Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería Informática

Trabajo Fin de Grado

Aplicación Android para Buses urbanos de Guadalajara

ESCUELA POLITECNICA SUPERIOR

Autor: Álvaro Yélamos San Andrés
Tutor: José María Gutiérrez Martínez

2018

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

GRADO EN INGENIERÍA INFORMÁTICA

Trabajo Fin de Grado

"APLICACIÓN ANDROID PARA BUSES URBANOS DE GUADALAJARA"

Autor: **Álvaro Yélamos San Andrés**

Director: **José María Gutiérrez Martínez**

Tribunal:

Presidente: .....

Vocal 1º: .....

Vocal 2º: .....

Fecha: .... de ..... de .....

Resumen

Este proyecto consiste en el desarrollo de una aplicación móvil para consultar información sobre la red de autobuses urbanos de Guadalajara. Para ello primero hemos analizado la aplicación oficial del sistema de buses para evitar sufrir los problemas de diseño que muestra y descubrir cómo debemos comunicarnos con su API. Adicionalmente hemos desarrollado un sistema de monitorización de los autobuses activos para complementar la información que nos suministra la infraestructura de autobuses. Esta información nos permitirá predecir la hora aproximada a la que normalmente llega el vehículo a cada parada de su recorrido.

Palabras clave

Android, Google Maps, SQLite, Autobús, Monitorización.

Abstract

This project involves the development of a mobile app to search for information about the city bus network of Guadalajara. To achieve this, we have analyzed the official bus app to avoid making the same design mistakes and to learn how to communicate with their API. Additionally, we have developed an active bus monitoring system to supplement the information of the bus infrastructure. This extra data will allow us to predict the approximate time at which each vehicle normally arrives at each stop along its route.

Keywords

Android, Google Maps, SQLite, Bus, Monitoring.

Glosario de acrónimos

Línea/Ruta: Representa un recorrido de un punto A a un punto B en cualquier sentido de la dirección.

Itinerario: Recorrido en un sentido de la dirección de una línea.

Expedición: Hora y día de la semana a la que está programada que un bus inicie el recorrido de un itinerario.

API (**Interfaz de Programación de Aplicaciones**): Conjunto de funciones, métodos o procedimientos que ofrece una librería para ser utilizado por otro programa como una capa de abstracción.

Web Service (Servicio Web): Tecnología que utiliza un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones.

JSF (Java Server Faces): Tecnología y framework para aplicaciones Java basadas en web que simplifica el desarrollo de interfaces de usuario en aplicaciones Java EE.

1. Introducción y planteamiento

En los últimos años las redes de comunicaciones móviles se han convertido en un recurso accesibles al público general. Gracias a la disponibilidad y velocidad que ofrecen las conexiones 3G y 4G podemos consultar en tiempo real información actualizada que nos ayude a tomar decisiones en nuestro día a día. Esta innovación tecnológica ha abierto un mundo de posibilidades en que su gradual aplicación a distintos elementos cotidianos supone una gran mejora a nuestras vidas. Estos progresos han afectado a todos los ámbitos imaginables, desde el entretenimiento y compras a la meteorología o navegación GPS. En este proyecto nos interesa únicamente aplicada al entorno del transporte.

Antiguamente cuando una persona deseaba desplazarse a otro lugar debía interpretar un mapa físico para obtener la información de cómo alcanzar su destino. En las últimas décadas gracias a la aparición de nuevas tecnologías como las mencionadas anteriormente es posible facilitar esta tarea y consultar esta misma información de una forma más rápida, sencilla y fiable. Estas facilidades se deben principalmente a la posibilidad de conocer la posición del usuario en el mapa y obtener una estimación del tiempo que tomarían las distintas opciones de transporte disponibles para llegar a su destino desde su posición actual.

Un servicio de transporte como es el de autobuses urbanos debería ofrecer la mayor calidad posible a sus usuarios ya que, como servicio público que es, está planteado para que beneficie a toda la sociedad independientemente de las circunstancias personales de los distintos individuos. Con esta premisa en mente podemos determinar que la aplicación de transportes que ha motivado este proyecto (autobuses urbanos de Guadalajara, España) es insuficiente y mejorable en muchos aspectos.

El objetivo de este proyecto consiste en el desarrollo de una aplicación móvil que solvente las carencias de la aplicación oficial, cubra casos de uso adicionales y tenga un diseño más usable. Para ello deberemos disponer de recursos adicionales como una base de datos alimentada por un sistema de monitorización adicional para poder consultar estimaciones aproximadas sin conexión y una interfaz mejorada con nuevos elementos gráficos que mejoren la experiencia del usuario.

2. Objetivos del proyecto

2.1 Principales

1- Desarrollar una aplicación para la consulta de información sobre el servicio público de autobuses de Guadalajara (España).
2- La aplicación debe mostrar información en tiempo real del tiempo de llegada de los buses a las distintas paradas. Esta información se obtendrá del propio sistema de autobuses.
3- La aplicación debe mostrar información sobre las distintas líneas de buses sin necesidad de que el usuario conozca los códigos de las paradas.
4- La aplicación debe permitir al usuario marcar las paradas que desee como favoritas para facilitar su uso.
5- Desarrollar un sistema para obtener datos sin conexión adicionales sobre tiempos de llegada que no proporciona la API del servicio de buses.

2.2 Secundarios

1- Mostrar la posición de los distintos buses de la línea en tiempo real.

3. Estado del arte

3.1 Software

Sistema Operativo Android

Android es un sistema operativo basado en el núcleo de Linux diseñado para dispositivos móviles. Desarrollado por Android Inc., compañía que sería adquirida por Google en 2005. Es un sistema abierto y multitarea que proporciona interfaces para permitir el desarrollo de aplicaciones que accedan a las funciones del teléfono usando el lenguaje de programación Java. En el año 2017 se añadió oficialmente la posibilidad de programar en Kotlin después de que Google lo aceptara como lenguaje oficial.

Desde el nacimiento del proyecto el sistema ha ido evolucionando en sus distintas versiones, a continuación, se indican las principales mejoras que incluyeron junto al año de lanzamiento de las versiones más importantes. Todas están documentadas en detalle en el portal oficial.

Lanzamiento	Versión	Mejoras principales
2009	1.6 Donut	- Cuadro de búsqueda rápida - Diversidad de tamaños de pantalla - Android market
2009	2.1 Eclair	- Google Maps navigation - Pantalla de inicio personalizada - Síntesis de voz
2010	2.2 Froyo	- Acciones de voz - Zona Wi-Fi portátil (tethering) - Mejora en rendimiento
2010	2.3 Gingerbread	- API para el desarrollo de juegos (OpenGL) - NFC (Near Field Communication) - Gestión y rendimiento de la batería
2011	3.0 Honeycomb	- Optimización del diseño para tablets. - Barra del sistema (botones de navegación en la interfaz en lugar de depender de botones físicos). - Ajustes rápidos, facilitando el acceso a la información más esencial
2011	4.0 Ice-Cream sándwich	- Mayor personalización de la pantalla de inicio - Control de uso de datos móviles - Android Beam (compartir contenido usando NFC)
2012	4.1 Jelly Bean	- Google Now - Notificaciones accionables - Multiusuario
2013	4.4 KitKat	- Comandos por voz (OK Google) - Diseño envolvente - Mejoras de usabilidad: Destaca acciones realizadas comúnmente, un filtro básico de spam…
2014	5.0 Lollipop	- Material Design - Multipantalla entre dispositivos Android - Notificaciones interactivas desde la pantalla de bloqueo
2015	6.0 Marshmallow	- Google Now accesible desde cualquier aplicación

		- Mejora en sistema de permisos, ahora las aplicaciones deberán solicitarlo activamente en lugar de únicamente en la instalación. - Optimización del uso de la batería
2016	7.0 Nougat	- Multiventana - Realidad virtual básica - Cifrado a nivel de archivos
2017	8.0 Oreo	- Imagen en Imagen (PiP) - Mejoras en las notificaciones - Eliminado "orígenes desconocidos" a favor de que el usuario autorice las instalaciones individuales de apps no oficiales.

La estructura general de directorios de una aplicación Android se puede encontrar en la documentación inicial. Las carpetas principales son las siguientes:

- **Manifest**: Fichero en que se declaran los permisos que requiere la aplicación, la versión mínima para que funcione la app y otros datos relacionados con las actividades de la app… etc.
- **/java**: Contiene los archivos de código fuente de Java separados por nombre de paquetes, incluido el código de prueba Junit.
- **/res**: Contiene todos los recursos sin código, como los diseños de las interfaces en formato XML, literales mostrados en la interfaz, imágenes… etc. Estos recursos están agrupados en distintas subcarpetas.

