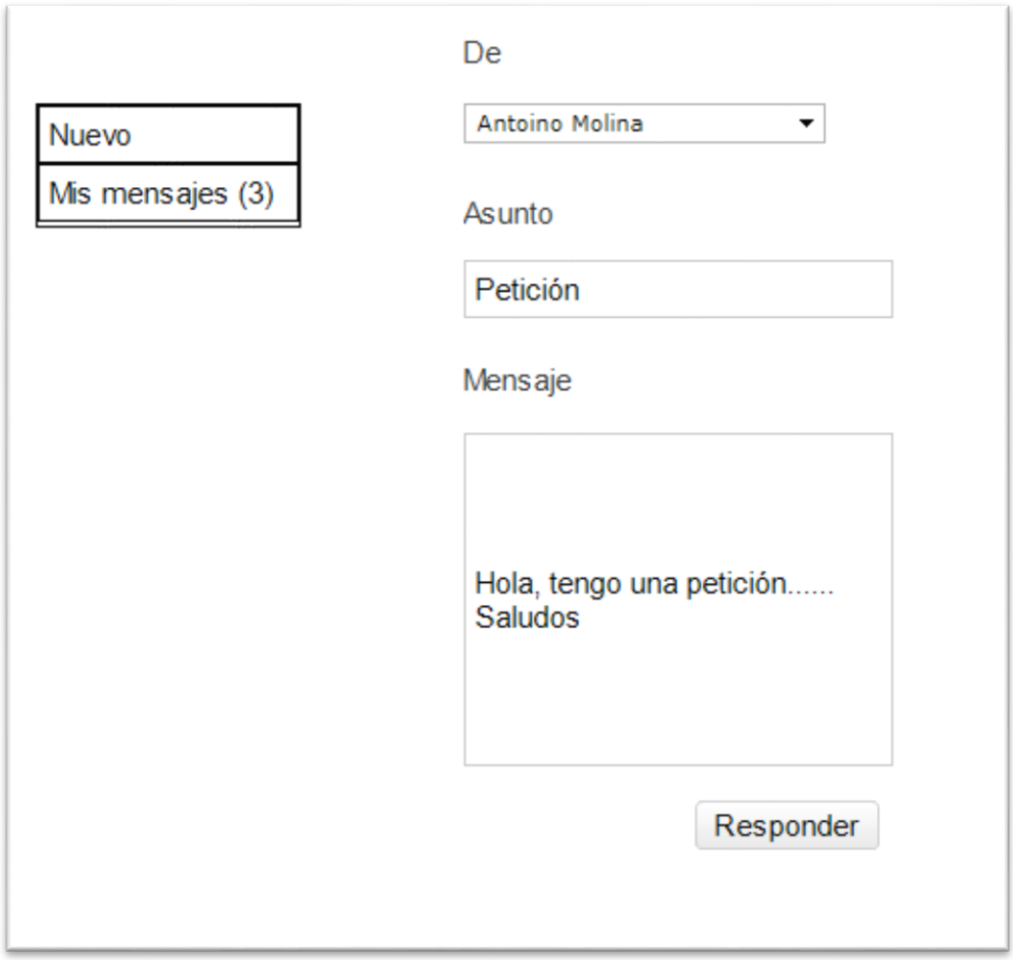
CU01.01 – Enviar un mensaje

Título	CU01.01 – Enviar un mensaje
Descripción	El usuario podrá enviar un mensaje
Actor	Padre, Profesor
Pre-condición	El usuario debe estar registrado y conectado en el sistema
Post-condición	El mensaje es enviado al destinatario indicado
Prioridad	Alta
Escenario principal	
1. Un usuario padre desea enviar un mensaje a un profesor. 2. El usuario pulsa el botón buscar 3. El sistema mostrara una lista con los posibles usuarios a los que el usuario puede mandar el mensaje 4. El usuario selecciona uno de los resultados obtenidos. 5. En el campo destinatario aparecerá el usuario seleccionado. 6. El usuario introduce el asunto en el campo correspondiente. 7. El sistema mostrará el texto introducido. 8. El usuario escribe el texto que desea enviar en el área de texto del mensaje. 9. En el área de texto del mensaje aparecerá el mensaje introducido por el usuario.	

10. El usuario pulsará el botón enviar 11. El sistema enviará el mensaje al destinatario indicado.
Escenario alternativo I
1. Un usuario profesor desea enviar un mensaje a un padre.
Escenario alternativo II
4. El usuario no selecciona ningún destinatario. 11. El sistema mostrará un mensaje de error indicando que debe introducir el destinatario.
Escenario alternativo III
6. El usuario deja el campo asunto vacío.
Maqueta de interfaz


CU01.02 – Leer un mensaje recibido

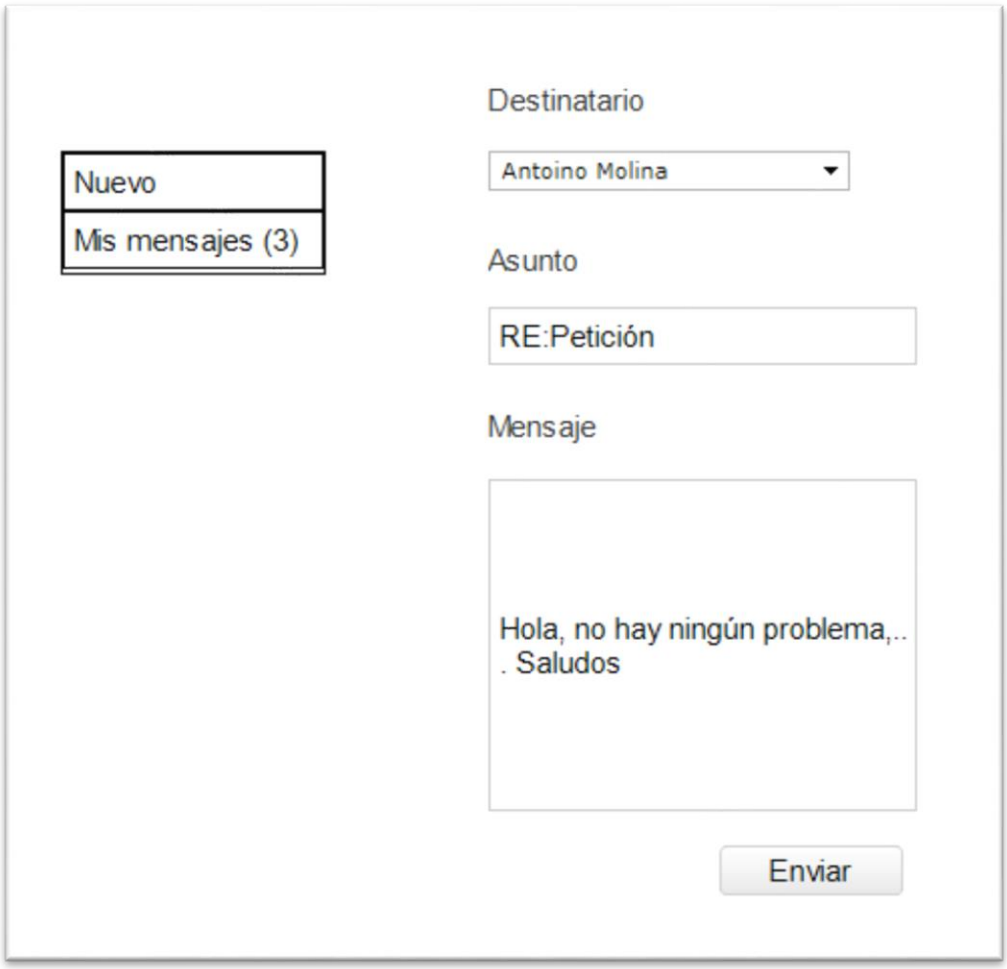
Título	CU01.02 – Leer un mensaje recibido
Descripción	El usuario podrá recibir un mensaje que le haya sido enviado
Actor	Padre, Profesor
Pre-condición	El usuario debe estar registrado y conectado en el sistema
Post-condición	El mensaje es leído por parte del usuario
Prioridad	Alta

Escenario principal
1. El usuario desea leer un mensaje recibido. 2. El usuario se dirige a la ventana de “mis mensajes”. 3. El usuario localiza el mensaje que desea leer. 4. El usuario pulsa el botón leer del mensaje deseado. 5. El sistema mostrará al usuario el contenido del mensaje seleccionado.
Escenario alternativo
Maqueta de interfaz


CU01.03 – Responder a un mensaje recibido

Título	CU01.03 – Responder a un mensaje recibido
Descripción	El usuario podrá responder a un mensaje que haya recibido
Actor	Padre, Profesor
Pre-condición	El usuario debe estar registrado y conectado en el sistema. Además, el usuario ha recibido un mensaje y se encuentra en la ventana de leer un mensaje recibido
Post-condición	El mensaje de respuesta es enviado

