YATA HTA: *app* para el seguimiento y control de población hipertensa y su medicación

YATA HTA: app for the monitoring and control of the hypertensive population and its medication

AUTORES:
Hermoso Cara, Fernando
Pallás Gozálvez, Ignacio
Plaza Campillo, Juan José

Grado en Ingeniería Informática
Facultad de Informática
UNIVERSIDAD COMPLUTENSE DE MADRID

TRABAJO DE FIN DE GRADO EN INGENIERÍA INFORMÁTICA
Curso académico 2021-2022

Director:
Pablo Rabanal Basalo
Co-director:
Alejandro Rabanal Basalo

Autorización de difusión y utilización

Los abajo firmantes, alumno/s y tutor/es del Trabajo Fin de Grado (TFG) en el Grado en Ingeniería Informática de la Facultad de Informática, autorizan a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el Trabajo Fin de Grado (TFG) cuyos datos se detallan a continuación. Así mismo autorizan a la Universidad Complutense de Madrid a que sea depositado en acceso abierto en el repositorio institucional con el objeto de incrementar la difusión, uso e impacto del TFG en Internet y garantizar su preservación y acceso a largo plazo.

TÍTULO DEL TFG: Desarrollo de una app para el seguimiento y control de población hipertensa.
Curso académico: 2021 / 2022

Nombre del Alumno/s:

- Fernando Hermoso Cara
- Ignacio Pallás Gozálvez
- Juan José Plaza Campillo

Tutor/es del TFG y departamento al que pertenece:

- Pablo Rabanal Basalo, Departamento de Sistemas Informáticos y Computación.
- Alejandro Rabanal Basalo, médico de Atención Primaria del centro de Salud Los Yébenes

Firma del alumno/s

Fernando Hermoso Cara

Fecha:

Juan José Plaza Campillo

Fecha:

Ignacio Pallás Gozálvez

Fecha:

Firma del tutor/es

Pablo Rabanal Basalo

Fecha:

Alejandro Rabanal Basalo

Fecha:

A nuestros profesores de Grado, por todas sus enseñanzas.

Agradecimientos

Quiero agradecer a todos los que confiaron en mí y el apoyo que me han dado en esta etapa.

Por otra parte quiero agradecer especialmente a Juanjo y a Nacho por todos los años compartidos, ha sido una suerte teneros como compañeros.

Fernando Hermoso Cara

A mi mujer y a mi hija, por su comprensión, paciencia y sus ánimos constantes, sin su apoyo no hubiera podido seguir adelante.

A mi familia y amigos por su apoyo incondicional y estar siempre ahí.

A mis compañeros de la facultad, en especial a Fernando y Nacho, mis acompañantes en esta “aventura” que ha sido el Trabajo Fin de Grado, sin su amistad y ayuda durante todos estos años no habría disfrutado ni aprendido tanto como lo he hecho.

Juan José Plaza Campillo

A mis padres y hermanos, por su apoyo incondicional y por siempre creer en mí, incluso cuando ni yo mismo lo hacía, sin importar lo difícil que se pusiera todo. Sin ellos esto no habría sido posible.

A mis amigos de toda la vida por todo su amor y apoyo, y a los que he tenido la suerte de hacer desde que empecé este viaje. Especialmente a Fer y Juanjo, con los que quiero seguir disfrutando y de los que espero seguir aprendiendo.

A Pablo y al equipo de médicos por su profesionalidad y cercanía en este proyecto.

Ignacio Pallás Gozálvez

Índice

Resumen

Hemos desarrollado una aplicación móvil para el seguimiento y control de población hipertensa. La aplicación es capaz de registrar, para cada usuario dado de alta, la medicación que este toma, generando recordatorios para que el paciente no olvide tomar su tratamiento. Además, es posible registrar la tensión arterial indicando la periodicidad necesaria según cada paciente. Para generar adherencia al tratamiento, se ha confeccionado un programa incentivador mediante puntos en el que el usuario irá ascendiendo puestos en una clasificación general.

Por otro lado, se ha realizado una aplicación web para que los médicos puedan tener un control de sus pacientes y así poder llevar a cabo un mejor diagnóstico y seguimiento. En esta aplicación web se permite acceder a sus tomas de tensión más recientes, medicación registrada y contactar con el paciente en cuestión mediante un chat.

Palabras clave

- Hipertensión arterial.
- Adherencia al tratamiento médico.
- Aplicación web.
- Aplicación móvil.
- Android.
- Django.

Abstract

We have developed a mobile application for the monitoring and control of the hypertensive population. It is able to record, for every registered user, the medication taken, generating reminders so that the patients do not forget to take their treatments. Furthermore, it is possible to record blood pressure indicating the necessary periodicity according to each patient. In order to generate adherence to treatment, we created an incentive programme of points in which the user will move up positions in a general classification.

On the other hand, a web application has been created so that doctors can keep track of their patients and, thus, they are able to carry out a better diagnosis and follow-up. This web application allows them to access their most recent blood pressure readings, recorded medication, and to contact via chat with any associated patient.

Keywords

- Arterial hypertension.
- Adherence to medical treatment.
- Web application.
- Mobile application.
- Android.
- Django.

Capítulo 1: *Introducción*

En este primer capítulo hacemos una pequeña introducción al problema de salud que supone la hipertensión arterial en España a la vez que repasamos su datos epidemiológicos más destacados. Esto nos da pie a introducir el porqué del proyecto “Yébenes Adherencia al Tratamiento Antihipertensivo” planteado por los médicos de atención primaria del Centro de Salud “Los Yébenes”.

En segundo lugar reproducimos los objetivos planteados en el citado proyecto, así como nuestros propios objetivos con el fin de que el proyecto cumpla con los suyos.

Finalmente, explicamos cómo nos hemos organizado como equipo y las metodologías de trabajo seguidas.

1.1 Antecedentes

La hipertensión arterial (HTA) supone hoy en día una de las patologías más prevalentes en la sociedad occidental, en España está presente en el 33% de la población (66% en mayores de 60 años) y se le atribuyen 40.000 muertes anuales [1]. Se caracteriza por su escasa sintomatología directa y su asociación con múltiples patologías de alto riesgo como la cardiopatía isquémica, los accidentes cardiovasculares o la insuficiencia renal.

Esta patología está muy influenciada por el estilo de vida, la alimentación y los hábitos. No obstante, aunque se siga un estilo de vida sano y, actualmente, dispongamos de un amplio arsenal terapéutico, las cifras tensionales pueden seguir siendo muy elevadas. Mediante estudios poblaciones se cree que solo el 26,6% de la población hipertensa mantiene las cifras tensionales indicadas como buen control según las guías clínicas.

El mal control de la HTA se explica por múltiples factores, entre ellos destaca el que se trata de un proceso indolente, con un importante número de pacientes hipertensos que desconocen que padecen esta patología. Por otro lado, se estima que únicamente una parte de éstos mantiene una correcta pauta y adherencia al tratamiento.

Desde el servicio de atención primaria del Centro de Salud “Los Yébenes”, que trataremos en la memoria como nuestros clientes, proponen el proyecto “Yébenes Adherencia al Tratamiento Antihipertensivo - YATA”. Con este proyecto pretenden abordar ambos problemas con la intención de obtener una mayor adherencia al tratamiento, un mayor control sobre las cifras tensionales y captar a pacientes hipertensos no diagnosticados previamente.

Hoy en día el uso de los dispositivos inteligentes, como nuestros teléfonos móviles, es prácticamente universal en nuestro entorno. Son ligeros, los llevamos siempre encima, y con unas pocas interacciones podemos completar tareas de diversa índole. Por ello, nuestros clientes se plantearon el desarrollo de una aplicación que les ayudase en la consecución de sus objetivos.

Con el desarrollo de nuestra aplicación pretendemos, por una parte, influir en las conductas de estos pacientes hipertensos de modo que mejoren su adherencia al tratamiento y, de esta forma, mejoren su tensión arterial. Y, por otra parte, facilitar a los especialistas sanitarios el desarrollo de su labor preventiva e investigadora.

Adicionalmente, planeamos publicar la aplicación en la tienda de aplicaciones de Android, *Google Play* [2], de modo que cualquier usuario pueda emplearla para el seguimiento individualizado de su tensión arterial y tomas de medicación.

1.1.1 Trabajos relacionados

Antes de comenzar con el diseño y desarrollo, todo el equipo investigó otras aplicaciones parecidas. A partir de ellas se pudo diseñar una interfaz intuitiva para el usuario. También se propuso como idea la implementación de un chat para que los médicos y los pacientes tuvieran una comunicación fluida. Las aplicaciones tomadas como referencia son las siguientes:

- Blood Pressure Log - MyDiary [3]
  Blood Pressure Log - MyDiary permite el registro de presión arterial con recordatorios, ofrece diversas estadísticas y capacidad para exportar datos en diversos formatos.
  Algunas capturas de pantalla son las siguientes:

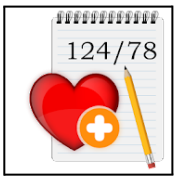
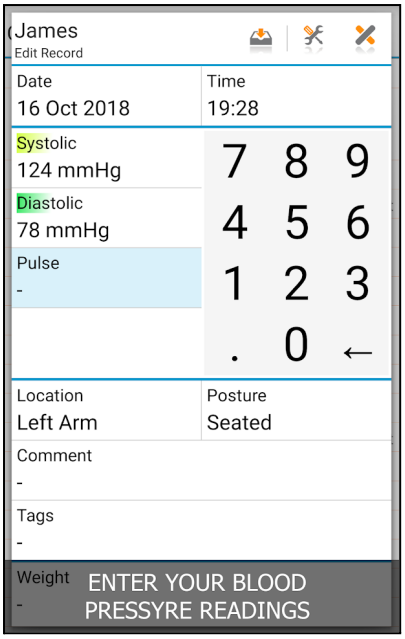
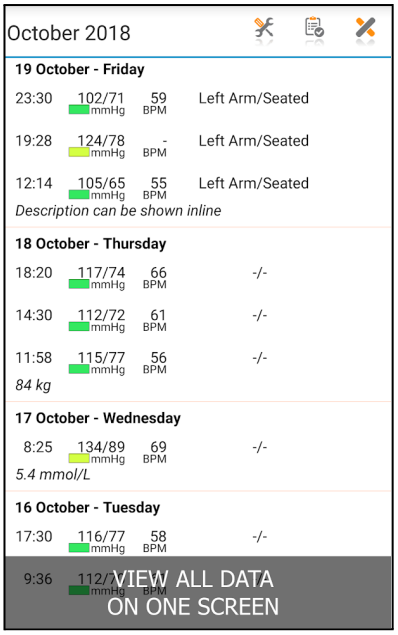


Imagen 1: Logo de la aplicación *Blood Pressure Log - MyDiary.*





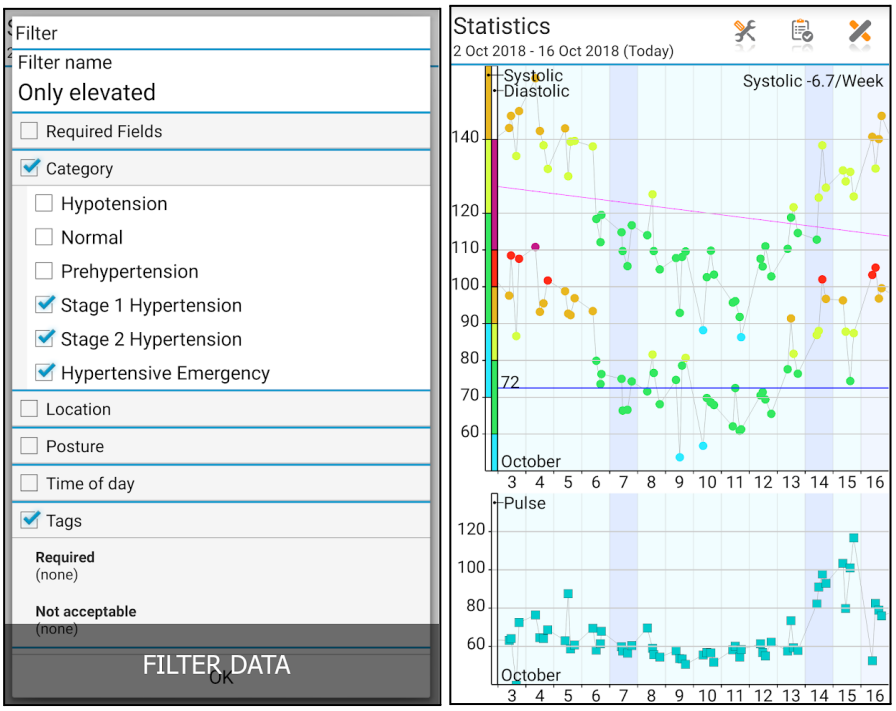


Imagen 2: Capturas de pantalla de la aplicación *Blood Pressure Log - MyDiary.*

- Qardio Heart Health [4]
  Qardio Heart Health permite el seguimiento de la presión arterial y el peso. Ofrece la posibilidad de transferir datos desde otras aplicaciones como MyFitnessPal [5], Google Fit [6], Samsung Health [7]. Ofrece también ver gráficos, establecer recordatorios y compartir datos.
  Algunas capturas de pantalla son las siguientes:

Imagen 3: Logo de la aplicación *Qardio Heart Health.*

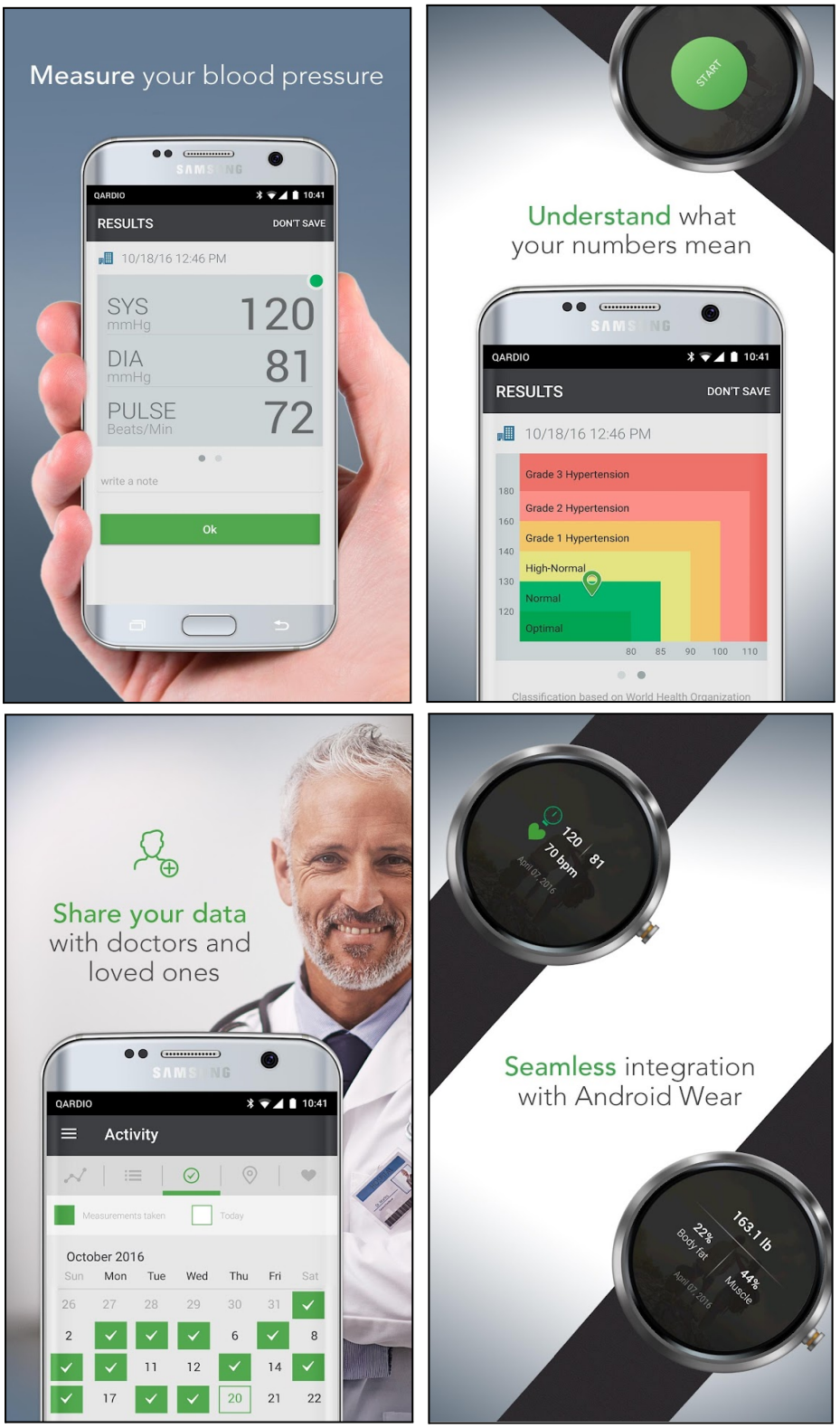
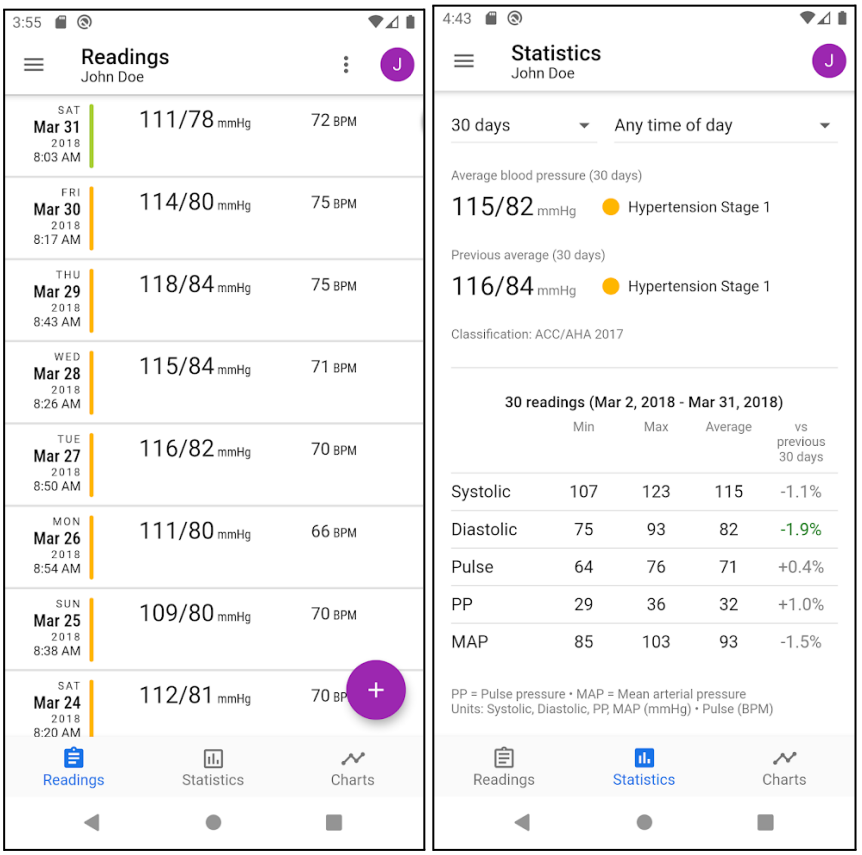


Imagen 4: Capturas de pantalla de la aplicación *Qardio Heart Health*.

- BP Journal: Blood Pressure Log [8]
  BP Journal permite el registro de presión arterial y pulso, genera estadísticas y gráficos con los que se podrán exportar o importar esos datos en formato CSV con otras aplicaciones y enviar informes a su médico.
  Algunas capturas de pantalla son las siguientes:

Imagen 5: Logo de la aplicación *BP Journal: Blood Pressure Log*.



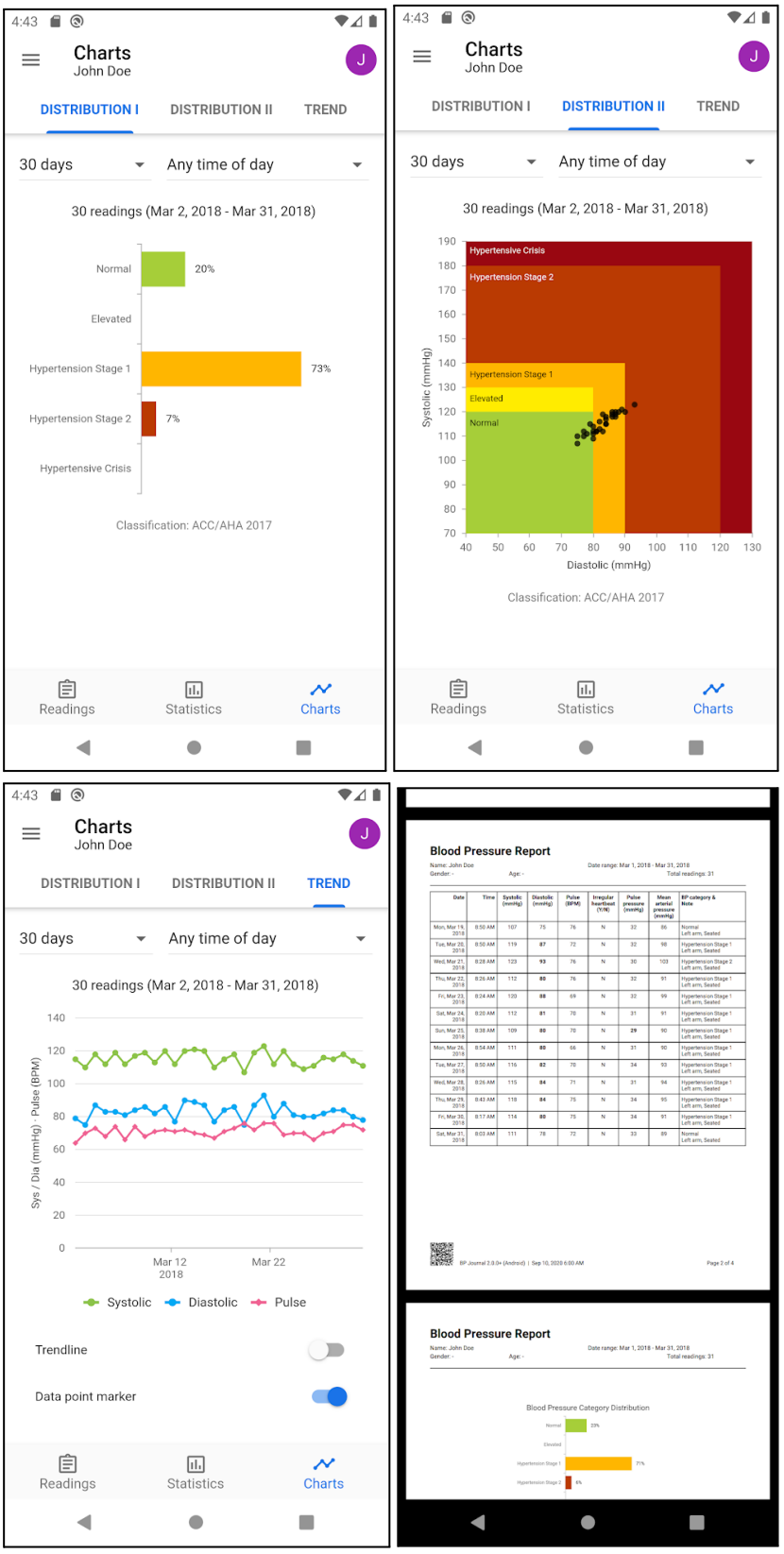


Imagen 6: Capturas de pantalla de la aplicación *BP Journal: Blood Pressure Log.*

- Registro de presión arterial (Health & Fitness AI Lab) [9]
  Al igual que las anteriores permite registrar tomas de tensión arterial incluyendo etiquetas a la toma, que luego puedan facilitar su búsqueda. Aporta filtros de fecha, etiquetas y zonas de presión sanguínea para las búsquedas. Permite exportar datos en formato CSV.
  Algunas capturas de pantalla son las siguientes:

Imagen 7: Logo de la aplicación *Registro de presión arterial (Health & Fitness AI Lab)*.

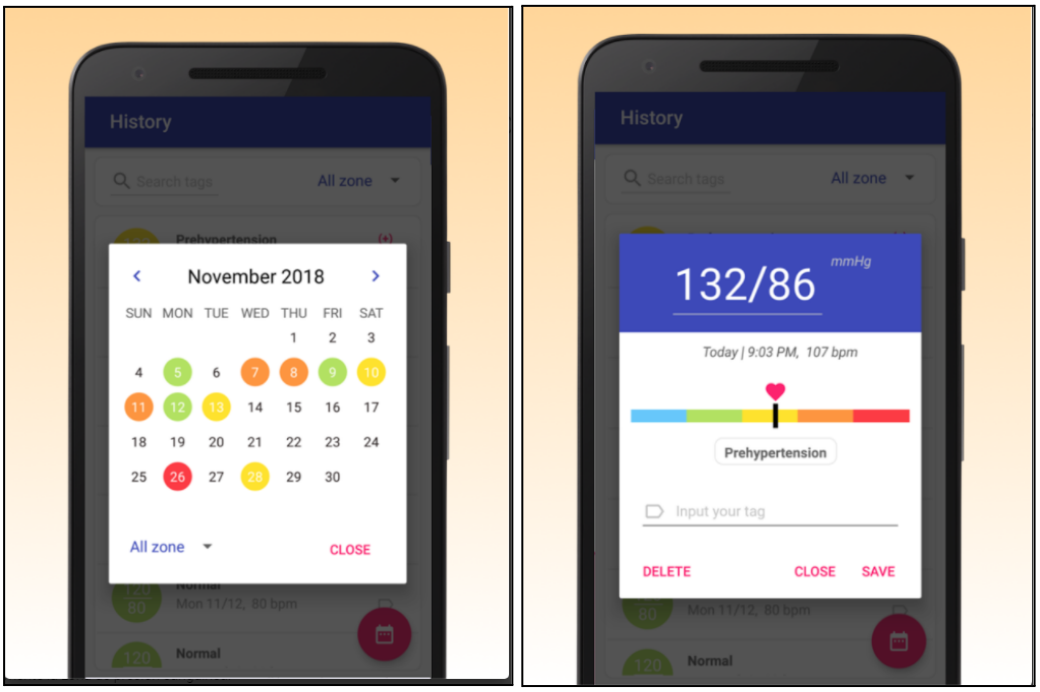


Imagen 8: Capturas de pantalla de la aplicación *BP Journal: Blood Pressure Log.*

Aunque hay más aplicaciones similares en la tienda de aplicaciones de Google, hemos destacado estas fundamentalmente por la similitud en cuanto a características, por el hecho de ser gratuitas y por tener una interfaz más cuidada.

1.2 Objetivos

Por parte del cliente plantean para su proyecto los siguientes objetivos:

1. Obtener una mayor adherencia al tratamiento.
2. Lograr un mayor control de las cifras tensionales.
3. Involucrar a los enfermos de hipertensión en su enfermedad y, en la medida de lo posible, empoderar a sus pacientes y otorgarles herramientas para aumentar su autonomía.
4. Aumentar la captación de pacientes hipertensos.

Por nuestra parte, para conseguir que nuestro cliente cumpla con sus objetivos, nos planteamos los siguientes:

1. Desarrollar una aplicación móvil para los pacientes del Centro de Salud “Los Yébenes”, que permita:
    a. Registrar medidas de tensión.
    b. Registrar la toma de medicación.
    c. Mostrar recordatorios para la toma de medicación.
    d. Realizar el test de Batalla [10].
    e. Realizar el test de Morisky-Green [11].
    f. Motivar a los usuarios mediante un sistema de competición por puntos.
    g. Formar a los usuarios en conceptos relacionados con la hipertensión arterial.
    h. Mandar mensajes privados con dudas a su médico asignado del centro de salud.
2. Desarrollar una aplicación web para los médicos y personal sanitario del Centro de Salud “Los Yébenes”, que permita:
    a. Visualizar la información registrada por sus pacientes para realizar su seguimiento.
    b. Obtener estadísticas de adherencia al tratamiento por parte de sus pacientes.
    c. Poder comunicarse con sus pacientes mediante un sistema de mensajes interno.

1.3 Plan de trabajo

El propio proyecto para el trabajo de final de grado exigía el desarrollo de una aplicación móvil, por lo que este requisito quedaba bastante claro. Sin embargo, el desarrollo de una aplicación móvil suponía un desafío para nuestro equipo de trabajo puesto que dos integrantes no habían programado nunca para plataformas móviles y, el otro tenía la experiencia de haber cursado la asignatura de "*Programación de aplicaciones para dispositivos móviles*".

Por otro lado, una funcionalidad importante es la del control y visión global que debe poder obtener nuestro cliente. De modo que pueda evaluar en qué medida se cumplen los objetivos de su proyecto. Para este fin concluimos que lo ideal sería el desarrollo de una aplicación web, que sería accesible desde cualquier dispositivo con conexión a internet, entre otros, sus propios ordenadores del centro de salud.

En cuanto a las tecnologías a emplear entraremos más en detalle del porqué de nuestras decisiones en el *Capítulo 4: Discusión*.

Una vez seleccionadas las tecnologías de base para el desarrollo de las aplicaciones, ahora tocaba tomar decisiones relacionadas con la organización del propio equipo de trabajo que formamos los tres compañeros.

1.3.1 ¿Cómo nos hemos organizado?

User Story Mapping

Para comenzar con el planteamiento de la aplicación empleamos la técnica de "*User Story Mapping*" [12], que sigue una idea bastante sencilla de entender. Habla sobre el viaje del usuario a través de la aplicación, construyendo un modelo simple que cuente la historia del usuario a medida que utiliza la aplicación. Esto permite trabajar con historias de usuarios en metodologías de desarrollo ágil.

Las historias de usuario se distribuyen como vemos en la siguiente imagen:

Imagen 9: *User Story Mapping* con las historias de usuario.

La primera fila corresponde a las "*User Activities*", son actividades de "grano grueso" que va a hacer el usuario con nuestra aplicación. Cada una de estas actividades da lugar a una columna.

La segunda fila corresponde a las "*User Tasks*", son actividades de "grano medio", algo más específicas que las anteriores.

Las siguientes filas contienen ya las verdaderas historias de usuario, repartidas por "*releases*" o diferentes versiones de lanzamiento de la aplicación. La división en *releases* es una estimación de lo que seremos capaces de alcanzar en nuestro proyecto.

La primera *release* sería una aplicación con las funcionalidades mínimas para ser de interés a su público objetivo.

Scrum

En este sentido hemos tratado de seguir una metodología ágil, en concreto *Scrum* [13], si bien esto no ha sido del todo fácil fundamentalmente porque requiere de una dedicación casi exclusiva al proyecto, cosa que ha sido imposible al compatibilizar el proyecto con asignaturas y prácticas de empresa.

En su lugar, hemos optado por intentar adaptarla a nuestro equipo. Hemos ido avanzando con pequeñas implementaciones de características en nuestra aplicación, y las hemos ido mejorando y ampliando. Hemos mantenido una frecuencia de reuniones del equipo prácticamente semanales. En estas reuniones repartíamos las tareas que teníamos que realizar cada miembro del equipo y discutíamos los avances planeados y dificultades encontradas. Siempre que hemos tenido alguna dificultad se la hemos transmitido a nuestros compañeros y las hemos ido solventando sobre la marcha.

Git/GitLab

Por otro lado, hemos utilizado como sistema de gestión de control de versiones de nuestro código la tecnología de *Git* [14], sobre la plataforma que ofrece *GitLab* [15] en su versión gratuita.

Hemos creado dos repositorios:

- Uno para la aplicación móvil Android: https://gitlab.com/jjopc/tfg
- Otro para la aplicación web Django y API REST: https://gitlab.com/jjopc/tfg-backend-docker

Como metodología de trabajo para la gestión de ramas y nombrado de las características que íbamos implementando hemos intentado seguir el sistema de *Gitflow* [16], para poder seguirla utilizamos la herramienta con el mismo nombre *git-flow*, que es multiplataforma y está disponible en su repositorio en github. A modo de ilustración mostramos en la siguiente imagen el flujo general que expone Vincent Driessen en su artículo [16]:

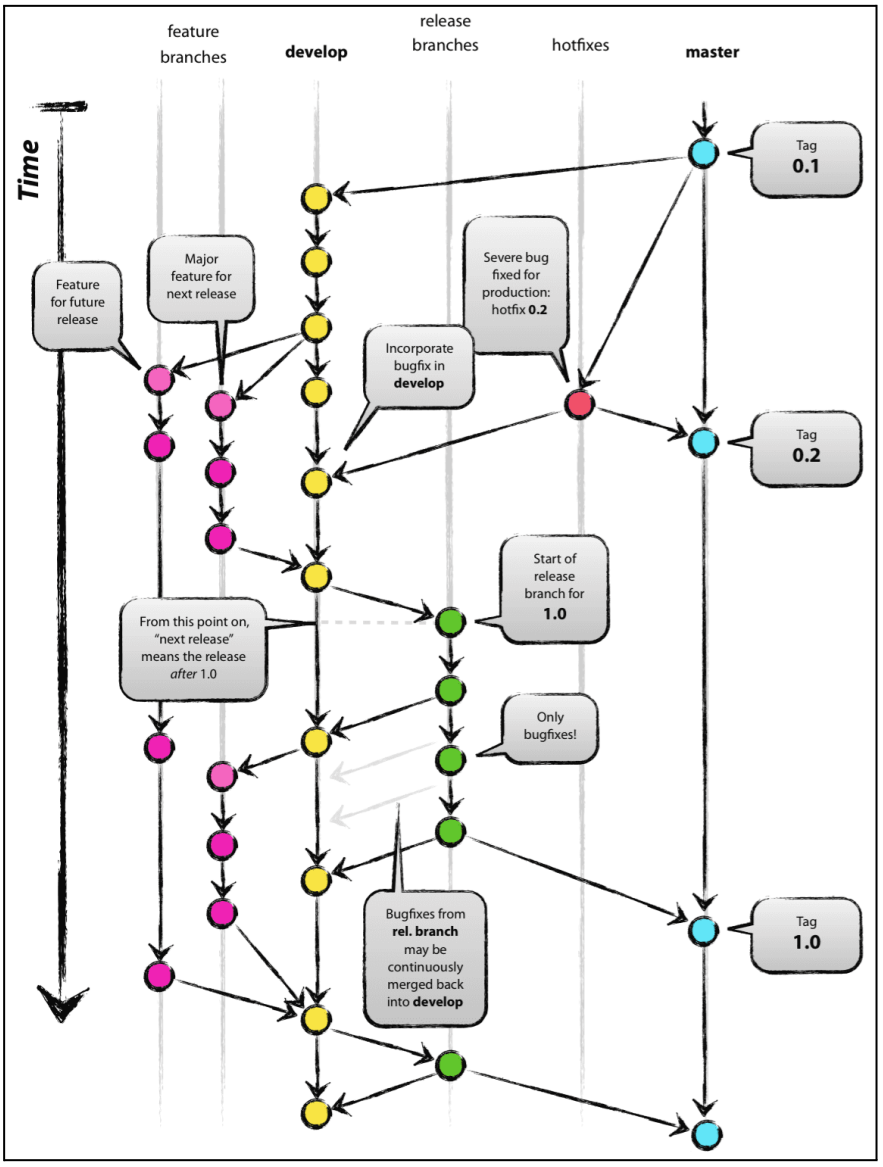


Imagen 10: Modelo de Git-flow. Author: Vincent Driessen. Original blog post: http://nvie.com/posts/a-succesful-git-branching-model. License: Creative Commons BY-SA.

Dentro de las herramientas proporcionadas por esta plataforma hemos usado un tablero para organizar las tareas. Las columnas de este tablero las organizamos en categorías:

- "Abierta": se corresponde con el *Product Backlog* de la metodología *Scrum*, es donde aparecen las *issues* que creamos. Estas *issues* se corresponden con las historias de usuario previamente planeadas.
- "*Sprint-backlog*", se corresponde con el *Sprint Backlog* de *Scrum*.
- "*Developing*", en esta columna, cada miembro tenía una tarea asignada en la que se encontraba trabajando en ese momento.
- "*Testing*", cuando se terminaba de implementar la tarea y previo al *merge* o fusión con la rama de desarrollo.
- "*Bugs*", cuando habíamos terminado con una tarea, pero posteriormente nos dábamos cuenta de algún error, la tarea se abría de nuevo y pasaba a esta columna.

- "*Closed*", cuando se había concluido la tarea y se había fusionado con la rama de desarrollo.

Además, hemos creado una serie de etiquetas para distinguir visualmente el tema del que trataba la tarea. Estas etiquetas son las "*User Activities*" del *User Story Mapping* que habíamos creado previamente.

Tal y como se puede ver en la siguiente imagen:

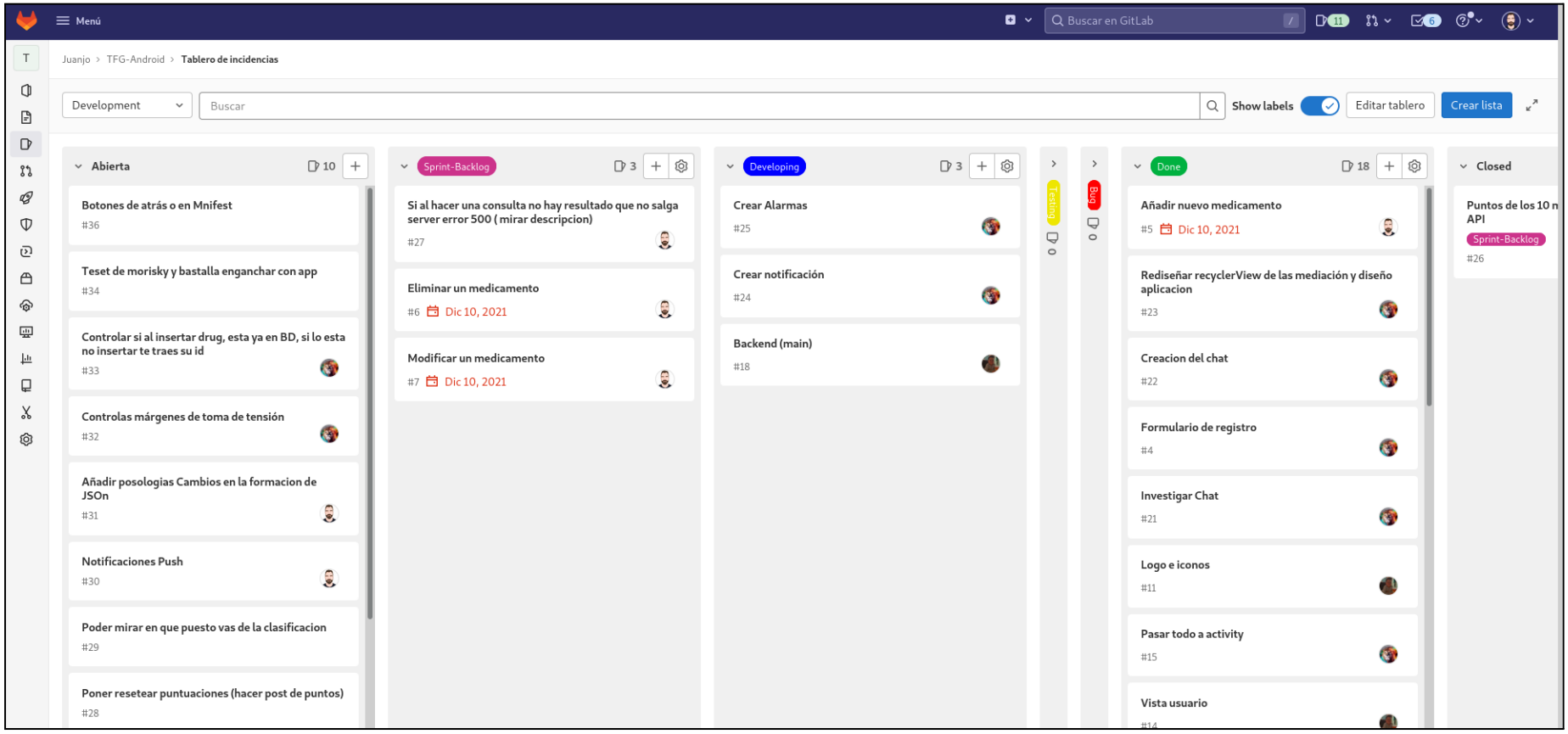


Imagen 11: Tablero que muestra las incidencias o *issues* en el repositorio para la aplicación móvil Android.

