

TRABALLO DE FIN DE GRADO
GRADO EN ENXEÑERÍA INFORMÁTICA
Mención en Sistemas de Información

Aplicación móvil para actividades a campo abierto

Alumno: Alberto Ramil Fernández
Directores: Javier Parapar López
Óscar Pedreira Fernández

A Coruña, 14 de febrero de 2018

Datos del trabajo

Título del Trabajo:	Aplicación móvil para actividades a campo abierto
Clase a la que pertenece:	Trabajo Clásico de Ingeniería
Alumno:	Alberto Ramil Fernández
Directores:	Javier Parapar López Óscar Pedreira Fernández

Miembros del tribunal:

Fecha de lectura y defensa: A Coruña, 14 de febrero de 2018

Calificación:

JAVIER PARAPAR LÓPEZ
Profesor de Universidad
Departamento de Computación
Universidade da Coruña

ÓSCAR PEDREIRA FERNÁNDEZ
Profesor de Universidad
Departamento de Computación
Universidade da Coruña

CERTIFICAN: Que la memoria titulada *Aplicación móvil para actividades a campo abierto* ha sido realizada por ALBERTO RAMIL FERNÁNDEZ bajo su dirección y constituye su Traballo de Fin de Grado en el Grado de Enxeñería Informática.

En A Coruña, a 14 de febrero de 2018

JAVIER PARAPAR LÓPEZ
Director

ÓSCAR PEDREIRA FERNÁNDEZ
Director

A mi familia

Agradecimientos

A los profesores Javier Parapar López y Óscar Pedreira Fernández por su paciencia y ayuda a lo largo de este proyecto. Y que junto a Daniel Varcarce hicieron que cambiara mi rumbo en la carrera.

A mis padres por su paciencia.

A mis hermanos Rufo y Jara por aguantarme los fines de semana.

Y Arturo por su ayuda.

Alberto Ramón Ramil Fernández

A Coruña, 14 de febrero de 2018

Resumen

El crecimiento del uso de dispositivos móviles en la vida cotidiana ha ido en incremento en los últimos años. Esto ha hecho que los usuarios también lo quieran usar en deportes al aire libre, como en caza y pesca. En estas actividades, normalmente se necesita recordar puntos clave para poder volver a visitarlos en otras ocasiones o las rutas seguidas para llegar a los mismos. Normalmente estas actividades se realizan en lugares de difícil acceso y un tanto peligrosos, bien por la peligrosa orografía de los accesos como del lugar en si mismo donde se practica.

Estos motivos nos llevaron a proponer este Trabajo de Fin de Grado.

El objetivo es desarrollar una aplicación móvil en Android que permita al usuario realizar un seguimiento de sus jornadas tanto de caza como de pesca y que le permita guardar sus puntos destacados para poder visitarlos en otras ocasiones. Sin olvidar que el poder monitorizar las jornadas conjuntamente siempre es un punto a favor en el tema de la seguridad en estas actividades.

Esta aplicación se realizará a través de una aplicación móvil Android la cual será la parte con la que interactuará el usuario y la cual mostrará la información que guardaremos en un servicio web. En ella usaremos otros servicios propios de Android como Google Maps, Firebase y Location. La información mencionada anteriormente será guardada en un servidor con el cual se comunicará la aplicación móvil mediante una API REST, de modo que siempre esté disponible para el usuario.

El proyecto seguirá una metodología ágil como es la Scrum, que hará que éste esté dividido en una serie de Sprints. Cada Sprint tendrá las siguientes fases análisis, planificación, diseño, implementación y pruebas.

Todo lo anteriormente mencionado será recogido en esta memoria y comentado más profundamente.

Palabras clave

ANDROID, GPS, LOCALIZACIÓN, FIREBASE, SPRING, SEGUIMENTO, SCRUM

Índice general

1. Introducción	7
1.1. Motivación	7
1.2. Objetivos	8
1.3. Estructura de la memoria	8
1.4. Estudio de mercado	9
1.4.1. iHunt Journal	9
1.4.2. My Fishing Companion	11
2. Tecnología	13
2.1. Android SDK	13
2.2. Eclipse	14
2.3. Spring	14
2.4. Android Studio	15
2.5. Postman	16
2.6. Git	16
2.7. Maven	18
2.8. Gradle	18
2.9. Jackson	18
2.10. Hibernate	18
2.11. PostgreSQL	19
2.12. JUnit	19
3. Proceso de ingeniería	21

ÍNDICE GENERAL

3.1.	Scrum	21
3.1.1.	Sprint	21
3.1.2.	Participantes	22
3.1.3.	Cómo funciona el Proceso	22
3.2.	Adaptación la metodología a este proyecto	25
3.2.1.	Participantes	25
3.2.2.	Sprints	26
3.2.3.	Reuniones	26
4.	Análisis	27
4.1.	Análisis de Requisitos	27
4.1.1.	Actores	28
4.2.	Historias de usuario	28
4.2.1.	Casos de uso	29
4.3.	Análisis de riesgos	33
4.3.1.	Riesgos identificados	33
4.3.2.	Planes de contingencia	34
4.3.3.	Seguimiento	34
4.4.	Interfaz y usabilidad	35
5.	Ejecución	41
5.1.	Pasos de ejecución	41
5.2.	Duración y costes	42
5.3.	Sprints	43
5.3.1.	Sprint 1	43
5.3.2.	Sprint 2	43
5.3.3.	Sprint 3	43
5.3.4.	Sprint 4	44
5.3.5.	Sprint 5	47
5.3.6.	Sprint 6	50
5.3.7.	Sprint 7	54
5.3.8.	Sprint 8	54

6. Diseño	55
6.1. Arquitectura	55
6.2. Capa de datos	57
6.3. Aplicación servidor	59
6.3.1. Servicio web	59
6.3.2. Organización dos paquetes	61
6.3.3. Transmisión de la información	61
6.3.4. Gestión de las clases persistentes	62
6.4. Aplicación Móvil	63
6.4.1. Servicios usados	63
6.4.2. Organización de los paquetes	64
7. Implementación	65
7.1. Implementación	65
7.1.1. Aplicación Servidor	65
7.1.2. Aplicación móvil Android	74
7.2. Pruebas	80
7.2.1. Pruebas de unidad	80
7.2.2. Pruebas de integración y aceptación	81
8. Conclusiones y trabajo futuro	83
8.1. Trabajo realizado	83
8.2. Trabajo futuro	84
Índice de Figuras	85
Apéndices	91
A. Glosario	91
B. Bibliografía	93

Capítulo 1

Introducción

En este primer capítulo de esta memoria se comentaran la motivación, los objetivos de la aplicación y la estructura de esta memoria.

1.1. Motivación

Dado el creciente uso de los dispositivos móviles en los deportes al aire libre, como en caza y pesca, se aprecia una creciente demanda de aplicaciones que permitan el registro de estas actividades.

Por otro lado, la seguridad en estas actividades es un dato a tener en cuenta. Durante las jornadas de caza los cazadores caminan por el monte en grupos, esto puede producir que en determinados momentos un cazador no conozca la situación de su compañero. Esta aplicación permitiría conocer la ubicación de cada cazador en todo momento pudiendo minimizar accidentes por disparos fortuitos. Las jornadas de pesca en ocasiones se realizan desde acantilados o lugares muy lejos de carreteras, esto lleva a que si tenemos una pequeña lesión indicar nuestra ubicación a nuestros compañeros sea de gran ayuda.

Según datos del Consejo superior de deportes en España en el año 2016 se tramitaron 343.130 licencias federativas de caza y 54.992 licencias de pesca. A estas personas son a las que iría dirigida esta aplicación. Cabe destacar que existen tanto cazadores como pescadores que realizan sus actividades sin tener que estar federados, por lo que el número de licencias federativas aumentaría.

Por todo esto se propone este Trabajo de Fin de Grado. El objetivo de este proyecto será diseñar y construir una herramienta que permita gestionar las actividades a

campo abierto como caza y pesca, permitiendo mediante geolocalización guardar los puntos de interés, las rutas seguidas, llevar control de los usuarios presentes, guardar información de los éxitos de las jornadas y poder realizar jornadas de caza y pesca con un seguimiento total de nuestros compañeros.

1.2. Objetivos

El objetivo es desarrollar una aplicación móvil en Android que permita al usuario realizar un seguimiento de sus jornadas tanto de caza como de pesca y que le permita guardar sus puntos destacadas para poder visitarlos en otras ocasiones. Sin olvidar que el poder monitorizar la jornadas conjuntamente siempre es un punto a favor para seguridad de los participantes en estas actividades.

Los principales objetivos serán:

- Registro de usuarios y adición de amigos a usuario.
- El usuario podrá conocer su ubicación en tiempo real en un mapa.
- El usuario podrá comenzar y parar una jornada.
- El usuario podrá visualizar la ruta seguida durante la jornada.
- Administrar puntos clave para el usuario.
- Clasificación de los puntos de interés según su tipología.
- Visualizar dichos puntos en un mapa.
- Gestionar grupos de usuarios participantes en la jornada y poder ver su ubicación en tiempo real.

1.3. Estructura de la memoria

La memoria del presente proyecto está estructurada del siguiente modo:

- **Introducción** Contextualiza el proyecto, introduce el tema a tratar y detalla los objetivos del mismo desde un punto de vista global. También comenta la estructura de la memoria.

CAPÍTULO 1.

Introducción

1.4 Estudio de mercado

- **Tecnología** Describe y justifica las principales tecnologías empleadas para desarrollar el proyecto.
- **Proceso de ingeniería** Detallamos el proceso de ingeniería: la metodología empleada, como funcionada, la adaptación de la misma y los participantes.
- **Análisis** Comentamos las funcionalidades de la aplicación, comentando más detenidamente los casos de uso y requisitos.
- **Seguimiento** Comentamos por las fases que pasa el proyecto usando la metodología Scrum.
- **Diseño** Describimos la arquitectura del proyecto.
- **Implementación** Describiremos aspectos concretos destacados y las pruebas realizadas.
- **Conclusiones y trabajo futuro** Comentamos el producto obtenido y futuros cambios que se podrían realizar.

1.4. Estudio de mercado

En esta sección comentaremos otras aplicaciones que presentan funcionalidades similares a las de este trabajo de fin de grado.

1.4.1. iHunt Journal

iHunt Journal es una aplicación de software de cacería que presenta las siguientes características:

- Ver puntos en el mapa, figura 1.1
- Guardar puntos en un mapa, figura 1.2
- Puede grabar sus búsquedas.
- Graba estadísticas.
- Permite planificar las próximas cacerías según meteorología, figura 1.3.
- Puede grabar sus trofeos figura 1.4.

iHunt Journal está disponible desde iPhone y Android .

CAPÍTULO 1.

1.4 Estudio de mercado

Introducción



Figura 1.1: Visualizar mapa con iHunt

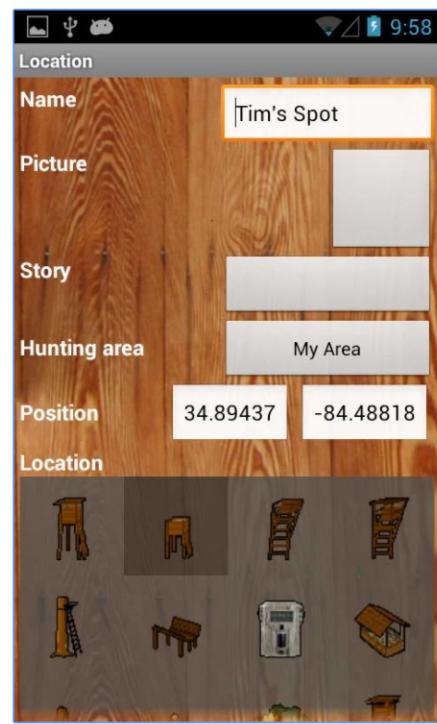


Figura 1.2: Registro de un punto con iHunt



Figura 1.3: Planificación con iHunt



10

Figura 1.4: Registro de trofeo con iHunt

CAPÍTULO 1.

Introducción

1.4 Estudio de mercado

1.4.2. My Fishing Companion

My Fishing Companion se trata de una aplicación para salir de pesca para Smartphone Android. Nos da la posibilidad de:

- Localizar nuestros lugares de capturas véase figura 1.6
- Añadir vídeos y fotos
- Ver el tiempo que nos espera véase figura 1.8
- Marcar capturas véase figura 1.7

También es interesante la comunidad en la que se comparten todos estos datos y que te pueden ayudar a elegir el mejor lugar y el mejor día de pesca.

CAPÍTULO 1.

1.4 Estudio de mercado

Introducción

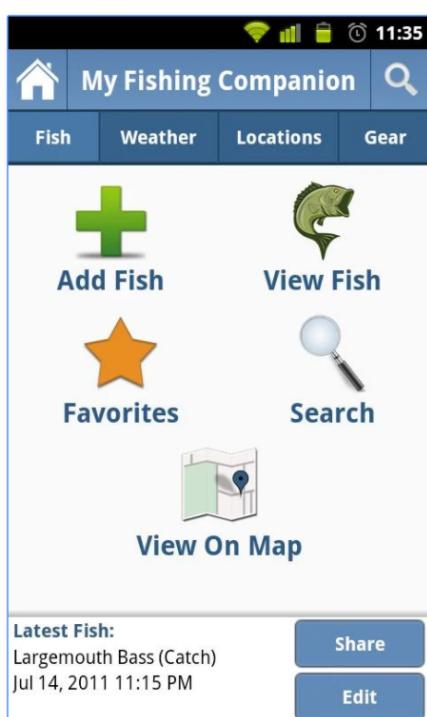


Figura 1.5: Opciones de la aplicación

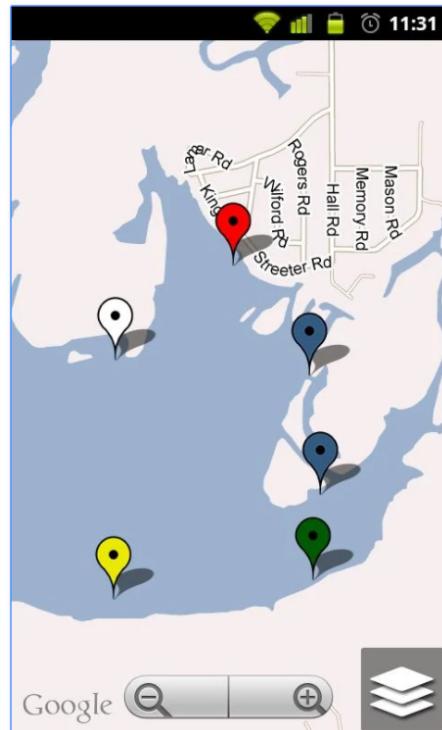


Figura 1.6: Puntos de pesca guardados

This screenshot shows the 'Edit Fish' screen. The top bar has a 'Edit Fish' title and navigation buttons. Below it is a dropdown for 'Select type of event' set to 'Catch'. A dropdown for 'Select fish species' shows 'Largemouth Bass'. There are input fields for 'Pounds' (5) and 'Ounces' (11), and for 'Length (Inches)' (24) and 'Girth (Inches)' (18). Below these are radio buttons for 'Select fish sex': Male (selected), Female, and Unknown. A dropdown for 'Technique Used' shows 'Slow Retrieve'. At the bottom are 'Update' and 'Cancel' buttons.

Figura 1.7: Datos de una captura

12



Figura 1.8: Datos sobre la meteorología

Capítulo 2

Tecnología

Una buena elección de la tecnología será una condición necesaria, aunque no suficiente, para llevar a cabo un proyecto con éxito.

En este capítulo presentamos las tecnologías utilizadas en el desarrollo de este proyecto.

2.1. Android SDK

Android [2] [3] es un sistema operativo especialmente diseñado para dispositivos móviles. Permite el desarrollo de sus aplicaciones en un lenguaje similar al Java. El sistema operativo nos proporciona de manera sencilla todas las interfaces para el desarrollo de las aplicaciones que puedan acceder a las funciones del teléfono.

Android SDK

El SDK de Android ofrece un conjunto de herramientas para el desarrollo de aplicaciones incluyendo:

- **Depurador de código**
- **Simulador de dispositivos de la plataforma. basado en QEMU.** Siendo QEMU un emulador de procesadores.
- **Bibliotecas.**
- **Documentación.**

- Ejemplos de código.

2.2. Eclipse

Eclipse es un IDE en Java para (ver Figura 2.1) para el desarrollo de software. Fue creado por IBM como sucesor de una de sus herramientas y ahora está en manos de la Fundación Eclipse que es quien se encarga de seguir desarrollándolo. Se decidió usar esta herramienta de desarrollo ya que es totalmente gratuita por lo que no encarece el precio del proyecto y porque el entorno que usamos en Android esta basado en este.

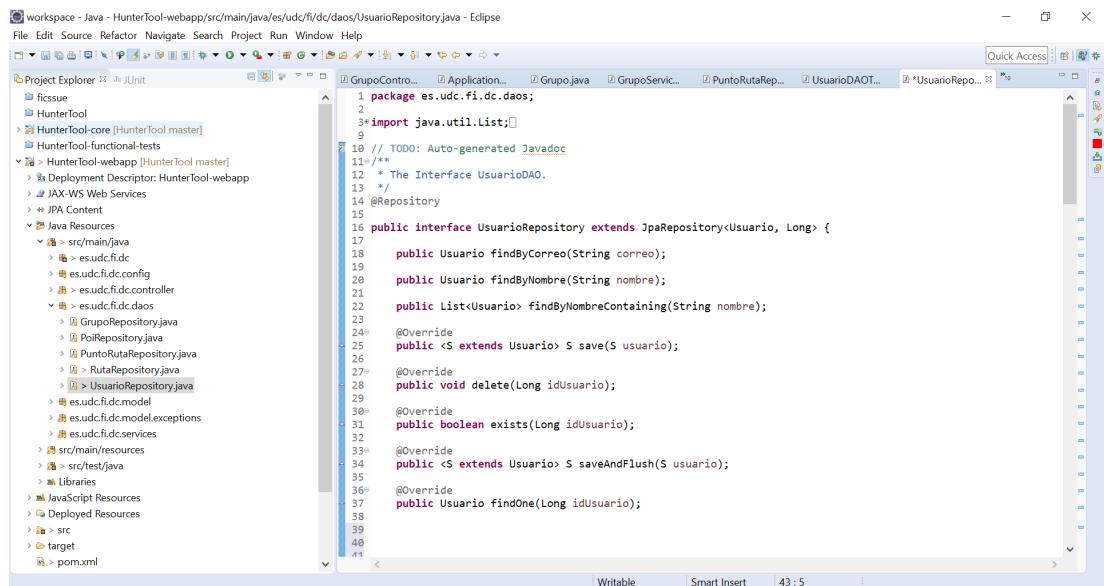


Figura 2.1: Entorno de trabajo Eclipse

2.3. Spring

Spring es un framework [5] de código abierto que tiene como objetivo ayudar al desarrollador a trabajar con otras APIs de manera más sencilla. Nos proporciona un modelo de programación y configuración integral para aplicaciones empresariales basadas en Java, en cualquier tipo de plataforma de implementación.

