Escuela Técnica Superior de Ingeniería Informática
Grado en Ingeniería de la salud

Desarrollo de un servicio de asistencia al triaje integrable en sistemas de telemedicina
Development of a triage support service which can be integrated into telehealth systems

Realizado por

MIGUEL GONZÁLEZ SOSA

Dirigido por

JOSÉ MARÍA ÁLVAREZ PALOMO
FERNANDO MORENO JABATO

Departamento

DEPARTAMENTO DE LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN

UNIVERSIDAD DE MÁLAGA
Málaga, septiembre de 2019.

Fecha de la defensa:
El Secretario del Tribunal

Resumen

Palabras clave optimización de recursos sanitarios, triaje, telemedicina, arquitectura de software, sistemas distribuidos, microservicios, *customer relationship management*

Gestionar las grandes afluencias de los servicios de urgencias es un proceso clave para optimizarlos recursos y mejorar las prestaciones del servicio de salud. Los casos con niveles bajos de emergencia son inapropiados para un servicio de urgencias hospitalario, siendo más adecuados la atención primaria o el autocuidado. Para evitar estas consultas inapropiadas es muy útil el triaje telemático, ya que minimiza que casos poco urgentes se desplacen hacia el hospital, y son especialmente interesantes durante épocas de pandemia. En este proyecto llevamos a cabo la implementación de un servicio que asiste y guía a diferentes profesionales en la aplicación de estos triajes, minimizando errores, aumentando la precisión, aumentando la rapidez y optimizando económicamente el proceso. El sistema es flexible respecto a los diferentes sistemas de triaje estructurados actuales y permite su total auditoría e integración en un sistema ya existente, como puede ser un CRM (*Customer Relationship Management*) sanitario. Afrontamos un escenario en el que, además de tener que darle solución a este problema, desconocemos en su totalidad el sistema en el que quiere ser integrado y su infrastructura. A esta problemática se le dio solución haciendo uso de arquitecturas orientadas a eventos, basadas en microservicios, permitiendo la implantación de sistemas distribuidos y la independencia de funcionamiento y agnosticismo respecto al entorno del servicio de triaje. Como entregable adicional, se ha desarrollado una interfaz web que permite ejecutar las funcionalidades principales del servicio de triaje.

Abstract

Keywords: optimization of health resources, triage, telehealth, software architechture, distributed systems, microservices, customer relationship management

Managing large influxes of emergency services is a key process for optimizing resources and improving health service delivery. Low-emergency level cases are inappropriate for a hospital emergency department, with primary care or self-care being more appropriate. Telematic triage is very useful to avoid these inappropriate consultations, as it minimizes the need for non-emergent cases to travel to the hospital, and they are especially interesting during pandemic times. In this project we implemented a service that assists and guides different professionals in the application of these triage, minimizing errors, increasing accuracy, increasing speed and economically optimizing the process. The system is flexible regarding the different current structured triage systems and allows its total audit and integration into an existing system, such as a 'healthcare' CRM (Customer Relationship Management). We faced a scenario in which, in addition to having to provide a solution to this problem, we did not fully know the system in which it wants to be integrated and its infrastructure. This problem was solved using event-oriented architectures, based on microservices, allowing the implementation of distributed systems and the independence of operation and agnosticism to the triage service environment. As an additional deliverable, a web interface has been developed that allows the main functionalities of the triage service to be executed.

Índice general

Índice de figuras

Capítulo 1

Introducción

Gestionar la alta demanda de los servicios de urgencia hospitalaria es un proceso crítico a la hora garantizar los recursos sanitarios necesarios para los casos con mayor urgencia. Para ello se han desarrollado estrategias de clasificación de los demandantes del servicio, que reciben el nombre de *triaje* [1] o protocolo de Recepción, Acogida y Clasificación (RAC), como prefieren llamarlo en la Sociedad Española de Enfermería de Urgencias (SEEU) [2]. Este sistema de gestión del riesgo clínico se usa, con diferentes enfoques, en todo el mundo. En un primer momento, esta clasificación dependía directamente de la mezcla de experiencia e intuición de los facultativos más versados. Más tarde, para permitir la aplicación de estos triajes por parte de enfermeros y médicos menos experimentados, se desarrollaron algoritmos más formales, constituyendo un sistema que toma en cuenta la queja principal de la visita, el cuadro sintomático y las señales vitales con una estratificación en un número fijo de niveles de urgencia.

Más necesario aún es este sistema en los casos de pandemia, en los que los recursos son más demandados. Es por esto que los sistemas de información están siendo integrados en el sector sanitario permitiendo una gestión más eficaz y facilitando el escalado de los servicios de urgencia.

En este trabajo presentamos el desarrollo de un servicio informatizado para la aplicación de triajes. Concretamente, los triajes que se realizan de forma remota, como los telefónicos, que tienen diferencias con los triajes tradicionales *in situ* y que sirven

como primer filtro para el ahorro de recursos hospitalarios y la disminución del número de casos no urgentes derivados al servicio de urgencias.

1.1. Motivación

Un ejemplo de la informatización del servicio sanitario es el uso de CRMs (*Customer Relationship Management*). Salud Responde, un centro de información y servicios de salud del Servicio Andaluz de Salud (SAS), utiliza este tipo de sistema para asistir la atención a los ciudadanos que la demanden. La información y la asistencia se organiza en varios niveles de especialización y dependiendo de la dificultad de la consulta del ciudadano es atendida por diferentes perfiles de profesionales [3].

Ya sea en el ámbito telemático o no, para derivar el caso al nivel de atención que corresponda se aplican triajes. El triaje consta de varias preguntas dirigidas al paciente con el objetivo de conocer el cuadro sintomático y realizar la clasificación de éste según un algoritmo que toma en cuenta los distintos signos y síntomas en relación con la queja principal.

En el año 2009, con la pandemia de la gripe A, el sistema de Salud Responde se apoyó en un sistema de asistencia de aplicación del triaje de dicha enfermedad, cuyo resultado representa un caso de éxito: el personal técnico pudo dar respuesta a muchos más casos que habitualmente con un grado de confianza alto [4]. Sin embargo, el desarrollo de este sistema se limitó al triaje de la gripe A y no ha sido readaptado para otras situaciones de riesgo.

Para este tipo de procedimientos se requieren conocimientos médicos. Por lo tanto, aunque se disponga de un servicio telemático, el número de profesionales sanitarios para aplicar los triajes es un factor limitante que no permite escalar el servicio de forma óptima. Sin embargo, la posibilidad de realizar parcialmente el triaje por parte de profesionales no sanitarios disminuye el número de sanitarios necesarios para la realización del proceso.

1.2. Impacto

Los sistemas CRM están implantados en diferentes sectores, ya que son conocidos sus beneficios a la hora de gestionar el trato con el cliente [5] (en este contexto, los ciudadanos beneficiarios del Servicio Andaluz de Salud). Con la implantación del ser-

vicio de triaje que aquí presentamos para la extensión de un CRM existente, podemos conseguir los siguientes beneficios adicionales:

- Reducción del tiempo de respuesta.
- Clasificación de los casos más precisa.
- Minimización de errores de factor humano típico de las tareas repetitivas.
- Minimización de desplazamientos innecesarios. En este punto en concreto podemos identificar que el beneficio afecta a diferentes niveles:
  - A nivel clínico, ya que se evita el contacto de personas relativamente sanas con individuos con enfemedades contagiosas y disminuye la probabilidad de propagación;
  - A nivel social, porque favorece la opinión del ciudadano al obtener un servicio más cómodo y con mejor gestión de recursos;
  - Y a nivel económico ya que, como demuestran diversos informes [6, 7, 8, 9], la proporción de casos inadecuados en el servicio de urgencias oscila entre el 20 % y el 80 %. En 2017, el servicio de urgencias hospitalarias de Andalucía registró 4.310.874 casos en urgencias [10]. Si siendo conservadores, consideramos que el 30 % de esas consultas pueden haber sido inadecuadas y teniendo en cuenta que el precio medio de consulta de urgencias en Andalucía es de 250€ [11], más de 300.000.000€ estarían destinados inapropiadamente.
- Gestión más eficiente de los recursos del servicio de salud. El 31.7 % del presupuesto en Andalucía se destinó a Sanidad en 2018, lo que se traduce en 9.735.000.000€, unos 1.158€ por andaluz. Se incluye aquí la partida de Mutualismo Administrativo y otras partidas que no están directamente relacionadas con los recursos e infrastructura del servicio de salud, pero nos sirve para entender la magnitud de la responsabilidad que conlleva optimizar este proceso.

1.3. Objetivos

Se propone un producto software que sirva como apoyo a un servicio sanitario telemático, ya sea para el sector público o privado, integrable y agnóstico a la presencia

de un CRM u otro tipo de sistema cualquiera. Con este software se conseguirá automatizar, en cierto grado y con la cooperación de un trabajador sanitario o no sanitario, la aplicación de triajes para la clasificación de pacientes. Actualmente, este proceso depende de un facultativo, independientemente de si el servicio es telemático o presencial, y es un factor limitante en este tipo de tareas.

Este sistema será capaz de realizar una evaluación y, por consiguiente, una clasificación de un paciente en el nivel de emergencia correspondiente en función de la información recopilada durante la anamnesis[1] telefónica, haciendo uso de un conjunto de protocolos previamente almacenados. Con este objetivo, el programa permitirá cargar y almacenar nuevos protocolos de intervención para diferentes "quejas principales" o "motivos de llamada". Un requisito adicional es que el sistema sea auditable, de forma que queden registradas todas las acciones que se realicen o intenten realizarse sobre el sistema.

Desde el punto de vista técnico, se deben implementar estos requisitos en un servicio web con las funcionalidades CRUD (*Create, Read, Update, Delete*) sobre los algoritmos de triaje y los *endpoints* relacionados con las evaluaciones de un paciente.

[1] Proceso de recopilación de información por parte de un(a) especialista de la salud mediante preguntas específicas, formuladas bien al propio paciente o bien a otras personas relacionadas para obtener datos útiles, y elaborar información valiosa para formular el diagnóstico y tratar al paciente.

Capítulo 2

Estado del arte

En el desarrollo de soluciones software no se concibe la elaboración de un proyecto de calidad sin la colaboración de los expertos del dominio que se esta abordando durante el modelado y aceptación de las iteraciones a lo largo de todo el proceso. El enfoque de desarrollo software por excelencia para dominios y entornos de negocio complejos, el diseño guiado por el dominio (*Domain Driven Design*, DDD) [12], indica que el equipo de desarrollo y expertos de dominio deben cooperar para crear y madurar *lenguajes ubicuos* [13] manteniendo sucesivas reuniones, con el objetivo de elaborar un modelo de dominio rico [14] y evitar confusiones originadas por la ambigüedad y la subjetividad del lenguaje natural.

A falta de un experto de dominio en este proyecto, requerimos comprender y asimilar previamente por nosotros mismos las entrañas y las complicaciones del ámbito de los triajes. Hemos realizado una investigación exhaustiva en cuanto a metodologías para la clasificación de casos en urgencias por nivel de emergencia y en cuanto a las diferentes alternativas de herramientas informáticas que hay en el mercado que den, en cierto modo, soporte a este proceso médico.

2.1. La metodología del triaje

A día de hoy los protocolos de triaje se ejecutan en condiciones de "normalidad" debido a la masificación de los servicios de urgencia hospitalaria por factores como la utilización de estos servicios para la atención de situaciones no urgentes [15]. Sin embargo, su origen reside en las necesidades de controlar el flujo de demandantes en situaciones de emergencia extrema como en catástrofes (*Mass-Casualty Indicents*, MCI) y pandemias [16, 17, 18]. Más tarde, el triaje telefónico nacería para convertirse en el canal principal de solicitud de atención entre el paciente y la central que gestiona los recursos sanitarios.

Diferentes sistemas de triaje estructurado se han desarrollado y aplicado alrededor del mundo mostrando mayor eficacia con respecto a los triajes basados en la experiencia de los médicos [19] entre los cuales están; la *Australian Triage Scale* (ATS); la *Canadian Emergency Department Triage and Acuity Scale* (CTAS); el *Manchester Triage System* (MTS) [20]; el *Emergency Severity Index* (ESI); y el Sistema Español de Triaje (SET) adoptado por la Sociedad Española de Medicina de Emergencias (SEMES) a partir del *Model Andorrá de Triatge* (MAT). En España se han implementado principalmente el Sistema Español de Triaje y el Sistema de Triaje de Manchester junto con otros de implantación local [15]. Todos los nombrados son triajes de 5 niveles de urgencia:

- Nivel I: prioridad absoluta con atención inmediata y sin demora.
- Nivel II: situaciones muy urgentes de riesgo vital, inestabilidad o dolor muy intenso.
- Nivel III: urgente pero estable hemodinámicamente con potencial riesgo vital que probablemente exige pruebas diagnósticas y/o terapéuticas.
- Nivel IV: urgencia menor, potencialmente sin riesgo vital para el paciente.
- Nivel V: no urgencia. Poca complejidad en la patología o cuestiones administrativas, citaciones, etc.

En la figura 2.1 se muestra un ejemplo de algoritmo de clasificación en el Sistema de Triaje de Manchester. El algoritmo comienza con la búsqueda de alguno de los discriminantes del nivel de más urgencia (más arriba). Si se encuentra alguno de éstos, el algoritmo finaliza devolviendo el nivel asociado a ese discriminador (flecha hacia la

derecha). Si no se halla ninguno de los discriminantes, se continúa del mismo modo con el nivel inmediatamente inferior (flecha hacia abajo).

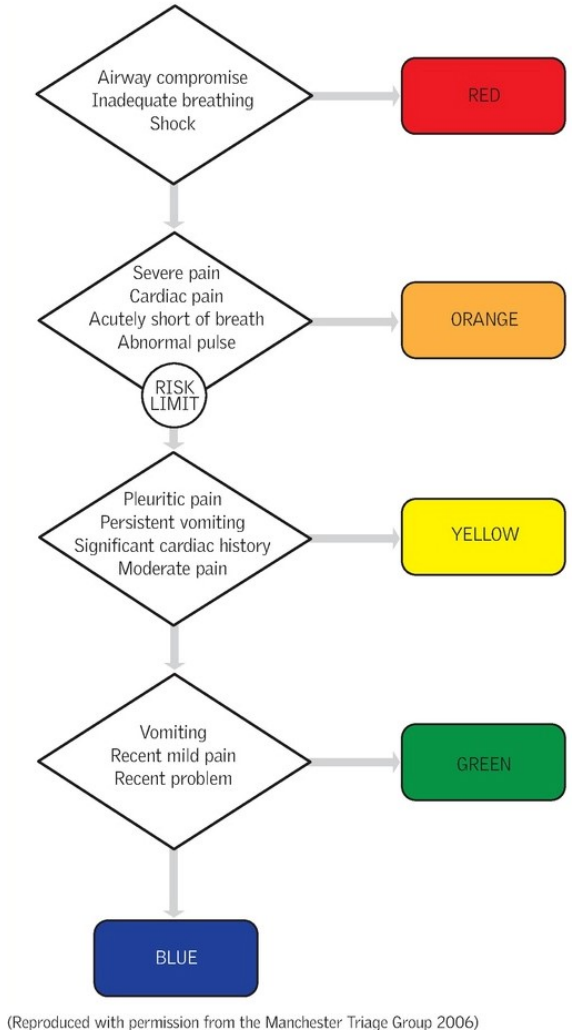

Figura 2.1 | **Ejemplo de algoritmo de triaje del Sistema de Triaje de Manchester.**

La situación de alerta de pandemia requiere un tipo de actuación especial enfocado a evitar la propagación de la misma [21] procurando que los pacientes sospechosos de contraer la enfermedad mantengan el mínimo contacto con los pacientes de enfermedades comunes. Además se desarrollan *triajes avanzados* que incluyen un protocolo de actuaciones validado por el equipo médico y de enfermería, como la realización de determinadas pruebas complementarias y acciones terapéuticas previas al diagnóstico. El triaje telefónico y los triajes de autoevaluación han demostrado funcionar eficazmente especialmente para evitar el desbordamiento de las instalaciones y el contagio en situaciones de pandemia, y han minimizado el número de traslados innecesarios [22, 23, 24, 25].

El SET (Sistema Español de Triaje, o Sistema Estructurado de Triaje) da soporte al

triaje telefónico y funciona de la siguiente forma:

1. Un teleoperador recepciona la llamada del paciente. La tarea del teleoperador es descartar los niveles de emergencia I y II. En el caso de que se den las condiciones para que se considere nivel I o II, se envía un transporte de emergencia y/o atención al paciente.
2. En el caso de que sean descartados los dos niveles más urgentes, la llamada se deriva a un enfermero o médico que valida el triaje del operador y profundiza en la anamnesia para la clasificación del paciente en los niveles III, IV o V.

Con la llegada de la informatización y las nuevas tecnologías en el sector sanitario se han desarrollado productos IT que dan soporte a este proceso, agilizándolo y llevándolo un paso más allá en cuanto a estructuración y minimización de subjetividad [26].

2.2. Productos y programas informáticos

El estado del arte en el ámbito de los productos de asistencia al triaje se concentra casi totalmente en el sector privado. Numerosos productos comerciales se ofrecen para dar soporte a los procesos de clasificación:

- web_e-PAT (Programa de Ayuda al Triaje del SET) [27]. Está implantado en más de 100 hospitales de España y Latinoamérica. Ofrece asistencia a triaje de adultos, triaje pediátrico, triaje telefónico y triaje embarcado para ambulancias, así como triajes para emergencias de incidentes de múltiples víctimas. web_e-PAT es propiedad de la empresa Treelogic.
- Infermedica [28]. Es una empresa que ha desarrollado una plataforma de soporte a servicios médicos como diagnóstico y triaje, además de otros más enfocados al desarrollador. Tienen productos listos para ser usados por los usuarios finales como un *chat-bot* para realizar un triaje o primer diagnóstico y un programa para dar soporte al triaje telefónico por parte de los enfermeros.
- ehCOS Triage de Everis [29]. Basado en el Sistema de Triaje de Manchester, pero no se especifica que dé soporte al triaje telefónico.
- En Estados Unidos existen numerosas empresas y productos como ClearTriage [30], TriageLogic myTriageChecklist [31], CAPITA [32], Keona Health [33] o

TeamHealth [34] que, o utilizan sistemas de triaje no estandarizados o no lo especifican.

Los productos comerciales no suelen tener una especificación técnica pública. El web_e-PAT sí que descubre algunas de las tecnologías de las que hace uso. Utiliza el estándar ICD-9 para clasificar las quejas principales en categorías sintomáticas. También hace uso de estándares de comunicación entre sistemas sanitarios como el de mensajería HL7. Infermedica hace uso del estándar SNOMED CT para mapear conceptos médicos al hacer uso de su ontología propietaria. SNOMED CT es especialmente útil para este tipo de mapeos y extracción de las quejas en textos no-estructurados [35, 36].

Hasta ahora, todas las estrategias utilizadas en los productos del mercado se basan en reglas: algoritmos en cascada haciendo uso de diferentes discriminantes[1] para la clasificación de los pacientes. Sin embargo, en el campo de la investigación están surgiendo nuevos planteamientos que ayudarían a aprovechar de manera más efectiva las bondades de la informática. Un enfoque novedoso y eficaz es usar tecnologías de la web semántica y la computación ubicua, ontologías y razonadores, como proponen Sánchez *et al.* (2011) [37] y demuestran Wunsch *et al.* (2017) [38]. San Pedro *et al.* (2004) [39] proponen un sistema móvil de asistencia a la decisión tomando en cuenta estrategias basadas en reglas, como las ya mencionadas, y otras basadas en *soft computing*, como la lógica difusa, útiles cuando la información que se obtiene del paciente es escasa o incompleta.

