



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, 18 de enero 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, 18 de enero de 2018
Calificación:	

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

y

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 Trabajo de Fin de Grado en el Grado de Enxeñería Informática.

En A Coruña, a 18 de enero de 2018

JAVIER PARAPAR LÓPEZ
Director

ÓSCAR PEDREIRA FERNÁNDEZ
Director

Dedicatoria

Agradecimientos

A rufo y a isima

Author name

A Coruña, 18 de enero de 2018

Resumen

This is the summary of the project. No more than one page, please.

Palabras clave

FIRST KEYWORD, SECOND KEYWORD, ETC.

Índice general

1. Introducción	7
1.1. Motivación	7
1.2. Objetivos	7
1.3. Estructura de la memoria	7
1.4. Plan de trabajo	8
2. Tecnología	9
2.1. Eclipse Java EE IDE for Web Developers Version Neon2	9
2.2. Spring	10
2.3. Android Studio	11
2.4. Git	14
2.5. Maven	14
2.6. Gradle	15
2.7. Jackson	15
2.8. JPA/Hibernate	15
2.9. PostgreSQL	15
2.10. JUnit	16
3. Proceso de ingeniería	17
3.1. Scrum	17
3.1.1. Sprint	18
3.1.2. Participantes	18
3.1.3. Cómo funciona el Proceso	19

3.2. Adaptación de este proyecto a la metodología	22
4. Analisis	23
4.1. Análisis de Requisitos	23
4.1.1. Actores	24
4.1.2. Casos de uso	24
4.2. Maquetas	29
5. Planificación y seguimiento	35
5.1. Planificación	35
5.2. Estimación de tiempos y costes	36
5.3. Seguimiento	37
5.3.1. Sprint 1	37
5.3.2. Sprint 2	37
5.3.3. Sprint 3	37
5.3.4. Sprint 4	38
5.3.5. Sprint 5	38
5.3.6. Sprint 6	39
5.3.7. Sprint 7	40
5.3.8. Sprint 8	40
6. Diseño	41
6.1. Esquema general de la arquitectura	41
6.2. Modelo de datos	44
6.3. Servidor	44
6.3.1. Servicio web	45
6.3.2. Organización dos paquetes	46
6.3.3. Transmisión de la información	46
6.3.4. Gestión de las clases persistentes	47
6.4. Aplicación Móvil	48
6.4.1. Servicios	48
6.4.2. Organización de los paquetes	49

7. Implementación	51
7.1. Implementación	51
7.1.1. Servidor	51
7.1.2. Aplicación móvil Android	51
7.2. Pruebas	51
7.2.1. Pruebas de unidad	51
7.2.2. Pruebas de unidad y de integración	51
8. Conclusiones y trabajo futuro	53
8.1. Investigación realizada	53
8.2. Características del proyecto	53
8.3. Trabajo futuro	54
Índice de Tablas	55
Índice de Figuras	57
Apéndices	59
A. Glosario	59
B. Bibliografía	63

Capítulo 1

Introducción

Intro

1.1. Motivación

1.2. Objetivos

- Y
- X

1.3. Estructura de la memoria

La memoria del presente proyecto está estructurada del siguiente modo:

- **Introducción** Explica el contexto en el que se enmarca el proyecto, introduce la problemática a tratar y detalla el alcance y objetivos del mismo desde un punto de vista global. También muestra la estructura de la memoria y el plan de trabajo seguido.
- **Tecnología** Describe y justifica las principales tecnologías empleadas para desarrollar el objeto del proyecto atendiendo a los requisitos del mismo.
- **Conceptos**

- **Proceso de ingeniería** Detalla el proceso de ingeniería: la metodología, la planificación y la gestión del proyecto.
- **Desarrollo** Realiza una descripción detallada del análisis, diseño, implementación, pruebas y despliegue del sistema.
- **Conclusiones y trabajo futuro** Proporciona una evaluación global del producto obtenido así como futuras líneas de trabajo que se podrían explotar alrededor del proyecto.
- **Apéndices** Está compuesto por las siguientes secciones complementarias:
 - **Glosario** Define los términos y acrónimos técnicos empleados en la memoria del proyecto.
 - **Bibliografía** Recoge la documentación bibliográfica sobre la que se apoya el proyecto.

1.4. Plan de trabajo

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 esta sección explicaremos los entornos en los que se trabajó a la hora de desarrollar este proyecto, comentando los fundamentos usados en ellos.

2.1. Eclipse Java EE IDE for Web Developers Version Neon2

Herramienta para desarrolladores integrado (IDE) en Java para el desarrollo de software. Fue creado por IBM como sucesor de uno de sus herramientas y ahora esta en manos de la Fundación Eclipse que es quien se encarga de seguir desarrollándolo. Esta herramienta de desarrollo es totalmente gratuita por lo que no encarece el precio del proyecto. Se decidió usar esta herramienta por el parecido que tiene con el entorno de Android y sus facilidades para la creación de código.

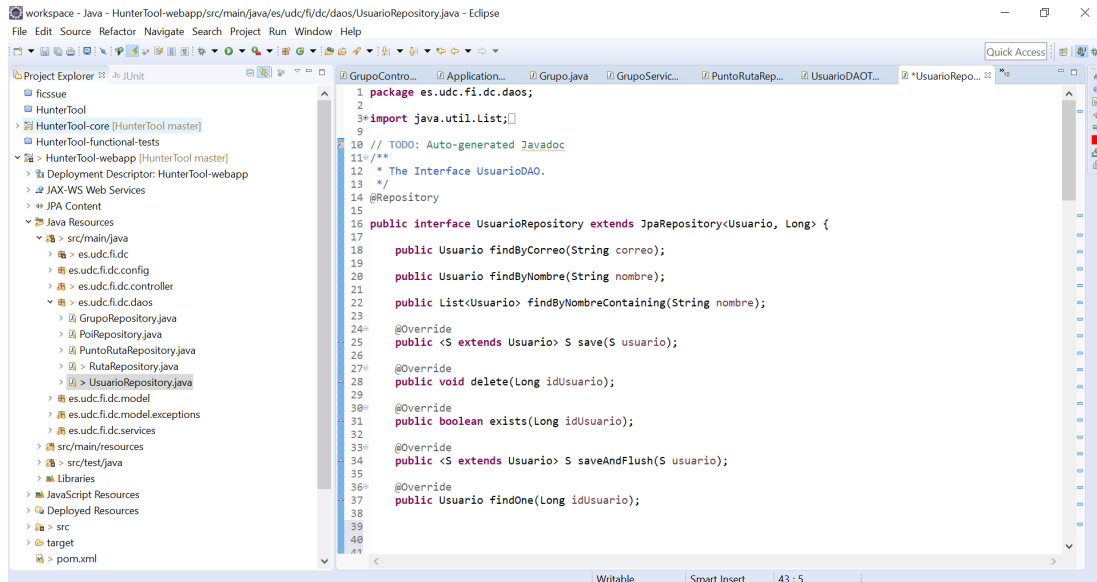


Figura 2.1: Entorno de trabajo Eclipse

2.2. Spring

Spring es un framework 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. Simplemente es necesario añadir una dependencia, si se usa Eclipse, en el archivo pom.xml.

```

64 <dependency>
65     <groupId>org.springframework</groupId>
66     <artifactId>spring-context</artifactId>
67     <version>4.3.6.RELEASE</version>
68 </dependency>

```

Figura 2.2: Dependencia de Spring en el pom.xml

Como funcionalidades principales que nos ofrece y por ello elegí, serían:

- **Programación orientada aspectos**, paradigma que nos ayuda a eliminar depen-

dencias entre módulos acortando las líneas de código de los servicios para así poder centrarnos en la lógica de la aplicación. Lo que conlleva a 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 lo consigue haciendo que una clase le proporcione a otra sus dependencias haciendo que la otra no tenga que crearlas ella misma. De este modo a través de un interfaz una clase tiene las dependencias de otra sin tener que preocuparse de la implementación de las mismas, lo que es favorable para reducir el acoplamiento.

```
@Autowired
GrupoRepository grupoDAO;

public void setGrupoDAO(GrupoRepository grupoDAO) {
    this.grupoDAO = grupoDAO;
}
```

Figura 2.3: Ejemplo de inyección de dependencia

2.3. Android Studio

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

- Realizar compilaciones de nuestro código para comprobar el estado de nuestras variables en los puntos que nosotros le indiquemos sin necesidad de generar un APK.

