



UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH



Desarrollo de un sistema web sanitario para los centros de atención primaria

Trabajo Fin de Grado

Memoria del proyecto

Autor

Nuria Antón Calle

Grado en Ingeniería Informática - Especialidad de Ingeniería del software

Directora

Cristina Gómez Seoane

Departamento de Ingeniería de Servicios y Sistemas de Información

Tutora GEP

Carolina Maria Consolación Segura

Doctora en Administración y dirección de Empresas

24 de enero de 2023

Agradecimientos

A todos los sanitarios de los centros de atención primaria que han querido participar en el proyecto, por brindarme su tiempo.

A mi tutora Cristina, por darme la libertad y el asesoramiento que necesitaba.

A Mar, por proponerme la idea y haber sido mi soporte durante toda la carrera.

A mi hermana Cristina, por escucharme hablar durante horas sobre el proyecto.

Y a mis padres, por darme la oportunidad de estudiar informática y apoyarme desde el primer momento.

RESUM

Actualment, el sistema sanitari espanyol està col·lapsat i es produeixen diàriament llargues llistes d'espera als centres d'atenció primària. Aquest problema s'ha intensificat amb la pandèmia Covid-19, que ha revelat a les grans masses les mancances del nostre sistema sanitari. Els metges es veuen obligats a augmentar la quota de pacients que visiten en un dia per donar abast, cosa que acaba afectant tant la qualitat de la visita del pacient com la seva pròpia salut mental.

A l'aplicació web que es desenvoluparà en aquest projecte, s'implementaran una sèrie de funcionalitats dissenyades per solucionar alguns dels problemes que pateix l'actual programa sanitari d'atenció primària de Catalunya. Aquestes funcionalitats tindran com a objectiu principal reduir el temps que el personal sanitari inverteix introduint o buscant informació al sistema.

RESUMEN

Actualmente, el sistema sanitario español está colapsado y se producen diariamente largas listas de espera en los centros de atención primaria. Este problema se ha visto intensificado con la pandemia Covid-19, que ha revelado a las grandes masas las carencias de nuestro sistema sanitario. Los médicos se ven obligados a aumentar el cupo de pacientes que visitan en un día para dar abasto, lo que acaba afectando tanto a la calidad de la visita del paciente como a su propia salud mental.

En la aplicación web que se desarrollará en este proyecto, se implementarán una serie de funcionalidades diseñadas para solucionar algunos de los problemas que padece el actual programa sanitario de atención primaria de Cataluña. Estas funcionalidades tendrán como objetivo principal reducir el tiempo que el personal sanitario invierte introduciendo o buscando información en el sistema.

ABSTRACT

Nowadays, the Spanish health system is collapsed and there are long waiting lists in primary care centers everyday. This problem has intensified with the Covid-19 pandemic, which has revealed to the people the shortcomings of our health system. Doctors are forced to increase the number of patients they visit in a day to cope, which ends up affecting both the quality of the patient's visit and their own mental health.

In the web application that will be developed in this project, it will be implemented some functionalities designed to solve the problems suffered by the current primary care health application in Catalonia. The main objective of these functionalities will be to reduce the time that healthcare staff spend entering or searching for information in the system.

ÍNDICE

1. Introducción y contextualización	8
1.1. Descripción del problema	8
1.2. Actores implicados	9
1.3. Motivación	10
2. Justificación del proyecto	11
2.1. Soluciones existentes	11
2.2. Justificación de la solución	11
2.2.1. Comparativa con el ECAP	11
2.2.2. Comparativa con las soluciones existentes	14
3. Alcance del proyecto	15
3.1. Objetivos	15
3.2. Requisitos	15
3.2.1. Requisitos funcionales	15
3.2.2. Requisitos no funcionales	16
3.3. Obstáculos y riesgos	18
4. Metodología	19
4.1. Metodología de trabajo	19
4.2. Herramientas de seguimiento	20
5. Planificación temporal	22
5.1. Descripción de las tareas	22
5.1.1. Gestión del proyecto	22
5.1.2. Inception	23
5.1.3. Sprint 1	23
5.1.3. Sprint 2	24
5.1.3. Sprint 3	24
5.1.3. Sprint 4	25
5.1.3. Fase final	25
5.2. Recursos	26
5.2.1. Recursos humanos	26
5.2.2. Recursos materiales	26
5.3. Estimaciones	27
5.4. Diagrama de Gantt	28
5.5. Gestión de riesgo	29
5.5.1. Falta de conocimientos	29
5.5.2. Falta de tiempo	29
5.5.3. No tener acceso al ECAP	29
6. Gestión económica	30
6.1. Coste de los recursos humanos	30

6.2. Coste de los recursos materiales	32
6.3. Imprevistos	32
6.4. Contingencia	33
6.5. Presupuesto final	33
6.6. Control de gestión	34
7. Especificación	35
7.1. Diagrama de casos de uso	35
7.2. Descripción de los casos de uso	38
7.3. Modelo conceptual de datos	43
7.3.1. Esquema conceptual de datos	43
7.3.2. Restricciones de integridad	45
7.3.3. Descripción de las clases	45
7.4. Modelos de comportamiento	48
7.5. Historias de usuario	53
8. Diseño	56
8.1. Arquitectura de 3 capas	56
8.2. Patrón de diseño MVC	58
8.3. Arquitectura lógica	59
8.4. Patrones de diseño	60
8.4.1. Module pattern	60
8.4.2. Middleware pattern	60
8.4.3. Hook pattern	61
8.4.4. Provider pattern	61
8.4.5. Compound components pattern	61
8.5. Diseño del front-end	62
8.5.1. Mapa navegacional	62
Imagen 10. Mapa navegacional	62
8.5.2. Diseño de la capa de presentación	63
8.6. Diseño del back-end	71
8.6.1. API REST	71
8.6.2. Diagrama de clases	79
8.6.3. Diagrama de secuencia	81
8.6.4. Modelo de la base de datos	83
9. Implementación	89
9.1. Tecnologías	89
9.1.1. MERN Stack	89
9.2. Tecnologías aplicadas	91
9.2.1. Estructura del proyecto	91
9.2.2. Implementación de las funcionalidades	92
10. Validación y pruebas	97
10.1. Validación de los requisitos funcionales	97
10.1.1. Back-end	97

10.1.2. Front-end	99
10.2. Validación de los requisitos no funcionales	103
11. Identificación de leyes y regulaciones	104
12. Resultados de la gestión del proyecto	106
12.1. Resultados a nivel de metodología	106
12.1.1. Métodos de trabajo	106
12.1.1. Herramientas de seguimiento	106
12.1.1. Método de validación	107
12.2. Resultados a nivel de planificación	107
12.3. Resultados a nivel económico	109
12.4. Informe de sostenibilidad	111
12.4.1. Sostenibilidad ambiental	111
12.4.1. Sostenibilidad económica	112
12.4.1. Sostenibilidad social	113
13. Conclusiones	114
13.1. Conclusiones del proyecto	114
13.2. Reflexiones personales	114
13.3. Integración de conocimientos	115
13.4. Justificación de las competencias	116
13.5. Trabajo futuro	117
14. Bibliografía	110
Anexo 1. Índice de imágenes	122
Anexo 2. Índice de tablas	123
Anexo 3. Casos de uso	124
Anexo 4. Modelos de comportamiento	130
Anexo 5. Historias de usuario	142

1. Introducción y contextualización

1.1. Descripción del problema

En España contamos con la suerte de disponer de un sistema sanitario que nos cubre los gastos de prácticamente cualquier problema de salud que podamos padecer. De hecho, según el ranking de *CeoWorld* [1], España se encuentra en el puesto número 8 de países con mejor sistema sanitario a nivel mundial. Es un sistema muy complejo y sensible al que se ha de destinar muchísimos recursos anualmente. Sin embargo, pese a la importancia que tiene este sistema en nuestra sociedad, a diario recibimos noticias haciendo referencia a que no se están destinando suficientes recursos a este sector. Actualmente, el sistema se encuentra colapsado, provocando que cuando acudimos a pedir hora para una cita, se nos comunique que hay una larga lista de espera, incluso para casos graves que pese a no ser mortales nos provocan gran malestar o incluso incapacidad para poder seguir con nuestra vida a un ritmo normal. Además, estos problemas se han visto intensificados con la pandemia Covid-19, que ha revelado a las grandes masas las carencias de nuestro sistema sanitario.

Como ya se ha comentado, uno de los problemas más significativos que tenemos y con el que lamentablemente ya nos encontramos familiarizados, son las largas listas de espera. El crecimiento de la población y la pandemia del Covid-19 son los principales causantes del aumento del número de consultas. Cataluña es la comunidad autónoma más preocupante, pues los pacientes deben esperar de media 12,38 días para conseguir una cita con su médico de cabecera [2]. El sistema colapsa y los médicos han de aumentar el cupo de pacientes que visitan en un día para dar abasto, lo que acaba afectando tanto a la calidad de la visita hacia el paciente como a su propia salud mental.

El sistema sanitario es muy complejo y se descompone de un gran número de sectores, pero en este trabajo nos centraremos en los problemas que podemos hallar en el primer nivel de nuestro sistema sanitario: el centro de urgencias de atención primaria y el centro de atención primaria, los cuales a partir de ahora nos vamos a referir a ellos por sus siglas CUAP y CAP.

En la aplicación web que se desarrollará a lo largo de este proyecto se buscará optimizar algunas de las funcionalidades del actual programa sanitario de atención primaria de Cataluña. Para ello, se implementará un prototipo que logre enseñar cómo funcionan los nuevos cambios en una primera instancia y demostrar qué puede aportar nuestro nuevo sistema al actual sistema sanitario.

Como no se va a poder tener acceso al código del programa que se utiliza actualmente y no podemos saber qué tecnologías utilizan para formar el sistema, se creará una aplicación web completamente desde cero replicando únicamente las funcionalidades que vayamos a mejorar.

Nuestro principal objetivo será reducir el tiempo que el personal sanitario invierte introduciendo información en el sistema. Según un estudio de *healthAffairs* [3], se estima que un doctor pasa más tiempo en el ordenador que visitando al propio paciente, realizando

actividades como: mirar el curso clínico¹ del paciente, redactar informes o entradas², enviar mensajes al resto del personal sanitario, comprobar tareas pendientes, o revisar resultados de pruebas.

Al margen de todo esto, otro problema preocupante del actual sistema sanitario español es que cada una de las 17 comunidades autónomas utiliza su propio *software* para cumplir con sus funciones. Esto no hace nada más que entorpecer el intercambio de información entre diferentes comunidades autónomas, y los profesionales sanitarios reclaman que exista un único sistema sanitario para todo el territorio español. La parte positiva es que el gobierno español ya está trabajando en ello, aunque como podemos ver en el artículo que nos presenta Iván Fernández [4], no se planea que estén todas las medidas aplicadas hasta el año 2026. Por lo que se puede determinar que es muy importante tener este factor en cuenta a lo largo de todo el proyecto.

Otro de los objetivos que se espera conseguir es mejorar la importación de datos al sistema sanitario. Actualmente se recoge muy poca información del paciente en cada consulta a causa del poco tiempo que dispone el personal sanitario para realizarla y la carencia de herramientas. Esto provoca que a posteriori se complique la exportación de información, afectando tanto al propio personal de los centros de atención primaria, como a los encargados de realizar investigaciones sanitarias. Por lo que si se mejora la importación, se podrá crear un mejor sistema de exportación de información para que el personal sanitario obtenga más datos sobre sus pacientes, y además se pueda conseguir listados de pacientes complejos de manera más sencilla y rápida para facilitar la ejecución de futuros estudios sanitarios.

1.2. Actores implicados

Como bien nos explican Nick Rozanski y Eoin Woods [5], los actores implicados (*stakeholders* en inglés) son todas las personas que se van a ver afectadas por nuestra aplicación web. Es nuestra función como ingenieros informáticos identificar correctamente todos nuestros *stakeholders*, así como sus intereses, preocupaciones y necesidades para poder hacer un balance, definir prioridades y diseñar la aplicación que mejor se adapte a estos requerimientos.

A continuación, vamos a analizar los actores principales de nuestro proyecto:

- Usuarios: Personal sanitario que utilizará la aplicación web final como herramienta para poder cumplir con sus funciones.
- Clientes: Organización para la cual se implementa la aplicación web. En este caso, sería para el ICS³, que también sería el responsable de asegurarse del mantenimiento y actualización de la aplicación web.

¹ Curso clínico: historial médico del paciente que contiene la información, consultas y documentos de cada paciente.

² Entrada: apartado dentro del curso clínico que especifica toda la información relevante de una consulta a un paciente.

³ ICS: Institut Català de la Salut.

- **Desarrollador:** Persona encargada de diseñar, implementar y documentar el proyecto. Ya que es una única persona, ejecutará el trabajo de diseñador, programador, ingeniero de requisitos y tester.
- **Directora del proyecto:** Persona encargada de supervisar y guiar el proyecto para asegurarse de que se cumplen todos los objetivos con la calidad mínima requerida.

1.3. Motivación

Desde que era pequeña siempre he sido una persona muy curiosa. Eso ha provocado que tenga mucho interés en muchos ámbitos muy diferentes entre sí y que esté dispuesta a aprender un poco de todos. Esto me supuso un problema, cuando en segundo de bachillerato tenía que elegir qué iba a estudiar a continuación. Lo que me llevó a estudiar informática fue mi pasión por los ordenadores. Crecí con uno de ellos, ya trasteando con pocos añitos, y fue lo que decantó mi decisión de estudiar ingeniería informática en la FIB. La carrera me enseñó dos cosas fundamentales para mí. Que disfruto mucho diseñando sistemas y construyéndolos con mis propias manos, y que la informática realmente se encuentra en todos lados. Puedo trabajar en diferentes sectores, aprendiendo de cada uno de ellos, detectando problemas y aportando mi granito de arena desarrollando una aplicación hecha a medida para ellos. Y esta es una idea que realmente me apasiona de la ingeniería del *software*.

El sistema sanitario siempre ha sido un tema muy presente para mí. Tengo muchos conocidos y familiares que trabajan en este sector, y he podido ver desde cerca como les ha afectado el golpe de la pandemia del Covid-19. Considero que este proyecto realmente puede ser útil y aportar ideas para mejorar el sistema informático que se utiliza actualmente en los centros CUAP y CAP. Como ciudadanos que disponemos de una sanidad pública, privilegio que no se encuentra en todos los países, deberíamos reflexionar y valorar el gran trabajo que hace el personal sanitario por nosotros, y tratarlos con el respeto que se merecen. Espero de verdad poder ayudarles con este proyecto.

Finalmente, como reto personal quiero que este proyecto me sirva como consolidación de todos los conocimientos que he adquirido durante todos estos años. La FIB me ha dado la oportunidad de trabajar en diferentes aplicaciones, cada una con diferentes tecnologías. Pero nunca he podido diseñar e implementar un sistema desde cero sin ayuda de otros estudiantes de la FIB. Creo que el TFG es una oportunidad fantástica para desarrollar mis conocimientos tanto de front-end⁴ como de back-end⁵, especialmente en mi ámbito favorito que es el desarrollo web.

⁴ **Front-end:** parte de la aplicación encargada de interactuar con los usuarios.

⁵ **Back-end:** parte de la aplicación que se encarga de gestionar la base de datos y el servidor.

2. Justificación del proyecto

2.1. Soluciones existentes

Estudiar correctamente los sistemas *software* que ya existen en el mercado nos ayuda a entender mejor qué funcionalidades son imprescindibles, coger nuevas ideas y ver qué podemos ofrecer nosotros para conseguir que nuestro producto destaque sobre el resto. Estos serían algunos de los *software* más relacionados con nuestra aplicación web:

- ECAP: programa actual que se utiliza en los CUAP y CAP para que el personal sanitario cumpla con sus funciones. Es el programa que vamos a tratar de mejorar en el transcurso de este proyecto.
- Nimbo Clinical [\[6\]](#): es un expediente clínico práctico fácil de usar que intenta disminuir el tiempo que pasa un doctor delante del ordenador para dedicarle más tiempo a sus pacientes. Es un programa multiplataforma, que se adapta a la especialidad de cada doctor. Permite ver información detallada de los pacientes, envía recordatorios de las citas y realizar consultas con videollamada.
- MediCloud [\[7\]](#): *software* multiplataforma en línea desarrollado por Microsoft Azure que nos permite controlar citas y acceder a expedientes, recetas y exámenes con un solo clic. Almacena toda la información en la nube, con lo cual se puede acceder a la misma en cualquier momento y lugar. La aplicación al principio no es muy intuitiva y no es visualmente atractiva, pero cumple con su propósito.
- Medilink [\[8\]](#): aplicación multiplataforma que permite gestionar tu propio centro médico. Permite acceder a la agenda médica, ficha clínica y documentos de los pacientes, y a una serie de reportes gráficos para ayudarte a mantener una buena gestión de tu clínica.

2.2. Justificación de la solución

2.2.1. Comparativa con el ECAP

A continuación, explicaremos con un poco más de detalle los problemas con los que se encuentra el personal sanitario a la hora de utilizar el ECAP. Esta información se ha obtenido tras la realización de diversas entrevistas con diferentes *stakeholders*, todos ellos especializados en un rol distinto dentro del ámbito sanitario.

Para empezar hablaremos de funcionalidades que ya dispone el programa del ECAP, pero que van a ser mejoradas en nuestro prototipo:

- Carga visual: Uno de los problemas más recurrentes que el personal sanitario comenta es que el ECAP es muy poco intuitivo. Contiene muchos botones que no saben para qué sirven o no se utilizan y tienen muchas maneras de llegar a un mismo lugar, provocando que muchos de ellos sean inútiles y solo aporten carga visual. Para solucionar esto, implementaremos un buen diseño, lo más claro e intuitivo que podamos, dentro de la propia complejidad del sistema.

- **Agendas:** El personal sanitario dispone de diferentes agendas que les permiten saber qué tareas han de realizar cada día, pero disponen de una agenda distinta para cada tipo de tarea. A causa de esto, todos los días han de comprobar agenda por agenda que no tengan tareas pendientes, sin mencionar que hay muchas de ellas que han quedado obsoletas y no se utilizan más, pero que siguen ahí ocupando espacio. Para solucionar este problema, se le enseñará al usuario las agendas en las que ese día tenga alguna tarea pendiente y solo se le mostrarán todas las agendas si el usuario lo solicita.
- **Objetivos:** Los objetivos son una serie de metas que buscan la prevención de problemas de salud de los pacientes. Un objetivo podría ser conseguir que un porcentaje de tus pacientes que no han realizado un análisis en los últimos años vayan a consulta para realizarlo. El problema es que el listado de pacientes que obtienen para realizarlas no se actualiza a tiempo real, sino que han de esperar a que lo actualicen (cosa que nos comunican que suele tardar unas dos semanas). Lo que hace bastante confuso saber qué pacientes han consultado ya o cuántos les quedan por consultar. Aparte de esto, a ellos les sale el porcentaje de personas que han de consultar en vez del número de pacientes directamente, obligándoles a calcular el porcentaje para cada uno de sus objetivos. Para mejorar este sistema, se actualizarán los objetivos cada vez que se detecte una nueva consulta a un paciente y se le otorgará al usuario el número de pacientes que tiene pendientes por consultar.

A continuación, comentaremos nuevas funcionalidades diseñadas para solventar problemas actuales del ECAP que aportarán mucho valor al sistema:

- **Listados predefinidos:** Los listados de pacientes son listados simples que contienen los nombres de los pacientes que padecen por ejemplo cierta enfermedad. El personal sanitario los utiliza tanto para realizar investigaciones a gran escala como a nivel personal. Por ejemplo, pueden decidir dejar de recetar un fármaco si ven que este no está haciendo el efecto esperado a sus pacientes. El problema yace, en que el personal sanitario no puede crear los listados con la información que desean, ya que únicamente tienen una serie de listados predefinidos en formato texto que no les es suficiente. Para solucionar este problema, se le permitirá al personal sanitario crear listados de pacientes con más de un parámetro de búsqueda, para que puedan crear una lista que se adapte a sus necesidades en tiempo real. Además, se permitirá acceder al curso clínico de un paciente directamente desde esa sección, para agilizar las tareas de seguimiento de un paciente.
- **Curso clínico:** Uno de los problemas que comentábamos en el apartado 1.1. es que el personal sanitario se ve obligado a invertir demasiado tiempo en frente del ordenador. Para mejorar este problema, se automatizará lo máximo su trabajo, dejando únicamente aquello que sí que esté obligado a rellenar manualmente en una consulta. Además, se creará un sistema de recomendaciones para el diagnóstico y los medicamentos, según lo que el personal sanitario redacte. Finalmente, se enseñará en todo momento información relevante del paciente, reduciendo así el número de veces que el personal sanitario ha de cambiar de pestaña.

- Globalización de información: Otro de los problemas que sufre el *software* sanitario español es que no tiene un sistema globalizado de información. Actualmente, cada comunidad autónoma dispone de su propio *software* sanitario, lo que dificulta a gran escala el intercambio de información entre los diferentes sistemas. Y este problema no se encuentra únicamente entre las diferentes comunidades autónomas, sino también entre diferentes sectores de una misma comunidad autónoma. Por ejemplo, aquí en Cataluña, si una persona de Lleida viene a algún centro de atención primaria de Barcelona, no se puede acceder a su curso clínico de manera directa. Hay que hacer un trámite intermedio para poder visualizar e importar la información que hay guardada en el otro sector. Y este problema se agrava si el paciente viene de otra comunidad autónoma, debido a que el trámite se complica y no te permite importar la información del paciente. Como cada comunidad autónoma tiene su propia manera de trabajar y el idioma va variando, hay que tratar de generalizar todo lo máximo posible. Se creará la aplicación tanto en catalán como en castellano y se permitirá traducir las entradas de un paciente en caso de que estén en otro idioma.
- Aplicación web: El ultimo problema que vamos a tratar es la dependencia de los ordenadores. El programa del ECAP solo se encuentra instalado en los centros CUAP y CAP por motivos de seguridad, lo que dificulta al personal sanitario trabajar desde fuera de sus centros. Se ha tomado la decisión de crear una aplicación web porque nos permite acceder a los datos desde cualquier dispositivo y deshacernos de la dependencia de los ordenadores. Permite al personal sanitario realizar campañas fuera de los centros con más facilidad, mejorar el rendimiento de las consultas domiciliarias, o realizar trabajo desde casa. Además se puede tener varias páginas abiertas simultáneamente, cosa que ayuda a agilizar la búsqueda de información.

2.2.2. Comparativa con las soluciones existentes

Una vez vista nuestra competencia directa, vamos a analizar el resto de sistemas expuestos en el apartado de soluciones existentes. Para ello, se ha realizado un listado que expone las funcionalidades principales de cada sistema. En la tabla 1 se muestra cada funcionalidad con un ✓ o ✗ que indica si la aplicación cumple con esta funcionalidad o no.

Funcionalidad	Aplicaciones				
	ECAP	Nimbo	MediCloud	Medilink	Nuestro proyecto
Es intuitiva	✗	✓	✗	✗	✓
Visualmente atractiva	✗	✓	✗	✗	✓
Multiplataforma	✗	✓	✓	✓	✓
El curso clínico contiene información detallada de un paciente	✗	✓	✗	✓	✓
Tiene un método de añadir un diagnóstico eficiente	✗	✗	✓	✗	✓
Recomendación de receta electrónica según diagnóstico	✗	✓	✗	✗	✓
Filtrar la entrada clínica del paciente	✓	✗	✗	✓	✓
Sistema de citas	✓	✓	✓	✓	✓
Sistema de agenda	✓	✓	✓	✓	✓
Visualización de objetivos	✓	✗	✗	✗	✓
Listado de pacientes con filtro detallado	✗	✗	✗	✗	✓
Exportar listados de pacientes	✓	✗	✗	✗	✓
Ver métricas personales	✗	✓	✓	✓	✓

Tabla 1. Comparativa de todas las soluciones existentes

Como se puede observar, actualmente no hay ninguna solución existente que cumpla con todas las funcionalidades propuestas. Hace falta mencionar que todas las soluciones existentes, a excepción del ECAP, son aplicaciones diseñadas para clínicas privadas, tanto para los doctores como para los pacientes. Esto provoca que tengan funcionalidades como consulta a través de videollamadas, cálculo de presupuestos o chats con los pacientes. Estas son funcionalidades que en nuestra aplicación web carecen de sentido, ya que está diseñada únicamente para el personal sanitario, por lo que no se han añadido. Sin embargo, esto nos permite centrarnos más en mejorar las funcionalidades mencionadas en la tabla 1 y crear un prototipo sólido que abarque la parte del sector sanitario que nos interesa.

3. Alcance del proyecto

3.1. Objetivos

El objetivo principal del proyecto es desarrollar una aplicación web que actúe como prototipo para enseñar posibles mejoras que permitan al personal sanitario de los centros CUAP y CAP de Cataluña cumplir con sus funciones con mayor eficiencia. Para ello, se esperan cumplir con los siguientes objetivos:

- Proporcionar herramientas al personal sanitario para aumentar su productividad y aliviar su carga de trabajo.
- Conseguir reducir las largas listas de espera y el tiempo que invierte el personal sanitario en el ordenador.
- Facilitar la exportación de información sobre los pacientes para la realización de futuras investigaciones sanitarias y para mejorar la calidad de trabajo del personal sanitario.
- Crear un sistema para facilitar el intercambio de información del curso clínico y de las recetas electrónicas entre diferentes comunidades autónomas.
- Diseñar una página web intuitiva y fácil de utilizar.

3.2. Requisitos

3.2.1. Requisitos funcionales

Pese a que nos encontramos ante un trabajo ambicioso y complejo, el hecho de que tengamos tiempo limitado y que el trabajo sea desarrollado por una única persona, nos obliga a definir un alcance realista. Por lo tanto, nos centraremos en realizar las funcionalidades que consideremos más relevantes, e implementaremos las que afecten únicamente a doctores y enfermeros, excluyendo al resto de personal sanitario. Esto implica que funcionalidades como dar de alta a un paciente o un trabajador (tareas de las cuales se suele encargar el personal administrativo) no se añadirán como funcionalidad del sistema, aunque al ser tareas indispensables para el proyecto se creará la manera de hacerlo desde el lado del back-end. Para más información al respecto, mirar el apartado 8.6.1.

Aparte de esto, hay algunas funcionalidades que dependen de APIs⁶ externas al ECAP como por ejemplo dar la baja a un paciente. Estas funcionalidades no se implementarán debido a la falta de tiempo para hacer un recurso similar. Las funcionalidades que se buscará trabajar para este proyecto son:

- Gestión de usuario: Se creará el respectivo login y logout para el usuario y se le otorgará diferentes permisos o información según el rol que tenga o el centro en el que inicie sesión.
- Gestión de los pacientes: Poder observar el curso clínico de un paciente, así como crear, modificar o borrar entradas e informes. También se podrán realizar recetas electrónicas.
- Gestión de la agenda: Se creará una agenda global donde se podrá ver, crear, modificar y borrar citas a los pacientes.
- Gestión de datos y objetivos: Se visualizarán los diferentes objetivos y la información relevante que necesite el personal sanitario para realizarlos. Además, se podrán crear y exportar en pdf o excel listados de pacientes que permitirán al personal sanitario realizar estudios sanitarios, tanto de gran alcance como para mejora personal.
- Multilenguaje: ya que se busca que sea un sistema accesible para toda España, se permitirá a cada trabajador elegir el lenguaje que desea para la página web. Aparte de esto, se le traducirán las notas de las entradas en caso de que estas estén en otro lenguaje. Para el desarrollo del proyecto, los lenguajes disponibles se limitarán al Catalán y al Castellano por falta de tiempo.

3.2.2. Requisitos no funcionales

Los requisitos no funcionales nos ayudan a asegurar que nuestro *software* cumpla con la calidad mínima requerida. A diferencia de los requisitos funcionales, los no funcionales no especifican que ha de realizar nuestro sistema, sino más bien cómo lo harán.

Para seleccionar los requisitos no funcionales del proyecto, se ha seguido la plantilla de Volere [9]. Pese a que nuestro proyecto es un prototipo para mostrar cómo funcionan ciertas funcionalidades que se podrían aplicar al ECAP, escogeremos los requisitos no funcionales que nuestro sistema tendría que cumplir en caso de llegar a producción.

Para cada requisito se especificará el tipo de requisito según la plantilla Volere, una pequeña descripción del requisito, una justificación de la importancia que tiene en el proyecto y finalmente los criterios de aceptación que se utilizarán al final de proyecto para comprobar si se ha cumplido o no con las expectativas de dicho requisito no funcional.

⁶ API: conjunto de definiciones y protocolos utilizados para integrar y diseñar sistemas software.

Tipo de requisito de Volere	11a. Requisito de Facilidad de Utilización.
Descripción	El sistema ha de ser de uso fácil, eficiente y satisfactorio para los usuarios.
Justificación del requisito	Los usuarios tendrán que trabajar con el sistema, por lo que es muy importante que se sientan cómodos a la hora de utilizarlo.
Criterios de aceptación	Al 75% de los testers les ha de parecer fácil de utilizar.

Tipo de requisito de Volere	11b. Requisito de Personalización e Internacionalización.
Descripción	El sistema ha de estar disponible en más de un lenguaje.
Justificación del requisito	El sistema ha de ofrecer las lenguas más habladas por los usuarios que vayan a utilizarlo.
Criterios de aceptación	La aplicación web estará disponible tanto para catalán como para castellano.

Tipo de requisito de Volere	12a. Requisitos de Velocidad y Latencia.
Descripción	El tiempo de respuesta para realizar una tarea ha de cumplir la velocidad apropiada para el ambiente previsto.
Justificación del requisito	El sistema ha de tener un tiempo de respuesta ínfimo, debido a que no se puede malgastar tiempo de la consulta esperando la respuesta.
Criterios de aceptación	El sistema responderá en menos de 1 segundo y medio para el 90 por ciento de las llamadas API.

Tipo de requisito de Volere	12d. Requisitos de Confiabilidad y Disponibilidad.
Descripción	El sistema ha de estar disponible el máximo tiempo posible.
Justificación del requisito	Ya que es un <i>software</i> que se ha de usar 24 horas al día y 365 días al año, su disponibilidad es primordial. Una falla puede ser peligrosa para la salud de las personas.
Criterios de aceptación	El sistema se encontrará operativo las 24 horas al día, 365 días al año.

Tipo de requisito de Volere	15a. Requisitos de Acceso.
Descripción	El sistema ha de controlar quién tiene acceso autorizado a las funcionalidades y datos del sistema.
Justificación del requisito	Como el sistema contiene datos sensibles, se ha de controlar exhaustivamente los permisos de acceso a según qué información.
Criterios de aceptación	El sistema controlará que la api sólo pueda ser accedida si el usuario se encuentra logueado y las contraseñas se almacenarán en la base de datos encriptadas.

Tipo de requisito de Volere	15c. Requisitos de Privacidad.
Descripción	El sistema ha de asegurar la privacidad de las personas sobre quienes almacena información.
Justificación del requisito	Se ha de asegurar que el sistema cumple con la ley de privacidad actual y que protege la privacidad individual de los pacientes.
Criterios de aceptación	Se hará un estudio sobre las leyes y regulaciones vigentes, y se asegurará que el producto cumple con la RGPD.

3.3. Obstáculos y riesgos

A lo largo del proyecto nos vamos a encontrar con una serie de obstáculos y riesgos que pueden llegar a afectar el desarrollo del proyecto. Para reducir el impacto es importante identificarlos correctamente:

- Conocimientos básicos: Como se van a utilizar tecnologías con las que el desarrollador no tiene mucha experiencia, es normal que aparezcan imprevistos o bugs y que se deba invertir más tiempo para un correcto aprendizaje.
- Falta de tiempo: Como la fecha de entrega está establecida y disponemos de poco tiempo para poder desarrollar un proyecto ambicioso, se tiene que realizar una buena planificación y definir correctamente el alcance del proyecto para evitar con funcionalidades no acabadas.
- No tener acceso al ECAP: Al ser un programa que trabaja con datos sensibles, no podemos tener la oportunidad de tener acceso a él. Para suplir con este obstáculo, se ha de trabajar exhaustivamente con los *stakeholders* y utilizar como apoyo los programas vistos en el apartado de soluciones existentes.

4. Metodología

4.1. Metodología de trabajo

La metodología que marcará los métodos, reglas y procedimientos a seguir durante este proyecto será la *Agile*, más en específico la técnica *Scrum* [\[10\]](#). Esta técnica se utiliza para el desarrollo del *software* basado en procesos iterativos e incrementales. Permite al proyecto adaptarse mejor a los problemas y a las necesidades de los *stakeholders* gracias a su flexibilidad. En la metodología *Scrum* tenemos 3 roles bien diferenciados:

- *Scrum Master*: Encargado de liderar al equipo, asegurarse de que se cumplen las fechas y las reglas *Scrum*, y proporcionar orientación al resto del equipo. La encargada de ejecutar este rol será Cristina Gómez Seoane, tutora de este proyecto.
- *Product owner*: Actúa como representante de los *stakeholders*. Es su trabajo asegurarse de que el proyecto cumpla con sus requisitos. La encargada de cumplir con esta función será Mar Bueno, una de los *testers* del proyecto.
- *Team*: Grupo de desarrolladores encargados de realizar las historias de usuario que se les asigna en cada etapa. En el caso de este proyecto, solo yo, Nuria Antón, seré la encargada de este rol.

Para cumplir correctamente con las reglas *Scrum*, el primer paso es definir un “*Backlog*” con todas las historias de usuario creadas a partir de las funcionalidades de nuestro proyecto. Posteriormente, estas se dividen en diferentes *sprints* que suelen tener una duración de 2-3 semanas. Al final de cada *sprint*, las funcionalidades han de estar acabadas y testeadas, y se ha de realizar una reunión con el equipo de desarrollo para analizar cómo ha ido el *sprint* y cuál es el actual estado del proyecto.

Durante la planificación temporal, el proyecto se dividirá en tareas más pequeñas. Estas se distribuirán a lo largo de todo el desarrollo en las siguientes 3 fases:

- La primera fase se realizarán los documentos necesarios para GEP. En esta fase del proyecto se definirán: la contextualización, la justificación de la solución, el alcance, la planificación temporal, la gestión económica y el informe de sostenibilidad.
- Durante la segunda fase se realizarán los *sprints* y se ejecutará el proceso iterativo. Al principio de cada *sprint*, se definirán todas las historias de usuario, los casos de uso, el modelo de comportamiento y las llamadas a la api que se requieran para las tareas asignadas a ese *sprint*. Posteriormente, se procederá a la implementación del front-end y del back-end y de los *test*. Para ello, se dividirá cada historia de usuario en tareas más pequeñas y estas se clasificarán en “*to do*”, “*in progress*”, “*to test*”, o “*done*”. Una vez se cumplan todos los criterios de aceptación de las historias de usuario y se pasen todos los *test*, se pondrá en la sección de *done* y se dará la tarea como finalizada. Al final de cada *sprint* se realizará la reunión con la tutora y se comentará el estado del proyecto.
- La última fase se dedicará al depurado final del producto y se terminará la documentación final.

4.2. Herramientas de seguimiento

Para el correcto desarrollo de la metodología *Scrum* se utilizará *Taiga* [11]. Es una herramienta que nos permite agrupar las diferentes historias de usuarios en grupos más grandes llamadas “épicas”. Aparte de esto, permite añadirles los criterios de aceptación y darles un nivel de prioridad dependiendo de su relevancia para el proyecto. Se pueden seleccionar qué historias de usuario realizar en cada *sprint* y permite crear tareas independientes para cada una de ellas. Las tareas se podrán separar en “*to-do*” si no están empezadas, “*in progress*” si se encuentran en desarrollo, “*to test*” si están a la espera de ser testeadas y por último “*done*” para indicar que están acabadas. Para finalizar, también te permite crear un listado *issues*, para reportar *bugs* del sistema y especificar el tipo, la severidad y la prioridad de cada *bug*.

Para organizar las tareas a ejecutar dentro de cada *sprint* se utilizará *Todoist* [12]. Es una herramienta para aumentar la productividad, que permite crear tareas con diferentes prioridades y asignarles una fecha para realizarlas. Funciona muy bien para apuntar ideas, desglosar tareas grandes con subtareas más pequeñas y tener una mejor idea de como planificar el *sprint*.

Como sistema de control de versiones se utilizará Github [13], que permite crear un repositorio para almacenar todo tu código en la nube y facilitar el trabajo en diferentes entornos. Como estrategia para el flujo de trabajo se utilizara *gitflow* [14]. Este sistema se basa en crear una rama principal llamada “*develop*”, para poder integrar las nuevas funcionalidades sin miedo a que se rompa el funcionamiento de la rama principal. A partir de esta rama, se creará múltiples ramas específicas para cada funcionalidad y al finalizar su desarrollo se realiza un *pull request*⁷ a la rama *develop*. En este paso se comprueba que cumple con los tests y la calidad requeridos para unirse a esta rama. En nuestro caso, como el equipo solo está formado por una persona, esta se encargará de gestionar el *pull request*, aunque en la vida laboral esta función la debería realizar otro desarrollador. Cuando se considera que la rama *develop* ya puede pasar a producción, los cambios se suben a una rama específica llamada “*release*”. En esta rama se prepara la versión final, solucionando posibles bugs y asegurándose que está preparada para subir a producción. Finalmente se suben los cambios a la rama *master*. En el diagrama mostrado en la imagen 1, se puede observar de manera más gráfica cómo sería el flujo de trabajo completo siguiendo la estrategia *gitflow*.

⁷ Pull request: petición para notificar que una rama está completa y se puede realizar la validación y el merge con la rama principal.

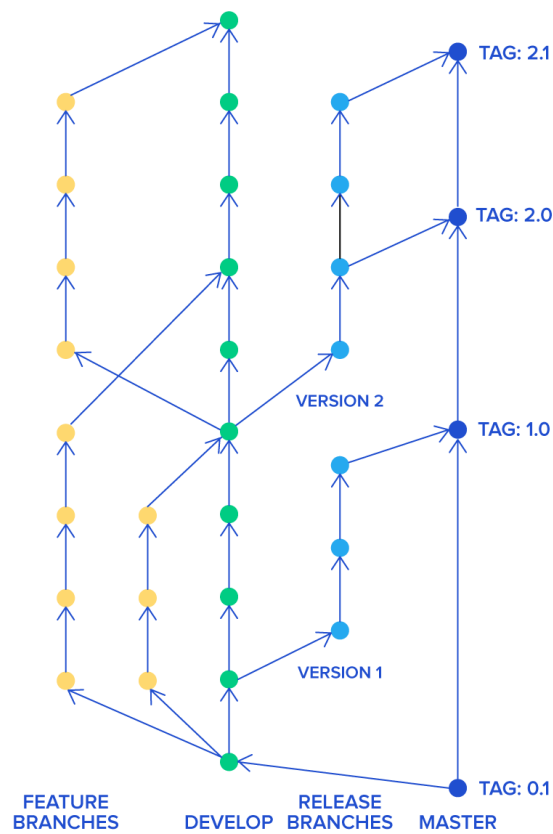


Imagen 1. Diagrama de flujo de *gitflow* [14]

Por último, para realizar la documentación necesaria para el proyecto se utilizará Google Docs [15], ya que permite almacenar todo en la nube para poder trabajar desde cualquier dispositivo y así no perder ningún progreso en caso de avería o problemas similares. Y para realizar las reuniones con la tutora se utilizará Google Meet [16].

