UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Grado en Ingeniería Informática en Ingeniería del Software

Trabajo Fin de Grado

Refactorización de Aplicaciones Android usando Patrones de Arquitectura

José Manuel Campanón Toro

Enero, 2018

UNIVERSIDAD DE EXTREMADURA
Escuela Politécnica
Grado en Ingeniería Informática en Ingeniería del Software

Trabajo Fin de Grado

Refactorización de Aplicaciones Android usando Patrones de Arquitectura

Autor: José Manuel Campanón Toro
Fdo:

Tutor: Roberto Rodríguez Echeverría
Fdo:

Tribunal Calificador

Presidente: Pedro J. Clemente
Fdo:

Secretario: José María Conejero
Fdo:

Vocal: Encarna Sosa
Fdo:

Índice general

Índice de figuras

Resumen

Entre otros objetivos que se describirán posteriormente, el objetivo principal del proyecto era elaborar una guía completa y muy meticulosa para la refactorización y migración de arquitecturas en aplicaciones Android. Por lo que el proyecto sirve como guía para otros desarrolladores interesados en ello.

Para elaborar la guía, se ha realizado una refactorización y una migración de arquitectura en una aplicación heredada. Puesto que la aplicación es heredada, la modificación de la misma requiere un estudio previo para conocer su funcionamiento, estructura y tecnologías utilizadas en el desarrollo. Todos estos pasos se han agregado a la guía para ayudar al desarrollador en todas las fases.

Además, en el proyecto se realiza un análisis en profundidad de los principales patrones de arquitectura, presentación y organización utilizados en el desarrollo de aplicaciones Android.

En el congreso de desarrolladores organizado anualmente por Google llamado Google I/O '17 el equipo de desarrolladores Android sacó a la luz un patrón de arquitectura estándar para la implementación en sus aplicaciones. Se ha estudiado en profundidad esta nueva arquitectura y su implementación en una aplicación.

La aplicación sobre la que se va aplicar la refactorización y migración se llama YourInstantApp y tiene por objetivo extraer información de conjuntos de datos a través de repositorios públicos de Open Data y dotar al usuario de un conjunto de herramientas que le permita, en definitiva, darles forma. Posibilitando que usuarios sin conocimientos puedan crear sus propias aplicaciones móviles con una serie de pasos guiados y con un tiempo de desarrollo mínimo.

Summary

Among other objectives that will be described later, the main objective of the project was to elaborate a complete and very meticulous guide for the refactoring and migration of architectures in Android applications. So the project serves as a guide for other developers interested in it.

In order to elaborate the guide, a refactoring and an architecture migration have been performed in a legacy application. Since the application is inherited, the modification of the same requires a previous study to know its operation, structure and technologies used in the development. All these steps have been added to the guide to help the developer in all phases.

In addition, the project carries out an in-depth analysis of the main architecture, presentation and organization patterns used in the development of Android applications.

At the developer congress organized annually by Google called Google I / O '17 the Android developer team brought to light a standard architecture pattern for implementation in their applications. This new architecture and its implementation in an application has been studied in depth.

The application on which to apply the refactoring and migration is called **YourInstantApp** and Its main objective consists in extracting information from public Open Data datasets and provide a complete set of tools to final users of the application that allows them, definitely, to manipulate that data. This manipulation is completely programming knowledge free (at least, this knowledge is not necessary at all); users can create their own apps following a few steps in order to achieve that creation. Of course, time spent making the app is minimum.

Capítulo 1

Introducción

El desarrollo de aplicaciones Android se realiza, en muchas ocasiones, dentro de plazos de desarrollo muy estrictos que inciden negativamente en la calidad del código final de las mismas. Es por ello, que surgen año tras año patrones de organización de código que mejoran el desarrollo de aplicaciones de manera que la calidad del código mejore considerablemente.

En ocasiones el desarrollo de aplicaciones se realiza sin utilizar patrones o técnicas de organización de código, o se utilizan de manera incorrecta ya que muchos desarrolladores no le dan la importancia que merecen. Ésto acaba pasando factura en el desarrollo de las aplicaciones y se puede convertir en un verdadero problema según aumenta la complejidad de la aplicación.

A raíz de esta mala práctica nace el término **refactorización**. La refactorización es el proceso de cambiar un sistema de software de manera que no se altera el comportamiento externo del código y se mejora su estructura interna [3]. En definitiva, mejorar el diseño después de haber sido escrito.

En algunas ocasiones se comprueba que la arquitectura diseñada para una aplicación no se ha implementado de forma correcta o realmente no tiene mucho sentido para la aplicación en cuestión, es por ello que se decide rediseñar la misma.

1.1. Objetivos

El principal objetivo es definir un proceso que nos permita tener una guía para la refactorización y migración de aplicaciones Android heredadas o mal diseñadas. Para ello se van a aplicar estos procesos en una aplicación desarrollada por otro alumno anteriormente. Además otros objetivos del proyecto son:

- Conocimiento de herramientas de análisis estático de código y su utilización en aplicaciones Android.
- Aprender a enfrentarse a aplicaciones heredadas, esta tarea puede resultar complicada en función de la organización y desarrollo de la aplicación y de la experiencia del desarrollador.
- Aprendizaje de patrones de arquitectura utilizados en aplicaciones Android, hay diferentes patrones para el desarrollo de aplicaciones por ello se debe aprender a valorar cual es eficaz en cada aplicación y que ventajas y desventajas tiene cada uno.
- Conocimiento de procesos básicos de refactorización en aplicaciones Android.

Capítulo 2

Antecedentes

Para poner en práctica el proceso de refactorización, se debe partir de software heredado o de software que no siga unos patrones de organización correctos en su código.

Por ello, se va a aplicar una refactorización en una aplicación desarrollada por otro alumno en su Trabajo Fin de Grado titulado **YourInstantApp: Herramienta para el desarrollo asistido de aplicaciones en entornos móviles**.

YourInstantApp es un proyecto que tiene por objetivo extraer información de conjuntos de datos a través de repositorios públicos de Open Data y dotar al usuario de un conjunto de herramientas que le permita, en definitiva, darles forma. Posibilitando que usuarios sin conocimientos en programación o con ellos, pero sin necesidad de usarlos, puedan crear sus propias aplicaciones móviles con una serie de pasos guiados y con un tiempo de desarrollo mínimo. Involucrando al propio usuario para que sea él quien decida qué datos desea consumir. Se trata de que el proceso de construcción de una aplicación no suponga un esfuerzo técnico, sino un esfuerzo creativo.

2.1. Estructura del proyecto

La estructura que sigue el proyecto antes de la refactorización es la siguiente:

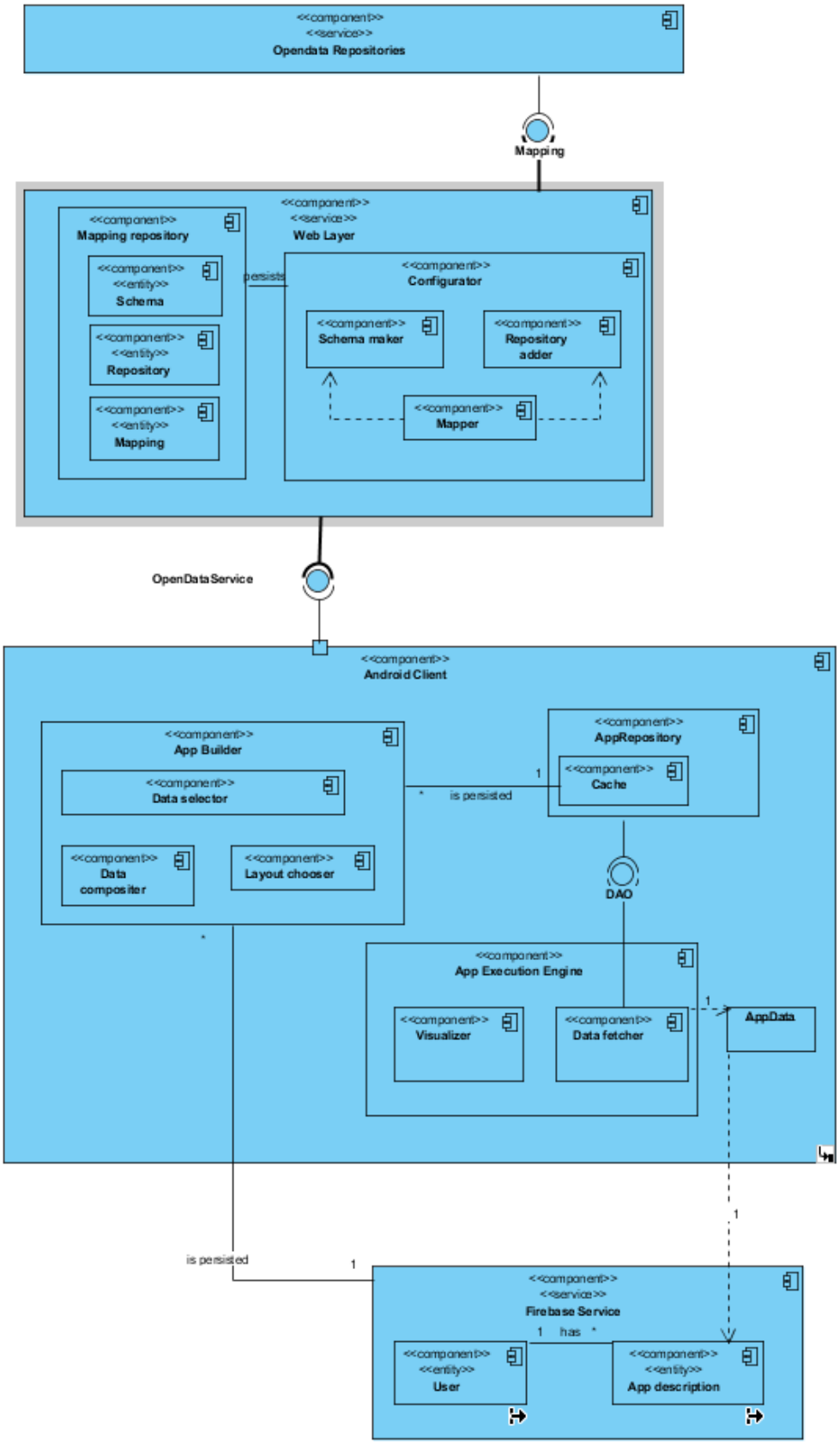

Figura 2.1: Estructura de YourInstantApp antes de la refactorización

En el diagrama anterior se observan los siguientes componentes:

1. Repositorios OpenData

   La aplicación extrae información de diferentes repositorios OpenData, como puede ser opendata-cáceres, opendata-tenerife, etc. Aunque son ajenos a nuestro control, se representa la conexión y su importancia en el sistema. Se realiza un mapeado de los datos para poder utilizarlos.

2. Capa Web

   A través de la página web cualquier usuario con los conocimientos adecuados puede relacionar conceptos, añadir esquemas, repositorios, etc... Su objetivo es servir a la plataforma móvil; la web extraerá datos de los repositorios, realizará el mapeado de los datos del repositorio utilizando el esquema indicado por el usuario y finalmente almacenará el mapeado.

3. Cliente Android

   La aplicación Android permite a los usuarios crear aplicaciones con los datos extraídos de Open Data de forma sencilla independientemente de sus conocimientos (no es necesario que sepan que es Open Data o saber programación). Dichas aplicaciones generadas se almacenan en la base de datos local.

4. Repositorio App

   Se ocupa de almacenar las aplicaciones creadas por el usuario en la plataforma móvil utilizando una base de datos SQLite.

5. Base de datos remota (FireBase)

   Firebase es un proveedor de servicios en la nube del cual en la la aplicación se utilizan sus componentes de autentificación y base de datos.

   La aplicación se sincronizará con la base de datos (Firebase) para traer o subir las aplicaciones que se tienen y gracias a la identidad federada lo único que hará falta para descargarse esas aplicaciones en otros dispositivos móviles será tener una cuenta en alguno de los proveedores registrados.

2.2. Cliente Android

El cliente Android del proyecto YourInstantApp va a ser donde se ponga en práctica el proceso de refactorización.

Esta aplicación es idónea para aplicar una refactorización porque es una aplicación heredada que se construyó tratando de seguir una arquitectura **Model - View - Presenter**, pero no se implementó de forma correcta y es muy tedioso añadir nuevas funcionalidades y continuar el desarrollo de la misma.

Como se ha explicado anteriormente, la principal funcionalidad de la aplicación es la generación de aplicaciones gracias a los datos extraídos de repositorios OpenData, en la aplicación Android este proceso se realiza siguiendo 5 pasos:

1. Selección de categorías (Eventos, restaurantes...).
2. Selección del repositorio de OpenData.
3. Filtrado de los atributos deseados.
4. Selección del formato de visualización de datos (Mapa, lista...).
5. Descripción y nombre de la aplicación creada.

La refactorización de la aplicación **no modifica** la funcionalidad de la aplicación heredada. También hay que destacar que este proceso tampoco afecta a la interfaz por lo que **no cambiará**.

Una forma de evitar fallos en la funcionalidad y cambios en la interfaz en el proceso de refactorización es desarrollar pruebas de implementación.

Capítulo 3

Tecnologías relacionadas

Puesto que el principal objetivo es definir un proceso que nos permita tener una guía para la refactorización y migración de aplicaciones android, se van a utilizar diferentes tecnologías relacionadas con los patrones de arquitectura, herramientas de análisis estático de código y con los procesos de refactorización en aplicaciones android.

3.1. Refactorización de aplicaciones

Tras leer el libro *Refactoring: Improving the design of existing code* de Martin Fowler [3], podría definir la refactorización como un arte, este libro es una biblia para aquellos que quieren dar una nueva vida a sus proyectos de software. En él se define la refactorización como el proceso de cambiar un sistema de software de manera que no se altera el comportamiento externo del código y se mejora su estructura interna. En definitiva, mejorar el diseño después de haber sido escrito.

Este libro me ha servido como guía de qué se debe hacer y qué no en la refactorización además de enseñarme muchas de las técnicas que se van a aplicar en el proyecto. Como dice Martin Fowler en su libro:

"Any fool can write code that a computer can understand. Good programmers write code that humans can understand. Martin Fowler "

3.2. Patrones de diseño y arquitectura

A continuación, se explican brevemente algunos de los patrones y prácticas utilizadas en el proceso de refactorización de la aplicación heredada explicado posteriormente.

3.2.1. Patrones de diseño

Model - View - ViewModel Architectural Pattern

Este patrón de diseño separa los datos de la aplicación y la interfaz de usuario pero en vez de controlar manualmente los cambios en la vista o en los datos, estos se actualizan directamente cuando sucede un cambio en ellos.[4]

Separation of concerns

Es un principio de diseño para separar la aplicación en secciones distintas, tal que cada sección enfoca un interés delimitado. Un ejemplo de separación de intereses es la división de la aplicación en capas.[5]

Data access object

Es un patrón de diseño utilizado para crear una capa de persistencia encapsulando el acceso a la base de datos. Por lo que cuando la capa lógica de negocio necesite interactuar con la base de datos, va a hacerlo a través de la API que le ofrece DAO. Generalmente esta API consiste en métodos CRUD (Create, Read, Update, y Delete).[6]

Dependency injection

Es un patrón de diseño orientado a objetos, en el que se suministran objetos a una clase en lugar de ser la propia clase la que cree el objeto. [7]

Observer pattern

Es un patrón de diseño de software en el que un objeto, llamado sujeto, mantiene una lista de sus dependientes, llamados observadores, y les notifica automáticamente cualquier cambio de estado, generalmente llamando a uno de sus métodos.[8]

Singleton pattern

Es un patrón de diseño que permite restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto. Su intención consiste en garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella.[9]

Factory Method

Es un patrón de diseño que define una interfaz para la creación de un objeto, este patrón deja que las subclases decidan que clase van a instanciar. [10]

Single source of truth

Es la práctica de estructurar el almacenamiento de datos de una aplicación de manera que cada dato se almacene exactamente una vez. Cualquier posible vínculo con este datos (posiblemente en otras áreas de la aplicación) es solo por referencia.[11]

Cada vez que almacena o recupera datos, debe provenir de un solo lugar. Si lo recuperas dos o más veces, cada instancia de los datos recuperados debe ser exactamente la misma.

3.2.2. Patrones de arquitectura

Programación por capas

La programación por capas es un patrón arquitectónico en el que el objetivo primordial es la separación (desacomplamiento) de las partes que componen una aplicación. Cada capa representa una agrupación de módulos que ofrecen un conjunto cohesivo de servicios y se crean para interaccionar según una relación de orden estricto. [12]

Este patrón es uno de los más usados en el desarrollo de aplicaciones.

Repository pattern

El repository pattern está íntimamente relacionado con el acceso a datos y nos permite tener una abstracción de la implementación de acceso a datos en nuestras aplicaciones, de modo que nuestra lógica de negocio no conozca ni esté acoplada a la fuente de datos.[13]

3.3. Análisis Estáticos y no estáticos

Además de cumplir con los requisitos funcionales, es importante asegurarse de que no existan problemas estructurales en el código de la aplicación. Ya que ésto podría llevar a tener que refactorizar el código.

El **análisis estático** de software es un tipo de análisis automatizado de software que se realiza sin ejecutar el programa. Este análisis se realiza sobre el código fuente de la aplicación utilizando herramientas creadas para ello.[14]

El Análisis no estático o dinámico de software supone la ejecución del programa y observar su comportamiento. Estos análisis deben ejecutarse con los suficientes casos de prueba como para producir un comportamiento interesante.