Las funcionalidades principales que ofrece son:

CAPÍTULO 2.

Tecnología

2.4 Android Studio

- **Programación orientada aspectos**, es un paradigma que nos ayuda separar aspectos trascendentales del resto del código. Lo que conlleva reducir la probabilidad de errores en la codificación o ineficiencias.
- **Inyección de dependencias**, patrón que ayuda a reducir el acoplamiento entre los distintos componentes de la aplicación. Esto se consigue haciendo que una entidad externa proporcione a otra sus dependencias y que esta no tenga que crearlas. Gracias a esto los componentes conocen sus dependencias solamente a través de un interfaz lo que hace posible que la implementación pueda variar de manera transparente para estas clases. Código 2.1 como ejemplo.

```
1 @Service
2 public class GrupoServiceImpl implements GrupoService {
3
4     @Autowired
5     GrupoRepository grupoDAO;
6
7     public void setGrupoDAO(GrupoRepository grupoDAO) {
8         this.grupoDAO = grupoDAO;
9     }
}
```

Código 2.1: Inyección de dependencias

En el código 2.1 se puede ver la anotación `@Service` y `@Autowired`. `@Service` indica que la clase es un componente, concretamente un servicio lo que permite que las clases de implementación sean detectadas automáticamente. `@Autowired` permite que se carguen las dependencias de `GrupoRepository` para poder ser usadas en esta clase.

2.4. Android Studio

Android Studio es el entorno de desarrollo integrado (IDE) oficial para Android que nos ofrece las herramientas necesarias para crear apps en dispositivos Android. Además de ser una potente herramienta para la creación de código permite:

- Realizar compilaciones de nuestro código.
- Renderizado en tiempo real.
- Soporte para construcción basada en Gradle.
- Plantillas para crear diseños comunes de Android.

CAPÍTULO 2.

2.5 Postman

Tecnología

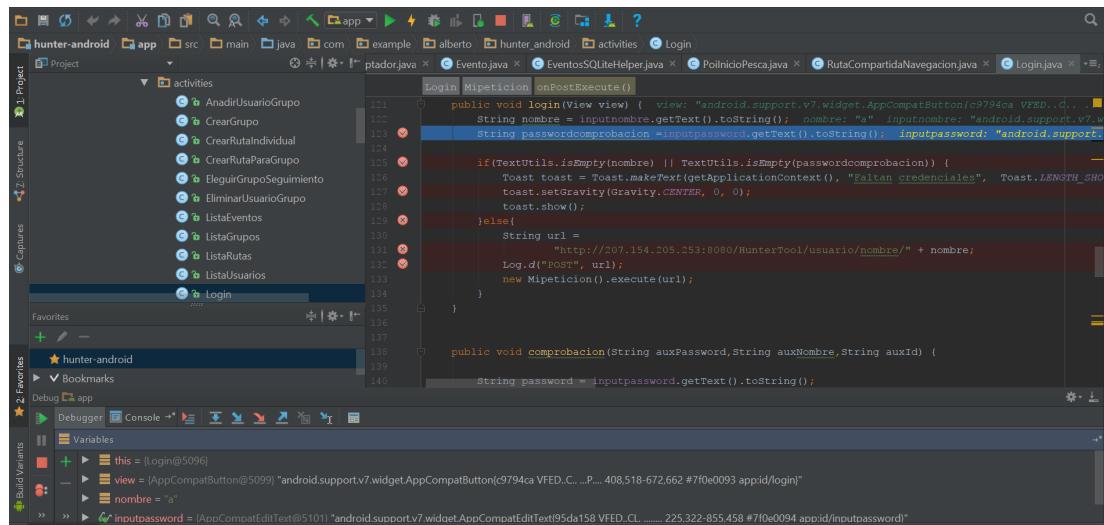


Figura 2.2: Captura de pantalla realizando app debug

- Uso de un depurador que te permite depurar apps que se ejecutan en el emulador de Android o en un dispositivo Android. Figura 2.2. Permitiendo:
 - Establecer puntos de interrupción.
 - Examinar variables en tiempo de ejecución.
- Generar un APK (Android Application Package) para la ejecución de nuestra aplicación móvil tanto de forma simulada, ayudándonos del simulador que nos proporciona o usando un móvil. (Figura 2.3)

2.5. Postman

Es una herramienta que nos permite construir y gestionar peticiones a servicios REST (Post,Get,etc), pudiendo variar en cada petición los header, body y ver los datos devueltos.

2.6. Git

Git27 es un sistema de control de versiones distribuido. Se trata del gestor de versiones moderno más empleado y por ello cuenta con una gran comunidad de desarro-

CAPÍTULO 2.

Tecnología

2.6 Git

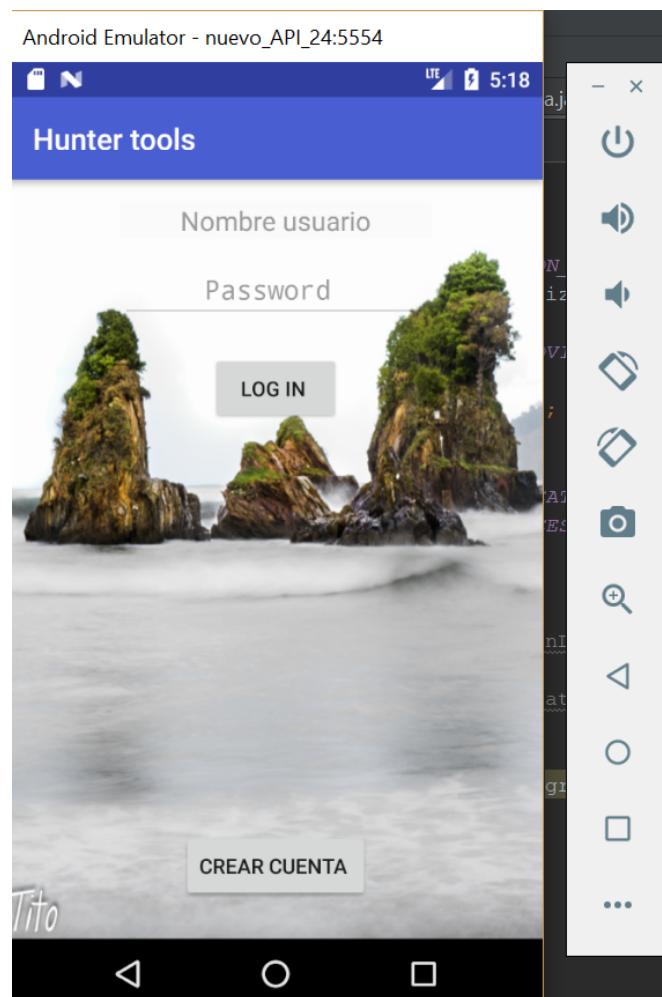


Figura 2.3: Emulador de Android con mi aplicación

lladores a su alrededor lo que favorece la integración de Git con múltiples herramientas.

2.7. Maven

Maven es una herramienta de software para la gestión y construcción de proyectos Java. Maven utiliza un Project Object Model (POM) en formato XML para describir sus dependencias del proyecto. Incluye por defecto con ayudas para la compilación de código o el empaquetado.

2.8. Gradle

Gradle es otra herramienta de software para la gestión, construcción de proyectos y gestor de dependencias. Ofrece herramientas de compilación avanzadas, para automatizar y administrar el proceso de compilación, y al mismo tiempo te permite definir configuraciones de compilación personalizadas.

2.9. Jackson

Jackson es parser para el desarrollo de Servicios Web en Java. Implementa un conjunto de herramientas de procesamiento de datos para Java haciendo que el desarrollo de servicios Rest sean más simples. Se eligió esta opción por lo sencillo que hacia el mapeo de objetos en las peticiones a JSON, transformando JSON en objetos del dominio y viceversa.

2.10. Hibernate

Hibernate [4] es una herramienta de mapeo objeto-relacional para la plataforma Java que ayuda a la transformación de objetos del modelo a una entidad de la base de datos persistente. Este mapeo simplifica el trabajo de desarrollo y evita errores en tareas repetitivas. Este mapeo ayuda a no tener que definir la entidad directamente en nuestro gestor de base de datos. Para conseguir esto debemos usar las anotaciones pertinentes, definir a nivel de base de datos las entidades, restricciones y relaciones.

Una vez puestas las anotaciones los accesos a datos de la bases de datos pasan a ser muy sencillos.

2.11. PostgreSQL

PostgreSQL es SGBD conocido, probado y fiable con un uso muy extendido en la industria. También nos proporciona el PgAdmin que facilita la gestión y administración de bases de datos ya sea mediante instrucciones SQL o con ayuda de un entorno gráfico. Permite acceder a todas las funcionalidades de la base de datos, consulta, manipulación y gestión de datos.

2.12. JUnit

JUnit es el framework de testing para Java más extendido. Permite la ejecución de clases Java para evaluar el comportamiento de los métodos a probar.

Capítulo 3

Proceso de ingeniería

En este capítulo se justifica la elección y se describe la metodología de desarrollo sobre la que se apoya el proceso de ingeniería. En este proyecto se ha empleado una metodología ágil, Scrum.

3.1. Scrum

El Scrum es una Metodología ágil [6] [9] que se usa para estructurar y gestionar proyectos, y para minimizar los riesgos durante la realización del mismo. Esta metodología esta basada en ciclo iterativos lo que se traduce en una gran flexibilidad a la hora de adaptarse a cambios que surjan duran el desarrollo del proyecto. Entre las ventajas se encuentran la baja carga de burocracia y documentación, productividad, calidad y que se realiza un seguimiento diario de los avances del proyecto, logrando una comunicación activa y constante en el equipo de trabajo.

3.1.1. Sprint

Por Sprint entendemos un bloque de tiempo fijo, normalmente de 3 o 4 semanas en el que se crea una versión de nuestra aplicación que se irá incrementando de tamaño según van avanzando los Sprints. Los Sprints se pueden considerar como pequeños proyectos ya que al final de cada uno de ellos debemos tener un producto operativo, es decir, que responda a la historia o historias de usuario planteadas en él.

3.1.2. Participantes

- **Product Owner**

Es el responsable del producto, de priorizar los user stories y de decidir si el sistema las cumple. En general , es un representante del cliente u otro stakeholder

- **Scrum Master**

Es el encargado de liderar la reuniones y ayudar al equipo en los problemas que tengan en la implementación de la metodología. Debe minimizar las imposibilidades que surgen en el desarrollo del proyecto para que se cumplan los objetivos. Además debe promover la comunicación entre los integrantes del grupo para generar la motivación necesaria para conseguir los objetivos en plazo.

- **Scrum Team**

Son los encargados de desarrollar y llevar acabo las tareas de los sprints. En si son quienes realizan las tareas que finalmente generan el producto que se va a entregar.

- **Cliente**

Recibe el producto y puede influir en el proceso, entregando sus ideas o comentarios respecto al desarrollo a través del feedback con el product owner y el equipo sobre las versiones del producto.

3.1.3. Cómo funciona el Proceso

El proceso comienza con la definición del Product Backlog. De este Product Backlog se irán obteniendo las historias para los distintos sprints. El proceso general se puede ver en la figura 3.1. A continuación comentaremos sus partes:

- **Definición del Product Backlog.** Figura 3.2 Es una lista con las funcionalidad de la aplicación. Estas funcionalidades que se definen en el Product Owner son las historias del usuario, es decir, es lo que el usuario quiere hacer en la aplicación y por tanto en muchos casos cada historia puede contener varios casos de uso. Está elaborado por el Product Owner y las funcionalidad están ordenadas de mayor a menor importancia para el cliente. La finalidad del Product Owner es plantear lo que hay que hacer. Con las historias ordenadas se indicarán los Sprints que serán necesarios pudiendo hacer varias historias en un mismo Sprint pero nunca una historia dividida en dos Sprints.

CAPÍTULO 3.

Proceso de ingeniería

3.1 Scrum

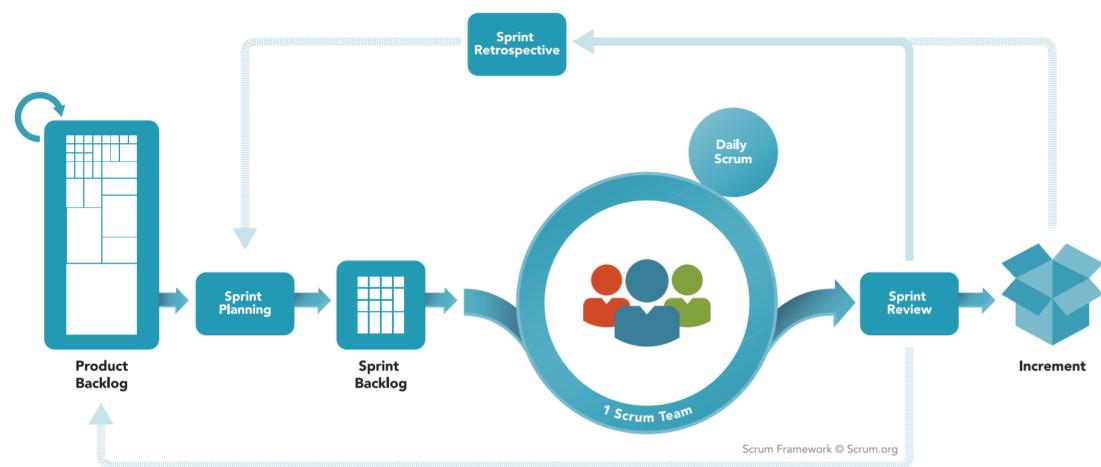


Figura 3.1: Metodología Scrum

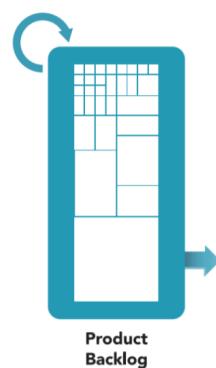


Figura 3.2: Parte metodología Scrum, Product Backlog

- **Sprint Planning Meeting.** Figura 3.3

Esta reunión se hace al comienzo de cada Sprint y se define cómo se va a enfocar el proyecto, se revisan las historias con mayor prioridad y se decide el Sprint backlog.



Figura 3.3: Parte metodología Scrum, Sprint Planning

- **Sprint Backlog.** figura 3.4

Es el conjunto de historias del Product Backlog que se decidirán hacer en el Sprint, estando ordenadas de mayor a menor prioridad. Una vez completado los miembros del equipo dividirán las historias en tareas más pequeñas y más manejables, si es necesario.

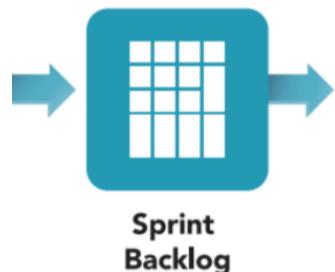


Figura 3.4: Parte metodología Scrum, Sprint Backlog

- **Daily Scrum o Stand-up Meeting. Figura 3.5**

Es una reunión breve que se realiza cada mañana mientras dura el periodo de Sprint. En estas reuniones si algún miembro del equipo encuentra algún proble-

CAPÍTULO 3.

Proceso de ingeniería

3.2 Adaptación la metodología a este proyecto

ma se comenta y se trata de buscar la solución. El scrum master es el encargado de lidiar con los inconvenientes encontrados.



Figura 3.5: Parte metodología Scrum, Daily Scrum

■ **Sprint Review**

Esta sería la primera reunión una vez acabado el Sprint. En ella se deberían plantear las tareas acabadas y se debería ver un avance claro para ser presentado al cliente. Las tareas inacabadas las deberían ser devueltas al Product Backlog y si hiciera falta volver a calcular las prioridades.

■ **Sprint Retrospective**

El equipo revisa los objetivos cumplidos del Sprint terminado. Se anota lo bueno y lo malo, para no volver a repetir los errores. Esta etapa se centra en el cómo se realiza el proyecto para si hay algún error en el desarrollo poder resolverlo en el siguiente Sprint.

3.2. Adaptación la metodología a este proyecto

Debido a la naturaleza de un Trabajo de fin de grado debimos realizar una serie de cambios o adaptaciones en ciertos elementos como fueron los siguientes :

3.2.1. Participantes

Los participantes fueron los siguientes:

- El Product Owner interpretado por los directores.
- El cliente tambien fue desempeñado por los directores.

- El Scrum Master no se usó puesto que se decidió user scrum program.
- Y el equipo solo estuvo formado por el alumno.

3.2.2. Sprints

Cada sprint siempre tuvo una duración establecida de 4 semanas y 5 horas cada día aunque la carga de trabajo variaba en algún Sprint.

3.2.3. Reuniones

En el apartado de las reuniones siempre se establecían las tareas a realizar en cada jornada de trabajo. Al finalizar cada Sprint se realizaban reuniones con los directores y se aprovechaba para la planificación de la siguiente.

Una vez adaptada la metodología a este proyecto encontramos las siguientes ventajas a su uso:

- Ayuda a estar más centrado en el producto durante el desarrollo del proyecto ya que no existe la preocupación por establecer las siguientes tareas. Estas ya quedaron indicadas en el sprint.
- En ocasiones al ir avanzando en el desarrollo aparecen nuevas funcionalidades que se pueden ir añadiendo al Product Backlog. Esto lo podemos hacer siempre y cuando nos ciñamos al Sprint Backlog para saber la tarea que realizar en ese momento.
- Las historias de usuario se irán dividiendo en tareas que se puedan realizar durante el sprint. De este modo una vez acabado el sprint tengamos un producto potencialmente entregable.

