



Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA  
Mención en Ingeniería del Software

---

Desarrollo de microservicios y consumo de recursos a través de una aplicación móvil basada en Ionic para un sistema de gestión de alquileres de plazas de aparcamiento particulares por horas

---

Alumno: Pablo Verdejo Santana

Tutor: Yania Crespo González-Carvajal



---

*A mis padres por el apoyo constante*



# Agradecimientos

A mis padres y a mi familia por haberme apoyado y acompañado a lo largo de toda la carrera universitaria.

A mi novia, Nerea, por acompañarme en esos largos días de estudio.

A mi tutora del TFG, Yania, por su dedicación constante e implicación tanto en este proyecto como en la universidad.

A mi compañero de TFG, Carlos, por ofrecerme la posibilidad de realizar este proyecto junto a él y por la ayuda en el proceso.

Al resto de compañeros con los que he podido aprender tanto dentro, como fuera de la carrera.

**Gracias a todos**



# Resumen

El propósito de este Trabajo de Fin de Grado es crear una aplicación web orientada a la gestión de alquileres de plazas de garaje para un uso similar a las plazas de parking de una ciudad.

RentApp es una aplicación web para dispositivos móviles que permite dicha gestión y que, gracias al uso de Ionic para su desarrollo, permite su compilación tanto para Android como para iOS.

Este proyecto ha sido desarrollado utilizando el framework de Ionic para la parte de frontend junto a Spring y Node para el desarrollo de los microservicios que componen el backend.





# Abstract

The purpose of this Final Degree Project is to create a web application oriented towards the management of parking space rentals for a use similar to that of parking spaces in a city.

RentApp is a web application for mobile devices that allows such management and that, thanks to the use of Ionic for its development, allows its compilation for both Android and iOS.

This project has been developed using the Ionic framework for the frontend together with Spring and Node for the development of the microservices that make up the backend.



# Índice general

|   |             |
|---|-------------|
| <b>Agradecimientos</b>                          | <b>III</b>  |
| <b>Resumen</b>                                  | <b>V</b>    |
| <b>Abstract</b>                                 | <b>VII</b>  |
| <b>Lista de figuras</b>                         | <b>XIII</b> |
| <b>Lista de tablas</b>                          | <b>XV</b>   |
| <b>1. Introducción</b>                          | <b>1</b>    |
| 1.1. Contexto . . . . .                         | 1           |
| 1.2. Motivación . . . . .                       | 2           |
| 1.3. Objetivos . . . . .                        | 2           |
| 1.4. Estructura de la memoria . . . . .         | 3           |
| <b>2. Requisitos y Planificación</b>            | <b>5</b>    |
| 2.1. Planificación . . . . .                    | 5           |
| 2.1.1. Metodología Ágil . . . . .               | 5           |
| 2.1.2. Scrum . . . . .                          | 6           |
| 2.2. Definición y análisis de riesgos . . . . . | 9           |
| 2.3. Requisitos . . . . .                       | 15          |

IX

|   |           |
|---|-----------|
| <b>3. Análisis</b>                                  | <b>17</b> |
| 3.1. Modelo del dominio inicial . . . . .           | 17        |
| 3.2. División en microdominios . . . . .            | 18        |
| 3.3. Actividad básica de la aplicación . . . . .    | 19        |
| <b>4. Tecnologías utilizadas</b>                    | <b>21</b> |
| 4.1. Desarrollo del proyecto . . . . .              | 21        |
| 4.1.1. Ionic . . . . .                              | 21        |
| 4.1.2. Node.js . . . . .                            | 22        |
| 4.1.3. Angular . . . . .                            | 22        |
| 4.1.4. Spring Framework . . . . .                   | 22        |
| 4.1.5. SASS . . . . .                               | 24        |
| 4.1.6. jQuery . . . . .                             | 24        |
| 4.1.7. Mapbox . . . . .                             | 24        |
| 4.1.8. Swiper . . . . .                             | 25        |
| 4.1.9. NgCircleProgress . . . . .                   | 26        |
| 4.1.10. CryptoJS . . . . .                          | 27        |
| 4.1.11. MySQL . . . . .                             | 27        |
| 4.2. Gestión y documentación del proyecto . . . . . | 28        |
| 4.2.1. GitLab . . . . .                             | 28        |
| 4.2.2. Telegram . . . . .                           | 29        |
| 4.2.3. Webex . . . . .                              | 31        |
| 4.2.4. Overleaf . . . . .                           | 31        |
| 4.2.5. Astah . . . . .                              | 32        |
| <b>5. Diseño</b>                                    | <b>35</b> |
| 5.1. Modelo Vista Controlador (MVC) . . . . .       | 35        |

|  |           |
|--|-----------|
| 5.1.1. Ejemplos de uso en la aplicación . . . . .                          | 36        |
| 5.2. Arquitectura basada en microservicios . . . . .                       | 37        |
| 5.2.1. Arquitectura Monolítica vs Arquitectura en Microservicios . . . . . | 37        |
| 5.2.2. ¿Por qué usar microservicios? . . . . .                             | 39        |
| 5.2.3. LoginAPI . . . . .  | 41        |
| 5.2.4. EmailAPI . . . . .  | 43        |
| 5.3. Diseño de la interfaz gráfica . . . . .                               | 48        |
| <b>6. Implementación y pruebas</b>   | <b>51</b> |
| 6.1. Implementación . . . . .  | 51        |
| 6.1.1. Distribución de paquetes de Ionic . . . . .                         | 51        |
| 6.1.2. Distribución de paquetes de Spring . . . . .                        | 53        |
| 6.1.3. Distribución de paquetes de Node.js . . . . .                       | 55        |
| 6.2. Pruebas . . . . .   | 56        |
| 6.2.1. Pruebas de Ionic . . . . .  | 56        |
| 6.2.2. Pruebas de LoginAPI . . . . .                                       | 57        |
| 6.2.3. Pruebas de EmailAPI . . . . .                                       | 57        |
| <b>7. Seguimiento del proyecto</b>   | <b>59</b> |
| 7.1. Sprint 1 (23/02/2022 - 01/03/2022) . . . . .                          | 61        |
| 7.2. Sprint 2 (02/03/2022 - 15/03/2022) . . . . .                          | 61        |
| 7.3. Sprint 3 (16/03/2022 - 28/03/2022) . . . . .                          | 63        |
| 7.4. Sprint 4 (29/03/2022 - 12/04/2022) . . . . .                          | 64        |
| 7.5. Sprint 5 (20/04/2022 - 03/05/2022) . . . . .                          | 67        |
| 7.6. Sprint 6 (04/05/2022 - 17/05/2022) . . . . .                          | 68        |
| 7.7. Sprint 7 (18/05/2022 - 31/05/2022) . . . . .                          | 69        |
| 7.8. Sprint 8 (01/06/2022 - 14/06/2022) . . . . .                          | 70        |
| 7.9. Sprint 9 (15/06/2022 - 28/06/2022) . . . . .                          | 71        |

|   |           |
|---|-----------|
| <b>8. Conclusiones</b>  | <b>73</b> |
| 8.1. Cumplimiento de objetivos y adecuación del proyecto y sus herramientas . . . | 73        |
| 8.1.1. ¿Se han cumplido los objetivos? . . . . .                                  | 73        |
| 8.1.2. ¿En que medida ha funcionado la metodología de trabajo elegida? . .        | 74        |
| 8.1.3. ¿Se han adecuado las tecnologías elegidas al trabajo realizado? . . . .    | 74        |
| 8.1.4. ¿Se ha elegido una arquitectura adecuada? . . . . .                        | 75        |
| 8.2. Líneas de trabajo futuras . . . . .  | 75        |
| 8.2.1. Mejoras . . . . .  | 76        |
| 8.2.2. Evolución . . . . .  | 76        |
| 8.2.3. Modelo de negocio . . . . .  | 76        |
| <b>Bibliografía</b>   | <b>77</b> |
| <b>A. Manuales</b>  | <b>79</b> |
| A.1. Manual de despliegue e instalación . . . . .                                 | 79        |
| A.1.1. Frontend . . . . .   | 79        |
| A.1.2. Backend . . . . .  | 80        |
| A.1.3. Base de datos . . . . .  | 81        |
| A.2. Manual de mantenimiento . . . . .  | 81        |
| A.3. Manual de usuario . . . . .  | 81        |
| A.3.1. Inicio de sesión . . . . .   | 82        |
| A.3.2. Registro . . . . .   | 84        |
| A.3.3. El mapa . . . . .  | 87        |
| A.3.4. Plazas de parking y reservas . . . . .                                     | 88        |
| A.3.5. Reservas activas . . . . .   | 91        |
| A.3.6. Perfil de usuario . . . . .  | 92        |
| A.3.7. Creación y gestión de propiedades . . . . .                                | 93        |
| <b>B. Resumen de enlaces adicionales</b>  | <b>97</b> |

# Lista de Figuras

|  |    |
|--|----|
| 2.1. Ejemplo de uso de Kanban para el Sprint 5 . . . . .                                     | 9  |
| 3.1. Diagrama de clases de RentApp . . . . .   | 18 |
| 3.2. Diagrama de actividades de los dos actores principales . . . . .                        | 20 |
| 4.1. Clase main de LoginAPI usando Spring. . . . .   | 23 |
| 4.2. Clase UserRepository de LoginAPI usando Spring Data JPA. . . . .                        | 24 |
| 4.3. Página principal de búsqueda de plazas de parking en RentApp. . . . .                   | 25 |
| 4.4. Página principal de una plaza de parking en RentApp. . . . .                            | 26 |
| 4.5. Componente Circle-Progress representando la valoración de una plaza de parking. . . . . | 27 |
| 4.6. Campo de contraseñas en base de datos con el contenido encriptado. . . . .              | 27 |
| 4.7. Esquemas de las APIs que utilizan base de datos. . . . .                                | 28 |
| 4.8. Grupo del proyecto: TFG Rent. . . . .   | 29 |
| 4.9. Almacenamiento en forma de chat privado de Telegram . . . . .                           | 31 |
| 4.10. Ejemplo de desarrollo en Overleaf . . . . .  | 32 |
| 5.1. Arquitectura MVC en aplicaciones web [9] . . . . .                                      | 36 |
| 5.2. Estructura basada en microservicios de RentApp . . . . .                                | 40 |
| 5.3. Diseño de la API Gateway . . . . .  | 40 |
| 5.4. Diagrama de despliegue . . . . .  | 41 |

|   |    |
|---|----|
| 5.5. Diagrama de clases de LoginAPI . . . . .                                       | 43 |
| 5.6. Funcionamiento de EmailAPI . . . . .   | 44 |
| 5.7. Correo de verificación . . . . .   | 45 |
| 5.8. Correo de cambio de contraseña . . . . .                                       | 46 |
| 5.9. Diagrama de secuencia Envío de mensaje de recuperación de contraseña . . . . . | 47 |
| 6.1. Estructura de directorios de Ionic . . . . .                                   | 53 |
| 6.2. Estructura de directorios de LoginAPI . . . . .                                | 55 |
| 6.3. Estructura de directorios de EmailAPI . . . . .                                | 56 |
| 7.1. Comparación del diseño inicial y final de la pantalla del mapa . . . . .       | 65 |
| A.1. Pantalla de inicio de sesión . . . . .   | 82 |
| A.2. Pantalla de contraseña olvidada . . . . .                                      | 83 |
| A.3. Pantalla de cambio de contraseña . . . . .                                     | 84 |
| A.4. Primera pantalla de registro . . . . .   | 85 |
| A.5. Segunda pantalla de registro . . . . .   | 86 |
| A.6. Pantalla de verificación de usuario . . . . .                                  | 87 |
| A.7. Pantalla del mapa . . . . .  | 88 |
| A.8. Pantalla de información de una plaza . . . . .                                 | 89 |
| A.9. Pantalla de alquiler de una plaza . . . . .                                    | 90 |
| A.10. Pantalla de reservas activas . . . . .  | 91 |
| A.11. Pantalla de perfil de usuario . . . . .                                       | 92 |
| A.12. Pantalla de registrar método de pago . . . . .                                | 93 |
| A.13. Pantalla de lista de propiedades de un propietario . . . . .                  | 94 |
| A.14. Pantalla de añadir una propiedad . . . . .                                    | 95 |



# Lista de Tablas

|   |    |
|---|----|
| 2.1. Significado de los valores numéricos asignados a características de los riesgos. . . . . | 10 |
| 2.2. Riesgo de estimación de tiempo incorrecta. . . . .                                       | 10 |
| 2.3. Riesgo de desconocimiento de las herramientas utilizadas. . . . .                        | 11 |
| 2.4. Riesgo de compatibilidad entre componentes . . . . .                                     | 11 |
| 2.5. Riesgo de caída del servidor de base de datos. . . . .                                   | 12 |
| 2.6. Riesgo de cambios de requisitos del sistema. . . . .                                     | 12 |
| 2.7. Riesgo de rechazo de la política de privacidad por parte de los usuarios. . . . .        | 13 |
| 2.8. Riesgo de que un cliente se quede sin fondos en medio de una reserva. . . . .            | 13 |
| 2.9. Riesgo de realización de ofertas falsificadas por parte de algún usuario. . . . .        | 14 |
| 2.10. Riesgo de identificación de un usuario. . . . .   | 14 |
| 2.11. Riesgo de que un usuario menor intente registrarse en la aplicación. . . . .            | 14 |
| 2.12. Historias de Usuario . . . . .  | 15 |
| 6.1. Batería de pruebas de LoginAPI . . . . .   | 57 |
| 6.2. Batería de pruebas de EmailAPI . . . . .   | 58 |
| 7.1. Historias de usuario . . . . .   | 60 |
| 7.2. Sprint 1 . . . . .   | 61 |
| 7.3. Sprint 2 . . . . .   | 63 |
| 7.4. Sprint 3 . . . . .   | 64 |

|                          |    |
|--------------------------|----|
| 7.5. Sprint 4 . . . . .  | 66 |
| 7.6. Sprint 5 . . . . .  | 68 |
| 7.7. Sprint 6 . . . . .  | 69 |
| 7.8. Sprint 7 . . . . .  | 70 |
| 7.9. Sprint 8 . . . . .  | 71 |
| 7.10. Sprint 9 . . . . . | 72 |

# Capítulo 1

## Introducción

### 1.1. Contexto

Actualmente, la mayoría de los aparcamientos de las ciudades son gestionados por grandes empresas o ayuntamientos, lo que no deja mucho lugar a pequeños propietarios a sacar beneficio con ese modelo de negocio. Ofreciendo a particulares la posibilidad de alquilar por cortos periodos de tiempo entendiéndose como alquileres de menos de un mes, el mercado de las plazas de aparcamiento se liberalizaría, haciendo crecer la oferta en este tipo de servicios.

El uso de RentApp no solo podría traer un aumento en la oferta, sino también en la demanda. Al haber un mayor número de plazas disponibles, existe la posibilidad de que un Cliente encuentre una plaza más acorde a su situación particular. Es decir, con un aumento de plazas de aparcamiento, aumentaría las localizaciones en las que se ubicarían este tipo de servicios, como también aumentaría la variedad de precios disponibles.

Además, gracias a su diseño en microservicios nos encontramos con una aplicación escalable. Esto significa que en el futuro, el proyecto podría llegar a evolucionar de una forma sencilla para abarcar otros ámbitos como pueden ser los bienes raíces.

En resumen, tenemos una aplicación escalable dedicada a la gestión de alquileres, en específico los alquileres de plazas de parking.

El desarrollo del proyecto forma parte de la asignatura: Trabajo de Fin de Grado (Mención Ingeniería de Software). El objetivo de la asignatura es desarrollar un proyecto informático junto con una memoria del proyecto en el que se describan las diferentes características de este. Finalmente, se elaborará y defenderá una presentación pública de todo el trabajo realizado. [13]

### 1.2. Motivación

A lo largo de la carrera universitaria he desarrollado un gusto personal por el desarrollo web. Comenzó en la asignatura de Servicios y Sistemas Web (SSW) en la que se me presentaron las bases del desarrollo web. Sin embargo, en la asignatura de Desarrollo basado en Componentes y Servicios (DBCS) descubrí cómo haciendo un uso de lo aprendido en SSW en conjunto con varios frameworks podían llegar a desarrollarse aplicaciones web de mayor complejidad.

Más adelante, hablando con el otro desarrollador de esta misma aplicación, pensamos que se podía desarrollar la idea principal del proyecto haciendo uso de las tecnologías utilizadas en SSW y DBCS. Sin embargo, si pensamos de forma práctica en la gestión de alquileres de aparcamientos o plazas, no se nos viene a la cabeza una aplicación web, sino una móvil.

Investigando nos encontramos con Ionic, un framework que hacía uso de Angular (herramienta utilizada en DBCS) y de tecnologías web para desarrollar aplicaciones web. La particularidad de Ionic era que estaba orientado a un despliegue de dichas aplicaciones en dispositivos móviles, siendo una de sus ventajas el poder compilar la aplicación tanto para Android como para iOS. Con toda esta información y un diseño de la aplicación orientado a microservicios, teníamos todo lo necesario para comenzar con el desarrollo del proyecto.

### 1.3. Objetivos

El objetivo de este proyecto es diseñar una aplicación móvil orientada a la gestión de plazas de aparcamiento y de garajes, así como su alquiler. Podemos dividir los objetivos principales de la aplicación según los dos tipos de usuarios que harán uso de ella:

- **Arrendadores:** Se define arrendador como la persona que pone un bien en alquiler. En nuestro caso, se trata del propietario de alguna plaza de aparcamiento. El objetivo de la aplicación es permitir tanto a pequeños como a grandes propietarios el alquiler y gestión de las plazas que dispongan. Desde la aplicación se permitirá subir una o más plazas a cada propietario, así como especificar que tipo de uso prefiere. Además de especificar los detalles y características de sus propiedades, también se permite su alquiler, teniendo la posibilidad de recoger los ingresos obtenidos del alquiler de las mismas.
- **Arrendatario:** Es la persona que toma un bien en arrendamiento, es decir, los clientes de las plazas de aparcamiento. El objetivo del proyecto respecto a los arrendatarios es la de ofrecer una aplicación que permita visualizar todas las plazas disponibles para su arrendamiento, y ofrecerle la posibilidad de alquilarlas. Los clientes tendrán así la posibilidad de alquilar una plaza por un periodo de tiempo determinado, o si lo prefieren, alquilar un tipo de plaza Libre que no especifica un horario determinado, sino un precio por minuto. Los clientes dispondrán de interfaces desde las que gestionar los alquileres realizados.

En general, tenemos una aplicación que crea un ecosistema para los alquileres de plazas de aparcamiento particulares, ofreciendo una funcionalidad a los dos principales tipos de actores del ecosistema: arrendadores y arrendatarios.

### 1.4. Estructura de la memoria

Este documento se estructura de la siguiente forma:

**Capítulo 2 Requisitos y planificación:** Describe la metodología de trabajo seguida durante el desarrollo del proyecto, así como las herramientas de trabajo utilizadas. También se describen los riesgos y requisitos identificados.

**Capítulo 3 Análisis** Describe el análisis de la aplicación realizado a partir del cual se identifican los microservicios que la formarán. Se analiza también cómo harán uso de la aplicación ambos tipos de usuario.

**Capítulo 4 Tecnologías utilizadas** Describe las herramientas de las que se ha hecho uso tanto para el desarrollo del proyecto como para su gestión y documentación.

**Capítulo 5 Diseño** Describe el modelo elegido para el diseño de la aplicación, así como su arquitectura y las decisiones de diseño tomadas sobre la interfaz gráfica.

**Capítulo 6 Implementación y pruebas** Describe cómo se han implementado los distintos componentes y servicios. También se describen las pruebas realizadas para IonicRESTClient, LoginAPI y EmailAPI.

**Capítulo 7 Seguimiento del proyecto** Describe los Sprints realizados a lo largo del proyecto y las tareas realizadas en cada Sprint.

**Capítulo 8 Conclusiones** Describe las conclusiones a las que llegamos una vez acabado el proyecto así como las líneas de trabajo futuro.

**Anexo A Manuales:** Incluye manuales de instalación, despliegue, y de uso.

**Anexo B Resumen de enlaces adicionales:** Incluye enlaces de interés sobre el proyecto, como el repositorio de código o páginas utilizadas tanto para el desarrollo como para la memoria.



## Capítulo 2

# Requisitos y Planificación

## 2.1. Planificación

### 2.1.1. Metodología Ágil

Según el manifiesto Ágil, vemos que esta metodología de trabajo se basa en cuatro valores básicos:

- **Individuos e interacciones** sobre procesos y herramientas: Dado a que este proyecto ha sido desarrollado por dos usuarios, la interacción entre ambas partes ha sido esencial. La interconexión entre los componentes que estaban siendo desarrollados de forma paralela y sometidos a cambios constantes requerían de una comunicación prácticamente diaria. A lo largo del progreso, nos vimos obligados a mantener una serie de reuniones en las que debatir y compartir información de ambos desarrollos. Los principales puntos a tratar, tanto en las reuniones establecidas como en las espontáneas, eran las entradas y salidas de los componentes. Es decir, la forma en la que componentes y microservicios iban a comunicarse, y que información requerían para un correcto funcionamiento.
- **Software funcional** sobre documentación legible: A nuestro criterio, el proyecto elegido es bastante ambicioso y requiere de que todos los microservicios funcionen y se comuniquen correctamente. Para ello, debíamos de dedicar la mayor parte de tiempo a la calidad del software y, sobre todo, a la funcionalidad del mismo. Además, varios microservicios debían contener dos tipos de funcionalidades, tanto las del arrendador como las del arrendatario, lo que aumentaba el tiempo de desarrollo de. Es por esto que el foco de atención se ha puesto sobre las distintas funcionalidades y opciones que la aplicación ofrece al usuario final.
- **Colaboración con el cliente** sobre negociación contractual: Ya que la aplicación forma parte del contenido de un proyecto de fin de grado, no existe un cliente real sobre

el que aplicar este principio. Sin embargo, ese papel fue realizado en varias ocasiones por la tutora de este TFG. De este modo, en las reuniones semanales que manteníamos con la tutora se ofrecía nueva funcionalidad para añadir a la aplicación junto con posibles mejoras para la misma. Así conseguíamos de una forma similar el papel de ambos clientes, tanto arrendador como arrendatario.

- **Respuesta al cambio** sobre seguir un plan: Este es uno de los principios que más nos hicieron decantarnos por este tipo de metodología. Como hemos mencionado anteriormente, al ser dos personas las que desarrollan la aplicación hay una gran cantidad de cambios propios, que afectan en alguna medida al otro compañero. Al hacer un desarrollo en microservicios, la alteración de uno de ellos afectaba mínimamente al resto. Sin embargo, a lo largo del progreso de la aplicación se tomaron tanto decisiones en el diseño como en el funcionamiento que tenían una ligera reacción en cadena. Previendo tales casos, decidimos seguir una metodología Ágil, que favorecía el desarrollo de un proyecto con requisitos cambiantes que evolucionaban según las necesidades en un momento concreto del proyecto. Es decir, nos permitía mantener un desarrollo iterativo e incremental. [2]

Tras debatir varios marcos de trabajo, nos decantamos por Scrum.

### 2.1.2. Scrum

Scrum es un marco de trabajo que sigue la metodología Ágil definida en el apartado anterior. Debido a sus características, se hace un marco bastante adecuado para este proyecto en particular, que favorece el trabajo colaborativo. Dentro de Scrum, tenemos distintos roles:

- **Product Owner:** Su labor principal es la de hacer que el valor de los entregables se maximice. Es por esto que suele representar la voz de los clientes o del resto de interesados. También es el encargado de realizar el Product Backlog, el cual definiremos más adelante.

Al ser un TFG, no existen unos clientes reales por lo que se podría considerar que dicho papel lo realiza la tutora de este TFG. Mediante sugerencias y recomendaciones de cambio, obtuvimos lo que podría considerarse como la retroalimentación del cliente del cual carecíamos, adaptándonos así al marco de trabajo Scrum.

- **Desarrolladores:** Se encargan del desarrollo del producto final. Su labor principal es ir incrementando el valor de los entregables Sprint a Sprint. A diferencia de otras metodologías, en Scrum la organización de trabajo la llevan a cabo los propios desarrolladores.

Para este proyecto los desarrolladores fuimos tanto Carlos Noé Muñoz Bastardo como yo, Pablo Verdejo Santana. Sin embargo, debido a la separación de trabajo en microservicios que acordamos antes de comienzo del proyecto, los microservicios que se tratarán en este TFG han sido desarrollados íntegramente por Pablo Verdejo Santana. También forma parte de mi desarrollo el diseño e implementación del frontend (IonicREST-Client). En resumen, los desarrollos que conciernen a este TFG son tanto la API de



autenticación (LoginAPI) como la API de envío de correos electrónicos (EmailAPI), al igual que toda la parte frontend del proyecto (IonicRESTClient). De esta forma, el resto de APIs que aparecen descritas han sido desarrolladas por Carlos Noé Muñoz Bastardo.

- **Scrum Master:** Su función es la de asegurarse de que se cumplen todas las reglas de este marco de trabajo, así como del cumplimiento de los objetivos especificados en las reuniones. Es decir, lleva un seguimiento del trabajo de los desarrolladores y debe asegurarse de que la evolución del producto va por el buen camino y no se retrasa o desvía del objetivo principal.

De nuevo, este rol fue interpretado por la tutora de este TFG. Sus funciones como Scrum Master fueron las de mantener un seguimiento del trabajo desarrollado a lo largo del Sprint. En dicho seguimiento, se aseguraba de que el proyecto seguía el ciclo planificado originalmente y del cumplimiento de los objetivos establecidos en los Sprint Retrospective. En estas reuniones se realizaba una revisión del estado del Sprint y se comprobaba si se habían cumplido todas las tareas que incluía. A su vez, se planificaban las futuras tareas del siguiente Sprint. De este modo, se calculaba el estado y evolución global del proyecto.

### Flujo de trabajo

En cuanto al flujo de trabajo, el proyecto se divide en Sprints. Un Sprint es un periodo de tiempo específico, que normalmente suele acotarse entre una semana y un mes, en el que se especifican unas horas de trabajo por Sprint. Para este proyecto decidimos llevar a cabo Sprints de dos semanas, trabajando un total de 35 horas por sprint. En cada Sprint suelen llevarse a diario un Daily Scrum de unos quince minutos, reuniones diarias en las que se comenta el progreso del Sprint y se inspecciona si se va a ser capaz de llegar al objetivo establecido para el Sprint.

Para este proyecto y teniendo en cuenta el tamaño del equipo, consideramos que no tenía mucho sentido llevar a cabo reuniones diarias. En vez de eso, decidimos mantener conversaciones por chat, y hacer videoconferencias para los temas de diseño más complicados. Es decir, se acordaba en conjunto un diseño específico en una reunión y se desarrollaba dicho diseño a lo largo del Sprint, manteniendo una comunicación prácticamente diaria por chat para resolver alguna duda específica. La justificación de esta forma de trabajo se basa en que ninguno de los dos desarrolladores teníamos un horario de trabajo fijo, por lo que no podíamos establecer una hora para el Daily Scrum.

