



Universidade  
de Vigo

**Escola Superior de Enxeñaría Informática**

Memoria do Traballo de Fin de Grao que presenta

**D. Marcos Galiñanes Rivas**

para a obtención do Título de Graduado en Enxeñaría Informática

**Polen Alert: Sistema de monitorización de individuos y visualización de polen**



Xuño, 2015

**Traballo de Fin de Grao N°:** EI 14/15 – 022

**Titor/a:** Juan Carlos González Moreno

**Área de coñecemento:** Linguaxes e Sistemas Informáticos

**Departamento:** Informática



# ***Índice de Contenidos***

Índice de Contenidos.....	3
1. 1. Identificación del proyecto.....	5
1.2. Organización de la documentación.....	6
1.3. Introducción.....	7
1.4 Objetivos.....	9
1.5 Proceso de desarrollo.....	11
1.5 Descripción técnica.....	13
1.5.1 Android Studio.....	13
1.5.2 Java JDK.....	13
1.5.3 OpenOffice Writer.....	14
1.5.4 Visual Paradigm.....	14
1.5.5 GitHub.....	14
1.5.6 Microsoft Project 2007.....	14
1.6 Sobre Android.....	15
1.7 Planificación.....	18
1.7.2 Desarrollo real de la planificación.....	20
1.7.3 Justificación de los desvíos.....	21
1.8 Presupuesto.....	22
1.9 Conclusiones.....	24
1.9.1 Sobre las tecnologías.....	24
1.9.1 Sobre las tecnologías empleadas.....	24
1.9.2 Sobre la planificación.....	25
1.9.3 Conclusiones personales.....	25
1.10 Referencias y Bibliografía.....	26
Manual Técnico.....	29
2.1 Descripción del sistema.....	30
2.2 Especificación de requisitos.....	31
2.2.1 Actores.....	31
2.2.2 Metas.....	31
2.2.3 Funcionalidades.....	32
2.2.4 Atributos del sistema.....	33
2.2.5 Contexto de la aplicación.....	33
2.3 Historias de usuario.....	35
2.3.1 Diagrama de casos de uso.....	36
.....	37
2.3.2 Descripción de los casos de uso.....	37
2.4 Análisis de datos.....	43
2.4.1 Diagrama Entidad-Relación.....	43
2.4.2 Diccionario de datos.....	44
2.5 Desarrollo de aplicaciones en Android.....	45
2.5.1 MVC.....	46
2.5.2 Clases en Android.....	46
2.5.2.1 Clases generales.....	47
2.5.2.2 Modelos.....	47
2.5.2.3 Controladores.....	48
2.5.2.4 Vistas y archivos de configuración.....	48
2.5.3 Estructura de una aplicación en Android.....	48

.....	50
.....	50
.....	50
.....	50
2.6 Clases de la aplicación.....	50
2.6.1 Controladores.....	50
2.6.2 Componentes.....	52
2.6.3 Modelos.....	52
2.6.4 Librerías externas.....	53
.....	53
.....	53
.....	53
.....	53
.....	53
.....	53
.....	53
.....	53
2.7 Diagramas de secuencia.....	53
2.7.1 Observaciones sobre los diagramas.....	53
2.7.2 Diagrama de secuencia “Crear recorrido”.....	53
2.7.2 Diagrama de secuencia “Listar recorridos”.....	54
2.7.2 Diagrama de secuencia “Listar recorridos”.....	54
2.7.3 Diagrama de secuencia “Consultar datos recorrido”.....	55
2.7.4 Diagrama de secuencia “Eliminar recorrido”.....	55
2.8 Diagrama de componentes.....	57

## **1. 1. Identificación del proyecto**

**Título:** Polen Alert: Sistema de monitorización de individuos y visualización de polen

**Código:** El 14/15 – 022

**Autor:** Marcos Galiñanes Rivas

**DNI:** 53612464-Q

**Tutor:** Juan Carlos González Moreno

**Área:** Lenguajes y Sistemas Informáticos

**Departamento:** Informática

**Fecha:** Mayo, 2015

## 1.2. Organización de la documentación

La documentación del proyecto se encuentra organizada en un único volumen compuesto por los siguientes capítulos:

- **Memoria:** es una agenda de todo el proyecto que contiene una introducción al mismo, indicando los objetivos y soluciones, la planificación temporal y presupuesto, el ciclo de vida a lo largo del desarrollo y las conclusiones.
- **Manual técnico:** contiene el análisis, diseño y pruebas de la aplicación desarrollada. Se incluyen los diagramas de historias de usuario, de clases, de secuencia, etc. El objetivo es detallar la aplicación desde el punto de vista del desarrollo, facilitando su comprensión.
- **Manual de usuario:** contiene las instrucciones para que el usuario final pueda utilizar todas las funcionalidades de la aplicación.

Se incluye también un CD con el siguiente contenido:

- **Código:** contiene el código fuente de la aplicación.
- **Documentación:** contiene toda la documentación del proyecto en formato .pdf.
- **Software:** contiene el conjunto de aplicaciones necesarias para la instalación de la aplicación y su ejecución en un entorno local. En este caso será el archivo .apk de la aplicación y el fichero con los datos sobre el polen. Para la instalación es necesario permitir en el dispositivo la instalación de aplicaciones de orígenes desconocidos y copiar el archivo de polen en la carpeta descargas del dispositivo.

## 1.3. Introducción

El uso del teléfono móvil ha cambiado desde que se creó en 1979 el primer radio-telefono móvil en Japón. Diseñado como un dispositivo para comunicar por voz a dos personas de manera inalámbrica, ha evolucionado hasta ser prácticamente un ordenador portátil. Cámara, GPS, Internet, actualmente un teléfono móvil es un conglomerado de aplicaciones y dispositivos que permiten al usuario no solo comunicarse con otras personas, si no también hacer uso de herramientas que de otra manera debería llevar encima.

Uno de esas aplicaciones, es el sistema de geoposicionamiento global, conocido habitualmente como GPS, que es un sistema que permite localizar la posición en la Tierra de un objeto, persona o vehículo con cierto grado de error (centímetros o metros). Dicho sistema fue originalmente creado por el Departamento de Defensa de los Estados Unidos en los años 60 como uso primeramente militar, y en unos pocos años comercial(1). Desde ese entonces, su uso en distintos aspectos de la vida cotidiana se ha incrementado y se ha convertido en una de las funcionalidades básicas incluidas en los teléfonos y otros dispositivos inteligentes de hoy en día.

Una de las actividades que nos posibilita la localización por GPS es la monitorización de individuos. La monitorización de un individuo consiste en el análisis y seguimiento de dicho individuo sobre un ámbito de su vida concreto, en este caso su posición geográfica. Dicha monitorización se puede llevar a cabo con diferentes fines, siendo la vigilancia el más obvio y principal. Diferentes situaciones de una vigilancia geoposicional puede ser el hecho de controlar que los hijos de una persona realizan la ruta de casa al colegio y viceversa sin percances, que una persona con algún tipo de discapacidad se pierda en un su paseo matinal, o conocer si un vehículo con cargamento importante para la empresa sigue la ruta establecida en cuanto al tiempo programado.

En este contexto, surge la idea de la creación de una aplicación para recopilar puntos geográficos concretos a horas determinadas de un dispositivo para posteriormente realizar una monitorización de ese dispositivo sobre esos datos y así responder a las necesidades de los usuarios que requieren de la aplicación.

En este ámbito ya existen aplicaciones móviles que realicen una función similar. Una de ellas, por ejemplo, es Children Tracker. Mediante la instalación de esta aplicación en el dispositivo a monitorizar podemos observar la ubicación de ese dispositivo, así como si está conectado a internet o está realizando alguna llamada (2).

Otra conjunto de aplicaciones sobre monitorización por GPS son las llamadas

running apps. Con estas aplicaciones se pueden grabar el recorrido que se va haciendo mientras se sale a correr mientras nos van proporcionando datos sobre la velocidad, ritmo, tiempo transcurrido y otra información útil para los corredores. Actualmente, y con la puesta de moda de esta práctica, se han popularizado estas aplicaciones, siendo RunKeeper quizás la más famosa en estos momentos (3).

Otra aplicación para la monitorización por GPS es TrackMyRoute, que permite efectuar memorizaciones de recorridos para luego calcular la distancia recorrida y en base a ello, un precio de gasto según el medio de transporte. El objetivo es poder calcular las dietas por desplazamiento que irán incluidas en el sueldo de un trabajador. Toda la documentación incluida reside en la biblioteca de la ESEI (4).

En el área de la alergología está disponible Alergo Alarm, una aplicación que ofrece información diaria sobre los niveles de polen en una localidad, así como la predicción de los mismos para evaluar el posible riesgo de alergias (5).

Dentro de este ámbito y como último ejemplo, existe otra aplicación llamada Polen Control, que proporciona al usuario una visión de la cantidad aproximada de polen a lo largo de los días y según un informe de alergia previamente creado mediante preguntas al usuario, de modo que dicho informe es personalizado (6).

Por ello se propone Polen Alert, una aplicación para dispositivos móviles Android sencilla e intuitiva. Esta aplicación incorpora las funcionalidades necesarias para la recolección y vigilancia de recorridos de un dispositivo así como su fácil interacción y uso de la misma, facilitando a los usuarios de estos servicios la tarea de monitorización.

Además, como añadido, se ha pensado en la extensión e implementación de un caso práctico concreto, el de la monitorización de recorridos para personas alérgicas que proveerá a una persona alérgica al polen funcionalidades para evitar problemas de salud gracias a la monitorización de recorridos.



## 1.4 Objetivos

El objetivo principal de este proyecto es el desarrollo de una aplicación para la realización de recorridos según puntos geográficos en determinados momentos y su posterior monitorización.

Adicionalmente, se ha ofrecido la posibilidad por parte de un grupo de interesados de realizar un caso concreto de este tipo de monitorización, que consta de un módulo para la visualización de polen.

La aplicación base dispondrá de una opción para guardar los puntos geográficos concretos por donde transite el individuo y un modo seguimiento para monitorizar uno de esos recorridos previamente creados.

Por su parte, el módulo para la visualización del polen deberá proporcionar un mapa de la zona circundante a la posición del usuario mostrando la correspondiente concentración de polen a su alrededor.

Los objetivos concretos que debe cumplir la aplicación base son:

- Permitir crear recorridos de desplazamientos, en los que se guardará una serie de puntos asociados con fecha de paso, hora y lugar. Estos recorridos se identificarán con un nombre proporcionado por el usuario.
- Pausar y reanudar los recorridos mientras se están realizando.
- Crear estos recorridos manualmente o pasivamente.
- Listar los distintos desplazamientos creados y poder borrarlos si así se desea.
- La aplicación debe estar internacionalizada, permitiendo su uso en varios idiomas (inglés y español).
- Disponer de un modo “observador”, que consiste en comprobar si el desplazamiento actual coincide con el recorrido previamente guardado y seleccionado.

Por su parte, el módulo para la visualización de polen debe cumplir los siguientes objetivos:

- Visualizar en un mapa la posición del usuario y la concentración del polen a su alrededor.

## 1.5 Proceso de desarrollo

Teniendo en cuenta la inexperiencia en un proyecto de esta envergadura, en las tecnologías utilizadas y en el entorno de desarrollo se ha considerado usar una metodología de desarrollo ágil, permitiendo adaptarse a posibles cambios o problemas que surjan durante el desarrollo.

El proyecto se va a llevar a cabo siguiendo la metodología ágil mobile-D, una metodología diseñada específicamente para el desarrollo de aplicaciones móviles de una manera rápida y sencilla (7).

Una de las principales características de esta metodología es la posible adaptación de los requisitos cambiantes. En este proyecto los requisitos estaban inicialmente definidos pero se adaptaron debido a la extensión de la visualización del polen. Así mismo, el conocimiento que se ha ido adquiriendo durante el desarrollo ha evitado realizar tareas innecesarias y agilizar otras.

El ciclo de desarrollo en mobile-D consta de 5 fases: explorar, iniciar, producir, estabilizar y reparación del sistema (8):

- **Exploración:** consiste en la planificación y definición de los conceptos básicos del proyecto (requisitos iniciales y planificación inicial).
- **Inicialización:** se preparan e identifican todos los recursos necesarios. Se establece el entorno técnico y se identifican las posibles situaciones críticas.
- **Producción:** implementación de requisitos, funcionalidades e historias de usuario usando un ciclo de desarrollo incremental e iterativo.
- **Estabilización:** se llevan a cabo las acciones de integración para asegurar que el sistema completo funciona correctamente.
- **Pruebas y reparación:** tiene como meta la generación de una versión estable. Se revisa si la aplicación ha sido desarrollada de acuerdo con los requisitos del cliente y si contiene todas las funcionalidades requeridas.

En caso de añadir más funcionalidades a las ya definidas de manera inicial, se empezaría otra vez por la fase de exploración para identificar y planificar las siguientes funcionalidades a implementar.