3.3.1. Lint

Android Studio ofrece una herramienta de análisis estático de código denominada **lint**. Esta herramienta comprueba los archivos de origen del proyecto en busca de posibles errores y para realizar mejoras relacionadas con la precisión, la seguridad, el rendimiento, la usabilidad, la accesibilidad y la internacionalización. [15]

Lint tiene un archivo de configuración denominado lint.xml que se puede usar para especificar cualquier comprobación que desees excluir y para personalizar los niveles de severidad. Además permite añadir nuevas reglas para la comprobación de código.

En el siguiente diagrama se muestra la forma en que la herramienta lint procesa los archivos de origen de la aplicación:

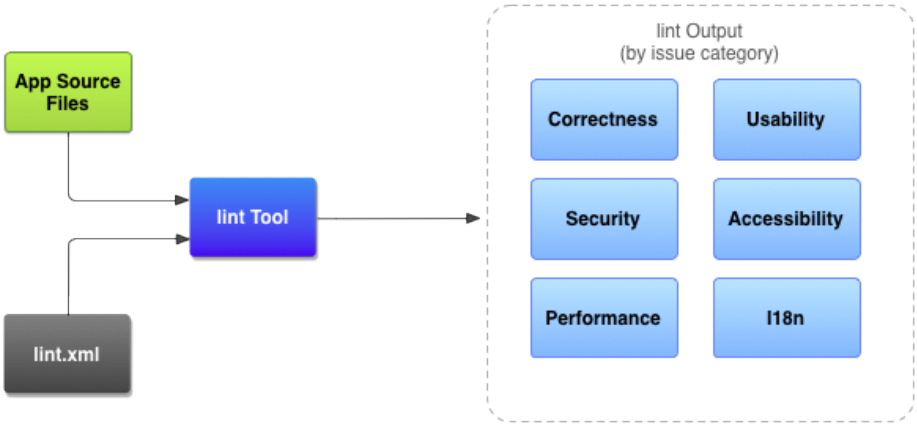

Figura 3.1: Procesamiento de los archivos de origen.

3.4. Android Architecture Components

Android fue presentado al mundo en 2005, y hasta ahora, había ignorando un área muy importante de la plataforma: un patrón de arquitectura estándar, capaz de manejar las peculiaridades de la plataforma y lo suficientemente simple como para ser entendido y adoptado por el desarrollador promedio.

En el Google I/O '17, el equipo de Android finalmente ha decidido abordar este problema anunciando una recomendación oficial para una arquitectura de aplicaciones de Android y proporcionando los componentes básicos para implementarla: **Android Architecture Components**[16]

Android Architecture Components es una colección de bibliotecas que ayudan a diseñar aplicaciones robustas, *'testeables'* y fáciles de mantener.[17] Los principales elementos que componen la Arquitectura de Componentes son:

- Lifecycle-Aware Components
- ViewModel
- LiveData
- Room

3.4.1. Lifecycle-Aware Components

Los **Lifecycle-Aware Components** realizan acciones en respuesta a cambios de estado en el ciclo de vida de otro componente, como *activities* y *fragments*. El paquete android.arch.lifecycle proporciona las clases e interfaces que permiten crear Lifecycle-Aware Components, las más importantes son:

- Lifecycle: es una clase que contiene la información sobre el estado del ciclo de vida de un componente (como una actividad o un fragmento) y permite que otros objetos observen este estado.
- LifecycleOwner: es una interfaz de método que denota que una clase tiene un ciclo de vida.

3.4.2. ViewModel

La clase **ViewModel** está diseñada para almacenar y administrar datos relacionados con la interfaz de usuario de manera consciente sobre el ciclo de vida. La clase ViewModel permite que los datos sobrevivan a cambios de configuración, como las rotaciones de pantalla.

Los ViewModels generalmente están asociados con un controlador de UI para el que proporcionan datos. Lo hacen usando las clases Lifecycle y LifecycleOwner explicadas anteriormente.

3.4.3. LiveData

Es una clase que nos proporciona la capacidad de almacenar datos y que éstos sean observados a través de los cambios en el ciclo de vida. La clase nos proporciona una colección de métodos que pueden utilizarse para gestionar cualquier *Observable* que esté en su lugar para los datos que se mantienen dentro de la clase.[18]

3.4.4. Room

La implementación de la base de datos SQLite que Android nos proporciona, exige escribir mucho código *boilerplate* y es muy dada a errores de sentencias SQL en tiempo de ejecución.

ROOM es una biblioteca de mapeo de objetos SQLite capaz de persistir *POJOs* de Java, de convertir consultas directamente a objetos, verificar errores en tiempo de compilación y producir datos *observables* de LiveData a partir de los resultados de las consultas. Posteriormente en su implementación se explicará con más detalle cómo funciona y cuál es su historia. [19]

Capítulo 4

Proceso para la refactorización y migración de aplicaciones

El proceso de refactorización es **iterativo** puesto que se puede repetir tantas veces como se desee realizar una nueva refactorización e **incremental** ya que consta de diversas etapas de desarrollo en cada incremento [20].Está formado por 4 fases, en la siguiente figura se ilustra como se organizan estas fases en cuanto a su desarrollo y en los apartados siguientes se profundizará en cada una de las fases:

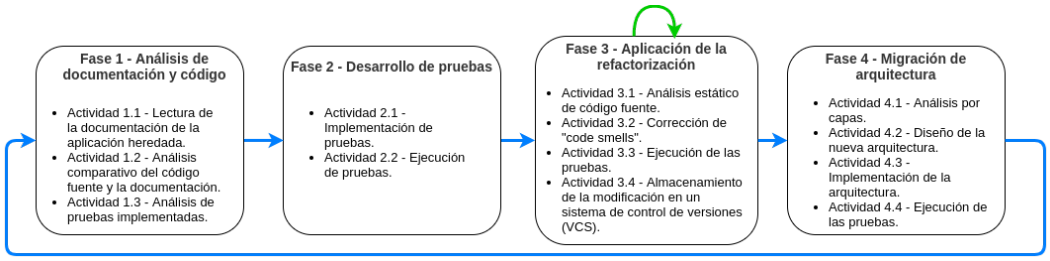

Figura 4.1: Esquema con las fases del proceso de refactorización

4.1. Fase 1 - Análisis de documentación y código

La primera fase consiste en documentarse sobre el proyecto que se va a refactorizar, de ésta forma podremos abordar la refactorización de manera

precisa y eficaz.

4.1.1. Actividad 1.1 - Lectura de la documentación de la aplicación heredada.

Si la aplicación tiene documentación, ésta será la primera actividad que se debe realizar. Se debe hacer una lectura exhaustiva de la documentación con el objetivo de entender cómo funciona la aplicación en detalle.

Es importante que sepas responder a las siguientes preguntas después de realizar la lectura:

- ¿Para que sirve la aplicación?
- ¿Qué pantallas tiene la aplicación?
- ¿Sigue una estructura bien definida?
- ¿Se ha desarrollado siguiendo un razonamiento correcto?
- ¿Cúales son los casos de uso de sus usuarios?
- ¿Tiene mapas de navegación?

Entrada

- Documentación de la aplicación heredada.
- Aplicación heredada.

Salida

- Notas significativas tomadas sobre el proyecto.

4.1.2. Actividad 1.2 - Análisis comparativo del código fuente y la documentación.

Utilizando las anotaciones sobre el proyecto, se comparan con el código fuente heredado con el objetivo de verificar si coinciden y no caen en incongruencias.

Es posible que la documentación no se ajuste a la realidad de la aplicación, por ello es importante verificar todas las partes de la aplicación y las relaciones entre capas en caso de que estén definidas.

Una buena estrategia es comenzar por la capa de datos e ir comprobando desde dónde se referencian las operaciones sobre los datos. Una vez comprobada la capa de datos ir a la siguiente capa y ver que los métodos que hay en esa capa tienen lógica y están bien implementados.

Una recomendación para realizar la comparación de forma efectiva es responder las siguientes cuestiones:

- ¿Se utiliza algún patrón de presentación? En caso afirmativo, ¿Esta implementado de forma correcta?
- ¿Existe desacoplamiento entre capas en el código fuente?
- ¿Tenemos algún patrón de arquitectura? En caso afirmativo, ¿Está implementado de forma correcta?
- ¿Se corresponden las funcionalidades explicadas en la documentación con la implementación?
- ¿Tenemos un mapa de navegación? En ese caso, ¿Se corresponde con el código fuente?
- Si existe diagrama de clases, ¿Las clases incluidas en el mismo se corresponden con el código?¿Las relaciones entre clases se corresponden con la realidad?

Entrada

- Notas significativas tomadas sobre el proyecto en la actividad anterior.
- Código fuente de la aplicación.

Salida

- Detección y anotación de futuras refactorizaciones.

4.1.3. Actividad 1.3 - Análisis de pruebas implementadas.

En esta actividad se deberán analizar las diferentes pruebas que haya implementadas en el proyecto.

Es importante saber que normalmente las pruebas se encuentran en los paquetes androidTest y test. También se debe tener en cuenta que hay diferentes tipos de test, por lo que hay que identificar de qué tipo es cada test y cómo se han elaborado ya que existen diferentes frameworks y liberías para su desarrollo.

Entrada

- Código fuente de la aplicación.

Salida

- Detección y comprobación de las pruebas implementadas.
- Identificar pruebas no implementadas.

4.2. Fase 2 - Desarrollo de pruebas

Desarrollando pruebas en la aplicación podemos verificar su corrección, funcionamiento y usabilidad antes de lanzarla públicamente. El desarrollo de pruebas es un paso esencial en el proceso de refactorización, por eso tenemos que ser cuidadosos en su implementación.

4.2.1. Actividad 2.1 - Implementación de pruebas.

Puesto que en la fase anterior hemos identificado que pruebas no se han implementado, en esta actividad se deben implementar el mayor número de pruebas posibles para asegurar que no surgen fallos en el proceso de refactorización.

Debemos tener en cuenta los diferentes tipos de pruebas que se pueden desarrollar en aplicaciones Android:

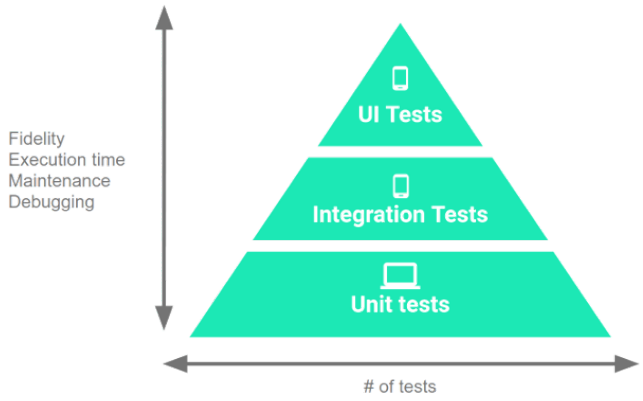

Figura 4.2: La pirámide de pruebas, muestra las tres categorías de pruebas que debes incluir en tu proyecto. [1]

Es importante implementar pruebas de los valores críticos de los diferentes métodos.

Entrada

- Código fuente de la aplicación.
- Pruebas no implementadas.

Salida

- Un amplio juego de pruebas que asegure en la medida de lo posible la refactorización.

4.2.2. Actividad 2.2 - Ejecución de pruebas.

Tras la implementación de nuevas pruebas, se debe probar su correcto desarrollo ejecutándolas y comprobando que el resultado es positivo.

Hay que comprobar también que si cambiamos los datos de las pruebas implementadas a sabiendas de que están mal, el resultado debe ser incorrecto.

Como en Android se pueden desarrollar diferentes tipos de pruebas, cada uno se ejecuta de manera diferente[1]:

- Los test unitarios pueden ejecutarse aislados del sistema por lo que su ejecución es muy rápida.
- Los tests de integración se componen de varios componentes y se ejecutan en un emulador o dispositivo real.
- Tests de integración y UI que se ejecutan en un flujo completo de trabajo sobre la UI.

Entrada

- Código fuente de la aplicación.
- Pruebas implementadas.

Salida

- Verificación de las pruebas desarrolladas en la actividad anterior.

4.3. Fase 3 - Aplicación de la refactorización

Hay que destacar que en esta fase las actividades se realizan como un ciclo, de manera que todas las actividades se deben realizar múltiples veces en conjunto antes de pasar a la siguiente fase. En el siguiente diagrama se ve el proceso:

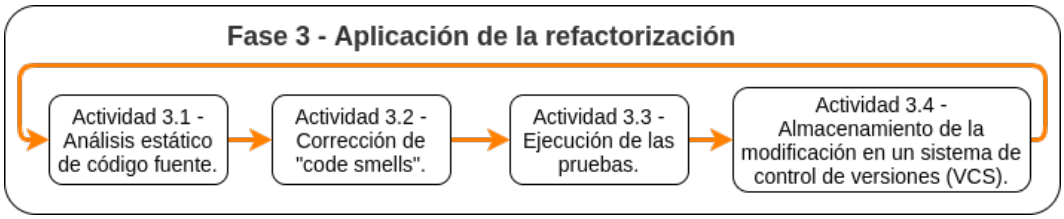

Figura 4.3: Diagrama de la fase.

Uno de los motivos para aplicar una refactorización en un proyecto es la detección de *code smells*. Los *code smells* indican deficiencias en el diseño que pueden ralentizar el desarrollo o aumentan el riesgo de errores o fallos en el futuro. En esta fase se buscarán y tratarán de eliminar estas deficiencias del proyecto.

4.3.1. Actividad 3.1 - Análisis estático de código fuente.

Utilizando herramientas de análisis estático se realizará una inspección del proyecto para obtener deficiencias en el código. En el caso de Android, el IDE Android Studio nos proporciona el analizador Lint para esta tarea.

Es recomendable acotar la inspección seleccionando solo los tipos de avisos y errores que queremos que nos detecte, de lo contrario nos arriesgamos a que nos aparezcan una gran cantidad de avisos no deseados en el proyecto. Se deben priorizar los problemas en base a los objetivos.

En el caso de que vayamos a realizar una migración en el proyecto, algo que debemos tener en cuenta es si debemos realizar esta fase antes o después de realizar la migración. La migración de una aplicación conlleva realizar grandes cambios de código y estructura en la misma, es por ello, que se presenta este dilema, ya que en el desarrollo de la migración se pueden generar otros *code smells*.

Entrada

- Código fuente de la aplicación.

- Herramienta de análisis estático.

Salida

- Listado de *code smells* en el proyecto.

4.3.2. Actividad 3.2 - Corrección de *code smells*.

Se analiza cada problema obtenido con el analizador estático en la actividad anterior y se busca la mejor solución para cada uno.

En el caso de que el nº de problemas detectados sea muy grande, lo recomendable es ordenar estos avisos por gravedad e ir resolviéndolos poco a poco.

Para aplicar una solución, debemos comprobar si existe un catálogo de soluciones para cada tipo de error, en ese caso, se recomienda aplicar la guía buscando cada error. Es posible que exista la guía pero las versiones de la plataforma no se correspondan con la guía y entonces no se debe aplicar esa solución. En caso de que no exista la guía, se debe elaborar a la vez.

Entrada

- Listado de *code smells* en el proyecto.
- Código fuente de la aplicación.

Salida

- Código fuente refactorizado sin *code smells*

4.3.3. Actividad 3.3 - Ejecución de las pruebas.

Se ejecutan las pruebas implementadas en el proyecto para comprobar que la refactorización se ha hecho de forma correcta y no se ha modificado la funcionalidad ni generado errores en la aplicación.

Hay pruebas que son tediosas de ejecutar, por ello solo debemos de ejecutarlas cuando hayamos modificado código que afecte a las mismas, pero esta decisión la debe tomar cada desarrollador en función de la cantidad de código, rendimiento del ordenador etc...

Entrada

- Código fuente de la aplicación.

Salida

- Refactorización verificada.

4.3.4. Actividad 3.4 - Almacenamiento de la modificación en un sistema de control de versiones (VCS).

Para el desarrollo de cualquier proyecto el uso de un sistema de control de versiones es esencial. El control de versiones es un sistema que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo, de modo que puedas recuperar versiones específicas más adelante.

Tras realizar un conjunto de refactorizaciones se deben registrar los cambios en un sistema de control de versiones. Consiguiendo de esta manera tener una copia por si en un futuro queremos revertir algún cambio.

Es recomendable crear una nueva rama para aplicar la refactorización y una nueva rama por cada refactorización como se expone en el siguiente diagrama:

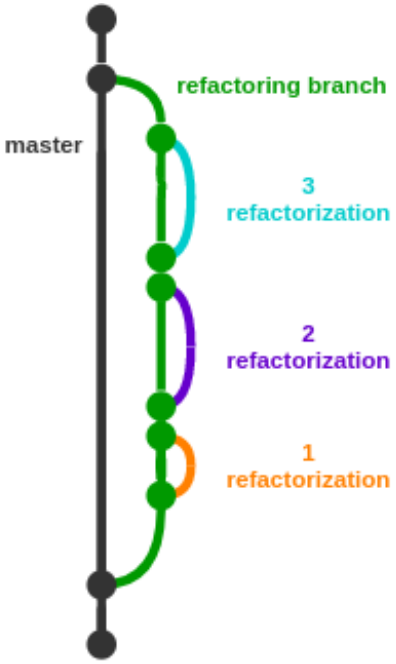

Figura 4.4: Arbol git de ejemplo para la refactorización.

Entrada

- Código fuente de la aplicación.
- Sistema de Control de versiones.

Salida

- Copia de los cambios registrada.

4.4. Fase 4 - Migración de arquitectura

En esta fase se realizará una migración de la arquitectura del proyecto con el objetivo de mejorar la estructura y organización del mismo.

4.4.1. Actividad 4.1 - Análisis por capas.

Puesto que analizar un proyecto es una tarea tediosa, lo mejor es dividir el análisis en las diferentes partes que componen el proyecto. Las diferentes partes en que podemos dividir el análisis pueden ser características de la aplicación, capas (lógica de negocios, capa de presentación y capa de datos) o cualquier otra división que estipulemos según como esté desarrollado.

