Universidad de **Valladolid**

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Ingeniería de Software

Retobici, aplicación para la promoción del uso del servicio de bicicletas municipal

Autor:
Jorge Plaza Lazo

Universidad de **Valladolid**

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Ingeniería de Software

Retobici, aplicación para la promoción del uso del servicio de bicicletas municipal

Autor:
Jorge Plaza Lazo

Tutor:
Mario Corrales Astorgano

A mi familia y a todos los que me han acompañado estos meses.
Gracias por apoyarme en todo momento.

Agradecimientos

Varias personas han ayudado a que este proyecto se realizase de la mejor manera y a tiempo. En primer lugar, agradecer a mi tutor Mario Corrales Astorgano por facilitarme en todo momento el desarrollo de este proyecto y adaptarse a mis necesidades. Gracias también a mis familiares y amigos que a pesar de estar lejos de ellos, me han ayudado en todo momento.

Resumen Uno de los mayores problemas a los que se enfrenta la sociedad actual es el cambio climático y cómo nos afecta como sociedad, individualmente y en su conjunto. Uno de los puntos cruciales en la lucha contra el cambio climático es la forma en que trasladamos a las personas y las mercancías. Así, se está promoviendo el uso del transporte sostenible en las ciudades. Este proyecto pretende desarrollar un sistema de alquiler o gestión de bicicletas que facilite su uso a las personas que no disponen de una propia. Puede ser integrado tanto por empresas públicas como privadas.

El sistema permite gestionar la reserva, el alquiler y la utilización de las bicicletas, que se distribuyen por una ciudad en diferentes paradas. Además, se integra un sistema de gamificación, basado en la obtención de puntos al utilizar la aplicación, que luego se pueden canjear por recompensas. De este modo, se incentiva el uso de la aplicación y, en consecuencia, se fomenta la movilidad sostenible en las ciudades.

Abstract One of the biggest issues that faces the society today is climate change and how it affects us, individually and as a whole. One of the crucial points in the fight against climate change is the way we move people and goods. Thus, the use of sustainable transport in cities is being promoted. This project aims to develop a bicycle rental or management system that facilitates the use of bicycles for people who do not have their own. It can be integrated by both public and private companies.

The system makes it possible to manage the reservation, rental, and usage of the bicycles, which are distributed throughout a city at different stops. In addition, a gamification system is integrated, based on obtaining points when using the application, which can then be exchanged for rewards. In this way, the use of the application is encouraged and, consequently, sustainable mobility in cities is promoted.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Contexto

El calentamiento global y como consecuencia el cambio climático, es el mayor desafío medioambiental al que se enfrenta el planeta, y unos de los mayores desafíos en general a los que se enfrenta la sociedad. La acción humana desde la revolución industrial, el crecimiento de la población, el aumento de la demanda y producción de energía junto a un nuevo modelo de consumo ha dado como resultado un aumento de la temperatura del planeta de 1.1 °C entre 1850 y 2017 [1]. El aumento del nivel medio de temperatura del planeta provoca variaciones en el clima conocidas como cambio climático, que de forma natural no se producirían.

La principal causa del calentamiento global es el efecto invernadero [2]. Este fenómeno, que ayuda a mantener el nivel medio de temperatura en la superficie terrestre, afecta negativamente cuando los gases de efecto invernadero retienen en calor, provocando la intensificación de este fenómeno, produciendo, por tanto, el calentamiento global. Otras causas son el aumento de población, la destrucción de los ecosistemas, tanto marinos como terrestres y principalmente el uso de combustibles fósiles como el carbón y el petróleo también influyen en el calentamiento global.

Las consecuencias del calentamiento global son muchas y en diversos ámbitos, desde cambios en el medio ambiente y los ecosistemas, hasta problemas humanitarios y económicos. Algunas de las consecuencias más destacadas son, el derretimiento de los cascos polares que, junto al aumento del nivel del mar, producirán inundaciones de ciudades costeras y consiguientemente la migración forzada de dichas poblaciones [2], aumento de desastres naturales, incremento de incendios debido a olas de calor y otros fenómenos meteorológicos como huracanes y lluvias torrenciales.

Todas las soluciones para el problema del calentamiento global se enfocan en lo acordado por todas las naciones en el acuerdo climático de París del 2015 [3], mantener el aumento de las temperaturas globales muy por debajo de los 2° C en este siglo y continuar los esfuerzos para mantenerlos por debajo de los 1.5° C. Para lograrlo se necesita una adaptación global por parte de todas las naciones, empresas y las personas de la sociedad. Algunas de las estrategias para reducir las emisiones de gases de efecto invernadero son cambiar los hábitos de consumo y de producción de las empresas,

fomentar el uso de energías renovables y reducir el uso de energías fósiles.

Relacionado con la reducción de consumo de energías fósiles es donde entra este proyecto. La movilidad y los desplazamientos, tanto de personas, como de mercancías, es uno de los grandes involucrados en el uso de energías fósiles y se debe hacer una transición a un modelo de movilidad sostenible. En la ciudad de Valladolid, en los últimos años se ha presentado un plan de movilidad urbana sostenible y segura [4]. En este plan se pretende promover la movilidad del ciudadano de modo no motorizado, es decir, de forma peatonal o en bicicleta. También reducir el uso del vehículo privado en favor del trasporte público como el autobús. Con esto se pretende conseguir una mejor en la calidad de vida de la ciudadanía a la vez que se contribuye a la lucha contra el cambio climático.

El servicio público de alquiler de bicicletas de Valladolid se encuentra en una situación de poca utilización, ya que los usuarios encuentran problemas de disponibilidad de bicicletas en las paradas y el estado de las mismas es bastante mejorable. Pero a final del año 2021 el Consejo de Administración de Auvasa aprobó la licitación del nuevo Sistema Público de Bicicleta de la ciudad [5]. Con una inversión de casi 6 millones de euros se pretende adquirir nuevas bicicletas, de las cuales un 25 % serán eléctricas. También se van a mejorar las infraestructuras, como puntos de anclaje y estaciones. Con esto se pretende abordar una «total renovación y ampliación completa del servicio», además de dar un impulso «definitivo» de dicho sistema.

Además de lo anterior, en el marco del plan de movilidad, se han recibido inversiones por parte de fondos europeos [6], que se destinarán en lo que concierne a este proyecto al desarrollo de infraestructuras para peatones y ciclistas, como la dotación de 60.000 euros para la instalación de estacionamientos de bicicletas cerrados y la ampliación de la red de carril bici [7].

1.2. Motivación

Teniendo en cuenta el contexto de la ciudad y la inversión que se está realizando en materia de movilidad sostenible y en concreto la movilidad en bicicleta, se necesita una aplicación capaz de ofrecer los servicios necesarios para el correcto funcionamiento de la red pública de bicicletas, tanto la consulta del estado y la ubicación de bicicletas disponibles, como el alquiler de las mismas y su uso como mapa/guía durante el trayecto.

A mayores de estas funcionalidades, que se podrían considerar básicas para una aplicación de este ámbito, también se cree necesario que la aplicación incentive su uso. Uno de los problemas que tiene el sistema actual son los pocos incentivos que tiene la ciudadanía para usarlo, dando lugar al uso de otros medios de transporte en lugar de la bicicleta, como el vehículo privado. Para incentivar que tanto nuevos usuarios empiecen a utilizar la aplicación, como a los usuarios ya existentes que continúen usándola, es necesario un sistema de gamificación que recompense al usuario de manera relevante. Este sistema de gamificación por puntos es necesario que se traduzca en recompensas tangibles para el usuario, como por ejemplo, descuentos en restauración o bonos culturales. De esta manera, el usuario sentirá que el progreso en la aplicación y su uso tiene una recompensa, lo que hará que continúe usándola y, por tanto, promoviendo el uso del trasporte sostenible.

1.3. Estado de la cuestión

Actualmente, la aplicación para el alquiler y uso de las bicicletas públicas de Valladolid se encuentra en estado des continuado, además de todos los inconvenientes y problemas que tiene. Según las reseñas y los comentarios de Google Play [8], la aplicación tiene muchos problemas para adquirir bonos y desbloquear las bicicletas.

Existen aplicaciones para el alquiler y uso de bicicletas, ya sean de empresas privadas o de un servicio público de una ciudad en concreto. La gran mayoría de estas aplicaciones no implementan un sistema de gamificación para fomentar su uso, por lo que en ese sentido no existe una alternativa clara o que busque el mismo propósito.

De la búsqueda realizada se van a destacar las ventajas y desventajas de las aplicaciones similares o relacionadas, dado que se busca combinar las mejores características de cada una de las aplicaciones relacionadas, al mismo tiempo que innovar.

1.3.1. Vallabici

Vallabici [8] es la aplicación que se pretende sustituir con este proyecto. Es la aplicación de alquiler de bicicletas en Valladolid disponible en este momento. Su estado es des continuado y poco funcional dado que gran parte de las funcionalidades no están activas. En su estado actual sirve como punto de información para conocer la disponibilidad de bicicletas en las estaciones distribuidas por la ciudad. El mapa, como se puede ver en la figura 1.1, no funciona y muestra las diferentes paradas de una manera que la información no es útil. Tampoco permite el alquiler de bicicletas desde la aplicación, delegando esta funcionalidad al sistema de paradas distribuidas por la ciudad.

1.3.1.1. Desventajas

- La interfaz está anticuada y poco trabajada.
- Las funcionalidades que te proporciona la aplicación son meramente informativas, ya que solo permite consultar el estado de las paradas y de las bicicletas en las mismas.
- El sistema de mapas no funciona y por lo observado su funcionalidad sería informativa en vez de interactiva.

Figura 1.1: Pantalla principal que muestra el mapa de paradas o estaciones

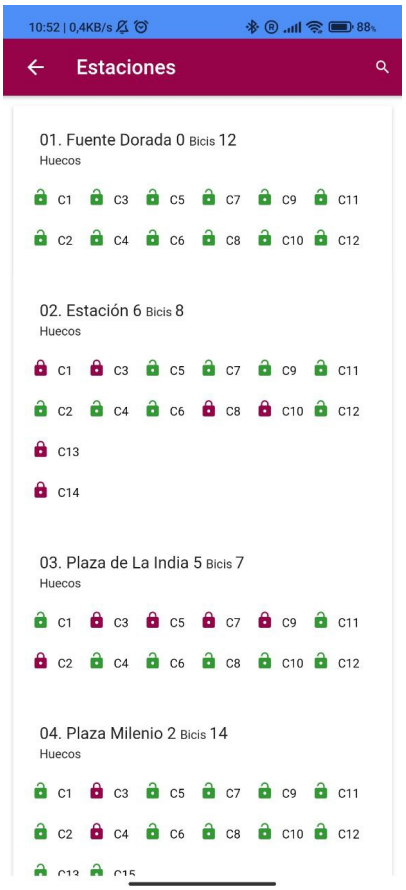


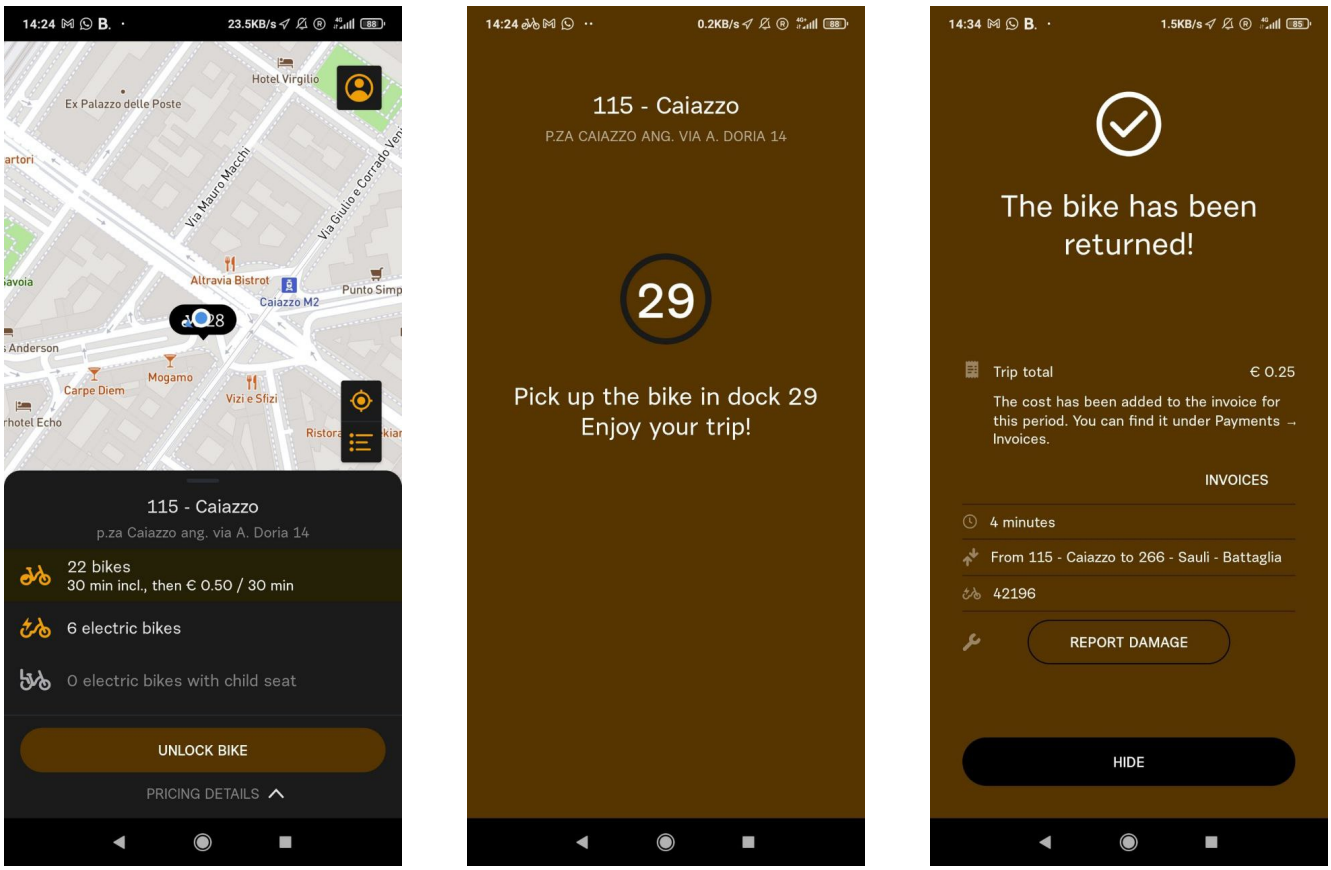
Figura 1.2: Pantalla de estaciones con la información de bicicletas

1.3.2. BikeMi

BikeMi [9] es la aplicación para alquiler y uso de bicicletas públicas de la ciudad de Milán, similar a la idea que se plantea en este proyecto, pero sin el componente de gamificación para fomentar su uso. Las bicicletas se ubican en paradas distribuidas por la ciudad, se puede consultar en el mapa las distintas paradas y el número de bicis y el tipo de estas que hay en cada parada. El sistema te permite desbloquear una bicicleta utilizando un tótem ubicado en cada parada o la aplicación móvil, seleccionando la parada de la que se quiere extraer la bicicleta y el tipo de la misma, normal, eléctrica o con soporte para niños. Es necesario colocar la bicicleta en otra parada para terminar su uso.

1.3.2.1. Ventajas

- Sencilla de utilizar e interfaz intuitiva, no tiene más funciones de las necesarias



(a) Pantalla principal con mapa y estado de una estación

(b) Pantalla confirmación desbloqueo de bicicleta

(c) Pantalla resumen ruta realizada tras devolver la bici

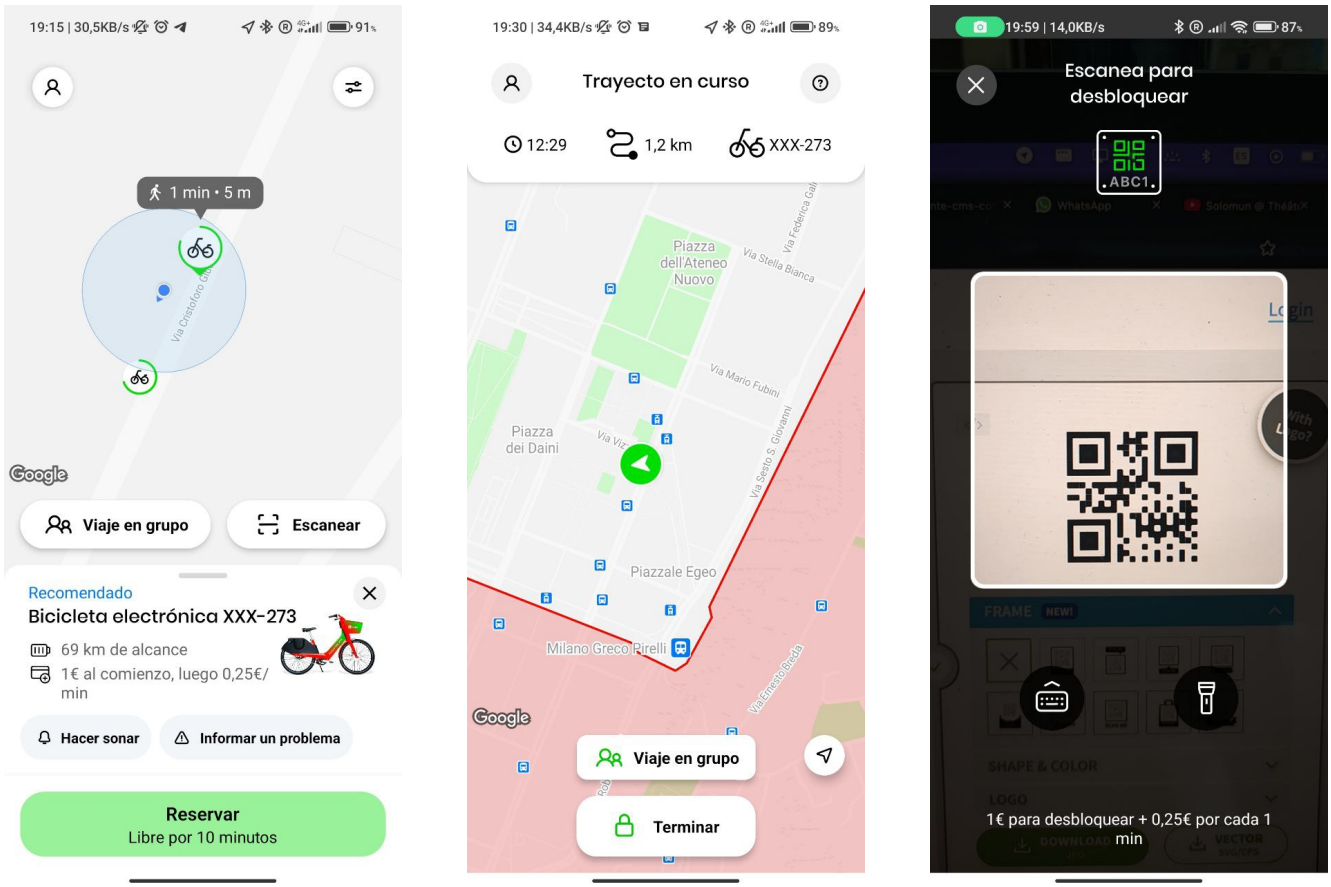
Figura 1.3: Aplicación de BikeMi

1.3.2.2. Desventajas

- Solo se pueden encontrar bicicletas en las distintas paradas ya predefinidas, por lo que en ciertas situaciones puede ocurrir que una parada no disponga de bicicletas en cierto momento.
- Tener que colocar la bicicleta en una parada al terminar el trayecto puede limitar hasta que zonas de la ciudad puedes utilizar las bicicletas.
- Necesitas una suscripción obligatoriamente para utilizar el servicio, no se puede pagar solo por un único uso.

1.3.3. Lime

Lime [10] es una de las aplicaciones privadas para el alquiler de bicicletas, patinetes y motocicletas más utilizada y extendida, tanto en ciudades Europeas como en el resto del mundo. En este caso, los diferentes vehículos se encuentran distribuidos alrededor de la ciudad sin estar ubicados en paradas. Para alquilar un vehículo lo que tienes que hacer es escanear o introducir un código QR del vehículo que se desee, también se pueden reservar desde el mapa de la aplicación, lo que permite que otros usuarios no te quiten el vehículo que quieres utilizar. Tras finalizar el trayecto se bloquea el vehículo volviendo a escanear el código o seleccionando la opción de finalizar trayecto desde la app. No es necesario estacionar el vehículo en una parada determinada, se puede estacionar en cualquier ubicación mientras no incumpla ninguna normativa de estacionamiento.



(a) Pantalla de reserva de bici (b) Pantalla de ruta en curso (c) Pantalla desbloqueo con QR

Figura 1.4: Aplicación de alquiler de bicicletas Lime

1.3.3.1. Ventajas

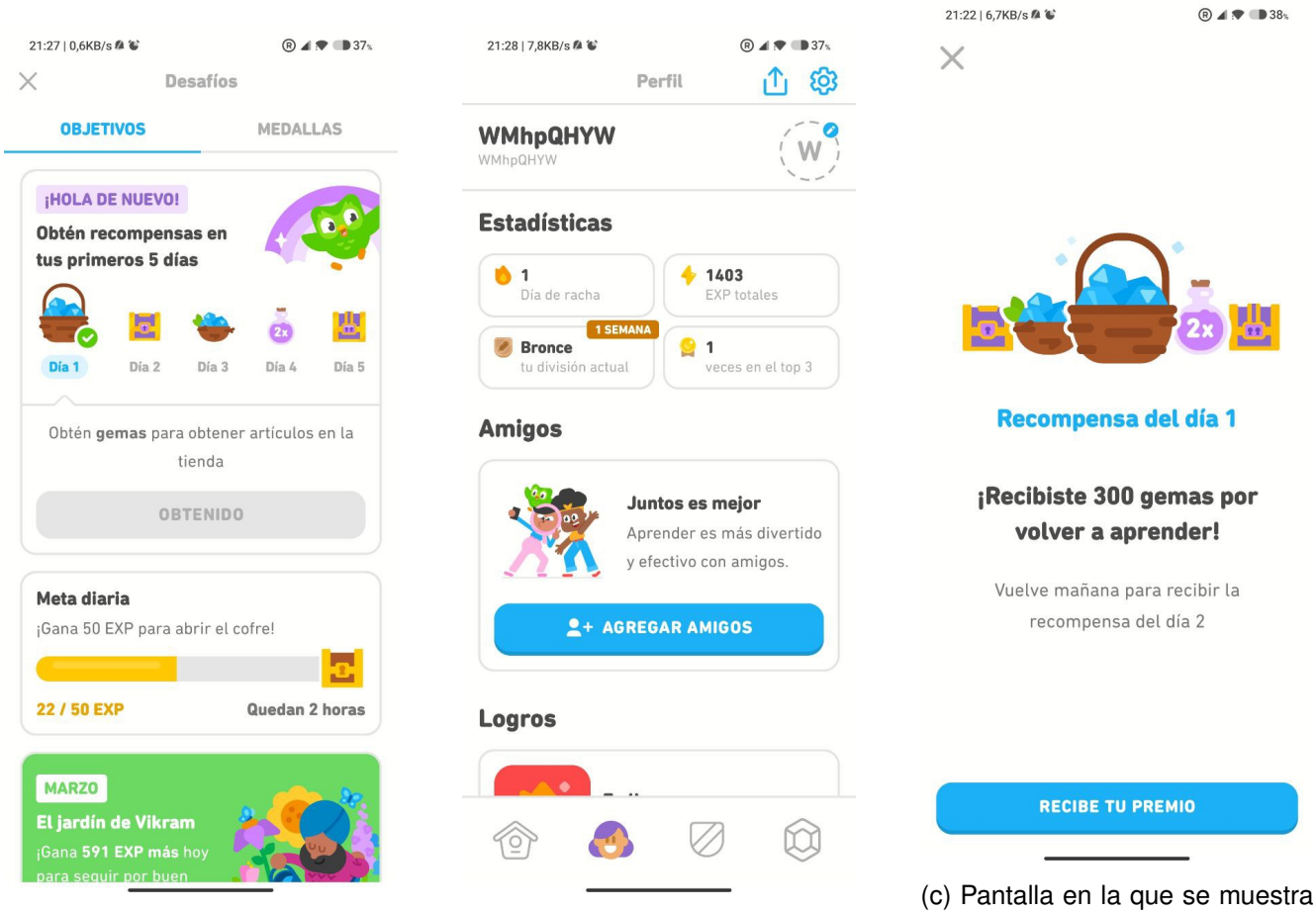
- Facilidad de localización de los vehículos y de desbloqueo de los mismos
- Para trayectos ocasionales es mejor, ya que no necesitas una suscripción
- No estar limitado a estacionar los vehículos en paradas fijas, permite mayor libertad en las rutas

1.3.3.2. Desventajas:

- Como los vehículos no tienen unas paradas fijas puede ocurrir que no tengas vehículos cerca
- En ciertas zonas no se permite bloquear los vehículos, por lo que si vives en dichas zonas lo tendrás que estacionar fuera de ellas

1.3.4. Duolingo

Duolingo [11] es la aplicación perfecta para ejemplificar la gamificación de una tarea que puede resultar repetitiva o aburrida, en este caso aprender idiomas. Las lecciones se presentan como juegos interactivos de diferentes tipos, que cuando se realizan correctamente se te otorgan recompensas, en forma de experiencia, gemas o medallas. De esta manera se pretende incentivar el uso de la aplicación. Su sistema de monetización se basa en la compra de gemas que se puede emplear para recargar las vidas u oportunidades de juego, revisar errores en las pruebas y eliminar anuncios.



(a) Pantalla principal de progreso de desafíos y recompensas

(b) Pantalla del perfil con estadísticas y logros

(c) Pantalla en la que se muestra recompensas obtenidas por iniciar la aplicación

Figura 1.5: Aplicación de aprendizaje Duolingo

1.3.4.1. Ventajas

- Interfaz sencilla y adaptada perfectamente a la tarea que se pretende realizar.
- Sistema de notificaciones y recordatorios cumple su función de incentivar que se practique diariamente.
- La conexión y competición con amigos fomenta que se siga usando la aplicación.

1.3.4.2. Desventajas

- El sistema de niveles termina siendo irrelevante debido a las pocas recompensas tangibles que se obtienen de él. Esto quiere decir que, la recompensa es aprender el idioma, pero el sistema de niveles no tiene por qué reflejar que se haya aprendido el idioma, sino que se ha realizado la tarea diaria durante un tiempo.
- El sistema de notificaciones puede ser pesado debido a que diariamente se reciben muchas notificaciones o recordatorios para continuar jugando, lo que puede dar lugar a que se desinstale la aplicación.
- La metodología de aprendizaje no es la más adecuada para el aprendizaje de un idioma, ya que se fomenta poco la práctica hablada.

1.3.5. Venchi

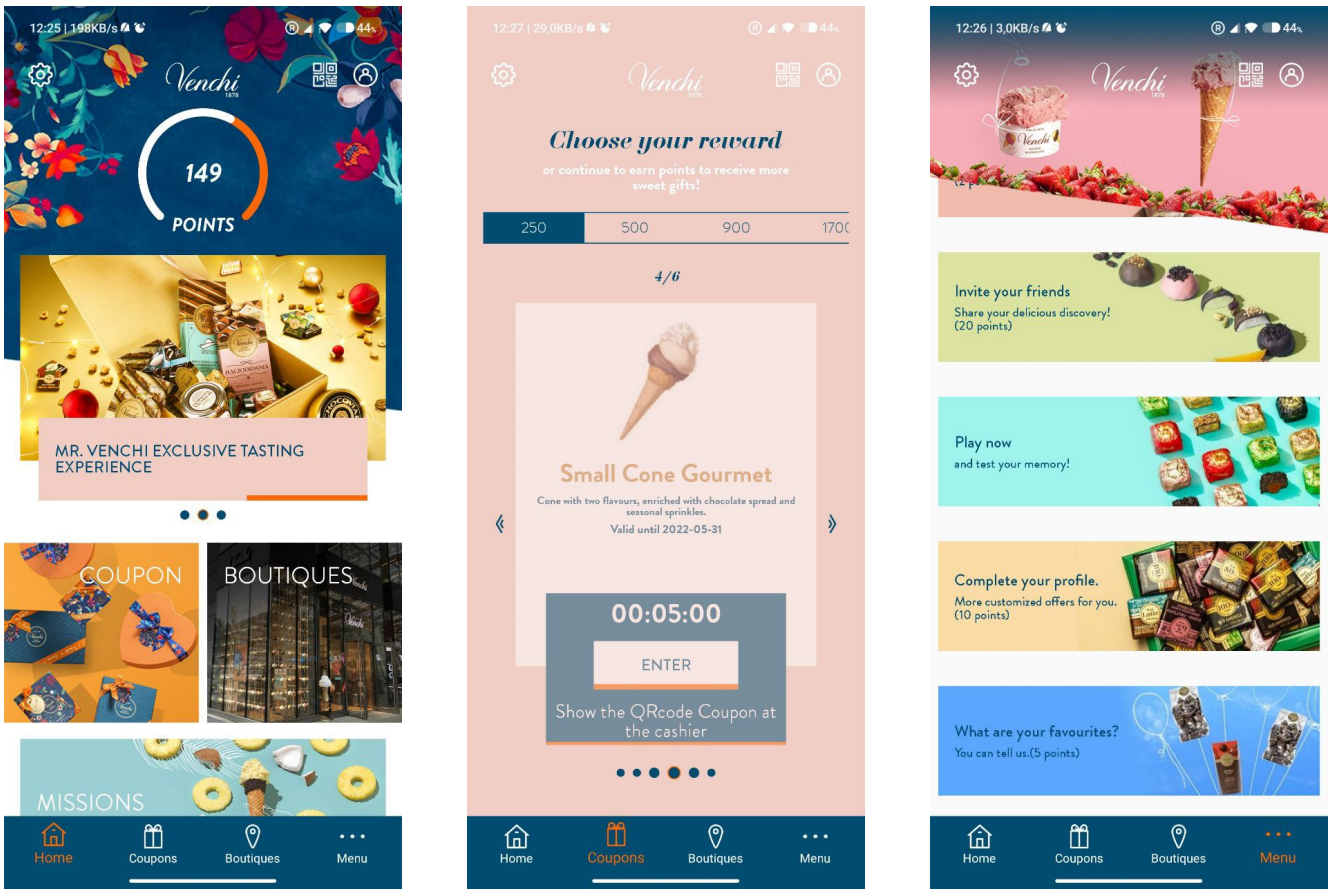
Venchi [12] es una cadena de heladerías italianas que ofrecen una app que funciona como tarjeta de fidelización, en la que vas acumulando puntos a medida que realizas compras. También se pueden obtener puntos realizando otras tareas dentro de la app, como realizar tests, completar el perfil o invitar a tus amigos a que usen la aplicación. Con los puntos se pueden obtener cupones que se pueden cambiar por productos de la tienda. De esta forma se fideliza a los clientes para que continúen comprando y utilizando la app para obtener premios tangibles.

1.3.5.1. Ventajas

- Los puntos se traducen en recompensas tangibles, como helados, chocolate u otros dulces, por lo que el usuario siente que el uso de la aplicación sirve para algo.
- Las tareas que se realizan para obtener puntos son sencillas y llevan poco tiempo, por lo que se pueden realizar en cualquier lugar o momento.

1.3.5.2. Desventajas

- La obtención de puntos es muy lenta, se necesitan 250 puntos para obtener la primera recompensa, mientras que las tareas para obtener puntos como mucho te dan 20 puntos.



(a) Pantalla principal con número de puntos

(b) Pantalla para canjear puntos por cupones

(c) Pantalla de tareas para obtener puntos

Figura 1.6: Aplicación de Venchi

- La forma más rápida de obtener puntos es comprando productos, por lo que si intentas ahorrar dinero no es la mejor opción para obtener puntos.

1.4. Entorno y herramientas utilizadas

En esta sección se van a desarrollar las principales tecnologías que se han empleado para desarrollar este proyecto.

Como este proyecto se ha partido desde cero, sin ningún código base, se ha tenido completa libertad en la elección de tecnologías. En la sección 3.2.1 se detalla en profundidad el proceso de elección de cada una de las tecnologías, pero de manera resumida, el criterio que se ha empleado al elegir es primero intentar utilizar las tecnologías más modernas para el Front End y tecnologías ya conocidas para agilizar el desarrollo del Back End.

Figura 1.7: Logo de Android

1.4.1. Android

Android [13] [14] (ver figura 1.7) es un sistema operativo para dispositivos móviles que parte de una versión modificada y adaptada del Kernel de Linux para una gran variedad de dispositivos con pantalla táctil, como teléfonos inteligentes, tabletas y multitud de otros dispositivos. Actualmente, es propiedad de Google, que publicita y desarrolla en conjunto con el consorcio *Open Handset Alliance*.

En la figura 1.8 se puede ver como está organizada la arquitectura de Android. La base del sistema es el kernel de Linux, como se ha comentado con anterioridad, que es el encargado de gestionar a bajo nivel los procesos de ejecución y la gestión de memoria, al mismo tiempo que facilita a los productores de hardware desarrollar controladores hardware para un kernel bien conocido y estandarizado.

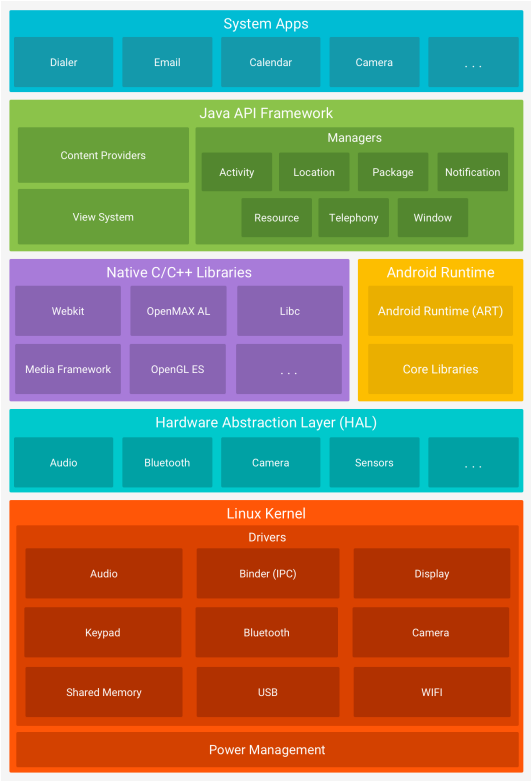


Figura 1.8: Arquitectura dividida por capas de Android [14]

La siguiente capa es HAL o *capa de abstracción de hardware*, en esta se exponen interfaces para que las capas superiores utilicen diferentes componentes de hardware como la cámara o el bluetooth, cargando la librería o módulo correspondiente en el momento correcto. En la capa superior a esta se

encuentran las librerías nativas de C y C++ junto al entorno de ejecución de Android, las librerías de C y C++ se encargan de facilitar el uso de estos lenguajes si fuera necesario desde la aplicación nativa de Android, mientras que el entorno de ejecución es donde se encuentran las diferentes máquinas virtuales donde se ejecutan las aplicaciones. Finalmente, en la capa superior se encuentra el Java API Framework, compuesta por las diferentes funcionalidades que los desarrolladores pueden emplear para desarrollar aplicaciones de manera más sencilla. Por último están las aplicaciones instaladas por defecto en el dispositivo Android, como aplicaciones para mail, SMS o calendario, estas pueden ser sustituidas por aplicaciones de terceros sin problema.

Hay que destacar también el sistema de versiones o niveles de la API de Android. Android es un sistema que evoluciona con rapidez y nuevas funcionalidades aparecen en poco tiempo, así como otras quedan en desuso [15]. Como se puede ver en la figura 1.9 la versión de API 21 engloba al 98,6 % de los dispositivos, mientras que la 32 no aparece en la tabla, por lo que el número de dispositivos en muy reducido. Por otro lado, Android te permite utilizar ambas, empleando, por ejemplo, la versión 32 como *target SDK,* siendo la versión ideal para el uso de la aplicación [16], por otro lado, *minSDK* es la versión más antigua de la SDK de Android en la que se puede ejecutar la aplicación, esto se consigue con las librerías de compatibilidad integradas en Android.

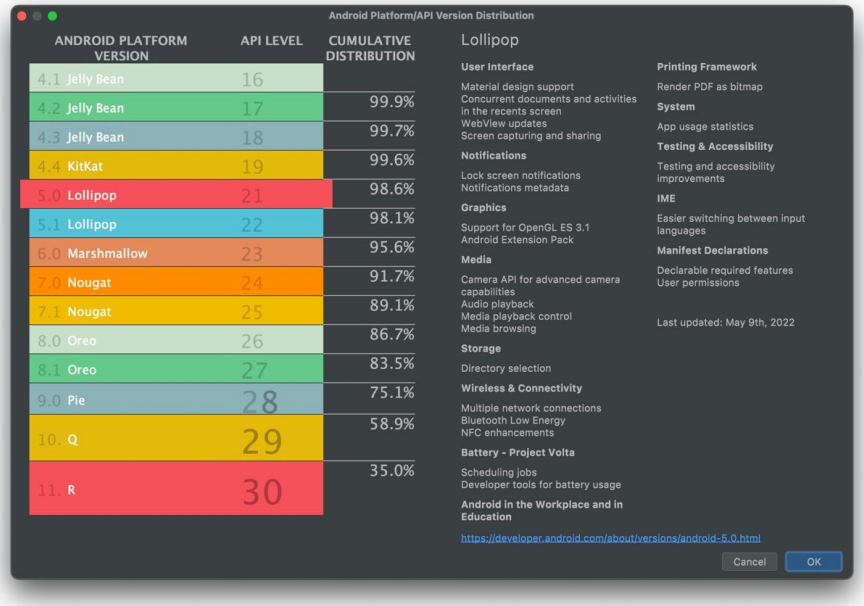


Figura 1.9: Sistema de versiones y compatibilidad en Android

1.4.2. Kotlin

Kotlin [17] [18] (ver figura 1.10) es un lenguaje de programación multi plataforma de propósito general y de *tipado* estático. Está diseñado para operar con el lenguaje de Java y su máquina virtual JVM, además de poder ser compilado a JavaScript para el desarrollo de aplicaciones web. Está desarrollado por la empresa *JetBrains,* que también es la empresa que ha desarrollado los editores de código Android Studio y PHPStorm.

