GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

Las nuevas tecnologías aplicadas a la sostenibilidad

Autor: Carlos Rosety Alonso

Director: Jose Miguel Ordax Cassa

Madrid

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Las nuevas tecnologías aplicadas a la sostenibilidad

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2018/19 es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: Carlos Rosety Alonso Fecha: ……/ ……/ ……

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Jose Miguel Ordax Cassa Fecha: ……/ ……/ ……

AUTORIZACIÓN PARA LA DIGITALIZACIÓN, DEPÓSITO Y DIVULGACIÓN EN RED DE PROYECTOS FIN DE GRADO, FIN DE MÁSTER, TESINAS O MEMORIAS DE BACHILLERATO

1º. Declaración de la autoría y acreditación de la misma.

El autor D. Carlos María Rosety Alonso

DECLARA ser el titular de los derechos de propiedad intelectual de la obra: Las nuevas tecnologías aplicadas a la sostenibilidad, que ésta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual.

2º. Objeto y fines de la cesión.

Con el fin de dar la máxima difusión a la obra citada a través del Repositorio institucional de la Universidad, el autor **CEDE** a la Universidad Pontificia Comillas, de forma gratuita y no exclusiva, por el máximo plazo legal y con ámbito universal, los derechos de digitalización, de archivo, de reproducción, de distribución y de comunicación pública, incluido el derecho de puesta a disposición electrónica, tal y como se describen en la Ley de Propiedad Intelectual. El derecho de transformación se cede a los únicos efectos de lo dispuesto en la letra a) del apartado siguiente.

3º. Condiciones de la cesión y acceso

Sin perjuicio de la titularidad de la obra, que sigue correspondiendo a su autor, la cesión de derechos contemplada en esta licencia habilita para:

a) Transformarla con el fin de adaptarla a cualquier tecnología que permita incorporarla a internet y hacerla accesible; incorporar metadatos para realizar el registro de la obra e incorporar "marcas de agua" o cualquier otro sistema de seguridad o de protección.
b) Reproducirla en un soporte digital para su incorporación a una base de datos electrónica, incluyendo el derecho de reproducir y almacenar la obra en servidores, a los efectos de garantizar su seguridad, conservación y preservar el formato.
c) Comunicarla, por defecto, a través de un archivo institucional abierto, accesible de modo libre y gratuito a través de internet.
d) Cualquier otra forma de acceso (restringido, embargado, cerrado) deberá solicitarse expresamente y obedecer a causas justificadas.
e) Asignar por defecto a estos trabajos una licencia Creative Commons.
f) Asignar por defecto a estos trabajos un HANDLE (URL *persistente).*

4º. Derechos del autor.

El autor, en tanto que titular de una obra tiene derecho a:

a) Que la Universidad identifique claramente su nombre como autor de la misma
b) Comunicar y dar publicidad a la obra en la versión que ceda y en otras posteriores a través de cualquier medio.
c) Solicitar la retirada de la obra del repositorio por causa justificada.
d) Recibir notificación fehaciente de cualquier reclamación que puedan formular terceras personas en relación con la obra y, en particular, de reclamaciones relativas a los derechos de propiedad intelectual sobre ella.

5º. Deberes del autor.

El autor se compromete a:

a) Garantizar que el compromiso que adquiere mediante el presente escrito no infringe ningún derecho de terceros, ya sean de propiedad industrial, intelectual o cualquier otro.
b) Garantizar que el contenido de las obras no atenta contra los derechos al honor, a la intimidad y a la imagen de terceros.
c) Asumir toda reclamación o responsabilidad, incluyendo las indemnizaciones por daños, que pudieran ejercitarse contra la Universidad por terceros que vieran infringidos sus derechos e intereses a causa de la cesión.
d) Asumir la responsabilidad en el caso de que las instituciones fueran condenadas por infracción

de derechos derivada de las obras objeto de la cesión.

6º. Fines y funcionamiento del Repositorio Institucional.

La obra se pondrá a disposición de los usuarios para que hagan de ella un uso justo y respetuoso con los derechos del autor, según lo permitido por la legislación aplicable, y con fines de estudio, investigación, o cualquier otro fin lícito. Con dicha finalidad, la Universidad asume los siguientes deberes y se reserva las siguientes facultades:

- La Universidad informará a los usuarios del archivo sobre los usos permitidos, y no garantiza ni asume responsabilidad alguna por otras formas en que los usuarios hagan un uso posterior de las obras no conforme con la legislación vigente. El uso posterior, más allá de la copia privada, requerirá que se cite la fuente y se reconozca la autoría, que no se obtenga beneficio comercial, y que no se realicen obras derivadas.
- La Universidad no revisará el contenido de las obras, que en todo caso permanecerá bajo la responsabilidad exclusive del autor y no estará obligada a ejercitar acciones legales en nombre del autor en el supuesto de infracciones a derechos de propiedad intelectual derivados del depósito y archivo de las obras. El autor renuncia a cualquier reclamación frente a la Universidad por las formas no ajustadas a la legislación vigente en que los usuarios hagan uso de las obras.
- La Universidad adoptará las medidas necesarias para la preservación de la obra en un futuro.
- La Universidad se reserva la facultad de retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

Madrid, a …….. de ……... de …….

ACEPTA

Fdo……

Motivos para solicitar el acceso restringido, cerrado o embargado del trabajo en el Repositorio Institucional:

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

Las nuevas tecnologías aplicadas a la sostenibilidad

Autor: Carlos Rosety Alonso

Director: Jose Miguel Ordax Cassa

Madrid

Agradecimientos

Al equipo de Liight, en especial a mi socio y CEO, Santiago Jiménez y a mi padre por introducirme a la programación allá en el año 2008 y enseñarme prácticamente todo lo que se.

LAS NUEVAS TECNOLOGÍAS APLICADAS A LA SOSTENIBILIDAD

Autor: Rosety Alonso, Carlos.
Director: Ordax Cassa, Jose Miguel.
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

Desarrollo de una aplicación para el sistema operativo Android que motiva a las personas a ser más sostenibles en su día a día a través de mecánicas de gamificación y sistemas de *smartcity*. El desarrollo hasta la fecha es satisfactorio, logrando más de 8000 descargas en Google Play, métricas de retención propias de las 500 mejores aplicaciones de Google Play, y logrando ser la 9ª aplicación social más popular de Google Play.

Palabras clave: Sostenibilidad, Smart City, Inteligencia Artificial, IoT, Cloud Computing

1. Introducción

Madrid es una de las ciudades con peor calidad del aire a nivel europeo. Esto viene como consecuencia del bajo índice de hábitos sostenibles en nuestra población, traídos por una falta de concienciación o motivación. Por ejemplo, en cuanto a movilidad, el abuso del vehículo privado trae consigo problemas de atascos y aparcamiento, así como graves problemas de contaminación que perjudican nuestra salud con estrés y problemas respiratorios, además de tener consecuencias negativas para el medio ambiente.

2. Definición del Proyecto

Mediante tecnología IoT, Cloud e Inteligencia Artificial, se pretende motivar a los ciudadanos a realizar acciones y adquirir hábitos sostenibles como el uso del transporte público, la bicicleta o el reciclaje. Al realizar estas acciones los usuarios son recompensados con Liights, una unidad de medida energética entendida como ahorro de CO2eq. que les motiva a competir por ser más sostenibles en su día a día. Los Liights además son una moneda virtual con la que se pueden obtener premios directos y descuentos en servicios como actividades de ocio, restaurantes, moda o diversos productos. De esta manera, se consigue aumentar la concienciación entre los ciudadanos y reducir la huella de carbono en nuestras ciudades.

3. Descripción del modelo/sistema/herramienta

La aplicación consiste en 5 pantallas principales (y más pantallas secundarias) en las que residen las funcionalidades básicas de la aplicación. Estas pantallas son:

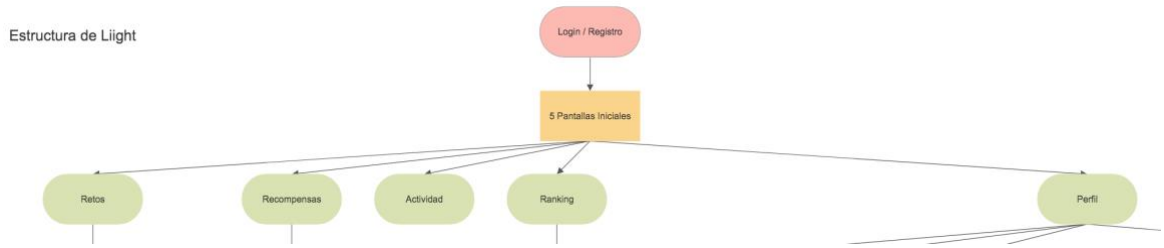

Figura 1- Estructura de pantallas principales de Liight

- **Realizar actividad:** En esta pantalla se puede consultar información útil de movilidad sostenible como el tiempo de espera al autobús. También es desde esta pantalla desde donde se comienza la actividad sostenible elegida (de entre Andar, Ir en bici, el uso de transporte público y reciclar).

- **Recompensas:** Esta pantalla muestra las diferentes recompensas disponibles para los usuarios en cada momento, así como información básica de cada una de ellas tipo foto, descripción, coste en Liights, tipo de recompensa (descuento directo o boleto para sorteo), descuento aplicado o tiempo para el sorteo, etc. También incluye el proceso de canjear los liights por la recompensa.

- **Retos:** Son una parte básica de la aplicación. Los retos son concursos temporales (normalmente duran 2 semanas) a los que los usuarios se unen y empiezan a competir. Tiene su propio ranking y uno o más premios (normalmente suele ser uno para el ganador y otro para sortear entre los 10 mejores). Incluye también mostrar información básica (foto, descripción, etc.), funcionalidad de unirse al reto y segmentación de retos (para poder realizar retos solo para empresas o para un grupo de amigos).

- **Perfil:** Desde esta pantalla se puede consultar información básica como la foto de perfil, el nombre de usuario, liights, experiencia, nivel, CO_2 reducido (por no haber realizado esas actividades en coche), actividades realizadas (cada una con su información básica) y estadísticas de uso.

- **Ranking:** Esta pantalla muestra un ranking general de todos los usuarios y los puntos y la posición del usuario que consulta el ranking.

4. Resultados

A día de hoy, la aplicación tiene más de 8000 descargas en Android, más de 900 usuarios recurrentes semanales y se estima que ha ayudado a reducir la cantidad de CO_2 en la atmósfera en 20 toneladas. Además, como se puede observar en el gráfico de abajo, las métricas de retención están entre las de las 5000 mejores aplicaciones de Google Play, y el pasado mes de Junio, Liight fue la 9ª aplicación social más popular de Google Play.

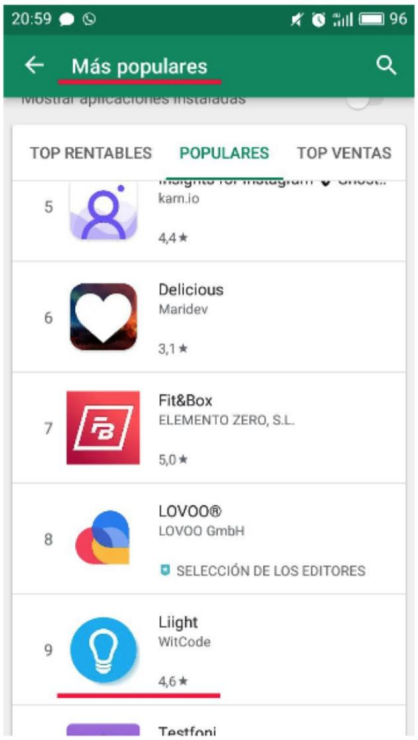

Figura 2 - Liight es la 9ª aplicación social más popular

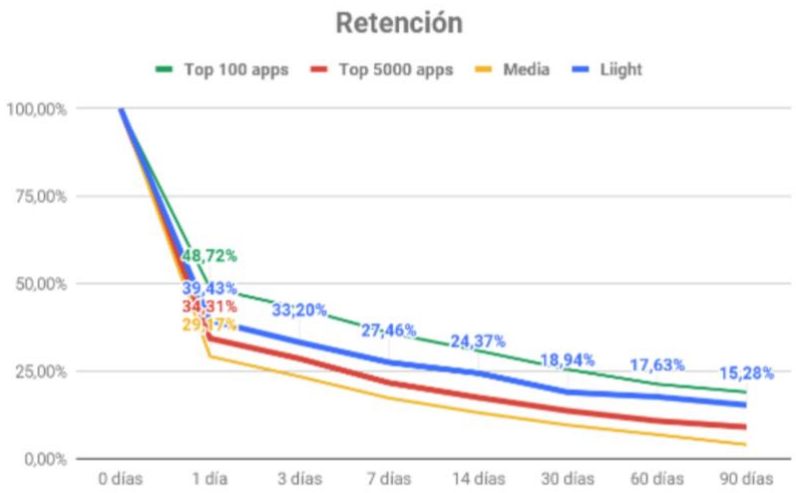

Figura 3 - Gráfica de retención

5. Conclusiones

El resultado del Proyecto es satisfactorio, y se espera que el proyecto siga creciendo de manera exponencial a partir de septiembre de 2019, mes en el que se espera recibir 100.000€ de financiación.

La aplicación tiene mucho recorrido de mejoras, pero a día de hoy, se puede decir que el mercado la ha aceptado y parece que tiene un largo y exitoso futuro.

NEW TECHNOLOGIES APPLIED TO SUSTAINABILITY

Author: Rosety Alonso, Carlos.
Supervisor: Ordax Cassa, Jose Miguel.
Collaborating Entity: ICAI – Universidad Pontificia Comillas

ABSTRACT

Development of an Android app that motivates people to be more sustainable in their every day life through gamification and smart city systems. Development up to date has been successful, having achieved more than 8000 downloads at Google Play and having been Google Play's 9th most popular social app.

Keywords: Sustainability, Smart City, Artificial Intelligence, IoT, Cloud Computing

1. Introduction

Madrid is one of the worst cities in terms of air quality at a European level. This is because of the low levels of sustainable habits within our population, brought by a lack of awareness and motivation. For example, in terms of mobility, the abuse of the private vehicle can bring with it jams and parking problems and also health problems, and negative consequences for the environment.

2. Project definition

Through IoT, Cloud and Artificial Intelligence, the idea is to motivate citizens to acquire sustainable habits like the use of public transport, biking or recycling. By performing these actions, users are rewarded with Liights, an energy measurement unit understood as saving of CO2, that incentivates competition to be more sustainable in every day life. Liights are also a virtual coin that can be exchanged by discounts in direct rewards and discounts in products and services such as restaurants, clothes, cinemas and much more. This way, awareness can be raised among citizens and the carbon footprint can be reduced.

3. Description of the platform

The app consists of 65main screens (and more secondary screens) that host multiple functionalities. These main screens are:

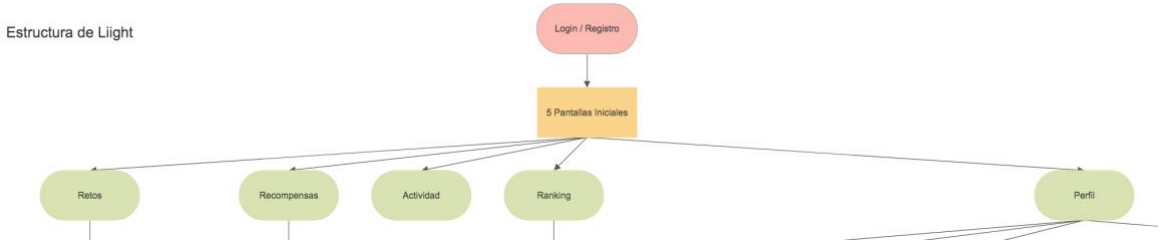

Figura 4- Estructura de pantallas principales de Liight

- **Perform activity:** In this screen, useful information such as waiting times can be consulted. It is also through this window, how sustainable activities can be started.

- **Rewards:** This screen shows the different rewards available to the users in Liight, and also basic information of each of them such as description, cost in Liights, type of reward, discount applied and time for the raffle. It also includes the process of exchanging lights for the reward.

- **Challenges:** They are a basic part of the application. Challenges are temporary competitions (they normally last 2 weeks) that can be joined by users to start to compete. They have their own ranking and one or more rewards. It also includes showing basic information (such as photo or description), join the challenge functionality and challenge segmentation (to make it possible for companies or groups of friends to compete among themselves).

- **Profile:** From this screen, basic information can be consulted such as profile picture, username, liights, experience, level, CO2 reduced, performed activities and usage statistics.

4. Results

Up to date, the app has more tan 8000 downloads in Android, more than 900 recurrent weekly users and it is estimated that 20 tons of CO2 have been reduced because of Liight. Also, engagement metrics show that Liight is among the best 5000 apps in Google Play, and has also been announced to be the 9th most popular social app.

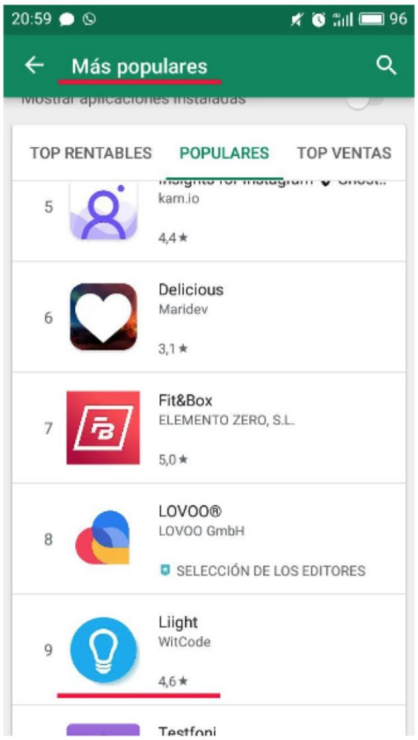

Figura 5 - Liight is the 9th most popular social app

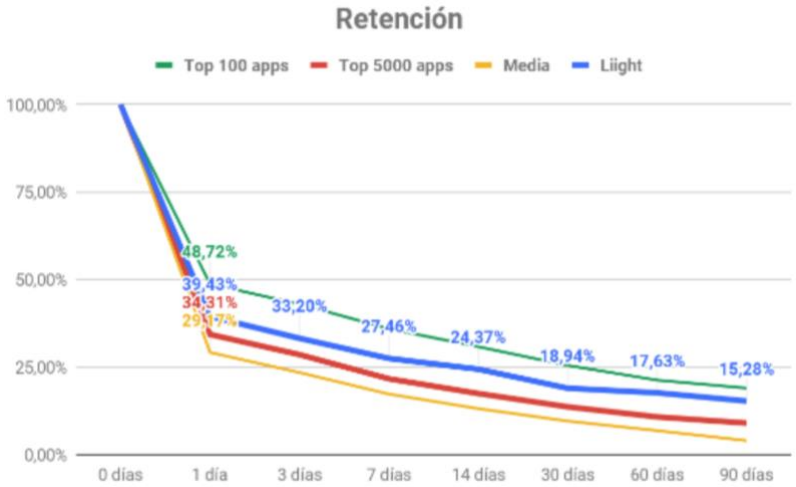

Figura 6 – Retention graph

5. Conclusions

The result of the project is satisfactory, and it is expected that the project keeps growing exponentially from September of 2019, when 100.000€ will be financed.

Índice de la memoria

Índice de figuras

Índice de tablas

Capítulo 1. INTRODUCCIÓN

Madrid es una de las ciudades con peor calidad del aire a nivel europeo. Esto viene como consecuencia del bajo índice de hábitos sostenibles en nuestra población, traídos por una falta de concienciación o motivación. Por ejemplo, en cuanto a movilidad el abuso del vehículo privado trae consigo problemas de atascos y aparcamiento, así como graves problemas de polución que perjudican nuestra salud con estrés y problemas respiratorios, además de tener consecuencias negativas para el medio ambiente.

Algunos datos:

- Amenazas de +380M de euros de multa por la UE a la ciudad de Madrid (Barcelona tiene abierto un proceso similar) por superar niveles de contaminación.
- Episodios recurrentes en Madrid donde se superan las 200µg/m3 de NO2 siendo enormemente perjudicial para la salud e ilegal a nivel europeo.
- Consecuente activación del protocolo anticontaminación ante estos escenarios con el despliegue mediático y descontento ciudadano
- El 51% de los desplazamientos en Madrid son en vehículos privados con una ocupación media de 1,2/5 plazas siendo la principal causa del problema.
- 34,815 Millones de toneladas de CO2 generadas al año en Madrid.

Aprobación del Plan A en el Ayuntamiento con 30 medidas para reducir la contaminación y los gases de efecto invernadero. Normativa de apoyo e implementación de iniciativas encuadradas en movilidad sostenible, regeneración urbana y sensibilización-comunicación. Presupuesto destinado de 543,9 millones de euros objetivos 2020-30. (Aprobado Marzo de 2017).

El ayuntamiento trata de frenar el aumento en los niveles de contaminación con diferentes medidas, entre ellas:

- Dejar entrar sólo a coches de matrícula par/impar
- Instaurar un sistema de pegatinas que identifican a los coches según su nivel de contaminación
- Cerrar calles cuando se alcanzan niveles altos de contaminación
- Instaurar Madrid Central (Prohibir la entrada a todos los coches que no tengan un nivel extremadamente bajo de contaminación)

Estas medidas no están siendo muy populares y muchas investigaciones comentan que estas medidas no serán suficientes para permitir un nivel razonable de salud en la ciudad y además evitar la multa con la que amenaza la Unión Europea.

Fuera del ámbito público, hay pocas sino ninguna entidad que esté implementando medidas para mejorar el estado de la ciudad de Madrid en cuanto a contaminación (o cualquier otra ciudad española).

Por otra parte, el campo de la tecnología está entrando en la inmensa mayoría de los mercados provocando cambios disruptivos y haciendo de la innovación una necesidad en lugar de una oportunidad, sin embargo, en el sector de la sostenibilidad no se ven demasiadas iniciativas acompañadas de soluciones tecnológicas innovadoras.

Es en este contexto donde nace Liight, una aplicación multiplataforma que incentiva a las personas a ser más sostenibles en su día a día a través de la gamificación y sistemas de smart city. Mediante tecnología IoT, Cloud e Inteligencia Artificial, se plantea motivar a los ciudadanos a realizar acciones y adquirir hábitos sostenibles como el uso del transporte público, la bicicleta o el reciclaje, consiguiendo ciudadanos más ecológicos y responsables

con el medio ambiente. La idea es ayudar a los ciudadanos a reducir su huella de carbono a través de retos incentivando hábitos ecológicos como el uso del transporte público, la bicicleta o el reciclaje. Al realizar estas acciones los usuarios son recompensados con Liights, una unidad de medida energética entendida como ahorro de CO2eq. que les motiva a competir por ser más sostenibles en su día a día. Los Liights además son una moneda virtual con la que se pueden obtener premios directos y descuentos en servicios como actividades de ocio, restaurantes, moda o diversos productos.

Figura 7 - Logo de Liight

Capítulo 2. ESTADO DE LA CUESTIÓN

En esta sección se procede a explicar los conceptos relacionados con la tecnología que se usa en Liight, dedicándole más extensión a los apartados más relevantes.

2.1 TIPOS DE APLICACIONES

Existen diferentes tipos de aplicaciones móviles en la actualidad, cada una con sus ventajas y sus desventajas. Además, muchas empresas luchan por lanzar nuevas plataformas que se adueñen del mercado, cuyos ingresos son mayores al 10% (htt) del PIB español.

La mayor parte de las aplicaciones son o nativas o web apps, pero en el último año hay otros tipos de aplicaciones que están cobrando cada vez más fuerza. A continuación se explicará en detalle los diferentes tipos de aplicaciones que existen.