Abad-Grau *et al.* (2008) [40] ya revisaron los sistemas computacionales basados en métodos tradicionales de triaje e indicaron la necesidad de evolucionar hacia las tecnologías semánticas y abrieron paso a estrategias basadas en *machine learning*, todavía bajo un fuerte desarrollo, pero que viene progresando desde hace tiempo partiendo desde los modelos bayesianos [41] y explorando algoritmos más sofisticados como redes neuronales o máquinas de soporte vectorial de manera muy prometedora [42, 43, 44]. Sin embargo, Schenkel y Wears (2018) [45] han advertido de las limitaciones de estos métodos, sobre todo en cuanto a los datos de entrenamiento: si se utilizan las decisiones de un humano, el modelo aprenderá los sesgos y vicios que causan las imperfecciones de los sistemas actuales. Además de utilizar *machine learning* para predecir el nivel de emergencia de un paciente, se ha aprovechado para mejorar el proceso de anamnesis, elaborando herramientas para la sugerencia y el autocompletado de términos de las ontologías en la interfaz de usuario [46, 47] y

[1] Término utilizado en el ámbito de los triajes para hablar de signos, síntomas o hallazgos clínicos.

agentes conversacionales para proveer asistencia al triaje de auto-evaluación, como es el caso de Quro [48].

Capítulo 3

Metodología

Para llevar a cabo este proyecto se ejecutó una metodología “ágil” que, aunque no corresponda específicamente con alguna de las conocidas, se nutre de su esencia. El método consiste en hacer continuas iteraciones de tiempo variable (según los requisitos a implementar) con un máximo de dos semanas. En cada iteración se ejecutaron las distintas fases del proceso de desarrollo software (Análisis de requisitos; Diseño; Implementación). Se consideró como objetivo de la fase de prototipado el generar pequeños productos que muestren las funcionalidades desarrolladas en cada iteración. Al final de cada iteración, el autor del proyecto (el alumno) se reunió con los clientes (los tutores) para valorar el resultado obtenido y establecer los objetivos de la siguiente iteración.

En la fase inicial hubo una investigación exhaustiva del estado del arte en cuanto a los triajes informatizados; luego de conocerlos definimos los requisitos y los casos de uso que consideramos en aquel momento y, a continuación, comenzamos con la fase iterativa. Cada una o dos semanas definíamos los objetivos para este sprint, retocábamos los requisitos si era necesario con el feedback de la semana anterior, y usábamos la metodología de trabajo TDD (*test-driven development*) para la lógica de negocio, que consiste en escribir primero el test (en nuestro caso centrándonos en un caso de uso); después implementar la lógica necesaria para satisfacer el test; y, por último, refactorizar si es necesario. Luego integrábamos la lógica con la infraestructura, es decir, todo lo relacionado con la entrada-salida: el *framework* web, los gestores

de bases de datos, el gestor de colas, etc. Al final del sprint nos reuníamos para revisar los objetivos y definir los del siguiente sprint.

3.1. Fases de trabajo

A continuación se enumeran las fases de trabajo en las que consistió el Trabajo de Fin de Grado:

- Estudio inicial.
  - Investigación del estado del arte. Explorar proyectos, tecnologías y herramientas ya implementadas con el mismo objetivo, relacionado o que puedan ser útiles para nuestros proyectos para determinar qué se ha alcanzado y qué puntos de mejora y necesidad presenta el mercado. Llegar a conocer en profundidad los medios en los que se apoyará el software para llevar a cabo la solución. También profundizar en la materia de la ingeniería del software para conseguir un software mantenible, seguro y fiable.
  - Análisis de requisitos. Estudiar y comprender los principales requisitos que debe cumplir el producto del TFG y sus posibles puntos críticos. Especificar los casos de uso. Una pequeña parte del análisis de requisitos se realizó al inicio de cada iteración para la revisión e inclusión de nuevos requisitos.
- Diseño. Diseño lógico del software a implementar para cumplir los requisitos mencionados en el apartado anterior.
- Implementación. Llevar a cabo la implementación de los casos de uso acordados con el cliente de forma iterativa.
- Redacción de memorias y documentación. Describir el trabajo realizado y proveer información acerca de los resultados.
- Reuniones para la retroalimentación en las iteraciones. Como parte de la metodología de desarrollo software que se adoptó, se deben establecer citas para evaluar el curso del trabajo realizado después de una o varias iteraciones.

Capítulo 4

Análisis del problema

En esta sección se especifican los requisitos funcionales y no funcionales que han madurado durante la etapa inicial y las iteraciones de la fases de desarrollo y que describirían un producto mínimo viable, junto con los respectivos casos de uso en forma de texto y un diagrama UML de casos de uso. A continuación, se propone una solución a nivel conceptual que se amolda a dicha especificación.

4.1. Análisis de requisitos

Puesto que el proyecto no se desarrolló para un cliente definido, estos requisitos han sido determinados por nosotros y validados por los co-directores del trabajo de fin de grado.

4.1.1. Requisitos no funcionales

Los requisitos no funcionales del producto mínimo viable son los siguientes:

- El servicio será agnóstico a la presencia de otros servicios y sistemas.
- El servicio debe poder ser integrable en un sistema externo.
- El servicio debe ser accesible mediante un API HTTP.

- El servicio debe tener una disponibilidad cercana al 100 % de las veces en que un usuario intente accederlo.
- El servicio tendrá soporte multiplataforma.
- El servicio contará con un manual de usuario y documentación sobre las APIs.
- El servicio trabajará con información semánticamente interoperable.
- El software desarrollado será fácilmente mantenible y extensible.

4.1.2. Requisitos funcionales

Los requisitos funcionales del producto mínimo viable son los siguientes:

- El sistema debe gestionar distintos roles de usuario, entre los que estarán, como mínimo, teleoperador, facultativo y administrador.
- El administrador debe poder insertar nuevo algoritmo de triaje.
- El administrador debe poder actualizar un algoritmo de triaje.
- El administrador debe poder leer los algoritmos de triajes existentes.
- El administrador debe poder eliminar un algoritmo de triaje.
- El teleoperador debe poder descartar o confirmar que un paciente requiere asistencia médica urgente.
- El facultativo debe poder recibir y revisar el *pre-triaje* realizado por un teleoperador y obtener la clasificación del paciente en la escala de emergencia dada la información recopilada (discriminantes, cuadro sintomático, factores de riesgo).

4.1.3. Casos de uso

La especificación de los casos de uso se describe apoyándose en una hipotética interfaz de usuario para facilitar su definición. Aún así, casos de uso propios de una interfaz de usuario, como el de iniciar sesión, no se incluyen, ya que queda fuera de las responsabilidades del servicio web objetivo.

CU1 Un usuario administrador inserta un nuevo triaje.

Actor principal: Usuario con rol de administrador.

Participantes: Usuario administrador, teleoperadores, enfermeros/médicos reguladores, beneficiarios del servicio de antención sanitaria.

Sistema: Módulo de gestión de triajes.

Precondición: el usuario ha accedido a la aplicación, está autenticado en el sistema, tiene rol de administrador y se encuentra en la página principal de administrador.

Garantía mínima: se notificará al usuario si ocurre un error en el registro del nuevo triaje.

Garantía de éxito: el nuevo triaje quedará registrado en el sistema y se notificará al usuario el éxito del proceso.

Escenario principal de éxito:

1. El usuario selecciona la opción “insertar nuevo triaje”.
2. El sistema muestra un formulario de registro de un nuevo triaje.
3. El usuario completa correctamente el formulario.
4. El usuario selecciona la opción “enviar”.
5. El sistema muestra una notificación de “registro exitoso”.
6. El sistema muestra la página de detalle del nuevo triaje.

CU2 Un usuario administrador visualiza la página de detalle de un triaje existente.

Actor principal: Usuario con rol de administrador.

Participantes: Usuario administrador, teleoperadores, enfermeros/médicos reguladores, beneficiarios del servicio de antención sanitaria.

Sistema: Módulo de gestión de triajes.

Precondición: el usuario ha accedido a la aplicación, está autenticado en el sistema, tiene rol de administrador y se encuentra en la página principal de administrador.

Garantía mínima: se notificará al usuario en caso de que ocurra un error.

Garantía de éxito: el sistema mostrará la página de detalle del triaje.

Escenario principal de éxito:

1. El usuario selecciona la opción “ver detalles”de uno de los triajes del catálogo, haciendo uso o no del buscador.
2. El sistema muestra la página de detalle del triaje.

CU3 Un usuario administrador actualiza triaje existente.

Actor principal: Usuario con rol de administrador.

Participantes: Usuario administrador, teleoperadores, enfermeros/médicos reguladores, beneficiarios del servicio de antención sanitaria.

Sistema: Módulo de gestión de triajes.

Precondición: el usuario ha accedido a la aplicación, está autenticado en el sistema, tiene rol de administrador y ha realizado el **CU2** para el triaje que se desea actualizar.

Garantía mínima: se notificará al usuario en caso de que ocurra un error en la actualización.

Garantía de éxito: el sistema actualizará el triaje seleccionado y notificará al usuario del éxito del proceso.

Escenario principal de éxito:

1. El usuario edita los campos de la página de detalle a modo de formulario.
2. El usuario selecciona la opción “guardar”.
3. El sistema muestra una notificación de “actualización exitosa”.
4. El sistema muestra la página de detalle del triaje actualizado.

CU4 Un usuario administrador elimina un triaje existente.

Actor principal: Usuario con rol de administrador.

Participantes: Usuario administrador, teleoperadores, enfermeros/médicos reguladores, beneficiarios del servicio de antención sanitaria.

Sistema: Módulo de gestión de triajes.

Precondición: el usuario ha accedido a la aplicación, está autenticado en el sistema, tiene rol de administrador y ha realizado el **CU2** para el triaje que se desea eliminar.

Garantía mínima: se notificará al usuario en caso de que ocurra un error en la eliminación.

Garantía de éxito: el sistema eliminará el triaje seleccionado y notificará al usuario del éxito del proceso.

Escenario principal de éxito:

1. El usuario selecciona la opción de “eliminar”.
2. El sistema muestra un aviso pidiendo la confimación de la eliminación del triaje seleccionado.
3. El usuario confirma la eliminación.
4. El sistema muestra la notificación del “eliminación exitosa”.
5. El sistema muestra la página de catálogo de triajes.

CU5 Un usuario realiza un triaje de detección de situación crítica.

Actor principal: Usuario con rol de teleoperador.

Participantes: Usuario administrador, teleoperadores, enfermeros/médicos reguladores, beneficiarios del servicio de antención sanitaria.

Sistema: Módulo de aplicación de triajes.

Precondición: el usuario ha accedido a la aplicación, está autenticado en el sistema, tiene rol de teleoperador, enfermero-médico o administrador, se encuentra en la página principal y existe en la base de datos el triaje que se desea evaluar.

Garantía mínima: se notificará al usuario en caso de que ocurra un error en la evaluación.

Garantía de éxito: el sistema indicará si el paciente está en estado crítico o no y, en caso de no estarlo, encolará el caso para la posterior evaluación completa.

Escenario principal de éxito:

1. El usuario selecciona el algoritmo de triaje que desea evaluar identificado por la queja principal con ayuda del buscador y selecciona la opción “nuevo triaje parcial”.
2. El sistema muestra la página de asistencia para la evaluación del triaje.
3. El usuario recorre los discriminantes del algoritmo indicando si se han hallado o no en base a las respuestas del paciente.
4. El sistema muestra si el paciente se encuentra en estado crítico en base a las respuestas.
   4.1. El usuario finaliza la evaluación seleccionando la opción “confirmar”.

4.2. El usuario vuelve a la página de asistencia para la evaluación del triaje seleccionando la opción “volver”. Ir al paso 2.

5. El sistema detecta si el caso es no crítico y lo añade a la cola de casos pendientes de evaluación completa.
6. El sistema muestra la página principal.

CU6 Un usuario realiza un triaje completo.

Actor principal: Usuario con rol de profesional sanitario.

Participantes: Usuario administrador, teleoperadores, enfermeros/médicos reguladores, beneficiarios del servicio de antención sanitaria.

Sistema: Módulo de aplicación de triajes.

Precondición: el usuario ha accedido a la aplicación, está autenticado en el sistema, tiene rol de profesional sanitario o administrador, se encuentra en la página principal y existe en la base de datos el triaje que se desea evaluar.

Garantía mínima: se notificará al usuario en caso de que ocurra un error en la evaluación.

Garantía de éxito: el sistema indicará el nivel de urgencia del paciente.

Escenario principal de éxito:

1. El usuario selecciona el algoritmo de triaje que desea evaluar identificado por la queja principal con ayuda del buscador y selecciona la opción “nuevo triaje completo”.
2. El sistema muestra la página de asistencia para la evaluación del triaje.
3. El usuario recorre los discriminantes del algoritmo indicando si se han hallado o no en base a las respuestas del paciente.
4. El sistema muestra el nivel de urgencia asociado al paciente en base a las respuestas.
   4.1. El usuario finaliza la evaluación seleccionando la opción “confirmar”.
   4.2. El usuario vuelve a la página de asistencia para la evaluación del triaje seleccionando la opción “volver”. Ir al paso 2.
5. El sistema muestra la página principal.

CU7 Un usuario completa un caso no crítico pendiente.

Actor principal: Usuario con rol de profesional sanitario.

Participantes: Usuario administrador, teleoperadores, enfermeros/médicos reguladores, beneficiarios del servicio de antención sanitaria.

Sistema: Módulo de aplicación de triajes.

Precondición: el usuario ha accedido a la aplicación, está autenticado en el sistema, tiene rol de profesional sanitario o administrador, se encuentra en la página principal y existe al menos un caso pendiente.

Garantía mínima: se notificará al usuario en caso de que ocurra un error en la evaluación.

Garantía de éxito: el sistema indicará el nivel de urgencia del paciente.

Escenario principal de éxito:

1. El usuario selecciona la opción “evaluar un caso pendiente”.
2. El sistema muestra la página de asistencia para la evaluación del triaje y la información parcial recopilada en el proceso anterior.
3. El usuario recorre los discriminantes del algoritmo indicando si se han hallado o no en base a las respuestas del paciente.
4. El sistema muestra el nivel de urgencia asociado al paciente en base a las respuestas.
   4.1. El usuario finaliza la evaluación seleccionando la opción “confirmar”.
   4.2. El usuario vuelve a la página de asistencia para la evaluación del triaje seleccionando la opción “volver”. Ir al paso 2.
5. El sistema muestra la página principal.