La siguiente figura muestra el ciclo de desarrollo según esta metodología:

Ciclo de desarrollo de Mobile-D

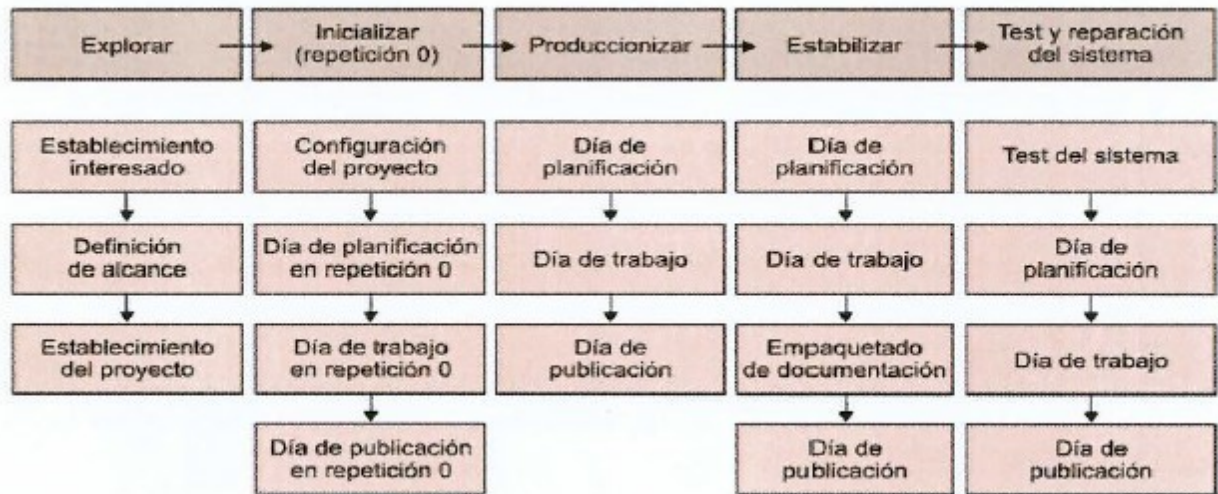


Figura 1: Ciclo de desarrollo de Mobile-D

## **1.5 Descripción técnica**

A continuación se detallan las tecnologías y herramientas necesarias para el desarrollo de este proyecto.

Todo el proyecto se desarrollará en el sistema operativo Windows 7 SP1.

### **1.5.1 Android Studio**

Esta aplicación se desarrollará utilizando Android Studio. Android Studio es un entorno de desarrollo integrado (IDE) de código abierto y multiplataforma. Concretamente se usará la versión 1.0.1, que es la más reciente en el momento en que se inicia el desarrollo de este proyecto y aunque en el instante de entrega de esta documentación se encuentra en la versión 1.0.3 se ha optado por mantenerse en la versión mencionada para evitar problemas de estabilidad o compatibilidades entre versiones (9).

Además, Android Studio también incluye Android SDK tools que facilita enormemente el desarrollo de aplicaciones para Android, junto con APIs integradas para el uso de diferentes módulos de los dispositivos Android.

Otra de las razones por las que se ha escogido este IDE es la facilidad de instalación, ya que trae consigo todas las herramientas ya configuradas y autodetecta en el sistema la versión más reciente de JDK instalado.

### **1.5.2 Java JDK**

El kit de desarrollo java contiene todo lo necesario para poder programar aplicaciones en este lenguaje de desarrollo, que es uno de los más extendidos actualmente. Dado que Android utiliza Java para desarrollar sus aplicaciones, es necesario instalar el JDK y enlazarlo con el IDE que se va a emplear.

En el momento del inicio del proyecto, se ha instalado la versión 1.8.0, que sigue siendo la última versión.

### **1.5.3 OpenOffice Writer**

Para realizar toda la documentación del proyecto se usará un procesador de textos de alto nivel y totalmente gratuito como es OpenOffice Writer en su versión 4.0.1, actualmente la última versión disponible.

### **1.5.4 Visual Paradigm**

Visual Paradigm es un programa para el diseño de Software usando UML. Se ha utilizado este programa en su versión 11.2 para todos los diagramas necesarios incluidos en el Manual Técnico.

El motivo de la elección de dicha versión es la disponibilidad de la licencia. Visual Paradigm usa licencias de pago o para estudiar para su uso, y de forma que la universidad ha proporcionado a los alumnos licencias para la versión 11.2, esa ha sido la versión escogida.

### **1.5.5 GitHub**

Git es un software de control de versiones que permite almacenar archivos en la nube, compartirlos y poder realizar cambios en ellos subiendo solo los cambios que se han modificado. Se ha utilizado GitHub para Windows en la versión 2.1.4.3 en las últimas semanas del proyecto para poder tener acceso al código y efectuar la validación del mismo.

### **1.5.6 Microsoft Project 2007**

Microsoft Project es un software que permite la planificación de un proyecto y obtener distintos diagramas como puede ser el diagrama de Gantt. Se ha pensado en esta aplicación para realizar toda la planificación, tanto la inicial como la resultante, y los diagramas de Gantt del desarrollo.

## 1.6 Sobre Android

A continuación se explicará en qué consiste el sistema Android y se detallará su estructura para poder comprender mejor todas las tecnologías que se usarán en este proyecto (10).

Esta es la arquitectura básica de Android:

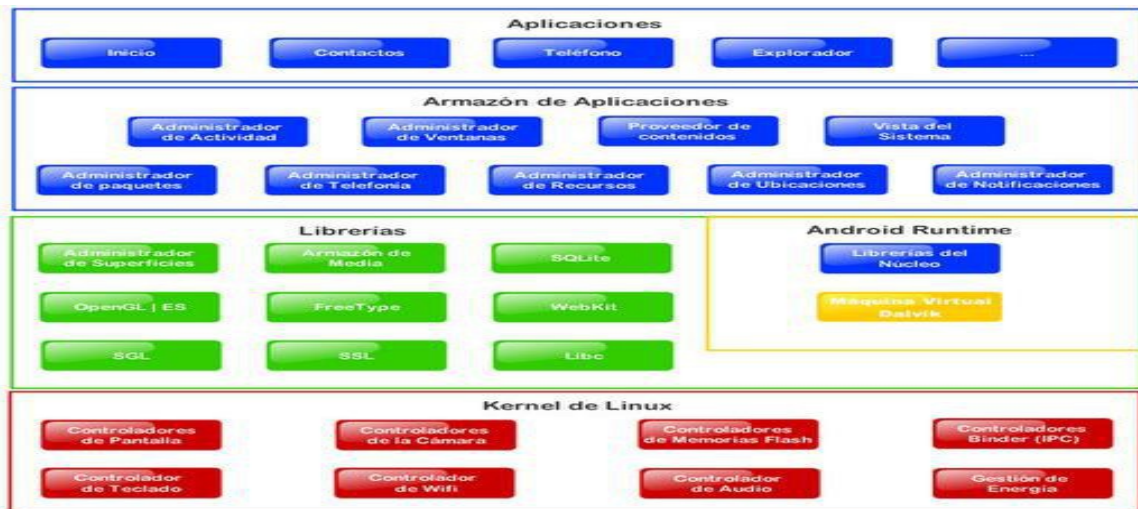


Figura 2: Arquitectura Sistema Android

Como se puede observar la arquitectura está dividida en 4 capas: aplicaciones, armazón de aplicaciones, librerías y el kernel de Linux.

- **Aplicaciones:** son las aplicaciones base que puede tener todo sistema operativo para terminales móviles. Incluyen clientes de email, programa de SMS, calendario, mapas, navegadores, contactos, GPS, juegos, etc . La mayor parte de estas aplicaciones están escritas en lenguaje Java. Aquí es donde se encuentra la aplicación que se desarrollará en este proyecto.
- **Armazón de aplicaciones:** módulos escritos en Java que permiten interactuar con las aplicaciones de la capa superior. Los desarrolladores tienen acceso a todos los frameworks usados por las aplicaciones base. Esto favorece la reutilización de componentes.
- **Librerías:** Están divididas en dos subcapas:
  - **Librerías:** estas se encargan de las funcionalidades básicas. Están escritas en C/C++, pueden ser accedidas directamente desde el armazón de aplicaciones de Android. Las hay de varios tipos, como pueden ser librerías

en C del sistema, multimedia, gestor de superficies, librería del núcleo web, SQLite, etc.

- Runtime de Android: Android incluye un grupo de librerías base las cuales proporcionan muchas de las funciones disponibles en las librerías base del lenguaje Java. Cada aplicación Android corre su propio proceso en la máquina virtual Dalvik. La máquina Dalvik ha sido optimizada para que un dispositivo pueda correr múltiples máquinas virtuales de forma eficiente.
- Kernel de Linux: el núcleo de Android es Linux. Algunos servicios base del sistema dependen de Linux. La versión 2.1 de Android toma como núcleo la versión 2.6.29 de Linux. También actúa como capa de abstracción entre el hardware y el resto del software.

El esqueleto básico de una aplicación Android está formado por los siguientes directorios:

- Directorio Bin: aquí se guardan los archivos binarios compilados de la aplicación.
- Directorio Src: aquí se encuentra todo el código Java de la aplicación.
- AndroidManifest.xml: archivo requerido para cada aplicación. Describe los valores globales de su paquete, incluida la aplicación, componentes, clases para cada uno de los componentes, qué tipos de datos puede manejar cada uno, y donde puede ser lanzado.
- Directorio Res: los recursos que usa la aplicación, ordenados en tres carpetas:
  - Drawable: ficheros de imágenes.
  - Layout: ficheros de diseño de las interfaces de usuario. Son archivos .xml que describen los elementos gráficos de las interfaces.
  - Values: definición de variables y constantes.

Otro aspecto a tener en cuenta de Android es que es compatible con cualquier tipo de transmisión; esto posibilita que funcione tanto con dispositivos que usen redes GPRS como con dispositivos 3G.



Para la construcción de una base de datos adaptable al sistema operativo Android se usará SQLite. Éste es un sistema de gestión de bases de datos relacional, contenida en una pequeña librería de C.

A diferencia de la mayoría de sistemas de gestión de bases de datos cliente-servidor, el motor de SQLite no es un proceso independiente que se conecta con el programa principal, si no que enlaza su biblioteca con el programa pasando a ser parte integral del mismo.

Aunque existen más opciones para alojar los datos que necesitará la aplicación, como JSON o XML, se ha escogido esta opción dado que a lo largo de la carrera se ha trabajado mucho más con bases de datos del tipo SQL, además de que al ser un estándar de referencia posibilita la opción de ampliaciones por parte de desarrolladores externos en caso de ser necesario (11).

Para la parte de ubicación en el mapa se deberá usar el GPS y/o la conexión a Internet de cada dispositivo. Para ello se emplearán las APIs que el sistema operativo Android pone a disposición del desarrollador.

Una API o interfaz de programación de aplicaciones es el conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizada por otro software como una capa de abstracción. Es una interfaz de comunicación entre dos componentes software. Proporcionan funciones de uso general. Mediante su uso el programador se evita tener que desarrollar todas las operaciones que va a necesitar, fomentando la reutilización.

## 1.7 Planificación

La estimación inicial, realizada a 9 de marzo de 2015 contemplaba las siguientes tareas:

- Fase de exploración del sistema, entorno, requisitos, etc (2 días).
- Etapa de inicialización (15 días).
- Producción (31 días).
- Estabilización (14 días).
- Test y reparación del sistema (7 días).

En las figuras 3 y 4 se muestra de forma detallada la duración de las tareas planificadas, y su solapamiento durante el tiempo estimado de realización del proyecto.





















		Task Name	Duration	Start	Finish
1		Definicion de grupos de int	1 hr	Tue 17/03/15	Tue 17/03/15
2		Requisitos iniciales	3 hrs	Tue 17/03/15	Tue 17/03/15
3		Limitaciones	1 hr	Tue 17/03/15	Tue 17/03/15
4		Identificar recursos	1 hr	Wed 18/03/15	Wed 18/03/15
5		Identificar documentación	1 hr	Wed 18/03/15	Wed 18/03/15
6		Dependencias	1 hr	Wed 18/03/15	Wed 18/03/15
7		Planificación inicial	3 hrs	Thu 19/03/15	Thu 19/03/15
8		Configuración del entorno	5 hrs	Sat 21/03/15	Sun 22/03/15
9		Funcionalidad a implementi	4 hrs	Tue 24/03/15	Tue 24/03/15
10		Planificación inicial actualiz	5 hrs	Tue 24/03/15	Wed 25/03/15
11		Investigaciones tecnológic	8 hrs	Wed 25/03/15	Thu 26/03/15
12		Historias de usuario	5 hrs	Fri 27/03/15	Fri 27/03/15
13		Pruebas de aceptación	8 hrs	Fri 27/03/15	Sat 28/03/15
14		Diseño UI	22 hrs	Tue 31/03/15	Sat 04/04/15
15		Implementacion del sistema	95 hrs	Sun 05/04/15	Tue 05/05/15
16		Pruebas de funcionalidade	30 hrs	Sat 25/04/15	Tue 05/05/15
17		Integracion de subsistema	46 hrs	Tue 05/05/15	Tue 19/05/15
18		Test de integración.	36 hrs	Fri 08/05/15	Tue 19/05/15
19		Pruebas finales	25 hrs	Tue 19/05/15	Tue 26/05/15

Figura 3: Planificación inicial.