Figura 1.10: Logo de Kotlin

Kotlin, al estar diseñado para operar con Java, es un lenguaje orientado a objetos que además de las funcionalidades o herramientas de Java, también proporciona sus propias funcionalidades que facilitan el desarrollo orientado a objetos, como *Sealed classes* [19], que restringen la jerarquía de las sub-clases definiendo todas ellas en tiempo de compilación, siendo imposible agregar nuevas sub-clases tras la compilación y las *Data classes* [20] empleadas esencialmente para almacenar datos, facilitando la redefinición de métodos como *equals toString*. También proporciona *Null Safety* que facilita la gestión de valores *null*, a diferencia de la gestión mediante *Excepciones* que proporciona Java. Y por último el uso de *Corrutinas* [21] para la ejecución de código asíncrono.

1.4.3. Laravel

Laravel [22] (ver figura 1.11) es uno de los frameworks más populares para el desarrollo de aplicaciones web con el lenguaje PHP. Con él se pueden desarrollar aplicaciones completas o Full-stack empleando diferentes Front Ends, como *Livewire* en PHP o *React* y *Vue* en Javascript. También es posible desarrollar el Back End o Rest API [23], sin implementar un Front End.

Figura 1.11: Logo de Laravel

Laravel está basado en el ya bien conocido patrón arquitectónico MVC [24] o Modelo Vista Controlador. Es un Framework que sigue activo, siendo la última versión, la 9.0, a la que se puede acceder desde Github [25], ya que es de código abierto y está bajo la licencia MIT.

Laravel te proporciona herramientas y funcionalidades que te permiten agilizar más el desarrollo, como por ejemplo un sistema de autenticación [26] sencillo y que necesita de mucha configuración y una sencilla conexión con la base de datos. También es importante que el despliegue de la aplicación es sencillo, gracias a la herramienta **artisan** [27], que facilita diversas tareas a lo largo del proceso de desarrollo y despliegue.

Con respecto al lenguaje de programación PHP, cabe considerar que es un lenguaje de *scripting* o interpretado, enfocado al desarrollo web. A pesar de las críticas que recibe el lenguaje, sigue siendo uno de los más utilizados [28], siendo en enero de 2022, usado en el 78,1 % de páginas web de las que

se conoce el lenguaje de programación del lado del servidor.

1.4.4. Mapbox

Mapbox [29] [30] (ver figura 1.12) es un proveedor de mapas online, en los que se incluyen una serie de librerías que te permiten interactuar con las diferentes tareas relacionadas con mapas. El proyecto nació en el 2010 y se expandió con rapidez debido a ser una alternativa como proveedor de mapas, ya que en esa época solo se disponía de los mapas de Google Maps. Actualmente, es utilizado tanto por grandes empresas como *Snapchat*, como por la comunidad *Open Soruce* con aplicaciones como *LeafLet* [30].

Mapbox proporciona gran variedad de librerías, pero las principales son el *Maps SDK* [31] para poder visualizar mapas y agregar elementos que contengan información al mismo y la *Navigation API* [32] que permite obtener una ruta detallada a partir de diferentes coordenadas, de esta manera es posible obtener información de duración, distancia y otros detalles a partir de diferentes puntos del mapa.

Figura 1.12: Logo de Mapbox

Además de los datos que se pueden obtener del uso de los mapas, también hay que considerar que es altamente configurable en términos de estilos [33], permitiendo adaptarse a diferentes necesidades como mostrar carriles bici o las diferentes paradas.

1.5. Objetivos

El objetivo principal del proyecto es diseñar e implementar una aplicación que simplifique el alquiler de bicicletas, al mismo tiempo que su uso se ve incentivado por un sistema de gamificación y recompensas.

Este objetivo se podría resumir en los siguientes objetivos:

- Desarrollar un mapa que permita el visualizar, alquilar y facilitar el uso de las bicicletas públicas.
- Desarrollar un sistema de puntuación o incentivos que complemente a las funcionalidades del mapa.

1.6. Metodología

Teniendo en cuenta la naturaleza del proyecto, las limitaciones de ser solo un desarrollador y los requisitos base fijados, que posteriormente pueden ser ampliados, se va a adoptar una metodología iterativa e incremental. Gracias a esta metodología, en cada iteración se podrá tener un grado de adaptación en cuanto a los requisitos y el diseño, pudiendo ampliar o reducir las funcionalidades del proyecto si fuera necesario.

En la figura 1.13 se puede observar el proceso en cada iteración. Proporcionando al cliente, el tutor, en este caso, cierta funcionalidad implementada que permita reevaluar los requisitos y el diseño, a la vez que es posible probar la aplicación.

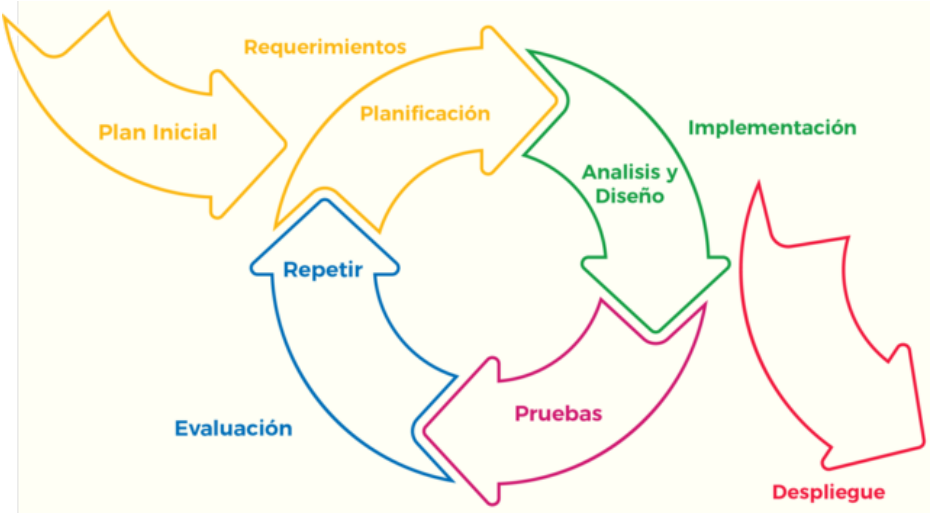


Figura 1.13: Esquema metodología iterativa

1.7. Estructura de la memoria

Esta memoria se encuentra dividida en los siguientes capítulos:

- **Introducción**: La presentación de manera general del proyecto, el contexto y las motivaciones que han llevado a la realización del mismo, el estado de la cuestión o aplicaciones similares, objetivos generales que se pretenden alcanzar en el proyecto y la metodología que se va a seguir durante el desarrollo del mismo.
- **Planificación**: Se detallará y analizará la estimación del trabajo a realizar y la organización que se debe seguir para completarlo con éxito, además se especifican los riesgos que podrían aparecer durante el desarrollo y la forma de evitarlos y mitigarlos. Se describen también inicialmente en que van a consistir las iteraciones del proyecto. Se incluye también el entorno tecnológico en el que se ha desarrollado el proyecto y una estimación de los costes del mismo. Por último, una revisión de como se ha planificado finalmente el proyecto, que permite hacer una comparación con la planificación inicial.

- **Iteraciones**: En este capítulo se detalla el trabajo realizado en cada iteración, de esta manera se podrá contrastar con la planificación inicial, además de facilitar el entendimiento de como se ha desarrollado el proyecto y que tareas se han ido realizando a lo largo del mismo.
- **Estado de la Aplicación**: Es el capítulo dedicado a la descripción del análisis, diseño y la estructura del proyecto. Además de los diagramas que lo describen y documentan.
- **Pruebas**: Se describen las pruebas que se han realizado para el proyecto, tanto de usabilidad como unitarias.
- **Conclusiones**: Recapitulación de los conocimientos y otros detalles obtenidos durante la realización del proyecto.
- **Apéndices**: Documentación adicional como manuales de uso de la aplicación, acrónimos, manual de despliegue y otros datos que se hayan obtenido durante la realización del proyecto.
- **Bibliografía**: Conjunto de referencias bibliográficas que se han empleado en el proyecto.

Capítulo 2

Planificación

2.1. Planificación inicial

A lo largo de esta sección se tratará de realizar la planificación y organización del trabajo necesario para completar las iteraciones del proyecto y las consiguientes tareas del mismo para cumplir los requisitos al finalizar el proyecto.

2.1.1. Distribución temporal

El trabajo se va a realizar a lo largo del segundo cuatrimestre del curso académico 2021-2022, empezando la primera iteración el 1 de marzo de 2022 y una fecha de finalización estimada del 10 de julio del 2022. Iniciando el trabajo en dicha fecha da como resultado una duración de 4 meses y 10 días, es decir, 18 semanas con un total de 130 días hasta la fecha marcada. La normativa de la Universidad de Valladolid establece que el Trabajo de Fin de Grado tiene que tener una duración mínima de 300 horas. Se estima que serían necesarias 2.3 horas diarias para llegar al mínimo, sin embargo, se van a emplear fines de semana en los cuales se podrán dedicar más horas, en concreto 18 fines de semana, es decir, 36 días. Con lo descrito se estima que duplicando las horas los fines de semana, 4 horas cada día, y dedicando 2 horas al día los días entre semana, se alcanzan las 324 horas en total, lo que proporciona un margen que se puede emplear si hubiera cualquier inconveniente.

Cabe destacar que cada 2 semanas se realizarán reuniones de seguimiento con el tutor. Esto puede variar dependiendo de la iteración y de las necesidades u objetivos en dicho momento del proyecto. En estas reuniones se expondrá el estado del proyecto en la iteración correspondiente, se plantearan dudas o cuestiones si existieran y se planificarán los próximos objetivos a cumplir.

El tiempo de trabajo se repartirá en a lo largo de las previamente mencionadas iteraciones, la propuesta inicial es la siguiente:

- **1ª Iteración** Duración estimada 3 semanas. En esta primera iteración se plantearán las bases

del proyecto, como objetivos básicos, el contexto y las motivaciones del mismo, y se realizará un análisis de proyectos similares ya existentes, con el objetivo de descubrir sus puntos débiles para evitarlos en este proyecto, como sus puntos fuertes para aplicarlos. Esta iteración servirá principalmente como punto de partida del proyecto y planificación del mismo, estableciendo todas las iteraciones y plazos para completar el mismo de la mejor manera posible.

- **2ª Iteración** Duración estimada de 2 semanas, en esta iteración, junto a la información obtenida de aplicaciones relacionadas, se investigará las diferentes opciones tecnológicas que hay disponibles para desarrollar el proyecto, lenguajes de programación, frameworks y librerías, tanto para el desarrollo del Front-End como del Back-End. Una parte importante de la aplicación es la librería o SDK de mapas que se va a emplear, ya que puede condicionar el uso y la implementación de la misma.

  Con las tecnologías escogidas se tomará la decisión de realizar una aplicación nativa, ya sea para Android o iOS o una aplicación multi-plataforma.

  Además de las tecnologías directamente relacionadas con la aplicación, también se establecerán las herramientas auxiliares para el proyecto, como editores, sistemas de control de versiones y repositorios.

- **3ª Iteración** Duración estimada 2 semanas, a continuación del trabajo y de las decisiones tomadas en la iteración previa, se procederá a aprender o familiarizarse con las tecnologías escogidas. Para esta tarea se realizarán aplicaciones de prueba y una integración básica de todos los componentes de la misma, con la intención de detectar en una primera fase si las tecnologías escogidas satisfacen los requisitos no funcionales del proyecto. De esta manera podrán detectar problemas o incompatibilidades en una primera fase antes del haber comenzado con el desarrollo principal de la aplicación.

- **4ª Iteración** Duración estimada 8 semanas, en esta iteración se realizará el diseño, implementación y testeo principal del proyecto. A lo largo de esta se realizarán reuniones con el tutor que servirán para hacer un desarrollo cíclico del proyecto, esto quiere decir que en cada reunión se revisará el trabajo realizado previamente y se establecerán nuevos objetivos, que pueden ir desde cambios en el diseño a la implementación. De esta manera se conseguirá mitigar los problemas que puedan surgir a lo largo del desarrollo del proyecto.

- **5ª Iteración** Duración estimada 3 semanas, en la iteración final servirá para mitigar los retrasos del proyecto que se hayan podido ocasionar, al mismo tiempo se deberá completar la documentación del proyecto, tanto la documentación académica como la técnica.

Como es lógico, la iteración que se le van a dedicar más horas es a la 4ª, ya que es la que más carga de trabajo tiene.

2.1.2. Análisis de riesgos

En la planificación de todos los proyectos se tiene que elaborar un plan de riesgo, ya que en la planificación inicial hay suposiciones, lo que genera un grado de incertidumbre que es necesario gestionar.

En esta sección se van a definir los posibles riesgos que pueden ocasionar problemas o retrasos en el proyecto. En la tabla 2.1 se definen cada riesgo, junto a los siguientes elementos relacionados con el riesgo:

- **Descripción** una breve descripción del mismo y el contexto en el que se puede llegar a dar.
- **Probabilidad** medida en tanto por 1 de que el riesgo ocurra, siendo 0,00 la probabilidad de que nunca ocurra y 1 la probabilidad de que ocurra seguro.
- **Impacto** o consecuencia de que ocurra el riesgo, en este caso lo vamos a medir en días de retraso que ocasionaría en el proyecto.

Por otro lado, como se han definido los riesgos, también se puede definir un plan de acción para los mismos. En la tabla 2.2 se define para cada riesgo los siguientes elementos:

- **Plan de prevención** son una serie de acciones que se pueden llevar a cabo para reducir la probabilidad de que el riesgo ocurra.
- **Plan de mitigación** son una serie de acciones que se pueden llevar a cabo una vez el riesgo ocurra, de tal forma que se minimice el impacto de este en el proyecto.
- **Exposición** al riesgo es el tiempo promedio que se perdería con cada uno de los riesgos, si estos llegan a ocurrir. Para calcularlo se multiplican la probabilidad de que ocurra con el tiempo estimado que se perdería.

	Riesgo	Descripción	Probabilidad	Retraso/días
1	Enfermedad	El alumnos cae enfermo	0.2	5
2	Carga de trabajo elevada las prácticas	Debido a las prácticas que se están haciendo en paralelo a este proyecto, puede ocurrir que la carga de trabajo aumente y no se le pueda dedicar la misma cantidad de horas a este proyecto	0.5	5
3	Falta de experiencia con desarrollo móvil	Se empleará mas tiempo en el proceso de aprendizaje de la aplicación móvil, ya sea Android, IOS o multi-plataforma	0.6	15
4	Error al elegir las tecnologías de desarrollo	Escoger erróneamente las tecnologías para el proyecto y tener que cambiarlas tiempo después de haber comenzado con la implementación	0.05	20
5	Extravío o ruptura del ordenador de trabajo	El ordenador que se utiliza como estación de trabajo queda inutilizado por cualquier razón	0.01	4
6	Planificación incorrecta	La planificación del proyecto no satisface los requisitos del proyecto, como recortes de plazos o falta de experiencia en planificación y no considerar tiempos realistas o no considerar riesgos	0.3	20
7	El ordenador de trabajo no cumple los requisitos técnicos para desarrollar una aplicación móvil	El desarrollo de aplicaciones móviles suele necesitar equipos con mas recursos, como memoria, procesador o almacenamiento	0.05	5
8	Cambio en los requisitos del proyecto	Los requisitos del proyecto cambian por parte del cliente, en este caso el cliente es el alumno o el tutor. Los cambios pueden surgir por falta de tiempo o por la implementación de nuevas funcionalidades	0.2	10

Tabla 2.1: Riesgos identificados que pueden afectar al proyecto durante su desarrollo

ID	Riesgo	Plan de prevención	Plan de Mitigación	Exposición
1	Enfermedad	Llevar una vida saludable y evitar actividades que puedan poner en riesgo mi salud	Redistribuir la carga de trabajo en las siguientes semanas si es una enfermedad leve. Si es una enfermedad mas prolongada, replantear los objetivos y reducir funcionalidades de la app	1
2	Carga de trabajo elevada las prácticas	No trasladar el trabajo de las prácticas fuera del horario de las mismas	Aumentar las horas de trabajo los fines de semana de las siguientes semanas al momento de mas carga para no sobrecargar de trabajo al desarrollador	2,5
3	Falta de experiencia con desarrollo movil	Dedicar el tiempo necesario para aprender los conocimientos necesarios para desarrollar el proyecto antes de comenzar con el y no durante el desarrollo del mismo	Dedicar mas horas durante los fines de semana para cubrir las carencias de experiencia	9
4	Error al elegir las tecnologías de desarrollo	Dedicar el tiempo necesario en la fase de investigación y elección de tecnologías en especial el sistema de mapas que se vaya a elegir	Cambiar a las tecnologías adecuadas lo antes posíble y adaptar el código que se tuviera en ese momento	1
5	Extravío o ruptura del ordenador de trabajo	Cuidar el material de trabajo de la forma adecuda	Se dispone de un segundo equipo personal menos potente pero que se prodría emplear para el desarrollo mientras se obtiene un equipo adecudo	0.25
6	Planificación incorrecta	Ser realista con las estimaciones de tiempo y consultar con el tutor para obtener las directrices adecuadas	Dedicar mas tiempo durante los fines de semana para cubrir el tiempo perdido	6
7	El ordenador de trabajo no cumple los requisitos técnicos para desarrollar una aplicación movil	Utilizar herramientas que permitan aligerar la carga de trabajo sobre el equipo, como utilizar dispositivos externos en vez de virtuales para el desarrollo de la aplicación	Adquirir un nuevo equipo o mejorar el actual	0.25
8	Cambio en los requisitos del proyecto	Definir correctamente los requisitos en las primeras fases del proyecto	Recortar los requisitos menos prioritarios o dedicar mas horas los fines de semana para completar los nuevos requisitos	2

Tabla 2.2: Plan de acción definido para los riesgos identificados para el proyecto

2.2. Entorno tecnológico

2.2.1. Herramientas utilizadas

A continuación se listan todas las herramientas empleadas para el desarrollo del proyecto.

- **Android SDK API 31** [34]: Suite proporcionada por Google para el desarrollo de aplicaciones para la plataforma móvil Android.
- **Mínimo Android SDK compatibilidad 21** [35]: Necesario para el uso del SDK de Mapbox y compatibilidad con versiones previas.
- **Android Studio Bumblebee** [36]: Versión para Mac del entorno de desarrollo para Android.
- **Kotlin** [17]: lenguaje de programación para el desarrollo de la aplicación Android.
- **Mapbox Maps SDK para Android**: Implementación nativa para el uso de mapas proporcionado por Mapbox [31].
- **PostgreSQL** [37]: Base de datos relacional.
- **Postman** [38]: Entorno de desarrollo, uso y pruebas para la API.
- **PHP 8.0.8** [39]: Lenguaje de programación empleado para el desarrollo del Back End.
- **PHPStorm** [40]: Editor de código para el desarrollo del Back End.
- **Balsamiq Wireframes** [41]: Herramienta para desarrollar bocetos de la interfaz de usuario.
- **Git 2.32.0** [42]: Sistema de control de versiones.
- **Gitlab** [43]: Repositorio remoto de la Escuela de Ingeniería Informática de la Universidad de Valladolid para almacenar el código e integrar el sistema de versiones.
- **Overleaf Web**: Editor web para realizar la documentación de este proyecto en el lenguaje LaTeX.
- **Firefox** [44]: Navegador web empleado para acceder a los diferentes recursos y plataformas web.
- **Telegram** [45]: Plataforma de comunicación con el tutor del proyecto.
- **Trello** [46]: Organizador de tareas del proyecto.

2.2.2. Entorno de desarrollo

Con respecto al entorno de desarrollo, como se ve reflejado en las siguientes tablas, se han empleado solo el ordenador portátil y el dispositivo móvil del desarrollador.

Ordenador	
MacBookPro14.3	
Hardware	
Procesador	Quad-Core Intel Core i7 2.8GHz
Memoria	16GB DDR3 2133 MHz
Disco Duro	250,69 GB SSD
Tarjeta Gráfica	Intel HD Graphics 630
Software	
Sistema Operativo	macOS 12.2.1
Arquitectura	64 bits

Tabla 2.3: Ordenador empleado durante el desarrollo

Movil	
XiaoMi Mi 10T Lite	
Hardware	
Procesador	Octa-core Max 2.2GHz
Memoria	8GB
Disco Duro	128 GB
Tarjeta Gráfica	Desconocida
Software	
Sistema Operativo	Android 12
Arquitectura	64 bits

Tabla 2.4: Dispositivo móvil empleado durante el desarrollo

2.3. Estimación de costes

En esta sección se detalla la estimación de costes del proyecto, tanto recursos humanos como materiales. Se va a contemplar desde un punto de vista académico y profesional, incluyendo y excluyendo en cada caso los costes pertinentes. Cabe destacar que los importes correspondientes al hardware empleado.

2.3.1. Salario del trabajador

Para estimar el salario que tendría un desarrollador con capacidades para realizar este proyecto se ha consultado varias fuentes que comparan los salarios que se publican de forma online, se ha encontrado que el salario bruto para un desarrollador de aplicaciones Android en España es de 41.857€ por año [47] que tras aplicar los impuestos correspondientes [48] se quedan en 2.295€ al mes. Se considera que en un mes hay 22 días laborales y una jornada completa de 8 horas, lo que suponen unas 176 horas al mes. La hora de trabajo se paga entonces a 13,03€/h. Como en la planificación inicial se estimaron unas 324 horas de trabajo total, se obtiene un resultado de, **4433,52€** como salario para desarrollar el proyecto.

No se ha tenido el salario de un desarrollador PHP, ya que se considera que un desarrollador Android también conoce tecnologías para realizar un sistema de Back End similar.

2.3.2. Espacio de trabajo y servicios

El proyecto se ha desarrollado en la vivienda del desarrollador en la ciudad de Milán, con un coste de 600€ al mes, si se considera que la duración del proyecto desde el inicio ha sido de 5 meses, se obtiene un total de **3000€**.

Los servicios básicos para el desarrollo del proyecto son la electricidad e internet. Se ha utilizado el proveedor de internet Tim [49] con un coste de 30€ al mes, como se ha indicado antes, la jornada laboral será unas 176 horas al mes y el total de horas son 324, aproximadamente 2 meses, por lo que el coste total de internet serían **60€**. La electricidad se ha utilizado el coste de esta en Italia [50], ya que se está desarrollando el proyecto desde este país, con un precio 0,256 €/kWh, dando como coste total por todas las horas empleadas en el proyecto **87,04€**.

2.3.3. Hardware y dispositivos

Los dos dispositivos que se han empleado principalmente para el desarrollo del proyecto son el ordenador portátil y el dispositivo móvil que se muestran en la sección 2.2.2. Para el cálculo del coste se utiliza el precio de compra del producto. El MacBook Pro 15"de 2017, tuvo un coste de 2400€ [51]. El dispositivo móvil costó 226€ [52].

2.3.4. Coste servicio de mapas

El servicio de mapas Mapbox tiene un sistema de precios [53], pero para el uso en este proyecto no supone ningún coste, ya que no se supera la cuota del plan gratuito. Por esto, el sistema de mapas no agrega ningún coste para el desarrollo del proyecto, pero que podría cambiar si se desplegara en un escenario de producción real con un gran cantidad de usuarios.

2.3.5. Coste total

El coste total del proyecto se ha dividido en dos tablas, diferenciando entre un coste simulado sin incluir el salario del desarrollador y otro coste realista en el que sí que se incluye el salario.

Material	Coste unitario	Unidades	Coste total
Espacio de trabajo	600€/mes	5	3000€
Ordenador	2400€	1	2400€
Dispositivo Móvil	226€	1	226€
Internet	30€/mes	2 meses	60€
Electricidad	0,256€/kWh	324h	87,04€
Salario	-	-	-
Total	-	-	5773,04€

Tabla 2.5: Coste real del proyecto

Material	Coste unitario	Unidades	Coste total
Espacio de trabajo	600€/mes	5	3000€
Ordenador	2400€	1	2400€
Dispositivo Móvil	226€	1	226€
Internet	30€/mes	2 meses	60€
Electricidad	0,256€/kWh	324h	87,04€
Salario	13,03 €/h	324h	4433,52€
Total	-	-	10.206,56€

Tabla 2.6: Coste simulado del proyecto

2.4. Planificación final

En esta sección se va a comentar como ha resultado la planificación tras la realización del proyecto y como ha diferido de la planificación inicial que se estableció al principio de este capítulo, comentando tanto la comparación de horas empleadas como los distintos cambios que han sufrido la estructura de las iteraciones.

Lo primero que se debe comentar es que la entrega prevista para el día 10 de julio fue pospuesta al día 15 de julio, dado que por falta de tiempo se tuvo que desplazar la entrega para poder completar las tareas restantes también dichos días.

Antes de comentar las diferencias entre horas empleadas en las iteraciones, se considera importante detallar el por qué de estas diferencias. Las iteraciones 1, 4 y 5 no han sufridos cambios en cuanto a las tareas que se habían planificado y que se han realizado, quitando en la iteración 4, que el proceso de diseño se trasladó a la iteración 3. Estas iteraciones sí que han sufrido modificaciones en cuanto a los tiempos.

Las iteraciones que más cambios han sufrido, en cuanto a tareas a realizar, han sido la iteración 2 y la 3. Inicialmente, se planificó que la iteración 2 correspondería con la investigación de las tecnologías que se iban a utilizar en el proyecto, mientras que la iteración 3 se dedicaría a familiarizarse con las tecnologías escogidas. El resultado final es que la iteración 3 se fusionó con la iteración 2, ya que a mayores de la investigación de tecnologías, también se consideró parte de la misma iteración la prueba de las mismas para determinar si las tecnologías escogidas eran las correctas. Como se comenta en el capítulo correspondiente a dichas iteraciones 3.2, en especial la 2, se consideró inicialmente desarrollar el proyecto con un lenguaje multi-plataforma, pero finalmente, durante las pruebas de estas tecnologías,

por problemas con la librería de mapas, se decidió implementar el sistema de forma nativa en Android, por lo que también se tuvo que probar dicha implementación de Android, dando lugar a la fusión de ambas iteraciones y por consiguiente la dedicación de más horas a la iteración 2.

Con respecto a la iteración 3, en vez de emplearla para las pruebas de tecnologías, se empleó para el análisis y diseño, incluyendo requisitos, diagramas y bocetos de la interfaz, como se describe en el capítulo correspondiente 3.3. Al mismo tiempo se utilizó tiempo de esta iteración para agregar la información correspondiente en la memoria al mismo tiempo que se cambiaban cosas propuestas por el tutor.

A lo largo de estas iteraciones se han ido teniendo reuniones regularmente cada dos semanas, con ciertas excepciones, como en el periodo de Semana Santa, que se dilató el tiempo entre reuniones y durante la iteración 4 que se realizaron con más frecuencia. En total han sido 8 reuniones. No es un número muy elevado, pero no se considera que hayan sido necesarias más, debido a que en cada reunión se definía bien cuáles eran las tareas a realizar y junto a la comunicación vía Telegram.

Las modificaciones a la estructura de las iteraciones han supuesto un cambio también en el número de horas que se han dedicado a las mismas. Las iteraciones que no han sufrido cambios han sido, la primera que se ha mantenido en 3 semanas y la iteración 3 que, a pesar de haber cambiado las tareas que se han realizado en ella, se han mantenido las 2 semanas estimadas inicialmente.

Con respecto a la iteración 2, al haber incluido también las tareas iniciales de la iteración 3, se ha alargado a las **5 semanas**. Esto se puede considerar un riesgo de mala planificación, pero se contempló la posibilidad, como se puede ver en la tabla 2.2 el riesgo número 4. Gracias a ello se aplicó el plan de prevención, dedicándole así el tiempo necesario a la fase de investigación.

Como consecuencia, las semanas dedicadas a la iteración 4 se han visto reducidas a **6 semanas**, pero las pruebas realizadas en la iteración 2 han permitido conocer mejor las tecnologías y agilizar el trabajo requerido en esta iteración. También hay que destacar que se definieron unos requisitos iniciales, que al principio de esta iteración se vieron recortados, para centrarse en los requisitos más fundamentales, dejando de lado los requisitos con baja prioridad. Esto también estuvo contemplado en el plan de acción, como se puede ver en la tabla 2.2, riesgo número 8. Aplicando el plan de mitigación, recortando los requisitos con menor prioridad para el proyecto.

También para ajustarse a la fecha de entrega, la iteración 5 se vio reducida de 3 semanas a **2 semanas**, en la que se completó la memoria y se prepararon todos los entregables.

De esta manera se nivelan las semanas que se han dedicado el proyecto, no afectando al número de semanas totales. Pero esto no es así para las horas empleadas en dichas semanas. Debido a que a la primera y tercera iteración se les dedicaron menos horas, mientras que a al resto de iteraciones se les dedicó más horas de las estimadas. En la iteración 2, por lo comentado previamente, se le dedicaron más horas. Por otro lado, las dos últimas iteraciones, debido a la fecha de entrega, se decidió incrementar las horas de trabajo, aproximadamente unas 27,5 horas semanales. De esta manera, se puede ver tanto en las siguientes gráficas 2.1 2.2 y en la tabla 2.7 como se han sobrepasado las horas estimadas, con un total de 347 horas.

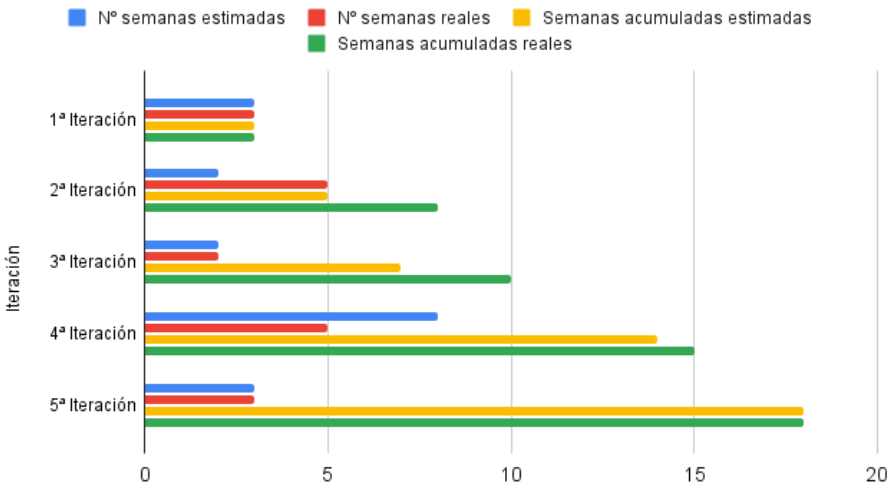


Figura 2.1: Comparación del número de semanas estimado y real en cada iteración

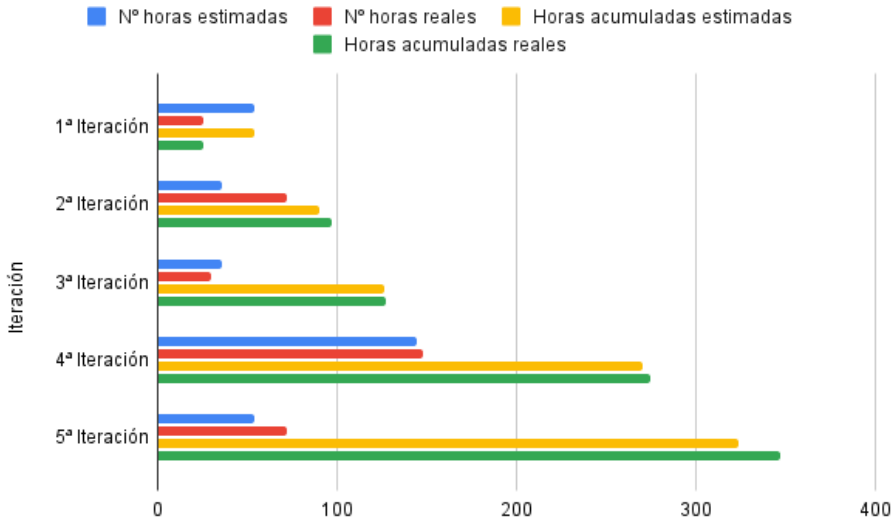


Figura 2.2: Comparación del número de horas estimado y real en cada iteración

Iteración	Nº horas estimadas	Nº horas reales	Horas acumuladas estimadas	Horas acumuladas reales
1ª Iteración	54	25	54	25
2ª Iteración	36	72	90	97
3ª Iteración	36	30	126	127
4ª Iteración	144	148	270	275
5ª Iteración	54	72	324	347

Tabla 2.7: Cálculo de horas estimadas frente a horas reales empleadas en el proyecto

Capítulo 3

Descripción de las iteraciones

3.1. Iteración 1

Siendo esta la iteración inicial, lo primero que se hizo fue establecer las bases del proyecto. Con la idea inicial de desarrollar una aplicación de alquiler o gestión de bicicletas, incluyendo además el factor de la gamificación, se buscó información sobre el contexto actual en el ámbito de este tipo de aplicaciones y en especial en la ciudad de Valladolid.

Se realizó un estudio de las aplicaciones similares que existen actualmente en el mercado, esto se comenta en la sección 1.3. De esta manera se pudo extraer información útil para desarrollar el proyecto, valorando los puntos fuertes y los débiles de las aplicaciones similares ya existentes.

También se establecieron los objetivos principales, la metodología que se va a segur, en este caso iterativa, ya que es la que mejor se adapta al proyecto y por último se definió la estructura del proyecto y como se organiza esta memoria.

Por último se realizó la planificación inicial, en la que se establecen un plan de trabajo para completar el proyecto dentro del margen temporal disponible. Se describen la planificación para cada iteración y el tiempo estimado que se va a emplear en cada una. Además, se describe el análisis de riesgos 2.1, y el plan de acción o mitigación de los mismos. Por último, se estimaron los costes que tendría el proyecto y en que entorno tecnológico se realizará el proyecto.

Durante esta iteración se han realizado dos reuniones con el tutor, en las que se han comentado las secciones realizadas para agregar información o refinarla.

Para el desarrollo de esta iteración se ha empleado al rededor de 3 semanas, lo cual cuadra con lo especificado en la planificación.

3.2. Iteración 2

Esta segunda iteración dio comienzo tras una reunión con el tutor del proyecto, estableciendo los trabajos a realizar, que junto con la planificación inicial, establecieron una hoja de ruta para esta iteración.

Se completó la documentación que restaba de la introducción y como tarea principal se investigaron y exploraron las diferentes opciones tecnológicas en lo referente al Front-End o aplicación móvil cliente. Aunque en la siguiente reunión de control de esta iteración se decidió también explorar las diferentes opciones para el Back End. Además, también se decidió realizar de la prueba de todas las tecnologías con una pequeña implementación que sirvió para asegurarse de que las tecnologías escogidas fueran las correctas, al mismo tiempo sirvieron para familiarizarse con los lenguajes y tecnologías.

3.2.1. Estudio de Tecnologías

En esta sección se van a detallar las diferentes opciones que se han barajado, explorado y que finalmente se han escogido para cada parte del proyecto. Se puede dividir en dos secciones, estas secciones son la aplicación Android como Front-End y el Back-End compuesto por la API y la base de datos que es consumida por el Front End.

3.2.2. Aplicación cliente y Front End

La parte principal de la aplicación móvil cliente es el mapa y sus funcionalidades, por lo que se hizo un análisis de cuál sería la mejor opción. Las dos opciones principales, debido a la madurez de ambas plataformas, son, Google Maps Platform [54] y Mapbox [55].

Se buscó una comparación inicial entre Google Maps y Mapbox [56][57], ambos SDK ofrecen las principales funcionalidad que el proyecto requiere, como mostrar un mapa y elementos interactivos sobre él, obtener direcciones para una ruta definida y crear mapas personalizados, en este caso para agregar los diferentes carriles bici en la ciudad de Valladolid.

Se ha escogido Mapbox debido a tres razones, buena y extensa documentación, la documentación de los productos de Google no es la mejor y a veces carece de explicaciones; Mapbox es de código abierto, por lo que ofrece mayor flexibilidad dando la oportunidad de modificar el código base; por último y la razón con más peso son los términos de privacidad de la API de Google Maps [58], en la que se especifica lo siguiente.

