



TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN ENXEÑARÍA DO SOFTWARE



Aplicación móbil de rutas a caballo

Estudiante: Antón Gendra Rodríguez

Dirección: María Noelia Barreira Rodríguez

A Coruña, junio de 2022.

Para mi familia, por apoyarme durante todo este tiempo

Agradecimientos

Para empezar, agradecer a mi madre por todo el apoyo hasta ahora durante todo este tiempo, que sobrepasa la veintena.

A mi hermano y mis primos, por estar a mi lado toda la vida.

Por supuesto, a mi tutora, María Noelia Barreira Rodríguez, por tener a bien ayudarme en este camino y contribuir con su tiempo a todo lo que he necesitado desinteresadamente.

Resumen

Este proyecto intenta proporcionar un servicio a todo aquel que tenga interés en realizar una actividad al aire libre como es la hípica y, además, no quiera llevarla a cabo en solitario.

La aplicación permite a la gestión de rutas y la coordinación de los asistentes. También soportará que distintos usuarios con relaciones cercanas puedan realizar un seguimiento sobre ellos para poder participar juntos en eventos futuros así como un sistema de seguridad para poder asistirse entre ellos. Además, los usuarios podrán gestionar sus propios caballos y registrar los datos de un recorrido para consultar a conveniencia.

El público objetivo son jinetes con una mínima base, pues se entiende que deben tener los conocimientos necesarios para participar de forma segura, que además tengan el interés de realizar esta actividad acompañados y compartir experiencias con otros usuarios. Aunque la aplicación también dispone de otras funcionalidades de las que podrá hacer uso un jinete solitario, la mayor carga funcional se centra en la coordinación de más de un usuario.

El proyecto surge de la necesidad de dar soporte a una necesidad que actualmente no es cubierta por la tecnología, dado que la mayor parte de aplicaciones hípicas se centran en el desarrollo deportivo e individual de los usuarios.

Para el desarrollar este proyecto se sigue una arquitectura cliente-servidor. Para la parte backend se usa el framework Django REST framework, mientras que Kotlin es la tecnología seleccionada para desarrollar la parte frontend nativa para Android. PostgreSQL será la tecnología elegida para almacenar los datos usados por la aplicación, integración de la cual se encarga el framework Django.

Abstract

This project tries to provide a service to anyone who is interested in carrying out a outdoor activity such as horse riding and, moreover, do not want to carry it out alone.

The application allows the management of routes and the coordination of attendees. It will also support that different users with close relationships can make a follow-up on them to participate together in future events. It will also include a security system to assist each other. In addition, users will be able to manage their own horses and record the data of a tour to consult at your convenience.

The target audience are riders with a minimum base, since they must have the necessary knowledge to participate safely, who are also interested in carrying out this activity accompanied and share experiences with other users. Although the application also has other features

that a rider can use, the main functional load is focused on the coordination of more than one user.

The project arises from the necessity to support a need that is currently not covered by technology, given that most equestrian applications are focused on the sporting and individual development of users.

To develop this project, a client-server architecture is followed. For the backend part, the Django REST framework is used, while Kotlin is the technology selected to develop the native frontend for Android. PostgreSQL will be the technology selected to store the data used by the application, which will be implemented using the Django framework.

Palabras clave:

- Android
- Mapa
- Rutas a caballo
- Rutas en grupo
- Django
- Kotlin
- Backend web
- Geolocalización

Keywords:

- Android
- Map
- Horse routes
- Group routes
- Django
- Kotlin
- Web backend
- Geolocalization

Índice general

1	Introducción	1
1.1	Sleipnir	3
1.2	Motivación	3
1.3	Objetivos	4
1.4	Estructura de la memoria	5
2	Contextualización	6
2.1	Análisis del entorno	6
2.1.1	Necesidades de un usuario	6
2.2	Situación tecnológica	7
2.2.1	Equilab	8
2.2.2	Rideable	8
2.2.3	Equisense	8
2.3	Mercado	11
3	Entorno de desarrollo	14
3.1	Lenguajes y frameworks	14
3.1.1	PostgreSQL	14
3.1.2	Python	14
3.1.3	Django REST framework	15
3.1.4	Kotlin	15
3.2	Tecnologías de apoyo	15
3.2.1	Sistema de control de versiones	15
3.2.2	IDE y editores de texto	16
3.2.3	Automatización de empaquetado	16
3.2.4	Postman	16

4	Metodología y planificación	17
4.1	Metodología de desarrollo	17
4.2	Planificación y seguimiento	19
5	Análisis	24
5.1	Requisitos	24
5.2	Arquitectura del sistema	28
5.3	Interfaz de usuario	29
6	Diseño	32
6.1	Arquitectura tecnológica del sistema	32
6.2	Diseño de la aplicación	32
6.2.1	Modelo conceptual de datos	33
6.2.2	Servicio	34
6.2.3	Frontend	47
7	Implementación	50
7.1	Backend	50
7.1.1	Serializadores	50
7.1.2	Seguridad	52
7.2	Frontend	53
7.2.1	Internacionalización	53
7.2.2	Paleta de colores	53
7.2.3	Conexión con el backend	53
7.2.4	Compartir ruta	54
7.2.5	Google Maps	56
7.2.6	Localización	56
7.2.7	Almacenamiento de configuración	58
8	Solución desarrollada	59
8.1	Usuarios	59
8.1.1	Creación de usuario	59
8.1.2	Acceso a la aplicación	59
8.1.3	Gestión de amigos	62
8.1.4	Gestión de caballos	62
8.1.5	Ubicación de observados	62
8.1.6	Rutas de amigos	65
8.2	Rutas	67
8.2.1	Ver rutas futuras	67

ÍNDICE GENERAL

8.2.2	Rutas pasadas	68
8.2.3	Registrar datos de ruta	71
8.2.4	Registros de rutas	71
9	Conclusiones	75
9.1	Trabajo futuro	76
	Lista de acrónimos	78
	Bibliografía	79

Índice de figuras

1.1	Dos de las razas más antiguas existentes. (a) Caballo Przewalskii. Presenta características de caballos salvajes antiguos. (b) Caballo Árabe. Con cola elevada y con una vértebra menos que las demás razas, característica que no pasó a las razas que contribuyó pese a ser una de las más usadas para la creación y mejora de otras.	1
1.2	Dos de las razas españolas. (a) Caballo Andaluz. De gran renombre mundial. (b) Caballo Pura Raza Galega.	2
2.1	Aplicación Equilab. Pantalla principal.	9
2.2	Aplicación Rideable. Pantalla principal.	10
2.3	Aplicación Equisense. Pantalla principal.	12
2.4	Aplicación Equisense. Posibilidades de registro de datos.	13
4.1	Diagrama de Gantt de las tareas de desarrollo.	19
4.2	Gráfico <i>burn-down</i> . <i>Sprint 1</i>	22
4.3	Gráfico <i>burn-down</i> . <i>Sprint 2</i>	23
4.4	Gráfico <i>burn-down</i> . <i>Sprint 3</i>	23
5.1	Arquitectura de la aplicación.	29
5.2	Flujo de pantallas.	31
6.1	Entidad relación.	35
6.2	Componentes del servicio.	38
6.3	Diagrama de clases del servicio.	39
6.4	Frontend. Diagrama de clases con los objetos del dominio.	47
6.5	Frontend. Diagrama de actividades que controlan la vista de la aplicación. . . .	49
7.1	Paleta de colores.	55

7.2	Gestión del código de una ruta. (a) Obtención del código. (b) Introducción del código.	55
8.1	Menú drawer.	60
8.2	Pantalla de registro.	61
8.3	Pantalla de acceso.	61
8.4	Pantalla de acceso.	62
8.5	Pantalla para ver los caballos del usuario.	63
8.6	Pantalla para añadir caballo.	64
8.7	Pantalla para ver observados.	65
8.8	Pantalla para ver rutas en las que un amigo está inscrito.	66
8.9	Pantalla para ver rutas futuras.	67
8.10	Pantalla para ver detalles de una ruta.	68
8.11	Pantalla para consultar chat de una ruta.	69
8.12	Pantalla para ver participantes de una ruta.	70
8.13	Pantalla de consulta de rutas pasadas. (a) Rutas a cincuenta kilómetros. (b) Rutas a una cantidad elevada de kilómetros, en concreto, treinta mil.	70
8.14	Pantalla para registrar ruta.	71
8.15	Pantalla para ver lista de registros.	72
8.16	Pantalla para ver detalles de un registro.	73
8.17	Pantalla para consultar gráfico de altitudes.	74

Índice de tablas

4.1	Costes del proyecto.	20
4.2	<i>Sprint 1. Sprint backlog.</i>	21
4.3	<i>Sprint 2. Sprint backlog.</i>	21
4.4	<i>Sprint 3. Sprint backlog.</i>	23
5.1	<i>Product backlog.</i> Conjunto de historias de usuario.	26
6.1	Diccionario de datos. Rider.	34
6.2	Diccionario de datos. Observers.	35
6.3	Diccionario de datos. Friends.	35
6.4	Diccionario de datos. Record.	36
6.5	Diccionario de datos. Route.	36
6.6	Diccionario de datos. Participants.	36
6.7	Diccionario de datos. Message.	36
6.8	Diccionario de datos. Point.	37
6.9	Diccionario de datos. Horse.	37

Capítulo 1

Introducción

Los caballos son animales con los que los seres humanos hemos convivido durante centenares de años, ayudándonos en la realización de multitud de tareas. A día de hoy, se conservan razas cuyas líneas genealógicas se remontan a miles de años atrás. Como ejemplo de estos, se mencionan las dos razas expuestas en la Figura 1.1.



(a)



(b)

Figura 1.1: Dos de las razas más antiguas existentes. (a) Caballo Przewalskii. Presenta características de caballos salvajes antiguos. (b) Caballo Árabe. Con cola elevada y con una vértebra menos que las demás razas, característica que no pasó a las razas que contribuyó pese a ser una de las más usadas para la creación y mejora de otras.

Nuestro propio país posee una cultura ecuestre muy rica, poseyendo una gran variedad de razas, entre ellas el [Pura Raza Española \(PRE\)](#), también llamado caballo Andaluz, que fue muy requerido por reyes y nobleza europea durante la Edad Media. Esta raza también es antigua en relación con la mayoría de las razas existentes hoy en día y colaboró en la creación y mejora de muchas otras. Sobresale en gran medida su papel como reproductor en América, pues la gran mayoría de razas en ese continente tiene un origen español, debido a la importación de estos animales por parte de los conquistadores, continente en el cual se habían extinguido estas criaturas. Otro símbolo de la cultura del país con respecto a estos impresionantes animales son las famosas y antiguas Caballerizas Reales de Córdoba.

Más localmente, en Galicia, también existió una gran cultura ecuestre. Nosotros mismos poseemos una de las numerosas razas autóctonas de España, el Pura Raza Galega, cuya principal característica es la existencia de un gen en ella que le permite andar de una forma típica de caballos de paso: el paso llano. Estos caballos son reconocidos por su participación en la fiesta folclórica “A Rapa das Bestas”. En la Figura 1.2 se ejemplifican los caballos a los que se está haciendo mención.



(a)



(b)

Figura 1.2: Dos de las razas españolas. (a) Caballo Andaluz. De gran renombre mundial. (b) Caballo Pura Raza Galega.

La compañía de estos impresionantes animales domésticos ha sido históricamente muy importante para las personas hasta la aparición de las tecnologías modernas, el punto de inflexión a partir del cual se produjo el desuso de los caballos en el día a día de los ciudadanos fue la puesta en uso del motor de vapor a gran escala. A partir de este momento, este nuevo invento comenzó a masificarse por su gran eficiencia y capacidad de producción, permitiendo realizar los trabajos que antes requerían del uso de fuerza animal de una forma más eficaz y menos costosa. Este hecho propició que los caballos fueran dejados de lado, que se notó mucho más notoriamente en razas especialmente concebidas para el trabajo. En países menos desarrollados, donde la revolución industrial llegaría más tarde, solo sería parcial, el desuso sería menos acusado, resultando en que hoy en día aún hay lugares que usan el trabajo animal de manera habitual.

La consecuencia de estos hechos fue la extinción de razas y que algunas de ellas entrasen en serio peligro de desaparecer. Por fortuna, surgieron distintas iniciativas para preservar o recuperar ciertas razas, como sucedió con el caso de los espectaculares caballos Frisones, muy apreciados para espectáculos y películas.

Actualmente, los caballos en los países desarrollados, donde la revolución industrial caló completamente, ya no se usa para labores, quizás una de las pocas excepciones sea Estados Unidos, donde en ciertas partes se sigue usando en la ganadería, más por motivos tradicionales que por eficiencia real.

En cambio, a día de hoy, su uso mayoritario es en competiciones hípicas. Salto, rodeo, exposición, carrera de barriles, doma clásica, etc, son algunas de las múltiples disciplinas donde el caballo es el principal foco de atención. Incluso existen países, de gran cultura ecuestre, que tienen sus propias prácticas tradicionales, como es el caso de España y sus caballos rejoneos, Estados Unidos y el rodeo o México y la charrería, entre otros.

Uno de los tipos de apuestas más populares tienen que ver con los deportes hípicos: las mundialmente conocidas carreras de caballos. Esta es una industria que mueve cifras de dinero muy grandes, el volumen de dinero entre apuestas de Australia, Francia, Hong Kong, Japón y Estados Unidos juntos ascendió hasta los 59.333.451.189 euros en 2018[1].

Otro uso actual de los caballos es con fines recreativos, sin ánimo de competir, y es su monta como pasatiempo, por mero gusto. Este, pese a no mover tanto dinero como en el ámbito más competitivo, es una faceta de la que disfruta una gran cantidad de personas que poseen uno de estos animales, o están en disposición de hacerse con uno mediante la contratación de un servicio. Este es el uso al que este proyecto trata de proveer servicio.

1.1 Sleipnir

Odín, rey de los dioses nórdicos, montaba un caballo de ocho patas que recibía el nombre de *Sleipnir*. Este conducía al dios a la batalla sobre su lomo. Los vikingos creían que un paso de este caballo creó el cañón de Ásbyrgi, al norte de Islandia, debido a su forma de casco (pezuña). Debido a la relevancia mitológica de estos dioses, y aún más en la época actual, dónde series y películas sobre vikingos están cobrando gran protagonismo. Esto está catapultando la cultura nórdica recientemente.

La importancia de estos mamíferos no es algo específico de la mitología nórdica. También la antigua civilización griega poseía referencias a ellos en su mitología. Es curioso que el caballo, siendo un animal terrestre, fuera de una de las criaturas características del dios Poseidón, dios de los mares y los océanos. No obstante, en esta cultura se creía en una criatura marina con forma de caballo, el hipocampo, mitad pez, mitad caballo.

1.2 Motivación

Las rutas a caballo son una actividad popular para familias o individuos que quieran pasar un tiempo relajante y ameno en la naturaleza. No obstante, las posibilidades son bastante reducidas hoy en día, dado que la hípica no es un deporte excesivamente extendido. Las opciones son, o contratar un servicio en una hípica para dar una vuelta programada por los instructores, o planificar una por cuenta propia con el reducido grupo de conocidos que posean caballos y estén dispuestos a acompañar al organizador en la actividad. Considerando

esto, esta aplicación está pensada para aquellos que no conozcan a una gran cantidad de personas que practiquen este deporte y puedan permitirse un ejemplar de estos animales, una posibilidad muy factible dado la poca popularidad de la que goza hoy en día, además de la gran carga económica que supone.

La facilidad para poseer internet y estar conectados grandes volúmenes de personas actualmente hace muy sencillo el cooperar entre individuos para realizar planes o compartir intereses mutuos. Por eso, para suplir la falta acompañantes conocidos, surge este proyecto, que se centra en gran medida en la sincronización de personas para realizar rutas en compañía.

Además, la preocupación de algunas personas sobre las incidencias que puedan ocurrir durante el trayecto también es un factor determinante a la hora de decidir si tomar parte en una travesía de varias horas de duración. Es algo sabido que los accidentes en estas circunstancias son algo relativamente común. Por esto, la aplicación contempla solventar esta preocupación, permitiendo así que los integrantes puedan disfrutar de la actividad con un sistema de respaldo en caso de que surja una situación de necesidad.

1.3 Objetivos

La principal prioridad de este proyecto es la de prestar un servicio de sincronización de usuarios que quieran realizar salidas a caballo en compañía. Para esto, se desarrollará una aplicación nativa en Android siguiendo una arquitectura cliente-servidor. Contará con un backend, que proporcionará una API REST al frontend, con la cual la vista interactuará para prestar servicio al jinete.

Para permitir un correcto desarrollo de la actividad a realizar se deberá suministrar al usuario toda la información relevante para llevarla a cabo, así como proporcionar un sistema que le permita crear nuevos eventos fácilmente o aprender de otros de los que haya participado anteriormente. Para cumplir con estas necesidades el proyecto se descompone en estos objetivos específicos:

- Gestión de rutas.
- Chat para la coordinación entre participantes.
- Recopilación de datos sobre un recorrido.
- Mostrar detalles sobre un recorrido llevado a cabo.
- Gestión de observadores.
- Monitorización de ubicación de los usuarios observados.
- Gestión de amigos.

- Consulta de actividades en las que participan amigos.
- Gestión de caballos.

1.4 Estructura de la memoria

En un documento la organización y exposición correcta de cada uno de los apartados es relevante. Para una redacción adecuada, es muy necesario organizarlos de la manera adecuada. Esta sección expondrá la estructura del propio documento. Esta memoria constará de un total de ocho capítulos, sin contar la introducción.

El primer capítulo mencionado en esta sección será el de *Contextualización*, dónde se expondrán las necesidades tecnológicas de un jinete, así como de qué facilidades cuenta en el sector a día de hoy.

El siguiente capítulo lleva el título de *Entorno de desarrollo*, en el cual se estudiarán los diferentes instrumentos tecnológicos de los cuales se hará uso para la implementación de la solución, así como los motivos de su elección.

Cada proyecto consiste en una serie de procesos estructurados que finalizarán en la construcción del software. Este conjunto de procesos es conocido como metodología y tendrá un capítulo dedicado a ello. Este capítulo se titula *Metodología y planificación* y en él se expondrá en qué consiste, por qué se ha seleccionado para el desarrollo de esta aplicación y su uso concreto.

En análisis es una etapa esencial de cada proyecto. Se estudiará en el capítulo *Análisis*. En él se recogerán todos los elementos estudiados para la adecuación de la solución a las necesidades del usuario.

El sexto capítulo es titulado *Diseño*. En este apartado se expondrán las principales decisiones de diseño relevantes para el proyecto.

Es imprescindible mencionar todos los detalles técnicos que puedan tener cierta importancia en la solución, a fin de clarificarlos y documentarlos para su fácil comprensión. Esto se llevará a cabo en el capítulo de *Implementación*, en el cual se comentarán detalles del desarrollo con gran detalle que puedan considerarse de interés.

En el penúltimo capítulo se expondrá el resultado del desarrollo. Se titula *Solución*, y en él se mostrará la aplicación final y las funcionalidades acabadas de las que el cliente podrá hacer uso.

Finalmente, para concluir esta enumeración, se menciona el capítulo *Conclusiones y trabajo futuro*. Es el cual se redactarán las opiniones finales de autor sobre el conjunto del proyecto, qué produjo mayor o menor satisfacción y qué conllevó más esfuerzo. Además también se estudiará posibilidades futuras de la aplicación y posibles cambios que podrá sufrir la aplicación final en labores de mantenimiento siguientes.

Contextualización

La mayor parte del interés del que goza la hípica se centra en el ámbito competitivo, mientras que, el uso recreativo no profesional no consta de las mismas herramientas y facilidades. Durante este apartado, se estudiará el entorno donde se sitúa esta práctica así como su situación en cuanto la tecnología y de qué servicios goza.

2.1 Análisis del entorno

En cada país suele haber un deporte que destaca más que el resto en popularidad, en el caso de España, es el fútbol, y en Estados Unidos, el fútbol americano, por ejemplo. Los deportes hípicos en general, han decaído en gran medida en todos estos años, pese a que hay países con gran cultura vaquera dónde siguen teniendo bastante importancia como puede ser el caso de Estados Unidos o México. No obstante, en la gran mayoría de países no son deportes mayoritarios ni muchísimo menos. Unido a su baja popularidad, los costes que conlleva apuntarse a unas instalaciones hípicas y tomar clases, así como tutelar un caballo y obtener el equipamiento requerido para llevar a cabo la actividad son muy grandes, resultando en ser una opción disponible para no tantas personas como sería adecuado para ganar popularidad.

Así pues, las personas que posean un caballo son contadas en cada municipio, haciendo bastante problemático el conocerse a entre ellas como para llevar cabo actividades como las que pretende fomentar este proyecto.

2.1.1 Necesidades de un usuario

Un jinete con ambiciones de disfrutar de rutas al aire libre en compañía de otros amantes del arte requiere de facilidades para poder realizar este ejercicio.

Generalmente, este tipo de personas usan estos caballos con fines competitivos, lo cual no es algo contemplado por este proyecto y, sin embargo, importante, dado que son estos jinetes

los que a la vez, por gusto, realizan estas excursiones lúdicas y que, por lo tanto, demandan el servicio ofrecido por esta aplicación.

En primer lugar, la necesidad de conocer a otros jinetes dispuestos a participar en el ejercicio es esencial. Mucha gente disfruta de compartir gustos con otras personas y poder llevar a cabo esa actividad común acompañados, convirtiéndolo en un proceso todavía más ameno. Desgraciadamente, no es una tarea sencilla para los amantes de lo ecuestre en estas coordenadas del mundo.

Por otro lado, es necesario que tengan la capacidad de coordinarse entre sí, con el fin de poder organizarse de manera que todos tengan a su disposición la información necesaria para llevar a cabo el plan.

Por otro lado, es muy importante disponer de datos del rendimiento de la actividad, posibilitando así, una mejor planificación de las actividades futuras o un mayor conocimiento sobre si el trabajo es efectuado de una manera correcta en relación a la salud de la montura del jinete.

Además, como es habitual en este tipo de ejercicios, la seguridad es un tema primordial. Es muy recurrente oír consejos dados a personas que se disponen a realizar eventos de este tipo como “informa a familiares o conocidos a dónde vas y cuanto durará la travesía” o “vete informando de tu localización periódicamente a alguien de confianza”, por ejemplo.