2.3 Android Studio

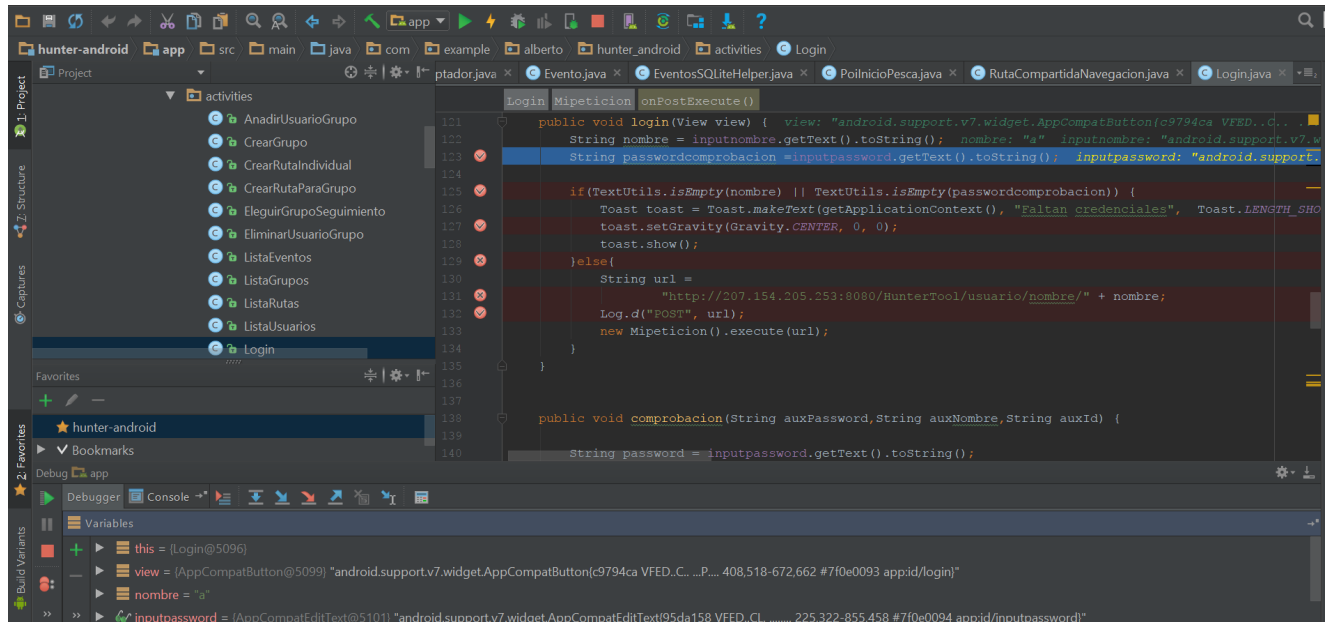


Figura 2.4: Captura de pantalla realizando app debug

- 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.

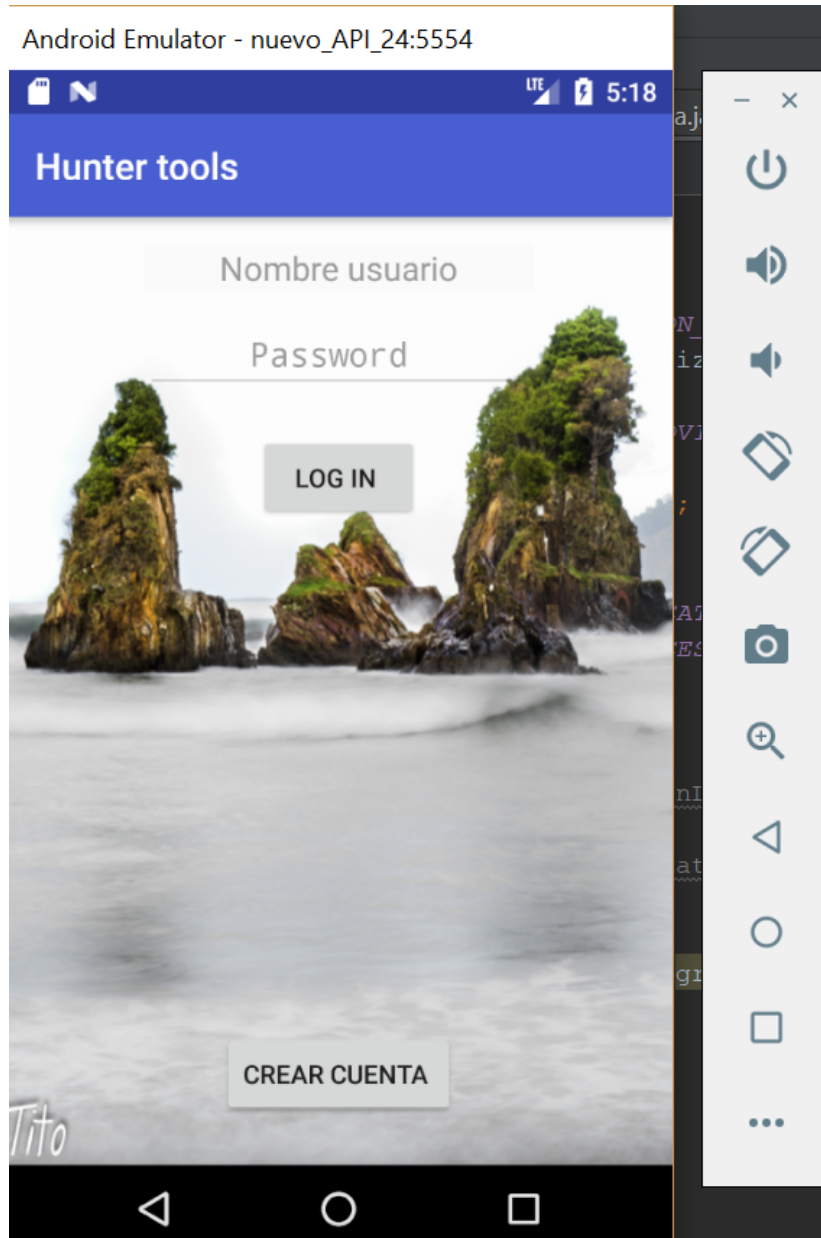


Figura 2.5: Emulador de Android con mi aplicación

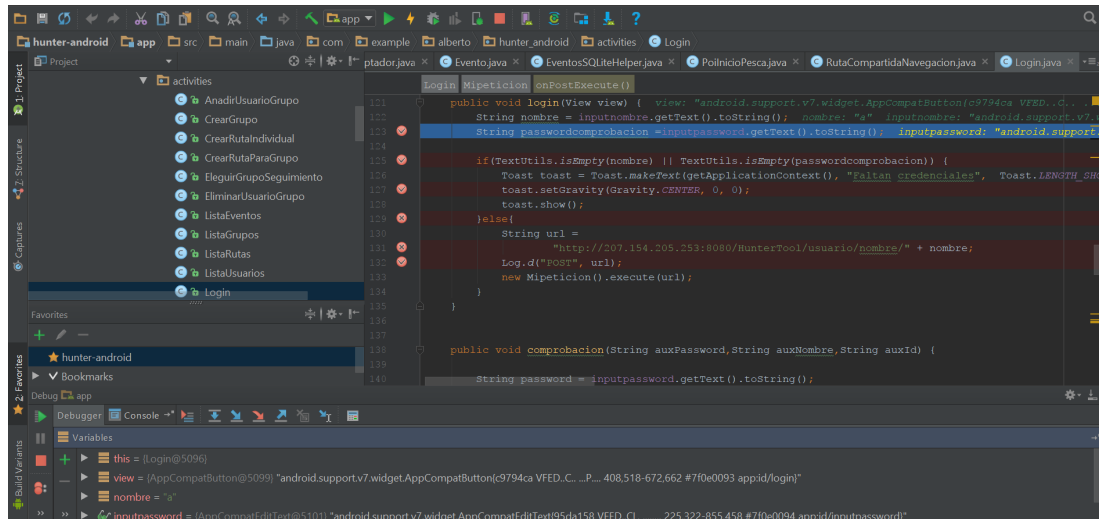


Figura 2.6: Entorno de trabajo Android Studio

2.4. Git

Git es un software para ayudarnos con el control de versiones pensado para el mantenimiento de múltiples versiones de aplicaciones cuando estas tienen un número muy alto de archivos y queremos poder volver a un punto anterior o como copias de seguridad. A su vez Git ofrece 2 opciones para el control de versiones una mediante un interfaz y otra mediante línea de comandos. Se escogió esta opción por las facilidades de uso que ofrece a la hora de realizar los commits y el fácil nombrado de los mismos cuando queremos actualizar el repositorio.