3.2.3 Restrictions Against Misusing the Services apartado (d), "No Re-Creating Google Products or Features. Customer will not use the Services to create a product or service with features that are substantially similar to or that re-create the features of another Google product or servicecombine data from the Directions API, Geolocation API, and Maps SDK for Android to create real-time navigation functionality substantially similar to the functionality provided by the Google Maps for Android mobile app".

Esta restricción podría suponer un problema en un futuro para el despliegue de la aplicación, ya que no se permite recrear productos de Google empleado el SDK de Google Maps, por lo que se ha preferido evitar cualquier inconveniente optando por el SDK de Mapbox.

En lo referente a las tarifas de uso de Mapbox, se ha comprobado que no suponen un problema para el desarrollo del proyecto en un ámbito de pruebas, ya que los límites de cuota de uso son lo suficientemente altos.

Habiendo escogido la plataforma de mapas se evalúa sobre que plataforma se va a implementar. Mapbox ofrece múltiples plataformas en las que se pueden implementar sus mapas [29], para Android e iOS ofrece una SDK nativa y para web ofrece una librería de JavaScript.

Inicialmente, se propuso realizar la aplicación móvil con una tecnología multi plataforma, de esta forma se podría disponer de la aplicación para las dos principales plataformas móviles, Android e iOS. Se consideraron Flutter [59] y React Native [60] como las principales opciones, ya que son las tecnologías para desarrollo de aplicaciones multi plataformas más maduras del ecosistema [61]. De entre estas dos opciones se escogió Flutter debido a que encajaba mejor con este proyecto, ya que a diferencia de React Native, dispone de una potente interfaz que soporta actualizaciones de la interfaz frecuentemente y complejas animaciones, parte clave para el desarrollo del proyecto, ya que principalmente se trata de un mapa dinámico e interactivo. También se consideró el gran soporte que tiene la plataforma por parte de Google y su reciente popularidad.

Dentro del repositorio de paquetes de Flutter [62], se localizaron dos de los paquetes principales para la implementación de los mapas de Mapbox en una aplicación Flutter, estos son, *flutter_mapbox_gl* [63] y *flutter_mapbox_navigation* [64], cabe resaltar que ambos paquetes, son implementaciones "3rd party", esto quiere decir que no es una implementación directamente proporcionada por Mapbox, sino que está realizada por personas ajenas al proyecto oficial de Mapbox. Se creó una aplicación de ejemplo para probar su funcionamiento tanto en dispositivos Android como en dispositivos iOS, este último mediante el uso de un simulador. Se utilizó la guía propia de los paquetes junto al código del siguiente repositorio [65]. Por parte del sistema Android no hubo grandes problemas, por otro lado, la implementación del código más básico para el uso de los mapas en iOS, fue más complicada debido a las diferentes versiones y configuraciones que son necesarias para que funcione el sistema de mapas. Por último se intentó implementar el sistema de navegación, pero no se pudo implementar debido a que se empleó la última versión de los paquetes, como se describe en esta *issue* o incidencia [66], el paquete de navegación para Flutter no funciona correctamente debido a incompatibilidades entre el lenguaje de programación de Flutter y la última versión de Mapbox. De este modo, se considera que para este proyecto, es mejor optar por el uso de últimas versiones, en vez de versiones anteriores del sistema de mapa, por lo que, al no poder implementar actualmente la funcionalidad de navegación en una aplicación Flutter, se descarta el desarrollo multi plataforma, dando paso a un desarrollo centrado en dispositivos Android.

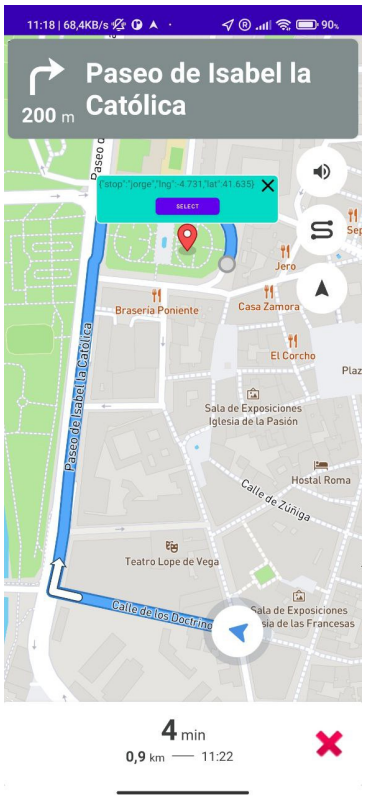
Tras decidir que el proyecto se va a implementar en Android, se comenzó a probar el SDK de mapa nativo de Mapbox [31] y el de navegación [32]. Se siguieron las guías de ambos productos y se comprobó que la diferencia de rendimiento entre los SDK nativos y los paquetes de Flutter, era mucho mejor en los SDK nativos. También se implementaron los ejemplos de navegación de este repositorio

[67] para explorar las diferentes opciones que se proporcionan y entender como podrían encajar en el futuro diseño e implementación del proyecto.

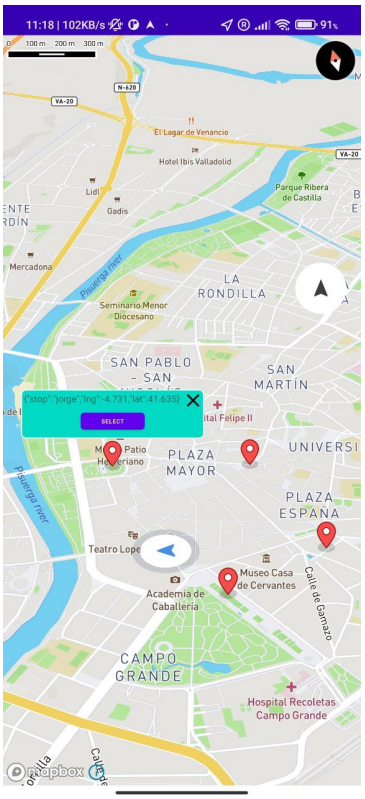
Además de todo lo mencionado previamente, la elección de Kotlin como lenguaje para el desarrollo de la aplicación Android, principalmente se basa en emplear las tecnologías más modernas, teniendo en cuenta esto Kotlin pasó a ser el lenguaje de programación recomendado para Android en 2019 [68] debido a múltiples factores, como un código más limpio y más legible que Java, el soporte de nuevas librerías como *Jetpack* y que además sigue siendo operativo con el código Java.

Por último, también cabe destacar que a petición del tutor, se buscó la forma de indicar el recorrido de los diferentes carriles bici que hay distribuidos a lo largo de la ciudad. Para esta tarea se ha empleado el editor de mapas que proporciona Mapbox Studio [33], este te permite agregar elementos, estilos y capas para personalizar el mapa. La ubicación de dichos carriles bicis se ha obtenido de la página de GEOCYL [69]. Desde esta página se ha exportado el mapa de Google Maps y se ha importado al Mapbox Studio para agregarlo como elemento al mapa que se va a emplear.

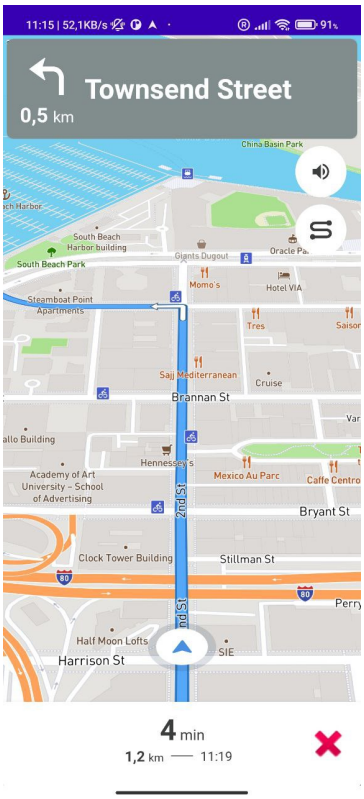
A continuación se muestran unas imágenes de una primera implementación del sistema de mapas. Con esta prueba se ha podido comprobar algunas de las funcionalidades que se emplearán en el desarrollo del proyecto, como puede ser agregar elementos interactivos al mapa para representar las paradas de bicis y el sistema de navegación.



(a) Visión ruta con marcador de parada



(b) Visión múltiple de paradas con información



(c) Pantalla resumen ruta realizada tras devolver la bici

Figura 3.1: Visión sistema de navegación y direcciones

3.2.3. Back End, API y Base de Datos

Para el desarrollo del Back-End se consideran necesarias dos partes esenciales. La creación de una API para la comunicación entre la aplicación móvil y la lógica de negocio, para gestionar de esta manera las operaciones sobre usuarios, rutas, paradas, alquileres y el sistema de puntuación. En la sección de requisitos 4.1 se detallan este tipo de funcionalidades.

La otra parte fundamental del Back End es la base de datos que se va a emplear para almacenar toda la información necesaria para el funcionamiento de la aplicación.

Para la API del Back-End se han considerado diferentes frameworks y diferentes lenguajes, entre ellos:

- **Laravel**[22] uno de los frameworks web más populares del lenguaje PHP, sirve tanto para aplicaciones monolíticas como para el desarrollo de APIs.
- **Django**[70] es un framework web para el desarrollo tanto de aplicaciones monolíticas como APIs. Se utiliza el lenguaje de Python.
- **Node**[71] es un entorno de ejecución para el lenguaje JavaScript empleado para el desarrollo de aplicaciones web y APIs. Actualmente, es la opción más popular para el uso de JavaScript del lado del servidor.

Estas tres tecnologías son las más populares en sus respectivos lenguajes de programación para el desarrollo de una aplicación Back-End como la que se busca en este proyecto.

Para la elección de la tecnología se valoraron comparaciones técnicas entre dichas tecnologías [72]. En el momento del desarrollo de este proyecto, el alumno no tiene conocimiento del lenguaje de programación Python y, por tanto, tampoco del framework Django. Por otro lado, sí que se conocen las otras tecnologías, habiendo realizado proyectos tanto con Laravel como con Node, en concreto el desarrollo de APIs Rest. Por lo que Django quedó descartado inicialmente debido a que sería necesario aprender tanto el lenguaje como el framework para el desarrollo del proyecto.

Para la elección entre Node y Laravel se buscaron comparaciones [72] para evaluar pros y contras del uso de ambas tecnologías. Las mayores diferencias son las facilidades y las herramientas que ya vienen por defecto en cada tecnología. En el caso de Laravel, esta tiene un ecosistema con paquetes que te permiten implementar funcionalidades que pueden ser consideradas esenciales, sin extender el frameworks con paquetes o librerías de terceros, como por ejemplo el sistema de APIs web, Autenticación, Testing y WebSockets entre otras. Por otro lado, Node no te provee de este tipo de ecosistema por defecto, sino que tienes que utilizar paquetes para el desarrollo de estas funcionalidades, por ejemplo para el desarrollo de APIs web, una de las alternativas es Expressjs [73].

A pesar de que Laravel en cuanto a rendimiento no está tan bueno como Node, todas las facilidades que te proporcionan los paquetes del ecosistema, junto a que el desarrollador del proyecto se siente más cómodo con este framework que con Node. Se vio que la mejor opción para el desarrollo del Back-End y la API fue emplear Laravel.

Por el lado de la base de datos, desde un principio se planteó el uso de una base de datos relacional, ya que en el momento del desarrollo del proyecto, no se conoce de forma fluida otro tipo de bases de datos, como podrían ser no relacionales. En concreto, se ha escogido PostgreSQL [37], debido a su fácil integración con Laravel y su facilidad de uso.

Como herramienta de para facilitar el desarrollo de la API se ha considerado Postman [38], ya que se ha empleado con anterioridad y permite definir colecciones de peticiones [74] para hacer llamadas HTTP. También se ha considerado el uso de [75] *Mock Servers* que te permite simular el flujo de peticiones, tanto las peticiones como las respuestas obtenidas, facilitando el desarrollo del Front End sin tener un Back End completamente operativo.

Por último, también se consideraron otras tecnologías que dependiendo del tiempo que se dispusiera para el desarrollo se podrían emplear para agregar funcionalidades. En concreto, el uso de Websockets, mediante la plataforma Pusher Channels [76]. El uso de esta plataforma junto a su sencilla implementación en Laravel, permitiría a la aplicación obtener actualizaciones en tiempo real del estado de las paradas y de la ubicación de las bicicletas.

3.3. Iteración 3

Durante esta iteración se ha realizado el proceso de análisis, detallado en la sección 4.1 y el diseño inicial de la aplicación, detallado en la sección 4.4. Durante el proceso de análisis de la aplicación se desarrollaron las historias de usuario, que se pueden sintetizar en los requisitos iniciales que se pueden encontrar a continuación.

Se ha establecido un código identificativo a cada requisito funcional y una prioridad que determinará el orden en el que se desarrollarán e implementarán las funcionalidades de la app, comenzando por las funcionalidades más necesarias o de alta prioridad y si se dispone del tiempo necesario agregar funcionalidades adicionales, o por contrario, si no se dispone de tiempo suficiente recortar los requisitos de menor prioridad.

Código	Nombre	Descripción	Prioridad
RF01	Visualizar paradas	El sistema debe permitir visualizar todas las paradas en el sistema con su información relacionada y su ubicación	Alta
RF02	Desbloquear bicicleta	El sistema debe permitir desbloquear una bicicleta que esté ubicada en una parada, reservada por ese mismo usuario o que no esté reservada y de esta manera poder iniciar una ruta	Alta
RF03	Iniciar una ruta	El sistema debe permitir iniciar una ruta desde una parada inicial donde se desbloqueará una bicicleta, hasta una parada final donde se bloqueará esa misma bicicleta	Alta
RF04	Bloquear bicicleta	El sistema debe permitir bloquear una bicicleta que previamente ha sido desbloqueada por el mismo usuario, dando como resultado la finalización de la ruta	Alta
RF05	Visualizar recompensas	El sistema debe permitir a un usuario visualizar sus recompensas obtenidas	Media
RF06	Obtener recompensas	El sistema debe permitir a un usuario obtener una recompensa a cambio de los puntos obtenidos al realizar rutas	Media
RF07	Login de usuario	El sistema debe permitir a un usuario registrado ingresar en la app para su posterior uso	Media
RF08	Reservar bicicleta	El sistema debe permitir reservar una bicicleta de una parada durante un tiempo para su posterior recogida sin que otros usuarios puedan acceder a ella	Media
RF09	Registro de usuario	El sistema debe permitir crear un usuario único para usar la app	Baja
RF10	Visualizar rutas	El sistema debe permitir a un usuario visualizar sus rutas realizadas con anterioridad y la información relacionada con estas	Baja
RF11	Búsqueda de paradas	El sistema debe permitir buscar las paradas cercanas a una dirección proporcionada por el usuario	Baja
RF12	Datos de usuario	El sistema debe permitir visualizar a un usuario sus datos en la app y poder modificarlos	Baja

Tabla 3.1: Requisitos funcionales

También se han desarrollado los requisitos no funcionales, estos hacen referencia a ciertas características técnicas que el sistema debe cumplir para el correcto funcionamiento de la app. Se ha seguido la guía FURPS [77], en la que se definen una serie de categorías en las que se pueden englobar los requisitos. A diferencia de los requisitos funcionales, a estos no se les ha asignado una prioridad, dado que son una serie de características necesarias para que el sistema funcione correctamente.

Código	Nombre	Descripción	Categoría
RNF01	Sistema Android	El sistema debe pemritir ejecutarse en un sistema Android, con como mínimo la versión 21 del SDK Android	Funcional
RNF02	Sistema Mapbox	El sistema de mapas debe ser Mapbox con la versión 10.0 como mínimo	Funcional
RNF03	Sistema Laravel	El sistema referente al servidor debe ser desarrollado con el framework Laravel y como mínimo la versión 9.0	Funcional
RNF04	Base de datos	El sistema debe utilizar la base de datos PostgreSQL con la versión 10.0 como mínimo	Funcional
RNF05	Facil de usar e intuitiva	El sistema utilizará Material Desing para la interfáz de usuario para que esta sea facil e intuitiva de utilizar para cualquier usuario que esté familiarizado con el uso de disposivos móviles	Usabilidad
RNF06	Gestión de errores	El sistema no mostrará errores que un usuario no sepa interpretar, como errores de base de datos o de sistema. Se mostrará un error genérico y no se detendrá la ejecución de la app	Fiabilidad
RNF07	Depuración	El sistema empleará logs para la depuración de la aplicación, tanto para el front end y el back end	Soporte
RNF08	Tiempo de respuesta	El sistema tendrá un tiempo de respuesta lo suficientemente bajo para que el usuario tenga una buena experiencia de usuario, para ello se configurará correctamente el servidor	Rendimiento

Tabla 3.2: Requisitos no funcionales

3.3.1. Requisitos de Información

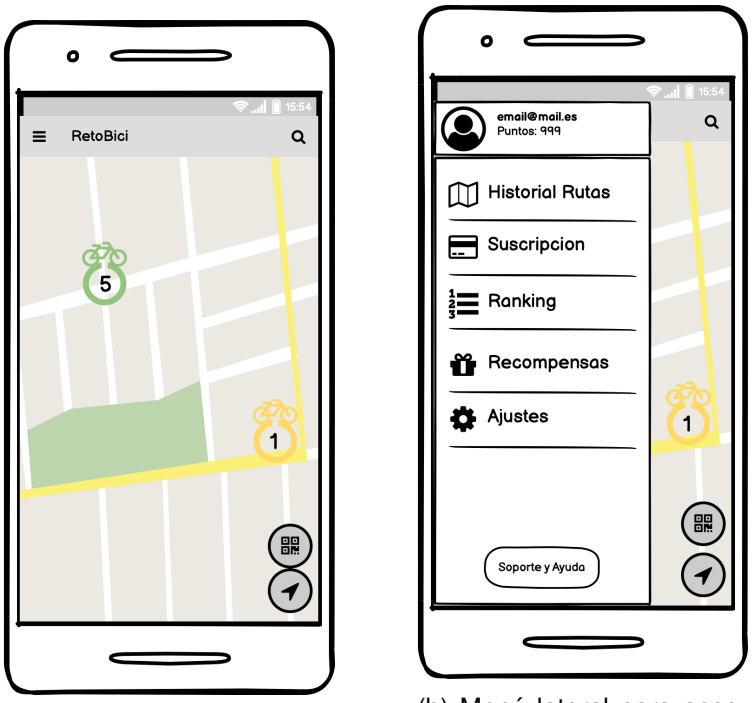
Los requisitos de información se podrían considerar como parte de los requisitos funcionales, pero en este caso se van a diferenciar, ya que a partir de estos se obtendrá el modelo de dominio inicial 4.1.3

Código	Nombre	Descripción
RI01	Información de **Usuario**	El sistema deberá almacenar el nombre de usuario, email, contraseña, imagen de perfil y puntuación obtenida. También se utilizará la Localización del usuario aunque no se almacenará.
RI02	Información de **Bicicletas**	El sistema deberá almacenar un identificador, el tipo de bicicleta y si es électrica el nivel de batería
RI03	Información de **Paradas**	El sistema deberá almacenar un identificador, la ubicación (longitud,latitud), dirección y el número de bicicletas y espacios disponibles
RI04	Información de **Rutas**	El sistema deberá almacenar un identificador, el usuario que ha realizado la ruta, la bicicleta con la que se ha realizado la ruta, las paradas de inicio y de final de la ruta, el tiempo estimado y el tiempo real empleado, la distancia recorrida y la puntuación obtenida al realizar la ruta
RI05	Información de **Recompensas**	EL sistema deberá almacenar un identificador, el nombre, la puntuación necesaria para ser obtenida y la descripción

Tabla 3.3: Requisitos de información

3.3.2. Mockups y diseño de la Interfaz de usuario

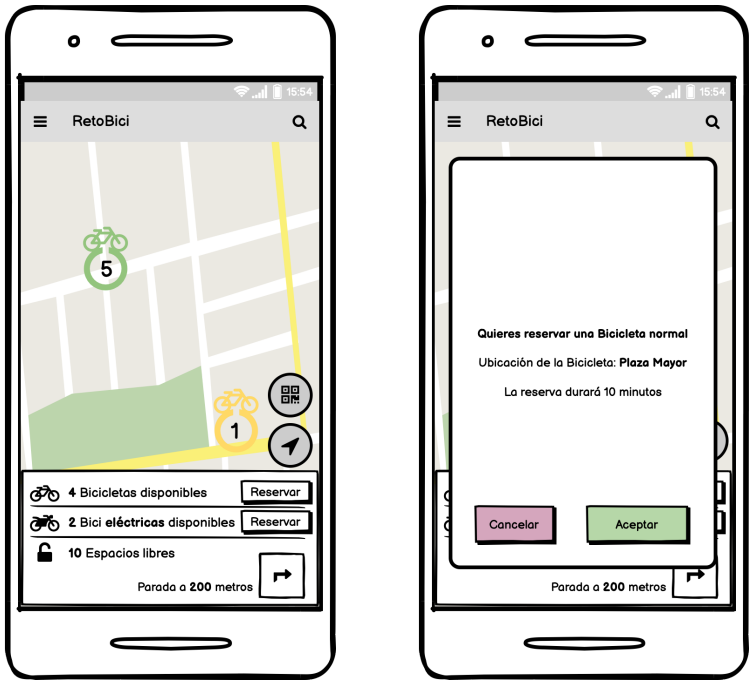
Previamente a comenzar con la implementación del código se realizaron una serie de bocetos o *mockups* de la interfaz de usuario. De esta manera se puede mostrar un prototipo de la aplicación a los clientes potenciales, al mismo tiempo que facilita la posterior implementación de la interfaz de usuario, ya que se parte de una idea. Estos bocetos se han realizado siguiendo las pautas de *Material Design* [78], esto es una serie de principios y buenas prácticas en cuanto al diseño de interfaces, además proporciona una implementación para sus componentes en Android, lo que facilita la aplicación de los bocetos a la app final.



(a) Pantalla principal con parada y número de bicis

(b) Menú lateral para acceder a las diferentes secciones

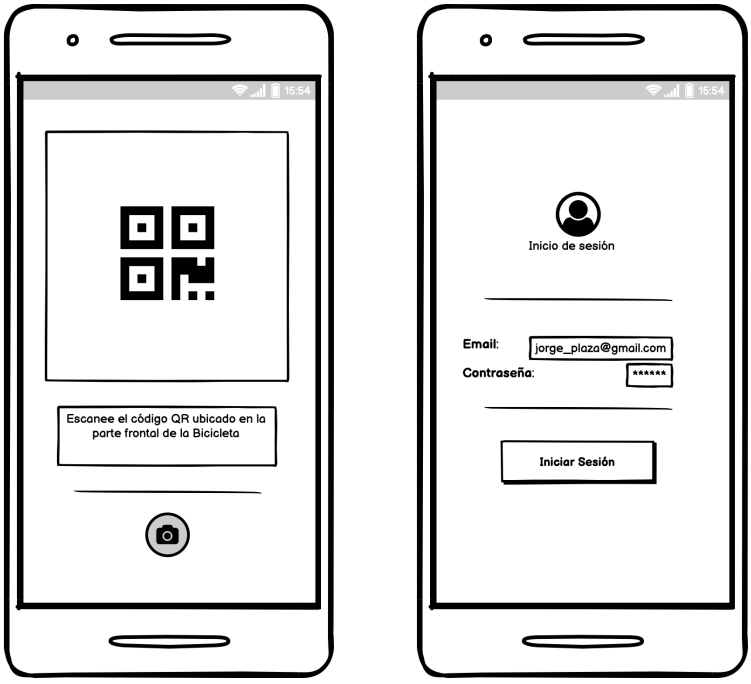
Figura 3.2: Boceto de la pantalla principal y del menú lateral



(a) Información de una parada tras seleccionarla

(b) Pantalla para reservar una bici de una parada

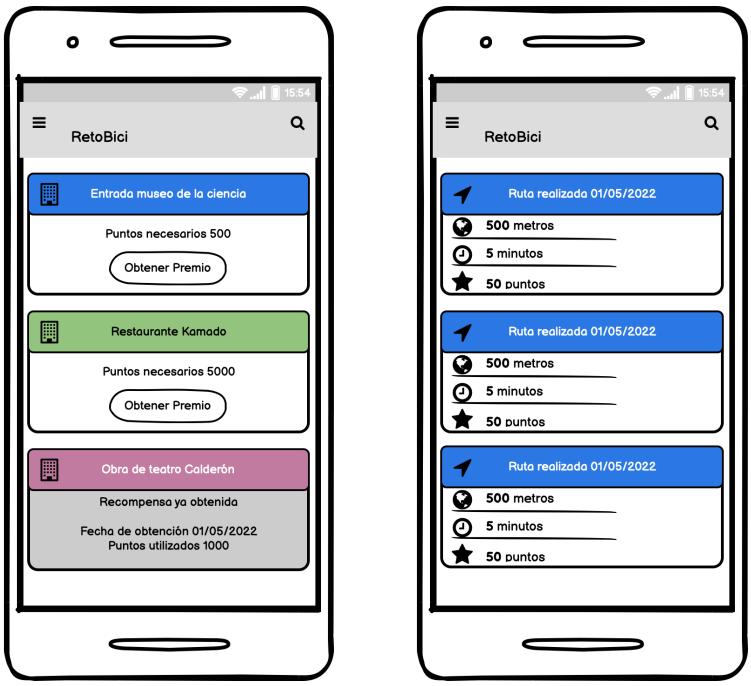
Figura 3.3: Boceto de las pantallas de visualización de paradas y reserva de bicicleta



(a) Escáner QR para desbloquear y bloquear una bici

(b) Pantalla de inicio de sesión de usuario

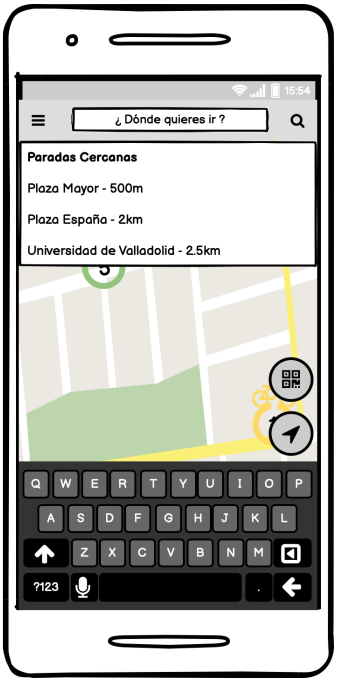
Figura 3.4: Bocetos de las pantallas de escáner códigos QR y pantalla de inicio de sesión



(a) Listado de recompensas y su información

(b) Listado de rutas realizadas por el usuario

Figura 3.5: Boceto de las pantallas con listado de recompensas y rutas



(a) Búsqueda de paradas para realizar ruta

(b) Resumen de la ruta que se ha realizado

Figura 3.6: Boceto de las pantallas de búsqueda de paradas y de resumen de ruta

(a) Ruta propuesta por el sistema tras búsqueda

(b) Sistema de navegación con las direcciones

Figura 3.7: Bocetos de las pantallas de visualización de una ruta y pantalla de direcciones

3.4. Iteración 4

Como todas las iteraciones, esta también comenzó con una reunión en la que se organizaron las tareas a realizar. En este caso se decidió comenzar por la implementación de los requisitos en la parte del Front End para posteriormente implementar la parte del Back End.

Tras dos semanas se tuvo otra reunión en la que, viendo el avance del proyecto y la proximidad con las fechas de entrega, se decidió recortar los requisitos que abarcaría el proyecto, quedando los requisitos con prioridad más baja descartados para futuras mejoras de la aplicación. En la tabla 3.4 se pueden ver los requisitos que fueron recortados.

Código	Nombre	Descripción	Prioridad
RF09	Registro de usuario	El sistema debe permitir crear un usuario único para usar la app	Baja
RF10	Visualizar rutas	El sistema debe permitir a un usuario visualizar sus rutas realizadas con anterioridad y la información relacionada con estas	Baja
RF11	Búsqueda de paradas	El sistema debe permitir buscar las paradas cercanas a una dirección proporcionada por el usuario	Baja
RF12	Datos de usuario	El sistema debe permitir visualizar a un usuario sus datos en la app y poder modificarlos	Baja

Tabla 3.4: Requisitos funcionales descartados en la iteración 4

Posteriormente, se tuvieron otras dos reuniones, pero con un espacio temporal entre ellas de una semana en vez de dos, debido a que la fecha de entrega del proyecto se acercaba y se quería llevar un seguimiento más cercano del avance del proyecto.

Respecto a las funcionalidades implementadas, tanto del Front End como del Back End se desarrollaron de forma independiente. Esto quiere decir, como son dos sistemas programados con tecnologías y lenguajes de programación diferentes, se desarrollaron incomunicados inicialmente. Para en el último tramo del proyecto conectar ambas partes y formar el sistema completo. Para esta tarea se empleó principalmente la herramienta Postman [38] para la emulación de los recursos necesarios sin que estos estuvieran implementados, como las llamadas a la API y las respuestas de la misma.

3.4.1. Postman

De la herramienta de Postman se utilizaron principalmente dos herramientas en esta iteración para facilitar la implementación tanto del Front End como del Back End.

La primera son las *Collections* [74] o colecciones de peticiones HTTP, con ellas se pueden definir como van a ser las llamadas a la API del Back End, indicando los parámetros, las cabeceras o la autenticación necesaria para las llamadas. De esta manera se pudo ir probando los requisitos implementados en el Back End sin que fuera necesario ser usados por el Front End.

De forma análoga, para simular las llamadas a la API necesarias del Back End desde la aplicación Android, se utilizó un *Mock Server* o Servidor simulado [75]. Este permite desplegar un servidor en una dirección URL que te permite emplear las mismas peticiones o llamadas definidas en las *Collection*

para emular un servidor funcional, de manera que no es necesario tener implementadas todas las funcionalidades correspondientes en el Back End. Para su uso solo es necesario definir un ejemplo de respuesta que se obtendrá si se llama desde cualquier parte utilizando la URL proporcionada por el *Mock Server*.

Postman también te facilita el proceso de documentación de la API, generado automáticamente la documentación de las Colecciones de llamadas junto a los ejemplos que te devolvería el *Mock server*. La documentación de la API del proyecto se encuentra publicada en [79].

3.4.2. Front End

Lo primero que se decidió implementar fue la interfaz básica de la aplicación Android, ya que resultaría más fácil trabajar en el resto de funcionalidades cuando esta tuviera unas bases. En esta interfaz se establecieron los menús, el mapa principal y alguno de los botones.

Tras la interfaz básica se decidió dedicar varios días para investigar y organizar la arquitectura y las librerías de Android que se iban a utilizar. La arquitectura empleada está detallada en profundidad en la sección 4.4.1. De esta manera se establecieron las bases de como se iban a organizar los paquetes y las clases empleadas en el sistema Android para facilitar el futuro desarrollo de las funcionalidades.

Con la arquitectura base y la interfaz básica se comenzó con la implementación de los tres requisitos que se consideraron fundamentales. El orden fue el siguiente, primero la visualización de las paradas y su información, así como la interacción con ellas en el mapa. En segundo lugar, se implementó el desbloqueo de una bicicleta junto al escáner de QR. Como tercer requisito implementado se visualizaron las recompensas disponibles.

En este punto es cuando se recortaron los requisitos mencionados con anterioridad, a la vez que se indicaron cuáles iban a ser los siguientes requisitos a implementar. Se tomó la decisión de centrarse en los requisitos relacionados con la realización de las rutas y la reserva de bicicletas. De esta manera se implementaron, en el siguiente orden, el inicio y seguimiento de la ruta, el bloqueo de la bicicleta y por consiguiente finalización de la ruta y por último la reserva de bicicletas en de una parada con bicicletas disponibles.

Con todos los requisitos de alta prioridad implementados se consideró que el código era difícil de manejar y mantener. Debido a la naturaleza del mismo, se necesitan hacer llamadas asíncronas al Back End para obtener la información necesaria ubicada en el servidor y posteriormente actualizar la interfaz de usuario. Se utilizan dos herramientas, las corrutinas de Kotlin [21] y la librería *LiveData* [80] de Android. El flujo de ejecución es el siguiente, primero el usuario realiza una acción que desencadena en el lanzamiento de una corrutina en segundo plano, esta corrutina hace la llamada al Back End y espera la respuesta, mientras tanto el usuario puede seguir haciendo otras actividades en la aplicación, ya que no se ha bloqueado la ejecución. Cuando la corrutina obtiene la respuesta del servidor, esta es transformada al modelo de dominio correspondiente y almacenado en un objeto *LiveData*. La interfaz de usuario se encarga de observar los cambios del *LiveData* y modificar la interfaz según sea necesario. Este flujo de ejecución dio lugar a un código difícil de mantener, ya que los objetos *LiveData* son

compartidos y observados en diferentes puntos de la interfaz, por lo que era complicado entender en que punto exacto estaba el estado de la información, dando lugar a comportamientos indeseados en la interfaz.

Para solventar lo previamente comentado se decidió utilizar las *sealed classes* de Kotlin [19] junto a la expresión *when* de Kotlin, similar a la típica expresión *switch*. La combinación de estas herramientas establecieron una serie de clases que representaban los estados de la información y los modelos, como por ejemplo si una ruta estaba activa o si el usuario disponía de una reserva. Esto no solo mejoró la calidad del código, sino que también era más mantenible y sencillo de leer.

Como ultima fase de implementación en el Front End se agregó lo relacionado con el usuario, esto son, los requisitos de *login* o inicio de sesión y la obtención de recompensas. Esto también engloba el manejo de la autenticación, en este caso con un *Bearer Token* que es expedido en el Back End con Laravel Sanctum [26] y almacenado en la aplicación Android de forma local para mantener una sesión que es enviado en las peticiones al servidor que deban ser autenticadas, como la reserva o el desbloqueo de una bicicleta.

3.4.3. Back End

El desarrollo y la implementación de los requisitos en la parte del servidor se dividió en dos bloques, el primero la implementación de los modelos y de la base de datos y el segundo la implementación de los controladores y de la API.

Los modelos y la base de datos en Laravel están altamente relacionados. Laravel utiliza un Eloquent, un ORM u *Object Relational Mapping* [81] que facilita la creación e interacción entre los modelos en el código con la base de datos. En Laravel se definen las *migrations* [82] que permiten definir la estructura de tablas de tu base de datos a la vez que se definen los atributos de los modelos o clases en el código. También se definieron las relaciones entre modelos utilizando la herramienta que Laravel te proporciona, *Eloquent Relationships* [83] que te permiten obtener los distintos modelos relacionados directamente desde el código sin tener que escribir consultas complejas a la base de datos en las que intervengan múltiples tablas.

También se definieron *Database Seeder* [84]. Estos facilitan rellenar la base de datos con información de una manera estable, es decir, si se decide resetear la información de la base de datos, siempre se obtendrá el mismo estado en la base de datos. Esto es de gran utilidad para establecer casos de uso o escenarios concretos.

Por último se implementaron los controladores de los modelos y la API. Estas secciones es donde está ubicada principalmente la lógica de negocio, ya que es donde se reciben los datos del Front End, se transforman y se almacenan. Se implementó un controlador por cada modelo existente en el Back End y en cada uno de ellos se agruparon los métodos que manejarían dichos modelos.

3.5. Iteración 5

Esta última iteración, como en las demás, también comenzó con una reunión en la que se cerraba el código implementado para poder centrarse en la memoria. Solo se hizo esta reunión inicial y otra tras las dos semanas que duró esta iteración, ya que los trabajo a realizar se consideraron que se podían revisar de forma asíncrona.

A lo largo de esta iteración se completaron los capítulos que faltaban de la memoria y se corregían los escritos con anterioridad con ayuda del tutor.

También se organizaron los entregables, tanto la aplicación Android en forma de APK, como el proyecto Laravel, que puede ser desplegado en la revisión del código. De esta manera se realizó el manual de uso de la aplicación Android y el manual de despliegue del Back End.

Capítulo 4

Estado final de la aplicación

A lo largo de este capítulo se va a desarrollar tanto el análisis como el diseño de la aplicación, incluyendo requisitos, historias de usuario, diagramas y técnicas o patrones que se han empleado.

4.1. Análisis

En esta sección se van a desarrollar los requisitos finales del sistema, requisitos funcionales, no funcionales y de información. También se incluyen los diagramas pertinentes para representar el Modelo de Dominio y para cada historia de usuario se agrega el diagrama de actividad representando la interacción entre el usuario, el sistema Front End y el sistema Back End.

4.1.1. Historias de Usuario

Se van a desarrollar los requisitos funcionales como historias de usuario para mejor comprensión de los mismos. Se han ordenado por prioridad y en el orden que se considera que se implementarán. Cada historia de usuario está relacionada con su requisito funcional.