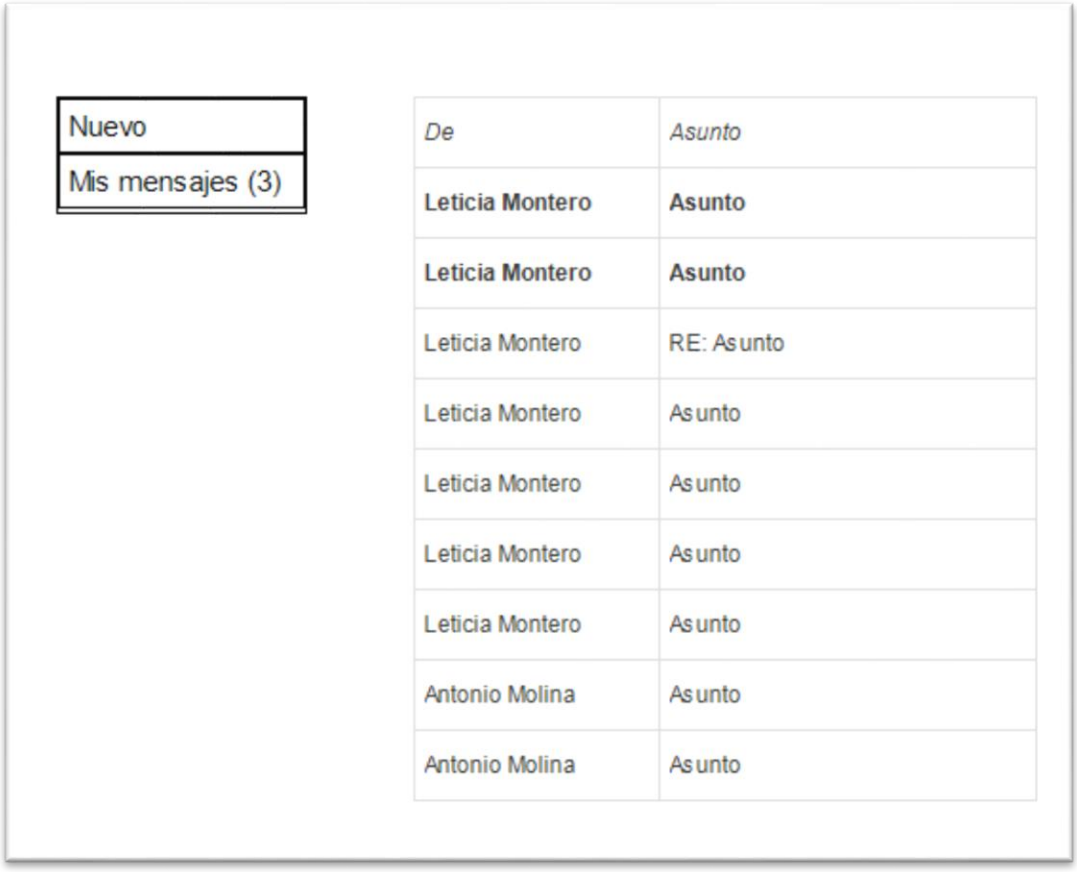
Prioridad	Alta
Escenario principal	
1. El usuario desea responder a un mensaje que ha recibido. 2. El usuario pulsa el botón responder. 3. El sistema muestra una nueva ventana con los campo destinatario y asunto rellenados. 4. El usuario introducirá el mensaje deseado. 5. El sistema mostrará el mensaje introducido. 6. El usuario pulsará el botón enviar. 7. El sistema enviará el mensaje.	
Escenario alternativo	
Maqueta de interfaz	



CU01.04 – Visualizar mensajes recibidos

Título	CU01.04 – Visualizar mensajes recibidos
Descripción	El usuario podrá consultar todos los mensajes recibidos
Actor	Padre, Profesor
Pre-condición	El usuario debe estar registrado y conectado en el sistema

Post-condición	Los mensajes son consultados por el usuario
Prioridad	Alta
Escenario principal	
1. El usuario Padre/profesor desea consultar sus mensajes. 2. El usuario pulsa en el label Mis mensajes. 3. El sistema mostrará una lista con los mensajes recibidos de éste usuario.	
Escenario alternativo	
3. El sistema mostrará una lista vacía al no encontrar mensajes asociados al usuario en cuestión.	
Maqueta de interfaz	

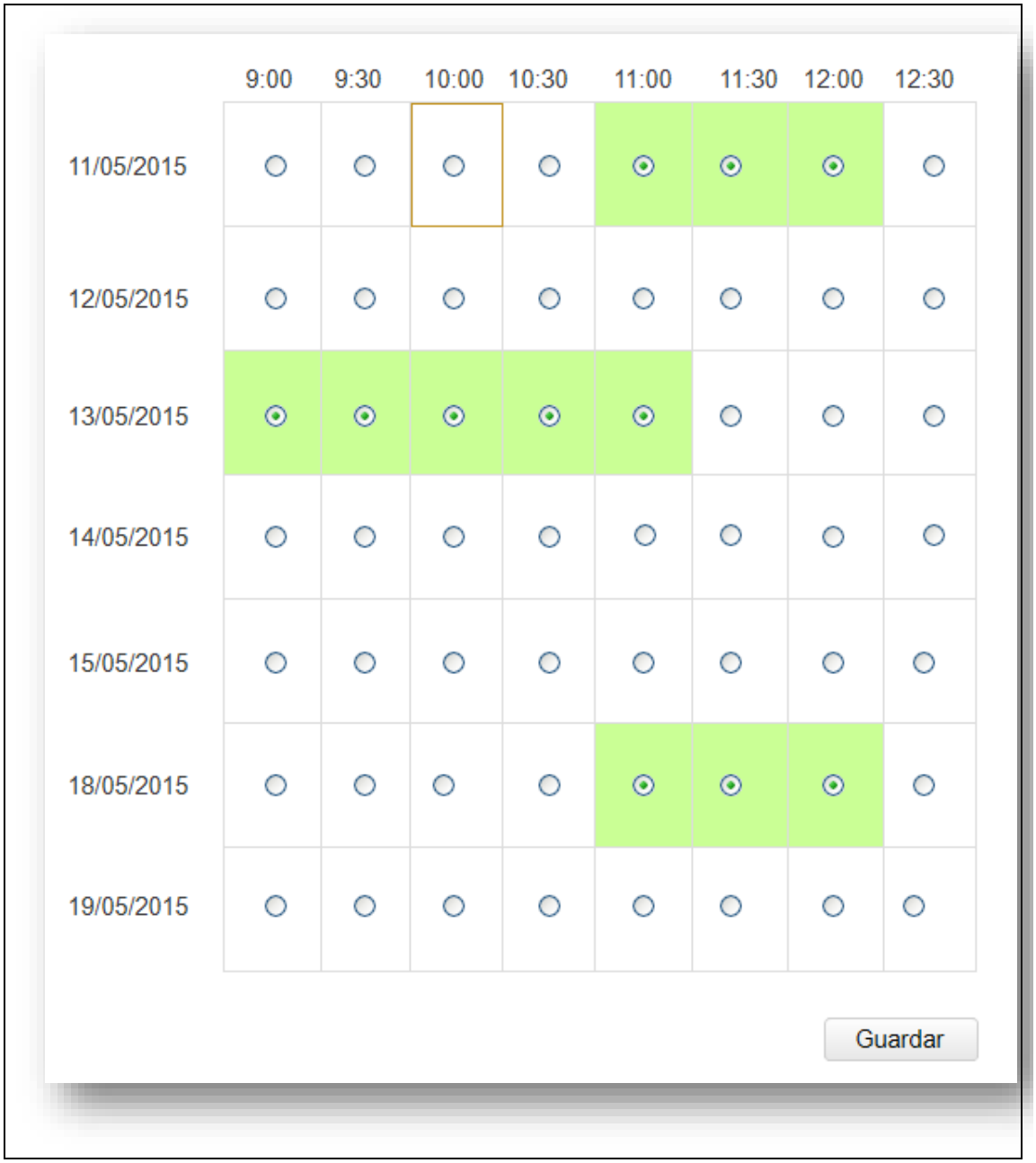


8.1.2. CU02 – Módulo de tutorías online

CU02.01 – Habilitar/deshabilitar franjas horarias para tutorías

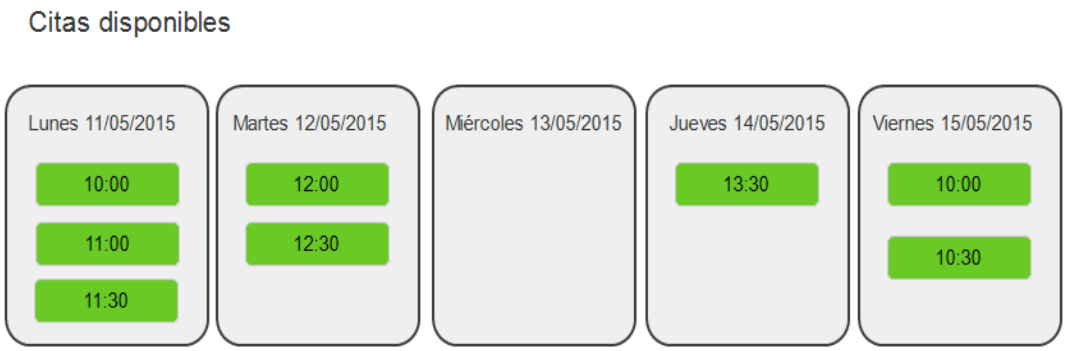
Título	CU02.01 Habilitar/Deshabilitar franjas de tiempo para tutorías
Descripción	El usuario podrá habilitar/deshabilitar horas libres para tutorías
Actor	Profesor