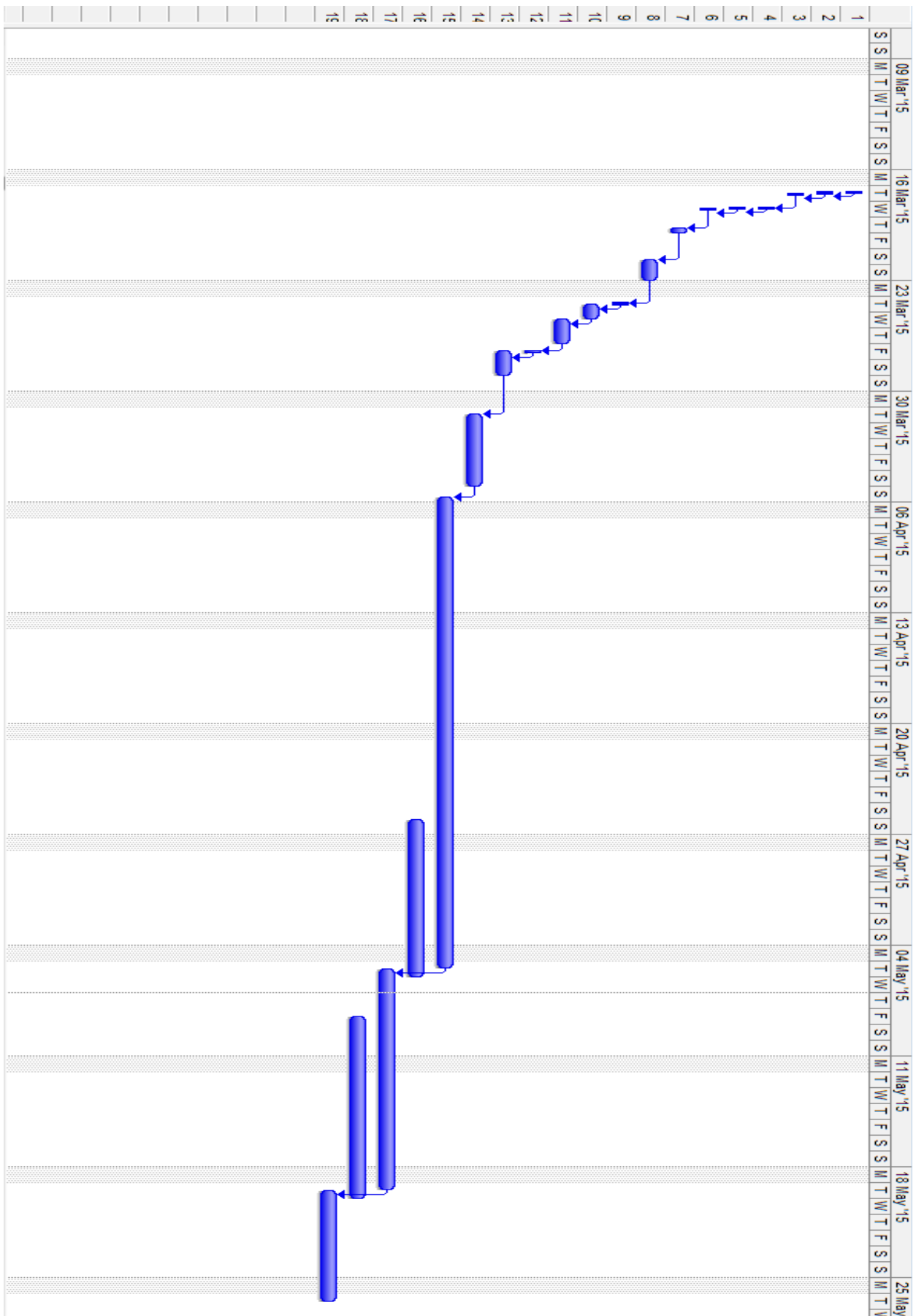






















Figura 4: Diagrama Gantt inicial.

Para la planificación inicial se consideró una carga de trabajo de 25 horas a la semana, distribuidas en 5 horas el martes, miércoles y viernes, 3 horas el jueves y 7 horas el fin de semana, resultando en un total de 300 horas.

### 1.7.2 Desarrollo real de la planificación

A pesar de la planificación, no ha sido posible mantener un ritmo constante de trabajo durante todo el periodo de desarrollo, por lo que se han producido desvíos en los plazos marcados.

Durante el desarrollo del proyecto, la duración final de las tareas realizadas ha sido la mostrada en la Figura 5.

		Task Name	Start	Finish
1		Definicion de grupos de interés	Tue 17/03/15	Tue 17/03/15
2		Requisitos iniciales	Wed 18/03/15	Thu 19/03/15
3		Limitaciones	Thu 19/03/15	Sat 21/03/15
4		Identificar recursos	Sat 21/03/15	Sun 22/03/15
5		Identificar documentación	Tue 24/03/15	Tue 24/03/15
6		Dependencias	Tue 24/03/15	Tue 24/03/15
7		Planificación inicial	Tue 24/03/15	Wed 25/03/15
8		Configuración del entorno de desarr	Thu 26/03/15	Fri 27/03/15
9		Funcionalidad a implementar	Fri 27/03/15	Sun 29/03/15
10		Planificación inicial actualizada	Tue 31/03/15	Wed 01/04/15
11		Investigaciones tecnológicas	Thu 02/04/15	Mon 06/04/15
12		Historias de usuario	Tue 07/04/15	Tue 07/04/15
13		Pruebas de aceptación	Wed 08/04/15	Sun 12/04/15
14		Diseño UI	Tue 14/04/15	Sun 19/04/15
15		Implementacion del sistema	Tue 21/04/15	Wed 10/06/15
16		Pruebas de funcionalidades	Wed 10/06/15	Sun 14/06/15
17		Integracion de subsistemas	Sun 14/06/15	Wed 17/06/15
18		Test de integración.	Wed 17/06/15	Thu 18/06/15
19		Pruebas finales	Thu 18/06/15	Fri 19/06/15

*Figura 5: Planificación final.*

La duración final del proyecto se ha incrementado en 24 días y se han empleado 275 horas. Sin embargo, aunque los días necesarios han aumentado, esto se debe a no haber podido trabajar las horas planificadas los días previstos, o encontrar problemas que dificultaran la finalización de una etapa.

### 1.7.3 Justificación de los desvíos

Como se puede observar en las secciones anteriores, el desarrollo del proyecto ha sufrido un importante retraso respecto a la planificación inicial. Los motivos de esta desviación se exponen a continuación:

- El estudio de las tecnologías y el lenguaje de programación se planificó con un período de tiempo pequeño, dado que ya se disponían ciertos conocimientos de Java.
- Una estimación demasiado optimista del tiempo de desarrollo e implementación, ya que el hecho de estar cursando a su vez 4º curso de la Ingeniería Informática supuso períodos de estudio para exámenes o trabajos a entregar no previstos.
- Una vez iniciado el desarrollo, se estudió y decidió añadir la funcionalidad para mostrar las concentraciones de polen alrededor del usuario en un mapa. Esto supuso una nueva toma de requisitos, estudio de tecnologías y más tiempo de implementación.

## 1.8 Presupuesto

Para la realización de este proyecto se han utilizado, en la medida de lo posible, herramientas gratuitas, ya sea por disponer de licencias de libre distribución o por la posibilidad de conseguir una licencia sin coste.

Para el cálculo de los presupuestos se ha tenido en cuenta una amortización de los recursos hardware a cinco años (60 meses), mientras que para los recursos software la amortización que se contempla es de dos años (24 meses).

En las tablas siguientes se detalla el coste de recursos hardware y software, así como los costes de recursos humanos, considerando la duración total del desarrollo (3 meses) para las amortizaciones.

Hardware		
	Coste	Amortización
PC		
Procesador Intel Core i3-350M		
Tarjeta gráfica NVIDIA GT 320M		
4GB de memoria RAM		
CD-DVD RW		
Disco Duro de 500 Gb		
	599€	30€
Móvil LG E-460	100 €	5 €
Total	699 €	35 €

*Figura 6: Presupuesto hardware.*

Software		
	Coste	Amortización
SO Windows 7 Professional 64bit (licencia ELMS)	0 €	
Microsoft Project 2007 (licencia ELMS)	0 €	
Visual Paradigm (licencia ELMS)	0 €	
GitHub	0 €	
Android Studio	0 €	
OpenOffice Writer	0 €	
Total	0 €	0 €

*Figura 7: Presupuesto software.*

Recursos humanos		
	Horas trabajadas	Coste / hora
Ingeniero de Software	275	18 €
Total	4.950 €	

*Figura 8: Recursos humanos.*

Presupuesto total		
	Coste	Amortización
Hardware	699 €	35 €
Software	0 €	0 €
Recursos humanos	4.950 €	0 €
Total	5.649 €	35 €

*Figura 9: Presupuesto total.*

De este modo, tal y como indica la Figura 9 el coste final del proyecto asciende a 5649€.

## 1.9 Conclusiones

### 1.9.1 Sobre las tecnologías empleadas

Android Studio ha sido un IDE sumamente útil para que un programador sin experiencia mejore sus aptitudes de desarrollo en Android, ya que ofrece muchas facilidades para su instalación, su uso y sobre todo en el informe de errores en líneas de código o de resultados de la compilación y ejecución de la aplicación.

Sus APIs integradas son de muy fácil acceso, siendo la importación de librerías y paquetes totalmente transparente para el usuario y facilitando así el desarrollo.

En cuanto a la documentación existente en Internet, aunque bien es cierto que muchas páginas de ayuda y foros van orientados a Eclipse, no se ha tenido ningún problema con encontrar soluciones que funcionasen en Android Studio.

Por lo tanto, se considera que la decisión de usar Android Studio como entorno de desarrollo ha sido muy acertada y de llevar a cabo otro proyecto de características similares, se tendría en cuenta este software como primera opción.

La aplicación resultante ha cumplido con los requisitos y funcionalidades previstos desde el inicio, siendo su interfaz sencilla e intuitiva y proporcionando al usuario una herramienta completamente funcional en cuanto a los objetivos principales de este proyecto.

El cálculo pasivo de recorridos se implementó de dos maneras diferentes: bien usando el dispositivo GPS del terminal o a través del servicio Google Services que ofrece Google, usando la red de Internet en lugar del GPS. Sin embargo, las actualizaciones constantes sobre Google Play Services, que obligaban a cambiar parte del código desarrollado, y la falta habitual de cobertura completa que sufren los dispositivos llevaron a la decisión de eliminar esta opción, siendo el cálculo de recorridos por GPS la mejor y más eficaz opción.

El acceso a la base de datos del polen, que se encuentra en formato .mdb (Microsoft Access), ocasionó un problema de acceso a dicho archivo, ya que por defecto Java no dispone de librerías para ello. Esto se solucionó exitosamente importando la librería “jackcess” (12), una librería para Java de uso libre diseñada para permitir el acceso y manipulación de archivos de Microsoft Access.



### **1.9.2 Sobre la planificación**

En cuanto a la planificación y los plazos sobre cada etapa en la planificación inicial, finalmente no han sido cumplidos. Las razones son varias, pero destacan la falta de experiencia en proyectos de este desarrollo y la mala planificación en cuanto a que este proyecto no es lo único que se ha desarrollado en estos meses, y se debería haber tenido en cuenta esta posibilidad desde un inicio.

Aún así y todo, se han superado los objetivos propuestos, con un leve retraso pero el tiempo empleado finalmente ha sido el estimado y los objetivos se han alcanzado con éxito.

### **1.9.3 Conclusiones personales**

La realización de este proyecto ha resultado satisfactoria e interesante en muchos aspectos. Quizá el más destacable de todos ellos ha sido enfrentarse al desarrollo de una aplicación de gran tamaño, con toda la planificación y análisis que ello requiere.

Me ha servido para aplicar los conocimientos sobre distintos lenguajes de programación y como funcionan internamente, sobre todo de Java, así como usar los conocimientos teóricos sobre gestión, planificación y documentación de proyectos y observar la utilidad de los mismos.

Desde luego la planificación ha sido incorrecta en muchas ocasiones, lo cual ha contribuido a dilatar el tiempo de desarrollo. También muchas veces el análisis inicial de los requisitos contenía errores o escenarios no esperados que obligaron a replantearse la manera de enfocar su implementación.