ID	HU01 - RF01
Nombre	Visualización de paradas
Prioridad	Alta
Riesgo	Media
Descripción	Como usuario quiero poder visualizar las diferentes paradas de bicicletas disponibles alrededor de la ciudad y poder conocer la información relativa a las mismas, como número de bicicletas disponibles y su tipo, ya sea normales o eléctricas y el número de espacios libres en dicha parada
Validación	- Quiero poder visualizar el número de bicicletas - Quiero poder visualizar el número de espacios de estacionamiento disponibles - Quiero poder visualizar y seleccionar todas las paradas alrededor de la ciudad

Tabla 4.1: Historia de usuario Visualización de paradas

ID	HU02 - RF02
Nombre	Desbloquear bicicleta
Prioridad	Alta
Riesgo	Bajo
Descripción	Como usuario quiero poder desbloquear o alquilar una bicicleta desde la aplicación
Validación	- Quiero poder seleccionar el tipo de bicicleta que quiero desbloquear - Quiero poder desbloquear la bicicleta seleccionada de una parada - Quiero poder desbloquear la bicicleta escaneando el código QR incorporado en la misma bicicleta o en la parada - Quiero que se registre el inicio de la ruta tras desbloquear una bicicleta

Tabla 4.2: Historia de usuario Desbloquear bicicleta

ID	HU03 - RF03
Nombre	Realizar ruta
Prioridad	Alta
Riesgo	Alto
Descripción	Como usuario quiero poder iniciar una ruta desbloqueando una bicicleta
Validación	- Quiero que la ruta pase por dos paradas de bicicletas, la primera para coger la bici y la segunda para depositarla - Quiero que la primera parada esté ubicada lo mas cerca de la ubicación de inicio de la ruta - Quiero que la segunda parada esté ubicada lo mas cerca posible de la ubicación de destino de la ruta - Quiero poder desbloquear y obtener una bicicleta en la primera parada de la ruta - Quiero poder bloquear y depositar la bicicleta previamente obtenida en la segunda parada de la ruta - Quiero poder tener la posibilidad de bloquear y depositar la bicicleta previamente obtenida en la misma parada de inicial - Quiero poder visualizar el resumen de la ruta que he realizado con información como distancia recorrida, tiempo empleado y puntuación obtenida

Tabla 4.3: Historia de usuario Realizar ruta

ID	HU04 - RF04
Nombre	Bloquear bicicleta
Prioridad	Alta
Riesgo	Bajo
Descripción	Como usuario quiero poder bloquear una bicicleta desde la aplicación tras haberla desbloqueado
Validación	- Quiero poder depositar la bicicleta desbloqueada en una parada con al menos un espacio disponible - Quiero poder bloquear la bicicleta escaneando el código QR incorporado en la parada tras haberla depositado - Quiero que se registre la ruta que he realizado tras depositar la bicicleta

Tabla 4.4: Historia de usuario Bloquear bicicleta

ID	HU05 - RF05
Nombre	Visualizar recompensas
Prioridad	Alta
Riesgo	Bajo
Descripción	Como usuario quiero poder visualizar las recompensas disponibles y las recompensas que ya he obtenido con anterioridad
Validación	- Quiero poder visualizar una lista de las recompensas disponibles y sus requisitos para obtenerlas - Quiero poder visualizar una lista de las recompensas obtenidas con anterioridad y su información

Tabla 4.5: Historia de usuario Visualizar recompensas

ID	HU06 - RF06
Nombre	Login de un usuario
Prioridad	Media
Riesgo	Bajo
Descripción	Como usuario quiero poder acceder a la aplicación a través de una pantalla de login
Validación	- Quiero poder introducir mi email y contraseña para acceder a mi perfil de usuario - Quiero que mi email y mi nombre de usuario sean únicos

Tabla 4.6: Historia de usuario Login de usuario

ID	HU07 - RF07
Nombre	Obtener recompensas
Prioridad	Media
Riesgo	Bajo
Descripción	Como usuario quiero poder obtener recompensas gracias al sistema de puntuación
Validación	- Quiero poder utilizar mi puntuación para obtener recompensas

Tabla 4.7: Historia de usuario obtener recompensas

ID	HU08 - RF08
Nombre	Reservar bicicleta
Prioridad	Media
Riesgo	Medio
Descripción	Como usuario quiero poder reservar una bicicleta durante un tiempo para posteriormente desbloquearla cuando me encuentre cerca de la parada
Validación	- Quiero poder reservar una bicicleta disponible de una parada que seleccione - Quiero que la reserva se mantenga 10 minutos para que durante ese periodo nadie pueda desbloquear la bici bloqueada - Quiero poder elegir que tipo de bici puedo reservar

Tabla 4.8: Historia de usuario Reservar bicicleta

4.1.2. Diagramas de actividad

En esta sección se muestran los diagramas de actividad de las historias de usuario. En estos diagramas se puede ver como interaccionan cada parte del sistema y el usuario.

Hay que destacar que en la gran mayoría de los procesos la llamada a la API se hace en un proceso en segundo plano para no bloquear el hilo principal de la ejecución, de esta manera se puede seguir interactuando con la aplicación mientras se completa el proceso en segundo plano. Esto se ve reflejado en los diagramas con los nodos *fork* y *join*.

4.1.2.1. HU01 Visualizar Paradas

La visualización de paradas (ver figura 4.1) ocurre al inicio de la aplicación sin que el usuario tenga que realizar ninguna acción, pero también ocurre en ciertos momentos para actualizar la información de las paradas, como cuando se desbloquea, bloquea o reserva una bicicleta. Cuando se completa la llamada a la API se obtiene una lista con las Paradas que hay que mostrar en el mapa. La interfaz observa el cambio de estado y muestra las distintas paradas en el mapa.

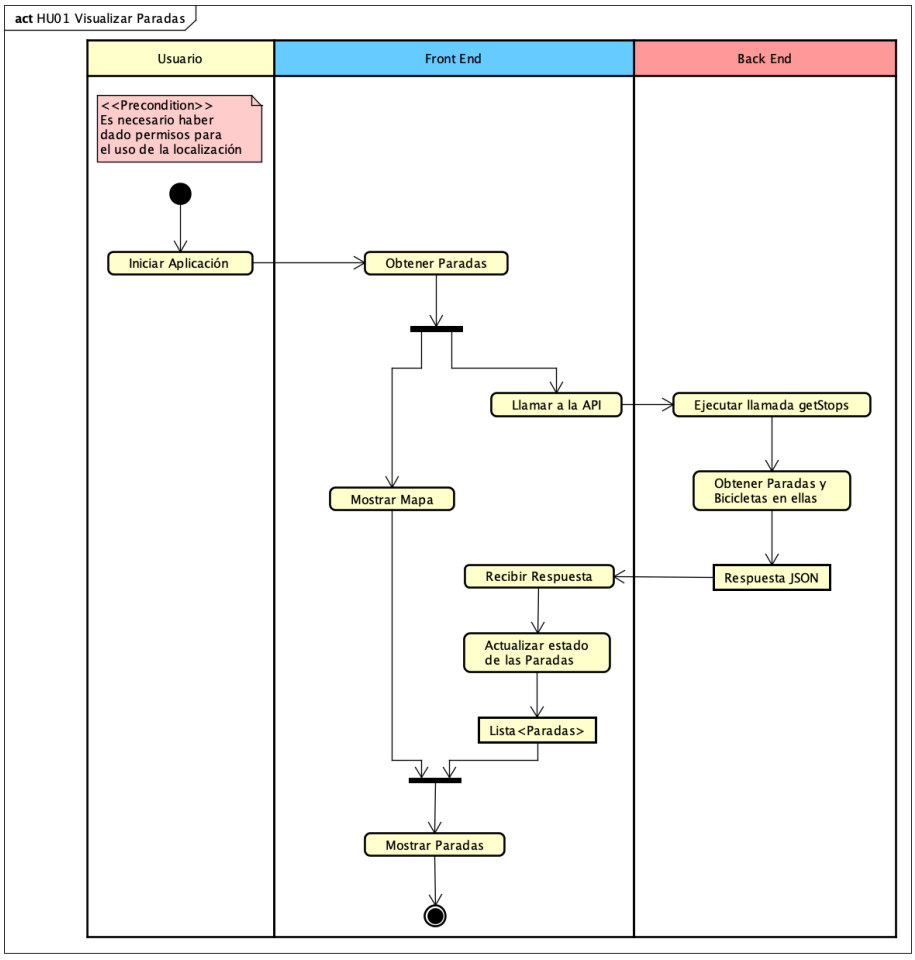


Figura 4.1: Diagrama actividad de historia de usuario Visualizar Parada HU01

4.1.2.2. HU02 Desbloquear bicicleta

El usuario puede desbloquear una bicicleta (ver figura 4.2) accediendo a la pantalla del Escáner QR desde el mapa. Para comprobar si se puede desbloquear la bicicleta, el Back End comprueba si el usuario tiene una reserva, lo que le permitiría desbloquearla, de no tener la reserva, es necesario comprobar si hay más bicicletas del mismo tipo disponibles y que no estén reservadas por otros usuarios. Puede darse el caso que haya una bicicleta en una parada, pero que ya esté reservada por otro usuario, por lo que si se escanea el código QR no va a estar disponible para ser desbloqueada. Si hay disponibilidad para desbloquear la bicicleta, se recibe como respuesta la información de la bicicleta y se genera el objeto *Bicicleta* que se muestra en la interfaz y se habilita el botón para iniciar la ruta con esa bicicleta. En el caso de que la bicicleta no esté disponible, se muestra el mensaje de que no está disponible.

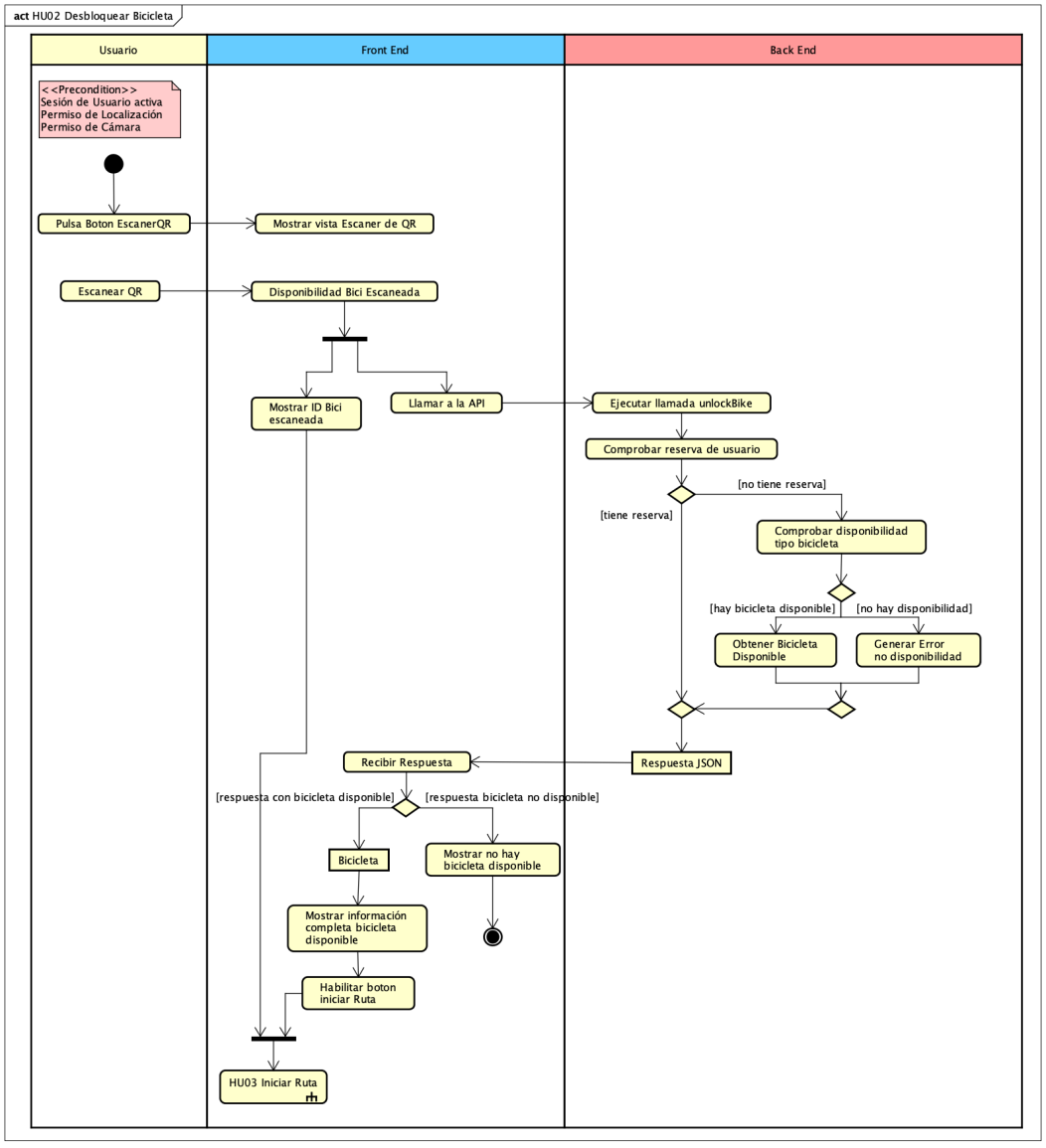


Figura 4.2: Diagrama actividad de historia de usuario Desbloquear Bicicleta HU02

4.1.2.3. HU03 Iniciar ruta

Para iniciar (ver figura 4.3) una ruta se parte desde la finalización de la historia de usuario anterior de Desbloquear Bicicleta (ver figura 4.2). Tras pulsar el botón de iniciar ruta se realiza una llamada al Back End para crear la nueva *Ruta*, con su información relacionada que se conoce desde el inicio, como la parada inicial, el usuario que realiza la ruta y con que bicicleta. Tras generar la ruta y guardarla en la base de datos, se envía como respuesta al Front End. En el sistema de Android se recibe la respuesta y se genera el objeto Ruta iniciando el temporizador para obtener posteriormente el tiempo de ruta. Luego se muestra en la parte inferior de la pantalla del mapa, razón por la que se vuelven a cargar las paradas ahora actualizadas, ya que una bici ha sido desbloqueada de una de ellas.

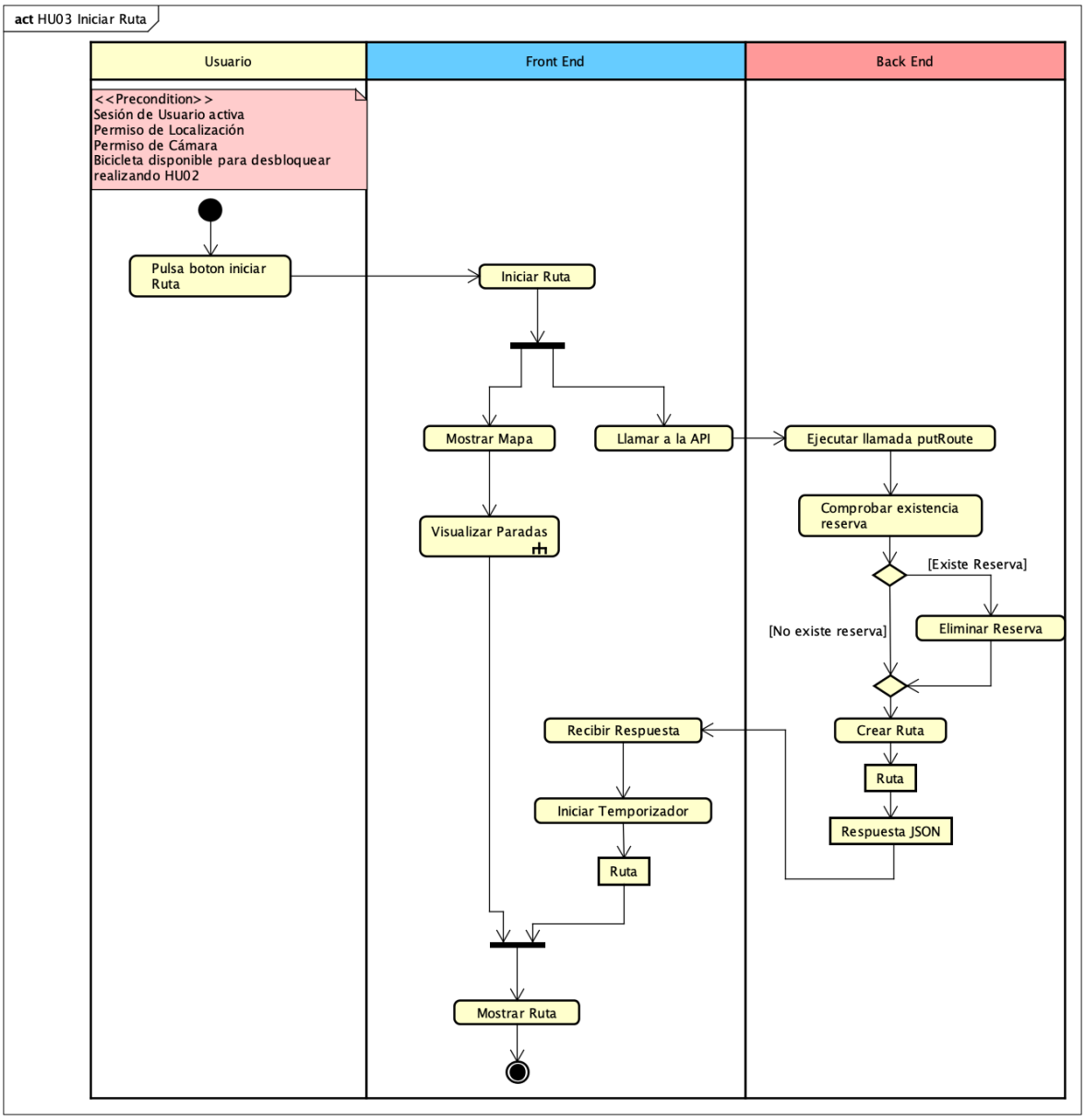


Figura 4.3: Diagrama actividad de historia de usuario Iniciar Ruta HU03

4.1.2.4. HU04 Bloquear bicicleta

Para bloquear la bicicleta que se está empleando en una ruta (ver figura 4.4) es necesario escanear el QR de la **parada** en la que se quiera depositar la bicicleta, de esta manera se realiza una llamada al Back End donde se comprueba que hay espacios disponibles para poder depositar la bicicleta. Cuando se recibe la respuesta se comprueba si es válida se habilita el botón que permite finalizar la ruta y anclar la bicicleta a la parada (ver figura 4.5) y si la respuesta no permite bloquear la bicicleta se deniega el botón de finalizar ruta.

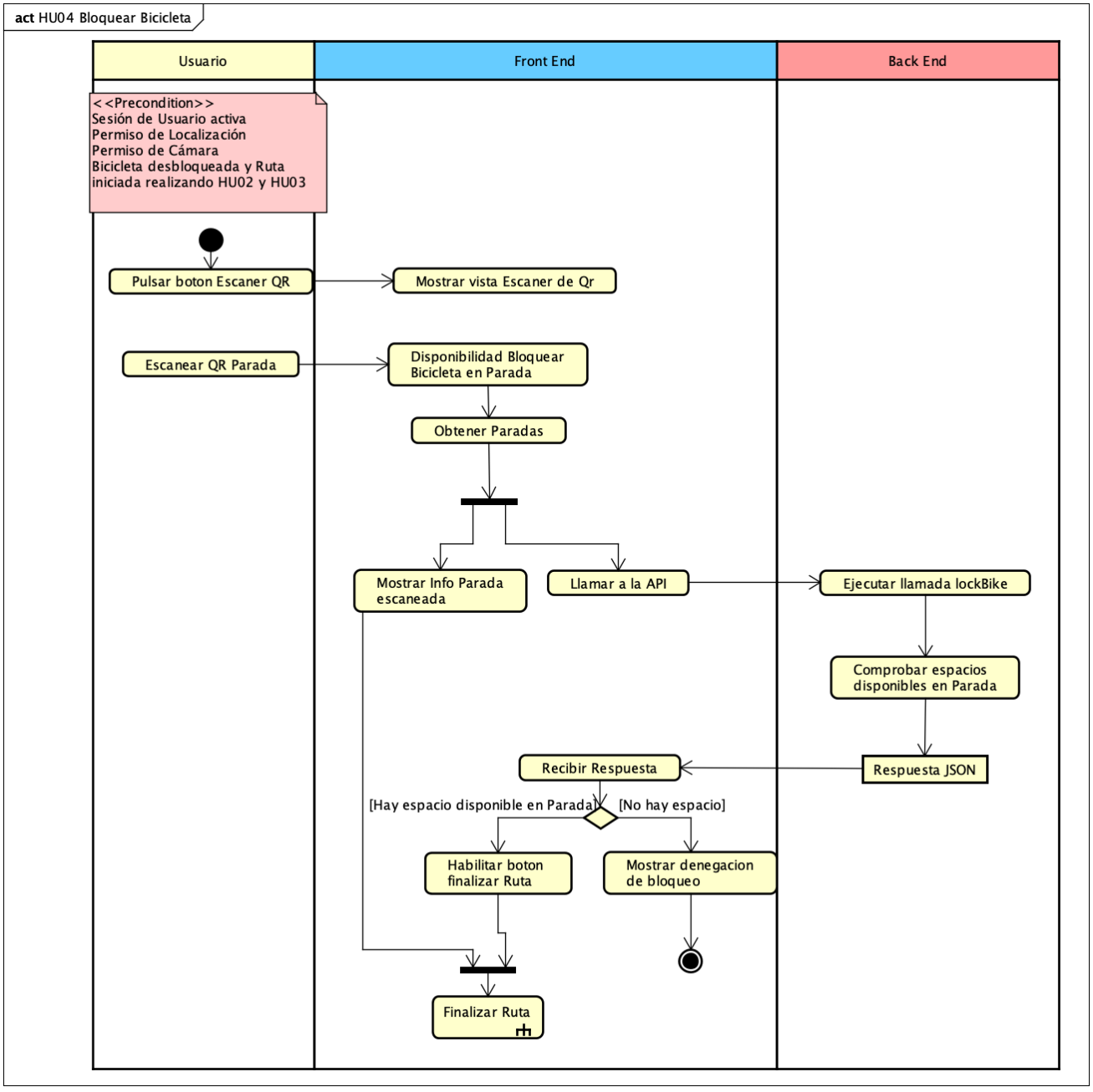


Figura 4.4: Diagrama actividad de historia de usuario Bloquear Bicicleta HU04

4.1.2.5. HU03-Parte 2 Finalizar ruta

Este diagrama de actividad consiste en la segunda parte de la historia de usuario de Realizar Ruta 4.3. Para finalizar una ruta en curso (ver figura 4.5) se debe haber completado la historia de usuario anterior de Bloquear Bicicleta (ver figura 4.4) que habilita el botón para finalizar la ruta. Cuando se pulsa el botón se realiza la llamada al Back End este hace una llamada a la API de Mapbox Navigation [32] para obtener la ruta, proporcionando las coordenadas de inicio y coordenadas de fin, como respuesta se obtiene la duración estimada, la distancia y otros datos. Con los datos obtenidos se calcula la puntuación y se actualiza el usuario. Por último, la bicicleta que se ha usado en la ruta es asignada a esa parada. Tras actualizar la base de datos con la nueva información, se envía como respuesta al Front End la Ruta completa con todos los datos. En la interfaz se recibe la respuesta con la Ruta completa para ser mostrada en la pantalla de resumen de ruta.

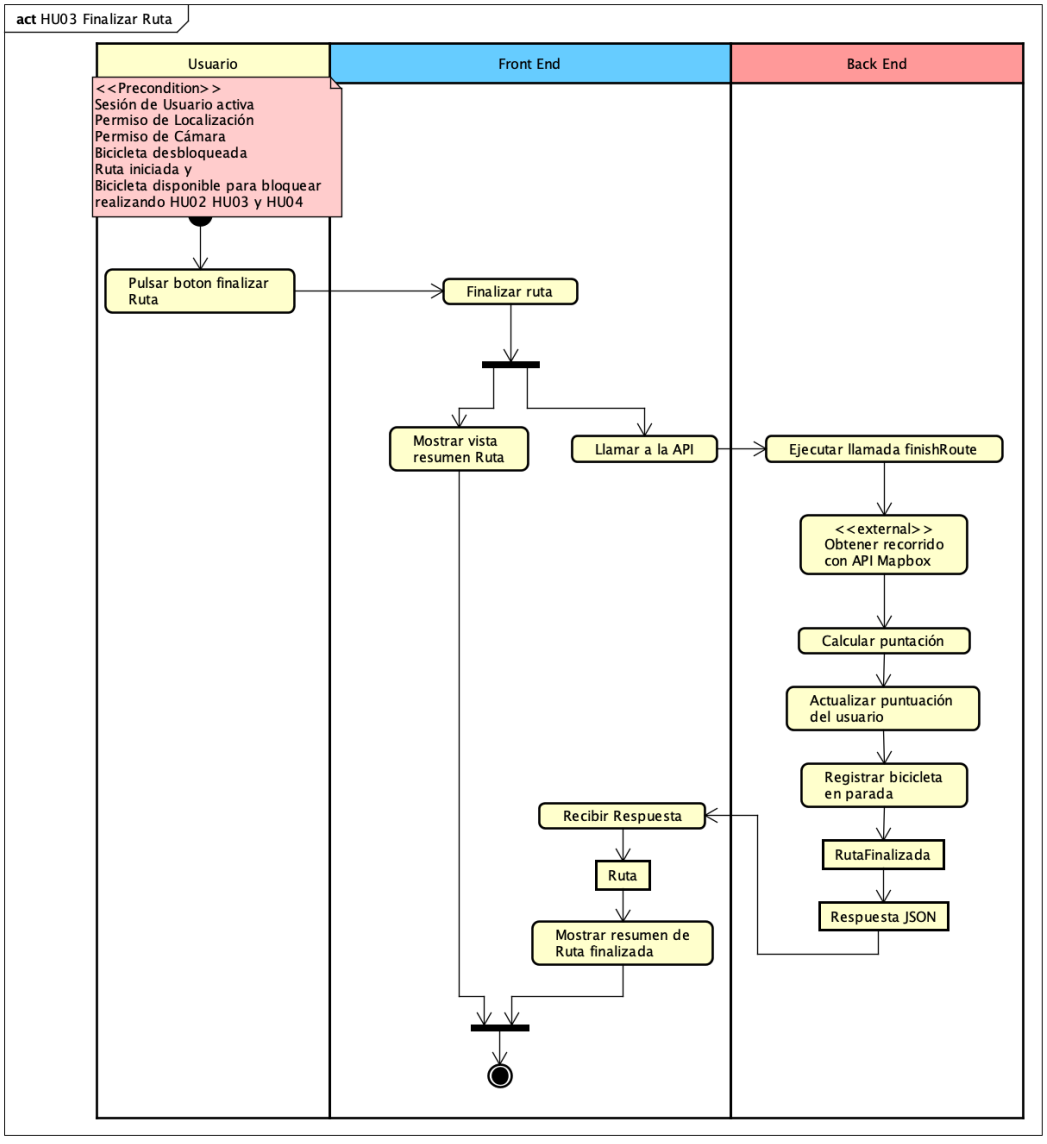


Figura 4.5: Diagrama actividad de Historia de usuario Finalizar Ruta HU03 segunda parte

4.1.2.6. HU05 Visualizar Recompensas

Para visualizar las recompensas (ver figura 4.6) no es necesario iniciar sesión. Si no se ha iniciado sesión se visualizan todas las recompensas y no se podrá obtener ninguna. En el momento en el que se inicie sesión se podrán visualizar las recompensas obtenidas y las que aún no se han obtenido. Hay que destacar que las ya obtenidas no es posible obtenerlas otra vez. Gracias al token de autenticación se puede saber en el Back End si el usuario tiene una sesión activa o no y fácilmente obtener las recompensas asociadas al usuario.

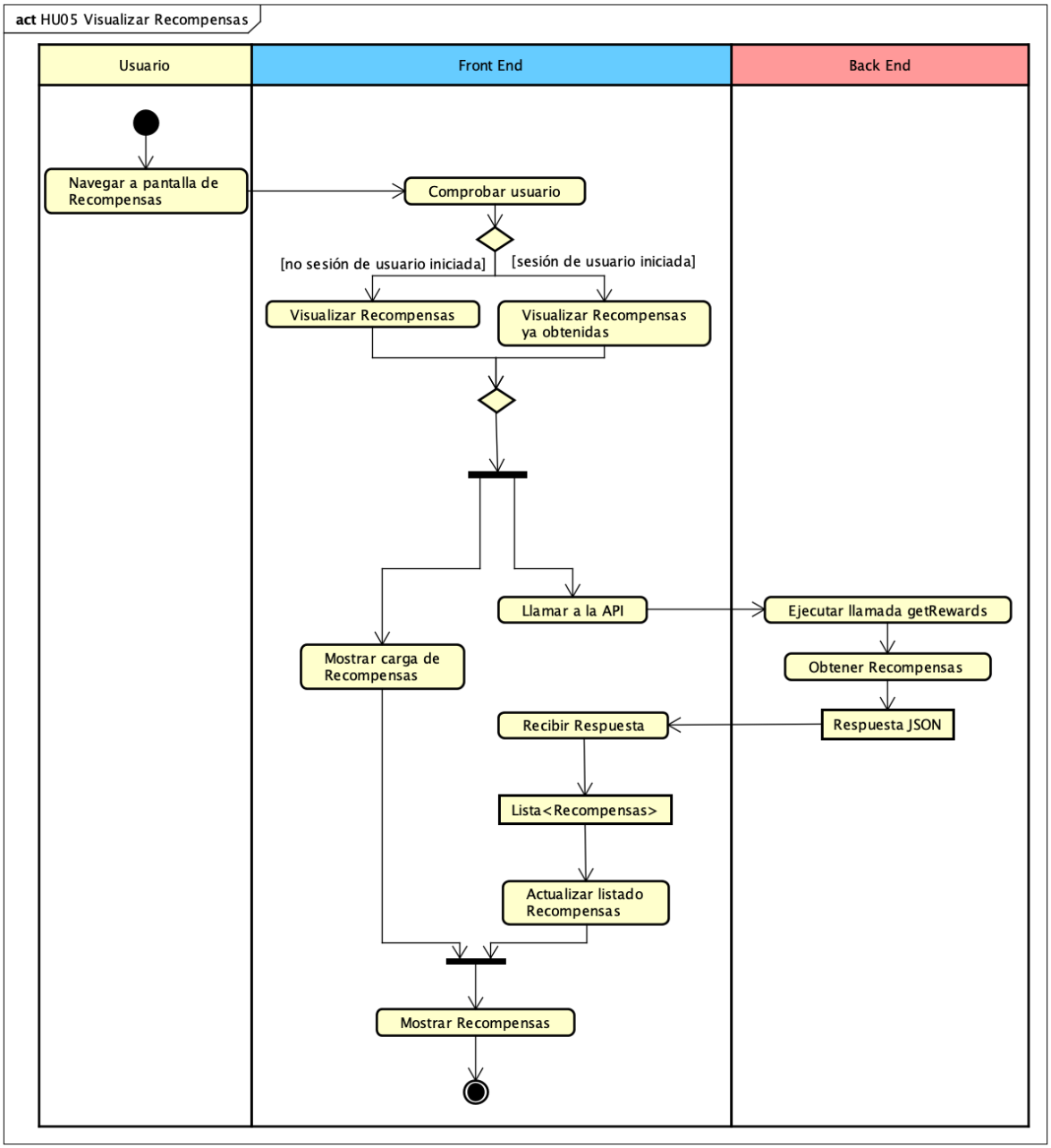


Figura 4.6: Diagrama actividad de historia de usuario Visualizar Recompensas HU05

4.1.2.7. HU06 Login de usuario

El proceso de login de usuario (ver figura 4.7) se realiza utilizando el formulario para iniciar sesión, este formulario valida el formato de la entrada del usuario, comprobando que la entrada del email sea un email válido y que la contraseña tenga más de seis caracteres. Cuando las credenciales introducidas tienen un formato correcto, se realiza una llamada al Back End donde validan las credenciales. Si las credenciales son válidas se genera un *Bearer Token* y se devuelve para ser almacenado en la sesión de la aplicación Android, en caso contrario se responde con un mensaje de error que será mostrado por el Front End. El token generado y almacenado es enviado en cada petición que necesite autenticación de usuario. Tras cerrar la sesión, este token es revocado y no se podrá utilizar de nuevo.

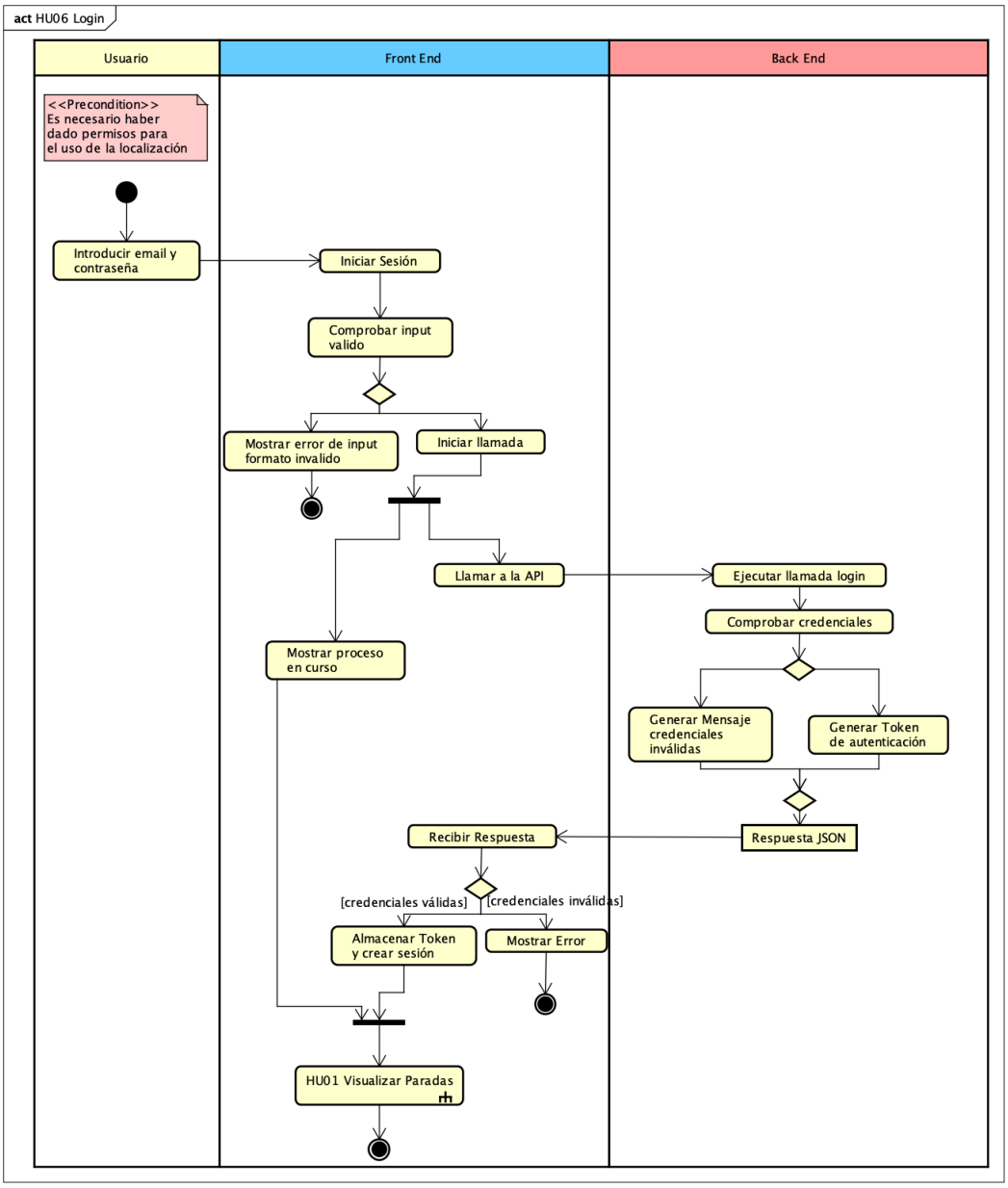


Figura 4.7: Diagrama actividad de historia de usuario Login de usuario HU06

4.1.2.8. HU07 Obtener recompensas

Para obtener una recompensa (ver figura 4.8) es necesario iniciar sesión (ver figura 4.7). Tras navegar hasta la pantalla donde se muestran las recompensas sin canjear, una recompensa se puede obtener pulsando el botón correspondiente y si se tienen los puntos necesarios. Tras la llamada al Back End se comprueba que tenga los puntos disponibles y que haya recompensas suficientes de la seleccionada. En caso de que haya disponibilidad, se actualizan tanto el número de veces que ha sido canjeado esa recompensa como los puntos del usuario, devolviendo la Recompensa actualizada al Front End. En el Front End se sustituye la recompensa modificada en la lista de recompensas obtenidas, generando el objeto Lista de Recompensas que serán mostradas en la interfaz.