Pre-condición	El usuario debe estar registrado y conectado en el sistema
Post-condición	Las horas seleccionadas quedan habilitadas/deshabilitadas
Prioridad	Alta
Escenario principal	
1. El usuario desea habilitar franjas de tiempo para tutorías. 2. El sistema muestra una tabla con los días de la semana 3. El usuario hace un tick en las franjas de tiempo que desea habilitar. 4. El usuario pulsa en Guardar. 5. El sistema marca las franjas habilitadas con un color diferente.	
Escenario alternativo I	
1. El usuario desea deshabilitar franjas anteriormente habilitadas. 2. El usuario hace un tick en las franajas que desea deshabilitar 3. El usuario pulsa en Guardar. 4. El sistema desmarca las franjas en cuestión y vuelven a estar sin color.	
Escenario alternativo II	
1. El usuario desea deshabilitar franjas anteriormente habilitadas. 2. El usuario hace un tick en las franajas que desea deshabilitar 3. El usuario pulsa en Guardar. 4. El sistema muestra un mensaje de error indicando que la cita seleccionada no puede ser deshabilitada, ya que ha sido solicitada por un usuario.	
Maqueta de interfaz	



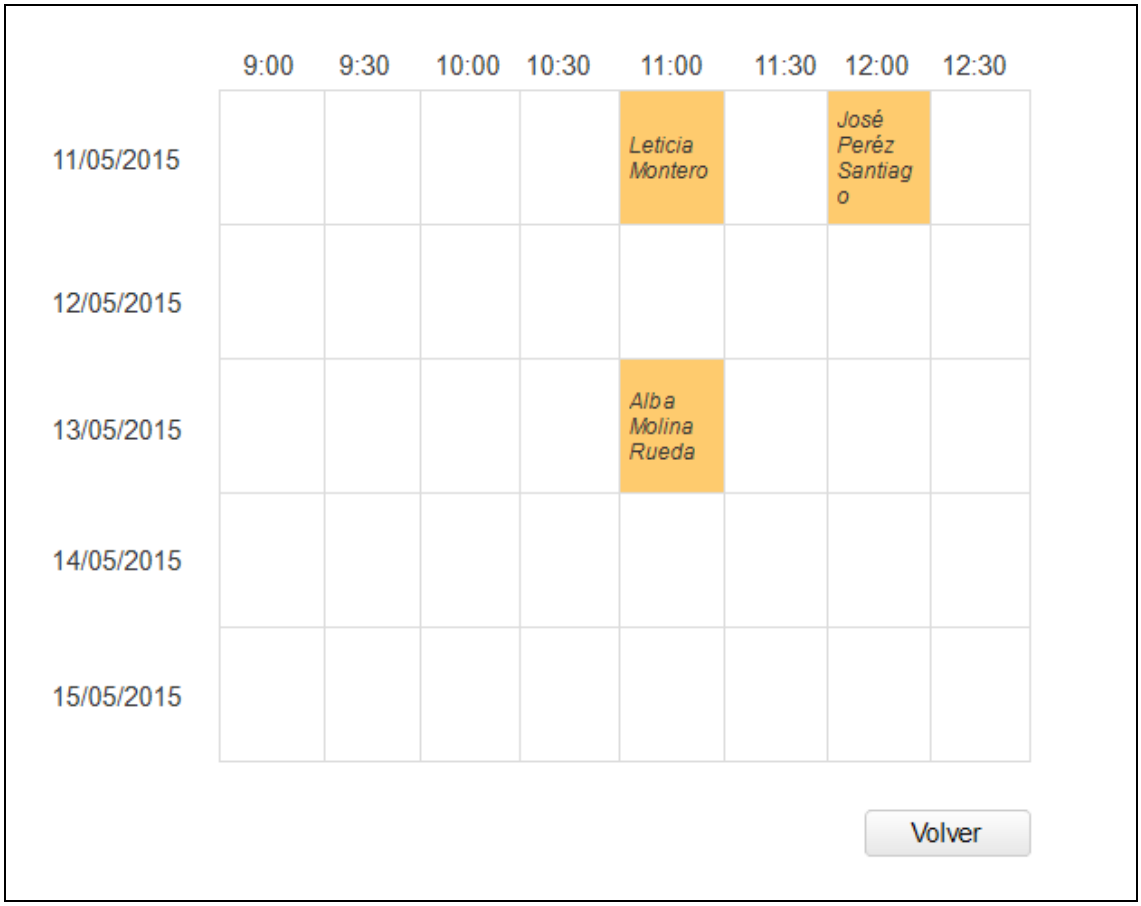
CU02.02 – Solicitar tutoría

Título	CU02.02 – Solicitar tutorías
Descripción	El usuario podrá solicitar tutorías seleccionando alguna de las franjas horarias que le aparecen como libres
Actor	Padre
Pre-condición	El usuario debe estar registrado y conectado en el sistema
Post-condición	Las horas seleccionadas pasan a estar ocupadas
Prioridad	Alta
Escenario principal	
1. El usuario desea solicitar tutoría.	

2. El usuario pulsa para ver las franjas libres. 3. El sistema mostrará las citas disponibles para la semana actual. 4. El usuario escoge la cita que más le conviene y pulsa el botón correspondiente. 5. El sistema mostrará un mensaje para confirmar la solicitud. 6. El usuario confirma su solicitud. 7. La cita escogida queda guardada en el sistema. 8. La franja de la cita en cuestión queda marcada en color rojo en la pantalla de habilitar/deshabilitar. 9. La franja de la cita en cuestión queda marcada en otro color en la pantalla de visulizar tutorías, mostrando información acerca de la cita.
Escenario alternativo I
6. El usuario cancela la solicitud. 7. El sistema vuelva a mostrar la pantalla de citas disponibles.
Maqueta de interfaz


C U02.03 – Visualizar horas de tutorías

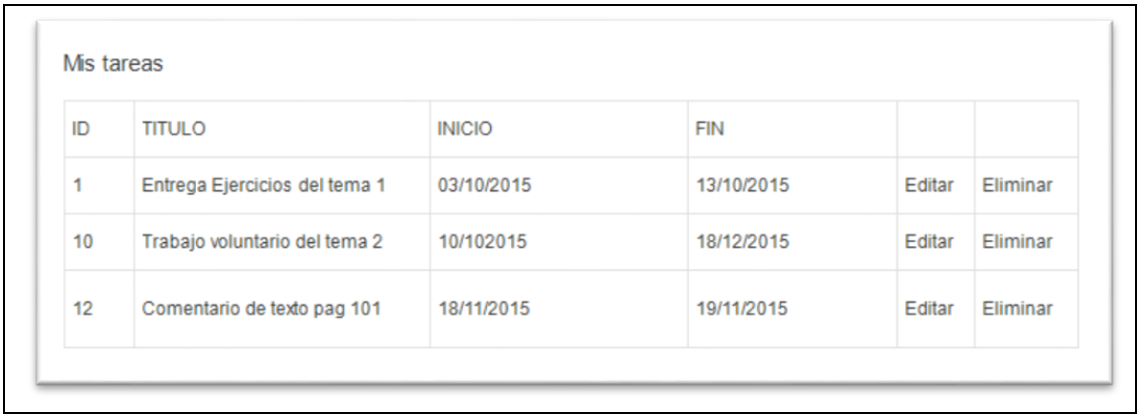
Título	CU02.03 – Visualizar horas de tutorías
Descripción	El usuario podrá visualizar las tutorías que han sido solicitadas, así para poder atenderlas.
Actor	Profesor
Pre-condición	El usuario debe estar registrado y conectado en el sistema
Post-condición	El resultado es mostrado al usuario
Prioridad	Alta
Escenario principal	
1. El usuario desea visualizar las tutorías. 2. El usuario pulsa para ver las tutorías. 3. El sistema mostrará una tabla con las horas que han sido escogidas, indicando el usuario que la haya solicitado.	
Escenario alternativo I	
3. El sistema muestra una tabla vacía, ya que ninguna hora ha sido solicitada.	
Maqueta de interfaz	