Entrada

- Código fuente de la aplicación.

Salida

- Anotación de las capas del proyecto junto a detalles de su implementación.

4.4.2. Actividad 4.2 - Diseño de la nueva arquitectura.

Utilizando las anotaciones tomadas en la actividad anterior se debe diseñar la nueva arquitectura que se va a implementar en la aplicación. Una buena opción para el diseño es dividir el proyecto en capas y diseñar cada capa en profundidad.

Para diseñar la arquitectura es importante informarse de los diferentes tipos de arquitecturas que hay y decidir cual es la que encaja mejor en nuestra aplicación.

Entrada

- Código fuente de la aplicación.
- Anotación de las capas del proyecto junto a detalles de su implementación.

Salida

- Diseño de la nueva arquitectura de la aplicación.

4.4.3. Actividad 4.3 - Implementación de la arquitectura.

Puesto que ya tenemos diseñada la nueva arquitectura de la aplicación, en ésta actividad se realizará la implementación de manera ordenada y meticulosa. En este punto es muy importante tener claro cómo se va a implementar la arquitectura y dividir la implementación en partes que faciliten el proceso.

Entrada

- Código fuente de la aplicación.
- Diseño de la nueva arquitectura de la aplicación

Salida

- Aplicación refactorizada.

4.4.4. Actividad 4.4 - Ejecución de las pruebas.

En el caso de que se hubiera desarrollado un juego de pruebas, se ejecutan para comprobar que la migración se ha hecho de forma correcta y no se ha modificado la funcionalidad ni generado errores en la aplicación.

Entrada

- Código fuente de la aplicación.

Salida

- Migración verificada.

Capítulo 5

Desarrollo del proceso de refactorización

5.1. Fase 1 - Análisis de documentación y código

Normalmente los proyectos van acompañados de una documentación, en ese caso se debe realizar un análisis de la misma junto al código fuente.

5.1.1. Actividad 1.1 - Lectura de la documentación de la aplicación heredada

Puesto que la aplicación es heredada y tiene documentación, lo primero ha sido realizar una lectura exhaustiva de la misma. Tras ello, se han elaborado 2 documentos con las ideas principales del proyecto, a continuación se expone un resumen de los documentos elaborados:

1.Funcionalidad del proyecto.

Este documento se elaboró para entender cómo funcionaba el proyecto y qué utilidad tenía el mismo. Siguiendo la estructura explicada en el documento elaborado posteriormente tenemos los siguientes elementos:

- Repositorios OpenData:

  Ofrecen datos que se van a utilizar para resolver las peticiones realizadas por los usuarios.

- Página Web:

  Su objetivo es servir a la plataforma móvil; la web extraerá datos de los repositorios (nuestros Data Sources), realizará el mapeado de datos del repositorio utilizando el esquema indicado por el usuario y finalmente almacenará el mapeado. También permitirá crear nuevos esquemas, mapeados, categorías y añadir repositorios.

- Aplicación Android:

  La aplicación Android permitirá a los usuarios crear aplicaciones con los datos extraídos de Open Data de forma sencilla independientemente de sus conocimientos (no es necesario que sepan que es Open Data o saber programación). Los usuarios podrán realizar las siguientes operaciones desde la aplicación Android:

Figura 5.1: Casos de uso de la aplicación Android.

El proceso completo de creación de aplicaciones desde la app Android es el siguiente:

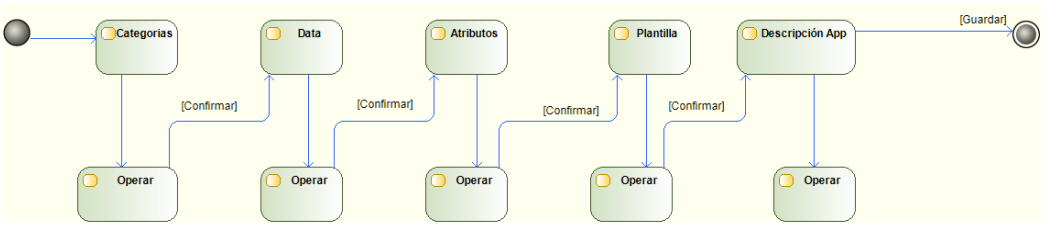

Figura 5.2: Pasos para la creación de apps.

- Base de datos utilizando FireBase:

  Firebase es un proveedor de servicios en la nube del cual se utilizan sus componentes de autentificación y base de datos. La aplicación se sincronizará con Firebase para traer o subir las aplicaciones que se tienen y gracias a la identidad federada lo único que hará falta para descargarse esas aplicaciones en otros dispositivos móviles será tener una cuenta en alguno de los proveedores registrados.

2.Estructura del proyecto.

En el siguiente documento se realizan anotaciones acerca de la estructura del proyecto:

El proyecto se divide principalmente en 4 partes:

- Repositorios OpenData.
- Página Web.
- Aplicación Android.
- Base de datos utilizando FireBase.

La **página Web** sirve para que los usuario con conocimientos adecuados puedan relacionar conceptos, añadir esquemas, repositorios, etc... Su objetivo es servir a la plataforma móvil. Se ha elaborado usando las siguientes tecnologías:

Backend:

- PHP y Javascript como lenguajes de programación.

- Symfony como framework para la optimización del desarrollo en la parte del servidor.
- OpenShift para garantizar la disponibilidad y el escalado.
- Arquitectura basada en REST y mediante ficheros JSON.

Frontend:

- JQuery UI como framework que posee un robusto y completo set para la realización de la interfaz usuario web.
- Bootstrap como framework para el diseño de la interfaz visual gracias a los componentes HTML y CSS y plugins de jQuery que proporciona.

Además se ha usado Apache para el servidor web y PostgreSQL para la gestión de la base de datos.

La **aplicación Android** se divide principalmente en 3 componentes:

1. App Builder: Es el constructor de aplicaciones, es un subsistema que muestra una interfaz basada en cinco pasos, en cada paso se recogen datos para finalmente almacenar la descripción de la aplicación. (Explicado con más detalle en el documento de funcionalidad).
2. App Repository: Se ocupa de almacenar las aplicaciones creadas en la plataforma móvil.
3. App Execution Engine: Es el motor que se encarga de interpretar la descripción de la aplicación y darle forma.

Patrones de arquitectura Android

La aplicación se ha realizado utilizando el patrón de diseño MVP, por ello, tenemos 3 paquetes que forman el patrón:

- model: Provee los datos que se mostrarán en la vista. **Lógica de acceso a datos.**

- presenters: Actúa como intermediario entre la vista y el modelo. Recupera los datos del modelo y devuelve los datos formateados a la vista. También decide qué pasa cuando se interactúa con la vista. **Lógica de negocio.**
- views: Normalmente implementada por una Activity, contendrá una referencia al presenter. La única cosa que hará la vista es llamar al método del presenter cada vez que haya una acción en la interfaz, como un click de botón. **Lógica de presentación.**

A continuación se explica qué patrones se han utilizado en cada capa:

Lógica de acceso a datos:

Para la lógica de acceso a datos se han utilizado varios patrones:

Repository Pattern: Permite tener una abstracción de la implementación de acceso a datos en nuestra aplicación, de modo que nuestra lógica de negocio no conozca ni esté acoplada a la fuente de datos.

El patrón repositorio lo tenemos en el paquete repositories, la clase AppRepository y SchemaRepository extienden de Repository. Repository se comunica con DbService y crea una instancia de DataSource que es una interfaz de AppRepository y SchemaRepository. En el siguiente diagrama podemos ver cómo se utiliza de forma clara:

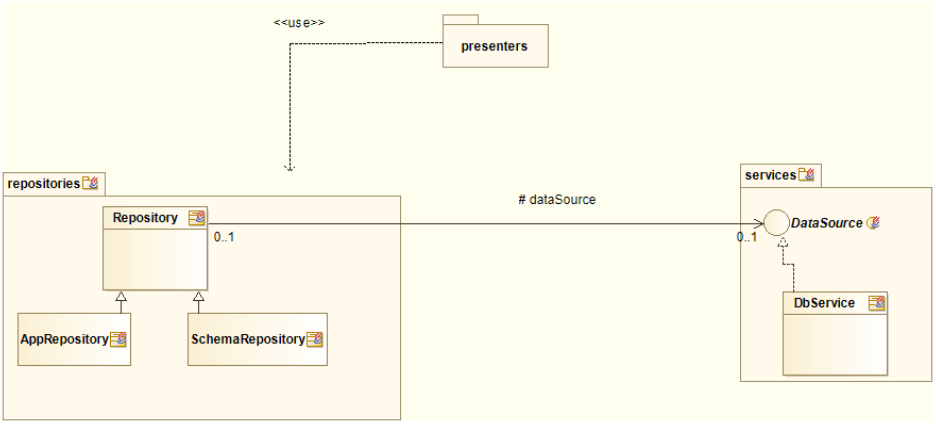

Figura 5.3: Aplicación del patrón repositorio en el proyecto.

Patrón DAO: Con fin de mejorar el código se añade una capa de

abstracción para el acceso a datos que evita que hagamos llamadas en el código específico de la base de datos. Para ello se ha utilizado el ORM GreenDAO que facilita esta tarea. Todo el código generado por GreenDAO para esta tarea está almacenado en el paquete dao.

Patrón DTO: Data Transfer Object es un objeto que transporta datos entre procesos. En nuestra aplicación la comunicación entre procesos es usualmente realizada mediante interfaces remotas (ej. Servicios Web), donde cada llamada es una operación costosa. Como la mayor parte del costo de cada llamada está relacionado con el tiempo round-trip entre el cliente y servidor, una forma de reducir el número de llamadas es usando un objeto (el DTO) que agrega los datos que habrían sido transferidos por cada llamada, pero que son entregados en una sola invocación.

Arquitectura REST: La comunicación con la API REST se realiza desde la clase RestService utilizando la librería RetroFit junto con la librería GSON para realizar la conversión JSON-Java. Desde RestService se obtiene el modelo de datos de la web a AbstractModel (Es un modelo con tres valores fijos (name, geolocation, id) para ser accedidos de forma directa, que generalmente van a estar presentes sea cual sea el esquema). También se realizan las llamadas GET a la API utilizando la clase OpenDataService.

Desacoplamiento de los esquemas: En la aplicación tenemos una colección de esquemas, cada uno de ellos distintos entre sí, por ello no se ha creado un POJO por cada esquema (no es viable). La solución que se ha utilizado ha sido crear un AbstractModel con los atributos fijo que todos los esquemas deben tener y además tener una HashTable en esta clase con atributos variables (SchemaValue).

Como se puede ver la lógica de acceso a datos es bastante compleja, lo único destacable que falta por explicar es la clase BaseEntity, esta clase almacena el estado de la aplicación (Las categorías seleccionadas, los atributos seleccionados, los marcados y las posiciones que se eligieron, los repositorios, absolutamente todo). Para que ésta clase sea accesible desde toda la aplicación se ha creado la clase DataContent, esta clase es una instancia Singleton que almacena BaseEntity; además recupera los datos que se muestran, como pueden ser todas las categorías o el modelo abstracto de datos, es decir, la aplicación ejecutada.

Lógica de negocios:

La lógica de negocio se realiza desde las clases almacenadas en el paquete presenters. MainPresenter: Tiene la lógica de todas las acciones que realiza el usuario desde la vista principal y el navigationDrawer. Además avisa a la vista con los cambios realizados. CategoriesPresenter: Se encarga de validar la selección de categorías. DataPresenter: Contiene la lógica de la pantalla de elección de datos.

Lógica de presentación:

La lógica de presentación está compuesta por las interfaces que se encuentran en el paquete views junto con los Activities y Fragments del paquete activities. La único que hacen las vistas es llamar al método del presenter cada vez que haya una acción en la interfaz, como un click de botón.

Los documentos anteriores se han elaborado a modo de resumen, para comprender el proyecto en profundidad. Tras la lectura debemos saber responder a las siguientes preguntas:

- ¿Para que sirve la aplicación? La aplicación permite que usuarios sin conocimientos en programación o con ellos, pero sin necesidad de usarlos, puedan crear sus propias aplicaciones con datos extraídos de repositorios opendata gracias a una serie de pasos guiados.
- ¿Qué pantallas tiene la aplicación?
    1. Pantalla principal.
    2. Selección de categorías.
    3. Selección de repositorio.
    4. Selección de atributos.
    5. Selección de nombre.
    6. Pantalla de la aplicación generada.
- ¿Sigue una estructura bien definida? Si, el proyecto se desarrolló en base a patrones que permitían estructurar el código.
- ¿Se ha desarrollado siguiendo un razonamiento correcto? Sí, considero que la aplicación se planteó de forma razonable, gracias a que la generación de aplicaciones se realiza paso a paso y es intuitiva.
- ¿Cúales son los casos de uso de sus usuarios?

Figura 5.4: Casos de uso de la aplicación android.

- ¿Tiene mapas de navegación? No, no se ha incluido ningún mapa de navegación en la aplicación.

5.1.2. Actividad 1.2 - Análisis comparativo del código fuente y la documentación

Tras la lectura de la documentación, se ha hecho un análisis del código fuente comparándolo con la documentación del proyecto.

- ¿Se utiliza algún patrón de presentación? En caso afirmativo, ¿Está implementado de forma correcta? La aplicación se ha desarrollado utilizando el patrón MVP. No está implementado correctamente puesto que no existe desacoplamiento entre las capas.
- ¿Existe desacoplamiento entre capas en el código fuente? Se ha desarrollado pensando en el desacoplamiento de capas, pero lo cierto es que las capas están acopladas.
- ¿Tenemos algún patrón de arquitectura? En caso afirmativo, ¿Está implementado de forma correcta? Se utiliza la programación por capas, pero no se ha implementado de forma correcta puesto que los métodos no están bien situados en las capas.

- ¿Se corresponden las funcionalidades explicadas en la documentación con la implementación? La mayoría de funcionalidades si se corresponden, pero el almacenamiento remoto por ejemplo no está bien implementado.
- ¿Tenemos un mapa de navegación? En ese caso, ¿Se corresponde con el código fuente? No existe mapa de navegación en la documentación.
- Si existe diagrama de clases, ¿Las clases incluidas en el mismo se corresponden con el código?¿Las relaciones entre clases se corresponden con la realidad? Existen varios diagramas de clases pero no se ajustan al código.

Tras la respuesta anteriores y la comparativa con el código se ha llegado a la siguiente conclusión:

- El patrón repository no se ha aplicado de forma correcta puesto que existen referencias a la base de datos desde los presenters y activities sin utilizar los repositorios.
- La implementación de firebase es incorrecta, ya que no realiza el registro y posterior logueado de forma correcta.
- La comprensión del código es muy dificultosa, puesto que está muy desorganizado.
- Los paquetes y código fuente están muy desorganizados.

5.1.3. Actividad 1.3 - Análisis de pruebas implementadas

El proyecto no tiene implementadas pruebas de ningún tipo, por lo que es difícil su mantenimiento y adicción de nuevas características.

5.2. Fase 2 - Desarrollo de pruebas

Puesto que posteriormente se va a realizar una migración de la arquitectura (uno de los objetivos de la migración es facilitar el desarrollo de pruebas) y esto conlleva la modificación de una gran parte del código se ha decidido no desarrollar un juego de pruebas en el proyecto antes de realizar la misma.

Posteriormente no se han desarrollado un juego de pruebas por falta de tiempo, ésto es una de las ampliaciones para el futuro en el proyecto. Ya que el desarrollo de pruebas es una parte esencial que no se le debe *dejar a un lado*.

5.3. Fase 3 - Aplicación de la refactorización

Se han realizado dos grandes refactorizaciones en la aplicación, la primera de más bajo nivel utilizando el analizador estático Lint y la segunda realizando una migración de arquitectura en la aplicación.

5.3.1. Actividad 3.1 - Análisis estático de código fuente

Anteriormente se ha explicado qué es un análisis estático y qué es la herramienta Lint, a continuación se expone cómo se ha utilizado y cuáles han sido los resultados tras su uso en el proyecto. El proceso de refactorización usando Lint tiene varias fases:

1. Inspección automatizada del código.
2. Registro del resultado de la inspección.
3. Corrección manual de los errores y avisos.