4.1. Análisis de requisitos

Diagrama de casos de uso

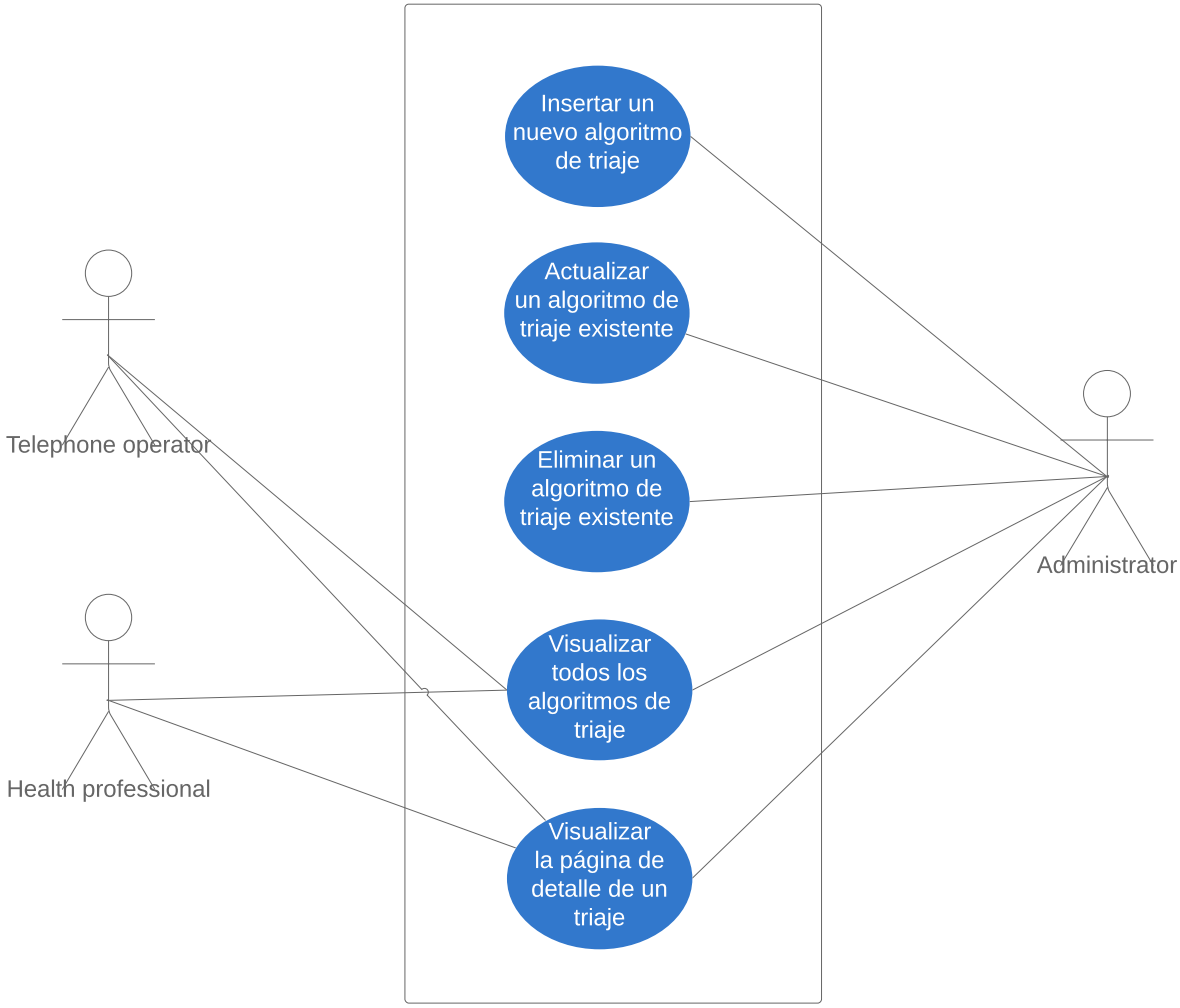

Figura 4.1 | **Diagrama de casos de uso (I).**

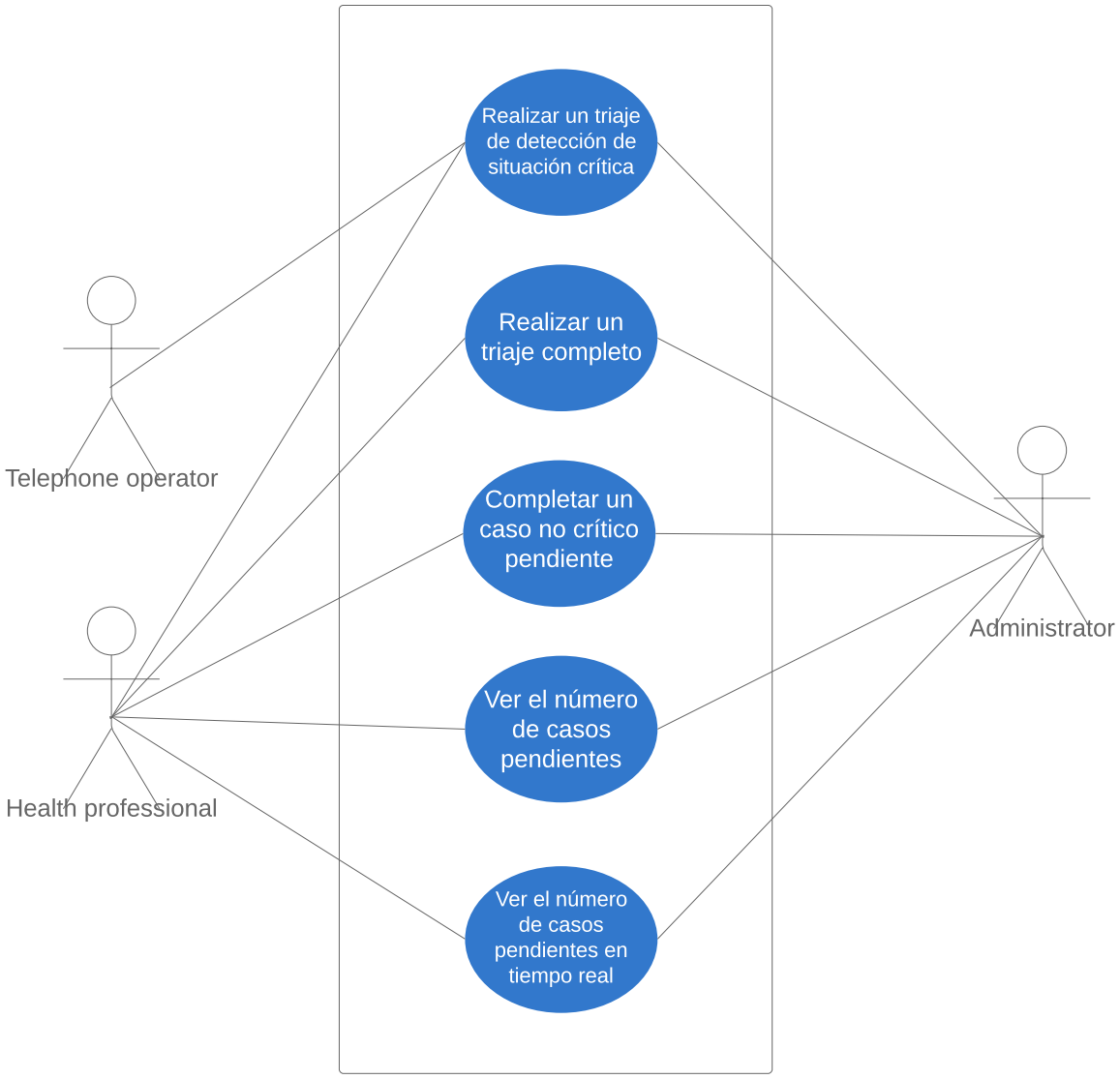

Figura 4.2 | **Diagrama de casos de uso (II).**

4.2. Solución propuesta

Aunque las nuevas estrategias que exploran el campo del aprendizaje automático y los metadatos semánticos y ontológicos demuestran un futuro prometedor, aún están bajo una fuerte investigación y desarrollo y queda camino para que tenga la suficiente madurez que aporte la confianza necesaria para ser usado en un entorno real. Dicho esto, el servicio de asistencia al triaje que proponemos se basa en clasificaciones basadas en reglas, algoritmos previamente validados por un equipo de expertos. En la sección del estado del arte se discute el gran número de estándares diferentes usados de manera relevante, además de el hecho de la variabilidad de la adopción y la convivencia de estos en un mismo entorno, como en hospitales de la misma comunidad autónoma o incluso de un mismo municipio [15, 49]. En esta solución no se aferra a ninguno de los nombrados específicamente y es una característica que se presenta como ventaja. Asimismo, como apoyo para el análisis, diseño e implementación de la solución, se ha tomado como referencia el Sistema de Triaje telefónico de Manchester dado que era el más accesible en el momento del desarrollo del proyecto.

El flujo de trabajo pensado para dar solución al problema tiene en cuenta a dos tipos de profesionales:

- Teleoperador auxiliar de regulación médica. Se trata de un profesional no médico pero entrenado que se encarga de recibir la llamada del paciente y de detectar, de la forma más temprana posible, si el paciente se encuentra entre los niveles críticos del triaje con la ayuda del software.
- Médico o enfermero regulador. Es el personal cualificado que se encarga de clasificar a los pacientes una vez que se han descartados los niveles de emergencia críticos por parte del teleoperador.

Este enfoque de sistema con teleoperadores como primer filtro se lleva a la práctica en el sistema de triaje telefónico del SET. Teniendo en cuenta estos dos tipos de usuarios del sistema, el flujo de trabajo consiste en los siguientes pasos:

- Para el teleoperador:
    1. El teleoperador recibe la llamada de un paciente y hace las preguntas pertinentes para identificar la queja principal que corresponderá a un algoritmo de triaje determinado.

2. El teleoperador formula las preguntas del algoritmo de triaje para determinar si el paciente se encuentra entre los niveles críticos de emergencia.
3. *a*) Si el teleoperador encuentra alguno de los discriminantes que indican que el paciente se ubica en alguno de los niveles críticos, envía el resultado del triaje y despacha los recursos necesarios para cubrir la emergencia, concluyendo el proceso.

   b) Si el teleoperador no encuentra ningún discriminante que indique una situación crítica del paciente, concluye su *pre-triaje* y despacha la solicitud de triaje por parte de un médico o enfermero regulador.

- Para el médico o enfermero regulador:

1. El médico solicita recibir una llamada de un paciente en cola del cual haya sido descartado una situación crítica.
2. El médico recibe la información del *pre-triaje* realizado por el teleoperador, en el que se incluye la queja principal seleccionada.
3. El médico realiza las preguntas pertinentes para confirmar o descartar la información recibida y ampliarla con el objetivo de completar el proceso de clasificación.

Se puede ver un esquema de este flujo de trabajo en la figura 4.3.

Figura 4.3 | **Flujo de trabajo en la atención telefónica.**

En todos los casos, la asignación del nivel de prioridad se lleva a cabo encontrando el discriminante aplicable más alto. A continuación se toman las decisiones sobre dónde debe ser visto el paciente y/o sobre el transporte que debe ser despachado,

dar los consejos pertinentes al paciente e indicar qué medidas tomar si la situación empeora.

Incluyendo nuestro servicio de triaje en este flujo de trabajo, este constaría de 5 participantes diferentes: el paciente que llama para solicitar la atención; el teleoperador de regulación médica, que es un profesional no sanitario pero entrenado para este trabajo; el médico o enfermero; nuestro servicio de triaje; y un usuario administrador. El teleoperador recibe la llamada del paciente y lo evalúa con la asistencia del servicio de triaje (que además va auditando los usos del mismo) de forma parcial, para encontrar si el paciente se encuentra entre los niveles por encima del límite de riesgo que veíamos antes. Si, en efecto, el paciente está en un estado en el que necesita atención urgente, el teleoperador despacha los recursos necesarios. Si el estado no es crítico, el servicio de triaje envía el caso a una cola, a la espera de que un profesional sanitario la reciba. Los médicos, del otro lado, van consumiendo los casos de la cola y los evalúan también con ayuda del servicio de triaje. Por otro lado, el administrador se encargaría de mantener los algoritmos de triaje, añadiendo nuevos, actualizándolos o eliminándolos. Así, se definen los 3 roles que va a tener el servicio: teleoperador, que sólo tendrá permisos para hacer triajes parciales, profesional sanitario, que podrá hacer triajes completos y parciales, y administrador, que podrá realizar cualquier tipo de triaje y cualquier operación sobre los triajes.

En la figura 4.4 se puede ver un esquema que describe el flujo de trabajo teniendo en cuenta nuestro servicio de triaje.

Aunque hemos tomado como sistema de referencia el Sistema de Triaje de Manchester, el sistema ha sido diseñado para dotarle de la máxima flexibilidad a la hora de tratar con triajes estructurados:

- Número de niveles de emergencia no fijado.
- Número de discriminantes por nivel no fijado.
- Número de consejos por nivel no fijado.
- Capacidad de incluir preguntas en lenguaje natural (cero o más para cada discriminante) con el objetivo de asistir en el proceso de anamnesis al usuario que aplica el triaje.
- Capacidad de incluir la definición de cada uno de los discriminantes.
- Nomenclatura de los diferentes niveles de emergencia a disposición del usuario.

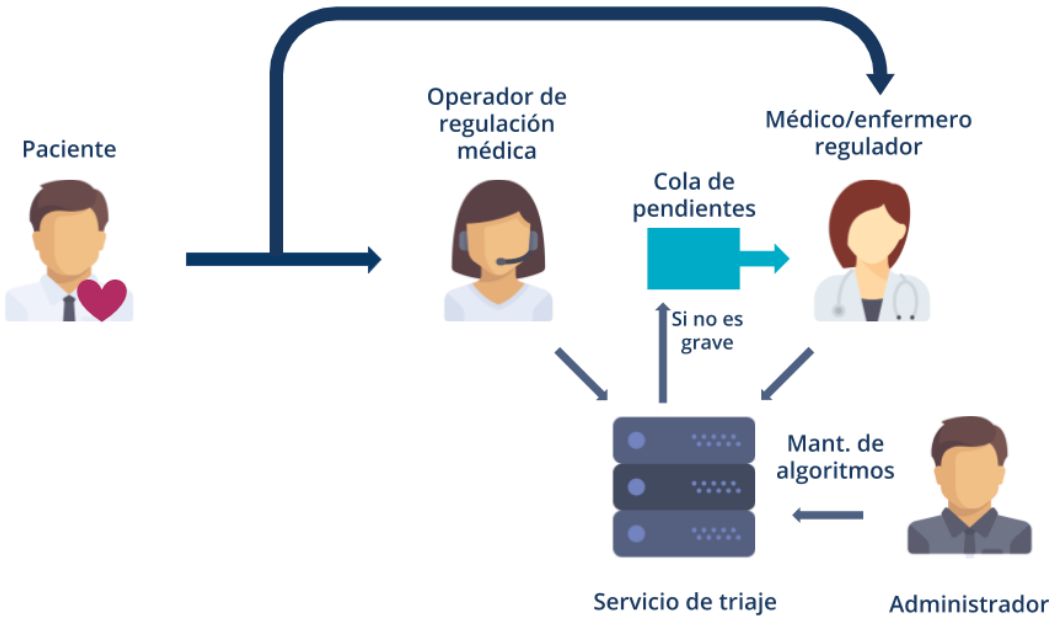

Figura 4.4 | **Flujo de trabajo en la atención telefónica haciendo uso del servicio de triaje.**

Algo que todos los triajes telefónicos deben incluir es una queja, signo o síntoma principal que identificará el algoritmo en cuestión. Algunos ejemplos son *abcesos o infecciones locales*, *dolor de espalda*, *llanto de bebé* y, como un caso especial, *petición de medicamento*.

Es responsabilidad del sistema de triaje utilizado garantizar la cobertura de todas las quejas y posibles situaciones que se puedan dar y la consistencia en sus evaluaciones, ya que es común que los signos principales reportados por un paciente puedan corresponderse con más de una *queja principal* que identifican a diferentes algoritmos de triaje. En este sentido, sea cual sea el algoritmo elegido entre los aplicables, el resultado de la clasificación debería ser el mismo.

Aunque no entraba dentro de los objetivos iniciales de este proyecto, se ha desarrollado adicionalmente un cliente web sencillo del servicio de asistencia al triaje, que sirve como ejemplo de consumidor de la API HTTP y nos permitirá demostrar de forma más visual el funcionamiento de ésta. Para este nuevo cliente web, hemos añadido dos nuevos casos de uso que corresponden con la visualización síncrona y en tiempo real del número total de casos pendientes por ser completamente resueltos.

4.2. Solución propuesta

Capítulo 5

Diseño e implementación

5.1. Arquitectura software

En esta sección se describe el enfoque que se ha llevado a cabo en cuanto a disposición, modularización, aislamiento y comunicación entre los componentes del sistema.

5.1.1. Arquitectura a nivel de macrodiseño

En esta primera subsección se describe el diseño de la capa de más alto nivel. Concretamente se esclarece cuales serán los distintos servicios dentro del sistema sanitario y la comunicación entre ellos.

La estrategia a nivel de arquitectura es el aspecto técnico de mayor importancia de este proyecto. Nosotros asumimos el sistema de telemedicina existente como un sistema completo que actuará como consumidor de nuestro sistema de asistencia al triaje, que presentamos como otro microservicio independiente. De este modo, las únicas responsabilidades que tiene el microservicio de triaje son las de gestión de protocolos y la evaluación de estos. Es decir, casos de uso como el de autenticación o el de auditoría prevalece únicamente en el sistema externo, donde realmente pertenece, a diferencia del enfoque de mantener una gestión usuarios y roles en el caso de la au-

tenticación y autorización, lo que conllevaría la replicación de los ya existentes en el sistema externo, o mantener la información *fuera* del sistema externo en una base de datos propia en el caso de la auditoría. En la figura 5.1 se muestra la arquitectura que asumimos que tendrá el sistema una vez completa la integración y el despliegue.

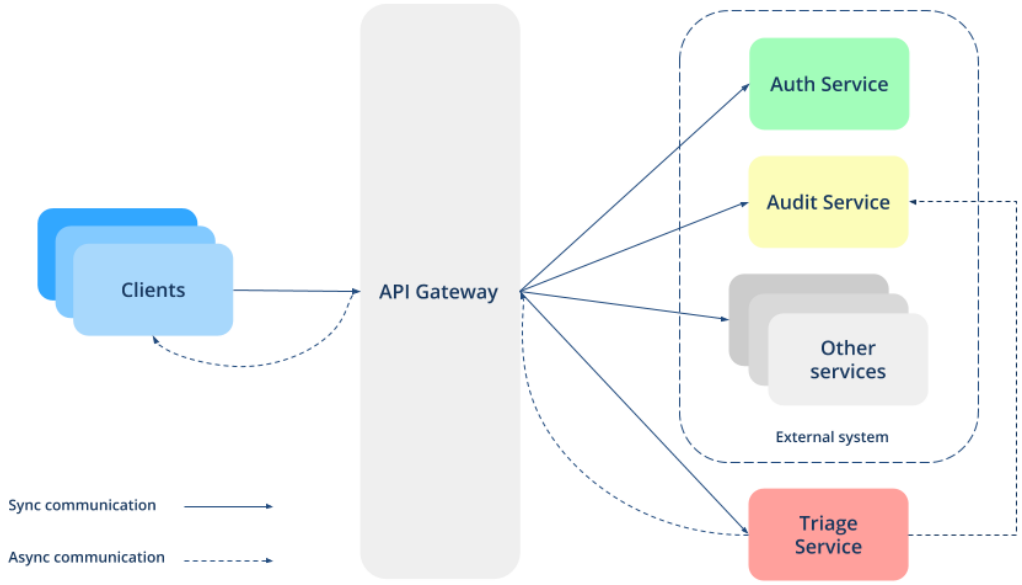

Figura 5.1 | **La arquitectura del sistema propuesta.**

Así, la seguridad del servicio de triaje, es decir, la autorización de uso de los *endpoints*, se llevará a cabo recibiendo desde el cliente la información mínima necesaria para validar esta autorización. Al ser un sistema basado en microservicios y al caber la posibilidad de usar una arquitectura distribuida, se debe evitar la gestión del estado de la aplicación en la medida de lo posible. Una aproximación a este problema es manejar la autenticación y autorización haciendo uso de *tokens*. En este *token* será provisto por un servicio de autenticación del sistema existente y contendrá toda la información que necesita el servicio de triaje para conocer si tiene autorización para ejecutar un caso de uso determinado, así como información más general, como el tiempo de expiración del *token*. Toda la información debe cifrarse usando un algoritmo criptográfico y una clave secreta que deben conocer ambos servicios, de forma que cuando el servicio de triaje recibe una petición con el *token*, éste puede descifrarla y validar la información que contiene. Una solución posible es usar el estándar JSON Web Token. En el diagrama de secuencia 5.2 se muestra cómo es el flujo de trabajo para hacer una petición a un *endpoint* protegido con autenticación. La petición al servicio de autenticación del sistema externo se realizará únicamente cuando el usuario usa la aplicación por

primera vez o cuando el *token* se haya expirado.

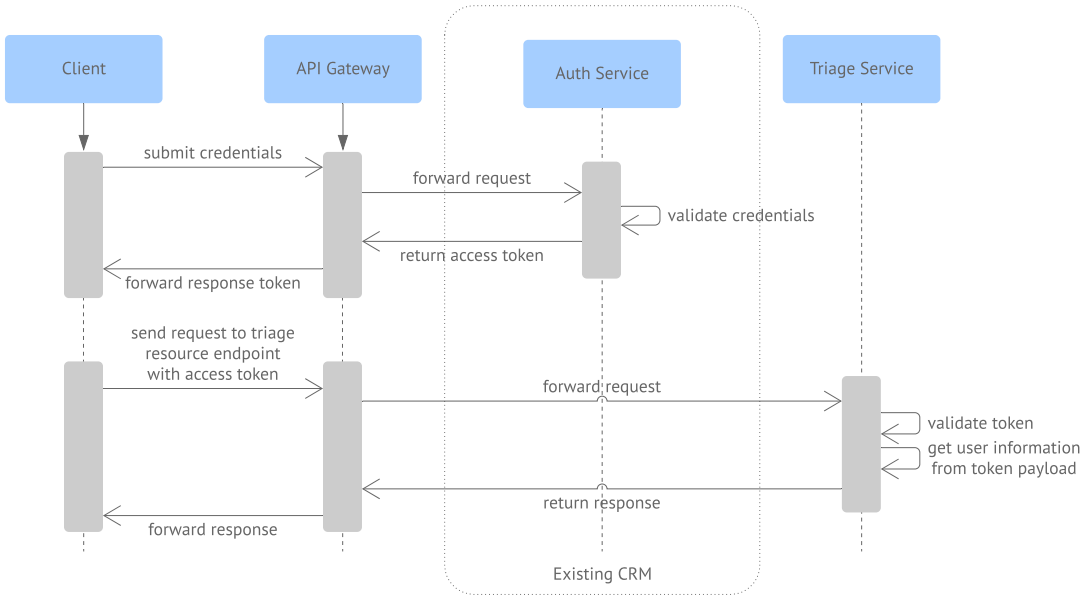

Figura 5.2 | **Diagrama de secuencia de una petición a un *endpoint* protegido del servicio de triaje.**

Para el caso de la auditoría, se aprovecha la estrategia de las arquitecturas orientadas a eventos que se usan extensamente en entornos distribuidos. Nuestro sistema publicará eventos a los cuales se puede suscribir cualquier otro componente sin tener que extender el servicio de triaje. Estos eventos pueden publicarse a un sistema de colas, como RabbitMQ, y, dado que es un proceso asíncrono, en algún momento un *worker* que esté suscrito a éstos lo consume y lo envía al servicio de auditoría del sistema externo para su almacenaje. En nuestro caso, hemos considerado que son eventos de interés para la auditoría el haber realizado un triaje parcial o el haber realizado un triaje completo.