Sin embargo, termino con la sensación de haber adquirido una experiencia útil para afrontar nuevos desarrollos,. Además, el verme ante el desarrollo de un proyecto de tal envergadura y sin conocimientos técnicos sobre el mismo, me han permitido conocerme a mi mismo y saber hasta donde llegan mis conocimientos, mi capacidad de adaptarme a los errores o mejorar y conocer mis límites.

## 1.10 Referencias y Bibliografía

A continuación se detallan todas las fuentes de información y documentación que se ha consultado para la realización de este proyecto así como su documentación.

- 1. Pogge, Richard W: *“Real World Relativity: The GPS Navigation System”*  
25 de Enero de 2008.
- 2. SafeT.me: Web principal de Children Tracker.
- 3. androidcentral.com: Web sobre novedades software y hardware para dispositivos móviles con Android.
- 4. Identificación del proyecto: *El 13/14 – 025*.
- 5. polenes.com/alergoalarm.html o bien se encuentra en la App Store.
- 6. polencontrol.com: Web principal de la aplicación. También disponible en la Play Store.
- 7. Presentación expuesta en la conferencia de desarrolladores CASE25.  
Enlace: <http://www.slideshare.net/ZlatkoStapi/using-mobile-d-methodology-in-development-of-mobile-applicationspptx>
- 8. Abrahamsson, P., Hanhineva, A., y Hulkko, H: *“MobileD: An Agile Approach for Mobile Application Development”*.  
Canada, OOPSLA 2004.
- 9. developer.android.com: Web oficial del entorno de desarrollo Android Studio
- 10. Smieh: *“Arquitectura del sistema Android”*.  
29 de Junio de 2012.
- 11. sqlite.org: *“Distinctive Features of SQLite”*.  
3 de Marzo de 2008.
- 12. jackcess.sourceforge.net: Web oficial del proyecto Jackcess, librería Java para el acceso a ficheros en formato Microsoft Access.

- 13. Se usa el formato post-it que se ha visto en la asignatura Desarrollo de Aplicaciones Ágiles impartida por Miguel Reboiro Jato.
- 14. [omg.org/spec/UML](http://omg.org/spec/UML): “*UML Infrastructure Specification*”. 5 de Agosto de 2011.
- 15. Fabien Potencier: “*El Tutorial Jbeet*”, capítulo 4.
- 16. [developer.android.com](http://developer.android.com): Web principal de soporte para el desarrollo en Android con Java

# ***Manual Técnico***

## **2.1 Descripción del sistema**

Polen Alert es una aplicación móvil desarrollada para dispositivos Android. Con ella, un usuario puede crear recorridos (conjunto de puntos geográficos identificados en una hora concreta) a medida que se traslada, haciéndolo de manera pasiva o manual, para posteriormente controlar que un futuro recorrido se ajusta en cierta medida a uno de esos previamente creados.

Con esto se pretende que los usuarios puedan, sin tener que llevar una agenda o de memoria, controlar que los recorridos que suelen hacer no se salgan de ruta y cumplan con el trayecto acordado.

Adicionalmente, se ha desarrollado una extensión, que es un caso práctico de las diferentes utilidades que se le puede dar a esta aplicación, para visualizar el polen cercano. El objetivo es que a medida que el usuario se va trasladando pueda observar la cantidad de polen a su alrededor, representada en un color en función de la cantidad total, y actuar en consecuencia.

## 2.2 Especificación de requisitos

En esta sección se analizará en detalle el sistema que se va a desarrollar en este proyecto, identificando sus actores o grupos de interés, las metas que éste debe alcanzar y las funciones necesarias para hacerlo.

### 2.2.1 Actores

Dentro del sistema se identifican los siguientes actores:

- **Usuario:** representa al usuario normal a la cual va dirigida la aplicación. Usa la aplicación para crear los recorridos que considere necesarios, poder consultar sus datos y/o borrarlos y finalmente usar el modo observador para monitorizar uno de los recorridos previamente creados.
- **Usuario alérgico:** Este usuario está más enfocado a la consulta del polen que tiene a su alrededor y a la monitorización de su desplazamiento. Además, puede estar también interesado en las funcionalidades del usuario normal de la aplicación.

### 2.2.2 Metas

Las metas principales que debe cumplir el proyecto se dividen entre metas de la aplicación base y metas del módulo de visualización del polen.

Las metas a alcanzar por la aplicación base son:

- Ofrecer un sistema rápido y sencillo para poder crear recorridos o desplazamientos (que consiste en una serie de puntos geográficos en un momento concreto, identificados con un nombre). Estos recorridos se irán “grabando” de forma transparente mientras el usuario se desplaza o bien manualmente.
- Proporcionar la capacidad de poder visualizar una lista con los recorridos añadidos, junto con sus datos relacionados y, en caso de así desearlo, poder borrar alguno de ellos.

- Implementar lo necesario para la internacionalización de la aplicación. Esto es, poder cambiar de idioma de interfaz de forma simple.
- Ofrecer un sistema para la monitorización de recorridos, pudiendo, previamente seleccionado un recorrido, comprobar que el desplazamiento actual del individuo coincide con cierto margen de error con el recorrido seleccionado, avisando de ello en caso negativo.

Las metas a alcanzar por el módulo de visualización del polen son:

- Facilitar un mapa con el punto actual del usuario, y visualizar la concentración de polen que le rodea, actualizándose esta información según se desplace.

Como meta común a ambos, tanto la aplicación base como el módulo de visualización de polen deben ofrecer una interfaz de usuario lo más intuitiva y amigable posible para cada una de sus funcionalidades.

### 2.2.3 Funcionalidades

Para la consecución de las metas establecidas en la anterior sección de esta documentación el sistema deberá implementar las funcionalidades mostradas en las siguientes tablas.

Las funcionalidades son expresiones que reflejan de manera tácita lo que la aplicación debe poder hacer. Estas funcionalidades luego se expresarán en forma de historias de usuario como es común en las metodologías ágiles (13).

Sin embargo, y para la facilitación de la lectura y comprensión de este proyecto, dichas historias de usuario se asociarán a casos de uso (los mismos que las funcionalidades), los cuales se explicarán en detalle y permitirá la realización de diagramas de notación UML (14).

Grupo AB. Aplicación Base	
AB1	Crear recorrido pasivo
AB2	Crear recorrido manual
AB3	Consultar recorridos

AB4	Consultar datos
AB5	Eliminar recorrido
AB6	Monitorizar recorrido
AB7	Cambiar idioma
AB8	Escoger email predeterminado

*Figura 10. Funcionalidades Base.*

Grupo MP. Módulo de visualización de Polen	
MP1	Mostrar datos de polen
MP2	Cambiar tipo de polen

*Figura 11: Funcionalidades modelo de polen.*

## 2.2.4 Atributos del sistema

El sistema está pensado para ser utilizado por usuarios normales sin conocimientos de ningún campo en especial, sea cual sea la funcionalidad para que usen la aplicación de este proyecto. Por lo tanto, un atributo importante del sistema será disponer de una interfaz amigable que presente la información lo más clara posible, sin renunciar por ello a ninguna de sus funcionalidades.

En algunos casos se manejarán listados que podrán contener una gran cantidad de elementos. El sistema deberá presentar estos listados de forma manejable.

Por otra parte, es un atributo deseable ofrecer un tiempo de respuesta reducido para la navegación por los contenidos y apartados de la aplicación, si bien para algunas funciones es admisible un tiempo de respuesta más largo, como pueden ser las funciones relacionadas con el manejo de los datos sobre polen, ya que la base de datos proporcionada ofrece una gran cantidad de información a procesar.

## 2.2.5 Contexto de la aplicación

El sistema será una aplicación para dispositivos móviles, alojada en el propio dispositivo, a la que se podrá acceder mediante el menú de aplicaciones del mismo. Está desarrollada en el lenguaje de programación JAVA , almacenando la información



en una base de datos en SQLite, exceptuando el módulo de visualización de polen que extrae los datos de un archivo .mdb. Dicho archivo debe estar alojado en la carpeta “*Descargas*” del dispositivo.

## 2.3 Historias de usuario

En esta sección se listan las historias de usuario identificadas en el sistema junto con los casos de uso asociados a cada una de ellas. Posteriormente se detallará cada uno de ellos.

- **COMO** Usuario  
**QUIERO** poder crear un recorrido que almacene los puntos por donde me desplazo  
**PARA** posteriormente poder monitorizar dichos recorridos.

Casos de uso asociados: Crear recorrido pasivo (AB1), crear recorrido manual (AB2).

- **COMO** Usuario  
**QUIERO** consultar los recorridos creados  
**PARA** estar informado de los desplazamientos realizados.

Casos de uso asociados: consultar recorridos (AB3), consultar datos recorrido (AB4).

- **COMO** Usuario  
**QUIERO** eliminar un recorrido creado  
**PORQUE** ya no me interesa su existencia.

Caso de uso asociado: eliminar recorrido (AB5).

- **COMO** Usuario  
**QUIERO** monitorizar un recorrido  
**PARA** controlar que el desplazamiento es el correcto.

Caso de uso asociado: monitorizar recorrido (AB6).

- **COMO** Usuario  
**QUIERO** cambiar el idioma de la aplicación  
**PARA** estar más cómodo al usar la aplicación.

Caso de uso asociado: cambiar idioma (AB7).

- **COMO** Usuario  
**QUIERO** cambiar el email predeterminado  
**PARA** escoger el email destinatario del aviso de fuera de recorrido.

Caso de uso asociado: cambiar email (AB8).

- **COMO** Usuario alérgico  
**QUIERO** poder visualizar el polen a mi alrededor  
**PARA** estar informado de la concentración de polen y no tener problemas de salud.

Caso de uso asociado: mostrar datos de polen (MP1) y cambiar tipo de polen (MP2).

### 2.3.1 Diagrama de casos de uso

Como en este caso, el usuario alérgico puede querer tener acceso al resto de funcionalidades de la aplicación, se considerará un único actor en todo el sistema a partir de ahora llamado “usuario”.

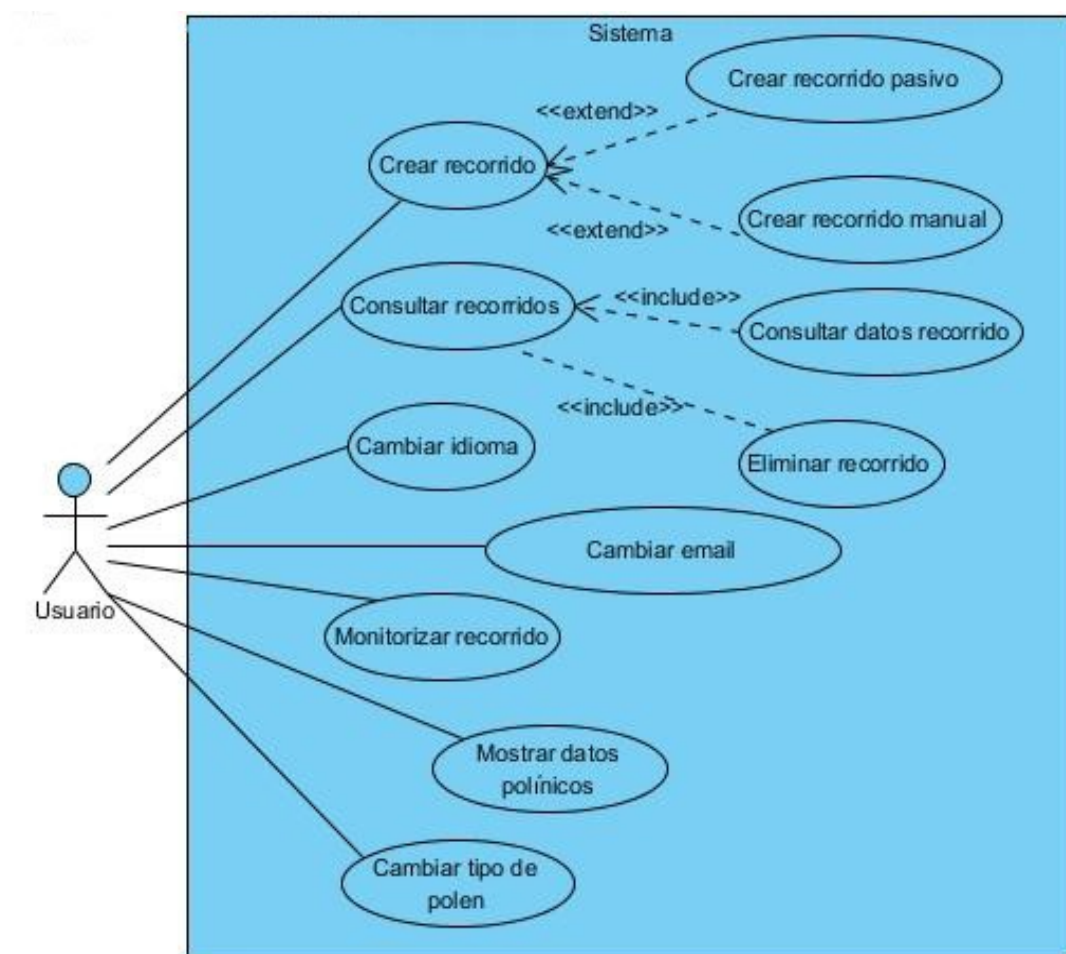


Figura 12: Diagrama de casos de uso.