5. Planificación temporal

Para asegurar que el proyecto cumpla con todos los objetivos y funcionalidades antes la fecha de entrega, es necesario que se realice previamente una planificación que se adapte lo máximo posible a todo los requerimientos.

El proyecto oficialmente comienza el 19 de septiembre y la fecha prevista para su finalización es el 22 de enero, donde todo tendrá que estar finalizado para la defensa oral. Por lo que en total disponemos de 18 semanas para realizar todo el proyecto. Partiendo de la base que la FIB estima que se requieren 540 horas para realizar el TFG [\[17\]](#), se dedicará de lunes a viernes una media de 6 horas diarias, a compensar en el fin de semana en caso de que algún día no se cumpla con las horas estimadas o que el proyecto se retrase por algún problema inesperado.

5.1. Descripción de las tareas

Se dividirán todas las tareas en 7 bloques. El primero será la gestión del proyecto, que según las entregas y fechas que nos propone GEP estará formado por las siguientes tareas: La contextualización y alcance, la planificación temporal, el presupuesto y sostenibilidad, y la integración del documento final. A continuación tendremos un bloque para la fase de *Inception* y 4 posteriores para cada *sprint*. Por último, tendremos el bloque de la fase final, que estará compuesto por una fase de testeo y depurado del prototipo, seguido de la finalización de la memoria y preparación de la defensa.

5.1.1. Gestión del proyecto

- **GP1, Contextualización y alcance:** Documentación sobre la contextualización, el alcance, la justificación de la solución, los obstáculos y riesgos, y metodología del proyecto.
 - **Duración:** 30 horas.
 - **Dependencias:** Ninguna.
- **GP2, Planificación temporal:** Documentación sobre la planificación de las tareas del proyecto y sus estimaciones. Se realizará un diagrama de gantt y se definirán los recursos y los planes para amortiguar los riesgos.
 - **Duración:** 15 horas.
 - **Dependencias:** GP1.
- **GP3, Presupuesto y sostenibilidad:** Estimación sobre el coste económico del proyecto y el estudio de sostenibilidad.
 - **Duración:** 15 horas.
 - **Dependencias:** GP2.

- **GP4, Documento final:** Documento final del curso de GEP que contendrá todas las entregas de los apartados anteriores con las correcciones aplicadas.
 - **Duración:** 15 horas.
 - **Dependencias:** GP3.

5.1.2. Inception

- **I1, Diseño de la arquitectura:** Definir la arquitectura del sistema, esquema de la base de datos, diagramas de secuencia, diagrama de clases, casos de uso y el diseño de la capa de presentación.
 - **Duración:** 40 horas.
 - **Dependencias:** GP1.
- **I2, Configuración del entorno de trabajo:** Crear los *github* y el *taiga* para el proyecto, e instalar toda la tecnología necesaria.
 - **Duración:** 5 horas.
 - **Dependencias:** I1.
- **I3, Familiarización con la tecnología:** Se creará el proyecto base y se aprenderán a realizar acciones básicas para ganar soltura.
 - **Duración:** 10 horas.
 - **Dependencias:** I2.

5.1.3. Sprint 1

- **S1-1, Sistema de autenticación:** Implementación y especificación del sistema de *login* y de *logout* del sistema, la gestión de permisos y el cambio de centro.
 - **Duración:** 20 horas.
 - **Dependencias:** I3.
- **S1-2, Visualizar curso clínico:** Implementación y especificación de la visualización del curso clínico de un paciente con sus documentos, entradas y recetas electrónicas.
 - **Duración:** 35 horas.
 - **Dependencias:** S1-1.
- **S1-3, Buscar curso clínico y entradas:** Implementación y especificación de buscar el curso clínico de un paciente según nombre, DNI, teléfono o número de la tarjeta sanitaria, y filtrar entradas según categoría.
 - **Duración:** 25 horas.
 - **Dependencias:** S1-2.
- **S1-4, Retrospectiva *sprint* 1:** Analizar si se han cumplido los objetivos del sprint, posibles errores, reunión con la tutora y reajustar el próximo *sprint* si es necesario.
 - **Duración:** 5 horas.
 - **Dependencias:** S1-1, S1-2 y S1-3.

- **S1-5, Documentación *sprint* 1:** Documentación sobre el progreso e información más detallada sobre las funcionalidades implementadas en ese *sprint*.
 - **Duración:** 10 horas.
 - **Dependencias:** Ninguna.

5.1.3. Sprint 2

- **S2-1, Añadir entrada al curso clínico:** Implementación y especificación de añadir una nueva entrada o informe al curso clínico de un paciente y sistema de recomendación de receta electrónica según diagnóstico.
 - **Duración:** 40 horas.
 - **Dependencias:** S1-3.
- **S2-2, Visualizar objetivos y métricas personales:** Implementación y especificación de la vista de los objetivos asignados al personal sanitario y de las métricas personales.
 - **Duración:** 35 horas.
 - **Dependencias:** S1-1.
- **S2-3, Retrospectiva *sprint* 2:** Analizar si se han cumplido los objetivos del *sprint*, posibles errores, reunión con la tutora y reajustar el próximo *sprint* si es necesario.
 - **Duración:** 5 horas.
 - **Dependencias:** S2-1 y S2-2.
- **S2-4, Documentación *sprint* 2:** Documentación sobre el progreso e información más detallada sobre las funcionalidades implementadas en ese *sprint*.
 - **Duración:** 10 horas.
 - **Dependencias:** Ninguna.

5.1.3. Sprint 3

- **S3-1, Visualizar listados de pacientes:** Implementación y especificación de la creación, y visualización de los listados de pacientes y su exportación a pdf o excel.
 - **Duración:** 35 horas.
 - **Dependencias:** S1-2 y S2-2.
- **S3-2, Sistema de citas:** Implementación y especificación de crear, modificar, borrar una cita, y ver una cita desde el curso clínico del paciente.
 - **Duración:** 25 horas.
 - **Dependencias:** S1-2.
- **S3-3, Retrospectiva *sprint* 3:** Analizar si se han cumplido los objetivos del *sprint*, posibles errores, reunión con la tutora y reajustar el próximo *sprint* si es necesario.
 - **Duración:** 5 horas.
 - **Dependencias:** S3-1 y S3-2.

- **S3-4, Documentación *sprint* 3:** Documentación sobre el progreso e información más detallada sobre las funcionalidades implementadas en ese *sprint*.
 - **Duración:** 10 horas.
 - **Dependencias:** Ninguna.

5.1.3. Sprint 4

- **S4-1, Visualizar agenda global:** Implementación y especificación de la agenda global, filtrado según tipo de tarea e implementación del sistema de citas desde la agenda.
 - **Duración:** 30 horas.
 - **Dependencias:** S3-2.
- **S4-2, Despliegue de la aplicación web:** Desplegar la aplicación web a un servidor.
 - **Duración:** 5 horas.
 - **Dependencias:** S4-1.
- **S4-3, Feedback con los *stakeholders*:** Facilitar a los *stakeholders* la aplicación web para recibir *feedback* sobre las funcionalidades y modificar las propuestas.
 - **Duración:** 25 horas.
 - **Dependencias:** S4-2.
- **S4-4, Retrospectiva *sprint* 4:** Analizar si se han cumplido los objetivos del *sprint*, posibles errores, reunión con la tutora y reajustar la fase final si es necesario.
 - **Duración:** 5 horas.
 - **Dependencias:** S4-1, S4-2 y S4-3.
- **S4-5, Documentación *sprint* 4:** Documentación sobre el progreso e información más detallada sobre las funcionalidades implementadas en ese *sprint*.
 - **Duración:** 10 horas.
 - **Dependencias:** Ninguna.

5.1.3. Fase final

- **F1, Depurado del prototipo:** Acabar de pulir posibles bugs o detalles visuales y refactorizar el código.
 - **Duración:** 20 horas.
 - **Dependencias:** S4-3.
- **F2, Finalización de la memoria:** Acabar de redactar los apartados restantes de la memoria y corregir posibles errores ortográficos.
 - **Duración:** 30 horas.
 - **Dependencias:** Ninguna.
- **F3, Preparación de la defensa oral:** Realizar las diapositivas y preparar la presentación para el turno de lectura.
 - **Duración:** 25 horas.
 - **Dependencias:** F2.

5.2. Recursos

A continuación, vamos a analizar los recursos tanto humanos como materiales que vamos a necesitar para la realización del proyecto.

5.2.1. Recursos humanos

Para comenzar, el estudiante encargado del proyecto dedicará una media de 30 horas semanales desde el inicio hasta la finalización del proyecto. Seguidamente tendremos el apoyo de los *testers*, que se encargarán de probar la aplicación web y de dar un feedback para mejorar el funcionamiento del sistema. Y para acabar tendremos las profesoras de la FIB: la profesora Cristina Gómez Seoane, que proporcionará ayuda al estudiante para que el proyecto vaya bien encaminado y aportará feedback en cada *sprint*, y la profesora Carolina Maria Consolación Segura, que corregirá los documentos relacionados con el curso de GEP.

5.2.2. Recursos materiales

Para la realización de este proyecto vamos a disponer de la residencia del estudiante como oficina principal, con un ordenador sobremesa, conexión a *Internet* y dos monitores. Como recurso adicional, se utilizará el portátil del estudiante en caso de realizar trabajo fuera de casa, o para las reuniones por *Google Meet* [16] que se realizarán con la tutora del proyecto.

Como herramientas de *software*, se utilizarán las herramientas que proporciona *Google*. Especialmente *Google Docs* [15], *Google Meet* [16] y *Google Drive* [18]. Y para realizar los diagramas necesarios para diseñar la arquitectura del sistema, se utilizará *Draw.io* [19], una herramienta gratuita que permite almacenar los diagramas en *Google Drive* [18].

Para implementación de código, se utilizará *Visual Studio Code* [20] como *IDE*⁸. En el *back-end* se utilizará *MongoDB Atlas* [21] como *host*⁹ para nuestra base de datos, *node.js* [22] para gestionar el servidor y *Postman* [23] para testear el funcionamiento de la *API*. Finalmente, para el despliegue de la aplicación web se utilizará *Netlify* [24].

⁸ **IDE**: De las siglas Integrated Development Environment, es un entorno digital utilizado para desarrollar software.

⁹ **Host**: Servicio que utiliza supercomputadoras para ofrecernos almacenar datos o páginas web.

5.3. Estimaciones

Seguidamente, en la tabla 2 se puede ver las horas estimadas de todas las tareas del proyecto con sus respectivos códigos, dependencias y recursos.

Bloque	Código	Tarea	Dependencias	Duración estimada	Recursos
Gestión de proyecto	GP1	Contextualización y alcance		30	Google Docs, Todoist
	GP2	Planificación temporal	GP1	15	Google Docs, Todoist
	GP3	Presupuesto y sostenibilidad	GP2	15	Google Docs, Todoist
	GP4	Documento final	GP3	15	Google Docs, Todoist
Inception	I1	Diseño de la arquitectura	GP1	40	Google Docs, Draw.io
	I2	Configuración del entorno de trabajo	I1	5	Visual Studio Code, MongoDB, Github
	I3	Familiarización con la tecnología	I2	10	Visual Studio Code, MongoDB, Github
Sprint 1	S1-1	Sistema de autenticación	I3	20	Visual Studio Code, MongoDB, Github, Taiga, Todoist
	S1-2	Visualizar curso clínico	S1-1	35	Visual Studio Code, MongoDB, Github, Taiga, Todoist
	S1-3	Buscar curso clínico y entradas	S1-2	25	Visual Studio Code, MongoDB, Github, Taiga, Todoist
	S1-4	Retrospectiva sprint 1	S1-1, S1-2, S1-3	5	Taiga, Google Meet
	S1-5	Documentación sprint 1		10	Google Docs, Todoist
Sprint 2	S2-1	Añadir entrada al curso clínico	S1-3	40	Visual Studio Code, MongoDB, Github, Taiga, Todoist
	S2-2	Visualizar objetivos y métricas personales	S1-1	35	Visual Studio Code, MongoDB, Github, Taiga, Todoist
	S2-3	Retrospectiva sprint 2	S2-1, S2-2	5	Taiga, Google Meet
	S2-4	Documentación sprint 2		10	Google Docs, Todoist
Sprint 3	S3-1	Visualizar listados de pacientes	S1-2, S2-2	35	Visual Studio Code, MongoDB, Github, Taiga, Todoist
	S3-2	Sistema de citas	S1-2, S2-2	25	Visual Studio Code, MongoDB, Github, Taiga, Todoist
	S3-3	Retrospectiva sprint 3	S3-1, S3-2	5	Taiga, Google Meet
	S3-4	Documentación sprint 3		10	Google Docs, Todoist
Sprint 4	S4-1	Visualizar agenda global	S3-2	30	Visual Studio Code, MongoDB, Github, Taiga, Todoist
	S4-2	Despliegue de la aplicación web	S4-1	5	Visual Studio Code, Github, Netlify
	S4-3	<i>Feedback con los stakeholders</i>	S4-2	25	Testers, Visual Studio Code, MongoDB, Github, Taiga, Todoist
	S4-4	Retrospectiva sprint 4	S4-1, S4-2, S4-3	5	Taiga, Google Meet
	S4-5	Documentación sprint 4		10	Google Docs, Todoist
Fase final	F1	Depurado del prototipo	S4-3	20	Visual Studio Code, MongoDB, Github, Taiga, Todoist
	F2	Finalización de la memoria		30	Google Docs, Todoist
	F3	Preparación de la defensa oral	F2	25	Google Docs, Todoist
Total				540	

Tabla 2. Estimación de horas y recursos de cada tarea

5.4. Diagrama de Gantt

En la tabla 3, se muestra la planificación de las tareas distribuidas a lo largo de las 18 semanas con las horas previstas y con un color según su bloque.

Bloque	Código	Tarea	Duración en horas	Semana																	
				19/09 25/09	26/09 02/10	03/10 09/10	10/10 16/10	17/10 23/10	24/10 30/10	31/10 06/11	07/11 13/11	14/11 20/11	21/11 27/11	28/11 04/12	05/12 11/12	12/12 18/12	19/12 25/12	26/12 01/01	02/01 08/01	09/01 15/01	16/01 22/01
Gestión de proyecto	GP1	Contextualización y alcance	30	30																	
	GP2	Planificación temporal	15		15																
	GP3	Presupuesto y sostenibilidad	15			15															
	GP4	Documento final	15				15														
Inception	I1	Diseño de la arquitectura	40		15	15	10														
	I2	Configuración del entorno de trabajo	5				5														
	I3	Familiarización con la tecnología	10					10													
Sprint 1	S1-1	Sistema de autenticación	20					18	2												
	S1-2	Visualizar curso clínico	35						26	9											
	S1-3	Buscar curso clínico y entradas	25							19	6										
	S1-4	Retrospectiva sprint 1	5								5										
	S1-5	Documentación sprint 1	10					2	2	2	4										
Sprint 2	S2-1	Añadir entrada al curso clínico	40								15	25									
	S2-2	Visualizar objetivos y métricas personales	35									3	28	4							
	S2-3	Retrospectiva sprint 2	5											5							
	S2-4	Documentación sprint 2	10									2	2	6							
Sprint 3	S3-1	Visualizar listados de pacientes	35												13	22					
	S3-2	Sistema de citas	25													6	19				
	S3-3	Retrospectiva sprint 3	5														5				
	S3-4	Documentación sprint 3	10												2	2	6				
Sprint 4	S4-1	Visualizar agenda global	30															28	2		
	S4-2	Despliegue de la aplicación web	5																5		
	S4-3	Feedback con los stakeholders	25																21	4	
	S4-4	Retrospectiva sprint 4	5																	5	
	S4-5	Documentación sprint 4	10															2	2	6	
Fase final	F1	Depurado del prototipo	20																	15	5
	F2	Finalización de la memoria	30															15	15		
	F3	Preparación de la defensa oral	25																		25
Horas totales				30	60	90	120	150	180	210	240	270	300	330	360	390	420	450	480	510	540

Tabla 3. Diagrama de gantt

5.5. Gestión de riesgo

Como ya se explicaba en el apartado de obstáculos y riesgos, tenemos 3 principales riesgos a tener en cuenta: la falta de conocimientos y de experiencia, la falta de tiempo y no tener acceso al ECAP. No definir un plan de mitigación, puede desencadenar que no se acaben cumpliendo los objetivos de este proyecto, con lo cual es muy importante que se haga correctamente.

5.5.1. Falta de conocimientos

A lo largo del proyecto, se tocarán varias tecnologías con las que el estudiante no está muy familiarizado. Esto implica, un añadido en horas para aprender a realizar ciertas tareas y una alta probabilidad de que aparezcan *bugs* a lo largo del proyecto que hagan perder el tiempo al estudiante. Para solucionar este problema, se ha añadido una tarea de familiarización con la tecnología, para que el estudiante tenga tiempo para aprender a realizar las acciones más simples, que serán obligatorias para las tareas posteriores. Sin embargo, esto no es suficiente en caso de que aparezca algún bug en alguna tarea más avanzada. En caso de que suceda, se utilizarán horas extras durante el fin de semana y se asegurará que el estudiante no se aleje demasiado de la planificación inicial.

5.5.2. Falta de tiempo

Como la fecha de entrega ya está establecida, el alcance del proyecto es un factor limitante. Ya se ha tenido en cuenta y el alcance se ha limitado para las tareas más prioritarias que se pueden realizar en este periodo de tiempo. En caso de necesitar más horas, se utilizarán las sobrantes de otras tareas. Y en el caso de necesitar aún más, las horas extras quedarán registradas para posteriormente poder calcular la desviación y su presupuesto extra. Sin embargo, en caso de no llegar a tiempo para la fecha de entrega, se eliminarán los casos de uso menos prioritarios y se destinará menos tiempo al apartado visual de la aplicación web para asegurar que las funcionalidades principales salgan adelante.

5.5.3. No tener acceso al ECAP

Uno de los obstáculos más importantes con el que nos encontramos es no tener acceso al programa que utiliza el personal sanitario actualmente. Esto se debe a que contiene información sensible de los pacientes. Para solucionar este problema, se dispondrá de la ayuda de 4 trabajadores de centros CUAP y CAP, que aparte de ser los *testers* principales, se encargarán de detallar al estudiante cómo es su trabajo y cómo lo realizan a través del programa. Además, se utilizará como apoyo los programas vistos en el apartado de soluciones existentes, como referencia visual más próxima.

6. Gestión económica

Una vez determinado los recursos tanto materiales como humanos de nuestro proyecto, se puede realizar una estimación del coste necesario para cada tarea y un presupuesto completo. El presupuesto nos permitirá decidir si el proyecto es viable o si hay que reducir costes.

6.1. Coste de los recursos humanos

Para calcular los costes de los recursos humanos, pese a que la mayoría de tareas las realiza el estudiante, se separará su trabajo según el rol que le corresponde en el mundo laboral. En la tabla 4 se enseña todos los roles que participan en el proyecto y lo que costaría cada rol por hora. Este resultado se ha calculado a través de los sueldos en bruto, que se han obtenido a través de la plataforma *GlassDoor* [\[25\]](#), multiplicado por 1.3 para añadir el coste de la seguridad social.

Rol	Coste (€/h)
Jefe de proyecto (JP)	33,17
Arquitecto de <i>software</i> (AS)	33,85
Desarrollador back-end (DB)	26,77
Desarrollador front-end (DF)	22,12
Tester (TS)	18,66

Tabla 4. Sueldo/hora de cada rol

Ahora que ya sabemos cuánto nos cuesta por hora cada trabajador implicado en el proyecto, podemos hacer la estimación del coste total de los recursos humanos. En la tabla 5, se puede visualizar las horas trabajadas por cada rol y su coste final.

Código	Horas					Duración en horas	Coste Total (€)
	JP	AS	DB	DF	TS		
GP1	30					30	995,01
GP2	15					15	497,50
GP3	15					15	497,50
GP4	15					15	497,50
I1		40				40	1.354,17
I2			3	2		5	124,56
I3			5	5		10	244,45
S1-1		4	8	6	2	20	519,64
S1-2		7	14	10	4	35	907,64
S1-3		5	10	7	3	25	647,82
S1-4	5					5	165,83
S1-5	10					10	331,67
S2-1		10	15	10	5	40	1.054,65
S2-2		6	14	12	3	35	899,36
S2-3	5					5	165,83
S2-4	10					10	331,67
S3-1		10	14	8	3	35	946,31
S3-2		5	9	9	2	25	646,62
S3-3	5					5	165,83
S3-4	10					10	331,67
S4-1		5	10	12	3	30	758,41
S4-2			2,5	2,5		5	122,23
S4-3		3	7	5	10	25	586,21
S4-4	5					5	165,83
S4-5	10					10	331,67
F1		2	3	5	10	20	445,26
F2	30					30	995,01
F3	25					25	829,17
Total	190	97	114,5	93,5	45	540	15.559,03

Tabla 5. Coste de los recursos humanos

6.2. Coste de los recursos materiales

Como ya se ha comentado anteriormente, para desarrollar el trabajo se utilizará un ordenador sobremesa, dos monitores, el resto de accesorios *hardware* necesarios y un portátil para las reuniones meet con la tutora encargada del proyecto. El *software* que se utilizará es completamente gratuito, por lo que no se añadirá dentro de los costes.

Para calcular el coste del material *hardware*, se ha de calcular su amortización. Este resultado se ha obtenido con la siguiente fórmula:

$$\text{amortización} = \frac{\text{precio (euros)} * \text{horas totales del proyecto}}{\text{vida útil (años)} * \text{días laborables/año} * \text{horas diarias trabajadas}}$$

En la tabla 6 se puede ver el coste directo de los recursos *hardware*, donde se ha aplicado la fórmula anterior con los valores siguientes: 540 como horas totales del proyecto, 5 años como vida útil de cada dispositivo, 220 días laborables al año y 6 horas diarias trabajadas en el proyecto.

Dispositivo	Coste (€)	Amortización (€)
Ordenador de sobremesa	990	81,00
Teclado Ozone Gaming Gear Strike Pro	84	6,87
Ratón Logitech G502 Hero	34	2,78
Monitor Samsung S24D330H	249	20,37
Monitor Benq ET0025-N	50	4,09
Portátil matebook pl-w19	899	73,55
Total	2306	188,67

Tabla 6. Coste directo de los recursos materiales

Finalmente, como se utiliza la casa del estudiante como oficina principal y no supone ningún coste la estancia, únicamente se añadirá como coste indirecto el precio mensual de la luz (120 euros/mes) y del internet (19,95 euros/mes), multiplicado por los 5 meses de duración del proyecto. Con lo cual, el coste final de los recursos materiales es **888,45** euros.

6.3. Imprevistos

Durante el desarrollo del proyecto pueden surgir imprevistos que pueden afectar al presupuesto. En el apartado de riesgos ya se ha especificado el plan a seguir para esos casos y una de las propuestas era necesitar horas extras. Ya se ha dado margen extra en la planificación de las tareas para evitar que esto suceda, por lo que se estima que las probabilidades de que esto ocurra son de un 30%, pero se asignarán 30 horas extras si es necesario. Si asumimos 15 horas para el desarrollador back-end y 15 para el desarrollador front-end, nos sale una media de 24,45 euros la hora.

Por último, también se considerará como imprevisto una posible avería en el *hardware*. Como es un caso poco probable y además se dispone de dos dispositivos principales (el ordenador sobremesa y el portátil), se estima una probabilidad del 5%, asumiendo un precio de 300 euros en el caso de que ocurra.

En la tabla 7 podemos observar el coste final estimado para los imprevistos.

Imprevisto	Coste (€)	Probabilidad (%)	Coste final (€)
Horas extras	733,35	30	220,01
Avería del <i>hardware</i>	300	5	15,00
Total	1033,35	35	235,01

Tabla 7. Coste para los imprevistos

6.4. Contingencia

Las contingencias son sucesos que pueden afectar al presupuesto del proyecto, pero no se tuvieron en cuenta durante la planificación. Para calcular el presupuesto destinado a este tipo de problemas, vamos a conceder un porcentaje del 15%, tanto a los costes directos como indirectos. En la tabla 8 se puede observar los costes finales de contingencia.

Tipo de coste	Coste (€)	Probabilidad (%)	Coste final (€)
Coste recursos humanos	15.559,03	15	2.333,85
Coste recursos materiales	888,42	15	133,26
Total	16.447,45	30	2.467,12

Tabla 8. Coste para la contingencia

6.5. Presupuesto final

En la tabla 9, se muestran los costes definitivos de cada apartado para calcular el coste final del proyecto.

Tipo de coste	Coste (€)
Coste recursos humanos	15.559,03
Coste recursos materiales	888,42
Imprevistos	235,01
Total	16.682,46

Tabla 9. Presupuesto final

6.6. Control de gestión

Finalmente, para calcular las desviaciones que ocurran en el transcurso del proyecto, se tendrá que calcular el tiempo real invertido en cada tarea. Es un paso importante para el presupuesto, ya que nos permite ver dónde se ha producido una desviación, y analizar el motivo y la magnitud de cada una. Para conseguirla, se calculará la diferencia entre el coste estimado y el coste real de cada tarea o apartado. En caso de que la desviación sea negativa, será necesario utilizar el dinero destinado a contingencias para cubrir el gasto resultante. A continuación se muestran las fórmulas oficiales que se usarán para calcular las desviaciones pertinentes de cada apartado:

- Desviación de los costes humanos por tarea = $(\text{coste estimado} - \text{coste real}) * \text{horas reales}$
- Desviación de los costes materiales = $\text{coste material estimado} - \text{coste material real}$
- Desviación de los costes indirectos = $\text{coste indirecto estimado} - \text{coste indirecto real}$
- Desviación de imprevistos = $\text{coste estimado imprevistos} - \text{coste real imprevistos}$
- Desviación de horas por tarea = $\text{horas estimadas} - \text{horas reales}$
- Desviación de las horas totales = $\text{horas estimadas} - \text{horas reales}$
- Desviación de los costes totales = $\text{costes estimados} - \text{costes reales}$

7. Especificación

Gracias a los apartados anteriores, ya tenemos una visión de cuál va a ser el alcance de nuestro proyecto y los objetivos que pretendemos conseguir. En este capítulo, nos centraremos en especificar de manera detallada todos los requisitos mencionados en el apartado 3.2.1 y delimitar así el entorno de nuestro trabajo. Se definirán todos los casos de uso, el modelo conceptual de datos, los modelos de comportamiento y las historias de usuario del sistema. Cabe destacar que pese a que el proyecto se ha realizado en *sprints*, se ha decidido presentar la documentación de manera secuencial en vez de iterativa para mejorar su legibilidad.

7.1. Diagrama de casos de uso

El primer paso será definir los casos de uso de nuestro sistema. Los casos de uso son una herramienta que describe como un usuario interacciona con el sistema para cumplir con cierto objetivo [\[26\]](#). A continuación, se mostrarán los casos de usos representados gráficamente a través de un diagrama de casos de uso, donde se puede observar para cada uno de ellos todos los actores involucrados y cuando se relacionan entre ellos.

Los actores que participan en los casos de uso serán los usuarios, que engloba tanto a los doctores como a los enfermeros (no se crea distinción porque todos pueden realizar los mismos casos de uso, solo que los enfermeros tendrán algunas limitaciones), y la API externa MyMemory que en los apartados posteriores hablaremos de su función dentro del sistema.

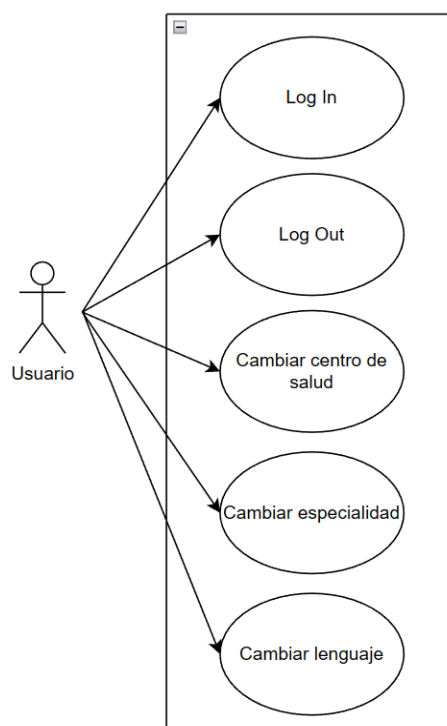


Imagen 2. Diagrama de casos de uso para gestionar la cuenta del usuario



Imagen 3. Diagrama de casos de uso para gestionar los pacientes

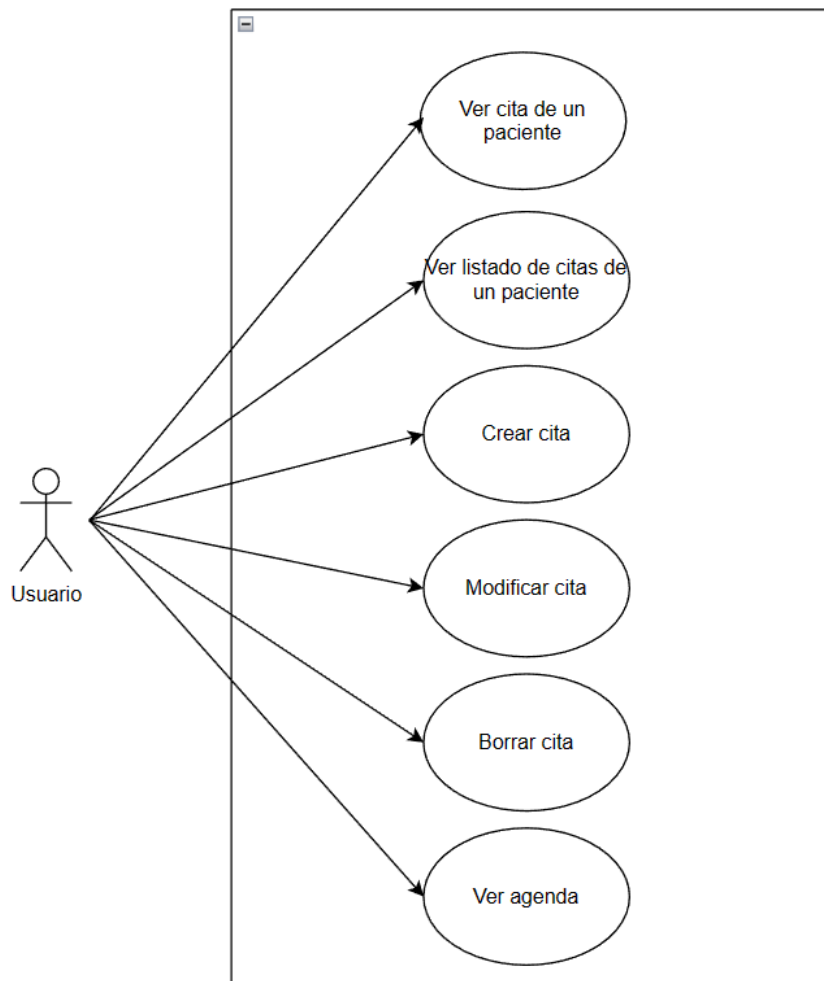


Imagen 4. Diagrama de casos de uso para gestionar la agenda del usuario

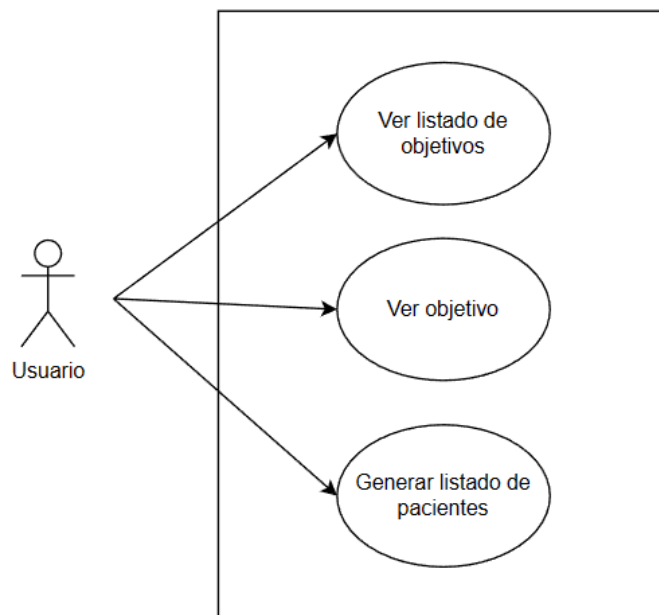


Imagen 5. Diagrama de casos de uso para gestionar la recaptación de información

7.2. Descripción de los casos de uso

Como hemos explicado en el apartado anterior, utilizamos los casos de uso para describir las posibles interacciones del usuario con el sistema. Está formado por un conjunto de escenarios que describen la secuencia de eventos que realizan los actores con el sistema para conseguir un determinado objetivo.

En este apartado veremos los casos de uso en formato completo, es decir, con todos los pasos y variantes explicados con detalle. Para cada caso de uso se describirán las siguientes cláusulas:

- El actor principal: actores que consiguen un objetivo a través del caso de uso.
- Las precondiciones: condiciones que se han de cumplir para que el usuario pueda realizar el caso de uso.
- El disparador: condición que inicia el caso de uso.
- El escenario principal: secuencia de eventos a seguir por parte del actor primario.
- Las extensiones: secuencia de eventos que se realiza en caso de desviación en el escenario principal.

Como muchos de los casos de uso son muy similares, se ha decidido mostrar solo los que se han considerado más relevantes. El resto de casos de uso se podrán encontrar en el apartado Anexo 3. Casos de uso.

Usuarios: Gestión de la cuenta del usuario

Caso de uso	#1 Log In	Actor principal	Usuario
Precondiciones			
Disparador	El usuario quiere acceder al sistema.		
Escenario principal de uso			
1. El sistema solicita al usuario las credenciales para identificarse. 2. El usuario proporciona las credenciales al sistema. 3. El sistema valida las credenciales.			
Extensiones			
2a. El usuario ha dejado algún campo vacío. 2a1. El sistema indica al usuario que ha de completar todos los campos. 2a2. El usuario sigue con el proceso desde el punto número 1. 2b. La contraseña es incorrecta. 2b1. El sistema indica al usuario que la contraseña es incorrecta. 2b2. El usuario sigue con el proceso desde el punto número 1.			

Caso de uso	#3 Cambiar centro de salud	Actor principal	Usuario
Precondiciones	El usuario ha de estar logueado en el sistema.		
Disparador	El usuario quiere cambiar el centro de salud.		
Escenario principal de uso			
<div>1. El usuario indica el centro de salud al que desea cambiar.</div> <div>2. El sistema indica al usuario que el centro de salud ha sido cambiado.</div>			

Usuarios: Gestión del paciente

Caso de uso	#6 Ver curso clínico	Actor principal	Usuario
Precondiciones	El usuario ha de estar logueado en el sistema.		
Disparador	El usuario quiere buscar según unos parámetros un paciente en concreto.		
Escenario principal de uso			
<ol style="list-style-type: none">1. El usuario proporciona los parámetros con los que desea buscar al paciente e indica al sistema que quiere realizar una búsqueda.2. El sistema muestra al usuario todos los pacientes que encajan con su solicitud.3. El usuario indica al sistema que paciente desea ver.4. El sistema muestra al usuario el curso clínico del paciente que ha seleccionado.			

Caso de uso	#10 Traducir entrada	Actor principal	Usuario
Precondiciones	El usuario ha de estar logueado en el sistema.		
Disparador	El usuario quiere traducir una entrada.		
Escenario principal de uso			
<div>1. El usuario indica al sistema que desea traducir la entrada.</div> <div>2. El sistema muestra al usuario la entrada traducida al idioma con el que trabaja el usuario.</div>			

Caso de uso	#11 Filtrar entrada.	Actor principal	Usuario
Precondiciones	El usuario ha de estar logueado en el sistema.		
Disparador	El usuario quiere filtrar las entradas según su diagnóstico.		
Escenario principal de uso			
<div>1. El usuario indica que desea filtrar las entradas según un diagnóstico.</div> <div>2. El sistema le muestra las entradas que coinciden con ese diagnóstico.</div>			

Caso de uso	#13 Ver nota	Actor principal	Usuario
Precondiciones	El usuario ha de estar logueado en el sistema.		
Disparador	El usuario quiere visualizar la nota de un paciente.		
Escenario principal de uso			
<div>1. El usuario indica al sistema que quiere ver la nota de un paciente.</div> <div>2. El sistema muestra al usuario la nota que el usuario quiere ver.</div>			

Caso de uso	#14 Crear nota	Actor principal	Usuario
Precondiciones	El usuario ha de estar logueado en el sistema.		
Disparador	El usuario quiere crear una nota al paciente.		
Escenario principal de uso			
<div>1. El usuario indica que quiere añadir una nota al curso clínico de un paciente.</div> <div>2. El sistema solicita los datos de la nueva nota al usuario.</div> <div>3. El usuario proporciona los datos e indica que desea guardar los cambios.</div> <div>4. El sistema muestra el listado con la nueva entrada.</div>			
Extensiones			
<div>3a. El usuario quiere añadir una prescripción.<div>3a1. El usuario indica que quiere añadir una prescripción al sistema.</div><div>3a2. Empieza el caso de uso de crear prescripción.</div><div>3a3. Se continúa el caso de uso desde el punto 3.</div></div> <div>3b. El usuario no ha seleccionado un diagnóstico.<div>3b1. El sistema indica al usuario que ha de seleccionar un diagnóstico.</div><div>3b2. El usuario selecciona un diagnóstico.</div><div>3b3. Se continúa el caso de uso desde el punto 3.</div></div>			

Caso de uso	#15 Modificar nota	Actor principal	Usuario
Precondiciones	El usuario ha de estar logueado en el sistema.		
Disparador	El usuario quiere modificar una nota.		
Escenario principal de uso			
<div>1. El usuario indica que desea realizar cambios en la nota.</div> <div>2. El sistema solicita al usuario los datos de la nueva nota.</div> <div>3. El usuario realiza los cambios en la nota e indica al sistema que desea guardar los cambios.</div> <div>4. El sistema indica al usuario que los cambios se han guardado correctamente.</div>			
Extensiones			
<div>1b. El usuario no ha seleccionado un diagnóstico.</div> <div>1b1. El sistema indica al usuario que ha de seleccionar un diagnóstico.</div> <div>1b2. El usuario selecciona un diagnóstico.</div> <div>1b3. Se continúa el caso de uso desde el punto 3.</div>			

Caso de uso	#16 Borrar nota	Actor principal	Usuario
Precondiciones	El usuario ha de estar logueado en el sistema.		
Disparador	El usuario quiere borrar una nota.		
Escenario principal de uso			
<div>1. El usuario indica al sistema que desea eliminar una nota.</div> <div>2. El sistema pregunta al usuario si está seguro de querer eliminar la nota.</div> <div>3. El usuario indica que sí quiere eliminar la nota.</div> <div>4. El sistema muestra el listado de entradas sin la nota.</div>			
Extensiones			
<div>2a. El usuario decide no borrar la nota.</div> <div>2a1. El usuario indica que no quiere borrar la nota.</div> <div>2a2. El sistema muestra al usuario de nuevo la nota y finaliza el caso de uso.</div>			

Usuarios: Gestión de agenda

Caso de uso	#25 Crear cita	Actor principal	Usuario
Precondiciones	El usuario ha de estar logueado en el sistema.		
Disparador	El usuario quiere crear una cita a un paciente.		
Escenario principal de uso			
<ol style="list-style-type: none">1. El usuario indica que quiere añadir una cita a un paciente.2. El sistema solicita al usuario los datos para crear una nueva cita.3. El usuario proporciona los datos e indica al sistema que desea guardar los cambios.4. El sistema muestra el listado de citas con la nueva cita que acaba de crear el usuario.			
Extensiones			
<ol style="list-style-type: none">3a. El usuario ha seleccionado una fecha ya pasada.<ol style="list-style-type: none">3a1. El sistema indica al usuario que ha de seleccionar una fecha que aún no haya ocurrido.3a2. El usuario selecciona una nueva fecha.3a3. Se continúa el caso de uso desde el punto 3.3b. El usuario encargado de realizar esa cita no tiene turno en la fecha seleccionada.<ol style="list-style-type: none">3b1. El sistema indica al usuario que el trabajador no tiene turno en esa fecha.3b2. El usuario selecciona una nueva fecha.3b3. Se continúa el caso de uso desde el punto 3.			