Dado el poco conocimiento de la metodología Scrum antes de la realización del proyecto, el ir avanzando en el proceso ayuda a adquirir conocimientos. Los cuales ayudarán a estimar mejor la duración e importancia de los elementos que se añaden al Sprint Backlog.

- Durante el desarrollo del proyecto hay ciertas funcionalidades que pueden cambiar. Empleando los principios de las metodologías ágiles en cada iteración haremos que el proyecto sea mas adaptable antes nuevos requisitos y cambios.

Capítulo 4

Análisis

4.1. Análisis de Requisitos

A la hora de cumplir los objetivo de este proyecto, se establecieron una serie de requisitos. Estos requisitos fueron obtenidos al hacer una entrevista con los directores del proyecto. A lo largo del proyecto y siguiendo la metodología, se fueron refinando y añadiendo requisitos. El registro del usuario sería el primero para poder darle acceso y comenzar a usarla, ya que sin él no tiene acceso a ninguna funcionalidad de la aplicación.

Después surgió la creación de puntos de interés marcados en un mapa con nombre, descripción y un punto en el mapa con o sin señal GPS clasificados en tipo, caza o pesca.

Creación y almacenamiento de las rutas seguidas por un usuario en sus caminatas por cualquier tipo de terrero.

Adicionalmente, también se completó la posibilidad de que el usuario pueda crear grupos con los usuarios que le parezca oportuno y los integrantes del mismo poder añadir a otros. El resultado de esta funcionalidad es la que permitirá posteriormente crear rutas conjuntas ya para crear una ruta conjunta primero se elige el grupo del que se hará el seguimiento. Una vez elegido se enviarán las invitaciones para participar en él a cada integrante del grupo. Estas invitaciones en el caso de ser aceptadas permitirán al usuario navegar por un mapa y periódicamente se irán realizando actualizaciones de las posiciones del resto de integrantes. Esta funcionalidad tendrá un especial interés en las jornadas de caza ya que el conocer en todo momento la posición del resto de compañeros evita peligros innecesarios. Con todo ello se podrán ver las

rutas conjuntas igual que las individuales.

4.1.1. Actores

Los únicos actores que se presentan en la aplicación son los siguientes:

- **Usuario no autenticado.** Usuario que no está autenticado en la aplicación y que solo se le permite registrarse en el sistema o iniciar sesión si ya se registró en otro momento.
- **Usuario autenticado.** Usuario autenticado que puede acceder a todas las funcionalidades del sistema.

4.2. Historias de usuario

Las historias de usuario son los requisitos vistos desde el punto de vista del cliente, es decir, acciones que el usuario llevará a cabo durante el uso de la aplicación. Son la unidad básica detrabajo y se caracteriza por ser:

- Independientes
- Negociables
- Estimables
- Pequeñas
- Tangibles

Las historias en este proyecto serán las siguientes:

- Diseño capa intermedia
- Implementación del servicio Rest
- Iniciar sesión
- Gestión de puntos de interés (PDI)
- Gestión de grupos

CAPÍTULO 4.

Análisis

4.2 Historias de usuario

- Iniciar una ruta individual
- Iniciar la ruta compartida
- Realización de la memoria.

Estas historias anteriormente citadas serán divididas en cada Sprint en pequeñas tareas mas fáciles de manejar. Una tarea es la acción que debe implementar el desarrollador para que se pueda ejecutar parte de la historia, esta está en un lenguaje técnico. Estas tareas reflejan en la mayoría de los casos los denominados casos de uso que comentaremos posteriormente.

4.2.1. Casos de uso

A continuación, en esta sección, se exponen los requisitos funcionales que surgen de los requisitos generales planteados en el punto anterior.

• Usuario no autenticado

- **R1 Registrarse en la aplicación.** El usuario podrá darse de alta en el sistema introduciendo sus datos en el formulario que se le indican. Una vez registrado se cambia de pantalla para el usuario inserte los datos.
- **R2 Iniciar sesión en la aplicación.** El usuario ya registrado podrá, con sus credenciales, autenticarse en el sistema. Se pedirá el nombre del usuario y su contraseña. Se guardará el estado en el terminal hasta que el usuario decida desconectarse.

Estos dos casos de uso quedan reflejados en la figura 4.1.

• Usuario autenticado

Para el caso que el usuario ya esté autenticado dividiremos en 3 grupos los casos de usos, los cuales podremos ver en la figura 4.2:

■ Gestión de puntos de interés

Aquí se describirán los casos de uso relacionados con la gestión de la información del los puntos de interés.

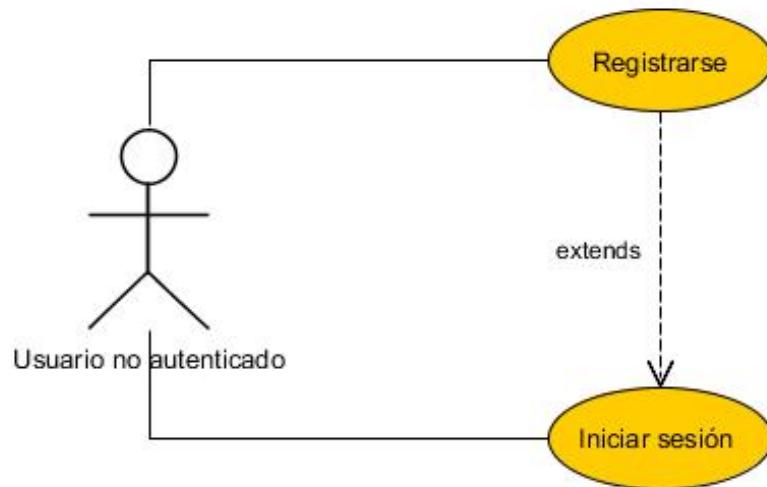


Figura 4.1: Casos de uso del actor Usuario No Autenticado

- **R-PDI-1 Guardar Punto De Interés caza**, el usuario podrá guardar un punto concreto, de caza, asociado a un par de coordenadas pudiendo añadirle un nombre y una descripción.
- **R-PDI-2 Guardar Punto De Interés pesca**, el caso de uso es similar al de anterior pero este es para el tipo de pesca.
- **R-PDI-3 Eliminar PDI**, el usuario podrá seleccionar un punto o una lista de puntos para ser borrados.
- **R-PDI-4 Buscar los PDI**, permite ver todos los puntos de interés de cada tipo en un mapa y pudiendo clicar en ellos para conocer su nombre y descripción.

CAPÍTULO 4.

Análisis

4.2 Historias de usuario

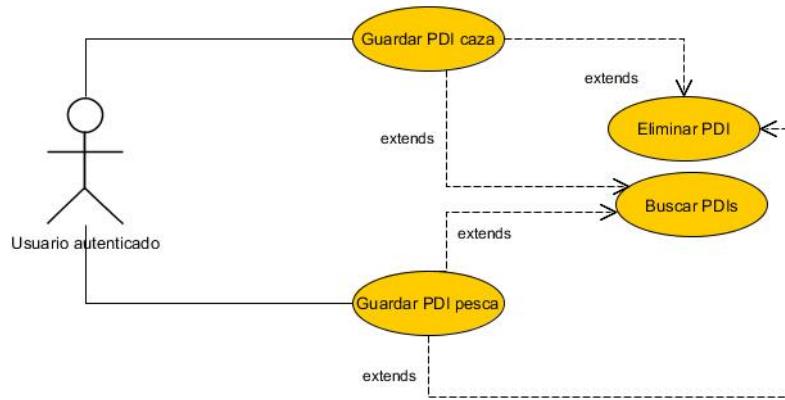


Figura 4.2: Casos de uso de gestión de puntos de interés

■ Gestión de grupos. Ver figura 4.3

- *R-G-1 Crear grupo*, el usuario crea un grupo con nombre único.
- *R-G-2 Añadir integrantes*, el usuario busca los usuarios que quiere integrar en el grupo previamente seleccionado.
- *R-G-3 Eliminar integrantes*, el usuario puede eliminar los integrantes que considere.
- *R-G-4 Ver grupos*, el sistema lista los grupos a los que el usuario pertenece.
- *R-G-5 Ver integrantes grupo*, el sistema permitirá ver los integrantes del grupo que el usuario indique, previo listado del caso de uso R-G-4.

■ Gestión de rutas. Ver figura 4.4

- *R-R-1 Crear ruta privada*, el usuario registra la ruta con un nombre.
 - *R-R-1.1 Iniciar ruta*, el sistema comienza a guardar las coordenadas por la que el usuario está desplazándose y dibujando la ruta en el mapa. Las coordenadas se irán guardando periódicamente, no solo al finalizar la ruta y visualizarla.
 - *R-R-1.2 Parar ruta*, permite parar la navegación, tanto de guardar las coordenadas como de pintar la ruta seguida.
 - *R-R-1.3 Reanudar ruta*, permite reanudar la navegación.
 - *R-R-1.4 Guardar ruta*, el sistema guarda los últimos puntos que quedaban sin actualizar y ejecuta el caso de uso ver ruta en el mapa(R-R-4).

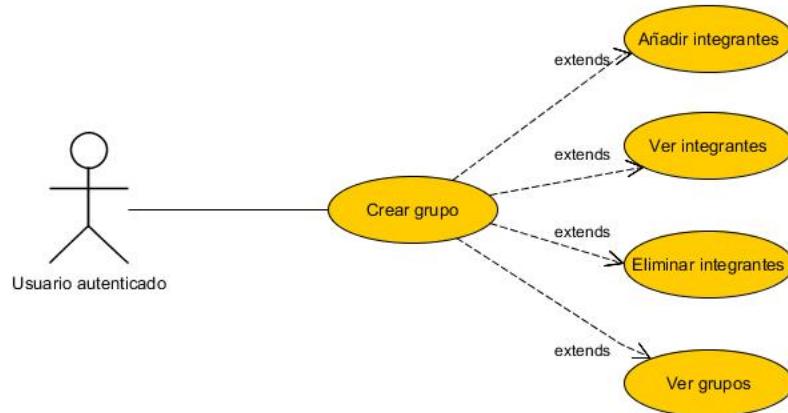


Figura 4.3: Casos de uso de gestión de grupos de usuarios

- **R-R-2 Crear ruta compartida**, este caso de uso permite guardar la ruta seguida por el usuario y al mismo tiempo ver la posición del resto de integrantes de un grupo, anteriormente seleccionado, en tiempo real. Este caso de uso también enviaría a los integrantes del grupo una invitación a dicha ruta.
 - **R-R-2.1 Iniciar ruta**, se comienza a guardar y dibujar la ruta en el mapa. Por otra parte se comienza el seguimiento del resto de usuarios que estén también desplazándose, como también una actualización parcial de la ruta seguida en el servidor.
 - **R-R-2.2 Parar ruta**, se para la navegación y se deja de actualizar la posición al resto de usuario de la ruta compartida.
 - **R-R-2.3 Reanudar ruta**, se reanuda la navegación y se actualizar la posición al resto de usuario de la ruta compartida
 - **R-R-2.4 Finalizar ruta**, se guardan los puntos que faltan de enviar al servidor y se deja de enviar datos al resto de integrantes.
- **R-R-3 Listar rutas**, permite al usuario ver todas las rutas realizadas tanto de manera privada como de manera compartida.
- **R-R-4 Ver ruta en mapa**, el sistema dibuja en un mapa la ruta seguida y previamente seleccionada.
- **R-R-5 Eliminar ruta**, permite al usuario borrar de la aplicación la ruta indicada. Dado que el tratamiento de rutas compartidas una vez guardadas es igual a la de rutas individuales, el borrado se hace igual.

CAPÍTULO 4.

Análisis

4.3 Análisis de riesgos

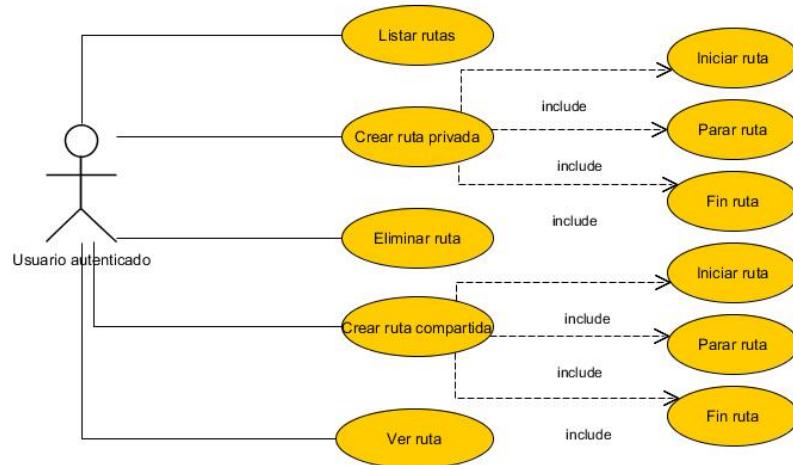


Figura 4.4: Casos de uso de gestión de rutas

4.3. Análisis de riesgos

Un aspecto fundamental que debe gestionarse a lo largo del desarrollo de un proyecto son los riesgos. Mediante un tratamiento de la incertidumbre podremos evitar las posibles la exposición y el impacto.

4.3.1. Riesgos identificados

Código	Descripción	Probabilidad	Impacto	Exposición
R1	Curva aprendizaje Android	Alta	Alto	Alta
R2	Precisión en la obtención de coordenadas GPS	Media	Medio	Media
R3	Adaptación de la metodología Scrum en el proyecto	Media	Medio	Media

Figura 4.5: Identificación y clasificación de riesgos del proyecto

Una vez identificados y clasificados los riesgos debemos realizar dos acciones diferentes dependiendo del riesgo que conlleven:

4.3.2. Planes de contingencia

Los riesgos que tengan una alta exposición pueden hacer peligrar el futuro del proyecto por ello, debemos analizarlos en fases tempranas, antes de comenzar el diseño o la implementación o vigilarlos. En el caso de este proyecto la manera de solventarlos fue diferente. A continuación se comenta el R1.

- **R1 - Curva aprendizaje Android.** Este riesgo era algo que se conocía antes de comenzar ya con el proyecto. Por ello era algo que se tenía muy en cuenta. Para atenuar este riesgo antes de comenzar con el proyecto se realizaron una serie de cursos on-line en *Pluralsight* ya que era la manera de menguar esta curva de aprendizaje.

4.3.3. Seguimiento

Para el resto de riesgos se hará el seguimiento.

- **R2 Precisión en la obtención de coordenadas GPS**

Este riesgo venía dado por el temor a que el GPS del móvil no fuera lo suficientemente preciso en lugares donde se podían realizar rutas. Esto era algo comprensible ya que los GPS de los móviles no son muy muy precisos y en el caso de pintar la rutas seguidas podían producir algún que otro error. Un problema encontrado fue que las coordenadas cambiaban repentinamente sin que el usuario se desplazara tan rápido. Este problema fue solventado quitando las coordenadas que variaran demasiado en un corto periodo de tiempo, esta solución será comentada en el capítulo de implementación de la aplicación.

- **R3 Adaptación de la metodología Scrum en el proyecto**

Antes de la realización de este proyecto mi conocimiento sobre la metodología Scrum era bastante escaso. Para solucionar esto antes del comienzo del proyecto hubo un periodo de familiarización con esta metodología.

4.4. Interfaz y usabilidad

Una vez obtenidos los casos de uso, comenzamos creando las maquetas de cómo debería ser la interfaz del usuario. Se busca que sea una interfaz simple intuitiva. Para hacer esto posible seguiremos las pautas y usaremos los elementos de Material Design [1].

Material Design es un lenguaje de diseño para distintas plataformas y dispositivos. Ofrece una guía que se debe seguir para que los usuarios tengan una experiencia común y habitual entre distintas aplicaciones.

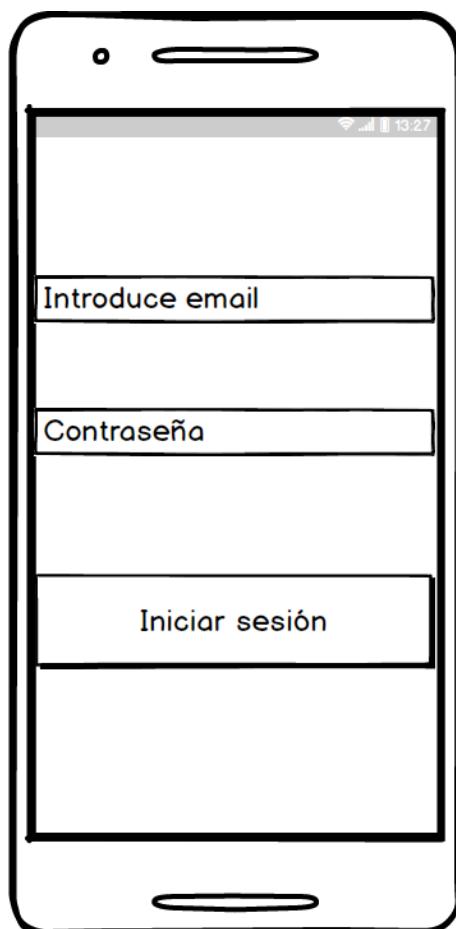


Figura 4.6: Iniciar sesión

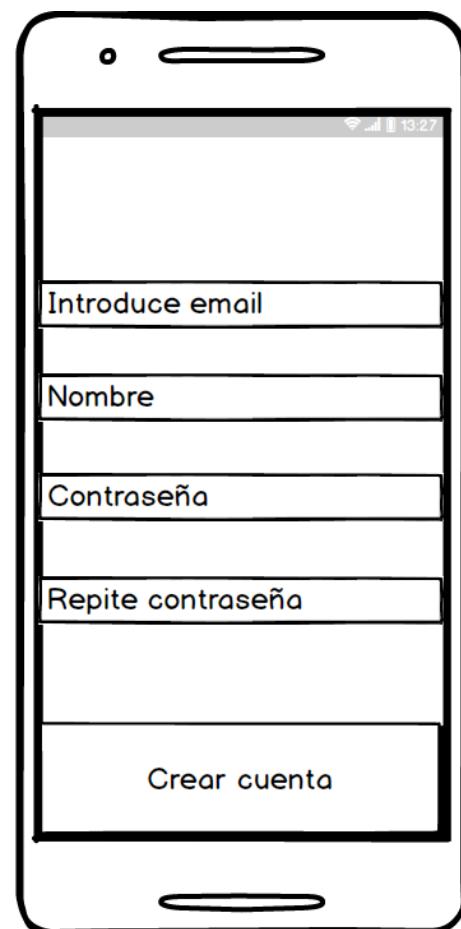


Figura 4.7: Registrar usuario

En la Figura 4.6 podemos ver la maqueta para el inicio de la sesión en la que se

piden los datos del email y la contraseña, mientras que en la Figura 4.7 vemos los datos necesarios para registrar al usuario en la aplicación.