Sin embargo, hay otro tipo de reuniones que si que realizábamos periódicamente que es el Sprint Retrospective. Esta reunión se realiza al final de cada Sprint junto con el Scrum Master con el objetivo de mostrar el trabajo realizado a lo largo del Sprint. En ella se reflejan los aspectos más relevantes por los que se ha ido pasando durante el Sprint, hablando sobre lo que ha ido bien, los problemas que aparecieron y que podría mejorarse para el próximo Sprint. En estas reuniones también establecíamos objetivos para el siguiente Sprint. Esta reunión puede parecerse a un Sprint Review, reuniones que se realizan al final de cada Sprint junto con los interesados del proyecto en las que se presenta el trabajo realizado a modo

demostración. En estas reuniones también se recibe una retroalimentación sobre el trabajo realizado y recomendaciones para el trabajo venidero en el próximo Sprint.

A pesar de que llevábamos una Sprint Retrospective al final de cada Sprint, a la mitad del Sprint realizábamos una reunión de duración similar (una hora aproximadamente) en la que comentábamos el avance del Sprint y consultábamos dudas. Del mismo modo, el Scrum Master realizaba una pequeña retroalimentación del trabajo terminado a mitad del Sprint.

### Herramientas de trabajo

Para la organización de los Sprints, utilizamos tableros Kanban, junto con la metodología de Scrum mencionada anteriormente. A esta forma de trabajo que adapta Scrum al método de gestión de trabajo Kanban se le conoce como Scrumban. Es considerado como una forma de gestión Ágil ya que comparte las características Ágiles mencionadas anteriormente. Kanban se basa en cuatro principios:

- **Empieza con lo que haces ahora:** Cuando se implementan otras metodologías de trabajo a un proyecto, dichas metodologías traen consigo cambios necesarios para adaptarse a la nueva forma de trabajo. Dichos cambios conllevan riesgos de adaptación ya que obliga a adaptarse a los nuevos cambios, con los costes que acarrearán. Sin embargo, Kanban trabaja de una forma contraria a este tipo de metodologías. Con Kanban, se respetan los procesos, roles y formas de trabajo en el proyecto. El objetivo es mantener los puntos fuertes del proyecto, intentando solucionar los puntos más débiles. Dichos cambios se harán de forma progresiva, lo que implica un bajo coste y en consecuencia, un bajo riesgo.
- **Busca e implementa cambios progresivos y evolutivos:** Realizar cambios evolutivos y progresivos frente a grandes cambios tiene una serie de ventajas. En primer lugar, los costes de dichos cambios serán mucho menores al ser más pequeños. En segundo lugar, a raíz de realizar cambios a pequeña escala podemos volver atrás en el caso de que una nueva implementación no haya tenido el resultado esperado. Esta facilidad para solucionar los problemas que puedan surgir reduce en gran medida los riesgos acarreados por los cambios.
- **Respetar los procesos, roles y responsabilidades actuales:** Kanban no especifica ni roles ni procesos específicos que obliguen a cambiar la forma de trabajo original. Es decir, la estructura y procesos del equipo pueden mantenerse de la misma forma a la vez que se implementa Kanban.
- **Impulsa el liderazgo en todos los niveles:** Las decisiones o ideas sobre los cambios deben de poder venir de cualquier miembro del equipo. La idea es incentivar todo tipo de ideas e iniciativas de trabajo.

Cuando aplicamos esta metodología de trabajo, intentamos ilustrar el estado de trabajo de una iteración apoyándonos en un tablero visual. De esta forma, podemos ver el estado del Sprint de una forma gráfica. El objetivo es que se reflejen los procesos del flujo de trabajo

en dicho tablero. Como podemos ver en la Figura 2.1, se dividía el tablero en 4 columnas distintas.

- **Open:** Al inicio de un Sprint se establecen todas las tareas que hay que llevar a cabo a lo largo de ese Sprint. Dichas tareas que tan solo han sido planificadas se sitúan en la columna 'Open'.
- **In progress:** En esta columna se situarán todas las tareas que estén en un proceso activo de desarrollo. En el momento en el que se empieza a desarrollar una de las tareas que estaba en la columna 'Open', debe moverse a 'In progress'. Al mismo tiempo, si vamos a realizar modificaciones en alguna de las tareas de la columna 'Review', también deberemos de moverla a la columna 'In progress' en lo que duren los cambios.
- **Review:** A esta columna pertenecen todas las tareas que ya han sido terminadas y están pendientes de revisión por parte del Scrum Master.
- **Closed:** Al final del Sprint el Scrum Master revisará las tareas que se encuentren en estado 'Review'. En caso de que las tareas hayan sido acabadas y cumplan con la funcionalidad prometida, se marcarán como 'Closed'. En caso contrario, se terminarán de desarrollar en el siguiente Sprint.

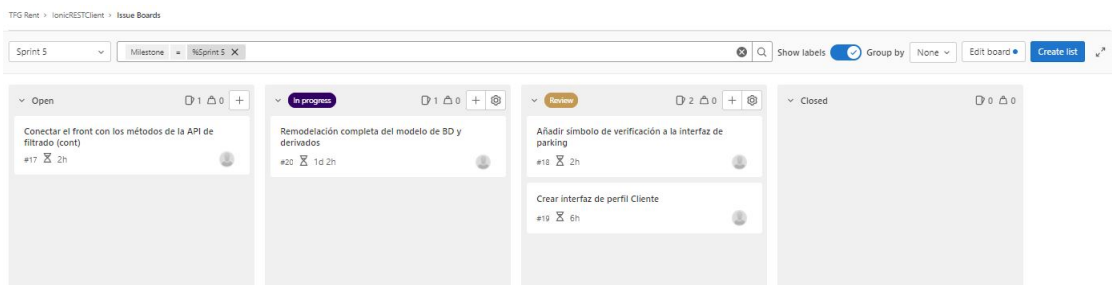


Figura 2.1: Ejemplo de uso de Kanban para el Sprint 5

## 2.2. Definición y análisis de riesgos

En esta sección enumeramos una serie de riesgos que podrían afectar al correcto desarrollo del proyecto. Para presentar los riesgos, de los describirá y se les asignará una serie de características:

- **Riesgo:** Descripción general del riesgo.
- **Probabilidad:** Valor numérico indicando la facilidad con la que el riesgo podría ocurrir.
- **Impacto:** Valor numérico indicando el daño que dicho riesgo tendría sobre el proyecto.

## 2.2. DEFINICIÓN Y ANÁLISIS DE RIESGOS

---

- **Plan de Prevención:** Son planes que se aplican antes de que el riesgo ocurra con el objetivo de reducir las probabilidades de que este se materialice.
- **Plan de Contingencia:** Son planes que se ejecutan cuando el riesgo ya se ha materializado cuyo objetivo es reducir el impacto sobre el proyecto.

A continuación, podemos ver en la Tabla 2.1 el significado de la numeración que se aplicará para valorar tanto la probabilidad del riesgo como su impacto.

| Valor | Probabilidades de Ocurrencia | Impacto        |
|-------|------------------------------|----------------|
| 1     | Muy bajo                     | Muy leve       |
| 2     | Bajo                         | Leve           |
| 3     | Medio                        | Medio          |
| 4     | Alto                         | No muy crítico |
| 5     | Muy alto                     | Crítico        |

Tabla 2.1: Significado de los valores numéricos asignados a características de los riesgos.

A continuación se procede a enumerar los diferentes riesgos que se han identificado para este proyecto específico con todas sus características.

Este riesgo se ha materializado de forma habitual a lo largo del desarrollo del proyecto. Las causas principales por las que se ha llegado a materializar han sido el desconocimiento y la poca familiarización inicial con las tareas del proyecto. A pesar de estar utilizando una tecnología conocida, había muchas nuevas herramientas con las cuales no había tratado las cuales retrasaron ciertos desarrollos. Además, para la estimación de tiempo de una tarea se dejaba un pequeño margen de error en el que se incluían posibles errores que pudiesen surgir. Sin embargo, este margen no siempre era suficiente y hubo ocasiones en las que fue superior llevando a emplear un tiempo mayor al estimado.

|                             |  |
|-----------------------------|--|
| <b>Riesgo</b>               | Estimación incorrecta del tiempo que requiere una tarea.   |
| <b>Probabilidad</b>         | 4  |
| <b>Impacto</b>              | 3  |
| <b>Plan de Prevención</b>   | Basar las estimaciones de tiempo en estimaciones de proyectos pasados similares y estudiar la realización de tareas detenidamente. |
| <b>Plan de Contingencia</b> | Dedicación de horas extra al Sprint y en el peor de los casos realizar una reformulación de la división de tareas.                 |

Tabla 2.2: Riesgo de estimación de tiempo incorrecta.

Como hemos visto en la descripción del riesgo anterior, el uso de nuevas tecnologías y herramientas tenían una gran posibilidad de causar retrasos en el desarrollo. A pesar de aplicar tanto los planes de prevención como los de contingencia, hubo ocasiones en los que el riesgo acabó apareciendo y causando retrasos. El plan de contingencia aplicado a este riesgo consiguió reducir el impacto en gran medida. Sin embargo, había desarrollos para los que la documentación existente era bastante escasa lo cual complicaba la solución del problema.

|                             |  |
|-----------------------------|--|
| <b>Riesgo</b>               | Desconocimiento de nuevas herramientas utilizadas durante el desarrollo que retrasen el progreso.  |
| <b>Probabilidad</b>         | 4  |
| <b>Impacto</b>              | 2  |
| <b>Plan de Prevención</b>   | Realizar una planificación de las actividades teniendo en cuenta un periodo de tiempo aceptable para el aprendizaje de las nuevas herramientas utilizadas. |
| <b>Plan de Contingencia</b> | Búsqueda en la documentación entre otros lugares para resolver las dudas lo antes posibles. Preguntar a alguien experimentado en la herramienta.           |

Tabla 2.3: Riesgo de desconocimiento de las herramientas utilizadas.

Este TFG está realizado sobre un proyecto común en el que participamos dos personas. El trabajo principal del otro desarrollador era crear una serie de APIs para los diferentes microservicios identificados para este proyecto. A lo largo del desarrollo de la parte frontend he ido recogiendo entradas de datos las cuales tenía que conectar a las APIs de mi compañero. Para solucionar los riesgos de realizar una mala conexión y acabar pasando datos equivocados, seguimos el plan de prevención especificado: mantener una buena comunicación. Las reuniones en las que debatíamos sobre el diseño de las APIs y diseñábamos las interfaces que conectarían ambas partes: frontend y backend.

Aunque a medida que avanzábamos en el proyecto hubo algunos cambios en requisitos que obligaron a cambiar dichas interfaces. La solución fue reunirnos con la mayor rapidez posible para solucionar el problema cuanto antes y reducir el impacto sobre el progreso del Sprint.

|                             |   |
|-----------------------------|---|
| <b>Riesgo</b>               | Errores de comunicación entre mis componentes y los componentes de mi compañero a la hora de integrarlos juntos en la misma aplicación. |
| <b>Probabilidad</b>         | 3   |
| <b>Impacto</b>              | 4   |
| <b>Plan de Prevención</b>   | Buena comunicación acompañado de un buen diseño junto a interfaces antes de empezar el desarrollo de las APIs.                          |
| <b>Plan de Contingencia</b> | Refactorización del código y revisión del diseño de los componentes.  |

Tabla 2.4: Riesgo de compatibilidad entre componentes

Como bien estimamos a la hora de identificar los riesgos, el problema de una caída del servidor de datos era muy poco probable. Por suerte, la base de datos se mantuvo operativa a lo largo de todo el desarrollo por lo que no tuvimos ningún problema al respecto. No obstante, después de realizar cualquier cambio significativo en la base de datos se procedía a realizar una copia de seguridad.

## 2.2. DEFINICIÓN Y ANÁLISIS DE RIESGOS

---

|                             |   |
|-----------------------------|---|
| <b>Riesgo</b>               | Caída del servidor de base de datos.  |
| <b>Probabilidad</b>         | 1   |
| <b>Impacto</b>              | 5   |
| <b>Plan de Prevención</b>   | Mantener copias de seguridad en varios dispositivos y realizar dichas copias periódicamente para tener los datos actualizados.                      |
| <b>Plan de Contingencia</b> | Esperar a que la base de datos vuelva a estar operativa y realizar una copia de seguridad en cuanto sea posible por si se repitiesen los problemas. |

Tabla 2.5: Riesgo de caída del servidor de base de datos.

A lo largo del proyecto ha habido más cambios en los requisitos de los estimados inicialmente. El problema con estos cambios de requisitos es que conllevan un trabajo de adaptación tras de sí. Aunque a pesar de haberse materializado en más ocasiones de las esperadas, el impacto de dichos cambios si que ha sido el estimado al principio. Gracias al diseño del código y al uso de buenas prácticas a la hora de diseñar las interfaces y modelos, el impacto producido ha sido el esperado y los cambios han podido aplicarse sin mayor problema.

|                             |   |
|-----------------------------|---|
| <b>Riesgo</b>               | Cambios repentinos de requisitos del sistema.   |
| <b>Probabilidad</b>         | 2   |
| <b>Impacto</b>              | 2   |
| <b>Plan de Prevención</b>   | Asegurarse que los requisitos del sistema están bien definidos.   |
| <b>Plan de Contingencia</b> | Reunión con mi compañero para redefinir los requisitos de manera que puedan ser implementados lo mejor posible. |

Tabla 2.6: Riesgo de cambios de requisitos del sistema.

El riesgo que vemos en la Tabla 2.7 no ha podido ser probado. Esto se debe a que la aplicación desarrollada tan solo ha sido probada por los desarrolladores y no se ha sacado al mercado. En caso de que la aplicación pasase a estar disponible al público, debería de hacerse una encuesta sobre los aspectos de privacidad de la plataforma. A pesar de que actualmente no podemos realizar dicha encuesta, la aplicación sí que dispone de una política de privacidad visible al usuario a la hora de registrarse, política la cual deberá de aceptar en caso de querer hacer uso de la aplicación.

|                             |   |
|-----------------------------|---|
| <b>Riesgo</b>               | Rechazo por parte de los usuarios respecto a la parte de privacidad relacionada con el microservicio de la ubicación del que dispone la aplicación.           |
| <b>Probabilidad</b>         | 3   |
| <b>Impacto</b>              | 1   |
| <b>Plan de Prevención</b>   | Avisar de los permisos que va a requerir la aplicación y de el tratamiento que se realizarán a los datos. Intentar tomar la menor cantidad de datos posibles. |
| <b>Plan de Contingencia</b> | Esclarecer cualquier tipo de duda respecto a los datos del usuario de los que dispone la aplicación.  |

Tabla 2.7: Riesgo de rechazo de la política de privacidad por parte de los usuarios.

En caso de que un cliente se quede sin fondos en medio de una reserva, el procedimiento a aplicar no es otro que el de una plaza de parking convencional. El algoritmo de los costes requiere una fianza, dinero con el cual se podría amortiguar un uso indebido de la plaza. Sin embargo, en caso de que dicha fianza no cubra el costo total de los problemas ocasionados, deberá de intentar solucionar el problema con las autoridades competentes.

|                             |  |
|-----------------------------|--|
| <b>Riesgo</b>               | Un cliente se quede sin fondos en la tarjeta en medio de una reserva.  |
| <b>Probabilidad</b>         | 3  |
| <b>Impacto</b>              | 2  |
| <b>Plan de Prevención</b>   | Antes de pagar la reserva se obtendrá una fianza en proporción al coste total de la misma.                     |
| <b>Plan de Contingencia</b> | Se cobrará la fianza depositada anteriormente y se le propondrán soluciones para solventar el dinero restante. |

Tabla 2.8: Riesgo de que un cliente se quede sin fondos en medio de una reserva.

A pesar de que para reservar una plaza hace falta dejar una fianza, es posible que algún usuario pudiese hacer un uso indebido de la aplicación realizando reservas falsas, o cometiendo faltas habituales en el tiempo de reserva de una propiedad. En estos casos, es trabajo del administrador del sistema anular dicho perfil para impedir su uso de la aplicación. Con la anulación de la cuenta, también se podrán anotar los datos personales del cliente para pasarle a una “lista negra” la cual impida un nuevo registro.

## 2.2. DEFINICIÓN Y ANÁLISIS DE RIESGOS

|                             |   |
|-----------------------------|---|
| <b>Riesgo</b>               | Aparición de usuarios problemáticos que realicen ofertas falsificadas.  |
| <b>Probabilidad</b>         | 3   |
| <b>Impacto</b>              | 5   |
| <b>Plan de Prevención</b>   | Realizar un sistema de registro en profundidad para poder denunciar este tipo de actos. Crear un perfil de administrador que pueda eliminar estos perfiles. |
| <b>Plan de Contingencia</b> | Denunciar al sujeto y banear su usuario por correo o móvil para complicar el que vuelva a registrarse.  |

Tabla 2.9: Riesgo de realización de ofertas falsificadas por parte de algún usuario.

Uno de los problemas más comunes a la hora de disponer de un inicio de sesión en una aplicación es que al usuario se le olvide la contraseña. Ya que el envío de una contraseña mediante correo electrónico no es una práctica segura, se ha decidido por ofrecerle un cambio de contraseña. Al usuario se le enviará un link único por correo en el cual podrá realizar un cambio de contraseña.

|                             |  |
|-----------------------------|--|
| <b>Riesgo</b>               | Problema de identificación de un usuario.  |
| <b>Probabilidad</b>         | 2  |
| <b>Impacto</b>              | 3  |
| <b>Plan de Prevención</b>   | Establecer métodos para que se pueda recuperar la contraseña.  |
| <b>Plan de Contingencia</b> | Permitir la identificación temporal proporcionando otros datos distintos hasta que recupere la contraseña. |

Tabla 2.10: Riesgo de identificación de un usuario.

A pesar de que la aplicación no permite el registro de un menor de edad en la aplicación, es común ver a personas menores utilizando aplicaciones con estos requisitos. En el caso de que se identificase un uso de la aplicación por parte de un menor de edad se procedería a la eliminación total de su cuenta. Además, se procederá a poner las credenciales de la cuenta en una “lista negra” para evitar que el usuario vuelva a intentar un registro.

|                             |  |
|-----------------------------|--|
| <b>Riesgo</b>               | El registro sea efectuado por un menor de edad por lo que podría realizar pagos siendo menor.    |
| <b>Probabilidad</b>         | 3  |
| <b>Impacto</b>              | 2  |
| <b>Plan de Prevención</b>   | Establecer métodos de registro en la aplicación que aseguren la mayoría de edad de los usuarios. |
| <b>Plan de Contingencia</b> | Proceder a la anulación de la cuenta.  |

Tabla 2.11: Riesgo de que un usuario menor intente registrarse en la aplicación.



## 2.3. Requisitos

En Scrum, las historias de usuario representan las características que debe de tener el proyecto a desarrollar. Al principio del proyecto, se detallan unas historias de usuario con el objetivo de capturar la funcionalidad final deseada del proyecto. Una de las características que tienen es que puede usarse un lenguaje coloquial, ya que suelen surgir de conversaciones con el Cliente del proyecto.

Para este proyecto, se han definido 12 historias de usuario para el proyecto global. De esas 12 historias de usuario, 2 de ellas son específicas del desarrollo del otro desarrollador del proyecto. En la Tabla 2.12

| Historia de Usuario  | Desarrollo    |
|--|---------------|
| Como usuario quiero poder identificarme para acceder a mi cuenta personal  | Pablo / Común |
| Como usuario quiero poder registrarme  |               |
| Como usuario quiero poder recuperar mi contraseña  |               |
| Como usuario quiero poder ver mis datos personales y editar los más relevantes                                       |               |
| Como arrendatario quiero poder filtrar las plazas disponibles para encontrar la que más se ajuste a mis preferencias |               |
| Como arrendatario quiero poder ver toda la información de una plaza  |               |
| Como arrendatario quiero poder ver las plazas disponibles a mi alrededor   |               |
| Como arrendatario quiero poder realizar un alquiler  |               |
| Como arrendatario quiero poder ver el estado de mi reserva   |               |
| Como arrendador quiero poder alquilar bienes   |               |
| Como arrendatario se usara el sistema de Arduino para la apertura de puertas   | Carlos Noé    |
| Como usuario, quiero tener seguridad y privacidad cuando este realizando un pago.                                    |               |

Tabla 2.12: Historias de Usuario



## Capítulo 3

# Análisis

### 3.1. Modelo del dominio inicial

En el primer Sprint, ambos desarrolladores del proyecto mantuvimos varias reuniones para crear un modelo del dominio del cual partir. El resultado final de las reuniones es el diagrama de clases de la Figura 3.1. En el diseño inicial de la aplicación se consideraron más características y funcionalidades de las que abarcaría este TFG, como puede ser la sección de comentarios.

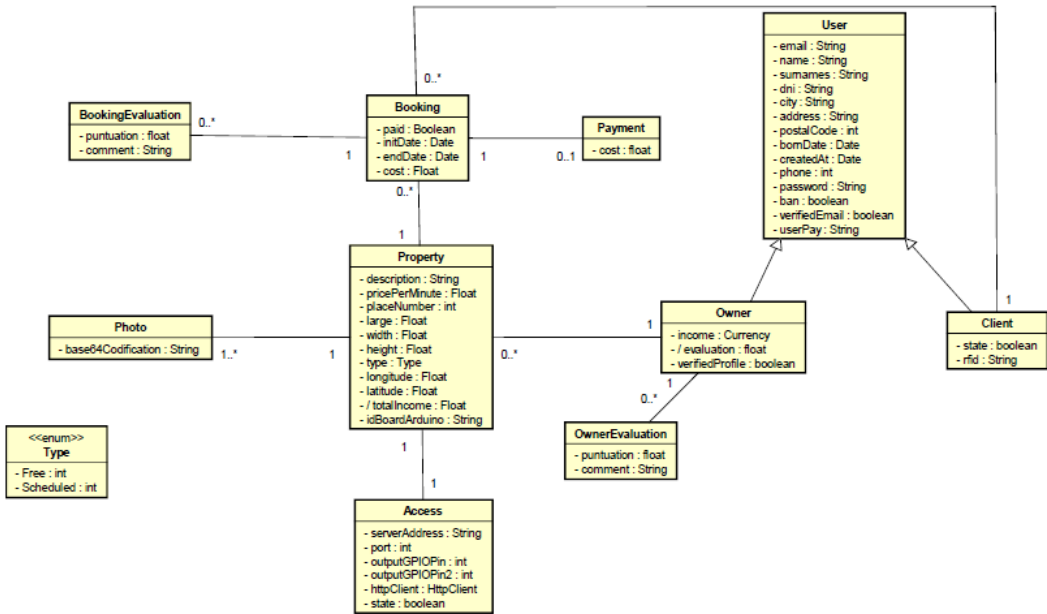


Figura 3.1: Diagrama de clases de RentApp

### 3.2. División en microdominios

A partir del diagrama anterior se extrajeron servicios que pudiesen ser implementados de forma autónoma y que permitiesen encapsular funcionalidades específicas. De esta forma, obtenemos servicios especializados en desempeñar funcionalidades particulares.

La idea detrás de la descomposición en microservicios se explicará más adelante, pero en resumen, consiste en crear componentes enfocados en resolver problemas específicos. Estos microservicios se desarrollan de forma independiente y se comunican a través de APIs.

Tras un estudio del modelo de dominio y de los servicios de los que dispondría la aplicación, se identificaron 4 servicios básicos:

- **Servicio de autenticación (LoginAPI):** En este servicio se almacenan los datos relativos a los usuarios. También se llevan a cabo los servicios de registro, autenticación y modificación de los datos de los usuarios.
- **Servicio de envío de correos electrónicos (EmailAPI):** Esta API implementa el envío de correos electrónicos a los usuarios registrados en la aplicación. Actualmente se envían dos tipos de correos: verificación de usuario y cambio de contraseña.
- **Servicio de alquileres (AlquileresAPI):** Trata toda la funcionalidad relativa a los alquileres de plazas. Dentro se encuentra el modelo de plazas de parking, así como el

de las reservas. Es el encargado de gestionar todas las operaciones que se llevan a cabo en la reserva de una plaza.

- **Servicio de pagos (PagosAPI):** Gestiona todos los aspectos relativos a pagos. Gestiona desde el registro de cuentas y tarjetas bancarias hasta la conexión con una API externa para la tramitación de pagos.

De estos 4 microdominios y un cliente frontend (IonicRESTClient) se compone la aplicación. La idea final con los microservicios es que sean dockerizados y desplegados en un servidor de forma independiente.

### 3.3. Actividad básica de la aplicación

El objetivo principal de la aplicación es el de gestionar los alquileres de plazas de garaje que un arrendador pone en alquiler y un arrendatario alquila. A partir de este flujo básico se crean el resto de casos de uso y requisitos de la aplicación.

En la Figura 3.2 podemos ver el diagrama de actividades de los dos actores principales. La función del arrendador (propietario) es la de registrar nuevas propiedades en la aplicación. La función del arrendatario (cliente) es buscar una plaza que se ajuste a sus requisitos y alquilarla.

Como puede apreciarse en el diagrama, no cualquier cliente puede realizar un alquiler. Son necesarias dos condiciones. En primer lugar, tiene que haber plazas disponibles. Esto significa que, a parte de estar registradas, no deben de estar ocupadas por otro cliente. La segunda condición es que el cliente haya registrado un método de pago. Dado a que en el registro de un usuario de tipo cliente no es necesario introducir un método de pago, es posible que este no lo haya registrado. En este caso, deberá de acceder a su perfil y acceder a la interfaz de registrar un método de pago. Posteriormente, el cliente podrá alquilar las plazas disponibles.