Figura 8 - App nativa vs web app

2.1.1 WEBAPPS

Las WebApps son aplicaciones 'escritas' en lenguaje web (HTML, CSS y JS) y se ejecutan a través de un motor web, como el que tienen Chrome y Safari. Existen diferentes tipos:

a) **WebApp tradicional:** Las web apps tradicionales son realmente páginas web *responsive* visibles desde un navegador tipo Chrome o Safari. Si son muy complejas y dinámicas, suelen escribirse en plataformas que se compilan sobre JavaScript, como React o AngularJS. Estas apps no pueden utilizarse offline salvo que el navegador las guarde en cache o que se descargue la web manualmente. Son relativamente sencillas de desarrollar y tienen la ventaja de que se desarrollan una vez y es compatible con todas las plataformas que tengan un navegador, únicamente habría que asegurar que los diferentes navegadoras soportan todas las funcionalidades de la app. Si se necesitara acceder al hardware del dispositivo, estas apps tendrían que pedirle permiso al navegador que a su vez le pediría permiso al usuario, y aún así seguirían limitadas en cuanto al acceso a las capacidades totales del hardware.

b) **Apps híbridas o WebApps nativas:** Este tipo de aplicación es también una página web, pero va embebida en una *WebView* de una aplicación nativa, y por tanto, es empaquetada como *.apk* o *.app*. Al contrario que las WebApps tradicionales, estas aplicaciones pueden funcional offline puesto que el código web se encuentra en el propio paquete de la aplicación y la *URL* que renderiza la *WebView*, contiene una dirección local al propio paquete. En estas apps, el código web sólo tendría que ser escrito una vez, pero la parte nativa tendría que escribirse en cada plataforma, aunque realmente es muy sencilla. Si se quisiera acceder al hardware, tendrían que escribirse APIs que enlazaran con código nativo de cada plataforma y así crear una especie de capa abstracta del hardware.

Este tipo de aplicación (htt1) se ha hecho muy popular en los últimos años y en consecuencia, han surgido numerosas plataformas que tratan de hacer más sencillo el proceso de desarrollo. *Cordova* es un ejemplo de plataforma (con licencia *Apache*) que desarrolla plugins para otras plataformas, a través de los cuales se puede acceder a diferentes capacidades del hardware sin tener que escribir código nativo. Otras plataformas como *Ionic, React Native, PhoneGap, Framework 7* o *NativeScript*, ofrecen herramientas para desarrollar apps híbridas sin tener que tener conocimiento de lenguajes nativos.

Figura 9 - Logo de Ionic

Las WebApps en general se caracterizan por tener un bajo rendimiento y limitaciones al acceder a las capacidades del hardware. Sin embargo, tienen ventajas como el aprovecharse de la comunidad web o el poder escribir una vez el código y desplegar en diferentes plataformas.

2.1.2 APPS NATIVAS

Las apps nativas están 'escritas' en el lenguaje propio de la plataforma. Por esta razón, estas aplicaciones tienen un rendimiento mayor al de las WebApps y pueden acceder a todas las capacidades del hardware que el sistema operativo le permita. En el caso de Android, el *kernel* de Linux es el contacto directo con el hardware y por tanto, con permisos de superusuario se podría llegar a controlar el hardware sin límites.

Las dos grandes plataformas en las que se suelen desarrollar las plataformas nativas son Android e iOS, que entre las 2 hacen el 99,8% de la cuota de mercado.

En el caso de Android, el lenguaje nativo original es Java, que se compila en bytecode (*.class*) y luego es nuevamente compilado mediante dx en un archivo *.dex*, equivalente a un *.jar* de java y empaquetado mediante aapt en un *.apk* (que realmente es un *.zip* cambiado de nombre). La razón por la que el .class es compilado a .dex es porque Android no contiene una máquina virtual de Java (JVM) sino que incluye una máquina virtual Dalvik (realmente por razones de licencias de Oracle). En 2017, Google incluyó a Kotlin (htt2) como otro posible lenguaje para desarrollar aplicaciones nativas. Este lenguaje no tipificado está diseñado para poder compilarse directamente en bytecode de Java (.class), luego no cambia el rendimiento de la aplicación, y muchos desarrolladores agradecen la simplicidad y rapidez de este lenguaje creado por JetBrains. En Mayo de 2019, Google anunció que Kotlin pasaba a ser el lenguaje preferido para desarrollar aplicaciones nativas en Android.

En el caso de Apple, el lenguaje nativo original es Objective-C pero fue reemplazado por Swift en 2015.

Figura 10 - Comparación entre tipos de aplicaciones

2.1.3 OTROS TIPOS DE APLICACIONES

Aunque las apps nativas y las WebApps son las más usadas por desarrolladores, cada vez se empiezan a ver más otros tipos de aplicaciones que luchan por llevarse el mercado por delante.

Aunque hay muchos otros tipos de aplicaciones, se explicarán únicamente los dos más importantes, ambos desarrollados por Google. Es interesante preguntarse por qué Google se haría competencia a sí mismo, y la respuesta se haya en el propio ADN de la compañía, y es que, desde el momento en el que los trabajadores se incorporan a la empresa, los Googlers son animados a pensar en ideas que puedan echar al propio Google del mercado, y estas plataformas existen a raíz de esto.

- **Progressive Web Apps (PWA):** Estas aplicaciones realmente son WebApps alteradas por Google para beneficiarse de características de las apps nativas. Se encuentran de la misma manera que se encontraría una WebApp tradicional, a través de un buscador. Sin embargo, la apariencia de estas apps no es la de una página web sino que se parece mucho más a la de una aplicación nativa. Además tienen la opción de crear un icono en la pantalla de escritorio que sirva como acceso directo a la app, la cual también tiene la opción de descargarse y usarse en modo offline. Google apuesta fuertemente por las PWA como substitutas de cualquier WebApp y de muchas apps nativas o híbridas, ya que se ha observado que existe un sentimiento de rechazo a descargar aplicaciones de las App Stores, mientras que el uso de WebApps es mucho más efectivo para determinadas funcionalidades.

- **Flutter:** Flutter es un SDK que permite desarrollar aplicaciones de una manera distinta. Funciona sobre una máquina virtual distinta a la Dalvik llamada Shell. Esta máquina virtual vuelca el procesado en mayor medida sobre la GPU que sobre la CPU, lo que hace las apps más como un juego que como una aplicación nativa de Java/Kotlin. Esto hace que la interfaz de las aplicaciones sea mucho más rápida y dinámica, obteniendo un rendimiento incluso mejor que las aplicaciones nativas tradicionales.

  El lenguaje que utiliza Flutter es Dart, diseñado para escribir lógica y diseñar la interfaz en el mismo documento. Flutter todavía tiene sus limitaciones, sobretodo en

cuanto al control de las capacidades del hardware, ya que necesita de librerías desarrolladas *ad hoc* para la plataforma. Hasta que existan librerían que consigan acceder sin limitaciones al hardware, Flutter ofrece la opción de una programación mixta entre la máquina Dalvik y la Shell, es decir, que se podrían programar partes específicas en Java/Kotlin (en el caso de Android) o Swift (en el caso de iOS).

Cada vez la comunidad de esta plataforma es mayor y las actualizaciones llegan con más frecuencia y más mejoras, lo que hace pensar que Google está apostando fuertemente por Flutter como una potente plataforma capaz de reemplazar muchas otras maneras de desarrollar aplicaciones.

Figura 11 - Arquitectura de Flutter

2.2 ELECCIÓN DE APLICACIÓN

Llega el momento de decidir que tipo de aplicación es la más indicada para usar en este trabajo. Para ello, en primer lugar se enumeran las necesidades y capacidades de Liight:

- Necesidad de un alto rendimiento a la hora de manejar el GPS y los sensores del acelerómetro y giroscopio, así como de gestionar servicios en segundo plano que puedan detectar automáticamente patrones en los sensores y así validar actividades sostenibles.

- Necesidad de concurrencia, es decir, importa más la cantidad de veces que el usuario entre en la aplicación que la calidad de cada visita.

- Baja necesidad de optar por una opción 'barata' de desarrollo puesto que el equipo es interno y no 'cuesta dinero'.

- El proyecto sigue el modelo lean-startup por lo que hay una necesidad de adaptación al continuo cambio y una gran capacidad de detección de errores. Además, el objetivo del proyecto (como de cualquier startup) es validar el modelo de negocio con los recursos mínimos posibles (una vez validado entrarían rondas de inversión de crecimiento que cambiarían las reglas del juego), por lo que validar para una plataforma sería suficiente, especialmente si esa plataforma es Android, que tiene una cuota de mercado del 75% en España.

Teniendo en cuenta estas necesidades y capacidades, la opción más indicada de aplicación es la de desarrollar una app nativa de Android en Java. Este tipo de aplicación tiene las ventajas de tener una alta capacidad de control del hardware y de recursos en segundo plano, además de poseer la robustez de Java de cara a tener una aplicación de un alto nivel de complejidad y muchas probabilidades de errores por su continuo y necesario cambio. Sus desventajas son que una aplicación únicamente de Android no abarca el mercado completo si no que estaría disponible para el 75% de los dispositivos móviles en España, pero como se menciona anteriormente, al ser el objetivo validar el modelo de negocio, es posible hacerlo desarrollando en una sola plataforma.

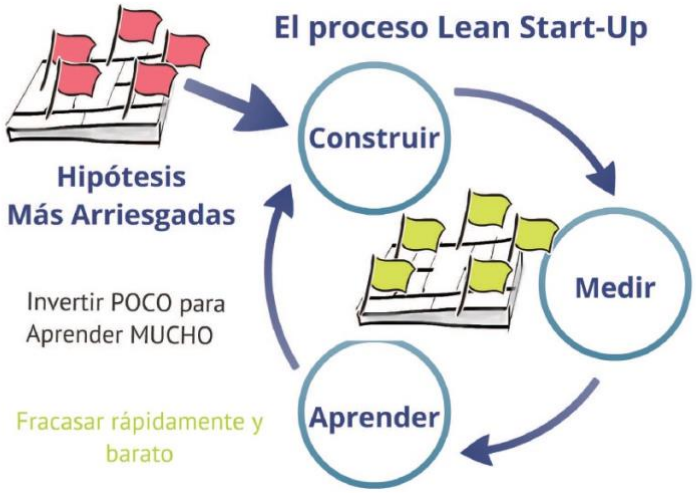

Figura 12 - Método Lean Startup

2.3 CLOUD

Debido a que Liight es un tipo de aplicación con alta necesidad de un servidor centralizado, se procede a explicar el estado del arte de esta tecnología y así analizar las mejores opciones para el trabajo.

2.3.1 HISTORIA

La historia y evolución del Cloud, como la mayoría de tecnologías, está atada a la de Internet y las aplicaciones informáticas, según como los usuarios de Internet se iban adaptando e integrando los hábitos digitales en su día a día.

Las primeras páginas web, estaban alojadas en servidores propios, los cuales tenían que ser mantenidos (almacenamiento, electricidad, temperatura, conexión) a un alto precio. Esto hacía que la barrera de entrada al mercado de Internet fuera muy alta. Además, si el servicio de Internet empezaba a crecer y cada vez exigía más recursos del servidor, desde el momento en el que se pedía un servidor nuevo hasta que este llegaba, se configuraba y estaba

operativo, podía transcurrir más de un mes, tiempo suficiente para perder los clientes o usuarios que se pudieran tener, debido a problemas del servidor. Todo esto obligaba al sector a evolucionar y hacer el mercado de Internet más seguro para empresarios.

Es así como surgen los primeros *IaaS* (Infraestructura como servicio), un modelo de servicio que permitía poseer un servidor sin necesidad de mantenerlo físicamente.

Primero surgieron modelos de alquiler de servidores en los que el servidor estaba en un lugar del mundo y el cliente tenía acceso remoto a él, tal y como si fuera suyo. De esta manera, se ahorraba los costes de mantenimiento y el tiempo de contratación era muchísimo más rápido que en la etapa anterior. Las empresas que poseían los servidores podían reducir los costes de mantenimiento a través de economía de escala.

Poco después surgió la idea de alquiler de máquinas virtuales que permitían adquirir el permiso de uso de parte de un servidor, pero no de la máquina física en su totalidad. Para el cliente, era una manera de poder alquilar pocos recursos cuando empezaba con su idea de negocio, mientras que para la empresa que poseía los servidores era una manera de aumentar la demanda.

Esta idea se fue haciendo cada vez más abstracta hasta el punto en el que el cliente no tenía que elegir el sistema operativo que quería que tuviera el servidor, no tenía que preocuparse por la seguridad y configuración de red e incluso no tenía por qué preocuparse de la RAM o procesador que usaba el servidor.

Así surgen las *Paas* (Plataforma como servicio), un modelo de servicio que permitía poder crear páginas web dinámicas y complejas sin prácticamente saber nada de servidores.

Un servicio Cloud que ha cobrado mucha popularidad en los últimos años es el de *Serverless*. Este modelo permite escribir código de un lenguaje determinado, y directamente ejecutarlo en la nube y conectarlo a un dominio, una base de datos o lo que se prefiera. La idea es que

cada vez, los desarrolladores necesitan saber menos de la parte física y se pueden centrar más en la lógica de sus programas.

Sin embargo, cuanto más abstracto el modelo, más limitaciones se tienen. Por ejemplo, un problema que tienen los *Serverless* o los *Paas*, es que se ejecuta código directamente sobre el Cloud, en un *Runtime* que no se controla por el desarrollador. Estos *Runtimes* tienen su propia configuración, y muchas veces es necesario conocerla y poder cambiarla para lograr lo que se quiere.

Este problema es el que resuelven los *Contenedores*. El modelo de contenedor está a medias entre un Iaas y un Paas. Es decir, no llega a tener una máquina virtual con un sistema operativo dentro, pero tampoco se despliega únicamente el código. Realmente, un Contenedor es como la capa directamente superior a la de un Sistema Operativo. Contiene el código que se quiere ejecutar, y también los runtimes en los que se ejecutan con sus propias configuraciones. El modelo de Contenedor ha cobrado mucha fuerza en los últimos años y cada vez hay más empresas que confían en él. La empresa que encabeza la tecnología de Contenedores es Docker.

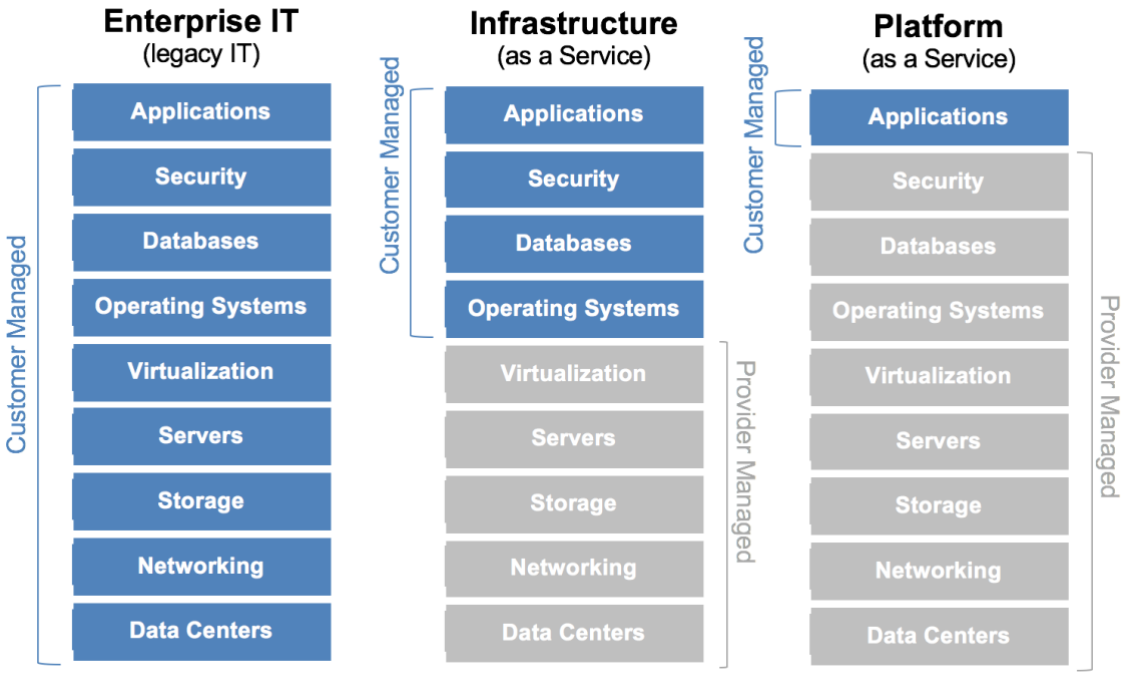

Figura 13 - Evolución del Cloud

2.3.2 AMAZON WEB SERVICES (AWS)

El Cloud de Amazon fue la primera empresa en ofrecer servicios *IaaS* de calidad y sigue siendo un referente a día de hoy. Se explorará esta opción de alojamiento debido a que es la elegida para servir como back-end en el proyecto.

Desde el principio, Amazon ha ido aumentando sus servicios para adaptarse e innovar ofreciendo servicios que podrían interesarle a los desarrolladores. De todos ellos, en Liight se ha decidido usar los siguientes:

Figura 14 - Logo de Amazon Web Services

2.3.2.1 Elastic Beanstalk

Este servicio Cloud Computing es de tipo PaaS, pero con algunas particularidades. Amazon permite crear máquinas virtuales (llamadas EC2 del modelo IaaS) y configurarlas como se quiera, sin embargo, esto requiere conocimiento técnicos que no cualquier desarrollador tiene, por lo que Elastic Beanstalk permite automatizar el proceso de alquilar y configurar una máquina EC2, además de ofrecer un programa de escalabilidad que permite ajustar los recursos dedicados según la demanda y así, minimizar los costes.

Elastic Beanstalk funciona por *environments* (entornos). Estos entornos realmente son máquinas EC2 preconfiguradas y que contienen runtimes (Docker, PHP, Tomcat, Python, etc.). Para desplegar el código se puede hacer desde el propio repositorio en la nube de Amazon, CodeCommit (que permite despliegues incrementales) o directamente con archivos de proyecto completo (.zip, .war, .jar, etc.)

2.3.2.2 Relational Database Service (RDS)

https://aws.amazon.com/es/rds/

RDS es un servicio que permite desplegar instancias de bases de datos relacionales en máquinas virtuales. Además, permite configurarlas sin limitaciones, gestionar backups y

asegurar disponibilidad y entrega. Los motores de bases de datos disponibles son: *Amazon Aurora, PostgreSQL, mySQL, MariaDB, Oracle* y *Microsoft SQL Server*.

2.3.2.3 Lambda

https://aws.amazon.com/es/lambda/

Figura 15 - Logo de Lambda

Lambda es la opción Serverless de Amazon. Permite escribir código y desplegarlo sin necesidad si quiera de escoger un Runtime. Se puede conectar a múltiples triggers como llamadas HTTP, Cron o eventos propios de la plataforma de AWS y puede utilizar todos los recursos disponibles de AWS.

Esta opción es particularmente útil para realizar trabajos de servidor no iniciados por llamadas HTTP, como copias de seguridad, mantenimiento o analíticas.

2.3.2.4 Otros servicios de AWS

Otros servicios de Amazon son:

a) **CloudCommit:** Repositorio en la nube de control de versiones. También tiene la opción de desplegar y compilar código.

b) **S3:** Servicio básico de almacenamiento que permite alojar páginas web.

c) IAM: Servicio de gestión de usuarios y permisos.

d) Route 53: Servicio de gestión de dominios y DNS.

e) CloudWatch: Servicio de monitorización y administración de aplicaciones.

2.4 COMPETENCIA

A día de hoy, existe un aumento de concienciación en respecto a la sostenibilidad y cada vez las empresas dedican más presupuesto a sus departamentos de RSC (Responsabilidad Social Corporativa) ya que ven que cada vez los clientes o consumidores finales valoran más el impacto social de las empresas.

En cuanto a movilidad, los coches eléctricos cada vez cobran más importancia y la prohibición del diésel en España a partir de 2040 es una clara demostración de la tendencia hacia una movilidad más sostenible. Tesla, Uber y Cabify, empresas que lideran el frente innovador con respecto a la movilidad, tienen como misión conseguir un sector mucho más respetuoso con el medio ambiente.

Aunque es cierto que la concienciación respecto al medio ambiente está creciendo de la mano de los más jóvenes (especialmente los millenials), es la propia conciencia moral la que hace actuar a los ciudadanos de una manera u otra, es decir, tener hábitos sostenibles no se recompensa en nuestra sociedad.

Para cambiar esto, hay varias iniciativas de ayuntamientos y gobiernos sobre todo del norte de Europa, que recompensan actividades sostenibles como el reciclaje, dando monedas al entregar botellines de vidrio. Sin embargo, no hay ninguna solución reconocida globalmente que haga en la práctica que los ciudadanos se sientan recompensados por tener hábitos sostenibles.

Es por esto por lo que es el mejor momento para que surja Liight, una aplicación que valida las actividades sostenibles y las recompensa a través de dopamina (subes de nivel, consigues logros), aumento de ego (destacas socialmente por tener hábitos sostenibles) y productos y servicios tangibles (descuentos o premios directos).

En este campo de aplicaciones que recompensan hábitos sostenibles se encuentran pocos competidores, entre ellos:

a) CICLOGREEN: Aplicación para incentivar y premiar el uso de la bicicleta en las ciudades.

Procedentes de Málaga ya ofrecen retos y premios en otras ciudades españolas. Sus puntos "ciclos" se corresponden con los km recorridos en bici. Se puede asociar con otras apps de deporte como strava. Los premios son fundamentalmente participación en sorteos con regalos para comunidad ciclista.

b) GIVEO2: App de tracking automática capaz de conocer nuestra huella en desplazamientos a través de geoposicionamiento. Ofrece diferentes premios como bonos de transporte o tiquets para festivales. Región geográfica: Procedencia Amsterdan (Holanda), Volumen de descarga desconocido. Últimas noticias 2012.

c) AllGreenUp: Este sería nuestro competidor más directo, valida acciones de movilidad sostenible desde runkepper y acciones de reciclado por checking en geoposición de puntos limpios. Surgen en Chile aunque tienen presencia en distintas ciudades de América Latina conociéndose valoraciones en México por ejemplo. Cuenta con entre 10.000-50.000 descargas en Android. Premios en alimentación ecológica y gimnasios.

d) ChangersCO2 Fit: El sistema de monitorización es similar al de GiveO2, usa el sensor GPS en segundo plano registrando cada desplazamiento y su velocidad. La diferencia fundamental se enfoca en su forma de monetización, se enfoca especialmente a empresas gamificando acciones entre empleados.

Capítulo 3. DESCRIPCIÓN DE LAS TECNOLOGÍAS

Para el desarrollo de la aplicación, se han usado múltiples herramientas, las cuales serán tratadas en este capítulo, dándole más extensión a las más importantes.

3.1 ANDROID STUDIO

Android Studio es el entorno más usado con diferencia para la creación de aplicaciones Android. Este IDE es desarrollado y mantenido por una colaboración entre Google y Jetbrains. La base, interfaz y concepto es de Jetbrains, mientras que Google aporta las funcionalidades necesarias para la integración con su sistema operativo.

Figura 16 - Logo de Android Studio

Al igual que los demás IDEs de Jetbrains, Android Studio aporta funcionalidades como sugerencias inteligentes, live code (plantillas de código frecuente), shortcuts útiles, etc.

Este programa permite además monitorizar el CPU, memoria, red, y más partes del rendimiento del dispositivo.

3.2 PHPSTORM E INTELLIJ IDEA

Se ha optado por estos IDEs para escribir los códigos de PHP y Java (a nivel servidor) respectivamente, por ser producto de Jetbrains, y por tanto, mantener numerosas similitudes con otros IDEs usados como Android Studio o DataGrip. Además, tanto PHPStorm como IntelliJ son propuestos por la mayoría de la comunidad de desarrolladores como los mejores IDEs existentes en su lenguaje. La única desventaja sería el precio, pero al ser estudiante, la licencia es gratis.

Figura 17 - Logos de PhpStorm e IntelliJ IDEA

Ambos tienen numerosas ventajas frente a otros IDEs, entre ellas:

- Son integrable con fuentes de datos como bases de datos MySQL, por lo que las queries SQL escritas desde los entornos de PHP y Java para acceder a la base de datos, son inteligentes y permiten detectar errores y ver previews de consultas a la base de datos directamente desde el propio programa.
- Tienen la capacidad de desplegar el código en un servidor tanto local como remoto y ejecutarlo, así ahorrando mucho tiempo en la etapa de desarrollo.
- Tiene funcionalidades que permiten la depuración del código.

- Integración completa con todos los sistemas de control de versiones como Git.
- Poseen las funcionalidades típicas de Jetbrains que agilizan enormemente el desarrollo (sugerencias inteligentes, live code, shortcuts útiles, generadores de código frecuente, etc.).

