



UNIVERSIDAD COMPLUTENSE MADRID

TRABAJO DE FIN DE GRADO Esd2: Cuaderno de recogida de datos para un estudio médico

Eduardo Gonzalo Montero

&

Sergio Pacheco Fernández

Profesor director: Pablo Manuel Rabanal Basalo

Codirector: Alejandro Rabanal Basalo

Curso académico: 2019-2020

Identificación asignatura: Trabajo de fin de Grado, Grado en Ingeniería del Software, Facultad de Informática



UNIVERSIDAD COMPLUTENSE MADRID

FINAL DEGREE PROJECT **Esd2: Data collection notebook for a medical research**

Eduardo Gonzalo Montero

&

Sergio Pacheco Fernández

Director professor: Pablo Manuel Rabanal Basalo

Co-director: Alejandro Rabanal Basalo

Academic year: 2019-2020

Subject identification: Final Degree Project, Degree in Software Engineering, Computer Science Faculty

Índice

Resumen	4
Palabras clave	5
Abstract	6
Keywords	7
1. Introducción	8
1.1. Objetivo	8
1.2. Motivación	9
1.3. Metodología de trabajo	10
1.3.1. Transparencia	10
1.3.2. Inspección	11
1.3.3. Adaptación	12
1. Introduction	13
1.1. Objective	13
1.2. Motivation	14
1.3. Work methodology	15
1.3.1. Transparency	15
1.3.2. Inspection	16
1.3.3. Adaptation	17
2. Tecnologías	18
2.1. Lenguajes de programación	18
2.1.1. TypeScript	18
2.1.2. HTML-5	18
2.1.3. CSS-3	19
2.1.4. Java 8	19
2.1.5. SQL	19
2.2. Entornos de desarrollo	19
2.2.1. Visual Studio Code	19
2.2.2. MySQL Workbench	19
2.2.3. PhpMyAdmin	19
2.2.4. GitHub	20
2.2.5. Bitbucket	20
2.2.6. Eclipse	20
2.2.7. Overleaf	20

2.2.8. MobaXterm	20
2.3. Frameworks	21
2.3.1. Java Spring	21
2.3.2. Angular	22
3. Desarrollo	23
3.1. Arranque del desarrollo	23
3.2. Desarrollo durante los sprints	30
4. Implementación	40
4.1. Frontend	40
4.1.1. Generación Proyecto Angular	40
4.1.2. Arquitectura de la aplicación Angular	41
4.1.3. Inicialización del proyecto	47
4.2. Backend	47
4.2.1. Generación Proyecto Java Spring Boot	47
4.2.2. Arquitectura de la aplicación Java Spring Boot	48
4.2.3. Inicialización del proyecto	57
5. Despliegue	58
5.1. Frontend	58
5.2. Backend	59
5.3. Base de Datos	61
6. Mantenimiento	63
7. Conclusiones y Trabajo Futuro	66
7.1. Conclusiones	66
7.2. Trabajo Futuro	67
7. Conclusions and Future Work	68
7.1. Conclusions	68
7.2. Future Work	69
8. Contribuciones individuales	70
8.1. Eduardo Gonzalo Montero	70
8.2. Sergio Pacheco Fernández	72
Bibliografía	74

Agradecimientos

A todos los familiares, amigos y profesores que nos han acompañado en nuestro camino y nos han apoyado en momentos oscuros, que han compartido nuestra carga y han festejado nuestros éxitos, sin ellos no estaríamos aquí. A título personal, mención especial al profesor Antonio Navarro, sin el cual no habríamos llegado tan lejos y al que agradecemos todos los conocimientos adquiridos.

“Si quieres ir rápido, ve solo. Si quieres llegar lejos, ve acompañado”.
-Proverbio africano.

Resumen

La diabetes mellitus se encuentra entre las 10 principales causas de muerte a nivel mundial (en 2017, 4 millones de personas entre 20 y 79 años fallecieron debido a ella). Se calcula que alrededor de 425 millones de personas tienen diabetes actualmente en el mundo a pesar de que muchos de los casos permanecen sin registrar. Solo en Europa alrededor del 38 % de los casos de diabetes aún están sin diagnosticar, lo que supone unos 22 millones más de afectados. En España se estima que más de 5 millones de personas padecen esta enfermedad, dándose más de 380.000 nuevos pacientes cada año.

En el caso de la diabetes mellitus tipo 2 se estima que 9 de cada 10 casos pueden atribuirse a hábitos de vida que podrían modificarse promoviendo estilos de vida saludables como el deporte o seguir una dieta equilibrada, ya que la obesidad es uno de los mayores factores de riesgo. Sin embargo, en los últimos años se han planteado también otros importantes factores de riesgo, entre ellos el déficit de Vitamina D. No obstante, los umbrales de niveles correctos de esta vitamina son muy controvertidos y el estudio de su impacto, por tanto, es complejo.

Este proyecto consiste en desarrollar un portal web que dará soporte a un equipo médico recopilando datos de contraste en pacientes con diabetes mellitus tipo 2. La aplicación les permitirá recopilar datos sobre los niveles de Vitamina D entre otros factores como la exposición diaria a la luz solar, el ejercicio físico diario o hábitos como el tabaquismo, todos ellos vía formularios para posteriormente, compararlos y extraer conclusiones que esperan ayuden a delimitar mejor la enfermedad, promover su prevención y, en general conseguir una mejor comprensión de la misma.

Se busca tras la finalización del proyecto, proporcionar a los médicos una herramienta útil y ajustada que les permita recopilar datos para avanzar en su investigación.

Palabras clave

- Servicio web
- Estudio médico
- Diabetes
- Vitamina D
- API-REST
- Investigación
- Sanidad

Abstract

Diabetes mellitus is one of the 10 main causes of death worldwide (in 2017, 4 million people between 20 and 79 years passed away due to it). It is estimated that around 425 million people currently suffer diabetes in the world, despite there are a lot of unregistered cases. Only in Europe, around 38 % of diabetes cases even remain undiagnosed, which means 22 million people affected. In Spain, it is estimated more than 5 million people suffer this disease, with more than 380.000 new patients every year.

In the case of diabetes mellitus type 2, it is estimated 9 in 10 cases can be attributed to lifestyle that could be changed by promoting healthy lifestyles such as sport or having a balanced diet, since obesity is one the main risk factors. Nevertheless, in recent years other important risk factors have been raised, among them the vitamin D deficit. However, the correct levels threshold of this vitamin are controversial, and the research impact therefore is complex.

This project consists of developing a web portal which will provide support to a medical team collecting contrast data in patients with type 2 diabetes mellitus. The application will allow them to gather information on vitamin D levels, among other factors such as daily exposure to sunlight, daily physical exercise, or habits such as smoking, all of them through forms to later compare them, and draw conclusions which are expected to define the disease in a better way, to promote its prevention and, generally, to get a better understanding.

After project completion, is sought to provide a useful and calibrated tool to the doctors to let them collect data to make progress in their research.

Keywords

- Web service
- Medical research
- Diabetes
- Vitamin D
- API-REST
- Study
- Health

CAPÍTULO 1

Introducción

En este capítulo se detalla la motivación que llevó a la realización del proyecto, los objetivos que este comprende y las pautas que se tomaron para lograr llevarlo a cabo

1.1. Objetivo

Este Trabajo de Final de Grado busca la implementación de un portal web que permita la recopilación de datos de interés para el estudio por parte del equipo médico. Para ello contará con formularios personalizados con los campos de interés solicitados por el mismo que serán rellenados por los médicos en consultas rutinarias con los pacientes que hayan consentido participar en el proyecto. Estos datos serán luego accesibles tanto vía web como en archivos de Excel descargables desde el propio portal.

Con el fin de ajustar el producto final a las necesidades del equipo la aplicación se entregará con tiempo suficiente (aproximadamente en febrero de 2020) para que pueda ser probada, revisada y modificada a medida. Esto es especialmente importante ya que el estudio se extenderá muchos meses más allá de la finalización de este TFG.

El objetivo final, y por supuesto el más importante, es que todo este desarrollo sirva de herramienta para ajustar los criterios de evaluación en los pacientes con diabetes mellitus tipo 2, permitiendo encontrar pautas en sus estados de salud subsanables que puedan en un futuro prevenir más casos de esta enfermedad.

1.2. Motivación

Nuestra primera motivación para adentrarnos en este proyecto son las tecnologías empleadas en el mismo. Ambos participantes estamos interesados en centrar nuestras carreras en tecnologías web, uno de ellos trabajando ya de hecho como desarrollador *full-stack*. El proyecto da además libertad para ser implementado sin ningún tipo de restricciones de diseño o rendimiento, lo que plantea un escenario ideal para experimentar durante su desarrollo.

El otro motivo principal para decantarnos por este proyecto es su cercanía a un proyecto real. El cliente, la aplicación, los plazos y las necesidades del mismo no son algo creado para un ambiente académico, como otros desarrollos ya efectuados durante la carrera, sino un problema real que precisa una solución efectiva contando con todas las personas que lo acabarán utilizando. Además el proyecto requerirá de un mantenimiento post entrega, otro ámbito nunca explorado en la carrera y que será una valiosa experiencia de cara a nuestro futuro.

Por último remarcar que uno de los miembros, Sergio, ya tiene buenas experiencias con un proyecto anterior para el ámbito médico desarrollado en solitario para una empresa en Suiza y Eduardo siempre ha tenido interés en los hábitos de vida saludables y la nutrición, posible principal remedio para la diabetes extraído de los resultados de este estudio.

1.3. Metodología de trabajo

El proyecto será planificado bajo los estándares de la metodología Scrum[1] aunque distendiendo un poco sus cotas temporales, pues al ser solo dos miembros no es necesario hacer un hincapié tan diario en la organización para mantener el orden. Lo primero será definir cómo mantendremos los tres pilares básicos de la metodología: **transparencia, inspección y adaptación**.

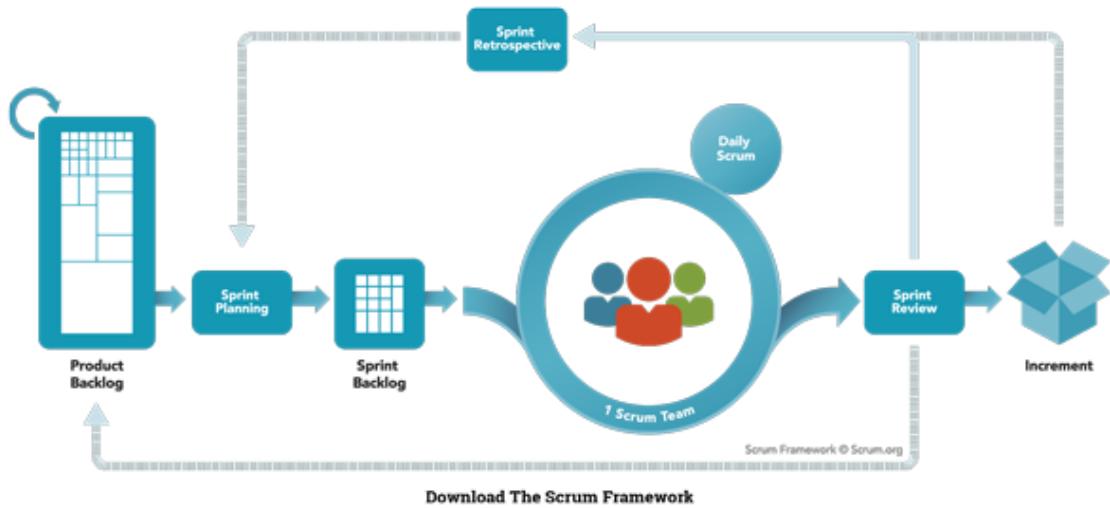


Figura 1.1: Flujo de trabajo en Scrum. Imagen obtenida de Scrum.org [2]

1.3.1. Transparencia

Para mantener a ambos miembros al día de cualquier avance o variación en el proceso se empleará la herramienta online de Trello[3], que nos permitirá ir viendo en tiempo real los mismos. Este será estructurado de la siguiente forma:

- Se crearán dos columnas, una con la lista de tareas pendientes para el sprint actual y otra con las tareas actualmente en desarrollo. Además se creará una columna por cada sprint en la que se irán almacenando todas las tareas finalizadas. Cada tarea podrá encapsularse en una o más de estas categorías: despliegue, documentación, front, bug, investigación, modelo de datos, pendiente de resolver (cuestión a esperar de la próxima reunión con el tutor), back, varios, resuelto (cuestiones ya solucionadas en anteriores reuniones que quedan como recordatorio). Además cada tarea llevará una o más etiquetas indicando los desarrolladores que han trabajado en ella.

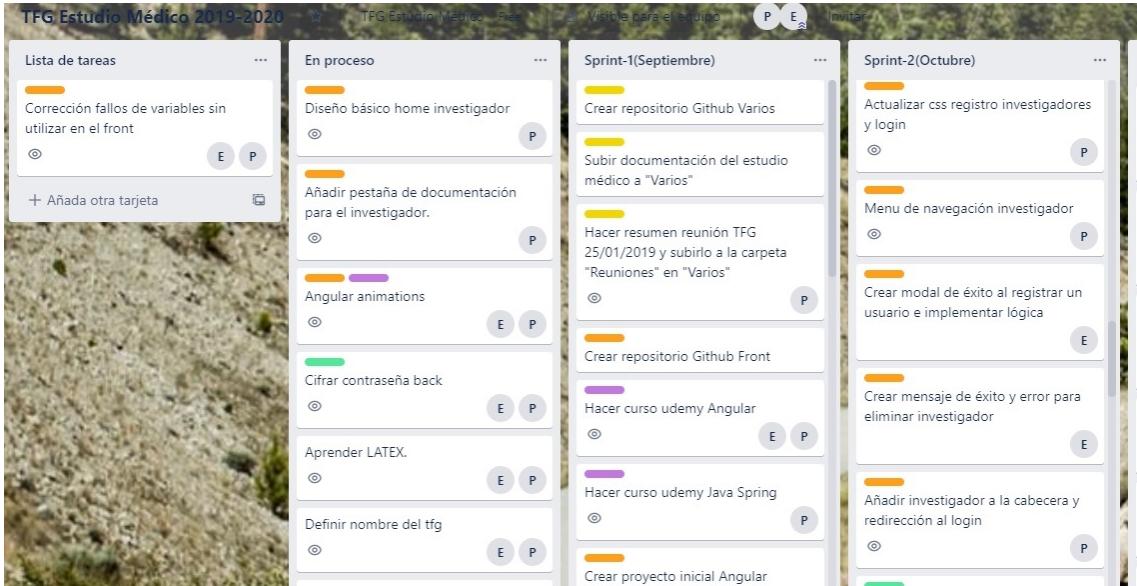


Figura 1.2: Captura de ejemplo de la herramienta Trello.

- Asimismo, todo el código del proyecto estará subido y actualizado en un repositorio, en este caso GitHub[4], a través del cual se podrá ver un histórico preciso de los cambios realizados en cada *commit* junto a comentarios explicativos del desarrollador que los haya realizado.

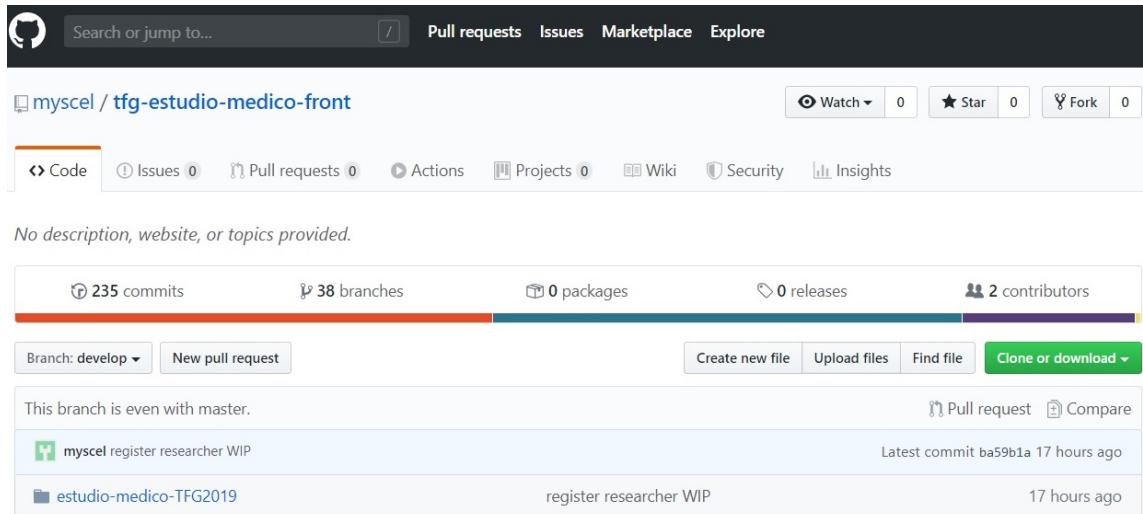


Figura 1.3: Captura de ejemplo del repositorio front en GitHub.

1.3.2. Inspección

Con el fin de mantener este principio, nuestro repositorio no solo albergará código, sino también un espacio separado para todos los artefactos derivados de Scrum, así como la documentación que se vaya creando durante el proceso que sea relevante a efectos de esta memoria. Estos artefactos serán por tanto accesibles constantemente por ambos miembros, los cuales avisarán en caso de cualquier añadido o modificación para que su compañero pueda revisarlo.

No description, website, or topics provided.

19 commits	1 branch	0 packages	0 releases	1 contributor	
Branch: master	New pull request	Create new file	Upload files	Find file	Clone or download
myscel capítulo despliegue terminado					Latest commit 589934f yesterday
Aprendizaje LATEX	Realización portadas				15 days ago
Documentos memoria	URI utilidad creado				7 days ago
memoria	capítulo despliegue terminado				yesterday
README.md	Initial commit				16 days ago

Figura 1.4: Captura de ejemplo con todos los documentos referentes a la memoria en GitHub tomada a posteriori.

1.3.3. Adaptación

Al final de cada sprint se realizará una reunión tanto con el tutor del proyecto como con un representante del equipo médico para revisar el proceso y obtener *feedback* del mismo. Los errores o cambios urgentes extraídos de estas reuniones pasarán a ser las tareas más prioritarias del proyecto y su resolución será notificada de inmediato a ambos asistentes de la reunión. Además, para mantener el proyecto lo más afín a las necesidades del cliente cualquier cambio significativo en mitad de un *sprint* será notificado por correo previamente.

Además, de toda la documentación y artefactos mencionados disponibles durante el desarrollo los miembros mantendrán un contacto diario hablando de qué se ha hecho, se va a hacer y los problemas que se van encontrando, el cual hará las veces de *Daily Scrums*.

En cuanto a los roles típicos de Scrum, en nuestro caso, no serán utilizados. El *Product Owner* será suplido con la comunicación directa y constante de los miembros con el cliente que asegurará la priorización del valor para este durante el desarrollo. El *Scrum Master* no será preciso ya que ambos miembros del equipo tienen experiencia utilizando metodologías ágiles, tanto en la carrera como fuera en trabajos o proyectos propios. Por tanto solo existirá el equipo de desarrollo y este suplirá las funciones necesarias del resto de roles.

Los *sprints* serán de una duración mensual. Se opta por esta cuantía porque al estar uno de los miembros aún con clases y otro trabajando a jornada completa no se estima que en una semana el desarrollo pueda tener un avance suficientemente significativo. Estos sprints se iniciarán con una reunión de planificación con el tutor del proyecto y el representante del equipo médico, estipulando qué se hará y cómo. Asimismo, concluirán con otra reunión de misma índole que mostrará una demo funcional al representante y recogerá su *feedback*, para acto seguido comenzar con la reunión de planificación del siguiente *sprint*. Se decide aunar así el inicio y final del sprint en una sola reunión para evitar problemas de horarios difíciles de encajar por parte de los cuatro. Tras esta reunión con todos los miembros se realizará una pequeña reunión de retrospectiva solo del equipo de desarrollo para determinar si la metodología de trabajo funciona y qué se podría modificar o añadir.

Introduction

This chapter details the motivation that led to the realization of this project, the objectives that it comprises and the guidelines that were taken to achieve it.

1.1. Objective

This Final Degree Project seeks the implementation of a web portal that allows the collection of data of interest for the study of the medical team. To do this, it will have personalized forms with the fields of interest requested by them, which will be filled in by doctors in routine appointments with patients who have consented to participate in the project. These data will then be accessible both via the web and in a downloadable Excel file from the portal itself.

In order to adjust the final product to the needs of the team, the application will be launched with enough time (approximately in February 2020) so that it can be tested and modified to measure. This is especially important since the study will extend many months beyond the end of this FDP.

The final objective, and of course the most important one, is that all this development serves as a tool to adjust the evaluation criteria in patients with type 2 diabetes mellitus, allowing to find guidelines to improve in their life habits that may prevent more cases of this disease in the future.

1.2. Motivation

Our first motivation to get into this project is the technologies used in it. Both participants are interested in focusing their career on web technologies, in fact one of them is already working as a full-stack developer. The project also gives freedom to be implemented without any design or performance restrictions, which presents an ideal scenario to experiment during its development.

The other main reason for opting for this project is its proximity to a real project. The client, the application, the deadlines, and the needs of this project are not something created for an academic environment, like other developments already carried out during the degree, but a real problem that requires an effective solution with all the people who will end up using it. In addition, the project will require post-launch maintenance, another area never explored in the degree and which will be a valuable experience for our future.

Note that one of the members, Sergio, already has good experiences with a previous project in the medical field developed alone for a Switzerland company, and Eduardo has always had an interest in healthy lifestyle habits and nutrition, possible main remedy for diabetes extracted from the results of this study.

1.3. Work methodology

The project will be planned under the standards of the Scrum[1] methodology, although it will slightly loosen its temporary limits. Since there are only two members it is not necessary to place such daily emphasis on the organization to maintain order. The first thing will be to define how we will maintain the three basic pillars of the methodology: **transparency, inspection and adaptation**.

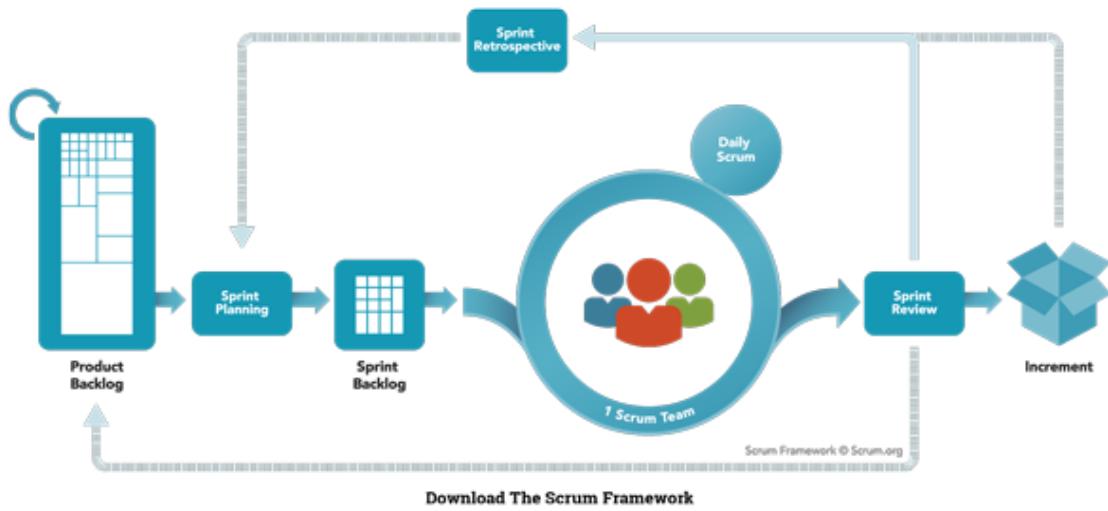


Figura 1.5: Workflow on Scrum. Picture obtained from Scrum.org [2]

1.3.1. Transparency

To keep both members up to date with any progress or variation in the process we will be using Trello's[3] online tool, which will allow us to see them in real time. This will be structured in the following way:

- Two columns will be created, one with the list of pending tasks for the current sprint, and another with the tasks currently in progress. In addition, a column will be created for each sprint in which all the completed tasks will be stored. Each task can be encapsulated in one or more of these categories: deployment, documentation, FrontEnd, bug, research, data model, pending resolution (questions for the next meeting with the tutor), BackEnd, variety, resolved (questions already solved in previous meetings that stay as a reminder). Also, each task will have one or more labels indicating the developers who have worked on it.

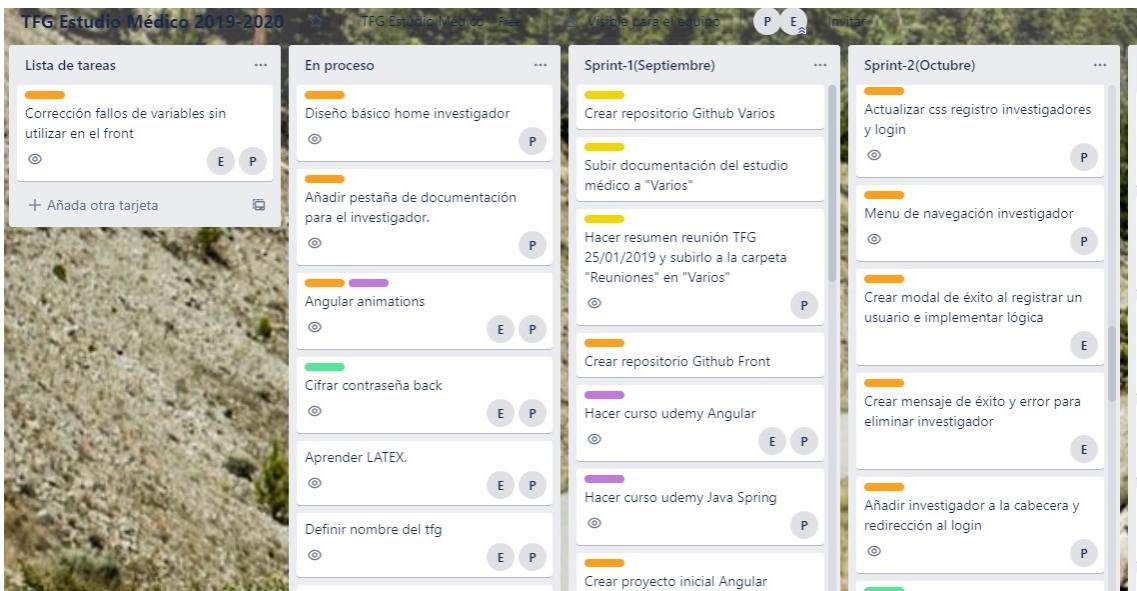


Figura 1.6: Example image of Trello's online tool.

- Also, all the project code will be uploaded and updated in a repository, in this case GitHub[4], through which you can see an accurate history of the changes made in each commit along with explanatory comments from the developer that realised them.

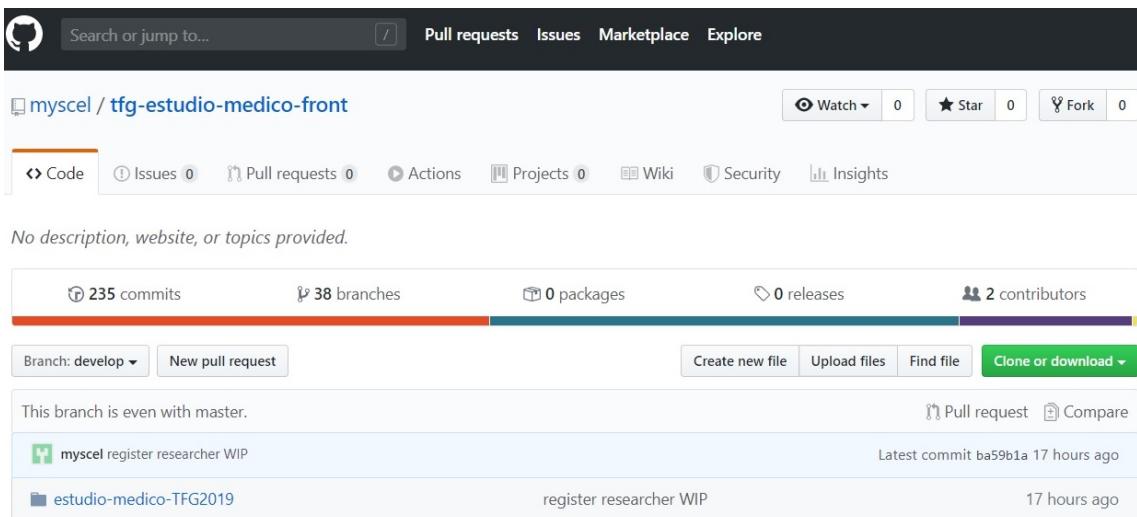


Figura 1.7: Example image of FrontEnd repository on GitHub.

1.3.2. Inspection

In order to uphold this principle, our repository will not only store code, but also a separate space for all the artifacts derived from Scrum, as well as the documentation that is created during the process that is relevant for the purposes of this report. These artifacts will therefore be constantly accessible by both members, who will notify in case of any addition or modification so that their partner can review them.

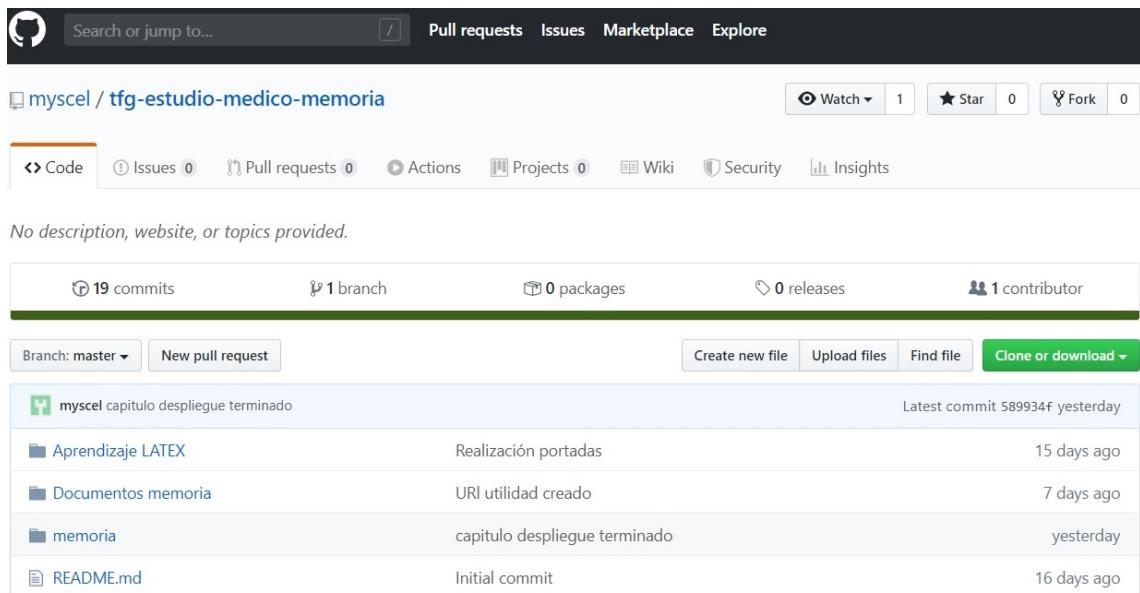


Figura 1.8: Example image of all the documents about this report on GitHub.

1.3.3. Adaptation

At the end of each sprint there will be a meeting with both the project tutor and a representative of the medical team to review the process and obtain feedback on it. Errors or urgent changes extracted from these meetings will become the highest priority of the project and their resolution will be immediately notified to both meeting attendees. In addition, to keep the project as close as possible to the client's needs, any significant change in the middle of a sprint will be notified by mail in advance to them.

In addition of all the documentation and artifacts mentioned available during the development, the members will keep a daily contact talking about what has been done, what is going to be done and the problems they are encountering, which will act as Daily Scrums.

As for the typical Scrum roles, in our case, they will not be used. The Product Owner will be replaced with the direct and constant communication of the members with the client that will ensure the prioritization of the most valuable features for him during the development. The Scrum Master will not be precise since both team members have experience using agile methodologies, both in the career and outside in their own work or projects. Therefore, only the development team will exist and this will supply the functions for the rest of the roles.

Sprints will be monthly. This amount is chosen because one of the members is still with classes and the other is working full time. It is not estimated that in one week the development can have a significant advance. These sprints will begin with a planning meeting with the project tutor and the medical team representative, stipulating what will be done. Likewise, they will conclude with another meeting of the same nature that will show a functional demo to the representative and collect their feedback, to begin with the planning meeting for the next sprint. It was decided to combine the start and end of the sprint in a single meeting in order to avoid problems with schedules that were difficult for all four to fit into. After this meeting with all the members there will be a small retrospective meeting only for the development team to determine if the work methodology works and what could be modified or added.

CAPÍTULO 2

Tecnologías

En este capítulo se detalla todo lo relativo a los lenguajes de programación, los entornos de desarrollo y los *frameworks* elegidos para llevar a cabo este proyecto, así como sus características principales y la razón de su uso.

2.1. Lenguajes de programación

2.1.1. TypeScript

TypeScript[5] es un lenguaje de programación orientado a objetos (OO) el cual es un superconjunto de JavaScript. Decimos que una tecnología es un superconjunto de un lenguaje de programación, cuando puede ejecutar programas de la tecnología . En resumen, ejecutará el código como si fuese JavaScript.

TypeScript se diferencia de JavaScript principalmente en que posee inferencia de tipos, es decir, está fuertemente tipado, además de algunas funcionalidades extra.

Este lenguaje se utiliza en el *frontend* del proyecto, dado que el *framework* elegido para realizar esta parte es Angular, el cual se explicará en detalle más adelante.

2.1.2. HTML-5

HTML-5[6] (HyperText Markup Language) es la quinta versión del lenguaje básico de la página web. Se trata de un lenguaje de marcación para la elaboración del contenido de las páginas web. Hoy en día es el lenguaje estándar que aceptan la gran mayoría de los navegadores a la hora de la construcción de las páginas web.

HTML-5 se diferencia de sus versiones anteriores en que incorpora nuevas etiquetas (*section*, *article*, *header*, *footer* etc...) con las cuales se busca mejorar y estandarizar la estructura de las páginas web además de otras actualizaciones como la mejora de los formularios o la inclusión de elementos de audio y vídeo.

Se ha optado por utilizar este lenguaje de marcación debido a su popularidad y a la inclusión de nuevas etiquetas que favorecen la lectura de la página web por parte de los navegadores.

2.1.3. CSS-3

CSS[7] (Cascading Style Sheets) es un lenguaje de diseño gráfico para gestionar el aspecto visual de un documento estructurado escrito en un lenguaje de marcación. Se utiliza en la mayoría de sitios web junto con HTML para generación de páginas web. De esta manera es mucho más sencillo generar páginas web, ya que, el diseño (CSS) se encuentra separado del contenido (HTML).

2.1.4. Java 8

Java[8] es un lenguaje de programación y una plataforma informática comercializada por primera vez en 1995 por Sun Microsystems[9]. Proviene de los lenguajes C y C++, y sus aplicaciones tienen la capacidad de ejecutarse en cualquier JVM (Java Virtual Machine).

Para este proyecto se ha utilizado la versión 8 porque esta versión es la que menos *bugs* tiene y la que mejora más la eficacia en el desarrollo y la ejecución de programas Java[10]. Además de estas razones, escogimos Java por ser un lenguaje orientado a objetos ideal para desarrollar proyectos *API-REST*.

2.1.5. SQL

SQL[11] es un lenguaje declarativo estándar de comunicación dentro de las bases de datos que nos permite el acceso, gestión y manipulación de datos en una base de datos.

Se decidió utilizar este lenguaje debido a su uso en el SGDB (Sistema de Gestión de Base de Datos) que utilizamos en el proyecto, MySQL. Además de esto, SQL es ideal para trabajar con JPA (Java Persistence API) en el *backend* del proyecto.

2.2. Entornos de desarrollo

2.2.1. Visual Studio Code

Visual Studio Code[12] es un editor de código fuente desarrollado por Microsoft para Windows, Linux y macOS. Incluye, entre otras características, soporte para el proceso de depurar, finalización inteligente de código o refactorización de código.

En este proyecto se ha utilizado este entorno de desarrollo para gestionar un proyecto Angular, ya que este editor posee gran versatilidad a la hora de instalar plugins y gestionar diferentes lenguajes de programación de manera simultánea.

2.2.2. MySQL Workbench

MySQL Workbench[13] se trata de un programa para gestionar, diseñar y administrar bases de datos relacionales, utilizado en nuestro proyecto a la hora de manejar datos de manera local.

Se decidió utilizar debido a que se ha usado previamente en nuestros estudios de grado en diversas asignaturas de manera productiva, además de que posee una versión gratuita.

2.2.3. PhpMyAdmin

Al igual que MySQL Worbench, se trata de una herramienta de gestión de bases de datos MySQL, pero con la diferencia que el acceso a esta herramienta es vía web, alojándose en un servidor.

Esta herramienta la utiliza Hostinger[14], proveedor de alojamiento web donde se ha decidido alojar el proyecto para desplegarlo en la web.

2.2.4. GitHub

GitHub[4] es un sistema de gestión de proyectos y control de versiones de código que funciona con git, el cual hemos utilizado para trabajar en equipo.

Se decidió utilizar este sistema debido a la familiaridad que poseemos con el mismo, lo que se traduce en efectividad y productividad en el flujo de trabajo del equipo.

2.2.5. Bitbucket

Bitbucket[15] es, al igual que GitHub, un sistema de gestión de proyectos y control de versiones. No se tenía planeado utilizar este sistema, pero un cambio en la política de GitHub por el cual se nos impedía gestionar nuestros repositorios de manera privada entre otros cambios, nos obligó a buscar otras opciones con versiones gratuitas en las cuales guardar una copia auxiliar del proyecto en caso de que algo fallase con GitHub.

Tras una búsqueda de sistemas similares, nos decantamos por utilizar Bitbucket debido a la posibilidad de crear repositorios privados gratuitos e ilimitados para equipos pequeños (menos de 5 personas).

2.2.6. Eclipse

Eclipse[16] es una plataforma de software formado por herramientas *open source* para el desarrollo de aplicaciones.

A lo largo de nuestro paso por los estudios de grado en Ingeniería del Software, hemos utilizado este programa para desarrollar código, razón por la cual lo hemos escogido para la realización de la parte *backend* del proyecto.

2.2.7. Overleaf

Overleaf[17] es un gestor online de proyectos escritos en L^AT_EX. Gracias a los cursos formativos impartidos por la Oficina de Software Libre[18] realizado en años anteriores y a los recursos puestos a disposición de los alumnos[19] nos animamos a realizar la memoria con este editor online y en lenguaje de maquetación L^AT_EX.

2.2.8. MobaXterm

MobaXterm[20] es una herramienta muy versátil, entre sus funciones destacan la emulación de terminales o la conexión a un cliente SSH.

En este proyecto lo utilizamos con el objetivo de conectarse al servidor remoto proporcionado por Hostinger[14], en el cual albergamos el despliegue de la parte *backend* de nuestro proyecto.

2.3. Frameworks

2.3.1. Java Spring

Spring[21] es un *framework* del lenguaje de programación Java el cual nos permite desarrollar aplicaciones de manera más rápida, eficaz y corta, saltándose tareas repetitivas y ahorrando líneas de código.

Una de las características que define Java Spring de otros *frameworks* es la inversión de control, por la cual, es el *framework* y no el programador el que ejecuta ciertas operaciones sobre la aplicación, invirtiendo de esta manera los métodos de programación tradicionales. Otra de las características propias de Java Spring es la inyección de dependencias, por la cual Spring es capaz de crear o instanciar aquellos objetos que el programador necesite, favoreciendo el desacoplamiento y la manejabilidad de los mismos. Además de esto da soporte a una gran cantidad de *frameworks* a través de dependencias, facilitando la tarea del programador.

Hemos escogido este *framework* para realizar la parte *backend* de nuestra aplicación por las características citadas anteriormente, construyendo una *API-REST*. Esta *API-REST* actúa como un servidor, siendo consumida por un cliente.

Spring da soporte a una gran cantidad de herramientas a través de dependencias. Dichas dependencias se gestionan mediante Maven[22], una herramienta que simplifica los procesos de instalación.

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
```

Figura 2.1: Instalación Maven en Java Spring

Swagger

Swagger[23] es un proyecto *open source* para describir, gestionar, consumir y visualizar aplicaciones *API REST*. Para este proyecto vamos a utilizarlo con código Java, con el objetivo de generar la documentación de los puntos de acceso y de las entidades que se envían y se reciben.

Mockito

Mockito[24] es un *framework* utilizado junto con JUnit[25] en Java para realizar tests de manera sencilla. En este proyecto se va a utilizar para testear las capas de *Business* y *Controller*, con el objetivo final de conseguir un 100 % de cobertura en las clases testeadas.

JPA

JPA[26] (Java Persistence API) es una especificación de Java para acceder, persistir y gestionar datos entre Clases-Objetos de Java y bases de datos relacionales.

Gracias a los conocimientos de JPA adquiridos y desarrollados en las asignaturas de IS (Ingeniería del Software) y MS (Modelado de Software), nos decantamos por esta opción para gestionar las llamadas a las bases de datos correspondientes.

2.3.2. Angular

Angular[27] es un *framework* de desarrollo de aplicaciones SPA (Single Page Applications), el cual utiliza Typescript como lenguaje de programación. Este *framework* tiene la ventaja de que no refresca el navegador al modificar los elementos de la página web, dando una sensación de dinamismo y de inmersión al usuario.

Nos decantamos por utilizar este *framework* debido a su popularidad y al gran uso que se le da a nivel empresarial.

Bootstrap

Bootstrap[28] es una herramienta para crear interfaces de usuario limpias y totalmente adaptables a todo tipo de dispositivos y pantallas, sea cual sea su tamaño.

Escogimos esta herramienta ya que nos proporciona resultados óptimos y limpios de manera rápida, además de que posee una extensa documentación[29] a la que acudir en caso de duda.

CAPÍTULO 3

Desarrollo

En este capítulo se relata el proceso de desarrollo de esta aplicación web. Primero veremos la planificación inicial hablando de los diferentes artefactos que se generaron, posteriormente de cómo estos evolucionaron y se fueron adaptando, por último, del resultado final del desarrollo y lo aprendido en el camino.

3.1. Arranque del desarrollo

El primer artefacto que se generó es el Product Backlog, que recoge todas las funcionalidades deseadas por el equipo médico, en forma de historias de usuario (pequeños fragmentos de texto estructurado que intentan clarificar al máximo las mismas). Este se compone de una tabla con tres columnas: la primera de ellas recoge el grupo en el que se engloba la historia de usuario y su ID, la segunda una descripción corta de la misma y la última los criterios del comportamiento que debe tener la funcionalidad en los diversos escenarios posibles.

La generación de este artefacto fue compleja pues los médicos tenían claras dos o tres funcionalidades, que supondrán el núcleo de la aplicación (como veremos posteriormente). Del resto de la aplicación, o no tenían ninguna petición, o no había una opinión firme sobre ellas. Por ello, muchas de las funcionalidades de mediana o baja importancia fueron propuestas por el equipo de desarrollo o el director del TFG y el equipo médico se mostró más que contento de acogerlas prácticamente todas. A continuación se relatará y explicará cada sección de dicho Product Backlog y las modificaciones que sufrió. El documento completo en pdf para su cómoda lectura va adjunto a esta memoria.

El núcleo del que hablábamos anteriormente, y por tanto lo primero a remarcar en nuestro Product Backlog, se compone de estas tres funcionalidades:

HISTORIA DE USUARIO ID	DESCRIPCIÓN	CRITERIOS DE ACEPTACIÓN
TestMédico_RellenarDatosPaciente	Como investigador quiero llenar los campos del formulario con datos de un paciente para poder compararlos con otros a futuro y extraer conclusiones.	<ul style="list-style-type: none"> Cuando el investigador rellene los campos del formulario, si ha llenado todos los campos, entonces podrá pulsar en "Guardar" para confirmar y almacenar los datos. Cuando el investigador rellene los campos del formulario, si ha llenado los campos con valores pero alguna está en blanco, entonces podrá pulsar en "Guardar" pero se le impedirá confirmar el guardado indicándole un error.
TestMédico_ExtraerDatosTests	Como investigador quiero descargar un excel con todos los datos de los test almacenados para poder almacenarlos y compartirlos en el formato que me es más cómodo.	<ul style="list-style-type: none"> Cuando el investigador pulse "descargar excel" desde su vista principal, si ha llenado ya al menos un test y lo ha guardado, entonces se le descargará un excel con una fila por paciente con al menos un test y ambos test uno consecutivo al otro en la misma línea. Cuando el investigador pulse "descargar excel" desde su vista principal, si no ha llenado ya al menos un test y lo ha guardado, entonces el botón estará inhabilitado y no producirá ningún efecto además de verse más claro y sin relieve.
Login_AccesoComoInvestigador	Como investigador quiero acceder a la aplicación con mi DNI/NIE y contraseña para asegurar que esta solo está siendo utilizada por mí y mis compañeros.	<ul style="list-style-type: none"> Cuando el investigador pulse "login" desde la vista inicial de acceso a la aplicación, si ha llenado los campos con su DNI/NIE y contraseña correctos, entonces se le redirige a su vista principal. Cuando el investigador pulse "login" desde la vista inicial de acceso a la aplicación, si no ha llenado los campos con su DNI/NIE y contraseña correctos, entonces se le indicará con un mensaje de error que alguno de los dos campos es incorrecto.

Figura 3.1: Historias de usuario de mayor importancia, Walking Skeleton de la aplicación.

Como se puede apreciar en las historias de usuario lo principal era tener unos test donde llenar los datos de los pacientes de forma cómoda, que estos datos pudieran luego ser extraídos en un Excel y que el acceso a la aplicación fuese privado a los miembros del equipo médico. Las dos primeras suponen la entrada y salida más básica de datos que permitiría al equipo médico extraer sus conclusiones y la tercera, a pesar de ser solo el login, el acceso a la aplicación, es también de gran importancia. Uno de los requisitos principales del estudio es el control de la procedencia y gestión de los datos. Únicamente pacientes que se hayan ofrecido voluntariamente vía formulario pueden tener sus datos representados y solo los miembros oficiales del estudio pueden tener acceso a ellos. Esto se consigue mediante un login cerrado que no permite ningún tipo de registro desde el exterior y la creación de todos los perfiles de investigador manualmente por el administrador.

El siguiente bloque de funcionalidades complementa las tres primeras que en el primer prototipo emplearían perfiles introducidos a mano en la base de datos para probarse:

AdministrarPacientes_RegistrarPaciente	Como investigador quiero registrar a un paciente en el estudio para poder realizarle test y recopilar sus datos.	<ul style="list-style-type: none"> • Cuando el investigador pulse "registrar" desde su vista principal, si lo introducido en el campo de registro es un número de 8 dígitos, entonces se registrará al paciente y se habilitarán sus test para realizarse. • Cuando el investigador pulse "registrar" desde su vista principal, si lo introducido en el campo de registro no es un número de 8 dígitos, entonces se le indicará con un error que lo introducido en el campo de registro deben ser los últimos 8 dígitos de la tarjeta sanitaria del paciente.
AdministrarInvestigadores_RegistrarInvestigador	Como administrador quiero registrar manualmente cada investigador para asegurar que tengo el control del acceso a la aplicación.	<ul style="list-style-type: none"> • Cuando el administrador pulse "registrar" desde la vista principal para su perfil en la aplicación, si ha rellenado todos los campos del formulario que lo acompaña incluyendo un DNI/NIE válido y que no estaba registrado previamente, entonces se registrará un nuevo investigador notificándole el éxito de la operación. • Cuando el administrador pulse "registrar" desde la vista principal para su perfil en la aplicación, si no ha rellenado todos los campos del formulario que lo acompaña o no incluyendo un DNI/NIE válido y que no estaba registrado previamente, entonces se le mostrará un mensaje indicándole el error cometido para que lo subsane si lo desea.
Login_AccesoComoAdmin	Como administrador quiero acceder a la aplicación con mi DNI/NIE y contraseña para asegurar que solo yo tengo acceso a todas las funcionalidades asociadas a dicho perfil.	<ul style="list-style-type: none"> • Cuando el administrador pulse "login" desde la vista inicial de acceso a la aplicación, si ha rellenado los campos con su DNI/NIE y contraseña correctos, entonces se le redirige a su vista principal. • Cuando el administrador pulse "login" desde la vista inicial de acceso a la aplicación, si no ha rellenado los campos con su DNI/NIE y contraseña correctos, entonces se le indicará con un mensaje de error que alguno de los dos campos es incorrecto.
AdministrarInvestigadores_ModificarContraseñaInvestigador	Como administrador quiero modificar la contraseña de cualquier investigador para recuperar su perfil en caso de que tenga problemas para acceder a él con la actual.	<ul style="list-style-type: none"> • Cuando el administrador pulse en el ícono de modificación de un investigador en la vista de investigadores, si rellena el recuadro de nueva contraseña consiguiente y pulsa "modificar", entonces se sustituirá la contraseña del investigador seleccionado con la nueva introducida. • Cuando el administrador pulse en el ícono de modificación de un investigador en la vista de investigadores, si no rellena el recuadro de nueva contraseña consiguiente y pulsa "modificar", entonces se le notificará que el campo de nueva contraseña debe ser rellenado para realizar esta acción.

Figura 3.2: Historias de usuario sobre la inclusión del perfil de administrador y los diversos registros.

En este bloque se recogían los registros tanto de pacientes como investigadores así como el perfil de administrador y la modificación de contraseñas. Aparentemente, por lo explicado de parte del equipo médico, es un problema usual la pérdida de contraseña o el filtrado de las mismas por las condiciones de su ambiente de trabajo, y tener un acceso cómodo a su modificación es de gran importancia para ellos.

Las funcionalidades que prosiguieron en la escala de valor para el cliente fueron las siguientes:

AdministrarInvestigadores_ListarInvestigadores	Como administrador quiero listar todos los investigadores registrados en la aplicación para poder visualizarlos y modificarlos.	<ul style="list-style-type: none"> • Cuando el administrador navegue a su página principal de investigadores, si existe al menos un investigador registrado, entonces se le mostrará una lista con todos los investigadores registrados mostrando su DNI/NIE, nombre y género. • Cuando el administrador navegue a su página principal de investigadores, si no existe al menos un investigador registrado, entonces se le mostrará una lista vacía y se le indicará que aún no hay investigadores registrados.
Perfiles_ModificarContraseña	Como investigador/administrador quiero poder modificar mi contraseña para colocar una que me sea más fácil de recordar y solo yo conozca por seguridad.	<ul style="list-style-type: none"> • Cuando el investigador/administrador pulse "Modificar contraseña" desde la vista de su perfil, si ha rellenado todos los campos, su antigua contraseña coincide con la que tenía y la nueva es válida, entonces se le pide confirmación y se realiza el cambio de contraseña de ser afirmativa. • Cuando el investigador/administrador pulse "Modificar contraseña" desde la vista de su perfil, si no ha rellenado todos los campos, o su antigua contraseña no coincide con la que tenía o la nueva no es válida, entonces se le comunica el error mediante un mensaje para que lo subsane si lo deseas.
TestMédico_VisualizarDatosTest	Como investigador quiero visualizar todos los campos de un test ya guardado para comprobar que sean correctos y poder recordarlos cuando quiera.	<ul style="list-style-type: none"> • Cuando el investigador pulse el icono de "visualizar" del test de una cita en su tabla de citas completadas, si ha rellenado esa cita anteriormente, entonces se le redirigirá a una página con todos los campos de la cita acompañados de su valor en la cual no podrá modificar nada. • Cuando el investigador pulse el icono de "visualizar" del test de una cita en su tabla de citas completadas, si no ha rellenado esa cita anteriormente, entonces el botón le redirigirá en su lugar al formulario para la realización de dicha cita.
TestMédico_ModalesDeConfirmaciónyErrores	Como investigador quiero obtener feedback mediante modales al intentar guardar un test para saber en qué me he equivocado o no guardar sin estar seguro de quererlo.	<ul style="list-style-type: none"> • Cuando el investigador pulse el icono de "guardar" del test que esté realizando, si ha rellenado todos los campos con datos de los tipos esperados, entonces se mostrará un modal pidiéndole confirmación del guardado. • Cuando el investigador pulse el icono de "guardar" del test que esté realizando, si no ha rellenado todos los campos con datos de los tipos esperados, entonces se mostrará un modal indicándole que hay errores de tipología o algún campo vacío, referenciéndole el primero que se detecte.
AdministrarPacientes_ModificarCitaRealizada	Como administrador quiero modificar los campos del test de una cita ya realizada para subsanar errores que se hayan cometido en su creación.	<ul style="list-style-type: none"> • Cuando el administrador pulse el icono de "modificar" en un test realizado dentro de su vista de citas, si dicho test está completado, entonces le redirigirá a una vista con los datos previos del test y campos en los que pueda introducir valores nuevos para todos, alguno o ningún campo que luego pueda guardar. • Cuando el administrador pulse el icono de "modificar" en un test realizado dentro de su vista de citas, si dicho test no está completado, entonces no aparecerá siquiera como opción en la lista.
AdministrarPacientes_ListarPacientes	Como administrador quiero listar todos los pacientes registrados en la aplicación para poder visualizarlos y modificarlos.	<ul style="list-style-type: none"> • Cuando el administrador navegue a su página de pacientes, si existe al menos un paciente registrado, entonces se le mostrará una lista con todos los pacientes registrados mostrando su identificador. • Cuando el administrador navegue a su página de pacientes, si no existe al menos un paciente registrado, entonces se le mostrará una lista vacía y se le indicará que aún no hay pacientes registrados.
AdministrarPacientes_ListarCitasRealizadas	Como administrador quiero listar todas las citas realizadas en la aplicación para poder visualizarlas y modificarlas.	<ul style="list-style-type: none"> • Cuando el administrador navegue a su página de citas, si existe al menos una cita registrada, entonces se le mostrará una lista con todas las citas realizadas mostrando identificador de la cita, paciente e investigador implicados. • Cuando el administrador navegue a su página de citas, si no existe al menos una cita registrada, entonces se le mostrará una lista vacía indicándole que aún no se ha realizado el test para ninguna cita.

Figura 3.3: Historias de usuario sobre listado de usuarios y diversas modificaciones.

Estas funcionalidades se componen de dos tablas que listan de forma limpia tanto investigadores para el perfil de administrador, como pacientes para el de investigador. La funcionalidad de modificar contraseña de nuevo aparece, pero esta vez es para que cada investigador pueda cambiar su propia clave. Esta funcionalidad es principalmente para que el administrador pueda crear sus perfiles con contraseñas que ellos mismos puedan modificar al obtener las cuentas a algo fácil de recordar para ellos mismos. Además, se añade la posibilidad de visualizar en una plantilla los datos de un test realizado, ahorrando tener que extraer el Excel y consultarla cada vez, cuando solo se quiere revisar el test de una cita en concreto. Se añaden modales para indicar errores y confirmación en los test, importante ya que estos no pueden ser modificados una vez guardados a no ser que lo haga el administrador manualmente. Esta posibilidad de modificar una cita por el administrador es la última funcionalidad no comentada de este bloque junto con su listar particular para facilitar el trabajo de este.

El penúltimo bloque de funcionalidades que se extrajo al principio del desarrollo, empieza a corresponder ya a elementos de gestión necesarios pero no prioritarios que en su mayoría fueron propuestos por el equipo de desarrollo como posibles añadidos prácticos:

AdministrarInvestigadores_EliminarInvestigador	Como administrador quiero eliminar a un investigador para privarle de acceso a la aplicación.	<ul style="list-style-type: none"> • Cuando el administrador pulse el icono de "eliminar" en un investigador dentro de su vista de investigadores, si dicho investigador existe y no tiene pacientes asociados, entonces se le pedirá confirmación de su intención y en caso afirmativo se eliminará al investigador. • Cuando el administrador pulse el icono de "eliminar" en un investigador dentro de su vista de investigadores, si dicho investigador no existe, entonces no aparecerá siquiera en la lista. • Cuando el administrador pulse el icono de "eliminar" en un investigador dentro de su vista de investigadores, si dicho investigador tiene pacientes asignados, entonces lo indicará en un mensaje de error que no se pueden eliminar investigadores con pacientes actualmente a su cargo.
AdministrarInvestigadores_FiltrarInvestigadoresPorDNI	Como administrador quiero filtrar los investigadores existentes en la aplicación por DNI/NIE para encontrarlos más fácilmente.	<ul style="list-style-type: none"> • Cuando el administrador pulse el icono de "buscar" en su vista principal, si ha rellenado el recuadro adjunto con un DNI/NIE válido y que está registrado en la aplicación, entonces le filtrará la lista de investigadores a una única tupla con el DNI/NIE buscado. • Cuando el administrador pulse el icono de "buscar" en su vista principal, si ha rellenado el recuadro adjunto con un DNI/NIE válido, entonces se le indicará en un mensaje de error que este solo acepta DNI/NIE en formato completo. • Cuando el administrador pulse el icono de "buscar" en su vista principal, si ha rellenado el recuadro adjunto con un DNI/NIE válido pero que no está registrado en la aplicación, entonces mostrará una lista vacía sin resultados indicando que ningún investigador ha sido registrado con dicho DNI/NIE.
AdministrarInvestigadores_FiltrarInvestigadoresEnOrdenAlfabético	Como administrador quiero filtrar los investigadores existentes en la aplicación por orden alfabético para encontrarlos más fácilmente.	<ul style="list-style-type: none"> • Cuando el administrador pulse el icono de "ascendente" sobre la lista de investigadores en su vista principal, si existen al menos dos investigadores registrados en la aplicación, entonces se recolocarán de menor a mayor según su DNI/NIE. • Cuando el administrador pulse el icono de "descendente" sobre la lista de investigadores en su vista principal, si existen al menos dos investigadores registrados en la aplicación, entonces se recolocarán de mayor a menor según su DNI/NIE. • Cuando el administrador pulse el icono de "descendente" o "ascendente" sobre la lista de investigadores en su vista principal, si no existen al menos dos investigadores registrados en la aplicación, entonces la lista de investigadores no sufrirá ninguna modificación.
AdministrarPacientes_EliminarPaciente	Como administrador quiero eliminar a un paciente para sacarle del estudio médico.	<ul style="list-style-type: none"> • Cuando el administrador pulse el icono de "eliminar" en un paciente dentro de su vista de pacientes, si dicho paciente existe, entonces se le pedirá confirmación de su intención y en caso afirmativo se eliminará al paciente y sus test asociados. • Cuando el administrador pulse el icono de "eliminar" en un paciente dentro de su vista de pacientes, si dicho paciente no existe, entonces no aparecerá siquiera en la lista.

Figura 3.4: Historias de usuario sobre eliminación de perfiles y agilidad en las búsquedas.

Como se puede observar, este bloque recogía principalmente la eliminación de perfiles, necesaria si algún paciente o investigador salía del proyecto. Esta funcionalidad, aunque simple, realmente no se añadió a este artefacto en un primer momento porque generaba dudas por la naturaleza del estudio. Los datos extraídos de la aplicación, para ser válidos, deben ser privados, de distintos períodos temporales y completos. La eliminación en este caso de un paciente podía dejar datos incompletos, o la de un investigador pacientes con citas a medio completar. Cualquiera de estos escenarios invalidaría el estudio. Por ello, en primera instancia, se pensó en no permitir ningún tipo de eliminación, pero tras debatir con el equipo médico este mismo aseguró que las eliminaciones serían pocas o prácticamente nulas y solo se usarían para subsanar erratas a la hora de crear perfiles nuevos sin afectar a los datos. Tras ello se decidió añadirlas, no sin delimitar claramente en sus criterios de aceptación las condiciones para poder realizar las eliminaciones.

La otra funcionalidad que apareció fueron los filtrados, los cuales habían sido implementados en otras aplicaciones por el equipo de desarrollo así que no suponían un gran esfuerzo y podían ser moderadamente útiles. Los médicos no se mostraron entusiastas con la idea pues apenas habría 15 investigadores pero ciertamente era una funcionalidad que podrían usar eventualmente así que se decidió añadirla.

El último grupo de funcionalidades que cerraba este artefacto eran las menos prioritarias. Entre ellas un filtrado también para pacientes, la muestra de datos para cada investigador en su propio perfil y la inclusión de estadísticas. Esta última funcionalidad lamentablemente nunca pudo llevarse al desarrollo pues se prefirió cerrar la aplicación para poder tenerla funcionando y ajustarla a las necesidades del equipo médico a embarcarse en esta funcionalidad que nunca despertó mucho interés en el equipo:

AdministrarPacientes_FiltrarPacientesPorIdentificador	Como administrador quiero filtrar los pacientes por identificador para encontrarlos más rápido.	<ul style="list-style-type: none"> Cuando el administrador pulse el icono de "buscar" en su vista de pacientes, si ha rellenado el recuadro adjunto con un identificador de paciente válido y que está registrado en la aplicación, entonces le filtrará la lista de pacientes a una única tupla con el identificador buscado. Cuando el administrador pulse el icono de "buscar" en su vista de pacientes, si no ha rellenado el recuadro adjunto con un identificador de paciente válido, entonces se le indicará en un mensaje de error que este solo acepta identificadores de pacientes formados por los últimos 8 dígitos de sus tarjetas sanitarias. Cuando el administrador pulse el icono de "buscar" en su vista de pacientes, si ha rellenado el recuadro adjunto con un identificador de paciente válido pero que no está registrado en la aplicación, entonces mostrará una lista vacía sin resultados indicando que ningún paciente ha sido registrado con dicho identificador.
Perfiles_PerfilInvestigadorConDatos	Como investigador quiero ver los datos de mi perfil para comprobar que son correctos y recordar como los rellene.	<ul style="list-style-type: none"> Cuando el investigador navegue a su pestaña de perfil, si no ha habido ningún problema con su registro, entonces mostrará una ventana con los datos de su cuenta y la opción de modificar su contraseña si lo desea. Cuando el investigador navegue a su pestaña de perfil, si ha habido algún problema con su registro, entonces ni siquiera tendrá acceso a la navegación en la aplicación pues no podrá hacer login y deberá ponerse en contacto con el administrador.
Estadísticas_EstadísticasGlobalesTestRealizados	Como investigador quiero unas gráficas que engloben los resultados de todos los test que he realizado para poder tener una referencia visual del estado actual de sus datos.	<ul style="list-style-type: none"> Cuando el investigador navegue a su pestaña de estadísticas, si ha realizado al menos un test, entonces se le mostrará una gráfica que engloba las incidencias totales de cada campo entre todos sus test. Cuando el investigador navegue a su pestaña de estadísticas, si no ha realizado al menos un test, entonces se le mostrará un mensaje indicando que aún no ha realizado ningún test así que no hay datos con los que alimentar las gráficas.

Figura 3.5: Historias de usuario sobre filtrado de pacientes, perfiles de investigador y estadísticas.

Estas fueron todas las funcionalidades añadidas al Product Backlog al principio del desarrollo. En el siguiente apartado, el de desarrollo, se irán comentando los cambios que sufrieron estas historias y como aparecieron incluso algunas nuevas.

Junto a este artefacto se creó también un User Story Map, que permite visualizar mejor todas las historias comentadas anteriormente y, además, marca un flujo de uso de las mismas de izquierda a derecha y secciona las funcionalidades en las releases mensuales en las que esperábamos tenerlas:

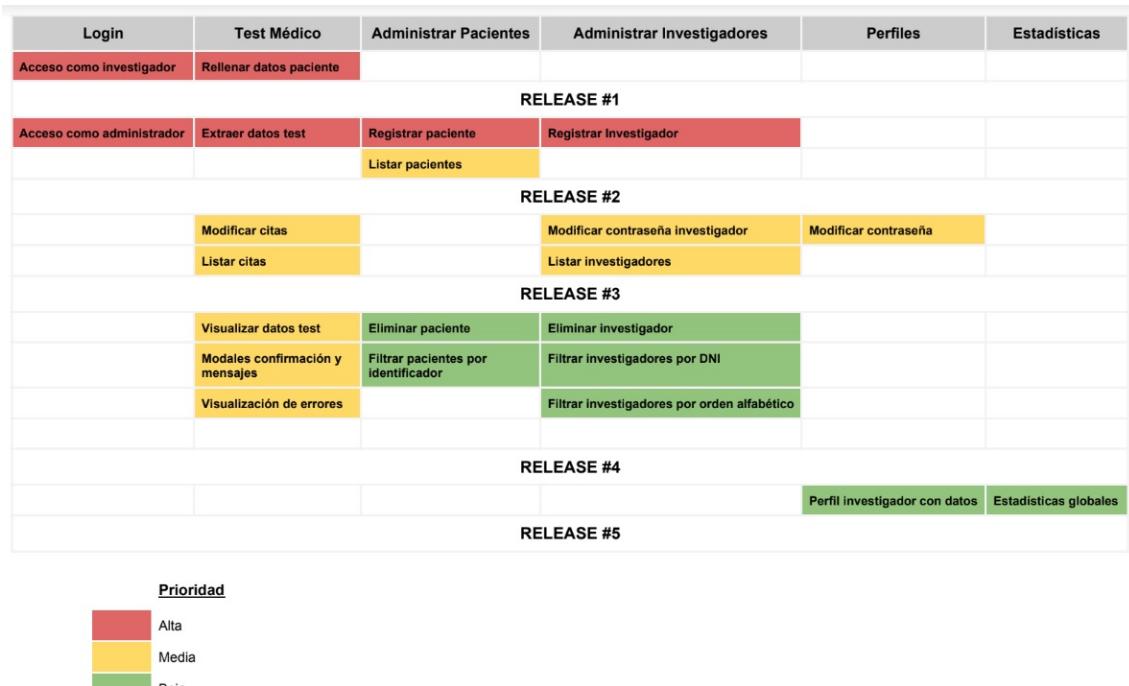


Figura 3.6: Mapa de historias de usuario.

Este mapa recoge todas las historias relatadas en este apartado y marca las divisiones en releases que seguirá el próximo para hablar de como fue el desarrollo de la aplicación, así que puede ser consultado en cualquier momento para clarificar de qué secciones de este apartado habla cualquier punto del desarrollo durante los sprints. Nótese que en ocasiones se hablará de sprints y en otras de releases, ya que en nuestro caso cada uno de los primeros coincide con una de las segundas.

3.2. Desarrollo durante los sprints

Con nuestros artefactos listos y la primera reunión fijada para principios de octubre comenzó nuestro primer sprint en septiembre. Nuestro objetivo, como se puede apreciar en el User Story Map anterior, era el acceso a la aplicación como investigador y la posibilidad de llenar tests de pacientes. La idea era empezar con el modelo de datos de los test, pensar en su extracción y después hacer el login, sin embargo, faltaban detalles sobre la estructura y limitaciones de los tests así que intercambiamos el orden a la espera de información más precisa. Visto lo cual comenzamos con el acceso a la aplicación.

A pesar de haber estado durante el mes de agosto viendo tutoriales y practicando con las tecnologías al ponernos de verdad sobre el proyecto comprobamos que aún teníamos que estudiar en profundidad muchos aspectos. Eduardo por suerte en el trabajo que había comenzado ese mismo verano había utilizado parte de las herramientas aunque no fuese en profundidad, pero Sergio no conocía herramientas como Angular o Spring Boot hasta hace un par de semanas. El primer mes se nos escapó de las manos prácticamente en configuración del software necesario y de los proyectos tanto en frontend como backend, la conexión de todo el sistema y el diseño de la simple página de login así como su funcionalidad. Para cuando llegó octubre el consumo de tareas iba mucho más lento de lo esperado y la primera reunión prácticamente solo sirvió para cerrar el diseño de la aplicación en líneas generales, la paleta de colores, el tipo de letra, el espaciado y tamaño, etc, y clarificar más detalles sobre el formulario de test para las citas. Este retraso en las tareas y desajuste del orden de desarrollo se mantendría lamentablemente para el siguiente sprint aunque se iría ajustando en los siguientes. A continuación se recoge una captura del login (ver figura 3.7).

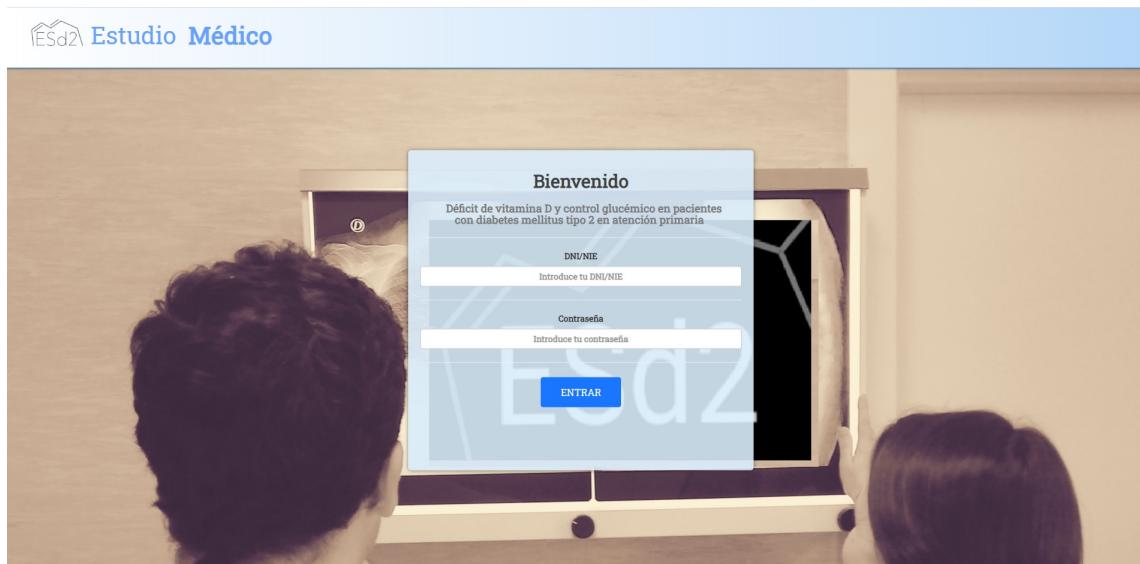


Figura 3.7: Captura del login de la aplicación.

Tras este primer sprint con retraso, el equipo de desarrollo al fin tenía todo listo y ordenado para trabajar en condiciones óptimas. Mientras Sergio se centraba en el formulario y el registro de pacientes, Eduardo pasó a las primeras tareas que no requerían el modelo de datos del test, el registro de investigadores y, a su vez, la creación y acceso del perfil de administrador. Al final de este segundo sprint el formulario estaba finalizado y los pacientes tenían su propio registro como se muestra en las siguientes figuras.

La captura de pantalla muestra la interfaz web de 'Estudio Médico'. En la parte superior, hay un menú con 'PACIENTES' y 'DOCUMENTACIÓN'. La barra superior también incluye el nombre del investigador ('Investigador | Eduardo'), botones para 'ADMINISTRAR', 'PERFIL' y 'DESCONECTAR', y un icono de perfil. La sección principal es 'LISTA PACIENTES', que muestra una tabla con los datos de los pacientes. Los campos visibles en la tabla son 'IDENTIFICADOR PACIENTE' (que muestra '12345678' y '1111111'), 'PRIMERA CITA' (con botones 'Consultar' y 'Identificador') y 'SEGUNDA CITA' (con botones 'Consultar' y 'Cita 2'). Hay botones para 'Generar Excel' y 'Añadir Paciente' en la parte superior de la tabla.

Figura 3.8: Vista principal del investigador con su lista de pacientes.

Esta captura (figura 3.8) muestra la imagen final de la vista principal del investigador con su lista de pacientes. En ese momento no existía aún el botón de generar Excel ni el de arriba a la derecha en el que se lee “administrar”. Por lo demás el diseño se ideó y se mantuvo así desde un primer momento en toda la página a excepción de los iconos, que vendrían a futuro como un nuevo requisito.

La captura de pantalla muestra el formulario de test para pacientes. Se divide en tres secciones principales: 'Variables principales', 'Variables sociodemográficas' y 'Hábitos de vida'. La sección 'Variables principales' incluye campos para 'Vitamina D' (que muestra un cuadro vacío), 'HbAlc' (que muestra un cuadro vacío) y 'Estación del año' (que muestra 'Verano' y 'Invierno' con radios). La sección 'Variables sociodemográficas' incluye campos para 'Sexo' (que muestra radios para 'Hombre' y 'Mujer'), 'Nivel de estudios' (que muestra un cuadro desplegable) y 'Fecha de nacimiento' (que muestra un cuadro vacío). La sección 'Hábitos de vida' incluye campos para 'Tabaco' (que muestra radios para 'Sí' y 'No'), 'Consumo de riesgo de alcohol' (que muestra radios para 'Sí' y 'No'), 'Tiempo de exposición solar al día' (que muestra un cuadro vacío), 'Uso de crema SPF' (que muestra un cuadro vacío), 'Puntuación SPF' (que muestra un cuadro vacío) y 'Ejercicio físico' (que muestra un cuadro vacío).

Figura 3.9: Formulario de test para pacientes.

En cuanto al formulario, tanto el diseño como los campos (que eran imposible incluir en su totalidad en la figura 3.9) se crearon y mantuvieron como se ven en la figura anterior. Posteriormente se le añadirían algunas funcionalidades extra pero sin cambiar su aspecto.

Figura 3.10: Vista principal del administrador con la lista de investigadores.

En la figura 3.10 se observa la lista de investigadores en la ventana principal de administrador, con el mismo diseño que la de pacientes pero con dos botones añadidos que permitirían posteriormente modificar la contraseña de investigador y eliminarlo. Al lado izquierdo se ve el formulario de registro de investigador que aparece completo en la figura 3.11.

Figura 3.11: Formulario para registrar nuevos investigadores como administrador.

Las claves de acceso a la aplicación son el DNI y la contraseña. Como se observa en la figura, se puede utilizar un NIE como clave, este detalle fue introducido a posteriori con el despliegue de la web ya que una de las investigadoras es extranjera.

Con estas funcionalidades implementadas, la segunda release se llevó a cabo con todas las tareas consumidas a excepción de la extracción de datos, que sería lo primero a retomar en el siguiente sprint. El equipo médico mostró conformidad con todo el desarrollo hasta la fecha aunque solicitó algunos añadidos para el formulario. Durante la demo realizada en la reunión el médico representante del equipo y codirector del TFG, Alejandro, notó que el formulario no daba indicaciones de qué valores se esperaban en los diferentes campos y no daba suficiente información de los errores. Los miembros del equipo de desarrollo no sabíamos nada acerca de qué valores serían válidos o la información que les sería útil en un mensaje de error en el formulario así que Alejandro recopiló por cuenta propia todos estos datos y nos los envió en un documento por correo al par de días.

VARIABLES DEL CUESTIONARIO y rangos de dichas variables

Variables principales:

Vitamina D (ng/ml) 0-500
HbA1c (%) 1-40
Estación del año: verano/ invierno (V/I)

Variables Sociodemográficas:

Sexo: hombre/mujer
Nivel de estudios: sin estudios, primaria, ESO, bachillerato, universitarios
Edad en años: 18-100
Fecha de nacimiento
Nivel socioeconómico (ingreso unidad familiar en euros mensuales): 0 – 1000 | 1001 – 2000 | 2001 – 3000 | >3000

Hábitos de vida:

Tabaco: si/no
Consumo de riesgo de Alcohol : si/no
Tiempo de exposición solar al día (min): 0-1200
Uso de crema con SPF: si/no
Puntuación SPF: 10/15/20/25/30/50
Ejercicio físico (horas a la semana): 0-700

Variables clínicas:

Control adecuado de la DM2 (HbA1c<7 en pacientes sin complicaciones y <8,5 en pacientes anciano): sí/no
Glucemia (mg/dl): 0-700
IMC (kg/m²): 10-70
Obesidad: sí/no
TAS (mmHg): 20-300
TAD (mmHg): 20-200
Hipertensión arterial (TA ≥140/90): SI/No
Colesterol total (mg/dl): 50-900
LDL -col (mg/dl): 1-400
HDL -col mg/dl: 1-400
TG (mg/dl): 1-10000
Dislipemia (colesterol total> 240) si/no
Creatinina (mg/dl): 0-30
Estimación del filtrado glomerular (MDRD4): 0-200
Insuficiencia renal crónica: sí/no
Fototipo: I-VI (I-II-III-IV-V-VI)
Tratamiento para la diabetes (intensificación farmacológica): sí/no
Suplementación con vitamina D: sí/no

Figura 3.12: Documento con rangos de variables y tipo de campos esperados en el formulario.

Estos valores se añadieron posteriormente al formulario en forma de tooltips, pequeños bocadillos de información al pasar el cursor sobre los nombres de cada campo del formulario.



Figura 3.13: Tooltip de información sobre el rango de valores esperado en un campo del formulario.

Además se añadió un remarcado en rojo para cuando el valor estuviese fuera de ese rango o no fuese siquiera un número y uno en verde cuando el contenido del campo cumpliese los requisitos establecidos.

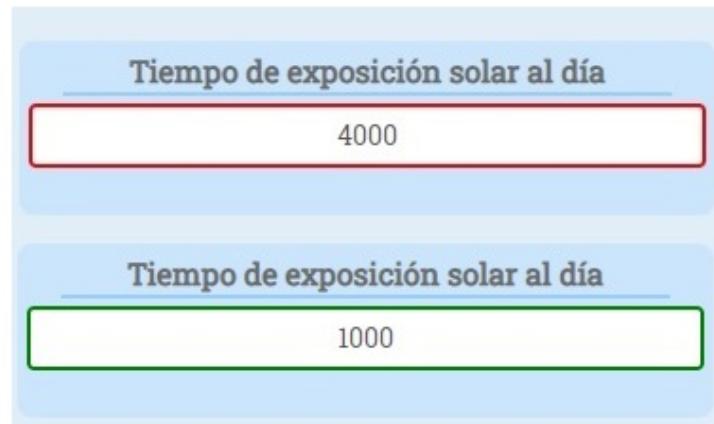


Figura 3.14: Remarcado de colores para visualizar si el valor cumple los requisitos del campo del formulario.

En el siguiente sprint todas las funcionalidades se centraban en la modificación de los diferentes elementos de la aplicación, sin embargo tras hablar con el equipo médico se llevaron a cabo varios cambios en el orden de prioridad:

- Las funcionalidades referentes al perfil (en este punto sobre todo la modificación de contraseña) se relegaron a una prioridad baja. Ciertamente la posibilidad de cambiar la contraseña por el usuario era importante para los médicos pero no más que la visualización de los datos del formulario o las funcionalidades aún restantes de gestión para los usuarios.
- Relacionado con lo dicho en el punto anterior, eliminar perfiles y filtros pasaron a una prioridad media y se pusieron al nivel de visualizar datos test. Estas historias de usuario pasaron a la release 3.
- En cuanto a la historia de modificar citas y listarlas, estas mantuvieron su prioridad, pero su carga de trabajo, con todos los nuevos detalles de rangos y validaciones añadidos al formulario, se había vuelto abrumadora. El equipo lo comentó con el representante de los médicos y este dio el visto bueno a posponerla ya que era una herramienta de corrección de errores y todo nuestro trabajo en el momento sobre el formulario se encaminaba a la prevención de los mismos al máximo.

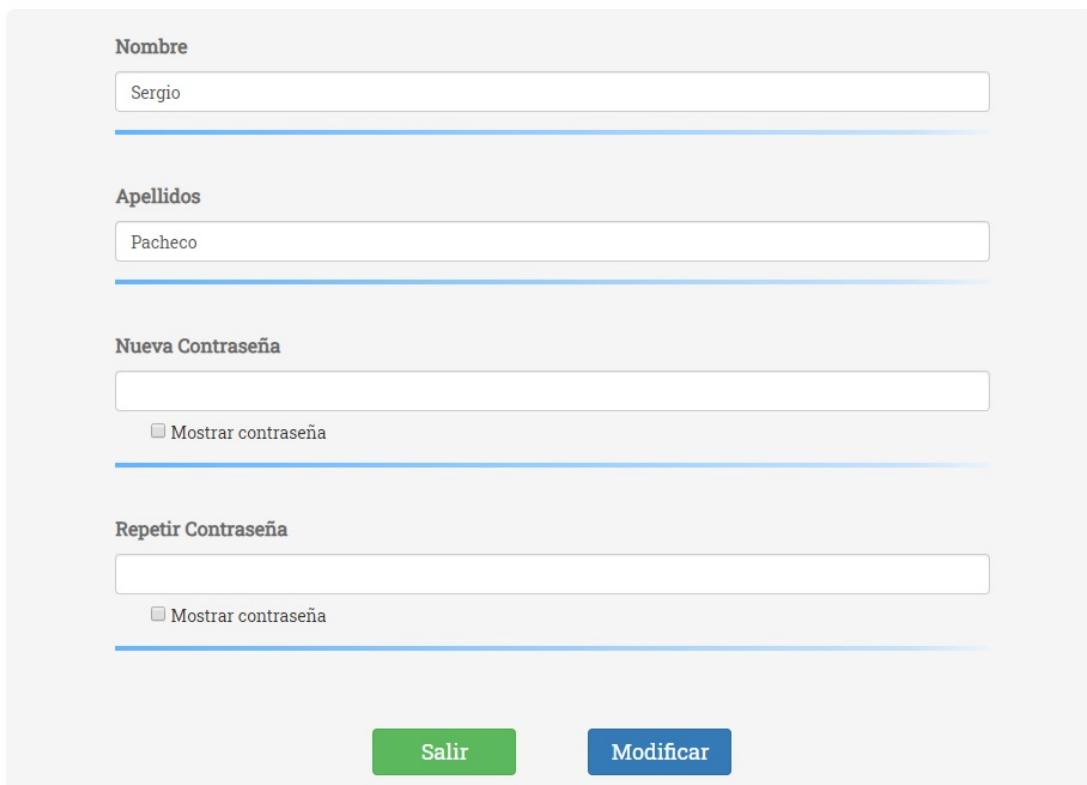


Figura 3.15: Calendario mejorado, mucho más fácil de manejar que escribir la fecha manualmente, lo que llevaba a errores.

- Por último, aunque no se añadiese una nueva, se mantuvo abierta la historia de llenar datos paciente ya que durante todo este sprint se siguió trabajando en mejoras para el mismo y detalles que se iban comentando por correo con el equipo médico.

Con esta nueva organización se continuó con el desarrollo. La historia de “Modificar contraseña investigador” se amplió, ya que suponía poco trabajo extra y resultaba interesante, a modificación de cualquier campo del investigador.

Modificar Investigador Sergio Pacheco



The form is titled "Modificar Investigador" and has the name "Sergio Pacheco" pre-filled. It includes fields for "Nombre" (Name) and "Apellidos" (Last Name). There are also fields for "Nueva Contraseña" (New Password) and "Repetir Contraseña" (Repeat Password), each with a "Mostrar contraseña" (Show password) checkbox. At the bottom are "Salir" (Exit) and "Modificar" (Modify) buttons.

Figura 3.16: Formulario para la modificación de datos de un investigador por parte del administrador.

Los listar, tanto pacientes como investigadores, fueron finalizados y junto con otros elementos visuales rediseñados para incluir iconos. Estos iconos fueron una propuesta del profesor director en la que nunca se pensó pero que ciertamente añadían usabilidad a la interfaz general de la aplicación.

En cuanto a la eliminación de perfiles, como se comentó en el punto de arranque del proyecto no está permitido eliminar investigadores con pacientes asignados y los pacientes pueden eliminarse tengan o no citas realizadas, pero en el caso de tenerlas se pedirá una confirmación extra para evitar posibles problemas. En caso de confirmar la eliminación, sus citas completadas por el momento, así como el perfil del paciente serán eliminadas de la base de datos.



Figura 3.17: Mensaje de confirmación para la eliminación de un paciente con algún test de sus citas relleno.

Se añadieron los filtros tanto para pacientes como para investigadores. En el caso de los investigadores se sometió a debate la posibilidad de implementar una lista desplegable en vez de un buscador por DNI escrito, pero el numero de investigadores (unos 15) dejaba la lista algo larga como para ser cómoda y el buscador algo amplio para ser tan pocos. Finalmente, se decantó por el buscador, principalmente por mantener una concordancia y similitud con el sistema empleado para los pacientes.

DNI/NIE Investigador

Buscar 

Nº Identificación Paciente

Buscar 

Figura 3.18: Filtros para la búsqueda por DNI de investigador o identificador de paciente.

Terminada esta tercera release, la reunión con el representante del equipo médico y el profesor director del TFG se llevó a cabo. En cuanto al formulario de test para las citas, quedó cerrado a excepción de un pequeño añadido que hizo ver el profesor: en los campos que puedan admitir o no decimales el mensaje de información sobre su contenido debería clarificarlo. Por la parte de gestiones de usuarios tanto modificación, eliminación o filtrado no hubo ninguna queja, aunque era de esperar pues sus condiciones ya habían sido habladas múltiples veces tanto de forma presencial como por email. Con todo revisado y los detalles de algunas historias apuntados se procedió con el cuarto sprint, el último, con una gran carga de tareas.

Finalmente, se llevó a cabo la extracción de datos de la aplicación a un Excel. En un primer momento se hizo dejando una línea para cada cita de cada paciente, pero por petición del equipo médico tras ver el resultado se modificó a una sola línea por paciente con sus dos citas consecutivas en la misma.

Éxito! Documento excel generado correctamente

Lista Pacientes									
<input type="button" value="Actualizar Lista"/> <input type="button" value="Generar Excel"/>									
A	B	C	D	E	F	G	H	I	
IDENT_PACIENTE	FECHA_REAL	VITAMINA_D	HBA1C_1	ESTACIÓN	SEXO	NIVEL_ESTUDIOS	FECHA_NACIMI	NIVEL_SODA	
12345678	06/05/1997	77	20	verano	mujer	Primaria	06/05/1997	1001-200	
11111111	04/01/1981	200	20	invierno	mujer	Universitarios	04/01/1981	>3000	

Figura 3.19: Mensaje de confirmación de la extracción de datos y documento Excel resultante.

La visualización de los datos del test se implementó siguiendo el diseño del formulario de recopilación de datos pero listando los campos para facilitar su lectura. Además, se mantuvieron los tooltip para permitir su consulta en todo momento y no ofuscar ninguna información.

The screenshot shows a user interface for viewing test results. At the top left is a section titled "Datos de la cita" containing fields for "Identificador de paciente: 12345678" and "Fecha de realización: 6-5-1997". Below this is a section titled "Variables principales" containing fields for "Vitamina D: 77" (with a note "(0-500) ng/ml"), "HbA1c: 20", and "Estación del año: verano".

Figura 3.20: Fragmento de la visualización de un test realizado.

Se añadieron y mejoraron mensajes de error y modales de confirmación para la inserción de datos en la aplicación, ya fuese de usuario o de los propios test sobre pacientes.



Figura 3.21: Modal de confirmación para el guardado de un test.

Se cerraron todas las historias que aún acumulaban modificaciones hasta la fecha como *Rellenar datos paciente* y se completaron ambas historias de la sección de *Perfiles* con la modificación de contraseña propia para cada investigador y la visualización de sus datos.

The screenshot shows an "INFORMACIÓN INVESTIGADOR" page. It displays the user's name as "NOMBRE: Eduardo" and "APELLIDOS: Gonzalo". At the bottom are two buttons: "Volver" (blue) and "Modificar contraseña" (orange).

Figura 3.22: Perfil de investigador.

Por último, en este sprint se llevó a cabo la implementación de las historias *Modificar citas* y *Listar citas*, que se llevaron una gran parte de la carga de trabajo del mismo. Para la lista de citas se decidió añadir una nueva historia de usuario *Filtrar citas por identificador de paciente* para dar alguna herramienta de filtrado en esta lista que será presumiblemente la más larga. Además, se mantuvieron los filtros de menor a mayor ya implementados en anteriores listas. Se añadió un pequeño botón de actualización sobre la lista ya que las citas son los componentes más cambiantes de la aplicación y pueden estarse añadiendo frecuentemente al mismo tiempo que se consulta esta tabla. Para evitar que el usuario tenga que recargar la página para ver estos cambios y recordarle que pueden estarse sucediendo se añade este botón.

Lista Citas Pacientes		
Nº Identificación Paciente	NÚMERO IDENTIFICACIÓN	CITA REALIZADA
<input type="text" value="12345678"/>	12345678	Cita 1 
<input type="text" value="11111111"/>	11111111	Cita 1 
<input type="text" value="12345678"/>	12345678	Cita 2 

Figura 3.23: Perfil de investigador.

En cuanto a la modificación de las citas por parte del administrador, se decidió mantener el estilo de la visualización de citas realizadas ya que era más compacto pero añadiendo campos para poder llenar aquellos datos que quisieran modificarse. Estas modificaciones siguen los mismos criterios de entrada que el test original, reaccionan de igual manera con el patrón de colores a las inserciones válidas o inválidas y tienen los mismos mensajes de error, a excepción de que este formulario no requiere todos los campos rellenos.

Datos de la cita	Identificador de paciente: 12345678
	Fecha de realización: 6-5-1997
Variables principales	Vitamina D: 77 HbA1c: 20 Estación del año: verano
	(0-500) ng/ml. Los decimales se introducen separados por un punto
	<input checked="" type="radio"/> Invierno <input type="radio"/> Verano

Figura 3.24: Formulario de modificación para un test realizado sobre una cita.

Como se ha visto en las figuras y se recoge en el mapa de historias de usuario de la sección anterior, con esto quedaba completada la mayor parte de la aplicación. La ultima reunión en diciembre de 2019 se llevó a cabo y aunque la siguiente técnicamente debía ser a finales de enero con las festividades y los exámenes se tuvo que posponer a febrero. En vistas de las fechas que se manejaban ya, como se comentó anteriormente, se dejó de lado el apartado de estadísticas y se empezó a trabajar en el despliegue de la aplicación del que se hablará en un próximo capítulo. Además, los artefactos utilizados tanto para planificación como gestión se mantuvieron durante esta fase de despliegue, y tras esta, durante el mantenimiento del que igualmente se hablará en su propio capítulo.

CAPÍTULO 4

Implementación

En este capítulo se explican los detalles de la implementación de las partes *frontend* y *backend* de nuestro proyecto, tanto de la generación del proyecto inicial como de la infraestructura del mismo.

4.1. Frontend

En esta sección se va a hablar del proyecto Angular el cual compone la parte *frontend* de nuestra aplicación. Este *framework* nos ofrece un desarrollo más rápido con respecto a otros *frameworks*, ya que se estructura en diferentes componentes web y genera una página web dinámicamente. Dicho proyecto se almacenó y gestionó mediante este repositorio de GitHub[30].

4.1.1. Generación Proyecto Angular

Para la generación de un proyecto base en Angular se utilizó *Angular Cli*, un herramienta de línea de comandos que facilita la creación y gestión de proyectos Angular y de sus diferentes componentes. *Angular Cli*, al tratarse de una herramienta NodeJS, requiere tenerlo instalado en nuestro sistema operativo. Para ello la primera tarea a realizar es instalar NodeJS a través de su página oficial[31], escogiendo la versión que soporte nuestro sistema operativo.

A continuación, a través del terminal que nos proporciona la herramienta de desarrollo *Visual Studio Code*, ejecutamos el comando `npm install -g @angular/cli@latest`, el cual nos instala *Angular Cli* en la última versión disponible.



```
Angular CLI: 8.3.4
Node: 10.15.1
OS: win32 x64
Angular: 8.2.6
... animations, common, compiler, compiler-cli, core, forms
... language-service, platform-browser, platform-browser-dynamic
... router
```

Figura 4.1: Versión Angular Cli

Una vez instalada la herramienta de comandos *Angular Cli*, procedemos a crear nuestro proyecto base Angular ejecutando el comando `ng new estudio-medico-tfg2019`, generando un proyecto completo Angular junto con toda su estructura de carpetas.

4.1.2. Arquitectura de la aplicación Angular

Angular es una herramienta que se sustenta en la idea de modular del código en componentes. Estos componentes son fragmentos de código independientes que la página web carga o descarga según las acciones del usuario.

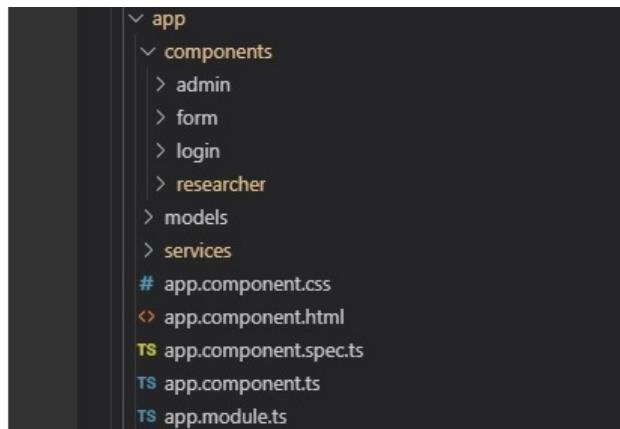


Figura 4.2: Estructura de componentes en Angular.

En el caso de nuestro proyecto se emplearon cuatro grandes componentes (admin, form, login y researcher) así como algunos más pequeños que se desglosarán más adelante. El componente app que los engloba es creado por Angular automáticamente y sirve como base para montar y desmontar el resto de componentes.

login

El primer componente que se creó en la aplicación y sobre el que se desarrolló el diseño general que se seguiría en el resto de la misma. Como todos los componentes de esta aplicación se compone de un archivo HTML con la estructura del elemento, un archivo CSS con el diseño del mismo y finalmente un archivo de TypeScript con los comportamientos posibles del componente.

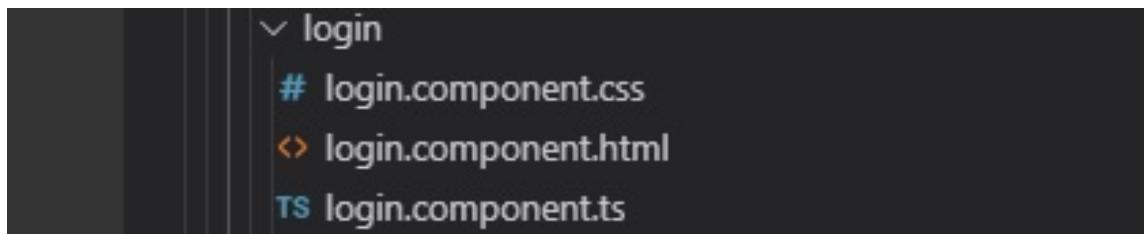


Figura 4.3: Estructura del componentes de login.

En cuanto a la estructura el elemento se compone solamente de la cabecera, un formulario para introducir usuario y contraseña y un modal para mensajes de error. Aprovecharemos que este es uno de los formularios más sencillos de la aplicación para explicarlo un poco más en detalle, ya que estos son una parte fundamental de la misma.

```

<body>
  <header>
    <div id="headerLeft">
      
      Estudio <strong>Médico</strong>
    </div>
  </header>
  <div class="form-box">
    <h2>Bienvenido</h2>
    <h4><strong>Déficit de vitamina D y control glucémico en pacientes con diabetes mellitus tipo 2 en atención primaria </strong></h4>
    <hr>
    <form [formGroup]="loginForm" (ngSubmit)="onSubmit()">
      <div class="form-group">
        <label for="dni">DNI/NIE</label>
        <input type="text" [(ngModel)]="inputDni" placeholder="Introduce tu DNI/NIE" onfocus="placeholder = ''" onblur="this.placeholder = 'DNI/NIE'" formControlName="dni" class="form-control" required autofocus/>
      </div>
      <hr>
      <div class="form-group">
        <label for="password">Contraseña</label>
        <input type="password" [(ngModel)]="inputPassword" placeholder="Introduce tu contraseña" onfocus="placeholder = ''" onblur="this.placeholder = 'Contraseña'" formControlName="password" class="form-control" required/>
      </div>
      <hr>
      <button id="buttonLogin" class="btn btn-primary">ENTRAR</button>
    </form>

    <div id="modalAlert" class="alert alert-danger" [hidden]={!alertHidden}>
      <strong>Error! </strong> {{errorMessage}}
    </div>
  </div>

```

Figura 4.4: Código HTML del login.

Los formularios constan de un FormGroup con nombre propio (loginForm en este caso) que engloba uno o varios elementos de control con nombre propio (formControlName, en este caso dni y password) como campos de entrada de datos para el formulario y finalmente un botón para disparar la validación del formulario, cuyo resultado se representa como mensaje de error en el modal que hay justo a continuación o con la resolución del mismo, que en este caso permitiría acceder a la aplicación con el perfil introducido. Este proceso de validación y respuesta se produce en el archivo de TypeScript.

```

ngOnInit() {
  this.loginForm = this.formBuilder.group({
    dni: ['', Validators.required],
    password: ['', Validators.required]
  });
}

get f() { return this.loginForm.controls; }

onSubmit() {
  this.alertHidden = false;

  if(!this.passwordInputService.validateEmptyField(this.f.password.value) ||
    !this.dniInputService.validateEmptyField(this.f.dni.value)){
    this.errorMessage = "DNI/NIE y/o contraseña vacíos";
    this.alertHidden = true;
    this.inputDni = "";
    this.inputPassword = "";
  }
  else{
    if(!this.dniInputService.validateDNI(this.f.dni.value) && !this.dniInputService.validateNIE(this.f.dni.value)){
      this.errorMessage = "DNI/NIE formato incorrecto";
      this.alertHidden = true;
      this.inputDni = "";
      this.inputPassword = "";
    }
    else{
      this.alertHidden = false;
      this.userToLog.username = this.f.dni.value;
      this.userToLog.password = this.f.password.value;

      this.doLogin();
    }
  }
}

```

Figura 4.5: Código TypeScript del login.

En la figura 4.5 se observan dos bloques, uno es el del método ngOnInit, que es lanzado por el componente justo al montar la página y enlaza los campos del FormGroup explicado anteriormente con las variables que se utilizan en este archivo mediante sus nombres (dni y password). El segundo bloque recoge el método onSubmit que se lanza al presionar el botón del formulario y como se puede

apreciar primero realiza las validaciones, las cuales en caso de ser erróneas rellenan el modal con un mensaje de error, y finalmente llama al método doLogin que conecta al usuario a la aplicación si todas han sido correctas. Sin embargo estas validaciones son llamadas desde dos elementos externos a este documento, el dniInputService y el passwordInputService. Estos servicios son una colección de métodos que no se encuentran dentro de ningún componente y sirven tanto para ser utilizados de forma recurrente por cualquiera de estos como para conectar la aplicación *frontend* al *backend*.

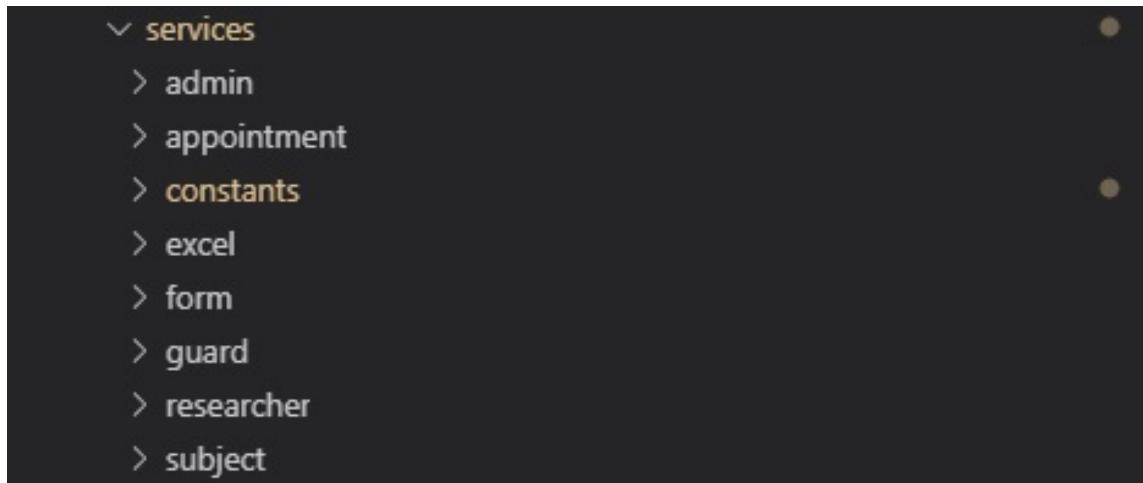


Figura 4.6: Servicios disponibles en la aplicación.

Al igual que los componentes, los servicios están englobados en cuatro grandes grupos: admin y researcher, los dos perfiles existentes en la aplicación y subject y form, los dos elementos principales a manejar en la misma. El resto de grupos de servicios corresponden a métodos puntuales que no encajaban en ninguno de estos grupos, de los cuales hablaremos según vayan apareciendo. En el caso del login sus servicios están incluidos en el grupo de researcher, ya que fue el primer perfil para el que se diseñó el mismo.

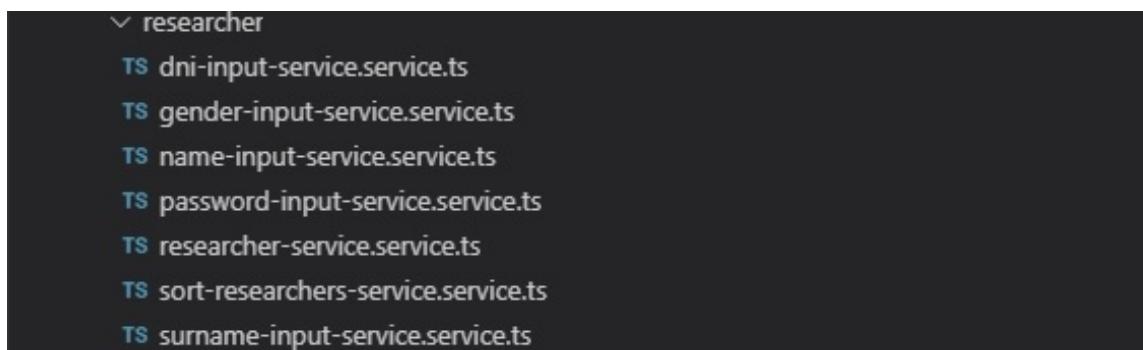


Figura 4.7: Servicios asociados al perfil de investigador.

Estos servicios engloban las validaciones de campos (los servicios con el sobrenombre input) tanto para hacer login en la aplicación como para registrar investigadores. El servicio researcher-service comprende todos los métodos de acceso al *backend* referentes a este perfil.

form

El componente de formulario se compone de dos partes: un componente appointment que engloba el test a realizar en los pacientes durante las citas y el appointment-view que es una visualización no editable del mismo una vez completado.

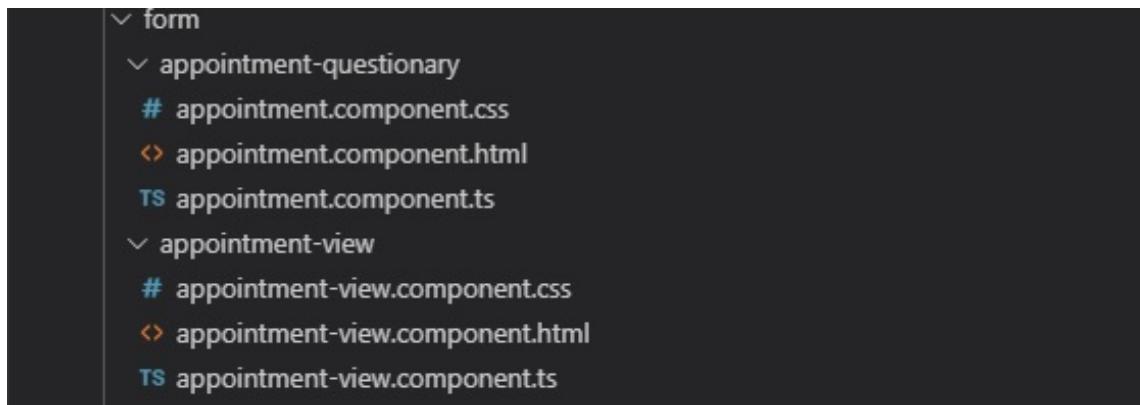


Figura 4.8: Componentes del test para las citas y su visualización una vez completados.

El primer componente corresponde al formulario de la figura 3.9, el cual tiene una estructura similar a la explicada para el de login solo que con una longitud de 600 líneas. Así mismo, tiene un componente de TypeScript en el que se manejan las validaciones, mensajes de error y el guardado del contenido en la base de datos. Se sirve de los servicios del form-service para realizar las validaciones de todos sus campos y de varios métodos del researcher-service para realizar los accesos a la base de datos a través del *backend*.

```
researcherUrl: string = `http://${this.constantsService.remoteHost}:8080/api/researcher`;

public registerAppointment(appointment: Appointment): Observable<any>{
  let userLogged: User = JSON.parse(localStorage.getItem("userLogged"));

  if(userLogged === null){
    return null;
  }

  return this.http.post(`${this.researcherUrl}/registerInvestigationDetails`, appointment, {headers: this.researcherHeaders});
}
```

Figura 4.9: Método para el guardado del test referente a una cita completada.

En la figura 4.9 se puede observar el método encargado de guardar la información del test en el sistema. Utiliza una url para saber donde encontrar el *backend* que le está ofreciendo los métodos, en la cual se utiliza una variable externa exportada desde constantService. Esta variable simplemente sirve como manera rápida de cambiar entre la url de acceso local, con la que los desarrolladores pueden trabajar en sus bases de datos privadas, y la url que conecta con el servidor operativo en Hostinger. Mediante la terminación de esta url, que se añade dentro de cada método, el *backend* es capaz de distinguir a qué controlador y qué método esté haciendo la llamada el *frontend*, el cual además le pasa los datos (en este caso el test relleno) como un parámetro junto a la llamada.

El segundo componente corresponde a la figura 3.20 y es solo la visualización de los datos de un test ya realizado, obtenidos en una sola llamada al cargar el componente y no comprende ningún otro comportamiento.

admin

El siguiente componente incluye todas las vistas del perfil de administrador. Comprende cuatro componentes, uno para cada elemento de la aplicación (investigadores, pacientes y test) y uno para la edición de perfiles de investigadores.

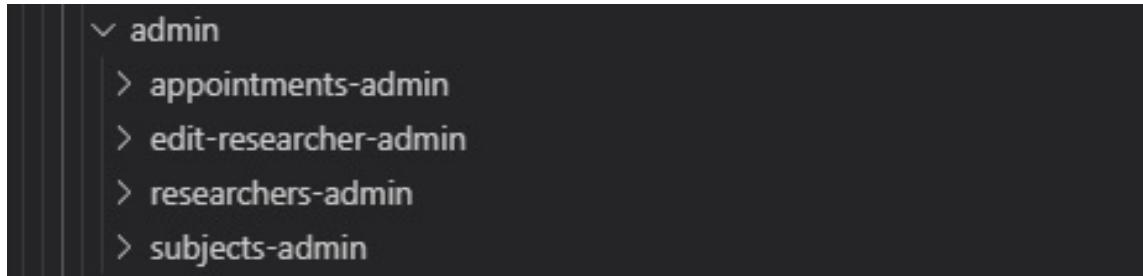


Figura 4.10: Componentes relacionados con el perfil de administrador.

El primer componente corresponde a la vista con el listado de test realizados en la aplicación por todos los investigadores y a la figura 3.24, que es la modificación de una de estas. Para el componente del listado se trae de la base de datos todos los test realizados con el investigador que los realizó y se rellena una tabla dinámicamente con dicha información.

```
<tbody>
  <tr *ngFor="let appointment of appointments; let i = index"
      [ngStyle]="{'background-color': (i % 2) !== 1 ? 'rgb(240, 240, 240)' : 'white'}">
    <td> {{appointment.subjectIdentificationNumber}}</td>
    <td *ngIf="appointment.numberInvestigation == 1; else secondAppointment">
      | Cita 1
      |</td>
      <ng-template #secondAppointment>
        <td>
          | Cita 2
          |</td>
      </ng-template>
    <td>
      <button id="buttonModify" type="button" (click)="goToModifyAppointment(appointment.investigationDetailsId,
        appointment.subjectIdentificationNumber)"></button>
    </td>
  </tr>
</tbody>
```

Figura 4.11: Bucle de generación para la tabla de test realizados.

La tabla emplea una variable en el archivo de TypeScript asociado en la que se han guardado los test llamada appointments (al principio del bucle for) y para cada iteración escribe el número de identificación del paciente, si es el test es de su cita 1 o 2 y un botón de editar que permite abrir la vista al segundo componente.

El siguiente componente de administrador es edit-researcher-admin, que es una sencilla vista con un formulario que permite editar nombre, apellidos y la contraseña de un investigador que se puede ver en la figura 3.16. A este componente se accede mediante una tabla, muy similar a la utilizada para los test, que se encuentra en el componente de researcher-admin. Este es a su vez la vista principal del administrador nada más identificarse en la aplicación y contiene dicha tabla de investigadores y un formulario que se puede observar en la figura 3.11 que permite el registro de investigadores. El último componente, referente a los pacientes, incluye una tabla en la línea de las anteriores y unos filtros para la misma como se puede ver en la figura 3.18.

researcher

El último grupo de componentes son los referentes al perfil de investigador. En este grupo se añadiría el componente de estadísticas comentado en capítulos anteriores.

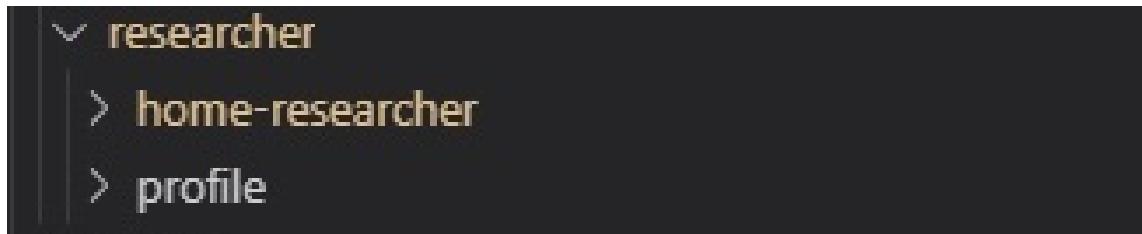


Figura 4.12: Componentes relacionados con el perfil de investigador.

Como se observa en la figura 4.12 existen dos componentes para este perfil. El primero implementa la vista principal del investigador, donde se ve una tabla con los pacientes que tiene asociados y un pequeño campo para introducir nuevos pacientes mediante su código de identificación como se puede ver en la figura 3.8. Incluye también un modal para mensajes de error y un formulario de inclusión. Este formulario mediante CSS aparece sobre el resto del contenido al añadir un paciente para solicitar que marques si se cumplen todos los criterios de admisión (mayor de 18 años, al menos una visita previa en el centro, etc). Aprovecharemos este componente para explicar como se manejan los errores y las llamadas de observables.

```
observable.subscribe(responseData =>{
  var subjectCreated: Subject = responseData;
  this.successMessage = "Paciente con Nº" + subjectCreated.identificationNumber + " registrado correctamente";
  this.setSuccessDeleteModal();
  this.ngOnInit();
}, error => {
  this.setAlertDeleteModal();
  if(error.status === 409){
    this.alertMessage = "Error, el paciente ya existe";
  }
  else if(error.status === 410){
    this.alertMessage = "Error, el investigador ya no existe";
  }
  else if(error.status === 500){
    this.alertMessage = "Error en el servidor";
  }
});
```

Figura 4.13: Llamada a un servicio mediante un observable y manejo de errores.

En la figura 4.13 vemos como un observable utiliza un método subscribe que dentro incluye el código del que hablaremos. Este observable es básicamente una llamada asíncrona a un método, en este caso registrar paciente, a la cual nos suscribimos para manejar su resultado cuando se obtenga. Dentro del método vemos un primer bloque de código para responseData, el cual es llamado en caso de devolver el método a parte de su contenido un código HTTP 200, que significa que todo ha ido sin problemas. En caso de devolver otro código se accede al segundo bloque en el que se activa el modal con el mensaje de error y se muestra una explicación del mismo más ilustrativa que solo un número. Estos códigos de error han sido decididos por los desarrolladores siguiendo la filosofía original de HTTP y por ese motivo se conoce exactamente cuales se pueden dar y su significado.

4.1.3. Inicialización del proyecto

Local

Para la inicialización del proyecto en local es necesario tener algún entorno de desarrollo compatible, como Visual Studio Code el cual hemos utilizado nosotros, y tener instalada la interfaz de línea de comandos de Angular (Angular CLI). Una vez instalada solo necesitamos abrir el proyecto en nuestro entorno, abrir un terminal y ejecutar `ng serve`. En ocasiones el instalador de paquetes npm tiene problemas para reconocer el comando (nos ha ocurrido en algunos equipos) y será necesario escribir una sentencia algo más completa: `npm run ng serve`.

Remoto

En el caso de querer acceder a la aplicación en remoto no se requiere de ningún requisito previo más que utilizar un navegador. La página desde la que se accedía a la aplicación es <http://tfg-estudio-medico.com/>, la cual se utilizó de manera temporal para hacer pruebas y actualmente no se encuentra disponible.

4.2. Backend

En esta sección se va a hablar del proyecto Java Spring Boot el cual compone la parte *backend* de nuestra aplicación. La aplicación es una API-REST, la cual será consumida por el *frontend* mediante llamadas en protocolo HTTP. Dicho proyecto se almacenó y gestionó mediante este repositorio de GitHub[32].

4.2.1. Generación Proyecto Java Spring Boot

Para la generación del proyecto inicial se utilizó Spring Initializr[33], a través de su página web. Escogimos gestionar las dependencias del proyecto con Maven dado el alto grado de familiaridad que poseemos con esta tecnología. A continuación se escogió el lenguaje Java para implementar la aplicación debido a sus características intrínsecas (polimorfismo y herencia). Para finalizar se añadieron los paquetes de Web y JPA, necesarios para gestionar las conexiones externas y para la capa de persistencia.

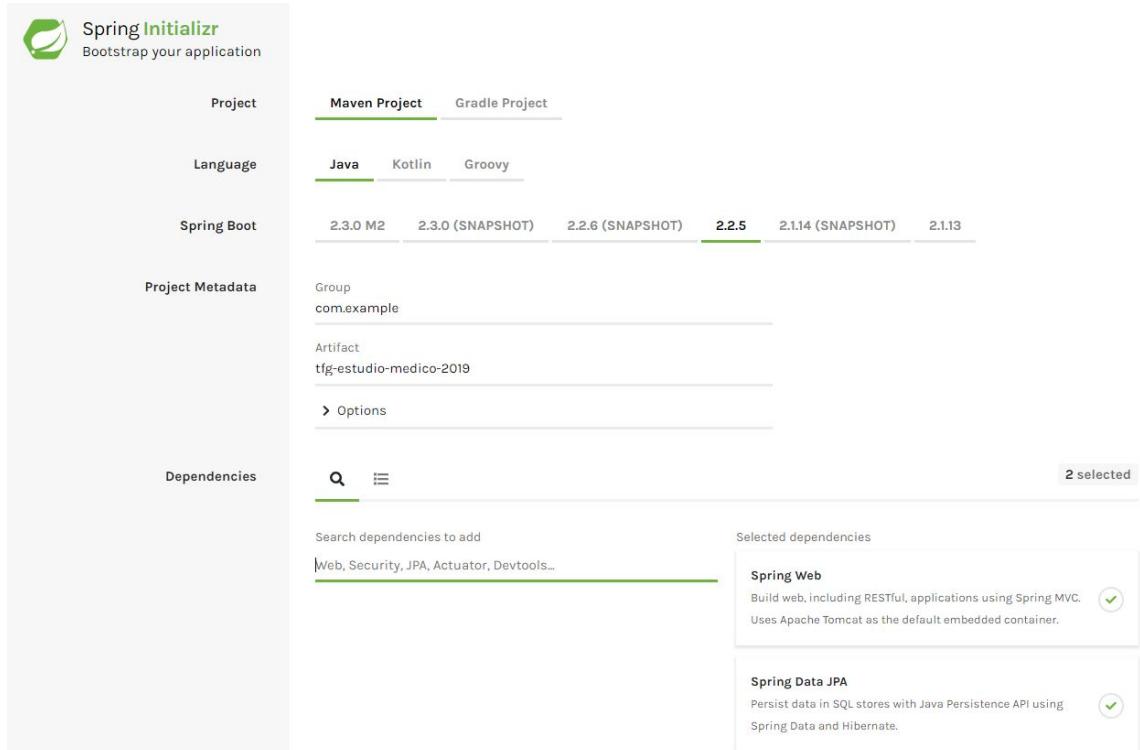


Figura 4.14: Proyecto inicial Spring Boot

Una vez generado el proyecto Spring Boot, hubo de importarse el mismo en el entorno de desarrollo Eclipse.

4.2.2. Arquitectura de la aplicación Java Spring Boot

La arquitectura de la aplicación Spring Boot desarrollada se divide principalmente en los apartados de código, documentación y tests.

Código

El proyecto *backend* de la aplicación está escrito en Java 8, y dividido en los siguientes bloques o paquetes, los cuales se describen a continuación.

■ Controladores

Son los elementos del código encargados de recibir las diferentes peticiones HTTP y mapear los objetos enviados en las mismas, ya sea en el cuerpo de la petición en caso de ser una petición de tipo POST, o en la propia url si es una petición de tipo GET. Además de esto, son los encargados de realizar la lógica de navegación, llamando a la capa de Negocio y, con el resultado obtenido, generar una respuesta a la parte *frontend* de la aplicación.

Para identificar los diferentes controladores de nuestra aplicación, la nomenclatura de los mismos siempre terminará con la palabra *Controller*.

En nuestra aplicación disponemos de 3 controladores: El *UserController*, encargado principalmente del proceso de login de la aplicación. El *AdminController*, encargado de obtener todos los recursos que demande el usuario de tipo administrador en nuestra aplicación. Finalmente el *ResearcherController*, encargado de obtener los recursos que demanden los usuarios de tipo investigador en nuestra aplicación.

Todos los controladores se han implementado con el patrón de Diseño *Interface*, separando los métodos de su implementación y haciendo al código más legible y mantenable de cara al futuro.

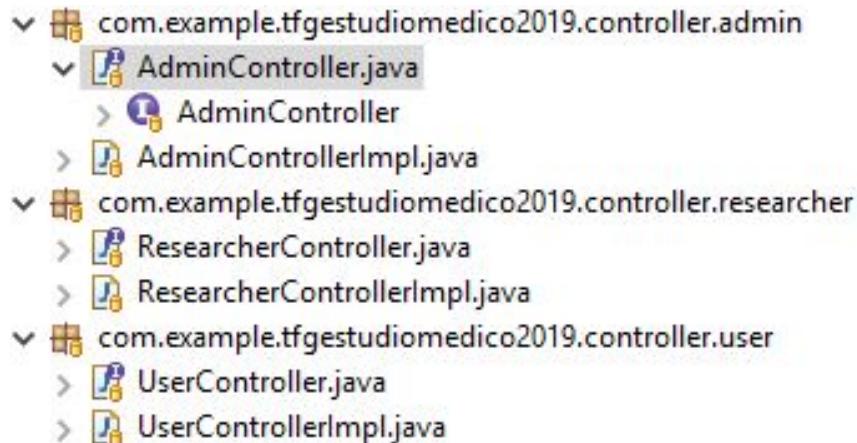


Figura 4.15: Controladores del proyecto *backend*

```

15
16 /**
17  * Controller for All API Users.
18 */
19 @RequestMapping("/api/user")
20 public interface UserController {
21
22     @ApiOperation(value = "Login user")
23     @ApiResponses(value = {
24         @ApiResponse(code = 200, message = "Successfully logued"),
25         @ApiResponse(code = 409, message = "Fail login"),
26         @ApiResponse(code = 409, message = "Internal server error")
27     })
28     @PostMapping(path = "/login", produces = "application/json")
29     public ResponseEntity<?> login(@RequestBody UserToLoginDto userToLoginDto);
30 }
31
  
```

Figura 4.16: Ejemplo Interfaz Controlador UserController

■ Servicios de la aplicación

Son los elementos del código encargados de centralizar la lógica principal de la aplicación. Los controladores los llaman, estos llaman a los repositorios para obtener los recursos solicitados, y devuelven dichos recursos al controlador.

Para identificar los diferentes servicios de nuestra aplicación, la nomenclatura de los mismos siempre terminará con la palabra *Business*.

En nuestra aplicación disponemos de 3 servicios de aplicación: El *UserBusiness*, encargado principalmente de gestionar a los diferentes usuarios de la aplicación. El *ResearcherBusiness*, encargado de gestionar las diferentes funcionalidades que puede realizar un usuario de tipo investigador en nuestra aplicación. Por último el *SubjectBusiness*, encargado de gestionar a los pacientes y a sus respectivas citas y cuestionarios.

Al igual que los controladores, todos los servicios de aplicación se han implementado con el patrón de diseño *Interface*

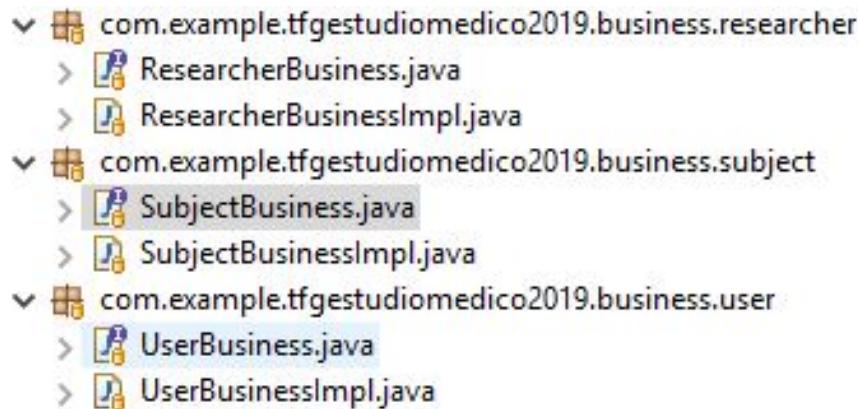


Figura 4.17: Servicios de Aplicación del proyecto *backend*

```
6
7 /**
8 * User business.
9 */
10 public interface UserBusiness {
11     public UserEntity findByUsernameAndPassword(String username, String password);
12     public UserEntity saveUser(UserEntity user);
13     public UserEntity findByUsername(String username);
14     public UserEntity findById(Integer id);
15     public List<UserEntity> getAllResearchers();
16     public List<UserEntity> getAllAdmins();
17     public boolean deleteResearcher(String username);
18     public UserEntity updateUser(UserEntity user);
19 }
20
```

Figura 4.18: Ejemplo Interfaz Servicio de aplicación UserBusiness

■ **Repositorios**

Son los elementos encargados de comunicarse con la base de datos correspondiente, ya sea guardando recursos, actualizar, listar, borrar etc...

Todos los repositorios extienden de la clase *JpaRepository*, una clase perteneciente a Spring-Data, la cual implementa el código necesario para realizar una llamada a la base de datos. De esta manera no ha sido necesario implementar dichas clases.

Para identificar los diferentes repositorios de nuestra aplicación, la nomenclatura de los mismos siempre terminará con la palabra *Repository*.

Nuestra aplicación dispone de 4 repositorios: El *InvestigationDetailsRepository*, encargado de gestionar los detalles de los cuestionarios realizados a los pacientes. El *InvestigationRepository*, encargado de gestionar los cuestionarios realizados a los pacientes. El *UserRepository*, encargado de gestionar a los diferentes usuarios administradores e investigadores de la aplicación. Y finalmente, el *SubjectRepository*, encargado de gestionar a los pacientes del estudio.

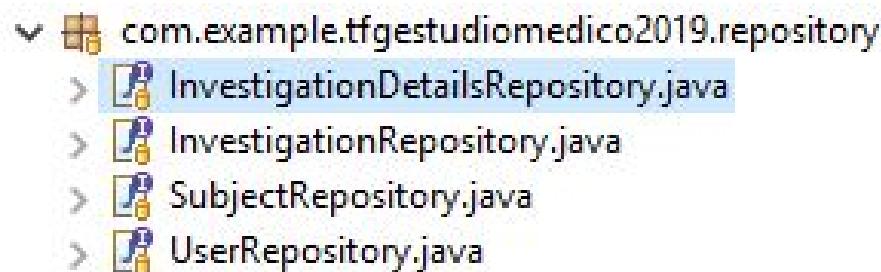


Figura 4.19: Repositorios del proyecto *backend*

```

10 /**
11  * Interface to generate User queries.
12 */
13 @Repository
14 public interface UserRepository extends JpaRepository<UserEntity, Long> {
15
16     UserEntity findByUsernameAndPassword(String username, String password);
17     UserEntity findByUsername(String username);
18     UserEntity findById(Integer id);
19     List<UserEntity> findByRole(String role);
20     Long deleteByUsername(String username);
21 }
```

Figura 4.20: Interfaz Repositorio UserRepository

■ Modelos

Son los elementos encargados de almacenar la información de las bases de datos y de las peticiones HTTP, delimitando el dominio de las mismas.

Hay 3 tipos de modelos en nuestra aplicación:

- Dominio.

Son los elementos que definen un rango de posibles valores. En nuestra aplicación disponemos del enumerado *Role* para definir los tipos de usuarios que pueden acceder a la aplicación.

```

1 /**
2  * Enum to define the API roles.
3  */
4 public enum Role {
5     ADMIN, RESEARCHER
6 }
```

Figura 4.21: Ejemplo Dominio Enumerado Role

- Entidades.

Son los elementos encargados de interactuar con las bases de datos de nuestra aplicación. Estas entidades JPA, también llamadas POJO's (Plain Old Java Objects), se encuentran mapeadas mediante anotaciones y representan las tablas de nuestra base de datos relacional.

Para identificar las diferentes entidades de nuestra aplicación, la nomenclatura de las mismas siempre terminará con la palabra *Entity*.

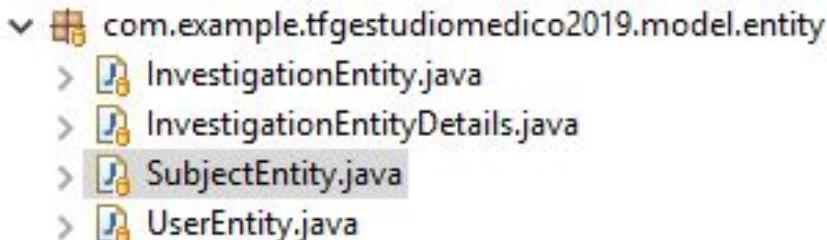


Figura 4.22: Entidades del proyecto *backend*

```
15 *  
16  * Entity for investigation table.  
17 */  
18 @Entity  
19 @Table(name = "investigation")  
20 public class InvestigationEntity {  
21  
22     @Id  
23     @GeneratedValue(strategy = GenerationType.AUTO)  
24     private Integer id;  
25  
26     @ManyToOne  
27     @JoinColumn(name = "idsubject")  
28     private SubjectEntity subject;  
29  
30     @Column(name = "numberinvestigation")  
31     private Integer numberInvestigation;  
32  
33     @Column(name = "completed")  
34     private Boolean completed;  
35  
36     @OneToOne(cascade = {CascadeType.ALL})  
37     @JoinColumn(name="idinvestigationdetails")  
38     private InvestigationEntityDetails investigationEntityDetails;  
39
```

Figura 4.23: Ejemplo Entidad InvestigationEntity

- Dtos (Data Transfer Objects).

Son los elementos encargados de mapear los objetos de tipo JSON enviados en las peticiones HTTP para ser tratados en la aplicación. También se encargan de devolver la información necesaria mediante una respuesta HTTP.

Para identificar a los dtos (Data Transfer Objects) de nuestra aplicación, la nomenclatura de los mismos siempre terminará con la palabra *Dto*.

De esta manera, los controladores de la aplicación tratarán los dtos y los mapearán a entidades para ser tratadas en la capa de Negocio, proporcionando seguridad y robustez a la aplicación, evitando posibles inyecciones de código y protegiendo los campos sensibles que no deban mostrarse en el exterior.

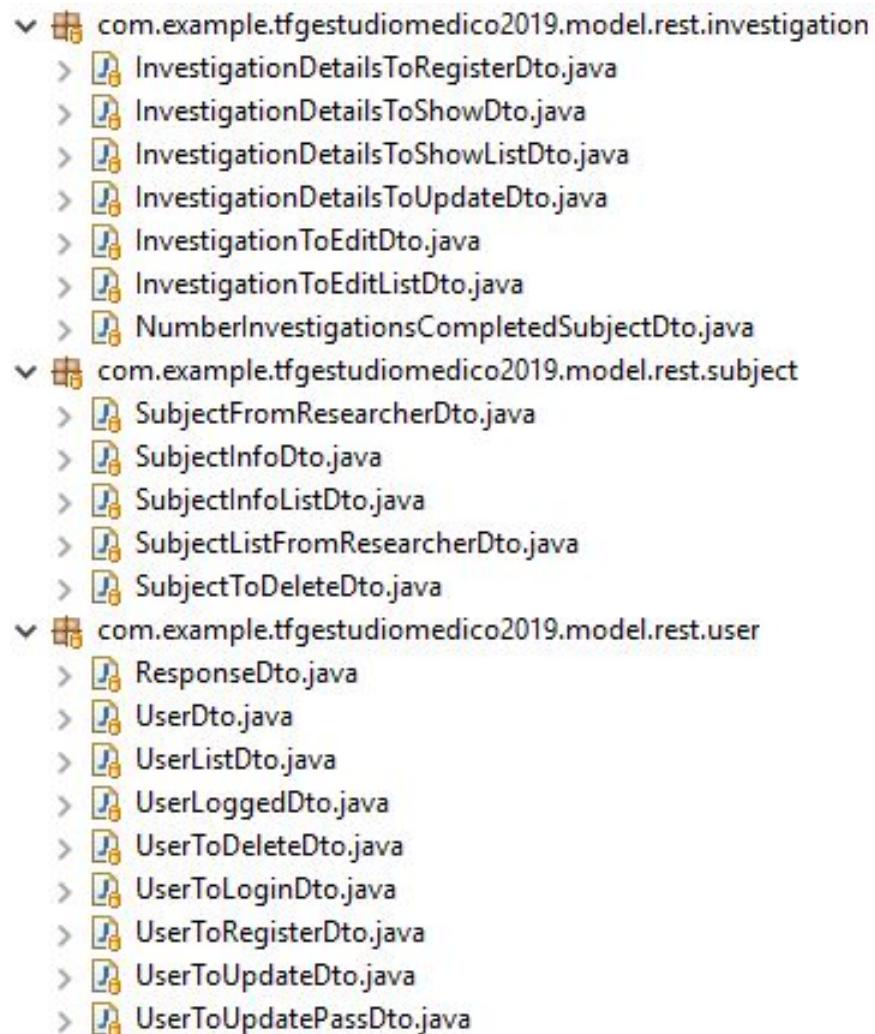


Figura 4.24: Dtos del proyecto *backend*

```

5 /**
6 * User dto general.
7 *
8 */
9 public class UserDto {
10
11     @ApiModelProperty(value = "The username of the user", example = "12345678A", dataType = "java.lang.String")
12     private String username;
13     @ApiModelProperty(value = "The name of the user", example = "Sergio", dataType = "java.lang.String")
14     private String name;
15     @ApiModelProperty(value = "The gender of the user", example = "hombre", allowableValues="hombre, mujer",
16     dataType = "java.lang.String")
17     private String gender;
18     @ApiModelProperty(value = "The id of the user", example = "23", dataType = "java.lang.Integer")
19     private Integer id;
20     @ApiModelProperty(value = "The surname of the user", example = "Pacheco Fernández", dataType = "java.lang.String")
21     private String surname;

```

Figura 4.25: Ejemplo dto UserDto

Documentación

A la hora de documentar la aplicación *backend*, por una parte, se utilizó *Javadoc* al principio de cada clase Java, y por otro lado, se utilizó *Swagger*.

Para utilizar el *framework Swagger* en la aplicación, hubo de añadir las correspondientes dependencias en el archivo *pom.xml*.

```

53
54     <!-- SWAGGER -->
55     <dependency>
56         <groupId>io.springfox</groupId>
57         <artifactId>springfox-swagger2</artifactId>
58         <version>2.9.2</version>
59     </dependency>
60
61     <dependency>
62         <groupId>io.springfox</groupId>
63         <artifactId>springfox-swagger-ui</artifactId>
64         <version>2.9.2</version>
65     </dependency>
66

```

Figura 4.26: Dependencias Swagger añadidas en pom.xml

Swagger nos permitió documentar tanto los puntos de acceso como los dtos de nuestra aplicación a través de anotaciones, las cuales se añadieron en las respectivas clases implicadas.

```

54     @ApiOperation(value = "Register a researcher")
55     @ApiResponses(value = {
56         @ApiResponse(code = 201, message = "Successfully user registered", response= UserDto.class),
57         @ApiResponse(code = 409, message = "User already exists", response= ResponseDto.class),
58         @ApiResponse(code = 500, message = "Server error", response= ResponseDto.class)
59     })
60     @PostMapping(path = "/registerResearcher", produces = "application/json")
61     public ResponseEntity<?> registerResearcher(@RequestBody UserToRegisterDto user);

```

Figura 4.27: Ejemplo Documentación *Swagger* en el endpoint Registrar Investigador

Swagger posee una interfaz llamada *swagger.ui* a la cual se puede acceder una vez arrancado el proyecto *backend* para realizar pruebas y comprobar la documentación generada, como se muestra

a continuación en la figura 4.28. Dicha interfaz documenta tanto la información general de la aplicación, como los puntos de acceso y los dtos.

The screenshot shows the Swagger UI interface for the 'API REST TFG Estudio Médico 2019'. At the top, there's a green header bar with the 'swagger' logo and a dropdown menu labeled 'Select a spec' with 'default' selected. Below the header, the title 'API REST TFG Estudio Médico 2019 1.0.0' is centered, with a small '1.0.0' badge next to it. Underneath the title, there are two lines of text: '[Base URL: localhost:8080 /]' and 'http://localhost:8080/v2/api-docs'. Below this, a note says 'TFG developed by Eduardo Gonzalo and Sergio Pacheco'. The main content area is organized into sections: 'admin-controller-impl' (Admin Controller Impl), 'researcher-controller-impl' (Researcher Controller Impl), 'user-controller-impl' (User Controller Impl), and 'Models'. Each section has a right-pointing arrow icon at its end. The 'researcher-controller-impl' section is expanded, showing a list of API endpoints:

Figura 4.28: Interfaz swagger.ui

The screenshot shows the detailed API documentation for the 'researcher-controller-impl' section. The title 'researcher-controller-impl Researcher Controller impl' is at the top. Below it, a list of API endpoints is shown in a table-like format:

Method	Endpoint	Description
GET	/api/researcher/{id}/subjects	Get all subjects and investigations who belong to a researcher
GET	/api/researcher/getAllInvestigationDetails/{idSubject}	Get a list of investigation details based on the idSubject
GET	/api/researcher/getInvestigationDetails/{idSubject}/{numberInvestigation}	Get the investigation details based on the idSubject and numberInvestigation
GET	/api/researcher/investigationsCompletedSubjectResearcher/{identificationNumber}	Get the number of investigations completed from a subject based on his/her identification number
POST	/api/researcher/registerInvestigationDetails	Register an investigation details
POST	/api/researcher/registerSubject	Register a subject
POST	/api/researcher/updatePassword	Update the password of a user

Figura 4.29: Ejemplo Documentación Swagger Researcher Controller

```

UserDto <-
  {
    gender      string
    example: hombre
    The gender of the user
    Enum:
      ▼ [ hombre, mujer ]
    id          integer($int32)
    example: 23
    The id of the user
    name        string
    example: Sergio
    The name of the user
    surname     string
    example: Pacheco Fernández
    The surname of the user
    username    string
    example: 12345678A
    The username of the user
  }

```

Figura 4.30: Ejemplo Documentación Swagger UserDto

Tests

A la hora de garantizar la calidad del *software*, se han realizado tests con Mockito y JUnit. Se han testeado todos los métodos de todas las clase que poseen lógica (Controladores y Servicios de Aplicación), probando todos los caminos y excepciones posibles, así como el camino feliz.

En la aplicación hay un total de 157 tests realizados, los cuales se ejecutan cada vez que se instala la aplicación, garantizando así el correcto funcionamiento de la misma.

```

[INFO] -----
[INFO] T E S T S
[INFO] -----
[INFO] Running business.researcher.ResearcherBusinessTest
[INFO] Tests run: 11, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.633 s - in business.researcher.ResearcherBusinessTest
[INFO] Running business.subject.SubjectBusinessTest
[INFO] Tests run: 24, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.063 s - in business.subject.SubjectBusinessTest
[INFO] Running business.user.UserBusinessTest
[INFO] Tests run: 17, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.01 s - in business.user.UserBusinessTest
[INFO] Running controller.admin.AdminControllerTest
[INFO] Tests run: 62, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.273 s - in controller.admin.AdminControllerTest
[INFO] Running controller.researcher.ResearcherControllerTest
[INFO] Tests run: 49, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.161 s - in controller.researcher.ResearcherControllerTest
[INFO] Running controller.user.UserControllerTest
[INFO] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0 s - in controller.user.UserControllerTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 157, Failures: 0, Errors: 0, Skipped: 0

```

Figura 4.31: Tests realizados en la aplicación

Para garantizar que se han testeado todos los caminos posibles, se ha conseguido un 100% de cobertura en todas las clases con tests.

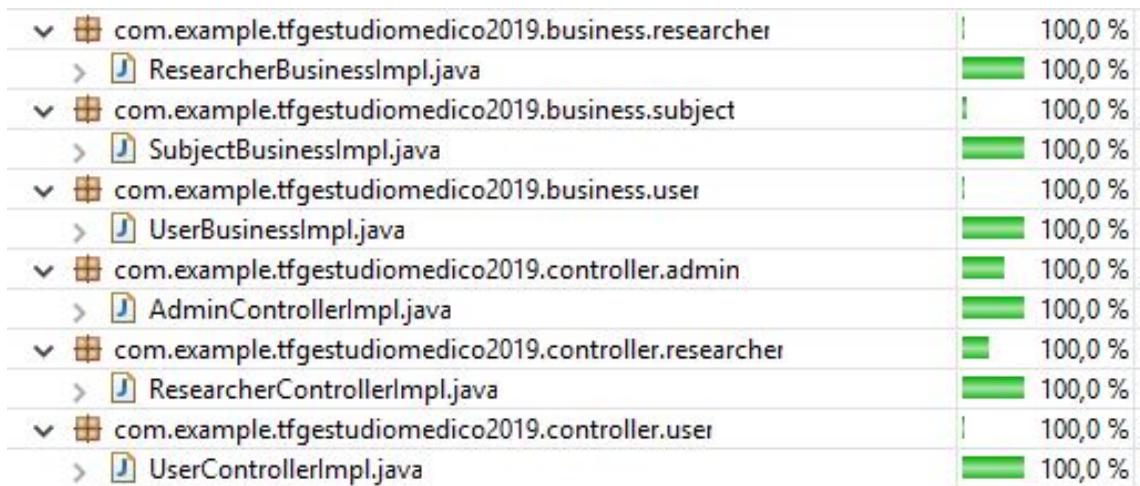


Figura 4.32: Cobertura de las clases testeadas

4.2.3. Inicialización del proyecto

A la hora de inicializar la aplicación *backend*, al tratarse de un proyecto Maven, lo primero que hay que ejecutar son los siguientes comandos:

1. `mvn clean`.
Elimina los archivos generados anteriormente y descarga las librerías añadidas como dependencias en el archivo pom.xml
2. `mvn install`.
Genera el archivo .jar definido en el pom.xml

Si queremos ejecutar la aplicación *backend* de manera local, lo único que tenemos que hacer es ejecutar la clase `Main` de la aplicación en el entorno de desarrollo elegido, en este caso, Eclipse.

Si queremos ejecutar la aplicación *backend* de manera remota, solo tenemos que almacenar el archivo .jar generado anteriormente en el servicio de Hosting escogido, en este caso, en Hostinger[14].

CAPÍTULO 5

Despliegue

En este capítulo se detalla cómo, una vez terminada una versión funcional de la aplicación, se comienza con el proceso de habilitar el uso de la aplicación desde la web. Para ello decidimos utilizar la plataforma Hostinger[14], y todos los servicios que la misma proporciona a sus usuarios.

Elegimos Hostinger y no otras alternativas como Microsoft Azure[34] debido a ciertas características que lo hacían diferente de los demás. Por una parte Hostinger posee un sistema de configuración fácil, tanto para el lado *backend* como el *frontend* y para la gestión de las bases de datos relacionales mediante phpMyAdmin. Otra razón de peso es que este servicio de *hosting* posee un chat en español disponible 24 horas al día durante todo el año, de esta manera ante cualquier imprevisto contamos con la ayuda de los mismos.

El despliegue de la aplicación consta de tres partes; el despliegue de la aplicación *frontend*, el despliegue de la aplicación *backend*, y el despliegue de la base de datos.

5.1. Frontend

Para desplegar la aplicación Angular que representa el *frontend* de nuestra aplicación, se utilizó la herramienta “Administrador de archivos” que nos ofrece Hostinger y que nos proporciona una interfaz de usuario para administrar archivos y directorios en nuestro dominio *tfg-estudio-medico.com*.

En dicho administrador, se procedió a subir la carpeta *dist* generada por el proyecto Angular.

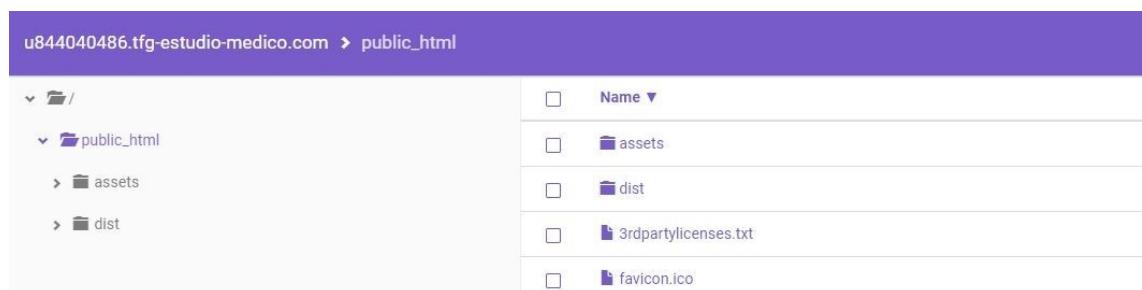


Figura 5.1: Administrador de archivos de Hostinger

5.2. Backend

Para desplegar la aplicación Java Spring que constituye el *backend* de nuestra aplicación, se utilizó un servidor proporcionado por Hostinger[14]. En dicho servidor instalamos un sistema operativo Ubuntu 18.04 64bit.

The screenshot shows the Hostinger VPS management interface for the server VPS48044501.LOCAL. The main section displays the 'Estatus del servidor' (Server Status) which is 'Corriendo' (Running) indicated by a green triangle icon. Below this, there are two buttons: 'Detener' (Stop) and 'Reiniciar' (Restart). A table provides detailed server specifications: Núcleos de CPU (2), Velocidad Total de CPU (2.4Ghz), Memoria (1Gb), Espacio en Disco (20Gb), and Ancho de Banda (1000Gb). It also shows the plan as 'Plan 1', an expiration date of '2020-04-05', and two buttons: 'Mejorar Cuenta' (Upgrade Account) and 'Renovar' (Renew). Under 'Otras características' (Other Features), the 'Nombre del Host' is set to 'vps48044501.local' with a 'Guardar cambios' (Save changes) button. The 'Sistema operativo' is set to 'Ubuntu 18.04 64bit' with a 'Guardar cambios' button. A note below states: 'IMPORTANTE : El cambio del sistema operativo destruirá todos los datos existentes en el disco duro.' (IMPORTANT : Changing the operating system will destroy all existing data on the hard drive.). There is also a 'Habilitar la cuota de disco' (Enable disk quota) switch set to 'OFF' and a 'Firewall' button with a 'Reiniciar' (Restart) sub-button.

Figura 5.2: Servidor en Hostinger

Para conectarnos al servidor utilizamos el programa MobaXterm, conectándonos al servidor a través de SSH. Nuestro objetivo era mantener el archivo .jar ejecutándose en la máquina incluso si no estamos conectados a la misma, para ello generamos un *servicio*, esto es, un script que se mantiene arrancado incluso si cerramos la sesión.

Lo primero que tenemos que hacer es generar el archivo ejecutable de nuestro proyecto *backend*.

Una vez generado el archivo SNAPSHOT.jar de nuestro proyecto Java Spring, nos disponemos a subirlo a la carpeta /root/back.

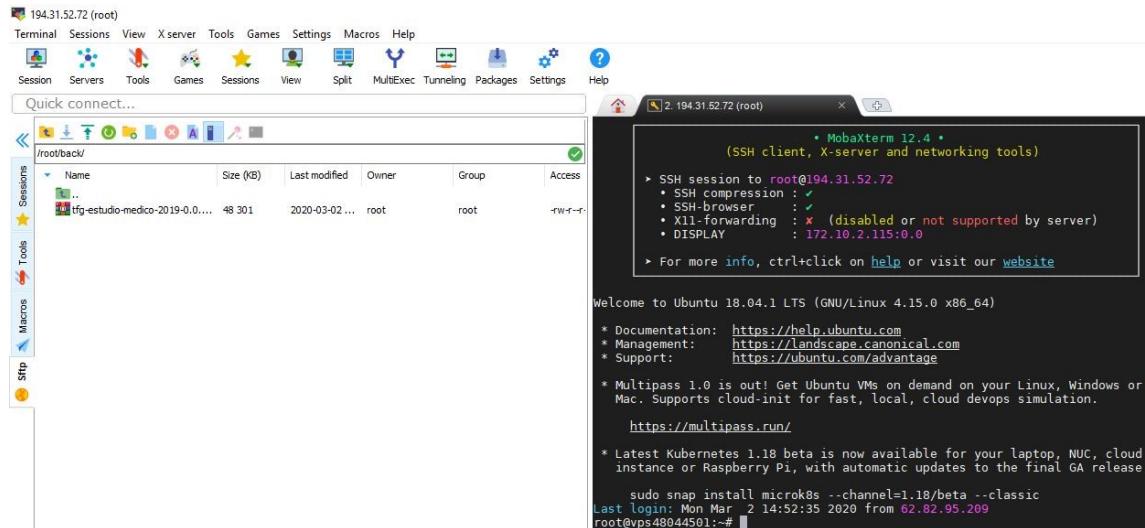


Figura 5.3: Jar desplegado en MobaXterm

A continuación, generamos un script llamado *start_back.sh* que simplemente ejecuta el comando *java -jar* para arrancar el ejecutable .jar, el cual vamos a guardar en el directorio /usr/local/bin. Lo guardamos en dicho directorio debido a que este directorio es accesible por todos los usuarios.

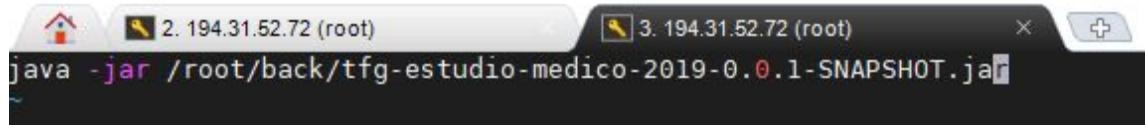


Figura 5.4: Script start_back.sh desplegado en MobaXterm

Después hemos generado un *servicio* llamado *daemonback.service* en la carpeta /etc/systemd/system, que es el directorio indicado en sistemas Linux para almacenar y gestionar demonios. Este servicio se encarga de ejecutar en segundo plano el script *start_back.sh*.

```

[Unit]
Description = Init BACK service.

[Service]
Type=simple

ExecStart=/bin/bash /usr/local/bin/start_back.sh

[Install]
WantedBy=multi-user.target
~
```

Figura 5.5: Servicio daemonback.service desplegado en MobaXterm

Por último, tan solo tenemos que ejecutar el servicio generado previamente con el comando **sudo service daemonback start**. Para ver el estado del .jar, ejecutamos el comando **sudo service daemonback status** y nos muestra lo siguiente:

```

● daemonback.service - Init BACK service.
   Loaded: loaded (/etc/systemd/system/daemonback.service; enabled; vendor preset: enabled)
   Active: active (running) since Tue 2020-03-03 09:52:31 UTC; 1min 14s ago
     Main PID: 23864 (bash)
        Tasks: 31 (limit: 4194)
       CGroup: /system.slice/daemonback.service
               └─23864 /bin/bash /usr/local/bin/start_back.sh

Mar 03 09:52:27 vps48044501.local bash[23864]: 2020-03-03 09:52:27.877 INFO 23865 ... [main] o.h.HibernateUtil: Annotati... : HCANN000001: Hibernate Commons Annotations (5.0.4.Final)
Mar 03 09:52:28 vps48044501.local bash[23864]: 2020-03-03 09:52:28.037 INFO 23865 ... [main] o.h.HibernateUtil: Annotati... : HCANN000002: Hibernate Commons Annotations (5.0.4.Final)
Mar 03 09:52:33 vps48044501.local bash[23864]: 2020-03-03 09:52:33.015 INFO 23865 ... [main] o.h.EntityManagerFactoryBean: Initialized JPA EntityManagerFactory for persistence unit 'default'
Mar 03 09:52:35 vps48044501.local bash[23864]: 2020-03-03 09:52:35.015 WARN 23865 ... [main] wbc:ConfigurationJpaWebMvCConfiguration: spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed even if no @View annotation is present on methods annotated with @Query or @PostProcessResultMapping
Mar 03 09:52:37 vps48044501.local bash[23864]: 2020-03-03 09:52:37.106 INFO 23865 ... [main] org.springframework.context.support.ClassPathXmlApplicationContext: Context refreshed
Mar 03 09:52:37 vps48044501.local bash[23864]: 2020-03-03 09:52:37.290 INFO 23865 ... [main] d.s.w.p.DocumentationPluginsBootstrapper: Found 1 custom documentation plugin(s)
Mar 03 09:52:38 vps48044501.local bash[23864]: 2020-03-03 09:52:38.089 INFO 23865 ... [main] o.s.b.w.embedded.tomcat.TomcatWebServer: Tomcat started on port(s): 8080 (http) with context path ''
Mar 03 09:52:38 vps48044501.local bash[23864]: 2020-03-03 09:52:38.097 INFO 23865 ... [main] c.e.t.TfgEstudioMedico2019Application: Started TfgEstudioMedico2019Application in 25.691 seconds (JVM running for 27.005)
```

Figura 5.6: Estado servicio

5.3. Base de Datos

Para desplegar la base de datos en la web, se utilizó un servicio que ofrece Hostinger para desplegar bases de datos MySQL.

A través de PhpMyAdmin y con la opción de generar la base de datos automáticamente al ejecutar un proyecto Java Spring con JPA, se generó la siguiente base de datos:

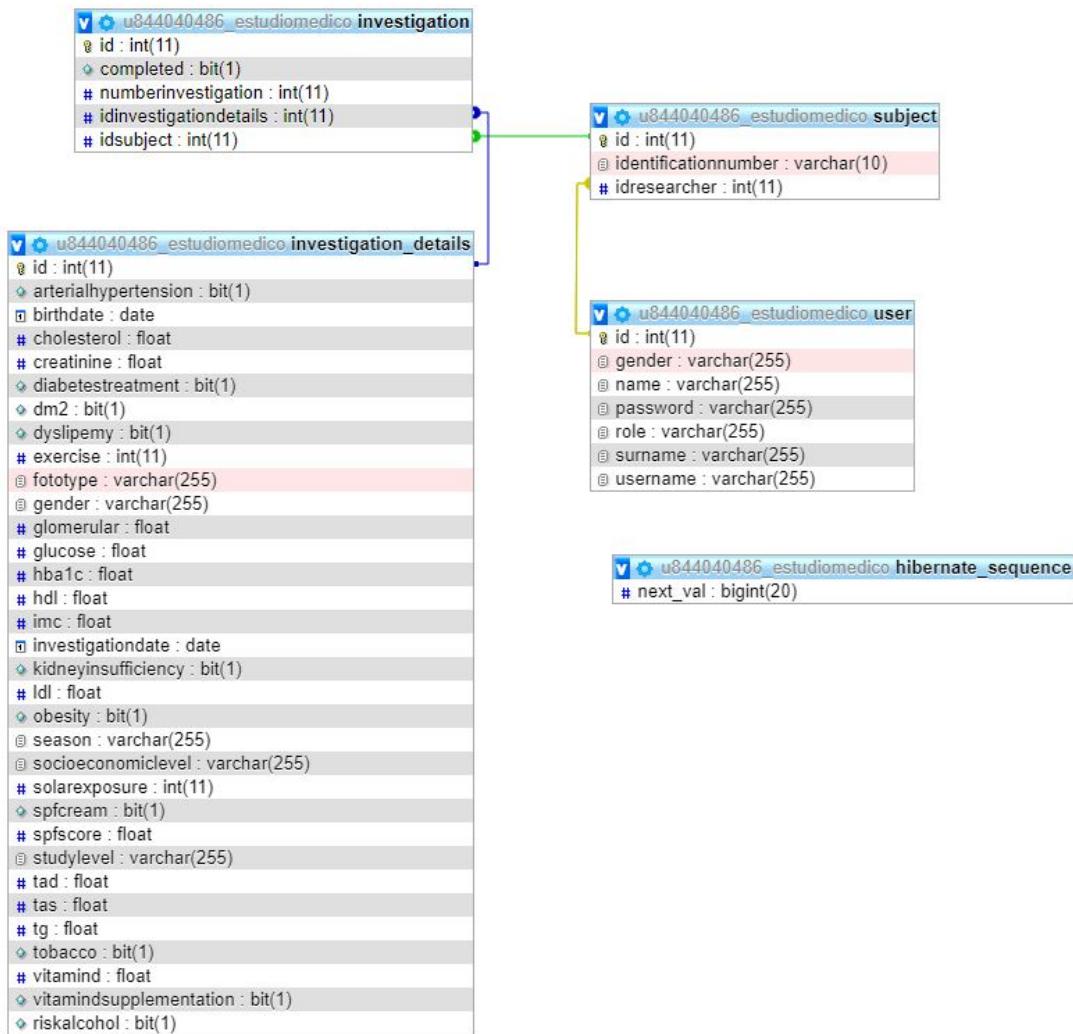


Figura 5.7: Modelo de datos generado en PhpMyAdmin

CAPÍTULO 6

Mantenimiento

En este capítulo se van a desarrollar todas las tareas de mantenimiento realizadas, ya que, una vez desplegada la aplicación, hay que dar soporte a la misma, acercándose de esta manera a como sería un proyecto real.

El mantenimiento es una parte vital del TFG y uno de los motivos por los cuales nos decantamos por escoger el mismo. Al hablar de mantenimiento nos referimos tanto a nuevas funcionalidades previamente no establecidas, corrección de errores y bugs, actualizaciones de funcionalidades ya implementadas, mejoras, etc.

Toda aplicación web necesita de tareas de mantenimiento, ya que el mercado y sus necesidades evolucionan de una manera vertiginosa y sino las aplicaciones se quedarían desfasadas.

Con el objetivo de dedicarle el tiempo suficiente y necesario a las tareas de mantenimiento, la aplicación se desplegó lo más pronto posible, a finales de febrero. Una vez desplegada la aplicación, el médico Alejandro Rabanal y su equipo de investigadores nos dieron el *feedback* necesario para empezar las tareas de mantenimiento.

A continuación se describe una lista de las tareas de mantenimiento solicitadas por el equipo de investigadores, ordenada por orden cronológico.

- **Registro de nuevos administradores.**

La primera tarea a realizar una vez desplegada la aplicación fue dar de alta a usuarios con perfil Administrador, teniendo estos permiso para gestionar la mayoría de elementos de la aplicación.

Se procedió a dar de alta como administradores, con sus respectivos documentos de identificación, a Pablo Rabanal, Alejandro Rabanal, Sergio Pacheco, Eduardo Gonzalo y a otros dos investigadores del estudio más.

De esta manera dichos usuarios podían acceder a la aplicación con su perfil y empezar a probar las funcionalidades de la misma, dando un *feedback* temprano y preciso.

- **Imposibilidad de registrar investigadores extranjeros.**

Cuando un administrador quiere registrar a un nuevo investigador, a la hora de llenar el formulario e introducir el campo DNI, este no admitía NIEs, de manera que es imposible registrar también a investigadores extranjeros.

Este bug se solucionó añadiendo una nueva validación al campo DNI/NIE para que admitiese ambos formatos, pudiendo registrar a investigadores extranjeros.

- **El administrador no puede exportar los datos de todos los pacientes.**

Se requiere de una nueva funcionalidad que consiste en que el administrador pueda generar un documento de tipo Excel que contenga los datos de todos los pacientes de todos los investigadores del estudio.

Esta funcionalidad se implementó sin ningún problema destacable, añadiendo un botón el cual generaba el documento Excel descargándolo a través del navegador en el dispositivo del usuario.

- **Falta de información en los mensajes de error de formato.**

A la hora de registrar un investigador, los administradores de nuestro proyecto nos solicitaron una mejora. Al llenar el campo DNI/NIE, si el valor introducido no es correcto y el formato es erróneo, se debe de añadir al mensaje de error mostrado información del formato válido.

La mejora solicitada se implementó correctamente, añadiendo al mensaje de error de formato del campo DNI/NIE información del formato de DNI y del NIE admitidos.

- **Falta de información en los campos de la cita.**

Se nos hizo saber que faltaba información relativa al formato correcto en la gran mayoría de los campos del formulario de llenar cita, tanto indicando el formato del campo como del rango de valores admitidos.

La mejora solicitada se implementó de manera exitosa a través de *tooltips* o pequeñas etiquetas en los campos, ayudando al usuario a saber como llenar los campos.

- **Cambio de formato en los campos de la cita.**

Tras un tiempo de pruebas, los investigadores nos solicitaron una nueva mejora. Tanto el campo de la cita *Tiempo de exposición solar*, como el campo *Ejercicio Físico*, en un principio, admitían números con decimales. La mejora consistía en que estos dos campos no admitiesen decimales, ya que carecía de sentido que alguien tomase el sol, por ejemplo, 50.3 minutos al día, además de que esto provocaba posibles errores a la hora de registrar una cita.

La mejora se implementó con éxito, cambiando el formato de ambos campos tanto en base de datos, como en *backend* y *frontend*, además de cambiar las validaciones para ajustarse a dicha mejora.

- **Necesidad de nuevas validaciones en los campos de la cita.**

Dado que los campos *Puntuación SPF* y *Uso de crema SPF* están relacionados, se nos solicitó añadir nuevas validaciones a la hora de llenar dichos campos para completar la cita.

La mejora consiste en que si el usuario marca como falso el campo *Uso de crema SPF*, el valor del campo *Puntuación SPF* debe ser cero para que tenga sentido. De la misma manera, si el usuario marca como verdadero el campo *Uso de crema SPF*, el valor del campo *Puntuación SPF* debe ser mayor que cero.

- **Bug encontrado a la hora de generar documento Excel.**

Los investigadores se dieron cuenta de que, al generar un documento Excel con los datos de las citas de los pacientes, por cada cita se generaba una fila. Esto no era el comportamiento esperado, ya que se especificó que se generase una única fila por paciente conteniendo información de todas sus citas.

Este *bug* necesitó de bastante trabajo, ya que no resultó fácil el tratamiento de los datos y la generación del documento Excel. Al final se realizaron los cambios oportunos de manera exitosa.

- **Falta de criterios de inclusión y exclusión al registrar un paciente.**

Los investigadores nos solicitaron mejorar una funcionalidad ya implementada. A la hora de registrar un nuevo paciente, tras introducir el número de identificación del paciente, debía de implementarse una ventana emergente en la cual el usuario rellenaba una serie de criterios de inclusión y exclusión.

Esta mejora se implementó, requiriendo de bastante tiempo y esfuerzo a la hora de implementarla de manera limpia y estética.

CAPÍTULO 7

Conclusiones y Trabajo Futuro

7.1. Conclusiones

En este proyecto hemos abordado el desarrollo de un portal web para la recopilación de datos de un estudio médico basado en la búsqueda de una correlación clara de múltiples hábitos de vida y su influencia en los niveles de vitamina D con el desarrollo de diabetes mellitus. Para ello hemos utilizado Angular como herramienta de desarrollo en *frontend* con HTML, CSS y JavaScript, y Java Spring para el *backend*. El portal ha sido terminado con tiempo para poder desplegarlo en Hostinger y realizar un mantenimiento real del mismo mientras era utilizado por los miembros del estudio.

Una vez finalizado el desarrollo y avanzados varios meses en el mantenimiento de la aplicación podemos hacer una valoración de la experiencia obtenida y los resultados del proyecto.

En cuanto al resultado obtenido en la aplicación web, tanto el equipo médico como nosotros, los desarrolladores, no podríamos estar más satisfechos. La aplicación no incluye nada que ellos no necesiten, está simplificada y solo contiene las funcionalidades buscadas, agilizando el proceso lo máximo posible, ya que este proyecto de investigación es algo extra a hacer a parte de su trabajo y no puede consumirles mucho tiempo. El acceso a la aplicación es privado, al no permitir ningún tipo de registro desde el exterior, además, los datos de los pacientes están asegurados, siendo imposible identificar a cada paciente con una persona real (no almacenamos su DNI), teniendo los datos de estos restringidos a detalles médicos poco sensibles. Las funcionalidades están muy probadas y durante el mantenimiento nos hemos asegurado de conseguir las funcionalidades lo más intuitivas y con prevención de errores posible.

Por otro lado, este proyecto nos ha aportado una buena experiencia en tecnologías que dominábamos poco o nada en muchos casos. Nos ha servido de experiencia profesional con un cliente real, un terreno totalmente inexplorado durante la carrera universitaria, desplegando la aplicación en un servidor real y experimentando el proceso de mantenimiento de la aplicación.

No podemos estar más contentos y orgullosos de haber escogido este proyecto y esperamos que sirva en años futuros como base para muchos más.

7.2. Trabajo Futuro

Finalmente, en este apartado nos gustaría resaltar algunas funcionalidades extra que, aunque no son imprescindibles, nos habría gustado tener tiempo de implementar:

- **Gráficas comparativas en detalle.**

Aunque el equipo médico aseguró en numerosas ocasiones que ellos funcionaban siempre con tablas de Excel para el manejo de datos, no cabe duda de que unas gráficas ilustrativas de la información recopilada en el estudio habrían sido útiles. Quizá no aportasen nada nuevo a este proyecto de investigación en concreto, pero a futuro podrían permitir descubrir otros patrones en los pacientes o servir de base a investigaciones distintas.

- **Formularios dinámicos.**

Una idea que surgió durante el desarrollo de los últimos detalles de la aplicación y habría ampliado muchísimo su potencial. Básicamente se planteó la idea de poder añadir, eliminar o modificar campos en los formularios de tal forma que la plataforma sería totalmente reutilizable para cualquier proyecto. Esto por supuesto supone un desafío a nivel de modelo de datos, la información pasaría a ser más cómoda almacenada en JSON que en estructuras estáticas, la comunicación entre componentes debería saber adaptarse, etc. En definitiva un trabajo complejo que nos habría encantado realizar.

- **Implementación en servidores médicos.**

Nuestro primer despliegue fue realizado en un servidor de Hostinger sencillo en el que pensábamos realizar las pruebas y el mantenimiento con pacientes reales, para luego pasar la aplicación perfeccionada a un servidor del entorno sanitario que asegurase que la aplicación se mantendría tras la finalización de este proyecto. Por desgracia, a causa de la crisis sanitaria del Covid-19, este plan no pudo completarse y la implementación ha quedado pendiente.

- **Sistema de notificaciones.**

Por último, actualmente la comunicación entre investigadores y administradores se realiza de forma externa a la aplicación. Esto actualmente no supone un problema pues todos son compañeros de trabajo y tienen sus propios móviles para hablar en caso de no poder en persona, pero si a futuro se buscara ampliar el proyecto tendría que haber algún tipo de sistema de mensajería o notificaciones. Dicho sistema serviría principalmente para que los investigadores hicieran peticiones de cambios al administrador, ya que ellos mismos no pueden gestionar las citas ya completadas, y para que el administrador les comunicase errores o cambios detectados. En principio un sistema de mensajes cortos sería más que suficiente pero al ponerlo en uso quizás se descubriesen otras herramientas de comunicación más útiles o eficientes.

Conclusions and Future Work

7.1. Conclusions

In this project we have addressed the development of a web portal for data collection from a medical study based on the search for a clear correlation of multiple lifestyle habits and their influence on vitamin D levels with the development of diabetes mellitus. For this, we have used Angular in the *frontend* with HTML, CSS and JavaScript, and Java Spring for the *backend*. The portal has been completed in time to be able to deploy it in Hostinger, and to perform an actual maintenance of it while it was used by the members of the study.

Once the development is finished, and after several months in application maintenance, we can make an assessment of the experience obtained and the project results.

In terms of the result obtained in the web application, both the medical team and us, the developers, we could not be more happier and satisfied. The application does not include anything they do not need, it is simplified and it only contains the desired functionalities, speeding up the process as much as possible, since this research project is something extra to do apart from their job and it can not be time-consuming. Application access is private, since it does not allow any kind of registration from outside. Besides, patient data are assured, making it impossible to identify each patient with a real person (we do not store their DNI), having their data restricted to insensitive medical details. Functionalities are tested, and during maintenance we have ensured that the functionalities are intuitive and error free.

On the other hand, this project has given us a great experience in technologies that we mastered little or nothing in many cases. It has provided us with professional experience with a real client, a completely unexplored field during the university career, deploying the application on a real server, and feeling the application maintenance process.

We could not be more pleased and proud to have chosen this project and we hope that it will serve as a base for many more in future years.

7.2. Future Work

Finally, in this section we would like to highlight some extra features that, although they are not indispensable, we would have liked to have time to implement:

- **Comparative graphs in detail.**

Although the medical team assured on numerous occasions that they always worked with Excel tables for data management, there is no doubt that some illustrative graphs of the information collected in the study would have been useful. Perhaps they did not contribute anything new to this specific research project, but in the future they could allow the discovery of other patterns in patients or serve as a basis for different investigations.

- **Dynamic forms.**

An idea that came up during the development of the last details of the application, and would have greatly expanded its potential. Basically, the idea was raised of being able to add, remove or modify fields in these forms in such a way that the platform would be entirely reusable for any project. Of course this is a challenge at the data model level, the information would become more comfortable stored in JSON than in static structures, communication between components should know how to adapt, etc. In short, a complex job that we would have loved to do.

- **Implementation in medical servers.**

Our first deployment was on a simple Hostinger server where we planned to test and maintain with real patients, and then pass the refined application to a server in the healthcare environment to ensure that the application would be maintained after the completion of this project. Unfortunately, due to the Covid-19 health crisis, this plan could not be completed and implementation is pending.

- **Notification system.**

Finally, currently communication between researchers and administrators is done externally to the application. This is not currently a problem since they are all coworkers and have their own mobile phones to speak in case of not being able to do so in person, but if in the future they seek to expand the project, there should be some kind of messaging or notification system. This system would serve mainly for investigators to make requests for changes to the administrator, since they themselves can not manage the appointments already completed, and for the administrator to report them detected errors or changes. In principle, a short messaging system would be more than enough, but putting it into use may reveal other more useful or efficient communication tools.

CAPÍTULO 8

Contribuciones individuales

En este capítulo se detalla el trabajo realizado por cada uno de los integrantes de este proyecto.

8.1. Eduardo Gonzalo Montero

Al tratarse de un proyecto colaborativo entre 2 personas y dado que tengo experiencia en el terreno debido a mi trabajo, lo primero que hice fue crear los repositorios de Github encargados de guardar y gestionar las partes *frontend* y *backend* de nuestra aplicación, además de establecer como colaborador del repositorio a Sergio. De esta manera disponemos de un sistema de versiones, evitando problemas al trabajar juntos en el mismo proyecto y estableciendo un flujo de trabajo óptimo.

Lo siguiente que hice y dado que poseía cierto manejo con las tecnologías que se iban a utilizar en el proyecto, fue la creación inicial de los proyectos, tanto de la parte *frontend* como de la *backend*. Este proyecto inicial se trataba de una POC (prueba de concepto) en la cual la parte *frontend* de la aplicación, mediante un botón, enviaba una petición HTTP muy simple al *backend*, el cual la recibía y devolvía un mensaje de confirmación en la respuesta, mostrando dicho mensaje en el navegador. Una vez realizado esto, me encargué de suministrar recursos tales como bibliografía, cursos, apuntes y demás a mi compañero Sergio con el objetivo de que aprendiese las tecnologías que íbamos a utilizar, además de estar disponible para cualquier duda que le surgiese en su proceso de aprendizaje.

A continuación y con el objetivo de mejorar la comunicación, la asignación de tareas y el seguimiento del trabajo, creé un tablero Trello y lo compartí con Sergio. De esta manera nos podíamos asignar tareas con diferentes etiquetas, conociendo el estado del proyecto en cada momento.

Seguidamente, creé una base de datos relacional a través de MySQL, gestionándola con el programa MySQL Workbench, conectando la aplicación *backend* con dicha base de datos. Para manejar las tablas de la misma, implementé la tecnología JPA para generar automáticamente las tablas cuando el proyecto compila, además de para gestionar las propias tablas y las relaciones existentes entre ambas, todo ello con el uso de entidades implementadas mediante anotaciones en el código. Todas las entidades JPA de la aplicación así como sus relaciones las fui implementando a lo largo del proyecto según se requerían.

En la parte *backend* del proyecto implementé el controlador, la lógica de negocio y el repositorio necesarios para llevar a cabo el acceso de investigadores a nuestra aplicación web, gestionando todos los posibles errores que pudieran darse, sumando a esto las validaciones de campo necesarias

para detectar posibles errores humanos al llenar el formulario.

Hablando de la calidad del código implementado en la aplicación *backend*, me encargué de realizar tests unitarios a todas las clases posibles. Además de esto me hice cargo de la documentación, mediante comentarios Javadoc en las clases y mediante de utilización de anotaciones Swagger, generando así una interfaz donde podemos gestionar de un vistazo rápido todos los puntos de acceso de la aplicación y las entidades que recibe la misma.

Lo siguiente de lo que me encargué fue de realizar el flujo completo de las funcionalidades de registrar investigador y de listar investigadores desde el perfil de usuario administrador, añadiendo las validaciones necesarias en los campos y la gestión de todos los posibles errores que pudieran darse durante estos procesos. A petición de los médicos, implementé al listar investigadores un mecanismo de ordenación por campos como el nombre o el DNI/NIE, mejorando de este modo la experiencia de usuario.

Para aportar una capa de seguridad al proyecto y que el usuario no pudiese acceder a recursos sin acceso, implementé en la aplicación *frontend* un mecanismo de guardas para proteger las direcciones o *urls*. Dichas direcciones requieren de una identificación y de un tipo de usuario concreto, redirigiendo al usuario a la ventana de registro en caso de que el mismo no posea los permisos o no se encuentre identificado.

Hablando de la experiencia de usuario (UX), me encargué de implementar en la aplicación *frontend* todas las plantillas de ventanas modales para mostrar al usuario los diferentes mensajes de éxito, advertencia o error. A continuación implementé los mecanismos necesarios para que las ventanas modales fuesen dinámicas, activándose o desactivándose cuando el usuario realice diferentes acciones en nuestra aplicación.

En cuanto a la sección de las citas, me encargué de implementar la lógica de validaciones de todos los campos del formulario a la hora de realizar el cuestionario. Dichas validaciones muestran mediante un sistema de colores si el investigador introduce valores permitidos en cada campo, controlando tanto el formato del campo como su rango de valores permitido. Este sistema de validaciones se reutilizó también para modificar citas realizadas previamente. A continuación implementé en la parte *backend* de la aplicación los puntos de accesos necesarios para guardar en nuestra base de datos la información del cuestionario realizado.

Posteriormente me encargué de implementar tanto en la parte *frontend* como en la parte *backend* de nuestra aplicación la funcionalidad de generar un documento en formato de hojas de cálculo *Excel* en el cual se encontraba toda la información relativa a las citas realizadas por cada paciente. Para esta funcionalidad desarrollé el botón, el acceso a la base de datos de las citas y la generación y posterior descarga en el navegador del documento *Excel*.

Dicho esto, a la hora de mostrar todos los pacientes de la aplicación, desarrollé los filtros de búsqueda de pacientes a partir del DNI/NIE de un investigador y a partir del número de identificación de un paciente, actualizando la tabla de los pacientes de manera dinámica cuando el usuario aplica dichos filtros.

Llegado el momento del despliegue, lo primero que realicé fue un proceso de investigación a la hora de elegir la plataforma de *hosting* en la cual deseábamos desplegar la aplicación para comprobar su funcionamiento en un entorno real y para facilitar las revisiones por parte del equipo médico y nuestro tutor, evitando esplazamientos innecesarios a su oficina. Una vez elegido el servidor de *hosting*, desplegué la base datos relacional, el archivo comprimido que contenía la aplicación *backend* y la estructura de paquetes que contenían la aplicación *frontend*. Para ello tuve que configurar toda la aplicación para funcionarse correctamente en la web.

Dado que había participado en una actividad formativa de la tecnología L^AT_EX impartida en la facultad de Informática UCM, me encargué de crear la estructura inicial de la memoria partiendo

de los recursos puestos a disposición a disposición de los alumnos por parte de la Oficina de Software Libre. Además de esto ayudé a Sergio a que se familiarizase con esta tecnología.

8.2. Sergio Pacheco Fernández

Este proyecto fue inicialmente un desafío para mí al desconocer todas las tecnologías usadas en el mismo, a excepción de unos conocimientos bastante rudimentarios de HTML y CSS. Por ello mi primera labor fue ponerme al día, sobretodo con Angular y Java Spring. Para el primero, realicé un curso en Udemy, al cual acudía constantemente cada vez que se presentaba algo nuevo en el proyecto que no sabía resolver. Para el segundo, mi compañero Eduardo me dio unas bases sencillas y al haber usado Java varios años y utilizado JavaFX para implementar interfaces se me hizo fácil adaptarme a las etiquetas y al funcionamiento. Pero antes de ponerme con estas tecnologías tenía un primer reto que superar, Git.

Durante mis años de carrera he utilizado multitud de repositorios y controles de versiones diferentes, siendo el último de estos Git. El problema yacía en que mi experiencia con Git era a través de la plataforma GitHub, y desde su web que facilita mucho las cosas, por lo que jamás había usado Git desde consola y eso fue lo primero que tuve que aprender. Mientras aprendía las nuevas tecnologías antes mencionadas, hacía pruebas sobre un prototipo de Angular y volcaba mis versiones en el repositorio intentando acostumbrarme a la forma de trabajar.

Al tener una idea inicial mejor del desarrollo en *frontend* que en *backend*, de lo primero que me encargué fue del diseño de la página de login. Mi experiencia anterior con un proyecto médico me ayudaba a saber qué patrones de colores y organización elegir, y el resultado fue de buena acogida por el equipo médico. Tras esto, decidimos prioritariamente separarnos las implementaciones de los perfiles de investigador y administrador, encargándose Eduardo más de este último y yo del primero.

El diseño de la página principal de investigador fue lo primero que realicé. Inicialmente Eduardo hizo un diseño en su parte de administrador y yo otro (similar) para el investigador, pero tras sopesarlo con el equipo médico se decidieron por el mío y adapté los componentes de Eduardo para mantener un estilo uniforme. Creé la tabla de pacientes para el investigador y el pequeño formulario para añadir nuevos con el modal de error y más adelante iconos y mejoras de diseño menores. Durante este primer sprint Eduardo disponía de más tiempo que yo entre las clases y aprender las tecnologías y me ayudó con los métodos del *backend* para los cuales me guiaba y revisaba posteriormente.

Después de las primeras reuniones, decidimos que yo haría los resúmenes de las mismas, recopilando en un documento la respuesta a nuestras preguntas y cualquier detalle mencionado durante las mismas, además de los dibujos e indicaciones a papel tomadas. Estos documentos serían claves para la elaboración del Product Backlog y el Mapa de Historias de Usuario que han aparecido durante el capítulo 3 de esta memoria, de los cuales me encargué también por mi cuenta. Mi compañero revisaba y leía todo lo que iba añadiendo y modificando, pero la elaboración del documento y el plasmado de las necesidades del cliente es mío. Con el fin de extraer datos para estos documentos preparábamos un serie de preguntas a modo de entrevista para el equipo médico entre ambos antes del día de la reunión, cuyas respuestas íbamos recogiendo en papel extrayendo lo importante a estos artefactos y los resúmenes de reuniones.

Tras estos inicios, mi primer gran desafío fue el test de las citas para pacientes. El test se compone de más de 30 variables cada una con su validación, rangos, mensajes de error, diseño en *frontend* y representación en la base de datos. Creé HTML y CSS del mismo, los rangos y tooltips de los campos, el objeto transfer en el que guardar los datos del mismo, los servicios necesarios para guardarlo, a excepción de las validaciones de campos, y solucionar algunos errores que se sucedieron con el *Camel Case* que utiliza nuestra base de datos, que provocaba que algunos campos no se guardasen correctamente. Los campos inicialmente eran todos input de escritura que se

fueron modificando a ComboBox, calendarios y selectores radiales. Este test sufría modificaciones con cada reunión en la que obteníamos *feedback* de él y aunque ya no fuese el objeto principal de mi ocupación siempre tenía que volver a hacer pequeñas modificaciones. Además, este test sirve de base para los componentes de visualización y modificación del mismo, que debían revisarse a su vez para asegurar la concordancia.

Lo siguiente de lo que me encargué fue la visualización de test realizados, para lo que reutilicé parte del diseño de la realización de los mismos, cambiando los input por campos de texto estáticos y reorganizándolos en una lista compacta más fácil de leer. Ese mismo diseño sería reutilizado más adelante para la modificación de citas creando un híbrido entre las dos anteriores, manteniendo la forma de listado que facilita la lectura y añadiendo a cada fila un elemento de input idéntico al utilizado para la realización del test, con las mismas validaciones de rangos, tooltips y patrones de colores.

Me ocupé después del apartado de citas en la parte de administrador. En un principio ibamos a dividir los dos roles, pero como me encargué de los test en investigador y Eduardo acabó ayudándome en algunos aspectos de ellos, me encargué también de esta parte. Creé los componentes del listado de citas realizadas tanto en *frontend* como en *backend*, el formulario para modificar citas para el que implementé igualmente todo en ambos lados (a excepción de las validaciones de campos que reutilicé de mi compañero con algunas modificaciones). En el *backend* creé de cero todo el recorrido para los detalles de las citas, desde el repositorio, pasando por entidades y data transfer object hasta los métodos en modelo y controlador que añadí en los ya creados para el perfil de administrador. En este punto tuve el mencionado en otros capítulos error con el *Camel Case*, ya que había mayúsculas en algunos campos del test se respetaban en *frontend* y no en *backend* y viceversa, lo cual me llevó a renombrar y revisar todos los campos desde el modelo de datos a los mismos componentes de Angular.

Tras terminar con todo el tema de los test, retomé el perfil de investigador que había dejado a medias anteriormente finalizando el diseño e implementando la modificación de contraseña, reutilizando parte del código en *backend* creado para modificarla desde el administrador. En este momento el equipo médico nos pidió una modificación en los roles que permitiese a un administrador ser a su vez investigador y tener multitud de administradores, lo que me llevó a tener que montar la navegación entre perfiles de investigador y administrador en *frontend*. Finalmente me dediqué a remediar errores menores mientras mi compañero se encargaba del despliegue de la aplicación.

Tras el despliegue me encargué de errores menores como una errata con el campo de consumo de alcohol en el guardado del formulario, pero el cambio más significativo fue el añadido de los formularios de inclusión. Hasta este momento para añadir un paciente únicamente se necesitaba su código de la tarjeta sanitaria, pero el equipo médico solicitó añadir un formulario con preguntas de elementos que debían y no debían estar para permitir la inclusión de un paciente, como medida de seguridad. Este formulario aparece sobre la página al intentar añadir un nuevo paciente bloqueando el resto de la misma y se compone de una primera parte de requisitos de inclusión, los cuales tienen que ser todos cumplidos, y una segunda con los de exclusión. Junto a este nuevo formulario se añadieron mensajes de error y validaciones, aunque todas las modificaciones se quedaron en la parte de *frontend*.

Finalmente, en cuanto a este documento se refiere me encargué de la elaboración de la introducción, originalmente muy extensa ya que englobaba el desarrollo completo y todo lo referente a la metodología de trabajo, que más adelante se trocearon y separaron, y su traducción. El capítulo entero de desarrollo para el que tuve que limpiar y adecuar los artefactos de *Scrum*, la parte referente al *frontend* en el capítulo de implementación, las conclusiones y trabajo a futuro y por supuesto, esta aportación.

Bibliografía

- [1] K. S. y Jeff Sutherland, *Guía básica de Scrum.* <https://www.scrumguides.org/docs/scrumguide/v1/scrum-guide-es.pdf>.
- [2] J. Francia, *¿Qué es Scrum?* <https://www.scrum.org/resources/blog/que-es-scrum>.
- [3] Atlassian, *Trello.* <https://trello.com/es>.
- [4] *Página oficial GitHub.* <https://github.com/>.
- [5] U. Hernandez, “Qué es TypeScript.” <https://codigofacilito.com/articulos/typescript>.
- [6] Wikipedia, “HTML5,” 2019. <https://es.wikipedia.org/wiki/HTML5>.
- [7] Wikipedia, “CSS,” 2019. https://es.wikipedia.org/wiki/Hoja_de_estilos_en_cascada.
- [8] Wikipedia, “Java,” 2019. <https://www.java.com/es/download/faq/java8.xml>.
- [9] Wikipedia, “Sun Microsystems.” https://es.wikipedia.org/wiki/Sun_Microsystems.
- [10] Wikipedia, “Java8,” 2019. [https://es.wikipedia.org/wiki/Java_\(lenguaje_de_programacion\)](https://es.wikipedia.org/wiki/Java_(lenguaje_de_programacion)).
- [11] C. E. P. Prado, “SQL.” <https://devcode.la/blog/que-es-sql/>.
- [12] Wikipedia, “Visual Studio Code.” https://es.wikipedia.org/wiki/Visual_Studio_Code.
- [13] “MySQL Workbench.” <https://www.mysql.com/products/workbench/>.
- [14] “Hostinger.” <https://www.hostinger.es/>.
- [15] *Página oficial Bitbucket.* <https://bitbucket.org>.
- [16] Wikipedia, “Eclipse.” [https://es.wikipedia.org/wiki/Eclipse_\(software\)](https://es.wikipedia.org/wiki/Eclipse_(software)).
- [17] *Página oficial Overleaf.* <https://www.overleaf.com>.
- [18] David Pacios Izquierdo José Luis Vázquez Poletti, “Oficina Software Libre UCM - Curso Básico LaTeX.” <https://www.ucm.es/oficina-de-software-libre/actividad-formativa-latex-fdi-ucm>.
- [19] “Publicaciones Oficina Software Libre UCM.” <https://www.ucm.es/oficina-de-software-libre/publicaciones>.
- [20] *Página oficial MobaXterm.* <https://mobaxterm.mobatek.net/>.

- [21] Jonathan Zea, “Spring Framework ¿Qué es y para qué sirve? – Java,” 2017. <https://curiotek.com/2017/06/16/java-que-es-spring/>.
- [22] *Página oficial Maven.* <https://maven.apache.org/>.
- [23] *Página oficial Swagger.* <https://swagger.io/>.
- [24] *Página oficial Mockito.* <https://site.mockito.org/>.
- [25] Wikipedia, “JUnit.” <https://es.wikipedia.org/wiki/JUnit>.
- [26] G. González, “¿Qué es JPA? Diferencia con Hibernate?,” 2019. <http://gregorgonzalez.com.ve/blog/que-es-jpa-diferencia-con-hibernate/>.
- [27] *Página oficial Angular.* <https://angular.io/>.
- [28] Álvaro Fontela, “¿Qué es Bootstrap?,” 2015. <https://raiolanetworks.es/blog/que-es-bootstrap/>.
- [29] B. team, “¿Qué es Bootstrap?.” <https://getbootstrap.com/docs/4.4/getting-started/introduction/>.
- [30] Eduardo Gonzalo Montero y Sergio Pacheco Fernández, “Repositorio frontend.” <https://github.com/myscel/tfg-estudio-medico-front>.
- [31] *Página oficial descarga NodeJS.* <https://nodejs.org/es/download/>.
- [32] Eduardo Gonzalo Montero y Sergio Pacheco Fernández, “Repositorio backend.” <https://github.com/myscel/tfg-estudio-medico-back>.
- [33] *Página oficial Spring initializr.* <https://start.spring.io/>.
- [34] *Página oficial Microsoft Azure.* <https://azure.microsoft.com/es-es/>.