Caso de uso	#26 Modificar cita	Actor principal	Usuario
Precondiciones	El usuario ha de estar logueado en el sistema.		
Disparador	El usuario quiere modificar una cita a un paciente.		
Escenario principal de uso			
<div>1. El usuario indica que desea realizar cambios en la cita.</div> <div>2. El sistema solicita al usuario los datos de la nueva cita.</div> <div>3. El usuario realiza los cambios en la cita e indica al sistema que desea guardar los cambios.</div> <div>4. El sistema indica al usuario que los cambios se han guardado correctamente.</div>			
Extensiones			
<div>3a. El usuario ha seleccionado una fecha ya pasada.</div> <div><div>3a1. El sistema indica al usuario que ha de seleccionar una fecha que aún no haya ocurrido.</div><div>3a2. El usuario selecciona una nueva fecha.</div><div>3a3. Se continúa el caso de uso desde el punto 3.</div></div> <div>3b. El usuario encargado de realizar esa cita no tiene turno en la fecha seleccionada.</div> <div><div>3b1. El sistema indica al usuario que el trabajador no tiene turno en esa fecha.</div><div>3b2. El usuario selecciona una nueva fecha.</div><div>3b3. Se continúa el caso de uso desde el punto 3.</div></div>			

Usuarios: Gestión de recaptación de información

Caso de uso	#31 Generar listado de pacientes	Actor principal	Usuario
Precondiciones	El usuario ha de estar logueado en el sistema.		
Disparador	El usuario quiere generar un listado personalizado de pacientes.		
Escenario principal de uso			
<div>1. El usuario indica al sistema que desea generar un nuevo listado.</div> <div>2. El sistema solicita al usuario los datos del listado que desea generar.</div> <div>3. El usuario proporciona los datos de la búsqueda que desea realizar e indica al sistema que desea comenzar la búsqueda.</div> <div>4. El sistema muestra al usuario el listado de pacientes.</div>			
Extensiones			
<div>4a. El usuario indica que desea ver más información de los pacientes.</div> <div>4a1. El usuario indica qué información desea que tenga el listado.</div> <div>4a2. El sistema muestra al usuario el listado con la nueva información.</div> <div>4b. El usuario quiere exportar el listado.</div> <div>4b1. El usuario indica que desea exportar el listado y el formato.</div> <div>4b2. El sistema exporta el listado en el formato indicado por el usuario.</div>			

7.3. Modelo conceptual de datos

A continuación, se definirá el modelo conceptual de datos. Estos nos ayudan a representar todas las entidades que participan en nuestro sistema y ver cómo se relacionan entre ellas con mayor precisión. Los modelos de datos nos sirven para detectar las clases de nuestro sistema, los atributos de las clases y las restricciones de integridad, gráficas o textuales de nuestros datos.

7.3.1. Esquema conceptual de datos

El esquema conceptual de datos es una herramienta que nos permite ver de manera gráfica las relaciones entre las clases de objetos de nuestro sistema. Es importante remarcar que existen dos tipos de esquemas de datos: el conceptual y el lógico. La diferencia entre los dos es que el esquema lógico de datos busca construir un esquema que pueda ser utilizado posteriormente para construir la base de datos, mientras que el conceptual busca describir los requisitos funcionales del sistema.

En nuestro caso realizaremos el esquema conceptual de datos. Este esquema tiene como objetivo representar los problemas y las entidades del mundo real en un marco conceptual que pueda ser entendido por todos nuestros *stakeholders*, independientemente de si estos son expertos o no [27]. En la imagen 7 podemos observar el esquema de datos conceptual de nuestro sistema, representado en UML¹⁰.

¹⁰ UML: El Lenguaje Unificado de Modelado es un estándar utilizado para representar arquitecturas, diseños o implementaciones de sistemas software complejos.

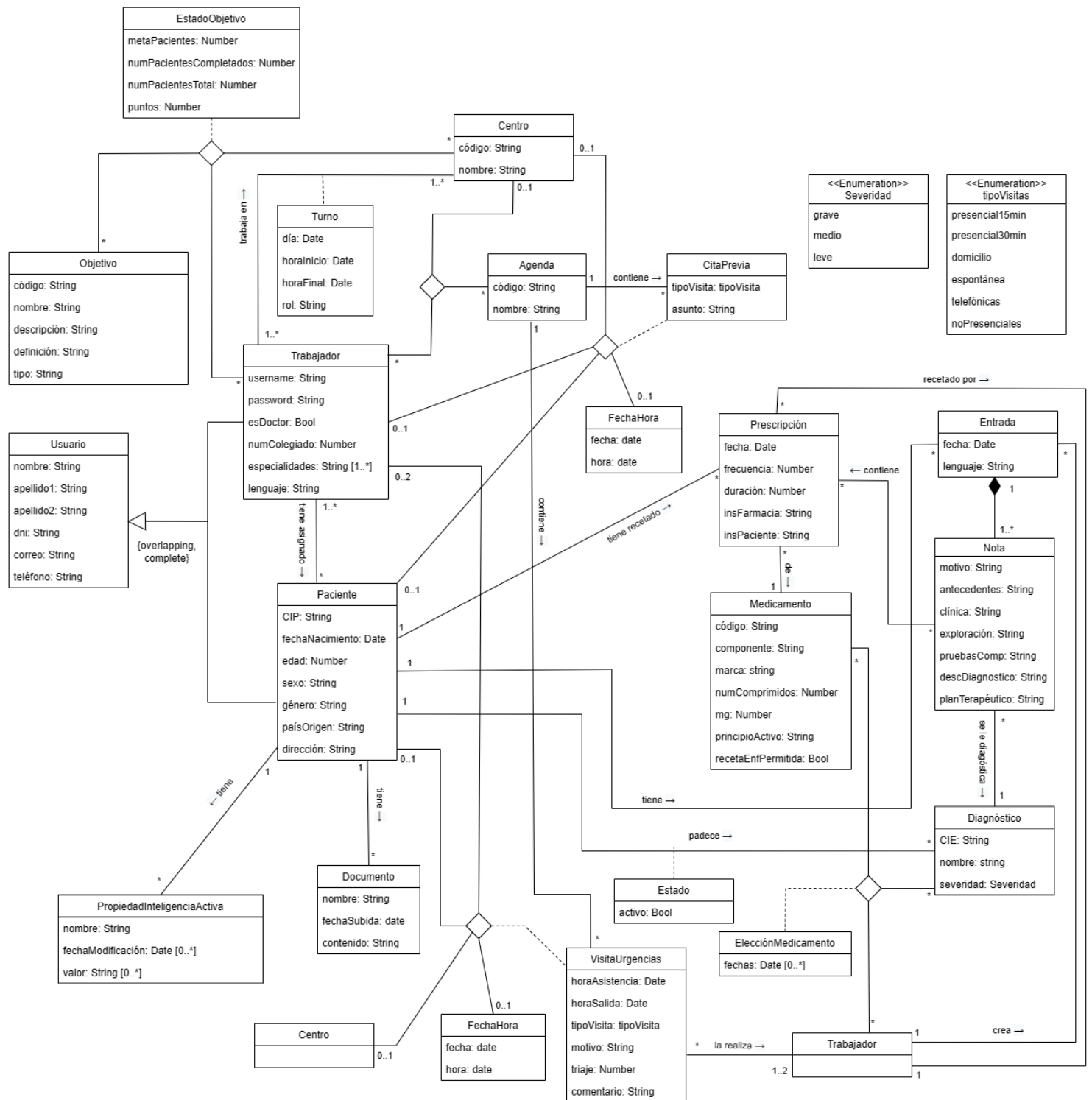


Imagen 6. Esquema conceptual de datos

7.3.2. Restricciones de integridad

Claves externas: (Objetivo, código), (Centro, código), (Usuario, DNI), (Prescripción, fecha + CIP), (Entrada, fecha + CIP), (Medicamento, código), (Diagnóstico, CIE), (Agenda, código), (Documento, nombre + fechaSubida), (PropiedadInteligenciaActiva, nombre + CIP), (FechaHora, fecha + hora)

RT1: En el caso de que haya dos personas encargadas de una visita de urgencias, solo uno de los dos podrá ser y ha de ser enfermero.

RT2: El trabajador asignado a una cita previa o a una visita de urgencias ha de tener un turno en ese centro a esa hora.

RT3: Si el trabajador es enfermero, no puede recetar un medicamento que no tenga permitido.

RT4: Un paciente no puede estar en dos citas, dos visitas de urgencia, o en una cita y en una visita de urgencias a la vez.

RT5: Un trabajador no puede estar en dos citas previas, dos visitas de urgencia, o en una cita previa y en una visita de urgencias a la vez.

RT7: El trabajador y el paciente de una cita previa o una visita de urgencias no puede ser el mismo.

RT8: Un trabajador no puede tenerse asignado a sí mismo como paciente.

7.3.3. Descripción de las clases

A continuación, se definirá cada una de las clases y los atributos que participan en el modelo conceptual de datos.

Usuario: Representa todos los usuarios del sistema.

- nombre: Nombre del usuario.
- apellido1: Primer apellido del usuario.
- apellido2: Segundo apellido del usuario.
- DNI: Documento nacional de identidad del usuario.
- correo: Correo electrónico del usuario.
- teléfono: Teléfono del usuario.

Trabajador: Representa a los doctores y enfermeros del sistema.

- username: Código oficial del trabajador que servirá para loguearse.
- password: Contraseña del trabajador.
- esDoctor: Booleano que indica si el trabajador es doctor o enfermero.
- numColegiado: Número compuesto de 9 dígitos que sirve para identificar al trabajador dentro del colegiado.
- especialidades: Lista con todas las especialidades de las que puede ejercer el trabajador.
- lenguaje: Lenguaje escogido por el usuario para trabajar.

Paciente: Representa a los pacientes del sistema.

- CIP: Código de Identificación de la administración sanitaria de cada paciente.
- fechaNacimiento: Fecha de nacimiento del paciente.
- edad: Edad del paciente.
- sexo: Sexo del paciente.
- género: Género del paciente.
- paísOrigen: País origen del paciente.
- dirección: Dirección del hogar de referencia del paciente.

Objetivo: Representa los objetivos que han de lograr los trabajadores.

- código: Código del objetivo.
- nombre: Nombre del objetivo.
- descripción: Descripción breve del objetivo.
- definición: Requisitos de un objetivo para ser cumplido.
- tipo: Categoría que se utiliza para clasificar a los objetivos.

EstadoObjetivo: Representa el progreso de un objetivo.

- metaPacientes: Número de pacientes mínimo para completar el objetivo.
- numPacientesCompletados: Número de pacientes que tienen el tratamiento especificado en la definición del objetivo.
- numPacientesTotal: Número de pacientes que padecen los síntomas especificados en la definición del objetivo.
- puntos: Número de puntos obtenidos al progresar en el objetivo.

Centro: Representa los centros que hay en el sistema.

- código: Código que identifica al centro.
- nombre: Nombre del centro.

Agenda: Representa las agendas de los trabajadores del sistema.

- código: Código que identifica a la agenda.
- nombre: Nombre de la agenda.

Turno: Representa los turnos que tienen asignados los trabajadores.

- día: Día en el que empieza el turno.
- horaInicio: Hora en la que da comienzo el turno.
- horaFinal: Hora en la que finaliza el turno.
- rol: Función que tiene asignado el trabajador a lo largo de ese turno.

CitaPrevia: Representa las citas previas de los pacientes que hay en el sistema.

- tipoVisitas: Especifica si el tipo de visita es presencial, telefónica o a domicilio.
- asunto: Especifica el motivo de la visita.

Prescripción: Representa las prescripciones de los medicamentos que hay asignadas a los pacientes.

- fecha: Fecha que indica el inicio del tratamiento con cierto fármaco.
- frecuencia: Número de horas que han de pasar entre la ingestión o aplicación de un medicamento.
- duración: Número de días que ocupa el tratamiento.
- insFarmacia: Instrucciones definidas para el farmacéutico.
- insPaciente: Instrucciones definidas para el paciente.

Medicamento: Representa los medicamentos que pueden recetar los trabajadores.

- código: Código que identifica al medicamento.
- componente: Nombre del medicamento.
- marca: Nombre de la marca que produce el medicamento.
- numComprimidos: Número de comprimidos que vienen dentro de la caja del medicamento.
- mg: Número de miligramos que vienen por comprimido.
- principioActivo: Componente que produce el efecto terapéutico del medicamento.
- recetaEnfPermitida: Booleano que indica si el medicamento puede ser recetado por un enfermero.

Entrada: Representa las entradas que hay en el sistema. Se suele crear una entrada por visita para dejar constancia de algún diagnóstico al paciente.

- fecha: Día y hora en la que se crea la entrada.
- lenguaje: Lenguaje con el que se ha escrito la entrada.

Nota: Representa las notas que hay en una entrada. Una nota especifica la información necesaria para describir una consulta.

- motivo: Motivo de la visita del paciente o el motivo de la entrada.
- antecedentes: Descripción de los antecedentes personales y familiares del paciente.
- clínica: Descripción de los síntomas de un paciente.
- exploración: Descripción de la exploración al paciente.
- pruebasComp: Descripción de las pruebas complementarias realizadas al paciente y sus resultados destacados.
- descDiagnostico: Detalles destacables del diagnóstico que padece el paciente.
- planTerapéutico: Descripción detallada del tratamiento asignado al paciente.

Diagnóstico: Representa los diagnósticos que un trabajador puede diagnosticar a un paciente.

- CIE: Abreviatura de Clasificación Internacional de las Enfermedades. Sirve para identificar cada uno de los posibles diagnósticos.
- nombre: Nombre del diagnóstico.
- severidad: Indica el grado de severidad de un diagnóstico.

Estado: Representa el estado de un diagnóstico del paciente.

- activo: Indica si el paciente sigue padeciendo o no un diagnóstico.

ElecciónMedicamento: Representa las veces que un trabajador ha recetado un medicamento para un diagnóstico en específico.

- fechas: Lista de fechas en las que el trabajador ha recetado un cierto medicamento para un diagnóstico.

VisitaUrgencias: Representa las visitas de urgencias que hay en el sistema.

- horaAsistencia: Hora en la que se atiende al paciente.
- horaSalida: Hora en la que se cierra la visita.
- tipoVisita: Especifica el tipo de la visita.
- motivo: Motivo por el cual el paciente ha acudido a urgencias.
- triaje: Número que indica la prioridad del paciente para ser atendido.
- comentario: Comentario extra que puede escribir el personal administrativo sobre un paciente.

PropiedadInteligenciaActiva: La inteligencia activa representa la recolección de datos relacionados con la salud que hay sobre un paciente, como por ejemplo el peso o su estatura. La propiedad representa cada una de las secciones de la inteligencia activa.

- nombre: Nombre de la propiedad de la inteligencia activa.
- fechaModificación: Vector que contiene las fechas en la que se ha modificado algún valor de la propiedad activa.
- valor: Vector que contiene todos los valores que ha tenido una propiedad de la inteligencia activa.

7.4. Modelos de comportamiento

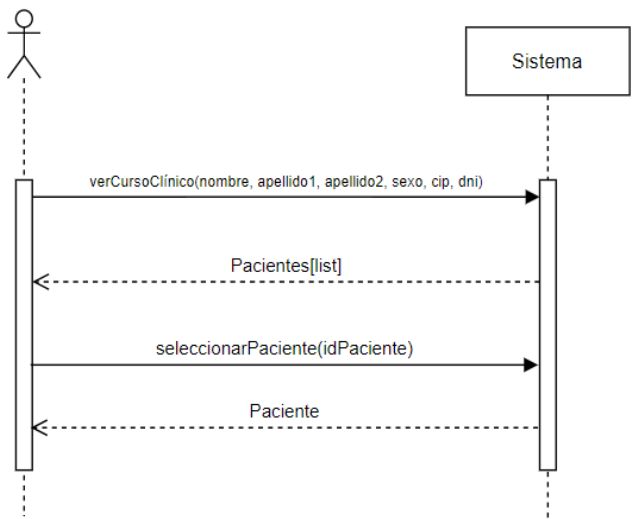
Para definir cómo interactúa el usuario con el sistema, se detallará un modelo de comportamiento para cada caso de uso. En cada caso, se especificará mediante un diagrama de secuencia las operaciones que puede realizar el usuario y los efectos que estas producirán en nuestro sistema.

Para cada acción se especificará la precondition (requisitos previos para poder realizar la operación), la postcondition (requisitos que se han de cumplir tras ejecutar la operación) y el body (la información que recibiremos de parte del sistema en cada operación).

En nuestros diagramas de secuencia van a aparecer 3 entidades: el usuario encargado de empezar con la operación y que tomará el papel de actor principal, el sistema en representación del *software* encargado de ejecutar las operaciones y MyMemory representando la API que participa en el caso de uso de traducción de entradas.

De igual manera que con los casos de uso, se mostrará únicamente un modelo de comportamiento de cada tipo a causa de su similitud. El resto se podrán encontrar en el apartado Anexo 4. Modelos de comportamiento.

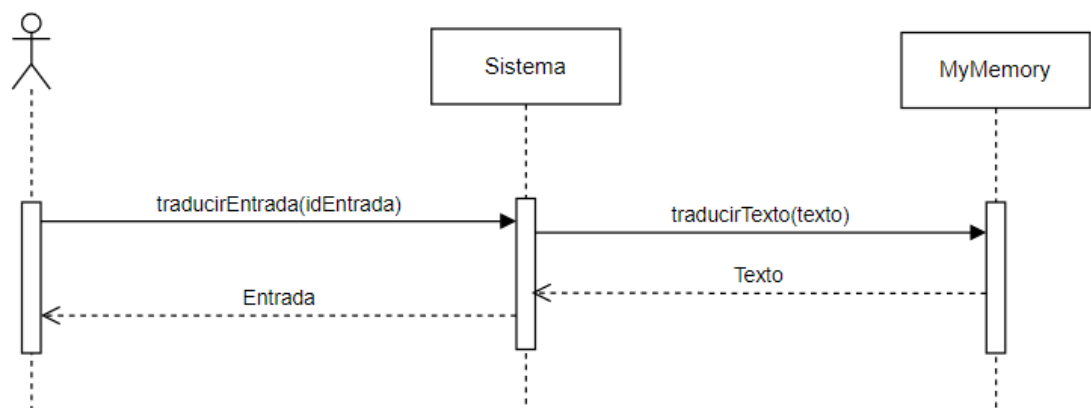
Ver el curso clínico de un paciente



Caso de uso	verCursoClinico(nombre: String, apellido1: String, apellido2: String, sexo: String, cip: String, dni:String)
Precondiciones	
Postcondiciones	
Body	Se devuelve una lista con los pacientes que coinciden con la búsqueda o un código de error.

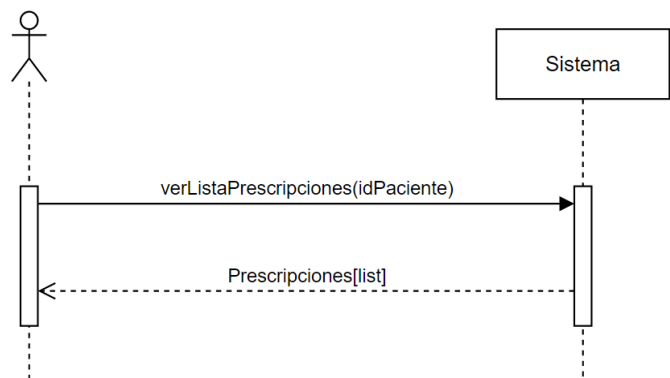
Caso de uso	seleccionarPaciente(idPaciente: String)
Precondiciones	idPaciente existe en el sistema.
Postcondiciones	
Body	Se devuelve el paciente o un código de error.

Traducir la entrada de un paciente



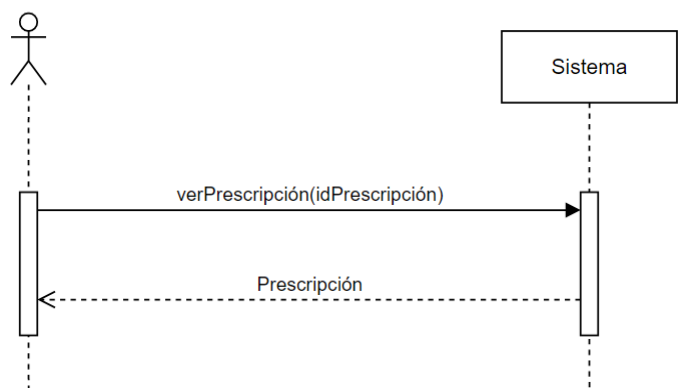
Caso de uso	traducirEntrada(idEntrada: String)
Precondiciones	idEntrada existe en el sistema.
Postcondiciones	El sistema invoca la operación traducirTexto() del sistema MyMemory.
Body	Se devuelve la entrada traducida o un código de error.

Ver lista de prescripciones de un paciente



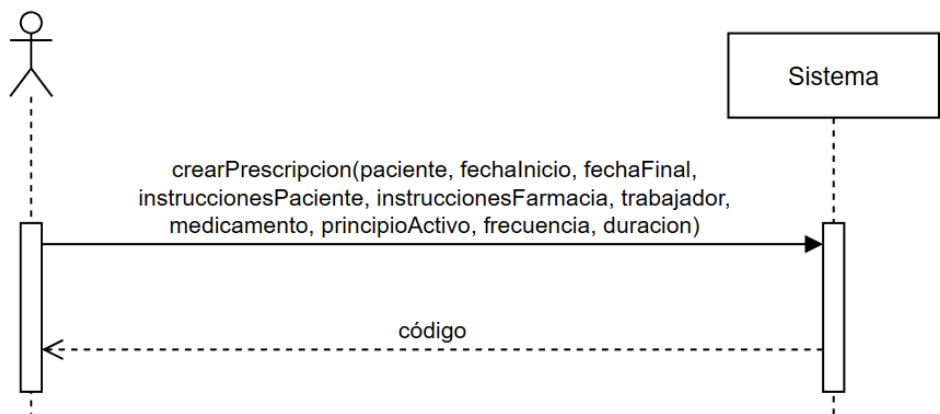
Caso de uso	verListaPrescripciones(idPaciente: String)
Precondiciones	idPaciente existe en el sistema.
Postcondiciones	
Body	Se devuelve la lista de prescripciones o un código de error.

Ver prescripción de un paciente



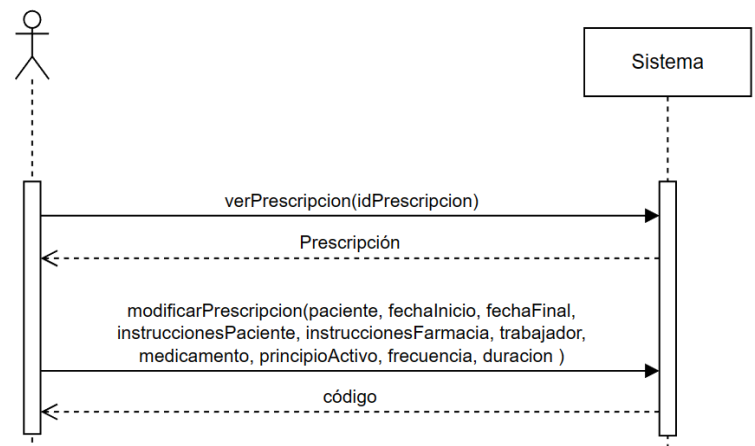
Caso de uso	verPrescripción(idPrescripción: String)
Precondiciones	idPrescripción existe en el sistema.
Postcondiciones	
Body	Se devuelve la prescripción o un código de error.

Crear una prescripción a un paciente



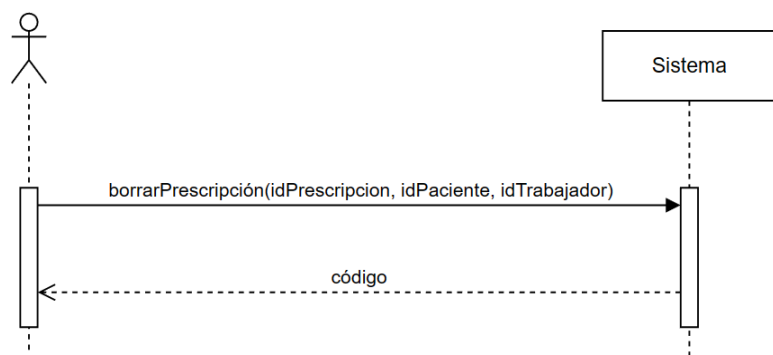
Caso de uso	crearPrescripcion(paciente: String, fechaInicio: Date, fechaFinal: Date, instruccionesPaciente: String, instruccionesFarmacia: String, trabajador: String, medicamento: String, principioActivo: String, frecuencia: String, duración: Number)
Precondiciones	paciente existe en el sistema. trabajador existe en el sistema. medicamento existe en el sistema.
Postcondiciones	Se crea una prescripción con los datos introducidos, se añade la prescripción a la lista de prescripciones del paciente, se añade el medicamento a la lista de medicamentos escogidos por el trabajador y se actualizan los objetivos del trabajador si estos se ven afectados por la nueva prescripción.
Body	Se devuelve el código de aceptación si se ha creado correctamente la prescripción.

Modificar una prescripción de un paciente



Caso de uso	modificarPrescripcion(paciente: String, fechaInicio: Date, fechaFinal: Date, instruccionesPaciente: String, instruccionesFarmacia: String, trabajador: String, medicamento: String, principioActivo: String, frecuencia: String, duración: Number)
Precondiciones	paciente existe en el sistema. trabajador existe en el sistema. medicamento existe en el sistema.
Postcondiciones	Se actualiza la información de la prescripción y se actualizan los objetivos del trabajador si estos se ven afectados por la prescripción.
Body	Se devuelve el código de aceptación si se ha modificado correctamente la prescripción.

Borrar una prescripción de un paciente



Caso de uso	borrarPrescripcion(idPrescripcion: String, idPaciente: String, idTrabajador: String)
Precondiciones	idPrescripcion existe en el sistema. idPaciente existe en el sistema. idTrabajador existe en el sistema.
Postcondiciones	Se borra la prescripción de la lista de prescripciones del paciente, y se actualizan los objetivos del trabajador.
Body	Se devuelve el código de aceptación si se ha borrado la prescripción correctamente.

7.5. Historias de usuario

Las historias de usuario y los casos de uso son herramientas muy utilizadas en las metodologías agile. Los casos de uso se centran en definir todas las interacciones entre el usuario y el sistema, mientras que las historias de usuario buscan representar las necesidades del usuario y qué tenemos que cumplir para poder considerarlas como satisfechas.

Las historias de usuario son explicaciones informales de una funcionalidad del sistema que se encuentran escritas desde el punto de vista del usuario final. Para realizar correctamente una historia de usuario, tenemos que asegurarnos de cumplir con las tres C de: *card*, conversación y confirmación. En la *card* (tarjeta) se describe de manera informal el objetivo del usuario. En la conversación el *Product Owner* (representante de los *stakeholders*) y el equipo de desarrollo del *software* pactan los detalles. Finalmente, en la confirmación se describen las condiciones que confirman si la historia se ha desarrollado con éxito.

Para asegurarnos de que nuestras historias de usuario tienen la calidad mínima requerida seguiremos el modelo INVEST. Este modelo desarrollado por Bill Wake [\[28\]](#) nos permite asegurarnos de que nuestro *software* cumple con las necesidades reales de nuestro cliente. Para ello, una historia de usuario debe cumplir con las siguientes características:

- I - Independiente: Una historia de usuario no ha de depender del resto de historias, lo que permite poder clasificarlas según prioridad.
- N - Negociable: Para desarrollar una historia se ha de haber mantenido un diálogo entre el equipo de desarrollo y el cliente.
- V - Valiosa: Cada historia de usuario debe añadir valor al software.
- E - Estimable: El esfuerzo requerido para cada historia de usuario ha de poder expresarse de manera objetiva.
- S - Pequeña (small): Una historia de usuario ha de ser lo suficientemente pequeña como para caber en un *sprint*.
- T - Testable: Cada historia de usuario ha de poder ser validada para comprobar si se ha resuelto con éxito.

Por otra parte, los criterios de aceptación también han de seguir una serie de características:

- Atomicidad: Solo pueden tener 2 resultados únicos: cumplidos o no cumplidos.
- No ambiguos: Deben ser interpretados de la misma manera por todas las personas.
- Verificables: El cliente ha de poder verificar que están cumplidos.
- Completos: Todos los criterios de aceptación juntos han de incluir todos los requisitos funcionales.

De igual manera que en el apartado anterior, se mostrará únicamente una cuantas historias de usuario a causa de su similitud. El resto se podrán encontrar en el apartado Anexo 5. Historias de usuario.

Historia de usuario	#6 Ver curso clínico
Descripción	Como usuario, quiero poder buscar a un paciente y poder ver su curso clínico para poder acceder a toda su información.
Criterios de aceptación	Se debe poder buscar a un paciente según nombre, sexo, dni, o número de tarjeta sanitaria. Al buscar a un paciente, debe aparecer una lista con todos los pacientes que encajan con la búsqueda. Los pacientes deberán estar ordenados en sentido ascendente según su nombre completo. Se debe poder ver el curso clínico de un paciente.

Historia de usuario	#10 Traducir entrada
Descripción	Como usuario, quiero poder traducir las entradas al idioma que tengo seleccionado para poder entender sin problemas la entrada de un paciente si esta se encuentra en otro lenguaje.
Criterios de aceptación	Si el lenguaje del usuario es diferente al idioma de una entrada, se debe poder traducir una entrada. Una vez se ha traducido una entrada, al ver o modificar una nota deben aparecer todos los apartados traducidos. Si le das de nuevo al botón de traducir, la entrada debe volver al lenguaje original.

Historia de usuario	#18 Ver listado de prescripciones
Descripción	Como usuario, quiero poder ver el listado de todas las recetas electrónicas de un paciente para ver qué medicamentos tiene recetados.
Criterios de aceptación	En el curso clínico, debe aparecer una lista con los principios activos recetados a un paciente. En la pestaña del listado de prescripciones, debe aparecer una lista con todas las prescripciones vigentes a ese paciente. Se debe visualizar el listado de alergias que tiene el usuario. Se debe resaltar el nombre de un medicamento en color rojo si la prescripción activa de un medicamento coincide con alguna alergia.

Historia de usuario	#19 Ver prescripción
Descripción	Como usuario, quiero poder ver una prescripción de un paciente para visualizar su información detallada.
Criterios de aceptación	El nombre del medicamento debe parecer en rojo si coincide con la alergia de un paciente. Se debe visualizar toda la información relacionada con la prescripción. Se debe permitir al usuario volver a la pestaña anterior.

Historia de usuario	#20 Crear prescripción
Descripción	Como usuario, quiero poder crear una prescripción para poder realizar recetas electrónicas a un paciente.
Criterios de aceptación	<p>Se debe poder crear una nueva prescripción.</p> <p>Si el usuario tiene una nota abierta con un diagnóstico seleccionado, deben aparecer recomendados los medicamentos que el usuario suele recetar para el diagnóstico seleccionado.</p> <p>Se debe poder buscar un medicamento tanto por nombre como por principio activo.</p> <p>Un enfermero no debe ver en el buscador los medicamentos que no puede recetar.</p> <p>Al seleccionar un medicamento, debe aparecer su nombre y la posología que tiene por defecto automáticamente.</p> <p>Se debe poder importar las instrucciones del medicamento.</p> <p>Si no se ha seleccionado ningún medicamento y el usuario crea la prescripción, debe aparecer un mensaje que notifique al usuario de que es obligatorio seleccionar un medicamento.</p> <p>Se debe permitir al usuario volver a la pestaña anterior.</p>

Historia de usuario	#21 Modificar prescripción
Descripción	Como usuario, quiero poder modificar una prescripción en caso de que me haya equivocado o desee añadir más información.
Criterios de aceptación	<p>Se debe poder modificar una prescripción.</p> <p>Al guardar los cambios, debe aparecer el listado de prescripciones con los cambios realizados.</p> <p>Si no se ha seleccionado ningún medicamento y el usuario modifica la prescripción, debe aparecer un mensaje que notifique al usuario de que es obligatorio seleccionar un medicamento.</p> <p>Se debe permitir al usuario volver a la pestaña anterior.</p>

Historia de usuario	#22 Borrar prescripción
Descripción	Como usuario, quiero borrar una prescripción en caso de haber cometido un error o de que considere que ya no es necesaria para el paciente.
Criterios de aceptación	<p>Desde el listado de prescripciones, debe aparecer un botón que permita borrar la prescripción.</p> <p>Al clicar al botón aparece un mensaje para asegurarnos de que el usuario quiere borrar la prescripción.</p> <p>Al borrar una prescripción, debe aparecer un mensaje de confirmación para asegurarnos de que el usuario quiere borrar la prescripción.</p>

8. Diseño

Una vez visto el apartado de especificación, ya tenemos una visión más clara de cuáles son las fronteras de nuestro proyecto y podemos empezar a definir la estructura y el funcionamiento interno de todos los componentes que participarán en nuestro sistema. En este apartado se detalla la arquitectura del sistema, así como todas las tecnologías y patrones de diseño que se utilizarán tanto en el front-end como en el back-end.

8.1. Arquitectura de 3 capas

La arquitectura dentro del ámbito de las páginas web nos permite diseñar un plan para definir cómo todos los componentes van a interactuar entre ellos. Mostrará cómo viajan los datos desde una base de datos hasta el navegador, manteniendo una buena escalabilidad y un buen mantenimiento. Los principales componentes que podemos encontrar en una arquitectura web son:

- Navegador: también conocido como *client-side* o front-end. Es el componente encargado de interactuar con el usuario y de recibir o validar todos sus inputs.
- Servidor web: también conocido como *server-side* o back-end. Es el componente encargado de gestionar la lógica de negocio¹¹ y de procesar las peticiones HTTP que necesite el *client-side*.
- Base de datos: componente encargado de almacenar toda la información requerida para la aplicación web.

En arquitecturas tradicionales se utiliza una arquitectura conocida como la arquitectura de 2 capas, divididas por el *client-side* y el *server-side*. La parte negativa de esta arquitectura es que su rendimiento se ve afectado cuando hay un gran número de usuarios utilizando el sistema. Es por eso, que en este proyecto se utilizará una arquitectura de 3 capas. En esta arquitectura se dividen los componentes en: la capa de presentación o *client layer*, la capa de aplicación o *business logic layer* y la capa de datos. En la imagen 8 se puede observar cómo interactúan las 3 capas con todos los componentes.

¹¹ Lógica de negocio: parte del sistema que se encarga de transformar el mundo real en información que pueda ser almacenada y modificada.

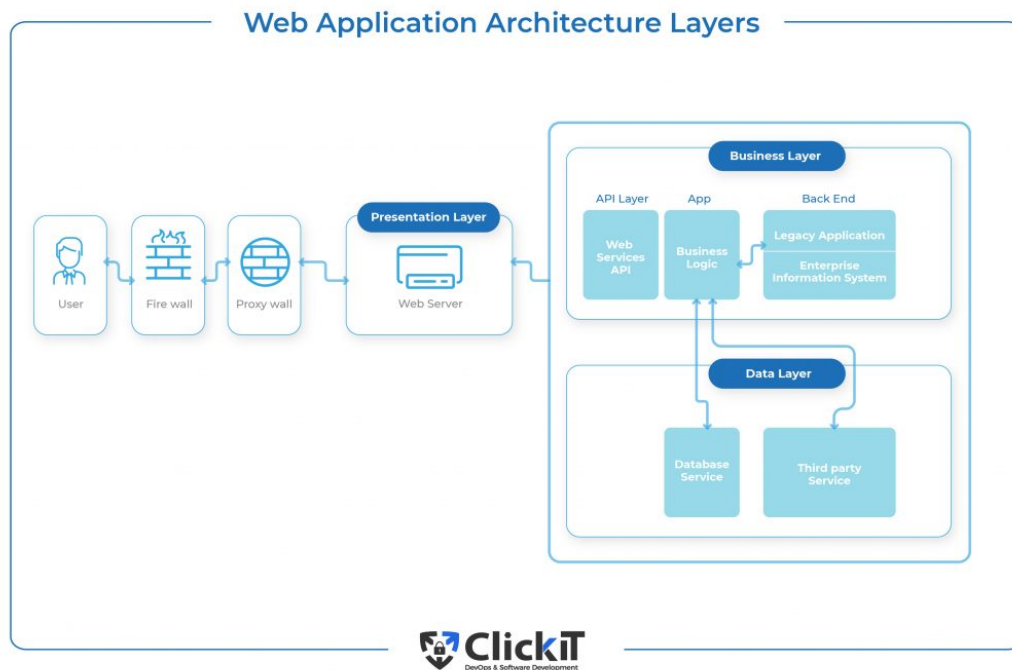


Imagen 7. Arquitectura de 3 capas [29]

En la arquitectura de 3 capas, cada capa ocupará una responsabilidad diferente dentro del sistema y estarán separadas físicamente en un servidor distinto. La capa de presentación es la encargada de recibir todos los inputs del usuario y enviar las peticiones a la capa de aplicación. Esta se encarga de procesar su petición y se comunica si es necesario con la base de datos para poder devolverle a la capa de presentación la respuesta de su petición. Tener un paso intermedio nos permite poder tener mayor escalabilidad, debido a que la capa de aplicación se puede hostear en múltiples máquinas para que estas trabajen en paralelo. Este concepto se le conoce como escalabilidad horizontal.

Además, esta arquitectura nos proporciona mayor seguridad, ya que la capa de presentación solo puede acceder a los datos almacenados en la base de datos a través de las peticiones API que se encuentran controladas por la capa de aplicación. Esto implica que se puede controlar en todo momento qué datos se pasan a la capa de presentación y quién tiene la autorización para poder acceder a esos datos.