En definitiva, lo que conseguimos con esta arquitectura es que nuestro servicio esté totalmente desacoplado y sea agnóstico a los distintos servicios que puedan acompañarle.

5.1.2. Arquitectura a nivel de microdiseño

El servicio web se ha desarrollado adoptando el modelo de arquitectura hexagonal, también conocido como arquitectura de puertos y adaptadores, arquitectura de capas de cebolla o arquitectura limpia. Usando este tipo de arquitectura, podemos crear apli-

caciones con muy poco acoplamiento e independiente de detalles de implementación, como es el caso de todo lo relacionado con entrada/salida (por ejemplo, el sistema de persistencia, o las interacciones con plataforma o sistema operativo) o el uso de *frameworks*. Las aplicaciones resultan fácilmente testeables, extensibles y mantenibles. Consiste en separar y agrupar los componentes de la aplicación en varias capas "concéntricas" según el nivel de abstracción de su lógica, por lo que nos hace cumplir ya en cierto grado el principio de responsabilidad única. Podemos reducir estas capas a las de dominio, aplicación e infraestructura. El dominio se encuentra en el corazón del programa y por encima de él se encuentran la capa de aplicación y la de infraestructura, en ese orden.

En la figura 5.3 se puede ver un esquema explicativo de la arquitectura limpia de Robert C. Martin. A la izquierda, la capa amarilla constituye la capa de dominio, la capa roja constituye la capa de aplicación y las capas verde y azul constituyen la capa de infraestructura. A la derecha se clarifica cómo se satisface el principio de inversión de dependencias y el flujo de control.

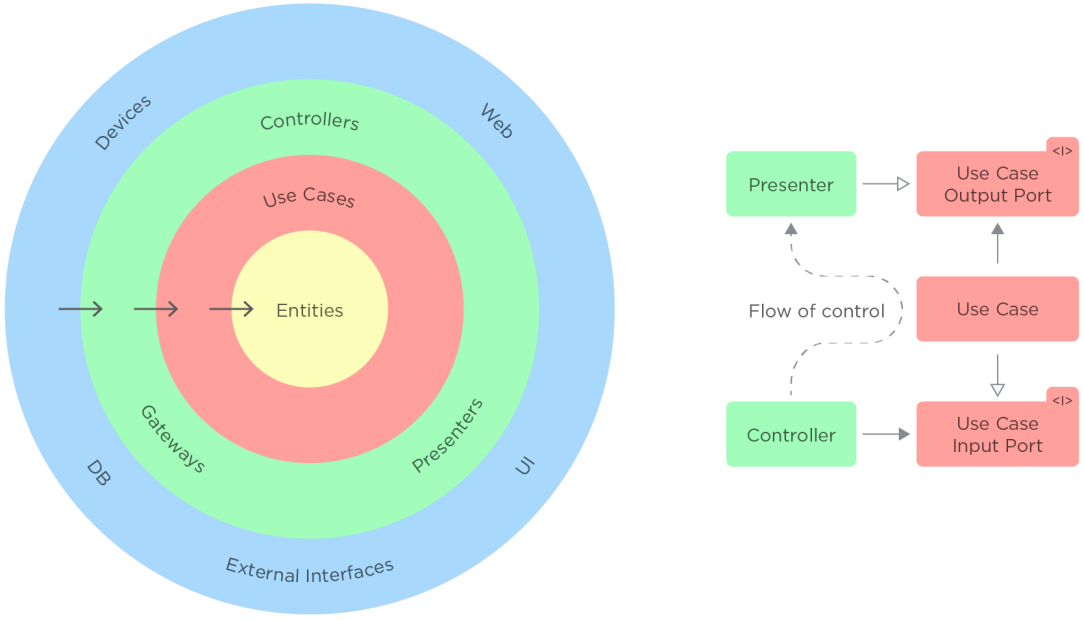

Figura 5.3 | **La arquitectura limpia.**

En la capa de dominio se encuentra la lógica de negocio, con los tipos y las entidades que modelan el dominio real, libre de referencias a las tecnologías, así como los repositorios, los eventos de dominio, los despachadores de eventos y sus respectivos suscriptores. La capa de aplicación alberga lógica ajena al dominio pero sin tener ninguna dependencia sobre la infraestructura, como las representaciones de las en-

tidades de dominio en datos planos y sus respectivos mapeadores y los casos de uso o *application services*, que son clases que contienen la lógica que se encarga de orquestar el comportamiento del dominio en cada uno de los casos de uso de la aplicación. Por último, en la capa más externa, se encuentra el código que de alguna forma se acopla a alguna tecnología ajena a la aplicación, como los controladores del *framework* web, las implementaciones de los repositorios o código con interacción con el sistema de ficheros. Estas capas tienen una estricta regla de dependencia de fuera hacia dentro, es decir, la capa dominio no puede depender de ningún componente de otra capa, la capa de aplicación solo puede depender de la capa de dominio y la capa de infraestructura no tiene restricción. Para asegurar esto, se cumple el principio de inversión de dependencias y, cuando capas más internas interactúan con lógica de más bajo nivel, lo hacen a través de interfaces implementadas por capas más externas. Esto es, las capas internas *exponen puertos* y las capas externas *proveen sus adaptadores*. Un ejemplo es la persistencia de los datos: el dominio hace uso de repositorios conociendo únicamente su API, y en la capa de infraestructura se encontrarán los adaptadores que implementen esa API, que podrían ser una implementación usando MongoDB o una implementación manteniendo una tabla *hash* en memoria siendo ambas totalmente intercambiables, cumpliendo con el principio de sustitución de Liskov. Este enfoque de puerto-adaptador se basa directamente en el principio de segregación de interfaces y permite, en última instancia, cumplir con el principio abierto-cerrado.

Además de el aislamiento del microservicio de triaje con respecto a su entorno, existe también una separación de la lógica en el interior de éste en base a la responsabilidad o interés desde el punto de vista de negocio. Concretamente, se separa la lógica de la evaluación de los triajes y su gestión de la lógica relacionada con la identidad y el acceso. Es por esto que existen los directorios `identity-access` y `triage-evaluation`. Estos dos entornos distintos reciben el nombre de *bounded contexts* o contextos limitados, según describe la metodología del diseño guiado por el dominio. Con esto conseguimos mayor mantenibilidad e independencia entre los dos contextos distintos.

Los dos contextos limitados nombrados, junto al contexto `shared`, que contiene lógica reutilizable en todos los contextos y el directorio `applications`, que contiene la verdadera API HTTP, son proyectos que forman parte de la construcción final de Gradle.

Ciclo de vida de la petición

Para dar una buena visión general de alto nivel de cómo funciona la aplicación respecto a los distintos componentes que se han nombrado en esta sección, se explica a continuación el flujo de control durante el ciclo de vida de una petición HTTP.

Tradicionalmente, cuando llega una nueva petición, ésta pasa por los filtros del *framework* que haya configurados. Nosotros, para tener mayor control sobre la lógica que se ejecuta entorno a los casos de uso (o *application services*) y tener mayor capacidad de reutilización al no acoplarnos al *framework*, implementamos una serie de *proxies*/decoradores que envolverán a las instancias de los *application services* en lugar de usar estos filtros del *framework*.

En el siguiente segmento de código se puede ver la lógica del controlador que se ejecuta en el *endpoint* `POST - /triage`, que se corresponde con el caso de uso ïnsertar nuevo algoritmo triaje".

```
@PostMapping
public ResponseEntity insertTriage(
    @RequestBody final CreateTriageRequest request,
    @RequestBody String body,
    @RequestHeader("Authorization") String authHeader
) {
    new EventPublisherResetApplicationService<>(
        new SecuredApplicationService<>(
            new TriageJSONSchemaValidationApplicationService<>(
                new CreateTriage(repository, mapper),
                jsonSchema,
                body
            ),
            authService,
            JWTAuthorizationData.fromAuthorizationHeaderString(
                authHeader,
                CreateTriage.class.getName(),
                body
            )
        ),
        eventStore,
        queue,
        rabbitMQChannel
    ).execute(request);
```

```
    return ResponseEntity.ok().build();
}
```

Podemos ver que el *application service* principal que representa el caso de uso, `CreateTriage`, está siendo "extendido" por tres decoradores. `EventPublisherResetApplicationServ` se encarga de resetear el *event publisher* y configurar los suscriptores que van a escuchar los eventos que se disparen durante la petición, tal y como se ve en el siguiente segmento de código.

```
public class EventPublisherResetApplicationService<S, T extends
    ApplicationRequest> implements ApplicationService<S, T> {
    private ApplicationService<S, T> proxiedApplicationService;
    private EventStore eventStore;
    private PendingTriagesQueue queue;
    private Channel rabbitMQChannel;
    public EventPublisherResetApplicationService(
        ApplicationService<S, T> proxiedApplicationService,
        EventStore eventStore,
        PendingTriagesQueue queue,
        Channel rabbitMQChannel
    ) {
        this.proxiedApplicationService = proxiedApplicationService;
        this.eventStore = eventStore;
        this.queue = queue;
        this.rabbitMQChannel = rabbitMQChannel;
    }
    @Override
    public S execute(T request) {
        EventPublisher.instance().reset();
        EventPublisher.instance().subscribe(new
            AuthorizationCheckedSubscriber(eventStore));
        EventPublisher.instance().subscribe(new
            AuditingCriticalCheckTriageAssessedSubscriber(eventStore));
        EventPublisher.instance().subscribe(new
            AuditingFullTriageAssessmentSubscriber(eventStore));
        EventPublisher.instance().subscribe(new
            RabbitMQQueueSizeChangedNotificationSubscriber(rabbitMQChannel,
            queue));
        return proxiedApplicationService.execute(request);
```

```
    }
}
```

Después, se ejecuta la lógica del decorador `SecuredApplicationService`, que se encargará de autorizar al usuario la ejecución del caso de uso según sus permisos. En este caso, se le inyecta al decorador el puerto `AuthorizationService`. En el entorno de producción real, un adaptador debe ser implementado haciendo uso de un proveedor externo, como otro microservicio, o incluyendo toda la lógica necesaria en el mismo adaptador. En este caso, hemos simulado el servicio de autorización con una implementación *dummy*.

El tercer y último decorador, `TriageJSONSchemaValidationApplicationService`, se encarga de validar el *payload* de la petición en formato JSON, siguiendo el esquema que se encuentra en el repositorio (`applications/main/resources/triage-json-schema.json`).

Los *application services* y sus decoradores deben conocer únicamente interfaces de dominio o componentes de la capa de aplicación para cumplir la regla de dependencia. En nuestro caso se cumple para todos los *application services* principales, aunque para algunas dependencias de los decoradores se ha roto la regla para agilizar el desarrollo. Como se explicará en el capítulo 7, en las conclusiones, este tipo de arquitectura conlleva ingeniería excesiva para dominios poco cambiantes y sencillos.

En cualquier caso, todas las dependencias de los casos de uso y sus decoradores son inyectadas previamente en el controlador haciendo uso del potente contenedor de dependencias del *framework*.

Siguiendo con el ciclo de vida de la petición, una vez llegados a la ejecución del *application service* principal, éste normalmente necesita recuperar el estado desde algún sistema de persistencia. Lo que conoce el caso de uso es una interfaz de dominio que representa esta responsabilidad, que tiene una implementación en MongoDB. En el siguiente segmento de código se muestra la lógica del caso de uso de evaluar el estado crítico del paciente, es decir, realizar un triaje parcial.

```
public CriticalCheckAssesmentOutput execute(
    CheckForCriticalStateRequest request
) {
    ChiefComplaint selectedChiefComplaint = new ChiefComplaint(
        new ClinicalFindingId(request.triageChiefComplaintId), null);
    Triage selectedTriage = repository.find(selectedChiefComplaint);
    if (null == selectedTriage) {
```

```
            throw
                TriageDoesNotExistException.withChiefComplaint(selectedChiefComplaint);
        }
        CriticalCheckAssesmentOutput output = selectedTriage.checkForCriticalState(
            request.findingIds.parallelStream().map(fStr -> new
                ClinicalFinding(new ClinicalFindingId(fStr),
                null)).collect(Collectors.toList())
        );
        if (!output.hasCriticalState) {
            queue.enqueue(
                CriticalCheckTriageAssessed.create(selectedTriage.chiefComplaint(),
                output));
            EventPublisher.instance().publish(
                new PendingCasesQueueSizeChanged());
        }
        return output;
    }
```

Una vez que el caso de uso consigue las entidades necesarias, ejecuta las acciones pertinentes, que disparan los eventos de dominio correspondientes, de forma que los *subscribers* saben cuándo tienen que ejecutarse. En el caso del segmento de código anterior, si el estado resulta ser no crítico, se publica el evento `PendingCasesQueueSizeChanged`, cuyo *subscriber* es `RabbitMQQueueSizeChangedNotificationSubscriber`. Este *subscriber* se encargará de publicar al *topic* `queue-size` de RabbitMQ, que es el canal de notificación en tiempo real del número de casos pendientes. Se puede ver la lógica descrita en el siguiente segmento de código.

```
public class RabbitMQQueueSizeChangedNotificationSubscriber implements
    EventSubscriber<PendingCasesQueueSizeChanged> {
    private final Channel rabbitMQClient;
    private final PendingTriagesQueue queue;
    public RabbitMQQueueSizeChangedNotificationSubscriber(Channel
        rabbitMQClient, PendingTriagesQueue queue) {
        this.rabbitMQClient = rabbitMQClient;
        this.queue = queue;
    }
    @Override
    public void handleEvent(PendingCasesQueueSizeChanged anEvent) {
```

```
        long queueSize = queue.numberOfEnqueuedCases();
        try {
            rabbitMQClient.basicPublish(
                "amq.topic",
                "queue-size",
                null,
                Long.toString(queueSize).getBytes());
        } catch (IOException e) {
            throw new RuntimeException("Couldn't publish queue size
                notification.");
        }
    }
    @Override
    public Class<PendingCasesQueueSizeChanged> subscribedToEventType() {
        return PendingCasesQueueSizeChanged.class;
    }
}
```

Al final de la ejecución del caso de uso, si fuera necesario, el mapeador de los datos transformaría las entidades en objetos planos de transporte de datos (*DTO*, *data transfer object*) y éstos serían retornados. Puede verse, como ejemplo, el siguiente segmento de código perteneciente al caso de uso de la evaluación del triaje completo.

```
AlgorithmLevel output = selectedTriage.fullyAssess(
    request.findingIds.parallelStream().map(fStr -> new ClinicalFinding(new
        ClinicalFindingId(fStr), null))
    .collect(Collectors.toList()),
    prevAssesment
);
return mapper.mapAlgorithmLevel(output);
```

Por último, estos datos los utilizaría el presentador de *framework* para representarlo en JSON o el formato que fuere.

En la figura 5.4, podemos ver que efectivamente se cumple la regla de dependencia: todas las flechas quedan en la misma capa o alguna interior.

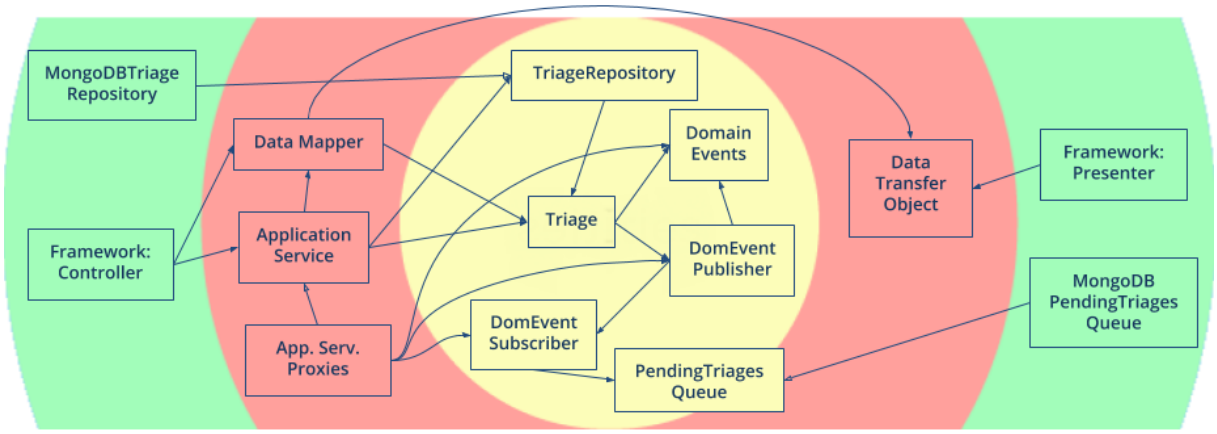

Figura 5.4 | **Reglas de dependencias entre las diferentes capas de la aplicación.**

5.2. Interfaz de usuario

Como ejemplo de consumidor del servicio de triaje, hemos implementado una interfaz web que utiliza todos los casos de uso disponibles en el *back-end*. Esta interfaz web está desarrollada en VueJS como una SPA (*single page application*), que evita recargas de la página cuando se cambia de vista aprovechando la ténica de desarrollo AJAX (*Asynchronous Javascript and XML*).

Los estilos de la interfaz se han desarrollado sin hacer ningún uso de *frameworks* o preprocesadores de CSS.

Con el *framework* VueJS desarrollamos basándonos en componentes, que son instancias reusables del propio Vue con un nombre concreto que usamos como etiqueta. Por ejemplo:

```
<TriageProcess
:selectedTriage="selectedTriage"
:step="startingStep"
@confirmed="onConfirmation"
></TriageProcess>
```

Es un paradigma muy similar a la nueva especificación de Web Components de la w3c.

A continuación se detallan las principales características del cliente web.

5.2.1. Acceso

La autorización funciona de forma simulada en el *backend* esperando tres posibles *tokens* que corresponden a los tres roles diferentes de la aplicación: `token-admin`, `token-health-professional` y `token-telephone-operator`. Por otra parte, en el cliente se simula la autenticación, interceptando las llamadas a cierto *endpoint*. En nuestro caso, existen tres usuarios distintos con mismo nombre de usuario y contraseña: `admin`, `healthprofessional` y `telephoneoperator`. Con cada una de las credenciales, el cliente web almacenará el respectivo token de acceso, antes nombrado, que habrá "devuelto" la llamada.