## 2.3.2 Descripción de los casos de uso

### *Caso de uso "Crear recorrido pasivo"*

<b>NOMBRE</b>	<b>Crear recorrido pasivo</b>
<b>ACTORES</b>	Usuario
<b>OBJETIVOS</b>	El usuario crea un nuevo recorrido de forma automática
<b>PRECONDICIÓN</b>	
<b>POSTCONDICIÓN</b>	Se ha creado un recorrido y sus puntos asociados en la base de datos.
<b>FLUJO PRINCIPAL</b>	
<b>ACTOR</b>	<b>SISTEMA</b>
1. El usuario quiere crear un nuevo recorrido.  3. Introduce los datos del nuevo recorrido.  6. Inicia el grabado del recorrido.  8. Para el recorrido.  11. Acepta para terminar.	2. Se muestra el formulario para crear un nuevo recorrido.  4. Se comprueba la validez de los datos y salta a la siguiente pantalla. 5. Se configura el Gps y se informa al usuario.  7. Se van guardando los puntos periódicamente.  9. Se para el gps y salta hacia la siguiente pantalla. 10. Se muestran los datos recabados hasta el momento.  12. Se introduce el recorrido y sus puntos asociados
<b>FLUJO ALTERNATIVO [A4]</b>	1. Si el campo introducido muestra un mensaje informativo y no se hace nada.
<b>FLUJO ALTERNATIVO [A11]</b>	1. Si se pulsa "Cancelar" se vuelve al paso 5.

<b>FRECUENCIA DE USO</b>	Media.
<b>ANOTACIONES</b>	Mientras se está grabando el recorrido se puede pausar y reanudar con posterioridad.

*Figura 13: Descripción crear recorrido pasivo.*

*Caso de uso “Crear recorrido manual”*

<b>NOMBRE</b>	<b>Crear recorrido manual</b>
<b>ACTORES</b>	Usuario
<b>OBJETIVOS</b>	El usuario crea un nuevo recorrido de forma manual e instantánea
<b>PRECONDICIÓN</b>	
<b>POSTCONDICIÓN</b>	Se ha creado un recorrido y sus puntos asociados en la base de datos
<b>FLUJO PRINCIPAL</b>	
<b>ACTOR</b>	<b>SISTEMA</b>
1. El usuario quiere crear un nuevo recorrido.	2. Se muestra el formulario para crear un nuevo recorrido.
3. Introduce los datos del nuevo recorrido.	4. Se comprueba la validez de los datos y salta a la siguiente pantalla. 5. Se configura la visualización de Google Maps y se muestra al usuario.
6. Pulsa en el mapa, de forma sucesiva, el punto que quiere introducir.	7. Se van guardando los puntos periódicamente.
8. Termina de seleccionar los puntos.	9. Se salta hacia la siguiente pantalla 10. Se muestran los datos recabados hasta el momento.
11. Acepta para terminar.	12. Se introduce el recorrido y sus puntos asociados

<b>FRECUENCIA DE USO</b>	Alta.
<b>ANOTACIONES</b>	

Figura 14: Descripción crear recorrido manual.

*Caso de uso "Listar recorridos"*

<b>NOMBRE</b>	<b>Listar recorridos</b>
<b>ACTORES</b>	Usuario
<b>OBJETIVOS</b>	Se listan los recorridos creados
<b>PRECONDICIÓN</b>	
<b>POSTCONDICIÓN</b>	Hay una lista de los recorridos creados en la pantalla
<b>FLUJO PRINCIPAL</b>	
<b>ACTOR</b>	<b>SISTEMA</b>
1. El usuario quiere listar los recorridos.	2. Se carga la siguiente pantalla y se muestran los recorridos creados en ella.
<b>FRECUENCIA DE USO</b>	Alta
<b>ANOTACIONES</b>	Si no hay recorridos la pantalla estará vacía.

Figura 15: Descripción listar recorridos.

*Caso de uso "Consultar datos recorrido"*

<b>NOMBRE</b>	<b>Consultar datos recorrido</b>
<b>ACTORES</b>	Usuario
<b>OBJETIVOS</b>	Consultar todos los datos (nombre, comentarios y puntos) de un recorrido
<b>PRECONDICIÓN</b>	Es necesario haber seleccionado un recorrido
<b>POSTCONDICIÓN</b>	En la pantalla se muestran todos los datos del recorrido seleccionado
<b>FLUJO PRINCIPAL</b>	
<b>ACTOR</b>	<b>SISTEMA</b>
1. En la pantalla de listado de recorridos,	

el usuario pulsa sobre uno de estos.	2. Se salta a la siguiente pantalla mostrando en la pantalla todos los datos relacionados con el recorrido seleccionado.
<b>FRECUENCIA DE USO</b>	Media
<b>ANOTACIONES</b>	

*Figura 15: Descripción consultar datos recorrido.*

*Caso de uso “Eliminar recorrido”*

<b>NOMBRE</b>	<b>Eliminar recorrido</b>
<b>ACTORES</b>	Usuario
<b>OBJETIVOS</b>	Eliminar del sistema un recorrido y sus puntos asociados
<b>PRECONDICIÓN</b>	Es necesario haber seleccionado un recorrido
<b>POSTCONDICIÓN</b>	El recorrido y sus puntos asociados no existen en la base de datos
<b>FLUJO PRINCIPAL</b>	
<b>ACTOR</b>	<b>SISTEMA</b>
1. En la pantalla de consulta de datos de un recorrido el usuario borra ese recorrido.	2. Se borran de la base de datos el recorrido y todos sus puntos asociados.
<b>FRECUENCIA DE USO</b>	Media
<b>ANOTACIONES</b>	

*Figura 15: Descripción eliminar recorrido.*

*Caso de uso “Monitorizar recorrido”*

<b>NOMBRE</b>	<b>Monitorizar recorrido</b>
<b>ACTORES</b>	Usuario
<b>OBJETIVOS</b>	Monitorizar el recorrido actual según uno previamente seleccionado

<b>PRECONDICIÓN</b>	Es necesario tener al menos un recorrido creado
<b>POSTCONDICIÓN</b>	
<b>FLUJO PRINCIPAL</b>	
<b>ACTOR</b>	<b>SISTEMA</b>
<p>1. El usuario entra en el modo observador.</p> <p>3. Selecciona uno de ellos pulsando sobre el mismo.</p> <p>6. Comienza la monitorización.</p> <p>8. Para la monitorización del recorrido.</p>	<p>2. Se salta a la siguiente pantalla y se muestran los recorridos disponibles.</p> <p>4. Se pasa a la siguiente pantalla.</p> <p>5. Se cargan los datos del recorrido seleccionado y se configura el gps avisando al usuario al finalizar.</p> <p>7. Se van tomando los puntos actuales y se comprueba que coinciden en posición y tiempo con alguno de los puntos del recorrido creado (con un cierto margen de error).</p> <p>9. Se vuelve al menú principal.</p>
<b>FLUJO ALTERNATIVO [A7]</b>	Con el primer fallo de coincidencia se muestra un mensaje de error.
<b>FLUJO ALTERNATIVO-2 [A7]</b>	Con el segundo fallo de coincidencia el dispositivo vibrará.
<b>FLUJO ALTERNATIVO-3 [A7]</b>	Con el tercer fallo de coincidencia la aplicación solicitará mandar un email pre escrito al destinatario por defecto. Se vuelve a la pantalla principal.
<b>FRECUENCIA DE USO</b>	Alta
<b>ANOTACIONES</b>	Mientras se está monitorizando el recorrido se puede pausar y reanudar con posterioridad

Figura 15: Descripción monitorizar recorrido.



*Caso de uso "Cambiar idioma"*

<b>NOMBRE</b>	<b>Cambiar idioma</b>
<b>ACTORES</b>	Usuario
<b>OBJETIVOS</b>	Cambiar el idioma de la interfaz a uno más cómodo
<b>PRECONDICIÓN</b>	
<b>POSTCONDICIÓN</b>	El idioma de la aplicación ha cambiado
<b>FLUJO PRINCIPAL</b>	
<b>ACTOR</b>	<b>SISTEMA</b>
1. En el menú principal, el usuario presiona el botón de idioma de la barra superior.  3. Selecciona uno pulsando sobre el mismo.	2. Aparece una ventana emergente con los idiomas disponibles.  4. La aplicación cambia su configuración para usar el idioma seleccionado
<b>FRECUENCIA DE USO</b>	Media
<b>ANOTACIONES</b>	

*Figura 15: Descripción cambiar idioma.**Caso de uso "Cambiar email"*

<b>NOMBRE</b>	<b>Cambiar email</b>
<b>ACTORES</b>	Usuario
<b>OBJETIVOS</b>	Cambiar el email predeterminado al que irán los avisos de fallo del modo observador.
<b>PRECONDICIÓN</b>	
<b>POSTCONDICIÓN</b>	El email predeterminado ha cambiado
<b>FLUJO PRINCIPAL</b>	
<b>ACTOR</b>	<b>SISTEMA</b>
1. En cualquier momento, el usuario	

<p>presiona el botón de email de la barra superior.</p> <p>3. Cubre el campo con el email deseado.</p>	<p>2. Aparece una ventana emergente solicitando el nuevo email.</p> <p>4. La aplicación cambia su configuración para usar el email seleccionado.</p>
<b>FRECUENCIA DE USO</b>	Media
<b>ANOTACIONES</b>	

*Figura 15: Descripción cambiar email.*

*Caso de uso “Mostrar datos de polen”*

NOMBRE	Mostrar datos de polen
ACTORES	Usuario
OBJETIVOS	Mostrar los datos de polen de un tipo de polen en concreto
PRECONDICIÓN	Es necesario poseer en la carpeta descargas del dispositivo el archivo con los datos
POSTCONDICIÓN	En el mapa aparece la zona dibujada de un color según la concentración de polen
FLUJO PRINCIPAL	
ACTOR	SISTEMA
1. El usuario va al mapa de visualización del polen.	2. Se salta a la siguiente pantalla. 3. Se recoge la ubicación actual y se coloca en un mapa de Google Maps. 4. Se dibuja el área circundante de un color, según la cantidad de polen
FRECUENCIA DE USO	Alta
ANOTACIONES	Se toma el primer tipo de polen por defecto.

*Figura 16: Descripción mostrar datos de polen.*

*Caso de uso "Cambiar tipo de polen"*

<b>NOMBRE</b>	<b>Cambiar tipo de polen</b>
<b>ACTORES</b>	Usuario
<b>OBJETIVOS</b>	Mostrar los datos de polen de un tipo de polen en concreto
<b>PRECONDICIÓN</b>	Es necesario poseer en la carpeta descargas del dispositivo el archivo con los datos
<b>POSTCONDICIÓN</b>	En el mapa aparece la zona dibujada de un color según la concentración de polen para el tipo de polen seleccionado
<b>FLUJO PRINCIPAL</b>	
<b>ACTOR</b>	<b>SISTEMA</b>
1. El usuario pulsa sobre uno de los tipos de polen disponible	2. Se desdibuja la concentración de polen activa. 3. Se recoge del archivo los datos del tipo de polen seleccionado y se dibuja el área de concentración de polen
<b>FRECUENCIA DE USO</b>	Alta
<b>ANOTACIONES</b>	

*Figura 17: Descripción cambiar tipo de polen.*

## 2.4 Análisis de datos

### 2.4.1 Diagrama Entidad-Relación

Siguiendo el modelo Entidad-Relación, en los diagramas que se presentan a continuación se muestran las entidades identificadas para el sistema, junto con las interrelaciones entre ellas y los atributos que las forman.

Dado que el proyecto es desarrollado para Android como una aplicación sencilla, el número de entidades es reducido, de modo que se ahorra espacio en el dispositivo en cuanto al uso de tablas de SQL.