8.1.3. CU03 – Gestión de tareas

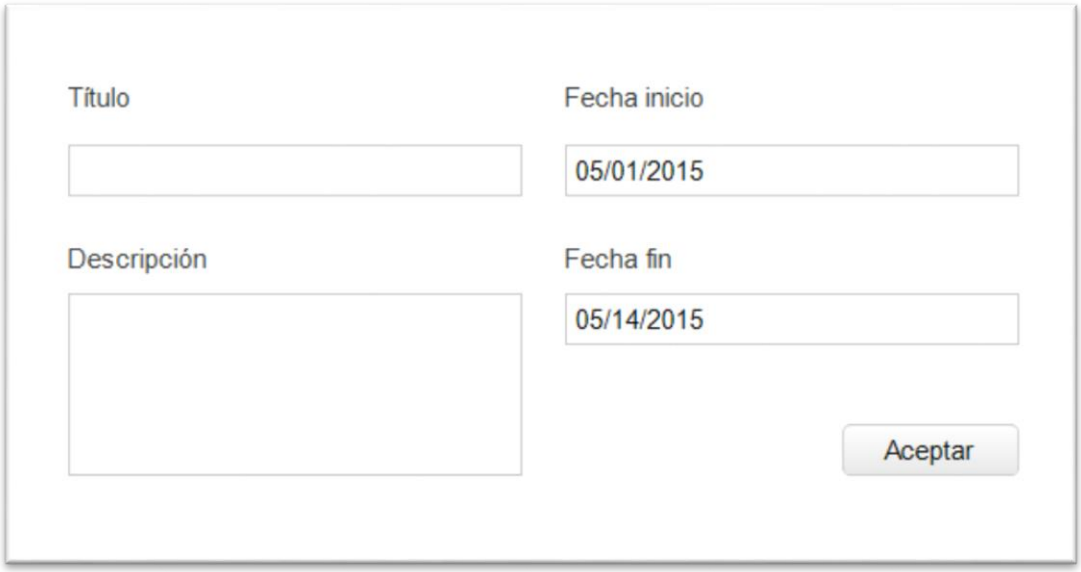
CU03.01 – Consultar tareas

Título	CU03.01 – Consultar tareas
Descripción	El usuario podrá consultar las tareas que tiene disponibles según su rol
Actor	Profesor, Padre, Alumno
Pre-condición	El usuario debe estar registrado y conectado en el sistema
Post-condición	Las tareas son consultadas
Prioridad	Alta
Escenario principal	
1. El usuario desea consultar sus tareas. 2. El usuario pulsa la sección Mis tareas. 3. El sistema muestra todas las tareas que puede visualizar el usuario.	
Escenario alternativo I	
3. El sistema informa al usuario que no tiene tareas.	
Maqueta de interfaz	



CU03.02 – Crear tarea

Título	CU03.02 – Crear una tarea
Descripción	El usuario podrá crear una tarea y establecer sus parámetros
Actor	Profesor
Pre-condición	El usuario debe estar registrado y conectado en el sistema
Post-condición	La tarea es creada
Prioridad	Alta
Escenario principal	
1. El usuario desea añadir una tarea nueva. 2. El usuario pulsa el botón para añadir una nueva tarea. 3. El sistema muestra el formulario a rellenar para crear una nueva tarea. 4. El usuario selecciona la asignatura. 5. El usuario introduce el título de la tarea. 6. El usuario selecciona una fecha de inicio de la tarea. 7. El usuario selecciona una fecha de fin de la tarea. 8. El usuario introduce la descripción de la tarea. 9. El usuario pulsa el botón Aceptar. 10. El sistema muestra un mensaje de que la tarea ha sido creada.	
Escenario alternativo I	
5. El usuario no introduce el título de la tarea. 10. El sistema muestra un mensaje de error indicando que falta el título de la tarea.	
Escenario alternativo II	
5. El usuario no introduce la fecha de inicio.	
Escenario alternativo III	
7. El usuario no introduce la fecha de fin. 10. El sistema muestra un mensaje de error indicando que falta la fecha de fin de la tarea.	
Escenario alternativo IV	

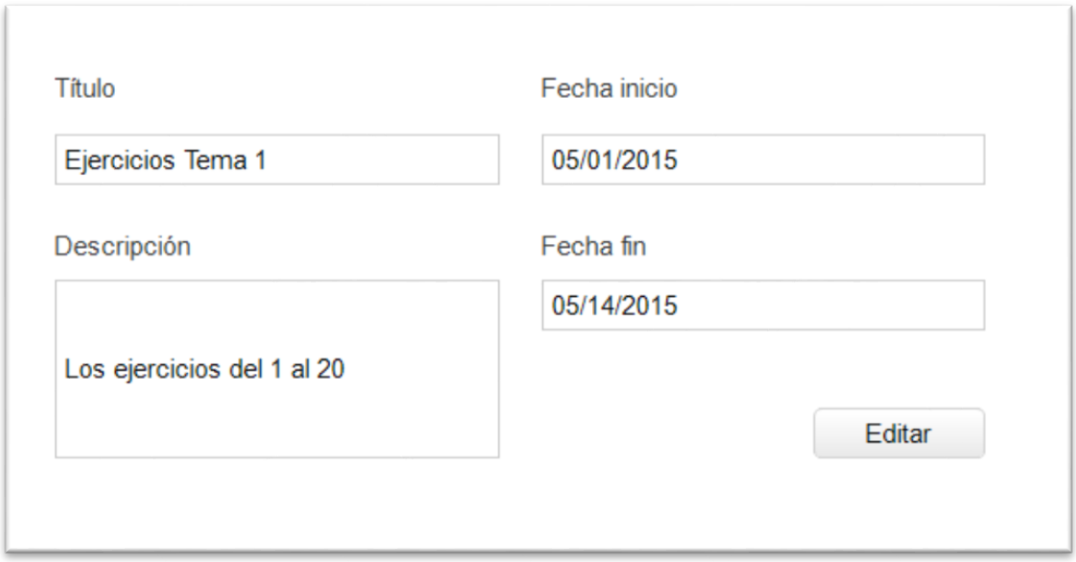
8. El usuario no introduce la descripción.
Maqueta de interfaz


CU03.03 – Detalle tarea

Título	CU03.03 – Detalle tarea
Descripción	El usuario podrá consultar la información relativa a una tarea
Actor	Administrador, Profesor, Padre, Alumno
Pre-condición	El usuario debe estar registrado y conectado en el sistema
Post-condición	La información de la tarea es consultada
Prioridad	Alta
Escenario principal	
1. El usuario desea consultar una tarea. 2. El usuario pulsa el botón ver de la tarea. 3. El sistema muestra un formulario con la información de la tarea.	
Escenario alternativo	
Maqueta de interfaz	

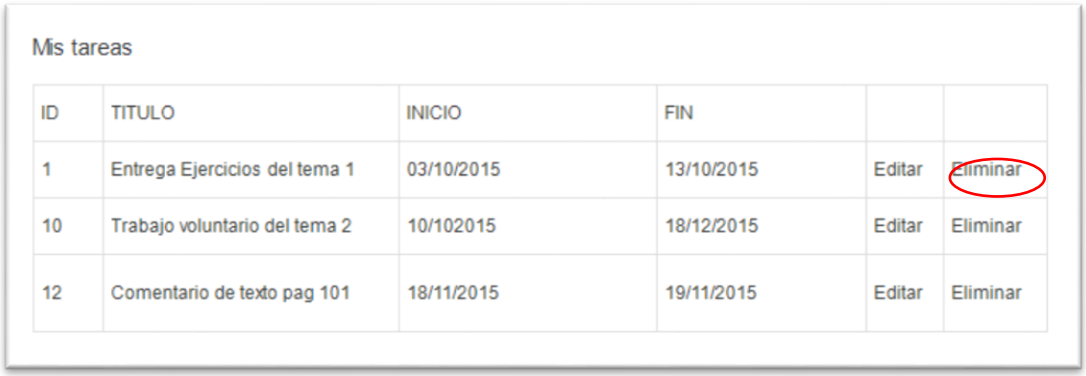
CU03.04 – Editar tarea

Título	CU03.04 – Modificar una tarea
Descripción	El usuario podrá modificar los datos de una tarea, como la fecha de inicio/fin, la descripción, pero no podrá modificar el título
Actor	Profesor
Pre-condición	El usuario debe estar registrado y conectado en el sistema

Post-condición	La tarea es modificada
Prioridad	Alta
Escenario principal	
4. El usuario desea modificar una tarea. 5. El usuario pulsa el botón editar de la tarea que desea editar. 6. El sistema muestra un formulario con los datos de la tarea a editar. 7. El usuario modifica los datos deseados. 8. El usuario pulsa el botón editar. 9. El sistema muestra un mensaje de que la tarea ha sido modificada.	
Escenario alternativo	
Maqueta de interfaz	
	

CU03.05 – Eliminar tarea

Título	CU03.05 – Eliminar una tarea
Descripción	El usuario podrá eliminar una tarea que haya creado
Actor	Profesor
Pre-condición	El usuario debe estar registrado y conectado en el sistema, además la tarea tiene que haber sido creada, y por el mismo usuario
Post-condición	La tarea es eliminada
Prioridad	Alta
Escenario principal	
1. El usuario desea eliminar una tarea. 2. El usuario pulsa el botón eliminar de la tarea que desea eliminar. 3. El sistema muestra una alerta de que la tarea va a ser eliminada. 4. El usuario confirma que desea eliminar la tarea.	

5. El sistema elimina la tarea, y ésta ya no aparece en el listado de tareas.
Escenario alternativo I
4. El usuario cancela la operación. 5. La tarea no se elimina.
Maqueta de interfaz


8.1.4. CU04 – Gestión de cursos

CU04.01 – Consultar curso

Título	CU04.01 – Consultar cursos
Descripción	El usuario podrá consultar los cursos disponibles.
Actor	Administrador
Pre-condición	El usuario debe estar registrado y conectado en el sistema
Post-condición	Los cursos son consultados
Prioridad	Media
Escenario principal	
12. El administrador desea consultar los cursos disponibles 13. El administrador pulsa en la sección “Cursos”. 14. El sistema muestra un listado con los cursos que hay disponibles.	
Escenario alternativo I	
2. El sistema muestra un listado vacío ya que no hay cursos creados en la base de datos	
Maqueta de interfaz	

CU04.02 – Crear curso

Título	CU04.02 – Crear curso
Descripción	El administrador podrá crear los distintos cursos necesarios.
Actor	Administrador
Pre-condición	El usuario debe estar registrado y conectado en el sistema
Post-condición	El curso es creado.

Prioridad	Media
Escenario principal	
6. El administrador desea crear un curso. 7. El administrador pulsa en el botón crear curso. 8. El administrador introduce el alías del curso a crear. 9. El administrador pulsa el botón crear. 10. El curso queda almacenado en la base de datos del sistema.	
Escenario alternativo	
Maqueta de interfaz	

CU04.03 – Detalle curso

Título	CU04.03 – Detalle Curso
Descripción	El administrador podrá consultar el detalle de curso seleccionado
Actor	Administrador
Pre-condición	El usuario debe estar registrado y conectado en el sistema
Post-condición	El detalle del curso es consultado
Prioridad	Baja
Escenario principal	
4. El usuario desea consultar el detalle de un curso. 5. El usuario se dirige al listado de cursos. 6. El usuario escoge el curso cuyo detalle desea consultar. 7. El usuario pulsa el botón Ver del curso escogido. 8. El sistema muestra el detalle del curso seleccionada.	
Maqueta de interfaz	

CU04.04 – Editar curso

Título	CU04.04 – Editar un grupo
Descripción	El administrado podrá modificar un curso creado con anterioridad.
Actor	Administrador
Pre-condición	El usuario debe estar registrado y conectado en el sistema. Además, el curso a modificar debe existir en la base de datos.
Post-condición	El curso queda modificado
Prioridad	Media
Escenario principal	
8. El administrador desea modificar un curso creado anteriormente.	

9. El administrador pulsa el botón editar. 10. El sistema muestra los datos del curso en cuestión con los campos que se puedan editar en modo editable. 11. El administrador modifica los datos deseados. 12. El administrador pulsa el botón Guardar. 13. El sistema guarda el curso modificado en la base de datos.
Escenario alternativo
Maqueta de interfaz

CU04.05 – Eliminar curso

Título	CU04.05 – Eliminar un curso
Descripción	El administrador podrá eliminar un curso creado anteriormente.
Actor	Administrador
Pre-condición	El usuario debe estar registrado y conectado en el sistema, además el curso a eliminar tiene que existir en la base de datos del sistema.
Post-condición	El curso es eliminado de la base de datos.
Prioridad	Media
Escenario principal	
6. El usuario desea eliminar un curso. 7. El usuario pulsa el botón eliminar del curso en cuestión. 8. El sistema elimina el curso.	
Escenario alternativo	
Maqueta de interfaz	

8.1.5. CU06 – Gestión de grupos

CU06.01 – Consultar grupos

Título	CU06.01 – Consultar grupos
Descripción	• Un usuario profesor podrá consultar el grupo que tiene asignado. • El administrador del sistema podrá consultar todos los grupos disponibles en el sistema.
Actor	Profesor, Administrador
Pre-condición	El usuario debe estar registrado y conectado en el sistema
Post-condición	Los grupos son consultados

Prioridad	Media
Escenario principal	
1. Un usuario profesor desea consultar el listado de grupos que tiene asignados. 2. El usuario pulsa en la sección “mis grupos”. 3. El sistema muestra un listado con los grupos que el usuario tiene asignados.	
Escenario alternativo I	
3. El sistema muestra un listado vacío ya que el usuario conectado no tiene grupos asignados.	
Escenario alternativo I	
1. El administrador desea consultar el listado de todos los grupos. 2. El usuario pulsa en la sección “grupos”. 3. El sistema muestra un listado con todos los grupos del sistema.	
Maqueta de interfaz	

CU06.02 – Crear grupo

Título	CU06.02 – Crear grupo
Descripción	El administrador podrá crear los distintos grupos necesarios.
Actor	Administrador
Pre-condición	El usuario debe estar registrado y conectado en el sistema
Post-condición	El grupo es creado.
Prioridad	Media
Escenario principal	
1. El administrador desea crear un grupo. 2. El administrador pulsa en el botón crear grupo. 3. El administrador introduce el alías del grupo a crear. 4. El administrador selecciona, si lo desea, un tutor para el grupo. 5. El administrador pulsa el botón crear. 6. El grupo queda almacenado en la base de datos del sistema.	
Escenario alternativo	
Maqueta de interfaz	

CU06.03 – Detalle grupo

Título	CU06.03 – Detalle Grupo
Descripción	• Un usuario profesor podrá consultar el detalle del grupo que tenga asignado. • El administrador podrá consultar el detalle de un grupo seleccionado

Actor	Administrador, Profesor
Pre-condición	El usuario debe estar registrado y conectado en el sistema
Post-condición	El detalle del curso es consultado
Prioridad	Baja
Escenario principal	
9. El usuario desea consultar el detalle de un grupo. 10. El usuario se dirige al listado de grupos. 11. El usuario escoge el grupo cuyo detalle desea consultar. 12. El usuario pulsa el botón Ver del grupo escogido. 13. El sistema muestra el detalle del grupo seleccionado.	
Maqueta de interfaz	

CU06.04 – Editar grupo

Título	CU06.04 – Modificar un grupo
Descripción	El administrado podrá modificar un grupo creado con anterioridad.
Actor	Administrador
Pre-condición	El usuario debe estar registrado y conectado en el sistema. Además, el grupo a modificar debe existir en la base de datos.
Post-condición	El grupo queda modificado
Prioridad	Media
Escenario principal	
1. El administrador desea modificar un grupo creado anteriormente. 2. El administrador pulsa el botón editar. 3. El sistema muestra los datos del grupo en cuestión con los campos que se puedan editar en modo editable. 4. El administrador modifica los datos deseados. 5. El administrador pulsa el botón Guardar. 6. El sistema guarda el grupo modificado en la base de datos.	
Escenario alternativo	
Maqueta de interfaz	

CU06.05 – Eliminar grupo

Título	CU06.05 – Eliminar un grupo
Descripción	El administrador podrá eliminar un grupo creado anteriormente.
Actor	Administrador
Pre-condición	El usuario debe estar registrado y conectado en el sistema, además el grupo a eliminar tiene que existir en la base de datos del sistema.

Post-condición	El grupo es eliminado de la base de datos.
Prioridad	Media
Escenario principal	
1. El usuario desea eliminar un grupo. 2. El usuario pulsa el botón eliminar del grupo en cuestión. 3. El sistema elimina el grupo.	
Escenario alternativo	
Maqueta de interfaz	

CU05.06 – Asignar tutor a grupo

Título	CU07.06 – Asignar tutor a un grupo
Descripción	El administrador podrá asignar un tutor a un grupo anteriormente creado
Actor	Administrador
Pre-condición	El usuario debe estar registrado y conectado en el sistema
Post-condición	La asignación profesor-grupo queda almacenada en la base de datos.
Prioridad	Media
Escenario principal	
4. El administrador desea asignar un tutor a un determinado grupo. 5. El administrador accede al listado de grupos. 6. El administrador selecciona el grupo en cuestión. 7. El administrador selecciona el profesor para el grupo seleccionado. 8. El administrador pulsa en el botón Guardar. 9. El sistema guarda la asignación seleccionada en la base de datos.	
Escenario alternativo	
6. El sistema muestra un mensaje de error indicando que el grupo asignado ya tiene profesor asignado.	
Maqueta de interfaz	

8.1.6. CU07 – Gestión de asignaturas

CU07.01 – Consultar asignaturas

Título	CU07.01 Consultar asignaturas
Descripción	• Un usuario alumno podrá consultar el listado de asignaturas en las que se encuentra matriculado.

	• Un usuario padre podrá consultar el listado de asignaturas a las que su perfil se encuentra vinculado. • Un usuario profesor podrá consultar las asignaturas que tiene asignadas. • El administrador del sistema podrá consultar el conjunto de todas las asignaturas.
Actor	Alumno, Padre, Profesor, Administrador
Pre-condición	El usuario debe estar registrado y conectado en el sistema
Post-condición	Las asignaturas son consultadas.
Prioridad	Alta
Escenario principal	
6. El usuario desea consultar sus asignaturas. 7. El usuario pulsa en la sección "mis asignaturas". 8. El sistema muestra un listado con las asignaturas del usuario.	
Maqueta de interfaz	

CU07.02 – Crear asignatura

Título	CU07.02 – Crear asignatura
Descripción	El administrador del sistema podrá crear asignatras.
Actor	Administrador
Pre-condición	El usuario debe estar registrado y conectado en el sistema
Post-condición	La asignatura es creada y almacenada en el sistema
Prioridad	Media
Escenario principal	
10. El administrador desea crear una asignatura. 11. El administrador pulsa en el botón "Nueva asignatura". 12. El sistema mostrará el formulario a rellenar para crear la asignatura. 13. El administrador introduce el nombre de la asignatura. 14. El administrador selecciona el curso al que pertenece la asignatura. 15. El administrador selecciona, si lo desea, el profesor de la asignatura. 16. El administrador pulsa en el botón Crear. 17. El sistema almacena la asignatura creada.	
Escenario alternativo I	
7. El usuario cancela la creación. 8. El sistema vuelva a mostrar la pantalla de listado de asignaturas.	
Maqueta de interfaz	

CU07.03 – Detalle asignatura

Título	CU07.03 – Detalle asignatura
Descripción	El usuario podrá consultar el detalle de una asignatura seleccionada.
Actor	Alumno, Padre, Profesor, Administrador
Pre-condición	El usuario debe estar registrado y conectado en el sistema
Post-condición	El detalle de la asignatura es consultado
Prioridad	Alta
Escenario principal	
14. El usuario desea consultar el detalle de una asignatura. 15. El usuario se dirige al listado de asignaturas. 16. El usuario escoge la asignatura cuyo detalle desea consultar. 17. El usuario pulsa el botón Ver de la asignatura escogida. 18. El sistema muestra el detalle de la asignatura seleccionada.	
Maqueta de interfaz	

CU07.04 – Editar asignatura

Título	CU07.04 – Editar asignatura
Descripción	El administrador podrá editar la información de una asignatura
Actor	Administrador
Pre-condición	El usuario debe estar registrado y conectado en el sistema
Post-condición	La información de la asignatura es modificada
Prioridad	Media
Escenario principal	
1. El administrador desea editar la información de una asignatura. 2. El administrador se dirige al listado de asignaturas. 3. El administrador escoge la asignatura cuyo detalle desea editar. 4. El administrador pulsa el botón Editar de la asignatura escogida. 5. El sistema muestra el detalle de la asignatura seleccionada con los campos editable. 6. El administrador edita los campos que desea. 7. El administrador pulsa el botón Guardar. 8. El sistema guarda la asignatura modificada.	
Maqueta de interfaz	

CU07.05 – Eliminar asignatura

Título	CU07.05 – Eliminar asignatura
Descripción	El administrador podrá eliminar una asignatura del sistema

Actor	Administrador
Pre-condición	El usuario debe estar registrado y conectado en el sistema
Post-condición	La asignatura es eliminada
Prioridad	Baja
Escenario principal	
1. El administrador desea eliminar una asignatura. 2. El usuario se dirige al listado de asignaturas. 3. El usuario escoge la asignatura que desea eliminar. 4. El usuario pulsa el botón Eliminar de la asignatura escogida. 5. El sistema elimina la asignatura y redirige el usuario al listado de asignaturas.	
Maqueta de interfaz	

8.1.7. CU08 – Gestión de usuarios

CU08.01 – Consultar listado de usuarios

Título	CU08.01 Consultar listado de usuarios
Descripción	• El administrador podrá consultar el listado de usuarios del sistema. • Un usuario profesor podrá consultar el listado de alumnos de un grupo, o de una asignatura que imparte.
Actor	Profesor, Administrador
Pre-condición	El usuario debe estar registrado y conectado en el sistema
Post-condición	Las asignaturas son consultadas.
Prioridad	Media
Escenario principal	
1. El administrador desea consultar el listado de usuario del sistema. 2. El administrador se dirige a la sección usuarios. 3. El sistema muestra un listado con todos los usuarios del sistema.	
Escenario alternativo	
1. Un usuario profesor desea consultar el listado de alumnos de un grupo. 2. El usuario se dirige a la sección grupos. 3. El usuario selecciona uno de los grupos mostrados y pulsa en ver. 4. El sistema muestra la información relativa al grupo y un botón de consultar alumnos. 5. El usuario pulsa el botón consultar alumnos. 6. El sistema muestra un listado con los alumnos del grupo en cuestión.	

CU08.02 – Dar de alta usuario padre

Título	CU08.02 Consultar listado de usuarios
Descripción	• Una persona puede darse de alta como padre en el sistema indicando el id de un alumno.
Actor	
Pre-condición	El id del alumno ha de existir en la base de datos.
Post-condición	El usuario es dado de alta
Prioridad	Media
Escenario principal	
1. Un usuario desea darse de alta como padre. 2. El usuario pulsa el botón de Darse de Alta. 3. El sistema muestra un formulario con los datos necesarios. 4. El usuario rellena los datos solicitados. 5. El usuario pulsa el botón Aceptar. 6. El sistema guarda el nuevo usuario en la base de datos vinculado al id de alumno introducido.	
Escenario alternativo	
6. El sistema muestra un mensaje de error ya que el id de alumno indicado no existe en la base de datos.	

CU08.03 – Ver perfil de usuario

Título	CU08.03 Ver perfil de usuario
Descripción	Todos los usuarios podrán consultar la información de su perfil
Actor	Administrador, Profesor, Padre, Alumno
Pre-condición	El usuario debe estar registrado y conectado en el sistema
Post-condición	El perfil de usuario es consultado.
Prioridad	Media
Escenario principal	
1. Un usuario desea consultar la información de su perfil. 2. El usuario se dirige a la sección de ver perfil de usuario. 3. El sistema muestra la información relativa al usuario conectado.	
Escenario alternativo	
1. Un usuario padre desea consultar la información del alumno vinculado a su perfil. 2. El usuario se dirige a la sección de ver perfil de usuario. 3. El sistema muestra la información relativa al usuario conectado, además de un link al perfil del alumno que tenga vinculado. 4. El usuario pincha en el link mencionado en el paso anterior. 5. El sistema muestra la información relativa al alumno vinculado al usuario conectado.	

8.2. Código fuente

En este apartado, exponemos fragmentos de código que hemos pensado que es necesario incluir para la comprensión de lo expuesto en otros apartados.