2.5. 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 con otros módulos como jpa, firebase o postgres. Viene con objetivos predefinidos para la compilación del código o su empaquetado. Se escogió por la buena estructuración de los paquetes que genera y por las similitudes que presenta frente al Gradle que usaremos en el entorno de desarrollo Android Studio.

2.6. Gradle

Gradle es una herramienta de software para la gestión, construcción de proyectos Android 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.

2.7. Jackson

Jackson es un parseador 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 y dado que se usa el framework Spring este parseador ya viene integrado.

2.8. JPA/Hibernate

Hibernate 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 ayuda a no tener que definir la entidad directamente en nuestro gestor de base de datos. Para conseguir esto debemos de usar las anotaciones pertinentes en el entorno de desarrollo (Eclipse) sin necesidad de acceder al gestor de base de datos en ningún momento. Una vez puestas las anotaciones los accesos a datos de la bases de datos pasan a ser muy sencillos.

2.9. PostgreSQL

PostgreSQL es un gestor de bases de datos objeto-relacional que permite trabajar con grandes cargas de datos consiguiendo una tolerancia alta a errores. Se decidió usar este gestor porque tiene una gran adaptabilidad a otros entornos de trabajo lo que ayuda a ganar agilidad y eficiencia. 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.

2.10. 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 testar.

Con estas tecnologías se ha llevado a cabo la construcción de los distintos componentes del sistema siguiendo los pasos que comentaremos en el próximo capítulo.

Capítulo 3

Proceso de ingeniería

En este capítulo se justifica 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. Se ha optado por esta solución frente a otras clásicas como la de proceso unificado porque al usar un modelo incremental, basado en iteraciones, aseguramos que al final de cada etapa tengamos un producto operativo y que en sucesivas etapas se irá completando. Además pone el foco en la calidad del producto y no en la calidad de los procesos.

3.1. Scrum

El Scrum es un proceso de la Metodología Ágil que se usa para minimizar los riesgos durante la realización de un proyecto. Esta metodología está basada en ciclo iterativos lo que se traduce en una gran flexibilidad a la hora de adaptarse a cambios en el proyecto o sobre el control de riesgos.

Entre las ventajas se encuentran la productividad, calidad y que se realiza un seguimiento diario de los avances del proyecto, logrando que los integrantes estén unidos, comunicados. Esta cualidad no será palpable ya que en el caso de este proyecto el equipo estará formado por una sola persona.

3.1.1. Sprint

Por Sprint entendemos un bloque de tiempo fijo de entorno a 1 mes 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 planteadas en él.

3.1.2. Participantes

- **Product Owner**

Habla por el cliente, y asegura que el equipo cumpla las expectativas. Es “el jefe”, sus decisiones deben ser respetadas y acatadas. Estas decisiones influirán en la prioridad de la lista de tareas o Product Backlog.

- **Scrum Master**

Es el encargado de liderar la reuniones y ayudar al equipo en los problemas que tengan. Debe minimizar las imposibilidades que surgen en el desarrollo del proyecto para que se cumplan los objetivos del Sprint. 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 cumplir lo que les asigna el Product Owner. En sí 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.

3.1.3. Cómo funciona el Proceso

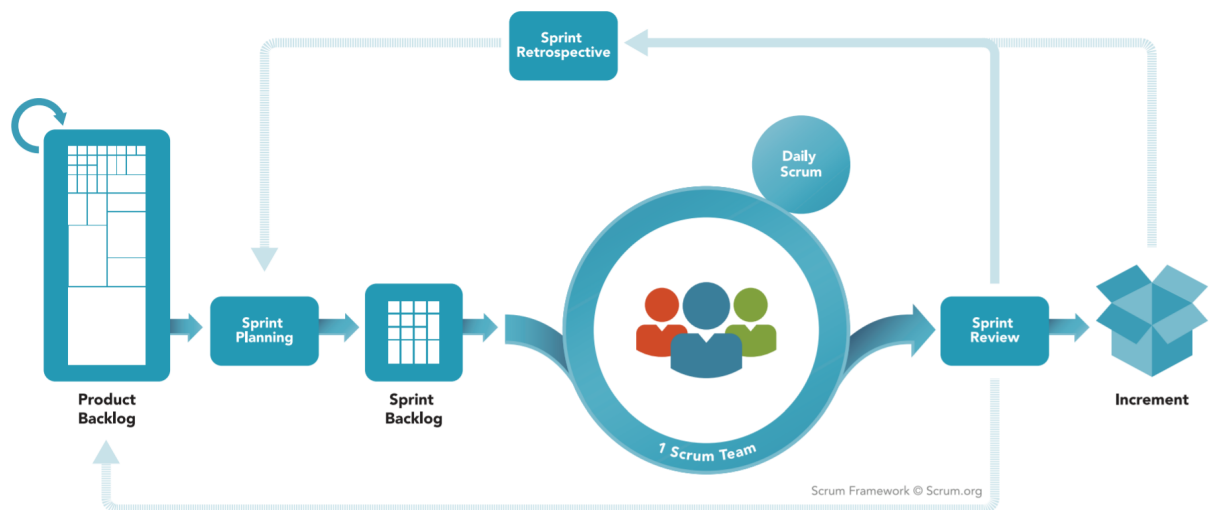


Figura 3.1: Metodología Scrum

El proceso comienza con la definición del Product Backlog, se dará pie a los futuros Sprints.

■ Definición del Product Backlog

Es una lista con las funcionalidad de la aplicación.

Estas funcionalidad 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 contendrá varios casos de uso.

Está elaborado por el Product Owner y las funcionalidad están ordenadas de mayor a menor importancia dentro de la aplicación. La finalidad del Product Owner es plantear lo que hay que hacer. Con las historias ordenas se indicaran los Sprints que serán necesarios pudiendo hacer varias historias en un mismo Sprint pero nunca una historia dividida en 2 Sprints.

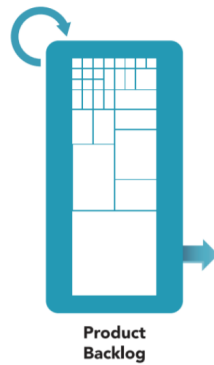


Figura 3.2: Parte metodología Scrum

- **Sprint Planning Meeting**

Esta reunión se hace al comienzo de cada Sprint y se define cómo se va a enfocar el proyecto que viene del Product Backlog las etapas y los plazos.



Figura 3.3: Parte metodología Scrum

- **Sprint Backlog**

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

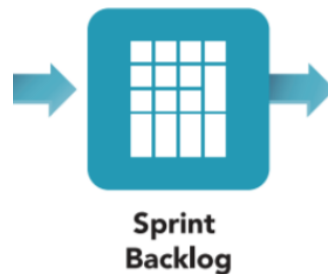


Figura 3.4: Parte metodología Scrum

- **Daily Scrum o Stand-up Meeting**

Es una reunión breve que se realiza a diario mientras dura el periodo de Sprint. En estas reuniones si algun miembro del equipo encuentra algun problema esto se comenta y se trata de ayudarlo. Este tipo de problemas debería resolverlos el Scrum Master.

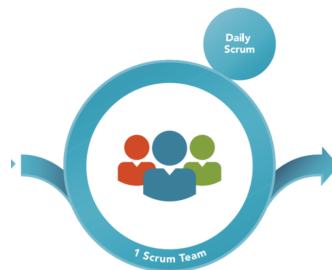


Figura 3.5: Parte metodología Scrum

- **Sprint Review**

Esta sería la primera reunión una vez acabado el Sprint. En ella se deberían plan-tear las tareas acabadas y se debería ver un avance claro para ser presentado al cliente. Las tareas inacabadas 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 poderlo resolver en el siguiente Sprint.

3.2. Adaptación de este proyecto a la metodología

Capítulo 4

Analisis

4.1. Análisis de Requisitos

En esta aplicación móvil objetivo de este proyecto, se establecieron una serie de requisitos que debería cumplir la aplicación. Estos requisitos fueron obtenidos al hacer una entrevista con los directores del proyecto.

El registro del usuario sería el primero para poder darle acceso y comenzar a usarla, ya que sin el no tiene acceso a ninguna funcionalidad de la aplicación.

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 terreno.

El usuario también podrá 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. Finalmente 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 la ya se registro en otro momento.
- **Usuario autenticado.** Usuario autenticado que puede acceder a todas as funcionalidades del sistema.