### 3.3. ACTIVIDAD BÁSICA DE LA APLICACIÓN

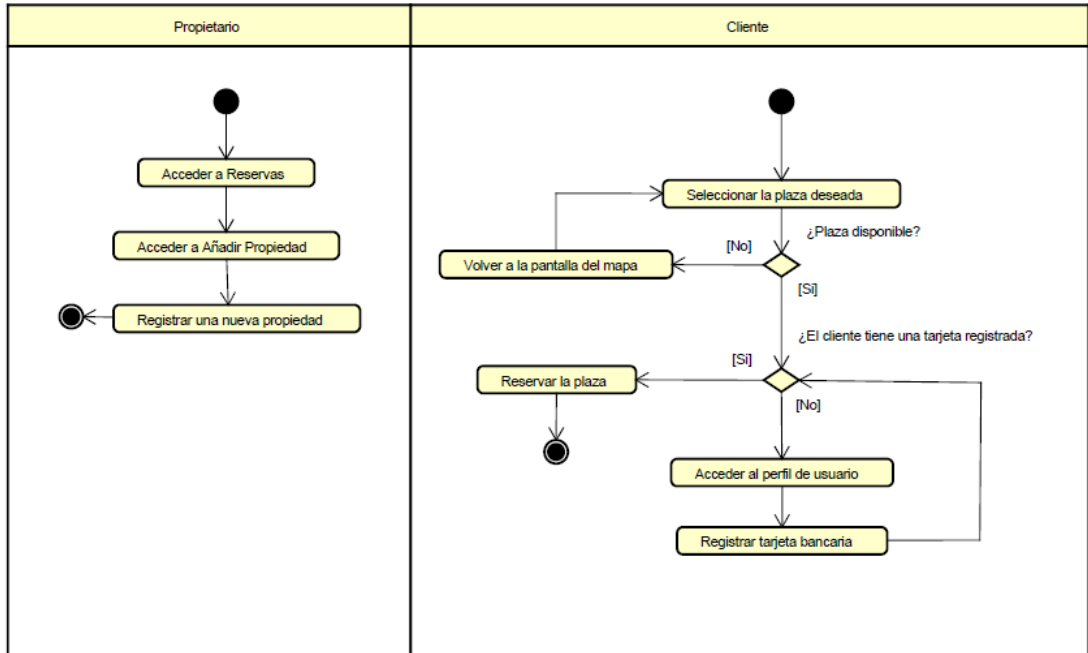


Figura 3.2: Diagrama de actividades de los dos actores principales

## Capítulo 4

# Tecnologías utilizadas

En este Capítulo se presenta un resumen de las tecnologías utilizadas tanto para la gestión del proyecto, como para el desarrollo.

### 4.1. Desarrollo del proyecto

A continuación se enumeran una lista de tecnologías y herramientas utilizadas para el desarrollo del proyecto. Las tecnologías indicadas han sido utilizadas para el diseño, programación, compilación y despliegue de la aplicación.

#### 4.1.1. Ionic

Ionic es la herramienta principal utilizada en el desarrollo de la aplicación. Como se puede ver en la documentación, se trata de un kit de desarrollo software (SDK) de código libre (<https://github.com/ionic-team/ionic-framework>). Dicho framework permite la integración junto con otras librerías o frameworks, como pueden ser Angular, React o Vue. En esta aplicación, utilizaremos Angular.

Una de las razones por las que se decidió utilizar Ionic es que se desarrollará una aplicación que podrá desplegarse en varias plataformas con un solo código base. Esto significa que podremos desplegar la aplicación tanto para Android como para iOS, pudiéndose ejecutar también como una Aplicación Web Progresiva. Para un despliegue como aplicaciones móviles (Android o iOS) tendríamos que hacer uso de entornos de desarrollo de aplicaciones móviles. Desde la web de Ionic se indica que mediante Capacitor o Cordova podremos realizar despliegues nativos con dicho fin.

La otra razón que favoreció la elección de Ionic como framework de desarrollo fue que permitía una integración junto a Angular. Debido al uso previo de Angular; lo cual quitaría un

tiempo de aprendizaje considerable, y a su modularidad que permitiría realizar la aplicación separada en componentes, se eligió Angular como framework de desarrollo junto con Ionic. [6]

### 4.1.2. Node.js

Node.js es un entorno de ejecución JavaScript, multiplataforma que se encarga de ejecutar código JavaScript de forma independiente al buscador web. Esta herramienta funcionaba muy bien con la aplicación web que iba a ser desarrollada debido a las operaciones web. Node.js trabaja con HTTP, lo que ofrece un soporte para operaciones que agiliza el trabajo con frameworks web, como es el caso de Ionic.

Gracias a su arquitectura dirigida por eventos, Node.js optimizaba la escalabilidad en las operaciones de entrada y salida con Ionic de forma asíncrona. Como se ha mencionado anteriormente, este tipo de entornos favorece el uso de aplicaciones web en tiempo real.

### 4.1.3. Angular

Angular es un framework construido en TypeScript de código libre (<https://github.com/angular/angular>) que se utiliza para crear aplicaciones web de una sola página. La elección de Angular como uno de los frameworks que se utilizarán para desarrollar la aplicación se debe a dos razones.

La primera es que, como se mencionó previamente, disponía de experiencia trabajando con esta herramienta, lo cual reduciría considerablemente el tiempo en empezar a desarrollar la aplicación.

La segunda razón que impulsó un desarrollo en Angular fue su componentización. Los componentes en Angular son los bloques que forman la aplicación. Queríamos un diseño lo más modularizado posible y el uso de los componentes de Angular ayudaría a conseguir nuestro objetivo. A su vez, existen multitud de componentes y librerías creadas por usuarios y de uso libre. [1]

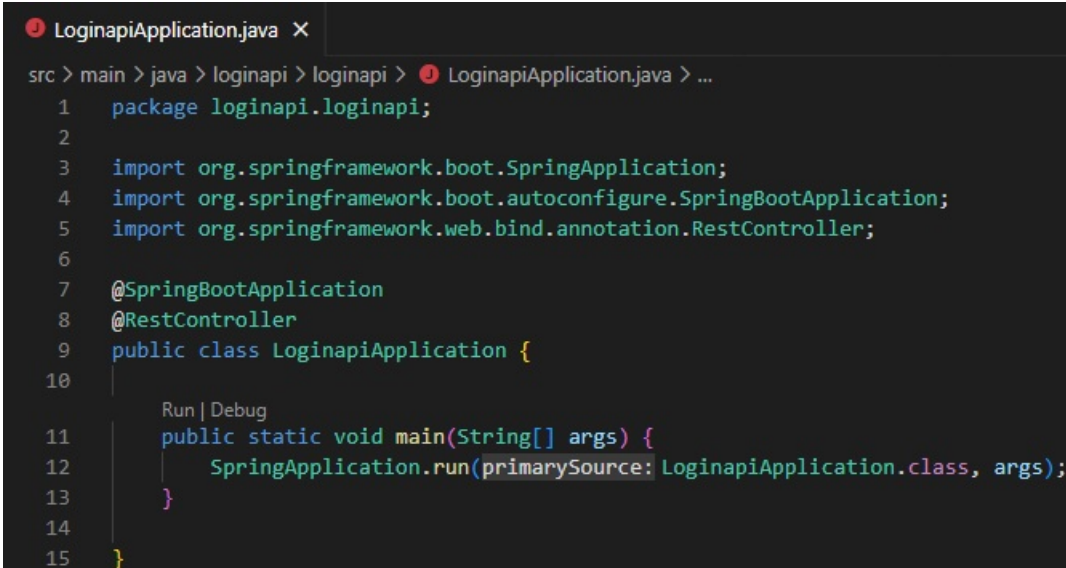
### 4.1.4. Spring Framework

Se trata de un framework orientado a crear aplicaciones Java de una forma más comprensible en cualquier plataforma de desarrollo. En estos últimos años, el uso de Spring junto con aplicaciones con una arquitectura basada en microservicios se ha popularizado bastante. Esto se debe a podremos iterar sobre los microservicios desarrollados de forma rápida, reduciendo a su vez el espacio ocupado. Es decir, disponemos de la posibilidad de desarrollar un microservicio de una forma ágil, que cuando se encuentre finalizado podrá ser empaquetado en un archivo JAR, listo para trabajar con él.



En la aplicación, Spring se ha utilizado para desarrollar el único microservicio existente por parte de este TFG, el servicio de Login. LoginAPI cuenta con la funcionalidad principal orientada a los usuarios. Es decir, dentro se encuentran las operaciones (registro, login...) y los modelos de los distintos usuarios que participan en el sistema.

Como podemos observar en la Figura 4.1, creamos una clase principal (LoginapiApplication) que se encargará de ejecutar todos los servicios necesarios para que spring se ponga en marcha, permitiendo así el uso de toda la funcionalidad definida en el resto de la API. [11]

A screenshot of an IDE window titled 'LoginapiApplication.java'. The code is as follows:

```
src > main > java > loginapi > loginapi > LoginapiApplication.java > ...
1  package loginapi.loginapi;
2
3  import org.springframework.boot.SpringApplication;
4  import org.springframework.boot.autoconfigure.SpringBootApplication;
5  import org.springframework.web.bind.annotation.RestController;
6
7  @SpringBootApplication
8  @RestController
9  public class LoginapiApplication {
10
11     Run | Debug
12     public static void main(String[] args) {
13         SpringApplication.run(primarySource: LoginapiApplication.class, args);
14     }
15 }
```

Figura 4.1: Clase main de LoginAPI usando Spring.

### Spring Data JPA

JPA se trata de una API que describe el funcionamiento de datos relacionales en aplicaciones Java. El objetivo de esta API es la de tratar con objetos POJO (Plain Old Java Object), sigla utilizada para designar clases simples e independientes en Java.

Dentro de todas las herramientas que Spring nos ofrece, nos encontramos con algunas orientadas al acceso de datos. Ese es el caso de Spring Data JPA. Su función es la de facilitar el acceso a repositorios JPA, simplificando las peticiones y mejorando el acceso a base de datos. Para utilizar Spring Data JPA el desarrollador tan solo tiene que crear las interfaces correspondientes con el modelo y con los métodos que se utilizarán para acceder a los datos y Spring se encargará del resto (implementación de los métodos) automáticamente.

En LoginAPI, se usa JPA para acceder y realizar consultas a la base de datos que contiene los usuarios registrados en la aplicación. Como podemos ver en la Figura 4.2, mediante una interfaz que utilice el repositorio JPA (JpaRepository) podemos definir métodos de consulta a la base de datos.

```
UserRepository.java X
src > main > java > loginapi > loginapi > Repository > UserRepository.java > ...
 1 package loginapi.loginapi.Repository;
 2 import loginapi.loginapi.Model.User;
 3 import java.util.List;
 4 import java.util.Optional;
 5
 6 import org.springframework.data.jpa.repository.JpaRepository;
 7
 8 public interface UserRepository extends JpaRepository<User, String>{
 9
10     List<User> findAll();
11
12     Optional<User> findByEmailAndPassword(String email, String password);
13
14     Optional<User> findByEmail(String email);
15 }
```

Figura 4.2: Clase UserRepository de LoginAPI usando Spring Data JPA.

### 4.1.5. SASS

SASS es un lenguaje de estilos. Esto significa que es un lenguaje que se utiliza para el diseño y presentación de documentos o páginas. SASS se compila a lenguaje CSS, el cual es otro lenguaje de estilos. La diferencia entre SASS y CSS es que SASS dispone de variables, funciones entre otras muchas herramientas que facilitan y estructuran el diseño de las plantillas web. [10]

### 4.1.6. jQuery

jQuery es una librería de JavaScript que contiene características y herramientas para trabajar con HTML, manejo de eventos, Ajax entre otras muchas funcionalidades. Su objetivo es ofrecer una API fácil de usar en multitud de entornos.

En varios de los componentes de nuestra aplicación Ionic se hace uso de jQuery, principalmente para trabajar con la interfaz, utilizando los métodos para la manipulación de HTML y eventos de los que dispone. Para poder utilizar jQuery, tuvimos que instalar la librería mediante el gestor de paquetes: NPM. [8]

### 4.1.7. Mapbox

Mapbox es un proveedor de mapas en línea, el cual dispone de una gran variedad de servicios (navegación, ubicación...) y mapas (satélite, relieve...). Para la aplicación Ionic se

ha utilizado Mapbox GLJS, una librería JavaScript que ofrece mapas vectoriales en la web.

Mapbox GLJS se ha utilizado para la presentación de plazas de parking en una ubicación en concreto. De esta forma permitimos al usuario de la aplicación ver todas las plazas disponibles con sus características en un mapa interactivo por el que poder navegar. Es así como podrá elegir la plaza más adecuada en su situación.

Como podemos ver en la Figura 4.3, se presenta un mapa con todas las plazas disponibles en la zona centro de Valladolid. Si pulsamos a una plaza, podemos ver una pequeña descripción de sus características y un botón que nos permitiría acceder a una página con toda la información de la plaza seleccionada. [7]

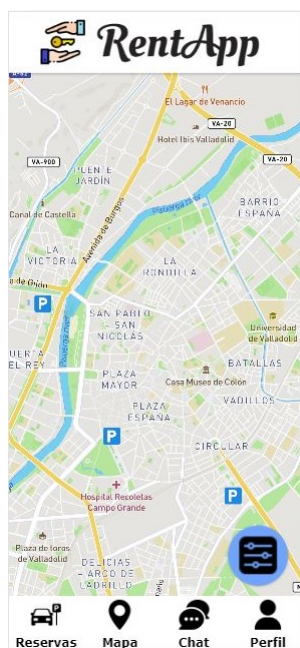


Figura 4.3: Página principal de búsqueda de plazas de parking en RentApp.

### 4.1.8. Swiper

Swiper es un deslizador táctil (slider) gratuito. Siendo el principal componente deslizador de Ionic, es también parte Framework7, un framework para construir aplicaciones Android e iOS.

En la aplicación hacemos uso de Swiper en la página que representa una plaza de parking particular, con todos sus datos. Específicamente, se utiliza para mostrar todas las fotos de una plaza en concreto. La elección de utilizar este método para mostrar las imágenes es que permitía mostrar una cantidad de fotos ilimitada, ocupando un espacio en específico. De esta forma, el resto del tamaño de la página no dependía de la cantidad de fotografías asociadas

a la plaza. En la Figura 4.4 podemos ver el componente en la aplicación. [12]



Figura 4.4: Página principal de una plaza de parking en RentApp.

### 4.1.9. NgCircleProgress

Se trata de un círculo que representa el progreso en forma de porcentaje. Es un componente público de Angular que se basa en gráficos SVG. Este componente permite el cambio de varios aspectos tanto visuales como funcionales.

En la aplicación se utiliza este componente para mostrar de una forma más visual, tanto la valoración de la plaza de parking como la valoración del propietario al que pertenece dicha plaza. En la Figura 4.5 podemos ver un ejemplo de como utilizamos el círculo de progreso para representar la valoración de la plaza de parking. [3]



Figura 4.5: Componente Circle-Progress representando la valoración de una plaza de parking.

### 4.1.10. CryptoJS

Se trata de una implementación en JavaScript de distintas funciones y algoritmos de criptografía. En la aplicación se utilizan para cifrar cadenas de texto. Sin embargo, la mayor utilidad que se le ha dado a esta librería es el implementar un algoritmo que guardase las contraseñas cifradas en base de datos. Antiguamente las contraseñas eran guardadas como texto plano, lo cual era muy susceptible de ser visto con un simple vistazo a base de datos. Ahora las contraseñas se guardan siguiendo un cifrado como podemos ver en la Figura 4.6 de forma que viendo la base de datos tan solo se ve una cadena alfanumérica. [4]

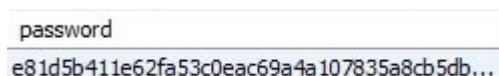


Figura 4.6: Campo de contraseñas en base de datos con el contenido encriptado.

### 4.1.11. MySQL

Se trata del sistema de gestión de base de datos que se ha elegido para almacenar el contenido del proyecto. El modelo de MySQL es relacional, lo que significa que está basado en el uso de relaciones entre los registros de datos. Las relaciones suelen establecerse mediante un vínculo que se forma cuando los registros tienen un campo en común, al que suele denominarse clave (ID). Entre las ventajas de las que dispone este tipo de modelado están la de evitar duplicidades entre los datos almacenados, garantizar que en la eliminación de un registro se eliminen todos los registros dependientes y favorecer la normalización.

Para este proyecto se creó una base de datos común la cual se almacenó en un servidor de la UVA. Debido a que cada uno de los desarrolladores teníamos que crear APIs independientes, creamos un esquema por API en la base de datos. Los esquemas pueden apreciarse en la Figura 4.7 están asociados cada uno a su API correspondiente. La razón por la que la

aplicación no comparte un esquema común es debido al diseño en microservicios, que hace que cada microservicio tenga que ser independiente del resto. Cabe destacar que no todas las APIs utilizadas en la aplicación hacen uso de la base de datos, es por esto que tan solo hay tres esquemas relacionados con las APIs: Access (AlquileresAPI), Payments (PagosAPI) y Users (LoginAPI). Existe un cuarto esquema llamado sys el cual se genera por defecto al crear la base de datos, en el que se almacena información relativa a la propia base de datos.

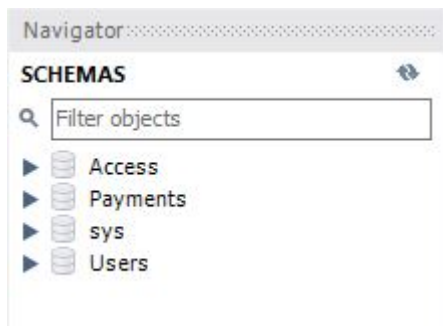


Figura 4.7: Esquemas de las APIs que utilizan base de datos.

## 4.2. Gestión y documentación del proyecto

A continuación se enumeran las herramientas y aplicaciones utilizadas a lo largo del proyecto para su gestión. Entre ellas encontramos herramientas para la planificación y seguimiento del proyecto, a la vez que aplicaciones para la comunicación entre los participantes del desarrollo.

### 4.2.1. GitLab

GitLab es un proveedor de software que proporciona un controlador de versiones basado en Git. Git es un software de control de versiones orientado a que se pueda mantener un registro y control de los cambios que se realizan en un proyecto. De esta forma, varias personas pueden trabajar en un proyecto de forma segura sin miedo a que un cambio erróneo pueda estropear el desarrollo. Han habido tres razones por las que se ha decidido utilizar un controlador de versiones, en específico, GitLab.

En primer lugar, el uso de GitLab de forma particular se veía indispensable. A lo largo del desarrollo del proyecto, se realizan una gran cantidad de cambios que pueden afectar a partes del código anteriores. Gracias al uso de un controlador de versiones podemos mantener un historial de cambios. Al mismo tiempo, tenemos una copia de seguridad del proyecto en la nube (los servidores de GitLab) en el que ver cambios previos y diferentes versiones del proyecto. Si juntamos todo esto, nos damos cuenta de que podemos reducir riesgos relativos a la integridad del proyecto a un bajo coste.

En segundo lugar, mantener repositorios comunes entre los desarrolladores del proyecto favorece la comunicación y conexión entre ambos desarrollos. En GitLab, disponíamos de un grupo en el que se guardaban todos los repositorios del proyecto. Esto permitía que pudiese obtener las últimas versiones de las APIs creadas por mi compañero para integrarlas en mi desarrollo. Además se agilizaba la comunicación ya que en cualquier momento podía consultar el desarrollo de mi compañero y obtener la última versión de cualquiera de sus APIs. Mantener estos niveles de conexión en un proyecto software es muy recomendable, siendo una de las prácticas más comunes en los proyectos software. En la Figura 4.8 podemos ver el grupo de GitLab en el cual alojamos todos los repositorios del proyecto.

The screenshot shows the GitLab group page for 'TFG Rent'. At the top, there is a group header with the name 'TFG Rent', a lock icon, and the Group ID: 1226. There are buttons for 'New subgroup' and 'New project'. Below the header, there are statistics: 'Recent activity Last 90 days', 'Merge Requests created 35', 'Issues created 55', and 'Members added 0'. The main content area is titled 'Subgroups and projects' and contains a list of projects with their names, icons, star counts, and update times.

| Project Name      | Star Count | Last Updated |
|-------------------|------------|--------------|
| AlquileresAPI     | 0          | 1 hour ago   |
| IonicRESTClient   | 0          | 19 hours ago |
| PagosAPI          | 0          | 1 week ago   |
| CompositeRent     | 0          | 1 month ago  |
| LoginAPI          | 0          | 3 weeks ago  |
| ArduinoRESTClient | 0          | 1 month ago  |
| Documentación     | 0          | 1 month ago  |
| EmailAPI          | 0          | 2 months ago |

Figura 4.8: Grupo del proyecto: TFG Rent.

Finalmente, GitLab dispone de funcionalidad orientada a la planificación de proyectos. Como se ha mencionado anteriormente, este TFG ha seguido una metodología de trabajo Ágil. Para ello, hemos utilizado marcos de trabajo como son Scrum y Kanban. La gestión relativa a Kanban se realizó íntegramente haciendo uso de las herramientas de tableros de GitLab. Otro tipo de funcionalidad que ofrecía GitLab era el control de tareas. Para la evolución del proyecto se fueron creando Sprints, los cuales contenían tareas a desarrollar antes de la finalización del Sprint.

### 4.2.2. Telegram

Telegram es una aplicación gratuita de mensajería instantánea que destaca por su privacidad y transparencia en su funcionamiento. La aplicación dispone de varias funcionalidades,

aunque la gestión de este proyecto se utilizaron tres funcionalidades en concreto:

- **Chat privado:** Se trata de una conversación entre dos particulares. En mi caso, lo utilizaba para mantener conversaciones con el otro desarrollador del proyecto. En dichas conversaciones cada participante daba un feedback diario de los avances de su parte. A su vez, se aprovechaba también para especificar que tareas requerían de una prioridad mayor.

Otro de los usos que le dábamos a las conversaciones privadas era la de compartir información que el otro desarrollador iba a necesitar en un futuro a la hora de programar cierta funcionalidad. Es decir, para que ambos desarrolladores pudiéramos seguir con nuestro trabajo independientemente del avance de la otra parte del proyecto, lo que se hacía eran compartir interfaces. De esta forma, podíamos saber que entradas (parámetros) iban a requerir las funciones de la API en desarrollo y las salidas que tendrían. Esta forma de trabajo agilizó bastante el proceso de codificación.

Por último, las conversaciones también servían para establecer reuniones de trabajo en las que tratar temas que afectasen a la estructura general del proyecto.

- **Grupos:** Los grupos eran conversaciones de chat de 2 o más personas en las que todos los participantes podían hablar. En este proyecto se creó un grupo de Telegram de tres participantes en el cual nos encontrábamos tanto los dos desarrolladores como la tutora. En dicho grupo se establecían las reuniones semanales que tomaban lugar a la mitad y al final de un Sprint.

Otro uso que le fue dado al grupo fue el de compartir información de interés para el desarrollo del proyecto. En general, solían compartirse APIs, herramientas y programas que podrían resultar de utilidad para la realización de alguna tarea particular. Al mismo tiempo, también se aprovechaba para preguntar dudas específicas y ofrecer posibles soluciones a problemas que surgieron a lo largo del diseño y codificación.

- **Almacenamiento:** Telegram te permite guardar mensajes y ficheros en una conversación privada contigo mismo. Esto funciona gracias a que al enviar un mensaje o fichero a dicha conversación, este se almacena en la nube, pudiendo acceder a el mediante cualquier dispositivo en el que el usuario se autentificase con la misma cuenta con la que le archivo o mensaje fue enviado.

En mi caso, la función principal de esta herramienta era la de mantener una copia de seguridad tanto de partes del proyecto como de ficheros auxiliares para la realización de la memoria. De esta forma, en caso de que mi dispositivo fallase, siempre podría recuperar la última copia almacenada en mi cuenta de Telegram. En la Figura 4.9 se muestra dicho chat privado en el que se pueden apreciar dos ficheros almacenados: una copia de seguridad de esta misma memoria y una copia de seguridad del Excel que utilicé para mantener un registro de los tiempos de desarrollo de cada tarea realizada.



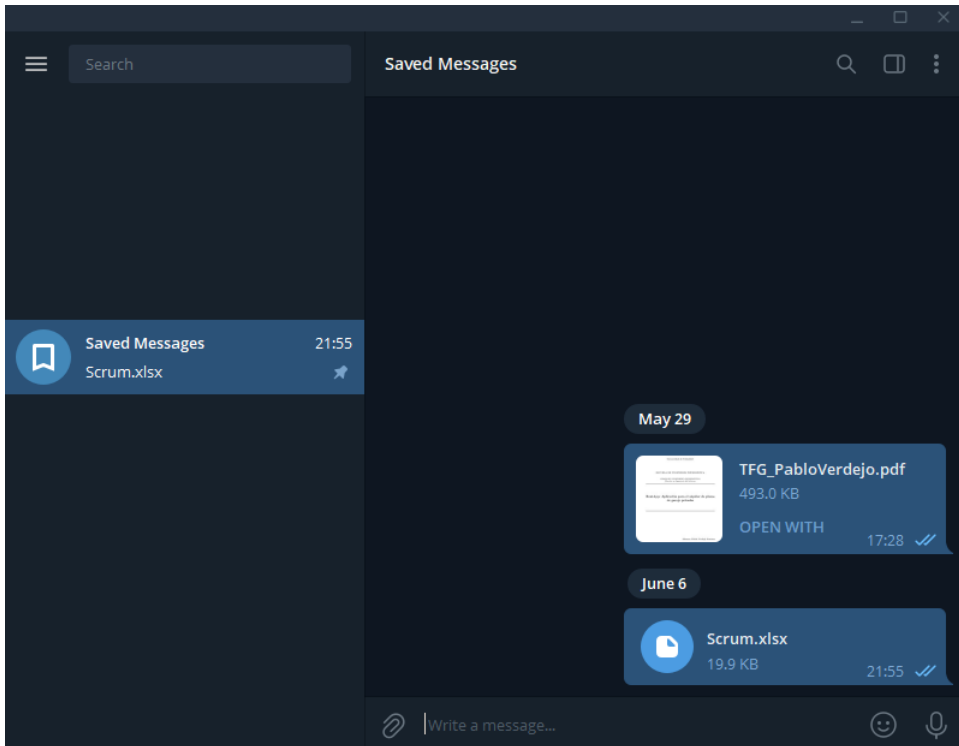


Figura 4.9: Almacenamiento en forma de chat privado de Telegram

### 4.2.3. Webex

Webex es una aplicación que permite realizar videoconferencias entre varios participantes. Para realizar videoconferencias solo necesitaremos estar registrados en la aplicación. De esta forma, podremos usarla ya sea en nuestro navegador web o descargando una versión para nuestro dispositivo.

A lo largo de este proyecto hemos realizado reuniones semanales. A mitad de Sprint, realizábamos una reunión en la que comentábamos el estado actual del Sprint y se resolvían posibles dudas que hubiesen surgido en el desarrollo. En las reuniones a final de Sprint, se daba un feedback del proyecto a final de Sprint y se planificaba el siguiente Sprint. Todas estas reuniones se llevaron a cabo en Webex.

### 4.2.4. Overleaf

Para redactar esta memoria sobre el TFG, se utilizó Overleaf. Overleaf es un editor de texto de LaTeX colaborativo que trabaja en la nube. Dicho editor suele utilizarse para

desarrollos grupales ya que varios usuarios pueden editar el mismo fichero al mismo tiempo. Sin embargo, en este caso ha sido utilizado de forma independiente.

Como ya se ha mencionado, Overleaf es un editor de texto de LaTeX. LaTeX es un sistema software orientado a la preparación de documentos. A diferencia de otros editores de texto, LaTeX dispone de etiquetado y funciones que permiten personalizar completamente el documento que se está desarrollando. De entre todas las funcionalidades de las que dispone, las que más se han utilizado en este documento es la inserción de imágenes, referencias a secciones y referencias bibliográficas. Cuando se ha escrito el documento al completo, tan solo hay que compilarle y descargarlo para obtener la versión final en un formato PDF.

En la Figura podemos ver una parte de la memoria que se está desarrollando actualmente. En ella se aprecia la inserción y referencia de una imagen, referencias bibliográficas y la descripción de una sección del documento.