En la siguiente imagen mostramos una *issue* o historia de usuario en detalle:

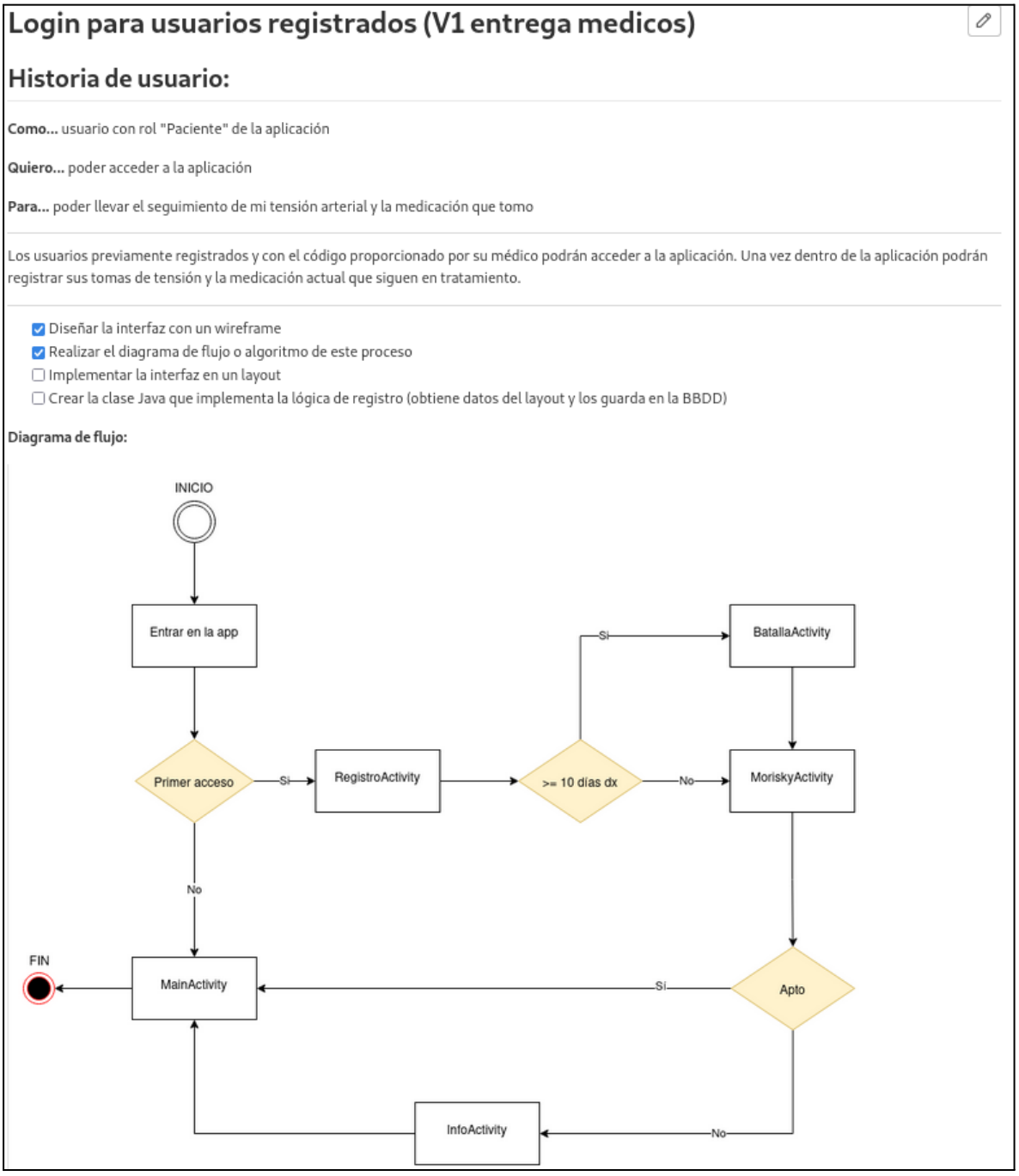


Imagen 12: Detalle de un *issue* en el que se muestra la historia de usuario, junto con subtareas a realizar y un diagrama de flujo. Pertenece a la aplicación móvil Android.

En el Apéndice III: Historias de usuario se muestran de forma detallada todas estas. Las historias de usuario las repartimos en las reuniones y las asignamos en Gitlab.

Servidor Virtual Privado

Desde un principio el *stack* de tecnologías que íbamos a utilizar era complejo de gestionar localmente en nuestros equipos personales. Rápidamente empezaron a surgir conflictos con las versiones de los programas y lenguajes que utilizamos en nuestros equipos, y además, para trabajar con la base de datos localmente requería su instalación y configuración adecuada.

Adicionalmente, entornos de desarrollo como el que proporciona *Android Studio* consumen unos recursos desmesurados, por lo que cuanto menos carga tuviéramos localmente, mejor podríamos trabajar.

Esto nos llevó a utilizar un Servidor Virtual Privado [17] (*VPS*, de sus siglas en inglés), uno de los integrantes del equipo ya tenía uno contratado en la infraestructura que ofrece *Contabo* [18], por lo que su configuración para usarlo como base de datos centralizada para desarrollo y entorno de pruebas fue nuestra elección final. Esto nos ha facilitado mucho el trabajo ya que en nuestros equipos sólo hemos usado el Entorno Integrado de Desarrollo (*IDE*, de sus siglas en inglés) y en todo momento hemos tenido una base de datos centralizada en el servidor.

Chapter 1: Introduction

In this first chapter we make a brief introduction to the health problem that arterial hypertension represents in Spain, and at the same time we review the most important epidemiological data. This gives us the opportunity to introduce the reason for the "Yébenes Adherence to Antihypertensive Treatment" project proposed by the primary care doctors of the "Los Yébenes" Health Centre.

Secondly, we reproduce the objectives set out in the mentioned project, as well as our own objectives so that the project meets theirs.

Finally, we explain how we have organized ourselves as a team and the work methodologies followed.

1.1 Background information

Arterial hypertension (AHT) is currently one of the most prevalent pathologies in western society; in Spain it is present in 33% of the population (66% in those over 60 years old) and 40,000 deaths per year are attributed to it [1]. It is characterised by its scarce direct symptomatology and its association with multiple high-risk pathologies such as ischaemic heart disease, cardiovascular accidents or renal failure.

This pathology is strongly influenced by lifestyle, diet and habits. However, even if a healthy lifestyle is followed and a wide range of therapies are currently available, blood pressure levels can still be very high. Population studies suggest that only 26.6% of the hypertensive population maintains the blood pressure levels indicated as good control according to clinical guidelines.

The poor control of AHT can be explained by multiple factors, including the fact that it is an indolent process, with a significant number of hypertensive patients unaware that they suffer from this pathology. On the other hand, it is estimated that only some of these patients maintain a correct treatment regimen and adherence to treatment.

The primary care service of the "Los Yébenes" Health Centre, which we will refer to in the report as our clients, proposes the project "Yébenes Adherence to Antihypertensive Treatment - YATA (due to its acronym in Spanish)". With this project they aim to tackle both problems with the intention of obtaining better adherence to treatment, better control over blood pressure figures, and to attract previously undiagnosed hypertensive patients.

Today the use of smart devices, such as our mobile phones, is almost universal in our environment. They are light, we carry them with us all the time, and with just a few interactions we can complete a wide range of tasks. For this reason, our clients considered the development of an application that would help them achieve their goals.

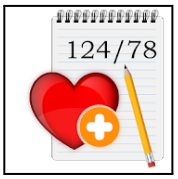
With the development of our application we aim, on the one hand, to influence the behavior of these hypertensive patients so that they improve their adherence to treatment and therefore improve their blood pressure. And, on the other hand, to make it easier for healthcare specialists to carry out their preventive and research work.

In addition, we plan to publish the application in the Android app store, Google Play [2], so that any user can use it for individualized monitoring of their blood pressure and medication intake.

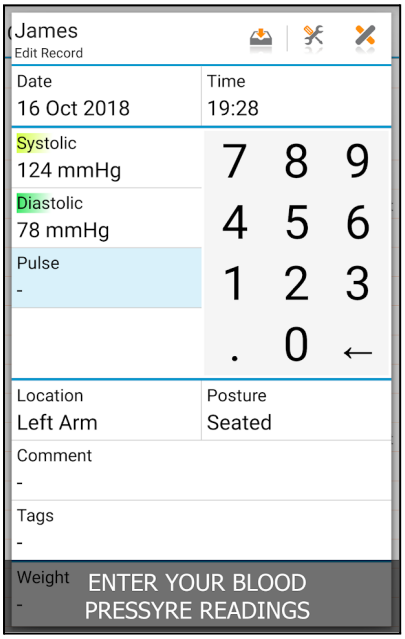
1.1.1 Related work

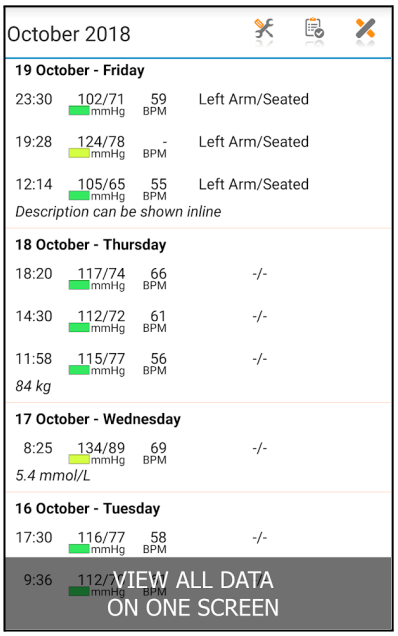
Before starting the design and development, the whole team researched other similar applications. From these, an intuitive user interface was designed. The implementation of a chat was also proposed as an idea for doctors and patients to have a fluid communication. The applications taken as a reference are the following:

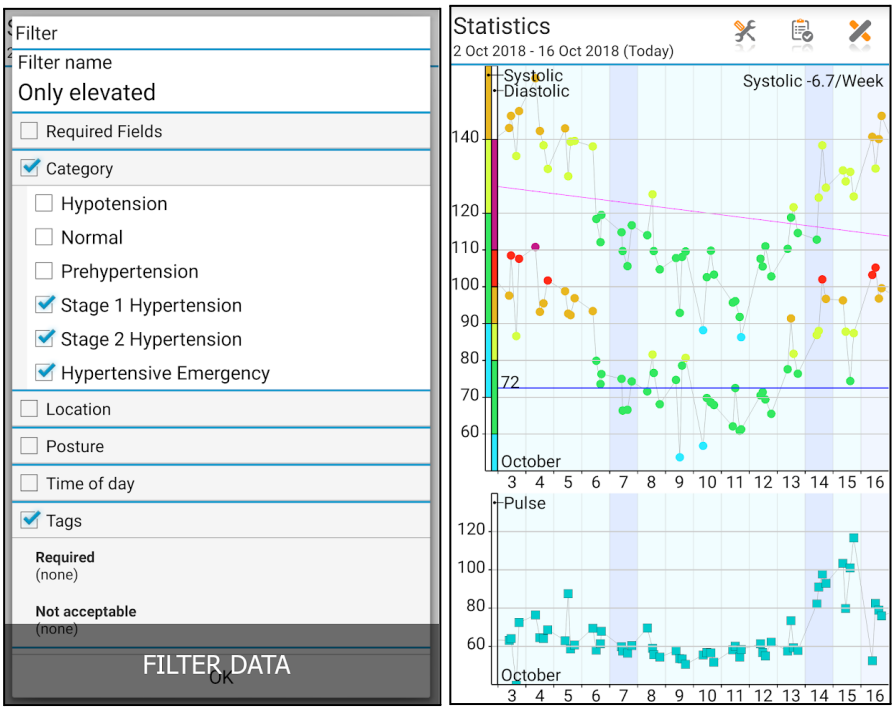
- Blood Pressure Log - MyDiary [3]
  Blood Pressure Log - MyDiary allows blood pressure logging with reminders, offers various statistics and the ability to export data in various formats.
  Some screenshots are as follows:



Picture 1: *Blood Pressure Log - MyDiary app logo.*



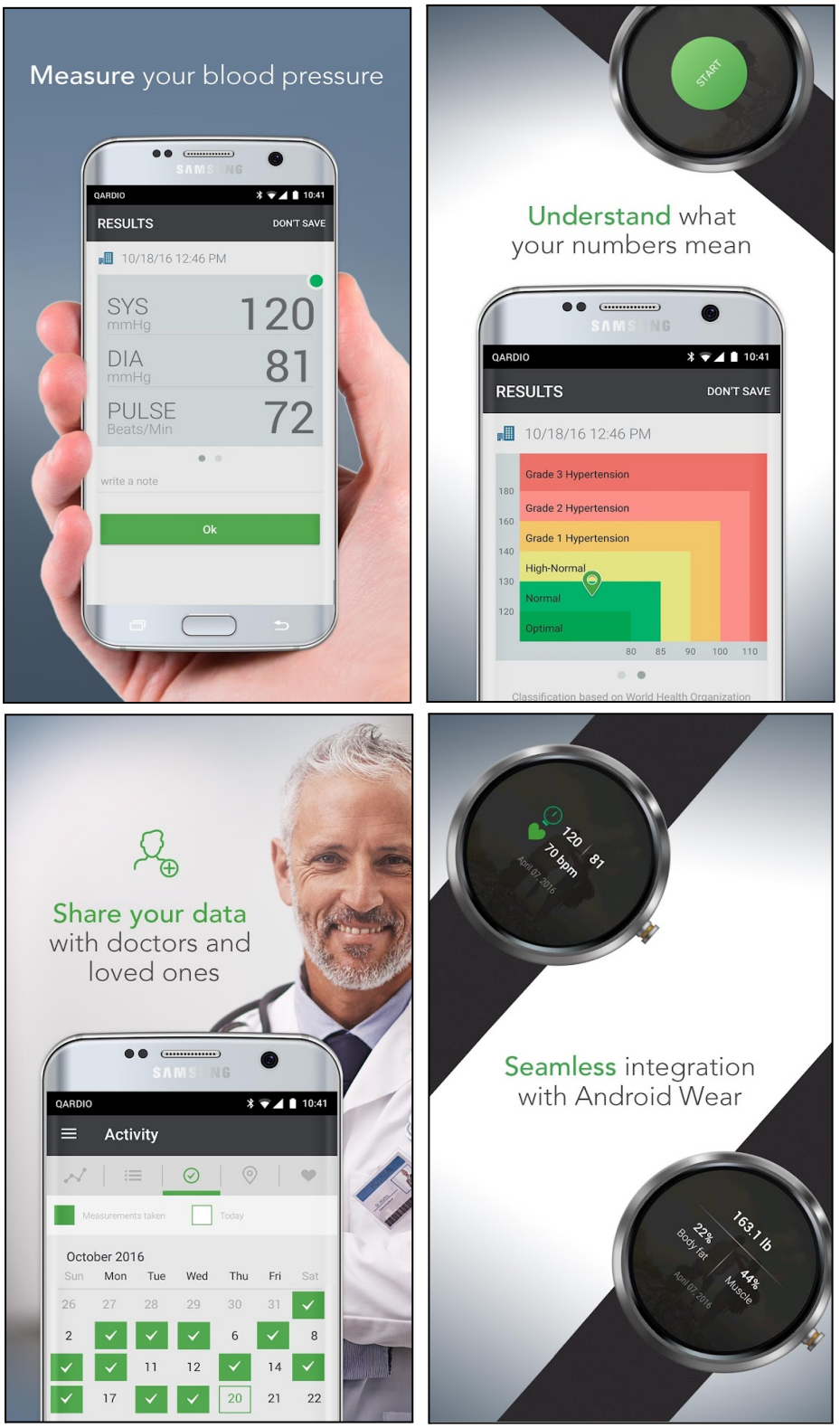




Picture 2: *Blood Pressure Log - MyDiary* app screenshots.

- Qardio Heart Health [4]
  Qardio Heart Health allows the tracking of blood pressure and weight. It offers the possibility to transfer data from other apps such as MyFitnessPal [5], Google Fit [6], Samsung Health [7]. It also offers the ability to view graphs, set reminders and share data.
  Some screenshots are the following:

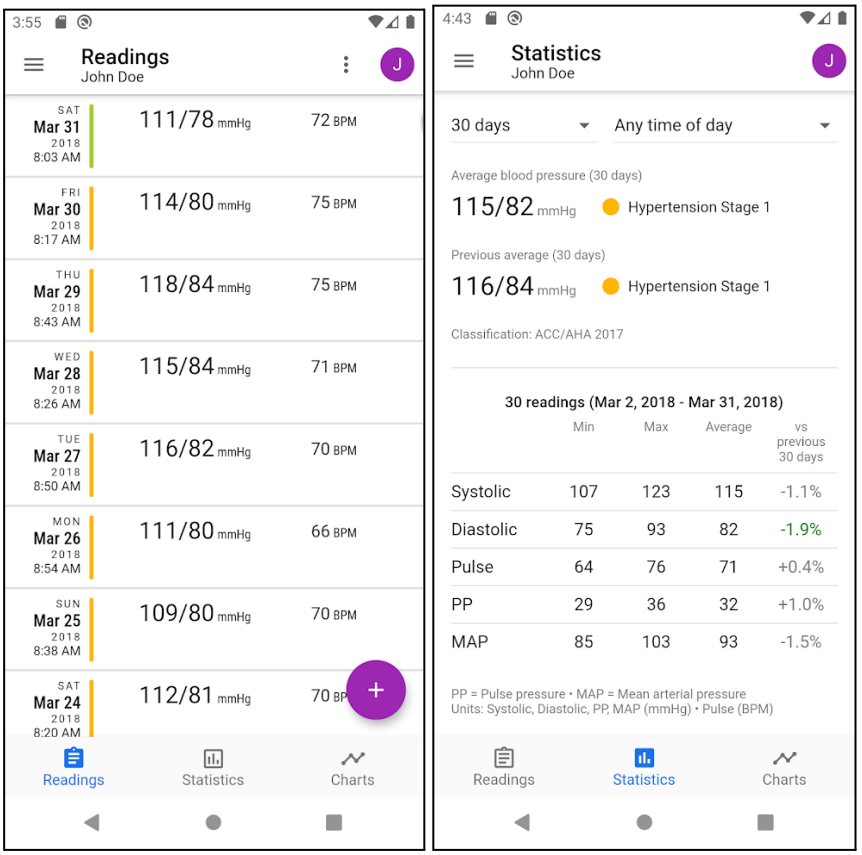
Picture 3: *Qardio Heart Health* app logo.

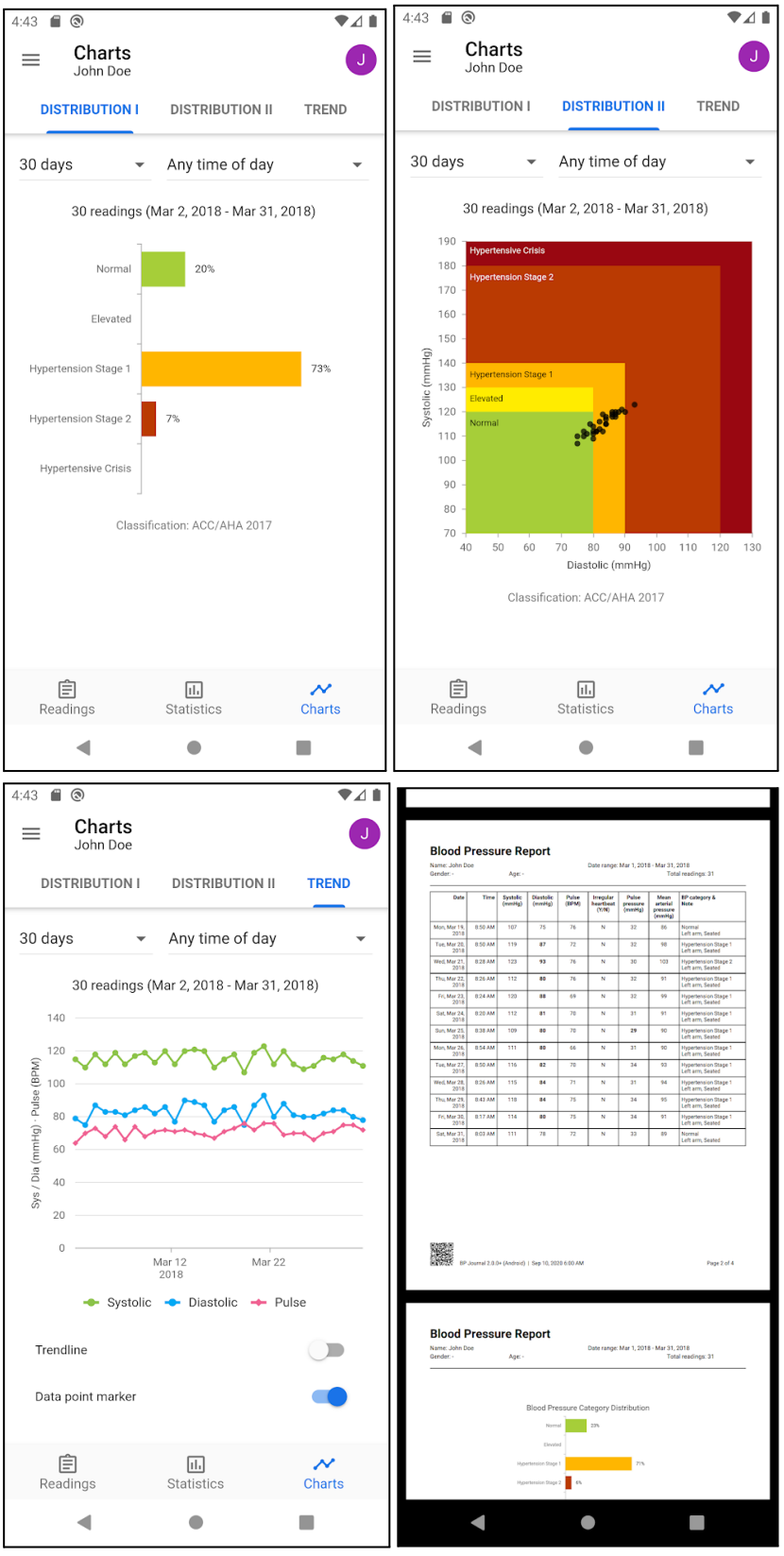


Picture 4:*Qardio Heart Health* app screenshots.

- BP Journal: Blood Pressure Log [8]
  BP Journal allows blood pressure and pulse logging, generates statistics and graphs with which you can export or import this data in CSV format with other applications and send reports to your doctor.
  Some screenshots are the following:

Picture 5: *BP Journal: Blood Pressure Log* app logo.

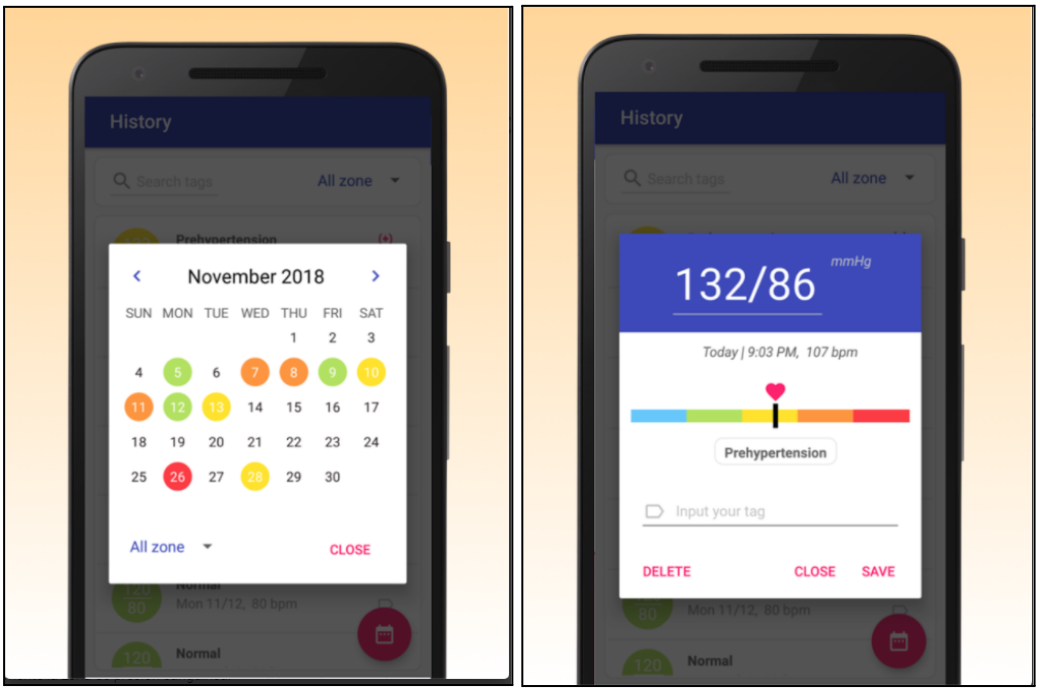




Picture 6: *BP Journal: Blood Pressure Log* app screenshots.

- Blood pressure recording (Health & Fitness AI Lab) [9]
  Like the previous ones, it allows you to record blood pressure readings by including labels on the readings, which can then be searched later. It provides date filters, labels and blood pressure zones for searches. It allows data to be exported in CSV format.
  Some screenshots are the following:

Picture 7: *Registro de presión arterial (Health & Fitness AI Lab)* app logo.



Picture 8: *BP Journal: Blood Pressure Log* app screenshots.

Although there are more similar applications in Google's app shop, we have highlighted these mainly because of their similarity in terms of features, the fact that they are free and because they have a better interface.

1.2 Objectives

The client proposes the following objectives for their project:

1. To obtain better adherence to treatment.
2. To achieve better control of blood pressure.
3. To involve patients with hypertension in their disease and, as far as possible, to empower their patients and give them tools to increase their autonomy.
4. To increase the recruitment of hypertensive patients.

In our behalf, in order for our client to achieve its objectives, we set out the following:

1. To develop a mobile application for the patients of the "Los Yébenes" Health Centre, which allows:
    a. To record blood pressure measurements.
    b. To record the medication taken.
    c. To display reminders for taking medication.
    d. To perform the *Batalla* test [10].
    e. To carry out the Morisky-Green test [11].
    f. To motivate users through a points-based competition system.
    g. To educate users on concepts related to hypertension.
    h. To send private messages with questions to their assigned doctor at the health center.
2. To develop a web application for doctors and health staff at the "Los Yébenes" Health Centre, which allows them to:
    a. To visualize the information recorded by their patients to monitor them.
    b. To obtain statistics on adherence to treatment by their patients.
    c. To be able to communicate with their patients through an internal messaging system.

1.3 Work plan

The final degree project itself required the development of a mobile application, so this requirement was quite clear. However, the development of a mobile application was a challenge for our team, since two members had never programmed for mobile platforms before and the other had the experience of having taken the subject "Programming applications for mobile devices".

On the other hand, an important functionality is the control and global vision that our client must be able to obtain. So that they can evaluate the level of fulfillment of the project's goals. To this end, we concluded that the ideal solution would be the development of a web application, which could be accessed from any device with an internet connection, including the health center's computers.

As for the technologies to be used, we will go into more detail on why we made our decisions in Chapter 4: Discussion.

Once the basic technologies for the development of the applications had been selected, it was now time to make decisions related to the organization of the work team formed by the three colleagues.

1.3.1. How did we organize?

User Story Mapping

To start with the application approach we employed the "User Story Mapping" technique [12], which follows a fairly simple to understand idea. It talks about the user's journey through the application, building a simple model that tells the user's story as they use the application. This allows working with user stories in agile development methodologies.

User stories are distributed as shown in the following image:

Picture 9: *User Story Mapping* with user`s stories.

The first row corresponds to the *"User Activities"*, which are "coarse-grained" activities that will be carried out by the user with our application. Each of these activities creates a column.

The second row corresponds to the *"User Tasks"*, which are "medium-grain" activities, slightly more specific than the previous ones.

The following rows contain the real user stories, divided into *"releases"* or different versions of the application. The division into releases is an estimate of what we will be able to achieve in our project.

The first *release* would be an application with the minimum functionality to be of interest to its target audience.

Scrum

In this sense, we have tried to follow an agile methodology, specifically Scrum [13], although this has not been entirely easy, mainly because it requires an almost exclusive dedication to the project, which has been impossible due to the fact that the project is compatible with subjects and company internships.

Instead, we have opted to try to adapt it to our team. We have been moving forward with small implementations of features in our application, and we have been improving and extending them. We have maintained a frequency of almost weekly team meetings. In these meetings we distributed the tasks to be performed by each team member and discussed the planned progress and difficulties encountered. Whenever we have had any difficulties, we have always passed them on to our colleagues and we have solved them as we went along.

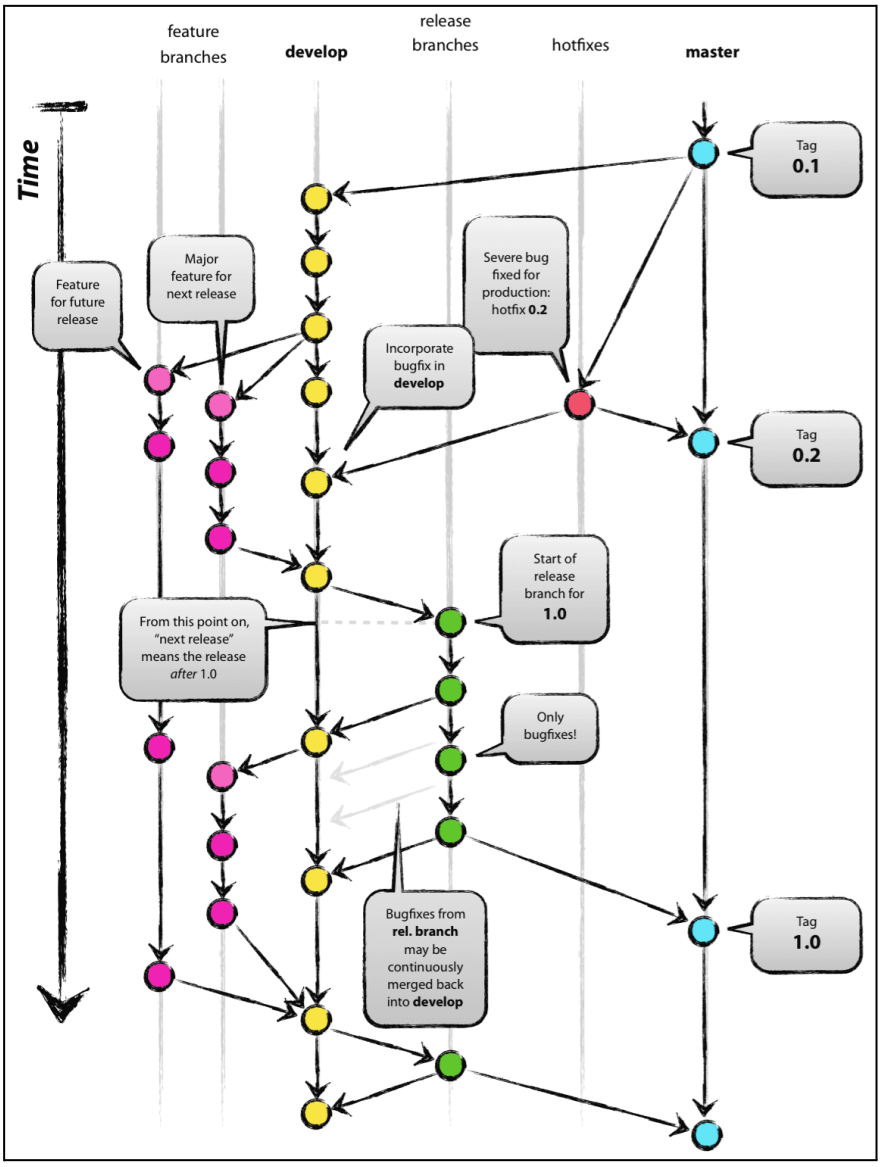
Git/GitLab

On the other hand, we have used *Git* technology [14] as our code version control management system, on the platform offered by *GitLab* [15] in its free version.

We have created two repositories:

- One for the Android mobile app: https://gitlab.com/jjopc/tfg
- Another one for the Django web application and REST API: https://gitlab.com/jjopc/tfg-backend-docker

As a working methodology for managing branches and naming the features that we were implementing, we have tried to follow the *Gitflow* [16] system, using the tool with the same name *git-flow*, which is multiplatform and is available in its repository on github. By way of illustration, the following image shows the general flow described by Vincent Driessen in his article [16]:



Picture 10: Git-flow model. Author: Vincent Driessen. Original blog post: http://nvie.com/posts/a-succesful-git-branching-model. License: Creative Commons BY-SA.

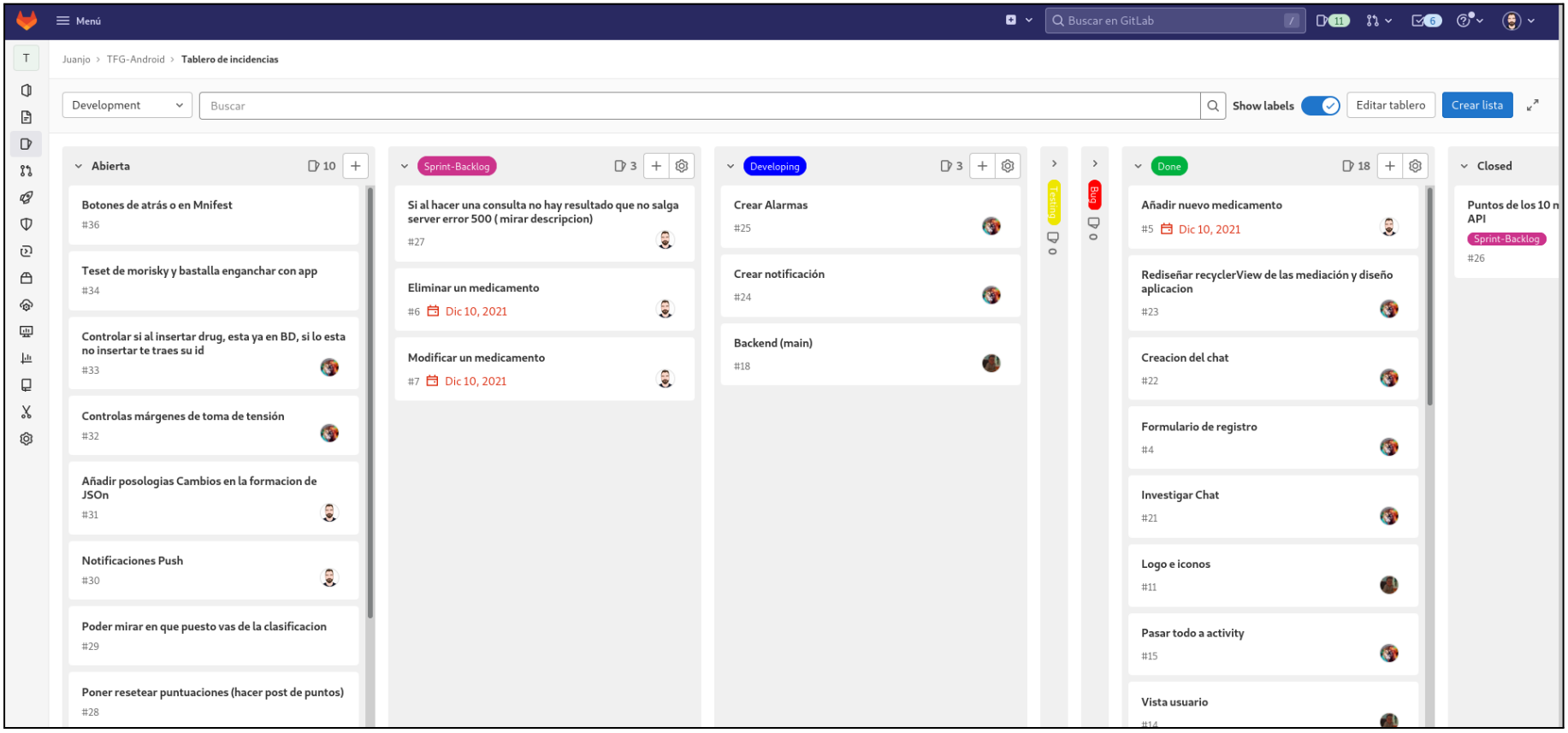
Within the tools provided by this platform we have used a dashboard to organize the tasks. The columns of this board are organized into categories:

- *"Open"*: Corresponds to the Product Backlog of the Scrum methodology, it is where the issues that we create appear. These issues correspond to the previously planned user stories.
- *"Sprint-backlog"*, corresponds to the Scrum Sprint Backlog.
- *"Developing*", in this column, each member had an assigned task they were working on at the time.
- *"Testing"*, when we had finished implementing the task and prior to the merge or merge with the development branch.
- *"Bugs",* when we finished a task, but later on we noticed a bug, the task was re-opened and moved to this column.

- *"Closed"*, when the task had been completed and merged with the development branch.

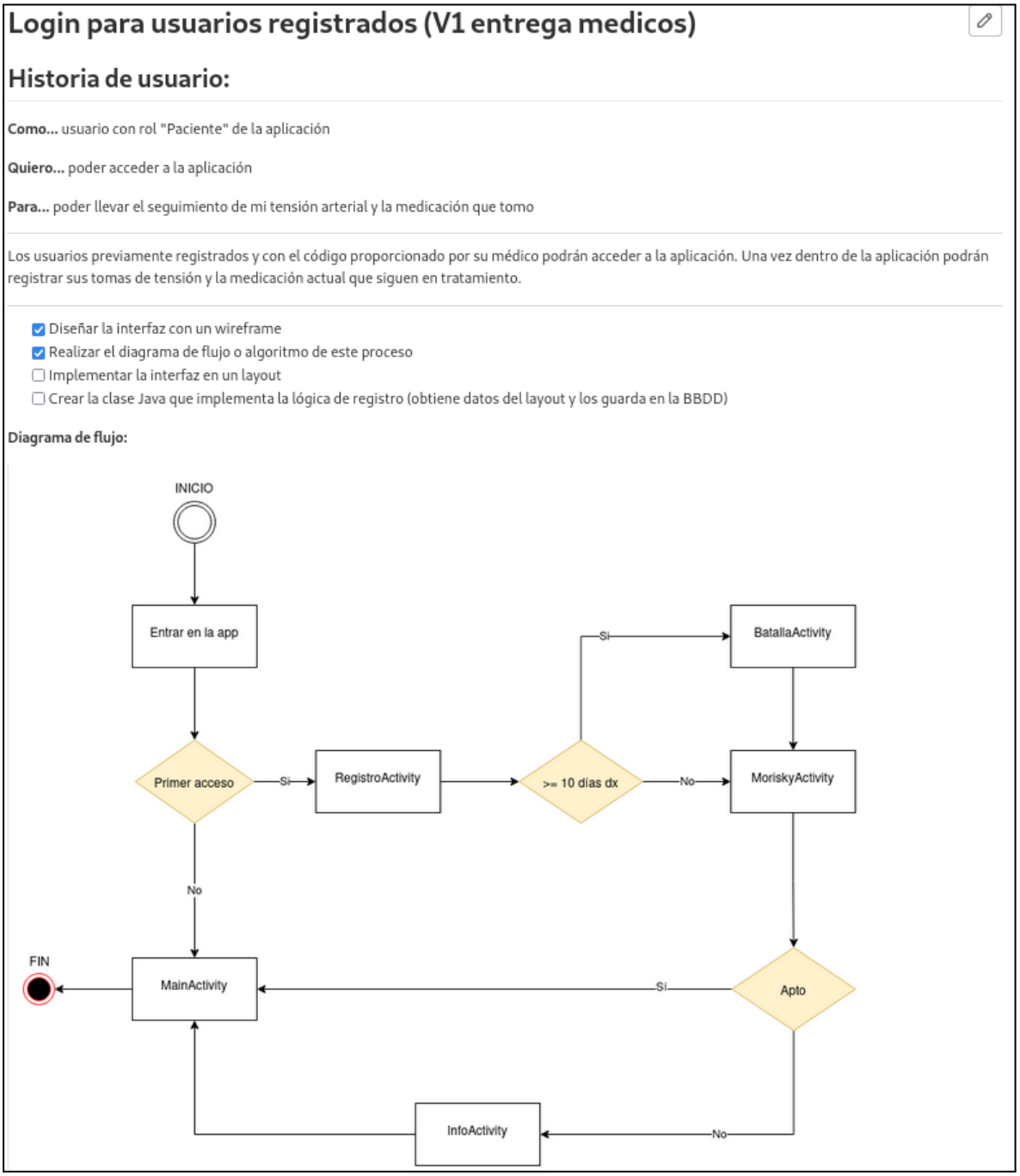
In addition, we created a series of labels to visually distinguish what the task was about. These labels are the *"User Activities"* from the User Story Mapping we had previously created.

As you can see in the following picture:



Picture 11: Dashboard showing issues in the repository for the Android mobile application.

In the following image it is shown an *issue* or user story in detail:



Picture 12: Detail of an issue showing the user story, along with subtasks to be performed and a flowchart. It belongs to the Android mobile application.

The Appendix III: User Stories shows all of these in detail. The user stories were distributed in the meetings and assigned in Gitlab.

Virtual Private Server

From the beginning, the stack of technologies we were going to use was complex to manage locally on our personal computers. Conflicts quickly started to arise with the versions of the programmes and languages we used on our computers, and also, working with the database locally required its installation and proper configuration.
Additionally, development environments such as the one provided by Android Studio consume excessive resources, so the less load we had locally, the better we could work.
This led us to use a Virtual Private Server [17] (VPS), one of the team members already had one contracted in the infrastructure offered by Contabo [18], so its configuration to use it as a centralized database for development and test environment was our final choice. This has made our work much easier as we have only used the Integrated Development Environment (IDE) in our teams and at all times we have had a centralized database on the server.

Capítulo 2: *Requisitos*

En este capítulo exponemos los requisitos tanto funcionales como de la interfaz de usuario que consideramos necesarios para el correcto funcionamiento de la aplicación.

2.1 Requisitos funcionales

En este apartado se detallan los requisitos y precondiciones necesarias para el correcto funcionamiento de la aplicación.

2.1.1 Aplicación móvil

Todos los requisitos requieren *smartphone* con sistema operativo Android y conexión a internet.

- Registrar usuario:
  - Código de médico y paciente proporcionado por parte del médico.
  - Contraseña con cierto nivel de complejidad (al menos 8 caracteres no ser completamente numérica, no puede ser una contraseña común ni parecerse a la información personal del usuario) [19].
  - Nombre de usuario válido (que no esté ya en la base de datos).

- Iniciar sesión:
  - Tener un usuario ya registrado con un nombre de usuario.
  - Contraseña que se estableció en el registro.

- Registrar la presión arterial:
  - Valores de presión sistólica entre los rangos de 90-170 mm Hg.
  - Valores de presión diastólica entre los rangos de 50-110 mm Hg.
  - Indicar el brazo de la toma.

- Registrar la toma de medicación:
  - Añadir el nombre del medicamento.
  - Añadir el número de pastillas a tomar en el desayuno, comida, cena o al acostarse.
  - Añadir la fecha de inicio de la toma del medicamento.
  - Añadir la fecha de finalización de toma del medicamento.

- Comunicar al paciente con su doctor asignado mediante un sistema de mensajería:
  - Abrir el chat privado con el doctor asignado.
  - Mandar mensajes al doctor asignado.
  - Recibir mensajes del doctor asignado.

- Cumplimentar el test de Batalla:
  - Responder a cada una de las preguntas del test.
  - Obtener *feedback* acerca de cómo ha hecho el test.
  - Si no pasa el test debe mostrar información acerca de la hipertensión arterial.