4.1.2. 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 iniciará sesión automáticamente con el nuevo perfil.
- **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.

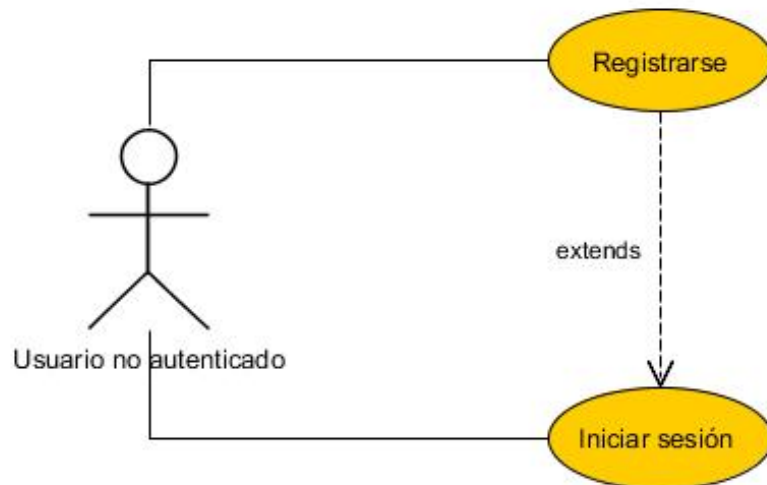


Figura 4.1: Casos de uso del actor Usuario No Autenticado

- **Usuario autenticado**

Para el caso de que el usuario ya esté autenticado dividiremos en 3 grupos los casos de usos:

- **Gestión de puntos de interés**

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

- ***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.

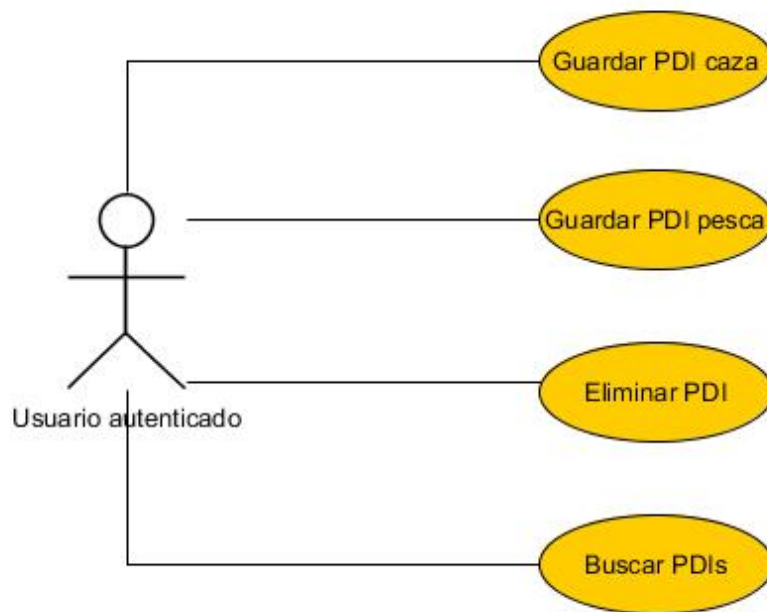


Figura 4.2: Casos de uso de gestión de puntos de interés

■ Gestión de grupos

- *R-G-1 Crear grupo*, el usuario crea un grupo con nombre único.
- *R-G-2 Añadir integrantes*, el usuario busca en la base de datos los usuarios que quiere integrar en el grupo previamente seleccionado.
- *R-G-3 Eliminar integrantes*, el usuario puede eliminar los integrantes que vea pertinentes.
- *R-G-4 Ver grupos*, el sistema listar los grupos en los que el usuario está registrado.
- *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.

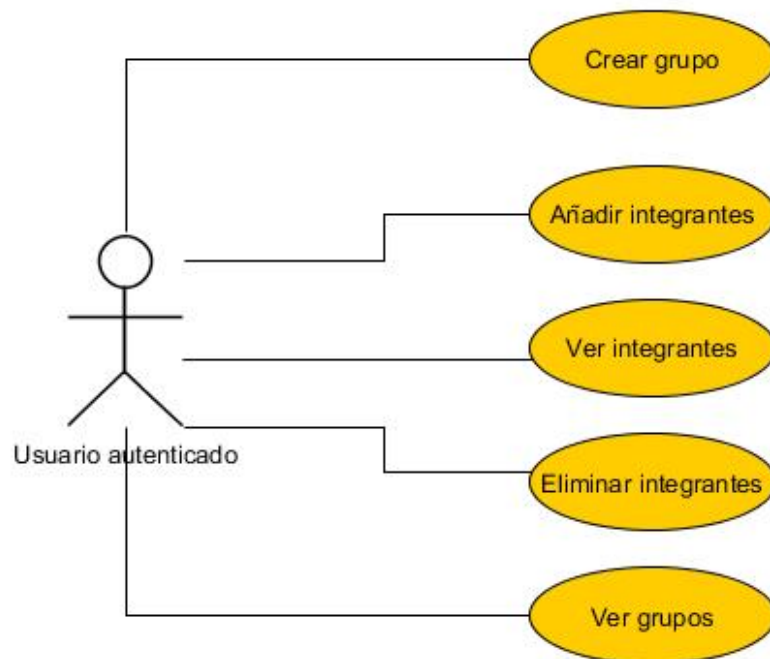


Figura 4.3: Casos de uso de gestión de grupos de usuarios

- **Gestión de rutas**, el usuario iniciará una navegación privada siendo guardada la ruta seguida.
 - **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 navegando 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 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).
 - **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 usuario que estén también navegando. 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 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.

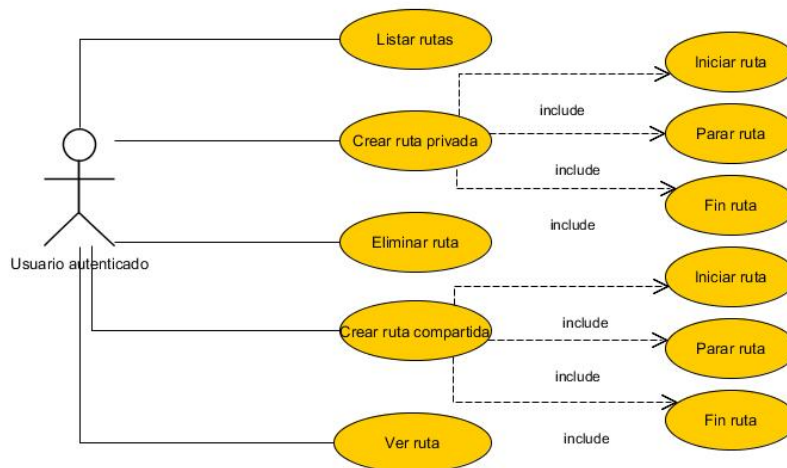


Figura 4.4: Casos de uso de gestión de rutas

4.2. Maquetas

Una vez obtenidos los casos de uso, comenzamos creando las maquetas de como debería que ser a interface del usuario. Se busca que sea una interface simple y intuitiva. Para hacer esto posible seguiremos las pautas y usaremos los elementos de Material Design.

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.

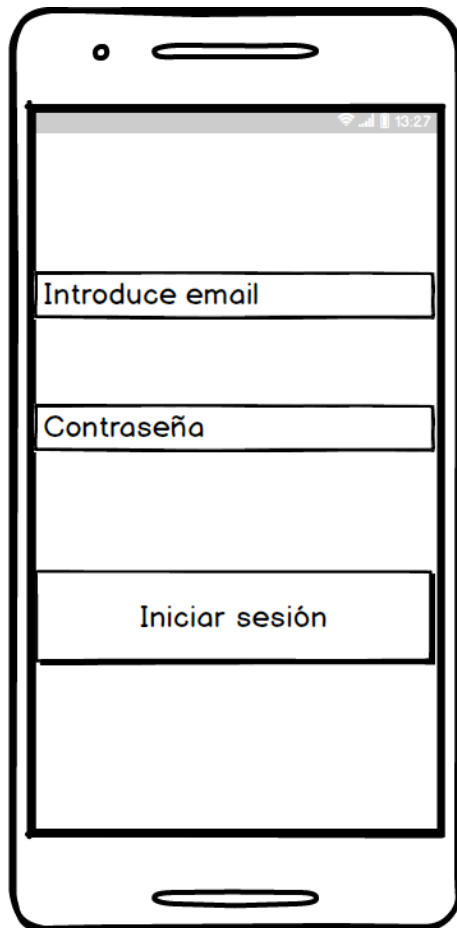


Figura 4.5: Iniciar sesión

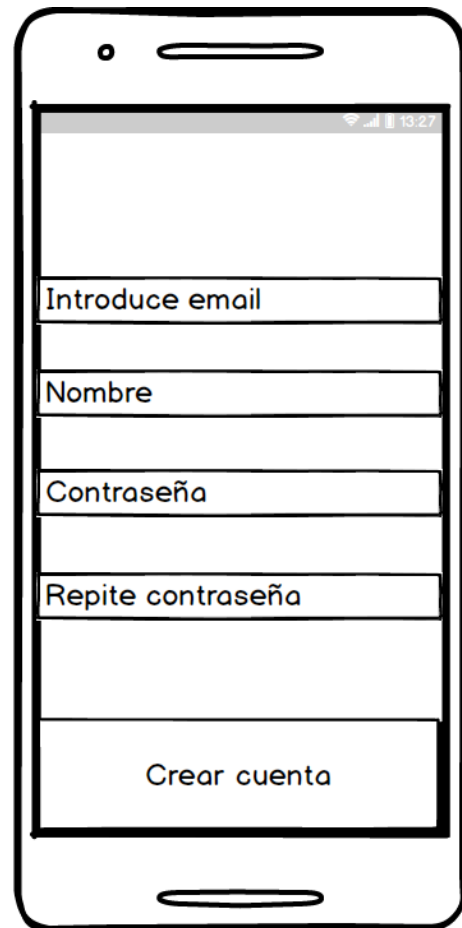


Figura 4.6: Registrar usuario

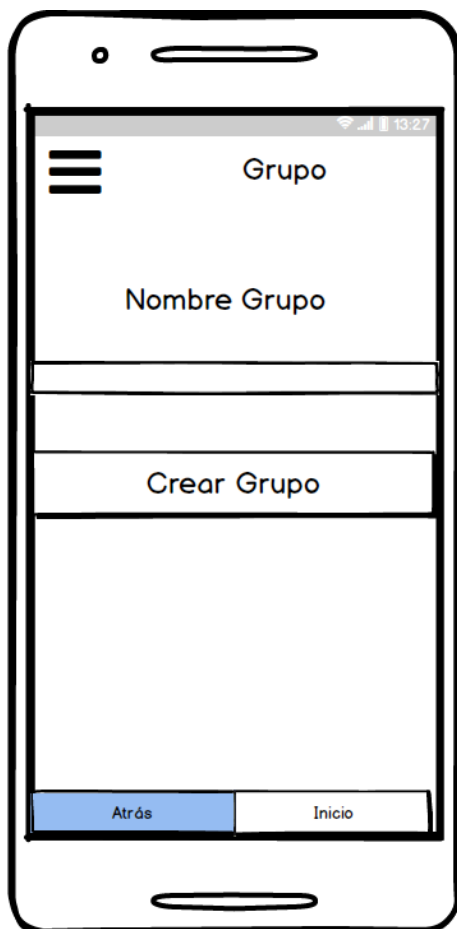


Figura 4.7: Crear grupo

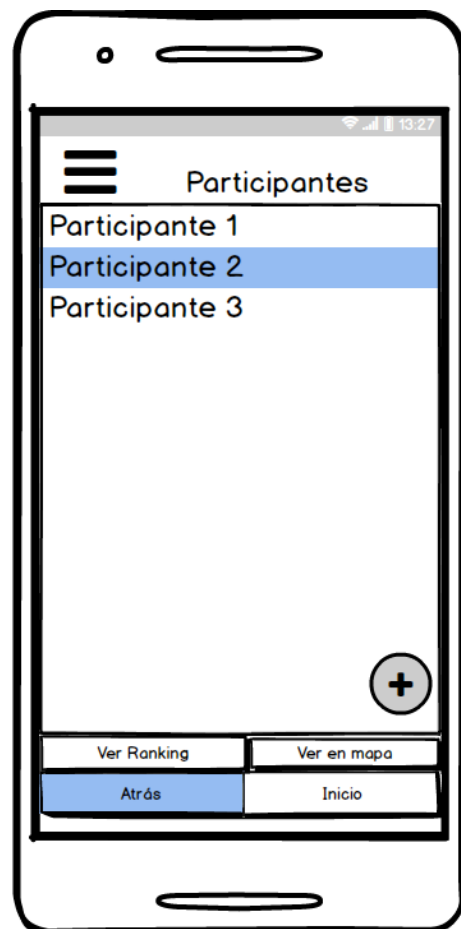


Figura 4.8: Añadir usuarios a grupo

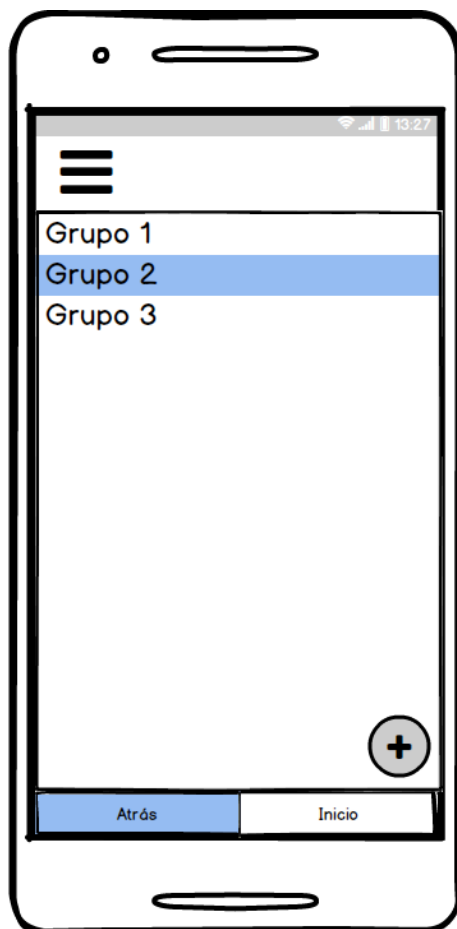


Figura 4.9: Listar grupos

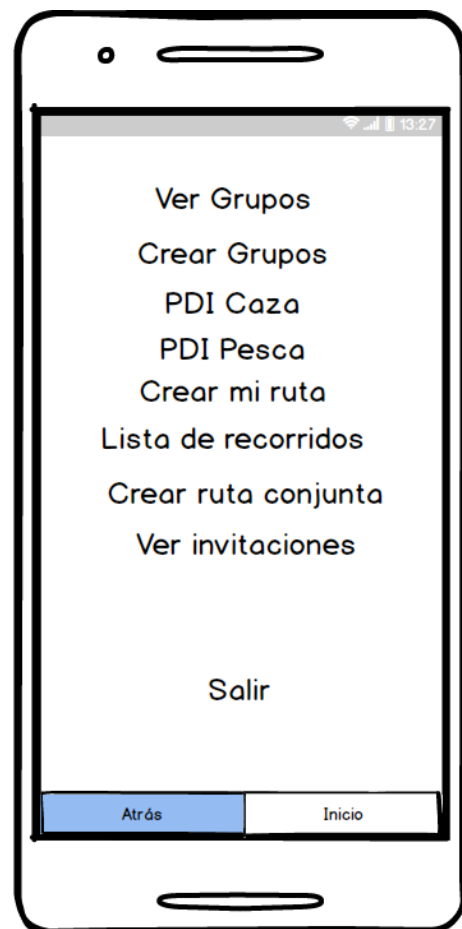


Figura 4.10: Opciones generales

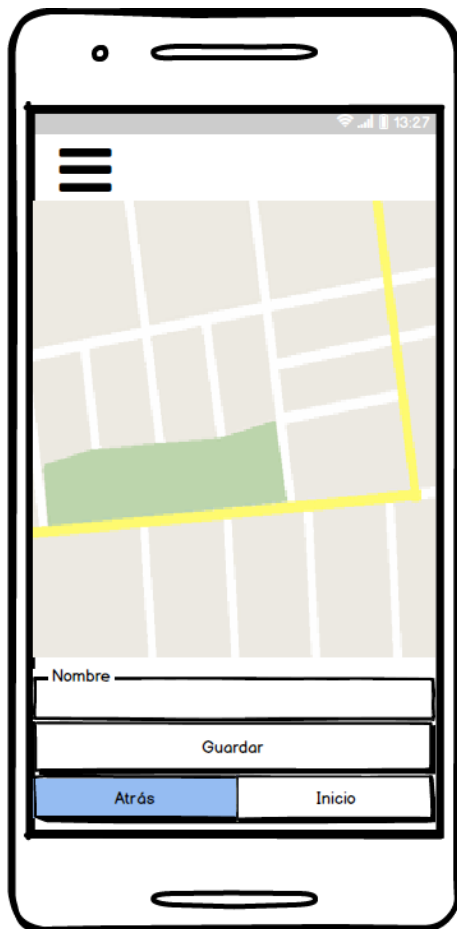


Figura 4.11: Crear PDI



Figura 4.12: Visualizar PDIs

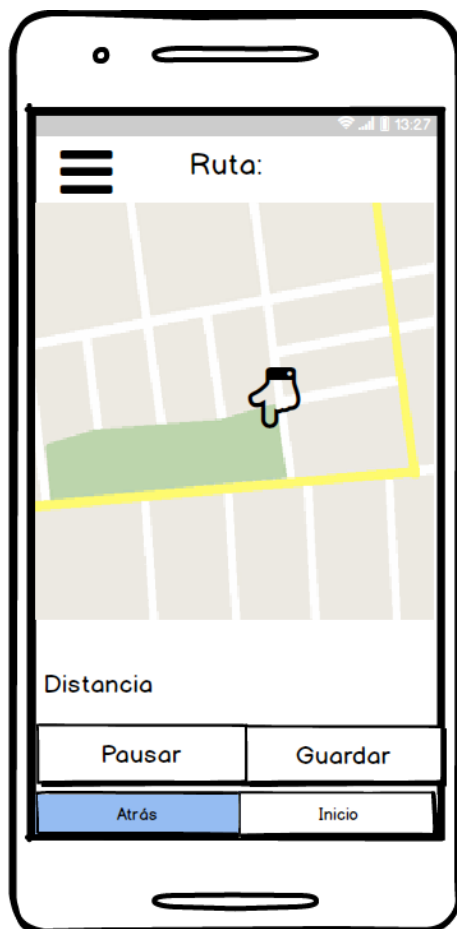


Figura 4.13: Ruta individual

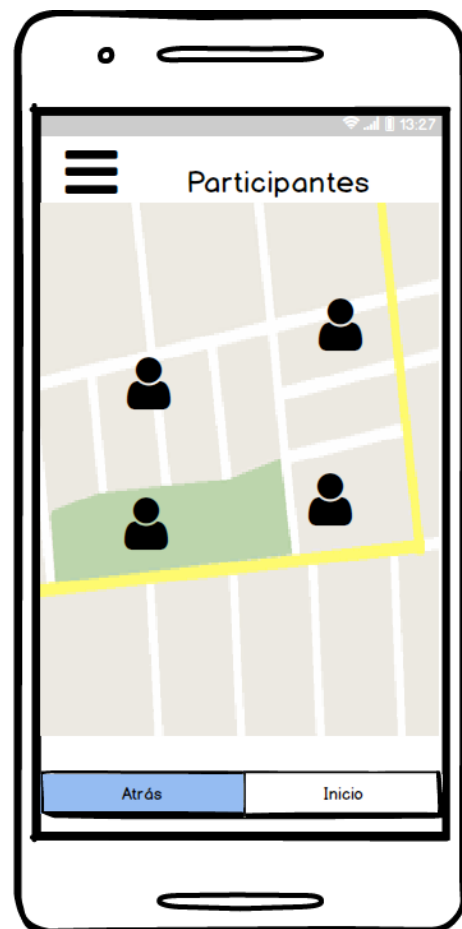


Figura 4.14: Ruta compartida

Capítulo 5

Planificación y seguimiento

En este capítulo se comentará el proceso seguido para la realización de este proyecto detallando la planificación, diseño y el desarrollo del mismo estructurado en etapas.

5.1. Planificación

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. Éste 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 tareas a realizar y quién las desempeña. En este caso al solo trabajar yo en el proyecto haré todas las tareas.

- División del Sprint Backlog en tareas más sencillas y abordables (Backlog items).
- Diseño de cada una de las tareas del punto anterior.
- Implementación de las tareas.
- Pruebas.

5.2. Estimación de tiempos y costes

A continuación en la tabla 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 materias 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. Para calcular los costes de los recursos humanos, se han usados los salarios medios de cada uno de los roles en la provincia de A Coruña y no están incluidos beneficios, exclusivamente los costes.

La jornada laboral sería de 5 horas y la duración de cada Sprint 20 días, lo que nos hace obtener el siguiente coste:

Rol	Coste por hora (€/h)	Horas	Total
Director del proyecto	40.00 €	25 h	1,000.00 €
Analista/Programador	19.00 €	800 h	15,200.00 €