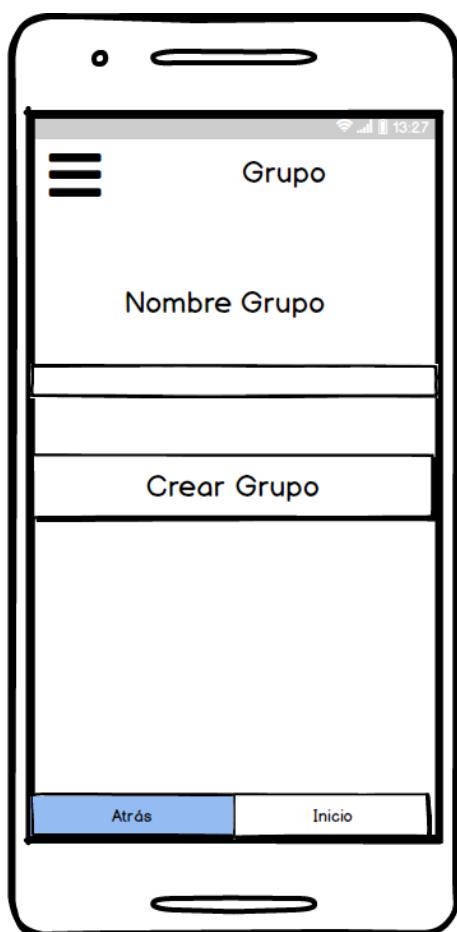


Figura 4.8: Crear grupo

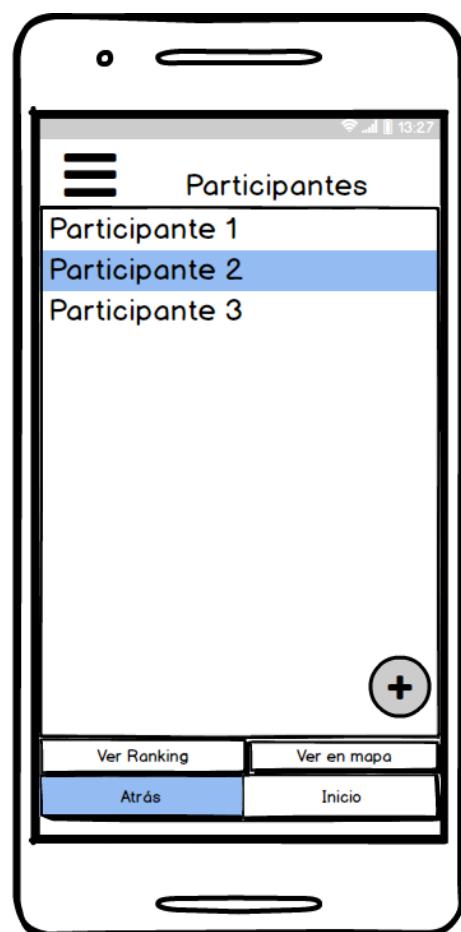


Figura 4.9: Añadir usuarios a grupo

En las figuras 4.8 y en la 4.9 podemos ver como se crearía un grupo y después como se añadirían los usuarios al grupo creado.

CAPÍTULO 4.

Análisis

4.4 Interfaz y usabilidad

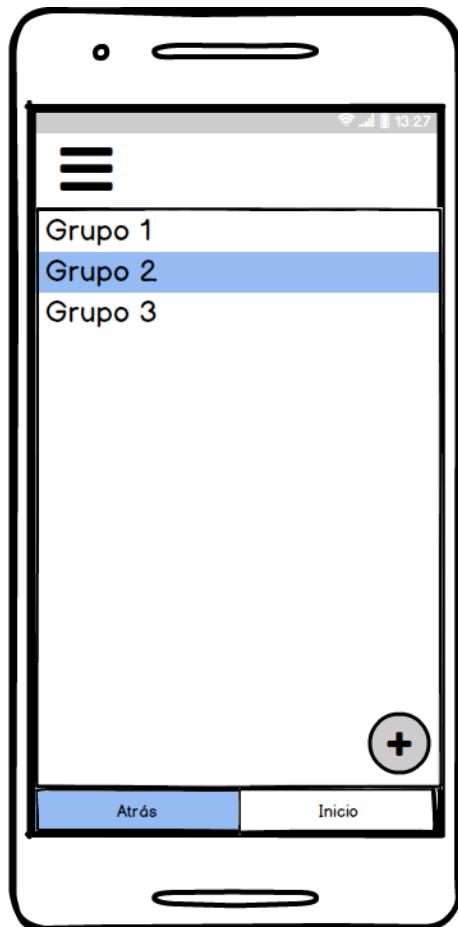


Figura 4.10: Listar grupos

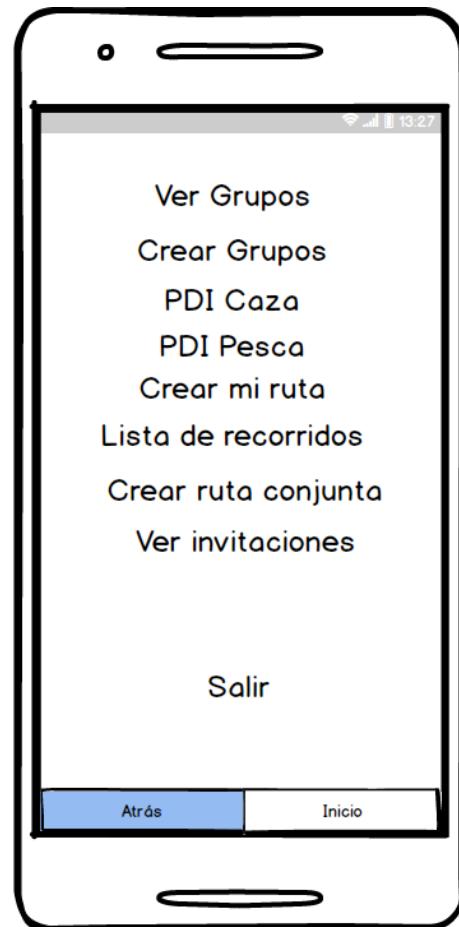


Figura 4.11: Opciones generales

En la figura 4.10 vemos como se mostrarían los grupos a los que pertenece el usuario y a continuación vemos la maqueta de como serían las opciones que tiene el usuario para interactuar con la aplicación, como vemos en la figura 4.11.

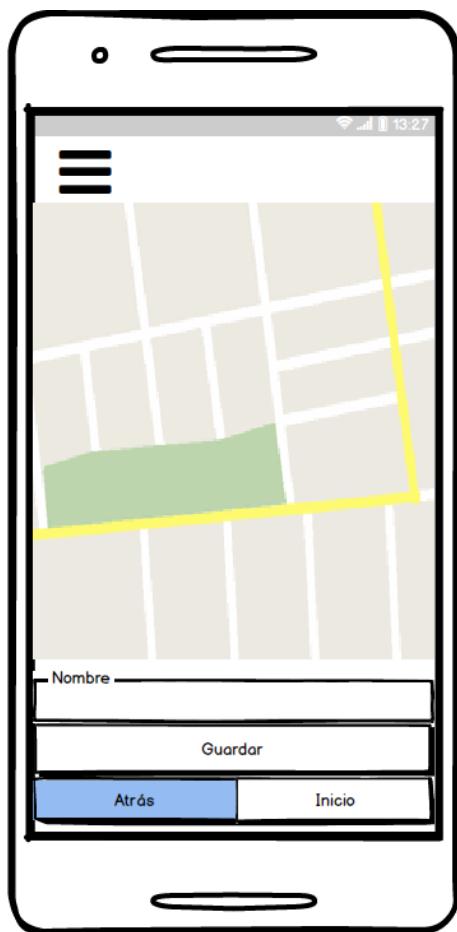


Figura 4.12: Crear PDI

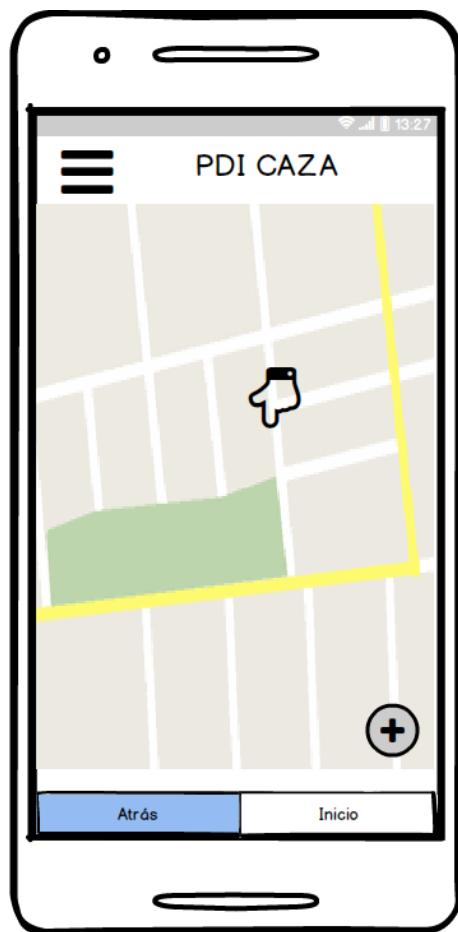


Figura 4.13: Visualizar PDIs

A continuación mostraremos como se crearía un PDI en la figura 4.12 y como lo veríamos una vez creado la figura 4.13.

CAPÍTULO 4.

Análisis

4.4 Interfaz y usabilidad

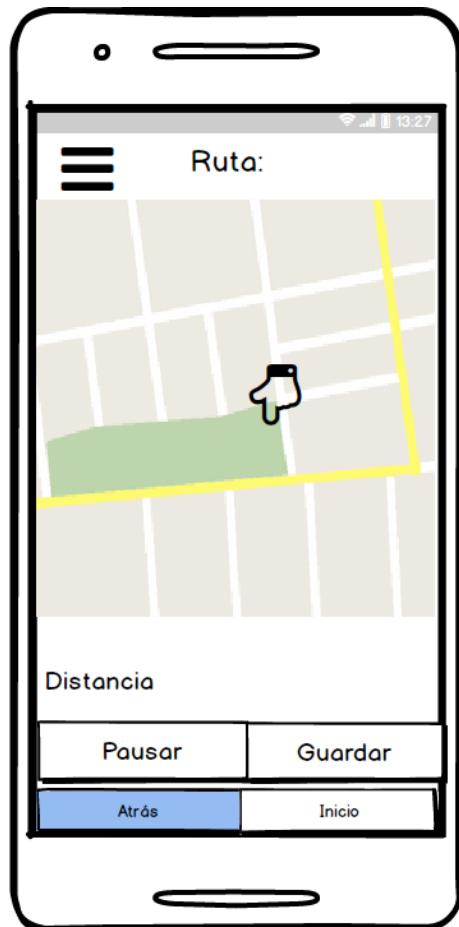


Figura 4.14: Ruta individual

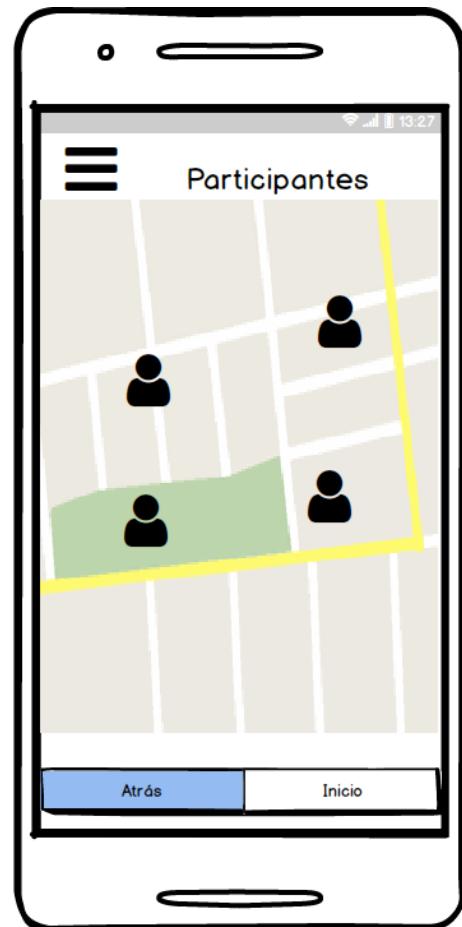


Figura 4.15: Ruta compartida

Por último vemos las maquetas de como seria una ruta individual, como podemos ver en la figura 4.14 y como seria la compartida en la figura 4.15.

Capítulo 5

Ejecución

En este capítulo se comentará el proceso seguido para la realización de este proyecto detallando la planificación, diseño y el desarrollo de interfaces.

5.1. Pasos de ejecución

Una vez hecho el análisis, conocidos los objetivos que se quieren alcanzar en este proyecto y las herramientas que se utilizarán empezaremos con la planificación del mismo.

Un punto muy importante a tener en cuenta a la hora de realizar una planificación sería conocer los recursos y el tiempo del que disponemos. Este punto tan crucial se acentúa ya que este proyecto será realizado por una única persona lo que producirá sobrecargas en la planificación. Esto normalmente no es así ya que los proyectos se realizan por grupos de trabajos y muchas tareas pueden realizarse en paralelo disminuyendo apreciablemente la duración del proyecto.

Para realizar la planificación del proyecto hemos dividido el proyecto en tareas con unos objetivos bien marcados(Product Backlog). A su vez al seguir la metodología Scrum dividimos el proyecto en etapas (Sprint) de tiempo donde incluimos las tareas a realizar. En todas estas etapas incluimos los siguientes pasos:

- Definición del Sprint Backlog, documento que recoge las historias de usuario y quién las desempeña. En este caso al solo trabajar el estudiante en el proyecto haré todas las tareas.
- División de las historias de del Sprint en tareas más sencillas y abordables.

- Diseño de cada una de las tareas del punto anterior.
- Implementacion de las tareas.
- Pruebas.

5.2. Duración y costes

A continuación en la tabla 5.1 siguiente se ha realizado una estimación de tiempos y de costes de este proyecto. Cuando se realiza una estimación de costes tenemos que tener en cuenta los gastos en material, licencias y los gastos propios. En mi caso los gastos de material y de licencias es nulo ya que el material es el del alumno y licencias y gastos propios son nulos. En la siguiente tabla se detalla el esfuerzo horas-hombre de cada uno de los roles dentro del proyecto.

La jornada laboral sería de 5 horas, 5 días a la semana y la duración de cada Sprint 4 semanas, lo que nos hace obtener el siguiente coste:

Rol	Coste	Tiempo	Total
Director del proyecto	40.00 (€/h)	25 h	1,000.00 €
Analista/Programador Junior	23.00 (€/h)	800 h	18,400.00 €
Servidor online	5.00 € al mes	8 meses	40.00 €
Amortización del móvil	400.00 € de compra	8 meses	66.00 €
Amortización del portatil	1,000.00 € de compra	8 meses	166.00 €
			19,673.00 €

Figura 5.1: Costes asociados a este proyecto

- En el rol de jefe de proyecto estaría a cargo de los que supervisan y evalúan el proyecto. En este caso los directores realizarán este rol.
- Analista/programador Junior, en este rol estaría la persona encargada de las tareas de análisis, diseño, desarrollo y de la creación de las pruebas para este proyecto. Al estar realizado por el estudiante este se encargaría de este rol.

5.3. Sprints

Como se comentó a lo largo de este capítulo la metodología usada será Scrum y el proyecto será dividido en Sprints de 4 semanas cada uno. El número de Sprint estimados sería de 8.

Antes de empezar con los Sprints comenzamos creando el Product Backlog. Esto es una lista con todas las funcionalidades de la aplicación priorizadas de más a menos importancia para nuestro proyecto.

5.3.1. Sprint 1

En este primer Sprint nos centramos en las tareas relacionadas con la capa intermedia. Para ello comenzamos diseñando el modelo de datos, algo fundamental en cualquier aplicación para no heredar carencias en capas superiores, y acabamos implementando el modulo de servicios.

5.3.2. Sprint 2

Una vez implementado el modulo servicio , el objetivo en este Sprint era hacer las pruebas para él. Una vez acabas las pruebas el siguiente paso dentro de este Sprint era la creación de las maquetas que sirvieron para la capa de presentación/cliente, es decir, el interfaz del usuario. Estas maquetas se fueron revisando y refinando a lo largo del proyecto.

5.3.3. Sprint 3

En este Sprint se comenzó con el desarrollo de la aplicación Android. Comenzamos permitiendo que el usuario fuera capaz que conocer su localización dentro del mapa y que al ir desplazándose cambie su ubicación como también obtener las coordenadas de su posición. Este punto será básico para el resto de la aplicación ya que nos proporciona los datos necesarios para guardar puntos y rutas. Las historias que se implementarán serán **creación de los puntos de interés (caza y pesca), búsqueda de PDIs por tipo y borrado de PDIs** los casos de uso realizados son los siguientes:

- **R-PDI-1 Guardar Punto De Interés caza** (vea Figura 5.3)
- **R-PDI-2 Guardar Punto De Interés pesca** (vea Figura 5.2)

CAPÍTULO 5.

5.3 Sprints

Ejecución

- **R-PDI-3 Eliminar PDI** (vea Figura 5.5)
- **R-PDI-4 Buscar los PDI** (vea Figura 5.4)



Figura 5.2: PDI pesca



Figura 5.3: PDI caza

5.3.4. Sprint 4

En este Sprint nos centraremos en las historias de **Iniciar sesión** **Registro** contiene los siguientes casos de uso:

- **R1 Registrarse en la aplicación.** (vea Figura 5.6)

CAPÍTULO 5.