2.2 Situación tecnológica

Como se ha visto a lo largo de este capítulo, en el mundo equino, la mayor parte del interés y esfuerzo está invertido en el ámbito más competitivo. Esto no es diferente si estudiamos las diferentes soluciones tecnológicas que se ofrecen al público ecuestre. La mayoría de las aplicaciones orientadas a los caballos tienen como objetivo mantener un registro de la salud del animal y sus necesidades veterinarias, así como llevar la cuenta de los regímenes a los que se somete el caballo para lograr su excelencia técnica.

Siguiendo la línea anterior también abundan las aplicaciones que tienen como objetivo lograr mejorar el desempeño del animal en las diferentes disciplinas competitivas, con el fin de ir mejorando la actuación de tanto el jinete como el caballo y conseguir mayores logros.

El caso de las rutas a caballo, de las cuales disfrutaban habitualmente muchos jinetes, es distinto. No es fácil encontrar soluciones que traten de proporcionarle un servicio específico. Por supuesto, existen, pero normalmente más centradas en el desempeño que en el punto principal de esta aplicación, que es fomentar la posibilidad de realizar estas actividades grupalmente. Las aplicaciones destinadas al seguimiento de esta actividad están enfocadas en el desarrollo individual de la misma, en lugar de contemplar la posibilidad de realizarla acompañado con jinetes de intereses similares. Se pondrán como ejemplo tres de las aplicaciones

más completas que se han podido localizar: Equilab, Rideable y Equisense.

2.2.1 Equilab

Se trata de una aplicación mantenida por la empresa Equestrian Insights AB cuyo nombre completo es Equilab Equestrian Tracker [2], especialmente diseñada para llevar la cuenta de los entrenamientos realizados, así como del rendimiento de jinete y caballo en cada uno de ellos. Cuenta con múltiples funcionalidades, muchas de ellas, realmente interesantes. Definitivamente, su objetivo principal, que no es otro que el de mejorar el resultado competitivo del binomio jinete/equino, lo cumple.

Pese a dar soporte, y bastante completo, como se aprecia en la Figura 2.1, a una ruta a caballo como pretende dar este proyecto, no contempla, la posibilidad de realizar ninguna de estas actividades acompañado. Aun así, aún ofrece cierta interacción con otros jinetes, aunque bastante superficial, siendo un mero escaparate de los entrenamientos realizados por el usuario, la más importante funcionalidad social que ofrece.

De este producto, se debe destacar la funcionalidad de seguridad que ofrece en las rutas, muy completa, y, sin lugar a dudas, de gran ayuda para un jinete que decida realizar una actividad de este tipo en la naturaleza [3].

2.2.2 Rideable

Grey Pony Equestrian Lab [4] es la empresa creadora de la aplicación Rideable, mostrada en la Figura 2.2. Esta es, de nuevo, una aplicación completamente orientada al entrenamiento y la mejora competitiva del binomio. En esta ocasión está sumamente orientada a la programación y mejora de rendimiento en las diferentes disciplinas deportivas que el mundo equino tiene que ofrecer. Su principal característica es la gestión de sesiones de entrenamiento y la facilitación de documentación educativa.

Además, esta aplicación destaca en gran medida por la necesidad de invertir en ella para acceder a sus funcionalidades, teniendo que pagar una membresía para poder disfrutar del contenido que ofrece. Incluso contiene una pantalla de compras para obtener pagando material complementario para mejorar las sesiones de entrenamiento.

2.2.3 Equisense

Equisense, de Equisense Team [5], cuya pantalla principal puede observarse en la Figura 2.3, al contrario que la aplicación anterior, si tiene funcionalidades más acordes a las de este proyecto, no obstante, como todas las opciones anteriores, está orientada al entrenamiento competitivo. Aunque disponga de funcionalidades de localización GPS de las rutas realizadas,

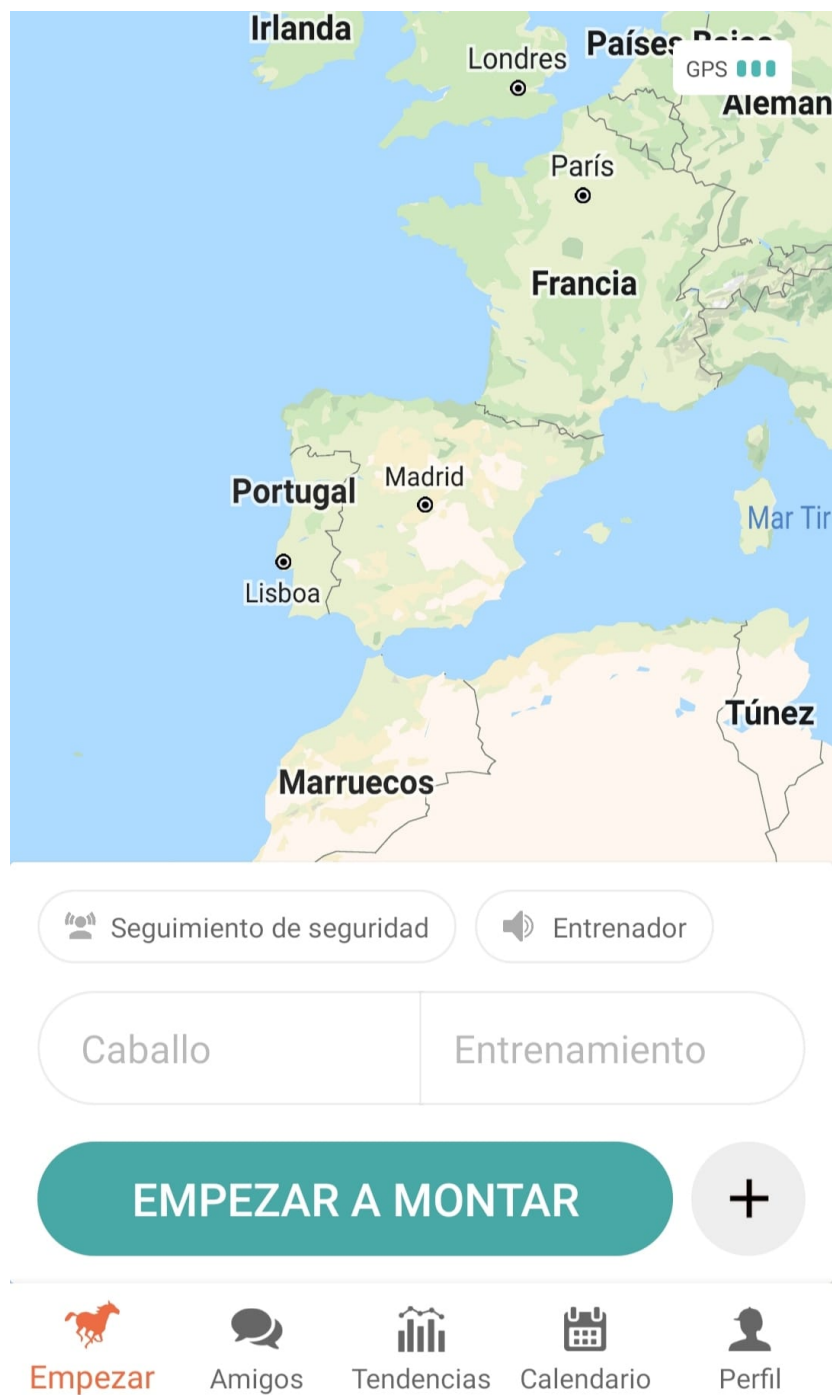


Figura 2.1: Aplicación Equilab. Pantalla principal.

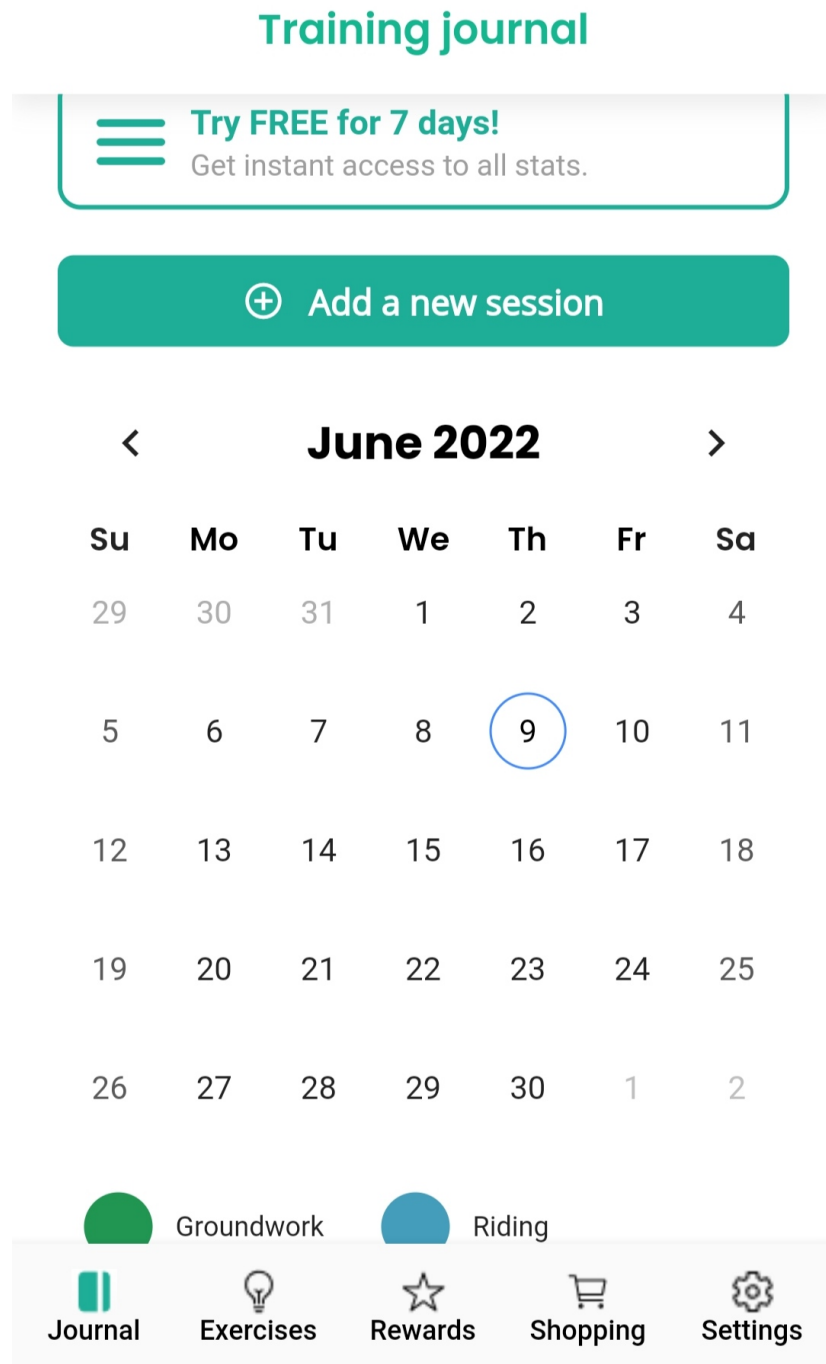


Figura 2.2: Aplicación Rideable. Pantalla principal.

la mayor carga funcional y objetivo de la aplicación es la de permitir mejorar el rendimiento del jinete y caballo en las diferentes competiciones que puedan interesar al usuario.

Esta aplicación tiene realmente funcionalidades muy destacables. Por ejemplo, es realmente útil para mejorar técnicas en particular que puedan estar causando problemas al jinete. Por otro lado, dispone de una amplia variedad de ejercicios específicos para diferentes situaciones concretas, incluso, a diferencia de muchas otras opciones, contempla la posibilidad de entrenar físicamente al propio jinete. Esto hace de esta opción, una aplicación en gran medida versátil.

Uno de los servicios más interesantes que ofrece es llevar más allá el registro del desempeño de un caballo. Esto es gracias a que Equisense Team tiene a disposición vía web [5] un servicio de comercio de productos equinos, entre ellos, un conjunto de sensores de medición para el animal, integrados con la aplicación, que pueden tomar datos muy precisos, como se puede observar en la Figura 2.4.

2.3 Mercado

La gran mayoría de proyectos se focalizan en el entrenamiento destinado a fomentar la eficacia en competición tanto de los jinetes como de los caballos. Por eso, esta aplicación trata de cumplir con las necesidades de la gente que trata de disfrutar del mundo ecuestre mediante actividades no destinadas a la competición, en este caso, el realizar rutas por la naturaleza. No obstante, sin focalizarse en ello, hay ciertas aplicaciones ya existentes, como por ejemplo, la mencionada Equilab, que de cierta manera cubren parcialmente este servicio. No obstante la solución propuesta en este proyecto tendrá como principal foco y objetivo este caso, asumiendo en él la mayor carga funcional.

Se destaca además, como principal elemento diferenciador de esta aplicación frente a las demás, el hincapié en la realización grupal de esta actividad, permitiendo al usuario colaborar con otros para disfrutar de los ejercicios ecuestres de manera conjunta.

Todo esto implica que el proyecto tiene como objetivo un servicio único y pretende cubrir las necesidades de parte de la población que no han sido cubiertas por la tecnología hasta ahora. Por supuesto, no es una tarea sencilla. Gran parte del éxito de un proyecto similar recae en las habilidades de la empresa que lo mantiene para fomentar su uso, realizar las adecuadas campañas publicitarias para dar a conocer el producto, y seguir proporcionando un servicio satisfactorio al usuario en todo momento. Esto implica conocer las necesidades del jinete objetivo mediante una buena identificación de incidencias y, en general, datos relevantes del desempeño de la aplicación. Esto permitiría llevar a cabo el mantenimiento y mejoras pertinentes que propicien hacer cada vez más atractivo el producto así como cubrir las necesidades del usuario, tanto presentes como futuras.

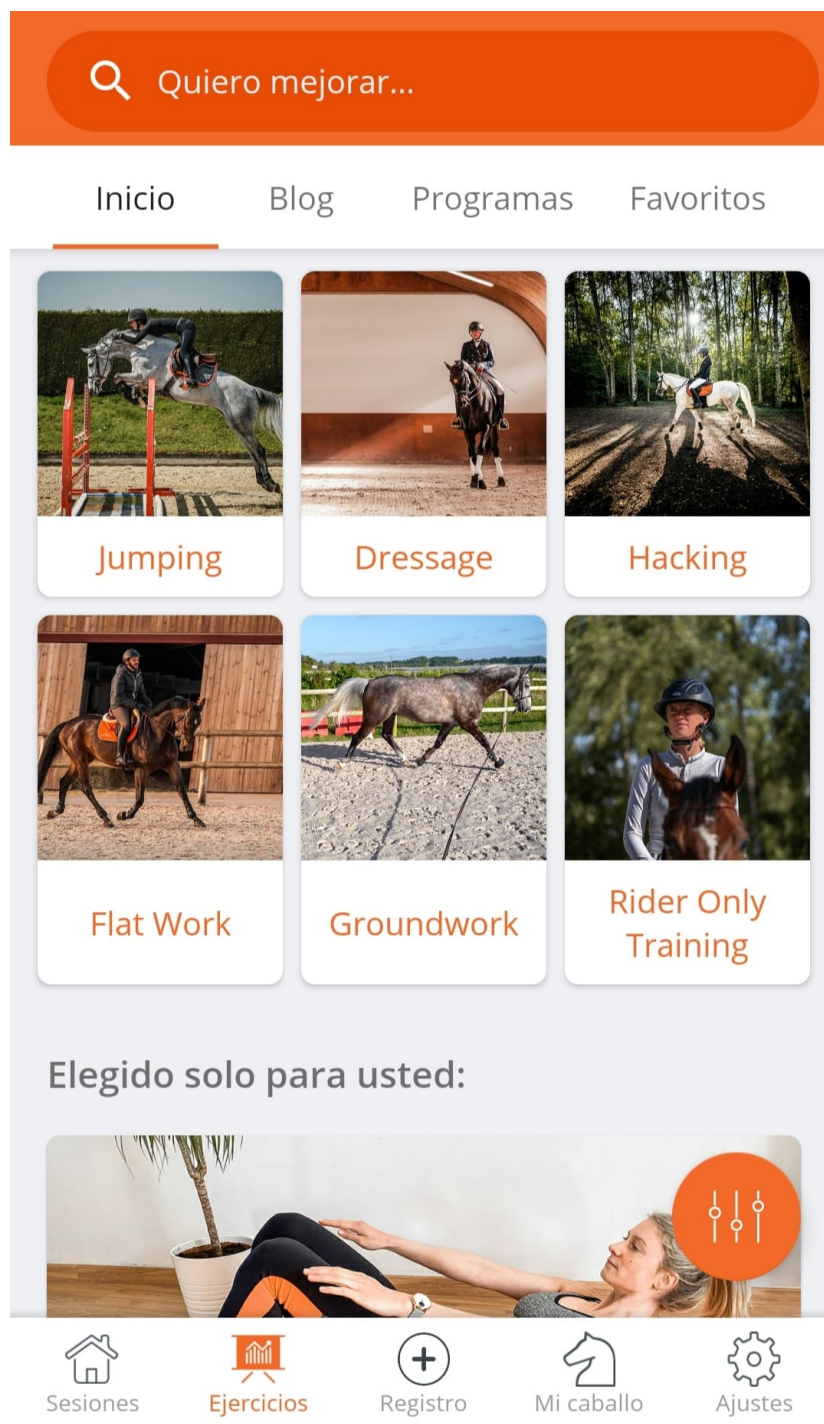


Figura 2.3: Aplicación Equisense. Pantalla principal.

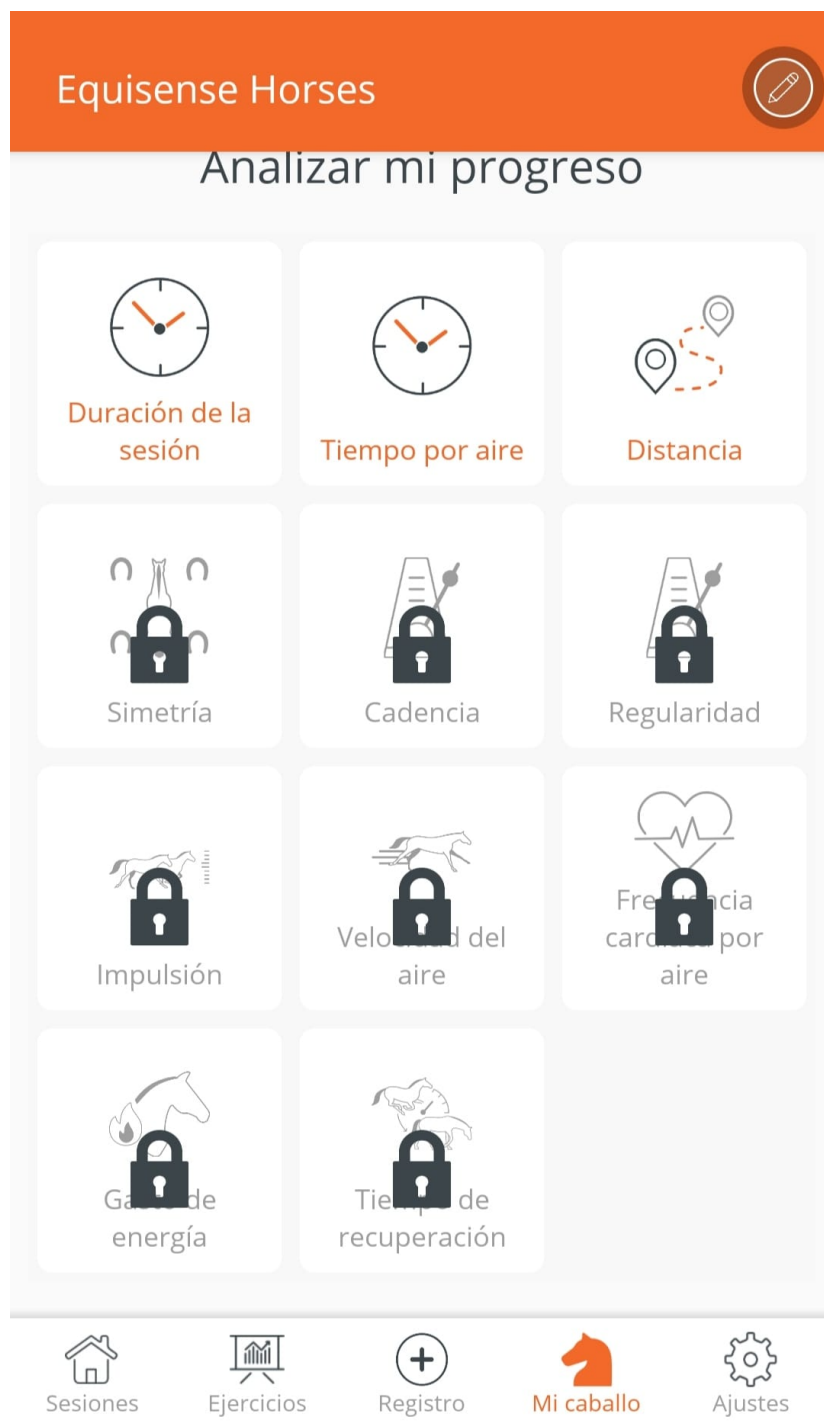


Figura 2.4: Aplicación Equisense. Posibilidades de registro de datos.

Entorno de desarrollo

ESTE capítulo tiene como objetivo informar sobre qué tecnologías se realizará el desarrollo de la aplicación así como de qué otras se harán uso para dar soporte al mismo. La elección de estas es importante. Primero, para poderse adecuar a los requisitos funcionales impuestos, dado que unas soluciones aportarán más facilidades que otras. Y, segundo, para conseguir adaptarse a otras necesidades del proyecto, tales como, por ejemplo, la fecha límite de entrega. Esto nos permitirá cumplir en mayor medida con los requisitos del cliente.

3.1 Lenguajes y frameworks

La exposición de las tecnologías en las que se basará este proyecto es el principal objetivo de este apartado. En él se expondrán las características que las hacen adecuadas para el desarrollo de la aplicación así como los problemas que pudieron derivar de su elección.

