|  |
| --- |
| A description...  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. Índice 6](#_Toc10795925)

[1. Introducción 7](#_Toc10795926)

[1.1. Objetivos 8](#_Toc10795927)

[1.2. Motivación 9](#_Toc10795928)

[1.3. Antecedentes 10](#_Toc10795929)

[1.3.1. Back-end 10](#_Toc10795930)

[Front-end 13](#_Toc10795931)

[2. DESARROLLO DEL PROYECTO 14](#_Toc10795932)

[2.1. Herramientas tecnológicas 14](#_Toc10795933)

[2.2. Planificación 20](#_Toc10795934)

[2.3. Descripción del trabajo realizado 22](#_Toc10795935)

[2.3.1. Back-end 22](#_Toc10795936)

[2.3.2. Front-end 28](#_Toc10795937)

[2.4. Resultados y validación 35](#_Toc10795938)

[3. CONCLUSIONES 36](#_Toc10795939)

[3.1. Innovación 37](#_Toc10795940)

[3.2. Trabajo futuro 38](#_Toc10795941)

[4. BIBLIOGRAFÍA Y WEBGRAFÍA 39](#_Toc10795942)

[5. ANEXOS 41](#_Toc10795943)

# 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.

## 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.

## 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.

## 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.

### 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 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.

# DESARROLLO DEL PROYECTO

## Herramientas tecnológicas

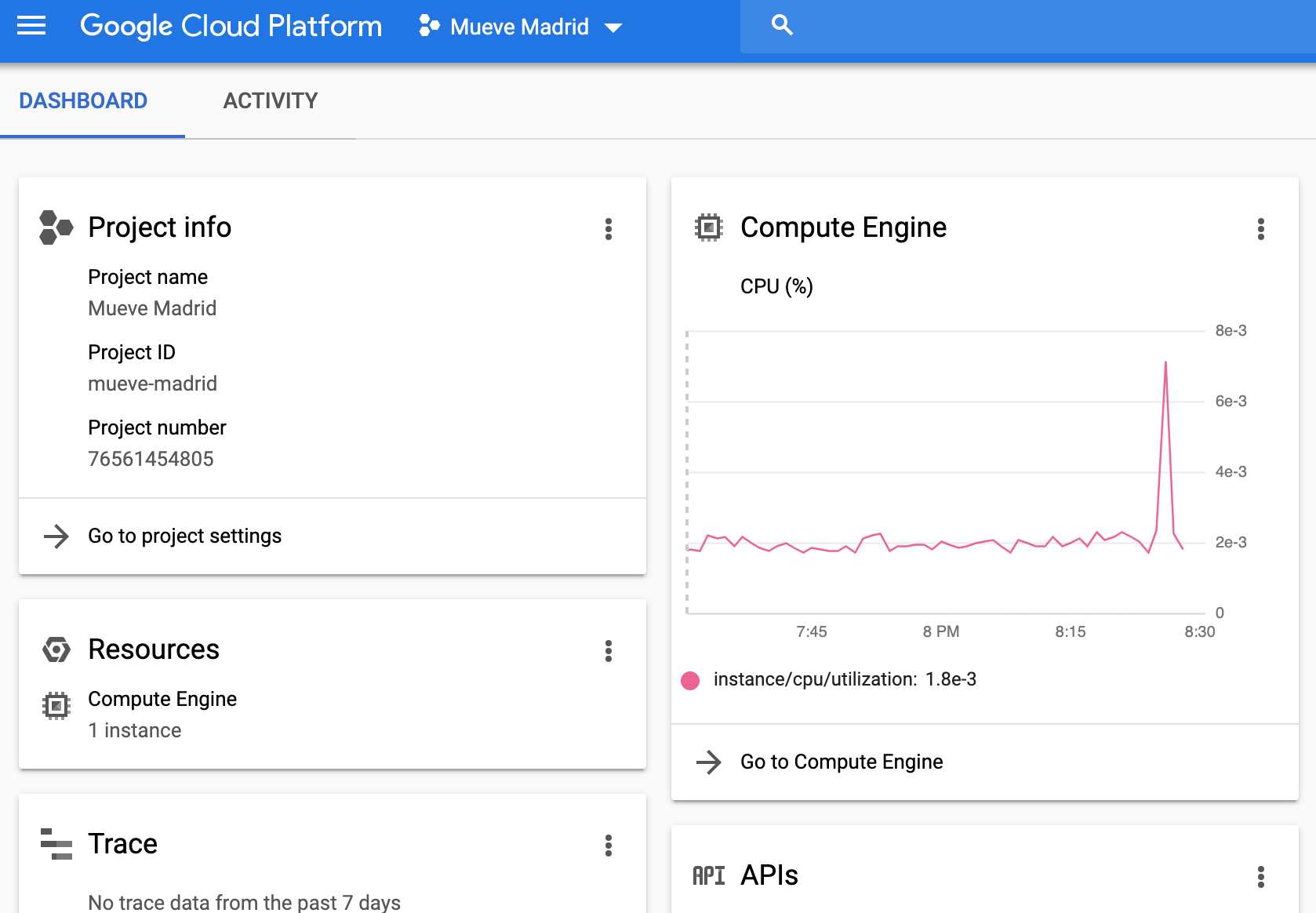
A lo largo del proyecto, hemos trabajado con diversas herramientas, tanto en la parte back-end como en la Font-end.