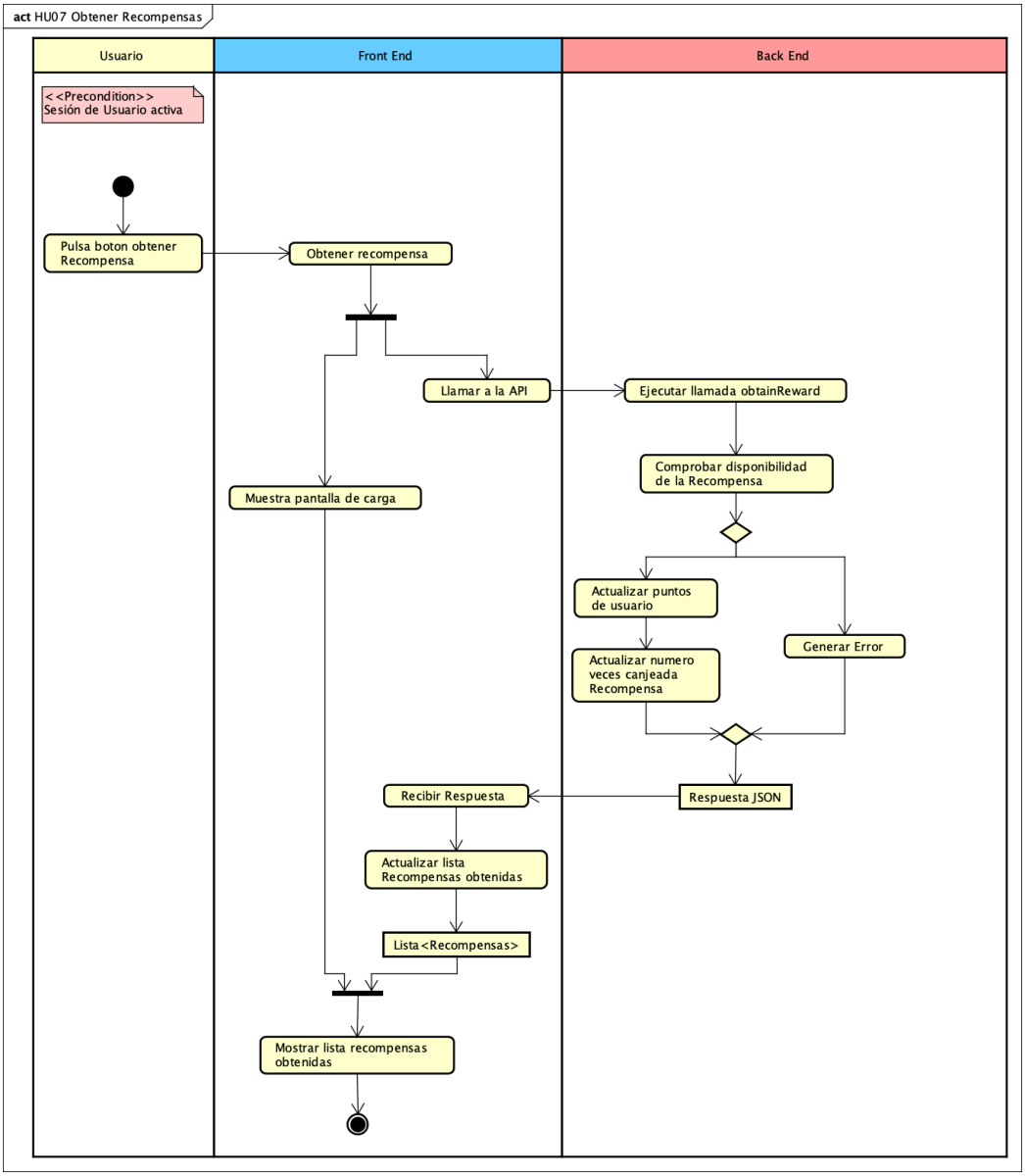


Figura 4.8: Diagrama actividad de historia de usuario Obtener Recompensas HU07

4.1.2.9. HU08 Reservar bicicleta

Para reservar una bicicleta (ver figura 4.9) es necesario haber iniciado sesión (ver figura 4.8) y no tener una ruta activa. Al seleccionar una parada se muestran las bicicletas que hay disponibles y al pulsar sobre ellas se reserva del tipo de bicicleta seleccionado. Tras la llamada a la API se comprueba en el Back End si hay disponibilidad para el tipo de bicicleta seleccionado. En caso de que haya disponibilidad, se genera la reserva asociada al usuario y se devuelve la ruta actualizada con una bicicleta disponible menos, para que no sea desbloqueada por otros usuarios. La reserva expira a los 10 minutos de haberla realizado o si el usuario desbloquea una bicicleta de cualquier tipo en cualquier parada. Esto es debido a que si el usuario al llegar a la parada decide desbloquear una bicicleta de otro tipo, la bicicleta previamente reservada es liberada para que otros usuarios la puedan desbloquear.

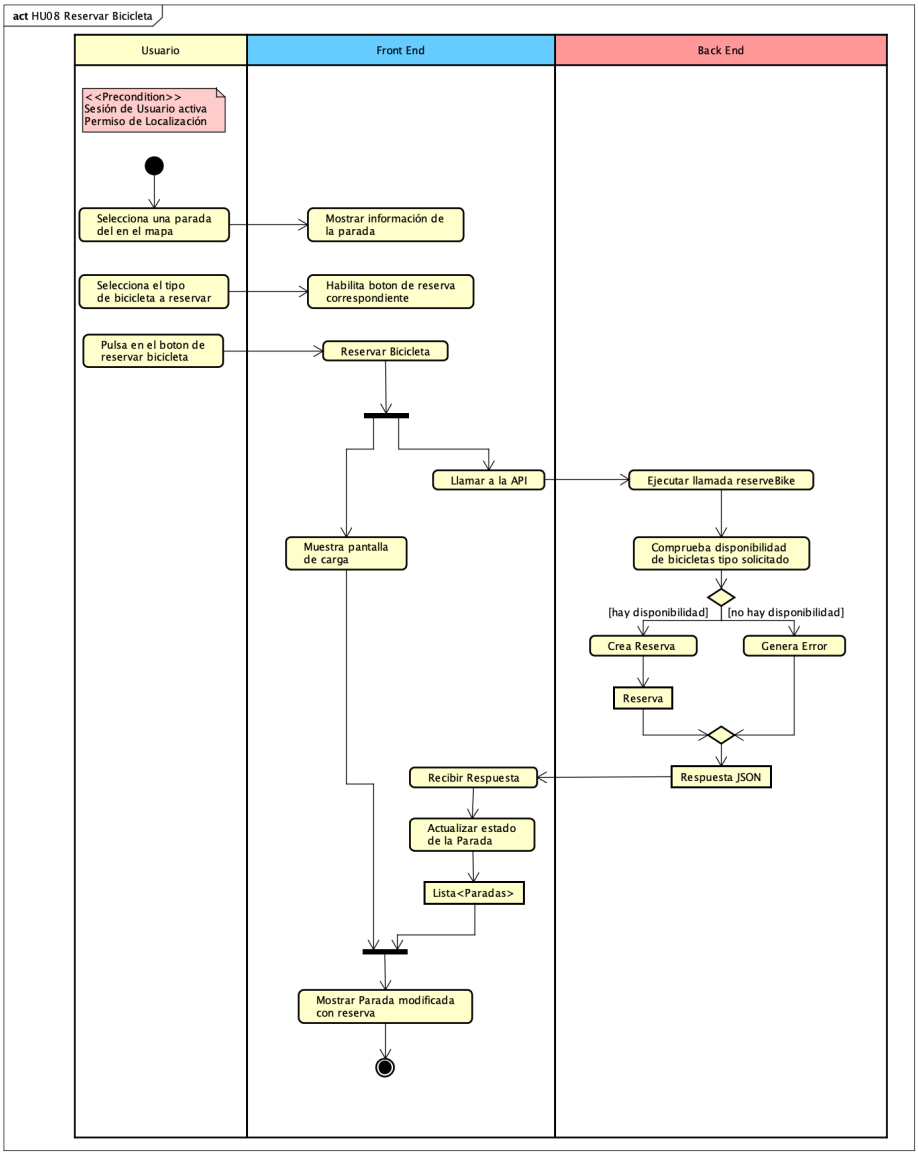


Figura 4.9: Diagrama actividad de historia de usuario Reservar Bicicleta HU08

4.1.3. Modelo de dominio inicial

A partir de los requisitos de información descritos 3.3 se ha desarrollado el modelo de dominio inicial de la aplicación representado en la figura 4.10. El diagrama no representa como es el sistema con respecto a base de datos, sino que representa el sistema de manera general.

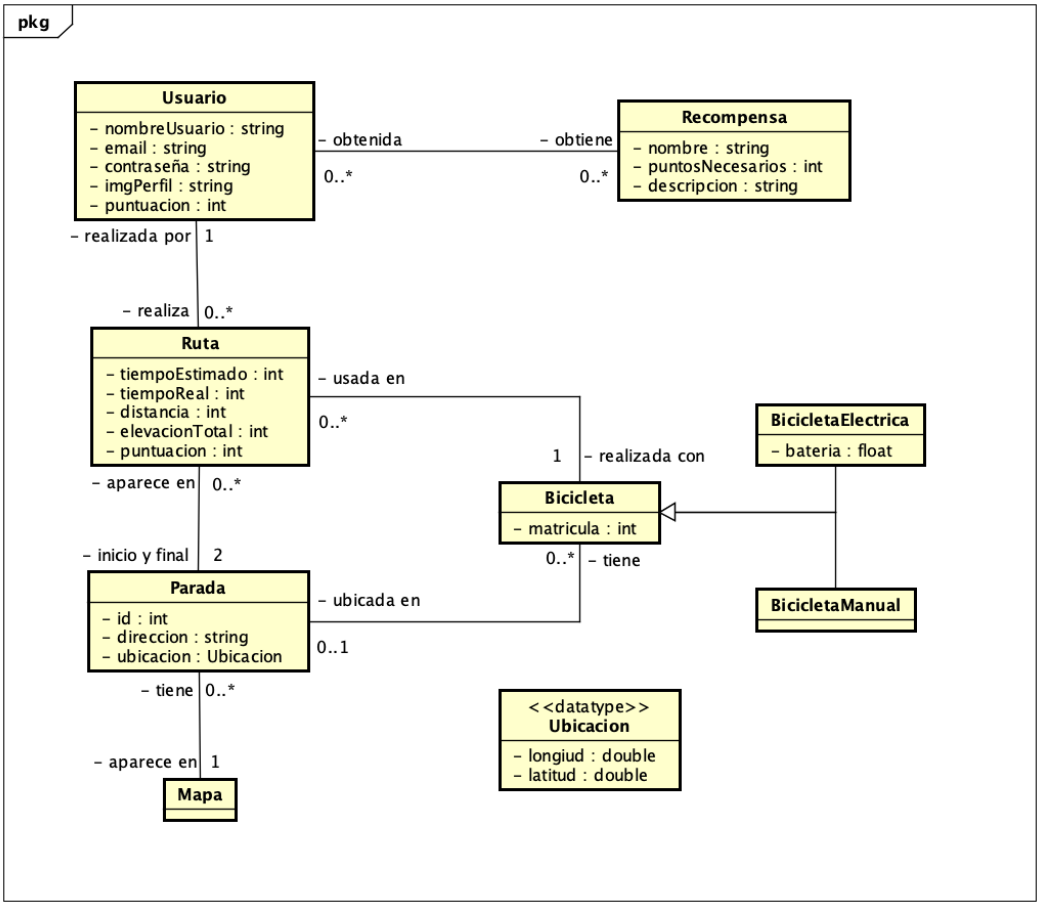


Figura 4.10: Modelo de dominio inicial

Comenzando por una de las partes centrales del sistema, el **Usuario** contiene la información típica de un usuario en cualquier aplicación, a mayores tiene la *puntuación* que va obteniendo al realizar rutas y gastando al canjear recompensas. Los usuarios pueden obtener **Recompensas** si disponen de la puntuación necesaria para ello. Una misma recompensa puede ser obtenida por múltiples usuarios, ya que si no se limitaría demasiado el uso de cada recompensa y no se fomentaría tanto el uso de la aplicación.

Por otro lado, se tienen las entidades relacionadas con la realización de rutas. Los usuarios pueden realizar múltiples **Rutas** que tras finalizarse contiene el tiempo que se ha tardado en realizar la ruta y otros datos geográficos, además se obtiene la puntuación que será sumada al usuario. La ruta comienza y termina en una **Parada**, puede darse el caso de que la parada sea la misma, pero generalmente serán paradas diferentes. Por último, en la ruta se emplea una **Bicicleta**, independientemente de su tipo. Esta misma bicicleta puede ser empleada en múltiples rutas, pero no al mismo tiempo.

En referencia al estacionamiento de las **Bicicletas**, cada bicicleta puede estar estacionada solo en una **Parada** al mismo tiempo, también es posible que la bicicleta no esté estacionada, dado que estaría siendo empleada en una ruta. Por último, en lo referente a las bicicletas, existen dos tipos, **BicicletaManual** que no dispone de motor eléctrico y las **BicicletaElectrica** que si y, por tanto, se registra el estado de su batería.

Por último, hay que destacar dos entidades auxiliares. El **Mapa** es donde se registran las paradas, en principio solo existirá un mapa que contendrá el total de paradas. La otra entidad auxiliar es **Ubicación** que permite representar una localización geográfica usando las coordenadas, esto es empleado para localizar la ubicación de las distintas paradas ubicadas en el mapa.

4.2. Estructura del Proyecto Front End

En la figura 4.11 se puede ver la estructura de paquetes y recursos de la aplicación Android. Se está utilizando la vista de proyecto Android, por lo que no se muestran todas las carpetas intermedias, lo que facilita la visualización de las más relevantes. Como se comentará en la siguiente sección 4.4.1 se ha seguido la guía de Android para escoger una arquitectura, dando como resultado a tres paquetes importantes y otro adicional.

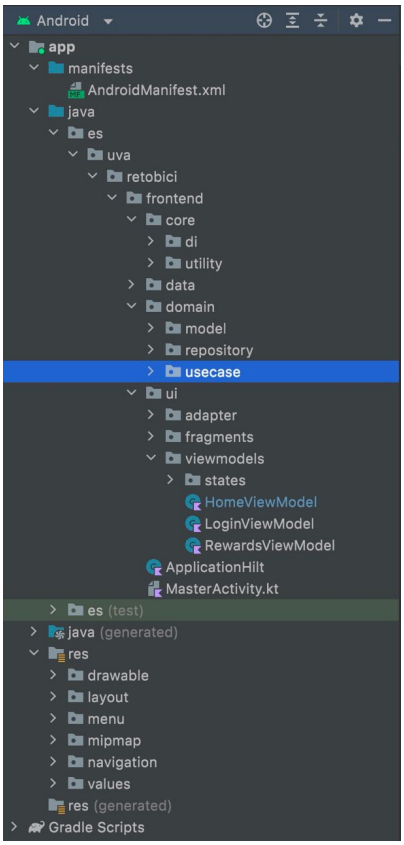


Figura 4.11: Organización de paquetes de la arquitectura del proyecto Android

El paquete **ui** hace referencia a la capa de interfaz de usuario 4.4.1.4. Dentro del paquete **fragments** se encuentran los diferentes fragmentos que representan los **UI elements**. Por otro lado, en el paquete

viewmodels se encuentran, como su propio nombre lo indica, los *ViewModel* que hacen referencia a los **State Holders** y los *states* para representar fácilmente los estados de los modelos. Por último, el paquete *adapter* contiene los elementos necesarios para utilizar un *Recycler View* [85], que se emplea para mostrar las recompensas con una lista dinámica.

El siguiente paquete es el **domain**, que hace referencia a la capa de dominio 4.4.1.5. Dentro del paquete se encuentran tres paquetes, el primero **model** encapsula los modelos, representados mayormente por *data classes* de Kotlin [20]. El segundo paquete más relevante sería el paquete **usecase**, en el que se encuentran los casos de uso. Por último, el paquete **repository**, contiene las interfaces que definen las operaciones de los repositorios, estas interfaces representan el **Repositoy Contract** que se menciona en la figura 4.19.

Como tercer y ultimo paquete relevante en la arquitectura, se encuentra el paquete **data**, que hace referencia a la capa de datos 4.4.1.6. Dentro se encuentran dos paquetes, el primero **repositories** contiene la implementación concreta de las interfaces mencionadas con anterioridad. Estos repositorios pueden ser remotos o locales y existe uno como mínimo por cada modelo en el paquete **model**. El segundo paquete que se encuentra dentro es **source** que contiene otros dos sub-paquetes que hacen referencia al **Data Source** mencionado con anterioridad. El paquete **api** contiene las clases que definen las llamadas a la API que aplican el patrón DAO 4.4.3.1, mientras que el paquete **dto** contiene los DTO o *data transfer object*, explicados más en profundidad en la sección 4.4.3.1.

El paquete **core**, que no hace referencia a la arquitectura. Es un paquete en el que se encuentran diferentes utilidades para la aplicación, como el temporizador para las rutas o como las clases necesarias para utilizar la inyección de dependencias, explicada más en profundidad en 4.4.3.3.

Otros dos paquetes o carpetas relevantes son, la carpeta *manifests* en la que se encuentra el manifiesto donde se declaran los permisos necesarios y las actividades que existen en la aplicación. La otra carpeta sería *res*, en la que se encuentran los recursos de Android, esta contiene varias carpetas con diferentes recursos como, *drawable* contiene iconos e imágenes que se usan en el proyecto, *layout* contiene los ficheros que definen las interfaces de la aplicación, *menu* contiene la definición de los menús que se usan en la aplicación, *mipmap* contiene el icono que se utiliza en el menú de Android, *navigation* contiene los elementos que se usan en el componente de navegación de Android [86] y por último *values* que contiene diferentes recursos como cadenas, dimensiones, estilos y colores.

4.3. Estructura del Proyecto Back End

En la figura 4.12 se puede ver la estructura de carpetas de la parte Back End del proyecto. Laravel aplica automáticamente una estructura de la cual no es necesario modificar más que las clases que contienen la lógica de dominio y las entidades.

El directorio *app* es donde se encuentra gran parte de la lógica de negocio. El directorio *Models* contiene los modelos **Eloquent** de Laravel [87]. Eloquent es un sistema ORM [81] que aparte de gestionar la lógica de los modelos, también facilita la tarea de gestión de los mismos con la base de datos.

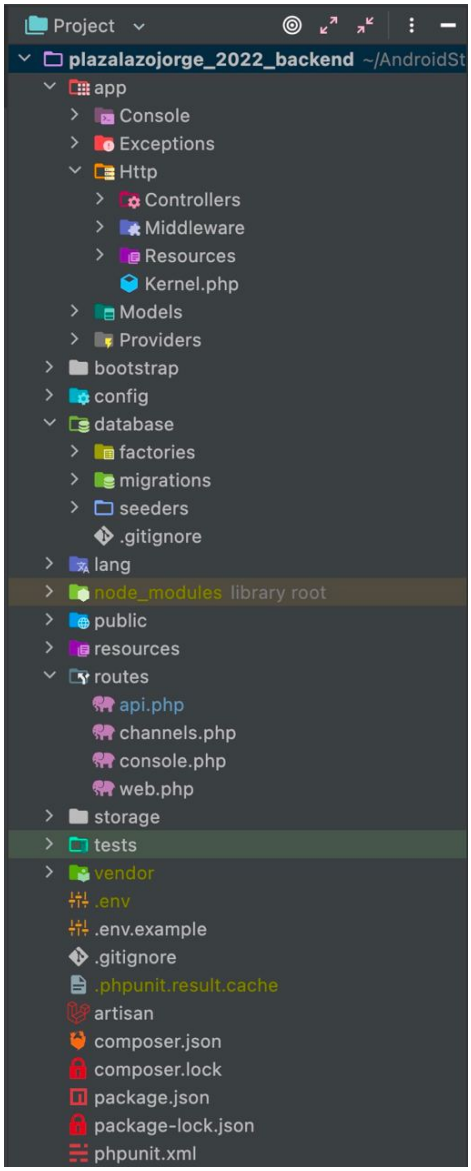


Figura 4.12: Organización de paquetes de la arquitectura del proyecto Laravel

Dentro del directorio *Http* los sub-directorio importantes son *Controllers* en el que, como su nombre lo indica, se encuentran los controladores [88] que son empleados principalmente como contenedores de lógica relacionada con un modelo y que son llamados desde la API. Por último, el directorio *Resources* que contiene los recursos de los modelos que hacen la función de DTO como se comentará en la siguiente sección. También existe el directorio *Middleware*, pero la funcionalidad que contiene no ha sido modificada en este proyecto.

El directorio *database* contiene todo lo relacionado con la base de datos. El subdirectorio *factories* incluye las clases que permiten generar modelos de forma programática. El directorio *migrations* contiene las migraciones de la base de datos que permiten generar la estructura de tablas y por último, el directorio *seeders* contiene las clases que permiten poblar la base de datos con información.

Otro directorio relevante es el *routes*, en este se encuentra el fichero *api.php*, que es donde se definen todas las rutas a la API del proyecto. También hay otros ficheros como *web.php*, pero en este

proyecto como no se tiene una interfaz web no se han empleado.

El resto de directorios no han sido modificados para el proyecto, pero se van a comentar con brevedad. El directorio *bootstrap* contiene los el fichero que agrupa los recursos de la app para ser usado en un navegador. El directorio *config* contiene toda la configuración del proyecto, pero todos estos ficheros, se emplea el contenido del fichero *.env*. El directorio *lang* contiene las traducciones de cadenas en diferentes idiomas que se podría usar en el proyecto. El directorio *node_modules* contiene todos los paquetes JavaScript que pueda usar el proyecto. En el directorio *public* se encuentran los ficheros que son accesibles desde un navegador, como imágenes o *scripts* de JavaScript. El directorio *resources* almacena las diferentes vistas que se utilizarían en la interfaz web, pero que en este proyecto no se ha usado. El directorio *storage* contiene los archivos que se almacenan en el servidor donde se ejecute el proyecto, como imágenes que sean subidas por los usuarios. Por último, el directorio *vendor* contiene los paquetes PHP que se hayan instalado. También comentar algunos de los ficheros relevantes que están en el ámbito del proyecto. El fichero *.env* contiene las variables de entorno que se utilizan para configuración y otras tareas. El ejecutable *artisan* es una herramienta de línea de comandos que se emplea para gestionar el proyecto Laravel, así como desplegar el proyecto como crear los ficheros necesarios del mismo. Por último, los ficheros relacionados con la gestión de paquetes, *composer.json* y *package.json* contienen la lista de paquetes y de versiones que se necesitan instalar para usar el proyecto.

4.4. Diseño

4.4.1. Arquitectura Front End

4.4.1.1. The Clean Architecture

The Clean Architecture [89] (ver figura 4.13) se puede considerar una *meta arquitectura*, es decir, otras arquitecturas se pueden basar en ella y se puede aplicar en múltiples ámbitos. La mayoría de las arquitecturas se utilizan con el objetivo de separar responsabilidades, principalmente dividiendo el software por capas. De esta manera se consiguen sistema que son:

1. Independencia de Frameworks o librerías. De esta manera se pueden utilizar estos Frameworks como herramientas en vez de basar todo el sistema en estos y, por tanto, restringir el sistema.
2. Testeable, la lógica de negocio se puede probar independientemente de elementos externos como la UI, la base de datos o el servidor web.
3. Independencia de UI, se puede intercambiar diferentes interfaces de usuario sin cambiar el resto del sistema.
4. Independencia de la base de datos, al igual que con la interfaz de usuario, la base de datos se puede intercambiar por diferentes implementaciones.

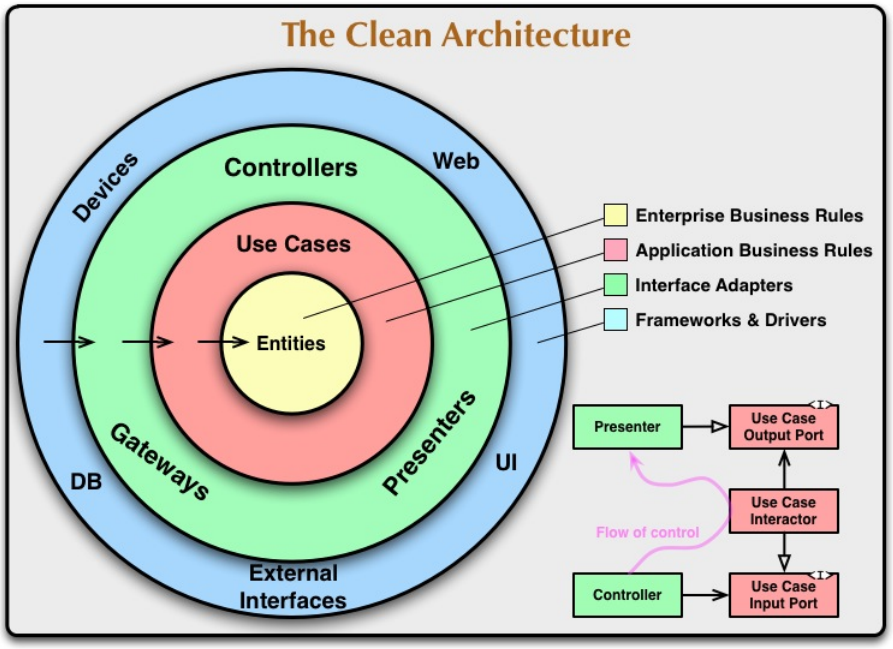


Figura 4.13: Esquema de *Clean arquitecture* dividido por capas [89]

5. Independencia de cualquier agente externo, el sistema o lógica de negocio no conoce nada del "mundo exterior".

Para este proyecto se han seguido las reglas que estableció Robert C. Martin (Uncle Bob)[89].

- **Regla de Dependencia**: las dependencias de código siempre deben apuntar hacia el interior del diagrama, es decir, las capas internas no son conscientes de la implementación del las capas más externas. De esta manera, los círculos externos representan **mecanismos** y los círculos internos **políticas**.
- **Entidades**: son los componentes que encapsulan la lógica de negocio. Por lo que estas no cambian si algo externos al sistema cambia, debido a que estas son independientes y no conocen nada más del sistema por encontrarse en el centro.
- **Casos de uso**: encapsulan las reglas de negocio en las que se aplican las entidades. Los casos de uso dirigen el flujo de datos para conseguir el objetivo de estos. Esta capa solo se vería afectada si los detalles de los casos de uso se vieran modificados.
- **Adaptadores de Interfaz o Controladores**: esta capa se encarga de transformar los datos provenientes de las capas inferiores, casos de uso y entidades, a un formato conveniente para ciertos agentes externos, como una base de datos o una interfaz de usuario.
- **Frameworks**: la última capa está compuesta de herramientas externas que son empleadas como *pegamento* para comunicarse con las capas inferiores. estas herramientas pueden ser implementaciones concretas de bases de datos, servidores web o interfaces de usuario, como por ejemplo las librerías de Android.

Cabe destacar que el número de capas no es fijo y puede variar dependiendo de las necesidades del sistema. Para el desarrollo del proyecto se han aplicado estas reglas, juntos a otras arquitecturas más concretas, dando como resultado la arquitectura que se va a comentar a continuación.

4.4.1.2. Patrón MVVM

MVVM o **Model-View-ViewModel**[90] (ver figura 4.14)es un patrón arquitectónico que aplica los conceptos previamente vistos de *Clean Architecture*, para lograr la separación de responsabilidades. Deriva del patrón MVP o Modelo-Vista-Presenter [91] que este a su vez deriva del conocido MVC o Modelo-Vista-Controlador[24].

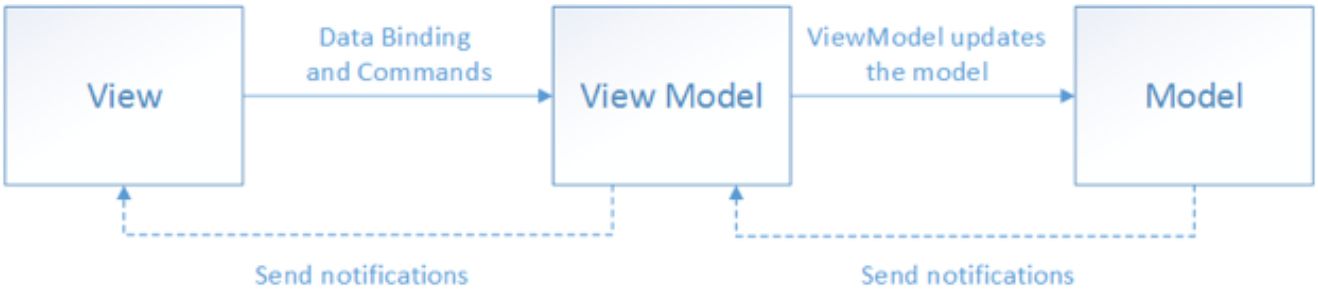


Figura 4.14: Esquema del patrón MVVM [92]

El patrón presenta tres partes diferenciadas:

- **Model**. Representa la lógica de negocio del sistema, en términos de Clean Architecture sería la capa de *Entidades*.
- **View** o vista, se encarga de mostrar la información al usuario a la vez que recibe *inputs* o entradas del usuario de manera que se ejecuta la lógica de la interfaz de usuario.
- **ViewModel**. Es un actor o controlador intermedio entre el Modelo y la Vista. Su principal función en notificar de los cambios de estado sobre los datos a la interfaz a la que esté conectada, para que de esta manera la vista decida si tiene que modificarse o no.

Como se puede ver en la figura 4.14 la jerarquía sigue las reglas de *Clean Architecture* las clases superiores interactúan con las inferiores, pero no en el otro sentido. Las clases inferiores notifican a las superiores de sus cambios, pero esto no quiere decir que dependan de ellas, a estas comunicaciones se les llama *data bindings* o enlace de datos.

La aplicación de este patrón arquitectónico se ve reflejada en la sección 4.4.1.4

4.4.1.3. Arquitectura de Android de tres capas

Se han seguido la guía para una aplicación Android recomendada por Google [93]. Se recomienda dividir la aplicación en tres capas (ver figura 4.15)

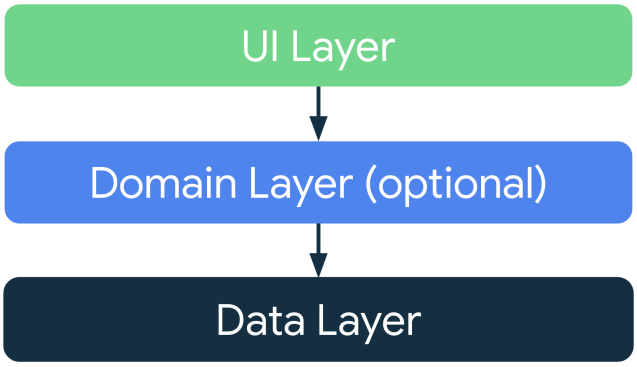


Figura 4.15: Esquema general arquitectura por capas de Android [93]

4.4.1.4. Capa de Interfaz de Usuario

La función de esta capa es mostrar la información en la pantalla a la vez que sirve de punto de entrada principal para la interacción del usuario [94].

Esta capa contiene dos componentes, como se puede ver en la figura 4.16, la interfaz o **elementos UI**, como pueden ser los botones de Android o el sistema de Mapas y los **State Holders** que son clases que se encargan de mantener y proveer a la interfaz de usuario de su estado, en este caso se han empleado la librería de Android *ViewModel* [95].

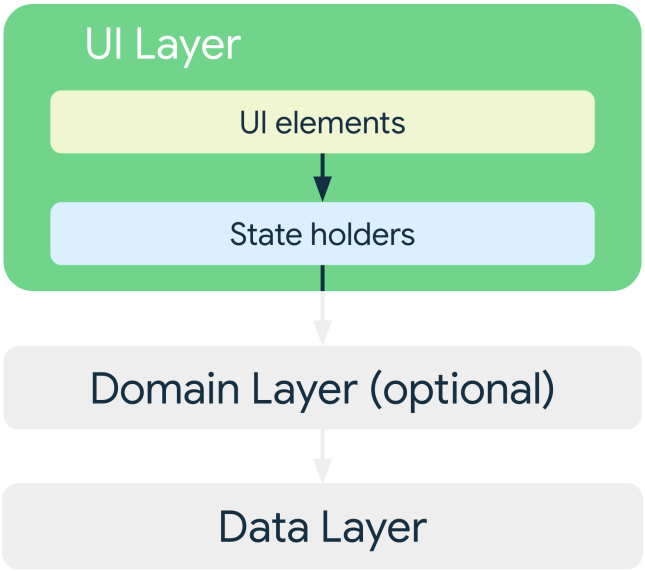


Figura 4.16: Esquema de la capa de Interfaz de Usuario [94]

Aquí es donde se aplica el patrón **MVVM**, en concreto la parte de *View* como *elementos UI* y de *ViewModel* como *State Holders*. La parte del *Model* se ha decidido encapsular en la capa de dominio, descrita en profundidad en la sección 4.4.1.5.

En cuanto a la **View** se ha decidido emplear también la **single-activity architecture** [96], recomendada por *Google*. Esta consiste en que la aplicación Android y en concreto la Vista o View está representada por una única *Activity* o actividad [97], que sirve como contenedor de *Fragments* o frag-

mentos [98] que muestran las diferentes pantallas. Las ventajas de esta son mayormente internas de como se comporta el ciclo de vida de las actividades y fragmentos y la manera en la que se comunican. Además, permite integrar herramientas o componentes como *Navigation*[86]

En cuanto al **ViewModel** se han empleado la implementación de la clase homónima *ViewModel* [95], diseñada para almacenar y administrar los datos relacionados con una actividad o fragmento, de una manera consciente del ciclo de vida de estos elementos en una aplicación Android.

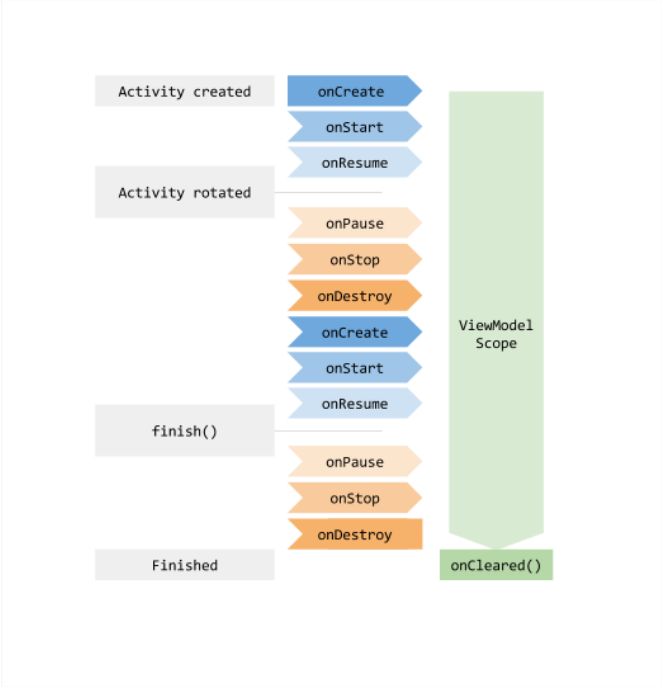


Figura 4.17: Ciclo de vida de un ViewModel [95]

Como se observa en la figura 4.17 el *ViewModel* mantiene su estado desde la creación de la actividad a la que está asignada hasta la destrucción de la misma. De esta manera se mantiene el estado a pesar de que se cambie o pause la actividad. Para los fragmentos ocurre de la misma manera.

Respecto a los *data binding* o enlace de datos, se han empleado las clases **LiveData** y **Observer** [80] que ayudan a mantener los datos en el *ViewModel* a la vez que en la *View* se observan los cambios para poder decidir si se debe modificar o no la interfaz.

4.4.1.5. Capa de Dominio

Según la guía de arquitectura de Android [93], esta capa es opcional y solo es recomendada para encapsular lógica de negocio más compleja de una manera que sea más fácil de reutilizar [99]. Esta capa sería la análoga a los **casos de uso** de *Clean Architecture*.

En esta capa se encuentran los casos de uso relacionados con la aplicación Android, como por ejemplo obtener las diferentes paradas para mostrar su información.

4.4.1.6. Capa de Datos

Esta capa encapsula la lógica de negocio [100]. Tiene dos componentes principales, los **Repositories** o *repositorios* y las **Data Sources** o *fuentes de datos*. Cada Entidad del sistema tiene una clase repositorio encargada de, exponer los datos de la entidad, centralizar los cambios de dichos datos y abstraer a la aplicación de donde provienen los datos. Por otro lado, las fuentes de datos deben obtener los datos siempre del mismo lugar, ya sea de una base de datos local, una API remota o un fichero.

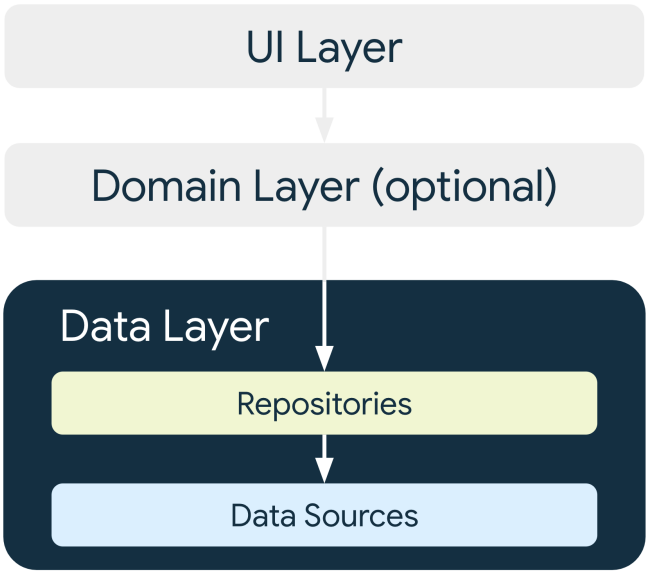


Figura 4.18: Esquema de la capa de datos [100]

Con la separación de repositorios y fuentes de datos se consigue que los consumidores de los datos no conozcan si los datos provienen de local o de remoto.

Por último, hay que destacar que en la capa de dominio se han agregado un *Repository Contract*, en este caso en forma de interfaz, de esta manera la implementación del repositorio es definida en la capa de datos y, por tanto, se desacopla el código. En la figura 4.19 se ve reflejado el flujo de dependencias.