3.1.1 PostgreSQL

La tecnología para la construcción de bases de datos *PostgreSQL* [6] facilitará la creación y mantenimiento de la estructura de datos de la aplicación.

En este caso, la tecnología usada no es especialmente relevante desde el punto de vista del desarrollo mientras que el *framework* usado para el desarrollo posea las correspondientes librerías para poder interactuar con ella.

3.1.2 Python

Este es un lenguaje [7] cuya filosofía se focaliza en proporcionar un entorno de desarrollo ágil y sencillo para el programador, intentando evitar la escritura excesiva. Además, es realmente popular, y su uso crece cada vez más, en gran medida ocasionado por la importancia que cobra en el desarrollo de soluciones de inteligencia artificial y aprendizaje máquina.

Debido al corto rango de tiempo del que se dispone para la realización del proyecto, el carácter ágil de este lenguaje, así como la familiaridad del alumno con el mismo, hacen de esta una elección lógica para su uso en el *backend* de esta aplicación.

3.1.3 Django REST framework

Este marco de desarrollo goza de gran popularidad entre los usuarios de *Python*. Proporciona una estructura de ficheros y una simplicidad en el código que resultarán de mucha utilidad para el desarrollo del proyecto [8].

Proporcionará las funcionalidades para gestionar la base de datos desde el mismo marco, mientras ofrece una gran cantidad de opciones y utilidades para tratar los datos presentarlos al cliente como resulte oportuno.

3.1.4 Kotlin

El lenguaje *Kotlin* [9], goza de un gran reconocimiento popular referente a su utilidad en el desarrollo en *Android*. Esto es debido a que *emphGoogle* lo catalogó como el lenguaje estándar para la implementación de soluciones en dispositivos móviles.

Por este motivo se ha seleccionado como la opción para desarrollar el *frontend* de la aplicación. Dado que el alumno, en general, no posee una relevante experiencia en el desarrollo *Android*, utilizar la tecnología estándar sería la opción más simple y adecuada.

3.2 Tecnologías de apoyo

Esta sección se centrará en la exposición las herramientas usadas para apoyar la implementación del proyecto, es decir, no se desarrollará con ellas la aplicación como tal, sino que servirán para agilizar y simplificar el proceso. Esto resulta muy beneficioso para llevar a cabo un desarrollo adecuado y bien estructurado. Tanto es así, que es algo completamente esencial en el mundo laboral, siendo un punto muy crítico el correcto uso de estos instrumentos.

3.2.1 Sistema de control de versiones

Como sistema de control de versiones para el proyecto se usará *Git* [10], que se trata de una herramienta muy extendida en el mercado. Es muy completa y proporciona todas las utilidades que un proyecto de estas características podría necesitar. Gracias a este software, se dispondrá de un repositorio ¹ en Github donde se irán almacenando los incrementos de la aplicación y nos permitirá la gestión de las diferentes versiones del código.

¹ <https://github.com/gen-4/TFG-Sleipnir.git>

3.2.2 IDE y editores de texto

Un editor de texto al que el trabajador esté acostumbrado puede ayudar en gran medida a propiciar un ambiente de trabajo más cómodo, además de aumentar la productividad. Además, trabajar con un software que presente una variedad de funcionalidades útiles para la situación, ayudará en gran medida a mejorar la rapidez de construcción del proyecto.

En el caso de esta aplicación, se usará el editor de texto *Visual Studio Code* [11] para realizar la implementación del backend de la aplicación. Al estar el alumno muy acostumbrado a este entorno, y ser Python ya de por sí, un lenguaje de muy ágil escritura donde un IDE con funcionalidad muy potente no es requerido, este se convierte en la opción perfecta.

Por otro lado, para la construcción de la vista de la aplicación, se usará *Android Studio* [12], una herramienta completamente desarrollada con el objetivo de servir al propósito del desarrollo nativo en Android. Dado que esto es exactamente lo que se pretende para este proyecto, las utilidades que se incluyen en esta herramienta resultarán de gran ayuda para agilizar el proceso de desarrollo.

3.2.3 Automatización de empaquetado

Este tipo de herramientas permite la gestión de los ficheros de un proyecto. Permite al desarrollador acordar unas directrices que el software tendrá en cuenta para la construcción automática de la solución a desarrollar.

En este proyecto, ese papel recaerá sobre *Gradle* [13], configurado automáticamente por Android Studio al generar un proyecto Kotlin.

3.2.4 Postman

Esta herramienta [14] es ampliamente usada para probar las conexiones entre diferentes aplicaciones. Gracias a ella, será posible probar los endpoints de los que dispondrá el API a construir a medida que se vayan desarrollando.

Es un software que resulta muy sencillo de usar, y, desde luego, proporciona todas las necesidades que se puedan llegar a requerir en este proyecto.

Metodología y planificación

EL foco de este capítulo consiste en la exposición de la metodología propuesta para el desarrollo del proyecto, así como de los motivos en los que se basa su elección. También se explicarán las técnicas concretas que fueron empleadas durante la implementación de la aplicación así de herramientas que pudieron ayudar con su empleo.

4.1 Metodología de desarrollo

La metodología usada en este proyecto es la muy conocida metodología *SCRUM* [15], de reciente desarrollo. Se trata de una metodología completamente basada en el Manifiesto Ágil, haciendo de esta, una de las metodologías ágiles modernas que tiene como principal característica el desarrollo iterativo del software.

Al tratarse de un proyecto realizado por un equipo pequeño y, especialmente, al no disponer de una gran cantidad de tiempo para su desarrollo, el uso de una metodología ágil es una decisión muy coherente con el contexto. Esto permite la realización del proyecto de una manera más veloz pero sin perder calidad en el proceso. Además, al hacer mucho hincapié en proporcionar valor al cliente en cada iteración, ayuda en gran medida a priorizar ciertas funcionalidades de la aplicación.

La principal unidad funcional en la que *SCRUM* basa el desarrollo es la historia de usuario. Esta es una concepción de cada fragmento de software que aporta valor al usuario componiendo el producto final. Esto implica que cada una de estas historias es entendible e identificable para el cliente. Es decir, que es capaz de comprender el valor que le aporta sin tener el cliente conocimientos específicos de informática.

Cada una de las iteraciones de las que consta esta metodología es conocida como *sprint*. En cada uno de estos *sprints*, se desarrollará una parte de la aplicación que aporte valor al usuario. Esta categorización se debe a la jerarquía asignada a cada historia de usuario por parte de los desarrolladores.

Los principales roles que tomarán los implicados en este proyecto de acuerdo con la metodología *SCRUM* son los siguientes:

- **Product Owner** (Propietario del Producto): Se trata de un responsable que posee el conocimiento sobre los intereses de la empresa. Será el nexo de unión entre el equipo de desarrollo y el cliente, transmitiendo las necesidades y inquietudes del cliente al equipo de desarrollo. Puede ser un miembro de la empresa cliente con conocimientos técnicos o podría ser un miembro de la empresa contratada que hay adquirido conocimiento sobre la empresa cliente y sus necesidades. En este caso, este rol es representado por el tutor.
- **SCRUM Master**: Sobre este rol recae la responsabilidad de que la metodología sea correctamente empleado durante todo el proceso. Además, ejerce de moderador y responsable durante las diferentes reuniones ocurridas en el desarrollo. En este proyecto el tutor ejercerá de *SCRUM Master*, asegurándose de que el proceso se realice adecuadamente.
- **Development Team** (Equipo de Desarrollo): Este es el conjunto de desarrolladores que se ocuparán de la implementación de la aplicación. Tendrán gran capacidad de decisión en el proyecto y velarán por cumplir con las necesidades del cliente siempre que sea posible. En un proyecto *SCRUM*, típicamente, cada equipo está conformado de pocos miembros, siendo estos capaces de realizar cualquier actividad requerida. Este proyecto contará con un único equipo de un solo miembro, el alumno.

A continuación, se definen los principales documentos que juegan un factor relevante en la aplicación de la metodología:

- **Product backlog** (pila de producto): Este es un documento que contiene el conjunto de las historias de usuario requeridas por el cliente. Es mantenido por el *Product Owner* y puede ser objeto de modificaciones futuras por parte del mismo.
- **Sprint backlog** (pila de *sprint*): Esta lista no puede tener mayor longitud que el *product backlog*, dado que es creada a partir de este. Su principal objetivo es identificar las historias de usuario a implementar en un *sprint* concreto. Este backlog es mantenido por el *equipo de desarrollo* y la adición de historias se basa en la prioridad de las mismas.

Antes de cada *sprint* se realiza una reunión llamada *sprint planning* en la cual el equipo de desarrollo decide qué historias de usuario del *product backlog* se introducen en el *sprint backlog* de esa iteración. Además, los miembros del equipo estiman la duración de las tareas mediante técnicas heurísticas y se organizan para su desarrollo.

Por otro lado, al finalizar cada *sprint*, el equipo se vuelve a reunir, en la reunión conocida como *sprint review*, para poner en común lo que se ha conseguido avanzar en el proyecto, así como identificar posibles fallos y posibilidades de mejora en el proceso seguido.

4.2 Planificación y seguimiento

El desarrollo de un proyecto software, o de cualquier otro sector, consiste en la sucesión de múltiples tareas. En nuestro caso, estas tareas son:

- **Análisis:** En en cual se recogen los requisitos y necesidades del cliente, con el fin de conocer mejor el producto a desarrollar y contar con la información necesaria para diseñar un sistema acorde.
- **Diseño:** En la fase de diseño, se deciden los pormenores de la aplicación. Esta es una fase importante en el desarrollo dado que influye en la eficiencia e integración de la aplicación.
- **Sprint planning:** Al principio de cada iteración se llevará a cabo esta reunión para decir que habrá que realizar y asegurarse que todo el mundo tenga claro su tarea.
- **Implementación:** Cada *sprint* constará de una fase de implementación en la cual el equipo desarrollará las historias de usuario que se les fueron asignadas en la reunión de la iteración.
- **Sprint Review:** Al final de cada *sprint*, el equipo se volverá a reunir para acordar aspectos de mejora en el proceso.

La interacción entre estas tareas, contando con que el proyecto estará compuesto por tres *sprints*, se puede observar en la Figura 4.1.

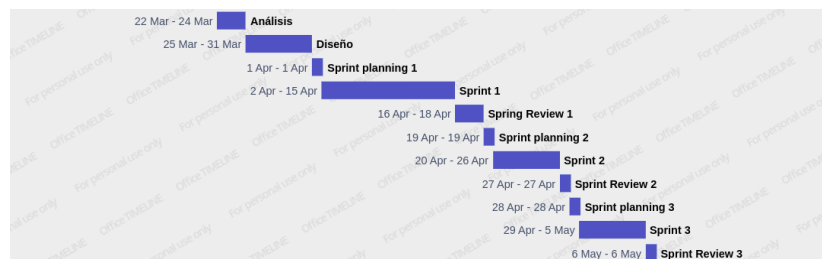


Figura 4.1: Diagrama de Gantt de las tareas de desarrollo.

El proceso estará dividido en tres *sprints*. En cada uno de ellos se implementarán varias historias de usuario intentando proporcionar valor al cliente. Esto implica que historias con una prioridad mayor tenderán a realizarse en los primeros *sprints*, disminuyendo en prioridad cuanto más tardía sea la iteración.

En primer sprint se compone de los aspectos más funcionales más prioritarios. Estos son los referentes a la gestión de rutas y usuarios. La gestión de observadores y caballos fue el

apartado que mayor carga funcional supone del segundo sprint. Por último, el tercer sprint se compone principalmente de funcionalidades de baja prioridad. Pudiendo el usuario prescindir de ellas y aun así poder disfrutar de una aplicación muy completa. Las funcionalidades que forman parte de este sprint son la gestión de amigos y la consulta de rutas de formas alternativas.

Un factor de planificación de gran relevancia son los costes de proyecto. Una adecuada concordancia entre el tiempo del proyecto, el esfuerzo y los costes supone la diferencia entre un desarrollo adecuado y uno inadecuado. El equipo de desarrollo cuenta con un integrante. El desarrollador cobrará un sueldo alrededor de la media de un informático en España, 17 euros la hora. En el coste estimado, se añaden 10 horas por parte de la investigación inicial de las tecnologías y 40 por parte de la memoria final. Para el cálculo del coste real, se tendrán en cuenta, adicionalmente al tiempo de desarrollo, 14 horas de estudio de las herramientas inicial, antes de proceder al desarrollo, y 42 horas de la realización de esta memoria. En la Tabla 4.1 se muestran tanto los costes estimados del proyecto como los costes reales:

Tabla 4.1: Costes del proyecto.

Tipo	Horas	Coste
Estimado	137 horas	2329 euros
Real	158,18 horas	2689,06 euros

Durante cada una de las reuniones de *sprint* se estimaron los tiempos que llevaría desarrollar cada una de las funcionalidades de cada *sprint*. Así mismo, a medida que se iban completando se registró la duración real de cada una. En las siguientes tablas se podrán observar los datos mencionados. El primer *sprint* corresponderá con la Tabla 4.2, el segundo con la Tabla 4.3 y el último con la Tabla 4.4. La columna *Prioridad* en las tablas refleja la importancia de cada funcionalidad de cara al usuario, los valores introducidos se encuentran en el rango de entre 1 y 5, siendo el 5 el valor que indicaría mayor prioridad. Tanto la estimación como la duración real están referenciados en horas.

La primera iteración sufrió una gran desviación con respecto a lo estimado en cada funcionalidad. Esto es debido a que durante esta primera etapa ocurrió la fase de estudio y asimilación de las herramientas usadas. Otra causa de esta variabilidad es la aún escasa familiaridad con la propia aplicación, teniendo que acostumbrarse a la complejidad real que presenta el proyecto. Puede observarse en las tablas como las estimaciones van resultando más adecuadas a medida que avanza el desarrollo.

Aun así, al tratar de estimarse el proyecto siguiendo números redondos, causando problemas en la estimación de funcionalidades muy simples. Como observamos en la Tabla 4.4, muchas estimaciones erraron por lo alto al estimar por lo mínimo, 1 hora.

Tabla 4.2: *Sprint 1. Sprint backlog.*

Funcionalidad	Prioridad	Estimación	Duración real
Login/Signup	5	6 horas	14,5 horas
Crear rutas	5	8 horas	17,5 horas
Unirse a ruta	5	7 horas	6 horas
Ver rutas	5	6 horas	7,25 horas
Abandonar ruta	4	5 horas	1 hora
Recabar información de ruta	5	10 horas	9,25 horas
Ver historial de rutas	4	6 horas	4,75 horas
Ver información de ruta realizada	4	5 horas	1,2 horas

Tabla 4.3: *Sprint 2. Sprint backlog.*

Funcionalidad	Prioridad	Estimación	Duración real
Ver chat de ruta	3	5 horas	7,5 horas
Postear mensaje en chat	3	3 horas	0,5 horas
Ver observadores	3	2 horas	3,8 horas
Agregar observador	3	2 horas	0,75 horas
Eliminar observador	3	1 hora	0,6 hora
Ver ubicación de observados	3	4 horas	4,25 horas
Ver caballos	2	2 horas	3,25 horas
Ver detalle de caballo	2	1 hora	1 hora
Agregar caballo	2	1 hora	7,25 horas
Eliminar caballo	2	1 hora	0,1 horas
Ver participantes y caballos	3	2 horas	1,75 horas

Durante cada sprint, para visualizar mejor el desempeño del equipo de desarrollo, se crea un gráfico de *burn-down*. Este es actualizado diariamente y tiene como objetivo mostrar cómo se va acercando el equipo al objetivo de la iteración. El eje Y corresponde al esfuerzo, en este caso, medido a partir de la carga funcional terminada. El eje de las X corresponde al tiempo, medido en días. El gráfico del primer sprint se corresponde con la Figura 4.2, el del segundo sprint, con la Figura 4.3, y, por último, el tercer incremento es representado por la Figura 4.4.

La utilidad de los gráficos de *burn-down* es medir la diferencia con respecto a una situación ideal. Esta situación es aquella en la cual el esfuerzo se mantiene constante a lo largo del tiempo. Es decir, que se mantiene una línea de pendiente negativa perfecta. Como se observa, no es el caso de ninguno de estos gráficos, pero tampoco se espera. Esto es porque es una situación ideal, y por tanto, irreal. No obstante sí que se intenta asemejarse a ella. Como podemos observar, el primer gráfico, la Figura 4.2, se aleja más que los dos restantes del gráfico ideal debido a tener el alumno que acostumbrarse a las tecnologías usadas.

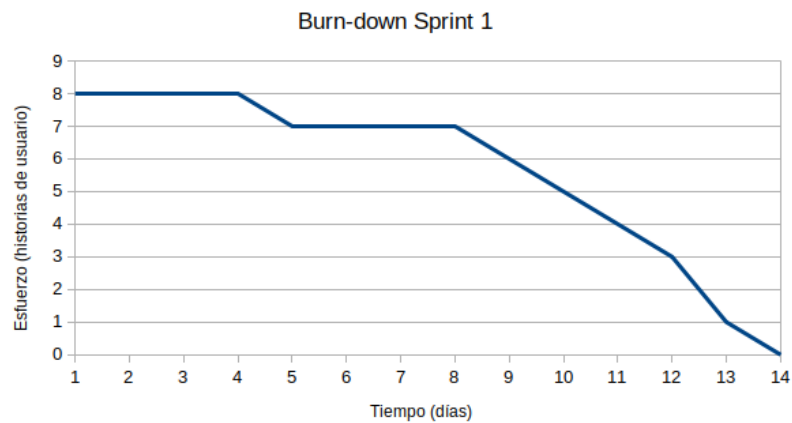


Figura 4.2: Gráfico *burn-down*. *Sprint 1*.

Tabla 4.4: *Sprint 3. Sprint backlog.*

Funcionalidad	Prioridad	Estimación	Duración real
Historial de rutas por cercanía	3	2 horas	1,25 horas
Crear ruta repetida	1	1 hora	0,5 horas
Compartir ruta por código	2	2 horas	7 horas
Agregar conocido	2	1 hora	0,16 horas
Eliminar conocido	2	1 hora	0,16 horas
Ver conocidos	2	1 hora	0,16 horas
Ver rutas de conocidos	2	2 horas	0,75 horas

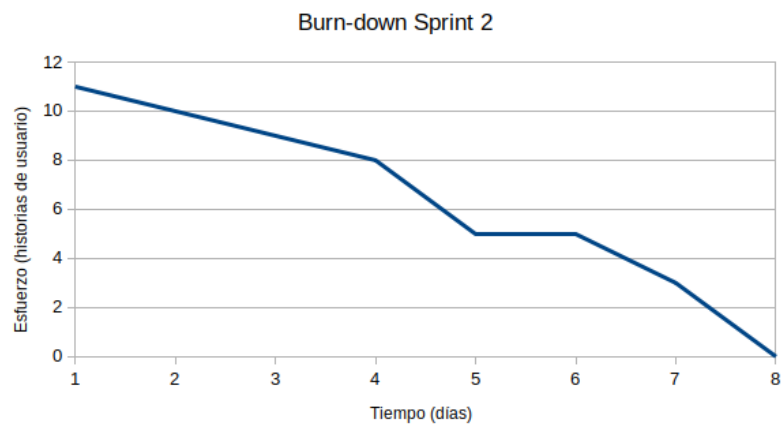


Figura 4.3: Gráfico *burn-down*. *Sprint 2.*

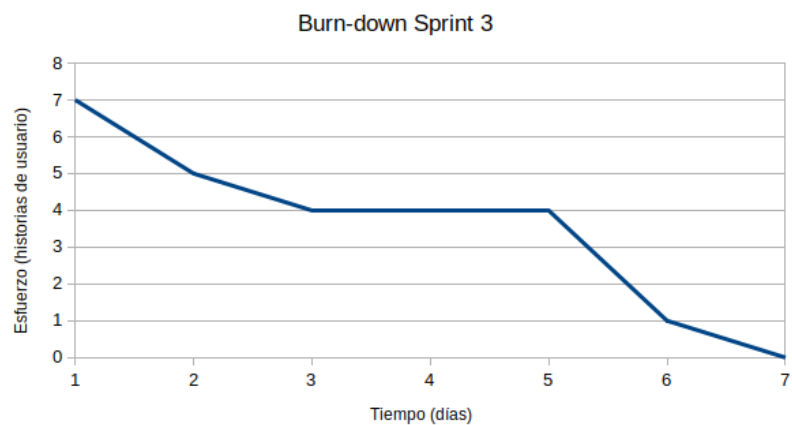


Figura 4.4: Gráfico *burn-down*. *Sprint 3.*

Capítulo 5

Análisis

CUALQUIER proyecto software debería constar de su correspondiente fase de análisis. Esta parte del desarrollo resulta completamente imprescindible para identificar las necesidades de la aplicación. De ella se consiguen los primeros productos del proceso de creación de la aplicación.

Normalmente este proceso se lleva a cabo mediante el estudio de la empresa y su modelo negocio por parte de uno o varios analistas, que se trasladan a la empresa cliente para lograr recabar esta información. También se llevan a cabo reuniones para consultar al cliente sobre sus necesidades.

En un proyecto *SCRUM*, este papel lo lleva a cabo el *product owner*, que concentra todo el conocimiento del cliente junto con nociones técnicas de informática.

5.1 Requisitos

Este proyecto también comenzó con su debida recopilación de requisitos. En esta sección se dispondrán los diferentes requisitos divididos en su dos variantes, funcionales y no funcionales. Además, se expondrá el producto creado a partir de ellos, el *product backlog*.

El conjunto de requisitos no funcionales de este proyecto son:

- Seguridad con respecto a la identificación del usuario (1).
- Servicio centralizado. Capacidad de gestionar los datos desde un único punto (2).

En cuanto a los requisitos funcionales, son los siguientes:

- Crear usuario (3).
- Poder loggarse (4).
- Crear ruta (5).

- Unirse y abandonar ruta (6).
- Cualquier usuario debe poder la información de cada ruta (7).
- El creador de la ruta debe poder limitar los participantes a su juicio (8).
- Después de crear una ruta, no se podrá eliminar, dado que otros usuarios ya cuentan con su celebración (9).
- Existirá un chat en cada ruta para poder preguntar las dudas que le puedan surgir a algún usuario (10).
- En el chat, el creador debe estar identificado, para que cualquier otro usuario pueda saber quién tiene mayor credibilidad sobre la ruta en cuestión (11).
- Cualquier usuario debe poder gestionar sus propios caballos (12).
- Al crear o unirse a una ruta, el usuario deberá introducir con qué caballo participará, a fin de que otros interesados puedan saber si su propio animal no es compatible con el resto (13).
- El usuario podrá registrar los datos de cualquier ruta que haga, sin tener que ser esta ninguna ruta de la aplicación (14).
- El usuario podrá consultar los datos de cualquier ruta de la cual haya recogido datos (15).
- Un usuario podrá acceder a rutas que ya se hayan celebrado, filtradas por cercanía, a fin de poder realizarlas de nuevo en caso de que no haya tenido una idea nueva (16).
- Un usuario podrá gestionar sus amigos. Cuando un usuario agrega a otro, el otro también lo tendrá como amigo (17).
- Un usuario podrá seleccionar a otros para que sepan su última ubicación mientras este estuviera grabando un recorrido. El mismo usuario, en cambio, no podrá consultar la ubicación del otro a menos que éste lo agregue también (18).
- El usuario podrá conocer todas las últimas ubicaciones de los jinetes que lo hayan seleccionado como observador (19).
- El usuario podrá saber qué carreras son en las que hay inscrito un amigo (20).
- Un usuario podrá compartir un código para que otro pueda acceder fácilmente a la información de una ruta (21).

- Un usuario podrá acceder a una ruta mediante la introducción de un código (22).

A partir de todos los requisitos, se elabora el *product backlog*, que comprende todas las historias de usuario que se desarrollarán en el conjunto de los *sprints*. Este puede consultarse en la Tabla 5.1.

Tabla 5.1: *Product backlog*. Conjunto de historias de usuario.

Referencia	Requisitos	Funcionalidad	Prioridad	Historia de usuario
1	1, 3, 4	Login/Signup	5	Como usuario, quiero poder acceder y registrarme para poder hacer uso de la aplicación y garantizar seguridad.
2	5	Crear rutas	5	Como usuario creador, quiero poder crear nuevas rutas para que otros puedan unirse.
3	6	Unirse a ruta	5	Como usuario, quiero poder unirme a rutas para poder realizar la actividad junto a otros usuarios.
4	7	Ver rutas	5	Como usuario, quiero tener acceso a las rutas creadas para poder unirme en caso de conveniencia.
5	6	Abandonar ruta	4	Como usuario, quiero poder abandonar una ruta para dejar claro a los participantes que no contarán conmigo.
6	14	Recabar información de ruta	5	Como usuario, quiero poder recoger información para poder consultarla en el futuro.
7	15	Ver historial de rutas	4	Como usuario, quiero poder ver las rutas de las que tomé información para poder ver la información de la que más me convenga.
8	15	Ver información de ruta realizada	4	Como usuario, quiero poder ver la información de una ruta realizada para poder valorar mi desempeño.
9	10	Ver chat de ruta	3	Como usuario, quiero poder consultar preguntas y respuestas sobre una ruta.

....(continúa en la página siguiente)....

Tabla 5.1 – (viene de la página anterior)

Referencia	Requisitos	Funcionalidad	Prioridad	Historia de usuario
10	10	Postear mensaje en chat	3	Como usuario, quiero poder escribir mensajes en el chat para poder preguntar dudas o solucionárselas a otro.
11	18	Ver observadores	3	Como usuario, quiero ver los observadores que tengo para poder gestionarlos.
12	18	Agregar observador	3	Como usuario, quiero añadir un nuevo observador para que supervise mi seguridad.
13	18	Eliminar observador	3	Como usuario, quiero eliminar un observador para que no pueda consultar mi estado en el futuro.
14	1, 19	Ver ubicación de observados	3	Como usuario observador, quiero ver la ubicación de mis observados para asegurarme de que se encuentran seguros.
15	12	Ver caballos	2	Como usuario, quiero ver mis caballos para poder gestionarlos.
16	12, 7, 13	Ver detalle de caballo	2	Como usuario, quiero ver el detalle de mis caballos y de otros usuarios para poder consultar la compatibilidad entre ambos.
17	12	Agregar caballo	2	Como usuario, quiero añadir un caballo para poder usarlo en una ruta.
18	12	Eliminar caballo	2	Como usuario, quiero eliminar un caballo para no tenerlo registrado en la aplicación si sé que no lo voy a montar.
19	7, 13	Ver participantes y caballos	3	Como usuario participante, quiero ver los caballos de cada participante para poder ver si podría haber algún problema con mi propia montura.
20	16	Historial de rutas por cercanía	3	Como usuario, quiero poder ver un historial de rutas pasadas cercanas para crear rutas futuras en caso de que no se me ocurra ninguna nueva.

....(continúa en la página siguiente)....

Tabla 5.1 – (viene de la página anterior)

Referencia	Requisitos	Funcionalidad	Prioridad	Historia de usuario
21	16	Crear ruta repetida	1	Como usuario creador, quiero poder volver a poner disponible una ruta pasada para poder añadir una nueva actividad en caso de que no se me ocurra ninguna novedosa.
22	21, 22	Compartir ruta por código	2	Como usuario, quiero poder copiar un código para que un amigo pueda acceder rápidamente a una ruta.
23	17	Agregar conocido	2	Como usuario, quiero poder agregar amigos para poder interactuar con ellos en la aplicación.
24	17	Eliminar conocido	2	Como usuario, quiero poder eliminar un amigo para no recibir más información sobre él.
25	17	Ver conocidos	2	Como usuario, quiero ver los amigos agregados para poder gestionarlos.
26	20	Ver rutas de conocidos	2	Como usuario, quiero ver las rutas en las que están inscritos mis amigos para poder participar con ellos en caso de que me interese.

5.2 Arquitectura del sistema

La arquitectura de una aplicación afecta en gran medida el rendimiento de la misma. Seleccionar la arquitectura adecuada según el contexto y el uso que se le va a dar se convierte en un factor crítico en un proyecto.

En este caso, se trata de un proyecto que pretende proporcionar un servicio estándar. La mayoría de proyectos de características similares siguen una arquitectura cliente-servidor y es la seguida por esta aplicación, como se observa en la Figura 5.1. El cliente se haya instalado en el terminal *Android* de cada usuario mientras que el servidor será accedido remotamente por conexión internet. Así mismo, se trata de un servidor centralizado donde todos los datos están almacenados en un único lugar. Debido a que no se espera un gran flujo de usuarios ni que el tamaño del proyecto vaya a variar en gran medida en el futuro se considera la forma

adecuada de implementarlo.

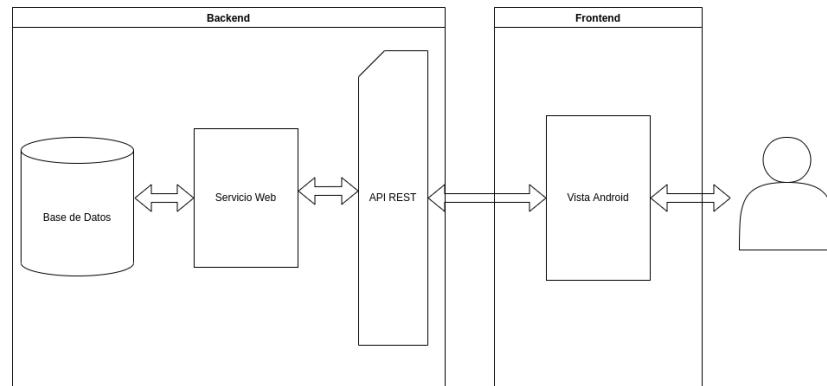


Figura 5.1: Arquitectura de la aplicación.

5.3 Interfaz de usuario

La vista cuenta con un gran número de pantallas. Esto es debido a que, al ser una aplicación móvil, el espacio disponible es más reducido que un proyecto para computadora convencional.

A continuación se dispondrán las pantallas que componen la interfaz:

- **Pantalla de Signup:** Registrar usuario con sus datos.
- **Pantalla de Login:** Acceder a la aplicación.
- **Pantalla de Ver Rutas:** Ver rutas que se celebrarán.
- **Pantalla de Detalles de Ruta:** Ver detalle de una ruta.
- **Pantalla de Crear Ruta:** Seleccionar recorrido de la ruta.
- **Pantalla de Introducir Datos de Ruta:** Introducir datos restantes de la ruta.
- **Pantalla de Listar Caballos:** Listar caballos del usuario.
- **Pantalla de Añadir Caballo:** Añadir un nuevo caballo en posesión del usuario.
- **Pantalla de Detalles de Caballo:** Ver características del caballo.
- **Pantalla de Listar Observadores:** Ver observadores seleccionados.
- **Pantalla de Ver Ubicación de Observados:** Ver última ubicación geográfica del observado.
- **Pantalla de Chat:** Ver chat de ruta.

- **Pantalla de Listar Participantes:** Ver participantes de una ruta y caballos con los que participarán.
- **Pantalla de Registrar Datos de Ruta:** Registrar estadísticas del recorrido que se esté realizando.
- **Pantalla de Listar Registros:** Ver recorridos registrados.
- **Pantalla de Detalles de Registro:** Ver estadísticas del recorrido seleccionado.
- **Pantalla de Gráfico de altitudes:** Ver gráfico con la altitud de cada punto por el que se ha pasado en el recorrido seleccionado.
- **Pantalla de Listar Amigos:** Ver amigos añadidos.
- **Pantalla de Ver Rutas de Amigos:** Ver rutas en las que participará un amigo.
- **Pantalla de Ver Rutas Pasadas:** Ver rutas que han sido realizadas en el pasado, a una distancia determinada seleccionada/seleccionable por el usuario.
- **Pantalla de Actualizar Ruta Pasada:** Actualizar aspectos de la ruta pasada, como la nueva fecha de celebración.

La interacción entre las pantallas mencionadas se muestra en la Figura 5.2. En la imagen se observa el flujo existente entre las distintas vistas.

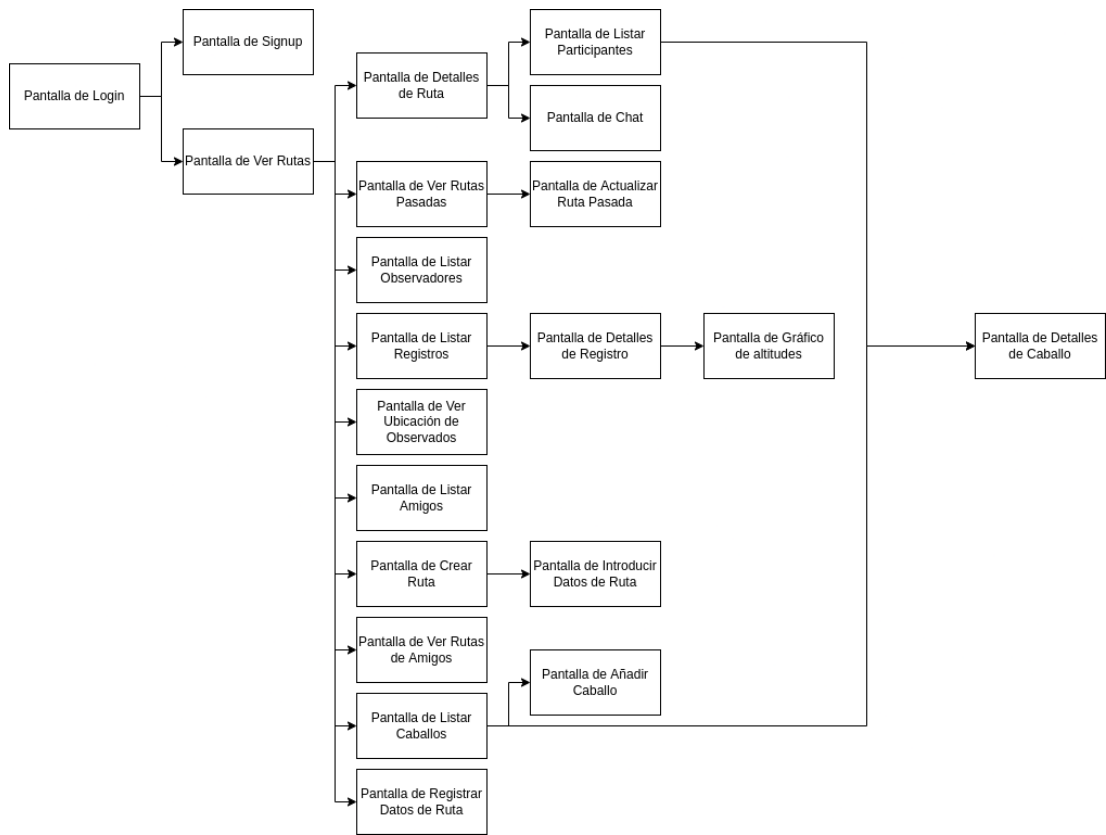


Figura 5.2: Flujo de pantallas.

Capítulo 6

Diseño

UNA parte crucial del proyecto es el diseño. Este consiste en la planificación de los componentes que formarán la aplicación. Un proceso de diseño especialmente largo es un elemento distintivo de las metodologías tradicionales que se basaban en la planificación pormenorizada de todo el proyecto.

En el caso de las metodologías ágiles, la carga que supone esta fase es más liviana, dado que se entiende que el proyecto estará sujeto a cambios. Además, las fases de desarrollo tienden a organizarse en las distintas iteraciones.

En este capítulo se analizarán las distintas tareas realizadas para el diseño de la aplicación y los productos que se obtuvieron de ellas.

6.1 Arquitectura tecnológica del sistema

La arquitectura de la aplicación consiste en un típico cliente-servidor, muy característico de proyectos con necesidades similares. En la Figura 5.1 se pueden observar gráficamente los distintos componentes que forman parte del sistema.

La base de datos usa la tecnología *PostgreSQL*, mientras que la realización de tanto el servicio como la *API* es resultado del uso del *framework Drango REST framework*. Por otro lado, la vista se implementa usando las distintas facilidades que proporciona el lenguaje *Kotlin*, ampliamente usado para el desarrollo *Android*.

6.2 Diseño de la aplicación

El diseño de la parte lógica de una aplicación es de suma importancia. Esto es así porque puede marcar la diferencia en el rendimiento de una aplicación. Una lógica compleja puede suponer una gran pérdida de eficiencia. Por esto, desarrollarla siguiendo una estructura cuidada ayuda en gran medida a paliar posibles defectos. Además, cabe destacar la capacidad de

mantener una aplicación. Numerosos estudios confirman que la mayor parte del esfuerzo y el gasto no provienen de la creación de la aplicación, sino de la posterior fase de mantenimiento y mejora continua.

6.2.1 Modelo conceptual de datos

La gestión de los datos de una aplicación es una cuestión muy importante para cualquier empresa. Para empezar, en una economía cada vez más orientada al conocimiento, los datos son un elemento clave para la optimización del proceso de una empresa y la maximización de las ganancias.

Además, existe una fuerte legislación a favor de la protección de datos, en el caso de Europa, parece que aún está por hacerse más dura en el futuro. Esto implica que el interés que puede tener cada organización en cómo trata y almacena sus datos debe ser de gran importancia.

Por esto, un correcto diseño de la estructura de datos debe ser una prioridad en el proyecto. Durante esta sección se expondrá su diagrama entidad-relación, como se puede apreciar en la Figura 6.1 y el diccionario de datos

Los diccionarios de datos tienen como principal objetivo facilitar la comprensión de las entidades. Se expondrán a continuación las entidades de las que se compone la estructura de datos del proyecto:

- *Rider*: En la Tabla 6.1 se observan los datos almacenados por la entidad *Rider*. Contiene todos los datos relevantes de un usuario.

Existen dos tablas de relaciones a mayores que relacionan esta entidad consigo misma. Estas son, la de amigos, cuyo diccionario puede consultarse en la Tabla 6.3, y la de observadores, siendo, a su vez, la Tabla 6.2 la que contiene su diccionario de datos.

- *Route*: Esta es la entidad que mayor carga funcional poseerá dado que todo el proyecto está centrado en la gestión de rutas. Se puede consultar su diccionario de datos en la Tabla 6.5. Almacenará todos los datos relevantes en los que un usuario pueda estar interesado.

A mayores, hay que hacer mención la tabla de participantes que relaciona cada ruta con los participantes inscritos en ella. Su diccionario de datos está disponible en la Tabla 6.6.

- *Record*: El diccionario de datos de la entidad *Record* se expone en la Tabla 6.4. Esta entidad almacenará los datos pertenecientes a los recorridos de los que el usuario haya decidido recopilar información.

- *Message*: Se recopilarán todos los mensajes de la aplicación de cada ruta en esta entidad. La base de datos, además, registrará automáticamente la fecha de creación de cada uno de ellos para mayor gestión de la mensajería, como se puede consultar en la Tabla 6.7.
- *Point*: Esta entidad recoge cada uno de los puntos geográficos que necesite la aplicación. Puede estar relacionada con dos entidades distintas dependiendo del contexto. Estas son *Route* y *Record*. Los datos que almacena son consultables a través de la Tabla 6.8.
- *Horse*: La última entidad, *Horse*, se encarga del almacenamiento de todas las características relevantes de un caballo para un usuario. En la Tabla 6.9 se observan los datos de los que está compuesta.

Cabe destacar que la entidad *Horse* existen tres datos que solo pueden tener una serie de valores específicos. No se añadieron en la Tabla 6.9 para no maximizar su tamaño en demasía. Estos son:

- Fila *coat*, los valores 0 (White), 1 (Black), 2 (Brown), 3 (Sorrel), 4 (Trush), 5 (Appaloosa), 6 (Overo), 7 (Roan), 8 (Bay), 9 (Elizabethan).
- Fila *gender*, los valores 0 (Stallion), 1 (Gelding), 2 (Mare).
- Fila *breed*, los valores Andaluz (PRE), Pura Raza Inglesa, Pura Raza Galega, Árabe, Akhal-Teke, Cuarto de milla, Appaloosa, Azteca, Paso Peruano, Painted Horse, Paso Tennessee, Mustang, Shire, Frisón, Percherón, Marwari, Lusitano, Bretón, Bereber, Criollo, Gelder, Hannoveriano, Hispano-Árabe, Kentucky Mountain, Lippizzano, Mongol, 'Morgan', Przewalski.

Tabla 6.1: Diccionario de datos. Rider.

Nombre	Tipo de dato	Restricciones	Descripción
id	bigint	Clave primaria	Identificador de cada elemento, autogenerado por la base de datos.
last_x_coord	float	-	Última coordenada x registrada del usuario durante una ruta que llevó a cabo.
last_y_coord	float	-	Última coordenada y registrada del usuario durante una ruta que llevó a cabo.
username	char	Único. Not null	Nick elegido por el usuario para identificarlo pseudoanónimamente.
first_name	char	Not null	Nombre de pila del usuario.
last_name	char	Not null	Apellido del usuario. En el caso de países latinos, dos apellidos.
email	char	Not null	Email del usuario.

6.2.2 Servicio

En este proyecto, al emplearse el *framework* de *Django*, el *backend* poseerá una estructura dirigida por este marco [16].

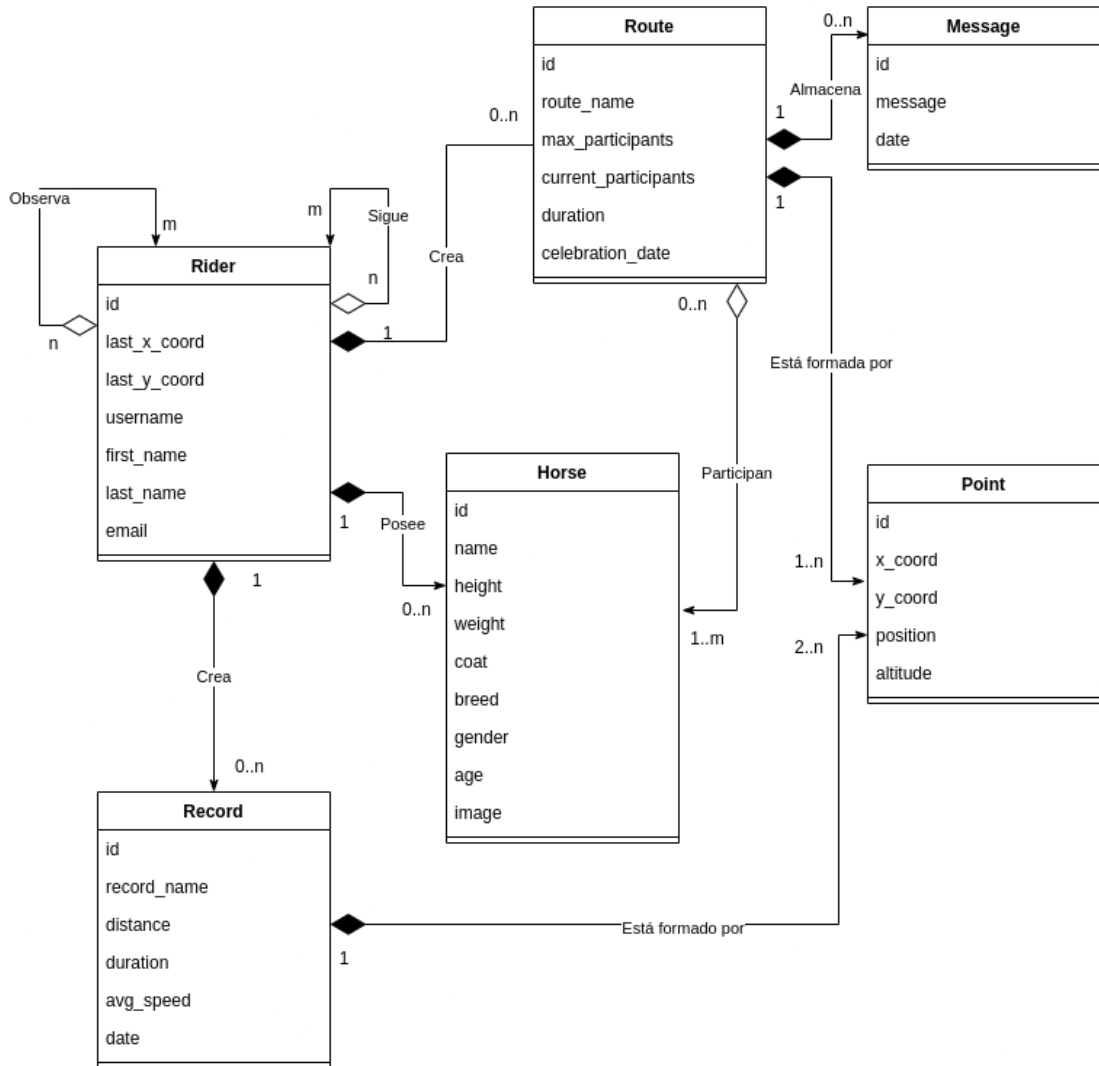


Figura 6.1: Entidad relación.

Tabla 6.2: Diccionario de datos. Observers.

Nombre	Tipo de dato	Restricciones	Descripción
observer	bigint	Not null, clave foránea	Usuario con la capacidad de acceder a la última ubicación del observado.
observed	bigint	Not null, clave foránea	Usuario que permite que el observador acceda a su última ubicación.

Tabla 6.3: Diccionario de datos. Friends.

Nombre	Tipo de dato	Restricciones	Descripción
friend1	bigint	Not null, clave foránea	Usuario que puede ver las rutas en las que su amigo está inscrito.
friend2	bigint	Not null, clave foránea	Usuario que puede ver las rutas en las que su amigo está inscrito.

Tabla 6.4: Diccionario de datos. Record.

Nombre	Tipo de dato	Restricciones	Descripción
id	bigint	Clave primaria	Identificador de cada elemento, autogenerado por la base de datos.
record_name	char	Not null	Nombre del registro.
distance	float	Not null	Distancia total recorrida en la ruta registrada.
duration	decimal	Not null	Duración total de la ruta registrada.
avg_speed	float	Not null	Velocidad media registrada en el total de la ruta.
date	datetime	Not null	Fecha de creación del registro. Generada automáticamente al guardar un nuevo elemento.
rider	bigint	Not null, clave foránea	Usuario que recorrió ruta.

Tabla 6.5: Diccionario de datos. Route.

Nombre	Tipo de dato	Restricciones	Descripción
id	bigint	Clave primaria	Identificador de cada elemento, autogenerado por la base de datos.
route_name	char	Not null, único	Nombre de la ruta.
max_participants	decimal	Not null	Participantes que se pueden unir a la ruta como máximo.
current_participants	decimal	Not null	Participantes que están actualmente inscritos.
duration	decimal	Not null	Estimación de la duración de la ruta.
celebration_date	datetime	Not null	Fecha y hora de celebración de la ruta.
creator	bigint	Not null, clave foránea	Usuario creador de la ruta.

Tabla 6.6: Diccionario de datos. Participants.

Nombre	Tipo de dato	Restricciones	Descripción
participant	bigint	Not null, clave foránea	Caballo inscrito a esa ruta, por ende, también usuario.
route	bigint	Not null, clave foránea	Ruta a la que está inscrito el participante.

Tabla 6.7: Diccionario de datos. Message.

Nombre	Tipo de dato	Restricciones	Descripción
id	bigint	Clave primaria	Identificador de cada elemento, autogenerado por la base de datos.
message	char	Not null	Mensaje escrito por el usuario.
date	datetime	Not null	Fecha y hora en la que el mensaje fue escrito. Generada automáticamente al introducir un nuevo elemento.
writer	bigint	Not null, clave foránea	Usuario escritor del mensaje.
route	bigint	Not null, clave foránea	Ruta en la que fue escrito el mensaje.

Tabla 6.8: Diccionario de datos. Point.

Nombre	Tipo de dato	Restricciones	Descripción
id	bigint	Clave primaria	Identificador de cada elemento, autogenerado por la base de datos.
x_coord	float	Not null	Coordenada x del punto geográfico.
y_coord	float	Not null	Coordenada y del punto geográfico.
position	decimal	Not null	Posición del punto en la secuencia de puntos a la que pertenece.
altitude	float	-	Altitud sobre el nivel del mar al que se encuentra el punto.
route	bigint	Clave foránea	Ruta a la que pertenece el punto.
record	bigint	Clave foránea	Registro al que pertenece el punto.

Tabla 6.9: Diccionario de datos. Horse.

Nombre	Tipo de dato	Restricciones	Descripción
id	bigint	Clave primaria	Identificador de cada elemento, autogenerado por la base de datos.
name	char	Not null	Nombre del caballo.
height	float	Not null	Altura a la cruz del caballo.
weight	float	Not null	Peso del caballo.
coat	integer	Not null	Color de capa del caballo.
breed	char	Not null	Raza del caballo.
gender	integer	Not null	Género del caballo.
age	integer	Not null, número positivo	Edad del caballo.
image	char	-	Imagen del caballo. Almacena el <i>path</i> de la imagen en el sistema.
owner	bigint	Not null, clave foránea	Propietario del caballo.

La parte servidor de la aplicación debe proporcionar una serie de funciones para prestar servicio a la capa vista de la aplicación.

Las funciones que permitirán acceder a los datos internos de la aplicación estarán divididas en dos ficheros según su relación con las dos entidades más importantes del proyecto, *Rider* y *Route*. Las funciones que prestarán servicio más cercano a la entidad *Rider* se implementarán en el fichero *riderViews*. A su vez, las funciones relacionadas con la entidad *Route*, se desarrollarán en el fichero *routeViews*. Estas funciones se muestran en la Figura 6.2.

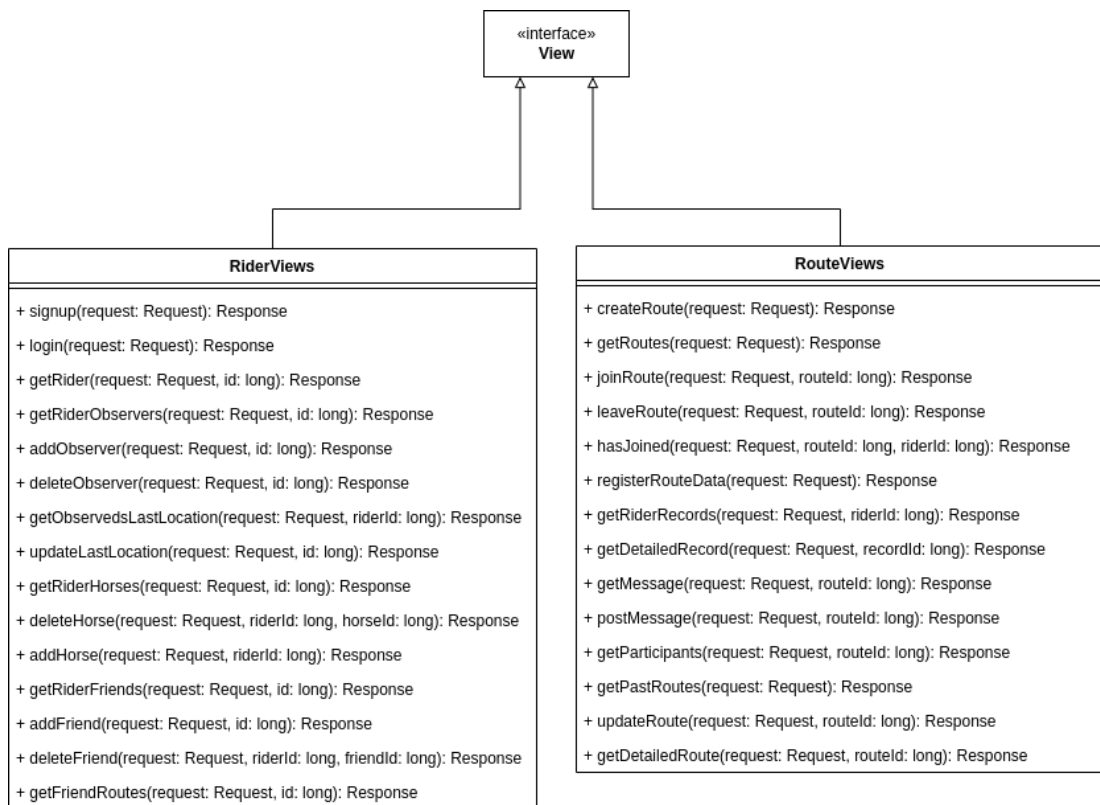


Figura 6.2: Componentes del servicio.

El conjunto de estas funciones que pretenden prestar servicio a las diferentes solicitudes hacen uso del sistema de datos internos del servicio. Son las clases que representan los datos almacenados por la base de datos, facilitando esto su manipulación y organización a la hora de exponerlos al usuario de la aplicación. Al estar relacionadas estas clases con las entidades del sistema, resultan muy similares. No obstante, en este caso, toman de *Django* diferentes particularidades. En este marco de desarrollo existen varias entidades predeterminadas que facilitan la implementación e interactúan de un modo más cercano con el mismo. Este es el caso de clase *User*, que se observa en el diagrama de clases interno del *backend* 6.3. Esta clase facilita ciertas funciones al desarrollador. En este caso particular, resultará de gran ayuda para

el control de la autenticación de los distintos usuarios.

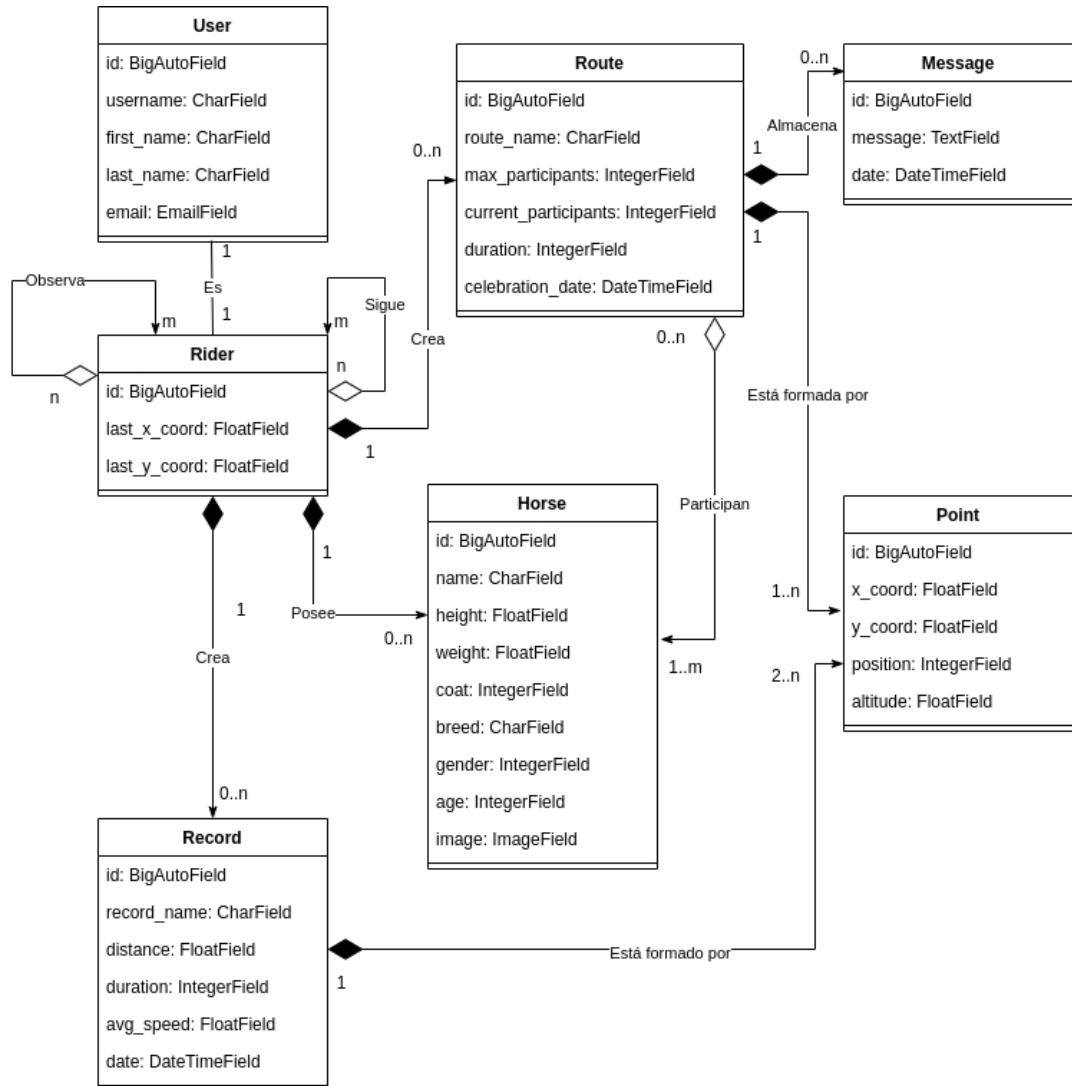


Figura 6.3: Diagrama de clases del servicio.

Para proporcionar un servicio de integración correcto, se debe facilitar la documentación necesaria para comprender la integración con la parte lógica de la aplicación. A continuación, se detallarán los *endpoints* con los que se puede interactuar¹:

- /user/signup/

¹ Los códigos de respuesta de las peticiones HTTP tienen el siguiente significado:

- 200: Petición correcta.
- 400: Petición incorrecta, problema con algún dato proporcionado.
- 401: Fallo de autenticación. El *token* no existe o no es válido.
- 403: Petición prohibida para el usuario.
- 500: Fallo interno, ha ocurrido un error en el lado servidor.

- Método HTTP: POST
 - Cabeceras: -
 - Parámetros: -
 - Respuesta: 400, 500, 200 (devuelve objeto json con identificador, nombre de usuario, nombre, apellidos, y email del usuario)
 - Descripción: Crear nuevo usuario.

- /user/login/
 - Método HTTP: POST
 - Cabeceras: -
 - Parámetros: -
 - Respuesta: 400, 500, 200 (devuelve objeto json con identificador, nombre de usuario, nombre, apellidos, y email del usuario)
 - Descripción: Loggear usuario.

- /user/<riderId>/observers/
 - Método HTTP: GET
 - Cabeceras: Authorization (token de autorización de usuario)
 - Parámetros: riderId (identificador del usuario)
 - Respuesta: 400, 500, 401, 403, 200 (devuelve lista de objetos json con identificador y nombre de usuario del observador)
 - Descripción: Obtener lista de observadores del jinete.

- /user/<riderId>/add_observer/
 - Método HTTP: POST
 - Cabeceras: Authorization (token de autorización de usuario)
 - Parámetros: riderId (identificador del usuario)
 - Respuesta: 400, 500, 401, 403, 200 (devuelve objeto json con identificador y nombre de usuario del observador)
 - Descripción: Añadir nuevo observador.

- /user/<riderId>/delete_observer/<observerId>/

- Método HTTP: DELETE
 - Cabeceras: Authorization (token de autorización de usuario)
 - Parámetros: riderId (identificador del usuario), observerId (identificador del usuario observador)
 - Respuesta: 400, 500, 401, 403, 200
 - Descripción: Borrar observador.
- /user/<riderId>/observeds/
 - Método HTTP: GET
 - Cabeceras: Authorization (token de autorización de usuario)
 - Parámetros: riderId (identificador del usuario)
 - Respuesta: 400, 500, 401, 403, 200 (devuelve lista de objetos json con identificador, nombre de usuario, última latitud y última longitud del observado)
 - Descripción: Obtener últimas ubicaciones de observados.
- /user/<riderId>/modify_last_location/
 - Método HTTP: POST
 - Cabeceras: Authorization (token de autorización de usuario)
 - Parámetros: riderId (identificador del usuario)
 - Respuesta: 400, 500, 401, 403, 200
 - Descripción: Modificar última ubicación.
- /user/<riderId>/horses/
 - Método HTTP: GET
 - Cabeceras: Authorization (token de autorización de usuario)
 - Parámetros: riderId (identificador del usuario)
 - Respuesta: 400, 500, 401, 403, 200 (devuelve lista de objetos json con identificador, nombre, peso, altura, capa, raza, género edad y ruta de imagen del caballo)
 - Descripción: Obtener caballos del jinete.
- /user/<riderId>/horse/<horseId>/delete/
 - Método HTTP: DELETE

- Cabeceras: Authorization (token de autorización de usuario)
 - Parámetros: riderId (identificador del usuario), horseId (identificador del caballo)
 - Respuesta: 400, 500, 401, 403, 200
 - Descripción: Borrar caballo.

- /user/<riderId>/add_horse/
 - Método HTTP: POST
 - Cabeceras: Authorization (token de autorización de usuario)
 - Parámetros: riderId (identificador del usuario)
 - Respuesta: 400, 500, 401, 403, 200 (devuelve objeto json con identificador, nombre, peso, altura, capa, raza, género edad y ruta de imagen del caballo)
 - Descripción: Añadir caballo.

- /user/<riderId>/friends/
 - Método HTTP: GET
 - Cabeceras: Authorization (token de autorización de usuario)
 - Parámetros: riderId (identificador del usuario)
 - Respuesta: 400, 500, 401, 403, 200 (devuelve lista de objetos json con identificador y nombre de usuario del amigo)
 - Descripción: Obtener lista de amigos.

- /user/<riderId>/add_friend/
 - Método HTTP: POST
 - Cabeceras: Authorization (token de autorización de usuario)
 - Parámetros: riderId (identificador del usuario)
 - Respuesta: 400, 500, 401, 403, 200 (devuelve objeto json con identificador y nombre de usuario del amigo)
 - Descripción: Añadir amigo.

- /user/<riderId>/delete_friend/<friendId>/
 - Método HTTP: DELETE
 - Cabeceras: Authorization (token de autorización de usuario)

- Parámetros: riderId (identificador del usuario), friendId (identificador del usuario amigo)
 - Respuesta: 400, 500, 401, 403, 200
 - Descripción: Eliminar amigo.
- /user/<riderId>/get_friend_routes/
 - Método HTTP: GET
 - Cabeceras: Authorization (token de autorización de usuario)
 - Parámetros: riderId (identificador del usuario), friendId (identificador del usuario amigo)
 - Respuesta: 400, 500, 401, 403, 200 (devuelve lista de objetos json con identificador, identificador del creador, nombre, participantes máximos, participantes actuales, duración, fecha celebración y puntos de la ruta, siendo puntos a su vez una lista de objetos json con latitud, longitud y posición del punto)
 - Descripción: Obtener rutas en las que participan amigos.
- /route/create_route/
 - Método HTTP: POST
 - Cabeceras: Authorization (token de autorización de usuario)
 - Parámetros: -
 - Respuesta: 400, 500, 401, 200 (devuelve objeto json con identificador, identificador del creador, nombre, participantes máximos, participantes actuales, duración y fecha celebración de la ruta)
 - Descripción: Crear ruta.
- /route/get_routes/
 - Método HTTP: GET
 - Cabeceras: Authorization (token de autorización de usuario)
 - Parámetros: -
 - Respuesta: 500, 401, 200 (devuelve lista de objetos json con identificador, identificador del creador, nombre, participantes máximos, participantes actuales, duración, fecha celebración y puntos de la ruta, siendo puntos a su vez una lista de objetos json con latitud, longitud y posición del punto)

- Descripción: Obtener rutas que se celebrarán.
- /route/<routeId>/join_route/
 - Método HTTP: POST
 - Cabeceras: Authorization (token de autorización de usuario)
 - Parámetros: routeId (identificador de la ruta)
 - Respuesta: 400, 500, 401, 403, 200 (devuelve objeto json con identificador, identificador del creador, nombre, participantes máximos, participantes actuales, duración y fecha celebración de la ruta)
 - Descripción: Inscribir jinete en una ruta.
- /route/<routeId>/leave_route/
 - Método HTTP: POST
 - Cabeceras: Authorization (token de autorización de usuario)
 - Parámetros: routeId (identificador de la ruta)
 - Respuesta: 400, 500, 401, 403, 200
 - Descripción: Abandonar ruta.
- /route/<routeId>/has_joined/<riderId>/
 - Método HTTP: GET
 - Cabeceras: Authorization (token de autorización de usuario)
 - Parámetros: routeId (identificador de la ruta), riderId (identificador del usuario)
 - Respuesta: 400, 500, 401, 200 (devuelve objeto json con un valor numérico según el estado del jinete en la ruta)
 - Descripción: Obtener si el usuario participa en la ruta.
- /route/register_route_data/
 - Método HTTP: POST
 - Cabeceras: Authorization (token de autorización de usuario)
 - Parámetros: -
 - Respuesta: 400, 500, 401, 403, 200 (devuelve objeto json con identificador, identificador del creador, nombre, distancia, duración y velocidad media del registro)

- Descripción: Registrar datos obtenidos de la ruta realizada.
- /route/rider_records/<riderId>/
 - Método HTTP: GET
 - Cabeceras: Authorization (token de autorización de usuario)
 - Parámetros: riderId (identificador del usuario)
 - Respuesta: 400, 500, 401, 403, 200 (devuelve lista de objetos json con identificador, nombre y fecha del registro)
 - Descripción: Obtener rutas realizadas.
- /route/detailed_record/<recordId>/
 - Método HTTP: GET
 - Cabeceras: Authorization (token de autorización de usuario)
 - Parámetros: recordId (identificador del registro)
 - Respuesta: 400, 500, 401, 403, 200 (devuelve objeto json con identificador, identificador del creador, nombre, distancia, duración, velocidad media y puntos del registro, siendo puntos a su vez una lista de objetos json con latitud, longitud y posición del punto)
 - Descripción: Obtener información detallada de ruta registrada.
- /route/<routeId>/messages/
 - Método HTTP: GET
 - Cabeceras: Authorization (token de autorización de usuario)
 - Parámetros: routeId (identificador de la ruta)
 - Respuesta: 400, 500, 401, 200 (devuelve lista de objetos json con identificador del emisor, nombre de usuario del emisor, texto y fecha del mensaje)
 - Descripción: Obtener los mensajes de la ruta.
- /route/<routeId>/post_message/
 - Método HTTP: POST
 - Cabeceras: Authorization (token de autorización de usuario)
 - Parámetros: routeId (identificador de la ruta)

- Respuesta: 400, 500, 401, 403, 200 (devuelve objeto json con identificador del emisor, nombre de usuario del emisor, texto y fecha del mensaje)
 - Descripción: Añadir nuevo mensaje a la ruta.
- /route/<routeId>/participants/
 - Método HTTP: GET
 - Cabeceras: Authorization (token de autorización de usuario)
 - Parámetros: routeId (identificador de la ruta)
 - Respuesta: 400, 500, 401, 200 (devuelve lista de objetos json con identificador del jinete, nombre de usuario del jinete, identificador del caballo, nombre, peso, altura, capa, raza, género edad y ruta de imagen del caballo)
 - Descripción: Obtener lista de participantes.
- /route/get_historic/
 - Método HTTP: GET
 - Cabeceras: Authorization (token de autorización de usuario)
 - Parámetros: -
 - Respuesta: 400, 500, 401, 200 (devuelve lista de objetos json con identificador, identificador del creador, nombre, participantes máximos, participantes actuales, duración, fecha celebración y puntos de la ruta, siendo puntos a su vez una lista de objetos json con latitud, longitud y posición del punto)
 - Descripción: Obtener rutas pasadas.
- /route/<routeId>/update/
 - Método HTTP: POST
 - Cabeceras: Authorization (token de autorización de usuario)
 - Parámetros: routeId (identificador de la ruta)
 - Respuesta: 400, 500, 401, 403, 200 (devuelve objeto json con identificador, identificador del creador, nombre, participantes máximos, participantes actuales, duración y fecha celebración de la ruta)
 - Descripción: Crear ruta a partir de una antigua.
- /route/<routeId>/detailed/

- Método HTTP: GET
- Cabeceras: Authorization (token de autorización de usuario)
- Parámetros: routeId (identificador de la ruta)
- Respuesta: 400, 500, 401, 200 (devuelve objeto json con identificador, identificador del creador, nombre, participantes máximos, participantes actuales, duración, fecha celebración y puntos de la ruta, siendo puntos a su vez una lista de objetos json con latitud, longitud y posición del punto)
- Descripción: Obtener información detallada de una ruta.

6.2.3 Frontend

Para gestionar los datos recuperados desde el modelo, la aplicación móvil implementa también un sistema de entidades. Este sistema permitirá una adecuada estructuración de los mismos y acceso rápido al valor necesitado. En la Figura 6.4 se muestra un diagrama del mencionado sistema.

Gracias a una mejor organización de los diferentes recursos, se consigue una solución mucho más legible y mantenible. Esto es de gran utilidad para disminuir futuros costos del equipo en el futuro y hacer más sencillo el trabajo.

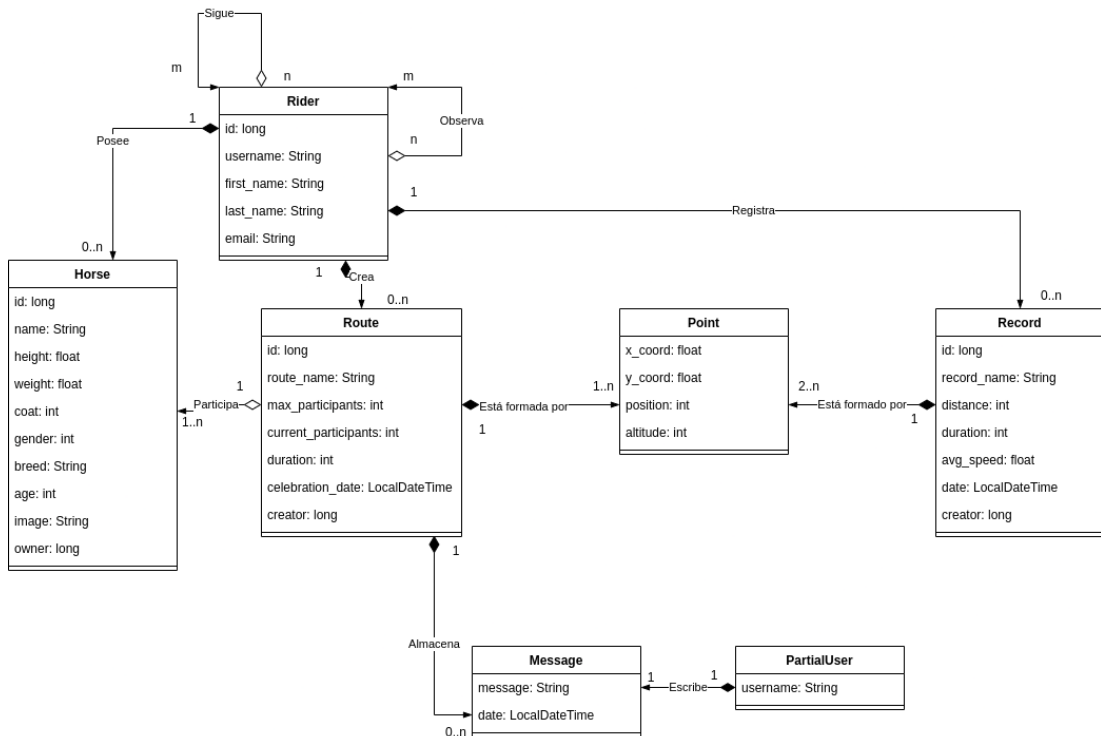


Figura 6.4: Frontend. Diagrama de clases con los objetos del dominio.

En cuanto a la estructura del software que controlaba las distintas pantallas existentes, este se puede observar en la Figura 6.5. En Kotlin, las pantallas son administradas por clases nombradas actividades o fragmentos, dependiendo de su naturaleza, y son las encargadas de renderizar lo que va a mostrarse por pantalla, así como de llevar a cabo las operaciones necesarias para conseguir los datos correctos para su visualización.

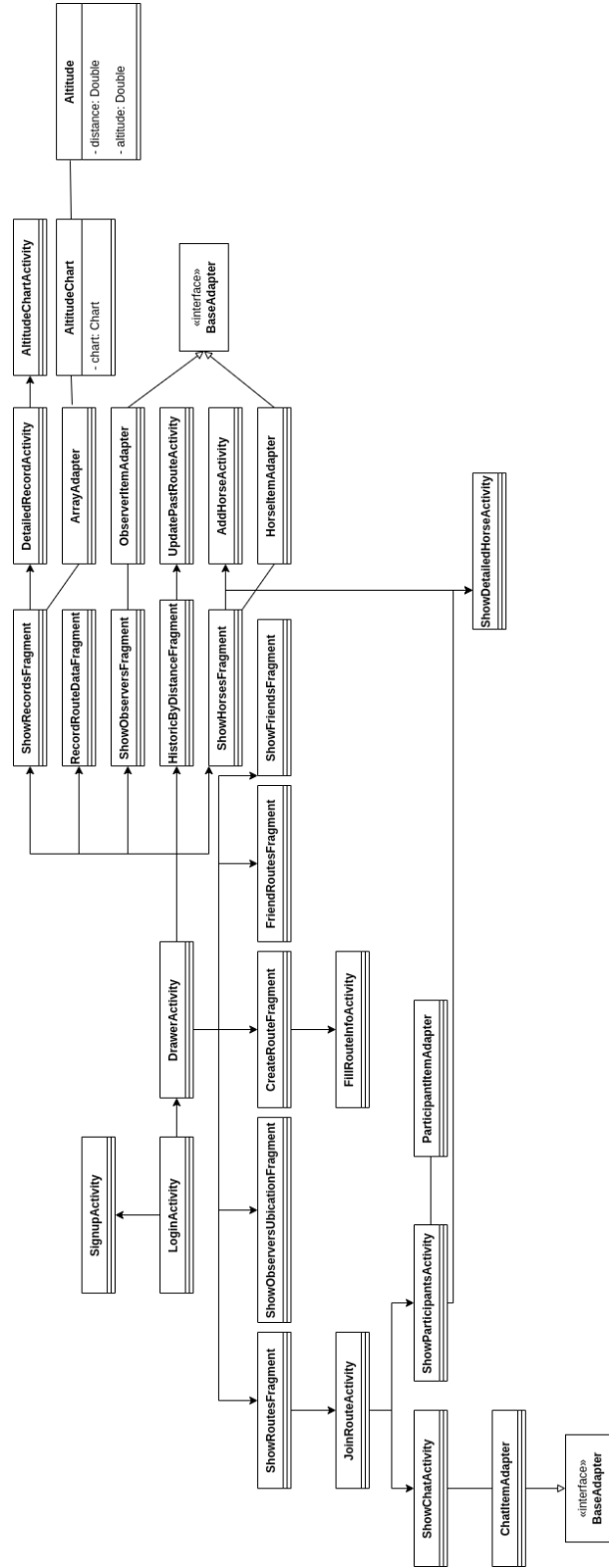


Figura 6.5: Frontend. Diagrama de actividades que controlan la vista de la aplicación.

Implementación

DURANTE este capítulo se expondrán los diferentes algoritmos y aspectos clave de la implementación con el objetivo clarificar posibles dudas que puedan surgir con respecto a ellos. A continuación se detallarán las decisiones tomadas para las distintas situaciones surgidas, estructuradas en dos aspectos diferenciadores, si pertenecen al modelo o a la interfaz de usuario.

7.1 Backend

El *backend* de la aplicación se encarga de prestar servicio a la interfaz de usuario, permitiendo almacenar datos de forma permanente, así como actualizarlos, consultarlos y tratarlos. Con grandes cantidades de datos la eficiencia de este componente es realmente muy importante para el correcto funcionamiento de la aplicación. Un bajo rendimiento y altos tiempos de carga suponen un gran revés para el usuario. Esto puede conllevar su abandono de la aplicación y es responsabilidad del equipo asegurarse del buen funcionamiento del *software*.

7.1.1 Serializadores

Django REST framework proporciona un esqueleto y forma de programación muy concretos, haciendo ágil y sencillo la implementación de la solución. Concretamente, se basa en el uso de vistas y serializadores. Las vistas toman las peticiones del cliente, acceden a los datos del modelo y devuelven una respuesta *HTTP*. La interacción con el cliente se realiza mediante el intercambio de estructuras de datos *JSON*. De estos archivos es de los que se encargan los serializadores, haciendo posible la conversión entre modelos de datos de *Django* a tipos de datos *JSON*. Existen varias implementaciones en *Django* para un serializador, en este caso, cada serializador se generalizará de la clase *serializers.ModelSerializer*, que ofrece gran simplicidad. A continuación se muestra un serializador tipo:

```
1 class HorseSerializer(serializers.ModelSerializer):
```

```

2
3     class Meta:
4         model = Horse
5         fields = '__all__'

```

Existen casos dónde se hace necesario acceder a otros modelos a mayores en un único serializador. Para este caso, puede usarse el parámetro *depth* para indicar la profundidad máxima de los datos a los que debe acceder el serializador, como se muestra en el código siguiente, en el que se puede observar cómo se accedería a la entidad *Horse* y el propio serializer se encargaría de recuperar a mayores la entidad *Rider* cuyo *id* es especificado en el parámetro *owner* del caballo:

```

6 class DepthExampleHorseSerializer(serializers.ModelSerializer):
7
8     class Meta:
9         model = Horse
10        fields = '__all__'
11        depth = 1

```

No obstante en la implementación se optó mayormente por un método que permita mayor control sobre los datos objetivos. Esto se realiza accediendo a otro serializador gracias a indicar la columna en la que se debe basar el serializador automáticamente para la consulta. Se realizaría gracias a haber indicado la relación en el parámetro del modelo *related_name*:

```

12 class ObserverUserSerializer(serializers.ModelSerializer):
13
14     class Meta:
15         model = User
16         fields=['username']
17
18 class LastLocationSerializer(serializers.ModelSerializer):
19     user = ObserverUserSerializer()
20
21     class Meta:
22         model = Rider
23         fields = ['user', 'last_x_coord', 'last_y_coord']
24
25
26 observeds_serializer = LastLocationSerializer(observeds, many=True)
27 observeds_serializer.data

```

El ejemplo anterior nos dará un resultado *JSON* como el siguiente:

```

1 {
2     "user": {
3         "username": "x"

```

```

4         },
5         "last_x_coord": x,
6         "last_x_coord": x
7     }

```

Los serializadores permitirán recuperar una serie de objetos, en vez de un único elemento, mediante la especificación del parámetro booleano *many* como se puede observar a continuación:

```

28 route_serializer = GetRoutesSerializer(routes_list, many=True)

```

7.1.2 Seguridad

Los datos están sujetos a una fuerte legislación y, cada año que pasa, se hace más estricta. Ya no solo por estas obligaciones, sino también por ética común, los datos manejados por el proyecto deben ser protegidos. El aspecto de seguridad en el que hace especial hincapié esta aplicación es la identificación del usuario, con el objetivo de que una persona con objetivos maliciosos no pueda interferir en el desarrollo normal del servicio. Esto implica, por ejemplo, impedir que un usuario pueda inscribir a otro en una ruta arbitraria.

En el caso concreto de esta aplicación, *Django REST framework* ofrece varias medidas para asegurar la seguridad del modelo. Específicamente, se hará uso del módulo *rest_framework.auth_token*. Este nos proporciona la posibilidad de crear distintos tokens con los que poder identificar inequívocamente al usuario que envía la petición. En este caso, se hace uso de un token no percedero que será creado o recuperado cada vez que el usuario inicie sesión como se puede observar a continuación:

```

29 token, _ = Token.objects.get_or_create(user = user)
30
31 rider = Rider.objects.get(user=user)
32 user_serialized = RiderSerializer(rider)
33
34 return Response({
35     'token': token.key,
36     'user': user_serialized.data
37 }, status=HTTP_200_OK)

```

Un usuario solo podrá acceder al API sin contar con el token específico en caso de interactuar con las funcionalidades de *signup* y *login*, dado que no es posible que lo tengan a su disposición a la hora de solicitar sus servicios. La forma de deshabilitar esta opción es mediante anotaciones como muestra siguiente fragmento:

```

38 @api_view(['POST'])
39 @permission_classes((AllowAny,))

```

```
40 def login(request)
```

En relación con la seguridad de la aplicación, *Django* también ofrece facilidades para la autenticación del usuario en el *login* mediante la función *authenticate*:

```
41 user = authenticate(  
42     username = signin_serializer.data['username'],  
43     password = signin_serializer.data['password']  
44 )  
45 if not user:  
46     return Response({'detail': 'Invalid Credentials or activate  
account'}, status=HTTP_404_NOT_FOUND)
```

7.2 Frontend

La fluidez de las pantallas y su facilidad de uso son factores críticos para la fidelización del usuario. Crear controles intuitivos ya conocidos por el usuario, acelerará en gran medida la rapidez de aprendizaje de manejo. En esta sección se explicarán detalles de implementación de la interfaz de la aplicación.

7.2.1 Internacionalización

La aplicación implementa la internacionalización de la interfaz. La tecnología utilizada en el desarrollo facilita este proceso mediante la sencilla creación de archivos de recursos *xml* en el directorio de recursos. Actualmente el proyecto puede presentarse en tres idiomas distintos dependiendo de la configuración del dispositivo móvil en el que esté instalado el *software*. Estos son: inglés, el idioma por defecto, castellano y gallego.

7.2.2 Paleta de colores

Para seleccionar unos colores adecuados para todos los elementos, que contrasten bien entre sí, se utilizó una aplicación web de generación de paletas [17]. Son almacenados en el fichero de colores del directorio de recursos. Los colores que conforman la aplicación se muestran en la Figura 7.1.

7.2.3 Conexión con el backend

El acceso al *backend* es una parte crítica de cualquier aplicación. La decisión correcta a la hora de elegir el método de conexión marca la diferencia en la experiencia de usuario. Por ejemplo, una conexión asíncrona, proporciona una experiencia ininterrumpida al usuario, para evitar que la interfaz no responda debido a una petición no atendida.

En la aplicación se usa la librería *Volley* [18], que permite conexiones asíncronas con el servidor. Su uso es sencillo y permite una gestión adecuada de los datos obtenidos del servicio mediante peticiones como *JsonObjectRequest* y *JSONArrayRequest*. En caso de ocurrir un error en la respuesta, se le notificará al usuario mediante un *Toast*, que es una notificación flotante que se crea en la parte inferior de la pantalla.

Para acceder al modelo, necesitará incluir el token en la cabecera de la petición, esto se hará sobrescribiendo el método *getHeaders* de la petición, como se muestra a continuación:

```
1 override fun getHeaders(): MutableMap<String, String> {  
2     val headers = HashMap<String, String>()  
3     headers["Authorization"] = "Token $token"  
4     return headers  
5 }
```

7.2.4 Compartir ruta

Para implementar esta función se usan los *application links*. Estos permiten la llamada a actividades de otra aplicaciones mediante *Intents*.

Los *application links* nos permitirán, para este caso particular, o bien guardar el código de la ruta en el portapapeles, o bien compartirlo o guardarlo mediante otra aplicación, ya sea *Whatsapp*, *Telegram*, *Twitter*, por ejemplo. A continuación se muestra el código que permite esta interacción entre aplicaciones:

```
6 try {  
7     val shareIntent = Intent(Intent.ACTION_SEND)  
8     shareIntent.type = "text/plain"  
9     shareIntent.putExtra(Intent.EXTRA_SUBJECT, "Sleipnir")  
10    var shareMessage = "\nShare route\n\n"  
11    shareMessage = shareMessage + "#" + jsonRoute.getInt("id")  
12    shareIntent.putExtra(Intent.EXTRA_TEXT, shareMessage)  
13    startActivity(Intent.createChooser(shareIntent, "choose one"))  
14 } catch (e: Exception) {  
15     Toast.makeText(this, R.string.error_sharing,  
16     Toast.LENGTH_SHORT).show()  
16 }
```

Como se observa, el código que representa una ruta concreta no es más que la unión del carácter “#” y el identificador de la ruta.

El usuario que pretenda encontrar la ruta concreta a partir de este código deberá introducirlo en el campo de búsqueda adecuado de la pantalla en la que se muestran todas las rutas que se celebrarán en el futuro (Figura 7.2).

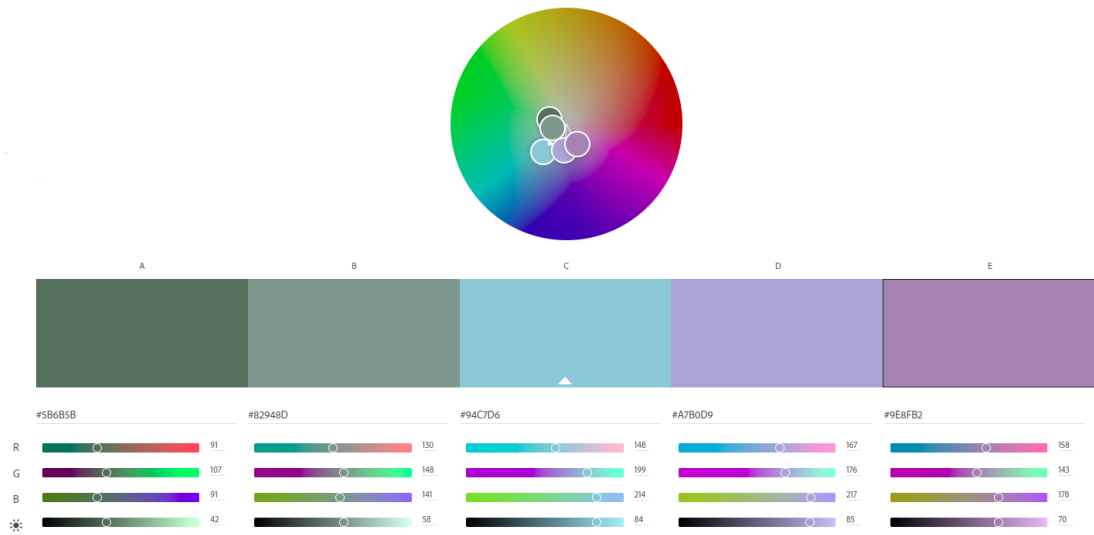


Figura 7.1: Paleta de colores.

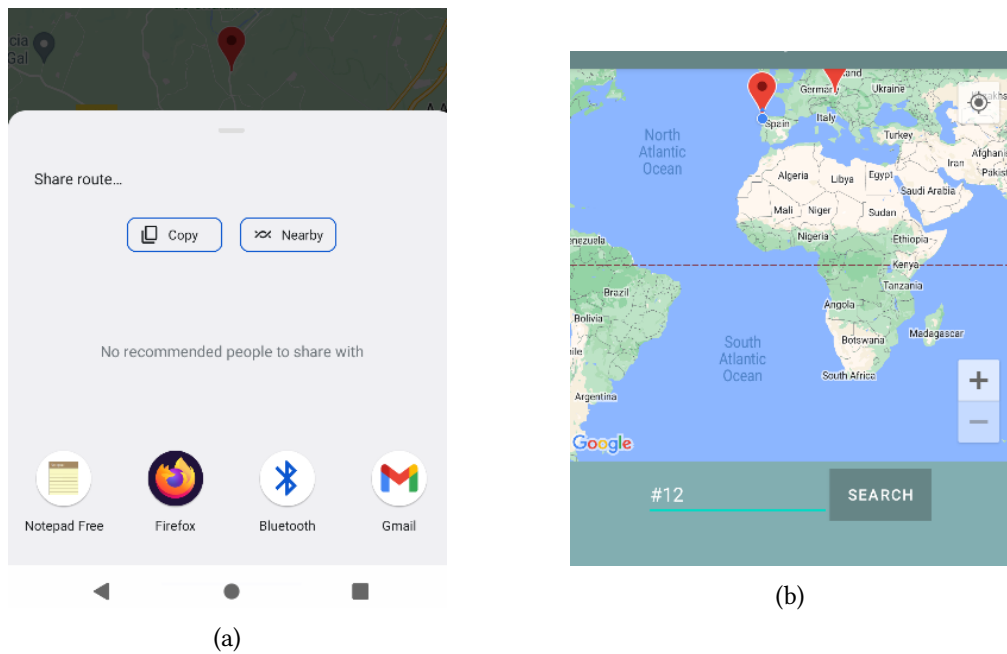


Figura 7.2: Gestión del código de una ruta. (a) Obtención del código. (b) Introducción del código.