#### 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 backend.



**Figura 1: Vista de Google Cloud**

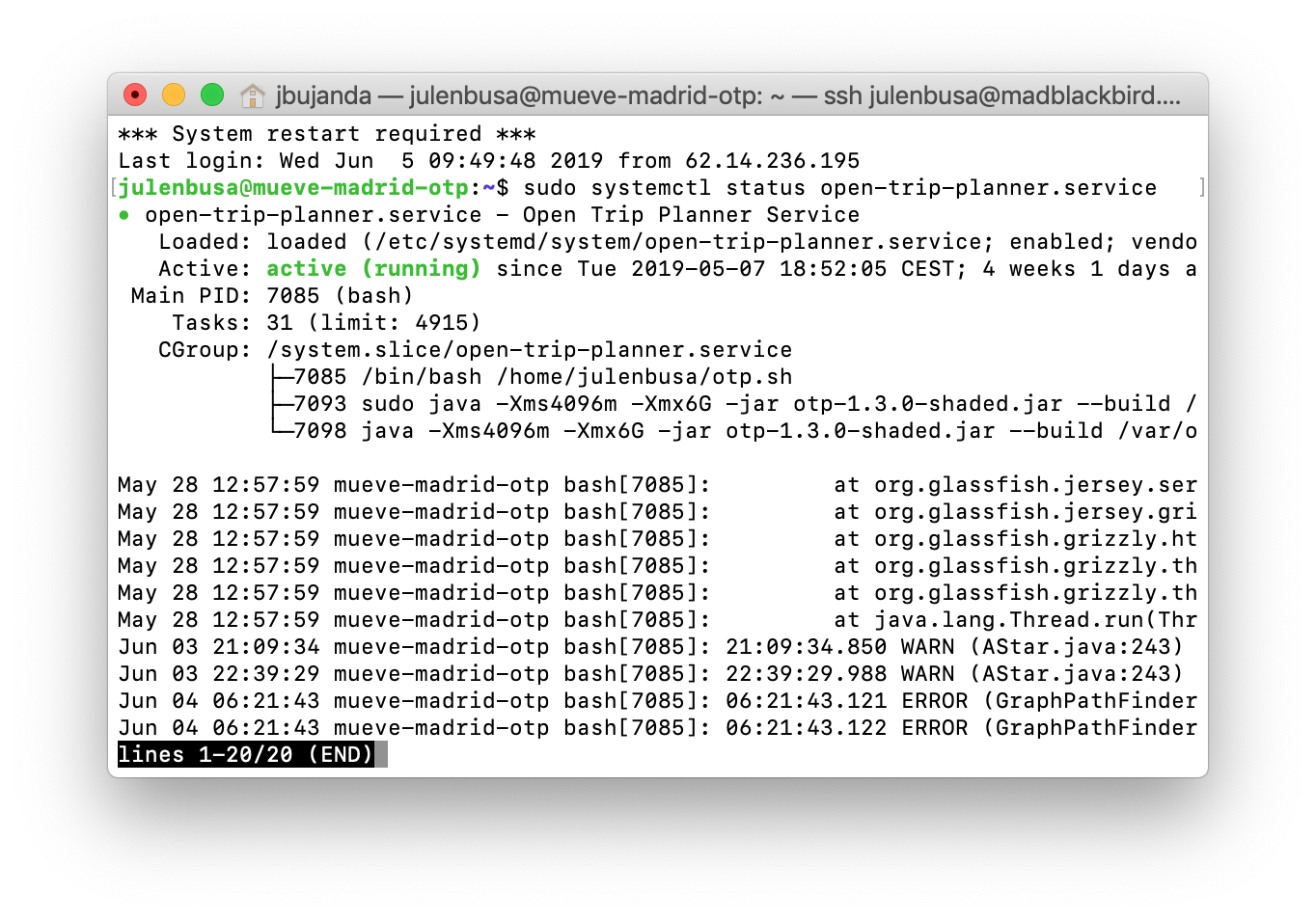
#### Servidor Linux

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 |

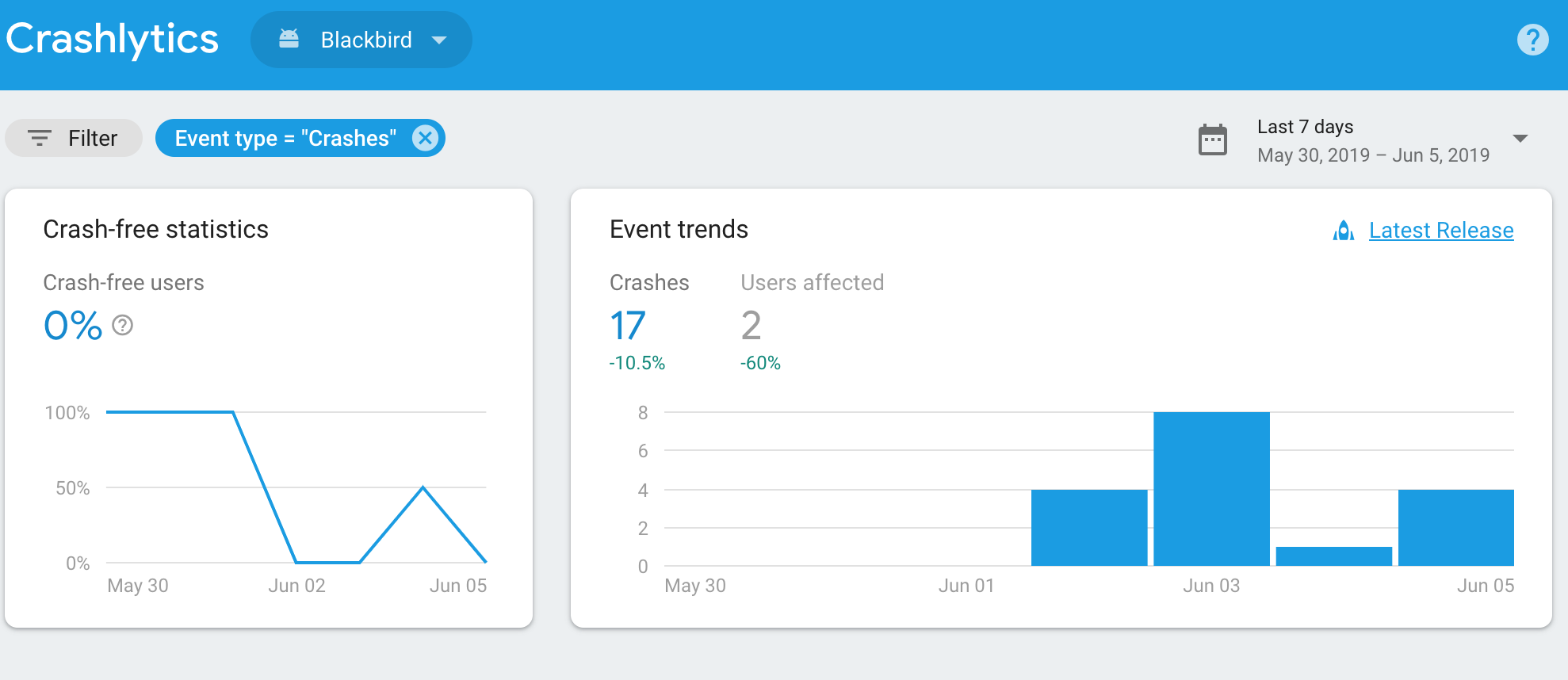
Ya que OTP es una API que acepta peticiones web, también configuramos las peticiones HTTPS para que estas sean más seguras. Para ello, creamos un certificado SSL de Let’s Encrypt, que permite crear certificados firmados de manera gratuita. Esto lo hicimos gracias a la herramienta Certbot, disponible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