- Cumplimentar el test de Morisky-Green:
  - Responder a cada una de las preguntas del test.
  - Conservar los datos del test para valorar adherencia al tratamiento.
  - Se realiza al inicio, a los 3 meses y a los 6 meses.

- Realizar un registro AMPA (Automedición de la presión arterial):
  - Sólo en caso de presión arterial mal controlada.
  - Valores de presión sistólica entre los rangos de 90-170 mm Hg.
  - Valores de presión diastólica entre los rangos de 50-110 mm Hg.
  - Indicar el brazo de la toma.
  - Posibilidad de añadir comentarios a la toma.
  - Registrar fecha y hora de la toma.
  - Registrar 3 tomas seguidas por la mañana y por la tarde durante 4 días consecutivos.
  - Calcular la media de la mañana y la media de la tarde.
  - Dar un resultado de AMPA al final del protocolo.

- Mostrar sección de Preguntas y Respuestas:
  - ¿Qué hacer si tengo la tensión muy baja?
  - ¿Qué hacer si tengo la tensión muy alta?
  - ¿Qué es la hipertensión arterial?
  - ¿Cómo tomar correctamente la tensión arterial?
  - Consejos sobre dieta sana, hábitos saludables, etc.

2.1.2 Aplicación web

Todos los requisitos requieren conexión a Internet.

- Iniciar sesión:
  - Tener un código de doctor asociado.
  - Proporcionar formulario de inicio de sesión para que el doctor tenga acceso a la aplicación.

- Consultar datos de tensión de un paciente:
  - Ver listado de pacientes asociados al doctor.
  - Seleccionar un paciente.
  - Ver su listado de tomas de tensión.
  - Poder visualizar una toma en concreto.

- Consultar medicación de un paciente:
  - Ver listado de pacientes asociados al doctor.
  - Seleccionar un paciente.

  - Ver su listado de medicamentos.
  - Poder visualizar un medicamento en concreto, su posología.
  - Ver cómo de bien toma un paciente un medicamento (adherencia al tratamiento).

- Consultar los tests de Moriski-Green realizados por un paciente:
  - Ver listado de pacientes asociados al doctor.
  - Seleccionar un paciente.
  - Ver su listado de tests realizados.
  - Poder visualizar un test en concreto.

- Consultar estadísticas generales de todos los pacientes:
  - Ver el número de pacientes totales de la aplicación.
  - Ver el número de controles de tensión arterial de un paciente.
  - Ver un listado de medicamentos empleados.
  - Ver cuántos pacientes toman un medicamento en concreto.
  - Ver cuántos AMPAs se han realizado y los pacientes que lo han realizado.

2.2 Requisitos de interfaz de usuario

El análisis previo descrito en el punto 1.1.1 Trabajos relacionados del Capítulo 1: Introducción nos ha permitido esbozar nuestra aplicación. De este análisis obtuvimos algunas ideas en cuanto a la forma de introducir los datos de las tomas de tensión de los pacientes, la medicación que toman, o la forma de representar toda la información que puede ser interesante para el paciente. Algunas se adoptaron para las primeras versiones y se han mantenido, o se han ido desechando conforme el proyecto ha evolucionado.

Finalmente para la representación de la medicación en la aplicación móvil se ha adoptado un diseño lo más parecido posible a lo que los pacientes están acostumbrados cuando el médico les proporciona su hoja de medicamentos y posologías correspondientes. También se han usado gráficos y tablas que hacen más visual y sencillo para el usuario el seguimiento de sus tomas recientes de tensión. Para el chat con el médico se ha intentado que fuera lo más familiar posible para el usuario, comparado con otras aplicaciones de mensajería que pueda usar.

Fundamentalmente el requisito que ha guiado nuestro diseño es intentar hacer una interfaz gráfica sencilla de utilizar.

Como es de suponer, tanto la aplicación móvil como la aplicación web han estado sujetas a un conjunto de cambios con respecto a la idea inicial. En concreto la que más modificaciones ha sufrido es la aplicación móvil, y esto se ha debido a algunos factores como que fuera la aplicación principal y que hemos entendido que era la más importante del proyecto, porque es la que más tiempo se ha tenido para desarrollar, o porque es para la que más ideas iniciales se tenían, así como más *feedback* por parte de los médicos.

2.2.1 Bocetos

Para el desarrollo de la interfaz gráfica, realizamos unos bocetos estos fueron presentados a nuestro cliente en una reunión y cuando nos dieron su visto bueno comenzamos con el desarrollo. Estos bocetos los hemos hecho a mano y con la herramienta Draw.io [20] que es gratuita, multiplataforma y cuenta con una aplicación web que permite el trabajo simultáneo con otros compañeros.

Aquí tan solo mostramos tres de ellos, el resto están en los apéndices.

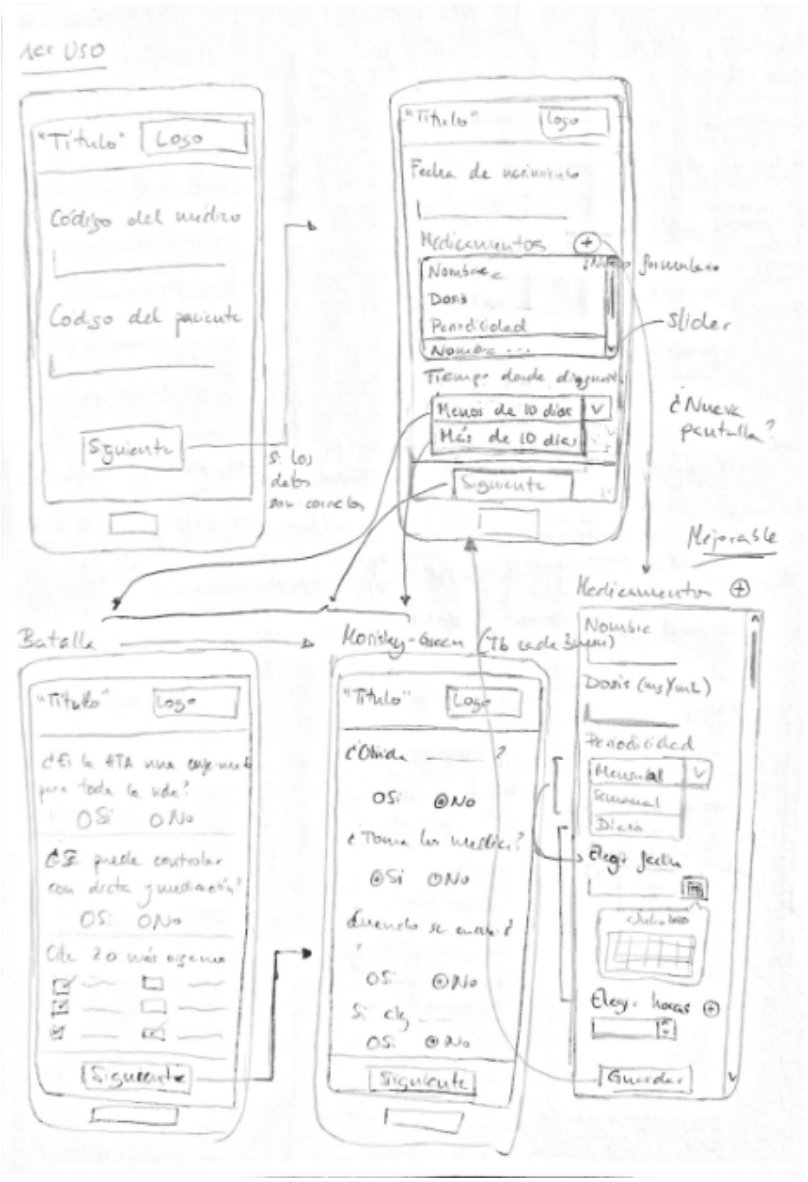


Imagen 13: Boceto a mano alzada del primer uso de la aplicación móvil.

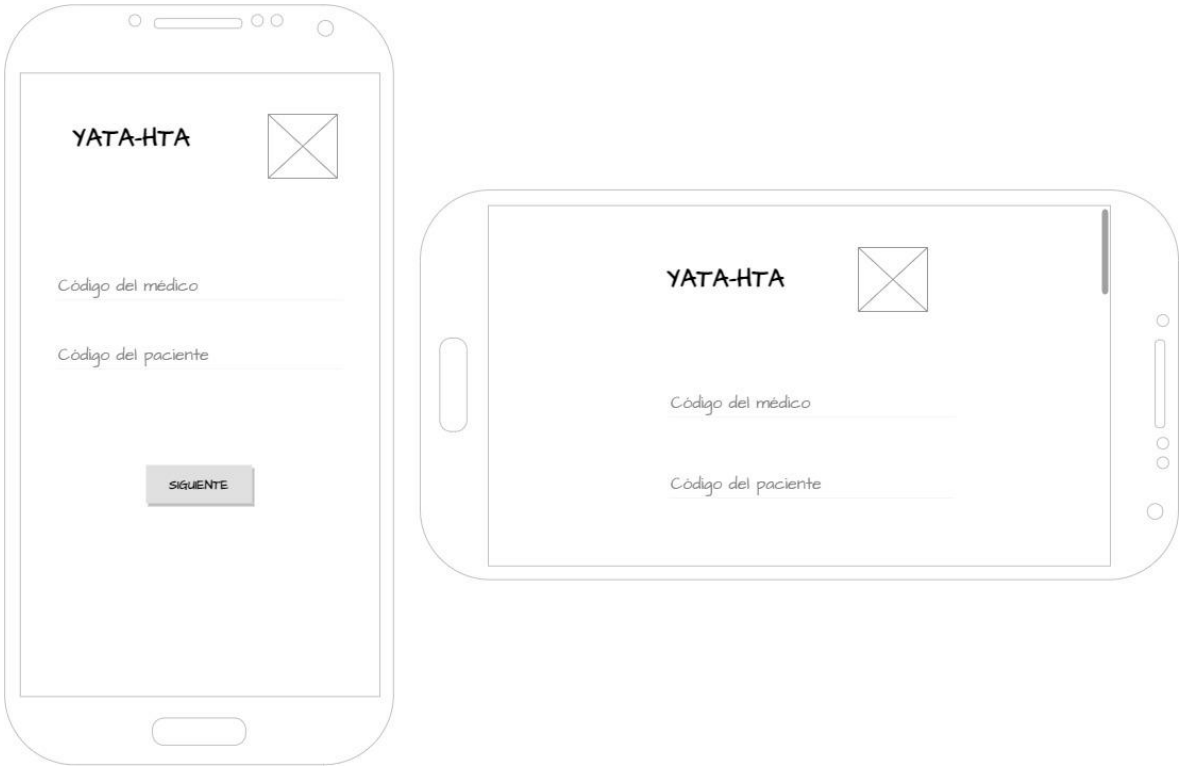


Imagen 14: Pantalla de registro de nuevos pacientes - vista 1.

Imagen 15: Pantalla de registro de nuevos pacientes - vista 2.

Capítulo 3: *Resultados*

En este capítulo muestra todas las vistas de la app Android y de la app Web con una explicación de su funcionalidad.

3.1 Aplicación móvil

3.1.1 Login

El primer contacto que vamos a tener con la app va a ser con la pantalla de **login**. Para acceder es necesario autenticarse mediante el usuario y contraseña. Este usuario y contraseña previamente han tenido que ser registrados ya que para acceder se realiza una consulta a la base de datos (BD). En el caso de que se quiera mantener la sesión iniciada y no volver a iniciar sesión cada vez que iniciamos la app se tiene que marcar el *checkbox* de "Recuerda mis datos" como muestra la imagen.
En el caso de que se pulse el botón de iniciar sesión con los dos campos de nombre usuario y contraseña vacíos, se nos mostrará en la pantalla un mensaje de que tenemos que introducir los datos.

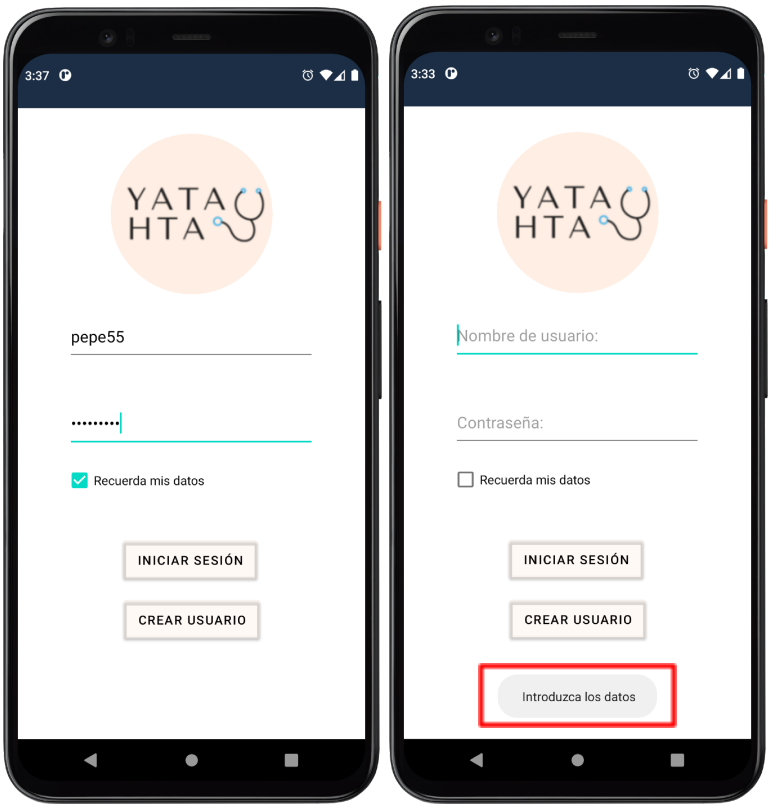


Imagen 16: Vista de Login.

3.1.2 Registro

Para la creación de un nuevo usuario, el formulario solicita la siguiente información: código médico y código de paciente (estos códigos previamente se han tenido que generar desde la aplicación web del médico y ser entregados al usuario), nombre de usuario, contraseña y fecha de nacimiento. Una vez creado el nuevo usuario, ya se podrá loguear.

En el caso de que se produzca algún error, como no rellenar todos los datos, que las contraseñas no coincidan o tenga poca seguridad, nos aparece un mensaje indicando el tipo de fallo que hemos cometido.

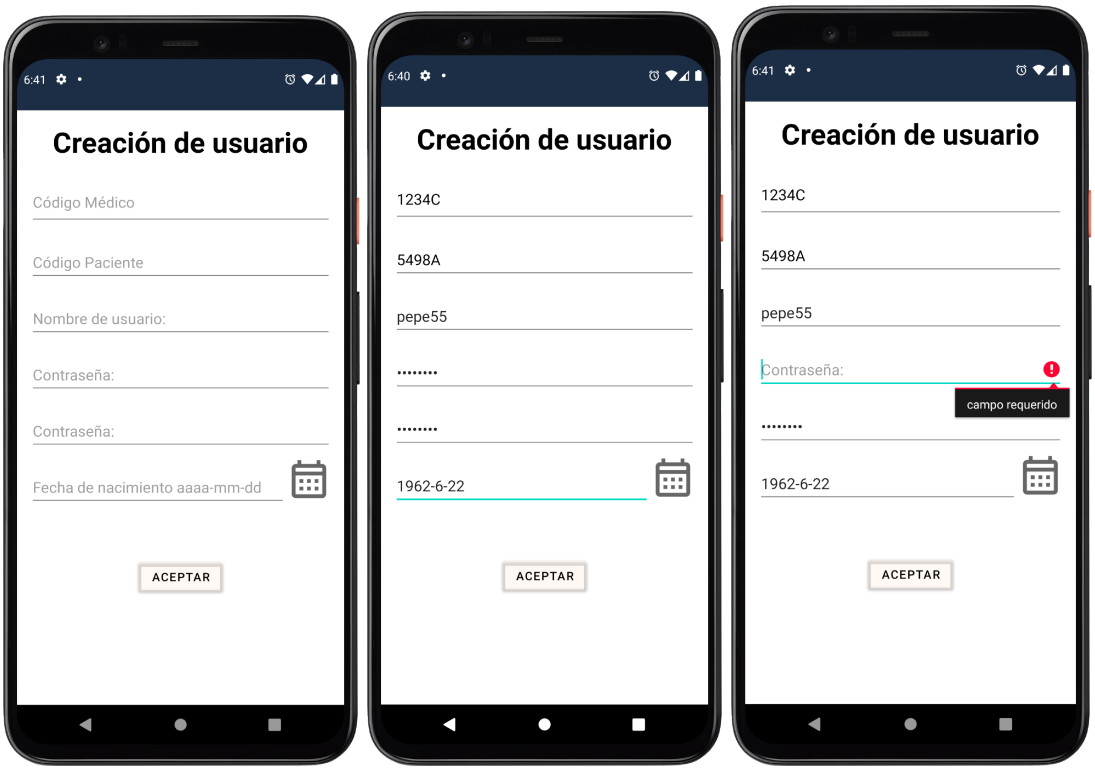


Imagen 17: Vista de Registro.

3.1.3 Perfil de usuario

La sección del perfil de usuario es una de las cuatro vistas principales. En esta podemos ver cúal es su nombre de usuario, la puntuación que tiene y su posición en la tabla de clasificación.

Otras opciones que están implementadas son la de cerrar sesión y la de configuración de alarmas. Se accede pulsando a los respectivos iconos que aparecen en la parte superior derecha.

Imagen 18: Vista de perfil usuario.

3.1.4 Configuración de alarmas

Para acceder al configurador se tiene que pulsar en el icono del reloj que aparece en la parte superior derecha de la pantalla del perfil de usuario. Nos permite seleccionar a qué hora queremos que la alarma se active y nos avise de que tenemos que tomar la medicación.

Para guardar una hora, lo primero que tienes que indicar es a qué hora quieres el aviso y una vez introducido el valor, se tiene que pulsar el icono de la campana para que esta se active.

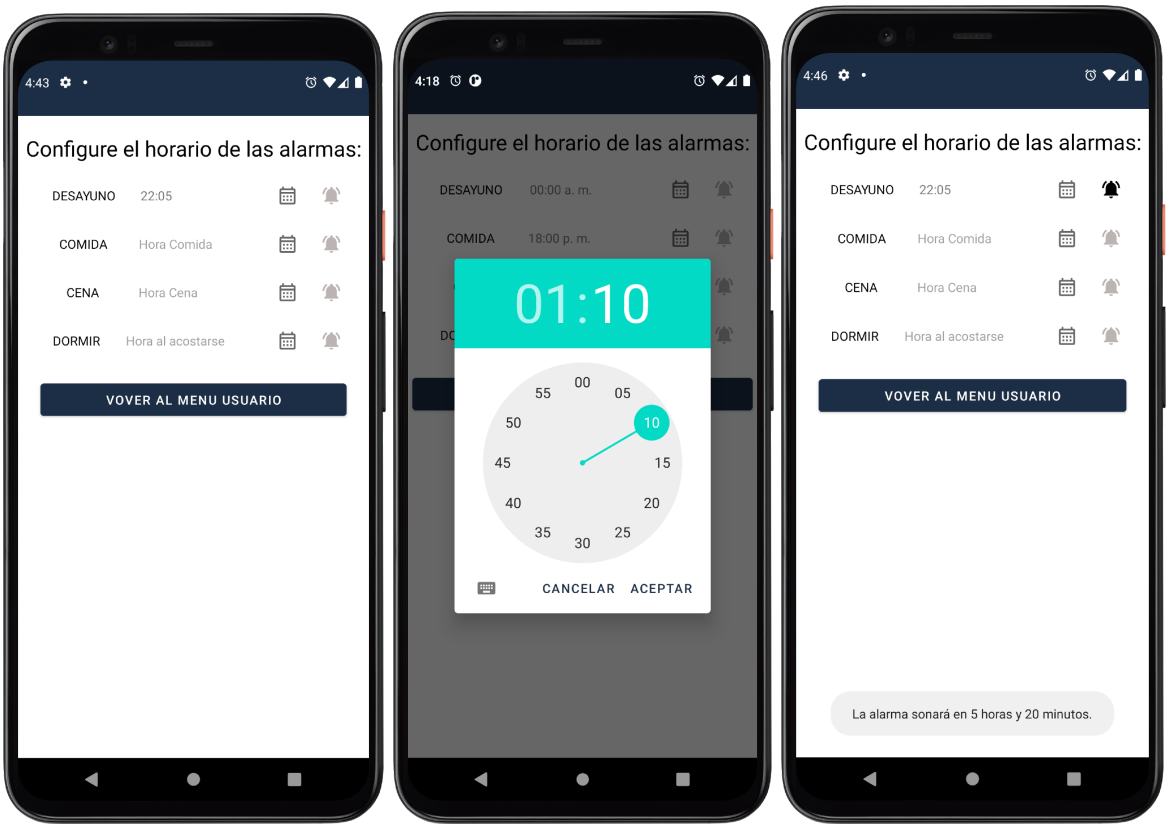


Imagen 19: Vista de configuración de alarmas.

3.1.5 Tensión arterial

Esta es la segunda de las cuatro vistas principales. En esta vista se nos permite añadir una toma nueva de tensión y podremos verla tanto en la gráfica como en la tabla. Para añadir una nueva toma de tensión se tiene que pulsar el icono que aparece en la parte inferior derecha.

Imagen 20: Vista de tensión arterial.

3.1.6 Añadir tensión arterial

El registro de una nueva toma de tensión solicita la presión arterial sistólica, la presión arterial diastólica, el pulso del corazón y el brazo en que se ha realizado la toma de la tensión.

Para que la toma de tensión sea válida, hay unos márgenes superior e inferior tanto para la presión sistólica, la presión diastólica como las pulsaciones que se deben cumplir, si no se cumplen estos márgenes, la propia aplicación nos avisará con el tipo de error..

Los valores son los siguientes:

Sistólica	90-170 mm Hg
Diastólica	50-110 mm Hg
Pulsaciones	40-250 ppm

Tabla 1: Márgenes correctos de toma de tensión arterial.

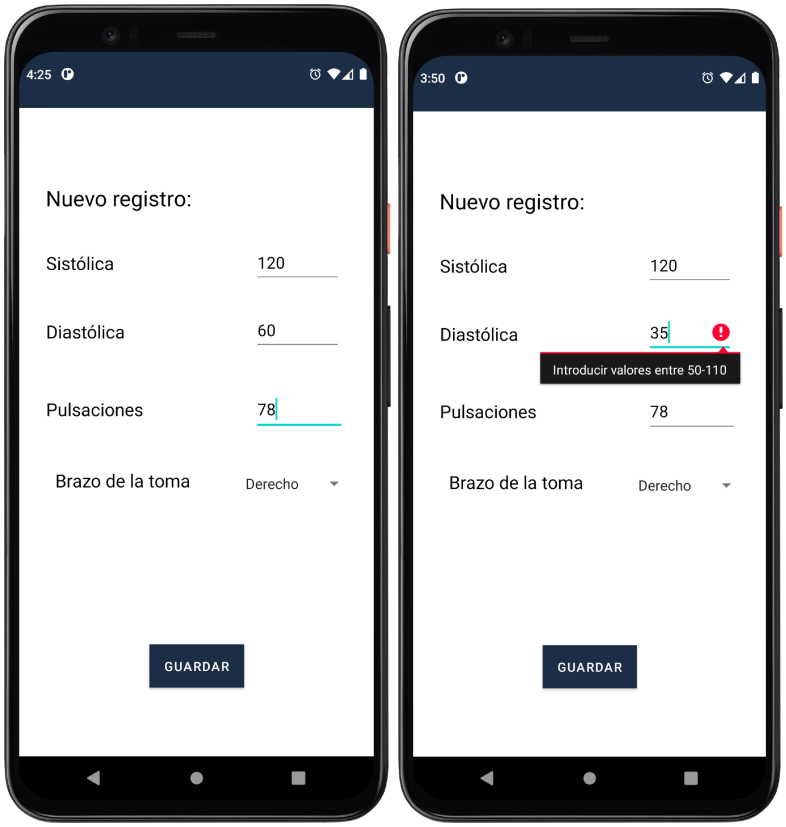


Imagen 21: Vista del registro de tensión arterial.

3.1.7 Medicación

Es la tercera de las cuatro pantallas principales, en ella se encuentran implementadas dos funcionalidades: medicamentos y puntos.

Medicamentos

La primera pantalla que se nos muestra es la lista de medicamentos del usuario. Esos medicamentos se han tenido que introducir previamente. Como podemos ver en la imagen 19*, se* nos muestra el nombre de la medicación, cuándo y qué cantidad tomar. Para eliminar un medicamento de la lista simplemente hay que pulsar el icono de la papelera que aparece en la tarjeta que contiene la medicación elegida a eliminar y pulsar en “SI”.

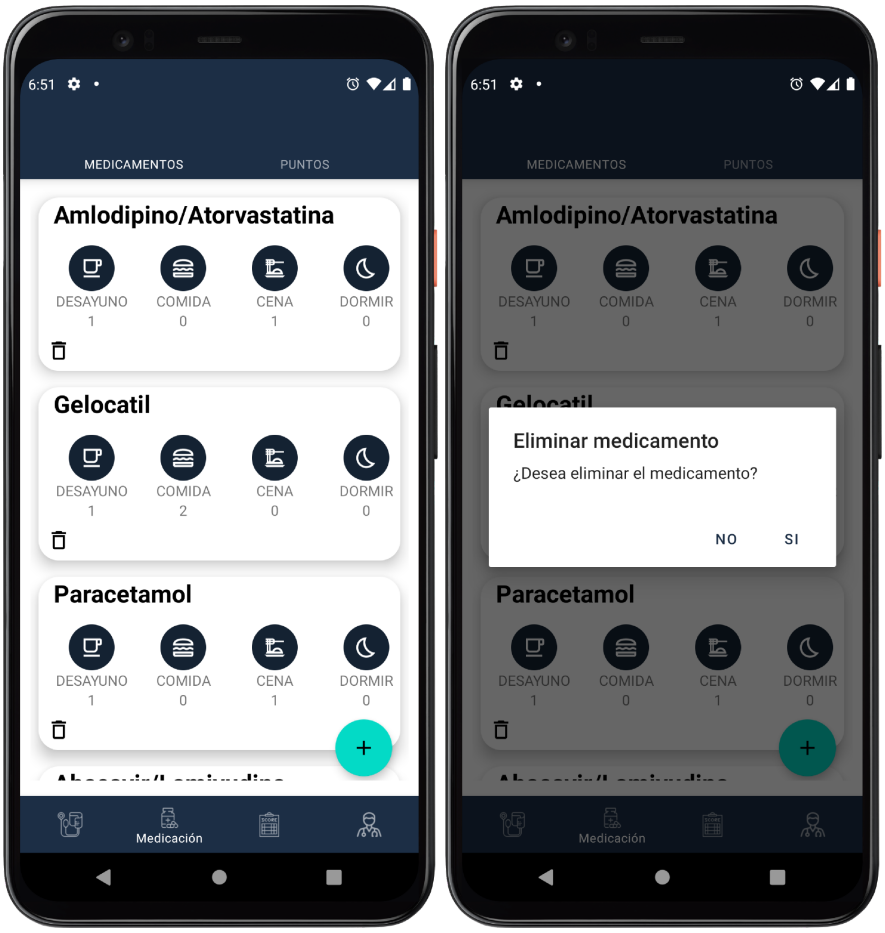


Imagen 22: Vista de posologías.

Para añadir un nuevo medicamento a la lista hay que pulsar al botón de + que se encuentra en la parte inferior derecha. Se abrirá el formulario de registro de nuevos medicamentos junto a su posología como muestra la imagen 20.

Imagen 23: Vista de posologías.

El nombre del medicamento se puede añadir de dos formas distintas. Con el nombre en que el usuario le sea más fácil recordar la medicación, o si lo prefiere puede introducir el nombre o una parte del nombre y darle a la lupa. Cuando se le da a la lupa, se lanza una consulta a la API REST de la Agencia Española de Medicamentos y Productos Sanitarios (AEMPS) [21] que nos devuelve una lista con el nombre introducido o con los medicamentos que contengan las letras introducidas.

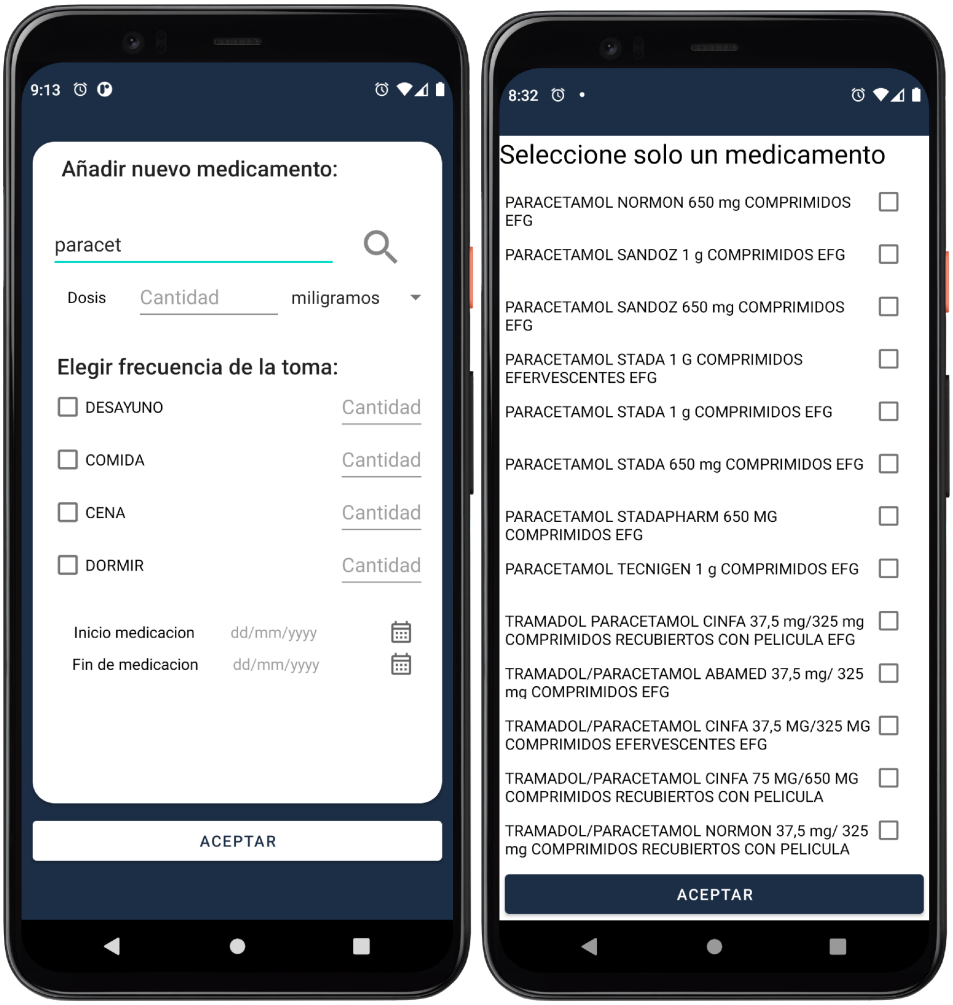


Imagen 24: Vista de añadir medicamento.

Una vez introducido el medicamento se tiene que marcar la frecuencia de la toma, la fecha de inicio y de fin y pulsar aceptar.

Si queremos ver la información más detallada de una medicación de la que nos muestra la vista de la imagen 22, tendremos que pulsar sobre la mediación de la que queremos ver más en detalle.

A continuación se nos mostrará toda la información y tendremos la posibilidad de modificar los valores de la frecuencia de la toma. Para actualizarla, simplemente se tendrá que pulsar el botón de aceptar una vez cambiado los parámetros.

Imagen 25: Vista de detalle de un medicamento. Permite editar valores previamente introducidos.

Puntuación

La función de la pantalla de puntos es la forma de motivar la adherencia al tratamiento farmacológico tal y como hemos comentado anteriormente en el Capítulo 1: Introducción. Este sistema debería suponer un pequeño incentivo a modo de competición, de modo que mensualmente se renueva la clasificación y motive al paciente a cumplir con su pauta de medicación.

El usuario debe acceder a la pestaña de puntos y marcar aquella mediación que ha tomado. Dependiendo del número marcado se le adjudicarán los puntos.

Porcentaje de las tomas	Puntuación recibida
100%	100 puntos
75%	75 puntos
25%	25 puntos
0%	0 puntos

Tabla 2: Asignación de puntos.

Como indica la tabla, la puntuación obtenida por el paciente al final del día será como máximo de 100 puntos. Estos puntos sólo se le darán a aquel paciente que haya tomado todas sus medicinas ese día. Para que el sistema sea lo más justo posible para aquellos pacientes que toman pocas medicinas en comparación a aquellos que toman más, la puntuación se asignará en función al porcentaje, y no al número de medicinas tomadas. De esta forma, el paciente que haya tomado por ejemplo un 80% de las medicinas correspondientes a ese día, obtendrá 80 puntos.

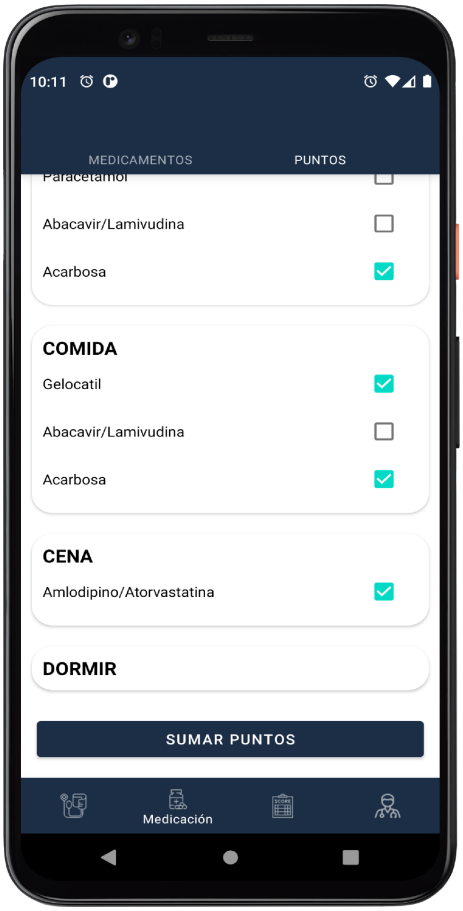


Imagen 26: Vista de agregar puntuación.

3.1.8 Chat de mensajería

El chat de mensajería es la última de las cuatro pantallas principales, en ella se podrá interactuar con el profesional médico asignado.

Imagen 27: Vista del chat de mensajería.

3.2 Aplicación web

3.2.1 Inicio de sesión

En la imagen 28 se muestra la pantalla de inicio de sesión del médico en la aplicación en la que debe introducir su nombre de usuario y contraseña.

En caso de que se quiera añadir un nuevo médico a la aplicación, debe acceder a la url de autorización de Django en la que se rellena el formulario correspondiente.

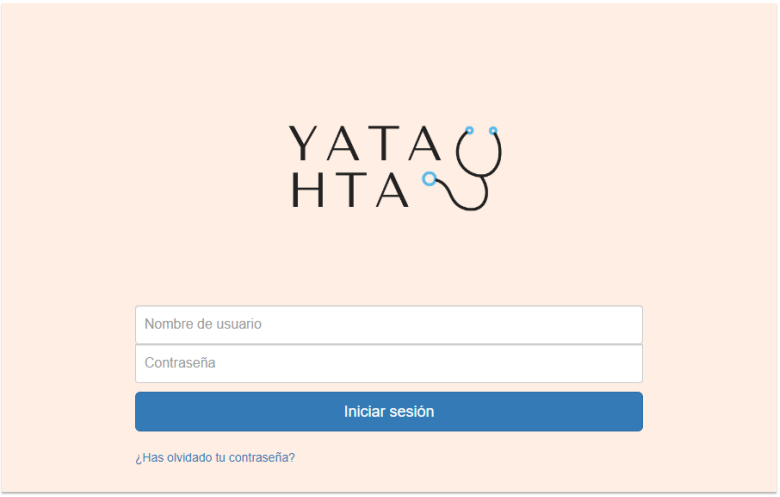


Imagen 28: Vista de inicio de sesión.

3.2.2 Página principal

Página inicial de la aplicación, en la que el médico podrá ver los 15 pacientes mejor posicionados en la clasificación general, estén o no asignados a él.

También tiene datos de la magnitud actual de la aplicación, con el número total de usuarios, el número de medicamentos almacenados en la base de datos, el número total de posologías, el número de tomas de tensión de todos los pacientes y el número de mensajes intercambiados.

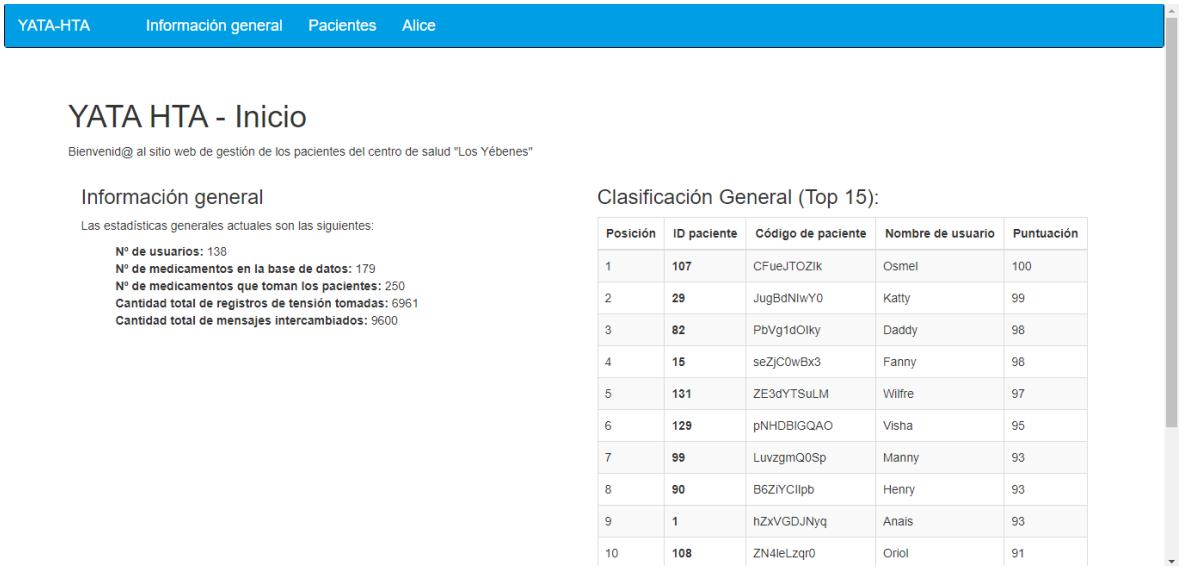


Imagen 29: Página principal de la aplicación web.

3.2.3 Listado de pacientes

En esta pantalla el médico tiene acceso a un listado de todos los pacientes asignados a él. Puede elegir mediante los botones de cada fila la información del paciente que quiera ver en relación a su información tanto personal como los registros de tensión arterial, los medicamentos que toma el paciente y la conversación que mantienen a través de la aplicación.

Los pacientes están ordenados según la puntuación de cada uno de mayor a menor.

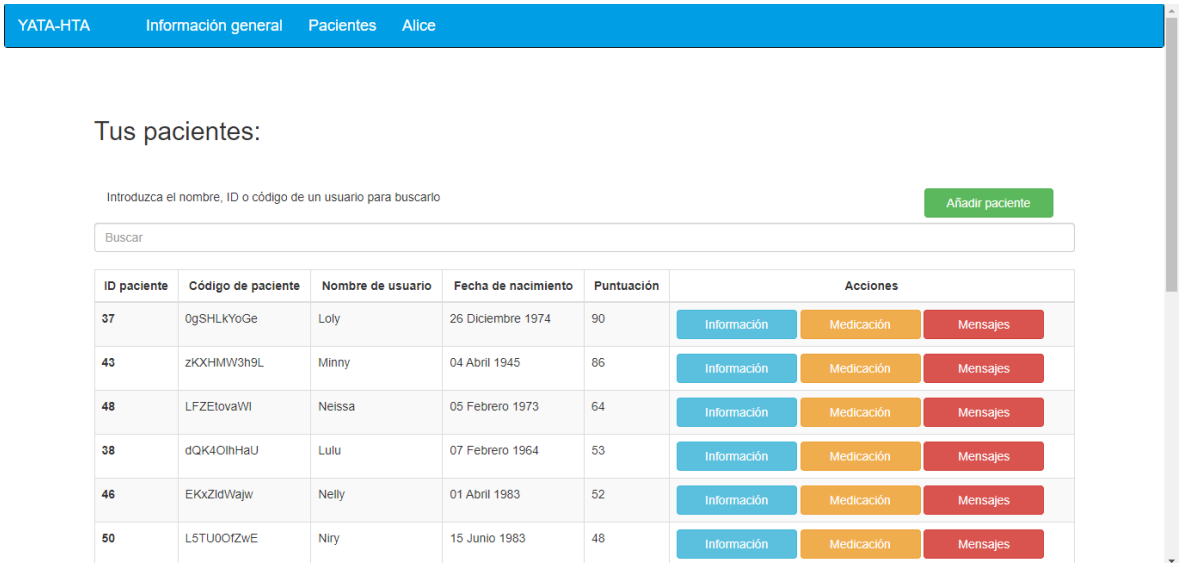


Imagen 30: Listado de pacientes.

Al darle al botón “Añadir paciente” en la parte superior derecha de la pantalla, se redirige al médico a una nueva página en la que se detalla la información que debe facilitar

al paciente para que pueda darse de alta, es decir, su código de médico (que se corresponde con su ID) y el código de paciente. Este último se genera aleatoriamente de forma que no coincida con ninguno ya existente y se introduce en la base de datos para que quede a disposición del paciente. De este modo, se asegura que un paciente no pueda acceder con un código inventado, si no que ha tenido que ser previamente generado por el médico, y se le tiene que haber facilitado.

Imagen 31: Generación de código de paciente.

En caso de que la aplicación tenga un tamaño relativamente alto en términos de pacientes asociados a cada médico, se puede hacer uso de la barra localizada encima de la tabla de pacientes para filtrar de forma ágil un paciente, en función de cualquiera de las categorías que se detallan en la tabla. En la imagen 32 de ejemplo este proceso se realiza a través del nombre de usuario del paciente.