8.2.1. Entidad Usuario

```
package com.tfg.scpp.entity;



/**
 * The persistent class for the usuario database table.
 *
 */
@Entity
@DiscriminatorColumn(name="ROL")
@NamedQuery(name="Usuario.findAll", query="SELECT u FROM Usuario
u")
public class Usuario implements Serializable {
	private static final long serialVersionUID = 1L;
	private String id;
	private String apellidos;
	private String email;
	private Estado estado;
	private String nombre;
	private String password;
	private String fullName;
	private List<Cita> citasSolicitante;
	private List<Mensaje> mensajeEnviados;
	private List<Mensaje> mensajesRecibidos;
	private Rol rol;

	public Usuario() {

	}

	public Usuario(Usuario usuario){
		this.id = usuario.getId();
```

```
            this.nombre = usuario.getNombre();
            this.apellidos = usuario.getApellidos();
            this.password = usuario.getPassword();
            this.fullName = usuario.getFullName();
            this.email = usuario.getEmail();
            this.estado = usuario.getEstado();
            this.rol = usuario.rol;
      }


      @Id
      @NotEmpty(message="Campo obligatorio",
groups=MensajeUno.class)
    @Valid
      public String getId() {
            return this.id;
      }

      public void setId(String id) {
            this.id = id;
      }

      @NotEmpty(message="Campo obligatorio")
    @Valid
      public String getApellidos() {
            return this.apellidos;
      }

      public void setApellidos(String apellidos) {
            this.apellidos = apellidos;
      }

      @NotEmpty(message="Campo obligatorio")
    @Valid
      public String getEmail() {
            return this.email;
      }

      public void setEmail(String email) {
            this.email = email;
      }
      @Enumerated(EnumType.STRING)
      public Estado getEstado() {
            return this.estado;
      }

      public void setEstado(Estado estado) {
            this.estado = estado;
      }

      @NotEmpty(message="Campo obligatorio")
    @Valid
      public String getNombre() {
            return this.nombre;
      }
```

```
public void setNombre(String nombre) {
		this.nombre = nombre;
	}

	@NotEmpty(message="Campo obligatorio")
    @Valid
	public String getPassword() {
		return this.password;
	}

	public void setPassword(String password) {
		this.password = password;
	}

	//bi-directional many-to-one association to Cita
	@OneToMany(mappedBy="solicitante")
	public List<Cita> getCitasSolicitante() {
		return this.citasSolicitante;
	}

	public void setCitasSolicitante(List<Cita> citasSolicitante) {
		this.citasSolicitante = citasSolicitante;
	}
//bi-directional many-to-one association to Mensaje
	@OneToMany(mappedBy="fuente")
	public List<Mensaje> getMensajesEnviados() {
		return this.mensajeEnviados;
	}

	public void setMensajesEnviados(List<Mensaje> mensajes1) {
		this.mensajeEnviados = mensajes1;
	}

	//bi-directional many-to-one association to Mensaje
	@OneToMany(mappedBy="destino")
	public List<Mensaje> getMensajesRecibidos() {
		return this.mensajesRecibidos;
	}

	public void setMensajesRecibidos(List<Mensaje> mensajes2) {
		this.mensajesRecibidos = mensajes2;
	}

	@Transient
	public String getFullName(){
		this.fullName = this.nombre + " " + this.apellidos;
		return this.fullName;
	}

	public void setFullName(String str){
		fullName = str;
	}
```

```
// bi-directional many-to-one association to Rol
	@ManyToOne
	@JoinColumn(name = "ROL", insertable = false, updatable =
false)
	public Rol getRol() {
		return this.rol;
	}

	public void setRol(Rol r) {
		this.rol = r;
	}

	public boolean equals(Object o){
		boolean res = false;

		if(o instanceof Usuario){
			Usuario u = (Usuario) o;
			res =  id.equalsIgnoreCase(u.id);
		}
		return res;
	}

	public int hashCode(){
		return id.hashCode();
	}



	@Override
	public String toString() {
		return "Usuario [id=" + id + ", apellidos=" + apellidos
		+ ", estado=" + estado + ", nombre=" + nombre + "]";
	}
}
```

8.2.2. Entidad Mensaje

```
package com.tfg.scpp.entity;

import java.io.Serializable;
import java.sql.Timestamp;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.NamedQuery;
import javax.validation.Valid;
import javax.validation.constraints.NotNull;

import org.hibernate.validator.constraints.NotEmpty;


/**
 * The persistent class for the mensaje database table.
 *
 */
@Entity
@NamedQuery(name="Mensaje.findAll", query="SELECT m FROM Mensaje
m")
public class Mensaje implements Serializable {
	private static final long serialVersionUID = 1L;
	private long id;
	private String asunto;
	private String cuerpo;
	private Timestamp fecha;
	private boolean leido;
	private Usuario fuente;
	private Usuario destino;

	public Mensaje() {
	}


	@Id
	@GeneratedValue(strategy=GenerationType.AUTO)
	public long getId() {
		return this.id;
	}

	public void setId(long id) {
		this.id = id;
	}

	@NotEmpty(message="Campo obligatorio")
    @Valid
	public String getAsunto() {
		return this.asunto;
	}

	public void setAsunto(String asunto) {
		this.asunto = asunto;
	}
```

```
@NotEmpty(message="Campo obligatorio")
    @Valid
      public String getCuerpo() {
            return this.cuerpo;
      }

      public void setCuerpo(String cuerpo) {
            this.cuerpo = cuerpo;
      }

      public Timestamp getFecha() {
            return this.fecha;
      }

      public void setFecha(Timestamp fecha) {
            this.fecha = fecha;
      }

      public boolean getLeido() {
            return this.leido;
      }

      public void setLeido(boolean leido) {
            this.leido = leido;
      }

      //bi-directional many-to-one association to Usuario
      @ManyToOne
      @JoinColumn(name="FROM_USER")
      public Usuario getFuente() {
            return this.fuente;
      }

      public void setFuente(Usuario usuario1) {
            this.fuente = usuario1;
      }

      //bi-directional many-to-one association to Usuario
      @ManyToOne
      @JoinColumn(name="TO_USER")
      @NotNull(message="Campo obligatorio")
    @Valid
      public Usuario getDestino() {
            return this.destino;
      }

      public void setDestino(Usuario usuario2) {
            this.destino = usuario2;
      }

      @Override
      public String toString() {
            return "Mensaje [id=" + id + ", asunto=" + asunto + ",
cuerpo="
                        + cuerpo + ", fecha=" + fecha + ", leido=" +
leido
                        + ", fuente=" + fuente.getFullName() + ",
destino=" + destino.getFullName() + "]";
      }

}
```

8.3. Casos de prueba

En este apartado, expopnemos algunos de los casos de prueba especificados para validar las funcionalidades del sistema.

Titulo		CP01.01 – Enviar un mensaje		
Caso de uso		CU01.01 – Enviar un mensaje		
Precondición		Usuario registrado y conectado		
ID	**Estado**	**Entrada / acción del usuario**	**Salida esperada**	**Resultado**
1	Pantalla de enviar nuevo mensaje	1. El usuario selecciona un usuario para el destinatario. 2. El usuario introduce el asunto “asunto”. 3. El usuario introduce el texto a enviar “texto a enviar”. 4. El usuario pulsa el botón enviar.	5. El sistema redirigirá el usuario a la pantalla de los mensajes recibidos	OK
2	Pantalla de enviar nuevo mensaje	1. El usuario introduce el asunto “asunto”. 2. El usuario introduce el texto a enviar “texto a enviar”. 3. El usuario pulsa el botón enviar.	4. El sistema mostrará una advertencia de que el usuario debe seleccionar un destinatario.	OK
3	Pantalla de enviar nuevo mensaje	1. El usuario selecciona un usuario para el destinatario. 2. El usuario introduce el asunto “asunto”. 3. El usuario pulsa el botón enviar.	4. El sistema mostrará una advertencia de que el usuario debe introducir un texto a enviar.	OK

Titulo		CP01.02 – Leer un mensaje recibido		
Caso de uso		CU01.02 – Leer un mensaje recibido		
Precondición		• El usuario debe estar registrado y conectado • El usuario ha recibido un mensaje nuevo		
ID	**Estado**	**Entrada / acción del usuario**	**Salida esperada**	**Resultado**

4	Pantalla de mensajes recibidos	1. El usuario localiza el mensaje que desea leer. 2. El usuario pulsa el botón leer del mensaje correspondiente.	3. El sistema mostrará el contenido del mensaje.	OK

Titulo		CP01.03 – Responder a un mensaje recibido		
Caso de uso		CU01.03 – Responder a un mensaje recibido		
Precondición		• El usuario debe estar registrado y conectado • El usuario ha recibido un mensaje nuevo		
ID	**Estado**	**Entrada / acción del usuario**	**Salida esperada**	**Resultado**
5	Pantalla de visualización del contenido de un mensaje	1. El usuario pulsa el botón responder. 3. El usuario introduce la respuesta. 4. El usuario pulsa el botón Enviar.	2. El sistema muestra la página para responder al mensaje. 5. El sistema redirigirá el usuario a la pantalla de los mensajes recibidos.	OK

Titulo		CP01.04 – Visualizar mensajes recibidos		
Caso de uso		CU01.04 – Visualizar mensajes recibidos		
Precondición		El usuario debe estar registrado y conectado		
ID	**Estado**	**Entrada / acción del usuario**	**Salida esperada**	**Resultado**
6	Pantalla de visualización de los mensajes	1. El usuario accede a la pantalla de los mensajes recibidos.	2. El sistema muestra los mensajes recibidos del usuario.	OK
7	Pantalla de visualización de los mensajes	1. El usuario accede a la pantalla de los mensajes recibidos.	2. El sistema muestra una lista vacía, indicando que el usuario no tiene ningún mensaje recibido	OK