Ejecución

5.3 Sprints



Figura 5.4: Guardar PDI

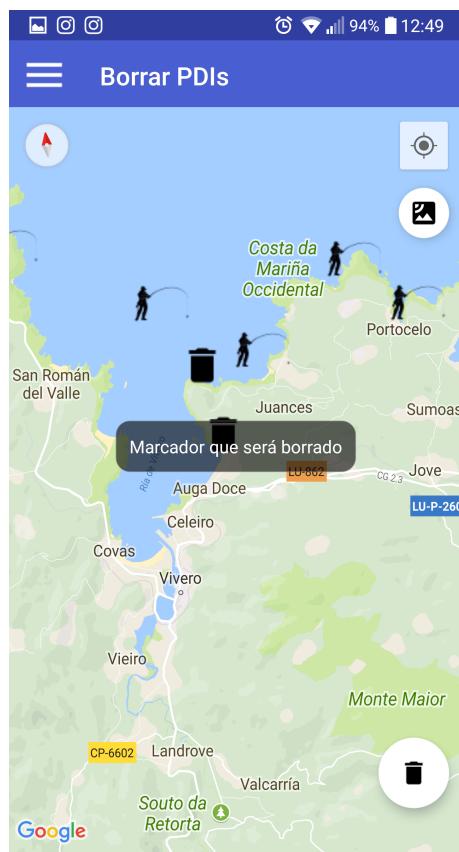


Figura 5.5: Borrar PDI

- **R2 Iniciar sesión en la aplicación.** (vea Figura 5.7)



Figura 5.6: Iniciar sesión

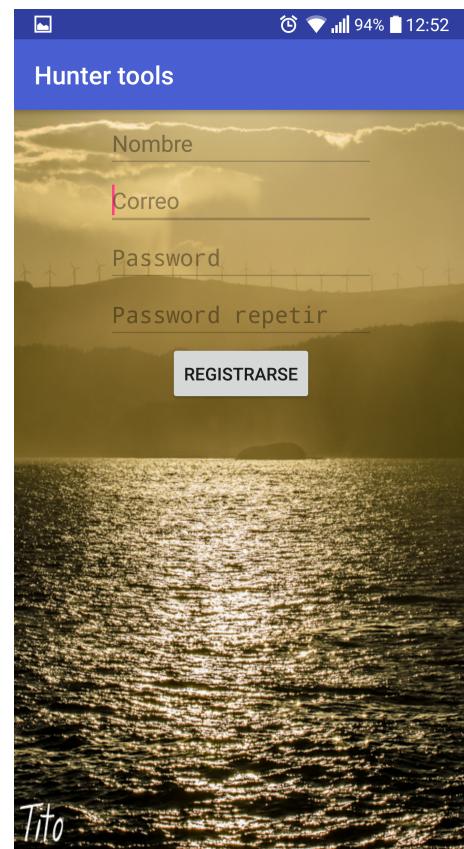


Figura 5.7: Registro del usuario

Además en este Sprint hemos desarrollado un toolbar para gestionar todas las opciones y mejorar la navegación del usuario en la aplicación.

CAPÍTULO 5.

Ejecución

5.3 Sprints

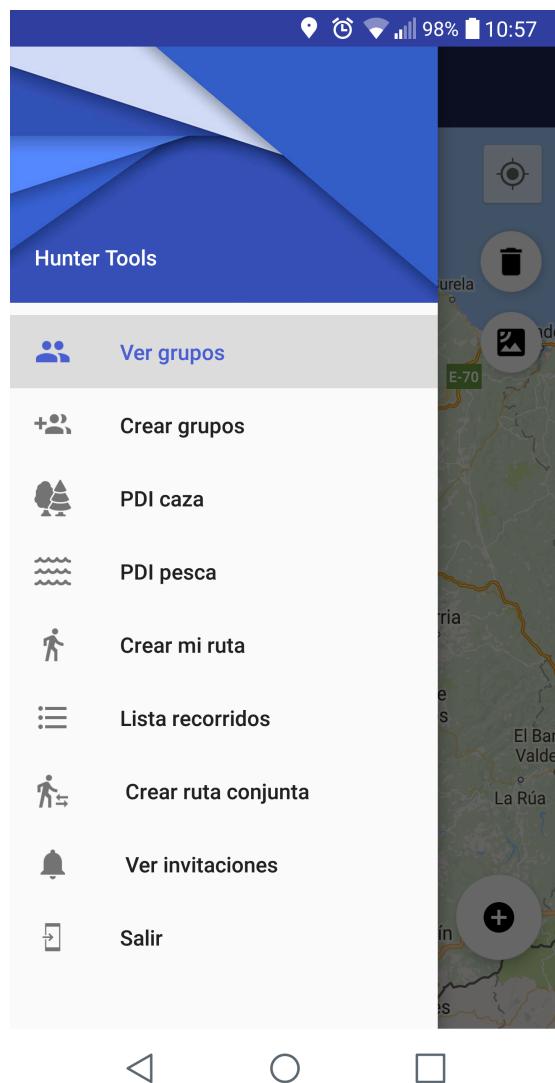


Figura 5.8: Captura del menú de navegación

5.3.5. Sprint 5

Este Sprint se centrará en las historias:

- *Creación de grupos*, contiene los siguientes casos de uso:

- *R-G-1 Crear grupo* (vea Figura 5.9)

- **R-G-2 Añadir integrantes**(vea Figura 5.10)
- **Eliminar grupos**, que contiene el siguiente caso de uso:
 - **R-G-3 Eliminar integrantes**
- **Visualización de grupos** , contiene los siguientes casos de uso:
 - **R-G-4 Ver grupos** (vea Figura 5.11)
 - **R-G-5 Ver integrantes grupo** (veaFigura 5.12)



Figura 5.9: Crear grupo

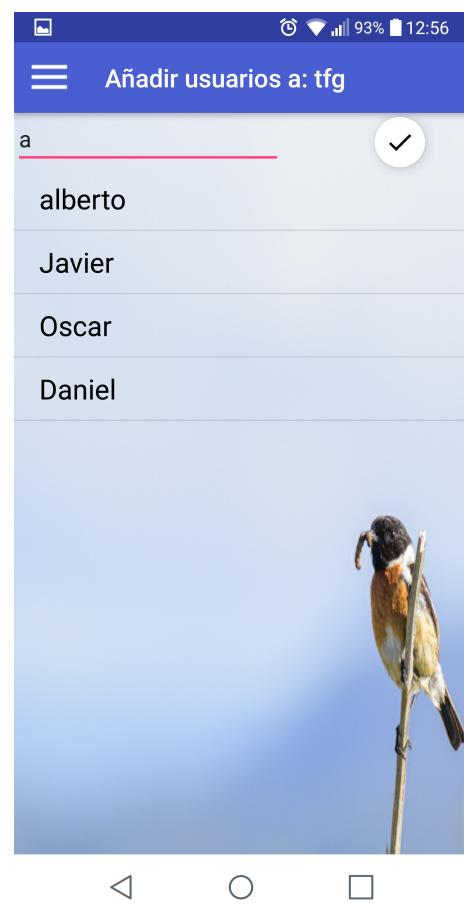


Figura 5.10: Añadir usuarios a grupo

CAPÍTULO 5.

Ejecución

5.3 Sprints

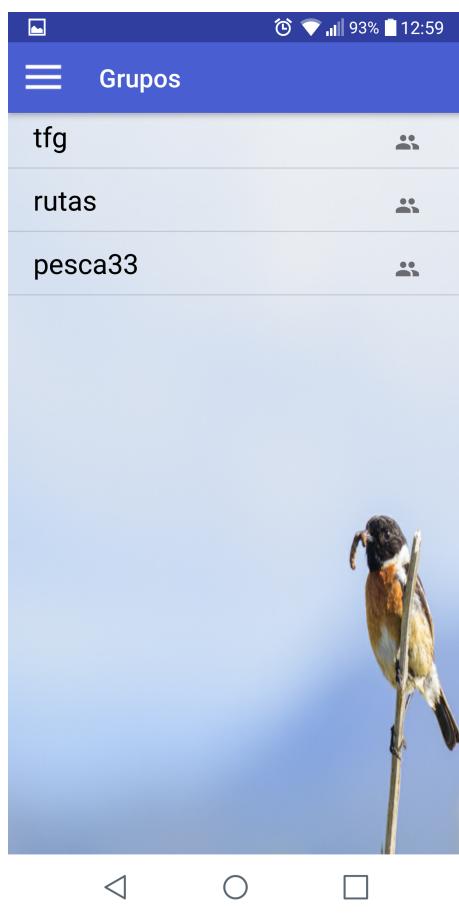


Figura 5.11: Ver grupos del usuario

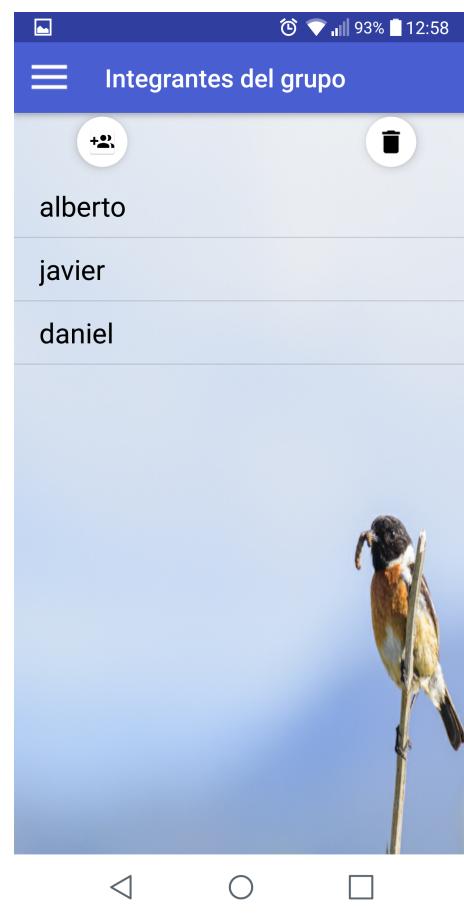


Figura 5.12: Ver integrantes del grupo

5.3.6. Sprint 6

Este Sprint se centrará en **Iniciar una ruta individual**.

- Creación de la ruta, contiene los siguientes casos de uso:

- **R-R-1 Crear ruta** (vea Figura 5.13)
- **R-R-1.1 Iniciar ruta**
- **R-R-1.2 Parar ruta**
- **R-R-1.3 Guardar ruta**

Figura 5.14 uniendo los catos de uso R-R-1.1, R-R-1.2 y R-R-1.3 .

- Ver las rutas seguidas por el usuario
 - **R-R-3 Listar rutas** (vea Figura 5.17)
- Ver una ruta en el mapa
 - **R-R-4 Ver ruta en mapa** (vea Figura 5.16)
- Eliminar ruta
 - **R-R-5 Eliminar ruta** (vea Figura 5.15)

CAPÍTULO 5.

Ejecución

5.3 Sprints

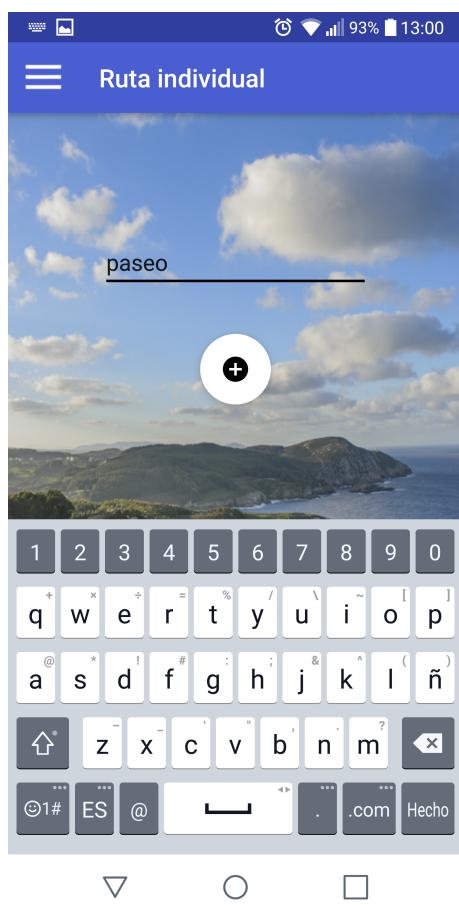


Figura 5.13: Crear ruta

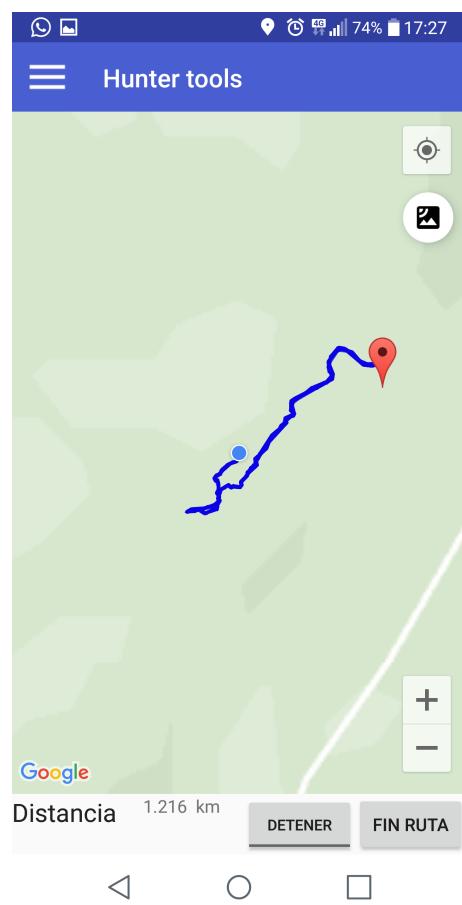


Figura 5.14: Navegación ruta individual

CAPÍTULO 5.

5.3 Sprints

Ejecución

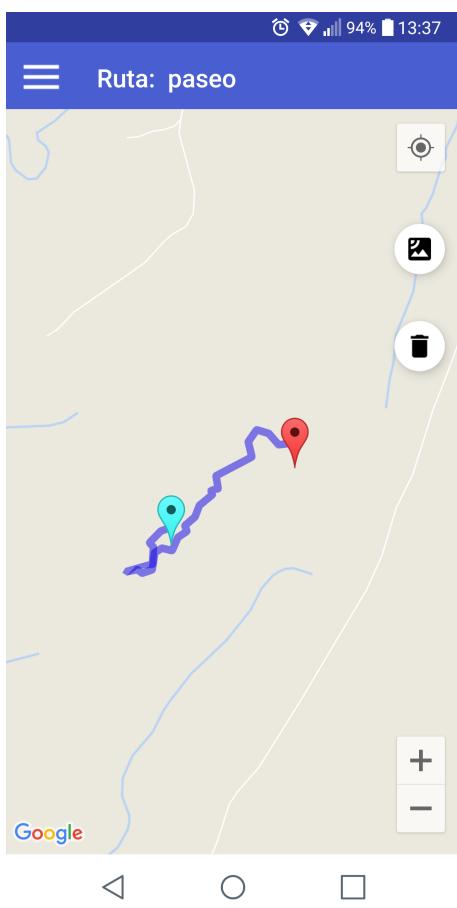


Figura 5.15: Eliminar ruta

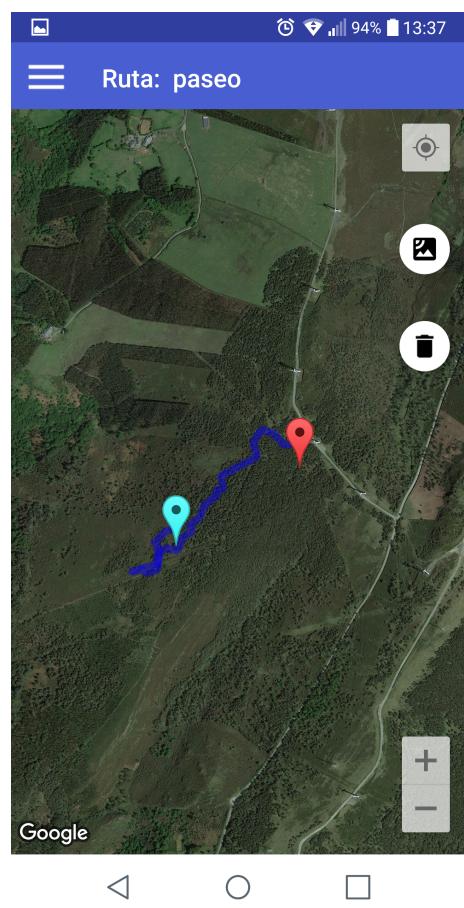


Figura 5.16: Ruta seguida vista satélite

CAPÍTULO 5.

Ejecución

5.3 Sprints

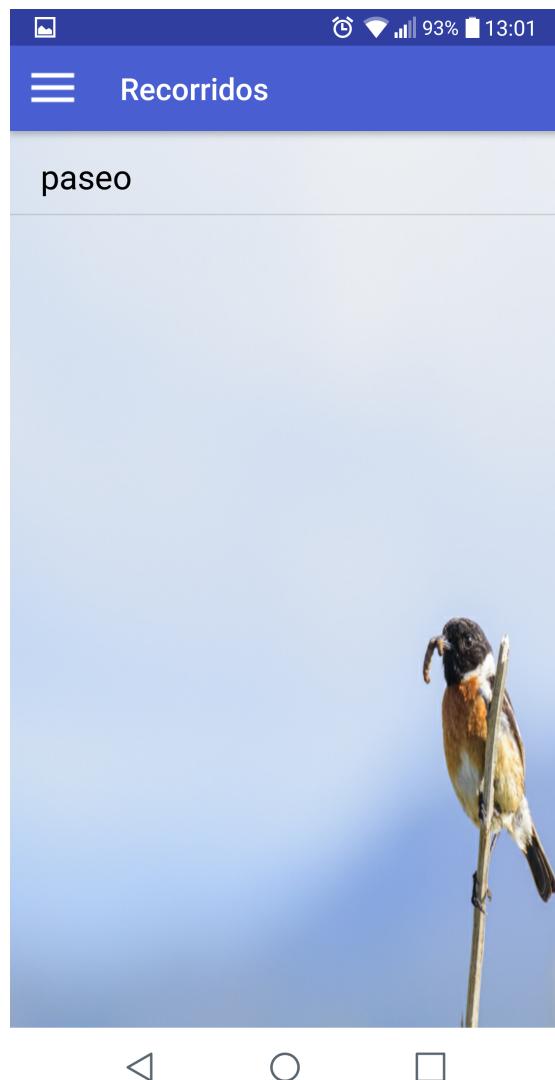


Figura 5.17: Lista de rutas creadas

5.3.7. Sprint 7

Este Sprint contiene la historia más compleja de todas, **gestionar una ruta compartida**, el interfaz es igual al de una ruta individual pero la lógica es diferente. Figura 5.14.