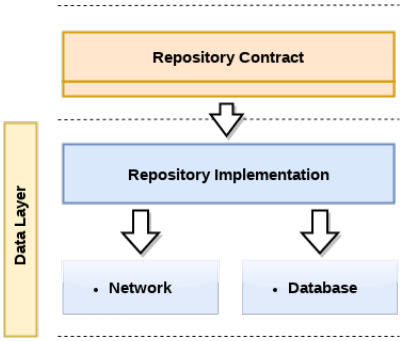


Figura 4.19: Esquema de uso de repositorios [101]

4.4.1.7. Resumen de la arquitectura

El siguiente diagrama 4.20 representa como está organizado el proyecto y sus diferentes capas. Debido a que solo se utiliza una API remota como única fuente de datos, no se utiliza una fuente de datos local.

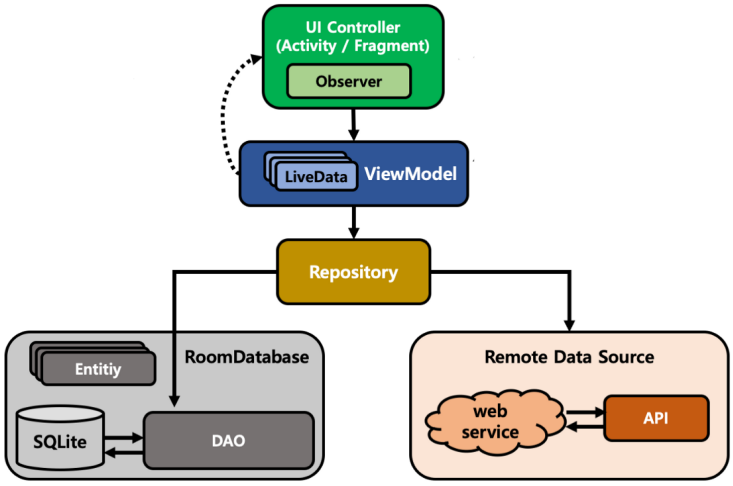


Figura 4.20: Esquema arquitectura general Android [102]

4.4.2. Arquitectura Back End

4.4.2.1. Patrón MVC

Laravel utiliza por defecto la arquitectura **MVC** o Modelo Vista Controlador [24] (ver figura 4.21). Este patrón arquitectónico está dividido en tres partes como su nombre lo indica.

- El **Modelo** encapsula las entidades que representan la información en el sistema. Sin embargo, los modelos o entidades no conocen ni deben conocer ninguna lógica de presentación o de interfaz de usuario.

- La **Vista** se encarga de mostrar la información de los modelos al usuario, pero no interviene en como se modifican o se emplean estos modelos.

- El **Controlador** es el elemento que se ubica entre los dos anteriores, comunicando las otras dos capas. Es el encargado de indicar como se tienen que modificar los modelos dependiendo de las entradas recibidas en la vista y viceversa.

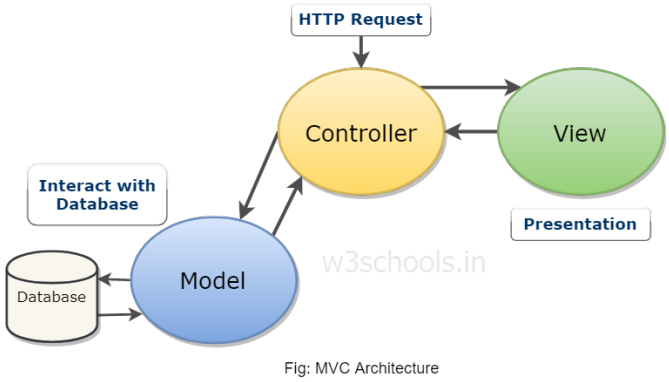


Figura 4.21: Esquema arquitectura MVC [103]

4.4.3. Patrones de Diseño

4.4.3.1. Patrón DAO/DTO

El patrón DAO/DTO hace referencia a dos patrones que se suele utilizar en conjunto.

Por un lado, se tiene el patrón **DAO** o *Data Access Object* [104]. Es un patrón en el que se tiene una interfaz abstracta para separar la lógica de negocio del acceso a datos. Se declara una interfaz DAO para cada Modelo, que proporciona métodos de acceso a la fuente de datos sin exponer los detalles de la misma fuente, pudiendo ser una base de datos o un repositorio remoto.

Por el otro lado se tiene el patrón **DTO** o *Data Transfer Object* [105]. Aplicando este patrón se tiene un objeto que se encarga de almacenar y serializar en ciertos casos la información que se va a emplear en procesos de comunicación, como puede ser enviar o recibir un objeto de un servicio web.

La combinación de ambos se ve reflejada en la siguiente figura 4.22.

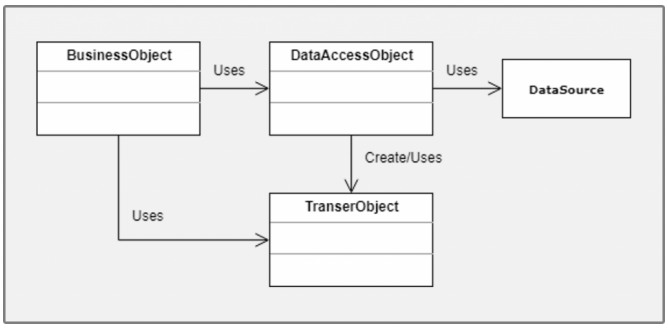


Figura 4.22: Patrón DAO DTO [106]

En el proyecto se ha empleado tanto en el Front End como en el Back End. Del lado del Front End se han utilizado los dos patrones, el DAO son las interfaces que se encuentran dentro del paquete **data.source.api** que define los métodos de acceso para la obtención de datos del Back End. Y los DTO son *data classes* [20] ubicadas en el paquete **data.source.dto** que deserializan los objetos obtenidos del Back End y los trasforman en Modelos, para que puedan ser usados en el Front End, para ello se utiliza la librería *GSON* de Google [107]. Del lado del Back End se emplean únicamente los DTO

para serializar los modelos y poder ser enviados como respuesta al Front End, para ello se utilizan los *Eloquent API Resources* de Laravel. [108].

4.4.3.2. Patrón Observador

El patrón **Observador** [109] (ver figura 4.23) consiste en un **subject** o sujeto que mantiene una lista de **observadores** que dependen de él y son notificados de cualquier cambio en el estado del sujeto. Suele ser usado para sistemas basados en eventos o sistemas distribuidos.

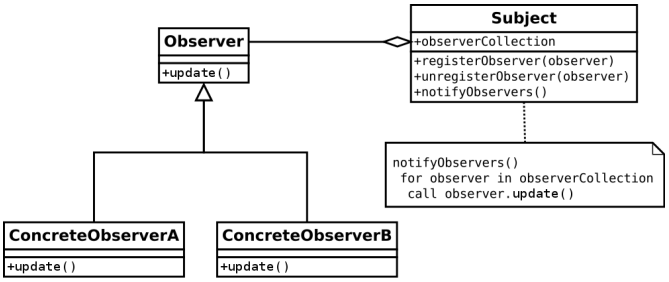


Figura 4.23: Patrón Observador [109]

En el proyecto se ha empleado en la parte del Front End con el uso de las clases **LiveData** de Android [80]. Los objetos LiveData almacenan el estado de ciertos objetos, como una Ruta. Están ubicados principalmente en los *ViewModel*, y son observados principalmente desde los fragmentos o actividades de la interfaz de usuario, de esta manera cuando se modifica el estado de uno de estos objetos, por ejemplo finalizando una ruta, el objeto LiveData notifica a todos sus *observers* activos que se ha actualizado y estos *observers* realizan su lógica de interfaz de usuario que suele ser modificar o actualizar elementos gráficos para mostrarlos al usuario.

4.4.3.3. Inyección de Dependencias

La Inyección de dependencias es un patrón de diseño que consiste en que si un objeto depende de otro, este recibe el objeto del que depende, en vez de tener que construirlo él. Se emplea principalmente para la separación de responsabilidades, garantizando que si un objeto quiere utilizar otro objeto o servicio, no tenga que conocer como se construye dicho servicio. Esto da lugar a código más desacoplado o independiente entre módulos.

En el proyecto se ha empleado la inyección de dependencias en el Front End, se ha empleado la librería **Dagger Hilt** recomendada por Android [110]. Hilt proporciona una forma de aplicar inyección de dependencias sin tener que hacerla de forma manual, proporcionando contenedores para cada clase de Android del proyecto, además de manejar automáticamente los ciclos de vida de Android. Por otro lado, Dagger genera las dependencias a tiempo de compilación, lo que otorga beneficios como detección de errores en tiempo de compilación, mejor rendimiento de ejecución y mayor escalabilidad del código. Para usar Dagger Hilt hay que crear una clase con la anotación **@HiltAndroidApp** que sirva como contenedor de dependencias en el ámbito de aplicación. Además, para cada fragmento o

actividad que necesite inyectar dependencias hay que agregar la anotación **@AndroidEntryPoint**. Con estas dos configuraciones ya es posible inyectar las clases del proyecto en cualquier otro punto del mismo, pero también es necesario inyectar código de librerías externas al proyecto, para ello se definen módulos en el paquete **core.di** que construyen los servicios externos, para que posteriormente puedan ser inyectados.

4.5. Diagramas de paquetes y clases

En esta sección se van a mostrar y comentar los diferentes diagramas de paquetes y la organización de ambas partes del proyecto.

En la figura 4.24 se puede observar la estructura de paquetes del Front End ya comentada con anterioridad. Se puede ver que sigue la regla de dependencia de *Clean Architecture* en la que las capas superiores son las que acceden a las inferiores y no en el otro sentido. También se puede ver como todas las capas acceden al paquete *core*, este contiene funcionalidades comunes, como la organización de módulos para la inyección de dependencias.

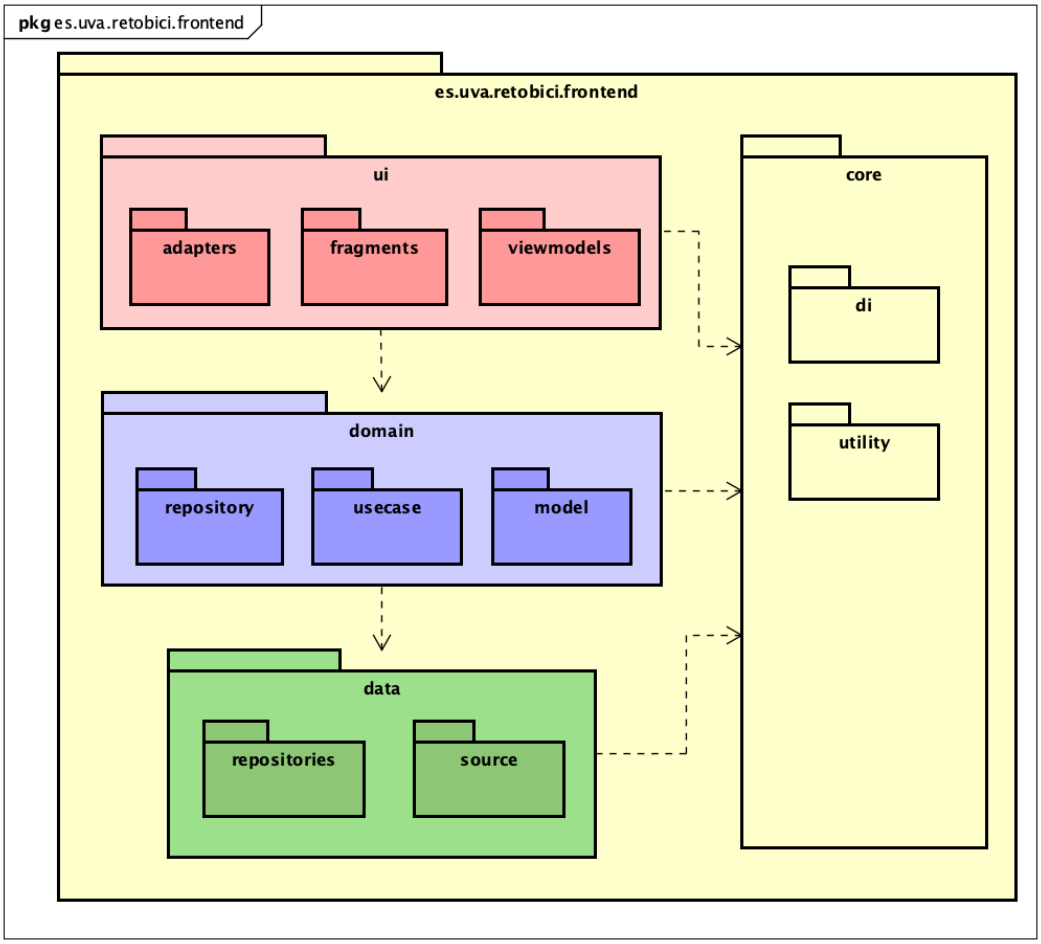


Figura 4.24: Diagrama de paquetes en el ámbito del proyecto Front End

En la figura 4.25 se puede ver como se organizan los paquetes y las clases encargados de la interfaz de usuario, aquí es donde se aplica el patrón MVVM. Varias cosas a detallar sobre este diagrama son la clase *MasterActivity* que es dependencia de todos los fragmentos, esto se debe a que esta actividad sirve de contenedor de los fragmentos, siendo estos capaces de modificar ciertos aspectos de la actividad. Una de las cosas más relevantes es la relación entre los fragmentos y los *ViewModel*, los fragmentos de *login* y de *recompensas* tienen su propio *viewmodel*, por otro lado, los fragmentos de *home*, *escáner de QR* y *resumen de ruta* comparten el mismo *ViewModel* ya que necesitan compartir información entre ellos, como que bicicleta se ha desbloqueado o la ruta en curso. Las clases dentro del paquete *states* representan los diferentes estados de entidades como la ruta, de esta manera es más sencillo manejar la interfaz utilizando las *sealed classes* [19] junto a la expresión *when*. Por último el paquete *adapter* contiene las clases necesarias para mostrar la lista de recompensas, el componente *RecyclerView* [85] se emplea para ello, siendo necesario el uso de las clases *ViewHolder* y *Adapter*, con la implementación de estos componentes se obtiene una lista de elementos dinámica y eficiente para el sistema Android.

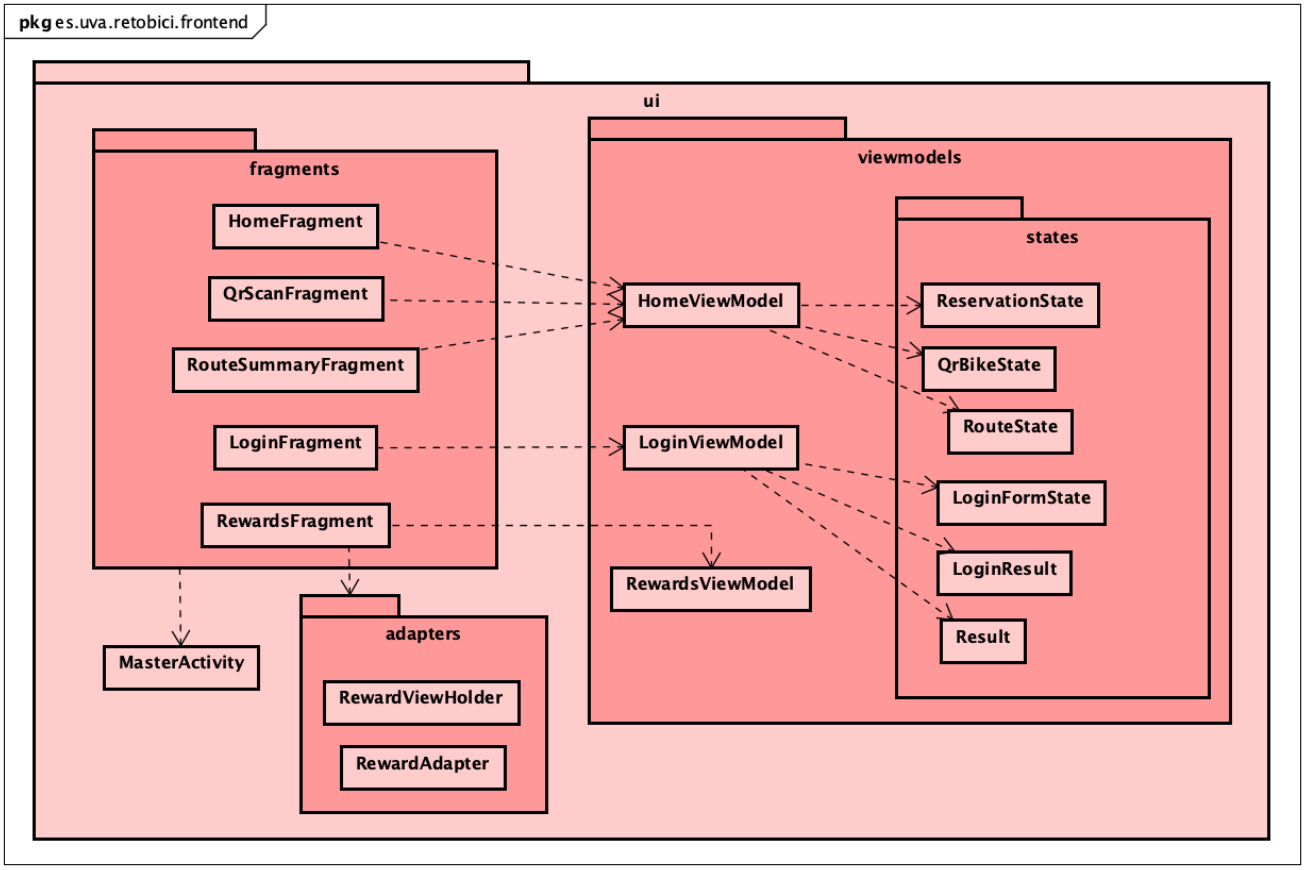


Figura 4.25: Diagrama de paquetes y clases ui

En la figura 4.26 se puede ver como está organizado el paquete de dominio, en este se encuentran los casos de uso que encapsulan el trabajo necesario en el ámbito de reglas de dominio, además de emplear los repositorios para acceder a las diferentes fuentes de datos sin importar si están implementadas en remoto o en local.

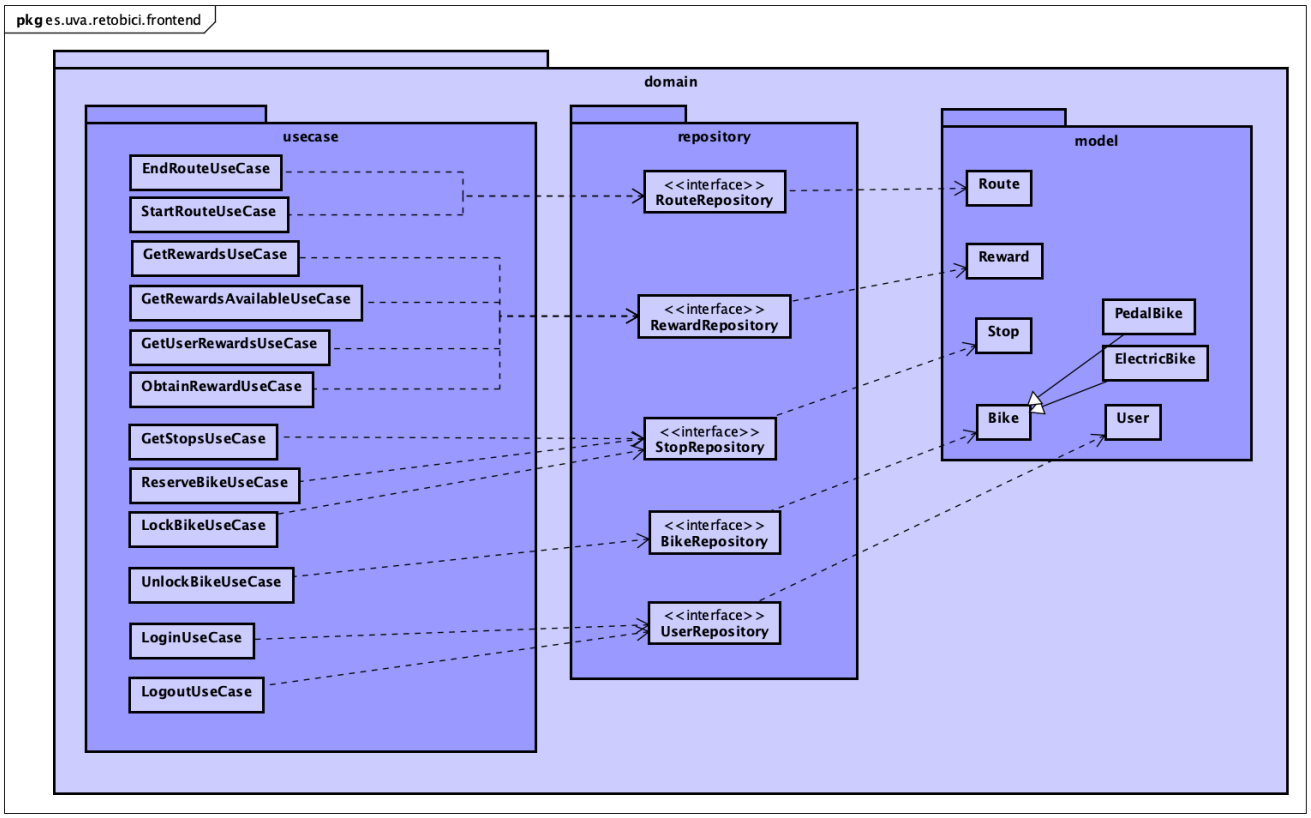


Figura 4.26: Diagrama de paquetes y clases domain

En la figura 4.27 se puede ver como las fuentes de datos remotas emplean las interfaces API para comunicarse con el BackEnd, las interfaces API como se ha comentado previamente serían los DAO que obtienen la información del Back End realizando llamadas con *Retrofit* [111], una librería que facilita realizar llamadas a APIs Rest. Las respuestas del Back End son obtenidas en formato JSON y transformadas con las clases DTO con la librería Gson [107] para obtener los modelos de dominio. Por otro lado, se puede ver la fuente de datos local para el usuario, empleando la clase *UserPreferences* con la que se almacena la sesión y el token de un usuario cuando hace login. Esta clase emplea la librería *DataStore* de Android [112].

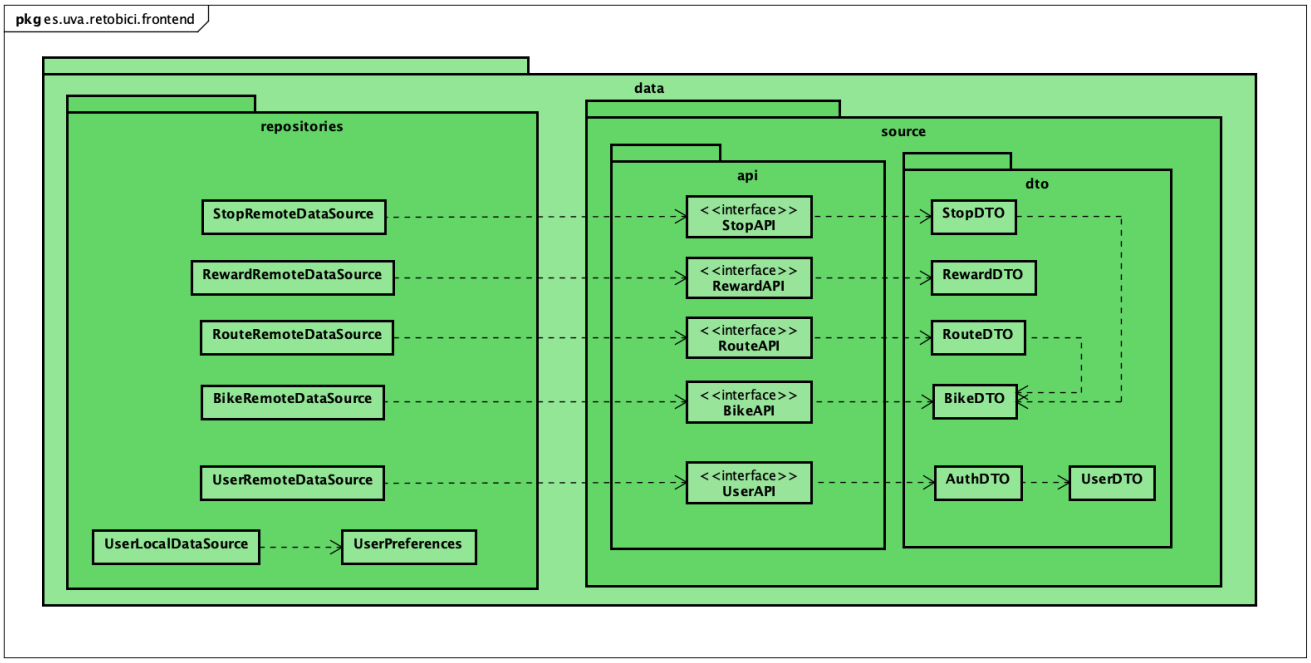


Figura 4.27: Diagrama de paquetes y clases data

En la figura 4.28 se pueden ver las clases que son compartidas por todo el sistema, en el paquete *utility* se encuentran las constantes empleadas en el proyecto y el temporizador empleado en el transcurso de la ruta. Por otro lado, en el paquete *di* están las clases que configuran los módulos de librerías externas que necesiten ser inyectados en clases del sistema.

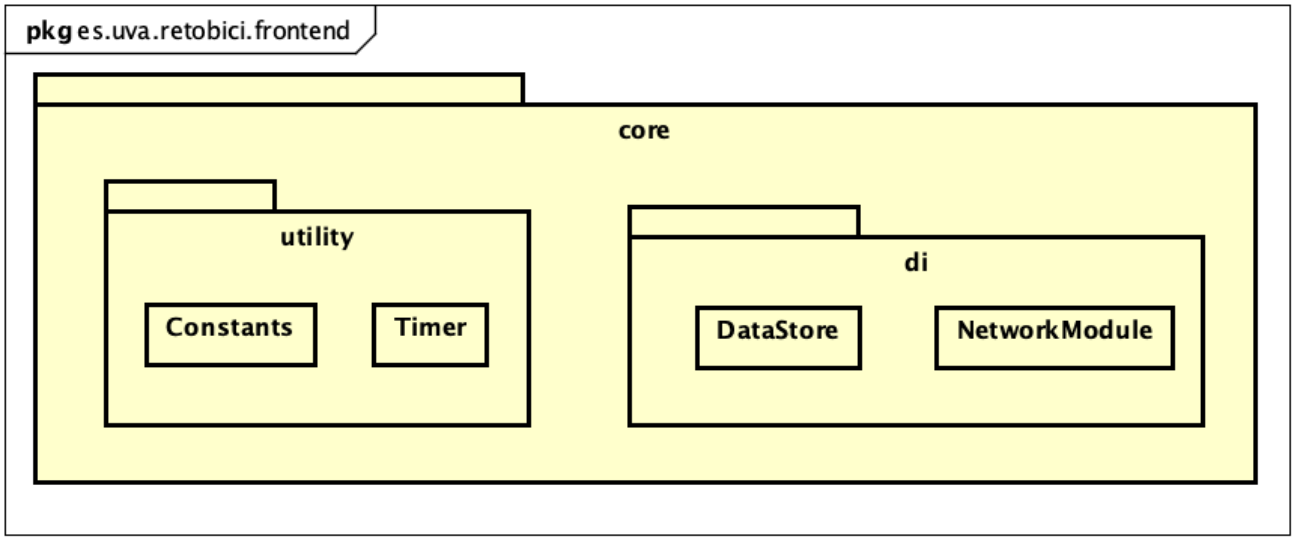


Figura 4.28: Diagrama de paquetes y clases core

En la figura 4.29 se puede ver los paquetes más importantes de la parte del Back End, esta sigue el patrón de arquitectura MVC, pero sin el uso de la vista, ya que eso es manejado por el Front End. Laravel tiene de forma predeterminada una forma de organizar los paquetes, se han excluido del diagrama todos los paquetes que han sido creados por Laravel, mostrando solo a los que se ha agregado clases durante el desarrollo. Con respecto al paquete *routes* contiene varias clases que gestionan las rutas tanto para la interfaz web como para la API, en este proyecto solo se ha empleado la clase para configurar las rutas de la API.

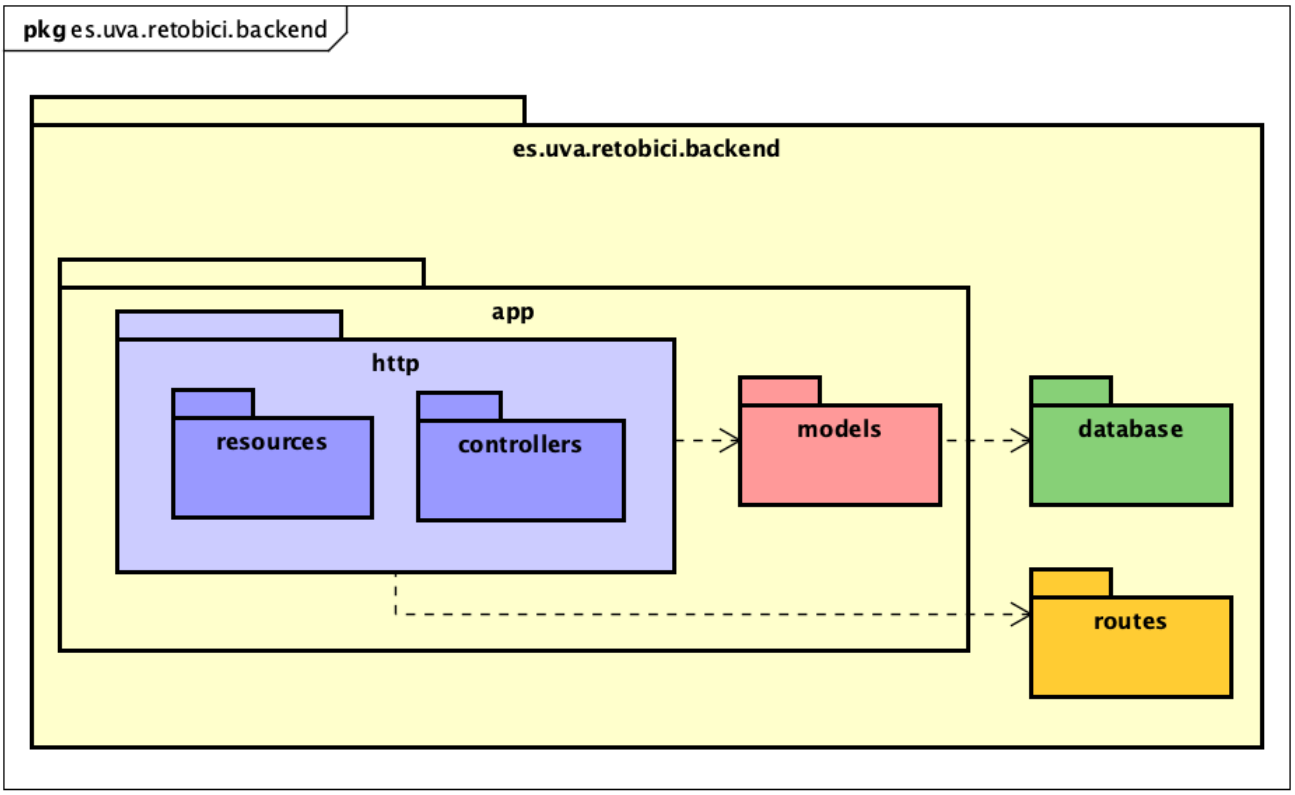


Figura 4.29: Diagrama de paquetes en el ámbito del proyecto Back End

En la figura 4.30 se puede ver la organización del paquete *http*, compuesto por el paquete de controladores que manejan las reglas de negocio, esto es, manejar los modelos y como deben comportarse, en esencia como se comporta la aplicación, obteniendo, almacenando o modificando los modelos. Por otro lado, el paquete de *resources* actúa como DTO para transferir las respuestas de las llamadas a la API transformando los modelos a objetos JSON. Es necesario comentar dos clases del paquete de controladores. Primero, el controlador de usuario es usado por otros controladores que necesiten manejar lógica relacionada con un usuario que haya iniciado sesión, como gestionar una ruta iniciándola o finalizándola. También es necesario comentar la clase *MapBoxApiClient*, es la encargada de hacer las peticiones a la API de Mapbox Navigation [32] para obtener el recorrido de la ruta cuando es finalizada proporcionando una serie de coordenadas, en este caso las coordenadas de la parada de inicio y la de parada de fin.

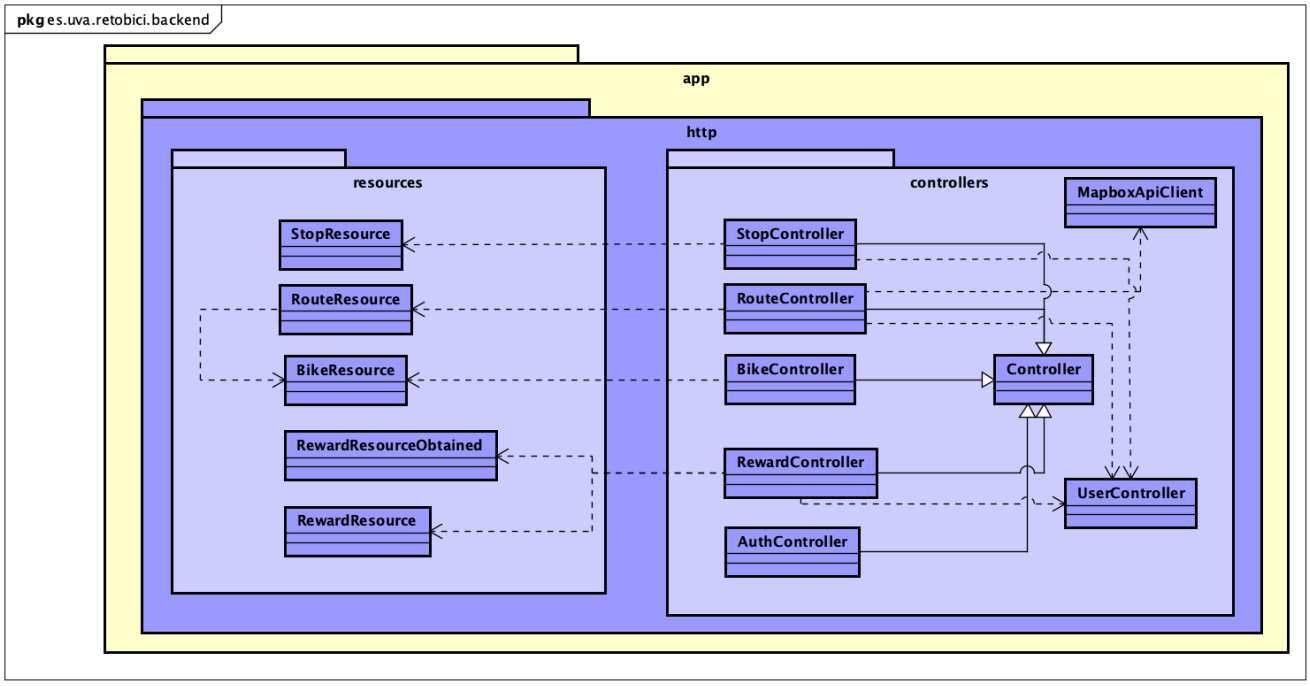


Figura 4.30: Diagrama de paquetes y clases http

En la figura 4.31 se pueden ver los diferentes modelos que son empleados en el Back End. A diferencia del Front End solo hay dos clases para las bicicletas, esto se debe a como se organiza la herencia en Laravel y PHP y como se almacena en la base de datos.

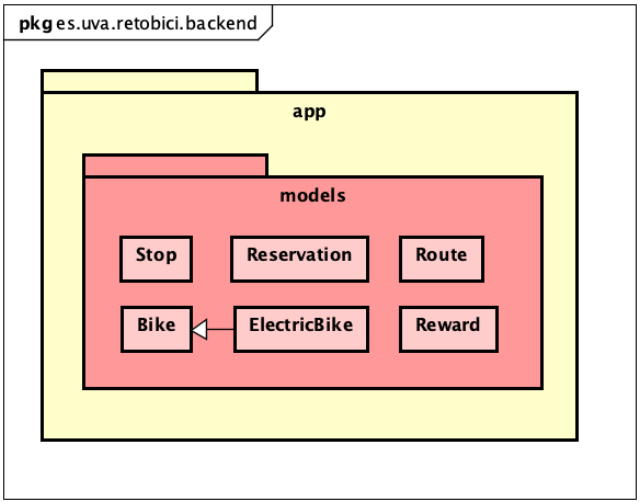


Figura 4.31: Diagrama de paquetes y clases models

En la figura 4.32 se puede ver algunas de las clases que se encuentran en el paquete *database*. Los *seeders* son clases que rellenan la base de datos con información de una manera sencilla y automática, funcionan de una forma similar a un *script* SQL que inserta datos en la base de datos. El paquete *migrations* contiene las migraciones de la base de datos, estos son ficheros que generan la estructura de la base, definida en código, abstrayendo de esta manera del lenguaje SQL. No se muestran las clases porque se considera que no aportan información.