Titulo		CP02.01 Habilitar/Deshabilitar franjas de tiempo para tutorías		
Caso de uso		CU02.01 Habilitar/Deshabilitar franjas de tiempo para tutorías		
Precondición		• Usuario registrado y conectado. • Rol Profesor		
ID	**Estado**	**Entrada / acción del usuario**	**Salida esperada**	**Resultado**
8	Pantalla de habilitar/ deshabilitar tutorías	1. El usuario selecciona las franjas que desea habilitar. 2. El usuario pulsa el botón Aceptar.	3. El sistema guarda la selección del usuario, y muestra las franjas escogidas en color verde.	OK
9	Pantalla de habilitar/ deshabilitar tutorías	1. El usuario deselecciona algunas de las franjas que había habilitado anteriormente. 2. El usuario pulsa el botón Aceptar.	3. El sistema borra las franjas que han sido deshabilitadas, y las muestra en color blanco.	OK
10	Pantalla de habilitar/ deshabilitar tutorías	1. El usuario no realiza ningún cambio. 2. El usuario pulsa el botón Aceptar.	3. El sistema recarga la página, mostrando la información de la página.	OK

Titulo		CP02.02 Solicitar tutoría		
Caso de uso		CU02.02 Solicitar tutoría		
Precondición		• Usuario registrado y conectado. • Rol Padre		
ID	**Estado**	**Entrada / acción del usuario**	**Salida esperada**	**Resultado**
11	Pantalla de solicitar tutoría	1. El usuario escoge una de las citas que le muestra la pantalla. 2. El usuario pulsa en el botón correspondiente.	3. El sistema guarda la cita escogida por el usuario y redirige a la pantalla principal.	OK

Titulo	CP02.03 Visualizar horas de tutoría
Caso de uso	CU02.02 Visualizar horas de tutoría

Precondición		• Usuario registrado y conectado. • Rol Profesor		
ID	**Estado**	**Entrada / acción del usuario**	**Salida esperada**	**Resultado**
12	Pantalla de Visualizar horas de tutoría	1. El usuario accede a la pantalla de consulta de las horas de tutoría.	2. El sistema muestra una tabla con franjas horarias, indicando en otro color las que hayan sido solicitadas, además de indicar el nombre de la persona que la ha solicitado	OK
13	Pantalla de Visualizar horas de tutoría	1. El usuario accede a la pantalla de consulta de las horas de tutoría.	2. El sistema muestra una tabla vacía al no haber tutoráis solicitadas.	OK

Titulo		CP03.01 Consultar tareas		
Caso de uso		CU03.01 Consultar tareas		
Precondición		• Usuario registrado y loguado		
ID	**Estado**	**Entrada / acción del usuario**	**Salida esperada**	**Resultado**
14	Pantalla de consultar tareas	1. El usuario accede a la pantalla de consultar tareas.	2. El sistema muestra un listado de tareas que hay creadas para él.	OK
15	Pantalla de consultar tareas	1. El usuario accede a la pantalla de consultar tareas.	2. El sistema muestra un listado vacío al no existir tareas creadas.	OK

Titulo		CP03.02 Crear una tarea		
Caso de uso		CU03.02 Crear una tarea		
Precondición		• Usuario registrado y conectado. • Rol Profesor.		
ID	**Estado**	**Entrada / acción del usuario**	**Salida esperada**	**Resultado**

16	Pantalla de añadir una tarea nueva.	1. El usuario selecciona la asignatura 2. El usuario introduce el título “tarea1” de la tarea. 3. El usuario introduce la descripción de la tarea “Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean commodo ligula eget dolor.” 4. El usuario introduce la fecha y hora de inicio “30/08/2016 12:00” 5. El usuario introduce la fecha de finalización “15/09/2016 12:00”. 6. El usuario pulsa el botón Aceptar.	7. El sistema guarda la tarea creada y redirige el usuario a la pantalla de consultar tareas, dónde podrá consultar la nueva tarea creada.	OK
17	Pantalla de añadir una tarea nueva.	1. El usuario introduce la descripción de la tarea “Descripción tarea”. 2. El usuario introduce la fecha y hora de inicio “12/08/2016 12:00”. 3. El usuario introduce la fecha y hora de finalización “17/08/2016 13:00”. 4. El usuario pulsa el botón Aceptar.	5. El sistema muestra un mensaje de error, ya que el usuario no ha introducido el título de la tarea.	OK
18	Pantalla de añadir una tarea nueva.	1. El usuario introduce el título de la tarea “Tarea 2”.	5. El sistema muestra un mensaje de error, ya que el usuario no ha	OK

		2. El usuario introduce la fecha y hora de inicio “12/08/2016 12:00”. 3. El usuario introduce la fecha y hora de finalización “17/08/2016 13:00”. 4. El usuario pulsa el botón Aceptar.	introducido la descripción de la tarea.	
19	Pantalla de añadir una tarea nueva.	1. El usuario introduce el título de la tarea “Tarea 2”. 2. El usuario introduce la descripción de la tarea “Descripción tarea”. 3. El usuario introduce la fecha y hora de finalización “17/08/2016 13:00”. 4. El usuario pulsa el botón Aceptar.	5. El sistema muestra un mensaje de error indicando que el usuario no ha introducido la fecha de inicio de la tarea.	OK
20	Pantalla de añadir una tarea nueva.	1. El usuario introduce el título de la tarea “Tarea 2”. 2. El usuario introduce la descripción de la tarea “Descripción tarea”. 3. El usuario introduce la fecha de inicio “12/08/2016”. 4. El usuario introduce la fecha y hora de finalización “17/08/2016 13:00”. 5. El usuario pulsa el botón Aceptar.	6. El sistema mostrará un mensaje de error indicando que el campo fecha es incompleto.	OK
21	Pantalla de añadir una tarea nueva.	1. El usuario introduce el título de la tarea “Tarea 2”.	5. El sistema mostrará un mensaje de error	

		2. El usuario introduce la descripción de la tarea “Descripción tarea”. 3. El usuario introduce la fecha de inicio “12/08/2016”. 4. El usuario pulsa el botón Aceptar.	indicando que falta la fecha de finalización de la tarea.	
22	Pantalla de añadir una tarea nueva.	1. El usuario introduce el título de la tarea “Tarea 2”. 2. El usuario introduce la descripción de la tarea “Descripción tarea”. 3. El usuario introduce la fecha de inicio “12/08/2016”. 4. El usuario introduce la fecha y hora de finalización “17/08/2016”. 5. El usuario pulsa el botón Aceptar.	6. El sistema muestra un mensaje de error indicando que el campo de fecha fin está incompleto.	OK
23	Pantalla de añadir una tarea nueva.	1. El usuario introduce el título de la tarea “Tarea 2”. 2. El usuario introduce la descripción de la tarea “Descripción tarea”. 3. El usuario introduce la fecha de inicio “12/08/2016”. 4. El usuario introduce la fecha y hora de finalización “07/08/2016 12:15”. 5. El usuario pulsa el botón Aceptar.	6. El sistema mostrará un mensaje de error indicando que la fecha fin no puede ser anterior a la fecha de inicio.	

Titulo		CP03.03 Detalle tarea		
Caso de uso		CU03.03 Detalle tarea		
Precondición		• Usuario registrado y conectado. • Todos los roles		
ID	**Estado**	**Entrada / acción del usuario**	**Salida esperada**	**Resultado**
24	Pantalla de consultar tareas	1. El usuario localiza la tarea que desea consultar. 2. El usuario pulsa el botón Ver correspondiente.	3. El sistema mostrara el detalle de la tarea en cuestión.	OK

Titulo		CP03.04 Modificar una tarea		
Caso de uso		CU03.04 Modificar una tarea		
Precondición		• Usuario registrado y conectado. • Rol Profesor.		
ID	**Estado**	**Entrada / acción del usuario**	**Salida esperada**	**Resultado**
25	Pantalla de consultar tareas	3. El usuario localiza la tarea que desea modificar. 4. El usuario pulsa el botón Editar correspondiente. 4. El usuario cambia la fecha de finalización de la tarea. 5. El usuario pulsa el botón Aceptar.	4. El sistema mostrara la tarea en cuestión con los campos editables. 6. El sistema guarda los cambios de la tarea y redirigirá el usuario a la pantalla de consulta de tareas dónde podrá comprobar los cambios efectuados.	OK

Titulo		CP03.05 Eliminar una tarea		
Caso de uso		CU03.05 Eliminar una tarea		
Precondición		• Usuario registrado y conectado. • Rol Profesor.		
ID	**Estado**	**Entrada / acción del usuario**	**Salida esperada**	**Resultado**

26	Pantalla de consultar tareas	1. El usuario localiza la tarea que desea eliminar. 2. El usuario pulsa el botón Eliminar. 4. El usuario confirma la operación.	3. El sistema mostrará un mensaje de confirmación. 5. El sistema recargará la página después de eliminar la tarea en cuestión.	OK
27	Pantalla de consultar tareas	1. El usuario localiza la tarea que desea eliminar. 2. El usuario pulsa el botón Eliminar. 4. El usuario cancela la operación.	3. El sistema mostrará un mensaje de confirmación. 5. La tarea no se elimina.	OK