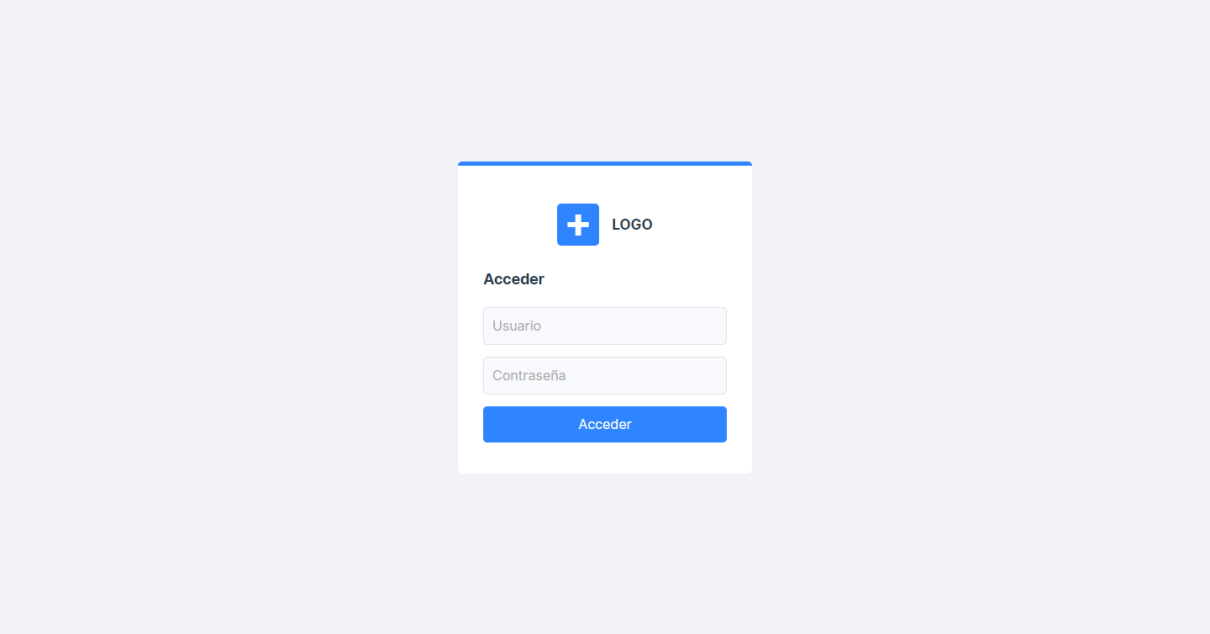

Figura 5.5 | **Pantalla de acceso.**

5.2.2. Pantalla principal del administrador

En la figura 5.6 se puede ver la pantalla principal del administrador. Es posible realizar una evaluación de un caso pendiente, o iniciar un nuevo triaje parcial o un nuevo triaje completo. Además, hay un indicador del número de casos en cola en tiempo real, haciendo uso de la comunicación vía WebSockets.

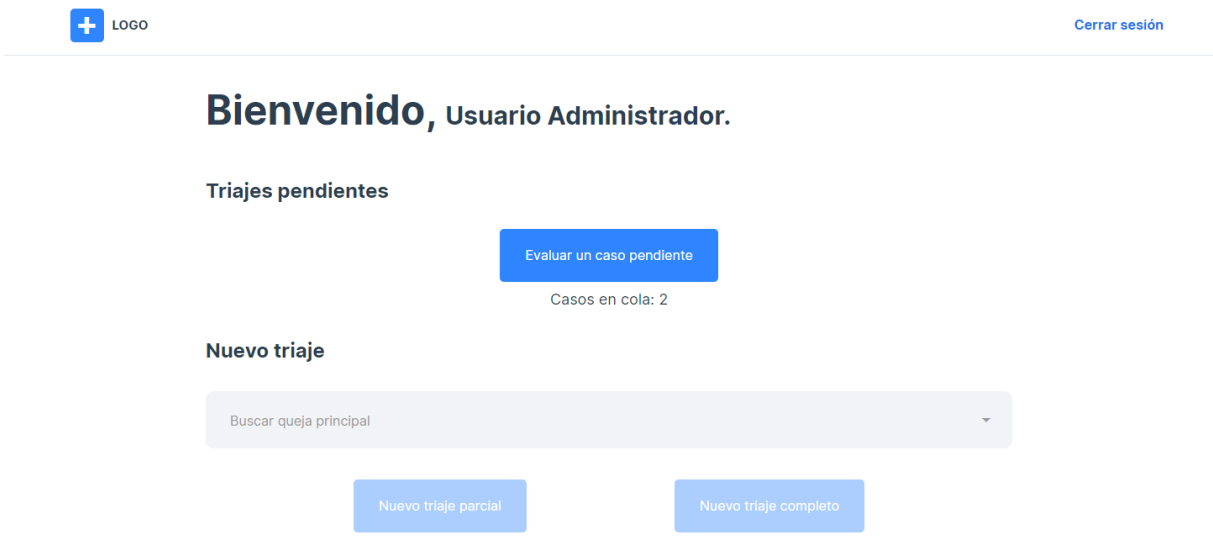

Figura 5.6 | **Pantalla principal del administrador.**

5.2.3. Pantalla principal del profesional sanitario

En la figura 5.7 se puede ver la pantalla principal del profesional sanitario. A diferencia del administrador, los usuarios con este rol no podrán realizar un triaje parcial.

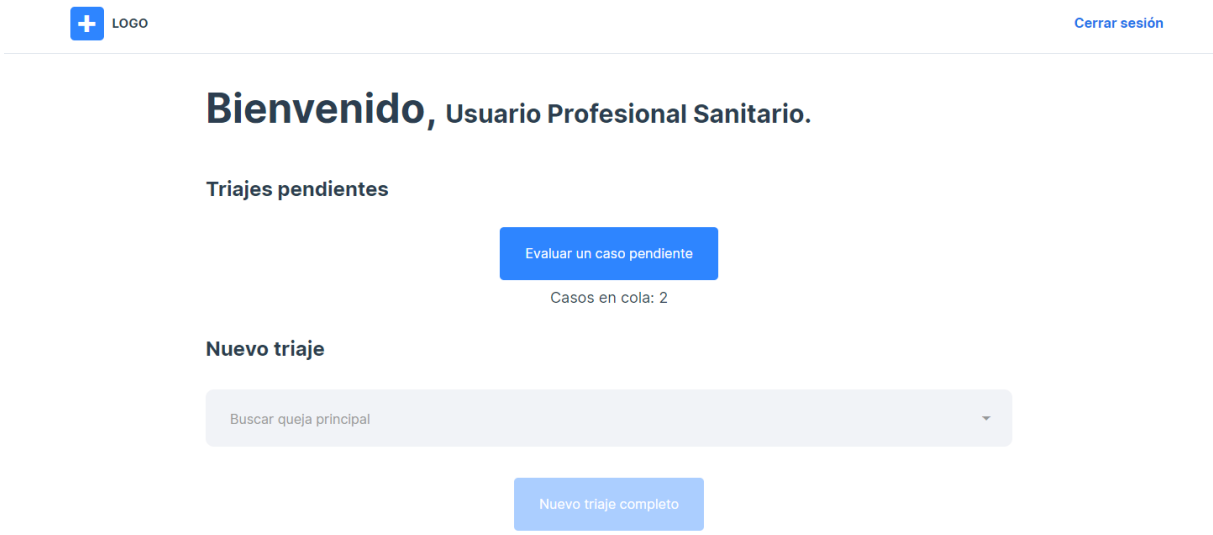

Figura 5.7 | **Pantalla principal del profesional sanitario.**

5.2.4. Pantalla principal del teleoperador

En la figura 5.8 se puede ver la pantalla principal del teleoperador. Los usuarios con este rol solo tienen permitido realizar triajes parciales.

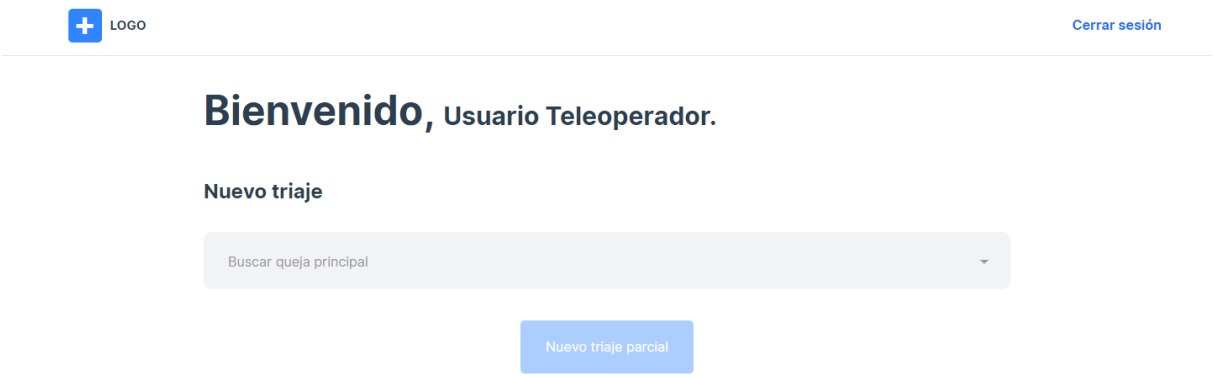

Figura 5.8 | **Pantalla principal del teleoperador.**

5.2.5. Buscador

Un componente en común para todos los usuarios es el buscador de la queja principal de la llamada, que es identificador de cada algoritmo de triaje. En este caso, se cargan todos los triajes almacenados para hacer una búsqueda en memoria en el propio cliente, puesto que el número de algoritmos de triaje es reducido en un sistema bien diseñado.

Una vez seleccionado la queja principal y, por ende, el algoritmo de triaje que se quiere aplicar, las opciones de ejecutar un nuevo triaje parcial o completo son seleccionables, como muestra la figura 5.10.

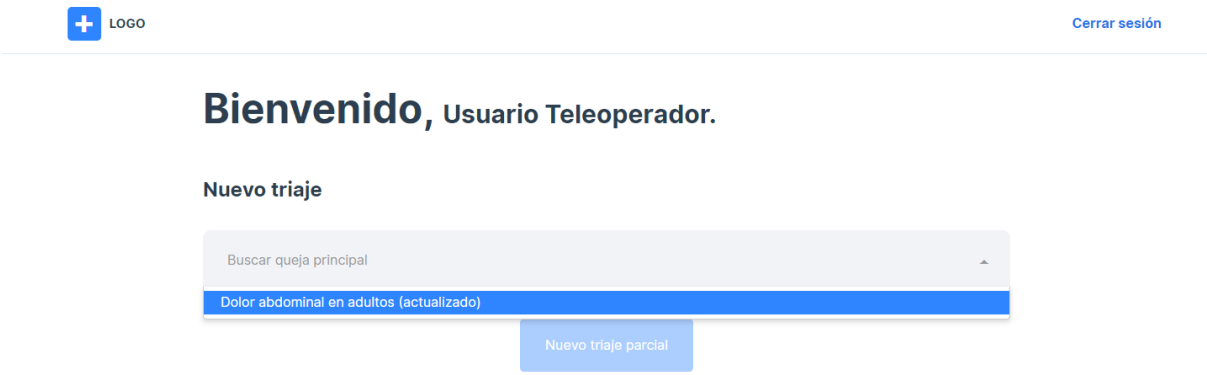

Figura 5.9 | **Buscador.**

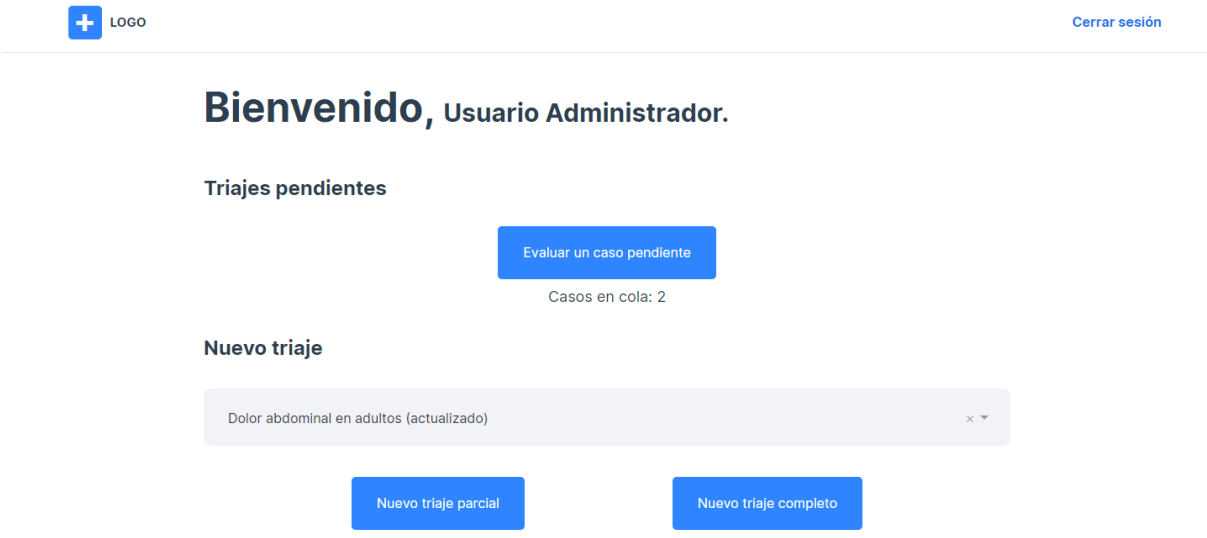

Figura 5.10 | **Opciones de iniciar triaje parcial o completo disponibles después de seleccionar el algoritmo en el buscador.**

5.2.6. Pantalla de carga

Uno de los aspectos más importantes en el diseño de interfaces es el de asegurar el ofrecimiento *feedback* informativo. Por ello, es importante añadir pantallas de carga, sobre todo en las *simple page applications*, donde la pestaña del navegador nunca recarga.

En nuestra interfaz, aparece la sección del contenido de la aplicación en blanco y una animación en el centro de ésta para indicar el estado “cargando”.

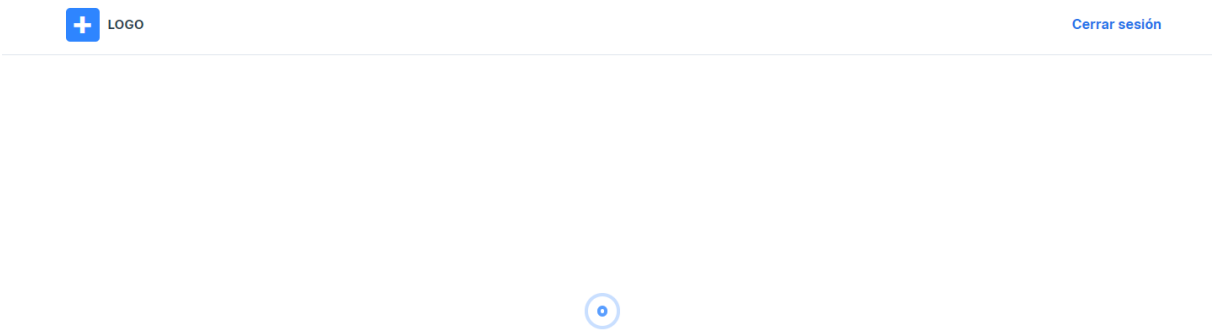

Figura 5.11 | **Pantalla de carga.**

5.2.7. Triaje parcial

En la figura 5.12 se puede ver la pantalla del proceso de triaje parcial. A la izquierda se muestran los discriminantes que pertenecen a los niveles críticos del algoritmo. Los discriminantes que pertenecen a niveles más urgentes aparecen más arriba. A la derecha se muestran la queja principal que identifica al algoritmo, el nombre del discriminante, su definición y las preguntas sugeridas. Abajo se encuentran los botones que sirven para indicar si se ha hallado el discriminante o no, omitir el paso o volver al paso anterior.

En la figura 5.13 se puede ver la pantalla para la confirmación y finalización del triaje. En este caso se muestra un estado no crítico y no se indican recomendaciones,

ya que más tarde un profesional sanitario se encargará de evaluar de manera más profunda el caso.

Por otro lado en la figura 5.14 se muestra la pantalla de confirmación y finalización habiendo encontrado un estado crítico y, por lo tanto, mostrando las recomendaciones oportunas mientras llegan los recursos de emergencia.

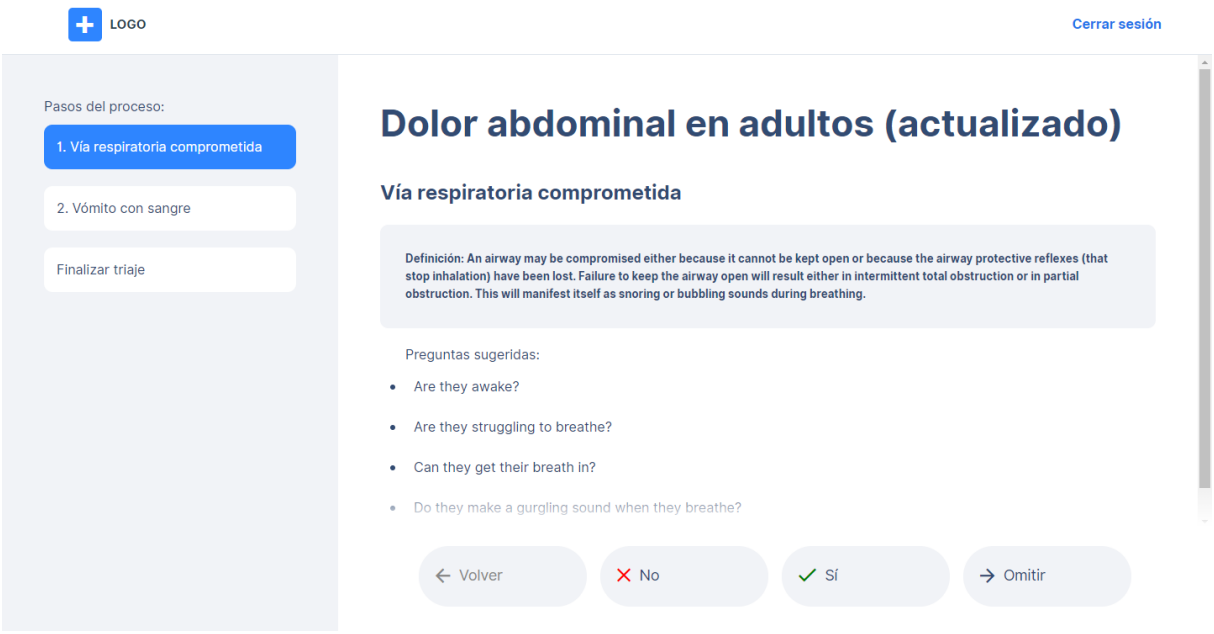

Figura 5.12 | **Inicio del proceso de triaje parcial.**

Figura 5.13 | **Confirmación y finalización del triaje parcial.**

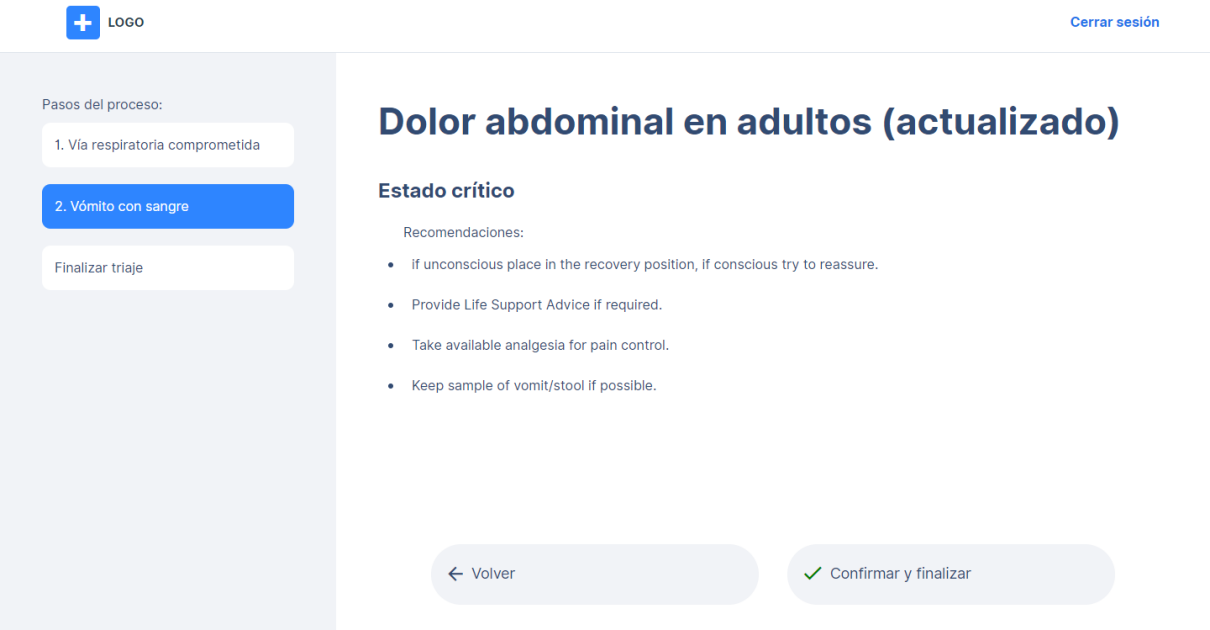

Figura 5.14 | **Confirmación y finalización del triaje parcial con estado crítico.**

5.2.8. Triaje completo

El caso de la pantalla del triaje completo es parecido al del triaje parcial. A diferencia de éste, a la izquierda se muestran los discriminantes de todos los niveles. A la derecha, la información mostrada por cada discriminador y los botones de navegación son idénticos. Se puede ver en la figura 5.15.