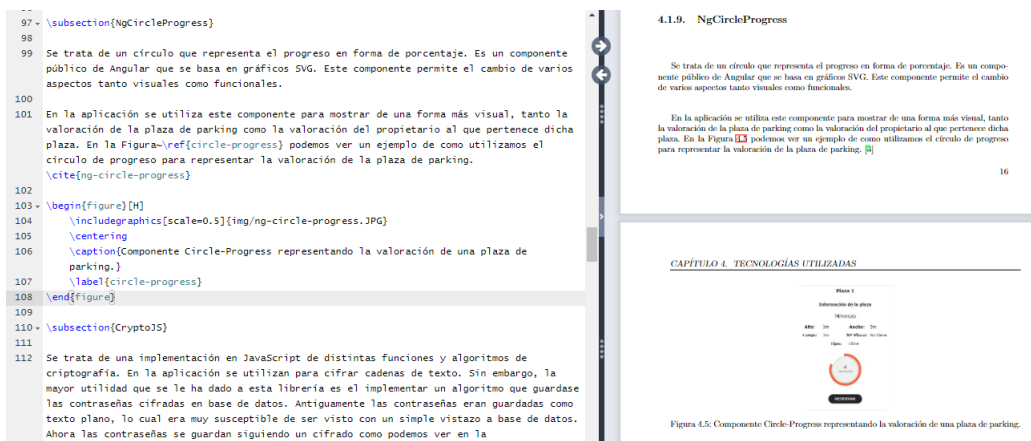


Figura 4.10: Ejemplo de desarrollo en Overleaf

### 4.2.5. Astah

Astah es una aplicación para el modelado de diagramas mediante UML. UML (Unified Modeling Language) es un lenguaje de modelado para aplicaciones software con el que crear diseños y diagramas de un sistema. Para este proyecto, se ha utilizado Astah para crear distintos tipos de diagramas:

- **Diagrama de Clases:** Este diagrama muestra las clases, atributos y operaciones de un sistema, así como su relación y características. Inicialmente se creó un diagrama de clases para esta aplicación que nos permitiese diferenciar los diferentes servicios que había que proporcionar. De esta forma, podríamos aislar los servicios para tratarlos como microservicios (APIs). También ayudó a la hora de crear las clases y elementos en el código, ya que había una idea clara de lo necesario.

- **Diagramas de Secuencia:** En un diagrama de secuencia se describe una sucesión de eventos u operaciones de un fragmento de código. Todas las sucesiones se muestran como intercambios de mensajes entre los elementos del diagrama.
- **Diagrama de Despliegue:** Se utiliza para mostrar los componentes físicos en los que la aplicación estaría operando. También se muestra la conexión entre los diferentes elementos desplegados. Hay dos tipos de nodos (elementos) que se utilizan en este diagrama: nodo de dispositivo y nodo de entorno de ejecución. Los nodos de dispositivos hacen referencia a recursos informáticos físicos con memoria de procesamiento y servicios para ejecutar software. Los nodos de entorno de ejecución representan recursos informáticos software que se ejecuta en un nodo externo y que provee de servicios para alojar y ejecutar software.



## Capítulo 5

# Diseño

En esta sección explicaremos las distintas decisiones de diseño que se han tomado a lo largo del desarrollo. Entre estas decisiones se encuentran tanto patrones de diseño como arquitecturas y decisiones de diseño sobre la interfaz gráfica de la aplicación.

### 5.1. Modelo Vista Controlador (MVC)

A lo largo del desarrollo de la aplicación, uno de los patrones arquitectónicos más utilizados ha sido el patrón Modelo Vista Controlador (MVC). En el patrón MVC se divide la lógica de un programa en tres elementos conectados entre sí. Se aprovecha la división para definir como se van a mostrar y tratar los datos de entrada y salida del usuario. En la Figura 5.1, se muestran los tres componentes del patrón junto con distintos elementos de una aplicación web.

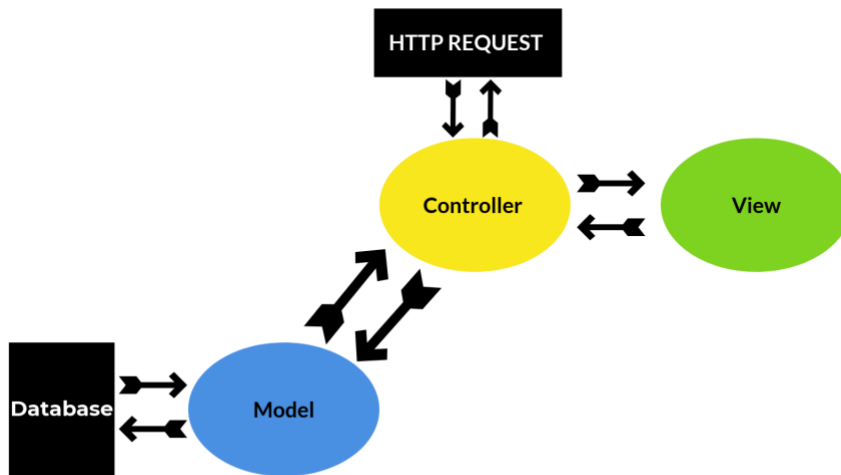


Figura 5.1: Arquitectura MVC en aplicaciones web [9]

Al ser RentApp una aplicación web para su uso en dispositivos móviles, la interfaz de usuario tiene una gran importancia en el desarrollo. En todas las pantallas nos encontramos ya sea con entradas o salidas de datos. Es por esto que el patrón MVC es uno de los que mejor se adapta a la aplicación, manteniendo un código más limpio y estructurado. A continuación tenemos la descripción de los tres componentes del patrón MVC:

- **Modelo:** Es la representación de la información que se encuentra en el sistema. Pese a ser también el encargado de las consultas y modificaciones a la información, la cual se almacena en base de datos, las peticiones de dichas operaciones llegan a él desde el Controlador. Es por esto que el modelo de las aplicaciones que siguen un patrón MVC controlan la lógica de negocio.
- **Vista:** Se encarga de presentar al usuario la información que proviene del Modelo.
- **Controlador:** Se encarga de gestionar los eventos que ocurren en la Vista, así como de enviarla comandos en el caso de que ocurran cambios en el Modelo que requieran de la actualización de la interfaz. También es el encargado de enviar las peticiones al Modelo.

### 5.1.1. Ejemplos de uso en la aplicación

Podemos apreciar este patrón en el inicio de sesión de la aplicación, en donde se sigue el siguiente flujo:

1. El usuario introduce sus credenciales y pulsa el botón en la interfaz de usuario (Vista) para iniciar sesión.
2. El Controlador recibe el evento de un envío de datos en el formulario de inicio de sesión. Como encargado de manejar el evento, este componente obtendrá los datos del formulario y formulará la petición correspondiente.
3. El controlador realizará la petición a través del Modelo, el cual se actualizará dependiendo de los datos introducidos.
4. La Vista obtendrá la respuesta del Modelo y realizará los cambios pertinentes en función de la respuesta recibida. Para el caso particular que estamos viendo, en caso de recibir que el usuario se ha autenticado correctamente (recibe un JWT), la Vista se actualizará, dando paso a la pantalla del mapa con las plazas disponibles. En caso de que no se reciba el JWT, la Vista se actualizará para mostrar que ha habido un problema en la autenticación del usuario.
5. Finalmente volveremos al principio del flujo, en donde la Vista espera nuevas interacciones del usuario.

## 5.2. Arquitectura basada en microservicios

La arquitectura basada en microservicios (Micro Services Architecture, MSA) consiste en dividir una aplicación en servicios, los cuales se ejecutan de forma independiente y se comunican mediante mecanismos ligeros. Se conoce como mecanismos ligeros a los protocolos de comunicación con una carga pequeña en la transmisión de datos. Para este proyecto, los microservicios lo conforman APIs que se comunican mediante recursos HTTP.

### 5.2.1. Arquitectura Monolítica vs Arquitectura en Microservicios

A la hora de pensar en el desarrollo de un proyecto, se nos ocurren dos enfoques posibles para la arquitectura de la aplicación: Monolítica o en Microservicios. La arquitectura monolítica es la forma tradicional de construir las aplicaciones en las que toda la funcionalidad y la lógica se encuentran desarrolladas en un único lugar. En este tipo de aplicaciones, tanto la interfaz del usuario como los servicios de backend, así como las bases de datos se encuentran unificadas en una aplicación. Entre las ventajas de esta arquitectura se encuentran:

- **Menos problemas transversales:** Se refieren a los problemas que afectan a la aplicación en su conjunto. En caso de tener una aplicación dividida en microservicios, estos problemas requerirían de una solución más costosa.
- **Depuración y pruebas más sencillas:** Los test se efectúan más rápidamente en aplicaciones monolíticas. Al mismo tiempo, dichos test son más fáciles de realizar.
- **Facilidad para desplegar:** El despliegue es mucho más sencillo ya que solo hay que desplegar un programa o directorio.

- **Facilidad para el desarrollo:** Al ser la forma más común de desarrollar las aplicaciones, el equipo de desarrollo tendrá más práctica a la hora de diseñar y programar este tipo de aplicaciones.

En cuanto a las desventajas tenemos que:

- **Comprensibilidad:** A medida que una aplicación monolítica evoluciona, su complejidad aumenta. Esto hace que su mantenimiento y trabajo sean más complejos.
- **Realización de cambios:** Los cambios que se realicen afectarán a toda la aplicación. Esto implica que en el caso de que un grupo trabaje sobre la aplicación, se requerirá una coordinación y cuidado especial a la hora de realizar cambios sobre el desarrollo.
- **Escalabilidad:** Los componentes no pueden ser escalados de forma independiente, sino toda la aplicación.
- **Barreras a las nuevas tecnologías:** A la hora de añadir una nueva tecnología para una parte específica, hay que tener en cuenta que afectará a toda la aplicación. Esto hace que un cambio particular pueda conllevar grandes variaciones en la aplicación.

Respecto a la arquitectura basada en microservicios, como se ha mencionado previamente se basa en una aplicación cuyos servicios están separados de forma independiente unos de otros. Cada servicio formará una API, la cual se comunicará mediante recursos HTTP. Cada uno de los servicios de la aplicación podrá ser actualizado, desplegado y escalado de forma independiente. Entre las ventajas de una aplicación basada en microservicios nos encontramos con:

- **Componentes independientes:** El trabajo sobre los componentes de la aplicación es totalmente independiente. Esto facilita los cambios y la evolución de los componentes ya que estos solo afectarán al componente sobre el que se está trabajando, aislándolo del resto de la aplicación.
- **Mayor comprensibilidad:** La manejabilidad que puedes tener sobre una aplicación en la que cada uno de los servicios se encuentra desarrollado de forma independiente es mayor. De esta forma solo tenemos que centrarnos en el microservicio relacionado con el servicio de negocio sobre el que queremos trabajar.
- **Mayor escalabilidad:** Como cada componente puede ser escalado de forma independiente, los cambios sobre la aplicación pueden aplicarse más rápido que en un aplicación monolítica. Esto se debe a que los cambios tan solo afectarán al componente sobre el que se está trabajando, ya que trabajamos de forma aislada sobre el proyecto en vez de trabajar sobre la aplicación en su conjunto.
- **Flexibilidad con las nuevas tecnologías:** Si queremos añadir una nueva tecnología en un componente, no será necesario preocuparse por el resto de la aplicación ya que el cambio no la afectará.



- **Mayor agilidad:** Los cambios se efectúan sobre componentes aislados por lo que serán implementados con un riesgo menor y con menos errores.

En cuanto a las desventajas de los microservicios tenemos que:

- **Complejidad extra:** El desarrollo en microservicios es más complejo que un desarrollo monolítico. Cada servicio incluye conexiones entre módulos y algunos de ellos también incluyen conexión a una base de datos. Todas estas operaciones deben de hacerse para cada servicio, por lo que la complejidad es mayor que si se hiciese una única vez.
- **Distribución del sistema:** Todas las conexiones entre microservicios deben de hacerse cuidadosamente ya que la implementación de un servicio a otro puede variar bastante.
- **Problemas transversales:** El crear una arquitectura basada en microservicios conlleva problemas transversales como pueden ser distintas configuraciones entre otros incidentes.
- **Pruebas:** Se deben realizar más pruebas y de mayor complejidad que en una aplicación monolítica.

Sabiendo todas las características principales de ambas arquitecturas con sus ventajas y desventajas, queda elegir una de ellas para construir la aplicación. [5]

### 5.2.2. ¿Por qué usar microservicios?

RentApp está construida siguiendo una arquitectura basada en microservicios. Pese a que el equipo de desarrollo es pequeño y que la complejidad sería menor, la realidad es que las ventajas de los microservicios se superpusieron a la facilidad de desarrollo.

Uno de los puntos que nos hizo decantarnos por un desarrollo basado en microservicios fue la sincronización. Dado a que este proyecto ha sido desarrollado por dos alumnos, teníamos que contemplar la posibilidad de que alguno de nosotros no continuase con el proyecto sin que esto afectase de forma crítica al desarrollo de la otra persona. Para esto, la independencia entre microservicios ofrecía una solución en el caso de que dicha situación se acabase materializando. Es decir, buscábamos la forma de realizar la aplicación de la forma más independiente posible, y de modo que las decisiones de un desarrollador afectasen lo mínimo al progreso del otro compañero.

A pesar de que la independencia entre ambos proyectos fue un punto clave para descartar un desarrollo monolítico, la razón principal que impulsó la decisión fue la escalabilidad de la arquitectura de microservicios. Pese a que la idea principal de RentApp era el alquiler de plazas de garaje de forma similar a un parking, el diseño de la aplicación abre la puerta a aumentar el tipo de bien a arrendar. Con un diseño que favorece la escalabilidad, y unos servicios independientes, no sería muy complicado introducir nuevos bienes a alquilar, como pueden ser inmuebles. De esta forma aumentaría las opciones que RentApp ofrece a sus

## 5.2. ARQUITECTURA BASADA EN MICROSERVICIOS

usuarios pasando de ser una aplicación de alquiler de cocheras, a ser una aplicación de alquiler de bienes raíces. Adicionalmente este enfoque ayudaría en la escalabilidad si la aplicación se implanta y tiene éxito para poder manejar muchos usuarios y muchas transacciones.

Con la decisión de una estructura basada en microservicios, la aplicación quedaría como se muestra en la Figura 5.2.

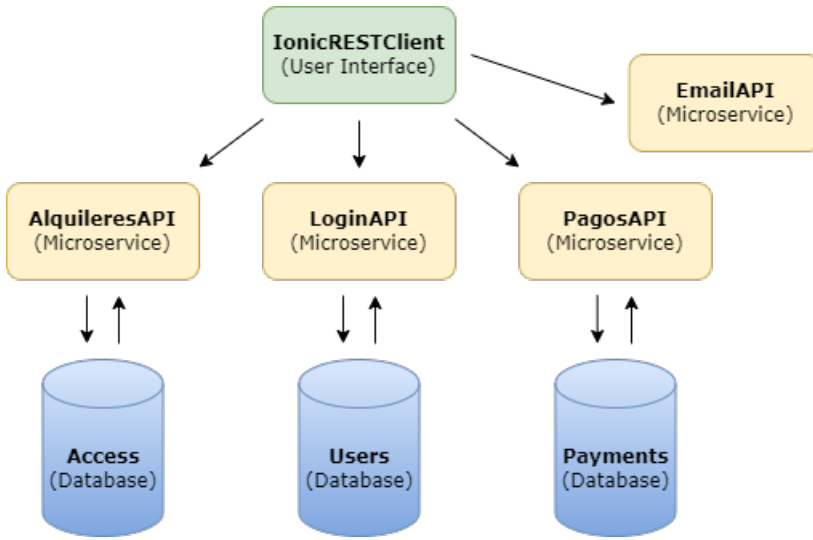


Figura 5.2: Estructura basada en microservicios de RentApp

En la Figura 5.3 podemos ver el diagrama de enrutamiento que sigue la aplicación. En este diagrama se muestran las rutas con las que el cliente front (IonicRESTClient) se comunica con los microservicios.

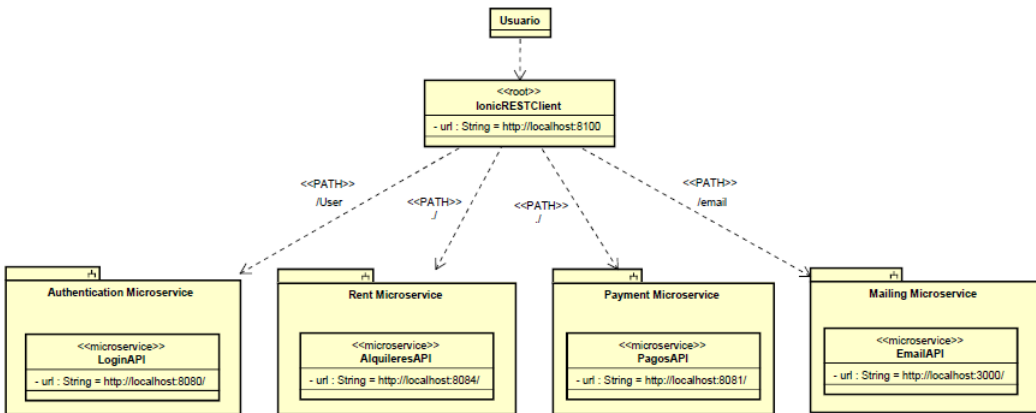


Figura 5.3: Diseño de la API Gateway

A continuación se muestra en la Figura 5.4 el diagrama de despliegue de la aplicación. En este TFG la aplicación se desarrolla en local, por lo que excepto la base de datos, el resto de servicios se encuentran desplegados en el mismo dispositivo físico que el cliente front (el dispositivo del usuario).

En el diagrama se muestra como se desplegaría la aplicación en una versión final. En la figura podemos ver como los microservicios son desplegados en un servidor diferente al dispositivo del usuario. Al mismo tiempo, vemos que la base de datos se encuentra en otro servidor distinto al de los microservicios.<sup>1</sup>

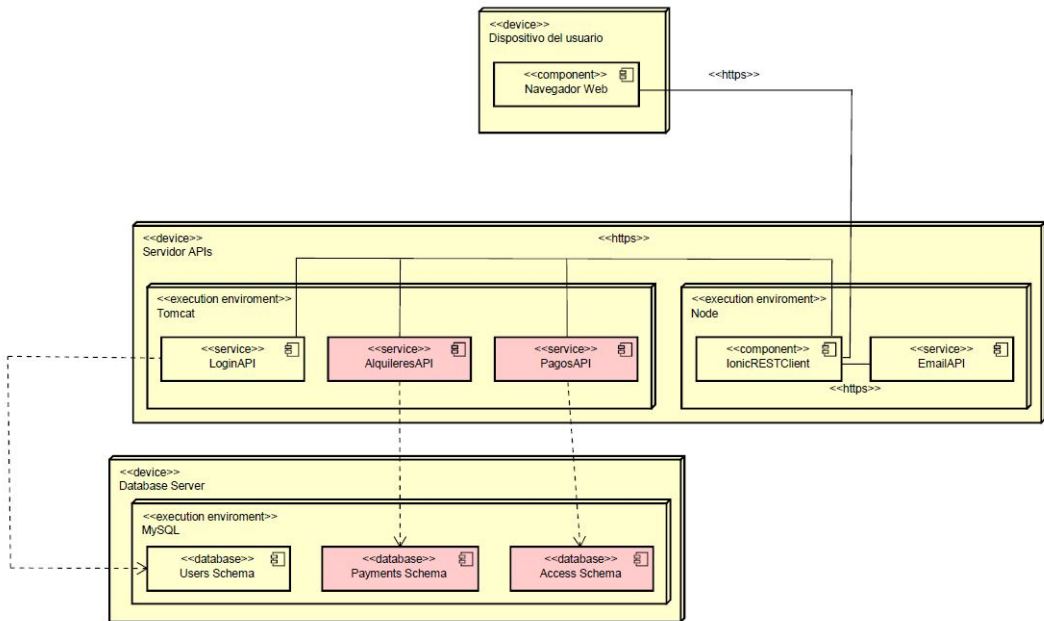


Figura 5.4: Diagrama de despliegue

### 5.2.3. LoginAPI

Se trata de la primera API desarrollada a lo largo del proyecto. Los servicios que comprende son todos los relacionados con los usuarios de la aplicación. LoginAPI se encarga de registrar, autenticar y modificar los datos de los usuarios, así como mostrar su información cuando esta se requiera.

En el modelo tenemos tres clases con las que trabajaremos para realizar la gestión de usuarios:

<sup>1</sup>Los servicios y bases de datos pintados de color rojo indican que forman parte del TFG del otro desarrollador: Carlos Noé Muñoz Bastardo. El resto en color crema forman parte de este TFG y han sido desarrollados por mi: Pablo Verdejo Santana

- **User:** Se trata de la representación de un usuario cualquiera que utiliza la aplicación. Es la clase de la que heredan las otras dos clases del modelo. User se utiliza para representar todos los elementos comunes que comparten ambos tipos de usuarios de la aplicación como pueden ser un correo electrónico, contraseña, etc.
- **Client:** Se trata del cliente de las plazas. Es el usuario que alquila las plazas ofertadas en la aplicación. Tiene un atributo que lo diferencia de los propietarios: el rfid. El rfid es un identificador que será utilizado en AlquileresAPI, que está relacionado con las placas arduino utilizadas en el proyecto del otro desarrollador de la aplicación.
- **Owner:** Es el arrendador de las plazas. Este tipo de usuario tiene más atributos particulares, como son la valoración general, los beneficios totales del alquiler de sus plazas, etc.

A continuación tenemos el diagrama de clases de LoginAPI. Se han omitido las clases relacionadas con los test ya que no se utilizaron para las pruebas. También se han omitido los métodos de las clases del modelo para dar una mayor claridad al diagrama y al no aportar gran información al ser los getters y setters de los atributos de las clases.

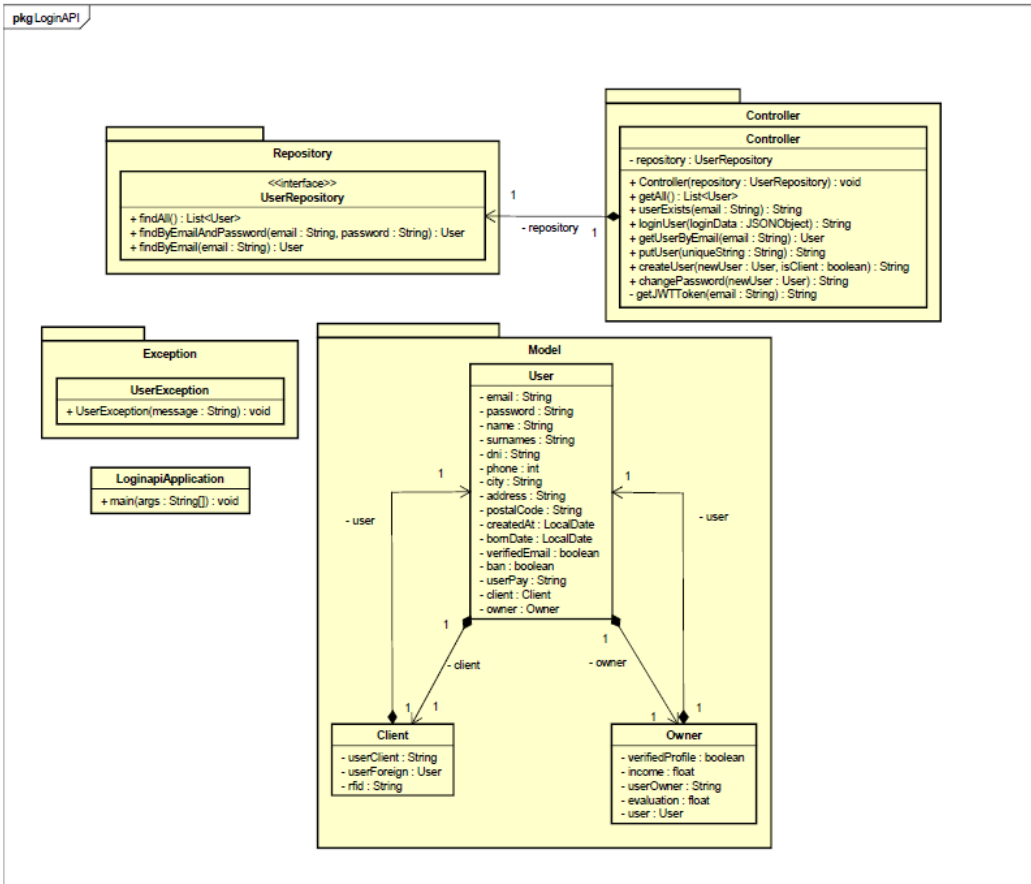


Figura 5.5: Diagrama de clases de LoginAPI

### 5.2.4. EmailAPI

EmailAPI se encarga de los envíos de mensajes a los usuarios. Para el envío de mensajes se utiliza el protocolo SMTP. Se trata de un protocolo de red de conexión de internet utilizado para el intercambio de mensajes de correo electrónico entre dispositivos. Dado a que EmailAPI no dispone de este servicio, utilizaremos los servicios ofrecidos por Gmail de Google. Como podemos ver en la Figura 5.6, lo que hacemos es enviar los datos del mensaje al correo electrónico de este proyecto. Es desde los servicios de Gmail desde donde se envía el mensaje al correo electrónico del usuario.

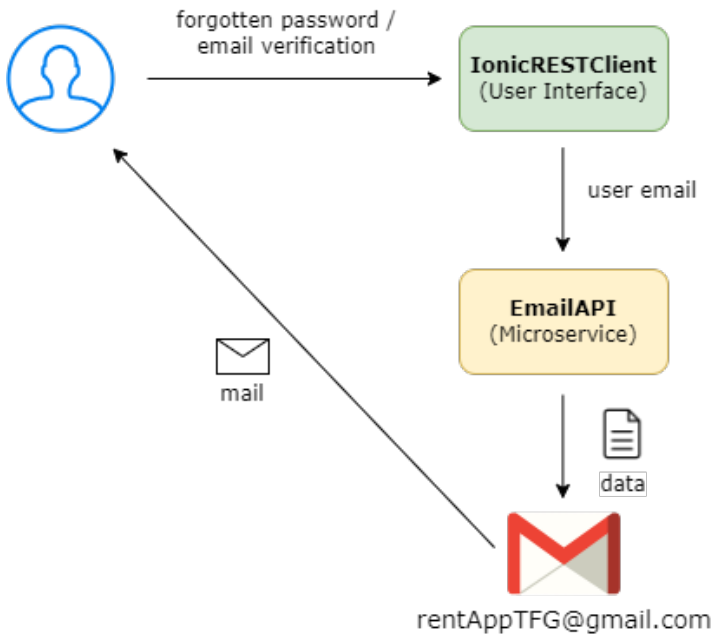


Figura 5.6: Funcionamiento de EmailAPI

Hay dos casos en los que se envían correos electrónicos. Uno de ellos es al registrar un nuevo usuario. Al final del registro, si todo ha salido correctamente, se enviará un mensaje para que el usuario confirme el registro. Hasta que el usuario no acceda al enlace que se adjunta en el cuerpo del correo la cuenta no se verificará y el usuario no podrá acceder a la aplicación con su cuenta.