3.3 DATAGRIP

DataGrip es el programa elegido para la gestión de la base de datos MySQL. Este IDE para bases de datos también es desarrollado por Jetbrains y por tanto posee las funcionalidades típicas de los demás IDEs de esta empresa, como sugerencias inteligentes, live code (plantillas de código frecuente), shortcuts útiles, etc. Además, posee una funcionalidad de visualización de la lógica relacional entre las tablas que permite entender mejor la relación entre estas.

Capítulo 4. DEFINICIÓN DEL TRABAJO

4.1 JUSTIFICACIÓN

Como se ha mencionado anteriormente, creemos que es el mejor momento para desarrollar esta aplicación. Hay poca competencia en un campo que está en pleno auge y que con una masa suficientemente grande de usuarios podría abrir numerosas oportunidades sociales y empresariales.

Además, es una oportunidad para poder desarrollar tecnología innovadora como IoT, Cloud e Inteligencia Artificial, aprender en el proceso y aportar nuestro granito de arena a la sociedad mejorando el medio ambiente.

4.2 OBJETIVOS

El objetivo del proyecto es desarrollar en Android nativo (Java + XML) una aplicación con las mínimas funcionalidades para poder validar que hay usuarios que utilizarían la aplicación en su día a día, es decir, desarrollarla en estado MVP (Mínimo Producto Viable). Esto se puede dividir en 5 diferentes objetivos:

1. **Desarrollar una primera versión de la aplicación en Android.**

Para ello hay que desarrollar las pantallas básicas de la aplicación entre ellas:

- Login

- Registro

- Tutorial

- Realizar actividad
- Noticias
- Ranking
- Retos
- Recompensas
- Ajustes

2. **Desarrollar un Backend en el Cloud que permita realizar las validaciones y servir como base para funcionalidades sociales en la app.**

Este objetivo implicaría:

a) Montar una plataforma en AWS (Amazon Web Services) que contenga 2 máquinas EC2 (máquinas virtuales) para Java y PHP, una base de datos mySQL, una base de datos Redis mediante el producto AWS Elasticache, configurar la subred IP, contratar dominios mediante Route53, configurar SSL, balanceos de carga, autoescalados, etc. Desarrollar un estudio de precios y carga necesaria para decidir que tipo de servicios contratar.

b) Crear configuración y tablas de la base de datos mySQL así como un esquema relacional de las tablas y un plan de optimización mediante índices.

c) Desarrollar una API REST en PHP que se comunique con la base de datos y sirva como interfaz entre el cliente (Android) y el Backend, volcando los datos necesarios que necesita la aplicación para funcionar.

3. **Desarrollar algoritmo que valide actividades de Andar, Bici y Transporte público en Madrid.**

Desarrollar en Java Cloud un algoritmo que a través de los datos que proporcionan los sensores del móvil, pueda verificar que el usuario está realizando las acciones que dice estar realizando. Incluye:

a) Diseñar la lógica de clases de Java para conseguir un sistema robusto y escalable.

b) Conseguir datos de transporte de la comunidad de Madrid y normalizarlos para poder usar en la plataforma.

c) Diseñar un sistema escalable que permita añadir ciudades en el futuro con datos normalizados y que el algoritmo funcione en cualquier sitio del mundo.

d) Diseñar la gestión de los datos de cada actividad mediante Redis (la base de datos de mySQL se vería desbordada con la continua entrada y salida de datos)

e) Diseñar y desarrollar los protocolos de comunicación de alto nivel entre el cliente (Android) y el servidor (Java Tomcat) para que sea el servidor el que valide las actividades sin producir latencias en el cliente y habilitando el modo *off-line*.

f) Diseñar y desarrollar la lógica del propio algoritmo de validación (Conseguir porcentaje de credibilidad de la actividad que dice estar realizando el usuario).

4. Integrar validación de reciclaje mediante geoposición o con un prototipo IoT que detecta cuando un usuario recicla.

Implica desarrollar en Java Cloud y Android la validación de la actividad de reciclaje en sus distintos tipos explicados en orden de menor a mayor nivel de validación:

a) Geoposición: El usuario podría decir que está reciclando y se le verificaría la actividad si está cerca de un contenedor y no ha reciclado en las últimas 10 horas. Implica recoger los datos de todos los contenedores de la comunidad de Madrid.

b) QR estático: La aplicación deberá contar con un lector de QR y reconocer pegatinas que colocamos en diversos contenedores por Madrid, validando las actividades sólo por el hecho de reconocer que las pegatinas son nuestras. 1 actividad cada 10 horas.

c) Dispositivos IoT: La app se debe sincronizar con dispositivos que tenemos desplegados en algunos contenedores. Estos dispositivos están conectados a Internet y disponen de pantallas LED así como sensores de infrarrojos para detectar que se ha introducido algo en el contenedor.

5. Desarrollar detección automática de actividades de andar, bici y cercanías (usando algoritmos de Inteligencia Artificial).

Implica:

a) Sincronizar la aplicación con la API de Google Fit y de Detección Automática para que Liight pueda contar los pasos de los usuarios sin que tengan la aplicación funcionando y tampoco tengan que tener otras aplicaciones instaladas.

b) Desarrollar diagrama de flujo para decidir si empezar o no actividad de cercanías (en segundo plano con un trigger de la API de Inteligencia Artificial de Google que avisa cuando el usuario está en un vehículo)

c) Vincular las actividades detectadas automáticamente con el resto de la plataforma de Liight.

4.3 METODOLOGÍA

Al comienzo del desarrollo de este trabajo, ya había desarrollada una base (desarrollada por mí durante el año anterior) y por tanto, la metodología seguida este año, era la de seguir desarrollando funcionalidades. Estas funcionalidades eran elegidas siguiendo un modelo agile, específicamente *Scrum* junto con *Kanban*. En la figura de abajo se puede observar nuestra herramienta de Kanban, Trello, junto a las funcionalidades por desarrollar, en desarrollo y ya desarrolladas.

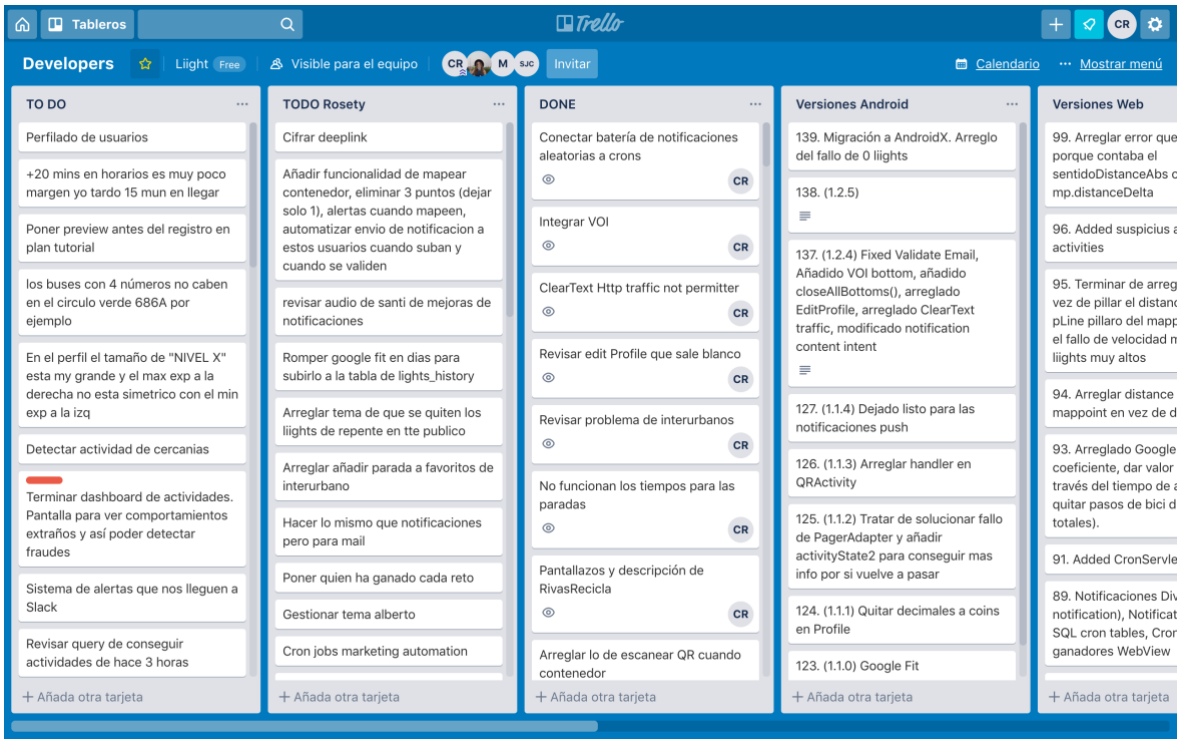

Figura 18 - Ejemplo del tablero de Trello de Liight

Durante el curso, la idea es que una vez a la semana (normalmente los lunes), nos sentaríamos a debatir la prioridad de las nuevas funcionalidades en función de los eventos y el calendario que tuviéramos marcado en ese momento, las opiniones de los usuarios e intuición nuestra.

4.4 PLANIFICACIÓN Y ESTIMACIÓN ECONÓMICA

En febrero de 2019, el calendario propuesto era el siguiente:

25Feb – 3Mar	4Mar-10Mar	11Mar -17Mar	18Mar -24Mar	25Mar – 31Mar	1Abr - 7Abr
Desarrollar herramienta para mandar notificaciones desde el dashboard	Añadir funcionalidad de *¿Has olvidad tu contraseña?*	Implementar Redis en AWS	Implementar Redis en AWS	Integrar APIs de BiciMad en Cloud	Implementar clustering

8Abr – 14Abr	15Abr-21Abr	22Abr – 28Abr	29Abr -5May	6May – 12May	13May – 19May
Implementar clustering	Implementar clustering	Implementar clustering	Detección automática de cercanías	Detección automática de cercanías	Memoria

Tabla 1 - Calendario de trabajo propuesto

Sin embargo, en Liight seguimos el modelo Lean Startup, y teniendo mucha flexibilidad, tendemos a repriorizar las funcionalidades cada poco tiempo, por lo que una planificación a medio-largo plazo, aunque es importante, no se suele cumplir en nuestro caso.

En cuanto a la planificación económica, a continuación se muestra información de nuestro plan financiero, también hecho por mi (trabajado desde una plantilla):

CUENTA DE RESULTADOS	Inicio	Año 1	Año 2	Año 3	Año 4	Año 5
Ingresos		15.000	238.494	1.835.257	5.418.153	9.040.653
Trabajos Realizados para el Inmovilizado		5.600				
Gastos Personal		15.900	97.200	201.600	256.800	256.800
Marketing/Comercialización		22.500	190.000	1.110.000	1.500.000	1.500.000
Otros Gastos de Explotación		7.408	16.224	20.424	24.024	28.824
Amortización		2.000	3.120	3.120	3.120	3.120
Resultado de Explotación		**-27.208**	**-68.050**	**500.113**	**3.634.209**	**7.251.909**
Gastos Financieros		0	0	0	0	0
Resultado Antes de Impuestos		**-27.208**	**-68.050**	**500.113**	**3.634.209**	**7.251.909**
Impuesto Sociedades		-6.802	-17.013	125.028	908.552	1.812.977
Resultado del Ejercicio		**-20.406**	**-51.038**	**375.085**	**2.725.657**	**5.438.932**

Figura 19 - Cuenta de resultados de Liight

CASH FLOW	Inicio	Año 1	Año 2	Año 3	Año 4	Año 5
Cobros		18.150	288.578	2.220.661	6.555.965	10.939.190
Adquisición Inmovilizado		12.100				
Pagos Generales		52.089	346.731	1.569.413	2.100.869	2.106.677
Pago IVA Hacienda		0	0	65.339	627.775	275.760
Impuesto Sociedades					70.597	908.552
Cash Flow Operativo		-58.139	635.309	3.855.414	9.355.206	14.230.179
Cash Flow Acumulado	20.000	-38.139	597.170	4.452.584	13.807.790	28.037.969

Figura 20 - Cashflow de Liight

En resumen, las aportaciones financieras son las siguientes:

- Una aportación inicial de los socios de 20.000€ (realmente el dinero viene de ganar concursos).
- Esperamos cerrar una ronda de inversión (ya en proceso de cierre) de 100.000€ para Septiembre.
- Hemos pedido un préstamo ENISA de 60.000€ que sería concedido en Mayo de 2020.
- Además, el año pasado pedimos 150.000€ al Neotec, una subvención europea a empresas tecnológicas en creación. No nos la concedieron, pero estuvimos realmente

cerca, y según información de expertos, el año que viene finalmente nos la concederán.

Los ingresos que esperamos obtener vienen de dos modelos de negocio diferentes:

a) **Publicidad CPM:** Este modelo de publicidad es el de 'coste por mil impresiones'. Nuestros clientes nos pagarían alrededor de 5€ por cada 1000 veces que se 've' o se interactúa con una recompensa suya. Es nuestro principal modelo de negocio por su escalabilidad.

b) **Corporate:** Consiste en proporcionar a empresas grandes una herramienta para que puedan monitorizar la huella de carbono de sus empleados, así como motivarles y recompensar sus hábitos sostenibles. No es tan escalable como el modelo CPM pero proporciona un dinero importante en los primeros momentos de la startup.

	Año 1	Año 2	Año 3	Año 4	Año 5
INGRESOS	15.000	238.494	1.835.257	5.418.153	9.040.653
EBITDA	-27.208	-68.050	500.113	3.634.209	7.251.909
(Beneficio antes de intereses, Impuestos, Amortización)					

Figura 21 - Ingresos y EBITDA de Liight

A continuación se muestran los costes fijos y de personal de Liight:

Coste mensual para la empresa de cada trabajador						
	CARGO	**Año 1**	**Año 2**	**Año 3**	**Año 4**	**Año 5**
Trabajador/socio	CEO	700	1.200	2.500	3.500	3.500
Trabajador/socio	CTO	700	1.200	2.500	3.500	3.500
Trabajador/coordinador	Desarrollador los	0	1.200	2.500	3.000	3.000
Trabajador	Marketing	0	700	2.500	3.000	3.000
Trabajador/coordinador	Community Manager	500	700	800	800	800
Trabajador	Comercialización	500	700	800	1.000	1.000
Trabajador	Responsable Recompensas	0	700	1.500	1.800	1.800
Trabajador	Responsable Financiero	0	500	1.200	1.800	1.800
Coste total mensual para la empresa		**2.400**	**6.900**	**14.300**	**18.400**	**18.400**
Número de empleados previsto		4	8	8	8	8

Figura 22 - Costes de personal de Liight

Gastos generales			
Concepto		**Importe mensual**	
Alquiler		200	€/M
Comunicaciones		100	€/M
Material oficina		60	€/M
Gestoría		250	€/M
Seguros médicos		178	€/M
SS administrador		254	€/M
Seguro Oficina		10	€/M
Suministros (electricidad, agua, internet)		150	
TOTAL FIJOS MENSUALES		**1202**	

Figura 23 - Costes generales de Liight

Capítulo 5. ANDROID

5.1 INTRODUCCIÓN A ANDROID

En este capítulo se pretende exponer el desarrollo hasta la fecha de la aplicación de Android, así como la lógica seguida para el desarrollo y una introducción del sistema operativo para así poder entender mejor sobre lo que está construida la aplicación.

El sistema operativo Android fue creado en 2003 y comprado por Google en 2005 pero no fue hasta 2008 que la primera versión de este sistema operativo fue lanzada al mercado. A día de hoy, Android tiene una cuota de mercado mundial del 90.8% (en España es de un 75%), lo que lo hace el sistema operativo más usado del mundo. Recordemos también que el mercado de aplicaciones genera unos ingresos de unos 180.000 millones de euros, cerca del 10% del PIB español.

Se dice que Android está basado en Linux (htt3), y es cierto que el núcleo del sistema operativo es el Linux Kernel, la parte de más bajo nivel de un sistema operativo que controla el acceso al hardware y optimiza el procesador. Sin embargo, Android no posee la mayoría de librerías GNU como la famosa *glibc* ni servidores de interfax X como Xorg, por lo que no es el Linux que estamos acostumbrados a ver.

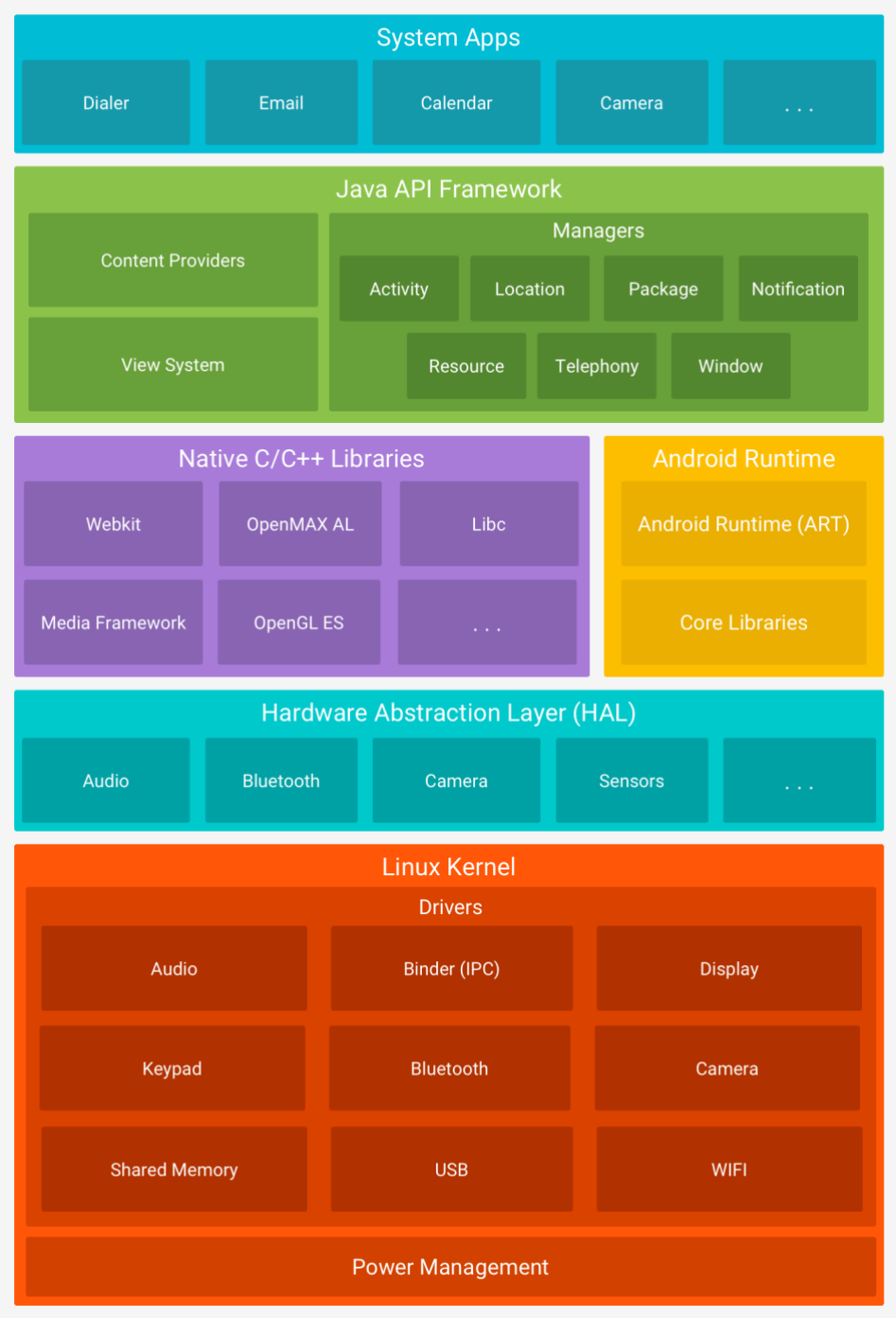

Figura 24 - Arquitectura de Android

Como se ha mencionado anteriormente, por razones de patentes, Android no puede incluir la máquina virtual de Java directamente en el sistema operativo, porque aunque el propio lenguaje Java no tiene problemas de licencias, la máquina virtual de Java no viene gratis. Es por esto que se decidió crear una máquina virtual propia, la máquina Dalvik (htt4).

La máquina Dalvik ejecuta bytecode dex, el equivalente al .jar de java, el cual ha sido compilado con el compilador dx desde un .class (tal y como se muestra en la figura de abajo) que a su vez ha sido compilado por un compilador de Java (javac) desde los archivos .java.

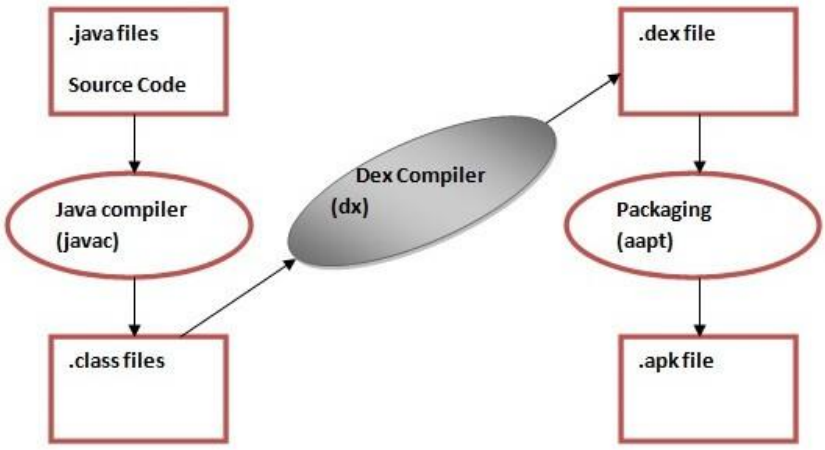

Figura 25 - Esquema del compilador Dalvik (dx)

5.1.1 APLICACIONES NATIVAS DE ANDROID

Como se ha mencionado anteriormente, en la actualidad existen dos opciones de desarrollar aplicaciones nativas en Android, Java o Kotlin. Aunque Kotlin es desde Mayo de 2019 la opción preferida de Google para desarrollar aplicaciones, Java sigue siendo el lenguaje con mayor comunidad, algo muy ventajoso considerando la poca experiencia de desarrollo en aplicaciones móviles y por tanto las altas probabilidades de errores en la aplicación.

5.1.1.1 Estructura

Las aplicaciones de Android se empaquetan en un archivo con extensión .apk (Android Application Package), que realmente es una carpeta comprimida con el formato zip. Esta carpeta contiene los siguientes archivos:

- **'AndroidManifest.xml':** El Manifiesto de la aplicación, especifica información como el nombre de la Aplicación, el logo, los diferentes componentes, idioma, permisos, pantalla de arranque, etc.
- **'classes.dex':** El archivo mencionado anteriormente, análogo al .jar de java. Contiene todos los .java compilados a .class y a su vez compilados mediante dx. Son los archivos de lógica que podrán ser leídos por la máquina Dalvik.
- **Carpeta 'res/':** Una carpeta que contiene los recursos básicos para que la app pueda funcionar (como imágenes o videos).
- **Otros:** Las APK también pueden contener otros tipos de archivos como el 'resources.arsc' o librerías nativas que no se explicarán en detalle por su poca relevancia en este trabajo.

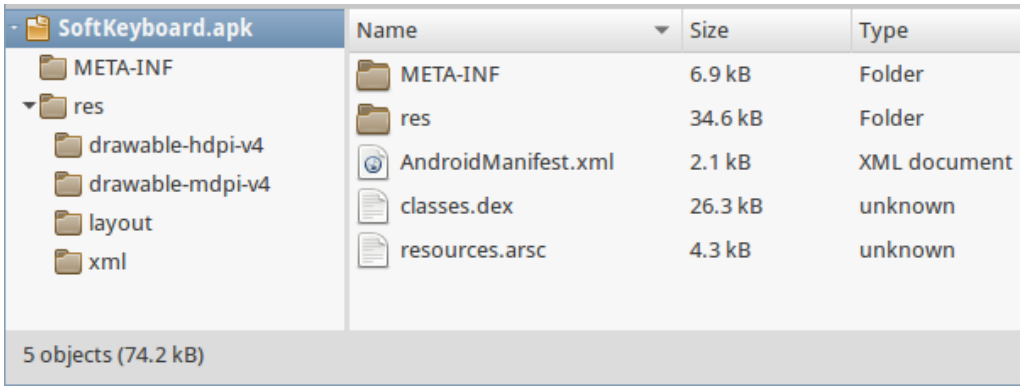

Figura 26 - Estructura de un APK

Aunque la APK es lo que se le entrega al sistema operativo para instalar la aplicación, el SDK (Software Development Kit) de Android crea este archivo por nosotros, siempre que cumplamos con sus especificaciones.

5.1.1.2 Componentes fundamentales

En Android existen cuatro componentes fundamentales (htt5) que sirven como bloques desde los que se construye el resto de la aplicación. Estos componentes se caracterizan por 'heredar' (por hablar en términos de POO) de la clase *Context*, una clase que inicializa el sistema operativo y a través de la cual se tiene acceso a todo el hardware del dispositivo. Los cuatro componentes son:

a) **Actividad:** En Android, una Actividad representa una pantalla de la aplicación, esto quiere decir que está obligada a contener una interfaz que en Android se diseña en un archivo .xml.

b) **Servicio:** Un Servicio se comporta de manera muy similar a una Actividad, con la diferencia que un Servicio no está vinculado con ninguna interfaz, es sólo un contexto desde el cual ejecutar código. Por ejemplo, un código en segundo plano que descargue una foto para mostrársela más tarde al usuario, tendría que estar alojado en un servicio.

c) Broadcast receivers: Son componentes que hacen posible responder a eventos de fuera de la aplicación, como una baja batería o un cambio de conectividad.

d) **Content providers:** Son componentes que permiten almacenar datos que pueden ser accedidos por distintas aplicaciones, siempre que su configuración lo permita.

5.2 LIIGHT

Tras la introducción del sistema operativo en el que se aloja la aplicación, es momento de explicar la estructura de la aplicación, tanto técnica como de cara al usuario.

En primero lugar, es importante comentar que cada sistema operativo tiene su propia identidad en cuanto a diseño tanto visual como de experiencia de usuario. Es por ello, que para desarrollar esta aplicación, se ha intentado aplicar los conceptos de *Material Design*, expuestos por Google en su página web (htt6).

5.2.1 EXPERIENCIA DE USUARIO

Para lograr una experiencia de usuario óptima, es importante ponerse en la piel del usuario e intentar no hacerle pensar demasiado. La aplicación tiene que serle útil al usuario y ofrecerle todas las funcionalidades posibles, pero siempre con la idea de que si un usuario no entiende la aplicación, por mucha utilidad que le pueda dar, éste no la va a usar. Por ello es importante diseñar pensando en el *mapa mental* (htt7) del usuario.

Por mapa mental se refiere a la estructura mental que se hace el usuario en su cabeza sobre la aplicación. Al instalar una aplicación, cada uno de nosotros subconscientemente estamos creando un mapa lógico de pantallas para poder entender mejor la aplicación y acordarnos de cómo llegar a cada parte de la aplicación. Por ejemplo, un buen sistema operativo tiene lo que llamamos carpetas que contienen otras carpetas o archivos. Estas carpetas no existen realmente, pero ayudan al usuario a crear un mapa mental que les permita entender mejor el producto.

Liight está pensado para tener muchas funcionalidades, pero deben estar bien estructuradas para no abrumar al usuario. Material design recomienda usar como elemento de navegación, una barra inferior como la de la figura de abajo.

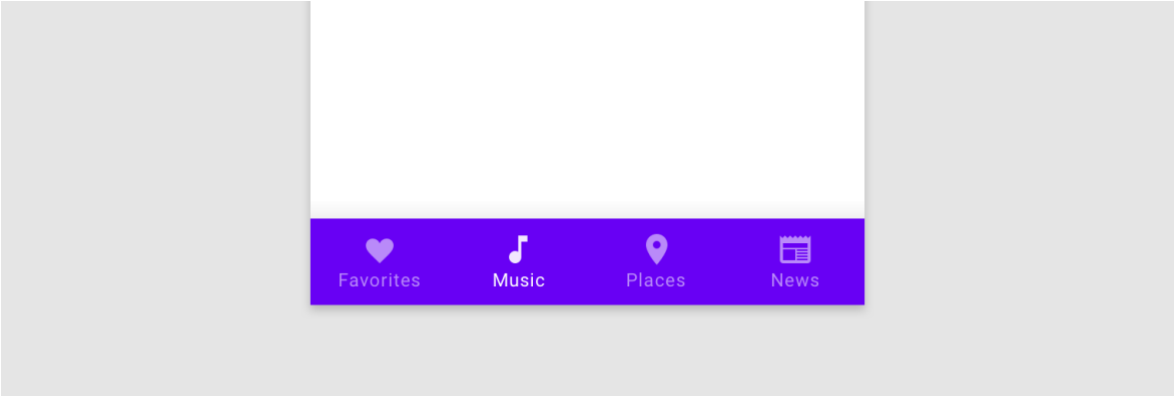

Figura 27 - Ejemplo de navegación inferior

Además, recomiendan tener entre 3 y 5 pantallas principales, y si se requieren más, hacer des estas pantallas accesibles únicamente desde las 5 principales. Es importante también que no se pueda 'llegar' al mismo sitio desde dos sitios diferentes (error común), ya que hace el mapa mental mucho más confuso. El mapa mental ideal es jerárquico en forma de árbol, aunque esto no es siempre posible en aplicaciones complejas.

5.2.2 FUNCIONALIDADES DE LIIGHT

Las funcionalidades básicas de Liight (por orden de más prioritario a menos prioritario) son las siguientes:

- **Realizar actividad sostenible:** Incluye empezar una actividad (elegir entre las diferentes opciones), información y validación mientras se realiza la actividad, informe final y asignación de puntos.

- **Login/Registro:** Incluye registro y login con Google, Facebook y email normal. También incluye aceptar términos y condiciones, introducir información básica (como nombre de usuario) y aceptar permisos como el de ubicación o el de detección automática de pasos.

- **Recompensas:** Incluye mostrar las diferentes recompensas disponibles para los usuarios en cada momento, así como información básica de cada una de ellas tipo foto, descripción, coste en Liights, tipo de recompensa (descuento directo o boleto para sorteo), descuento aplicado o tiempo para el sorteo, etc. También incluye el proceso de canjear los liights por la recompensa y la posterior comunicación para que finalmente se pueda disfrutar.

- **Retos:** Son una parte básica de la aplicación. Los retos son concursos temporales (normalmente duran 2 semanas) a los que los usuarios se unen y empiezan a competir. Tiene su propio ranking y un o más premios (normalmente suele ser uno para el ganador y otro para sortear entre los 10 mejores). Incluye también mostrar información básica (foto, descripción, etc.), funcionalidad de unirse al reto y segmentación de retos (para poder realizar retos solo para empresas o para un grupo de amigos).

- **Perfil:** Incluye poder consultar información básica como foto de perfil, nombre de usuario, liights, experiencia, nivel, CO2 reducido (por no haber realizado esas actividades en coche), actividades realizadas (cada una con su información básica) y estadísticas de uso.

- **Ranking:** Incluye mostrar un ranking general de todos los usuarios y los puntos y la posición del usuario que consulta el ranking.

Otras funcionalidades:

- **Ajustes:** Incluye pantallas para editar perfil, ver términos y condiciones, configuración de detección automática, contactar con nosotros y cerrar sesión.

- **Noticias:** Consiste en mostrar información de interés al usuario (como noticias relacionadas con la sostenibilidad, nuevas funcionalidades de la app, nuevas recompensas, nuevos retos, etc.)

- **Logros:** Consiste en motivar al usuario dándole la posibilidad de obtener puntos por realizar determinadas acciones, como canjear X descuentos, reducir una cantidad X de CO2 o participar en X retos. Estos logros además tienen cada uno su propio nivel, que cada vez recompensa con más puntos según el número de unidades de cada logro que se hayan realizado.

- **Notificaciones Push:** Consiste en desarrollar un sistema de notificaciones push para avisar de información de interés o simplemente motivar al usuario a seguir utilizando la aplicación. Incluye poder segmentar las notificaciones, automatizarlas y relacionarlas con partes de la aplicación de manera que si se hace click en una notificación, te pueda llevar a la parte de interés de la aplicación.

- **Notificaciones InApp:** Consiste en mostrar avisos mientras se usa la aplicación. Por ejemplo, cuando se sube de nivel, se consigue un nuevo logro, etc.

5.2.3 PANTALLAS DE LA APLICACIÓN

Tras hacer un estudio de lo que define una buena experiencia de usuario y las funcionalidades de Liight, se muestra el resultado de juntar ambos conceptos. La estructura de la aplicación es la siguiente:

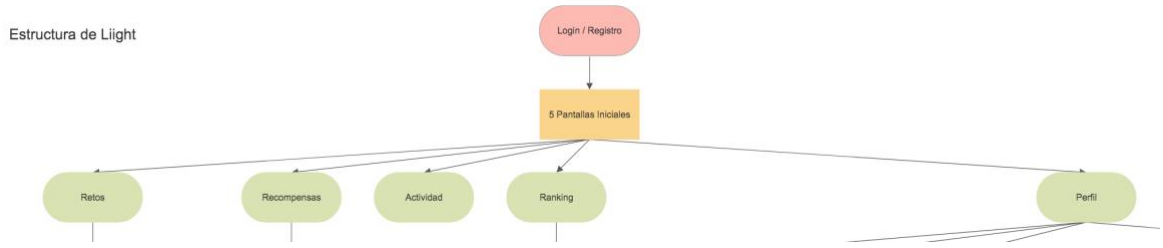

Figura 28 - Estructura de las pantallas básicas de Liight

5.2.3.1 Proceso de Registro/Login

Al iniciar por primera vez la aplicación, el usuario es bienvenido con una pantalla de Login/Registro. El proceso de Registro/Login está diseñado para ser lo más intuitivo posible. Al contrario que la mayoría de aplicaciones, la pantalla de Registro/Login cuenta únicamente con botones para iniciar sesión/registrarse con Google o Facebook, y un campo donde introducir el correo electrónico.

La experiencia dice que existe un alto porcentaje de usuarios que o todavía no sabe diferenciar entre registro y login, no se acuerda de si ya tiene cuenta, o el ver un campo donde poner su correo electrónico impulsivamente le hace meter su correo sin saber si se está registrando o está iniciando sesión. Por esta razón, se ha optado por proponer al usuario opciones que, independientemente de si el usuario debe registrarse o iniciar sesión, la opción que elija será la correcta. Esta pantalla se puede observar en la siguiente figura:

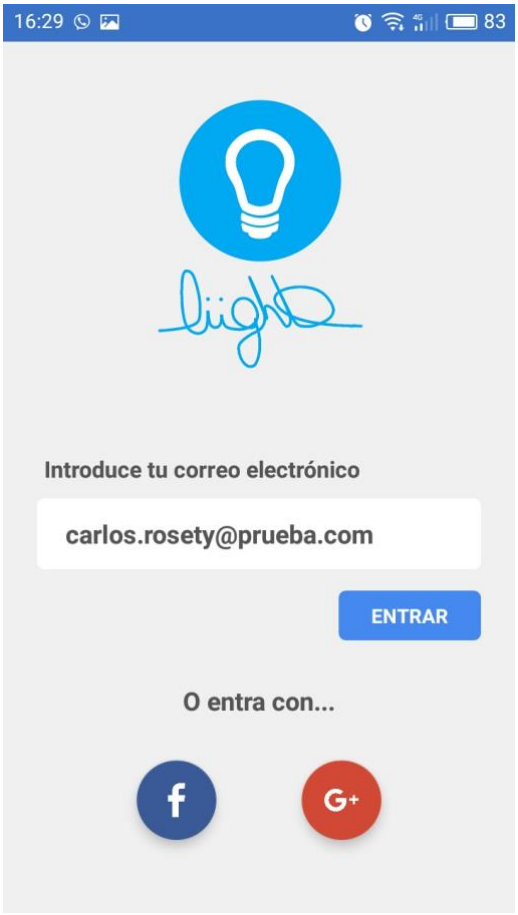

Figura 29 - Pantalla 1 de Registro/Login

Una vez se introduzca una opción, la aplicación hará una consulta a la base de datos para consultar si ese correo electrónico ya existe. Si el usuario no existiera, se le llevaría a un proceso de registro en el que, si ha introducido un correo electrónico, se le instará a crear una contraseña (como se muestra en la figura de abajo) y si ha entrado por Facebook o Google, será llevado directamente a la siguiente pantalla.

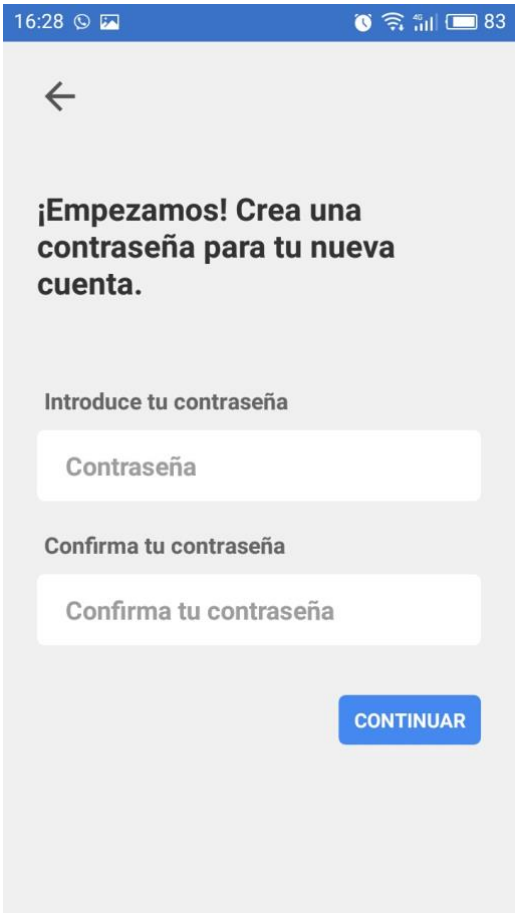

Figura 30 - Pantalla 2 de Registro/Login

El siguiente paso de registro es el de introducir el nombre completo y un nombre de usuario que debe de ser único, para poder aparecer en rankings y retos. También se le pide que confirme su correo electrónico por si ha entrado por Google o Facebook y el correo que estas plataformas devuelven es antiguo. El campo de correo electrónico es rellenado automáticamente. Se puede ver esta pantalla en la figura de abajo:

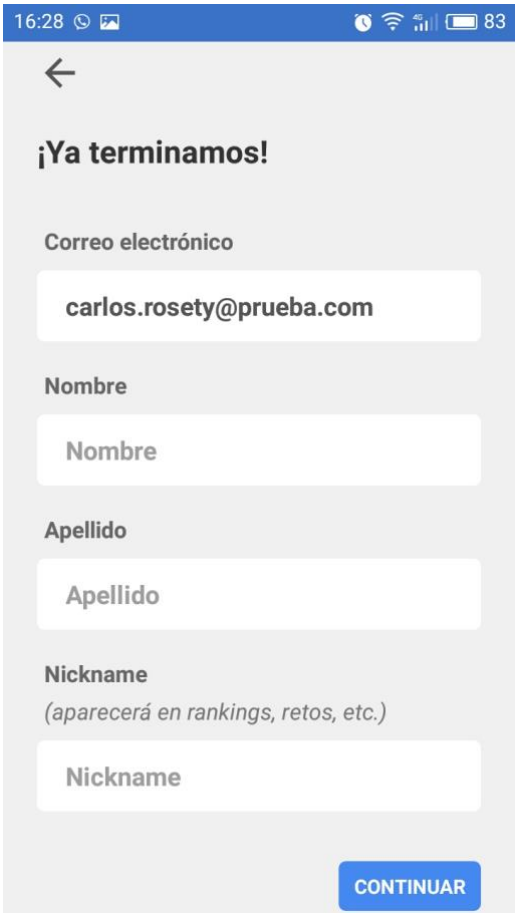

Figura 31 - Pantalla 3 de Registro/Login

Por último, se le pide al usuario que acepte los permisos de ubicación y detección automática para el correcto funcionamiento de la aplicación, así como aceptar los términos y condiciones:

Figura 32 - Pantalla 4 de Registro/Login

Esta pantalla llevaría directamente a la pantalla principal de la aplicación.

Si el usuario ya existiera en la base de datos, si entró a través de Facebook o Google se le llevaría directamente a la pantalla principal de la aplicación, mientras que si entró con correo electrónico, se le llevaría a introducir la contraseña y entonces entrar en la aplicación.

Además, si el usuario elige una opción que no es la que eligió para registrarse, la aplicación se dará cuenta y le redireccionará directamente a la opción de registro.

5.2.3.2 Pantalla de Actividad

Una vez pasado el proceso de registro, la aplicación introduce al usuario la barra de navegación donde se encuentran 5 pantallas: Retos, Recompensas, Actividades, Ranking y Perfil. Por defecto estará seleccionada la pantalla de actividad.

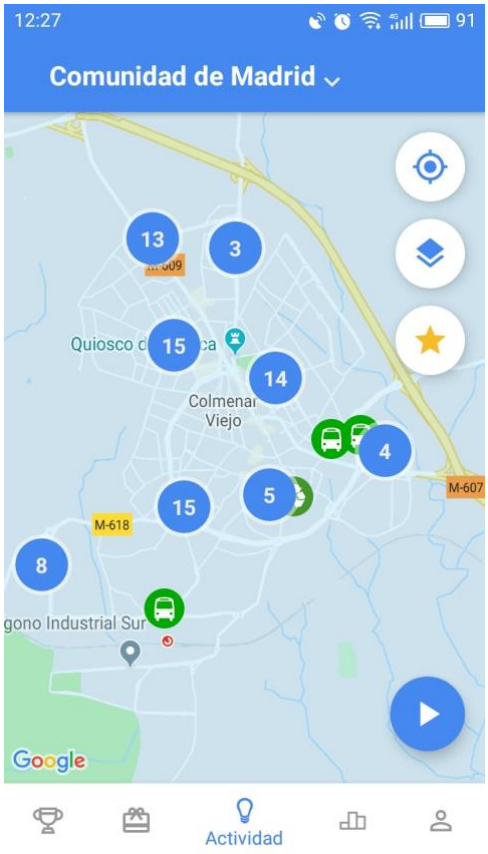

Figura 33 - Pantalla de Actividad

Esta es la pantalla principal de la aplicación y tiene numerosas funcionalidades, todas sin salir de la pantalla en sí, mediante popups o diálogos inferiores. A continuación se explican las diferentes funcionalidades:

a) Elegir área: Las actividades disponibles en Liight son Andar, Ir en bici, usar el Transporte Público y reciclar. Sin embargo, para poder validar actividades como el uso del transporte público o reciclar, hace falta incorporar una gran cantidad de datos normalizados a la base de datos de Liight, algo que no acostumbra a ser sencillo. Por ello, las actividades disponibles lo serán en función del área en el que se encuentre (o seleccione).

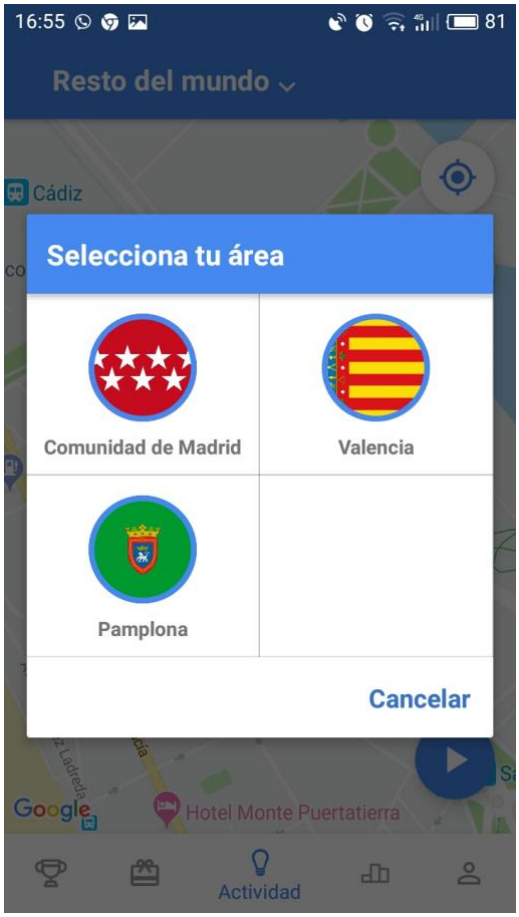

Figura 34 - Selección de Área

b) Filtrar mapa: Además de la satisfacción de ayudar al medio ambiente y aportar la motivación para hacerlo, uno de los principios de Liight es aportar al usuario servicios de conveniencia, es decir, simplificar la vida del usuario. Entre las funcionalidades que ayudan a esto, está la de mostrar información de utilidad en el mapa a los usuarios. Ahora mismo, en el mapa se muestran: paradas de autobús, contenedores de reciclaje (sólo en Madrid), comercios sostenibles conseguidos a través de nuestra colaboración con Tu Ciudad Saludable (TCS) y patinetes eléctricos de VOI. En el futuro se esperan introducir más soluciones de movilidad sostenible.

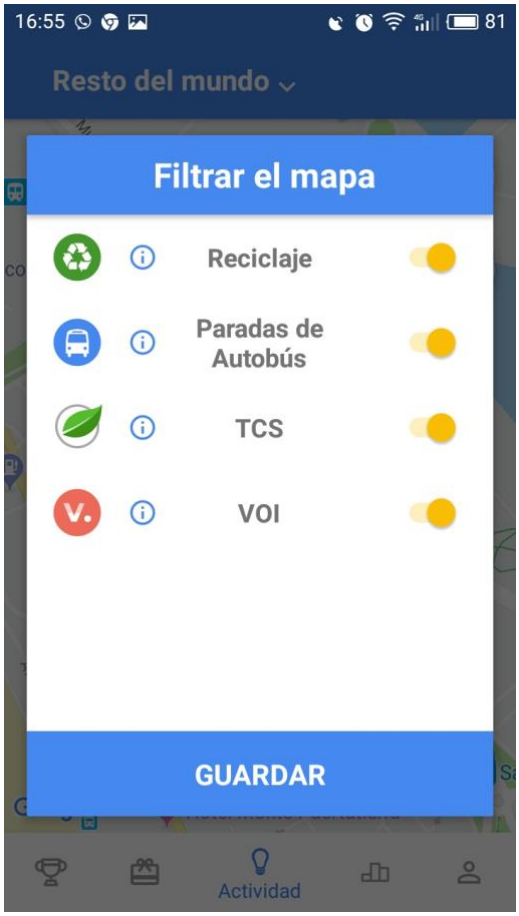

Figura 35 - Filtrado de capas

Además, se muestra información de cada tipo de marcador pulsando el botón de información:

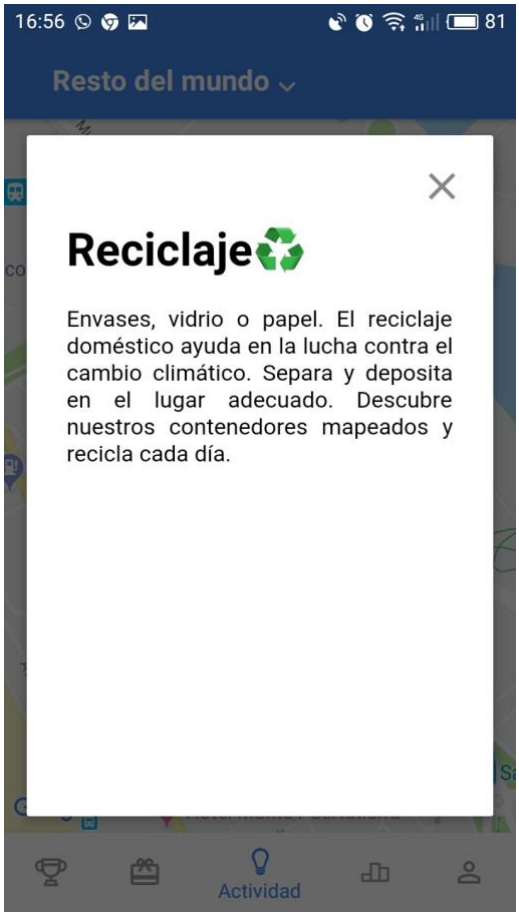

Figura 36 - Ejemplo de información de capa

c) **Tiempos de espera:** Esta funcionalidad permite a los usuarios obtener información de cuánto le queda a su autobús para pasar. Para ello, los usuarios pueden seleccionar una parada de autobús del mapa y se les abrirá un diálogo inferior con los tiempos de espera de cada línea. Además, tienen la opción de guardar las paradas como favorito introduciendo un nombre elegido por ellos, y acceder a ellas directamente desde el botón de favoritos en la esquina superior derecha. Desde este diálogo

también tendrá la opción de añadir una nueva parada introduciendo el código de parada.

Figura 37 - Paradas favoritas y tiempos de espera

d) Iniciar actividad: Esta es la funcionalidad principal de la aplicación. Se inicia pulsando el botón de 'Play' en la parte inferior izquierda de la pantalla. Esto lleva a elegir el tipo de actividad que se quiere iniciar:

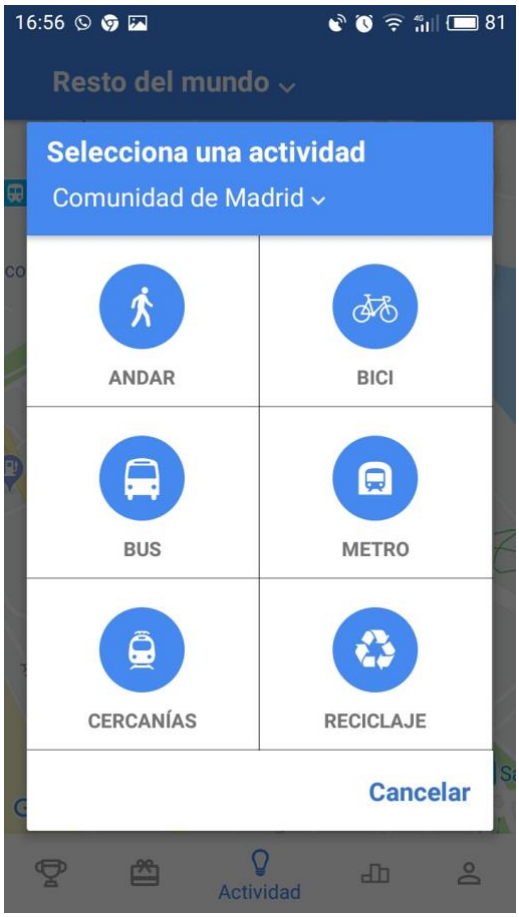

Figura 38 - Selección de tipo de actividad

Las opciones que se muestran en esta pantalla dependen del área seleccionada. Moverse en el mapa hará que el área se cambie automáticamente.

Una vez elegida la opción, se mostrará un diálogo inferior con la información de la actividad, y se pintará en el mapa el recorrido, en azul lo que está por validar y en verde lo ya validado.

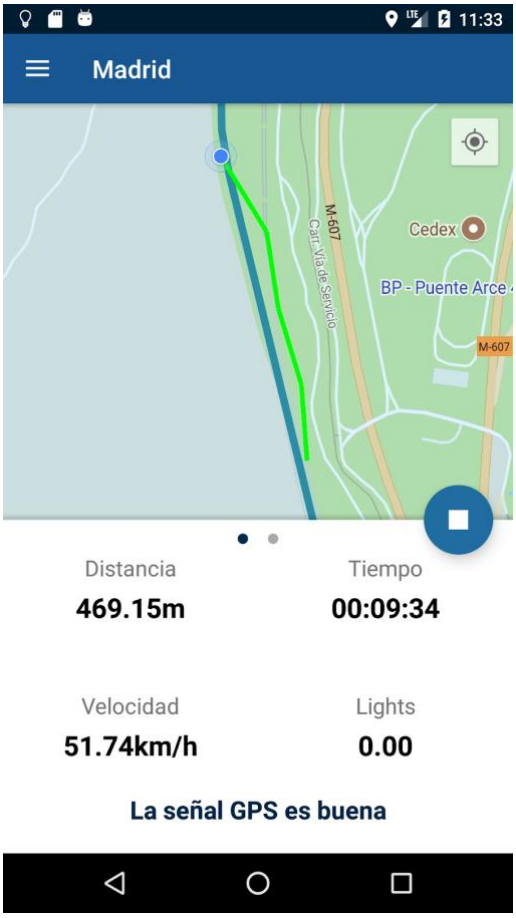

Figura 39 - Ejemplo de actividad en proceso

Durante la actividad de transporte público, el algoritmo del servidor detecta las líneas probables y las devuelve, el usuario tiene la opción de seleccionar su línea, y entonces se le pinta en el mapa el recorrido que está siguiendo, y las paradas que le quedan por pasar.

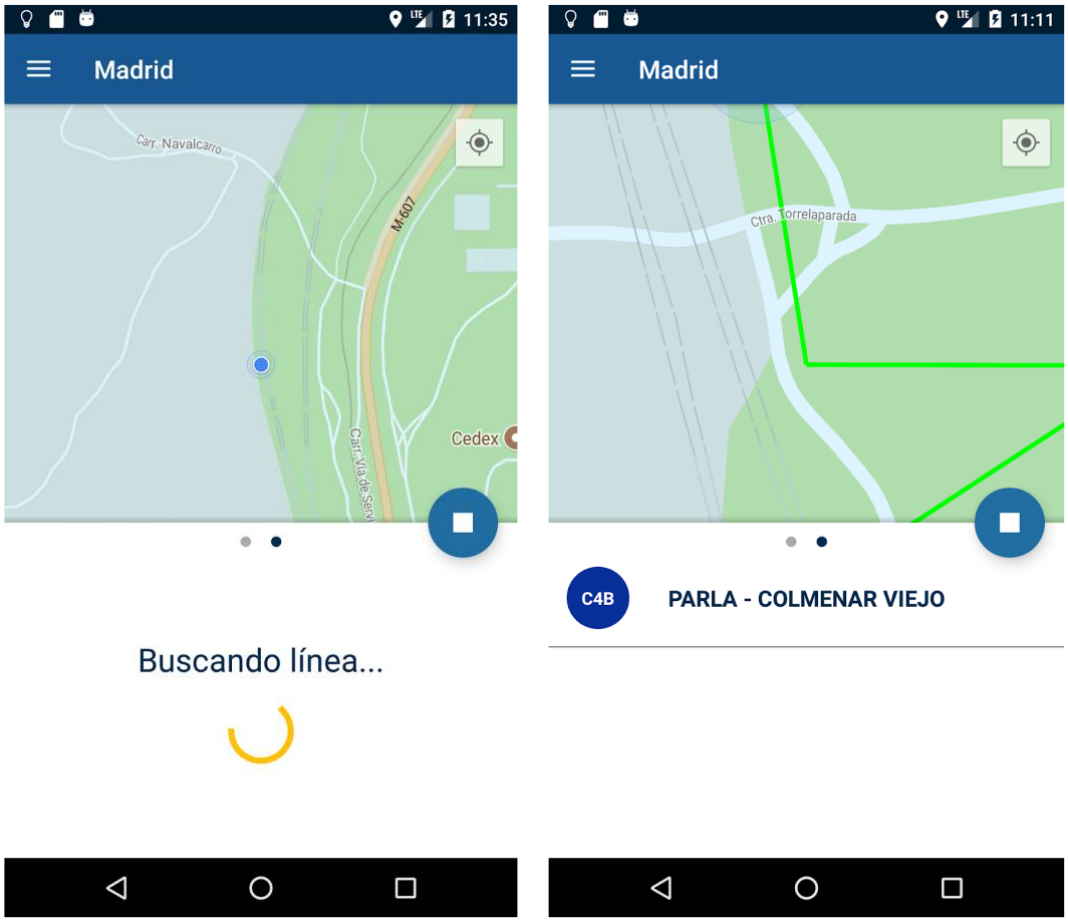

Figura 40 - Ejemplo de detección de línea

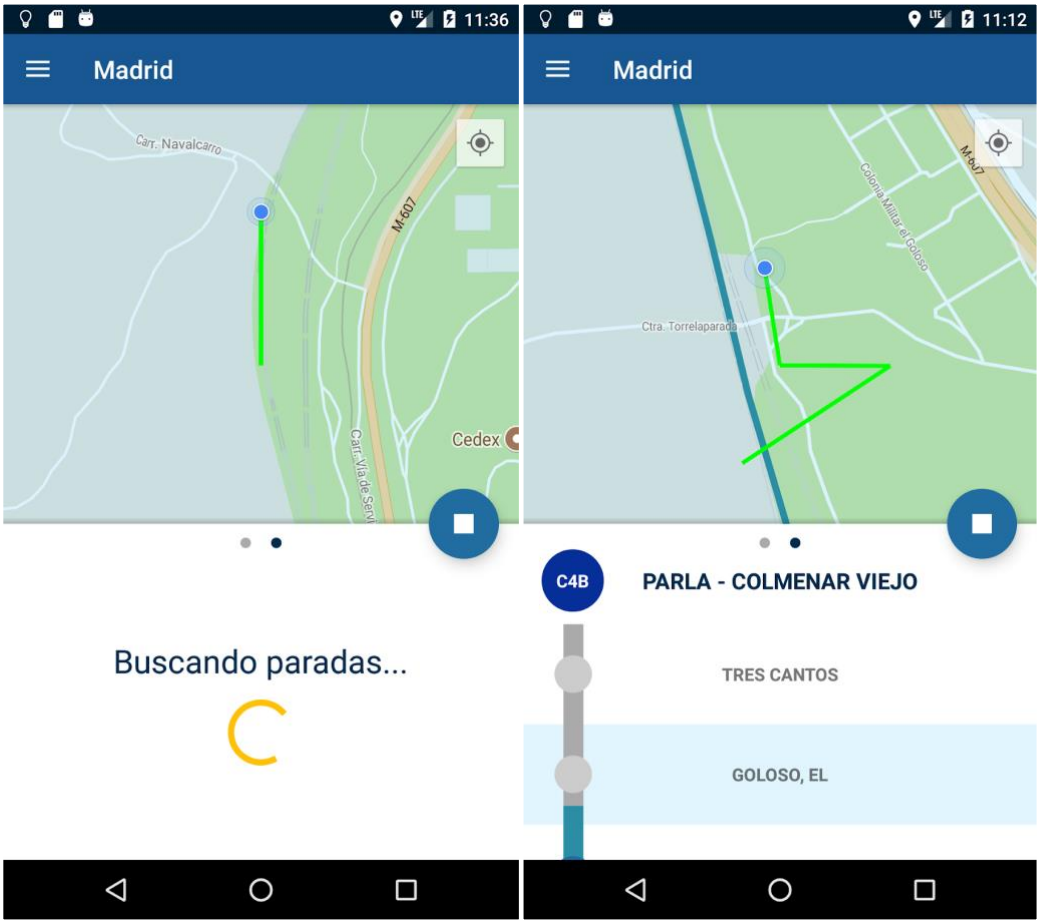

Figura 41 - Ejemplo de detección de parada

Cuando acaba la actividad, se muestra una pantalla resumen, y se añaden los puntos al usuario.

Figura 42 - Pantalla de fin de actividad

5.2.3.3 Pantalla de retos

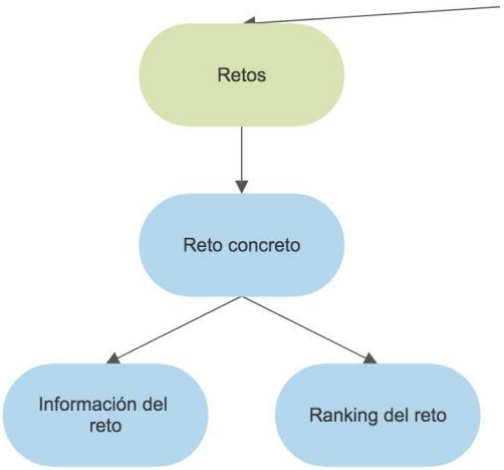

Figura 43 - Estructura de pantallas de retos

De las 5 pantallas principales, la de la izquierda del todo es la de retos. Esta pantalla muestra todos los retos disponibles y si están terminados o no.

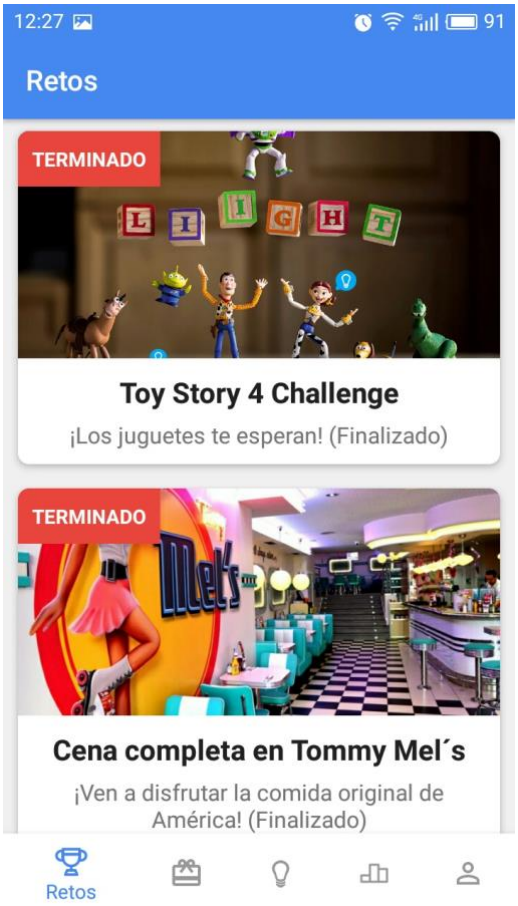

Figura 44 - Pantalla de retos

Al pulsar en un reto, se abre la información del reto junto con un menú superior para ver el ranking:

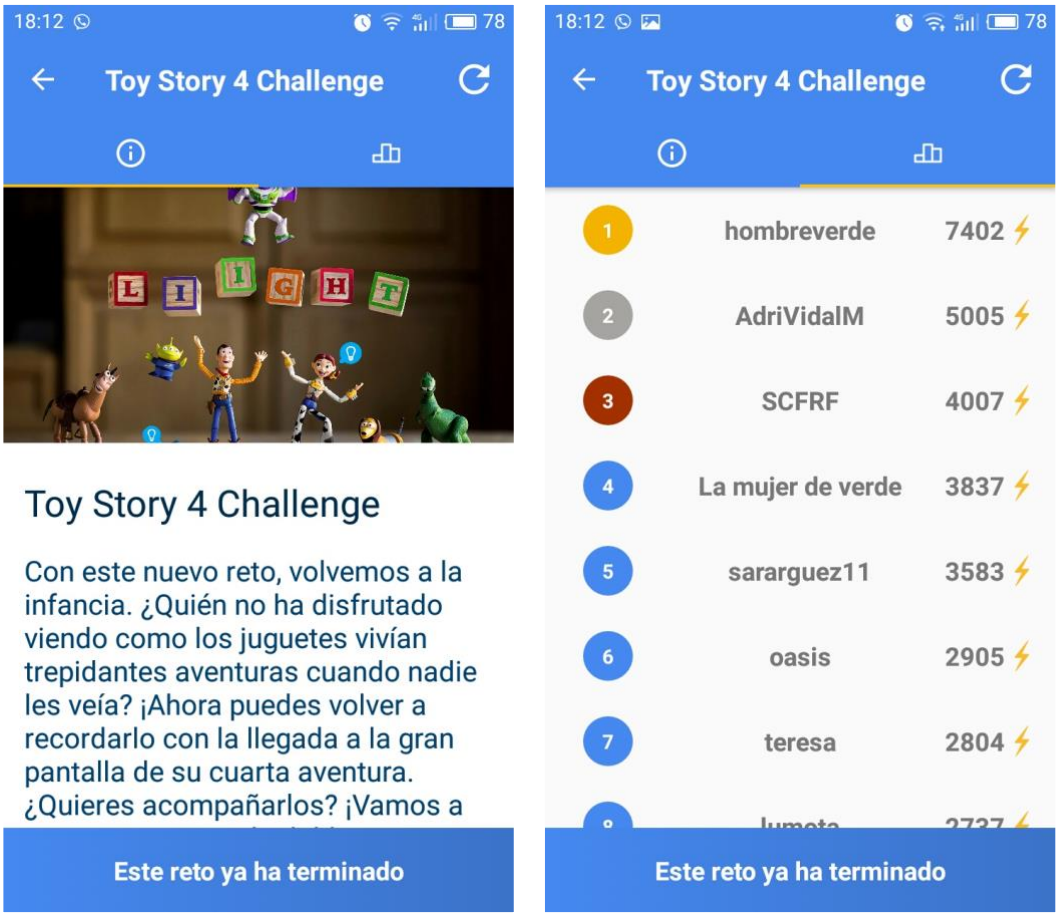

Figura 45 - Pantallas de información y ranking de reto

5.2.3.4 Pantalla de Recompensas

En esta pantalla se muestran todas las recompensas disponibles para el usuario e información del tipo de recompensa (descuento directo o boleto) y del coste en Liights.

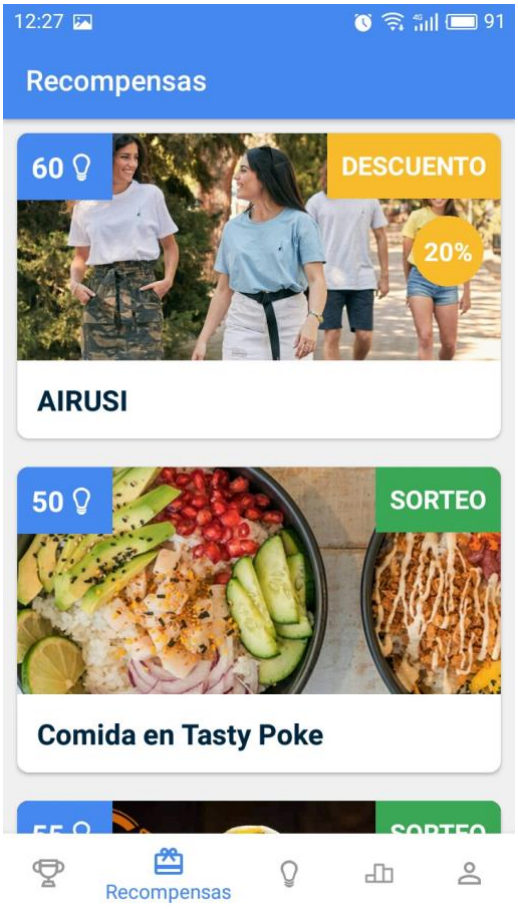

Figura 46 - Pantalla de recompensas

Al pulsar la recompensa, se puede ver más información como la descripción, y la cuenta atrás en caso del sorteo.

Figura 47 - Pantalla de descripción de sorteo

5.2.3.5 Pantalla de Ranking

Esta pantalla muestra el ranking general de usuarios y sus puntos. También muestra la posición del usuario que consulta el ranking y sus puntos. Si se pulsa sobre un usuario, te lleva a su perfil, donde se puede ver información muy básica del usuario.

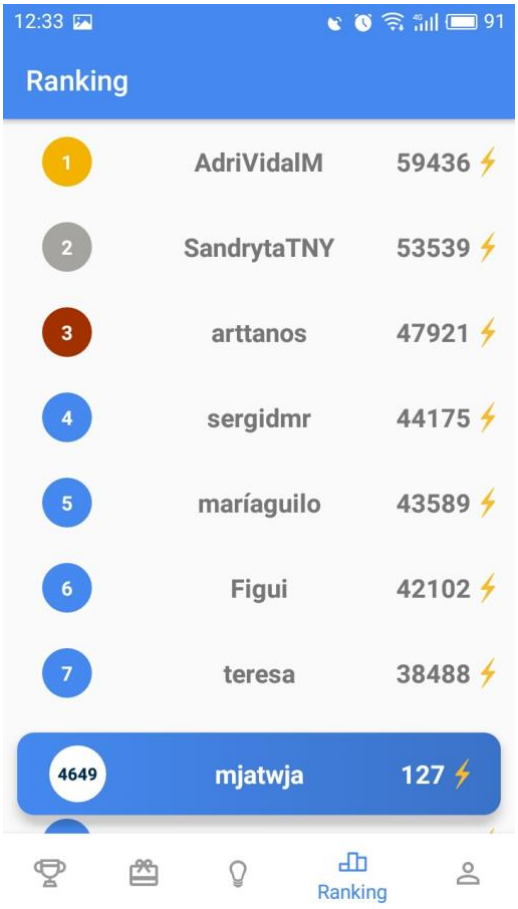

Figura 48 - Pantalla de ranking general

5.2.3.6 Pantalla de perfil

En esta pantalla se puede ver información básica como los liights, experiencia, co2 equivalente reducido y nivel. Desde esta pantalla se puede acceder también al menú de ajustes y a las pantallas de estadísticas, historial de actividades y logros.

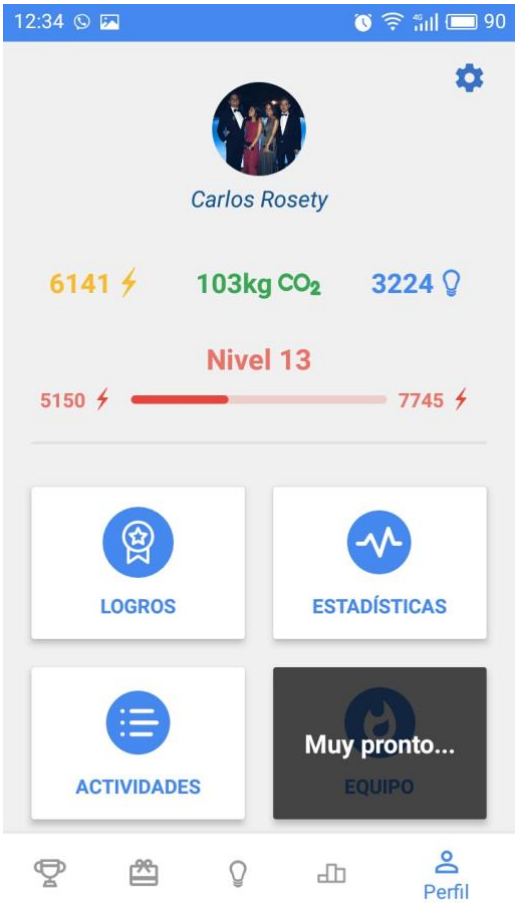

Figura 49 - Pantalla de perfil

5.2.3.7 Pantalla de logros

Esta pantalla permite al usuario ver los diferentes logros disponibles, el nivel en el que está cada uno, qué premio recibirá y cuánto le queda para el siguiente nivel.

Figura 50 - Pantalla de logros

5.2.3.8 Pantalla de estadísticas

Esta pantalla muestra las estadísticas básicas de cada usuario como las actividades realizadas, la distancia recorrida y los liights conseguidos por cada tipo de actividad.

Figura 51 - Pantalla de estadísticas

5.2.3.9 Pantalla de historial de actividades

Muestra las últimas 30 actividades, cada una indicando el tipo, la duración, la fecha y los puntos conseguidos.

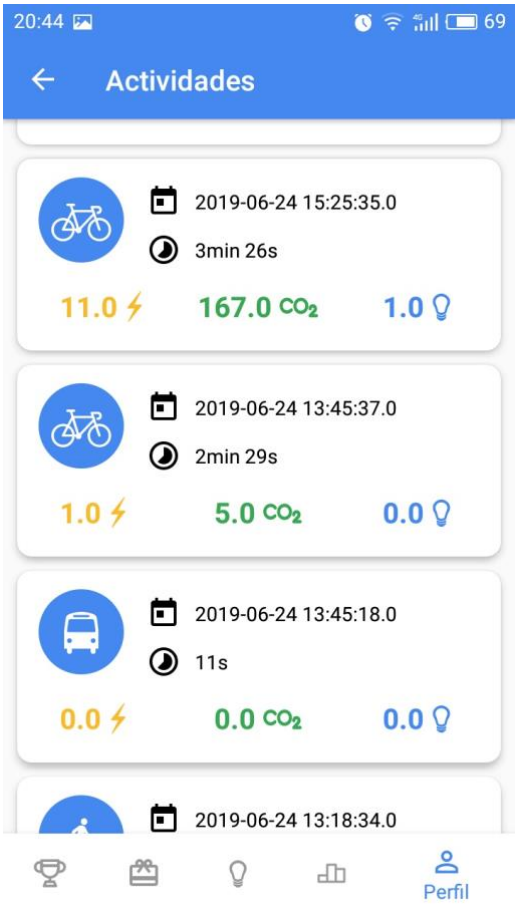

Figura 52 - Pantalla de historial de actividades

5.2.3.10 Pantalla de Ajustes

En esta pantalla de ajustes se puede editar el perfil, ir a configuración de detección automática, acceder al tutorial, ir a políticas de privacidad, contactar con los desarrolladores y cerrar sesión.

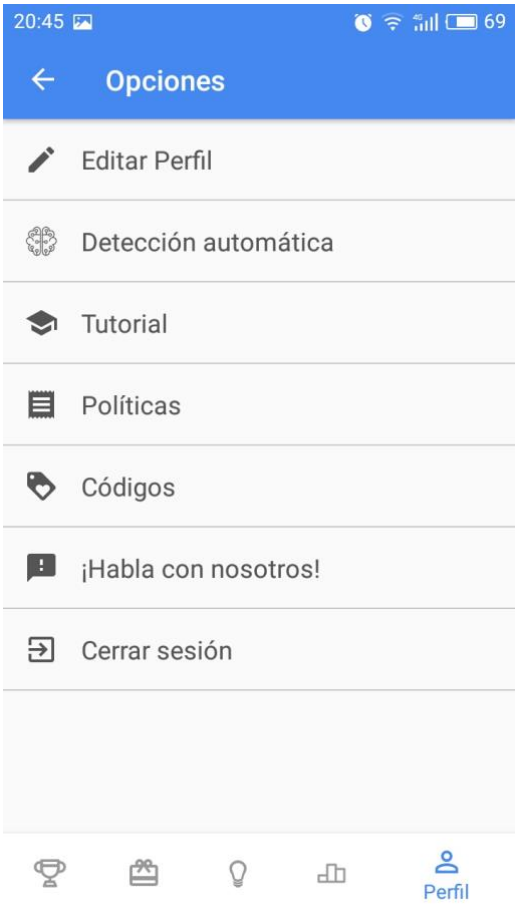

Figura 53 - Pantalla de opciones

Capítulo 6. INTEGRACIÓN CON EL BACK-END

El back-end de Liight está alojado en el Cloud de Amazon, AWS (Amazon Web Services). La lógica del servidor está en entornos Elastic Beanstalk de PHP y Tomcat. La idea es que todas las llamadas de la aplicación que sean simples GETs, UPDATE, PUT o DELETE a la base de datos, se hagan mediante el servidor de PHP, sencillo y rápido para hacer de API. El servidor de Java sin embargo, es usado para llamadas que requieran una lógica más compleja, como el algoritmo de validación de actividades de transporte público o la gestión de avisos y asignación de puntos de los logros y las subidas de nivel.

6.1 SERVIDOR PHP

Ejemplos de llamadas importantes al servidor de PHP son los siguientes:

- **https://www.php.liightes.com/api/v2/app/login.php - GET**

Parámetros:

id: id de facebook o de google, nada si es con mail

firid: firebase id

login_type: google, facebook, email

name: nombre completo

second_name: apellido

gender: male, female

email

img_uri: url de la foto de perfil, 'none' si no está disponible o si se registra con email

platform: "ios" o "android"

Respuesta: Devuelve "ok" cuando debe ir a la pantalla normal de la aplicación y devuelve "no_nickname" cuando debe pasar por las pantallas de tutorial y la de introducir nombre de usuario, esto en teoría sólo debería pasar cuando el usuario entra por primera vez.

- **https://www.php.liightes.com/api/v2/app/update_user.php - GET**

Los campos marcados como *opcional* quieren decir que si los pasas se actualizan, sino no. En el caso de la pantalla final del tutorial para mandar el username mandar sólo este campo. Esta query además valida los campos así que atento a la respuesta.

Parámetros:

firid: firebase id

username: *Opcional*

email: *Opcional*

name: *Opcional*

second_name: *Opcional*

terms: *Opcional* (siempre debe ser = a 'accepted', sino es como si no se incluyera)

Respuesta: JSON de la siguiente forma:

{"username": "ok" (se ha actualizado correctamente) | "already_in_use" (ya existe) | "invalid" (está vacío, es un espacio o cualquier otro filtro típico) | "null" (no se ha pasado como parámetro),

"email": "ok" (se ha actualizado correctamente) | "already_in_use" (ya existe) | "invalid" (está vacío, es un espacio o cualquier otro filtro típico) | "null" (no se ha pasado como parámetro),

"name": "ok" (se ha actualizado correctamente) | "invalid" (está vacío, es un espacio o cualquier otro filtro típico) | "null" (no se ha pasado como parámetro),

"second_name": "ok" (se ha actualizado correctamente) | "invalid" (está vacío, es un espacio o cualquier otro filtro típico) | "null" (no se ha pasado como parámetro) }

"terms": "ok" (se ha actualizado correctamente) | "error" (ha habido algún error con el nickname, el email o el campo GET de 'terms' no es igual a 'accepted' o no se ha incluido. No se registra constancia en el servidor de haber aceptado términos y condiciones, no se deberá pasar a la siguiente pantalla si este campo no está en 'ok')

- **https://www.php.liightes.com/api/v2/app/get_notifications.php - GET**

Parámetros:

firid

limit (10 son los 10 primeros, 50 son del 41 al 50)

Respuesta: en JSON

En el campo *action* de cada notificación podrán haber los siguientes tipos con sus respectivas etiquetas:

Comenzar actividad -> "*start_activity_action"*

Ranking -> "*ranking_action"*

Market -> "*market_action"*

Perfil-> "*profile_action"*

MarketItem -> "*market_item_action" (extra= id del marketItem)*

ChallengeItem -> "*challenge_item_action" (extra= id del challengeItem)*

New -> "*new_action" (extra= id del newItem, pantalla como el challenge pero sin la tab del ranking, sólo una foto, un título y una descripción, actualmente no está en la app de Android)*

WebView -> "*webview_action" (extra= url que mostrar en la webView, tiene que estar integrada en la app)*

AppStore -> "*app_url_action"*

URL con Safari -> "*url_action" (extra=url a la que ir desde el navegador predeterminado)*

Pantallas de ajustes -> "*settings_action" (extra= en principio no hay porque sólo habrá una pantalla de ajustes pero puede cambiar)*

- **https://www.php.liightes.com/api/v2/app/get_ranking.php - GET**

Parámetros:

firid

Respuesta: en JSON

- **https://www.php.liightes.com/api/v2/app/get_challenge_item.php - GET**

Parámetros:

firid

challenge_id

Respuesta: en JSON {challenge (titulo, foto, descripcion), ranking, joined (true, false, devuelve si el usuario ya se ha unido al reto)

- **https://www.php.liightes.com/api/v2/app/send_join_challenge.php - GET**

Parámetros:

firid

challenge_id

Respuesta: Siempre “ok”

- **https://www.php.liightes.com/api/v2/app/get_market.php - GET**

Devuelve todos los articulos disponibles en el market ordenados, esto cambiará cuando haya muchos artículos para no sobrecargar al cliente. La información de cada artículo no es completa, sólo la imprescindible para mostrarla en la pantalla principal del market

Parámetros:

firid

Respuesta: en JSON

Un ejemplo de código PHP que se conecta a la base de datos y sirve de API Rest:

```
<?php

$noAuth = false;
include "../res/filter_in.php";

$challenge_id = isset($_GET['challenge_id']) ? $_GET['challenge_id'] : '';

date_default_timezone_set("Europe/Madrid");
$timestamp = date_default_timezone_get();
$mydate = date('Y-m-d H:i:s', strtotime($timestamp));


//OWN UID
$sql = "SELECT id FROM usuarios WHERE firebase_id='{$firid}'";
$result = $mysqli->query($sql);
$row = mysqli_fetch_array($result);
$ownUID = $row['id'];


$sql = "SELECT * FROM challenges WHERE id={$challenge_id}";
$result = $mysqli->query($sql);

$challenge = array();
$challenge = mysqli_fetch_assoc($result);

$sql = "SELECT start_date, finish_date, ranking_query FROM challenges WHERE
id={$challenge_id}";
$result = $mysqli->query($sql);
$row = mysqli_fetch_array($result);
```

```
$sql = stripslashes($row['ranking_query']);

$sql = str_replace('[start_date]', $row['start_date'], $sql);
$sql = str_replace('[challenge_id]', $challenge_id, $sql);
$sql = str_replace('[finish_date]', $row['finish_date'], $sql);


$result = $mysqli->query($sql);

$rankings = array();

$ownName="null";
$ownLiights="null";
$joined=false;

$i = 0;
while ($r = mysqli_fetch_assoc($result)) {

    if ($i < 50) {
        $r['lights'] = (string)floor($r['lights']);
        $rankings[] = $r;
    }


    if($ownUID==$r['uid']){
        $ownPos=$i+1;
        $ownName=$r['name'];
        $ownLiights=$r['lights'];
        $joined=true;
    }

    $i++;
}

unset($challenge['ranking_query']);

//isFinished
$sql = "SELECT id FROM challenges WHERE {$challenge_id} = id and
finish_date>'{$mydate}'";
$result = $mysqli->query($sql);
if (!$result->num_rows > 0) {
    $isFinished = true;
} else {
    $isFinished = false;
}
```

```
$res = array("challenge" => $challenge, "rankings" => $rankings, "joined" =>
$joined, "is_finished" => $isFinished, "own_uid" => $ownUID, "own_pos" =>
$ownPos, "own_name" => $ownName, "own_liights" => $ownLiights);


echo toUtf8(json_encode($res));

include "../res/filter_out.php";
?>
```

6.2 SERVIDOR DE JAVA

Figura 54 - Logo de Tomcat

Como se ha mencionado anteriormente, este servidor se usa principalmente para tareas más complejas que una simple API Rest. La lógica que se procesa en este servidor es la siguiente:

a) **API tiempos de espera transporte público:** Esta API es compleja ya que necesita normalizar los datos que se reciben de la API dinámica de la comunidad de Madrid.

b) **Notificaciones Push:** Todo lo referente al envío de notificaciones push se procesa aquí. Incluye la posibilidad de segmentar las notificaciones por audiencias, gestionar la información extra que se envía en la notificación, notificar del éxito o fracaso del envío, etc. Además, estas notificaciones se envían a través de Firebase Cloud Messaging (FCM), el cual tiene una limitación de 1000 recipientes por envío, por lo

que esta sección también se ocupa de automáticamente gestionar el envío, rompiéndolo en varios más pequeños y volviendo a juntar la respuesta.

c) **CronJobs:** A través de Lambda Serverless, el servicio de AWS que permite ejecutar código sin tener que preocuparse por runtimes y servidores, se ha configurado que se llame a un determinado endpoint del servidor de Java cada minuto. Esto cuesta menos de 1 céntimo al año y permite alojar los CronJobs en el mismo servidor de Java.

   Los CronJobs son trabajos que realiza el servidor sin que ningún usuario haya solicitado nada. Por ahora únicamente hay 3 activos, pero se espera que haya más. Estos 4 son:

   - **NotificationCronJob:** Desde el dashboard de Liight, se puede programar el envío de una notificación push para que se envíe en un determinado momento, o una vez a la semana o por ejemplo, a todos los usuarios que se hayan registrado pero no hayan usado la aplicación en 2 días, al pasar 48 horas se le manda. Para eso, hace falta que haya un script funcionando, validando todas estas opciones.

   - **ActivityCronJob:** A veces, una actividad se cierra por parte del cliente pero el servidor no se llega a enterar de que el cliente la ha cerrado (por ejemplo, se le acaba la batería al móvil o se queda sin internet). La idea es que si pasan 3 horas sin que el móvil de señales de vida, el servidor da automáticamente por cerrada la actividad.

   - **WarningsCronJob:** Este CronJob está hecho para avisar a los responsables de las recompensas y los retos de fechas clave como fecha de finalización de un reto o de un sorteo.

d) Achievements: Esta sección se ocupa de asegurar que cada usuario ha sido bonificado por los logros, así como de avisar mediante una notificación push (si el usuario no está en el momento en la app) o mediante una notificación inApp (si el usuario está usando la app) de la subida de nivel del logro. También se encarga de calcular el nivel que le corresponde a cada usuario por su experiencia.

e) GoogleFit: Esta sección se ocupa de procesar las actividades detectadas automáticamente por Google Fit. Primero valida que lo que sube el móvil tiene sentido y no es falso, y posteriormente normaliza la información, computa los puntos que merece y actualiza la base de datos. También se ocupa de que no se suba la misma actividad más de una vez.

f) Actividad: Este es el núcleo de la aplicación y se explica en más detalle en el capítulo siguiente.

6.3 BASE DE DATOS MYSQL

La base de datos de Liight está alojada también en el Cloud de Amazon. Cuenta con numerosas tablas, pero las más importantes son:

- **Usuarios:** Tabla con los datos de los usuarios. Las contraseñas no se guardan aquí sino que se almacena un hash en los servidores de Firebase, el cual es el encargado de la autenticación en la aplicación. En esta tabla se encuentran campos el *firid*, que identifica inequívocamente a un usuario, la información básica (email, nombre, nombre de usuario, plataforma, fecha de registro), versión de la aplicación, foto de perfil, etc.

- **Liights History:** Esta tabla contiene toda la información relativa a la asignación de puntos dentro de la aplicación. En vez de designar un campo de la tabla *usuarios* como puntos que tiene en un momento dado un usuario, se ha optado por insertar una

nueva fila en esta tabla cada vez que se le añaden o se le quitan puntos a un usuario. Además estas filas tienen información de la fecha en la que se produjo y la acción que causó el cambio en puntos.

- **Market:** Contiene toda la información relativa a las recompensas de Liight.
- **Activity History:** Contiene toda la información relativa a cada actividad que se realiza.
- **Challenges:** Contiene toda la información relativa a los retos disponibles para los usuarios de Liight.
- **Analytics:** Esta tabla sirve para subir todo tipo de información que puede ser de utilidad desde la aplicación. Ejemplos de filas de esta tabla son: *activity_resume*, que se inserta cada vez que el usuario entra en la aplicación, *ranking_exit o market_item_exit*, que son filas insertadas cada vez que un usuario sale de la pantalla de ranking o de un artículo de recompensa. Además, van acompañadas de un campo que indica el tiempo en segundos que se ha estado en esa pantalla. En el caso de *market_item_exit*, también se incluye el identificador de recompensa, muy útil para saber qué recompensas están llamando más la atención a los usuarios.

Capítulo 7. SISTEMA DE VALIDACIÓN DE ACTIVIDADES

El sistema de validación de actividades es el núcleo técnico de la aplicación, y lo que le da realmente el valor que tiene. De los competidores que tenemos, ninguno valida las actividades, algo que creemos que es clave para el éxito de una aplicación de este tipo.

De todos los tipos de actividad que se pueden y se podrán realizar en Liight, el proceso de validación se separa en actividades continuas y actividades no continuas. Las continuas son las que tienen un proceso de iniciar actividad, realizar actividad y detener actividad, es decir todas las actividades menos reciclar.

7.1 VALIDACIÓN DE ACTIVIDADES CONTINUAS

Para validar actividades continuas, desde Android se arranca un servicio en segundo plano, de manera que si el usuario sale de la aplicación mientras está en el autobús para mandar un mensaje, la actividad no se pierda.

La idea del algoritmo de validación de actividades continuas, es que el dispositivo va mandado lo que en la aplicación se llaman MapPoints. Estos MapPoints realmente son puntos en el mapa junto con un timestamp, que indica el momento en el que se consiguió ese punto. Los MapPoints se recogen cuando se cumplen una serie de requisitos (que pasen mínimo 3 segundos, que haya un mínimo de desplazamiento y que la precisión sea suficiente) y posteriormente son añadidos a la cola de MapPoints para mandar.

La comunicación con el servidor es a modo Rest, aunque realmente es Stateful en cuanto a que el servidor no sólo computa su respuesta con la información de la request, si no que necesita información de la actividad activa, MapPoints anteriores e información externa relativa a transporte público (en el caso de este tipo de actividades). Además, la idea es que si en algún momento el dispositivo se queda sin conectividad, la información se va guardando a nivel local, y es enviada una vez se ha recuperado la conexión hasta que se confirme que el servidor ha recibido los mappoints nuevos.

El servidor se ocupa de validar los mappoints y decidir si es una actividad sostenible válida o no, y si lo es, cuantos puntos merece el tramo que se acaba de enviar.

7.1.1 ACTIVIDADES DE ANDAR/BICI

La validación de este tipo de actividades difiere de las de transporte público en cuanto a que no se necesita información de *smartcity* relativa a rutas de transporte público.

Realmente, la validación que se hace a este nivel es por velocidad y está previsto que en el futuro, también tenga en cuenta la probabilidad que devuelve la red neuronal de Google de ser una actividad de andar o bici dados los datos del acelerómetro y el giroscopio del dispositivo.

A continuación se muestra el código de la asignación de puntos a una actividad de tipo Andar/Bici dadas la distancia y la velocidad:

```
public static double processLiightsExp(String activityType, double distance,
double speed) {

            if (speed == 0 || distance == 0) {
                return 0;
            }

            switch (activityType) {

                case Activity.WALK:
                    return 20 * (distance / (speed * 360))
```

```
                            * (0.3 * Math.pow(speed, 0.5)
                            * (1 / (1 + Math.exp(0.3 * (speed - 25)))))
                            * generalLiightsExpMultiplier;

                case Activity.BIKE:
                    return 20 * (distance / (speed * 360))
                            * (0.3 * Math.pow(speed, 0.5)
                            * (1 / (1 + Math.exp(0.3 * (speed - 55)))))
                            * generalLiightsExpMultiplier;

            }
            return 0;
        }
```

7.1.2 ACTIVIDADES DE TRANSPORTE PÚBLICO

Este tipo de actividades son las más complicadas de validar. El mayor problema es distinguir entre un coche y un autobús, puesto que la sofisticada red neuronal de Google sólo proporciona la probabilidad de que el usuario esté en un vehículo, pero no sabe distinguir entre autobús, tren/metro y coche, por lo que un usuario podría ir en su coche y la red de Google le validaría.

Puesto que la red neuronal de Google no es una opción, se debe recurrir a datos de rutas de transporte público para distinguir si el usuario está siguiendo una ruta de autobús o si se encuentra en una carretera en su coche.

Además, puesto que la escalabilidad debe ser uno de los principios de Liight, el sistema debe funcionar en cualquier ciudad, luego deben ser los datos los que se adapten al algoritmo, y no el algoritmo el que se adapte a cada ciudad. También los datos usados no deben de ser demasiado difíciles de conseguir.

Una idea que soluciona estos problemas, es la de usar el formato de transporte público en ciudades de Google el GTFS (Google Transit Feed System). Esta idea consiste en que cada ciudad, si quiere aparecer en Google Maps (no es Maps el que va detrás de cada ciudad sino al revés), debe proporcionar a Google un archivo gtfs.zip en el que cumpla con unos

requisitos mínimos de datos y de formato. Estos requisitos mínimos son suficientes para hacer funcionar el algoritmo de Liight, y puesto que prácticamente todas las ciudades medianas y grandes del mundo están en Google Maps, estos archivos existirán para cada una de ellas, y el trabajo únicamente sería hacerse con él.

Para simplificar el proceso, se ha desarrollado una herramienta que introduciendo el archivo gfit.zip, normaliza los datos y los introduce en la base de datos de Liight de manera optimizada (usando índices espaciales), listos para ser usados en la aplicación, de manera que si se tuviera el archivo gtfs.zip, se tardaría literalmente 1 minuto en estar disponible el transporte público de esa ciudad en la aplicación.

El algoritmo.

El algoritmo en sí, consiste en, para cada MapPoint que se recibe, ver la proximidad a una ruta de transporte público de esa ciudad. Esta información se almacena en la base de datos y cuando llega el siguiente MapPoint, mira la proximidad únicamente con las líneas anteriormente encontradas como 'próximas'. Se va calculando y guardando la media de proximidad a cada ruta, y junto con otro algoritmo sencillo que devuelve si cumple con el sentido de la ruta, se establece una probabilidad de que el usuario realmente esté realizando esa ruta.

Debido a la capacidad de optimización de los índices espaciales, el tiempo de ejecución del algoritmo es sorprendentemente rápido, siempre inferior a 300ms, y por tanto factible para la aplicación.

En el futuro se podría mejorar creando una red neuronal que sepa diferenciar entre el movimiento de un autobús y el de un coche.

7.2 VALIDACIÓN DE ACTIVIDADES NO CONTINUAS

A día de hoy, estas actividades realmente son únicamente las de reciclaje. Existen varios niveles de validación de este tipo de actividad, cada uno con sus ventajas y desventajas:

- **Nivel alto de validación:** Consistiría en instalar en los contenedores un equipo de sensores tipo haz infrarojos o cámara de fotos con red neuronal de detección. Esta instalación tendría un alto coste por contenedor y no sería factible a día de hoy en España. Esta opción no se ha llegado a testear.

- **Nivel medio de validación:** Consiste en colocar sensores de ultra sonidos que validen que "algo se ha echado". Estos sensores son más baratos y no tendrían que necesitar una conexión a internet puesto que se podría codificar la información en un QR que leyera la aplicación. Este proyecto ha sido ya testeado en un proyecto conjunto Ecoembes-Liight y desde Liight se ha desarrollado tanto la parte Arduino del contenedor (subcontratada), como la gestión de las tarjetas IoT y la comunicación al servidor (desarrollada por mí). Esta opción sigue siendo inviable en formato escalable por su elevado coste de instalación y mantenimiento.

- **Nivel bajo de validación:** Consiste en alojar la ubicación de los contenedores en la base de datos y validar únicamente que el usuario se encuentre cerca de un contenedor. Además, se debe incluir un límite de una actividad de reciclaje cada X horas (actualmente son 10) para evitar que se hagan trampas. Esta opción es viable pero lógicamente, por su bajo nivel de validación no es óptima. Igualmente, un usuario activo que realiza actividades continuadas además de reciclaje, verá que el porcentaje de puntos que gana por esta actividad es muy bajo en relación a lo que gana con las demás, por lo que no desmotivaría a los usuarios.

  Para esta opción además, se debe tener el dato de la ubicación de los contenedores, algo que no es fácil de encontrar. De hecho, al ser los contenedores propiedad y

responsabilidad de cada ayuntamiento, la mayoría de municipios ni si quiera tienen una base de datos de todos los contenedores. Desde Liight existe una funcionalidad llamada "mapea mi contenedor", disponible para los usuarios de nivel 10 o superior, que permite mandar ubicaciones de contenedores que el equipo de Liight posteriormente valida a través de Street View.

Capítulo 8. ANÁLISIS DE RESULTADOS

Tras dos años de desarrollo de lo que es ahora la aplicación de Liight, se puede decir que los resultados son satisfactorios.

Como se ha mencionado anteriormente, la metodología seguida es la de Lean-Startup, lo cual significa que el trabajo realizado siempre ha ido orientado a mejorar los resultados de la iteración anterior, y esto ha sido clave para el crecimiento y el éxito de la aplicación.

A día de hoy, la aplicación tiene más de 8000 descargas en Android, más de 900 usuarios recurrentes semanales y se estima que ha ayudado a reducir la cantidad de CO2 en la atmósfera en 20 toneladas. Además, como se puede observar en el gráfico de abajo, las métricas de retención están entre las de las 5000 mejores aplicaciones de Google Play, y el pasado mes de Junio, Liight fue la 9ª aplicación social más popular de Google Play.