- *R-R-2 Crear ruta compartida*
- *R-R-2.1 Iniciar ruta*
- *R-R-2.2 Parar ruta*
- *R-R-2.3 Finalizar ruta*

5.3.8. Sprint 8

Por último en este Sprint se realizarán las pruebas finales, cierre del proyecto y la redacción de la memoria.

Capítulo 6

Diseño

En este capítulo se comentará el diseño de la arquitectura general, el modelo de datos y también aspectos más concretos del diseño del servidor y la aplicación móvil.

6.1. Arquitectura

La aplicación que estamos desarrollando seguirá la arquitectura en tres capas. En este patrón arquitectónico cliente/servidor se diferencian tres capas, una capa de interface de usuario , una capa de persistencia y una capa intermedia llamada de servicios que permite la llamada de forma remota a la capa modelo(capa que contiene la lógica) por parte del cliente. Este esquema se puede apreciar en la Figura 6.1. Este esquema esta compuesto por:

- **Capa de presentación/cliente:**

Capa que nos proporciona la información relacionada con los servicios que pue-de invocar el cliente. Es la encargada de comunicarse con las otras capas para guardar la información de cada usuario. En sí es la capa con la que va a interac-tuar el usuario cada vez trabaje con esta aplicación.

- **Capa intermedia:**

Esta capa es la encargada de enlazar la capa de persistencia con la capa de pre-sentación/cliente. Lo que hace es recoger los datos que provienen de la base de datos, necesarios para satisfacer el servicio invocado y enviárselos al cliente.

- **Capa de persistencia:**

Esta capa es la encargada de almacenar los datos del sistema en la base de datos,

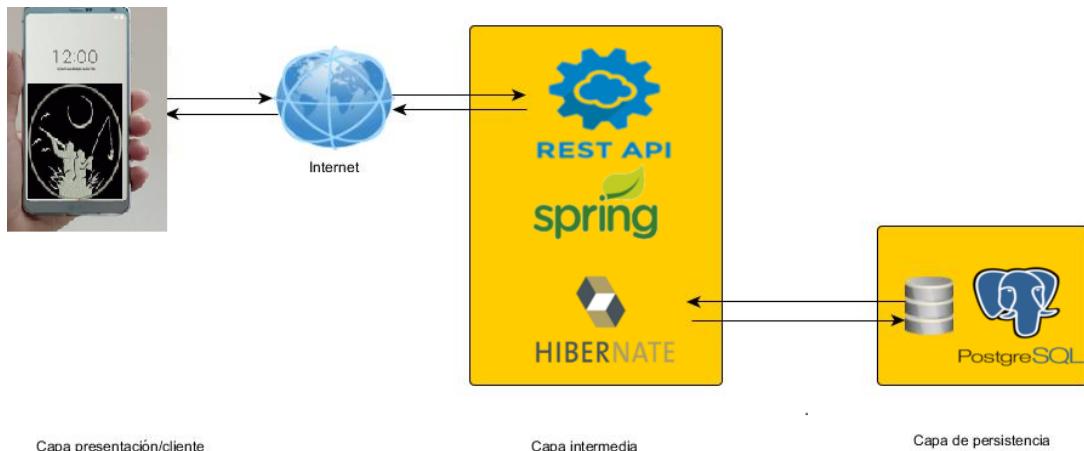


Figura 6.1: Esquema general de la arquitectura del sistema

además contiene todos los mecanismos de acceso a datos necesarios para poder hacer persistentes los datos. Esta capa debe ofrecer una interface que ayude a la comunicación con la capa intermedia, de manera que se abstraiga de la tecnología usada en el sistema de almacenamiento y no cree una dependencia con ella. Esta abstracción permitirá hacer cambios o actualizaciones en la tecnología sin afectar a otras capas con las que pudiera interaccionar en un futuro.

Una vez comentada la arquitectura general del sistema, pasaremos a comentar la capa intermedia un poco más a fondo ya que en ciertas ocasiones, la capa intermedia puede estar compuesta de N-capas(Arquitectura en N-capas). Esta es una de esas ocasiones.

Los servidores que siguen el patrón Modelo-Vista-Controlador consiguen separar los datos de la aplicación, la lógica de negocio y el envío de información por la red. Ésta separación ayuda al desarrollo de la aplicación tanto a la hora de crear la como a la hora de hacer su mantenimiento ya que marca al desarrollador a distribuir el código en una capa concreta.

Los componentes capas que componen al patrón MVC son:

- **Modelo:** Está compuesta por clases que tienen acceso a los datos ofreciendo unos métodos para ser usados de manera sencilla por las capas superiores. Éstos métodos son los encargados de acceder a la base de datos y proporcionar los datos

CAPÍTULO 6.

Diseño

6.2 Capa de datos

persistentes.

- **Vista:** Está formada por el interfaz donde se realizan las llamadas entre el servicio web, peticiones HTTP en las que la información va en formato JSON, y la aplicación cliente.
- **Controlador:** Esta capa será la encargada de implementar la lógica del interface llamando a las operaciones que ofrece el modelo y seleccionando la vista asociada a cada petición.

Esta descripción anteriormente reflejada representa a la siguiente Figura 6.2

6.2. Capa de datos

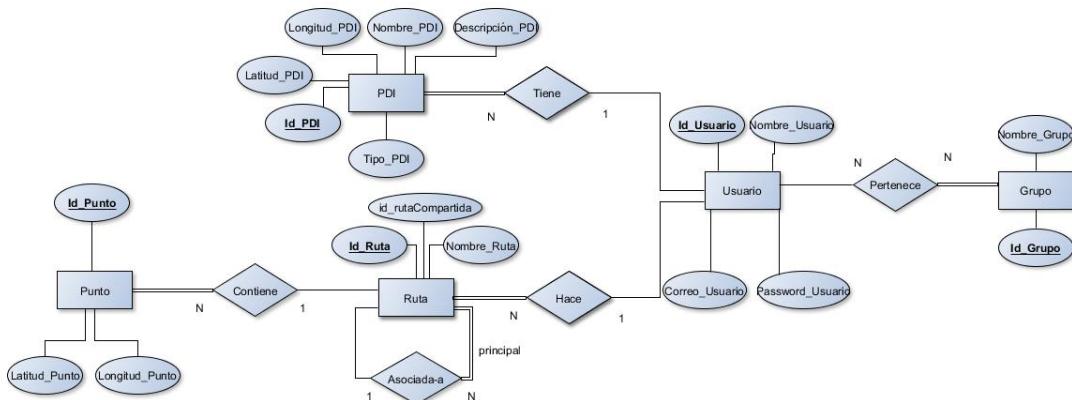


Figura 6.3: Entidad relación

En la Figura 6.3 queda reflejado el modelo entidad-relación de nuestro sistema. A continuación comentaremos el dominio.

La entidad principal sobre la que gira la aplicación es **Usuario**. Este puede crear **Grupos** a los que pasa a pertenecer al ser creados y los cuales desaparecen del sistema una vez que queden sin integrantes automáticamente. El usuario puede crear **PDI(Puntos De Interés)** los cuales son propios y privados de un solo usuario con los atributos que aparecen reflejados en la figura. El usuario también puede realizar **Rutas** que a su vez pueden tener **Puntos** asociados a ellas, es decir, el usuario crea una ruta asociándola a el y la ruta tiene asociada a ella puntos. Un caso especial serían las rutas compartidas

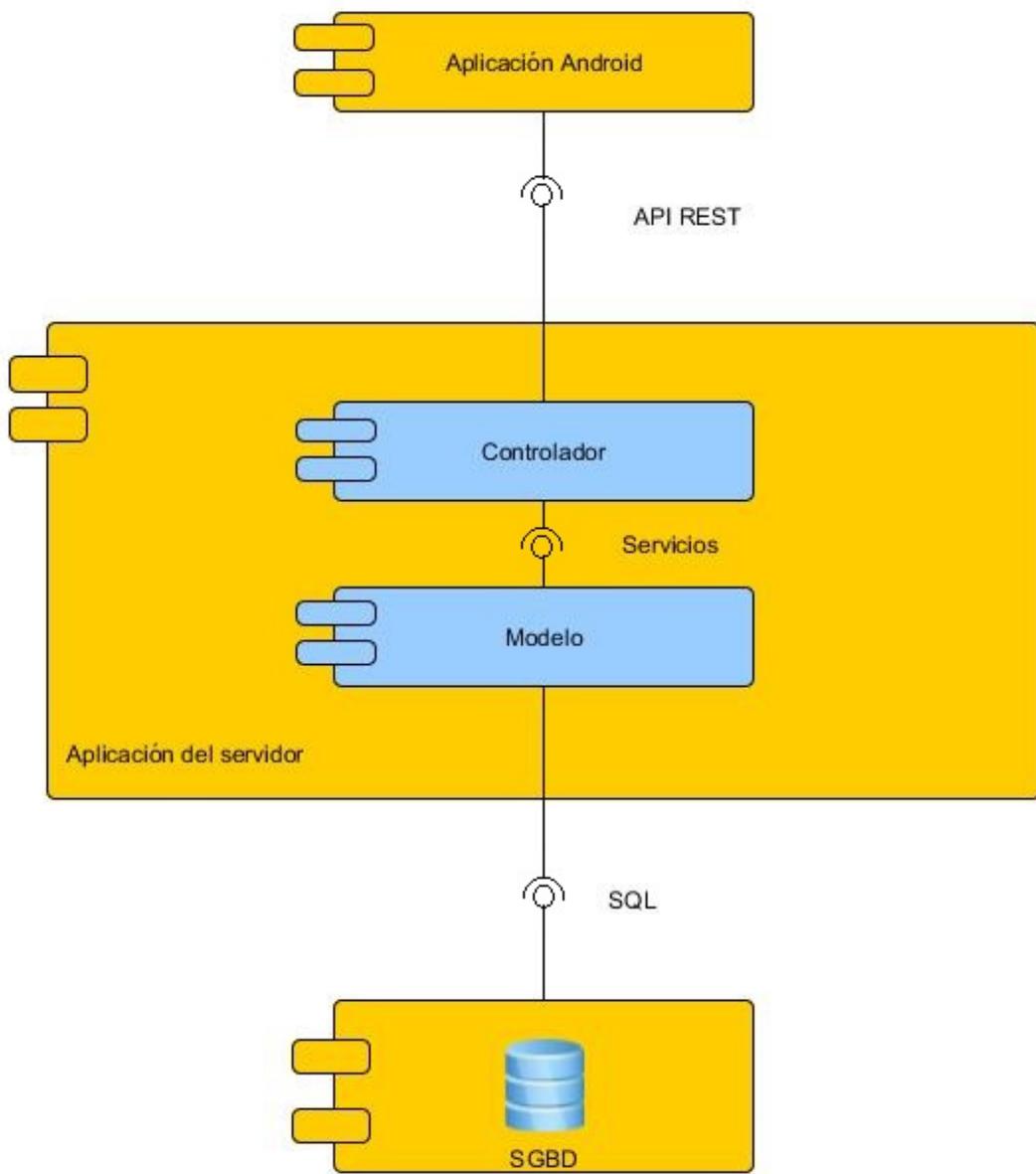


Figura 6.2: Diagrama de componentes del sistema

asociadas a una principal, en las cuales el usuario crea una ruta y después la comparte con otros usuarios. Para quedar esto reflejado en el entidad-relación tenemos la relación *Asociada-a principal*.

6.3. Aplicación servidor

Para permitir una comunicación entre la aplicación del cliente y la capa de persistencia hemos desarrollado un servidor desplegado en un proveedor de internet , al que se podrá acceder de manera remota y que permitirá tener acceso a las funcionalidades de la capa persistente. La arquitectura del servidor la podremos ver en la Figura 6.4

La solución mencionada anteriormente será un servidor, en Java, que usará Spring [10] ya que facilita la creación de aplicaciones de forma cómoda y rápida.

Durante el desarrollo de este proyecto el servidor ha sido desplegado en un proveedor de servidores virtuales llamado DigitalOcean, que ofrece distintas localizaciones donde poder ubicarlo. Hemos decidido hacerlo en uno concreto de Alemania para que hubiera menos latencia y mayor rapidez.

6.3.1. Servicio web

La comunicación en esta capa se hará mediante una API. Una API describe la forma en que los programas o los sitios webs intercambian datos, cuyo formato de intercambio de datos normalmente es JSON. En este servicio hemos decidido usar una API REST, un tipo de arquitectura de desarrollo web que se apoya totalmente en el estándar HTTP para la obtención de datos. A través de los siguientes peticiones podremos acceder a los distintos recursos.

- Los métodos de acceso indican la acción se queremos realizar sobre los recursos de nuestro sistema. Siguiendo esta línea tenemos el GET para solicitar un recurso pedido. POST para crear el recurso en el sistema y para eliminar recursos usaremos DELETE.
- Dependiendo del método usado se irán cambiando la ruta en la petición HTTP y el contenido de ellas(cuerpo). Un POST tendrá una URL sencilla ya que los datos del objetos para ser creado irán en el cuerpo mientras que el GET y DELETE

CAPÍTULO 6.

6.3 Aplicación servidor

Diseño

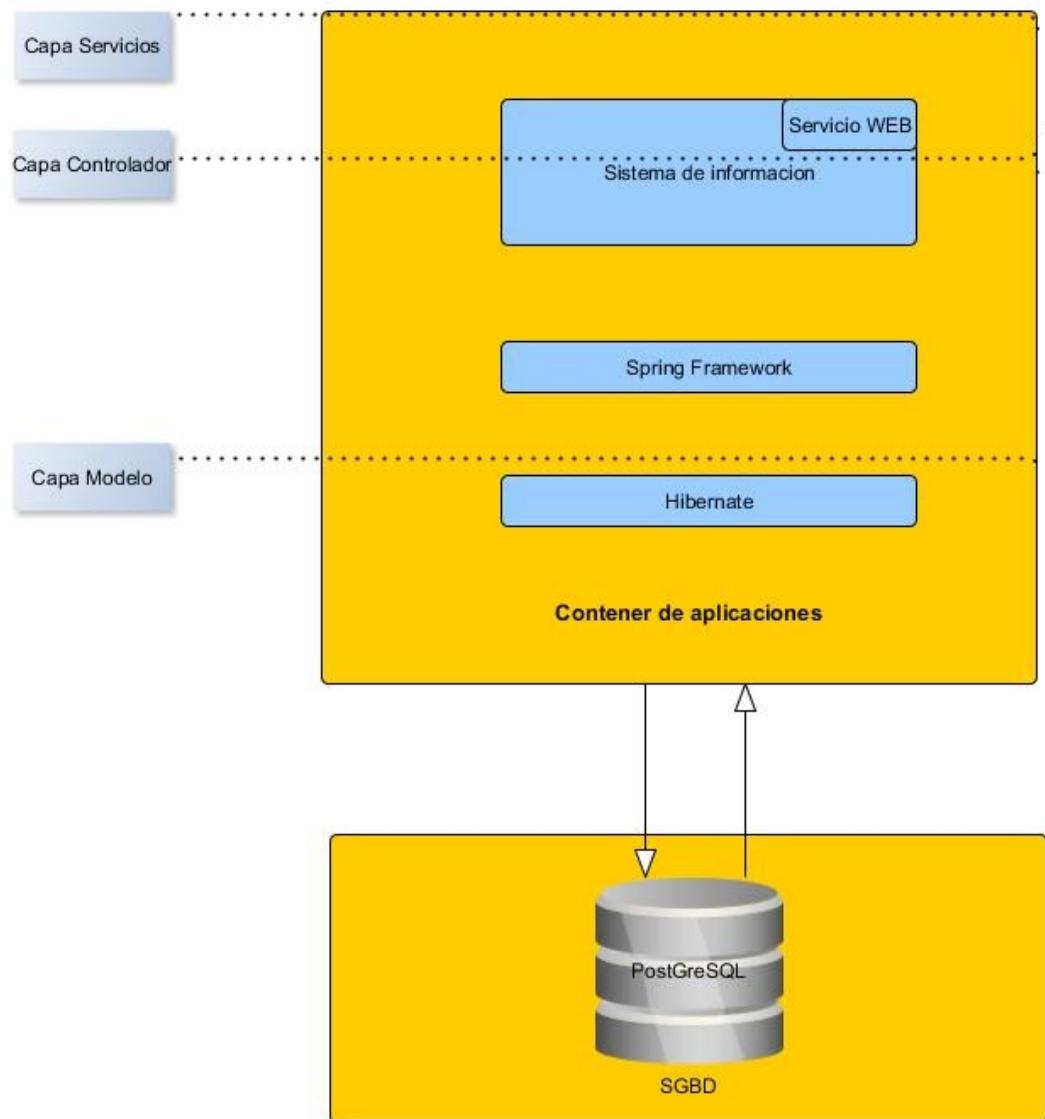


Figura 6.4: Arquitectura del servidor

CAPÍTULO 6.

Diseño

6.3 Aplicación servidor

TE la tendrán de la forma `/usuario/idusuario` para obtener un objeto concreto o eliminarlo.

6.3.2. Organizacion dos paquetes

La aplicación del servidor sigue un arquetipo en 5 paquetes distintos de clases Java y uno para los test:

java

- **es.udc.fi.dc.config** Contiene las clases Java de configuración de Spring.
- **es.udc.fi.dc.model** Contiene todos los modelos de datos que estarán almacenados en la base de datos.
- **es.udc.fi.dc.daos** Contiene los interfaces (JPARespository) que acceden a los datos del sistema.
- **es.udc.fi.dc.controller** Contiene las clases relacionadas con las peticiones remotas.
- **es.udc.fi.dc.services** Contiene tanto los interfaces de los servicios como la implementación de los mismos.

test

es.udc.fi.dc.test Contiene las clases que realizan los test.

6.3.3. Transmisión de la información

Como se mencionó anteriormente para el servicio web usaremos la API REST, esta necesita un lenguaje de intercambio para la transmisión de información. El formato que seguiremos para la transmisión será el JSON y como parseador Jackson. JSON, siglas JavaScript Object Notation, es un formato estándar de transmisión de la información muy cómodo de usar que emplea pares de clave-valor. Ejemplo de Json en la Figura 6.5

```

1  {
2      "idusuario": 2,
3      "nombre": "alberto",
4      "correo": "alberto@udc.es",
5      "token": "dQUoL9K4zUY:APA91bER1EbPGpnufqQyDtSESPgHeDEn0-crXnGfGD6evwI
6          YP3kPFg662SX-PpKRr-1THUwMKy3DSIW-oRjuSzgOKS0XgRDTQNkbwZSm1qkQcAPmzb4C0t10jXupxHR7yAHWSKZsFIws"
7 }

```

Figura 6.5: JSON

6.3.4. Gestión de las clases persistentes

Para gestionar las clases persistentes de nuestra aplicación usaremos el patrón de diseño basado en DAO(Data Access Object).

Un DAO define un interfaz que contiene conjunto de operaciones de persistencia y a su vez una implementación permitiendo la gestión de las entidades en la base de datos. Es decir oculta la gestión de la base de datos y a su vez la tecnología que usamos en ella, dotando a cada clase persistente de un DAO asociado a ella. En éste proyecto hemos usado Spring Data JPA que nos proporciona una interfaz para la gestión de los objetos del dominio sin tener que escribir nosotros la implementación de los métodos. Esto nos permite ganar agilidad a la hora de crear los DAOs ya que permite extender esta interfaz a nuestra clase DAO proporcionando todos los métodos CRUD, métodos para modificar o eliminar objetos de ese tipo. Éste interfaz se llama JpaRepository.

Otro motivo por el cual usamos JpaRepository es que permite extender la creación de una buena serie de métodos de búsqueda por el nombre de los atributos de nuestras clases persistentes simplemente definiendo los como un método de nuestro interfaz del DAO.

```

1
2 public interface UsuarioRepository
3     extends JpaRepository<Usuario, Long> {
4
5     public Usuario findByCorreo(String correo);
6
7     public Usuario findByNombre(String nombre);
8
9     public List<Usuario> findByNombreContaining
10        (String nombre);
11
12    @Override
13    public <S extends Usuario> S save(S usuario);

```

CAPÍTULO 6.

Diseño

6.4 Aplicación Móvil

```
14  
15 @Override  
16 public void delete(Long idUsuario);  
17  
18 @Override  
19 public boolean exists(Long idUsuario);  
20  
21 @Override  
22 public <S extends Usuario> S saveAndFlush(S usuario);  
23  
24 @Override  
25 public Usuario findOne(Long idUsuario);  
26 }
```

Código 6.1: JpaRepository

6.4. Aplicación Móvil

La capa de presentación será accesible por el usuario a través de una aplicación móvil desarrollada en Android. Los aspectos más relevantes serán expuestos a continuación.

6.4.1. Servicios usados

- **Google Maps API.** Es un servicio web proporcionado por Google que nos permite usar los mapas en nuestras API y también conocer la ubicación del usuario en todo momento. Esta ubicación con un punto azul en el mapa pero no suministra coordenadas(latitud y longitud).
- **Google Location.** Es un servicio de Google que nos suministra las coordinadas del usuario, según varía la posición ciertos metros o cada cierto tiempo.
- **RestSystemService.** Este servicio es el encargado de ejecutar nuestro propio servidor para guardar o devolver los datos de los usuarios, puntos de interés, rutas o grupos del usuario.
- **Firebase Cloud Messaging(FCM)** es una solución de mensajería multiplataforma que te permite enviar mensajes de forma segura y gratuita. En este caso lo hemos utilizado para enviar notificaciones push cuando se inicia una ruta compartida.

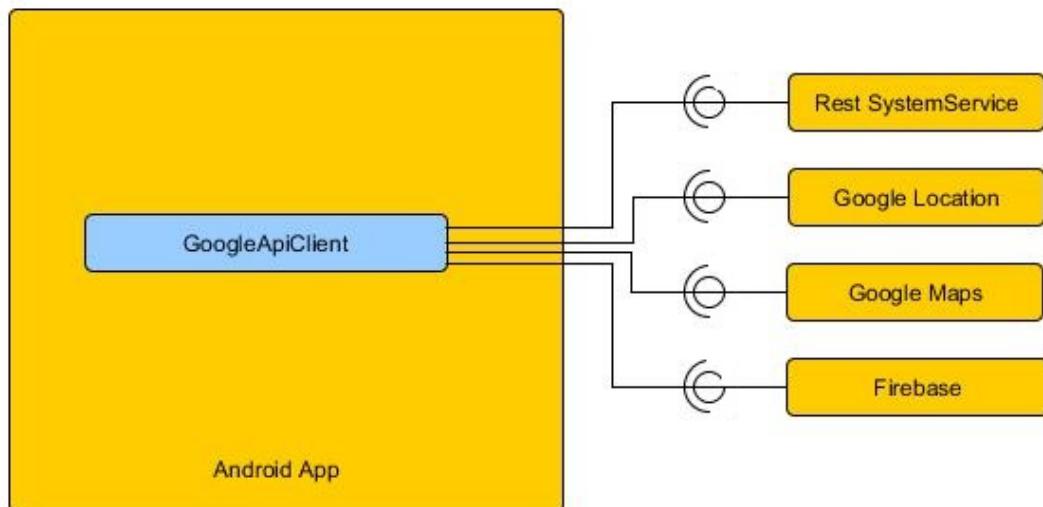


Figura 6.6: Servicios de la aplicación móvil

6.4.2. Organización de los paquetes

La organización seguida en la aplicación móvil es la de agrupar las clases por funcionalidades comunes.

- **alberto.hunter-android.activities**, contiene todas las actividades. Las actividades contienen la lógica que hay detrás de cada interfaz con la que interactua el usuario.
- **alberto.hunter-android.adapter**, contiene los adaptadores. Son clases que reciben datos y los manipula para poder ser usados en otra actividad.
- **alberto.hunter-android.model**, contiene todos los modelos de datos.
- **alberto.hunter-android.utils**, contiene las clases auxiliares.

Con esta organización por funcionalidades comunes creamos un proyecto claro y organizado lógicamente.

Capítulo 7

Implementación

7.1. Implementación

En este capítulo expondremos la implementación tanto de la aplicación servidor como de la aplicación móvil.

7.1.1. Aplicación Servidor

En capítulos anteriores ya comentamos la estructura de paquetes seguida para desarrollar el servidor, en este hablaremos de JPARepository.

JPARepository es un repositorio que nos ofrece métodos genéricos de gestión de clases persistentes como también métodos más concreto que nos permiten realizar operaciones complejas abstrayendones de su implementacion.

Además este repositorio se adapta a la clase con la que va a trabajar.

```
1 public interface UsuarioRepository  
2 extends JpaRepository<Usuario, Long> {  
3 }  
4 }
```

Código 7.1: Adaptación a la clase Usuario

Como se puede ver en la figura 7.2, este interfaz importa la clase Usuario accediendo a todos sus atributos y generando una lista de métodos para estos atributos concretos como podemos ver en la siguiente.

```

1  public interface UsuarioRepository
2   extends JpaRepository<Usuario, Long> {
3
4   public Usuario findByCorreo(String correo);
5
6   public Usuario findByNombre(String nombre);
7
8   public List<Usuario> findByNombreContaining
9     (String nombre);
10
11  @Override
12  public <S extends Usuario> S save(S usuario);
13
14  @Override
15  public void delete(Long idUsuario);
16
17  @Override
18  public boolean exists(Long idUsuario);
19
20  @Override
21  public <S extends Usuario> S saveAndFlush(S usuario);
22
23  @Override
24  public Usuario findOne(Long idUsuario);
25
26 }
```

Código 7.2: Interfaz de UsuarioRepository

API REST

■ UsuarioController

- **/usuario/correo/{correo} [GET]**

Busca al usuario cuyo correo es el parámetro correo.

- Param
 - ◊ String correo
- Reponse
 - ◊ Usuario

- **/usuario/nombre/{nombre} [GET]**

Busca al usuario cuyo nombre es el parámetro nombre.

CAPÍTULO 7.

Implementación

7.1 Implementación

- Param
 - ◊ String nombre
 - Reponse
 - ◊ Usuario
- **/usuarios/lista/{nombre} [GET]**
Busca los usuarios que tienen el parámetro nombre como parte de su nombre.
- Param
 - ◊ String nombre
 - Reponse
 - ◊ List<Usuario>
- **/usuario/ruta/{idUsuario} [GET]**
Busca las rutas cuyo propietario es el que tiene como identificador el idUsuario.
- Param
 - ◊ Long idUsuario
 - Reponse
 - ◊ List<Ruta>
- **/usuario/pois/{idUsuario} [GET]**
Busca los pois cuyo propietario es el que tiene como identificador el idUsuario.
- Param
 - ◊ Long idUsuario
 - Reponse
 - ◊ List<Poi>
- **/usuario/pois/usuario/{idUsuario}/tipo [GET]**
Busca los pois cuyo propietario es el que tiene como identificador el idUsuario y por tipo de poi el tipo que le pasamos.
- Param
 - ◊ Long idUsuario
 - ◊ String tipo

- Reponse
 - ◊ List(Poi)
- **/usuario/ [POST]**
Crea un usuario.
 - Body
 - ◊ Usuario
 - Reponse
 - ◊ Usuario
- **/usuario/grupos/{idUsuario} [GET]**
Busca los grupos a los que pertenece el usuario con identificador idUsuario.
 - Param
 - ◊ Long idUsuario
 - Reponse
 - ◊ List<Grupo>
- **/usuario/{idUsuario} [DELETE]**
Elimina un usuario
 - Param
 - ◊ Long idUsuario
- **/usuario/grupo/{idGrupo} [POST]**
Añade un usuario a un grupo.
 - Param
 - ◊ Long idGrupo
 - Body
 - ◊ Usuario usuario
- **/usuario/grupo/{idGrupo}/usuario/{nombre} [DELETE]**
Elimina a un usuario de un grupo.
 - Param
 - ◊ Long idGrupo
 - ◊ String nombre

- **PoiController**

CAPÍTULO 7.

Implementación

7.1 Implementación

- **/poi/ [POST]**

Crea un Poi.

- Body
 - Poi
- Reponse
 - Poi

- **/poi/{idPoi} [DELETE]**

Elimina un poi

- Param
 - Long idPoi

■ GrupoController

- **/grupo/ [POST]**

Crea un grupo.

- Body
 - Grupo

- **/grupo/{nombreGrupo} [GET]**

Busca el grupo cuyo nombre es nombreGrupo.

- Param
 - String nombreGrupo
- Reponse
 - Grupo

- **/grupos/lista/{nombreGrupo} [GET]**

Devuelve una lista con los integrantes de un grupo cuyo nombre es nombreGrupo-

- Param
 - String nombreGrupo
- Reponse
 - List<Usuario>

■ RutaController

- **/ruta/ [POST]**

Crea una ruta.

- Body
 - Ruta
- Reponse
 - Ruta

- **/rutacompartida/{nombreGrupo} [POST]**

Crea una ruta compartida para cada uno de los integrantes del grupo cuyo nombre es nombreGrupo. Además envía una notificación push a cada integrante para indicar que la ruta ya está disponible para comenzar.

- Param
 - String nombreGrupo
- Body
 - Ruta
- Reponse
 - Ruta

- **/rutacompartida/{idRuta} [GET]**

Busca todas las rutas cuyo parámetro idrutacompartida es idRuta.

- Param
 - Long idRuta
- Reponse
 - List<Ruta>

- **/rutaCompartida/{idRuta}/estado/{estado} [POST]**

Cambia el estado de la ruta cuyo identificador es idRuta.

- Param
 - Long idRuta
 - String estado
- Reponse
 - Ruta

- **/rutacompartida/cerrar/{idRuta} [POST]**

Cierra la ruta cuyo identificador es idRuta. Y en el caso de que el resto de

CAPÍTULO 7.

Implementación

7.1 Implementación

usuario ya hubieran cerrado sus rutas, elimina la invitación restante del usuario que queda.

- Param
 - ◊ Long idRuta
- Reponse
 - ◊ Ruta

- **/rutacompartida/seguimiento/ruta/{idRuta}/grupo/{nombreGrupo} [POST]**

Este servicio es el encargado de realizar el seguimiento de los usuarios. Envía una lista con los puntos seguidos durante la ruta cuyo identificador es idRuta del usuario y además recibe una lista con los últimos puntos de cada uno de los integrantes del grupo cuyo nombre es nombreGrupo.

- Param
 - ◊ String nombreGrupo
 - ◊ Long idRuta
- Body
 - ◊ List<PuntoRuta> puntosRutaEntrada
- Reponse
 - ◊ List<PuntoRuta>

- **/ruta/{idRuta} [DELETE]**

Elimina la ruta con identificador idRuta.

- Param
 - ◊ Long idRuta

- **/ruta/puntos/{idRuta} [GET]**

Obtiene todos los puntos de la ruta que tiene como identificador el idRuta.

- Param
 - ◊ Long idRuta
- Reponse
 - ◊ List<PuntoRuta>

- **/ruta/lista/{idRuta} [GET]**

Obtiene una lista con rutas que fueron compartidas a la vez que la ruta cuya idRutaCompartida es idRuta.

- Param
 - ◊ Long idRuta
- Reponse
 - ◊ List(Ruta)

■ **PuntoRutaController**

• **/puntoruta/{idRuta} [POST]**

Guarda una lista de puntos asociados a una ruta que tiene como identificador idRuta.

- Param
 - ◊ Long idRuta
- Body
 - ◊ List<PuntoRuta>puntosRutaEntrada
- Reponse
 - ◊ List<PuntoRuta>

En la anterior API se usan una serie de objetos que hacen referencia a las siguientes interfaces:

- Usuario ver figura 7.1
- Grupo ver figura 7.2
- Ruta ver figura 7.3
- PuntoRuta ver figura 7.4
- Poi ver figura 7.5

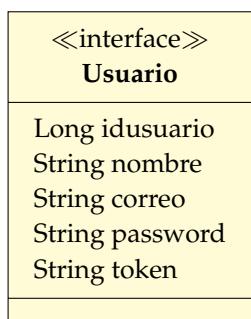


Figura 7.1: Interfaz Usuario

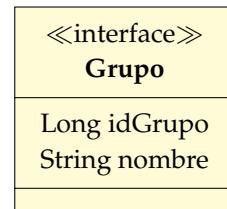


Figura 7.2: Interfaz Grupo

CAPÍTULO 7.

Implementación

7.1 Implementación

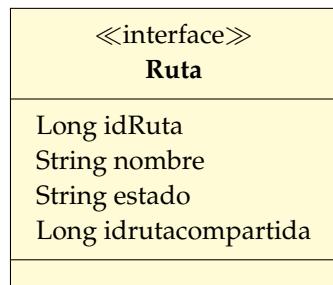


Figura 7.3: Interfaz Ruta

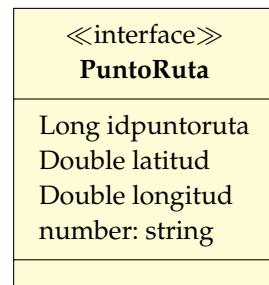


Figura 7.4: Interfaz PuntoRuta

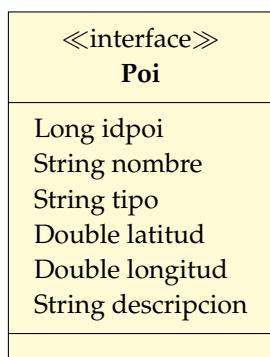


Figura 7.5: Interfaz Poi

7.1.2. Aplicación móvil Android

En este capítulo comentaremos aspectos concretos de la implementación de la aplicación móvil [8].

Mapas

Para la creación de rutas tanto individuales como compartidas como para crear PDI el usuario necesita conocer las coordenadas de los puntos por lo que transcurre su ruta. Para ello necesitamos los mapas de Google Maps y métodos de sus APIs.

```
1 compile 'com.google.android.gms:play-services-maps:10.2.0'
```

Código 7.3: Dependencia de Google Maps en gradle

Con la dependencia de la figura 7.3 permitimos a nuestra aplicación que use los servicios de Google Maps.

Para marcar un punto en el mapa y que este quede visible en este necesitamos implementar un método que capture los clic en el mapa y que nos devuelva las coordenadas del punto marcado.

```
1
2 mMap.setMyLocationEnabled(true);
3 mMap.getUiSettings().setCompassEnabled(true);
4 mMap.setOnMapClickListener(new GoogleMap
5     .OnMapClickListener() {
6         public void onMapClick(LatLng point) {
7             mMap.clear();
8             poilongitud = point.longitude;
9             poilatitud = point.latitude;
10            LatLng Yo = new LatLng(poilatitud, poilongitud);
11            Marker mensaje =
12                mMap.addMarker(new MarkerOptions()
13                    .position(Yo)
14                    .title("Guardar este punto?"));
15            mensaje.showInfoWindow();
16            VerTodosPois();
```

Código 7.4: Captura de clic en la pantalla

Con el fragmento de código de la figura 7.4 se capturaría ese clic y aparecería el Marker común de todos los mapas de Google Maps acompañado del mensaje "*Guardar este punto?*". En nuestro proyecto personalizamos los Marker de modo que cuando

CAPÍTULO 7.

Implementación

7.1 Implementación

guardamos ese punto pase a a representarse con icono de un pescador o de un cazador dependiendo del PDI que estuvieramos guardando. Para ello usamos el siguiente fragmento de código.

```
1 protected Marker createMarkerPesca(double
2     latitude ,Double longitude , String nombre ,
3     String descripcion ) {
4         return mMap.addMarker(new MarkerOptions()
5             .position(new LatLng(latitude ,
6                 longitude )) .anchor(0.5 f , 0.5 f)
7             .title(nombre) .snippet(descripcion )
8             .icon(BitmapDescriptorFactory
9                 .fromResource(R.drawable.pescador)));
10    }
11 }
```

Código 7.5: Creación de Marker personalizados

Y así es como quedaría en la figura ?? y figura ??

Localización

Para la creación de ruta necesitamos un método que nos proporcione las coordenadas(latitud y longitud) de los puntos por los que transcurre el usuario en la ruta. Para ellos necesitamos la siguiente dependencia.

```
1 compile 'com.google.android.gms:play-services-location:10.2.0'
```

Código 7.6: Dependencia de Location en gradle

Una vez añadida esta dependencia necesitamos un método que nos proporciones las coordenadas concretas, para ellos usaremos el siguiente fragmento de código.

```
1 mlocManager.requestLocationUpdates(LocationManager
2 .GPS_PROVIDER 15 , 0 ,(LocationListener) Local);
```

Código 7.7: Obtencion de coordenadas por intervalo de tiempo

El método ofrece el par de coordenadas cuando el usuario se mueve 15 metros o bien por un intervalo de tiempo en segundos.