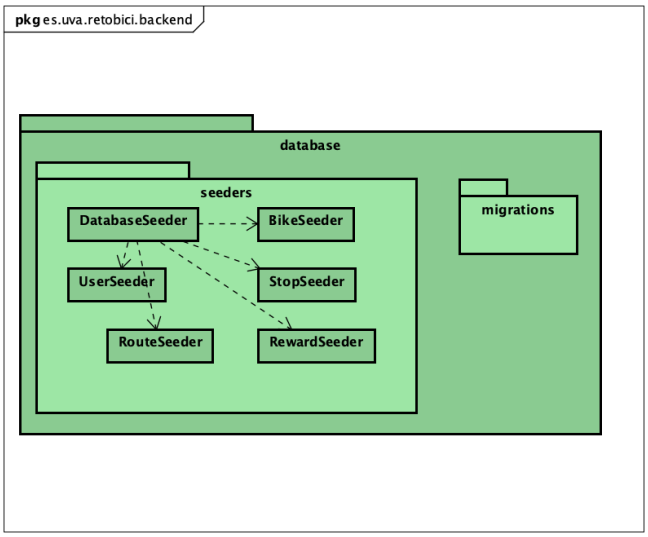


Figura 4.32: Diagrama de paquetes y clases database

4.6. Diagrama de Base de datos

En la figura 4.33 se puede ver el esquema de la base de datos. El esquema es generado con las migraciones del paquete *database* (ver figura 4.32). Cada tabla de la base de datos tiene relacionado un modelo del paquete *models* (ver figura 4.31). Para representar la herencia de los modelos de bicicleta se ha decidido utilizar una tabla base que contiene la información genérica de una bicicleta y el identificador en otra tabla de una bicicleta en el caso de que sea eléctrica. La tabla referente a la bicicleta eléctrica solo contiene la información adicional, como el nivel de batería. Con respecto a las tablas de usuario y recompensa, se tiene una relación muchos a muchos, Laravel genera una tabla intermedia que almacena las recompensas que han sido obtenidas y por qué usuario.

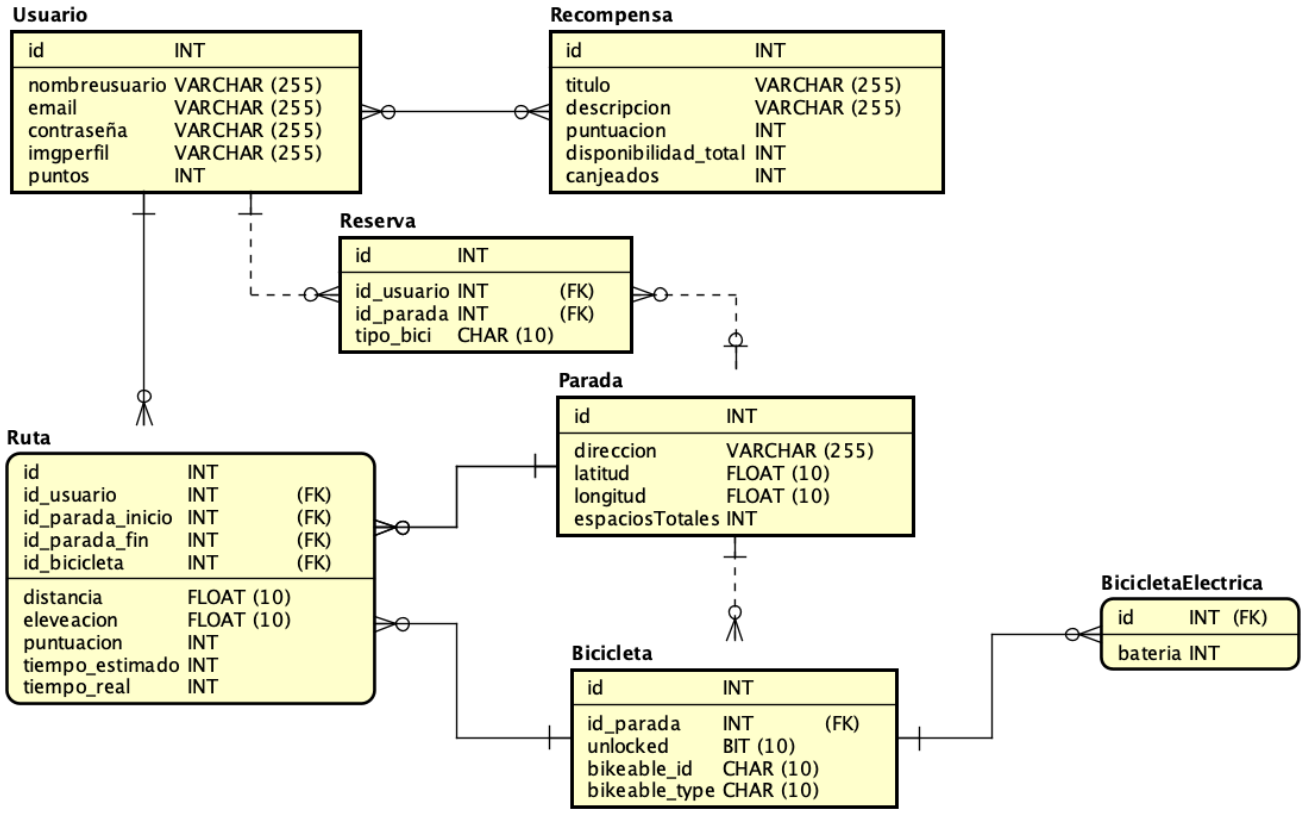


Figura 4.33: Diagrama Entidad Relación de la base de datos

4.7. Diagrama de Despliegue

En la figura 4.34 se puede ver el diagrama de despliegue. En este caso se utiliza el portátil como servidor que aloja el sistema Back End. En el servidor se ha instalado el sistema de ejecución PHP y la base de datos PostgreSQL, que son el conjunto de herramientas necesarias para ejecutar una aplicación Laravel. El proyecto Laravel gestiona la conexión con la base de datos y con el servidor de forma automática, configurando los parámetros del proyecto necesarios. El Back End expone a la red la API, que puede ser accedida a través de la URL que se configure en la red local. El Front End es compuesto por cualquier dispositivo móvil que instale la aplicación. La aplicación establecerá conexiones HTTP con la API del Back End a través de las cuales puede realizar las tareas que han sido programadas.

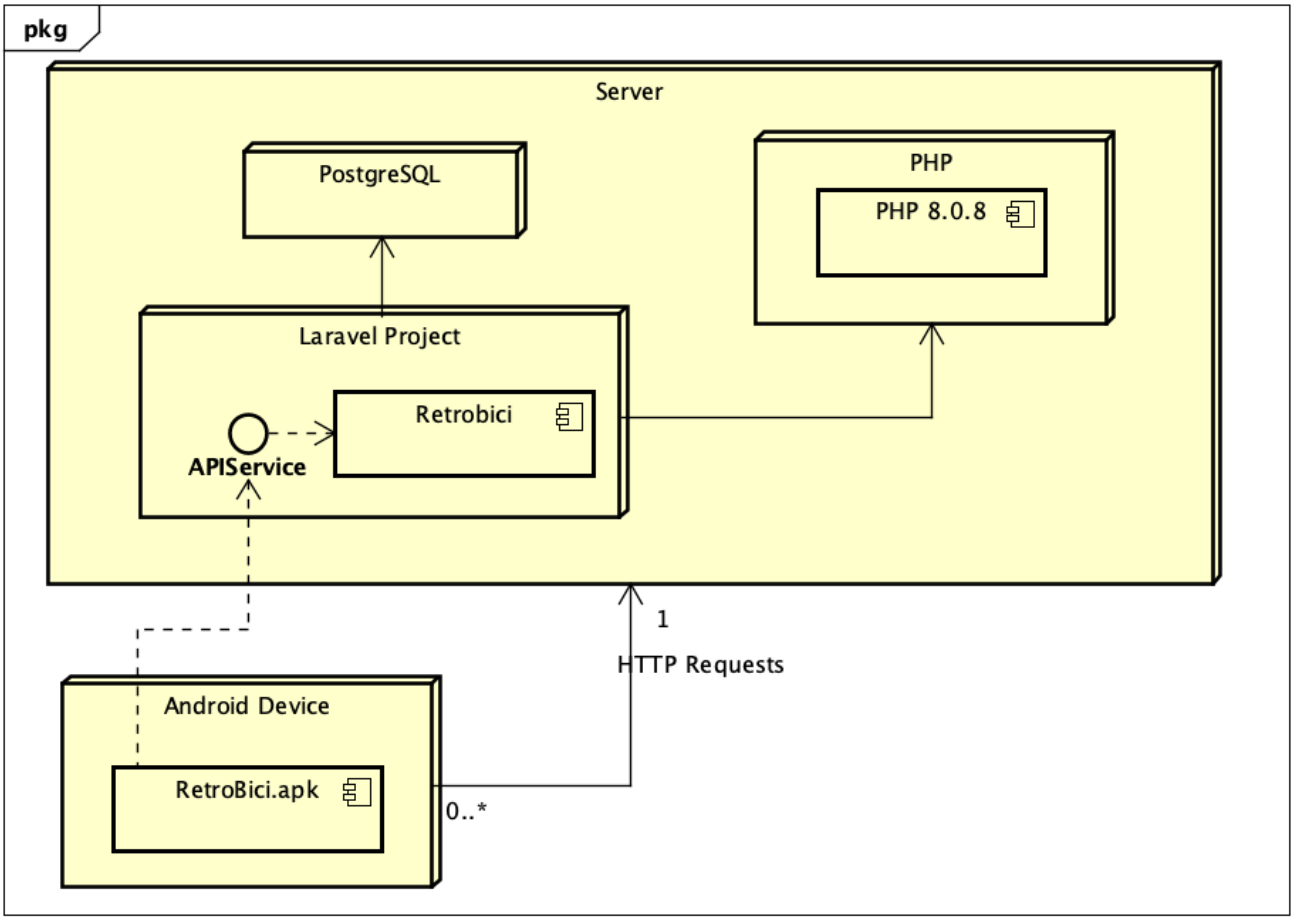


Figura 4.34: Diagrama de despliegue del sistema

Capítulo 5

Pruebas

A lo de este capítulo se comentarán las diferentes formas y opciones para probar ambas partes del proyecto. Debido a la falta de tiempo, se ha optado por hacer un estudio de como se harían los diferentes tipos de pruebas junto a una implementación de cada uno de ellos. Probar una aplicación completamente es una tarea muy extensa en tiempo y recursos, podría considerarse otro proyecto aparte, existen diferentes maneras de probar una aplicación, desde probar cada parte manualmente emulando los diferentes casos de uso y posibles resultado, pero esta manera de probar aplicaciones *manualmente* no es sostenible y escalan mal en términos de tiempo y esfuerzo. Una mejor alternativa son las pruebas automáticas, en las que se utilizan otras tecnologías que realizan los test por ti, algunas de estas herramientas pueden ser *phpunit* [113] o *junit* [114].

Una forma de hacer que un código sea más fácil de probar es escoger una asignatura adecuada para el proyecto, en este caso *Clean architecture* [89], como se ha comentado con anterioridad, facilita las pruebas del código, ya que se separan las responsabilidades.

5.1. Pruebas Front End

Una aplicación Android es un sistema muy complejo que interactúa en múltiples contextos y ecosistemas. Para poder probar cada parte de una aplicación Android existen varios tipos de pruebas [115]. Como se puede ver en la figura 5.1 existen tres tipos principales de pruebas. Las pruebas unitarias son las encargadas de probar fragmentos pequeños de código de manera aislada, se suelen utilizar para probar clases o métodos. *End to end* son pruebas que engloban pruebas más grandes como pruebas de flujos de tareas junto a una interfaz. Y por último, unas pruebas de tamaño intermedio que se llaman de *integración*, que prueban dos o más módulos o clases.

Además, en Android se consideran dos entornos en los que se pueden realizar pruebas. Las pruebas locales [116] que se ejecutan en la máquina local o en un servidor, suelen ser pequeñas pruebas que se ejecutan con rapidez. El otro tipo de pruebas son las *Instrumented* [117], estas se ejecutan en dispositivos Android, pudiendo ser dispositivos emulados o reales, en estos es necesario instalar la

aplicación sobre la que se van a ejecutar los tests.

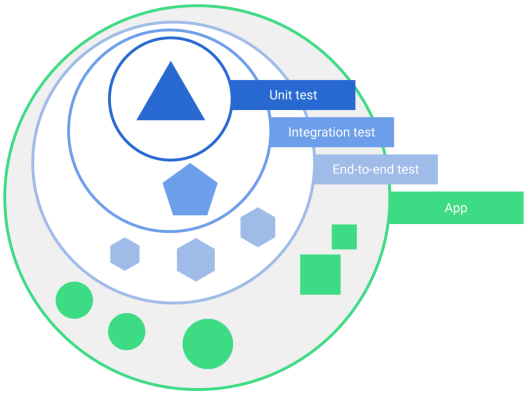


Figura 5.1: Ámbito de las pruebas en Android [115]

Como se ha comentado previamente, no se ha tenido el tiempo para hacer todas las pruebas que necesitaría el proyecto, pero sí que se han realizado dos tests para probar las herramientas de pruebas en Android. Se ha realizado un test unitario y un test de integración, ambos se han hecho de forma local.

La primera prueba (ver figura 5.2) es una prueba unitaria del caso de uso para obtener las paradas. Se han empleado *Mocks* para simular el comportamiento del repositorio. También hay que destacar que para probar esta porción de código se ha tenido lanzar en un hilo distinto al principal, ya que en el código se llama a una corrutina para que se ejecute la tarea en segundo plano, esto se consigue con **runBlocking coEvery** y **coVerify**.

```
package es.uva.retobici.frontend.domain.usecase

import ...

class GetStopsUseCaseTest {

    @RelaxedMockK
    private lateinit var stopRepository: StopRemoteDataSource
    lateinit var getStopsUseCase: GetStopsUseCase

    @Before
    fun setUp() {
        MockKAnnotations.init( ...obj: this)
        getStopsUseCase = GetStopsUseCase(stopRepository)
    }

    @Test
    fun `get stop list from the api`() = runBlocking {  this: CoroutineScope
        //Given
        val listStops = listOf(Stop( id: 1, lng: -4.731, lat: 41.653, address: "Plaza de Poniente",  totalSpaces: 10, listOf(),  reservedPedalBikes: 0, reservedElectricBikes: 0))
        coEvery { stopRepository.getAllStops() } returns listStops
        //When
        val response = getStopsUseCase()

        //Then
        coVerify(exactly = 1) { stopRepository.getAllStops() }
        assertEquals(response, listStops)
    }
}
```

Figura 5.2: Código del test para obtener paradas

La segunda prueba (ver figura 5.3) es una prueba de integración en la que se prueba un *ViewModel*, los *ViewModel* son componentes más complejos que almacenan información a la vez que operan con ella, por eso se ha realizado una prueba de integración junto a la prueba de un caso de uso. En esta

prueba se comprueba que el proceso de inicio de sesión es correcto y válido.

```
package es.uva.retobici.frontend.ui.viewmodels

import ...

@OptIn(ExperimentalCoroutinesApi::class)
class LoginViewModelTest {

    @RelaxedMockK
    private lateinit var loginUseCase: LoginUseCase
    @RelaxedMockK
    private lateinit var logOutUseCase: LogOutUseCase
    private lateinit var loginViewModel: LoginViewModel

    @get:Rule
    var rule: InstantTaskExecutorRule = InstantTaskExecutorRule()

    @Before
    fun setUp() {
        MockKAnnotations.init( ...obj: this)
        loginViewModel = LoginViewModel(loginUseCase,logOutUseCase)
        Dispatchers.setMain(Dispatchers.Unconfined)
    }

    @After
    fun tearDown() {
        Dispatchers.resetMain()
    }

    @Test
    fun `use log in test`() = runTest {  this: TestScope
        val email = "jorge@uva.es"
        val password = "12345678"
        //Given
        val user = User( id: 1,email, token: "token",  points: 100)
        val response: Result<User> = Result.Success(user)
        coEvery { loginUseCase(email,password) } returns response as Result.Success

        //When
        loginViewModel.login(email,password)

        //Then
        coVerify(exactly = 1) { loginUseCase.invoke(email,password) }
        assertEquals(loginViewModel.loginResult.value?.success, response.data )
    }
}
```

Figura 5.3: Código del test operación de login

5.2. Pruebas Back End

Por la parte del Back End, Laravel [118] proporciona una serie de herramientas que facilitan las pruebas unitarias y las pruebas de integración. Las pruebas se ejecutan con *phpunit* [113]. En este caso se ha decidido hacer una prueba de integración que pruebe el funcionamiento de la llamada a la API para desbloquear una bicicleta.

Como se puede ver en la figura 5.4 en la prueba, primero se hace una llamada a la API para iniciar sesión y obtener el token de autenticación que luego es enviado en la segunda llamada. Laravel te

permite comprobar el contenido de la respuesta en formato JSON, además de comprobar el estado de la llamada, si ha sido satisfactoria o no.

```php
<?php

namespace Tests\Feature;

use ...

class UnlockBikeTest extends TestCase
{
    private const BIKEID = 1;
    private const ENDPOINT = "/api/stops/1/unlock";
    private const LOGIN = "/api/login";
    private const PASSWORD = "123456";
    private const EMAIL = "jorge@uva.es";

    /**
     * A basic test example.
     *
     * @return void
     */
    public function test_check_unlocking_availability_endpoint()
    {
        $login = $this->postJson( uri: self::LOGIN, [
            'email' => self::EMAIL,
            'password' => self::PASSWORD
        ]);
        $token = $login->json( key: 'token');
        $this->withHeaders([
            'Authorization' => 'Bearer '. $token,
            'Accept' => 'application/json'
        ])
            ->postJson( uri: self::ENDPOINT)
            ->assertStatus( status: 200)
            ->assertJson(fn(AssertableJson $json) =>
            $json
                ->where( key: 'bike_id',  expected: self::BIKEID)
                ->where( key: 'unlocked',  expected: false)
                ->etc()
            );
    }
}
```

Figura 5.4: Código del test realizado en Laravel

5.3. Test de usabilidad

Se ha decidido realizar un test de usabilidad para comprobar que la aplicación es sencilla e intuitiva de utilizar para un usuario que la utiliza por primera vez. El test se realizará con el dispositivo que se ha desarrollado el proyecto, partiendo de un estado inicial de la base de datos generados gracias a los *seeders*. La prueba será realizada por dos usuarios, que no han tenido contacto con la aplicación previamente, pero que son personas jóvenes y, por tanto, habituados a la tecnología. Realizarán dos tareas en la aplicación. Durante el proceso se estará observando como interactúan con la aplicación y se anotarán los resultados de las siguientes métricas.

- Tiempo empleado para completar la tarea.
- Número de clics/pulsaciones en la pantalla para completar la tarea.
- Errores encontrados durante el proceso.
- Comentarios realizados por los usuarios.

A partir de estas métricas se medirán los siguientes atributos de usabilidad.

- **Eficiencia**: será mayor o menor si el usuario consigue realizar la tarea sin ayuda externa.
- **Eficacia**: será mayor si el usuario consigue realizar la tarea sin pulsaciones innecesarias y en un tiempo razonable establecido.
- **Satisfacción**: se considerarán los comentarios que pueda hacer el usuario, además de las respuestas del usuario a las siguientes preguntas:

  ¿Considera la interfaz intuitiva?

  ¿En algún momento no sabía como realizar la tarea?

  ¿La interfaz le ha parecido, adecuada, textos, iconos?

Las dos tareas a realizar son:

- **Realizar una ruta**: El usuario deberá primero visualizar una parada y reservar una bicicleta tras iniciar sesión, si es posible en dicha parada, tras reservar debe desbloquear una bicicleta escaneando el código QR de la misma e iniciar la ruta, por último deberá bloquear la bicicleta en una parada escaneando el QR de la parada.
- **Obtener una recompensa**: El usuario deberá obtener una recompensa que no haya obtenido ya y que tenga puntos disponibles para ello.

Antes de comenzar el test se pondrá en contexto a los usuarios, se les explicará en que consiste la aplicación y las tareas que tienen que realizar. En el momento de realización de la prueba no se

pudo encontrar a personas de diferentes edades, por lo que ambos usuarios tienen una edad en torno a los 22 años, esto no se considera un problema, ya que se considera que gran parte de usuarios de la aplicación serían personas jóvenes.

5.3.1. Resultados test usabilidad

A continuación se pueden ver los resultados obtenidos de las dos pruebas, hay que destacar que la primera tarea se ha dividido en tres subtareas, iniciar sesión, reservar una bicicleta y realizar la ruta.

Como se puede ver en la tabla 5.1 ambos usuarios completaron la tarea en un tiempo razonable para ser la primera vez que usaban la aplicación, hicieron más pulsaciones de las necesarias, pero se considera algo normal para la primera vez de uso.

Iniciar sesión	Usuario 1	Usuario 2	Baremo
Éxito	Si	Si	-
Tiempo utilizado	30s	27s	25s
Pulsaciones	5	4	3
Interfaz intuitiva	si	si	-
No saber realizar la tarea	no	no	-
Interfaz adecuada	si	si	-

Tabla 5.1: Métricas test usabilidad de la tarea iniciar sesión

Respecto a la sub-tarea de reservar una bicicleta, el primer usuario se quedó bloqueado, ya que para reservar una bicicleta es necesario iniciar sesión y no recordó que era necesario hasta que después de varias pulsaciones leyó el mensaje en el botón que indica que tiene que iniciar sesión. Tras finalizar consideró que era necesario un aviso más claro para indicar que era necesario iniciar sesión, también comentó que podría ser redireccionado a la pantalla de login en el momento que se necesitara, para así evitar que el usuario no supiera que hacer. Por otro lado, el segundo usuario de prueba no tuvo ningún problema y el flujo de ejecución fue normal.

Reservar Bicicleta	Usuario 1	Usuario 2	Baremo
Éxito	Si	Si	-
Tiempo utilizado	20s	25s	22s
Pulsaciones	7	4	4
Interfaz intuitiva	si	si	-
No saber realizar la tarea	si	no	-
Interfaz adecuada	si	si	-

Tabla 5.2: Métricas test usabilidad de la tarea reservar bicicleta

En la sub-tarea de realizar la ruta (ver tabla 5.3) todas las métricas han sido positivas, con un tiempo y un número de pulsaciones razonable para ser la primera vez de uso. También comentaron que la interfaz era intuitiva gracias a los códigos QR y que se proporcionaba la información necesaria.

Por último comentar la tarea de obtener una recompensa (ver figura 5.4), las métricas reflejan que ambos usuarios fueron capaces de realizar la tarea en un tiempo y pulsaciones razonables y opinaron

Realizar Ruta	Usuario 1	Usuario 2	Baremo
Éxito	Si	Si	-
Tiempo utilizado	50s	45s	40s
Pulsaciones	8	8	7
Interfaz intuitiva	si	si	-
No saber realizar la tarea	no	no	-
Interfaz adecuada	si	si	-

Tabla 5.3: Métricas test usabilidad de la tarea realizar ruta

que les resultó intuitivo el proceso para obtener una recompensa.

Obtener Recompensa	Usuario 1	Usuario 2	Baremo
Éxito	Si	Si	-
Tiempo utilizado	30s	23s	20s
Pulsaciones	6	5	4
Interfaz intuitiva	si	si	-
No saber realizar la tarea	no	no	-
Interfaz adecuada	si	si	-

Tabla 5.4: Métricas test usabilidad de la tarea obtener recompensa

5.3.2. Conclusiones test de usabilidad

Ambas pruebas han sido satisfactorias, pero durante el proceso se han identificado varios puntos de mejora que podrían ser implementados en un futuro. Lo primero es informar de forma más clara la necesidad de iniciar sesión o una guía inicial que proporciona la información necesaria para el uso de la aplicación. Cabe destacar que un estudio más amplio sería de gran ayuda antes de tomar cualquier tipo de decisión de interfaz. Además, al no tener acceso a bicicletas, la prueba no refleja completamente el flujo de las tareas.

Capítulo 6

Conclusiones

En este capítulo se comentan las diferentes ideas y conclusiones que se han obtenido y desarrollando a lo largo del transcurso del proyecto. También se comentan los diferentes trabajos futuros que se podrían aplicar para mejor el proyecto en término de implementación y en funcionalidades.

Lo primero a destacar es que como el proyecto no está basado en ningún otro trabajo de fin de grado anterior ni un tema propuesto por la propia universidad, sino que ha sido una idea original, se ha podido amoldar hasta cierto nivel a las necesidades o gustos del alumno, teniendo plena libertad de elegir tecnologías e implementación. De esta manera se considera que ha sido de gran valor el proceso de elección de tecnologías, ya que ha permitido explorar diferentes opciones, valorando sus puntos negativos y positivos, acercándose a un escenario más realista en cuanto a lo laboral, ya que se considera que se ha desarrollado un sistema lo más cercano a un producto realista.

Lo esencial de cuando se finaliza un proyecto es evaluar si se han cumplido los objetivos que se establecieron inicialmente. Se considera que se han cumplido los dos objetivos que se propusieron, el primero, desarrollar un mapa que te permite visualizar las paradas y facilitar el uso y el alquiler de bicicletas, al haberse implementado las funcionalidades que se establecieron como de alta prioridad, se ha conseguido un mapa funcional que permite gestionar el proceso de alquiler/desbloqueo de bicicletas, realización de la ruta y bloqueo de la bicicleta. El segundo objetivo consistía en desarrollar un sistema de puntuación que incentivase el uso de la aplicación, se considera que las recompensas que se ofrecen al usuario, aunque dependan de acuerdos con los diferentes ofertantes, incentivaría el uso de la aplicación por los usuarios.

En cuanto a las nuevas tecnologías empleadas, como se decidió en parte emplear las más modernas y recomendadas como Kotlin o las últimas versiones de Android, también se considera un punto positivo, porque ha facilitado el aprendizaje de estas. También sobre el uso de herramientas auxiliares como Postman para el desarrollo de la API, se considera muy positivo para el futuro, ya que son herramientas empleadas en un entorno laboral.

Las pruebas facilitaron la identificación de mejoras, principalmente sobre la interfaz, que durante el desarrollo no se tuvieron en cuenta, ya que al desarrollar el proyecto de manera individual no se obtiene ningún tipo de retroalimentación de usuarios o mismamente no se tiene el conocimiento de

un diseñador. Con respecto a las pruebas unitarias y de integración, ha sido de utilidad las pruebas realizadas, que aunque escasas en cantidad, emplean varias herramientas de ambas plataformas que cubren considerablemente como se harían el resto de pruebas.

Por último, comentar que el punto negativo que se ha encontrado has sido la falta de tiempo, no por problemas de organización, porque se han cumplido sin mucho desfase los plazos de la planificación inicial. Pero se considera que un proyecto como este, que implementa ambas partes, tanto el Front End como el Back End necesita más tiempo del que puede abarcar un trabajo de final de grado. Como se va a comentar a continuación existen multitud de ámbitos en los que se puede mejorar la aplicación o agregar nuevas funcionalidades.

6.1. Trabajos futuros

A lo largo del desarrollo del proyecto se ha ido teniendo nuevas ideas de posibles mejoras o ampliaciones de la aplicación con nuevas funcionalidades. La siguiente lista muestra estas nuevas funcionalidades en orden de prioridad algunas de las ideas:

- **Nuevas funcionalidades**: La **navegación** que permita al usuario escoger su punto de inicio y fin al comienzo de la ruta, obteniendo de esta manera una serie de direcciones para realizar la ruta. La **búsqueda** de direcciones, que muestre paradas cercanas a dicha búsqueda.
- **Mejoras de funcionalidades ya existentes**: Como **mejoras de usuario,** incluyendo el registro de usuario, **Sistema de puntuación,** ampliando el sistema de incluyendo clasificaciones de usuarios, competiciones enfocadas al uso de la bicicleta o retos como realizar un trayecto al día durante una semana para ganar más puntos.
- **Integración sistema real**: La aplicación se podría proponer a diferentes entidades públicas o privadas como punto de partida para un desarrollo más profesional de un sistema de alquiler de bicicletas, incluyendo un **sistema de pago** o de suscripciones.
- **Pruebas**. Para completar el proyecto de manera profesional se considera necesario el desarrollo completo de las pruebas en ambas partes. Tanto con elementos reales como bicicletas y paradas que tengan integrados los códigos QR.
- **Actualizaciones en tiempo real**. Al ser una aplicación que sería empleada por múltiples usuarios al mismo tiempo, se debe considerar las actualizaciones en tiempo real de los elementos compartidos por todos los usuarios.
- **Interfaz**. Se podría mejorar la interfaz, tanto con las recomendaciones de los usuarios como con información que se ha obtenido durante el desarrollo, pero que por falta de tiempo no ha sido posible aplicar.
- **Android**. Android es un sistema que evoluciona constantemente y aunque se han intentado emplear las librerías más modernas, se podría mejorar la aplicación con la migración a las nuevas librerías disponibles de Android.

Apéndices

Apéndice A

Acrónimos

- **API**: Application Programming Interface.
- **APK**: Android Package Kit
- **DAO**: Data Access Object
- **DTO**: Data Transfer Object
- **FURPS**: Functionality Usability Reliability Performance Supportability
- **HAL**: Hardware Abstraction Layer
- **HTTP**: Hyper Text Transfer Protocol
- **JVM**: Java Virtual Machine
- **MVC**: Model View Controller
- **MVVM**: Model View View Model
- **ORM**: Object Relational Mapping
- **SDK**: Software Development Kit
- **UI**: User Interface

Apéndice B

Manual de despliegue

B.1. Despliegue Back End

A continuación se muestran los pasos a seguir para instalar e iniciar del sistema Back End. Cabe destacar que estos pasos son los referentes al entorno que se ha utilizado en el proyecto, en este caso se ha utilizado el propio ordenador portátil como servidor, con un sistema operativo macOS, por lo que en otro sistema operativo los comandos de instalación serán diferentes. MacOS ofrece un sistema de paquetes *Homebrew* [119] que facilita enormemente el proceso de instalación y configuración de la gran mayoría de herramientas.

B.1.1. Instalación PHP 8.0.8

1. Iniciar un emulador de terminal en cualquier directorio del sistema
2. Ejecutar el comando `brew install php`
3. Si el sistema no detecta automáticamente la instalación, se puede cargar en el arranque del terminal de la siguiente manera
   a) Editar el contenido del fichero `.bashrc` o `.zshrc` dependiendo de que lenguaje de bash utilices. Normalmente ubicado en el directorio principal del usuario. Un comando para editar `vim /.bashrc`
   b) Exporta la ubicación de la instalación de PHP agregando la siguiente línea `export PATH=/Applications/MAMP/bin/php/php8.0.8/bin:$PATH`.

B.1.2. Instalación PostgreSQL

1. Iniciar un emulador de terminal en cualquier directorio del sistema
2. Ejecutar el comando `brew install postgresql`

3. Iniciar el servicio con `brew services start postgres`
4. Entrar en la interfaz de comandos de postgresql con `psql postgres`
5. Cambiar la contraseña del usuario que se vaya a usar, en este caso root y de contraseña también root, es recomendable que para un entorno de producción no se utilice ni el usuario root ni una contraseña tan débil. El comando sería `ALTER USER root WITH PASSWORD 'root';`
6. Crear la base de datos con el comando `create database retobici;`

B.1.3. Generar claves API Mapbox

Para utilizar el sistema de mapas es necesaria una cuenta y generar las claves para el uso del sistema. Una vez creada la cuenta, en la dirección "https://account.mapbox.com/çomo se puede ver en la figura B.1 se generan las dos claves, tanto la pública como la privada, que se usarán más adelante en ambas partes del proyecto.

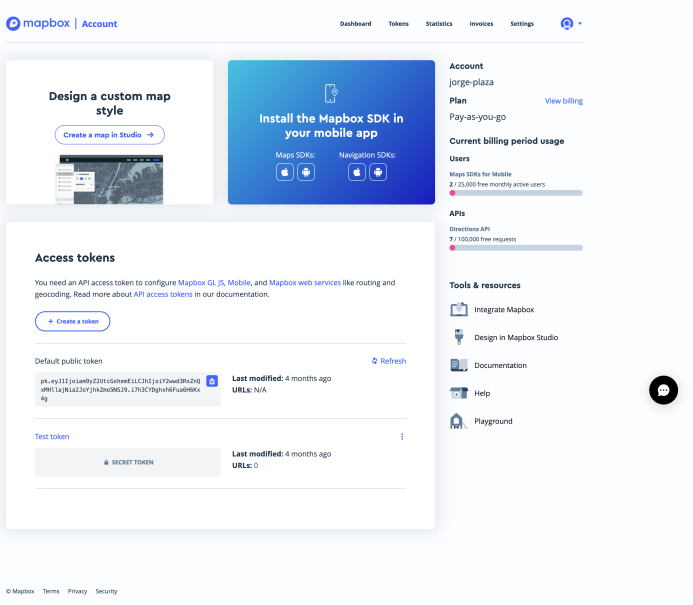


Figura B.1: Página generación de claves de Mapbox

B.1.4. Instalación Back End

El código se encuentra alojado en el repositorio GitLab de la escuela en la siguiente dirección https://gitlab.inf.uva.es/jorplaz/plazalazojorge_2022_backend.

1. Si no está instalado el sistema de versiones git, el comando para instalarlo en macOS `brew install git`.
2. Descargar o clonar el repositorio en el directorio deseado con el comando `git clone https://gitlab.inf.uva.es/jorplaz/plazalazojorge_2022_backend.git`.

Ahora, con el código del repositorio descargado es necesario configurar el entorno del proyecto:

1. Desde la carpeta raíz del proyecto copiar el fichero de entorno de ejemplo con `cp .env.example .env`.
2. Agregar los contenidos al fichero .env
   a) `DB_CONNECTION=pgsql`
   b) `DB_HOST=127.0.0.1`
   c) `DB_PORT=5432`
   d) `DB_DATABASE=retobici`
   e) `DB_USERNAME=root`
   f) `DB_PASSWORD=root`
   g) `MAPBOX_TOKEN=clave pública generada en el paso anterior`
3. Generar la clave de aplicación con el comando `php artisan key:generate`
4. Instalar las dependencias del fichero `composer.yml` con el comando `composer install`

Con la instalación y la configuración completada solo sería necesario lanzar el servidor, para ello hay que tener en cuenta de que el proyecto se ha desarrollado de manera local, entonces para que se comunique la aplicación Android con el Back End, ambos tienen que estar en la misma red WiFi.

1. Obtener la dirección IP del servidor en la red WiFi con el comando `ifconfig`
2. Lanzar el servidor apuntando a la IP de la red WiFi obtenida en el paso anterior
   `php artisan serv -host=192.168.65.9 -port=8000`
3. Migrar la base de datos para generar la estructura de las tablas y poblar la base de datos de información con el comando `php artisan migrate:fresh -seed`

B.2. Instalación Front End

La aplicación se debe instalar en un dispositivo Android con una versión 5.0 (API 21) como mínimo, pero es recomendable por rendimiento utilizar un dispositivo con la última versión disponible. Junto a esta memoria se entrega el fichero `retobici.apk`, al descargarse en el dispositivo móvil, normalmente se ubica en la carpeta *Descargas* del dispositivo, al ejecutarse el sistema Android pedirá permisos para instalar aplicaciones externas, tras confirmar la aplicación será instalada.

Hay que detallar ciertos aspectos de la configuración del proyecto Android que podrían variar dependiendo del entorno. Hay que destacar que todos estos valores ya se encuentran definidos en la APK y que esta sección se detalla por si fuera necesario en un futuro.

Sería necesario agregar las claves de Mapbox que se han generado previamente.

1. La clave secreta tiene que agregarse en dos ubicaciones

   a) En un fichero de configuración de Gradle, editar o crear si no existe el fichero `/.gradle/gradle.properties` agregando el siguiente contenido `MAPBOX_DOWNLOADS_TOKEN=clave-secreta-mapbox`

   b) En el fichero `Retobici/plazalazojorge_2022_frontend/settings.gradle` se tiene que agregar el repositorio Mapbox (ver figura B.2)

2. La clave pública se agrega en el fichero `res/values/mapbox_access_token.xml`

```
dependencyResolutionManagement { DependencyResolutionManagement it ->
    repositoriesMode.set(RepositoriesMode.FAIL_ON_PROJECT_REPOS)
    repositories { RepositoryHandler it ->
        google()
        mavenCentral()
        maven {
            url 'https://api.mapbox.com/downloads/v2/releases/maven'
            authentication { AuthenticationContainer it ->
                basic(BasicAuthentication)
            }
            credentials { PasswordCredentials it ->
                // Do not change the username below.
                // This should always be `mapbox` (not your username).
                username = "mapbox"
                // Use the secret token you stored in gradle.properties as the password
                password = "mapbox-secret-key"
            }
        }
        maven { url 'https://jitpack.io' }
    }
}
```

Figura B.2: Contenido de settings.gradle

Por último destacar que para que el Front End se comunique con el Back End es necesario que en el fichero `es/uva/retobici/frontend/core/di/NetworkModule.kt` en el que se configura *Retrofit*, el cliente HTTP, configurar la misma dirección IP que la que se use en el Back End, como se puede ver en la figura B.3

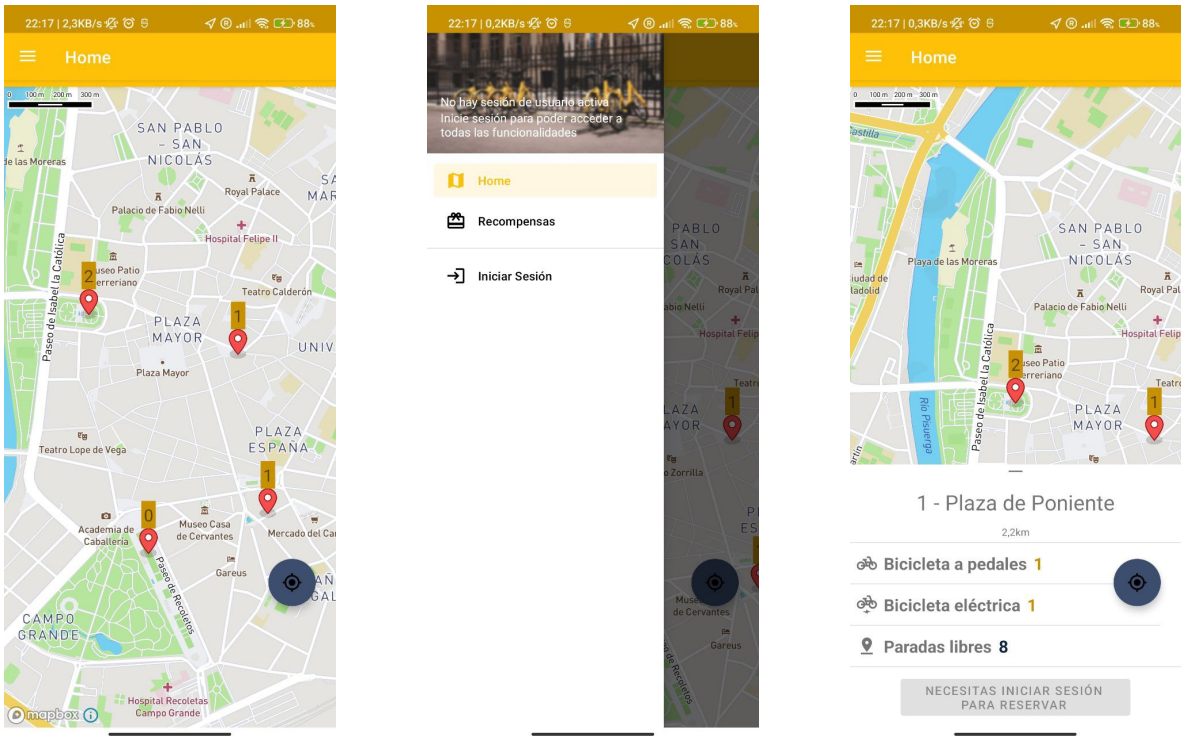
```
@Singleton
@Provides
fun providesRetrofit():Retrofit{
    return Retrofit.Builder()
        .baseUrl( baseUrl: "http://192.168.65.9:8000/api/")
        .addConverterFactory(GsonConverterFactory.create())
        .build()
}
```

Figura B.3: Configuración Retrofit en Android

Apéndice C

Manual de uso

Lo primero que ve el usuario tras abrir la aplicación es la vista del mapa con las distintas paradas (ver figura C.1a). El usuario no tiene una sesión de usuario, puede seleccionar una parada y ver la información relacionada con ella (ver figura C.1c), como las bicicletas disponibles y los espacios para depositar bicicletas, pero no puede reservar ni desbloquear ninguna. Para poder acceder a esas funcionalidades es necesario iniciar sesión. Para iniciar sesión primero debe acceder al menú lateral (ver figura C.1b) en el apartado *Iniciar Sesión*.



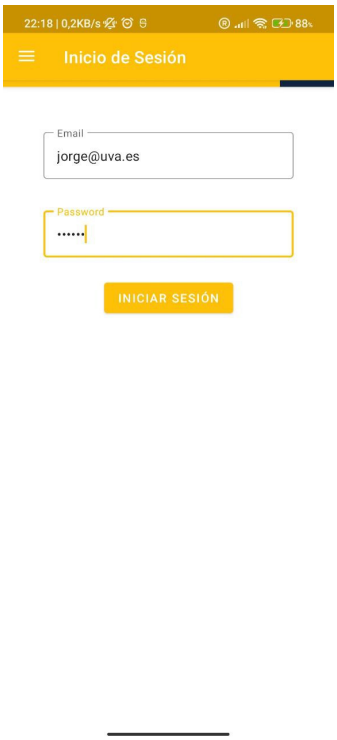
(a) Pantalla principal sin sesión activa

(b) Menú lateral sin sesión activa

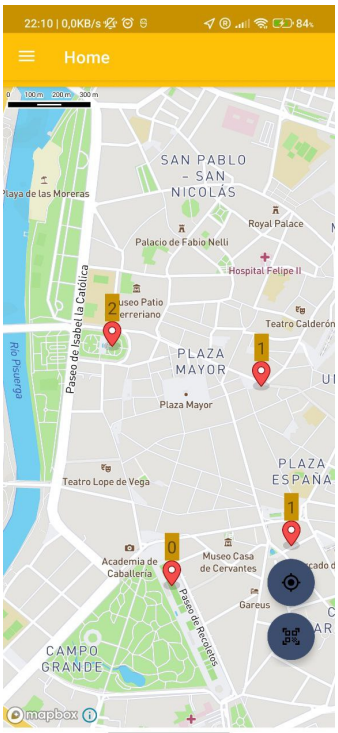
(c) Parada de bicicleta sin sesión activa

Figura C.1: Pantallas principales sin sesión de usuario activa

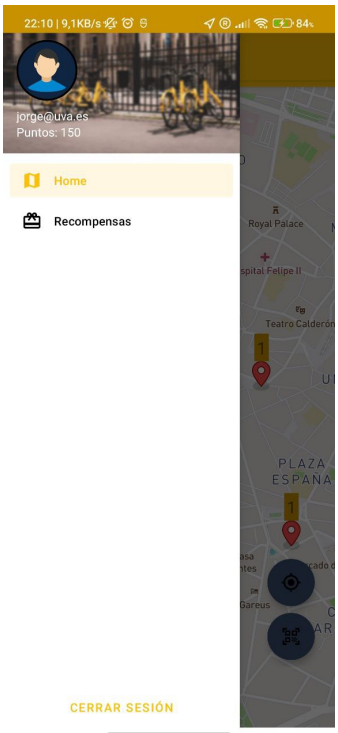
Tras acceder a la pantalla de inicio de sesión (ver figura C.2a) se introducen las credenciales de usuario. Si las credenciales son correctas, el usuario es redirigido a la pantalla principal, pero con la disponibilidad de realizar las tareas de un usuario con una sesión activa (ver figura C.2b). Ahora el botón para acceder al escáner QR está disponible en la esquina inferior derecha. También ha cambiado el menú lateral (ver figura C.2c), ahora no está disponible la posibilidad de iniciar sesión, pero está activa la función de cerrar sesión en la parte baja del menú, además la información del usuario se puede ver en la parte superior.



(a) Pantalla de inicio de sesión



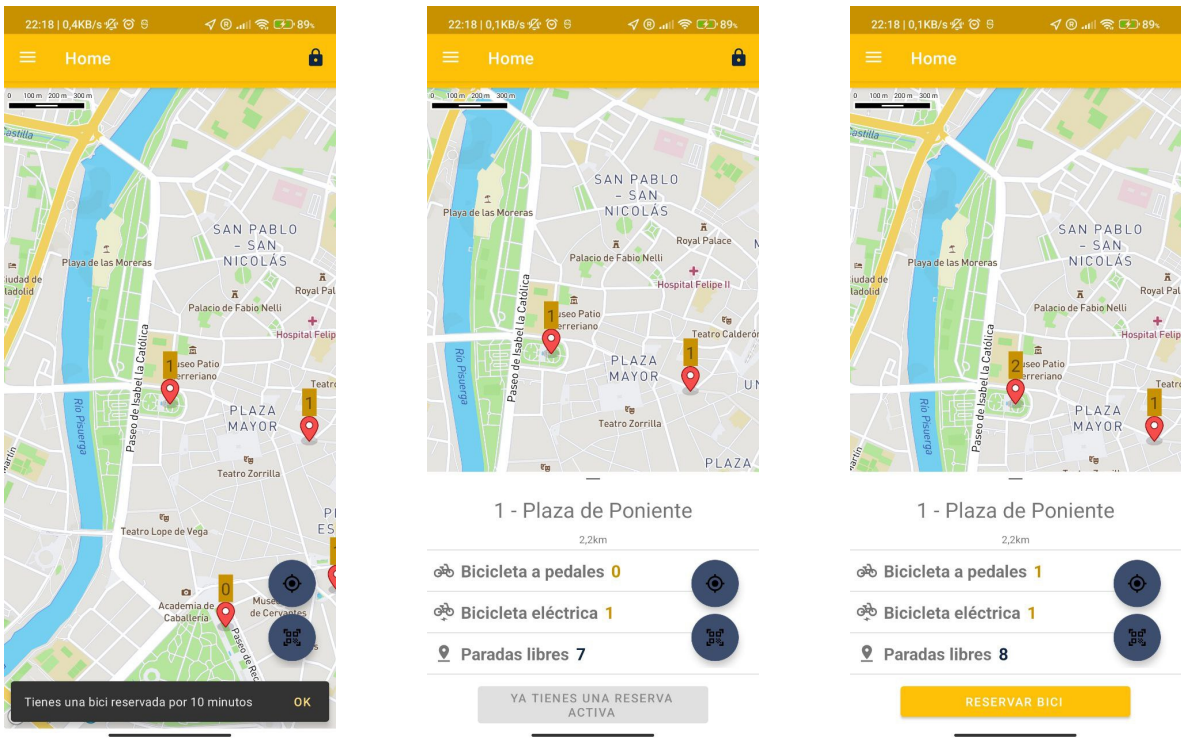
(b) Pantalla principal con sesión activa



(c) Menú lateral con sesión activa

Figura C.2: Pantallas principales para login y después de inicio de sesión

Habiendo iniciado sesión el usuario puede reservar una bicicleta, primero tiene que pulsar sobre la parada donde quiera reservar una bicicleta, tras visualizar la información de la parada, si hay bicicletas disponibles, puede pulsar sobre el tipo de bicicleta que quiera reservar (ver figura C.3a), de esta manera se habilita el botón para reservar, si se pulsa se reserva una bicicleta de ese tipo para el usuario en esa parada. Con una reserva activa (ver figura C.3b) se muestra la pantalla principal pero con un icono en la parte superior, que te informa de que la reserva sigue vigente, cuando se pulsa se muestra el mensaje con el recordatorio. Por último, el usuario puede seguir navegando por la aplicación mientras tiene la reserva activa, si pulsa sobre cualquier parada no tendrá la posibilidad de reservar otra bicicleta (ver figura C.3c), ya que ya tiene una reserva activa.



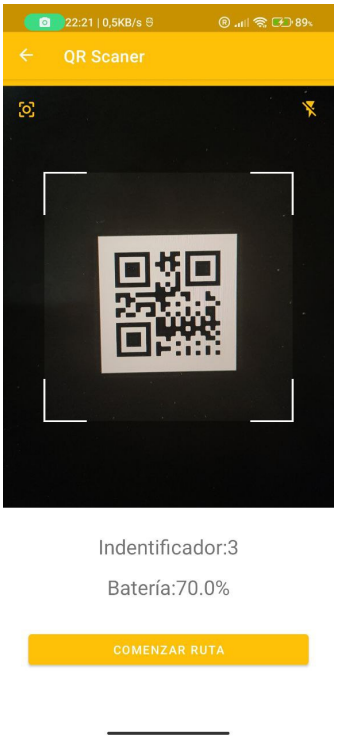
(a) Parada de bicicletas disponible para reserva

(b) Pantalla principal con reserva activa

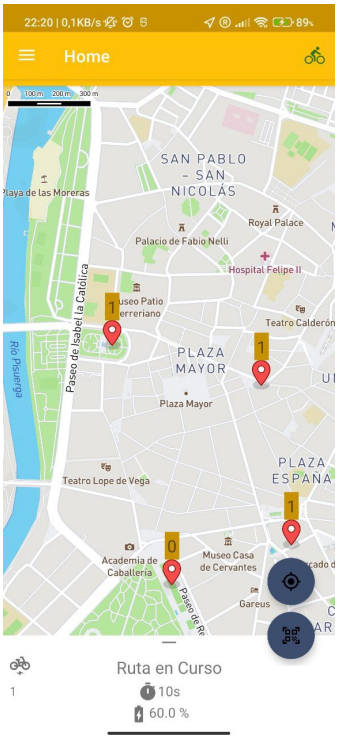
(c) Parada de bicicletas con reserva activa

Figura C.3: Pantallas relacionadas con la reserva de una bicicleta

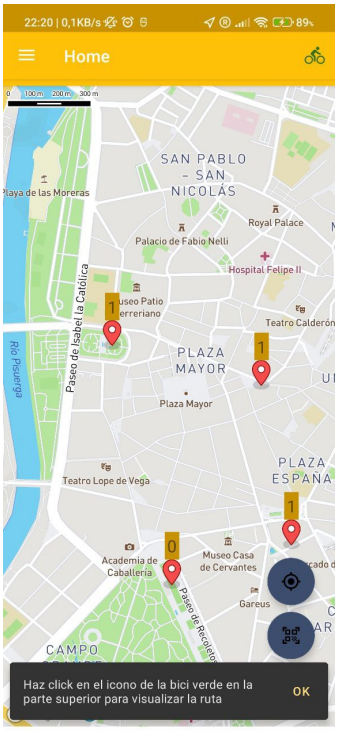
Tanto si el usuario tiene una reserva activa como si no, con una sesión activa, el usuario puede comenzar el proceso de realizar una ruta. Lo primero que tiene que hacer es navegar a la pantalla del escáner QR pulsando sobre el botón con el icono del QR ubicado en la pantalla principal. En el escáner QR (ver figura C.4a) el usuario deberá escanear el código QR de una parada o de una bicicleta, el sistema determinará si hay bicicletas disponibles para desbloquear en esa parada y de ese tipo de bicicleta, en el caso de que haya disponibilidad se mostrará el identificador de la bicicleta y en el caso de que sea eléctrica la batería de la que dispone. Cuando el usuario pulsa sobre el botón de iniciar ruta, es redirigido a la pantalla principal donde se muestra el estado de la ruta (ver figura C.4b). En el caso de que el panel inferior se oculte, se muestra un mensaje de como abrirlo de nuevo, de forma similar a una reserva activa, en la parte superior se muestra un icono (ver figura C.4c) que si es pulsado se abre de nuevo la sección con la información de la parada.



(a) Escáner QR para desbloquear una bicicleta



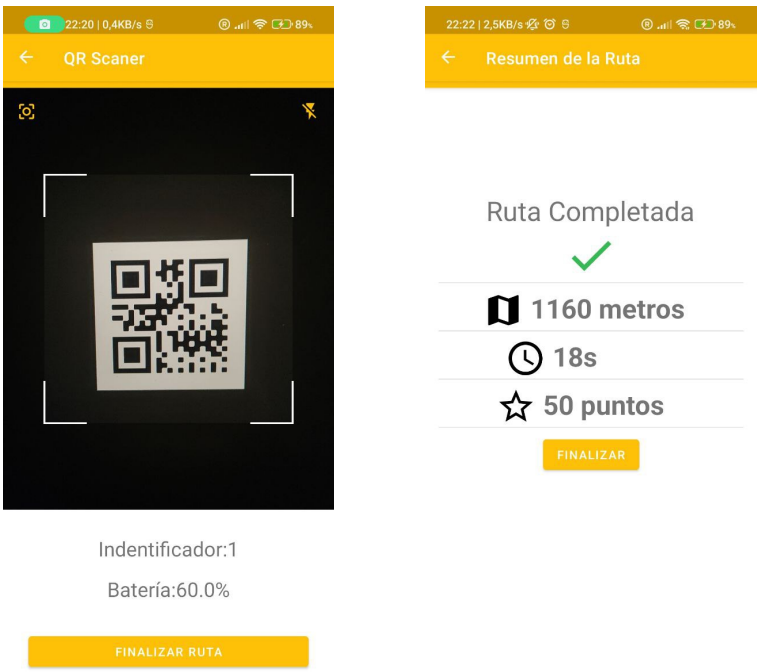
(b) Pantalla principal con ruta activa



(c) Pantalla principal con ruta activa pero información oculta

Figura C.4: Pantallas relacionadas con la realización de una ruta

En el momento que el usuario haya completado su ruta y haya llegado a la parada de destino, el usuario deberá escanear el código QR ubicado en la parada, el sistema determinará si se puede bloquear la bicicleta o no, en el caso de que sea posible el usuario colocará la bicicleta en el espacio libre en la parada y pulsará el botón de finalizar ruta (ver figura C.5a). Tras pulsar el botón la ruta se finalizará y el sistema calculará la puntuación y otros datos y se los mostrará al usuario (ver figura C.5b). Tras pulsar en el botón de finalizar, el usuario es redirigido a la pantalla principal en la que se vuelven a mostrar las paradas.

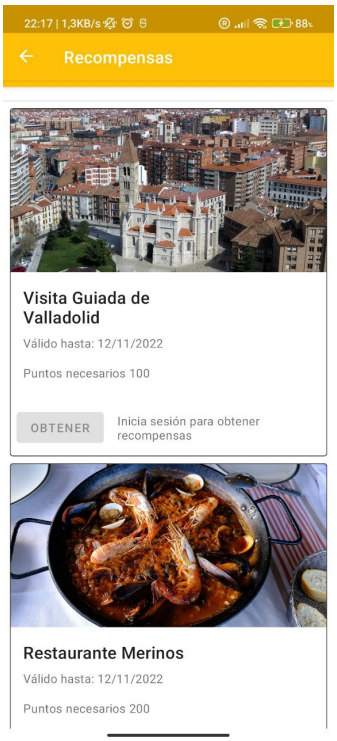


(a) Escáner QR para bloquear una bicicleta

(b) Pantalla resumen de ruta tras finalizar

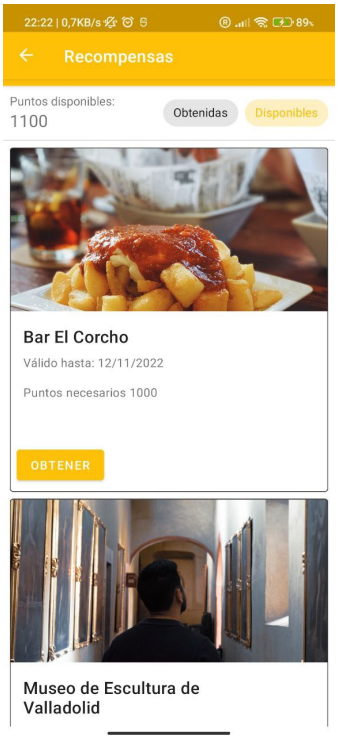
Figura C.5: Pantallas relacionadas con la finalización de una ruta

Respecto a la obtención de recompensas, se puede acceder a esta sección a través del menú lateral. Se puede acceder sin una sesión de usuario activa y con ella, pero el contenido de la sección será diferente. En el caso de acceder sin una sesión de usuario activa se mostrarán todas las recompensas (ver figura C.6a) pero sin la posibilidad de obtenerlas. Una vez se haya iniciado sesión se mostrará la lista de recompensas obtenidas previamente (ver figura C.6b) con la información de las mismas. Si se selecciona en la parte superior el filtro de *disponible* se muestran las recompensas que el usuario puede obtener (ver figura C.6c), en el caso de que quiera obtener una debe de tener los puntos necesarios para ello, si dispone de los puntos necesarios puede pulsar sobre el botón para obtener la recompensa y esta se registrará en la lista de recompensas obtenidas, mostrando el código QR para utilizar la recompensa.



(a) Pantalla de recompensas sin sesión iniciada

(b) Pantalla de recompensas obtenidas por usuario



(c) Pantalla de recompensas disponibles para obtener

Figura C.6: Pantallas de recompensas sin sesión de usuario y con sesión iniciada

Apéndice D

Contenido del TFG

Junto con la memoria del TFG, cuyo nombre es MemoriaTFGJorgePlazaLazo.pdf encontramos otro archivo PDF (DatosTFGJorgePlazaLazo.pdf) que contiene una URL a los repositorios del GitLab de la Escuela de Ingeniería Informática de la Universidad de Valladolid. Los contenidos de los repositorios son los siguientes:

- **plazalazojorge_2022_frontend**: Contiene el código fuente del proyecto Android junto al fichero APK *retrobici.apk* que permite instalar la aplicación sin compilar el código fuente. También se incluyen los bocetos en formato de Balsamiq Wireframes.
- **plazalazojorge_2022_backend**: Contiene el código fuente del proyecto Laravel junto a las instrucciones para desplegar el proyecto.

Bibliografía

[1] El calentamiento global en 2022: causas y consecuencias. Visitado: 06/03/2022. [Online]. Available: https://climate.selectra.com/es/que-es/calentamiento-global#:~:text=Calentamiento%20global%20y%20cambio%20clim%C3%A1tico,manera%20natural%20no%20se%20producir%C3%ADan

[2] Efecto invernadero: causas y consecuencias en el clima. Visitado: 06/03/2022. [Online]. Available: https://climate.selectra.com/es/que-es/efecto-invernadero

[3] El acuerdo de parís. Visitado: 06/03/2022. [Online]. Available: https://unfccc.int/es/process-and-meetings/the-paris-agreement/el-acuerdo-de-paris

[4] Plan integral de movilidad urbana sostenible y segura de la ciudad de valladolid. Visitado: 06/03/2022. [Online]. Available: https://www.valladolid.es/es/temas/hacemos/plan-integral-movilidad-urbana-ciudad-valladolid-pimuva

[5] Auvasa licita el nuevo servicio de préstamo de bicis a partir de febrero de 2023. Visitado: 06/03/2022. [Online]. Available: https://www.elnortedecastilla.es/valladolid/auvasa-licita-nuevo-20211221154456-nt.html

[6] El Área de movilidad y espacio urbano, con un presupuesto de 48,7 millones de euros, incrementa un 62 % las inversiones. Visitado: 06/03/2022. [Online]. Available: https://www.Valladolid.es/es/actualidad/noticias/area-movilidad-espacio-urbano-presupuesto-48-7-millones-eur

[7] El ayuntamiento finaliza la implantación de un amplio tramo de carril bici bidireccional en el paseo de zorrilla. Visitado: 06/03/2022. [Online]. Available: https://www.valladolid.es/es/actualidad/noticias/ayuntamiento-finaliza-implantacion-amplio-tramo-carril-bici

[8] Aplicación vallabici appstore. Visitado: 06/03/2022. [Online]. Available: https://play.google.com/store/apps/details?id=com.ingeniasolucionesenmovilidad.vallabici&gl=US

[9] Bikemi servicio de bicicletas de milán. Visitado: 06/03/2022. [Online]. Available: https://bikemi.com/

[10] Lime servicio de trasporte. Visitado: 06/03/2022. [Online]. Available: https://www.li.me/es-es/

[11] Aplicación duolingo. Visitado: 06/03/2022. [Online]. Available: https://play.google.com/store/apps/details?id=com.duolingo&hl=en

[12] Aplicación venchi. Visitado: 06/03/2022. [Online]. Available: https://play.google.com/store/apps/details?id=com.atono.venchi&gl=US

[13] Android operating system. Visitado: 8/04/2022. [Online]. Available: https://en.wikipedia.org/wiki/Android_(operating_system)

[14] Android platform architecture. Visitado: 8/04/2022. [Online]. Available: https://developer.android.com/guide/platform

[15] Android api levels. Visitado: 06/03/2022. [Online]. Available: https://apilevels.com/

[16] What is the difference between min sdk version/target sdk version vs. compile sdk version? Visitado: 8/04/2022. [Online]. Available: https://stackoverflow.com/questions/24510219/what-is-the-difference-between-min-sdk-version-target-sdk-version-vs-compile-sd

[17] Kotlin. Visitado: 8/04/2022. [Online]. Available: https://kotlinlang.org/

[18] Kotlin programming language. Visitado: 8/04/2022. [Online]. Available: https://en.wikipedia.org/wiki/Kotlin_(programming_language)

[19] Kotlin sealed classes. Visitado: 15/06/2022. [Online]. Available: https://kotlinlang.org/docs/sealed-classes.html#sealed-classes-and-when-expression

[20] Kotlin data classes. Visitado: 31/05/2022. [Online]. Available: https://kotlinlang.org/docs/data-classes.html

[21] Kotlin coroutines on android. Visitado: 8/04/2022. [Online]. Available: https://developer.android.com/kotlin/coroutines

[22] Laravel. Visitado: 21/04/2022. [Online]. Available: https://laravel.com/

[23] Rest apis. Visitado: 10/04/2022. [Online]. Available: https://www.ibm.com/cloud/learn/rest-apis

[24] Architecture pattern mvc. Visitado: 10/04/2022. [Online]. Available: https://www.w3schools.in/mvc-architecture

[25] Github laravel. Visitado: 10/04/2022. [Online]. Available: https://github.com/laravel/framework

[26] Laravel sanctum authentication. Visitado: 10/04/2022. [Online]. Available: https://laravel.com/docs/9.x/sanctum

[27] Laravel artisan console. Visitado: 06/03/2022. [Online]. Available: https://laravel.com/docs/9.x/artisan

[28] Usage statistics of php for websites. Visitado: 10/04/2022. [Online]. Available: https://w3techs.com/technologies/details/pl-php

[29] Mapbox documentation. Visitado: 7/04/2022. [Online]. Available: https://docs.mapbox.com/

[30] Mapbox wikipedia. Visitado: 06/03/2022. [Online]. Available: https://en.wikipedia.org/wiki/Mapbox

[31] Maps sdk for android. Visitado: 8/04/2022. [Online]. Available: https://docs.mapbox.com/android/maps/guides/

[32] Search sdk for android. Visitado: 8/04/2022. [Online]. Available: https://docs.mapbox.com/android/navigation/guides/

[33] Mapbox studio. Visitado: 19/04/2022. [Online]. Available: https://www.mapbox.com/mapbox-studio/

[34] Android sdk version 31. Visitado: 8/04/2022. [Online]. Available: https://developer.android.com/studio/

[35] Android sdk minimun version for mapbox. Visitado: 8/04/2022. [Online]. Available: https://docs.mapbox.com/android/maps/guides/#requirements

[36] Android studio bumblebee version. Visitado: 8/04/2022. [Online]. Available: https://developer.android.com/studio/

[37] Postgresql. Visitado: 21/04/2022. [Online]. Available: https://www.postgresql.org/

[38] Postman. Visitado: 21/04/2022. [Online]. Available: https://www.postman.com/

[39] Php 8.0.8. Visitado: 20/04/2022. [Online]. Available: https://www.php.net/

[40] Php storm. Visitado: 20/04/2022. [Online]. Available: https://www.jetbrains.com/phpstorm/

[41] Balsamiq wireframes. Visitado: 20/04/2022. [Online]. Available: https://balsamiq.com/wireframes/

[42] Git. Visitado: 20/04/2022. [Online]. Available: https://git-scm.com/

[43] Gitlab universidad de valladolid. Visitado: 20/04/2022. [Online]. Available: https://gitlab.inf.uva.es/

[44] Firefox. Visitado: 20/04/2022. [Online]. Available: https://www.mozilla.org/en-US/firefox/new/

[45] Telegram. Visitado: 20/04/2022. [Online]. Available: https://telegram.org/

[46] Trello. Visitado: 20/04/2022. [Online]. Available: https://trello.com

[47] Salario desarrollador de aplicaciones android. Visitado: 20/04/2022. [Online]. Available: https://es.indeed.com/career/android-developer/salaries

[48] Calculadora de sueldo. Visitado: 20/04/2022. [Online]. Available: https://es.indeed.com/career/android-developer/salaries

[49] Tarifa internet tim. Visitado: 20/04/2022. [Online]. Available: https://www.tim.it/fisso-e-mobile/fibra-e-adsl/fibra-internet-casa

[50] Precio electricidad italia. Visitado: 20/04/2022. [Online]. Available: https://luce-gas.it/guida/tariffe/kwh

[51] Macbook pro 15"2017. Visitado: 20/04/2022. [Online]. Available: https://prices.appleinsider.com/15-macbook-pro-with-touch-bar-mid-2017

[52] Xiaomi mi 10t lite. Visitado: 20/04/2022. [Online]. Available: https://amzn.eu/d/6PLPBqg

[53] Mapbox pricing. Visitado: 20/04/2022. [Online]. Available: https://www.mapbox.com/pricing/

[54] Google maps platform. Visitado: 07/04/2022. [Online]. Available: https://mapsplatform.google.com/

[55] Mapbox. Visitado: 07/04/2022. [Online]. Available: https://www.mapbox.com/

[56] How to develop a gps navigation app like waze in 2022 | process and tip. Visitado: 06/04/2022. [Online]. Available: https://www.thedroidsonroids.com/blog/how-to-develop-a-gps-navigation-app-like-waze#What_is_the_best_technology_stack_for_a_navigation_app

[57] Best map api for location-based services: Mapbox vs google maps vs openstreetmap. Visitado: 06/04/2022. [Online]. Available: https://topdevs.org/blog/choosing-the-best-mapping-services

[58] Google maps platform terms of use and privacy. Visitado: 06/04/2022. [Online]. Available: https://cloud.google.com/maps-platform/terms/

[59] Flutter. Visitado: 07/04/2022. [Online]. Available: https://flutter.dev/

[60] React native. Visitado: 07/04/2022. [Online]. Available: https://reactnative.dev/

[61] Flutter vs. react native. Visitado: 07/04/2022. [Online]. Available: https://blog.udemy.com/flutter-vs-react-native/

[62] Flutter package portal. Visitado: 07/04/2022. [Online]. Available: https://pub.dev/

[63] Flutter mapbox gl. Visitado: 7/04/2022. [Online]. Available: https://pub.dev/packages/mapbox_gl

[64] flutter mapbox navigation. Visitado: 7/04/2022. [Online]. Available: https://pub.dev/packages/flutter_mapbox_navigation

[65] Flutter mapbox example. Visitado: 08/04/2022. [Online]. Available: https://github.com/Imperial-lord/mapbox-flutter.git

[66] Flutter mapbox navigation issue. Visitado: 7/04/2022. [Online]. Available: https://github.com/eopeter/flutter_mapbox_navigation/issues/145

[67] Mapbox navigation android sdk examples. Visitado: 18/04/2022. [Online]. Available: https://github.com/mapbox/mapbox-navigation-android-examples

[68] Kotlin for android. Visitado: 8/04/2022. [Online]. Available: https://kotlinlang.org/docs/android-overview.html

[69] Mapa de movilidad en bicicleta en valladolid y entorno. Visitado: 19/04/2022. [Online]. Available: https://www.geocyl.com/mapa-movilidad-ciclista-valladolid/

[70] Django. Visitado: 21/04/2022. [Online]. Available: https://www.djangoproject.com/

[71] Node. Visitado: 21/04/2022. [Online]. Available: https://nodejs.org/es/about/

[72] Laravel vs nodejs. Visitado: 21/04/2022. [Online]. Available: https://www.simform.com/blog/laravel-vs-nodejs/

[73] Exrepss js. Visitado: 21/04/2022. [Online]. Available: http://expressjs.com/

[74] Postman colections. Visitado: 21/04/2022. [Online]. Available: https://www.postman.com/collection/

[75] Postman mocking a server. Visitado: 21/04/2022. [Online]. Available: https://learning.postman.com/docs/designing-and-developing-your-api/mocking-data/setting-up-mock/

[76] Pusher channels. Visitado: 21/04/2022. [Online]. Available: https://pusher.com/channels

[77] Furps. Visitado: 07/05/2022. [Online]. Available: https://en.wikipedia.org/wiki/FURPS

[78] Material design. Visitado: 07/05/2022. [Online]. Available: https://material.io/design

[79] Documentación api retobici. Visitado: 15/06/2022. [Online]. Available: https://documenter.getpostman.com/view/19868584/UzJERyFz#c30c046a-1c57-42d1-b6cf-1afe9c929824

[80] Android live data. Visitado: 31/05/2022. [Online]. Available: https://developer.android.com/reference/androidx/lifecycle/LiveData

[81] Object relational mapping. Visitado: 15/06/2022. [Online]. Available: https://www.ictshore.com/software-design/what-is-orm/

[82] Laravel migrations. Visitado: 15/06/2022. [Online]. Available: https://laravel.com/docs/9.x/migrations

[83] Laravel eloquent relationships. Visitado: 15/06/2022. [Online]. Available: https://laravel.com/docs/9.x/eloquent-relationships

[84] Laravel database seeders. Visitado: 15/06/2022. [Online]. Available: https://laravel.com/docs/9.x/seeding

[85] Recycler view android. Visitado: 8/04/2022. [Online]. Available: https://developer.android.com/guide/topics/ui/layout/recyclerview

[86] Android navigation component. Visitado: 31/05/2022. [Online]. Available: https://developer.android.com/guide/navigation/navigation-getting-started

[87] Laravel eloquent. Visitado: 10/04/2022. [Online]. Available: https://laravel.com/docs/9.x/eloquent

[88] Laravel controllers. Visitado: 10/04/2022. [Online]. Available: https://laravel.com/docs/9.x/controllers

[89] Clean architecture. Visitado: 28/05/2022. [Online]. Available: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

[90] Arquitectura mvvm. Visitado: 28/05/2022. [Online]. Available: https://medium.com/hongbeomi-dev/create-android-app-with-mvvm-pattern-simply-using-android-architecture-component-529d983eaabe

[91] Arquitectura mvp. Visitado: 28/05/2022. [Online]. Available: https://docs.microsoft.com/en-us/archive/msdn-magazine/2006/august/design-patterns-model-view-presenter

[92] Mvvm diagram. Visitado: 28/05/2022. [Online]. Available: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/mvvm

[93] Android guide to app architectura. Visitado: 30/05/2022. [Online]. Available: https://developer.android.com/topic/architecture

[94] Android ui layer. Visitado: 30/05/2022. [Online]. Available: https://developer.android.com/topic/architecture/ui-layer

[95] Android view model. Visitado: 30/05/2022. [Online]. Available: https://developer.android.com/topic/libraries/architecture/viewmodel

[96] Single activity architecture. Visitado: 31/05/2022. [Online]. Available: https://oozou.com/blog/reasons-to-use-android-single-activity-architecture-with-navigation-component-36

[97] Android activity. Visitado: 31/05/2022. [Online]. Available: https://developer.android.com/guide/components/activities/intro-activities

[98] Android fragments. Visitado: 31/05/2022. [Online]. Available: https://developer.android.com/guide/fragments

[99] Android domain layer. Visitado: 31/05/2022. [Online]. Available: https://developer.android.com/topic/architecture/domain-layer

[100] Android data layer. Visitado: 30/05/2022. [Online]. Available: https://developer.android.com/topic/architecture/data-layer

[101] Android clean architecture and mvvm. Visitado: 30/05/2022. [Online]. Available: https://github.com/android10/Android-CleanArchitecture-Kotlin

[102] Mvvm android architecture. Visitado: 30/05/2022. [Online]. Available: https://medium.com/hongbeomi-dev/create-android-app-with-mvvm-pattern-simply-using-android-architecture-component-529d983eaabe

[103] Mvc architecture. Visitado: 31/05/2022. [Online]. Available: https://www.w3schools.in/wp-content/uploads/2019/03/MVC-Architecture.png

[104] Dao pattern. Visitado: 02/06/2022. [Online]. Available: https://en.wikipedia.org/wiki/Data_access_object

[105] Dto pattern. Visitado: 02/06/2022. [Online]. Available: https://martinfowler.com/eaaCatalog/dataTransferObject.html

[106] Dao dto diagram. Visitado: 02/06/2022. [Online]. Available: https://www.oscarblancarteblog.com/2018/12/10/data-access-object-dao-pattern/

[107] Gson library. Visitado: 02/06/2022. [Online]. Available: https://github.com/google/gson

[108] Laravel eloquent api resources. Visitado: 02/06/2022. [Online]. Available: https://laravel.com/docs/9.x/eloquent-resources

[109] Observer pattern. Visitado: 31/05/2022. [Online]. Available: https://en.wikipedia.org/wiki/Observer_pattern

[110] Dagger hilt. Visitado: 8/04/2022. [Online]. Available: https://developer.android.com/training/dependency-injection/hilt-android

[111] Retrofit. Visitado: 8/04/2022. [Online]. Available: https://square.github.io/retrofit/

[112] Android data store. Visitado: 8/04/2022. [Online]. Available: https://developer.android.com/topic/libraries/architecture/datastore

[113] Phpunit. Visitado: 30/06/2022. [Online]. Available: https://phpunit.de/

[114] Junit. Visitado: 30/06/2022. [Online]. Available: https://junit.org/junit5/

[115] Android testing fundamentals. Visitado: 30/06/2022. [Online]. Available: https://developer.android.com/training/testing/fundamentals

[116] Android local testing. Visitado: 30/06/2022. [Online]. Available: https://developer.android.com/training/testing/local-tests

[117] Android instrumented testing. Visitado: 30/06/2022. [Online]. Available: https://developer.android.com/training/testing/instrumented-tests

[118] Laravel testing. Visitado: 30/06/2022. [Online]. Available: https://laravel.com/docs/9.x/testing

[119] Homebre macos. Visitado: 30/06/2022. [Online]. Available: https://brew.sh