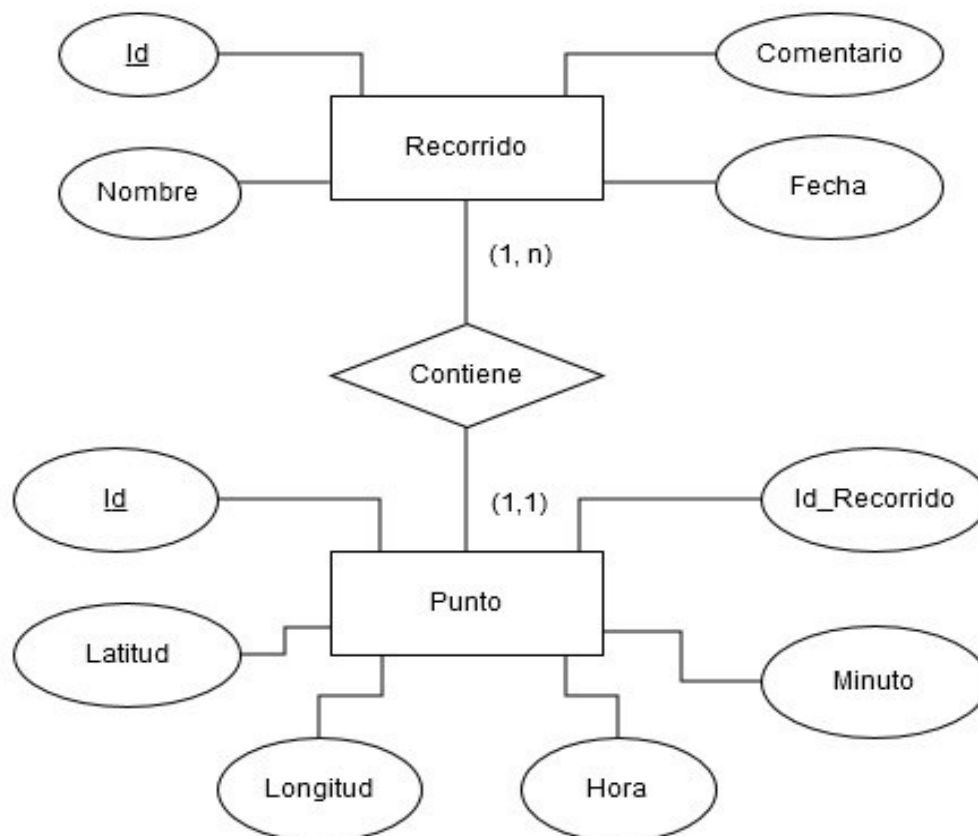


Figura 18: Diagrama entidad-relación base.

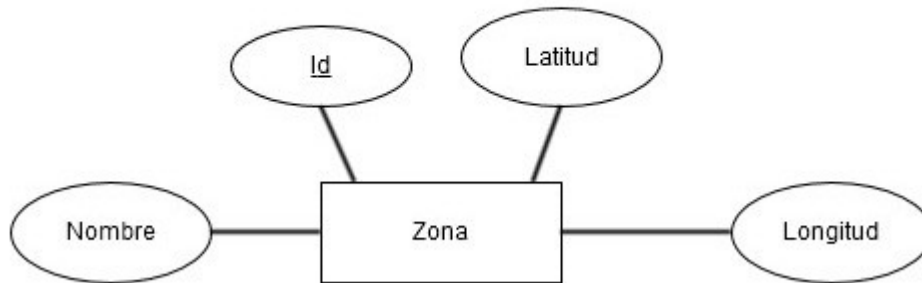


Figura 19: Diagrama entidad-relación polen.

## 2.4.2 Diccionario de datos

### Entidades

Las entidades identificadas fueron:

- **Recorrido (track):** representa los recorridos creados manualmente mediante la opción “crear recorrido”.
- **Punto (point):** representa un punto geográfico en un instante determinado.
- **Zona (zone):** representa las zonas que están presentes en el archivo de datos de polen (Orense, Lugo, etc).

### Relaciones

Las relaciones identificadas fueron:

- **Contiene:** representa que cada recorrido está formado por varios puntos geográficos, mientras que los puntos geográficos están asociados a un único recorrido.

### Atributos

Los atributos identificados para cada entidad fueron:

- **Recorrido:**
  - **\_id:** Identificador único de cada recorrido.

- **name:** Nombre del recorrido seleccionado por el usuario.
- **comments:** Comentarios que el usuario quiera realizar sobre el recorrido. Opcional.
- **date:** Fecha en la que el recorrido se llevó a cabo. Se tomará la fecha en el momento que se confirme el recorrido creado.
- **Punto:**
  - **\_id:** Identificador único del punto.
  - **latitude:** Latitud geográfica del punto.
  - **longitude:** Longitud geográfica del punto.
  - **track\_id:** Identificador del recorrido al que está asociado el punto.
  - **hour:** hora en la que se tomó ese punto.
  - **minute:** minuto en el que se tomó ese punto.
- **Zona:**
  - **\_id:** Identificador único para cada zona.
  - **name:** Nombre de la zona. Coincide con las tablas del archivo de datos de polen.
  - **latitud:** Latitud central de la zona.
  - **longitud:** Longitud central de la zona.

## 2.5 Desarrollo de aplicaciones en Android

La elección de desarrollar la aplicación para el sistema Android implica adaptar el diseño de la misma a la estructura proporcionada establecida por dicho sistema.

Android, además de permitir el desarrollo siguiendo el modelo MVC, opta por una filosofía en que las pantallas tienen asociadas una serie de archivos, de forma que sugiere unas pautas para el nombre de las clases o los archivos asociados, así como limitaciones en cuanto a la comunicación entre las clases. Si bien esas restricciones suponen en cierta medida un problema, a su vez facilita la separación de las funciones y sus distintas clases usadas para la implementación.

A continuación se explicarán las principales características y como su utilización han influido en el diseño de la aplicación.

### 2.5.1 MVC

MVC es el acrónimo de Modelo-Vista-Controlador (15), un patrón de diseño de software que consta de tres capas con funciones diferenciadas:

- Los modelos se encargan del manejo de datos en la aplicación. Implementan funciones para comunicarse con la base de datos y modificar o procesar los datos que obtiene de ella o los controladores
- Las vistas muestran al usuario los resultados de las operaciones realizadas por modelos y controladores.
- Los controladores se encargan de la lógica de la aplicación, comunicando operaciones a los modelos, recibiendo datos de éstos y pasándoselos a las vistas para su visualización.

Esta arquitectura en tres capas de funcionalidades diferenciada aporta múltiples ventajas:

- Simplifican el desarrollo al facilitar la identificación de los problemas por su naturaleza (operaciones con datos, visualización o lógica del sistema).
- Facilita el mantenimiento del código.

- Permite realizar el trabajo de programadores y diseñadores a la vez.
- Simplifica la modularización de las aplicaciones.

## 2.5.2 Clases en Android

Android consta de varios tipos de archivos que implementan los distintos niveles de MVC. Todo lo relacionado con las vistas se implementan mediante “layouts” y archivos en formato .xml (16). El resto se implementa usando archivos .class de Java que pueden hacer uso de la gran biblioteca de funcionalidades que implementa ya Android.

### 2.5.2.1 Nociones Básicas

Para atender a todas las acciones y peticiones que se hagan en la pantalla del dispositivo es necesario disponer de una clase para cada pantalla asociada. Esta clase debe asociarse mediante el archivo `AndroidManifest.xml` a su correspondiente pantalla e instanciar el layout mediante el método `setContentView()`. A partir de ahora se llamarán a estas clases Controladores (Activity para Android). Generalmente, cada Activity efectúa como controladora de la vista que renderiza.

Los controladores deben implementar el método `onCreate()`, que es el método llamado cuando la clase se crea y se instancia.

A continuación, como en cualquier otra aplicación JAVA, se pueden crear las clases que se consideren necesarias para implementar la lógica de la aplicación, usando, si así se desea, toda la biblioteca de paquetes con métodos y utilidades que Android pone a disposición del usuario.

### 2.5.2.2 Entidades

Los modelos son los encargados de manejar los datos de la aplicación. Lo más habitual es que estos datos se encuentren almacenados en una base de datos. Para



comunicar los modelos con las bases de datos se utilizan datasources, clases que abstraen la comunicación de un modelo y su origen de datos. En este caso se utilizará SQLite para crear y comunicar la base de datos con el resto de la aplicación.

Para cada modelo particular en el sistema se creará una clase, que se nombrará por convenio y simplificación igual que el objeto que va a manejar, con la primera letra en mayúscula y se encuentran en el paquete “Entities”. En cada modelo se implementarán sus métodos específicos y la definición de sus atributos.

### **2.5.2.3 Controladores**

Los controladores se encargan de recibir los datos de una petición y lanzar los métodos necesarios para atenderla, obteniendo los datos de los modelos y pasando los resultados a las vistas. De forma análoga a los modelos, los controladores son clases que se definen para realizar esas acciones de control.

Para identificar y separar las clases que implementan modelos o conjuntos de métodos de las controladoras, a estas últimas se le añadirá después de su nombre la palabra “Controller” e estarán ubicadas en el paquete “Controller” de la aplicación.

### **2.5.2.4 Clases de soporte**

Las clases de soporte son las clases creadas específicamente para esta aplicación. Como su nombre indica, proporcionan métodos que ayudarán a alcanzar los objetivos planteados para este proyecto.

Un ejemplo es “Menu” que define y maneja el comportamiento de los botones superiores que conforman el menu global de la aplicación o SqlDAO, una clase que suministra los métodos necesarios para poder comunicarse con la base de datos.

Los nombres de estas clases intentarán ser lo más autoexplicativos posibles, pudiendo con solo echarles un vistazo conocer su funcionalidad final. Están ubicadas en el paquete “Support”.

### 2.5.2.5 Vistas y archivos de configuración

Las vistas muestran la salida que genera una petición del usuario. Para ello, Android utiliza “layouts” y objetos “View” que se anidan unos con otros para dar la estructura visual deseada en la pantalla del dispositivo.

Estos archivos se almacenan en formato xml para que el sistema los pueda leer y procesar de manera mucho más rápida. Así mismo, existen elementos contenidos en esas vistas que están definidos en otros archivos .xml, como puede ser el texto que se muestra, el formato de texto o la apariencia de los botones. Además, todos estos elementos son accesibles desde el código Java de nuestra aplicación gracias al paquete R., que mediante sus extensiones R.string, R.drawable, etc nos permite acceder a los elementos y valores concretos de las vistas.

Android clasifica estos elementos como recursos y los empaqueta automáticamente en el paquete “res” del proyecto.

### 2.5.3 Estructura de una aplicación en Android

La estructura final de la aplicación según el framework de desarrollo Android Studio se divide de la siguiente forma:

- *src*: Contiene los archivos necesarios para compilar y ejecutar la aplicación. Se divide en varias carpetas:
  - *src/main/java*: Una vez el proyecto está compilado, todos los archivos .class necesarios para la aplicación estarán en esta carpeta.
  - *src/main/res/layout*: Los archivos .xml correspondientes a las distintas pantallas o vistas de la aplicación.
  - *src/main/res*: El resto de recursos usados por la aplicación (imágenes, archivos .xml con la estructura de formato de texto, etc) están contenidos en este directorio, estructurado a su vez en subcarpetas.
  - *src/main/AndroidManifest.xml*: Este archivo describe las características fundamentales de la aplicación y los componentes que la forman.
- *libs*: Contiene archivos de librerías externas que se han enlazado a la aplicación para desarrollar ciertas partes concretas que hacían uso de esas librerías.

## **2.6 Clases de la aplicación**

### **2.6.1 Controladores**

Primeramente, se procederá a explicar las clases controladores de la aplicación. Se ha intentado que los nombres de las clases y sus métodos sean lo más auto explicativos posibles, para que con un simple vistazo a su nombre se pueda deducir

su función.

### Clase *MainController*

<b>MainController</b>
+onCreate() : void
+NewTrack() : void
+ListTrack() : void
+ShowMap() : void
+Observer() : void

La clase *MainController* se ejecuta al iniciar la aplicación. Su finalidad es la de controlar la pantalla principal, presentar al usuario dicha pantalla y sus diversas opciones y redirigirlo hacia cualquier opción escogida, así como de servir como nuevo punto de encuentro al completar alguna acción.

Con el método onCreate() únicamente le indicamos que layout asociado debe cargar (esto método existe para todas las clases controladoras y es obligado cargar el layout aquí) y luego dispone de los métodos NewTrack(), ListTrack(), ShowMap() y Observer() encargados de redirigir al usuario a otra Activity según la opción escogida.

### Clase *NewTrackController*

<b>NewTrackController</b>
-name : EditText
-comment : EditText
-MESSAGE_NAME : String
-MESSAGE_COMMENT : String
+onCreate() : void
+nextGoogle() : void
+nextGps() : void

La clase *NewTrackController* es llamada cuando se presiona el botón “Crear recorrido” del menú principal. Se encarga de controlar la pantalla de creación de nuevo recorrido, validar los datos del formulario y continuar a la siguiente pantalla.

Dispone de los métodos nextGoogle() y nextGps(), que efectúan la comprobación de que el nombre del recorrido no está vacío y luego redirige al usuario a la siguiente pantalla cargando el controlador GoogleController o GpsController, respectivamente.

