



**Universidad
Europea Madrid**

LAUREATE INTERNATIONAL UNIVERSITIES



CICLO FORMATIVO DE GRADO SUPERIOR
DESARROLLO DE APLICACIONES MULTIPLATAFORMA

PROYECTO FIN DE CICLO

Blackbird

Desarrollo de una App de movilidad orientada a la Comunidad de
Madrid

Autores

Julen Bujanda Blanco

Adrián García Montón

CURSO 2018-19

TÍTULO: Blackbird: Desarrollo de una App de movilidad orientada a la Comunidad de Madrid

AUTORES: Julen Bujanda Blanco

Adrián García Montón

TUTOR DEL PROYECTO: Ernesto Ramiro Córdoba

FECHA DE LECTURA: 10 de junio de 2019

En Madrid

Ernesto Ramiro Córdoba
Tutor del PFC

RESUMEN

Como desarrolladores, nos dimos cuenta de la dificultad de encontrar datos en tiempo real que se encuentren públicamente, pues mucha información se mantiene en privado, de forma que solo pueden acceder a ella las empresas y los desarrolladores a los que la propia Comunidad de Madrid.

Con esta misma inquietud, Medialab contactó con nosotros, con la esperanza de que pudiésemos desarrollar una aplicación que consumiese datos en vivo de los medios de transporte públicos de la Comunidad de Madrid y planificase rutas en base a ellos en un mapa, mostrando horarios, tiempos de viaje e incluso rutas alternativas, con la finalidad de demostrar a las entidades públicas que se pueden desarrollar aplicaciones útiles y funcionales de código libre, y plantearles la necesidad de abrir sus datos, pues en una sociedad tan informatizada, con tanta gente capaz y con la voluntad de crear y ayudar, se pueden conseguir cosas increíbles.

Nos propusimos plantar unas bases sólidas de una aplicación escalable, gratuita y de código abierto, con la esperanza de demostrar la importancia de la accesibilidad de los datos y de incentivar a otros desarrolladores a trabajar tanto sobre ella como en base a ella.

Así pues, este proyecto es un llamamiento, tanto a las entidades públicas para que den acceso a su información como a otros desarrolladores que quieran colaborar, tanto a expandir esta aplicación como a crear sus propios proyectos en base a ella.

ABSTRACT

As developers, we realized how difficult was to find real time transit data publicly, since a lot of information is kept in private, therefore, it is only accessible for companies and the developers that have explicit consent by the Community of Madrid.

With the same concern, Medialab contacted us, hoping that we could develop an application that consumes real time data of the public transport of the Community of Madrid and it could plan routes based on them on a map, including schedules, time travel and even alternate paths, with the purpose of demonstrating to the public entities that it is possible to develop a useful and functioning open-source application, in order to present the necessity of the open data, because in a such technological society, with capable people and the will to contribute and help, amazing things can be achieved.

We proposed ourselves to put the solid bases of a scalable and free application, with the hope of demonstrating the importance of data accessibility and to encourage other developers to work with it.

Therefore, this project is a call, to public entities for them to give access to more information, and to developers that want to collaborate improving this application or making their own projects based on this.

AGRADECIMIENTOS

En primer lugar, hay que agradecer a Ernesto Ramiro Córdoba, nuestro profesor y tutor del PFC, por habernos guiado en la parte estética de esta aplicación y mostrarnos la importancia de la colaboración en el mundo del desarrollo, así como por ponernos en contacto con Medialab y con Gabriel Lucas.

Gabriel Lucas, por habernos abierto las puertas a este proyecto, dejándonos al cargo desde el primer momento, pero consiguiendo que todo estuviese claro desde el principio. Por su rápida respuesta, su acogida y sus ganas de cambiar la visión de la Comunidad frente a los datos privados, gracias.

Índice

1. INTRODUCCIÓN.....	7
1.1. Objetivos	8
1.2. Motivación.....	10
1.3. Antecedentes.....	11
1.3.1. Back-end.....	11
Front-end.....	14
2. DESARROLLO DEL PROYECTO	15
2.1. Herramientas tecnológicas	15
2.2. Planificación	23
2.3. Descripción del trabajo realizado	25
2.3.1. Back-end.....	25
2.3.2. Front-end	33
2.4. Resultados y validación.....	42
3. CONCLUSIONES	44
3.1. Innovación	45
3.2. Trabajo futuro	46
4. BIBLIOGRAFÍA Y WEBGRAFÍA	47
5. ANEXOS	49

1. INTRODUCCIÓN

El propósito de este proyecto, en colaboración con Medialab, es no sólo el de realizar una aplicación funcional y escalable, pues además de ello, teníamos como objetivo lograr un producto capaz de competir con otras aplicaciones similares creadas por grandes equipos de trabajo.

Medialab nos contactó con la propuesta de desarrollar una aplicación de trazado de rutas usando el transporte público que tuviese un crecimiento continuo gracias a la comunidad de desarrolladores, creando así un producto que no solo fuese eco-friendly, sino que además sirviese como punto de unión para otros que compartan nuestra pasión por crear un futuro sostenible gracias a la ayuda de todos, así como facilitar el desarrollo de aplicaciones futuras similares y el aprendizaje de los desarrolladores que deseen trabajar sobre la misma.

La mayor restricción de este trabajo es la de encontrar datos en tiempo real de diversos medios de transporte público para generar un listado de rutas eficientes con diferentes opciones.

Este proyecto también tiene un objetivo mayor del que ya supone crear un producto fiable y útil, como es concienciar a las entidades del Estado de lo que se conseguiría si se facilitase el acceso a los datos para el público, tanto con fines educativos como para su consumo con la finalidad de crear herramientas o aplicaciones open-source.

1.1. Objetivos

El principal propósito de este proyecto, más allá de el desarrollo del producto final en sí, es el de demostrar que la comunidad de desarrolladores de código abierto es capaz de resolver un problema de manera eficiente.

En este caso, queremos dejar ver que, con más recursos y desarrolladores que formen parte de este proyecto, podemos crear una aplicación que realmente ayude a los ciudadanos de la Comunidad de Madrid, y que a la vez, evidencie la necesidad de cambiar la forma en la que los órganos públicos desarrollan aplicaciones informáticas de cara a los ciudadanos.

Nuestros objetivos:

- Crear una comunidad de desarrolladores dirigida al Consorcio Regional de Transportes de Madrid.
- Implementar OpenTripPlanner en nuestra aplicación Android.
- Ofrecer un servicio de planificación de rutas desarrollado de forma gratuita.
- Proporcionar una aplicación base potente que la comunidad pueda usar o seguir elaborando.
- Combatir la percepción de que desarrollar una aplicación es un proceso de elevado coste.
- Demostrar lo que se puede conseguir con colaboración y código abierto.
- Proponer a la Comunidad de Madrid una mayor apertura de sus datos con fines educativos y de desarrollo.

Nos propusimos desarrollar, de forma gratuita, una aplicación móvil capaz de planificar rutas con los datos del transporte público (Autobús, Metro, Cercanías...) que fuese tanto funcional como ampliamente escalable.

También queríamos formar parte de un bien mayor, como es cambiar la percepción de que un desarrollo es altamente costoso, y dejando unas bases firmes sobre las que otros desarrolladores puedan trabajar o estudiar, siempre por el beneficio común, ofreciendo una herramienta tan potente como barata gracias a la ayuda de todos.

Este proyecto tiene intención de ser un referente, una aplicación que pueda ser mostrada a las organizaciones del Estado con la finalidad de combatir la forma privada en la que se almacenan los datos, pues creemos que una mayor accesibilidad a ellos es beneficioso tanto para estudiantes, la comunidad de desarrolladores y finalmente para los usuarios que se beneficien de estos productos.

1.2. Motivación

Gracias a Medialab y a su interés por colaborar con nosotros vimos la posibilidad, no simplemente desarrollar una aplicación, sino de servir a un fin mayor, como es concienciar de la importancia de los datos abiertos y de ayudar a otros desarrolladores en su proceso formativo, así como aunar una comunidad de desarrolladores con intención de crear programas open-source que compitan con otros proyectos similares de gran inversión y que mantengan unos valores éticos beneficiosos para todos, bien con mentalidad eco-friendly o con el objetivo de ayudar a la gente que lo necesita.

Esta motivación surgió gracias a nuestra colaboración con Medialab, y a lo que nos han ayudado a hacer posible. Gracias al taller Madrid Escucha que organizaron, conseguimos formar este grupo de trabajo, y una de las partes positivas, respecto al desarrollo a largo plazo de la aplicación, es su desarrollo ya se ha alargado más allá de este taller, al que asistimos a la presentación final donde descubrimos proyectos tanto interesantes como importantes para los ciudadanos de Madrid, desde aplicaciones sobre puntos negros de violencia machista hasta iniciativas para hacer de Madrid una ciudad más sostenible.

1.3. Antecedentes

Nuestro proyecto ha pasado por diferentes fases de investigación, a la hora de encontrar las mejores tecnologías para poder implementar nuestra idea y está inspirado en la comunidad de desarrolladores finlandesa HSL.

Respecto a la investigación, la dividiremos en dos principales fases, el desarrollo back-end, que abarca gran parte del proyecto, y el front-end, desarrollando una aplicación nativa en Android.