Android Studio

Android Studio es un entorno de desarrollo integrado destinado a la plataforma móvil Android. Fue presentado como la alternativa oficial al plugin de Eclipse en mayo de 2013 como el IDE oficial para el desarrollo de aplicaciones para Android. Está basado en el software IntelliJ IDEA de JetBrains.

Las herramientas que lo hacen el mejor entorno de desarrollo disponible son las siguientes:

- **ADB (Android Debug Drive)**: Herramienta de línea de comando que permite comunicarse con un dispositivo Android virtual (mediante emulador) o físico conectado al ordenador. ADB nos da acceso a una Shell Unix del dispositivo conectado, de modo que podemos acceder a sus recursos, instalar y depurar aplicaciones mediante la línea de comandos…etc. Si queremos usar esta herramienta con nuestro dispositivo físico primero deberemos configurar nuestro smartphone activando las "opciones de desarrollador" dentro del menú de configuración.

- **AVD (Android Virtual Device)**: Dispositivo Virtual Android. Para poder probar correctamente nuestra aplicación deberemos ejecutarla en las distintas versiones objetivo de Android para observar cómo se comporta. Para ello crearemos un dispositivo virtual con las características que queremos probar, como el tamaño de pantalla, la versión del sistema operativo o si queremos que disponga de los servicios de Google play.

- **Apksigner**: Facilita la firma de las aplicaciones y verifica que esta firma será reconocida por todas las versiones de Android.

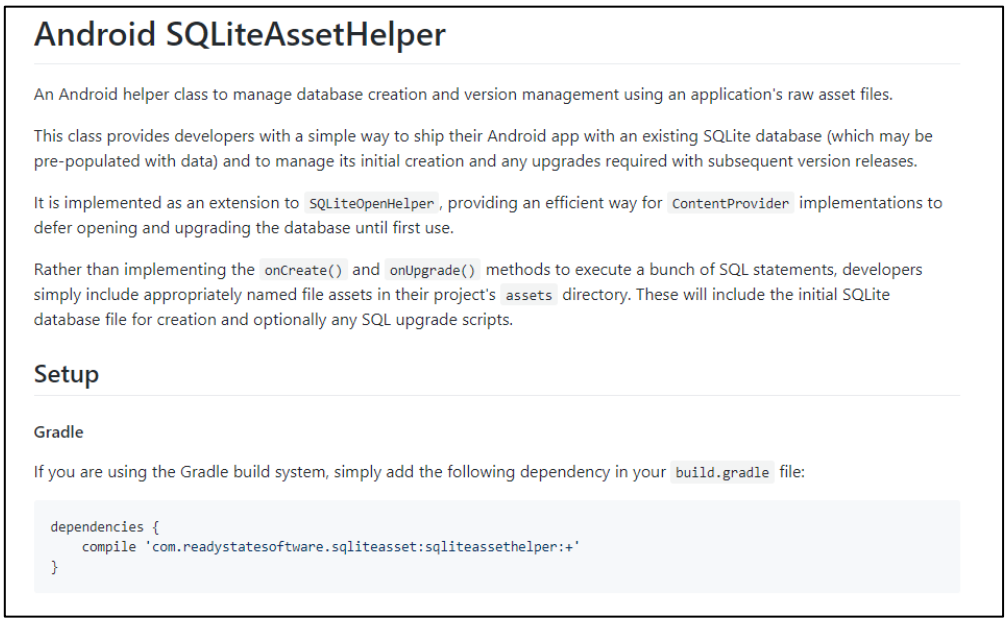
- **Gradle**: Herramienta para automatizar la construcción de los proyectos que permite configurar y administrar la compilación de estos y gestionar sus dependencias.

- **Editor gráfico**: Muestra una vista previa de los cambios que realicemos sobre los archivos XML que definen las interfaces.

Android SQLiteAssetHelper

Clase helper que permite cargar una base de datos SQLite externa en una aplicación Android, además de facilitar el control y actualización de las distintas versiones de bases de datos. Esto facilita enormemente el trabajo ya que sin esta clase de ayuda tendríamos que poblar la base de datos dentro de la propia aplicación mediante queries SQL. Para su uso basta con añadirla como dependencia en el archivo build.gradle de nuestro proyecto.

https://github.com/jgilfelt/android-sqlite-asset-helper



Android SQLiteAssetHelper

An Android helper class to manage database creation and version management using an application's raw asset files.

This class provides developers with a simple way to ship their Android app with an existing SQLite database (which may be pre-populated with data) and to manage its initial creation and any upgrades required with subsequent version releases.

It is implemented as an extension to `SQLiteOpenHelper`, providing an efficient way for `ContentProvider` implementations to defer opening and upgrading the database until first use.

Rather than implementing the `onCreate()` and `onUpgrade()` methods to execute a bunch of SQL statements, developers simply include appropriately named file assets in their project's `assets` directory. These will include the initial SQLite database file for creation and optionally any SQL upgrade scripts.

Setup

Gradle

If you are using the Gradle build system, simply add the following dependency in your `build.gradle` file:

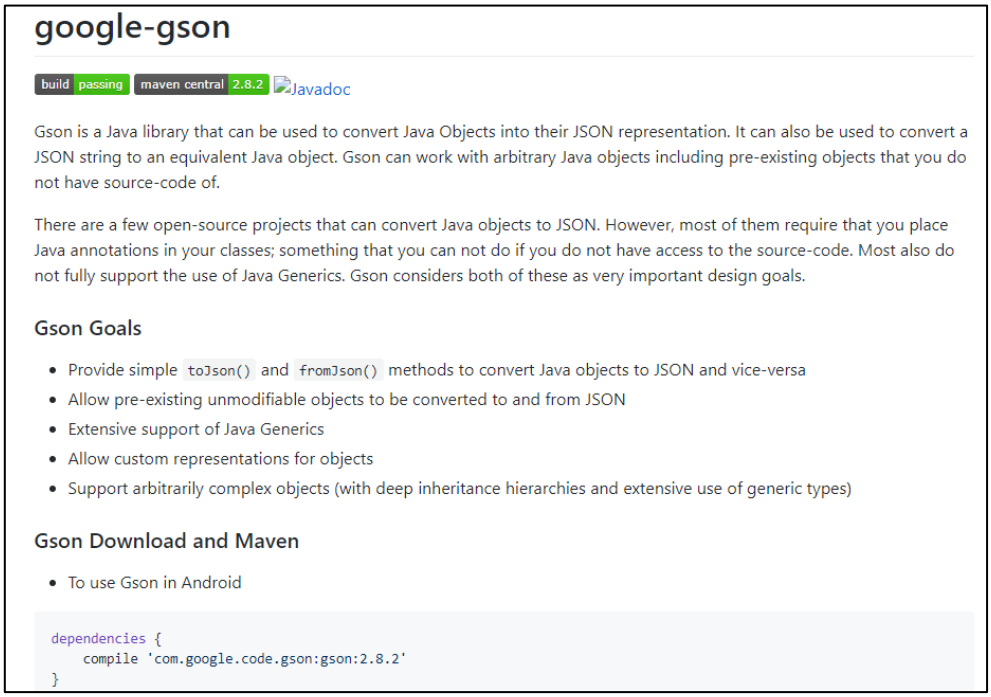
```
dependencies {
    compile 'com.readystatesoftware.sqliteasset:sqliteassethelper:+'
}
```

Librería GSON

Librería de código abierto para Java que permite la serialización y deserialización entre objetos Java y su representación en notación JSON. Gracias a esta librería se ahorra tiempo de desarrollo cuando se trabaja con información en formato JSON, como es el caso de la API del sistema de autobuses. En lugar de tener que programar manualmente el deserializador indicando qué clave corresponde a qué atributo del objeto Java campo por campo, GSON facilita enormemente la tarea mapeando estos

valores automáticamente con una clase Java que le suministremos (para que funcione automáticamente deben coincidir clave JSON y nombre de atributo).

https://github.com/google/gson



google-gson

build passing maven central 2.8.2 Javadoc

Gson is a Java library that can be used to convert Java Objects into their JSON representation. It can also be used to convert a JSON string to an equivalent Java object. Gson can work with arbitrary Java objects including pre-existing objects that you do not have source-code of.

There are a few open-source projects that can convert Java objects to JSON. However, most of them require that you place Java annotations in your classes; something that you can not do if you do not have access to the source-code. Most also do not fully support the use of Java Generics. Gson considers both of these as very important design goals.