**Figura 2: Vista del estado del servicio OpenTripPlanner**

#### Firebase

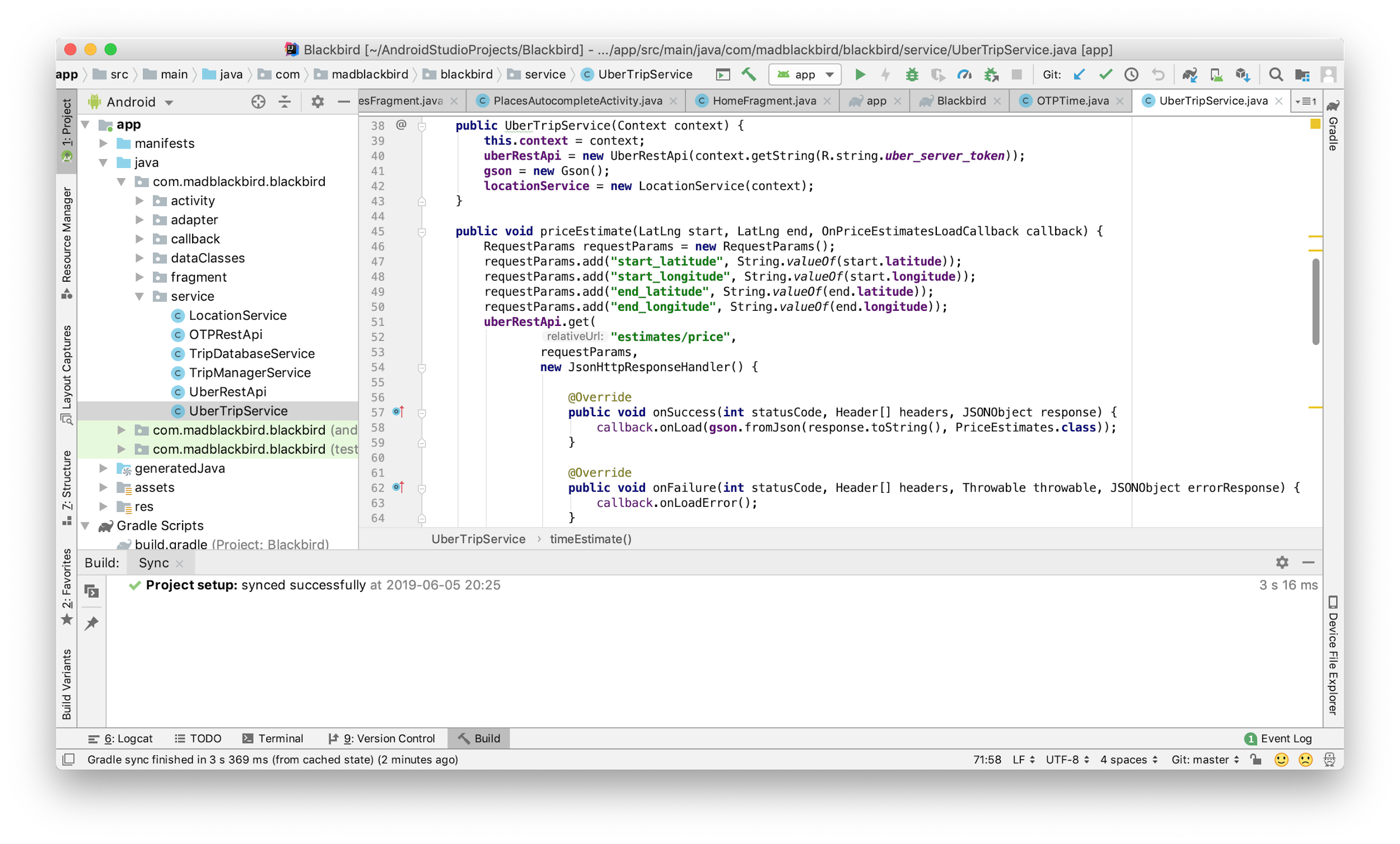
Para poder gestionar los usuarios de la aplicación y sus datos, utilizamos Firebase, que permite hacerlo de manera eficiente y a la vez da muchas más posibilidades que otros servicios, por ejemplo, el que los usuarios se puedan loguear a través de diferentes servicios como Google o por mensaje de texto 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**);  ... |

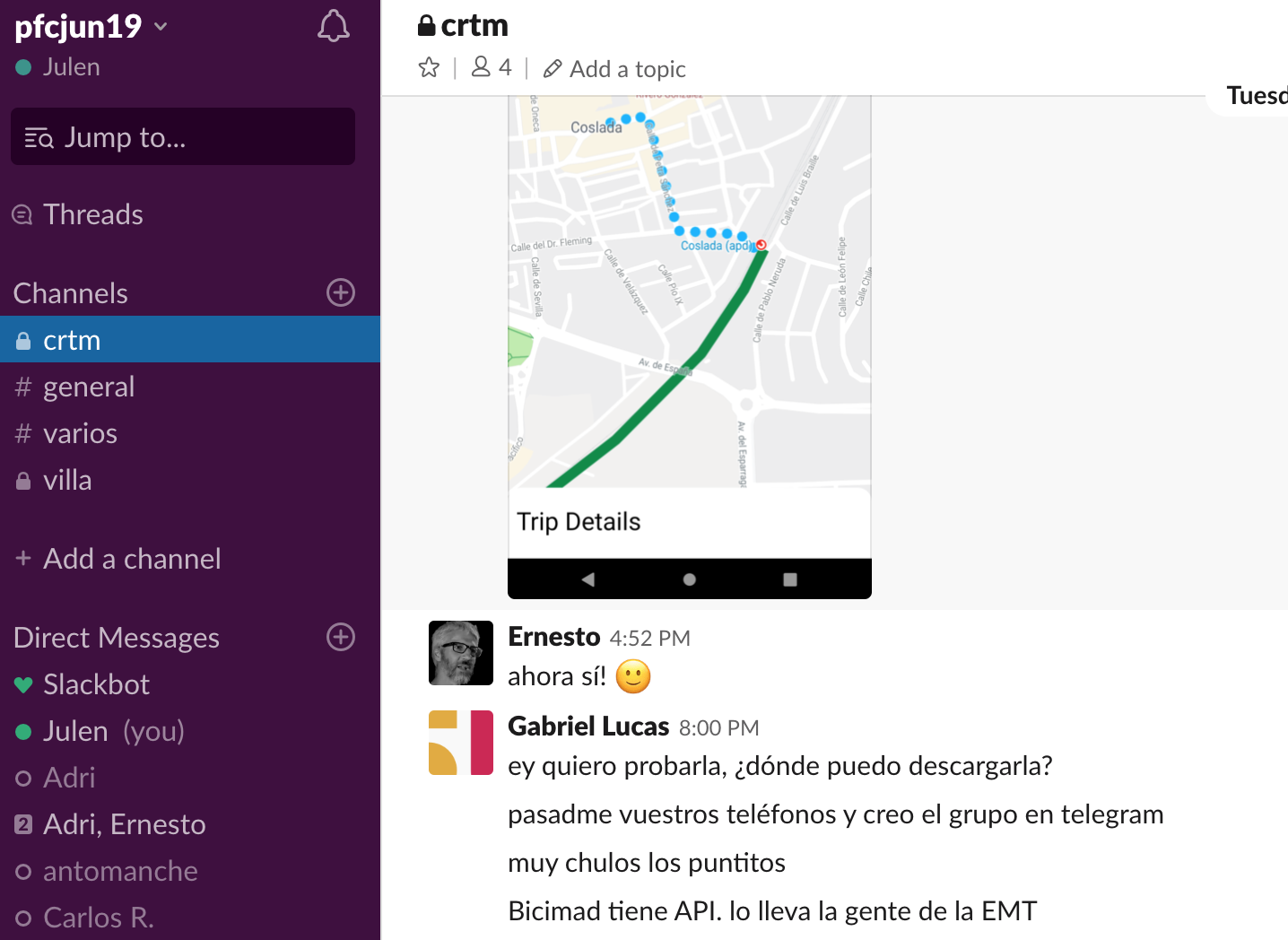
**Extracto de PlacesAutocompleteActivity.java**