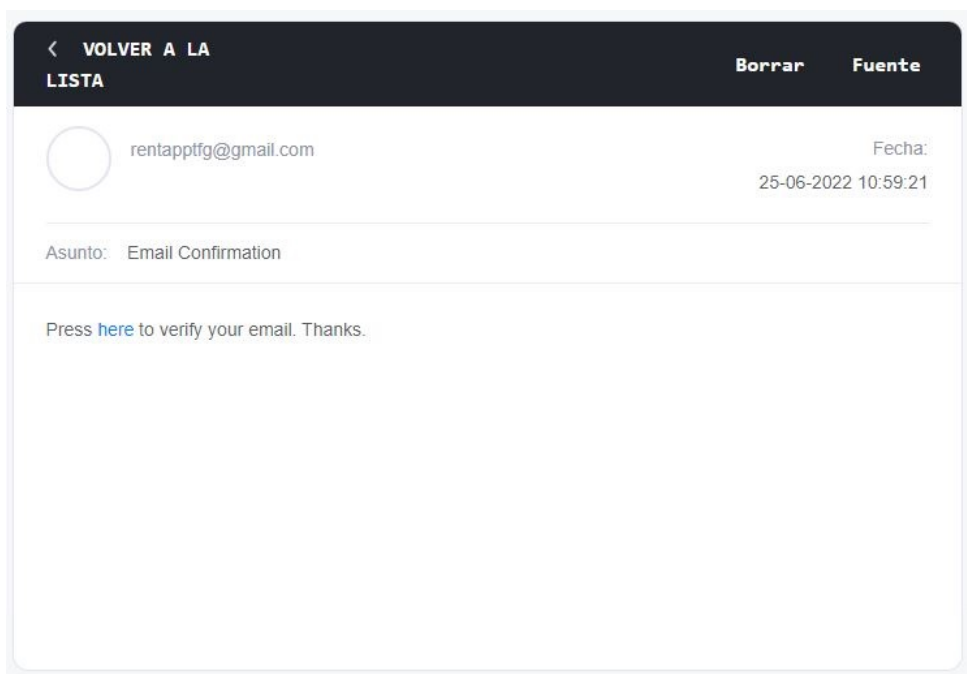


Figura 5.7: Correo de verificación

El otro caso en el que hacemos uso de EmailAPI es para cambiar la contraseña. En el caso de que a un usuario registrado en la aplicación se le olvide la contraseña, este podrá introducir su correo en la aplicación para que se le envíe un correo con acceso a una pantalla desde la que podrá cambiar la contraseña.

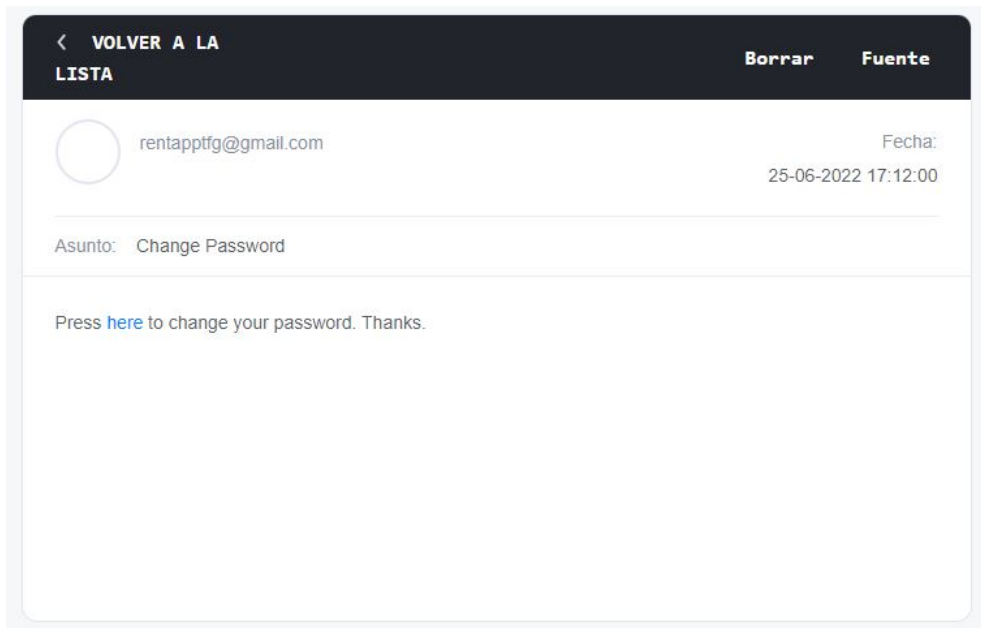


Figura 5.8: Correo de cambio de contraseña

A diferencia de LoginAPI, EmailAPI no utiliza Spring, sino Node. A continuación, tenemos un diagrama de secuencia representando el envío de un correo electrónico debido a un cambio de contraseña (el funcionamiento es similar para el envío de correo de verificación). En el diagrama vemos que el usuario introduce su dirección de correo electrónico en la pantalla y envía la solicitud. Una vez se ha comprobado que el correo es válido, se ejecuta una llamada POST a EmailAPI. Desde EmailAPI se crea un objeto Transporter y se hace uso del correo de la aplicación para enviar el correo final al usuario con todos los datos necesarios.





## 5.3. Diseño de la interfaz gráfica

Como se ha explicado previamente, este proyecto está dividido en dos partes. Dentro de la parte que comprende este TFG se encuentra todo el diseño de la interfaz gráfica de la aplicación. Desde un primer momento, se han ido haciendo bocetos para el diseño de las diferentes interfaces y para calcular la disposición de los elementos que iban a presentarse en cada una de ellas.

La interfaz de la aplicación se ha diseñado para que sea responsive. Esto significa que para un mismo código la interfaz se adapta a distintos tamaños de dispositivo. Cabe destacar que pese a ser responsive, la interfaz no está optimizada para dispositivos extremadamente pequeños como podrían ser los SmartWatch. Tampoco es una aplicación pensada para su uso en pantallas grandes como puede ser un portátil u ordenador. Como se ha mencionado, el objetivo de este proyecto era diseñar una aplicación para dispositivos móviles, por lo que la interfaz responderá bien ante estos dispositivos.

Ionic da la opción de elegir el tema por defecto en el que se mostrará la aplicación (tema oscuro, tema claro, etc). Por defecto, Ionic selecciona el tema del dispositivo móvil en el que se encuentra la aplicación. Sin embargo, tras haber diseñado ya varias de las interfaces vi que un tema oscuro iba a obligar a reconsiderar el contraste y tema entre los diferentes elementos ya que había diseñado las pantallas fijándome en un tema claro. Si hubiese dejado el tema por defecto, varias de las interfaces tendrían textos ilegibles en modo oscuro debido a que no habían sido adaptadas a dicho tema. Es por esto que decidí poner el tema claro por defecto.

Otra de las decisiones de diseño que hubo que pensar fue el menú. La idea inicial fue crear un menú lateral oculto en el que meter los diferentes apartados. Sin embargo, esta decisión no tenía mucho sentido debido a los pocos elementos que aparecerían en el menú. Finalmente se optó por diseñar un menú inferior, que estuviese visible en todo momento. En este menú se encuentran los apartados de Reservas, Mapa y Perfil de Usuario. Si navegamos dentro de cada opción encontraremos más funcionalidades.

Pese a que las aplicaciones Ionic pueden ser compiladas para varios sistemas operativos, esto significa que la aplicación deberá adaptarse a todos ellos. Un problema que se apreció a mitad del desarrollo fue que con un sistema operativo Android, podemos navegar atrás con las opciones del menú del propio sistema operativo. Con iOS esto no es posible. Por esta razón se ha optado por crear un pequeño botón al lado del logotipo superior de la aplicación. De esta forma, podemos navegar hacia atrás con ambos sistemas operativos.

Todos los diseños se han hecho a partir de bocetos realizados a mano alzada. Pese a que existen aplicaciones orientadas a la realización de bocetos, personalmente prefería realizar los bocetos a mano con sus ventajas e inconvenientes. La ventaja principal era la rapidez. Al no tener que aprender a utilizar una aplicación ni a familiarizarme con todos sus componentes la parte de realizar bocetos era realizada más rápidamente.

Sin embargo no todo son ventajas. Las aplicaciones para realizar bocetos ofrecen muchas más posibilidades a la hora de presentar la aplicación y de estructurar sus componentes. Una de las funcionalidades más útiles era simular la navegación entre pantallas de la aplicación. A pesar de que las pantallas en este tipo de aplicaciones no tienen ningún tipo de funcionalidad,

son muy útiles para trazar las distintas rutas internas del proyecto y los caminos a seguir para realizar una acción específica. Además muchas de ellas permiten generar una presentación digital a partir de los diseños, muy útil a la hora de presentar el proyecto ya que ofrecen una visión mucho más realista de lo que se espera de la aplicación.



## Capítulo 6

# Implementación y pruebas

### 6.1. Implementación

En este apartado se describen al distribución de directorios de los proyectos desarrollados. Para una mayor claridad y simplicidad se han omitido directorios de configuración o directorios que no han sido utilizados. Se describen los directorios en los que se ha trabajado o los directorios de mayor importancia.

#### 6.1.1. Distribución de paquetes de Ionic

Empezando desde arriba en la Figura 6.1 tenemos la raíz con el nombre del proyecto: IonicRESTClient. En el siguiente nivel se encontrarían los ficheros de configuración encargados de la gestión de paquetes entre otras cosas. En este nivel tenemos cuatro directorios. El directorio de android y de ios han sido generados mediante una compilación del proyecto Ionic con Cordova. Con la compilación se generan los ficheros necesarios para que la aplicación pueda ser utilizada en dispositivos Android e iOS.

Otro directorio en este nivel es el de node\_modules, carpeta que contiene todas las dependencias de nuestro proyecto. A continuación, tenemos el directorio src, muy importante ya que será el directorio principal donde se ha trabajado a lo largo del proyecto, y el directorio que contiene todo el código.

Dentro de src tenemos varios ficheros muy importantes para el proyecto. Destacan index.html y global.scss, siendo la página html principal que cargará en cada pantalla y la hoja de estilos global respectivamente. En main.ts es el fichero Type Script en donde se incluirán las configuraciones globales del proyecto.

Situándonos dentro de src, tenemos varios directorios. Entre ellos se encuentra /theme, donde se incluyen ficheros con estilos globales de Ionic que se usarán en toda la aplicación.

También tenemos `/enviroments`. Dentro de este directorio se encuentran las configuraciones y variables de entorno del proyecto. En este caso, se encuentra la API key de Mapbox. Siguiendo tenemos también la carpeta de `/assets` utilizada para alojar las imágenes e iconos del proyecto. Dentro de este directorio tenemos `/icon`, en donde se guarda el icono principal de la aplicación y `/images`, donde se guardan todas las imágenes que se han utilizado.

Finalmente tenemos el directorio de `/app`, en donde se aloja la mayoría del código. Si miramos dentro de `/app` nos encontramos con un fichero de módulos global, fichero de componentes y sobre todo, el fichero de enrutado. Este fichero (`app-routing.module.ts`) contiene todas las rutas del proyecto. Es el fichero gracias al cual podemos navegar por la aplicación.

Dentro de `/app` también tenemos otros directorios. Para simplificar el esquema he decidido sustituir todos los directorios de páginas por `/pages` dado a que todos ellos comparten la misma estructura. Dentro de los directorios de `/pages` tenemos un fichero `html` en el que diseñar la página, un fichero `scss` en el que añadir los estilos de la página y tres ficheros `Type Script`. En los ficheros `*-routing.modules.ts` tenemos la configuración del enrutado de ese componente. El fichero `*-module.ts` se utiliza para cargar los módulos del componente. Finalmente, el fichero `*.page.ts` contiene las configuraciones particulares del componente. Dentro de estas páginas pueden crearse otras nuevas como subdirectorios.

Otro de los directorios a destacar dentro de `/app` es `/guards`. Aquí se ubican los fuentes encargados de asegurarse de que no se acceda a ciertas rutas sin permiso. En nuestro caso tenemos tres, aunque pueden dividirse en dos tipos:

- **Autenticación:** Es el encargado de que ningún usuario que no haya iniciado sesión acceda a páginas para usuarios registrados.
- **Cliente y Propietario:** Se aseguran de que un cliente no acceda a pantallas o funcionalidades pertenecientes a un propietario y viceversa. En ocasiones se usan de forma complementaria con el guard de Autenticación.

Finalmente tenemos el directorio de `/services`. Dentro de este directorio hay dos tipos de ficheros. El primero es `app.model.ts` en el que se definen interfaces para los modelos utilizados en la aplicación. El otro tipo de ficheros son servicios creados para este proyecto. Cada servicio agrupa funcionalidades específicas para ser utilizadas por los diferentes componentes.

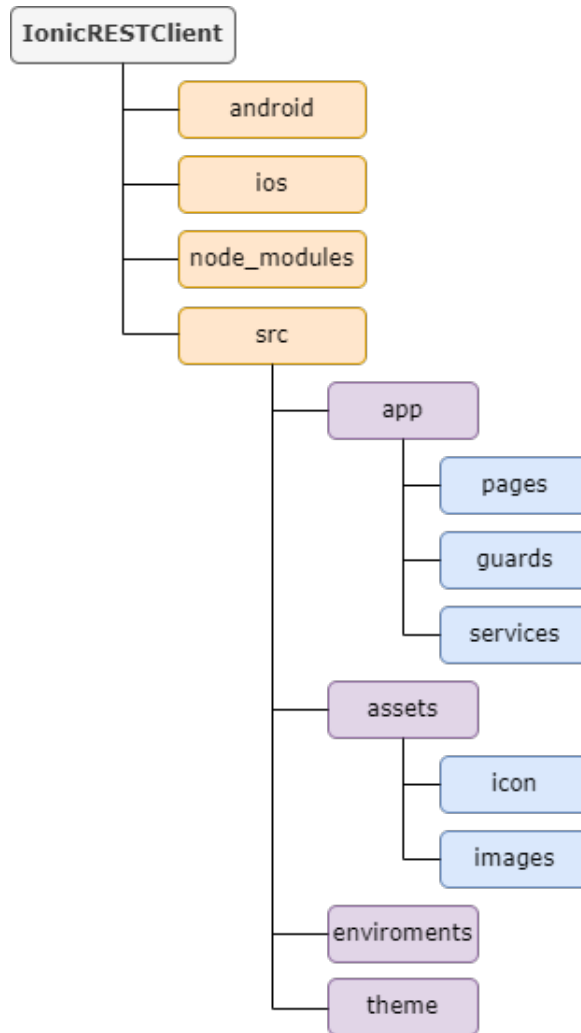


Figura 6.1: Estructura de directorios de Ionic

### 6.1.2. Distribución de paquetes de Spring

Como ya se ha mencionado, para crear la API de autenticación (LoginAPI) se hizo uso de Spring. Si comenzamos desde la raíz tenemos que el fichero más destacable es pom.xml, el cual se utiliza para declarar la configuración de Maven. Si nos fijamos en el directorio, tenemos que dentro de /src/main tenemos dos apartados principales.

En el apartado de /resources se encuentra el fichero application.properties encargado de la configuración de la aplicación de Spring. El uso dado a este fichero fue establecer los datos de conexión con el servidor que alojaba la base de datos para así conectarla a LoginAPI. En otras APIs también se utilizaba para declarar el puerto sobre el que la API iba a trabajar.

El otro apartado es un conjunto de directorios que nos llevan al subdirectorio `/loginapi` en donde se alojará todo el código.

Dentro de este directorio nos encontramos con la primera clase java: `LoginapiApplication.java`. Esta clase es la encargada de lanzar la API, conteniendo el método `main`. El resto del código está estructurado en subdirectorios dentro de `/loginapi`.

Siguiendo un orden descendente en la Figura 6.2 el primer directorio con el que nos encontramos es `/Controller`. En este directorio se encuentra `Controller.java`, clase que contiene todos los métodos `http` definidos en esta API. A continuación tenemos `/Exception`, que contiene la clase `UserException.java` usada para mostrar algunas excepciones en los métodos de `Controller`.

Si continuamos descendiendo tenemos `/Model`, donde se define el modelo de la API. Dentro de este directorio tenemos una clase java para cada tipo de usuario que tiene la aplicación y una clase más para representar los atributos generales de los usuarios. Existe una clase java por entidad en la base de datos debido a que API y base de datos están conectadas. Finalmente tenemos `/Repository` con `UserRepository.java`. Este fuente contiene una interfaz java que extiende de `JpaRepository`. En ella se definen métodos de consulta a base de datos.



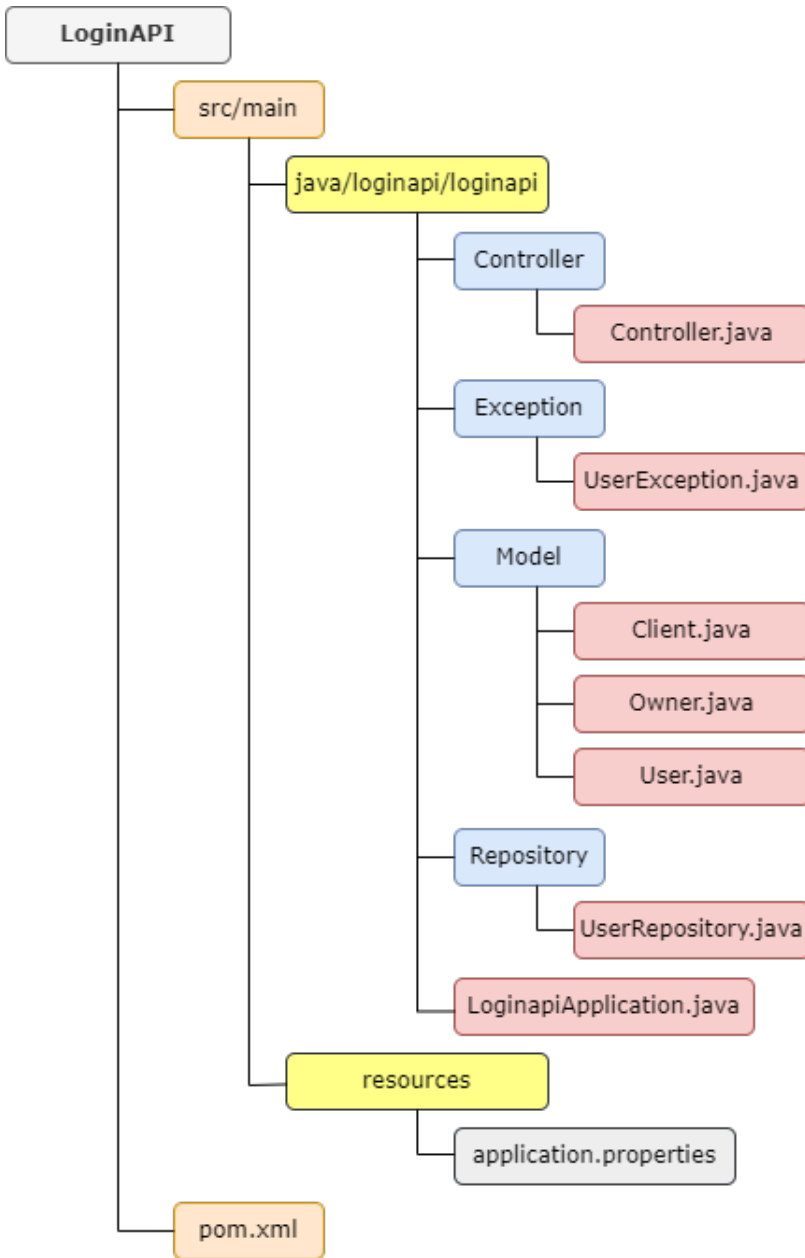


Figura 6.2: Estructura de directorios de LoginAPI

### 6.1.3. Distribución de paquetes de Node.js

La API en la que se ha usado Node.js ha sido EmailAPI. Esta es la API más pequeña del programa debido a su sencillez y funcionalidad. Para explicar su distribución, se han

recopilado los ficheros y directorios más relevantes de esta API. Al igual que hemos visto en Ionic, si miramos un nivel por debajo de la raíz veremos los archivos de configuración. En ellos se encuentran las dependencias de la API.

Respecto a los directorios, los más relevantes son `/node_modules` y `/routes`. En el primero de ellos se encuentran todos los módulos que utiliza la API. Entre ellos está Nodemailer, que es el módulo que se encarga del envío de correos electrónicos. Dentro de `/routes` tenemos los dos ficheros en los que hemos diseñado la funcionalidad de los dos tipos de correos que envía la API: correo de recuperación de contraseña y de validación de cuenta de usuario.

En el fichero `emailRouter.js` se encuentra la funcionalidad para los correos de validación de usuario, mientras que en `passwordRouter.js` tenemos la funcionalidad para los correos de recuperación de contraseña.

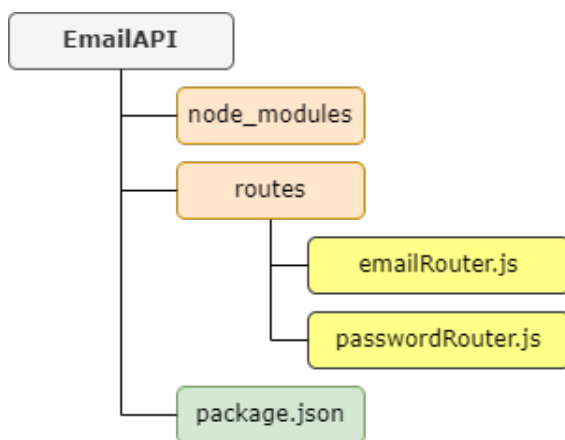


Figura 6.3: Estructura de directorios de EmailAPI

## 6.2. Pruebas

Para este proyecto se han hecho pruebas para los tres desarrollos principales: Ionic, LoginAPI y EmailAPI.

### 6.2.1. Pruebas de Ionic

Las pruebas de Ionic vienen definidas en el Excel “Pruebas IonicRESTClient.xlsx” que se adjunta junto al resto de contenidos de este proyecto. Las pruebas descritas son pruebas funcionales de caja negra que comprueban que la funcionalidad ofrecida por todas las interfaces de la aplicación funcione de la forma en que se espera.

Las pruebas diseñadas no han sido automatizadas, de forma que para ejecutarlas se debe de probar la funcionalidad descrita. Pese a ello, a la hora de probar diferentes entradas si que

fueron creadas interfaces y JSONs para evitar el introducir datos constantemente y agilizar el trabajo.

### 6.2.2. Pruebas de LoginAPI

Para las pruebas de LoginAPI, se ha creado una batería de pruebas de caja negra para comprobar su funcionamiento. Las pruebas de caja negra consisten en pruebas en las que se ignora la estructura interna del código, sólo se evalúa la funcionalidad. Para esta API, se ha comprobado el funcionamiento de todos los métodos HTTP de los que dispone con sus correspondientes casos alternativos.

| Código | Descripción   | URL                  | Parámetro    | Body       | Operación | Return       | Resultado |
|--------|---|----------------------|--------------|------------|-----------|--------------|-----------|
| 1.1    | Obtener todos los usuarios  | /users               | -            | -          | GET       | List<User>   | OK        |
| 1.2.1  | Obtener usuario por email   | /getUser/email       | -            | -          | GET       | User         | OK        |
| 1.2.2  | Obtener usuario por email pasando un email no registrado en base de datos |                      | -            | -          |           | Exception    | OK        |
| 1.3.1  | Comprobar que un usuario existe en base de datos                          | /getUserExists/email | -            | -          | GET       | String = "1" | OK        |
| 1.3.2  | Comprobar que un usuario no existe en base de datos                       |                      | -            | -          |           | String = "0" | OK        |
| 1.4.1  | Autenticar a un usuario registrado en base de datos                       | /login               | -            | JSONObject | POST      | String       | OK        |
| 1.4.2  | No autenticar a un usuario no registrado en base de datos                 |                      | -            |            |           |              | OK        |
| 1.5.1  | Crear un usuario de tipo Propietario                                      | /createUser          | isClient     | User       | POST      | String       | OK        |
| 1.5.2  | Crear un usuario de tipo Cliente  |                      |              |            |           |              | OK        |
| 1.5.3  | Crear un usuario que ya esté registrado en la aplicación                  |                      |              |            |           |              | OK        |
| 1.6.1  | Validar un usuario  | /validateUser        | uniqueString | -          | PUT       | String       | OK        |
| 1.6.2  | Validar un usuario que ya ha sido validado                                |                      |              | -          |           |              | OK        |
| 1.6.3  | Validar un usuario no existente en la base de datos                       |                      |              | -          |           | Exception    | OK        |
| 1.7.1  | Cambiar la contraseña de un usuario                                       | /changePassword      | -            | User       | PUT       | String       | OK        |
| 1.7.2  | Cambiar la contraseña de un usuario no registrado en base de datos        |                      | -            |            |           |              | OK        |

Tabla 6.1: Batería de pruebas de LoginAPI

### 6.2.3. Pruebas de EmailAPI

Para las pruebas de EmailAPI tan solo se probaron dos casos, uno por cada tipo de correo.

## 6.2. PRUEBAS

---

| <b>Código</b> | <b>Descripción</b>                           | <b>URL</b> | <b>Body</b> | <b>Operación</b> | <b>Return</b> | <b>Resultado</b> |
|---------------|--|------------|-------------|------------------|---------------|------------------|
| 1.1           | Enviar correo de verificación                | /email     | JSONObject  | POST             | void          | OK               |
| 1.2           | Enviar correo de recuperación de contraseñas | /password  | JSONObject  | POST             | void          | OK               |

Tabla 6.2: Batería de pruebas de EmailAPI

## Capítulo 7

# Seguimiento del proyecto

Debido a que para el desarrollo del proyecto hemos seguido la metodología Scrum, el desarrollo del proyecto se ha dividido en Sprints. Este proyecto consta de 9 Sprints de dos semanas cada uno, a excepción del primero, que duró una semana. Para cada uno de los Sprints se estimaba una carga de trabajo de 35 horas, a excepción del primero, al cual se le dedicaron 15 horas.

Antes de empezar con el desarrollo del proyecto y con la planificación en tareas del primer Sprint, se identificaron las historias de usuario que iban a componer el desarrollo. En Scrum, las historias de usuario son tareas de desarrollo descritas de forma general que representan una funcionalidad para el usuario final. Estas funcionalidades deben proporcionar al usuario final un valor. En la Tabla 7.1 podemos ver la lista de todas las historias de usuario definidas para mi parte del proyecto.