Imagen 32: Filtrado de pacientes en el listado.

3.2.4 Información del paciente

Si un doctor desea ver la información de un paciente, se encontrará en esta pantalla tras haber pulsado el botón “Información” en la tabla.

En la vista se detalla la información personal del paciente: fecha de nacimiento, el doctor asociado y los puntos que tiene actualmente. Para facilitar la navegación del médico una vez ha comenzado a ver información de un paciente determinado, se puede hacer uso de los dos botones situados en la esquina superior derecha, que permiten acceder a la medicación de ese paciente y a la conversación entre ambos usuarios.

En cuanto a la información relacionada con la tensión arterial, se descompone en dos elementos diferenciados. El primero es el gráfico que permite hacer un seguimiento a lo largo del tiempo de la presión arterial y las pulsaciones en cada toma.

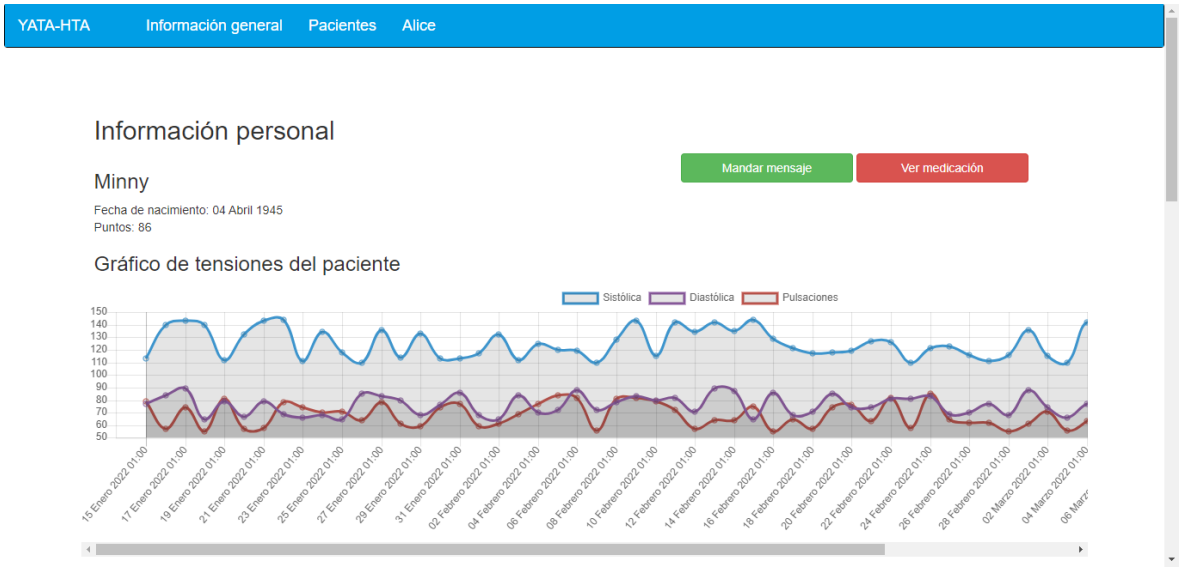


Imagen 33: Información personal y arterial del paciente.

El doctor puede hacer un filtrado de los datos que se reflejarán haciendo uso de los botones que se ven en el gráfico. Puede elegir ver o no la presión diastólica, la sistólica o las pulsaciones para hacer más sencillo un seguimiento más o menos exhaustivo.

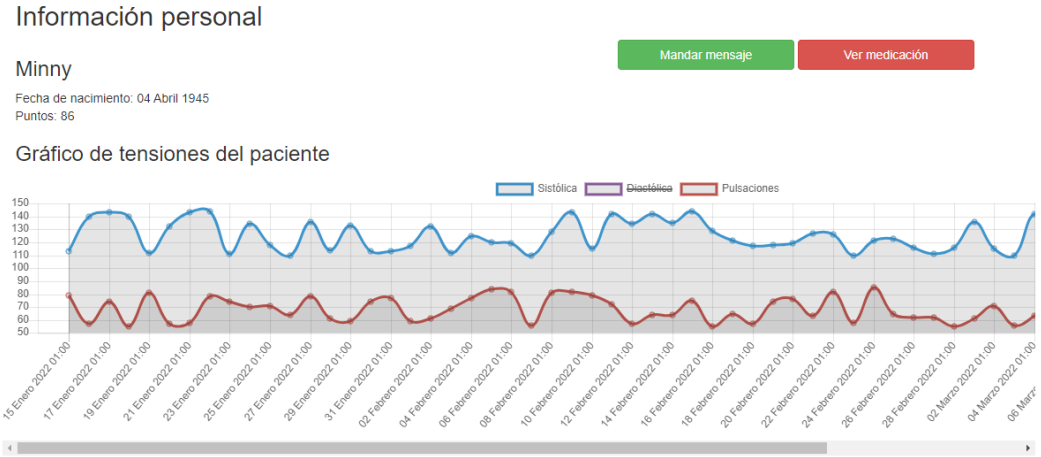


Imagen 34: Filtrado y recorrido de datos en gráfica.

Para facilitar la obtención de los valores en cada toma, en caso de situar el ratón encima de un punto de la gráfica, aparece un cartel con el valor de dicho punto en la fecha determinada.

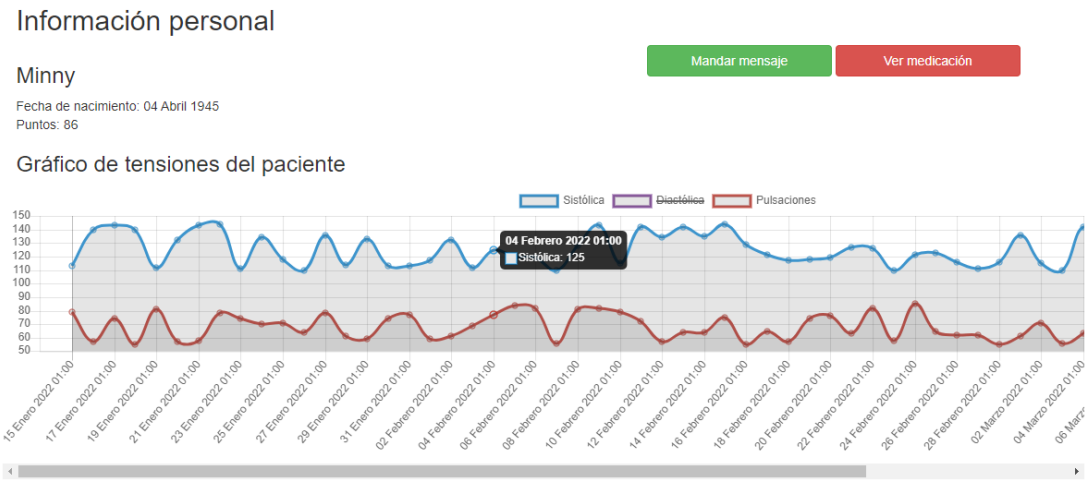


Imagen 35: Obtención de valores concretos en gráfica.

El segundo elemento de información relativa a la tensión arterial del paciente es la tabla en la que se detalla cronológicamente todas las presiones del paciente, el número de pulsaciones por minuto y el brazo en el que la toma fue realizada. En la tabla se ven únicamente los valores de las últimas 30 tomas realizadas.

Tabla de tomas de tensión

Fecha	Sistólica	Diastólica	puls/min	Brazo de la toma
16 Marzo 2022 01:00	118	81	66	Izquierdo
15 Marzo 2022 01:00	144	71	70	Derecho
14 Marzo 2022 01:00	120	78	62	Derecho
13 Marzo 2022 01:00	129	79	59	Derecho
12 Marzo 2022 01:00	110	76	74	Derecho
11 Marzo 2022 01:00	136	69	56	Derecho
10 Marzo 2022 01:00	113	85	80	Derecho
09 Marzo 2022 01:00	119	78	63	Izquierdo
08 Marzo 2022 01:00	140	73	74	Derecho
07 Marzo 2022 01:00	131	69	56	Derecho
06 Marzo 2022 01:00	136	87	85	Izquierdo
05 Marzo 2022 01:00	123	78	77	Izquierdo

Imagen 36: Listado de tomas de tensión del paciente.

3.2.5 Medicación del paciente

En caso de haber pulsado el botón “Medicación” en la tabla de pacientes, aparecerá esta pantalla en la que se detalla la pauta de medicación de un paciente. Del mismo modo que en la pestaña de información del paciente, el doctor puede hacer uso de los dos

botones situados en la esquina superior derecha para acceder a la información de tensión arterial del paciente o al chat entre ellos.

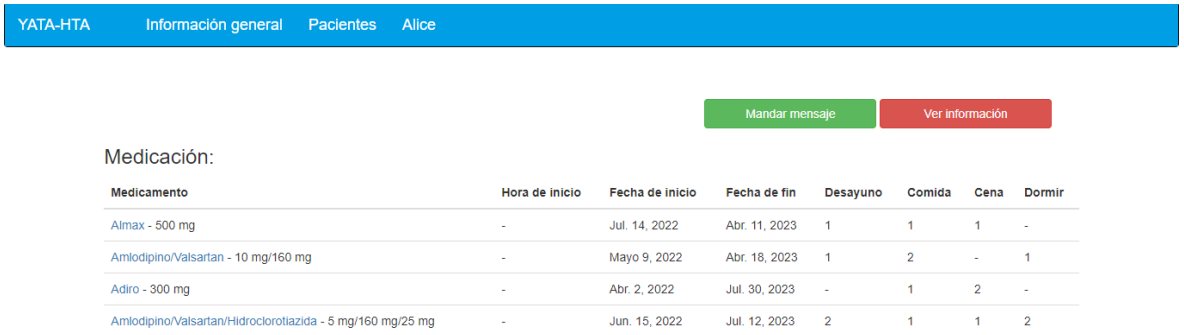


Imagen 37: Información de la posología del paciente.

Si en el momento en que el paciente introdujo el medicamento que debía tomar, lo hizo con los datos de la API que se facilitan en la aplicación móvil, se podrá acceder a la siguiente ficha de medicamento pulsando en el nombre del medicamento de la tabla. En ella se detalla el nombre completo de dicho medicamento, el tamaño de la dosis, y, en caso de haberla, una imagen del mismo.

Imagen 38: Información de un medicamento concreto.

3.2.6 Chat con el paciente

Si el médico pulsa el botón de “Mensajes” del paciente deseado, accederá al chat en el que podrán intercambiar mensajes entre ellos. Al igual que en la medicación y en la información, el médico puede acceder de forma más dinámica a dichos apartados (para el usuario en cuestión) mediante los dos botones de la esquina superior derecha. Los botones situados en la parte inferior derecha sirven para enviar el mensaje escrito, y para recargar la página de forma que se vean los mensajes recibidos.

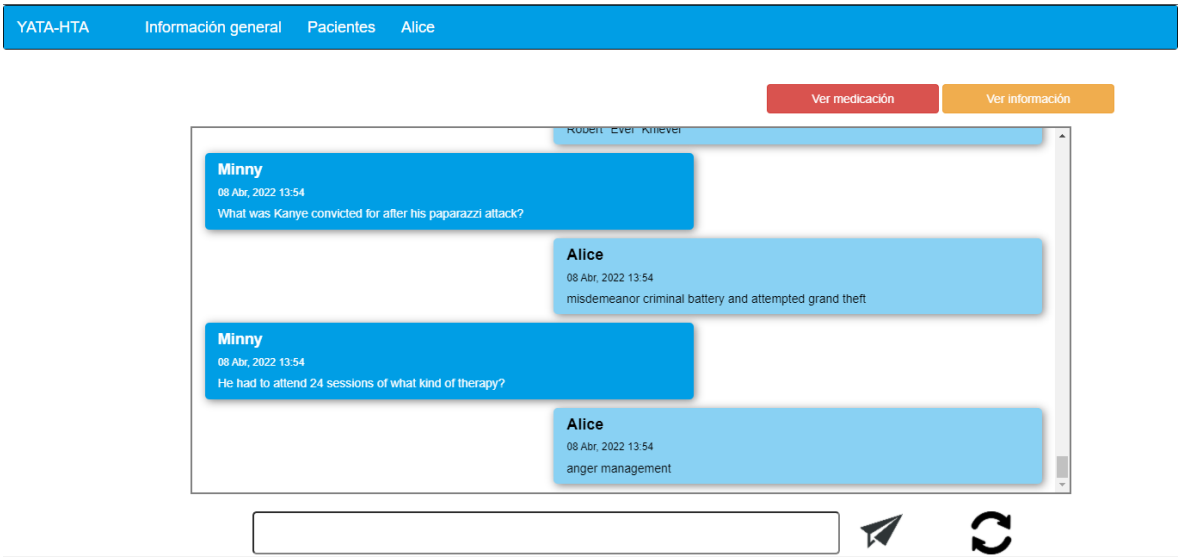


Imagen 39: Chat interactivo entre médico y paciente.

3.2.7 Gestión de médicos

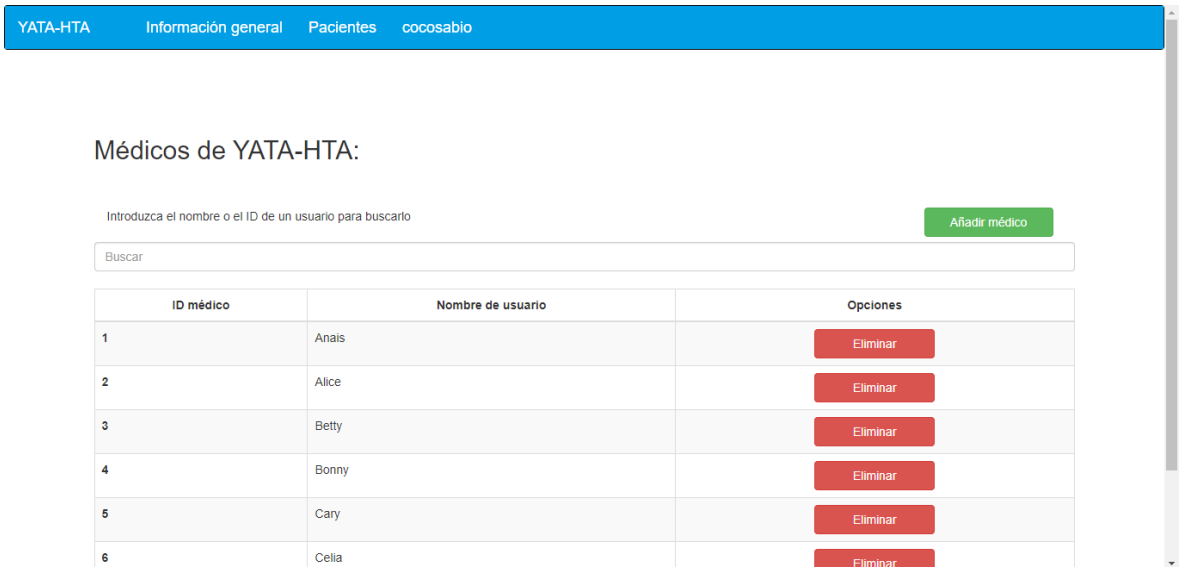


Imagen 40: Vista del listado de médicos de la aplicación.

Para facilitar y dinamizar la gestión de la plataforma, se ha de designar un médico que empeñará el papel de administrador de la aplicación. Como tal, además de tener acceso a todas las funcionalidades disponibles para los médicos del proyecto, tiene acceso a un listado en el que se detalla el nombre de usuario e ID (que coincide con el código de médico) de cada uno de sus compañeros.

Dado que se pretende que esta aplicación web sea de uso exclusivo para médicos involucrados en el proyecto, se ha descartado la posibilidad de que se den de alta a través de una página accesible desde la pantalla de Inicio de sesión, ya que cualquiera que la encontrara podría darse de alta como médico.

Imagen 41: Imagen del formulario para añadir un nuevo médico.

Para la creación de un nuevo médico, se ha diseñado el formulario de la imagen 41. Se puede acceder a él a través del botón ubicado en la parte superior derecha “Añadir médico”. Una vez se asigna el nombre de usuario, email y se introduce dos veces la contraseña del nuevo médico, queda un nuevo perfil listo para ser usado por parte del médico en cuestión.

Obviamente, en caso de que el médico recién creado quiera mantener su privacidad y asignar una contraseña de su elección, puede hacerlo gracias a la pestaña accesible a partir del apartado correspondiente en el menú de la parte superior de la aplicación web con texto “Cambiar contraseña”.

En caso de que el administrador deba borrar un médico de la plataforma, lo hará mediante el botón correspondiente a ese doctor en la tabla de la imagen 40. Una vez pulsado dicho botón, se redirigirá al administrador a la página de confirmación de borrado.

Imagen 42: Página de confirmación de borrado de un médico.

Si el administrador confirma el borrado con el botón de la zona inferior de color blanco con mensaje “Sí, borrar”, el médico quedará borrado de la aplicación y se redirigirá de nuevo a la página de Gestión de médicos, que aparecerá actualizada con los médicos que participan en el proyecto. En caso contrario, el administrador pulsará el botón de color rojo con mensaje “Cancelar” y simplemente volverá a la página anterior sin que se haya efectuado ningún cambio en la base de datos.

3.2.8. Cambiar contraseña

En caso de que un médico quiera cambiar su contraseña por temas de seguridad o privacidad, puede acceder al formulario pertinente gracias al botón “Cambiar contraseña” del menú superior, que se hace visible al situar el cursor encima del nombre de usuario cuya sesión está activa.

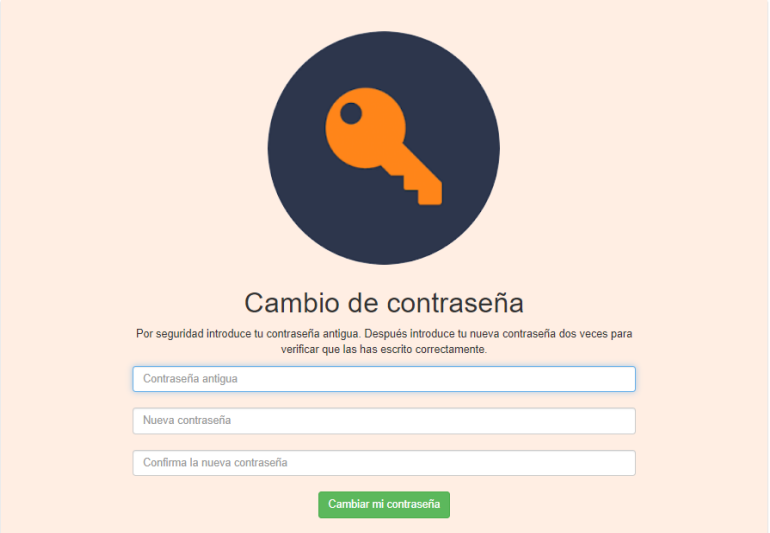


Imagen 43: Formulario del cambio de contraseña.

Para que el cambio de contraseña se haga efectivo debe introducir la contraseña que venía usando hasta ese momento, seguida de la nueva contraseña (dos veces para mayor seguridad). Una vez terminado este proceso, el cambio de contraseña se hará efectivo mientras no haya habido error en alguno de los pasos anteriormente descritos. Seremos conscientes de ello ya que llegaremos a la página de confirmación de la imagen 44.

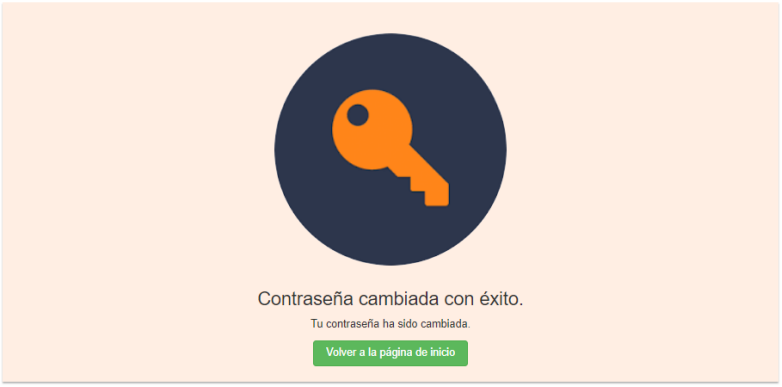


Imagen 44: Confirmación de cambio de contraseña.

3.2.9. Olvidé mi contraseña

Con el objetivo de que los médicos puedan recuperar su contraseña en caso de que la hayan olvidado, se ha facilitado un proceso como este al que se accede a través del enlace situado en la parte inferior de la pantalla de Inicio de sesión en el texto “¿Has olvidado tu contraseña?”.

Si el usuario pulsa en este enlace llegará a esta vista en la que se requiere que introduzca el correo electrónico con el que se dió de alta en la aplicación.

Imagen 45: Solicitud de correo electrónico para recuperar la contraseña.

Una vez introducido el correo correctamente, deberá pulsar en el botón con texto “Restablecer contraseña”. Al hacerlo llegará a esta página en la que se le hace saber que recibirá en el correo electrónico correspondiente un enlace para que pueda reasignarse una contraseña.

Imagen 46: Aviso de que el correo ha sido enviado.

Una vez hemos abierto el correo y pulsado en el enlace que contiene, llegamos a esta página en la que podremos elegir una nueva contraseña para la cuenta.

Imagen 47: Selección de nueva contraseña.

Efectuado el cambio de contraseña, se llega a esta página de confirmación. Además, se puede hacer uso del enlace que aparece en el texto en la parte inferior de la pantalla para ir de nuevo a la página de Inicio de sesión.

Esta vez, con la nueva contraseña, se puede acceder a la plataforma.

Imagen 48: Confirmación de que el cambio se ha realizado satisfactoriamente

Capítulo 4: *Discusión*

En este apartado vamos a discutir y justificar más detalladamente el porqué de las decisiones que hemos tomado en el desarrollo de nuestro proyecto. En el primer apartado abordamos las tecnologías empleadas para el desarrollo del proyecto. En segundo lugar tratamos la implementación del proyecto y explicamos las decisiones tomadas.

4.1 Herramientas y tecnologías utilizadas

4.1.1 Android y Java

Para el desarrollo de la aplicación móvil rápidamente nos decantamos por Android como sistema operativo, en lugar de iOS, y el lenguaje de programación Java. Esta decisión la basamos tanto en datos objetivos como que el sistema operativo Android tiene una cuota de mercado de más del 50% en España, muy por encima del resto de sistemas operativos [22], y en concreto para *smartphones*, su cuota supera el 84% a final de 2021 según el portal web de *Statista* [23]. Esto posibilita que la mayor parte de la población diana de nuestro cliente utilice este sistema operativo en sus teléfonos móviles.

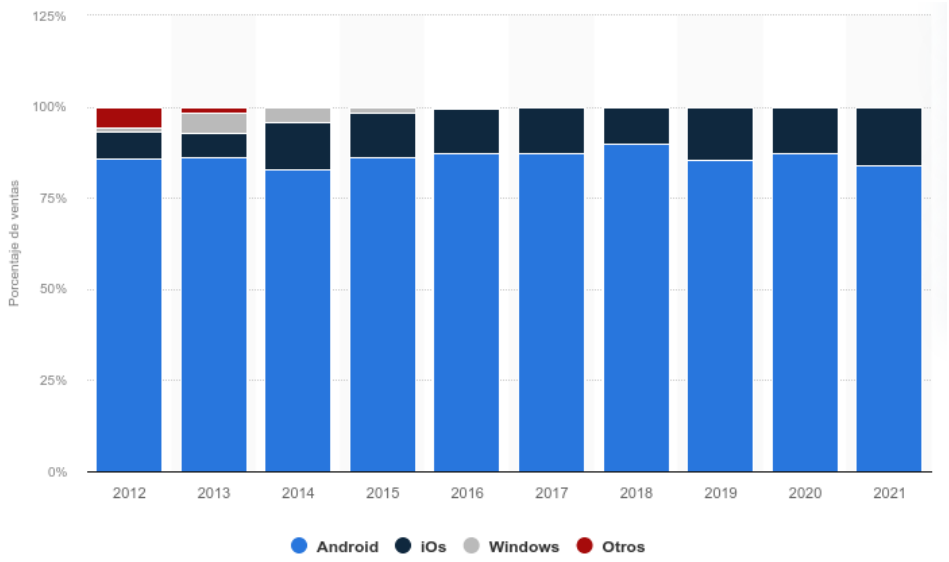


Imagen 49: Cuota de mercado de los principales sistemas operativos en España. Fuente: Statista.

Desde un punto de vista más práctico, no teníamos muchos conocimientos en cuanto a programación en dispositivos móviles, pero sí de Java como lenguaje de programación, y este se puede emplear para desarrollar en Android. Adicionalmente, tanto Java como lenguaje de programación como Android como sistema operativo tienen versiones de código abierto [24], [25], lo que posibilita utilizar estas tecnologías sin costes adicionales. Estos motivos decantaron la balanza a favor de Android en lugar de iOS.

Para el desarrollo de la app optamos por Android Studio [26] ya que es el entorno de desarrollo oficial para Android.

4.1.2 Django

Para la otra vertiente de nuestro proyecto, la aplicación web que va a usar nuestro cliente directo (los médicos), nos decidimos por el *framework* de desarrollo web *Django* [27].

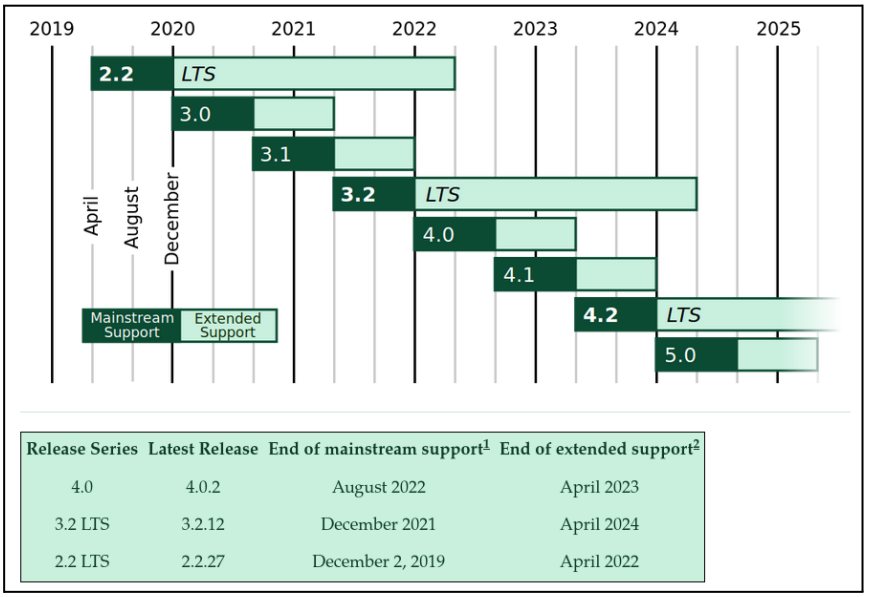
Para ello es fundamental tener previamente una base adecuada en Python, lenguaje del que dos miembros del equipo ya teníamos conocimientos previos. Python, tal y como muestra en los últimos años el índice Tiobe [28], es un lenguaje de programación muy popular entre la comunidad de desarrolladores y muy demandado en la industria.

Tal y como lo describen en su sitio web oficial: "*Django es un marco de desarrollo web de alto nivel escrito en Python que fomenta un desarrollo rápido y un diseño limpio y pragmático. Creado por desarrolladores experimentados, se ocupa de gran parte de las molestias del desarrollo web, por lo que el desarrollador se puede concentrar en escribir su aplicación sin necesidad de reinventar la rueda. Es gratis y de código abierto.*" [27].

Por otra parte, Django es un framework ampliamente empleado y en continuo crecimiento, de ahí que sea empleado en aplicaciones webs con millones de visitas como *Instagram*, *Pinterest* o *Mozilla* [29].

En este caso la decisión fue también más práctica, lo conocíamos 2 de los integrantes del equipo, es *open source* y aporta muchas facilidades para un desarrollo rápido y seguro de aplicaciones web.

Al principio tuvimos que decidir de forma razonada qué versión de Django emplear. Optamos por desarrollar con la actual versión de soporte prolongado (LTS - Long Term Support) [30], ya que establece una base sólida de código sobre la que poder trabajar. Tal y como aparece en su documentación, adjuntamos el gráfico que muestra el ciclo de desarrollo del framework:



Release Series	Latest Release	End of mainstream support[1]	End of extended support[2]
4.0	4.0.2	August 2022	April 2023
3.2 LTS	3.2.12	December 2021	April 2024
2.2 LTS	2.2.27	December 2, 2019	April 2022

Imagen 50: Política de soporte de versiones de Django. Fuente: https://www.djangoproject.com/download/.

Django aporta muchas facilidades fundamentales para el desarrollo web, entre otras:

- La *Asignación Objeto-Relacional (ORM - Object Relational Mapping)* [31], es una facilidad que aporta Django (al igual que otros *frameworks* de desarrollo web) que facilita muchísimo el desacople de la capa de negocio de la aplicación con respecto a la capa de persistencia. En este caso, como Sistema Gestor de Base de Datos SQL [32], hemos empleado *PostgreSQL* [33].
- Gestión de sesiones web.
- Gestión de permisos para distintos tipos de usuarios de nuestra aplicación.
- Trabajo con formularios web.
- *Testing* de la aplicación web, fundamentalmente con test unitarios [34].

Aparte de la base de código que aporta Django, hemos empleado algunos módulos extra que complementan este framework para aportar ciertas funcionalidades.

Django REST framework

Para dar soporte a ambas aplicaciones pensamos que lo mejor era desarrollar una API REST [35], [36] como estilo arquitectónico [37]. Este estilo arquitectónico para el desarrollo de aplicaciones basadas en la infraestructura de red de Internet es ampliamente utilizado en la industria, y es el estándar de facto actual. Por lo tanto es una tecnología imprescindible para aprender y aplicar actualmente. Como tecnología de soporte, y puesto que habíamos decidido usar Django para desarrollar la aplicación web, hemos usado *Django REST framework (DRF)* [38].

DRF aporta muchas facilidades para el desarrollo rápido y eficaz de APIs, entre otras características aporta:

- Una API navegable desde el navegador web, lo que facilita mucho su exploración.
- Diferentes políticas de autenticación de los usuarios.
- Serialización y deserialización de datos tanto para sistemas basados en ORM como en no ORM.
- Posibilidad de personalización de las vistas devueltas muy extensa.

4.1.3 PostgreSQL

Es una base de datos relacional de código abierto con más de 30 años de desarrollo, lo que le ha permitido ganarse la fama de robusta, fiable y con un gran rendimiento [33]. Durante el Grado, en la asignatura de “Bases de datos”, hemos usado el Sistema Gestor de Bases de Datos (SGBD) de Oracle, pero esta es de pago.

El hecho de que PostgreSQL es software libre y que cuando comenzamos a desarrollar con *Django* la mayor parte de tutoriales que hemos seguido utilizan este SGBD, junto con el hecho de que las Plataformas como Servicios (*PaaS,* de sus siglas en inglés) utilizaran en sus versiones gratuitas *PostgreSQL* como SGBD, hizo que finalmente nos decantásemos por esta opción.

4.1.4 Docker

Docker permite “empaquetar” en un contenedor la aplicación que se está desarrollando junto con todas sus dependencias, de esta forma nos aseguramos que siempre iba a

funcionar tal y como la habíamos desarrollado en nuestros equipos localmente. Además facilita el compartir la imagen de la aplicación con otros desarrolladores y dentro del equipo, así como su despliegue. Un artículo más extenso y descriptivo de esta tecnología es el de *IBM Cloud Education* [39].

Como hemos comentado en el Capítulo 1: Introducción, en un principio tuvimos ciertos problemas con las versiones de los Entornos Integrados de Desarrollo (*IDE*, de sus siglas en inglés), las propias versiones de los lenguajes o librerías a utilizar, la instalación y funcionamiento de la base datos, etc. Tampoco teníamos claro cómo íbamos a desplegar la aplicación en un entorno de producción, por lo que tuvimos que investigar y empezar a hacer distintas pruebas.

Inicialmente empezamos usando la plataforma que ofrece *Heroku* [40], aunque esta facilita mucho el despliegue de aplicaciones tanto para desarrollo como para producción, la versión gratuita del plan que ofrece se queda corta para lo que estábamos probando. La versión gratuita de la plataforma ofrece un almacenamiento de 8.000 registros en el SGBD, esto se nos quedó muy corto en cuanto empezamos a poblar la base de datos de pruebas con pacientes, medicamentos, registros de tomas de tensión y mensajes entre pacientes y médicos.

Como uno de los integrantes del grupo ya tenía contratado un Servidor Privado Virtual (*VPS*, de sus siglas en inglés) y ciertos conocimientos de la tecnología de *Docker* [41] nos decidimos a utilizar esta tecnología.

Con *Docker* hemos empleado una configuración para el entorno de desarrollo y otra para un hipotético despliegue en producción de la aplicación. Hemos usado un contenedor para la aplicación web y la API y otro contenedor para el SGBD con *PostgreSQL*. Además, al principio teníamos desplegada la base de datos centralizada en un contenedor individual con PostgreSQL. El hecho de haber usado Docker nos ha permitido emplear exactamente las mismas dependencias, versiones de paquetes, módulos, librerías, etc. a todos los miembros del equipo, sin mayor conflicto. Creemos que ha acelerado y facilitado el desarrollo de la aplicación.

4.2 Implementación

En este apartado vamos a detallar el diseño arquitectónico de nuestra aplicación, así como el diseño detallado de los componentes.

4.2.1 Arquitectura de la aplicación

Desde el inicio el proyecto de TFG exigía el desarrollo de una aplicación para dispositivos móviles, pero en el planteamiento inicial ya pudimos distinguir claramente que íbamos a tener dos tipos de usuarios bien diferenciados, por un lado los pacientes del centro de salud que iban a entrar en el proyecto *Yébenes Adherencia al Tratamiento Antihipertensivo (YATA)* y, por otro lado, el personal facultativo y sanitario del centro de salud. La aplicación móvil se adapta perfectamente a las exigencias de los pacientes, pero decidimos que para los doctores se ajustaría mejor una aplicación web.

Por lo tanto, como primera aproximación a la arquitectura de la aplicación podemos decir que hemos seguido una arquitectura distribuida basada en un estilo arquitectónico de cliente-servidor.

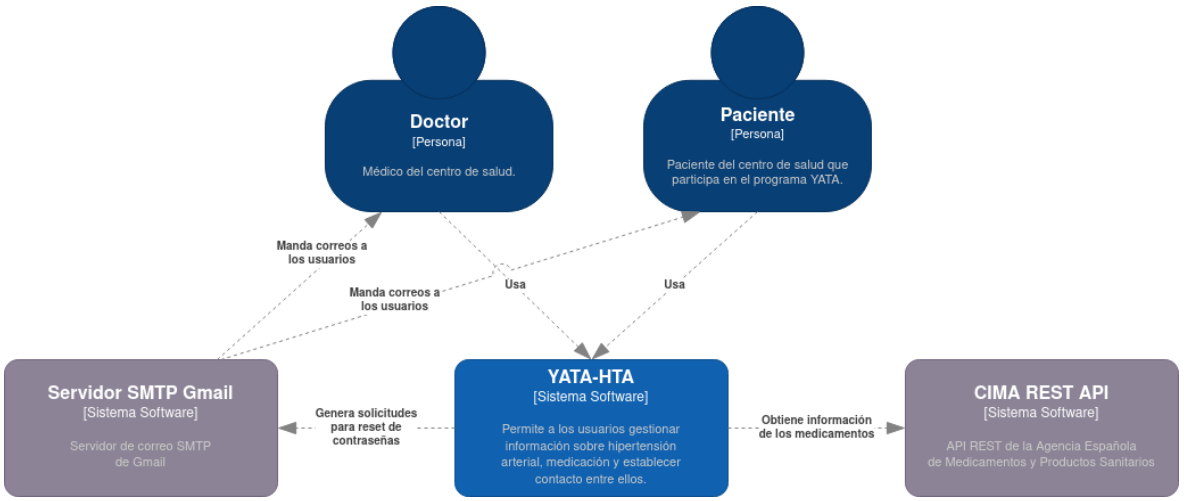


Imagen 51: Diagrama correspondiente con el contexto del sistema para la aplicación YATA-HTA.

Como se puede ver en este esquema de nivel 1 de C4 [42], nuestros usuarios serían los doctores o personal sanitario del centro de salud que dirigen el proyecto y los pacientes del centro de salud. Estos usuarios van a poder gestionar y acceder a información relativa a los registros de tensión arterial, la medicación y van a poder intercambiar mensajes. Para cada tipo de usuario tenemos una aplicación "cliente": una aplicación web en el caso de los médicos y una aplicación Android para los pacientes. A su vez, tendremos alojada la aplicación YATA-HTA en un servidor, de ahí este estilo arquitectónico tan común. Adicionalmente, tal y como se ha descrito anteriormente en el apartado 3.1 Aplicación móvil - Medicamentos del capítulo 3, hacemos uso de la API REST que proporciona la Agencia Española de Medicamentos y Productos Sanitarios (AEMPS). Finalmente, también hacemos uso del servidor SMTP que proporciona Google con las cuentas de correo de Gmail para mandar los *emails* de recuperación de contraseñas de la aplicación.

Centrándonos en la organización interna de nuestra aplicación podemos distinguir al menos 4 capas bien diferenciadas: presentación, negocio, persistencia y base de datos.

Una primera capa de presentación estaría formada por las vistas web de nuestra aplicación, servidas mediante las *templates* de Django y las *activities* de la aplicación Android. Esta capa es con la que interactúa el usuario desde su navegador web o su *smartphone* y es la que manda las peticiones a la capa inmediatamente inferior, que se corresponde con la capa de negocio. Como ya se ha mencionado anteriormente en el apartado 4.1 Herramientas y tecnologías utilizadas de este mismo capítulo, en esta capa hacemos uso de tecnología HTML, CSS (*Bootstrap*) y JavaScript, tanto puro como con AJAX.

En la capa de negocio tenemos la propia aplicación web desarrollada con Django que recibe las peticiones desde el navegador y una interfaz REST desarrollada con *Django-REST-framework* con la que interactúa la aplicación móvil Android. Estas peticiones las recepciona Django a través del módulo *urls.py*, que hace las veces de *dispatcher* y redirige la petición a la *view* correspondiente (en realidad es una función *callback* para una URL particular), las *views* de Django se encuentran en el módulo *views.py* y se encargan de recuperar la información que se va a devolver a la petición que se acaba de recibir. Este mecanismo propio de Django, en el que funcionan conjuntamente *urls.py* y *views.py* haría

las veces de un controlador, tal y como aparece en la sección de Preguntas Frecuentes de Django [43].

Inmediatamente por debajo encontramos la capa de persistencia. En esta capa intervienen los modelos de Django, descritos en el módulo *models.py*, que describen qué información vamos a guardar y cómo se va a comportar. Normalmente, cada modelo es una clase Python que se corresponde con una tabla en la base de datos, y cada atributo es una columna en la tabla. Pero para poder interactuar con los distintos sistemas de bases de datos Django aporta una capa intermedia, entre los modelos y la propia base de datos, de Mapeo Objeto-Relacional (*ORM*, de sus siglas en inglés) [44]. Este *ORM* es una abstracción que permite ejecutar operaciones *CRUD* [45] desde nuestros modelos, mediante una llamada a la API del *ORM*, independientemente de la base de datos subyacente.

En un diseño arquitectónico de tipo Modelo-Vista-Controlador (MVC) [46], la Vista (V) es la primera capa de nuestra aplicación, el Controlador (C) sería la segunda capa y el Modelo (M) finalmente sería esta tercera capa de la aplicación. Aunque en Django no sigue exactamente este modelo, pero es prácticamente igual y lo llaman *Model-View-Template (MVT)*. Siendo el *View* o Vista el Controlador, y *Template* o plantilla la Vista.

Finalmente, en la capa de base de datos hemos hecho uso de la tecnología aportada por PostgreSQL.

Imagen 52: Arquitectura en capas de la aplicación YATA-HTA.

Continuando con el modelo de diagramas de software descrito, ahora vamos a describir más detalladamente los distintos componentes que forman parte de nuestra aplicación.

Entrando en más detalle, el sistema software del diagrama anterior lo podemos dividir en varios contenedores básicos.

En primer lugar el contenedor propio de la aplicación web: éste es el encargado de servir las páginas web al navegador del cliente. Las distintas vistas web de la aplicación forman en sí mismas otro contenedor diferenciado dentro de la aplicación. Éstas proveen toda la funcionalidad de la aplicación a los usuarios de tipo “doctor” a través de su navegador web. Mediante estas vistas, el usuario con sus acciones ejecuta una serie de peticiones HTTP al servidor donde está la aplicación y este devuelve el contenido estático. Además, este contenedor también gestiona la lógica de reseteo de contraseñas de los usuarios en caso de estos la hayan olvidado, para el envío de correos electrónicos a los usuarios emplea el servidor SMTP de una cuenta de Gmail que tenemos asociada.

Por otra parte tenemos la aplicación Android que realiza peticiones HTTP a la API REST. Siguiendo este estilo arquitectónico podemos aprovechar las primitivas del protocolo HTTP para implementar operaciones CRUD [45] en nuestra aplicación. Esta interfaz REST es la que mapeamos con el contenedor que llamamos API REST de la aplicación.