Como se puede observar en la figura 7.7 en nuestro proyecto indicamos que los segundos que deberían transcurrir para devolver unas coordenadas sería de 15.

Cuando nosotros estamos realizando la ruta en ocasiones el GPS pierde precisión y sitúa al usuario en punto un tanto lejano, lo cual es imposible ya que no se puede mover

CAPÍTULO 7.

7.1 Implementación

Implementación



Figura 7.6: Marker antes de guardar el PDI



Figura 7.7: Marker después

CAPÍTULO 7.

Implementación

7.1 Implementación

tan rápido. Este error del GPS lo hemos resuelto de la siguiente manera, el código 7.8 .

```
1     iniTramo = finTramo;
2     finTramo = new LatLng(latitud, longitud);
3     Location location1 = new Location("localizacion 1");
4     location1.setLatitude(iniTramo.latitude); //latitud
5     location1.setLongitude(iniTramo.longitude); //longitude
6     Location location2 = new Location("localizacion 2");
7     location2.setLatitude(finTramo.latitude); //latitud
8     location2.setLongitude(finTramo.longitude); //longitude
9     double distance = location1.distanceTo(location2);
```

Código 7.8: Fragmento para el calculo de distancias entre puntos

En este código lo que hacemos es calcular la distancia entre dos puntos consecutivos. Usamos un método que pertenece a la API, el cual necesita un par de coordenadas. El dato que devuelve viene dado en metros y dado tiene en cuenta la curvatura de la tierra para calcularlo. Si esta distancia es superior a 15 metros desechamos ese punto. El resultado de los puntos válidos lo podemos ver en Figura 7.8

Firebase

Cuando el usuario inicia una ruta compartida el sistema avisa al resto de integrantes de grupo de que acaban de ser invitados a ella, esta acción la realizamos mediante las Notificaciones Push con Firebase [7]. Para ello tenemos que empezar añadiendo las dependencias siguientes.

```
1 compile 'com.google.firebaseio:firebase-core:10.2.0'
2 compile 'com.google.firebaseio:firebase-messaging:10.2.0'
```

Código 7.9: Dependencia de Firebase en gradle

Del lado del servidor enviamos una notificación con una serie de campos necesarios para identificar la ruta a un usuario concreto. Este usuario lo identificamos por un token que almacenamos en la base de datos, este token es característico de cada móvil. Para que nuestra aplicación pueda recibir y entender esta notificación necesitaremos el siguiente fragmento de código.

```
1 @Override
2 public void onMessageReceived(RemoteMessage remoteMessage) {
3     Map<String, String> data = remoteMessage.getData();
4     titulo= data.get("titulo");
5     nombreGrupo= data.get("nombreGrupo");
```

CAPÍTULO 7.

Implementación

7.1 Implementación

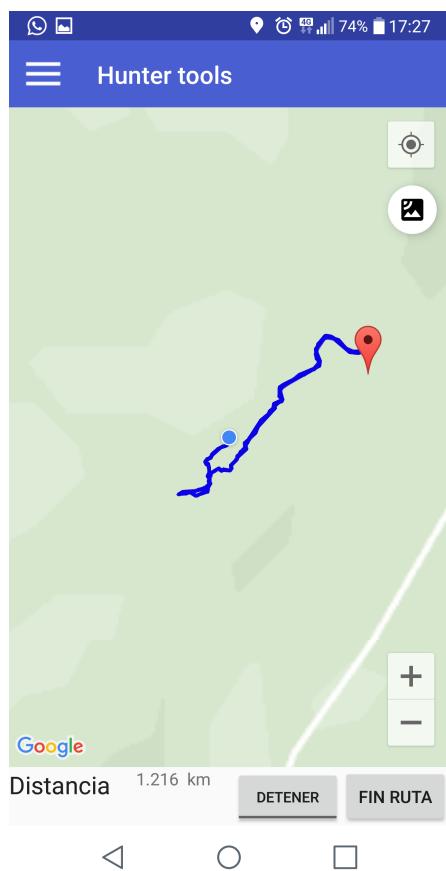


Figura 7.8: Visualización de la ruta mientras se realiza

CAPÍTULO 7.

Implementación

7.1 Implementación

```
6     accion= data .get("accion");
7     idRuta= data .get("idRuta");
8     verAccionString ();
9     EventBus.getDefault() .post(remoteMessage
10    .getData() .toString()));
11
12 }
```

Código 7.10: Recepción de mensajes

Este método pasearía el JSON que envía el servidor y guardaría los datos necesario en el móvil. En la siguiente captura veríamos la notificación en el móvil.

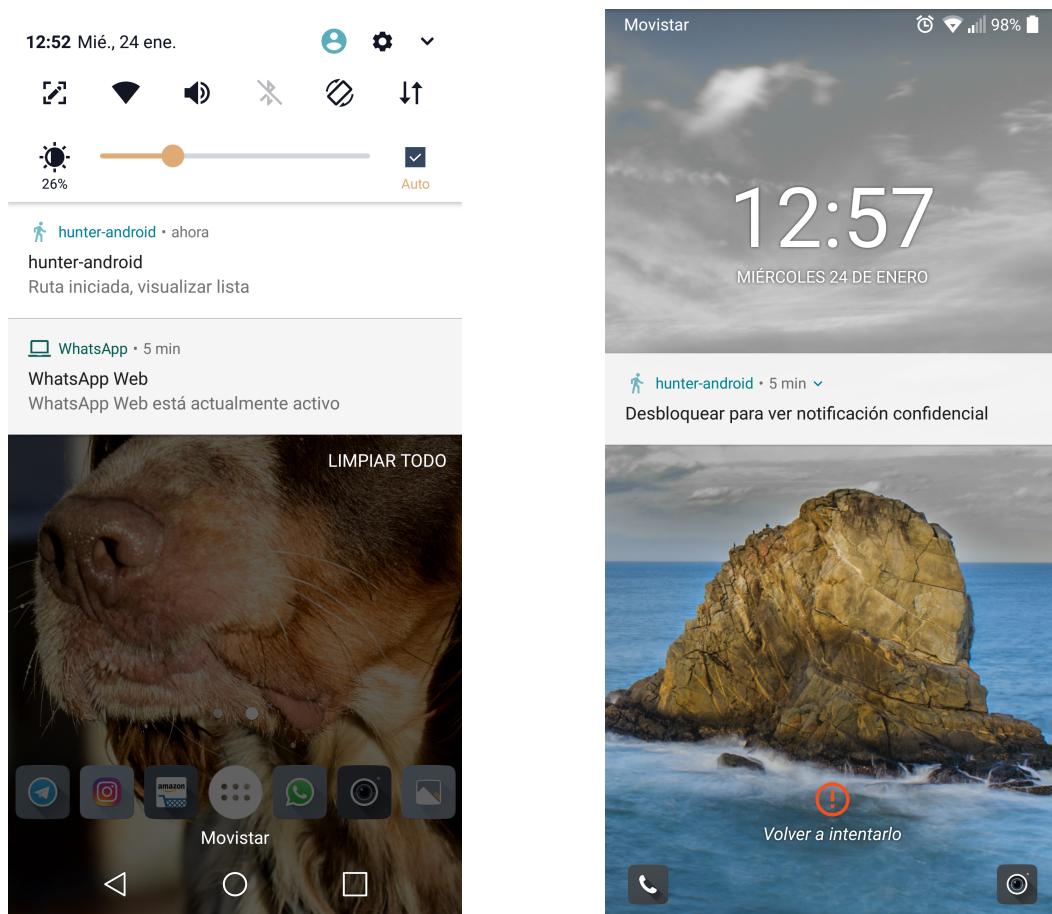


Figura 7.9: Notificación push

7.2. Pruebas

Las pruebas son un tema muy importante a tener en cuenta en cualquier proyecto de software. Nos permiten conocer ciertos aspectos que fallan en el sistema de manera objetiva y conocer ciertos riesgos que se nos pasaron por alto en la implementación. Dado que usamos la metodología Scrum y como describimos en el capítulo de seguimiento al final de cada Sprint se harán las pruebas para esas historias de usuario.

7.2.1. Pruebas de unidad

Las pruebas de unidad se encargan de comprobar el buen funcionamiento de partes de código. Esta comprobación se realiza normalmente a nivel de clase asegurándonos que el funcionamiento es el adecuado.

En nuestro caso las pruebas de unidad se realizaron en el servidor ayudándonos del framework de pruebas JUnit. Como ya se comentó JUnit es una tecnología que ayuda a la ejecución de pruebas para la comprobación del funcionamiento de métodos o clases.

Como también se usó el framework Spring para el desarrollo este también fue empleado para los test ya que ayuda a la inyección de dependencias y otras gestiones de las transacciones.

```
1 @RunWith(SpringJUnit4ClassRunner.class)
2 @WebAppConfiguration
3 @Profile("test")
4 @ContextConfiguration("classpath:root-context.xml")
5 public class UsuarioServiceTest {
6
7     @Autowired
8     private UsuarioService usuarioService;
9
10    @Test
11    public void testPruebaUsuario() {
12        Usuario u1 = new Usuario("tito", "titef@udc.es", "pass");
13        u1 = usuarioService.save(u1);
14        Usuario u2;
15        u2 = usuarioService.findByCorreo("titef@udc.es");
16        (u2.getNombre()).equals("tito");
17        usuarioService.delete(u1.getIdusuario());
18    }
```

7.2.2. Pruebas de integración y aceptación

Las pruebas de integración entre los componentes del sistema y las de aceptación se realizaron de forma manual. En ellas se comprobaba el funcionamiento y la respuesta antes las acciones realizadas.

Como el servidor estaba desplegado en un servidor de aplicaciones las pruebas se podían realizar de 2 maneras:

- Mediante el emulador propio de Android.
Esta opción fue la utilizada para depurar la aplicación en los primeros Sprint ya que era más cómodo a la hora realizar las pruebas usar el teclado.
- Mediante un móvil en el cual se instalaba nuestra aplicación. En cambio esta opción fue la más utilizada en los últimos sprints ya que requerían unas pruebas más reales en lugares abiertos, ya que se perseguía la comprobación de la precisión del GPS en casos reales de uso. Esto fue clave para mejora de la creación de rutas.

Capítulo 8

Conclusiones y trabajo futuro

8.1. Trabajo realizado

Una vez finalizado este Trabajo de Fin de Grado tenemos una aplicación móvil en Android que cumple todos los objetivos marcados al inicio del proyecto y en particular de esta memoria. La aplicación permitirá al usuario gestionar sus grupos, gestionar sus puntos de interés y guardar sus rutas tanto las que hace de manera individual como las que hace de manera conjunta. Esto es posible gracias al sistema de localización del GPS.

A continuación se muestran las principales características del producto construido:

- El usuario podrá registrarse a través de las funcionalidades que ofrecemos y llevar un seguimiento de sus actuaciones en el campo.
- El usuario podrá guardar puntos estratégicos en el mapa acompañados de un nombre clave y una breve descripción. Esto nos ayudará a recordar puntos para futuras aventuras.
- Crear grupos de compañeros y poder añadirlos a los mismos para compartir rutas.
- Guardar las rutas seguidas en nuestras aventuras de pesca y de caza, como también la posibilidad de rememorar esas rutas al poder revisarlas gracias a nuestras lista con las rutas realizadas.

- Y por último la posibilidad de realizar rutas conjuntas con un grupo de amigos que nosotros queramos y así compartir nuestra ubicación en todo momento. Esto nos ayudará en la pesca ya que al conocer la ubicación del resto de integrantes del grupo podríamos socorrerlo si algo le pasa en un sitio de difícil acceso. Como también por tema de seguridad en una jornada de caza ya que si estamos cerca de otro usuario lo sabríamos en el momento pudiendo evitar accidentes.

8.2. Trabajo futuro

La aplicación cumple con los objetivos marcados pero como todo proyecto software puede ser ampliado y mejorado. Una vez finalizado este proyecto, se podrían añadir nuevas funcionalidades:

- Añadir nuevos tipos de PDIs como es el caso de fotografía. Ya que tiene varias semejanzas con la caza y la pesca por los lugares donde se realiza sería una funcionalidad interesante.
- Como toda actividad que se realiza al aire libre esta condicionada para bien o para mal por condiciones meteorológicas, se podrían usar los servicios de Open-WeatherMap. Esto nos ayudaría a planear mejor nuestras jornadas de pesca, caza o fotografía.
- Otro punto interesante también sería poder añadir a cada PDI una foto asociada a él, lo que ayudaría a acordarse mejor del lugar y ubicarse con precisión.
- Dar al usuario la capacidad de enviar mensajes a otros usuarios cuando comparten una ruta.

Índice de Figuras

1.1. Visualizar mapa con iHunt	10
1.2. Registro de un punto con iHunt	10
1.3. Planificación con iHunt	10
1.4. Registro de trofeo con iHunt	10
1.5. Opciones de la aplicación	12
1.6. Puntos de pesca guardados	12
1.7. Datos de una captura	12
1.8. Datos sobre la meteorología	12
2.1. Entorno de trabajo Eclipse	14
2.2. Captura de pantalla realizando app debug	16
2.3. Emulador de Android con mi aplicación	17
3.1. Metodología Scrum	23
3.2. Parte metodología Scrum, Product Backlog	23
3.3. Parte metodología Scrum, Sprint Planning	24
3.4. Parte metodología Scrum, Sprint Backlog	24
3.5. Parte metodología Scrum, Daily Scrum	25
4.1. Casos de uso del actor Usuario No Autenticado	30
4.2. Casos de uso de gestión de puntos de interés	31
4.3. Casos de uso de gestión de grupos de usuarios	32
4.4. Casos de uso de gestión de rutas	33
4.5. Identificación y clasificación de riesgos del proyecto	33

ÍNDICE DE FIGURAS

ÍNDICE DE FIGURAS

4.6. Iniciar sesión	35
4.7. Registrar usuario	35
4.8. Crear grupo	36
4.9. Añadir usuarios a grupo	36
4.10. Listar grupos	37
4.11. Opciones generales	37
4.12. Crear PDI	38
4.13. Visualizar PDIs	38
4.14. Ruta individual	39
4.15. Ruta compartida	39
5.1. Costes asociados a este proyecto	42
5.2. PDI pesca	44
5.3. PDI caza	44
5.4. Guardar PDI	45
5.5. Borrar PDI	45
5.6. Iniciar sesión	46
5.7. Registro del usuario	46
5.8. Captura del menú de navegación	47
5.9. Crear grupo	48
5.10. Añadir usuarios a grupo	48
5.11. Ver grupos del usuario	49
5.12. Ver integrantes del grupo	49
5.13. Crear ruta	51
5.14. Navegación ruta individual	51
5.15. Eliminar ruta	52
5.16. Ruta seguida vista satélite	52
5.17. Lista de rutas creadas	53
6.1. Esquema general de la arquitectura del sistema	56
6.3. Entidad relación	57
6.2. Diagrama de componentes del sistema	58

*ÍNDICE DE FIGURAS**ÍNDICE DE FIGURAS*

6.4. Arquitectura del servidor	60
6.5. JSON	62
6.6. Servicios de la aplicación móvil	64
7.1. Interfaz Usuario	72
7.2. Interfaz Grupo	72
7.3. Interfaz Ruta	73
7.4. Interfaz PuntoRuta	73
7.5. Interfaz Poi	73
7.6. Marker antes de guardar el PDI	76
7.7. Marker después	76
7.8. Visualización de la ruta mientras se realiza	78
7.9. Notificación push	79

Índice de Códigos

2.1. Inyección de dependencias	15
6.1. JpaRepository	62
7.1. Adaptación a la clase Usuario	65
7.2. Interfaz de UsuarioRepository	66
7.3. Dependencia de Google Maps en gradle	74
7.4. Captura de clic en la pantalla	74
7.5. Creación de Marker personalizados	75
7.6. Dependencia de Location en gradle	75
7.7. Obtencion de coordenadas por intervalo de tiempo	75
7.8. Fragmento para el calculo de distancias entre puntos	77
7.9. Dependencia de Firebase en gradle	77
7.10. Recepción de mensajes	77

Apéndice A

Glosario

Framework soporte tecnológico compuesto por artefactos o módulos de software que sirven para el desarrollo del software.

Iteración, en las metodologías ágiles, como es la Scrum, es una unidad de tiempo en la que se desarrollan las funcionalidades de la aplicación.

MVC Model-view-controller: es un patrón arquitectónico para implementar interfaces de usuario. Divide el software en tres componentes. El modelo es el encargado de gestionar los datos y la lógica de negocio. La vista se encarga de la representación gráfica de la información. Por último, el controlador es el encargado de poner en comunicación los otros componentes.

Apéndice B

Bibliografía

- [1] Google, Inc. [2014]. *Material design guidelines*.
<https://material.io/guidelines/>
- [2] Google, Inc. [n.d.]. *Google's android developer guide*.
<https://developer.android.com>
- [3] Girones3 J. T. [2016]. *El gran libro de Android*, 5a edn, Marcombo, Barcelona, España.
- [4] Red Hat Inc. [2016]. *Hibernate orm documentation 5.2*.
<http://hibernate.org/orm/documentation/5.2/>
- [5] Walls, C. and Castrillo, L. A. [2015]. *Spring in Action*, 4th edn, Manning.
- [6] Scrum en la Metodología Ágil.
<http://www.i2btech.com/blog-i2b/tech-deployment/para-que-sirve-el-scrum-en-la-metodologia-agil/>
- [7] Google Cloud Platform + Firebase.
<https://firebase.google.com/?hl=es-419>
- [8] Miguel Ángel Moreno Edición Informática64, 2013. *Desarrollo de aplicaciones Android seguras*
- [9] Henrik Kniberg [2008]. *Scrum y XP desde las trincheras*, 2th edn, InfoQ.
- [10] Spring RequestMapping
<http://www.baeldung.com/spring-requestmapping>

CAPÍTULO B.
Bibliografía

- [11] *Getting start with Android Material Designer*
<https://app.pluralsight.com/>