		825 h	16,200.00 €

Figura 5.1: Costes asociados a este proyecto

- En el rol de director de proyecto estarían los que supervisan y evalúan el proyecto. En este caso los directores realizarán este rol.
- Analista/programador, 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 mi yo me encargaría de este rol.

5.3. Seguimiento

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 que estimamos sería de 8.

Antes de empezar con los Sprint comenzamos creando el Product Backlog. Ésto 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 centraremos en las tareas relacionadas con la capa intermedia. Para ello comenzamos diseñando la base de datos y modelo de datos, algo fundamental en cualquier aplicación para no heredar carencias en capas superiores. Posteriormente los servicios necesarios para el servicio REST.

5.3.2. Sprint 2

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

5.3.3. Sprint 3

En este Sprint se comenzará con el desarrollo de la aplicación Android. Comenzaremos permitiendo que el usuario sea capaz de 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 nos proporciona los datos necesarios para guardar puntos y rutas. Los casos de uso realizados son los siguientes:

- *R-PDI-1 Guardar Punto De Interés caza*
- *R-PDI-2 Guardar Punto De Interés pesca*
- *R-PDI-3 Eliminar PDI*

- *R-PDI-4 Buscar los PDI*

5.3.4. Sprint 4

En este Sprint nos centraremos en el registro de usuarios y gestión parcial de grupos.

- *R1 Registrarse en la aplicación.*
- *R2 Iniciar sesión en la aplicación.*
- *R-G-1 Crear grupo*
- *R-G-2 Añadir integrantes*

5.3.5. Sprint 5

Este Sprint se centrará en acabar con la gestión de grupos.

- *R-G-3 Eliminar integrantes,*
- *R-G-4 Ver grupos.*
- *R-G-5 Ver integrantes grupo*

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

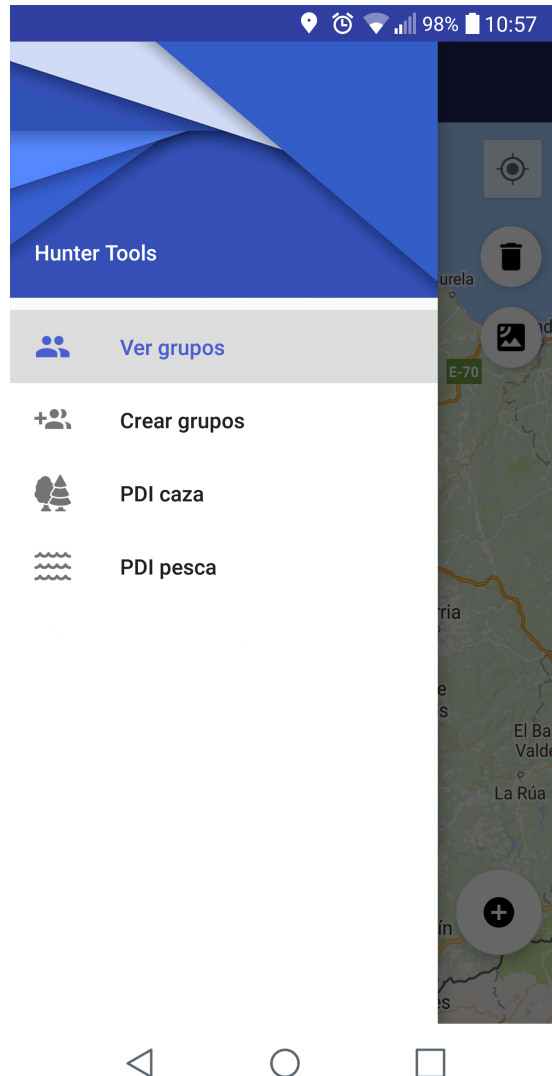


Figura 5.2: Captura del menú de navegación

5.3.6. Sprint 6

Este Sprint se centrará en ruta individuales.