Por último, dividir todo en 3 capas nos permite una mayor reusabilidad y mantenimiento. En el caso de que se quiera cambiar la implementación de algún componente, esto no afectará al funcionamiento de los demás, ya que el cómo funciona cada componente es completamente transparente entre las diferentes capas y solo se ha de asegurar que los contratos de las peticiones API se mantengan. Asimismo, dividir la base de datos del lado del servidor nos permite que esta pueda ser accedida simultáneamente por diferentes aplicaciones al mismo tiempo. Esta característica es muy importante para nuestro sistema, debido a que será imprescindible que otros sistemas sanitarios independientes al nuestro sean capaces de acceder a la base de datos.

8.2. Patrón de diseño MVC

El patrón de diseño MVC es un patrón arquitectónico que divide el sistema en 3 componentes lógicos formados por el modelo, la vista y el controlador. Fue creado por Trygve Reenskaug [\[30\]](#), en un principio para crear una aplicación de escritorio, aunque actualmente el MVC se utiliza también para aplicaciones de móvil o aplicaciones web como la nuestra.

Para empezar, el patrón MVC dividirá la capa de presentación en dos componentes llamados vista y controlador, y la capa de aplicación en el componente modelo. El componente modelo es el nivel más bajo de este patrón y actúa como intermediario entre el controlador de la capa de presentación y la base de datos. Se encarga de gestionar las peticiones para poder obtener, añadir, modificar o borrar datos, y de controlar toda la lógica de negocio. El componente vista se encarga de generar la interfaz de usuario para que este interactúe con el sistema. Y finalmente, el controlador será el encargado de gestionar los eventos que reciba de parte del componente vista y de solicitar la información necesaria al componente modelo. En la imagen 9 podemos observar cómo el controlador se relaciona con la vista y el modelo.

MVC Architecture Pattern

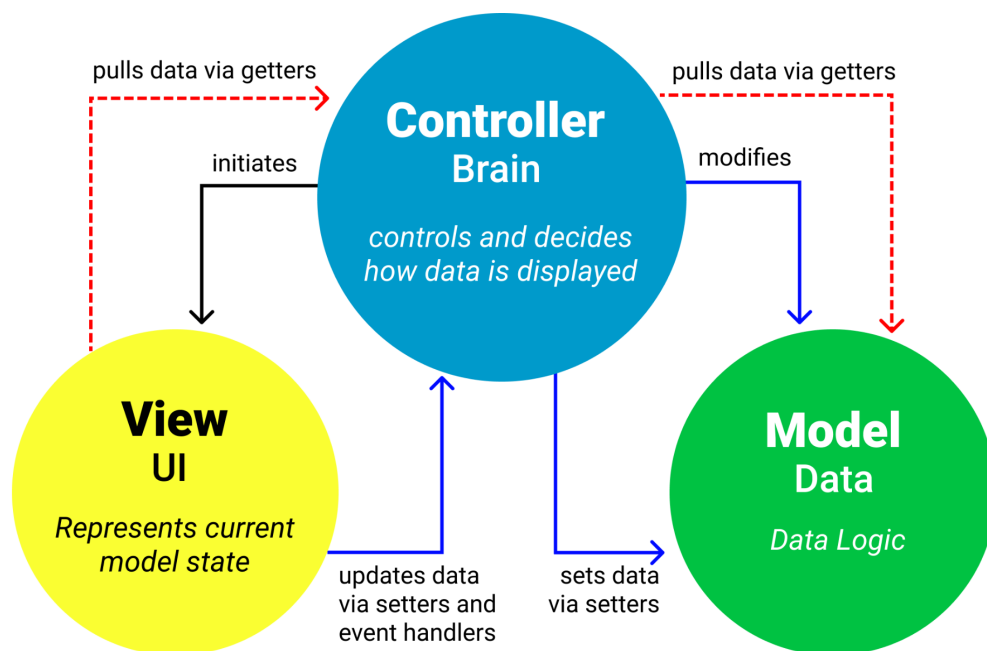


Imagen 8. Patrón de diseño MVC [\[31\]](#)

Este patrón nos facilita la reusabilidad de cada componente y su mantenimiento. Consigue que cada componente sea encapsulado de manera independiente con su propia lógica y aporta mayor flexibilidad. Aparte de esto, también facilita el testeo de nuestros componentes, ya que nos permite testear sus funcionalidades de manera individual.

8.3. Arquitectura lógica

En la imagen 11 podemos observar como quedaría nuestro sistema con el patrón MVC aplicado y una arquitectura de 3 capas.

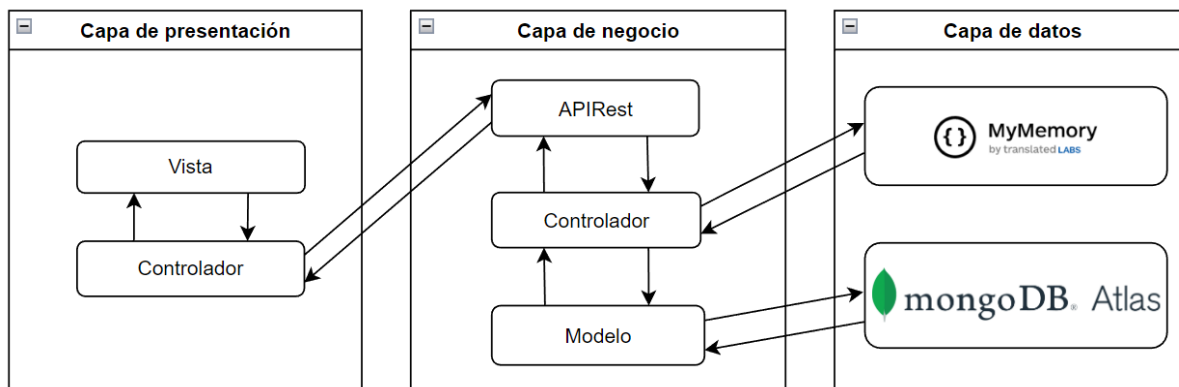


Imagen 9. Arquitectura del sistema

Como podemos ver gracias a la imagen 11, la arquitectura lógica se divide según las 3 capas vistas en el apartado 8.1. La capa de presentación, compuesta por los componentes vista y controlador, se implementará utilizando React (hablaremos más en profundidad de esta tecnología en el apartado 9). Esta se comunicará con la capa de negocio a través de las api calls que proporciona la API REST, que estará formada por nuestros endpoints¹² y middlewares¹³, y pasará la petición HTTP a los controladores encargados de la lógica de negocio para que se encarguen de procesarla.

En la capa de datos podemos ver nuestros dos sistemas externos. En primer lugar la API gratuita de MyMemory, que se encargará de hacer las traducciones de texto para permitir el multilinguaje, y por otro lado MongoDB atlas, que hará de *host* de nuestra base de datos. Para comunicarnos con esta última, se utilizará el componente Modelo, que se encargará de realizar las peticiones o los cambios pertinentes a la base de datos a través de Mongoose, una librería de Node.js que nos permite escribir consultas para la base de datos de MongoDB.

¹² **Endpoint:** extremo de una conexión API donde se recibe la petición.

¹³ **Middleware:** Capa de abstracción que funciona como intermediario entre las diferentes capas de software y nos permite acceder a los datos de una petición HTTP.

8.4. Patrones de diseño

Los patrones de diseño son técnicas reutilizables que ayudan a los desarrolladores a solucionar problemas comunes que surgen a la hora de desarrollar cualquier tipo de *software*. En el apartado 9.2 ya vimos el patrón de diseño MVC, un patrón que será vital para la estructuración de nuestro proyecto. Pero en este apartado hablaremos del resto de patrones de diseño que aparecerán durante su implementación. En este apartado únicamente definiremos cada uno de los patrones. En el apartado de implementación veremos cuándo y cómo los vamos a aplicar.

8.4.1. Module pattern

En Javascript el concepto de clase es un poco distinto al que conocemos en otros lenguajes de programación como por ejemplo Java. En Javascript, las clases no son más que una función que actúa como plantilla donde puedes definir datos y métodos para crear objetos. No permite conceptos como herencia múltiple, interfaces o clases abstractas. Es en parte por lo que han surgido variaciones de Javascript como TypeScript que las añaden. A pesar de esto, Javascript también es capaz de mantener un orden gracias al module pattern. Un módulo no es más que código encapsulado que permite tener objetos públicos [\[32\]](#). Ahora bien, este objeto puede ser lo que nosotros queramos. Un array, una función, un objeto con atributos, o incluso un objeto con atributos y métodos. Como en Javascript cada vez que se crea una función se crea un nuevo *scope*¹⁴ local, todos los atributos y métodos que se creen dentro serán “privados”. Es decir, no podrán ser accedidos desde fuera de este *scope* a menos que se especifique como objeto público a través de la *keyword*¹⁵ “export”.

8.4.2. Middleware pattern

Las funciones middleware [\[33\]](#) son funciones que tienen acceso a los objetos *request* y *response* que se crean a partir de una petición HTTP. Estas funciones se encargan de recibir las peticiones y parsear¹⁶ el formato JSON que recibimos. Además, se suelen utilizar para comprobar si el usuario que ha hecho la petición tiene la autorización necesaria, o para manejar posibles errores.

Una de las características más importantes que tienen estas funciones es la función next. Esta función permite llamar automáticamente a la siguiente función que contenga un parámetro next. Si el middleware ejecuta el next con un error, buscará la siguiente función con los parámetros next y error, y esta se encargará de procesar el error y devolver una respuesta para petición. Si el middleware no arroja ningún error, la siguiente función con el parámetro next será la función del controlador, que se encargará ahora sí de gestionar la petición.

¹⁴ **Scope**: Contexto actual de ejecución que especifica qué objetos pueden ser referenciados.

¹⁵ **Keyword**: Palabras reservadas por el lenguaje de programación que forman parte de su sintaxis.

¹⁶ **Parsear**: Analizar una secuencia de símbolos para transformar una entrada de texto a una estructura de datos.

8.4.3. Hook pattern

En React, los hooks son funciones que almacenan lógica que puede ser reutilizada en múltiples componentes. React ofrece algunos hooks nativos, como por ejemplo `useState` o `useEffect`, pero también permite al desarrollador crear sus propios *custom hooks*. El hook pattern design [\[34\]](#) consiste en extraer lógica repetida en un único módulo que permite manejar el estado de un componente y controlar su ciclo de vida. De esta manera, mejoramos el mantenimiento de nuestro código y su legibilidad. En este proyecto los veremos principalmente para poder encapsular las peticiones a nuestra API REST.

8.4.4. Provider pattern

En react los componentes son piezas de código independientes. Esto implica que para pasar datos de un componente a otro se ha de utilizar una propiedad nativa de React llamada *props*, que no son más que parámetros pasados entre componentes. Ahora bien, hay ocasiones en las que no tiene mucho sentido pasar ciertas variables a través de los props, ya que no es información específica de un único componente sino de un grupo de componentes. Este caso se podría dar por ejemplo a la hora de pasar información global de la aplicación. El componente no puede acceder a esta información a menos que se le pase como prop ya que no está dentro de su scope, y utilizar el patrón Singleton no es una buena opción ya que crear variables globales se considera una mala práctica en Javascript. Por lo tanto, surge como alternativa el provider pattern [\[34\]](#), un patrón de diseño utilizado para compartir datos globales entre múltiples componentes del árbol de componentes de React.

Los contextos nos permiten compartir valores y métodos a un grupo de componentes sin necesidad de utilizar los props. Al crear un contexto se crea un objeto que contiene un *Provider*, que recibirá como prop toda la información a compartir. Finalmente, los componentes podrán acceder a esta información a través del `useContext` hook, que recibirá como parámetro el contexto en cuestión para poder modificarlo. Es importante remarcar que solo tendrán acceso al contexto los componentes que estén por debajo de él en el árbol de componentes.

8.4.5. Compound components pattern

A lo largo de la implementación, nos encontraremos en varias ocasiones componentes que dependen de otros componentes padre. El compound component pattern [\[34\]](#) es un patrón que crea un contexto para manejar la lógica de un conjunto de componentes de manera conjunta. De esta manera se evita el prop drilling, qué sucede cuando tienes que pasar un prop por diversos componentes (aunque estos no lo vayan a utilizar), ya que es la única manera de que le llegue al componente hijo. Con el compound component pattern se consigue que la lógica de un componente hijo no afecte al componente padre y este quede libre de dependencias. Esto vuelve mucho más flexible los componentes, ya que si queremos añadir un nuevo tipo de componente hijo, no será necesario modificar el componente padre.

8.5. Diseño del front-end

A continuación, describiremos cómo estará formada la arquitectura del front-end, empezando con ver cómo sería el mapa navegacional de nuestra aplicación web, y finalizando con cómo será la interfaz de usuario y la justificación de la toma de decisiones de este apartado.

8.5.1. Mapa navegacional

En la imagen 12 podemos observar el mapa navegacional del proyecto, que muestra todas las vistas que tendrá disponible nuestro sistema. Cada cuadro representa una vista, mientras que las flechas representan las acciones que se pueden realizar utilizando los botones.

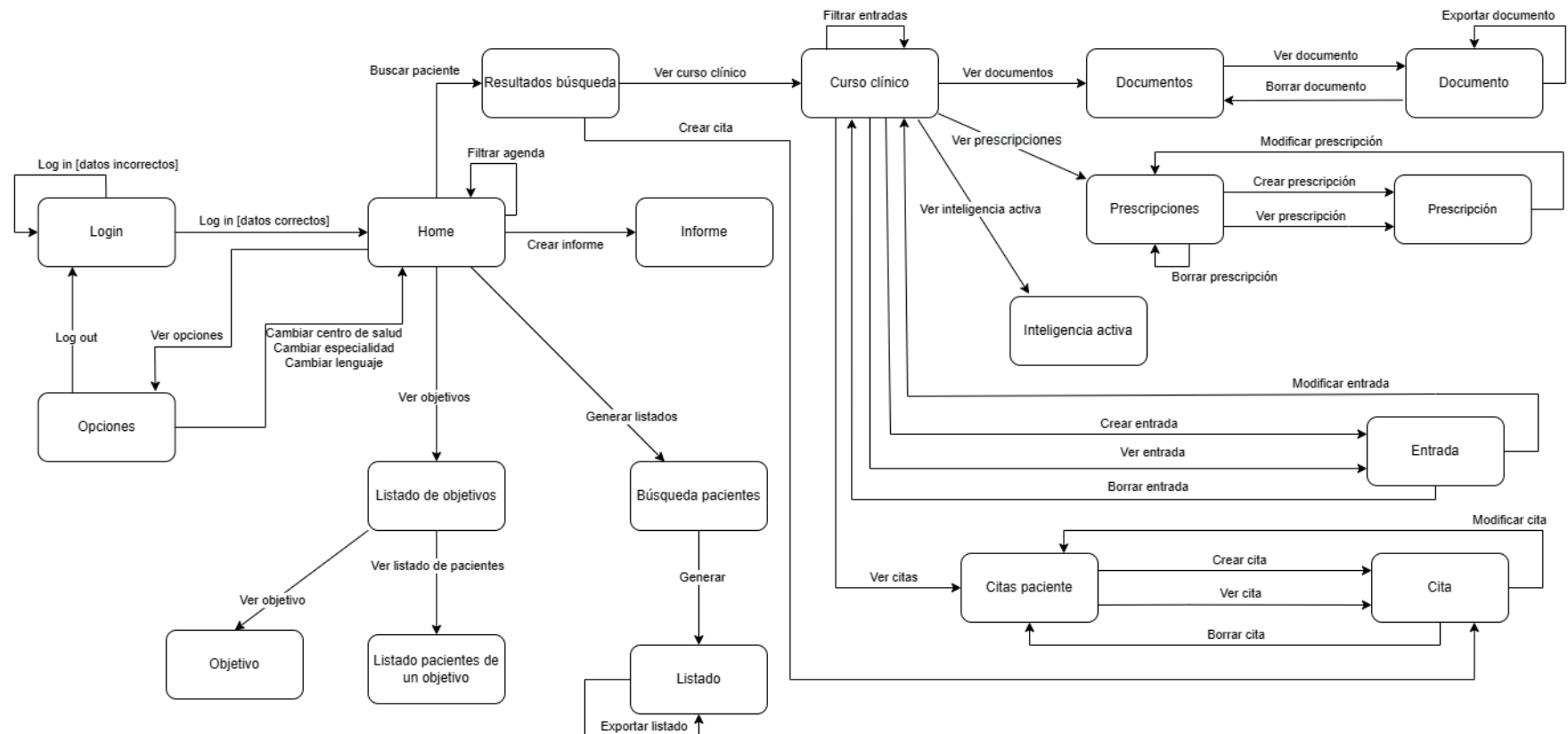


Imagen 10. Mapa navegacional

8.5.2. Diseño de la capa de presentación

En este apartado mostraremos los *mockups*¹⁷ del diseño de la aplicación final. En todo momento se ha intentado mantener la distribución de los componentes del programa original del ECAP, para que el personal sanitario le sea más fácil acostumbrarse a la aplicación web, pero se ha cambiado el estilo de todo el programa. Para cada pantalla explicaremos su función, y lo que cambia respecto al ECAP. También comentar que para simplificar sólo mostraremos el diseño de la aplicación web, pero también estará disponible la versión móvil que será muy similar pero adaptado al tamaño del dispositivo.

Inicio de sesión

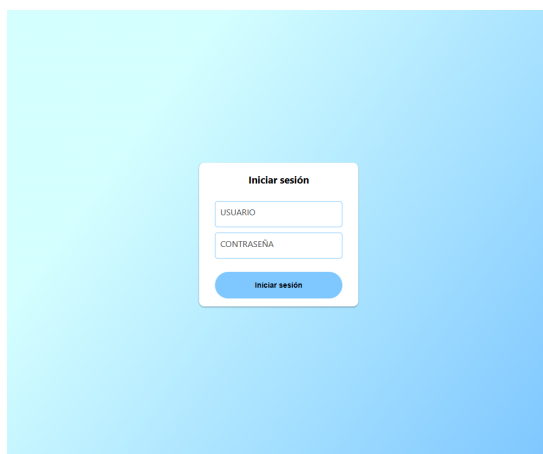


Imagen 11. Diseño pantalla de iniciar sesión

La primera pestaña que verá el usuario nada más entrar será la pantalla de la imagen 13. Si el usuario intenta entrar en alguna url sin estar autenticado, automáticamente se le redirigirá a esta pantalla. En caso de que al iniciar sesión algún parámetro esté incorrecto, aparecerá un mensaje estándar en rojo para indicar el error al usuario. Como el usuario no ha de registrarse ni puede cambiar contraseña desde la aplicación, será una pestaña muy sencilla.

Pantalla principal

Hora	Nombre complet	Sexe	Edat	T.Vis.
14:30	FÁTIMA MENÉNDEZ BECERRA	F	47	Presencial 15min

CIP: FAME1111111111 DNI: 86457356H

Direccion: C/ Eras 33, 28743 - Canencia Telefono: 736524563

Imagen 12. Diseño pantalla principal

Al iniciar sesión se redirigirá al usuario a la pestaña principal, que es la que se enseña en la imagen 14. A la izquierda nos encontraremos con la selección de la agenda actual y el día,

¹⁷ Mockup: Prototipo que se utiliza para exhibir o probar un diseño.

y con el buscador del paciente. A la parte derecha nos encontraremos con todas las visitas de la agenda. Y en la parte inferior el panel con la información del paciente y la posibilidad de ver su curso clínico o crear una nueva consulta. Lo único que cambia respecto al ECAP es la posición de la información del paciente, que se ha movido de la izquierda a la parte inferior para ganar espacio y para hacer más intuitivo la manera de acceder al curso clínico o de crear una nueva consulta.

Configuración

Configuració

Informació del compte

Nom complet: Marco Carreño Millán
DNI: 836475828
Número de col·legiat: 364320
Usuari: 1Q2W3E4R
Correu: MarcoCarrenoMillan@gmail.com
Número de telèfon: 765897698

Ajustaments del perfil

Canvia les dades relacionades al teu compte.

Canvia el centre: CAP Antón de Borja

Canvia l'especialitat: Medicina general

Canvia el llenguatge: Catalán

Aplicar canvis

Tancar sessió

Imagen 13. Diseño pantalla de configuración

Si le damos al engranaje de la barra de navegación, se nos abrirá la pestaña de configuración, que es la que podemos ver en la imagen 15. Esta nos muestra la información del trabajador que ha abierto la sesión y nos permite cambiar el centro, la especialidad y el lenguaje. Esta vista es bastante similar al ECAP exceptuando el apartado de cambiar lenguaje, ya que el programa del ECAP solo está disponible en catalán. Además ellos tenían listas, lo cual se ha cambiado por desplegables para hacerlo más cómodo e intuitivo.

Curso clínico

FÁTIMA MENÉNDEZ BECERRA
Edat: 47 anys - Sexe: F - Gènere: F

Intel·ligència Activa

IMC: 17.14 kg/m²

Hàbits tòxics: Sí

Al·lèrgies: Sí

Prescripcions | Documents | Visites

Prescripcions actuals

- PARACETAMOL
- ATORVASTATINA
- FORMOTEROL
- BUSPIRONA

Notas previas | Añadir/modificar nota

08/01/2023 10:28 CARREÑO MILLAN, MARCO - ENFERMERIA

VIRIASI

M Proceso gripal de 2 días de evolución

A No AMI

C Odinofagia, rinorrea, tos productiva y fiebre

Orofaringe hiperemica con hipertrofia amigdal, no exudados. AR MVC sin ruidos sobreañadidos. No

E palpación de adenopatías laterocervicales. Niega dolor a palpación de senos paranasales. TA 115/78 FC 109 Sat O2 100% T 38,5°

PC TAR Negativo

D Viriasí vs Gripe

P PARACETAMOL MABO 500MG COMPRIMIDOS 1 x 8h. cada 5d.

07/01/2023 13:25 CARREÑO MILLAN, MARCO - MEDICINA GENERAL

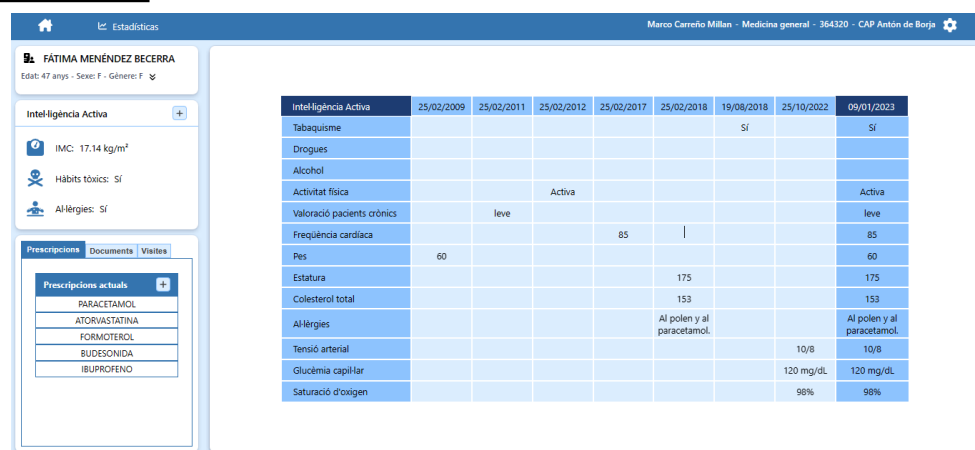
Actu | **Inactiu**

- VIRIASI
- GRIPE A
- FIBRILACIÓN AURICULAR

Imagen 14. Diseño pantalla del curso clínico

Desde la pantalla de la imagen 16 se podrá observar el curso clínico de un paciente. A la izquierda podremos ver la información del paciente y si hacemos clic en la flecha podremos obtener información más detallada del paciente. Abajo de la información, podremos ver un resumen de la inteligencia activa, prescripciones, documentos y visitas de un paciente. En la parte central nos encontraremos con sus entradas y en la parte derecha sus diagnósticos tanto activos como inactivos. Si le hacemos clic a un diagnóstico se filtrarán las entradas según ese diagnóstico. La parte central y derecha son muy similares al ECAP, sin embargo la parte izquierda se ha añadido completamente nueva. Este nuevo panel nos permitirá ver información importante con un solo vistazo, sin necesidad de ir a esa sección. Se ha buscado incorporar la información más relevante ocupando el menor espacio posible y los *stakeholders* han participado en la elección de la información y lo han valorado positivamente.

Inteligencia activa



Estadísticas Marco Carreño Millán - Medicina general - 364320 - CAP Antón de Borja

FÁTIMA MENÉNDEZ BECERRA
Edat: 47 anys - Sexe: F - Gènere: F

Inteligència Activa +

IMC: 17.14 kg/m²

Hàbits tòxics: Sí

Al·lèrgies: Sí

Prescripcions Documents Visites

Prescripcions actuals +

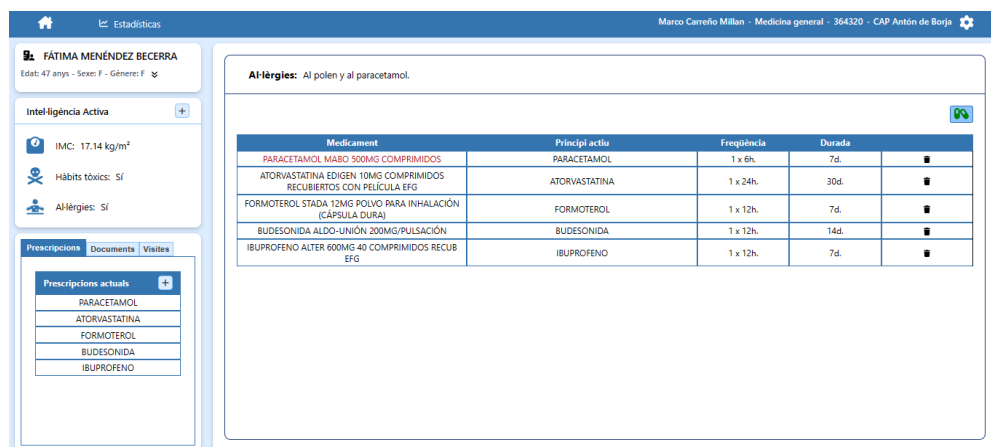
- PARACETAMOL
- ATORVASTATINA
- FORMOTEROL
- BUDESONIDA
- IBUPROFENO

Inteligència Activa	25/02/2009	25/02/2011	25/02/2012	25/02/2017	25/02/2018	19/08/2018	25/10/2022	09/01/2023
Tabaquisme						Sí		Sí
Drogues								
Alcohol								
Activitat física			Activa					Activa
Valoració pacients crònics		leve						leve
Freqüència cardíaca				85				85
Pes	60							60
Estatura					175			175
Colesterol total					153			153
Al·lèrgies				Al·lèrgies al polen i al paracetamol.				Al·lèrgies al polen i al paracetamol.
Tensió arterial						10/8		10/8
Glucèmia capil·lar						120 mg/dL		120 mg/dL
Saturació d'oxigen						98%		98%

Imagen 15. Diseño pantalla de la inteligencia activa

Si hacemos clic en el botón de “más” de la inteligencia activa, se nos abrirá el panel mostrado en la imagen 17. A la izquierda podremos ver el nombre de la propiedad de la inteligencia activa, en la parte central la fecha en la que se produjo una modificación y su valor, y en la derecha el último valor de esa propiedad acompañado de la fecha actual.

Prescripciones, documentos y visitas



Estadísticas Marco Carreño Millán - Medicina general - 364320 - CAP Antón de Borja

FÁTIMA MENÉNDEZ BECERRA
Edat: 47 anys - Sexe: F - Gènere: F

Inteligència Activa +

IMC: 17.14 kg/m²

Hàbits tòxics: Sí

Al·lèrgies: Sí

Prescripcions Documents Visites

Prescripcions actuals +

- PARACETAMOL
- ATORVASTATINA
- FORMOTEROL
- BUDESONIDA
- IBUPROFENO

Al·lèrgies: Al·lèrgies al polen i al paracetamol.

Medicament	Principi actiu	Freqüència	Durada	
PARACETAMOL MABO 500MG COMPRIMIDOS	PARACETAMOL	1 x 6h.	7d.	■
ATORVASTATINA EDIGEN 10MG COMPRIMIDOS RECUBIERTOS CON PELICULA EFG	ATORVASTATINA	1 x 24h.	30d.	■
FORMOTEROL STADA 12MG POLVO PARA INHALACIÓN (CAPSULA DURA)	FORMOTEROL	1 x 12h.	7d.	■
BUDESONIDA ALDO-UNION 200MG/PULSACIÓN	BUDESONIDA	1 x 12h.	14d.	■
IBUPROFENO ALTER 600MG 40 COMPRIMIDOS RECUB EFG	IBUPROFENO	1 x 12h.	7d.	■

Imagen 16. Diseño pantalla de listado de prescripciones

Imagen 17. Diseño pantalla de listado de documentos

Imagen 18. Diseño pantalla de listado de visitas

Como se puede ver, las tres pantallas mostradas en la imagen 18, 19 y 20 son muy similares. Se especificará la información de cada prescripción, documento, o visita a través de una tabla. En el caso del listado de prescripciones, aparecerán en rojo los medicamentos que interfieren con alguna alergia. El botón de las cápsulas te permitirá añadir una nueva prescripción y el de nueva consulta te permitirá crear una nueva visita.

Crear prescripción o visita

Imagen 19. Diseño pantalla de crear prescripción

Para crear una nueva prescripción nos aparecerá un formulario mostrado en la imagen 21. Si seleccionamos un medicamento a través de un buscador, nos importará por defecto la posología y duración estándar que aparece en el prospecto del medicamento. Además, si se viene desde crear una entrada, nos aparecerá una serie de recomendaciones según lo que solemos recetar para el diagnóstico que hemos seleccionado. Esto nos agiliza la búsqueda del medicamento. También podremos importar las instrucciones del prospecto tanto para el paciente como para la farmacia.

Imagen 20. Diseño pantalla de crear visita

En la imagen 22 podemos ver el formulario para crear una nueva visita. Podremos seleccionar la hora, fecha, agenda a la cual le queremos crear la cita, paciente, motivo y tipo de la visita. En este caso, la agenda predeterminada siempre será la del trabajador que está creando la cita.

Crear entrada o informe

Imagen 21. Diseño pantalla de crear entrada

En la imagen 23 podemos ver el formulario para crear una entrada (como el formulario es extenso se muestra dividido en dos imágenes). La pantalla para crear un informe será prácticamente idéntica pero con algunas adaptaciones. Para ahorrar tiempo al personal sanitario se han hecho las siguientes modificaciones. Primero de todo los antecedentes se podrán importar con el botón colocado a su derecha y se podrán actualizar por unos nuevos. La segunda mejora es que aparecerán unas recomendaciones con los diagnósticos más populares que encajen con los síntomas descritos en el apartado de clínica. Además, también se podrá seleccionar el diagnóstico a través del panel de la derecha, que nos indica los diagnósticos activos e inactivos del paciente. Finalmente, si recetamos algún medicamento desde crear entrada, se escribirá automáticamente el nombre de medicación conjunto, la posología y la duración para evitar así tener que escribirlo manualmente.

Objetivos

Estadísticas				
Marco Carreño Millán - Medicina general - 364320 - CAP Antón de Borja				
Indicadores	Resultat (%)	Gràfica	Pacients Pendent	Puntu
Malaltia cardiovascular				
EQA0201 - Tractament adequat de la fibril·lació auricular	100		0	46 de 46
EQA0237 - Tractament amb IECA o ARAII en la HTA o DM2 amb nefropatia	50		1	23 de 46

Imagen 22. Diseño pantalla de visualizar objetivos

La pantalla de la imagen 24 nos muestra el listado de objetivos de un usuario. Los objetivos estarán divididos por categorías, e irán acompañados con el porcentaje del progreso, el cual cambiará de color dependiendo del porcentaje. También veremos una gráfica mensual, el número de pacientes pendientes asignados al objetivo y su progreso en puntos.

EQA0201 - Tratamiento adecuado de la fibrilación auricular

Resolts	No resolts	Pacients
3	0	3

Llista de pacients

CIP	Nom complet	Telefon
No s'han trobat dades.		

Pacients resolts

CIP	Nom complet	Telefon
FAME1111111111	FÁTIMA MENÉNDEZ BECERRA	736524563
GINO1111111111	ALEXANDRA GILAVERT NOVOA	657836735
CARU1111111111	DARIO CARMONA RUEDA	632571390

Imagen 23. Diseño pantalla de visualizar pacientes asignados a un objetivo

Si se le hace clic a un objetivo, se mostrará la pantalla de la imagen 25. Esta mostrará una tabla resumiendo el progreso del objetivo y dos listados de pacientes. Uno con los pacientes pendientes y otro más con los pacientes ya resueltos, ambos acompañados con su información. Si se le hace clic a un paciente, se podrá visualizar su curso clínico.

Generación de listados

Marco Carreño Millán - Medicina general - 364320 - CAP Antón de Borja

PARÀMETRES DE CERCA

Data d'inici:
10/12/2022
Data final:
09/01/2023

Seleccionar pacients de:
Tots els pacients

Diagnòstic
Fibrilación Auricular

Estat:
Actiu
Inactiu
Tots

Medicament

Edat:
Min: 23 - Max: 117
Sexe:
Masculí
Femení
Tots

Cerca

Imagen 24. Diseño pantalla de generar listado

LLISTA DE PACIENTS

Resultats finals

Pacients	Homes	Dones
3	1	2

Llista de pacients

Tabaquisme

Drogues

Alcohol

Activitat Física

Valoració Pacients Crònics

Freqüència Cardíaca

Pes

Estatura

Colesterol Total

Colesterol Total

Tensió arterial

Glucèmia capil·lar

Saturació d'oxigen

CIP	Nom complet	Edat	Sexe	Valoració Pacients Crònics	Colesterol Total
GINO1111111111	ALEXANDRA GILAVERT NOVOA	28	F	leve	153
CARU1111111111	DARIO CARMONA RUEDA	75	M	leve	153
FAME1111111111	FÁTIMA MENÉNDEZ BECERRA	47	F	leve	153

Imagen 25. Diseño pantalla de listado generado

Para generar un listado se mostrará la pantalla de la imagen 26, donde se podrá insertar los parámetros de búsqueda del listado que queremos generar. Podremos buscar pacientes que tengan un cierto diagnóstico y/o medicamento tanto vigente como caducado y se podrá filtrar según fecha, sexo y centro. Al darle a buscar, aparecerá un panel en la parte inferior como el mostrado en la imagen 27. Este panel contendrá una tabla resumen de todos los pacientes y el listado de pacientes resultante. Además, se nos permitirá seleccionar aspectos de la inteligencia activa para añadir al listado. Finalmente, podremos exportar el listado de pacientes tanto en formato pdf como en excel.

8.6. Diseño del back-end

Una vez definido cómo será la arquitectura del front-end, veremos con más detalle la arquitectura del servidor. Empezaremos describiendo todas las llamadas api que formarán nuestra API REST. Seguidamente, veremos el diagrama de clases definitivo y la explicación de un diagrama de secuencia de una de las funcionalidades del sistema. Por último veremos el diseño del modelo de la base de datos.

8.6.1. API REST

En este apartado se definirán todas las llamadas de la API que formarán parte de nuestra API REST, junto a una breve descripción de lo que hace y los errores que vamos a manejar. Las operaciones CRUD nos permiten realizar todas las operaciones que necesitamos para manipular nuestra base de datos y están compuestas por:

- Create: donde se utilizará la operación **POST**.
- Read: donde se utilizará la operación **GET**.
- Update: donde se utilizarán las operaciones **PUT** y **PATCH**.
- Delete: donde se utilizará la operación **DELETE**.

En caso de que durante la llamada a la API se produzca un error, se devolverá en todos los casos el respectivo código de estado HTTP y un mensaje predefinido explicando el error. Los códigos que manejaremos a lo largo de este proyecto son:

- 200: La solicitud ha tenido éxito.
- 201: La solicitud ha tenido éxito y se ha creado un nuevo recurso.
- 400: No se ha podido interpretar la solicitud debido a una sintaxis inválida.
- 401: Es necesario autenticar para obtener una respuesta.
- 403: El cliente no posee los permisos necesarios para obtener una respuesta.
- 404: El servidor no ha podido encontrar el contenido solicitado.
- 500: El servidor se ha encontrado con una situación que no sabe cómo manejarla.

A continuación se enseñarán todas las llamadas divididas en sus respectivos módulos.

Trabajadores

Llamada API	/login	Operación	POST
Descripción	Se intenta iniciar sesión con los parámetros del body.		
Códigos	200, 400, 500		

Llamada API	/home	Operación	GET
Descripción	Se verifica si el trabajador está autenticado.		
Códigos	200, 401, 403, 404, 500		

Llamada API	/trabajadores/idTrabajador/updateLanguage	Operación	PATCH
Descripción	Se intenta modificar los valores del trabajador con el id igual a idTrabajador.		
Códigos	200, 400, 401, 403, 404, 500		

Llamada API	/trabajadores/idTrabajador/getRecs/idDiagnosis	Operación	GET
Descripción	Se intenta conseguir la lista de medicamentos que suele recetar un trabajador para el diagnóstico especificado en el body de la petición.		
Códigos	200, 401, 403, 404, 500		

Pacientes

Llamada API	/patients/idPatient	Operación	GET
Descripción	Se intenta conseguir el paciente con el id igual a idPaciente.		
Códigos	200, 401, 403, 404, 500		

Llamada API	/patients/idPatient/activeIntelligence	Operación	GET
Descripción	Se intenta conseguir la inteligencia activa del paciente con el id igual a idPaciente.		
Códigos	200, 401, 403, 404, 500		

Llamada API	/patients/idPatient/updatePatient	Operación	PUT
Descripción	Se intenta modificar los valores de un paciente.		
Códigos	200, 400, 500		

Llamada API	/patients/?	Operación	GET
Descripción	Se intenta conseguir la lista de pacientes que encajan con los parámetros especificados en la url.		
Códigos	200, 400, 401, 403, 500		

Llamada API	/patients/idPatient/deleteDoc/idDoc	Operación	DELETE
Descripción	Se intenta eliminar el documento identificado con idDoc el paciente identificado con idPatient.		
Códigos	200, 401, 403, 500		

Llamada API	/patients/report/upload	Operación	POST
Descripción	Se intenta conseguir crear un informe nuevo y se añade el documento a la lista de documentos del paciente.		
Códigos	200, 401, 403, 404, 500		

Llamada API	/patients/report/download	Operación	GET
Descripción	Se intenta conseguir el documento según los parámetros especificados en el body de la llamada.		
Códigos	200, 400, 401, 403, 500		

Entradas

Llamada API	/entries/getDiagnosisRec	Operación	GET
Descripción	Se intenta conseguir la lista de diagnósticos recomendados para una clínica.		
Códigos	200, 401, 403, 500		

Llamada API	/entries/createEntry	Operación	POST
Descripción	Se intenta conseguir crear una nueva entrada, se añade la entrada a la lista de entradas del paciente y se actualizan los objetivos del trabajador.		
Códigos	200, 401, 403, 500		

Llamada API	/entries/createNote	Operación	POST
Descripción	Se intenta conseguir crear una nueva nota a una entrada, se añade la nota a la lista de notas de la entrada del paciente y se actualizan los objetivos del trabajador.		
Códigos	200, 401, 403, 500		

Llamada API	/entries/translateEntry	Operación	POST
Descripción	Se intenta conseguir traducir la lista de notas de una entrada al lenguaje de un trabajador.		
Códigos	200, 401, 403, 500		

Llamada API	/entries/updateNote/idEntry	Operación	PATCH
Descripción	Se intenta conseguir modificar la lista de notas de la entrada identificada con idEntry con una nueva lista de notas y se actualizan los objetivos del trabajador.		
Códigos	200, 401, 403, 500		

Llamada API	/entries/deleteNote/idEntry	Operación	DELETE
Descripción	Se intenta conseguir modificar la lista de notas de la entrada identificada con idEntry con una nueva lista de notas sin la nota eliminada y se actualizan los objetivos del trabajador.		
Códigos	200, 401, 403, 500		

Llamada API	/entries/updateNote/idEntry	Operación	PATCH
Descripción	Se intenta conseguir modificar la lista de notas de la entrada identificada con idEntry con una nueva lista de notas y se actualizan los objetivos del trabajador.		
Códigos	200, 401, 403, 500		

Estadísticas

Llamada API	/goals/getGoals/idWorker	Operación	GET
Descripción	Se intenta conseguir la lista de objetivos del trabajador con el identificador igual a idWorker.		
Códigos	200, 401, 403, 500		

Llamada API	/goals/getPatientsListGoal	Operación	GET
Descripción	Se intenta conseguir la información de todos los pacientes especificados en el body de la llamada.		
Códigos	200, 401, 403, 500		

Llamada API	/goals/getPatientsLists	Operación	GET
Descripción	Se intenta conseguir un listado de pacientes según los parámetros especificados en el body de la llamada.		
Códigos	200, 401, 403, 500		

Prescripciones

Llamada API	/prescriptions/searchMed	Operación	GET
Descripción	Se intenta conseguir la lista de medicamentos que encajen con el nombre del medicamento o del principio activo pasado por el body de la petición.		
Códigos	200, 401, 403, 500		

Llamada API	/prescriptions/createPrescriptions	Operación	POST
Descripción	Se intenta crear una nueva prescripción y se añade la prescripción a la lista de prescripciones de un paciente, se actualiza la lista de elecciones de medicamentos del trabajador que la ha recetado y se actualizan los objetivos del trabajador.		
Códigos	200, 401, 403, 500		

Llamada API	/prescriptions/updatePrescription/idPrescription	Operación	PATCH
Descripción	Se intenta conseguir modificar la prescripción con id igual a idPrescription y se actualizan los objetivos del trabajador.		
Códigos	200, 401, 403, 500		

Llamada API	/prescriptions/deletePrescription/idPrescription	Operación	DELETE
Descripción	Se intenta conseguir eliminar la prescripción con id igual a idPrescription y se actualizan los objetivos del trabajador.		
Códigos	200, 401, 403, 500		

Agenda

Llamada API	/schedules/getSchedule/:idSchedules	Operación	GET
Descripción	Se intenta conseguir la lista de visitas de la agenda con el identificador igual a idSchedules.		
Códigos	200, 401, 403, 500		

Llamada API	/schedules/getAppointments/:idPatient	Operación	GET
Descripción	Se intenta conseguir la lista de visitas del paciente con el identificador igual a idSchedules.		
Códigos	200, 401, 403, 500		

Llamada API	/schedules/getSchedules	Operación	GET
Descripción	Se intenta conseguir la lista de agendas del centro especificados en el body de la llamada.		
Códigos	200, 401, 403, 500		

Llamada API	/schedules/createAppointment	Operación	POST
Descripción	Se intenta crear una nueva cita previa y se añade la cita a la lista de citas previas del paciente y de la agenda especificados en el body de la llamada.		
Códigos	200, 401, 403, 500		

Llamada API	/schedules/updateAppointment/:idAppointment	Operación	PATCH
Descripción	Se intenta conseguir modificar la cita previa con id igual a idAppointment.		
Códigos	200, 401, 403, 500		

Llamada API	/schedules/deleteAppointment/:idAppointment	Operación	DELETE
Descripción	Se intenta conseguir eliminar la cita previa con id igual a idAppointment.		
Códigos	200, 401, 403, 500		

Llamada API	/schedules/deleteUrgAppointment/:idAppointment	Operación	DELETE
Descripción	Se intenta conseguir eliminar la visita urgente con id igual a idAppointment.		
Códigos	200, 401, 403, 500		

Llamadas API extras

Para la implementación del código del proyecto se necesitarán llamadas a la API para facilitar tareas como la inserción de nuevos datos al sistema. A continuación se definirán las llamadas API que se han utilizado únicamente para desarrollar el proyecto, pero que no son utilizadas por parte del front-end.

Trabajadores

Llamada API	/auth/register	Operación	POST
Descripción	Se crea un nuevo trabajador con todos los atributos necesarios.		
Códigos	201, 500		

Llamada API	/trabajadores	Operación	GET
Descripción	Se intenta conseguir una lista de todos los trabajadores que hay en el sistema.		
Códigos	200, 500		

Llamada API	/trabajadores/idTrabajador	Operación	GET
Descripción	Se intenta conseguir el trabajador con el id igual a idTrabajador.		
Códigos	200, 401, 403, 404, 500		

Llamada API	/trabajadores/idWorker/updateWorker	Operación	PATCH
Descripción	Se intenta modificar el trabajador con identificador igual a idWorker.		
Códigos	200, 500		

Pacientes

Llamada API	/patients/createPatient	Operación	POST
Descripción	Se crea un nuevo paciente con todos los atributos necesarios.		
Códigos	200, 500		

Entradas

Llamada API	/entries/getEntry/idEntry		GET
Descripción	Se intenta conseguir la entrada con el id igual a patientId.		
Códigos	200, 401, 403, 404, 500		

Llamada API	/entries/updateDiagnosis/idDiagnosis	Operación	PATCH
Descripción	Se intenta modificar los valores de un diagnóstico.		
Códigos	200, 400, 500		

Llamada API	/entries/createDiagnosis	Operación	POST
Descripción	Se intenta crear un nuevo diagnóstico con todos los atributos necesarios.		
Códigos	200, 400, 500		

Objetivos

Llamada API	/goals/createGoal	Operación	POST
Descripción	Se intenta crear un nuevo objetivo con todos los atributos necesarios.		
Códigos	200, 400, 500		

Llamada API	/goals/updateGoal/idGoal	Operación	PATCH
Descripción	Se intenta modificar los valores de un objetivo.		
Códigos	200, 400, 500		

Prescripciones

Llamada API	/prescriptions/createMed	Operación	POST
Descripción	Se intenta crear un nuevo medicamento con todos los atributos necesarios.		
Códigos	200, 400, 500		

Agenda

Llamada API	/schedules/createUrgAppointment	Operación	POST
Descripción	Se intenta crear una nueva visita urgente con todos los atributos necesarios.		
Códigos	200, 400, 500		

Llamada API	/schedules/createSchedule	Operación	POST
Descripción	Se intenta crear una nueva agenda con todos los atributos necesarios.		
Códigos	200, 400, 500		

8.6.2. Diagrama de clases

En este apartado vamos a mostrar el diagrama de clases que forma nuestro sistema siguiendo la estructura propuesta en el apartado 8.3. Para simplificar el diagrama, se ha reducido los componentes de React según el componente principal. Cada componente de React será el encargado de manejar todos los eventos producidos por el usuario y de crear la vista correspondiente y estarán formados por los componentes vista y controlador del patrón de diseño MVC. Por otra parte, el componente modelo estará formado por los middlewares, los controladores y los modelos de mongoose de la capa de aplicación.

El flujo de nuestra aplicación web funciona de la siguiente manera. Los componentes de React se comunican con el servidor a través de los middlewares de la API REST, que a su vez utilizan el middleware de “handleError” en caso de que surja algún error y el de “verifyUser” para verificar que el usuario que ha realizado la petición está autorizado a acceder a los datos. Si el usuario está verificado se pasa la petición al controlador correspondiente.

Los controladores de la capa de aplicación son los encargados de gestionar toda la lógica de negocio de las peticiones recibidas y realizan en caso necesario las peticiones hacia nuestros servicios externos. En primer lugar tenemos el servicio de la API de MyMemory, la cual será accedida únicamente por el controlador “entries”, y en segundo lugar tenemos el servicio de la base de datos MongoDB, la cual utiliza como intermediario los modelos de mongoose para realizar las peticiones.

Seguidamente, en la imagen 13 mostramos el diagrama de clases definitivo. Como se puede observar, tenemos como middlewares “handleError”, “verifyUser” y todos los posibles endpoints de nuestra API REST. Luego hemos creado un controlador para cada endpoint, unificando el de home con auth al utilizar las mismas funciones y finalmente hemos creado un modelo por cada documento de la base de datos (hablaremos con más profundidad de los documentos en el apartado 9.4.4).

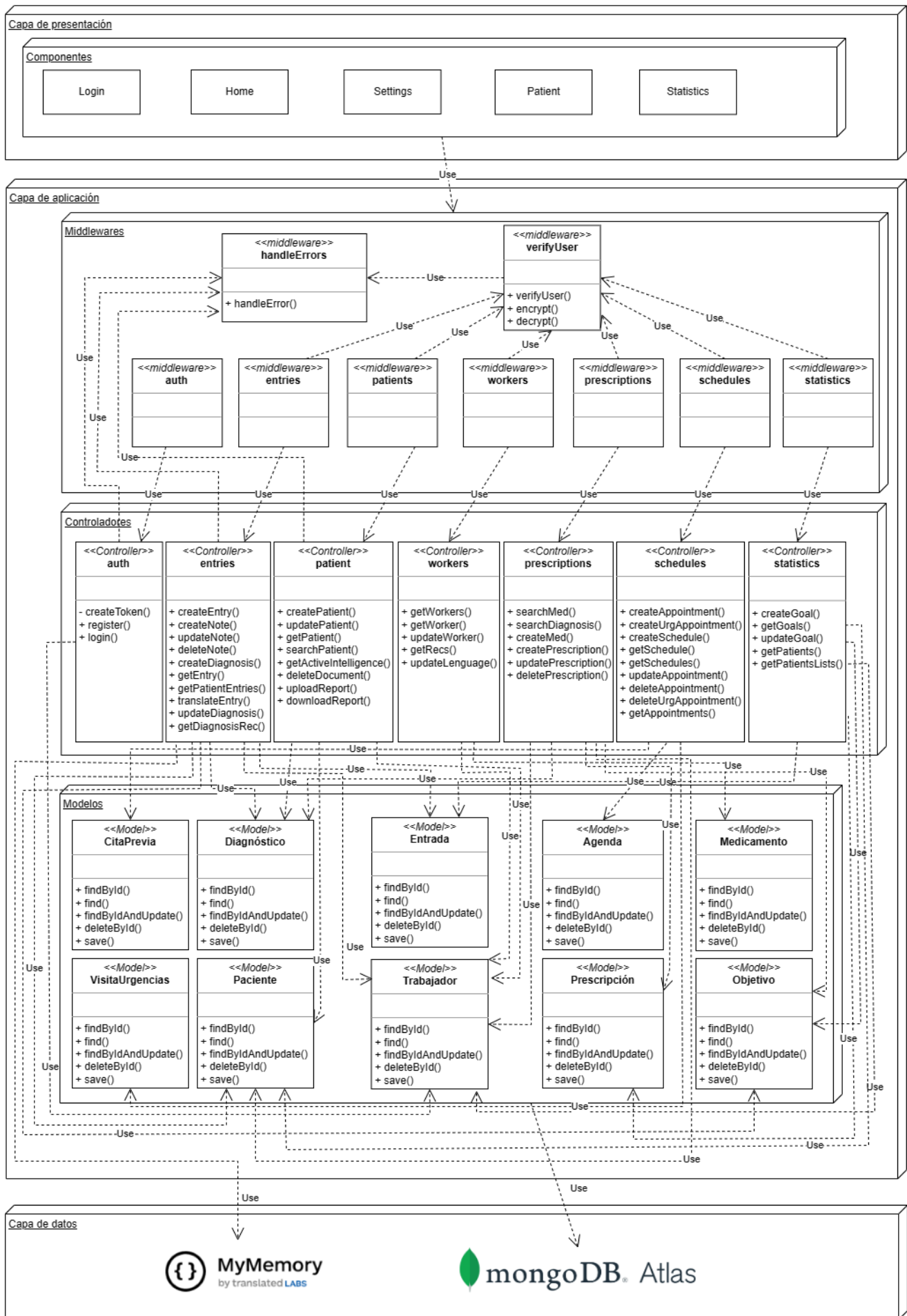


Imagen 26. Diagrama de clases

8.6.3. Diagrama de secuencia

A continuación, vamos a ver cómo funciona el flujo de ejecución de la aplicación web a través de un diagrama de secuencia. Para ello, vamos a describir los pasos que se producen si un usuario desea crear una nueva entrada. La imagen 14 muestra el diagrama de secuencia completo.

1. Desde el componente de React se recibe el input del usuario y se envía una petición a la API REST con todos los datos necesarios para crear una nueva entrada. Además, se transmite en el header de la petición el token que verifica la autenticidad del usuario.
2. La API REST recibe la petición, la envía al “entriesMiddleware” y este confirma la identidad del usuario a través del “verifyUserMiddleware”, el cual analiza el token y envía error si no existe, si no es válido o si no existe el usuario en la base de datos. Esta última comprobación se hace a través del “TrabajadorModel”, encargado de comunicarse con el documento “workers” de MongoDB.
3. Si “entriesMiddleware” recibe el error lo devuelve a la API REST, si no continúa con el proceso esta vez a través del “EntriesController”.
4. El “entriesController” indica al “EntradaModel” que cree una entrada nueva con los datos recibidos del body de la petición y este se encarga de crearla y guardarla en el documento de la BD. Si hay un fallo, salta una excepción con el código 500. Si el “entriesController” recibe el error, lo envía directamente al “entriesMiddleware”, el cual lo transmite posteriormente a la API REST, si no se seguirá con el proceso.
5. Desde el “entriesController” se consigue la información de los centros del trabajador, de nuevo desde el “trabajadorModel” y se procede a verificar si el diagnóstico de la nota de la nueva entrada coincide con los diagnósticos asignados a algún objetivo del trabajador. En caso positivo, se modifica el documento de la BD a través de “ObjetivosModel”. También se añade la entrada a la lista de entradas de un paciente a través del “PacienteModel”.
6. Si sucede algún error a la hora de modificar los valores en la BD salta la excepción y se envía el código de error al “entriesMiddleware”, sino se envía la respuesta con el código 201 con la nueva entrada creada.
7. Por último, “entriesMiddleware” recibe la respuesta o el código de error. Si este último existe, lo envía a “handleErrorMiddleware” que se encarga de crear la respuesta de la petición con el código de error correspondiente. Finalmente, se envía la respuesta al componente de React para que muestre la información al usuario.

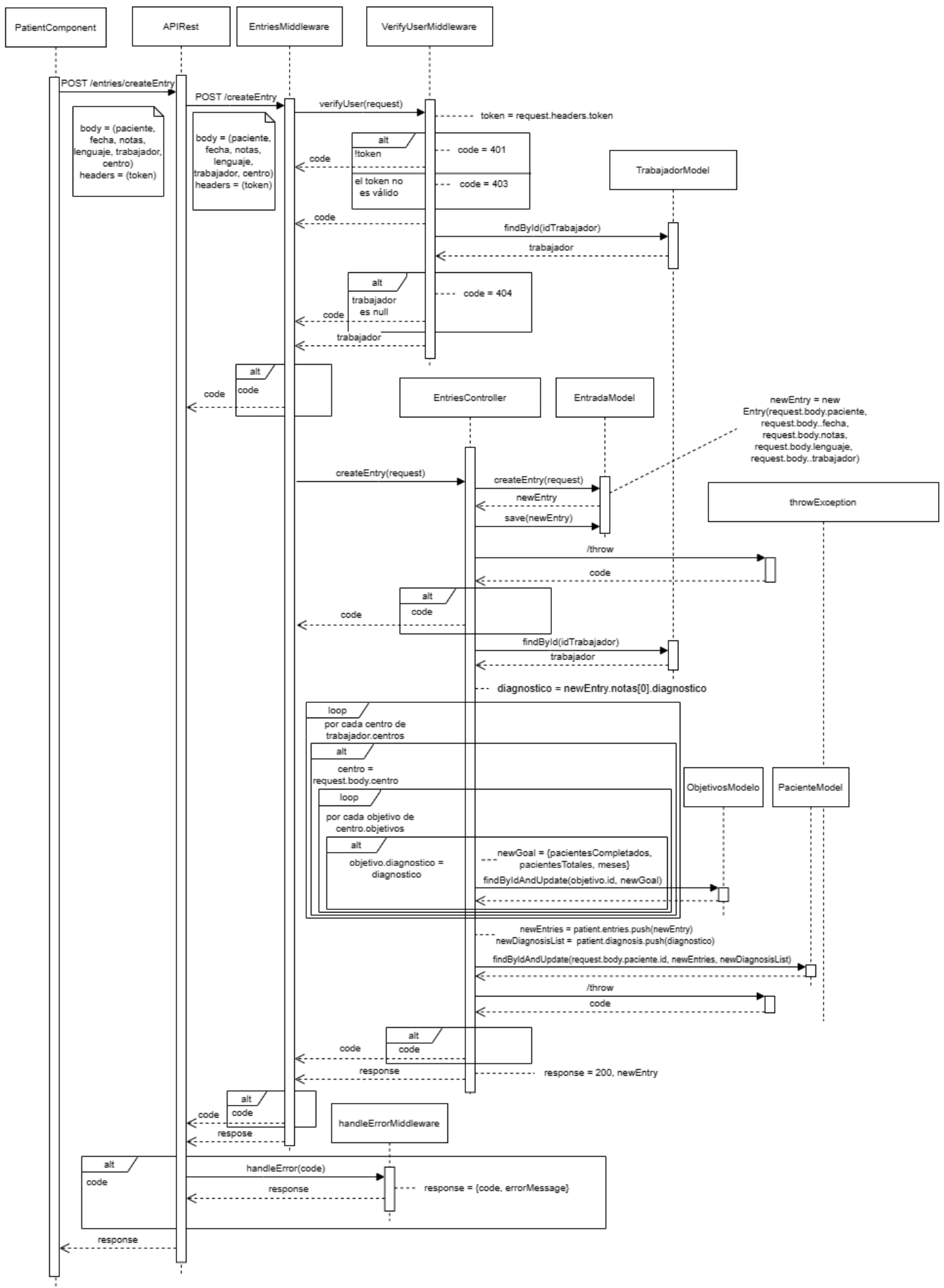


Imagen 27. Diagrama de secuencia de crear una entrada

8.6.4. Modelo de la base de datos

Para definir el modelo de la base de datos se utilizará la librería *Mongoose*. Esta librería nos permite escribir consultas a la base de datos de MongoDB y normalizar la información guardada sin sacrificar la flexibilidad que nos proporciona el noSQL.

Al ser una base de datos noSql, se ha priorizado juntar varias clases del esquema conceptual de datos del apartado 8.3.1 en un solo documento para optimizar las peticiones de lectura. Por ejemplo, las clases de “PropiedadesInteligenciaActiva” y “Documentos” se han unificado con el documento “Paciente”, ya que si utilizáramos documentos diferentes estaríamos obligados a realizar una *query*¹⁸ adicional cada vez que solicitásemos la información de un paciente. Sin embargo, las clases que participan en dos o más documentos, o que suelen ser modificadas se han mantenido como un solo documento. Esto es debido a que si sucede una modificación, solo se ha de aplicar la escritura en un solo documento y no en varios. Como MongoDB sufre mucho en las escrituras, especialmente las que afectan a más de un documento, este cambio nos es favorable.

Finalmente para representar las relaciones entre diferentes documentos se ha utilizado la propiedad “ref” de mongoose, que le indica al esquema que ese atributo es un identificador de otro documento. Y para encriptar los datos sensibles de la base de datos se ha asignado con la propiedad *get* y *set* dos funciones que se encargan de hacer el cifrado con el modelo de encriptación AES.

A continuación, se mostrarán todos los modelos que se han creado a partir de nuestro modelo conceptual de datos.

```
const DiagnosticoSchema = new mongoose.Schema({
  nombre: { type: String, required: true },
  severidad: { type: String, required: true },
  palabrasClave: [String]
})
```

```
const EntradaSchema = new mongoose.Schema({
  paciente: { type: Schema.ObjectId, ref: 'Paciente' },
  fecha: { type: Date, required: true },
  lenguaje: { type: String, required: true },
  notas: [{
    motivo: String,
    antecedentes: String,
    clinica: String,
    exploracion: String,
    pruebasComplementarias: String,
    planTerapeutico: String,
    diagnostico: { type: Schema.ObjectId, ref: 'Diagnostico', required: true }
  }],
})
```

¹⁸ Query: Petición que se realiza a las bases de datos para obtener o manipular datos almacenados.

```
descDiagnostico: String,
estado: { type: String, required: true },
prescripciones: [{ type: Schema.ObjectId, ref: 'Prescripcion' }],
trabajador: {
  id: { type: Schema.ObjectId, ref: 'Trabajador', required: true },
  role: { type: String, required: true }
}
})
```

```
const MedicamentoSchema = new mongoose.Schema({
  nombre: { type: String, required: true },
  principioActivo: { type: String, required: true },
  recetaEnfPermitida: { type: Boolean, required: true },
  insPaciente: String,
  insFarmacia: String,
  unidad: String,
  frecuencia: String,
  duracion: String
})
```

```
const PrescripcionSchema = new mongoose.Schema({
  paciente: { type: Schema.ObjectId, ref: 'Paciente' },
  fechaInicio: { type: Date, required: true },
  fechaFinal: { type: Date, required: true },
  trabajador: { type: Schema.ObjectId, ref: 'Trabajador' },
  instruccionesPaciente: String,
  instruccionesFarmacia: String,
  nombreMedicamento: { type: String, required: true },
  principioActivo: { type: String, required: true },
  frecuencia: { type: String, required: true },
  duracion: { type: String, required: true }
})
```

```
const PacienteSchema = new mongoose.Schema({
  nombre: { type: String, required: true, set: encrypt, get: decrypt },
  apellido1: { type: String, required: true, set: encrypt, get: decrypt },
  apellido2: { type: String, set: encrypt, get: decrypt },
  dni: { type: String, required: true, unique: true, set: encrypt, get:
decrypt },
  correo: { type: String, required: true, set: encrypt, get: decrypt },
  telefono: { type: String, required: true, set: encrypt, get: decrypt },
  cip: { type: String, required: true, unique: true, set: encrypt, get:
decrypt },
  fechaNacimiento: { type: Date, required: true },
  edad: { type: Number, required: true },
  sexo: { type: String, required: true },
  genero: { type: String, required: true },
  paisOrigen: { type: String, required: true },
  direccion: { type: String, required: true, set: encrypt, get: decrypt },
  antecedentes: { type: String },
  trabajadoresAsignados: [{
    rol: { type: String, required: true },
    trabajador: { type: Schema.ObjectId, ref: 'Trabajador' }
  }],
  inteligenciaActiva: [{
    name: { type: String, required: true },
    values: [{
      value: { type: String, required: true },
      date: { type: Date, required: true }
    }]
  }],
  documentos: [{
    nombre: { type: String, required: true },
    pdfUrl: { type: String, required: true },
    fechaSubida: { type: Date, required: true }
  }],
  citasPrevias: [{ type: Schema.ObjectId, ref: 'CitaPrevia' }],
  informes: [{ type: Schema.ObjectId, ref: 'Informe' }],
  diagnosticos: [{
    idDiagnostico: { type: Schema.ObjectId, ref: 'Diagnostico', required: true
},

```

```

    fecha: { type: Date, required: true },
    estadoDiagnostico: { type: String, required: true }
  }],
  prescripciones: [{ type: Schema.ObjectId, ref: 'Prescripcion' }],
  entradas: [{ type: Schema.ObjectId, ref: 'Entrada' }]
},
{
  versionKey: false,
  toObject: { getters: true, setters: true },
  toJSON: { getters: true, setters: true },
  runSettersOnQuery: true
}
)

```

```

const CitaPreviaSchema = new mongoose.Schema({
  paciente: { type: Schema.ObjectId, ref: 'Paciente', required: true },
  trabajador: { type: Schema.ObjectId, ref: 'Trabajador' },
  agenda: { type: Schema.ObjectId, ref: 'Agenda', required: true },
  tipoVisita: { type: String, required: true },
  centro: { type: String, required: true },
  fecha: { type: Date, required: true },
  especialidad: { type: String, required: true },
  motivo: { type: String }
})

```

```

const AgendaSchema = new mongoose.Schema({
  centro: { type: String, required: true },
  trabajador: { type: Schema.ObjectId, ref: 'Trabajador' },
  especialidad: { type: String },
  nombre: { type: String, required: true },
  citasPrevias: [{ type: Schema.ObjectId, ref: 'CitaPrevia' }],
  visitasUrgencia: [{ type: Schema.ObjectId, ref: 'VisitaUrgencias' }]
})

```

```

const VisitaUrgenciasSchema = new mongoose.Schema({
  paciente: { type: Schema.ObjectId, ref: 'Paciente', required: true },
  centro: { type: String, required: true },
  agenda: { type: Schema.ObjectId, ref: 'Agenda', required: true },
  fechaEntrada: { type: Date, require: true },

```

```
    fechaAsistencia: { type: Date },
    fechaSalida: { type: Date },
    triaje: { type: Number, required: true },
    comentario: String
  })
}
```

```
const TrabajadorSchema = new mongoose.Schema(
  {
    nombre: { type: String, required: true, set: encrypt, get: decrypt },
    apellido1: { type: String, required: true, set: encrypt, get: decrypt },
    apellido2: { type: String, set: encrypt, get: decrypt },
    dni: { type: String, required: true, unique: true, set: encrypt, get:
decrypt },
    correo: { type: String, required: true, set: encrypt, get: decrypt },
    telefono: { type: String, required: true, set: encrypt, get: decrypt },
    username: { type: String, required: true, unique: true },
    password: { type: String, required: true },
    esDoctor: { type: Boolean, required: true },
    numColegiado: { type: String, required: true, unique: true, set: encrypt,
get: decrypt },
    lenguaje: String,
    especialidades: { type: [String], required: true },
    pacientes: [{ type: Schema.ObjectId, ref: 'Paciente' }],
    centros: [{
      nombre: String,
      objetivos: [{ type: Schema.ObjectId, ref: 'Objetivo' }],
      pacientes: [{ type: Schema.ObjectId, ref: 'Paciente' }],
      agenda: { type: Schema.ObjectId, ref: 'Agenda' }
    }],
    turnos: [{ horaInicio: Date, horaFinal: Date, rol: String, centro: String
}],
    citasPrevias: [{ type: Schema.ObjectId, ref: 'CitaPrevia' }],
    visitasUrgencias: [{ type: Schema.ObjectId, ref: 'VisitaUrgencias' }],
    informes: [{ type: Schema.ObjectId, ref: 'Informe' }],
    entradas: [{ type: Schema.ObjectId, ref: 'Entrada' }],
    eleccionMedicamento: [{
      medicamento: { type: Schema.ObjectId, ref: 'Medicamento', required: true
},

```

```

diagnostico: { type: Schema.ObjectId, ref: 'Diagnostico', required: true
},
fechas: { type: [Date], required: true }
}]
},
{
  versionKey: false,
  toObject: { getters: true, setters: true },
  toJSON: { getters: true, setters: true },
  runSettersOnQuery: true
}
)

```

```

const ObjetivoSchema = new mongoose.Schema({
  tipo: { type: String, required: true },
  codigo: { type: String, required: true, unique: true },
  nombre: { type: String, required: true },
  descripcion: { type: String, required: true },
  definicion: { type: String, required: true },
  objetivo: { type: Number, required: true },
  pacientesTotales: [{ type: Schema.ObjectId, ref: 'Paciente' }],
  pacientesCompletados: [{ type: Schema.ObjectId, ref: 'Paciente' }],
  puntosTotales: { type: Number, required: true },
  diagnostico: { type: Schema.ObjectId, ref: 'Diagnostico', required: true
},
  medicamentos: [{ type: String, required: true }],
  edad: { type: String, required: true },
  months: [{ type: Number, required: true }]}
})

```


9. Implementación

A lo largo de este apartado, detallaremos el proceso de implementación. Primero de todo, especificaremos qué tecnologías hemos decidido utilizar para realizar el proyecto y el porqué, y acabaremos analizando la implementación de una parte del proyecto.

9.1. Tecnologías

Durante la fase de Inception se acabó de concretar cuáles era las especificaciones y la arquitectura lógica del sistema, y se decidió las tecnologías se iban a utilizar para la implementación de la aplicación web. Estas tecnologías fueron elegidas principalmente porque son tecnologías muy demandadas actualmente en el mercado y porque son muy flexibles y escalables.

9.1.1. MERN Stack

El MERN stack [\[35\]](#) es una fusión de diversas tecnologías que utilizan Javascript como lenguaje de programación. MERN son las siglas de MongoDB, Express.js, React.js y Node.js, que son las 4 tecnologías principales que se utilizarán a lo largo de este proyecto. En la imagen 10 podemos observar las 3 capas de nuestra arquitectura (vista en el apartado 8.3) y qué tecnologías del MERN vamos a usar en cada una de ellas.

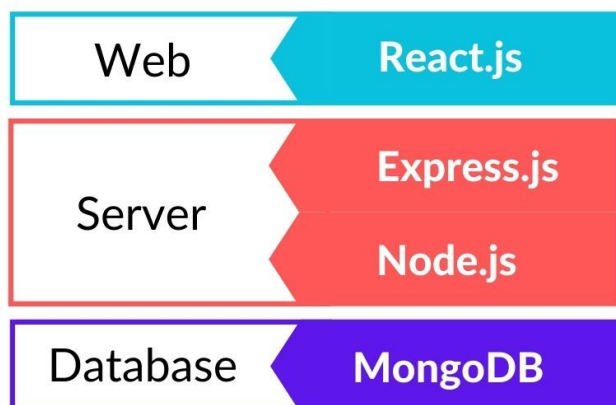


Imagen 28. Arquitectura de MERN Stack [\[36\]](#)

Capa de presentación

React [\[36\]](#) es un *framework*¹⁹ *open-source* de Javascript desarrollado por Facebook, que permite crear interfaces de usuario interactivas de manera rápida, eficiente y con menos código. Un componente de react es un bloque de código independiente y reutilizable, que devuelven código HTML. Esto lo realiza a través de JSX, una librería que transforma código escrito en Javascript en elementos HTML y los inserta en el DOM²⁰. Gracias a React, puedes crear un elemento de la interfaz una única vez y reutilizarlo todas las veces que

¹⁹ **Framework:** Marco de trabajo que ofrece una estructura base para simplificar tareas y asegurar código de calidad.

²⁰ **DOM:** Siglas de Document Object Model, que hace referencia al árbol de etiquetas HTML que forman una página web.

quieras. Los componentes son completamente independientes entre ellos, con lo cual son fáciles de mantener. Y la flexibilidad que permite poder montar una página web tratando cada componente como si fuera una pieza de lego, ha logrado que React sea una de las tecnologías más importantes del mercado actual.

Otra de las características más importantes que nos proporciona react es el DOM virtual. Es una representación del DOM guardada en memoria que actúa como intermediario entre los estados de los componentes en react y los estados del DOM. Cuando se va a realizar algún cambio en el DOM, React compara la diferencia entre el DOM y el Virtual DOM y actualiza únicamente las diferencias entre ambos, ahorrando recursos de procesamiento y proporcionando una interfaz mucho más fluida.

React es una tecnología bastante compleja de entender que tiene múltiples características. Las iremos explicando con mayor detalle en el apartado de implementación conforme vayamos avanzando en el proyecto. Así podremos ver cómo funcionan y cómo nos pueden ayudar en nuestro sistema.

Capa de aplicación

Node.js es un entorno de ejecución de Javascript que permite crear aplicaciones escalables, ya que gestiona todos los eventos de manera asíncrona. Permite ejecutar un gran número de operaciones de manera concurrente y en el caso de que no haya trabajo por hacer, se queda a la espera consumiendo muy pocos recursos. Aparte de esto, Node.js trae incluido NPM, un gestor de paquetes que permite a los desarrolladores compartir herramientas indispensables para el funcionamiento de sus aplicaciones web.

Por otro lado, express es un *framework* de Node.js que proporciona un sistema de enrutamiento que facilita las peticiones HTTP y la gestión de los *middlewares*. Aporta mayor velocidad de E/S de datos en las peticiones, utiliza la estructura MVC para simplificar las manipulaciones de datos de los sistemas de enrutamiento y es muy flexible y fácil de utilizar.

Capa de datos

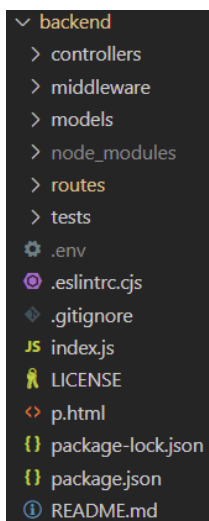
Finalmente para la capa de datos utilizaremos MongoDB Atlas. MongoDB es una base de datos NoSQL orientada a documentos, que ofrece gran escalabilidad y flexibilidad gracias a un modelo de consultas e indexación avanzado. En vez de utilizar tablas como podríamos ver en una base de datos SQL tradicional, utiliza una estructura de datos llamada BSON, que es muy similar al formato JSON. MongoDB nos proporciona múltiples herramientas para poder hacer consultas avanzadas o replicación de documentos y permite ejecutarse simultáneamente en varios servidores, lo cual aporta una mayor escalabilidad y protección en caso de un fallo de hardware.

9.2. Tecnologías aplicadas

Como ya se ha comentado anteriormente, la implementación del proyecto ha sido dividida en 4 *sprints*. En el primer *sprint*, a pesar de haber destinado las funcionalidades más sencillas de realizar del proyecto, supuso una carga de trabajo mayor de la esperada debido a mi inexperiencia con React. Se tuvo que aprender esta tecnología prácticamente desde cero y cómo aplicar correctamente los patrones de diseño vistos en el apartado 8.4. Además, también supuso bastante trabajo crear una UI fácil de utilizar para el usuario, ya que es un requisito no funcional importante para el proyecto.

9.2.1. Estructura del proyecto

Para empezar hablaremos de la estructura del proyecto. Tanto el back-end como el front-end van a compartir una serie de archivos. Primero de todo, tenemos los archivos “package” predefinidos de node.js que mantienen la configuración del proyecto, luego tenemos el archivo “.env” que mantiene todas las variables secretas de nuestro proyecto y por último el archivo de “eslint”, que es un linter²¹ de Javascript que se encarga de analizar el código para eliminar malas prácticas y solucionar errores de sintaxis. A partir de ahí, ambas estructuras serán diferentes.



Empecemos hablando del back-end. “Index.js” contiene el núcleo de nuestra API REST. Es el archivo que mantiene la configuración del servidor, así como la definición de todos los middleware del proyecto. A partir de ahí, podemos ver como el resto de las carpetas son las mismas que en la estructura definida en el apartado 8.6.2. La única diferencia es que añadimos una carpeta más llamada “Routes”, que se encarga de especificar que *middlewares* y controladores se van a ejecutar en cada url recibida en el servidor.

Finalmente, la última carpeta que tendríamos sería la de test, que contendrá todos los tests unitarios de las api call.

Imagen 29. Estructura back-end del proyecto

²¹ Lint: Herramienta que analiza código fuente con el objetivo de identificar fallos o problemas en el sistema.

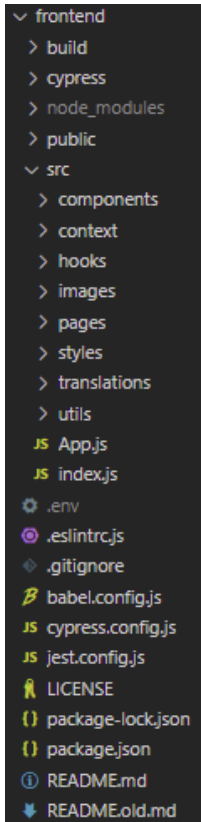


Imagen 30. Estructura front-end del proyecto

De igual manera que con el back-end, vamos a analizar la estructura del proyecto del front-end. La carpeta “public” contendrá el archivo html principal y todos los posibles assets, y la carpeta “build” los assets necesarios para subir la aplicación web a producción.

En la carpeta “src” encontraremos las siguientes carpetas: “components” contiene todos los componentes de React, “context” almacena todos los contextos del sistema (para más información mirar apartado 8.4.4), “hooks” contiene todos los custom hooks (para más información mirar el apartado 8.4.3), “pages” contiene los componentes React a ejecutar para cada uno de las posibles rutas, “styles” almacena los archivos css globales, “translations” contiene los archivos para poder tener la aplicación en versión catalana y castellana, y por último la carpeta “utils” que contiene funciones de uso general.

Los test estarán almacenados en la carpeta “cypress”, tecnología de la cual hablaremos en más detalle posteriormente.

9.2.2. Implementación de las funcionalidades

Para simplificar el apartado de la implementación, veremos como se ha implementado el diagrama de secuencia visto en el apartado 8.6.3. Este diagrama de secuencia nos definía el proceso para crear una nueva entrada a un paciente. Empezaremos hablando de los patrones de diseño y el código de la parte del front-end.

```
const [newEntryData, setNewEntryData] = useState({
  motivo: '',
  antecedentes: '',
  clinica: '',
  exploracion: '',
  pruebasComplementarias: '',
  diagnostico: null,
  estado: 'activo',
  descDiagnostico: '',
  planTerapeutico: '',
  prescripciones: []
})
```

Imagen 31. Estados de React

Una de las características más importantes que nos ofrece React son los estados. Un estado es un objeto que almacena información sobre un componente. Los estados son muy importantes en React, ya que cada vez que un estado se modifica se renderiza de nuevo todo el componente con los nuevos cambios aplicados. En la imagen 15 podemos ver como creamos un estado con el hook “useState” con todos los datos de una nueva entrada. Luego solo tenemos que mostrar estos datos al usuario a través de JSX.

Como vimos en el apartado 8.4.4, los contextos nos permiten compartir datos y métodos a un grupo de componentes sin necesidad de utilizar los props. Para crear una nueva entrada utilizaremos dos contextos diferentes: *globalContext* nos proporcionará la información del trabajador que ha abierto la sesión y *patientContext* la información del paciente al que el estamos creando la nueva entrada. Por ejemplo, en la imagen 16 podemos ver cómo hemos creado el *patientContext*. Primero de todo, creamos un contexto y se lo asignamos con el Provider en *PatientContextProvider*. Esta función nos devuelve el contexto con todos los datos y el método para modificarlos. Para modificar el contexto se utiliza el hook de react *useReducer*, una función muy parecida a *useState* que nos permite simplificar las modificaciones añadiendo una acción que especifica el cambio que desea hacer sobre el objeto.

```
const patientContext = createContext(INITIAL_STATE)

export const patientReducer = (state, action) => {
  let patient = null
  switch (action.type) {
    case 'UPDATEPATIENT':
      return { ...state, patient: action.payload.patient }
    case 'UPDATEPATIENTANDAI':
      patient = action.payload.dataPatient
      patient.inteligenciaActiva = action.payload.dataAi
      return { ...state, patient }
    default:
      return null
  }
}

export const PatientContextProvider = ({ children }) => {
  const [patientData, dispatch] = useReducer(patientReducer, INITIAL_STATE)

  return (
    <patientContext.Provider value={{ patientData, dispatch }}>
      {children}
    </patientContext.Provider>
  )
}
```

Imagen 32. Creación del *patientContext*

Por último, para comunicarnos con el servidor utilizaremos los custom hooks especificados en el apartado 8.4.3. Para todas las api call, hemos creado un custom hook encargado de separar la lógica de las peticiones API respecto a los componentes de react.

Para crear las api calls, hemos utilizado una librería llamada Axios que permite hacer peticiones HTTP a través de promesas²². En la imagen 17 podemos observar cómo utilizamos axios para crear una petición HTTP de tipo post. La url recibida por parte del componente de React es “/entries/createEntry” y los parámetros, los cuales se pasan como body de la petición, es la información necesaria para crear una entrada. También podemos observar cómo se añade en el apartado de *headers* de la petición el apartado “Authorization”, que va a llevar incorporado el token del trabajador, extraído del *globalContext*. Finalmente, al recibir el resultado del back-end, se actualiza “data” y el componente de React renderiza los nuevos datos al usuario a través de JSX.

²² Promesa: Función que se ejecuta de manera asíncrona y ejecuta un código al acabar.

```

const usePost = (url) => {
  const [data, setData] = useState()
  const [loading, setLoading] = useState(false)
  const [error, setError] = useState(false)
  const { globalData } = useGlobalContext()
  const navigate = useNavigate()

  const postData = async (url, params) => {
    setLoading(true)
    setError(false)
    setData()
    try {
      await axios.post(process.env.REACT_APP_URL + url, params, {
        headers: {
          Authorization: `Bearer ${globalData.token}`
        }
      }).then((response) => {
        setData(response)
      })
    } catch (error) {
      if (error.request.status === 403) navigate('/app/login')
      setError(error)
    }
    setLoading(false)
  }

  return { postData, data, loading, error }
}

```

Imagen 33. Custom hook de la api call

Una vez enviada la petición HTTP, el servidor del back-end la recibe y la envía al middleware de entries, el cual antes de enviar la petición al controlador se la envía al middleware de “verifyUser” para comprobar que el token es correcto. Como podemos observar en la imagen 18, esta función se encarga de obtener el token del apartado de *headers* de la petición y descryptar el id del trabajador a través de la librería “jsonwebtoken”. Se realiza una petición en la BD para comprobar que el trabajador exista y si lo hace se envía la petición al controlador con la función “next” vista en el apartado 8.4.2. De igual manera, en caso de que haya algún error porque el token no es válido o porque no existe, la petición se envía al middleware de “handleError” para enviar el error como respuesta de la petición.

```

export const verifyUser = async (req, res, next) => {
  // verify authentication
  const { authorization } = req.headers
  if (!authorization) {
    return next(handleError(401, 'You are not authenticated!'))
  }
  const token = authorization.split(' ')[1]
  try {
    const validation = jwt.verify(token, process.env.JWT_SECRET)
    try {
      req.user = await Worker.findOne({ _id: validation.id })
      next()
    } catch (error) {
      next(handleError(404))
      console.log(error)
    }
  } catch (error) {
    next(handleError(403, 'You are not authorized!'))
  }
}

```

Imagen 34. Función de verifyUser

Finalmente, la petición llega al controlador, el cual se encarga de crear la entrada gracias al esquema de “Entrada” del modelo, de añadir la entrada al paciente, actualizar los objetivos del trabajador si es necesario y finalizar la petición.

Con esto finaliza el apartado de implementación. Si se desea ver más información, puede acceder al repositorio tanto de back-end [\[37\]](#) como de front-end [\[38\]](#).

10. Validación y pruebas

En este apartado hablaremos de cómo hemos realizado la validación de los requisitos funcionales y no funcionales. En ambos casos se mostrarán qué herramientas se han utilizado, y los test y resultados que se han realizado a lo largo del proyecto.

10.1. Validación de los requisitos funcionales

La fase de testeo es muy importante durante la implementación de una aplicación porque podemos romper funcionalidades conforme vamos creando código sin darnos cuenta. Es por este motivo que en cada *sprint* se ha realizado una fase de testeo completamente automatizada, que se encarga de comprobar que toda nuestra aplicación web funcione según lo esperado, y en caso contrario detectar con facilidad dónde se encuentra el error.

El back-end y el front-end son partes de la implementación muy diferentes entre sí y a menudo se piensa que con crear test unitarios para el lado del back-end es más que suficiente para testear una aplicación, dejando de banda la interfaz de usuario. Como esta es una idea equivocada, en este trabajo se han creado tests tanto para back-end como para front-end. Aunque como veremos a continuación se ha utilizado una tecnología diferente para cada uno de ellos.

10.1.1. Back-end

Para empezar hablaremos de cómo se han implementado los tests del lado del back-end. En este apartado testeamos principalmente que las API calls funcionan como es esperado. Para realizar los test unitarios se ha utilizado Jest, el testing framework más popular para Javascript. Se han utilizado 3 de sus funcionalidades más importantes:

- Describe: Nos permite crear bloques de test para agrupar los test que estén relacionados.
- Test: Contiene el código para testear todas las funciones.
- Expect: Permite validar las respuestas a lo largo del test.

Para cada test, hacemos la api call con la url que queremos comprobar y con expect validamos que el resultado es el que esperamos. A continuación, se especificarán con una tabla los tests relacionados entre sí en el escenario que define el test, el resultado final y las observaciones que hayan surgido durante la ejecución del test.

Tests relacionados con la autenticación		
Escenario	Resultado	Observaciones
El usuario intenta iniciar sesión con las credenciales correctas.	Éxito	El servidor valida las credenciales y envía un código 200.
El usuario intenta iniciar sesión con un username que no existe.	Éxito	El servidor detecta que el usuario no existe y devuelve un error 403.
El usuario intenta iniciar sesión con una contraseña incorrecta.	Éxito	El servidor detecta que el usuario no está autenticado y devuelve un error 400.
El usuario intenta hacer una petición sin token.	Éxito	El servidor detecta que el usuario no está autenticado y devuelve un error 401.
El usuario intenta hacer una petición con un token inválido.	Éxito	El servidor detecta que el token no es correcto y devuelve un error 403.

Tests relacionados con los trabajadores		
Escenario	Resultado	Observaciones
El usuario intenta modificar su lenguaje de preferencia.	Éxito	El servidor realiza la modificación en la BD y envía un código 200.
El usuario intenta conseguir un listado de medicamentos recomendados.	Éxito	El servidor busca qué medicamentos suele recetar el usuario para un diagnóstico, y envía la respuesta y un código 200.

Tests relacionados con los pacientes		
Escenario	Resultado	Observaciones
El usuario intenta buscar a un paciente con unos parámetros de búsqueda.	Éxito	El servidor devuelve una lista con los pacientes que encajan con los parámetros de búsqueda y devuelve un código 200.
El usuario intenta conseguir la información de un paciente que existe.	Éxito	El servidor devuelve la información del paciente y un código 200.
El usuario intenta conseguir la información de un paciente que no existe.	Éxito	El servidor devuelve un error 404.
El usuario intenta conseguir la inteligencia activa de un paciente que existe.	Éxito	El servidor devuelve la inteligencia activa del paciente y un código 200.
El usuario intenta conseguir la inteligencia activa de un paciente que no existe.	Éxito	El servidor devuelve un error 404.

Tests relacionados con las entradas		
Escenario	Resultado	Observaciones
El usuario intenta crear una nueva entrada.	Éxito	Se crea la nueva entrada en la BD, se añade la entrada a la lista de entradas del paciente, se actualizan los objetivos del trabajador y el servidor devuelve un código 201.
El usuario intenta crear una nueva nota.	Éxito	Se crea la nueva nota en la BD, se añade la nota a la lista de notas de la entrada del paciente, se actualizan los objetivos del trabajador y el servidor devuelve un código 201.
El usuario intenta modificar una nota.	Éxito	Se modifica la entrada de la BD, se actualizan los objetivos del trabajador y el servidor devuelve un código 200.
El usuario intenta eliminar una nota.	Éxito	El servidor elimina la nota de la BD, elimina la entrada si es la última nota de la entrada, actualiza los objetivos del trabajador y devuelve un código 200.
El usuario intenta buscar un diagnóstico con unos parámetros de búsqueda.	Éxito	El servidor devuelve una lista con los diagnósticos que encajan con los parámetros de búsqueda y un código 200.
El usuario intenta traducir una entrada del catalán al castellano.	Éxito	El servidor devuelve la entrada traducida correctamente y un código 200.
El usuario intenta traducir una entrada del castellano al catalán.	Éxito	El servidor devuelve la entrada traducida correctamente y un código 200.

Tests relacionados con las prescripciones		
Escenario	Resultado	Observaciones
El usuario intenta crear una prescripción.	Éxito	Se crea la nueva entrada en la BD, se añade la entrada a la lista de entradas del paciente, se actualizan los objetivos del trabajador y el servidor devuelve un código 201.
El usuario intenta modificar una prescripción.	Éxito	Se modifica la prescripción de la BD, se actualizan los objetivos del trabajador y el servidor devuelve un código 200.
El usuario intenta eliminar una prescripción.	Éxito	El servidor elimina la prescripción de la BD y de la lista de prescripciones del paciente y devuelve un código 200.
El usuario intenta buscar un medicamento con unos parámetros de búsqueda.	Éxito	El servidor devuelve una lista con los medicamentos que encajan con los parámetros y un código 200.

Tests relacionados con las agendas		
Escenario	Resultado	Observaciones
El usuario intenta conseguir las agendas de un centro.	Éxito	El servidor devuelve una lista con todas las agendas asignadas al centro y devuelve un código 200.
El usuario intenta conseguir las visitas asignadas a una agenda.	Éxito	El servidor devuelve una lista con todas las visitas asignadas a la agenda y devuelve un código 200.
El usuario intenta crear una cita previa con datos válidos.	Éxito	Se crea la nueva cita en la BD, se añade a la lista de citas previas del paciente y de la agenda, y el servidor devuelve un código 201.
El usuario intenta crear una cita previa en un horario que el trabajador no tiene asignado.	Éxito	El servidor devuelve un mensaje de error para comunicar que el trabajador no tiene asignado ese horario.
El usuario intenta crear una cita previa con una fecha ya pasada.	Éxito	El servidor devuelve un mensaje de error para comunicar que la fecha no es válida.
El usuario intenta crear una cita previa en un horario en el que el trabajador ya tiene otra cita previa.	Éxito	El servidor devuelve un mensaje de error para comunicar que el trabajador ya tiene asignado ese horario a otra cita previa.
El usuario intenta modificar una cita previa.	Éxito	Se modifica la cita previa de la BD y el servidor devuelve un código 200.
El usuario intenta eliminar una cita previa.	Éxito	El servidor elimina la cita previa de la la BD, de la lista de citas del paciente y de la agenda, y devuelve un código 200.

Tests relacionados con los documentos		
Escenario	Resultado	Observaciones
El usuario intenta crear un nuevo documento.	Éxito	Se crea el nuevo documento en la BD, se añade el documento a la lista de documentos del paciente y el servidor devuelve un código 201.
El usuario intenta ver el documento de un paciente.	Éxito	Se devuelve el documento y un código 200.
El usuario intenta eliminar un documento.	Éxito	El servidor elimina el documento de la la BD y de la lista de documentos del paciente, y devuelve un código 200.

Tests relacionados con las estadísticas		
Escenario	Resultado	Observaciones
El usuario intenta generar un listado de pacientes con unos parámetros de búsqueda.	Éxito	El servidor devuelve una lista con los pacientes que encajan con los parámetros de búsqueda y devuelve un código 200.
El usuario intenta conseguir la lista de los objetivos que tiene asignados.	Éxito	El servidor devuelve la lista de objetivos que tiene el trabajador asignados y un código 200.
El usuario intenta conseguir la lista de pacientes asignados a un objetivo.	Éxito	El servidor devuelve la lista de pacientes que tiene el objetivo asignado y un código 200.

10.1.2. Front-end

Por último, explicaremos cómo hemos realizado los test del lado del front-end. Cypress es una herramienta de testing para Javascript “end-to-end”, un tipo de testing que busca testear las funcionalidades desde el punto de vista del usuario final. Gracias a Cypress podemos automatizar procesos como si fuésemos usuarios, inspeccionar elementos del DOM y comprobar si estos se han modificado según lo esperado. Para mostrar los tests, se seguirá el mismo procedimiento que en el apartado anterior.

Tests relacionados con la autenticación		
Escenario	Resultado	Observaciones
El usuario intenta acceder a una url sin estar autenticado.	Éxito	Se redirige al usuario a la página de login.
El usuario intenta iniciar sesión con las credenciales correctas.	Éxito	Se redirige al usuario a la página principal.
El usuario intenta iniciar sesión con un username que no existe.	Éxito	Se notifica al usuario que el usuario no existe.
El usuario intenta iniciar sesión con una contraseña incorrecta.	Éxito	Se notifica al usuario que la contraseña es incorrecta.
El usuario intenta cerrar sesión.	Éxito	Se redirige al usuario a la página de login.

Tests relacionados con la configuración		
Escenario	Resultado	Observaciones
El usuario intenta cambiar la configuración de la aplicación.	Éxito	Se notifica al usuario de que los cambios han sido guardados.

Tests relacionados con la pestaña principal		
Escenario	Resultado	Observaciones
El usuario intenta buscar a un paciente con unos parámetros de búsqueda.	Éxito	Se muestra al usuario la lista de los pacientes que encajan con los parámetros de búsqueda.
El usuario intenta ver la información de una agenda.	Éxito	Se muestra al usuario la lista de los pacientes que tienen una visita asignada a la agenda y fecha seleccionada.
El usuario intenta cambiar la fecha de la agenda.	Éxito	Se muestra al usuario la lista de los pacientes que tienen una visita asignada a la agenda y fecha seleccionada.
El usuario intenta crear una cita nueva.	Éxito	Se muestra la agenda actualizada con la nueva cita.
El usuario intenta abrir el curso clínico de un paciente.	Éxito	Se muestra el curso clínico de un paciente con toda la información relacionada.

Tests relacionados con las entradas		
Escenario	Resultado	Observaciones
El usuario intenta ver las entradas de un paciente.	Éxito	Se muestra al usuario las entradas de un paciente.
El usuario intenta filtrar las entradas según un diagnóstico.	Éxito	Se muestra al usuario la lista de entradas que tienen una nota con el diagnóstico seleccionado.
El usuario intenta traducir una entrada.	Éxito	Se muestra al usuario la entrada traducida.
El usuario intenta ver una nota de un paciente.	Éxito	Se muestra al usuario la nota.
El usuario intenta crear una nota nueva.	Éxito	Se muestra al usuario un formulario para crear la nota y una vez creada aparece la nueva nota en la lista de notas del paciente.
El usuario intenta importar el apartado de antecedentes.	Éxito	Aparece escrito el apartado de antecedentes asignado al paciente.
El usuario intenta ver una recomendación para el diagnóstico.	Éxito	Se muestra al usuario un listado de recomendaciones según lo que haya escrito en el apartado de clínica.
El usuario intenta modificar una nota.	Éxito	Se muestra al usuario un formulario para modificar la nota y una vez modificada aparecen los cambios en la lista de entradas del paciente.
El usuario intenta eliminar una nota.	Éxito	Se muestra al usuario la lista de notas del paciente sin la nota eliminada.

Tests relacionados con las prescripciones		
Escenario	Resultado	Observaciones
El usuario intenta ver los principios activos que tiene recetado el paciente.	Éxito	Se muestra al usuario los principios activos que tiene recetado actualmente el paciente.
El usuario intenta ver el listado de prescripciones del paciente.	Éxito	Se muestra al usuario el listado de prescripciones vigentes del paciente y si el paciente tiene una alergia a uno de los medicamentos aparece el nombre marcado en rojo.
El usuario intenta ver las recomendaciones de medicamentos.	Éxito	Se muestra al usuario un listado con los medicamentos que más ha recetado para ese diagnóstico.
El usuario intenta ver una prescripción de un paciente.	Éxito	Se muestra al usuario la prescripción.
El usuario intenta crear una prescripción.	Éxito	Se muestra al usuario un formulario para crear la prescripción y una vez creada aparece la nueva prescripción en la lista de prescripciones del paciente.
El usuario intenta modificar una prescripción.	Éxito	Se muestra al usuario un formulario para modificar la prescripción y una vez modificada aparecen los cambios en la lista de prescripciones del paciente.
El usuario intenta eliminar una prescripción.	Éxito	Se muestra al usuario la lista de prescripciones del paciente sin la prescripción eliminada.

Tests relacionados con los documentos		
Escenario	Resultado	Observaciones
El usuario intenta ver la tabla resumida de los documentos de un paciente.	Éxito	Se muestra al usuario la tabla resumida de los documentos del paciente.
El usuario intenta ver el listado de documentos del paciente.	Éxito	Se muestra al usuario el listado de documentos del paciente ordenados cronológicamente en orden descendente.
El usuario intenta ver el documento.	Éxito	Se muestra al paciente el documento abierto en una nueva pestaña.
El usuario intenta eliminar el documento.	Éxito	Se muestra al usuario la lista de documentos del paciente sin el documento eliminado.

Tests relacionados con la inteligencia activa		
Escenario	Resultado	Observaciones
El usuario intenta ver la inteligencia activa resumida de un paciente.	Éxito	Se muestra al usuario el resumen de la inteligencia activa del paciente.
El usuario intenta ver la inteligencia activa detallada de un paciente.	Éxito	Se muestra al usuario la inteligencia activa del paciente.

Tests relacionados con las visitas		
Escenario	Resultado	Observaciones
El usuario intenta ver la tabla resumida de las visitas de un paciente.	Éxito	Se muestra al usuario la tabla resumida de las visitas del paciente.
El usuario intenta ver el listado de visitas de un paciente.	Éxito	Se muestra al usuario el listado de visitas del paciente ordenadas cronológicamente en orden descendente.
El usuario intenta crear una nueva visita a un paciente.	Éxito	Se muestra al usuario un formulario para crear la visita y una vez creada aparece la nueva visita en la lista de visitas del paciente.
El usuario intenta ver una visita.	Éxito	Se muestra al paciente la vista del paciente.
El usuario intenta modificar una visita.	Éxito	Se muestra al usuario un formulario para modificar la visita y una vez modificada aparecen los cambios en la lista de visitas del paciente.
El usuario intenta eliminar una visita.	Éxito	Se muestra al usuario la lista de visitas del paciente sin la visita eliminada.

Tests relacionados con las estadísticas		
Escenario	Resultado	Observaciones
El usuario intenta ver la lista de objetivos que tiene asignado.	Éxito	Se muestra al usuario la lista de los objetivos y su progreso. Si el progreso es menor a la mitad del objetivo se muestra el porcentaje en rojo, si es mayor a la mitad del objetivo se muestra en amarillo, de lo contrario aparece en verde.
El usuario intenta ver la información de un objetivo.	Éxito	Se muestra al usuario la información relacionada a un objetivo.
El usuario intenta ver los listados de pacientes asignados a un objetivo.	Éxito	Se muestra al usuario el listado de pacientes asignados a un objetivo.
El usuario intenta generar un listado según unos parámetros de búsqueda.	Éxito	Se muestra al paciente el listado de pacientes que encajan con los parámetros de búsqueda.

10.2. Validación de los requisitos no funcionales

A continuación, vamos a comprobar que se hayan cumplido los requisitos no funcionales del proyecto. Para ello miraremos si hemos conseguido cumplir con los criterios de aceptación que definimos en el apartado 3.2.2.

11a. Requisito de Facilidad de Utilización.		
Criterio de satisfacción	Resultado	Observaciones
Al 75% de los testers les ha de parecer fácil de utilizar.	Éxito	A todos los <i>stakeholders</i> encuestados les ha parecido fácil de utilizar.

11b. Requisito de Personalización e Internacionalización.		
Criterio de satisfacción	Resultado	Observaciones
La aplicación web estará disponible tanto para catalán como para castellano.	Éxito	Se ha creado la aplicación en ambos idiomas.

12a. Requisitos de Velocidad y Latencia.		
Criterio de satisfacción	Resultado	Observaciones
El sistema responderá en menos de 1 segundo y medio para el 90 por ciento de las llamadas API.	Éxito	El 93,75% de las llamadas API tardan menos de 1 segundo y medio en responder.

12d. Requisitos de Confiabilidad y Disponibilidad.		
Criterio de satisfacción	Resultado	Observaciones
El sistema se encontrará operativo las 24 horas al día, 365 días al año.	Éxito	De los 3 servicios de host que se han escogido para desarrollar la aplicación, Nellify y Railway han presentado problemas de latencia pero no han afectado a la disponibilidad del sistema.

15a. Requisitos de Acceso.		
Criterio de satisfacción	Resultado	Observaciones
El sistema controlará que la api sólo pueda ser accedida si el usuario se encuentra logueado y las contraseñas se almacenarán en la base de datos encriptadas.	Éxito	Se ha creado una API REST que solo permite su uso a personas autorizadas para garantizar que nadie no autorizado tenga acceso a los datos y se ha encriptado la contraseña.

15c. Requisitos de Privacidad.		
Criterio de satisfacción	Resultado	Observaciones
Se hará un estudio sobre las leyes y regulaciones vigentes, y se asegurará que el producto cumple con la RGPD.	Éxito	Se ha estudiado con detalle la RGPD y se ha decidido encriptar todos los datos sensibles que se almacenan en la BD.

11. Identificación de leyes y regulaciones

Aunque este proyecto sea una aplicación web de uso académico, es imprescindible comprender a la perfección el Reglamento General de Protección de Datos [\[39\]](#). La comúnmente llamada RGPD es un reglamento que unifica todas las leyes de protección de datos a nivel europeo. Estas leyes están enfocadas principalmente a asegurar que se esté realizando un uso correcto y justificado de los datos, y que en todo momento estos hayan sido proporcionados con el permiso explícito del usuario al cual pertenecen.

Cabe remarcar que pese a que nosotros en ningún momento vamos a tratar con datos sensibles de verdad, vamos a aplicar las leyes como si estos datos fueran de pacientes reales.

A continuación, vamos a nombrar las normativas que afectan a nuestro proyecto:

- Artículo 5 - Principios relativos al tratamiento: Estipula que todos los datos recogidos serán utilizados de manera lícita, leal y transparente, y serán recogidos con fines determinados, explícitos y legítimos. Los datos han de ser limitados a lo necesario para cumplir con su fin. También justifica que podamos conservar los datos durante períodos más largos al ser utilizados para fines estadísticos y de investigación científica. Finalmente, se ha de garantizar una seguridad adecuada para los datos personales.
- Artículo 6 - Licitud del tratamiento: Se verifica la licitud del tratamiento de nuestro proyecto a través de la condición 1-E, la cual especifica lo siguiente “el tratamiento es necesario para el cumplimiento de una misión realizada en interés público o en el ejercicio de poderes públicos conferidos al responsable del tratamiento”.
- Artículo 7 y 17- Condiciones para el consentimiento: El paciente o trabajador podrá retirar su consentimiento en cualquier momento, pero no tendrá derecho a la supresión de datos debido al punto 3-D del Artículo 17, el cual informa que no será necesaria la supresión en caso de ser utilizados “con fines de archivo en interés público, fines de investigación científica o histórica, o fines estadísticos”.
- Artículo 9 - Tratamiento de categorías especiales de datos personales: Queda justificada que los datos revelen el origen étnico o racial, el tratamiento de datos genéticos, datos biométricos dirigidos a identificar de manera unívoca a una persona física, datos relativos a la salud o datos relativos a la vida sexual o la orientación sexuales de una persona física, debido a la circunstancia 2-H, la cual especifica lo siguiente “el tratamiento es necesario para fines de medicina preventiva o laboral, evaluación de la capacidad laboral del trabajador, diagnóstico médico, prestación de asistencia o tratamiento de tipo sanitario o social, o gestión de los sistemas y servicios de asistencia sanitaria y social”.
- Artículo 12 - Transparencia de la información, comunicación y modalidades de ejercicio de los derechos del interesado: En caso de que lo solicite el interesado, se ha de tomar las medidas oportunas para facilitarle todos sus datos.

- Artículo 32 - Seguridad del tratamiento: Se han de aplicar medidas de seguridad como el cifrado de datos personales o garantizar la confidencialidad, integridad, disponibilidad y resiliencia de los servicios del tratamiento. También se han de tener en cuenta los riesgos de destrucción, pérdida o alteración accidental o ilícita de datos personales y se ha de asegurar controlar que el responsable tiene la autoridad para acceder a los datos.
- Artículo 33 - Notificación de una violación de la seguridad de los datos personales a la autoridad de control: En caso de una violación de la seguridad de los datos, el responsable ha de notificar a las autoridades de control en un plazo de 72 horas.
- Artículo 34 - Comunicación de una violación de la seguridad de los datos personales al interesado: En caso de que sea probable una violación de seguridad de los datos personales, se ha comunicar al interesado sin mayor retraso. Sin embargo, esto no será necesario en caso de que se hayan adoptado medidas de seguridad que hagan que los datos personales sean ininteligibles para cualquier persona que no esté autorizada a acceder a ellos, como por ejemplo el cifrado. No obstante, si la autoridad de control lo considera oportuno, podrá exigirle que lo haga.

Para asegurar de que se cumplen con la ley vigente se ha decidido encriptar los datos sensibles de la base de datos. Además, como almacenamos datos sensibles en la memoria de almacenamiento del navegador, estos datos también se van a encriptar para mantener su seguridad. Por último, el servicio de MongoDB Atlas ya garantiza la confidencialidad, integridad, disponibilidad y resiliencia de los servicios del tratamiento de los datos, por lo que no tendremos que dedicar recursos en este aspecto.

12. Resultados de la gestión del proyecto

Una vez finalizado el proyecto, vamos a analizar cómo ha ido la gestión del proyecto a lo largo de los últimos 5 meses. Veremos en profundidad cómo se ha aplicado la metodología de trabajo, las desviaciones temporales y económicas, y por último el informe de sostenibilidad del proyecto.

12.1. Resultados a nivel de metodología

12.1.1. Métodos de trabajo

Durante todo el proyecto no se ha realizado ninguna modificación respecto a la metodología de trabajo. Gracias a la metodología scrum se han podido realizar historias de usuario, comprobar que se adecuan a los requisitos funcionales gracias a los criterios de aceptación, ordenarlas por prioridad, organizarlas en distintos *sprints*, entender a la perfección el estado del proyecto y posponer tareas de las historias de usuario a futuros *sprints* sin que esto suponga un problema de planificación.

Durante los primeros *sprints* hubo una carga de trabajo mayor a la esperada, pero gracias a la metodología agile, se pudo recuperar las historias de usuario pendientes sin ningún problema en los siguientes dos *sprints*. Además, dividir la implementación en diferentes *sprints* me ha permitido entender en todo momento cómo iba respecto al progreso del proyecto y por ende dedicar tiempo a otras tareas del proyecto como por ejemplo en la documentación o reuniones con los *stakeholders*.

Considero que la elección de la metodología ha sido un punto clave para el proyecto y una buena decisión, ya que me ha ayudado a entender de la mejor manera el abasto decidido durante el curso de GEP y conseguir realizar todas los requisitos funcionales propuestos.

12.1.1. Herramientas de seguimiento

Todas las herramientas de seguimiento han sido utilizadas a lo largo del proyecto y han cumplido con su objetivo esperado.

Para empezar, Taiga ha permitido crear las historias de usuario y criterios de aceptación de manera dinámica y sencilla. Ha permitido dividir las historias en tareas más pequeñas y obtener una mayor visión del estado real del proyecto. Poder ver las historias de usuario desglosadas en múltiples tareas ha permitido ver con facilidad el trabajo que quedaba por realizar durante el *sprint*, así como posibles mejoras de las funcionalidades que se podían implementar si al final sobraba tiempo.

La segunda herramienta fue Todoist, una herramienta más sencilla que ha sido de gran ayuda. Me ha permitido apuntar de manera más ágil ideas o tareas y asignarles una fecha límite para realizarlas. Se ha utilizado especialmente para crear tareas relacionadas con la documentación o ideas que proponían los *stakeholders*.

Finalmente, Github ha permitido trabajar de manera muy cómoda en diferentes entornos de trabajo gracias a tener el código en la nube. Además de mantener el código en un espacio seguro, ha permitido mantener un buen control de versiones, lo que permitía volver a versiones antiguas o consultarlas en caso de necesitarlo.

12.1.1. Método de validación

El método de validación ha sido muy efectivo a lo largo del proyecto. El hecho de que los test fueran automatizados ha facilitado mucho el trabajo del *testing* y ha permitido detectar con facilidad las funcionalidades que ya no funcionaban y dónde estaba el error.

Aparte de la implementación, a lo largo de los *sprints* se ha contado con las reuniones de la tutora del proyecto, Cristina Gómez, que ha aportado correcciones que han sido de gran utilidad tanto para el desarrollo del proyecto como de la documentación.

12.2. Resultados a nivel de planificación

Como ya se ha comentado anteriormente, se han producido ciertas desviaciones en la planificación respecto a la planificación inicial. En la tabla 10 podemos observar la comparativa entre las horas estimadas y las horas reales para cada tarea, así como su desviación.

Bloque	Código	Tarea	Duración estimada	Duración real	Diferencia
Gestión de proyecto	GP1	Contextualización y alcance	30	31	-1
	GP2	Planificación temporal	15	10	5
	GP3	Presupuesto y sostenibilidad	15	9	6
	GP4	Documento final	15	14	1
Inception	I1	Diseño de la arquitectura	40	46	-6
	I2	Configuración del entorno de trabajo	5	5	0
	I3	Familiarización con la tecnología	10	28	-18
Sprint 1	S1-1	Sistema de autenticación	20	31	-11
	S1-2	Visualizar curso clínico	35	45	-10
	S1-3	Buscar curso clínico	25	20	5
	S1-4	Retrospectiva sprint 1	5	2	3
	S1-5	Documentación sprint 1	10	6	4
Sprint 2	S2-1	Añadir entrada al curso clínico	40	39	1
	S2-2	Visualizar objetivos	35	34	1
	S2-3	Retrospectiva sprint 2	5	2	3
	S2-4	Documentación sprint 2	10	7	3

Sprint 3	S3-1	Realizar listados de pacientes y exportarlos	35	32	3
	S3-2	Sistema de citas	25	20	5
	S3-3	Retrospectiva sprint 3	5	0	5
	S3-4	Documentación sprint 3	10	6	4
Sprint 4	S4-1	Visualizar agenda global	30	28	2
	S4-2	Despliegue de la aplicación web	5	6	-1
	S4-3	<i>Feedback con los stakeholders</i>	25	22	3
	S4-4	Retrospectiva sprint 4	5		5
	S4-5	Documentación sprint 4	10	10	0
Fase final	F1	Depurado del prototipo	20	10	10
	F2	Finalización de la memoria	30	40	-10
	F3	Preparación de la defensa oral	25	25	0
Total			540	528	12

Tabla 10. Comparativa de horas estimadas y horas reales

Gestión del proyecto

Las primeras 4 semanas del proyecto fueron destinadas a realizar el curso de GEP. La contextualización del proyecto fue sin duda la tarea con más carga de trabajo, como ya se había previsto en la planificación inicial. Sufrió una pequeña desviación de una hora que se recuperó de las horas sobrantes del resto de tareas de la gestión del proyecto.

Inception

Como se puede observar, en este bloque se produjo una gran desviación. Esto es debido a que se subestimó el tiempo necesario para diseñar la arquitectura y crear su documentación, y a la gran inexperiencia con las tecnologías que se decidió utilizar para el proyecto. Ya que esta tarea era importante para crear unas buenas bases para el proyecto, se decidió no escatimar en horas y estudiar lo necesario antes de poder pasar a trabajar en el *sprint* 1.

Sprint 1

Durante este *sprint* se produjeron desviaciones en las dos primeras tareas de nuevo a causa de la inexperiencia con las tecnologías. Se tuvo que estudiar muchas librerías nuevas y como aplicar correctamente todos los patrones de diseño, lo que provocó tener que rehacer múltiples veces un mismo trabajo.

Sprint 2, 3 y 4

Durante el *sprint* 2 todo funcionó según lo esperado y no se produjo ninguna desviación negativa, lo cual permitió recuperar el ritmo perdido en los dos anteriores bloques. Sin embargo, la desviación era demasiado grande por lo que se tuvo que aplazar historias de usuario al *sprint* 3. A pesar de esto, gracias a haber entendido cómo funcionaban las tecnologías, se pudo recuperar en los *sprints* restantes el ritmo propuesto en el diagrama de

gantt. Por último, durante el *sprint* 4 hubo una pequeña desviación, debido a que se decidió cambiar el host del servidor por uno nuevo y se tuvo que rehacer parte del proceso.

Fase final

Como todavía no se ha realizado la tarea de la defensa de la presentación, se ha supuesto que esta tarea no sufrirá ninguna desviación. Partiendo de esta base, podemos observar que al final fueron requeridos 10 horas más de lo planeado inicialmente para la tarea de finalizar la memoria. Como nos encontrábamos ante una desviación positiva, se decidió dedicar más horas a la documentación para obtener el mejor resultado posible.

Por lo tanto se finaliza el trabajo con un excedente de 12 horas, que se pueden utilizar en la tarea de preparar la defensa oral si esta resulta necesitar más tiempo de lo esperado.

12.3. Resultados a nivel económico

Como hemos visto en el apartado anterior, ha habido desviaciones tanto negativas como positivas en el coste de los recursos humanos, lo que ha afectado al presupuesto del proyecto. Para poder calcular de manera correcta las desviaciones, se ha medido el tiempo real de cada rol para cada tarea y se ha comparado con el resultado de la planificación inicial.

Código	Duración Esperada (h)	Coste Esperado (€)	Horas Dedicadas Reales					Duración Real (h)	Diferencia (€)
			JP	AS	DB	DF	TS		
GP1	30	995,01	31					31	1.028,17
GP2	15	497,50	10					10	331,67
GP3	15	497,50	9					9	298,50
GP4	15	497,50	14					14	464,34
I1	40	1.354,17		46				46	1.557,29
I2	5	124,56			3	2		5	124,56
I3	10	244,45			14	14		28	684,47
S1-1	20	519,64		4	8	13	6	31	749,11
S1-2	35	907,64		7	14	15	9	45	1.111,55
S1-3	25	647,82		3	8	6	3	20	504,45
S1-4	5	165,83	2					2	66,33
S1-5	10	331,67	6					6	199,00
S2-1	40	1.054,65		8	14	12	5	39	1.004,39
S2-2	35	899,36		5	14	12	3	34	865,50
S2-3	5	165,83	2					2	66,33
S2-4	10	331,67	7					7	232,17

S3-1	35	946,31		10	12	7	3	32	870,64
S3-2	25	646,62		5	8	6	1	20	534,83
S3-3	5	165,83						0	0,00
S3-4	10	331,67	6					6	199,00
S4-1	30	758,41		5	8	12	3	28	704,86
S4-2	5	122,23			4	2		6	151,33
S4-3	25	586,21		3	4	7	8	22	512,79
S4-4	5	165,83						0	0,00
S4-5	10	331,67	10					10	331,67
F1	20	445,26			2	4	4	10	216,67
F2	30	995,01	40					40	1.326,68
F3	25	829,17	25					25	829,17
Total	540	15.559,03	162	96	113	112	45	528	14.965,49

Tabla 11. Comparativa de horas estimadas y horas reales

Como se puede observar, en los 3 primeros bloques se ha producido una desviación económica negativa a causa del incremento de horas dedicadas. Sin embargo, en el resto de los sprints se ha conseguido reducir las horas invertidas en cada rol, lo que ha conseguido disminuir **593,54** euros el coste total del proyecto por parte de los recursos humanos.

En cuanto a los recursos materiales, únicamente ha habido una pequeña desviación. Se ha añadido el servidor de pago Railway [\[40\]](#) para hostear la REST API del proyecto, ya que el gratuito no cumplía con los requisitos mínimos del proyecto. Como solo ha estado despegado el servidor 1 mes y el precio es de 5 €/mes, el coste final en recursos materiales ha sido de **893,5** euros. Como el excedente se puede pagar con el sobrante conseguido en los recursos humanos, no se necesita recurrir al plan de contingencia ni al de imprevistos para alcanzar el coste total.

12.4. Informe de sostenibilidad

En el pasado, la sostenibilidad siempre ha sido un aspecto que se solía dejar como segundo plano durante la ejecución de un proyecto. Hoy en día esto es impensable y existen organizaciones que se encargan de medir el impacto que nuestro proyecto dejará en el ámbito económico, social y medioambiental. Para presentar un informe de sostenibilidad que sea capaz de cumplir con estos estándares, se utilizará en práctica los conocimientos adquiridos durante la fase de aprendizaje en la FIB. Sin embargo, se ha de admitir que los conocimientos impartidos por la universidad son bastante escasos en algunos ámbitos, ya que hay ciertos aspectos que solo se dan en asignaturas optativas que puede que no haya cursado. En mi caso particular, he podido aprender bastante sobre: problemas sociales, económicos y ambientales que sufre actualmente la sociedad, el impacto ambiental que produce la creación de los dispositivos electrónicos, los indicadores para medir el impacto social y la viabilidad económica, a gestionar recursos y a interactuar con más agentes de mi mismo ámbito profesional. Pero por otra parte no he podido profundizar en temas como los aspectos deontológicos de mi profesión o consecuencias en la salud, seguridad y justicia social de mis proyectos (a excepción de accesibilidad y experiencia de usuario). Para acabar, mencionar que el informe de sostenibilidad es un apartado que considero muy importante y que aparecerá tanto en este proyecto como en futuros proyectos en los que participe.

Para realizar el informe, se utilizará la matriz de sostenibilidad que se nos ha sido otorgada durante el curso de GEP. Esta matriz se puede observar en la imagen 2.

	PPP	Vida Útil	Riesgos
Ambiental	Consumo de diseño	Huella ecológica	Ambientales
Económico	Factura	Plan de viabilidad	Económicos
Social	Impacto personal	Impacto social	Sociales

Imagen 35. Matriz de sostenibilidad [\[41\]](#)

12.4.1. Sostenibilidad ambiental

Consumo de diseño

En el aspecto medioambiental, gracias a que el proyecto se realiza con el *hardware* del estudiante, todos los dispositivos necesarios serán reutilizados y no habrá ninguna necesidad de adquirir nuevos. En todo momento se ha buscado reutilizar tanto *hardware* como *software*, buscando APIs externas para evitar invertir tiempo y por ende electricidad, o alojando la web en los servidores que cumplieran los requisitos mínimos para el proyecto. Como tampoco se ha producido CO₂ en la elaboración del proyecto, se estima que el coste ambiental en este sentido ha sido mínimo y que no se podía reducir más.

Huella ecológica

Para la vida útil, se ha de tener en cuenta principalmente la energía necesaria para su uso. Al ser una página web, no se requieren muchos recursos para acceder a ella. Y como los servidores son un servicio externo de nuestro sistema, se puede esperar que el consumo de energía que requiere nuestra aplicación web sea mínimo, ya que ellos buscan ganar el máximo beneficio posible. Por lo que nuestro proyecto no va a afectar apenas la huella ecológica.