<b>GpsController</b>
-gps : GpsTracker
-name_selected : String
-comment_selected : String
-actual_location : TextView
-MESSAGE_NAME : String
-MESSAGE_COMMENT : String
-points : ArrayList<Point>
-flag : int
+onCreate() : void
+setVariables() : void
+addPoint(Point) : void
+start() : void
+pauseLocation() : void
+stop() : void

### Clase *GpsController*

La clase *GpsController* es la encargada de manejar la pantalla de grabado del recorrido y uso del Gps, usando el Gps hardware del dispositivo del usuario.

Los métodos que usa son: onCreate(), que en este caso

además de cargar el layout correspondiente configura el Gps para su utilización; `setVariables()` para establecer los valores de la vista; `addPoint(Point)` que recibe un Punto y lo agrega a un array de Puntos; `start()` que inicia la toma de puntos y los va agregando a un array; `pauseLocation()` para pausar el recorrido y `stop()` que desconecta el Gps y carga la siguiente Activity.

Esta clase hace una implementación de `GpsTracker` (la cual se explicará más adelante) y sobrescribe el método `onLocationChanged()` para adaptarlo a las necesidades de esta clase concreta.

Además dentro de este controlador existe una implementación de `LocationListener`, cuyos métodos se explicarán más adelante.

### Clase *FinalTrackController*

<b>FinalTrackController</b>
-name_selected : String
-comment_selected : String
-points : List<Point>
-track : Track
-sql : SqlDAO
+onCreate() : void
+Cancel() : void
+Accept() : void
+insertTrack() : void
+insertPoints() : void

La clase *FinalTrackController* es la última clase controladora para crear un recorrido. Se encarga de volver atrás en caso de que el usuario así lo requiera o de hacer las inserciones oportunas en la base de datos y finalizar oficialmente el recorrido.

Sus métodos son: `onCreate()` donde se inicializan las variables y se abre la conexión con la base de datos; `Cancel()` para volver atrás; y `Accept()`, que llama a los métodos `insertTrack()` e `insertPoints()` para insertar el nuevo recorrido y sus puntos en la base de datos y posteriormente vuelve al menú principal.

### Clase *ListController*

<b>ListController</b>
-sql : SqlDAO
+onCreate() : void

Esta sencilla clase lo único de que se encarga es de mostrar al usuario la lista de recorridos creados y de manejar los datos que se llevarán a la siguiente pantalla cuando se haga click en algún recorrido, para poder visualizar sus datos detallados.

El único método que posee es `onCreate()`, en el que se cargan todos los recorridos creados de la base de datos, se muestran al usuario y se deja un listener para atender a qué recorrido hace click el usuario y poder redireccionarlo a la siguiente pantalla correctamente.

<b>DetailedController</b>
-name_selected : String
-comment_selected : String
-points : List<Point>
-track : Track
-sql : SqlDAO
+onCreate() : void
+Cancel() : void
+Delete() : void
+DeleteTrack() : void
+DeletePoint() : void

### Clase *DetailedController*

Con el uso de este controlador en la pantalla se le mostrarán al usuario los datos del recorrido que ha

previamente seleccionado, pudiendo además eliminar dicho recorrido del sistema (los puntos asociados se borrarán también de manera automática).

Sus métodos son: onCreate(), con el cual cargamos todos los valores necesarios de la base de datos y establecemos los elementos de la UI; Cancel() para volver al listado de recorridos y Delete() para borrar el recorrido del sistema, que usará deleteTrack() y deletePoint() para ello.

### Clase ObserverListController

ObserverListControll...
-sql : SqlDAO
+onCreate() : void

Análogamente a la clase ListController, esta clase lo único que hace es mostrar la lista de recorridos existentes y redireccionar a ObserverModeController en función del recorrido seleccionado.

El único método que posee es onCreate(), en el que se cargan todos los recorridos creados de la base de datos, se muestran al usuario y se deja un listener para atender a qué recorrido hace click el usuario y poder redireccionarlo a la siguiente pantalla correctamente.

### Clase ObserverModeController

ObserverModeController
-gps : GpsTracker
-name_selected : String
-comment_selected : String
-actual_location : TextView
-sql : SqlDAO
-points : List<Point>
-flag : int
-count : int
-flag_accept : int
-track : Track
+onCreate() : void
+setVariables() : void
+start() : void
+pause() : void
+stop() : void
+sendEmail() : void
+compareTime() : boolean
+compareCoords() : boolean
-now : Marker
-actual : LatLng
-name : String
-points : ArrayList<point>
-zones : List<Zone>
-db : Database
-polygon : Polygon
-value : int
-color : int
+onCreate() : void
+extractData() : void
+Draw() : void
+removeDraw() : void
+selectZone() : void

Con esta clase monitorizamos un recorrido proveniente de ObserverListController. Con ello observamos si los sitios por los que se va desplazando el usuario coinciden en tiempo y lugar (con un margen) con los sitios por los que transitó en el recorrido creado.

Tiene todos los métodos definidos en la clase GpsController, ya que usa el Gps para determinar la posición del usuario y utiliza el mismo layout con los botones “start”, “pause” y “stop”. Aunque hace una implementación diferente del método onLocationChanged de GpsTracker y añade los métodos compareTime() y compareCoords() para comparar que los puntos y horas de paso estean dentro los permitidos y sendEmail() para, en caso de haber dado 3 fallos, mandar un email a la dirección previamente configurada por el usuario.

### Clase PollenMapController

Probablemente esta sea la clase más extensa de la aplicación. Ella sola se encarga de mostrar la visualización en Google

Maps, recoger mediante GPS las ubicaciones del usuario según se vaya desplazando, acceder a la base de datos de polen y recoger los datos necesarios y dibujar las áreas de color en el mapa.

Dispone de abundantes métodos: `onCreate()` para inicializar todos los valores necesarios y las conexiones a bases de datos, así como configurar el gps y obtener un primer punto, el de la ubicación actual. `ExtractData()` usado para extraer del archivo de datos de polen los valores en cuanto al tipo de polen seleccionado, la zona donde se encuentre el usuario y la fecha actual. `Draw()` y `removeDraw()` para dibujar y desdibujar la zona coloreada de polen en el mapa y `selectZone()` que se ejecuta cada vez que el usuario se desplaza y comprueba en qué zona se encuentra.

## 2.6.2 Entidades

Cada entidad con representación en la base de datos necesita de una clase que efectuará de modelo, que será el encargado de guardar los valores e implementar el comportamiento de la clase que representa.

### Clase Track

Track
-id : int -name : String -comment : String -date : String
+getters() +setters() +toString()

La clase *Track* se utiliza como implementación de la entidad Recorrido. Unicamente implementa los métodos `get()` y `set()` comunes para tratar los valores de sus atributos y se reescribe `toString()` para formatear su visualización en la aplicación como puede ser por ejemplo en la funcionalidad “Listar recorridos”, entre otras.

### Clase Point

Point
-id : int -latitude : double -longitude : double -hour : int -minute : int -track_id : int
+getters() +setters() +toString() +writeToParcel() : void +createFromParcel() : Point +newArray() : Point

Esta clase implementa el modelo de la entidad Punto. Al igual que la anterior solo dispone de los métodos `get()`, `set()` y `toString()` comunes. Adicionalmente implementa la clase `Parcelable` y por lo tanto fue necesario sobrescribir los métodos `writeToParcel()`, `createFromParcel()` y `newArray()`. Esto fue debido a que Android no permite pasar entre `Activities` Listas de elementos que no sean de los tipos de datos básicos.



### Clase Zone

Zone
-id : int
-name : String
-latitude : double
-longitude : double
+getters()
+setters()
+toString()

Esta clase implementa la entidad Zona. Solamente disponer de los métodos get(), set() y toString() comunes.

## 2.6.3 Clases de soporte

Las clases de soporte proporcionan funcionalidades específicas para la aplicación. Solamente se mostrará una descripción detallada de cada clase sin su correspondiente diagrama de clases parciales debido a que aunque son clases separadas, su funcionalidad podría haber sido directamente implementada en la clase controladora que hiciese uso de ella, pero por razones de legibilidad y buena estructuración de la aplicación esa funcionalidad se decidió mover a una clase que se encargara única y exclusivamente de ello.

- **GlobalClass:** Usada para guardar variables globales a lo largo de toda la aplicación. Guarda el email seleccionado por el usuario y el contexto en el que se está ejecutando la aplicación (para que clases no controladoras puedan hacer uso de ciertos métodos).
- **Menu:** Define y maneja el menú superior de la aplicación.
- **EmailDialogFragment:** Muestra el popup para escribir el email deseado por el usuario y lo almacena.
- **LanguageDialogFragment:** Muestra el popup con los idiomas disponibles y cambia el idioma de la aplicación cuando se seleccione uno.
- **SqlController:** Crea la base de datos e inserta los primeros datos necesarios para que la aplicación funcione correctamente.
- **SqlDAO:** clase intermediaria para comunicarse con la base de datos y poder insertar y extraer información.



- **GpsTracker:** Es la implementación de LocationManager para este proyecto. Establece la conexión con el GPS y toma las coordenadas de posición del dispositivo. Es necesario sobrescribir el método onLocationChanged() acorde al controlador que la use, dado que este método se ejecuta siempre que la ubicación del dispositivo cambie.

## 2.6.4 Librerías externas

En esta aplicación ha sido necesario el uso de una librería externa para poder acceder a la base de datos de polen, que se encuentra en formato Microsfot Access.

Dicha librería, la cual ya ha sido detallada en secciones anteriores de esta documentación, se llama Jackcess y está disponible en Internet para cualquiera que requiera de su uso.

Para poder usar los métodos de dicha librería solo ha sido necesario agregarla al proyecto y posteriormente importar su paquete en la clase requerida.

## 2.7 Diagramas de secuencia

En esta sección se muestran los diagramas de secuencia de cada caso de uso, mostrando la interacción de las clases implicadas en su realización. Estos diagramas comparten una serie de elementos comunes, consecuencia de la estructura MVC y de las soluciones elegidas para implementar las funcionales.

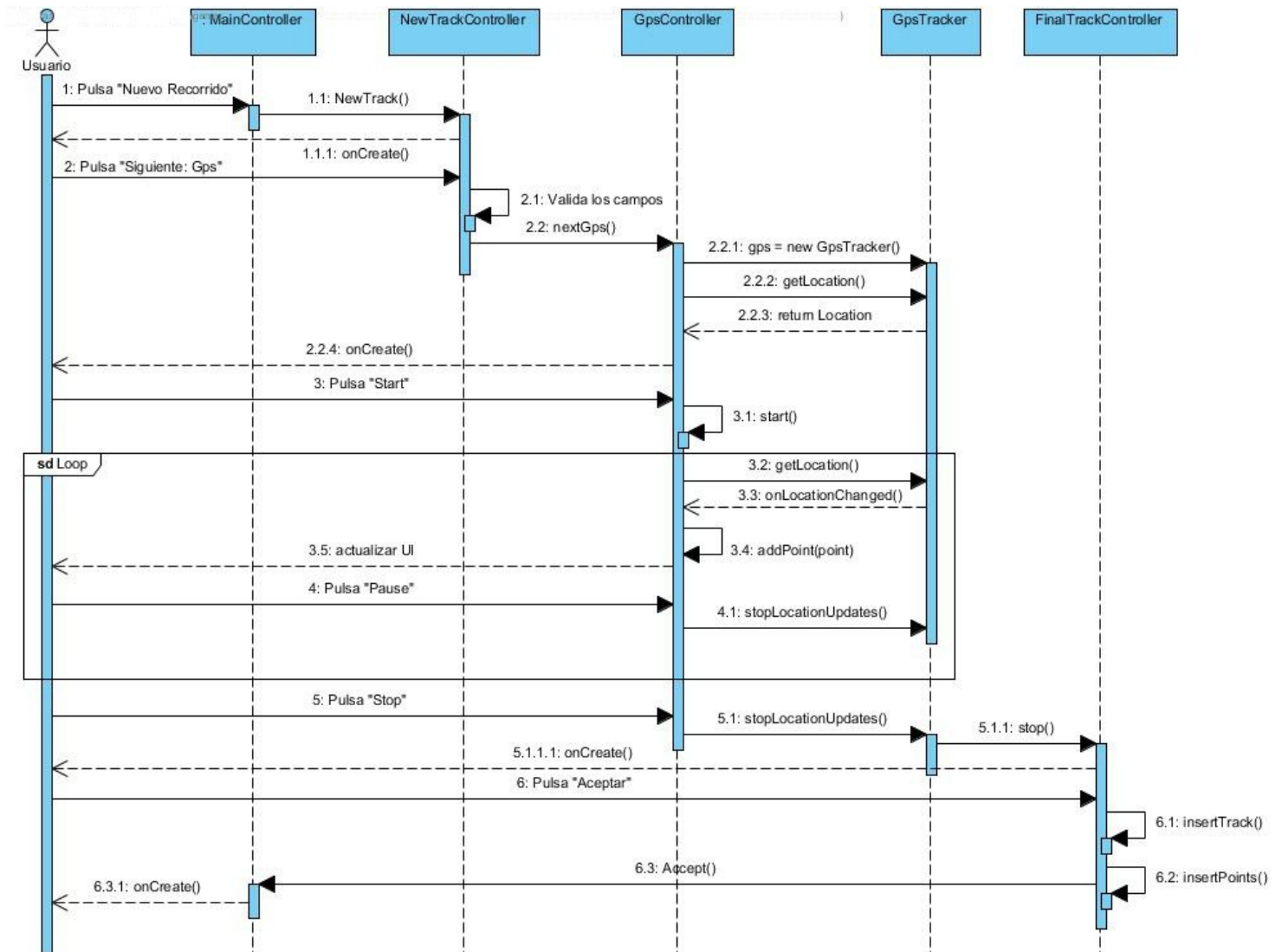
### 2.7.1 Observaciones sobre los diagramas

Todos los casos de uso comienzan de una forma similar: La clase controladora recibe la llamada a un método procedente de un elemento View de la vista y luego realiza las acciones pertinentes. Por ello, todos los casos de uso se inician con un actor que realiza una selección a través de un elemento de la pantalla de la aplicación.