- *R-R-1 Crear ruta*
- *R-R-1.1 Iniciar ruta*

- *R-R-1.2 Parar ruta*
- *R-R-1.3 Guardar ruta*
- *R-R-3 Listar rutas*
- *R-R-4 Ver ruta en mapa*
- *R-R-5 Eliminar ruta*

5.3.7. Sprint 7

Este Sprint contiene la funcionalidad más compleja de todas, gestionar una ruta compartida.

- *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 ultimo en este Sprint se realizaran 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 da aplicación móvil.

6.1. Esquema general de la arquitectura

La aplicación que estamos desarrollando seguirá la arquitectura en tres capas. 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 esta compuesto por:

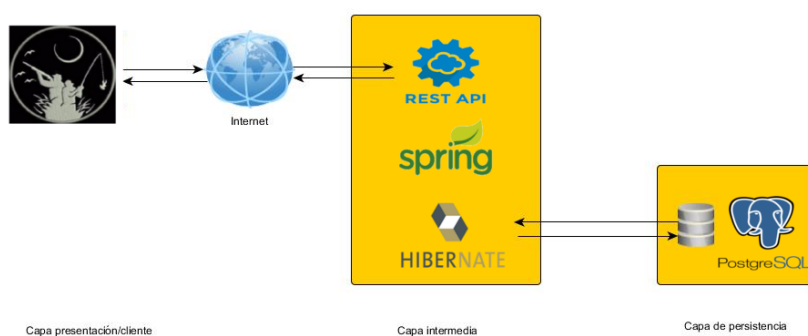


Figura 6.1: Esquema general de la arquitectura del sistema

- **Capa de presentación/cliente:**

Capa que nos proporciona la información relacionada con los servicios que puede invocar el cliente. Es la encargada de comunicarse con las otras capas para guardar la información de cada usuario. En si es la capa con la que va a interactuar 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 presentación/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 sistemas en la base de datos, 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.

Un 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). Ésta es una de esas ocasiones.

Los servidores que siguen el modelo 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 colocar 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 persistentes.
- **Vista:** Esta 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.

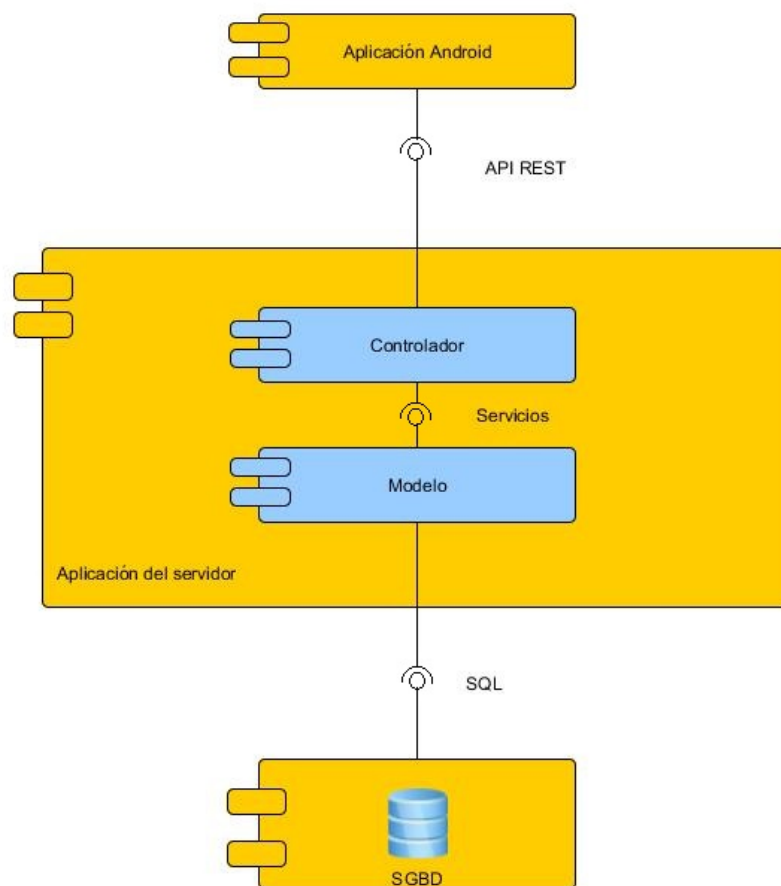


Figura 6.2: Diagrama de componentes del sistema

6.2. Modelo de datos

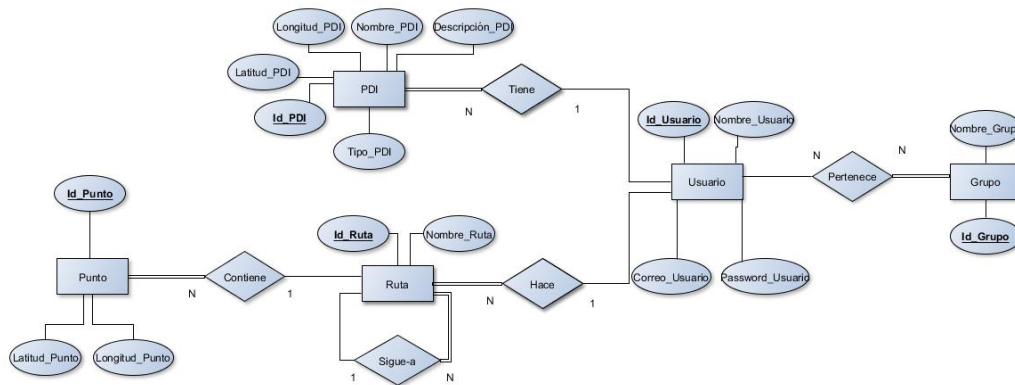


Figura 6.3: Entidad relación

6.3. Servidor

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

La solución mencionada anteriormente será un servidor, en java, que usará Spring 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, ya que ofrece distintos lugares donde poder ubicar lo hemos decidido hacerlo en uno concreto de Alemania para que el ping fuera más cercano y rápido.

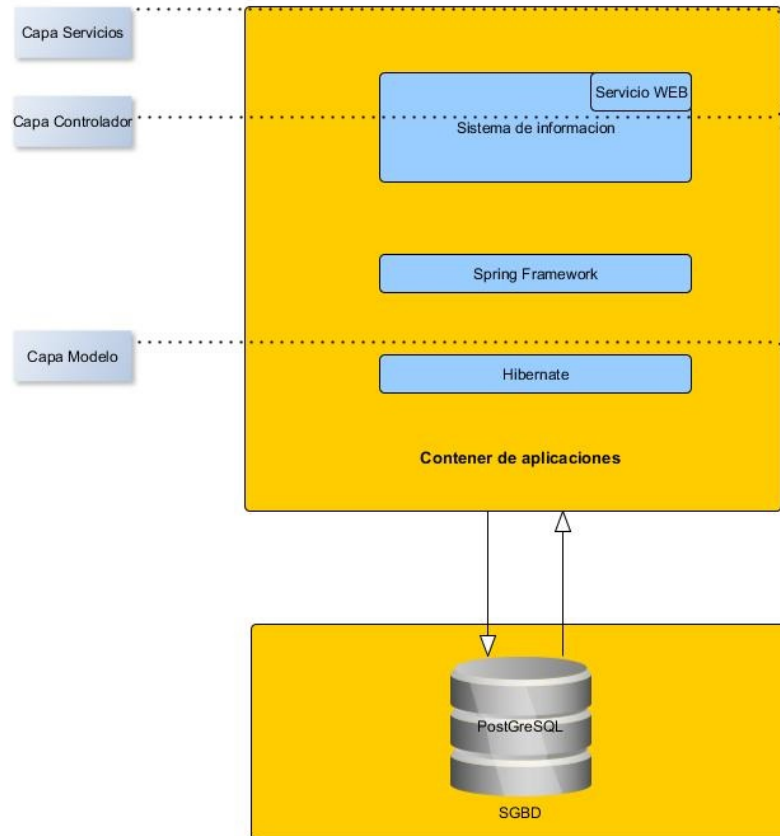


Figura 6.4: Arquitectura del servidor

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

6.3.2. Organización dos paquetes

A 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 (JpaRepository) que acceden a los datos del sistema.
- **es.udc.fi.dc.controller** Contiene as 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.

```
1 {  
2   "idusuario": 2,  
3   "nombre": "alberto",  
4   "correo": "alberto@dc.es",  
5   "token": "6QUdL9K4zUY:APA91bER1EBP6mufqQdTS5pneDen8-crxi6fG06evwl  
6   VP3kPfg6623k-PpRr-1THuWly3D51w-cr3u52Gn50ngROT@mbu2Sm1qlQcARezb4C0103XupdR7yHnSKZzFTus"  
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 implementacion 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 implementacion 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 seria 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.