Inspección

Lint permite seleccionar qué patrones va a buscar en la aplicación, además se pueden ordenar en función de la Seguridad, Rendimiento etc... Todo ello lo podemos configurar desde la interfaz que se proporciona dentro de la plataforma Android Studio:

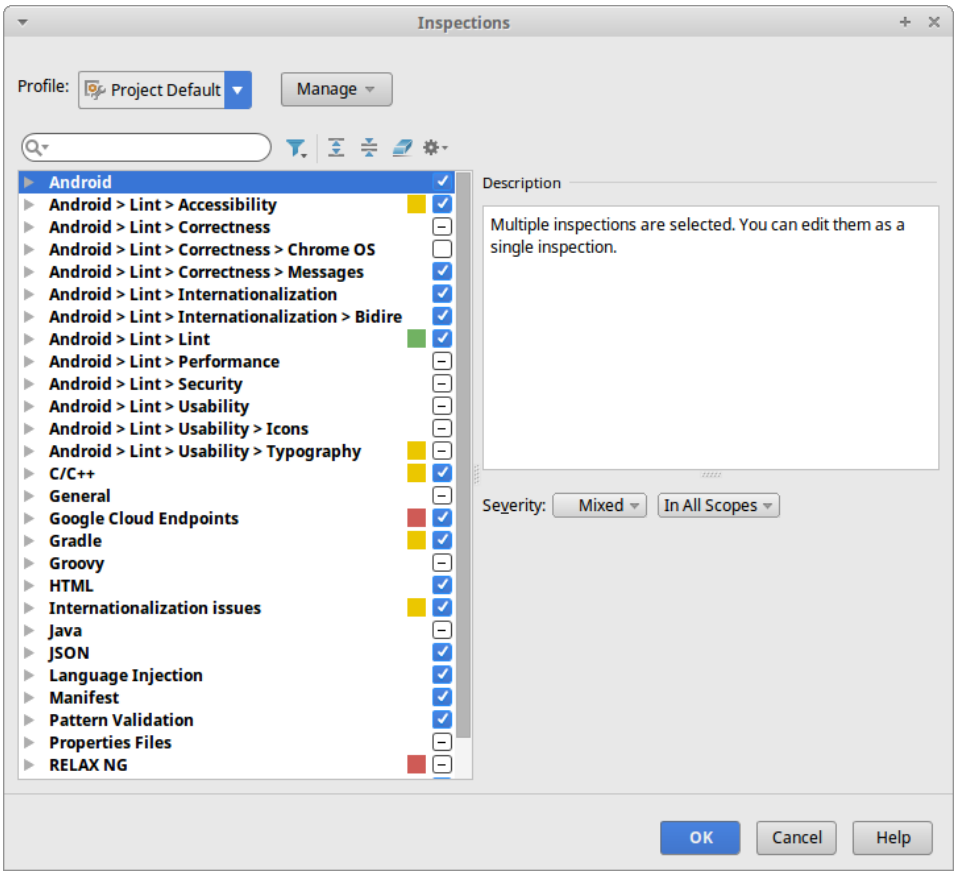

Figura 5.5: Configurar inspección.

Para la inspección con Lint se han seleccionado los siguientes patrones:

- Accesibility: Al integrar componentes y servicios accesibles, puede mejorar la usabilidad de su aplicación, especialmente para usuarios con discapacidades, Lint permite encontrar fallos de accesibilidad.
- Correctness: Comprueba que las diferentes etiquetas, propiedades y el código implementado se ha hecho de forma correcta.
- Internationalization: Busca problemas que compliquen la traducción de la aplicación o la utilización en otros países.
- Lint: Puesto que se pueden marcar avisos como ïnvisibles.[a] Lint, en caso de que en alguna ocasión se haga de forma incorrecta Lint detecta estos problemas.

- Performance: Busca problemas relacionados con el rendimiento de la aplicación.
- Security: Lint puede ayudar a detectar problemas de seguridad en la aplicación que analicemos.
- Usability: Detecta problemas relacionados con la usabilidad de la aplicación.
- General: Busca problemas en los comentarios, referencias, sintaxis etc...
- Gradle: Detecta errores de la herramienta de compilación Gradle.
- Groovy: Permite la búsqueda de avisos o errores en el lenguaje Groovy utilizado en el proyecto (Gradle).
- Internationalization issues: Busca problemas relacionados con la codificación de caracteres en otros idiomas.
- Java: Lint puede detectar problemas específicos de Java.
- JSON: Los archivos JSON son muy utilizados en las aplicaciones Android, por ello, Lint permite detectar problemas en los mismos.
- Language Injection: Puesto que en el proyecto es posible que haya inyección de dependencias, Lint permite detectar problemas en las mismas.
- Manifest: Buscar errores en el archivo Manifest del proyecto.
- Pattern Validation: Comprueba que no hay avisos o errores con el uso de la etiqueta @Pattern.
- Properties Files: Detecta problemas con los archivos de propiedades de Android.
- Spelling: Busca palabras que no están en el diccionario y te avisa.
- XML: Detecta errores en los archivos XML del proyecto.

Resultado

Para pasar a la siguiente fase de corrección de errores y advertencias, lo más coherente es ordenarlos por gravedad, de tal manera que primero se corrijan los errores y después las advertencias. Android Studio permite ordenarlos de esta manera con la opción "Group by Severity". Tras la inspección

de código, el resultado ordenado por gravedad que arroja Lint ha sido el siguiente:

Análisis de Lint	
Tipo de patrón	Nº de de errores
Android	207 errores
Android Lint: Correctness	10 errores
Javadoc issues	2 errores
XML	147 errores

Cuadro 5.1: Errores tras la inspección de Lint.

Análisis de Lint	
Tipo de patrón	Nº de Avisos
Android	5 avisos
Android Lint: Accessibility	17 avisos
Android Lint: Correctness	13 avisos
Android Lint: Internationalization	23 avisos
Android Lint: Performance	39 avisos
Android Lint: Security	1 avisos
Android Lint: Usability	17 avisos
Class structure	9 avisos
Code maturity issues	7 avisos
Code style issues	22 avisos
Compiler issues	4 avisos
Control flow issues	4 avisos
Data flow issues	10 avisos
Declaration redundancy	242 avisos
Error handling	4 avisos
General	25 avisos
Imports	43 avisos
Java 7	15 avisos
Java language level migration aids	1 avisos
Javadoc issues	52 avisos
Numeric issues	24 avisos
Performance issues	1 avisos
Probable bugs	23 avisos
Verbose or redundant code constructs	19 avisos
XML	63 avisos

Cuadro 5.2: Resultado de la inspección.

5.3.2. Actividad 3.2 - Corrección de *code smells*

La corrección de errores se realiza manualmente a partir de la localización proporcionada en los resultados de Lint. Puesto que no había un catálogo de soluciones, se ha elaborado en su desarrollo, este catalogo se ha añadido cómo anexo en el proyecto. A continuación se exponen varios ejemplos significativos de los errores y avisos dados por Lint, la ubicación de los mismos y la solución que se le ha dado:

Errores

Error 4 - AndroidManifest.xml

- Unresolved package 'auth'

Refactorización 4 - AndroidManifest.xml

Eliminar Módulo auth - Este módulo era el culpable de una gran cantidad de los errores que aparecían en Lint, este módulo estaba incluido en el proyecto en la versión original que incluía FireBase.

Android Accesibility

Aviso 2 - layout/app_bar_main.xml

- Missing 'contentDescription' attribute on image

```
<ImageView
   android:layout_width="200sp"
   android:layout_height="125sp"
   android:src="@drawable/logo_panel2"/>
```

Refactorización 2 - layout/app_bar_main.xml

- Añadir el atributo contentDescription con una descripción de la imagen.

```
<ImageView
   android:layout_width="200sp"
   android:layout_height="125sp"
   android:contentDescription="Panel with the logo of the app"
   android:src="@drawable/logo_panel2"/>
```

Android Correctness

Aviso 12 - app/build.gradle

- A newer version of com.google.android.gms:play-services-location 11.0.4 is available: 11.6.0
- A newer version of com.google.android.gms:play-services-places 11.0.4 is available: 11.6.0
- A newer version of com.google.android.gms:play-services-maps 11.0.4 is available: 11.6.0

Refactorización 12 - app/build.gradle

- Cambiar la versión de las dependencias a la 11.6.0.

Aviso 15 - values/styles.xml

- Should use “sp” instead of “dp” for text sizes.

```
<item name="android:textSize">16dp</item>
```

Refactorización 15 - values/styles.xml

- Cambiar unidad del tamaño de las letras a sp

```
<item name="android:textSize">16sp</item>
```

Android Lint: Internationalization

Aviso 17 - activity_categories.xml, activity_main_drawer.xml, activity_maps.xml, app_bar_main.xml, content_main.xml, appgen_list.xml, layout/nav_header_main.xml

- Hardcoded string "***", should use '@string' resource

Refactorización 17 - activity_categories.xml, activity_main_drawer.xml, activity_maps.xml, layout/app_bar_main.xml, layout/content_main.xml, menu/menu_appgen_list.xml, layout/nav_header_main.xml

- Eliminar el color del style.xml ya que ya estaba en las librerías de soporte de google.

Android Lint: Performance

Aviso 22 - AppgenActivity.java

This Asynctask class should be static or leaks might occur (anonymous android.os.AsyncTask)

Refactorización 22 - AppgenActivity.java

Aquí tenemos un ejemplo de refactorización que no se va a corregir en esta fase puesto que se va a ver afectado en la migración que se va a realizar posteriormente. Todos los asynctask del proyecto son anónimos y están creados en métodos, lo correcto es que sean clases estáticas.

Class structure

Aviso 28 - AppgenActivity.java, CategoriesActivity, CategoriesPresenter, DataPresenter, ListLayoutFragment, MainActivity

- Field can be converted to a local variable

Refactorización 28 - AppgenActivity.java, CategoriesActivity, CategoriesPresenter, DataPresenter, ListLayoutFragment, MainActivity

- Convertir las variables en variables de tipo local, ya que no necesitan ser atributos de la clase.

Java language level migration aids

Aviso 48 - ListFields.java

- 'if' statement replaceable with 'switch' statement

```
if(field.getType().equals("title"))
        field_title = field.getField();
else if(field.getType().equals("geolocation"))
        field_geolocation = field.getField();
else if(field.getType().equals("pkey"))
        field_id = field.getField();
else
        ordinary_fields.add(field);
```

Refactorización 48 - ListFields.java

- Cambiar el if por un switch:

```
switch (field.getType()) {
   case "title":
       field_title = field.getField();
       break;
   case "geolocation":
       field_geolocation = field.getField();
       break;
   case "pkey":
       field_id = field.getField();
       break;
```

```
    default:
        ordinary_fields.add(field);
        break;
}
```

Probable Bugs

Aviso 54 - ConnectionUtilities, DataActivity, DataPresenter.java

- Method invocation '***' may produce 'java.lang.NullPointerException'

Refactorización 54 - ConnectionUtilities.java, DataActivity.java, DataPresenter.java

- Añadir comprobaciones antes de utilizar los métodos que puedan producir 'java.lang.NullPointerException'

Puesto que había mas de 60 avisos y errores que corregir, los ejemplos que se han puesto son algunos de los más destacables de toda la refactorización.

5.3.3. Actividad 3.3 - Ejecución de las pruebas

Si se han desarrollado pruebas, tras la realización de la refactorización se deben ejecutar para comprobar que los cambios realizados en el proyecto no producen ningún error o comportamiento no deseado.

5.3.4. Actividad 3.4 - Almacenamiento de la modificación en un sistema de control de versiones (VCS).

Para asegurar los cambios realizados en la refactorización de la aplicación se ha utilizado Git como sistema de control de versiones junto Bitbucket, una plataforma colaborativa para alojar proyectos.

Se ha creado un repositorio privado para alojar el proyecto, por ello, solo se puede acceder al proyecto mediante invitación. La dirección del repositorio es https://campanoon@bitbucket.org/campanoon/yourinstapp.git

Tras realizar refactorizaciones se realiza un *commit* y *push* al repositorio para guardar las modificaciones. A continuación se expone el log del repositorio donde se ven muchos de los cambios realizados:

Author	Commit	Message	Date
Jose Manuel C...	5787330	Packages reestructurated.	2018-01-11
Jose Manuel C...	5fa0502	Some tests created. package ui created.	2018-01-11
Jose Manuel C...	49641a1	GPSTracker deleted. Adding Espresso Test.	2017-12-20
Jose Manuel C...	8460daf	BaseEntity Deleted Don't work Generate App baseentity	2017-12-15
Jose Manuel C...	99555e9	Dependencies updated	2017-12-01
Jose Manuel C...	bc0825d	Update build.gradle	2017-11-30
Jose Manuel C...	d4523ef	AppGen does not depend on DataContent	2017-11-30
Jose Manuel C...	c2bdc33	LiveData repos working.	2017-11-30
Jose Manuel C...	fe812e8	All ViewModels working	2017-11-27
Jose Manuel C...	a1eba33	CategoriesViewModel finished.	2017-11-27
Jose Manuel C...	82a12b0	ROOM finished. ActivityViewModel working. Observer getListApp working.	2017-11-27
Jose Manuel C...	ed47f94	All working, presenters call repository	2017-11-23
Jose Manuel C...	26461e6	All working with ROOM	2017-11-23
Jose Manuel C...	61f5a53	Deleted RootActivity and old DAOs Created class Error All works except Genera...	2017-11-23
Jose Manuel C...	984ac17	ROOM finished, deleted DBService, DataSource. Moved mayor of RootActivity ...	2017-11-22
Jose Manuel C...	12e281a	Starting ROOM	2017-11-20
Jose Manuel C...	2899f7b	Refactoring Lint Terminated.	2017-11-16

Figura 5.6: Log con los commits realizados en Bitbucket.

5.4. Fase 4 - Migración de arquitectura

La aplicación heredada se construyó con una arquitectura Model-View-Presenter y utilizando el patrón repository. A continuación se expone el desarrollo de la migración de arquitectura de la aplicación heredada:

5.4.1. Actividad 4.1 - Análisis por capas

Para realizar la refactorización, se ha optado por realizar un análisis por capas, de esta manera se reduce la complejidad de la misma. "Divide y vencerás":

- Capa de datos
- Capa de dominio
- Capa de presentación

Capa de datos

Es la capa donde residen los datos y es la encargada de acceder a los mismos. Reciben solicitudes de almacenamiento o recuperación de información desde la capa de dominio.

Almacenamiento local

El almacenamiento local de datos se han implementado utilizando el ORM(Object-Relational Mapping) **GreenDAO** que facilita la implementación de operaciones sobre la base de datos SQLite. Las clases implementadas que se encargan del almacenamiento local son:

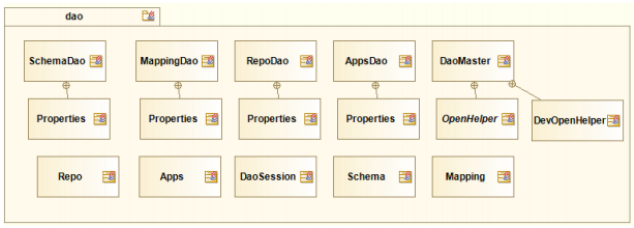

Figura 5.7: Paquete DAO del proyecto.

Las clases AppsDao, DaoMaster, DaoSession, MappingDao, SchemaDao y RepoDao que aparecen en la figura anterior sirven para hacer la traducción entre los objetos que se van a explicar a continuación y las tablas de la base de datos.

Los objetos o entidades que se almacenan localmente en la aplicación son:

Apps	
Nombre	Tipo
id	long
app_name	String
type	String
data	String

Cuadro 5.3: Clase Apps.

Éste modelo es utilizado para las aplicaciones generadas por los usuarios que se guardarán en la memoria secundaria. Se cargan las apps guardadas al inicio de la app desde DbService.java (getListApps()) y se crean nuevas apps desde BaseEntity.java (toDao()) y se almacenan desde DbService.java (saveState()) Sus campos son:

- id - Identificador de la aplicación, es autoincremental.
- app_name - Nombre de la aplicación generada.
- type - Tipo de la aplicación que se ha generado (Lista o Mapa).
- data - Almacena todos los datos de la aplicación seleccionados por el usuario en un String en forma de cadena JSON, a continuación se expone un ejemplo creado:

```
{"id":null,
"app_name":"restaurants cc",
"layout_type":"list",
"repository_id":1,
"schema_id":1,
"selected_attributes":[]}
```

Mapping	
Nombre	Tipo
id	long
schema	long
dataset	long
metadata	String

Cuadro 5.4: Clase Mapping

Permite guardar la relación entre los diferentes esquemas y los datasets. A pesar de estar en el proyecto, no se utilizaba.

- id - Identificador de la aplicación, es autoincremental.
- schema - Esquema
- dataset - Dataset
- metadata - Otros datos de los datasets.

Repo implements Comparable	
Nombre	Tipo
id	long
name	String
place	String
geo_lat	float
geo_long	float
keywords	String
timestamp	Integer

Cuadro 5.5: Clase Repo

Almacena los datos de los repositorios que hay en la app, implementa la interfaz Comparable(toString, equals, compareTo, hashcode) y se inicializan desde el RestService al acceder al fragment DATA (getRepositories()) con todos los repositorios traídos de la capa web, tiene los siguientes campos:

- Id - Identificador del repositorio en la capa web.

- Name - Nombre del repositorio.
- Place - Nombre del lugar del repositorio.
- Geo_lat - Latitud geográfica del repositorio .
- Geo_long - Longitud geográfica del repositorio.
- Keywords - Listado de datasets que contiene el repositorio (eventos, restaurantes, etc...)
- Timestamp - Marca de tiempo para calcular la caducidad del mismo.

Schema	
Nombre	Tipo
id	long
topic	String
timestamp	long
schema	String
android_package	String

Cuadro 5.6: Clase Schema

Almacena la estructura de los esquemas e información relevante. Se inicializan al inicio de la app desde el dto/Schema.java (getSchemaDaoCopy()) que a su vez se llama desde el RestService.java (getSchemas()). Esto se hace con el objetivo de tener una copia del esquema en la base de datos.

- Id: Identificador del esquema
- Topic: Nombre del esquema (Por ejemplo: Restaurante)
- Timestamp: Caducidad del esquema (Validez en caché)
- Schema: JSON con la estructura del esquema. Un ejemplo:

```
[{"field":"nombre","type":"title"},
{"field":"descripcion","type":"generic"},
{"field":"f_ini","type":"generic"},
{"field":"f_fin","type":"generic"},
{"field":"h_ini","type":"generic"},
{"field":"h_fin","type":"generic"},
```

```
{"field":"lugar","type":"generic"},
{"field":"geolocalizacion","type":"geolocation"},
{"field":"url","type":"url"},
{"field":"tipo_evento","type":"generic"}]
```


- Android_package: Los esquemas pueden tener una aplicación desarrollada en Android por un contribuyente, esa aplicación es descargada y ejecutada. Para poder hacer todo esto se requiere almacenar el nombre del paquete, que se hace precisamente con este atributo.