Gson Goals

- Provide simple `toJson()` and `fromJson()` methods to convert Java objects to JSON and vice-versa
- Allow pre-existing unmodifiable objects to be converted to and from JSON
- Extensive support of Java Generics
- Allow custom representations for objects
- Support arbitrarily complex objects (with deep inheritance hierarchies and extensive use of generic types)

Gson Download and Maven

- To use Gson in Android

```
dependencies {
    compile 'com.google.code.gson:gson:2.8.2'
}
```

SQLite

SQLite es un motor de bases de datos de código abierto, transaccional, que no necesita configuración ni requiere un servidor y se caracteriza por mantener el almacenamiento de información de forma sencilla. Es una tecnología cómoda y ligera para dispositivos móviles a pesar de que tiene limitaciones en determinadas operaciones.

SQLiteStudio

SQliteStudio es un programa de código abierto para la gestión de bases de datos SQLite que proporciona una interfaz gráfica intuitiva que facilita la creación y consulta de este tipo de bases de datos. Este software está disponible para las plataformas más comunes: Windows, MacOSX y Linux. Las ventajas de este software respecto a otros similares son:

- Permite la exportación de datos a distintos formatos (CSV, HTML, XML, PDF, JSON) y la importación de datos a través de un fichero CSV.
- No requiere instalación para su uso.
- Admite complementos oficiales y de terceros para expandir las funcionalidades. Cabe destacar el complemento "DbAndroid", que permite acceder directamente a bases de datos en el dispositivo Android, en lugar de tener que modificar la base de datos en una carpeta local para después enviar esta al dispositivo.

En el proyecto se ha usado para la creación y consulta de las tablas de la base de datos en que hemos almacenado los datos estáticos del sistema de autobuses, como pueden ser los nombres y posiciones de las distintas paradas.

Git

Git es un software de control de versiones de código abierto disponible para múltiples plataformas (Mac OS X, Windows, Linux y Solaris). Como herramienta de control de versione que es, sirve para llevar un registro de los cambios efectuados sobre los archivos que componen la aplicación y además facilitar la coordinación en el desarrollo cuando varias personas están involucradas en el mismo proyecto. Se ha usado para gestionar el repositorio de la aplicación móvil y el programa de monitorización a través de Github y Android Studio.

Github

GitHub es una plataforma de desarrollo colaborativo para alojar software utilizando el control de versiones Git. Con las cuentas gratuitas solo se pueden crear proyectos públicos, pero pagando una pequeña cuota anual para obtener una cuenta premium permiten crear repositorios privados. Android Studio dispone de una opción de menú para la integración con esta plataforma de modo que facilita mucho la gestión del desarrollo de la aplicación móvil.

Netbeans

NetBeans es un entorno de desarrollo integrado libre y gratuito con un gran número de módulos para extender sus funcionalidades. Es uno de los entornos de desarrollo más utilizados para la programación en el lenguaje Java junto con Eclipse. La principal ventaja de este frente a su competidor es que Netbeans dispone de un diseñador gráfico de interfaces por defecto, sin que sea necesario instalar ningún plugin adicional.

Se ha utilizado para la programación de la aplicación Java de escritorio que alimenta una base de datos con la información de todo el sistema y además monitoriza la actividad de los distintos autobuses que componen la red urbana.

Librería SQLITE JDBC

Librería que permite interactuar con una base de datos SQLite usando el lenguaje Java. Para poder usar la base de datos se deben configurar los parámetros de conexión a través de la API. Una vez establecida la conexión se puede realizar cualquier tipo de tarea con la base de datos para la que tenga permisos: consulta, actualización, creación, modificación y borrado de tablas, ejecución de procedimientos almacenados en la base de datos… etc.

Esta librería es indispensable para el funcionamiento de la aplicación de escritorio, ya que sin ella no podría realizar la carga inicial de los datos estáticos, ni almacenar los resultados fruto de la monitorización del sistema.

TOAD Data Modeler

Toad Data Modeler es una herramienta de modelado, diseño y documentación de bases de datos compatible con una gran variedad de DBRMs (MySQL, Oracle, SQL Server, PostgreSQL, DB2…). Permite construir modelos de datos lógicos y físicos, comparar varios modelos de base de datos entre sí y generar consultas SQL complejas. Una de sus funcionalidades añadidas más interesantes es que permite hacer ingeniería inversa a una base de datos, es decir, obtener el diagrama de una base de datos ya existente.

Se utilizó principalmente durante la etapa de análisis para crear y modificar los diagramas de tablas necesarias para conseguir cumplir con los objetivos del proyecto. Las imágenes de los esquemas de la base de datos de la documentación se generaron con este programa.

Selenium

Entorno de pruebas de software de código abierto para aplicaciones basadas en la web. Gracias a este framework se pueden realizar pruebas funcionales de aplicaciones, simulando el comportamiento que tendría un usuario real en un navegador. Debido a esta capacidad de simulación también se usa para la programación de bots o crawlers para obtener información de aplicaciones web de forma automatizada.

Además, provee la herramienta Selenium IDE para grabar y reproducir el comportamiento sobre la aplicación y así crear pruebas sin necesidad de tener que programarlas usando un lenguaje de scripting.

3.2 Hardware

Dispositivos móviles Android.

Se ha optado por usar un dispositivo físico para depurar en tiempo real la aplicación. Los dispositivos utilizados para el desarrollo han sido un "Alps W450" con sistema operativo Android 4.4 y un Xiaomi Redmi 4A con la versión 6.0.1 de Android.

4. Motivación del proyecto

4.1 La aplicación oficial

Este proyecto nace debido a las carencias de la aplicación oficial de la empresa de autobuses. La aplicación oficial únicamente contempla los siguientes dos casos de uso:

1- **Obtener información sobre una parada**: El usuario tiene dos opciones:
   a. Introducir manualmente el código numérico de la parada de bus de la que desea obtener información. Estos códigos se pueden obtener de dos formas:
      i. Leyéndolos de la esquina superior izquierda de la marquesina de la propia parada.
      ii. Consultando la documentación de la página web oficial con un navegador web. Desde esta página se puede usar un buscador por línea además de un buscador por código de parada/Nombre, además de disponer de enlaces a otra documentación en formato PDF. No se proporciona un mapa interactivo con las paradas.
   b. Leer el código QR usando la opción "Lectura de código QR". Estos códigos se encuentran impresos en las marquesinas de las paradas.

2- **Obtener información sobre un autobús**: El usuario introduce un código de identificación asociado a un autobús particular. Para conocer este número el usuario debe leer el número que se muestra en la esquina inferior derecha de la parte frontal exterior del autobús. Este código está impreso adicionalmente en un soporte superior que se encuentra en el interior de este.

4.2 Análisis de los problemas de la aplicación oficial y soluciones propuestas

En este trabajo no se entrará a analizar los errores de programación (bugs) que presenta la aplicación oficial (por ejemplo, "pop ups" de carga mostrado indefinidamente sin actualización de la interfaz…) ya que estos errores son relativamente fáciles de subsanar. Aunque se solucionaran estos problemas la aplicación seguiría siendo igual de inútil para el usuario de autobuses que en su versión actual, únicamente mejoraría ligeramente la experiencia general del usuario.

Nuestro objetivo es centrarnos en el problema más grave de la aplicación, en la razón que se puede extraer de los comentarios y puntuaciones negativas dentro de la tienda de aplicaciones Android: No se ha pensado en ningún momento en el usuario.

En la web de la aplicación de Android se puede leer los siguientes comentarios de los usuarios:

- "No hay mapa de líneas. No sirve para mucho.",

- "Faltan muchas opciones...".

Incluso un usuario que la puntuó con tres estrellas sobre cinco señala que "Estaría ideal un mapa con los números de paradas, planos, rutas y horarios.". Adicionalmente entrevisté en persona a cinco usuarios de la aplicación que me transmitieron las mismas quejas descritas anteriormente. Todos estos comentarios evidencian que no cumple con lo que se espera de una aplicación de transportes: Facilitar el uso del propio servicio de transportes.

A continuación, hago un listado de los principales problemas que sufre la aplicación y las soluciones que propongo:

1. **Obligar al usuario a personarse físicamente en la parada o a consultar otras páginas web**. Cuando el usuario desee consultar el tiempo de espera para una parada la aplicación le pedirá el código de la parada. Estos códigos, como se ha expuesto en la sección "Motivación del proyecto" se deben obtener personándose en la parada para leerlos o a través de una web externa en la que se puede filtrar por nombre de calle y línea. Una vez haya descubierto cual es el código de la parada tendrá que memorizar mentalmente el código, ya que la aplicación no ofrece ninguna forma de guardar los códigos introducidos.

2. **Para obtener información** sobre una línea o los horarios de esta **el usuario debe conocer el código de alguna parada relacionada** con esa línea.

3. **Obligatoriedad de disponer de una conexión a Internet**.
   En caso de no tener contratada una tarifa de datos o no tener cobertura o acceso a una red Wi-Fi el usuario no podrá usar la aplicación.

4. **No ofrecer un mapa para la consulta**.
   La documentación de la página oficial incluye un documento PDF con el mapa y todas las paradas y líneas. Debido al formato de esta documentación el usuario debe buscar en el mapa manualmente la parada deseada en lugar de facilitar el acceso a la información con un mapa interactivo, como por ejemplo Google Maps.

5. **Los horarios únicamente informan de la hora de salida desde la primera parada**
   Actualmente con los datos y el control que se tiene de la posición de los autobuses se puede calcular la hora media a la que habitualmente suele llegar a la parada cada "expedición" de cada línea. Se debe recalcar al usuario que los tiempos que calculemos son estimaciones que le ayudarán a hacerse una idea de la hora aproximada en situaciones en las que el sistema en tiempo real no esté disponible, ya sea por motivos técnicos del servidor, porque no dispone de conexión a internet o porque simplemente se desea consultar los horarios de días futuros.

6. **No mostrar la posición GPS de los buses en tiempo real**
   Para evitar depender únicamente de la estimación en minutos que nos proporcionen los servidores de la compañía sería una idea interesante mostrar, por ejemplo, todos los buses de una línea, de forma que el usuario sepa exactamente en qué punto de la ruta se encuentra su bus.
   Supongo que este era el objetivo inicial del modo "Introducción manual por bus", pero debido al mal diseño no cumple su función, ya que se pide al usuario que introduzca el código del bus que quiere consultar. Este código sólo puede conocerse cuando el bus se encuentre a la vista del usuario ya que el código está impreso en el frontal del vehículo, por lo que el usuario ya no tendrá interés en conocer la posición del bus.

7. **No favorece el uso habitual de la aplicación**
   Normalmente los usuarios consultan únicamente las paradas que usan durante su rutina diaria, por lo que se podría ayudar al usuario ahorrándole el esfuerzo de buscar manualmente la misma parada todos los días.
   Propongo añadir la funcionalidad de poder seleccionar una parada y una línea que pase por la misma como "favorita", y añadir una pantalla a la que se puede acceder desde el menú de inicio. Se ha optado por elegir la línea en particular y la parada en lugar de únicamente la parada debido a que algunas paradas tienen muchas líneas y ocuparían demasiado espacio en la pantalla de favoritos.

5. Justificación de decisiones

5.1 Aplicación móvil

Aplicaciones nativas VS Aplicaciones híbridas

A la hora de comenzar el desarrollo de una aplicación móvil debemos tomar la decisión de qué tipo de aplicación va a ser para así elegir entre un conjunto de tecnologías más pequeño. Según como se oriente el desarrollo se clasifican en grandes grupos:

1. **Nativas**
2. **Nativas Hibridas**
3. **Nativas Generadas**

1. Nativas: Desarrolladas en el lenguaje de programación propio (nativo) del dispositivo; es decir Java para Android y Objetive C o Swift para iOS. Este modelo depende enteramente de la plataforma y las aplicaciones no son portables a otras, por lo que se debe desarrollar una aplicación por plataforma.

Las principales ventajas de las aplicaciones nativas son:

- Lograr el mejor rendimiento posible.
- Conseguir un que la interfaz sea acorde al sistema operativo
- Acceso a todas las capacidades del dispositivo sin restricciones.

2. Nativas Hibridas: Desarrolladas usando tecnología web (HTML5, CSS y JavaScript) y desplegadas dentro de un contenedor nativo, como puede ser un navegador. Un ejemplo de este tipo son las aplicaciones que usan Phonegap/Cordova, las cuales tienen acceso a las capacidades del dispositivo de una forma estándar que no depende del sistema operativo. Este modelo permite una portabilidad máxima ya que funcionarán en cualquier dispositivo que disponga de estos contenedores estándar.

3. Generadas: Desarrolladas usando tecnologías como Xamarin o Genexus, su codificación se realiza usando técnicas y lenguajes específicos para la herramienta de generación de modo que después esta genere la aplicación de forma "nativa" para la plataforma objetivo. También son conocidas como falsas nativas, ya que éstas no terminan de cumplir las ventajas de las nativas que hemos descrito anteriormente. Se ha decidido programar una aplicación en código nativo del sistema para así poder acceder sin restricciones al hardware del dispositivo y poder ofrecer al usuario la mejor experiencia de usuario posible.

Sistema operativo

La aplicación será desarrollada para dispositivos Android principalmente por tres motivos:

- **Cuota de mercado en España**: El objetivo es que el mayor número de usuarios pueda acceder a nuestra aplicación, por lo que se orientará al sistema operativo que más cuota de mercado tenga en nuestro país. Según un estudio de "Kantar Insights España" el sistema operativo Android supuso un 92% de la cuota del mercado español durante los 3 meses tomados para el estudio (de febrero a abril 2017).

- **Coste de publicación de la aplicación en la tienda oficial**. La elección de SO entre los dos sistemas operativos móviles con mayor cuota de mercado (Android e IOS) cubren más del 99% del mercado total, por lo que debemos elegir entre estas dos opciones. El precio de alta como desarrollador en la tienda de aplicaciones iOS es de 99 dólares al año mientras que en Android (PlayStore) basta con un único pago de 25 dólares estadounidenses para poder ofrecer nuestra aplicación a los usuarios.

La versión Android mínima en la que se podrá ejecutar la aplicación es la 4.1 debido a que usa funciones no disponibles en anteriores al SDK correspondiente a esta versión (SDK 16). No se ha creído conveniente buscar alternativas al uso de estas funciones ya que según las estadísticas de la compañía únicamente un 0,7 % de los usuarios globales de Android quedaría excluido de usar la aplicación.

Base de datos local

Como la aplicación debe ser capaz de mostrar información sobre el sistema de autobuses sin conexión a Internet es necesario que tenga una base de datos local de la que obtener la información estática (como por ejemplo la información sobre las líneas y las ubicaciones de las paradas). Esta base de datos es SQLite ya que es el sistema de gestión de bases de datos nativo de Android. En la aplicación de escritorio se comenzó usando esta para reutilizar el código, y al final se usó la misma para la monitorización.

Si el proyecto hubiera tenido más tiempo para el desarrollo se podría haber orientado la aplicación de escritorio desde otra perspectiva, de modo que usara otro sistema de base de datos que facilite el análisis de la información recogida como puede ser MongoDB. Una vez generáramos la información deseada podríamos cargarla en la base de datos de la aplicación a través de un fichero intermedio.

5.2 Aplicación de escritorio para monitorización

Como se describe más adelante para conseguir cumplir el objetivo de "Desarrollar un sistema para obtener datos sin conexión adicionales sobre tiempos de llegada que no proporciona la API del servicio de buses" ha sido necesario el desarrollo de una aplicación adicional que monitorice los tiempos de llegadas de los autobuses de las distintas líneas. Esta aplicación se ha desarrollado en Java principalmente debido a que el desarrollador ya tiene experiencia en este lenguaje, y a que se puedan reutilizar código de la aplicación Android, como por ejemplo el módulo que convierte la información obtenida de la API a un objeto Java.

6. API del sistema de autobuses

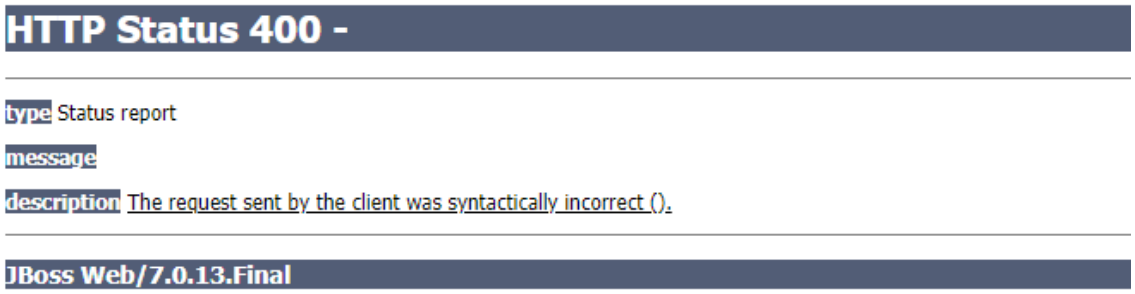
6.1 Servidor de la API

Para obtener la información del sistema de buses debemos usar su servicio web. Un servicio web es una tecnología que sirve para intercambiar datos entre aplicaciones mediante unos protocolos y estándares determinados. Esto permite que distintas aplicaciones desarrolladas en lenguajes totalmente distintos pueden intercambiar datos, abriendo un mundo de posibilidades a las comunicaciones máquina-máquina.