Finalmente, a través de la lógica implementada por los modelos de Django, el contenedor de Aplicación Web y API REST, haciendo uso del API del ORM de Django, podemos comunicarnos con el último contenedor: el contenedor de Base de datos. Esto lo podemos ver en el siguiente diagrama:

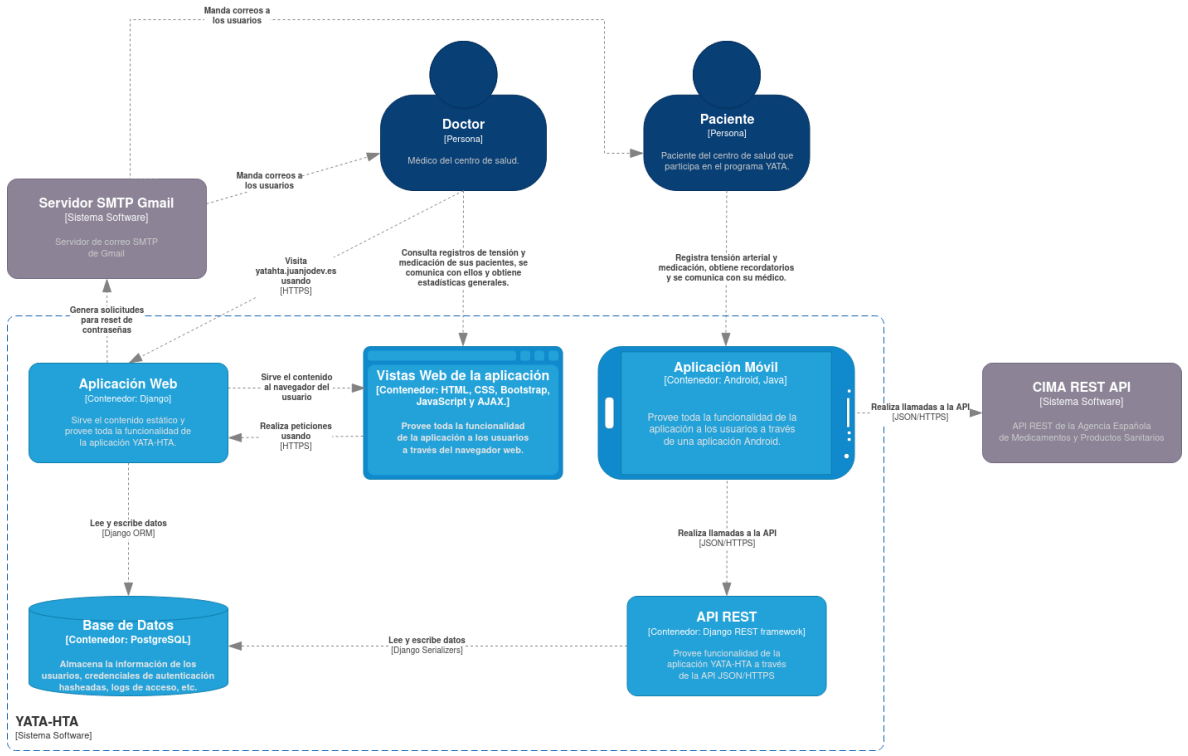


Imagen 53: Diagrama donde se muestran los contenedores del sistema software para la aplicación YATA-HTA.

Si empezamos a adentrarnos en el sistema software planteado en el diagrama anterior, vemos dos aplicaciones cliente, una Android y otra una aplicación web. Vamos a empezar por describir la primera.

Las aplicaciones Android, por lo general, están compuestas por múltiples componentes, entre los que se incluyen las actividades o *activities*. Por las características de los dispositivos móviles y la arquitectura del propio sistema operativo, es posible que los componentes de una aplicación se ejecuten individualmente y fuera de orden, e incluso que

el sistema operativo los destruya en cualquier momento tal y como refleja la documentación oficial [47]. Esto fuerza a los desarrolladores a crear componentes independientes y con bajo acoplamiento.

En la documentación oficial de Android se aconseja un modelo de arquitectura en capas, con al menos dos capas diferenciadas:

- Capa de Interfaz de Usuario o de presentación que muestra los datos en pantalla.
- Capa de datos o de negocio que contenga la lógica de negocio y exponga los datos de la aplicación.

En nuestro caso únicamente tenemos una capa, la de presentación. Esto es así porque tan solo buscamos que el tipo de usuario “paciente” pueda ingresar datos y visualizar los datos ingresados. Para este fin nos basta con la capa de presentación, compuesta por *activities* de Android. No tenemos ningún tipo de lógica de negocio implementada en la aplicación móvil.

Desde esta capa construimos la interfaz de usuario que se muestra en la aplicación móvil, la lógica necesaria para realizar peticiones HTTP a la API REST y procesar las respuestas del servidor.

A continuación se muestra el diagrama de este contenedor, la explicación del flujo se puede seguir en el propio diagrama, pero ya ha sido descrita en el apartado 3.1 Aplicación Móvil en el capítulo 3. Sólo se muestran los principales componentes por simplicidad.

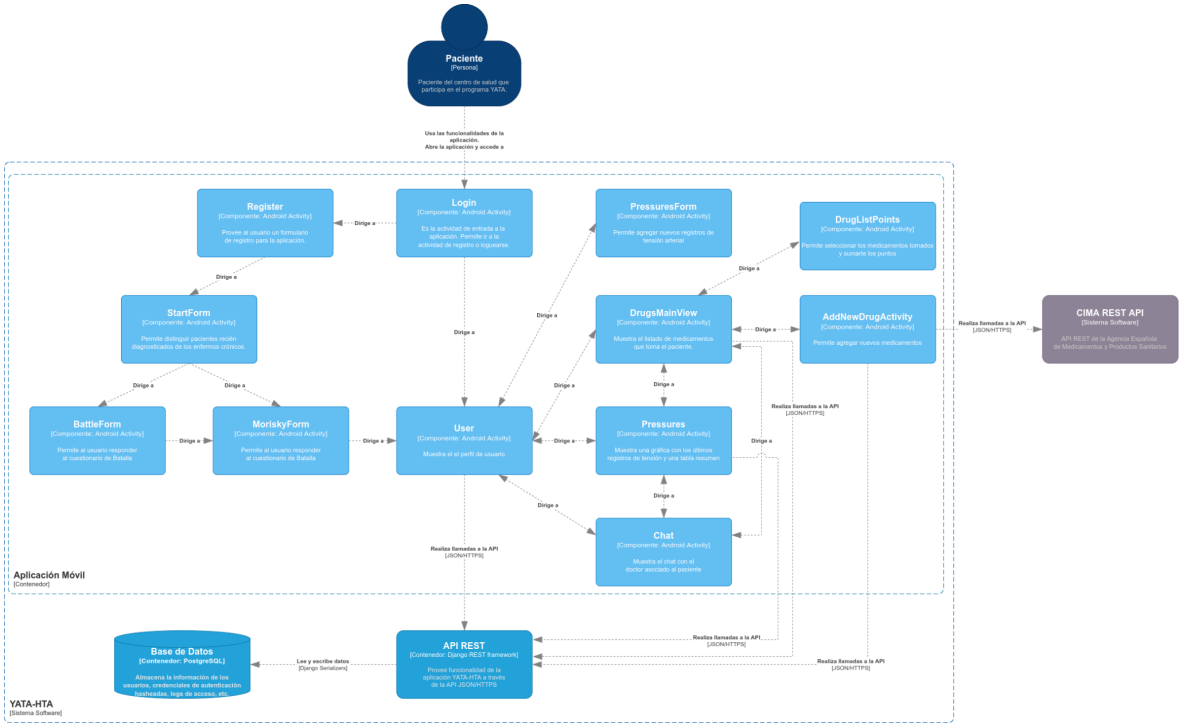


Imagen 54: Diagrama donde se muestran los componentes del contenedor “Aplicación Móvil” del sistema software para la aplicación YATA-HTA.

Ahora pasamos a describir el contenedor API REST, implementado con Django-REST-framework. Desde la aplicación Android, tal y como se acaba de explicar en

los párrafos anteriores, se realizan peticiones HTTP al contenedor "API REST". Este contenedor expone una interfaz con distintos puntos de acceso o *endpoints* que aceptan ciertas peticiones y en función del tipo de petición devuelve la respuesta adecuada.

Los puntos de entrada están descritos en el paquete *api* en el módulo *urls.py* dentro de la estructura de nuestro proyecto (*api/urls.py)*. Como también se ha descrito anteriormente, este módulo recibe las peticiones, y en función de la URL deriva la petición a la función correspondiente del módulo *api.views.py*. Este módulo, a su vez, hace uso de dos módulos: por una lado, a través del módulo *models.py* recupera o escribe la información necesaria en la base de datos y, por otro lado, a través del módulo *api.serializers.py* serializa y deserializa la información que recibe o envía por HTTP a la aplicación móvil cliente. Por defecto utilizamos el formato JSON [48] para mandar y recibir la información a través del protocolo HTTP.

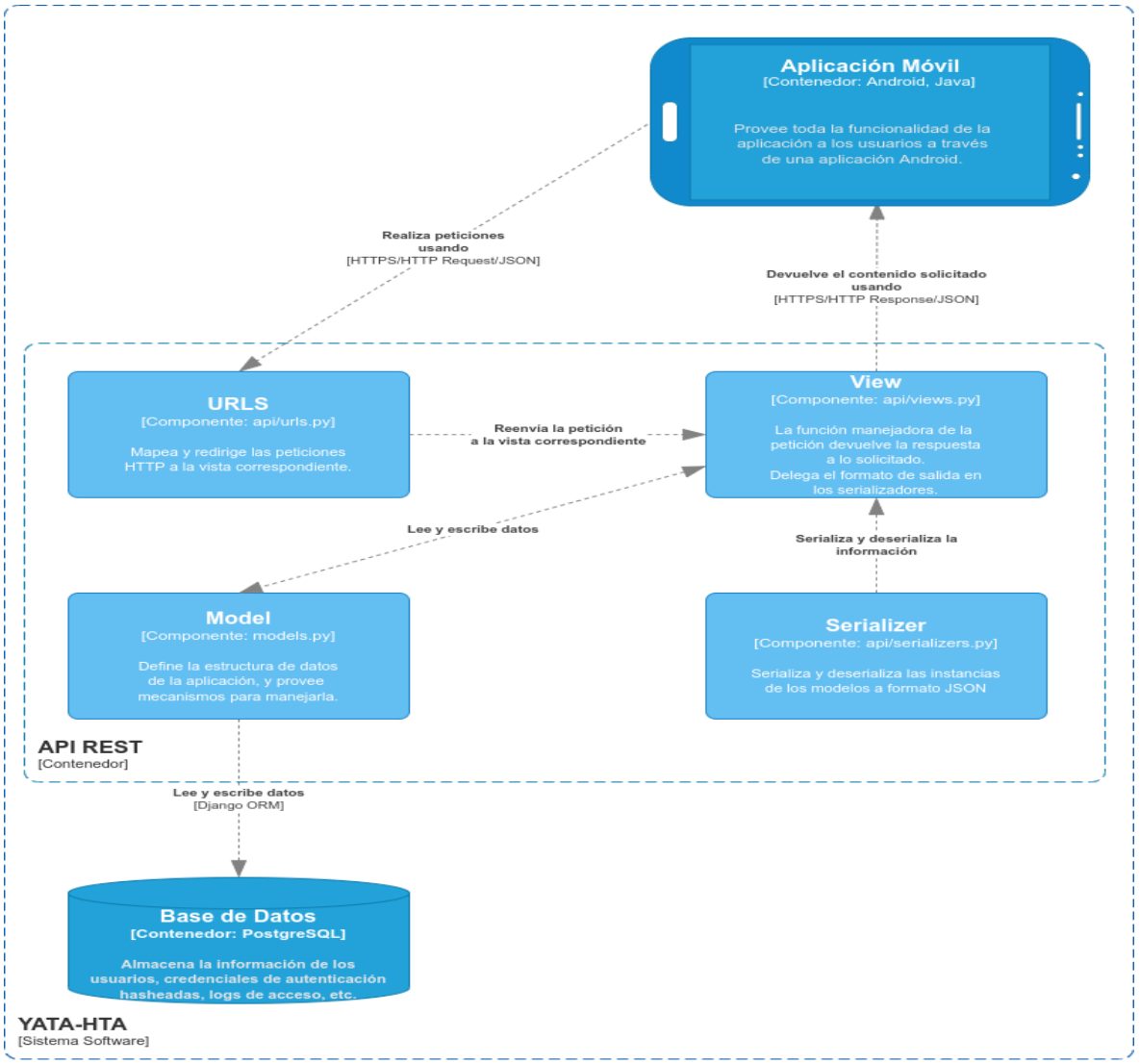


Imagen 55: Diagrama donde se muestran los componentes del contenedor "API REST" del sistema software para la aplicación YATA-HTA.

Por otra parte, tenemos la aplicación web propiamente dicha, como se puede observar en la siguiente imagen es muy similar al diagrama del contenedor “API REST”. Esto es así porque en general el flujo de trabajo de las aplicaciones Django es similar. En el caso anterior se trata de Django-REST-framework, que es un paquete que permite crear APIs de forma rápida, pero no deja de ser un complemento de Django propiamente dicho.

Al igual que en el contenedor anterior, el punto de entrada a la aplicación lo encontramos en el módulo *urls.py*, estas URLs se mapean con funciones del módulo *views.py* que son las que realmente ejecutan la lógica de negocio. De nuevo, este módulo se apoya, por una parte, en el módulo *models.py* que recupera o escribe la información necesaria en la base de datos y, por otro lado, en las plantillas o *templates* de Django. Estas plantillas no son más que archivos *html* con “huecos” que las vistas se encargan de rellenar con los datos extraídos de la base de datos en la respuesta que se le devuelve al cliente. En este caso, la información tiene formato *html* a través del protocolo HTTP.

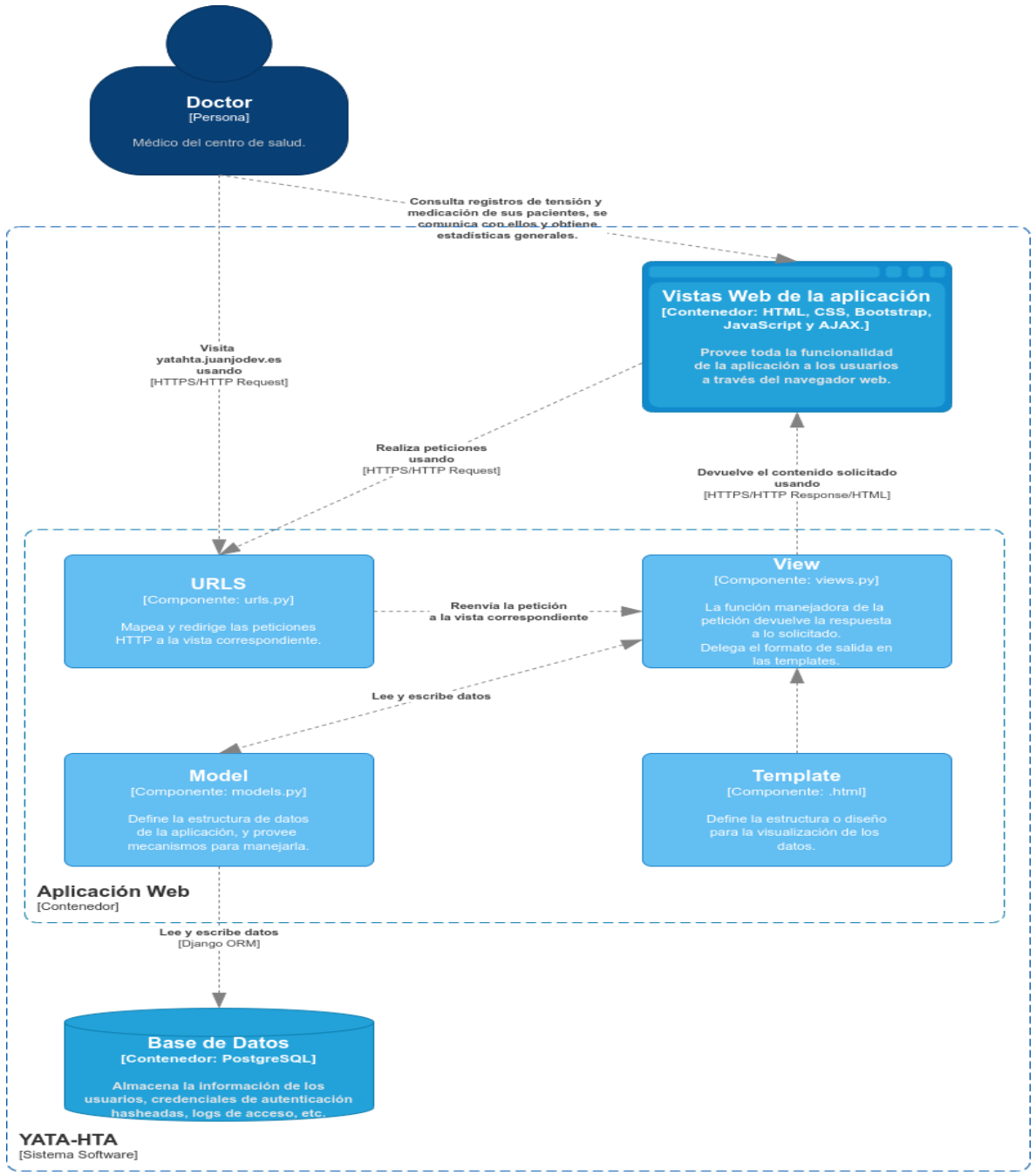


Imagen 56: Diagrama donde se muestran los componentes del contenedor “Aplicación Web” del sistema software para la aplicación YATA-HTA.

Finalmente, mostramos el diagrama del contenedor “Vistas Web”. La explicación del flujo se puede seguir en el propio diagrama, pero ya ha sido descrita en el apartado 3.2 Aplicación Web en el capítulo 3. Sólo se muestran los principales componentes por simplicidad.

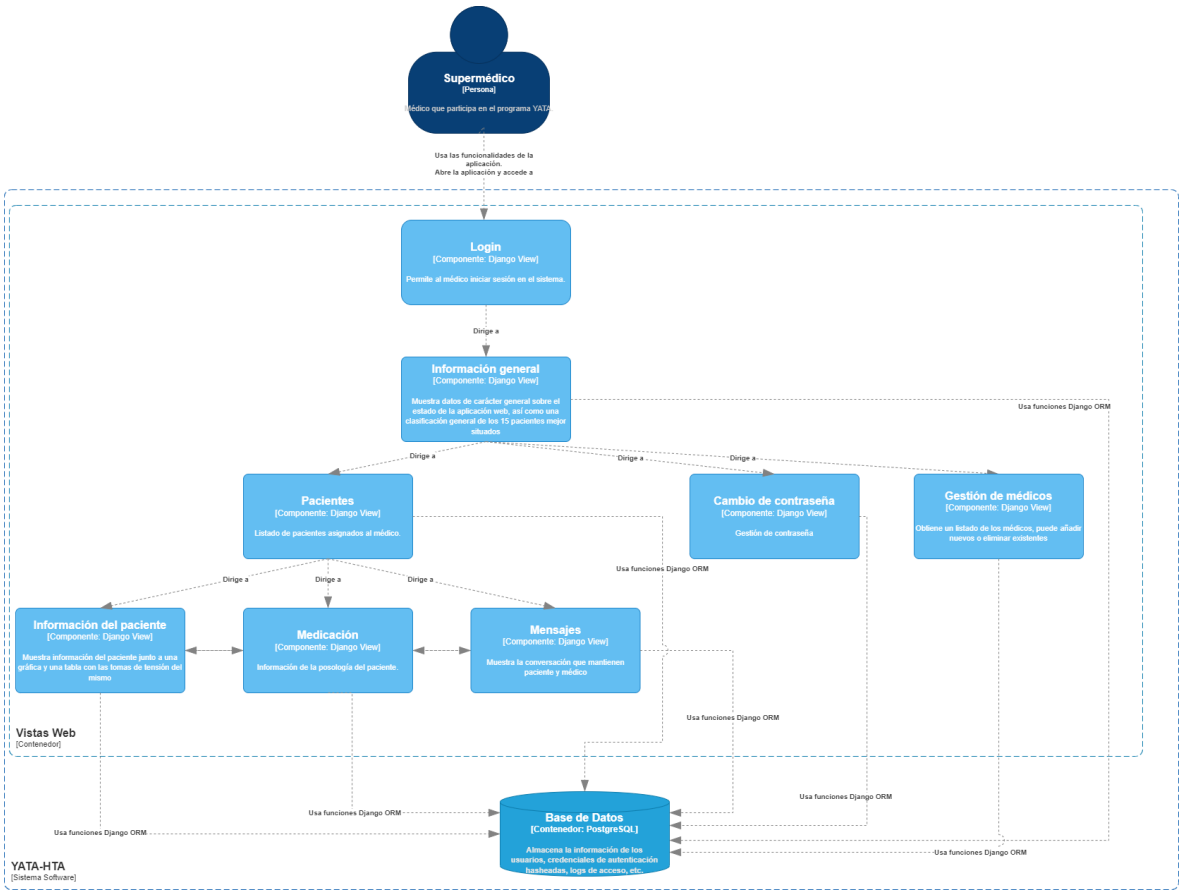


Imagen 57: Diagrama donde se muestran los componentes del contenedor “Vistas Web de la aplicación” del sistema software para la aplicación YATA-HTA.

4.2.2 Seguridad en la aplicación

Hoy en día es obligatorio implementar ciertas medidas de seguridad en cualquier aplicación que se desarrolle, más aún si los datos que se manejan son relativos a la salud de las personas. Aunque no almacenamos información que permita identificar a los pacientes directamente, hemos implementado ciertos niveles de seguridad.

En primer lugar, la aplicación está alojada en un servidor con certificado *TLS* [49], emitido por *Let’s Encrypt* [50], necesario para el protocolo HTTPS [51], por lo tanto, todas las comunicaciones entre el cliente y el servidor van cifradas.

En segundo lugar, para el uso de la aplicación es necesario que el usuario se pueda registrar, pero para poder registrarse primero debe tener un código facilitado por su médico o el personal sanitario del centro de salud. El personal sanitario es quien elige qué pacientes son susceptibles de entrar a formar parte del proyecto de investigación y, por lo tanto, serán quienes permitan darse de alta a los pacientes en la aplicación. Esta podría no considerarse una medida de seguridad, pero establece una barrera de entrada inicial a los usuarios no legítimos.

Los códigos que se facilitan a los usuarios para su registro son generados aleatoriamente por la aplicación web que utilizan los médicos. Son códigos alfanuméricos de 12 caracteres.

En tercer lugar, vamos a hablar de la autenticación. Esta la hemos implementado de dos formas en función de la aplicación cliente. Por una parte, la aplicación web a la que tiene acceso el personal sanitario utiliza el método de autenticación convencional basado en sesiones y *cookies*. Este método conserva el estado de la sesión en el cliente y en el servidor, por defecto Django guarda el estado de las sesiones en la base de datos.

Sin embargo, para la aplicación Android esto no es posible, en una arquitectura típica REST el servidor no guarda el estado de las conexiones. Es por esto que nos decidimos a implementar una autenticación basada en *tokens* para la API REST. Este método de autenticación es sin estado y el cliente guarda la información requerida para la autenticación en forma de *token*. Entre las distintas formas de generar *tokens*, *JSON Web Token (JWT)* [52] es el estándar de facto actual. El siguiente esquema resume cómo funciona este sistema:

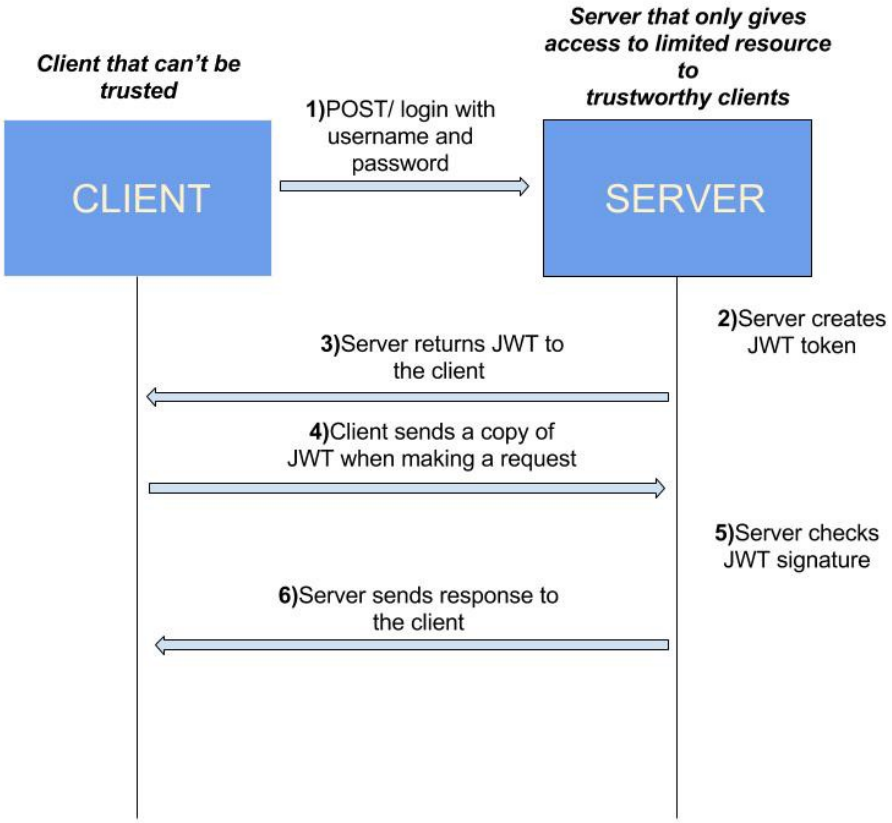


Imagen 58: Esquema resumen de JWT. Fuente: https://medium.com/python-pandemonium/json-web-token-based-authentication-in-django-b6dcfa42a332

Primero, el cliente realiza el proceso de *login* en la aplicación alojada en el servidor, para ello realiza una petición POST con el nombre de usuario y su contraseña. Si son correctos los datos, el servidor genera un JWT y lo devuelve al cliente. El cliente, a partir de este momento, deberá adjuntar en la cabecera de sus peticiones HTTP al servidor una cabecera de autenticación en la que se incluye este token. El servidor cuando reciba una petición que requiera de un usuario autenticado comprobará la firma del JWT y si es

correcta devolverá la respuesta a la petición, en caso contrario devolverá la respuesta con el mensaje de error de autenticación correspondiente.

Para implementar este sistema hemos hecho uso de la librería *Simple JWT* [53], que es un *plugin* para DRF. Hemos creado el paquete *auth* dentro de la aplicación Django para organizar mejor el código.

La API tiene los siguientes puntos de acceso para registro, login, actualización de *tokens*, cambio de contraseña y actualización del perfil del usuario:

URL	GET	POST	DELETE	PUT
/auth/change_password/<int:pk> Permite cambiar la contraseña del usuario	✗	✗	✗	✓
/auth/login/ Permite el login del usuario	✗	✓	✗	✗
/auth/login/refresh/ Permite el cambio del token cuando ha caducado	✗	✓	✗	✗
/auth/register/ Permite el registro de usuarios en la aplicación	✗	✓	✗	✗
/auth/update_profile/<int:pk> Permite actualizar datos del perfil de usuario	✗	✗	✗	✓

Tabla 3: URLs de los puntos de acceso de autenticación de la API.

En la sección de Apéndice I: Explicación detallada de los componentes mostramos en detalle el flujo y testeo de esta funcionalidad.

Finalmente, en cuarto lugar, vamos a hablar de los permisos de los usuarios. El permiso mínimo para usar la aplicación es estar registrado y haber realizado el proceso de *login*. Para registrarse diferenciamos a los dos tipos de usuarios. En el caso de los pacientes, han debido ser elegidos por el personal del centro de salud y le han debido facilitar un código de paciente. Para el registro de usuarios de tipo “personal sanitario” este proceso es un registro manual por un “médico administrador” tal y como se ha descrito en el punto 3.2.7 Gestión de médicos. Este médico administrador es dado de alta por nosotros en el sistema y se realiza a través de la página de administración de Django.

Por la construcción de la aplicación en teoría no debería ser posible acceder a los datos de otro usuario, pero siempre es posible que alguien con los conocimientos suficientes pudiese averiguar los puntos de acceso de la API e intentar acceder a esta información. En este caso, todas las vistas exigen primero estar *logueado* y, en segundo lugar, ser el propio usuario que accede a información propia o tener el rol de “personal sanitario”.

4.2.3 Esquema de la base de datos

En la primera reunión con los médicos del centro de salud empezamos a sacar requisitos conforme hablamos con ellos. Con esta primera aproximación pudimos elaborar el primer diagrama Entidad-Relación que satisficiera los requisitos:

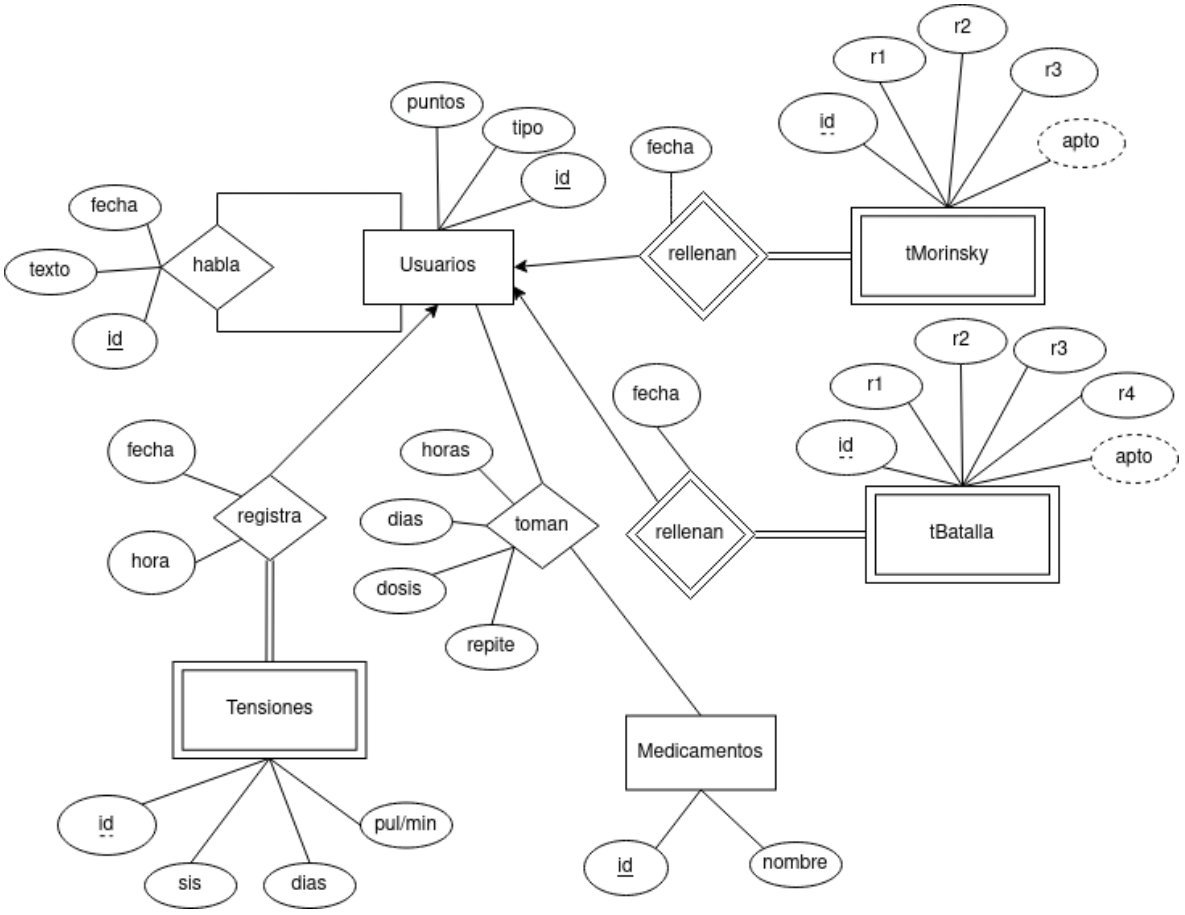


Imagen 59: Diagrama E-R de diseño para la base de datos.

Desde el principio se estableció que no teníamos que guardar datos personales de los usuarios, por lo que nos bastaba con saber qué tipo de usuario era (sanitario o paciente) y los puntos que iba adquiriendo para el sistema incentivador.

Otro requisito era poder registrar tomas de tensión, de las cuales lo básico era registrar la tensión sistólica, la diastólica y el pulso en ese momento, por lo que también incluimos fecha y hora en el registro. Modelamos esta entidad como una entidad débil puesto que no tiene sentido en sí misma sin el paciente al que pertenece. Lo mismo pensamos con respecto a las entidades que representan a los tests de Morisky-Green y Batalla. En los tests, además planteamos la posibilidad del campo calculado “apto” en base a las respuestas dadas a las preguntas del test.

La otra entidad fuerte que detectamos fue la de “Medicamento”, era necesario registrar qué medicación tomaban los usuarios y de qué forma la tomaban para poder generar recordatorios. Esto forma parte del incentivo y mejora de la adherencia al tratamiento.

Finalmente, otra funcionalidad a implementar era el poder comunicarse los pacientes con el personal del centro de salud, de ahí la relación recursiva.

A continuación se muestra el diagrama relacional obtenido:

- Usuarios(id_usuario, tipo, password, puntos, f_nacimiento)
- Medicamentos(id_medicamento, nombre)
- Usuarios_Medicamentos(id_usuario, id_medicamento, dosis, dia, hora, repite) con id_usuario *Foreign Key (FK)* en Usuarios y id_medicamento FK en Medicamentos
- Mensajes(id_mensaje, id_emisor, id_receptor, contenido, fecha, hora) con id_emisor e id_receptor FK en Usuarios
- Tensiones(id_toma, id_usuario, sistolica, diastolica, pulsaciones, fecha, hora) con id_usuario FK en Usuarios
- Moriskys(id_test, id_usuario, fehca, respuesta1, respuesta2, respuesta3, apto) con id_usuario FK en Usuarios
- Batallas(id_test, id_usuario, fehca, respuesta1, respuesta2, respuesta3, respuesta4, apto) con id_usuario FK en Usuarios

Se puede apreciar que no coinciden el diagrama E-R y el relacional completamente. Esto es debido a que durante el desarrollo hemos ido detectando posibles fallos y se han ido corrigiendo. Además, conforme han surgido nuevos requerimientos hemos ido adaptando también el esquema de la base de datos.

Finalmente, vamos a mostrar un diagrama de clases UML de los modelos Django que sacamos a partir de este esquema relacional.

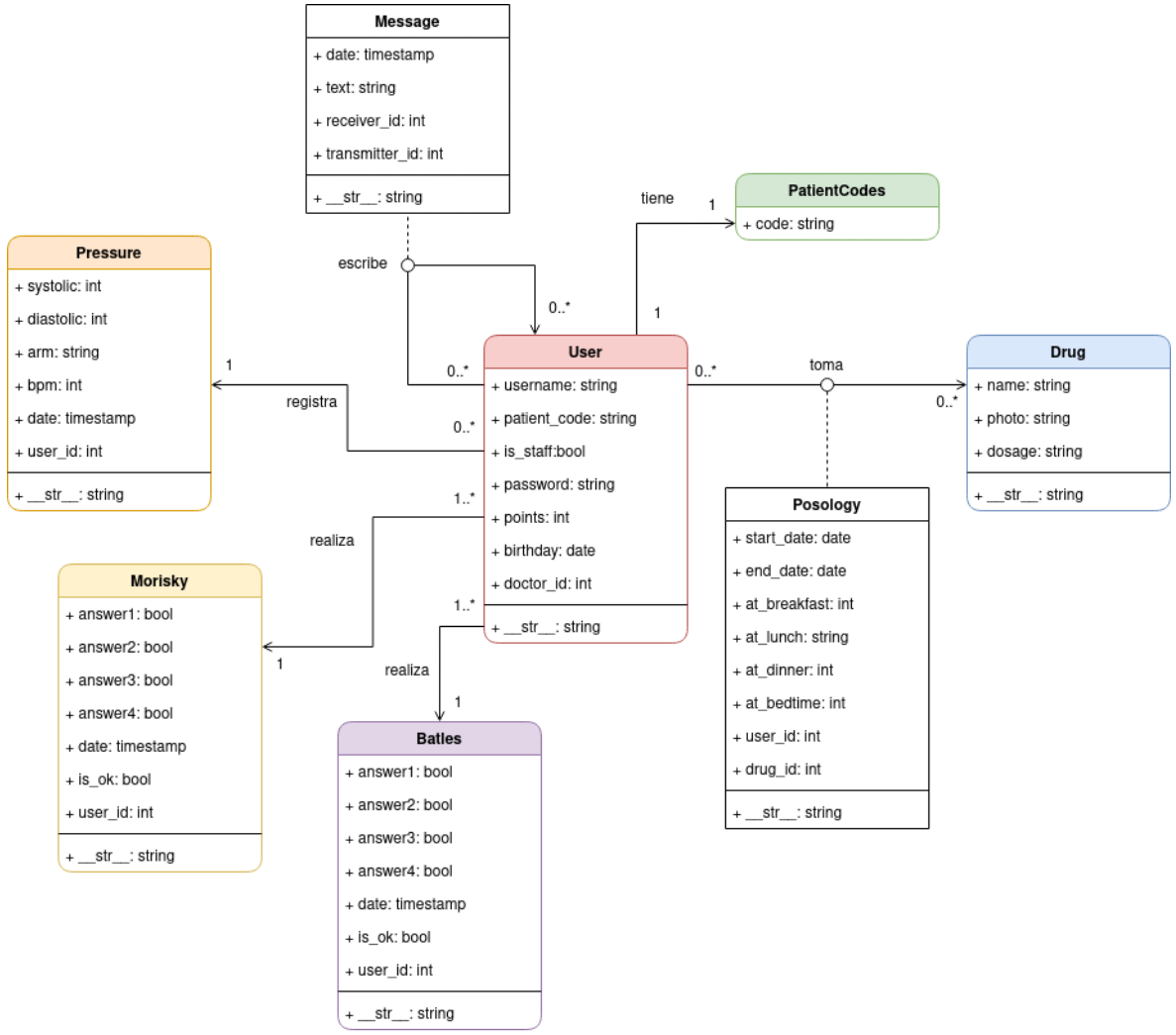


Imagen 60: Diagrama de clases UML de los modelos Django para la base de datos.

4.2.4 Despliegue de la aplicación

Como se ha comentado anteriormente, en el apartado 4.1.4 Docker, para poder tener un entorno de desarrollo común y aislado de nuestros sistemas operativos nos decidimos por usar la tecnología que pone Docker a nuestra disposición.

Adicionalmente, con Docker también nos aseguramos que nuestra aplicación va empaquetada con todas sus dependencias y que funciona tal y como la hemos estado probando en nuestros dispositivos.

Por todo ello, y con la disponibilidad de un VPS, nos hemos decidido a desplegar la aplicación en este servidor. El servidor es privado y no pretendemos que su despliegue definitivo sea en este equipo, pero sí que nos ha permitido hacer pruebas y demostraciones a nuestros clientes. Y puesto que usamos Docker, desplegar la aplicación es tan fácil como tener el código docker corriendo en el servidor y ejecutar un comando de consola, puesto que la creación de las imágenes docker está especificada en un *Dockerfile* y el *stack* de contenedores a desplegar en un *docker-compose*.

Actualmente la aplicación web está accesible desde la URL yatahta.juanjodev.es y la API REST en yatahta.juanjodev.es/api. Se encuentra detrás del *proxy* inverso *traefik* [54],

este se encarga además de renovar el certificado TLS con *Let's Encrypt* para el protocolo HTTPS.

Como hemos comentado anteriormente, distinguimos entre nuestro entorno de desarrollo y el entorno de producción. Esto hace que tengamos archivos distintos para desplegar la aplicación, ya sea en local para desarrollo o en el servidor para producción. En la siguiente tabla explicamos qué hace cada uno y su localización con respecto a la raíz del proyecto.

Archivo	Función
docker-compose.yml	Permite desplegar el entorno de desarrollo.
docker-compose.prod.yml	Permite desplegar la aplicación en el servidor.
.env/.env.dev	Archivo de configuración de variables de entorno para el entorno de desarrollo.
.env/.env.prod	Realmente no se ha compartido este archivo, se ha desplegado directamente al servidor mediante SFTP. Archivo de configuración de variables de entorno para el entorno de producción.
.env/.env.prod.db	Realmente no se ha compartido este archivo, se ha desplegado directamente al servidor mediante SFTP. Archivo de configuración de variables de entorno de la base de datos para el entorno de producción.
app/compose/local/django/entrypoint.sh	Script que permite que el contenedor de la aplicación web espere el inicio del contenedor de PostgreSQL y conecte con la base de datos correctamente, pero para la imagen de desarrollo.
app/compose/production/django/entrypoint.sh	Script que permite que el contenedor de la aplicación web espere el inicio del contenedor de PostgreSQL y conecte con la base de datos correctamente, pero para la imagen de producción.
app/compose/local/django/Dockerfile	Da las instrucciones al servicio de docker acerca de cómo construir la imagen de desarrollo.
app/compose/production/django/Dockerfile	Da las instrucciones al servicio de docker acerca de cómo construir la imagen de producción

Tabla 4: Archivos de configuración del proyecto para su despliegue.

El despliegue debería ser tan sencillo como ejecutar el comando *docker-compose -f docker-compose.prod.yml up -d --build* en la raíz del proyecto en el servidor de despliegue. Pero esto es algo que deberían hacer los técnicos encargados de los sistemas informáticos del centro de salud.

Por su parte, para el despliegue de la aplicación Android hemos optado por subir la aplicación empaquetada en un archivo en formato *.apk* [55] al VPS. Está disponible para su descarga en la siguiente dirección:

https://drive.google.com/file/d/19rlXMlZQ91wcK8NZgRKFslCRqUX5SczW/view?usp=sharing

Para su instalación es necesario descargar el archivo al móvil del paciente y ejecutarlo, normalmente el sistema operativo mostrará un aviso indicando que se deben habilitar las fuentes desconocidas y después de habilitarlas será posible su instalación.

4.2.5 Documentación de la API

Como se comentaba en el apartado 4.1 Herramientas y tecnologías utilizadas, la herramienta utilizada para desarrollar la API REST ha sido Django REST framework (DRF). Se pretendía crear una API REST con una tecnología ampliamente utilizada a día de hoy y que hiciera fácil y accesible el uso de la misma, especialmente desde el lado de la aplicación móvil.

