

UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC)  
· BARCELONATECH

FACULTAT D'INFORMÀTICA DE BARCELONA (FIB)  
TRABAJO FINAL DE GRADO

---

# Desarrollo de una herramienta de interacción entre trabajadores sociales y el CUV

---

*Autor:*

Joel Acedo Delgado

*Fecha de defensa:*

Abril, 2019

*Director:*

Xavier Molinero Albareda

*Ponente:*

Salvador Roura Ferret



UNIVERSITAT POLITÈCNICA  
DE CATALUNYA  
BARCELONATECH



*Grado de Ingeniería Informática  
Ingeniería de computadores*



UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC) ·  
BARCELONATECH

## *Resumen*

Grado de Ingeniería Informática

### **Desarrollo de una herramienta de interacción entre trabajadores sociales y el CUV**

por Joel Acedo Delgado

El Centre Universitari de la Visió (CUV) atiende desde hace más de 10 años a más de 2200 pacientes con dificultades económicas cada año, proporcionándoles tratamiento a un precio reducido o incluso gratuito. Para ello, el CUV utiliza la metodología de aprendizaje-servicio, consiguiendo así formar futuros profesionales en el ámbito de la salud visual y, al mismo tiempo, mejorar la calidad de vida de estas personas.

Este proyecto surge de la necesidad de mejora continua del CUV. Concretamente se enfoca en el diseño y la implementación de un nuevo programa para el CUV, que permite mejorar la comunicación entre los trabajadores sociales y el centro. Gracias a este proyecto también se podrá hacer un seguimiento de los montajes de las gafas que se proporcionan a este colectivo, seguir las peticiones de cada visita de los pacientes y gestionar las incidencias que se den ante posibles problemáticas.

El objetivo final del proyecto es permitir al CUV una mejora tanto en su gestión interna, como en la comunicación con las entidades colaboradoras. Al mismo tiempo, se pretende ampliar el número total de pacientes que pueden atender al año.

UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC) ·  
BARCELONATECH

## *Resum*

Grado de Ingeniería Informática

### **Desarrollo de una herramienta de interacción entre trabajadores sociales y el CUV**

per Joel Acedo Delgado

El Centre Universitari de la Visió (CUV) atén des de fa més de 10 anys a més de 2200 pacients amb dificultats econòmiques cada any, proporcionant-los tractament a un preu reduït o inclús gratuït. Per a això, el CUV utilitza la metodologia d'aprenentatge-servei, aconseguint així formar futurs professionals en l'àmbit de la salut visual i, al mateix temps, millorar la qualitat de vida dels pacients.

Aquest projecte sorgeix de la necessitat de millora contínua del CUV. Concretament, s'enfoca en el disseny i la implementació d'un nou programa per al CUV, que permet millorar la comunicació entre els treballadors socials i el centre. Gràcies a aquest projecte també es podrà fer un seguiment dels muntatges de les ulleres que es proporcionen a aquest col·lectiu, seguir les peticions de cada visita dels pacients i gestionar les incidències que es puguin donar.

L'objectiu final del projecte és permetre al CUV una millora tant en la seva gestió interna com en la comunicació amb les entitats col·laboradores. Al mateix temps, es pretén augmentar el nombre total de pacients que poden atendre a l'any.

UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC) ·  
BARCELONATECH

## *Abstract*

Grado de Ingeniería Informática

### **Desarrollo de una herramienta de interacción entre trabajadores sociales y el CUV**

by Joel Acedo Delgado

The Centre Universitari de la Visió (CUV) has been serving for more than 10 years more than 2200 patients with economic difficulties each year, providing treatment at a reduced price or even free. To do this, the CUV uses the service-learning methodology, developing future professionals in the field of visual health and, at the same time, improving the quality of life of these people.

This project starts with the need of constant improvement of the CUV. Specifically, it focuses on the design and the implementation of a new program for the CUV, which allows improving communication between social workers and the center. Thanks to this project, it will also be possible to monitor the assemblies of the glasses that are provided to each patient, follow the requests for visits and manage the incidents that occur in the face of possible problems.

The main goal of the project is to allow the CUV to improve its internal management and communication with collaborating entities, and to expand the total number of patients who can attend the year.

## *Agradecimientos*

En primer lugar me gustaría dar las gracias a todo el equipo del Centre Universitari de la Visió (CUV), en especial a la directora Núria Tomás y al director técnico y coordinador clínico Quique González. También al equipo de la Facultat d'Òptica i Optometria (FOOT), en especial al decano de la facultad Joan Gispets por su soporte a este proyecto.

Mención a parte, me gustaría dar las gracias al vicedecano de Calidad y Economía de la Facultat d'Òptica i Optometria, y director y tutor del proyecto, Xavier Molinero Albareda, por su inestimable ayuda y consejos que han hecho posible la realización de este proyecto.

También me gustaría dar las gracias a Salvador Roura por ser el ponente del trabajo, los servicios administrativos y jurídicos de la UPC por su ayuda en la materia legislativa y a los servicios informáticos del campus de Terrassa por proporcionarnos todo el equipo necesario para la puesta en producción del programa, además de ayuda y consejos para la realización del mismo.

Y por último pero no menos importante, me gustaría dar las gracias a mi familia y amigos que me han acompañado, apoyado y ayudado a lo largo de los buenos y malos momentos de la realización de este proyecto y durante toda la carrera.

A todos ellos, gracias por hacer posible este proyecto.

# Índice general

<b>Resumen</b>	<b>I</b>
<b>Resum</b>	<b>II</b>
<b>Abstract</b>	<b>III</b>
<b>Agradecimientos</b>	<b>IV</b>
<b>1. Contexto</b>	<b>1</b>
1.1. Estado del arte . . . . .	2
1.2. Partes interesadas . . . . .	6
1.3. Objetivos y alcance del proyecto . . . . .	8
1.4. Metodología . . . . .	10
1.5. Monitorización . . . . .	11
1.6. Validaciones . . . . .	11
1.7. Relación con la especialidad . . . . .	12
1.8. Estructura de la memoria . . . . .	14
<b>2. Planificación y recursos</b>	<b>16</b>
2.1. Especificación de las tareas . . . . .	16
2.2. Tiempo dedicado por tarea . . . . .	18
2.3. Diagrama de Gantt . . . . .	19
2.4. Recursos . . . . .	19
2.5. Cambios respecto a la planificación inicial . . . . .	21
<b>3. Presupuesto</b>	<b>24</b>
3.1. Presupuesto de hardware . . . . .	24
3.2. Presupuesto de software . . . . .	25

3.3.	Presupuesto de recursos humanos . . . . .	25
3.4.	Costes indirectos . . . . .	26
3.5.	Coste imprevistos . . . . .	27
3.6.	Coste de contingencia . . . . .	27
3.7.	Presupuesto total . . . . .	27
3.8.	Cambios respecto a la planificación inicial . . . . .	28
<b>4.</b>	<b>Diseño</b>	<b>30</b>
4.1.	UX - Diseño de la experiencia de usuario . . . . .	30
4.2.	UI - Diseño de la interfaz de usuario . . . . .	32
<b>5.</b>	<b>Desarrollo</b>	<b>34</b>
5.1.	Arquitectura y tecnologías . . . . .	34
5.2.	Front end . . . . .	45
5.3.	Back end . . . . .	48
5.4.	Interfaz de comunicación . . . . .	52
<b>6.</b>	<b>Puesta en producción</b>	<b>55</b>
6.1.	Prueba de estrés . . . . .	55
6.2.	Requisitos y selección del servidor . . . . .	62
6.3.	Documentación del usuario . . . . .	63
<b>7.</b>	<b>Plan de contingencia y mantenimiento</b>	<b>65</b>
7.1.	Backups . . . . .	65
7.2.	Plan de contingencia . . . . .	66
<b>8.</b>	<b>Sostenibilidad e impacto social</b>	<b>68</b>
8.1.	Dimensión ambiental . . . . .	68
8.2.	Dimensión económica . . . . .	70
8.3.	Dimensión social . . . . .	72
8.4.	Leyes y regulación . . . . .	73
<b>9.</b>	<b>Conclusiones</b>	<b>74</b>
9.1.	Cumplimiento de los objetivos y reflexiones . . . . .	74
9.2.	Ampliación y trabajo futuro . . . . .	75



**Bibliografía**

# Índice de figuras

2.1. Diagrama de Gantt . . . . .	20
4.1. Wireframe de la pantalla de inicio del personal de administración . . . . .	31
4.2. Wireframe de la pantalla de detalle de un montaje . . . . .	32
5.1. Wireframe de la pantalla de inicio del personal de administración . . . . .	35
5.2. Diagrama del funcionamiento de GraphQL . . . . .	40
5.3. Diagrama de la arquitectura del servidor . . . . .	42
5.4. Funcionamiento de una maquina virtual (Izquierda) vs Docker (Derecha) . . . . .	43
5.5. Estructura de las imágenes Docker, dependencias y volúmenes usados . . . . .	44
5.6. Implementación de la pantalla de detalles del montaje . . . . .	47
5.7. Diagrama de la base de datos . . . . .	49
6.1. Resultados de la prueba 1, 30 usuarios . . . . .	58
6.2. Resultados de la prueba 2, 60 usuarios . . . . .	59
6.3. Resultados de la prueba 3, 120 usuarios . . . . .	60
6.4. Distribución de la latencia de las peticiones en el servidor Apollo . . . . .	61
6.5. Uso de CPU durante las pruebas de estrés . . . . .	62
6.6. Uso de memoria RAM durante las pruebas de estrés . . . . .	62
7.1. GUI del script de recuperación . . . . .	67

# Índice de cuadros

2.1. Duración por tarea . . . . .	19
2.2. Horas de dedicación para cada tarea y rol . . . . .	22
3.1. Presupuesto destinado a hardware (en €) . . . . .	24
3.2. Presupuesto destinado a software (en €) . . . . .	25
3.3. Presupuesto destinado a recursos humanos (en €) . . . . .	26
3.4. Costes indirectos (en €) . . . . .	26
3.5. Costes imprevistos (en €) . . . . .	27
3.6. Costes totales (en €) . . . . .	28
3.7. Desglose de costes por cada tarea . . . . .	28
8.1. Matriz de sostenibilidad . . . . .	68
8.2. Costes anual del programa durante su vida útil (en €) . . . . .	71



# Capítulo 1

## Contexto

En el mundo actual donde las desigualdades y la pobreza desgraciadamente están a la orden del día, es crucial encontrar nuevas soluciones que nos permitan ayudar a la gente con pocos recursos y que necesita ayuda para tener unas condiciones de vida digna. Es por eso, que desde el sector educativo se presenta una oportunidad inmejorable para colaborar y poner en marcha proyectos con un fuerte impacto en este ámbito de la sociedad.

Y es aquí, donde el *Centre Universitari de la Visió (CUV)* [1] junto con la *Facultat d'Òptica i Optometria (FOOT)* [2] de la *Universitat Politècnica de Catalunya (UPC)* [3] son pioneros, desde el año 2008 [4], en acercar el ámbito educativo a un ámbito de exclusión social a través del denominado *Aprendizaje-Servicio* [5, 6].

Gracias a la aplicación de esta metodología son capaces de realizar tratamientos a gente con dificultades económicas, diagnosticando y proporcionando las gafas de manera totalmente gratuita en los casos necesarios. Y al mismo tiempo, permite a los estudiantes practicar en entornos reales, mejorando de esta manera la calidad de su formación académica y por lo tanto la habilidad y el futuro impacto que tendrán en la sociedad una vez terminada su formación.

Además, todo esto no sería posible sin el trabajo que se realiza desde el personal directivo y administrativo del CUV y de la FOOT, buscando y gestionando la colaboración con empresas del sector, las cuales hacen posible esta metodología a través de donaciones e implicación en los tratamientos de los pacientes.

Así, desde hace 10 años el *Centre Universitari de la Visió* es capaz de atender a más de 2200 pacientes al año, derivados por múltiples servicios sociales

de toda Cataluña [7]. Y es por eso que, con una gran experiencia acumulada a lo largo de los años, el CUV ha decidido plantear un nuevo programa informático que les permita seguir mejorando tanto su gestión interna como la gestión y comunicación que realizan con los diversos servicios sociales. Estos servicios hacen hincapié tanto en los trabajos sociales como en los pacientes con pocos recursos. De esta manera, el presente trabajo, pretende estudiar, diseñar e implementar dicho programa.

## 1.1. Estado del arte

### 1.1.1. Aprendizaje-Servicio

Las bases de esta metodología se remontan al 1903 en la universidad de Cincinnati, Estados Unidos, donde surgió como un modelo de educación cooperativa con el objetivo de integrar la educación junto con el trabajo o servicios, acercando así el sector educativo a la realización de acciones sociales que impacten directamente en el bienestar de la sociedad.

No obstante, no es hasta finales de los años 80 que se empieza a popularizar el término, expandiéndose por todo el sistema educativo estadounidense, y se definen una serie de buenas prácticas para aplicar esta metodología desde los institutos hasta las universidades.

Así pues, el impacto que presenta esta metodología para todas las partes implicadas es muy elevado. En concreto para los estudiantes, que pueden aplicar sus conocimientos en un entorno real, cubriendo una necesidad real de la población, y ayudando a la vez, a que estos aprendan como funcionan, participen y en un futuro creen actividades sociales y se hagan responsables y conocedores de las situaciones que les rodean en su comunidad.

Además, al tener que tratar con personas en un entorno real, por ejemplo diagnosticando enfermedades, como es nuestro caso, se consigue que los estudiantes mejoren su pensamiento crítico, ya sea a la hora de tomar decisiones, discutiendo posibles tratamientos para el paciente, o bien a la hora de encarar la gestión de una tarea dentro de un equipo, entre otros casos.

Esta metodología cada vez es más visible en nuestra sociedad y en el sector educativo, desde las escuelas hasta las universidades, y es habitual encontrar actividades que siguen estas prácticas de una u otra manera, e incluso con otro nombre o siendo desconocedores que se está aplicando esta metodología, consiguiendo mejorar poco a poco nuestra sociedad [8, 9].

### 1.1.2. Metodología actual del CUV

Actualmente la gestión interna del CUV se divide en dos grandes bloques. Por un lado, tenemos un software de pago donde gestionan toda la información relativa a los diagnósticos y tratamientos de los pacientes denominado OpenVision [10]. Mientras que por otro lado, la gestión de los montajes de gafas que se tienen que realizar y el seguimiento de estos se hace mediante una hoja de cálculo donde se va actualizando un montaje a medida que se va avanzando, introduciendo entre otros datos la referencia del caso, los estudiantes implicados que se han o tienen que encargarse de realizar las visitas, el montaje y la entrega y ajustes finales.

Por otro lado, a la hora de programar las visitas para los pacientes tienen que realizarlo vía email o teléfono, con el problema añadido que es habitual encontrarse que el CUV y el trabajador social<sup>1</sup> acuerdan día y hora, pero cuando el trabajador social se lo comunica al paciente a este no le va bien. En estos casos hay que buscar otra fecha y hora para dicha visita volviendo a ponerse en contacto con el CUV y repitiendo todo el proceso. Lo normal es que estas situaciones no se repitan en exceso, pero a veces surgen casos complicados en los que se tiene que reprogramar varias veces una visita.

Además, pueden darse incidencias en las visitas, por ejemplo: que un paciente no se presente, o bien no traiga alguna documentación necesaria. Estas incidencias se tienen que comunicar a los trabajadores sociales para que estos hagan los tramites necesarios para solucionarla.

Actualmente toda la comunicación entre el centro y los trabajadores sociales, se realiza por vía telefónica o enviando emails con el consiguiente impacto que esto supone para la dedicación del personal de administración

---

<sup>1</sup>El trabajador social es el encargado de derivar hacia el CUV a los pacientes que lo necesitan. Para ello el trabajador social, debe pertenecer a una de las entidades colaboradoras.

### 1.1.3. Soluciones actuales

A continuación se presentan algunas aplicaciones actuales que podrían suplir las necesidades que presenta el CUV. A modo de resumen la aplicación que estamos buscando tendría que tener:

1. Módulo para concertar visitas.
2. Módulo para gestionar incidentes en las visitas o durante el tratamiento, con comunicación a personas externas al centro.
3. Módulo para la gestión interna de los montajes de los pacientes.
4. Módulo para gestionar las tareas que deben realizar los estudiantes.
5. Módulo para la gestión de un gran número de usuarios con diversos roles.

Así pues, a continuación se presentan las aplicaciones estudiadas indicando, para cada una, cuales de estas características cumplirían:

- **Igaleno:**

Es una aplicación en la nube que permite una gestión integral de una clínica médica, desde la gestión de visitas y asignación de horas a toda la gestión de los tratamientos.

La aplicación es muy completa, cubriendo ampliamente las funciones de los módulos 1 y 3. No obstante la ausencia de características del resto de módulos, la complejidad derivada de la existencia de muchas funciones poco útiles para nuestro caso, además del alto coste por cada usuario que se quiera añadir, hacen fácilmente descartable esta opción [11].

- **DriCloud:**

Esta aplicación que permite una gestión integral de una clínica médica. Esta es una de las opciones más recomendadas, debido a su amplio abanico de características, contando entre ellos aplicación para iOS y un precio muy competitivo para un gran número de usuarios.

En nuestro caso particular encontramos reflejadas las funciones de los módulos 1, 3, 5 e incluso el modulo 4 si adaptamos a los estudiantes



como trabajadores de la clínica. No obstante la ausencia de un módulo de gestión de incidencias con personas externas, y la modificación que se tendría que hacer de los roles que tenemos para adaptarlos al programa, junto con un elevado precio mensual, aunque más competitivo que el resto de opciones, hacen que esta opción tampoco sea adecuada para las necesidades que conocemos del CUV [12].

- **Software de gestión de tareas:**

Por otro lado nos encontramos una gran variedad de software de gestión de tareas en equipos, como pueden ser: Redbooth [13] o Wrike [14], que nos permitirían cumplir con facilidad las funciones de los módulos 3, 4, 5.

Pero de nuevo es una opción que debemos descartar al faltar la parte relativa a los otros módulos, además de que la gestión de tareas y el seguimiento de los montajes se puede complicar debido a que los datos con los que trabajaremos son muy específicos y dependiendo del software que escogiésemos nos podrían limitar a la hora de configurar todos los datos necesarios, además de complicar sustancialmente la relación entre los montajes y los pacientes.

#### 1.1.4. Conclusiones

Una vez revisadas algunas de las soluciones que encontramos en el mercado actual, podemos concluir que no hay ninguna que podamos utilizar, ya que ninguna cumple los requisitos de una manera integral en una única aplicación y por el contrario tiene muchas funciones que no se usarían. Asimismo, la inviabilidad de adaptar estas herramientas a nuestras necesidades nos lleva a la necesidad de crear un nuevo programa adaptado a las necesidades específicas que presenta el CUV y los requisitos que podemos encontrar más adelante en el apartado 1.3.

Además la mayoría de aplicaciones actuales están enfocadas al uso en clínicas médicas y no para el sector educativo o como es el caso del CUV, para metodología de *Aprendizaje-Servicio*. Aunque encontramos una amplia variedad de software que permiten la gestión de horas y tratamientos, no permiten realizar de manera correcta la gestión de las tareas que los estudiantes

realizan. Por otro lado, este software tampoco permite el que personas externas, como son los trabajadores sociales, puedan realizar un seguimiento de los pacientes que han derivado.

Por último, el desarrollo de esta aplicación puede ser un primer paso para migrar el software de pago actual hacia un software open source y gratuito, pudiendo en un futuro expandirse para abarcar nuevas funcionalidades que permitan ir sustituyendo lentamente al software actual.

## 1.2. Partes interesadas

A continuación se detalla cuales han sido las partes interesadas e impulsoras de este proyecto.

### ■ **Desarrollador y diseñador:**

El desarrollador y/o diseñador es la persona a cargo de formular una lista de requisitos a partir de los objetivos propuestos por el cliente, y una vez recogidos, plantear un diseño que cumpla las especificaciones recibidas, además de tener en cuenta factores como la navegabilidad, la interacción con el usuario y no menos importante, el aspecto visual del programa.

Una vez resuelto el diseño, es el desarrollador el encargado de realizar la correcta implementación del software, siguiendo de nuevo las especificaciones y requisitos pactados con el cliente y el diseño propuesto por el diseñador. Además de realizar la documentación necesaria para el usuario y para futuros equipos que se incorporen o tomen el relevo de la programación del software.

Finalmente, en este proyecto, el desarrollador y/o diseñador también es un actor principal en la gestión del proyecto, ya que es la persona encargada de planificar toda la fase de diseño y desarrollo y el encargado de que se vaya avanzando de la manera prevista y resolviendo los probables contratiempos que vayan surgiendo.

### ■ **Director del proyecto:**

El director es el principal encargado de guiar y ayudar al desarrollador durante el desarrollo del proyecto, evitando y corrigiendo posibles errores que surjan, y asegurándose que, a lo largo del proyecto, se cumplan correctamente los requisitos y especificaciones pactados.

- **Personal administrativo y directivo del CUV:**

Es la dirección del *Centro Universitario de la Visión* la principal interesada en el desarrollo del programa ya que es quien ha propuesto el proyecto para mejorar la gestión interna del propio centro y la comunicación con terceras partes implicadas.