Almacenamiento remoto

La aplicación almacena las apps generadas en **firebase** para poder obtenerlas desde cualquier sitio, por ello, la aplicación tiene un servicio de registro e inicio de sesión creado con Firebase. Tras analizar la implementación de este servicio, se ha decidido **eliminar** esta función de la aplicación ya que estaba implementada sin seguir ningún patrón y sin desacoplamiento de las diferentes capas. Además su implementación era incorrecta ya que no realizaba correctamente el registro y posterior inicio de sesión.

Esta funcionalidad ha sido la **única** que se ha modificado en la aplicación, la implementación era nefasta y debía implementarse de nuevo.

Repository Pattern

Las referencias a la base de datos y las llamadas a la API Rest en el proyecto no siguen ningún patrón correctamente por lo que se ha tomado la decisión de abordar la refactorización aislando las operaciones de datos que se realizan y analizando cada una de ellas. Para ello se ha realizado una tabla donde se muestra cada operación de datos junto con el módulo desde el cual se referencia a las mismas:

Id	Módulo	Operación	Entidad	Tipo	Usos módulo
1	AppsDao	insert()	Apps	Inserción	AppRepository
2	DbService	getApp(name)	Apps	Consulta	AppRepository
3	DbService	getListApps()	Apps	Consulta	AppRepository
4	DbService	deleteAll()	Apps	Borrado	AppRepository
5	AppsDao	delete(App)	Apps	Borrado	AppRepository
6	DbService	getReposByKeyword()	Repo	Consulta	DataPresenter
8	DbService	insert(Repo)	Repo	Inserción	DataContent - RootActivity
9	SchemaDao	getSchemaById	Schema	Consulta	SchemaRepository - AppgenActivity
10	DbService	recycleSchemas()	Schema	Actualización	RootActivity

Como se observa en la tabla anterior, las referencias a datos de las entidades Repo y Schema se realizan desde los presenters y activities por lo que no se ha aplicado correctamente el desacoplamiento de la capa de datos y por consiguiente el patrón **repository**.

Capa de dominio

El proyecto no tiene una capa de dominio bien definida.

Capa de presentación

El proyecto se ha creado utilizando el patrón de presentación MVP:

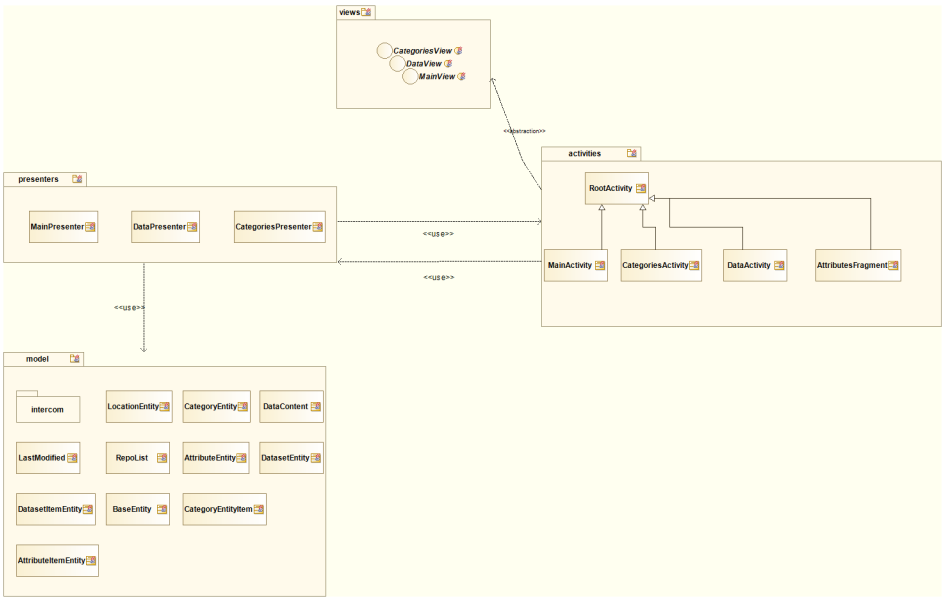

Figura 5.8: Estructura final del proyecto

Como ya se explicó en el análisis comparativo de documentación y código fuente, el patrón está mal implementado y no existe desacoplamiento real entre las capas.

Para la capa vista se habían creado interfaces tratando de conseguir un desacoplamiento de los presenters. Lo cierto es que estas interfaces son inútiles.

A continuación se exponen tablas con las clases que forman las capas del proyecto:

Vista	
Clase	Tipo
MainView	Vista
MainActivity	Activity
DataView	Vista
DataActivity	Activity
DataActivitySelectorRepoFragment	Fragment
DataActivitySelectorSchemaFragment	Fragment
CategoriesView	Vista
CategoriesActivity	Activity
AttributesFragment	Fragment

Cuadro 5.7: Capa vista

Presenters	
Clase	Tipo
MainPresenter	Presenter
DataPresenter	Presenter
CategoriesPresenter	Presenter

Cuadro 5.8: Capa presentadores

Modelo	
Clase/Paquete	Tipo
intercom	Paquete
LocationEntity	Entidad
CategoryEntity	Entidad
CategoryEntityItem	Entidad
AttributeEntity	Entidad
AttributeItemEntity	Entidad
DatasetEntity	Entidad
DatasetItemEntity	Entidad
BaseEntity	Entidad
DataContent	Clase aux
LastModified	Clase aux
RepoList	Lista

Cuadro 5.9: Capa modelo

En total tenemos la siguiente distribución:

Tipo	Número de clases
Vista	3
Activity	3
Fragment	3
Presenter	3
Entidad	8
Listas	1
Clases aux	2

Cuadro 5.10: Capa presentadores

5.4.2. Actividad 4.2 - Diseño de la nueva arquitectura

La estructura del proyecto a la que se quiere llegar es la que recomienda Google en su guía para la arquitectura de aplicaciones[17]:

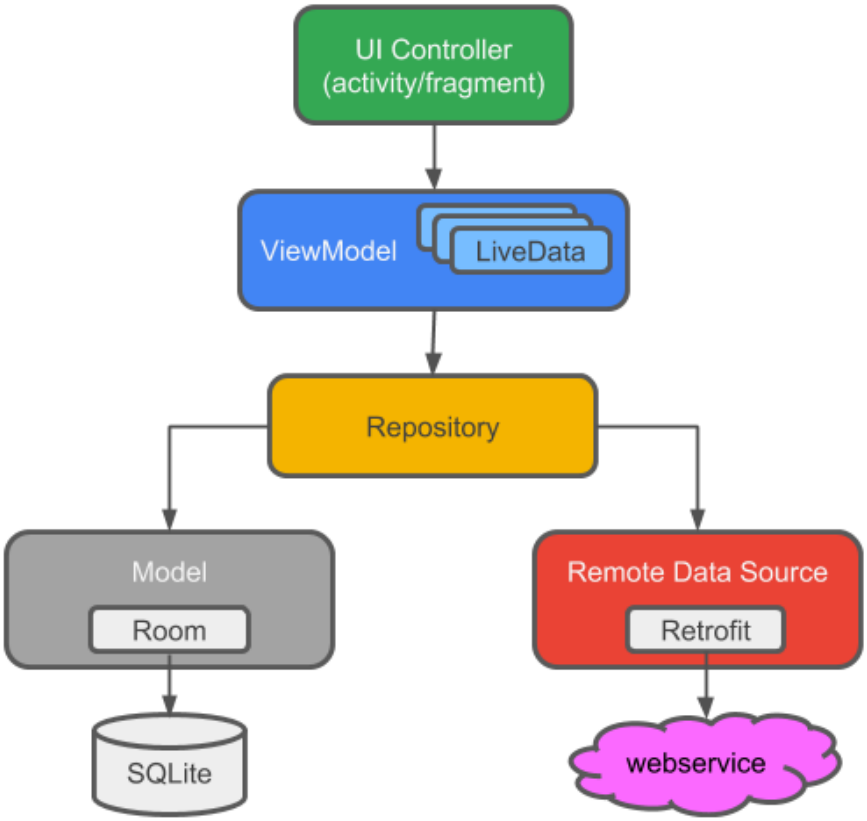

Figura 5.9: Estructura final del proyecto

Capa de datos

Almacenamiento local

La aplicación como se ha visto anteriormente utilizaba la librería **GreenDAO** para realizar las operaciones sobre la base de datos.La generación de código que realiza GreenDAO es menos comprensible que la de ROOM. Además de éste motivo, ROOM da otras ventajas:

- Menos código repetitivo en comparación con las API incorporadas (GreenDao, ORMLite, etc...). Realiza la asignación de objetos de base de datos a objetos Java, por lo que no hay que utilizar Cursores.
- Validación en tiempo de compilación de consultas SQL, por lo que las

sentencias SQL incorrectas se capturan en tiempo de compilación, no en tiempo de ejecución.

- Permite la observación de datos a través de LiveData (Que se utilizarán en la aplicación).

La librería ROOM tiene principalmente 3 componentes[2]:

1. Entidades @Entity - Representa una tabla en la base de datos.
2. DAO - @DAO - Contiene los métodos usados para el acceso a la base de datos.
3. Base de Datos @Database - Contiene una instancia de la base de datos y es el principal punto de acceso a la conexión subyacente a los datos relacionales persistentes de la aplicación.

Estos componentes, junto con sus relaciones con el resto de la aplicación, se exponen en la siguiente figura:

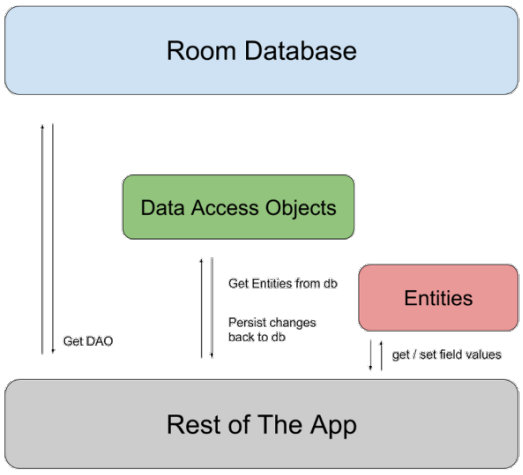

Figura 5.10: Diagrama de la arquitectura de Room [2].

ROOM sigue prácticamente la misma estructura que GreenDao por lo que para realizar la refactorización de la base de datos y sus operaciones podemos aprovechar parte del código que ya estaba implementado.

Almacenamiento Remoto

Puesto que se ha decidido eliminar el almacenamiento remoto de la aplicación, en esta actividad se ha profundizado en su implementación para posteriormente eliminarla.

Repository Pattern

Los repositorios proporcionan una API limpia para el resto de la aplicación. Saben de dónde obtener los datos y qué llamadas hacer cuando se actualizan los datos.

Se va a implementar un repositorio para cada entidad que se almacena en la base de datos:

- AppRepository
- RepoRepository
- SchemaRepository

Las clases repositorio que se van crear tienen que tener las siguientes características:

- Implementan el patrón Singleton para evitar la creación de más de 1 instancia.
- Guardan una referencia a la base de datos para realizar las operaciones.
- Puesto que ROOM no permite las operaciones sobre la base de datos en el hilo principal, las clases repositorios referencian a la clase Executor para ejecutar las operaciones en hilos secundarios.
- Tienen una serie de métodos con todas las operaciones que se realizan en la base de datos sobre la entidad del repositorio.

Tras la refactorización y aplicación del patrón repositorio el proyecto debe tener la siguiente estructura:

Id	Módulo	Operación	Entidad	Tipo	Usos módulo
1	AppsDao	insert()	Apps	Inserción	AppRepository
2	DbService	getApp(name)	Apps	Consulta	AppRepository
3	DbService	getListApps()	Apps	Consulta	AppRepository
4	DbService	deleteAll()	Apps	Borrado	AppRepository
5	AppsDao	delete(App)	Apps	Borrado	AppRepository
6	DbService	getReposByKeyword()	Repo	Consulta	RepoRepository
7	DbService	deleteAll()	Repo	Borrado	RepoRepository
8	DbService	insert(Repo)	Repo	Inserción	RepoRepository
9	SchemaDao	getSchemaById()	Schema	Consulta	SchemaRepository
10	DbService	recycleSchemas()	Schema	Actualización	SchemaRepository

Capa de dominio

La capa de dominio es ésta aplicación es innecesaria puesto que la aplicación no tiene una lógica compleja.

Capa de presentación

La aplicación seguía un patrón de presentación MVP pero se ha decidido cambiar la arquitectura a MVVM puesto que Google ha elaborado una completa guía para la implantación de la arquitectura MVVM en las aplicaciones Android (Android Architecture Components). Esta arquitectura facilita la elaboración de pruebas entre otras ventajas. Para la migración de arquitectura se han realizado los siguientes cambios:

Presentadores	
Presenter	ViewModel
MainPresenter	MainActivityViewModel
DataPresenter	DataActivityViewModel
CategoriesPresenter	CategoriesActivityViewModel

Cuadro 5.11: Refactorización de los presenters

Vistas	
MainView	*
MainActivity	MainActivity
DataView	*
DataActivity	DataActivity
DataActivitySelectorRepoFragment	DataActivitySelectorRepoFragment
DataActivitySelectorSchemaFragment	DataActivitySelectorSchemaFragment
CategoriesView	*
CategoriesActivity	CategoriesActivity
AttributesFragment	AttributesFragment

Cuadro 5.12: Refactorización de las vistas.

Modelo	
intercom	intercom
LocationEntity	LocationEntity
CategoryEntity	CategoryEntity
CategoryEntityItem	CategoryEntityItem
AttributeEntity	AttributeEntity
AttributeItemEntity	AttributeItemEntity
DatasetEntity	DatasetEntity
DatasetItemEntity	DatasetItemEntity
BaseEntity	BaseEntity
DataContent	*
LastModified	LastModified
RepoList	LastModified

Cuadro 5.13: Refactorización de la capa modelo.

*Se va a eliminar dicha clase en la refactorización.

5.4.3. Actividad 4.3 - Implementación de la arquitectura

En el proceso de refactorización por capas se ha tratado de utilizar las recomendaciones de Google junto con el uso de diferentes patrones que mejoran la comprensión del código, su organización, mantenimiento etc...

Capa de datos

Almacenamiento local

Paso 1 - Añadir las dependencias necesarias al proyecto

Puesto que la Arquitectura de Componentes está disponible en el repositorio Maven de Google, lo primero será añadirlo al proyecto:

```
allprojects {
    repositories {
        jcenter()
        maven { url 'https://maven.google.com' }
    }
}
```

Tras añadir el repositorio Maven, hay que añadir las dependencias específicas de ROOM[21]:

```
implementation "android.arch.persistence.room:runtime:1.0.0"
annotationProcessor "android.arch.persistence.room:compiler:1.0.0"
testImplementation "android.arch.persistence.room:testing:1.0.0"
```

Ya tenemos las dependencias anteriores añadidas al proyecto, ahora se debe realizar una sincronización de los archivos del proyecto para que descargue las librerías necesarias.

Paso 2 - Creación de las entidades que se almacenan en la base de datos.

GreenDao utiliza una sintaxis diferente para definir las entidades que se guardarán en la base de datos, pero los objetos ya están creados, así que simplemente hay que añadir las siguientes etiquetas para que ROOM los identifique:

- @Entity(tableName = "Nombre de la tabla en la base de datos"): Se añade antes del nombre de la clase.

- @PrimaryKey: Se añade en el atributo el cual va a ser la clave primaria de la tabla. Se le puede otorgar diferentes propiedades como que se autogenere.

Hay otras etiquetas que se pueden utilizar para las relaciones entre entidades, ignorar métodos etc... pero las anteriores son obligatorias.

En este paso se ha aprovechado para modificar los nombres de los atributos que se almacenaban en la base de datos para que sean lo más descriptivos posible. Un ejemplo de los cambios realizados:

Repo	
Antes	Después
id	repo_id
name	repo_name
place	repo_place
geo_lat	repo_geo_lat
geo_long	repo_geo_long
keywords	repo_keywords
timestamp	repo_timestamp

Cuadro 5.14: Ejemplo de refactorización de atributos

Paso 3 - Creación de los Objetos de Acceso a Datos (DAOs).

Los Objetos de Acceso a Datos que había en el proyecto habían sido generados por GreenDao y gran parte del código estaba "Oculto" en clases abstractas de las librerías de GreenDao.

Se ha creado un DAO por cada entidad que se almacena en la base de datos. Los DAOs son el componente principal de ROOM y pueden ser una clase abstracta o una interfaz. Se ha tomado la decisión de utilizar **interfaces** ya que son más robustas y flexibles.

En greenDAO las operaciones sobre la base de datos se generaban de forma automática, en ROOM hay que definir las operaciones que necesitemos en nuestra aplicación utilizando consultas SQL. Por ello, se han recopilado todas las operaciones que se realizaban sobre cada tabla de la base de datos y transformado en consultas SQL, a continuación se exponen algunas de las consultas que se realizan a la base de datos:

- Create:
  @Insert (onConflict=REPLACE) - Añadiendo ésta etiqueta al método, ROOM identifica este método para la inserción de una nueva entidad.
- Read:
  @Query("SELECT * FROM apps WHERE app_name = :name") - Añadiendo la etiqueta junto a la consulta, ROOM obtiene la entidad o entidades filtradas de la base de datos.
- Update:
  @Update - Ésta es la etiqueta que utiliza ROOM para modificar las entidades deseadas de la base de datos. (No se ha utilizado en el proyecto).
- Delete:
  @Query("DELETE FROM apps") - Con esta etiqueta ROOM identifica el método para borrar entidades de la base de datos.

Paso 4 - Creación de la Base de Datos.