Como puede apreciarse en la tabla, hay una columna llamada “Peso” con una serie de valores. Cada unidad de peso equivale a cinco horas de desarrollo. Es por esto que, por ejemplo para la primera historia de usuario con un peso de 4, se estima que para completar la funcionalidad descrita por dicha historia de usuario se tardarán un total de  $4 * 5h = 20h$ . No debe confundirse el tiempo estimado para la historia de usuario con la columna “Estimate”. La columna “Estimate” es la suma del tiempo estimado de todas las tareas asignadas a esa historia de usuario, así como “Spend” es la suma de todo el tiempo real empleado en las tareas asignadas a esa historia de usuario.

| <b>Historia de Usuario</b>   | <b>Peso</b> | <b>Estimate (h)</b> | <b>Spend (h)</b> |
|--|-------------|---------------------|------------------|
| Como usuario quiero poder identificarme para acceder a mi cuenta personal  | 4           | 17                  | 20               |
| Como usuario quiero poder registrarme  | 6           | 32                  | 37               |
| Como usuario quiero poder recuperar mi contraseña  | 2           | 10                  | 8                |
| Como usuario quiero poder ver mis datos personales y editar los más relevantes                                       | 6           | 22                  | 22               |
| Como arrendatario quiero poder filtrar las plazas disponibles para encontrar la que más se ajuste a mis preferencias | 3           | 17                  | 18               |
| Como arrendatario quiero poder ver toda la información de una plaza  | 5           | 28                  | 30               |
| Como arrendatario quiero poder ver las plazas disponibles a mi alrededor   | 6           | 28                  | 31               |
| Como arrendatario quiero poder realizar un alquiler  | 5           | 24                  | 22               |
| Como arrendatario quiero poder ver el estado de mi reserva   | 4           | 8                   | 8                |
| Como arrendador quiero poder alquilar bienes   | 6           | 23                  | 25               |

Tabla 7.1: Historias de usuario

A continuación, veremos en detalle el trabajo realizado en cada Sprint, así como problemas que aumentaron las horas de trabajo o tareas realizadas antes de tiempo que redujeron las horas de trabajo. En cada sección se indicarán las tareas realizadas para cada Sprint con su respectiva estimación y el tiempo total empleado para realizarlo. También se indicará con que repositorio se relaciona cada una de las tareas desarrolladas, así como la historia de usuario a la que pertenece. Cabe destacar que el orden de las tareas en las tablas no sigue un orden

de realización específico. Esto se debe a que en ocasiones se realizaban tareas en paralelo.

## 7.1. Sprint 1 (23/02/2022 - 01/03/2022)

Se trata del comienzo del proyecto. Este Sprint tuvo una duración de una semana cargada con tareas por valor de 15 horas. Al final del Sprint, se decidió que la duración del Sprint era bastante escasa por lo que que era complicado tener una versión entregable para el Sprint Review. Debido a este problema se decidió extender la duración del resto de Sprints a dos semanas, dedicándole un total de 35 horas a cada Sprint.

Respecto a las tareas desarrolladas en este primer Sprint, los objetivos principales era instalar todas las herramientas necesarias con las que se iba a desarrollar el proyecto. También mantuve reuniones junto al otro desarrollador para el diseño y creación de la base de datos.

Como podemos observar, hubieron dos estimaciones erróneas. La más destacable se encuentra en la creación y sincronización del usuario en la base de datos. La razón de dicho retraso era la falta de familiarización con el entorno de trabajo. A pesar de haber trabajado previamente con Spring, me faltaban refrescar los conocimientos, por lo que tardé más de lo esperado en acostumbrarme a esta tecnología.

Otro de los retrasos fue durante la instalación de Ionic. Esto es debido a que durante esa tarea también se instalaron más herramientas, necesarias para el proyecto.

| Historia de Usuario   | Peso | Issue   | Estimate (h) | Spend (h) | Repository      |
|---|------|---|--------------|-----------|-----------------|
| Como usuario quiero poder identificarme para acceder a mi cuenta personal | 4    | Realizar las consultas de identificación de usuario | 3            | 3         | LoginAPI        |
|   |      | Crear y sincronizar el User en la DB                | 2            | 4         |                 |
|   |      | Diseño de la Interfaz de Login                      | 3            | 3         | IonicRESTClient |
|   |      | Crear la DB   | 3            | 3         |                 |
|   |      | Instalación de Ionic                                | 4            | 5         |                 |

Tabla 7.2: Sprint 1

## 7.2. Sprint 2 (02/03/2022 - 15/03/2022)

Para este Sprint me centré en desarrollar LoginAPI, en específico, la parte de los métodos de usuarios y el método de registrar usuarios. También se finalizó la funcionalidad del inicio de sesión en la aplicación. Para el final de este Sprint, también nos encontramos con que se necesitaba una forma de confirmar a los usuarios que se registraban en la aplicación. La forma convencional para que un usuario confirme su registro es a través de un mensaje por correo electrónico. Esto, a parte de confirmar que el email introducido es el correcto, evita que un usuario pueda registrarse haciendo uso del correo electrónico de otra persona. Por esta necesidad, se creó la segunda y última API que iba a desarrollar: EmailAPI.

A lo largo del Sprint surgieron tres problemas en tres tareas diferentes:

- **Crear el servicio de los registros:** El exceso de tiempo fue debido a que había más comprobaciones que hacer de las que se pensaba en un principio. Otra causa fue que hubo que hacer cambios en la base de datos a mitad del desarrollo, con los cambios que esto conlleva al resto de la aplicación por lo que fue necesario refactorizar el código.
- **Crear las consultas en la DB de la funcionalidad de registro:** Durante el desarrollo del servicio tuve que conectar varias tablas de la base de datos de LoginAPI, lo que en un principio ocasionó errores en la ejecución. Más tarde se comprobó que los fallos se debían a un mal diseño de la base de datos y a la falta de un campo en la tabla de Owner necesario para su uso como foreign key (FK). Más tarde, se trasladaron los cambios de la base de datos al modelo con el que se trabaja desde Spring.
- **Crear API de envío de correos electrónicos:** Inicialmente se esperaba utilizar Nodemailer dentro de un servicio de Angular. Esto ocasionó una lista enorme de fallos de dependencias en Angular. Tras solventar los problemas de dependencias actualizando versiones compatibles entre los componentes utilizados en Angular, había un problema con Nodemailer que persistía y que causaba fallos en otras dependencias. Tras investigar, me di cuenta de que el error era que no se podía utilizar Nodemailer en un Cliente Front, por lo que tuve que crear una API a parte (EmailAPI) con toda la funcionalidad del servicio de correos en una aplicación Node backend.

A pesar de que no era el plan inicial crear una API para el envío de emails, a la larga se vio que era la solución correcta ya que más adelante se utilizaría este mismo servicio para enviar correos para la recuperación de contraseñas. Al externalizar toda la funcionalidad de los correos, conseguimos modularidad en la aplicación. Además, el realizar la funcionalidad en una API que se ejecutará en un servidor mejorará la integridad de la aplicación. Esto se debe a que para hacer uso de esta API, necesitamos un correo electrónico con su correspondiente contraseña para poder enviar los correos desde esa cuenta. Las credenciales de la cuenta de correo están más seguras en el servidor donde se alojarán las APIs de la aplicación que en el front, visible para cualquier usuario.



| Historia de Usuario   | Peso | Issue  | Estimate (h) | Spend (h) | Repository      |
|---|------|--|--------------|-----------|-----------------|
| Como usuario quiero poder identificarme para acceder a mi cuenta personal | 4    | Completar la funcionalidad del Login                           | 3            | 3         | IonicRESTClient |
| Como usuario quiero poder registrarme                                     | 6    | Crear las interfaces de Registro                               | 3            | 3         |                 |
|   |      | Crear el servicio de los Registros                             | 5            | 7         |                 |
|   |      | Crear las consultas de la funcionalidad de registro en la DB   | 5            | 7         | LoginAPI        |
|   |      | Modificación de servicio de registros y política de privacidad | 5            | 5         | IonicRESTClient |
|   |      | Servicio de verificación de usuario por correo                 | 7            | 7         |                 |
|   |      | Crear consultas para la verificación de usuario                | 3            | 3         | LoginAPI        |
| Crear API de envío de correos electrónicos                                | 4    | 5  | EmailAPI     |           |                 |

Tabla 7.3: Sprint 2

### 7.3. Sprint 3 (16/03/2022 - 28/03/2022)

En el Sprint 3 se desarrollaron dos funcionalidades principales: recuperación de contraseña y visualización de un mapa con una indicación por plaza.

La parte de recuperación de contraseña era necesaria debido a como se había construido el registro e inicio de sesión de la aplicación. El login no conecta con ninguna API de correos o de Google que permita la autenticación automática. Es decir, para la autenticación de un Usuario, éste debe de introducir su correo electrónico y la contraseña. En el caso de que se olvidase de la contraseña, autenticarse sería imposible, por lo que era conveniente crear alguna forma de solucionar ese problema.

El desarrollo de esta funcionalidad fue más rápido de lo esperado ya que se implemento correctamente desde un principio y los fallos que aparecieron pudieron ser solventados con relativa facilidad, lo que agilizó en gran medida el desarrollo.

La segunda parte principal de este Sprint era la implementación de un mapa interactivo que permitiese al usuario localizar las plazas que habían sido registradas en la aplicación. Para la implementación del mapa hicimos uso de la API de Mapbox GL JS, que nos ofrecía la interfaz del mapa y la posibilidad de pintar iconos con sus respectivos pop-up, que representarían las plazas con su información. Durante esta tarea me encontré con tres problemas que hicieron que tardase más tiempo de lo estimado en hacer el desarrollo:

- El primer problema le encontré al intentar cargar la imagen que utilizaríamos para señalar la ubicación de los parking (el icono). El error decía que no se reconocía al

mapa. Todo resultó ser un problema de instancias, ya que al estar anidando métodos, la referencia de una instancia cambiaba, haciendo que referenciase al objeto incorrecto e imprimiendo un error por pantalla. La solución fue guardar la instancia "map" (la que quería hacer referencia) en un objeto y trabajar con ese objeto en vez de estar haciendo "this.map" (referencia con la que se imprimía el error).

- El segundo problema fue al intentar customizar los iconos que indicarían la posición del parking en el mapa. El icono no cargaba por lo que no se veían las ubicaciones en el mapa. La solución fue muy sencilla ya que el error estaba en cómo construía el JSON que se pasaba al mapa. Habiendo creado ya la imagen con nombre "custom-marker", la estaba intentando introducir en el JSON como "custom-marker", cuando el formato correcto era "custom-marker". Las llaves de los laterales sobaban, lo que causaba que no se reconociese el nombre de la imagen. Quitando las llaves el problema fue resuelto.
- El último problema es que los pop-up no se pintaban en las coordenadas correctas, moviéndose al hacer zoom al mapa. El problema es que no se había cargado correctamente la hoja de estilos de Mapbox. Añadiendo la siguiente línea en la cabecera del HTML, el problema fue resuelto: `<link href='https://api.mapbox.com/mapbox-gl-js/v0.42.0/mapbox-gl.css' rel='stylesheet' />`

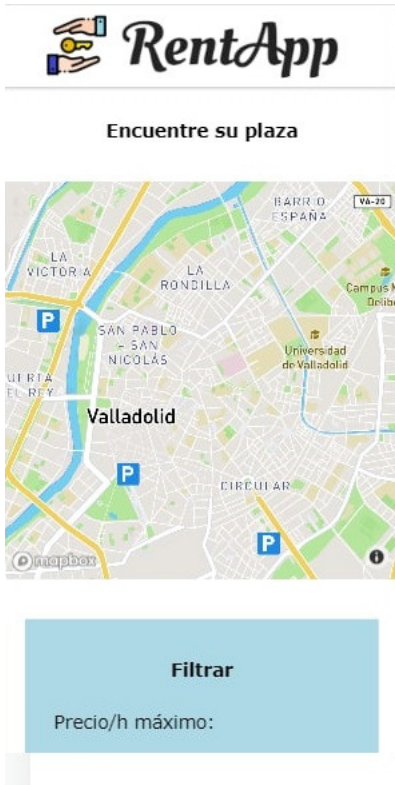
| Historia de Usuario  | Peso | Issue  | Estimate (h) | Spend (h) | Repository      |
|--|------|--|--------------|-----------|-----------------|
| Como usuario quiero poder recuperar mi contraseña                        | 2    | Crear servicio de recuperación de contraseñas        | 7            | 5         | IonicRESTClient |
|  |      | Crear método de comprobación de usuarios registrados | 1            | 1         | LoginAPI        |
|  |      | Crear método de cambio de contraseña                 | 1            | 1         |                 |
|  |      | Crear correo de recuperación de contraseña           | 1            | 1         | EmailAPI        |
|  |      | Solución de errores                                  | 1            | 1         | IonicRESTClient |
| Como arrendatario quiero poder ver las plazas disponibles a mi alrededor | 6    | Diseñar interfaz del mapa                            | 7            | 7         | IonicRESTClient |
|  |      | Pintar todas las ubicaciones en el mapa              | 7            | 10        |                 |
|  |      | Pintar las ubicaciones de DB                         | 10           | 10        |                 |

Tabla 7.4: Sprint 3

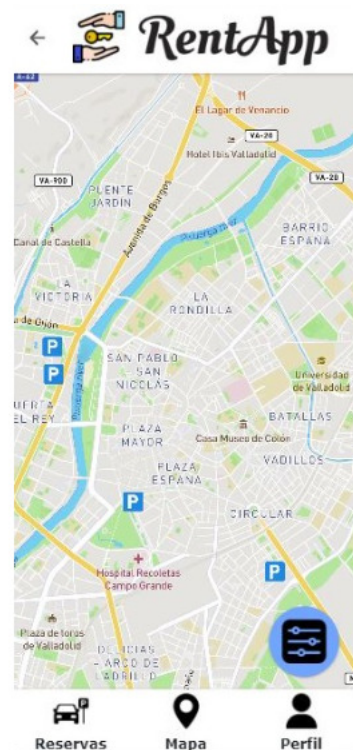
## 7.4. Sprint 4 (29/03/2022 - 12/04/2022)

Para este Sprint las tareas principales eran implementar alguna forma de filtrar todas las plazas que aparecían en el mapa dependiendo de sus características. Es por esto que para este Sprint se diseñó la parte front del filtro. En un diseño inicial, la mitad superior de la pantalla se dedicaba al mapa, mientras que la mitad inferior se dedicaba al filtro. Más

adelante nos dimos cuenta de que era bastante incómodo navegar por el mapa debido a su tamaño. Es por esto que cambié la disposición del filtro a como se muestra ahora. En el diseño actual, el formulario del filtro se encuentra en un botón flotante en la esquina derecha de la pantalla, mientras que el mapa ocupa toda la pantalla. Si hacemos click en el botón de filtrado nos encontraremos con una ventana emergente con el formulario del filtro. Si volvemos a seleccionar el botón, el formulario se vuelve a esconder. Esto puede apreciarse en la Figura 7.1 .La conexión del método de filtrado de AlquileresAPI no se completó en este sprint por falta de tiempo, por lo que se desarrolló entre el Sprint 4 y el Sprint 5.



Diseño Inicial



Diseño Final

Figura 7.1: Comparación del diseño inicial y final de la pantalla del mapa

La otra tarea era implementar la pantalla de información sobre una plaza. En esta pantalla se podrán visualizar las imágenes asociadas a la plaza de parking así como sus características principales. También se mostrará información sobre el propietario de la plaza. Para el desarrollo de esta tarea tenemos dos retrasos.

El retraso del diseño de la interfaz de la página de la plaza se retrasó debido a dos causas:

- La primera tiene que ver con un paquete de Angular (Swiper) utilizado para hacer un carousel en la interfaz. Al no haber utilizado nunca dicho paquete, tuve un periodo de aprendizaje más largo de lo esperado lo que hizo que me demorase con esa parte del diseño.
- La segunda causa se debe a otro paquete de Angular (ng-circle-progress) utilizado para hacer una representación de un progreso cualquiera con la imagen de un círculo. El problema aquí es que la herramienta pide un rango para calcular el porcentaje de círculo que se carga que en nuestro caso sería de 0 a 5 ya que iba a usarse para la representación de la valoración de la propiedad y el propietario. Sin embargo el mínimo permitido era de 0 a 50 por lo que tuve que crear métodos para cambiar la interfaz del círculo de progreso. Estos métodos no funcionaron debido a que se estaban ejecutando antes de que cargase la interfaz. Añadiendo dichos métodos al método interno de Angular ngAfterContentChecked los cambios se efectuaron correctamente.

El otro problema a la hora de desarrollar la pantalla de información de una plaza ocurrió en la tarea de conectar la interfaz con LoginAPI y AlquileresAPI. La implementación se retrasó por 2 horas debido a que la consulta a LoginAPI no retornaba el resultado correcto a pesar de hacerse bien la consulta. El problema estaba en que hacíamos una consulta a usuarios que tenían una relación con la tabla Owner, es decir, hacíamos una consulta a usuarios de la tabla Users que tenían una relación 1 a 1 con la tabla Owner. Al haber diseñado incorrectamente la relación entre ambas tablas se hacían consultas circulares (recursivas) entre User y Owner. Esto significa que cuando imprimía los campos de User, imprimían también los campos de Owner, que a su vez tenía campos de User, y así recursivamente. La solución se encontraba en ignorar el JSON de User que se generaba en los campos de Owner. Esto se consigue con la etiqueta @JsonIgnoreProperties(“user”). Tras añadir la etiqueta los campos se retornaban correctamente.

| Historia de Usuario  | Peso | Issue   | Estimate (h) | Spend (h) | Repository      |
|--|------|---|--------------|-----------|-----------------|
| Como arrendatario quiero poder filtrar las plazas disponibles para encontrar la que más se ajuste a mis preferencias | 3    | Validación de campos de filtrado  | 7            | 6         | IonicRESTClient |
|  |      | Conectar el front con los métodos de la API de filtrado   | 8            | 10        |                 |
| Como arrendatario quiero poder ver las plazas disponibles a mi alrededor   | 6    | Remodelación de la interfaz   | 4            | 4         |                 |
| Como arrendatario quiero poder ver toda la información de una plaza  | 5    | Diseño de la interfaz de la página de la plaza  | 10           | 11        |                 |
|  |      | Conectar la interfaz de la plaza con LoginAPI y con AlquileresAPI para recoger los datos de la DB | 4            | 6         |                 |
|  |      | Documentación de la memoria   | 2            | 2         |                 |

Tabla 7.5: Sprint 4

## 7.5. Sprint 5 (20/04/2022 - 03/05/2022)

Como puede apreciarse, este Sprint comienza una semana después a la finalización del Sprint 4. Esto se debe a que del 12/04/2022 al 19/04/2022 hubo una pausa en el desarrollo del proyecto.

A lo largo del Sprint 5 se realizaron dos tareas principales. Una es la de crear la interfaz del perfil de un usuario de tipo cliente. El objetivo de esta tarea es que el cliente tenga un apartado en la aplicación en el que poder ver la información sobre su perfil y donde le aparezca la opción de añadir una tarjeta bancaria. Esta opción es muy importante debido a que al único tipo de usuario que se le requiere una cuenta bancaria para el registro es a los propietarios. El cliente, deberá de añadir la cuenta bancaria posteriormente al registro en la aplicación, desde su perfil. Es por esto que desde el perfil del cliente podremos acceder a un formulario de registro de una cuenta bancaria. A su vez, en el perfil de cliente también aparece la opción de cerrar sesión, que eliminara la información almacenada en caché sobre el perfil en uso y volverá a la página de inicio.

La otra tarea que se llevó a cabo fue la remodelación de la aplicación. Tras un Daily Meeting se acordaron cambios en la base de datos con todo lo que eso conllevaba. Es decir, hubo que cambiar desde los métodos de registro hasta los componentes backend de LoginAPI. Todos estos cambios estaban orientados a la conexión con la API de pagos (PagosAPI). Como la herramienta que vamos a utilizar para los pagos pedía muchos más datos del usuario de los que le ofrecíamos, nos vimos obligados a rediseñar la base de datos y su contenido.

En este Sprint, en la tarea de adaptar el modelo de cliente en la API se tardó una hora más de lo estimado en un principio debido a que se introdujeron pequeños cambios que tuvieron una reacción en cascada, haciendo que el método de registro tuviese que ser modificado. Finalmente se solucionó. Sin embargo, al principio tenía el problema de que no se actualizaban las dos tablas implicadas en un registro (User y Owner o User y Client), actualizándose tan solo User. La solución fue añadir CascadeType.ALL en el tipo de unión, lo que extiende las operaciones sobre una entidad a sus hijos.

| Historia de Usuario  | Peso | Issue  | Estimate (h) | Spend (h) | Repository      |
|--|------|--|--------------|-----------|-----------------|
|  |      | Documentación de la memoria                                    | 2            | 2         |                 |
| Como arrendatario quiero poder filtrar las plazas disponibles para encontrar la que más se ajuste a mis preferencias | 3    | Conectar el front con los métodos de la API de filtrado (cont) | 2            | 2         | IonicRESTClient |
| Como arrendatario quiero poder ver toda la información de una plaza  | 5    | Añadir símbolo de verificación a la interfaz de parking        | 2            | 1         |                 |
| Como usuario quiero poder ver mis datos personales y editar los más relevantes                                       | 6    | Crear interfaz de perfil Cliente                               | 6            | 6         |                 |
|  |      | Adaptar el modelo de Cliente en la API                         | 3            | 4         | LoginAPI        |
|  |      | Remodelación completa del modelo de BD y derivados             | 10           | 10        | IonicRESTClient |
|  |      | Remodelación completa del modelo de BD en backend              | 5            | 5         | LoginAPI        |
| Como arrendatario quiero poder realizar un alquiler  | 5    | Conectar Client y Owner con PagosAPI                           | 5            | 5         | IonicRESTClient |

Tabla 7.6: Sprint 5

## 7.6. Sprint 6 (04/05/2022 - 17/05/2022)

En el Sprint 6, el desarrollo estuvo centrado en crear la interfaz del perfil de un propietario. En dicha interfaz el propietario podría ver su información personal, así como un botón desde el que poder cerrar sesión.

Otra de las tareas principales fue el crear una interfaz desde la que un propietario podría registrar una nueva propiedad. Para ello, tendría que introducir en un formulario una serie de características de la plaza, así como imágenes de la propiedad. Esta última parte fue de las más complicadas de implementar, por lo que hubo un retraso de dos horas en dicha tarea. Esto se debe al desconocimiento para tratar con las imágenes. Ya que nunca había implementado la funcionalidad de tomar imágenes del propio dispositivo, esta funcionalidad se complicó bastante llegando a dar errores y fallos al probar la aplicación en un dispositivo móvil real. Finalmente, se probó dicha funcionalidad en un dispositivo Android con un resultado exitoso. El problema con esta implementación inicial es que solo se permitía seleccionar las imágenes de una en una, cuando el objetivo era que el usuario seleccionase el número de imágenes que quisiera registrar de una sola vez, sin tener que estar seleccionando el botón una vez por imagen. Se dejó para Sprints posteriores esta mejora.

Inicialmente, tuve varias charlas con el otro desarrollador debido a que no sabíamos con que formato trabajar con las imágenes. Finalmente, vimos trabajar con un formato base64 facilitaba mucho el trabajo en el frontend a la hora de registrar las imágenes en base de datos. Sin embargo, el formato en la parte del registro de imágenes iba a ser distinto por

razones de optimización de la aplicación.

| Historia de Usuario  | Peso | Issue   | Estimate (h) | Spend (h) | Repository      |
|--|------|---|--------------|-----------|-----------------|
| Como usuario quiero poder ver mis datos personales y editar los más relevantes | 6    | Crear interfaz de perfil Owner  | 6            | 5         | IonicRESTClient |
| Como arrendador quiero poder alquilar bienes                                   | 6    | Crear interfaz de añadir Property   | 6            | 6         |                 |
|  |      | Documentación de la memoria   | 3            | 3         |                 |
|  |      | Planificación general del proyecto  | 3            | 3         |                 |
| Como arrendador quiero poder alquilar bienes                                   | 6    | Permitir la subida de imágenes desde el dispositivo móvil a la aplicación | 10           | 12        | IonicRESTClient |
| Como usuario quiero poder ver mis datos personales y editar los más relevantes | 6    | Unión de PagosAPI con front de perfiles de usuario                        | 7            | 7         |                 |

Tabla 7.7: Sprint 6

## 7.7. Sprint 7 (18/05/2022 - 31/05/2022)

Entre las tareas principales desarrolladas en este Sprint, se encuentran la de mostrar las imágenes de una propiedad en la pantalla de información de la plaza, crear las interfaces de reserva y crear la interfaz de las propiedades de un propietario.

Yendo por orden, para la primera tarea existe un problema. Para entender el problema, debemos saber como se almacenan las fotos. En primer lugar, cuando el propietario selecciona las fotos al registrar la propiedad, estas se convierten a base64. Una vez formateadas, se pasan mediante un método de AlquileresAPI para ser almacenadas. Ya en el backend, lo que se hace es convertir la imagen de base64 a formato jpg. Lo que almacenamos en base de datos es la ruta relativa a la imagen, que se almacenará en un servidor. Cuando la aplicación alcance su estado final, los servicios (APIs) se han desarrollado deberán de almacenarse en un servidor para funcionar. En ese momento, las rutas devueltas si que harán referencia a un servidor funcional. Sin embargo, en este momento ni las APIs ni tampoco las imágenes se almacenan en un servidor por lo que las rutas devueltas por dicho método no son funcionales.

La siguiente tarea fue crear las interfaces para reservar una plaza. Aquí tenemos dos pantallas, la de reserva de una plaza de tipo Libre y la de reserva de tipo Horario. La diferencia entre estos dos tipos de reservas, y por tanto, entre las dos interfaces es que para reservar una plaza de tipo horario, se debe de especificar la fecha y hora de entrada y salida. Para una reserva de tipo libre, se reserva en el momento de entrar en la plaza y finaliza cuando el usuario pague la plaza en función al tiempo que ha utilizado la propiedad.

Para la última tarea, creamos una interfaz en la que se enumeran todas las propiedades que ha creado un propietario, permitiendo eliminar las propiedades ya creadas. Otro cambio que se realizó sobre esta interfaz fue el añadir un botón que nos llevase directamente a la pantalla de crear una propiedad.

| Historia de Usuario   | Peso | Issue   | Estimate (h) | Spend (h) | Repository      |
|---|------|---|--------------|-----------|-----------------|
|   |      | Documentación de las tecnologías utilizadas             | 6            | 6         |                 |
| Como arrendatario quiero poder ver toda la información de una plaza | 5    | Mostrar las imágenes de una Property                    | 6            | 6         | IonicRESTClient |
| Como arrendatario quiero poder realizar un alquiler                 | 5    | Crear interfaz de reservar una Property de tipo horario | 10           | 8         |                 |
|   |      | Crear interfaz de reservar una Property de tipo libre   | 6            | 6         |                 |
| Como arrendador quiero poder alquilar bienes                        | 6    | Crear interfaz de propiedades de un Owner               | 7            | 7         |                 |

Tabla 7.8: Sprint 7

## 7.8. Sprint 8 (01/06/2022 - 14/06/2022)

En el Sprint 8 todas las funcionalidades principales de la aplicación estaban completadas. Sin embargo, todavía faltaba perfeccionar los métodos y añadir comprobaciones a las acciones. Esa fue la tarea principal desarrollada en este Sprint.