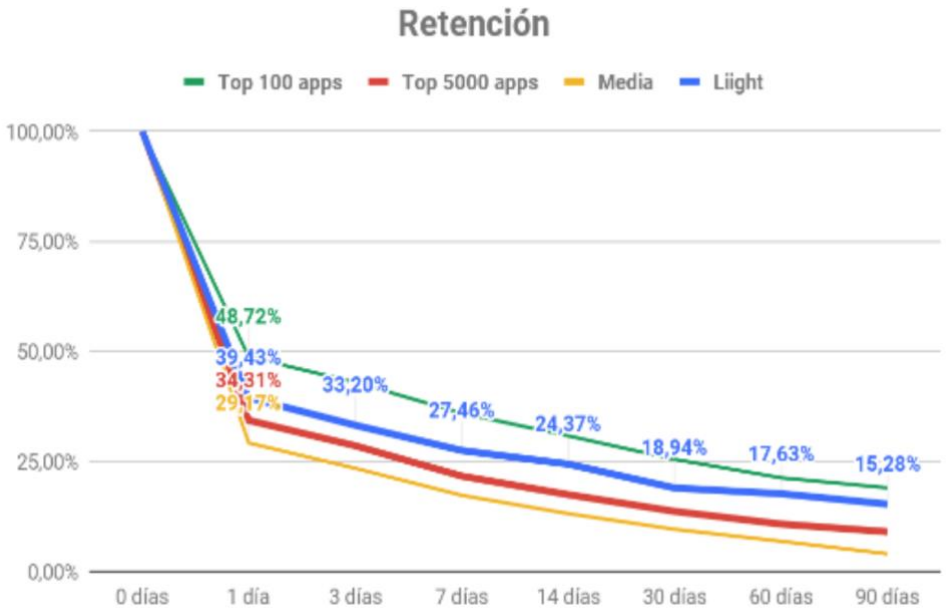

Figura 55 - Gráfica de retención de Liight

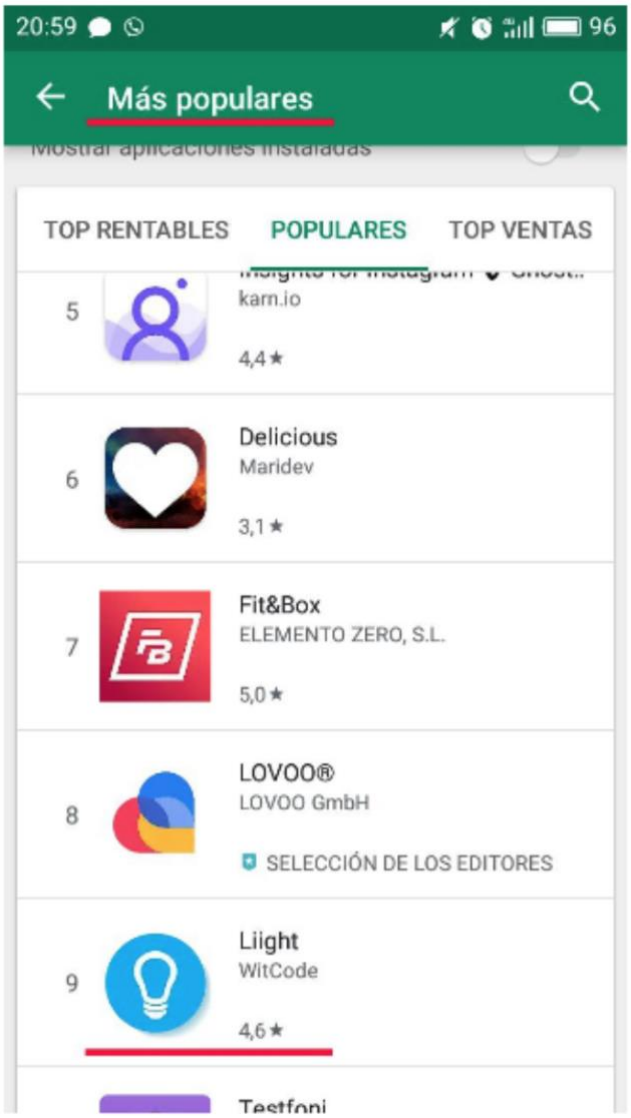

Figura 56 - Liight es la 9ª aplicación social más popular

Capítulo 9. CONCLUSIONES Y TRABAJOS FUTUROS

Esta aplicación, aunque se encuentra en una versión estable y aceptable de cara al mercado, realmente nunca dejará de cambiar y actualizarse. La lista de funcionalidades por hacer es todavía muy larga, y sin duda aparecerán muchas más durante el próximo año.

A día de hoy, el trabajo realizado es satisfactorio considerando las condiciones del equipo y los recursos financieros y humanos de los que se disponían.

A partir de septiembre, ya con inversión, se espera que el proyecto siga creciendo con un equipo profesionalizado trabajando full time.

En concreto, se espera que a partir de septiembre haya nuevas funcionalidades de gamificación como la de ligas y torneos, y se espera también inaugurar la opción de detección automática de actividades de cercanías. Además, también se espera cerrar acuerdos con una gran cantidad de empresas de movilidad para aumentar la utilidad de la aplicación y hacer de Liight un agregador de opciones de movilidad sostenibles.

A día de hoy existen relaciones con empresas como el BBVA, Santander, Ecoembes, Pascual, Sanitas, CocaCola, Everis, Ferrovial, la EMT y Sacyr, y se espera que con su apoyo, Liight consiga ser un referente nacional de movilidad sostenible en el próximo año.

Capítulo 10. BIBLIOGRAFÍA

(s.f.). Obtenido de https://www.xatakamovil.com/aplicaciones/el-mercado-de-aplicaciones-moviles-seguira-imbatible-se-espera-que-genere-110-mil-millones-en-2018

(s.f.). Obtenido de https://www.xatakamovil.com/aplicaciones/el-mercado-de-aplicaciones-moviles-seguira-imbatible-se-espera-que-genere-110-mil-millones-en-2018

(s.f.). Obtenido de https://www.campusmvp.es/recursos/post/frameworks-para-desarrollo-de-aplicaciones-moviles-hibridas.aspx

(s.f.). Obtenido de https://en.wikipedia.org/wiki/Kotlin_(programming_language)

(s.f.). Obtenido de https://www.howtogeek.com/189036/android-is-based-on-linux-but-what-does-that-mean/

(s.f.). Obtenido de https://www.87android.com/what-is-dalvik-virtual-machine-in-android/

(s.f.). Obtenido de https://developer.android.com/guide/components/fundamentals

(s.f.). Obtenido de https://material.io/design/

(s.f.). Obtenido de https://www.mindmeister.com/blog/ux-design-process-mind-maps/