La creación de la base de datos hasta ahora se realizaba desde la clase DaoMaster generada por GreenDAO. La nueva clase se anota con la etiqueta @Database y debe satisfacer las siguientes condiciones:

- Ser abstracta y extender de RoomDatabase
- Incluir una lista de las entidades asociadas a la base de datos:

```
@Database(entities = {App.class, Repo.class, Schema.class},
version = 1)
```

- Contener un método abstracto que no tenga argumentos y devuelva la clase anotada con @Dao:

```
public abstract AppDao appDao();

public abstract SchemaDao schemaDao();

public abstract RepoDao repoDao();
```

La creación de la base de datos se realiza utilizando el patrón Singleton, ya que no se puede crear más de 1 instancia de la base de datos. Para crearla se utiliza la siguiente sentencia:

```
Room.databaseBuilder(Contexto, Database.class, DATABASE_NAME).
fallbackToDestructiveMigration().build();
```

La base de datos ya está creada y lista para usar. El siguiente paso es actualizar el proyecto para utilizar las nuevos métodos de la base de datos.

Almacenamiento remoto

Se han eliminado todas las líneas de código referentes al almacenamiento remoto, limpiando mucho el código ya que estaba en todas las capas.

Repository Pattern

Mediante la siguiente refactorización se consigue una aplicación correcta del patrón repository y por consiguiente una separación de intereses aislando la capa de datos del resto de la aplicación.

Paso 1 - Creación de los Repositorios.

Gracias al análisis hecho anteriormente ya se pueden crear los repositorios con los métodos que se van a añadir a cada uno:

AppRepository.java
saveState()
deleteAll()
getListApps()
getApp(String app_name)
deleteApp(Apps app)

Cuadro 5.15: AppRepository.java

RepoRepository.java
insertRepo(Repo repo)
deleteRepos()
getReposByKeyword(List<String> categories)

Cuadro 5.16: RepoRepository.java

SchemaRepository.java
getSchemaById(Long id)
updateSchemas(List<Schema> schemas)
getCategoriesByNet()
getSchemasByNet()

Cuadro 5.17: SchemaRepository.java

Paso 2 - Actualización de las referencias a las operaciones de datos.

En las tablas mostradas en el análisis se observa como muchas operaciones de datos se realizan desde la clase DbService, esta clase se creó a modo de "Servicio de Base de datos" y actúa como intermediario entre los repositorios y la base de datos, en la siguiente tabla se muestran las operaciones que se realizan en dicha clase y las clases desde donde se referencian:

DbService.java	
Operación	Referencias
saveState()	AppRepository.java
deleteAll()	AppRepository.java
getListApps()	AppRepository.java
getApp(String app_name)	AppRepository.java
getSchemaById(Long schema_id)	AppgenActivity.java
recycleSchemas(List<Schema> schemas)	RootActivity.java
recycleRepos(List<Repo> repos)	DataContent.java
getReposByKeyword(HashSet <CategoryEntityItem> categories)	DataPresenter.java
deleteApp(Apps app)	AppRepository.java

Cuadro 5.18: Métodos en DbService.java

Tras analizar cual es la función de la clase DbService.java, la decisión que se ha tomado es eliminarla ya que añade complejidad al proyecto y no es necesaria ya que la función que realizaba hasta ahora va a pasar a los repositorios creados anteriormente, esta clase además estaba bajo una interfaz DataSource.java que también se va a eliminar.

Paso 3 - Creación e inicialización de instancias de los repositorios.

Ahora que ya tenemos todos los repositorios implementados y que se han eliminado las clases DbService y DataSource, debemos crear instancias de los repositorios e inicializarlas en las clases donde se realicen operaciones con los datos. Hay 2 patrones recomendados por Google para gestionar las dependencias[18]:

1. Dependency Injection: Permite a las clases definir sus dependencias sin inicializarlas y en tiempo de ejecución otras clases proveen dichas dependencias.
2. Service Locator: Es más sencillo de implementar que la inyección de dependencia, provee un registro donde las clases pueden obtener sus dependencias en lugar de inicializarlas.

Se ha decidido usar la inyección de dependencias ya que es lo más adecuado para lo que se quiere realizar. Para ello se ha creado la clase InjectorUtils.java que tiene las siguientes características:

- Un método proveedor para cada repositorio.
- Cada 'provider' crea un nuevo Executor, obtiene una instancia de la base de datos y retorna una instancia del repositorio.
- Los métodos 'provider' son de tipo 'static'.

A continuación se expone un ejemplo de los métodos 'provider' que hay en la clase InjectorUtils.java:

```
public static AppRepository provideAppRepository(Context context) {
        Executor executor = new ThreadPerTaskExecutor();
        YourInstantAppDatabase database = YourInstantAppDatabase.
        .getInstance(context.getApplicationContext());
        return AppRepository.getInstance(database, executor);
    }
```

Para obtener la instancia de un repositorio simplemente se deberá llamar a la función 'provider' del repositorio que corresponda en la clase donde se vaya a utilizar.

En este punto de la refactorización ya tenemos la capa de datos totalmente refactorizada utilizando la librería ROOM y aplicando el patrón repositorio.

Capa de dominio

La capa de domino interactúa con el modelo después de tener una acción en la vista. Ésta aplicación no tiene una lógica de dominio compleja, por ello no está definida. Solo hay objetos de dominio o de datos.

Capa de presentación

La aplicación se desarrolló tratando de utilizar una arquitectura MVP, para la refactorización se ha decidido utilizar la Android Architecture Components que se basa en la arquitectura MVVM explicada anteriormente. Para llegar a esta nueva arquitectura se han hecho muchos cambios:

La clase **RootActivity** es una activity de las que el resto de activities extienden, su utilidad es aunar todas las funciones que deben ser comunes en el resto de Activities. Lo cierto es que esta activity es innecesaria ya que no tenía sentido que todos los activities cargaran por ejemplo las categorías al iniciarse. Por ello, se ha eliminado esta activity y movido los métodos de la misma a donde corresponde, ahora todos los Activities extienden de AppCompatActivity, la superclase de Android.

Fruto de la refactorización anterior, se ha creado una clase **Error** que contiene los métodos que anuncian los errores que pueden surgir en la aplicación, esta clase se ha hecho estática para que sea accesible desde cualquier clase.

La arquitectura MVVM se caracteriza por el desacoplamiento del modelo, vista y vista-modelo. Para conseguir ésto, se han creado clases **ViewModel** que almacenan y administran datos relacionados con la interfaz de usuario, estas clases no tienen referencias al contexto del proyecto y su creación se ha realizado mediante inyección de dependencias y utilizando el patrón Factory Method. Para poder usar estas clases debemos añadir las siguientes dependencias al gradle [21]:

```
implementation "android.arch.lifecycle:extensions:1.0.0"
```

Puesto que los viewmodels sustituyen a los presenters, todos los métodos se deben trasladar y adaptar. Este proceso se ha ido haciendo para cada viewModel de la siguiente manera:

1. Creación del viewModel.
2. Traslado de métodos de los presenters a el viewModel.
3. Adaptación de los métodos a los viewModels (No puede haber referencias al contexto).
4. Creación del factory que permitirá la instanciación del viewModel.
5. Adición del viewModel a la inyección de dependencias.
6. Instanciación del viewModel desde el Activity.

Las vistas del proyecto son las diferentes activities y fragments que tiene la aplicación, la vista no debe realizar operaciones en los datos de la aplicación. Antes de la refactorización, tenemos las siguientes llamadas a los repositorios desde los Activities:

Repositorio	Operación	Activity
schemaRepository	updateSchemas()	CategoriesActivity
schemaRepository	getSchemaById(id)	DataActivity

Cuadro 5.19: Llamadas a repositorios desde Activities

A continuación se expone la solución a estos problemas de acoplamiento entre capas: La operación getSchemaById se ha solucionado moviendo la clase Asynctask que es donde se referencia al método al ViewModel para después llamarlo desde la vista de la siguiente manera:

```
viewModel.new getSchemas(repo, ft, this).execute();
```

La operación updateSchemas() se ha llevado a un nuevo método en el viewModel y desde el asynctask se le llama, de esta manera conseguimos que el repositorio se instancie en el viewModel y no en el Activity. Tras esta referencia ya no tenemos referencias a los datos desde los activities, otra ventaja de la Android Architecture Components es la posibilidad utilizar objetos LiveData explicados anteriormente. Ésto será el siguiente paso, la incorporación de LiveData que faciliten la actualización de los datos en las vistas.

En los ViewModels se añaden las llamadas a los repositorios y en el Activity simplemente se observa este método y en el momento que haya

cambios se actualiza automáticamente. La utilización de LiveDatas requiere de referencias a LifecycleOwner ya que su activación depende del ciclo de vida de los activities o fragments.En la consulta getReposByKeyword que se realiza desde el ViewModel tenemos el siguiente problema: ¿Cómo se suscribe al repositorio desde el ViewModel si no tiene acceso a LifecycleOwner? Una de las soluciones a este problema es utilizar el método observeForever(Observer). En este caso, el observador está siempre activo y por ello se le notifica siempre de cambios en el repositorio.

Tras la creación de los viewmodels, adaptación de las vistas y creación del repositorio ya tenemos la arquitectura MVVM implementada en el proyecto. Las siguientes refactorizaciones se han hecho con el objetivo de limpiar el código del proyecto:

- Reorganización del código del proyecto, cambiando el árbol de paquetes del mismo.
- Eliminación de clases inútiles: GPSTracker.java
- Modificación de los nombres de atributos de los objetos de datos, para que sean más descriptivos.
- Revisado y eliminación de las dependencias del proyecto.

5.4.4. Ejecución de las pruebas

Capítulo 6

Conclusiones

6.1. Consecución de objetivos

Para comprobar qué objetivos se han cumplido se irán enumerando los objetivos que se tenían en el apartado Objetivos con la debida explicación sobre de qué forma se han conseguido:

Objetivo principal

Definir un proceso que nos permita tener una guía para la refactorización y migración de aplicaciones Android heredadas o mal diseñadas.

Consecución

Se considera totalmente conseguido. Se ha desarrollado una guía muy completa, perfecta para realizar la refactorización y migración de aplicaciones Android.

Objetivo 2

Conocimiento de herramientas de análisis estático de código.

Consecución

Se valora como conseguido, se ha utilizado la herramienta de análisis estático de código Lint, que viene incluida en el IDE Android Studio y se ha explicado su funcionamiento en profundidad.

Objetivo 3

Aprender a enfrentarse a aplicaciones heredadas.

Consecución

Se considera conseguido, ya que para aplicar la refactorización han surgido muchos problemas fruto de la inexperiencia. Ésto hace que haya adquirido soltura para la resolución de dichos errores y por tanto, en las siguientes aplicaciones me será mas sencillo aplicarles el proceso.

Objetivo 4

Aprendizaje de patrones de arquitectura utilizados en aplicaciones Android.

Consecución

Se puede valorar como conseguido, se han estudiado una gran cantidad de patrones de arquitectura para el desarrollo de aplicaciones Android y se han aplicado en la aplicación.

Objetivo 5

Conocimiento de procesos básicos de refactorización en aplicaciones Android.

Consecución

Se puede considerar como conseguido, se han utilizado muchos procesos básicos en la aplicación de la refactorización y se han explicados en la guía.

6.2. Posibles ampliaciones

Las ampliaciones irían principalmente destinadas a la aplicación heredada ya que la guía se ha elaborado de forma muy meticulosa. Una ampliación de la aplicación heredada sería refactorizar la sección de la aplicación generada ya que como está ahora no tiene un desacoplamiento de capas correcto.

Puesto que en la refactorización se ha borrado el almacenamiento remoto de aplicaciones generadas, en el futuro, añadir esta funcionalidad de forma correcta sería necesario.

Para el futuro sería importante añadir un catálogo de pruebas completo en la aplicación, consiguiendo mejorar el mantenimiento y previniendo errores.

6.3. Conclusiones personales

La realización de este trabajo me ha mostrado la realidad de muchos proyectos en el mundo laboral y me ha ayudado a enfrentarme a ellos. Gracias a él he conseguido entender la importancia del uso de patrones en el desarrollo de aplicaciones. También he aprendido una gran cantidad de técnicas de refactorización que me ayudarán en mi futuro.

El proyecto ha conseguido ponerme a prueba debido a los problemas que han ido surgiendo en su desarrollo; estos problemas en gran parte han surgido por la utilización de tecnologías presentadas en los últimos meses, ya que cuando empecé a trabajar con ellas aún estaban en fase *beta*, esto se transforma en experiencia ganada que seguro me servirá más adelante.

Ha cambiado mi manera de programar para hacerla más organizada y meticulosa. Y por supuesto, me va abrir muchas puertas en el mundo laboral gracias al conocimiento que he adquirido en desarrollo Android y otras tecnologías como Lint o programación multihilo.

Mi sensación al terminar el proyecto es muy satisfactoria, me hubiera gustado profundizar aún más en algunos aspectos pero por falta de tiempo no ha podido ser posible. Me llevo una gran experiencia que pondré con orgullo en mi curriculum vitae y mostraré allá donde vaya.

Bibliografía

[1] Fundamentals of testing. https://developer.android.com/training/testing/fundamentals.html. Accedido 22-01-2018.

[2] Room persistence library. Saving Data Using the Room Persistence Library. Accedido 22-01-2018.

[3] William Opdyke Martin Fowler (with Kent Beck, John Brant and Don Roberts). *Refactoring: Improving the Design of Existing Code*. Addison Wesley, 1999.

[4] Mvc y mvvm (3-5) ¿qué son y en qué se diferencian? https://www.adictosaltrabajo.com/tutoriales/zk-mvc-mvvm/. Accedido 22-01-2018.

[5] Separación de intereses. https://en.wikipedia.org/wiki/Separation_of_concerns. Accedido 22-01-2018.

[6] Data access object. http://castellanosmiguel.blogspot.com.es/2013/07/dataaccess-object-d Accedido 22-01-2018.

[7] Inyección de dependencias. https://es.wikipedia.org/wiki/Inyección_de_dependencias. Accedido 22-01-2018.

[8] Observer pattern. https://en.wikipedia.org/wiki/Observer_pattern. Accedido 22-01-2018.

[9] Singleton pattern. https://es.wikipedia.org/wiki/Singleton. Accedido 22-01-2018.

[10] Ralph Johnson Erich Gamma, John Vlissides and Richard Helm. *Design Patterns: Elements of Reusable Object-Oriented Software*.

[11] Single source of truth data. https://thisbitofcode.com/single-source-truth-data/. Accedido 15-12-2017.

[12] Programación por capas. https://es.wikipedia.org/wiki/Programacion_por_capas. Accedido 23-01-2018.

[13] Implementando patrón repositorio - repository pattern en c parte i. http://www.eltavo.net/2014/08/patrones-implementando-patron.html. Accedido 22-01-2018.

[14] Análisis estático de software. https://en.wikipedia.org/wiki/Static_program_analysis. Accedido 22-01-2018.

[15] Mejora tu código con lint. https://developer.android.com/studio/write/lint.html?hl=es-41 Accedido 22-01-2018.

[16] Introduction to android architecture components. https://code.tutsplus.com/tutorials/introduction-to-android-architecture–cms-28749. Accedido 22-01-2018.

[17] Build and app with architecure components. https://codelabs.developers.google.com/codelabs/build-app-with-arch-components/#0. Accedido 22-01-2018.

[18] Guide to app architecture. https://developer.android.com/topic/libraries/architecture/gui Accedido 22-01-2018.

[19] Room persistence library. https://developer.android.com/topic/libraries/architect ure/room.html. Accedido 22-01-2018.

[20] Desarrollo iterativo y creciente. https://es.wikipedia.org/wiki/Desarrollo_iterativo_y _creciente. Accedido 22-01-2018.

[21] Adding components to your project. https://developer.android.com/topic/libraries/architecture/ adding-components.html. Accedido 22-01-2018.

Anexo: Catálogo ...

