# Análisis automatizado de analíticas para pacientes crónicos.

Grado en Ingeniería Informática
Especialización en Ingeniería del Software

---

Sergio Vázquez Montes de Oca
Director: Xavier Franch Gutiérrez
16 de junio de 2021 – Cuatrimestre de primavera

## Resumen

Actualmente, un gran sector de la población sufre de patologías crónicas, estas personas tienen que realizarse controles de forma periódica para verificar que dicha patología está siendo tratada correctamente. Este proceso no solo implica la realización de un análisis de sangre, sino que también implica la visita posterior con su médico de cabecera para realizar posibles ajustes en su medicación actual. Este procedimiento en muchos de los casos es rutinario y se puede automatizar. Este trabajo propone un sistema que procesa las analíticas introducidas directamente desde los laboratorios y genera los cambios en la medicación del paciente de forma automática. Gracias a esto, se ahorrará una gran cantidad de tiempo tanto por la parte médica como por la parte del paciente.

## Resum

Actualment, un gran sector de la població pateix malalties cròniques, aquestes persones s'han de realitzar controls de forma periòdica per tal de verificar si la malaltia està sent tractada correctament. Aquest procés no només implica la realització d'un anàlisis de sang, sinó que també implica la visita posterior amb el seu metge per tal de realitzar possibles ajusts a la seva medicació actual. Aquest procediment en molts dels casos és rutinari i es pot automatitzar. Aquest treball proposa un sistema que processa les analítiques introduïdes directament des dels laboratoris i genera els canvis en la medicació del pacient de forma automàtica, gràcies a això, s'estalviarà una gran quantitat de temps tant per la part mèdica com per la part del pacient.

## Abstract

Nowadays, a large sector of the population suffers from chronic diseases, these people have to undergo regular check-ups in order to verify if that pathology is being treated correctly. This process not only involves the blood test but also a subsequent visit to their GP to make any adjustments to their current medication. This process in many cases is a routine and can be automated. This thesis proposes a system that processes the analyses entered directly from the laboratories and generates the changes in the patient's medication automatically, thus saving a great deal of time on both the medical side and the patient's side.

# Agradecimientos

Este trabajo es la culminación de toda la carrera, es por eso por lo que me gustaría dar las gracias a todas las personas que me han aportado alguna cosa a lo largo de estos años.

En especial, a mis amigos, los cuales me han ayudado a encauzar mi camino y sin ellos no habría llegado a donde estoy ahora mismo. A mi pareja, la cual es un pilar fundamental de mi vida, y, además, ha supuesto parte de la inspiración de este trabajo.

También me gustaría agradecer a mi familia todo el esfuerzo que han tenido que hacer para que finalmente pudiera finalizar la carrera, no siempre es fácil seguir confiando cuando las cosas están tan cuesta arriba.

Infinitas gracias a las doctoras Natividad Pacheco y Àngels Escorsell, las cuales sacaron tiempo de sus apretadas agendas para ayudarme con toda la parte médica de este trabajo.

Finalmente, agradecer a Xavier Franch por aconsejarme, guiarme y ayudarme durante todos estos meses, y a Joan Subirats, por sus consejos durante el módulo de GEP.

# Glosario

**Análisis de sangre:** Es un análisis de laboratorio realizado en una muestra de sangre.

**API**: Interfaz de programación de aplicaciones.

**CAP**: Centro de atención primaria.

**Enfermedad crónica**: Afectación que por lo general dura 3 meses o más, y es posible que empeore con el tiempo.

**Hipercolesterolemia**: Aumento de los niveles óptimos del colesterol en sangre.

**Hipotiroidismo**: Trastorno de la glándula tiroides la cual no produce la cantidad suficiente de ciertas hormonas cruciales.

**Laboratorio clínico y biomédico**: Laboratorio cuya competencia general es la de realizar estudios analíticos de muestras biológicas.

**Parámetros médicos**: Distintos valores que se tienen en cuenta a nivel médico para determinar la gravedad de una enfermedad.

**Stakeholder:** Es cualquier individuo u organización que de alguna manera se ve afectado por la implantación del sistema.

**Teleasistencia**: Servicio dirigido a personas mayores que viven solas o a personas con diversidad funcional que permite pedir ayuda en caso de urgencia.

**Wearables**: Prenda de ropa o accesorio al cual se le ha incorporado un microprocesador.

# Glossari

**Anàlisis de sang:** És un anàlisis de laboratori realitzat en una mostra de sang.

**API**: Interfície de programació d'aplicacions.

**CAP**: Centre d'atenció primària.

**Malaltia crònica**: Afectació que per norma general dura 3 mesos o més, i que és possible que empitjori amb el temps.

**Hipercolesterolèmia**: Augment dels nivells òptims de colesterol en sang.

**Hipotiroïdisme**: Trastorn de la glàndula tiroide la qual no produeix la quantitat suficient de certes hormones crucials.

**Laboratori clínic** i **biomèdic**: Laboratori que té com a competència general la de realitzar estudis analítics sobre mostres biològiques.

**Paràmetres mèdics**: Diferents valors que es tenen en compte en l'àmbit mèdic per a determinar la gravetat d'una malaltia.

**Stakeholder:** És qualsevol individu o organització que, d'alguna forma, es veu afectat per la implantació del sistema.

**Teleassistència**: Servei dirigit a persones grans que viuen soles o a persones amb diversitat funcional que permet sol·licitar ajuda en cas d'emergència.

**Wearables**: Peça de roba o accessori al qual se li ha incorporat un microprocessador.

# Glossary

API: Application Programming Interface.

Blood tests: A laboratory test performed on a blood sample.

CAP: Primary Care Centre.

Chronic disease: A condition that usually lasts 3 months or more, and may worsen over time.

Clinical and biomedical laboratory: Laboratory whose general competence is to perform analytical studies on biological samples.

Hypercholesterolemia: Increase of the optimal levels of cholesterol in blood.

Hypothyroidism: A disorder of the thyroid gland which does not produce enough of certain crucial hormones.

Medical parameters: Different values that are taken into account at the medical level to determine the severity of a disease.

Stakeholder: It is any individual or organization that is in some way affected by the implementation of the system.

Telecare: Service aimed at elderly people living alone or people with functional diversity that allows them to ask for help in case of emergency.

Wearables: Garment or accessory to which a microprocessor is incorporated.

# Índice de Contenido

# Índice de Ilustraciones

# Índice de tablas

# 1. Contexto y justificación

## 1.1 Introducción y finalidad del proyecto

Esta memoria ha sido escrita para completar el Trabajo de Fin de Grado del Grado en Ingeniería Informática de la Universidad Politécnica de Cataluña. En particular, todo el trabajo está enfocado en los principios de la Ingeniería del Software, especialidad que he estado cursando en los últimos dos años del grado.

Considero que el trabajo de fin de grado debe tener una finalidad más allá de poner en práctica los conceptos aplicados durante la carrera. Es por eso por lo que me centré en pensar una idea que pudiera aportar valor real a la sociedad actual.

Si pensamos en nuestro sistema sanitario, sin ir más lejos, podemos encontrar muchos aspectos que pueden ser mejorados gracias a la tecnología. En mi caso, fue un día hablando con mi pareja, la cual se tiene que realizar analíticas periódicas para controlar su hipotiroidismo, cuando pensé: ¿Y por qué tienes que ir al doctor para que te suba o te baje la medicación?

Con una simple notificación en su teléfono, se evitaría el ir a su centro de salud para que el doctor le modifique la medicación, y, además, el doctor dispondría de ese tiempo ahorrado en la consulta al paciente crónico para otras visitas que requieran una verdadera presencialidad del paciente.

Si además nos paramos a pensar en la situación que estamos viviendo, evitar ir a un centro de salud puede resultar vital, pues además de ahorrar tiempo a los sanitarios, que en estos tiempos lo necesitan más que nunca, podemos evitar el contagio evitando estar expuestos en una zona de riesgo elevado como puede ser un centro de atención primaria.

Por lo tanto, y para sintetizar todo lo dicho, la finalidad principal del proyecto es la de ahorrar tiempo, algo muy preciado cuando de salud estamos hablando.

## 1.2 Los enfermos crónicos

Según datos ofrecidos en el Informe Cronos [1], cerca del 45% de la población española mayor de 16 años sufre de al menos un proceso crónico. Estos procesos son cada vez más comunes, y muchas de las personas que los sufren terminan falleciendo. En particular, son responsables de más de 300.000 muertes al año, lo cual representa casi un 75% del total de muertes anuales en España.

| Tensión alta | Infarto de miocardio | Otras enfermedades del corazón | Artrosis, artritis o reumatismo | Dolor espalda crónico (cervical) | Dolor espalda crónico (lumbar) | Alergia crónica (asma alérgica excluida) | Asma | Bronquitis crónica, enfisema, EPOC |
|---|---|---|---|---|---|---|---|---|
| 7.132,3 | 300,7 | 1.766,1 | 7.061,7 | 6.146,5 | 7.201,7 | 4.155,9 | 1.578,5 | 1.453 |

| Diabetes | Colesterol alto | Problemas de piel | Estreñimiento | Cirrosis, disfunción hepática | Depresión crónica | Ansiedad crónica | Otros problemas mentales |
|---|---|---|---|---|---|---|---|
| 2.690,7 | 6.308,5 | 1.677,7 | 1.464,5 | 281 | 2.282,1 | 2.600,1 | 626,3 |

| Hemorroides | Tumores malignos | Osteoporosis | Problemas de tiroides | Embolia, infarto cerebral, hemorragia cerebral | Migraña o dolor de cabeza frecuente | Úlcera estómago o duodeno |
|---|---|---|---|---|---|---|
| 1.603,2 | 491,1 | 1.602,1 | 1.720 | 252,5 | 3.233,5 | 911 |

*Ilustración 1: Enfermedades crónicas o de larga evolución padecidas en los últimos 12 meses y diagnosticadas por un médico en población de 15 y más años [1]. Fuente: informe Cronos.*

Estas personas tienen que vivir con su enfermedad, y eso significa tener que realizarse unos controles que en la mayoría de los casos son rutinarios y que sirven para controlar si la cantidad de medicación que están tomando es la correcta, o en caso contrario, hay que cambiarla o modificarla.

Por lo tanto, existe una gran parte de la población que tiene que ir periódicamente a su centro de salud más cercano a realizarse un análisis de sangre, que posteriormente será estudiado por su médico de cabecera. El doctor será el encargado de determinar si los valores de la analítica son los indicados para tratar de paliar los efectos de su enfermedad.

## 1.3 Proceso que siguen las analíticas actualmente

Actualmente el proceso que se lleva a cabo para realizar un análisis de sangre es el siguiente:

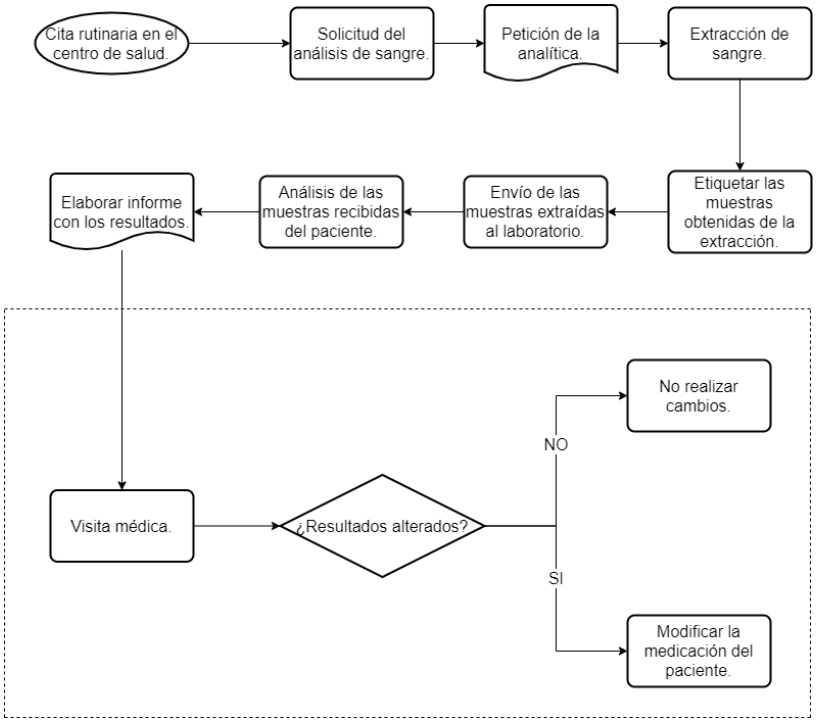



*Ilustración 2: Proceso actual que siguen los análisis de sangre [2]. Fuente: elaboración propia.*

En la ilustración 2 se puede ver el proceso llevado a cabo para realizar un control rutinario a un paciente crónico. La zona marcada con líneas discontinuas indica la parte de todo este proceso que el sistema software que se propone pretende automatizar.

Como podemos observar en el diagrama, cuando el laboratorio termina de realizar todas las pruebas pertinentes a las muestras, genera un informe con los resultados obtenidos tras la realización de una prueba en particular. Una vez se tiene el informe, se envía al centro médico para que citen al paciente a una visita médica en la cual se interpretarán los resultados de la prueba. En caso de que los resultados salgan alterados, se tendrá que modificar la medicación que el paciente ya está tomando, y, por otro lado, en caso de que todo esté en orden, no se tendrá que hacer nada.

## 1.4 El problema

El problema que se trata de resolver es el del dispendio del tiempo. Como ya todos sabemos, el sistema sanitario se encuentra en estos momentos colapsado, en gran parte por la pandemia causada por la COVID-19. Pero, si hacemos memoria, recordaremos que hace un par de años (cuando no había pandemia) el sistema sanitario no estaba tampoco en su mejor momento.

Es por eso, por lo que concertar una cita para la revisión periódica de un paciente crónico es una tarea complicada [3]. Primero se tiene que encontrar un hueco en la agenda del doctor, que, además, tiene que coincidir con la del paciente. Después, antes de que llegue el paciente, el doctor tiene que tomarse un tiempo en analizar con calma los resultados del análisis para extraer conclusiones y modificar la receta del paciente en caso de que sea necesario. Adicionalmente, el paciente tiene que desplazarse hasta su centro de salud, cosa que, en algunos casos, puede suponer un gran problema, como por ejemplo si se trata de una persona mayor o de alguien con movilidad reducida.

Sufrir una enfermedad crónica ya es en sí mismo algo negativo, y el hecho de que las personas que la sufren tengan que visitar cada cierto tiempo su centro de salud para su control rutinario, es algo que incrementa aún más el factor negativo asociado a sufrir dicha enfermedad. Y es que, tener una visita médica, no implica llegar y entrar directamente a consulta, como ya todos sabemos, es un proceso que normalmente suele implicar estar una hora como mínimo en el centro de salud. Y este no es el único problema, el grado de dependencia que tienen estos pacientes con su centro de salud es altísimo, siempre tienen que estar pendientes de cuándo tienen su próximo control debido a que las citas médicas hoy en día, no son muy flexibles.

En conclusión, el proceso de los análisis de sangre actual es muy ineficiente, la gran mayoría de veces que una analítica sale alterada no sería necesario ir al médico para que este le ponga remedio, muchos de estos casos son rutinarios y se podrían automatizar. Sobre todo, para el caso de los pacientes crónicos, a los cuales se les realizan muchas analíticas clínicas rutinarias simplemente para ajustar la medicación.

# 2. Viabilidad del sistema software

Para hablar de la viabilidad del sistema software se deben tener muchas cosas en cuenta, como, por ejemplo, los riesgos potenciales que puede tener, o si realmente será útil y utilizable por los *stakeholders* objetivo del sistema. Además, se tiene que valorar si realmente cubre una necesidad dentro del mercado.

## 2.1 Análisis de mercado

Si bien es verdad que existen diferentes máquinas [4] que analizan una muestra de sangre y te dan los resultados en poco tiempo indicando los parámetros que han salido alterados, no es algo comparable con el sistema que se plantea, ya que estos dispositivos son máquinas que tu personalmente tienes que comprar y, además, solo te resaltan los parámetros alterados. Estas máquinas están destinadas a los laboratorios clínicos, de hecho, ya son usadas por la mayoría para analizar las muestras que les llegan de los pacientes.

Sin embargo, el sistema que se propone seguiría la metodología tradicional de tener que ir a que alguien del departamento de enfermería te realice la analítica, y lo diferencial residiría en el viaje que realiza la información una vez procesada por estas máquinas.

Es decir, el sistema cubriría todo el viaje que realizan los resultados de los análisis de sangre desde el laboratorio al doctor, y posteriormente al paciente, y, no existe nada documentado en la actualidad que indique que existe algún sistema similar. Igualmente, se ha realizado un análisis de sistemas que tengan similitudes con el que se propone, en concreto en el envío de avisos médicos informativos, o relacionados con la gestión sanitaria.

### 2.1.1 La meva salut

La meva salut [5] es el sistema software que propone el departamento de salud catalán, desde la aplicación podemos realizar lo siguiente:

- Reservar citas en los CAP.
- Realizar consultas telemáticas a profesionales que trabajan en los CAP.
- Consultar las visitas programadas.
- Consultar tus informes, tu medicación, o resultados de analíticas u otras pruebas.

Como podemos ver, es un sistema bastante completo que intenta automatizar todas las tareas de índole administrativo. Gracias a este sistema, ahora los pacientes pueden, por ejemplo, pedir cita previa sin necesidad de llamar al CAP o ir presencialmente.

El sistema que yo propongo compartirá funcionalidades con La meva salut. Por ejemplo, la consulta de la medicación, pero, la gran diferencia reside en la funcionalidad principal de mi sistema, y es el de automatizar el proceso de las visitas posteriores a las analíticas que se realizan los pacientes crónicos.

### 2.1.2 HumanITcare

HumanITcare [6] es una empresa privada que ha construido un sistema de monitorización de pacientes. Gracias a la gran cantidad de *wearables* que existen hoy en día, esta compañía monitorea desde las pulsaciones por minuto del paciente, hasta su oxígeno en sangre.

Si bien considero que su función está alejada de la que proporcionará el sistema software que se propone, tienen algo en común: ahorran tiempo al personal sanitario. Y es que, gracias a HumanITcare, el paciente puede llevar a cabo una monitorización de sus constantes sin necesidad de ir periódicamente a su centro de salud. En consecuencia, tanto el personal sanitario como el paciente ahorran tiempo que podrán dedicar a otras cosas.

### 2.1.3 ehCOS

ehCOS es la plataforma de salud de Everis [7], esta plataforma nació con la idea de facilitar la consulta de las historias clínicas a los doctores, y por lo tanto ahorrarles tiempo para que lo puedan emplear en una mejor atención a sus pacientes. Con el tiempo han ido extendiendo esta plataforma y hoy en día también es capaz de monitorizar pacientes en la UCI y proporcionar teleasistencia.

De nuevo, se trata de un producto muy innovador, y que facilita mucho la vida a los profesionales de la salud, pero se encuentra lejos de asimilarse al producto que se propone.

### 2.1.4 Conclusión

El mundo está cambiando, hoy en día el proceso de digitalización que estamos viviendo avanza a pasos de gigante año tras año. A pesar de eso, el sector médico es quizás el que va más atrasado en este aspecto. Debido a la pandemia que estamos viviendo hoy en día, este proceso de digitalización ha ido creciendo y ahora este sector se encuentra mucho más digitalizado que a principios del año pasado.

Sin embargo, la gran mayoría de aplicaciones y sistemas que existen en la actualidad solo sirven para digitalizar tareas administrativas o de control de constantes. El sistema que se propone en este trabajo trata de digitalizar un proceso médico, y esto es algo que hoy en día apenas se ha visto. En particular, y como ya se ha comentado, no existe nada documentado que refleje la mera existencia de un sistema que se parezca lo más mínimo al se propone. Es por eso por lo que considero que el sistema propuesto supondrá una clara revolución en la digitalización del sistema sanitario. A pesar de esto, el sistema compartirá funcionalidades con La meva salud, y, por lo tanto, considero que lo ideal sería integrar toda la parte de automatización de analíticas en este sistema que ya se encuentra en producción.

# 3. Alcance del proyecto

Para que el proyecto cubra todas las necesidades que tendrán los futuros usuarios de la aplicación, necesitamos definir cuáles son los objetivos y subobjetivos que tiene que cumplir el sistema. Además, tenemos que considerar los riesgos de negocio y de desarrollo que harían que el proyecto no saliera adelante.

Debido a que, como se ha visto, el número de enfermedades crónicas que existen en la actualidad es muy elevado, en el desarrollo de este proyecto se implementará la automatización de dos de ellas, las cuales nos darán una idea del alcance que puede llegar a tener este proyecto. De todas formas, añadir nuevas enfermedades al sistema no será una tarea difícil, pues una de las características que tendrá el sistema es que será muy sencillo el poder añadir la automatización de nuevas enfermedades crónicas.

## 3.1 Objetivos

Principalmente podemos identificar los siguientes objetivos y sus correspondientes subobjetivos:

- Disponer de un sistema software para el intercambio de información entre los miembros del equipo sanitario y sus respectivos pacientes.
    - Disponer de tipos de cuentas distintas para que cada tipo de usuario pueda acceder a una información en concreto.
    - Mostrar y representar la información de forma clara para que sea fácilmente entendible por el paciente.
    - Notificar al paciente cada vez que tenga información nueva para consultar.
- Mejorar la comunicación, haciéndola inmediata, entre el sistema sanitario y los pacientes.
    - Construir un chat que sea lo menos invasivo posible para el personal sanitario y lo más eficaz para el paciente.
- Aumentar el número de visitas que puede realizar un doctor/a.
    - Eliminar visitas innecesarias gracias a la automatización del proceso de los análisis.
- Reducir el número de veces que un paciente crónico tiene que visitar su CAP.
    - Recibir los resultados de los análisis en su dispositivo.
    - Recibir los cambios de medicación en su dispositivo.
    - Consultar sus visitas sin necesidad de llamar o ir al centro de salud.
    - Consultar su medicación al instante sin necesidad de llamar o ir al CAP.
- Organizar y controlar de forma más eficiente la medicación que deben tomar para tratar su enfermedad.
    - Remarcar los cambios en la medicación del paciente.
    - Notificar al paciente si existen cambios en su medicación.

# 4. Metodología

## 4.1 Descripción de la metodología de trabajo

Para el desarrollo de este proyecto, he decidido dividir dicho desarrollo en dos partes, una primera en la cual se define el proyecto, aquí es donde incluimos toda la parte de la introducción, alcance del proyecto, planificación, definición de los requisitos iniciales, identificación de los posibles riesgos y la propuesta de arquitectura inicial junto con el diseño de unos *mockups* representativos de la aplicación.

Durante esta primera fase, se estudiarán qué enfermedades crónicas son las que incluirá esta primera versión de la aplicación. Además, se escogerán las tecnologías adecuadas para el desarrollo de la API y la aplicación.

Tras esta primera parte ya se tendrá claro cuáles serán los requisitos iniciales de la aplicación, y se podrán convertir en historias de usuario con un peso que representará el número de horas aproximadas que habrá que dedicarle. Gracias a esto, se podrá empezar con el desarrollo de la aplicación móvil y la API. Para las cuales se va a utilizar la metodología *Scrum* [8] (*Agile development*). Evidentemente, al tratarse de una sola persona se tendrá que adaptar.

El proceso *Agile* se trata de un proceso incremental, donde tanto el producto final como los requisitos que tiene que cumplir el sistema van evolucionando a medida que se va desarrollando, es por eso por lo que todo el periodo de desarrollo estará dividido en *sprints*. Cada *sprint* será de una cantidad determinada de tiempo, y, antes de comenzarlo, se valorarán las tareas que hay en el *backlog* (lista de tareas del producto que se va a desarrollar), y se decidirá cuáles incluir en el *sprint* que se va a comenzar. Este proceso se llama *sprint planning* [9], y es una de las ceremonias que tiene *Scrum*. Su función principal es, como ya se ha comentado, la de planificar el *sprint* y decidir qué tareas incluir para que al finalizar dicho *sprint* el producto haya evolucionado.

Una vez se haya finalizado el *sprint*, tendrán lugar dos ceremonias más, la primera, será la reunión retrospectiva [10], en la que principalmente se valora lo que se ha hecho bien, y, por ende, se tiene que seguir haciendo, lo que se ha hecho mal, y por lo tanto se tiene que dejar de hacer, y finalmente cosas que se tienen que mejorar, pero ya se está yendo por el buen camino. Al finalizar esta reunión, se extraen conclusiones que se aplicarán para que el siguiente *sprint* vaya mejor que el que se acaba de terminar.

Además de la retrospectiva, también se realizará el *sprint review* [10], esta reunión se hará con el director del proyecto para que pueda verificar todos los avances que se están haciendo en el producto que se está desarrollando. En esta reunión se hablará sobre cómo está avanzando el proyecto y de si están habiendo algunos problemas.

## 4.2 Software

A continuación, se van a detallar dodos los sistemas Software que han sido utilizados durante la elaboración de este proyecto.

Algunos de los sistemas que se proponen a continuación son de pago, pero cuentan con cuentas para estudiantes las cuales son totalmente gratuitas.

### 4.2.1 Github

Para el control de versiones, se utilizará Github [11], en el cual se va haciendo "*commit*" de todos los cambios que se van realizando en el proyecto.

Existirán dos repositorios, uno destinado a la API y otro a la aplicación, ambos tendrán la rama central *development* sobre la cual solo se hace *merge* una vez se está seguro de que los cambios funcionan. Además de la rama central, existirán diversas ramas creadas para desarrollar una funcionalidad en específico como se puede ver en la ilustración 3. Esto nos permite probar cada funcionalidad por separado sin necesidad de incluir el código nuevo en el bloque de código central (que ya funciona perfectamente) y por lo tanto nos va a evitar muchos problemas técnicos en un futuro.

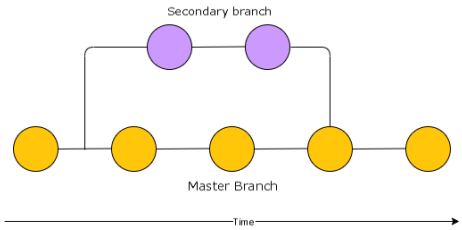



*Ilustración 3: Representación del sistema de ramas de Github [12]. Fuente: BitBucket.*

### 4.2.2 Jira

Jira [13] es la herramienta de gestión de tareas de Atlassian, la cual nos permitirá almacenar todas las historias de usuario del proyecto en un *backlog*, y crear y gestionar los diferentes *sprints* que se irán creando a lo largo del desarrollo del producto.

En Jira se encontrarán las tareas creadas a partir de los requisitos iniciales una vez definidos. Desde este programa se pueden evaluar las tareas (según su dificultad o tamaño) y por lo tanto nos permite realizar prácticamente todos los procesos comunes en *Agile* desde una sola plataforma.

Conforme vaya avanzando el desarrollo, será normal que aparezcan *bugs*, estos se añadirán de forma normal al *backlog* de Jira, pero marcándolos como *bug* tal y como se ve en la siguiente ilustración.


*Ilustración 4: Ejemplo de tarea de tipo Bug en Jira. Fuente: Elaboración propia.*

### 4.2.3 IntelliJ IDEA

IntelliJ IDEA [14] es el programa escogido para el desarrollo de la API. Este programa va a permitir tanto programar, como gestionar todo lo relacionado con el control de versiones. Además, tiene integración con Gradle [15], el cual automatiza todo el proceso de la construcción de código.

IntelliJ IDEA ofrece una manera muy eficiente de programar, cuenta con una gran librería de *plugins* que agilizan enormemente la programación. Es por esto, y por mi experiencia con el programa, que he decidido utilizarlo para el desarrollo del proyecto.

### 4.2.4 Android Studio

Android Studio [16] es el programa que se va a utilizar para el desarrollo de la App de Android. Al igual que IntelliJ, este programa nos permite tanto programar, como gestionar todo lo relacionado con el control de versiones. Y, como el anterior, también cuenta con la integración con Gradle.

Android Studio es hoy en día el programa referencia en cuanto al desarrollo de aplicaciones Android se refiere. Permite desarrollar todas las partes necesarias dentro de la aplicación y cuenta con una interfaz muy intuitiva para el diseño de las interfaces propias de la aplicación que estás desarrollando, facilitando enormemente el desarrollo. Además, está basado en IntelliJ IDEA, lo que hace que compartan interfaz y muchas de las funcionalidades.

### 4.2.5 PostgreSQL y Amazon Web Services

PostgreSQL [17] es un programa de código abierto que nos permite gestionar bases de datos relacionales.

Este software va a ser el utilizado para la gestión de las bases de datos en local, se creará un servidor que aloje las bases de datos que sean necesarias en el transcurso del proyecto. En cuanto se termine el desarrollo en local, se utilizarán los servicios de Amazon para poder alojar las bases de datos en sus servidores con la intención de poner el sistema al alcance de los distintos *stakeholder*.

### 4.2.6 Justinmind

Justinmind es una herramienta de diseño de interfaces, con ella se va a diseñar la totalidad de la aplicación. Gracias a este sistema, se pueden diseñar las diversas pantallas que va a tener nuestra aplicación Android, incluyendo todos sus componentes y las distintas interacciones con los mismos. Justinmind nos permite añadir eventos a los componentes los cuales hacen que dichos componentes se comporten como si se trataran de componentes reales dentro de un dispositivo.

### 4.2.7 Software relacionado con la documentación

Todo el Software mencionado en los apartados anteriores está directamente relacionado con el desarrollo del sistema. Pero, como se ha comentado, no todo el proyecto va a consistir en desarrollar. Es por eso por lo que vamos a necesitar diferentes programas que nos permitan documentar todo lo necesario para completar el trabajo. Estos son los siguientes:

- Microsoft Word: Este es el programa elegido para redactar la memoria.
- Microsoft Power Point: Este es el programa elegido para realizar la presentación de diapositivas el día de la lectura.
- Microsoft Excel: Programa elegido para el cálculo de presupuestos y el control de estos.
- Draw.io: Este es el programa elegido para la realización de todos los diagramas y figuras que se van a ir añadiendo a esta memoria.
- Monday planning: Página web utilizada para realizar el diagrama de Gantt.

Además, se utilizará Google Meet para las comunicaciones directas con el director del TFG y con el personal sanitario.

## 4.3 Análisis de alternativas

En esta sección se exponen las distintas alternativas a diferentes aspectos del proyecto. Además, se exponen los motivos por los cuales se ha llegado a escoger una u otra.

Se ha dividido dicho análisis en cuatro grandes secciones, una para cada gran tipo de decisión que se ha tenido que tomar cuando se estaba planificando el proyecto.

### 4.3.1 Metodología

A lo largo de la especialidad de software, hemos visto distintos tipos de metodologías que podrían funcionar perfectamente para desarrollar el sistema que se propone en esta memoria, de todas ellas, se ha escogido Scrum. A continuación, se van a listar todas las metodologías vistas a lo largo de la especialidad y las razones que han llevado a descartarlas.

#### Metodología en cascada

La metodología en cascada, o clásica, es el enfoque más tradicional en cuanto al desarrollo del software se refiere. Consiste en ir realizando todas las etapas que comprenden el desarrollo de forma secuencial.

El principal motivo que me ha llevado a descartar esta metodología es básicamente lo que la define. Considero que, al tratarse de un trabajo universitario, es muy beneficioso el poder ir desarrollando partes de la documentación a la vez que se va implementando el producto. Tras finalizar cada sprint, se pueden ir refinando los requisitos, o añadiendo nuevas funcionalidades que quizás, a priori, se pasaron por alto. Esto, en la metodología en cascada, no funciona así, pues antes de comenzar el proyecto todos los requisitos están ya fijados y son inamovibles.

Además, otra parte muy importante en el desarrollo de un sistema software son las pruebas, en la metodología en cascada se realizan al finalizar el desarrollo, esto comporta unos riesgos enormes. Por ejemplo, es posible que al finalizar el desarrollo nos encontremos con un error que nos haga modificar gran parte del código, y, que este cambio, acabe afectando a otras partes del sistema. Sin embargo, si se va probando el sistema a medida que se va desarrollando, nos evitamos sufrir este tipo de contratiempos.

Finalmente, otro de los factores claves a la hora de descartar esta metodología fue que los *stakeholders* no necesitan involucrarse mucho cuando se practica esta metodología. Esto es algo muy contradictorio con la idea principal del proyecto, pues vamos a necesitar en todo momento que estas personas, que al final son el público objetivo, nos den su opinión y nos ayuden a refinar el sistema lo máximo posible.

### Kanban

Kanban [18], al igual que Scrum, es otra metodología ágil, esta se basa en el desarrollo y en la entrega de forma continua, es decir, desarrollando pocas tareas de forma rápida y simultánea. Esta metodología está muy caracterizada por la tenencia de un tablero, en el cual se van moviendo las tareas entre distintas columnas, clásicamente hecho con *post-its*.

Al igual que se mencionaba en el apartado anterior, el principal motivo por el cual se descartó esta metodología fue su principal característica, el hecho de tener que realizar entregas continuas, y el de tener un flujo de continuo de desarrollo, es algo que no es provechoso para este tipo de trabajo. Es preferible realizar *sprints* de aproximadamente dos semanas que permitan recibir *feedback* por parte del *Product Owner* (el director del proyecto en este caso).

Otro de los motivos por los cuales se descartó Kanban fue el hecho de que las tareas están sujetas a cambios en cualquier momento. En el caso de este proyecto, se ha buscado un equilibrio entre la metodología en cascada y Kanban, y es que, en Scrum, durante los *sprints* no se pueden realizar cambios, pero si a lo largo de la implementación.

Sin embargo, y una de las cosas positivas que tiene Kanban respecto a Scrum en el marco de este proyecto es la inexistencia de roles. Al tratarse de un proyecto académico, solo hay una persona encargada de todos los procesos, en Scrum existen un gran número de roles y a excepción del *Product Owner*, todos los demás los realiza la misma persona, cosa que carece de sentido.

### Conclusiones

Las dos metodologías mencionadas en las dos secciones anteriores, y Scrum, han sido las tres que más hemos estudiado a lo largo de la especialidad de Software, si bien existen muchas otras, no me parece relevante mencionarlas aquí ya que no se han estudiado en profundidad.

Como conclusión, me gustaría remarcar que la decisión no fue fácil, la metodología en cascada se descartó rápidamente por los motivos comentados, pero el hecho de que Kanban carezca de roles fue algo que trajo serias dudas a la hora de escoger metodología. De todas formas, considero que la elección de Scrum ha sido la correcta y así se ha justificado a lo largo de esta sección.

### 4.3.2 Software

En cuanto al Software escogido para el desarrollo del trabajo, existen muchas alternativas que se pudieron haber escogido, sin embargo, se decidió elegir los mencionados en el informe del GEP por diversos motivos. Si bien existen softwares que se escogieron simplemente por la familiaridad que tenía el autor de la memoria con dicho programa, existen otros que fueron puestos por encima de otros debido a su potencial.

#### GitHub

Hoy en día existen muchas herramientas para el control de versiones, entre ellas podemos destacar Bitbucket, GitLab o AWS CodeCommit. Si bien se podría haber escogido cualquiera de las anteriores, se escogió GitHub por la experiencia del autor con dicha herramienta.

Las diferencias entre los distintos sistemas de control de versiones que se han mencionado son muchas, de todas formas, ninguna me parece lo suficientemente relevante en el marco de este proyecto como para comentarla. A pesar de eso, se estuvo dudando si escoger Bitbucket o GitHub por el simple motivo de que Bitbucket nos permite una integración completa con Jira (la herramienta de gestión). Al final se acabó descartando ya que no se consideró especialmente útil al tratarse de un proyecto desarrollado por una sola persona.

#### Jira

Al igual que ocurre con el control de versiones, existen muchas herramientas de gestión de tareas. Sin embargo, Jira es una de las más intuitivas que existen en el mercado actualmente. Permite crear un tablero Scrum con solo un *click*, añadir tareas de forma muy sencilla, crear *sprints* y gestionar dichos *sprints* sin ni siquiera saber utilizar el programa. A pesar de esto, y como pasaba en el caso anterior, se ha elegido esta herramienta por la experiencia que se tiene sobre la misma.

#### IDEs

Cuando hablamos de desarrollo de aplicaciones Android, se nos hace difícil no pensar en Android Studio. Hoy en día, es el programa referencia para el desarrollo de dichas aplicaciones. La guía de desarrollo de Android nos recomienda directamente este programa [19] y este es el principal motivo por el cual se escogió este IDE por encima de otros que también permiten el desarrollo Android, como puede ser Eclipse.

En cuanto al programa escogido para el desarrollo de la API, IntelliJ IDEA, fue elegido debido a que está directamente ligado a Android Studio, pues, a pesar de que Android Studio pertenece a Google, está basado en la herramienta de Jetbrains. Es por eso por lo que los dos IDE tienen interfaces muy similares, cosa que facilita enormemente el desarrollo, evitándonos tener que aprender a utilizar dos IDE completamente distintos.

### 4.3.3 Lenguajes de programación

Kotlin, el lenguaje empleado para el desarrollo de la aplicación Android, se convirtió en lenguaje oficial para Android en el año 2017. Este, junto a Java, son los dos lenguajes referencia en cuanto al desarrollo de aplicaciones Android se refiere.

La elección de Kotlin [20] fue sencilla, pues es el lenguaje recomendado por la guía de desarrollo de Android, a pesar de esto, me gustaría remarcar alguna ventaja que tiene Kotlin sobre Java. Y es que, el código en Kotlin es mucho más conciso, es decir, la escritura de código Kotlin es mucho más compacta y sencilla que en Java.

En cuanto al *framework* empleado en la API, se ha escogido Spring, esta elección está directamente relacionada con el hecho de que, durante la carrera, este ha sido el único *framework* que he estudiado para el desarrollo de una API REST en Java. Por lo tanto, la elección de una tecnología ya conocida es algo que nos evita algunos riesgos propios de la inexperiencia de un *framework*

### 4.3.4 Base de datos

La principal decisión en cuanto a la base de datos se refiere fue la de utilizar una relacional o una no relacional. Y, al igual que en el resto de las tecnologías, se ha escogido principalmente una base de datos relacional ya que son las que se han enseñado en la carrera.

Si más no, existen ciertas ventajas en utilizar bases de datos relacionales [21] que me gustaría comentar. En primer lugar, la atomicidad de la base de datos, esto es algo que se ha inculcado mucho en las asignaturas destinadas a bases de datos, una operación, o se realiza por completo o no se realiza. En segundo lugar, la simplicidad y el soporte que tienen este tipo de base de datos es mucho más elevado que en el caso de las no relacionales.

Además, y puesto que no se va a tratar con datos masivos, considero que utilizar bases de datos no relacionales no era necesario en el caso de este proyecto. Otro de los motivos para elegir una base de datos relacional es el uso de datos estáticos, pues no tenemos la necesidad de utilizar estructuras dinámicas almacenadas en la base de datos.

# 5. Riesgos asociados a la construcción del sistema

En esta sección se recogen todos los riesgos asociados a la construcción del sistema software que se propone, su identificación, evaluación, y finalmente los posibles planes para gestionar de forma eficiente esos riesgos en caso de que llegasen a ocurrir. Toda la información necesaria para el desarrollo de esta sección ha sido extraída del libro *Software Engineering - A Practitioner's Approach* [22].

## 5.1 Identificación de riesgos

### 5.1.1 Riesgos de negocio

Los riesgos de negocio [23] son los que en caso de que ocurran, el proyecto carecería de sentido, y podría llegar a cancelarse. A continuación, se exponen los distintos riesgos de negocio que se han considerado para este proyecto.

#### Utilidad

Si bien es verdad que esto ya ha sido mencionado con anterioridad, cabe recalcar que el tiempo es un bien muy preciado, tanto el del paciente como en especial el del personal médico. Con el sistema que se propone se ahorraría muchísimo tiempo de visitas innecesarias que se automatizarían completamente gracias al sistema. Esto convierte al sistema propuesto en algo realmente útil para este sector.

Y, como ya hemos comentado también, en estos tiempos tendríamos que añadir el factor contagio, el cual reduciremos mucho evitando tener que ir a un centro médico (foco de contagio) para que simplemente se modifique la medicación de los pacientes crónicos.

#### Estrategia

Quizás este es el riesgo que más deberíamos tener en cuenta, y el motivo es muy claro, actualmente el sistema sanitario español no está muy modernizado. Una gran parte del sector médico es reticente a los cambios que implican la inclusión de tecnología, esto podría generar un gran rechazo y la falta de colaboración de este sector con el proyecto.

A pesar de eso, no es un sistema software demasiado invasivo y por lo tanto tampoco debería tener una difícil integración dentro del sistema de salud. Pues a pesar de todo, casi todo el mundo dispone de un dispositivo móvil desde el cual poder utilizar el sistema.

#### Dificultad de uso

Es bien sabido que los sistemas software no son muy utilizados por personas de edad más avanzada. No obstante, estas personas suelen contar con cuidadores, ya sean familiares, cuidadores privados, o proporcionados por el gobierno. Estas personas podrían ayudarles a hacer uso del sistema. Si nos referimos ahora a personas más familiarizadas con el uso de estos dispositivos, no debería de representar ningún problema el manejo del sistema software propuesto.

#### Competencia

Durante el transcurso del proyecto, o una vez se haya finalizado, existe la posibilidad de que alguna entidad lance un sistema parecido al que se propone. En ese caso, el sistema correría un grave peligro, ya que es probable que, si este nuevo sistema cuenta con el apoyo directo de algún gobierno comunitario, del gobierno central, o incluso sea propuesto por una gran empresa como podría ser Accenture [24], se convierta en un sistema mucho más popular que el propuesto en esta memoria.

Para tratar de evitar esto, se tiene que construir un sistema que realmente sea difícil de imitar y que aporte funcionalidades únicas, innovadoras y realmente necesarias para el público objetivo.

### 5.1.2 Riesgos técnicos

Los riesgos de desarrollo son aquellos que surgen mientras se está implementando el sistema. En este caso, el sistema constará de dos partes, una API desarrollada en Java y una aplicación desarrollada en Kotlin.

#### Disponibilidad del sistema

El sistema tiene que estar disponible siempre, si un paciente se realiza una analítica la cual sale gravemente alterada, necesita modificar su medicación lo antes posible, si este aviso no le llega a su dispositivo es posible que no llegue a modificarla nunca, con las consecuencias en su salud que eso conlleva.

A pesar de ser un riesgo muy a tener en cuenta, las bases de datos estarán alojadas en Amazon Web Services [25], lo cual nos proporciona la seguridad de que van a estar disponibles la mayoría del tiempo. Adicionalmente, el sistema va a estar distribuido, AWS realiza este proceso de forma automática creando varias instancias que garantizan una disponibilidad total del sistema.

#### Bugs y errores

El principal riesgo asociado al desarrollo es el tema de los *bugs* y los errores en la ejecución de la aplicación. Estos errores pueden causar el cierre de la aplicación e incluso que no se vuelva a abrir hasta recibir actualizaciones. Por lo tanto, se tendrá que prestar especial atención a las pruebas de la aplicación para evitar este tipo de errores que harían que los usuarios dejaran de utilizar el sistema.

#### Seguridad

Finalmente, al tratarse de un sistema que está en contacto directo con datos extremadamente privados de las personas, existe la posibilidad de que el sistema diseñado no sea lo suficientemente seguro y existan ciertos puntos débiles que se puedan explotar para obtener de forma ilegal los datos de los usuarios del sistema.

#### Inexperiencia con las tecnologías

Cuando se comienza un proyecto con unas tecnologías completamente distintas, todo se complica. Las tecnologías que se van a utilizar para desarrollar el sistema son poco conocidas por el desarrollador principal, y es por eso por lo que existe un riesgo asociado a la dificultad de aprendizaje.

#### Imprecisión de las respuestas

El procesado de las analíticas tiene que ser tan exacto o más como lo sería el mismo procesado por parte del personal médico, existe el riesgo de que no lo sea y que por lo tanto las respuestas proporcionadas por el sistema no sean realmente útiles para tratar la enfermedad del paciente.

### 5.1.3 Riesgos de proyecto

Los riesgos de proyecto son esos que en caso de que ocurran, tanto el presupuesto como la planificación se verán afectados. Estos riesgos no están relacionados directamente con aspectos técnicos de la implementación, sino que pueden estar relacionados con distintos aspectos.

#### Tareas subestimadas

Nunca se puede llegar a conocer con exactitud el tiempo que te va a llevar el realizar una tarea, es por eso por lo que existe la posibilidad de que existan tareas planificadas para ser realizadas en un número de horas y que en realidad acaben comportando más. Esto puede ser debido a la complejidad de la tarea o a que ocurran problemas que no se hayan tenido en cuenta a la hora de cuantificar la cantidad de faena implicada en dicha tarea.

#### Fallos en el hardware

Todo el desarrollo será llevado a cabo en un ordenador de sobremesa, siempre existe la posibilidad de que algún componente de este deje de funcionar (pantalla, teclado, ratón o el PC per se). Esto sería un problema que tendría un verdadero impacto en la planificación del proyecto.

#### Incapacidad de contactar con personal sanitario

Para el desarrollo del módulo central del sistema (procesado de analíticas) se necesita estar en contacto directo con médicos para que puedan ayudar con la parametrización de las enfermedades, contactar con estas personas es vital para el buen desarrollo del proyecto. En la situación actual con la pandemia, estas personas están más desbordadas que nunca, es por eso por lo que el riesgo de no poder contar con estas personas es muy elevado.

#### Mala especificación de requisitos

Cuando los requisitos son descritos por una persona e implementados por otra, existe la posibilidad de que no se acaben de entender entre ellos, con todos los problemas que esto llega a causar en la planificación del proyecto. En este caso tanto la especificación de requisitos como la implementación son llevados a cabo por la misma persona cosa que hace disminuir el impacto de este riesgo.

## 5.2 Evaluación de riesgos

A continuación, podemos ver la tabla con el listado de riesgos ordenados por orden de importancia (de mayor a menor).

*Tabla 1: Listado de riesgos asociados al sistema. Fuente: Elaboración propia.*

| Identificador | Nombre | Categoría | Probabilidad (%) | Impacto |
|---|---|---|---|---|
| R1 | Incapacidad de contactar con personal sanitario | Proyecto | 40 | Catastrófico |
| R2 | Bugs y errores | Técnico | 60 | Crítico |
| R3 | Imprecisión de las respuestas | Técnico | 30 | Crítico |
| R4 | Competencia | Negocio | 20 | Crítico |
| R5 | Seguridad del sistema expuesta | Técnico | 20 | Crítico |
| R6 | Utilidad | Negocio | 5 | Catastrófico |
| R7 | Estrategia | Negocio | 10 | Crítico |
| R8 | Dificultad de uso | Negocio | 10 | Crítico |
| R9 | Disponibilidad del sistema | Técnico | 1 | Catastrófico |
| R10 | Inexperiencia con las tecnologías | Técnico | 50 | Marginal |
| R11 | Tareas subestimadas | Proyecto | 40 | Marginal |
| R12 | Problemas con el hardware | Proyecto | 2 | Crítico |
| R13 | Mala especificación de requisitos | Proyecto | 5 | Marginal |

El impacto de los riesgos ha sido evaluado siguiendo la escala siguiente:

1. Catastrófico.
2. Crítico.
3. Marginal.
4. Negligible.

La línea gruesa justo por encima de R12 implica que todos los riesgos por debajo de la misma no necesitan ser tratados, pues su probabilidad de ocurrir junto con su impacto no es suficientemente relevante como para hacerlo.

## 5.3 Mitigación, monitorización y gestión de riesgos (RMMM)

Para tratar de evitar que los riesgos listados anteriormente tengan un gran impacto sobre el proyecto, se van a trazar unos planes de mitigación, monitorización y finalmente de gestión de dichos riesgos. En primer lugar, se van a listar las acciones a realizar para tratar de mitigar la probabilidad de ocurrencia del riesgo. En segundo lugar, se indicarán los pasos a seguir para monitorizar dicho riesgo una vez comience el desarrollo. Y, finalmente, se propondrá un plan de gestión para el caso en que el riesgo se acabe convirtiendo en realidad.

### 5.3.1 Incapacidad de contactar con personal sanitario

Mitigación:

- Tener agendados a varios médicos.
- Hacer un estudio sobre que enfermedades se van a procesar para evitar realizar esta pregunta a los médicos agendados.
- Diversificar las dudas entre los diversos médicos agendados.
- Realizar consultas en grupos de estudiantes de medicina en años avanzados (quinto y sexto).

Monitorización:

La única monitorización posible en este caso es la de preguntar directamente a cada médico si va a tener tiempo real para poder atender las dudas que puedan ir apareciendo a lo largo del desarrollo, además del envío de información para la parametrización de las enfermedades.

Gestión:

En caso de ser incapaces de contactar con ningún médico, lo único que se puede hacer es tratar de buscar otros que puedan ayudar. En ese caso se trataría de contactar con estudiantes de medicina, priorizando personas que están cursando el MIR.

### 5.3.2 Bugs y errores

Mitigación:

- Tratar de separar todas las funcionalidades nuevas en ramas distintas.
- Dejar un tiempo destinado al tratamiento de bugs y errores al final de cada *sprint*.
- Anotar los posibles puntos críticos que pueda tener una tarea antes de comenzar la implementación.

Monitorización:

- Tener en cuenta todos los puntos críticos una vez se está implementando.
- Anotar en Jira todos los errores que se van encontrando.
- Realizar un control exhaustivo durante las pruebas para no pasar por alto ningún error.

Gestión:
En caso de que aparezcan estos problemas, lo que se realizará para arreglarlos es otra iteración (una o dos semanas) en la cual se tratará de paliar los errores que hayan ido apareciendo. En particular, esto haría que la fase de desarrollo se alargara el tiempo necesario para poder solucionar estos errores, y es por eso por lo que se ha dejado un margen de tiempo a principios de junio para poder solucionar estos inconvenientes. A priori, se estima que, en caso de ocurrir dichos errores, el proyecto se podría alargar como máximo dos semanas, si más no, no harán falta más recursos para tratar de paliar estos errores, simplemente hará falta tiempo y dedicación por parte del desarrollador del proyecto.

### 5.3.3 Imprecisión de las respuestas

Mitigación:

- Elaborar un documento de análisis de las enfermedades y sus respectivos parámetros.
- Contactar con varios médicos para contrastar respuestas.
- Realizar un plan de desarrollo que permita ajustar el procesado de analíticas de forma rápida.

Monitorización:
La monitorización de esta tarea tendrá que ser llevada a cabo por el personal médico, a lo largo de la implementación se irán realizando pruebas, generando resultados y enviándolos a los médicos que están colaborando con el proyecto.

Gestión:
Contactar con un equipo médico especializado en la enfermedad que está generando malos resultados para tratar de refinar el procesado de las analíticas.

### 5.3.4 Competencia

Mitigación:

- Realizar un producto realmente único y difícil de plagiar.
- Proporcionar un buen servicio al usuario para que no exista la necesidad de cambiar el sistema por otro.

Monitorización:
Monitorizar a la competencia es una tarea complicada, los proyectos en los que trabajan suelen ser secretos, y actualmente no existe ninguno que se asemeje para poder ir controlando los avances que realiza.

Gestión:
Como ya se ha comentado, la salida al mercado de un producto similar por parte de una gran empresa sería catastrófico para el proyecto, en caso de que ocurriera se trataría de hacer destacar partes del sistema que no existan en la competencia.

### 5.3.5 Seguridad del sistema expuesta

Mitigación:

- Seguir todos los criterios de seguridad incluidos en la documentación de Spring.
- Promover la no compartición de los datos de acceso entre los usuarios.
- Cifrar datos clave como las contraseñas.
- Aislar los datos de los usuarios para que solo sean accesibles por ellos mismos.

Monitorización:

- Comprobar que se están siguiendo los criterios de seguridad.
- Realizar comprobaciones de vulnerabilidad de la base de datos.
- Comprobar que se están guardando los datos clave cifrados en la base de datos.

Gestión:

Se tendrá que contratar a una persona que sea experta en ciberseguridad para que verifique el sistema con la intención de encontrar y reparar todos sus puntos débiles. Esto supondrá un incremento en el coste total del proyecto y además en su duración, pues no se puede estimar cuanto podría llegar a tardar esta persona en llegar a subsanar todas las posibles brechas de seguridad.

### 5.3.6 Utilidad

Mitigación:

- Seguir los criterios de usabilidad para el desarrollo de aplicaciones móvil.
- Desarrollar un sistema rápido y eficaz.
- Realizar un estudio de mercado que resalte las funcionalidades que realmente vayan a ser útiles.

Monitorización:

Para monitorizar la utilidad del sistema, se tendrán que realizar encuestas a los usuarios de la aplicación para comprobar si están contentos con la misma y si realmente les ayuda en su día a día.

Gestión:

En caso de que la aplicación no resulte útil por el tipo de formato (aplicación móvil), se podría trasladar toda la parte gráfica a una página web sin necesidad de realizar un cambio en la API.

En caso de que sea problema de un mal diseño de la aplicación, se debería de contratar a un experto en usabilidad para que pueda resaltarnos los problemas que tiene.

### 5.3.7 Estrategia

Mitigación:

- Realizar entrevistas con personal sanitario de renombre para que muestren interés con el sistema.
- Realizar una campaña de difusión entre los pacientes crónicos para que hagan uso del sistema.

Monitorización:

La monitorización será directamente con el número de usuarios que tenga el sistema y los comentarios que dejen en la plataforma de descargas de donde se bajen la aplicación.

Gestión:

En caso de que el sistema no tenga una buena acogida desde un inicio, se podrá contratar a una empresa externa para que se encargue de toda la parte de marketing y publicidad, permitiendo de esta manera hacer más visible el sistema.

### 5.3.8 Dificultad de uso

Mitigación:

- Diseñar una aplicación intuitiva y fácil de usar.
- Seguir los criterios propuestos por Google para el diseño de aplicaciones.
- Realizar planes de prueba para enviar la aplicación a *stakeholders* pertenecientes a distintos colectivos.

Monitorización:

- Recoger los resultados del *feedback* obtenido de los *stakeholder*.
- Estudiar el grado de dificultad actual de la aplicación a partir de los resultados obtenidos.

Gestión:

En caso de que la aplicación resulte en una difícil de utilizar, gracias a los estudios realizados, y a las respuestas obtenidas, se podrán modificar aquellas zonas de la aplicación que la gente considere de difícil acceso o uso.

### 5.3.9 Disponibilidad del sistema

Mitigación:

- Contratar un servicio de alojamiento de la API y BBDD de mucha fiabilidad.

Monitorización:

Llevar un registro exhaustivo de las caídas que pueda sufrir el servicio contratado (AWS).

Gestión:

En caso de que el servicio no sea el adecuado, y la disponibilidad del sistema se vea afectada, la mejor solución sería la de migrar el sistema a otro servicio que cuente con una disponibilidad mayor del contratado.

### 5.3.10 Inexperiencia con las tecnologías

Mitigación:

- Realizar un tutorial de Android previo al comienzo de la implementación.
- Realizar un tutorial de Spring previo al comienzo de la implementación.
- Recoger un gran listado de páginas de soporte de las tecnologías que se van a utilizar y anotarlas en un documento.
- Realizar una investigación previa al inicio de una tarea para clarificar los aspectos complicados de dicha funcionalidad.

Monitorización:

Controlar que las tareas se estén realizando en el tiempo previsto y que no exista un retraso añadido por causa de la inexperiencia con las tecnologías.

Gestión:

En el caso de que se tarde más en realizar una tarea por culpa de la inexperiencia, lo único que se podrá hacer es dedicarle más horas, ya sea para completar la tarea, o para adquirir la experiencia necesaria como para poder realizarla.

### 5.3.11 Tareas subestimadas

Mitigación:

- Detallar todas las tareas lo máximo posible para que no se escape ni un detalle.
- Asignar un tiempo determinado a cada tarea.

Monitorización:

Tras finalizar cada *sprint*, revisar de nuevo las tareas pendientes adaptando el numero de horas teniendo en cuenta la experiencia adquirida en el sprint finalizado.

Gestión:

En caso de que esto ocurra, lo único que se podrá hacer es dedicarle más horas a esa tarea en concreto. Y, como ya se ha comentado, al disponer de un par de semanas extra a principios de junio, no supondría un problema grave. Además, al tratarse de un proyecto *agile* es posible que el contenido de las iteraciones varíe, en caso de que eso ocurra, se podrán llegar a extender dichas iteraciones los días que sea necesario.

# 6. Planificación inicial del proyecto

Este proyecto, como ya se ha comentado, se realiza con la intención de finalizar el trabajo de fin de grado del grado en ingeniería informática. Por esta razón, el periodo de tiempo en el que se tiene que realizar es limitado, en concreto, el proyecto comenzó el día 1 de febrero de 2021 y terminará con la lectura del trabajo a finales de junio.

El proyecto está dividido en dos fases muy diferenciadas, la primera está destinada a toda la parte de gestión del proyecto. La segunda fase incluye todo lo relacionado con el desarrollo del sistema software que se plantea. Esta fase, estará dividida en *sprints* (Scrum), cada *sprint* será de doce días y habrá un total de 5.  En total, este proyecto está pensado para ser completado en 570 horas, repartidas desde principios de febrero hasta finales de junio.

## 6.1 Gestión del proyecto

Esta primera fase, destinada puramente a la gestión y organización del proyecto, está pensada para ser completada en 205 horas. Sin embargo, al tratarse de un proyecto que sigue la metodología *Agile*, habrá partes como la especificación de requisitos que irán evolucionando con el proyecto, y por lo tanto estas horas que he asignado a priori son las destinadas para el buen comienzo del trabajo.

Las tareas que se van a listar a continuación son una explicación más detallada de lo que se puede ver resumido en la tabla 1. Todas las tareas requieren del desarrollador del proyecto, y, además, algunas necesitarán del director del TFG (Xavier Franch), del tutor del GEP (Joan Subirats), y de personal médico. Las dependencias de cada tarea están especificadas en la tabla resumen para poder visualizarlas con una mayor claridad.

En primer lugar, tenemos las tareas del GEP, el orden que se ha seguido para completar estas tareas es el propuesto por los coordinadores y profesores organizadores del GEP.

- GEP1 – Contextualización y alcance: Definir el contexto y el alcance que va a tener el proyecto. Para la realización de esta tarea, se ha hecho uso de Microsoft Word, y draw.io. Esta tarea está pensada para ser realizada en unas 25 horas y consta de lo siguiente:
    - Introducir el proyecto, contextualizarlo y definir el problema que se trata de resolver.
    - Definir los actores implicados en el proyecto (a quien va dirigido, quien lo usa y quien se beneficia).
    - Describir y analizar las alternativas existentes en el mercado actualmente comparándolas con el sistema que se plantea.
    - Definir los objetivos del proyecto de forma genérica, así como los subobjetivos.
    - Definir los requisitos no funcionales.
    - Identificar y definir los posibles riesgos a los que se puede llegar a enfrentar el sistema.

- GEP2 – Planificación temporal: Establecer la planificación que va a seguir nuestro proyecto. Para el desarrollo de esta tarea se han utilizado Microsoft Word y Monday planning (para el diagrama de Gantt). Esta tarea está pensada para ser realizada en unas 15 horas y consta de lo siguiente:
    - Describir las diferentes tareas que van a componer el trabajo.
    - Realizar el diagrama de Gantt.
    - Tratamiento de riesgos.
- GEP3 – Presupuesto y sostenibilidad: Evaluar el posible coste del proyecto y el posible impacto medioambiental. Para esta tarea se utilizará Microsoft Word y Microsoft Excel. Esta tarea está pensada para ser realizada en unas 10 horas y consta de lo siguiente:
    - Identificación y estimación de los costes del proyecto.
    - Planteamiento de mecanismos para controlar las posibles desviaciones en el proyecto.
    - Informe de sostenibilidad.
- GEP4 – Entrega final: Aplicar el *feedback* obtenido del tutor de GEP en las entregas 1, 2 y 3 para generar la entrega final del módulo. Para esta fase se utilizará Microsoft Word y se necesitará de la obtención del *feedback* por parte del tutor del GEP. Esta tarea está pensada para ser realizada en 20 horas y consta de lo siguiente:
    - Analizar el *feedback* de cada entrega.
    - Aplicar los consejos sugeridos por el tutor del GEP.
    - Revisar el documento.
- GEP5 – Reuniones con personal sanitario y director TFG: Reuniones periódicas para resolver dudas y para la obtención de *feedback* por parte del personal sanitario con el que estoy en contacto y por parte del director del TFG. Para la realización de esta tarea es necesario contar con personal sanitario especializado en las enfermedades crónicas que voy a automatizar, mi director del TFG y una herramienta para tomar notas. Esta tarea se estima en unas 15 horas y consta de lo siguiente:
    - Preparar las reuniones.
    - Realizar las reuniones.
    - Extraer conclusiones.
- GEP6 – Gestión del *backlog* y tareas *agile*: Esta tarea se irá desarrollando a lo largo del proyecto, se trata de ir gestionando el backlog y todo tipo de tareas relacionadas con la metodología agile. Para la realización de esta tarea será necesario Jira y Word. Está pensada para ser realizada en 25 horas repartidas a lo largo del cuatrimestre y consta de lo siguiente:
    - Gestión del *backlog* (añadir/eliminar/modificar tareas).
    - Reuniones retrospectivas.
    - *Sprint planning*.
    - Gestión de los requisitos (añadir/eliminar/modificar).

A continuación, tenemos las tareas relacionadas con el comienzo del proyecto (*inception*). Este grupo de tareas son necesarias para lograr un buen inicio de proyecto.

- IN1 – *Stakeholders*: Definir las distintas partes interesadas en el proyecto. Esta tarea está pensada para ser realizada en aproximadamente 10 horas y consta de lo siguiente:
    - Extensión de los *stakeholders* propuestos en la fase del GEP.
    - División de los *stakeholders* según su interés en el proyecto.
    - Definición de los objetivos de cada *stakeholder*.
    - Definición del rol de cada *stakeholder*.
- IN2 – Objetivos de los principales *stakeholders*: Para las principales partes interesadas en el proyecto, extender sus principales objetivos. Esta tarea está pensada para ser realizada en 5 horas y consta de lo siguiente:
    - Realizar una encuesta a los principales *stakeholders*.
    - Analizar los datos de la encuesta.
    - Extender los objetivos.
- IN3 – Requisitos funcionales iniciales: Definir los requisitos funcionales iniciales del sistema. Esta tarea está pensada para ser completada en 20 horas y consta de lo siguiente:
    - Definir los casos de uso iniciales del sistema.
    - Realizar un diagrama de casos de uso.
    - Especificar de forma completa los casos de uso.
- IN4 – Requisitos no funcionales: Seguir la plantilla de Volere para definir todos los requisitos no funcionales del sistema. Esta tarea está pensada para ser completada en 10 horas.
- IN5 – Esquema conceptual inicial: Realizar el esquema conceptual inicial del sistema. Esta tarea está pensada para ser completada en 15 horas y consta de lo siguiente:
    - Realizar el diagrama UML.
    - Redactar las restricciones textuales.
    - Explicar con detalle las distintas clases incluidas en el diagrama.
- IN6 – Identificación y gestión de riesgos: Analizar y detallar los posibles riesgos a los que se puede llegar a enfrentar el sistema además de trazar planes iniciales para paliar estos riesgos. Esta tarea está pensada para ser completada en 20 horas y consta de lo siguiente:
    - Buscar y relacionar los riesgos existentes con el proyecto.
    - Documentar los posibles riesgos del sistema.
    - Analizar el impacto de cada riesgo identificado.
    - Refinar los riesgos previamente definidos.
    - Establecer planes de gestión, mitigación y monitorización para los riesgos identificados.
- IN7 – Diseño de la arquitectura inicial: Analizar y estudiar las posibles alternativas arquitectónicas que podrá tener la aplicación. Esta tarea está pensada para ser completada en 10 horas. Materiales: Microsoft Word y draw.io.
- IN8 – Diseño de *mockups* representativos: Diseñar toda la parte visual que tendrá la aplicación, esta tarea está pensada para ser completada en 5 horas. Materiales: Justinmind.

## 6.2 Desarrollo del proyecto

El desarrollo del proyecto va a estar compuesto por dos tipos de tareas principales: tareas de desarrollo de la aplicación móvil y tareas destinadas a la implementación de la API.

Es por eso por lo que habrá tareas que tendrán dos partes, una primera parte de implementación de toda la parte necesaria de la API, y la segunda, que será la implementación en Android de la tarea.

Además, para poder empezar con la implementación, se precisan de dos tareas fundamentales de preparación, las cuales no tienen dependencia con ninguna otra, pero serán esenciales para el buen desarrollo del proyecto.

- PP1 – Aprendizaje de las tecnologías: Tanto Kotlin como Spring (las tecnologías que voy a utilizar en la implementación) son relativamente nuevas para el principal desarrollador (a pesar de haber trabajado algo con Spring), es por eso por lo que se va a tener que dedicar un tiempo al aprendizaje de estas. Esta tarea requerirá de AndroidTutorials (para aprender Kotlin), Udemy (Spring) y Notepad para tomar notas de todo lo aprendido. Esta tarea está pensada para ser completada en 25 horas y constará de lo siguiente:
    - Completar un tutorial de Android.
    - Completar un tutorial de Spring.
- PP2 – Establecimiento del entorno: Para comenzar con el proyecto, es necesario establecer todo el entorno que se va a necesitar para el desarrollo. Esta tarea requerirá de Android Studio, IntelIJ, Github y Jira. Está pensada para ser completada en 20 horas y constará de lo siguiente:
    - Instalación y preparación del entorno de Android Studio.
    - Instalación y preparación del entorno de IntellIJ.
    - Creación de repositorio en Github y conectarlo con los IDEs.
    - Creación del proyecto en Jira.

Para la realización de todas las tareas que vienen a continuación, será necesario contar con los IDEs mencionados con anterioridad (Android Studio, IntellIJ), Github, y Jira para la gestión de los *sprints*. Además, puesto que al finalizar cada *sprint* se realizará una reunión con el director del TFG, también será necesaria su presencia.

*Sprint* 1 (13/04/2021 - 24/04/2021):

- DVP1 – Implementación inicial de la API: Implementar las bases de la API para poder empezar a utilizarla. Esta tarea es fundamental para poder empezar el proyecto. Está pensada para ser completada en 10 horas.
- DVP2 – Inicio de sesión: Se implementará todo lo necesario para la creación de una cuenta por parte del usuario y para el inicio de sesión. Esta tarea está pensada para ser completada en 20 horas, tiene que ser completada después de DVP1 y consta de lo siguiente:
    - Implementar la parte visual de la creación de cuenta.
    - Implementar la parte visual del inicio de sesión.
    - Implementar la parte de la API que se va a encargar de la gestión de usuario.

- DVP3 – Pantalla principal de la aplicación: Se creará la interfaz de la pantalla principal, la cual nos permitirá navegar hacia las distintas secciones que tendrá la aplicación. Esta tarea está pensada para ser completada en 15 horas y consta de lo siguiente:
    - Implementar la parte visual de la pantalla principal.
    - Diseñar e implementar el grafo de navegación de la aplicación.

*Sprint* 2 (25/04/2021 - 06/05/2021):

- DVP4 – Ajustes: Esta tarea recogerá todo lo relacionado con el perfil del usuario y sus ajustes. Está pensada para ser completada en 20 horas y consta de lo siguiente:
    - Creación de la interfaz que se corresponderá con la parte de ajustes y perfil de usuario.
    - Modificación de contraseña.
    - Modificación del idioma.
- DVP5 – Notificaciones: Con la realización de esta tarea se pretende desarrollar un sistema de notificaciones capaz de recoger datos desde la API y enviarlos a la aplicación para que esta la muestre al usuario. Esta tarea está pensada para ser completada en 15 horas y consta de lo siguiente:
    - Implementación en la API de todo el sistema de notificaciones.
    - Implementación de la recogida de notificaciones por parte de la aplicación.
    - Diseño e implementación de la notificación a mostrar al usuario.
- DVP6 – Creación de las tarjetas para el visionado de datos: La aplicación mostrará los datos en formato tarjeta, para que esto pueda suceder, necesitamos crear una manera sencilla y reutilizable para poder mostrar los distintos datos con los que contará la aplicación. Esta tarea está pensada para ser completada en 12 horas.

*Sprint* 3 (07/05/2021 - 19/05/2021):

- DVP7 – Procesado de analíticas: Esta tarea estará centrada en la API, y será la pieza central del sistema. Está pensada para ser completada en 45 horas y consta de lo siguiente:
    - Analizar los datos obtenidos de las entrevistas con los médicos especialistas en las enfermedades crónicas que trata el sistema.
    - Implementar un sistema que lea los datos de una analítica y los procese dependiendo de la enfermedad que estamos analizando.
    - Generar una respuesta acorde a los datos procesados.
    - Enviar la respuesta a la aplicación para que sea mostrada.
- DVP8 – Visionado de la medicación: Implementación de toda la parte visual donde se mostrará la medicación que debe de tomar el paciente para cada enfermedad que padezca. Para que esta tarea se pueda realizar, es fundamental que DVP6 esté completada. Esta tarea está pensada para ser completada en 20 horas y consta de lo siguiente:
    - Envío de los valores generados por la API.
    - Recoger los valores de la API y mostrarlos en la aplicación en formato tarjeta.
    - Envío de notificación cuando se añade/modifica/elimina una medicación.

*Sprint* 4 (20/05/2021 - 30/05/2021):

- DVP9 – Visionado de analíticas: Implementación de toda la parte visual donde se mostrarán los resultados de las analíticas a las que se ha sometido el paciente. Para que esta tarea se pueda realizar, es fundamental que DVP6 esté completada. Esta tarea está pensada para ser completada en 15 horas y consta de lo siguiente:
    - Recogida de las analíticas de la base de datos y pase de los datos a la aplicación.
    - Recoger los datos en la aplicación y mostrarlos en formato tarjeta.
    - Envío de notificación al añadir una nueva analítica.
- DVP10 – Calendario: Implementación del sistema de citas del que dispondrá la aplicación. Esta tarea está pensada para ser completada en 20 horas y consta de lo siguiente:
    - Recogida de las citas de base de datos.
    - Envío de la información a la aplicación para ser mostrada con claridad.
    - Modificar el formato del calendario.
    - Envío de notificación cuando se añade/modifica/elimina una cita.
    - Reclamar una cita.

*Sprint* 5 (31/05/2021 - 11/06/2021):

- DVP11 – Chat: Implementación de un sistema de chat instantáneo. Esta tarea está pensada para ser completada en 25 horas y consta de lo siguiente:
    - Implementación de la interfaz gráfica del chat.
    - Implementación del listado de chats.
    - Recogida y guardado de chats en base de datos.
    - Notificaciones.
- T1 – Pruebas: Realización de pruebas finales de la aplicación y de la API. Esta tarea está pensada para ser completada en 20 horas y consta de lo siguiente:
    - Pruebas funcionales de la aplicación Android.
    - Distribución de la aplicación y recogida de *feedback*.

## 6.3 Cierre del proyecto

Tras finalizar con la fase de desarrollo se comenzará con las tareas de cierre de proyecto, como son la de documentar todo lo que reste y preparación de la memoria.

- D1 – Documentación implementación: Documentar toda la parte relacionada con la implementación del proyecto. Esta tarea se irá haciendo de forma gradual y en total se estima que implicará unas 60 horas. Constará de lo siguiente:
    - Documentar todas las BBDD utilizadas.
    - Documentar el *testing*.
    - Documentar todo el proceso de implementación.
- D2 – Documentación final: Esta tarea consistirá en la revisión final del trabajo. Esta tarea está pensada para ser completada en 20 horas y constará de lo siguiente:
    - Revisión y corrección del trabajo.
    - Actualización de tablas de contenido y referencias.
- M1 – Lectura TFG: Esta tarea consiste en todo lo necesario para la preparación de la lectura del TFG. Esta tarea requerirá de Microsoft Word y Power Point. Está pensada para ser completada en 25 horas.

## 6.4 Resumen de las tareas

A continuación, podemos ver la tabla resumen de las tareas, cada color identifica a un tipo de tarea en particular. En el caso de las tareas de desarrollo, colores distintos identifican *sprints* distintos.

*Tabla 2: Resumen de las tareas a realizar. Fuente: Elaboración propia.*

| ID | Nombre | Tiempo(h) | Tareas predecesoras | Recursos y programas usados |
|---|---|---|---|---|
| GEP1 | Contextualización y alcance | 25 | | Word, draw.io |
| GEP2 | Planificación temporal | 15 | GEP1 | Word, Monday planning |
| GEP3 | Presupuesto y sostenibilidad | 10 | GEP2 | Word, Excel |
| GEP4 | Entrega final | 20 | GEP3 | Word, director GEP |
| GEP5 | Reuniones con personal sanitario y director TFG | 15 | | Notepad, Google meet, director TFG, Personal sanitario |
| GEP6 | Gestión del backlog y tareas agile. | 25 | | Jira, Google meet |
| | ***Total horas GEP*** | ***110*** | | |
| IN1 | *Stakeholders* | 10 | | Word |
| IN2 | Objetivos de los principales *stakeholders* | 5 | ER1 | Word |
| IN3 | Requisitos funcionales iniciales | 20 | ER2 | Word |
| IN4 | Requisitos no funcionales | 10 | | Word |
| IN5 | Esquema conceptual inicial | 15 | | Word, draw.io |
| IN6 | Identificación y gestión de riesgos | 20 | | Word |
| IN7 | Diseño de la arquitectura inicial | 10 | | Word, draw.io |
| IN8 | Diseño de mockups representativos | 5 | | Justinmind |
| | ***Total horas IN*** | ***95*** | | |
| PP1 | Aprendizaje de las tecnologías | 25 | | AndroidTutorials, Udemy, Notepad |
| PP2 | Establecimiento del entorno | 20 | PP1 | Android Studio, IntelIJ, Github, Jira |
| | ***Total horas PP*** | ***45*** | | |
| DVP1 | Implementación inicial de la API | 10 | PP2 | Android Studio, IntelIJ, Github, Jira |
| DVP2 | Inicio de sesión | 20 | DVP1 | Android Studio, IntelIJ, Github, Jira |
| DVP3 | Pantalla principal | 15 | | Android Studio, IntelIJ, Github, Jira |
| DVP4 | Ajustes | 20 | | Android Studio, IntelIJ, Github, Jira |
| DVP5 | Notificaciones | 15 | | Android Studio, IntelIJ, Github, Jira |
| DVP6 | Creación de las tarjetas para el visionado de datos | 12 | | Android Studio, IntelIJ, Github, Jira |
| DVP7 | Procesado de analíticas | 45 | | Android Studio, IntelIJ, Github, Jira |
| DVP8 | Visionado de la medicación | 20 | DVP5 | Android Studio, IntelIJ, Github, Jira |
| DVP9 | Visionado de analíticas | 15 | DVP5 | Android Studio, IntelIJ, Github, Jira |
| DVP10 | Calendario | 20 | | Android Studio, IntelIJ, Github, Jira |
| DVP11 | Chat | 25 | | Android Studio, IntelIJ, Github, Jira |
| T1 | Pruebas finales | 20 | DVP1 – DVP9 | Android Studio |
| | ***Total horas DVP*** | ***215*** | | |
| D1 | Documentación de la implementación | 60 | | Word |
| D2 | Documentación final | 20 | | Word |
| M1 | Lectura TFG | 25 | | Word, Power Point |
| **Total:** | | **570** | | |

## 6.5 Diagrama de Gantt

En esta sección, se presenta un resumen completo de toda la planificación explicada en las secciones anteriores. En concreto se puede ver en las ilustraciones 5, 6 y 7.

En la fase de desarrollo, una vez por cada sprint, se incluye una reunión con el director del TFG para comentar los avances realizados a lo largo de los 12 días. Además, la tarea GEP6 (gestión del backlog y tareas *agile*), está repartida entre los distintos *sprints* en los que será necesario de una gestión previa al inicio del desarrollo.

Como se puede ver, existen tareas que se extienden a lo largo del proyecto, esto representa que dicha tarea se irá realizando a medida que el proyecto vaya creciendo. Cabe recalcar que la estimación de fechas se ha realizado teniendo en cuenta que se va a dedicar una media de 6 horas cada día desde el momento que se termine el GEP.

A continuación, podemos ver la distribución de las tareas en el tiempo en formato texto, y más abajo, en las figuras 6 y 7 podemos ver el diagrama de Gantt completo. En cuanto al contenido de las iteraciones (*sprints*) al tratarse de un proyecto que sigue una metodología *agile* es posible que a lo largo del desarrollo se vaya ajustando el contenido y la duración de dichas iteraciones.

| | |
|---|---|
| **Gestión del proyecto** | |
| GEP1 - Contextualización y alcance | Feb 15 - Mar 1 |
| GEP2 - Planificación temporal | Mar 2 - 10 |
| GEP3 - Presupuesto y sostenibilidad | Mar 11 - 17 |
| GEP4 - Entrega final | Mar 18 - 24 |
| GEP5 - Reuniones con personal sanitar... | Apr 1 - Jun 4 |
| **Inception** | |
| IN1 - Stakeholders | Mar 22 - 23 |
| IN2 - Objetivos de los principales stake... | Mar 23 |
| IN3 - Requisitos funcionales iniciales | Mar 24 - 26 |
| IN4 - Requisitos no funcionales | Mar 27 - 28 |
| IN5 - Esquema conceptual inicial | Mar 29 - 31 |
| IN6 - Identificación y gestión de riesgos | Apr 1 - 3 |
| IN7 - Diseño de la arquitectura inicial | Apr 4 - 5 |
| IN8 - Diseño de mockups representativ... | Apr 6 |
| **Preparación del proyecto** | |
| PP1 - Aprendizaje de las tecnologías | Apr 7 - 10 |
| PP2 - Establecimiento del entorno | Apr 11 - 13 |
| **Sprint 1** | |
| Gestión del backlog y tareas agile | Apr 13 |
| DVP1- Implementación inicial de la API | Apr 14 - 15 |
| DVP2 - Inicio de sesión | Apr 16 - 19 |
| DVP3 - Pantalla principal | Apr 20 - 22 |
| Test | Apr 23 - 24 |
| **Sprint 2** | |
| Gestión del backlog y tareas agile | Apr 25 |
| DVP4 - Ajustes | Apr 26 - 29 |
| DVP5 - Notificaciones | Apr 30 - May 2 |
| DVP6 - Creación de las tarjetas para el ... | May 3 - 4 |
| Test | May 5 - 6 |
| **Sprint 3** | |
| DVP7 - Procesado de analíticas | May 7 - 14 |
| DVP8 - Visionado de la medicación | May 14 - 17 |
| Test | May 18 - 19 |
| **Sprint 4** | |
| Gestión del backlog y tareas agile | May 20 - 21 |
| DVP9 - Visionado de analíticas | May 22 - 24 |
| DVP10 - Calendario | May 25 - 28 |
| Test | May 29 - 30 |
| **Sprint 5** | |
| DVP11 - Chat | May 31 - Jun 3 |
| T1 - Testing final | Jun 4 - 11 |
| **Documentación y lectura** | |
| D1 - Documentación de la implementa... | Apr 12 - Jun 13 |
| M1 - Lectura TFG | Jun 11 - 25 |
| D2 - Documentación final | Jun 13 - 16 |

*Ilustración 5: Representación en texto del diagrama de Gantt. Fuente: Elaboración propia.*


*Ilustración 6: Diagrama de Gantt parte 1 (gestión del proyecto). Fuente: Elaboración propia.*

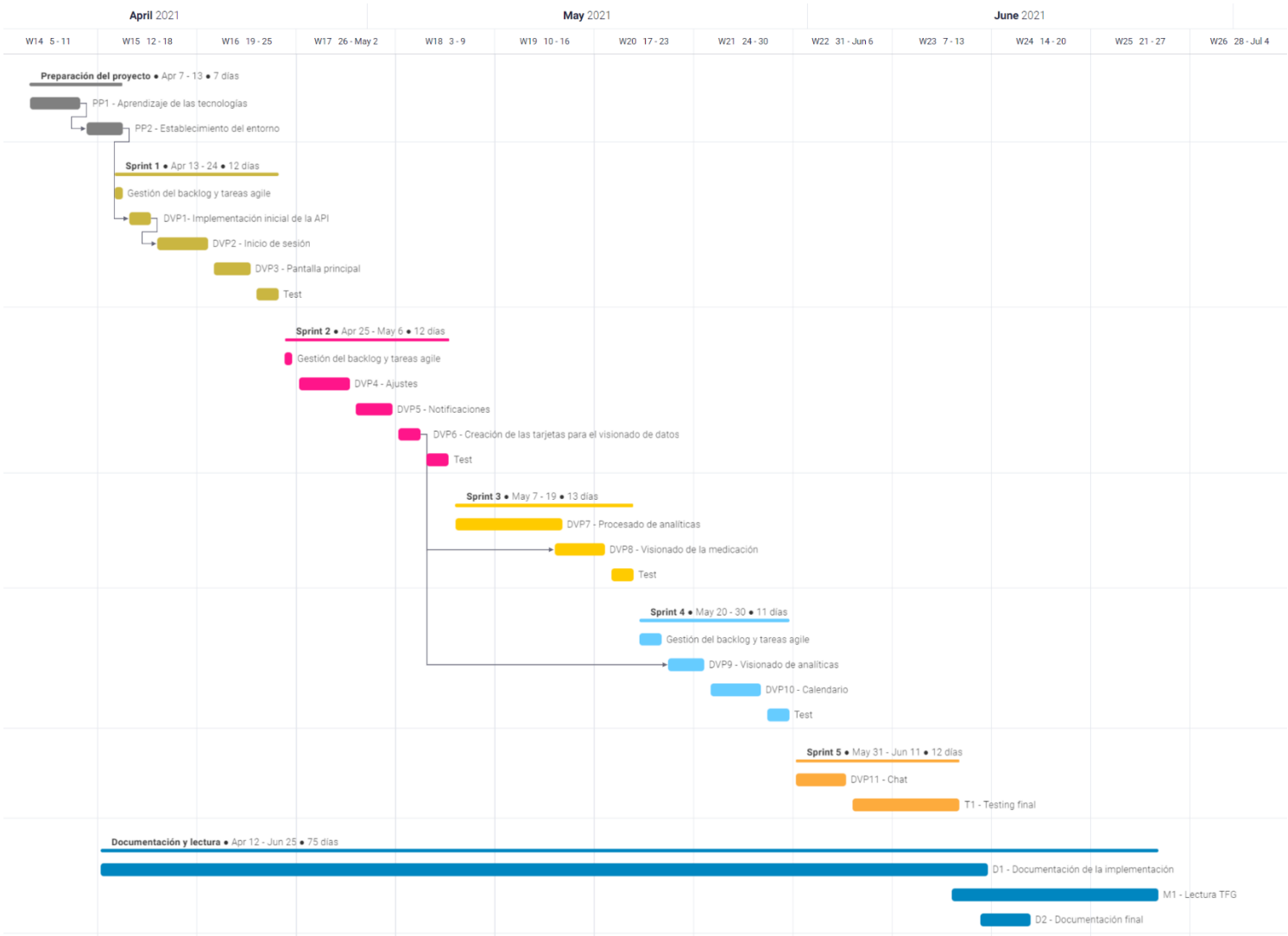



*Ilustración 7: Diagrama de Gantt parte 2 (desarrollo). Fuente: Elaboración propia.*

## 6.6 Viabilidad temporal del proyecto

A pesar de que el trabajo excede del número de horas estimadas para un trabajo de fin de grado, considero que dedicando cada día el número previsto que se ha comentado anteriormente (6 horas), será más que suficiente para poder completar el trabajo a tiempo. Además, se ha decidido realizar *sprints* de 12 días teniendo en cuenta la posibilidad de que alguna tarea se pueda alargar, pero, es posible que esto no pase y por lo tanto finalmente disponga de mayor tiempo para hacer pruebas funcionales sobre la aplicación.

Considero que para completar este trabajo de forma satisfactoria se han de realizar todas las tareas listadas anteriormente, es por eso por lo que no considero la opción de recortar el alcance del proyecto.

# 7. Gestión económica

## 7.1 Identificación de costes

Un proyecto de software, como cualquier otro proyecto, necesita de una gran cantidad de recursos tanto humanos como materiales para poder desarrollarse. Por ejemplo, bases de datos y servidores son claros ejemplos de materiales comúnmente utilizados en este tipo de proyectos. En cuanto a los recursos humanos, existen unos ciertos roles que son esenciales para el buen desarrollo del proyecto. En el caso de este proyecto, todos los roles los llevaré a cabo yo, pero lo que se presenta a continuación es un ejemplo de costes para el caso de que fuera un proyecto real que contara con distintos tipos de persona que llevaran a cabo las funciones de su rol en particular.

### 7.1.1 Costes humanos

En total, contaremos con un total de siete roles diferentes, cada uno tendrá su papel dentro del proyecto y será esencial para el buen funcionamiento de este. Estos son los siguientes:

*Tabla 3: Sueldos por rol. Fuente: Elaboración propia.*

| Rol | Abreviatura | Sueldo (€ por año) | Sueldo (€ por hora) |
|---|---|---|---|
| Product Manager | PM | 45000 | 23,43 |
| Ingeniero de Software Junior | ISJ | 24000 | 12,5 |
| Ingeniero de Software Senior | ISS | 50000 | 26 |
| Scrum master | SM | 43000 | 22,4 |
| UI Designer | UID | 40000 | 20,83 |
| Diseñador de bases de datos | DBD | 42000 | 21,88 |
| Q/A Tester | QAT | 24000 | 12,5 |

Todos los sueldos se han extraído de Glassdoor [26] y se han pasado de € por año, a € por hora dividiendo el total del sueldo entre 12 meses y teniendo en cuenta que cada mes tiene 4 semanas y por lo tanto 160 horas laborables (40h x 4 semanas). A partir de estos datos y teniendo en cuenta que cada tarea tiene un número de horas asignadas podemos calcular el presupuesto total del desarrollo del producto sin tener en cuenta los costes materiales. Podemos ver el cálculo en la tabla 3.

*Tabla 4: Presupuesto por rol. Fuente: Elaboración propia.*

| Tarea | Duración (horas) | PM (horas) | ISJ (horas) | ISS (horas) | UID (horas) | DBD (horas) | QAT (horas) | Coste (€) |
|---|---|---|---|---|---|---|---|---|
| GEP1 | 25 | 25 | | | | | | 585,75 |
| GEP2 | 15 | 15 | | | | | | 351,45 |
| GEP3 | 10 | 10 | | | | | | 243,3 |
| GEP4 | 20 | 20 | | | | | | 486,6 |
| GEP5 | 15 | 15 | | | | | | 351,45 |
| GEP6 | 25 | 25 | | | | | | 585,75 |
| IN1 | 10 | 10 | | | | | | 234,3 |
| IN2 | 5 | 5 | | | | | | 117,15 |
| IN3 | 20 | 20 | | | | | | 486,6 |
| IN4 | 10 | 10 | | | | | | 234,3 |
| IN5 | 15 | 15 | | | | | | 351,45 |
| IN6 | 20 | 20 | | | | | | 468,6 |
| IN7 | 10 | | | 5 | | 5 | | 239,4 |
| IN8 | 5 | | | | 5 | | | 104,15 |
| PP1 | 25 | | 25 | | | | | 312,5 |
| PP2 | 20 | | 10 | 10 | | | | 385 |
| DVP1 | 10 | | 6 | 4 | | | | 179 |
| DVP2 | 20 | | 12 | 8 | | | | 358 |
| DVP3 | 15 | | 9 | 6 | | | | 268,5 |
| DVP4 | 20 | | 12 | 8 | | | | 358 |
| DVP5 | 15 | | 9 | 6 | | | | 268,5 |
| DVP6 | 12 | | 7,2 | 4,8 | | | | 214,8 |
| DVP7 | 45 | | 15 | 10 | | | | 447,5 |
| DVP8 | 20 | | 12 | 8 | | | | 358 |
| DVP9 | 15 | | 9 | 6 | | | | 268,5 |
| DVP10 | 20 | | 12 | 8 | | | | 358 |
| DVP11 | 25 | | 15 | 10 | | | | 447,5 |
| T1 | 20 | | | | | | 20 | 250 |
| D1 | 60 | 20 | | 20 | | 10 | | 1207,4 |
| D2 | 20 | 20 | | | | | | 486,6 |
| | | | | | | | Total: | 10916,15 |

Toda la parte de gestión del proyecto y documentación recae principalmente en el *product manager*. En cuanto a la fase de desarrollo, entran en juego los ingenieros de software, tanto el senior como el junior. Como podemos observar, el junior realiza más horas que el senior, esto es debido a que, para realizar la misma faena, un ingeniero junior tarda siempre más que un senior debido a su falta de experiencia. Finalmente, existen tareas que solo son realizadas por roles concretos como puede ser el *testing* o el diseño de la interfaz.

### 7.1.2 Costes materiales

Los costes materiales son aquellos que están vinculados a algún bien material, en el caso que nos incumbe estos costes son los siguientes:

*Tabla 5: Costes de material. Fuente: Elaboración propia.*

| Material | Coste (€) | Vida útil (años) | Amortización (€) | Cantidad | Total (€) |
|---|---|---|---|---|---|
| Dell OptiPlex 3080 MFF [27] (Ordenador sobremesa) | 509 | 5 | 43,95 | 3 | 131,85 |
| Monitor Dell 27: E2720HS [28] | 153 | 5 | 13,21 | 3 | 39,63 |

En total hará falta un total de tres ordenadores de sobremesa y tres monitores, ya que el máximo de personas que estarán trabajando a la vez serán tres.

El ordenador y el monitor que se han escogido son los que he considerado que proporcionarán un rendimiento adecuado para la elaboración del proyecto. La fórmula que se ha utilizado para el cálculo de la amortización es la siguiente:

$$\frac{\text{Coste del material (€)}}{\text{Vida útil (años)} * 220\ días * 6\ horas/día} * 570\ horas$$

- 220 días → Total de días trabajables que tiene un año aproximadamente.
- 6 horas/día → Horas de dedicación diaria.
- 570 horas → Total de horas del proyecto.

Como se puede ver, no se incluyen ni gastos de servidor ni de bases de datos, el motivo es que se utilizarán servicios gratuitos como PostgreSQL y Amazon Web Services.

### 7.1.3 Costes genéricos

Otro coste importante en el desarrollo de un proyecto es el de la oficina, en nuestro caso, al tratarse de un proyecto pequeño no necesitaremos una oficina grande, pero necesitaremos contar con electricidad, agua, internet de alta velocidad, servicio de limpieza, calefacción, aire acondicionado, impresora, sala de reuniones y una sala de trabajo.

*Tabla 6: Costes de oficina. Fuente: Elaboración propia.*

| | Meses | Precio (€/mes) | Coste (€) |
|---|---|---|---|
| Oficina con gastos incluidos | 4 | 700 | 2800 |

### 7.1.4 Costes de contingencia

El presupuesto asignado para contingencias será un 10% del total de costes del proyecto, como podemos ver en la tabla siguiente:

*Tabla 7: Coste asignado a contingencias. Fuente: Elaboración propia.*

| | Costes totales (€) | % | Coste (€) |
|---|---|---|---|
| Contingencia | 13800 | 10 | 1380 |

### 7.1.5 Costes de imprevistos

Finalmente tenemos los imprevistos, como se vio en la gestión de riesgos existe una cierta probabilidad de que sucedan incidentes que afecten a la planificación y por lo tanto al presupuesto. En la siguiente tabla se detallan los posibles incidentes que podrían llegar a suceder:

*Tabla 8: Costes de los posibles incidentes. Fuente: Elaboración propia.*

| Incidente | Coste (€) | % | Coste total (€) |
|---|---|---|---|
| Bugs y errores que impliquen el retraso en la implementación 1 semana | 1092 | 20 | 218 |
| Bugs y errores que impliquen el retraso en la implementación 2 semanas | 2184 | 10 | 218 |
| Falta de seguridad en el sistema | 1500 | 20 | 300 |
| Imprecisión en las respuestas que impliquen el retraso en la implementación 1 semana. | 1092 | 30 | 328 |
| Dificultad de uso que implique el retraso en la implementación 1 semana. | 1092 | 10 | 109 |
| Migración de la API. | 273 | 1 | 3 |

En la tabla número 8 no se incluyen todos los riesgos que se detallaron en la sección 5, solo se incluyen aquellos que puedan afectar de forma directa al presupuesto del proyecto.

Como se puede ver, solo se incluyen los casos en los que el retraso pueda llegar a una semana, sin embargo, antes de llegar a este punto se tratará de abordar dicho problema con los métodos sugeridos en la gestión de riesgos para tratar de evitar de llegar al punto en el que se tenga que retrasar la implementación una semana completa.

### 7.1.6 Coste total del proyecto

La tabla siguiente muestra un resumen del coste total del proyecto. Como se puede ver, se muestran diferentes costes totales dependiendo de si tenemos en cuenta el presupuesto con contingencia, o con contingencias más imprevistos. En caso de que este presupuesto extra no hiciera falta, sería destinado a la continuación y expansión de la aplicación, añadiendo funcionalidades extra y mejorándola para obtener mejor experiencia de usuario.

*Tabla 9: Resumen de los costes. Fuente: Elaboración propia.*

| Tipo de coste | Coste (€) |
|---|---|
| Costes humanos | 10916 |
| Costes de material | 84 |
| Costes de oficina | 2800 |
| **Total: 13800€** | |
| Contingencia | 1380 |
| **Total con contingencia: 15180€** | |
| Retraso una semana | 218 |
| Retraso dos semanas | 218 |
| Falta de seguridad en el sistema | 300 |
| Imprecisión en las respuestas que impliquen el retraso en la implementación 1 semana. | 328 |
| Dificultad de uso que implique el retraso en la implementación 1 semana. | 109 |
| Migración de la API. | 3 |
| **Total con imprevistos y contingencia: 16356€** | |

## 7.2 Control de gestión

Para poder comprobar periódicamente las posibles desviaciones de presupuesto que se puedan dar durante el transcurso del proyecto se utilizarán las fórmulas siguientes las cuales estarán automatizadas en una hoja de Excel para tener una mayor rapidez a la hora de realizar cualquier consulta:

- Desviación total de los costes: Esto nos servirá para conocer la desviación del coste presupuestado frente al que hemos utilizado para el proyecto. Y se calcula restando el presupuesto estimado con el coste real →
  DTC = Coste presupuestado – Coste real
- Desviación total de horas: Esto nos servirá para conocer la desviación de las horas que hemos dedicado al proyecto con las planificadas. Se calcula restando las horas calculadas menos las horas reales de proyecto →
  DTH = Horas calculadas – Horas reales

Estas dos fórmulas utilizadas para calcular la desviación total también se podrán utilizar para conocer el desvío del coste y/o horas, por tarea, es decir se podrá realizar el mismo cálculo, pero por tarea en vez de en total. Además, las podemos aplicar para cualquier tipo de coste, como pueden ser los materiales, oficina, imprevistos o contingencias, simplemente realizando la diferencia del coste presupuestado con el coste real.

Finalmente, para los incidentes, se mantendrá una lista actualizada por cada incidente que vaya ocurriendo y el coste que suponga. Haciendo esto podremos consultar de forma rápida el coste total que nos han supuesto.

A pesar de que existen muchas otras fórmulas que nos ayudarían con la gestión del proyecto, considero que con todo lo anterior será suficiente para conocer la desviación o el estado actual del presupuesto del proyecto. El proyecto tiene un presupuesto muy detallado y es por eso por lo que considero que no es necesario disponer de métodos de gestión más avanzados.

# 8. Especificación de requisitos

Antes de comenzar con la implementación del sistema se requiere de otro paso más, la especificación de requisitos. En este apartado se van a detallar todos los requisitos que el sistema va a tener que cumplir. Es muy importante que esto suceda, pues el éxito del proyecto depende en parte de la buena especificación de requisitos. Además, se incluye el esquema conceptual en UML, el cual nos va a dar una visión global de las clases que tendrá el sistema.

## 8.1 Stakeholders

En este apartado se verá cuáles son las partes interesadas en este proyecto, es decir, las partes que se verán afectadas directa o indirectamente del sistema software que se propone. Además, se van a comentar los objetivos concretos que tendrá cada parte interesada por separado y cuál será su rol dentro del sistema.

### 8.1.1 Médicos

Al recibir los pacientes la medicación de forma automática en sus dispositivos, los médicos a cargo de dichos pacientes ya no tendrán que encargarse de realizar este trabajo. En consecuencia, podrán emplear esas horas anteriormente dedicadas a estas visitas en otras cosas. Es por esto por lo que la implantación del sistema afecta de forma directa al colectivo médico.

Además, tendrán que encargarse de supervisar que los resultados generados por el sistema sean correctos, por lo tanto, también se encargarán de la parte de validación de los resultados generados por el sistema.

### 8.1.2 Enfermeras

Las enfermeras son las que realizan los controles rutinarios a los pacientes crónicos, además, en muchos casos, existen protocolos que permiten a las enfermeras modificar directamente la medicación de los pacientes. Es por eso por lo que la implantación del sistema también ahorraría tiempo a este colectivo.

Además, el chat integrado en la aplicación establecerá un contacto directo entre el paciente y el departamento de enfermería. Por lo tanto, habrá personas dentro de este colectivo que se tendrán que encargar de resolver las consultas recibidas a través del chat.

### 8.1.3 Documentalistas sanitarios

Actualmente, la mayor parte de las tareas de índole administrativo que se llevan a cabo en un centro de salud requieren de la presencialidad del paciente, o bien de realizar una llamada telefónica. El sistema propuesto también incluirá unas secciones dedicadas a la gestión sanitaria, como puede ser la consulta y reclamación de citas y la consulta de medicación. Gracias a esto, se evitarán visitas y llamadas telefónicas simplemente para realizar consultas sencillas que la aplicación podrá responder.

### 8.1.4 Centros de salud

Estos centros se verán altamente beneficiados ya que se podrán ahorrar todas las citas de pacientes crónicos que no requieran de presencialidad, logrando así una mayor optimización de las citas que se conciertan en dichos centros de salud.

### 8.1.5 Pacientes crónicos

Los pacientes son otra de las partes interesadas que más se beneficiarán del proyecto, evitando tener que visitar su centro de salud para que el doctor le modifique la medicación, consiguiendo una mayor independencia. Además, contarán con un chat para poder resolver sus dudas y con distintos apartados que les ayudarán con la gestión de su enfermedad.

### 8.1.6 Personas a cargo de los pacientes

Muchos pacientes crónicos son personas mayores o personas con un alto grado de dependencia, estas personas dependen de sus familiares o de cuidadores para asistir a sus visitas médicas. Son estas personas las que gestionan la medicación del paciente, y es por eso por lo que son también una parte interesada en este proyecto.

### 8.1.7 Laboratorios clínicos

Las personas responsables de introducir los resultados de los análisis en el sistema serán trabajadores de los laboratorios que se encargan de realizar los estudios sobre las muestras de sangre. Su trabajo se verá afectado directamente por la implantación del sistema.

### 8.1.8 Gobierno o grupos hospitalarios privados

Con la integración del sistema se ahorrarían muchas visitas de pacientes crónicos, lo que permitiría la realización de más visitas de otro tipo. Esto ayudaría a reducir la saturación del sistema sanitario público, o, en caso de implementarse en un sistema privado, la realización de más visitas, y, por lo tanto, más beneficios.

### 8.1.9 Desarrollador del proyecto

El desarrollador principal del proyecto es la principal parte interesada, puesto que se trata de un TFG y por lo tanto se verá muy beneficiado si el proyecto sale bien. El autor será quien se encargue de documentar y desarrollar todas las partes del proyecto y ejercerá de todos los roles que existen dentro de un equipo de desarrollo de software: arquitecto de software, *tester*, desarrollador, diseñador, jefe de proyecto y *scrum master* entre otros.

### 8.1.10 Director del proyecto

El director de proyecto, Xavier Franch Gutiérrez, será quien supervise el proyecto y aconseje al desarrollador para la correcta realización de todas las partes del proyecto.

## 8.2 Requisitos funcionales

Los requisitos funcionales son las descripciones de las funcionalidades que va a tener un sistema software. En esta sección se van a detallar todas las funcionalidades del sistema que se propone y su especificación completa.

### 8.2.1 Diagrama de casos de uso

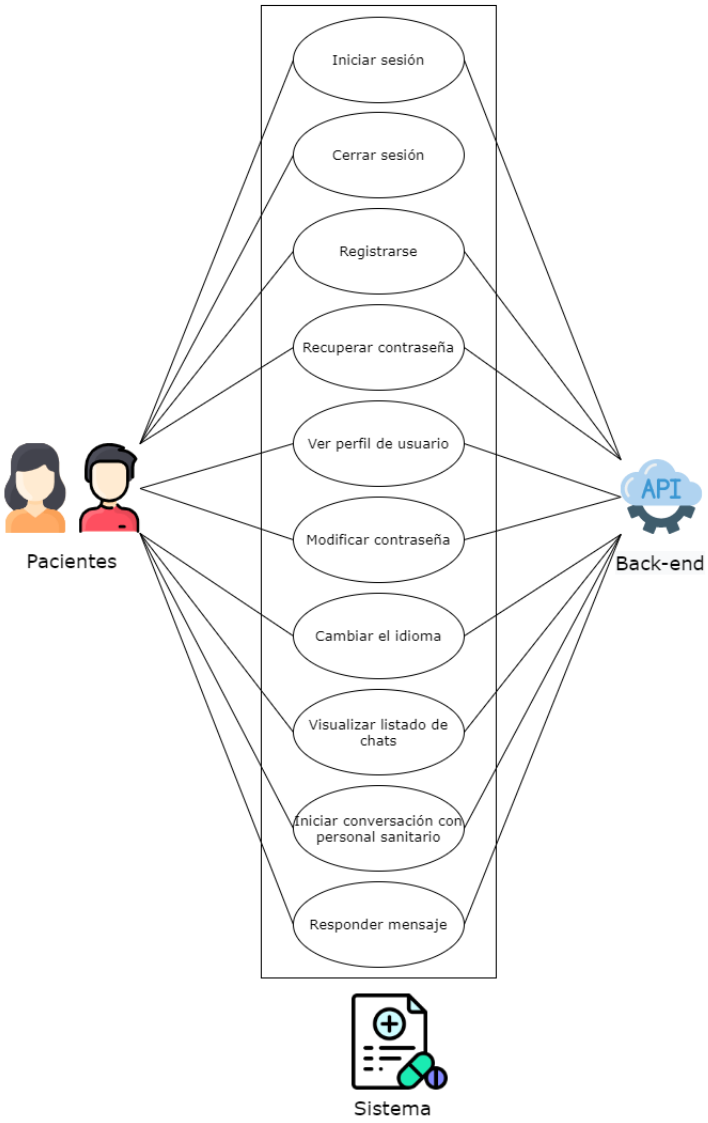



*Ilustración 8: Casos de uso del paciente parte 1. Fuente: Elaboración propia.*

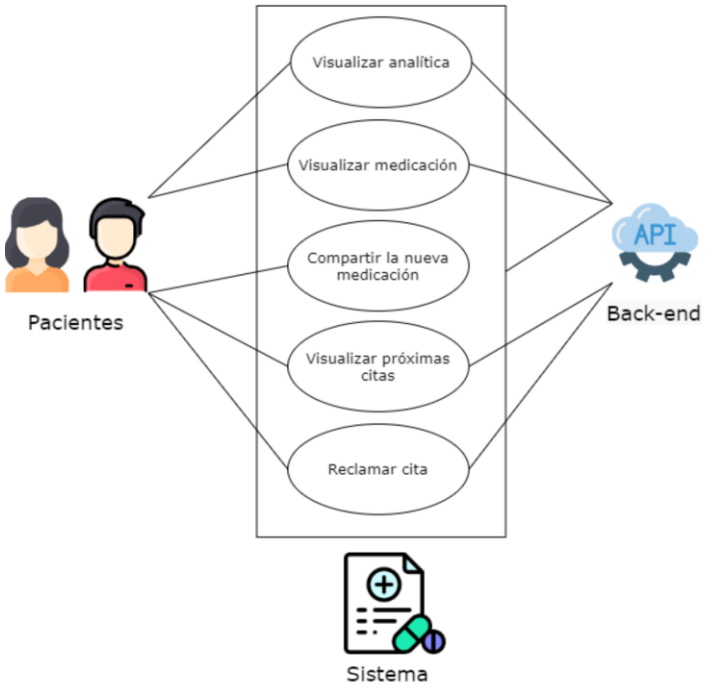



*Ilustración 9: Casos de uso del paciente parte 2. Fuente: Elaboración propia.*

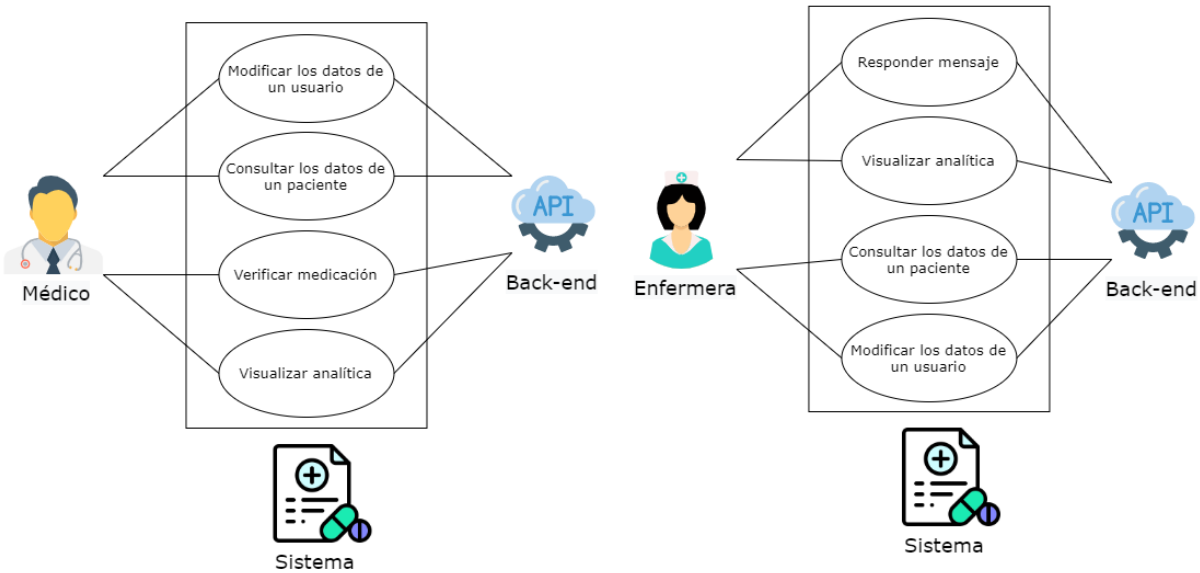



*Ilustración 10: Casos de uso del médico y la enfermera. Fuente: Elaboración propia.*

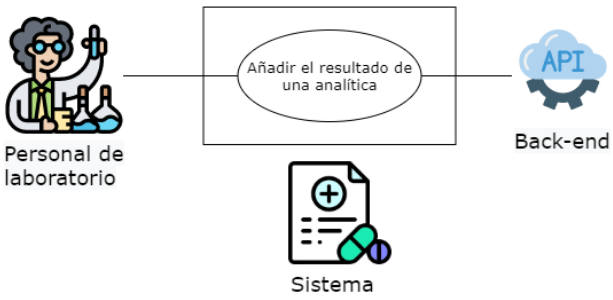



*Ilustración 11: Caso de uso del personal de laboratorio. Fuente: Elaboración propia.*

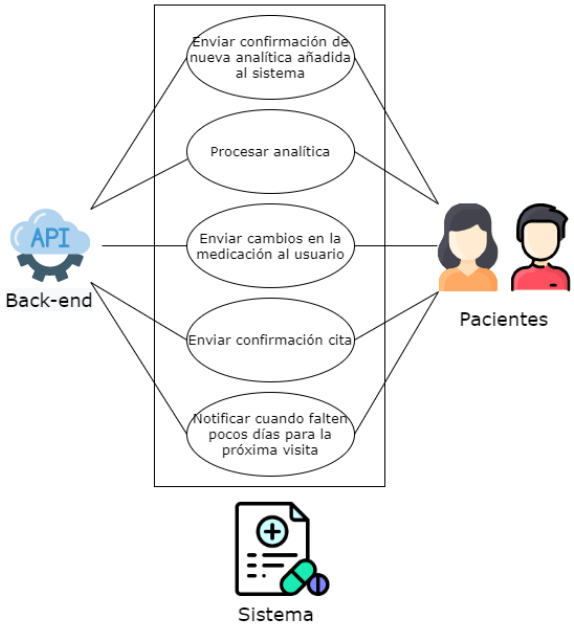



*Ilustración 12: Casos de uso del back-end. Fuente: Elaboración propia.*

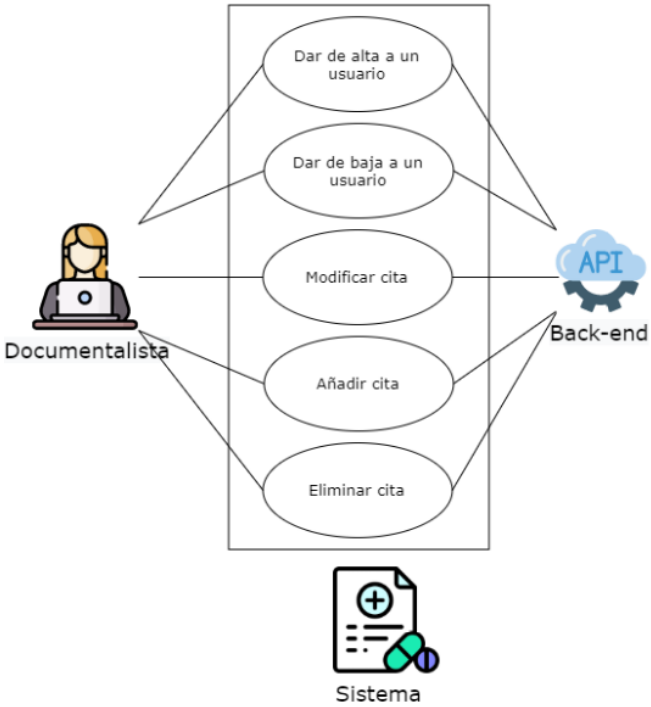



*Ilustración 13: Casos de uso del documentalista sanitario. Fuente: Elaboración propia.*

### 8.2.2 Especificación completa de los casos de uso e historias de usuario

Una vez vistos los diagramas de casos de uso, se van a volver a listar todos y se van a detallar. En las tablas siguientes podemos ver la descripción completa de cada caso de uso, así como la historia de usuario asociada a dicho caso.

| **#1 Dar de alta a un usuario** | |
|---|---|
| **Actor principal** | Documentalista sanitario. |
| **Precondiciones** | - |
| **Disparador** | Se tiene que añadir un nuevo usuario para que pueda utilizar el sistema. |
| **Historia de usuario** | **Como** documentalista sanitario **quiero** dar de alta a nuevos usuarios en el sistema **para** que puedan empezar a utilizarlo. |
| **Descripción** | El administrador del sistema da de alta a un nuevo usuario para que pueda utilizar el sistema, ya sea personal sanitario o paciente. |
| **Escenario principal** | 1. Se da la orden para poder dar de alta al paciente.<br>2. Se introducen los datos del paciente en el sistema.<br>3. Se confirma el alta. |
| **Extensión 1: El usuario ya existe** | 3. El sistema encuentra un usuario con los mismos datos que los introducidos.<br>4. El sistema informa al usuario de que el usuario ya existe. |

| #2 Dar de baja a un usuario | |
|---|---|
| Actor principal | Documentalista sanitario. |
| Precondiciones | El usuario que se quiere dar de baja tiene que estar registrado en el sistema. |
| Disparador | Se tiene que dar de baja a un usuario por la razón que sea. |
| Historia de usuario | **Como** documentalista sanitario **quiero** dar de baja a usuarios en el sistema **para** que dejen de tener acceso al mismo. |
| Descripción | El administrador del sistema da de baja a un usuario. |
| Escenario principal | 1. Se da la orden para dar de baja a un usuario.<br>2. Se introducen los datos del paciente en el sistema.<br>3. Se confirma la baja. |
| Extensión 1: Usuario no encontrado | 3. El sistema no encuentra a ningún usuario con ese DNI.<br>4. El sistema avisa al usuario que no existe ningún paciente registrado en el sistema con ese DNI.<br>5. El sistema recomienda al usuario que compruebe que los datos introducidos son correctos. |

| #3 Consultar datos de un paciente | |
|---|---|
| Actor principal | Médico. |
| Precondiciones | El usuario tiene que estar dado de alta y registrado en el sistema. |
| Disparador | Se realiza una consulta para obtener los datos del paciente. |
| Historia de usuario | **Como** médico **quiero** poder consultar los datos de mi paciente **para** verificar que han sido introducidos correctamente **o** realizar cualquier tipo de consulta sobre el paciente. |
| Descripción | El médico del paciente consulta los datos de un paciente para verificar si los datos son correctos o para realizar alguna consulta. |
| Escenario principal | 1. Se introduce el DNI del paciente en el buscador.<br>2. Se obtienen todos los datos del paciente.<br>3. Se muestran los datos al usuario. |
| Extensión 1: Usuario no encontrado | 4. El sistema no encuentra a ningún usuario con ese DNI.<br>5. El sistema avisa al usuario que no existe ningún paciente registrado en el sistema con ese DNI.<br>6. El sistema recomienda al usuario que compruebe que los datos introducidos son correctos. |

| #4 Registrarse | |
|---|---|
| Actor principal | Paciente. |
| Precondiciones | El paciente tiene que estar dado de alta en el sistema. |
| Disparador | Se solicita el registro por parte del paciente. |
| Historia de usuario | **Como** paciente **quiero** poder registrarme **para** poder hacer uso del sistema. |
| Descripción | El usuario se registra con sus datos personales. |
| Escenario principal | 1. El usuario pulsa sobre la opción de registrarse.<br>2. El usuario cumplimenta el formulario de registro.<br>3. El usuario confirma el registro pulsando en "registrarse".<br>4. La aplicación envía los datos a la API.<br>5. El usuario navega a la pantalla principal. |
| Extensión 1: Usuario ya registrado | 5. La API devuelve un error.<br>6. El sistema indica al usuario que ya está registrado y le sugiere que inicie sesión. |
| Extensión 2: DNI incorrecto | 4. Se informa al usuario de que el DNI introducido no es correcto. |
| Extensión 3: Las contraseñas no coinciden. | 4. Se informa al usuario de que las contraseñas no coinciden. |
| Extensión 4: Fecha inválida | 4. Se informa al usuario de que la fecha de nacimiento que ha introducido no es válida. |
| Extensión 5: Métodos de contacto vacíos | 4. Se informa al usuario de que tiene que añadir al menos un método de contacto. |
| Extensión 6: Campo obligatorio vacío | 4. Se informa al usuario que ha dejado un campo obligatorio vacío. Estos pueden ser: nombre, contraseña, DNI, primer apellido o fecha de nacimiento. |

| #5 Iniciar sesión | |
|---|---|
| Actor principal | Paciente. |
| Precondiciones | El paciente tiene que estar registrado en el sistema. |
| Disparador | El paciente introduce sus datos de inicio de sesión y solicita entrar. |
| Historia de usuario | **Como** paciente **quiero** iniciar sesión en el sistema **para** poder tener acceso al contenido de la aplicación. |
| Descripción | El usuario inicia sesión en el sistema con sus credenciales |
| Escenario principal | 1. El usuario introduce los datos de inicio de sesión.<br>2. La aplicación comprueba que no hay ningún error en los campos.<br>3. Se envían los datos a la API.<br>4. La API comprueba que los datos de inicio de sesión son correctos.<br>5. La API devuelve la información del usuario.<br>6. El usuario navega hacia la pantalla principal de la aplicación. |
| Extensión 1: Campo obligatorio vacío | 3. La aplicación detecta que existe un campo vació.<br>4. Se informa al usuario de que tiene que rellenar todos los campos. |
| Extensión 2: El usuario no existe | 5. La API devuelve un error.<br>6. Se le indica al usuario que no existe ningún usuario dado de alta con ese DNI. |

| | |
|---|---|
| Extensión 3: Fallo de inicio de sesión | 5. La API devuelve un error.<br>6. Se le indica al usuario que ha habido un error en el inicio de sesión, sin dar más pistas para evitar fugas de seguridad. |

| #6 Cerrar sesión | |
|---|---|
| Actor principal | Paciente. |
| Precondiciones | El paciente debe tener la sesión iniciada. |
| Disparador | El paciente solicita el cierre de sesión. |
| Historia de usuario | **Como** paciente **quiero** cerrar sesión **para** poder entrar al sistema con otra cuenta, o simplemente dejar de utilizarla. |
| Descripción | El usuario cierra sesión en el sistema. |
| Escenario principal | 1. El usuario pulsa sobre el botón de cerrar sesión.<br>2. Se navega hasta la pantalla de inicio de sesión. |

| #7 Modificar los datos de un usuario | |
|---|---|
| Actor principal | Médico. |
| Precondiciones | El usuario tiene que estar dado de alta en el sistema. |
| Disparador | El doctor responsable del paciente realiza la modificación de los datos. |
| Historia de usuario | **Como** médico **quiero** modificar los datos de mi paciente **para** poder corregir datos mal introducidos o añadir de nuevos. |
| Descripción | El personal sanitario responsable del paciente modifica los datos de un paciente. |
| Escenario principal | 1. El usuario introduce el DNI del paciente al que le quiere modificar los datos.<br>2. El sistema verifica que el paciente existe.<br>3. El sistema retorna un formulario con los datos del paciente.<br>4. El usuario introduce los datos modificados y los envía.<br>5. El sistema recibe los datos y los almacena. |
| Extensión 1: Usuario no encontrado. | 3. El sistema no encuentra a ningún usuario con ese DNI.<br>4. El sistema avisa al usuario que no existe ningún paciente registrado en el sistema con ese DNI.<br>5. El sistema recomienda al usuario que compruebe que los datos introducidos son correctos. |

| #8 Ver el perfil de un usuario | |
|---|---|
| Actor principal | Paciente. |
| Precondiciones | El usuario debe tener la sesión iniciada. |
| Disparador | El usuario quiere consultar su perfil. |
| Historia de usuario | **Como** paciente **quiero** ver mi perfil **para** poder consultar mis datos. |
| Descripción | El usuario podrá visualizar todos los datos en su perfil. Los datos a mostrar serán solo los de contacto. |
| Escenario principal | 1. El usuario pulsa sobre el botón de ajustes.<br>2. Se navega hasta la ventana de ajustes que contiene los datos del perfil del usuario. |

| #9 Modificar contraseña | |
|---|---|
| Actor principal | Paciente. |
| Precondiciones | El usuario debe de tener la sesión iniciada. |
| Disparador | El usuario quiere cambiar la contraseña. |
| Historia de usuario | **Como** paciente **quiero** cambiar mi contraseña **para** poder acceder al sistema con otra contraseña completamente diferente a la introducida en el registro. |
| Descripción | El usuario podrá modificar su contraseña introduciendo su contraseña actual y dos veces la nueva. |
| Escenario principal | 1. El usuario introduce su contraseña actual, la nueva, y una confirmación de la nueva.<br>2. La aplicación verifica que los datos introducidos son correctos.<br>3. Se envía la información a la API.<br>4. La API verifica que la contraseña introducida se corresponde con el usuario.<br>5. Se guarda la contraseña modificada. |
| Extensión 1: Las contraseñas no coinciden. | 3. El sistema indica al usuario que la nueva contraseña y su confirmación no son equivalentes. |
| Extensión 2: La contraseña introducida no se corresponde con la del usuario | 5. La API devuelve un error.<br>6. El sistema informa al usuario de que ha habido un error al tratar de modificar su contraseña. |

| #10 Cambiar el idioma | |
|---|---|
| Actor principal | Paciente. |
| Precondiciones | El usuario debe de tener la sesión iniciada. |
| Disparador | El usuario quiere cambiar el idioma de la aplicación. |
| Historia de usuario | **Como** paciente **quiero** modificar el idioma en el que se muestran los textos de la aplicación **para** poder tener la aplicación en el idioma de mi preferencia. |
| Descripción | El usuario podrá escoger el idioma con el que desee hacer uso de la aplicación. |
| Escenario principal | 1. El usuario modifica el idioma por defecto de la aplicación.<br>2. El sistema envía la modificación de idioma a la API.<br>3. Se tiene que recargar la aplicación para mostrar los datos en el idioma elegido. |

| #11 Añadir el resultado de una analítica | |
|---|---|
| Actor principal | Personal de laboratorio. |
| Precondiciones | El usuario tiene que estar dado de alta en el sistema. |
| Disparador | El trabajador del laboratorio quiere añadir una analítica al sistema. |
| Historia de usuario | **Como** trabajador/a del laboratorio **quiero** añadir analíticas al sistema **para** que este pueda procesarlas y entregar los resultados al paciente. |
| Descripción | Se tiene que poder añadir una analítica al sistema una vez el laboratorio genera el informe pertinente. |
| Escenario principal | 1. El trabajador del laboratorio introduce los resultados de la analítica en el sistema.<br>2. El sistema almacena la analítica en base de datos.<br>3. Se recibe la confirmación de que se ha introducido correctamente. |
| Extensión 1: Usuario no encontrado. | 2. El sistema no encuentra a ningún usuario con ese DNI.<br>3. El sistema avisa al usuario que no existe ningún paciente registrado en el sistema con ese DNI.<br>4. El sistema recomienda al usuario que compruebe que los datos introducidos son correctos. |

| #12 Enviar confirmación de nueva analítica añadida al sistema | |
|---|---|
| Actor principal | Sistema. |
| Precondiciones | • Existe la nueva analítica.<br>• El paciente de la analítica está dado de alta. |
| Disparador | Se ha añadido una nueva analítica al sistema. |
| Historia de usuario | **Como** sistema **quiero** enviar una notificación al usuario **para** que sepa que tiene disponible una nueva analítica a consultar. |
| Descripción | Cuando se sube una analítica al sistema, se notifica al usuario para que pueda verla en el apartado correspondiente de la aplicación. |
| Escenario principal | 1. La API envía un mensaje a la App.<br>2. La App recibe el mensaje.<br>3. La App procesa el mensaje y le envía una notificación al usuario. |

| #13 Procesar el resultado de una analítica | |
|---|---|
| Actor principal | Sistema. |
| Precondiciones | • Existe la nueva analítica.<br>• El paciente de la analítica está dado de alta. |
| Disparador | Se añade una analítica al sistema. |
| Historia de usuario | **Como** sistema **quiero** procesar las analíticas **para** poder generar los cambios en la medicación del paciente. |
| Descripción | Cuando se introducen los datos de una analítica en el sistema, este los procesa para extraer los posibles cambios en la medicación del paciente. |
| Escenario principal | 1. El sistema recibe una analítica.<br>2. El sistema procesa los datos extraídos de dicha analítica.<br>3. El sistema genera una respuesta en forma de cambios en la medicación del paciente. |
| Extensión 1: La analítica es correcta | 3. El sistema no genera cambios en la medicación del paciente ya que los resultados obtenidos en la analítica son buenos. |

| #14 Enviar cambios en la medicación al paciente | |
|---|---|
| Actor principal | Sistema |
| Precondiciones | • Existen cambios en la medicación.<br>• El paciente receptor de los cambios está dado de alta. |
| Disparador | Se producen cambios en la medicación del paciente. |
| Historia de usuario | **Como** sistema **quiero** notificar al paciente **para** que pueda conocer el estado de su medicación en cuanto se modifica. |
| Descripción | Cuando se termina el procesado de la analítica, se notifica al usuario con la nueva medicación que debe de tomar. |
| Escenario principal | 1. La API envía un mensaje a la App.<br>2. La App recibe el mensaje.<br>3. La App procesa el mensaje y le envía una notificación al usuario. |

| #15 Visualizar analítica | |
|---|---|
| Actor principal | Paciente. |
| Precondiciones | • El paciente debe tener analíticas disponibles.<br>• El paciente debe tener la sesión iniciada. |
| Disparador | El paciente quiere ver sus analíticas. |
| Historia de usuario | **Como** paciente **quiero** visualizar los resultados de mi analítica **para** ser conocedor del estado actual de mi enfermedad. |
| Descripción | El usuario tiene que poder acceder a la analítica completa desde la aplicación. |
| Escenario principal | 1. El usuario entra en la sección de analíticas.<br>2. La aplicación solicita los datos de las analíticas del paciente a la API.<br>3. La API le envía todas las analíticas que tiene el paciente.<br>4. La app las recoge y las muestra al usuario en formato tarjeta.<br>5. El usuario pulsa sobre una de las tarjetas.<br>6. Se abre un diálogo con toda la información de la analítica. |

| #16 Visualizar medicación | |
|---|---|
| Actor principal | Paciente. |
| Precondiciones | • El paciente debe tener medicación disponible.<br>• El paciente debe tener la sesión iniciada. |
| Disparador | El paciente quiere consultar su medicación. |
| Historia de usuario | **Como** paciente **quiero** ver mi medicación actual **para** saber cuál tengo que tomar en cada momento. |
| Descripción | El usuario puede ver su medicación completa organizada por enfermedad. |
| Escenario principal | 1. El usuario entra en la sección de medicación.<br>2. La aplicación solicita los datos de las medicaciones del paciente a la API.<br>3. La API le envía todas las medicaciones que tiene el paciente.<br>4. La aplicación las recoge y las muestra al usuario en formato tarjeta. |

| #17 Verificar medicación | |
|---|---|
| Actor principal | Médico. |
| Precondiciones | • El paciente tiene que estar dado de alta en el sistema.<br>• El paciente tiene medicación asignada para tratar su enfermedad.<br>• El médico tiene acceso al sistema. |
| Disparador | El médico del paciente quiere verificar la medicación actual. |
| Historia de usuario | **Como** médico **quiero** verificar la medicación de mi paciente **para** comprobar que es correcta y que el paciente está tomando lo que debe de tomar para tratar su enfermedad. |
| Descripción | El doctor del paciente podrá verificar que los resultados generados por el sistema son correctos. |
| Escenario principal | 1. El médico busca en el sistema a un usuario en particular por su DNI.<br>2. El sistema le retorna los datos del usuario.<br>3. El médico revisa los datos, entre ellos la medicación. |
| Extensión 1: Usuario no encontrado. | 2. El sistema no encuentra a ningún usuario con ese DNI.<br>3. El sistema avisa al usuario que no existe ningún paciente registrado en el sistema con ese DNI.<br>4. El sistema recomienda al usuario que compruebe que los datos introducidos son correctos. |

| #18 Compartir medicación | |
|---|---|
| Actor principal | Paciente. |
| Precondiciones | • El paciente debe tener medicación disponible.<br>• El paciente debe tener la sesión iniciada. |
| Disparador | El paciente quiere compartir su medicación. |
| Historia de usuario | **Como** paciente **quiero** compartir mi medicación actual **para** poder hacer conocedor/a de la misma a otra persona de forma rápida y sencilla. |
| Descripción | El usuario podrá compartir su medicación para facilitar la distribución de los cambios entre sus familiares o personas a cargo. |
| Escenario principal | 1. El usuario, desde la pantalla de medicación, pulsa sobre el botón de compartir que tienen las tarjetas.<br>2. El usuario escoge por donde quiere compartir esta información.<br>3. El sistema recoge los datos de la analítica y los comparte. |

| #19 Visualizar próximas citas | |
|---|---|
| Actor principal | Paciente. |
| Precondiciones | • El paciente tiene próximas citas.<br>• El paciente está dado de alta. |
| Disparador | El paciente quiere ver sus próximas citas. |
| Historia de usuario | **Como** paciente **quiero** ver mis próximas citas **para** conocer cuándo y dónde tengo que asistir a mi próxima cita médica. |
| Descripción | El usuario podrá ver todas sus próximas citas de control. |
| Escenario principal | 1. El usuario entra en la sección de calendario.<br>2. La aplicación solicita los datos de las citas del paciente a la API.<br>3. La API le envía todas las citas que tiene el paciente.<br>4. La aplicación las recoge y las muestra al usuario en formato tarjeta. |

| #20 Añadir nueva cita | |
|---|---|
| Actor principal | Documentalista sanitario. |
| Precondiciones | • El paciente está dado de alta en el sistema.<br>• El documentalista sanitario tiene acceso al sistema. |
| Disparador | Se solicita una nueva cita médica para el control de la enfermedad del paciente. |
| Historia de usuario | **Como** documentalista sanitario **quiero** añadir una nueva cita **para** que el paciente la pueda visualizar en su dispositivo. |
| Descripción | El personal responsable podrá añadir una cita al usuario. |
| Escenario principal | 1. El documentalista introduce los datos identificadores del paciente.<br>2. El documentalista escoge un día y una hora y añade la cita.<br>3. El sistema procesa la cita.<br>4. El sistema notifica al usuario que tiene una nueva cita disponible para consultar. |

| | |
|---|---|
| Extensión 1: Usuario no encontrado. | 2. El sistema no encuentra a ningún usuario con ese DNI.<br>3. El sistema avisa al usuario que no existe ningún paciente registrado en el sistema con ese DNI.<br>4. El sistema recomienda al usuario que compruebe que los datos introducidos son correctos. |

| #21 Eliminar cita | |
|---|---|
| Actor principal | Documentalista sanitario. |
| Precondiciones | • El paciente está dado de alta en el sistema.<br>• El documentalista sanitario tiene acceso al sistema.<br>• El usuario tiene una cita médica añadida en el sistema. |
| Disparador | Se solicita la eliminación de la cita. |
| Historia de usuario | **Como** documentalista sanitario **quiero** eliminar una cita previamente añadida **para** que el paciente reciba los cambios en su dispositivo. |
| Descripción | El personal responsable podrá eliminar una cita del usuario. |
| Escenario principal | 1. El documentalista introduce los datos identificadores del paciente.<br>2. El documentalista escoge la cita que quiere eliminar y la elimina.<br>3. El sistema procesa la eliminación de la cita.<br>4. El sistema notifica al usuario de que se le ha eliminado una cita. |
| Extensión 1: Usuario no encontrado. | 2. El sistema no encuentra a ningún usuario con ese DNI.<br>3. El sistema avisa al usuario que no existe ningún paciente registrado en el sistema con ese DNI.<br>4. El sistema recomienda al usuario que compruebe que los datos introducidos son correctos. |

| #22 Modificar cita | |
|---|---|
| Actor principal | Documentalista sanitario. |
| Precondiciones | • El paciente está dado de alta en el sistema.<br>• El documentalista sanitario tiene acceso al sistema.<br>• El usuario tiene una cita médica añadida en el sistema. |
| Disparador | Se solicita la modificación de la cita. |
| Historia de usuario | **Como** documentalista sanitario **quiero** modificar una cita **para** que el paciente reciba los nuevos datos en su dispositivo. |
| Descripción | El personal responsable podrá modificar una cita del usuario. |
| Escenario principal | 1. El documentalista introduce los datos identificadores del paciente.<br>2. El documentalista escoge la cita que quiere modificar y la modifica.<br>3. El sistema procesa la modificación de la cita.<br>4. El sistema notifica al usuario de que se le ha modificado una cita. |
| Extensión 1: Usuario no encontrado. | 2. El sistema no encuentra a ningún usuario con ese DNI.<br>3. El sistema avisa al usuario que no existe ningún paciente registrado en el sistema con ese DNI.<br>4. El sistema recomienda al usuario que compruebe que los datos introducidos son correctos. |

| #23 Enviar confirmación cita | |
|---|---|
| Actor principal | Sistema. |
| Precondiciones | • El paciente tiene una cita confirmada.<br>• El paciente está dado de alta. |
| Disparador | Se confirma una cita. |
| Historia de usuario | **Como** sistema **quiero** enviar una confirmación de cita **para** que el paciente tenga constancia de que se ha confirmado una cita. |
| Descripción | Cuando se produzca una nueva cita, una modificación o una eliminación, se notificará al usuario para que este conozca los nuevos datos sobre la cita. |
| Escenario principal | No tiene ya que está englobado en los distintos casos de uso que comprenden la adición, la eliminación y la modificación de la cita. |

| #24 Reclamar cita | |
|---|---|
| Actor principal | Paciente. |
| Precondiciones | • El usuario está dado de alta en el sistema.<br>• El usuario tiene citas disponibles. |
| Disparador | El usuario quiere reclamar una cita. |
| Historia de usuario | **Como** paciente **quiero** reclamar una cita disponible **para** poder indicar que día y hora me van mejor. |
| Descripción | En caso de que la cita asignada no le vaya bien al usuario, este la podrá reclamar para que se modifique. |
| Escenario principal | 1. El usuario, desde la sección de calendario, selecciona la opción de reclamar de una de las citas que tenga disponibles.<br>2. El sistema le muestra un diálogo con un calendario.<br>3. El usuario escoge el día que le va bien.<br>4. El sistema le muestra ahora la opción de escoger hora.<br>5. El usuario escoge una hora que le va bien.<br>6. El sistema le muestra un diálogo de confirmación. |

| #25 Recordar cita | |
|---|---|
| Actor principal | Sistema. |
| Precondiciones | • El paciente tiene una cita confirmada.<br>• El paciente está dado de alta.<br>• El tiempo restante hasta la cita es de menos de 2 días. |
| Disparador | El tiempo restante hasta la cita es de menos de 2 días. |
| Historia de usuario | **Como** sistema **quiero** notificar al usuario cuando falten pocos días para su cita **para** recordar al usuario que tiene una cita en los próximos días. |
| Descripción | El usuario recibirá una notificación informándole de que quedan pocos días para su visita. |
| Escenario principal | 1. El sistema envía una notificación generada automáticamente con los datos de la cita al usuario. |

| #26 Visualizar listado de chats | |
|---|---|
| Actor principal | Paciente. |
| Precondiciones | • El paciente tiene la sesión iniciada.<br>• El paciente tiene chats disponibles. |
| Disparador | El paciente quiere visualizar su listado de chats. |
| Historia de usuario | **Como** paciente **quiero** visualizar mi historial de chats **para** poder consultar todos mis chats. |
| Descripción | El usuario podrá ver todo su histórico de chats ordenados de más reciente a más antiguo. |
| Escenario principal | 1. El paciente navega hacia la sección de chats.<br>2. Se solicita a la API el listado de chats del usuario.<br>3. La aplicación recibe los datos.<br>4. Se muestran los chats. |

| #27 Iniciar conversación | |
|---|---|
| Actor principal | Paciente. |
| Precondiciones | • El paciente tiene la sesión iniciada. |
| Disparador | El paciente quiere iniciar un chat. |
| Historia de usuario | **Como** paciente **quiero** iniciar un chat **para** resolver mis dudas respecto a cualquier aspecto de mi enfermedad. |
| Descripción | El usuario podrá iniciar una conversación con el personal sanitario eligiendo a quien se va a dirigir la consulta que va a realizar. |
| Escenario principal | 1. El usuario, desde la sección de chats, pulsa sobre el botón de + para iniciar una conversación con alguien del departamento de enfermería.<br>2. Se muestra un diálogo con diferentes opciones para que el usuario pueda detallar su consulta.<br>3. Se abre un chat con el personal especializado en la consulta del usuario.<br>4. Se envían los datos del chat a la API para que los procese. |

| #28 Responder mensaje (paciente) | |
|---|---|
| Actor principal | Paciente. |
| Precondiciones | • El usuario tiene la sesión iniciada.<br>• El usuario tiene un mensaje disponible. |
| Disparador | El paciente quiere responder a un mensaje recibido. |
| Historia de usuario | **Como** paciente **quiero** responder a los mensajes recibidos **para** poder expresar mis inquietudes de forma directa. |
| Descripción | El usuario podrá responder mensajes. |
| Escenario principal | 1. Tras recibir un mensaje, el usuario pulsa sobre la notificación o entra en el chat.<br>2. El usuario escribe su respuesta.<br>3. Se envía la respuesta del usuario. |

| #29 Responder mensaje (enfermera) | |
|---|---|
| Actor principal | Enfermera. |
| Precondiciones | • El usuario tiene la sesión iniciada.<br>• El usuario tiene un mensaje disponible. |
| Disparador | La enfermera quiere responder a un mensaje recibido. |
| Historia de usuario | **Como** enfermera **quiero** responder a los mensajes recibidos **para** poder ayudar al paciente de la forma en que lo requiera. |
| Descripción | El usuario podrá responder mensajes. |
| Escenario principal | 1. Se recibe el mensaje.<br>2. La enfermera redacta su respuesta, o solicita al paciente más información para poder elaborar una buena respuesta.<br>3. Se envía la respuesta al usuario. |

## 8.3 Requisitos no funcionales

La propuesta de requisitos no funcionales de esta memoria se basa en la plantilla Volere de James y Suzanne Robertson [29].

Para la verificación de las distintas condiciones de satisfacción propuestas a continuación, se realizarán encuestas y *tests* de usuario que nos permitirán llegar a la conclusión necesaria para cada tipo de requisito.

### 8.3.1 Apariencia

**Descripción:** El sistema que se plantea tiene que ser visualmente atractivo.
**Justificación**: La estética en una aplicación es muy importante, si no está bien diseñada, puede llegar a causar que el usuario deje de utilizarla.
**Condición de satisfacción:** Una vez esté finalizada la aplicación, el 90% de los usuarios tienen que concluir en que la aplicación es estéticamente agradable.

### 8.3.2 Estilo

**Descripción:** El estilo de un sistema software son sus principales rasgos y características.
**Justificación:** El sistema está enfocado en su totalidad a llenar un vacío en el sistema sanitario, por lo tanto, se tiene que tratar de un sistema serio y, en consecuencia, el estilo no puede indicar lo contrario.
**Condición de satisfacción:** El sistema va a seguir los estilos ya definidos para Android. Adicionalmente, cerca del 90% del personal sanitario contactado debe tener la visión de que el sistema planteado es un producto serio.

### 8.3.3 Usabilidad

**Descripción:** El sistema software tiene que ser altamente usable tanto para pacientes como para personal sanitario.
**Justificación:** Este requisito no funcional es primordial en el caso de este sistema software, ya que el producto va a ser utilizado por gente de un amplio espectro de edad, y, por lo tanto, es esencial que el sistema sea fácil de usar. De lo contrario, el sistema podría resultar una molestia para el usuario o requerirá de terceros para aprender a usarlo.
**Condición de satisfacción:** El 80% de los usuarios tienen que ser capaces de acceder a las cuatro secciones principales de la aplicación (análisis, medicación, calendario y chat) de forma rápida y sin ninguna dificultad.

### 8.3.4 Internacionalización

**Descripción:** Este requisito implica la capacidad del sistema en disponer de distintas opciones de idioma.
**Justificación:** En España se hablan en la actualidad los siguientes idiomas: catalán, gallego, euskera, aranés y el castellano. Esta gran diversidad lingüística refleja la necesidad de que el sistema tenga la opción de poder escoger el idioma preferido para cada usuario.
**Condición de satisfacción:** Existirán dos idiomas de forma inicial en la aplicación (catalán y castellano) y será muy sencillo añadir de nuevos.

### 8.3.5 Aprendizaje

**Descripción:** Implica el nivel de dificultad que tiene el sistema para ser utilizado.
**Justificación:** Como ya se ha comentado, el espectro de edades será muy amplio, y es por eso por lo que no podemos construir un sistema que sea complejo de aprender a usar.
**Condición de satisfacción:**

- Un ingeniero puede utilizar la totalidad de funcionalidades de la aplicación sin problemas.
- Después de utilizar el sistema durante 10 horas, los usuarios tienen que haber aprendido cómo funciona cerca del 80% de la aplicación.

### 8.3.6 Velocidad y latencia

**Descripción:** El tiempo que va a tardar nuestro sistema en completar tareas específicas.
**Justificación:** El sistema tiene que ser capaz de lanzar notificaciones instantáneas para que el usuario conozca en todo momento la medicación que debe tomar.
**Condición de satisfacción:** Una vez se dispongan de los cambios en la medicación, el usuario los tiene que recibir en su teléfono en menos de 1 minuto.

### 8.3.7 Seguridad y privacidad

**Descripción:** Gestión de la privacidad y de la seguridad de los usuarios que van a utilizar la aplicación.
**Justificación:** El sistema software va a trabajar con datos privados sobre los usuarios, y es por eso que tienen que estar altamente protegidos y sin ser expuestos a terceros.
**Condición de satisfacción:** El sistema software mantendrá los datos claves de los usuarios cifrados en base de datos.

### 8.3.8 Exactitud

**Descripción:** La exactitud que necesitamos para que los resultados sean buenos.
**Justificación:** Los resultados que emita el sistema tendrán que ser tan precisos o más como los que pudiera emitir un doctor.
**Condición de satisfacción:** El 95% de los resultados emitidos durante la fase de prueba coinciden con el criterio de un doctor especializado en el tratamiento crónico para el cual se están generando resultados.

### 8.3.9 Fiabilidad y disponibilidad

**Descripción:** Este requisito refleja la necesidad de que el sistema sea fiable y estable.
**Justificación:** El usuario puede recibir el resultado de una analítica en cualquier momento, o puede querer consultar su medicación en cualquier otro, y es por eso por lo que el sistema tiene que estar siempre disponible.
**Condición de satisfacción:** El sistema tiene que estar disponible el 99% del año.

### 8.3.10 Escalabilidad y extensibilidad

**Descripción:** El tamaño que el producto puede llegar a alcanzar.
**Justificación:** Para que el producto sea útil, dada la gran cantidad de enfermedades crónicas que hay, se necesita una forma de añadir nuevas enfermedades de forma rápida y sencilla.
**Condición de satisfacción:** Se podrá añadir la automatización de una nueva enfermedad crónica sin la necesidad de modificar el código ya implementado.

### 8.3.11 Longevidad

**Descripción:** Determina la esperanza de vida del producto.
**Justificación:** El producto tiene que resultar rentable, y para que eso ocurra en el caso que nos incumbe, tiene que ser un sistema que sea usado por muchos años.
**Condición de satisfacción:** El sistema es sencillo de mantener, y, por lo tanto, es sencillo realizar actualizaciones que le permitan tener una vida más longeva.

### 8.3.12 Adaptabilidad

**Descripción:** Plataformas sobre las que se va a poder utilizar el sistema.
**Justificación:** Para que el producto se utilice al máximo, se tiene que poder utilizar en plataformas que sean usadas por la mayoría de los usuarios.
**Condición de satisfacción:** El producto inicialmente se tendrá que poder ejecutar en Android.

## 8.4 Esquema conceptual

El diagrama de clases que se verá a continuación ha sido diseñado utilizando el lenguaje UML, gracias al diseño de este esquema se consigue tener una visión del sistema a implementar previa a la implementación per se. En las siguientes secciones podremos ver el diagrama de clases, sus restricciones textuales y finalmente un glosario para terminar de entender el diagrama propuesto.

### 8.4.1 Diagrama UML

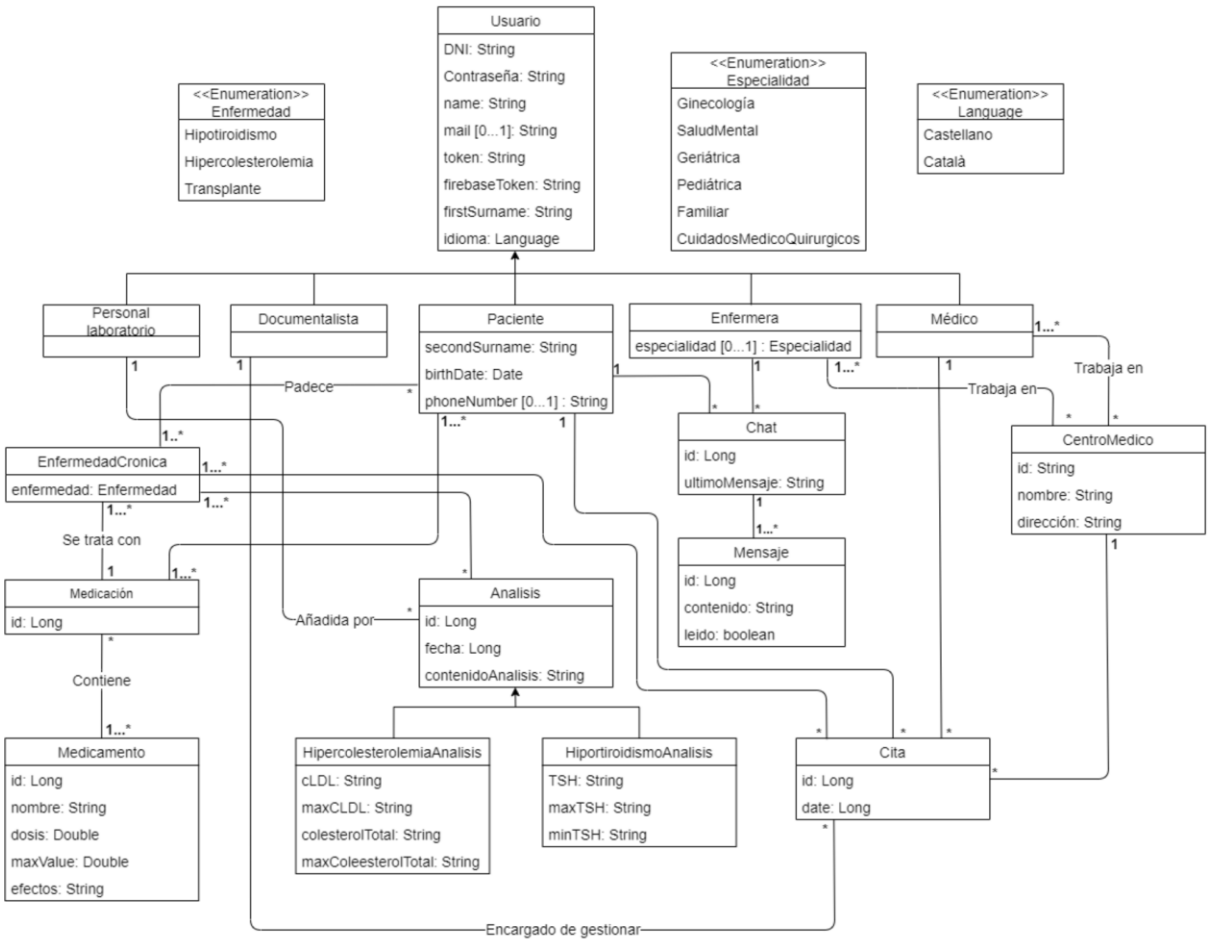



*Ilustración 14: Diagrama de clases en UML. Fuente: Elaboración propia.*

### 8.4.2 Restricciones textuales

En el diagrama UML no se pueden especificar todas las reglas y condiciones necesarias del sistema, es por eso por lo que necesitamos una serie de restricciones escritas que nos permitan acabar de definirlo.

1. Claves externas:
    - (Usuario, DNI)
    - (CentroMedico, id)
    - (Chat, id)
    - (Mensaje, id)
    - (Cita, id)
    - (Analisis, id)
    - (Medicamento, id)
    - (Medicación, id)
    - (EnfermedadCronica, enfermedad)
2. Un paciente no puede tener dos citas a la misma hora.
3. Un paciente no podrá tener más de 2 consultas abiertas (chats) al mismo tiempo.
4. Un análisis de sangre tiene que ser para controlar una enfermedad padecida por el paciente.
5. El médico responsable de la cita del paciente tendrá que trabajar en el centro médico en el que se va a atender la cita.
6. Solo las enfermeras con especialidad podrán responder a las consultas de los pacientes.
7. La dosis de un medicamento nunca podrá ser superior al valor máximo recomendado.
8. Los medicamentos que formen parte de la misma medicación no podrán tener efectos que anulen las capacidades del otro.
9. La fecha de nacimiento del paciente tendrá que ser siempre anterior a la fecha actual.
10. El DNI del usuario tendrá que ser un DNI válido.
11. El correo electrónico del usuario tendrá que ser un mail válido.
12. Un paciente deberá tener o un correo electrónico o un teléfono introducidos en el sistema como método de contacto, nunca podrán estar vacíos estos dos campos simultáneamente.

### 8.4.3 Glosario del esquema

#### Usuario

La clase usuario contiene toda la información necesaria de este. Además, tiene un total de 5 subclases que comparten los siguientes atributos:

- DNI: Documento Nacional de Identidad, sirve como identificador del usuario, pues es único para cada persona.
- Contraseña: contraseña que el usuario escoge para acceder al sistema.
- Nombre: nombre del usuario.
- Mail: correo electrónico que introduce el usuario a la hora de registrarse.
- Token: identificador que sirve como identificador del usuario, nos ahorrará el tener que pasar el DNI y la contraseña del usuario a la API para obtener los datos.
- firebaseToken: objeto que identifica el dispositivo del usuario y nos permite enviarle mensajes directamente.
- firstSurname: primer apellido del usuario registrado.
- Idioma: Parámetro de tipo enum que contiene el idioma preferido por el usuario.

### Personal laboratorio

El personal de laboratorio es una de las subclases de Usuario, estos son los encargados de introducir las analíticas en el sistema, y es por eso por lo que el esquema UML contiene una relación entre esta clase y la de análisis.

### Documentalista

El documentalista sanitario es el encargado de añadir/modificar/eliminar las citas que tendrá el paciente, es por eso por lo que necesitamos de esta clase. Como podemos ver existe una relación entre esta clase y la clase Cita.

### Paciente

La clase paciente es una de las principales del sistema, este tiene relación con un gran número de clases debido a que es el principal *stakeholder* de la aplicación. Además, cuenta con unos atributos únicos con los que no cuentan las otras subclases de Usuario.

- segundoApellido: el segundo apellido del paciente.
- fechaNacimiento: la fecha de nacimiento del paciente.
- teléfono: el número de teléfono del paciente.

### Enfermera

La enfermera será la encargada del chat, y es por eso por lo que guarda relación con el mismo dentro del diagrama. Adicionalmente, necesitaremos registrar su especialidad para poder gestionar de una mejor forma las conversaciones que puede tener con cierto tipo de pacientes.

- especialidad[0…1]: enumeración que representa la especialidad de la enfermera.

### Médico

El médico asistirá a las citas programadas al paciente, es por este motivo que dentro del sistema existe la relación entre esta clase y la clase cita.

### Centro médico

Los médicos y las enfermeras trabajan en centros médicos, será en estos lugares donde las citas se llevarán a cabo, razón principal por lo que es necesaria esta clase. Tiene los siguientes atributos:

- id: identificador del centro.
- nombre: nombre que tiene el centro médico.
- dirección: dirección completa del centro.

### Enfermedad crónica

Clase que sirve como núcleo de todo el sistema, un paciente tiene una enfermedad, los análisis se realizan para controlar dicha enfermedad, las citas son para el control de esta, y la medicación sirve para tratar la enfermedad.

- enfermedad: enumeración con todas las enfermedades que trata el sistema.

### Medicación

La medicación es el objeto que guarda toda la información referente al tratamiento del usuario.

- Id: identificador de la medicación.

### Medicamento

El medicamento es el objeto que guarda toda la información del fármaco que tiene que tomar el usuario.

- Id: identificador del medicamento.
- nombre: nombre del medicamento.
- dosis: cantidad, generalmente en mg, de medicamento que tiene que tomar el paciente.
- valorMax: valor máximo recomendado de un medicamento.
- efectos: Efectos que tiene un medicamento en el tratamiento de la enfermedad.

### Análisis

Objeto que guarda toda la información referente a los análisis de sangre que se realiza el usuario, esta clase tiene diferentes subclases ya que dependiendo de la enfermedad que se trate de controlar, se observan unos parámetros u otros, los atributos principales son los siguientes:

- id: identificador del análisis.
- fecha: fecha en la que se realizó la analítica.
- contenidoAnalisis: resultados de la analítica.

### Hipercolesterolemia Análisis

Objeto que almacena todos los parámetros específicos que nos permiten controlar la hipercolesterolemia.

- cLDL: parámetro que almacena la cantidad exacta de cLDL obtenida en la analítica.
- maxcLDL: valor fijado como máximo en cuanto al cLDL se refiere.
- colesterolTotal: colesterol total obtenido en la analítica.
- maxColesterolTotal: cantidad total de colesterol máxima permitida.

### Hipotiroidismo Análisis

Clase que nos permite guardar todos los parámetros que nos sirven para controlar el hipotiroidismo.

- TSH: atributo que nos permite guardar la cantidad de TSH obtenida por el paciente en el análisis de sangre.
- maxTSH: valor máximo permitido de TSH.
- minTSH: valor mínimo permitido de TSH.

### Chat

Objeto que nos permite almacenar todo lo necesario para mostrar el listado de chats dentro de la aplicación.

- id: identificador del chat.
- ultimoMensaje: último mensaje recibido/enviado, será lo que se muestre en el listado de chats.

### Mensaje

Objeto que nos permite almacenar toda la información referente a un mensaje intercambiado entre paciente y enfermera.

- id: identificador del mensaje.
- contenido: contenido del mensaje.
- leído: booleano que nos permite saber si el usuario ha leído o no el mensaje.

### Cita

Este objeto nos permitirá guardar todo lo referente a las citas que tiene el usuario para controlar su enfermedad crónica.

- Id: identificador de la cita.
- Date: fecha y hora de la cita del paciente.

### Enfermedad <<Enum>>

Este enum nos permite conocer en todo momento que enfermedades son las que está tratando el sistema.

### Especialidad <<Enum>>

Este enum contiene todas las especialidades que actualmente tiene la rama de enfermería [30].

### Language <<Enum>>

Gracias a este enum, podremos conocer el idioma preferido del usuario.

# 9. Descripción técnica del sistema

Tras realizar toda la especificación completa de requisitos todos los objetivos del sistema están descritos a un nivel que nos permite comenzar con la implementación de todo el sistema. Antes de comenzar el desarrollo, necesitamos definir y describir toda la parte técnica.

## 9.1 Visión global de la arquitectura

El sistema consta de dos grandes partes, una aplicación móvil y una API REST. La arquitectura que vemos en la imagen 15 refleja estos componentes y los sitúa como piezas centrales de la arquitectura global del sistema.

Además de los dos componentes principales, contamos con otros que de igual manera son necesarios para el correcto funcionamiento del sistema. De entre ellos, podemos destacar Postman. Postman es un programa que nos permite enviar peticiones HTTP sin la necesidad de desarrollar un cliente.

La aplicación móvil será la parte con la que interactuarán los pacientes. Sin embargo, tanto médicos, enfermeras, como el personal del laboratorio interactuarían con un programa informático (panel de control o similares) cada uno distinto y propio de su lugar de trabajo. Por ejemplo, el personal de laboratorio, a la hora de subir los resultados de los análisis al sistema, tendría un programa que les permitiría realizar esto. En el caso que nos incumbe, se ha decidido reemplazar todos estos programas (que se encuentran fuera del alcance del proyecto), por Postman, el cual realiza este papel.

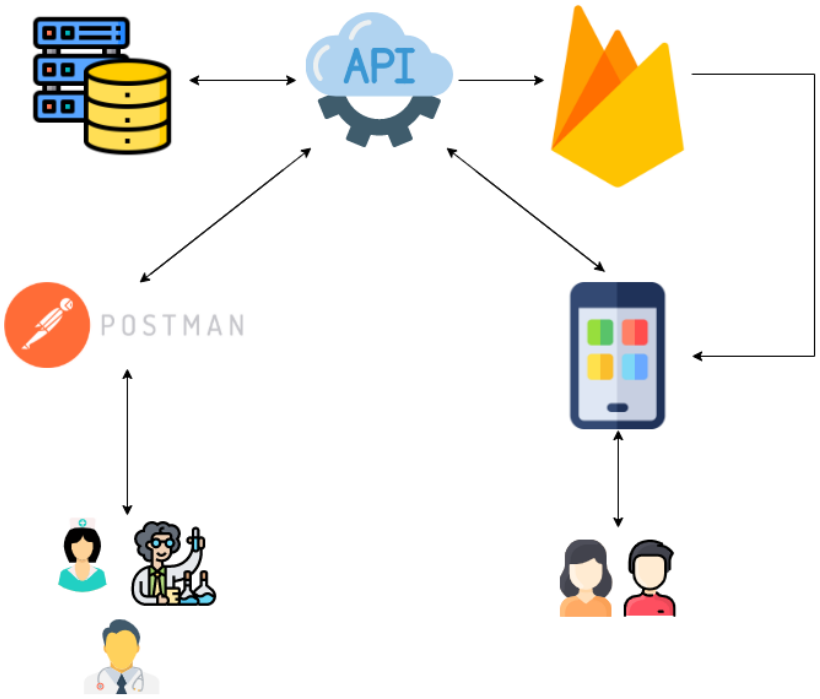



*Ilustración 15: Arquitectura global del sistema. Fuente: Elaboración propia.*

Los dos componentes que restan por explicar son la base de datos y Firebase Cloud Messaging (FCM). La base de datos alojada en AWS será la encargada de almacenar todos los datos necesarios para el buen funcionamiento del sistema en distintas tablas, explicadas más adelante. FCM es el módulo encargado de la mensajería, nos permite notificar al usuario de forma muy sencilla cuando tiene un nuevo análisis a consultar, una nueva cita, existen cambios en su medicación, y más.

Por último, las flechas que se pueden apreciar en la imagen representan las direcciones en las que viaja la información, ya sea a nivel de peticiones, o envío de datos.

## 9.2 Diagrama de componentes

En la ilustración 16 podemos ver el diagrama de componentes de la aplicación, este diagrama muestra la división actual del sistema y es la representación UML de la arquitectura mostrada en el apartado anterior. Por simplicidad, se ha evitado duplicar conexiones entre módulos simplemente realizando dicha conexión a nivel de subsistema siempre que sea posible.

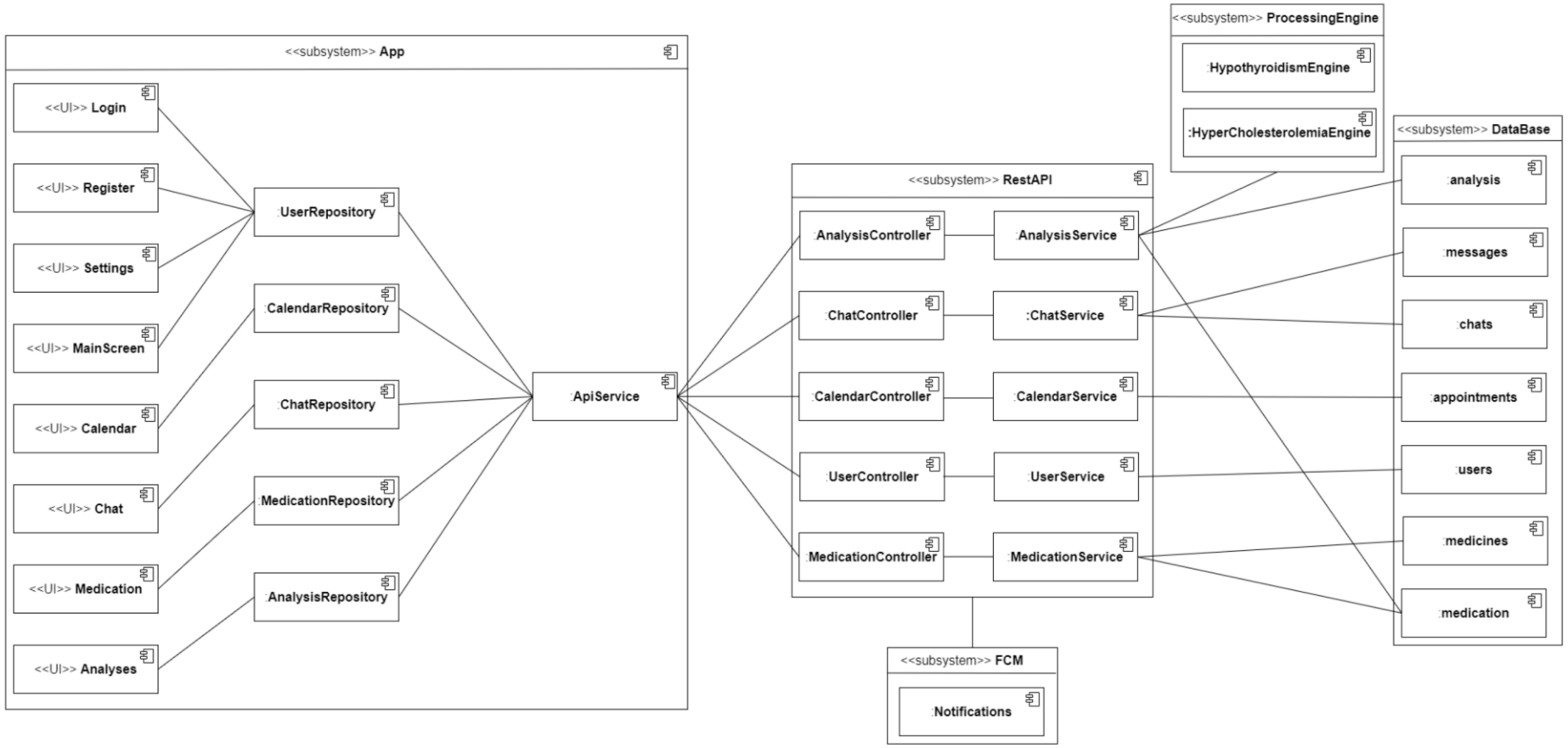



*Ilustración 16: Diagrama de componentes del sistema. Fuente: Elaboración propia*

## 9.3 Front-end

El front-end es toda la parte del sistema software con la cual el interactúa el usuario, esta parte es la encargada de recoger todos los datos o peticiones por parte de los usuarios del sistema y enviarlas al back-end para que sean procesadas.

En el caso que nos incumbe, el front-end consta de una aplicación desarrollada en Kotlin, a través de esta aplicación podemos acceder a todas las funcionalidades que el sistema es capaz de ofrecer. Adicionalmente, Postman será el front-end simulado del resto de los *stakeholders*.

### 9.3.1 Funcionalidades de la aplicación

#### Inicio de sesión y registro

Actualmente el sistema consta de una fase de registro, en la que un usuario introduce todos los datos que le pide el sistema y envía el formulario para poder iniciar sesión en un futuro. Sin embargo, esto se ha diseñado así al tratarse de un proyecto académico, pues de integrar este sistema en un sistema sanitario o en grupos sanitarios privados, estos nos proporcionarían las credenciales utilizando como identificador el numero de la seguridad social u otros identificativos diferentes.

En cuanto al inicio de sesión, se trata de un input para introducir el DNI, y otro para introducir la contraseña, tras pulsar sobre el botón de inicio de sesión, la aplicación envía los datos a la API y esta es la que verifica que los datos sean correctos, en caso de serlo, el usuario puede entrar y hacer uso del sistema. En caso contrario, la aplicación informa al usuario de que ha habido un problema con el inicio de sesión, sin dar indicativos sobre el campo incorrecto.

#### Consulta de analíticas

Dentro de la aplicación tenemos la sección de análisis. Allí encontramos la información de las distintas analíticas que nos hemos ido realizando a lo largo del tiempo en formato tarjeta. Como ya se ha comentado, podemos pulsar sobre cada tarjeta para desplegar los detalles particulares de cada análisis, tras pulsar, se nos abre un diálogo que muestra los parámetros concretos de la analítica que estamos consultando.

En cada tarjeta vemos información sobre la fecha en la que se realizó la analítica, la enfermedad para la que se realizó el control y si se modificó, o no, la medicación tras procesar esa analítica. En la parte inferior de la tarjeta tenemos dos botones, el primero nos permite consultar la analítica, y, tras pulsarlo, abre un diálogo con los datos de esta. El segundo nos permite compartir los resultados con la persona que queramos.

#### Consulta de medicación

En esta sección podemos consultar la medicación actual que tenemos que tomar para tratar una enfermedad en particular. De igual forma que en la sección de análisis, la información se nos presenta en formato tarjeta, en este caso se nos presenta en la tarjeta la medicación (nombre y dosis) y la enfermedad para la cual tenemos que tomar dicha medicación. Además, en la parte inferior de la tarjeta contamos con el mismo botón de compartir, para facilitar la distribución de la información.

### Calendario

Al pulsar sobre calendario, podemos acceder a una lista con todas nuestras próximas citas. Desde aquí podremos ver la fecha de nuestra próxima visita, la enfermedad que se nos va a controlar, la hora y la ubicación, todo esto en formato tarjeta Además, en este caso, aparte del botón de compartir, contamos con un botón de "reclamar" el cual nos abre un diálogo con un calendario permitiéndonos escoger una fecha y, posteriormente, una hora que nos vaya bien para realizar dicho control. Una vez se ha seleccionado, el sistema muestra otro diálogo informando al usuario de que se ha procesado su petición.

### Chat

El chat es la cuarta funcionalidad principal con la que cuenta la aplicación, al pulsar sobre el botón y navegar hasta esta sección nos encontramos con una lista de chats (en caso de que tengamos alguno), esta lista representa el histórico de chats que tiene el paciente. Para iniciar uno de nuevo, simplemente tenemos que pulsar sobre el "+" situado en forma de botón flotante en la parte inferior de la pantalla. Una vez iniciemos un nuevo chat, podremos escribir y enviar nuestra duda y esperar una respuesta por parte del departamento de enfermería.

### Sistema de notificaciones

Cuando el usuario dispone de información nueva, recibe una notificación en su dispositivo, esta es enviada por la API cuando sucede alguno de los siguientes eventos:

- Se añade una nueva analítica.
- Se modifica la medicación del paciente.
- Se añade una nueva cita.
- Se elimina una cita.
- Se modifica una cita.
- Se recibe un mensaje.

Una vez recibida en el dispositivo, el paciente puede visualizar una notificación con un título y un contenido representativos de la notificación que acaba de recibir. Si decide pulsar sobre esta, la aplicación se encargará de llevar al usuario a la sección correspondiente dependiendo siempre del tipo de notificación recibida.

### Ajustes y perfil de usuario

Al tratarse de una aplicación destinada al mundo sanitario, el perfil de usuario es escueto, en particular, solo cuenta con dos campos que nos permiten visualizar nuestros datos de contacto para poder verificar que sean correctos.

En la sección de ajustes, podemos modificar nuestra contraseña, introduciendo la actual y dos veces la nueva, y modificar el idioma en el cual queremos que se muestre toda la información.

### Drawer (Menú de navegación lateral)

Para facilitar la navegación entre secciones, se ha implementado un *drawer*, este es accesible desde cualquier sección de la aplicación, y nos permite, además de navegar hacia todas las secciones, cerrar sesión y acceder al perfil de forma rápida y sencilla.

### 9.3.2 Diseño de la interfaz gráfica

Para poder llegar a un nivel de detalle lo suficientemente elevado como para poder comenzar la implementación de la parte visual de la aplicación, se tuvo que realizar un proceso de diseño de la interfaz de usuario. Para poder desarrollar una interfaz con un alto grado de usabilidad, se han empleado las técnicas descritas en el libro de Ben Schneiderman llamado *Designing the user interface: strategies for effective humnan-computer interaction* [31].

A continuación, se van a exponer los distintos *mockups* que se realizaron previamente a la implementación de cada parte de la aplicación junto con el resultado final una vez terminada dicha implementación. En las imágenes que vamos a ver a lo largo de la sección, la de la izquierda corresponde con el diseño original y la de la derecha a la implementación final.

#### Pantalla principal

Como podemos ver en la ilustración 17, contamos con una pantalla principal muy intuitiva, consta de cuatro botones principales que nos permiten navegar hacia las distintas secciones de las que consta la aplicación. Además, contamos con dos botones pequeños en la parte superior derecha de la pantalla que nos permitirían navegar hacía ajustes y cerrar sesión.

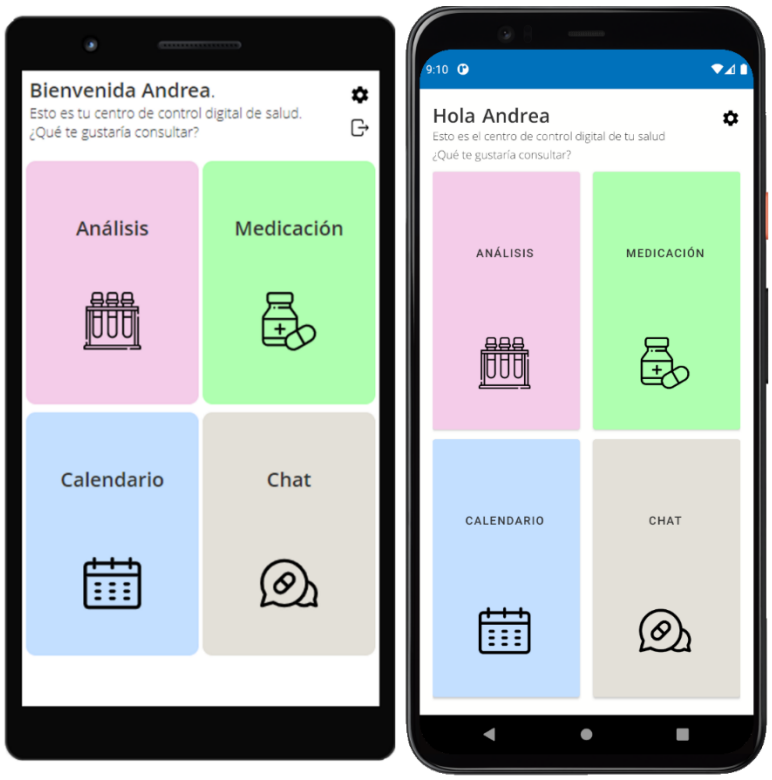



*Ilustración 17: Diseño original y actual de la pantalla principal. Fuente: Elaboración propia.*

En la implementación actual, el único cambio importante que se ha realizado respecto al diseño original es la eliminación del botón para cerrar sesión, el motivo de este cambio fue que se consideró que la pantalla principal quedaba demasiado cargada con la adición de este botón. En adición al cambio anterior, también se decidió sustituir el Bienvenido/a por un Hola, la razón fue puramente para evitar tener que realizar distinciones de género.

### Análisis

El diseño original de la sección de análisis ya planteaba un muestreo de la información en formato tarjeta, como podemos observar, se pretendía mostrar la fecha, si el análisis era rutinario y si se modificaba o no la medicación.

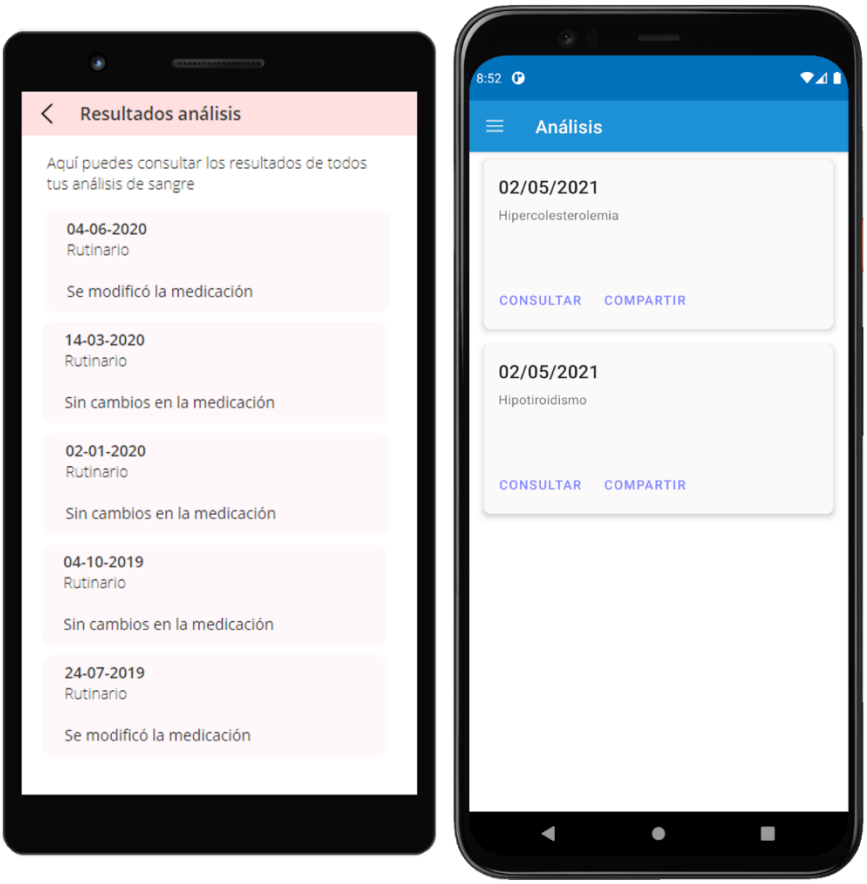



*Ilustración 18: Diseño original y actual de la pantalla de análisis. Fuente: Elaboración propia.*

Como se puede apreciar, el diseño ha cambiado bastante respecto al original, a pesar de que se sigue conservando el formato tarjeta, se han introducido varios botones nuevos como el de consultar y compartir. Además, se ha sustituido la zona donde indicaba originalmente si el análisis era rutinario por un campo que indica la enfermedad que se está controlando en dicho análisis.

La adición de los botones fue debido a que mientras se realizaba el diseño no se tuvieron en cuenta, y a la hora de implementar esta sección se tuvieron que añadir. Y, en cuanto a la adición de la enfermedad a controlar, se consideró que aportaba más información que un campo indicando si la analítica era o no rutinaria, pues todas las analíticas que finalmente se van a tratar en este sistema van a ser rutinarias. Finalmente, y algo que se va a repetir en las cuatro secciones principales (análisis, medicación, calendario y chat) hay una pequeña diferencia visual que acaba resultando en una mejora enorme en la usabilidad de la aplicación. Como podemos observar en estas secciones, todas los diseños originales cuentan con una flecha hacia atrás en la barra superior, esto se ha sustituido por un menú lateral (simbolizado con las tres líneas paralelas) que nos permite navegar hacia todas las secciones de la aplicación sin necesidad de pulsar sobre la flecha, ir a la pantalla principal y navegar desde ahí.

### Medicación

Respecto a la sección de la medicación, el diseño original nos planteaba de nuevo el formato tarjeta, el cual esta vez dividía claramente una sección que hacía de cabecera y otra que proponía mostrar la medicación a tomar (nombre y dosis).

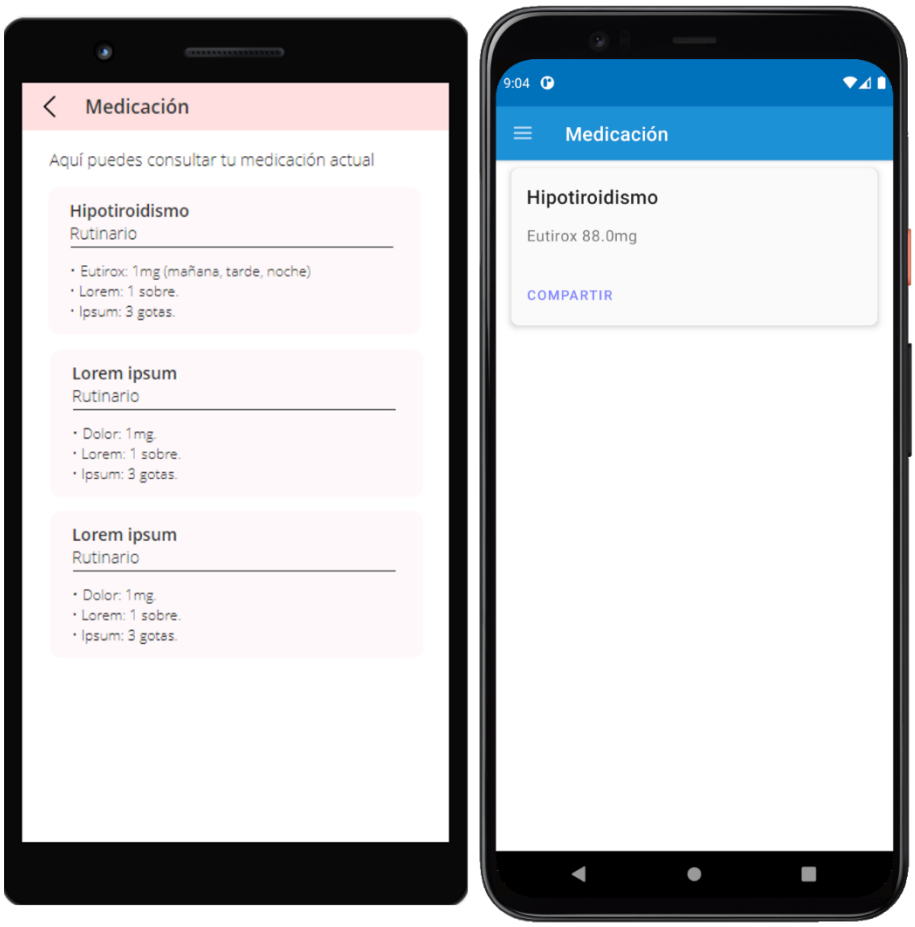



*Ilustración 19: Diseño original y actual de la pantalla de medicación. Fuente: Elaboración propia.*

Sin embargo, esta sección también ha cambiado bastante, de nuevo se elimina el campo de rutinario, se elimina la línea divisoria y además se añade el botón de compartir.

Al igual que en la parte de análisis, la adición del botón fue debido a que no se tuvo en cuenta durante el diseño. Respecto a los otros cambios, se decidió eliminar el campo de rutinario debido a que no aportaba información relevante, y la eliminación de la línea divisoria fue para mantener el mismo estilo que en las otras secciones de la aplicación.

### Calendario

El calendario ha sido de las secciones que menos ha cambiado respecto a su diseño original, como podemos ver, y, al igual que en las otras secciones, la información se muestra en formato tarjeta. En este caso se muestra la fecha, la enfermedad para la que se va a realizar el control, la hora y el lugar.

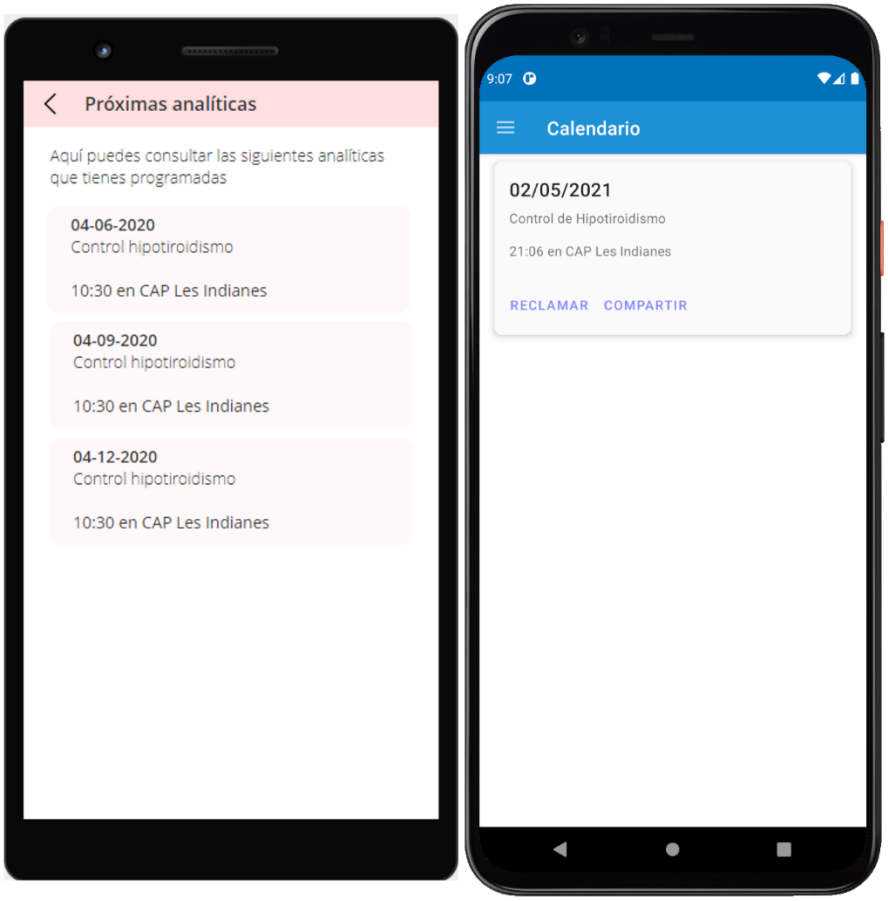



*Ilustración 20: Diseño original y actual de la pantalla de calendario. Fuente: Elaboración propia.*

En cuanto al muestreo de información, todos los campos que se planteaban originalmente se han mantenido, lo único que se ha añadido han sido los botones de reclamar y compartir, que, al igual que en los casos anteriores, se han tenido que añadir porque no se tuvieron en cuenta a la hora de realizar el diseño original.

### Ajustes

Los ajustes, que a su vez se plantearon como perfil de usuario, se presentan de una forma simple y minimalista, en la que el usuario puede consultar su información de contacto, modificar su contraseña y el idioma de uso de la aplicación.

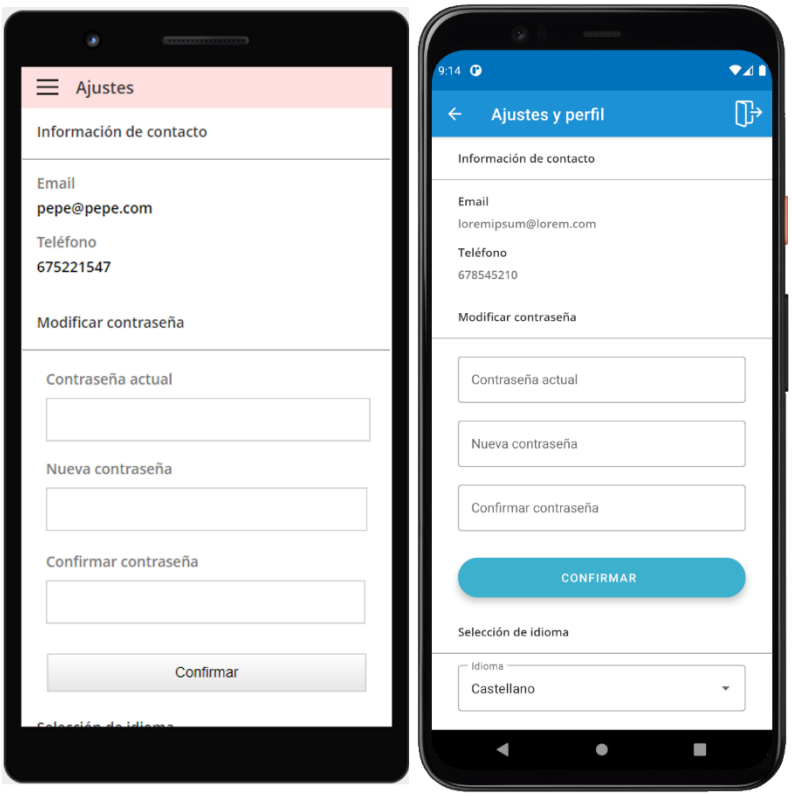



*Ilustración 21: Diseño original y actual de la pantalla de ajustes. Fuente: Elaboración propia.*

La simplicidad de esta pantalla es algo fundamental en la aplicación, ya que no es necesario complicar al usuario con ajustes que no va a llegar a entender, y que realmente no son necesarios para un tipo de aplicación como esta.

En cuanto a las diferencias respecto a la versión actual ya implementada, son prácticamente nulas, la pantalla de ajustes se ha realizado tal y como se diseñó incluyendo la barra superior. Sin embargo, se ha añadido un botón en esta barra con el cual podemos cerrar la sesión.

### 9.3.3 Navegación interna de la aplicación

La aplicación cuenta con un total de ocho secciones, explicadas en el apartado anterior. Para navegar entre ellas de forma eficiente se utilizó principalmente un menú lateral, el cual está presente en la mayoría de estas secciones. A continuación, podemos ver el diagrama de navegabilidad de la aplicación.

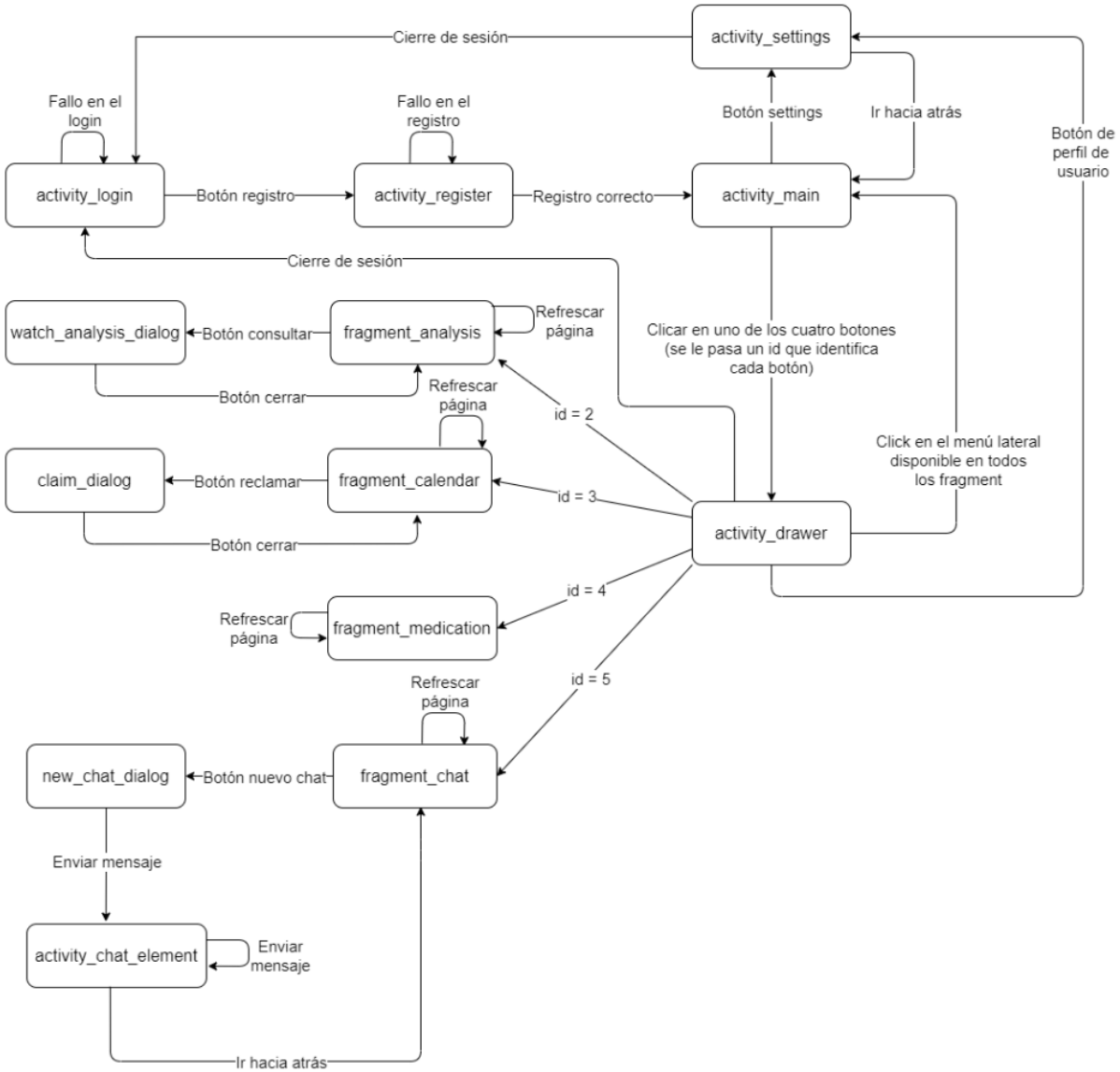



*Ilustración 22: Mapa navegacional de la aplicación. Fuente: Elaboración propia.*

### 9.3.4 Arquitectura

La arquitectura de la aplicación (MVVM) está basada en la propuesta por Google en su guía de arquitectura para aplicaciones Android [32]. Esta arquitectura nos permite tener las capas completamente desacopladas entre sí, consiguiendo tener toda la lógica de la aplicación separada por completo de la UI.

En primer lugar, se separaron las secciones en distintas Activities y Fragments. En particular, hay un Fragment o Activity por cada sección que tiene la aplicación. Esto se ha hecho principalmente para mantener en cada clase solo lo que es relativo a la misma.

En segundo lugar, se implementaron ViewModels para cada Activity/Fragment, desde estas clases, se gestiona toda la información que aparece en la pantalla (obteniéndola de bases de datos) y se realizan las llamadas al repositorio (para acceder a la base de datos). Para conseguir la "conexión" entre Activity y ViewModel, se utilizan LiveData, estos son unas clases que contienen datos observables. Estos objetos nos permiten observar los cambios a tiempo real y mostrarlos al instante en nuestra UI consiguiendo un desacoplamiento total de la UI y el dominio.

Los ViewModel retienen toda la información, y, por lo tanto, no son afectados por los posibles cambios en el ciclo de vida de la aplicación, logrando así, que por mucho tiempo que pase desde que el usuario, por ejemplo, cerró la aplicación, al retomar el uso de esta, se le muestren los mismos datos sin necesidad de realizar una recarga.

En tercer lugar, contamos con los repositorios. Estos son los encargados de comunicarse con el APIService para insertar, actualizar u obtener diferentes datos relevantes durante la ejecución de la aplicación. Adicionalmente, los repositorios nos aportan un aislamiento completo respecto a las fuentes de datos, por lo tanto, se pueden añadir tantas fuentes como necesitemos sin tener que modificar alguna otra cosa. En el caso que nos incumbe solo se dispone de una API, pero más adelante se podrían querer añadir diversas fuentes más.

Finalmente, tenemos el ApiService, desde aquí se realizan las llamadas directas a la API, la cual recogerá los datos enviados por este servicio y realizará los accesos que toquen en base de datos. A continuación, en la imagen 23 podemos ver una visión general de la arquitectura explicada:

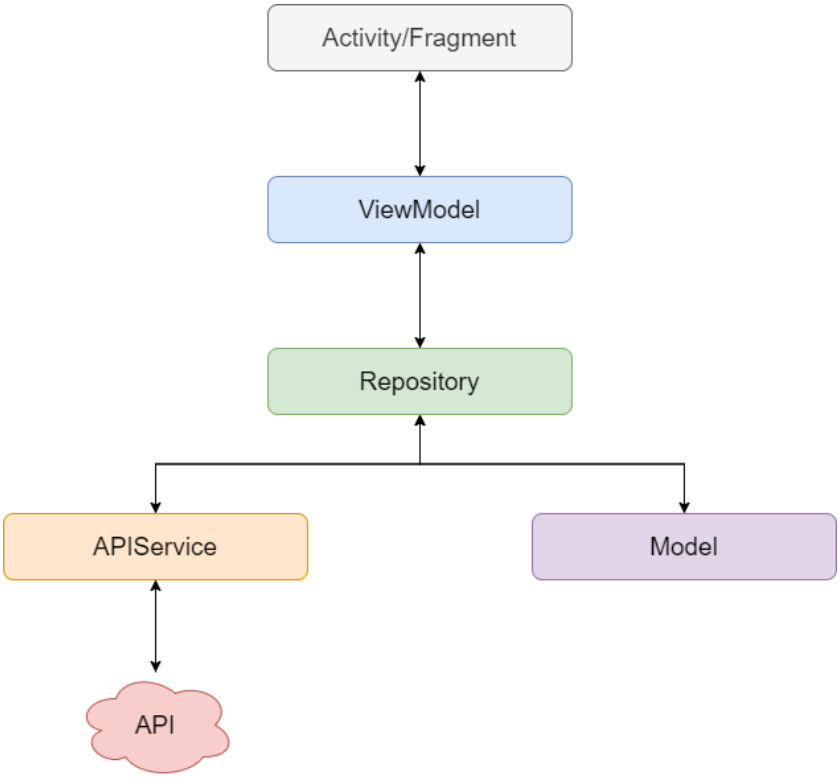



*Ilustración 23: Arquitectura de la aplicación. Fuente: Elaboración propia.*

Como podemos observar, la capa de vista está completamente aislada de la de datos, esto nos permite realizar cualquier modificación en cualquiera de las capas sin que la otra se entere. De hecho, podríamos cambiar por completo la tecnología empleada en la API, y la aplicación seguiría funcionando sin ningún problema.

## 9.4 Back-end

En contraposición con el front-end, el usuario no interactúa directamente con esta parte del sistema, sino que es la encargada a ejecutar toda la lógica demandada por la aplicación o Postman y de acceder a base de datos con los datos recogidos. En el caso de la API del sistema que se plantea, esta se encarga de toda la parte del procesado de analíticas, además de otras funcionalidades explicadas más adelante.

### 9.4.1 Arquitectura

La API sigue una arquitectura por capas, siguiendo el modelo de MVC (Modelo Vista Controlador). La capa de vista, en el caso que nos incumbe, es la capa a la cual le llegan todas las peticiones por parte de la aplicación y de Postman, esta está compuesta por los controladores, encargados de recoger las peticiones HTTP de tipo GET, POST, PUT y DELETE enviadas tanto por la aplicación móvil como por Postman. Desde esta capa accedemos directamente a los servicios, que son los encargados de llevar a cabo toda la lógica necesaria para el buen funcionamiento del sistema, estas dos capas están en contacto directo (pero no acopladas), pues se pasan información de una a otra para poder trabajar correctamente. Finalmente, tenemos los repositorios, que son los encargados de contactar con la base de datos ya sea para obtener, modificar, añadir o eliminar algún dato.

Se trata de tres capas bien diferenciadas y que nos permiten mantener toda la lógica separada de la "vista" y a su vez completamente separada de los accesos directos a datos.

En la ilustración 24, a parte de las tres capas comentadas se pueden ver dos módulos más, el primero (ProcessingEngine) es el encargado de procesar las analíticas, cada enfermedad cuenta con el suyo propio y es el servicio dedicado a los análisis de sangre el que llama a este motor de procesado para que realice un procesado de la analítica que acaba de recibir el sistema con la intención de actualizar la medicación de los pacientes en caso de que sea necesario. El segundo, FCM (Firebase Cloud Messaging), es el encargado de las notificaciones, este módulo nos permite enviar notificaciones directamente al usuario de forma muy sencilla simplemente indicando el título, el contenido y el tipo de notificación que estamos mandando a un usuario en particular.

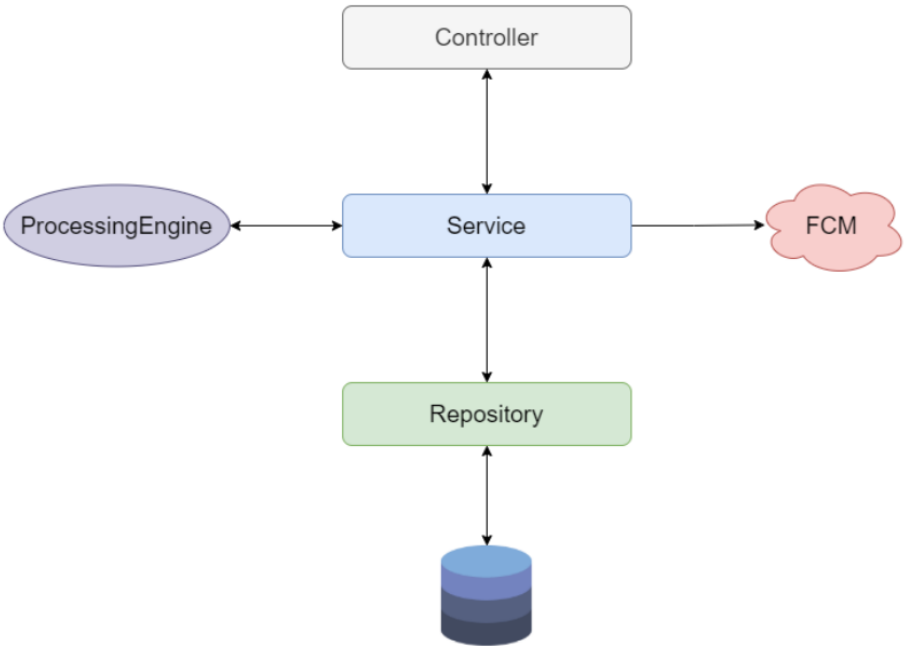



*Ilustración 24: Arquitectura de la API REST. Fuente: Elaboración propia.*

Esta arquitectura es la que se sigue en toda la API, cada funcionalidad cuenta con su controlador, su servicio, y su repositorio asociado, separando por completo toda la lógica correspondiente a cada funcionalidad. Sin embargo, el módulo de FCM es compartido por todos los servicios, pues la mayoría envían notificaciones. El de ProcessingEngine podría ser compartido de igual forma, pero en el caso que nos incumbe, solo tiene sentido acceder a él desde el Servicio correspondiente a las analíticas.

### 9.4.2 Funcionalidades

A continuación, se van a listar todas las funcionalidades que tiene la API de este sistema, todas con sus correspondientes diagramas de secuencia que pretenden ilustrar de forma exacta dicha funcionalidad. Para entender los es necesario tener una visión genérica de la anotación @AuthAwareRestController. Añadir esta anotación a una clase implica la realización de un preprocesado de la petición HTTP que se recibe donde se verifica el token introducido en la cabecera de autorización, una vez se finaliza este preprocesado se inyecta el usuario de forma automática en los distintos controladores de la API. Gracias a esto, nos evitamos tener que realizar el acceso de forma manual a base de datos para obtener el usuario para posteriormente utilizarlo en el tratamiento de la petición recibida. Este proceso se ve ilustrado en la figura 25.

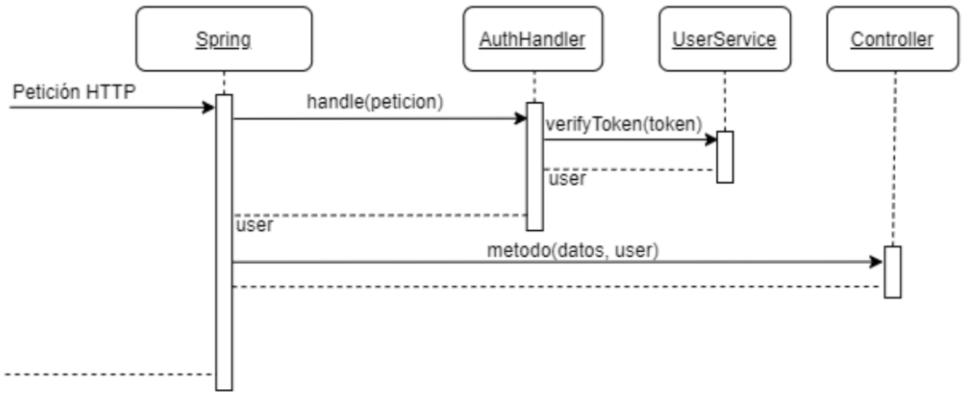



*Ilustración 25: Petición HTTP sobre controlador marcado con @AuthAwareRestController. Fuente: Elaboración propia.*

Gracias a este proceso, todos los controladores marcados con @AuthAwareRestController cuentan con el usuario ya inyectado cuando se comienza el proceso.

#### Registro

El sistema de identificación del sistema está basado en usuarios identificados por su Documento Nacional de Identidad (DNI). Cuando un usuario se registra desde la aplicación, a la API le llegan todos los datos necesarios para crear y guardar en base de datos dicho usuario. Además, se encarga de cifrar la contraseña (para que no sea visible en base de datos), y, en nuestro caso, al tratarse de un proyecto académico, les asigna enfermedades y medicaciones correspondientes a dichas enfermedades.

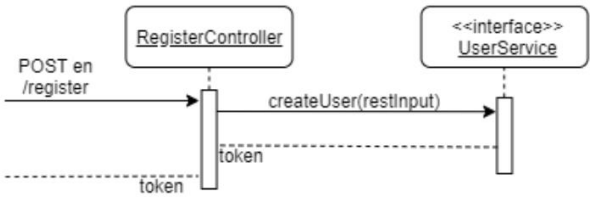



*Ilustración 26: Diagrama de secuencia del registro parte 1. Fuente: Elaboración propia.*

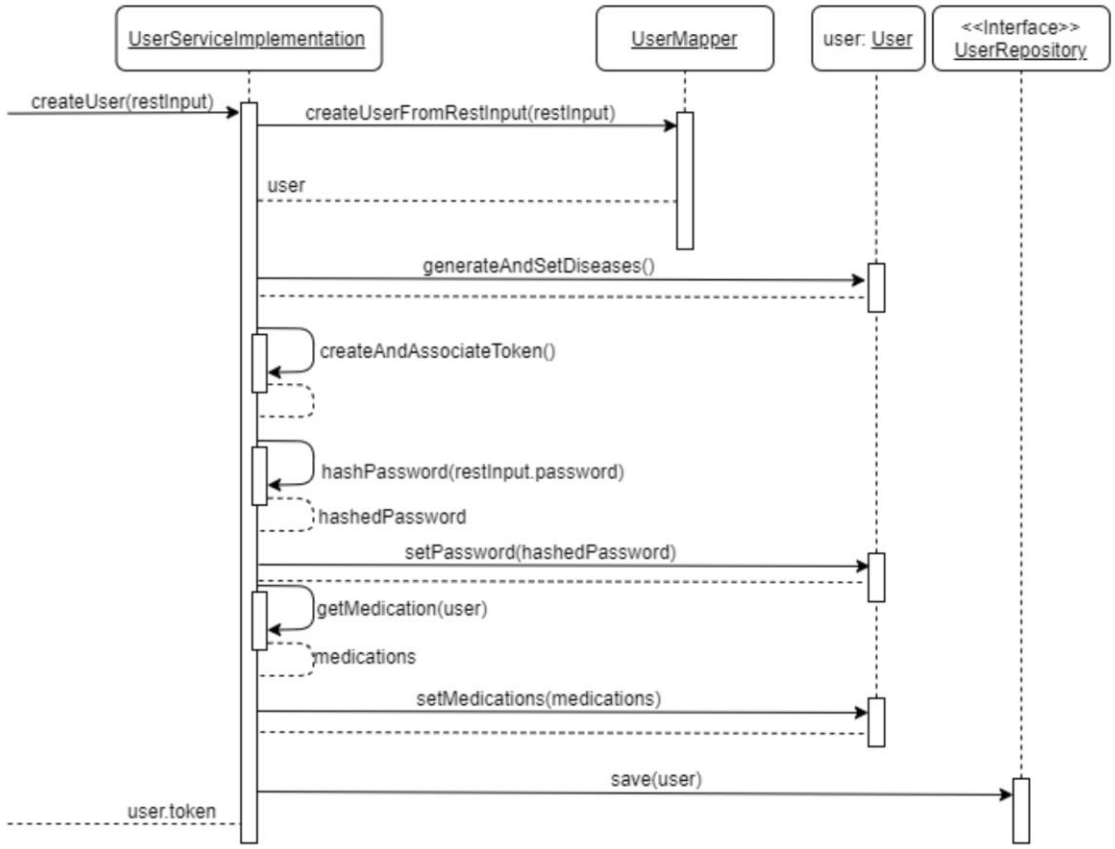



*Ilustración 27: Diagrama de secuencia del registro parte 2. Fuente: Elaboración propia.*

El proceso de creación de usuario comienza una vez se recibe la llamada en el controlador, este llama al servicio para que proceda a realizar lo necesario para la creación del usuario. Dentro del servicio, se llama a un mapper, este tipo de clase se va a ver repetidas veces a lo largo de la memoria, pues son las encargadas de recoger los datos enviados desde la aplicación o Postman y convertirlos en un objeto del modelo de nuestra API.

Como podemos ver en la ilustración 27, antes de guardar el usuario en base de datos se realizan unas cuantas acciones, se crea el token identificador del usuario, se cifra la contraseña, se le asignan enfermedades y medicaciones al usuario y finalmente se llama al repositorio para que este se encargue de guardar el objeto nuevo (user) en base de datos.

Finalmente, se devuelve el token previamente creado para que la aplicación pueda realizar las llamadas con este objeto, cosa que simplifica mucho las cosas evitando tener que pasar identificador y contraseña cada vez que se realice una petición.

### Obtención del Usuario

Existen diferentes secciones de la aplicación que requieren de datos del usuario, estos son el nombre, los datos de contacto, y su idioma. Para obtener estos datos, la API devuelve un usuario simplificado, el cual contiene exclusivamente los datos necesarios, evitándonos enviar ciertos datos que no interesan en esos casos particulares.

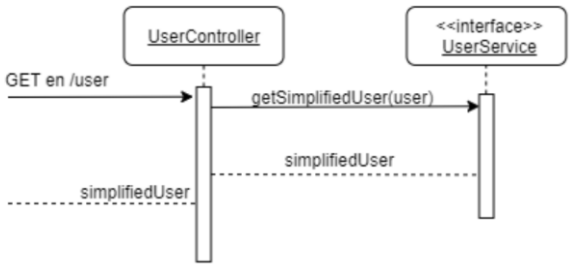



*Ilustración 28: Diagrama de secuencia de la obtención de usuario parte 1. Fuente: Elaboración propia.*

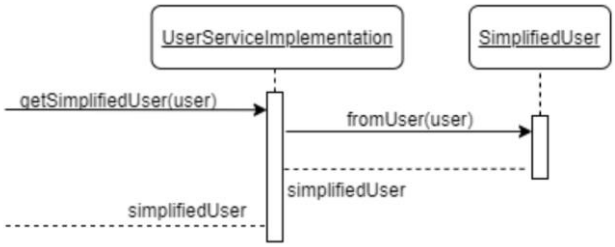



*Ilustración 29: Diagrama de secuencia de la obtención de usuario parte 2. Fuente: Elaboración propia.*

Como se puede ver en las ilustraciones 28 y 29, tras recibir un GET en /user se inicia todo el proceso para la obtención del usuario, como ya se ha comentado, no es necesario obtener todos los datos de este, es por eso por lo que con la llamada "fromUser(user)" sobre la clase SimplifiedUser obtenemos un objeto que contiene únicamente los datos necesarios.

### Modificar contraseña

El usuario, como se ha visto en las funcionalidades de la aplicación móvil, puede modificar su contraseña una vez tiene la sesión iniciada. Para realizar esto, se necesita ejecutar un proceso en la API, la cual realizará todas las comprobaciones pertinentes previas a la modificación final de la contraseña.

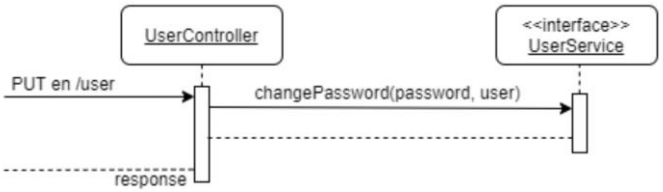



*Ilustración 30: Diagrama de secuencia del cambio de contraseña parte 1. Fuente: Elaboración propia.*

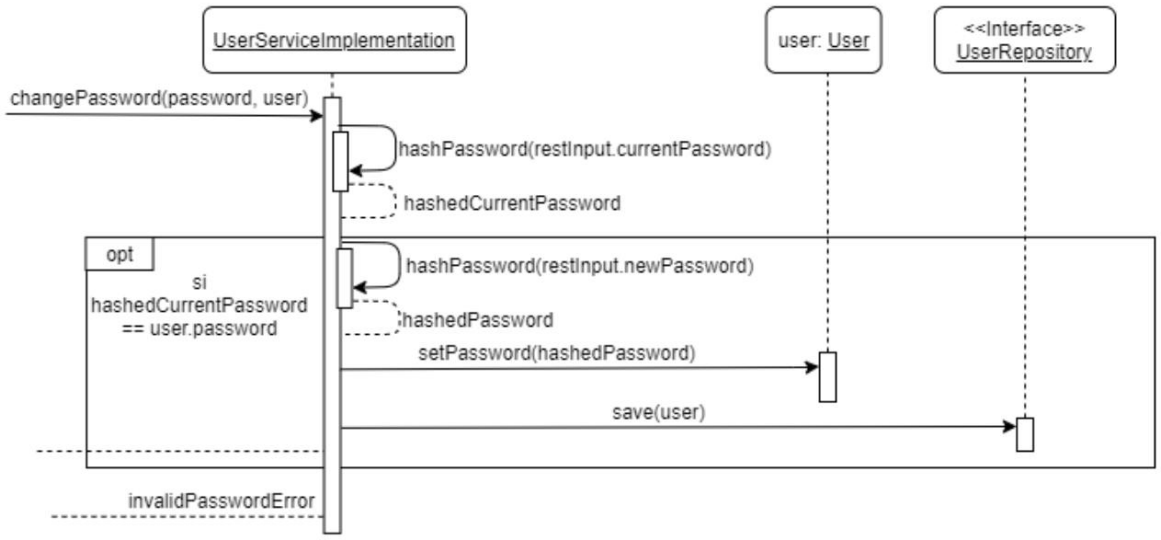



*Ilustración 31: Diagrama de secuencia del cambio de contraseña parte 2. Fuente: Elaboración propia.*

Tras realizar la petición del cambio de contraseña se inicia el proceso, el servicio se encarga de cifrar la contraseña recogida desde la aplicación (la original) y la compara con la que tiene guardada del usuario. En caso de que coincidan, se procede a cifrar la nueva contraseña y se le asigna al usuario, posteriormente se actualizan los datos del usuario en base de datos a través del repositorio.

En caso de que las contraseñas no coincidan, se devuelve un error (invalidPasswordError) esto nos permite mostrar un mensaje en la aplicación indicándole al usuario que ha habido un problema con la modificación de su contraseña.

### Modificación del idioma

Para que el usuario pueda disfrutar de la aplicación en su idioma nativo, se tiene que poder modificar el idioma por defecto (castellano). El proceso seguido para realizar este cambio es el siguiente:

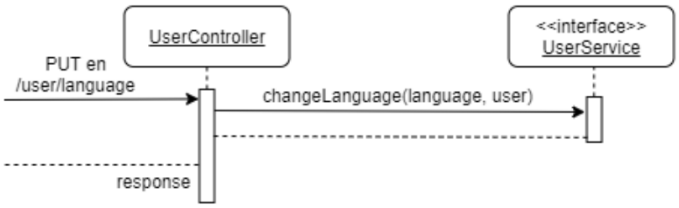



*Ilustración 32: Diagrama de secuencia del cambio de idioma parte 1. Fuente: Elaboración propia.*

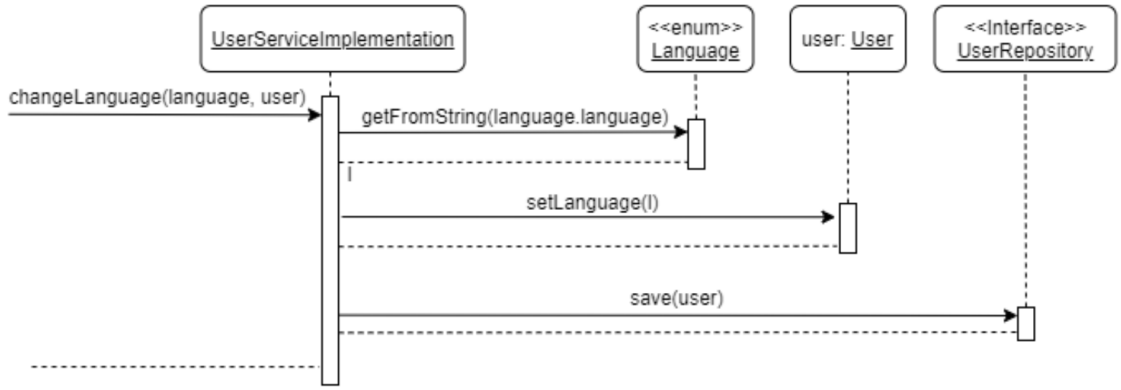



*Ilustración 33: Diagrama de secuencia del cambio de idioma parte 2. Fuente: Elaboración propia.*

Una vez el servicio tiene todos los datos necesarios para realizar este cambio, se mapean los datos obtenidos por parte de la aplicación de forma que la API los pueda entender, es por eso por lo que se realiza el "getFromString" sobre el *enum* Language. Una vez se tienen los datos correctamente mapeados, se asigna el nuevo idioma escogido al usuario y se guarda la información en base de datos a través del repositorio.

### Inicio de sesión

Cuando un usuario inicia sesión desde la aplicación, la API tiene que comprobar que los datos que ha introducido son correctos, es decir, que el DNI introducido coincide con la contraseña que el usuario indicó al registrarse. Esto se realiza mediante el proceso siguiente:

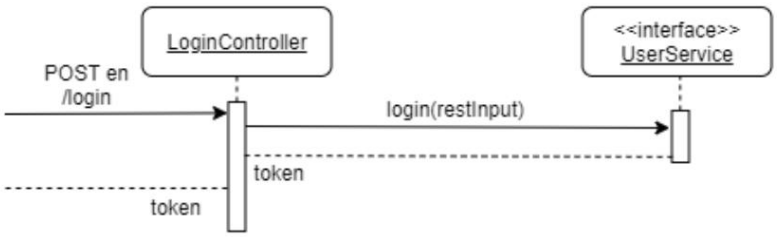



*Ilustración 34: Diagrama de secuencia del inicio de sesión parte 1. Fuente: Elaboración propia.*


*Ilustración 35: Diagrama de secuencia del inicio de sesión parte 2. Fuente: Elaboración propia.*

Una vez se reciben los datos, se intenta obtener el usuario mediante su identificador, en caso de que este exista se cifra la contraseña introducida en la pantalla de inicio de sesión y se compara con la del usuario (de forma equivalente a como se hace en la modificación de contraseña). Si coinciden, quiere decir que el usuario ha introducido los datos correctamente, en ese caso, se devuelve el token identificador del usuario. En caso contrario se lanza un error para indicar que los datos introducidos no son correctos, esto en si no da ninguna pista, pues realmente puede ser que la contraseña esté mal, o que la contraseña sea correcta pero no el identificador.

Finalmente, y para el caso en el que no se encuentre ningún usuario con el identificador proporcionado, se lanza un error distinto, que nos permitirá informar al usuario que tiene que registrarse antes de poder utilizar el sistema.

### Añadir análisis

Los análisis son una de las piedras angulares del sistema, para poder trabajar con ellos (procesarlos para extraer conclusiones), se tienen que poder añadir. Este proceso se inicia desde Postman, introduciendo los datos del análisis y la enfermedad que se está controlando con este. Una vez se envía la petición, el proceso que se sigue es el que podemos ver a continuación.


*Ilustración 36: Diagrama de secuencia de añadir una analítica parte 1. Fuente: Elaboración propia.*

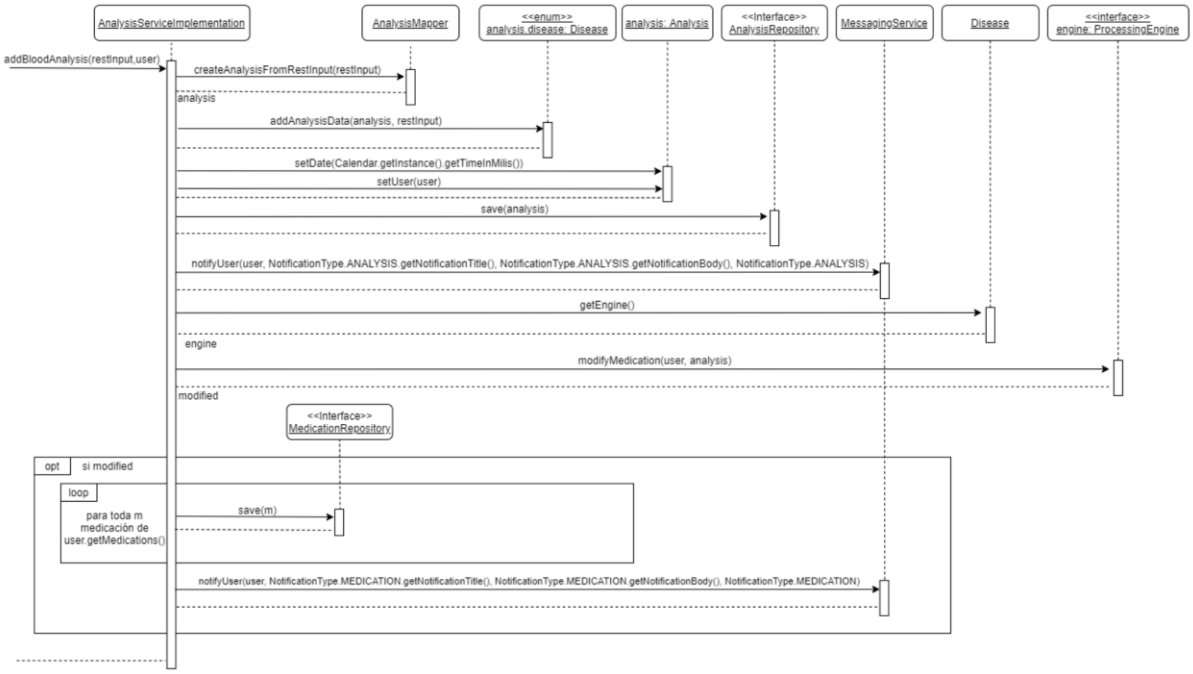



*Ilustración 37: Diagrama de secuencia de añadir una analítica parte 2. Fuente: Elaboración propia.*

Este es el proceso más importante del sistema, añadir una analítica implica todos los aspectos correspondientes a la adición de esta, pero, además, también entra en juego el procesado de analíticas y la posible generación de una nueva medicación.

En primer lugar, se mapea lo recibido de Postman y se obtiene un objeto Analysis, a este objeto se le añade la fecha actual y, además, se almacena el contenido de dicha analítica dentro de la enfermedad asociada a la misma, esto nos permite obtener un acceso directo a los distintos parámetros de la analítica de forma unificada, y sin la necesidad de tener una gran cantidad de clases para el control de este proceso (explicado en detalle en la sección 10.4).

En segundo lugar, se obtiene el motor de procesado asociado a la enfermedad que se está controlando en la analítica y se procede a verificar si se tiene que modificar la medicación, el método "modifyMedication" devuelve un booleano indicando si realmente se ha modificado o no dicha medicación, permitiéndonos de esta manera, saber cuándo tenemos que realizar el guardado en base de datos de la medicación del usuario. Es por eso por lo que, en caso de ser cierto, se recorren las medicaciones del usuario y se guardan actualizadas en base de datos a través del repositorio.

Finalmente, tenemos la parte de notificaciones, en este proceso se pueden llegar a enviar dos, una para el caso de la analítica (nueva analítica disponible) y otra para el caso en que se modifique la medicación.

### Obtener análisis

Cuando el usuario accede a la sección de analíticas en la aplicación, se lanza una petición para obtenerlas, el proceso que se sigue es este:

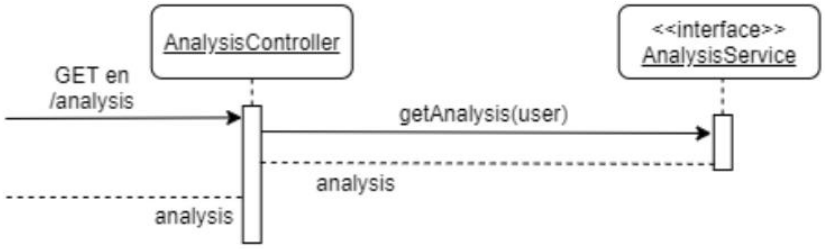



*Ilustración 38: Diagrama de secuencia de obtener análisis parte 1. Fuente: Elaboración propia.*

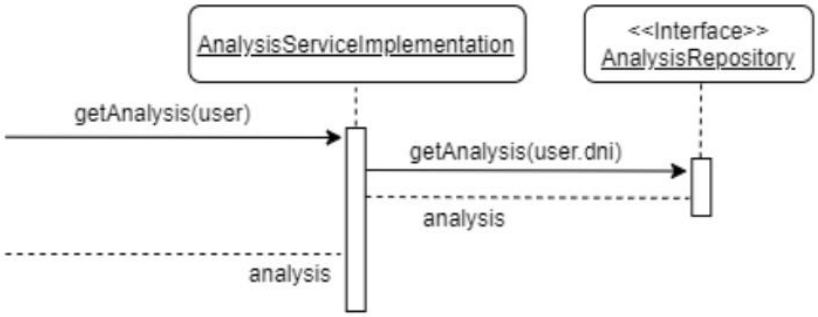



*Ilustración 39: Diagrama de secuencia de obtener análisis parte 2. Fuente: Elaboración propia.*

Como se puede observar, el proceso es muy simple, gracias a la inyección del usuario obtener las analíticas es tan sencillo como realizar una llamada al repositorio con el identificador de este, el repositorio nos devuelve una lista con todos los análisis del paciente y esto es lo que se devuelve.

### Medicación

Cuando se recibe una analítica, el sistema la procesa y comprueba que todos los parámetros estén dentro de los límites, en caso de no estarlo, se modifica la medicación del paciente para tratar de estabilizar estos parámetros alterados y que salgan dentro de los límites en la próxima analítica. Los diagramas de secuencia correspondientes al procesado de la medicación se pueden ver en el apartado 9.6.

Dentro de la aplicación, el usuario puede consultar su medicación, se envía una petición a la API y esta devuelve un listado con todas las medicaciones que toma el usuario. El proceso que se sigue es el siguiente:

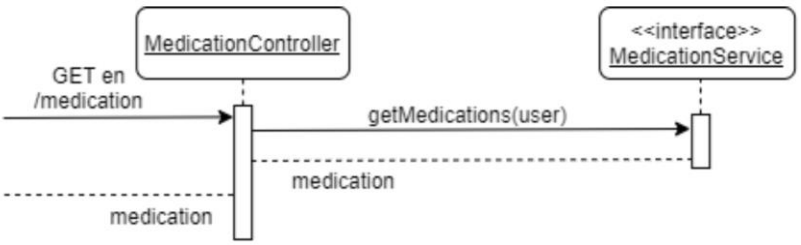



*Ilustración 40: Diagrama de secuencia de obtener medicación parte 1. Fuente: Elaboración propia.*

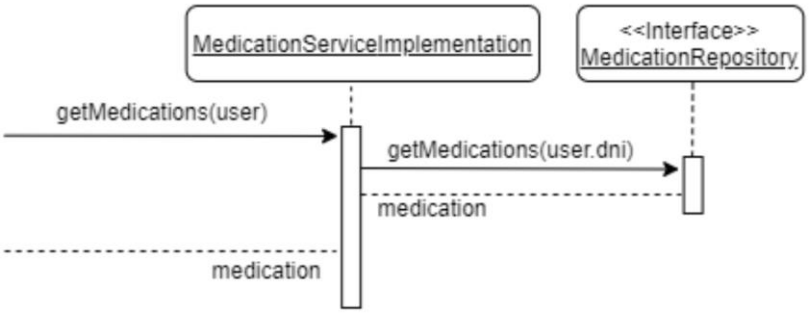



*Ilustración 41: Diagrama de secuencia de obtener medicación parte 2. Fuente: Elaboración propia.*

### Añadir cita

Para realizar los controles de la enfermedad del paciente, se pueden añadir citas. Toda esta información viene directa de Postman y se almacena en base de datos para posteriormente poder ser mostrada al usuario cuando este realice la petición desde la sección correspondiente de la aplicación.


*Ilustración 42: Diagrama de secuencia de añadir cita P1. Fuente: Elaboración propia.*

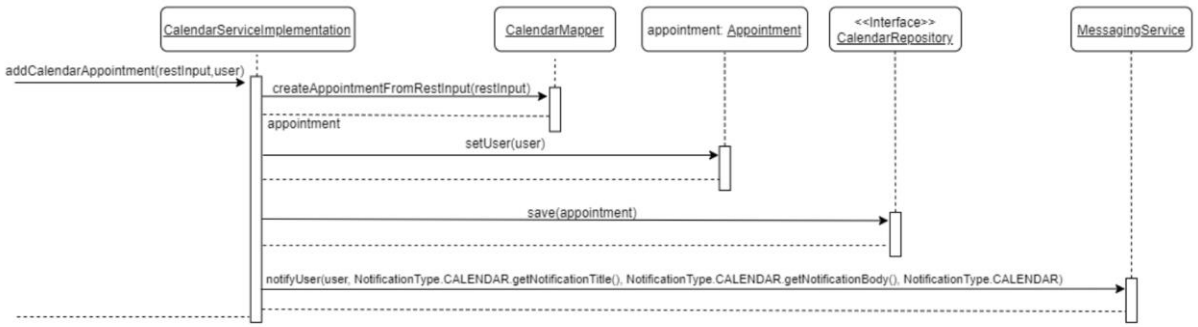



*Ilustración 43: Diagrama de secuencia de añadir cita Parte 2. Fuente: Elaboración propia.*

Una vez se recibe la petición en la API se mapea el objeto recibido a un objeto de modelo llamado Appointment. Adicionalmente, se le añade el usuario para posteriormente ser guardado en base de datos a través del repositorio correspondiente. Finalmente, se informa al usuario a través del MessagingService, el cual lanza una notificación del tipo indicado en la llamada realizada sobre el servicio.

### Obtener citas

Igual que ocurre con las analíticas y la medicación, cuando el usuario accede a la sección de calendario en la aplicación, se desencadena el siguiente proceso:

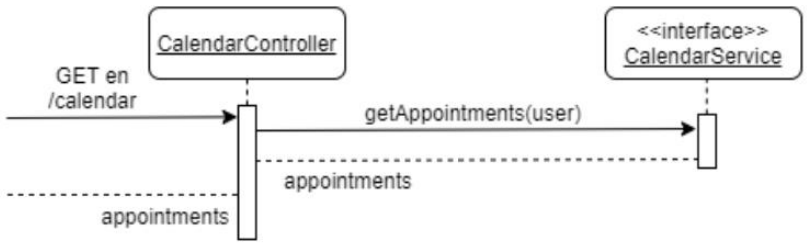



*Ilustración 44: Diagrama de secuencia de obtener cita parte 1. Fuente: Elaboración propia.*

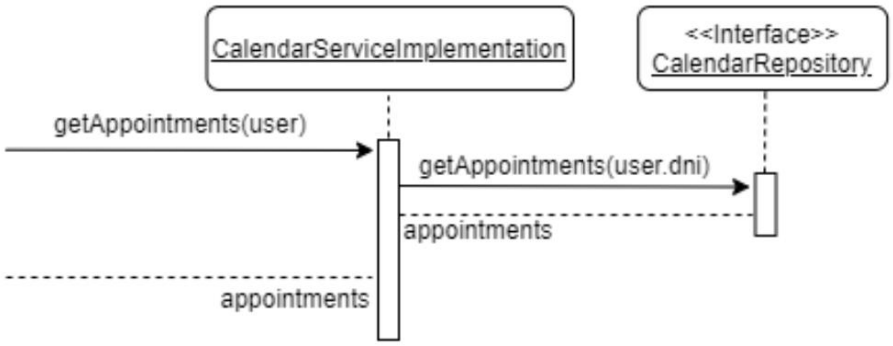



*Ilustración 45: Diagrama de secuencia de obtener cita parte 2. Fuente: Elaboración propia.*

Como podemos ver en las ilustraciones 44 y 45 es muy sencillo obtener las citas que tiene el usuario, con una simple llamada al repositorio se obtiene un listado de estas y es este listado el que se le devuelve a la aplicación.

### Chat

El caso del chat es particular, cuando un paciente inicia un chat, se envían todos los datos correspondientes a la API y esta lo crea. A continuación, cada vez que se envíe un mensaje sobre este chat, la API se encargará de irlos guardando. Estos mensajes pueden provenir del paciente (desde la aplicación) o de la enfermera (desde Postman), esta distinción es importante y nos permitirá luego mostrarlos a un lado u otro de la pantalla dependiendo del usuario que lo mande.

La obtención de chats se hace de forma equivalente al resto de objetos, se obtiene una lista y se muestra en formato tarjeta en la pantalla correspondiente. Si el usuario pulsa sobre uno en particular, es entonces cuando se obtienen todos los mensajes y se muestran a un lado u otro dependiendo del emisor de dicho mensaje.

### 9.4.3 Llamadas a la API

#### OBTENCIÓN DE USUARIO

| Método | Endpoint | Cabecera | Descripción |
|---|---|---|---|
| GET | /user | Authorization | Se obtiene un usuario simplificado con los datos mínimos necesarios. |

**Cuerpo:** -

**Respuestas**:

- 200 OK

```
{
    "mail": "sergio@gmail.com",
    "phoneNumber": 678652077,
    "language": "Castellano",
    "name": "Sergio"
}
```

- 401 Unauthorized

```
{
    "timestamp": "2021-04-25T15:18:54.398+0000",
    "status": 401,
    "error": "Unauthorized",
    "message": "401 UNAUTHORIZED",
    "path": "/user"
}
```

#### CAMBIAR CONTRASEÑA

| Método | Endpoint | Cabecera | Descripción |
|---|---|---|---|
| PUT | /user | Authorization | Se modifica la contraseña del usuario por la introducida. |

**Cuerpo:**

```
{
    "currentPassword": "somePassword",
    "password": "newPassowrd!!"
}
```

**Respuestas:**

- 200 OK
- 400 Bad Request

```
{
    "timestamp": "2021-04-25T15:35:36.306+0000",
    "status": 400,
    "error": "Bad Request",
    "message": "Password is not correct",
    "path": "/user"
}
```

- 401 Unauthorized

```
{
    "timestamp": "2021-04-25T15:34:31.961+0000",
    "status": 401,
    "error": "Unauthorized",
    "message": "401 UNAUTHORIZED",
    "path": "/user"
}
```

### INSERTAR TOKEN FCM

| Método | Endpoint | Cabecera | Descripción |
|---|---|---|---|
| PUT | /user/fcmtoken | Authorization | Se inserta un token FCM (mensajería) en el usuario. |

**Cuerpo:**

```
{
    "token": "someLongToken"
}
```

**Respuestas:**

- 200 OK
- 401 Unauthorized

```
{
    "timestamp": "2021-04-25T15:38:24.711+0000",
    "status": 401,
    "error": "Unauthorized",
    "message": "401 UNAUTHORIZED",
    "path": "/user/fcmtoken"
}
```

### MODIFICAR IDIOMA

| Método | Endpoint | Cabecera | Descripción |
|---|---|---|---|
| PUT | /user/language | Authorization | Se modifica el idioma del usuario. |

**Cuerpo:**

```
{
    "language": "Català"
}
```

**Respuestas:**

- 200 OK
- 401 Unauthorized

```
{
    "timestamp": "2021-04-25T15:41:05.555+0000",
    "status": 401,
    "error": "Unauthorized",
    "message": "401 UNAUTHORIZED",
    "path": "/user/language"
```

```
}
```

### REGISTRARSE

| Método | Endpoint | Cabecera | Descripción |
|---|---|---|---|
| POST | /register | - | Se crea un usuario con los datos introducidos. |

**Cuerpo:**

```
{
    "dni": "12321",
    "mail": "test@gmail.com",
    "name": "Lorem",
    "firstSurname" : "Ipsum",
    "secondSurname": "Dolor",
    "birthDate": "06/09/1997",
    "phoneNumber": "654785451",
    "language": "Castellano",
    "password": "123456789"
}
```

**Respuestas:**

- 200 OK

```
{
    "token": "eyJhbGciOiJIUzM4NCJ9.eyJpc3MiOiJBQUEiLCJpYXQiOjE2MTkzNjU1NzUsIn
    N1YiI6IjEyMzIxIiwianRpIjoiNjYwNjM2ZjUtNWFmNS00ODRiLTg0NmUtMGY2MmM4MmQzYzQ
    yIn0.EgXczw7vKFb85eiEFs26iavw9ozuE9h9li9ulEneyiCChZGOEy9G2AbcEvQ_BUBN"
 }
```

- 400 Bad Request

```
{
    "timestamp": "2021-04-25T15:51:47.964+0000",
    "status": 400,
    "error": "Bad Request",
    "message": "El usuario ya existe",
    "path": "/register"
}
```

### INICIAR SESIÓN

| Método | Endpoint | Cabecera | Descripción |
|---|---|---|---|
| POST | /login | - | Se devuelve el token del usuario que está iniciando sesión. |

**Cuerpo:**

```
{
    "dni": "12321",
    "password": "123456789"
}
```

Respuestas:

- 200 OK


```
{
    "token": "eyJhbGciOiJIUzM4NCJ9.eyJpc3MiOiJBQUEiLCJpYXQiOjE2MTkzNjU4MzYsInN1Y
iI6IjEyMzIxIiwianRpIjoiZWQzOTQ2Y2UtYjVlZS00OGE4LWFhODgtNDQ5NzExMDMwZmFmIn0.ztmAD
_Gppl-lzujlrqEfun1gMSvMyybM7-FNyIx4udK7skKYBKTTihvLEaJ2vlTw"
}
```


- 400 Bad Request


```
{
    "timestamp": "2021-04-25T16:03:40.693+0000",
    "status": 400,
    "error": "Bad Request",
    "message": "Password is not correct",
    "path": "/login"
}
```


- 404 Not Found


```
{
    "timestamp": "2021-04-25T16:04:22.619+0000",
    "status": 404,
    "error": "Not Found",
    "message": "User not found",
    "path": "/login"
}
```


## AÑADIR ANALÍTICA

| Método | Endpoint | Cabecera | Descripción |
|---|---|---|---|
| POST | /analysis | Authorization | Se añade una nueva analítica. |

Cuerpo:


```
{
    "disease": "HYPERCHOLESTEROLEMIA",
    "cldl": "50",
    "totalCholesterol": "150"
}
```


- 200 OK
- 401 Unauthorized


```
{
    "timestamp": "2021-04-25T16:11:59.038+0000",
    "status": 401,
    "error": "Unauthorized",
    "message": "401 UNAUTHORIZED",
    "path": "/analysis"
}
```

## OBTENER ANALÍTICAS

| Método | Endpoint | Cabecera | Descripción |
|---|---|---|---|
| GET | /analysis | Authorization | Se obtienen las analíticas del usuario. |

**Cuerpo: -**

**Respuestas:**

- 200 OK


```
[
    {
        "id": 132,
        "disease": "HYPERCHOLESTEROLEMIA",
        "date": 1619367254678,
        "analysisData": "{\"totalCholesterol\":\"150\",\"maxCLDL\":\"100\",\"max
TotalCholesterol\":\"200\",\"cldl\":\"50\"}"
    },
    {
        "id": 133,
        "disease": "HYPERCHOLESTEROLEMIA",
        "date": 1619367275982,
        "analysisData": "{\"totalCholesterol\":\"170\",\"maxCLDL\":\"100\",\"max
TotalCholesterol\":\"200\",\"cldl\":\"70\"}"
    }
]
```


- 401 Unauthorized


```
{
    "timestamp": "2021-04-25T16:19:40.854+0000",
    "status": 401,
    "error": "Unauthorized",
    "message": "401 UNAUTHORIZED",
    "path": "/analysis"
}
```


## AÑADIR CITA

| Método | Endpoint | Cabecera | Descripción |
|---|---|---|---|
| POST | /calendar | Authorization | Se añade una nueva cita. |

**Cuerpo:**


```
{
    "disease": "HYPERCHOLESTEROLEMIA",
    "date": "06/09/1997",
    "time": "12:30",
    "location": "Hospital Clinic"
}
```

Respuestas:

- 200 OK
- 401 Unauthorized


```
{
    "timestamp": "2021-04-25T16:22:37.740+0000",
    "status": 401,
    "error": "Unauthorized",
    "message": "401 UNAUTHORIZED",
    "path": "/calendar"
}
```


## OBTENER CITAS

| Método | Endpoint | Cabecera | Descripción |
|---|---|---|---|
| GET | / calendar | Authorization | Se obtienen las citas del usuario. |

Cuerpo: -

Resultados:

- 200 OK


```
[
    {
        "id": 134,
        "disease": "HYPERCHOLESTEROLEMIA",
        "date": 873541800000,
        "location": "Hospital Clinic"
    }
]
```


- 401 Unauthorized


```
{
    "timestamp": "2021-04-25T16:22:37.740+0000",
    "status": 401,
    "error": "Unauthorized",
    "message": "401 UNAUTHORIZED",
    "path": "/calendar"
}
```

OBTENER MEDICACIÓN

| Método | Endpoint | Cabecera | Descripción |
|---|---|---|---|
| GET | / calendar | Authorization | Se obtienen las medicaciones del usuario. |

Cuerpo: -

Resultados:

- 200 OK


```
[
    {
        "id": 131,
        "medicines": [
            {
                "id": 130,
                "name": "Eutirox",
                "dose": 75.0
            }
        ],
        "disease": "Hipotiroidismo"
    }
]
```


- 401 Unauthorized


```
{
    "timestamp": "2021-04-25T16:27:35.163+0000",
    "status": 401,
    "error": "Unauthorized",
    "message": "401 UNAUTHORIZED",
    "path": "/medication"
}
```


OBTENER CHATS

| Método | Endpoint | Cabecera | Descripción |
|---|---|---|---|
| GET | / chat | Authorization | Se obtienen los chats del usuario. |

Cuerpo: -

Resultados:

- 200 OK


```
[
    {
        "id": 195,
        "nursingSpeciality": "Mi enfermedad crónica",
        "lastMessageContent": "bien"
    }
]
```

- 401 Unauthorized


```
{
    "timestamp": "2021-06-16T08:25:48.894+0000",
    "status": 401,
    "error": "Unauthorized",
    "message": "",
    "path": "/chat"
}
```


### CREAR CHAT

| Método | Endpoint | Cabecera | Descripción |
|---|---|---|---|
| POST | /chat | Authorization | Se crea un nuevo chat. |

**Cuerpo:**


```
{
    "type" : "Mi enfermedad crónica",
    "description": "Siento nauseas, que puedo tomar?"
}
```


**Resultados:**

- 200 OK
- 401 Unauthorized


```
{
    "timestamp": "2021-06-16T08:29:02.091+0000",
    "status": 401,
    "error": "Unauthorized",
    "message": "",
    "path": "/chat"
}
```

ENVIAR MENSAJE

| Método | Endpoint | Cabecera | Descripción |
|---|---|---|---|
| POST | /chat/message | Authorization | Se añade un nuevo mensaje al chat. |

Cuerpo:

```
{
    "chatId" : 27,
    "content": "no te preocupes, todo va bien"

}
```

Resultados:

- 200 OK
- 401 Unauthorized

```
{
    "timestamp": "2021-06-16T08:19:05.002+0000",
    "status": 401,
    "error": "Unauthorized",
    "message": "",
    "path": "/chat/message"
}
```

### 9.4.4 Conexión con la base de datos

La base de datos está publicada en AWS, y, para que nuestra API se pueda comunicar con ella se tiene que realizar una configuración específica. Gracias a que estamos utilizando Spring, esta configuración es bastante simple y solo se requiere de la adición de unos parámetros en el archivo application.yml. En la ilustración 46 lo podemos ver.


*Ilustración 46: Configuración de la base de datos. Fuente: Elaboración propia.*

Dentro de *datasource* tenemos todos los parámetros a configurar para que dicha conexión sea exitosa, como podemos ver, contiene simplemente la url (con el puerto y la base de datos), y los datos de acceso a dicha base de datos.

También podemos ver en la imagen que los parámetros de configuración están duplicados, esto es debido a que inicialmente, la base de datos se encontraba alojada localmente (para un mejor manejo de los datos durante la implementación), y una vez se terminó todo el desarrolló se alojó en los servidores de Amazon.

Cuando se realiza una ejecución, Spring lee estos datos y realiza de forma automática la conexión con la base de datos, cosa que convierte este proceso en algo totalmente automático.

## 9.5 Patrones utilizados

Los patrones de diseño son muy útiles a la hora de resolver distintos problemas que pueden surgir durante la fase de desarrollo de un sistema software, o bien para mejorar la calidad de la implementación. En esta sección se van a enumerar y a explicar brevemente los utilizados a lo largo de la implementación del sistema que se propone. Toda la información referente a los patrones expuestos a continuación ha sido extraída del libro *Design patterns: elements of reusable object-oriented software* [33].

### 9.5.1 Factoría

El patrón factoría es utilizado para evitar la instanciación concreta de una clase en particular, es decir evitarnos llamar al constructor de una clase directamente. Para lograr esto, se definen un conjunto de clases que llamaremos factorías, como se puede ver en la ilustración 47. En primer lugar, existe una clase factoría abstracta la cual se encargará de delegar la creación del objeto a una de sus subclases, y estas serán las encargadas de crear finalmente el objeto que será devuelto al objeto que realizó la llamada inicial.

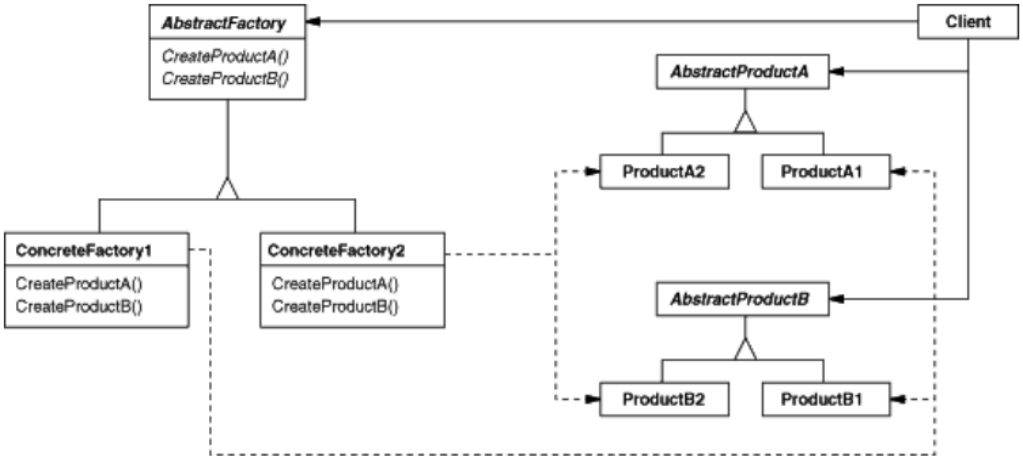



*Ilustración 47: Estructura del patrón factoría. Fuente: Design Patterns: Elements of Reusable Object-Oriented Software.*

A lo largo de la implementación se ha utilizado varias veces este patrón, sobre todo en la aplicación móvil, ya que toda la parte de creación de vistas y Viewmodels se realiza automáticamente mediante llamadas a unas factorías autogeneradas por el *framework* Dagger (explicado más adelante).

```
class ViewModelFactory @Inject constructor(private val viewModels: MutableMap<Class<out ViewModel>,
Provider<ViewModel>>) :
    ViewModelProvider.Factory {

    override fun <T : ViewModel> create(modelClass: Class<T>): T =
        viewModels[modelClass]?.get() as T
}
```

*Ilustración 48: Ejemplo de una factoría utilizada en la aplicación móvil para la creación de Viewmodels. Fuente: Elaboración propia.*

### 9.5.2 Observador

El uso de este patrón nace de la necesidad de "escuchar" a un objeto, es decir, mantener una conexión entre dos o más objetos para que cuando uno cambie, el otro se entere.

La arquitectura utilizada en la aplicación requiere de este patrón casi de forma obligatoria, pues es la manera en la que la actividad o el fragmento se sincronizan con sus Viewmodel correspondientes. Gracias al uso de los LiveData y los MutableLiveData.

```
viewModel.analysis.observe(viewLifecycleOwner, Observer {
    populateAnalysis(it)
})
```

*Ilustración 49: Ejemplo de uso del patrón observador. Fuente: Elaboración propia.*

El código de la ilustración 49 está situado en AnalysisFragment, gracias a esto, en cuanto se reciben los datos de la API (a través del repositorio conectado con el viewmodel) aparecen los análisis de forma automática en la aplicación.

### 9.5.3 Plantilla

A lo largo del desarrollo de un sistema software, surge la necesidad de reutilizar código en distintas partes. Si cogemos este código que se repite, y lo situamos en una clase abstracta de la cual heredarán todas las que necesiten utilizar dicho código, nos evitamos repetir una gran cantidad de líneas, dejando el código mucho más limpio. Además, en caso de no querer utilizar todos los métodos propuestos por la plantilla, siempre podemos reimplementar alguno y utilizarlo en lugar del propuesto en la clase de la que se hereda.

Adicionalmente, a parte de la reutilización de código, también es muy útil para marcar que una clase tenga que seguir esa plantilla, es decir, que tenga que implementar todos los métodos propuestos en la plantilla que se propone.

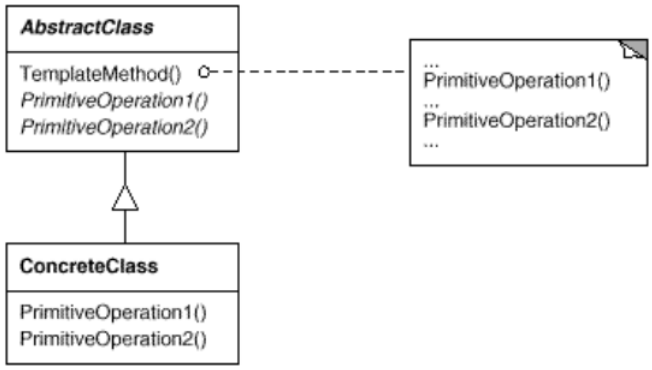



*Ilustración 50: Estructura del patrón plantilla. Fuente: Design Patterns: Elements of Reusable Object-Oriented Software.*

El patrón plantilla también es muy utilizado en el sistema, tanto en la aplicación, en clases como AbstractActivity, AbstractDrawerFragment, como en la API, con interfaces como ProcessingEngine, ChatService, CalendarService entre otros.

Estas clases nos permiten dejar el código mucho más limpio, o, en el caso de ChatService, por ejemplo, obligar a todas las clases que implementen dicha interfaz a implementar los métodos propuestos en esta.

```java
public interface ChatService {
    public ResponseEntity<Chat> createChat(ChatRestInput chatRestInput, final User user);

    public ResponseEntity<Void> createMessage(MessageRestInput messageRestInput, final User user);

    public ChatRestOutput getChat(Long id);

    public List<SimplifiedChat> getChats(final User user);
}
```

*Ilustración 51: Ejemplo de uso del patrón plantilla en la API. Fuente: Elaboración propia.*

### 9.5.4 Singleton

En muchos casos, existe la necesidad de que una clase solo tenga una instancia, es decir, que la clase en cuestión se cree una sola vez. Esto lo podemos conseguir mediante el uso del patrón Singleton, el cual consiste en declarar el constructor de una clase como privado, y, por lo tanto, instanciar dicha clase mediante un método estático (acceso global) getInstance(). Este método, en caso de que sea la primera vez que se llama (instance == null), creará la única instancia que tendrá, en caso de que instance ya se haya creado, devuelve dicha instancia.

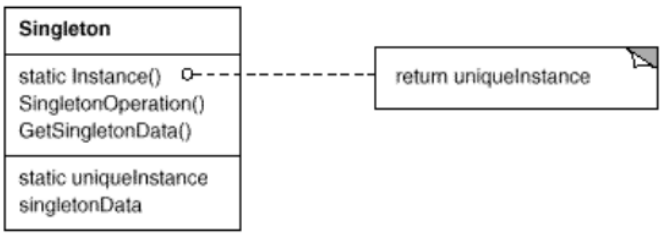



*Ilustración 52: Estructura del patrón singleton. Fuente: Design Patterns: Elements of Reusable Object-Oriented Software.*

El patrón Singleton ha sido utilizado repetidas veces a lo largo de la implementación, sobre todo para el caso de los repositorios (acceso a datos). Estos repositorios están marcados con la etiqueta @Singleton, lo cual le indica a Dagger que estas clases solo se tienen que instanciar una única vez.

```
@Singleton
class AnalysisRepository @Inject constructor(val api: ApiService, val context: Context) {
    private lateinit var preferences: SharedPreferences;
    var token: String

    init {
        preferences =
            PreferenceManager.getDefaultSharedPreferences(context)

        token = preferences.getString("token", "")!!
    }

    suspend fun getAnalysis(): List<Analysis> {
        return api.getAnalysis(token)
    }
}
```

*Ilustración 53: Ejemplo de uso del patrón singleton en la aplicación. Fuente: Elaboración propia.*

### 9.5.5 Repository

Para acceder a los datos, tanto desde la aplicación (llamadas a la API), como desde la API (accesos a base de datos), se utilizan unas estructuras llamadas repositorios. Estos nos permiten separar la lógica que recupera los datos, es decir introducir una capa entre el dominio y los datos per se.

Este patrón se utiliza en muchas ocasiones, tanto en la aplicación como en la API existen clases repositorio para cada tipo de funcionalidad que existe. Por ejemplo, tenemos el AnalysisRepository, CalendarRepository, entre otros, y estos se repiten tanto en la aplicación como en la API.

```java
public interface MedicineRepository extends JpaRepository<Medicine, Long> {
    //All the accessing methods are provided by JpaRepository
}
```

*Ilustración 54: Ejemplo de uso del patrón repositorio en la API. Fuente: Elaboración propia.*

### 9.5.6 Estrategia

El patrón estrategia nos permite ejecutar un código u otro dependiendo de las necesidades que tengamos en tiempo de ejecución. Se define una estrategia abstracta a la cual invocaremos, y desde aquí se derivará a una estrategia concreta u otra dependiendo de las necesidades actuales.

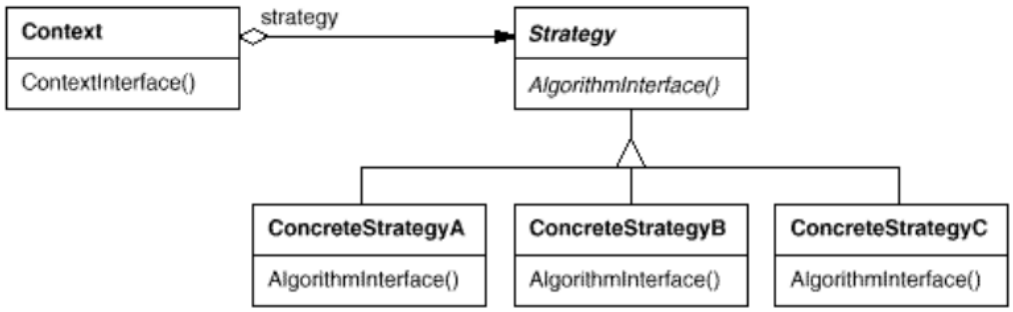



*Ilustración 55: Estructura del patrón estrategia. Fuente: Design Patterns: Elements of Reusable Object-Oriented Software.*

Este patrón se utiliza para el caso del procesado de analíticas, cada enfermedad tiene una estrategia de procesado distinto, es por eso por lo que cuando se recibe una analítica se extrae la enfermedad que se está controlando y se elige un motor de procesado u otro dependiendo del tipo de enfermedad que se está tratando.

## 9.6 Procesado de analíticas

Este es un proyecto puramente académico, a pesar de esto, se ha contactado con personal médico real para llevar a cabo un análisis exhaustivo de las enfermedades que se pueden llegar a tratar. De entre todas ellas, se escogieron el hipotiroidismo y la hipercolesterolemia. La elección de estas no fue trivial, pues se preguntó a un grupo de estudiantes de medicina en su cuarto año cuales serían las que elegirían ellos, y su elección fue mayoritariamente la hipercolesterolemia, por la gran cantidad de pacientes que la sufren. En cuanto al hipotiroidismo, fue más personal, pues tanto la pareja del autor como la madre sufren de esta enfermedad crónica.

Para el procesado de las analíticas de control de estas enfermedades se necesitó la ayuda de las doctoras Natividad Pacheco Rubio y Àngels Escorsell. Ellas han sido quienes han aportado la información necesaria para el control y tratamiento de las enfermedades escogidas.

Para la realización de este proceso, se han diseñado e implementado unos motores de procesado, que se encargan de recibir los datos de la analítica y del usuario y deciden si tienen que subir su medicación, bajarla o simplemente no hacer nada.

En la ilustración 56 podemos ver una interfaz llamada ProcessingEngine, esta interfaz es implementada por los dos motores de procesado que existen en este momento en el sistema.

```java
public interface ProcessingEngine {
    public List<Medicine> generateMedication(BaseAnalysis analysis);

    public void increaseMedication(List<Medication> medication);

    public void decreaseMedication(List<Medication> medication);

    public boolean modifyMedication(User user, BaseAnalysis analysis);

    public double getNextDose(int currentDose);

    public double getPreviousDose(int currentDose);
}
```

*Ilustración 56: Interfaz implementada por los motores de procesado. Fuente: Elaboración propia.*

Los métodos que se pueden ver en la ilustración son los siguientes:

- generateMedication(BaseAnalysis analysis) → Este método es exclusivamente de prueba, al tratarse de un proyecto académico, cuando se da de alta a un usuario a este se le asigna una enfermedad y una medicación acorde con dicha enfermedad. Este método sirve para automatizar este proceso.
- increaseMedication(List<Medication> medication) → Este método se utiliza para el caso en que haya que subirle la medicación al usuario, se recorre su listado de medicaciones y se modifica la única que coincide con la enfermedad que se está tratando.

- decreaseMedication(List<Medication> medication) → Es el caso contrario al anterior, este método es utilizado para decrementar la medicación del usuario, y, al igual que en el caso anterior, también se recorre el listado para modificar la medicación asociada a la enfermedad que se está controlando.
- modifyMedication(User user, BaseAnalysis analysis) → Este es el método invocado cuando se procesa una analítica, en este método se decide si se tiene que modificar o no la medicación, y en caso de que haga falta se decide si se tiene que incrementar o disminuir. Una vez tomada la decisión, se llama a uno de los métodos anteriores (*increase* o *decrease*) para que se encarguen de la modificación de la medicación del paciente.
- getNextDose(int currentDose) → Cuando se tiene que incrementar la medicación, se llama a este método pasándole la medicación actual para que nos devuelva la siguiente que se le tiene que asignar al paciente.
- getPreviousDose(int currentDose) → En el caso que se tenga que disminuir la medicación, se hace uso de este método para obtener una dosis más pequeña (o distinta) para tratar de controlar la enfermedad del paciente.

### 9.6.1 Hipercolesterolemia

El procesado de las analíticas de control de la hipercolesterolemia se basa principalmente en el control del colesterol total y el cLDL[1]. La tabla siguiente nos muestra los objetivos terapéuticos en el tratamiento de la hipercolesterolemia.

*Tabla 10: Tratamiento de la hipercolesterolemia. Objetivo terapéutico. Fuente: Elaboración propia con datos proporcionados por la doctora Natividad Pacheco.*

| | Control del cLDL mg/dl (mmol/l) | |
|---|---|---|
| | Buen control | Mal control |
| Prevención primaria: riesgo coronario > 10% | < 130 | ≥ 130 |
| Prevención secundaria (y diabéticos con nefropatía) | < 100 | ≥ 100 |

Cuando nos referimos a prevención primaria, es cuando se trata de prevenir un evento antes de que suceda por primera vez, por ejemplo, cuando tratamos con un paciente con riesgo cardiovascular, se le medica para tratar de evitar un posible infarto. Por otro lado, respecto a la prevención secundaria, se trata de realizar una acción para evitar que dicho evento vuelva a ocurrir, haciendo referencia al mismo ejemplo, la prevención secundaria en el caso del infarto constaría de medicar al paciente para que no le vuelva a suceder.

[1] cLDL: lipoproteínas de baja densidad, se trata del colesterol "malo" ya que un nivel alto de LDL lleva a una acumulación de colesterol en las arterias.

*Tabla 11: Porcentajes de reducción media de cLDL necesarios para alcanzar objetivos terapéuticos. Fuente: Elaboración propia con datos proporcionados por la doctora Natividad Pacheco.*

| Reducción media de colesterol LDL para alcanzar objetivos terapéuticos (se han sombreado los objetivos presumiblemente alcanzables con monoterapia[2]) | | |
|---|---|---|
| cLDL (mg/dl) | OBJETIVO: 130 mg/dl | OBJETIVO: 100mg/dl |
| 300 | ↓ 56% | ↓ 65% |
| 290 | ↓ 55% | ↓ 65% |
| 280 | ↓ 53% | ↓ 64% |
| 270 | ↓ 51% | ↓ 62% |
| 260 | ↓ 50% | ↓ 61% |
| 250 | ↓ 48% | ↓ 60% |
| 240 | ↓ 45% | ↓ 58% |
| 230 | ↓ 43% | ↓ 56% |
| 220 | ↓ 40% | ↓ 54% |
| 210 | ↓ 38% | ↓ 52% |
| 200 | ↓ 35% | ↓ 50% |
| 190 | ↓ 31% | ↓ 47% |
| 180 | ↓ 27% | ↓ 44% |
| 170 | ↓ 23% | ↓ 41% |
| 160 | ↓ 18% | ↓ 37% |
| 150 | ↓ 13% | ↓ 33% |
| 140 | ↓ 7% | ↓ 28% |
| 130 | - | ↓ 23% |
| 120 | - | ↓ 16% |

En la tabla 11 podemos ver los porcentajes de bajada en el cLDL necesarios para controlar la hipercolesterolemia. Estos son los objetivos que se siguen hoy en día a nivel médico y por lo tanto son los que tendrá que seguir el sistema.

Para el tratamiento de la hipercolesterolemia se utilizan unos fármacos llamados estatinas [34], estos fármacos sirven para disminuir el nivel de colesterol. En la tabla siguiente podemos ver diferentes tipos de estatinas, con diferentes dosis, y el tanto por ciento de colesterol que consiguen reducir.

*Tabla 12: Descensos previsibles del cLDL con distintas dosis de estatinas. Fuente: Elaboración propia con datos proporcionados por la doctora Natividad Pacheco.*

| Descensos previsibles del colesterol LDL con diversas dosis de estatinas | | | | | |
|---|---|---|---|---|---|
| **ESTATINA** | **27%** | **34%** | **41%** | **48%** | **55%** |
| Pravastatina | 20 mg | 40 mg | | | |
| Lovastatina | 20 mg | 40 mg | | | |
| Simvastatina | 10 mg | 20 mg | 40 mg | | |
| Atorvastatina | | | | 40 mg | 80 mg |

[2] Monoterapia: Implica que solo se utiliza un fármaco para el tratamiento de la enfermedad.

La tabla 12 es clave para el desarrollo del sistema, en ella se nos indica que dosis hay que asignar para descender un porcentaje determinado de cLDL. El tipo de estatina a asignar es irrelevante para este proyecto, pues la medicación ya viene asignada por parte del médico de cabecera, el sistema solo se encarga de subir o mantener las dosis de dicha medicación dependiendo del resultado de la analítica.

En caso de necesitar descender menos de un 27%, se utiliza de todos modos la dosis mínima, de esta forma se consigue antes el objetivo necesario y contra más bajo del límite esté el colesterol es mejor para la salud del paciente.

Durante el tratamiento de la hipercolesterolemia, puede suceder que para aumentar la dosis se tenga que cambiar de medicamento, por ejemplo, si estamos tomando Lovastatina de 40mg y necesitamos aumentar esta dosis debido a que nuestro cLDL sigue siendo elevado a pesar de estar tomando dicha medicación, tendríamos que pasar a tomar una estatina que nos permitiera descender aún más los niveles de cLDL. Para el cambio de medicamento se tiene que consultar con un médico, y, por lo tanto, en caso de que no existan dosis superiores a la que está tomando el paciente, se le notifica indicándole que tiene que contactar con su médico de cabecera para que le modifique el tipo de medicación.

A continuación, en las ilustraciones 57 y 58 podemos ver los diagramas de secuencia que corresponden al procesado de la hipercolesterolemia en su motor de procesado.

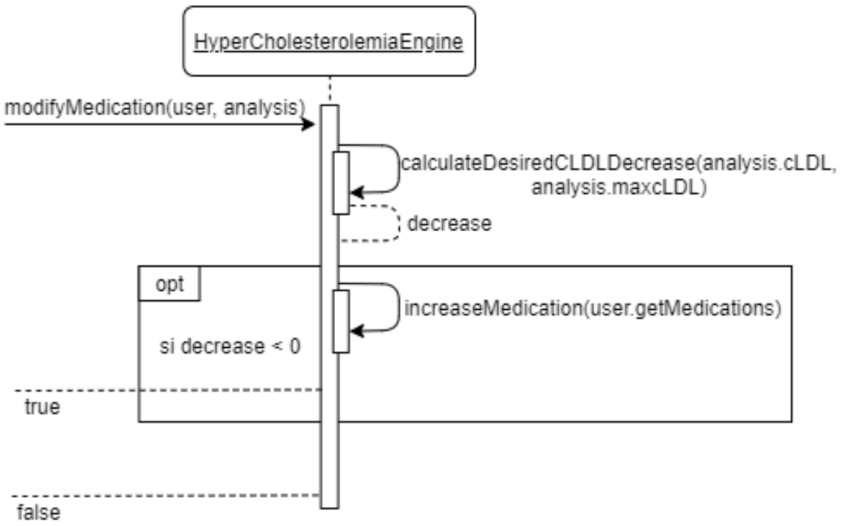



*Ilustración 57: Diagrama de secuencia del procesado de la hipercolesterolemia P1. Fuente: Elaboración propia.*

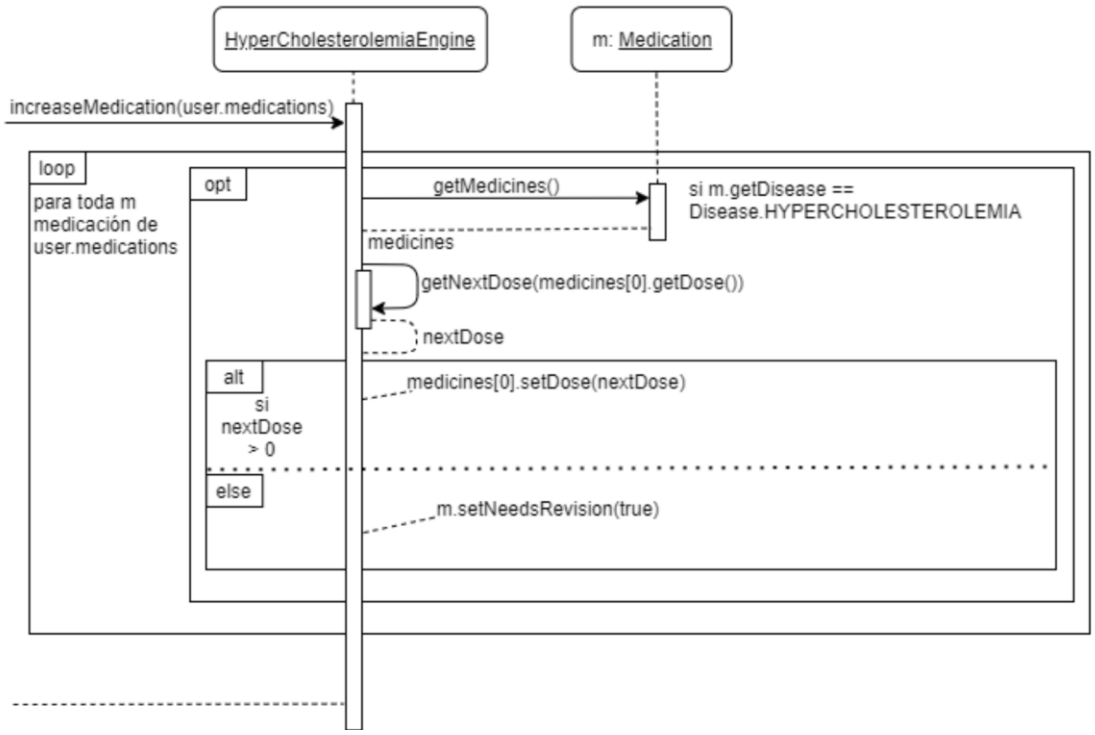



*Ilustración 58: Diagrama de secuencia del procesado de la hipercolesterolemia P2. Fuente: Elaboración propia.*

### 9.6.2 Hipotiroidismo

Para el buen control del hipotiroidismo se tiene que mirar que los valores de TSH[3] se encuentren entre el máximo (4.78 µUI/mL) y el mínimo (0.55 µUI/mL). En este caso todo se simplifica bastante, pues al venir la dosis ya asignada por defecto, el sistema simplemente tiene que incrementarla o disminuirla dependiendo de si nos encontramos por encima del máximo o por debajo del mínimo.

*Tabla 13: Dosis existentes de Eutirox. Fuente: Elaboración propia con datos proporcionados por la doctora Natividad Pacheco.*

| Dosis de Eutirox | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 25 | 50 | 75 | 88 | 100 | 112 | 123 | 137 | 150 | 175 | 200 |

Para el tratamiento del hipotiroidismo solo se utiliza un fármaco, el Eutirox, y está disponible en las dosis indicadas en la tabla 13. En el caso de estar por encima del máximo, se pasará a recetar la dosis siguiente disponible de Eutirox, y en el caso contrario la dosis anterior. Por ejemplo, en caso de estar tomando 88mg de Eutirox, y obtener un valor de TSH en la analítica de 5.37 µUI/mL, se pasaría a una dosis de 100mg.

En los diagramas de secuencia que se pueden ver en las ilustraciones 59 y 60 se ve el proceso que siguen los motores de procesado para modificar la medicación en pacientes con hipotiroidismo.

[3] TSH: Hormona estimulante de la glándula tiroides.

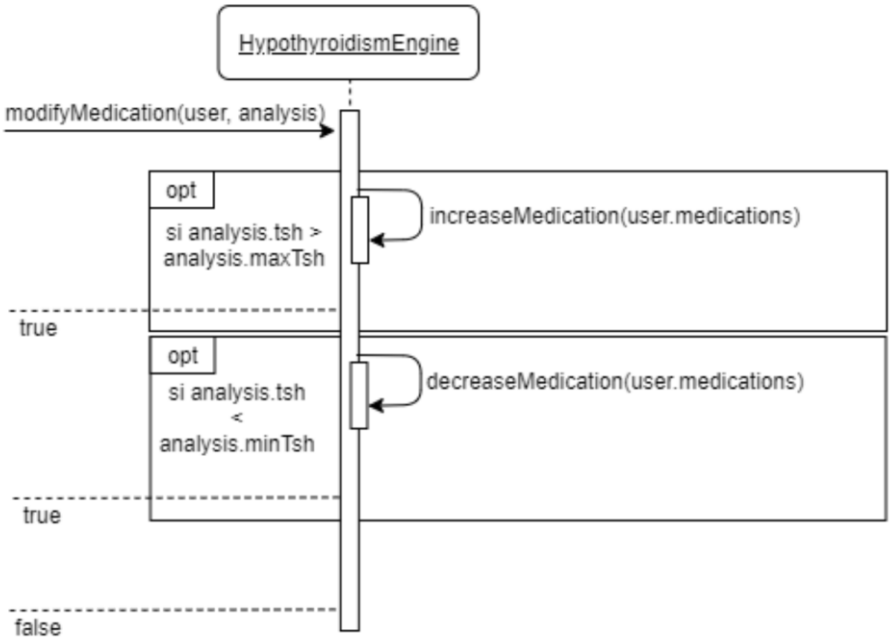



*Ilustración 59: Diagrama de secuencia del procesado hipotiroidismo P1. Fuente: Elaboración propia.*

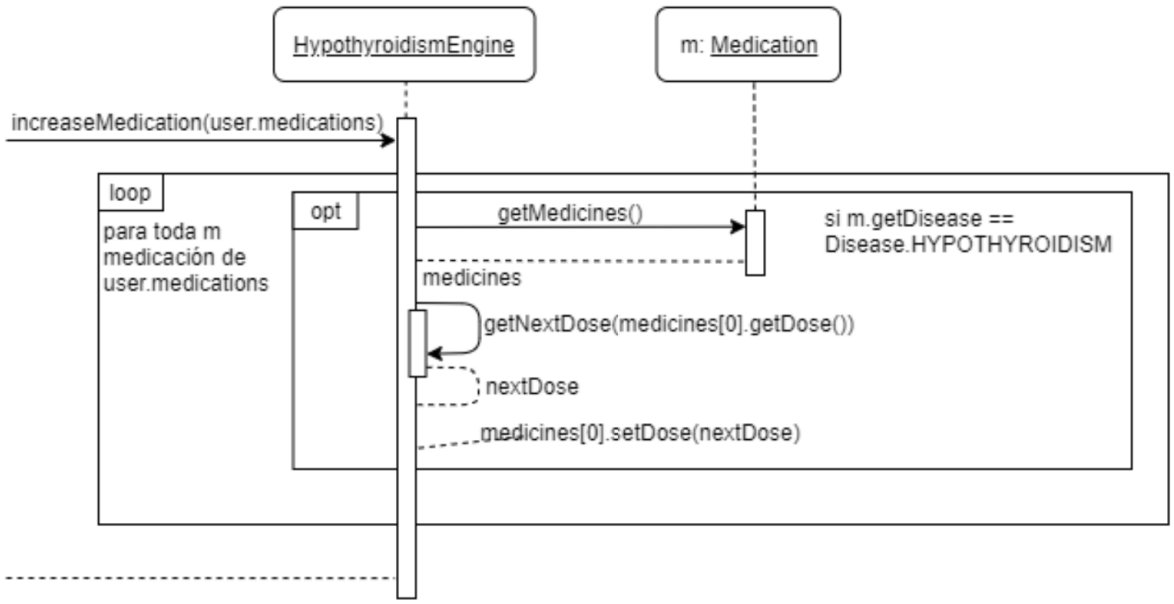



*Ilustración 60: Diagrama de secuencia del procesado hipotiroidismo P2. Fuente: Elaboración propia.*

En este caso, podemos ver el proceso para aumentar la medicación, el caso contrario (reducirla) es equivalente al de la ilustración 60 solo que se obtiene la dosis anterior en lugar de la siguiente.

# 10. Implementación

Antes de comenzar con la implementación, se creó un proyecto en JIRA. En la ilustración 61 se puede ver el *backlog* inicial del proyecto.


*Ilustración 61: Backlog inicial. Fuente: Elaboración propia.*

## 10.1 Proceso de desarrollo del producto

A continuación, se va a realizar un breve resumen de los cuatro *sprints* que han compuesto este proyecto, explicando brevemente como se han realizado las tareas, en que orden, los problemas que han ido saliendo, las reuniones realizadas y los avances hechos en cada *sprint.*

### 10.1.1 Sprint 1

El primer sprint, se llevó a cabo entre los días 13 de abril y 24 del mismo mes. A continuación, se va a exponer todo lo que se realizó durante este periodo de tiempo.

#### Valoración de tareas

En primer lugar, y antes de comenzar la implementación, se realizó una valoración inicial de las tareas que se iban a realizar a lo largo de este sprint. Al contar ya con una planificación detallada que indica el número aproximado de horas que, a priori, se le iban a dedicar a cada tarea, no fue necesario pararse demasiado en esta fase.

### Sprint planning

Durante esta ceremonia, se escogieron las tareas que se iban a realizar a lo largo de este sprint, al tratarse del primero, se decidió poner el foco sobre el usuario, y se añadieron las tareas de inicio de sesión y de registro como dos de las primeras tareas a desarrollar en el proyecto. Además, también contábamos con una tarea que nos iba a permitir implementar todo el esqueleto de la API, cosa que nos ayudaría a la hora de realizar las propias tareas. Finalmente, se decidió añadir la tarea que trataba de implementar la pantalla principal de la aplicación.

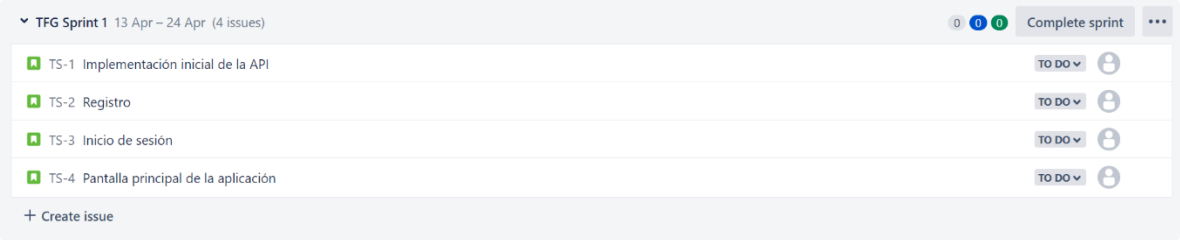



*Ilustración 62: Planificación del primer sprint. Fuente: Elaboración propia.*

### Fase de desarrollo

En primer lugar, se comenzó con la implementación inicial de la API. Durante el desarrollo de esta tarea se creó todo el esqueleto de carpetas que contendrían las futuras clases Java. Además, se crearon las clases necesarias para que el proyecto pudiera arrancar, se estableció la conexión con la base de datos y un primer *end-point* de prueba.

A continuación, se comenzó con el registro. Primeramente, se implementó toda la clase User en la API, la cual sería básicamente una referencia directa a la tabla usuario de base de datos. Además, se crearon tanto el UserService como el RegisterController, los cuales recibían un objeto RestInput y acababan almacenando un objeto usuario en base de datos. Tras finalizar toda la parte necesaria relacionada con el usuario en el back-end, se procedió a comenzar la implementación en Android. Siguiendo los *mockups* realizados para la parte del registro, se implementó primeramente toda la parte visual del registro, y, finalmente, toda la parte que acababa recogiendo los datos y enviándolos a la API para que esta creara el usuario.

Una vez terminada la fase de registro, ya se podía implementar el inicio de sesión, y, para ello, se siguieron los mismos pasos que con el registro. En primer lugar, se realizó toda la implementación necesaria en la API para el trato de datos recibidos por parte de la aplicación, después se implementó la UI en Android, y, finalmente, la recogida de datos por parte de la aplicación y su correspondiente envío.

La última tarea del sprint era la de realizar la pantalla principal de la aplicación, además de su implementación visual, también se tenía que diseñar todo el grafo de navegación que nos permitiría en un futuro visitar todas las partes de las que consta la aplicación.

#### Sprint retrospective

Al tratarse de la primera iteración del proyecto, todo se ha comenzado a hacer en este *sprint.* Gracias al buen trabajo realizado durante el GEP, se consiguió repartir la faena de forma equitativa durante los respectivos periodos de trabajo. En este *sprint* se ha podido implementar todo sin apenas problemas, a pesar de eso, se han dejado como "TODO" algunos aspectos, como por ejemplo el tratamiento de errores en la API. Esto es algo que se pretende mejorar de cara al próximo *sprint* y procurar eliminar todos los "TODO" del código.

El ritmo de trabajo ha sido muy bueno, y se ha visto reflejado en la finalización de todas las tareas planificadas, es por eso por lo que se pretende continuar con el mismo ritmo y trabajando de la misma forma.

Además, de cara al siguiente *sprint* se pretende comenzar a trabajar con Postman para las pruebas del código implementado en la API. Este programa nos permitirá saber si el código de la API es bueno sin necesidad de probarlo directamente con la aplicación, cosa que nos ahorrará mucho tiempo.

#### Sprint review

Durante el *sprint review*, se mostró el progreso actual de la aplicación y la API al director del proyecto, se comentó que no se había tenido prácticamente ningún problema y, además, que seguramente en el próximo sprint se avanzaría más de lo planificado debido a la soltura con la que se ha visto el autor desarrollando en este sprint. El director comentó que sería conveniente dejar de utilizar la base de datos a nivel local, pues es algo que puede dar problemas y dejarlo para última hora como se tenía previsto es una mala idea.

### 10.1.2 Sprint 2

#### Valoración de tareas

La estimación de las tareas que se realizó durante el módulo de GEP fue previa al comienzo de la implementación, es por eso por lo que se decidió realizar un proceso de revaloración al inicio de cada *sprint.* Al contar ya con la experiencia adquirida durante la primera iteración, se puede refinar la duración estimada de esas tareas, y esto es lo que se realizó al comienzo del segundo *sprint*.

En cuanto al sistema de notificaciones, se decidió dejar la estimación original, pues es algo completamente aislado y sobre lo que no se tenía ningún tipo de experiencia. Sin embargo, la tarea correspondiente a los ajustes y al perfil de usuario fue refinada asignándole un total de 15 horas (5 menos que en la planificación original). Este descenso fue debido a la soltura que ya se tenía en la creación de Actividades y en relacionarlas con sus respectivos ViewModel. Finalmente, la tarea de creación de las tarjetas para el visionado de los datos también sufrió un descenso, en este caso, de dos horas.

### Sprint planning

Cuando se finalizó la iteración anterior, ya se comentó con el director del proyecto que seguramente a lo largo de esta segunda se avanzaría más de lo previsto, esto fue debido a las buenas sensaciones que tuvo el desarrollador a lo largo del primer *sprint*. A pesar de esto, no se decidió añadir ninguna tarea a priori que se saliera de la planificación original, en la ilustración 62 se puede dicha planificación.

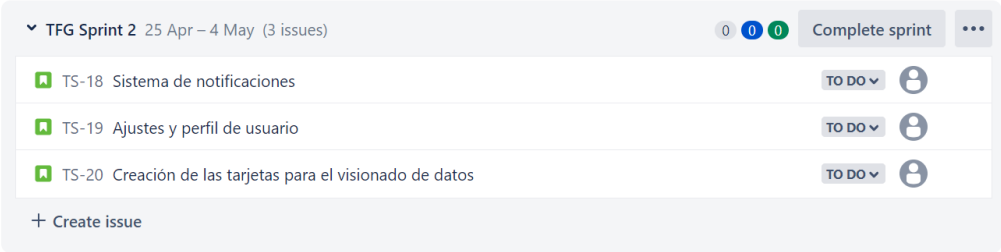



*Ilustración 63: Planificación del segundo sprint. Fuente: Elaboración propia.*

### Fase de desarrollo

Lo primero que se realizó tras comenzar la fase de desarrollo fue la parte de ajustes y perfil de usuario, se realizó en primer lugar toda la implementación necesaria en la API, esta implementación constaba de lo siguiente:

- End-point para el cambio de contraseña y su lógica correspondiente.
- End-point para el cambio de idioma y su lógica correspondiente.
- Obtención de los datos del usuario.

Una vez se finalizó la parte de la API se pasó a la aplicación, se diseñó toda la parte visual y se procedió a conectar todo de forma correcta para poder realizar el intercambio de información con la API.

En segundo lugar, se implementó la tarea de creación de las tarjetas para el visionado de datos, en particular se crearon tres distintas, una para los análisis, otra para la medicación y una última para las citas del calendario. Se dejó todo preparado para que cuando se realizaran las tareas correspondientes al muestreo de dicha información, no se tuviera que realizar casi nada. En tercer lugar, se implementó todo el sistema de notificaciones, que consiste en un servicio nuevo en la API (el cual es accesible desde todos los demás servicios), y un servicio en la aplicación, el cual recoge los mensajes enviados desde la API y lanza notificaciones en los dispositivos de los usuarios. Todo este proceso se realizó con Firebase Cloud Messaging.

Adicionalmente a las tareas propuestas en el *sprint*, también se siguió el consejo del director de dejar de alojar la base de datos localmente y se dedicó un tiempo a subir tanto la base de datos como la API a AWS.

Finalmente, y con el tiempo que nos sobraba hasta la finalización del *sprint*, se empezó el desarrollo de la visualización de las medicaciones, el cual se complicó bastante, pues se decidió hacer uso de una librería para poder mostrar las medicaciones de forma “sencilla” pero la implementación de esta resultó ser bastante complicada. Sin embargo, tras dedicarle aproximadamente 10 horas más de las planificadas originalmente se consiguió finalizar dicha tarea.

### Sprint retrospective

En este *sprint* se han comenzado a hacer cosas que se propusieron en la anterior reunión retrospectiva, como el dejar de hacer "TODOs", o hacer uso de Postman para probar las funcionalidades de la API sin necesidad de utilizar la aplicación. Todo esto ha supuesto un gran avance en la implementación, sin embargo, el hecho de tener que pararse a completar algunos "TODO" del *sprint* pasado ha supuesto una pérdida de tiempo considerable, se tendría que haber planificado algo mejor y haber dejado un par de días previos a la finalización del *sprint* para tratar de completar estos detalles.

### Sprint review

Este *sprint* ha sido muy provechoso, se han implementado varias tareas realmente importantes para el devenir del proyecto. Además, se ha conseguido finalizar a tiempo y con la inclusión de una tarea extra, el visionado de la medicación, el cual como ya se comentó, duró más de la cuenta, pero nos ha permitido aprender a utilizar la librería para el muestreo de datos, lo cual nos facilitará enormemente las cosas de cara a las siguientes tareas que lo requieran.

## 10.1.3 Sprint 3

### Valoración de tareas

Al igual que en la iteración anterior, se volvió a repasar las tareas que quedaban en el *backlog* reasignándoles el tiempo estimado. Gracias a todo lo aprendido con el visionado de la medicación, se consiguió reducir el tiempo a dedicar a las tareas de visionado de análisis y de citas médicas en un total de 5 horas cada una. Esto es debido a que con el uso de la librería que se comentaba en el *sprint* anterior, el muestreo de la información se realiza con un proceso rápido y sencillo.

En cuanto al resto de las tareas, se decidió dejarlas con el tiempo que se había previsto originalmente, pues no se contaba con la información suficiente como para poder refinarlas.

### Sprint planning

La planificación original de este *sprint* cambió debido a la realización de la tarea del visionado de la medicación a lo largo de la iteración anterior, fue por eso que se decidió incluir, en sustitución, el visionado de las analíticas. En la ilustración 63 se puede ver la planificación que se siguió a lo largo de esta tercera iteración.

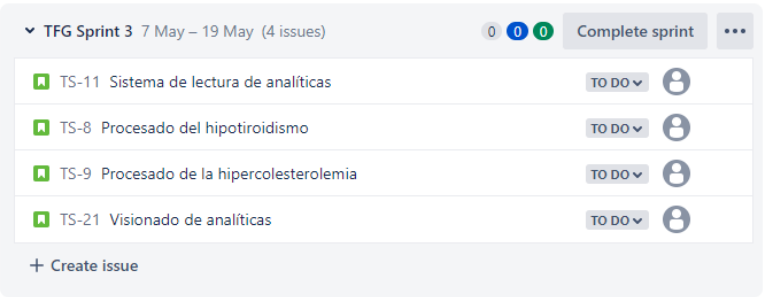



*Ilustración 64: Planificación del tercer sprint. Fuente: Elaboración propia.*

### Fase de desarrollo

Lo primero que se realizó en esta iteración fue la lectura de analíticas, pues para poder procesarlas primero se tienen que poder leer y convertir a un formato con el que la API pueda trabajar. Esta tarea fue completada con éxito, y se logró implementar un sistema de lectura altamente escalable.

En este *sprint* tocaba implementar el componente central del proyecto, el procesado de las enfermedades, este es un proceso que no solo requiere de implementar el procesado, sino que también de una cierta investigación médica para poder llegar a realizar una implementación lo más verídica posible. Para realizar esto, y como ya se comentó en el apartado 9.6, se contó con la ayuda de dos doctoras, y gracias al trabajo realizado analizado la información enviada por su parte antes del comienzo de este *sprint,* se pudo proceder a la implementación de forma inmediata una vez comenzó la iteración. Si bien el número de horas previsto inicialmente para esta tarea no se dedicó enteramente en esta iteración, si que se le dedicó de forma previa. Todo el proceso de implementación de esta tarea está detallado en los apartados 9.6 y 10.4.

Finalmente, contábamos con varios días extra, por ello se decidió realizar una fase de pruebas del procesado de las analíticas mucho más extensa de la inicialmente prevista. Además, se decidió incluir toda la parte de visionado de las citas en este *sprint* y se logró completar con éxito.

### Sprint retrospective

La realización del trabajo de investigación previo al comienzo del *sprint* ha resultado ser de lo más fructífero, no solo se ha conseguido implementar todo el procesado de analíticas a tiempo, sino que, además, se ha aprovechado el tiempo extra en poder probar esta funcionalidad a fondo.

### Sprint review

El buen trabajo recopilando información para el procesado de las analíticas hizo que al comenzar este *sprint* ya se tuviera todo lo necesario para comenzar la implementación, cosa que alivió el peso de la iteración. finalizar esta iteración, se mostraron los avances al director del proyecto para que pudiera verificar como iba avanzando.

### 10.1.4 Sprint 4

#### Valoración de tareas

En esta iteración no se realizó ningún refinamiento de las tareas propuestas, pues se consideró que las restantes en el backlog ya estaban bien valoradas.

#### Sprint planning

La planificación de este *sprint* no fue complicada, pues gracias al buen trabajo realizado durante los anteriores tres, se había llegado a un punto en el que solo restaban dos tareas para finalizar el proyecto. Y estas fueron las que se incluyeron en esta última iteración, se pueden ver en la ilustración 64.

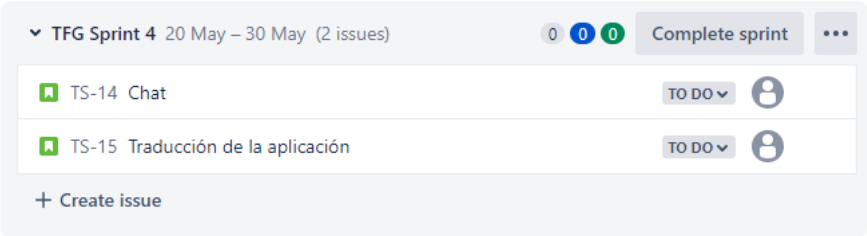



*Ilustración 65: Planificación del cuarto sprint. Fuente: Elaboración propia.*

#### Fase de desarrollo

La fase de desarrollo de este *sprint* no resultó nada especial, pues se trataba del último *sprint* y de ir cerrando las tareas que restaban. Se comenzó implementado toda la lógica necesaria para el chat en la API, y tras finalizar esta parte se procedió a implementar la parte de la aplicación. Si bien esta tarea resultó ser bastante complicada y fue fuente de bastantes errores (comentados más adelante), ya se planificó como tal, y se consiguió terminar con éxito.

Finalmente, se comenzó con la traducción de la aplicación, esto no es un proceso complicado, y, por lo tanto, se consiguió implementar de nuevo en el tiempo estimado.

#### Sprint retrospective

Al finalizar este *sprint* se hizo una retrospectiva general de todo el proyecto, los resultados de la cual se pueden ver en la conclusión de esta memoria.

#### Sprint review

Una vez se finalizó la última iteración, se hizo un repaso de todo lo implementado a lo largo de los cuatro *sprints*, y se generó una apk final la cual fue enviada al director del proyecto para que la pudiera probar y validar.

### 10.1.5 Bugs y errores

A lo largo de la implementación han ido surgiendo bugs y errores que han ido retrasando en mayor o menor medida la planificación inicial. Sin embargo, al tratarse de un proyecto con un solo desarrollador, cuando surgían estos problemas simplemente se le dedicaban más horas para tratar de solucionarlo en el momento que se detectaba. A continuación, se puede ver una lista de los bugs y errores más relevantes que han surgido durante el desarrollo.

- Cierre brusco de la aplicación al introducir datos de inicio de sesión incorrectos.
- La aplicación no recibe respuesta tras modificar la contraseña.
- No se reciben notificaciones en la aplicación.
- El chat no se refresca tras recibir un mensaje nuevo.
- El texto de las analíticas no se visualiza correctamente.
- Distintos errores en los textos que se envían tras compartir un análisis, medicación o cita.
- Tras pulsar sobre un botón de los cuatro de la pantalla principal, no nos lleva a la pantalla que toca, siempre lleva a la primera.
- Tras cerrar sesión, si se pulsa para volver hacia atrás en los menús de Android se regresa a la pantalla principal.
- Al pulsar sobre consultar analítica, siempre se muestra como valor resultante de la analítica el total.

## 10.2 Implementación del front-end

Como ya se ha comentado, el front-end del sistema consiste en una aplicación Android desarrollada en Kotlin, en esta sección se detallarán todos los aspectos que han resultado ser fundamentales a la hora de desarrollar esta aplicación.

### 10.2.1 Tecnologías empleadas

Para el desarrollo de la aplicación móvil se ha hecho uso del lenguaje Kotlin [35], actualmente, y según datos de Google, un 60% de los desarrolladores de Android utilizan este lenguaje para la implementación de sus aplicaciones. Además, se ha dado un gran uso a la cantidad de librerías y *frameworks* que existen actualmente para Android que nos ayudan en un gran número de ocasiones. Los principales han sido las siguientes:

- Dagger [36]. Este *framework* es el encargado de la inyección de dependencias. Gracias a Dagger podemos obtener de forma directa distintos elementos en nuestras clases Kotlin.
- Lifecycle [37]. Esta librería es la que nos permite contar con los ViewModel, como ya hemos visto estos ejercen un papel fundamental en la comunicación entre los Fragments o las Activities y los repositorios (encargados de acceder a datos).
- Corrutines [38]. Las corrutinas son empleadas para ejecutar tareas de forma asíncrona, en el caso que nos incumbe, se utilizan cada vez que se accede a datos para no parar la ejecución de la aplicación en ningún momento.
- Retrofit [39]. Construye todas las llamadas a la API a partir de una breve especificación implementada en una interfaz (ApiService).

- GSON [40]. Esta librería se utiliza principalmente para convertir objetos en su representación JSON y viceversa. Cuando enviamos un objeto a la API, este tiene que estar en formato JSON, gracias a esta librería el proceso es automático, al igual que cuando lo recibimos en JSON, es esta librería quien se encarga de convertirlo a un objeto de modelo.
- MaterialDrawer [41]. Esta librería nos permite incorporar de forma sencilla el menú lateral que podemos desplegar en los fragmentos. Este elemento nos permite navegar entre las distintas secciones de la aplicación sin la necesidad de volver a la pantalla principal.
- Material Design Components [42]. Material Design ha sido una de las librerías más utilizadas a lo largo del desarrollo de la aplicación, gracias a ella podemos acceder a una gran cantidad de componentes ya diseñados los cuales se pueden añadir directamente a las interfaces de la aplicación.
- FastAdapter [43]. Esta librería se utiliza principalmente para la creación de listas de elementos. Esto se hace mediante RecyclerViews, que son un tipo de vista que contiene un número indeterminado de elementos, en nuestro caso, tanto análisis, citas, chats y medicaciones utilizan RecyclerViews para ser mostrados.
- Material Dialogs [44]. Esta librería nos permite instanciar diálogos de forma rápida y sencilla, siguiendo el diseño de Material Design. Con la ayuda de esta librería se instancian los diálogos para ver analíticas, o reclamar citas entre otros.
- Firebase [45]. El módulo de Firebase que ha sido de especial interés en este proyecto ha sido Firebase Cloud Messaging, gracias al cual se reciben las notificaciones enviadas desde la API y se muestran al usuario.
- NavController [46]. La navegación interna de la aplicación está desarrollada utilizando esta librería, la cual nos permite crear un grafo de navegación de forma muy sencilla.

En adición a Kotlin, se ha hecho uso del lenguaje XML para el desarrollo de la parte visual de la aplicación, todas las interfaces han sido desarrolladas en este lenguaje.

### 10.2.2 Código relevante en el transcurso del proyecto

En la sección de patrones utilizados se mostraron distintos fragmentos de código que ilustraban el uso de dichos patrones dentro de la aplicación y la API, se trató de escoger fragmentos que realmente fueran relevantes para entender el uso de los patrones que se estaban mostrando. En este apartado se van a mostrar ejemplos de código que han sido cruciales en el devenir del sistema adicionalmente a los ya mostrados.

#### Inyección de dependencias

Para que Dagger pueda inyectar las distintas dependencias que necesitamos, se tienen que crear unas ciertas clases llamadas Modules, las cuales le permiten identificar tanto los objetivos de la inyección, como el material a inyectar. A continuación, podemos ver un ejemplo donde se incluyen todas las actividades que contiene la aplicación.

```
@Module
abstract class ActivitiesModule {
    @ContributesAndroidInjector
    abstract fun contributeMainActivity() : MainActivity

    @ContributesAndroidInjector
    abstract fun contributeIntroActivity() : IntroActivity

    @ContributesAndroidInjector
    abstract fun contributeRegisterActivity() : RegisterActivity

    @ContributesAndroidInjector
    abstract fun contributeLoginActivity() : LoginActivity

    @ContributesAndroidInjector
    abstract fun contributeSettingsActivity() : SettingsActivity

    @ContributesAndroidInjector
    abstract fun contributeChatItemActivity() : ChatItemActivity

    @ContributesAndroidInjector(modules = [FragmentsModule::class])
    abstract fun contributeDrawerActivity() : DrawerActivity
}
```

*Ilustración 66: Inyección de dependencias en las actividades. Fuente: elaboración propia.*

Además de indicar a Dagger todas las actividades sobre las cuales tiene que inyectar dependencias, también se indica que tiene que inyectarlas sobre FragmentsModule, el cual contiene todos los fragmentos de la aplicación.

Este no es la única clase necesaria para hacer funcionar Dagger, entre otras también existe ViewModelsModule, la cual inyecta las dependencias en los distintos ViewModels que contiene la aplicación que son accesibles desde DrawerActivity.

Dagger ha resultado ser extremadamente útil a lo largo del desarrollo, pues nos ha permitido simplificar mucho la instanciación de las clases. Si bien su implementación es bastante compleja, una vez se tiene todo el entorno necesario establecido simplemente es ir añadiendo o quitando fragmentos de código de los que se encuentran en los diferentes módulos correspondientes.

### Obtención de datos desde la aplicación

Para obtener los datos, la aplicación accede a la API, para poder realizar esto contamos con un módulo central llamado APIService, el cual es el encargado de recoger todas las posibles llamadas a esta, ejecutarlas y recoger su respuesta.

```
interface ApiService {
    @POST("register")
    suspend fun newUser(@Body user: UserBuilder): TokenResponse

    @POST("login")
    suspend fun getUser(@Body user: LoginUser): TokenResponse

    @GET("user")
    suspend fun getUserProfile(@Header("Authorization") token: String): User

    @PUT("user")
    suspend fun changePassword(
        @Header("Authorization") token: String,
        @Body password: Password
    ): Response<Unit>

    @GET("analysis")
    suspend fun getAnalysis(@Header("Authorization") token: String): List<Analysis>

    @PUT("user/fcmtoken")
    suspend fun insertFCMToken(
        @Header("Authorization") token: String,
        @Body fcmToken: FCMToken
    ): Response<Unit>

    @GET("calendar")
    suspend fun getAppointments(@Header("Authorization") token: String): List<Appointment>

    @GET("chat")
    suspend fun getChats(@Header("Authorization") token: String): List<Chat>

    @POST("chat")
    suspend fun createChat(
        @Header("Authorization") token: String,
        @Body chatBuilder: ChatBuilder
    ): Response<Chat>

    @GET("chat/{id}")
    suspend fun getChat(
        @Header("Authorization") token: String,
        @Path("id") id: Long
    ): Response<ChatItem>

    @POST("chat/message")
    suspend fun createMessage(
        @Header("Authorization") token: String,
        @Body message: SentMessage
    ): Response<Unit>

    @GET("medication")
    suspend fun getMedication(@Header("Authorization") token: String): Response<List<Medication>>

    @PUT("user/language")
    suspend fun changeLanguage(
        @Header("Authorization") token: String,
        @Body language: Language
    ): Response<Unit>

}
```

*Ilustración 67: ApiService. Fuente: Elaboración propia.*

Como podemos ver en la imagen 66, se pueden ver todas las llamadas a la API necesarias para el buen funcionamiento del sistema, estos métodos definidos aquí son llamados por los distintos repositorios.

### Recogida de notificaciones

Para que el usuario pueda ver las notificaciones, estas son recogidas por un servicio encargado exclusivamente de realizar esta tarea. Cuando llega se ejecuta el método “sendNotification” el cual es el encargado de hacer aparecer la notificación en el dispositivo del usuario.

```
private fun sendNotification(messageBody: String?, title: String?, category: String?) {
    val type = NotificationType.fromString(category!!)
    val intent = Intent(this, DrawerActivity::class.java)
    intent.putExtra("fragmentId", type.id)
    intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP)
    val pendingIntent = PendingIntent.getActivity(
        this, 0 /* Request code */, intent,
        PendingIntent.FLAG_ONE_SHOT
    )

    val channelId = "AAAChannel"
    val defaultSoundUri = RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION)
    val notificationBuilder = NotificationCompat.Builder(this, channelId)
        .setSmallIcon(R.drawable.ic_logo)
        .setContentTitle(title)
        .setContentText(messageBody)
        .setAutoCancel(true)
        .setSound(defaultSoundUri)
        .setContentIntent(pendingIntent)

    val notificationManager =
        getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager

    // Since android Oreo notification channel is needed.
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        val channel =
            NotificationChannel(
                channelId,
                "AAAChannel",
                NotificationManager.IMPORTANCE_DEFAULT
            )

        notificationManager.createNotificationChannel(channel)
    }

    notificationManager.notify(0 /* ID of notification */, notificationBuilder.build())
}
```

*Ilustración 68: Método sendNotification. Fuente: Elaboración propia.*

Este método, también es el encargado de recoger el tipo de notificación indicado originalmente en la API, gracias a esto, cuando el usuario pulse sobre la notificación, se le llevará directamente a la sección de la aplicación correspondiente a dicha notificación.

ViewModel y Corrutinas

Los ViewModel juegan un papel crucial en el intercambio de información entre la vista y los datos, es por eso por lo que se ha decidido incluir un ejemplo de uno de los que existen dentro de la aplicación, el correspondiente a los análisis.

```
class AnalysisViewModel @Inject constructor(private var repository: AnalysisRepository) :
    ViewModel() {
    val analysis: LiveData<List<Analysis>>
        get() = _analysis

    private val _analysis = MutableLiveData<List<Analysis>>()

    fun refreshAnalysis() {
        viewModelScope.launch {
            val data = repository.getAnalysis()
            _analysis.postValue(data)
        }
    }
}
```

*Ilustración 69: Ejemplo de ViewModel. Fuente: Elaboración propia.*

Como se puede ver en la imagen, contamos con el método refreshAnalysis() el cual se encarga de acceder a los datos y asignar la información recogida de la API a un LiveData que es escuchado desde el fragmento. Además, dentro del mismo método podemos ver un uso de las corrutinas dentro de la aplicación, todo el acceso a los datos está envuelto por una.

## 10.3 Implementación del back-end

El back-end está constituido por una API desarrollada en Spring (Java), esta API es la encargada de realizar toda la lógica necesaria para el buen funcionamiento del sistema y de acceder a base de datos. Al igual que en la sección del front-end, en esta se van a detallar todas las partes relevantes de la implementación de la API.

### 10.3.1 Tecnologías empleadas

El *framework* escogido para el desarrollo de la API ha sido Spring [47], el lenguaje empleado es Java y es actualmente uno de los más usados hoy en día. De igual manera que en la aplicación, se han hecho uso de librerías que han ayudado enormemente en el desarrollo. Estas son las siguientes:

- Spring [48]. Spring Boot nos permite crear APIs REST de forma muy sencilla, tiene un gran número de funcionalidades que hacen el desarrollo mucho más sencillo. Entre ellas podemos destacar la fácil integración con base de datos y la ejecución casi automática que logramos al ejecutar una aplicación Spring.
- Lombok [49]. Esta librería nos permite eliminar por completo tanto los *setters* como los *getters* de nuestras clases inyectándolos de forma automática. Marcando dicha clase con @Data hace que todos los campos privados que se encuentran en su interior puedan ser accedidos mediante unos *getters* y *setters* proporcionados por Lombok, lo cual hace que el código sea mucho más limpio y fácil de escribir. Además, gracias a Lombok, también se nos permite eliminar los constructores, estos se generan de forma automática.

- **Jackson** [50]. Jackson es el equivalente a GSON en la API, esta librería nos ayuda a convertir los JSON recibidos en objetos y los objetos enviados los transforma a JSON de forma automática.
- **Log4j** [51]. Esta librería se utiliza exclusivamente para la realización de logs dentro de la API.
- **JWTs** [52]. Para aumentar la seguridad de la API, se utilizan tokens para el intercambio de información entre la aplicación y la API, esta librería nos facilita la creación y comprobación de dichos tokens.
- **Firebase** [45]. Es la API quien haciendo uso de Firebase Cloud Messaging envía las notificaciones que la aplicación recibe y muestra al usuario, esto se consigue gracias al uso de la API de Firebase.
- **JPA** [53]. Este *framework* nos permite realizar transacciones con la base de datos haciendo uso de los objetos de modelo definidos. Mediante una serie de anotaciones, se indica que un objeto es a su vez una tabla en base de datos, y haciendo uso de los repositorios proporcionados por JPA podemos acceder a base de datos de forma muy sencilla, evitándonos tener que realizar las sentencias SQL pertinentes.

### 10.3.2 Código relevante en el transcurso del proyecto

Las funcionalidades de la API ya han sido detalladas mediante diagramas de secuencia, en esta sección se pretende mostrar distintas secciones del código que han sido importantes y que no se han mostrado a lo largo de la memoria.

#### Inyección del usuario en los controladores

Cuando se realiza una llamada a la API y esta contiene en la cabecera un token de usuario, al recibir dicha llamada se realiza un preproceso para tratar de obtener el usuario a partir del token y poder inyectarlo en el controlador que va a recibir la llamada, esto se hace mediante el siguiente código.

```java
@ControllerAdvice(annotations = AuthAwareRestController.class)
@RequiredArgsConstructor
public class AuthHandler
{
    final UserService userService;

    @ModelAttribute
    public void validateToken(@RequestHeader(HttpHeaders.AUTHORIZATION) String token, final Model model)
    {
        try {
            userService.validateToken(token).ifPresent(model::addAttribute);
        } catch (Exception e){
            throw new ResponseStatusException(HttpStatus.UNAUTHORIZED);
        }
    }
}
```

*Ilustración 70: Inyección del usuario en los controladores. Fuente: Elaboración propia.*

Además del código referente a la inyección del usuario, también se puede ver en la ilustración 69 un claro ejemplo de la utilidad de @RequiredArgsConstructor, se puede apreciar cómo se hace uso de UserService sin necesidad de inicializarlo a través de un constructor.

### Notificaciones

El envío de información a través de notificaciones se inicia desde la API, en particular, a través del método siguiente:

```java
@Service
@Slf4j
@RequiredArgsConstructor
public class MessagingService {

    @Async
    public void notifyUser(final User user, String title, String body, NotificationType category) {
        if (user.getFirebaseToken() != null) {
            try {
                Message message = Message.builder().setToken(user.getFirebaseToken())
                        .putData("title", title)
                        .putData("body", body)
                        .putData("category", category.name())
                        .build();
                FirebaseMessaging.getInstance().send(message);
            } catch (FirebaseMessagingException e) {
                e.printStackTrace();
            }
        }
    }
}
```

*Ilustración 71: Envío de notificaciones. Fuente: Elaboración propia.*

Este método es accesible desde cualquier Servicio que cuente con MessagingService, es una forma unificada de mandar notificaciones.

Adicionalmente al método mostrado en la ilustración 70, la clase MessagingService contiene un método específico para las notificaciones de tipo chat, gracias a este método podemos construir notificaciones que distan un poco de las clásicas enviadas por el método notifyUser. Dicho método se puede ver en la ilustración siguiente.

```java
@Async
public void notifyNewChatMessage(final User user, com.aaa.automaticanalyzer.model.chat.Message message, Long
chatId) {
    if (user.getFirebaseToken() != null) {
        try {
            Message notification = Message.builder().setToken(user.getFirebaseToken())
                    .putData("content", message.getContent())
                    .putData("user", user.getName())
                    .putData("chatId", String.valueOf(chatId))
                    .putData("category", NotificationType.CHAT.name())
                    .build();
            FirebaseMessaging.getInstance().send(notification);
        } catch (FirebaseMessagingException e) {
            e.printStackTrace();
        }
    }
}
```

*Ilustración 72: Envío de notificaciones de tipo chat. Fuente: Elaboración propia.*

## 10.4 Añadir nueva enfermedad al sistema

La cantidad de enfermedades crónicas que existen hoy en día es algo muy a tener en cuenta a la hora de desarrollar un sistema software como el que se propone en esta memoria. Es por este motivo, que la escalabilidad del sistema ha sido uno de los puntos clave de la implementación.

Actualmente se pueden procesar dos enfermedades crónicas, el hipotiroidismo y la hipercolesterolemia, sin embargo, se ha diseñado todo un mecanismo para que, en caso de querer añadir el procesado de nuevas enfermedades, el procedimiento que se tenga que llevar a cabo sea sencillo y no afecte para nada al resto del sistema. Una de las claves para conseguir esto ha sido el aislamiento total de la aplicación, y es que esta simplemente es un "vehículo" de muestreo de los datos enviados desde la API, permitiendo de esta manera cambiar de front-end sin que el procesado de analíticas sufra ningún tipo de problema.

Otra de las claves ha sido la construcción de un mecanismo que, gracias al patrón estrategia, nos permite realizar el procesado de una enfermedad u otra utilizando el mismo código, consiguiendo de esta forma no tener que modificar el código ya escrito para tener que añadir una enfermedad nueva. En caso de querer añadir una nueva enfermedad al sistema, se tendrían que seguir los pasos siguientes:

1. Crear una clase nueva correspondiente al tipo de análisis nuevo que se pretende procesar, con los parámetros que toquen. Esta clase tendrá que extender de BaseAnalysis que ya contiene todos los métodos necesarios para el envío de datos al front-end.
2. Crear un nuevo motor de procesador correspondiente a la nueva enfermedad que se va a procesar, este motor tendrá que implementar la interfaz ProcessingEngine.
3. Añadir un nuevo valor al enum Disease correspondiente a la nueva enfermedad que se va a tratar.

Como podemos ver, solo con tres pasos el sistema ya puede pasar a procesar un nuevo tipo de enfermedad, esto es gracias a que el procesado de analíticas se hace a partir de la enfermedad. Es decir, el enum Disease contiene una referencia tanto a la analítica que corresponde a dicha enfermedad (con sus parámetros), como a su motor de procesado. Esto permite recoger la enfermedad directamente de la analítica, e invocar al motor de procesado que le corresponde para que este se encargue de analizar si los valores recibidos son correctos. Esta referencia se puede ver ejemplificada en la imagen siguiente:

```
HYPOTHYROIDISM(HypothyroidismAnalysis.class, new HypothyroidismEngine()),
HYPERCHOLESTEROLEMIA(HypercholesterolemiaAnalysis.class, new HyperCholersterolemiaEngine());
```

*Ilustración 73: Ejemplo de las referencias guardadas en el enum Disease. Fuente: Elaboración propia.*

Los análisis contienen una referencia a Disease (el enum), esto permite identificar la enfermedad que se está controlando con dicha analítica, y aprovechándonos de esto se ha implementado todo el sistema explicado para poder automatizar aún más todo este proceso. Obteniendo la enfermedad de la analítica y el motor de procesado ligado a dicha enfermedad, se pueden procesar los datos de una enfermedad completamente nueva sin necesidad de realizar modificaciones de código más allá de los necesarios para la adición de dicha enfermedad.

La inclusión de la nueva clase referente a las analíticas es algo que no se puede evitar, cada enfermedad tiene unos parámetros específicos que se tienen que controlar para verificar si se está llevando a cabo un buen control. A pesar de esto, crear esta nueva clase gracias a Lombok no es algo muy complicado, pues simplemente se tienen que añadir los parámetros correspondientes, e implementar el método formatContent, que nos permite aislar por completo la visualización de los datos en la aplicación.

```
    @Override
    public String formatContent(String content, Language language) {
        String[] contents = content.split(",");
        String tshValue = contents[1];
        tshValue = tshValue.split(":")[1];
        tshValue = tshValue.replace("\"", "");
        return Strings.TSH.getLanguage(language) + ": " + tshValue + " " + units +
                "  (" + minTSH + " - " + maxTSH + ")";
    }
}
```

*Ilustración 74: Ejemplo del método formatContent para el caso del hipotiroidismo. Fuente: Elaboración propia.*

El método de la ilustración 73 corresponde a la clase que hace referencia a los análisis de control del hipotiroidismo. Cuando la aplicación quiere recoger los datos de una analítica, estos son formateados y entregados directamente en el formato a mostrar dentro de la aplicación. En adición, y gracias al parámetro Language y del enum Strings (que contiene todas las traducciones del sistema) devolvemos el contenido en el lenguaje que el usuario tiene seleccionado por defecto.

Finalmente, se tiene que añadir un motor de procesado, esto también es evidente pues cada enfermedad se procesa de forma diferente, se trata con distintos medicamentos, y en general, suele ser completamente distinta de cualquier otra ya añadida. De todas formas, se ha creado una interfaz que tiene que ser implementada por todo motor de procesado que se quiera añadir, la cual sirve como plantilla, introduciendo los métodos que serán necesarios a la hora de realizar dicho procesado.

# 11. Base de datos del sistema

A lo largo de esta sección, se va a documentar todo el proceso que se ha seguido para la especificación, diseño e implementación de la base de datos que se utiliza en el sistema software que se presenta. La elección de las tecnologías de la base de datos y el análisis de las alternativas se puede ver en las secciones 4.2.5 y 4.3.4 de la sección Metodología.

## 11.1 Especificación y diseño

Para realizar el diseño de la base de datos, se tuvo muy en cuenta el diagrama de clases diseñado en la especificación de requisitos, pues la base de datos tiene que almacenar todos los datos necesarios para el buen funcionamiento del sistema, y estos estaban definidos allí. A continuación, podemos ver el diagrama final de la base de datos.

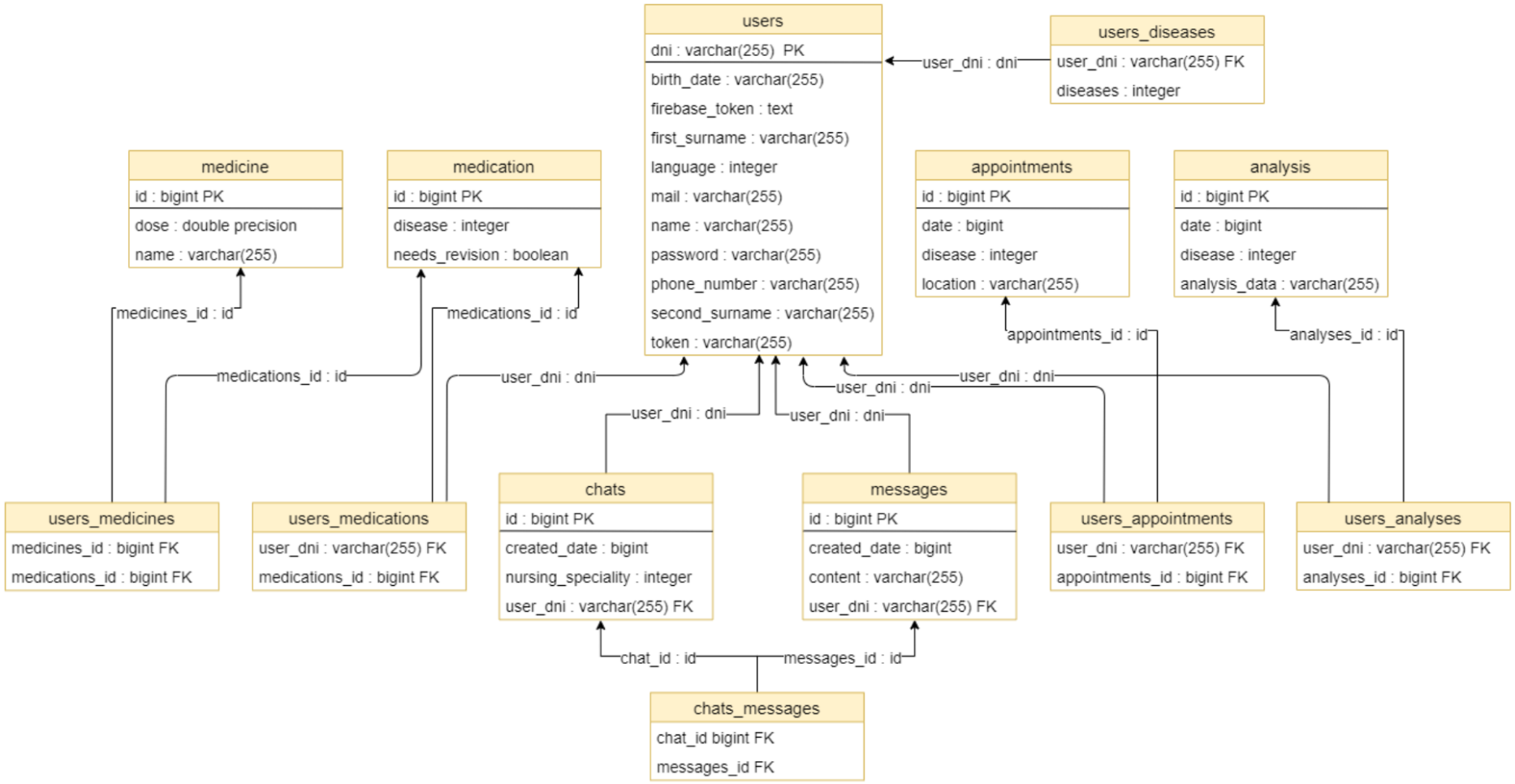



*Ilustración 75: Diagrama de la base de datos. Fuente: Elaboración propia.*

El diagrama anterior, nos deja con un total de 13 tablas, como se puede ver, las claves primarias de cada tabla se encuentran siempre por encima de una línea divisoria y marcadas con las letras "PK" (*Primary Key*). Sin embargo, existen algunas tablas que no cuentan con claves primarias, estas son tablas que se han creado para mantener relaciones entre las distintas tablas que hay. Este es el caso de las siguientes:

- User_diseases: La clase User contiene un listado de enfermedades (las que el padece), para poder almacenar este listado en base de datos se crea esta tabla que contiene una referencia al usuario, y otra a un enum que contiene todas las enfermedades tratadas en el sistema.
- User_analyses: Para tratar de simplificar la clase analítica, y la obtención de los análisis por parte de la aplicación, se guardó la referencia a las analíticas dentro de la clase User. De forma similar a como ocurre con las enfermedades, la clase User contiene un listado de analíticas, esta tabla contiene dos claves foráneas que hacen posible esta relación usuario-analítica.
- User_appointments: De forma equivalente a lo que ocurre con las analíticas ocurre con las citas del usuario, para hacer posible la relación usuario-cita se utiliza esta tabla, la cual contiene dos claves foráneas una para cada objeto del modelo.
- User_medications: De nuevo esta tabla sirve para mantener la relación entre el listado de medicaciones del usuario y el usuario per se.
- User_medicines: Una medicación puede estar compuesta por más de un medicamento, es por eso por lo que se guarda una relación entre la medicación y los medicamentos que la componen.

Respecto al modelo seguido en la base de datos, este ha sido el modelo ACID de transacciones [54], este consiste en lo siguiente:

- Atomicidad: Esta propiedad nos garantiza que una operación de acceso a base de datos no va a quedar a medias bajo ningún concepto.
- Consistencia: La consistencia se refiere a la capacidad de la base de datos de permitir únicamente las transacciones que siguen las reglas diseñadas en el diseño de esta. Es decir, evita que se realicen transacciones que se saltan las reglas.
- Aislamiento: El aislamiento se encarga de controlar el momento en el cual los cambios realizados con una transacción son visibles. Existen varios niveles, y un mayor nivel siempre implica la limitación de los accesos a datos de forma simultánea.
- Durabilidad: Esta propiedad indica la permanencia de los datos, es decir, que una vez se encuentran alojados en la base de datos, no se pierdan bajo ningún concepto.

Las transacciones son las operaciones que se realizan contra la base de datos, y, como ya se ha visto en la sección correspondiente del back-end, se realizan un gran número de operaciones distintas que acceden de una forma u otra a la base de datos.

## 11.2 Implementación

En el apartado 9.3.4 se vio como se conectaba la API con la base de datos, sin embargo, esta sección tiene la intención de mostrar cómo ha sido todo el proceso de implementación de esta.

Como se puede ver en la imagen 75, la clase Appointment del modelo de la API, sirve a su vez como representación de la tabla correspondiente en base de datos, esto es gracias a las anotaciones @Entity y @Table(name=”APPOINTMENTS”), las cuales nos permiten definir clases Java que se traducirán de forma automática en tablas dentro de la base de datos una vez se ejecute la aplicación Spring.

```java
@Entity
@Table(name = "APPOINTMENTS")
@Data
public class Appointment {
    @Column(unique = true)
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long id;

    private Disease disease;

    private Long date;

    private String location;
}
```

*Ilustración 76: Ejemplo de la representación en código de la tabla "Appointments". Fuente: Elaboración propia.*

Al igual que con las citas (*Appointments*), existe una clase con las mismas anotaciones (modificando el nombre dentro de los paréntesis de @Table) para cada tabla de la base de datos. Las claves foráneas (indicadas en el diagrama como FK) nacen de la existencia de referencias entre tablas dentro del modelo. Para indicar esto, se han hecho uso de las siguientes anotaciones*: ManyToOne, ManyToMany, OneToMany, OneToOne,* y de *ElementCollection* para el caso de la relación usuario-enfermedad. Estas etiquetas están directamente relacionadas con las relaciones que existen entre clases en el diagrama de clases. Gracias a estas anotaciones, la generación de las tablas es automática una vez se crean todos los objetos con las anotaciones correspondientes y se ejecuta la aplicación.

Para el acceso a base de datos, ya se ha comentado que se utiliza el patrón repositorio. La librería que se utiliza para realizar los accesos (JPA) nos permite acceder a la base de datos sin necesidad de escribir sentencias SQL. Las lecturas, las escrituras y las modificaciones, se realizan mediante la llamada a métodos ya definidos por JPA, lo único que tenemos que hacer es crear una clase que herede directamente de JPARepository y añadir allí los métodos que necesitamos para acceder a base de datos, pero no todos, pues los básicos ya están proporcionados directamente por JPA. A continuación, podemos ver un ejemplo.

```java
public interface ChatRepository extends JpaRepository<Chat, Long> {
    public List<Chat> getChatsByUserOrderByCreatedDateDesc(final User user);
}
```

*Ilustración 77: Ejemplo del acceso a datos. Fuente: Elaboración propia.*

Como se puede ver en la ilustración 76, el repositorio correspondiente al chat solo tiene un método descrito por nosotros, este método nos permite recuperar todos los chats del usuario ordenados de forma descendiente. Como podemos ver, no nos ha hecho falta ninguna sentencia SQL para realizar este acceso, JPA se encarga de la traducción automática. Además de este método contamos con los de la superclase (JpaRepository), por lo tanto, para guardar un chat en base de datos, simplemente tendríamos que llamar al repositorio y realizar una llamada al método *save* pasándole como parámetro el chat que queremos guardar, y para obtener algún elemento solo se tendrá que hacer un *findById(id)*.

# 12. Control de la calidad y pruebas del sistema software

## 12.1 Control de la calidad del software

La calidad del código es algo a tener en cuenta cuando se desarrolla un sistema software, esto es algo en lo que se ha hecho bastante inciso a lo largo de la especialidad de Software, remarcando sobre todo los conceptos de claridad, escalabilidad, reusabilidad y la facilidad que tiene que tener nuestro código para ser sometido a mantenimiento.

Al ser un proyecto realizado por una sola persona, no se pasa por procesos como las *Pull Requests*, este proceso implica una revisión del código generado (generalmente por otra persona). Cuando se realiza una *Pull Request*, una o más personas tienen que aprobar dicho código para que este pueda entrar en la rama central del producto. Este proceso hace que la calidad del código aumente considerablemente, pues no solo es revisado por una persona, sino que, además, son varias las personas que lo revisan y lo corrigen, poniendo comentarios allá donde crean que puede haber una mejora.

Por lo tanto, para asegurar la calidad del código generado a lo largo de la implementación, se ha decidido utilizar SonarQube [55], esta herramienta realiza un análisis exhaustivo del código y nos genera un informe que nos reporta posibles bugs, brechas de seguridad, y, en general, como es la calidad del código que estamos generando.

Como ya se ha comentado a lo largo de la memoria, el sistema propuesto está compuesto por una API, implementada en Java, y una aplicación móvil, implementada en Kotlin. Ambos lenguajes están soportados por SonarQube y, por lo tanto, se va a utilizar esta herramienta para validar el código tanto de la aplicación, como de la API.

En primer lugar, se realizará una primera ejecución de SonarQube para que nos detecte aquellos puntos del código que no acaban de ser buenos (fuentes de posibles bugs, código poco mantenible…). En particular, SonarQube nos va a ayudar con lo siguiente:

- Bugs: Nos va a detectar el número de posibles bugs del sistema.
- Vulnerabilidades: Nos va a proporcionar la cantidad de vulnerabilidades existentes en el sistema.
- Puntos críticos de seguridad: Devolverá el número de puntos críticos de seguridad existentes en el código.
- Puntos de mejora del código: Nos resaltará el número total de acciones que podemos realizar para mejorar el código.
- Cantidad de líneas duplicadas: Realizará un análisis exhaustivo por todo el código proporcionándonos el tanto por ciento de código duplicado.

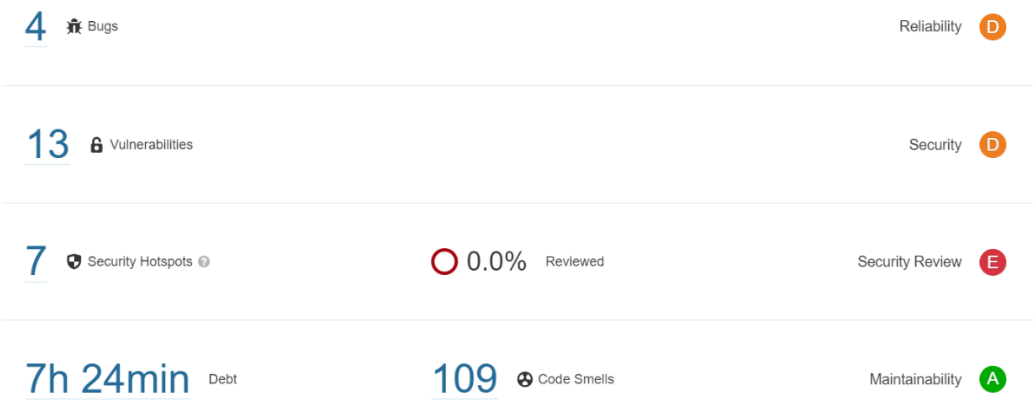



*Ilustración 78: Resultados SonarQube en el código de la API. Fuente: Elaboración propia.*

En la ilustración 77 podemos ver los resultados de la primera ejecución de SonarQube en la API. Como se puede ver, ha detectado un gran número de problemas que sin el uso de este programa hubiese sido muy complicado de detectar.

Tras realizar un análisis exhaustivo de todo lo expuesto por el programa, se han solucionado bastantes vulnerabilidades y dos posibles fuentes de errores. Además, se han implementado muchas de las sugerencias propuestas en el apartado de "*Code Smells*" y se ha conseguido obtener un código mucho más limpio.

Sin embargo, algunas de las propuestas que SonarQube nos ha realizado han sido marcados como falsos positivos, ya que, en el caso de nuestra aplicación, no suponían un verdadero problema. Finalmente, y tras la puesta en práctica de los consejos propuestos por SonarQube y de realizar el marcaje de los falsos positivos, obtuvimos los resultados que se muestra en la ilustración 78.

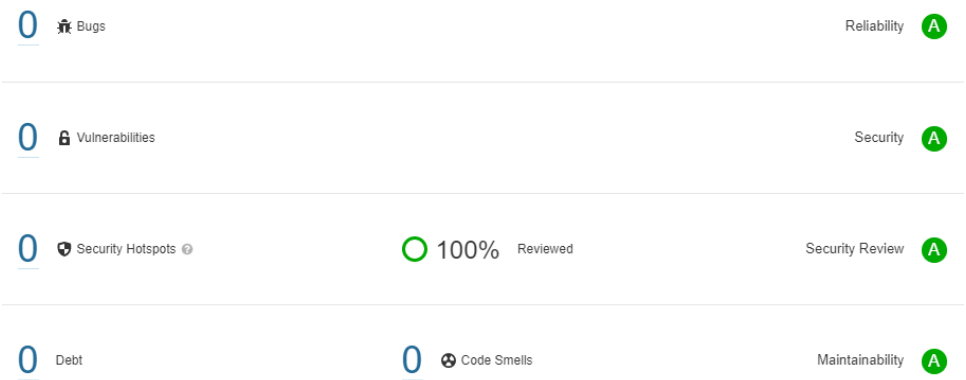



*Ilustración 79: Resultados de SonarQube tras implementar las sugerencias propuestas. Fuente: Elaboración propia.*

Para terminar con el análisis del código de la API, en la ilustración 79 podemos ver el tanto por ciento de código duplicado. El resultado ha sido de poco más del 1% en casi 2000 líneas, cosa que es prácticamente negligible, a pesar de ser un porcentaje muy bajo, se investigaron las fuentes de este código duplicado y se decidió dejarlo de esta forma, pues el código duplicado aportaba una mayor facilidad a la hora de leer el código que la extracción de dicho código en un método compartido.


*Ilustración 80: Tanto por ciento de código duplicado en la API. Fuente: Elaboración propia.*

Tras realizar las mejoras en el código de la API, se decidió realizar lo mismo para el código de la aplicación, se ejecutó SonarQube nuevamente y se obtuvieron los resultados que se pueden ver en la ilustración 80.

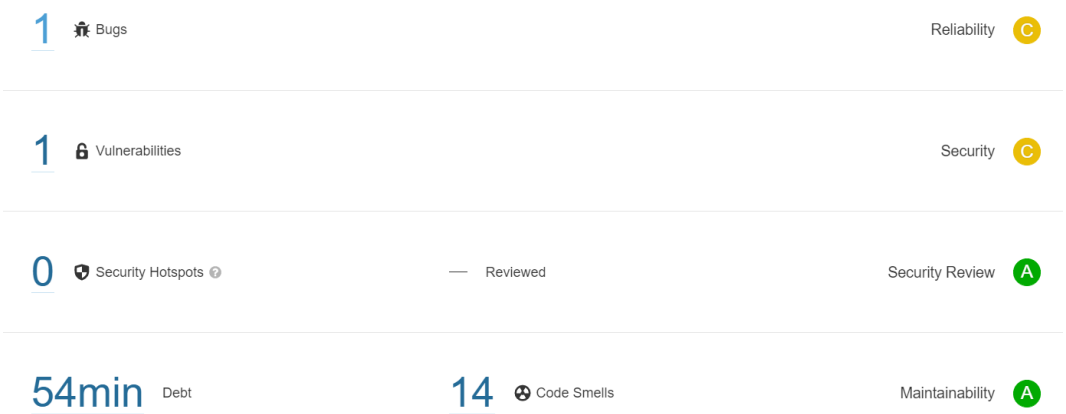



*Ilustración 81: Resultados SonarQube en el código de la aplicación. Fuente: Elaboración propia.*

La cantidad de "problemas" obtenidos en la aplicación fue mucho menor que en el caso de la API, en parte debido a que la programación en Kotlin requiere mucho menos código para realizar una acción que el que puede requerir Java.

Al igual que se hizo con la API, se analizaron las propuestas de SonarQube, algunas resultaron ser muy útiles para contribuir con la limpieza del código, y otras se tuvieron que marcar como falsos positivos ya que no eran relevantes en el devenir de este proyecto. A continuación, en la ilustración 81 podemos ver los resultados de la segunda ejecución de SonarQube tras implementar todo lo propuesto.

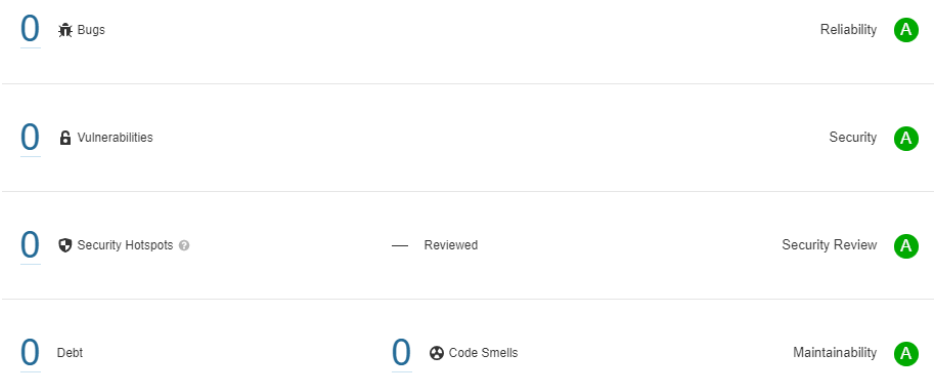



*Ilustración 82: Resultados de SonarQube tras implementar las sugerencias propuestas. Fuente: Elaboración propia.*

Para finalizar con el análisis del código de la aplicación, en la ilustración 82 podemos ver el tanto por ciento de código repetido. Como podemos observar, en este caso es del 0% en 4600 líneas, cosa que habla muy bien del código implementado.


*Ilustración 83: Tanto por ciento de código duplicado en la aplicación. Fuente: Elaboración propia.*

## 12.2 Plan de pruebas del sistema y verificación de requisitos funcionales

Al finalizar cada iteración, se elaboraron planes de pruebas que cubrían todas las funcionalidades implementadas a lo largo de ese periodo de tiempo. Cada plan consta de lo siguiente:

- Descripción de la funcionalidad.
- El input necesario por parte del usuario.
- Pasos a seguir para llevar a cabo la acción principal de la funcionalidad.
- Pasos a seguir para llevar a cabo las acciones secundarias de la funcionalidad.
- El resultado esperado de cada posible escenario.

La posible aparición de errores, y el análisis de las diferencias entre el resultado esperado y el obtenido son las claves para que los planes de pruebas funcionaran. Estos planes individuales divididos por funcionalidad sirven a su vez para hacer pruebas globales del sistema, y tras finalizar cada sprint, además de añadir los nuevos planes a la lista, se repetían los existentes para comprobar que todo seguía en orden tras la inclusión del nuevo código.

Además, estos planes de pruebas sirven, a su vez, para verificar todos los requisitos funcionales especificados en la sección 8.2. Todas las funcionalidades del sistema se implementaron siguiendo la especificación descrita en el apartado de especificación de requisitos, por lo tanto, el buen funcionamiento de todos los planes de pruebas implica la verificación de los requisitos funcionales propuestos.

A continuación, se pueden ver los distintos planes elaborados para probar el buen funcionamiento del sistema software expuesto a lo largo de la memoria.

| #1 | Registro |
|---|---|
| Descripción | Para poder registrarse, se tienen que introducir una serie de parámetros en la pantalla de registro, algunos tienen una serie de restricciones que se tienen que cumplir para que el registro se pueda completar. |
| Input | • DNI (obligatorio y válido).<br>• Nombre (obligatorio).<br>• Primer apellido (obligatorio).<br>• Contraseña (obligatoria y equivalente a confirmar contraseña).<br>• Confirmar contraseña (obligatoria y equivalente a contraseña).<br>• Fecha de nacimiento (obligatoria y válida).<br>• Segundo apellido.<br>• Correo electrónico (obligatorio si no se introduce número de teléfono).<br>• Número de teléfono (obligatorio si no se introduce correo electrónico). |
| Escenario principal | 1. En la pantalla de inicio de sesión, pulsar sobre el botón de registrarse.<br>2. Navegar hacia la pantalla de registro.<br>3. Cumplimentar el formulario.<br>4. Pulsar sobre el botón para completar el registro. |
| Resultado esperado | Una vez se pulsa el botón de registrarse, se navega hacia la pantalla principal donde se puede apreciar un mensaje de bienvenida con el nombre que se acaba de introducir en el formulario. Además, al tratarse de un proyecto académico, se añaden analíticas, citas y medicaciones de ejemplo que el usuario puede ver una vez registrado. |
| Escenario secundario | Se produce algún error en el formulario. |
| Resultado esperado | • Campo obligatorio no introducido → Se resalta el campo que no ha estado cumplimentado indicando la obligatoriedad de este.<br>• DNI no válido → Se resalta el campo correspondiente al DNI indicando que el DNI introducido no es válido.<br>• Contraseñas no equivalentes → Se resaltan los campos correspondientes a las contraseñas y se indica que no coinciden.<br>• Fecha de nacimiento no válida → Se resalta el campo y se indica que la fecha tiene que ser anterior hoy en día.<br>• Correo electrónico y número de teléfono no introducidos → Se resaltan ambos campos y se indica que se tiene que rellenar al menos uno. |

| #2 | Inicio de sesión |
|---|---|
| Descripción | Para iniciar sesión, el usuario tiene que introducir un usuario y una contraseña. |
| Input | • DNI (obligatorio).<br>• Contraseña (obligatoria). |
| Escenario principal | 1. Introducir los datos de inicio de sesión.<br>2. Pulsar sobre el botón de inicio de sesión. |
| Resultado esperado | Se navega hacia la pantalla principal donde se puede ver el nombre del usuario, si se accede a cualquier sección de la aplicación se puede apreciar cómo se han cargado todos sus datos. |
| Escenario secundario | Se produce algún error en el inicio de sesión. |

| | |
|---|---|
| Resultado esperado | • Usuario y contraseña no coinciden → Se informa al usuario de que existe algún problema en alguno de los campos para que lo vuelva a intentar.<br>• Usuario no existe → Se informa al usuario para que se registre. |

| #3 | Pantalla principal |
|---|---|
| Descripción | La pantalla principal es el eje de toda aplicación, des de este punto se puede navegar hacia el resto de las secciones y es fundamental que esto sea posible. |
| Input | Interacción del usuario con todos los botones (uno cada vez). |
| Escenario principal | • *Click* sobre el botón de ajustes.<br>• *Click* sobre el botón de análisis.<br>• *Click* sobre el botón de medicación.<br>• *Click* sobre el botón de calendario.<br>• *Click* sobre el botón de chat. |
| Resultado esperado | Tras pulsar sobre cada uno de los botones se navega correctamente hacia la pantalla que corresponda. Esto se puede apreciar en que la barra superior contiene el nombre de la sección sobre la cual se acaba de pulsar. |

| #4 | Navegación con el drawer |
|---|---|
| Descripción | El drawer (o menú lateral) nos permite navegar por las distintas secciones de la aplicación, además nos muestra el usuario que tiene la sesión iniciada y nos resalta la sección de la aplicación en la que nos encontramos. |
| Input | Interacción del usuario con alguno de los submenús que tiene el drawer. |
| Escenario principal | • *Click* sobre el submenú de ajustes.<br>• *Click* sobre el submenú de análisis.<br>• *Click* sobre el submenú de medicación.<br>• *Click* sobre el submenú de calendario.<br>• *Click* sobre el submenú de chat. |
| Resultado esperado | Al clicar sobre alguno de los submenús anteriores, se navega hacia dicha sección, la barra superior muestra en todo momento la sección en la que nos encontramos y, además, si volvemos a abrir el drawer este nos resalta la sección en la que nos encontramos. |

| #5 | Visualización de la pantalla de ajustes |
|---|---|
| Descripción | La pantalla de ajustes contiene los datos de contacto y una serie de configuraciones que el usuario puede hacer para tratar de adaptar la aplicación a sus gustos. |
| Input | Interacción del usuario con el botón de ajustes ubicado en la pantalla principal. |
| Escenario principal | Se pulsa sobre el botón de ajustes. |
| Resultado esperado | La página de ajustes se visualiza, dentro podemos observar dos campos que nos muestran los datos de contacto (número de teléfono y/o correo electrónico), un pequeño formulario para el cambio de contraseña, y un selector de idiomas. |

| #6 | Cambio de contraseña |
|---|---|
| Descripción | Dentro de la sección de ajustes existe un pequeño formulario que permite al usuario modificar su contraseña actual por una de nueva. |
| Input | • Contraseña actual (obligatorio).<br>• Nueva contraseña (obligatorio y equivalente a confirmar contraseña).<br>• Confirmar contraseña (obligatorio y equivalente a nueva contraseña). |
| Escenario principal | 1. Navegar hacia la sección de ajustes.<br>2. Cumplimentar el formulario introduciendo la contraseña actual y repitiendo dos veces la nueva contraseña.<br>3. Pulsar sobre el botón de confirmar.<br>4. Cerrar sesión.<br>5. Introducir los datos de acceso utilizando la contraseña que acabamos de introducir como nueva. |
| Resultado esperado | Aparece un mensaje en pantalla informando de que el cambio se ha realizado correctamente, ahora el usuario accede al sistema con la nueva contraseña.<br><br>Tras cerrar sesión, el sistema nos permite iniciarla de nuevo con la contraseña modificada y no con la antigua. |
| Escenario secundario | Error tras pulsar sobre el botón de confirmar. |
| Resultado esperado | • Contraseñas no coinciden → Se resaltan los campos de las contraseñas indicando que no coinciden.<br>• Cambio no realizado → La contraseña introducida no coincide con la actual del usuario, se le informa de que no se ha podido realizar el cambio. |

| #7 | Cambio de idioma |
|---|---|
| Descripción | Dentro de ajustes, se puede modificar el idioma por defecto de la aplicación. |
| Input | Interacción del usuario con el selector de idioma ubicado en la pantalla de ajustes. |
| Escenario principal | El usuario interacciona con el selector de idioma seleccionando un idioma distinto al actual. |
| Resultado esperado | Aparece un mensaje informando de la modificación del idioma y, tras reiniciar la aplicación se puede ver que el idioma ha cambiado. |

| #8 | Visualizar analíticas |
|---|---|
| Descripción | Una de las funcionalidades principales de la aplicación es visualizar analíticas. Dentro de su correspondiente sección aparecen en formato tarjeta todas las disponibles. Dichas tarjetas contienen dos botones: consultar y compartir. En esta prueba nos centraremos en el de consultar. |
| Input | Interacción del usuario con la UI. |
| Escenario principal | 1. El usuario navega hacia la sección de analíticas<br>2. Pulsa sobre el botón de consultar disponible en cada tarjeta. |
| Resultado esperado | Se abre un diálogo con los datos completos de la analítica que el usuario ha consultado.<br><br>La tarjeta de la analítica tiene que contener lo siguiente:<br>• Fecha de la analítica.<br>• Enfermedad que controla la analítica.<br>• Si se han realizado cambios o no en la medicación a partir de los resultados de esa analítica.<br>• Los botones de consultar y compartir. |
| Escenario secundario | El usuario no tiene analíticas para visualizar. |
| Resultado esperado | En la pantalla se muestra un texto informativo indicando que no hay analíticas para visualizar. |

| #9 | Añadir analítica |
|---|---|
| Descripción | Desde Postman se pueden añadir analíticas de los dos tipos que soporta actualmente el sistema. Tras añadirlas, la API las procesa y a partir de ese momento el usuario tendrá una nueva analítica disponible. |
| Input | Introducir los datos correspondientes a la analítica (depende de la enfermedad a controlar) en Postman. El input exacto se puede ver en el apartado 9.3.3. |
| Escenario principal | La API procesa la analítica y la guarda en base de datos correctamente. |
| Resultado esperado | Cuando el usuario entra de nuevo en la sección de analíticas (o refresca la página) le aparece la nueva que se acaba de añadir, con todos los datos introducidos desde Postman. Además, el usuario recibe una notificación que le informa de que tiene una nueva analítica disponible. |

| #10 | Procesado de analítica |
|---|---|
| Descripción | Cuando se añade una analítica, la API la procesa, y dependiendo de una serie de procedimientos ya comentados determina si es necesario o no un cambio en la medicación del usuario. |
| Input | Introducir los datos correspondientes a la analítica (depende de la enfermedad a controlar) en Postman. El input exacto se puede ver en el apartado 9.3.3. |
| Escenario principal | La API procesa la analítica y guarda los cambios en la medicación del paciente en caso de que se produzcan. |
| Resultado esperado | En caso de realizarse cambios en la medicación, se notifica al usuario informándole. Si este entra en el apartado de medicación, podrán verse dichos cambios reflejados. |

| #11 | Procesado de hipotiroidismo |
|---|---|
| Descripción | Este es una extensión del anterior, cada una de las enfermedades que están parametrizadas en el sistema tiene un procesado completamente distinto, es por eso por lo que necesitan probarse de forma distinta. |
| Input | Introducir los datos correspondientes a la analítica (TSH) en Postman. El input exacto se puede ver en el apartado 9.3.3. |
| Escenarios principales | • Incremento de la medicación → El valor de TSH contenido en la analítica es superior al máximo, por lo tanto, se tiene que incrementar la medicación que toma el paciente.<br>• Reducción de la medicación → El valor de TSH de la analítica es inferior al mínimo, se tiene que reducir la medicación del paciente.<br>• No hacer nada → El valor de TSH de la analítica se encuentra entre el mínimo y el máximo y por lo tanto no se tiene que modificar la medicación del paciente. |
| Resultados esperados | En caso de que la medicación resulte modificada, se notifica al usuario, este podrá ver las modificaciones en la sección correspondiente, en caso contrario, el usuario no verá nada. |

| #12 | Procesado de la hipercolesterolemia |
|---|---|
| Descripción | Esta prueba sirve para verificar que el procesado de la enfermedad se lleva a cabo de forma satisfactoria. |
| Input | Introducir los datos correspondientes a la analítica (cLDL y colesterol total) en Postman. El input exacto se puede ver en el apartado 9.3.3. |
| Escenarios principales | • Incremento de la medicación → El porcentaje calculado a partir del valor máximo de cLDL y el contenido en la analítica es superior al máximo, por lo tanto, se tiene que incrementar la medicación que toma el paciente.<br>• Cambio de medicación → No existen dosis superiores de la medicación que está tomando el paciente.<br>• No hacer nada → El valor de cLDL de la analítica es bueno y por lo tanto no se tiene que modificar la medicación del paciente. |
| Resultados esperados | En caso de que la medicación resulte modificada, se notifica al usuario, este podrá ver las modificaciones en la sección correspondiente. Para el caso en el que se tiene que cambiar de medicación, el sistema notifica al usuario informándole de que tiene que visitar su centro de salud. En el último caso (no hacer nada) no se notifica al usuario y por lo tanto no ocurre nada. |

| #13 | Visualizar medicación |
|---|---|
| Descripción | Esta prueba tiene la intención de comprobar que la medicación se está mostrando de forma correcta. |
| Input | Interacción del usuario. |
| Escenario principal | El usuario navega hacia la pantalla de medicación. |
| Resultado esperado | Se muestran las medicaciones del usuario en formato tarjeta, cada tarjeta tiene que contener lo siguiente:<br>• Enfermedad a tratar.<br>• Medicamentos seguidos de sus respectivas dosis.<br>• Botón de compartir. |
| Escenario secundario | El usuario no tiene medicaciones para visualizar. |
| Resultado esperado | En la pantalla se muestra un texto informativo indicando que no hay medicaciones para visualizar. |

| #14 | Compartir medicación |
|---|---|
| Descripción | El usuario tiene la posibilidad de compartir su medicación, esta prueba está diseñada para comprobar que está funcionalidad funciona y que el texto que se envía es correcto. |
| Input | Interacción del usuario con la aplicación. |
| Escenario principal | 1. El usuario navega hacia la pantalla de medicación.<br>2. El usuario interactúa con el botón “Compartir” de la tarjeta.<br>3. Se abre un menú de compartir (nativo del dispositivo del usuario).<br>4. El usuario selecciona un método para compartir el mensaje. |
| Resultado esperado | Se envía un mensaje con la siguiente información:<br>• Mensaje informando que se está compartiendo una medicación.<br>• Medicamentos y dosis asociadas. |

| #15 | Compartir analítica |
|---|---|
| Descripción | El usuario tiene la posibilidad de compartir su analítica, esta prueba está diseñada para comprobar que está funcionalidad funciona y que el texto que se envía es correcto. |
| Input | Interacción del usuario con la aplicación. |
| Escenario principal | 1. El usuario navega hacia la pantalla de análisis.<br>2. El usuario interactúa con el botón “Compartir” de la tarjeta.<br>3. Se abre un menú de compartir (nativo del dispositivo del usuario).<br>4. El usuario selecciona un método para compartir el mensaje. |
| Resultado esperado | Se envía un mensaje con la siguiente información:<br>• Mensaje informando que se está compartiendo una analítica.<br>• Fecha del análisis.<br>• Enfermedad controlada por el análisis.<br>• Resultados del análisis. |

| #16 | Añadir cita |
|---|---|
| Descripción | Desde Postman se pueden añadir citas. Tras añadirlas, la API las procesa y a partir de ese momento el usuario tendrá una nueva cita disponible. |
| Input | Introducir los datos correspondientes a la cita en Postman. El input exacto se puede ver en el apartado 9.3.3. |
| Escenario principal | La API procesa la cita y la guarda en base de datos correctamente. |
| Resultado esperado | Cuando el usuario entra de nuevo en la sección de calendario (o refresca la página) le aparece la nueva que se acaba de añadir, con todos los datos introducidos desde Postman. Además, el usuario recibe una notificación que le informa de que tiene una nueva cita disponible. |

| #17 | Visualizar citas |
|---|---|
| Descripción | Esta prueba tiene la intención de comprobar que las citas se están mostrando de forma correcta. |
| Input | Interacción del usuario. |
| Escenario principal | El usuario navega hacia la pantalla de calendario. |
| Resultado esperado | Se muestran las citas del usuario en formato tarjeta, cada tarjeta tiene que contener lo siguiente:<br>• Fecha de la cita.<br>• Enfermedad a tratar.<br>• Hora de la cita seguida del lugar.<br>• Botón de reclamar y de compartir. |
| Escenario secundario | El usuario no tiene citas para visualizar. |
| Resultado esperado | En la pantalla se muestra un texto informativo indicando que no hay citas para visualizar. |

| #18 | Compartir cita |
|---|---|
| Descripción | El usuario tiene la posibilidad de compartir su cita, esta prueba está diseñada para comprobar que está funcionalidad funciona y que el texto que se envía es correcto. |
| Input | Interacción del usuario con la aplicación. |
| Escenario principal | 1. El usuario navega hacia la pantalla de calendario.<br>2. El usuario interactúa con el botón “Compartir” de la tarjeta.<br>3. Se abre un menú de compartir (nativo del dispositivo del usuario).<br>4. El usuario selecciona un método para compartir el mensaje. |
| Resultado esperado | Se envía un mensaje con la siguiente información:<br>• Mensaje informando que se está compartiendo una cita.<br>• Enfermedad controlada por el análisis.<br>• Fecha de la cita.<br>• Hora de la cita.<br>• Lugar de la cita. |

| #19 | Reclamar cita |
|---|---|
| Descripción | Cuando a un usuario no le va bien la hora o el día asignados en una cita, la puede reclamar para solicitar una modificación en la misma. |
| Input | Interacción del usuario con la aplicación. |
| Escenario principal | 1. Navegar hacia la pantalla de calendario.<br>2. Pulsar sobre el botón de reclamar.<br>3. Se abre un diálogo para seleccionar día.<br>4. Se selecciona un día.<br>5. El mismo diálogo ahora muestra las horas de dicho día.<br>6. Se selecciona una hora.<br>7. Se confirma la reclamación. |
| Resultado esperado | Aparece un mensaje informando que la solicitud se ha procesado correctamente. |

| #20 | Visualizar chats |
|---|---|
| Descripción | Esta prueba tiene la intención de comprobar que los chats se están mostrando de forma correcta. |
| Input | Interacción del usuario. |
| Escenario principal | El usuario navega hacia la pantalla de chats. |
| Resultado esperado | Se muestran los chats del usuario en formato tarjeta, cada tarjeta tiene que contener lo siguiente:<br>• Especialidad de la consulta.<br>• Último mensaje. |
| Escenario secundario | El usuario no tiene chats para visualizar. |
| Resultado esperado | En la pantalla se muestra un texto informativo indicando que no hay chats para visualizar. |

| #21 | Iniciar nuevo chat |
|---|---|
| Descripción | Si el paciente tiene una consulta, tiene que poder iniciar un chat nuevo para poder realizarla. Esta prueba tiene la intención de comprobar que se puedan iniciar chats. |
| Input | • Interacción del usuario con la aplicación.<br>• Selección de especialidad a la que irá realizada la consulta.<br>• Redacción y envío de la consulta. |
| Escenario principal | 1. El usuario navega hacia la sección de chats.<br>2. Se pulsa sobre el botón de nuevo chat.<br>3. Se abre un diálogo con un desplegable para seleccionar especialidad y un input para redactar el mensaje.<br>4. Se cumplimentan los dos campos.<br>5. Se envía el mensaje. |
| Resultado esperado | Se abre una nueva pantalla, esta pantalla será la correspondiente a dicho chat, la cual ha de contener lo siguiente:<br>• En la barra superior: nombre de la especialidad y botón para regresar.<br>• Mensaje redactado en el input del diálogo.<br>• Input inferior para redactar mensajes.<br>• Botón para enviar mensajes. |
| Escenario secundario | El usuario no cumplimenta alguno de los campos requeridos en el diálogo. |

| | |
|---|---|
| Resultado esperado | Se remarca el campo no cumplimentado y se informa al usuario que tiene que hacerlo para poder tramitar su consulta. |

| #22 | Responder mensaje (enfermera) |
|---|---|
| Descripción | Cuando un usuario realiza una consulta, espera una respuesta, esta prueba sirve para verificar que se puede dar esta respuesta al paciente, y que este la puede ver en el chat correspondiente. |
| Input | Introducir los datos correspondientes al mensaje en Postman. El input exacto se puede ver en el apartado 9.3.3. |
| Escenario principal | La API procesa el mensaje y lo guarda en base de datos correctamente. |
| Resultado esperado | • El usuario recibe una notificación informando que hay un nuevo mensaje, dicha notificación contiene el contenido del mensaje.<br>• En el chat se puede ver la respuesta de la enfermera.<br>• En caso de estar dentro del chat, al recibir respuesta, este se refresca y aparece el nuevo mensaje. |

| #23 | Responder mensaje (paciente) |
|---|---|
| Descripción | El paciente, tras iniciar un chat, también puede volver a enviar otro mensaje o incluso responder tras recibir respuesta por parte de la enfermera. |
| Input | Contenido del mensaje nuevo e interacción con la aplicación para enviarlo. |
| Escenario principal | El usuario, dentro de un chat iniciado, redacta un mensaje y pulsa enviar. |
| Resultado esperado | El mensaje enviado aparece en la pantalla y el teclado se vuelve a guardar. |

| #24 | Cerrar sesión |
|---|---|
| Descripción | Tanto desde ajustes, como abriendo el menú lateral (drawer) se puede cerrar sesión. |
| Input | Interacción del usuario. |
| Escenario principal | Se cierra sesión desde todos los sitios en los que se pueda. |
| Resultado esperado | Se regresa a la pantalla de inicio de sesión, al pulsar sobre el botón de ir hacia atrás desde esta pantalla se cierra la aplicación. |

## 12.3 Verificación de los requisitos no funcionales

Para la verificación de muchos de los requisitos no funcionales se han realizado encuestas a los distintos *stakeholders* que han colaborado en la fase de pruebas en la aplicación. Los resultados de las encuestas se pueden encontrar en los anexos.