ANEXO A – CÓDIGO DEL MAINACTIVITY

El siguiente código es una representación del código de Android. Muestra el comportamiento de la MainActivity, la pantalla y el contexto central de la aplicación:

```
package com.witcode.light.light.activities;

import android.Manifest;
import android.annotation.SuppressLint;
import android.app.Activity;
import android.app.PendingIntent;
import android.content.Context;
import android.content.Intent;
import android.content.IntentSender;
import android.content.ServiceConnection;
import android.content.SharedPreferences;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.location.Location;
import android.net.Uri;
import android.os.Bundle;
import android.os.Looper;
import android.preference.PreferenceManager;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;

import com.google.android.material.bottomnavigation.BottomNavigationView;
import com.google.android.material.navigation.NavigationView;
import com.google.android.material.snackbar.Snackbar;

import androidx.core.app.ActivityCompat;
import androidx.fragment.app.FragmentManager;
import androidx.fragment.app.FragmentTransaction;
import androidx.core.content.ContextCompat;

import android.util.Log;
import android.view.View;

import androidx.core.view.GravityCompat;
import androidx.drawerlayout.widget.DrawerLayout;
import androidx.appcompat.app.AppCompatActivity;
```

```
import android.view.Menu;
import android.view.MenuItem;
import android.widget.ProgressBar;
import android.widget.TextView;

import com.crashlytics.android.Crashlytics;
import com.google.android.gms.common.ConnectionResult;
import com.google.android.gms.common.api.ApiException;
import com.google.android.gms.common.api.GoogleApiClient;
import com.google.android.gms.common.api.ResolvableApiException;
import com.google.android.gms.fitness.Fitness;
import com.google.android.gms.location.FusedLocationProviderClient;
import com.google.android.gms.location.LocationCallback;
import com.google.android.gms.location.LocationRequest;
import com.google.android.gms.location.LocationResult;
import com.google.android.gms.location.LocationServices;
import com.google.android.gms.location.LocationSettingsRequest;
import com.google.android.gms.location.LocationSettingsResponse;
import com.google.android.gms.location.LocationSettingsStatusCodes;
import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.OnSuccessListener;
import com.google.android.gms.tasks.Task;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.iid.FirebaseInstanceId;
import com.google.firebase.iid.InstanceIdResult;
import com.squareup.picasso.Picasso;
import com.witcode.light.light.Analytics;
import com.witcode.light.light.R;
import com.witcode.light.light.Services.ActivityService;
import com.witcode.light.light.Services.ServiceBinder;
import com.witcode.light.light.Utils;
import com.witcode.light.light.backend.GetUser;
import com.witcode.light.light.backend.MyServerClass;
import com.witcode.light.light.backend.ValidateRecycle;
import com.witcode.light.light.backend.interfaces.OnUserArrivedListener;
import com.witcode.light.light.backend.UpdateToken;
import com.witcode.light.light.backend.UpdateUserData;
import com.witcode.light.light.backend.interfaces.OnTaskCompletedListener;
import com.witcode.light.light.domain.LocationListener;
import com.witcode.light.light.domain.MyFirebaseAnalyticsApi;
import com.witcode.light.light.domain.MySendAnalyticsApi;
import com.witcode.light.light.domain.User;
import com.witcode.light.light.domain.UserObject;
import com.witcode.light.light.fragments.AchievementFragment;
import com.witcode.light.light.fragments.AchievementUpNotificationFragment;
import com.witcode.light.light.fragments.ActivitiesHistoryFragment;
import com.witcode.light.light.fragments.ActivityFragment;
import com.witcode.light.light.fragments.AdminFragment;
import com.witcode.light.light.fragments.AutomaticDetectionFragment;
```

```
import com.witcode.light.light.fragments.ChallengeFragment;
import com.witcode.light.light.fragments.DailyActivityNotificationFragment;
import com.witcode.light.light.fragments.EditProfileFragment;
import com.witcode.light.light.fragments.EmailSentDialogFragment;
import com.witcode.light.light.fragments.EndActivityFragment;
import com.witcode.light.light.fragments.GoogleFitFragment;
import com.witcode.light.light.fragments.GoogleFitReportDialogFragment;
import com.witcode.light.light.fragments.HomeFragment;
import com.witcode.light.light.fragments.LevelUpNotificationFragment;
import com.witcode.light.light.fragments.MapContainerFragment;
import com.witcode.light.light.fragments.MarketDetailFragment;
import com.witcode.light.light.fragments.MarketFragment;
import com.witcode.light.light.fragments.NewFragment;
import com.witcode.light.light.fragments.ObjectDialogFragment;
import com.witcode.light.light.fragments.ObjectsFragment;
import com.witcode.light.light.fragments.ObjectsMarketFragment;
import com.witcode.light.light.fragments.PrivacyFragment;
import com.witcode.light.light.fragments.ProfileFragment;
import com.witcode.light.light.fragments.RankingFragment;
import com.witcode.light.light.fragments.ReportFragment;
import com.witcode.light.light.fragments.SettingsFragment;
import com.witcode.light.light.fragments.StatisticsFragment;
import com.witcode.light.light.fragments.WebViewDialogFragment;
import com.witcode.light.light.fragments.WebViewFragment;

import java.util.Timer;
import java.util.TimerTask;
import java.util.concurrent.CopyOnWriteArrayList;

import de.hdodenhof.circleimageview.CircleImageView;
import io.fabric.sdk.android.Fabric;

public class MainActivity extends AppCompatActivity
        implements GoogleApiClient.ConnectionCallbacks,
        GoogleApiClient.OnConnectionFailedListener,
com.google.android.gms.location.LocationListener,
BottomNavigationView.OnNavigationItemSelectedListener {
    private static final int REQUEST_CHECK_SETTINGS = 5;

    private FragmentTransaction fragmentTransaction;
    private TextView tvLights;
    private Context mContext = this;
    private View header;
    public boolean serviceBound = false;
    private String mCurrentFragment = "";
    private ServiceBinder mServiceBinder = null;
    private ActivityService mActivityService = null;
    private CopyOnWriteArrayList<LocationListener> mLocationListeners = new
CopyOnWriteArrayList<>();
```

```
    private long timeIn;
    private Timer Timer;
    private GoogleApiClient mGoogleApiClient;
    private User user;
    private ServiceConnection mActivityServiceConnection;
    private PendingIntent pendingIntent;
    private MenuItem feed, activity, ranking, profile, admin;
    public final static int
MY_PERMISSIONS_REQUEST_LOCATION_FINE_ACTIVITY_FRAGMENT = 1;
    public final static int MY_PERMISSIONS_REQUEST_LOCATION_FINE_NO_ACTION = 2;
    public final static int MY_PERMISSIONS_REQUEST_LOCATION_FINE_START_FUSED = 3;
    public final static int MY_PERMISSIONS_REQUEST_CAMERA = 4;
    private static final int MY_PERMISSIONS_REQUEST_GOOGLE_FIT = 5;
    private TextView feed_unread_count, profile_unread_count,
ranking_activity_count, market_unread_count;
    private Menu menu;
    LocationSettingsRequest.Builder locSetRequestBuilder;
    private OnTaskCompletedListener locationSettingsListener;
    private boolean searchingLocationSettings = false;
    private ActivityFragment mActivityFragment;

    private SharedPreferences shPref;
    public MyFirebaseAnalyticsApi myFirebaseAnalyticsApi;
    public MySendAnalyticsApi mySendAnalyticsApi;
    public static final String TAG = "MainActivity";
    private GoogleFitFragment googleFitFragment;
    private FusedLocationProviderClient mFusedLocationClient;

    BottomNavigationView bottomNavigationView;


    private LocationCallback locationCallback = new LocationCallback() {
        @Override
        public void onLocationResult(LocationResult locationResult) {
            onLocationChanged(locationResult.getLastLocation());
        }
    };
    private Activity mActivity = this;

    public static boolean IN_FOREGROUND = false;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Fabric.with(this, new Crashlytics());
```

```
        mFusedLocationClient =
LocationServices.getFusedLocationProviderClient(this);

        updateLocation();

        mySendAnalyticsApi = new MySendAnalyticsApi(mContext);
        Bundle paramsAnalytics = new Bundle();
        paramsAnalytics.putString("mauri", "prueba");
        mySendAnalyticsApi.sendEvent("MainActivity", paramsAnalytics);

        if (FirebaseAuth.getInstance() != null) {
            if (FirebaseAuth.getInstance().getUid() != null) {

Crashlytics.setUserIdentifier(FirebaseAuth.getInstance().getUid());
            }

            if (FirebaseAuth.getInstance().getCurrentUser() != null) {
                if (FirebaseAuth.getInstance().getCurrentUser().getEmail() !=
null) {

Crashlytics.setUserEmail(FirebaseAuth.getInstance().getCurrentUser().getEmail());
                }
                if (FirebaseAuth.getInstance().getCurrentUser().getDisplayName()
!= null) {

Crashlytics.setUserEmail(FirebaseAuth.getInstance().getCurrentUser().getDisplayNa
me());
                }
            }

        }

        if (mGoogleApiClient == null) {
            this.mGoogleApiClient = new GoogleApiClient.Builder(this)
                    .enableAutoManage(this, new
GoogleApiClient.OnConnectionFailedListener() {
                        @Override
                        public void onConnectionFailed(@NonNull ConnectionResult
connectionResult) {
                            Log.wtf(TAG, "connection error google api: " +
connectionResult.getErrorMessage());
                        }
                    })
                    .addApi(LocationServices.API)
                    .addApi(Fitness.RECORDING_API)
                    .addApi(Fitness.SESSIONS_API)
                    .addApi(Fitness.HISTORY_API)
                    .build();
        }
```

```
        mGoogleApiClient.connect();

        //Fabric.with(this, new Crashlytics());

        bottomNavigationView = (BottomNavigationView)
findViewById(R.id.bottom_navigation);
        bottomNavigationView.setOnNavigationItemSelectedListener(this);


        if (getIntent() != null) {
            handleNewIntent(getIntent(), true);
        } else {
            GotoStartActivityFragment(false);
        }


        FirebaseInstanceId.getInstance().getInstanceId()
                .addOnCompleteListener(new OnCompleteListener<InstanceIdResult>()
{
                    @Override
                    public void onComplete(@NonNull Task<InstanceIdResult> task)
{
                        if (!task.isSuccessful()) {
                            Log.w(TAG, "getInstanceId failed",
task.getException());
                            return;
                        }

                        if (task.getResult() != null) {
                            new UpdateToken(mContext,
task.getResult().getToken(), new OnTaskCompletedListener() {
                                @Override
                                public void OnComplete(String result, int
resultCode, int resultType) {

                                }
                            }).execute();
                        }

                    }
                });

        Log.wtf("mytag", "starting activity fragment");

        getSupportFragmentManager().addOnBackStackChangedListener(new
FragmentManager.OnBackStackChangedListener() {
            @Override
            public void onBackStackChanged() {
```

```
                for (int i = 0; i <
getSupportFragmentManager().getBackStackEntryCount(); i++) {
                    Log.wtf("mytag", "backstack entry n" + i + ": " +
getSupportFragmentManager().getBackStackEntryAt(i) + ", name: " +
getSupportFragmentManager().getBackStackEntryAt(i).getName() + ", bctitle: " +
getSupportFragmentManager().getBackStackEntryAt(i).getBreadCrumbTitle());
                }

                if (getSupportFragmentManager().getBackStackEntryCount() > 0) {


                    switch
(getSupportFragmentManager().getBackStackEntryAt(getSupportFragmentManager().getB
ackStackEntryCount() - 1).getName()) {

                        case "activity":
                            mCurrentFragment = "activity";

bottomNavigationView.getMenu().findItem(R.id.nav_start).setChecked(true);
                            Log.wtf("mytag", "calling activity");
                            break;
                        case "home":
                            mCurrentFragment = "home";

bottomNavigationView.getMenu().findItem(R.id.nav_feed).setChecked(true);
                            Log.wtf("mytag", "calling home");
                            break;
                        case "ranking":
                            mCurrentFragment = "ranking";

bottomNavigationView.getMenu().findItem(R.id.nav_ranking).setChecked(true);
                            Log.wtf("mytag", "calling ranking");
                            break;
                        case "market":
                            mCurrentFragment = "market";

bottomNavigationView.getMenu().findItem(R.id.nav_market).setChecked(true);
                            Log.wtf("mytag", "calling market");
                            break;
                        case "profile":
                            mCurrentFragment = "profile";

bottomNavigationView.getMenu().findItem(R.id.nav_profile).setChecked(true);
                            Log.wtf("mytag", "calling profile");
                            break;
                        case "webview":
                            mCurrentFragment = "webview";
                            break;
                        case "report":
```

```
                            mCurrentFragment = "report";
                            break;
                    }
                } else {
                    Log.wtf("mytag", "backstack is 0");
                    finish();
                }

                switch (mCurrentFragment) {
                    case "challenge_detail":
                    case "market_detail":
                    case "report":
                    case "webview":
                    case "automatic_detection":
                    case "edit_profile":
                        bottomNavigationView.setVisibility(View.GONE);
                        break;
                    default:
                        bottomNavigationView.setVisibility(View.VISIBLE);
                        break;
                }

            }
        });
        // ATTENTION: This was auto-generated to handle app links.


    }

    public void processLocationSettings(@Nullable final OnTaskCompletedListener
listener) {
        locationSettingsListener = listener;

        if (!searchingLocationSettings) {
            if (locSetRequestBuilder == null)
                locSetRequestBuilder = new LocationSettingsRequest.Builder()
                        .addLocationRequest(new
LocationRequest().setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY))
                        .addLocationRequest(new
LocationRequest().setPriority(LocationRequest.PRIORITY_BALANCED_POWER_ACCURACY));

            Task<LocationSettingsResponse> result =

LocationServices.getSettingsClient(this).checkLocationSettings(locSetRequestBuild
er.build());

            result.addOnCompleteListener(new
OnCompleteListener<LocationSettingsResponse>() {
                @Override
```

```
                public void onComplete(Task<LocationSettingsResponse> task) {
                    try {
                        LocationSettingsResponse response =
task.getResult(ApiException.class);
                        // All location settings are satisfied. The client can
initialize location
                        // requests here.
                        if (locationSettingsListener != null)
                            locationSettingsListener.OnComplete("already on", 0,
0);

                    } catch (ApiException exception) {
                        switch (exception.getStatusCode()) {
                            case LocationSettingsStatusCodes.RESOLUTION_REQUIRED:
                                // Location settings are not satisfied. But could
be fixed by showing the
                                // user a dialog.
                                try {
                                    // Cast to a resolvable exception.
                                    ResolvableApiException resolvable =
(ResolvableApiException) exception;
                                    // Show the dialog by calling
startResolutionForResult(),
                                    // and check the result in
onActivityResult().

                                    searchingLocationSettings = true;

                                    Log.wtf("mytag", "oh bullocks");
                                    resolvable.startResolutionForResult(
                                            (Activity) mContext,
                                            REQUEST_CHECK_SETTINGS);


                                } catch (IntentSender.SendIntentException e) {
                                    // Ignore the error.
                                } catch (ClassCastException e) {
                                    // Ignore, should be an impossible error.
                                } catch (Exception e) {
                                    // Ignore, should be an impossible error.
                                }
                                break;
                            case
LocationSettingsStatusCodes.SETTINGS_CHANGE_UNAVAILABLE:
                                // Location settings are not satisfied. However,
we have no way to fix the
                                // settings so we won't show the dialog.
                                Analytics.send(mContext,
"location_settings_unavailable: ActivityService");
```

```
                                    Log.wtf("mytag", "location_settings_unavailable:
ActivityService");
                                    break;
                            }
                        }
                    }
                });
            }

        }


        @Override
        protected void onActivityResult(int requestCode, int resultCode, Intent data)
{

            switch (requestCode) {
                case REQUEST_CHECK_SETTINGS:
                    searchingLocationSettings = false;
                    switch (resultCode) {
                        case Activity.RESULT_OK:

                            if (locationSettingsListener != null)
                                locationSettingsListener.OnComplete("just_turned", 0,
0);
                            Log.wtf("mytag", "All okk! :)");

                            break;
                        case Activity.RESULT_CANCELED:
                            Log.wtf("mytag", "User canceled... :(");
                            break;
                        default:
                            break;
                    }
                    break;
                default:

                    super.onActivityResult(requestCode, resultCode, data);
                    break;
            }
        }


        @SuppressWarnings("StatementWithEmptyBody")
        @Override
        public boolean onNavigationItemSelected(MenuItem item) {
            // Handle navigation view item clicks here.
            int id = item.getItemId();
```

```
        if (id == R.id.nav_start) {
            GotoStartActivityFragment();
        } else if (id == R.id.nav_market) {
            GoToFragment("market");
        } else if (id == R.id.nav_feed) {
            Log.wtf("mytag", "to home7;");
            GoToFragment("home");
        } else if (id == R.id.nav_ranking) {
            GoToFragment("ranking");


        } else if (id == R.id.nav_profile) {
            GoToProfile(user);

        }


        return true;
    }

    public void GoToFragment(String fragment, Bundle extras, boolean toBackStack)
{

        //TODO set drawer checked false
        switch (fragment) {

            case "home":
                HomeFragment mFragment = new HomeFragment();
                mFragment.setArguments(extras);
                fragmentTransaction =
getSupportFragmentManager().beginTransaction();
                if (toBackStack)
                    fragmentTransaction.addToBackStack("home");
                fragmentTransaction.replace(R.id.container, mFragment);
                fragmentTransaction.commitAllowingStateLoss();
                mCurrentFragment = "home";

bottomNavigationView.getMenu().findItem(R.id.nav_feed).setChecked(true);
                Log.wtf("mytag", "checking home in menu");
                break;
            case "start_activity":
                GotoStartActivityFragment();
                break;
            case "ranking":
                RankingFragment mRankingFragment = new RankingFragment();
                mRankingFragment.setArguments(extras);
                fragmentTransaction =
getSupportFragmentManager().beginTransaction();
                if (toBackStack)
```

```
                    fragmentTransaction.addToBackStack("ranking");
                fragmentTransaction.replace(R.id.container, mRankingFragment);
                fragmentTransaction.commitAllowingStateLoss();
                mCurrentFragment = "ranking";

bottomNavigationView.getMenu().findItem(R.id.nav_ranking).setChecked(true);
                break;

            case "market":
                MarketFragment mMarketFragment = new MarketFragment();
                mMarketFragment.setArguments(extras);
                fragmentTransaction =
getSupportFragmentManager().beginTransaction();
                if (toBackStack)
                    fragmentTransaction.addToBackStack("market");
                fragmentTransaction.replace(R.id.container, mMarketFragment);
                fragmentTransaction.commitAllowingStateLoss();
                mCurrentFragment = "market";

bottomNavigationView.getMenu().findItem(R.id.nav_market).setChecked(true);
                break;

            case "map_container":
                MapContainerFragment mapContainerFragment = new
MapContainerFragment();

                fragmentTransaction =
getSupportFragmentManager().beginTransaction();
                if (toBackStack)
                    fragmentTransaction.addToBackStack("map_container");
                fragmentTransaction.replace(R.id.container,
mapContainerFragment);
                fragmentTransaction.commitAllowingStateLoss();
                mCurrentFragment = "map_container";
                break;

            case "admin":
                AdminFragment mAdminFragment = new AdminFragment();
                mAdminFragment.setArguments(extras);
                fragmentTransaction =
getSupportFragmentManager().beginTransaction();
                if (toBackStack)
                    fragmentTransaction.addToBackStack("admin");
                fragmentTransaction.replace(R.id.container, mAdminFragment);
                fragmentTransaction.commitAllowingStateLoss();
                mCurrentFragment = "admin";
                Log.wtf("mytag", "checking admin in menu");
                break;
```

```
            case "end_activity":

                EndActivityFragment endActivityFragment =
EndActivityFragment.getInstance();
                fragmentTransaction =
getSupportFragmentManager().beginTransaction();
                if (toBackStack)
                    fragmentTransaction.addToBackStack("end_activity");
                fragmentTransaction.replace(R.id.container, endActivityFragment);
                fragmentTransaction.commitAllowingStateLoss();
                mCurrentFragment = "end_activity";

                break;

            case "profile":
                ProfileFragment mProfileFragment = new ProfileFragment();
                mProfileFragment.setArguments(extras);
                fragmentTransaction =
getSupportFragmentManager().beginTransaction();
                if (toBackStack)
                    fragmentTransaction.addToBackStack("profile");
                fragmentTransaction.replace(R.id.container, mProfileFragment);
                fragmentTransaction.commitAllowingStateLoss();
                mCurrentFragment = "profile";

bottomNavigationView.getMenu().findItem(R.id.nav_profile).setChecked(true);
                break;
        }
    }

    public void GoToFragment(String fragment) {
        GoToFragment(fragment, new Bundle(), true);
    }

    public void GoToFragment(String fragment, boolean toBackStack) {
        GoToFragment(fragment, new Bundle(), toBackStack);
    }

    public void GoToMarketDetail(String marketId) {
        GoToMarketDetail(marketId, true);
    }

    public void GoToMarketDetail(String marketId, boolean addToBackStack) {
        MarketDetailFragment mMarketFragment =
MarketDetailFragment.getInstance(marketId);
        fragmentTransaction = getSupportFragmentManager().beginTransaction();
        if (addToBackStack)
            fragmentTransaction.addToBackStack("market_detail");
        fragmentTransaction.replace(R.id.container, mMarketFragment);
```

```
        fragmentTransaction.commitAllowingStateLoss();
        mCurrentFragment = "market_detail";
    }

    public void GoToProfile(User user) {
        GoToProfile(user, true);
    }

    public void GoToProfile(User user, boolean addToBackStack) {
        ProfileFragment mProfileFragment = ProfileFragment.getInstance(user);
        fragmentTransaction = getSupportFragmentManager().beginTransaction();
        if (addToBackStack)
            fragmentTransaction.addToBackStack("profile");
        fragmentTransaction.replace(R.id.container, mProfileFragment);
        fragmentTransaction.commitAllowingStateLoss();
        //TODO set checked true
        mCurrentFragment = "profile";
    }


    public void GoToSettings(User user) {
        SettingsFragment mSettingsFragment = SettingsFragment.getInstance(user);
        fragmentTransaction = getSupportFragmentManager().beginTransaction();
        fragmentTransaction.addToBackStack("settings");
        fragmentTransaction.replace(R.id.container, mSettingsFragment);
        fragmentTransaction.commitAllowingStateLoss();
        mCurrentFragment = "settings";
    }

    public void GoToEditProfile(User user) {
        EditProfileFragment mEditProfileFragment =
EditProfileFragment.getInstance(user);
        fragmentTransaction = getSupportFragmentManager().beginTransaction();
        fragmentTransaction.addToBackStack("edit_profile");
        fragmentTransaction.replace(R.id.container, mEditProfileFragment);
        fragmentTransaction.commitAllowingStateLoss();
        mCurrentFragment = "edit_profile";
    }

    public void GoToAutomaticDetection(@Nullable User user) {

        if (!mCurrentFragment.equals("automatic_detection")) {
            AutomaticDetectionFragment automaticDetectionFragment;
            if (user == null)
                automaticDetectionFragment =
AutomaticDetectionFragment.getInstance();
            else
                automaticDetectionFragment =
AutomaticDetectionFragment.getInstance(user);
```

```
            fragmentTransaction = getSupportFragmentManager().beginTransaction();
            fragmentTransaction.addToBackStack("automatic_detection");
            fragmentTransaction.replace(R.id.container,
automaticDetectionFragment);
            fragmentTransaction.commitAllowingStateLoss();
            mCurrentFragment = "automatic_detection";
        }

    }

    public void GoToGoogleFitFragment(User user) {
        GoogleFitFragment mGoogleFitFragment =
GoogleFitFragment.getInstance(user);
        fragmentTransaction = getSupportFragmentManager().beginTransaction();
        fragmentTransaction.addToBackStack("googlefit");
        fragmentTransaction.replace(R.id.container, mGoogleFitFragment);
        fragmentTransaction.commitAllowingStateLoss();
        mCurrentFragment = "googlefit";
    }

    public void GoToPrivacy(User user) {
        PrivacyFragment mPrivacyFragment = PrivacyFragment.getInstance(user);
        fragmentTransaction = getSupportFragmentManager().beginTransaction();
        fragmentTransaction.addToBackStack("privacy");
        fragmentTransaction.replace(R.id.container, mPrivacyFragment);
        fragmentTransaction.commitAllowingStateLoss();
        mCurrentFragment = "privacy";
    }


    public void GoToChallengeDetail(String challengeId, boolean addToBackStack) {
        ChallengeFragment mChallengeFragment =
ChallengeFragment.getInstance(challengeId);
        fragmentTransaction = getSupportFragmentManager().beginTransaction();
        if (addToBackStack)
            fragmentTransaction.addToBackStack("challenge_detail");
        fragmentTransaction.replace(R.id.container, mChallengeFragment);
        fragmentTransaction.commitAllowingStateLoss();
        mCurrentFragment = "challenge_detail";
    }

    public void GoToStatisticsFragment(User user) {
        StatisticsFragment mStatisticsFragment =
StatisticsFragment.getInstance(user);
        fragmentTransaction = getSupportFragmentManager().beginTransaction();
        fragmentTransaction.addToBackStack("statistics");
        fragmentTransaction.replace(R.id.container, mStatisticsFragment);
        fragmentTransaction.commitAllowingStateLoss();
```

```
        mCurrentFragment = "statistics";
    }

    public void GoToActivitiesHistory(User user) {
        ActivitiesHistoryFragment mActivitiesHistoryFragment =
ActivitiesHistoryFragment.getInstance(user);
        fragmentTransaction = getSupportFragmentManager().beginTransaction();
        fragmentTransaction.addToBackStack("activities_history");
        fragmentTransaction.replace(R.id.container, mActivitiesHistoryFragment);
        fragmentTransaction.commitAllowingStateLoss();
        mCurrentFragment = "activities_history";
    }


    public void GoToObjectsFragment(User user) {
        ObjectsFragment mObjectsFragment = ObjectsFragment.getInstance(user);
        fragmentTransaction = getSupportFragmentManager().beginTransaction();
        fragmentTransaction.addToBackStack("objects");
        fragmentTransaction.replace(R.id.container, mObjectsFragment);
        fragmentTransaction.commitAllowingStateLoss();
        mCurrentFragment = "objects";
    }

    public void GoToAchievementsFragment(User user) {
        AchievementFragment mAchievementsFragments =
AchievementFragment.getInstance(user);
        fragmentTransaction = getSupportFragmentManager().beginTransaction();
        fragmentTransaction.addToBackStack("achievements");
        fragmentTransaction.replace(R.id.container, mAchievementsFragments);
        fragmentTransaction.commitAllowingStateLoss();
        mCurrentFragment = "achievements";
    }

    public void GoToNew(String newId) {
        NewFragment mNewFragment = NewFragment.getInstance(newId);
        fragmentTransaction = getSupportFragmentManager().beginTransaction();
        fragmentTransaction.addToBackStack("new");
        fragmentTransaction.replace(R.id.container, mNewFragment);
        fragmentTransaction.commitAllowingStateLoss();
        mCurrentFragment = "new";
    }

    public void GoToObjectsMarketFragment() {
        ObjectsMarketFragment objectsMarketFragment =
ObjectsMarketFragment.getInstance();
        fragmentTransaction = getSupportFragmentManager().beginTransaction();
        fragmentTransaction.addToBackStack("objects_market");
        fragmentTransaction.replace(R.id.container, objectsMarketFragment);
        fragmentTransaction.commitAllowingStateLoss();
```

```
        mCurrentFragment = "objects_market";
    }


    public void GoToWebview(String url) {
        WebViewFragment mWebviewFragment = WebViewFragment.getInstance(url);
        fragmentTransaction = getSupportFragmentManager().beginTransaction();
        fragmentTransaction.addToBackStack("webview");
        fragmentTransaction.replace(R.id.container, mWebviewFragment);
        fragmentTransaction.commitAllowingStateLoss();
        mCurrentFragment = "webview";
    }

    public void GoToReportFragment() {
        ReportFragment mReportFragment = ReportFragment.getInstance();
        fragmentTransaction = getSupportFragmentManager().beginTransaction();
        fragmentTransaction.addToBackStack("report");
        fragmentTransaction.replace(R.id.container, mReportFragment);
        fragmentTransaction.commitAllowingStateLoss();
        mCurrentFragment = "report";
    }

    public void GotoStartActivityFragment(boolean addToBackStack) {


        if (ContextCompat.checkSelfPermission(this,
                Manifest.permission.ACCESS_FINE_LOCATION)
                != PackageManager.PERMISSION_GRANTED) {

            ActivityCompat.requestPermissions(this,
                    new String[]{Manifest.permission.ACCESS_FINE_LOCATION},
                    MY_PERMISSIONS_REQUEST_LOCATION_FINE_ACTIVITY_FRAGMENT);
        } else {


            Log.wtf("tagg", "gotostartactivityfragment");
            ActivityFragment mStartFragment = new ActivityFragment();
            fragmentTransaction = getSupportFragmentManager().beginTransaction();
            fragmentTransaction.replace(R.id.container, mStartFragment);
            if (true)
                fragmentTransaction.addToBackStack("activity");
            fragmentTransaction.commitAllowingStateLoss();
            mCurrentFragment = "activity";



bottomNavigationView.getMenu().findItem(R.id.nav_start).setChecked(true);
        }
    }
```

```
    public void GotoStartActivityFragment() {

        GotoStartActivityFragment(true);
    }


    public void showObjectDialog(UserObject userObject) {
        FragmentManager fm = this.getSupportFragmentManager();
        ObjectDialogFragment objectDialogFragment =
ObjectDialogFragment.newInstance("object", userObject);
        objectDialogFragment.show(fm, "object_dialog_fragment");
    }


    public void showMarketObjectDialog(UserObject userObject) {
        FragmentManager fm = this.getSupportFragmentManager();
        //BUTTON TYPE 2 ES EL DE COMRPAR
        ObjectDialogFragment objectDialogFragment =
ObjectDialogFragment.newMarketInstance("object", userObject);
        objectDialogFragment.show(fm, "object_dialog_fragment");
    }

    public void showGoogleFitDialog(int steps, int walkDist, int bikeDist, double
liights, double exp, double co2) {
        FragmentManager fm = this.getSupportFragmentManager();
        GoogleFitReportDialogFragment googleFitReportDialogFragment =
GoogleFitReportDialogFragment.newInstance(steps, walkDist, bikeDist, liights,
exp, co2);
        googleFitReportDialogFragment.show(fm,
"google_fit_report_dialog_fragment");

    }

    public void showWebViewDialog(String tittle, String url, String buttonText,
String action, String actionExtra) {
        FragmentManager fm = this.getSupportFragmentManager();
        WebViewDialogFragment webViewDialogFragment =
WebViewDialogFragment.newInstance(tittle, user, url, buttonText, action,
actionExtra);
        webViewDialogFragment.show(fm, "web_view_dialog_fragment");

    }

    public void showLevelUpNotification(int level, int liightsReward) {
        FragmentManager fm = this.getSupportFragmentManager();
        LevelUpNotificationFragment levelUpNotificationFragment =
LevelUpNotificationFragment.newInstance("levelUp", level, liightsReward);
        levelUpNotificationFragment.show(fm, "level_up_dialog_fragment");
```

```
    }

    public void showEmailSentNotification(User user) {
        FragmentManager fm = this.getSupportFragmentManager();
        EmailSentDialogFragment emailSentDialogFragment =
EmailSentDialogFragment.newInstance(user);
        emailSentDialogFragment.show(fm, "email_sent_dialog_fragment");
    }

    public void showAchievementUpNotification(String description, int level, int
liightsReward, int expReward) {
        FragmentManager fm = this.getSupportFragmentManager();
        AchievementUpNotificationFragment achievementUpNotificationFragment =
AchievementUpNotificationFragment.newInstance(description, level, liightsReward,
expReward);
        achievementUpNotificationFragment.show(fm,
"achievement_up_dialog_fragment");
    }

    public void showDailyActivityNotificationFragment(User user) {
        FragmentManager fm = this.getSupportFragmentManager();
        DailyActivityNotificationFragment dailyActivityNotificationFragment =
DailyActivityNotificationFragment.newInstance("levelUp", user);
        dailyActivityNotificationFragment.show(fm,
"daily_activity_dialog_fragment");
    }


    public void handleNewIntent(Intent intent, boolean oncreate) {
        if (intent.getExtras() != null &&
intent.getExtras().getBoolean("isnotif", false)) {
            Analytics.send(getApplicationContext(), "clicked_push_notification",
intent.getExtras().getString("id"));
        }

        Log.wtf("mytag", "intent: " + intent.toString());

        if (intent.getAction() != null) {


            String action = "";
            String actionExtra = null;

            if (intent.getDataString() != null &&
intent.getDataString().contains("deeplink")) {
                Uri uri = Uri.parse(intent.getDataString());
                if (uri.getQueryParameter("action") != null) {
                    action = uri.getQueryParameter("action");
                    actionExtra = uri.getQueryParameter("action_extra");
```

```
                }

            } else {
                action = intent.getAction();
                actionExtra = intent.getExtras() != null ?
intent.getExtras().getString("extra") : null;

            }


            switch (action) {
                case "end_activity":
                    GoToFragment("end_activity");
                    break;

                case "start_activity":
                case "started_activity":
                    if (!mCurrentFragment.equals("activity"))
                        GotoStartActivityFragment();
                    break;
                case "ranking":
                    GoToFragment("ranking", false);
                    break;
                case "profile":
                    GoToProfile(null, false);
                    break;
                case "market":
                    GoToFragment("market", false);
                    break;
                case "market_item_action":
                    GoToMarketDetail(actionExtra, false);
                    break;
                case "challenge_item_action":
                    GoToChallengeDetail(actionExtra, false);
                    break;
                case "feed_action":
                    Log.wtf("mytag", "to home4;");
                    GoToFragment("home", false);
                    break;
                case "url_action":
                    Intent i = new Intent(Intent.ACTION_VIEW);
                    i.setData(Uri.parse(actionExtra));
                    startActivity(i);

                    break;

                case "update_app":
                    final String appPackageName = mContext.getPackageName(); //
getPackageName() from Context or Activity object
```

```
                    try {
                        mContext.startActivity(new Intent(Intent.ACTION_VIEW,
Uri.parse("market://details?id=" + appPackageName)));
                    } catch (android.content.ActivityNotFoundException anfe) {
                        mContext.startActivity(new Intent(Intent.ACTION_VIEW,
Uri.parse("https://play.google.com/store/apps/details?id=" + appPackageName)));
                    }

                    break;

                case "webview_action":
                    if (actionExtra != null) {
                        GoToWebview(actionExtra);
                    }

                    break;


                case "action_stop":
                case "too_fast":
                case ActivityService.STOP_ACTIVITY:
                    Utils.startCommandActivityService(this, null, action);
                    break;

                case "ACTION_GET_USER":
                    GetUser.with(this).send();
                    break;

                default:
                    Log.wtf("mytag", "to home2;");
                    GotoStartActivityFragment();

                    break;
            }
        } else if (oncreate) {
            GotoStartActivityFragment(false);
        }

    }

    @Override
    protected void onNewIntent(Intent intent) {
        Log.wtf("tagg", "new intent: " + intent);

        handleNewIntent(intent, false);
        super.onNewIntent(intent);
    }
```

```
    @Override
    public void onRequestPermissionsResult(int requestCode,
                                           String permissions[], int[]
grantResults) {
        switch (requestCode) {
            case MY_PERMISSIONS_REQUEST_LOCATION_FINE_NO_ACTION:
            case MY_PERMISSIONS_REQUEST_LOCATION_FINE_ACTIVITY_FRAGMENT:
            case MY_PERMISSIONS_REQUEST_LOCATION_FINE_START_FUSED:
                // If request is cancelled, the result arrays are empty.
                if (grantResults.length > 0
                        && grantResults[0] == PackageManager.PERMISSION_GRANTED)
{

                    // permission was granted, yay! Do the
                    // location-related task you need to do.

                    if (requestCode ==
MY_PERMISSIONS_REQUEST_LOCATION_FINE_ACTIVITY_FRAGMENT) {
                        GotoStartActivityFragment();
                    } else if (requestCode ==
MY_PERMISSIONS_REQUEST_LOCATION_FINE_START_FUSED) {
                        try {

mFusedLocationClient.requestLocationUpdates(Utils.createLocationRequest(LocationR
equest.PRIORITY_HIGH_ACCURACY), locationCallback, null);
                        } catch (SecurityException e) {
                            e.printStackTrace();
                        }

                    }

                } else {

                    //TODO show Snackbar

                    // permission denied, boo! Disable the
                    // functionality that depends on this permission.
                }
                break;

            case MY_PERMISSIONS_REQUEST_CAMERA: {
                // If request is cancelled, the result arrays are empty.
                if (grantResults.length > 0
                        && grantResults[0] == PackageManager.PERMISSION_GRANTED
&& mActivityFragment != null) {

                    // permission was granted, yay!
```

```
                    mActivityFragment.showQRScreen();
                } else {
                    if (mActivityFragment != null) {

                    }

                    // permission denied, boo! Disable the
                    // functionality that depends on this permission.
                }
                break;
            }


            // other 'case' lines to check for other
            // permissions this app might request
        }
    }

    public void logOut() {
        FirebaseAuth.getInstance().signOut();
        Intent intent = new Intent(this, LoginActivity.class);
        intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
        startActivity(intent);

        finish();
    }

    @SuppressLint("MissingPermission")
    public void getLastKnownLocation(OnSuccessListener<Location> listener) {

        try {

mFusedLocationClient.getLastLocation().addOnSuccessListener(listener);
        } catch (Exception e) {
            e.printStackTrace();
        }

    }


    private class MyTimerTask extends TimerTask {
        @Override
        public void run() {


            SharedPreferences sharedPref = ((MainActivity)
mContext).getPreferences(Context.MODE_PRIVATE);
            int canAccess = sharedPref.getInt("can_access", -1);
```

```
            if (canAccess == 0) {
                FirebaseAuth.getInstance().signOut();
                Intent intent = new Intent(mContext, UpdateActivity.class);
                intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
                mContext.startActivity(intent);
                finish();
            }

            updateLocation();

        }
    }


    public void addLocationListener(LocationListener listener) {
        listener.setActivity(this);
        if (!mLocationListeners.contains(listener))
            mLocationListeners.add(listener);

        if (ContextCompat.checkSelfPermission(this,
                Manifest.permission.ACCESS_FINE_LOCATION)
                != PackageManager.PERMISSION_GRANTED) {

            ActivityCompat.requestPermissions(this,
                    new String[]{Manifest.permission.ACCESS_FINE_LOCATION},
                    MY_PERMISSIONS_REQUEST_LOCATION_FINE_START_FUSED);
        } else {
            try {

mFusedLocationClient.requestLocationUpdates(Utils.createLocationRequest(LocationR
equest.PRIORITY_HIGH_ACCURACY),
                        locationCallback, Looper.getMainLooper());

                Log.wtf("mytag", "creating location request");


            } catch (SecurityException e) {
                e.printStackTrace();
                listener.OnError(e.toString());
            }
        }


    }

    public void removeLocationListener(LocationListener listener) {
        mLocationListeners.remove(listener);

        Log.wtf("mytag", "locationListener removed: " + listener.getId());
```

```
        if (mLocationListeners.size() == 0) {
            mFusedLocationClient.removeLocationUpdates(locationCallback);
        }
    }

    @Override
    public void onLocationChanged(Location location) {

        updateRemoteLocationData();

        Log.wtf("mytag", "location recieved: " + location.toString());

        if (mLocationListeners.size() > 0) {
            int i = 0;
            for (LocationListener listener : mLocationListeners) {

                i++;
                Log.wtf("mytag", "location listener active number " + i + " with
id: " + listener.getId());

                if (listener != null)
                    listener.onLocationChanged(location);
                else
                    mLocationListeners.remove(listener);
            }
        } else {
            mFusedLocationClient.removeLocationUpdates(locationCallback);
        }
    }

    @Override
    protected void onResume() {


        IN_FOREGROUND = true;

        if (Timer != null)
            Timer.cancel();

        Timer = new Timer();
        Timer.scheduleAtFixedRate(new MainActivity.MyTimerTask(), 0, 20 * 60 *
1000);


        timeIn = System.currentTimeMillis();

        Analytics.send(this, "on_resume");

        GetUser.with(this, user).send();
```

```
        super.onResume();

    }

    @Override
    protected void onPause() {
        //Analytics: paso como valor el tiempo desde resume hasta pause

        IN_FOREGROUND = false;
        super.onPause();
    }

    @Override
    public void onConnected(@Nullable Bundle bundle) {

    }

    @Override
    public void onConnectionSuspended(int i) {

    }

    @Override
    public void onConnectionFailed(@NonNull ConnectionResult connectionResult) {

    }

    @Override
    protected void onStop() {
        Timer.cancel();
        super.onStop();
    }


    public ActivityFragment getActivityFragment() {
        return mActivityFragment;
    }

    public void setActivityFragment(ActivityFragment activityFragment) {
        this.mActivityFragment = activityFragment;
    }

    public void setGoogleFitFragment(GoogleFitFragment googleFitFragment) {
        this.googleFitFragment = googleFitFragment;
    }

    public void updateRemoteLocationData() {
        UpdateUserData.with(mActivity).sendGoodLocationReport("normal_report",
new OnTaskCompletedListener() {
```

```
            @Override
            public void OnComplete(String result, int resultCode, int resultType)
{
                //returns minimum supported version

                if (resultCode == MyServerClass.SUCCESSFUL) {
                    int actualVersion;
                    try {
                        PackageInfo pInfo =
mContext.getPackageManager().getPackageInfo(mContext.getPackageName(), 0);
                        actualVersion = pInfo.versionCode;
                    } catch (PackageManager.NameNotFoundException e) {
                        e.printStackTrace();
                        actualVersion = 10000;
                        Crashlytics.logException(new Exception("error when
getting version, e: " + e.toString()));
                    }


                    int minVersion = 0;
                    try {
                        minVersion = Integer.parseInt(result);
                    } catch (NumberFormatException e) {
                        Crashlytics.logException(new Exception("main activity:
result wasnt an integer: " + result));
                    }

                    SharedPreferences sharedPref =
PreferenceManager.getDefaultSharedPreferences(getApplicationContext());
                    SharedPreferences.Editor editor = sharedPref.edit();

                    if (actualVersion >= minVersion) {
                        //tienes acceso
                        editor.putInt("can_access", 1);
                        Log.wtf("mytag", "setting sp can access to 1");
                        editor.apply();
                    } else {
                        //no tienes acceso
                        editor.putInt("can_access", 0);
                        Log.wtf("mytag", "setting sp can access to 0");
                        editor.commit();

                        FirebaseAuth.getInstance().signOut();
                        Intent intent = new Intent(mContext,
UpdateActivity.class);
                        intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
                        mContext.startActivity(intent);
                        finish();
                    }
```

```
                } else if (resultCode == MyServerClass.NOT_CONNECTED) {

                }
            }
        });
    }


    public void updateLocation() {
        addLocationListener(new LocationListener() {
            @Override
            public void onLocationChanged(Location location) {
                removeLocationListener(this);
            }
        });
    }

}
```