Iniciamos el Sprint afianzando el enrutamiento de la aplicación. Esto es que un usuario no pueda acceder a ciertas páginas de la aplicación dependiendo de su estado. Por ejemplo, que no pueda acceder desde la pantalla de login hasta la pantalla del mapa sin haberse autenticado. Otras de las tareas respecto al enrutado era que un cliente no pudiese acceder a pantallas exclusivas de un propietario y viceversa. La razón por la que la estimación original de esta tarea difiere en una hora del tiempo que se ha tardado en realizarla ha sido por errores y desconocimiento a la hora de afianzar el enrutado. En particular, en la parte que respecta a que un Cliente no acceda a pantallas específicas de un Propietario y viceversa. El problema principal era que se utilizaba Storage para almacenar datos de la sesión para esta parte. El problema principal con esto es que varios de los métodos de los que dispone Storage para obtener los datos guardados se ejecutan asíncronamente. Además, el objeto devuelto era un Promise<any>, cuando lo que necesitábamos era un boolean (o string representando un boolean). La solución fue optar por utilizar localStorage para almacenar parte de los datos de sesión.

El resto de tareas del Sprint fueron la de diseñar una interfaz para cuando un cliente tuviese una reserva activa, poder ver su estado. La utilidad de esta pantalla es, para las reservas de plazas de tipo horario, ver el tiempo restante de la reserva y la información básica como puede ser el precio de la misma. Para las reservas de tipo libre, se mostrará el



coste actual de la reserva. Recordemos que en este tipo de reservas no hay una fecha de fin especificada, esta se genera cuando el usuario finaliza la reserva. Es por esto que no se puede poner un precio fijo a dicha reserva, y por lo que se mostrará el precio actual en el momento de la consulta.

El resto del trabajo del Sprint trató sobre conectar métodos de AlquileresAPI con la parte front.

| Historia de Usuario   | Peso | Issue   | Estimate (h) | Spend (h) | Repository      |
|---|------|---|--------------|-----------|-----------------|
|   |      | Conexión de interfaces con AlquileresAPI                | 6            | 6         | IonicRESTClient |
| Como usuario quiero poder identificarme para acceder a mi cuenta personal | 4    | Afianzar el enrutado de la aplicación y uso de JWT      | 6            | 7         |                 |
| Como arrendatario quiero poder ver el estado de mi reserva                | 4    | Crear interfaz para ver el estado de una reserva activa | 8            | 8         |                 |
|   |      | Conexión con AlquileresAPI y refactorización de métodos | 8            | 8         |                 |
|   |      | Documentar riesgos y nuevas tecnologías                 | 7            | 7         |                 |

Tabla 7.9: Sprint 8

## 7.9. Sprint 9 (15/06/2022 - 28/06/2022)

Este es el último Sprint de la planificación inicial. En el se han abarcado las últimas implementaciones y, sobre todo, la revisión del código con sus correspondientes refactorizaciones y solución de errores.

Al principio del Sprint se implementó la parte de obtener las imágenes de las propiedades del servidor. Sin embargo, al no estar las APIs subidas al servidor, las rutas devueltas por el método no llevan a ninguna imagen. Es por esto que se han utilizado imágenes fijas, a pesar de estar implementado el método en la aplicación.

Más adelante se realizaron conexiones con la API de alquileres para hacer uso de unos métodos que no habían sido implementados todavía, como puede ser el encargado de comprobar la disponibilidad de una plaza de tipo Libre. Al utilizar este método en primer lugar se pensó que funcionaba correctamente ya que pasó las pruebas iniciales. Sin embargo, con el uso de la aplicación descubrí que tenía un fallo al no reconocer algunas de las reservas y ofrecer información errónea. Tras una reunión con el otro desarrollador del proyecto, logramos solucionar el problema y hacer que la comprobación funcionase perfectamente.

A continuación, probé la API de envío de correos electrónicos (EmailAPI) que no funcionaba. El error se encontraba en que para enviar correos, la API accedía a una cuenta de Google mediante una contraseña en texto plano. Al principio del proyecto esto funcionaba sin problemas, pero el día 30 de Mayo de 2022, Google cambió su política respecto a lo que

consideraba el uso de Aplicaciones Poco Seguras y la cuenta de Google. Por esta razón, Google dejó de permitir el inicio de sesión a través de una aplicación mediante una contraseña en texto plano.

Para solucionar este problema se siguieron los siguientes pasos:

1. **Activar la autenticación en dos pasos:** El objetivo final es activar lo que Google considera una ‘Contraseña de aplicación’. Para poder activarlo, es necesario disponer de la autenticación en dos pasos de Gmail.
2. **Activar la Contraseña de aplicación:** Una vez activada la autenticación en dos pasos, aparecerá una opción para añadir contraseñas de aplicación. Para generar una de estas contraseñas, lo primero que hay que hacer es escribir el nombre de la aplicación que va a hacer uso de este servicio. Una vez definido el nombre, se generará una aplicación de 16 dígitos única para esa aplicación. Esta será nuestra contraseña de aplicación.
3. **Cambiar la autenticación en Nodemailer:** Cómo hemos explicado previamente, para enviar correos electrónicos lo que hace EmailAPI a través de Nodemailer es enviar el correo a la cuenta de Google especificada y a través de dicha cuenta enviar el correo al usuario final. Para poder hacer esto, tenemos que poner usuario y contraseña en nuestra API. Sin embargo, para que Google nos deje acceder a nuestra cuenta y no deniegue el acceso, utilizaremos la contraseña de aplicación en vez de la contraseña que utilizamos para acceder a nuestra cuenta. De este modo, Google permitirá el acceso sin ningún tipo de problema.

El resto del Sprint se utilizó para continuar con la documentación del trabajo y de la aplicación en la memoria.

| Historia de Usuario   | Peso | Issue   | Estimate (h) | Spend (h) | Repository      |
|---|------|---|--------------|-----------|-----------------|
| Como arrendatario quiero poder ver toda la información de una plaza | 5    | Recuperar fotos de las propiedades del servidor       | 6            | 6         | IonicRESTClient |
| Como arrendatario quiero poder realizar un alquiler                 | 5    | Comprobar la disponibilidad de la plaza al alquilarla | 3            | 3         |                 |
|   |      | Solucionar problema de la disponibilidad              | 2            | 2         |                 |
|   |      | Solucionar problemas con EmailAPI                     | 4            | 5         | EmailAPI        |
|   |      | Refactorización de código                             | 10           | 10        | IonicRESTClient |
|   |      | Documentación de la memoria                           | 10           | 10        |                 |

Tabla 7.10: Sprint 9

## Capítulo 8

# Conclusiones

### 8.1. Cumplimiento de objetivos y adecuación del proyecto y sus herramientas

#### 8.1.1. ¿Se han cumplido los objetivos?

Si recapitulamos hasta la sección de los objetivos, podemos ver que el objetivo principal era crear una aplicación dedicada a la gestión de los alquileres de plazas de garaje. Con la aplicación finalizada, podemos decir que las interfaces de las que dispone cumplen con la funcionalidad requerida. Existen interfaces para cada caso de uso, posibilitando el trabajo de ambos tipos de usuarios:

- **Arrendadores:** Son las personas que ponen un bien en alquiler. La aplicación les provee una interfaz para registrar las plazas que deseen poner en alquiler. También se permite obtener toda la información de una plaza así como la eliminación de la misma. Los arrendadores también tienen un soporte para los pagos de los importes de los pagos por los alquileres.
- **Arrendatarios:** Los arrendatarios son las personas que toman un bien en arrendamiento. Los arrendatarios pueden ver todas las plazas disponibles registradas en la aplicación y elegir una para alquilarla siempre y cuando cumplan con los requisitos establecidos. Una vez reservada la plaza pueden ver el estado de su reserva.

A parte de comprobar los objetivos de ambos tipos de usuarios, también se han cumplido los objetivos para ambos tipos de usuario. Dentro de estos objetivos se contemplaba que cada usuario pudiese registrarse en la aplicación para posteriormente autenticarse y hacer uso de ella. También se puede cerrar una sesión iniciada, a parte de dar soporte a los usuarios que hayan olvidado su contraseña de inicio de sesión.

En resumen, podemos concluir que la aplicación cumple con los objetivos especificados inicialmente.

### 8.1.2. ¿En que medida ha funcionado la metodología de trabajo elegida?

Para el desarrollo y organización de este proyecto se ha elegido una metodología de trabajo Ágil: Scrum. Con una metodología de trabajo Ágil se busca la funcionalidad y la rapidez frente a la documentación y a la planificación. Dado a que el desarrollo del proyecto global se ha dividido entre dos desarrolladores, necesitábamos una metodología que favoreciese la comunicación entre los diferentes componentes del equipo, y para esto, seguir una metodología Scrum tiene las ventajas de establecer reuniones tanto diarias como una por Sprint.

Gracias a Scrum hemos podido mantener conversaciones junto a la tutora de ambos TFG para que ofreciese una retroalimentación del trabajo desarrollado. En estas reuniones se revisaba el trabajo realizado y se especificaba el trabajo a realizar, al mismo tiempo que se resolvían las posibles dudas que surgiesen durante la codificación.

En resumen, realizar un desarrollo paralelo de una aplicación puede ser muy complicado. Si no se siguen unas reglas que obliguen a la comunicación entre desarrolladores y a comparar y recibir retroalimentación del trabajo realizado, el programa podría salirse de control causando inconvenientes como un desarrollo mucho más avanzado que otro, error en las comunicaciones de los microservicios entre otras cosas.

Sin embargo no se han aprovechado todas las ventajas de la metodología Scrum. Según Scrum, en cada Sprint se debe de tener un producto entregable del proyecto. Teniendo en cuenta que no había un horario de trabajo marcado si no que cada desarrollador seguía un horario completamente flexible, había ocasiones en las que las reuniones diarias se aplazaban, lo que dificultaba la finalización de ciertas funcionalidades. Esto daba lugar a no tener una versión completa o entregable al final de cada Sprint.

Como conclusión, podemos decir que sí que nos hemos podido adaptar en su mayoría a las reglas que establece el marco de trabajo Scrum. La decisión de utilizar este marco ha sido acertada ya que ha obligado a mantener un nivel de comunicación elevado, haciendo que el desarrollo no se descontrolase.

### 8.1.3. ¿Se han adecuado las tecnologías elegidas al trabajo realizado?

La principal tecnología que se ha utilizado en este proyecto ha sido Ionic. Este framework sí que se ha adaptado perfectamente a lo esperado. Gracias a que utiliza Angular (tecnología utilizada previamente durante el curso académico) la curva de aprendizaje no ha sido muy elevada. Además se ha podido comprobar que compilando el código Ionic con Cordova se podía construir una aplicación ejecutable tanto en Android como en iOS.

Respecto al resto de tecnologías utilizadas, la más destacable es Spring Framework, utilizada para el desarrollo de LoginAPI. En este caso ocurre igual que con Ionic, al haber utilizado este framework previamente, tan solo hubo que adaptar el trabajo y los conocimientos realizados en el pasado al microservicio deseado para el proyecto.

El otro microservicio, EmailAPI utilizaba Node.js. Sin embargo, Node.js se adaptó perfectamente al desarrollo. Pese a que este microservicio es más pequeño que LoginAPI, su realización fue más costosa por el uso de nuevas tecnologías como Nodemailer. Nodemailer se acabó adaptando bien al proyecto pero su uso requirió un periodo de aprendizaje más largo.

Sobre el resto de tecnologías no hay mucho más que añadir. Mapbox podría ser una de las que más tiempo de aprendizaje ha requerido pero en general todas se han adaptado correctamente al proyecto y han cumplido su función con éxito.

### 8.1.4. ¿Se ha elegido una arquitectura adecuada?

Como se ha explicado previamente, se ha elegido una arquitectura basada en microservicios. Esta decisión ha traído tanto ventajas como desventajas para el proyecto.

En cuanto a las ventajas, la división del proyecto en microservicios ha favorecido la independencia del trabajo de ambos desarrolladores. Gracias a que cada uno se encargó de microservicios distintos, pudimos mantener un desarrollo prácticamente independiente. La única parte que rompía con dicha independencia era la comunicación entre los microservicios. Al estar todos conectados con el frontend, era necesario establecer unas interfaces para que la comunicación se diese sin problemas. Esta división también favorece la modularidad del proyecto, y el aislamiento de los componentes. De esta forma, cuando ocurrían fallos en alguno de los componentes, estos no afectaban globalmente a la aplicación.

Sin embargo, los microservicios también tenían desventajas. Debido a que se trata de una aplicación “pequeña” en cuanto al número de componentes y servicios. Es posible que la arquitectura elegida tenga más sentido en un futuro, junto con las mejoras que se explicarán en la siguiente sección.

En resumen, no considero que la elección de una arquitectura basada en microservicios haya sido un error. Sin embargo, se habría visto potenciada en un futuro junto con nuevos componentes que aumentarían el tamaño de la aplicación.

## 8.2. Líneas de trabajo futuras

En esta sección se explorarán diferentes vías de mejora de la aplicación, así como su salida a mercado.

### 8.2.1. Mejoras

Pese a que el estado actual de la aplicación permite un uso completo de la aplicación, existe la posibilidad de añadir mejoras que completen y mejoren su estado actual:

- **Seguridad:** La aplicación dispone de una seguridad básica que evita prácticas maliciosas. Sin embargo, debido a que se trata con información sensible como datos bancarios, si quisiéramos que la aplicación pasase a estar disponible al público debería de realizarse un estudio completo de la seguridad de la aplicación. Esto comprende desde el cifrar toda la información y asegurar que no se trata con esta información sin filtrar en el frontend, hasta el almacenado de la información en el servidor y su tratamiento.
- **Refactorización del código:** Una vez terminadas ambas partes de la aplicación, deberán de ponerse en común y refactorizar el código de forma que siga unas mismas prácticas. Pese a que la aplicación ya sigue unos comportamientos comunes, habría que extenderlos de forma que el código se homogeneice.

### 8.2.2. Evolución

La aplicación se pensó inicialmente para gestionar los alquileres de plazas de garaje. Sin embargo, este modelo podría extenderse a alquileres de bienes. Es decir, no tan solo poder alquilar plazas de garaje, sino también coches, inmuebles, cajas fuertes, etc. Gracias a la arquitectura en microservicios la evolución de la aplicación reutilizaría la gran mayoría de servicios implementados, ahorrando tiempo de desarrollo y favoreciendo la adaptación al cambio.

### 8.2.3. Modelo de negocio

La idea de este proyecto es que sea subido a una plataforma de descarga de aplicaciones para que pase a estar disponible al público. Para ello deben de haberse cumplido por lo menos las mejoras indicadas anteriormente. Una vez subido, es cuestión de que los usuarios comiencen a hacer uso de la aplicación para que esta empiece a funcionar y que aumente progresivamente la oferta de bienes.

Los beneficios de la aplicación se obtendrían de pequeñas comisiones en los pagos de los alquileres. Un porcentaje del importe iría redirigido a la cuenta del proyecto. De esta forma, un porcentaje de los beneficios podría reinvertirse tanto en mantener la aplicación como en mejorarla.

También se ha hablado de extender el modelo de forma que podría añadirse cualquier tipo de bien que encaje con el formato de la aplicación.

# Bibliografía

- [1] Angular. Angular Docs. <https://angular.io/guide/what-is-angular>. Accessed: 2022-06-29.
- [2] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. Agile Manifesto. <http://agilemanifesto.org/>. Accessed: 2022-06-29.
- [3] bootsoon. ng-circle-progress. <https://github.com/bootsoon/ng-circle-progress/blob/master/projects/ng-circle-progress/README.md>. Accessed: 2022-06-29.
- [4] CryptoJS. CryptoJS. <https://cryptojs.gitbook.io/docs/>. Accessed: 2022-06-29.
- [5] Romana Gnatyk. Microservices vs Monolith: which architecture is the best choice for your business? <https://www.n-ix.com/microservices-vs-monolith-which-architecture-best-choice-your-business/>. Accessed: 2022-06-29.
- [6] Ionic Team. Ionic Docs. <https://ionicframework.com/docs>. Accessed: 2022-06-29.
- [7] Mapbox. Mapbox GLJS. <https://docs.mapbox.com/mapbox-gl-js/api/>. Accessed: 2022-06-29.
- [8] OpenJS Foundation y jQuery. jQuery API. <https://api.jquery.com/>. Accessed: 2022-06-29.
- [9] John Prabhu. Model View Controller. <https://techaffinity.com/blog/mvc-architecture-benefits-of-mvc/>. Accessed: 2022-06-29.
- [10] SASS-Team. SASS Documentation. <https://sass-lang.com/documentation>. Accessed: 2022-06-29.
- [11] Spring. Spring.io. <https://spring.io/>. Accessed: 2022-06-29.
- [12] Swiper. Swiper. <https://swiperjs.com/>. Accessed: 2022-06-29.
- [13] Universidad de Valladolid. Proyecto docente del trabajo de fin de grado 2019-2020 (Mención Ingeniería de Software). [https://alojamientos.uva.es/guia\\_docente/uploads/2019/545/46976/1/Documento.pdf](https://alojamientos.uva.es/guia_docente/uploads/2019/545/46976/1/Documento.pdf). Accessed: 2022-06-29.





# Apéndice A

## Manuales

### A.1. Manual de despliegue e instalación

En este apartado se explicará el procedimiento para descargar la aplicación en un entorno de desarrollo.

#### A.1.1. Frontend

La parte correspondiente al frontend es llevada a cabo por el framework Ionic. Para instalar y poder ejecutar Ionic necesitaremos tanto Node como su gestor de paquetes, npm. Para esta aplicación se han utilizado las siguientes versiones:

- **Ionic:**
  - **Ionic CLI:** 6.18.1
  - **Ionic Framework:** @ionic/angular 6.1.5
  - **@angular-devkit/build-angular:** 13.3.5
  - **@angular-devkit/schematics:** 13.0.4
  - **@angular/cli:** 13.0.4
  - **@ionic/angular-toolkit:** 5.0.3
- **Capacitor:**
  - **Capacitor CLI:** 3.4.1
  - **@capacitor/android:** 3.5.1
  - **@capacitor/core:** 3.4.1
  - **@capacitor/ios:** 3.5.1

- **Utility:**
  - **cordova-res:** not installed globally
  - **native-run:** 1.5.0
- **System:**
  - **NodeJS:** v14.18.0
  - **npm:** 6.14.15
  - **OS:** Windows 10

Una vez tengamos instalado Node y npm, procedemos a instalar Ionic con el siguiente comando:

```
npm install -g ionic
```

A continuación, desplegaremos un servidor de desarrollo local. Este servidor se ejecuta en el buscador y reaccionará ante cualquier cambio en los fuentes del código, recargando la página con la versión actualizada. Para desplegar el servidor, ejecutaremos el siguiente comando:

```
ionic serve
```

### A.1.2. Backend

Podemos dividir los microservicios en dos tipos: Desarrollados con Spring y desarrollados con Node. EmailAPI es el único microservicio que ha sido creado con Node por lo que para desplegarlo tan solo necesitaremos ejecutar el siguiente comando con el que el servicio comenzará esperar una petición (en nuestro caso, de tipo POST):

```
npm start
```

Para el resto de microservicios, utilizamos Spring. Para crear el microservicio de autenticación, creamos el cuerpo del servicio desde el inicializador de Spring.<sup>1</sup> Una vez que hemos creado la aplicación, para inicializarla ejecutaremos el código main de la aplicación, que ejecutará el método *run* de la clase *SpringApplication*.

Respecto al microservicio desarrollado como parte de este proyecto (LoginAPI), se han utilizado las siguientes versiones tanto para su desarrollo como para su despliegue:

- **Java:** 16.0.2

---

<sup>1</sup><https://start.spring.io/>

- **Spring-boot:** 2.6.3
- **io.jsonwebtoken:** 0.9.1

io.jsonwebtoken es una de las dependencias de LoginAPI que se utiliza para generar JWT.

### A.1.3. Base de datos

La base de datos se encuentra alojada en un servidor. Para conectarnos a ella, necesitamos establecer una conexión ssh al servidor. Una vez establecida la conexión y autenticándose podremos hacer uso de las funcionalidades que ofrece la base de datos. Para la gestión de la base de datos se ha utilizado MySQL Workbench, una herramienta para la gestión y diseño de bases de datos. Se ha utilizado la versión 8.0.26 de MySQL.

## A.2. Manual de mantenimiento

A continuación se detallan las herramientas necesarias para el mantenimiento de la aplicación.

En primer lugar, necesitaremos un editor de código. Pese a que cualquier editor de código es válido, para el desarrollo de este proyecto se ha utilizado Visual Studio Code.

Posteriormente tendremos que descargar el código de los componentes que queramos modificar. Por motivos de seguridad debido a que ambos microservicios contienen datos privados sobre credenciales, estos dos códigos se ofrecerán en un comprimido .zip con dichos datos suprimidos. En cuanto al código frontend, podrá descargarse desde el repositorio indicado.

Una vez descargado el código, en caso de querer hacer uso de la base de datos sería necesario conectarse al servidor en el cual se encuentra alojada e introducir las credenciales de un usuario válido. Por seguridad, las credenciales de la base de datos no aparecerán en esta memoria.

Finalmente, si se quiere ejecutar la aplicación en local será necesario tener instalados los entornos indicados en el anexo anterior. Principalmente Ionic y Node.js. Una vez iniciados los entornos e introducidas las credenciales omitidas en el código ofrecido, la aplicación estará lista para ser probada.

## A.3. Manual de usuario

A continuación se ofrece un recorrido por todas las interfaces de la aplicación, así como por sus distintas funcionalidades para ambos tipos de usuarios.

### A.3.1. Inicio de sesión

La interfaz de la Figura A.1 es la primera pantalla que vemos al abrir la aplicación. Desde ella podremos iniciar sesión introduciendo las credenciales en el formulario central. Esta primera versión de la aplicación solo tiene una forma de iniciar sesión por lo que esta pantalla es la única involucrada en la autenticación del usuario.

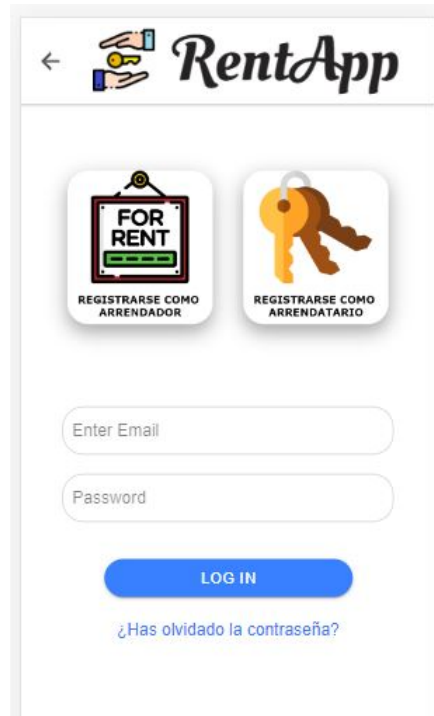


Figura A.1: Pantalla de inicio de sesión

Existe la posibilidad de que a un usuario se le olvide la contraseña de su cuenta, impidiendo así el inicio de sesión. Para solucionar este problema se ha creado una interfaz desde la que poder cambiar la contraseña. Desde la pantalla de login, podemos acceder a la interfaz de la Figura A.2 desde la que el usuario puede introducir su dirección de correo electrónico para que le llegue un mensaje para cambiar su contraseña.

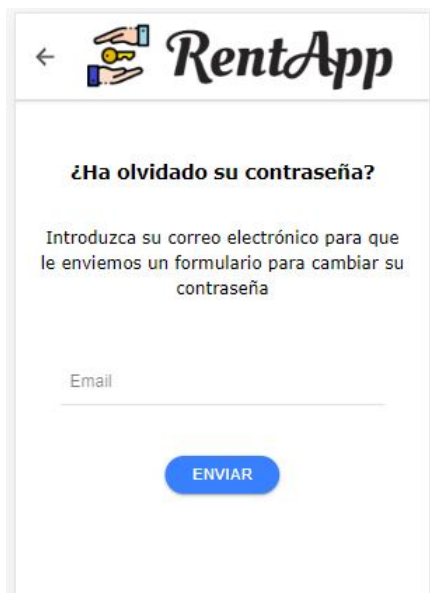
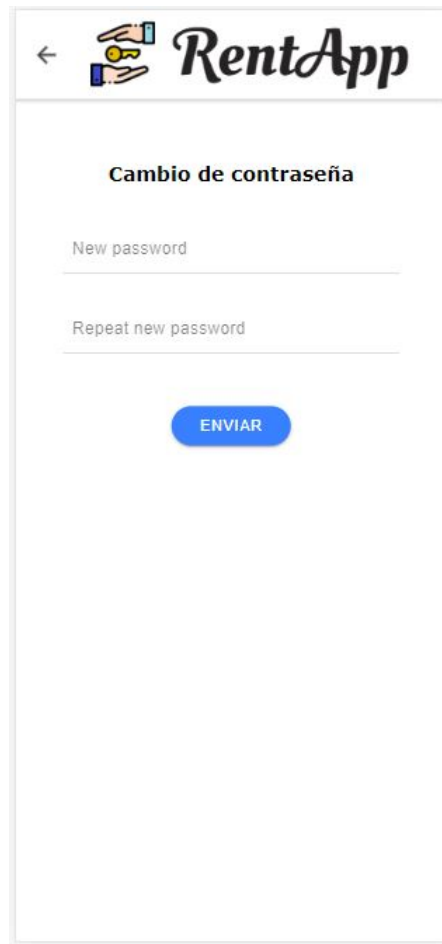


Figura A.2: Pantalla de contraseña olvidada

El correo que se envía al usuario contiene un enlace que nos redirige a la interfaz de cambio de contraseña, desde la que el usuario podrá introducir una nueva contraseña para su cuenta.



The image shows a mobile application interface for changing a password. At the top, there is a navigation bar with a back arrow on the left, a logo consisting of two hands holding a key, and the text 'RentApp'. Below the navigation bar, the title 'Cambio de contraseña' is centered. Underneath the title, there are two input fields: the first is labeled 'New password' and the second is labeled 'Repeat new password'. At the bottom of the form, there is a blue button with the text 'ENVIAR' in white capital letters.

Figura A.3: Pantalla de cambio de contraseña

### A.3.2. Registro

Desde la pantalla de inicio de sesión podemos acceder al registro de un nuevo usuario. Como puede apreciarse, disponemos de dos botones. Cada uno de ellos nos llevará a un formulario distinto, aunque a pesar de ser tipos de usuario diferentes los formularios mantienen bastantes similitudes ya que mantienen muchos atributos comunes. En la Figura A.4 se encuentra la primera pantalla que aparece al pulsar sobre cualquiera de los dos botones. Esta pantalla es genérica para ambos formularios de registro y recoge los datos básicos de un usuario de la aplicación.

The image shows a mobile application registration screen for 'RentApp'. At the top left, there is a back arrow and an icon of hands holding a key. The title 'RentApp' is in a stylized font. Below the title, the word 'REGISTRO' is centered. The form consists of the following fields from top to bottom: 'Nombre', 'Apellidos', 'Ciudad', 'Calle', 'Código Postal', 'DNI', 'Móvil', 'Nacido el:' (with a placeholder 'dd/mm/aaaa' and a calendar icon), 'Email', 'Password', and 'Repeat password'. At the bottom center, there is a blue button with the text 'SIGUIENTE'.

Figura A.4: Primera pantalla de registro

Si continuamos con el registro, nos encontraremos con dos pantallas distintas dependiendo del tipo de registro como se muestra en la Figura A.5. Si el usuario se está registrando como cliente, la pantalla que aparecerá será la de la izquierda. En ella se muestra la política de privacidad, la cual deberá aceptar si quiere hacer uso de la aplicación. A la derecha tenemos la pantalla de registro de un propietario. La pantalla tiene dos partes: la de la política de privacidad (similar a la del cliente) y la de los datos bancarios, en la que el usuario introducirá su número de cuenta y el número de enrutamiento del banco.

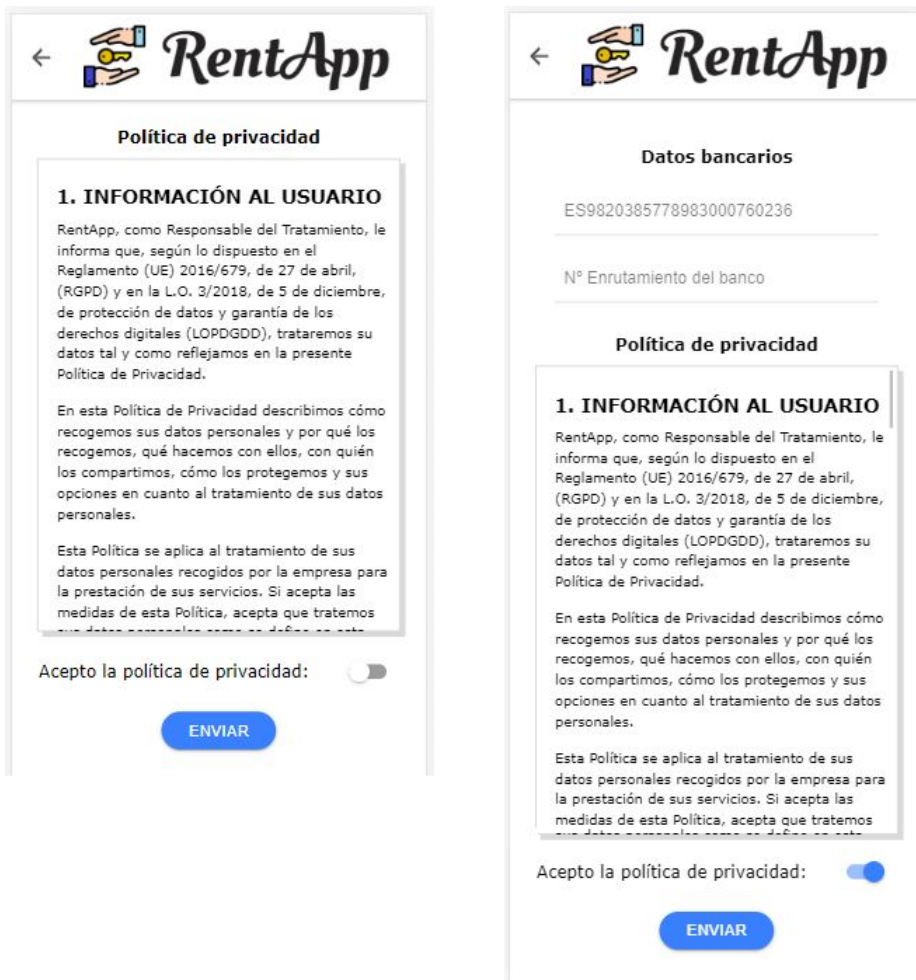


Figura A.5: Segunda pantalla de registro

Una vez que el usuario se ha registrado, le llegará un mensaje por correo electrónico desde el que poder acceder a un enlace que verificará su cuenta. Al acceder al enlace aparecerá la pantalla de la Figura A.6. La pantalla de la izquierda muestra que no ha habido ningún problema y la de la derecha que no se ha podido verificar la cuenta además de mostrar la dirección de correo electrónico del soporte de la aplicación.



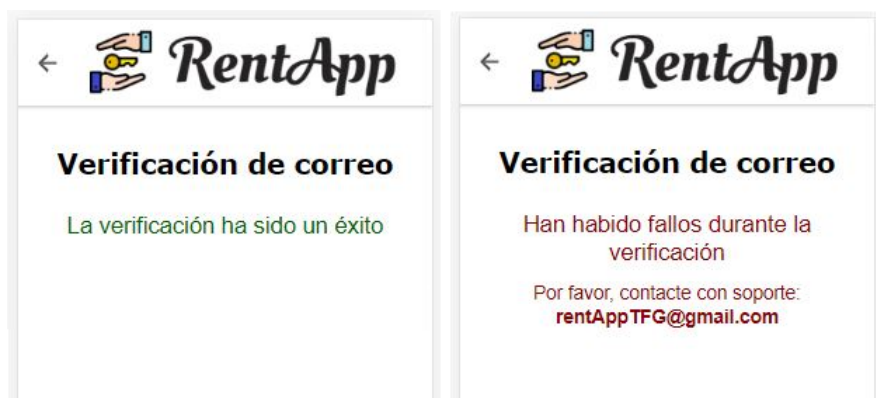


Figura A.6: Pantalla de verificación de usuario

### A.3.3. El mapa

Cuando el usuario se haya registrado e inicie sesión, se encontrará con la pantalla del mapa. En esta pantalla se muestran las plazas registradas en la aplicación como iconos de parking en un mapa. Es una interfaz común a ambos tipos de usuario a la cual también podremos acceder si pulsamos el botón de “Mapa” del menú inferior.

Como se aprecia en la Figura A.7, en la esquina inferior derecha hay un botón. Desde este botón accederemos al formulario de filtrado. La idea del filtro es realizar una selección más específica según las preferencias del usuario sobre las plazas que aparecen en el mapa. El mapa se mostrará.

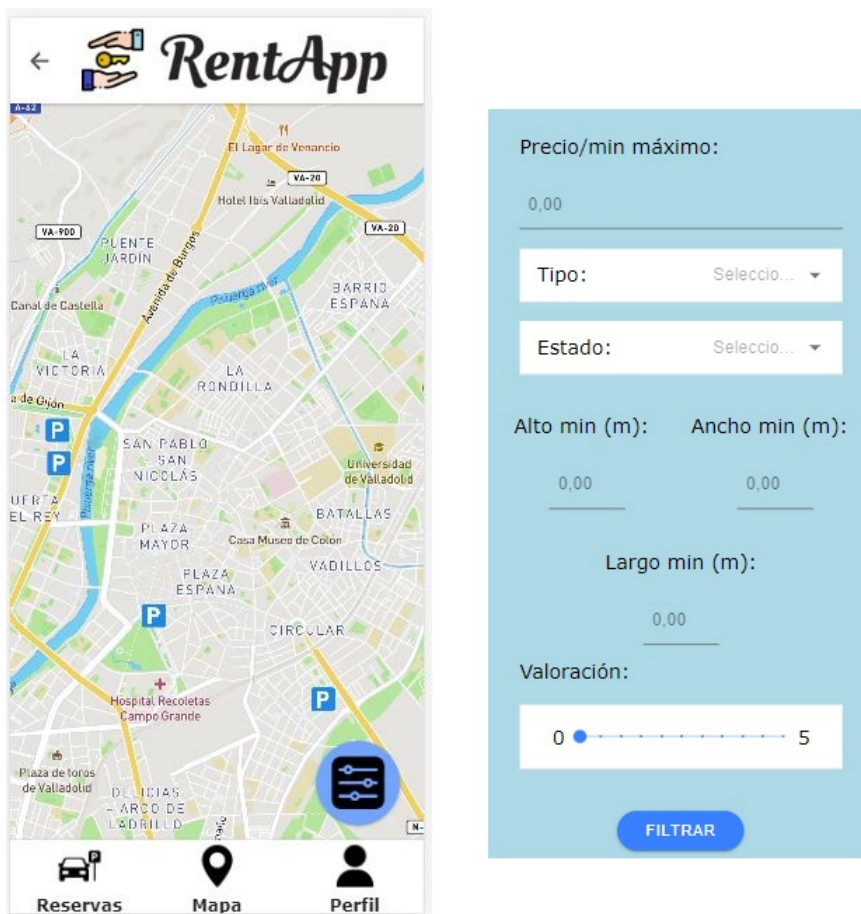


Figura A.7: Pantalla del mapa

#### A.3.4. Plazas de parking y reservas

Si siguiendo con las plazas que aparecen en el mapa, seleccionando una de ellas y pulsando el botón que accede a su información, llegaremos la pantalla de la Figura A.8. En esta pantalla se muestra toda la información de la plaza. En la parte de arriba tenemos un Slider con las fotos de la plaza que el propietario ha subido. A continuación tenemos toda la información de la plaza de parking. En la parte inferior de la pantalla tenemos también alguna información sobre el propietario de la plaza de parking.

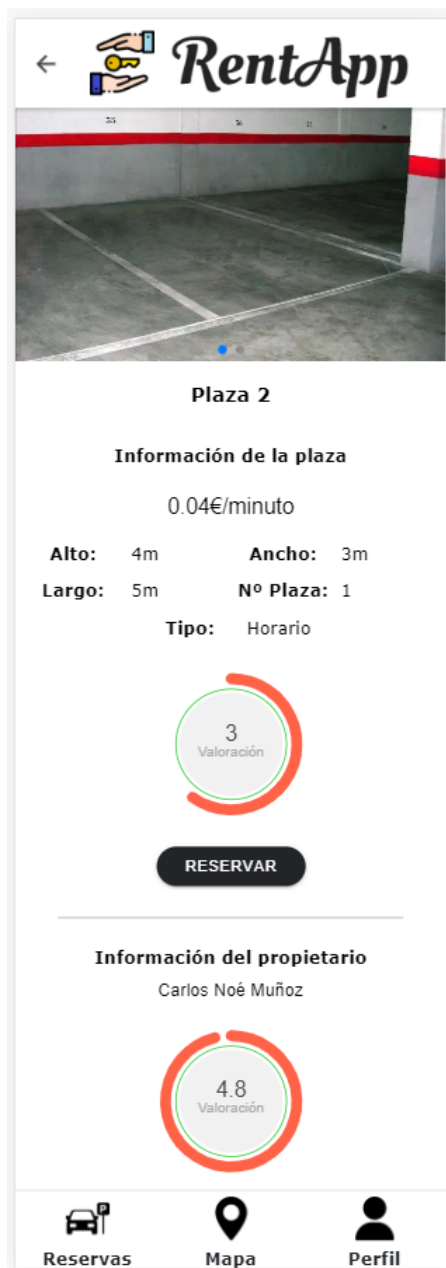


Figura A.8: Pantalla de información de una plaza

Desde la pantalla de parking podemos proceder a efectuar una reserva. Hay dos pantallas para la reserva de una plaza dependiendo del tipo. Esto se debe a que cada reserva requiere de parámetros distintos. A la izquierda de la Figura A.9 podemos ver la pantalla de alquiler de una plaza de tipo horario. Estas reservas requieren especificar la hora de entrada y la

hora de salida de la plaza. En la parte inferior de la pantalla veremos una caja con todas las reservas para esa plaza, de forma que el usuario pueda elegir unos horarios válidos para la reserva.

En la parte derecha tenemos la interfaz para una reserva de tipo libre. Para este tipo de plazas no se especifica ni la hora de entrada ni la hora de salida. El único requisito para reservarlas es que estén libres, por lo que la hora de inicio será cuando el usuario comience a hacer uso de la plaza y la hora de finalización se establecerá a cuando el usuario deje de utilizar la plaza.

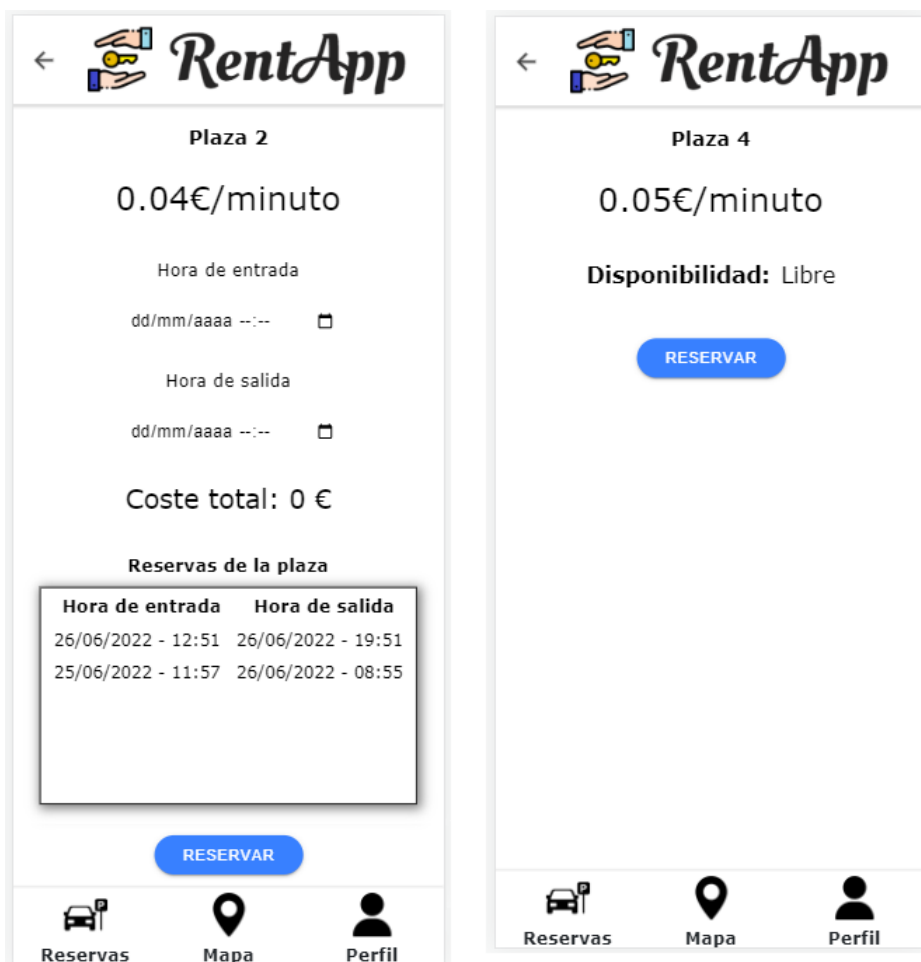


Figura A.9: Pantalla de alquiler de una plaza

### A.3.5. Reservas activas

Una vez que hemos realizado una reserva, es conveniente poder ver el estado actual de nuestra reserva. Para poder ver el estado, tenemos un botón llamado “Reservas” en el menú inferior de la aplicación. Si pulsamos a ese botón estando autenticados como un usuario de tipo cliente, nos llevará a una de las tres interfaces que se muestran en la Figura A.10.

La pantalla de la izquierda se mostrará cuando no tengamos ninguna reserva activa. En esta pantalla, si pulsamos al botón que dice “Comience a reservar!” accederemos a la pantalla del mapa.

En el caso de tener una reserva de tipo Libre, se mostrará la pantalla central. En ella podemos ver la hora en la que ha comenzado la reserva, así como el precio actual de la reserva calculado a partir de la hora de entrada y la hora de la consulta. Cabe destacar que los precios mostrados en estas pantallas son orientativos ya que una vez desarrollado en profundidad el modelo de negocio, se podrían aplicar costes adicionales, como por ejemplo una fianza o un precio fijo en función del tipo de reserva.

Si tenemos una reserva de tipo horario, se mostrará la pantalla de la derecha, en la que vemos todas las características de la reserva, el precio actual que para este tipo de reserva es fijo (no cambiará en el tiempo ya que la hora de salida ha sido establecida previamente) y también el tiempo restante de la reserva.

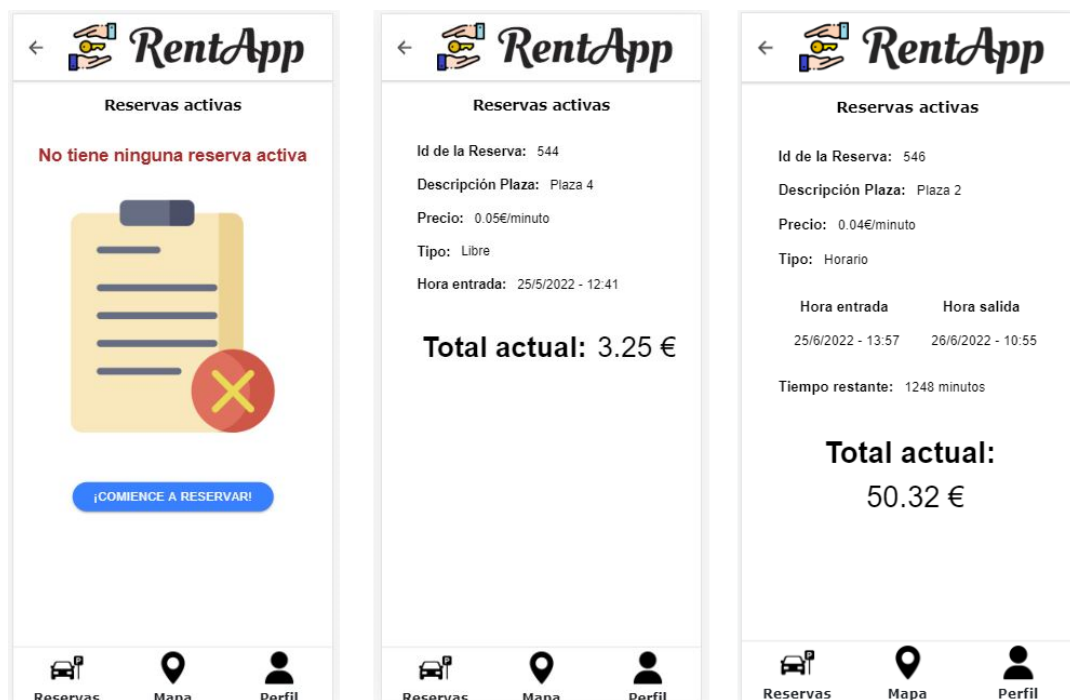


Figura A.10: Pantalla de reservas activas

Cabe destacar que toda la parte de reservar una plaza tan solo es posible si estamos registrados como clientes. Un propietario no puede realizar reservas desde la aplicación. Para ello, debería de crearse otra cuenta de tipo cliente.

### A.3.6. Perfil de usuario

El perfil de usuario de un cliente y de un propietario son ligeramente distintos. En la Figura A.11 tenemos los dos tipos de interfaces. En la interfaz de la izquierda tenemos la pantalla de perfil de un propietario. En ella se muestran datos personales, así como un botón para el cierre de sesión.

En las interfaces del centro y la derecha de la figura tenemos la pantalla de perfil de un cliente. La pantalla de la derecha es la que se mostraría si hubiésemos registrado un método de pago. En la pantalla central se muestra la interfaz para un cliente que no ha registrado un método de pago.

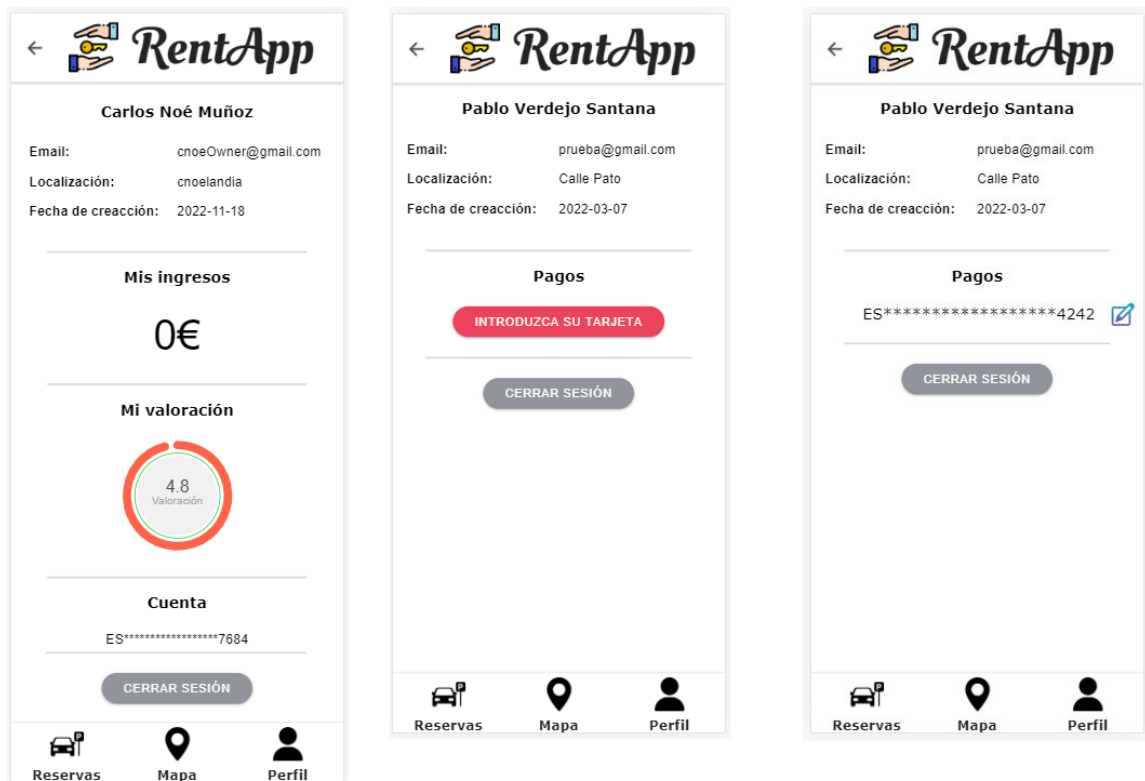


Figura A.11: Pantalla de perfil de usuario

Como podemos apreciar, en la pantalla central disponemos de un botón que nos permite

introducir un método de pago, en el caso de los clientes, una tarjeta bancaria. En el caso de tocar ese botón accederíamos a la interfaz de la Figura A.12, desde la que podemos registrar una tarjeta bancaria introduciendo todos los datos necesarios.

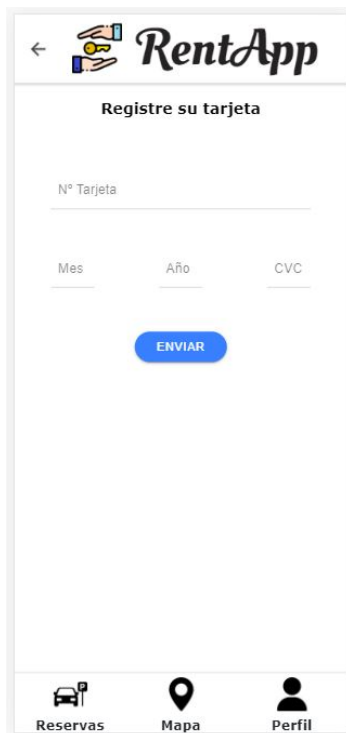


Figura A.12: Pantalla de registrar método de pago

### A.3.7. Creación y gestión de propiedades

Para la creación y gestión de propiedades tenemos que estar autenticados como un usuario propietario. Si accedemos al botón de “Reservas” del menú inferior de la pantalla, veremos la interfaz mostrada en la Figura A.13. Desde esta interfaz se enumera una lista con todas las propiedades que el propietario ha creado. A la derecha de cada propiedad disponemos de dos botones. El botón de la izquierda es un botón de información de la propiedad, que nos llevará a la pantalla de información explicada previamente. El botón de la derecha sirve para borrar una propiedad. En caso de pulsarlo aparecerá un mensaje emergente pidiendo la confirmación y, en caso de confirmar la acción, la propiedad será eliminada.

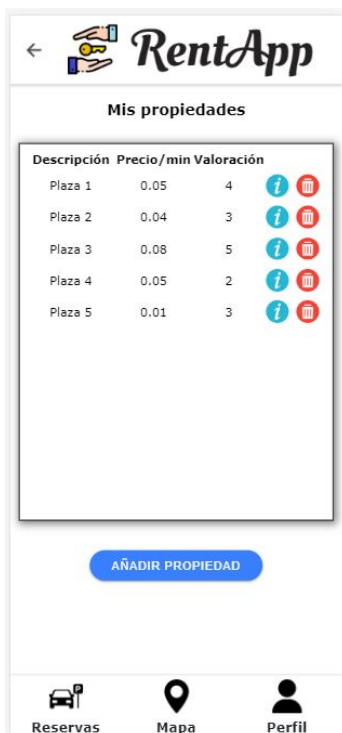


Figura A.13: Pantalla de lista de propiedades de un propietario

También tenemos la opción de crear una nueva plaza pulsando el botón de añadir propiedad. En este caso, accederemos a la interfaz de la Figura A.14. En esta pantalla nos encontramos con un formulario de registro en el que deberemos de especificar las características de la propiedad. A parte, en la parte superior de la pantalla nos encontramos con un botón utilizado para añadir imágenes a la propiedad. Por lo menos debe de añadirse una foto para que la aplicación permita el registro. Una vez creada la propiedad estará lista para usar estando ya activa en el mapa.



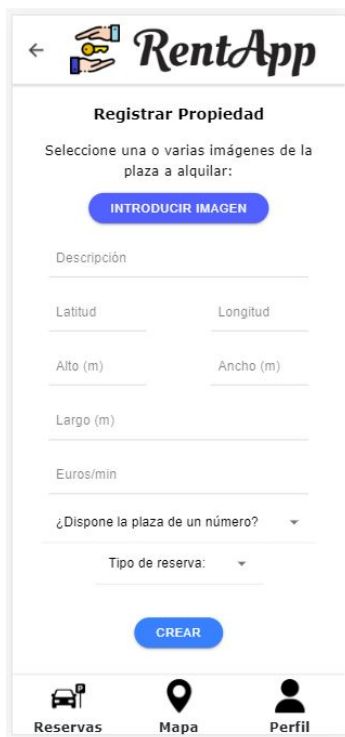


Figura A.14: Pantalla de añadir una propiedad



## Apéndice B

# Resumen de enlaces adicionales

Los enlaces útiles de interés en este Trabajo Fin de Grado son:

- Repositorio del código frontend, LoginAPI, EmailAPI y de la batería de pruebas de IonicRESTClient: <https://drive.google.com/drive/folders/1R4SZew9K4NSgYEW8Dbepx4xQUhH6c3PF?usp=sharing>