1.3.1. Back-end

Para el servicio back-end de nuestra aplicación, necesitábamos un servicio al que le pudiéramos proporcionar los datos del Consorcio Regional de Transportes de Madrid, y que los procesa para entregarnos la mejor ruta entre dos posiciones. Tras investigar, nos encontramos con dos diferentes tipos de implementaciones:

APIs web ya existentes (Google Places Transit, Mapzen)

En el caso de optar por esta opción, tendríamos la facilidad de no tener que desarrollar nuestro propio servicio, ni tener que configurar un sistema back-end en un servidor.

Pese a esto, nuestra apuesta por proporcionar los datos nosotros mismos, la hace incompatible con este tipo de APIs, ya que pueden llegar a ser incompletas en ciertas rutas de Madrid.

Crear nuestro propio servicio back-end

Finalmente, gracias al proyecto open-source OpenTripPlanner, conseguimos desarrollar una API REST a la que proporcionamos los archivos GTFS del Consorcio Regional de Transportes de Madrid, y nos devuelve las mejores rutas.

Una de nuestras inspiraciones ha sido la comunidad de desarrolladores finlandesa HSL, concretamente su proyecto Digitransit, que también se centra en una aplicación móvil con un back-end basado en OpenTripPlanner.

APIs

Cuando ya teníamos todo claro sobre la API que íbamos a utilizar para las rutas en transporte público, decidimos investigar sobre incluir otro tipo de opciones, ya que en Madrid existen medios muy diferentes para ir de un lugar a otro.

Nos queríamos centrar en medios de transporte como el de la bicicleta, pública o privada, patinetes eléctricos, e incluso taxis o servicios como Uber o Cabify. Con esto en mente, comenzamos a investigar cómo implementar este tipo de servicios en nuestra aplicación, sabiendo que la principal restricción sería el acceso a los datos.

Patinetes eléctricos

En este caso, la mayor parte de la investigación se basó en Lime, una empresa de alquiler de patinetes eléctricos disponibles en la ciudad de Madrid.

Lamentablemente, como la práctica totalidad de este tipo de servicios, no disponía de API pública hasta el momento, a través de la cual pretendíamos acceder a la localización en tiempo real de cada patinete.

A esto le siguió un proceso de investigación, en el dimos con la propia API que la app de Lime utiliza para comunicarse con sus servidores. Esta es una API cerrada y la intención de esta no es estar abierta al público. Posteriormente hablaremos sobre los retos tecnológicos que planteó.

BiciMAD y bicicletas privadas

Al poco tiempo de comenzar la investigación, nos dimos cuenta de que la situación en cuanto al acceso a los datos de las bicicletas de alquiler era muy similar a la de los patinetes. La única API pública que encontramos, es la API de la EMT.

En su API, la EMT devuelve únicamente la ubicación de las estaciones de BiciMAD, datos que no nos serían suficientes para poder realizar una integración del servicio en nuestra App. Pese a que conocimos que algún usuario consiguió acceder a la API interna de BiciMAD, no nos sería viable acceder de esta manera por los mismos motivos que a la de Lime.

Servicios de movilidad privada

En este punto, nos centramos en implementar una integración con servicios de movilidad de punto a punto, como el taxi, Uber o Cabify. En cuanto al taxi, lamentablemente no encontramos una API en Madrid que nos pudiera facilitar datos relevantes para el usuario.

Por lo tanto, decidimos realizar una integración con Uber, que tiene una API abierta con muchos servicios diferentes.

En nuestro caso, decidimos informar al usuario, en cuanto busque una ruta en transporte público, del precio al que le saldría solicitar un Uber a ese destino, el tiempo que tardaría, su precio, y en cuánto tiempo podría llegar un conductor a su ubicación.

Opinamos que esta información sería de gran ayuda para el usuario en caso de que viaje acompañado, ya que le facilita comprobar que opción es más económica.

En caso de que se decante por esta opción, al hacer click en ella, accedería directamente a la app de Uber y simplemente tendría que confirmar el viaje.

Front-end

Por nuestra experiencia desarrollando apps nativas en Android, nos decantamos finalmente por esta opción. Para desarrollarla, tendríamos que integrarla de manera fluida con las APIs que vayamos a usar, a través de clientes HTTP asíncronos.

Una parte muy importante de esta fase sería el desarrollo de la interfaz gráfica, que deberá ser lo más user-friendly posible. Ya a pesar de tener un gran desarrollo back-end, es la parte con la que juzgará el usuario final.

Decidimos que, para que la aplicación destaque, no sólo nos bastaría con proporcionar mejores rutas o datos que otras aplicaciones, si no que necesitaríamos integrar otras características que nos diferencien.

Parte de estas características sería la de mostrar otro tipo de servicios de los que previamente hemos hablado, para poder integrar en una misma aplicación, no sólo datos sobre transporte público, sino alternativas que puedan ayudar al usuario.

Otro tipo de peculiaridades que hemos integrado, son las opciones de guardar el historial de viajes realizados, o la de añadir un número indefinido de destinos favoritos, no sólo la casa y el trabajo.

2. DESARROLLO DEL PROYECTO

2.1. Herramientas tecnológicas

A lo largo del proyecto, hemos trabajado con diversas herramientas, tanto en la parte back-end como en la Font-end.

OpenTripPlanner

A través de esta herramienta de software libre, que descubrimos a lo largo de una larga investigación, podemos insertar los datos proporcionados por el Consorcio Regional de Transportes de Madrid, y a través de su API, poder conocer las mejores rutas para unir dos puntos en la ciudad.

Google Cloud

Con Compute Instance de Google Cloud, podemos tener fácilmente un servidor auto escalable que dará servicio a nuestra parte back-end.

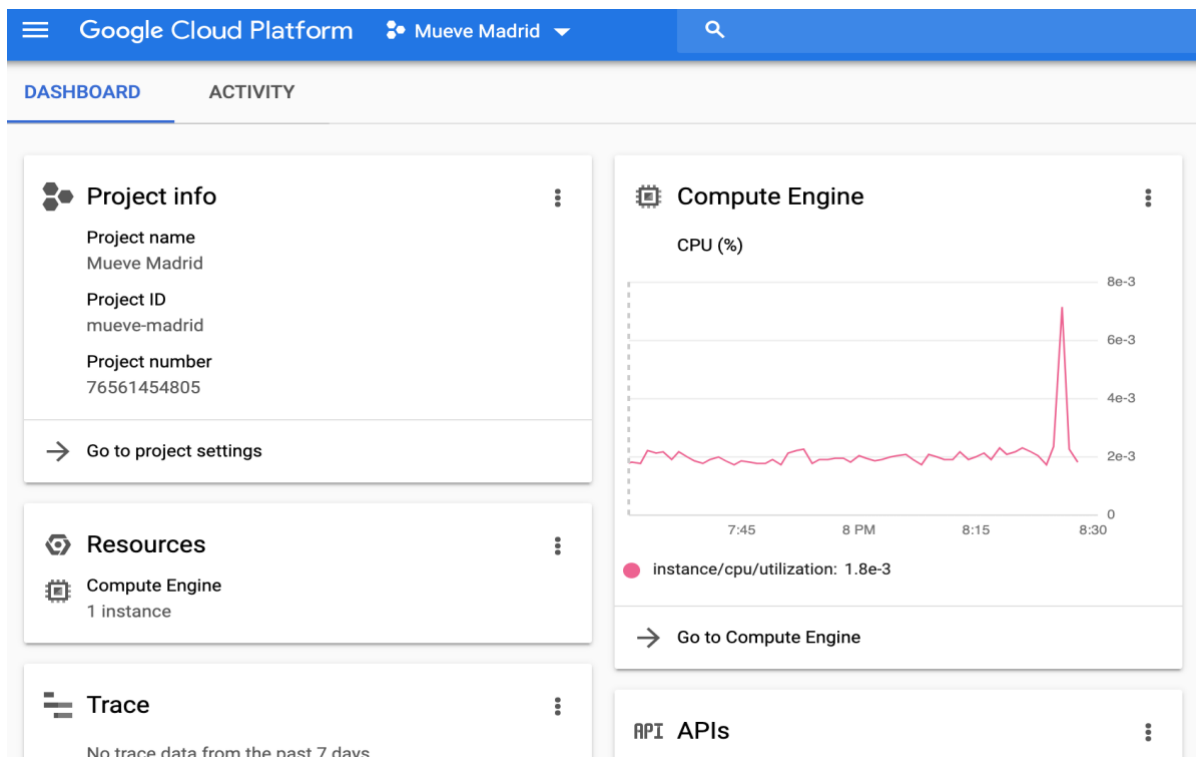


Figura 1: Vista de Google Cloud

Servidor Linux

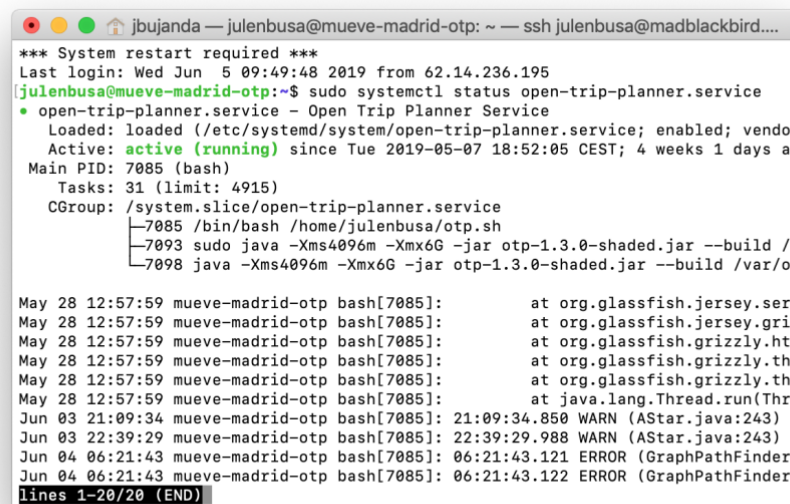
Configuramos un servidor Ubuntu 18.04, en el que configuramos OpenTripPlanner con sus respectivos archivos GTFS para los datos de transporte, y OSM para los mapas.

Dentro del servidor, configuramos OTP como un servicio. Para ello, creamos el servicio en /etc/systemd/system y le indicamos la ruta del script que inicia la aplicación.

```
[Unit]
Description=Open Trip Planner Service
[Service]
User=ubuntu
WorkingDirectory=/home/julenbusa/
ExecStart=/bin/bash /home/julenbusa/otp.sh
SuccessExitStatus=143
TimeoutStopSec=10
Restart=on-failure
RestartSec=5
[Install]
WantedBy=multi-user.target
```

open-trip-planner.service

Ya que OTP es una API que acepta peticiones web, también configuramos las peticiones HTTPS para que estas sean más seguras. Para ello, creamos un certificado SSL de Let's Encrypt, que permite crear certificados firmados de manera gratuita. Esto lo hicimos gracias a la herramienta Certbot, disponible en GitHub, que a través de comandos bastante simples genera los certificados públicos y privados que luego transformaremos a un formato aceptado por OpenTripPlanner.



```
jbulanda — julenbusa@mueve-madrid-otp: ~ — ssh julenbusa@madblackbird...
*** System restart required ***
Last login: Wed Jun  5 09:49:48 2019 from 62.14.236.195
julenbusa@mueve-madrid-otp:~$ sudo systemctl status open-trip-planner.service
● open-trip-planner.service - Open Trip Planner Service
   Loaded: loaded (/etc/systemd/system/open-trip-planner.service; enabled; vendor
   Active: active (running) since Tue 2019-05-07 18:52:05 CEST; 4 weeks 1 days a
 Main PID: 7085 (bash)
    Tasks: 31 (limit: 4915)
   CGroup: /system.slice/open-trip-planner.service
           └─7085 /bin/bash /home/julenbusa/otp.sh
             └─7093 sudo java -Xms4096m -Xmx6G -jar otp-1.3.0-shaded.jar --build /
               └─7098 java -Xms4096m -Xmx6G -jar otp-1.3.0-shaded.jar --build /var/o

May 28 12:57:59 mueve-madrid-otp bash[7085]:      at org.glassfish.jersey.ser
May 28 12:57:59 mueve-madrid-otp bash[7085]:      at org.glassfish.jersey.gri
May 28 12:57:59 mueve-madrid-otp bash[7085]:      at org.glassfish.grizzly.ht
May 28 12:57:59 mueve-madrid-otp bash[7085]:      at org.glassfish.grizzly.th
May 28 12:57:59 mueve-madrid-otp bash[7085]:      at org.glassfish.grizzly.th
May 28 12:57:59 mueve-madrid-otp bash[7085]:      at java.lang.Thread.run(Thr
Jun 03 21:09:34 mueve-madrid-otp bash[7085]: 21:09:34.850 WARN (AStar.java:243)
Jun 03 22:39:29 mueve-madrid-otp bash[7085]: 22:39:29.988 WARN (AStar.java:243)
Jun 04 06:21:43 mueve-madrid-otp bash[7085]: 06:21:43.121 ERROR (GraphPathFinder
Jun 04 06:21:43 mueve-madrid-otp bash[7085]: 06:21:43.122 ERROR (GraphPathFinder
lines 1-20/20 (END)
```

Figura 2: Vista del estado del servicio OpenTripPlanner

Firestore

Para poder gestionar los usuarios de la aplicación y sus datos, utilizamos Firestore, que permite hacerlo de manera eficiente y a la vez da muchas más posibilidades que otros servicios, por ejemplo, el que los usuarios se puedan loguear a través de diferentes servicios como Google o por mensaje de texto el tener acceso a una base de datos NoSQL, o identificar en tiempo real los problemas con la aplicación con Crashlytics.

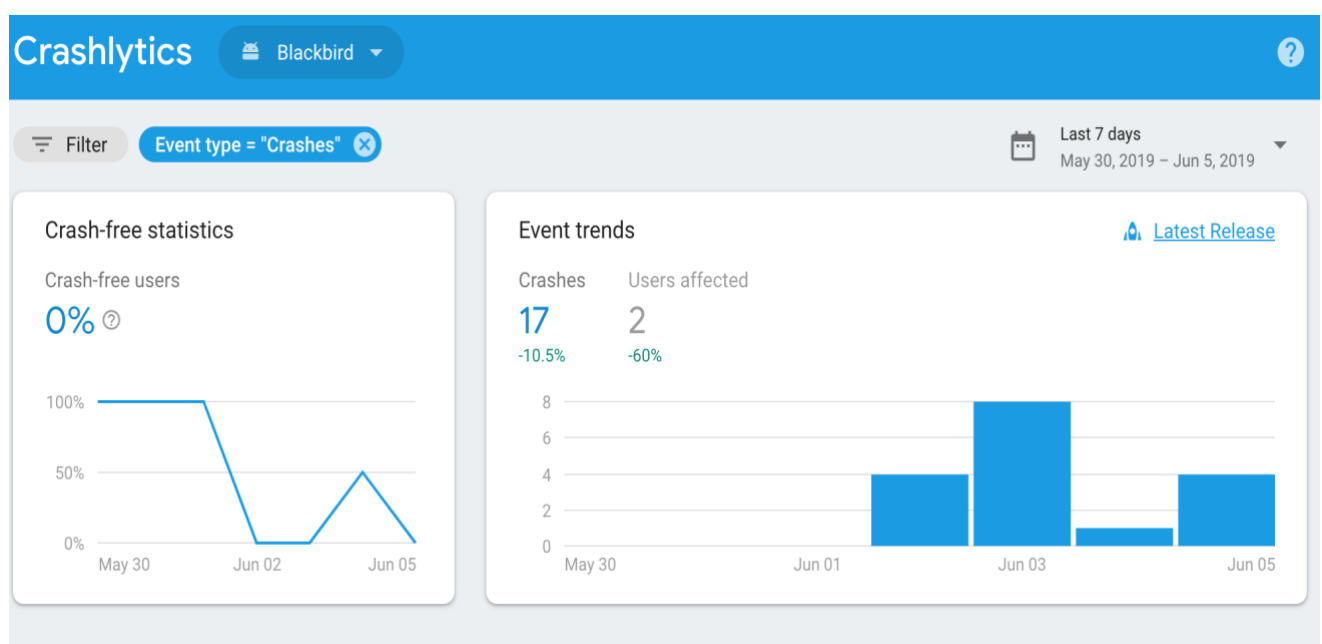


Figura 3: Vista de Crashlytics

Android

La aplicación front-end que hemos desarrollado ha sido una app nativa en Android, programando en Java y/o Kotlin. El desarrollar una app nativa para este sistema operativo, nos ha permitido utilizar un lenguaje de programación robusto, y a la vez diseñar una bonita interfaz gráfica. Uno de los mayores puntos fuertes de este tipo de implementación es el acceso a una cantidad enorme de librerías ya existentes.

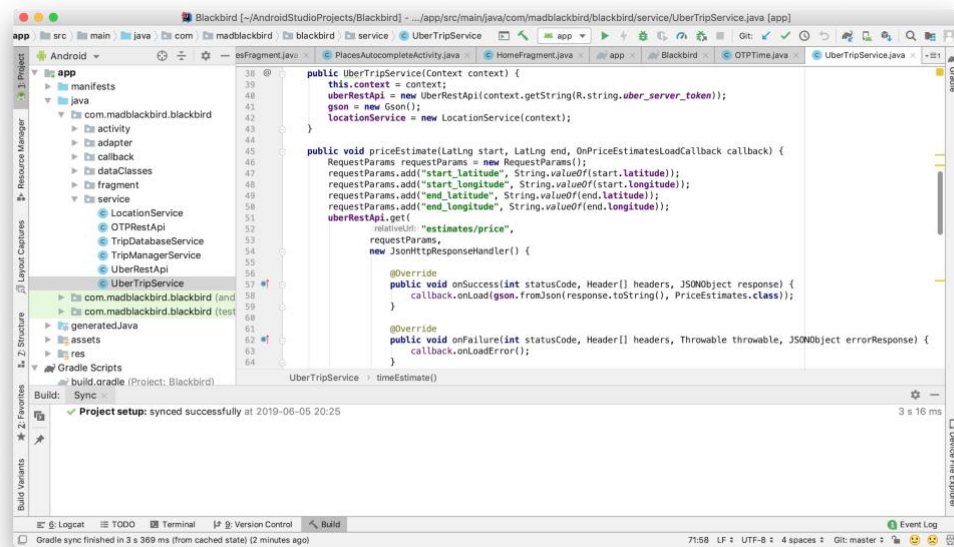


Figura 4: Vista de Android Studio

Android Async HTTP Client

Este cliente HTTP asíncrono para Android nos permite manejar con facilidad tanto las llamadas a la API REST de OpenTripPlanner como a la de Uber. A partir de este cliente fácilmente podemos implementar un servicio, que junto a Google Gson, nos devuelva objetos con los datos de los itinerarios de viaje para poder mostrarlos en nuestras vistas.

```
OTPRestApi.get(  
    context.getString(R.string.otp_server_url),  
    "plan",  
    requestParams,  
    new JsonHttpResponseHandler() {  
        @Override  
        public void onSuccess(int statusCode, Header[] headers, JSONObject response) {  
            TripPlan tripPlan = gson.fromJson(response.toString(), TripPlan.class);  
            callback.onItineraryLoaded(tripPlan.getPlan());  
        }  
    }  
);
```

Extracto de ripManagerService.java

Google Gson

Google Gson es una librería de serialización y deserialización, que como se observa en el ejemplo anterior, nos permite convertir fácilmente objetos Json en objetos Java. Gracias a sus anotaciones se puede personalizar mucho este proceso.

```
public class Itinerary implements Serializable {  
    @SerializedName("duration")  
    @Expose  
    private Integer duration;  
    @SerializedName("startTime")  
    @Expose  
    private Long startTime;  
    ...  
}
```

Extracto de Itinerary.java

Uber API

Con la API de Uber, hemos podido mostrar tanto las opciones que existen desde una localización para viajar en Uber al destino seleccionado, como los precios y los tiempos de espera. El acceso a esta, al igual que a OpenTripPlanner, ha sido a través de un cliente HTTP asíncrono.

```
public void priceEstimate(LatLng start, LatLng end, OnPriceEstimatesLoadCallback callback) {
    RequestParams requestParams = new RequestParams();
    requestParams.add("start_latitude", String.valueOf(start.latitude));
    requestParams.add("start_longitude", String.valueOf(start.longitude));
    requestParams.add("end_latitude", String.valueOf(end.latitude));
    requestParams.add("end_longitude", String.valueOf(end.longitude));
    uberRestApi.get(
        "estimates/price",
        requestParams,
        new JsonHttpResponseHandler() {

            @Override
            public void onSuccess(int statusCode, Header[] headers, JSONObject response) {
                callback.onLoad(gson.fromJson(response.toString(), PriceEstimates.class));
            }

        }
    );
    ...
}
```

Extracto de UberManagerService.java

Previamente, hemos configurado nuestra aplicación en el Uber Developer Dashboard e insertado nuestras credenciales en el proyecto Android. Para obtener los datos, primero solicitamos las opciones de transporte con su precio y duración del trayecto, y una vez obtengamos los productos disponibles, solicitamos cuál es el tiempo de llegada a nuestra localización, o la seleccionada.

Butterknife

Gracias a esta librería para Android que descubrimos a lo largo del desarrollo del proyecto, podemos unir las vistas con el código de una manera muy fácil a través de anotaciones de Java, por ejemplo:

```
@BindView(R.id.place_search)
EditText txtDestination;

...

@Override
protected void onCreate(Bundle savedInstanceState) {
    ...
    ButterKnife.bind(this);
    ...
}
```

Extracto de PlacesAutocompleteActivity.java

2.2. Planificación

Cómo se organizará el equipo: canales de comunicación (Slack, GitHub, Trello, Mail, etc.) tareas de cada uno, que se espera conseguir, temporalización.

El equipo se organizó gracias a distintas herramientas:

- Slack: Para la comunicación a través de mensajería instantánea entre los miembros equipo, y con Ernesto Ramiro y Gabriel Lucas, tutor de este PFC y asociado de Medialab respectivamente.

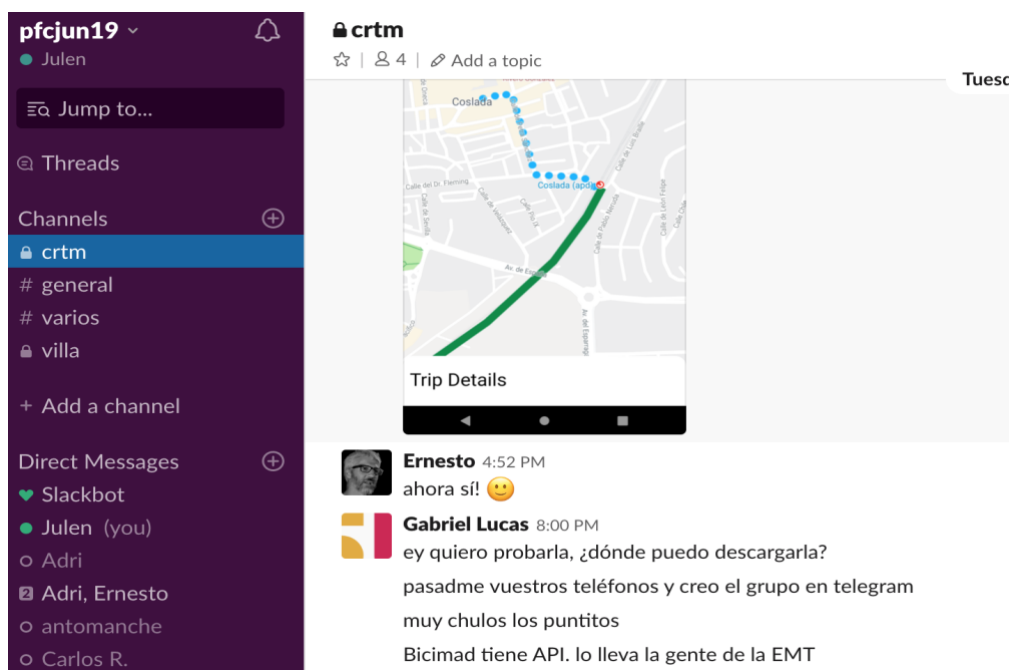


Figura 5: Vista de Slack

- GitHub: Para el control de versiones y el desarrollo conjunto de los miembros del equipo sobre la aplicación.

The screenshot shows a GitHub repository interface. At the top, it displays '291 commits', '2 branches', '0 releases', '3 contributors', and 'GPL-3.0'. Below this, there are buttons for 'Branch: master', 'New pull request', 'Create new file', 'Upload files', 'Find File', and 'Clone or download'. The main content is a table of files and their commit history.

File	Commit Message	Time Ago
.idea	POJO Classes	27 days ago
app	Remove unused dependency	5 hours ago
docs	Chnge to PDF	2 months ago
gradle/wrapper	Initial commit	a month ago
.gitignore	Initial commit	a month ago
LICENSE	Initial commit	a year ago
README.md	Update README.md	7 days ago
build.gradle	splash screen nuevo logo.	10 days ago
gradle.properties	Initial commit	a month ago
gradlew	Initial commit	a month ago
gradlew.bat	Initial commit	a month ago
settings.gradle	Initial commit	a month ago

Figura 6: Vista de GitHub

- Trello: Como administrador de tareas mediante pizarra kanban.

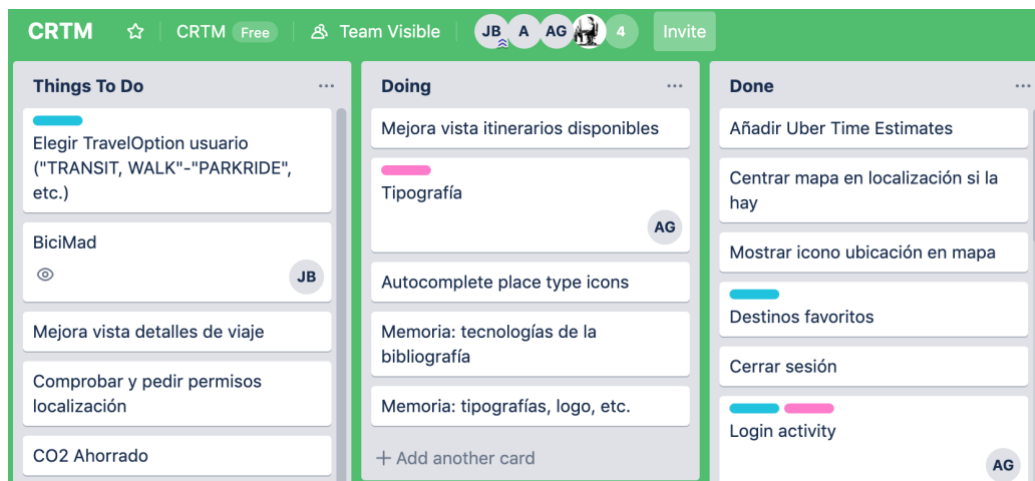


Figura 7: Vista de Trello

En lo referente al reparto de tareas, Julen Bujanda ha sido el responsable de los sistemas back end y de estructura, mientras Adrián García fue encargado con la parte front-end y de diseño.

Se espera conseguir una aplicación robusta de planificación de rutas, de aspecto profesional y altamente escalable en un intervalo de dos meses y medio.

2.3. Descripción del trabajo realizado

Después de la larga investigación que precedió al desarrollo del proyecto final, ya quedó muy claro qué había que hacer y cómo. Al igual que en apartados anteriores, dividiremos este proceso tanto en la parte de servidor como en la de usuario.

2.3.1. Back-end

Datos de transporte y mapas

Lo primero que hicimos fue descargarnos los datos de transporte más actualizados del Consorcio Regional de Transportes de Madrid. Para ello, accedimos a su página de datos abiertos, y los descargamos en formato GTFS para cada uno de los tipos de transporte, es decir, Metro, Cercanías, EMT, interurbanos, etc.

Tu Transporte Público



Metro

Selecciona un conjunto de datos para ver o descargar toda la información de la red de Metro ([líneas](#), [estaciones](#), [accesos](#), [andenes](#) y [vestíbulos](#)). [Más >](#)

También, puedes descargar la red y horarios en formato GTFS.



EMT

Selecciona un conjunto de datos para ver o descargar toda la información de la red de Autobuses Urbanos del municipio de Madrid ([líneas](#) y [paradas](#)). [Más >](#)

También, puedes descargar la red y horarios en formato GTFS.



Autobuses Urbanos

Selecciona un conjunto de datos para ver o descargar toda la información de la red de Autobuses urbanos de la Comunidad de Madrid ([líneas](#) y [paradas](#)). [Más >](#)

También, puedes descargar la red y horarios en formato GTFS.



Autobuses Interurbanos

Selecciona un conjunto de datos para ver o descargar toda la información de la red de Autobuses Interurbanos de la Comunidad de Madrid ([líneas](#) y [paradas](#)). [Más >](#)

También, puedes descargar la red y horarios en formato GTFS.



Metro Ligero/Tranvía

Selecciona un conjunto de datos para ver o descargar toda la información de la red de Metro Ligero/Tranvía ([líneas](#), [estaciones](#), [accesos](#), [andenes](#) y [vestíbulos](#)). [Más >](#)

También, puedes descargar la red y horarios en formato GTFS.



Cercanías

Selecciona un conjunto de datos para ver o descargar toda la información de la red de Cercanías ([líneas](#), [estaciones](#), [accesos](#), [andenes](#) y [vestíbulos](#)). [Más >](#)

También, puedes descargar la red y horarios en formato GTFS.

Después de esto, accederemos a la página de OpenStreetMap para descargarnos los ficheros OSM del área de la Comunidad de Madrid. Estos ficheros llevan los datos de mapas de la zona con las calles, necesarios para que OpenTripPlanner pueda procesar tanto los trasbordos entre estaciones, cómo por dónde se ha de viajar en una calle.

OpenStreetMap Edit History Export GPS Traces Use

Search Where is this? Go

Export

59.734

-20.522 20.347

41.476

[Manually select a different area](#)

Licence

OpenStreetMap data is licensed under the [Open Data Commons Open Database License \(ODbL\)](#).

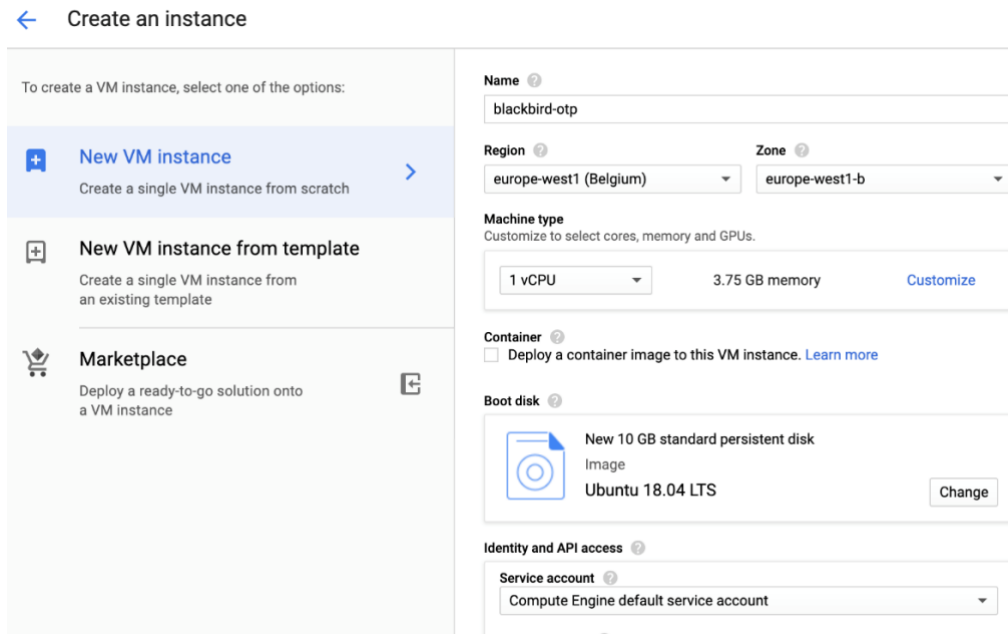
This area is too large to be exported as OpenStreetMap XML Data. Please zoom in or select a smaller area, or use one of the sources listed below for bulk data downloads.

[Overpass API](#)

Download this bounding box from a mirror of the OpenStreetMap database

Servidor back-end

Una vez hecho esto, llegó la hora de crear la instancia de Compute Engine en Google Cloud. Para ello creamos un proyecto nuevo, y creamos un servidor con Ubuntu 18.04, con las características técnicas adecuadas y la posibilidad de ser auto escalable. Después de esto, deberemos de transferir nuestra llave pública SSH al servidor para poder acceder, y configurar el firewall de Google Cloud con los puertos que vayamos a utilizar, en nuestro caso únicamente el 80 y 443.



← Create an instance

To create a VM instance, select one of the options:

- New VM instance**
Create a single VM instance from scratch
- New VM instance from template
Create a single VM instance from an existing template
- Marketplace
Deploy a ready-to-go solution onto a VM instance

Name ?
blackbird-otp

Region ?
europe-west1 (Belgium)

Zone ?
europe-west1-b

Machine type
Customize to select cores, memory and GPUs.
1 vCPU 3.75 GB memory [Customize](#)

Container ?
☐ Deploy a container image to this VM instance. [Learn more](#)

Boot disk ?
New 10 GB standard persistent disk
Image
Ubuntu 18.04 LTS [Change](#)

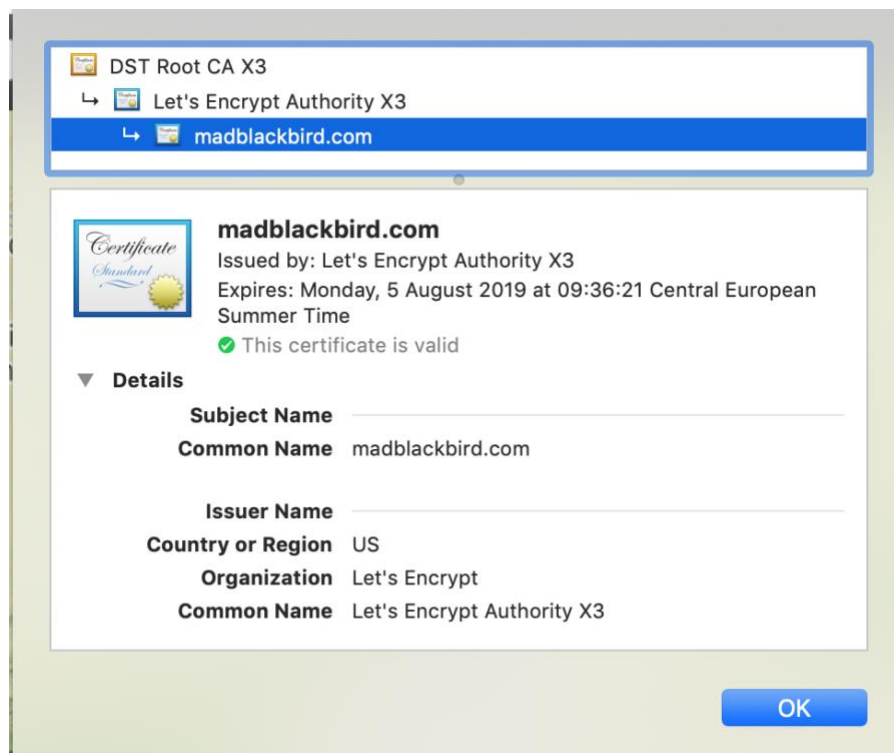
Identity and API access ?
Service account ?
Compute Engine default service account

Una vez creado el nuevo servidor, procederemos a actualizar y descargar los paquetes necesarios, configurar el JDK, y descargarnos el proyecto OpenTripPlanner mediante git. Una vez descargado, le haremos los cambios pertinentes, y pasaremos a configurar el servicio mediante el que se arrancará junto con el servidor.

También configuraremos la carpeta en la que se accederá a los ficheros GTFS de cada tipo de transporte, como a los datos de las calles en OSM. En nuestro caso se almacenarán en la carpeta /var/otp.

Para facilitar el acceso a la nuestra API, registramos el dominio <https://madblackbird.com>, de esta manera no tendremos que acceder al servidor a través de su dirección IP, que podría cambiar y los usuarios de la aplicación se verían afectados.

Nos aseguraremos de que la conexión entre nuestra app Android y el servidor sea segura, para ello, configuraremos un certificado SSL, firmado por Let's Encrypt, para acceder a ella, y que nadie pueda acceder a las rutas que realizan los usuarios de nuestra app. Posteriormente, lo convertiremos a un formato compatible con el servidor Grizzly, utilizado por OpenTripPlanner, y configuraremos el este servidor para que lo utilice.



APIs de terceros

API Interna de Lime

Como comentamos anteriormente, conseguimos dar con la API interna del servicio de alquiler de patinetes Lime. Esta es la API que utiliza la propia aplicación para comunicarse con sus servidores.

Después de descubrirla, dimos con la manera de obtener un token de acceso a la API. Para ello, necesitábamos un usuario previamente registrado en la aplicación. El procedimiento, utilizando el programa de línea de comandos curl, se puede demostrar de la siguiente manera:

- Primero, debíamos de realizar una petición GET, con el número de teléfono del usuario registrado, que solicita un código de acceso que se envía a este número.

```
curl --request GET --url 'https://web-production.lime.bike/api/rider/v1/login?phone=%2B34600111222'
```

- Una vez recibido este código, se lo devolvemos, guardándonos la cookie que nos devuelve, y el Token de acceso a la API.

```
curl --request POST --cookie-jar cookie --url 'https://web-production.lime.bike/api/rider/v1/login' --header 'Content-Type: application/json' --data '{"login_code": "961216", "phone": "+34600111222"}
```

- En cuanto recibamos estos datos, los utilizamos para realizar una petición GET que nos devuelva los patinetes cercanos a la ubicación deseada

```
curl --request GET \  
--url 'https://web-production.lime.bike/api/rider/v1/views/map?  
ne_lat=52.6&ne_lng=13.5&sw_lat=52.4&sw_lng=13.3&user_latitude=40.416943  
&user_longitude=-3.703618&zoom=16' \  
--header 'authorization: Bearer  
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX3Rva2VuljoiS1hXTUlyU0ZL  
QURHMyIsImxvZ2luX2NvdW50ljo0fQ.ALahBKepW8wpe9jI_dTYmPOkeKceEWiR  
ofnoBIMuHrA' \  
--cookie '_limebike-  
web_session=bllobmRZWWcxZ3ZKVvhRNzNjbDE5K2IJeG1QVk4vWDIvdzVSOX  
oNmZKOG8  
2cGR0Q3Y0NGFIUTFJV0xGNUVraEVtWWVETmtaQUJ3PT0tLTBEV1pKWjQ0aH  
BzSGlkT21qdU92cmc9PQ%3D%3D-- ec11208af65f17e0d7b2f7de2c171dc10ba7018e'
```

- Una vez terminemos este proceso, el resultado será el de una respuesta JSON con los datos deseados, como se observa en el siguiente ejemplo:

```
"bikes": [ {  
  "id": "ET-OIG2OLXARKYFZX2BKSRZGSOIP2Z5VBKFXAPH6QA",  
  "type": "bikes",  
  "attributes": {  
    "status": "locked",  
    "plate_number": "XXX-877",  
    "latitude": 52.496639,  
    "longitude": 13.395566,  
    "last_activity_at": "2019-05-16T16:45:05.000Z",  
    "bike_icon": null,  
    "type_name": "electric",  
    "battery_level": "low",  
    "meter_range": 20400,  
    "rate_plan": "€1 to unlock +\n€0.15 / 1 min",  
    "rate_plan_short": "<b><font color='#7AD319' size='16'  
face='Montserrat'>€1</font></b><font color='#444A57'  
size='12' face='Montserrat'> unlock + </font><b><font  
color='#7AD319' size='16' face='Montserrat'>€0.15 </font></b>  
<font color='#444A57' size='12' face='Montserrat'> / 1  
min</font>",  
    "bike_icon_id": 5,  
    "last_three": "877",  
    "license_plate_number": null  
  }  
} ]
```

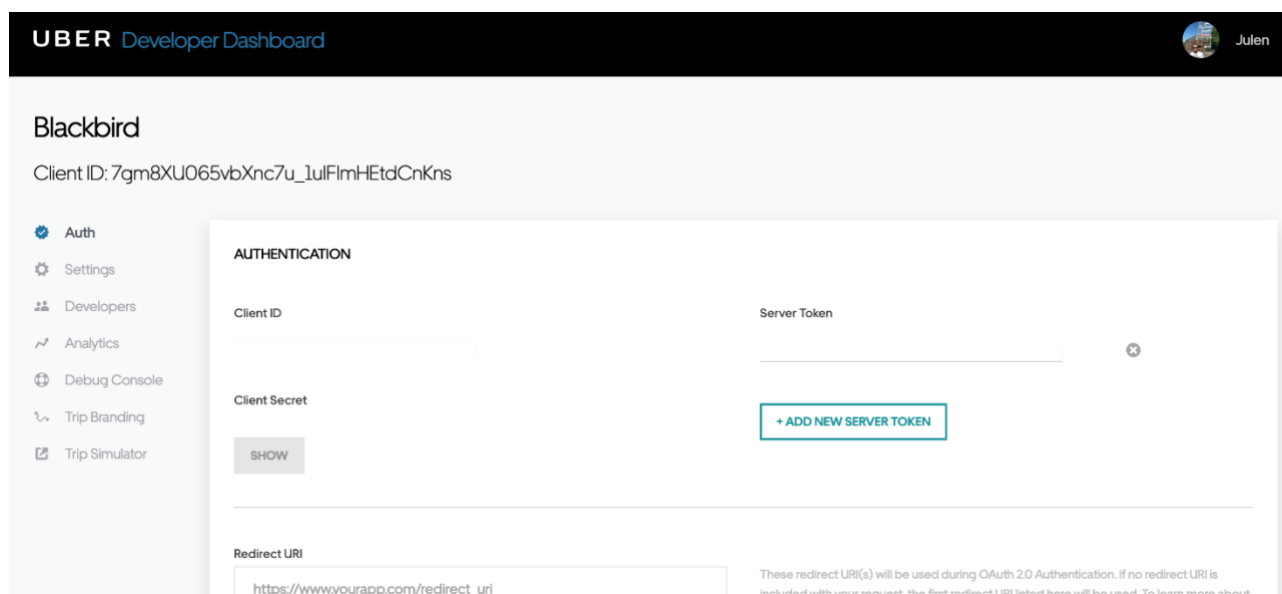
Esta investigación fue interesante, y pudimos recuperar satisfactoriamente datos que no están abiertos, pero pese a esto, y por la manera en la que se consigue el token de acceso a la API, no nos fue posible integrarlo en una aplicación a modo de producción.

API de Uber

Para poder mostrar información sobre alternativas en transporte privado, utilizaremos la API de Uber para informar al usuario de los tiempos de espera, el precio, y la duración del recorrido. Cabe destacar que a pesar de ser una compañía privada, el uso de esta herramienta es gratuito.

Para ello, nos deberemos de registrar en el portal de desarrolladores de Uber. Una vez allí, registraremos nuestra aplicación. Una vez hecho esto, Uber nos proporcionará diferentes tokens de acceso a la API según como vaya a ser nuestra implementación, lo que nos permitirá acceder a sus datos en tiempo real.

Posteriormente, realizaremos la implementación de este servicio en nuestra app Android.



The screenshot shows the Uber Developer Dashboard for an application named "Blackbird". The header includes the Uber logo and "Developer Dashboard" on the left, and a user profile icon labeled "Julen" on the right. Below the header, the application name "Blackbird" is displayed, followed by its Client ID: "7gm8XU065vbXnc7u_1ulFlmHEtdCnKns". A sidebar on the left contains navigation links: Auth, Settings, Developers, Analytics, Debug Console, Trip Branding, and Trip Simulator. The main content area is titled "AUTHENTICATION" and contains several fields: "Client ID" (pre-filled), "Client Secret" (with a "SHOW" button), "Server Token" (with a "+ ADD NEW SERVER TOKEN" button), and "Redirect URI" (pre-filled with "https://www.yourapp.com/redirect_uri"). A note at the bottom right explains that these URIs are used for OAuth 2.0 authentication.

UBER Developer Dashboard

Julen

Blackbird

Client ID: 7gm8XU065vbXnc7u_1ulFlmHEtdCnKns

Auth
Settings
Developers
Analytics
Debug Console
Trip Branding
Trip Simulator

AUTHENTICATION

Client ID

Server Token

+ ADD NEW SERVER TOKEN

Client Secret

SHOW

Redirect URI

https://www.yourapp.com/redirect_uri

These redirect URI(s) will be used during OAuth 2.0 Authentication. If no redirect URI is included with your request, the first redirect URI listed here will be used. To learn more about

2.3.2. Front-end

Una vez esté puesto en marcha nuestro servidor de rutas, y tengamos todas nuestras APIs configuradas, podremos proceder a crear nuestro front-end en forma de una app nativa Android.

El primer paso será el de crear un proyecto en Android Studio, como peculiaridad, esta vez daremos el paso a la librería AndroidX, sucesora de Android Support Library, esto nos permitirá que nuestro proyecto sea compatible con otras librerías de diseño que ya han dado y este paso, y a la vez sea retro compatible con las que continúen utilizando la librería Support.

Uno de los primeros pasos será integrar nuestra aplicación con el servicio de Google Firebase. Para ello será tan simple como crear un nuevo proyecto de Firebase e incluir en el proyecto su respectivo google-services.json, una vez hayamos indicado en este proyecto la llave SHA-1 de nuestra aplicación.

Estructura del Proyecto

En cuanto al código Java, dividiremos las clases creadas en 6 principales paquetes:

- **Activity:** contendrá las clases que hereden de AppCompatActivity, por lo tanto conformen las principales actividades de la aplicación Android
- **Adapter:** lo forman los diferentes adaptadores creados para sus respectivos RecyclerViews
- **Callback:** contiene los diferentes callbacks que hemos creado para facilitar la comunicación asíncrona con nuestras APIs
- **DataClasses:** como su nombre indica, forman parte de este paquete las clases que se utilizarán como estructuras de datos. Muchas de ellas

implementarán la interfaz Serializable para poder ser transferidas mediante Intents

- Fragment: contendrá las clases Fragment que se verán dentro de las actividades
- Service: este importante paquete, contiene las clases con las que nos comunicaremos con nuestros diferentes servicios a través de llamadas asíncronas

Librerías utilizadas

Durante el desarrollo de la aplicación, utilizamos diferentes librerías de terceros, como las siguientes:

- Firebase
 - Firebase Core: Incluye las clases fundamentales de Firebase y el acceso a Firebase Analytics
 - Firebase Auth: Necesitaremos esta librería para manejar los usuarios de nuestra aplicación
 - Firebase Database: Para la comunicación con la base de datos e tiempo real
 - Fabric Crashlytics: Nos permite ver en tiempo real cada excepción que se produzca en nuestra aplicación cuando la utiliza un usuario
- Google Auth: Necesaria para iniciar sesión con una cuenta de Google
- ButterKnife: Como comentábamos anteriormente, esta librería nos facilitará unir las vistas con nuestras actividades
- Google Places: Nos permitirá geo codificar direcciones reales, y ayudará al usuario a buscar el lugar de origen o destino de su viaje
- Android Async HTTP: Este cliente HTTP, vital para la comunicación asíncrona con nuestras APIs, nos permitirá recibir datos sin interrumpir el proceso normal de la aplicación
- Google Gson: nos permitirá serializar objetos y transformar objetos Json a Java
- AppIntro: Nos permite mostrar de manera sencilla una introducción a la aplicación la primera vez que se utiliza

Comunicación con las APIs de manera asíncrona

Esta sin duda es una de las partes más importantes del proyecto, ya que nuestros servicios back-end no nos servirán de nada si no los integramos de manera eficiente en nuestra aplicación.

Como comentamos anteriormente, utilizaremos la librería Android Async HTTP Client, y la implementaremos de las siguientes maneras:

OpenTripPlanner

Para ello, crearemos una clase tan simple como la siguiente:

```
public class OTPRestApi {

    private static final String BASE_URL = "/otp/routers/default/";

    private static AsyncHttpClient client = new AsyncHttpClient();

    public static void get(String siteUrl, String relativeUrl, RequestParams
requestParams, AsyncHttpResponseHandler responseHandler) {
        client.get(getAbsoluteUrl(siteUrl, relativeUrl), requestParams,
responseHandler);
    }

    private static String getAbsoluteUrl(String siteurl, String relativeUrl)
{
        return siteurl + BASE_URL + relativeUrl;
    }

}
```

OTPRestApi.java

Esta clase nos permite realizar peticiones GET especialmente a nuestra instancia de OpenTripPlanner. Para utilizarla, debemos de crear una petición GET con sus respectivos parámetros, y cuando recibamos la respuesta, devolverla a través del Callback:

```
OTPRestApi.get(
    context.getString(R.string.otp_server_url),
    "plan",
    requestParams,
    new JsonHttpResponseHandler() {

        @Override
        public void onSuccess(int statusCode, Header[] headers,
JSONObject response) {
            TripPlan tripPlan = gson.fromJson(response.toString(),
TripPlan.class);
            callback.onItineraryLoaded(tripPlan.getPlan());
        }
    }
);
```

Extracto de TripManagerService.java

Uber API

La implementación del acceso a esta API será muy parecida a la de OpenTripPlanner, salvo que para esta necesitaremos también generar peticiones de tipo POST.

Aquí podemos observar como solicitamos los tiempos de espera de cada servicio de Uber:

```
public void timeEstimate(LatLng start, TimeEstimatesLoadCallback callback) {
    RequestParams requestParams = new RequestParams();
    requestParams.add("start_latitude", String.valueOf(start.latitude));
    requestParams.add("start_longitude", String.valueOf(start.longitude));
    uberRestApi.get(
        "estimates/time",
        requestParams,
        new JsonHttpResponseHandler() {

            @Override
            public void onSuccess(int statusCode, Header[] headers,
JSONObject response) {

callback.onTimeEstimatesLoad(gson.fromJson(response.toString(),
TimeEstimates.class));
            }

        }
    );
}
```

Extracto de UberManagerService.java

En caso de este servicio, también lo utilizaremos para abrir la propia app de Uber y solicitar un viaje al destino que haya introducido el usuario, para ello, generaremos un deeplink a la aplicación, y en caso de que no esté instalada, dirigiremos al usuario al Play Store:

```
public void openUberApp(PriceEstimate priceEstimate, OTPPlace to) {
    try {
        Uri deeplink = Uri.parse("uber://?client_id=" +
context.getString(R.string.uber_client_id) +
        "&action=setPickup&pickup=my_location" +
        "&dropoff[latitude]=" + to.getLat() +
        "&dropoff[longitude]=" + to.getLon() +
        "&dropoff[nickname]=" + URLEncoder.encode(to.getName(),
"UTF-8") +
        "&dropoff[formatted_address]=" +
URLEncoder.encode(to.getAddressName(), "UTF-8") +
        "&product_id=" + priceEstimate.getProductid());
        Intent intent = new Intent(Intent.ACTION_VIEW);
        intent.setData(deeplink);
        try {
            context.startActivity(intent);
        } catch (ActivityNotFoundException e) {
            RideRequestDeeplink rideRequestDeeplink = new
RideRequestDeeplink.Builder(context)
                .setRideParameters(new RideParameters.Builder()
                    .build())
                .setSessionConfiguration(new
SessionConfiguration.Builder()

.setClientId(context.getString(R.string.uber_client_id))

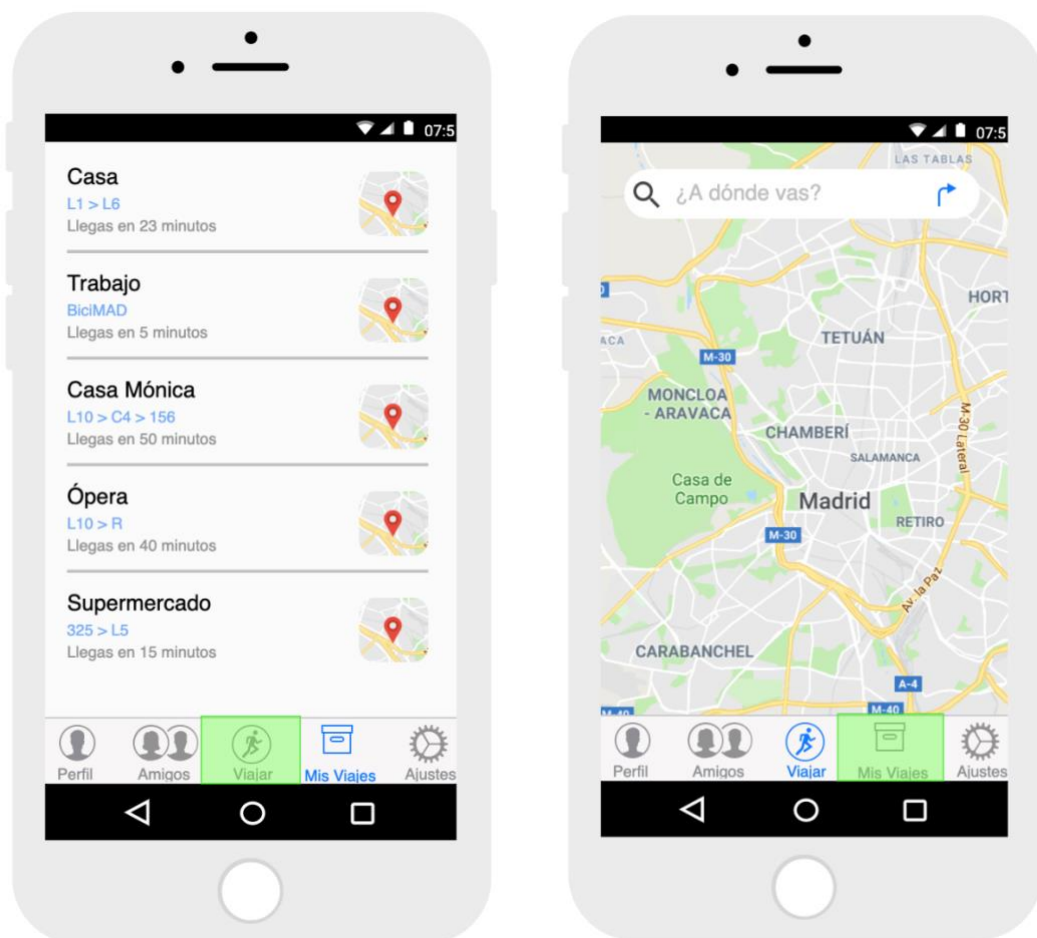
.setServerToken(context.getString(R.string.uber_server_token))
                    .build())
                .build();
            rideRequestDeeplink.execute();
        }
    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
    }
}
```