### 2.7.2 Diagrama de secuencia “Crear recorrido pasivo”

Este diagrama muestra el proceso seguido para crear un recorrido pasivo en el

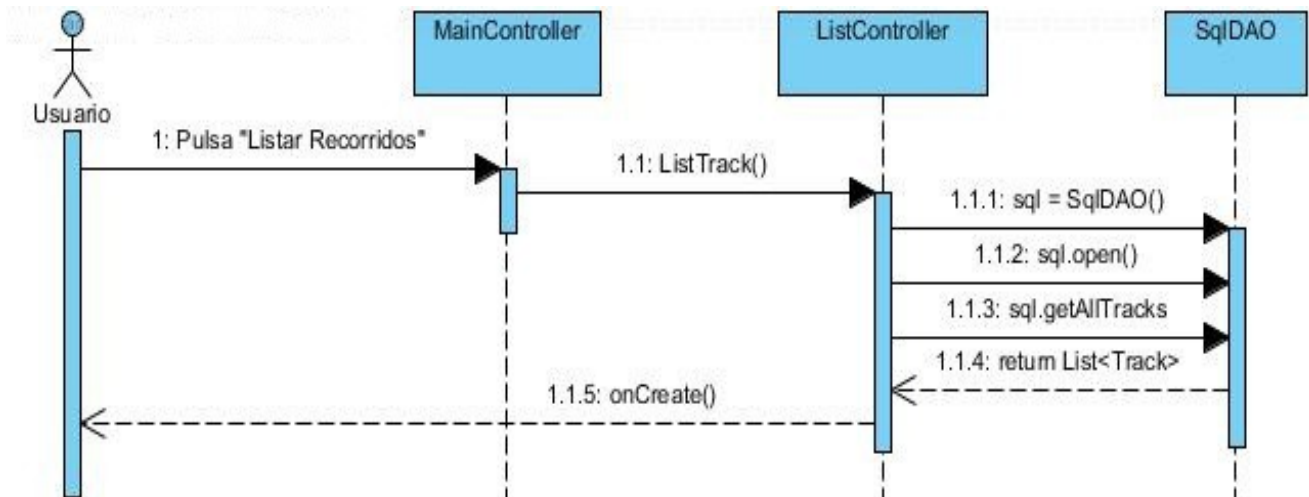
sistema.



### 2.7.3 Diagrama de secuencia “Crear recorrido manual”

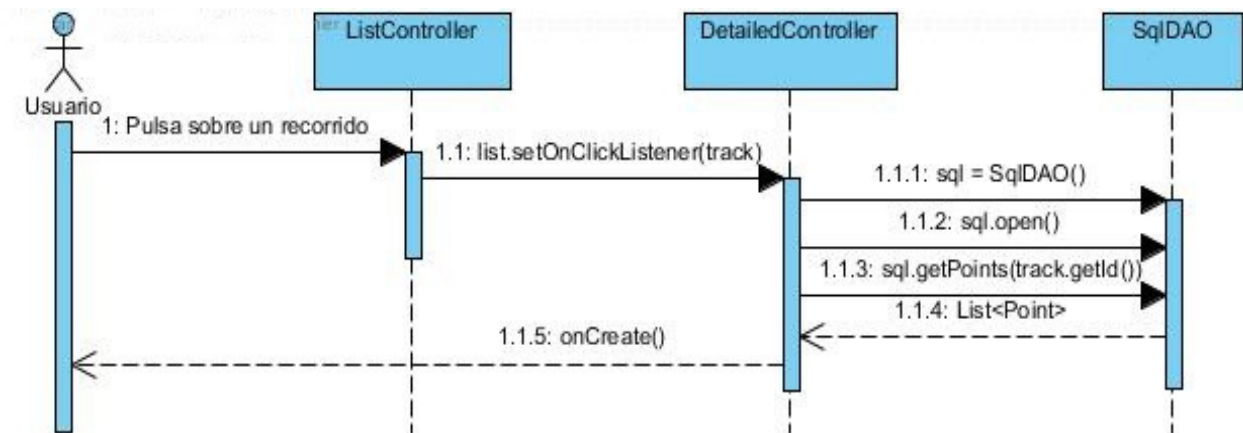
### 2.7.4 Diagrama de secuencia “Listar recorridos”

Este diagrama muestra el proceso seguido para listar los recorridos almacenados en el sistema.



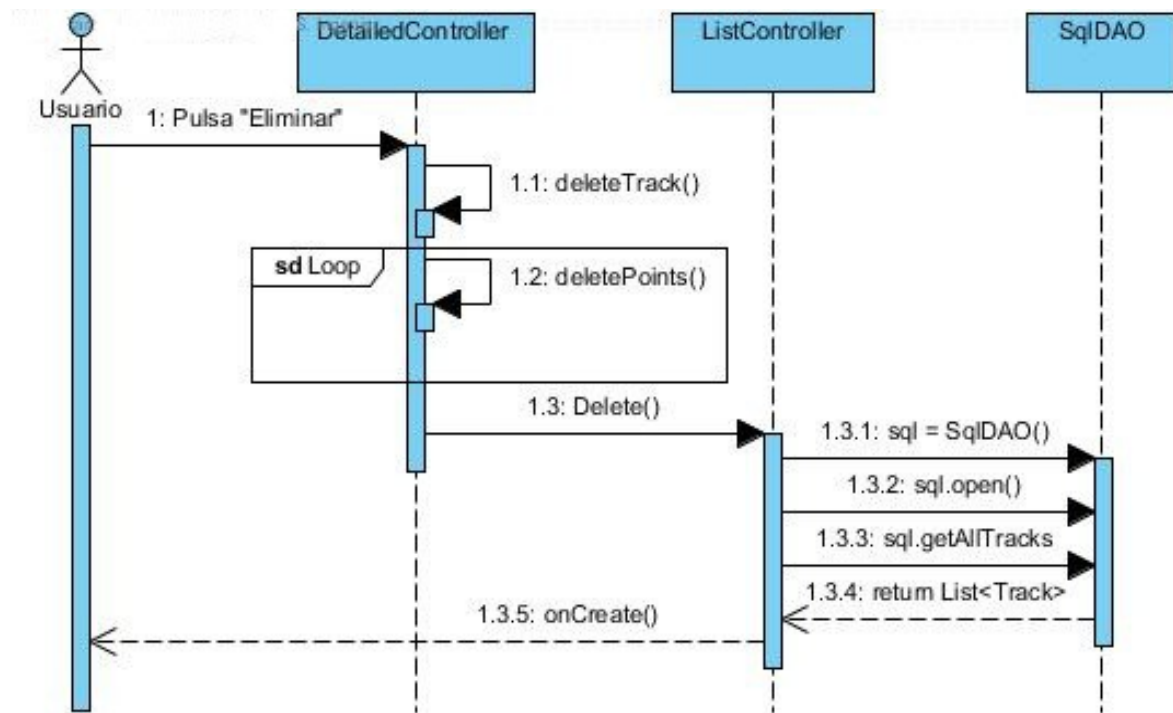
### 2.7.3 Diagrama de secuencia “Consultar datos recorrido”

Con este diagrama se puede observar el proceso llevado a cabo para consultar los datos sobre un recorrido.



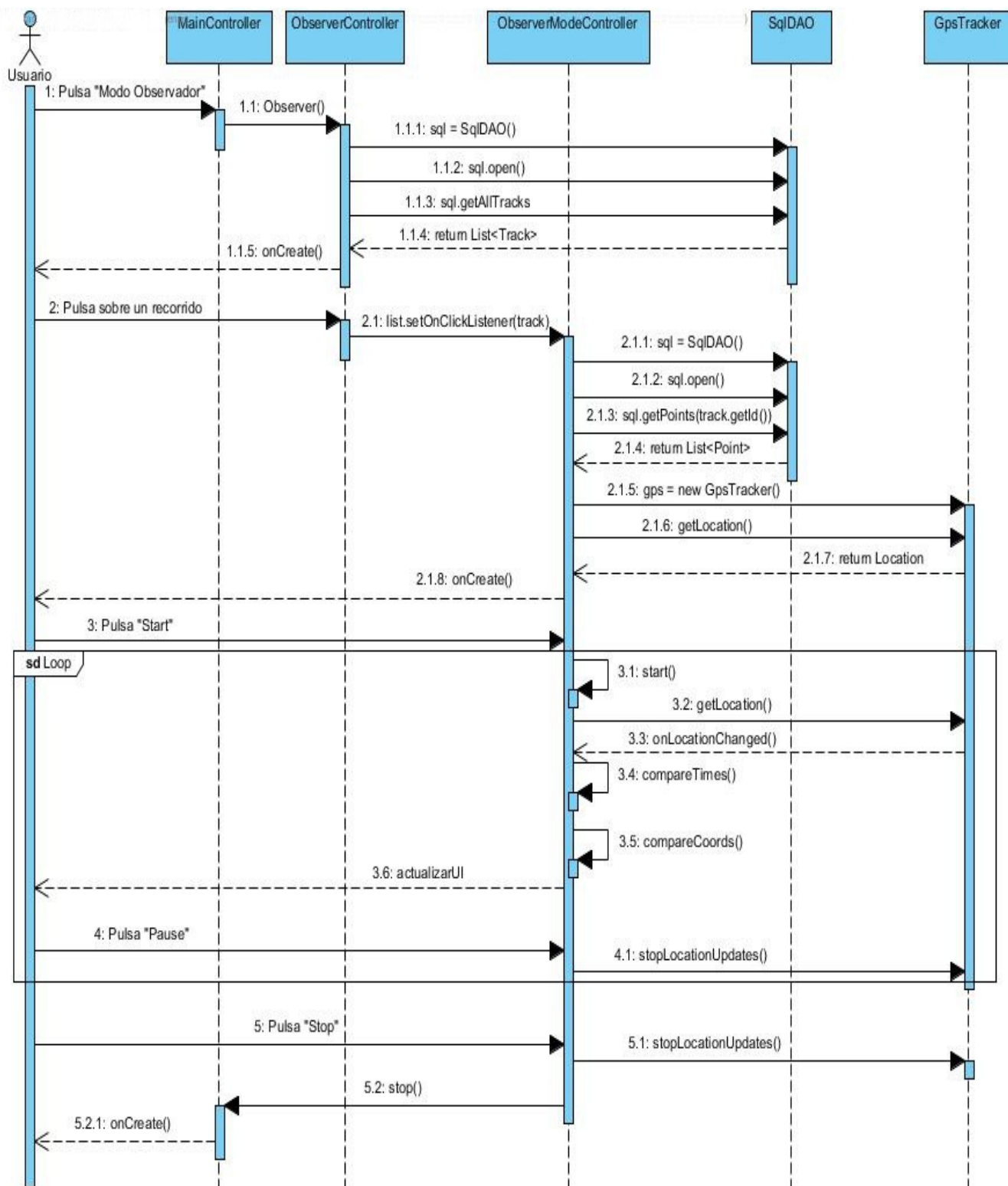
### 2.7.4 Diagrama de secuencia “Eliminar recorrido”

En este diagrama se muestra la secuencia necesaria para poder eliminar un recorrido de la base de datos.



### 2.7.4 Diagrama de secuencia “Monitorizar recorrido”

Con este diagrama se puede observar todo el proceso llevado a cabo para realizar el caso de uso monitorizar recorrido.



## 2.8 Diagrama de componentes

En esta sección se muestra el diagrama de componentes del sistema, en el cual se identifican los componentes principales del mismo y las relaciones que existen entre ellos. A pesar de que a lo largo de la documentación ya se han comentado todos los elementos que forman los sistemas, a través de este diagrama se obtendrá una visión global que permitirá comprender mejor la distribución diseñadafgdfdfdf.

Los componentes que forman el sistema se distribuyen de la siguiente manera: