Universidad Zaragoza
1542

Proyecto Fin de Carrera

Aplicación nativa ANDROID para la localización de autobuses más frecuentemente usados en Londres

Autora

Ana San Juan Andollo

Director

Víctor Viñals Yúfera

Departamento de Informática e Ingeniería de Sistemas
Universidad Zaragoza

Escuela de Ingeniería y Arquitectura

Septiembre 2017

Escuela de Ingeniería y Arquitectura
Universidad Zaragoza

Dedicado a mi padre

Agradecimientos:

A Víctor, porque durante estos 8 años ha tenido una paciencia infinita conmigo y me ha llamado a la puerta hasta que por fin lo hemos conseguido.

A Frankie por tu ayuda y por darme el empujón que necesitaba.

A Iván y a Lucía, por su apoyo, su compresión y su ayuda incondicional.

A Víctor C., porque desde que te dije lo del proyecto, me has ayudado y me has facilitado las cosas.

A mi familia y amigas por estar siempre a mi lado.

RESUMEN

Según estudios de Google, en 2012 el 41% de españoles tenían un smartphone, ahora en 2017, el porcentaje ha pasado a ser del 81%. Estos mismos estudios también indican que en torno a un tercio de la población mundial accede a internet a través de sus smartphones con más frecuencia que con un ordenador. Es por tanto innegable el dominio de este tipo de dispositivos y el uso de sus aplicaciones y servicios que, en definitiva, facilitan la vida al usuario. De la misma manera y cubriendo la demanda de aplicaciones móviles, el desarrollo de aplicaciones para este tipo de dispositivos ha crecido exponencialmente lo que ha propiciado una necesidad de desarrolladores tanto de Android como IOS.

Aplicaciones como la desarrollada en este PFC pretenden dar un servicio al usuario que, aunque normalmente disponible a través de otros medios, no son tan accesibles y no tienen la misma disponibilidad que un dispositivo móvil que se puede llevar en el bolsillo. También existen otras aplicaciones dentro del mismo marco como son "*Live London Bus Tracker*" o "*Bus London – Times and routes*" que aunque detallan información sobre rutas y planificaciones de viajes, no están orientadas a usuarios que frecuentemente usan los mismos autobuses y quieren simplemente ver de una manera fiable, rápida y en tiempo real la información de una línea en una parada en concreto.

Para el desarrollo del proyecto se han tratado de seguir los pasos lógicos y de la manera más real posible, que se habrían seguido para desarrollar un producto en un marco de trabajo moderno. Desde el desarrollo de un plan de proyecto que justificaría llevarlo a cabo, como la definición funcional y desarrollo del mismo. El empleo de los principios definidos en las metodologías ágiles, así como el empleo de la última arquitectura de componentes definida por Google para el desarrollo de aplicaciones Android han ayudado al autor a profundizar y ampliar conocimientos en áreas y tecnologías que se encontraban fuera del ámbito habitual de su trabajo.

TABLA DE CONTENIDOS

I. INTRODUCCIÓN

1 Motivación

Aunque existen distintas aplicaciones en el mercado que ofrecen todo tipo de información sobre los autobuses de la ciudad de Londres, facilitada por la "Transport For London Unified API", éstas no se enfocan en usuarios de autobús cuya rutina les lleva a usar con frecuencia la misma línea de autobús que cogen en las mismas paradas. Esta aplicación, que he llamado **buStop**, pretende llenar ese hueco, ofreciendo de manera rápida y requiriendo la mínima interacción posible del usuario, la información que necesita, cuándo la necesita y por supuesto de forma segura y en tiempo real.

2 Objetivos

El objetivo fundamental de este trabajo de fin de grado consiste en el desarrollo de una aplicación móvil para Android que, haciendo uso de la geolocalización ubique al usuario y le presente la información de su parada de autobús preferida en la zona.

A partir de este objetivo funcional, se considera que la elaboración de este PFC ayudará al autor a adquirir conocimientos y experiencia en las siguientes áreas:

1. Diseño, Desarrollo y Arquitectura Android.
2. Uso de REST API.
3. Uso de persistencia.
4. Metodologías agiles, en concreto Scrum.
5. Planificación y gestión de proyectos de Software.

3 Metodología

Dada la aproximación que se ha tomado con el proyecto en la que partimos de una idea que se ha ido refinando y ampliando a medida que se ha avanzado con el desarrollo y para poder hacer un estudio de las metodologías ágiles que actualmente se están imponiendo al uso de otras metodologías tradicionales, se ha decidido adoptar el modelo de desarrollo ágil Scrum.

4 Android

La primera decisión que se tomó fue si desarrollar la aplicación en un entorno Nativo o por el contrario usar una alternativa multi-plataforma, como por ejemplo **Apache Cordova** [W3ACOR], un *framework open-source* para desarrollo móvil que permite usar tecnología web standard como es HTML5, CSS3 o *Javascript*. Esta opción se descartó, puesto que uno de los objetivos del proyecto era adquirir nuevos conocimientos, como lo sería la programación Nativa y además porque el rendimiento de la aplicación puede resentirse. Por tanto, se eligió realizar el proyecto en un entorno Nativo.

La siguiente decisión que hubo que tomar fue seleccionar Android o iOS. La decisión de desarrollar la aplicación para Android, se tomó basada en el estudio realizado de ambas tecnologías, la Tabla 1 expone los PROS y CONTRAS desprendidos del estudio. En base a este estudió se optó por Android.

	Android	IOS
PROS	De acuerdo con Statista [W3STAT], Android domina el mercado de móviles, con el 86.1% del mercado en el primer cuarto de 2017. Al ser código libre, ofrece a los desarrolladores más posibilidades de creación, de *plugins*, *frameworks*... dando como resultado una plataforma mucho más flexible, posibilitando el uso desarrollos ajenos y beneficiarse así de la comunidad para crear mejoras. Sin embargo, por esto mismo, desarrollar en Android puede ser mucho más complejo y puede costar más llegar a ser un experto. El IDE para el desarrollo de aplicaciones de Android es Android Studio basado en *IntelliJ*, que es una herramienta muy robusta y con la que la autora está familiarizada, con lo cual puede centrar los esfuerzos en lo realmente importante. Android ha lanzado unas claras pautas a seguir para el diseño lo que hace que las aplicaciones en Android resulten más usables e intuitivas, aunque también implican más trabajo por parte de los desarrolladores.	Plataforma más estable y exclusiva (guías estrictas que como resultado dan unos diseños de aplicaciones rápidos, con una mejor respuesta y en general menos expuestas a piratería). En esto también influye el hecho de que solo existen unos 20 tipos diferentes de móviles iOS y no cientos como en el caso de Android. Esto es lo que se conoce como fragmentación de los dispositivos, a mayor fragmentación, mayor dificultad para conseguir un diseño que se adapte a todos los dispositivos de manera óptima. Como resultado de esto, el tamaño de las pantallas y la resolución juegan un papel muy pequeño en el proceso de desarrollo con iOS. De hecho, desarrollando una aplicación para iOS compatible con las 3 últimas actualizaciones cubriríamos aproximadamente el 90% de los usuarios iOS, como se puede ver en un detallado estudio encontrado en el blog de David Smith un experto en IOS [W3STAT]. Los usuarios de iOS, históricamente, gastaban más dinero en comprar aplicaciones que los usuarios de Android, aunque según la firma de análisis de aplicaciones móviles "Annie" [W3STAT], a partir de este año y siguiendo esta tendencia hasta el 2021, la combinación de los ingresos de Android App Store junto con Google Play va a generar más ingresos que la App Store de iOS, aunque ésta, por separado, siga generando más.
CONS	• Aunque Android sea libre tiene sus beneficios en cuanto al alcance que puede dar a tu aplicación, también supone un reto por la cantidad de dispositivos y resoluciones de pantalla que la aplicación debe ser capaz de resolver. Debido a esta fragmentación, Android suele requerir más tiempo de desarrollo que iOS • Del mismo modo que iOS obliga a usar ciertas herramientas.	• Plataforma mucho más restrictiva para sus desarrolladores, no permitiendo algunas de las personalizaciones que el desarrollador quiere realizar o incrementando el coste del desarrollo de las mismas con licencias. iOS se considera un sistema operativo más maduro, con standard predefinidos y reglas. Estos estándares pueden hacer más dificultoso la puesta en producción de una aplicación. • Estás atado a unas herramientas para desarrollar, se requiere el IDE Xcode, con el SDK de iOS. Este soporta muchos lenguajes siendo el más elegido por los desarrolladores Swift, ya que es un lenguaje creado por Apple y que aparentemente es menos propenso a errores. *Swift usa Cocoa Touch* como el API para construir la IU [W3APL]. • Desarrollos más costosos por las herramientas en las que se tiene que desarrollar y testear la aplicación. • iOS no tiene unas pautas tan desarrolladas como Android en temas de diseño de aplicaciones.

Tabla 1. Pros y contras del desarrollo en Android e IOS.

5 Estructura de la Memoria

Esta memoria de proyecto se ha estructurado en los siguientes capítulos:

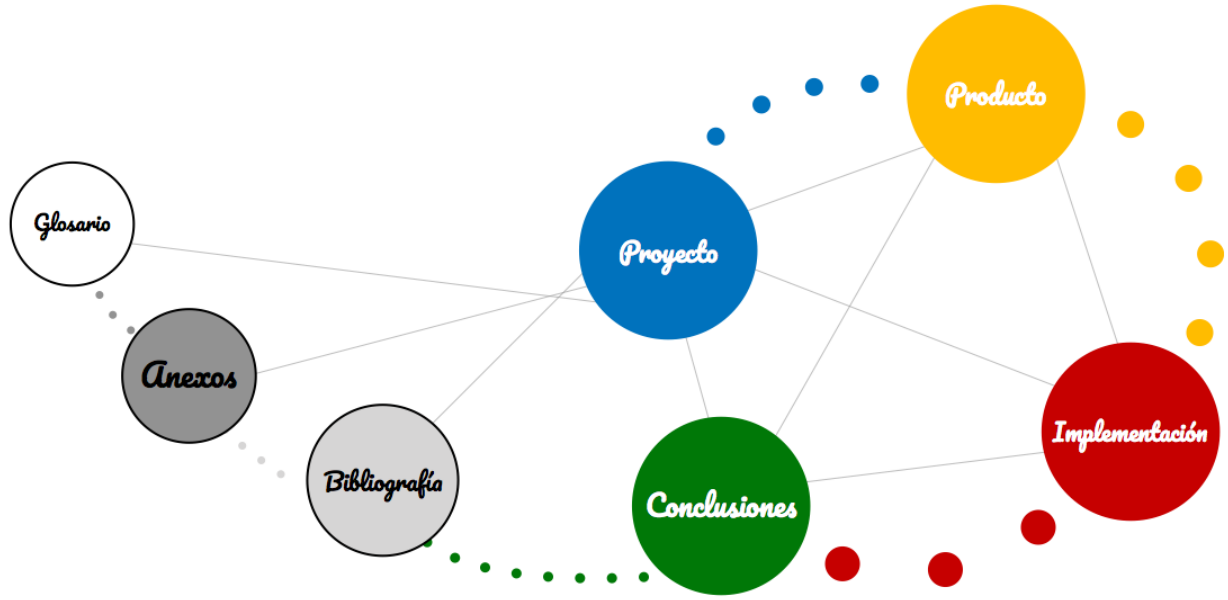


Figura 1. Visión gráfica del Proyecto, que refleja la relación entre los distintos capítulos de la memoria, y como del plan de Proyecto, se pasó a una definición del Producto con la cual se llevó a cabo una implementación, y las conclusiones desprendidas tras finalizar.

- ***PROYECTO***, visión estratégica y organizativa del proyecto, indicando cómo desde la idea inicial de construir una aplicación Android para el control de los horarios de autobuses se pretende llegar al Producto final que será la aplicación ***buStop***.
- ***PRODUCTO***, en el que se describirán las características de la aplicación a través de la elaboración y análisis de las historias de usuario, los criterios de aceptación, así como los ***user journeys*** que compondrán el ***Product Backlog***.
- ***IMPLEMENTACIÓN***, este capítulo detalla los distintos ***sprints*** que se han llevado a cabo durante el desarrollo del producto, indicando las distintas fases ejecutadas en cada uno de ellos: planificación, riesgos, desarrollo, revisión del desarrollo y retrospectiva.
- ***CONCLUSIONES***, contiene la valoración del trabajo realizado a modo de retrospectiva planteando los problemas y el valor obtenido durante la realización del proyecto.
- ***BIBLIOGRAFÍA Y RECURSOS.***
- ***ANEXOS***, el capítulo incluye la documentación complementaria que se considera necesaria para la comprensión del proyecto. Se han incluido los siguientes anexos: ANEXO I: Configuración entorno de desarrollo, ANEXO II: Arquitectura de Componentes, ANEXO III: Métricas, ANEXO IV: Diagrama de GANTT, ANEXO V: API de Transporte y ANEXO VI: Metodología.
- ***GLOSARIO***, definición de términos específicos empleados en el documento.

II. PROYECTO

1 Modelo de negocio

Previo al desarrollo de la aplicación, se ha creado un modelo de negocio para determinar qué se va a ofrecer al mercado, cómo se va a hacer, quién va a ser su público y qué método se empleará para generar un beneficio. Para ello me he servido de una plantilla negocio llamada "***Business Model Canvas***" que es una de las más populares y empleadas por las nuevas *startups*. Su definición se recoge en el libro "*Business model generation*" escrito por Alexander Osterwalder & Yves Pigneur [LOsPi2010].

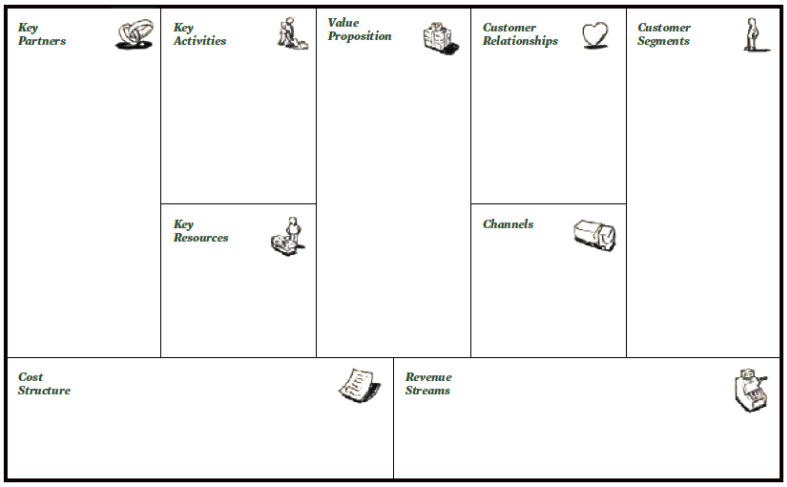


Figura 2. Plantilla de plan de proyecto "Business Model Canvas", se detallan los apartados: Segmentos de Clientes, Propuesta de valor, Canales, Relación con el cliente, Fuente de ingresos, Recursos clave, Actividades clave, Socios clave y Estructura de costes. Tomado de [LOsPi2010].

1.1 Segmento de clientes

La aplicación buStop estará destinada a un perfil de usuario muy específico que cumpliría con las siguientes características: usuario de teléfono móvil con Android, qué emplea el autobús en Londres cómo medio de transporte habitual y que suele usar ciertas líneas que coge siempre en las mismas paradas. Por tanto, se espera un perfil de usuario de entre 16 y 40 años que emplea siempre las mismas rutas para acudir a su lugar de trabajo o estudios y volver a casa.

1.2 Propuesta de valor

A pesar de que existen numerosas aplicaciones mediante las cual consultar el estado de los autobuses en Londres, buStop pretende diferenciarse ofreciendo la información de manera rápida y sencilla sin que el usuario tenga que intervenir de ninguna manera con la aplicación, es decir, simplemente con abrir la aplicación sepa cuanto falta para que su autobús pase por su parada. Mediante un sistema de favoritos, el sistema geo posicionará al usuario y, de entre sus favoritos, le presentará la información del autobús en la parada más cercana.

1.3 Canales

El principal canal mediante el cual la aplicación se ofrecerá a los clientes será mediante la Play store de google. La aplicación se ofrecerá al usuario de forma gratuita.

1.4 Relación con el cliente

El cliente podrá descargar la aplicación e instalarla en su terminal y podrá ofrecer *feedback* y valoraciones desde la *"Play store*". También se ofrecerá una canal de twitter mediante el que los usuarios podrán contactar para proponer mejoras o reportar fallos. *https://twitter.com/stop_bu.*

1.5 Fuente de ingresos

La aplicación se ha creado sin ánimo de lucro, pero en base al éxito de la aplicación se estudiará la posibilidad de incluir publicidad la cuál en base al número de usuarios podrá reportar un beneficio.

1.6 Recursos clave

Se consideran recursos claves la propia API de transporte de Londres, la cual puede sufrir actualizaciones y modificaciones que pueden llegar a impactar al funcionamiento de la aplicación, así como estar sujeto a su disponibilidad. De la misma forma se considera clave la disponibilidad de la Play Store de Google, desde la cual los usuarios podrán descargar la aplicación.

1.7 Actividades clave

La actividad en la que se basa la aplicación es en el servicio de autobuses de Londres.

1.8 Socios clave

No se ha identificado la necesidad de recurrir a personal externo para el desarrollo ni implantación de la aplicación.

1.9 Estructura de costes

El estudio económico llevado a cabo, se ha dividido en costes de personal y equipo informático, no se tendrá en cuenta el coste de ocupación dado que se ha desarrollado desde el domicilio del autor, ni el gasto en software y licencias ya que se han usado herramientas de software libre y por tanto a coste 0.

Aunque el proyecto ha sido realizado por una persona, se considerarán los distintos roles que habrían sido necesarios para el desarrollo de una aplicación Android, así como una estimación de tarifas basada en la información publicada en portales de empleo cómo "www.jobserve.com" [W3JS]. Del mismo modo se incluye el gasto incurrido en equipo informático.

1.9.1 Personal

Dado que el proyecto se ha desarrollado en Reino Unido se emplearán recursos contratados como "*Contractors*", una modalidad que podría ser semejante a contratar en España y se considerarán las tarifas, moneda e impuestos (VAT) usados en el país, ver Tabla 2.

Rol	Tarifa diaria	VAT (20%)	Total	Total por hora
Product Owner	£400.00	£ 80.00	£ 480.00	£ 60.00
Business Analyst	£300.00	£ 60.00	£ 360.00	£ 45.00
Android Developer	£300.00	£ 60.00	£ 360.00	£ 45.00
Tester	£250.00	£ 50.00	£ 300.00	£ 37.50

Tabla 2. Tarifas por personal.

La justificación de los recursos de personal empleados y sus tareas se desarrolla en mayor detalle en el punto "2. Estrategia". A grandes rasgos, dada la complejidad, extensión y metodología del proyecto se decidió formar un equipo con los recursos indicados en la siguiente tabla y mantenerlos a todos durante la duración del proyecto, ver Tabla 3.

Rol	Recursos	Tarifa/hora	Horas	Horas totales	Coste Total
Product Owner	1	£60.00	85	85	£5100.00
Business Analyst	1	£45.00	85	85	£3825.00
Android Developer	2	£45.00	85	170	£7650.00
Tester	1	£37.50	85	85	£3187.50
TOTAL			**340**	**425**	**£19762.50**

Tabla 3. Coste del equipo humano.

1.9.2 Equipo informático

Para los costes de equipo informático, he tenido en cuenta los activos empleados durante la realización del proyecto, así como el tiempo de vida estimado y el porcentaje del valor amortizado durante el tiempo de desarrollo del proyecto, ver Tabla 4.

Activo	Precio	Vida (meses)	Tiempo usado (meses)	Coste
Ordenador Portátil	£1294.93	36	4	£143.88
Monitor	£179.00	36	4	£19.89
Teléfono móvil	£109.00	24	4	£18.17
TOTAL				**£181.94**

Tabla 4. Gasto en equipo informático.

1.9.3 Total

El coste total asciende por tanto a casi 20000 libras, según muestra la Tabla 5.

Activo	Precio
Personal	£19762.50
Equipo personal	£181.94
TOTAL	**£19944.44**

Tabla 5. Gasto total.

2 Estrategia

Tras el estudio del modelo de negocio se detalla la estrategia, a distintos niveles, llevada a cabo para la ejecución del proyecto.

2.1 Metodología

Para facilitar el éxito de un proyecto es fundamental contar con una metodología claramente definida y comprendida dentro de nuestro equipo de desarrollo. Para la realización de este PFC se ha decidido emplear metodología ágil, en concreto Scrum. El "ANEXO VI: Metodología" contiene un estudio detallado de la metodología Scrum junto a comparativas con otras metodologías tradicionales y ágiles, en el cual se detallan los motivos por los que se ha decidido usar Scrum sobre otras.

¿Por qué Ágil? El principal motivo por el que se ha decidido gestionar el proyecto de forma ágil es que no se contará con todos los requisitos desde el primer momento y estos pueden estar sujetos a cambios durante el desarrollo del mismo.

Entre las metodologías ágiles, Scrum es una de las más usadas por su relativa sencillez con respecto a otras metodologías del mismo tipo más complejas como XP (*Extreme programming*) pero que, frente a otras más simples, como por ejemplo Kanban, es lo suficientemente prescriptiva como para definir una serie de reglas que servirán de guía y ayudarán al equipo a cumplir con los objetivos. Por la experiencia de la autora, Scrum funciona mejor para equipos menos maduros en los que las restricciones que impone la metodología facilita que no se caigan en errores que desvíen al equipo de seguir los principios de Agile, mientras que equipos con mayor experiencia en uso de metodologías ágiles, tienden a adaptarla a su forma de producir, derivando en muchas ocasiones a Kanban o a un punto intermedio conocido popularmente como "Scrumban".

A continuación, se exponen los conceptos más importantes de Scrum y cómo se emplearán en el proyecto:

- **Sprint:** Scrum define un proceso iterativo e incremental, es decir, el desarrollo se divide en iteraciones denominadas Sprints y al completar cada iteración se genera un incremento del producto final, que es posible probar y demostrar en sí mismo al finalizar dicha iteración. Se debe definir la duración del Sprint que debe ser la misma para todos los sprints del proyecto. Por lo general en Scrum no deberían ser inferiores a dos semanas para poder generar un incremental suficientemente completo, ni superior a cuatro semanas para evitar perder la visibilidad del incremental. No existe una duración óptima de Sprint, sino que cada equipo y cada proyecto deben determinar que se ajusta mejor a su forma de trabajo basándose en ciertos factores, como tipo de desarrollo, incrementales, velocidad de desarrollo, disponibilidad… Tras analizar el trabajo, el tiempo y la disponibilidad para el desarrollo de la aplicación BuStop se ha decidido una duración de Sprint de tres semanas.

- **Backlog:** un backlog no es más que una lista de tareas o requisitos. Lo que diferencia el backlog en Scrum del resto es que cada entrada del backlog tiene que aportar valor al cliente, están priorizadas por el valor que aportan, tienen una estimación asociada y no contiene acciones ni tareas de bajo nivel. En Scrum se diferencian dos tipos de backlog:

- ***Product Backlog***. Contiene todos los requisitos, en forma de historias de usuario, que definirán la funcionalidad del producto. El propietario o responsable del Product Backlog es el Product Owner, aunque todo el equipo puede contribuir. No es algo fijo, sino que puede, y se espera, que se amplíe o cambie a lo largo del proyecto.

- ***Sprint Backlog***. El equipo decide qué elementos del Product Backlog se abordarán en la iteración, y en base a ellos generará la lista de tareas del Sprint Backlog. Estas tareas estarán debidamente estimadas y asignadas.

- ***Otros artefactos.*** Se permite usar otros artefactos como por ejemplo *workflow descriptions*, *user interface guidelines*, *storyboards, o user interface prototypes*, que, aunque no pueden en ningún caso reemplazar el backlog, sirven para complementar su contenido.

- **Estimación:** se pueden usar distintos métodos de estimación, horas, jornadas, tallas de camiseta (S,M,L,XL)… En este proyecto se estimará usando puntos de historia, *Story Points* (0 - 0,5 - 1 - 2 - 3 - 5 - 8 - 13 - 20). Se estima que el tope por iteración serán 20, si una historia fuera mayor se deberá descomponer.

- **Scrum Board:** para el PFC se ha usado una plataforma online colaborativa llamada Trello "https://trello.com" en la que se ha generado una pizarra (*board*) para gestionar las tareas.

- **Velocidad:** se entiende por velocidad la capacidad de desarrollo del equipo, en este caso se estima que la velocidad, es decir, la cantidad de *story points* que podrá ejecutar el equipo por iteración, será de 20 *story points* por sprint.

- **Equipo**: cómo se indicaba en el punto "1.9 Estructura de costes" el proyecto ha sido desarrollado por una sola persona que ha tomado el rol de cada uno de los miembros del equipo en las distintas tareas realizadas. El equipo está formado por:

  - **Product Owner,** que se encargará de validar la funcionalidad y en definitiva será la persona que entienda que necesitan los usuarios del sistema.

  - **Scrum Master,** el experto en Scrum que se encargará de las labores de facilitación, y ayudará al equipo a seguir la metodología.

  - **Business Analyst:** aunque en la metodología Scrum no existe un puesto específico para el *business Analyst*, se considera cómo un rol necesario. Actúa como nexo entre el Product Owner y el equipo técnico, traduciendo las necesidades de negocio en requisitos o historias de usuario, generando y manteniendo las historias de usuario y colaborando en la definición del plan de pruebas. En ciertos proyectos puede llegar a actuar como Product Owner o formar parte del equipo de desarrollo.

  - **Equipo de desarrollo:** 2 desarrolladores Android, uno con mayor conocimiento en el área de UX (Experiencia de usuario) y otro en el área de *back end*. Como expertos en desarrollo Android los desarrolladores se han encargado del área de despliegues e infraestructura, no haciendo necesaria la incorporación de un ingeniero de sistemas, y también de la arquitectura Android, evitando incurrir en el coste de un arquitecto experto en Android.

  - **Tester:** el encargado de realizar las pruebas a medida que el software se va desarrollando. En un marco de trabajo en el que las pruebas se automatizan, el Tester estaría también encargado de generar y ejecutar los scripts de pruebas.

- **Pruebas:** se llevarán a cabo por los *testers* del equipo a medida que el desarrollo avanza basado en los criterios de aceptación descritos en las historias de usuario.

- **Criterio de aceptación:** una entrega iterativa con funcionalidad se considerará "Aceptada" cuando los *testers* hayan ejecutado de manera satisfactoria todos los criterios de aceptación (automática y manualmente) y el Product Owner esté satisfecho con el resultado. El Producto se considerará finalizado cuándo todas las entregas parciales hayan sido aceptadas según el criterio anterior y las pruebas *end-to-end* sean satisfactorias, así mismo se requerirá una aceptación por parte del cliente final, entregada por el Product Owner.

2.2 Planificación

Se estima que el tiempo del que se dispondrá para desarrollar el proyecto será de una media de 5 días semanales a razón de 6 horas diarias. Con fecha de inicio prevista el 15 de Mayo y fecha de finalización el 31 de Agosto.

Al haber empleado metodología ágil para el desarrollo del PFC, las fases identificadas no se han ejecutado de una manera lineal, sino que ciertas tareas de distintas fases han sido ejecutadas en paralelo, así como cíclicamente, es decir, se han revisado o completado a medida que el desarrollo ha avanzado, como se refleja en La Figura 3.

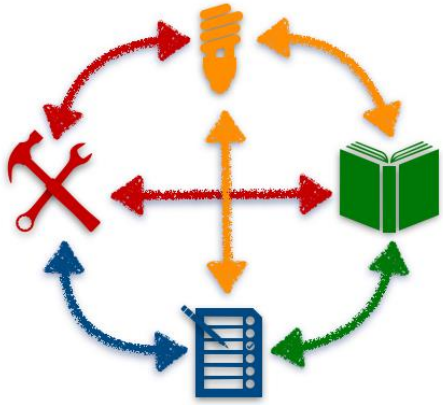


Figura 3. Ciclo del proyecto, el icono de la bombilla representa el plan de proyecto, el libro la formación, la lista de tareas la definición y las herramientas la implementación. Las líneas representan cómo todas las fases han estado relacionadas entre sí.

Plan de proyecto: en esta fase se identificó la necesidad (una aplicación móvil para conocer el estado de los autobuses habituales), se definió el modelo de negocio, así como la estrategia que se seguiría para llevar a cabo el proyecto (desarrollo de una aplicación Android nativa usando la arquitectura de componentes). Todos los aspectos de esta fase están detallados en este capítulo *"II. Proyecto"*. Esta fase se revisó y amplio a lo largo del proyecto, ya que, por ejemplo, durante la implementación se decidió cambiar la arquitectura empleada para el desarrollo en Android, este hecho se explica en el punto "2.5 Arquitectura".

Formación: una vez se decidió llevar a cabo el desarrollo de una aplicación Android nativo, se procedió con una fase de aprendizaje inicial, previa al desarrollo donde se siguieron varios cursos de la plataforma de formación online *Udacity* (https://classroom.udacity.com), ver Capítulo VI "Bibliografía y Recursos" [FOUD]. De la misma forma y como soporte al desarrollo se consultaron foros oficiales y artículos. Ver referencias en Capitulo IV Bibliografía, sección 2 Online, que facilitaron la resolución de problemas.

Definición: en la definición inicial se conformó el Backlog con todas las historias de usuario que se identificaron en el plan de proyecto, y fue siendo modificado a medida que se progresó en las fases de implementación y formación. El Backlog final aparece en el capítulo **"III. Producto"**.

Implementación: tras el plan de proyecto inicial, la formación adquirida para sentar las bases del desarrollo y la definición de las primeras historias de usuario, se pasó a dividir el desarrollo en sprints. Cada sprint albergó un número de historias de usuario, las cuales a su vez generaron una serie de tareas que finalmente compondrían cada uno de los *"Sprint backlog"*, se puede encontrar el detalle de cada sprint en el Capítulo **"IV. Implementación"**. La implementación ha seguido el ciclo reflejado en la Figura 4.

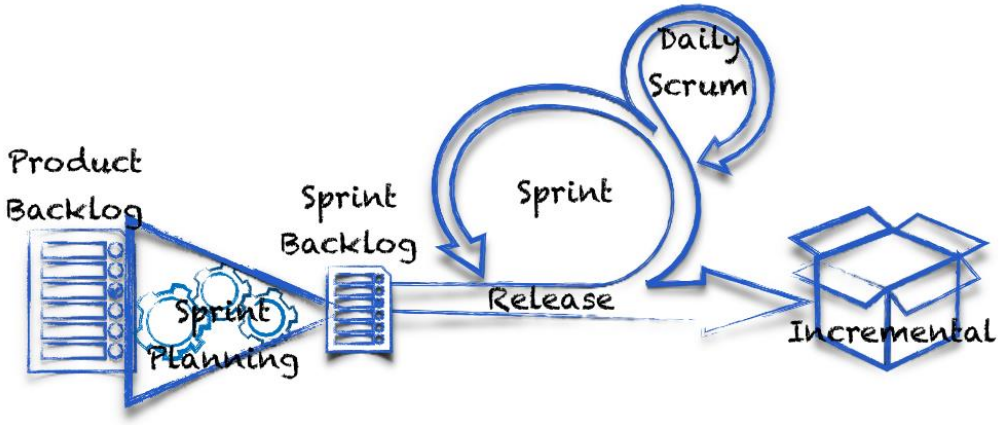


Figura 4. Ciclo de vida Scrum. Definido por una etapa inicial en la que se conforma el backlog del producto que recoge todas las historias de usuario que conforman el producto inicial, el sprint planning en el que se define el Sprint backlog, con las historias de usuario y tareas que se llevarán a cabo en la iteración (sprint), el "Daily Scrum" en el que el equipo se reúne y expone el estado de las tareas y el incremental que se genera al finalizar la iteración.

La Tabla 6 refleja el tiempo total, en horas, empleado en cada una de las fases en contraposición al tiempo esperado.

Fase	Estimado	Real
Plan de proyecto	80	46
Formación	60	109
Definición	20	21
Implementación	240	292
TOTAL	**400**	**468**

Tabla 6. Estimación/Real en horas por fase.

La Tabla 7 muestra un desglose de las horas dedicadas por mes a cada una de las fases.

Fase	Mayo	Junio	Julio	Agosto
Plan de proyecto	22	4	10	10
Formación	45	14	45	5
Definición	5	10	1	5
Implementación	0	92	70	130
TOTAL	72	120	126	150

Tabla 7. Desglose de tiempos.

En el ANEXO IV Diagrama Gantt, se puede encontrar un diagrama de tiempos que junto con las tablas anteriores refleja los tiempos empleados en el proyecto y las tareas alcanzadas.

2.3 Organización

En esta sección se detalla la estructura del equipo, así como las responsabilidades de los miembros del equipo. Cómo se indicó en el punto "1.9 Estructura de costes", el autor ha desarrollado distintas tareas durante la ejecución del PFC, correspondientes a los siguientes roles:

- Como **Product Owner**, se ha identificado la necesidad y se ha dado al equipo el punto de vista de usuario.
- Como **Scrum Master**, se han definido las guías a seguir desde un punto de vista metodológico.
- Como **Business Analyst**, se han creado las historias de usuario con la información provista por el Product Owner.
- Como **Equipo de desarrollo**, se ha desarrollado la funcionalidad descrita en las historias de usuario.
- Como **Tester**, se ha probado la funcionalidad completada por los desarrolladores.

2.4 Entorno tecnológico

2.4.1 Herramientas

- Ordenador portátil: Entroware Apollo: Core i7 7500U, 32GB DDR4 2133MHz, 256GB PCIe SSD, -Ubuntu 17.04
- Monitor: LG 28 pulgadas.
- Móvil Nexus 5 con sistema operativo Android 6.0.1
- Móvil XiaoMi Redmi 3 Pro 32GB ROM 4G Smartphone con sistema operativo Android 5.1

2.4.2 Entorno de desarrollo

- Android Studio: como IDE para el desarrollo de la aplicación.
- Emulador Android: para la ejecución de pruebas.

- API Transport London: API unificada de transportes de Londres que provee información sobre el estado de los distintos medios de transporte públicos de la ciudad de Londres.

- GIT: como repositorio de código.

- Trello: como herramienta para la gestión de backlog.

- Boxy SVG: software para la creación de imágenes vectoriales empleado para la creación y edición de imágenes.

- Pages: para la creación de wireframes y diagramas.

- Sonar QUBE: para la generación de métricas y gráficas de rendimiento.

2.5 Arquitectura de componentes

Este punto ha sido uno de los hitos más importante dentro de la realización del proyecto. El primer paso fue realizar un pequeño estudio de la arquitectura básica de componentes de una aplicación Android, intentando entender todos los componentes que hacen que funcione una aplicación en un sistema Android. La Figura 5 es una representación gráfica de esta arquitectura, en la que se detallan los distintos componentes de cada una de las capas.

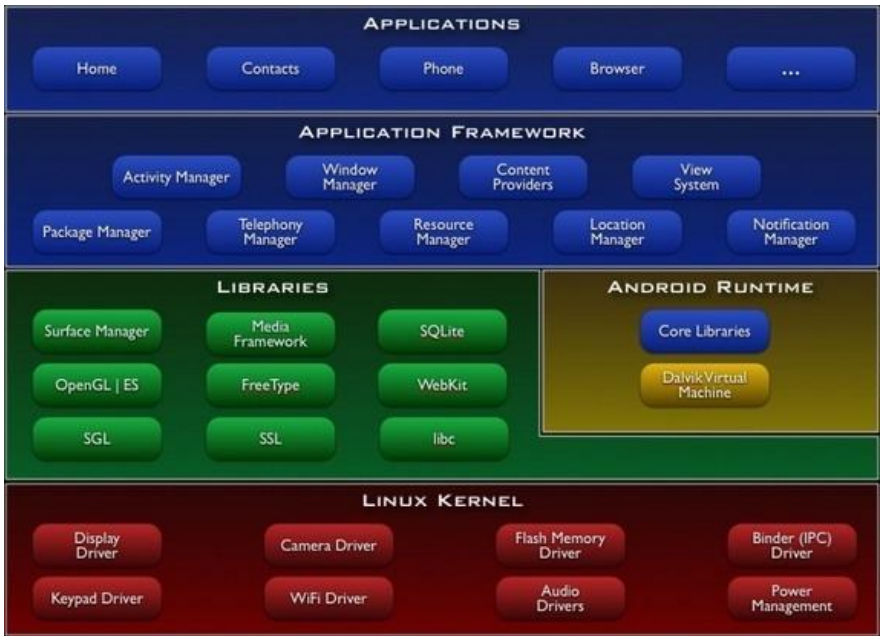


Figura 5. Arquitectura básica de los componentes de Android. Fuente de la imagen: https://www.linkedin.com/pulse/android-architecture-basic-components-zain-ul-abidin.

- ***Linux Kernel***: Es el corazón de la arquitectura de Android. Es el responsable de los drivers, del manejo de la batería, de la memoria, del dispositivo y del acceso a los recursos.

- ***Librerías Nativas***: Encima del kernel, están las librerías nativas como WebKit, OpenGL, FreeType, SQLite, Media, C runtime library (libc) etc. La librería WebKit es responsable del soporte al navegador, SQLite es para la BBDD, FreTyoe para el soporte a la fuente, Media para formatos de audio y video.

- ***Runtime***: formado por librerías nucleares (core) y la Dalvik Virtual Machine (DVM). Responsables de ejecutar la aplicación Android. DVM es como la JVM, pero optimizada para móviles. Consume menos memoria y da un mejor rendimiento.

- ***Application Framework***: Encima de las librerías Nativas y el runtime de Android, está el Entorno de Aplicación. Éste incluye todas las APIs como la Interfaz de Usuario (UI), el teléfono, los recursos, la localización, el proveedor de contenidos y los manejadores de paquetes. Provee muchas clases e interfaces para el desarrollo de aplicaciones Android.

- ***Aplicaciones:*** En la parte superior se encuentran las aplicaciones. Todas las aplicaciones como el *home*, los contactos, la configuración, los juegos, los navegadores están usando el framework de Android, que usa la runtime de Android y las librerías, que a su vez usan el kernel linux.

En el transcurso del segundo Sprint, en el que ya se habían empezado a cubrir ciertas historias de usuario, y aprovechando que me encontraba en continuo aprendizaje y que la metodología seguida lo permitía, se tomó la decisión de emplear la nueva Arquitectura de Componentes, que consiste en una estructura como se muestra en la Figura 6.

La idea bajo esta nueva Arquitectura de Componentes, es el código "*Clean*". La programación orientada a objetos debe cumplir cinco principios básicos que se recogen bajo el acrónimo SOLID:

- ***Single Responsibility***: cada unidad de código debe tener una única responsabilidad.

- ***Open/Closed:*** podemos extender las funcionalidades de un objeto, no modificarlas.

- ***Liskov:*** un objeto puede ser siempre sustituido por otro objeto subtipo

- ***Interface segregation:*** mejor varios interfaces específicos que uno general.

- ***Dependency inversion:*** nn objeto debe depender de abstracciones.

Las arquitecturas "*Clean*" intentan cumplir con estos principios, separando claramente las responsabilidades de cada capa: View, Model, Data Access… Además, mediante la inyección de dependencias conseguimos aislar las clases de las implementaciones concretas de sus colaboradores. Por ejemplo, Dagger de Google es un ejemplo de entorno de inyección estática de dependencias para Java en Android: (https://google.github.io/dagger/).

Con esto se consigue un código fácil de testear, ya que se puede probar cada capa o clase por separado. También es fácil de modificar puesto que los cambios en la fuente de datos, no afectan al modelo de negocio, o un rediseño completo de la UI no supondrá modificaciones en las capas exteriores.

Esta arquitectura surge como respuesta a que, en su diseño original, Android no pretendía ser "*Clean*" ni cumplir SOLID. Estaba más orientaba a facilitar el desarrollo de pequeñas aplicaciones para teléfonos de escasa potencia por desarrolladores no demasiado experimentados. Sin embargo, esta nueva arquitectura permite una separación de responsabilidades entre las diversas capas de la aplicación para cumplir con SOLID, como pueden ser: MVC, MVP, MVVM o Hexagonal. Aun cuando cada una de estas implementaciones estuviera bien resuelta, su diversidad ha creado otro problema: cada desarrollador creaba su versión. Y así surgió la necesidad de unificar la manera de hacer las cosas para aportar ventajas al ecosistema Android y a los desarrolladores. Para solucionar estos problemas, Google propone una arquitectura "Clean" basada en cuatro aspectos:

- ***Gestión del ciclo de vida***: mediante la implementación de un Interface por parte de Activities y Fragments (LifecycleOwner) estas clases pueden informar de su ciclo de vida (Lifecycle) a otras clases, que pueden actuar en consecuencia mediante métodos anotados. Actualmente estas clases con soporte de Lifecycle se proporcionan como ejemplos (LifecycleActivity y LifecycleFragment) en la librería de arquitectura, pero en la release final la librería de compatibilidad soportará directamente Lifecycle.

- ***ViewModel:*** las implementaciones de esta interface contienen los datos necesarios para mostrar la vista. Estos ViewModel se instancian a través de la clase ViewModelProvider, que se asegura de que sobreviven

a cambios de configuración de la Activity. La responsabilidad de conseguir y almacenar los datos recae en el ViewModel y no en la Activity.

- *LiveData*: es una clase de almacenamiento de datos que permite que estos sean observados. Se diferencia de un Observable en que es consciente del ciclo de vida de la Activity evitando, por ejemplo, mandarle datos cuando no está en primer plano.

- *Room:* las aplicaciones móviles deben soportar redes de baja calidad e incluso permitir su uso mientras estamos desconectados de la red. Para ello se utiliza el almacenamiento local, pero desafortunadamente 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.

Por lo tanto, una arquitectura que sigue el paradigma Model-View-ViewModel puede definirse con el diagrama de la Figura 6.

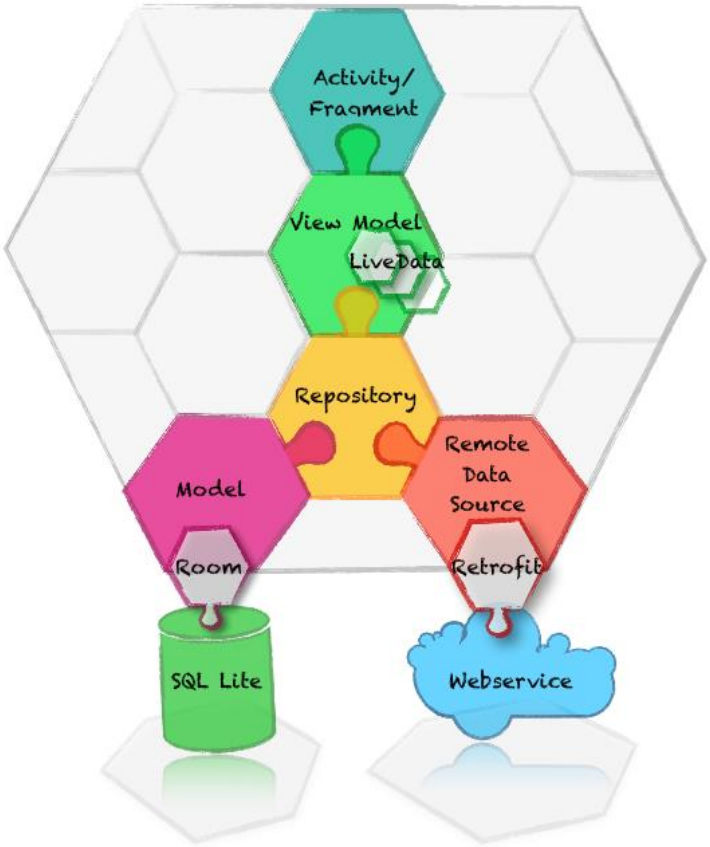


Figura 6. Arquitectura de componentes.

Para más detalle de cada una de las partes de la arquitectura y de fragmentos de código que muestran su implementación ver Anexo IV.

2.6 Gestión de riesgos

En esta sección se detallarán los posibles riesgos identificados previos al inicio del desarrollo del proyecto, el impacto que pueden causar, así como las medidas de contingencia que se pueden aplicar para evitarlos o paliarlos.

Se considera "Riesgo" cualquier acontecimiento que pueda afectar de forma negativa al proyecto. Se clasifican en: de proyecto y técnicos.

2.6.1 Riesgos de proyecto

Se refieren a aquellos que ponen en peligro el plan, relacionados con presupuesto, planificación, personal, recursos o requisitos. En el caso de este PFC se identificaron dos riesgos principales relacionados con el proyecto, la Tabla 8 refleja la descripción del riesgo, cómo se gestionó y las medidas de contingencia.

Riesgo	Descripción	Medidas de contingencia
Planificación	Al carecer de experiencia en desarrollo Android, se puede haber incurrido en una planificación demasiado optimista. Del mismo modo se han podido pasar por alto tareas necesarias o haber cometido incoherencias en los requisitos que obliguen a cambiarlos.	• Formación en desarrollo Android previa a la estimación, de modo que las estimaciones fueran más realistas. • Uso de Scrum, que permite redefinir requisitos y ajustar el alcance del proyecto a los sprints delimitados por el tiempo del proyecto, desarrollando primero las historias de usuario de mayor prioridad, y generando un producto con valor en cada ciclo.
Tiempo	El tiempo con el que se contaba para la finalización del proyecto estaba delimitada por la fecha límite para la presentación, dado que el autor trabaja a tiempo completo y da servicio guardia, podrían surgir complicaciones que limitaran el tiempo disponible.	Se trató de ajustar el alcance del proyecto al tiempo disponible, dejando un margen de tiempo libre para posibles imprevistos.

Tabla 8. Riesgos de proyecto.

2.6.2 Riesgos técnicos

Se refieren a aquellos que ponen en peligro la calidad y el desarrollo del proyecto. En el caso de este PFC se anticiparon los riesgos técnicos descritos en la Tabla 9.

Riesgo	Descripción	Medidas de contingencia
Tecnología desconocida y Diseño	• Dado que el PFC suponía el primer desarrollo de una aplicación Android para el autor, se identificó como riesgo que la curva de aprendizaje podría ser mayor a la esperada. • El hecho de que no existiera una arquitectura clara e inequívoca de desarrollo en Android, podría causar problemas a la hora de afrontar ciertas dificultades técnicas.	• De nuevo se intentó paliar con una formación previa al inicio del proyecto, para saber a qué nos enfrentamos. • Durante el desarrollo se decidió emplear la nueva arquitectura de componentes. Aunque supuso dedicar más tiempo en aprendizaje y cambiar el rumbo del proyecto, una vez comprendido, supuso un avance más sólido en el desarrollo y posibilitó la finalización del proyecto a tiempo.
Implementación	El tiempo disponible para implementación podía ser comprometido, por el tiempo requerido para la adquisición de los conocimientos técnicos requeridos.	Se trató de ajustar el alcance del proyecto al tiempo disponible, dejando un margen de tiempo libre para posibles imprevistos.
Dependencias externas	Se depende de la disponibilidad de la API TFL. A pesar de ser un servicio contrastado, y con bastante estabilidad, siempre se está sujeto a cambios y problemas de servicio.	• Se cuenta con margen de tiempo por si un cambio en la implementación de la API pudiera suponer cambio en las llamadas. • Se ha creado una Key única para la aplicación buStop que ofrece servicio dedicado.

Tabla 9. Riesgos técnicos.

III. PRODUCTO

1 Historias de usuario

Los requisitos se presentan en la forma de historias de usuario, estas historias de usuario definen de manera simple el comportamiento de funcionalidades específicas dentro del producto, las sumas de todas las historias de usuario completadas en el desarrollo darán forma al producto final. Se seguirá la plantilla detallada en la Tabla 10.

Campo	Descripción
Código	Código alfanumérico que servirá de referencia.
Descripción	La historia de usuario con la sintaxis: **Cómo** X **Quiero** Y **Para** Z. Siendo X el rol de la persona que hará uso de la funcionalidad, Y la funcionalidad deseada y Z el fin que se espera conseguir con la funcionalidad.
Prioridad	Prioridad de la historia, vendrá dada en un valor numérico. Mayor número, mayor prioridad. Esta prioridad suele darla el Product Owner.
Estimación	Estimación del coste de la historia de usuario en Story points. Una de las prácticas habituales de estimación en Scrum es la conocida como "Estimación de póquer", en la que se usan una serie de cartas con distinta numeración que suponen el número de Story points en los que vamos a estimar una historia de usuario. Una de sus variantes es usar la serie de Fibonacci para numerar las cartas (0, ½, 1, 2, 3, 5, 8, 13, 21), basado en el hecho de que al aumentar el tamaño de las tareas aumenta también el margen de error. Por ejemplo, una historia que se considere de 10, obligará a reconsiderar una postura más optimista y estimarla en 8 o más conservadora y llevarla a 13.
Criterios de aceptación	Las condiciones que un producto de software debe satisfacer para ser aceptado por un usuario, cliente o stakeholder. No sólo servirá para determinar los scripts de pruebas sino para ayudar a desarrollar la funcionalidad, puede haber de 1 a N criterios de aceptación dentro de la historia de usuario, que definen los distintos escenarios posibles (éxito, fallo, …). Es una buena práctica escribir los criterios de aceptación con la sintaxis: **Dado** que X **Cuando** Y **Entonces** Z. Siendo X los antecedentes o el trasfondo del escenario, Y las acciones ejecutadas en el escenario y Z el resultado esperado. Esta sintaxis pertenece al lenguaje "Gherkin" empleado en los entornos de pruebas de aceptación, como *Selenium*, para definir los scripts de prueba.

Tabla 10. Estructura de la historia de usuario.

2 Product Backlog

La Tabla 11 representa el *Product backlog* que detalla las historias de usuario identificadas.

Código	Descripción	Prioridad	Estimación
US01	**Como** usuario de buStop **Quiero** poder ver las paradas que se encuentran cerca de mi ubicación **Y** las líneas de autobús de estas paradas **Para** poder ver las posibilidades que tengo cerca de mi	Baja	15
US02	**Como** usuario de busStop **Quiero** que, al acceder a la app, si existen favoritos, la aplicación me muestre la información de mis favoritos ordenada por distancia a mi ubicación actual **Para** disponer de la información de la parada más cercana a primera vista, pero poder consultar otras	Muy Alta	4
US03	**Como** usuario de buStop **Quiero** que la aplicación detecte mi ubicación **Y** la muestre en un mapa **Para** poder usarla en las distintas funcionalidades de la aplicación	Alta	19

Código	Descripción	Prioridad	Estimación
US04	**Como** usuario de buStop **Quiero** poder usar la aplicación sin necesidad de registros ni logins **Para** no perder el tiempo	Alta	1
US05	**Como** usuario de buStop **Quiero** poder consultar mis autobuses/paradas favoritos **Para** ver la información de estos	Alta	2
US06	**Como** usuario de buStop **Quiero** tener un menú flotante **Para** poder acceder a mis favoritos o "Near me" desde cualquier pantalla	Media	3
US07	**Como** usuario de buStop **Quiero** poder gestionar mis favoritos, eliminando y añadiendo nuevos favoritos **Para** poder almacenar sólo la información que necesito	Alta	10

Tabla 11. Product Backlog.

3 User Journey

El "*User journey*" una herramienta que viene del *"Design thinking*" y es clave para diseñadores, a la hora de desarrollar la experiencia de usuario de un nuevo producto o servicio. Esta herramienta permite plasmar en un mapa, cada una de las etapas, interacciones, canales y elementos por los que pasa nuestro cliente desde un punto a otro de nuestro servicio. La Figura 7 es una representación gráfica del *user journey* y la Tabla 12 ofrece una explicación detallada de cada uno de los pasos representados en el *user journey*.

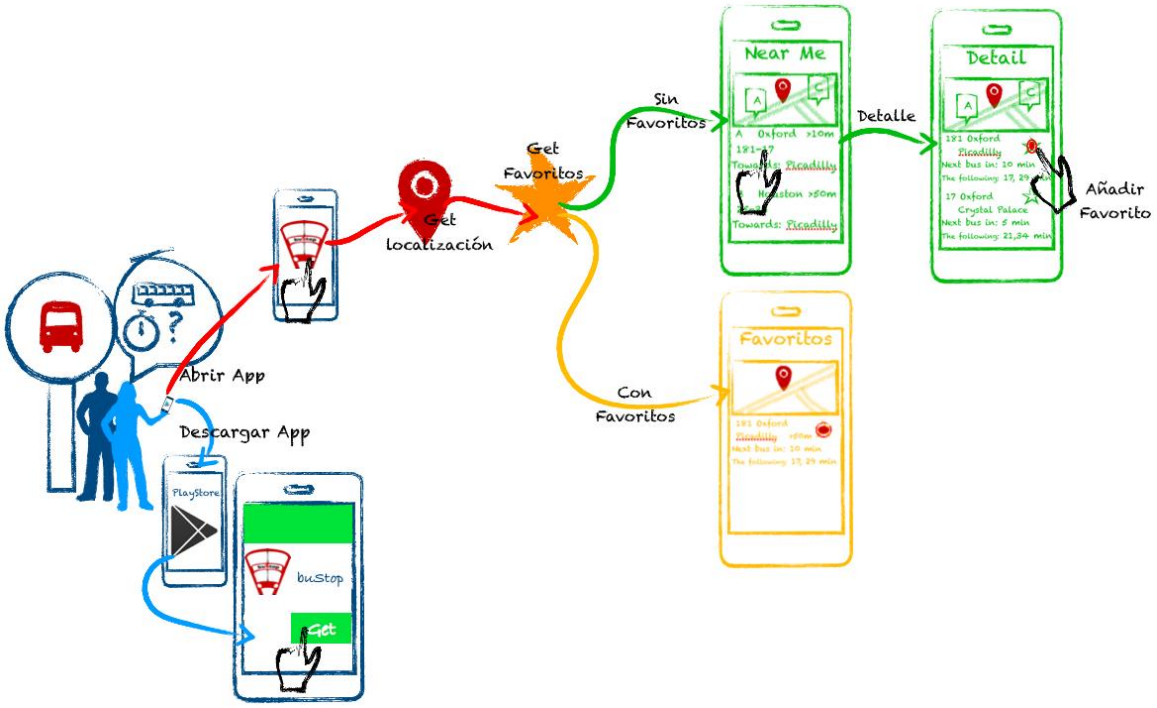


Figura 7. buStop User Journey.

Journey	Descripción
Descargar	Es el proceso que un usuario tiene que seguir para poder instalar la aplicación en su terminal. No es una funcionalidad específica de la aplicación
Near me / Detalle	A esta funcionalidad se accede de inicio en caso de que el usuario no haya configurado ningún favorito, o si se accede desde el menú. Esta vista presenta un listado con todas las paradas en un radio de 200 metros a la localización del usuario, añadiendo cierta información. El usuario podrá pulsar sobre cualquiera de las paradas en el listado para ir a la vista de detalle que presentará información detallada sobre las líneas que paran en ella y el tiempo que falta para que llegue el siguiente autobús de esa línea, así como la opción de quitar o añadir a favoritos.
Favoritos	Este journey se inicia automáticamente si al acceder a la aplicación el usuario dispone de favoritos o accediendo a través de la opción de menú. En ambos casos, se mostrará la vista detalle de favoritos, en la que se listarán los favoritos con la información de tiempos de forma similar a la vista detalle anteriormente descrita.

Tabla 12. Descripción User Journey.

4 Pre-Requisitos

Es requisito indispensable un dispositivo móvil, con sistema operativo Android superior a la versión ***Jelly Bean*** 4.2

Para el uso de esta aplicación es necesario que el usuario tenga instalado en su dispositivo móvil, Google Play Service, así como tener activada la localización.

5 Experiencia de usuario

En esta sección se presentan los diseños realizados para la aplicación buStop.

5.1 Iconografía

Se han diseñado las siguientes imágenes vectoriales para la aplicación. La Figura 8 se ha importado como icono de la aplicación y la Figura 9 es la imagen empleada como título de la aplicación.

Figura 8. Icono de título.

Figura 9. Imagen de título.

5.2 Wireframes

Los siguientes bocetos (*Wireframes*) son una representación del diseño empleado en las distintas pantallas de la aplicación. Las FigurasFigura 10, Figura 11, Figura 12, y Figura 13 muestran los bocetos de las distintas pantallas y componentes.

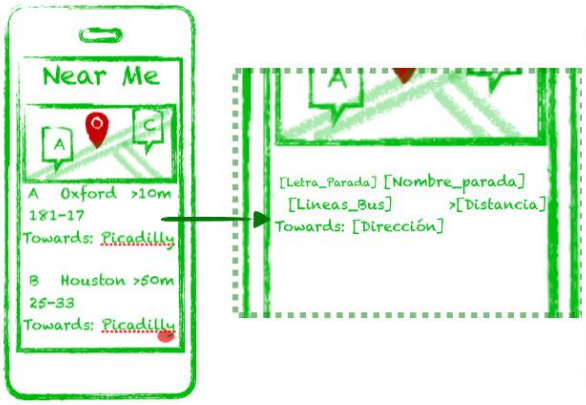
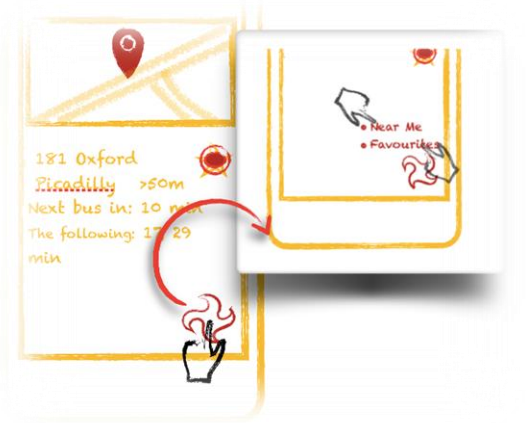


Figura 10. Diseño vista "Near me" y detalle de la información mostrada.

Figura 11. Vista detalle e información específica mostrada por línea de autobús.



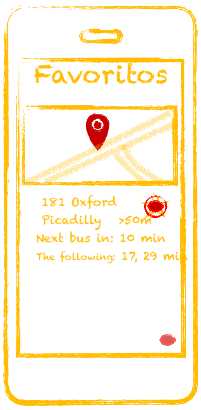


Figura 12.Vista de Favoritos, el detalle de la información es similar al de la vista de Detalle.

Figura 13. Vista del Menú flotante presente en todas las vistas de la aplicación.

IV. IMPLEMENTACIÓN

1 Introducción

En este capítulo se detallan las iteraciones o *sprints* que realizados durante el proyecto. Las características del tipo de iteración empleada se pueden encontrar en el Capítulo II, en la sección 2.1 "Metodología". Recordemos que se definieron ciclos de tres semanas y el máximo de puntos de historia por iteración es de 20 *Story points*. Cada sección corresponde con un *Sprint* y se divide en:

- ***Planificación,*** donde se explica las tareas que se llevaron a cabo en la iteración reflejadas en el *Sprint Backlog,* en el que se indicarán para cada una de las tareas o historias de usuario el código, una descripción y como estimación, la cantidad de puntos de historia que empleará dentro del Sprint.

- ***Riesgos,*** donde se reflejan los riesgos que se identificaron antes de la ejecución del sprint.

- ***Revisión y retrospectiva***, donde se analiza el resultado del Sprint, qué se ha conseguido y qué no y los problemas encontrados. A partir de la revisión se sacarán conclusiones, lecciones aprendidas e ideas que ayudarán a redefinir el proceso.

1.1 Sprint 0 - Implantación del Entorno de Desarrollo

1.1.1 Planificación

Se identificó la necesidad de realizar un Sprint inicial en el que se sentarían las bases del proyecto, se preparó el entorno de desarrollo y se hicieron las pruebas de concepto necesarias. No se desarrollarían por tanto ninguna de las historias de usuario del *Product Backlog*. Dada la peculiaridad de este *sprint* inicial sólo se empleó una semana en vez de las tres estipuladas, y en lugar de destinar los 20 puntos de historia para cada iteración se destinaron sólo 7.

El *Sprint Backlog* correspondiente al *sprint* 0 quedó conformado como explica la siguiente tabla.

Código	Descripción	Estimación
T00	Formación	1/2
T01	Prueba de concepto con Android Nativo (instalación IDE +SDK)	3
T02	Prueba de concepto con Apache Cordova (instalación node.js, ApacheCordova)	3
T03	Toma de decisión	1/2

Tabla 13. Sprint 0 Backlog.

1.1.2 Riesgos

Al ser un Sprint 0 en el que se van a sentar las bases y preparar los entornos de desarrollo no se identificaron riesgos específicos, se emplearía el tiempo estipulado y no más, en adquirir y ampliar los conocimientos necesarios para empezar con el desarrollo, así como a la preparación del entorno.

1.1.3 Revisión y retrospectiva

El Sprint comenzó el 15/05/17 y finalizó el 21/05/17, se completaron las tareas planificadas en el tiempo establecido, aunque se comprobó que el rendimiento del entorno de desarrollo no era el adecuado en el equipo informático del que se disponía y hubo que adquirir otro de mayores prestaciones por lo que las tareas T01 y T02 consumieron mayor tiempo del planificado ya que hubo que realizarlas, de nuevo, en el nuevo equipo.

Tras analizar el resultado de las pruebas de concepto realizadas con Apache Cordova y con Android Nativo se tomó la decisión de implementar el proyecto con Android Nativo, por el reto que suponía para la autora. Dada la complejidad del marco de trabajo de Android se decide reservar un tiempo en cada *sprint* para formación y adquisición de conocimientos.

1.2 Sprint 1. Ampliación de conocimientos y API (Volley)

1.2.1 Planificación

Del 22/05/2017 al 11/06/2017. Este *sprint* se destina a completar los conocimientos adquiridos en el *sprint 0*, así como al estudio del API de transporte de Londres, dado que se ha decidido que se va a realizar el proyecto con Android Nativo, se dedica gran cantidad de este sprint a formación. La primera parte con la que se comienza a desarrollar es con la recuperación de datos desde el API y para ello se usa Volley.

El *Sprint Backlog* correspondiente al *sprint* 1 quedó conformado por las tareas recogidas en la siguiente tabla.

Código	Descripción	Estimación
T00	Formación	5
T05	Recuperación de datos del API de transporte a través de Volley	13
T06	Estudio del API de transporte	2
T07	Gestión de licencias y *key* del API de transporte	1

Tabla 14. Sprint 1 Backlog.

1.2.2 Riesgos

De nuevo al ser un *sprint* destinado a la consolidación de conocimientos y prueba de concepto no se identifican riesgos ya que sólo se empleará el tiempo estipulado por el *sprint*.

1.2.3 Revisión y retrospectiva

Se completaron de modo satisfactorio las tareas planificadas en el tiempo establecido, de las tareas se desprendieron las siguientes conclusiones y resultados:

- Se profundiza en el conocimiento de la plataforma y el desarrollo en nativo.
- Se analiza el API de transportes de Londres y se identifican los métodos necesarios para recuperar la información requerida por la aplicación.
- Se genera una Key única para que nuestra aplicación tenga un servicio dedicado de la API.
- La recuperación de datos del API de transporte se realizará través de Volley, tecnología recomendada por google para interactuar con APIs, ver referencia [W3VOLL]

- Tras la prueba de concepto inicial, la primera toma de contacto con Android a través de Volley y con una aplicación muy simple, hace que la autora se dé cuenta de la complejidad y de la gran cantidad de sobre-información y no siempre actualizada que hay sobre Android.

1.3 Sprint 2. Geolocalización

1.3.1 Planificación

Del 12/06/2017 al 02/07/2017. En este *sprint,* se considera que con los conocimientos adquiridos se puede comenzar a desarrollar más funcionalidad especifica de la aplicación y a hacer que interactúen entre ellas las distintas funcionalidades. En concreto se decide completar la US03, que supondría obtener la ubicación del usuario y mostrarla en un mapa. De esta historia de usuario, se desprendieron tres tareas técnicas, T10 Geolocalización del usuario, T11 Geolocalización con fragmentos y T12 *ModelView* para interactuar con el fragmento de localización, que una vez completadas darían por completada la US03.

El *Sprint Backlog* correspondiente al *sprint* 2 quedó conformado por las tareas recogidas en la siguiente tabla.

Código	Descripción	Estimación
T00	Formación (Arquitectura de componentes) y documentación	1
T09	Solicitar usuario de Google Play	1/2
US03	**Como** usuario de buStop **Quiero** que la aplicación detecte mi ubicación **Y** la muestre en un mapa **Para** poder usarla en las distintas funcionalidades de la aplicación	19
	US03T10 Geolocalización del usuario	(1)
	US03T11 Geolocalización con fragmentos	(13)
	US03T12 *ModelView* para interactuar con fragmento de localización	(5)

Tabla 15. Sprint 2 Backlog.

1.3.2 Riesgos

Basado en la experiencia ganada con la interactuación con el API realizada en la anterior iteración, se anticipa la dificultad que supondrá la Geolocalización con fragmentos, dada la interacción y gestión de eventos en Android, por lo que se le da una estimación "pesimista" a la tarea en cuestión y se decide no incluir ninguna otra historia de usuario en el *sprint*.

1.3.3 Revisión y retrospectiva

Cómo se anticipó, surgieron problemas con la sincronización y el manejo de eventos, durante la búsqueda de información se encontró la información sobre la Arquitectura de Componentes.

- Se descartó seguir desarrollando sin seguir la arquitectura de Componentes liberada por primera por Google este año (2017) en el Google i/O 2017, ver referencia [W3GOOG] y que se encuentra en versión *alpha*.

- Se identifica la necesidad de ampliar conocimientos en la Arquitectura de Componentes.

- Se decide rehacer la Geolocalización con fragmentos y usando la arquitectura de componentes *LifecycleActivity y LifecycleFragment* y realizo el primer *ModelView* para interactuar con el fragmento de localización.

1.4 Sprint 3. API (Retrofit)

1.4.1 Planificación

Del 03/07/2017 al 23/07/2017. Una de las implicaciones de adoptar la Arquitectura de Componentes era rehacer la interacción con el API de transporte a través de Retrofit. Esto implica usar Observers, creando una clase que se encargue de la comunicación con el Repositorio Remoto y que interactúe con el *ModelView*. Se empieza a trabajar con la interacción entre los fragmentos y con el pintado de las listas mediante *RecyclerViews*.

La otra implicación consistía en volver a invertir tiempo en formación, para llegar a comprender todos los conceptos nuevos, que la nueva Arquitectura de Componentes introduce.

El *Sprint Backlog* correspondiente al *sprint* 3 quedó conformado por las tareas recogidas en la siguiente tabla.

Código	Descripción	Estimación
T00	Formación (Arquitectura de Componentes) y documentación	5
US01	**Como** usuario de buStop **Quiero** poder ver las paradas que se encuentran cerca de mi ubicación **Y** las líneas de autobús de estas paradas **Para** poder ver las posibilidades que tengo cerca de mi	15
	US01T14- Recuperación de las paradas cerca de mi ubicación con *Retrofit* del API de transporte	(5)
	US01T15- Recuperación de las líneas de nuevo del API con *retrofit*	(5)
	US01T16- Pintado en el mapa de los resultados devueltos por el API	(2)
	US01T17- Pintar las paradas cercanas y las líneas con *RecyclerView* de modo que la aplicación tenga buenos tiempos de respuesta	(3)

Tabla 16. Sprint 3 Backlog.

1.4.2 Riesgos

Dado que este proyecto se realiza con el API de transportes de Londres, pero se va a presentar en Zaragoza, al implementar "paradas cercanas a mi ubicación actual", se vio que en esa ubicación no aparecerían tales paradas.

1.4.3 Revisión y retrospectiva

Evaluado el riesgo, se decidió implementar un método que cuando la localización está desactivada, calcula una localización aleatoria dentro de Londres. Por el contrario si la ubicación esta activada, geolocaliza al usuario.

1.5 Sprint 4. SQLite (ROOM)

1.5.1 Planificación

Del 24/07/2017 al 13/08/2017. Para poder gestionar los favoritos, se hace necesario el almacenamiento de cierta información, se consideraron las distintas opciones, desde ficheros hasta bases de datos (BBDDs). Pensando en la escalabilidad de la aplicación y considerando el interés de la autora por trabajar con los diferentes componentes disponibles, se decidió usar ROOM y almacenar la información en SQLite.

El *Sprint Backlog* correspondiente al *sprint* 4 quedó conformado por las tareas recogidas en la siguiente tabla.

Código	Descripción	Estimación
T00	Formación (Arquitectura de Componentes)	8
US07	**Como** usuario de buStop **Quiero** poder gestionar mis favoritos, eliminando y añadiendo nuevos favoritos **Para** poder almacenar sólo la información que necesito	10
	US07T19- Añadir favoritos	(5)
	US07T20- Borrar favoritos	(5)
US05	**Como** usuario de buStop **Quiero** poder consultar mis autobuses/paradas favoritos **Para** ver la información de éstos	2
	US05T22- Hacer un "Select" a la BBDD	(2)

Tabla 17. Sprint 4 Backlog.

1.5.2 Riesgos

El uso de *LiveData* para informar a los diferentes fragmentos de los cambios que ocurren tanto en la BBDD como en el API, hace mucho más fácil la gestión de los eventos y la visualización de la información en tiempo real, pero el entender este tipo de objetos ha supuesto un reto y un retraso a la planificación.

1.5.3 Revisión y retrospectiva

El resultado final de la gestión de los eventos y de la información externa ya sea en repositorio local o remoto se considera mucho más eficiente y menos propensa a errores. La gestión de los objetos *LiveData* y de los métodos asíncronos supone un gran reto en el desarrollo de las aplicaciones Android

El uso de objetos *LiveData* para el paso de información entre las distintas partes de la arquitectura supone un gran avance y un gran cambio en el desarrollo de aplicaciones, haciendo que estos objetos (*Observers*) estén esperando cualquier modificación tanto de BBDD como de API y en cuanto esto suceda, se encarguen de avisar a las capas de IU.

1.6 Sprint 5. Favoritos y refinamiento.

1.6.1 Planificación

Del 14/08/2017 al 02/09/2017. En este último sprint nos centramos en el menú que muestra las paradas cercanas o los favoritos. También nos centraremos en que un usuario que ya tenga favoritos, sin interactuar con la aplicación, pueda obtener las siguientes llegadas de las líneas que tiene almacenadas en favoritos.

El *Sprint Backlog* correspondiente al *sprint* 5 quedó conformado por las tareas recogidas en la siguiente tabla.

Código	Descripción	Estimación
T00	Formación (Arquitectura de Componentes)	3
US06	**Como** usuario de buStop **Quiero** tener un menú flotante **Para** poder acceder a mis favoritos o "Near me" desde cualquier pantalla	3

Código	Descripción	Estimación
US02	**Como** usuario de busStop **Quiero** que, al acceder a la app, si existen favoritos, la aplicación me muestre la información de mis favoritos con la distancia a mi ubicación actual **Para** disponer de la información de mis favoritos a primera vista, pero poder consultar otras	**4**
	US02T24- Si existen favoritos que el primer fragmento que se muestre sea el de favoritos	(2)
	US02T25- Que cuanto muestre los favoritos, te geolocalice y calcule la distancia que hay entre ti y los favoritos.	(2)
US04	**Como** usuario de buStop **Quiero** poder usar la aplicación sin necesidad de registros ni logins **Para** no perder el tiempo	**1**
T26	Usar *dependency injection* para probarlo tanto en en repositorio remoto como en local	5
T27	Refinamiento del código, comentándolo y generando JavaDoc	3

Tabla 18. Sprint 5 Backlog.

1.6.2 Riesgos

Dado que es el último sprint y no hay mucho tiempo hasta la entrega, se pone en riesgo que alguna de las tareas no se acabe en el tiempo estimado, haciendo que el producto se ponga en riesgo.

1.6.3 Revisión y retrospectiva

Finalmente se ha conseguido terminar las tareas a tiempo, teniendo siempre en mente que todo se puede mejorar, y registrando un grupo de mejoras y componentes a probar para un futuro próximo.

El riesgo más importante que se ha identificado a nivel global es que la Arquitectura de Componentes está en versión *alpha*, esto hace que las funcionalidades usadas sean susceptibles a cambios. Este riesgo se ha asumido por considerar que las pautas que se han seguido y los objetos y el comportamiento son los que se van a usar desde ahora en el desarrollo de Aplicaciones Nativas, haciendo si cabe más interesante la opción de desarrollar en Android Nativo.

2 Código Fuente

Se ha incluido en el ANEXO II Arquitectura de componentes, la explicación y fragmentos del código de la aplicación. El código fuente se puede encontrar en:

https://github.com/anasjn/BuStop_With_ArchComponents.git

Para más información también se encuentra publicado el ***JavaDoc*** en:

https://anasjn.github.io

Se ha incluido un anexo con el análisis de métricas, ANEXO III Métricas, estáticas generadas con la herramienta SonarQube. También en el anexo se encuentra el análisis de métricas dinámicas generadas con Android Studio.

V. CONCLUSIONES

1 Objetivos alcanzados

El objetivo académico de este PFC no ha sido simplemente desarrollar una aplicación que cubra una necesidad, sino aprovechar "el camino" necesario para llegar a ese fin, para aprender y estudiar en detalle las distintas herramientas que se emplean en el mundo laboral para el desarrollo de proyectos de software. En concreto para este proyecto se han empleado técnicas y herramientas que actualmente, en 2017, son novedosas, de uso extendido y aún en auge, como la metodología Scrum, el *"Model Business Canvas"* para el desarrollo del modelo de negocio y la nueva Arquitectura de componentes para desarrollos nativos en Android. No se han elegido estas herramientas por el mero hecho de ser "las últimas" o las más empleadas en la actualidad, sino que se han analizado las alternativas y enfrentado entre sí para decidir cual se adaptaba mejor al tipo de proyecto que se pretendía realizar. Considero que gracias a este PFC no solo he cumplido los objetivos planteados al principio del proyecto, sino que, en el proceso he aprendido otras tecnologías y metodologías que no me planteé al comienzo, sirva de ejemplo el Sonarq, que empleé para obtener las métricas una vez completado el desarrollo, del que no tenía conocimiento y tras haberlo empleado en este PFC espero poderlo integrar en otros proyectos de mi ámbito profesional.

2 Retrospectiva

Tras haber finalizado y haciendo retrospectiva del "viaje" realizado, por supuesto que habría cosas que mejorar, principalmente haber dispuesto de más tiempo, pero en conjunto considero que el resultado ha sido muy satisfactorio, el proyecto ha sido bien dimensionado y se ha podido completar en el tiempo estipulado. También considero como acertada la decisión de haber desarrollado la aplicación en Android nativo en lugar de haber tomado una aproximación más segura como habría sido Apache Cordova, lo cual posiblemente hubiera recortado mucho el tiempo de aprendizaje y de implementación. Si bien es cierto que los desarrollos en nativo son mucho más complejos y más propensos a fallos, debido al número de eventos a procesar y a la gran cantidad de estados por los que pasa una aplicación y a los quebraderos de cabeza que, de forma recurrente, me ha causado a lo largo del proyecto, pienso que el haberme hecho salir de mi zona de confort constantemente, me ha hecho crecer a nivel, no sólo profesional, con todo lo aprendido, sino también en el plano personal.

3 Valoración personal

Después de seis años como administradora de sistemas, y ahora como responsable de equipo, en los que he dejado de lado el desarrollo, este proyecto ha conseguido despertar de nuevo en mí, el interés por el desarrollo y una nueva cualificación.

4 Líneas futuras

4.1 Mejoras

- Corrección de posibles errores no detectados durante implementación y pruebas.
- Mejoras sugeridas por el feedback de los clientes.
- Modificaciones en el interfaz para mejorar la experiencia de usuario.
- Uso y aprendizaje de herramientas de testeo como *Mock* y *Espresso*, junto con la elaboración de los criterios de aceptación como scripts de *Gherkin* para automatizar las pruebas unitarias y de integración.
- Refactorización de código (lambda)

4.2 Ampliaciones

- Inclusión de un buscador de rutas.
- Incluir alertas.

VI. BIBLIOGRAFIA

1 Libros

- [LOsPi2010] Business model generation - (2010) - *Alexander Osterwalder & Yves Pigneur*
  Escrito por Alexander Osterwalder junto con Yves Pigneur, ambos profesores de la Universidad de Laussanne y especialistas en Sistemas de Información. Este libro se ha convertido en un referente en lo que a la creación de modelos de negocio se refiere. En el libro se analizan multitud de casos prácticos para explicar la generación del modelo de negocio, estrategias, procesos… y se dan pautas para crear tu propio modelo de negocio siguiendo la plantilla creada por los autores y a la que hago referencia en el capítulo "Producto" donde detallo mi modelo de negocio.

- [LKnSk2010] Kanban and Scrum, making the most of both - (2010) - *Henrik Kniberg & Mattias Skarin*
  Un libro que he encontrado muy útil a la hora de entender dos de las metodologías ágiles más usadas con una primera parte teórica en la que se explican y comparan las metodologías Kanban y Scrum y una segunda parte en la que se aplican los conocimientos a un caso de estudio real.
  https://www.infoq.com/minibooks/kanban-scrum-minibook

2 Online

- [W3TFL] Transport for London Unified API (última visita: septiembre 2017)
  Sitio oficial de la API de transporte de Londres.
  https://api.tfl.gov.uk
- [W3ADVL] Android developers (última visita: septiembre 2017)
  Sitio oficial de desarrolladores Android, en el que se encuentra la documentación oficial de Android (Clases, métodos, javadoc, ejemplos…)
  https://developer.android.com/index.html
- [W3STOV] StackOverflow(última visita: septiembre 2017)
  Foro de desarrolladores que se ha usado como soporte para dudas y errores.
  https://stackoverflow.com
- [W3JS] Jobserve (última visita: septiembre 2017)
  Portal de búsqueda de empleo para Reino Unido.
  http://www.jobserve.com/gb/en/Job-Search/
- [W3AM] Agile Manifesto (última visita: septiembre 2017)
  Sitio oficial donde se publica el manifiesto agile.
  http://agilemanifesto.org/
- [W3SG] Scrum guide (última visita: septiembre 2017)
  Sitio web con información sobre la metodología Scrum
  http://scrumguides.org
- [W3MWC] Mobile World Congress 2017 (última visita: septiembre 2017)
  Artículo sobre el mobile world congress de 2017 de donde se han sacado las estadísticas sobre el uso de dispositivos mobiles presentadas en el resumen
  http://cadenaser.com/ser/2017/02/28/ciencia/1488281552_888684.html
- [W3ACOR] Apache Cordova (última visita: septiembre 2017)
  Documentación oficial para el framework Apache Cordova
  http://cadenaser.com/ser/2017/02/28/ciencia/1488281552_888684.html
- [W3APL] Apple Links (última visita: septiembre 2017)
  Distintos enlaces a los requisitos para el desarrollo de aplicaciones nativas IOS, IDE Xcode, lenguaje SWIFT
  https://developer.apple.com/xcode/
  https://developer.apple.com/swift/

https://developer.apple.com/library/content/documentation/General/Conceptual/DevPedia-CocoaCore/Cocoa.html

- [W3STAT] Estadisticas y estudios (última visita: septiembre 2017)
  Portales de estadísticas y estudios, mencionados en el Capítulo I, sección 3 "Metodología", comparativa Android Vs IOS.
  https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/
  https://david-smith.org/iosversionstats/
  https://www.appannie.com/en/insights/market-data/app-annie-forecast-2017-mobile-app-store-revenue-exceed-139-billion-2021/
- [W3VOLL] Volley (última visita: septiembre 2017)
  Librería http para interactuar con APIs desde aplicaciones Android.
  https://github.com/google/volley
- [W3GOOG] Evento de google donde se presenta la nueva arquitectura de componentes (última visita: septiembre 2017)
  https://events.google.com/io/recap

3 Formación online

- [FOUD] Udacity

https://classroom.udacity.com/

- ***Android Development for Beginners:*** Learn the basics of Android and java programming, and take the first steps on your journey to becoming an Android developer.
- ***Android Basics: Multiscreen Apps****:* learn to build multiscreen apps using the foundation of Android.
- ***Android Basics: User Interface:*** in this course, you will learn the fundamentals of layouts in Android. You will learn how to use XML and change what is shows up on screen.
- ***Android Basics: User Input:*** in this course, you will learn how to build an interactive app which makes use of object-oriented principles
- ***Add Google Maps to your Android:*** Maps on mobile devices have change the world for millions of users. Learn how to use the Google Maps API to extend this functionality to your apps.
- ***Google Location Services on Android:*** enhance your apps with Google's Fused Location Provider, Activity Recognition, and Geofencing API capabilities.

VII. ÍNDICE DE FIGURAS

VIII. ÍNDICE DE TABLAS

IX. ANEXOS

I. Configuración entorno de desarrollo

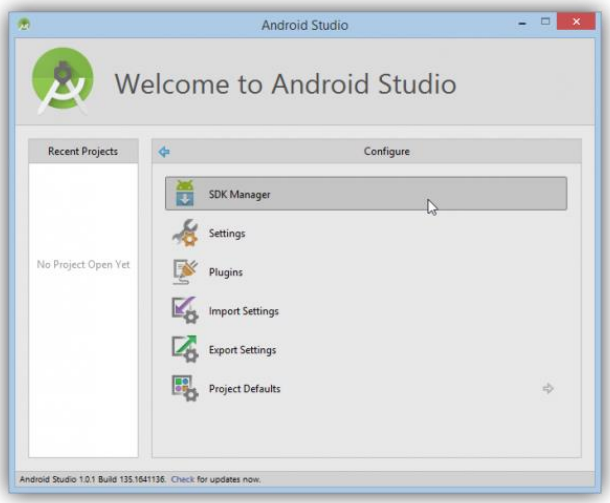
1.1 Configuración del IDE

Para poder trabajar sin problemas con Android Studio y poder aprovechar al máximo sus funciones tanto a nivel de programación como a nivel de virtualización es necesario instalar los componentes y los SDK correspondientes en nuestro ordenador de manera que el sistema sea capaz de disponer de todos los recursos del sistema operativo de Google siempre que los necesite.

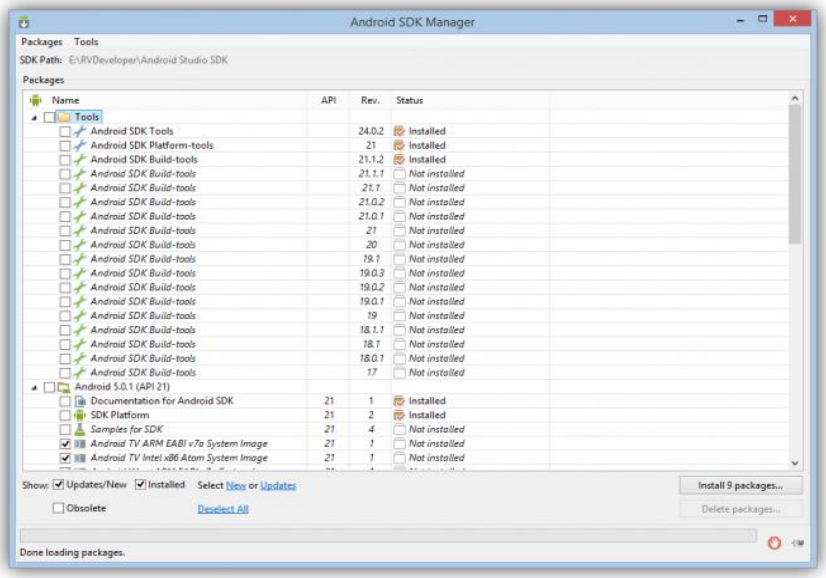
Una vez que hemos instalado ya Android Studio en nuestro ordenador lo ejecutamos y lo primero que veremos será el asistente principal de la herramienta.

Desde la siguiente ventana se pueden crear proyectos, abrir existentes y también tenemos acceso a la configuración:

Pulsando sobre ella llegaremos a una nueva ventana desde donde podemos cambiar los ajustes generales de Android Studio así como configurar los plugins y abrir el administrador del SDK. Esta última opción es la que debemos ejecutar para poder instalar nuevos SDK y componentes para la suite de desarrollo, por ello debemos pulsar sobre “SDK Manager”.



Se nos abrirá una ventana nueva con una completa lista de componentes y extensiones que podemos instalar y desinstalar de nuestra suite Android Studio.



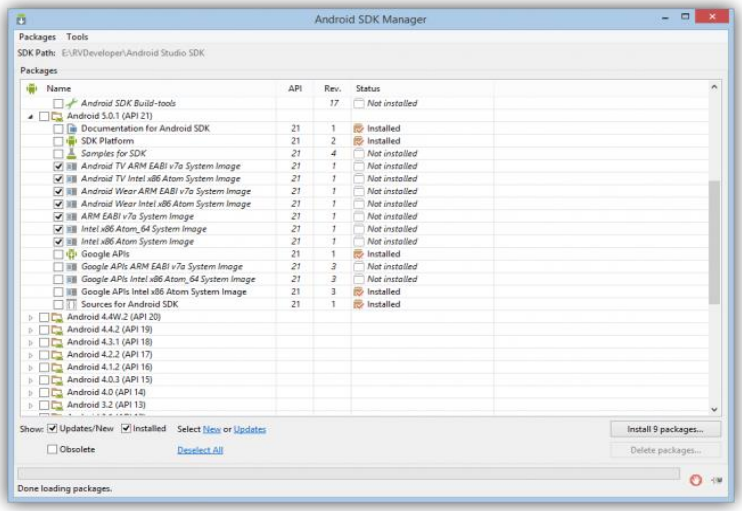
Por defecto al instalar esta herramienta de desarrollo se instalarán una serie de elementos base que, aunque pueden funcionar, es posible que necesitemos elementos adicionales para su correcto funcionamiento, por ejemplo, desarrollar para versiones anteriores a Android 5 o emularlas.

Podemos ver 3 elementos principales en esta ventana:

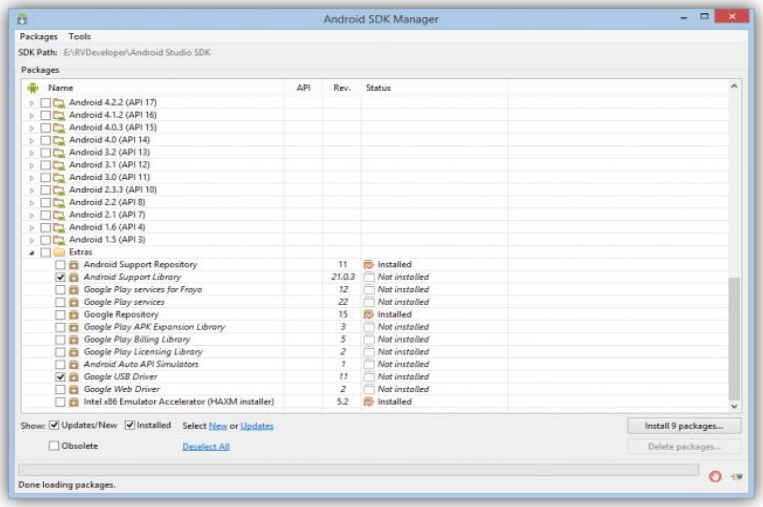
Tools: Desde este apartado instalaremos las herramientas seleccionadas con el SDK, necesarias para el desarrollo.

Android x.x (API X). Desde aquí instalaremos los componentes de la API necesarios para compilar las aplicaciones para las versiones correspondientes. Igualmente instalaremos las máquinas virtuales de dicha versión de Android, necesarias para poder montar una máquina virtual de Android.

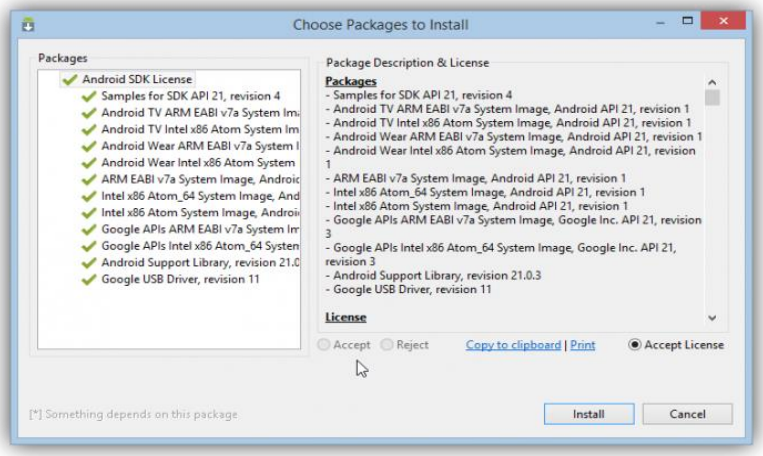
Desde aquí podemos instalar todas las API desarrolladas para el sistema operativo Android, desde la API 3 (correspondiente a Android 1.5) hasta la actual API 21 (correspondiente a Android 5.0.1).



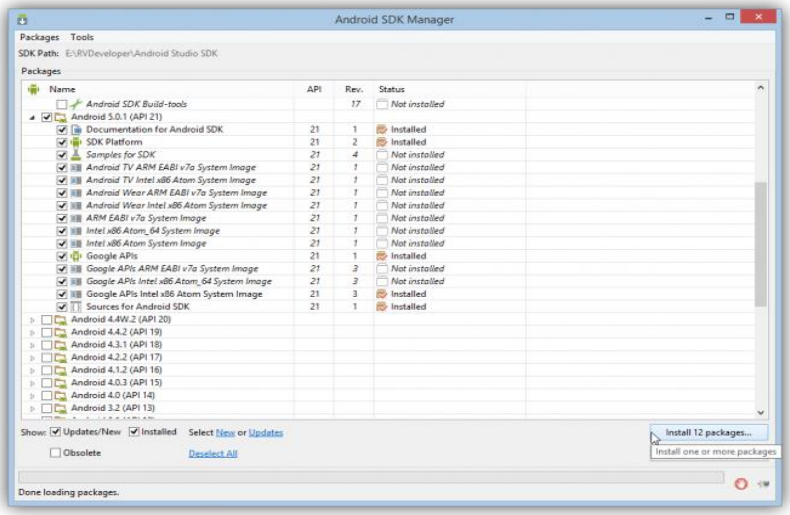
Extras. Desde este apartado podemos elegir si instalar o desinstalar componentes adicionales, por ejemplo, los servicios de Google Play, los drivers de aceleración para máquinas x86 o los controladores USB de desarrollo.



Desde aquí seleccionaremos todos los componentes que queremos descargar e instalar en nuestro sistema para que nuestro Android Studio funcione correctamente (por ejemplo, todos los elementos de la API 21 y los controladores USB si no están ya instalados) y pulsaremos sobre el botón "*Install*" para comenzar la descarga e instalación.



Debemos aceptar los términos de licencia antes de poder empezar con la descarga.

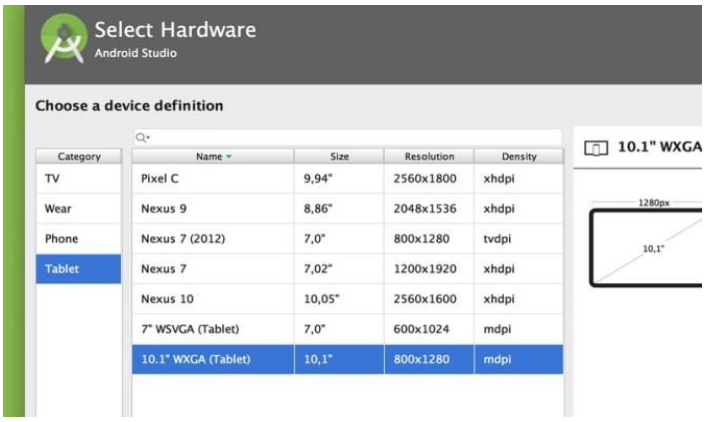


Una vez finalice el proceso los elementos estarán en nuestro disco duro, dentro de la carpeta SDK que hayamos configurado durante la instalación, y listos para ser utilizados.

1.2 Creación de un dispositivo virtual

Con el proyecto abierto, podemos diseñar el modelo que deseemos para el emulador sobre el que vamos a probar nuestra aplicación.

Desde **Tools > Android > AVD Manager** elegimos la opción **Create Virtual Device**. En la ventana se debe seleccionar qué tipo de dispositivo y la imagen de Android a descargar. Android Studio te recomienda algunas. Al decantarte por una, tendrás que pulsar en **Download** para que se descargue.



Una vez configurado el dispositivo virtual, al **hacer clic en el símbolo Play** se podrá elegir este dispositivo para la ejecución de la aplicación.

1.2.1 Cómo funciona el emulador

Dependiendo del ordenador en el que se ejecute, el emulador tardará más o menos en iniciarse. Básicamente funciona desde su ventana principal con una barra lateral que emula funciones del dispositivo como encender/apagar, **subir o bajar el volumen**, inclinar el dispositivo para verlo **vertical o apaisado**, botón Home, etc.

Desde la **barra lateral** tendrás acceso también a los demás componentes virtualizados del dispositivo.

El emulador de Android Studio muestra una versión funcional de Android con la mayoría de apps preinstaladas.

1.2.2 Opciones de desarrollo en los dispositivos Android

Las opciones de desarrollo es una característica de los terminales Android que proporciona un conjunto de herramientas para desarrolladores que crean aplicaciones y después son necesarias para probar y depurar aplicaciones de Android. **Estas opciones están creadas para el despliegue rápido de aplicaciones en Android**. También tienen diferentes características que hacen más fácil la conexión entre un ordenador y un terminal.

Podemos encontrar estas opciones en **Ajustes** y en el apartado de **Sistema**. Pero desde la introducción de las nuevas funciones de Jelly Bean, **los terminales actuales han ocultado esta función**, ya que un usuario normal no requiere esta opción. Pero si queremos acceder a estas opciones, **es muy fácil activarlo**.

1.2.3 Activar las opciones de desarrollo

- En Ajustes > Información del teléfono.
- Buscamos en la lista el Número de compilación.
- Pulsamos 7 veces seguidas el Número de compilación.

Ahora ya tendremos las Opciones de desarrollo activadas. Las podremos encontrar en Ajustes justo encima de la Información del teléfono.

1.2.4 Desactivar las opciones de desarrollo

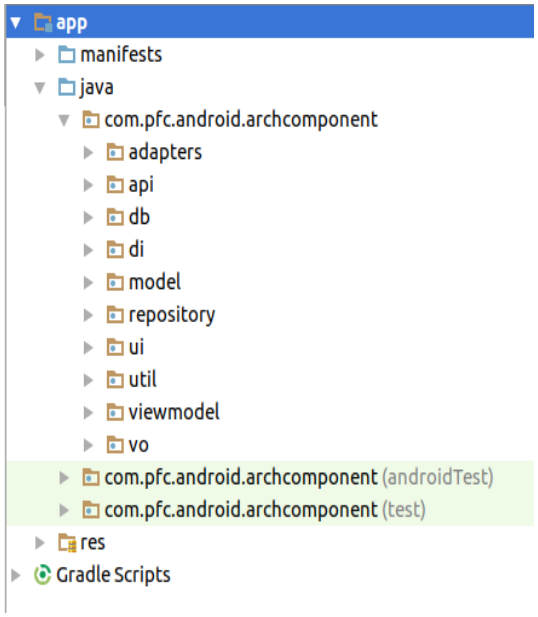
Ahora bien, si lo que queremos es desactivar estas opciones, podremos **desactivarlas fácilmente** siguiendo estos pasos:

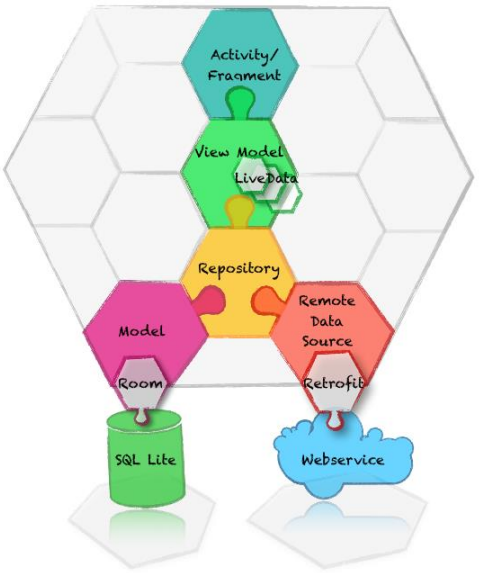
1. En **Ajustes** de nuestro terminal.
2. Entramos a la opción **Aplicaciones**.
3. Movemos las ventanas hasta llegar a **Todas**.
4. En la lista de aplicaciones, buscamos **Ajustes** y seleccionamos esa aplicación.
5. Presionamos el botón **Borrar datos**.

II. Arquitectura de componentes Android

En este apartado vamos a mostrar las características de la Arquitectura de componentes con ejemplos específicos del código.

Centrándonos primero en la estructura recomendada por Google para una aplicación móvil siguiendo las pautas marcadas por la arquitectura de componentes tendríamos la estructura de la derecha, que ha sido implementada como se ve en la imagen de la izquierda.





Vamos a empezar a analizar junto con el código de los repositorios (*SQLite* y API) hacia el Interfaz de Usuario (*Fragments* y *Activity*).

Para la recuperación de datos del API de transporte se ha usado *Retrofit*. Que a través de las clases de *Repository* deberá ponerse en contacto con el *ViewModel* como se puede observar en el diagrama.

La clase que se encarga de hacer la llamada al API es *TflApiService* y se encuentra en el directorio api.

```
public interface TflApiService {

    /**
     * Gets the list of arrival predictions for the given stop point id
     * <p>
     * https://api.tfl.gov.uk/StopPoint/{id}/Arrivals?app_id=xxx&app_key=xxxx
     *
     * @param id
     * @param api_transport_id
     * @param api_transport_key
     * @return Call<List<ArrivalsEntity>>
     */
    @GET("StopPoint/{id}/Arrivals")
    Call<List<ArrivalsEntity>> getArrivalInformation(
            @Path("id") String id,
            @Query("app_id") String api_transport_id,
            @Query("app_key") String api_transport_key
    );
```

La estructura de los elementos que se van a recuperar en el API, las tenemos definidas como *Entities* dentro de la carpeta vo. Un ejemplo de una de las *Entity* es:

```java
public class StopPointsEntity {
    private final String TAG = StopPointsEntity.class.getName();
    @SerializedName("$type")
    @Expose
    private String $type;
    @SerializedName("naptanId")
    @Expose
    private String naptanId;
    @SerializedName("indicator")
    @Expose
    private String indicator;
    @SerializedName("stopLetter")
    @Expose
    private String stopLetter;
```

Se ha delegado en Dagger, la generación de la URL,y el procesamiento para la llamada a través de Retrofit, de modo que:

```java
@BuStopAppScope
@Component(modules = {TflApiModule.class})
public interface TflApiComponent {
    /**...*/
    public TflApiService getRetrofit();

}
```

para conseguir una instancia de *TflApiService*, se llama a un método de la clase ***TflApiModule*** que devuelve un *TflApiService*. Esta clase, como se ve en la siguiente captura, tiene dos métodos, uno que devuelve el ***TflApiService***, pero que necesita un ***Retrofit*** de parámetro de entrada. Y otro método que devuelve un objeto ***Retrofit***, justo como el que necesitamos, pero que tiene un parámetro de entrada ***Context***. Esta clase tiene un *include* de la clase *ContextModulo*, de la que podemos obtener ese Context que nos hace falta.

```java
/**...*/
@Module(includes = {ContextModule.class})
public class TflApiModule {

    /**...*/
    @Provides
    @BuStopAppScope
    public TflApiService provideTflApiService(Retrofit retrofit) {
        return retrofit.create(TflApiService.class);
    }

    /**...*/
    @Provides
    @BuStopAppScope
    public Retrofit provideRetrofit(Context context) {
        return new Retrofit.Builder()
                .addConverterFactory(GsonConverterFactory.create())
                .baseUrl("https://api.tfl.gov.uk/")
                .build();
    }
}
```

En el diagrama podemos ver que la siguiente capa *Repository,* es la encargada de comunicar la información que le da el *TflApiService* al *ViewModel.* Esta clase en el código es *RemoteRepository:*

```java
public class RemoteRepositoryImpl implements RemoteRepository{

    public static final String TAG = RemoteRepositoryImpl.class.getSimpleName();

    private TflApiService mApiService;
    private String mAppId;
    private String mAppKey;

    /**
     * Contructor with a context parameter to interact with the API backend.
     * <p>
     * This method  allows the communication withe the API transport backend
     *
     */
    public RemoteRepositoryImpl(Context context) {...}

    /**
     * Method async with fourth parameters: latitude, longitude, radius and
     * a callback in order to make a call to the API transport. In the last one is where the answer
     * of the API is saved.
     * <p>
     * This method return the information about the Stops near to the location
     * given through the parameters (lat, lon and radious).
     *
     * @param  lat  a double number giving the latitude of the location.
     * @param  lon  a double number giving the longitude of the location.
     * @param  radius  an int number giving the radius of the search.
     * @param  callback  a Callback<StopLocationEntity> giving the object in order to keep the reply of the API.
     */
    public void getStopLocation(double lat, double lon, int radius, Callback<StopLocationEntity> callback) {
        Call<StopLocationEntity> call = mApiService.getStopLocation(mAppId,mAppKey,lat, lon, radius);
        call.enqueue(callback);
    }
```

Si analizamos la recuperación de la información de la BBDD, se ha seguido un modelo parecido al descrito para el API. Se ha creado una *Entity* que es nuestro modelo y que va a definir la estructura que va a tener nuestra BBDD:

```java
@Entity(tableName = "favourites")
public class ArrivalsFormatedEntity implements Serializable {

    @PrimaryKey(autoGenerate = true)
    public int mId;

    private String naptanId;

    private String lineId;
```

Se pueden usar una gran cantidad de anotaciones, en este caso con la *PrimaryKey* ha sido suficiente.

Ahora vamos a pasar a analizar el Data Access Object (DAO), que será la clase que realmente interactue con la BBDD, en esta clase también podemos ver unas cuentas anotaciones, como son el @Insert, @Query…

```java
@Dao
public interface FavouriteDao {
    @Query("SELECT * FROM favourites")
    LiveData<List<ArrivalsFormatedEntity>> getFavouritesLiveData();

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    void addFavourite(ArrivalsFormatedEntity favourite);

    @Delete
    void deleteFavourite(ArrivalsFormatedEntity favourite);

    @Query("SELECT COUNT(*) FROM favourites WHERE naptanid = :naptanid and lineid = :lineid")
    int elementInFav(String naptanid, String lineid);

}
```

Ahora pasamos a la creación de *Room Database*, Como podemos ver tenemos anotaciones que dicen *i)* qué entidad es la que nos da la estructura de la BBDD, *ii)* que vamos a necesitar un *TypeConverter* que hemos generado nosotros y *iii)* que el objeto de acceso a BBDD es el *FavouriteDao*.

```
@Database(entities = {ArrivalsFormatedEntity.class}, version = 1)
@TypeConverters({StringConverter.class})
public abstract class AppDatabase extends RoomDatabase {

    public abstract FavouriteDao favouriteDao();
}
```

Ahora el único punto que nos faltaría es la creación de la BBDD. Queremos que se haga una sola vez en la vida de la aplicación, y a través de un método *Singleton* para asegurarnos de que solo hay una instancia de la BBDD sobre la que se está consultando. En este caso hemos aprovechado a usar *Dagger (Dependency Injection framework)*

```
public class FavouriteApplication extends Application {

        private FavouriteComponent favouriteComponent;

        @Override
        public void onCreate() {
            super.onCreate();

            // Dagger%COMPONENT_NAME%
            favouriteComponent = DaggerFavouriteComponent.builder()
                    // list of modules that are part of this component need to be created here too
                    .favouriteModule(new FavouriteModule(this))
                    .build();
        }

        public FavouriteComponent getFavComponent() { return favouriteComponent; }

}
```

Como se puede ver, se ha delegado la creación de la BBDD a *Dagger*:

```
@Module
public class FavouriteModule {
    public static final String TAG = FavouriteModule.class.getSimpleName();

    static final String DATABASE_NAME = "favourite-db";
    private FavouriteApplication app;

    public FavouriteModule(FavouriteApplication app) { this.app = app; }

    @Provides
    Context getApplicationContext() { return app; }

    @Provides
    @Singleton
    AppDatabase getTaskDatabase(Context context) {

        //Remove the database information
        //context.deleteDatabase(DATABASE_NAME);
        //create the database with Room
        AppDatabase db = Room
                .databaseBuilder(context.getApplicationContext(), AppDatabase.class, DATABASE_NAME)
                .build();

        Log.d(TAG,"db created");
        return db;
    }
}
```

La manera de que esta información llegue a los ViewModel es a través del *LocalRepository* a través de métodos *async*, como el de añadir a BBDD:

```
public class LocalRepositoryImpl implements LocalRepository {

    private final String TAG = LocalRepositoryImpl.class.getName();

    public AppDatabase database;

    /**...*/
    public LocalRepositoryImpl(AppDatabase database) { this.database = database; }

    /**
     * Method that return a LiveData object without parameters. This method
     * made an async select in order to get all the elements that are in the database.
     * <p>
     * This method get a LiveData object with a list of ArrivalsFormatedEntitites.
     *
     * @return     a LiveData<List<ArrivalsFormatedEntity>> object giving to get all the favourites saved in the database.
     *
     */
    public LiveData<List<ArrivalsFormatedEntity>> getFavouritesLiveData(){
        return database.favouriteDao().getFavouritesLiveData();
    }

    /**
     * Method async with one parameter: ArrivalsFormatedEntity. This method
     * made an async insert to the database.
     * <p>
     * This method insert the favouriteEntity in the database.
     *
     * @param  favouriteEntity  a ArrivalsFormatedEntity object giving to add to the database.
     */
    public void addFavourite(ArrivalsFormatedEntity favouriteEntity) {
        new AsyncTask<ArrivalsFormatedEntity, Integer, Void>() {
            @Override
            protected Void doInBackground(ArrivalsFormatedEntity... params) {
                database.favouriteDao().addFavourite(params[0]);
                Log.d(TAG, "element added");
                return null;
            }
        }.execute(favouriteEntity);
    }
```

Por lo tanto, tendríamos ahora mismo en la capa de *Repository*, que está formada por un *Repository* local (DDBB) y un *Repository* remoto (API) la información.

Esta información, es el ***ModelView*** a través de los *LiveData/MutableLiveData* la que se la envía y actualiza al Interfaz de Usuario (UI), que son los *Fragment* y la *Activity.*

```java
public class UnifiedModelView extends AndroidViewModel {

    private final String TAG = UnifiedModelView.class.getName();

    private MutableLiveData<List<ArrivalsFormatedEntity>> mMutableArrivalsFormated;
    private LocationLiveData mLocationLiveData;
    private MutableLiveData<StopLocationEntity> mStopPointMutableLiveData;
    private MutableLiveData<List<ArrivalsFormatedEntity>> mMutableLineFav;

    @Inject
    public AppDatabase database;

    RemoteRepository mRemoteRepository;
    LocalRepository mLocalRepository;

    /**
     * Contructor with a application parameter to interact with all the livedata
     * <p>
     *
     * @param application Application
     */
    public UnifiedModelView(Application application) {
        super(application);
        ((FavouriteApplication)application).getFavComponent().inject(this);
        this.mMutableArrivalsFormated = new MutableLiveData<>();
        this.mRemoteRepository = new RemoteRepositoryImpl(application);
        this.mLocalRepository = new LocalRepositoryImpl(database);
        this.mStopPointMutableLiveData = new MutableLiveData<>();
        // This set own location if the GPS is enabled into the device.
        this.mLocationLiveData = new LocationLiveData(application.getApplicationContext());
        this.mMutableLineFav = new MutableLiveData<>();
    }
```

Así el ***ModelView*** que extiende del *AndroidViewModel* es el que crea y devuelve los objetos que están enlazados a un especifico *lifecycle*. Un ViewModel almacena el estado de los datos de la vista y comunica con otros componentes, como los repositorios de datos o la capa de lógica de negocio.

El ***LiveData*** permite observar los cambios de datos que ocurren a través de múltiples componentes sin crearlos explícitamente, ni crear dependencias muy rígidas. Los ***LiveData*** respetan la complejidad de los *Lifecycles*, incluyendo actividades, fragmentos servicios o cualquier *LifecycleOwner* definido. Los ***LiveData*** controlan las subscripciones de los observadores parando subscripciones para parar los *LyfecycleOwner*, y cancelan suscripciones de *LifecycleOwner* que han finalizado.

Así se puede ver el set y el get de un *MutableLiveData* que tenemos creado en el ***ViewModel***

```
/**
 * This Method return the MutableLiveData mMutableArrivalsFormated
 * <p>
 * @return   MutableLiveData<List<ArrivalsFormatedEntity>>
 */
public MutableLiveData<List<ArrivalsFormatedEntity>> getmMutableArrivalsFormated(){
    return mMutableArrivalsFormated;
}

/**
 * This Method makes the setValue of the MutableLiveData:mMutableArrivalsFormated
 * <p>
 * @param   naptanId String
 * @param   lat  String
 * @param   lon  String
 */
public void setmMutableArrivalsFormated(@NonNull String naptanId, @NonNull String lat, @NonNull String lon) {
    mRemoteRepository.getArrivalInformation(naptanId, new Callback<List<ArrivalsEntity>>() {
        @Override
        public void onResponse(Call<List<ArrivalsEntity>> call, Response<List<ArrivalsEntity>> response) {
            List<ArrivalsFormatedEntity> arrivalsFormatedEntity = convert(response.body(),lat,lon);
            mMutableArrivalsFormated.setValue(arrivalsFormatedEntity);
        }

        @Override
        public void onFailure(Call<List<ArrivalsEntity>> call, Throwable t) {
            Log.e(TAG, "error occured: " + t.toString());
            Toast.makeText(getApplication().getBaseContext(), t.getMessage(), Toast.LENGTH_SHORT).show();
        }
    });
}
```

El LifecycleOwner/LifecycleRegistryOwner, ambos son interfaces que son implementados en el LifecycleActivity y en el LifecycleFragment.

```
public class ListFragment extends LifecycleFragment {

    private static final String TAG = ListFragment.class.getName();
    protected RecyclerView mRecyclerView;
    protected DataAdapter mAdapter;
    private UnifiedModelView unifiedModelView;


    /**...*/
    @Override
    public void onCreate(Bundle savedInstanceState) {...}

    /**...*/
    @Override
    public void onResume() {...}
    /**
     * observeHandlers Method
     * <p>
     * In this method we handle changes emitted by the  location livedata and the stoppoints livedata
     * <p>
     */
    private void observeHandlers() {
        // Get current location
        unifiedModelView.getLmLocationLiveData().observe(this, defaultLocation -> {
            //set the stops near my current location
            unifiedModelView.setStopPointMutableLiveData(defaultLocation.getLatitude(), defaultLocation.getLongitude(),200);
        });

        // Handle changes emitted by StopPointMutableLiveData
        unifiedModelView.getmStopPointMutableLiveData().observe(this, stopLocationEntity -> {
            handleResponse( stopLocationEntity.getStopPoints() ) ;
        });
    }
```

Así el *ViewModel* es el encargado de avisar de cualquier cambio a los ***fragmentos*** y a la ***actividad*** que tenemos. Y estos junto con los ***Adapter*** son los encargados de pintar las distintas partes de la aplicación junto con los *layouts*, formando así parte de la Interfaz de Usuario, la última capa de nuestro diagrama.

Así podemos ver como se pinta uno de los listados de la aplicación a través de los ***RecyclerView***:

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {

    View rootView = inflater.inflate(R.layout.fragment_recycler, container, false);
    rootView.setTag(TAG);
    mRecyclerView = (RecyclerView) rootView.findViewById(R.id.recycler_view);
    mRecyclerView.setRecycledViewPool(new RecyclerView.RecycledViewPool());
    mRecyclerView.setLayoutManager(new LinearLayoutManager(getContext(), LinearLayoutManager.VERTICAL, false));
    mRecyclerView.hasFixedSize();

    CustomDetailClickListener mDetailClickListener = new CustomDetailClickListener() {
        @Override
        public void onItemClick(View v, int position) {
            FragmentManager fm = getFragmentManager();
            Bundle arguments = new Bundle();
            arguments.putString("naptanId", mAdapter.getNaptanIdByPosition(position));
            arguments.putString("lat", mAdapter.getLatByPosition(position));
            arguments.putString("lon", mAdapter.getLonByPosition(position));
            DetailFragment detailfragment = new DetailFragment();
            detailfragment.setArguments(arguments);
            fm.beginTransaction().replace(R.id.content_fragment, detailfragment).addToBackStack("detail").commit();
        }
    };
    mAdapter = new DataAdapter(getContext());
    mAdapter.setOnItemClickListener(mDetailClickListener);
    // Set CustomAdapter as the adapter for RecyclerView.
    mRecyclerView.setAdapter(mAdapter);
    return rootView;
}
```

Y de los *Adapter*:

```
    @Override
    public int getItemCount() { return mStopPoints.size(); }

    /**...*/
    @Override
    public Holder onCreateViewHolder(ViewGroup parent, int viewType) {
        View row = LayoutInflater.from(parent.getContext()).inflate(R.layout.item_row, parent, false);
        final Holder mViewHolder = new Holder(row);
        row.setOnClickListener(new View.OnClickListener(){
            @Override
            public void onClick(View v){
                if(detailListener!=null) {
                    detailListener.onItemClick(v, mViewHolder.getAdapterPosition());
                }
            }
        });
        return mViewHolder;

    }

    /**...*/
    @Override
    public void onBindViewHolder(final Holder holder, int position) {...}
```

Esta arquitectura facilita el testeo a través de diferentes componentes:

Test unitarios:

- *ViewModel* se puede testear usando *Mock* en el *Repository*.
- *Repository* se puede testear usando *Mock* en el *WebService* de *Retrofit* y en el DAO de *Room*.

- El DAO también se puede probar usando *Junit*, ya que *Room* permite que se le pase una implementación de *SupportSQLiteOpenHelper*.

- El *Webservice* se puede probar usando *MockWebServer* para simular las respuestas de backend.

- La UI y la interacción de usuario la probaremos mediante *Espresso*. Como la UI solo habla con el *ViewMode*, será suficiente con usar *Mock* en este.

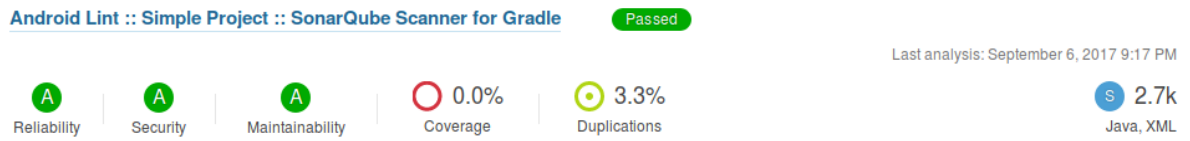
Debido a problemas de tiempo, no se han probado en el curso de este proyecto, pero se han propuesto como posibles mejoras.

III. Métricas

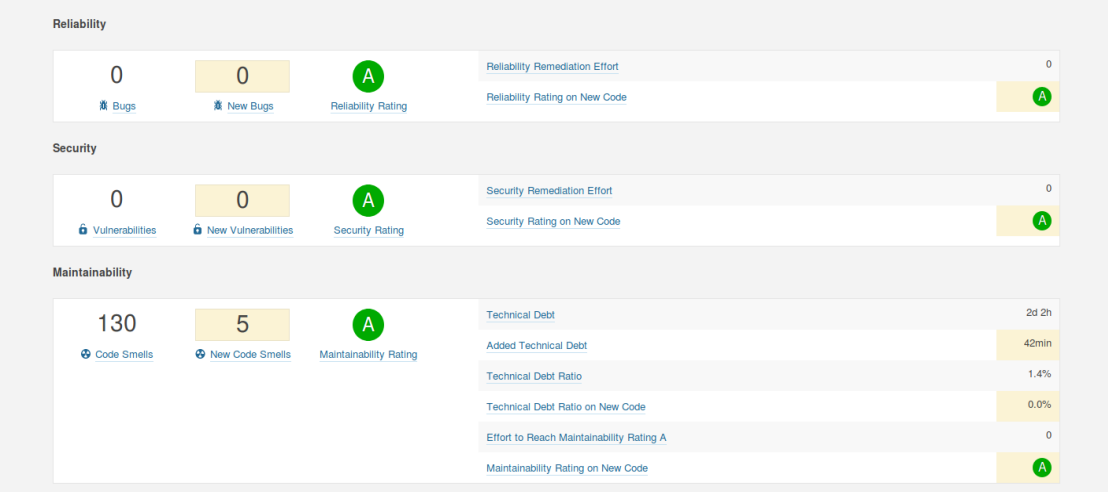
1.1 Métricas estáticas

Para poder obtener estos datos la herramienta que se ha usado es SonarQube (https://www.sonarqube.org). Según esta herramienta el código ha pasado los test.

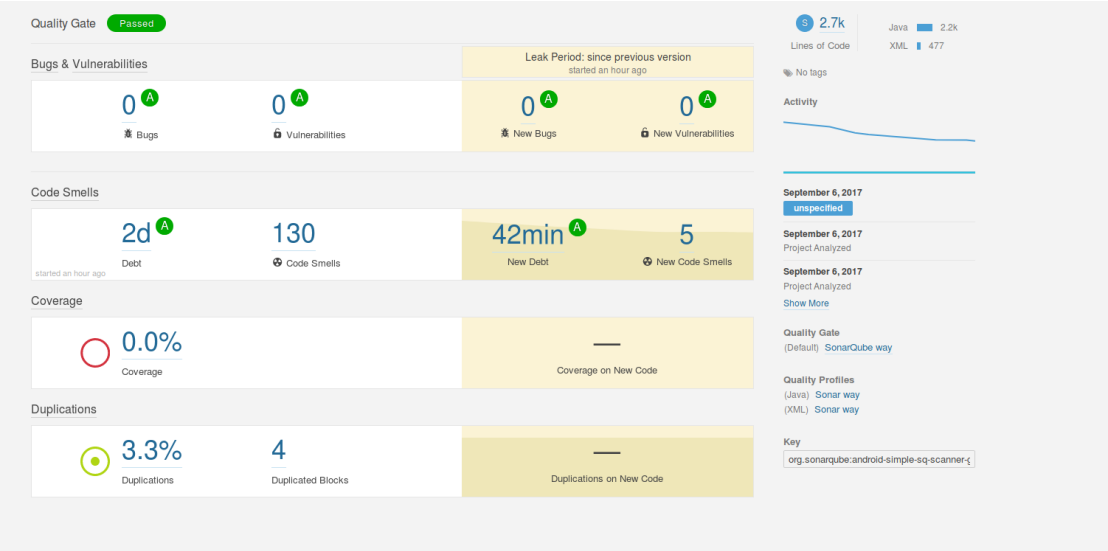
La primera captura es un resumen del estado del proyecto, en él, se puede ver que tiene una categoría A, que significa que ha pasado los test referentes a fiabilidad, seguridad y mantenimiento:



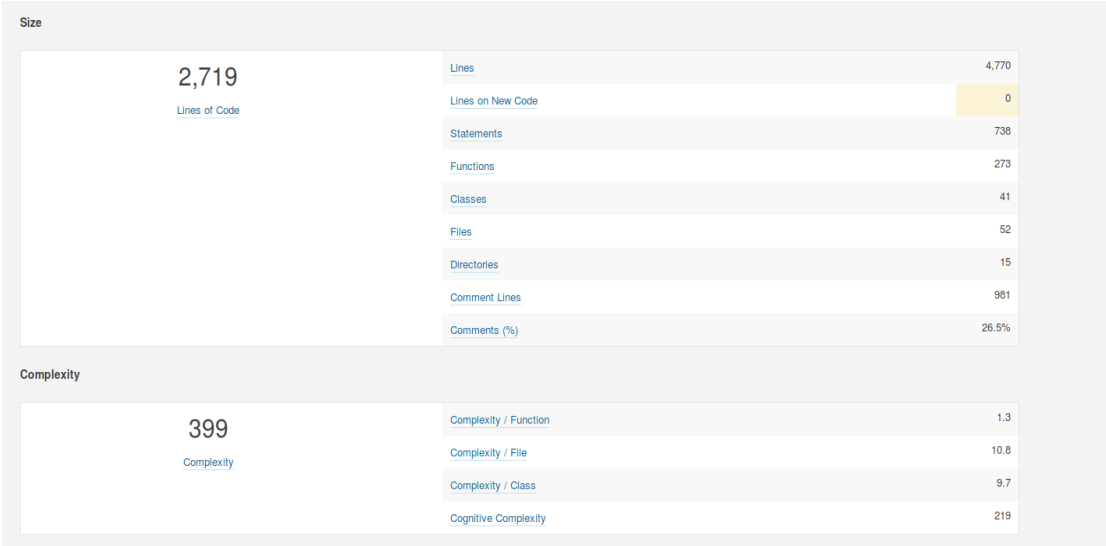
Más en detalle, podemos ver qué aspectos se han considerado para obtener estos resultados para fiabilidad, seguridad y mantenimiento:



En la página General del proyecto podemos ver el resultado de nuestro test, así como una visión general de cómo ha evolucionado el código después de algunos cambios ejecutados.



Este es el detalle sobre las líneas de código que son 2,719 con 2200 en java y 477 en xml:



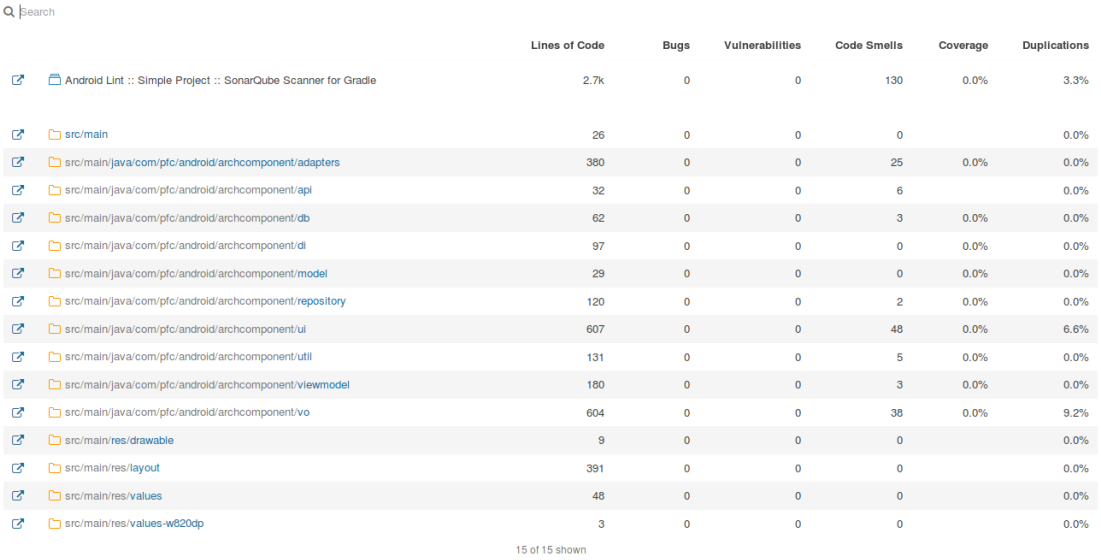
Size

2,719 Lines of Code	
Lines	4,770
Lines on New Code	0
Statements	738
Functions	273
Classes	41
Files	52
Directories	15
Comment Lines	981
Comments (%)	26.5%

Complexity

399 Complexity	
Complexity / Function	1.3
Complexity / File	10.8
Complexity / Class	9.7
Cognitive Complexity	219

Y en esta última captura se puede ver agrupados por paquetes las vulnerabilidades, incidencias, líneas de código



Search

	Lines of Code	Bugs	Vulnerabilities	Code Smells	Coverage	Duplications
Android Lint :: Simple Project :: SonarQube Scanner for Gradle	2.7k	0	0	130	0.0%	3.3%
src/main	26	0	0	0		0.0%
src/main/java/com/pfc/android/archcomponent/adapters	380	0	0	25	0.0%	0.0%
src/main/java/com/pfc/android/archcomponent/api	32	0	0	6		0.0%
src/main/java/com/pfc/android/archcomponent/db	62	0	0	3	0.0%	0.0%
src/main/java/com/pfc/android/archcomponent/di	97	0	0	0	0.0%	0.0%
src/main/java/com/pfc/android/archcomponent/model	29	0	0	0	0.0%	0.0%
src/main/java/com/pfc/android/archcomponent/repository	120	0	0	2	0.0%	0.0%
src/main/java/com/pfc/android/archcomponent/ui	607	0	0	48	0.0%	6.6%
src/main/java/com/pfc/android/archcomponent/util	131	0	0	5	0.0%	0.0%
src/main/java/com/pfc/android/archcomponent/viewmodel	180	0	0	3	0.0%	0.0%
src/main/java/com/pfc/android/archcomponent/vo	604	0	0	38	0.0%	9.2%
src/main/res/drawable	9	0	0	0		0.0%
src/main/res/layout	391	0	0	0		0.0%
src/main/res/values	48	0	0	0		0.0%
src/main/res/values-w820dp	3	0	0	0		0.0%

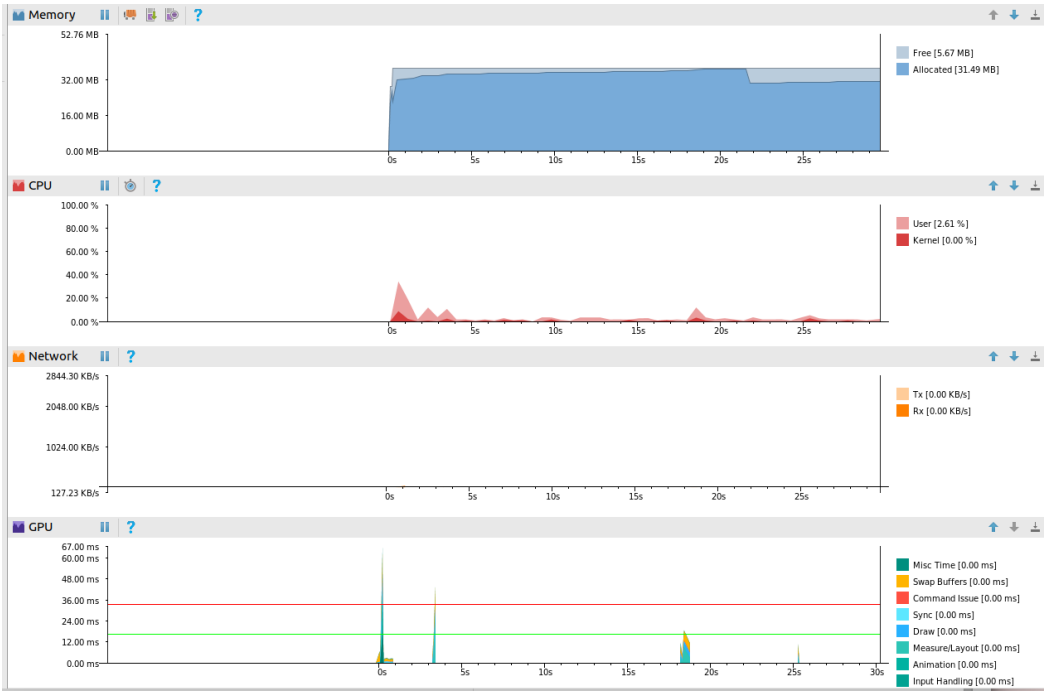
15 of 15 shown

1.2 Métricas dinámicas

Estas métricas se han obtenido directamente al ejecutar la aplicación con el Android Studio, mostrándose el comportamiento de la Memoria, la CPU, la red y la GPU.

La primera captura muestra el comportamiento de los distintos elementos nada más encender la aplicación.

El usuario, para esta prueba, entró en la aplicación por primera vez, seleccionó una de las paradas, añadió una de las líneas que aparecen en el detalle a favoritos y por último borró la entrada:

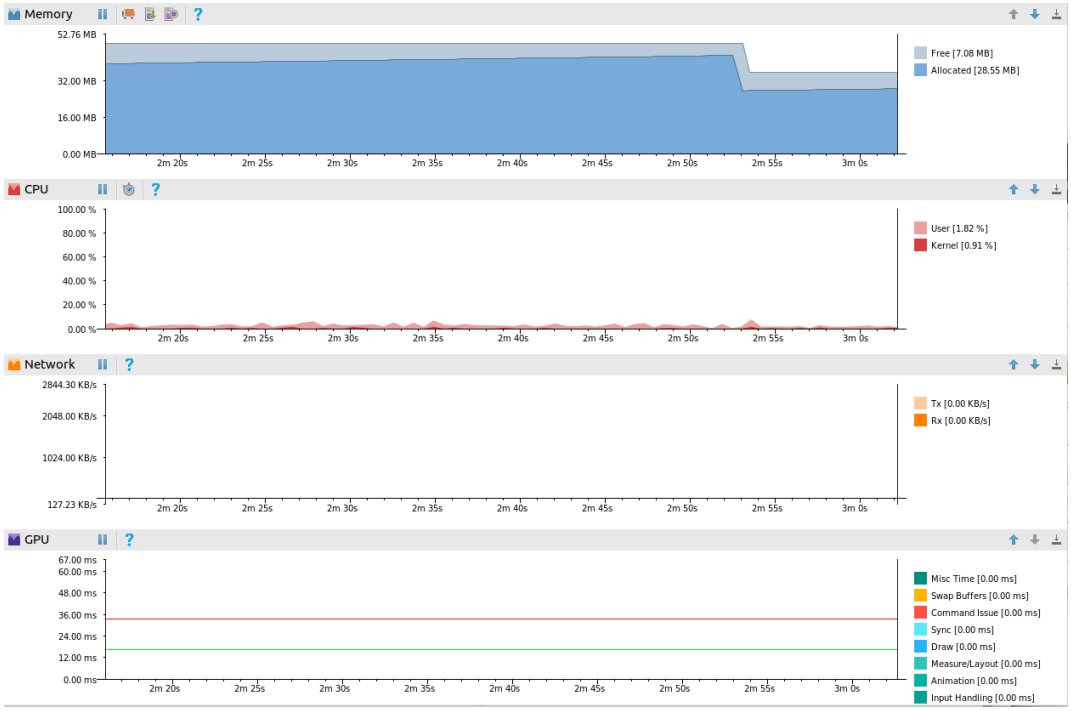


Lo que se puede observar es que el hecho de interactuar con la BBDD y el API no hace que ninguna de las gráficas se dispare, el garbage collector funciona liberando memoria, con lo cual no parece que cause memory leaks. Manteniéndose los valores dentro de unos rangos razonables.

Estas son las gráficas obtenidas al iniciar la aplicación y detectar algún favorito:

Esta vez se ve un *spike* en el procesamiento de imágenes por el cargado inicial del mapa, destacando este porque el resto de tiempos son muy buenos.

Esto es lo que pasa con las métricas de la aplicación cuando esta pasa a un segundo plano, porque se está ejecutando otra aplicación:

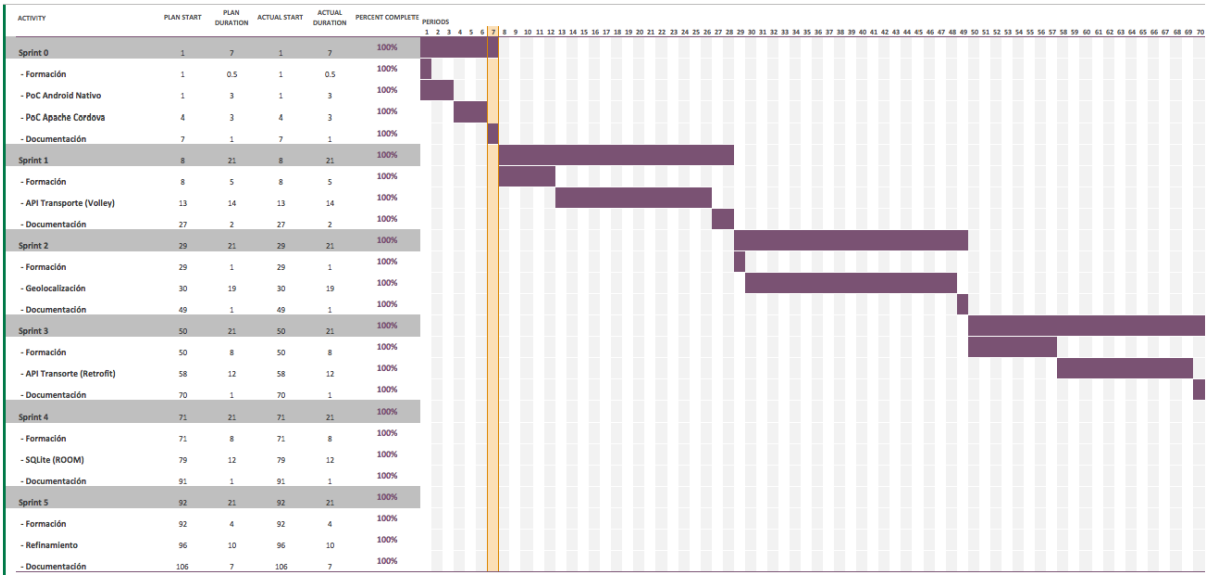


En esta gráfica cabe destacar el comportamiento de la memoria, donde se puede observar como se libera memoria al pasar la aplicación al segundo plano, haciendo que nuestra aplicación no consuma recursos del dispositivo.

IV. Diagrama GANTT

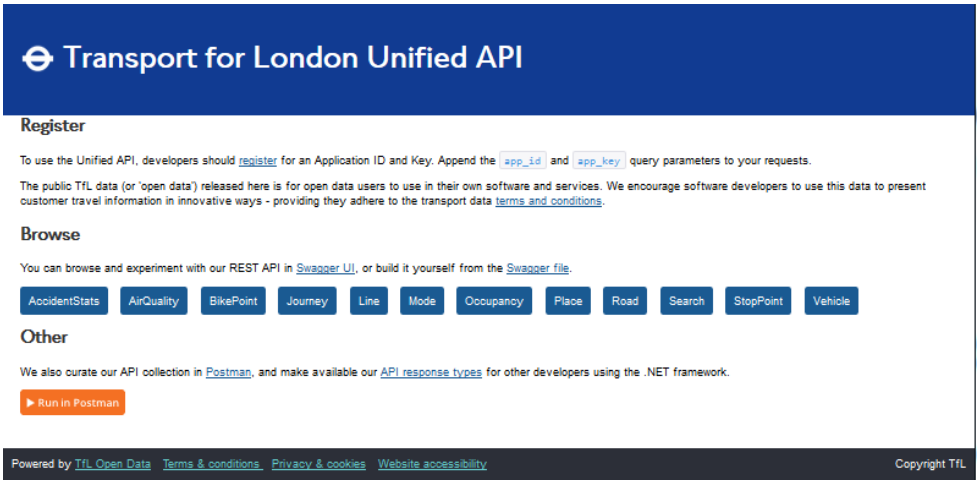
En este apéndice se muestra la inversión temporal que se ha empleado en el proyecto así como la secuencia de las actividades expuestas previamente en las Figuras 6 y 7 de la sección 2.2 Planificación del capítulo II Proyecto.

Este proyecto empezó el 15 de mayo y acabo el 4 de septiembre. Este diagrama GANTT ha sido incluido simplemente por si pudiera aportar más claridad a los tiempos empleados en las distintas tareas que forman parte de los Sprint, aunque no es un diagrama empleado en las metodologías agiles.



ACTIVITY	PLAN START	PLAN DURATION	ACTUAL START	ACTUAL DURATION	PERCENT COMPLETE
Sprint 0	1	7	1	7	100%
- Formación	1	0.5	1	0.5	100%
- PoC Android Nativo	1	3	1	3	100%
- PoC Apache Cordova	4	3	4	3	100%
- Documentación	7	1	7	1	100%
Sprint 1	8	21	8	21	100%
- Formación	8	5	8	5	100%
- API Transporte (Volley)	13	14	13	14	100%
- Documentación	27	2	27	2	100%
Sprint 2	29	21	29	21	100%
- Formación	29	1	29	1	100%
- Geolocalización	30	19	30	19	100%
- Documentación	49	1	49	1	100%
Sprint 3	50	21	50	21	100%
- Formación	50	8	50	8	100%
- API Transorte (Retrofit)	58	12	58	12	100%
- Documentación	70	1	70	1	100%
Sprint 4	71	21	71	21	100%
- Formación	71	8	71	8	100%
- SQLite (ROOM)	79	12	79	12	100%
- Documentación	91	1	91	1	100%
Sprint 5	92	21	92	21	100%
- Formación	92	4	92	4	100%
- Refinamiento	96	10	96	10	100%
- Documentación	106	7	106	7	100%

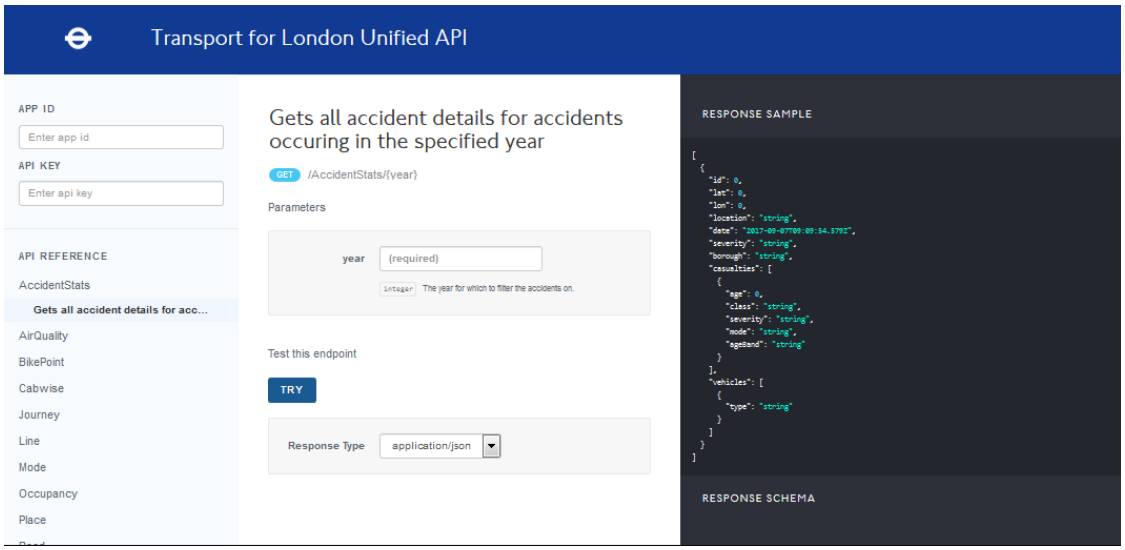
V. API de transporte



Para poder usar el API de transporte, se necesita estar registrado con un usuario y una contraseña como se puede ver en la página de información de la API:

Enlace al *swagger* de la aplicación desde el que se pueden probar los distintos métodos.

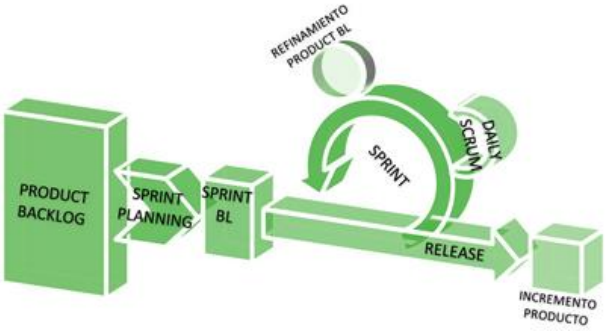
https://api.tfl.gov.uk/swagger/ui/index.html?url=/swagger/docs/v1#!/AccidentStats/AccidentStats_Get



VII. Metodología

No es el propósito del anexo explicar la metodología Scrum ya que hay una extensa documentación en internet al respecto, en concreto en los enlaces provistos en la bibliografía **[W3AM] [W3SG].** No obstante se detallan las dos comparativas realizadas como parte del estudio, en las que se enfrenta la metodología Ágil a la tradicional en cascada y otra en la que se enfrentan tres de las más usadas metodologías agiles.

1.1 Comparativa Tradicional vs. Ágil.

Extreme Programming (XP)	Kanban
	
Mayor dificultad para entender el estado del proyecto	Mayor transparencia y visibilidad del proyecto
Gestionada por project managers	Equipos autogestionados
Dificultad para introducir cambios	Facilidad para acomodar cambios
Dificultad para encontrar errores antes de la entrega	Temprana detección y correción de errores
Entregas predefinidas y completas	Entregas iterativas e incrementales
La prioridad es el proyecto	La prioridad es el producto
Todos los requisitos cerrados antes de desarrollo	Los requisitos cambiarán y se ampliarán durante el desarrollo
Implantación sólo cuando el proyecto/fases se han finalizado	Ciclos de Implantación más eficientes al usar integración continua
Mayor dificultad para entender el estado del proyecto	Mayor transparencia y visibilidad del proyecto

1.2 Scrum vs otras metodologías Ágiles

Existen numerosas metodologías ágiles que se pueden categorizar desde las más prescriptivas hasta aquellas con menos reglas. Cada organización deberá analizar cuál se adapta mejor a su producción. La comparativa siguiente muestra tres de las metodologías agiles más usadas, desde la más prescriptiva (XP) a la más liberal (Kanban), dejando Scrum en un punto medio, el cuál considero mejor para una empresa que desea adoptar agile desde cero, ya que impone una serie de pautas útiles para que la gente siga un proceso, pero sin ser demasiado complejo cómo podría resultar XP. Lo ideal es que la experiencia haga evolucionar y adaptar la metodología elegida, pero sólo cuándo el equipo tenga el conocimiento y la madurez requeridos para poder hacer este proceso de adaptación/cambio con criterio.

Extreme Programming (XP)	Scrum	Kanban
Iteraciones variables	Iteraciones fijas	Iteraciones fijas o variables según equipo
Limitación por iteración	Limitación por sprint	Limitación por estado
No introducir tareas durante iteración	No introducir tareas durante sprint	Pueden añadirse tareas si hay capacidad
13 Reglas	9 Reglas	3 Reglas
Facilitador: Coach/Big Boss	Facilitador: Scrum master	Facilitador: N/A
Iteraciones más cortas 1-3 semanas	Iteraciones más largas 2-4 semanas	N/A
La prioridad la marca el Cliente	La prioridad la marca el PO	N/A