Los puntos de acceso de la API los ha generado automáticamente DRF a partir del módulo *api.urls.py,* este proceso se explica con más detalle en los apéndices en el apartado Ciclo de una petición REST. La información más relevante para el usuario externo al desarrollo de la aplicación es, por tanto, la relativa a los puntos de acceso que describimos en la siguiente tabla.

Esta tabla muestra los distintos puntos de acceso a la API con los verbos HTTP permitidos. En la columna “URL” indicamos en negrita la URL propiamente dicha y debajo una descripción de la información devuelta. El método o verbo GET permite obtener información, POST permite crear nuevos registros, DELETE es para eliminar registros y PUT/PATCH son para actualizar información, con PUT la actualización del registro exige que la petición lleve todos los campos obligatorios y con PATCH la actualización es parcial, actualizando tan solo la información necesaria.

URL	GET	POST	DELETE	PUT/PATCH
/api/users/ Listado de usuarios de la aplicación	✔	✔	✘	✘
/api/users/<id>/ Información del usuario con ese id.	✔	✘	✔	✔
/api/users/<id>/messages/ Listado con los mensajes que envía o recibe el usuario con ese id.	✔	✘	✘	✘
/api/users/<id>/batles/ Listado con los test de batalla de ese usuario.	✔	✘	✘	✘

URL	GET	POST	DELETE	PUT/PATCH
/api/users/<id>/moriskys/ Listado con los test de Morisky-Green de ese usuario.	✔	✘	✘	✘
/api/users/<id>/posologies/ Listado con los medicamentos tomados y su posología por el usuario con ese id.	✔	✘	✘	✘
/api/users/<id>/pressures/ Listado de tomas de tensión de ese usuario.	✔	✘	✘	✘
/api/users/?username=<string>&doctor=<id>& birthday=<fecha> Información del usuario filtrada por nombre de usuario, id de un doctor o una fecha de nacimiento.	✔	✘	✘	✘
/api/users/classification/ Listado con la clasificación de los usuarios ordenada de mayor a menor puntuación.	✔	✘	✘	✘
/api/users/<id>/user_classification/ Listado con la clasificación del usuario en concreto, mostrando su entorno más cercano junto con los 3 primeros clasificados.	✔	✘	✘	✘
/api/users/users_count/ Devuelve el número total de usuarios de la aplicación. Esto lo empleamos en la vista del perfil de usuario de la aplicación Android.	✔	✘	✘	✘
/api/drugs/ Listado de todos los medicamentos almacenados.	✔	✔	✘	✘
/api/drugs/<id>/ Información del medicamento con ese id.	✔	✘	✔	✔
/api/drugs/?name=<string> Información de un medicamento dado un nombre.	✔	✔	✘	✘
/api/posologies/ Listado de todas las posologías almacenadas.	✔	✔	✘	✘
/api/posologies/<id>/ Información de una posología dado su id.	✔	✘	✔	✔
/api/posologies/?user=<id>&drug=<id> Listado de posologías filtradas por id de usuario e id de medicamento.	✔	✔	✘	✘
/api/messages/ Listado con todos los mensajes almacenados.	✔	✔	✘	✘
/api/messages/<id>/ Información de un mensaje dado su id.	✔	✘	✔	✔

URL	GET	POST	DELETE	PUT/PATCH
/api/messages/?transmitter=<id>&receiver=<id>&date=<fecha> Listado de mensajes dado un id de transmisor y/o un id de receptor.	✔	✔	✘	✘
/api/pressures/ Listado de todas las presiones arteriales almacenadas.	✔	✔	✘	✘
/api/pressures/<id>/ Información de una presión dado su id.	✔	✘	✔	✔
/api/pressures/?user=<id>&date=<fecha> Listado de presiones filtrado por id de usuario y fecha.	✔	✔	✘	✘
/api/moriskys/ Listado de todos los test de Morisky-Green realizados.	✔	✔	✘	✘
/api/moriskys/<id>/ Información de un test de Morisky dado su id.	✔	✘	✔	✔
/api/moriskys/?user=<id>&date=<fecha> Listado de los test de Morisky filtrado por id de usuario y fecha.	✔	✔	✘	✘
/api/batles/ Listado de todos los test de Batalla realizados.	✔	✔	✘	✘
/api/batles/<id>/ Información de un test de Batalla dado su id.	✔	✘	✔	✔
/api/batles/?user=<id>&date=<fecha> Listado de los test de Batalla filtrado por id de usuario y fecha.	✔	✔	✘	✘

Tabla 5: URLs de los puntos de acceso de la API mostrando los métodos HTTP permitidos.

Capítulo 5: *Conclusiones*

Como conclusiones para este Trabajo de Fin de Grado creemos importante hablar especialmente de tres aspectos. En primer lugar queríamos hacer una reflexión sobre los conceptos que hemos adquirido a lo largo del tiempo de desarrollo. En segundo lugar, destacar y hablar de algo que consideramos que será importante en nuestra vida de ahora en adelante: el trabajo con un cliente real y poder ajustarnos a sus necesidades. Además, somos conscientes de que como todo en la vida, el proyecto puede mejorar. Por ello, el último punto que trataremos en estas conclusiones serán las futuras mejoras que creemos que mejorarían el rendimiento de las aplicaciones y la experiencia general del usuario y que implementaríamos si contáramos con más tiempo.

5.1 Conceptos aprendidos

Bien es cierto que los tres integrantes del grupo habíamos cursado la asignatura de Aplicaciones Web (AW) y Juanjo tenía experiencia en desarrollo Android gracias a haber cursado Programación de Aplicaciones para Dispositivos móviles (PAD) en cursos anteriores. Tanto Fernando como Ignacio cursaron PAD durante el segundo cuatrimestre del curso en que este TFG se estaba desarrollando, razón por la cual tuvieron que aprender conceptos de Android antes de empezar con la asignatura para desarrollar la aplicación móvil a partir de septiembre. Sin embargo, hay tecnologías y formas de trabajo que hemos aprendido con el objetivo de hacer una aplicación lo más actual y completa posible.

Una de esas tecnologías a las que hacíamos referencia es Django. Es la herramienta que utilizamos para diseñar y desarrollar la API de la aplicación. Dos de los integrantes teníamos experiencia previa programando APIs, pero levantar una de esta magnitud y con una herramienta nueva para nosotros ha supuesto un reto.

Debemos mencionar además el hecho de que tuviéramos que acostumbrarnos a trabajar haciendo uso de Git. A pesar de que es algo que se intenta incentivar en muchas asignaturas de la carrera, no había sido nunca tan necesario como para este proyecto. Para avanzar de la forma más rápida posible, hemos aprendido la importancia de una comunicación fluida entre nosotros y la gestión de ramas para hacer un uso apropiado de las versiones que estaba desarrollando cada uno.

Otro aprendizaje que cabe destacar, es nuestra particular odisea a la hora de generar notificaciones en el sistema Android. Debíamos asegurarnos de que se preservasen en el tiempo y pudieran saltar independientemente del estado del teléfono en cada momento.

5.2 Trabajo con un cliente real

Como es de suponer, uno de los factores más importantes para nosotros como equipo era la comunicación lo más cercana posible con nuestros colaboradores del Centro de Salud de los Yébenes. Además, ha conformado el mayor cambio con respecto a lo que veníamos haciendo en proyectos relacionados con asignaturas de la carrera. Veníamos acostumbrados a hacer una serie de aplicaciones de distinto tipo en base a lo que decía un papel o un enunciado. El hecho de obtener periódicamente un *feedback* por parte de nuestro “cliente”, nos venía bien no sólo para saber qué pasos eran los siguientes que

debíamos dar como creadores de la aplicación, sino también para ver qué pensaban en cada momento acerca de nuestro trabajo.

Consideramos importante además destacar el hecho de que hayan estado tan involucrados con el proyecto. Han aportado una cantidad muy importante de ideas, y siempre han estado disponibles para dudas que tuviéramos.

En general, ha sido una experiencia muy positiva para nosotros, no solo porque nos ha venido bien para aprender una forma nueva de trabajo, sino porque ha sido muy fácil a lo largo de todo el proceso.

Sin embargo, también hemos aprendido que es importante establecer unos requisitos claros desde el principio. Tras las primeras reuniones pudimos establecer una serie de requisitos, pero con los sucesivos avances en la implementación de la aplicación también fueron surgiendo nuevas ideas y nuevos requisitos. Esto en general deberíamos haberlo controlado mejor porque siempre altera los planes previos que teníamos establecidos y desajusta las estimaciones iniciales de tiempos a emplear en las tareas.

5.3 Futuras mejoras / nuevas características

Por último queremos destacar posibles mejoras y nuevas características para el proyecto:

- Desacoplamiento total de la aplicación web y las vistas web, usando por ejemplo un *framework* como *React* [56]. Actualmente se emplean plantillas de Django, lo ideal sería un *frontend* web que sólo interactúa con la API REST y no directamente con el ORM de Django.
- De la primera se deriva que la aplicación web que haga uso de la API REST. Usaría JWT como método de autenticación.
- Implementar un almacenamiento local en la aplicación Android, de modo que la aplicación se pueda usar sin necesidad de tener acceso a internet.
- Mejorar la interfaz de usuario.
- Añadir notificaciones push en Android en lugar de recordatorios mediante alarmas, igualmente esto es válido para la aplicación web.
- Sistema de alertas basado en el registro que hacen los pacientes de su tensión arterial, de modo que si un pacientes está descontrolado le salte una alerta a su médico.
- Interacción con dispositivos inteligentes de tipo *wearables*, como pueden ser los *smartwatches*.
- Incluir tests para la aplicación web y la aplicación Android. Actualmente sólo la API REST tiene una buena cobertura de tests.
- Actualización automática de chat de forma que se muestren los mensajes recibidos sin necesidad de recargar manualmente, tanto en la aplicación móvil como en la aplicación web.
- Sistema de notificaciones para indicar si hay mensajes por leer en ambas aplicaciones.
- Proporcionar a los médicos los resultados de los tests de Morisky/Green y Batalla de sus pacientes asociados.
- Usar un servidor de correo propio para no depender del servicio de Gmail.

Chapter 5: Conclusions

As conclusions for this Final Degree Project (FDP), we believe it is important to talk about three aspects in particular. Firstly, we wanted to discuss the concepts that we have acquired during the development time. Secondly, emphasize and talk about something that we believe will be important in our lives from now on: working with a real client and being able to adjust to their needs. Moreover, we are aware that like everything in life, the project can be improved. Therefore, the last point we will discuss in these conclusions will be future features that we believe would improve the performance of the applications and the overall user experience and that we would implement if we had more time.

5.1 Concepts learned

It is true that the three members of the group had taken the Web Applications (AW) subject and Juanjo had experience in Android development thanks to having studied Programming Applications for Mobile Devices (PAD) in previous courses. Both Fernando and Ignacio took PAD during the second semester of the year in which this FDP was being developed, which is why they had to learn Android concepts before starting the subject in order to develop the mobile application from September onwards. However, there are technologies and ways of working that we have learnt in order to make an application as up-to-date and complete as possible.

One of these technologies that we referred to is Django. It is the tool we used to design and develop the application's API. Two of us had previous experience programming APIs, but building an application of this magnitude and with a new tool was a challenge for us.

We should also mention the fact that we had to get used to working with Git. Although it is something that is encouraged in many subjects in the degree, it had never been as necessary as it was for this project. In order to move forward as quickly as possible, we learned the importance of fluid communication between us and the management of branches to make appropriate use of the versions that each of us was developing.

Another learning that is worth highlighting is our particular odyssey when generating notifications in the Android system. We had to ensure that they were preserved over time and could be triggered regardless of the state of the phone at any time.

5.2 Working with a real client

As expected, one of the most important factors for us as a team was to communicate as closely as possible with our partners at the “Los Yébenes” Health Center. It has also been the biggest change from what we had been doing in projects related to degree subjects. We were used to creating a series of different types of applications based on what a paper or a statement said. The fact of periodically obtaining feedback from our client was good for us, not only to know what steps we should take next as creators of the application, but also to see what they thought about our work at any given moment.

We also think it is important to highlight the fact that they have been so involved in the project. They have provided a very important amount of ideas, and they have always been available for any doubts we had.

Overall, it has been a very positive experience for us, not only because it has been good for us to learn a new way of working, but also because it has been very easy throughout the whole process.

However, we have also learned that it is important to establish clear requirements from the beginning. After the first meetings we were able to establish a set of requirements, but as the implementation of the application progressed, new ideas and new requirements emerged. In general, we should have controlled this better, because it always alters the previous plans we had established and it distorts the initial estimates of the time to be spent on the tasks.

5.3 Future improvements / new features

Finally we would like to highlight possible improvements and new features for the project:

- Full decoupling of the web application and web views, using for example a framework such as React [54]. Currently Django templates are used, ideally a web frontend that only interacts with the REST API and not directly with the Django ORM.
- From the first, a web application that makes use of the REST API. It would use JWT as an authentication method.
- To implement local storage in the Android application, so that the application can be used without the need for internet access.
- To improve the user interface.
- To add push notifications on Android instead of reminders via alarms, this is also valid for the web application.
- To alert system based on patients' blood pressure recordings, so that if a patient's blood pressure is out of control, an alert will be sent to their doctor.
- Interaction with smart wearable devices, such as smartwatches.
- To include tests for the web application and the Android application. Currently only the REST API has good test coverage.
- Automatic update of chat so that messages received are displayed without the need to reload manually, both in the mobile application and in the web application.
- Notification system to indicate if there are messages to be read in both applications.
- To provide doctors with Morisky/Green and Batalla tests of their associated patients.
- To use an own mail server so as not to depend on the Gmail service.

Capítulo 6: *Contribuciones al proyecto*

En este capítulo presentamos las aportaciones realizadas por cada uno de los integrantes del equipo durante el desarrollo del proyecto, así como las partes en las que ha sido necesario el trabajo conjunto de todos.

6.1 Distribución de trabajo

Como se comentaba anteriormente en el punto 1.3 Plan de Trabajo, para el desarrollo del proyecto se han implementado dos tipos de aplicaciones distintas, una aplicación móvil para Android que usarán los pacientes y otra aplicación web accesible para los doctores.

En un principio se tomó la decisión de desarrollar entre todo el equipo la aplicación móvil, hasta que fuera hasta cierto punto funcional, principalmente para que los doctores pudieran darnos su punto de vista con respecto al diseño de la misma y pudieran ver avances en un corto periodo de tiempo. A partir de este momento Fernando se ha centrado en el desarrollo de la aplicación Android.

Cuando se acordó implementar la aplicación web para los doctores, Nacho y Juanjo que tenían algo de experiencia previa en implementación de APIs REST comenzaron con la investigación de la herramienta Django y las posibilidades que ofrece en este sentido, así como el *framework* para el diseño de las vistas de la web.

Una vez establecida la base para la aplicación web, Nacho ha implementado fundamentalmente el *frontend*, mientras que Juanjo se ha centrado en el *backend.*

Ahora vamos a mostrar las aportaciones a los dos repositorios con los que hemos trabajado. Si bien, es justo aclarar que medir el rendimiento de los usuarios por número de *commits* muchas veces no es indicativo de su aportación, ya que distintos usuarios tienen distintos hábitos en cuanto a cuándo subir código o hacer un *commit*.

En cuanto a los repositorios, primero vamos a tratar el de Android. Como se puede ver en la siguiente imagen, el trabajo ha sido constante a lo largo del curso, con pequeños parones en época de exámenes y entregas de proyectos de otras asignaturas, pero en general bastante uniforme:

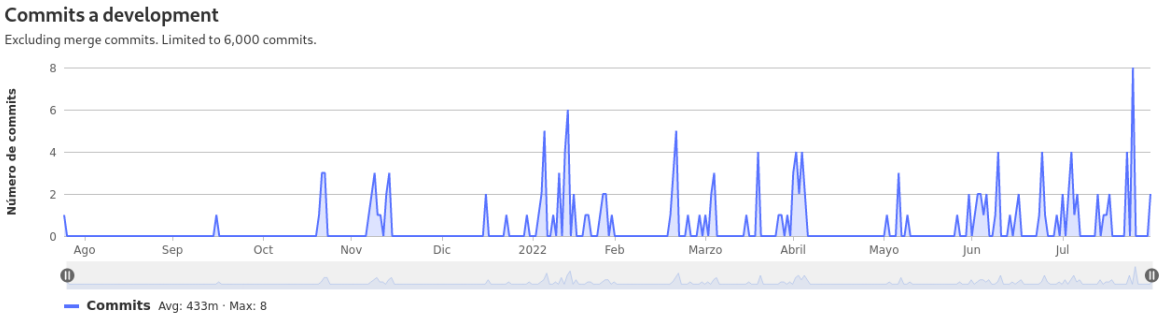


Imagen 61: Distribución a lo largo del tiempo del número de *commits* al repositorio de la aplicación Android.

Del mismo modo, si miramos los *commits* por miembros del equipo, se puede observar que Fernando ha sido el principal contribuidor a la aplicación Android:

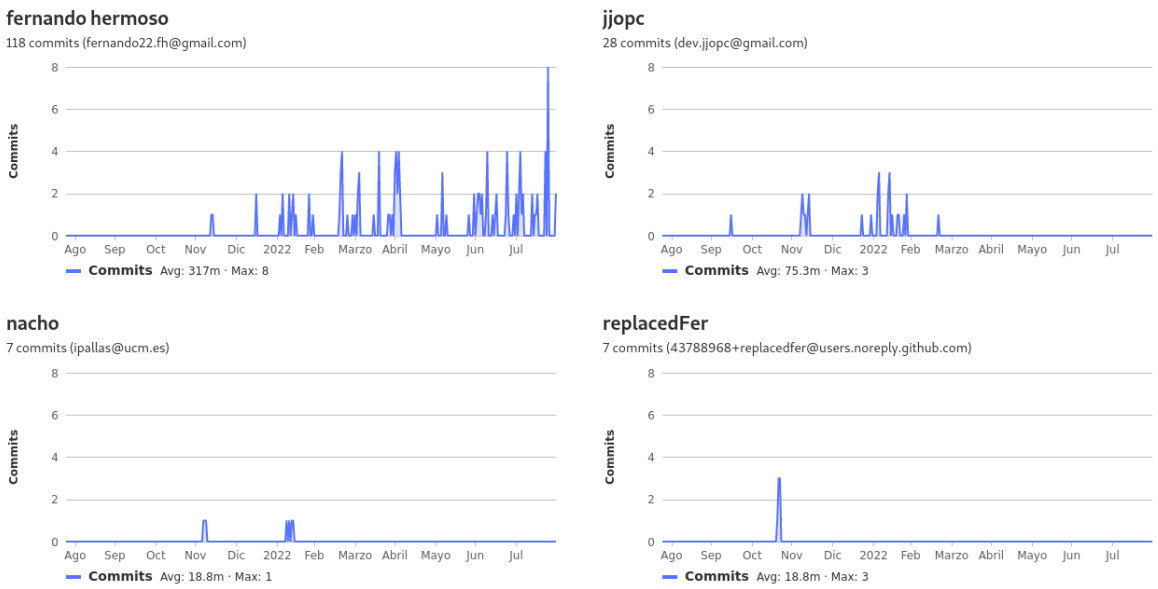


Imagen 62: Distribución a lo largo del tiempo del número de *commits* al repositorio de la aplicación Android por usuarios.

Nombre usuario en repositorio	Nombre autor
fernando hermoso	Fernando Hermoso Cara
jjopc	Juan José Plaza Campillo
nacho	Ignacio Pallás Gozálvez
replacedFer	Fernando Hermoso Cara

Tabla 6: Leyenda de autores del repositorio Android

En cuanto al repositorio de Django y la API REST la distribución en el tiempo es menos homogénea, y se aprecia que se comenzó con esta parte del proyecto un poco más tarde.

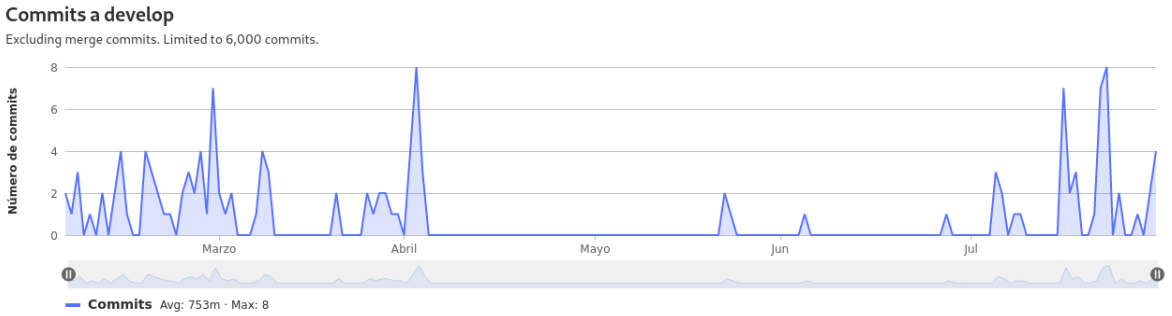


Imagen 63: Distribución a lo largo del tiempo del número de *commits* al repositorio de la aplicación Django y API REST.

Si vemos a los usuarios particulares en cuanto a número de commits, se aprecia que Juanjo es quien más ha aportado (con distintos correos por haber trabajado desde distintos equipos).

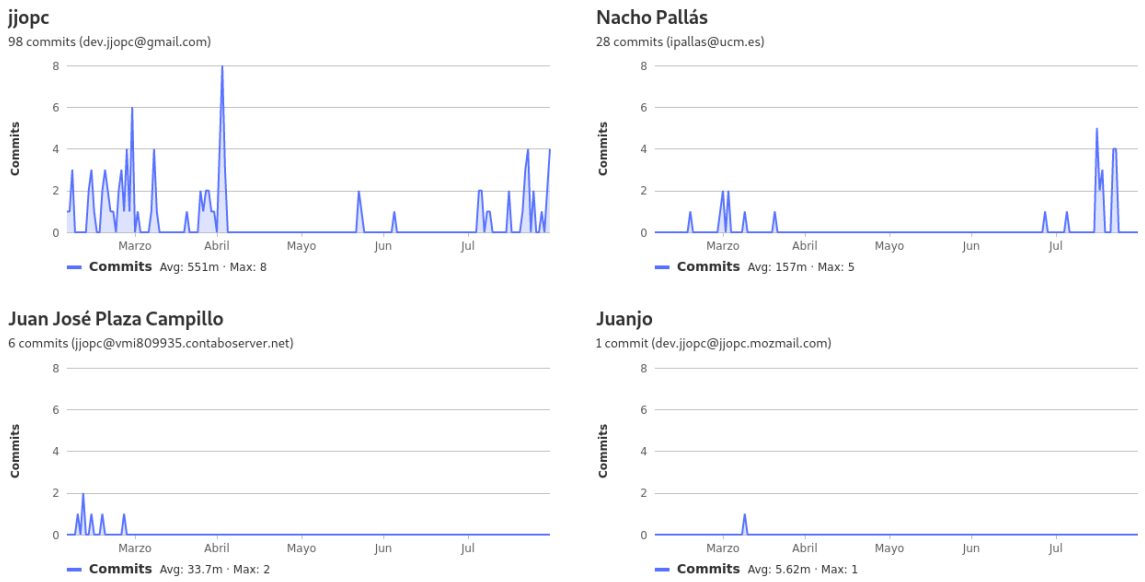


Imagen 64: Distribución a lo largo del tiempo del número de *commits* al repositorio de la aplicación Django y API REST por usuarios.

Nombre usuario en repositorio	Nombre autor
jjopc	Juan José Plaza Campillo
Nacho Pallás	Ignacio Pallás Gozálvez
Juan José Plaza Campillo	Juan José Plaza Campillo
Juanjo	Juan José Plaza Campillo

Tabla 7: Leyenda de autores del repositorio de la aplicación Django y API REST.

Finalmente, vamos a desarrollar un poco más en detalle las aportaciones concretas de cada uno de los miembros del equipo.

6.2 Juan José Plaza Campillo

Tal y como ya se ha comentado, en un principio todos los integrantes del equipo comenzamos trabajando en la investigación de aplicaciones similares a la que pretendíamos desarrollar. Esto nos ocupó las primeras semanas junto con el desarrollo de bocetos, elaboración de historias de usuario y *User Story Mapping*.

Al principio contribuí creando los repositorios en GitLab, creando algunos *issues* iniciales, y desarrollando algunas vistas iniciales de la aplicación Android. En general he

sido responsable de los repositorios y el encargado de realizar las fusiones de ramas con nuevas características con la rama de desarrollo.

Una vez llevadas a cabo las primeras reuniones con nuestros clientes, y cuando la aplicación móvil iba cogiendo forma, ya nos diversificamos y especializamos un poco más cada uno en un ámbito concreto del proyecto.

Después, entre todos hemos ido creando nuevas tareas en los repositorios. Cuando estas nuevas tareas no eran para nuestra área de responsabilidad entonces las asignamos al compañero responsable. Esto nos ha servido para poder seguir trabajando sin interrupciones y siempre eran acordadas previamente en las reuniones que manteníamos.

Para intentar ser un poco más exhaustivo voy a relatar cómo he contribuido a cada una de las partes de este proyecto. Mi labor se ha centrado fundamentalmente en la parte de *backend*, esto incluye fundamentalmente la capa de base de datos, persistencia y negocio de la aplicación.

En la capa de datos he llevado a cabo la configuración y despliegue de PostgreSQL sobre un contenedor Docker. El diseño de la base de datos ha sido conjunto y fue llevado a cabo durante las semanas iniciales de nuestro proyecto, aunque posteriormente ha ido evolucionando conforme han ido surgiendo nuevas características o algunos problemas.

En la capa de persistencia y negocio he investigado mucho sobre Django, que aunque tenía cierto conocimiento previo gracias a la asignatura de “Gestión de Información en la Web”, no era suficiente para el desarrollo de un proyecto tan complejo como este.

Concretamente he llevado a cabo el desarrollo de los modelos, vistas y URLs de la aplicación web, así como también he desarrollado la API REST con DRF, la gestión de permisos y el flujo de autenticación de usuarios. He llevado a cabo los tests que se realizan a la API REST con *pytest* [57].

Adicionalmente, he investigado y llevado a cabo el empaquetado del proyecto mediante la tecnología de Docker, así como su despliegue en el VPS. Para ello también he tenido que investigar y aprender a configurar el servidor, su seguridad y su mantenimiento.

En general, de todas las tecnologías empleadas tenía cierto grado de conocimiento por haber trabajado con ellas en algunas asignaturas de la carrera durante prácticas o proyectos pequeños, pero en ningún caso tenía el conocimiento suficiente para usarlas con soltura. He tenido que investigar, hacer muchos tutoriales, seguir algunos libros, etc. para poder emplearlas correctamente. Ahora considero que he adquirido cierta soltura con todas ellas, de modo que he podido depurar errores que han ido surgiendo y he profundizado en algunas características más allá de lo que abarcan la mayoría de tutoriales que se pueden encontrar en internet.

6.3 Ignacio Pallás Gozálvez

Cuando comenzamos con el proyecto en septiembre, mi labor fue la de recopilar información de otras aplicaciones con una utilidad parecida a la que queríamos desarrollar junto a mis compañeros.

Posteriormente, también propuse ideas para los bocetos de la aplicación móvil, y empecé a obtener información sobre el funcionamiento de Android y el desarrollo de aplicaciones en el entorno Android Studio. Diseñé el menú de navegación de la aplicación móvil y el frontend de algunas de las vistas que se mantienen en la versión final. No pude implementar gran parte de la lógica de la aplicación junto a mis compañeros en ese

momento ya que la herramienta exigía demasiado a mi ordenador y no funcionaba correctamente. En el momento en que concluímos que iba a ser necesaria una aplicación web aparte, diseñé los bocetos y la distribución de frontend de las páginas.

En el momento en que se comenzó a desplegar la API, tuve que informarme sobre el funcionamiento y desarrollo de la misma usando la tecnología Django. Una vez adquirí los conocimientos necesarios, desarrollé la parte necesaria para el tratamiento de urls de la aplicación web. Una vez eso estaba hecho, me encargué del frontend. En el momento en que todo tenía cierta forma, comencé a desplegar el chat. Tras investigar con la ayuda de Juanjo distintas formas de implementarlo, estuvimos de acuerdo en que fuera una herramienta propia de la aplicación, en vez de una aplicación diferenciada de las dos que ya estábamos desplegando y que podría haber consistido en salas de chat a las que los usuarios se conectarían para dialogar.

Una vez desarrollada una versión básica del chat, intenté investigar una forma que hiciera posible la actualización automática de mensajes mostrados en la conversación según estén o no leídos. Tras probar multitud de opciones, no conseguí los resultados esperados, por lo que decidí pulir los detalles que habían ido mencionando los médicos en las reuniones, ya que consideraba más importantes ciertos cambios propuestos antes que el hecho de que el chat fuera más o menos ágil, dado que de una forma u otra, cumplía su cometido.

6.4 Fernando Hermoso Cara

Al comienzo del proyecto, mi trabajo consistió en recopilar información de aplicaciones similares con el objetivo de obtener tener la mayor cantidad de ideas posibles con las que tener una base.

Tras la primera reunión con el equipo médico, pasé a crear Mockups de la aplicación con el fin de enseñarles un diseño inicial para que vieran ciertos avances en cuanto al diseño. Al mismo tiempo, me encargué de las historias de usuario.

Como se ha comentado anteriormente, en una reunión del equipo al ver la magnitud del proyecto, decidimos que yo me encargaría personalmente de todo el desarrollo de la aplicación android.

Una vez ya teníamos todas las indicaciones y pautas por parte de los médicos,empecé a formarme y familiarizarme con Android como con Android Studio dado que era algo nuevo que nunca había visto. Después de una temporada en este punto y, una vez que adquirí los conocimientos necesarios, empecé con el desarrollo. La fase de desarrollo es la que más tiempo y trabajo me ha llevado.

Primero comencé a desarrollar la parte más visual, las vistas de cada pantalla, ya que todavía no estaban implementadas ni la base de datos ni la API.

Una vez la API estuvo lista, pasé a conectar las vistas con la base de datos a través de la API para mostrar la información. Comencé por la vista de usuario, toma de tensión, el chat y, por último, la vista de medicinas.

Tras varias reuniones con el personal médico e intentando ajustarnos a sus necesidades, la aplicación cambió bastante, tanto en el diseño como en la funcionalidad, por lo que tuve que modificar cosas ya implementadas.

Una vez ya tenía una funcionalidad básica sólo quedaba añadir el sistema de puntos y las notificaciones, siendo estas últimas las que más problemas tuve a la hora de implementar, ya que la documentación que proporciona Android es un poco confusa.

Llegados a este punto, ya tenía desarrollada todas las funcionalidades, por lo que me puse a introducir pequeñas mejoras como el tratamiento de errores si no se añaden todos los campos en un formulario de toma de tensión, si no se respetan los márgenes de presiones arteriales o ventanas de alertas que saltan para confirmar si un paciente desea eliminar una medicación.

Apéndice I: Explicación detallada de los componentes

Consideramos que es interesante entrar más en detalle en ciertos componentes de la arquitectura de nuestra aplicación. Para entrar en más detalle necesitamos emplear diagramas UML (Lenguaje Unificado de Modelado) [58], capturas de código o diagramas similares con más nivel de detalle.

I.1 Ciclo de una petición web

Vamos a empezar describiendo con más detalle cómo Django recibe y envía una petición (HTTP *request*) a la vista correspondiente.

Un mapeador de URLs se suele definir en el módulo *urls.py* de la raíz del proyecto Django, posteriormente, cada paquete o aplicación nueva que tengamos en nuestro proyecto tendrá su propio módulo *urls.py* facilitando la separación de responsabilidades y el mantenimiento del código.

En este ejemplo, vamos a usar el módulo *yatahta.urls.py* que es el mapeador de la aplicación web. El objeto *urlpatterns* del módulo mapea una lista de rutas específicas (patrones URL específicos) y sus correspondientes funciones de vista en el módulo *yatahta.views.py*.

```python
# app_name = 'yatahta'
urlpatterns = [
    path("", views.index, name="index"),
    path("users/", views.UserListView.as_view(), name="user_list"),
    path("patients/<int:pk>/", views.PatientsListView.as_view(), name="patients_list"),
    path("users/<int:pk>/", views.UserDetailView.as_view(), name="user_detail"),
    path("users/<int:pk>/drugs/", views.UserDrugsDetail.as_view(), name="user_drugs"),
    path("users/create/", views.UserCreate.as_view(), name="user_create"),
    path("users/<int:pk>/update/", views.UserUpdate.as_view(), name="user_update"),
    path("users/<int:pk>/delete/", views.UserDelete.as_view(), name="user_delete"),
    path("drugs/", views.DrugListView.as_view(), name="drug_list"),
    path("drugs/<int:pk>/", views.DrugDetailView.as_view(), name="drug_detail"),
    path("messages/<int:pk>/", views.ReadMessageViews.as_view(), name="chat_view"),
    path("doctors/", views.DoctorView.as_view(), name="doctors_view"),
    path("addpatient/", views.NewPatientView.as_view(), name="new_patient"),
]
```

Imagen 65: Captura de pantalla del módulo *yatahta.urls.py.*

El objeto *urlpatterns* es una lista de funciones *path* en la que el primer argumento es la ruta (patrón) que se emparejará. El segundo argumento de la función *path* es la función de *callback* que será llamada cuando se produzca el emparejamiento entre la URL de la petición y la del primer argumento de la función *path*.

Por ejemplo, se puede apreciar que la URL “vacía” (el primero elemento de la lista *urlpatterns*) se mapea a la función *index* en el módulo *yatahta.views.py*. Esta función es muy

simple, se puede observar como utiliza la API del ORM de Django para extraer información de la base de datos y mostrar unas estadísticas globales. También se declaran algunas variables de sesión que facilitan la implementación y la construcción de la aplicación web. Devuelve una *HttpResponse* con la información a través del uso de la función *render* del módulo *django.shortcuts*. Tal y como mostramos en la siguiente imagen:

```python
@login_required
def index(request):
    """Vista para la página principal de la aplicación."""

    # Genera algunos contadores de los modelos que tenemos
    num_users = User.objects.count()
    num_drugs = Drug.objects.count()
    num_pressures = Pressure.objects.count()
    num_posologies = Posology.objects.count()
    num_messages = Message.objects.count()
    clasification_list = User.objects.all().order_by("-points")[:15]
    request.session["doctor"] = {"name": request.user.username, "id": request.user.id}
    context = {
        "num_users": num_users,
        "num_drugs": num_drugs,
        "num_pressures": num_pressures,
        "num_posologies": num_posologies,
        "num_messages": num_messages,
        "clasification_list": clasification_list,
    }

    # Render the HTML template index.html with the data in the context variable
    return render(request, "yatahta/index.html", context=context)
```

Imagen 66: Captura de pantalla de la función *index* del módulo *yatahta.views.py.*

Sin embargo, el resto se mapean a clases dentro del módulo *yatahta.views.py,* por ejemplo, el segundo elemento de la lista *urlpatterns* se mapea a la clase *UserListView* del mismo módulo. En este caso sobreescribimos la función *get_queryset* para obtener los datos tal y como nos interesan a nosotros, simplemente ordenados por nombre de usuario en lugar de id. En el caso de las clases es transparente al programador toda la lógica de cómo se obtienen los datos, se cargan plantillas, etc. Lo realiza todo Django. Nosotros nos tenemos que ceñir a sobreescribir lo que queremos personalizar, como es el caso de los atributos de clase como *model, template_name, context_object_name* y *paginate_by*, que son bastante autodescriptivos tal y como mostramos en la siguiente imagen:

```python
class UserListView(UserPassesTestMixin, generic.ListView):
    """
    Vista que muestra el listado de usuarios que hay registrados en la aplicación.
    Al usar la función de test para la vista que comprueba si es staff o no
    ya se da por incluído que el usuario está logueado.
    """

    model = User
    template_name = "yatahta/user_list.html"
    context_object_name = "user_list"
    paginate_by = 20

    def get_queryset(self):
        """Devuelve la lista de usuarios"""
        return User.objects.order_by("username")

    def test_func(self):
        """Para que los médicos puedan ver el listado de pacientes."""
        return self.request.user.is_staff
```

Imagen 67: Captura de pantalla de la clase *UserListView* del módulo *yatahta.views.py.*

El parámetro con nombre *name* de la función *path* sirve para acortar las URLs en el código y poder referirnos a ellas por nombre.

En la siguiente imagen vemos la plantilla que utiliza Django, en este caso *yatahta.user_list.html*:

```
{% extends 'base.html' %}

{% block content %}
  <h1>Lista de usuarios</h1>
  {% if user_list %}
  <ul>
    {% for user in user_list %}
      <li>
        {{ user.patient_code }} - <a href="{{ user.get_absolute_url }}">{{ user.get_username }}</a>
      </li>
    {% endfor %}
  </ul>
  {% else %}
    <p>No hay usuarios.</p>
  {% endif %}
{% endblock %}
```

Imagen 68: Captura de pantalla de la plantilla *yatahta.user_list.html.*

Finalmente, siempre se devuelve una instancia de *HttpResponse* con un contenido de tipo *string*. En este string se incluyen las cabeceras HTTP completas, donde se marca el tipo de contenido (casi siempre html) y el código de estado de la respuesta.

I.2 Ciclo de una petición REST

El flujo de una petición REST, como hemos mencionado en el punto anterior, es muy parecido al de una petición web. La diferencia fundamental radica en que en lugar de utilizar plantillas HTML utiliza serializadores. No obstante vamos a remarcar las diferencias a lo largo de este punto.

Los puntos de entrada de la API los marca el módulo *api.urls.py*, en este caso hemos hecho uso de una facilidad que aporta DRF, los *Routers*. Estos son una clase que abstraen completamente y hacen transparente la creación de las URLs. El proceso de emparejamiento entre una URL y las vistas es manejado automáticamente por DRF. Esto aporta dos ventajas, por un lado aporta consistencia a la API y el programador no se tiene que preocupar por el diseño de las URLs y, por otro lado, crea automáticamente la vista raíz de la API por nosotros. DRF crea automáticamente una API web navegable que facilita mucho el testeo y la depuración de la API.

Para utilizar los *Routers* tan solo tenemos que registrar una vista apropiada, en este caso de una clase *ViewSet* o derivada y el *Router* genera automáticamente las URLs. Esto podemos verlo en la siguiente captura de pantalla:

```
# Create a router and register our viewsets with it.
router = DefaultRouter()
router.register(r"users", UserViewSet)
router.register(r"drugs", DrugViewSet)
router.register(r"posologies", PosologyViewSet)
router.register(r"messages", MessageViewSet)
router.register(r"pressures", PressureViewSet)
router.register(r"moriskys", MoriskyViewSet)
router.register(r"batles", BatlesViewSet)

# The API URLs are now determined automatically by the router.
urlpatterns = [
    path("", include(router.urls)),
]
```

Imagen 69: Captura de pantalla del módulo *api.urls.py.*

Los *ViewSets* son clases que fundamentalmente ahorran mucho código, generan automáticamente, junto con los *Routers*, todas las URLs y proporcionan operaciones que permiten realizar las típicas operaciones CRUD.

```python
class DrugViewSet(viewsets.ModelViewSet):
    queryset = Drug.objects.all()
    pagination_class = None
    serializer_class = DrugSerializer
    filter_backends = [
        filters.SearchFilter,
        django_filters.rest_framework.DjangoFilterBackend,
    ]
    search_fields = ["name"]
    filterset_fields = ["name"]
```

Imagen 70: Captura de pantalla la clase *DrugViewSet* del módulo *api.views.py.*

Como se puede apreciar en la captura, de nuevo, sobreescribimos lo que nos interesa. En este caso hacemos uso de la API del ORM de Django para obtener todas las instancias de la clase *Drug*. Lo realmente importante es donde indicamos el serializador que vamos a utilizar, sobreescribiendo el atributo de clase *serializer_class* indicamos el serializador.

Los serializadores son los encargados de deserializar la información que recibimos a través de la API y que viene en formato JSON para transformarla en objetos y variables de "tipo Python" y, por tanto, poder trabajar con ellas. A la inversa, cuando queremos mandar información en la respuesta, se encargan de serializar esta información, de modo que la transforma de Python a JSON. A continuación mostramos el serializador *DrugSerializer*:

```python
class DrugSerializer(serializers.ModelSerializer):
    You, 3 months ago | 1 author (You)
    class Meta:
        model = Drug
        fields = ["id", "name", "dosage", "photo", "__str__"]
```

Imagen 71: Captura de pantalla la clase *DrugViewSet* del módulo *api.views.py.*

De nuevo, la abstracción que aporta el *framework* hace que la cantidad de código a escribir sea mínima. En este caso tenemos un serializador que mapea completamente el modelo base (*yatahta.models.Drug)* y al que simplemente tenemos que indicarle cuál es este modelo y los campos que queremos que muestre en las respuestas de la API.

Finalmente, siempre se devuelve una instancia de *HttpResponse* con un contenido de tipo *string*. En este string se incluyen las cabeceras HTTP completas, donde se marca el tipo de contenido (JSON) y el código de estado de la respuesta.

I.3 Seguridad de la aplicación

A continuación mostramos capturas de pantalla testeando la API desde el programa *Postman* [59]. Esta captura corresponde al proceso de *login*:

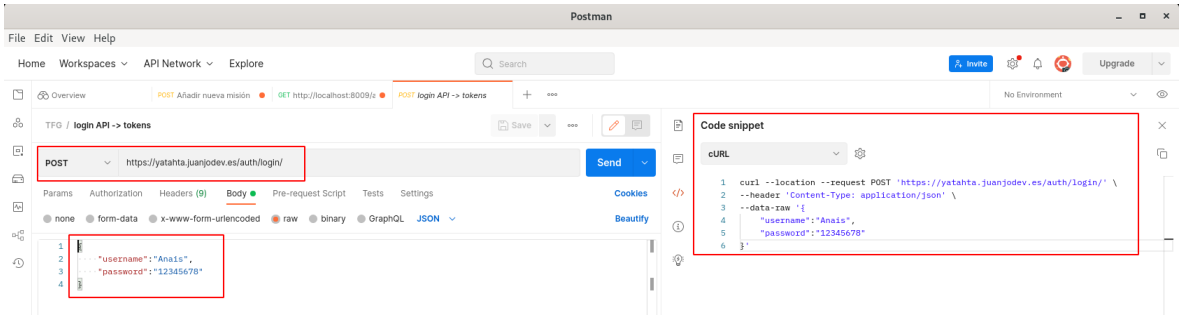


Imagen 72: Ejemplo de petición POST para loguear un usuario. Se señalan el punto de acceso, el formato *raw* del JSON que se envía, y el comando *cURL*.