Errores

Android

```
Error 1 - activity\_appgen.xml
 Unresolved package 'eneas'
 Unresolved package 'instantapp'
 Unresolved package 'app\_gen'
 Unresolved class 'AppgenActivity'
<android.support.design.widget.CoordinatorLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_
   android:layout_height="match_parent" android:fitsSystemWindows="true"
   tools:context="eneas.instantapp.app_gen.AppgenActivity">
Refactorización 1 - activity\_appgen.xml
<android.support.design.widget.CoordinatorLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_
   android:layout_height="match_parent" android:fitsSystemWindows="true"
  tools:context="es.unex.quercusseg.yourinstantapp.app_gen.AppgenActivity">

Error 2 - activity\_login.xml
Unresolved class 'LoginActivity'
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_
   android:layout_height="match_parent" android:gravity="center_horizontal"
   android:orientation="vertical" android:paddingBottom="@dimen/activity_verti
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   tools:context=".activities.LoginActivity">

Refactorización 2 - activity\_login.xml
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_
```

```
   android:layout_height="match_parent" android:gravity="center_horizontal"
   android:orientation="vertical" android:paddingBottom="@dimen/activity_verti
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   >
```

```
Refactorización 3 - AndroidManifest.xml
'es.unex.quercusseg.yourinstantapp.activities.AttributesFragment' is not assig
'es.unex.quercusseg.yourinstantapp.app_gen.MapLayoutFragment' is not assignabl
<activity
   android:name=".activities.AttributesFragment"
   android:label="@string/title_activity_attributes"
   android:theme="@style/AppTheme.NoActionBar" >
</activity>
<activity
   android:name=".app_gen.MapLayoutFragment"
   android:label="MapActivty" />
<activity

Refactorización 3 - AndroidManifest.xml
Eliminar etiquetas <activity> del AndroidManifest ya que estaban asignadas a F

Error 4 - AndroidManifest.xml
Unresolved package 'auth'
Refactorización 4 - AndroidManifest.xml
Eliminar Módulo auth - Este módulo era el culpable de una gran cantidad de los
```

AndroidLint: Correctness

```
Error 5 - activity_login.xml
Invalid resource type, expected integer value
<android.support.design.widget.TextInputLayout android:layout_width="match_par
   android:layout_height="wrap_content">
   <EditText android:id="@+id/password" android:layout_width="match_parent"
       android:layout_height="wrap_content" android:hint="@string/prompt_passw
       android:imeActionId="@+id/login"
       android:imeActionLabel="@string/action_sign_in_short"
       android:imeOptions="actionUnspecified" android:inputType="textPassword"
```

```
       android:maxLines="1" android:singleLine="true" />
</android.support.design.widget.TextInputLayout>
```

Refactorización 5 - activity_login.xml
Según la documentación de Google la definición del atributo imeActionId: Suppl

Error 6 - DataPresenter.java
Call requires permission which may be rejected by user: code should explicitly

```
private Location getLastKnownLocation() {
   LocationManager mLocationManager = (LocationManager) YourInstantApplication
   List<String> providers = mLocationManager.getProviders(true);
   Location bestLocation = null;
   for (String provider : providers) {
       Location l = mLocationManager.getLastKnownLocation(provider);
       if (l == null) {
           continue;
       }
       if (bestLocation == null || l.getAccuracy() < bestLocation.getAccuracy(
           // Found best last known location: %s", l);
           bestLocation = l;
       }
   }
   return bestLocation;
}
```

Refactorización 6 - DataPresenter.java
Se ha añadido la comprobación de permisos de forma que solo obtenga la localiz

```
private Location getLastKnownLocation() {
   LocationManager mLocationManager = (LocationManager) YourInstantApplication
   List<String> providers = mLocationManager.getProviders(true);
   Location bestLocation = null;
   if (ActivityCompat.checkSelfPermission(YourInstantApplication.appContext, M
       for (String provider : providers) {
           Location l = mLocationManager.getLastKnownLocation(provider);
           if (l == null) {
               continue;
           }
           if (bestLocation == null || l.getAccuracy() < bestLocation.getAccur
               // Found best last known location: %s", l);
```

```
                bestLocation = l;
            }
        }
    }
    return bestLocation;
}
```

```
Error 7 - GPSTracker.java
Call requires permission which may be rejected by user: code should explicitly
public Location getLocation() {
    try {
        locationManager = (LocationManager) mContext.getSystemService(LOCATION_
        GPSEnabled = locationManager.isProviderEnabled(LocationManager.GPS_PROV
        NetworkEnabled = locationManager.isProviderEnabled(LocationManager.NETW
        if (!GPSEnabled && !NetworkEnabled) {
            //no network
        }
        else {
            this.canGetLocation = true;
            if (NetworkEnabled) {
                locationManager.requestLocationUpdates(LocationManager.NETWORK_
                        MIN_TIME_FOR_UPDATES, MIN_DISTANCE_FOR_UPDATES, this);
                Log.d("Network", "Network");
                if(locationManager != null) {
                    location = locationManager.getLastKnownLocation(LocationMan
                    if(location != null) {
                        latitude = location.getLatitude();
                        longitude = location.getLongitude();
                    }
                }
            }
            if (GPSEnabled) {
                if (location == null) {
                    locationManager.requestLocationUpdates(LocationManager.GPS_
                            MIN_TIME_FOR_UPDATES, MIN_DISTANCE_FOR_UPDATES, thi
                    Log.d("GPS Enabled", "GPS Enabled");
                    if (locationManager != null) {
                        location = locationManager.getLastKnownLocation(Locatio
                        if (location != null) {
                            latitude = location.getLatitude();
```

```
                        longitude = location.getLongitude();
                    }
                }
            }
        }
    }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return location;
}
```

```
Refactorización 7 - GPSTracker.java
Añadida comprobación de permisos:
if (ActivityCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_L
}
```

```
Error 8 - Locator.java
Call requires permission which may be rejected by user: code should explicitly
```

```
Refactorización 8 - Locator.java
Añadida comprobación de permisos como en los errores solucionados anteriorment
```

JavaDoc Issues

```
Declaration has problems in Javadoc references
```

```
Error 9 - Connectivity.java
Cannot resolve symbol 'type'
/**
* Check if there is any connectivity to a Wifi network
* @param context
* @param type
* @return
*/
public static boolean isConnectedWifi(Context context){
    NetworkInfo info = Connectivity.getNetworkInfo(context);
    return (info != null && info.isConnected() && info.getType() == Connectivit
}
```

```
Refactorización 9 - Connectivity.java
Eliminar el parámetro type ya que no existe.
/**
* Check if there is any connectivity to a Wifi network
* @param context
* @return
*/
public static boolean isConnectedWifi(Context context){
   NetworkInfo info = Connectivity.getNetworkInfo(context);
   return (info != null && info.isConnected() && info.getType() == Connectivit
}
```

XML

```
Error 10 - checkstyle.xml
URI is not registered (Settings | Languages & Frameworks | Schemas and DTDs)
<!DOCTYPE module PUBLIC "-//Puppy Crawl//DTD Check Configuration 1.3//EN" "htt
Refactorización 10 - checkstyle.xml
Android Studio lo soluciona mediante la opción Fetch External Resources.
```

```
En este punto ya tenemos solucionados todos los errores que Lint detectaba en
La inspección de Lint actualmente tiene el siguiente estado:
```

Avisos

Android

Aviso 1 - activity_main_drawer.xml

```
 Element group is not allowed here
 Element menu is not allowed here
 Element item is not allowed here
<item android:title="My apps">
```

```
   <group  android:id="@+id/drawer_group_app"  android:checkableBehavior="sing
       <menu >
           <item android:id="@+id/nav_camara" android:icon="@android:drawable/
               android:title="Import" />
       </menu>
   </group>
</item>

Refactorización 1 - activity\_main\_drawer.xml
 Reestructurar el grupo de elementos.
<group  android:id="@+id/drawer_group_app" >
   <item android:title="My apps">
   <menu >
       <item android:id="@+id/nav_camara" android:icon="@android:drawable/ic_m
           android:title="Import" />
   </menu>
   </item>
</group>



Aviso 2 - layout/app_bar_main.xml
Missing ‘contentDescription’ attribute on image
<ImageView
   android:layout_width="200sp"
   android:layout_height="125sp"
   android:src="@drawable/logo_panel2"/>

Refactorización 2 -  layout/app_bar_main.xml
Añadir el atributo contentDescription con una descripción de la imágen.
<ImageView
   android:layout_width="200sp"
   android:layout_height="125sp"
   android:contentDescription="Panel with the logo of the app"
   android:src="@drawable/logo_panel2"/>


Aviso 3 - layout/content_main.xml
Missing ‘contentDescription’ attribute on image
<ImageView
   android:layout_width="wrap_content"
   android:layout_height="24dp"
   android:src="@drawable/ic_editor_border_color"/>
```

```
Refactorización 3 -  layout/content_main.xml
Añadir el atributo contentDescription con una descripción de la imágen.
<ImageView
   android:layout_width="wrap_content"
   android:layout_height="24dp"
   android:contentDescription="Image Edit"
   android:src="@drawable/ic_editor_border_color"/>


Aviso 4 - layout/nav_header_main.xml
Missing ‘contentDescription’ attribute on image
<ImageView
   android:layout_width="200sp"
   android:layout_height="125sp"
   android:src="@drawable/logo_panel2"/>

Refactorización 4 -  layout/nav_header_main.xml
Añadir el atributo contentDescription con una descripción de la imágen.
<ImageView
   android:layout_width="200sp"
   android:layout_height="125sp"
   android:contentDescription="Panel with the logo of the app"
   android:src="@drawable/logo_panel2"/>


Aviso 5 - layout/content_main.xml
Missing ‘contentDescription’ attribute on image
<ImageView
   android:layout_width="200sp"
   android:layout_height="125sp"
   android:src="@drawable/logo_panel2"/>

Refactorización 5 -  layout/content_main.xml
Añadir el atributo contentDescription con una descripción de la imágen.
<ImageView
   android:layout_width="200sp"
   android:layout_height="125sp"
   android:contentDescription="Panel with the logo of the app"
   android:src="@drawable/logo_panel2"/>
```

Android Lint: Accessibility

```
Aviso 6 - layout/content_main.xml
‘clickable’ attribute found, please also add ‘focusable’
<RelativeLayout
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:id="@+id/categories_item"
   android:layout_marginEnd="16dp"
   android:clickable="true">
```

```
Refactorización 6 -  layout/content_main.xml
Añadir el atributo focusable que permite que el elemento sea seleccionado con
<RelativeLayout
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:id="@+id/categories_item"
   android:layout_marginEnd="16dp"
   android:clickable="true"
   android:focusable="true">
```

Android Lint: Correctness

```
Aviso 7 - layout/activity_attributes_listview_items.xml
Attribute ‘drawableTint’ is only used in API level 23 and higher (current min
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/activity_attributes_listview_items_textview"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:drawableEnd="@drawable/ic_lock_black_24dp"
   android:drawableTint="@color/shadowColor"
   android:textAppearance="@style/tabs_item"
   android:background="@android:drawable/editbox_dropdown_light_frame"
   android:gravity="center_vertical"
   android:paddingStart="?android:attr/listPreferredItemPaddingStart"
   android:paddingEnd="?android:attr/listPreferredItemPaddingEnd"
```

```
   android:minHeight="?android:attr/listPreferredItemHeightSmall" />


Refactorización 7 -  layout/activity_attributes_listview_items.xml
Android ignora los atributos XML que no son soportados por un dispositivo, por



Aviso 8 - layout/app_bar_main.xml
Attribute ‘elevation’ is only used in API level 21 and higher (current min is
<android.support.v7.widget.Toolbar
   android:id="@+id/appbar"
   android:layout_width="fill_parent"
   android:layout_height="125sp"
   app:contentInsetStart="0dp"
   android:elevation="4dp"
   app:layout_scrollFlags="scroll|enterAlways">


Refactorización 8 -  layout/app_bar_main.xml
Android ignora los atributos XML que no son soportados por un dispositivo, por


Aviso 9 - es/unex/quercusseg/yourinstantapp/third_classes/GPSTracker.java
The ‘<service> es.unex.quercusseg.yourinstantapp.third_classes.GPSTracker’ is

Refactorización 9 - es/unex/quercusseg/yourinstantapp/third_classes/GPSTracker
Debemos añadir al Manifest.xml una referencia al Servicio GPSTracker:
<service android:name=".third_classes.GPSTracker" />


Aviso 10 - es/unex/quercusseg/yourinstantapp/activities/RootActivity.java
The ‘<activity> es.unex.quercusseg.yourinstantapp.third_classes.RootActivity i

Refactorización 10 - es/unex/quercusseg/yourinstantapp/activities/RootActivity
Debemos añadir al Manifest.xml una referencia al Servicio RootActivity:
<activity android:name=".activities.RootActivity" />


Aviso 11 - layout/app_bar_main.xml
Duplicate id @+id/view2, defined or included multiples times in layout/app_bar
<android.support.v7.widget.AppCompatImageButton
```

```
   android:layout_width="wrap_content"
   android:layout_height="33dp"
   android:scaleType="fitCenter"
   android:src="@drawable/ic_help_outline"
   android:textAlignment="textEnd"
   android:tint="@color/colorPrimaryDark"
   android:background="@android:color/transparent"
   android:layout_alignParentEnd="true"
   android:id="@+id/view2" />


Refactorización 11 - layout/app_bar_main.xml
Cambiar el nombre del id, ya que está duplicado en otro xml.
<android.support.v7.widget.AppCompatImageButton
   android:layout_width="wrap_content"
   android:layout_height="33dp"
   android:scaleType="fitCenter"
   android:src="@drawable/ic_help_outline"
   android:textAlignment="textEnd"
   android:tint="@color/colorPrimaryDark"
   android:background="@android:color/transparent"
   android:layout_alignParentEnd="true"
   android:id="@+id/config" />


Aviso 12 - app/build.gradle
A newer version of com.google.android.gms:play-services-location 11.0.4 is ava
A newer version of com.google.android.gms:play-services-places 11.0.4 is avail
A newer version of com.google.android.gms:play-services-maps 11.0.4 is availab

Refactorización 12 - app/build.gradle
Cambiar la versión de las dependencias a la 11.6.0.


Aviso 13 - es/unex/quercusseg/yourinstantapp/adapters/CategoriesAdapter.java
Do not treat position as fixed; only use immediately and call ‘holder.getAdapt
public void onBindViewHolder(ViewHolder holder, final int id) {
   // - get element from your dataset at this position
   // - replace the contents of the view with that element
   TextView mTextView = holder.getTextView();
   mTextView.setText(keywords.get(id).getKeyword());
   if(mAppData.getCategories().contains(keywords.get(id)))       mTextView.set
```

```
   mTextView.setOnClickListener(new View.OnClickListener() {
       @Override
       public void onClick(View v) {
           if( ! mAppData.getCategories().contains(keywords.get(id))) {
               mAppData.getCategories().add(keywords.get(id));
           }
           else {
               v.setBackground(null);
               mAppData.getCategories().remove(keywords.get(id));
           }
       }
   });

Refactorización 13 - es/unex/quercusseg/yourinstantapp/adapters/CategoriesAdap
Cambiar el holder a variable de tipo final y al id quitar el tipo final, cuand
public void onBindViewHolder(final ViewHolder holder, int id) {
   // - get element from your dataset at this position
   // - replace the contents of the view with that element
   TextView mTextView = holder.getTextView();
   mTextView.setText(keywords.get(id).getKeyword());
   if(mAppData.getCategories().contains(keywords.get(id)))    mTextView.setBac
   mTextView.setOnClickListener(new View.OnClickListener() {
       @Override
       public void onClick(View v) {
           if( ! mAppData.getCategories().contains(keywords.get(holder.getAdap
           }
           else {
               v.setBackground(null);              mAppData.getCategories().re
           }
       }
   });
}


Aviso 14 - layout/content_main.xml
The id \‘config’" is not referring to any views in this layout
<android.support.v7.widget.AppCompatImageButton
   android:id="@+id/attributes_check_button"
   android:layout_width="wrap_content"
   android:layout_height="31dp"
   android:scaleType="fitCenter"
   android:layout_marginEnd="20dp"
```

```
   android:src="@drawable/ic_action_done_custom"
   android:textAlignment="textEnd"
   android:tint="@color/colorAttributes"
   android:background="#fff"
   android:layout_toStartOf="@+id/config" />

Refactorización 14 - layout/content_main.xml
Cambiar el id por el de la vista de su layout.
<android.support.v7.widget.AppCompatImageButton
   android:id="@+id/attributes_check_button"
   android:layout_width="wrap_content"
   android:layout_height="31dp"
   android:scaleType="fitCenter"
   android:layout_marginEnd="20dp"
   android:src="@drawable/ic_action_done_custom"
   android:textAlignment="textEnd"
   android:tint="@color/colorAttributes"
   android:background="#fff"
   android:layout_toStartOf="@+id/img_categories" />


Aviso 15 - values/styles.xml
Should use \sp" instead of \dp" for text sizes.
<item name="android:textSize">16dp</item>

Refactorización 15 - values/styles.xml
Cambiar unidad del tamaño de las letras a sp
<item name="android:textSize">16sp</item>


Aviso 16 - values/styles.xml
Overriding ‘@color/material_grey_300’ which is marked as private in com.androi
<color name="material_grey_300">#E0E0E0</color>

Refactorización 16 - values/styles.xml
Eliminar el color del style.xml ya que ya estaba en las librerías de soporte d
```

Android Lint: Internationalization

```
Aviso 17 - layout/activity_categories.xml, menu/activity_main_drawer.xml, layo
Hardcoded string \***", should use ‘@string’ resource

Refactorización 17 - layout/activity_categories.xml, menu/activity_main_drawer

Eliminar el color del style.xml ya que ya estaba en las librerías de soporte d


Aviso 18 - layout/activity_categories.xml
‘@id/textView’ can overlap ‘@id/categories_check_button’ if @string/categories
android:drawableLeft="@drawable/ic_categories"
android:layout_width="wrap_content"


Refactorización 18 - layout/activity_categories.xml
Cambiar el atributo drawableLeft por drawableStart y cambiar el layout_width p
android:drawableStart="@drawable/ic_categories"
android:layout_width="match_parent"



Aviso 19 - layout/fragment_appgen_list_layout.xml
When you define ‘paddingLeft’ you should probably also define ‘paddingRight’ f
android:paddingLeft="@dimen/activity_horizontal_margin"

Refactorización 19 - layout/fragment_appgen_list_layout.xml
Añadir el atributo paddingRight.
android:paddingRight="@dimen/activity_horizontal_margin"
```