Todos los datos que se van a comentar a continuación han sido extraídos de las respuestas de 37 usuarios con el rol de pacientes y 19 con el rol de personal sanitario.

| Apariencia | |
|---|---|
| **Justificación de la condición de satisfacción 1:** El 90% de los usuarios consideran que la aplicación es estéticamente agradable. | Un total de 35 personas (95%) concluyeron que la aplicación era estéticamente agradable. |

| Estilo | |
|---|---|
| **Justificación de la condición de satisfacción 1:** El sistema ha de seguir los estilos definidos para Android. | El sistema ha sido desarrollado utilizando los componentes de Material Design y las guías de estilado para Android. |
| **Justificación de la condición de satisfacción 2:** El 90% del personal sanitario contactado debe tener la visión de que el sistema planteado es un producto serio | Las 19 (100%) personas contactadas del ámbito sanitario han afirmado que se trata de un producto serio y que su estilo da a entender exactamente lo que es, un sistema para ayudar con el control de las enfermedades crónicas del paciente. |

| Usabilidad | |
|---|---|
| **Justificación de la condición de satisfacción 1:** El 80% de los usuarios tienen que ser capaces de acceder a las cuatro secciones principales de la aplicación (análisis, medicación, calendario y chat) de forma rápida y sin ninguna dificultad. | 31 personas (83%) fueron capaces de acceder a las cuatro secciones sin ninguna dificultad añadida, el resto se trata probablemente de personas mayores o personas con dificultades a la hora de utilizar dispositivos electrónicos. |

| Internacionalización | |
|---|---|
| **Justificación de la condición de satisfacción 1:** Variedad de idiomas en la aplicación. | Actualmente se puede escoger entre catalán y castellano para hacer uso de la aplicación. |
| **Justificación de la condición de satisfacción 2:** Añadir un idioma nuevo es una tarea sencilla. | Añadir un idioma es tarea sencilla, simplemente requiere de traducir unos .xml al idioma que queramos añadir y el nuevo idioma al enum Languagues ya comentado. |

| Aprendizaje | |
|---|---|
| **Justificación de la condición de satisfacción 1:** Un ingeniero puede utilizar la totalidad de funcionalidades de la aplicación sin problemas. | Se ha consultado con 7 compañeros, todos ingenieros informáticos especializados en ingeniería del software, y todos han concluido que tanto hacer uso de la aplicación como aprender sus funcionalidades son tareas sencillas. |
| **Justificación de la condición de satisfacción 2:** Después de utilizar el sistema durante 10 horas, los usuarios tienen que haber aprendido cómo funciona cerca del 80% de la aplicación. | Al igual que en el caso del uso de las cuatro funcionalidades principales, solo 31 (83%) fueron capaces de aprender cómo funciona el 80% de la aplicación lo cual corrobora la teoría de que el resto pueden ser personas con dificultades asociadas al manejo de la tecnología. |

| Velocidad y latencia | |
|---|---|
| **Justificación de la condición de satisfacción 1:** Una vez se dispongan de los cambios en la medicación, el usuario los tiene que recibir en su teléfono en menos de 1 minuto. | Actualmente, el tiempo que se tarda en procesar una analítica y enviar los cambios en la medicación al usuario, es negligible e inferior a 1 segundo. |

| #8 | Seguridad y privacidad |
|---|---|
| **Justificación de la condición de satisfacción 1:** El sistema software mantendrá los datos claves de los usuarios cifrados en base de datos. | Los datos claves, como la contraseña, se encuentran cifrados en la base de datos y son ilegibles. Además, la base de datos solo es accesible a través de un usuario y una contraseña solo conocidas por el autor de la memoria. |

| #9 | Exactitud |
|---|---|
| **Justificación de la condición de satisfacción 1:** El 95% de los resultados emitidos durante la fase de prueba coinciden con el criterio de un doctor especializado en el tratamiento crónico para el cual se están generando resultados. | De los 19 sanitarios contactados, 5 son doctores que ejercen actualmente. Los 5 han concluido que todos los resultados emitidos por el sistema coinciden con los resultados que emitirían ellos en una visita médica. |

| #10 | Fiabilidad y disponibilidad |
|---|---|
| **Justificación de la condición de satisfacción 1:** El sistema tiene que estar disponible el 99% del año. | Desde que se puso en producción la base de datos y la API (15 de mayo de 2021) el sistema ha estado disponible todo el tiempo. |

| #11 | Escalabilidad y extensibilidad |
|---|---|
| **Justificación de la condición de satisfacción 1:** Se podrán ir incluyendo enfermedades crónicas progresivamente de forma muy sencilla. | La justificación de este requisito no funcional ya se ha visto en el apartado 10.4. |

| #12 | Longevidad |
|---|---|
| **Justificación de la condición de satisfacción 1:** El sistema es sencillo de mantener, y, por lo tanto, es sencillo realizar actualizaciones que le permitan tener una vida más longeva. | El análisis del código comentado en el apartado 12.1 nos demuestra que se cuenta con un código fácil de mantener, es por eso por lo que realizar actualizaciones que prolonguen la vida del sistema software no será una tarea complicada. |

| #13 | Adaptabilidad |
|---|---|
| **Justificación de la condición de satisfacción 1:** El producto inicialmente se tendrá que poder ejecutar en Android. | Actualmente, la aplicación Android se encuentra en producción y se puede ejecutar en todos los teléfonos con una versión de Android >= a la 8.0. |

# 13. Sostenibilidad

Cuando pensamos en sostenibilidad y lo relacionamos con la informática, siempre pensamos que se trata de un sector que contribuye poco o nada al bien del planeta, sin embargo, existen una gran cantidad de proyectos que nos demuestran que esto es realmente mentira.

Si bien durante la carrera de ingeniería informática no se le da mucho peso a la sostenibilidad, sí que hay ciertas asignaturas como AC (Arquitectura del Computador) y CI (*Computer Interfacing*) que le dedican parte del temario u organizan charlas sobre el tema. Es por este motivo que los conocimientos que llegamos a tener sobre sostenibilidad al finalizar la carrera no son excesivos, pero por lo menos tenemos una base, cosa que he visto reflejado al ir contestando la encuesta EDINSOST [56]. Sin embargo, creo que al finalizar cada asignatura se debería de poner en contexto el contenido impartido en la misma sobre el planeta, es decir, explicar toda la dimensión social y medioambiental implicada en dicha asignatura.

Como ya he comentado, hubo varias asignaturas que hicieron énfasis en la parte sostenible de los proyectos informáticos, pero estas eran de una especialidad muy diferente a la que yo estoy cursando (ingeniería del software). Considero que deberían de existir al menos una o dos asignaturas dentro de cada especialidad que remarcaran la importancia de diseñar e implementar sistemas realmente sostenibles.

En cuanto al alcance social de los proyectos, para mi es algo fundamental, y de hecho este proyecto trata de mejorar el ámbito sanitario, proporcionando una mayor independencia a personas que por desgracia sufren de patologías crónicas. Como ya dije al comienzo de la memoria, creo que todos deberíamos aprovechar este trabajo de fin de grado para intentar aportar soluciones innovadoras y que definitivamente ayuden a la sociedad a seguir evolucionando. Considero que, en definitiva, de eso se trata el ser un buen ingeniero.

## 13.1 Dimensión económica

El coste de un proyecto informático como el que se plantea en esta memoria es algo que se debe tener muy en cuenta, es por eso por lo que se han estimado de la forma más exacta posible todos los costes tanto humanos (trabajadores) como materiales (oficina y material). Además, en cuanto al coste asociado a la oficina, se tuvo en cuenta todo lo necesario para poder utilizar la misma: internet, luz, calefacción…

Para tratar de ajustarme a un presupuesto real, se escogió un material informático de coste reducido. No es necesario contar con un ordenador de última generación para realizar el proyecto, y es por eso por lo que se escogió uno con un nivel calidad-precio razonable. Sin embargo, el contar con oficina hace que los costes se disparen y aún se dispararían más si el proyecto se alargara, es por eso por lo que la opción del teletrabajo sería perfecta para tratar de reducir los costes asociados a la oficina.

Actualmente, y poniendo el foco sobre los usuarios finales del sistema, los pacientes crónicos tienen que visitar de forma presencial su centro de salud, esto hace que en algunos casos estas personas tengan que faltar al trabajo, y, por lo tanto, perder el dinero de esa jornada (autónomos o empresarios), y a eso hay que añadirle los costes relacionados con la llegada al centro (transporte público, combustible…). Además, los centros de salud ocupan horas para las visitas de dichos pacientes crónicos que podrían emplear en otras cosas que requieran verdaderamente presencialidad. Es por eso por lo que la implementación, en el sistema sanitario o en centros privados, de mi proyecto ahorraría dinero de forma directa al sistema de salud y a los pacientes, ya que estos no tendrían que visitar su centro de salud y podrían simplemente recibir los cambios en su medicación des de su casa.

Sin embargo, y debido a la pandemia mundial que estamos viviendo, el sistema de salud se está informatizando rápidamente. Gracias a esto, es posible que dentro de unos pocos años acabe existiendo un proyecto que se asemeje a lo que propongo y que pueda poner en verdadero riesgo su viabilidad. De todos modos, hoy en día, no existe nada documentado que refleje que esto está sobre la mesa.

## 13.2 Dimensión ambiental

Al tratarse de un sistema software, el impacto ambiental que tendrá será muy pequeño, si más no, se dispondría de una oficina con todo el impacto que eso representa, y, además, se tendría que adquirir material informático, ambas son cosas que impactan fuertemente en el planeta. De todos modos, se ha tratado de reutilizar todo tipo de material, como por ejemplo los monitores. Además, en caso de que el proyecto se alargue, podremos seguir contando con los mismos ordenadores evitando comprar más, con todo el ahorro medioambiental que eso significa.

Sin embargo, considero que el sistema que se propone mejorará mucho la condición ambiental actual, ya que, como se ha comentado, actualmente los pacientes tienen que trasladarse físicamente a su centro de salud, normalmente utilizando un vehículo que emite emisiones constantes de $CO_2$. Es por eso, que la implantación de este sistema contribuirá a la reducción de la huella ecológica que suponen los transportes en vehículos a motor.

## 13.3 Dimensión social

Tanto mi pareja como mi madre son pacientes crónicas que sufren de hipotiroidismo. Des de pequeño he vivido como mi madre tiene que ir periódicamente a su centro de salud para que le revisen la medicación. Es por eso por lo que construir un sistema software que realmente les pueda ayudar tanto a ellas como a miles de personas que además de su enfermedad tienen que lidiar con ir cada dos por tres a su centro de salud, supondrá para mí, todo un logro como persona y como ingeniero. Además, gracias a este proyecto estoy comprendiendo aún más cómo funciona el ámbito sanitario a nivel interno en cuanto a análisis de sangre y tratamiento de enfermedades se refiere.

Si este proyecto se llegara a implementar en el sistema de salud, supondría una mejora abismal en la calidad de vida de estos pacientes, gracias a este sistema ganarían una independencia inimaginable sin su existencia. Hoy en día, no existe nada que se le parezca y que pueda llegar a evitar que una persona deje de ir a su centro de salud para que le retoquen la medicación. Además, dado el colapso del sistema sanitario, es esencial el poder aliviar la carga que este sufre de una forma u otra.

Sin embargo, existe un grupo de usuarios que podrían ser reticentes al uso del sistema: las personas mayores. Son un colectivo que no está familiarizado con las nuevas tecnologías, y generalmente son personas que prefieren ir al centro de salud a que su doctor les explique cómo ha ido la analítica y los motivos de los cambios en la medicación. De todas formas, no considero que la implementación del sistema sea perjudicial para ellos, ya que siempre podrían decidir acudir presencialmente a su centro de salud.

## 13.4 Reflexión final

La oportunidad que se nos brinda al poder cursar un grado universitario como es una ingeniería informática es enorme. Esta oportunidad se ve mejorada por el hecho de que la FIB es una universidad pública, y, por lo tanto, los precios de la matrícula son reducidos o casi gratuitos para la gente que, como yo, contamos con beca de matrícula. Es por esto que considero que tenemos que aprovechar esta enorme oportunidad para dejar huella en la sociedad en la que vivimos, tratando de mejorar de una manera u otra la forma de vida de la gente, a nuestra manera.

Considero que el sistema software que planteo será con creces muy beneficioso para la sociedad actual, y, sin duda, será de mucha ayuda para esas personas que sufren de patologías crónicas. Este es el motivo por el que pienso que su puesta en producción supondría un antes y un después en el colectivo de personas con enfermedades crónicas.

# 14. Leyes y regulaciones

**Ley básica reguladora de la autonomía del paciente y de derechos y obligaciones en materia de información y documentación clínica (Ley 41/2002, 14 de noviembre)** [57].
Esta ley regula todos los derechos y obligaciones que tienen tantos los pacientes, profesionales, centros y servicios sanitarios tanto públicos como privados en materia de la autonomía del paciente y de la información y documentación clínica.

*Artículos 4 al 6*
El paciente tiene derecho a ser informado verbalmente de cualquier actuación en el ámbito de su salud y de no ser informado si no lo desea.

**Explicación:** El paciente siempre podrá escoger de hacer uso o no del sistema, nunca será obligatorio su uso, y, además, un paciente siempre se podrá dar de baja si no desea continuar recibiendo información en su dispositivo.

*Artículo 7: Confidencialidad de los datos*
**Explicación:** En el caso del sistema propuesto, es el paciente quien se da de alta mediante el registro, esto es debido a que se trata de un prototipo académico. En un caso real, se le proporcionarían unos datos de acceso al paciente, los cuales serían únicos y solamente distribuibles por su parte.

*Artículo 8: Consentimiento informado*
**Explicación:** El paciente podrá decidir darse de baja del sistema en cualquier momento, notificado su intención de dejar de hacer uso de este en su centro de salud más cercano.

**Ley 39/2006 de 14 de diciembre, de promoción de la autonomía personal y atención a las personas en situación de dependencia** [58].
Los principios de la ley más relevantes para este proyecto son los siguientes:

- Promoción de condiciones para que las personas dependientes puedan llevar una vida con más autonomía.
- Calidad, sostenibilidad y accesibilidad de los servicios.

**Explicación:** Uno de los principales objetivos de la construcción de este este sistema es el de proporcionar más autonomía a esas personas que padecen enfermedades crónicas. Evitar que tengan que visitar su CAP cada vez que reciben los resultados de una analítica es algo muy positivo y que sin duda logrará grandes cambios en las vidas de estas personas.

**Ley de protección de datos personales y garantía de los derechos digitales (Ley Orgánica 3/2018 de 5 de diciembre)** [59].
El objetivo de esta ley es el de adaptar el ordenamiento jurídico español al Reglamento de la Unión Europea 2016/679, relativo a la protección de las personas físicas en cuanto al tratamiento de sus datos se refiere.

Explicación: Los datos tratados en el sistema que se propone son puramente académicos, es decir, han sido introducidos manualmente por el desarrollador para demostrar que el sistema puede funcionar en un ámbito real. Es por este motivo, que no se incumple ninguna ley de protección de datos, pues todos los datos son inventados. En caso de comenzar a trabajar con datos de pacientes reales, este sistema estaría integrado o bien en el sistema de salud público, o bien en un sistema privado, los cuales ya cuentan con un control exhaustivo que verifican que estos datos sean seguros.

Ley General de Sanidad (Ley 14/1986, 25 de abril) [60].
De esta ley, nos interesa particularmente el artículo 10, el cual nos habla de los derechos que tienen los usuarios del sistema de salud, de entre todos ellos destacamos los siguientes:

- A la información de los servicios a los que puede acceder, siendo advertido de los procedimientos que se le vayan a aplicar.
- Confidencialidad de sus datos en el proceso.
- Ser advertido si va a formar parte de algún proyecto docente o de investigación, dando autorización para los mismos.

Explicación: De nuevo, esta memoria presenta un proyecto universitario, y por lo tanto no cuenta con datos reales. De todas formas, en caso de que el proyecto siguiera adelante, se tendría que aplicar esta ley para poder empezar a poner en producción el sistema.

# 15. Cierre del proyecto y conclusiones

## 15.1 Cumplimiento de los objetivos

Para verificar el cumplimiento de los objetivos, se van a listar a continuación los propuestos en el apartado 3.1 seguidos de la justificación sobre su cumplimiento.

### Disponer de un sistema software para el intercambio de información entre los miembros del equipo sanitario y sus respectivos pacientes.

Este objetivo tenía tres partes:

- Disponer de tipos de cuentas distintas para que cada tipo de usuario pueda acceder a una información en concreto.
- Mostrar y representar la información de forma clara para que sea fácilmente entendible por el paciente.
- Notificar al paciente cada vez que tenga información nueva para consultar.

En primer lugar, se distinguió entre el uso de la aplicación para los pacientes, y el uso de Postman para simular la interacción que tendrían el resto de los usuarios con sus respectivos Dashboards. De esta forma, un paciente nunca podrá realizar acciones que solo estarán disponibles para un cierto tipo de usuario. En segundo lugar, se han diseñado unas tarjetas contenedoras de información que siguen el mismo formato en toda la aplicación, facilitando de esta manera la comprensión de la información, y la lectura rápida de cara al usuario. Finalmente, cada vez que se añade nueva información de interés para el usuario se le notifica, y pulsando sobre la notificación el paciente puede acceder directamente a la nueva información disponible.

### Mejorar la comunicación, haciéndola inmediata, entre el sistema sanitario y los pacientes.

Este objetivo consistía en diseñar e implementar un chat que permitiera la comunicación directa entre paciente y enfermera, de esta forma, en caso de que un paciente tuviese una consulta de carácter no urgente, no tendría que llamar a su centro de salud o a los números de teléfono facilitados por el estado para este tipo de situaciones. El chat, como ya se ha visto, se ha implementado con éxito y está operativo actualmente. Sin embargo, no se ha podido probar en situaciones reales pues este proyecto es estrictamente académico.

### Aumentar el número de visitas que puede realizar un doctor/a.

La automatización del procesado de las analíticas de control de las enfermedades que se trabajan en esta memoria ha sido completada con éxito. En caso de añadir una analítica para una de las dos enfermedades, se procesa y se genera una respuesta, evitando de esta manera que este proceso lo tenga que realizar un doctor, con todo el ahorro de tiempo que esto implica.

Reducir el número de veces que un paciente crónico tiene que visitar su CAP.

El paciente ya no tiene que visitar su CAP para recibir los cambios en su medicación, realizar una consulta o recoger los resultados de la analítica que se realizó. Todo esto lo puede hacer desde el lugar que el desee haciendo uso de la aplicación propuesta en esta memoria.

Organizar y controlar de forma más eficiente la medicación que deben tomar para tratar su enfermedad.

Consultar la medicación ya no es algo que requiera de mantener las recetas físicamente, o llamar a tu doctor para que te la recuerde, gracias a la aplicación la puedes consultar las veces que quieras y en el momento que quieras sin necesidad de realizar nada más que entrar en la sección de medicación de la aplicación. Además, en caso de que se produzca algún cambio en dicha medicación, se notifica al usuario para que este pueda revisarla instantáneamente.

## 15.2 Desvíos en la planificación temporal

Cuando se realizó la planificación durante el módulo de GEP, se trató de hacer con la máxima precisión posible respecto a las posibilidades reales del desarrollador del sistema, tanto a nivel de conocimientos, como a nivel de tiempo. Fue por eso por lo que se escogió trabajar aproximadamente una media de seis horas diarias.

Durante el primer *sprint*, se sufrió un cierto retraso debido a la dificultad que comportaba todo el flujo del registro y el inicio de sesión, a pesar de eso, se le dedicó un número superior de horas para tratar de completar la tarea en el día estimado según la planificación. Gracias a esto, el segundo *sprint* pudo comenzar sin ningún tipo de retraso.

El segundo *sprint* fue clave en el devenir del proyecto, este contenía, además de los ajustes, todo el sistema de notificaciones y la creación de tarjetas para el muestreo de información. Gracias al buen trabajo realizado en la implementación de estos dos módulos clave del proyecto, se pudo avanzar mucho, y se consiguieron adelantar todas las tareas de visualización de datos y fueron terminadas al acabar este sprint. En consecuencia, se pudo dedicar todo el tercer *sprint* al procesado de las analíticas, realizando un análisis mucho más detallado del que se previno en un inicio.

Respecto al cuarto y al quinto *sprint,* los reajustes que se tuvieron que hacer ya están comentados en el apartado 10.1.4. En resumen, se eliminó por completo el quinto *sprint* y las tareas que iban a ser completadas en el mismo se pudieron hacer a lo largo de la cuarta iteración.

El hecho de completar el desarrollo antes de tiempo fue debido principalmente al aumento considerable de las horas diarias dedicadas al proyecto, si bien se comenzó en una media de 6 horas, hubo días que se duplicaron, y, por lo tanto, se avanzó más rápidamente. Este aumento no siempre fue debido a la aparición de errores, al tratarse de un proyecto realizado por una persona, es esta quien decide cuanto dedicarle al mismo, y en mi caso, al tratarse de algo en lo que estaba realmente interesado decidí dedicarle más de lo planificado.

## 15.3 Justificación de las competencias

CES1.1: Desarrollar mantener y evaluar sistemas y servicios software complejos y/o críticos. [En profundidad]

Este trabajo se basa principalmente en el desarrollo del sistema software propuesto a lo largo de la memoria, se le ha dedicado una gran parte del tiempo a la implementación de este, tanto de la aplicación Android como de la API REST. Además, previo al inicio de la implementación, se tuvo que diseñar toda la arquitectura que tendría que seguir el sistema, analizando las diferentes alternativas estudiadas a lo largo de la carrera y otras no tan conocidas. Todo esto se ha realizado siguiendo los criterios aprendidos sobre la ingeniería del software y el resultado se puede apreciar tanto en los productos (aplicación y API) como en los apartados 9 y 10 de esta memoria.

El sistema propuesto, se trata de un software complejo que trata de automatizar un proceso médico, realizar esto no es algo trivial, pues además de todos los conocimientos necesarios para la implementación de un sistema software que sea capaz de automatizar dicho proceso, se necesitan unas ciertas bases médicas para poder realizar toda la fase de implementación de los motores de procesado.

CES1.3: Identificar, evaluar y gestionar los riesgos potenciales asociados a la construcción de software que pudiesen presentarse. [Bastante]

Antes de comenzar el proceso de implementación, se realizó un análisis detallado de los posibles riesgos que podían llegar a tener impacto en el proyecto. Se han identificado y dividido en categorías según el tipo de riesgo que son, se han evaluado y finalmente se han realizado planes de mitigación, monitorización y gestión de dichos riesgos. El resultado de esto se puede apreciar en el apartado 5.

Además, se ha realizado una estimación del coste extra que pueden llegar a suponer estos riesgos, detallado en el apartado 7 (Gestión económica).

CES1.5: Especificar, diseñar, implementar y evaluar bases de datos. [Un poco]

Para el almacenaje de los datos, ha sido necesario contar con una base de datos relacional, la justificación de la elección de las tecnologías y el análisis de las distintas alternativas que se llegaron a barajar están incluidos en las secciones correspondientes a la base de datos dentro de la metodología (apartado 4). En cuanto a la especificación, diseño e implementación, se le ha dedicado un apartado entero donde se detalla todo el proceso que se ha ido siguiendo (apartado 11).

A pesar de ser una competencia que no se tenía que desarrollar mucho, se le ha dedicado un gran número de horas, pues tener una buena base de datos, que funcione bien y sea eficiente es fundamental para el buen funcionamiento del sistema. Por este motivo, se ha tratado de alojar dicha base de datos en un servicio potente como es AWS, con la dificultad que esto conlleva, y se ha tratado de optimizar al máximo para que cada acceso suponga el menor coste posible. En adición, se le ha dedicado un gran número de horas al diseño y especificación de las tablas, a lo largo de la implementación, se han realizado varios *refactorings* de las tablas para tratar que las devoluciones que se realizaban desde la capa de datos fueran lo más ligeras y concisas posible.

CES1.7: Controlar la calidad y diseñar pruebas en la producción de software. [Un poco]

Controlar la calidad del software desarrollado ha sido algo que se nos ha remarcado mucho a lo largo de la especialidad de software. A lo largo de la implementación tanto de la API como de la aplicación se han ido teniendo en cuenta los diferentes criterios estudiados a lo largo de las asignaturas que componen la especialidad. El resultado y el análisis de dicho control de calidad se encuentra en el apartado 12. QA & Testing.

Respecto a las pruebas, de igual forma se ha detallado el plan de pruebas puesto en práctica para la fase de *testing* del producto en el apartado 12.

Ambos aspectos han sido tratados y analizados para que el software producido cumpla con todos los criterios de calidad aprendidos y para que la puesta en producción de este no tenga ningún error conocido.

CES2.1: Definir y gestionar los requisitos de un sistema software. [Bastante]

La buena definición de los requisitos durante la fase de *Inception* ha resultado ser clave en la implementación. Cuando se realizaban las tareas, ya estaban completamente definidas y existía un requisito con el que se podían relacionar. Además, el diagrama de clases UML ayudó mucho a tener una visión genérica de como tenía que ser el sistema a nivel de clases.

En resumen, todas las características que debían tener tanto la aplicación como la API venían ya definidas de antemano gracias al buen trabajo realizado en la definición de los requisitos, el cual se puede ver en el apartado 8 de esta memoria.

## 15.4 Integración de conocimientos

A lo largo de la carrera, se han cursado diferentes asignaturas que han aportado conocimientos necesarios para la realización de este trabajo de fin de grado. Al tratarse de un proyecto centrado en la especialidad de ingeniería del software, la mayor parte del conocimiento para el desarrollo del proyecto se ha extraído de las asignaturas de esta especialidad. Si más no, también ha habido otras asignaturas que han aportado conocimiento para que este trabajo salga adelante.

### 15.4.1 PRO1, PRO2 y EDA

PRO1 fue la primera asignatura dedicada a la programación de la carrera, en ella te enseñan todo lo básico, desde los condicionales y los bucles, hasta vectores y matrices. Una vez finalizada la asignatura ya se tenía una primera visión sobre el mundo de la programación y se podían resolver pequeños problemas haciendo uso de código.

A continuación, vino PRO2, en esta asignatura se nos empezó a introducir diferentes tipos de contenedores, como son listas, mapas y demás. Además, se realizó el primer contacto con la orientación a objetos gracias al proyecto final de la asignatura, que además de poner en práctica las diferentes estructuras aprendidas a lo largo del curso, también nos introducía al concepto de objeto en C++.

Finalmente, y cerrando este bloque de asignaturas, tenemos EDA. Primeramente, se nos introdujo el concepto de coste, hasta ahora, simplemente nos dedicamos a hacer código sin importar el coste tanto temporal como espacial, pero, con los conceptos aprendidos en esta asignatura se comienza a entender lo importante que es realizar código que tenga el menor coste posible. En adición a esto, también volvimos a repasar muchas de las estructuras de datos impartidas en PRO2 y se añadieron de nuevas, como los grafos. Finalmente, se nos presentó un listado de algoritmos muy útiles para realizar búsquedas en espacios de elementos.

Gracias a estas asignaturas se ha sido capaz de llegar al nivel de programación necesario como para poder realizar este proyecto, estas consolidaron una base de programación totalmente necesaria para cualquier ingeniero informático.

### 15.4.2 BD y DBD

En BD fue donde por primera vez se comenzó a ver el lenguaje SQL, el concepto de base de datos, su diseño, y los diferentes modelos que existen. Esta asignatura nos consolida todos nuestros conocimientos relacionados con las bases de datos, y además resulta ser muy útil para aprender el lenguaje.

DBD ya formaba parte de la especialidad de Software, está más orientada al uso de distintas técnicas para optimizar bases de datos y para hacer un mejor uso de estas.

Este proyecto utiliza una base de datos para almacenar todos los datos, además de sentencias SQL para acceder a dichos datos, y, en adición se utiliza PostgreSQL, enseñado en la asignatura de BD.

### 15.4.3 IDI

IDI tiene dos partes principales, una dedicada más a la implementación de gráficos, y la segunda al diseño de interfaces, la que nos ha resultado más útil para el proyecto ha sido la segunda. IDI nos introdujo todos los conceptos básicos de usabilidad y de diseño de interfaces, gracias a esta parte de la asignatura comenzamos a comprender que un programa no es tan solo lo potente que es, si no también, lo fácil que es de saber usar.

El sistema software propuesto en esta memoria tiene una interfaz visual (la aplicación Android), para el diseño de esta se han seguido los consejos propuestos en IDI, y se ha seguido su bibliografía para poder llegar a realizar una interfaz lo más usable posible.

### 15.4.4 IES y AS

IES es la primera asignatura de la especialidad de ingeniería del software, a pesar de estar dentro del bloque de las comunes. En ella nos enseñan como diseñar sistemas software a partir de una especificación y se nos comienza a introducir el lenguaje UML.

Su continuación, AS, sigue un poco el modelo de IES, pero se nos introducen conceptos no del todo vistos en su predecesora, aquí fue cuando por primera vez se nos introducen las tres capas (modelo, vista, controlador) y se nos enseña a diseñar sistemas software que sigan esta arquitectura. Además, se explican distintos patrones realmente útiles cuando estamos diseñando e implementando software, y finalmente se nos introduce a distintas metodologías ágiles, haciendo énfasis en el TDD.

Estas asignaturas han sido clave en el desarrollo de este proyecto, tanto la arquitectura del sistema como su UML han sido realizados gracias a los conocimientos extraídos directamente de las mismas, además, se hace uso incontables veces de los patrones aprendidos en AS.

### 15.4.5 PROP

PROP es el primer gran proyecto de programación que se realiza durante la carrera, aquí se ponen en práctica todos los conceptos aprendidos a lo largo de los dos primeros años. Si más no, el cuatrimestre en el que el autor de la memoria cursó la asignatura, el proyecto estaba muy centrado en el desarrollo de un algoritmo de compresión. A pesar de eso, se consiguió poner en práctica diversos conceptos aprendidos.

La API del sistema está implementada en Java, lenguaje que se enseña por primera vez en PROP, gracias a esta puesta en práctica del lenguaje, hoy en día, el autor de la memoria tiene más soltura con este lenguaje.

### 15.4.6 ASW

ASW es la primera asignatura que nos pone en contacto con los distintos servicios web que existen, se nos enseñan los protocolos y tecnologías de estos servicios y los distintos lenguajes utilizados, además de los distintos formatos de intercambio de datos como JSON. A parte de todo esto, en esta asignatura se aprende a diseñar y a implementar de forma directa estos servicios web, es decir, no solo da una visión teórica, sino que también se pone en práctica.

Esta asignatura ha resultado ser fundamental para el desarrollo del sistema, toda la API que se propone está basada en lo aprendido en el transcurso de esta asignatura, tanto el formato REST, como el formato JSON para el intercambio de datos. Además de todo el tratamiento de los errores con sus respectivos códigos numéricos.

### 15.4.7 ER

ER nos enseña por primera vez lo importante que es la ingeniería de requisitos en el proceso de diseño de Software, se nos explica el concepto de parte interesada, y el por qué es tan importante adaptar y construir requisitos en función de estas partes. Además, se explican los casos de uso, las historias de usuario, y demás actividades propias de la ingeniería de requisitos.

Esta asignatura nos ha permitido realizar toda la parte de especificación de requisitos, sin ella no tendríamos los conocimientos necesarios para realizarla, y, como ya se ha visto, esta especificación es fundamental para el buen comienzo de la implementación.

### 15.4.8 GPS

GPS fue la primera asignatura de la carrera en mostrar en que consiste un proyecto de Software, la gestión de los riesgos, el contexto de este, y las distintas actividades que tiene la construcción de un sistema de este tipo.

A continuación, el curso se dividió en dos partes. La primera consistía en aprender cómo se gestionaban los proyectos de forma clásica, y la segunda en lo mismo, pero con proyectos gestionados de forma ágil.

Esta asignatura nos da una visión de las posibles metodologías existentes para completar un proyecto software, la elección de la metodología del sistema que se propone ha sido fundamentalmente inspirada en los conceptos aprendidos en esta asignatura. Además, toda la parte de gestión de riesgos ha sido extraída de la bibliografía de la asignatura.

### 15.4.9 PES

PES se podría resumir como la culminación de la especialidad de software, aquí se pone en práctica todo lo aprendido durante la especialidad para realizar un proyecto de software completo, desde la parte de incepción del proyecto, hasta la fase final de desarrollo y presentación del producto.

En esta asignatura se pone en práctica todo lo aprendido, desde los requisitos hasta la puesta en funcionamiento de la metodología Scrum aprendida en GPS.

La aplicación desarrollada está realizada en Kotlin, lenguaje que vi por primera vez gracias a la implementación de la aplicación móvil que se realizó en PES. Además, me ha dado la visión general, y la capacidad de realizar un proyecto software por mi cuenta, desde la parte de la documentación, hasta la implementación.

### 15.4.10 PAE

PAE es una de las optativas del GEI, consiste en la propuesta de un proyecto por parte de una empresa y la elaboración de este.

A lo largo del curso, se nos enseña a como llevar un proyecto desde un punto de vista más enfocado al cliente, es decir, nos enseñan a como vender el producto realizando presentaciones que realmente impresionen al objetivo final del sistema.

Esta asignatura resultará especialmente útil a la hora de la presentación de la memoria, pues ya se realizó algo parecido con la presentación final del proyecto realizado en esta.

## 15.5 Conclusión y reflexión final

Hoy en día, vivimos en un entorno de revolución tecnológica constante, en unos pocos años hemos pasado de llevar teléfonos con tapa y teclado, a *smartphones* plegables con varias pantallas. Sin embargo, ¿Qué pasa con la sanidad? Si bien ha ido evolucionando poco a poco, el crecimiento tecnológico que ha experimentado ha sido ínfimo si lo comparamos con otros sectores. Excusar esto en que se trata de un sector difícil de informatizar, es, bajo mi punto de vista, ponernos una venda en sobre los ojos.

Actualmente existen un gran número de enfermedades crónicas cuyo análisis se puede automatizar, como ya se ha visto en esta memoria, este proceso requiere tiempo, pero el que ahorra es mucho mayor. Es por eso por lo que trabajar en la inclusión de este sistema en el sistema sanitario sería clave para liberar tiempo a los sanitarios.

Con este proyecto se ha pretendido demostrar que es posible automatizar un proceso médico, ya no hablamos de pedir cita con el móvil, sino de evitarnos una visita médica gracias a un sistema informático. Las enfermedades que se han parametrizado (hipercolesterolemia e hipotiroidismo) son dos de las enfermedades crónicas más comunes, y su automatización ha sido realizada en un trabajo de fin de grado por un estudiante en su último año (sin ánimos de quitarme méritos). Si bien la inclusión de este sistema necesitaría pasar muchos procesos de verificación, se ha podido ver que es funcional y que podría ponerse en producción.

Hablando ahora de mis sensaciones durante el transcurso del proyecto, el hecho de involucrarme directamente con personal médico y exponerles mi proyecto para recibir sus más sinceras opiniones ha resultado ser algo muy fructífero para mí, además de poner en práctica todo lo aprendido durante la carrera, he aprendido como funciona una parte de este sector tan importante en nuestro día a día, y quien sabe, quizás he conseguido aportar mi granito de arena en este sector. Sin embargo, el hecho de convertir la automatización de una enfermedad en una realidad ha resultado ser algo muy complicado, pues evidentemente soy estudiante de ingeniería informática y no de medicina, y con la situación actual el contacto con los médicos cada vez se volvía más y más complicado. A pesar de eso, y, sobre todo, gracias a la doctora Natividad Pacheco, se pudo sacar el proyecto adelante dándole el realismo que yo quería.

Finalmente, me gustaría añadir que la realización de este trabajo a resultado en una gran satisfacción personal. Se pudo sacar adelante toda la faena inicialmente prevista, se puso en práctica todo lo aprendido durante la carrera, y se aprendieron cosas nuevas. Además, la realización de este proyecto me ha acabado de ayudar a darme cuenta de que me quiero centrar mi carrera profesional en entornos científicos y que realmente tengan impacto en las vidas de las personas.

# Bibliografía


[1] EsCrónicos, «Las cifras,» 2017. [En línea]. Available: http://www.escronicos.com/es/las-cifras. [Último acceso: 15 02 2021].

[2] SEQC, «El proceso de análisis de una muestra de sangre,» 25 09 2016. [En línea]. Available: https://labtestsonline.es/news/el-proceso-de-analisis-de-una-muestra-de-sangre. [Último acceso: 30 01 2021].

[3] Generalitat de Catalunya, «Manejo de la medicación en el paciente crónico: conciliación, revisión, desprescripción y adherencia.,» 10/2014.

[4] Sysmex, «Ejemplo de analizador automatizado,» [En línea]. Available: https://www.sysmex.com/US/en/Products/Hematology/Pages/Sysmex-Hematology-Overview.aspx. [Último acceso: 20 02 2021].

[5] G. d. Catalunya, «La meva salut,» [En línea]. Available: https://lamevasalut.gencat.cat/es. [Último acceso: 20 02 2021].

[6] N. Pastor, «HumanITcare,» [En línea]. Available: https://humanitcare.com/. [Último acceso: 20 02 2021].

[7] Everis, «ehCOS,» [En línea]. Available: https://www.ehcos.com/. [Último acceso: 20 02 2021].

[8] J. Rasmusson, «Introducing Agile,» de *The Agile Samurai*, 2010, pp. 16-19.

[9] J. Rasmusson, «Agile Project planning,» de *The Agile Samuray*, 2010, pp. 94-98.

[10] J. Rasmusson, «Creating an Agile Communication Plan,» de *The Agile Samurai*, 2010, pp. 179-182.

[11] G. Inc, «Github,» [En línea]. Available: https://github.com/. [Último acceso: 27 02 2021].

[12] Atlassian, «Git merge,» [En línea]. Available: https://www.atlassian.com/es/git/tutorials/using-branches/git-merge. [Último acceso: 27 02 2021].

[13] Atlassian, «Jira,» [En línea]. Available: https://www.atlassian.com/es/software/jira. [Último acceso: 27 02 2021].

[14] Jetbrains, «IntelliJ IDEA,» [En línea]. Available: https://www.jetbrains.com/es-es/idea/. [Último acceso: 27 02 2021].

[15] G. B. Tool, «Gradle,» [En línea]. Available: https://gradle.org/. [Último acceso: 27 02 2021].

[16] Google, «Android Studio,» [En línea]. Available: https://developer.android.com/studio. [Último acceso: 27 02 2021].

[17] PostgreSQL, «PostgreSQL,» [En línea]. Available: https://www.postgresql.org/. [Último acceso: 27 02 2021].

[18] Deloitte, «Kanvan vs. Scrum,» [En línea]. Available: https://www2.deloitte.com/es/es/pages/technology/articles/kanban-vs-scrum.html. [Último acceso: 16 04 2021].

[19] Google, «Android developers,» [En línea]. Available: https://developer.android.com/studio. [Último acceso: 18 04 2021].

[20] OpenWebinars, «Kotlin vs Java,» [En línea]. Available: https://openwebinars.net/blog/kotlin-vs-java/. [Último acceso: 16 04 2021].

[21] Pandorafms, «NoSQL vs SQL,» [En línea]. Available: https://pandorafms.com/blog/es/nosql-vs-sql-diferencias-y-cuando-elegir-cada-una/. [Último acceso: 16 04 2021].

[22] R. Pressman y B. M. H. Maxim, «Risk analysis and managment,» de *Software engineering: a practitioner's approach*, pp. 145-164.

[23] A. Rabadan, «Gestión de riesgos de proyectos software,» *Wikiversidad,* 2014.

[24] Accenture, «Accenture,» [En línea]. Available: https://www.accenture.com/es-es?c=acn_glb_brandexpressiongoogle_11608784&n=psgs_1020&gclid=Cj0KCQjw3duCBhCAARIsAJeFyPWmYrifVPzRg4HnMtrjsSQ0PvlZ8egpB9iAb1fMzdGHTt6y-J-To5waAv34EALw_wcB. [Último acceso: 15 02 2021].

[25] Amazon, «Amazon Web Services,» [En línea]. Available: https://aws.amazon.com/es/. [Último acceso: 15 02 2021].

[26] Glassdoor, «Sueldos,» [En línea]. Available: https://www.glassdoor.es/index.htm. [Último acceso: 10 03 2021].

[27] DELL, «OptiPlex 3080 MFF,» [En línea]. Available: https://www.dell.com/es-es/work/shop/sobremesas-y-estaciones-de-trabajo/optiplex-3080-mff/spd/optiplex-3080-micro/s012o3080mff. [Último acceso: 12 03 2021].

[28] DELL, «Monitor Dell 27: E2720HS,» [En línea]. Available: https://www.dell.com/es-es/shop/monitor-dell-27-e2720hs/apd/210-aurh/monitores-y-accesorios. [Último acceso: 12 03 2021].

[29] J. y. S. Robertson, «Requirements Specification Template,» 2012.

[30] U. I. d. Valencia, «Conoce las especialidades dee enfermería,» [En línea]. Available: https://www.universidadviu.com/es/actualidad/nuestros-expertos/conoce-las-especialidades-de-enfermeria. [Último acceso: 05 04 2021].

[31] B. Schneiderman, de *Designing the user interface: strategies for effective human-computer interaction*, 2000, pp. 110-118.

[32] Google, «Guía de arquitectura de apps,» [En línea]. Available: https://developer.android.com/jetpack/guide. [Último acceso: 10 04 2021].

[33] E. Gamma, R. Helm, R. Johnson y J. A.-W. Vlissides, Design patterns: elements of reusable object-oriented software, 1995.

[34] MedlinePlus, «Statins,» [En línea]. Available: https://medlineplus.gov/statins.html. [Último acceso: 28 04 2021].

[35] A. developers, «Kotlin,» [En línea]. Available: https://developer.android.com/kotlin?hl=es. [Último acceso: 26 04 2021].

[36] «Dagger,» [En línea]. Available: https://dagger.dev/. [Último acceso: 26 04 2021].

[37] Google, «Lifecycle,» [En línea]. Available: https://developer.android.com/jetpack/androidx/releases/lifecycle. [Último acceso: 26 04 2021].

[38] Google, «Corrutinas de Kotlin en Android,» [En línea]. Available: https://developer.android.com/kotlin/coroutines?hl=es-419. [Último acceso: 2021 04 26].

[39] Github, «Retrofit,» [En línea]. Available: https://square.github.io/retrofit/. [Último acceso: 27 04 2021].

[40] Google, «GSON,» [En línea]. Available: https://github.com/google/gson. [Último acceso: 2021 04 26].

[41] M. Penz, «Material Drawer,» [En línea]. Available: https://github.com/mikepenz/MaterialDrawer. [Último acceso: 26 04 2021].

[42] Google, «Components,» [En línea]. Available: https://material.io/components?platform=android. [Último acceso: 2021 04 26].

[43] M. Penz, «FastAdapter,» [En línea]. Available: https://github.com/mikepenz/FastAdapter. [Último acceso: 26 04 2021].

[44] Afollestad, «Material Dialogs,» [En línea]. Available: https://github.com/afollestad/material-dialogs. [Último acceso: 26 04 2021].

[45] Google, «Firebase Cloud Messaging,» [En línea]. Available: https://firebase.google.com/docs/cloud-messaging?hl=es-419. [Último acceso: 26 04 2021].

[46] Google, «Cómo navegar a un destino,» [En línea]. Available: https://developer.android.com/guide/navigation/navigation-navigate?hl=es. [Último acceso: 26 04 2021].

[47] Spring, «Spring Boot,» [En línea]. Available: https://spring.io/projects/spring-boot. [Último acceso: 2021 04 26].

[48] Spring, «Spring Boot,» [En línea]. Available: https://spring.io/projects/spring-boot. [Último acceso: 02 05 2021].

[49] P. Lombok, «Lombok,» [En línea]. Available: https://projectlombok.org/. [Último acceso: 26 04 2021].

[50] FasterXML, «Jackson Project,» [En línea]. Available: https://github.com/FasterXML/jackson. [Último acceso: 27 04 2021].

[51] Apache, «Log4j 2,» [En línea]. Available: https://logging.apache.org/log4j/2.x/. [Último acceso: 26 04 2021].

[52] «Introduction to JSON Web Tokens,» [En línea]. Available: https://jwt.io/introduction. [Último acceso: 26 04 2021].

[53] O. Blancarte, «Java Persistance API (JPA),» [En línea]. Available: https://www.oscarblancarteblog.com/tutoriales/java-persistence-api-jpa/. [Último acceso: 02 05 2021].

[54] dbadixit, «Modelo ACID de transacciones,» [En línea]. Available: http://dbadixit.com/modelo-acid-transacciones/. [Último acceso: 02 05 2021].

[55] SonarQuber, «Code Quality and Code Security,» [En línea]. Available: https://www.sonarqube.org/. [Último acceso: 03 05 2021].

[56] EDINSOST, «Cuestionario de Estudiantes de Ingeniería Informática.,» [En línea]. Available: https://docs.google.com/forms/d/e/1FAIpQLSelZixKIUFbCn1oVkd2JM3yxCc208E85RgKZclKd8eUu3GvBg/viewform. [Último acceso: 15 03 2021].

[57] G. d. España, «Agencia Estatal Boletín Oficial del Estado,» [En línea]. Available: https://www.boe.es/buscar/doc.php?id=BOE-A-2002-22188. [Último acceso: 17 04 2021].

[58] G. d. España, «Agencia Estatal Boletín Oficial del Estado,» [En línea]. Available: https://www.boe.es/buscar/act.php?id=BOE-A-2006-21990. [Último acceso: 25 04 2021].

[59] G. d. España, «Agencia Estatal Boletín Oficial del Estado,» [En línea]. Available: https://www.boe.es/buscar/doc.php?id=BOE-A-2018-16673. [Último acceso: 14 04 2021].

[60] G. d. España, «Agencia Estatal Boletín Oficial del Estado,» [En línea]. Available: https://www.boe.es/buscar/act.php?id=BOE-A-1986-10499. [Último acceso: 16 04 2021].

# Anexos

## A. Encuesta para la verificación de requisitos no funcionales a pacientes

1. ¿Consideras la aplicación estéticamente agradable?

a) Sí
b) No
c) Tal vez

2. ¿Has sido capaz de acceder a las cuatro secciones principales de la aplicación (Análisis, medicación, calendario y chat) de forma rápida?

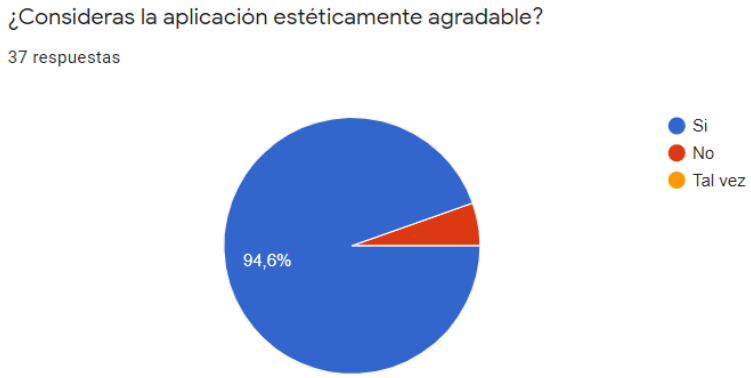
a) Sí
b) No

3. Consideras que, tras haber utilizado el sistema aproximadamente diez horas, eres capaz de utilizar todas las funcionalidades ofrecidas

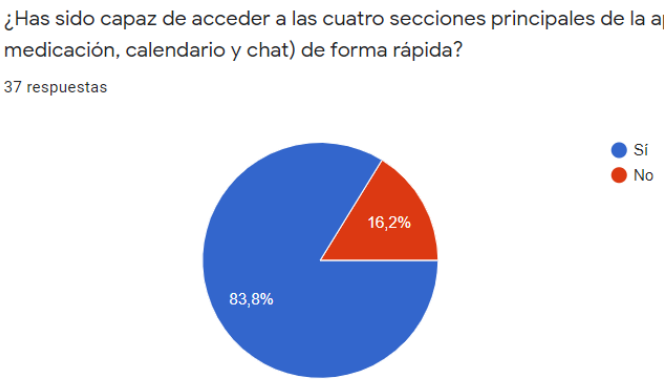
a) Sí
b) No
c) Las principales si, pero tengo la sensación de que no le estoy sacando el máximo partido a la aplicación

## B. Resultados de la encuesta a pacientes

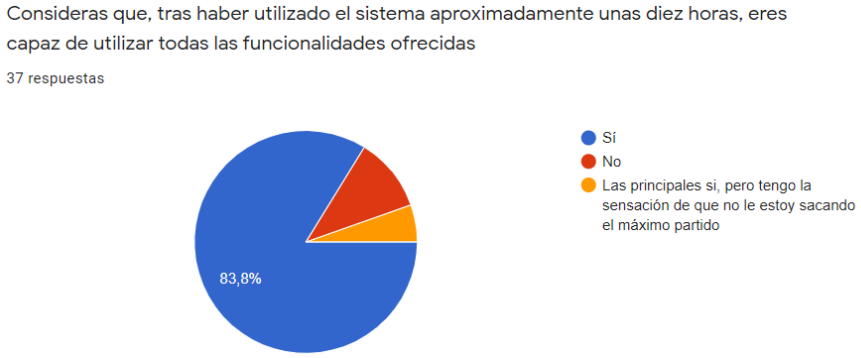
1.



2.




3.




## C. Encuesta para la verificación de requisitos no funcionales al personal sanitario

1. ¿Qué rol desempeñas dentro del ámbito sanitario?

a) Enfermero/a
b) Doctor/a
c) Auxiliar de enfermería
d) Documentalista sanitario
e) Otros

2. ¿Tienes la sensación de estar tratando con un sistema serio, y que inspira confianza a ser utilizado en sustitución al proceso tradicional?

a) Sí
b) No
c) Tal vez

3. ¿Has gestionado alguna vez los cambios en la medicación de un paciente crónico?

a) Sí
b) No

4. En caso de haber respondido que sí, marca si has tratado alguna de las siguientes

a) Hipercolesterolemia
b) Hipotiroidismo

5. En caso de haber tratado alguna de las anteriores, indica si, después de probar la aplicación, los cambios que ha generado el sistema son los mismos que hubieses generado tú en una consulta real.

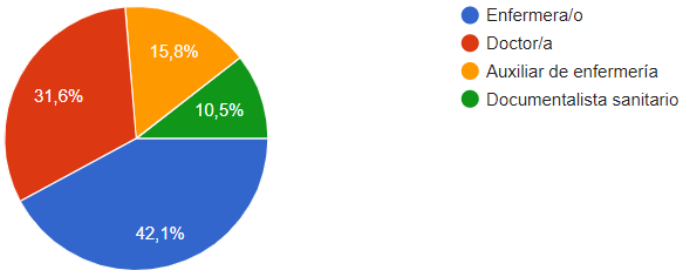
a) Sí
b) No
c) No siempre

## D. Resultados de la encuesta al personal sanitario

1.

¿Qué rol desempeñas dentro del ámbito sanitario?

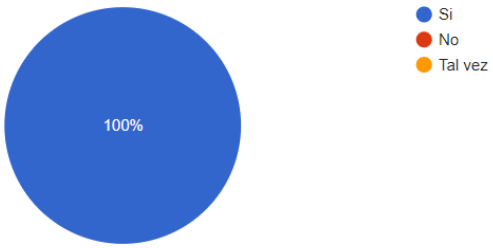
19 respuestas




2.

¿Tienes la sensación de estar tratando con un sistema serio, y que inspira confianza a ser utilizado en sustitución al proceso tradicional?

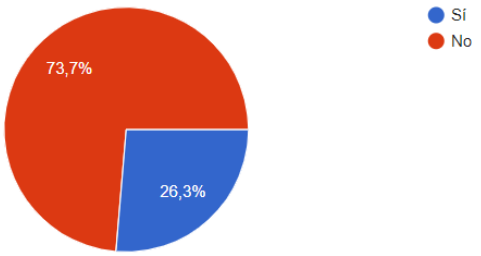
19 respuestas



3.

¿Has gestionado alguna vez los cambios en la medicación de un paciente crónico?

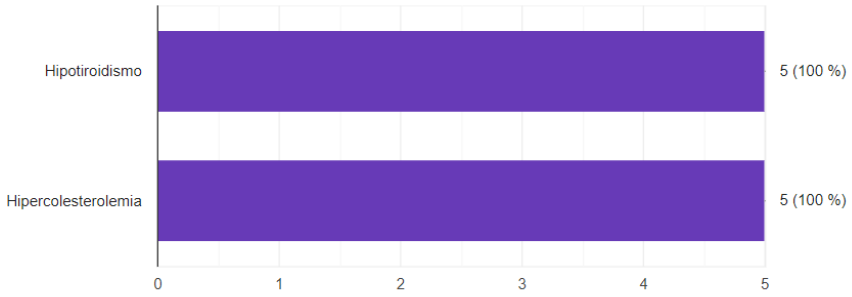
19 respuestas




4.

En caso de haber respondido que si, marca si has tratado alguna de las siguientes

5 respuestas




5.

En caso de haber tratado alguna de las anteriores, indica si, después de probar la aplicación, los cambios que ha generado el sistema son los mismos que hubieses generado tu en una consulta real.

5 respuestas