7.2.5 Google Maps

Para la implementación de todas las funcionalidades referentes a mapas se hace uso del servicio de *Google Maps*, que proporciona una gran cantidad de servicios adecuados a esta aplicación.

Para poder hacer uso de *Google Maps* es necesario poseer una cuenta de *Google* y registrarse como desarrollador para solicitar una clave del *API*. Es necesario haber especificado las funcionalidades concretas de las que se hará uso mediante esa clave. Es posible hacer uso de la *API* para fines comerciales, pudiendo proporcionar servicio a grandes cantidades de usuarios, mediante una suscripción de pago. Sin embargo, para fines de desarrollo, no es necesario realizar ningún pago.

La clave adquirida deberá ser especificada en los *metadatos* del manifiesto de la aplicación *Kotlin*. Para ello, se almacenará en el fichero *local.properties*. Para la creación y exposición de rutas se hará uso de *Markers* y *Polylines* que ofrece el servicio de *Google Maps*.

7.2.6 Localización

La ubicación del usuario es una parte esencial de la aplicación. La primera función que tendrá será la de centrar el mapa en un punto geográfico relevante para el usuario. Es decir, un usuario tendrá mayor interés en localizar posibles rutas a las que asistir en un radio alrededor de su posición actual. La otra función que realizará es tomar nota de los puntos que va recorriendo el usuario a medida que avanza en su ruta mientras está grabando el recorrido.

La seguridad del dispositivo es estricta con respecto a datos geográficos del usuario. Esto implica que el usuario deberá aceptar de manera inequívoca, su conformidad con que la aplicación puede acceder a estos datos. Con este fin, se deberá incluir en el manifiesto de la aplicación los privilegios que se le solicitarán al usuario como se muestra en el siguiente código:

```
1 <uses-permission
   android:name="android.permission.ACCESS_BACKGROUND_LOCATION" />
2 <uses-permission
   android:name="android.permission.ACCESS_FINE_LOCATION" />
3 <uses-permission
   android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

Para requerir al usuario los permisos que solicita el uso de la localización se usan las siguientes sentencias:

```
17 if (ActivityCompat.checkSelfPermission(requireContext(),
18     Manifest.permission.ACCESS_FINE_LOCATION)
19     != PackageManager.PERMISSION_GRANTED
20     && ActivityCompat.checkSelfPermission(requireContext(),
    Manifest.permission.ACCESS_COARSE_LOCATION)
    != PackageManager.PERMISSION_GRANTED) {
```

```

21
22     ActivityCompat.requestPermissions(
23         requireContext() as Activity, arrayOf(
24             Manifest.permission.ACCESS_FINE_LOCATION,
25             Manifest.permission.ACCESS_COARSE_LOCATION),
26         LOCATION_REQUEST_CODE
27     )
28     return
29 }

```

Hay varias formas de acceder a la ubicación del usuario. Mediante el *GPS* del dispositivo y, en caso de no poder acceder a él, el servicio automáticamente trata de triangular la posición del usuario mediante los repetidores de los que hace uso la conexión.

Cuando es necesario localizar al usuario una única vez se hace uso del siguiente código:

```

30 fusedLocationClient.lastLocation.addOnSuccessListener(requireContext()
31     as Activity) { location ->
32
33     if (location != null) {
34         lastLocation = location
35         val currentLatLng = LatLng(location.latitude,
36             location.longitude)
37     }
38 }

```

En caso de necesitar repetidas actualizaciones de localización, se usa esta otra modalidad:

```

38 val locRequest = LocationRequest()
39 locRequest.interval = 180000
40 locRequest.fastestInterval = 60000
41 locRequest.smallestDisplacement = 90f
42 locRequest.priority = LocationRequest.PRIORITY_HIGH_ACCURACY
43
44 val locationCallback = object : LocationCallback() { getPoints() }

```

Esta manera de registrar la localización del usuario permite obtener actualizaciones automáticas. Además, es posible configurar ciertos parámetros a fin de controlar de una manera más conveniente la toma de datos.

- *Interval*: Milisegundos entre cada actualización de la ubicación del usuario.
- *fastestInterval*: Milisegundos mínimos entre cada actualización de la ubicación del usuario.
- *smallestDisplacement*: Distancia mínima en metros entre la localización anterior y la actual para actualizar la localización.

7.2.7 Almacenamiento de configuración

En toda aplicación, y más en un dispositivo personal, se requiere de guardar ciertos datos de forma permanente, bien porque se usará de forma muy recurrente, o bien porque se necesitará en futuros usos sin requerirlos del usuario o del servicio.

Para almacenar esta información se usará *SharedPreferences* que ofrece *Kotlin*. El registro de datos se lleva a cabo como se observa a continuación:

```
45 val sharedPref : SharedPreferences =
    applicationContext.getSharedPreferences("userPreference",
        MODE_PRIVATE)
46 val user = response.getJSONObject("user")
47 with (sharedPref.edit()) {
48     putString("token", response.getString("token"))
49     putInt("userId", user.getInt("id"))
50     putString("userName", userNameInput.text.toString())
51     commit()
52 }
```

Este código nos permite almacenar el token, el identificador y el nombre de usuario del jinete al acceder a la aplicación desde la pantalla de *login*. Para recuperarla, se hace como se muestra en este código:

```
53 val sharedPref : SharedPreferences =
    requireActivity().getSharedPreferences("userPreference",
        AppCompatActivity.MODE_PRIVATE)
54 val userId = sharedPref.getInt("userId", -1)
55 val token = sharedPref.getString("token", "")
```

Como se aprecia de forma notable, es necesario especificar un valor por defecto en caso de no ser posible la recuperación de la información.

Solución desarrollada

DURANTE este capítulo se llevará a cabo un recorrido por las principales pantallas de la aplicación desarrollada. El objetivo es mejorar la comprensión del lector sobre la aplicación final.

El acceso a cada una de las funciones se llevará a cabo gracias a un menú *Drawer*, que permitirá la existencia de un menú lateral muy simple e intuitivo, como se muestra en la Figura 8.1. El resto de la navegación consiste en acceder a pantallas concretas que forman parte de una misma funcionalidad. Esto se realizará mediante simples botones.

8.1 Usuarios

En esta sección se mostrarán las diferentes funciones existentes en la aplicación para poder el propio usuario configurar su entorno perfil. Esto implica, gestionar sus observadores, sus caballos, y sus amigos. Además, también se incluyen otras pantallas relacionadas con estos apartados mencionados.

8.1.1 Creación de usuario

En caso de no disponer de un usuario registrado en la aplicación, el jinete podrá realizarlo mediante la introducción de los datos exigidos en la pantalla de registro, mostrada en la Figura 8.2.

8.1.2 Acceso a la aplicación

El usuario podrá hacer uso de la aplicación después de introducir su usuario y contraseña en la típica pantalla para tal proceso, mostrada en la Figura 8.3. En caso de no contar con una cuenta, podrá dirigirse a la pantalla de registro.

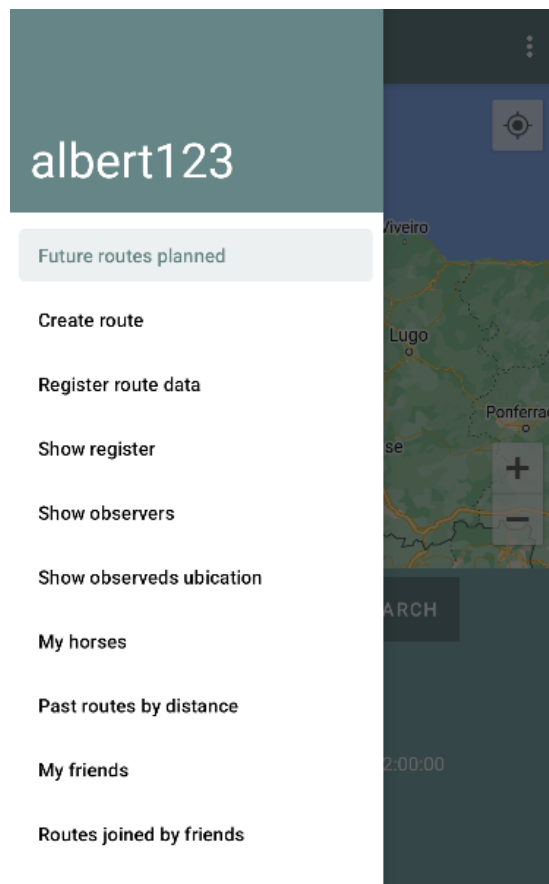
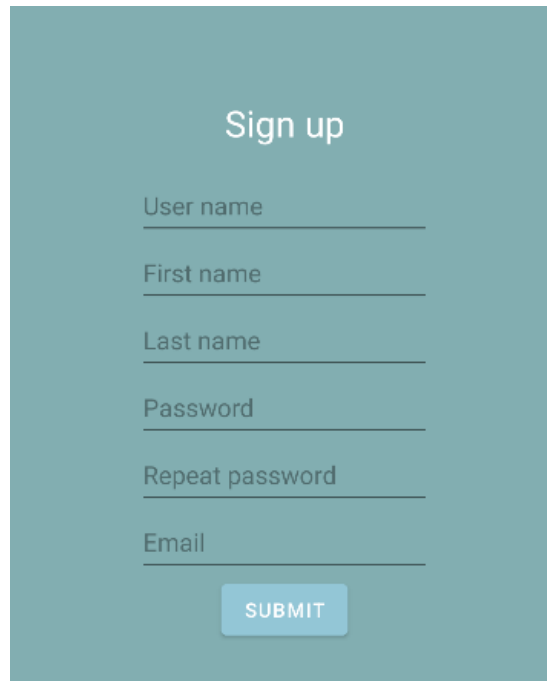


Figura 8.1: Menú drawer.



Sign up

User name _____

First name _____

Last name _____

Password _____

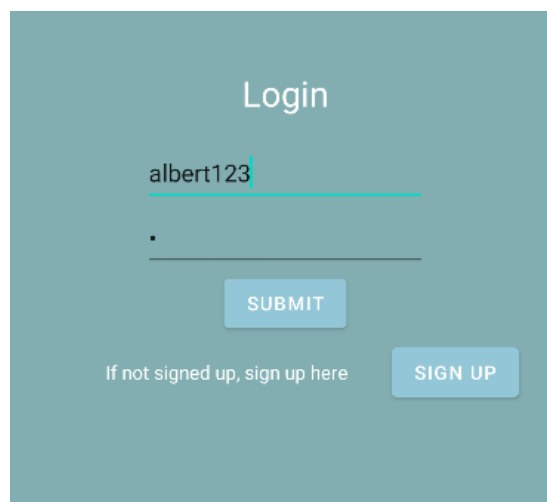
Repeat password _____

Email _____

SUBMIT

The image shows a registration form on a teal background. The title 'Sign up' is centered at the top. Below it are seven input fields, each with a label and a horizontal line for text entry. The labels are 'User name', 'First name', 'Last name', 'Password', 'Repeat password', and 'Email'. At the bottom center is a light blue button with the text 'SUBMIT' in white capital letters.

Figura 8.2: Pantalla de registro.



Login

albert123 _____

• _____

SUBMIT

If not signed up, sign up here

SIGN UP

The image shows a login form on a teal background. The title 'Login' is centered at the top. Below it are two input fields. The first field contains the text 'albert123' and has a red cursor at the end. The second field contains a single dot '•'. Below the second field is a light blue button with the text 'SUBMIT' in white capital letters. At the bottom left is the text 'If not signed up, sign up here' and at the bottom right is a light blue button with the text 'SIGN UP' in white capital letters.

Figura 8.3: Pantalla de acceso.

8.1.3 Gestión de amigos

Los usuarios que el jinete haya agregado como amistades a lo largo del tiempo se mostrarán en la pantalla que se muestra en la Figura 8.4 para su gestión. Desde esta pantalla, podrán ser eliminados como amigos y añadir nuevos, mediante la introducción de su nombre de usuario en el recuadro de la parte inferior de la pantalla.

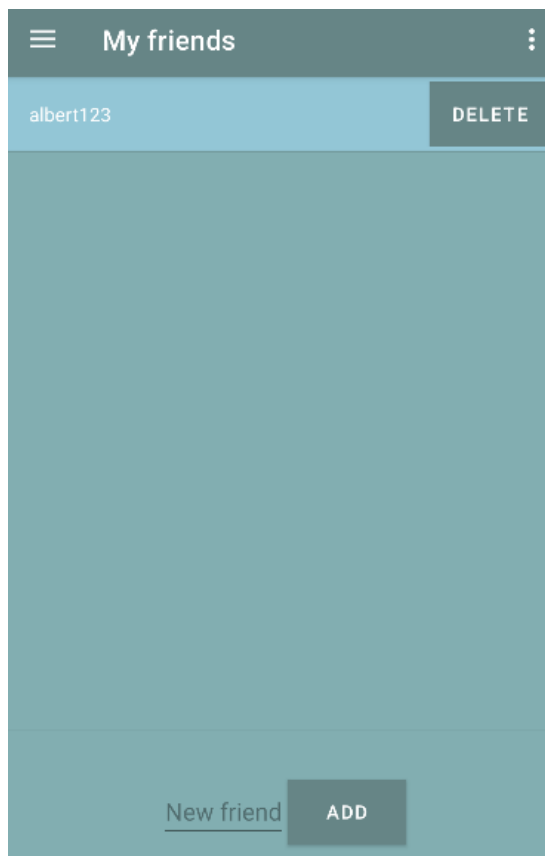


Figura 8.4: Pantalla de acceso.

8.1.4 Gestión de caballos

Como se muestra en la Figura 8.5, se dispone de una interfaz que facilita conocer todos los caballos registrados por el usuario en la aplicación. La manera de añadir un nuevo caballo se observa en la Figura 8.6.

8.1.5 Ubicación de observados

Para asegurar la seguridad de un jinete, la aplicación ofrece una pantalla dónde un usuario puede consultar la última ubicación en una ruta de los jinetes que lo hayan seleccionado como

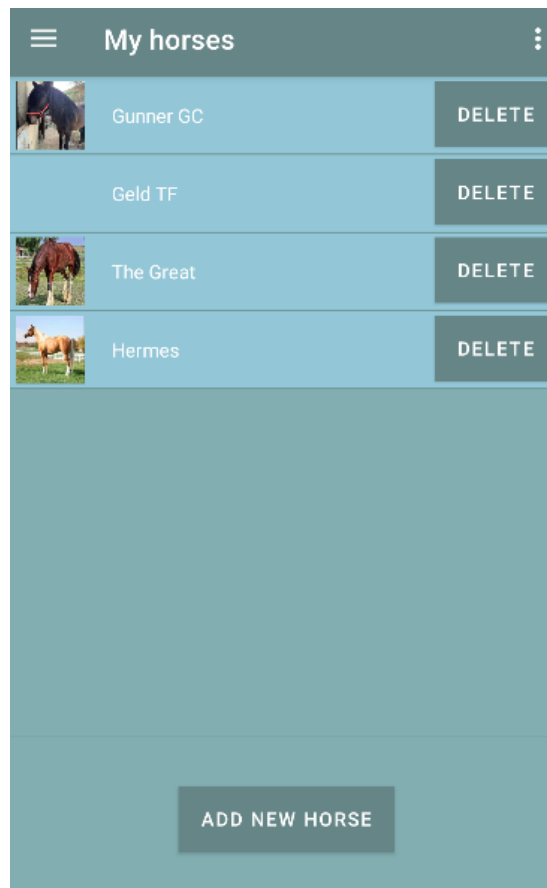
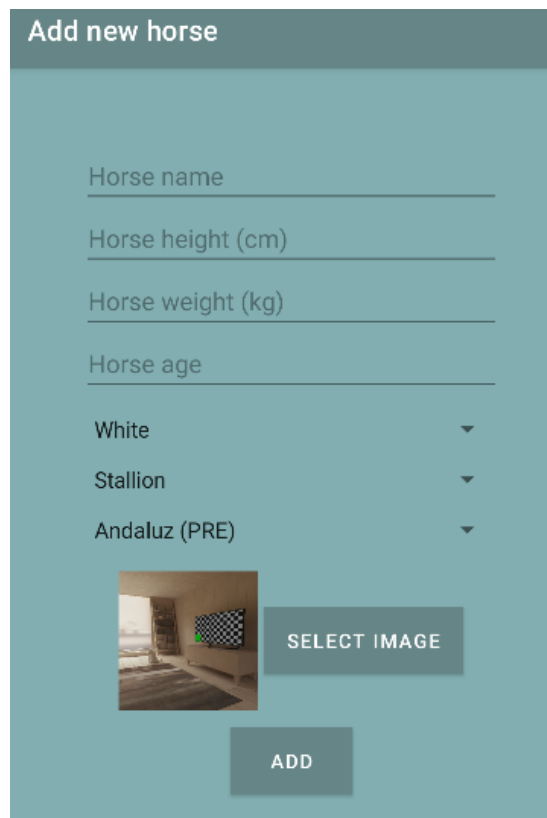


Figura 8.5: Pantalla para ver los caballos del usuario.



The screenshot shows a mobile application interface for adding a new horse. The title bar at the top is dark teal and contains the text "Add new horse" in white. Below the title bar, the form consists of several input fields and dropdown menus, all with a light teal background and dark teal text. The fields are: "Horse name", "Horse height (cm)", "Horse weight (kg)", and "Horse age". Below these are three dropdown menus with the following selected values: "White", "Stallion", and "Andaluz (PRE)". At the bottom of the form, there is a small square image placeholder showing a room interior, a "SELECT IMAGE" button to its right, and an "ADD" button centered below the image placeholder.

Figura 8.6: Pantalla para añadir caballo.

observador. La Figura 8.7 muestra un ejemplo de esta funcionalidad. Cabe destacar que, al usar datos de prueba, la ubicación del observado es la misma que en la que se encuentra el observador en ese momento. La pantalla también consta, por simplicidad de un botón para actualizar la localización de los jinetes.



Figura 8.7: Pantalla para ver observados.

8.1.6 Rutas de amigos

En la aplicación se permite el registro de otros jinetes como amigos. La finalidad de esta funcionalidad es conocer las rutas en las que están inscritos para poder realizarlas juntos. Como se muestra en la Figura 8.8, debajo del nombre de la ruta se destaca el nombre del conocido inscrito en ella. El botón de detalles permite acceder a la misma pantalla de detalles que se mencionó anteriormente, que posibilita unirse a la actividad.

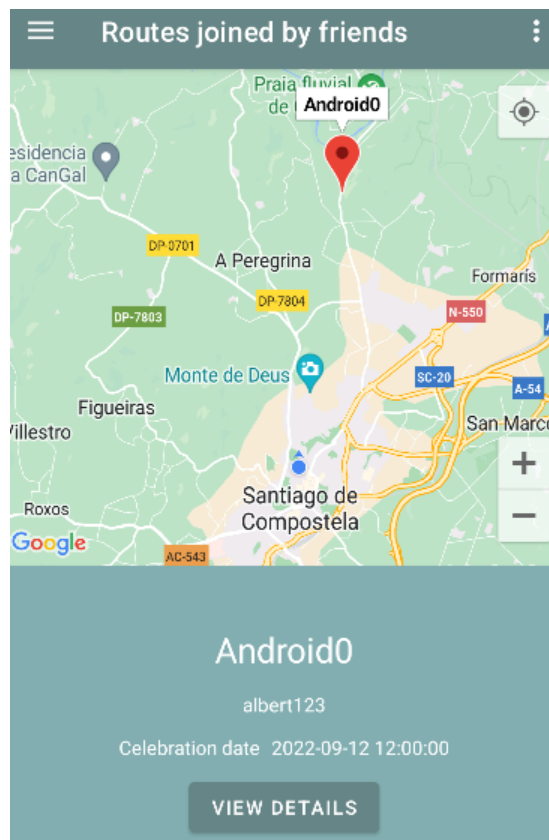


Figura 8.8: Pantalla para ver rutas en las que un amigo está inscrito.

8.2 Rutas

En esta sección se expondrán las funciones relevantes relacionadas con las rutas. Generalmente estas pantallas harán uso de mapas.

8.2.1 Ver rutas futuras

El usuario debe ser capaz de ver cualquier ruta que se vaya a celebrar en un momento futuro desde la invocación de esta función. Se verán marcadores en los puntos dónde esté programada alguna ruta. Al pinchar en uno, se podrá consultar su nombre y se posibilitará la opción de acceder a sus detalles, como se observa en la Figura 8.9.

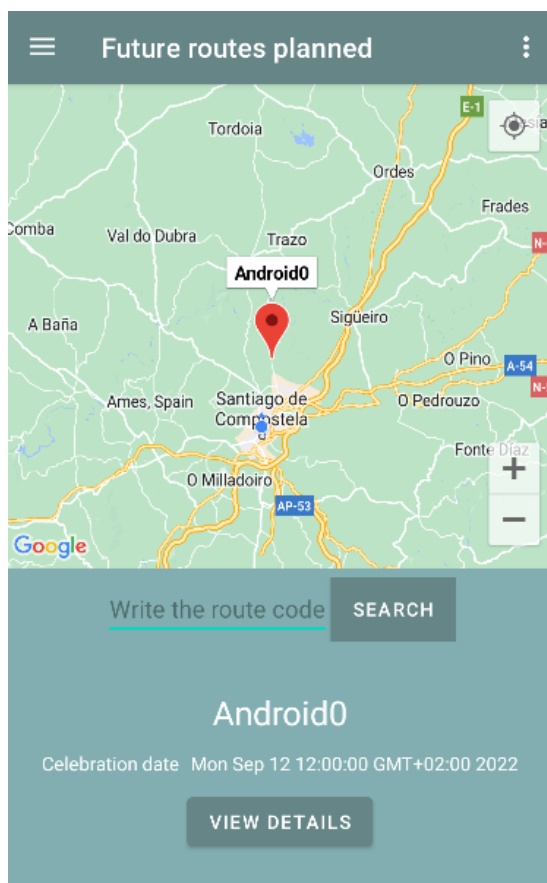


Figura 8.9: Pantalla para ver rutas futuras.

En la pantalla de detalles de una ruta, mostrada en la Figura 8.10, se podrá acceder al *chat*, como el que se observa en la Figura 8.11 y los participantes de la propia ruta, que, a su vez, se puede consultar en la Figura 8.12. Como se puede observar en la anterior Figura, en caso de no poseer imagen el caballo, solo se mostrarán sus datos. El botón que posibilita el unirse

a una ruta solamente es visible para alguien habilitado para tal acción. En caso de haberse inscrito con anterioridad, aparecerá uno botón de abandonarla en su lugar y, en caso de ser el creador de la ruta, no aparecerá ninguna de las dos opciones. Además, solo en caso de ser posible unirse a la ruta, se vería el selector de caballos del usuario, el cual tendría que ser seleccionado para inscribirse.

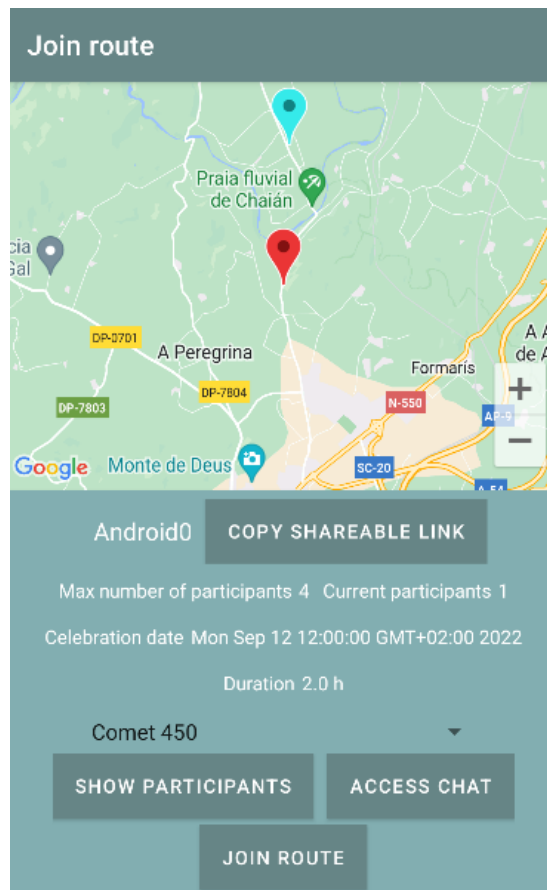


Figura 8.10: Pantalla para ver detalles de una ruta.

8.2.2 Rutas pasadas

Esta función fue ideada para facilitar la creación de nuevas rutas, al entender que pueden existir muchas rutas muy interesantes que hubieran ocurrido en el pasado, o simplemente que algún usuario tenga el interés de crear una, pero no sea capaz de idearla por sí mismo en ese momento. Como se observa en la Figura 8.13, es posible seleccionar la distancia a la que se quiere consultar. Al pinchar en un marcador, se habilitará la opción de actualizar ciertos datos de la ruta para que pueda volver a ser llevada a cabo, tales como la fecha de celebración y el número máximo de participantes.

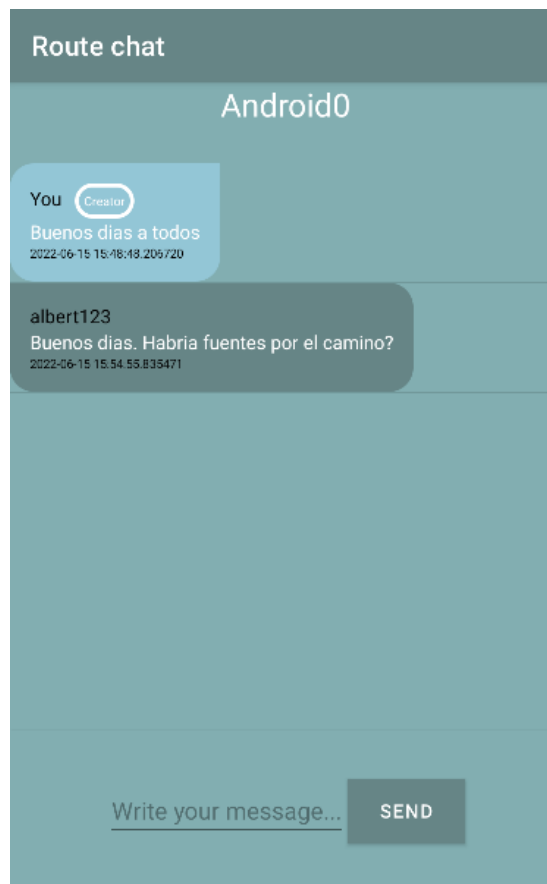


Figura 8.11: Pantalla para consultar chat de una ruta.

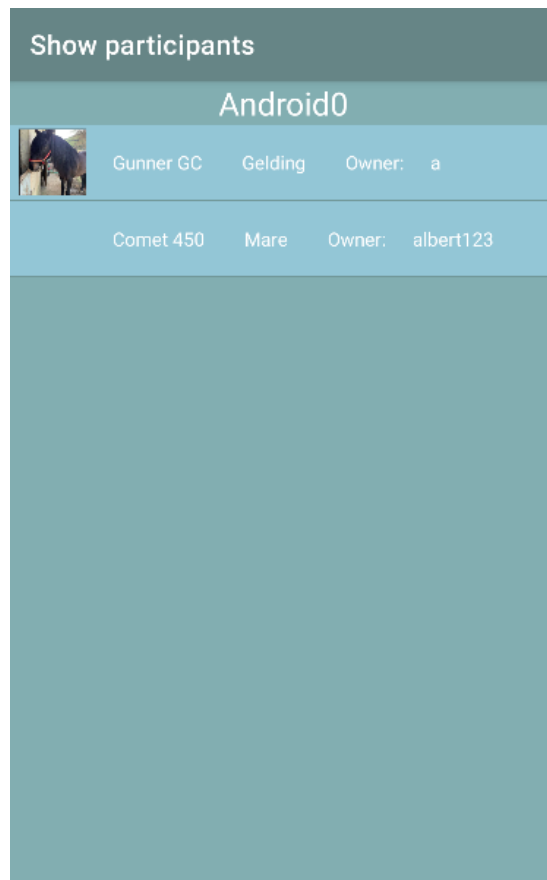


Figura 8.12: Pantalla para ver participantes de una ruta.

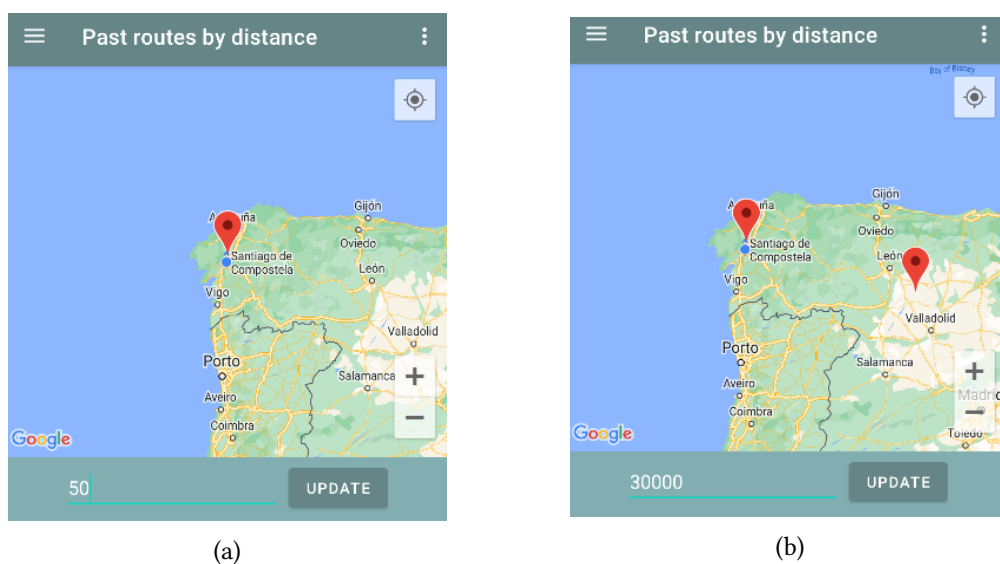


Figura 8.13: Pantalla de consulta de rutas pasadas. (a) Rutas a cincuenta kilómetros. (b) Rutas a una cantidad elevada de kilómetros, en concreto, treinta mil.

8.2.3 Registrar datos de ruta

Esta función permite que un usuario pueda registrar los datos de cualquier ruta en la que participe, independiente de que esta haya sido organizada desde la misma aplicación. Al presionar el botón de empezar, el cronómetro empezará a registrar la duración de la actividad. La aplicación tomará puntos periódicos a medida que pase el tiempo, actualizando la interfaz. Al presionar el botón de finalizar, que aparecerá en lugar del de empezar, los datos serán enviados al servicio de la aplicación. La interfaz mencionada se expone en la Figura 8.14.

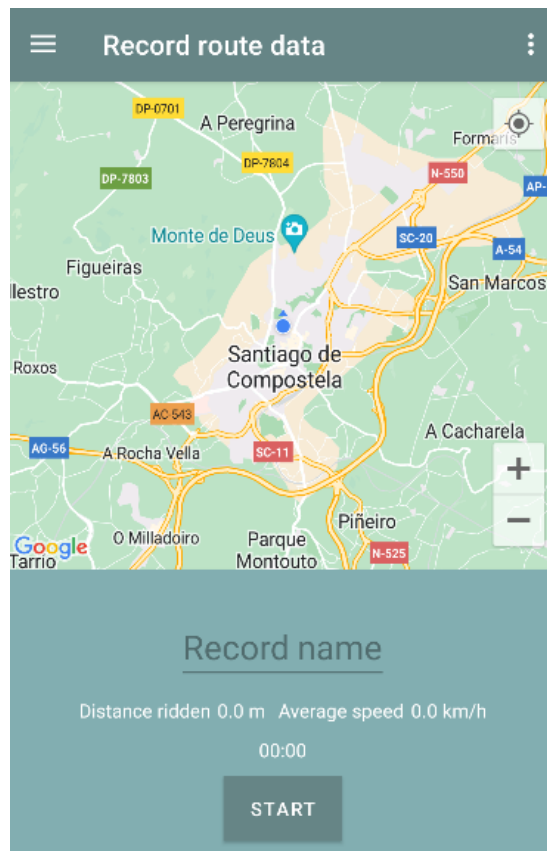
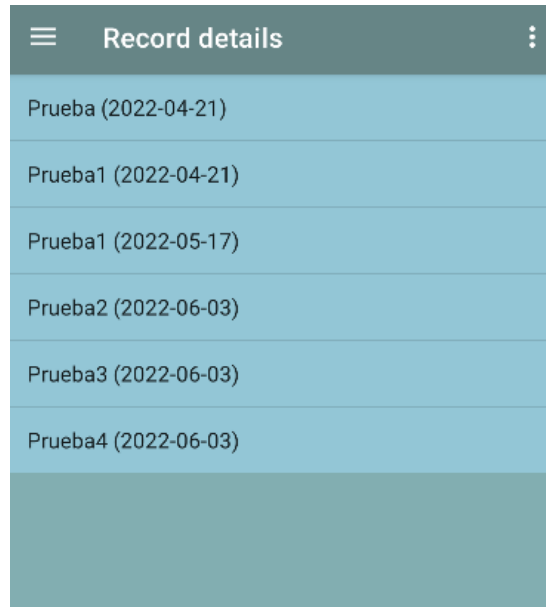


Figura 8.14: Pantalla para registrar ruta.

8.2.4 Registros de rutas

Se podrá acceder a todas las rutas de las que se tomó datos anteriormente. Como se muestra en la Figura 8.15, en caso de dos registros posean el mismo nombre, pueden ser distinguidos por su fecha de realización. Al pinchar en alguno de ellos, se accederá a los detalles de ese registro, como se expone en la Figura 8.16. En esta pantalla se observará la línea que une cada punto por los cuales el usuario pasó en la realización de la actividad. El mapa, además, comen-

zará centrado en el punto desde el que inició la ruta, pudiendo moverse el mapa en caso de que sea necesario para observar el resto del recorrido. A partir de esta pantalla, se podrá acceder al gráfico de altitudes, que representa la altitud de cada punto del registro. Puede observarse en la Figura 8.17. Al tratarse de unos datos de prueba, solo cuenta con tres puntos.



The screenshot shows a mobile application interface with a dark teal header bar containing a hamburger menu icon on the left, the text 'Record details' in the center, and a vertical ellipsis icon on the right. Below the header is a list of six items, each on a light blue background with a thin white border. The items are: 'Prueba (2022-04-21)', 'Prueba1 (2022-04-21)', 'Prueba1 (2022-05-17)', 'Prueba2 (2022-06-03)', 'Prueba3 (2022-06-03)', and 'Prueba4 (2022-06-03)'. Below the list is a solid dark teal rectangular area.

Record details
Prueba (2022-04-21)
Prueba1 (2022-04-21)
Prueba1 (2022-05-17)
Prueba2 (2022-06-03)
Prueba3 (2022-06-03)
Prueba4 (2022-06-03)

Figura 8.15: Pantalla para ver lista de registros.



Figura 8.16: Pantalla para ver detalles de un registro.

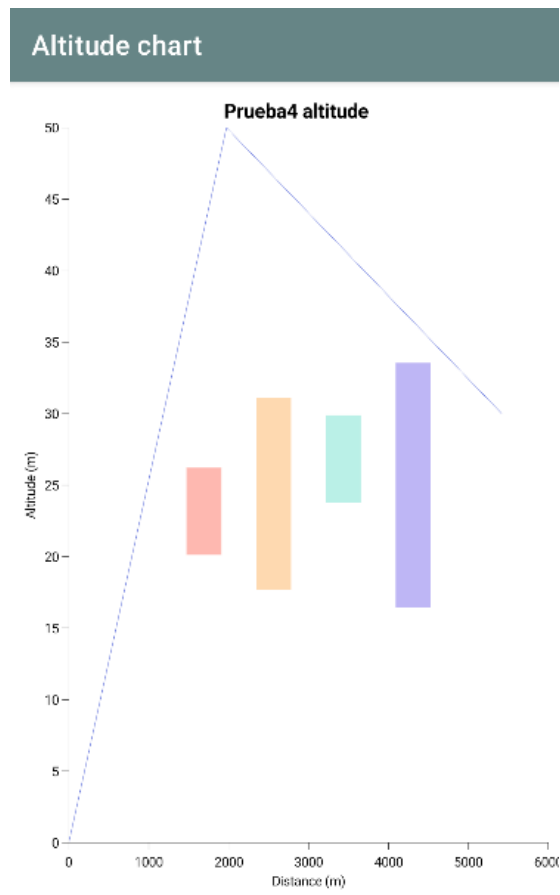


Figura 8.17: Pantalla para consultar gráfico de altitudes.

Conclusiones

ESTE proyecto tenía como principal objetivo proporcionar un servicio de rutas a jinetes que no estaba siendo cubierto por la tecnología actualmente. Consistía en la creación de una herramienta que permitiera la participación conjunta y segura en actividades de rutas de caballo.

Para lograr esto, se ha desarrollado una plataforma que cuenta con un *backend* implementado usando el marco de desarrollo *Django*. El *frontend*, a su vez, debía permitir a los jinetes una gran movilidad, es decir, poder llevar la aplicación con ellos a cualquier localización. Por lo tanto se ha optado por desarrollar una aplicación móvil en *Android*, la cual permite, no solo gestionar los usuarios, peticiones de amistad o rutas, sino también monitorear los recorridos en cada momento, uno de los requisitos críticos de la herramienta.

Debido a las características del proyecto, se seleccionó la metodología *SCRUM* para desarrollarlo. Esto permitió una implementación ágil del software. Además, al ser un proyecto de tamaño reducido y de gran flexibilidad, esta metodología resultaba muy adecuada para el contexto.

El proyecto hizo hincapié en la creación de una aplicación que permitiera al usuario cooperar con otros para realizar actividades. Este objetivo ha sido completado satisfactoriamente gracias a la gestión de rutas implementada, que facilita al usuario toda la información sobre la ruta relevante, además de indicar al resto de usuarios si pueden contar con su asistencia o no. A mayores, como parte de la información de una ruta, se incluyó un chat interno que permite la comunicación entre los participantes o posibles participantes.

La seguridad de la aplicación se ha logrado gracias al uso de las herramientas proporcionadas por el *framework Django*, que facilita la autenticación del usuario. Esto impedirá que un usuario ajeno pueda interactuar con el software en su nombre, mientras el usuario cumpla con las adecuadas medidas de seguridad, como, por ejemplo, resguardar su contraseña.

La aplicación ha puesto su grano de arena para asegurar la integridad física del usuario mediante la implementación de un sistema de vigilancia interno. Este sistema permite el

monitoreo de la situación geográfica del jinete, siempre que este lo permita, de forma que los usuarios que este considere dignos de confianza puedan ser conscientes de dónde se encuentra en caso de ser necesario.

Adicionalmente, para facilitar el uso de la aplicación, se ha implementado un historial de actividades pasadas con el fin de volverlas a celebrar. De esta forma se simplifica el proceso de creación de rutas, permitiendo al usuario acceder a los datos de rutas cercanas celebradas y actualizar sus datos.

Es posible también el estudio de recorridos hechos por el usuario. Se puede acceder a los datos de rutas que el jinete haya decidido conservar. Esto responde a la necesidad de jinetes que tengan como objetivo mejorar sus capacidades, así como asegurarse de que las rutas han sido adecuadas para la salud de sus equinos.

Para favorecer la cooperación, se han implementado funcionalidades con el fin de coordinar jinetes que hayan desarrollado una amistad con anterioridad. Estas funcionalidades son la de consultar las rutas en las que estén inscritos los conocidos y la de poder acceder fácilmente a los detalles de un recorrido gracias a la compartición de un código correspondiente a la ruta.

El proyecto se concretó en una aplicación que proporciona las utilidades requeridas desde su ideación. La aplicación desarrollada cumple todos los objetivos propuestos para esta primera versión. No obstante, la naturaleza de la misma la abre a una gran cantidad de posibilidades futuras que permitirían ofrecer un resultado más completo.

9.1 Trabajo futuro

El proyecto ofrece un gran abanico de posibilidades. Al tratarse de una aplicación dónde se espera la cooperación de distintos usuarios, el posible desarrollo de múltiples funcionalidades sociales podría hacer derivar el proyecto en una red social equina de relevancia. A continuación se mostrarán posibles mejoras en el servicio al usuario:

- Inclusión imágenes geolocalizadas en las rutas por parte de los propios jinetes. Tendría el fin de dar a conocer posibles características de importancia de la ruta a otros usuarios.
- Adición de una nueva pantalla de *feed* en la cual los amigos puedan compartirse rutas, imágenes o cualquier tipo de estados.
- Actualizar diferentes elementos de estéticos a la aplicación. Por ejemplo el cambio de los marcadores genéricos de los mapas por unos personalizados y relacionados con el mundo equino.
- Implementación de otros modos de compartir las actualizaciones de ubicación de seguridad. Esto permitiría seleccionar como observadores a individuos ajenos a la aplicación a

los cuales les llegaría información periódica por medio de aplicaciones como *Whatsapp*, *Telegram* o similares.

- Calendario donde el jinete pueda localizar de una manera sencilla todas las citas futuras.
- Integrar la aplicación con otro software de temática veterinaria para garantizar la salud de los caballos.

Lista de acrónimos

PRE Pura Raza Española. 1

Bibliografía

- [1] R. A. Alvez, “Industria de las carreras de caballos,” 2020, consultado el 2022-06-28. [En línea]. Disponible en: https://2playbook.com/industria-opina/industria-carreras-caballos-camino-espana-debe-volver-recorrer_1529_102.html
- [2] E. I. AB, “Equilab | horse and rider tracking,” consultado el 2022-06-28. [En línea]. Disponible en: <https://equilab.horse/>
- [3] —, “Start using safety tracking - equilab,” consultado el 2022-06-28. [En línea]. Disponible en: <https://support.equilab.horse/hc/en-us/articles/360015590040-Start-using-Safety-Tracking>
- [4] G. P. Films, “Grey pony films and elaine heney,” consultado el 2022-06-28. [En línea]. Disponible en: <https://greyponyfilms.com/films>
- [5] E. Team, “Equisense | datas for riders,” consultado el 2022-06-28. [En línea]. Disponible en: <https://equisense.com/>
- [6] P. G. D. Group, “Postgresql: The world’s most advanced open source relational database,” 2022, consultado el 2022-06-28. [En línea]. Disponible en: <https://www.postgresql.org/>
- [7] P. S. Foundation, “Welcome to python.org,” 2022, consultado el 2022-06-28. [En línea]. Disponible en: <https://www.python.org/>
- [8] E. O. Ltd, “Home - django rest framework,” 2016, consultado el 2022-06-28. [En línea]. Disponible en: <https://www.django-rest-framework.org/>
- [9] J. . O. source Contributors, “Kotlin programming language,” 2022, consultado el 2022-06-28. [En línea]. Disponible en: <https://kotlinlang.org/>
- [10] “Git,” consultado el 2022-06-28. [En línea]. Disponible en: <https://git-scm.com/>
- [11] Microsoft, “Visual studio code for the web,” 2022, consultado el 2022-06-28. [En línea]. Disponible en: <https://code.visualstudio.com/docs/editor/vscode-web>

- [12] Google, “Android studio preview | android developers,” 2022, consultado el 2022-06-28. [En línea]. Disponible en: <https://developer.android.com/studio/preview>
- [13] Gradle, “Gradle build tool,” 2022, consultado el 2022-06-28. [En línea]. Disponible en: <https://gradle.org/>
- [14] P. Inc, “Postman api platform,” 2022, consultado el 2022-06-28. [En línea]. Disponible en: <https://www.postman.com/>
- [15] K. Schwaber, “Home | scrum.org,” 2022, consultado el 2022-06-28. [En línea]. Disponible en: <https://www.scrum.org/>
- [16] “Tutorial django parte 2: Creación del esqueleto del sitio web,” consultado el 2022-06-28. [En línea]. Disponible en: https://developer.mozilla.org/es/docs/Learn/Server-side/Django/skeleton_website
- [17] F. Bianchi, “Colors - the super fast color palettes generator!” consultado el 2022-06-28. [En línea]. Disponible en: <https://colors.co/>
- [18] “Descripción general de volley,” 2021, consultado el 2022-06-28. [En línea]. Disponible en: <https://developer.android.com/training/volley>