Ahora vemos la respuesta a la petición anterior:

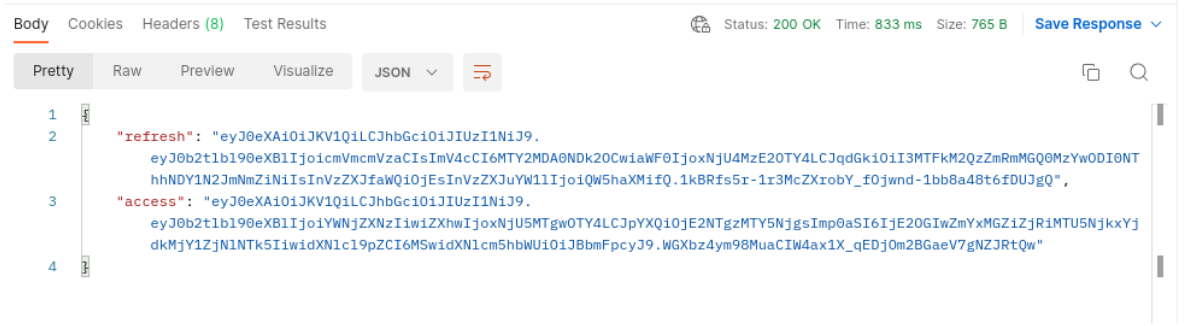


Imagen 73: Respuesta con los *tokens* de acceso y de refresco en formato JSON.

La siguiente captura corresponde al proceso de cambio de la contraseña, en la que se debe acompañar la petición de la cabecera de autenticación por token:

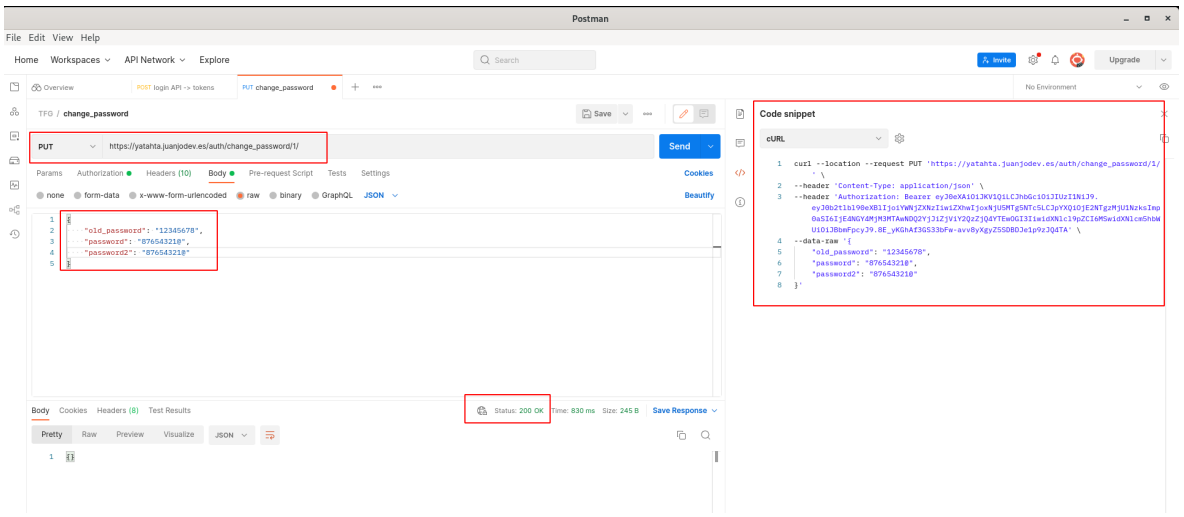


Imagen 74: Petición PUT para actualizar la contraseña de un usuario. Se ha señalado el tipo de petición y URL destino, el contenido en JSON, la respuesta del servidor y el comando *cURL* donde se aprecia que se introduce en la cabecera el *token* de acceso.

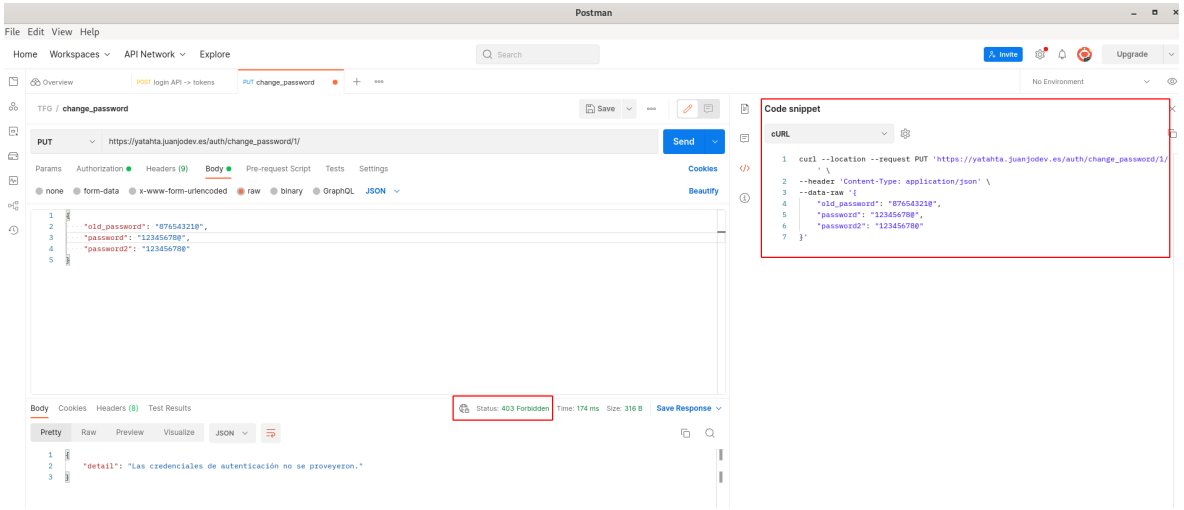


Imagen 75: Petición PUT para actualizar la contraseña de un usuario. En este caso no se aporta el token de acceso y se muestra el error devuelto por el servidor.

Con estos ejemplos es suficiente para demostrar el funcionamiento de JWT.

Por su parte, la aplicación Android, que es la que hace uso de la API REST, implementa la lógica de renovación de los tokens cuando estos han caducado. Cada vez que el usuario inicia la aplicación se comprueba si el token ha expirado, para ello se coge el token de SharedPreferences , se le aplica un decoded para sacar el texto en claro y se comprueba el timestamp de la fecha de expiración.

```
if (!access.equals("")){
    decodeAccess = JWTUtils.decoded(access);
    if(JWTUtils.isAccessTokenExpired(decodeAccess)){
        newToken();
    }
}
```

Imagen 76: Comprobación para la renovación del token.

Si se ha superado el tiempo de expiración , se realiza una nueva petición POST a la siguente URL: “https://yatahta.juanjodev.es/auth/login/refresh/” con el token de *refresh* que tenemos guardado también en *SharedPreferences*.

```
private void newToken (){
    JSONObject param = new JSONObject();
    try {
        refresh = preferences.getString( key: "refresh", defValue: "");
        param.put( name: "refresh",refresh);
    } catch (JSONException e) {
        e.printStackTrace();
    }
    JsonObjectRequest request = new JsonObjectRequest(Request.Method.POST, URL_REFRESH, para
            new Response.Listener<JSONObject>() {
                @Override
                public void onResponse(JSONObject response) {
                    try {
                        String access = response.getString( name: "access");
                        String refresh= response.getString( name: "refresh");
                        editor.putString("access", access);
                        editor.putString("refresh", refresh);
                        editor.apply();
                        closeSession();
                        startActivity(new Intent(context, Login.class));

                    } catch (JSONException e) {
                        e.printStackTrace();
                    }
                }
            },
            new Response.ErrorListener() {
                @Override
                public void onErrorResponse(VolleyError error) {
                    if(error instanceof ServerError){
                        Log.d( tag: "TAG", msg: "Error en servidor");
                    }
                    if(error instanceof NoConnectionError){//no conexion internet
                        Log.d( tag: "TAG", msg: "No hay conexion a internet");
                    }
                    if(error instanceof NetworkError){//desconexionsocket
                        Log.d( tag: "TAG", msg: "NetWorkError");
                    }
                }
            });
    VolleySingleton.getInstaceVolley(getApplicationContext()).addToRequestQueue(request);
}
```

Imagen 77: Petición POST para la renovación del token.

La respuesta que obtendremos es un objeto JSON con nuestros nuevos tokens de *access* y *refresh*.

I.4 Documentación de la API

En este apartado explicamos con más detalle el funcionamiento de la API, para ello nos ayudamos de capturas de pantalla realizadas a la API web que facilita DRF. La raíz de la API web nos muestra las URLs raíz de cada punto de acceso y se conforman de la siguiente manera:

```
GET /api/
```

```
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "users": "http://yatahta.juanjodev.es/api/users/",
    "drugs": "http://yatahta.juanjodev.es/api/drugs/",
    "posologies": "http://yatahta.juanjodev.es/api/posologies/",
    "messages": "http://yatahta.juanjodev.es/api/messages/",
    "pressures": "http://yatahta.juanjodev.es/api/pressures/",
    "moriskys": "http://yatahta.juanjodev.es/api/moriskys/",
    "batles": "http://yatahta.juanjodev.es/api/batles/"
}
```

Imagen 78: Raíz de la API web de DRF, URL: https://yatahta.juanjodev.es/api/.

Cuando se accede mediante la API web a la raíz de la API, o se hace una petición GET a este mismo punto de acceso, se obtienen las URLs de los distintos puntos de acceso a las entidades de nuestra aplicación. Cabe destacar que en cada punto de acceso, el campo "*Allow*" nos indica los verbos HTTP permitidos para dicho punto de acceso.

En las direcciones que se muestran en la imagen se obtendrá un JSON cuyo contenido lo compone un listado con los registros de esa tabla, con una paginación de 10 registros por página. Por ejemplo, en la siguiente imagen mostramos el resultado devuelto por una petición GET al punto de acceso "*/api/posologies/*":

```
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "count": 253,
    "next": "http://yatahta.juanjodev.es/api/posologies/?page=2",
    "previous": null,
    "results": [
        {
            "id": 3,
            "drug_name": "Amlodipino",
            "at_breakfast": null,
            "at_lunch": null,
            "at_dinner": null,
            "at_bedtime": null,
            "start_time": "05:00:00",
            "start_date": "2022-07-31",
            "end_date": "2023-04-07",
            "x_times": 0,
            "frequency": "DM",
            "days_of_week": null,
            "days_of_month": null,
            "months_of_year": null,
            "user": 103,
            "drug": 152
        },
        {
            "id": 4,
            "drug_name": "Augmentine",
            "at_breakfast": null,
            "at_lunch": null,
            "at_dinner": null,
            "at_bedtime": null,
            "start_time": "07:00:00",
            "start_date": "2022-04-15",
            "end_date": "2023-01-14",
```


Imagen 79: Muestra de retorno de /api/posologies/.

Sin embargo, hay una excepción en el enlace “users”. Además de devolver los valores propios de esa fila de la base de datos, devuelve a la vez el listado de presiones del usuario en cuestión. Esto es así para facilitar el proceso de obtener la información necesaria de forma que, en el momento en que el paciente inicia la aplicación móvil, sólo sea necesaria una llamada para mostrar la información de la página principal.

Aunque lo lógico es que esto se hiciera en dos peticiones distintas, por comodidad nuestra y por construcción de la aplicación, hemos decidido implementarlo así porque a este punto de acceso sólo acceden los usuarios de la aplicación móvil desde su teléfono. Como hemos descrito anteriormente, esto facilita la carga inicial de la gráfica que mostramos la vista Tensión arterial.

En la siguiente imagen se muestra el resultado devuelto por una petición GET al punto de acceso “*/api/users/*”, se puede observar que los dos primeros resultados corresponden a

personal sanitario (campo "*is_staff*" a *true* y campo "*doctor*" *a null*) y el tercer registro muestra el inicio del listado de presiones tomadas (campo *"pressurelist"*):


```
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "count": 145,
    "next": "http://yatahta.juanjodev.es/api/users/?page=2",
    "previous": null,
    "results": [
        {
            "id": 2,
            "url": "http://yatahta.juanjodev.es/api/users/2/",
            "patient_code": "2",
            "username": "Alice",
            "password": "pbkdf2_sha256$260000$UNnrVZ60HXqbwADOkIWd29$OoTcU300Of5ZnwCQvL3AxlE12PckNOBGiq2pO3AT/P8=",
            "is_staff": true,
            "points": 17,
            "birthday": "1955-03-15",
            "doctor": null,
            "pressurelist": []
        },
        {
            "id": 1,
            "url": "http://yatahta.juanjodev.es/api/users/1/",
            "patient_code": "1",
            "username": "Anais",
            "password": "pbkdf2_sha256$260000$4qFEVlzazq3SVrOfB4Z95j$ngydahPGmod2mCRI1PJwPtuaQBR1r4EG8EeAgVjXNVo=",
            "is_staff": true,
            "points": 43,
            "birthday": "1971-01-14",
            "doctor": null,
            "pressurelist": []
        },
        {
            "id": 76,
            "url": "http://yatahta.juanjodev.es/api/users/76/",
            "patient_code": "G15ZzoqcQO",
            "username": "Andy",
            "password": "pbkdf2_sha256$260000$jkUJC4NIAi8nq9qFLkDiy7$wcLAWOk6Suk/bDR2o9RIwSxAuwAhTaLtKcDrXuOyKXA=",
            "is_staff": false,
            "points": 65,
            "birthday": "1945-05-06",
            "doctor": 4,
            "pressurelist": [
                {
                    "id": 3358,
                    "systolic": 136,
                    "diastolic": 82,
                    "arm": "RT",
                    "bpm": 66,
                    "date": "2022-01-15T00:00:00+01:00",
                    "user": 76
                },
                {
                    "id": 3359,
                    "systolic": 118,
```


Imagen 80: Muestra de retorno de /api/users/.

Si se desea hacer un POST en este enlace, debe entonces indicarse en la petición HTTP enviada a la API, y acompañar dicha petición de un JSON con un formato como el siguiente:

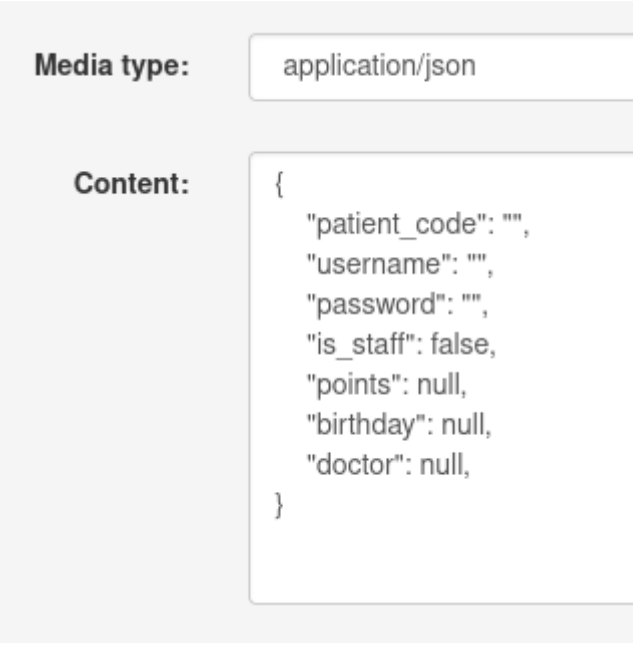


Imagen 81: Formato JSON para petición POST de “user”.

Teniendo esto en cuenta, como es de suponer, se da la opción de obtener información de cada fila en concreto para la base de datos si añadimos a la dirección el identificador del registro que se necesite.

```
GET /api/users/2/
```

```
HTTP 200 OK
Allow: GET, PUT, PATCH, DELETE, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "id": 2,
    "url": "http://yatahta.juanjodev.es/api/users/2/",
    "patient_code": "2",
    "username": "Alice",
    "password": "pbkdf2_sha256$260000$UNnrVZ60HXqbwADOkIWd29$OoTcU300Of5ZnwCQvL3Ax1El2PckNOBGiq2pO3AT/P8=",
    "is_staff": true,
    "points": 17,
    "birthday": "1955-03-15",
    "doctor": null,
    "pressurelist": []
}
```

Imagen 82. Muestra de retorno de /api/users/<id>/.

En general todos los puntos de acceso tienen un funcionamiento idéntico por lo que no vamos a extendernos más en su explicación.

Apéndice II: Aspectos visuales

II.1 Bocetos

II.1.1 Aplicación móvil

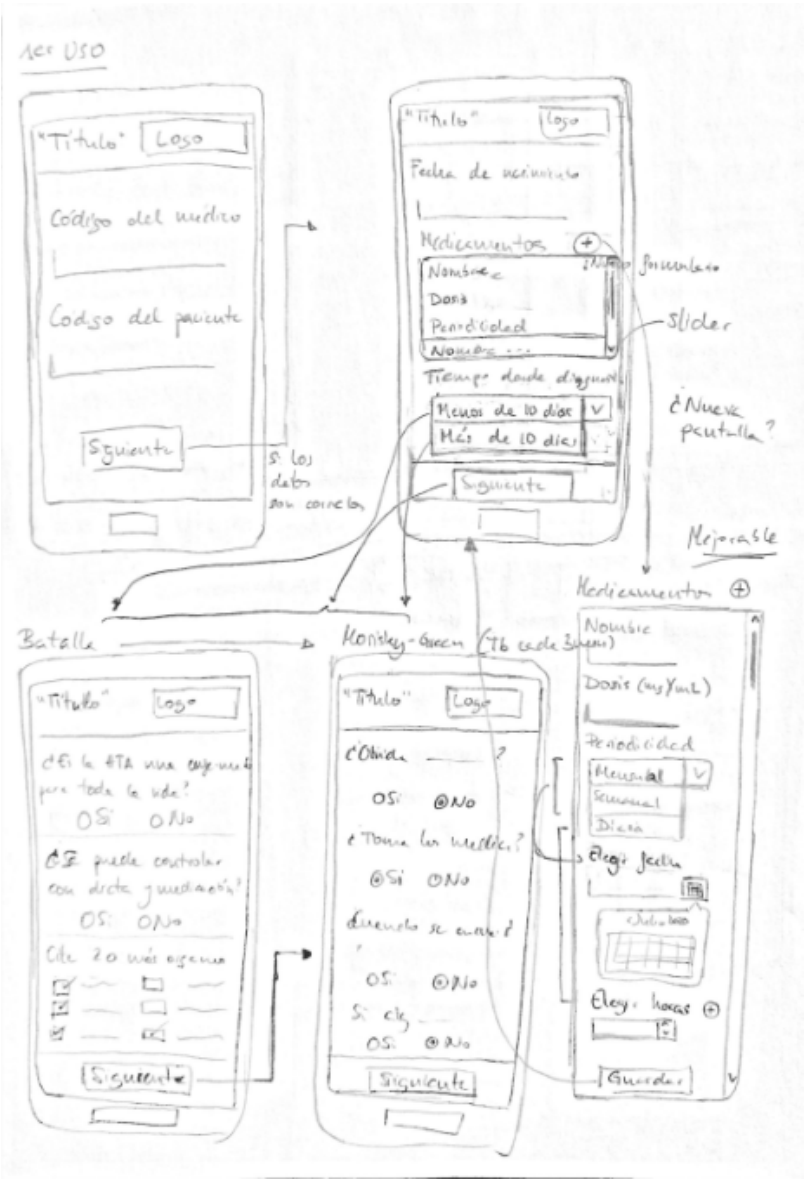


Imagen 83: Boceto a mano alzada del primer uso de la aplicación móvil.

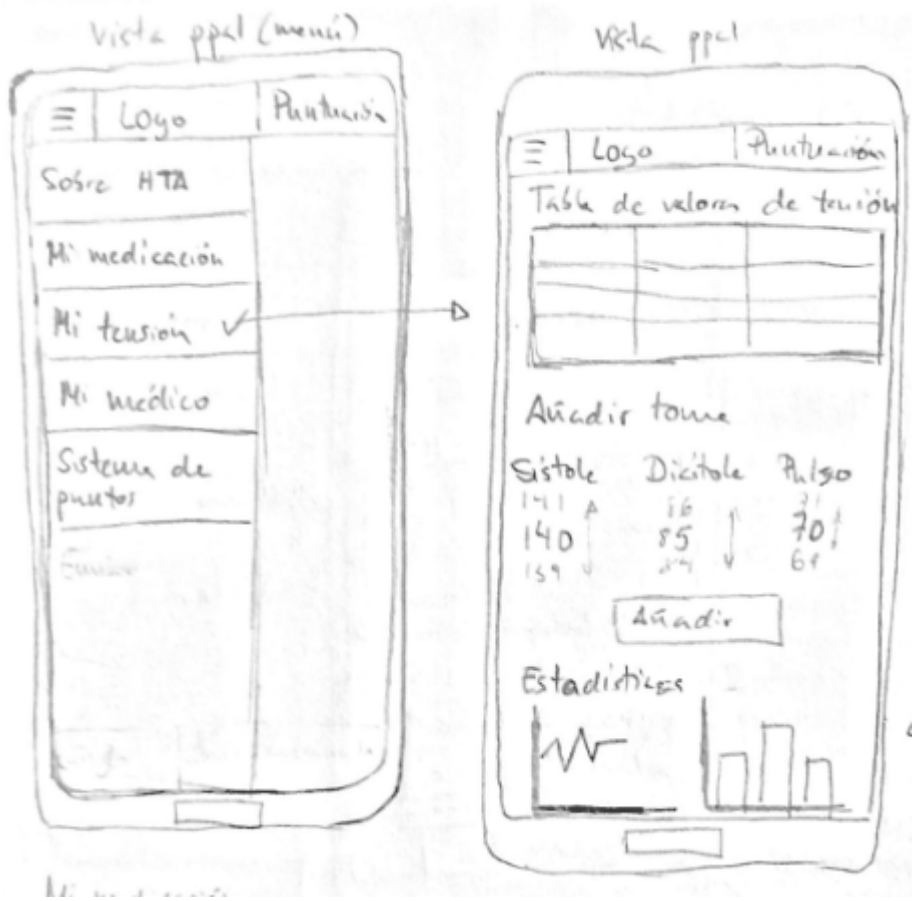


Imagen 84: Boceto a mano alzada de menú y vista principal de la aplicación.

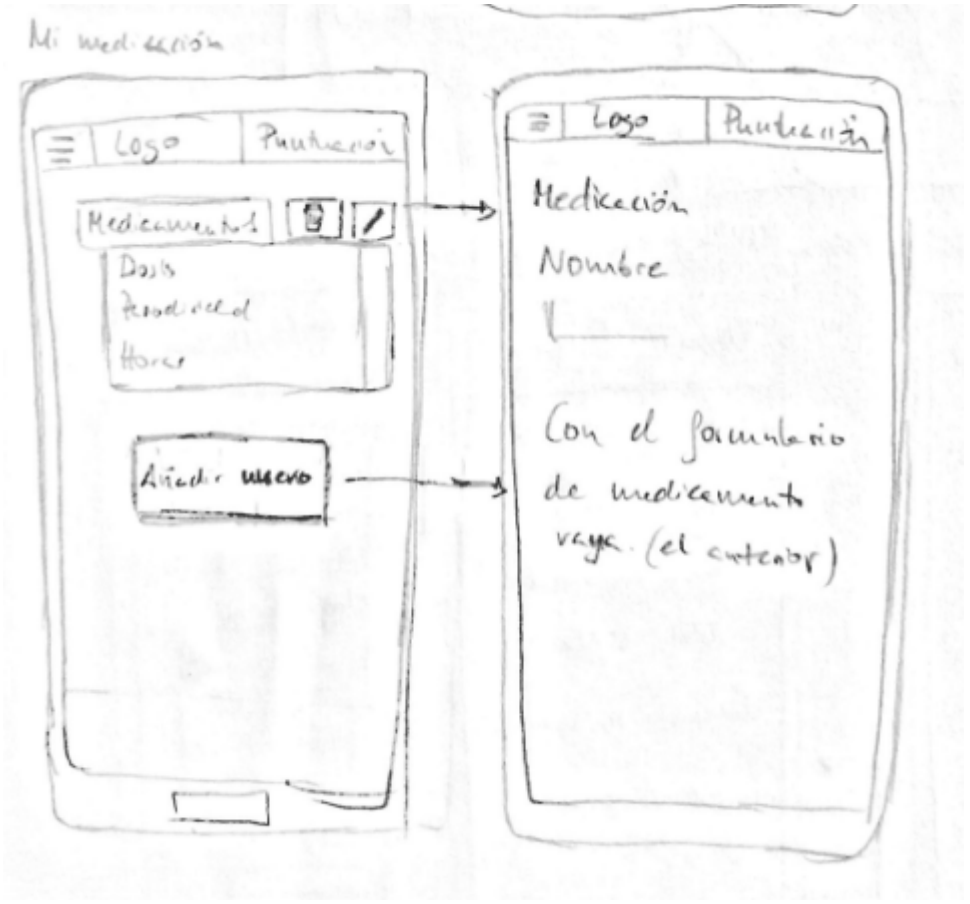


Imagen 85: Boceto a mano alzada de la pestaña de medicación de la aplicación.

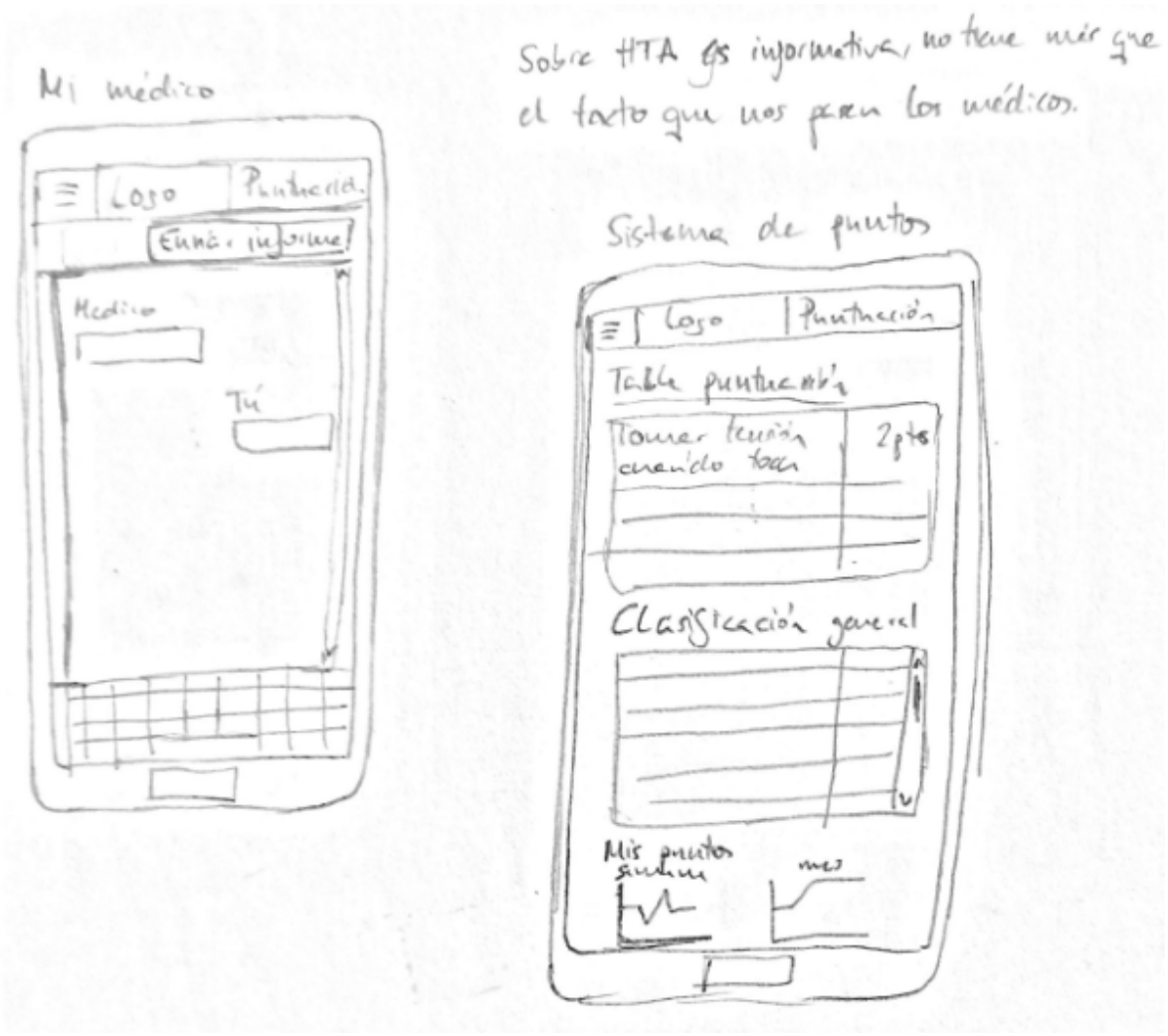


Imagen 86: Boceto a mano alzada de la vista del chat con el médico y el sistema de puntuación de la aplicación.

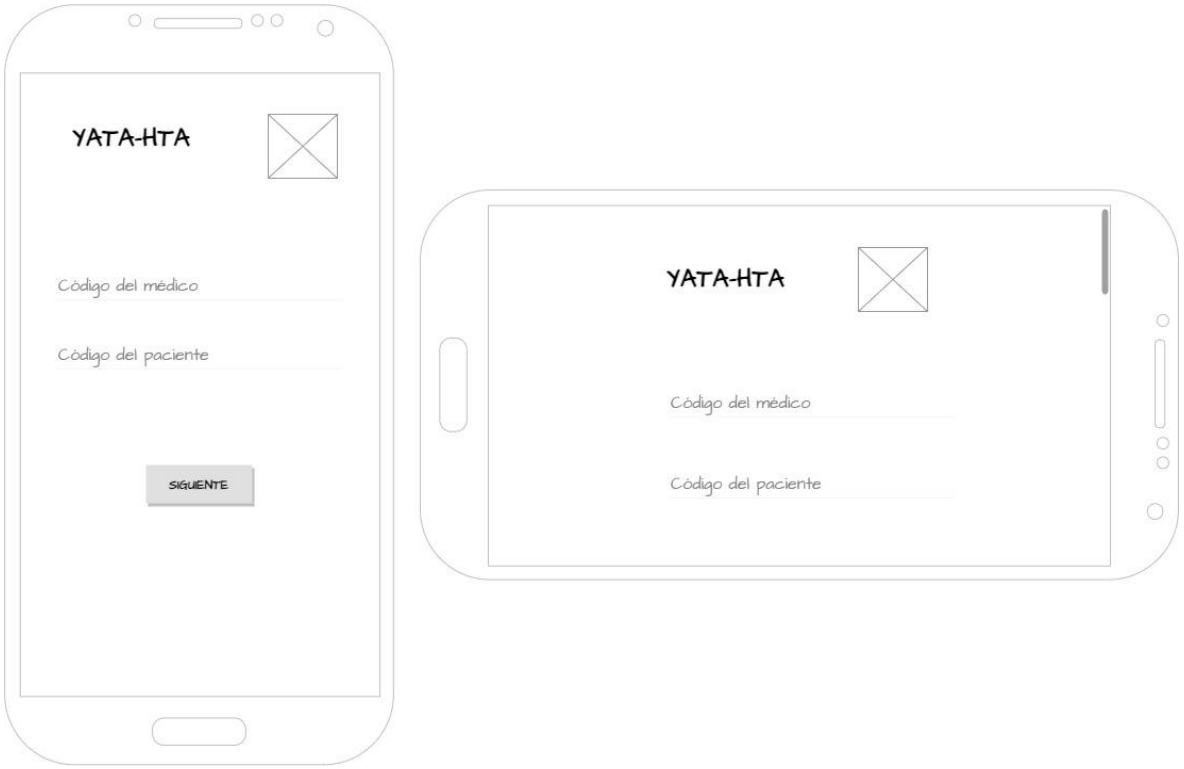


Imagen 87: Pantalla de registro de nuevos pacientes - vista 1.

Imagen 88: Pantalla de registro de nuevos pacientes - vista 2.

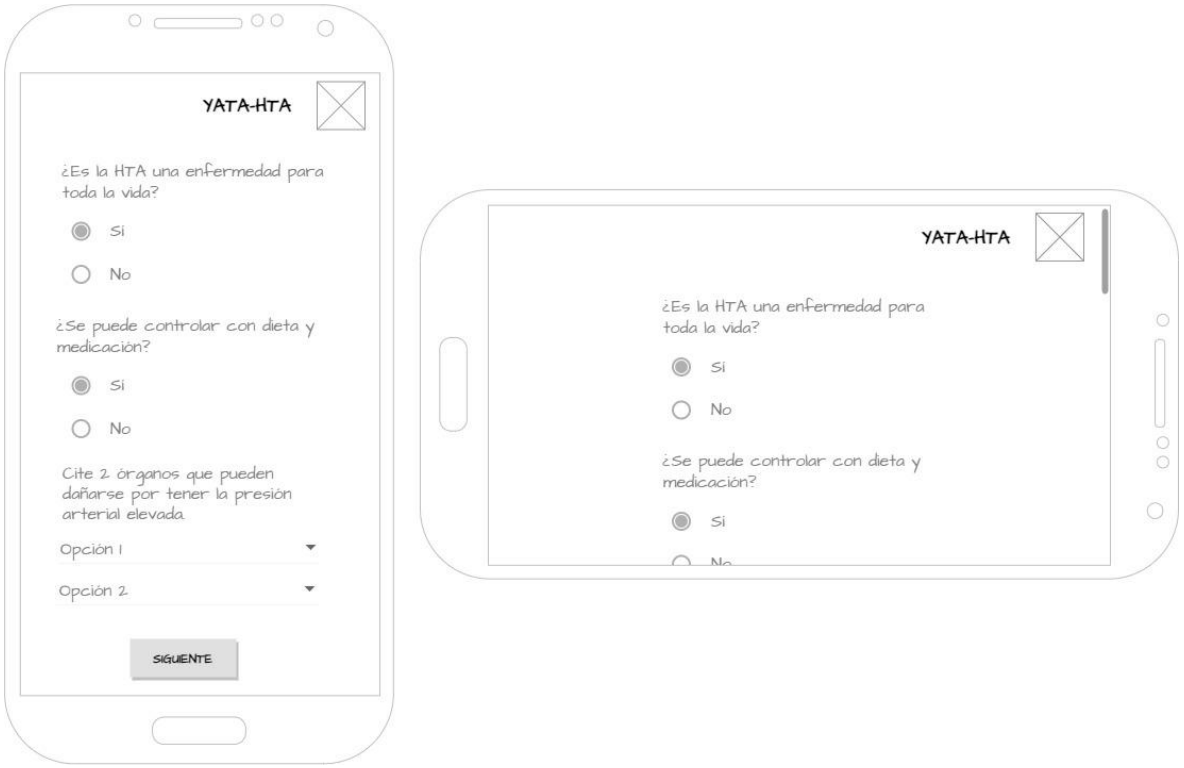


Imagen 89: Pantalla de test de Batalla.

Imagen 90: Pantalla de test de Morisky-Green.

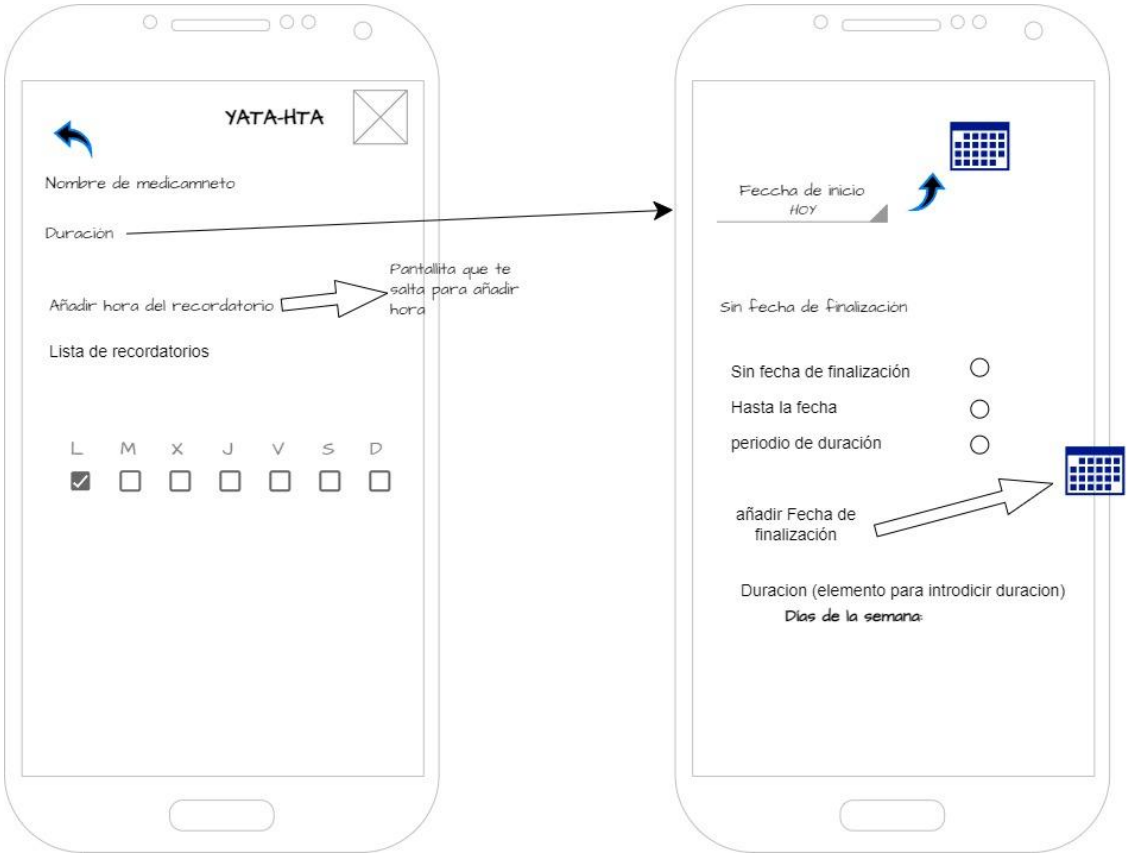


Imagen 91: Pantalla para añadir la posología de un medicamento.

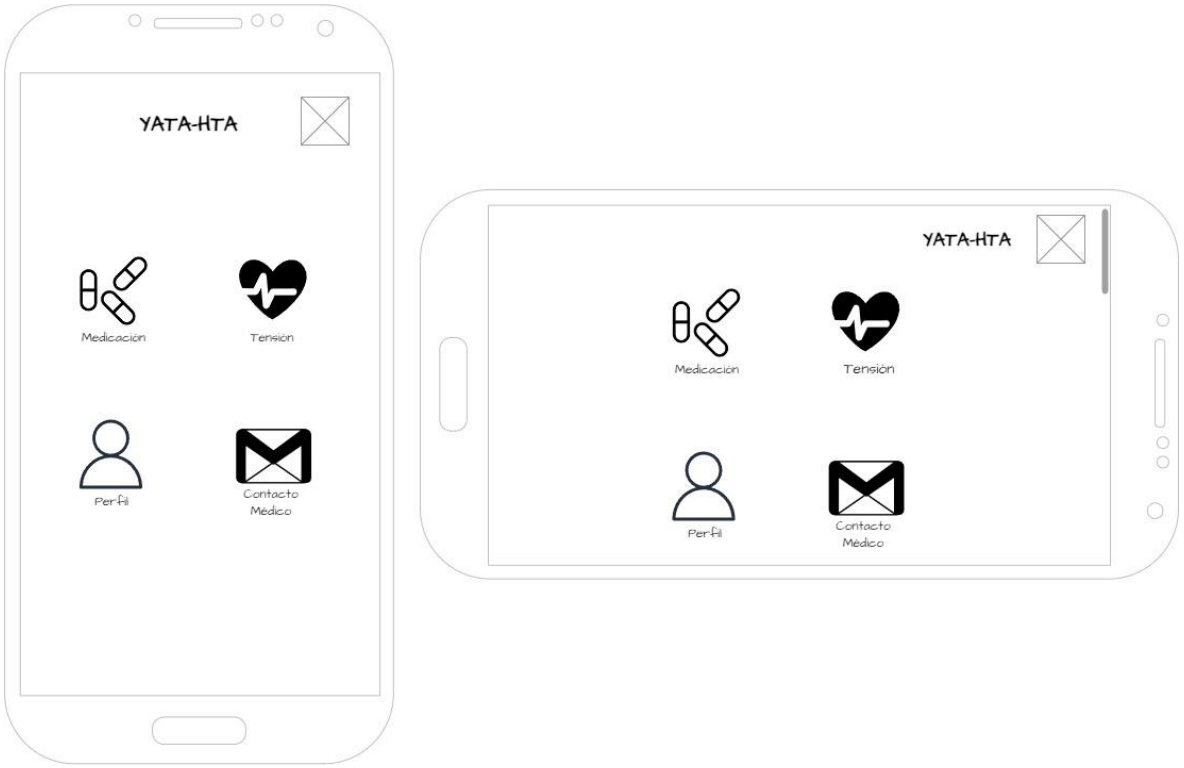


Imagen 92: Pantalla propuesta vista inicial de la aplicación.