Android Lint: Performance

```
Aviso 20 - layout/fragment_appgen_list_layout.xml
Possible overdraw: Root element paints background ‘#fff’ with a theme that als


Refactorización 20 - layout/fragment_appgen_list_layout.xml
```

```
Aviso 21 - layout/nav_header_main.xml
Possible overdraw: Root element paints background ‘@drawable/logo_panel2’ with


Refactorización 21 - layout/nav_header_main.xml
La solución es sustituir el estilo AppTheme por un nuevo estilo que ha sido cr
<style name="nav_header_main" >
   <item name="android:background">@drawable/logo_panel2</item>
</style>


Aviso 22 - es/unex/quercusseg/yourinstantapp/app_gen/AppgenActivity.java
This Asynctask class should be static or leaks might occur (anonymous android.
new AsyncTask<Void, Integer, Integer>() {
   ProgressDialog progDialog = new ProgressDialog(AppgenActivity.this);
   @Override
   protected void onPreExecute() {
       super.onPreExecute();
       Log.d("APP", "Arrancando la app: " + appinfo.getApp_name() + " - Type:
       progDialog.setMessage("Downloading dataset");
       progDialog.setIndeterminate(true);
       progDialog.setProgressStyle(ProgressDialog.STYLE_SPINNER);
       progDialog.setCancelable(false);
       progDialog.show();
   }

   @Override
   protected Integer doInBackground(Void... params) {
       boolean result = false;
       try {
           result = DataContent.getInstance().getDatasetByNet(appinfo.getRepos
           if( DataContent.getInstance().getDataset().isEmpty() )
               return OpenDataService.KErrors.RC_NORESULTS;
           if (appinfo.getLayout_type().equals(BaseEntity.MAP) && result) {
               if(hasGeolocations(DataContent.getInstance().getDataset()))
                   showMapFragment();
               else
                   return KErrors.RC_NOGEOLOCATIONS;
           } else if (appinfo.getLayout_type().equals(BaseEntity.LIST) && resu
                   showListFragment();
           }
       } catch(IOException | NullPointerException e) {}
```

```
        return ( result ) ? OpenDataService.KErrors.RC_OKAY : OpenDataService.K
    }

    @Override
    protected void onPostExecute(Integer result) {
        super.onPostExecute(result);
        progDialog.dismiss();
        switch (result) {
            case KErrors.RC_CONNECTION_PROBLEMS:
                criticalError("Couldn't execute generated app because of a conn
                break;
            case KErrors.RC_NORESULTS:
                criticalError("There are no results.");
                break;
            case KErrors.RC_NOGEOLOCATIONS:
                criticalError("No results with geolocations, use list layout.")
        }
    }
}.execute();

Refactorización 22 - es/unex/quercusseg/yourinstantapp/app_gen/AppgenActivity.
Cambiar el asynctask a una subclase como indica la guía de desarrollo.
private class DownloadDatasets extends AsyncTask<Void, Integer, Integer> {
    ProgressDialog progDialog = new ProgressDialog(AppgenActivity.this);
    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        Log.d("APP", "Arrancando la app: " + appinfo.getApp_name() + " - Type:
        progDialog.setMessage("Downloading dataset");
        progDialog.setIndeterminate(true);
        progDialog.setProgressStyle(ProgressDialog.STYLE_SPINNER);
        progDialog.setCancelable(false);
        progDialog.show();
    }

    @Override
    protected Integer doInBackground(Void... params) {
        boolean result = false;
        try {
            result = DataContent.getInstance().getDatasetByNet(appinfo.getRepos
            if( DataContent.getInstance().getDataset().isEmpty() )
                return OpenDataService.KErrors.RC_NORESULTS;
```

```
            if (appinfo.getLayout_type().equals(BaseEntity.MAP) && result) {
                if(hasGeolocations(DataContent.getInstance().getDataset()))
                    showMapFragment();
                else
                    return KErrors.RC_NOGEOLOCATIONS;
            } else if (appinfo.getLayout_type().equals(BaseEntity.LIST) && resu
                showListFragment();
            }
        } catch(IOException | NullPointerException e) {}
        return ( result ) ? OpenDataService.KErrors.RC_OKAY : OpenDataService.K
    }

    @Override
    protected void onPostExecute(Integer result) {
        super.onPostExecute(result);
        progDialog.dismiss();
        switch (result) {
            case KErrors.RC_CONNECTION_PROBLEMS:
                criticalError("Couldn't execute generated app because of a conn
                break;
            case KErrors.RC_NORESULTS:
                criticalError("There are no results.");
                break;
            case KErrors.RC_NOGEOLOCATIONS:
                criticalError("No results with geolocations, use list layout.")
        }
    }
}
```

```
Aviso 23 - es/unex/quercusseg/yourinstantapp/services/DbService.java
Do not place Android context classes in static fields (static reference to ‘Db
private static DbService instance;
private Context context;
public DbService() {
    this.context = YourInstantApplication.appContext;
}
```

```
Refactorización 23 - es/unex/quercusseg/yourinstantapp/services/DbService.java
La solución es eliminar la variable contexto y en cada referencia a la misma,
SQLiteDatabase db = new DaoMaster.DevOpenHelper(YourInstantApplication.appCont
```

```
Aviso 24 - activity_login.xml, colors.xml, common_plus_signin_btn_icon_dark.xm
```

```
The resource ‘***’ appears to be unused
Refactorización 24 - activity_login.xml, colors.xml, logo_header.png, strings.
Eliminando los recursos no utilizados desaparecen los avisos.
```

Android Lint: Security

```
Aviso 25 - app/src/main/AndroidManifest.xml
On SDK version 23 and up, your app data will be automatically backed up and re

Refactorización 25 - app/src/main/AndroidManifest.xml
Añadiendo el atributo fullBackupContent se soluciona.
android:fullBackupContent="false"
```

Android Lint: Usability

```
Aviso 26 - res/drawable/**.png
Found bitmap drawable ‘***.png’ in desityless folder

Refactorización 26 - res/drawable/**.png
Poniendo cada imagen con diferentes tamaños en cada carpeta drawable-**dpi par

Aviso 27 - app/src/main/AndroidManifest.xml
App is not indexable by Google Search; consider adding at least one Activity w

Refactorización 27 - app/src/main/AndroidManifest.xml
Añadiendo un link a la página web de tu app consigues que Google pueda rastrea
```

Class structure

```
Aviso 28 - AppgenActivity.java, CategoriesActivity, CategoriesPresenter, DataP
Field can be converted to a local variable
```

```
Refactorización 28 - es/unex/quercusseg/yourinstantapp/app_gen/AppgenActivity.
Convertir las variables en locales, ya que no necesitan ser atributos de la cl
```

Code maturity issues

```
Aviso 28 - ListLayoutFragment.java, RootActivity.java
Deprecated API usage
final MenuItem item = menu.findItem(R.id.action_search);
final SearchView searchView = (SearchView) MenuItemCompat.getActionView(item);

MenuItemCompat.setOnActionExpandListener(item,  new
MenuItemCompat.OnActionExpandListener() { }

Refactorización 28 - ListLayoutFragment.java, RootActivity.java
Cambiar esas funciones por otras admitidas:
final MenuItem item = menu.findItem(R.id.action_search);
final SearchView searchView = (SearchView) item.getActionView();

item.setOnActionExpandListener( new MenuItem.OnActionExpandListener() {}

Aviso 29 - SettingsActivity.java
‘addPreferencesFromResource(int)’ is deprecated as of API 16:Android 4.1 (Jell
‘findPreference(java.lang.CharSequence)’ is deprecated as of API 16:Android 4.
addPreferencesFromResource(R.xml.preference_settings);
PreferenceScreen loggedAs = (PreferenceScreen)findPreference("logged_as");

Refactorización 29 - SettingsActivity.java
No hay métodos alternativos para estas funciones, en su lugar se ha modificado
@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   getFragmentManager().beginTransaction().replace(android.R.id.content, new S
}
public static class SettingsFragment extends PreferenceFragment {
   @Override
   public void onCreate(final Bundle savedInstanceState)   {
       super.onCreate(savedInstanceState);
       addPreferencesFromResource(R.xml.preference_settings);
   }
```

```
}
```

Code style issues

```
Aviso 30 - es/unex/quercusseg/yourinstantapp/third_classes/Locator.java
Modifier ‘static’ is redundant for inner enums
static public enum Method {
   NETWORK,
   GPS,
   NETWORK_THEN_GPS
}

Refactorización 30 - es/unex/quercusseg/yourinstantapp/third_classes/Locator.j
Eliminando el modificador static se soluciona el problema.
public enum Method {
   NETWORK,
   GPS,
   NETWORK_THEN_GPS
}


Aviso 31 - es/unex/quercusseg/yourinstantapp/services/DataSource.java, es/unex
Modifier ‘public’ is redundant for interface methods

Refactorización 31 - es/unex/quercusseg/yourinstantapp/services/DataSource.jav
Eliminando el modificador public se soluciona el aviso.


Aviso 32 - AppsDao.java, ListLayoutFragment, MainActivity, MappingDao, RepoDao
Unnecessary semicolon ‘;’

Refactorización 32 - AppsDao.java, ListLayoutFragment, MainActivity, MappingDa
Eliminando los ‘;’ innecesarios se soluciona.
```

Compiler issues

```
Aviso 33 - es/unex/quercusseg/yourinstantapp/activities/DataActivitySelectorRe
Unchecked cast: ‘java.io.Serializable’ to ‘java.util.HashSet<es.unex.quercusse
repos = (HashSet<Repo>)getArguments().getSerializable("repos");

Refactorización 33 - es/unex/quercusseg/yourinstantapp/activities/DataActivity
El compilador no conoce si el objeto serializable obtenido es de tipo HashSet
if(getArguments().getSerializable("repos") instanceof HashSet){
   repos = (HashSet<Repo>)getArguments().getSerializable("repos");
}
```

Control flow issues

```
Aviso 34 - es/unex/quercusseg/yourinstantapp/app_gen/AppgenActivity.java, es/u
‘return’ is unnecessary as the last statement in a ‘void’ method

Refactorización 34 - es/unex/quercusseg/yourinstantapp/app_gen/AppgenActivity.
Eliminar los ‘returns’ innecesarios.
```

Data flow issues

```
Aviso 35 - AppsDao.java, AttributesAdapter.java, BaseEntity.java, CategoriesAd
Local variable ‘***’ is redundant
Refactorización 35 - AppsDao.java, AttributesAdapter.java, BaseEntity.java, Ca
Eliminar las variables innecesarias y comprimir el código
```

Declaration redundancy

```
Aviso 36 - es/unex/quercusseg/yourinstantapp/activities/DataActivity.java
Static member ‘es.unex.quercusseg.yourinstantapp.services.RestService.getSchem
RestService restService = new RestService();
   List<Schema> schemas = restService.getSchemasForRepository(repo.getId().int

Refactorización 36 - es/unex/quercusseg/yourinstantapp/activities/DataActivity
```

```
Eliminar la variable restService y referenciar mediante la clase static.
List<Schema> schemas = RestService.getSchemasForRepository(repo.getId().intVal

Aviso 37 - es/unex/quercusseg/yourinstantapp/dao/DaoMaster.java, DataPresenter
Actual method parameter \***" is the same constant

Refactorización 37 - es/unex/quercusseg/yourinstantapp/dao/DaoMaster.java
Modificar los parámetros para quitar los valores constantes.

Aviso 38 - AppgenActivity.java, AppRepository.java, Attribute.java, BaseEntity
Declaration access can be weaker

Refactorización 38 - AppgenActivity.java, AppRepository.java, Attribute.java,

Cambiar las declaraciones de acceso para conseguir mejor encapsulación.

Aviso 39 - AppgenActivity.java, AttributesAdapter.java, AttributesFragment.jav
Declaration can have final modifier

Refactorización 39 - AppgenActivity.java, AttributesAdapter.java, AttributesFr
Añadir los modificadores 'final' en las declaraciones de estas variables.

Aviso 40 - es/unex/quercusseg/yourinstantapp/model/BaseEntity.java
Redundant default parameter value assignment
@Expose(serialize = true)

Refactorización 40 - es/unex/quercusseg/yourinstantapp/model/BaseEntity.java
Eliminar los parámetros redundantes.
@Expose()

Aviso 41 - DataActivitySelectorRepoFragment.java, ListLayoutFragment.java, Loc
Method only calls its super
All implementations of this method are empty

Refactorización 41 - DataActivitySelectorRepoFragment.java, ListLayoutFragment
Eliminar los métodos que no tienen ninguna utilidad.
```

Aviso 42 - BaseEntitiy.java, DataContent.java, GPSTracker.java, MainView.java
Return value of the method is never used

Refactorización 42 - BaseEntitiy.java, DataContent.java, GPSTracker.java, Main
Eliminar los métodos que no tienen ninguna utilidad.

Aviso 43 - AppRepository.java, AppsDao.java, Attribute.java, AttributesAdapter
Unused declaration

Refactorización 43 - AppRepository.java, AppsDao.java, Attribute.java, Attribu
Estos métodos no se usan actualmente, pero en el futuro es probable que se use

Aviso 44 - AppgenActivity.java, ConnectionUtilities.java, RestService.java, Ro
Empty ‘catch’ block

Refactorización 44 - AppgenActivity.java, ConnectionUtilities.java, RestServic
Capturar los errores de forma correcta, añadiendo información en las cláusulas

General

Aviso 45 -
Default File Template

Refactorización 45 -
Modificar las cabeceras de los archivos .java.

Imports

Aviso 46 -
Unused Import

Refactorización 46 -
Eliminar los ‘imports’ no utilizados.

Java 7

```
Aviso 47 - BaseEntity.java, DataActivitySelectorRepoFragment.java, DataActivit
Explicit type can be replaced with <>

Refactorización 47 - BaseEntity.java, DataActivitySelectorRepoFragment.java, D
Eliminar los argumentos redundantes.
```

Java language level migration aids

```
Aviso 48 - es/unex/quercusseg/yourinstantapp/model/intercom/ListFields.java
'if' statement replaceable with 'switch' statement
if(field.getType().equals("title")) field_title = field.getField();
else if(field.getType().equals("geolocation")) field_geolocation = field.getFi
else if(field.getType().equals("pkey")) field_id = field.getField();
else ordinary_fields.add(field);

Refactorización 48 - es/unex/quercusseg/yourinstantapp/model/intercom/ListFiel
Cambiar el if por un switch:
switch (field.getType()) {
   case "title":
       field_title = field.getField();
       break;
   case "geolocation":
       field_geolocation = field.getField();
       break;
   case "pkey":
       field_id = field.getField();
       break;
   default:
       ordinary_fields.add(field);
       break;
}
```

Javadoc issues

```
Aviso 49 - AttributesAdapter.java, Field.java, RestService.java
Dangling JavaDoc comment

Refactorización 49 - AttributesAdapter.java, Field.java, RestService.java
Eliminar los comentarios sin sentido.


Aviso 50 - AppsDao.java, MappingDao.java, RepoDao.java, SchemaDao.java
Wrong tag ‘inheritdoc’

Refactorización 50 - AppsDao.java, MappingDao.java, RepoDao.java, SchemaDao.ja
Eliminar la etiqueta inheritdoc que ya no se utiliza.


Aviso 51 - Connectivity.java, RestService.java
‘@param context’ tag description is missing
‘@return’ tag description is missing

Refactorización 51 - Connectivity.java, RestService.java
Eliminar la etiqueta inheritdoc que ya no se utiliza.
```

Numeric issues

```
Aviso 52 - AppsDao.java, MappingDao.java, RepoDao.java, SchemaDao.java
‘offset + 0’ can be replaced with ‘offset’

Refactorización 52 - AppsDao.java, MappingDao.java, RepoDao.java, SchemaDao.ja
Reemplazar ‘offset + 0’ por ‘offset’ ya que es lo mismo pero más simplificado
```

Performance issues

```
Aviso 53 - ConnectionUtilities.java
```

```
String concatenation as argument to ‘StringBuilder.append()’ call
sb.append(line + "\n");


Refactorización 53 - ConnectionUtilities.java
Reemplazar con llamadas encadenadas a ‘append()’
sb.append(line).append("\n");
```

Probable Bugs

```
Aviso 54 - ConnectionUtilities.java, DataActivity.java, DataPresenter.java, Ma
Method invocation ‘***’ may produce ‘java.lang.NullPointerException’

Refactorización 54 - ConnectionUtilities.java, DataActivity.java, DataPresente
Añadir comprobaciones antes de utilizar métodos que puedan producir ‘java.lang


Aviso 55 - BaseEntity.java
Contents of collection ‘selected_categories’ are updated, but never queried.
Contents of collection ‘selected_markers’ are updated, but never queried.

Refactorización 55 - BaseEntity.java
El problema es que estas variables tienen datos pero no se usan para nada, la


Aviso 56 - DataActivity.java, MainActivity.java, Repo.java
Not annotated parameter overrides @NonNull parameter

Refactorización 56 - DataActivity.java, MainActivity.java, Repo.java
La solución es añadir la anotación @NonNull


Aviso 57 - Utils.java
Result of ‘file.delete()’ is ignored

Refactorización 57 - Utils.java
La solución es añadir la anotación @NonNull
```

Aviso 58 - DataActivity.java, DetailsFragment.java, GPSTracker.java ‘**’ state

Refactorización 58 - DataActivity.java, DetailsFragment.java, GPSTracker.java
La solución es eliminar el código vacío o añadir sentencias.

Aviso 59 - ConnectionUtilities.java, RestService.java, Utils.java
Variable ‘**’ initializer ‘**’ is redundant

Refactorización 59 - ConnectionUtilities.java, RestService.java, Utils.java
La solución es eliminar el código redundante.

Verbose or redundant code constructs

Aviso 60 - AppgenActivity.java, AttributesAdapter.java, AttributesFragment.jav
Casting ‘findViewById(****) to ***" is redundant

Refactorización 60 - AppgenActivity.java, AttributesAdapter.java, AttributesFr
La solución es eliminar los castings redundante.

XML

Aviso 61 - google_maps_api.xml
Namespace is not bound

Refactorización 61 - google_maps_api.xml
Se ha borrado el xml puesto que no se usaba

Aviso 62 - activity_attributes_listview_items.xml, activity_maps.xml, details_
Namespace declaration is never used
xmlns:tools="http://schemas.android.com/tools"

Refactorización 62 - activity_attributes_listview_items.xml, activity_maps.xm
Borrar la declaración

```
Aviso 63 - arrays.xml, integers.xml, refs.xml
XML tag has empty body

Refactorización 63 - arrays.xml, integers.xml, refs.xml
Borrar los archivos xml que no se estaban usando.
```