De esta manera, tanto la directora del CUV, Núria Tomas, como el Director Técnico y Coordinador del Área Clínica del CUV, Enrique González, han estado en todo momento pendientes del diseño y de la implementación del programa con el objetivo de proporcionar los requisitos y las características finales que tenía que tener el proyecto, así como del seguir la evolución de este.

Por otro lado el personal administrativo junto con la dirección serán los encargados de llevar a cabo toda la gestión de los tratamientos y de la comunicación con los trabajadores sociales y las entidades. Siendo así uno de los principales usuarios del programa final.

- **Estudiantes:**

Por otro lado encontramos a los estudiantes de la *Facultat d'Òptica i Optometria*, que son los encargados de atender a los pacientes y posteriormente realizar el montaje de las gafas que necesiten siempre bajo la supervisión del profesorado. Para ello, tendrán que acceder el programa para conocer que tareas tienen pendientes y modificar los datos oportunos.

- **Trabajadores sociales y entidades:**

Los trabajadores sociales, pertenecientes a una entidad colaboradora del CUV, serán los encargados de programar las visitas de los pacientes que lo necesiten. Proporcionando la información inicial necesaria y proponiendo una franja horaria, en la cual el paciente podría asistir a la visita, y comunicándole la hora definitiva al paciente una vez aprobada por el personal del CUV.

Estos usuarios también serán los encargados de proponer soluciones a las posibles incidencias que se deriven de las visitas o de los montajes relacionados con los pacientes del cual es responsable, y que habitualmente se comunicarán mediante el nuevo programa

### 1.3. Objetivos y alcance del proyecto

El objetivo principal de este proyecto es: Crear una herramienta de trabajo para mejorar la comunicación con los trabajadores sociales que derivan pacientes al CUV, así como realizar un seguimiento de los pasos que se siguen en el montaje de las gafas de cada uno de los pacientes.

#### 1.3.1. Requisitos

El programa tiene que cubrir, como mínimo, las siguientes necesidades:

- **Asistente para la petición de horas:**

Una de las principales funcionalidades que ha de tener el programa es un apartado que permita a los trabajadores sociales solicitar una nueva visita para uno de sus pacientes. De esta manera, el trabajador indicará la información básica del paciente, en caso de que sea su primera vez, o bien seleccionará un paciente que ya tenga en la base de datos y procederá a indicar una franja horaria para su visita.

Una vez realizada la petición, el CUV propondrá una hora que tendrá que ser aceptada por el paciente o bien ponerse en contacto, a través del asistente de incidencias, para gestionar otra visita.

- **Asistente para la gestión de incidencias:**

Otra sección importante, es la gestión de incidencias. Es necesario un apartado a través del cual tanto el personal del CUV como los trabajadores sociales puedan crear una nueva incidencia relacionada con un paciente y/o un montaje y la otra parte pueda dar respuesta desde el mismo programa.

En la misma línea, otro aspecto importante es la inmediatez con la que el destinatario recibe esa incidencia, y por la naturaleza del programa, será habitual no estar constantemente trabajando con él, sobre todo los trabajadores sociales. Es por eso, que se debe incluir un sistema de notificación por email para que el destinatario pueda percatarse antes de lo sucedido y buscar una solución.

- **Gestión y seguimiento de los montajes:**

El programa también debe incluir un apartado para la gestión de los montajes de cada uno de los pacientes. Para ello, cada vez que se inicie un nuevo montaje se introducirá la información necesaria, como puede ser: Número del caso, identificación del paciente, proveedor de los materiales, encargado del montaje, estudiante encargado de realizarlo y fecha prevista de entrega, entre otros campos, y posteriormente se ira actualizando dicha información a medida que se vaya avanzando.

El programa ha de permitir a los trabajadores sociales ver el estado de los montajes relacionados con sus pacientes. Ello mejora el seguimiento que podrán realizar y la información que podrán proporcionar a sus pacientes.

- **Gestión de los usuarios:** Por último, también es necesario crear un apartado para la gestión de todos los usuarios que podrán acceder al programa, de manera que se pueda dar de alta a nuevos usuarios o bien dar de baja o modificar a los existentes. Y por supuesto indicar los roles y permisos que tendrán cada uno.

### 1.3.2. Requisitos técnicos

Además de las especificaciones mencionada en el anterior apartado, también es importante definir cuales son los requisitos de carácter técnico que deben cumplirse. A continuación se definen estos requisitos:

- **Aplicación web:** El programa se ha de poder acceder a través de internet desde cualquier punto con conexión, y ha de ser compatible y visualizarse correctamente con multitud de dispositivos incluidos ordenadores y tablets, siendo menos habitual el uso en dispositivos móviles por ser la pantalla demasiado reducida.

- **Seguridad:** Por la naturaleza de los datos que se tratarán en el programa, aunque los datos médicos del paciente no llegarán a estar expuestos ya que para ello se utiliza otro programa interno, es muy importante garantizar un nivel elevado de seguridad y cumplir con la Ley Orgánica de Protección de Datos y Garantía de Derechos Digitales (LOPD-GDD).
- **Backups y plan de contingencia:** Al mismo tiempo es crucial establecer una política de copias de seguridad para garantizar la durabilidad y fiabilidad de los datos en caso de errores en el programa o terceros agentes que puedan provocar una pérdida de datos. También se debe definir un plan alternativo para poder seguir trabajando en caso de que el programa sufra un fallo crítico y pase un tiempo hasta que vuelva a estar operativo.

## 1.4. Metodología

Aunque cada día es más habitual encontrarnos proyectos guiados por una de las denominadas metodologías *Agile* en este proyecto se optó por utilizar metodologías más cercanas a la clásica *Cascada*. Es decir, plantear primero unos requisitos y especificaciones, posteriormente pasar a la fase de diseño, implementación y finalmente realizar pruebas, documentación y entrega del producto.

Principalmente se optó por esta metodología en *Cascada* debido a la dificultad de reunirse todas las partes implicadas de manera habitual, como es necesario en las metodologías ágiles, en las cuales es necesario estar en constante comunicación e intercambiando feedback continuamente.

Así pues, se ha optado por realizar reuniones más distanciadas, una cada tres/cuatro semanas, en las que se presentó cada una de las etapas, se proporcionó el feedback necesario y finalmente se dio el visto bueno para la siguiente etapa.

No obstante y como es lógico, se mantuvo una comunicación más habitual a través de otras herramientas como email o teléfono para solucionar posibles dudas o errores que se detecten.

## 1.5. Monitorización

Como se ha comentado en el apartado de Metodología, el seguimiento con todas las partes interesadas, especialmente la dirección del CUV y, en la última etapa de ejecución, los servicios informáticos del Campus de Terrassa de la UPC, se realizó mediante reuniones periódicas cada tres/cuatro semanas. No obstante también se mantuvo una comunicación más habitual vía email y telefónica para solucionar dudas o errores de carácter más urgente.

El seguimiento por parte del director, Xavier Molinero, se ha realizado a través de una comunicación constante, vía teléfono, email o whatsapp, además de videollamadas, y el envío de manera prácticamente semanal de los avances realizados, pudiendo así recibir feedback rápidamente, además del soporte y la ayuda necesaria a lo largo de todo el proyecto.

Finalmente, a nivel de seguimiento del programa se usó Git junto con Github donde se fueron publicando los cambios que se implementaban. Además los cambios estables se han ido publicado en un servidor de prueba de manera que se pudiese visualizar e interactuar con el programa de manera cómoda.

## 1.6. Validaciones

La validación del proyecto se ha realizado principalmente con las sucesivas entregas de prueba del programa y con la corrección de los detalles necesarios. Por último, también se ha validado con la entrega del programa funcional de manera que cumpla los requisitos, sea estable, seguro y fiable usarlo de manera habitual. En el último estadio del trabajo se ha elaborado la documentación necesaria para los diferentes usuarios y futuros desarrolladores que trabajen en el proyecto.

Además también se ha ido validando si se estaba procediendo correctamente en las diferentes etapas del proyecto a través de las reuniones mencionadas anteriormente y el seguimiento continuo del director del proyecto.

## 1.7. Relación con la especialidad

A lo largo de la carrera se han obtenido diferentes conocimientos que de una u otra forma han influenciado en el desarrollo de este proyecto. Así pues, gracias a estos conocimientos tanto de la especialidad de Ingeniería de Computadores como de todas las otras asignaturas, ha sido posible diseñar e implementar este programa con el uso de gran variedad de tecnologías para conseguir un resultado final estable, seguro y sostenible.

### 1.7.1. Competencias técnicas:

A continuación se detalla como se han trabajado las competencias técnicas de la carrera en este proyecto. Dichas competencias, suponen una combinación de conocimientos, habilidades y valores, que capacitan a una persona para afrontar los retos y proyectos a los que se enfrenta, resolviendo para ello los distintos problemas que se encuentre a lo largo de la realización de estos proyectos, ya sea en un ámbito académico, o bien profesional.

Las competencias técnicas trabajadas en este proyecto se exponen a continuación:

- **CEC2.3:** Desarrollar y analizar software para sistemas basados en microprocesadores y sus interfaces con usuarios u otros dispositivos [En profundidad]

Desde el inicio del proyecto se ha trabajado en el diseño e implementación del software con el objetivo que fuese intuitivo, seguro y compatible con la mayor cantidad de dispositivos. Al mismo tiempo, y teniendo en mente que el software será utilizado en dispositivos portátiles, se ha tenido en cuenta el consumo energético y se han tomado medidas como la reducción de las peticiones de datos al servidor con el objetivo de reducir el impacto en este ámbito.

De la misma forma, también se ha tenido en cuenta todo el aspecto de la integridad de los datos, teniendo presente y diseñando diversos planes de contingencia. Así pues, se han implementado diversas medidas, desde un sistema de backups, hasta la realización de un script que nos



permita convertir los datos del nuevo programa al sistema antiguo para así poder seguir trabajando en caso de un error crítico del software.

Por otro lado, se ha tenido que realizar pruebas de estrés para probar la solvencia del programa en un entorno real y permitir así plantear de forma más precisa las características que tenía que tener el servidor en el cual se desplegaría el programa. Consiguiendo así reducir el coste tanto económico como medioambiental del proyecto al limitar el uso de los recursos a lo que realmente es necesario.

Así pues, a lo largo de las diferentes etapas del proyecto se ha desarrollado en profundidad los aspectos relacionados con esta competencia técnica.

- **CEC2.4:** Diseñar e implementar software de sistema y de comunicaciones [En profundidad]

A lo largo del proyecto se ha tenido que diseñar las diversas partes del programa, las cuales son el front end y el back end, junto con su interfaz de comunicación para realizar la integración entre las dos partes. Pero no solo eso, sino que también se ha diseñado e implementado las bases de datos y un servidor Nginx trabajando como reverse proxy, como explicaremos más adelante, el cual, a parte de redireccionar las peticiones correctamente, también se ha tenido que configurar para que utilice el protocolo TLS con el objetivo de garantizar la seguridad en las comunicaciones.

Una vez terminado el desarrollo, también se ha tenido que idear una manera para migrar el programa al servidor de producción proporcionado por la UPC. Así pues, se ha optado por el uso de Docker y el uso de sus contenedores combinados para virtualizar las aplicaciones necesarias, como son: front end, back end, base de datos y servidor. Pudiendo así migrar a este nuevo servidor de una manera rápida y sencilla.

De esta manera, se puede concluir que a lo largo del proyecto, se ha trabajado en profundidad esta competencia. Ya que se ha tenido que realizar el diseño e implementación de las diversas partes, de su interfaz de comunicación, además de toda la configuración necesaria para facilitar la migración y escalabilidad del programa gracias a Docker.

## 1.8. Estructura de la memoria

Esta memoria se divide en distintos capítulos, desde los objetivos, pasando por el desarrollo hasta las conclusiones. A continuación se describe el contenido de cada uno de ellos.

- **Capítulo 1. Contexto:** En este capítulo se describe el contexto del proyecto junto con el estado del arte, los objetivos y la metodología seguida.
- **Capítulo 2. Planificación y recursos:** En este capítulo encontraremos toda la planificación temporal del proyecto junto con los recursos que han sido necesarios para llevarlo a cabo.
- **Capítulo 3. Presupuesto:** A lo largo de este capítulo se expone toda la información de índole económica del proyecto, es decir, los costes de cada uno de los recursos, imprevistos, coste total y finalmente una comparación con la planificación inicial.
- **Capítulo 4. Diseño:** En este capítulo se profundiza en las tareas de diseño del programa que se han realizado a lo largo del proyecto.
- **Capítulo 5. Desarrollo:** En este capítulo encontramos todo lo relativo a la implementación del programa, desde las tecnologías utilizadas hasta la implementación de la versión final de este.
- **Capítulo 6. Puesta en producción:** En este capítulo se tratan todos los aspectos que van desde la finalización del programa hasta su puesta en ejecución en un entorno de producción.
- **Capítulo 7. Plan de contingencia y mantenimiento:** A lo largo de este capítulo veremos las medidas que se han tomado para garantizar la fiabilidad e integridad de los datos, así como los planes a realizar en caso de errores críticos del software.
- **Capítulo 8. Sostenibilidad e impacto social:** En este capítulo encontraremos toda la información relativa a la sostenibilidad del proyecto en los ámbitos económicos, sociales y medioambientales así como la legislación que se ha tenido que tener en cuenta a lo largo del proyecto.

- **Capítulo 9: Conclusiones:** Finalmente nos encontramos con el apartado destinado a las conclusiones y futuras ampliaciones del proyecto.

## Capítulo 2

# Planificación y recursos

La duración final de este proyecto ha sido de **8 meses**. Comenzando en **Septiembre del 2018** y terminando en **Abril del 2019**. Esta duración ha aumentado significativamente respecto a la planificación que se hizo inicialmente por los motivos que se explican más adelante en el apartado 2.5.

### 2.1. Especificación de las tareas

En esta sección se detallan las tareas que han conformado la realización del proyecto. El tiempo necesario para la realización de reuniones ya se considera dentro del tiempo asignado a cada una de las tareas.

- **T1: Gestión de proyectos:**

Esta tarea ha consistido en la realización del curso de gestión de proyectos impartido por la FIB al inicio del proyecto. La tarea constó de diversas entregas a lo largo del curso con una dedicación de unas **75 horas**.

- **T2: Wireframes del programa:**

Esta tarea ha consistido en la realización del diseño inicial del programa a un nivel muy básico similar a un boceto. El objetivo es definir cuales eran las pantallas que tendría el programa, qué información contendría cada una de ellas, además de diseñar y estudiar la interacción que habría en cada pantalla y la navegabilidad existente entre ellas. Esta tarea ha tenido una duración de **20 horas**.

- **T3: Diseño del programa:**

Una vez finalizados los wireframes y obtenido el visto bueno por parte del CUV y el director del proyecto, se realizó un diseño de los componentes que conformaban el programa. Es decir, en lugar de realizar un diseño de cada una de las pantallas, se realizó el diseño de cada uno de los componentes, por ejemplo: botones, listados, menús, notificaciones...

De esta manera se redujo el tiempo necesario de diseño ya que se pudo reutilizar estos componentes en cada una de las pantallas. Esto también ha permitido realizar cambios en el diseño rápidamente y sin complicación.

Junto con el diseño de estos componentes, también se realizó el diseño de un par de pantallas, que usaran los componentes previamente diseñados, para que fuera posible hacerse una idea de como quedaría finalmente el programa. Para la realización de esta tarea se ha necesitado unas **20 horas**.

#### ■ **T4: Desarrollo del programa:**

Una vez se terminó y se aceptó el diseño del programa se realizó la implementación. Para ello hubo que tener en cuenta que se debía desarrollar tanto la parte front end como el back end, además de la creación de toda la estructura de la base de datos necesaria.

Por otro lado, también se tenía que implementar diversas medidas de seguridad, cifrado TSL, varios roles con diferentes permisos para la BD y para el programa entre otras medidas, con el objetivo de garantizar la confidencialidad y la protección de los datos de acuerdo con la legislación vigente.

La duración de esta tarea se ha incrementado en unas **30 horas** debido a los imprevistos que han ido surgiendo a lo largo del desarrollo. Por lo tanto, la realización de esta tarea ha conllevado un total de **310 horas**.

#### ■ **T5: Control de calidad y solución de posibles errores:**

Una vez se terminó el desarrollo del programa se realizó una revisión en profundidad de todos los apartados y de los casos de uso, para detectar posibles carencias y errores, y poder solucionarlos antes de la

puesta en producción y la entrega final del programa. Esta tarea se han dedicado unas **15 horas**.

■ **T6: Configuración del servidor de producción y copias de seguridad:**

Una vez se terminó el desarrollo del programa, fue necesario preparar todo el entorno donde se desplegaría. Para ello se realizó una prueba de estrés que se explicará más adelante. Simultáneamente se hizo la puesta a punto de un sistema alternativo para continuar trabajando ante posibles errores graves que puedan suceder. Esta tarea ha requerido de unas **10 horas**.

■ **T7: Desarrollo de la documentación del usuario:**

A lo largo del proyecto también se realizó la documentación necesaria para los diversos usuarios del programa, que incluye información sobre como utilizar el programa, posibles errores y como solucionarlos. Esta tarea ha necesitado unas **5 horas** de trabajo.

■ **T8: Finalización del proyecto:**

Por último, se realizó la entrega del programa al CUV, junto con toda la documentación necesaria y se revisó que estuviese todo en orden. En esta parte se han necesitado unas **5 horas**.

■ **T9: Finalización del TFG:**

Por otro lado, con el objetivo de concluir el TFG se ha realizado la redacción y revisión de la memoria y la preparación de la presentación y la correspondiente defensa ante el tribunal. Por ello, esta tarea ha requerido unas **20 horas**.

## 2.2. Tiempo dedicado por tarea

En la Tabla 2.1 encontramos el tiempo que ha sido necesario para cada una de las tareas.

Tarea	Duración estimada (h)
T1: Gestión de proyectos	75
T2: Wireframes del programa	20
T3: Diseño del programa	20
T4: Desarrollo del programa	310
T5: Control de calidad y solución de posibles errores	15
T6: Configuración del servidor de producción y copias de seguridad	10
T7: Desarrollo de la documentación del usuario	5
T8: Finalización del proyecto	5
T9: Finalización del TFG	20
Total	480

CUADRO 2.1: Duración por tarea

### 2.3. Diagrama de Gantt

En la Figura 2.1 podemos ver el Diagrama de Gantt de la planificación final del proyecto.

Como se puede observar la tarea T4 es la que conforma la mayoría de la duración del proyecto, abarcando un total de 310 horas repartidas en casi 5 meses de trabajo. Previamente a esta tarea encontramos el desarrollo de la asignatura de GEP, es decir la tarea T1, la cual representa la segunda tarea más larga del proyecto.

Por último, también encontramos las tareas, T6, T7, T8 y T9 las cuales, han supuesto en conjunto el tercer y último bloque de tareas destinadas a terminar el desarrollo del proyecto.

### 2.4. Recursos

A continuación se detallan los diferentes recursos utilizados durante el desarrollo y la vida útil del proyecto.

El hardware del proyecto consiste básicamente en el equipo de trabajo, junto con los servidores de pruebas y de producción.

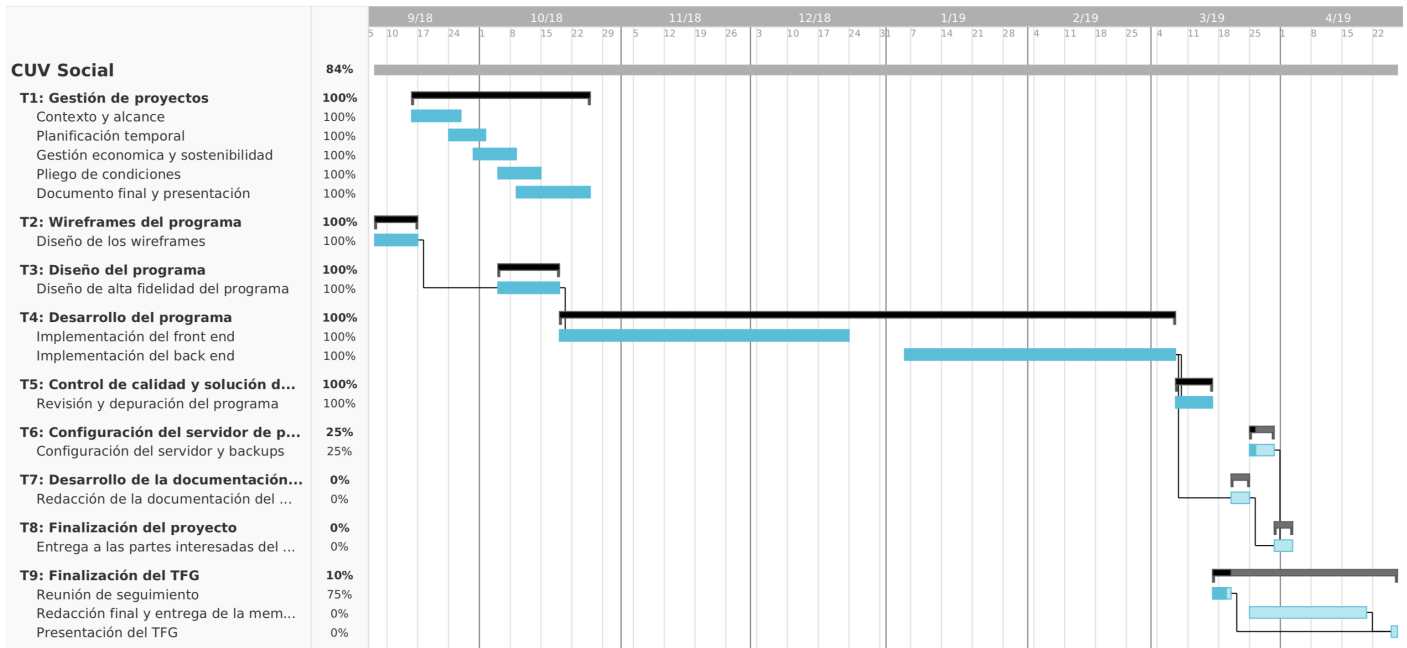


FIGURA 2.1: Diagrama de Gantt

- **Equipo de trabajo:** MacBook Pro 2017 con un procesador Intel Core i7 de cuatro núcleos a 2,8 GHz, 16 GB RAM, 256 GB SSD [15]. Se utiliza en todas las tareas del proyecto.
- **Servidor:** Inicialmente se empleó un servidor de pruebas, consistente en una instancia T2 micro de Amazon Elastic Compute Cloud (Amazon EC2) [16], el cual nos permitió ir probando las diversas versiones estables del programa.

Posteriormente, utilizamos un servidor proporcionado por la UPC para la puesta en producción del programa, que como mínimo debía tener una distribución Linux como sistema operativo y poder ejecutar Node JS y Nginx, además de soporte de bases de datos SQL y suficiente espacio como para gestionar datos de unos 500 usuarios, 3000 pacientes y unos 4000 montajes al año.

Mientras que el software utilizado consiste básicamente en estas 5 herramientas:

- **macOS Mojave:** Sistema operativo del equipo de trabajo. Usado en todas las tareas del proyecto [17].
- **Balsamiq Mockup y Sketch:** Herramientas de prototipaje y diseño. Usados en las tareas T2 y T3 [18, 19].



- **Suite JetBrains:** IDE utilizado para el desarrollo del programa. Usado en las tareas T4, T5 y T6 [20].
- **Overleaf:** Software online usado para la redacción de textos en Latex. Usado en las tareas T1, T7, T8 y T9 [21].
- **TeamGantt:** Software online usado para la creación de diagramas de Gantt. Usado en las tareas T1, T9 [22].

A continuación se describe las personas y/o roles implicados en el desarrollo del proyecto, tal y como se comentó en el apartado 1.2.

- **Gestor del proyecto:** Es el encargado de realizar toda la planificación y supervisión del proyecto. En total se han dedicado unas **115 horas** de trabajo.
- **Diseñador:** Es el encargado de plasmar las especificaciones del programa en un diseño intuitivo, útil y visualmente atractivo. En total se han necesitado unas **46 horas**.
- **Desarrollador:** Es el encargado de realizar la implementación del programa siguiendo los requisitos y el diseño propuestos por el diseñador. En total se han dedicado unas **283 horas** de trabajo.
- **QA Tester:** Finalmente encontramos al encargado de calidad del proyecto. Se encarga de verificar que el programa cumple los requisitos técnicos así como los objetivos y que no reviste ningún fallo de gravedad. En total se han necesitado unas **36 horas**.

En la Tabla 2.2 se desglosa el tiempo dedicado por cada uno de estos roles a cada una de las tareas definidas en el apartado 2.1.

## 2.5. Cambios respecto a la planificación inicial

Respecto a la planificación realizada en la parte inicial han habido cambios importantes en la duración del proyecto. Por otro lado el coste se ha mantenido prácticamente sin modificaciones ya que el aumento de horas sucedido estaba contemplado en los costes de contingencia de la planificación inicial.

Tareas	Duración (h)	Dedicación (h)			
		Gestor	Diseñador	Desarrollador	QA Tester
T1: Gestión de proyectos	75	75	-	-	-
T2: Wireframes del programa	20	-	18	-	2
T3: Diseño del programa	20	-	18	-	2
T4: Desarrollo del programa	310	10	10	270	20
T5: Control de calidad	15	-	-	5	10
T6: Configuración del servidor	10	-	-	8	2
T7: Documentación	5	5	-	-	-
T8: Finalización del proyecto	5	5	-	-	-
T9: Finalización del TFG	20	20	-	-	-
<b>Total</b>	<b>480</b>	<b>115</b>	<b>46</b>	<b>283</b>	<b>36</b>

CUADRO 2.2: Horas de dedicación para cada tarea y rol

Así pues, la principal modificación ha sido que la fecha final del proyecto y de su presentación se ha retrasado de **Enero del 2019** a **Abril del 2019**. Las causas de este retraso han sido principalmente:

- Aumento de la duración de la tarea "**T4: Desarrollo del programa**", y concretamente la subtarea relativa al desarrollo del back end de unas **30 horas**. Esto ha sido debido a una reducción de horas de dedicación del desarrollador debido a causas externas, por lo cual se ha reducido el número de horas que se le ha podido dedicar al desarrollo por día, provocando que haya sido necesario algo más de tiempo para finalizar el desarrollo y por otro lado el aumento de la complejidad de la integración entre el front end y el back end de la estimada inicialmente.

No obstante este retraso no sólo no ha supuesto el incumplimiento de ninguno de los requisitos que había de tener el programa, sino que, gracias al tiempo extra, ha permitido añadir algunas funcionalidades no previstas en un primer momento como:

- Nueva funcionalidad para que el personal docente pueda evaluar el trabajo realizado por los estudiantes desde el programa.
- Opción para añadir nueva información relativa a los trabajadores sociales como el número de colegiado e información de contacto
- Gestión de los convenios de colaboración que tiene el CUV con entidades y proveedores.

- Solución de mayor cantidad de bugs que se han detectado en las pruebas realizadas.

## Capítulo 3

# Presupuesto

En este capítulo veremos cual ha sido el coste final del proyecto y las variaciones que han habido respecto a la etapa inicial.

### 3.1. Presupuesto de hardware

A lo largo del proyecto se han utilizado principalmente dos componentes de hardware definidos en el apartado 2.4. Básicamente son: un equipo de trabajo para diseñar y desarrollar el programa, y un servidor para almacenar y ejecutarlo una vez terminado. El coste de cada uno de ello se detalla en la Tabla 3.1. El coste final se ha calculado teniendo en cuenta una amortización de la duración del proyecto, es decir, 8 meses.

Producto	Precio	Unidades	Vida útil	Amortización	Coste
MacBook Pro 2017	2.463,12	1	5 años	328,41	328,41
Servidor	120 *	1	4 años	-	0 **
<b>Total</b>	<b>2.583,12</b>				<b>328,41</b>

\* Coste anual promedio de un servidor.

\*\* No supone ningún coste directo para el proyecto ya que el servidor lo proporciona la universidad de manera gratuita, y el servidor de desarrollo empleado se encuentra dentro de la capa gratuita de Amazon EC2.

CUADRO 3.1: Presupuesto destinado a hardware (en €)

## 3.2. Presupuesto de software

Además del hardware, también debemos tener en cuenta el presupuesto utilizado a lo largo del proyecto por las diversas herramientas de software que hemos usado tanto por el diseño, la programación del programa, como la redacción de textos y la gestión del proyecto. Así pues, los costes que ha supuesto cada una de estas herramientas se definen en la Tabla 3.2. El coste de final de cada herramienta se ha calculado teniendo en cuenta una amortización de 8 meses.

Producto	Precio	Unidades	Vida útil	Amortización	Coste
macOS Mojave	0	1	-	-	0
Balsamiq Mockup (Prueba)	0 <sup>*</sup>	1	-	-	0
Sketch	49 <sup>**</sup>	1	1 año	32,66	32,66
Suite JetBrains	0 <sup>***</sup>	1	-	-	0
Overleaf	0	1	-	-	0
TeamGantt	0	1	-	-	0
<b>Total</b>	<b>49</b>			<b>32,66</b>	<b>32,66</b>

<sup>\*</sup> Se utiliza la versión de prueba disponible gratuitamente durante 30 días.

<sup>\*\*</sup> El coste original de esta licencia serian 99 €, pero como el desarrollo se trata de un trabajo final de grado se puede utilizar el descuento educativo pasando a costar 49 €[23].

<sup>\*\*\*</sup> El coste original de esta licencia serian 249 €, pero como el desarrollo se trata de un trabajo final de grado se puede utilizar el descuento educativo pasando a costar 0 €[24].

CUADRO 3.2: Presupuesto destinado a software (en €)

## 3.3. Presupuesto de recursos humanos

En este proyecto la parte más importante del presupuesto se ha destinado a la persona encargada de realizar el prototipo, diseño e implementación del programa, al mismo tiempo que también se ha encargado de realizar el papel de gestor del proyecto y el control de calidad. Por todo ello, en la Tabla 3.3 se especifica los costes para cada uno de estos roles y las horas que se han realizado en cada uno de ellos.

Si se quiere ver en detalle el desglose de horas por tarea para cada uno de los roles, se puede consultar en la Tabla 2.2 del apartado 2.4 donde se desglosaba el tiempo dedicado por cada rol a cada una de las tareas definidas en la planificación del proyecto.

Rol	Horas	€/hora	Coste
Gestor del proyecto [25]	115	30	3.450
Diseñador [26]	46	20	920
Desarrollador [27]	283	25	7.075
QA Tester [28]	36	20	720
<b>Total</b>	<b>480</b>		<b>12.165</b>

CUADRO 3.3: Presupuesto destinado a recursos humanos (en €)

### 3.4. Costes indirectos

Los principales costes indirectos que hemos encontrado en nuestro proyecto son: electricidad, internet, espacio de trabajo y transporte para las reuniones. Así pues los costes se especifican en la tabla 3.4.

El calculo del consumo eléctrico se ha calculado según un consumo medio del equipo de trabajo de **76 Wh** [15], un precio de kWh de **0,16 /€kWh** y teniendo en cuenta que durante un 90% del proyecto se estará usando el equipo de trabajo, es decir, unas **432 h**. Esto da un total de **5,25 €**, como se observa en la siguiente formula:

$$0,076kWh * 432h * 0,16€/h = 5,25€$$

Para el cálculo del coste del espacio de trabajo, se ha calculado cogiendo el pago mensual de la hipoteca (unos 500 €), ya que la mayoría del trabajo se efectuará desde casa, y calculando el coste para las horas en las que se está trabajando en el proyecto al mes (480 h/8 meses).

Por otro lado el coste de transporte está calculado a partir del precio de un billete de ida y vuelta de cercanías desde Barcelona hasta el CUV en Terrassa, para las reuniones que se hagan a lo largo del proyecto.

Producto	Precio	Unidades	Coste
Electricidad	5,25	-	5,25
Internet	53	8 meses	424
Espacio de trabajo	60	8 meses	480
Transporte	8,40	8	67.2
<b>Total</b>	<b>122,45</b>		<b>976,45</b>

CUADRO 3.4: Costes indirectos (en €)

### 3.5. Coste imprevistos

Además de los costes anteriores, también se tuvo que tener presente los costes destinados a imprevistos. Así pues, en nuestro proyecto el principal factor de riesgo de imprevistos era el desarrollo del programa, por lo tanto, para poder cubrir posibles gastos derivados de esa tarea, estimamos unas horas extras y los costes correspondientes para cada uno de los roles en la Tabla 3.5.

Rol	Horas	€/hora	Riesgo	Coste total	Coste
Gestor del proyecto	10	30	20 %	300	60
Diseñador	5	20		100	20
Desarrollador	20	25		500	100
QA Tester	5	20		100	20
<b>Total</b>	<b>40</b>		<b>20 %</b>	<b>1000</b>	<b>200</b>

CUADRO 3.5: Costes imprevistos (en €)

Cabe mencionar, que como se ve en la Tabla 3.5, se estimó que todos los roles tendrían que hacer más horas para cubrir una desviación, ya que el gestor tendría que planificar como actuar, el diseñador aplicaría los cambios en caso de que la desviación viniese provocada por un cambio en las especificaciones, y el desarrollador y el QA Tester serían los encargados de implementar y probar las nuevas funcionalidades.

### 3.6. Coste de contingencia

En un primer momento y teniendo en cuenta que era posible que sucediesen desviaciones debido a retrasos en el desarrollo, como luego acabo sucediendo, se estimó un presupuesto para contingencias de un 10 % del subtotal del proyecto. Así pues, esta cantidad ascendía hasta los 1.237,47 €.

### 3.7. Presupuesto total

Sumando todos los costes mencionados en los apartados anteriores el presupuesto total del proyecto asciende a 16.338,04 € como se puede observar

en la Tabla 3.6, lo cual supone un ahorro de 132,7 € respecto al presupuesto inicial.

Concepto	Coste
Recursos hardware	328,41
Recursos software	32,66
Recursos humanos	12.165
Costes indirectos	976,45
<b>Subtotal</b>	<b>13.502,52</b>
<b>EBITA</b>	<b>13.502,52</b>
Impuestos (21 %)	2.835,52
<b>Total</b>	<b>16.338,04</b>

CUADRO 3.6: Costes totales (en €)

En la Tabla 3.7 se muestra un desglose del coste, según la duración y los recursos que ese utilizan, para cada una de las tareas del proyecto descritas en el apartado 2.1 y planificadas en el diagrama de Gantt del apartado 2.3.

Tarea	Horas	Costes				Total
		Hardware	Software	RRHH	Indirectos	
T1: Gestión de proyectos	75	51,31	-	2.250	152,57	2.453,88
T2: Wireframes del programa	20	13,68	-	400	40,68	454,36
T3: Diseño del programa	20	13,68	32,66	400	40,68	487,02
T4: Desarrollo del programa	310	212,12	-	7.650	630,65	8.492,77
T5: Control de calidad	15	10,26	-	325	30,51	365,77
T6: Configuración del servidor	10	6,84	-	240	20,34	267,18
T7: Documentación	5	3,42	-	150	10,17	163,59
T8: Finalización del proyecto	5	3,42	-	150	10,17	163,59
T9: Finalización del TFG	20	13,68	-	600	40,68	654,36
<b>Total</b>	<b>480</b>	<b>328,41</b>	<b>32,66</b>	<b>12.165</b>	<b>976,45</b>	<b>13.502,52</b>

CUADRO 3.7: Desglose de costes por cada tarea

### 3.8. Cambios respecto a la planificación inicial

Como se ha mencionado antes, la principal diferencia respecto a la planificación inicial del proyecto es que se ha conseguido reducir el coste total unos 132,7 €. Esto es debido a que, pese al retraso en el desarrollo del proyecto, los gastos extras que esto ha supuesto ya estaban contemplados en la planificación inicial, concretamente en los gastos para imprevistos y contingencias.



Así pues, pese al aumento del coste destinado a recursos de hardware, software, recursos humanos y costes indirectos que ha supuesto el retraso del desarrollo, estos han sido menores que los fondos destinados para subsanar estas desviaciones, gracias a lo cual el presupuesto final se ha mantenido ceñido al planificado inicialmente.

## Capítulo 4

# Diseño

Todo proyecto de desarrollo software que tenga que tener una interfaz ha de ir acompañado de un importante trabajo de diseño, primero de un diseño de la experiencia del usuario y posteriormente del diseño gráfico de ese primer prototipo.

Esta sección se divide en dos grandes apartados: **UX - Diseño de la experiencia de usuario** y **UI - Diseño gráfico de la interfaz**. En los cuales veremos los detalles de cada una de estas tareas.

### 4.1. UX - Diseño de la experiencia de usuario

El diseño de la experiencia de usuario engloba todo lo relativo a la estructura que se tiene que hacer de la información, la navegabilidad entre pantallas, lo intuitivo que resulta para un usuario realizar una acción determinada... Es decir, engloba todo lo relativo a como el usuario va a interactuar con el software.

Así pues, lo primero que era necesario hacer para llevar a cabo el diseño, era una reunión entre las principales partes interesadas. En este caso, entre el personal directivo del CUV, el director del proyecto, y el desarrollador y diseñador con el objetivo de extraer toda la información que pudiera ser útil para entender como iba a interactuar los distintos usuarios con el programas. Por eso mismo, en estas reuniones también asistieron puntualmente estudiantes del CUV, personal docente y trabajadores de entidades, para expresar su opinión acerca de lo que se estaba diseñando y al mismo tiempo proponer mejoras que habían pasado por alto las otras partes implicadas.

Una vegada vam tenir clar què era la informació i casos d'ús que se tenia que dissenyar i implementar, vam passar al següent pas, el disseny dels wireframes, és a dir, un prototip de baixa fidelitat que nos permetia plasmar tota la informació i la navegabilitat, però sense molts detalls per facilitar els canvis que poguessin sorgir.

Aquesta tasca es va realitzar amb la versió de prova de **Balsamiq Mockup**. Aquesta eina nos permet dissenyar wireframes a través de components bàsics que nos proporciona i ir construint sobre un canvas buit cada capa de la interfície.

A continuació es mostra el wireframe de la pantalla principal (Figura 4.1) i de la pantalla de detall de muntatge (Figura 4.2).<sup>1</sup>

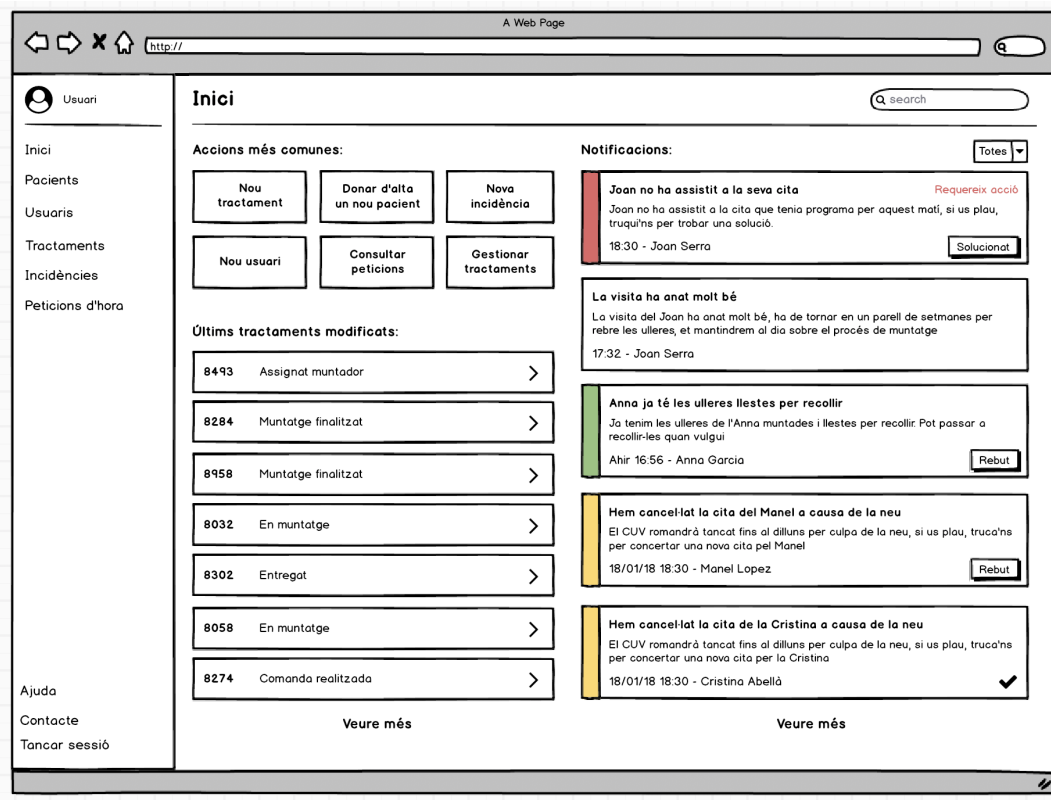


FIGURA 4.1: Wireframe de la pantalla de inici del personal de administració

Una vegada se va tenir una primera versió dels wireframes i davant les dubtes que van sorgint, com per exemple: ¿Què accions eren les més utilitzades?, ¿Què informació podia visualitzar un estudiant i què modificar?, ¿Com es podria repetir un muntatge? o, ¿Es necessita un historial del pacient?, se

<sup>1</sup>En este enlace se puede encontrar todos los wireframes diseñados: <http://bit.ly/2IgJNoP>

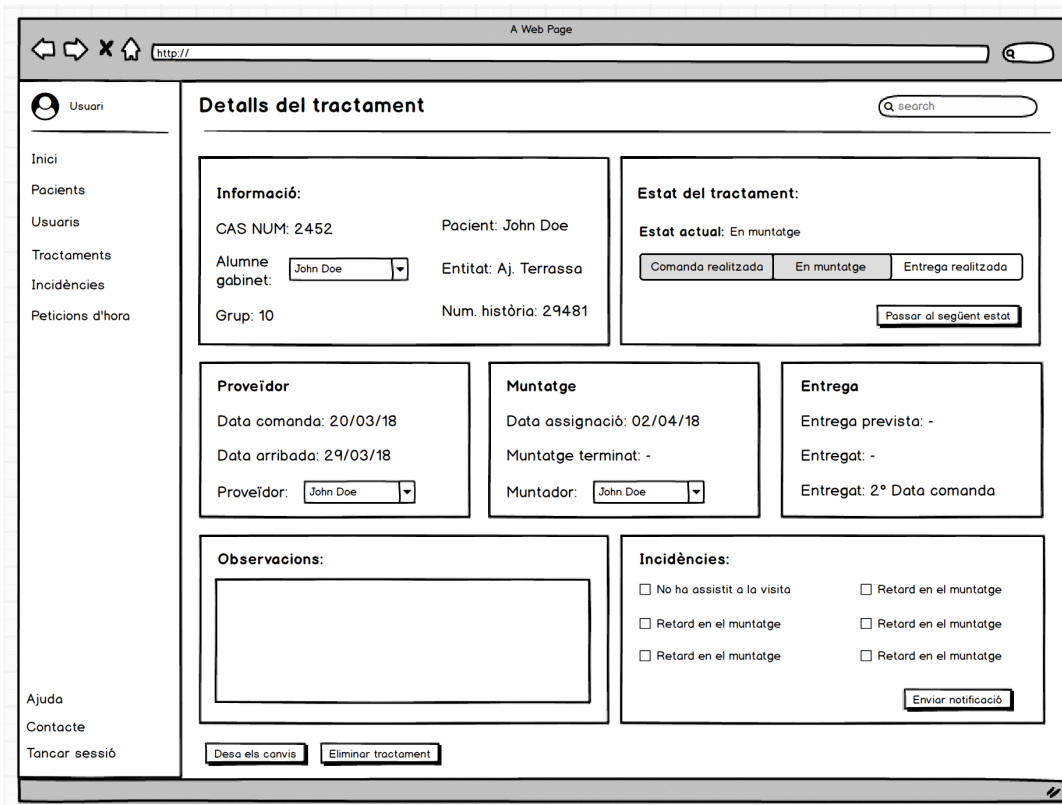


FIGURA 4.2: Wireframe de la pantalla de detalle de un montaje

planteó la necesidad de hacer otra reunión para debatir sobre cuales eran la mejor solución para cada una de ellas y plasmarlo de nuevo en el diseño del programa.

Una vez todas las partes estuvimos de acuerdo, se dio por terminado el diseño de los wireframes, pese a que siempre podían sufrir cambios en un futuro como así acabo sucediendo, y con ellos el diseño de la experiencia del usuario. Así pues, era momento, de comenzar el diseño gráfico de la interfaz.

## 4.2. UI - Diseño de la interfaz de usuario

El diseño gráfico del sistema es el responsable de crear una interfaz atractiva, accesible y adaptable a todos los dispositivos que ejecutaran el software.

Así pues, en una primera reunión fue necesario definir aspectos básicos, como que paleta de colores, que logos utilizar, que dispositivos se tenían que soportar, entre otras cuestiones. Una vez se tuvo claro, y con el objetivo de reducir el tiempo necesario para el desarrollo del programa se optó por utilizar

el framework **Material-UI** [29], el cual nos proporciona una gran cantidad de componentes con el que conformar el diseño de nuestras pantallas de una manera adaptativa y de manera mucho más sencilla que si se tuviese que implementar todo desde cero.

Dicho framework está basado en el sistema de diseño denominado **Material Design** [30] creado por Google en el año 2014, con el objetivo de crear un lenguaje visual para su sistema operativo Android, que permitiera unificar el diseño de las apps y facilitar así la interacción y experiencia del usuario a través de todo su ecosistema.

No obstante, poco a poco este sistema ha ido extendiéndose a otros dispositivos y entornos como es el caso del desarrollo web, y ha terminado convirtiéndose en todo un referente en el mundo del diseño.

Una vez se dio el visto bueno al uso de esta interfaz, y se acordaron todos los elementos del diseño, como color, logos, idioma, etc. se procedió a dar por cerrado el tema del diseño gráfico y pasar, por lo tanto, al desarrollo del programa.

## Capítulo 5

# Desarrollo

En este apartado vamos a ver los detalles del desarrollo del programa, desde las tecnologías que se han utilizado a los detalles de la implementación tanto del front end como del back end y su interfaz de comunicación, además de ver que decisiones se han tenido que tomar a lo largo del proyecto y los cambios que se han realizado respecto a la planificación inicial.

### 5.1. Arquitectura y tecnologías

A continuación veremos el stack tecnológico del proyecto, con una descripción de que es cada herramienta y para que se ha utilizado en este proyecto.

#### 5.1.1. React JS

Como se definen ellos mismos, React JS es una librería Javascript para construir interfaces de usuario.

Esta librería open source, creada por el equipo de ingenieros de Facebook en el 2013, nos permite crear toda la interfaz de usuario a través de componentes y la composición de estos. En cada uno de estos componentes podemos distinguir dos grandes bloques:

- **Interfaz visual:** Los aspectos visuales del componente se implementan mediante **JSX**, una extensión de la sintaxis de Javascript, que nos permite unir la lógica del renderizado con otras lógicas relativas a la UI, como puede ser: gestión de eventos, gestión de los estados, visualización de los datos...

Aunque su uso no es obligatorio con React, esta sintaxis nos permite construir y gestionar toda la interfaz de una manera potente y sencilla a su vez.

- **Gestión de datos:** Una vez la interfaz está construida gracias a JSX, se ha de implementar toda la gestión de los datos que se han de mostrar en la interfaz, como se van a modificar, como se van a obtener de una API, etc.

Por ejemplo, cuando un usuario entre en la pantalla de detalles de un paciente, lo primero que sucede es que se carga los componentes necesarios. Posteriormente, una vez que se ha cargado un componente se solicitan los datos a la API. Una vez estos datos se han obtenido correctamente se actualiza el estado de estos datos en el componente, lo cual a su vez, causa un re-renderizado de los componentes de la interfaz que dependan de los nuevos datos obtenidos, es decir, se modifica el nodo del DOM (Document Object Model) [31] que ha de ser renderizado junto con sus hijos.

Todo esto se gestiona mediante una serie funciones que proporciona React, y que juntas componen el ciclo de vida de un componente que podemos encontrar en la figura 5.1.

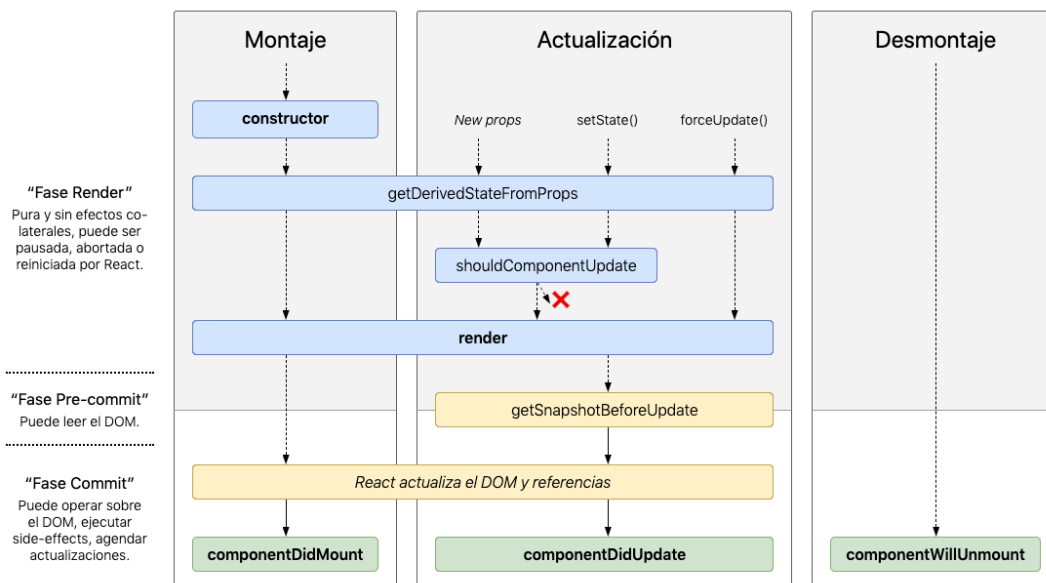


FIGURA 5.1: Wireframe de la pantalla de inicio del personal de administración

Estas funciones, entre otras cosas, nos permite realizar las peticiones a la API en el momento oportuno, actualizar los datos si se detecta un cambio en los datos procedentes del componente padre, o bien decidir si un componente se ha de renderizar de nuevo o no según las condiciones deseadas.

Una vez tenemos un componente, este se puede ir combinando con otros para ir creando la interfaz de una manera reusable, consiguiendo así un código mas optimizado, limpio, comprensibles y mas sencillo de depurar.

### 5.1.2. Node JS

**Node JS** [32] es un entorno de ejecución, lanzado el 2009, bajo un motor V8 Javascript, el mismo que emplea Google en su navegador Chrome, gracias a esto podemos ejecutar código Javascript en el lado del servidor, reduciendo así la duración y complejidad del proyecto al permitir implementar todo el proyecto bajo un mismo lenguaje de programación, es decir, tanto front end, back end como la interfaz de comunicación se implementan en Javascript.

Además de la facilidad que supone no tener que aprender la curva de aprendizaje de nuevos idiomas, Node JS nos permite crear una aplicación escalable lo cual era uno de los principales objetivos con el que se creó.

Esta escalabilidad es gracias a como Node JS trata las peticiones que recibe.

Por un lado PHP o Java genera un nuevo hilo para cada una de las peticiones, asignando una cierta capacidad de procesamiento y memoria RAM a ese nuevo hilo, y ejecutando el código de manera síncrona, bloqueando los threads al realizar tareas de I/O o calculo complejas, por lo tanto este comportamiento lleva a que el numero de peticiones que se puede ejecutar paralelamente va ligado al número de hilos que pueden ejecutarse simultáneamente y por ende limitado a los recursos del servidor.

Mientras que en el otro lado encontramos a Node JS. El cual trata todas las peticiones que recibe en un mismo hilo de entrada, pero una vez se ha recibido la petición, se comprueba que evento se está solicitando e inmediatamente se pasa a ejecutar el resto de funciones necesarias de manera asíncrona en una piscina de threads. Es decir, en Node JS nos encontramos un único hilo de entrada y salida del servidor, pero como este hilo lo único que hace es



identificar que evento se solicita e inmediatamente se ejecuta asincrónamente, se evita la saturación del servidor, consiguiendo así tratar muchas más peticiones con los mismos recursos.

Es decir, con Node JS, se consigue poder soportar un mayor número de peticiones aunque estas puedan tardar más en responderse debido a la cola de peticiones asíncronas, que las que serían posible con PHP o Java, los cuales llegados a un cierto número de peticiones rechazarían las nuevas hasta que no se liberase alguno de los hilos y volviese a haber capacidad para tratar la nueva petición.

Así pues, gracias a este comportamiento, Node JS funciona especialmente bien con aplicaciones web que no son intensivas en cálculos o en operaciones I/O, sino que las peticiones que recibe son sencillas como puede ser el funcionamiento de una API para devolver, crear o modificar datos, como es nuestro caso.

### 5.1.3. MySQL

Como muchos ya conocerán **MySQL** [33] es uno de los sistemas de gestión de base de datos relacionales (RDBMS por sus siglas en inglés) más conocidos.

Se optó por usar este sistema para la base de datos, gracias a que es una de las opciones más utilizadas mundialmente, con una gran comunidad detrás que facilitaría la solución de los posibles problemas, y que para la cantidad de datos que se gestionan en el aplicativo es más que suficiente.

No obstante, inicialmente se optó por el uso de SQLite [34], principalmente para la fase de desarrollo, pero se terminó descartando debido a que su propia sencillez y buen rendimiento cuando la cantidad de datos es muy reducida, hace que no sea una buena elección para un aplicativo escalable y que ha de garantizar la viabilidad de los datos en todo momento.

Una vez se descartó SQLite, también se barajó el uso de PostgreSQL [35], aunque por la cantidad de datos que se gestionarían tanto actualmente como en el futuro, y que si que hacían descartable la opción de SQLite, no se apreciaba diferencias significativas entre esta opción y la seleccionada finalmente.

### 5.1.4. Sequelize

**Sequelize** [36] es un Object-Relational mapping (ORM) para Node JS, que nos permite realizar el mapeo de los objetos y entidades del código en JavaScript del back end a su correspondiente modelo en la base de datos MySQL.

De esta manera, se ha podido implementar de una manera más sencilla y rápida la comunicación y gestión de datos entre el back end y la base de datos. Además de permitirnos abstraer el back end de la BD de manera que sea fácil realizar cambios en una de las partes sin que afecte a la otra.

Esto ha sido especialmente útil a la hora de cambiar de SQLite a MySQL, ya que lo único que se ha tenido que hacer, ha sido modificar la configuración de Sequelize para que apuntase a la nueva base de datos, sin necesidad de modificar ninguna de las peticiones que se realizan a la base de datos.

### 5.1.5. GraphQL

**GraphQL** [37] es un lenguaje de consulta y manipulación para una API. Creado, al igual que React JS, por el equipo de ingenieros de Facebook en el año 2012 y lanzado públicamente en el 2015, nace con el objetivo de ser una alternativa a las predominantes API Restful, y ofrecer a los clientes una forma más directa, sencilla y eficiente para obtener exactamente los datos que requieren, a través de un protocolo potente y dinámico.

Gracias a GraphQL, cambiamos el paradigma de comunicación. Mientras que con las API's REST, era necesario crear un endpoint para cada una de las llamadas que se realizaban y cada uno de estos endpoints devolvía unos resultados concretos. Por lo que, si en algún momento necesitábamos datos que estaban repartidos entre varios endpoints, era necesario realizar las diferentes peticiones y una vez obtenido los resultados, modificarlos en local para que encajasen con nuestras necesidades.

Así pues, con GraphQL, esto cambia. Mientras que antes necesitábamos varias peticiones, ahora sólo necesitamos una, en la que indicamos exactamente que datos necesitamos. De esta manera reducimos el número de peticiones a una, al mismo tiempo que reducimos la complejidad del código del cliente, ya que en la mayoría de casos no será necesario modificar ni tratar esos

datos, ya que estamos obteniendo justamente los que deseábamos y ningún dato más.

Además de simplificar toda la gestión y claridad del código, también se consigue reducir considerablemente el uso de la red, tanto en la cantidad de peticiones como el tamaño de estas, y su respuesta ya que no tratamos ningún dato extra que no necesitamos y que en una API REST sería devuelto tanto si es necesario como si no.

Así pues esta reducción de consumo y de datos es especialmente importante en dispositivos móviles y/o portátiles, que también ven reducido de esta manera el consumo de batería en las operaciones de red.

Antes de ver el proceso mediante el cual GraphQL resuelve una petición hemos de definir una serie de conceptos:

- **Schemas:** Un schema o esquema en castellano, representa un modelo de objeto el cual contiene una serie de datos, queries y mutations. Por ejemplo, tenemos el esquema de Paciente, que contiene: nombre, edad y email.
- **Query:** Una Query es una petición que tiene como objetivo obtener una serie de datos, es similar a un GET en una API REST
- **Mutation:** Una Mutation es una petición que tiene como objetivo añadir, modificar o eliminar, los datos seleccionados. Sería similar a un PUT, PUSH, DELETE, etc. en una API REST.
- **Resolver:** Función que tiene por objetivo resolver, crear o modificar los datos de un esquema que ha solicitado un usuario.

Así pues, en el proceso de funcionamiento de GraphQL encontramos diferentes etapas:

1. **Petición enviada por el cliente:** El cliente envía una query, si quiere obtener datos, o una mutation, si quiere añadir o modificarlos, para ello indica que schema y que datos desea obtener o modificar de la API. En nuestro caso esta petición se realiza mediante la sintaxis propia de GraphQL denominada GraphQL schema language, y se envía a través del cliente de Apollo que se explica en detalle en el apartado 5.1.6.

Una vez se envía la petición, esta se recibe en el servidor back end en el cual se está ejecutando Apollo, que será el encargado de resolver la petición como veremos en el siguiente punto.

2. **Petición recibida en el servidor:** Una vez se recibe la petición, se comprueba que esquemas se están solicitando, y para cada uno de ellos que datos en concreto se están solicitando, así pues si todo esta correcto, es decir los esquemas y datos existen, se llama a las funciones o resolvers que serán las encargadas de resolver ese esquema y devolver los datos concretos que se han solicitado.
3. **Resolvers:** Los resolvers son los encargados de resolver la petición, por lo tanto se puede implementar multitud de opciones diferentes para conseguir esos datos, como puede ser el uso de otra API, un micro-servicio, realizar ciertos cálculos o como es nuestro caso interactuar con la base de datos para obtener o modificar los datos solicitados y devolvérselos al usuario.
4. **Resultado:** Una vez se han resuelto todos los datos de los esquemas solicitados, y si no ha habido ningún error, se devuelven estos datos al usuario. Éste obtendrá simplemente los datos que había solicitado evitando así tanto el overfetch como el underfetch, es decir, obtener más o menos datos de los necesarios.

En la figura 5.2 podemos ver un esquema del funcionamiento de GraphQL.

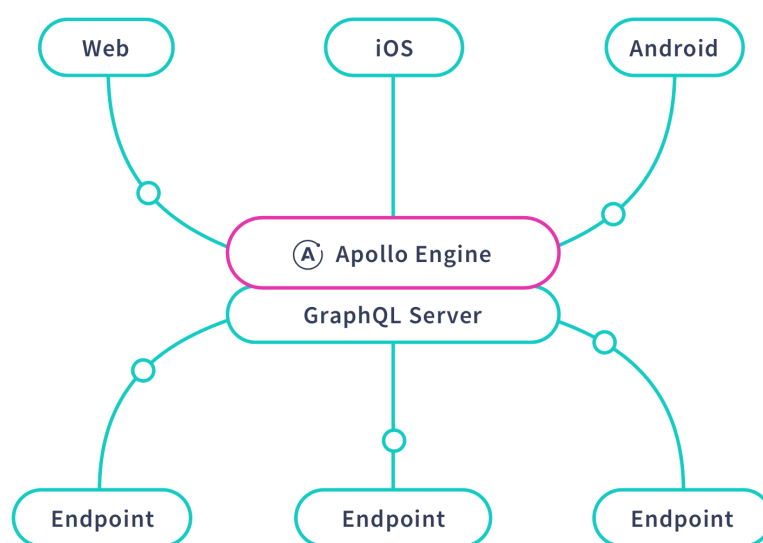


FIGURA 5.2: Diagrama del funcionamiento de GraphQL

### 5.1.6. Apollo GraphQL

**Apollo GraphQL** [38] es una implementación del protocolo indicado en el apartado anterior. En nuestro caso se han utilizado dos de sus versiones:

- **Apollo Client para React JS:** Esta versión es la que se integra con el front end y nos permite realizar las peticiones necesarias al back end, y tratar los resultados o errores devueltos. Así pues, su integración con React JS es bastante sencilla teniendo que configurar algunos parámetros como la URL del back end, el protocolo de autenticación o la política de cache, es decir si se quiere cachear las queries y las respuestas para evitar realizarlas de nuevo. Una vez realizada esta configuración solo queda implementar las peticiones y ejecutarlas cuando sea necesario.
- **Apollo Server:** Por otro lado tenemos esta versión para el back end que se integra con el server de Node JS, haciendo que todas las peticiones que recibe el server vayan redirigidas hacia este framework que es el encargado de tratarlas y devolver un resultado. Esta implementación es la que nos permite ante una petición concreta, ver que esquemas y que datos de estas solicitando el usuario, y ejecutar las funciones adecuadas para resolverlas.

### 5.1.7. Nginx

**Nginx** [39] es un servidor web, similar a Apache o Tomcat, pero que ofrece un rendimiento muy superior en cuanto al número de peticiones por segundo que se pueden tratar. Así pues esta, junto con otras características como la posibilidad de usar Nginx como balanceador de carga o reverse proxy es lo que hace que este servidor sea cada vez más popular.

Al utilizar Nginx como reverse proxy, podemos redirigir las peticiones a una u otra ruta y servir así tanto nuestro front end como el back end con un solo servidor. Además, se consigue que toda la comunicación vaya cifrada con SSL, sin necesidad de implementarlo individualmente para cada una de las aplicaciones que tengamos. Por otro lado, Nginx nos permitiría escalar en un futuro el aplicativo de manera sencilla al poder ejecutar la aplicación en diversos servidores simultáneamente y configurarlo para que redirija la petición a un u otro servidor en función de la carga que tenga cada uno de ellos.

En la figura 5.3 podemos ver a grandes rasgos como sería la arquitectura final de nuestro servidor con Nginx funcionando como reverse proxy.

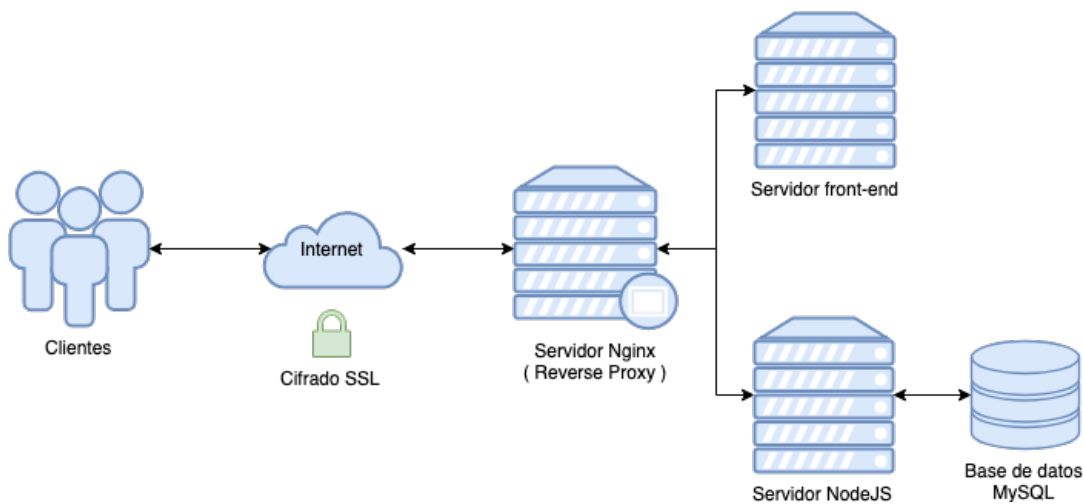


FIGURA 5.3: Diagrama de la arquitectura del servidor

Por lo tanto, gracias a esta velocidad y a la posibilidad de usarlo como reverse proxy con las ventajas que hemos visto, han hecho que Nginx sea la opción más adecuada para este proyecto.

### 5.1.8. Docker

Para la puesta en producción del programa se optó por usar Docker [40], que nos permite a grosso modo crear imágenes de nuestras aplicaciones, para que se pueda desplegar prácticamente en cualquier dispositivo que soporte Docker de una manera rápida y sencilla.

Entrando en detalle lo que nos permite hacer Docker, es crear una imagen de nuestra app, o de cada una de las partes que lo componen como se explicara a continuación, de manera que siempre se compile y ejecute en las mismas condiciones independientemente de la configuración que exista en el equipo real.

Esto es posible, gracias a lo que se denominan Docker containers, es decir, un contenedor de Docker. Esta tecnología se basa en los containers del entorno de los sistemas operativos Linux, los cuales permite virtualizar una parte del código pero compartiendo y usando los recursos y librerías que proporciona el sistema operativo del host. Al contrario que pasa con una máquina virtual,

en la cual se virtualiza tanto la aplicación como el sistema operativo deseado, hecho que provoca que el tamaño final de la imagen sea considerable.

En la figura 5.4 se puede observar como funciona una maquina virtual (izquierda), en comparación con Docker (derecha)

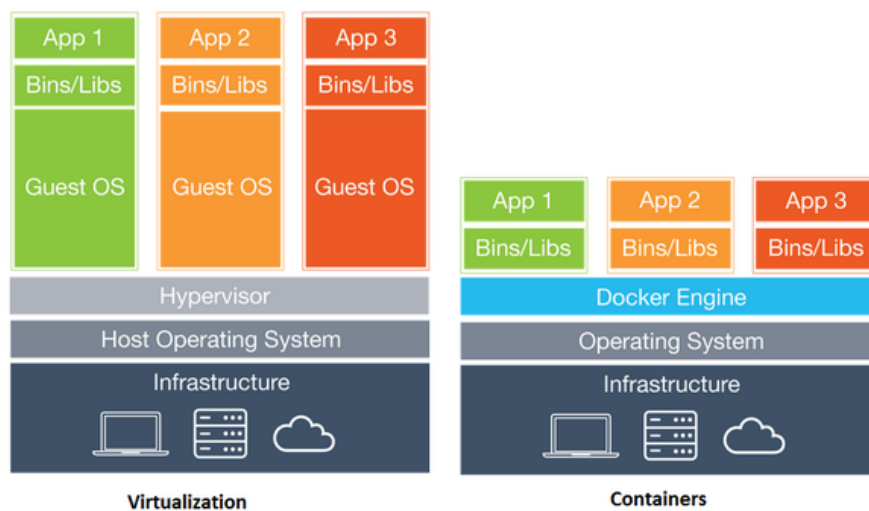


FIGURA 5.4: Funcionamiento de una maquina virtual (Izquierda) vs Docker (Derecha)

Una vez tenemos un Docker container para cada una de las partes individuales de nuestra app, estos se pueden combinar con el objetivo de que se genere una imagen Multi-stage o multi etapas, que comience compilando y ejecutando las imágenes de una manera jerárquica y con dependencias, de manera que finalmente acabe ejecutándose todos los containers necesarios de manera ordenada, y simultanea una vez ya estén todos en ejecución.

A continuación se detalla las imágenes que se han creado y como se han combinado:

- **MySQL:** Imagen que contiene y configura una base de datos MySQL utilizada posteriormente por el back end. Cuenta con un volumen de persistencia que nos permite almacenar los datos con independencia de si se ha detenido o reiniciado el contenedor Docker. A parte, también expone el puerto 3306 (Puerto por defecto de MySQL) para que el contenedor pueda aceptar y redirigir conexiones a ese puerto.
- **API o Back end:** Imagen que contiene y configura toda la parte back end. Se comunica con la imagen de MySQL y por lo tanto depende de esta. Tiene que esperar a que la imagen anterior este en ejecución y

haya terminado la configuración inicial. Igual que la imagen anterior también exponemos un puerto para aceptar conexiones, en este caso es el puerto 4000.

- **Front end:** Esta imagen contiene y configura toda la parte front end. El cometido principal de esta imagen es compilar la aplicación React JS y generar una carpeta build que será la que se retornará a un usuario que entre en la web del programa. Cuenta con un volumen de datos que nos permite compartir la carpeta build generada anteriormente con el servidor Nginx de la imagen correspondiente.
- **Nginx:** Esta imagen contiene y configura toda la parte relativa al servidor Nginx que actúa de reverse proxy como se ha comentado anteriormente. Esta imagen depende de las imágenes de API y front end, y no se ejecuta hasta que estas no están funcionando correctamente. Por otro lado, comparte un volumen de datos con la imagen front end que permite obtener la carpeta build y redirigirla a la correspondiente ruta para que sea devuelta a los usuarios. Por último mencionar, que esta imagen expone los puertos 80 y 443 para aceptar conexiones de HTTP y HTTPS.

En la figura 5.5 podemos ver un diagrama con las imágenes y las dependencias que se han descrito. Cabe mencionar que es una configuración muy parecida a la mostrada en la figura 5.3:

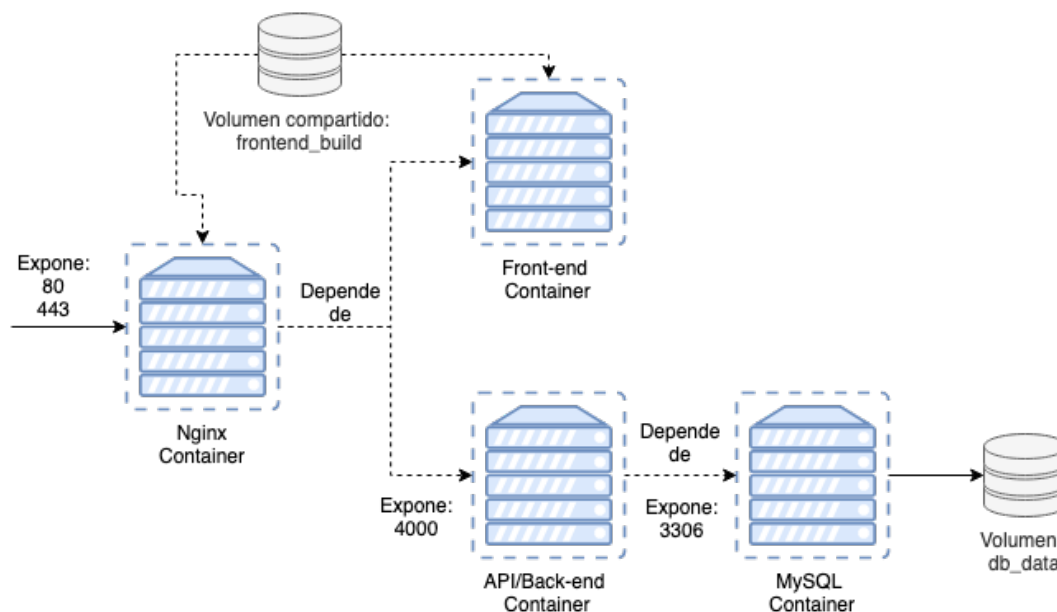


FIGURA 5.5: Estructura de las imágenes Docker, dependencias y volúmenes usados



Por último mencionar que, gracias al uso de Docker se consigue reducir considerablemente la dificultad para desplegar el programa en nuevos servidores, cosa que permite migrar a nuevos servidores sin complicación y/o escalar la aplicación de manera muy sencilla. Ya que otra de las ventajas que ofrece Docker, es la posibilidad de crear replicas del programa actuando, el mismo como balanceador de carga, consiguiendo de esta manera que una misma aplicación se pueda desplegar en entornos completamente distintos y aun así en cualquier caso aprovechen al máximo las capacidades de estos con un cambio de configuración mínimo.

## 5.2. Front end

Una vez terminado el diseño, y estudiado que tecnologías eran las más adecuadas para llevar a cabo la implementación de este proyecto, era momento de pasar al desarrollo.

Concretamente se optó por desarrollar primero la interfaz del programa, es decir el front end, con el objetivo de integrarlo de forma gradual con el back end y poder ir testeando que todo funcionase correctamente.

El primer dilema que surgió era como se iba a separar el programa para dar cabida a los distintos usuarios que tendríamos, es decir, cada usuario tendrá una interfaz con funcionalidades diferentes según su rol. Por lo tanto, teníamos que decidir si hacíamos versiones distintas para cada usuario, que se pudiesen acceder desde diferentes direcciones web, o bien hacer una única versión pero que reflejase esas diferencias.

Vistas las alternativas, se terminó optando por la primera opción, ya que de esta manera, toda la gestión del código sería más sencilla, mientras que además todo quedaba unido bajo un mismo núcleo de código. Es decir, tenemos una serie de funcionalidades básicas y posteriormente las necesarias para cada usuario.

Concretamente se decidió separar el programa en tres grandes bloques internos, y otro más que aglutinara las opciones compartidas. Así pues, nos encontramos con estos bloques:

- **Bloque común:** Aúna las funciones compartidas por todos los usuarios, como pueden ser las pantallas de login, recuperación de contraseña, registros, información de usuario, logout, etc.
- **Bloque del trabajador social:** En este bloque encontramos las funciones y pantallas a las que tendrá acceso únicamente los trabajadores sociales de las diferentes entidades. Por ejemplo: ver lista de pacientes de la entidad, dar de alta un nuevo paciente en esta entidad, ver el seguimiento del montaje de uno de sus pacientes, crear o responder a las incidencias y recibir notificaciones de los eventos de los pacientes que tiene asignados, entre otras funciones.
- **Bloque del personal administrativo:** Bajo este bloque encontramos las pantallas y funciones a las cuales tendrá acceso únicamente el personal de administración. Como por ejemplo: ver el listado de todos los pacientes, el listado de montajes, el de peticiones de visitas, el de incidencias, el de usuarios, etc.

Cabe mencionar que este es el grupo más grande de los cuatro, ya que el personal de administración ha de poder gestionar la totalidad de los datos del programa.

- **Bloque de los estudiantes y personal docente:** Por último, encontramos este bloque, que aúna las funciones y pantallas de los estudiantes, tanto normales como de gestión, y del personal docente. Este grupo incluye: ver listado de tareas asignadas, ver listado de montajes por entregar, ver tareas realizadas, etc.

En este grupo se ha optado por unir diferentes usuarios, ya que la diferencia de funciones y pantallas es nimia y por lo tanto añadiendo un poco de lógica a las pantallas ya se podía distinguir entre un rol u otro.

Una vez decidida la estructura del desarrollo, se comenzó la implementación. Para ello lo primero que se tuvo que hacer era crear y configurar todo el proyecto, y añadir las librerías y/o frameworks necesarios, como es el caso de la librería de diseño utilizada, Material-UI, o la librería usada para la gestión de fechas de manera cómoda, MomentJS [41].

Terminada la configuración, se comenzó la implementación de las diferentes pantallas, comenzando por el bloque común y de trabajador social, para posteriormente realizar la implementación de los otros dos. También se optó

por crear una serie de datos falsos para ir mostrándolo en la interfaz, ya que en este momento del desarrollo no había ningún back end que nos pudiera proporcionar datos reales.

Mientras se avanzaba en la implementación y siguiendo lo descrito en los apartados de metodología y monitorización 1.4, se tuvo que poner operativo un servidor para ir publicando las versiones estables del programa. Así pues, se optó por crear un servidor gratis de Amazon Web Service (AWS) al que pudieran acceder todas las partes interesadas para ver como iba avanzando el desarrollo.

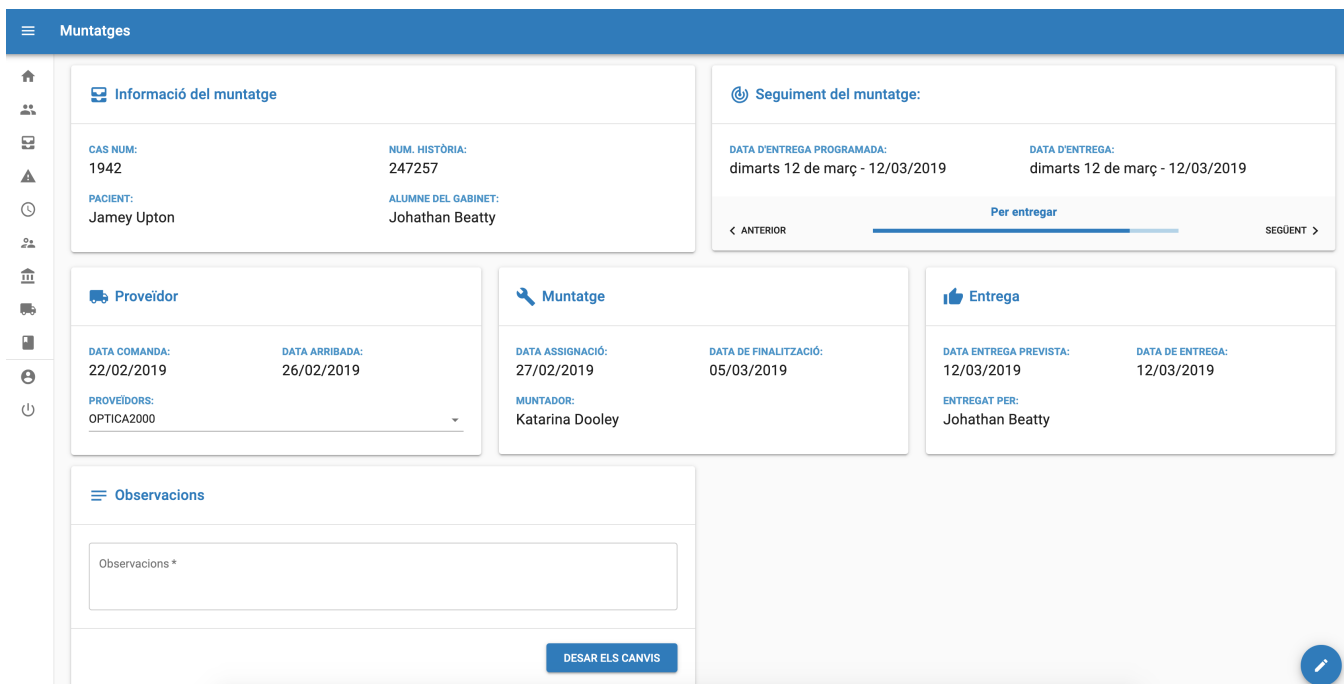


FIGURA 5.6: Implementación de la pantalla de detalles del montaje

Una vez prácticamente terminada la implementación de la interfaz gráfica, y obteniendo el visto bueno por parte del CUV, se dio por zanjado el desarrollo de la interfaz, siendo el resultado final como el que podemos ver en la figura 5.6 que corresponde a la pantalla de detalles de un montaje, la misma que pudimos ver en la figura 4.2.

Zanjada la implementación de la interfaz, se comenzó el desarrollo del back end que se explicará en el siguiente apartado. Así pues, una vez este desarrollo terminó, se tuvo que realizar la integración del back end con el front end.

Para ello, como se mencionó en el apartado de tecnologías utilizadas 5.1, se optó por usar Apollo Client para React JS que nos permitía realizar la integración con la API de GraphQL de manera sencilla. A continuación, se dispuso un nuevo apartado del programa donde se agruparan todas las queries y mutations que se tenían que realizar, y una vez terminado, estas peticiones se llamaban desde los diversos componentes del programa que las ejecutaban en el momento que fuesen necesarias.

Así pues, una vez ya terminado todo el desarrollo y la integración, se comenzaron a hacer pruebas con usuarios reales para detectar el mayor número de errores e ir corrigiéndolos. Además de conseguir feedback para ver si era necesario mejorar o modificar alguna de las características, como por ejemplo el caso de añadir una funcionalidad para permitir modificar la fecha de visita a un trabajador social, la cual en un primer momento no se había descrito o la posibilidad de gestionar los convenios con las entidades desde el propio programa.

Por lo tanto, estos errores o mejoras, se fueron implementando hasta llegar a un estado usable y estable, que pudiera dar por concluido el desarrollo del programa, al menos dentro del ámbito de este trabajo final de grado.<sup>1</sup>

### 5.3. Back end

Una vez se terminó el desarrollo del front end se comenzó con el desarrollo de la parte que se encargaría del tratamiento de los datos y de dar respuesta a las peticiones de los clientes, es decir, el back end.

Para el desarrollo del back end se optó por usar una serie de herramientas que facilitasen el desarrollo y a la vez dieran estabilidad y seguridad al desarrollo final. Estas herramientas, que se pueden ver en detalle en el apartado 1.4, son las siguientes: NodeJS, Apollo Server/GraphQL, MySQL, Sequelize.

Lo primero que se llevó a cabo, fue el diseño de una base de datos SQL, con sus correspondientes tablas, que reflejaran los diferentes elementos u objetos de los cuales tendríamos que gestionar los datos. En la figura 5.7 podemos

---

<sup>1</sup>El código se puede solicitar al autor del presente proyecto a través de la dirección [contact@joelacedo.com](mailto:contact@joelacedo.com), o bien a través del director del proyecto [xavier.molinero@upc.edu](mailto:xavier.molinero@upc.edu)

ver el diagrama de las distintas tablas que conforman la base de datos y la navegabilidad entre ellas.

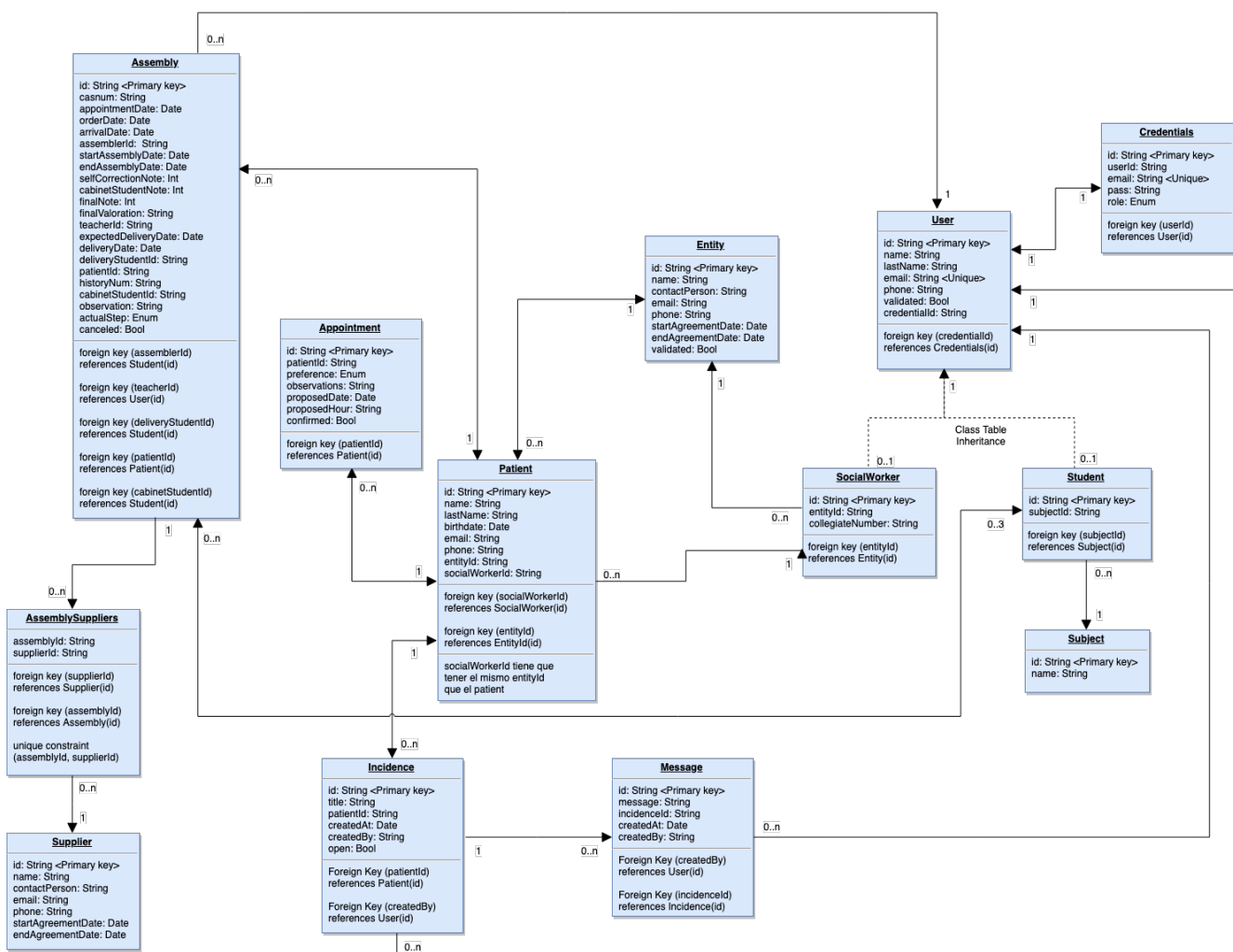


FIGURA 5.7: Diagrama de la base de datos

Terminado el diseño de la BD, lo siguiente que se hizo fue la implementación de los modelos necesarios en el ORM que hemos usado, Sequelite, para plasmar el diseño de la BD de manera que el ORM nos permita realizar las queries necesarias y la navegación propuesta. Así pues para cada una de las tablas de la imagen se realizó un modelo indicando los campos necesarios y sus tipos, si pueden ser nulos, el valor por defecto y las claves primarias.

Una vez terminado los modelos, se pasó a definir la relaciones necesarias para llevar a cabo la navegabilidad mediante el uso del mecanismo que nos proporciona el ORM para crear estas relaciones y las respectivas restricciones y claves foráneas.

Finalizada la puesta a punto de toda la BD y la comunicación con esta a través del ORM, era momento de pasar a la implementación de la API. Para ello, se planificó que schemas, queries y mutations íbamos a necesitar. Así pues, y siguiendo el diseño de la BD se optó por realizar un schema para cada modelo del ORM, de manera que por ejemplo, si se pedía el listado de pacientes desde la API, simplemente se tuviese que llamar al resolver indicado y realizar una petición al ORM para que devolviera ese listado del modelo de paciente.

No obstante, se hicieron excepciones con algunos modelos, por ejemplo el modelo “AssemblySuppliers” ya que este modelo no se va a acceder nunca de manera individual, sino que simplemente es una tabla auxiliar para relacionar los montajes con los proveedores, por lo tanto no tenía sentido hacer un schema y peticiones de la API si nunca se iba a acceder.

Por otro lado también se añadió otro schema auxiliar, encargado de la autenticación de los usuarios.

Llegados a este punto y teniendo ya gran parte del back end desarrollado se comenzó la integración con el front end. A medida que se realizaba la integración y las pruebas correspondiente se fueron detectando y subsanando diferentes errores de distinta gravedad.

Una vez se concluyó esta primera integración se pasó a desarrollar las funciones más complejas de la API, como puede ser añadir distintas funcionalidades como son: búsqueda, filtraje, paginación y ordenación.

Debido a la introducción de estas nuevas funciones, fue necesario ir implementando lógica extra en la mayoría de las queries de los schemas. En concreto se tuvo que implementar que en una misma query, y de manera simultánea, fuera posible:

- Realizar una búsqueda, de manera que los atributos habilitados para la búsqueda contuvieran el texto introducido por el usuario, no necesariamente al principio de la frase.
- Filtrar según una serie de parámetros los datos que se están obteniendo, por ejemplo de un usuario podemos filtrar por los distintos roles que tenemos, seleccionando uno de ellos.
- Realizar una paginación de los resultados, indicando un offset y limit deseados, es decir cuantos resultados quieres obtener a la vez y cuantos

quieres saltarte. Así pues, con estos dos atributos ya tenemos suficiente información para poder realizar una paginación en los listado, que pueden llegar a tener miles de resultado y que en caso contrario sería imposible obtener los datos con un buen rendimiento.

- Ordenar de manera ascendente o descendiente el campo indicado por el usuario a la hora de realizar la petición.

Por lo tanto, en cada query se tenía que comprobar o bien establecer valores por defecto con el objetivo de enviar las peticiones al ORM de una manera estandarizada y que nos permitiera lanzar estas peticiones con todas estas opciones simultáneamente. Por ejemplo, podemos pedir todo el listado de usuarios, que sean trabajadores sociales (Filtraje), que su nombre comience por "An" (Búsqueda), ordenados por el nombre de manera alfabética (Ordenación) y en rangos de 10 en 10 (Paginación).

Y es esta simultaneidad lo que supuso un reto considerable, ya que emitir una query ordenado alfabéticamente, o bien filtrado por un tipo de usuario, es relativamente sencillo. Pero en nuestro caso se ha tenido que implementar una lógica extra para que la query se componga de manera dinámica para permitir aceptar queries básicas hasta queries realmente complejas que usen todas estas características.

Finalmente se implementó la parte relativa a la autenticación, que cuenta con varios controles por así decirlo, que se describen a continuación.

- **Primer punto de autenticación:** Este punto lo encontramos nada más se empieza a gestionar la petición y antes de comenzar a resolver los datos solicitados. En este nivel se verifica que la petición incluye en sus Headers un token valido. En caso afirmativo, se descifra el token y se obtiene el id del usuario. Una vez tenemos el id de usuario se verifica que es un usuario que tiene acceso concedido, y en caso que sea un trabajador social también se verifica que su entidad también tenga el acceso concedido.

Si todo lo anterior es correcto se acepta la petición y se comienza a resolver los schemas solicitados. En caso contrario, si se detecta algún error en lo anterior, se deniega la petición por error en la autenticación redirigiendo al usuario a la pantalla de inicio de sesión.

No obstante hay una serie de peticiones que se aceptan sin necesidad de autenticación, las cuales son las relativas a: inicio de sesión, registro de usuario y recuperación de contraseña. Como se puede observar, estas son peticiones propias de la autenticación del usuario y por lo tanto no se pueden denegar ya que haría imposible iniciar sesión o registrarse en el aplicativo.

- **Segundo punto de autenticación:** Este punto, lo encontramos al inicio de la mayoría de las queries y mutations que se implementan en los schemas, y se encarga de verificar que el usuario que ha realizado la petición dispone de los permisos necesarios para obtener esos datos. Por ejemplo, se comprueba que nadie que no sea un usuario de administración pueda crear o dar acceso a otros usuarios.

Mientras que el primer control es bastante general y solo se encarga de verificar si un usuario existe y tiene el acceso concedido, este segundo nivel es el que, de forma, más granular concede, o no, permisos para que un usuario acceda a ciertos datos.

Cabe mencionar que, gracias a la flexibilidad de GraphQL, se podría implementar hasta un tercer nivel extremadamente granular que comprobase los permisos de un usuario para un campo concreto de los datos. No obstante no se ha creído oportuno implementarlo en este proyecto ya que aumentaría la complejidad del código sin que supusiera un beneficio real para la seguridad del programa.<sup>2</sup>

## 5.4. Interfaz de comunicación

En este último apartado se explicará como funciona la interfaz de comunicación entre el front end y el back end. Así pues, y como se ha mencionado antes en el apartado 5.1, se han utilizado principalmente las siguientes herramientas: Apollo Server, Apollo Client y Nginx.

Por un lado, toda la gestión de las APIs se ha realizado gracias a la implementación de GraphQL que nos proporciona Apollo, concretamente en sus

---

<sup>2</sup>Se recuerda que, como se ha mencionado anteriormente, el código se puede solicitar al autor del presente proyecto a través de la dirección [contact@joelacedo.com](mailto:contact@joelacedo.com), o bien a través del director del proyecto [xavier.molinero@upc.edu](mailto:xavier.molinero@upc.edu)



versiones para Apollo Server, el cual se ejecuta en el servidor Node del back end, y Apollo Client que se integra con el front end desarrollado en React JS.

De esta manera, ha sido relativamente sencillo realizar la integración, ya que simplemente se ha tenido que escribir las peticiones en la sintaxis de GraphQL, pasárselas al cliente y este era el encargado, previa configuración que veremos a continuación, de enviarlas al servidor.

La configuración necesaria del cliente de Apollo es básicamente, la dirección web a la que se desea apuntar, la integración del token de autenticación en los headers de la petición y por último la política de cache deseada.

Una vez zanjado la comunicación entre las dos partes, era necesario añadir una capa intermedia que fuese la encargada de garantizar la seguridad y fiabilidad de las transmisiones. Aquí es donde entra Nginx, actuando como un reverse proxy, con lo cual obtenemos por un lado la facilidad para gestionar la encriptación de las comunicaciones y por otro, una capa de abstracción entre los clientes externos y el servidor back end, de manera que sea fácil realizar o escalar el servidor interno sin necesidad de realizar cambios en los clientes.

Se tuvo que configurar Nginx para que aceptase conexiones a dos rutas. La primera de las conexiones tenía que dar acceso al aplicativo, es decir, tenía que retornar el front end para que el usuario visualizara el programa en su dispositivo, como sucede con cualquier página web. Mientras que la segunda es la encargada de recibir las peticiones dirigidas a la API, para ello se dispuso de la ruta “\graphql”.

Una vez establecidas las rutas, era necesario proporcionar un mecanismo para garantizar la seguridad en las comunicaciones. Para ello se optó por el uso de un certificado SSL. Por lo tanto, lo primero que necesitábamos era dicho certificado, la obtención del cual se logró de forma gratuita gracias a Let’s Encrypt y Certbot [42, 43], gracias a los cuales obtuvimos, tras un sencillo proceso de verificación, un certificado SSL apto para nuestro servidor.

Una vez con el certificado en nuestro poder, se configuró Nginx para que expusiese este certificado de manera que todas las comunicaciones fuesen encriptados gracias a estas claves. Y a continuación, y con el objetivo de facilitar a los usuarios navegar de una manera segura en nuestro aplicativo, se reconfiguró Nginx para que redirigiera todas las peticiones recibidas en el

puerto 80, es decir, el puerto por defecto de HTTP, al puerto 443, de esta manera se fuerza, de manera transparente y sin que el usuario tenga que realizar ninguna acción, a que este navegue con el protocolo HTTPS, utilizando nuestro certificado SSL y por lo tanto encriptando las comunicaciones de manera que se haga un uso seguro de nuestro aplicativo.

Desarrollado el front end, el back end e integrados de manera segura se dio por concluido el desarrollo del programa, a falta de posibles bugs, y se pasó a la etapa de despliegue en producción que veremos en el próximo capítulo.

## Capítulo 6

# Puesta en producción

Una vez terminado el desarrollo del programa, siempre teniendo en cuenta que surgirían imprevistos que se tendrían que ir corrigiendo, era momento de pasar a la fase de despliegue en un servidor de producción para comenzar a realizar pruebas más completas de la aplicación ya en un entorno prácticamente real.

A continuación vemos las diferentes etapas que han sido necesarias para realizar este despliegue.

### 6.1. Prueba de estrés

Lo primero que se tenía que hacer era la realización de una serie de pruebas de estrés para ver como respondía el servidor cuando se recibían peticiones procedentes de varios usuarios conectados simultáneamente.

Así pues, la estimación de usuarios que podrían usar al programa a la vez es bastante reducida. En concreto se estima un máximo de 50 usuarios conectados a la vez, teniendo en cuenta todos los tipos de usuarios del programa. Y por lo tanto, los requisitos hardware que necesitaremos para el servidor serán bastante moderados.

Estas pruebas de estrés se han realizado sobre el servidor de desarrollo, y por lo tanto podrían variar una vez se ejecute en el servidor final, a pesar de que las características del servidor final se estima que sean muy parecidas a la del desarrollo. Las características básicas de este servidor, las cuales corresponden a una instancia T2 micro de Amazon EC2, son las siguientes:

- SO: Ubuntu 18.04
- CPU: 1 vCPU correspondiente con un thread de un Intel Xeon o un AMD EPYC.
- Memoria RAM: 1GB
- Almacenamiento: 15GB, unos 9GB para el SO y 6 GB disponibles para el programa.

Una vez vistas las características del servidor sobre el que se ejecutarán las pruebas, se ha pasado a diseñar estas pruebas. Para ello y teniendo en cuenta la estimación de que en muy raras ocasiones habrá como máximo unos 50 usuarios, se diseñaron 3 pruebas con diferente número de usuarios. Concretamente, la primera de ellas se simulan unos 30 usuarios que podría ser un uso normal/alto del programa, la segunda 60, que ya supondría un uso ligeramente por encima de lo estimado como máximo y en la tercera y última unos 120 usuarios que sería más del doble del uso máximo estimado del programa.

A continuación vamos a ver brevemente que tecnologías se han empleado para la realización de estas pruebas de estrés:

- **Nginx Amplify:** Herramienta de monitorización que nos permite controlar el consumo de recursos del sistema que realiza el servidor Nginx, además de proporcionar información sobre la cantidad de peticiones que se están recibiendo, las conexiones simultáneas, las operaciones I/O entre otros muchos parámetros [44].
- **Apollo Engine:** Similar a la herramienta anterior pero centrado en el servidor Apollo. De esta manera podemos monitorizar el rendimiento del servidor, junto con sus tiempos de respuesta sin tener en cuenta todo lo relacionado con la comunicación o pasos intermedios de una petición. Es decir, con esta herramienta podemos saber cuánto se tarda en resolver una petición desde que llega realmente al servidor. Esta información nos servirá en un futuro para saber que peticiones se tendrían que optimizar [45].
- **Artillery.io:** Herramienta que nos permite simular el comportamiento de un usuario a la hora de utilizar el programa, y generar diversos escenarios con una serie de parámetros como la duración de la prueba, la

tasa de llegada de nuevos usuarios virtuales, entre otros muchos. Una vez terminada la prueba, esta herramienta nos genera un reporte donde se pueden observar detalladamente los resultados de esta [46].

Cada prueba se basa en ejecutar un script de Artillery.io, durante 60 segundos, que nos permite simular el comportamiento de un usuario que utiliza el programa. Concretamente se ha simulado el acceso a las pantallas y datos de los pacientes, montajes, usuarios, incidencias y visitas, tanto a las pantallas de listado como a las de detalle. Para ello se han extraído las peticiones que se efectúan en esas pantallas y se han reproducido en el script.

En este mismo script se define la tasa de llegada que se quiere simular, es decir, a que velocidad van llegando los usuarios al programa. Así pues, teniendo en cuenta que la duración de cada prueba son unos 60 segundos, la tasa de llegada se establece entre usuarios/60, para simular así un uso habitual del servidor entre que el usuario se conecta y realiza las diferentes acciones.

Por otro lado, todas las pruebas se ejecutan en las mismas condiciones. Contamos para ello con una base de datos con la siguiente cantidad de registros de datos falsos para realizar la prueba. Cabe decir que esta cantidad de datos es algo elevada para tensar más las pruebas de estrés, y por lo tanto en un uso real del programa se tardaría cerca de un año y medio o dos en llegar a esta cantidad. A continuación se detalla la cantidad de registros de la BD:

- 300 usuarios, principalmente trabajadores sociales.
- 3000 pacientes
- 4000 montajes
- 3000 visitas
- 1000 incidencias

### 6.1.1. Prueba 1: 30 usuarios

Esta primera prueba, se encuentra basada en lo que podría ser un uso normal e incluso algo superior al habitual. De esta manera es importante que el programa responda de forma rápida y estable a esta cantidad de usuarios. En la figura 6.1 se muestran los resultados:

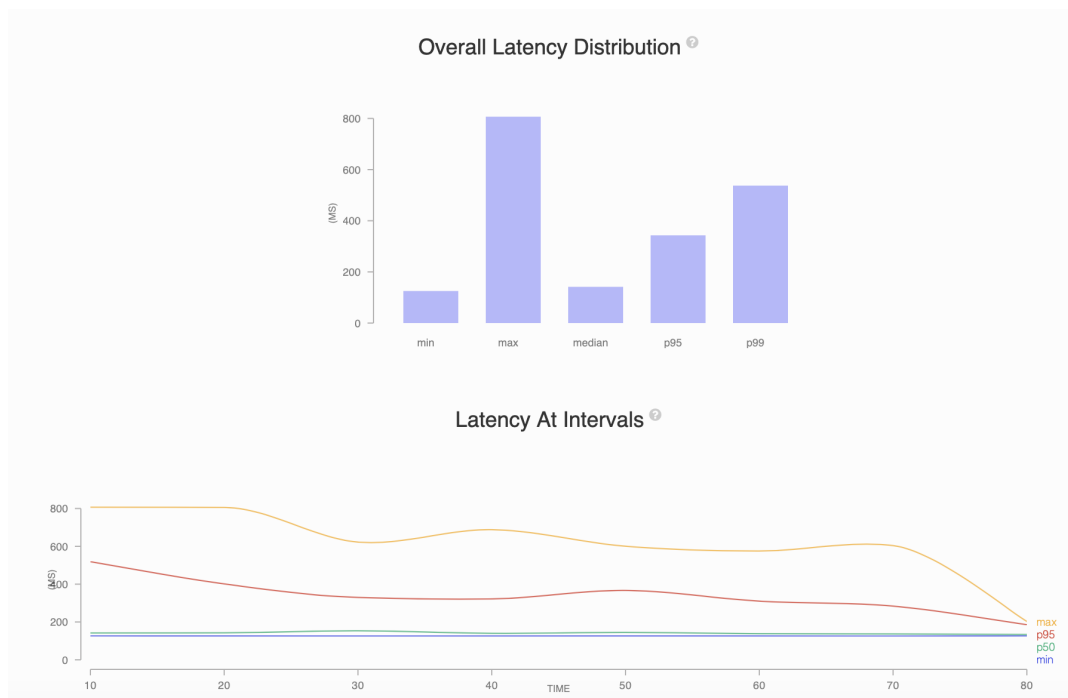


FIGURA 6.1: Resultados de la prueba 1, 30 usuarios

Como se puede ver en los resultados, tras realizar unas **2910 peticiones** al servidor, encontramos que el tiempo de respuesta del 95 % de las peticiones son unos **343 ms** mientras que el 99 % de las peticiones se resuelven en menos de **537.4 ms**, con un máximo del tiempo de respuesta de **806.8 ms**.

Así pues, podemos concluir que para un uso habitual del programa tanto las capacidades del servidor, como la implementación del back end son más que suficientes, respondiendo así, la mayoría de peticiones en menos de 0.5 segundo y la totalidad en menos de 1 s.

### 6.1.2. Prueba 2: 60 usuarios

Mientras que en la primera prueba encontramos lo que sería un uso razonable del programa, en esta segunda ya nos encontramos con un uso bastante intenso para el comportamiento estimado y la cantidad de usuarios que estarán conectados como máximo. Que como ya se ha mencionado serán unos 50. En la figura 6.2 encontramos los resultados:

Como podemos observar, el tiempo de respuesta ha aumentado considerablemente llegando a ser casi el doble de los valores que en la primera prueba. Así pues, encontramos que tras realizar unas **5820 peticiones** el tiempo de

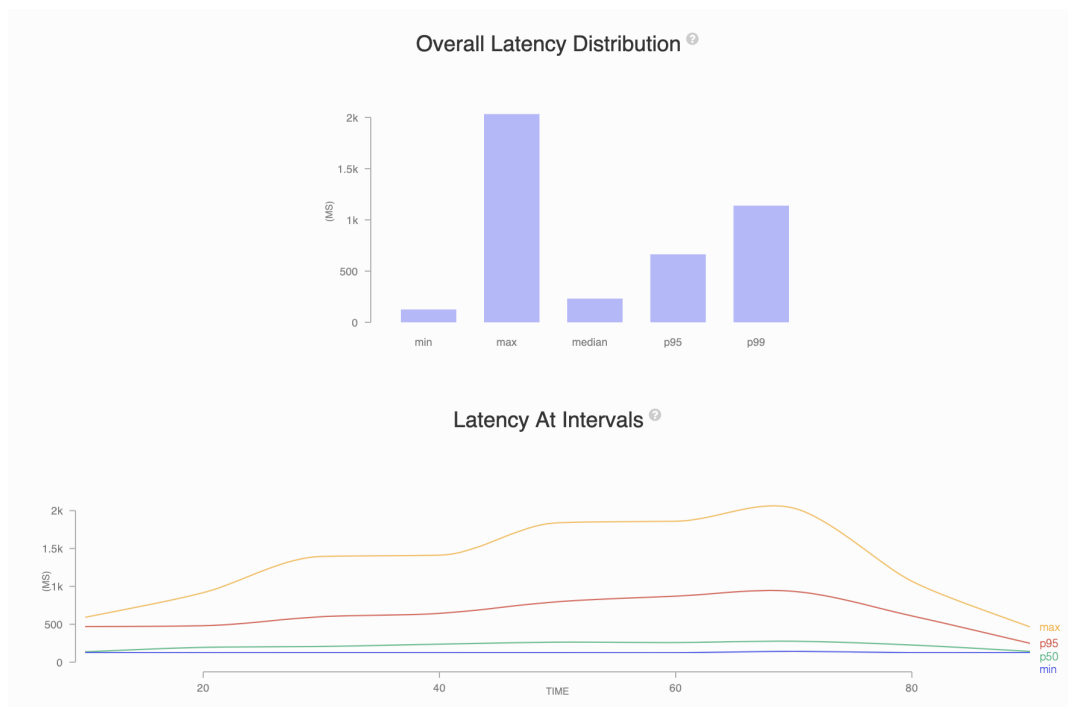


FIGURA 6.2: Resultados de la prueba 2, 60 usuarios

respuesta del 95 % de las peticiones es de **663.5 ms**, mientras que el 99 % de las peticiones quedan resueltas en menos de **1,138.9 ms**. No obstante encontramos alguna petición que puede llegar a tardar hasta unos 2 segundos en ser respondida.

De esta manera, vemos que los tiempos de respuesta han aumentado prácticamente de manera lineal respecto a los de la prueba 1, y que en caso de llegar a darse que haya unos 60 usuarios simultáneamente utilizando el programa, este sería perfectamente usable, aunque alguna petición resultaría bastante lenta.

### 6.1.3. Prueba 3: 120 usuarios

Finalmente se ha realizado una prueba con mas del doble del máximo de usuarios estimados para ver como se comportaría el back end en caso que aumentase en mas del doble la capacidad del CUV y el número de usuarios que utilizan el programa. En la figura 6.3 se muestran los resultados:

Como se puede observar, los tiempos de respuesta aumentan más del cuádruple respecto a la segunda prueba y más de 8 veces respecto a la primera. Obtenemos así, que el 95 % de las peticiones se resolverían con un tiempo de

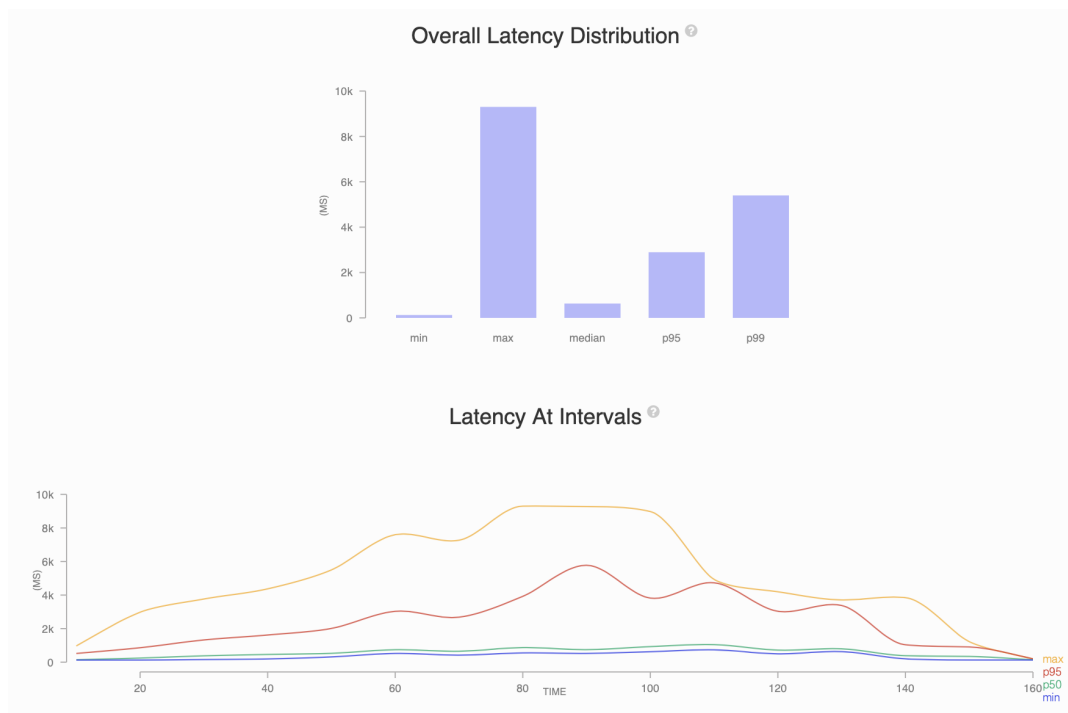


FIGURA 6.3: Resultados de la prueba 3, 120 usuarios

**2,894.4 ms**, mientras que el 99 % lo haría en unos **5,398.8 ms**, llegando incluso algunas llamadas ha tardar mas de 9 segundo.

Así pues, estos resultados harían que en caso de llegar a esta cantidad de usuarios simultáneamente el programa fuera prácticamente inutilizable, ya que pese a que no se produce ningún error y los datos se devuelven correctamente, el tiempo de espera para cada una de las peticiones haría muy difícil la utilización del programa.

#### 6.1.4. Conclusiones:

Tras analizar los resultados de las pruebas realizadas podemos concluir que el programa es perfectamente usable y estable en unas condiciones de uso normales, como las testeadas en la primera prueba, y también lo es para soportar picos improbables de unos 50/60 usuarios conectados simultáneamente.

Los resultados también muestran que en caso de que aumentase considerablemente la capacidad del CUV sería necesario realizar diversas optimizaciones del programa de la base de datos y de la comunicación para reducir este tiempo de espera. No obstante, la capacidad actual del CUV está al límite y,



a no ser que se ampliaran las instalaciones, no sería viable doblar el número de pacientes ni de entidades colaboradoras.

Por ejemplo, una de las optimizaciones que se podrían realizar sería la optimización de las peticiones correspondiente a los listados de datos. En concreto el listado de montajes, ya que la obtención de estos datos genera una petición a la base de datos bastante compleja con múltiples condiciones y uniones. Lo cual provoca que cuando se reciben una gran cantidad de estas peticiones simultáneamente se acabe produciendo un cuello de botella en la base de datos.

Request latency distribution

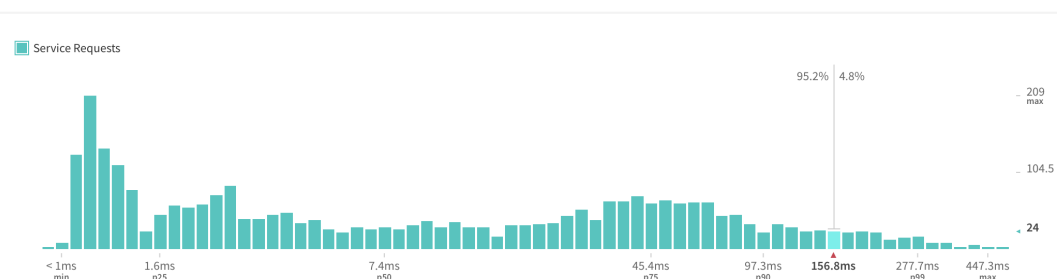


FIGURA 6.4: Distribución de la latencia de las peticiones en el servidor Apollo

Por otro lado, cabe mencionar que gran parte del tiempo de espera de las dos primeras pruebas, viene dado por el tiempo extra de la comunicación. Como podemos observar en la figura 6.4, procedente de las métricas del servidor Apollo, en el caso de la primera prueba el tiempo de respuesta del 95 % de las peticiones se resuelven **156.8 ms** respecto a los **343 ms** que reporta el resultado del script de Artillery.io. No obstante, este no es el caso de la tercera prueba, en la cual el problema viene dada por la complejidad de las peticiones y la comunicación con la base de datos.

Por último, mencionar que las características del servidor de desarrollo, y por ende el de producción, son más que suficiente para un uso normal del programa como podemos ver en las figuras 6.5 y 6.6. En estas figuras podemos ver que el uso de la CPU no ha llegado al 100 % excepto en la última de las pruebas y que el uso de la memoria RAM no ha superado los 750 MB, dejando un margen prudencial hasta el total del servidor (1 GB) para poder soportar picos puntuales de consumo.

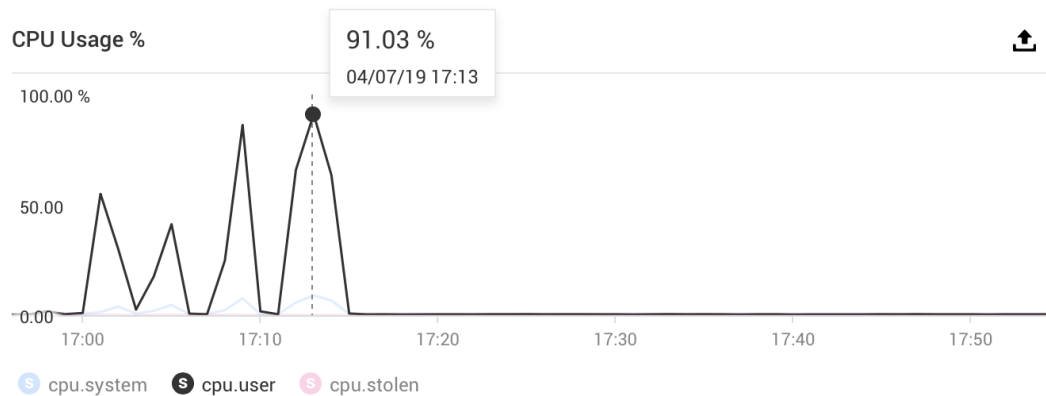


FIGURA 6.5: Uso de CPU durante las pruebas de estrés



FIGURA 6.6: Uso de memoria RAM durante las pruebas de estrés

## 6.2. Requisitos y selección del servidor

Una vez finalizada la prueba de estrés, era momento de extraer unas conclusiones y estudiar cuales eran los requisitos hardware que íbamos a necesitar y que metodología se iba a realizar para que el despliegue fuera lo más sencillo y rápido posible.

Así pues, y ante los resultados observados en el anterior apartado llegamos a la conclusión, de que los requisitos necesarios eran bastante corrientes y no variaban significativamente de los planificados inicialmente. Lo cual, facilitaría toda la gestión que se tendría que realizar con los servicios informáticos de la universidad para conseguir un servidor en el que ejecutar el programa.

Así pues los requisitos necesarios eran los siguientes:

- Sistema operativo Linux, preferiblemente **Ubuntu 18.04 o superior**.

- Unos **2.5GB de almacenamiento**, más el espacio necesario para el SO.
- **1 GB de memoria RAM** como mínimo.
- **Dominio .upc.edu** para acceder al aplicativo.
- **Certificado SSL** del dominio.

Una vez definidos los requisitos mínimos, y con la inestimable colaboración del personal directivo del CUV, se comenzó a realizar todas las gestiones y reuniones necesarias para solicitar un servidor con las características mencionadas o similares.

En este punto era necesario diseñar como se iba a migrar el sistema a este nuevo servidor. Después de estudiar una serie de alternativas, como la elaboración de una guía de instalación o la migración de la imagen completa del servidor de desarrollo al nuevo, se terminó optando por la utilización de Docker, y más concretamente la opción de realizar imágenes Multi-stage que nos permitiesen transferir el programa a este u otros servidores de una manera rápida y sencilla.

Por lo tanto, se comenzó el desarrollo de un contenedor Docker, como se puede ver en detalle en el apartado 5.1, para posteriormente transferírsele a los servicios informáticos de la UPC para su puesta en marcha en el nuevo servidor

### 6.3. Documentación del usuario

Finalmente, junto con el despliegue en el servidor de producción del programa, también fue necesaria la confección de una serie de documentación tanto para los servicios informáticos como para los usuarios del programa.

Así pues, la documentación que se tuvo que realizar fue la siguiente:

- **Manual de uso para trabajadores sociales:** Se tuvo que redactar un manual para que los nuevos trabajadores sociales que se vayan uniendo al programa sepan cuales son las funciones básicas y como utilizarlas. Además también se incluye información relacionada con el registro de nuevos trabajadores de la misma entidad.<sup>1</sup>

---

<sup>1</sup>La última versión de este manual se puede encontrar en: <http://bit.ly/2Gito0i>

- **Manual de uso para personal administrativo, estudiantes y PDI:** Similar al anterior documento, también se tuvo que efectuar otro manual más completo para los usuarios internos del CUV, es decir, el personal de administración, estudiantes y personal docente.<sup>2</sup>
- **Guía de instalación:** Finalmente también se realizó un guía de instalación y configuración detallada destinado a los servicios informáticos del campus de Terrassa de la UPC. No obstante, como se mencionó en el apartado anterior, finalmente se descartó esta alternativa en favor del uso de Docker. Por lo cual, se tuvo que rehacer la guía para simplificarla en gran medida y reflejar los pasos necesarios para la puesta a punto del contenedor Docker.

Una vez terminada la puesta a punto en el nuevo servidor y entregada la documentación necesaria se dio por cerrado el despliegue del programa.

---

<sup>2</sup>La última versión de este manual se puede encontrar en: <http://bit.ly/2VE01tj>

## Capítulo 7

# Plan de contingencia y mantenimiento

En este capítulo veremos los mecanismos que se han diseñado para garantizar la integridad de los datos a lo largo de la vida del programa.

### 7.1. Backups

Con el objetivo de garantizar la integridad de los datos en caso de errores, ha sido necesario diseñar un sistema de copias de seguridad. Este sistema ha sido proporcionado por los sistemas informáticos del campus de Terrassa de la UPC, ya que al proporcionarnos ellos el servidor también son los encargados de la realización de estas copias de seguridad según su normativa.

De esta manera tenemos un sistema de backups periódico de todo el servidor, haciendo una copia total con una periodicidad mensual y copias incrementales diarias, que nos garantiza la accesibilidad de los datos de los últimos 30 días.

Este sistema es más que suficiente para nuestro programa, ya que nos permite tener seguros los datos hasta el día anterior mientras que los del propio día se podrían reconstruir a través de los logs de la base de datos. De esta manera, garantizamos la integridad de los datos vitales para el día a día del CUV sin molestar al usuario con backups constantes durante las horas de más uso del programa.

Una vez garantizado la integridad de los datos pasamos a ver qué planes de contingencia se han diseñado.

## 7.2. Plan de contingencia

Una vez planificado el sistema de backups, pasamos al diseño de los distintos planes de contingencia del aplicativo. Que tienen como objetivo prever y solventar los posibles errores que puedan surgir a lo largo del tiempo.

En nuestro caso encontramos dos planes de contingencia diferentes para distintos fallos.

### 7.2.1. Restauración de backups

Este plan entraría en acción en caso de que se produjese un error leve en la ejecución del aplicativo. El cual provocase que el programa quedase en un estado inestable, o bien que hubiese un fallo en la configuración del servidor, que de nuevo, provocase que no se pudiera usar el aplicativo con normalidad.

La solución que se tendría que llevar a cabo sería una parada del servidor temporal, para revisar toda la configuración y en caso necesario restaurar uno de los backups realizados. De esta manera volveríamos a un punto estable del sistema antes de que se produjera el error inicial.

### 7.2.2. Plan alternativo

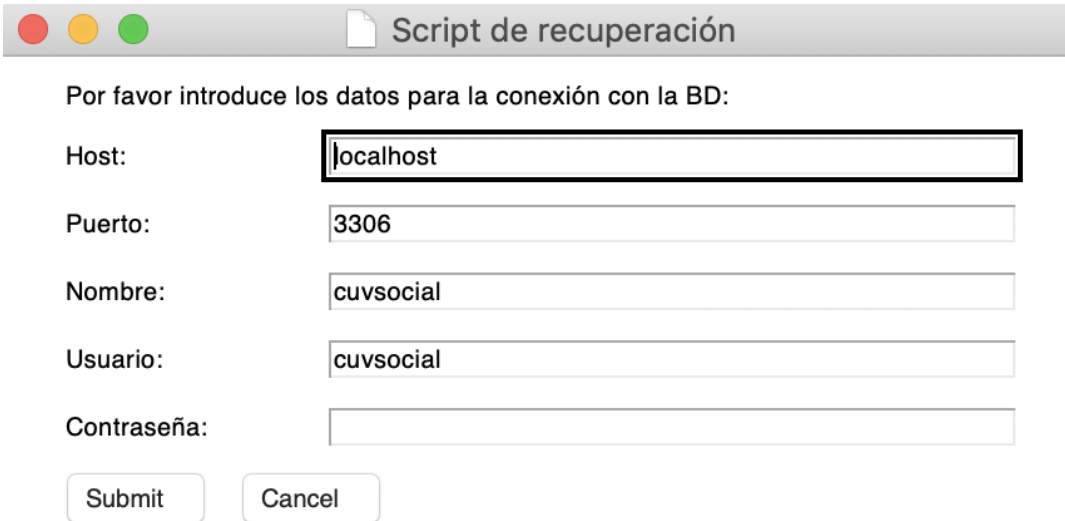
Al contrario que en el plan anterior, este plan entraría en acción en caso de que se produjese o se detectase un error crítico en el programa que fuese un riesgo para la seguridad de los usuarios y de los datos gestionados.

Según la normativa de los servicios informáticos, el servidor se apagaría hasta que se revisara y corrigiera el fallo detectado. Por esta razón, y con el objetivo de permitir al CUV y a los trabajadores sociales seguir trabajando con normalidad, era necesario la creación de un mecanismo alternativo para poder gestionar los datos.

Después de realizar un estudio de las alternativas, se optó por la opción de convertir los datos actuales del programa al formato que utilizaba el CUV con anterioridad, es decir, un excel donde se realizaba toda la gestión de los montajes.

Así pues, para llevar esto a cabo se implementó un script en Python, con una sencilla interfaz gráfica en la cual se solicitan los datos de acceso a la base de datos. Dicha base de datos puede ser la del programa en el momento en que se produjo el fallo, si esta se estaba ejecutando en otro servidor diferente, o bien la base de datos de una de las copias de seguridad.

Una vez introducidos estos datos, el programa se conecta con la base de datos y descarga la información necesaria para acabar generando un archivo excel con el formato que utilizaba el CUV pero con los últimos datos del programa. En la figura 7.1 podemos ver el programa en ejecución.



Por favor introduce los datos para la conexión con la BD:

Host: localhost

Puerto: 3306

Nombre: cuvsocial

Usuario: cuvsocial

Contraseña:

Submit Cancel

FIGURA 7.1: GUI del script de recuperación

Por último también, sería necesario poner una plantilla sencilla en el dominio del programa para informar a los usuarios de que el programa está en mantenimiento y que por favor realicen la comunicación a través de teléfono o email, como se hacía antes del programa.

Con este plan, aunque se espera que no sea nunca necesario, se garantizara que mientras el programa se está revisando, el CUV podrá seguir realizando la gestión de los montajes sin perder todos los datos de estos y realizando el seguimiento de sus pacientes aunque la comunicación se tenga que realizar vía telefónica, a través de email u otras alternativas.

## Capítulo 8

# Sostenibilidad e impacto social

En este capítulo vamos a ver el informe de sostenibilidad del proyecto, además de otras consideraciones como las leyes y regulaciones que se han tenido que cumplir. De esta manera, en la tabla 8.1 encontramos la denominada como matriz de sostenibilidad.

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

CUADRO 8.1: Matriz de sostenibilidad

Dicha matriz se compone de 3 columnas principales correspondientes a las fases de un proyecto, las cuales son: proyecto puesto en producción (PPP), vida útil y finalmente los riesgos que pueden surgir según los ámbitos. Por otro lado, encontramos 3 filas principales correspondientes a las 3 dimensiones a evaluar, concretamente son: Dimensión ambiental, económica y social. A continuación veremos el estudio de sostenibilidad para cada una de estas casillas según la dimensión a la que pertenezcan.

### 8.1. Dimensión ambiental

En el ámbito ambiental, encontramos que el impacto que causa este proyecto es bastante reducido, ya que el consumo energético y de recursos se reducen prácticamente al uso del servidor donde se ejecuta el programa una vez puesto en producción. No obstante, durante el desarrollo del programa también



se han utilizado otros recursos como el dispositivo donde se ha desarrollado el programa y un servidor de prueba.

Así pues, el consumo del equipo de desarrollo se mencionó en el capítulo 3, concretamente se estimó en unos **0.076 kWh**. Por otro lado, el impacto del servidor de producción es difícil de estimar, ya son datos gestionados por la UPC, y de igual manera ha sido imposible encontrar datos del consumo del servidor de pruebas que corresponde a una instancia T2 micro del servicio de Elastic Compute Cloud (Amazon EC2).

No obstante, podemos estimar el consumo basándonos en las características de un servidor similar con 1 core y 1 GB de memoria RAM. Según un estudio sobre el consumo en un entorno de Cloud Computing [47], el consumo de un servidor de estas características en un datacenter puede rondar alrededor de unos **0.04 kWh**.

Cabe resaltar, que el hecho que este servidor se encuentre en un datacenter hace que se reduzca el consumo y la contaminación. Ya que gracias a la escala de estos centros, se consigue optimizar al máximo el uso de recursos, en contra posición de lo que supondría tener un servidor propio separado solo para este programa.

Podemos observar que el impacto ambiental del proyecto es muy reducido, aunque si que es cierto que la huella digital del mismo es previsiblemente un poco superior a la metodología empleada actualmente por el CUV consistente en el uso del teléfono o vía email. No obstante, esta diferencia es prácticamente negligible.

Por otra parte, mencionar que se podría reducir ligeramente el consumo energético así como el coste del servidor, si se realizase una optimización del programa llevado a extremos. Es decir, si el programa se optimizase al máximo, se podría utilizar un servidor con menos recursos que consumiese menos. No obstante el trabajo y coste que supondría para el ahorro nimio que obtendríamos hace rápidamente descartable esta idea.

Por último, podemos observar que el principal riesgo ambiental sería que se necesitase aumentar las características del servidor debido a un aumento del número de usuarios que utilizan el programa. Este hecho provocaría que se necesitase un servidor con más capacidad para dar cabida a estos usuarios y por ende que aumentase el consumo de este.

No obstante el aumento del consumo sería bastante reducido según lo observado en el estudio sobre el consumo en Cloud Computing, ya que con un servidor con más del doble de recursos podría aumentar entre un **0.005 kWh** y un **0.01 kWh**. Y por otro lado, como ya se ha mencionado ante, la capacidad actual del CUV se encuentra limitada por las instalaciones existentes, por lo cual es un riesgo que no se produciría a corto plazo.

## 8.2. Dimensión económica

Respecto a la dimensión económica, nos encontramos que el coste de la realización del proyecto se ha ceñido a lo inicialmente planificado tal y como se detalla en el capítulo 3. No obstante, mencionar a modo de resumen, que como se comenta en el capítulo, pese a que el proyecto ha tenido una duración superior a la estimada, los costes han sido cubiertos por lo estimado en los fondos para imprevistos y contingencia. Así pues, el coste total ha sido **137.7 €** inferior a lo planificado inicialmente.

Además, durante el desarrollo del proyecto se tomaron una serie de decisiones para reducir el coste y la duración del proyecto, como puede ser el cambio del lenguaje de programación del back end el cual inicialmente iba a ser Ruby on Rail pero se terminó optando por el uso de Node JS, hecho que, gracias a la experiencia previa del desarrollador, ha logrado reducir, aun teniendo en cuenta el retraso, la duración que habría supuesto la adopción de Ruby on Rails.

Por otro lado, a la hora de desplegar el servidor se ha optado por realizarlo con Docker para facilitar en gran medida el trabajo a los servicios informáticos de la UPC, y por ende, las horas que se tendrían que haber dedicado para la puesta en producción del programa.

Una vez terminado el desarrollo del programa, y su puesta en producción, tenemos que tener en cuenta el coste que va a suponer durante su vida útil. Así pues, en la tabla 8.2 se detalla los costes anuales que supone el uso de este programa.

De esta manera, el coste anual del programa durante su vida útil sería de unos **485 €**, teniendo en cuenta el coste anual de un servidor junto con el dominio correspondiente, y un servicio de mantenimiento y actualizaciones

<b>Concepto</b>	<b>Precio anual</b>
Servidor	120
Dominio	15
Actualizaciones y mantenimiento	350
<b>Total</b>	<b>485</b>

CUADRO 8.2: Costes anual del programa durante su vida útil (en €)

básico, es decir, que solo cubriría el mantenimiento y solución de bugs pero no la inclusión de nuevas funcionalidades. No obstante también habría que tener presente que podrían haber picos importantes si se detectasen errores críticos que hicieran tener que reprogramar partes importantes o si se desea añadir nuevas características.

Al mismo tiempo, hay que tener presente que existen una serie de riesgos previsibles que podrían hacer incrementar el coste del proyecto como puede ser lo comentado en el párrafo anterior. O el posible aumento de la capacidad del CUV, cosa que en caso que fuese bastante elevado podría producir un mal funcionamiento del servidor como se vio en el apartado 6.1, si se llegase a dar este caso sería necesario ampliar la capacidad del servidor cosa que supondría un coste extra para el proyecto.

De otra manera, también podría darse el caso que se produjese un error crítico que dejase inservible el programa por un tiempo. Por lo que, aun teniendo en cuenta el plan de contingencia previsto, este hecho podría provocar un resentimiento en la productividad del CUV y por consiguiente una reducción en la capacidad paciente que atienden.

Por último, y como se explicó en la fase inicial, el uso de este proyecto reducirá las horas que se tendrían que destinar a tareas rutinarias como responder email, llamadas o poner al día el excel de montajes. De manera que se podrá destinar ese tiempo a tareas más relevantes, que a su vez harán aumentar la productividad del centro y por ende la calidad de vida de las personas que atienden cada día.

### 8.3. Dimensión social

A nivel personal este proyecto me ha permitido descubrir y disfrutar de una nueva forma de trabajar a través de la metodología que emplea el CUV, la metodología de aprendizaje-servicio.

Esta metodología es realmente útil ya que al mismo tiempo permite a los estudiantes conseguir experiencia real en su sector, mientras que también se mejora la calidad de vida de las personas y por consiguiente la calidad de la sociedad que nos rodea. Es por eso que el haber descubierto esta metodología a través de este proyecto me ha abierto la posibilidad para tenerla en cuenta en futuros proyectos y poder ayudar así a más personas que lo necesiten.

Por otro lado, y como se mencionó en la planificación inicial, este programa al haber cumplido todos los objetivos propuestos inicialmente, permitirá mejorar la capacidad de personas sin recursos que podrá atender el CUV, mejorando la calidad de vida de estas.

Además, gracias a la mejor gestión, se abre la puerta a realizar o ampliar también otros proyectos del gran abanico que realizan día a día. Como es el caso de *Mirades Solidaries*, que tiene por objetivo ayudar a niños y niñas que no disponen de recursos, proporcionándoles un tratamiento gratuito, con el objetivo de mejorar su calidad de vida y evitar la exclusión educativa que podrían sufrir por no disponer de los recursos necesarios para el aprendizaje.

Además, mientras que este proyecto mejora en gran medida el impacto que realiza el CUV en la vida de estas personas, cabe decir, que no se perjudica a ningún colectivo ni se destruyen puestos de trabajo, ya que lo que se consigue con este proyecto es mejorar la gestión y la comunicación liberando recursos que luego se podrán emplear en tareas más importantes.

Respecto a los riesgos sociales que podría suponer el proyecto, cabe decir que a priori no se detecta ningún riesgo que pudiese perjudicar a algún colectivo. No obstante, se puede mencionar que podría resultar difícil o más bien lenta la implementación del programa debido a la gran cantidad de entidades que trabajan con el CUV. Así pues, si la implementación se dilata demasiado en el tiempo o bien, si por las razones que sean no se acaba implementando este software, esto podría suponer una ralentización del aumento de la capacidad

del CUV, que indirectamente supondría no poder atender a todos los pacientes que lo necesitan. Por eso se ha de trabajar codo con codo con las entidades para favorecer y facilitar la implementación de este software.

## 8.4. Leyes y regulación

Como prácticamente la mayoría de aplicaciones online, este programa también está regido por el cumplimiento de las leyes y regulaciones en materia de protección de datos y seguridad.

Así pues la ley más importante a la que se ha de dar cumplimiento es la denominada Ley Orgánica de Protección de Datos y Garantía de Derechos Digitales (LOPD-GDD [48]). La cual entró en vigor en el transcurso del desarrollo de este trabajo, concretamente el día 7 de diciembre del 2018

Teniendo en mente esta ley, se ha diseñado e implementado el programa con el objetivo de que el tratamiento de los datos que se tenga que realizar sea de nivel básico, evitando así la implementación de medidas de seguridad y gestión que escapen al ámbito de un Trabajo Final de Grado. Es por esto, que se ha evitado la inclusión de cualquier dato de carácter médico de los pacientes tratados, manteniendo simplemente la fecha de visita, y la gestión de los montajes de gafas que sean necesarios pero sin especificar que características tiene ese montaje, que es información que se encuentra almacenada en otro de los programas que utilizan en el CUV.

Por otro lado para que los trabajadores sociales puedan acceder a la información del seguimiento de sus pacientes, estos han de firmar un documento de confidencialidad con el CUV. Además, los pacientes también firman un documento cuando piden hora a través de su trabajador social, conforme aceptan la cesión de sus datos tanto al CUV como a la entidad desde la que está siendo derivado por el trabajador social que tiene asignado.

La gestión del cumplimiento de las leyes ha sido realizado con la ayuda de las partes interesadas del proyecto, junto con los servicios informáticos y jurídicos de la UPC.

## Capítulo 9

# Conclusiones

En este último capítulo se consideran dos apartados. Por un lado, se considera el cumplimiento de los objetivos y reflexiones del presente trabajo. Por otro lado, se trata el trabajo futuro, así como otras posibles ampliaciones que se podrían realizar.

### 9.1. Cumplimiento de los objetivos y reflexiones

Una vez terminado el proyecto podemos concluir que se han cumplido todos los objetivos fijados. Se ha diseñado y desarrollado el programa con las funcionalidades necesarias, y con el presupuesto definido en la planificación inicial. No así con la duración del mismo, la cual ha sido más larga de lo previsto tal y como se explica en el capítulo 2.

Gracias al desarrollo de esta herramienta se ha conseguido que el CUV pueda mejorar su gestión y facilitar la comunicación con las entidades colaboradoras y sus respectivos trabajadores. Logrando mejorar poco a poco la capacidad de personas a las que pueden atender y por consiguiente su calidad de vida.

Por otro lado, también se ha logrado facilitar la gestión que realizan los estudiantes y los profesores que los supervisan. En esta línea se ha conseguido mejorar el seguimiento de los montajes y la detección de errores durante los montajes.

Gracias a este proyecto he podido consolidar y mejorar los conocimientos adquiridos a lo largo de toda la carrera y descubrir nuevas tecnologías como

Docker, Apollo, GraphQL o Nginx y metodologías como la de aprendizaje-servicio las cuales me han permitido mejorar como profesional y como persona.

Así pues, pese a que CUV Social aún tiene mucho margen de mejora y ampliación, me siento agradecido de haber podido trabajar en un proyecto en el que he aprendido y disfrutado muchísimo. Y que sé que va a permitir impactar en el día a día de cada vez más personas sin recursos que acuden al CUV para mejorar su calidad de vida.

## 9.2. Ampliación y trabajo futuro

Actualmente, podemos decir que el programa se encuentra en una versión estable y lista para comenzar a utilizarse, no obstante como cualquier programa software siempre quedan cosas por realizar, desde la detección y solución de errores hasta la implementación de nuevas características u optimizaciones. Así pues, a continuación se detallan algunos de los pasos que se podrían realizar en un futuro:

- **Optimización y mantenimiento:** Sin duda alguna, a medida que se vaya usando el programa y vaya creciendo el número de usuarios será necesario optimizar los puntos críticos que se vayan detectando y realizar una actualización periódica del software para que no quede obsoleto e inseguro.
- **Mejoras procedentes del feedback:** De manera similar al punto anterior, el programa se deberá ir mejorando con la inestimable ayuda del feedback que nos vayan proporcionando los usuarios, sobre que características incluir o como mejorar las existentes, con el objetivo de que el programa sea cada vez un poco más usable e intuitivo.
- **Nuevas características:** A parte de las recomendaciones que se vayan obteniendo de los usuarios, también se puede ir ampliando poco a poco con nuevas funciones, que vayan simplificando la gestión del CUV.

Una de las nuevas características que se podrían incluir e incluso se comentó en alguna reunión pero se acabó descartando por la limitación

de tiempo del proyecto, sería la inclusión de la gestión de otros proyectos que realicen des de el CUV como es el caso de Mirades Solidaries [49].

Este proyecto por las características similares que comparte con el programa desarrollado, como puede ser la gestión de pacientes, visitas e incidencias, se observó que sería factible implementarlo añadiendo un nuevo modulo o sección al programa actual, sin tener que modificar apenas la parte existente.

De esta manera, y gracias al diseño y desarrollo modular del proyecto sería posible implementar este u otros proyectos para seguir facilitando la gestión del CUV, y ampliar o mejorar así la magnifica labor que hacen día a día para ayudar a personas con dificultades económicas.

- **Open source:** Otro de los posibles pasos que se podrían dar, pasa por realizar una versión estándar y liberar su uso para que otras entidades sin ánimo de lucro sean capaces de mejorar su gestión de manera gratuita, mejorando así su gestión y por consiguiente la calidad de vida de las personas que necesitan su ayuda.



# Bibliografía

- [1] *Centro Universitario de la Visión. CUV — UPC. Universitat Politècnica de Catalunya.* URL: <https://cuv.upc.edu/es> (visitado 19-09-2018).
- [2] *Facultad de Óptica y Optometría de Terrassa. FOOT — UPC. Universitat Politècnica de Catalunya.* URL: <https://foot.upc.edu/es> (visitado 20-09-2018).
- [3] *UPC - UPC Universitat Politècnica de Catalunya.* URL: <https://www.upc.edu/es> (visitado 20-09-2018).
- [4] *Història i Dades de la FOOT — Facultat d'Òptica i Optometria de Terrassa. FOOT — UPC. Universitat Politècnica de Catalunya.* URL: <https://foot.upc.edu/ca/lescola/historia> (visitado 24-09-2018).
- [5] *Acción social — Centro Universitario de la Visión. CUV — UPC. Universitat Politècnica de Catalunya.* URL: <https://cuv.upc.edu/es/servicios/accion-social> (visitado 20-09-2018).
- [6] *Voluntariat en l'àmbit de l'òptica i l'optometria (VOOL)? — Facultat d'Òptica i Optometria de Terrassa. FOOT — UPC. Universitat Politècnica de Catalunya.* URL: <https://foot.upc.edu/ca/iniciatius-solidaries-a-la-foot> (visitado 24-09-2018).
- [7] *Servicios — Centro Universitario de la Visión. CUV — UPC. Universitat Politècnica de Catalunya.* URL: <https://cuv.upc.edu/es/servicios> (visitado 20-09-2018).
- [8] Silvia Barrios Araya y col. *Aprendizaje-servicio como metodología para el desarrollo del pensamiento crítico en educación superior.* Inf. téc. 2012. URL: <http://scielo.sld.cu594>.
- [9] Alexander W Astin y col. *How Service Learning Affects Students.* Inf. téc. 2000. URL: <http://digitalcommons.unomaha.edu/slcehigheredhttp://digitalcommons.unomaha.edu/slcehighered/144>.
- [10] *OpenVision – OPENTIC.* URL: <https://opentic.eu/openmedical/> (visitado 25-09-2018).
- [11] *Igaleno - Programa en la nube para la gestión de clínicas y profesionales médicos.* URL: <https://www.igaleno.com/> (visitado 20-09-2018).

- [12] *DriCloud - Software Medico para Clinicas y Consultorios. Expediente Electronico*. URL: <https://dricloud.com/> (visitado 20-09-2018).
- [13] *Redbooth - Software de Gestión de Proyectos, Actividades y Tareas | Redbooth*. URL: <https://redbooth.com/es/> (visitado 20-09-2018).
- [14] *Wrike - Tus herramientas de software de gestión de proyectos en línea: Wrike*. URL: <https://www.wrike.com/es/> (visitado 20-09-2018).
- [15] *MacBook Pro (15 pulgadas, 2017) - Especificaciones técnicas*. URL: <https://support.apple.com/kb/SP756?viewlocale=es\&locale=pl> (visitado 04-10-2018).
- [16] *Tipos de instancias de Amazon EC2 - Amazon Web Services*. URL: <https://aws.amazon.com/es/ec2/instance-types/> (visitado 12-04-2019).
- [17] *macOS Mojave - Apple (ES)*. URL: <https://www.apple.com/es/macOS/mojave/> (visitado 01-10-2018).
- [18] *Balsamiq. Rapid, effective and fun wireframing software. | Balsamiq*. URL: <https://balsamiq.com/> (visitado 01-10-2018).
- [19] *Sketch - The digital design toolkit*. URL: <https://www.sketchapp.com/> (visitado 01-10-2018).
- [20] *JetBrains: Developer Tools for Professionals and Teams*. URL: <https://www.jetbrains.com/> (visitado 01-10-2018).
- [21] *Overleaf, Online LaTeX Editor*. URL: <https://www.overleaf.com/> (visitado 01-10-2018).
- [22] *Online Gantt Chart Software | TeamGantt*. URL: <https://www.teamgantt.com/> (visitado 01-10-2018).
- [23] *Sketch - Pricing*. URL: <https://www.sketchapp.com/pricing/> (visitado 08-10-2018).
- [24] *For Students: Free Professional Developer Tools by JetBrains*. URL: <https://www.jetbrains.com/student/> (visitado 08-10-2018).
- [25] *Salary: Project Manager in Barcelona, Spain | Glassdoor*. URL: [https://www.glassdoor.com/Salaries/barcelona-project-manager-salary-SRCH\\\_\\\_IL.0,9\\\_\\\_IM1015\\\_\\\_K010,25.htm](https://www.glassdoor.com/Salaries/barcelona-project-manager-salary-SRCH\_\_IL.0,9\_\_IM1015\_\_K010,25.htm) (visitado 11-10-2018).
- [26] *Sueldos en Diseñador/a ux en España | Indeed.es*. URL: <https://www.indeed.es/salaries/Dise\~{n}ador/a-ux-Salaries> (visitado 11-10-2018).
- [27] *Salary: Software Engineer in Barcelona, Spain | Glassdoor*. URL: [https://www.glassdoor.com/Salaries/barcelona-software-engineer-salary-SRCH\\\_\\\_IL.0,9\\\_\\\_IM1015\\\_\\\_K010,27.htm](https://www.glassdoor.com/Salaries/barcelona-software-engineer-salary-SRCH\_\_IL.0,9\_\_IM1015\_\_K010,27.htm) (visitado 11-10-2018).

- 
- [28] *Sueldos en Analista QA en España* | Indeed.es. URL: <https://www.indeed.es/salaries/Analista-QA-Salaries> (visitado 11-10-2018).
- [29] *The world's most popular React UI framework - Material-UI*. URL: <https://material-ui.com/> (visitado 06-10-2018).
- [30] *Design - Material Design*. URL: <https://material.io/design/> (visitado 13-03-2019).
- [31] *Introduction to the DOM - Web APIs* | MDN. URL: <https://developer.mozilla.org/en-US/docs/Web/API/Document%5B%5DObject%5B%5DModel/Introduction> (visitado 14-03-2019).
- [32] *Node.js*. URL: <https://nodejs.org/en/> (visitado 26-11-2018).
- [33] *MySQL*. URL: <https://www.mysql.com/> (visitado 09-10-2018).
- [34] *SQLite*. URL: <https://www.sqlite.org/index.html> (visitado 14-12-2018).
- [35] *PostgreSQL*. URL: <https://www.postgresql.org/> (visitado 22-01-2019).
- [36] *Manual* | Sequelize. URL: <http://docs.sequelizejs.com/> (visitado 15-12-2018).
- [37] *GraphQL* | *A query language for your API*. URL: <https://graphql.org/> (visitado 02-12-2018).
- [38] *Apollo GraphQL*. URL: <https://www.apollographql.com/> (visitado 06-12-2018).
- [39] *NGINX* | *High Performance Load Balancer, Web Server, & Reverse Proxy*. URL: <https://www.nginx.com/> (visitado 20-03-2019).
- [40] *Docker* | *Enterprise Application Container Platform*. URL: <https://www.docker.com/> (visitado 28-03-2019).
- [41] *Moment.js* | *Home*. URL: <https://momentjs.com/> (visitado 20-03-2019).
- [42] *Let's Encrypt - Free SSL/TLS Certificates*. URL: <https://letsencrypt.org/> (visitado 14-02-2019).
- [43] *Certbot*. URL: <https://certbot.eff.org/> (visitado 14-02-2019).
- [44] *NGINX Amplify - Get real-time diagnostics for your apps*. URL: <https://www.nginx.com/products/nginx-amplify/> (visitado 28-03-2019).
- [45] *Apollo Engine* | *Learn about the Apollo platform: Client, Engine, GraphQL servers, GraphQL support, and more*. URL: <https://www.apollographql.com/platform> (visitado 28-03-2019).
- [46] *Artillery - a modern load testing toolkit*. URL: <https://artillery.io/> (visitado 28-03-2019).
- [47] Yan Bai y Haiyang Wang. *Power Consumption of Virtual Machines in Cloud Computing: Measurement and Enhancement* A THESIS SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL OF THE UNIVERSITY OF MINNESOTA BY. Inf. téc. 2016. URL: <https://conservancy.org>.

- umn.edu/bitstream/handle/11299/182708/BAI{\\_}umn{\\_}0130M{\\_}17353.pdf?sequence=1{\&}isAllowed=y.
- [48] *Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales*. URL: <https://www.boe.es/eli/es/lo/2018/12/05/3> (visitado 23-02-2019).
- [49] *Miradas solidarias. Apadrina la salud visual — Centro Universitario de la Visión. CUV — UPC. Universitat Politècnica de Catalunya*. URL: <https://cuv.upc.edu/es/servicios/accion-social/miradas-solidarias-apadrina-la-salud-visual> (visitado 08-10-2018).