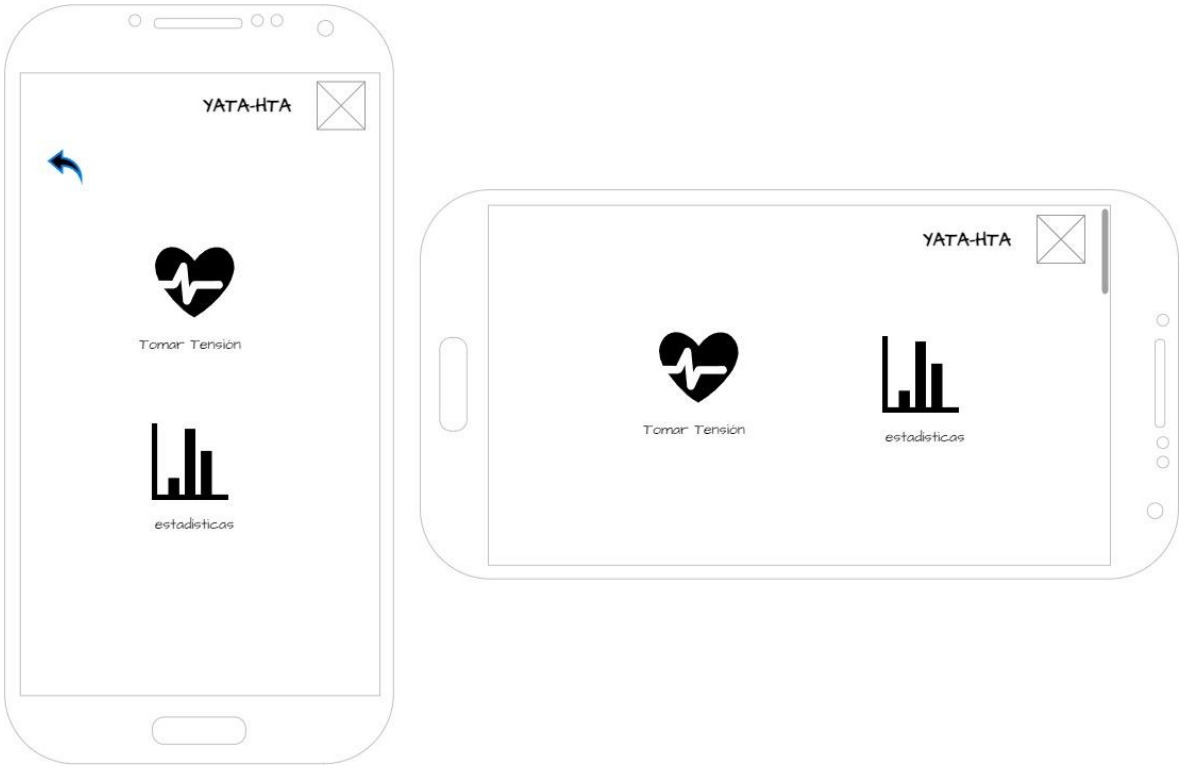


Imagen 93: Pantalla propuesta inicial de submenú para añadir nuevas tensiones y ver estadísticas de tensión para un usuario.

Imagen 94: Ejemplo del boceto de la vista que permite registrar una nueva toma de tensión, y cómo la comparamos con una de las aplicaciones anteriores.

Imagen 95: Ejemplo del boceto de la vista que muestra la vista principal de los medicamentos que toma un usuario y el paso a la vista que permite registrar un nuevo medicamento.

II.1.2 Aplicación web

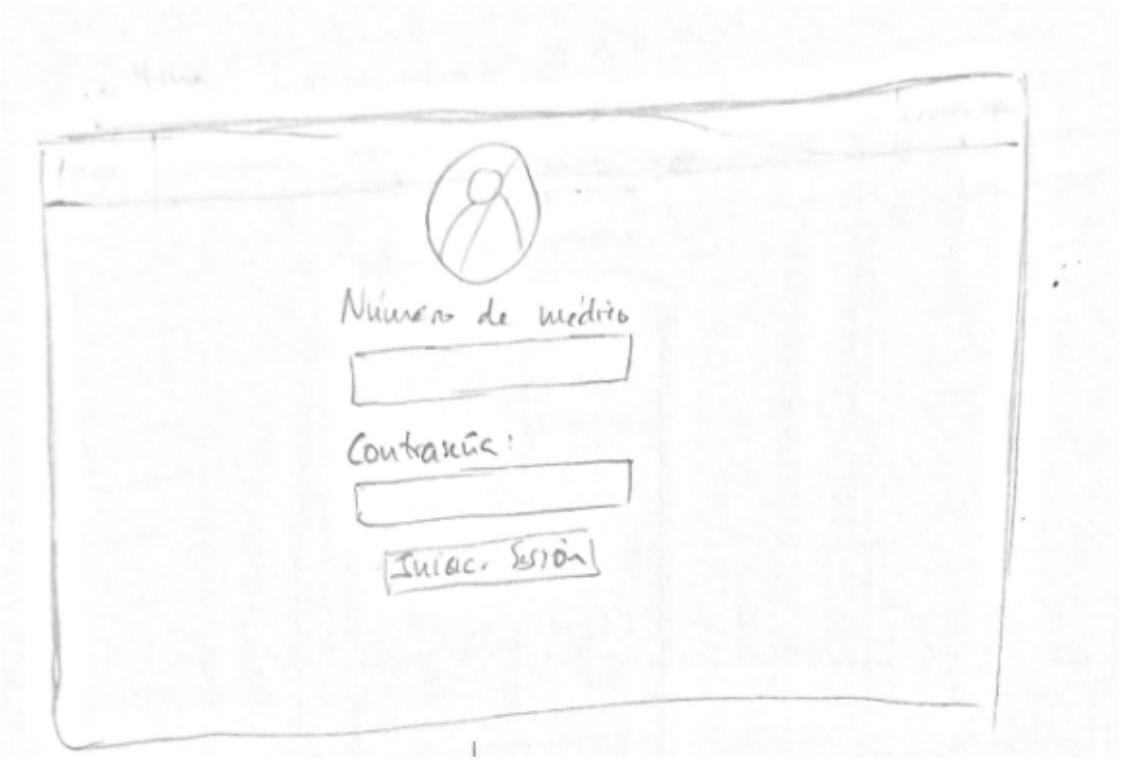


Imagen 96: Boceto a mano alzada de login de la aplicación web.

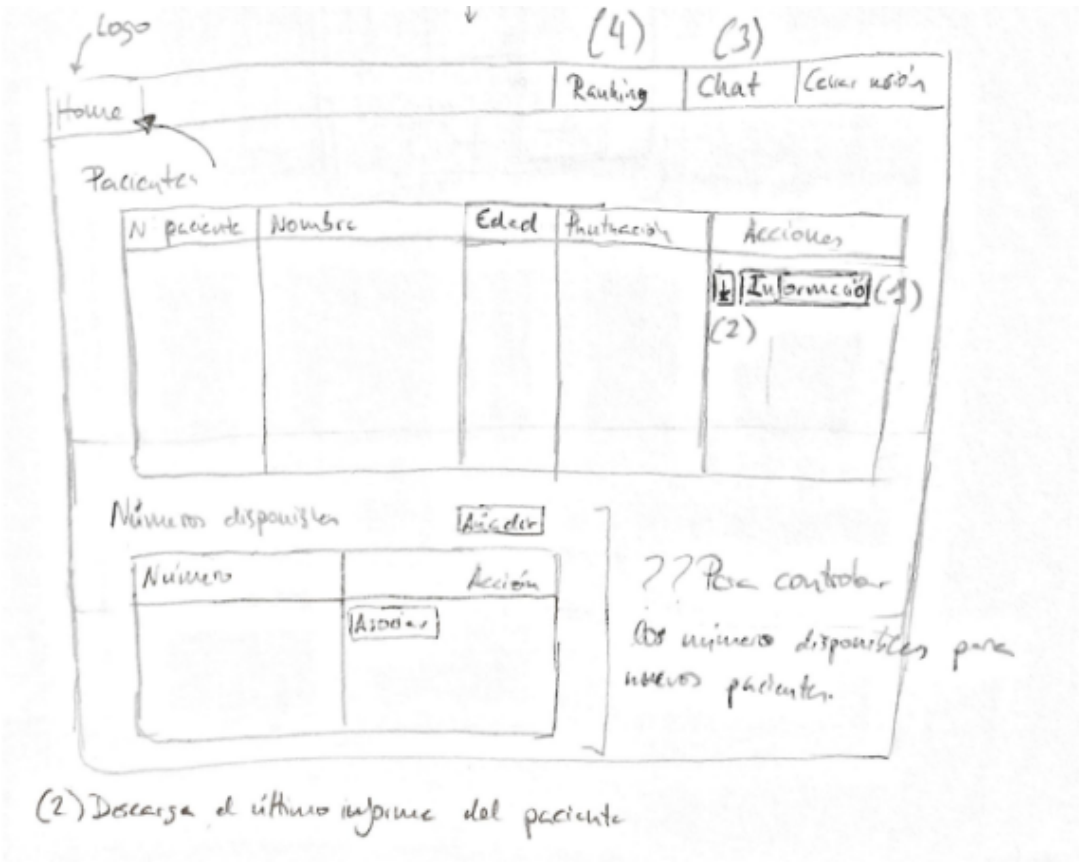


Imagen 97: Boceto a mano alzada de vista principal de la aplicación móvil. Dado que fue solicitud de los médicos no tener que añadir ellos mismos de la aplicación web a nuevos pacientes, esta pantalla tuvo cambios notables.

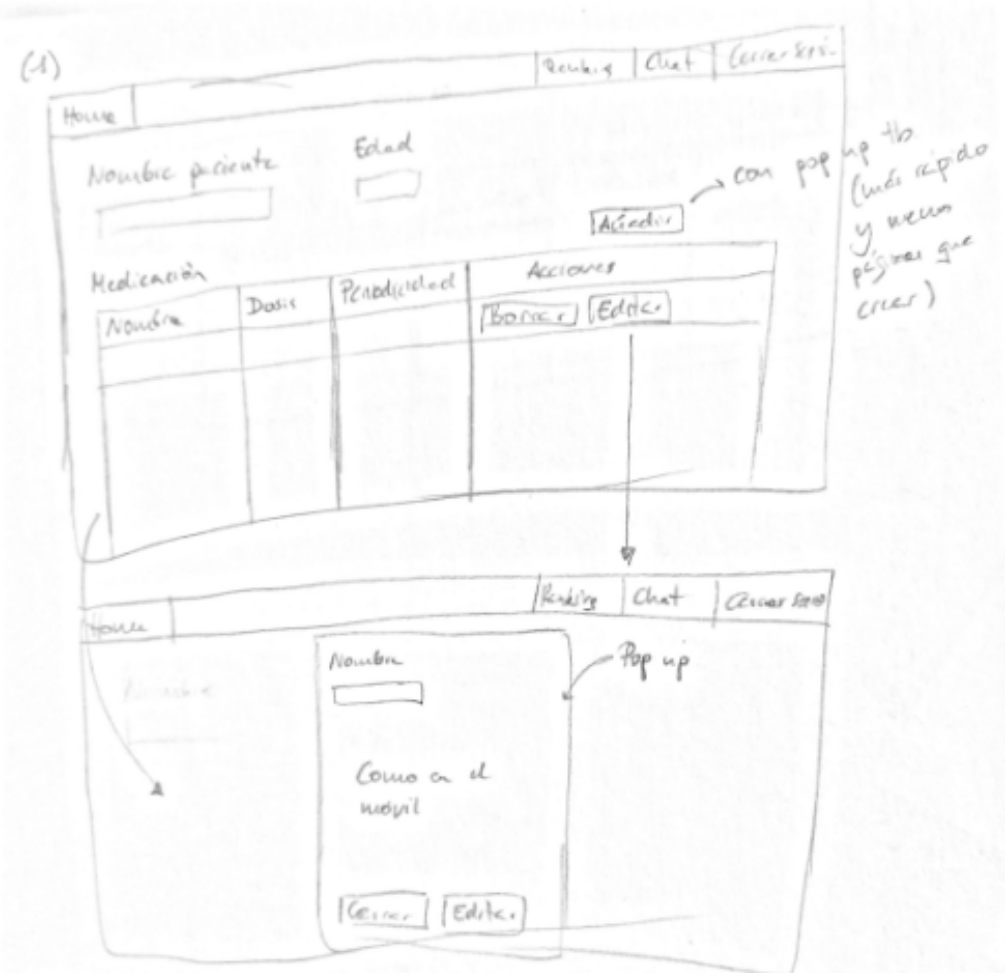


Imagen 98: Boceto a mano alzada del flujo de uso de la posología de un paciente.

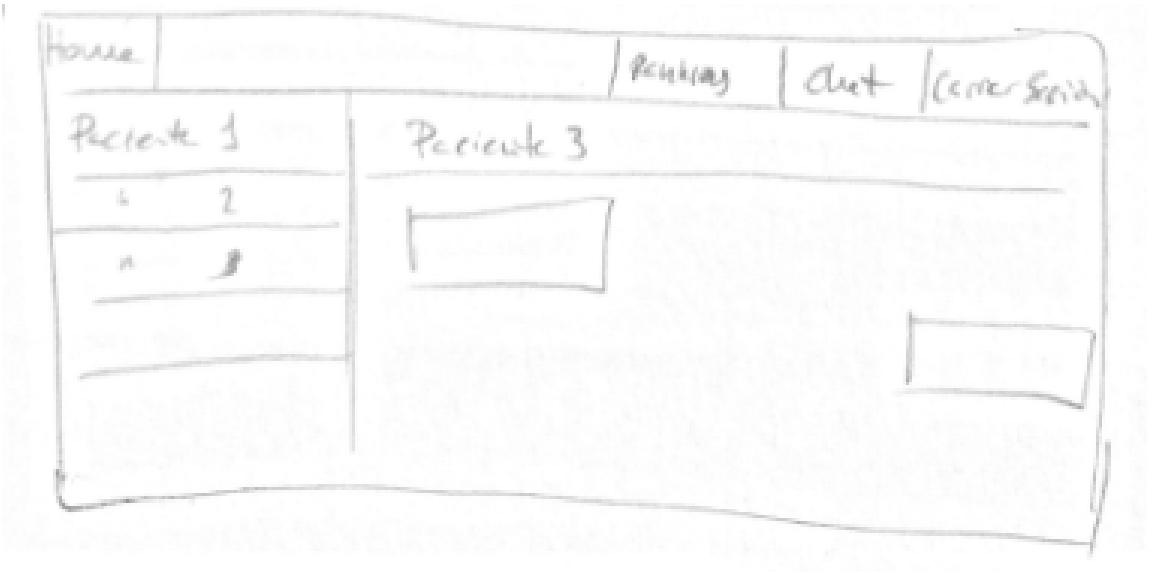


Imagen 99: Boceto a mano alzada del chat de la aplicación. A la izquierda una lista con los pacientes asociados al médico. A la derecha, la conversación con el paciente seleccionado en el listado.

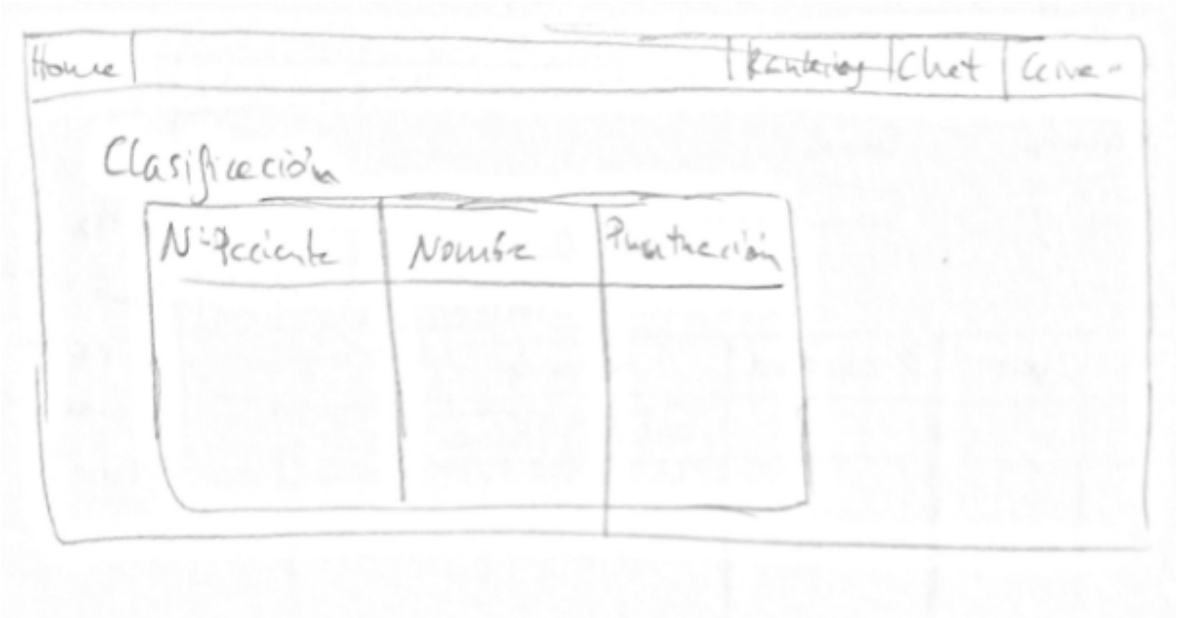


Imagen 100: Boceto a mano alzada de la clasificación general por puntuación de la aplicación.

Apéndice III: Historias de usuario

III.1 Login para usuarios registrados

Login para usuarios registrados (V1 entrega medicos)

Historia de usuario:

Como... usuario con rol "Paciente" de la aplicación

Quiero... poder acceder a la aplicación

Para... poder llevar el seguimiento de mi tensión arterial y la medicación que tomo

Los usuarios previamente registrados y con el código proporcionado por su médico podrán acceder a la aplicación. Una vez dentro de la aplicación podrán registrar sus tomas de tensión y la medicación actual que siguen en tratamiento.

- [x] Diseñar la interfaz con un wireframe
- [x] Realizar el diagrama de flujo o algoritmo de este proceso
- [ ] Implementar la interfaz en un layout
- [ ] Crear la clase Java que implementa la lógica de registro (obtiene datos del layout y los guarda en la BBDD)

Diagrama de flujo:

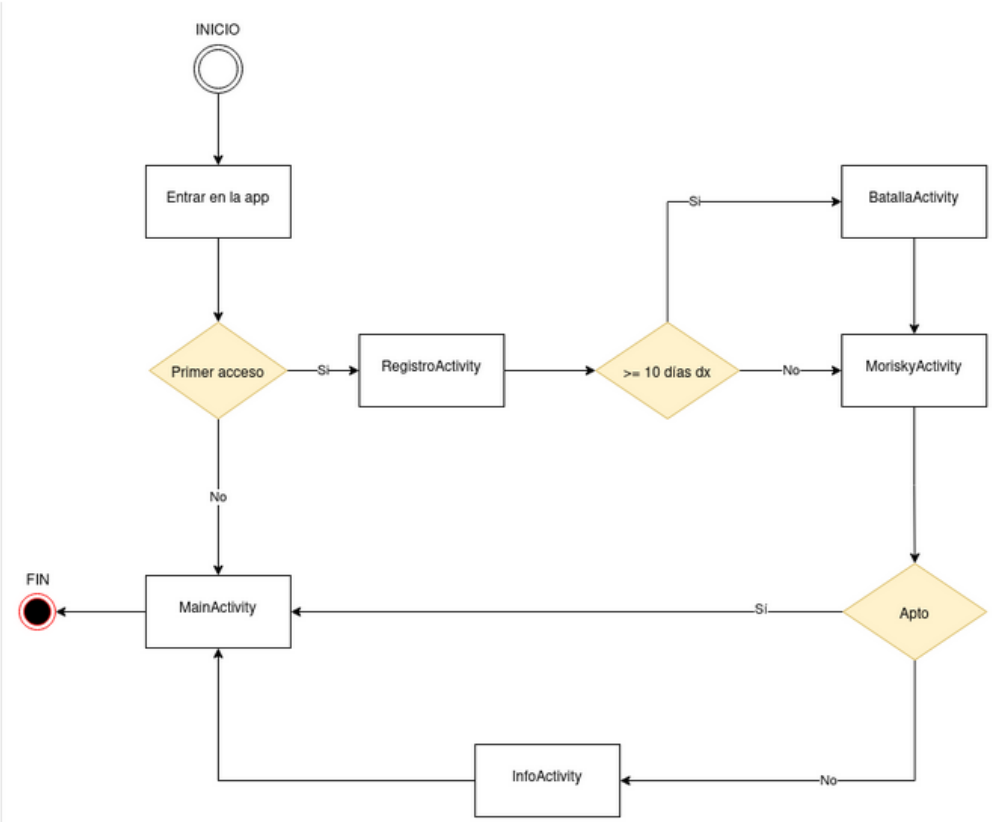


Imagen 101: Detalle de un *issue* en el que se muestra la historia de usuario relativa a la actividad de hacer login de un usuario registrado para la aplicación Android.

III.2 Añadir nuevo medicamento

Añadir nuevo medicamento

Historia de usuario:

Como... usuario con rol "Paciente" de la aplicación

Quiero... poder registrar un nuevo medicamento

Para... poder llevar un seguimiento de toda la medicación que tomo

Los usuarios con rol "Paciente" podrán añadir nuevos medicamentos a su lista de medicación actual, con la posología, frecuencia y duración del tratamiento. De este modo la aplicación se encargará de enviarle recordatorios para cuando tenga que tomar la medicación y además podrá registrar su adherencia al tratamiento pautado por su médico.

- [x] Diseñar la interfaz con un wireframe
- [x] Realizar el diagrama de flujo o algoritmo de este proceso
- [ ] Implementar la interfaz en un layout
- [ ] Crear la clase Java que implementa la lógica de registro (obtiene datos del layout y los guarda en la BBDD)
  - [ ] Introducir nombre de medicamento: Debería tomar el nombre de la API de AEMPS
  - [ ] Introducir dosis y unidades
  - [ ] Introducir frecuencia y esto provoca una toma de decisión, hay que seguir lo del diagrama

Diagrama de flujo:

Imagen 102: Detalle de un *issue* en el que se muestra la historia de usuario correspondiente a añadir un nuevo medicamento en la aplicación Android.

III.3 Eliminar medicamento

Eliminar un medicamento

Historia de usuario:

Como... usuario con rol "Paciente" de la aplicación

Quiero... poder eliminar un medicamento

Para... que la aplicación deje de llevar su seguimiento

Los usuarios con rol "Paciente" podrán eliminar medicamentos de su lista de medicación actual. De este modo la aplicación dejará de enviarle recordatorios para cuando tenga que tomar la medicación y además dejará de registrar su adherencia al tratamiento pautado por su médico.

- ☐ Diseñar la interfaz con un wireframe
- ☐ Realizar el diagrama de flujo o algoritmo de este proceso
- ☐ Implementar la interfaz en un layout
- ☐ Crear funcionalidad en la clase Java correspondiente

Diagrama de flujo:

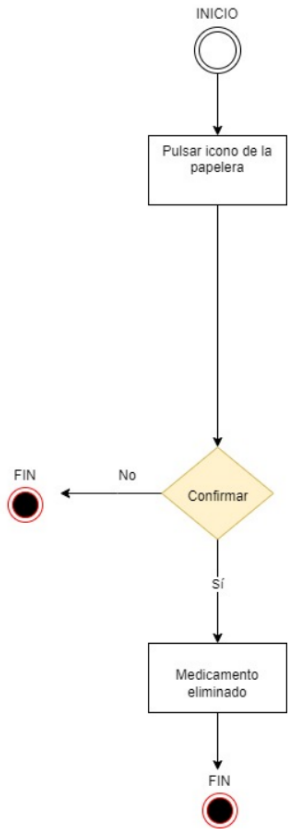


Imagen 103: Detalle de un *issue* en el que se muestra la historia de usuario correspondiente a eliminar un medicamento en la aplicación Android.

III.4 Modificar medicamento

Modificar un medicamento

Historia de usuario:

Como... usuario con rol "Paciente" de la aplicación

Quiero... poder modificar un medicamento previamente guardado

Para... corregir posibles fallos introducidos previamente o porque ha cambiado la forma de toma

Los usuarios con rol "Paciente" podrán modificar los medicamentos de su lista de medicación actual. Estos cambios pueden ser debidos a errores introducidos en el momento de registrar su medicación o por cambios en la pauta indicados por su médico. De este modo la aplicación actualizará los parámetros para enviarle recordatorios para cuando tenga que tomar la medicación y además registrará su adherencia al tratamiento pautado por su médico.

- [ ] Diseñar la interfaz con un wireframe
- [ ] Realizar el diagrama de flujo o algoritmo de este proceso
- [ ] Implementar la interfaz en un layout
- [ ] Crear la clase Java que implementa la lógica de modificar mediacamento

Diagrama de flujo:

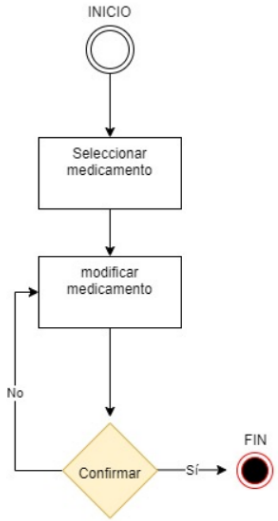


Imagen 104: Detalle de un *issue* en el que se muestra la historia de usuario correspondiente a modificar un medicamento en la aplicación Android.

III.5 Añadir toma de tensión

Formulario toma de tensión

Historia de usuario:

Como... usuario con rol "Paciente" de la aplicación

Quiero... poder añadir una toma de tensión

Para... llevar historial de tomas

Los usuarios con rol "Paciente" podrán añadir una nueva toma de tensión para poder tener un historial y hacer un seguimiento

- ☐ Diseñar la interfaz con un wireframe
- ☐ Realizar el diagrama de flujo o algoritmo de este proceso
- ☐ Implementar la interfaz en un layout
- ☐ Crear la clase Java que implementa la lógica de añadir tensión

Diagrama de flujo:

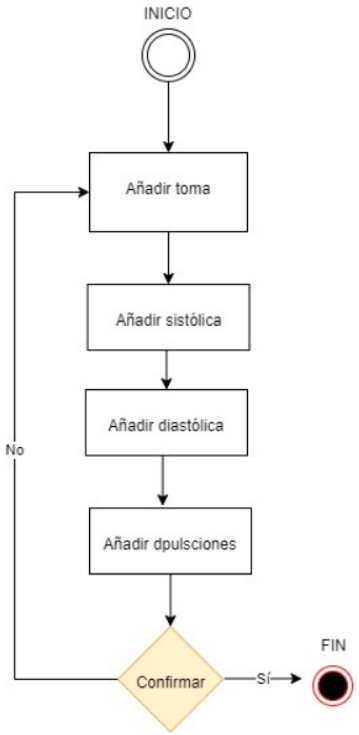


Imagen 105: Detalle de un *issue* en el que se muestra la historia de usuario correspondiente a añadir una nueva toma de tensión en la aplicación Android.

III.6 Test de batalla

Test Batalla

Historia de usuario:

Como... usuario con rol "Paciente" de la aplicación

Quiero... poder hacer el test de batalla

Los usuarios con rol "Paciente" podrán realizar el test de batalla

- ☐ Diseñar la interfaz con un wireframe
- ☐ Realizar el diagrama de flujo o algoritmo de este proceso
- ☐ Implementar la interfaz en un layout
- ☐ Crear funcionalidad en la clase Java correspondiente

Diagrama de flujo:

Imagen 106: Detalle de un *issue* en el que se muestra la historia de usuario correspondiente a rellenar el test de Batalla en la aplicación Android.

III.7 Test de Morisky-Green

Test Morisky-Green

Historia de usuario:

Como... usuario con rol "Paciente" de la aplicación

Quiero... poder hacer el Morisky-Green

Los usuarios con rol "Paciente" podrán realizar el test de Morisky-Green

- ☐ Diseñar la interfaz con un wireframe
- ☐ Realizar el diagrama de flujo o algoritmo de este proceso
- ☐ Implementar la interfaz en un layout
- ☐ Crear funcionalidad en la clase Java correspondiente

Diagrama de flujo:

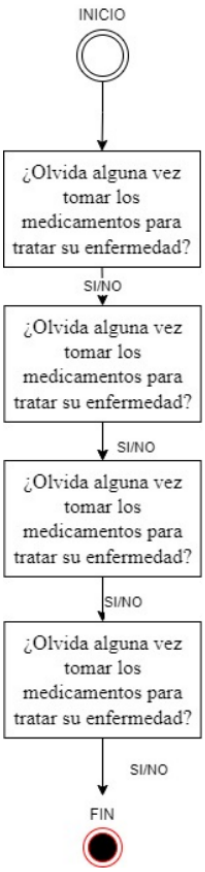


Imagen 107: Detalle de un *issue* en el que se muestra la historia de usuario correspondiente a rellenar el test de Morisky-Green en la aplicación Android.

III.8 Chat

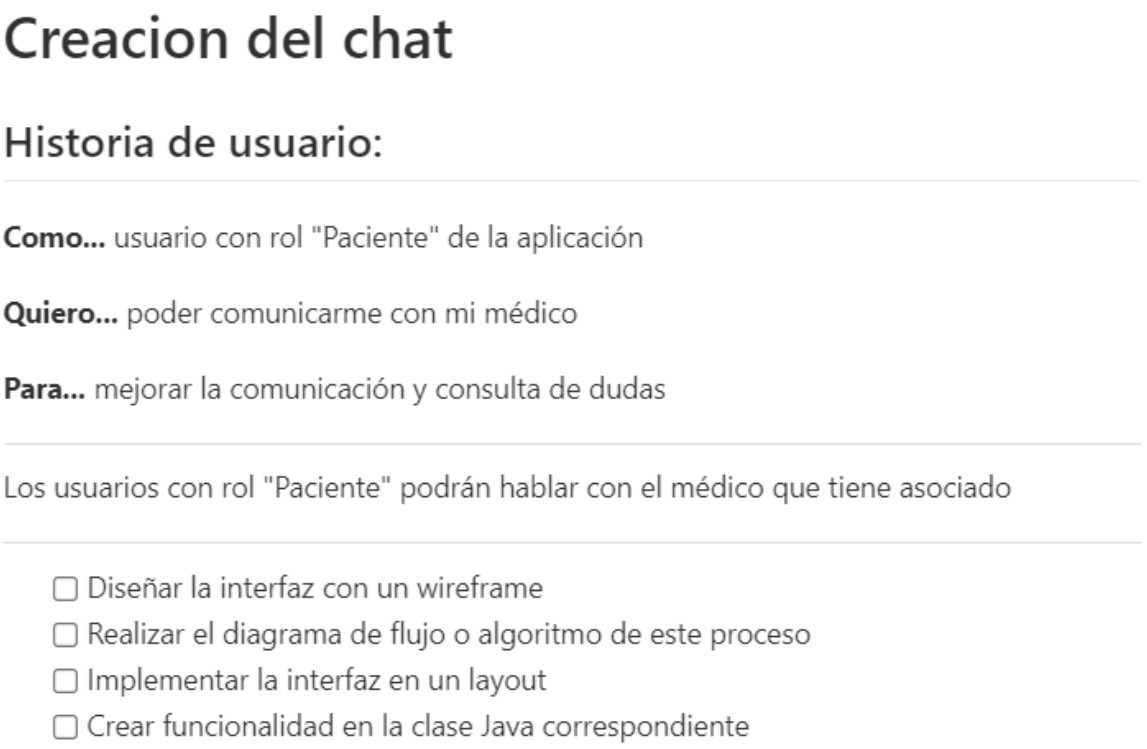


Imagen 108: Detalle de un *issue* en el que se muestra la historia de usuario correspondiente a iniciar el chat en la aplicación Android.

Glosario de Acrónimos y Abreviaturas

Abreviatura/Acrónimo	Significado
AEMPS	Agencia Española de Medicamentos y Productos Sanitarios
API	*Application Programming Interface*
BD	Base de Datos
CRUD	*Create, Read, Update and Delete*
DRF	*Django REST Framework*
FK	*Foreign Key*
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
JSON	*JavaScript Object Notation*
JWT	*JSON Web Token*
ORM	*Object Relational Mapping*
REST	*Representational State Transfer*
SGBD	Sistema Gestor de Bases de Datos
SQL	*Structured Query Language*
URL	*Uniform Resource Locator*
VPS	*Virtual Private Server*

Tabla 8: Abreviaturas y acrónimos empleados.

Índice de figuras y tablas

Figuras

Tablas

Bibliografía

[1] J. R. Banegas y T. Gijón-Conde, «Epidemiología de la hipertensión arterial», *Hipertens. Riesgo Vasc.*, vol. 34, pp. 2-4, ene. 2017, doi: 10.1016/S1889-1837(18)30066-7.
[2] «Google Play». https://play.google.com/store?hl=es&gl=ES (accedido 5 de mayo de 2022).
[3] «Blood Pressure Log - MyDiary - Apps on Google Play». https://play.google.com/store/apps/details?id=com.zlamanit.blood.pressure&hl=en_US&gl=US (accedido 9 de mayo de 2022).
[4] «Qardio Heart Health - Apps on Google Play». https://play.google.com/store/apps/details?id=com.getqardio.android&hl=en_US&gl=US (accedido 9 de mayo de 2022).
[5] «MyFitnessPal: Calorie Counter - Apps on Google Play». https://play.google.com/store/apps/details?id=com.myfitnesspal.android&hl=en_US&gl=US (accedido 9 de mayo de 2022).
[6] «Google Fit: Reg. de actividad - Aplicaciones en Google Play». https://play.google.com/store/apps/details?id=com.google.android.apps.fitness&hl=es&gl=ES (accedido 9 de mayo de 2022).
[7] «Samsung Health - Aplicaciones en Google Play». https://play.google.com/store/apps/details?id=com.sec.android.app.shealth&hl=es&gl=ES (accedido 9 de mayo de 2022).
[8] «BP Journal: Blood Pressure Log - Apps on Google Play». https://play.google.com/store/apps/details?id=com.portalgroove.bpjournal&hl=en_US&gl=US (accedido 9 de mayo de 2022).
[9] «Registro de presión arterial - Aplicaciones en Google Play». https://play.google.com/store/apps/details?id=com.bluefish.bloodpressure&hl=es&gl=ES (accedido 4 de septiembre de 2022).
[10] C. Batalla *et al.*, «Cumplimiento de la prescripción farmacológica en pacientes hipertensos», *Aten Primaria*, vol. 1, n.º 4, pp. 185-91, 1984.
[11] D. E. Morisky, L. W. Green, y D. M. Levine, «Concurrent and predictive validity of a self-reported measure of medication adherence», *Med. Care*, vol. 24, n.º 1, pp. 67-74, ene. 1986, doi: 10.1097/00005650-198601000-00007.
[12] «User Story Mapping – Help your organization focus on successful outcomes». https://www.jpattonassociates.com/story-mapping/ (accedido 7 de julio de 2022).
[13] «What is Scrum?», *Scrum.org*. https://www.scrum.org/resources/what-is-scrum (accedido 12 de abril de 2022).
[14] «Git». https://git-scm.com/ (accedido 12 de abril de 2022).
[15] «Iterate faster, innovate together | GitLab». https://about.gitlab.com/ (accedido 12 de abril de 2022).
[16] «A successful Git branching model», *nvie.com*. http://nvie.com/posts/a-successful-git-branching-model/ (accedido 18 de abril de 2022).
[17] «Servidor virtual privado», *Wikipedia, la enciclopedia libre*. 27 de junio de 2022. Accedido: 13 de agosto de 2022. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Servidor_virtual_privado&oldid=144436395
[18] «Contabo 🥇 Quality VPS & Dedicated Servers At Incredible Prices 🥇 ». https://contabo.com/en/contabo.com/en/ (accedido 10 de julio de 2022).
[19] «Password management in Django | Django documentation | Django». https://docs.djangoproject.com/en/4.0/topics/auth/passwords/#module-django.contrib.auth.password_validation (accedido 4 de septiembre de 2022).
[20] «draw.io – Diagrams for Confluence and Jira», *draw.io*. https://drawio-app.com/ (accedido 9 de mayo de 2022).

[21] :: «CIMA ::. Nomenclátor de prescipción». https://cima.aemps.es/cima/publico/nomenclator.html (accedido 18 de julio de 2022).
[22] «Operating System Market Share Spain», *StatCounter Global Stats*. https://gs.statcounter.com/os-market-share/all/spain (accedido 5 de mayo de 2022).
[23] «Sistemas operativos para smartphone: penetración en España», *Statista*. https://es.statista.com/estadisticas/473759/tasa-penetracion-sistema-operativo-smartphone-espana/ (accedido 5 de mayo de 2022).
[24] «Android Open Source Project», *Android Open Source Project*. https://source.android.com/ (accedido 5 de mayo de 2022).
[25] «OpenJDK». https://openjdk.java.net/ (accedido 5 de mayo de 2022).
[26] «Meet Android Studio», *Android Developers*. https://developer.android.com/studio/intro (accedido 15 de agosto de 2022).
[27] «The web framework for perfectionists with deadlines | Django». https://www.djangoproject.com/ (accedido 20 de abril de 2022).
[28] «index | TIOBE - The Software Quality Company». https://www.tiobe.com/tiobe-index/ (accedido 5 de mayo de 2022).
[29] «Django overview | Django». https://www.djangoproject.com/start/overview/ (accedido 5 de mayo de 2022).
[30] «Long-term support», *Wikipedia*. 24 de febrero de 2022. Accedido: 20 de abril de 2022. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=Long-term_support&oldid=1073817233
[31] «Asignación objeto-relacional», *Wikipedia, la enciclopedia libre*. 13 de marzo de 2022. Accedido: 18 de abril de 2022. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Asignaci%C3%B3n_objeto-relacional&oldid=142248581
[32] «SQL», *Wikipedia*. 11 de abril de 2022. Accedido: 19 de abril de 2022. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=SQL&oldid=1082173977
[33] P. G. D. Group, «PostgreSQL», *PostgreSQL*, 18 de abril de 2022. https://www.postgresql.org/ (accedido 18 de abril de 2022).
[34] «Prueba unitaria», *Wikipedia, la enciclopedia libre*. 22 de marzo de 2022. Accedido: 18 de abril de 2022. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Prueba_unitaria&oldid=142444955
[35] «API», *Wikipedia*. 19 de abril de 2022. Accedido: 19 de abril de 2022. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=API&oldid=1083514505
[36] «Representational state transfer», *Wikipedia*. 19 de abril de 2022. Accedido: 19 de abril de 2022. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=Representational_state_transfer&oldid=1083480269
[37] R. T. Fielding, «Architectural Styles and the Design of Network-based Software Architectures», University of California, Irvine, California, 2000. [En línea]. Disponible en: https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf
[38] «Home - Django REST framework». https://www.django-rest-framework.org/ (accedido 5 de mayo de 2022).
[39] «¿Qué es Docker?», 14 de septiembre de 2021. https://www.ibm.com/mx-es/cloud/learn/docker (accedido 10 de julio de 2022).
[40] «Cloud Application Platform | Heroku». https://www.heroku.com/ (accedido 10 de julio de 2022).
[41] «Home - Docker», 10 de mayo de 2022. https://www.docker.com/ (accedido 10 de julio de 2022).
[42] «The C4 model for visualising software architecture». https://c4model.com/ (accedido 30 de marzo de 2022).
[43] «FAQ: General | Django documentation | Django». https://docs.djangoproject.com/en/3.2/faq/general/#django-appears-to-be-a-mvc-framew

ork-but-you-call-the-controller-the-view-and-the-view-the-template-how-come-you-don-t-use-the-standard-names (accedido 13 de julio de 2022).
[44] «Object–relational mapping», *Wikipedia*. 7 de julio de 2022. Accedido: 13 de julio de 2022. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=Object%E2%80%93relational_mapping&oldid=1096856011
[45] «Create, read, update and delete», *Wikipedia*. 26 de junio de 2022. Accedido: 13 de julio de 2022. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=Create,_read,_update_and_delete&oldid=1095124142
[46] «Model–view–controller», *Wikipedia*. 1 de junio de 2022. Accedido: 13 de julio de 2022. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=Model%E2%80%93view%E2%80%93controller&oldid=1090913287
[47] «Guide to app architecture», *Android Developers*. https://developer.android.com/topic/architecture (accedido 12 de julio de 2022).
[48] «JSON», *Wikipedia*. 10 de julio de 2022. Accedido: 17 de julio de 2022. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=JSON&oldid=1097421495
[49] «Seguridad de la capa de transporte», *Wikipedia, la enciclopedia libre*. 2 de agosto de 2022. Accedido: 15 de agosto de 2022. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Seguridad_de_la_capa_de_transporte&oldid=145137720
[50] «Let's Encrypt». https://letsencrypt.org/ (accedido 15 de agosto de 2022).
[51] «Protocolo seguro de transferencia de hipertexto», *Wikipedia, la enciclopedia libre*. 11 de agosto de 2022. Accedido: 15 de agosto de 2022. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Protocolo_seguro_de_transferencia_de_hipertexto&oldid=145303531
[52] «JSON Web Token», *Wikipedia, la enciclopedia libre*. 16 de junio de 2022. Accedido: 15 de agosto de 2022. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=JSON_Web_Token&oldid=144234501
[53] «Simple JWT — Simple JWT 5.2.0.post5+gcd4ea99 documentation». https://django-rest-framework-simplejwt.readthedocs.io/en/latest/index.html (accedido 19 de julio de 2022).
[54] «Traefik, The Cloud Native Application Proxy | Traefik Labs», *Traefik Labs: Makes Networking Boring*. https://traefik.io/traefik/ (accedido 20 de julio de 2022).
[55] «apk (file format)», *Wikipedia*. 16 de agosto de 2022. Accedido: 17 de agosto de 2022. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=Apk_(file_format)&oldid=1104797068
[56] «React – A JavaScript library for building user interfaces». https://reactjs.org/ (accedido 16 de agosto de 2022).
[57] «pytest: helps you write better programs — pytest documentation». https://docs.pytest.org/en/7.1.x/ (accedido 16 de agosto de 2022).
[58] «Unified Modeling Language», *Wikipedia*. 15 de marzo de 2022. Accedido: 19 de abril de 2022. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=Unified_Modeling_Language&oldid=1077269626
[59] «Postman API Platform | Sign Up for Free», *Postman*. https://www.postman.com/ (accedido 20 de julio de 2022).