Extracto de UberManagerService.java

Diseño de la Aplicación

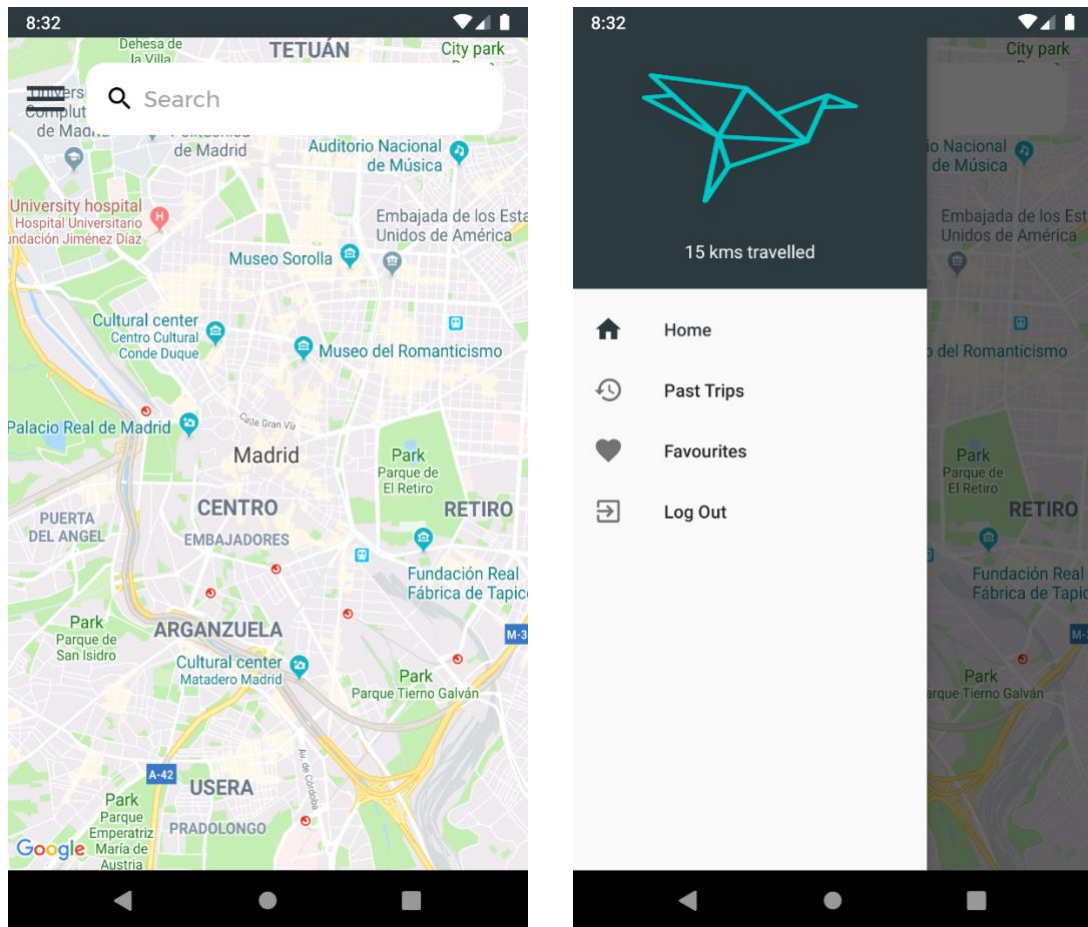
En cuanto al diseño, ya que la mayoría de usuarios que van a utilizar esta aplicación, habrán utilizado ya otras como Google Maps o Citymapper, las cuales tienen ciertas características en común, hemos optado por crear una interfaz lo más familiar posible, que el usuario sepa utilizar desde el primer momento.

Estos son los primeros mockups que diseñamos:



Estos mockups se realizaron cuando el proyecto no estaba muy avanzado, y nos dimos cuenta de particularidades que se podrían mejorar de cara a la interacción con el usuario, como por ejemplo, el cambio de un menú en la parte inferior de la aplicación que restaba importancia al mapa, a un menú desplegable lateral.

Finalmente, aquí podemos ver partes del diseño en un estado avanzado de la aplicación:



Como se puede apreciar, las diferencias son grandes, pero la parte más importante de la aplicación, el mapa, continúa siendo la que más notoriedad tiene.

2.4. Resultados y validación

Tras este desarrollo, conseguimos los siguientes resultados:

- Una aplicación Android capaz de calcular rutas en transporte público de un lugar a otro

Esta es la principal característica de la aplicación, y a través de todos los test que hemos realizado, funciona de una manera muy satisfactoria, incluso proporcionando datos más completos que otras aplicaciones mucho más grandes, como Google Maps, que falla en proporcionar resultados en viajes a municipios más pequeños de la Comunidad de Madrid.

- Mostrar información sobre alternativas en transporte privado

Respecto a esta característica hemos comprobado que funciona muy bien en todo tipo de situaciones. Se consigue mostrar el tiempo que tarda un Uber en llegar el destino, el precio aproximado del trayecto, y el tiempo que tardará en llegar el conductor más cercano.

Además de esto, muestra los diferentes tipos de productor que tiene Uber, como UberX, el más económico, o Uber Black, de transporte en vehículos de lujo.

Cuando el usuario hace click en la opción de Uber deseada, se abre la aplicación de esta compañía y con una sola pulsación solicitará un vehículo.

- Guardar los viajes realizados y los destinos favoritos

Si el usuario lo desea, e inicia sesión en la aplicación, a través de Google, mail, o número de teléfono, se procederá a almacenar cada viaje en transporte público que realice, además se almacenará cuál es su total de kilómetros recorridos como referencia.

En caso de que desee guardar sus destinos favoritos, como su trabajo, casa o cualquier otro lugar, lo podrá hacer una vez haya solicitado información sobre ese trayecto. Posteriormente sólo se tendrá que dirigir a la pestaña de favoritos y hacer click en el destino deseado.

Como conclusión en este aspecto, aunque no se hayan logrado incluir todos los servicios de transporte privado, la aplicación funciona bien en todos los aspectos que incluye.

3. CONCLUSIONES

Durante el desarrollo del proyecto, conseguimos resultados satisfactorios, dentro de las restricciones de hemos conseguido crear una aplicación real que puede llegar a competir a nivel de calidad de los datos con otras mucho más grandes.

En nuestra opinión, una de las cosas que demuestra este trabajo, es la dificultad para acceder a datos de movilidad en tiempo real en la Comunidad de Madrid, tanto de fuentes públicas como privadas. Y por esto mismo hemos centrado el proyecto en poder facilitar a los ciudadanos este acceso, y pensamos continuar haciéndolo.

Es necesario comentar, que, si no hubiera sido por nuestra colaboración con Gabriel Lucas, coordinador de IT en Medialab, no podríamos haber llegado tan lejos en este proyecto. Y si nos tenemos que quedar con una experiencia positiva que hayamos tenido a lo largo del proyecto, desde luego sería con esta colaboración.

3.1. Innovación

En nuestra opinión, el aspecto más novedoso de esta aplicación es la de tener acceso a todos los datos del Consorcio Regional de Transportes de Madrid de forma directa, a lo contrario de otras aplicaciones que recurren a APIs de terceros para acceder a datos incompletos.

Otro de los aspectos novedosos, y que pensamos mejorar, es el de integrar diferentes tipos de transporte en una aplicación, recurriendo directamente a los datos oficiales de cada uno de ellos. Pensamos que esto es la característica que más calidad aporta, y seguiremos avanzando desde este punto de vista.

3.2. Trabajo futuro

Desde que comenzamos a desarrollar este proyecto, y gracias a nuestra colaboración con Medialab, hemos tenido la intención de continuar con él.

No sólo eso, sino que, aunque nosotros no pudiéramos seguir desarrollando el proyecto, esperamos que la comunidad de desarrolladores que queremos ayudar a construir continúe con este.

Estas son unas de las características que deseamos añadir en el futuro:

- Integración con BiciMAD y servicios de bicicletas privadas
- Integración con servicios de patinetes
- Mejora de la interfaz gráfica
- Implementación de datos en tiempo real
- Gamificación
- Estadísticas de ahorro de emisiones

Aún queda mucho por avanzar para poder integrar en nuestra aplicación todos los datos, y avanzaremos en medida de lo posible. Ya que lo único que nos separa de conseguir este objetivo es el acceso a los datos, trabajaremos para conseguirlos.

4. BIBLIOGRAFÍA Y WEBGRAFÍA

- MadridEscucha 2019 <https://www.medialab-prado.es/actividades/presentacion-de-madrid-escucha-2019>
- Consorcio Regional de Transportes de Madrid <http://datos.crtm.es/>
- EMT OpenData <https://opendata.emtmadrid.es/>
- Android Developer <https://developer.android.com/>
- Whim - <https://whimapp.com/>
- Mapzen <https://www.mapzen.com>
- HSL (Comunidad de Desarrolladores Finlandesa) <https://dev.hsl.fi/>
- Digitransit <https://digitransit.fi/>
- OpenTripPlanner <https://www.opentripplanner.org/>
- Uber Developers <https://developer.uber.com/>
- Firebase <https://firebase.google.com/>
- Google Cloud <https://cloud.google.com/>
- Google Gson <https://github.com/google/gson>
- Android Async HTTP Client <https://loopj.com/android-async-http/>
- AppIntro <https://github.com/AppIntro/AppIntro>
- Google Places API <https://cloud.google.com/maps-platform/places/>
- Fabric Crashlytics (Ahora Firebase) <https://fabric.io/kits/android/crashlytics>
- Google Maps Utility Library <https://developers.google.com/maps/documentation/android-sdk/utility/>
- Mi transporte <https://play.google.com/store/apps/details?id=com.crtm.mitransporte>
- 1&1 Ionos <https://www.ionos.es/>
- Butterknife <https://jakewharton.github.io/butterknife>

- GTFS File Format <https://gtfs.org/> - <https://developers.google.com/transit/gtfs/>
- OSM File Format https://wiki.openstreetmap.org/wiki/OSM_XML
- Let's Encrypt <https://letsencrypt.org/>
- Certbot <https://github.com/certbot/certbot>
- Slack <https://slack.com>
- Trello <https://trello.com/>
- Lime <https://www.li.me/>
- BiciMAD <https://www.bicimad.com/>
- Hackeo API BiciMAD
https://cadenaser.com/emisora/2017/09/21/radio_madrid/1505997762968128.html

5. ANEXOS

A. Respuestas de las APIs

- A.1. Respuesta API propia OpenTripPlanner
- A.2. Respuesta API tiempos de Uber
- A.3. Respuesta API precios de Uber
- A.4. Respuesta API Lime