```

16 public interface UsuarioRepository extends JpaRepository<Usuario, Long> {
17
18     public Usuario findByCorreo(String correo);
19
20     public Usuario findByNombre(String nombre);
21
22     public List<Usuario> findByNombreContaining(String nombre);
23
24     @Override
25     public <S extends Usuario> S save(S usuario);
26
27     @Override
28     public void delete(Long idUsuario);
29
30     @Override
31     public boolean exists(Long idUsuario);
32
33     @Override
34     public <S extends Usuario> S saveAndFlush(S usuario);
35
36     @Override
37     public Usuario findOne(Long idUsuario);

```

Figura 6.6: Ejemplo de interfaz con 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

- **Google Maps API.** Es un servicio web proporcionado por google que nos permite usar los mapas en nuestras aplicaciones y también conocer la ubicación del usuario en todo momento. Esta ubicación la marca con un punto azul pero no suministra coordenadas(latitud y longitud).
- **Google Location.** Es un servicio de google que nos suministra las coordenadas del usuario, según varia la posición ciertos metros o cada cierto tiempo.
- **Rest SystemService.** Este punto seria el que ejecuta nuestro propio servidor para guardar o devolver lo datos de los usuario, puntos de interés, rutas o grupos del usuario.
- **Firestore Cloud Messaging** Firestore 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.

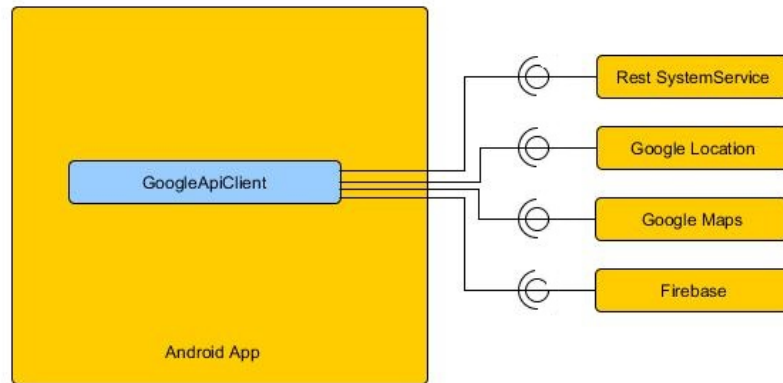


Figura 6.7: Servicios de la aplicación móvil

6.4.2. Organización de los paquetes

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

- **com.example.alberto.hunter-android.activities**, contiene todas las actividades.
- **com.example.alberto.hunter-android.adapter**, contiene los adaptadores.
- **com.example.alberto.hunter-android.model**, contiene todos los modelos de datos.
- **com.example.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 tecnología usada tanto en el servidor como en la aplicación móvil. También se comentarán las pruebas realizadas.

7.1.1. Servidor

- Engadir funcionalidades a medida nos repositorios de Spring Data

7.1.2. Aplicación móvil Android

- Mapas

-
-
-

7.2. Pruebas

7.2.1. Pruebas de unidad

7.2.2. Pruebas de unidad y de integración

Capítulo 8

Conclusiones y trabajo futuro

fsfsfsf

8.1. Investigación realizada

-
-

8.2. Características del proyecto

El uso de los paradigmas adecuados junto con herramientas consolidadas nos ha permitido construir una plataforma en la que servir una aplicación web donde los usuarios puedan buscar, valorar y obtener recomendaciones de contenidos audiovisuales.

A continuación se muestran las principales características del producto construido:

-
-

8.3. Trabajo futuro

Una vez finalizado este proyecto, se abren nuevas vías de trabajo, tanto de explotación comercial como de investigación científica:

-
-
-

Índice de Tablas

Índice de Figuras

2.1. Entorno de trabajo Eclipse	10
2.2. Dependencia de Spring en el pom.xml	10
2.3. Ejemplo de inyección de dependencia	11
2.4. Captura de pantalla realizando app debug	12
2.5. Emulador de Android con mi aplicación	13
2.6. Entorno de trabajo Android Studio	14
3.1. Metodología Scrum	19
3.2. Parte metodología Scrum	20
3.3. Parte metodología Scrum	20
3.4. Parte metodología Scrum	21
3.5. Parte metodología Scrum	21
4.1. Casos de uso del actor Usuario No Autenticado	25
4.2. Casos de uso de gestión de puntos de interés	26
4.3. Casos de uso de gestión de grupos de usuarios	27
4.4. Casos de uso de gestión de rutas	28
4.5. Iniciar sesión	29
4.6. Registrar usuario	29
4.7. Crear grupo	30
4.8. Añadir usuarios a grupo	30
4.9. Listar grupos	31
4.10. Opciones generales	31

4.11. Crear PDI	32
4.12. Visualizar PDIs	32
4.13. Ruta individual	33
4.14. Ruta compartida	33
5.1. Costes asociados a este proyecto	36
5.2. Captura del menú de navegación	39
6.1. Esquema general de la arquitectura del sistema	41
6.2. Diagrama de componentes del sistema	43
6.3. Entidad relación	44
6.4. Arquitectura del servidor	45
6.5. JSON	47
6.6. Ejemplo de interfaz con JpaRepository	48
6.7. Servicios de la aplicación móvil	49

Apéndice A

Glosario

ACID Atomicity, Consistency, Isolation and Durability: es un conjunto de propiedades que garantizan la fiabilidad de las transacciones de una base de datos. La atomicidad implica que cada transacción se ejecuta de forma completa o no se ejecuta. La consistencia garantiza que cada transacción llevará la base de datos de un estado consistente a otro consistente. El aislamiento permite que parezca que las transacciones se ejecutan de forma serializada cuando en realidad existe concurrencia. Por último, la durabilidad garantiza que una vez finalizada con éxito una transacción sus cambios se guardarán de forma persistente.

AM Agile Modeling: una metodología complementaria para modelar y documentar sistemas software mediante una serie de buenas prácticas basadas en el desarrollo ágil.

Big Data Es el término que se utiliza para llamar a los sistemas que procesan grandes colecciones de datos que son inmanejables bajo las herramientas tradicionales.

CF Collaborative Filtering: los algoritmos de filtrado colaborativo son una familia de algoritmos de recomendación cuyo funcionamiento se basa en generar sugerencias basándose en las preferencias de otros usuarios.

HDFS Hadoop Distributed File System: es un sistema de fichero distribuido, escalable y portable escrito en java y diseñado para servir de forma de almacenamiento a Hadoop.

IR Information Retrieval: la recuperación de información es la ciencia que estudia la obtención de información relevante a partir de una colección de documentos.

-
- ML *Machine Learning*: el aprendizaje automático es una rama de la inteligencia artificial que busca construir y estudiar sistemas que sean capaces de aprender de los datos.
- 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.
- NoSQL *Not Only SQL*: clase de sistemas de gestión de bases de datos que difieren del modelo relacional y cuyo principal objetivo es proporcionar escalabilidad horizontal.
- ORM *Object-Relational Mapping*: El mapeo objeto-relacional es una herramienta de programación que establece una correspondencia entre clases y objetos de un lenguaje de programación orientado a objetos con tablas y filas de una base de datos relacional siguiendo el patrón arquitectónico *active record*.
- RDBMS *Relational DataBase Management System*: un sistema de gestión de base de datos relacional es un software diseñado para gestionar la creación, modificación y consulta de bases de datos bajo el esquema relacional.
- RecSys *Recommender Systems*: los sistemas de recomendación tienen como objetivo predecir la preferencia de un usuario hacia un producto.
- RM *Relevance Models*: forma abreviada empleada en esta memoria referirse a los *Relevance-based Language Models*, una técnica de IR que introduce el concepto de relevancia en los modelos de lenguaje estadísticos.
- RPC *Remote Procedure Call*: es un tipo de comunicación entre procesos que permite la ejecución de una rutina en un equipo remoto.
- RUP *Rational Unified Process*: se trata de la propuesta de IBM para implementar una metodología de desarrollo basada en el marco que establece el Proceso Unificado.
- SQALE *Software Quality Assessment based on Lifecycle Expectations*: es un método genérico para evaluar la calidad de un código mediante unos índices y unos indicadores.

CAPÍTULO A.

Glosario

WSGI *Web Server Gateway Interface*: es una interfaz estándar de Python para la comunicación entre servidores y aplicaciones web.

Apéndice B

Bibliografía