Riesgos ambientales

El único riesgo a la vista sería en el caso de que la web se encuentre muy mal optimizada, ya que esto provocaría un gasto innecesario de energía por parte del navegador. En nuestro caso, siempre se ha buscado escoger la mejor opción en términos de optimización, aunque la falta de experiencia y de conocimientos pueda ser un factor limitante.

12.4.1. Sostenibilidad económica

Factura

Para calcular el coste, se ha tenido en cuenta los recursos tanto materiales como humanos que son necesarios para realizar el proyecto. Al ser un prototipo que busca mostrar posibles mejoras para el programa sanitario actual, no se busca ningún tipo de recaudación de dinero, aunque se ha buscado que los costes sean mínimos utilizando únicamente los recursos necesarios para desarrollar la aplicación web. En cuanto a cada tarea, se añadió tiempo de más debido a la poca experiencia del estudiante durante la planificación inicial. Sin embargo, al final la mejora en experiencia del estudiante se ha visto recompensada, y no se han requerido tantas horas como se había esperado, por lo que el coste ha sido ligeramente menor al esperado.

Plan de viabilidad

Si queremos mantener que la aplicación web se mantenga operativa, se deberá pagar el coste del servidor de la REST API, el cual son 5 euros al mes. Actualmente no se ha encontrado mejor servidor para el proyecto, sin embargo es posible que en el futuro aparezca una opción igual de viable pero más barata. Por último, en el caso de que el proyecto se quisiera agrandar una vez finalizado, sería necesario realizar un nuevo presupuesto y adaptar el proyecto a los requerimientos mínimos.

Riesgo económico

El peor caso económico en el que nos podemos encontrar es que las mejoras propuestas para el sistema sanitario sean rechazadas, ya que esto supondrá el malgasto de tiempo y de recursos y por lo tanto se perdería toda la inversión inicial. Obviamente en nuestro caso el proyecto es un trabajo educativo, por lo que el impacto es mucho menor que si se tratara de un proyecto real.

12.4.1. Sostenibilidad social

Impacto personal

A nivel personal, este proyecto me ha permitido tratar con tecnologías con las que no había tenido la oportunidad de trabajar y profundizar más en las que sí. Esto me ha permitido obtener experiencia y mejorar mi formación. Además, para los testers que han intervenido en la elaboración del proyecto, les ha encantado la experiencia de participar y se han visto muy satisfechos con el producto final.

Impacto social

En cuanto a la vida útil, si el trabajo consigue llegar a manos del equipo encargado de desarrollar el nuevo programa sanitario, se conseguirá mejorar la calidad de trabajo de un sector tan importante como es el sanitario. Esto implica, no únicamente mejorar su calidad de vida, sino también la de los ciudadanos que requieran de este servicio. Por lo tanto, considero que el proyecto sí que cubre una necesidad real.

El proyecto ha conseguido cumplir con sus objetivos iniciales, consiguiendo simplificar el trabajo del sistema sanitario, mejorando la exportación de datos del sistema y facilitando la comunicación entre diferentes comunidades autónomas.

Riesgos sociales

No considero que puedan existir escenarios en los que el proyecto pueda llegar a ser perjudicial para ningún sector ni que se pudiera crear algún tipo de dependencia, ya que únicamente es una herramienta de trabajo.

13. Conclusiones

Finalmente, para dar por concluido el proyecto realizaremos una retrospectiva sobre si hemos conseguido cumplir con los objetivos propuestos y con las competencias técnicas del proyecto, analizaremos las reflexiones personales que me ha aportado y definiremos el futuro del proyecto.

13.1. Conclusiones del proyecto

Al principio del trabajo definimos una serie de objetivos relacionados con el producto final. Una vez concluida esa parte, podemos ver que los objetivos han sido superados con satisfacción. Se ha conseguido realizar una web intuitiva y fácil de utilizar para el personal sanitario, reducir el tiempo que se invierte en el ordenador para realizar una consulta, mejorar el sistema de exportación de datos y crear una aplicación multilenguaje para que pueda ser utilizada en diferentes comunidades autónomas. Todo ello gracias a los conceptos aprendidos en el grado y a los *stakeholders* que han participado en el proyecto, que han ido aportando ideas para los requisitos del sistema y además han aceptado ser los testers del sistema.

A lo largo del proyecto se han podido aplicar muchos de los conocimientos adquiridos en el grado. Se ha podido realizar una metodología *agile*, detectar y definir unos requisitos adecuados para el proyecto, aplicar técnicas de calidad como los patrones de diseño, trabajar a fondo la implementación de una REST API y trabajar con una base de datos noSQL. Ha sido muy gratificante poder poner a prueba todos estos conocimientos y ser capaz de desarrollar un producto completamente desde cero.

Al principio el proyecto sufrió algunos contratiempos, pero en ningún momento se ha requerido trabajar más de las horas planificadas. Todo ha ido según lo planeado y no ha surgido ningún imprevisto. Obviamente hay muchos aspectos que se pueden mejorar tanto del diseño como de la implementación, pero estoy muy orgullosa del producto final y en general del avance del proyecto.

13.2. Reflexiones personales

Como retrospectiva personal, estoy muy contenta por cómo ha ido el proyecto. Diseñar un sistema desde cero ha sido una muy buena experiencia y he podido poner en práctica muchos conocimientos aprendidos durante el grado. Además, también me ha otorgado la oportunidad de trabajar con tecnologías desconocidas para mí, pero que son muy relevantes en el mercado actualmente, lo cual ha mejorado en gran manera mi formación profesional. Considero que este trabajo es la prueba de los 5 años que llevo estudiando esta carrera y no podría sentirme más orgullosa del resultado final. Sin duda el TFG es una experiencia completamente diferente al resto de asignaturas de la FIB. Sin embargo, gracias a mi tutora Cristina Gómez, he podido adaptarme sin problemas al ritmo y sus ideas y correcciones han sido de gran ayuda a lo largo del proyecto.

Para acabar, estoy muy contenta con la recepción que ha tenido el producto por parte de los *stakeholders*. Desde el principio, mi objetivo era crear una aplicación web que les ayudara

de alguna manera, y conseguirlo y recibir sus valoraciones positivas ha sido muy gratificante. Considero que todas las horas de trabajo han merecido mucho la pena.

13.3. Integración de conocimientos

A lo largo del proyecto se han aplicado una serie de conocimientos que han sido adquiridos durante varios años en las asignaturas de la FIB. A continuación, se nombrarán las asignaturas que han tenido más peso para el proyecto y se describirán los conocimientos que hemos podido aplicar en el proyecto:

- PRO1 y PRO2 - Programación 1 y programación 2: Proporcionaron los conceptos base para aprender la lógica básica para poder programar no solo en C++, sino en cualquier lenguaje de programación.
- EDA - Estructura de Datos y Algoritmos: Proporcionaron conceptos sobre el coste de los algoritmos y su eficiencia, y se profundizó el aprendizaje de las estructuras de datos más imprescindibles.
- BD - Bases de datos: Proporcionó conocimientos sobre bases de datos Sql, que pese a no es la opción escogida para el proyecto, es imprescindible para poder entender correctamente el funcionamiento de una base de datos.
- IDI - Interacción y diseño de interfaces: Proporcionó conocimientos básicos sobre cómo crear una interfaz de usuario eficiente, intuitiva y accesible, aspectos fundamentales para mejorar la experiencia de usuario.
- IES - Introducción de la Ingeniería del Software: Proporcionó conocimientos básicos para la elaboración de la especificación y diseño de un buen sistema *software*, entre los cuales se encuentran la notación UML y los principales patrones de diseño.
- PROP - Proyectos de programación: Proyecto que permitió consolidar los conocimientos adquiridos en las anteriores asignaturas, especialmente el patrón de diseño MVC.
- AS - Arquitectura del Software: Asignatura que profundizó los conocimientos adquiridos en IES como la arquitectura en capas o las metodologías. También se aprendió a realizar test unitarios.
- ASW - Aplicaciones y servicios web: Proporcionó conceptos tanto de back-end como de front-end, se profundizó en la importancia de los frameworks y permitió desarrollar un proyecto completamente desde 0 utilizando diferentes tecnologías.
- DBD - Diseño de base de datos: Ha proporcionado conocimientos sobre la concurrencia en una base de datos y sobre cómo mejorar el rendimiento de las bases de datos a través de estructuras como clusters o índices.

- ER - Ingeniería de Requisitos: Proporcionó conocimientos básicos sobre cómo elaborar un buen análisis de los requisitos de un sistema y la importancia de los *stakeholders*.
- GPS - Gestión de Proyectos de Software: Proporcionó conocimientos relacionados con las diferentes metodologías y permitió realizar la especificación completa y el diseño de un proyecto.
- PES - Proyecto de Ingeniería del Software: Permitted elaborar un proyecto utilizando una metodología *agile* y proporcionó conocimientos sobre cómo contextualizar un proyecto, desarrollar una memoria, realizar un buen análisis del mercado y finalmente utilizar herramientas de soporte como Taiga.
- CAP - Conceptos Avanzados de Programación: Proporcionó conocimientos básicos sobre Javascript y profundizó en conceptos avanzados sobre orientación a objetos como el Prototype o los scopes.
- CBDE - Conceptos para las Bases de Datos Especializadas: Proporcionó conocimientos básicos sobre cómo funcionan las bases de datos NoSql, enseñó las principales diferencias entre las más populares y permitió experimentar con ellas.

13.4. Justificación de las competencias

A continuación, se detalla cómo se han cumplido las competencias técnicas escogidas en la matrícula del proyecto.

CES1.1 - Desarrollar, mantener y evaluar sistemas y servicios software complejos y/o críticos. [Alta]

A partir de un problema real se ha construido un sistema *software* complejo desde cero para solventarlo. Se han utilizado los conocimientos adquiridos en el grado para identificar correctamente los requisitos del sistema, diseñar la arquitectura y la especificación del sistema, implementarlo, crear un buen sistema de testing y evaluar el sistema. Se ha conseguido implementar dos aplicaciones independientes (REST API y aplicación web), que se comunican entre sí y con más servicios externos como la BD y APIs externas para cumplir con los requisitos del sistema. Debido a esto, se considera esta competencia como lograda en un nivel alto.

CES1.4 - Desarrollar, mantener y evaluar sistemas y servicios y aplicaciones distribuidas con el soporte de la red. [Media]

Para la elaboración del proyecto, se ha tenido que implementar un back-end y un front-end que se encuentran alojados en distintos servidores. Además, también ha sido necesario comunicarse con servicios externos como MongoDB o la API externa de MyMemory. Se ha aprendido a realizar peticiones entre los distintos servidores y a utilizar funciones asíncronas para no tener que parar en ningún momento la aplicación web. Como se ha

conseguido realizar una comunicación entre todos los componentes de manera eficiente, se considera esta competencia lograda con el nivel medio.

CES1.5 - Especificar, diseñar, implementar y evaluar base de datos. [Media]

Pese a no ser el objetivo principal del proyecto, la elección de la base de datos ha sido un punto clave para el proyecto. Se ha realizado un estudio sobre las bases de datos más populares y se ha escogido la que más beneficia al proyecto. Se tuvo que diseñar la estructura de los datos de la BD completamente desde cero y se han aprendido conceptos avanzados para realizar las *query* necesarias. Es por esto, que se considera como logrado el nivel medio de esta competencia técnica.

CES1.7 - Controlar la calidad y diseñar pruebas en la producción de software [Baja]

El testing es un punto clave para implementar cualquier sistema *software*. Es por ello que se ha puesto mucho énfasis en conocer las tecnologías actuales más óptimas para realizar esta tarea. Se han realizado test automatizados de todas las historias de usuario, tanto para front-end como para back-end, y se han realizado encuestas y estudios de la aplicación para validar los requisitos no funcionales. Se considera esta competencia técnica lograda con el nivel indicado.

CES2.1 - Definir y gestionar los requisitos de un sistema software. [Baja]

Desde el principio del proyecto se le ha dado mucha importancia definir con todos los *stakeholders* los requisitos funcionales y no funcionales del sistema. Se ha realizado un estudio detallado de cada requisito y de los criterios de aceptación que se tenían que cumplir para validarlos. Para los requisitos no funcionales, se ha estudiado en profundidad la plantilla de Volere y se han escogido unos criterios de aceptación adecuados. Es por ello que se considera esta competencia técnica lograda con el nivel indicado.

13.5. Trabajo futuro

Pese a que el trabajo se considera como finalizado, hay un par de ideas que me hubiera gustado implementar si hubiera dispuesto de más tiempo.

- Inteligencia artificial: Actualmente el sistema de recomendaciones de un diagnóstico es un algoritmo muy sencillo. Sería fácilmente reemplazable por un sistema de inteligencia artificial que aprendiese de las escrituras del personal sanitario conforme estos vayan escribiendo nuevas consultas. No se ha podido implementar debido a que no tengo suficiente conocimiento para hacerlo y hubiera requerido mucho más tiempo del que se tenía planificado. Sin embargo, considero que aportaría muchísima más precisión y valor al actual sistema de recomendaciones.
- Añadir funcionalidades de los administrativos: Como ya se comentó en el apartado de abasto, todas las funcionales del personal administrativo quedaron descartadas por la falta de tiempo. Me hubiera gustado tener la oportunidad de profundizar en más detalle sobre sus funciones y poder haber creado algo para ellos también, ya que varios de ellos se mostraron receptivos a participar en el proyecto.

- Añadir vacunas y pruebas complementarias: Las vacunas y las pruebas complementarias es una parte importante para el personal sanitario de los centros de atención primaria. En un principio se miró de intentar añadirlas de alguna forma al sistema, pero era una tarea que requería demasiado tiempo, puesto que eran sistemas muy complejos. Por este motivo se decidió descartar estas dos tareas para centrarse en realizar mejor el resto de funcionalidades.
- Mejorar la interfaz de usuario: A pesar de estar muy contenta con el resultado, creo que se pueden añadir o modificar elementos para que la interfaz fuera más intuitiva y más cómoda de usar.
- Mejorar prescripciones: Finalmente, los stakeholders me comentaron que estaría muy bien poder ver las prescripciones antiguas de un paciente y volver a recetarlas fácilmente. Es una funcionalidad que añadí en el Taiga para implementar, pero el atraso del primer y segundo sprint lo hicieron inviable.

14. Bibliografía

- [1] Revealed: Countries With The Best Health Care Systems, 2021. (2021, April 27). CEOWORLD magazine. Retrieved September 22, 2022, from <https://ceoworld.biz/2021/04/27/revealed-countries-with-the-best-health-care-systems-2021/>
- [2] Cervera, X., López, C., Fita, J., & Verdaguer, J. (2021, December 25). Colapso de la Atención Primaria: 48 pacientes por médico y 5,7 días de espera. La Vanguardia. Retrieved September 22, 2022, from <https://www.lavanguardia.com/vida/20211225/7952109/atencion-primaria-colapso-pacientes-medico-dias-espera.html>
- [3] Ming Tai-Seale, Cliff W. Olson, Jinnan Li, Albert S. Chan, Criss Morikawa, Meg Durbin, Wei Wang, and Harold S. Luft. (2017, April). Electronic Health Record Logs Indicate That Physicians Split Time Evenly Between Seeing Patients And Desktop Medicine. HealthAffairs. Retrieved September 22, 2022, from <https://www.healthaffairs.org/doi/full/10.1377/hlthaff.2016.0811>
- [4] FERNÁNDEZ, I. (2022, May 9). El sistema informático único, en el horizonte de Sanidad. Redacción Médica. Retrieved October 24, 2022, from https://www.redaccionmedica.com/secciones/ministerio-sanidad/el-sistema-informatico-unico-para-todo-el-sns-en-el-horizonte-de-sanidad-2025&utm_source=publicacionmedica&utm_medium=email&utm_campaign=boletin
- [5] Rozanski, N., & Woods, E. (2011). Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives. Addison-Wesley.
- [6] (n.d.). Nimbo: Software para clínicas y consultorios médicos. Retrieved September 22, 2022, from <https://www.nimbo-x.com/>
- [7] (n.d.). MediCloud.me | Software gratuito para clínicas médicas. Retrieved September 22, 2022, from <https://medicloud.me/>
- [8] (n.d.). MEDILINK | El Software Médico para gestionar tu clínica. Retrieved September 22, 2022, from <https://www.softwaremedilink.com/>
- [9] Volere Plantilla de Especificación de Requisitos Edición 11—Febrero 2006 por James & Suzanne Robertson rectores del Atl. (n.d.). Volere Requirements. Retrieved September 22, 2022, from https://www.volere.org/wp-content/uploads/2018/12/template_es.pdf
- [10] What Is Scrum Methodology? & Scrum Project Management. (n.d.). Digite. Retrieved September 22, 2022, from <https://www.digite.com/agile/scrum-methodology/>
- [11] Tu herramienta de gestión de proyectos ágil y opensource: Kanban & Scrum. (n.d.). Taiga. Retrieved September 22, 2022, from <https://www.taiga.io/es>
- [12] Home. (2018, December 5). YouTube. Retrieved September 22, 2022, from https://todoist.com/home?gspk=c2VtYW50aWNsYWJzNzMxNw&utm_campaign=strategic_partner&utm_content=semanticlabs7317&utm_medium=strategic_partner&utm_source=partnerstack&sid=1-g-Cj0KCQjwj7CZBhDHARIsAPPWv3fzkNrxgm161ZiDdpg17cnS794mePsh9fRhV8KcDZhFazUJ9NGcElaAn
- [13] (n.d.). GitHub: Where the world builds software · GitHub. Retrieved September 22, 2022, from <https://github.com/>
- [14] Git Flow vs. Trunk Based Development | Toptá
- [15] Documentos de Google: editor de documentos online. (n.d.). Google. Retrieved September 22, 2022, from <https://www.google.com/intl/es/docs/about/>
- [16] (n.d.). Google Meet. Retrieved October 15, 2022, from <https://meet.google.com/>

- [17] TheOfficialHalie - Represent J.A ft. Chris Money. (2016, February 19). YouTube. Retrieved October 16, 2022, from <https://www.fib.upc.edu/sites/fib/files/documents/estudis/normativa-tfg-mencio-addicional-gei.pdf>
- [18] Plataforma de almacenamiento personal en la nube y uso compartido de archivos. (n.d.). Google. Retrieved October 16, 2022, from <https://www.google.com/intl/es/drive/>
- [19] (n.d.). Flowchart Maker & Online Diagram Software. Retrieved October 16, 2022, from <https://app.diagrams.net/>
- [20] (n.d.). Visual Studio Code - Code Editing. Redefined. Retrieved October 16, 2022, from <https://code.visualstudio.com/>
- [21] MongoDB Atlas Database | Multi-Cloud Database Service. (n.d.). MongoDB. Retrieved October 16, 2022, from <https://www.mongodb.com/atlas/database>
- [22] (n.d.). Node.js. Retrieved October 16, 2022, from <https://nodejs.org/en/>
- [23] (n.d.). Postman API Platform | Sign Up for Free. Retrieved October 16, 2022, from <https://www.postman.com/>
- [24] (n.d.). Netlify: Develop & deploy the best web experiences in record time. Retrieved October 16, 2022, from <https://www.netlify.com/>
- [25] (n.d.). Búsqueda de empleo en Glassdoor | Te mereces una empresa que te valore. Retrieved October 17, 2022, from <https://www.glassdoor.es/index.htm>
- [26] What is a Use Case? - Definition from Techopedia. (2014, September 23). Techopedia. Retrieved November 11, 2022, from <https://www.techopedia.com/definition/25813/use-case>
- [27] La diferencia entre el modelo de datos conceptual y lógico. (2021, September 17). Talent Garden. Retrieved December 6, 2022, from <https://talentgarden.org/es/data/the-difference-between-conceptual-and-logical-data-model/>
- [28] Felip, R. (2020, March 21). Buenas historias de usuario: título, descripción y ejemplos. Apiumhub. Retrieved December 21, 2022, from https://apiumhub.com/es/tech-blog-barcelona/como-escribir-buenas-historias-de-usuario/#Como_escribir_buenas_historias_de_usuario_-_MODELO_INVEST
- [29] Web Application Architecture: The Latest Guide 2022. (n.d.). ClickIT. Retrieved November 12, 2022, from <https://www.clickittech.com/devops/web-application-architecture/>
- [30] Everything you need to know about MVC architecture. (2020, May 29). Towards Data Science. Retrieved November 12, 2022, from <https://towardsdatascience.com/everything-you-need-to-know-about-mvc-architecture-3c827930b4c1>
- [31] Hernandez, R. D. (2021, April 19). The Model View Controller Pattern – MVC Architecture and Frameworks Explained. freeCodeCamp. Retrieved November 12, 2022, from <https://www.freecodecamp.org/news/the-model-view-controller-pattern-mvc-architecture-and-frameworks-explained/>
- [32] Module Pattern | JavaScript Patterns. (n.d.). JavaScript Patterns Workshop. Retrieved December 6, 2022, from <https://javascriptpatterns.vercel.app/patterns/design-patterns/module-pattern>
- [33] A Complete Guide on How to Build Middleware For Node.js. (n.d.). Turing. Retrieved December 6, 2022, from <https://www.turing.com/kb/building-middleware-for-node-js>
- [34] React component design patterns for 2022. (2022, March 2). LogRocket Blog. Retrieved December 6, 2022, from <https://blog.logrocket.com/react-component-design-patterns-2022/>

- [35] Santos's, A. (2022, March 31). What is MERN stack and how does it work? Imaginary Cloud. Retrieved November 12, 2022, from <https://www.imaginarycloud.com/blog/what-is-mern-stack-and-how-does-it-work/>
- [36] Herbert, D. (2022, June 27). What is React.js? (Uses, Examples, & More). HubSpot Blog. Retrieved November 12, 2022, from <https://blog.hubspot.com/website/react-js>
- [37] *tfgBackend*. (2023, January 6). github. Retrieved January 10, 2023, from <https://github.com/nuriia99/tfgBackend>
- [38] *tfgFrontend*. (2023, January 6). github. Retrieved January 10, 2023, from <https://github.com/nuriia99/tfgFrontend>
- [39] (n.d.). RGPD - Reglamento General de Protección de datos. Retrieved January 10, 2023, from <https://rgpd.es/>
- [40] (n.d.). Railway.app. Retrieved January 10, 2023, from <https://railway.app/>
- [41] (n.d.). Atenea. Retrieved October 24, 2022, from https://atenea.upc.edu/pluginfile.php/5058165/mod_folder/content/0/M%C3%B2dul%202.6%20-%20El%20informe%20de%20sostenibilidad%202018.pdf?forcedownload=1

Anexo 1. Índice de imágenes

Imagen 1. Diagrama de flujo de gitflow [14]	22
Imagen 2. Diagrama de casos de uso para gestionar la cuenta del usuario	36
Imagen 3. Diagrama de casos de uso para gestionar los pacientes	37
Imagen 4. Diagrama de casos de uso para gestionar la agenda del usuario	38
Imagen 5. Diagrama de casos de uso para gestionar la recaptación de información	38
Imagen 6. Esquema conceptual de datos	45
Imagen 7. Arquitectura de 3 capas [29]	58
Imagen 8. Patrón de diseño MVC [31]	59
Imagen 9. Arquitectura del sistema	60
Imagen 10. Mapa navegacional	63
Imagen 11. Diseño pantalla de iniciar sesión	64
Imagen 12. Diseño pantalla principal	64
Imagen 13. Diseño pantalla de configuración	65
Imagen 14. Diseño pantalla del curso clínico	65
Imagen 15. Diseño pantalla de la inteligencia activa	66
Imagen 16. Diseño pantalla de listado de prescripciones	66
Imagen 17. Diseño pantalla de listado de documentos	67
Imagen 18. Diseño pantalla de listado de visitas	67
Imagen 19. Diseño pantalla de crear prescripción	67
Imagen 20. Diseño pantalla de crear visita	68
Imagen 21. Diseño pantalla de crear entrada	69
Imagen 22. Diseño pantalla de visualizar objetivos	69
Imagen 23. Diseño pantalla de visualizar pacientes asignados a un objetivo	70
Imagen 24. Diseño pantalla de generar listado	70
Imagen 25. Diseño pantalla de listado generado	71
Imagen 26. Diagrama de clases	81
Imagen 27. Diagrama de secuencia de crear una entrada	83
Imagen 28. Arquitectura de MERN Stack [36]	90
Imagen 29. Estructura back-end del proyecto	92
Imagen 30. Estructura front-end del proyecto	93
Imagen 31. Estados de React	93
Imagen 32. Creación del patientContext	94
Imagen 33. Custom hook de la api call	95
Imagen 34. Función de verifyUser	95
Imagen 35. Matriz de sostenibilidad [41]	112

Anexo 2. Índice de tablas

Tabla 1. Comparativa de todas las soluciones existentes	15
Tabla 2. Estimación de horas y recursos de cada tarea	28
Tabla 3. Diagrama de gantt	29
Tabla 4. Sueldo/hora de cada rol	31
Tabla 5. Coste de los recursos humanos	32
Tabla 6. Coste directo de los recursos materiales	33
Tabla 7. Coste para los imprevistos	34
Tabla 8. Coste para la contingencia	34
Tabla 9. Presupuesto final	34
Tabla 10. Comparativa de horas estimadas y horas reales	109
Tabla 11. Comparativa de horas estimadas y horas reales	111

Anexo 3. Casos de uso

Caso de uso	#2 Log Out	Actor principal	Usuario
Precondiciones	El usuario ha de estar logueado en el sistema.		
Disparador	El usuario quiere cerrar sesión.		
Escenario principal de uso			
<div>1. El usuario indica al sistema que quiere cerrar sesión.</div> <div>2. El sistema cierra sesión y le indica al usuario que se ha cerrado la sesión con éxito.</div>			

Caso de uso	#4 Cambiar especialidad	Actor principal	Usuario
Precondiciones	El usuario ha de estar logueado en el sistema.		
Disparador	El usuario quiere cambiar su especialidad.		
Escenario principal de uso			
<div>1. El usuario indica la especialidad a la que desea cambiar.</div> <div>2. El sistema indica al usuario que la especialidad ha sido cambiada.</div>			

Caso de uso	#5 Cambiar lenguaje	Actor principal	Usuario
Precondiciones	El usuario ha de estar logueado en el sistema.		
Disparador	El usuario quiere cambiar el lenguaje.		
Escenario principal de uso			
<div>1. El usuario indica el lenguaje al que desea cambiar.</div> <div>2. El sistema indica al usuario que el lenguaje ha sido cambiado.</div>			

Caso de uso	#7 Ver lista de documentos	Actor principal	Usuario
Precondiciones	El usuario ha de estar logueado en el sistema.		
Disparador	El usuario quiere visualizar la lista de documentos de un paciente.		
Escenario principal de uso			
<div>1. El usuario indica al sistema que quiere acceder a la lista de documentos de un paciente.</div> <div>2. El sistema muestra al usuario la lista de documentos de ese paciente.</div>			

Caso de uso	#8 Ver documento	Actor principal	Usuario
Precondiciones	El usuario ha de estar logueado en el sistema.		
Disparador	El usuario quiere visualizar un documento de un paciente.		
Escenario principal de uso			
<div>1. El usuario indica al sistema que quiere ver el documento de un paciente.</div> <div>2. El sistema muestra al usuario el documento.</div>			

Caso de uso	#9 Borrar documento	Actor principal	Usuario
Precondiciones	El usuario ha de estar logueado en el sistema.		
Disparador	El usuario quiere borrar un documento de un paciente.		
Escenario principal de uso			
<div>1. El usuario indica al sistema que quiere borrar un documento.</div> <div>2. El sistema pregunta al usuario si está seguro de querer borrar el documento.</div> <div>3. El usuario confirma que quiere borrar el documento.</div> <div>4. El sistema comunica al usuario que el documento ha sido borrado.</div>			
Extensiones			
<div>3a. El usuario decide no borrar el documento.</div> <div>3a1. El usuario indica que no quiere borrar el documento.</div> <div>3a2. El sistema muestra al usuario la lista con los documentos del paciente y finaliza el caso de uso.</div>			

Caso de uso	#12 Ver inteligencia activa	Actor principal	Usuario
Precondiciones	El usuario ha de estar logueado en el sistema.		
Disparador	El usuario quiere visualizar la inteligencia activa de un paciente.		
Escenario principal de uso			
<div>1. El usuario indica al sistema que quiere visualizar la inteligencia activa de un paciente.</div> <div>2. El sistema muestra al usuario la inteligencia activa del paciente.</div>			

Caso de uso	#17 Crear informe	Actor principal	Usuario
Precondiciones	El usuario ha de estar logueado en el sistema.		
Disparador	El usuario quiere crear un informe al paciente.		
Escenario principal de uso			
<div>1. El usuario indica que quiere añadir un nuevo informe a un paciente.</div> <div>2. El sistema solicita al usuario la información para crear un nuevo informe.</div> <div>3. El usuario proporciona la información al sistema e indica que desea guardar los cambios.</div> <div>4. El sistema vuelve a mostrar al usuario la agenda con los pacientes.</div>			
Extensiones			
<div>3a. El usuario quiere añadir una prescripción.<div>3a1. El usuario indica que quiere añadir una prescripción al sistema.</div><div>3a2. Empieza el caso de uso de crear prescripción.</div><div>3a3. Se continúa el caso de uso desde el punto 3.</div></div> <div>3b. El usuario no ha seleccionado un diagnóstico.<div>3b1. El sistema indica al usuario que ha de seleccionar un diagnóstico.</div><div>3b2. El usuario selecciona un diagnóstico.</div><div>3b3. Se continúa el caso de uso desde el punto 3.</div></div>			

Caso de uso	#18 Ver listado de prescripciones	Actor principal	Usuario
Precondiciones	El usuario ha de estar logueado en el sistema.		
Disparador	El usuario quiere visualizar el listado de prescripciones de un paciente.		
Escenario principal de uso			
<div>1. El usuario indica al sistema que quiere ver el listado de prescripciones de un paciente.</div> <div>2. El sistema muestra al usuario el listado de prescripciones.</div>			

Caso de uso	#19 Ver prescripción	Actor principal	Usuario
Precondiciones	El usuario ha de estar logueado en el sistema.		
Disparador	El usuario quiere visualizar la prescripción de un paciente.		
Escenario principal de uso			
<div>1. El usuario indica al sistema que quiere ver la prescripción de un paciente.</div> <div>2. El sistema muestra al usuario la prescripción.</div>			

Caso de uso	#20 Crear prescripción	Actor principal	Usuario
Precondiciones	El usuario ha de estar logueado en el sistema.		
Disparador	El usuario quiere crear una prescripción.		
Escenario principal de uso			
<div>1. El usuario indica que quiere añadir una prescripción a un paciente.</div> <div>2. El sistema solicita los datos al usuario para crear una nueva prescripción.</div> <div>3. El usuario proporciona los datos e indica que desea crear la prescripción.</div> <div>4. El sistema muestra el listado de las prescripciones con la nueva prescripción.</div>			
Extensiones			
<div>3a. El usuario no ha seleccionado un medicamento.</div> <div>3a1. El sistema indica al usuario que ha de seleccionar un medicamento.</div> <div>3a2. El usuario selecciona un medicamento.</div> <div>3a3. Se continúa el caso de uso desde el punto 3.</div>			

Caso de uso	#21 Modificar prescripción	Actor principal	Usuario
Precondiciones	El usuario ha de estar logueado en el sistema.		
Disparador	El usuario quiere modificar una prescripción.		
Escenario principal de uso			
<div><div>1.</div><div>El usuario indica que desea realizar cambios en la prescripción.</div></div> <div><div>2.</div><div>El sistema solicita al usuario los datos de la nueva prescripción.</div></div> <div><div>3.</div><div>El usuario realiza los cambios en la prescripción e indica al sistema que desea guardar los cambios.</div></div> <div><div>4.</div><div>El sistema indica al usuario que los cambios se han guardado correctamente.</div></div>			
Extensiones			
<div><div>1a.</div><div>El usuario no ha seleccionado un medicamento.</div></div> <div><div>1a1.</div><div>El sistema indica al usuario que ha de seleccionar un medicamento.</div></div> <div><div>1a2.</div><div>El usuario selecciona un medicamento.</div></div> <div><div>1a3.</div><div>Se continúa el caso de uso desde el punto 3.</div></div>			

Caso de uso	#22 Borrar prescripción	Actor principal	Usuario
Precondiciones	El usuario ha de estar logueado en el sistema.		
Disparador	El usuario quiere borrar una prescripción.		
Escenario principal de uso			
<div>1. El usuario indica al sistema que quiere borrar una prescripción.</div> <div>2. El sistema pregunta al usuario si está seguro de querer borrar la prescripción.</div> <div>3. El usuario confirma que quiere borrar la prescripción.</div> <div>4. El sistema muestra al usuario la lista de prescripciones sin la prescripción.</div>			
Extensiones			
<div>3a. El usuario decide no borrar la prescripción.</div> <div>3a1. El usuario indica que no quiere borrar la prescripción.</div> <div>3a2. El sistema muestra al usuario la lista con las prescripciones del paciente y finaliza el caso de uso.</div>			

Caso de uso	#23 Ver cita previa	Actor principal	Usuario
Precondiciones	El usuario ha de estar logueado en el sistema.		
Disparador	El usuario quiere ver una cita de un paciente.		
Escenario principal de uso			
<div>1. El usuario indica al sistema que quiere ver una cita de un paciente.</div> <div>2. El sistema muestra al usuario la cita.</div>			

Caso de uso	#24 Ver listado de citas	Actor principal	Usuario
Precondiciones	El usuario ha de estar logueado en el sistema.		
Disparador	El usuario quiere visualizar el listado de citas de un paciente.		
Escenario principal de uso			
<div>1. El usuario indica al sistema que quiere ver el listado de citas de un paciente.</div> <div>2. El sistema muestra al usuario el listado de citas que el usuario desea ver.</div>			

Caso de uso	#27 Borrar cita	Actor principal	Usuario
Precondiciones	El usuario ha de estar logueado en el sistema.		
Disparador	El usuario quiere borrar una cita a un paciente.		
Escenario principal de uso			
<div>1. El usuario indica al sistema que desea eliminar una cita.</div> <div>2. El sistema pregunta al usuario si está seguro de querer eliminar la cita.</div> <div>3. El usuario indica que sí quiere eliminar la cita.</div> <div>4. El sistema muestra el listado de entradas sin la cita.</div>			
Extensiones			
<div>2a. El usuario decide no borrar la cita.</div> <div>2a1. El usuario indica que no quiere borrar la cita.</div> <div>2a2. El sistema muestra al usuario el formulario de la cita y finaliza el caso de uso.</div>			

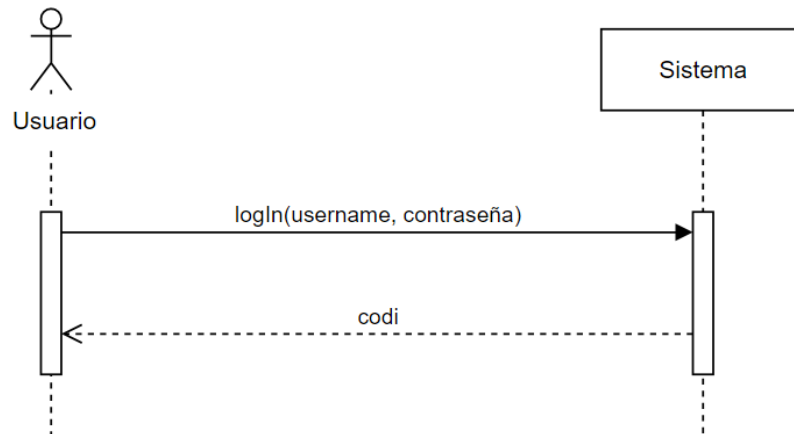
Caso de uso	#28 Ver agenda	Actor principal	Usuario
Precondiciones	El usuario ha de estar logueado en el sistema.		
Disparador	El usuario quiere ver una agenda.		
Escenario principal de uso			
<div>1. El usuario indica al sistema que quiere ver una agenda.</div> <div>2. El sistema muestra al usuario las agendas que hay disponibles.</div> <div>3. El usuario indica la agenda que desea ver.</div> <div>4. El sistema muestra la agenda al usuario.</div>			

Caso de uso	#29 Ver listado de objetivos	Actor principal	Usuario
Precondiciones	El usuario ha de estar logueado en el sistema.		
Disparador	El usuario quiere ver los objetivos que tiene asignados.		
Escenario principal de uso			
<div>1. El usuario indica al sistema que quiere acceder al apartado de objetivos.</div> <div>2. El sistema muestra al usuario el listado con todos sus objetivos.</div>			

Caso de uso	#30 Ver objetivo	Actor principal	Usuario
Precondiciones	El usuario ha de estar logueado en el sistema.		
Disparador	El usuario quiere ver información relacionada con el objetivo.		
Escenario principal de uso			
<div>1. El usuario indica al sistema qué información quiere ver sobre el objetivo.</div> <div>2. El sistema muestra al usuario la información sobre el objetivo.</div>			

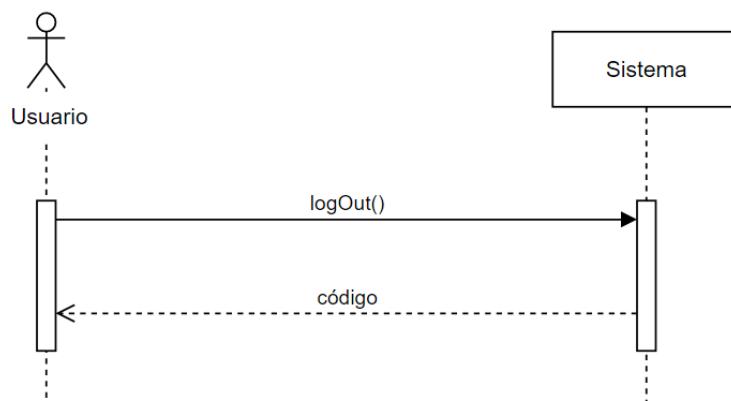
Anexo 4. Modelos de comportamiento

Log In



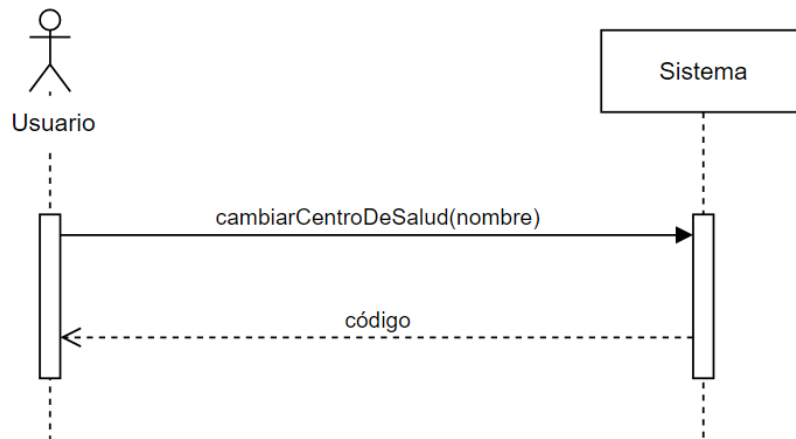
Caso de uso	login(username: String, contraseña: String)
Precondiciones	
Postcondiciones	Se crea un token para validar al trabajador.
Body	Se devuelve la información del trabajador del username o un código de error.