## Planificación

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

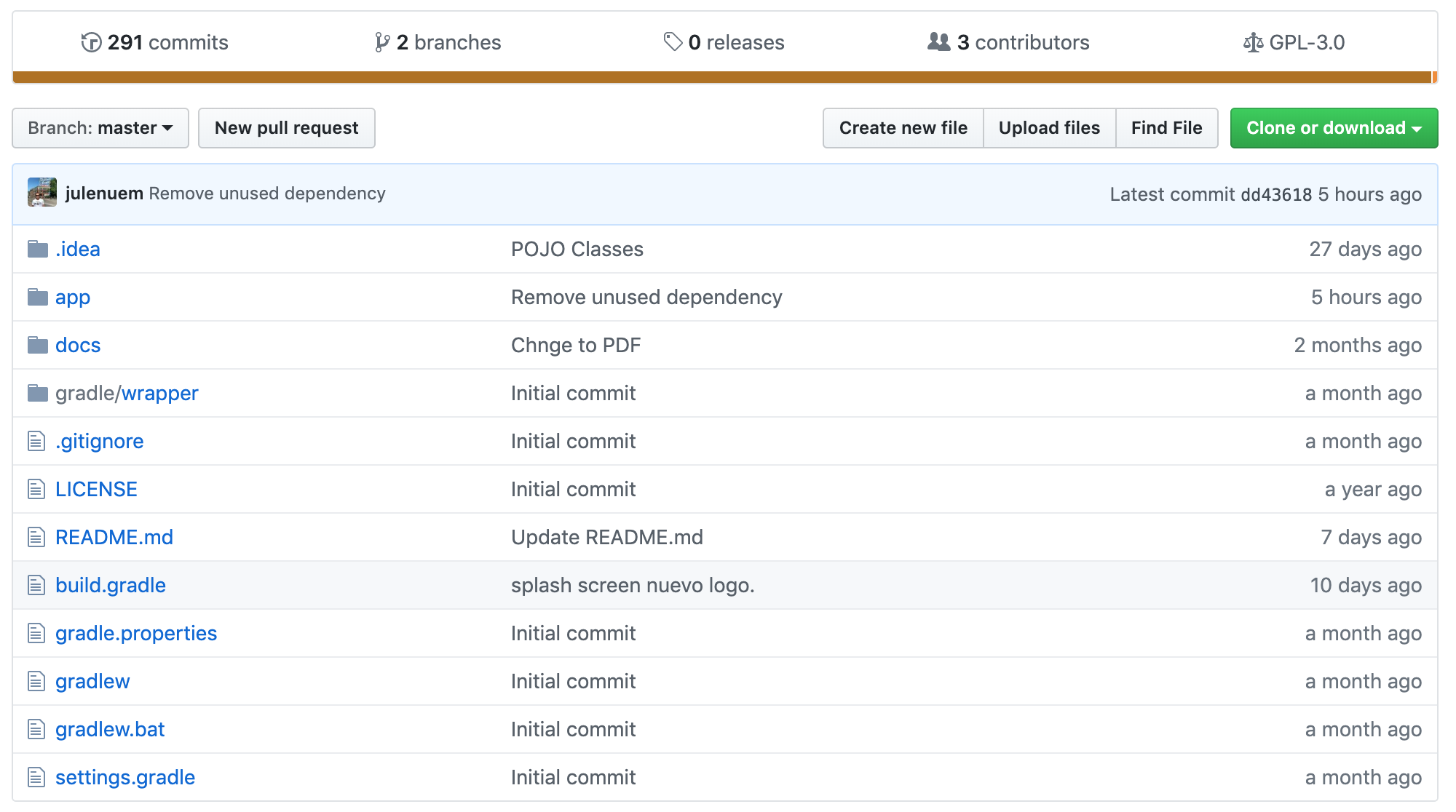
El equipo se organizó gracias a distintas herramientas:

* 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.



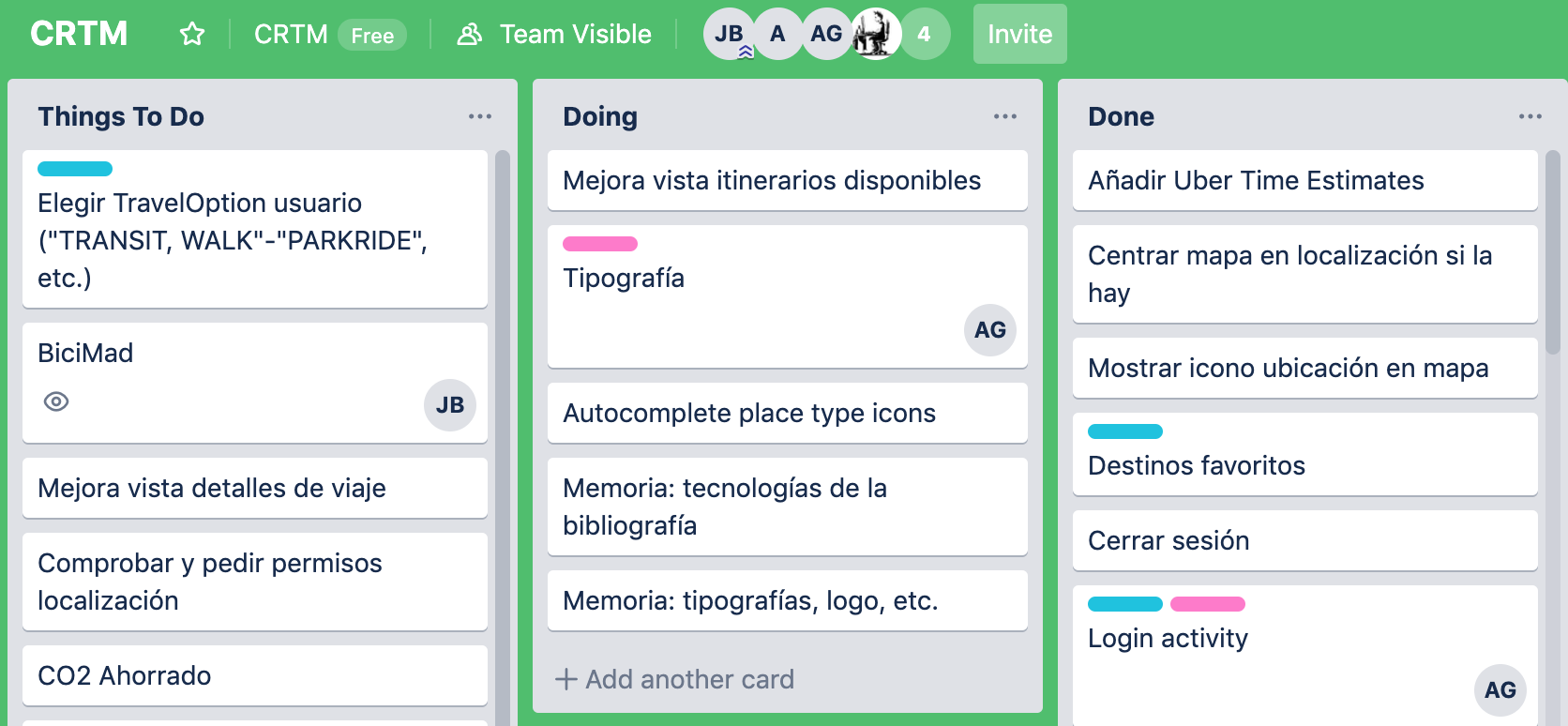
**Figura 5: Vista de Slack**

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



**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.

## 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.

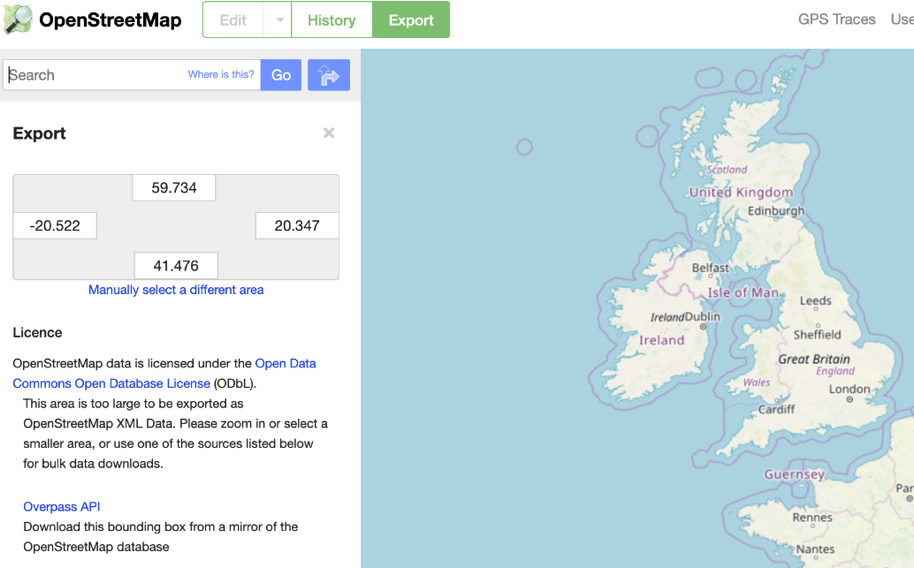
### Back-end

#### 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.

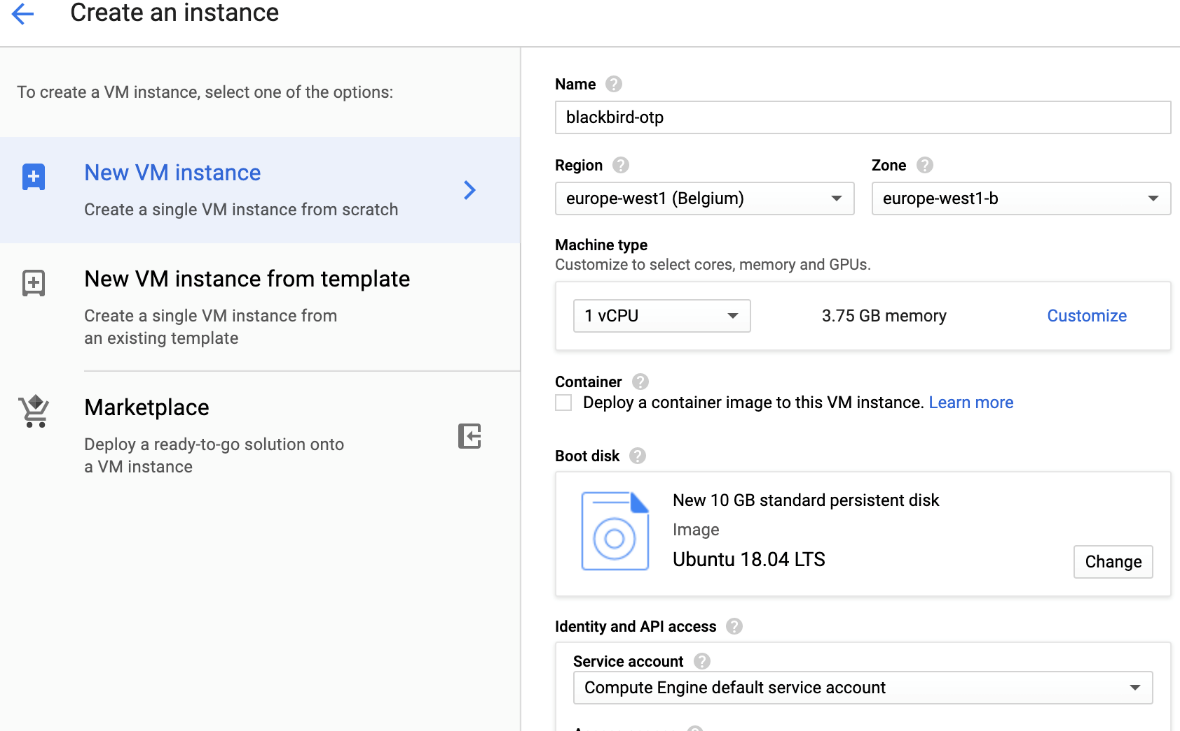


Después de esto, accederemos a la página de OpenStreetMaps para descargarnos los ficheros OSM del área de la Comunidad de Madrid. Estos ficheros llevan los datos de mapas de la zona con las calles, necesarios para que OpenTripPlanner pueda procesar tanto los trasbordos entre estaciones, cómo por dónde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



#### 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.

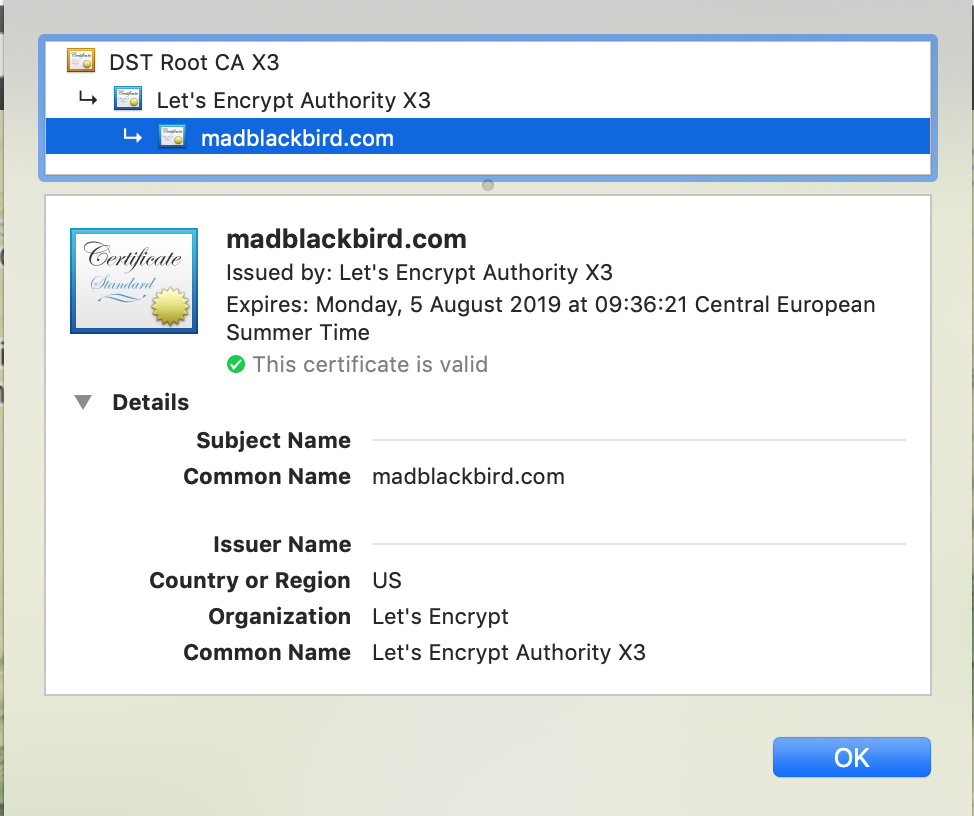


Una vez creado el nuevo servidor, procederemos a actualizar y descargar los paquetes necesarios, configurar el JDK, y descargarnos el proyecto OpenTripPlanner mediante git. Una vez descargado, le haremos los cambios pertinentes, y pasaremos a configurar el servicio mediante el que se arrancará junto 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.eyJ1c2VyX3Rva2VuIjoiS1hXTUIyU0ZL QURHMyIsImxvZ2luX2NvdW50Ijo0fQ.ALaHBKepW8wpe9jI\_dTYmPOkeKceEWiR ofnoBlMuHrA' \

--cookie '\_limebike- web\_session=bllobmRZWWcxZ3ZKVVhRNzNjbDE5K2lJeG1QVk4vWDIvdzVSOX 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**

} } ]

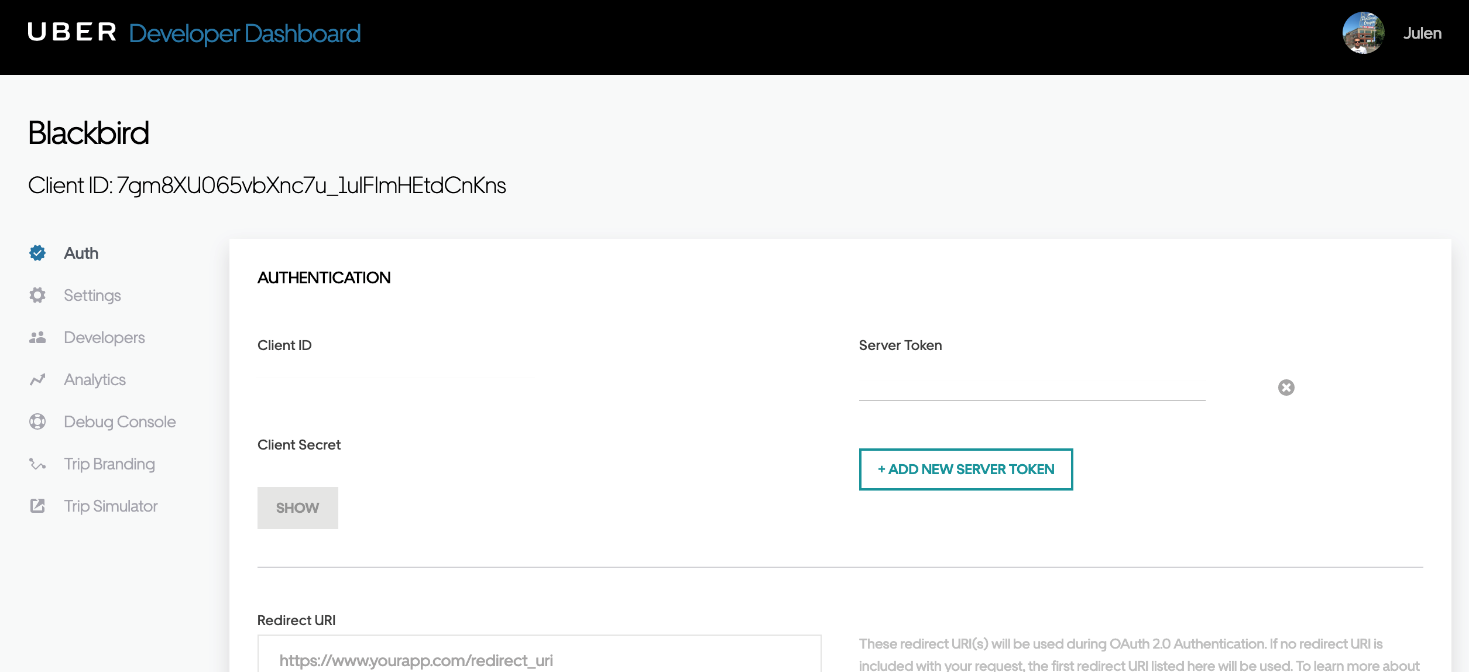
Esta investigación fue interesante, y pudimos recuperar satisfactoriamente datos que no están abiertos, pero pese a esto, y por la manera en la que se consigue el token de acceso a la API, no nos fue posible integrarlo en una aplicación a modo de producción.

* 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.



### 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 cuánto 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:

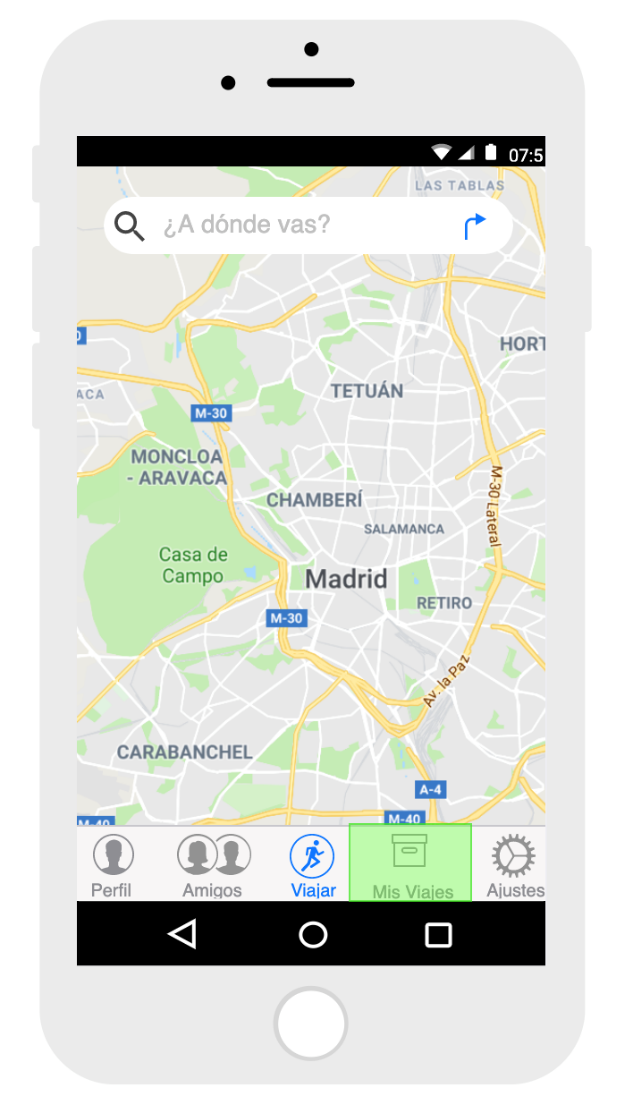
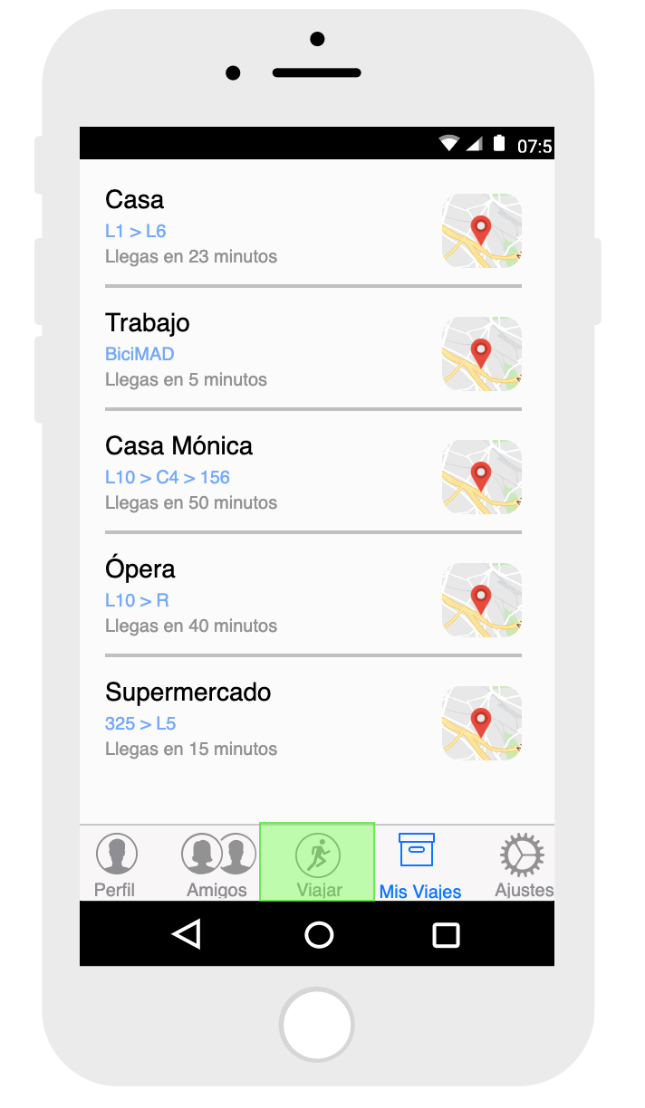
|  |
| --- |
| **public void** openUberApp(PriceEstimate priceEstimate, OTPPlace to) {  **try** {  Uri deeplink = Uri.*parse*(**"uber://?client\_id="** + **context**.getString(R.string.***uber\_client\_id***) +  **"&action=setPickup&pickup=my\_location"** +  **"&dropoff[latitude]="** + to.getLat() +  **"&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

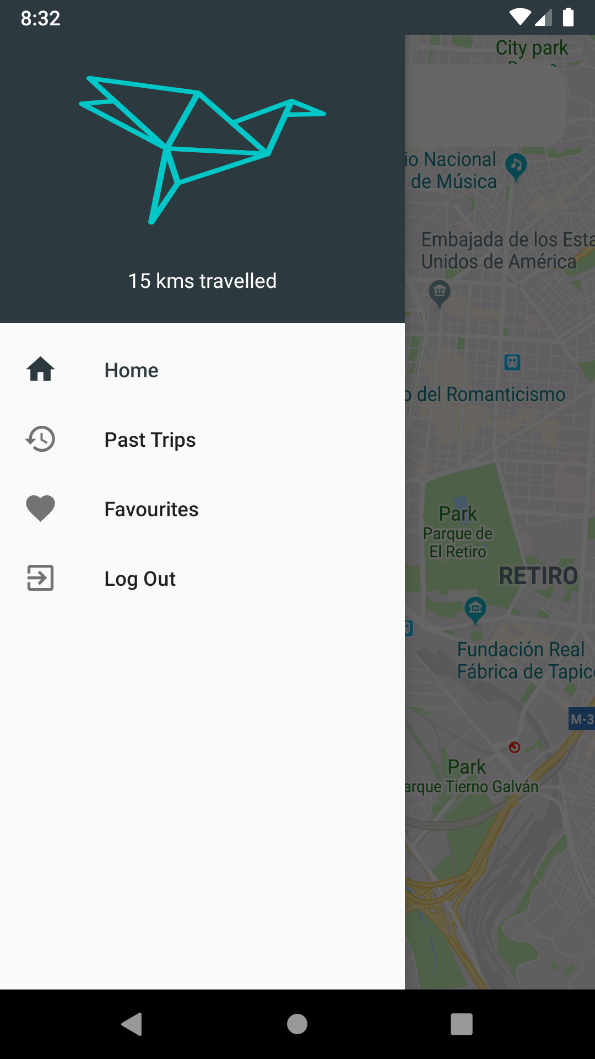
En cuanto al diseño, ya que la mayoría de usuarios que van a utilizar esta aplicación, habrán utilizado ya otras como Google Maps o Citymapper, las cuales tienen ciertas características en común, hemos optado por crera una interfaz lo más familiar posible, que el usuario sepa utilizar desde el primer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

## 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 cual 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.

# 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.

## 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.

## 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.

# 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](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](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/1505997762_968128.html>

# ANEXOS

En los anexos se recoge información técnica subsidiaria al trabajo realizado y que no tiene cabida directa en el esquema presentado hasta ahora. Es el lugar ideal para los listados de código, esquemas de circuitos, demostración de teoremas, etc.

Numerar con letras mayúsculas (pueden tener (sub)secciones: A.1, A.3.1, etc.). Incluir en el anexo todo aquello que ocupe muchas páginas y tenga una estructura repetitiva que pueda hacer tediosa la lectura (resumir en el texto principal y referenciar el anexo). El Manual de Usuario puede incluirse como un anexo o como un documento aparte (comenzando en la página 1 o no). Debería ser una guía de uso donde se explica cómo utilizar cada funcionalidad que ofrezca el sistema, explicando los pasos a dar y apoyándonos masivamente en pantallazos.

Consideraciones importantes para la elaboración de la MEMORIA

Lo más importante de un trabajo es sin duda el trabajo en sí y sus resultados, sin embargo, un buen trabajo mal presentado arrojará dudas sobre lo allí expuesto y sobre la/s persona/s que han realizado el trabajo y la memoria. Por lo tanto resulta conveniente cuidar la presentación del mismo:

* Hay que evitar radicalmente las faltas de ortografía y gramaticales. Así como cuidar la formalidad del texto (no escribir como se habla y no escribir en primera persona “He considerado...” mejor “Se considera...” o “Hay o habría que considerar...”). Un trabajo presentado con faltas de ortografía, aunque sea bueno, dará una impresión lamentable.
* Es mejor utilizar un tipo de letra estándar y un tamaño también estándar. El tamaño de letra debe estar entre 10 y 12 pt., y la fuente puede ser Verdana, Arial o similar. Por lo general la letra Verdana de 12 pt. suele ser la más utilizada. Nunca hay que usar tipos raros o poco legibles y nunca tamaños inferiores a 10 pt. (10 pt. ya es realmente pequeño) ni superiores a 12 pt.
* Los listados de los programas se hacen con una fuente que tenga espaciado fijo (Courier por ejemplo). Estas fuentes suelen ser más grandes que las normales por lo que puede ser interesante cambiar también el tamaño de letra y dejarlo entre 8 y 10 pt.
* El interlineado debe ser simple y nunca menor de una línea. En algunos casos se puede agrandar algo pero sin sobrepasar la mitad del interlineado simple (1.5 líneas).
* En todos los trabajos resulta interesante incluir un índice con los capítulos, secciones y subsecciones que contenga, así como la página en la que se encuentra.
* La inclusión de imágenes ilustrativas y figuras -sin abusar- ayudan a la claridad y mejoran la presentación de un trabajo.

Directrices particulares para el proyecto

* La presentación es muy importante precisamente porque hace brillar su contenido cuando es valioso.
* En la portada añadir el título del proyecto y los integrantes.
* No dejar hojas completas en blanco como separación, más allá de las que puedan aparecer en este documento.
* Abrir una nueva página al comenzar una nueva sección, salvo para evitar que un título quede suelto. Hacer un salto de página para empezar una nueva sección.
* La extensión del proyecto debe ser **entre 40 y 60 páginas** incluyendo todo el trabajo realizado, las conclusiones, etc., sin extenderse en detalles superfluos.

Directrices particulares para los listados de programas

En los proyectos es bastante común la realización de memorias sobre la creación de algún programa, da igual el lenguaje, que haga alguna cosa. En estos casos las directrices dadas en los puntos anteriores son de obligada aplicación, aunque resulta conveniente resaltar los siguientes aspectos:

* El listado del programa debe estar comentado siempre. Si un programa no tiene comentarios casi es mejor no incluirlo.
* Un listado del programa, por muy bien comentado que esté, no es por sí solo la memoria del proyecto, de hecho debería ser una de las partes menos importantes.
* Si se incluye el listado completo del programa se pondrá en los anexos al final de la memoria. Sólo en el caso de que el programa sea muy pequeño (apenas una rutina) se puede poner en la parte de desarrollo o resultados.
* Si el listado es muy grande (más de 3 ó 4 páginas) es mejor no incluirlo o ponerlo aparte en un cd. También se puede intentar reducir el tamaño de letra, los espaciados, etc.
* En la parte de desarrollo o resultados se describe el funcionamiento del programa, por qué se ha hecho así, se explicará si funciona bien o no, bajo qué casos, etc. En estos apartados se pueden incluir trozos del listado del programa, pero sólo aquellas partes significativas para la explicación. Si el trozo que se incluye ocupa una página o más entonces no es aconsejable ponerlo.
* Los listados, tanto si son fragmentos como si es el listado global, se deben hacer con un tipo de letra de tamaño fijo (por ejemplo Consolas). Por otro lado, el tamaño debe ser lo menor posible para que ocupe poco espacio.
* Es recomendables insertar los listados de código en una tabla e insertarse un título (Referencias -> Insertar Título), si hay muchos listados puede ser interesante añadir un Índice de listados.

Algunas reglas mecanográficas

Estas reglas son casi tan importantes como las ortográficas y su incumplimiento produce una mala impresión del trabajo que se está realizando. Se listan a continuación algunas de las más importantes o utilizadas:

* Los signos de puntuación como los puntos, comas, dos puntos, etc., van siempre unidos a la palabra que preceda; es decir, nunca hay que poner un espacio delante de cualquiera de estos signos.
* En cuanto a los signos que se abren y cierran como las comillas, los paréntesis, las llaves, etc., el signo que abre debe ir unido a la palabra que sigue, y el signo que cierra debe ir unido a la palabra precedente; es decir, detrás de un signo que abre nunca va un espacio al igual que delante de un símbolo de cierre donde tampoco se pone espacio.
* Los títulos de las secciones o subsecciones no deben ir sueltos. Si no hay espacio en la página para empezar una sección se debe empezar en página nueva.