En la figura 5.16 se puede ver la pantalla de confirmación y finalización del proceso de triaje completo. En este caso se indica el nombre del nivel de urgencia (*FtF Now*, *Face to Face Now*, en el ejemplo).

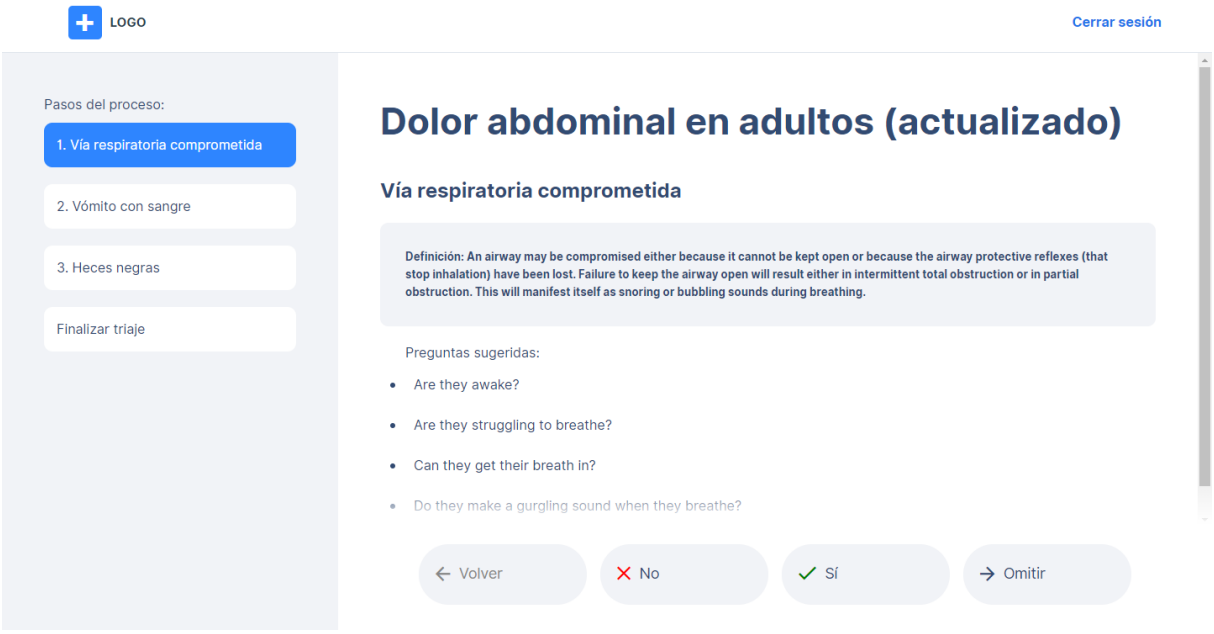

Figura 5.15 | **Inicio del proceso de nuevo triaje completo.**

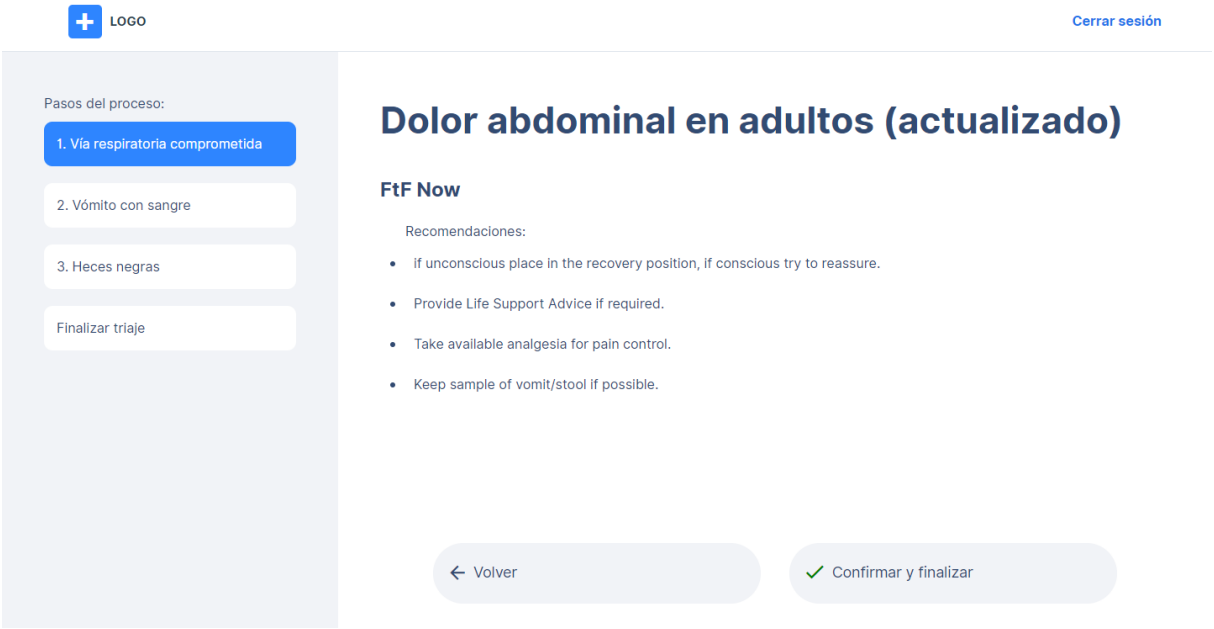

Figura 5.16 | **Confirmación y finalización del triaje completo.**

5.2.9. Triaje pendiente

La pantalla del triaje pendiente es idéntica a la del triaje completo, sólo que utiliza la información del triaje parcial previo. Se muestra en la figura 5.17.

La opción de evaluar un caso pendiente se muestra desactivada si la aplicación detecta que no hay casos en cola, como muestra la figura 6.1.

Figura 5.17 | **Inicio del proceso de completar triaje pendiente.**

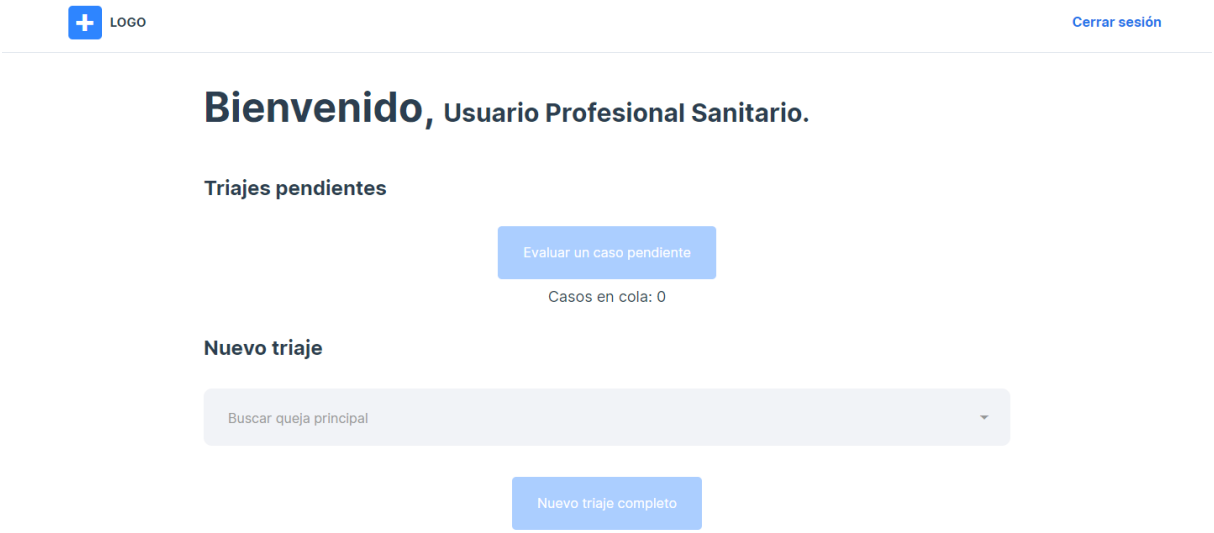

Figura 5.18 | **Opción de completar un triaje pendiente desactivada si no hay ninguno en cola.**

5.3. Análisis de las herramientas

Para satisfacer el requisito de soporte multiplataforma se ha escogido la plataforma de desarrollo de JVM y, en concreto, el lenguaje de programación Java. Asimismo, el software podrá ser distribuido en un contenedor de Docker, por lo que se facilita la instalación y despliegue en cualquier sistema operativo.

De alta importancia es también dotar al sistema de interoperabilidad semántica para facilitar la comunicación entre distintos proveedores y así mejorar los servicios sanitarios. Con este objetivo, es necesario trabajar con vocabularios estándar para la representación de información médica. Desde el inicio del proyecto se preveía usar la ontología HPO (*Human Phenotype Ontology*) para conseguir la interoperabilidad semántica y para aprovechar características propias de las ontologías, como la capacidad de computar valores de similitud entre conceptos, que se creían que podían ser necesarias para dar solución al problema. Tras el estudio del estado del arte, se clarificó que una mejor opción sería la ontología de SNOMED CT. SNOMED CT (*Systematized Nomenclature of Medicine – Clinical Terms*) tiene versiones bastante más maduras en otros idiomas aparte del inglés, incluso con extensiones localizadas (por ejemplo, hay una versión en español de España y en español de Argentina y español de Uruguay), es el vocabulario sanitario más amplio y preciso que existe e incluye los conceptos necesarios para definir todas las quejas principales, todos los hallazgos clínicos y todos los discriminantes. Sin embargo, la HPO incluyó hace muy poco algunos conceptos de signos y síntomas comunes, más allá de fenotipos relacionados con enfermedades hereditarias, lo cual se presenta como un problema por su poca madurez. Además, la HPO no ha sido traducida al español. La principal ventaja de SNOMED CT es que es un vocabulario extensible, de modo que se pueden construir expresiones inequívocas que identifiquen conceptos médicos con cualquier nivel de refinamiento. La IHTSDO (*International Health Terminology Standards Development Organisation*), la organización que posee SNOMED CT, ha desarrollado una librería para Java con el objetivo de analizar y validar sintácticamente estas expresiones, de la cual se hace uso en este programa.

Para la representación de los algoritmos de triaje a nivel de datos se utiliza el formato JSON (*JavaScript Object Notation*), que está ampliamente adoptado y es más liviano y legible por un humano que su principal alternativa, el formato XML (*eXtensible Markup Language*). Además, utilizamos el lenguaje de esquema JSON Schema (como reemplazo al XML Schema del formato XML) para describir el formato de los datos, proveerlo como documentación legible por humanos y máquinas y utilizarlo como va-

lidador de los datos que recibe la aplicación. Usando este estándar, podemos hacer uso de librerías que realicen la validación por nosotros, pasándole los datos recibidos y el *schema* que debe cumplir.

Uno de los requisitos más complejos es el de hacer el servicio agnóstico a otros sistemas externos y proveer la capacidad de ser integrable en un CRM. Para dar solución a esta condición es clave la arquitectura del sistema completo. Se ha pensado el servicio de triaje como un microservicio que es, por definición, independientemente funcional y que puede ser integrado con otros servicios (en este caso, todo el sistema CRM existente) a través de su API expuesta. Se detalla este enfoque en profundidad en la sección 5.1.1.

Esta API expuesta por el microservicio puede ser desarrollada sobre diferentes protocolos como HTTP, AMQP o incluso directamente sobre TCP. Sin embargo, HTTP es un candidato suficientemente bueno y conocido como para explorar las otras alternativas.

Teniendo claro el principal punto de entrada de la aplicación, una API HTTP, el siguiente paso lógico es elegir un *framework* sobre el que apoyarse: no es justificable implementar desde cero toda la lógica relacionada con la comunicación HTTP y el funcionamiento de un servidor en Java. Así pues, un candidato competente es el *framework* Spring Boot, uno de los más usados y conocidos, si no el que más.

La persistencia de los algoritmos de triaje se ha realizado haciendo uso del gestor de bases de datos NoSQL MongoDB, debido a su facilidad de uso y manejo de documentos JSON, el formato de datos que se va a utilizar a lo largo de todo el proyecto.

Con respecto a la interfaz web, se ha escogido Vue como *framework* de desarrollo por su facilidad de aprendizaje, por su entorno de desarrollo modular y flexible y por su rico ecosistema y versatilidad.

Una de las características que tiene la interfaz web es la de poder visualizar, en la pantalla de inicio de los usuarios con rol de administrador y de profesional sanitario, el número de casos pendientes en cola en tiempo real. Este requisito hubiera sido resuelto hace unos años haciendo uso del *polling* o *long-polling* con AJAX, que se basa en preguntar constantemente al servidor (o mantener la petición durante un tiempo en el caso del *long-polling*) si hay cambios nuevos. Hoy en día existen soluciones que se adecúan a estos casos de uso habilitando la comunicación servidor$\rightarrow$cliente. Estas dos soluciones son los *server-side events* (SSE) que funcionan en el navegador a través de la API de EventSource, y el protocolo WebSocket que funciona nativamente también haciendo uso de la API de WebSockets. En este caso, hemos escogido

trabajar con WebSockets por su mayor facilidad de integración con Spring Boot y el *message broker* RabbitMQ, a través del formato de datos STOMP.

Igualmente, la elección de RabbitMQ como *message broker* ha sido influida por la facilidad de integración con Spring Boot, aunque también es remarcable ser uno de los *message broker* más utilizados.

5.3.1. Herramientas de desarrollo

IntelliJ IDEA

IntelliJ IDEA es un entorno de desarrollo integrado (*integrated development environment*, IDE) para desarrollar software. Está desarrollado por el equipo de JetBrains y está disponible una edición *community* con una licencia Apache 2 y una edición comercial con una licencia propietaria [50].

Entre sus funcionalidades se encuentra una asistencia en la escritura de código muy potente (autocompletado de código analizando el contexto, navegación, refactorización, *linting* o detección de errores y sugerencias para arreglar inconsistencias); herramientas incorporadas e integración con otras herramientas de desarrollo como gestores de dependencias, control de versiones, gestores de bases de datos, Docker, etc.; un gran ecosistema de *plugins* que permiten añadir funcionalidades al IDE; y soporte de múltiples lenguajes de programación, *frameworks* y tecnologías (la mayoría a través de *plugins*).

Visual Studio Code

Visual Studio Code es un editor de código multiplataforma y con soporte multilenguaje. Permite abrir uno o más directorios que pueden ser guardados como *workspaces* para su reutilización. Tiene una potente *command palette* que permite acceder a numerosas características que no son expuestas en los menús.

Se puede extender en funcionalidad vía *plugins*, que están a disposición a través de un repositorio central. Una característica notable es la capacidad de crear extensiones que añaden asistencia para nuevos lenguajes, apariencias, depuradores, análisis de código estático o *linters* haciendo uso del *Language Server Protocol* y conectarse a servicios adicionales.

Además, permite a los usuarios configurar el carácter de salto de línea para Windows/Linux y el lenguaje de programación del documento activo. Esto permite que

pueda usarse en cualquier plataforma, en cualquier lugar y para cualquier lenguaje de programación [51].

Git

Git es un sistema de control de versiones distribuido para supervisar los cambios en el código durante el desarrollo de software. Es diseñado para coordinar el trabajo entre programadores, pero puede ser usado para monitorear cambios en cualquier conjunto de archivos. Sus objetivos incluyen la rapidez, la integridad de los datos y el soporte para flujos de trabajo distribuidos y no lineales.

Fue creado por Linus Torvalds en 2005 para el desarrollo del kernel de Linux con otros desarrolladores del kernel que contribuyeron a su desarrollo inicial. Su principal encargado de mantenimiento desde 2005 es Junio Hamano. Tal y como la mayoría de los otros sistemas de control de versiones distribuido y a diferencia de la mayoría de los sistemas cliente-servidor, cada directorio Git en cada ordenador es un repositorio integral, con la historia completa y con capacidades completas, independientemente de la existencia de conexión a internet o de un servidor central.

Es, además, software libre y de código abierto, distribuido bajo la licencia GPL 2.

Github

GitHub es una empresa que provee de un servicio de alojamiento de código para el control de versiones del desarrollo de software usando el anteriormente mencionado Git.

Ofrece todos las funcionalidades del control de versiones distribuido y de la gestión del código, además de proveer sus propias prestaciones. Proporciona control de acceso y varias funciones de colaboración como el monitoreo de *bugs*, peticiones de nuevas características, gestión de tareas y *wikis* para cada proyecto.

Ofrece planes gratuitos, profesionales y empresariales. Todos los planes incluyen repositorios privados desde enero de 2019. En mayo de 2019, GitHub reportó tener más de 37 millones de usuarios y más de 100 millones de repositorios, siendo, al menos, 28 millones repositorios públicos, haciéndoles el sitio más grande de alojamiento de código del mundo.

Gradle

Gradle es un sistema de automatización de *builds* de código abierto que se basa en los conceptos de Apache Ant y Apache Maven e introduce un DSL (*domain specific language*) basado en el lenguaje de programación Groovy en vez de configurarse utilizando archivos XML, como es el caso de Apache Maven. Gradle usa un grafo dirigido acíclico para determinar el orden en el que las tareas configuradas pueden ejecutarse.

Fue diseñado para construcciones (*biulds*) multiproyecto, que pueden crecer para llegar a ser repositorios muy pesados. Incluye soporte para construcciones incrementales determinando de forma inteligente qué partes del árbol de construcción están actualizados, de forma que las tareas dependientes de estas partes no necesitan ser ejecutadas de nuevo. Esto lo hace una herramienta mucho más veloz y eficiente que Maven y Ant.

Yarn

Yarn es un gestor de dependencias para JavaScript lanzado y mantenido por Facebook y Google. Es compatible con el otro gestor de dependencias de JavaScript más usado, NPM, pero se diferencia de éste en un enfoque a la hora de instalar paquetes. Utiliza archivos de bloqueo (`yarn.lock`) y un algoritmo de instalación determinista que le permite mantener la misma estructura de los directorios de la carpeta `node_modules`, donde se guardan las dependencias, para todos los usuarios involucrados en un repositorio, ayudando así a reducir los errores relacionados con las versiones de las dependencias.