Log Out



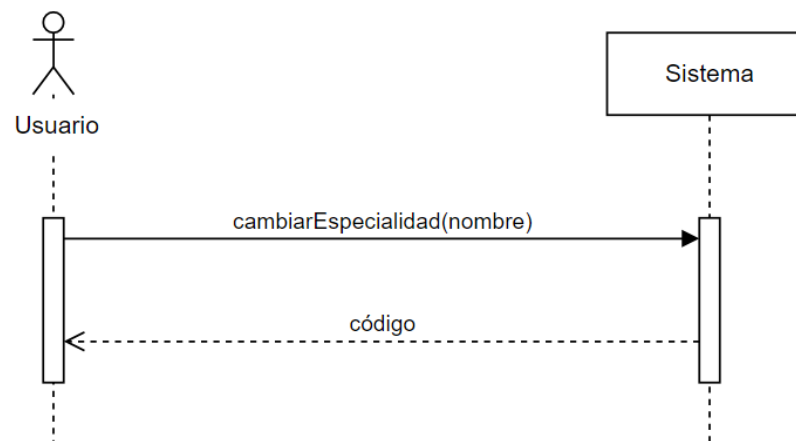
Caso de uso	logout()
Precondiciones	
Postcondiciones	Se borra el token asignado al trabajador.
Body	Se devuelve el código de aceptación.

Cambiar de centro de salud



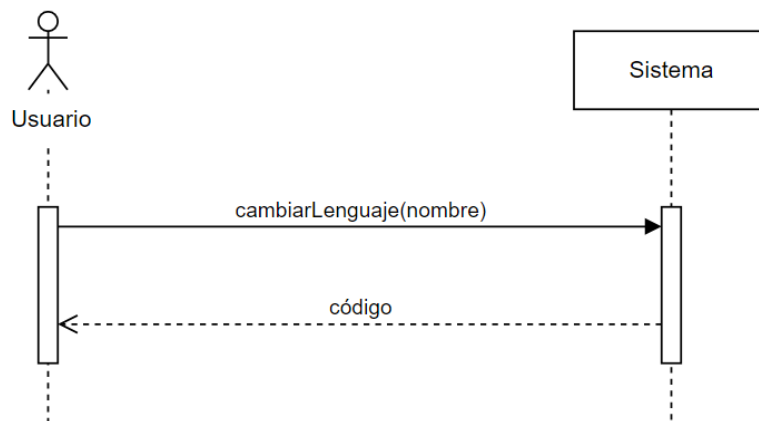
Caso de uso	cambiarCentroDeSalud(nombre: String)
Precondiciones	nombre existe en el sistema.
Postcondiciones	
Body	Se devuelve el código de aceptación.

Cambiar de especialidad



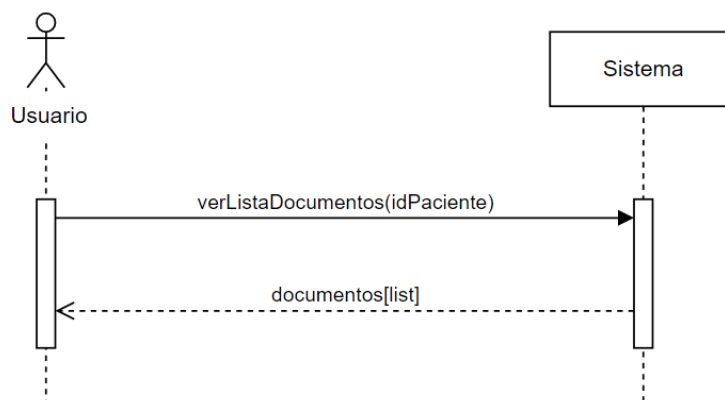
Caso de uso	cambiarEspecialidad(nombre: String)
Precondiciones	
Postcondiciones	
Body	Se devuelve el código de aceptación.

Cambiar de lenguaje



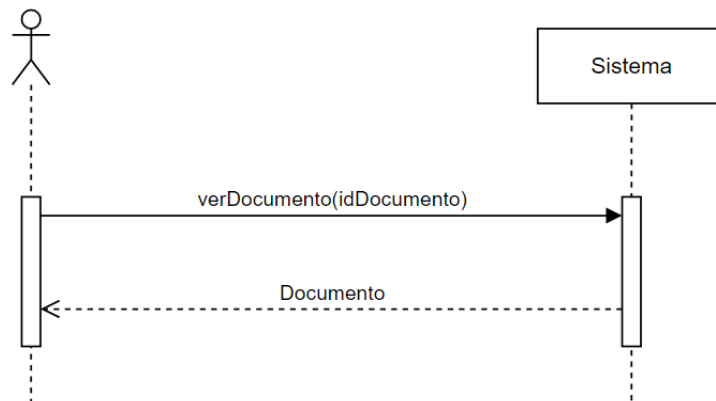
Caso de uso	cambiarLenguaje(nombre: String)
Precondiciones	
Postcondiciones	
Body	Se devuelve el código de aceptación.

Ver la lista de documentos de un paciente



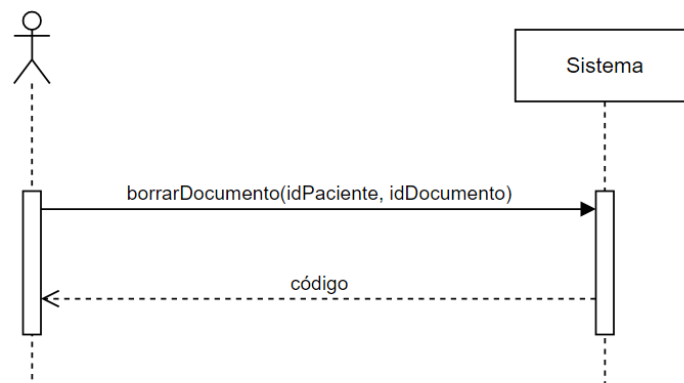
Caso de uso	verListaDocumentos(idPaciente: String)
Precondiciones	idPaciente existe en el sistema.
Postcondiciones	
Body	Se devuelve una lista con todos los documentos del paciente o un código de error.

Ver un documento de un paciente



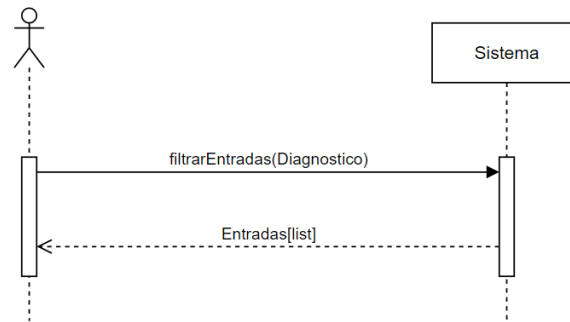
Caso de uso	verDocumento(idDocumento: String)
Precondiciones	idDocumento existe en el sistema.
Postcondiciones	
Body	Se devuelve el documento o un código de error.

Borrar un documento de un paciente



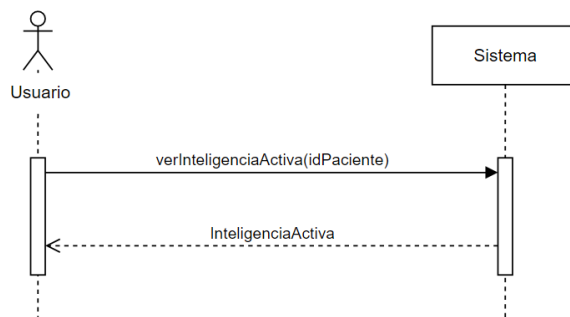
Caso de uso	borrarDocumento(idPaciente: String, idDocumento: String)
Precondiciones	idPaciente existe en el sistema. idDocumento existe en el sistema.
Postcondiciones	Se ha borrado el documento de la lista de documentos del paciente.
Body	Se devuelve el código de aceptación si se ha borrado el documento correctamente.

Filtrar la lista de entradas de un paciente



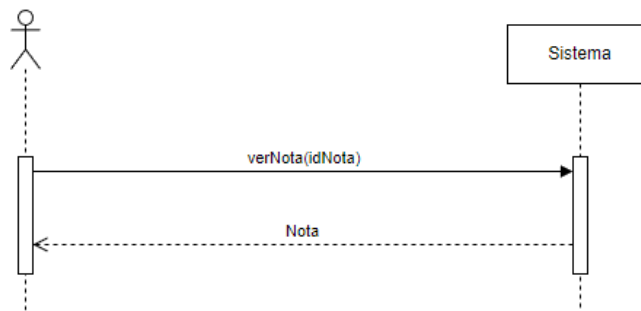
Caso de uso	filtrarEntradas(diagnóstico: String)
Precondiciones	diagnóstico existe en el sistema.
Postcondiciones	
Body	Se devuelve una lista con todas las entradas o un código de error.

Ver la inteligencia activa de un paciente



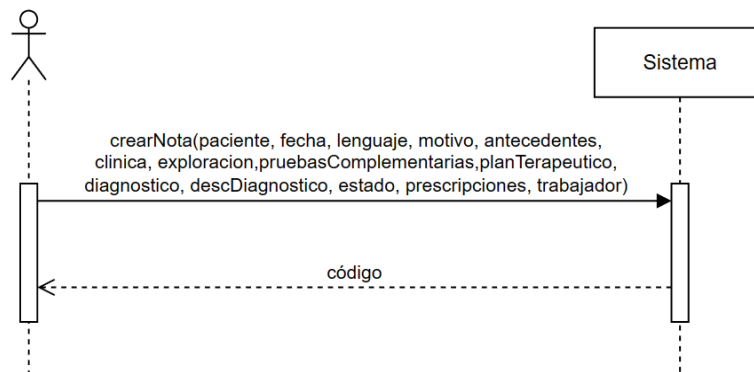
Caso de uso	verInteligenciaActiva(idPaciente: String)
Precondiciones	idPaciente existe en el sistema.
Postcondiciones	
Body	Se devuelve la inteligencia activa del paciente o un código de error.

Ver una nota de un paciente



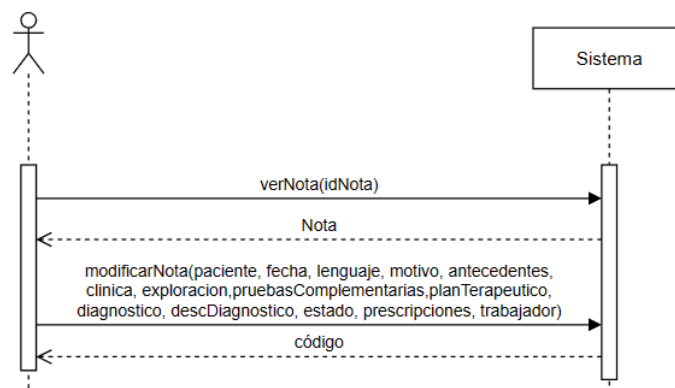
Caso de uso	verNota(idNota: String)
Precondiciones	idNota existe en el sistema.
Postcondiciones	
Body	Se devuelve la nota o un código de error.

Crear una nota a un paciente



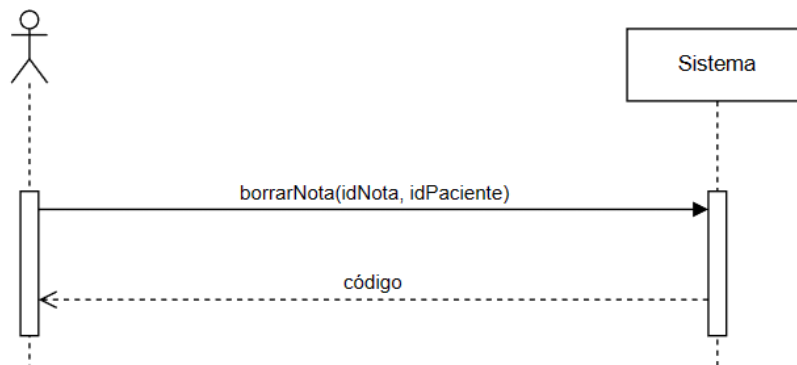
Caso de uso	crearNota(paciente: String, fecha: Date, lenguaje: String, motivo: String, antecedentes: String, clínica: String, exploración: String, pruebasComplementarias: String, planTerapeutico: String, diagnóstico: String, descDiagnóstico: String, estado: String, prescripciones: List<String>, trabajador: String)
Precondiciones	paciente existe en el sistema. trabajador existe en el sistema. diagnóstico existe en el sistema. las prescripciones existen en el sistema.
Postcondiciones	Se crea una nota con los datos introducidos, se añade la nota a la lista de notas de la entrada del paciente y se actualizan los objetivos del trabajador si estos se ven afectados por la nueva nota. Si no existe una entrada el día de hoy se crea una nueva entrada con los datos introducidos y se añade a la lista de entradas del paciente.
Body	Se devuelve el código de aceptación si se ha creado correctamente la nota.

Modificar una nota de un paciente



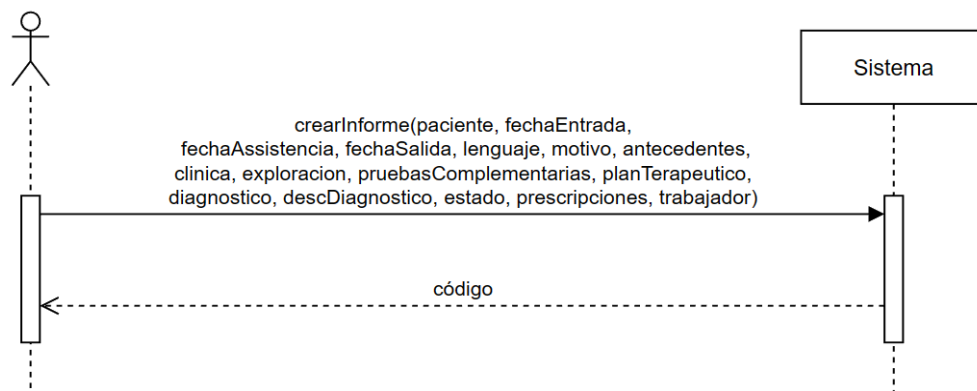
Caso de uso	modificarNota(paciente: String, fecha: Date, lenguaje: String, motivo: String, antecedentes: String, clínica: String, exploración: String, pruebasComplementarias: String, planTerapeutico: String, diagnóstico: String, descDiagnóstico: String, estado: String, prescripciones: List<String>, trabajador: String)
Precondiciones	paciente existe en el sistema. trabajador existe en el sistema. diagnóstico existe en el sistema. las prescripciones existen en el sistema.
Postcondiciones	Se modifica la nota con los datos introducidos y se actualizan los objetivos del trabajador si estos se ven afectados por la nueva nota.
Body	Se devuelve la nota o un código de error.

Borrar una nota de un paciente



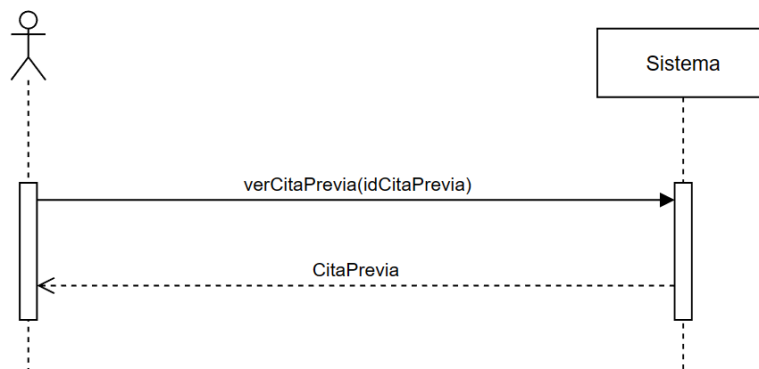
Caso de uso	borrarNota(idNota: String, idPaciente: String)
Precondiciones	idNota existe en el sistema. idPaciente existe en el sistema.
Postcondiciones	Se borra la nota de la lista de notas de la entrada del paciente y en caso de que la lista quede vacía se borra la entrada.
Body	Se devuelve el código de aceptación si se ha borrado la nota correctamente.

Crear un informe



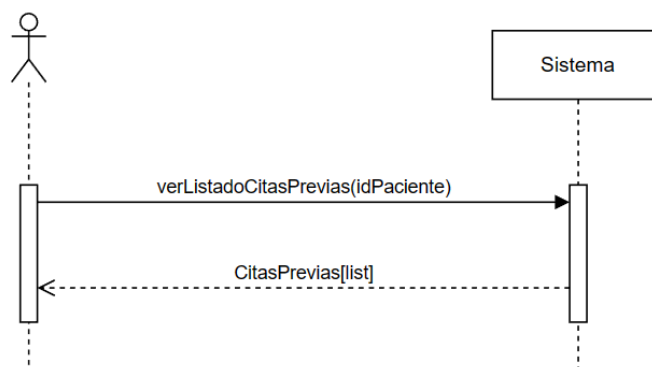
Caso de uso	crearInforme(paciente: String, fechaEntrada: Date, fechaAsistencia: Date, fechaSalida: Date, lenguaje: String, motivo: String, antecedentes: String, clinica: String, exploracion: String, pruebasComplementarias: String, planTerapeutico: String, diagnostico: String, descDiagnostico: String, estado: String, prescripciones: List<String>, trabajador: String)
Precondiciones	paciente existe en el sistema. trabajador existe en el sistema. diagnostico existe en el sistema. las prescripciones existen en el sistema.
Postcondiciones	Se crea un informe con los datos introducidos y se añade el informe a la lista de documentos del paciente.
Body	Se devuelve el código de aceptación si se ha creado correctamente el informe.

Ver una nota de un paciente



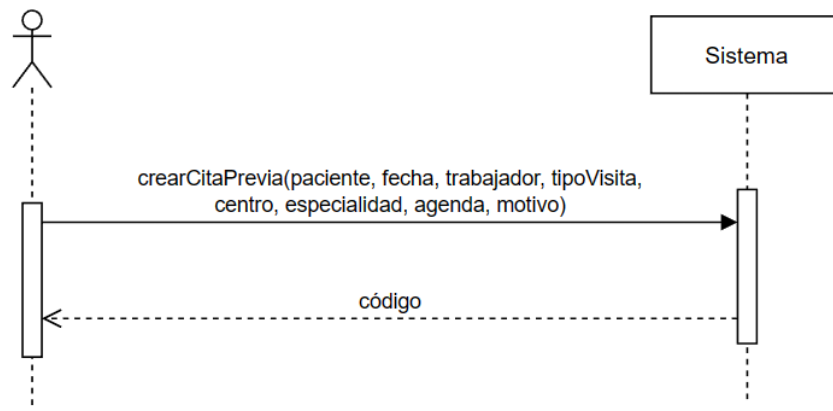
Caso de uso	verCitaPrevia(idCitaPrevia: String)
Precondiciones	CitaPrevia existe en el sistema.
Postcondiciones	
Body	Se devuelve la cita previa o un código de error.

Ver listado de citas previas de un paciente



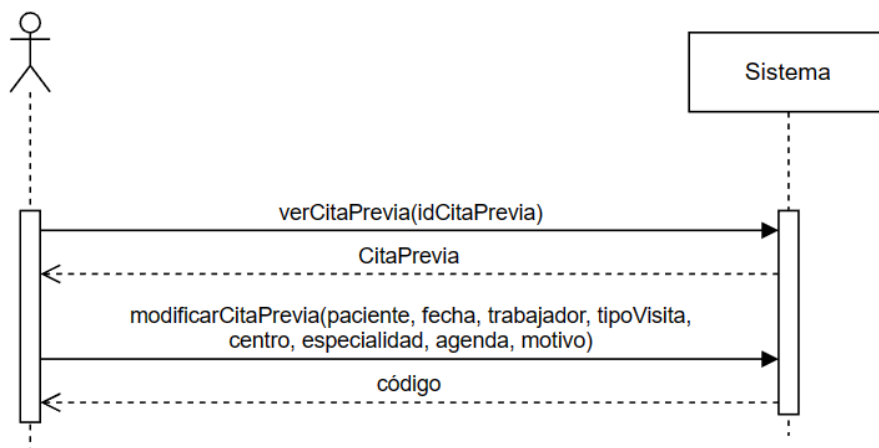
Caso de uso	verListadoCitasPrevias(idPaciente : String)
Precondiciones	idPaciente existe en el sistema.
Postcondiciones	
Body	Se devuelve la lista de citas previas del paciente o un código de error.

Crear una cita previa



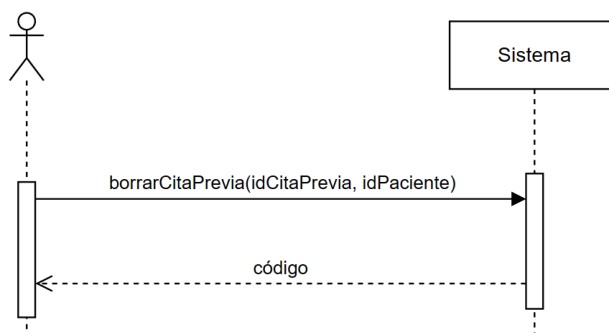
Caso de uso	crearCitaPrevias(paciente: String, fecha: Date, trabajador: String, tipoVisita: String, centro: String, especialidad: String, agenda: String, motivo: String)
Precondiciones	paciente existe en el sistema. trabajador existe en el sistema. centro existe en el sistema. agenda existe en el sistema.
Postcondiciones	Se crea una cita previa con los datos introducidos y se añade la cita a la lista de citas previas del paciente.
Body	Se devuelve el código de aceptación si se ha creado correctamente la cita previa.

Modificar una nota de un paciente



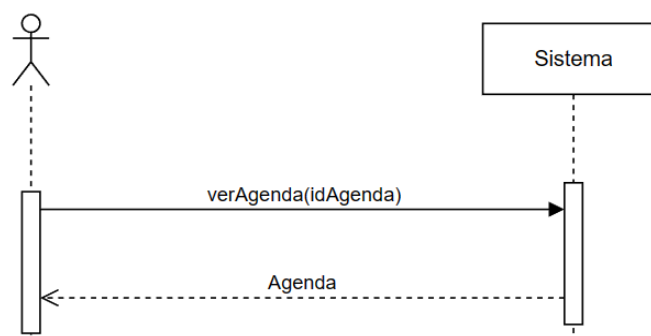
Caso de uso	modificarCitaPrevia(paciente: String, fecha: Date, trabajador: String, tipoVisita: String, centro: String, especialidad: String, agenda: String, motivo: String)
Precondiciones	paciente existe en el sistema. trabajador existe en el sistema. centro existe en el sistema. agenda existe en el sistema.
Postcondiciones	Se modifica la cita con los datos introducidos.
Body	Se devuelve el código de aceptación si se ha modificado correctamente la cita previa.

Borrar una cita previa de un paciente



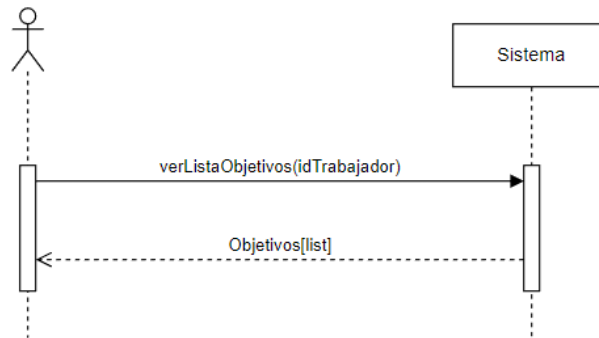
Caso de uso	borrarCitaPrevia(idCitaPrevia: String, idPaciente: String)
Precondiciones	idCitaPrevia existe en el sistema. idPaciente existe en el sistema.
Postcondiciones	Se borra la cita previa de la lista de citas del paciente.
Body	Se devuelve el código de aceptación si se ha borrado la cita correctamente.

Ver la agenda de un trabajador



Caso de uso	verObjetivo(idObjetivo: String)
Precondiciones	idAgenda existe en el sistema.
Postcondiciones	
Body	Se devuelve la agenda o un código de error.

Ver la lista de objetivos de un trabajador



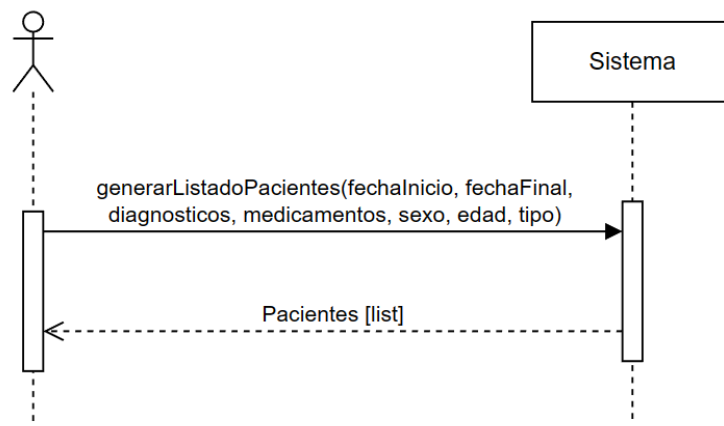
Caso de uso	verListaObjetivos(idTrabajador : String)
Precondiciones	idTrabajador existe en el sistema.
Postcondiciones	
Body	Se devuelve la lista de objetivos del trabajador o un código de error.

Ver un objetivo de un trabajador



Caso de uso	verObjetivo(idObjetivo: String)
Precondiciones	idObjetivo existe en el sistema.
Postcondiciones	
Body	Se devuelve el objetivo o un código de error.

Generar listado de pacientes



Caso de uso	<code>generarListadoPacientes(fechaInicio: Date, fechaFinal: Date, diagnosticos: List<String>, medicamentos: List<String>, sexo: String, edad: Number, tipo: String)</code>
Precondiciones	los diagnósticos existen en el sistema. los medicamentos existen en el sistema.
Postcondiciones	
Body	Se devuelve la lista de pacientes o un código de error.

Anexo 5. Historias de usuario

Historia de usuario	#1 Log In
Descripción	Como usuario, quiero loguearme en mi cuenta para poder utilizar la aplicación.
Criterios de aceptación	Al introducir los datos, los datos deben validarse y en caso de ser correctos se ha de redireccionar al usuario a la pantalla principal. En la pantalla principal, debe aparecer automáticamente la información del centro y la especialidad que está asignada al turno actual del usuario que ha abierto sesión. Si el usuario introduce un username que no existe, se debe notificar al usuario de que el usuario no existe. Si el usuario introduce una contraseña incorrecta, se debe notificar al usuario de que la contraseña es incorrecta.

Historia de usuario	#2 Log Out
Descripción	Como usuario, quiero poder cerrar mi sesión para que nadie más pueda usar mi cuenta.
Criterios de aceptación	El usuario debe poder cerrar su cuenta desde el apartado de configuración. Una vez el usuario cierra la cuenta, se debe redirigir al usuario a la pantalla de login. En caso de que suceda algún error, se debe notificar al usuario que no se ha cerrado la cuenta y que lo vuelva a intentar.

Historia de usuario	#3 Cambiar centro de salud
Descripción	Como usuario, quiero poder cambiar de centro de salud para poder ver la información relacionada a ese centro.
Criterios de aceptación	En el apartado de opciones, deben aparecer todos los centros en los que trabaja el usuario. El usuario debe poder cambiar el centro en el que está trabajando. En caso de que suceda algún error, se debe notificar al usuario de que no se ha podido cambiar de centro de salud.

Historia de usuario	#4 Cambiar especialidad
Descripción	Como usuario, quiero poder cambiar de especialidad para indicar al sistema el rol que estoy ejerciendo.
Criterios de aceptación	En el apartado de opciones, deben aparecer todas las especialidades del usuario. El usuario debe poder cambiar la especialidad en la que está trabajando. En caso de que suceda algún error, se debe notificar al usuario de que no se ha podido cambiar de especialidad y que lo vuelva a intentar.

Historia de usuario	#5 Cambiar lenguaje
Descripción	Como usuario, quiero poder cambiar de lenguaje para poder trabajar en el idioma que desee.
Criterios de aceptación	En el apartado de opciones, deben aparecer todos los lenguajes disponibles. El usuario debe poder cambiar el lenguaje en el que está trabajando. En caso de error, se debe de notificar al usuario que no se ha podido cambiar el lenguaje y se debe permitir al usuario intentarlo de nuevo.

Historia de usuario	#7 Ver lista de documentos
Descripción	Como usuario, quiero poder ver la lista de los documentos de un paciente para poder visualizar sus documentos.
Criterios de aceptación	En el curso clínico, debe aparecer una lista con el nombre de los documentos. El usuario debe poder ver una lista detallada de los documentos. Los documentos deben estar ordenados por fecha descendente.

Historia de usuario	#8 Ver documento
Descripción	Como usuario, quiero poder acceder a un documento para poder leer el contenido del mismo.
Criterios de aceptación	Se debe poder abrir un documento en una pestaña aparte desde el listado de documentos y el curso clínico. Se debe permitir descargar el documento e imprimirlo.

Historia de usuario	#9 Borrar documento
Descripción	Como usuario, quiero poder borrar un documento en caso de que lo considere oportuno.
Criterios de aceptación	Desde el listado de documentos, debe aparecer un botón que permita borrar el documento. Al borrar el documento, debe aparecer un mensaje de confirmación para asegurarnos de que el usuario quiere borrar el documento. Una vez borrado el documento, debe aparecer la lista sin el documento.

Historia de usuario	#11 Filtrar entrada
Descripción	Como usuario, quiero poder filtrar las entradas según diagnósticos tanto activos como inactivos, para poder acceder con mayor facilidad a las entradas que me interesan.
Criterios de aceptación	En el curso clínico de un paciente, deben aparecer todos los diagnósticos activos e inactivos de un paciente. Cada diagnóstico debe ir acompañado de un color que muestre su severidad. Se debe poder filtrar las notas del curso clínico según diagnóstico.

Historia de usuario	#12 Ver inteligencia activa
Descripción	Como usuario, quiero poder acceder a la inteligencia activa de un paciente para visualizar la información general de un paciente.
Criterios de aceptación	<p>En el curso clínico, deben aparecer los parámetros de IMC, el resumen de hábitos tóxicos y el resumen de alergias.</p> <p>Al darle al botón de enseñar más datos, se debe acceder a la pantalla donde aparecen todos los datos relacionados con la inteligencia activa.</p> <p>En la tabla de la inteligencia activa, debe aparecer todas las fechas donde hubo una modificación y el valor cambiado.</p> <p>En la tabla de inteligencia activa, debe aparecer una columna con la fecha actual que muestre los datos actuales de cada propiedad activa.</p>

Historia de usuario	#13 Ver nota
Descripción	Como usuario, quiero poder visualizar una entrada para poder visualizar la información detallada de la visita de un paciente.
Criterios de aceptación	<p>Se debe poder visualizar la información de una nota.</p> <p>No debe aparecer el sistema de recomendación de diagnóstico.</p>

Historia de usuario	#14 Crear nota
Descripción	Como usuario, quiero crear una entrada para poder registrar toda la información relacionada con una consulta a un paciente y actualizar su curso clínico.
Criterios de aceptación	<p>Se debe poder crear una nota nueva.</p> <p>Se debe poder mirar el contenido de las notas previas sin perder el progreso de la nueva nota.</p> <p>Se debe poder borrar el contenido de la nota.</p> <p>Debe aparecer un listado de diagnósticos recomendados al haber acabado el apartado de la clínica.</p> <p>Se debe poder seleccionar el diagnóstico de la nota con los diagnósticos actuales o antiguos del paciente.</p> <p>Debes poder añadir una prescripción mientras creas la nota.</p> <p>Al guardar la nota, se muestra al usuario la lista de todas las notas del paciente.</p> <p>Si el usuario crea una entrada sin haber seleccionado ningún diagnóstico, debe aparecer un mensaje de error comunicando al usuario que debe seleccionar un diagnóstico.</p>

Historia de usuario	#15 Modificar nota
Descripción	Como usuario quiero modificar una entrada para poder corregir algún error o añadir información.
Criterios de aceptación	<p>Se debe poder modificar una nota.</p> <p>Al guardar una nota, se debe mostrar al usuario la lista de las notas del paciente con los cambios efectuados.</p> <p>Si el usuario guarda la entrada sin haber seleccionado ningún diagnóstico, debe aparecer un mensaje de error comunicando al usuario que debe seleccionar un diagnóstico.</p> <p>No debe aparecer el apartado de recomendación de diagnóstico.</p>

Historia de usuario	#16 Borrar nota
Descripción	Como usuario, quiero borrar una nota en caso de haber cometido un error y el paciente no padezca de ese diagnóstico.
Criterios de aceptación	Se debe poder eliminar una nota. Al borrar una nota, se debe mostrar al usuario la lista de las notas del paciente sin la nota borrada. Si la nota eliminada era la última nota de una entrada, se debe borrar también la entrada. Al borrar una nota, debe aparecer un mensaje de confirmación para asegurarnos de que el usuario quiere borrar la nota.

Historia de usuario	#17 Crear informe
Descripción	Como usuario, quiero crear un informe para poder registrar a un paciente toda la información relacionada con una visita de urgencia.
Criterios de aceptación	Se debe poder crear un informe nuevo a un paciente. Se debe poder abrir el curso clínico del paciente en una pestaña aparte mientras creas el informe. Debe aparecer un listado de diagnósticos recomendados al haber acabado el apartado de la clínica. Se debe poder seleccionar el diagnóstico de la nota con los diagnósticos actuales o antiguos del paciente. Debes poder añadir una prescripción mientras creas el informe. Al crear el informe, este debe aparecer en la lista de documentos del paciente. Si el usuario crea el informe sin haber seleccionado ningún diagnóstico, debe aparecer un mensaje de error comunicando al usuario que debe seleccionar un diagnóstico. Si el usuario crea el informe sin haber seleccionado quien ha atendido al paciente, debe aparecer un mensaje de error comunicando al usuario que debe seleccionar quien ha atendido la visita.

Historia de usuario	#23 Ver cita previa
Descripción	Como usuario, quiero poder ver la cita previa de un usuario para poder acceder a una información más detallada.
Criterios de aceptación	Si haces clic en una cita, debe aparecer toda la información de toda la cita. Se debe permitir al usuario volver a la pestaña anterior.

Historia de usuario	#24 Ver listado de citas
Descripción	Como trabajador, quiero poder acceder al listado de citas futuras de un paciente para saber qué citas tiene ya programadas.
Criterios de aceptación	Debe aparecer el listado de todas las vistas de un paciente con la información más relevante. Las citas deben estar ordenadas según hora y fecha en sentido ascendente.

Historia de usuario	#25 Crear cita
Descripción	Como trabajador, quiero poder crear una cita previa para acordar una hora de visita entre el paciente y el sanitario.
Criterios de aceptación	<p>Se debe poder crear una nueva desde el apartado de listado de citas y desde la agenda.</p> <p>Debes de poder seccionar la agenda que quieras que esté asignada al centro que tienes seleccionado.</p> <p>Si la hora de la cita solapa con otra, debe aparecer un mensaje de error y no se debe crear la cita.</p> <p>Si la fecha de la visita es menor a la actual, debe aparecer un mensaje de error y no se debe crear la cita.</p> <p>Se debe poder seleccionar el tipo de visita.</p> <p>Al crear la visita se actualiza las visitas de la agenda.</p>

Historia de usuario	#26 Modificar cita
Descripción	Como trabajador, quiero poder modificar una cita previa en caso de que haya algún error, se quiera asignar la visita a otro trabajador o se quiera modificar la fecha de la visita.
Criterios de aceptación	<p>Deben aparecer la cita con toda la información rellena.</p> <p>Se debe poder modificar cualquier apartado de la cita previa.</p> <p>Al guardar debe aparecer el listado de citas con la información modificada.</p> <p>Si la hora de la cita solapa con otra, debe aparecer un mensaje de error y no se debe modificar la cita.</p> <p>Si la fecha de la visita es menor a la actual, debe aparecer un mensaje de error y no se debe modificar la cita.</p>

Historia de usuario	#27 Borrar cita
Descripción	Como trabajador, quiero poder borrar una cita previa en caso de que ya no sea necesaria.
Criterios de aceptación	<p>Desde el listado de citas, debe aparecer un botón que permita borrar la cita.</p> <p>Al clicar al botón aparece un mensaje para asegurarnos de que el usuario quiere borrar la cita.</p> <p>Al darle clic a borrar una cita, debe aparecer un mensaje de advertencia asegurándose de que el trabajador quiere borrar la cita.</p>

Historia de usuario	#28 Ver agenda
Descripción	Como trabajador, quiero poder acceder a una agenda para poder ver las visitas que tengo asignadas en un día.
Criterios de aceptación	<p>Se debe abrir por defecto la agenda del usuario con las visitas del día actual.</p> <p>Se debe poder seleccionar el día y ver las citas asignadas para ese día.</p> <p>Se debe filtrar las agendas en las que tienes alguna visita de las que no.</p> <p>Debes poder ver la agenda que quieras.</p> <p>Debes poder ver el curso clínico o crear una consulta a un paciente.</p>

Historia de usuario	#29 Ver listado de objetivos
Descripción	Como usuario, quiero poder ver el listado de todos los objetivos que tengo asignados para poder obtener la información necesaria para completarlos.
Criterios de aceptación	<p>Se debe poder ver los objetivos asignados al usuario divididos por su sección.</p> <p>Para cada uno de los objetivos, debe aparecer su progreso representado tanto gráficamente como numéricamente.</p> <p>Si el resultado del objetivo es menor a la mitad del objetivo, el resultado debe aparecer en rojo.</p> <p>Si el resultado del objetivo es menor al objetivo y mayor a la mitad del objetivo, el resultado debe aparecer en amarillo.</p> <p>Si el resultado del objetivo es mayor al objetivo, el resultado debe aparecer en verde.</p>

Historia de usuario	#30 Ver objetivo
Descripción	Como usuario, quiero poder acceder a un objetivo para saber de qué se trata y poder ver la lista de pacientes asignados a ese objetivo.
Criterios de aceptación	<p>Se debe poder ver la información del objetivo y el listado de pacientes completados y por completar que están asignados a ese objetivo.</p> <p>Debe aparecer una tabla resumen del progreso del objetivo.</p> <p>Se debe poder ver el curso clínico de los pacientes.</p>

Historia de usuario	#31 Generar listado de pacientes
Descripción	Como usuario, quiero poder generar un listado de pacientes para analizar y mejorar mi rendimiento en el trabajo.
Criterios de aceptación	<p>Se debe poder generar un formulario según diagnósticos, estado del diagnóstico, prescripciones, estado de las prescripciones, rango de edad, sexo y periodo de tiempo.</p> <p>Al añadir un diagnóstico o medicamento, debe aparecer un nuevo buscador para añadir más diagnósticos o medicamentos.</p> <p>Se debe poder borrar un diagnóstico o un medicamento.</p> <p>Se debe poder filtrar si mostrar todos los pacientes, todos los pacientes del usuario, o solo los pacientes de un centro.</p> <p>Si haces clic en una fila, se debe mostrar el curso clínico de un paciente.</p> <p>Para cada fila se debe mostrar el sexo de la persona y el nombre completo de la persona.</p> <p>Se debe poder exportar el listado en formato pdf o excel.</p> <p>Al inicio del listado debe aparecer una tabla resumen con los resultados.</p> <p>En el listado de pacientes, se debe poder escoger qué columnas de la inteligencia activa añadir.</p> <p>En el listado de pacientes, se debe poder ordenar la tabla según el valor de la columna que se desee.</p>