Las principales ventajas que proporcionan los servicios web son:

- Interoperabilidad total entre aplicaciones independientemente de sus características o las de su plataforma.
- Facilitan el acceso al contenido y el funcionamiento de los protocolos al estar basados en texto.
- Permiten que servicios y software de diferentes compañías se puedan integrar y combinar fácilmente.

Por la información que hemos podido obtener de los mensajes de error del servidor podemos deducir que la API usa la versión 7.0.13.Final de JBoss, proyecto que fue renombrado a WildFly a finales de 2014.



Wildfly/JBoss es un servidor de aplicaciones Java EE de código libre y abierto implementado en Java puro. Las principales características de WildFly son:

- **Escalabilidad:** Permite escalar el sistema hasta más de un millón de conexiones mediante el uso de herramientas propias, como Undertow.
- **Optimización en el uso de memoria:** La gestión que hace de la memoria permite minimizar la asignación de posiciones, evitando la carga de clases duplicadas y objetos, aligerando de este modo el peso que recae sobre el recolector de basura de Java.
- **Basado en estándares:** Aprovecha las nuevas mejoras de Java EE7 y soporta los últimos estándares de la web como REST, JAX-RS 2 y JSON-P.
- **Modular**: Su arquitectura permite crear módulos aislados, enlazando únicamente con los distintos JAR cuando la aplicación lo necesita.

Todas las peticiones de la API únicamente aceptan peticiones HTTP GET. La información de las respuestas siempre está en formato JSON. Las distintas funciones disponibles están detalladas en la al final de esta página.

6.2 JSON

Acrónimo de JavaScript Object Notation, es un formato de texto para representar datos estructurados en la sintaxis de objetos de JavaScript. El uso más común que se le da es el de transmitir datos entre aplicaciones web y el servidor ya sea a través de AJAX o como respuesta a una petición HTTP contra un Servicio web.

A raíz de las limitaciones de JSON han nacido otros formatos como por ejemplo YAML, que es más legible para el ojo humano y robusto cuando se combina con otros formatos de serialización.

6.3 Funciones

El significado de los códigos de intervalo y de los puntos que recibimos como respuesta en las distintas funciones puede encontrarse en el anexo 1 y anexo 2 respectivamente.

Función "Información de la posición de un bus activo"

Esta función nos informa sobre la posición del autobús consultado, el itinerario que está siguiendo y la parada a la que se dirige en el momento de la consulta. Opcionalmente se puede solicitar la lista de coordenadas geográficas del itinerario que está efectuando en el momento. Estas coordenadas recibidas están clasificadas en distintos tipos, el significado de estos tipos está detallado en el anexo 2.

En caso de que el código del vehículo enviado no exista, o no esté en activo la API devuelve el mensaje "NO DATOS". Al no distinguir entre estos dos casos se complica la búsqueda de los códigos de autobuses existentes, dificultando la monitorización como se explica en la sección "Monitorización de los tiempos de los autobuses".

Endpoint: /SSIIMovilWS/ws/cons/datosVehiculo.json

Método HTTP de la petición: GET

Parámetros de la petición:

- codVehiculo: Entero, correspondiente al código del autobús que está impreso en la esquina inferior izquierda del vehículo.
- codEmpresa: Entero, únicamente afecta a los datos recibidos en que la propiedad "idEmpresa" tendrá el mismo valor que le enviemos en este parámetro.
- petItinerario: {true,false} Indica si se desea que la respuesta incluya un array todas las coordenadas del itinerario que está llevando a cabo el bus.

Estructura de la respuesta

Definida en el anexo 3.

Ejemplo de respuesta

Petición del ejemplo (sin la IP del endpoint):
"/SSIIMovilWS/ws/cons/datosVehiculo.json?codVehiculo=3482&codEmpresa=1&petItinerario=true"

NOTA: Para el resultado de esta petición se muestran únicamente los cuatro primeros "puntos" que devuelve, ya que si incluyéramos los quinientos veintidós puntos de la respuesta real ocuparía demasiado.


```
{
	"estado": "DATOS",
	"vehiculo": {
		"id": "21",
		"cod": "3482",
		"idEmpresa": "1",
		"descEmpresa": "UTE Guadalajara",
		"pos": {
			"lon": -3.1480494362118066,
			"lat": 40.64367169482822
		}
	},
	"idParadaActual": "93",
	"idItinerarioActual": "53",
	"itinerarios": [{
		"id": "53",
		"cod": "60001_100",
		"destino": null,
		"nombre": null,
		"linea": {
			"id": "1",
			"cod": "C1",
			"empresa": null,
			"nombre": null,
			"color": -16776961
		},
		"actual": true,
		"puntos": [{
			"pos": {
				"lon": -3.181780190608464,
				"lat": 40.64381270955806
			},
			"tipo": 3,
			"parada": {
				"id": "6",
				"cod": "6",
				"desc": "RECINTO DE RENFE",
				"ordenParada": 1
			},
			"ordenPunto": 1
		},
		{
			"pos": {
				"lon": -3.1810937926190945,
				"lat": 40.64365162976264
			},
			"tipo": 0,
			"parada": null,
			"ordenPunto": 2
		},
```

```
                {
                        "pos": {
                                "lon": -3.180715357282054,
                                "lat": 40.64366123138452
                        },
                        "tipo": 0,
                        "parada": null,
                        "ordenPunto": 3
                },
                {
                        "pos": {
                                "lon": -3.180242306893348,
                                "lat": 40.643670979480035
                        },
                        "tipo": 0,
                        "parada": null,
                        "ordenPunto": 4
                }
        }]
}
```


Función "Líneas que transcurren por una parada"

Esta función detalla el código, id y la descripción de todas las líneas que transcurren por la parada indicada a través del idParada. La diferencia entre los dos primeros datos es que el id es un entero identificativo mientras que el código está compuesto de una letra y un número que identifica la línea de forma más amigable (por ejemplo, el código del "Circular 1" es el "C1").

Endpoint: /SSIIMovilWS/ws/cons/lineasParada.json

Método HTTP de la petición: GET

Parámetros de la petición:

- idParada: Numérico, el código asignado a la parada de la que se deseen conocer los itinerarios.
- tipoDia: String, debe coincidir con uno de los valores de la tabla de días (IMG). //TODO REFERENCIA A LA TABLA
- hInicio: Numérico, la hora del día desde la que se quieren consultar los itinerarios en formato HHmm
- hFin: Numérico, la hora del día hasta la que se quieren consultar los itinerarios en formato HHmm.

Estructura de la respuesta

Definida en el anexo 4.

Ejemplo de respuesta

Petición del ejemplo (sin la IP del endpoint): "/SSIIMovilWS/ws/cons/lineasParada.json?idParada=31&tipoDia=H&hInicio=600&hFin=1159"

```
{
	"estado": "DATOS",
	"lista": [{
		"id": "4",
		"cod": "L4",
		"empresa": null,
		"nombre": "L4 Hospital-Los Manantiales",
		"color": -16776961
	},
	{
		"id": "6",
		"cod": "L5",
		"empresa": null,
		"nombre": "L5 Los Manantiales-Los Valles-Los Manantiales",
		"color": -16776961
	},
	{
		"id": "8",
		"cod": "L7",
		"empresa": null,
		"nombre": "L7 Est.Autobuses-Iriepal-Taracena",
		"color": -16776961
	},
	{
		"id": "9",
		"cod": "L8",
		"empresa": null,
		"nombre": "L8 Est.Autobuses-El Clavin",
		"color": -16776961
	},
	{
		"id": "15",
		"cod": "D1",
		"empresa": null,
		"nombre": "Demanda 1 Est.Autobuses-Valdenoches",
		"color": -16777216
	}]
}
```


Función "Tiempo de llegada a una parada"

Esta función informa sobre los minutos estimados para que llegue el próximo bus de los distintos itinerarios a la parada indicada. Se informa de la estimación de todos los itinerarios que transcurren por la parada seleccionada, sin posibilidad de seleccionar únicamente el itinerario deseado.

Endpoint: /SSIIMovilWS/ws/cons/tiemposParada.json

Método HTTP de la petición: GET

Parámetros de la petición:

- codParada: Entero, el código asignado a la parada de la que se deseen conocer las estimaciónes.

Estructura de la respuesta

Definida en el anexo 5.

Ejemplo de respuesta

Petición del ejemplo (sin la IP del endpoint): "/SSIIMovilWS/ws/cons/tiemposParada.json?codParada=6"


```
{
	"estado": "DATOS",
	"parada": {
		"id": "6",
		"cod": "6",
		"desc": "RECINTO DE RENFE",
		"ordenParada": 0
	},
	"tiempos": [{
		"itinerario": {
			"id": "53",
			"cod": "60001_100",
			"destino": null,
			"nombre": "Circular 1 Estacion de RENFE",
			"linea": {
				"id": "1",
				"cod": "C1",
				"empresa": null,
				"nombre": "C1 RENFE-Hospital-RENFE",
				"color": -16776961
			},
			"actual": false,
			"puntos": null
		},
		"creacion": "11/03/2018 17:42:12",
		"minutos": 6,
		"cabecera": false,
		"tipo": 0
	}]
}
```


Función "Expediciones de itinerario"

Esta función devuelve las expediciones programadas para un día de tipo "tipoDia" entre las horas "hInicio" y "hFin" sobre el itinerario con "idItinerario", siendo "idParada" el id de una parada del itinerario seleccionado. Si el id de parada enviado no pertenece al itinerario con id enviado la API nos devolverá, por algún motivo incomprensible para el autor, el mensaje "NO EXPEDICIONES ITI".

Endpoint: /SSIIMovilWS/ws/cons/expedicionesItinerario.json

Método HTTP de la petición: GET

Parámetros de la petición:

- idParada: Entero, el id asignado a una parada perteneciente al itinerario deseado.

- idItinerario: Entero, el id asignado al itinerario del que se desean conocer las expediciones.
- tipoDia: String, debe coincidir con uno de los valores de la tabla de días (IMG). //TODO REFERENCIA A LA TABLA
- hInicio: Numérico, la hora del día desde la que se quieren consultar las expediciones en formato HHmm
- hFin: Numérico, la hora del día hasta la que se quieren consultar las expediciones en formato HHmm.

Estructura de la respuesta

Definida en el anexo 6.

Ejemplo de respuesta:

Petición del ejemplo (sin la IP del endpoint):

"/SSIIMovilWS/ws/cons/expedicionesItinerario.json?idParada=143&tipoDia=F&hInicio=0000&hFin=2400&idItinerario=16"


```
{
        "estado": "DATOS",
        "itinerario": {
                "id": "16",
                "cod": "60008_2",
                "destino": "Urbanizacion El Clavin-Est.Autobuses",
                "nombre": null,
                "linea": {
                        "id": "9",
                        "cod": "L8",
                        "empresa": null,
                        "nombre": null,
                        "color": 0
                },
                "actual": false,
                "puntos": null
        },
        "horas": ["09:30",
                        "13:30",
                        "16:30",
                        "20:30"]
}
```


Función "Itinerarios de línea"

Esta función devuelve la información sobre los itinerarios de una línea. Como cada línea tendrá un mínimo de 1 itinerario la propiedad "lista" nunca vendrá vacía.

Endpoint: /SSIIMovilWS/ws/cons/itinerariosLinea.json

Método HTTP de la petición: GET

Parámetros de la petición:

- idLinea: Entero, el id asignado a la línea de la que se desean conocer los itinerarios.

Estructura de la respuesta

Definida en el anexo 7.

Ejemplo de respuesta:

Petición del ejemplo (sin la IP del endpoint):

"/SSIIMovilWS/ws/cons/itinerariosLinea.json?idLinea=33"

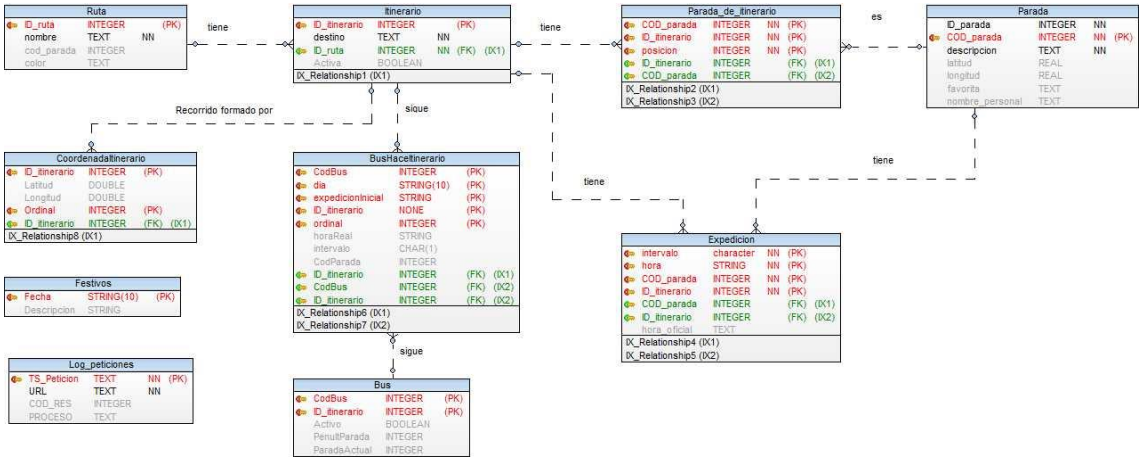

```
{
   "estado": "DATOS",
   "linea": {
      "id": "33",
      "cod": "D1Esp",
      "empresa": null,
      "nombre": null,
      "color": -16777216
   },
   "lista": [
      {
         "id": "66",
         "cod": "60024_1",
         "destino": "Est.Autobuses-Valdenoches",
         "nombre": "Est.Autobuses-Valdenoches",
         "linea": {
            "id": "33",
            "cod": "D1Esp",
            "empresa": null,
            "nombre": null,
            "color": -16777216
         },
         "actual": false,
         "puntos": null
      },
      {
         "id": "74",
         "cod": "60024_4",
         "destino": "Valdenoches-Est.Autobuses",
         "nombre": "Valdenoches-Est.Autobuses",
         "linea": {
            "id": "33",
            "cod": "D1Esp",
            "empresa": null,
            "nombre": null,
            "color": -16777216
         },
         "actual": false,
         "puntos": null
      }
   ]
}
```

7. Días festivos

Se han consultado los días festivos de varios años en una página web de consulta y se ha insertado manualmente en la tabla "Festivos" de la base de datos las fechas obtenidas junto a una pequeña descripción de la festividad para que la aplicación conozca sin depender de la conexión a Internet cuando el día es festivo. Esta información es necesaria para la consulta del número de expediciones de cada itinerario ya que se reduce su número durante los días festivos.

La carga inicial se ha efectuado hasta el año 2020, por lo que una vez llegue esa fecha se deberá cargar con los siguientes días.

8. Modelo de datos



La base de datos cuenta con las siguientes entidades:

Entidad Ruta

Representa las distintas líneas del servicio de autobuses urbanos de Guadalajara (España). Almacena información interna para poder obtener los datos de la API para la base de datos: Id_ruta y cod_parada. También contiene otros datos relativos al color en que se debe pintar la ruta en el mapa (color) y dos caracteres descriptivos de la misma (nombre).

Entidad Itinerario

Representa los distintos "sentidos" de cada ruta; por ejemplo, la ruta L3 transcurre desde A en sentido hacia B y desde B en sentido hacia A. El atributo destino es un pequeño texto obtenido de la API que describe el Itinerario en formato "origen - destino". El atributo "activa" indica si es un itinerario existente en la actualidad, ya que hay itinerarios que obtenemos de la API que ya no existen o de los que no podemos obtener información, como puede ser el itinerario "Fuerzas Armadas 27 mayo 2017".

Entidad Parada_de_itinerario

La entidad que relaciona itinerarios y paradas; también almacena la posición de la parada dentro del itinerario.

Entidad Parada

Representa una parada de autobuses. Contiene la longitud y latitud para mostrar los puntos en el mapa de la aplicación. Además, incluye la descripción de la parada (normalmente la calle en la que está situado) y, si es favorita, incluye los códigos de itinerario de la parada seleccionados como favoritos concatenados con un punto y coma como separador. El "nombre personal" es un texto descriptivo que el usuario le ha asignado a la parada.

Entidad Expedición

Registra una salida de un bus programada por una parada (cod_parada), siguiendo un itinerario (ID_itinerario) a una hora almacenada como String (formato HH:mm), y en un tipo de día (lunes - domingo, festivo o el día actual). El atributo "hora_oficial" tiene un doble propósito:

- Diferenciar entre las expediciones oficiales y las expediciones que hemos calculado con la aplicación de monitorización.
- En el caso de las "expediciones monitorizadas" nos indica a qué expedición oficial corresponde, es decir, cual fue la expedición original con la que inicia el itinerario.

Para mantener el comportamiento de la API guardamos la hora de la expedición sin tener en cuenta que a las 00:00AM empieza el siguiente día; Es decir, que una expedición del domingo de madrugada a las 00:30AM la grabaremos como "sábado" aunque técnicamente corresponda al domingo.

Entidad CoordenadaItinerario

Describe una coordenada del mundo por la que transcurre un itinerario. El atributo "Ordinal" nos indica la posición del punto dentro de la secuencia de coordenadas del recorrido del itinerario.

Entidad Festivos

Guarda los días que son festivos en la provincia de Guadalajara junto con una pequeña descripción de la festividad.

Entidad Bus

Representa un autobús al que se le asigna un itinerario. Se ha incluido el atributo itinerario con el objetivo de optimizar el cacheado de los autobuses, ya que cada bus suele seguir los mismos itinerarios. Los atributos "ParadaActual" y "Activo" nos sirven de apoyo para mantener el seguimiento en la aplicación de monitorización y en la app móvil cuando el usuario esté consultando los buses de un itinerario. El atributo PenultParada nos indica cuando el bus va a terminar su recorrido, ya que al alcanzar la última parada dejará de estar activo.

Entidad BusHaceItinerario

Esta entidad únicamente existe en la de la aplicación de monitorización ya que no se utiliza en ningún proceso de la app móvil. Describe la llegada a una parada de un autobús que sigue un itinerario en particular en una expedición determinada. En ese registro guardaremos la hora real a la que, llegado a esa parada, y que luego nos servirá para calcular la estimación de llegada offline.

Entidad LogPeticiones

Esta entidad únicamente existe en la de la aplicación de monitorización ya que no se utiliza en ningún proceso de la app móvil. Es un registro de cada petición HTTP que ha hecho la aplicación de monitorización contra la API. Esto nos da información sobre el uso que estamos haciendo de sus servidores y nos permite regular el número de peticiones que hacemos para evitar abusar de ella.

9. Aplicación de escritorio

La aplicación de escritorio ofrece las siguientes funcionalidades, necesarias para complementar a la aplicación móvil:

- Carga de base de datos
- Barrido de buses activos
- Monitorización de los autobuses
- Cálculo de las estimaciones de llegada

Carga de la base de datos

La funcionalidad de "carga de la base de datos" inserta la información obtenida del sistema en las distintas tablas de la base de datos. Estos datos incluyen desde los distintos recorridos de las líneas, las paradas, los horarios hasta las distintas expediciones de las líneas.

Puesto que partimos de un estado en que la base de datos está completamente vacía no disponemos de información alguna sobre los identificadores de los distintos elementos y debido al mal diseño de la API no existe una función que nos los proporcione. Por ello deberemos hacer fuerza bruta contra el ID de algunos recursos necesarios para poder obtener el resto otros gracias a la información obtenida con este método.

1. **Rutas** e **itinerarios**

Para comenzar con la carga primero obtendremos la información de las rutas disponibles. Como se ha indicado anteriormente la API no tiene una función que nos devuelva todas las rutas del sistema por lo que tendremos que obtenerlos por fuerza bruta. Durante el desarrollo se supuso que los IDs se asignan de forma autoincremental, por lo que iniciamos el barrido llamando a la función "itinerariosLinea" usando como parámetro idLinea los números enteros desde el 1 hasta el 100. Para diferenciar entre las rutas reales y las no existentes basta con comprobar el valor del atributo "estado" de la respuesta, si es "DATOS" indica que es una ruta.

2. Paradas y orden de las paradas en los itinerarios

Usando los identificadores de los itinerarios obtenidos en el anterior paso podemos usar la función "paradasItinerario" para obtener los datos de las paradas y el orden de las paradas en los distintos itinerarios de los que forman parte. La función "paradasItinerario" nos proporciona toda la información salvo su posición geográfica, la forma de obtener este dato se explica en el último paso.

El problema de este paso es que la función que usamos requiere como parámetro el id de una parada perteneciente al itinerario consultado. Como no conocemos este dato tendremos que hacer fuerza bruta contra el id de parada hasta obtener los datos del itinerario. Inicialmente este código se buscaba manualmente por cada itinerario.

3. Expediciones iniciales

Las expediciones varían dependiendo de los distintos intervalos por lo que tendremos que usar la función "expedicionesDeItinerario" contra todos los intervalos posibles con cada itinerario. De nuevo para usar esta función necesitaremos el identificador de una parada de cada itinerario, en este caso ya podemos utilizar los obtenidos en el paso anterior.

4. Buses y puntos geográficos de paradas e itinerarios

Debido al mal diseño de la API no se pueden obtener las coordenadas exactas de cada parada usando las funciones de los pasos anteriores; la única forma conocida consiste en usar la función "Información de la posición de un bus activo" enviando el parámetro "petItinerario" con valor "true" contra un bus que en el momento de la petición esté realizando el itinerario deseado. Como la petición debe realizarse en determinados momentos de días específicos de la semana se delega esta tarea en el algoritmo que sigue a los buses.

Barrido de buses activos

Al inicio del proyecto surgió la necesidad de conocer los distintos códigos de los autobuses del sistema. Desgraciadamente la API no tiene ninguna función que nos informe de los buses activos, ni siquiera que informe de los buses existentes. La función "Información de la posición de un bus activo" es la única que nos devuelve datos relativos a los vehículos como son la parada hacia la que se dirige, los puntos geográficos de su recorrido… etc. Pero usar esta función nos lleva a un problema distinto, ya que necesita obligatoriamente del id del bus que se quiere consultar, información de la que no disponemos inicialmente. Este caso no fue tan sencillo como los anteriores ya que no se podía hacer fuerza bruta contra el id desde el número 1 hasta encontrar todos los distintos buses por dos motivos:

1. Tomando dos ejemplos conocidos de identificadores como el 3479 o el 3483 lo que nos indica que no es un id autoincremental que comienza en 1, por lo que se tuvo que buscar un patrón que tampoco nos aseguraba que fuéramos a obtener todos los vehículos de la flota.

2. La función únicamente nos dará la información del bus únicamente cuando este esté en ruta. Esto implica que haya que rastrearlos realizando la petición contra la API en el momento en que estén de servicio.

Búsqueda por fuerza bruta de buses activos

Para evitar estos problemas inicialmente se buscó un patrón con los códigos que se recogieron manualmente de la estación de buses. Los identificadores obtenidos parecían indicar que los valores estaban dentro del rango 3470-3500, por lo que para intentar se configuró para que barriera todos los números desde el 3460 a 3510 en caso de que hubiera elegido mal los límites de intervalo.

Para la carga de datos inicial se programó este algoritmo para que se ejecutara cada 30 minutos durante las horas en que más expediciones (y en consecuencia buses activos) había, normalmente al mediodía. Con este método se consiguieron recolectar 23 autobuses, cuyos datos guardamos para futuro uso en nuestra base de datos local. Al usar estos identificadores de autobús con el monitorizador (más detalles en la siguiente sección) se logró seguir a un gran porcentaje de las expediciones del sistema, pero había varias para las que no se encontraba el bus que la estaba acometiendo; esto solo podía

indicar que nos faltaban algunos identificadores que estaban fuera del rango que se intuyó inicialmente. La solución a este problema quedó estancada al decidir no usar fuerza bruta contra todos los identificadores de 4 dígitos ya que supondría demasiado uso de la API. Afortunadamente unos meses después de desistir en esta tarea se ideó un nuevo método alternativo mucho más preciso usando Selenium.

Búsqueda por fuerza bruta del bus activo asignado a una expedición

Para la monitorización necesitaremos encontrar el bus activo que esté realizando la expedición objetivo de un itinerario en particular. Para optimizar esta búsqueda nos aprovechamos de que cada vez que encontramos un vehículo guardamos sus datos y el itinerario que está siguiendo.

Durante la búsqueda es posible que encontremos otros vehículos haciendo el itinerario objetivo, pero que están haciendo una expedición distinta. El problema es que la función de la API no nos indica qué expedición realiza cada bus, por lo que cuando encontramos un bus activo del itinerario objetivo tendremos que discernir si es el que está realizando la expedición deseada. Para descubrirlo se comprobará si el bus está transitando por una de las dos primeras paradas del itinerario. Si el vehículo todavía está en el inicio significará que es el bus de la expedición buscada ya que acaba de iniciar la ruta y el resto de las expediciones de ese mismo itinerario deberían estar mucho más adelantadas por la diferencia de tiempo entre expediciones.