El proceso de instalación y control que introduce Yarn tiene distintos pasos:

- *Resolution*: se resuelve las dependencias entre paquetes o librerías JavaScript haciendo solicitudes al registro y revisando cada dependencia que se encuentre ya gestionada dentro del directorio.
- *Fletching*: se revisa el directorio global almacenado en la memoria caché y se comprueba que el paquete o librería JavaScript que se quiere descargar no fue instalado con anterioridad. Si Yarn comprueba que no lo tiene, descarga el paquete y lo instala en la caché para evitar instalar y gestionar en el futuro la misma dependencia.
- *Linking*: se copian todos los archivos de la memoria caché al directorio `node_modules`

del repositorio local para que el desarrollador pueda empezar a trabajar con el paquete JavaScript.

Además, Yarn es capaz de instalar paquetes de forma paralela, aumentando la productividad.

5.3.2. Tecnologías y estándares

REST

Representational **S**tate **T**ransfer, en español Transferencia de Estado Representacional, es un estilo de arquitectura software que define un conjunto de principios para la creación de servicios web.

Es un protocolo cliente/servidor sin estado, es decir, cada petición HTTP contiene toda la información necesaria para ejecutar la acción deseada. Se basa en utilizar los verbos HTTP POST, GET, PUT y DELETE para realizar acciones sobre los recursos del servicio: crear, visualizar o consultar, editar y eliminar, respectivamente. Estos recursos deben ser identificados de forma inequívoca por la URI. Esto facilita la existencia de una interfaz uniforme que sistematiza el proceso con la información.

SNOMED CT

Systematized **N**omenclature of **Med**icine – **C**linical **T**erms es la terminología clínica integral, multilingüe y codificada de mayor amplitud, precisión e importancia desarrollada en el mundo. SNOMED CT es, también, un producto terminológico que puede usarse para codificar, recuperar, comunicar y analizar datos clínicos permitiendo a los profesionales de la salud representar la información de forma adecuada, precisa e inequívoca. La terminología se constituye, de forma básica, por conceptos, descripciones y relaciones. Estos elementos tienen como fin representar con precisión información y conocimiento clínico en el ámbito de la asistencia sanitaria [52].

Importante destacar por la utilidad en este proyecto que SNOMED CT provee un mecanismo que permite representar frases clínicas, incluso cuando un concepto único de SNOMED CT no transmite el nivel de detalle requerido (expresiones poscoordinadas). Esto permite registrar una amplia gama de significados clínicos sin necesidad de que la terminología incluya un concepto separado para cada combinación detallada de ideas que potencialmente podría ser necesario registrar [53].

JSON y JSON schema

JavaScript Object Notation (JSON) es un estándar abierto de formateo de datos que usa texto legible por los humanos para transmitir objetos de datos basados en pares de atributo-valor y *arrays* u cualquier otro valor serializable. Es un formato de datos muy común, con un amplio abanico de aplicaciones, como servir de sustituto de XML en sistemas AJAX.

Es un formato de datos independiente del lenguaje de programación, aunque fue originado a partir de la sintaxis de JavaScript. Aun así, muchos lenguajes de programación modernos incluyen la lógica para generar y parsear datos en formato JSON.

Por otro lado, JSON Schema sirve para especificar un formato basado en JSON. Define la estructura de datos JSON para su validación, su documentación y su control de interacción. Proporciona un contrato para los datos JSON requeridos por una aplicación dada, e indica cómo se pueden modificar esos datos. Está basado en los conceptos de XML Schema, pero basándose en el formato JSON.

STOMP sobre Websockets

STOMP (*Simple (or Streaming) Text Orientated Messaging Protocol*) es un protocolo de mensajería orientado a texto. Define un formato interoperable de forma que cualquiera de los clientes de STOMP disponibles pueden comunicarse con cualquier *message broker* de STOMP.

Por otro lado, la API de WebSockets capacita a las aplicaciones web mantener comunicaciones bidireccionales con procesos del lado del servidor de forma sencilla. A parte de esta API del navegador, se especificó un nuevo protocolo, (el protocolo WebSocket), que el navegador usa para comunicarse con los servidores. No se trata de puro TCP porque necesita proporcionar el modelo de seguridad *same-origin* del navegador. Tampoco se trata de HTTP porque difiere de entre en el modelo de petición-respuesta.

WebSockets sirve como protocolo base para enviar mensajes con formato STOMP, que es el protocolo que entiende nuestro *message broker* implementado con Spring Boot y RabbitMQ, habilitando la comunicación en tiempo real.

Java y JVM

La *Java Virtual Machine* (JVM) es una máquina virtual que permite al ordenador ejecutar programas escritos en lenguajes que compilan a *Java bytecode*. Uno de estos lenguajes es el utilizado en este proyecto para el *backend*, Java. Es un lenguaje de programación de propósito general que se basa en clases, es orientado a objetos (aunque no puro, ya que posee varios tipos primitivos), y está diseñado para tener las mínimas dependencias posibles.

Esta combinación de JVM y Java permite a los desarrolladores escribir código una única vez y que sea ejecutado en cualquier plataforma (*write once, run anywhere, WORA*) que esté soportada por la JVM, ya que nos permite abstraernos de detalles de arquitectura de computadores y sistemas operativos.

En 2019, Java fue uno de los lenguajes de programación más populares en uso según GitHub, particularmente para aplicaciones web cliente-servidor.

JUnit

JUnit es un *framework* de pruebas unitarias para el lenguaje de programación Java. Ha sido de gran importancia en el desarrollo de la metodología TDD, y es miembro de la familia de *frameworks* de pruebas unitarias conocidos como xUnit.

Spring Boot

Spring Framework es un *framework* y un contenedor de inversion de control para la plataforma Java. Las principales prestaciones pueden ser usadas en cualquier aplicación Java, pero hay extensiones para construir aplicaciones web sobre la plataforma Java EE (*Enterprise Edition*).

Aunque el *framework* no impone ningún modelo de programación específico, se ha vuelto popular en la comunidad de Java como sustituto del modelo Enterprise JavaBeans (EJB).

Spring Framework es un software de código abierto.

MongoDB

MongoDB es un gestor de bases de datos de propósito general, basado en documentos (por tanto, NoSQL) y distribuido. Almacena los datos en documentos total-

mente flexibles con formato similar a JSON. Los campos de estos documentos pueden variar de un documento a otro y la estructura de los datos pueden cambiar a lo largo del tiempo.

Tiene capacidad de ejecutar consultas *ad hoc*, indexación y agregación en tiempo real. También soporta el tipo de replicación primario-secundario y el balanceo de carga. Puede ser utilizado, además, como un sistema de archivos distribuido, aprovechando el balanceo de carga y la replicación de datos en múltiples servidores. Asimismo, MongoDB tiene capacidad de realizar consultas utilizando JavaScript, haciendo que estas sean enviadas directamente a la base de datos para ser ejecutadas.

Es un software gratuito y desde octubre de 2018 se distribuye bajo la licencia Server Side Public License (SSPL) v1.

Rabbit MQ

RabbitMQ es un *message broker* de código abierto que originalmente implementaba el protocolo AMQP (*Advanced Message Queuing Protocol*) y y desde entonces se ha ampliado con una arquitectura de plug-in para soportar Streaming Text Oriented Messaging Protocol (STOMP), Message Queuing Telemetry Transport (MQTT) y otros protocolos.

Docker

Docker es un proyecto de código abierto que automatiza el despliegue de aplicaciones en paquetes llamados contenedores usando una virtualización a nivel de sistema operativo. Los contenedores están aislados unos de otros y empaquetan su propio software, librerias y archivos de configuración. Docker utiliza características de aislamiento de recursos del kernel Linux, tales como *cgroups* y espacios de nombres (*namespaces*) para permitir que çontenedoresïndependientes se ejecuten dentro de una sola instancia de Linux, evitando la sobrecarga de iniciar y mantener máquinas virtuales.

HTML5, CSS y Javascript

HTML5 es la última versión de HTML (*HyperText Markup Language*), el componente más básico de la Web. HTML define el significado y la estructura del contenido de la web.

"Hipertexto" hace referencia a los enlaces que conectan páginas web entre sí, ya sea dentro de un mismo sitio web o entre sitios web. Los enlaces son un aspecto fundamental de la Web. HTML utiliza "marcado" para anotar texto, imágenes y otros contenidos para su visualización en un navegador Web. El marcado HTML incluye "elementos" especiales como `<head>`, `<title>`, `<body>`, `<head>`, `<header>`, `<footer>`, `<article>`, `<section>`, `<p>`, `<div>`, `<span>`, `<img>`, `<aside>`, `<audio>`, `<canvas>`, `<datalist>`, `<details>`, `<embed>`, `<nav>`, `<output>`, `<progress>`, `<video>`, `<ul>`, `<ol>`, `<li>` y muchos otros.

Generalmente se utilizan otras tecnologías además de HTML para describir la apariencia/presentación (CSS, *Cascading Style Sheets*) o la funcionalidad/comportamiento (JavaScript) de una página web. Javascript, el lenguaje disponible en los navegadores, es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.

5.3.3. VueJS

Vue es un *framework* para construir interfaces de usuario. A diferencia de otros frameworks monolíticos, Vue está diseñado para ser adoptado de forma incremental. La librería principal se centra únicamente en la capa de la vista, y es muy sencillo de integrar con otras librerías o proyectos.

Por otro lado, Vue es suficientemente potente para crear *Sigle Page Applications* muy sofisticadas haciendo uso de su rico ecosistema que incluye herramientas como una interfaz de línea de comandos (Vue CLI), un manejador de estado centralizado de tipo Flux (Vuex) y un *router* de URLs (vue-router), así como una gran cantidad de componentes y *plugins* desarrollados por la comunidad.

Otras de las grandes ventajas es su ligero peso, su facilidad de adopción y aprendizaje y su detallada documentación.

Capítulo 6

Verificación y validación

En esta sección se encuentran las pruebas que aseguran que el software desarrollado concuerda con su especificación. En este proyecto se ha llevado a cabo dos tipos de pruebas: las pruebas unitarias automáticas y las pruebas manuales.

6.1. Pruebas unitarias automáticas

Estas pruebas unitarias automáticas se han implementado para el código del *backend* haciendo uso de la librería JUnit. Con la arquitectura hexagonal, las unidades que son objeto de testeo son los *application services*, es decir, la clase que representa a un caso de uso orquestando el comportamiento de las entidades de dominio. De esta forma se está llevando a cabo la metodología *Outside-in TDD*, también conocida como escuela de Londres. Este enfoque se centra en comprobar que se lleva a cabo la funcionalidad final que se necesita, desarrollando las funcionalidades desde el exterior hacia dentro.

Gracias a la arquitectura de software adoptada en el proyecto, los componentes naturalmente pertenecientes a la capa de infraestructura (por ejemplo, los repositorios) pueden ser reemplazados en los tests por implementaciones sencillas en memoria sin hacer uso de librerías de *mocking*.

Se ha creado una clase de *test* para cada caso de uso.

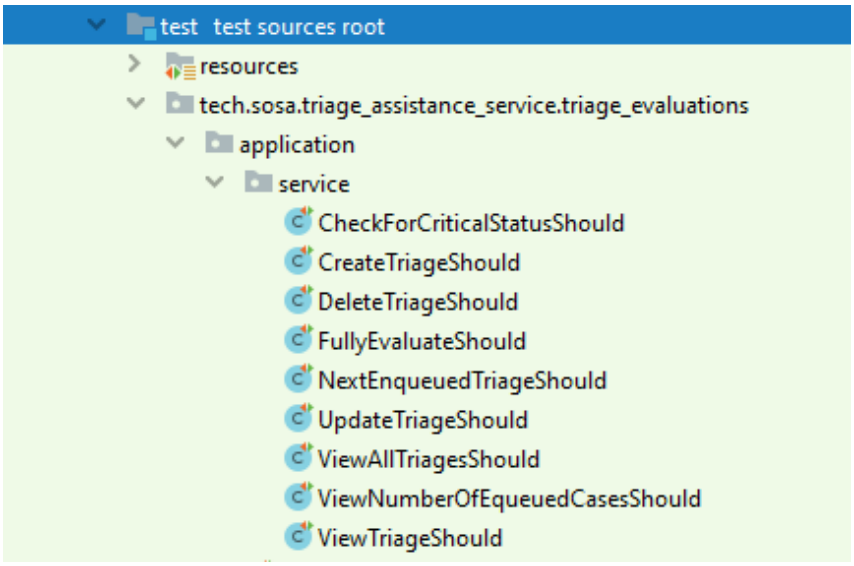

Figura 6.1 | **Clase de test por caso de uso.**

En cada test, se evalúa el comportamiento completo del caso de uso, desde el estado en la persistencia de datos y la respuesta hasta los eventos de dominios y la ejecución de sus suscriptores.

En el siguiente fragmento de código se ve un ejemplo de *test*.

```
public class FullyEvaluateShould extends TestWithUtils {

        private PendingTriagesQueue queue;
        private TriageRepository repository;
        private TriageMapper mapper;
        private InMemoryEventStore eventStore;

        @Before
        public void setUp() throws IOException, URISyntaxException {
                mapper = new TriageMapper();
                repository = TriageRepositoryStub.with(
                        mapper.from(
                                readJSON(readFromResource("triageExample.json"),
                                TriageDTO.class)
                        )
                );
                queue = new InMemoryPendingTriagesQueue(new ArrayList<>());
                eventStore = new InMemoryEventStore();
```

```
            EventPublisher.instance().reset();
            EventPublisher.instance().subscribe(new
                AuditingFullTriageAssessmentSubscriber(eventStore));
    }

@Test
public void return_FtFSoon_output_with_applicable_critical_findings() {
    AlgorithmLevelDTO actualOutput = new FullyEvaluate(repository, mapper,
        queue)
    .execute(new FullyEvaluateRequest(
            "21522001:246454002=41847000",
            Collections.singletonList("267055007")
    ));

    assertEquals(
            new AlgorithmLevelDTO(
                    "FtF Soon",
                    Arrays.asList(
                            "Take available analgesia for pain control.",
                            "Call back if symptoms worsen or concerned.",
                            "Keep sample of stool if possible."
                    ),
                    Collections.singletonList(
                            new DiscriminatorDTO(
                                    "267055007",
                                    "Heces negras",
                                    "Something",
                                    Collections.emptyList()
                            )
                    )
            ),
            actualOutput
    );

    assertEquals(eventStore.events().size(), 1);
    FullTriageAssessed capturedEvent = (FullTriageAssessed)
        eventStore.events().get(0);
    assertEquals(capturedEvent.getChiefComplaintId(),
        "21522001:246454002=41847000");
```

```
        assertNull(capturedEvent.getPreviousAssessment());
}
```

Se prueba que el triaje devuelva el nivel de urgencia correcto con los hallazgos dados. Además se comprueba que el evento de interés para la auditoría, `FullTriageAssesed`, es almacenado con éxito en el *event store*. Se incluye el método `setUp` y los atributos para mostrar la inyección de las dependencias, que en su mayoría son dobles de test, es decir implementaciones sencillas de los componentes que pertenecerían a la capa de infraestructura.

Además de los test para los *application services*, también se han implementado test unitarios automáticos para las entidades de dominio más complejas en cuanto a reglas de negocio, y para el *data mapper*.

6.2. Pruebas manuales

Como se ha explicado en la sección anterior, las pruebas unitarias automáticas llevadas a cabo comprueban el correcto funcionamiento de las capas de aplicación y de dominio. Para comprobar el buen funcionamiento del sistema completo, incluida la capa de infraestructura, se han realizado pruebas manuales. Para realizar la comprobación es necesario iniciar la ejecución de los contenedores de Docker con el script `run-docker.sh` localizado en la raíz del repositorio. Una vez desplegados todos los servicios podemos ejecutar las peticiones HTTP especificadas en el archivo `applications/test/requests.http`. El IDE utilizado en el desarrollo de este proyecto permite la ejecución de estas peticiones directamente desde el mismo archivo.

Unos ejemplos de peticiones incluidas son las mostradas en los siguientes segmentos de código.

Este primer ejemplo se corresponde con el caso de uso de insertar un nuevo algoritmo de triaje. Notar que en todas las peticiones se espera un *token* de autenticación en el *header* `Authorization`.

```
POST http://localhost:8080/triage/
Content-Type: application/json
Authorization: Bearer token-admin

{
    "triage": {
```

```
        "chiefComplaint": {
            "id": "21522001:246454002=41847000",
            "title": "Dolor abdominal en adultos"
        },
        "algorithm": {...}
    }
}
```

El siguiente ejemplo se corresponde con el caso de uso de eliminar un algoritmo de triaje. En este caso la ejecución no estaría autorizada, ya que el *token* se corresponde con un usuario con rol de profesional sanitario.

```
DELETE http://localhost:8080/triage/21522001:246454002=41847000
Authorization: Bearer token-health-professional
```

En este último ejemplo se muestra la llamada que se corresponde con el caso de uso de completar un triaje parcial previo.

```
POST http://localhost:8080/triage/full
Content-Type: application/json
Authorization: Bearer token-health-professional

{
    "previousPreTriageId": "9f937903-d2a7-47a6-8dab-9a28dff1529b",
    "triageChiefComplaintId": "21522001:246454002=41847000",
    "findingIds": [
        "161891005",
        "418290006 |prurito (hallazgo)| : 363698007 |sitio ..."
    ]
}
```

Además de realizar pruebas manuales con peticiones HTTP, se ha comprobado el buen funcionamiento del sistema completo haciendo uso del cliente web desarrollado. El caso de uso de visualizar en tiempo real el número de casos pendientes únicamente se ha probado haciendo uso del cliente web manualmente.

Capítulo 7

Conclusiones y líneas futuras

Los objetivos del producto mínimo viable se han cumplido. En la fase inicial del proyecto, el estudio del estado del arte en la metodología de los triajes, fue dedicada gran parte de la planificación, puesto que la solución metodológica a utilizar en los triajes era una elección totalmente abierta, era un campo totalmente desconocido para nosotros y no disponíamos de un experto del dominio que hiciera de guía. Necesitábamos, por tanto, estudiar y analizar en profundidad las diferentes maneras de solucionar el problema para escoger la estrategia más adecuada a las tecnologías actuales. Una vez seleccionada y justificada la metodología más adecuada para obtener un producto realmente usable, decidimos aplicar conceptos y paradigmas emergentes y de gran relevancia en la actualidad como es el paradigma de microservicios, las arquitecturas orientadas a eventos y la arquitectura hexagonal, que son tres conceptos que van de la mano, así como cuidar al máximo la calidad del software aplicando las buenas prácticas del desarrollo software, como el TDD, el Clean Code, los principios SOLID, etc. Finalmente, el resultado es un software facilmente mantenible, testeable y desacoplado al *framework* y un servicio independiente del “ecosistema” que lo rodea.

7.1. Limitaciones

Una de las principales limitaciones del proyecto ha sido la falta de un usuario final que demostrara la utilidad del producto.

En cuanto a la arquitectura hay que remarcar que, para soluciones muy sencillas, es ingeniería excesiva, ralentizando el desarrollo para garantizar la mantenibilidad del que se pueden beneficiar más las aplicaciones más sofisticadas.

También en relación con la arquitectura, ciertas partes de la implementación se dificultan usando ciertos frameworks. Por ejemplo, haciendo uso de Spring Boot, hemos tenido dificultades para desarrollar la autorización en la subscripción al *topic* de la visualización del número de casos pendientes en tiempo real vía WebSockets.

7.2. Líneas futuras

Como líneas futuras de trabajo y oportunidades de mejora, podemos sugerir los siguientes:

- Dotar al sistema de recordatorios de seguimiento a los pacientes no críticos.
- Dotar al sistema de un método de monitoreo y seguimiento de pacientes, útil sobre todo para ver la evolución de las pandemias. Minería de datos.
- Dotar al sistema de un método automático de detección de epidemias incipientes.
- Desarrollar un agente conversacional haciendo uso del *natural language processing* (y del *speech recognition*, si se le quiere dar soporte por audio) para la app de Salud Responde con el objetivo de guiar el triaje de auto-evaluación, tratándose de un primer filtro antes de la coumnicación directa vía telefónica.
- Desarrollar un servicio de autenticación y autorización real implementando, por ejemplo, el estándar JSON Web Token.
- Integración con una base de datos documental para la auditoría y con un sistema EHR/EMR. Por ejemplo, usar el *stack* ELK (ElasticSearch, Logstash y Kibana) y un servidor que implemente los estándares HL7 FHIR. De este modo, tendríamos un ecosistema bastante completo para su utilización en una organización sanitaria.

- “Envolver” el servicio de triaje en una arquitectura de tenencia múltiple y presentarla como un producto PaaS (*Platform as a Service*).

Bibliografía

[1] Fundéu BBVA. ¿Existe la palabra triaje? ¿Cómo se escribe triaje o triage? URL https://www.fundeu.es/consulta/triaje-triage/. (Citado en la página 1.)

[2] Nicolás González Casares, Carmen Martínez Lores, Francisco J. and Ureta Guzmán, and Sonia Alonso Juanes. Protocolo de triage o Recepción, Acogida y Clasificación (RAC) de Enfermería en Urgencias del Hospital Do Salnés. URL http://www.enferurg.com/articulos/protocolorac.htm. (Citado en la página 1.)

[3] Manuel Quero, María Belén Ramos, Wilfredo López, Juan José Cubillas, José María González, and José Luis Castillo. Uso de customer relationship management para mejorar la atención sanitaria de la ciudadanía. Servicio Salud Andalucía 24 horas. Salud Responde. *Gaceta Sanitaria*, 30(5):397–400, sep 2016. ISSN 0213-9111. doi: 10.1016/J.GACETA.2016.01.001. URL https://www.sciencedirect.com/science/article/pii/S0213911116000030?via{%}3Dihub. (Citado en la página 2.)

[4] Importancia de los Customer Relationship Management (CRM) sanitarios en las pandemias y alertas sanitarias. *Atención Primaria*, 47(5):267–272, may 2015. ISSN 0212-6567. URL https://www.sciencedirect.com/science/article/pii/S0212656714002315?via{%}3Dihub. (Citado en la página 2.)

[5] I Chen and K Popovich. Understanding customer relationship management (CRM): People, process and technology. *Business Process Management Journal*, 9(5):672–688, oct 2016. ISSN 1463-7154. doi: 10.1108/14637150310496758. (Citado en la página 2.)

[6] Teresa Sempere Selva, Salvador Peiró, Pilar Sendra Pina, Consuelo Martínez Espín, and Inmaculada López Aguilera. The Validity of the Hospital Emergency Suitability Protocol. *Revista Española de Salud Pública*,

73(4):461–475, 1999. URL http://scielo.isciii.es/scielo.php?script=sci{_}arttext{&}pid=S1135-57271999000400004. (Citado en la página 3.)

[7] MA Prieto Rodríguez, J Cantero Hinojosa, E Sánchez-Cantalejo Ramírez, J Martínez Olmos, J Maeso Villafaña, JJ Rodríguez Jiménez, and JM Jiménez Martín. Inadecuación de las visitas a un servicio de urgencias hospitalario y factores asociados. *Atención Primaria*, 28(5):326–332, 2001. ISSN 02126567. URL https://medes.com/publication/4534. (Citado en la página 3.)

[8] Jesús M Aranaz Andrés, Rafael Martínez Nogueras, M Teresa Gea Velázquez de Castro, Vicenta Rodrigo Bartual, Pedro Antón García, and Fernando Gómez Pajares. Why do patients use hospital emergency services on their own initiative? *Gaceta Sanitaria*, 20(4):311–315, 2006. URL http://scielo.isciii.es/scielo.php?script=sci{_}arttext{&}pid=S0213-91112006000400010. (Citado en la página 3.)

[9] Salvador Peiró, Julián Librero, Manuel Ridao, and Enrique Bernal-Delgado. Variability in Spanish National Health System hospital emergency services utilization. *Gaceta Sanitaria*, 24(1):06–12, 2010. URL http://scielo.isciii.es/scielo.php?script=sci{_}arttext{&}pid=S0213-91112010000100002. (Citado en la página 3.)

[10] CMBD Andalucía. Número de casos atendidos en urgencias para los grupos clínicos más frecuentes y porcentaje según sexo. 2017., 2018-04-26. URL https://www.juntadeandalucia.es/servicioandaluzdesalud/archivo-estadisticas/atencion-hospitalaria-2. (Citado en la página 3.)

[11] Servicio Andaluz de Salud Consejería de Salud y Familias. Junta de Andalucía. https://www.juntadeandalucia.es/servicioandaluzdesalud/profesionales/recursos-para-profesionales/precios-publicos., 2019-04-29. URL https://www.juntadeandalucia.es/servicioandaluzdesalud/profesionales/recursos-para-profesionales/precios-publicos. (Citado en la página 3.)

[12] Eric Evans. *Domain-driven design : tackling complexity in the heart of software*. Addison-Wesley, 2004. ISBN 0132181274. URL https://books.google.es/books/about/Domain{_}Driven{_}Design.html?id=hHBf4YxMnWMC{&}redir{_}esc=y. (Citado en la página 5.)

[13] Martin Fowler. Ubiquitous Language, 2006. URL https://martinfowler.com/bliki/UbiquitousLanguage.html. (Citado en la página 5.)

[14] Martin Fowler. Anemic Domain Model, 2003. URL https://www.martinfowler.com/bliki/AnemicDomainModel.html. (Citado en la página 5.)

[15] W Soler, M Gómez Muñoz, E Bragulat, and A Álvarez. Triage: a key tool in emergency care. *An. Sist. Sanit. Navar*, 33:55–68, 2010. URL http://scielo.isciii.es/pdf/asisna/v33s1/original8.pdf. (Citado en las páginas 6 y 22.)

[16] Michael D Christian, Cindy Hamielec, Neil M Lazar, Randy S Wax, Lauren Griffith, Margaret S Herridge, David Lee, and Deborah J Cook. A retrospective cohort pilot study to evaluate a triage tool for use in a pandemic. *Critical Care*, 13(5):R170, 2009. ISSN 1364-8535. doi: 10.1186/cc8146. URL http://ccforum.biomedcentral.com/articles/10.1186/cc8146. (Citado en la página 6.)

[17] Kayode A Adeniji and Rebecca Cusack. The Simple Triage Scoring System (STSS) successfully predicts mortality and critical care resource utilization in H1N1 pandemic flu: a retrospective analysis. *Critical Care*, 15(1):R39, 2011. ISSN 1364-8535. doi: 10.1186/cc10001. URL http://www.ncbi.nlm.nih.gov/pubmed/21269458http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC3221968http://ccforum.biomedcentral.com/articles/10.1186/cc10001. (Citado en la página 6.)

[18] Z. Khan, J. Hulme, and N. Sherwood. An assessment of the validity of SOFA score based triage in H1N1 critically ill patients during an influenza pandemic. *Anaesthesia*, 64(12):1283–1288, dec 2009. ISSN 00032409. doi: 10.1111/j.1365-2044.2009.06135.x. URL http://doi.wiley.com/10.1111/j.1365-2044.2009.06135.x. (Citado en la página 6.)

[19] Michael Christ, Florian Grossmann, Daniela Winter, Roland Bingisser, and Elke Platz. Modern Triage in the Emergency Department. *Deutsches Aerzteblatt Online*, dec 2010. ISSN 1866-0452. doi: 10.3238/arztebl.2010.0892. URL https://www.aerzteblatt.de/10.3238/arztebl.2010.0892. (Citado en la página 6.)

[20] Thereza Raquel Machado Azeredo, Helisamara Mota Guedes, Ricardo Alexandre Rebelo de Almeida, Tânia Couto Machado Chianca, and José Carlos Amado Martins. Efficacy of the Manchester Triage System: a systematic review. *International Emergency Nursing*, 23(2):47–52, apr 2015. ISSN 1755-599X. doi: 10.1016/J.IENJ.2014.06.001. URL https://www.sciencedirect.com/science/article/pii/S1755599X14000512?via{%}3Dihub. (Citado en la página 6.)

[21] Ignacio Párraga Martínez. Información y Triage: Puntos clave frente a la pandemia de la Gripe A. *Rev. Clin. Med. Fam.*, 2(8):375–377, 2009. URL http://www.msps.es/. (Citado en la página 7.)

[22] Paul Rutter, Oliver Mytton, Benjamin Ellis, and Liam Donaldson. Access to the NHS by telephone and Internet during an influenza pandemic: an observational study. *BMJ Open*, 4:4174, 2014. doi: 10.1136/bmjopen-2013-004174. URL http://bmjopen.bmj.com/. (Citado en la página 7.)

[23] Catherine S. Eppes, Patricia M. Garcia, and William A. Grobman. Telephone triage of influenza-like illness during pandemic 2009 H1N1 in an obstetric population. *American Journal of Obstetrics and Gynecology*, 207(1):3–8, jul 2012. ISSN 00029378. doi: 10.1016/j.ajog.2012.02.023. URL https://linkinghub.elsevier.com/retrieve/pii/S000293781200186X. (Citado en la página 7.)

[24] Arthur L. Kellermann, Alexander P. Isakov, Ruth Parker, Michael T. Handrigan, and Seth Foldy. Web-Based Self-Triage of Influenza-Like Illness During the 2009 H1N1 Influenza Pandemic. *Annals of Emergency Medicine*, 56(3):288–294.e6, sep 2010. ISSN 01960644. doi: 10.1016/j.annemergmed.2010.04.005. URL https://linkinghub.elsevier.com/retrieve/pii/S0196064410003549. (Citado en la página 7.)

[25] Rebecca Anhang Price, Daniel Fagbuyi, Racine Harris, Dan Hanfling, Frederick Place, Todd B. Taylor, and Arthur L. Kellermann. Feasibility of Web-Based Self-Triage by Parents of Children With Influenza-Like Illness. *JAMA Pediatrics*, 167(2):112, feb 2013. ISSN 2168-6203. doi: 10.1001/jamapediatrics.2013.1573. URL http://archpedi.jamanetwork.com/article.aspx?doi=10.1001/jamapediatrics.2013.1573. (Citado en la página 7.)

[26] Dominik Aronsky, Ian Jones, Bill Raines, Robin Hemphill, Scott R Mayberry, Melissa A Luther, and Ted Slusser. An integrated computerized triage system in the emergency department. *AMIA ... Annual Symposium proceedings. AMIA Symposium*, 2008:16–20, nov 2008. ISSN 1942-597X. URL http://www.ncbi.nlm.nih.gov/pubmed/18999190http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC2656061. (Citado en la página 8.)

[27] Programa de Ayuda al Triaje. URL https://www.triajeset.com/. (Citado en la página 8.)

[28] Infermedica. URL https://infermedica.com/. (Citado en la página 8.)

[29] ehCOS Triage. Productos para Hospitales y Centros de Salud. URL https://www.ehcos.com/productos/ehcos-triage/. (Citado en la página 8.)

[30] Schmitt-Thompson Online Telephone Triage Software - ClearTriage. URL https://www.cleartriage.com/. (Citado en la página 8.)

[31] Telephone triage software - TriageLogic. URL https://triagelogic.com/telephone-triage-software/. (Citado en la página 8.)

[32] Telephone Triage - CAPITA Healthcare Decisions. URL https://capitahealthcaredecisions.com/solutions/all/telephone-triage/. (Citado en la página 8.)

[33] Triage Software — Keona Health. URL http://keonahealth.com/triage-software/. (Citado en la página 8.)

[34] Technology — IntefleCS Triage Software — TeamHealth. URL https://www.thmedicalcallcenter.com/about-us/technology/. (Citado en la página 9.)

[35] Amol S Wagholikar, Michael J Lawley, David P Hansen, and Kevin Chu. Identifying Symptom Groups from Emergency Department Presenting Complaint Free Text using SNOMED CT. Technical report. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3243271/pdf/1446{_}amia{_}2011{_}proc.pdf. (Citado en la página 9.)

[36] Michael Lawley, Donna Truran, David Hansen, Norm Good, Andrew Staib, and Clair Sullivan. SnoMAP: Pioneering the Path for Clinical Coding to Improve Patient Care. *Studies in health technology and informatics*, 239:55–62, 2017. ISSN 1879-8365. URL http://www.ncbi.nlm.nih.gov/pubmed/28756437. (Citado en la página 9.)

[37] Eider Sanchez, Carlos Toro, Eduardo Carrasco, Gloria Bueno, Carlos Parra, Patricia Bonachela, Manuel Graña, and Frank Guijarro. An Architecture for the Semantic Enhancement of Clinical Decision Support Systems. pages 611–620. Springer, Berlin, Heidelberg, 2011. doi: 10.1007/978-3-642-23863-5_62. URL http://link.springer.com/10.1007/978-3-642-23863-5{_}62. (Citado en la página 9.)

[38] Guilherme Wunsch, Cristiano A. da Costa, and Rodrigo R. Righi. A Semantic-Based Model for Triage Patients in Emergency Departments. *Journal of Medical Systems*, 41(4):65, apr 2017. ISSN 0148-5598. doi: 10.1007/

s10916-017-0710-y. URL http://www.ncbi.nlm.nih.gov/pubmed/28283999http://link.springer.com/10.1007/s10916-017-0710-y. (Citado en la página 9.)

[39] Jocelyn San Pedro, Frada Burstein, Patrick Cao, Leonid Churilov, Arkady Zaslavsky, and Jeff Wassertheil. Mobile Decision Support for Triage in Emergency Departments. Technical report, 2004. URL http://www.diva-portal.org/smash/get/diva2:1002680/FULLTEXT01.pdf. (Citado en la página 9.)

[40] María M. Abad-Grau, Jorge Ierache, Claudio Cervino, and Paola Sebastiani. Evolution and challenges in the design of computational systems for triage assistance. *Journal of Biomedical Informatics*, 41(3):432–441, jun 2008. ISSN 1532-0464. doi: 10.1016/J.JBI.2008.01.007. URL https://www.sciencedirect.com/science/article/pii/S1532046408000130. (Citado en la página 9.)

[41] Sarmad Sadeghi, Afsaneh Barzi, Navid Sadeghi, and Brent King. A Bayesian model for triage decision support. *International Journal of Medical Informatics*, 75(5):403–411, may 2006. ISSN 13865056. doi: 10.1016/j.ijmedinf.2005.07.028. URL http://www.ncbi.nlm.nih.gov/pubmed/16140572https://linkinghub.elsevier.com/retrieve/pii/S1386505605001437. (Citado en la página 9.)

[42] Scott Levin, Matthew Toerper, Eric Hamrock, Jeremiah S. Hinson, Sean Barnes, Heather Gardner, Andrea Dugas, Bob Linton, Tom Kirsch, and Gabor Kelen. Machine-Learning-Based Electronic Triage More Accurately Differentiates Patients With Respect to Clinical Outcomes Compared With the Emergency Severity Index. *Annals of Emergency Medicine*, 71(5):565–574.e2, may 2018. ISSN 01960644. doi: 10.1016/j.annemergmed.2017.08.005. URL http://www.ncbi.nlm.nih.gov/pubmed/28888332https://linkinghub.elsevier.com/retrieve/pii/S0196064417314427. (Citado en la página 9.)

[43] Tadahiro Goto, Carlos A. Camargo, Mohammad Kamal Faridi, Robert J. Freishtat, and Kohei Hasegawa. Machine Learning–Based Prediction of Clinical Outcomes for Children During Emergency Department Triage. *JAMA Network Open*, 2(1):e186937, jan 2019. ISSN 2574-3805. doi: 10.1001/jamanetworkopen.2018.6937. URL http://jamanetworkopen.jamanetwork.com/article.aspx?doi=10.1001/jamanetworkopen.2018.6937. (Citado en la página 9.)

[44] Yoshihiko Raita, Tadahiro Goto, Mohammad Kamal Faridi, David F. M. Brown, Carlos A. Camargo, and Kohei Hasegawa. Emergency depart-

ment triage prediction of clinical outcomes using machine learning models. *Critical Care*, 23(1):64, dec 2019. ISSN 1364-8535. doi: 10.1186/s13054-019-2351-7. URL http://www.ncbi.nlm.nih.gov/pubmed/30795786http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC6387562https://ccforum.biomedcentral.com/articles/10.1186/s13054-019-2351-7. (Citado en la página 9.)

[45] Stephen M. Schenkel and Robert L. Wears. Triage, Machine Learning, Algorithms, and Becoming the Borg. *Annals of Emergency Medicine*, 71 (5):578–580, may 2018. ISSN 01960644. doi: 10.1016/j.annemergmed.2018.02.010. URL http://www.ncbi.nlm.nih.gov/pubmed/29530655https://linkinghub.elsevier.com/retrieve/pii/S0196064418301410. (Citado en la página 9.)

[46] Nathaniel R. Greenbaum, Yacine Jernite, Yoni Halpern, Shelley Calder, Larry A. Nathanson, David Sontag, and Steven Horng. Contextual Autocomplete: A Novel User Interface Using Machine Learning to Improve Ontology Usage and Structured Data Capture for Presenting Problems in the Emergency Department. *bioRxiv*, page 127092, apr 2017. doi: 10.1101/127092. URL https://www.biorxiv.org/content/10.1101/127092v1.full. (Citado en la página 9.)

[47] Yacine Jernite, Yoni Halpern, Steven Horng, and David Sontag. Predicting Chief Complaints at Triage Time in the Emergency Department. Technical report. URL https://people.csail.mit.edu/dsontag/papers/JerniteEtAl{_}nips13health.pdf. (Citado en la página 9.)

[48] Shameek Ghosh, Sammi Bhatia, and Abhi Bhatia. Quro: Facilitating User Symptom Check Using a Personalised Chatbot-Oriented Dialogue System. *Studies in health technology and informatics*, 252:51–56, 2018. ISSN 1879-8365. URL http://www.ncbi.nlm.nih.gov/pubmed/30040682. (Citado en la página 10.)

[49] R. Sánchez Bermejo, Carmen Cortés Fadrique, Beatriz Rincón Fraile, Esther Fernández Centeno, S. Peña Cueva, and E.M. De las Heras Castro. El triaje en urgencias en los hospitales españoles. *Emergencias : revista de la Sociedad Española de Medicina de Emergencias.*, 25(1):66–70, 2013. ISSN 1137-6821. URL https://dialnet.unirioja.es/servlet/articulo?codigo=4153479. (Citado en la página 22.)

[50] Duane K Fields. *IntelliJ IDEA in action*. Manning, 2006. ISBN 1932394443. (Citado en la página 50.)

[51] Microsoft. Documentation for Visual Studio Code. URL https://code.visualstudio.com/docs. (Citado en la página 51.)

[52] Gobierno de España Ministerio de Sanidad Consumo y Bienestar Social. ¿Qué es SNOMED CT? URL https://www.mscbs.gob.es/profesionales/hcdsns/areaRecursosSem/snomed-ct/quees.htm. (Citado en la página 53.)

[53] SNOMED International. Expresiones de SNOMED CT. URL https://confluence.ihtsdotools.org/display/RMT/7.+Expresiones+de+SNOMED+CT. (Citado en la página 53.)