Respecto a por qué identificador de bus empezaremos a buscar, el primer paso es comprobar en la base de datos los códigos de bus que ya hayan hecho alguna vez ese itinerario ya que los vehículos suelen usarse en el mismo conjunto de itinerarios. Si no lo encontramos pasamos a comprobar el resto de los identificadores de autobús que tenemos en la base de datos que no coincidan en itinerario. Si aun así no lo encontramos significa que es un identificador que todavía no hemos guardado en la base de datos, por lo que intentaremos encontrarlo por fuerza bruta en el intervalo descrito anteriormente. En los sucesivos pasos se guardan los itinerarios ya intentados para así no comprobar dos veces el mismo.

El algoritmo de búsqueda simplificado es el siguiente:

1- Buses cacheados que hagan el itinerario buscado
2- Buses cacheados que NO hagan el itinerario buscado
3- Fuerza bruta al intervalo de búsqueda.

Búsqueda por Selenium

Un tiempo después del desarrollo del programa anterior la empresa creó una nueva página web en que se podían consultar los buses que estaban realizando las distintas, información que no se proporciona en la aplicación móvil oficial ni (que se sepa) en la API. Esta página podría solucionar el problema de los identificadores desconocidos sustituyendo la poco efectiva fuerza bruta sobre el intervalo estimado por una petición contra la web y analizar el HTML de la respuesta. Lamentablemente esto no fue posible aplicar esta aproximación al completo por los siguientes motivos:

1. En la página no se muestran todos los itinerarios disponibles, por lo que los itinerarios que no contemplan los tenemos que buscar por fuerza bruta.

2. La página en ocasiones se bloquea al cambiar de itinerario.

3. La web está programada en JSF, un framework con estado (stateful), y actualiza sus datos mediante AJAX. Esto implica que obtener la información no es tan sencillo como realizar peticiones HTTP modificando el parámetro de itinerario y analizar directamente la respuesta, sino que habría que tratar cookies y parámetros de estado que se modifican vía JavaScript para realizar las distintas peticiones, como por ejemplo el parámetro "stateView" que guarda encriptado el estado de los distintos componentes.

Debido a la complejidad que añade la restricción número 3 a la automatización por peticiones HTTP directas se optó por una solución rápida para así empezar a obtener datos tan pronto como fuera posible: Selenium. Gracias a esta herramienta podemos emular a un usuario con un navegador haciendo clic en los selectores del itinerario. Como es un navegador cada vez que se simula un clic en un itinerario distinto se ejecuta el JavaScript adecuado, siguiendo así el funcionamiento normal de la web. De este modo solo tendríamos que programar los clics sobre los distintos itinerarios, indicando que espere a que cargue la información del itinerario seleccionado para continuar con el proceso. Para mejorar la eficiencia del algoritmo se ha configurado Selenium para que no cargue la ventana de la interfaz ("headless") y así ahorrar tiempo de procesamiento.

Combinando el método basado en Selenium con la fuerza bruta para alcanzar los casos que no contempla la página se ha conseguido obtener la información de 29 vehículos, 6 más que el método inicial por fuerza bruta. El identificador más alto encontrado ha sido 5090, por lo que suponemos que esto se debe a que con el tiempo han ampliado la flota y que el identificador lo asigna la empresa a nivel general; por lo que los identificadores que se han "saltado" desde el máximo anterior (3496) probablemente se hayan usado en otras provincias/países.

Puntos geográficos de itinerarios y paradas

Esta función también sirve para la obtención de las coordenadas geográficas de los distintos itinerarios ya que esta información no está disponible de forma estática. La API del sistema de autobuses únicamente nos informa de las coordenadas del recorrido de un itinerario cuando hacemos una petición contra un bus activo de ese itinerario, por lo que en cuanto el sistema encuentra un bus comprueba si ya disponemos de esas coordenadas en la base de datos y si no las tiene las solicita y almacena.

Monitorización de los tiempos de los autobuses

Para realizar nuestras propias estimaciones de llegada a las distintas paradas necesitaremos saber a qué hora real están llegando los distintos buses a las respectivas paradas para así poder realizar el cálculo. Este "log" lo generará una aplicación que monitoriza la actividad de los autobuses constantemente para obtener toda la información de llegadas que nos sea posible.

Restricciones para evitar abusar de la API

Puesto que monitorizar todos los itinerarios diariamente implicaría demasiadas peticiones al servidor de la empresa se han divido los itinerarios más comunes en grupos equilibrados según el peso que suponen a los servidores, cada día se monitorizará únicamente uno de los grupos. El grupo a monitorizar rota automáticamente con cada ejecución.

Para la división en estos grupos se han tenido en cuenta las variables que afectan al número de peticiones: El tiempo del trayecto medio y el número de expediciones totales. Esta restricción de grupos no se ha aplicado a algunos itinerarios que, por sus características, no ocurren con frecuencia como por ejemplo los servicios "búho" de fin de semana o los servicios bajo demanda.

Cada petición que realiza el programa contra la API queda registrada en la tabla "Log_peticiones" para mantener el control del número de peticiones y comprobar que no hay ningún problema que puede llevar a realizar peticiones en bucle por error. Para evitar abusar de la API se ha establecido un límite máximo de 10.000 peticiones diarias, lo que equivale aproximadamente a siete peticiones por minuto.

Funcionamiento general

Al iniciar el programa lo primero que hace es consultar qué expediciones hay programadas para el día actual y programar un *hilo de búsqueda de buses* por cada hora única de expedición (sin repetidos). Además, se programa un *hilo de monitorización de expedición* por cada expedición que despertará cuando pasen N segundos desde la hora de la expedición, para que tenga tiempo para terminar el hilo de búsqueda de buses. Se puede configurar simplemente modificando una constante para que inicie al día siguiente en lugar de en el actual para el caso en que queramos dejarlo preparado para la madrugada del día siguiente.

Hilo de búsqueda de buses

El *hilo de búsqueda de buses* despierta a la hora de la expedición asignada con un pequeño retardo de 20 segundos para dar tiempo a que los vehículos comiencen los viajes ya que no arrancan a los 0 segundos de la hora de salida. Este hilo rastrea los buses activos que se van a monitorizar para indicarle directamente sus identificadores a los hilos que los monitorizan. Para ello primero comprueba de qué tipo de itinerarios debe cargar los buses activos para elegir que método usará: el de "Búsqueda por fuerza bruta", el "Búsqueda Selenium" o ambos.

En la versión inicial era el propio hilo de monitorización el que buscaba el identificador del bus, por lo que cuando había varias expediciones simultaneas se podían repetir algunas peticiones a la API que eran innecesarias. Con este modelo productor-consumidor se rastrean todos los buses buscados una única vez por un hilo para que luego los N hilos consumidores acceden directamente a los datos del bus que necesitan a través de la tabla. En la versión final es el propio hilo de búsqueda de buses el que lanza los hilos de monitorización al terminar de obtener los identificadores, en lugar de tener un pequeño tiempo de espera para que le diera tiempo a cargar todos los datos.

Hilo de monitorización de expedición

El *hilo de monitorización de expedición* despertará después de que el hilo de búsqueda de buses haya terminado su trabajo. Lo primero que hace este hilo es buscar qué bus es al que se le ha asignado realizar la expedición. Para ello consultará la información de ese bus en la base de datos, ya que el *hilo monitorización de buses* ya los habrá obtenido y guardado el identificador del bus asociado al itinerario y la expedición que sigue. Una vez encontrado comienza la vigilancia: El hilo consultará el estado del bus con el identificador encontrado cada cierto periodo de tiempo, preguntando por la parada a la que se dirige el vehículo. En el momento en que la parada a la que se dirige cambie, significará que el

bus acaba de superar la parada, por lo que guardaremos un registro en el log con la hora del sistema y otros datos necesarios para los cálculos, como el identificador del itinerario, la expedición a la que corresponde…etc.

Determinar el tiempo de refresco óptimo para el hilo es una tarea difícil, ya que si el valor seleccionado es muy pequeño el servidor puede sufrir problemas por recibir demasiadas peticiones seguidas, mientras que si es demasiado grande los datos que obtengamos para el log serán muy imprecisos. Actualmente se ha configurado con 25 segundos entre peticiones del mismo hilo.

Coste para la obtención de datos

Se ha realizado un cálculo aproximado del coste que ha supuesto la monitorización para el servidor; teniendo en cuenta que responder cada petición de estado del vehículo ocupa una media de 227 Bytes y que según el log se han realizado 217.887 peticiones desde el inicio de la monitorización hace un total de 47,1 Megabytes de información enviada a nuestro programa.

Cabe destacar la gran mejoría que se ha producido con la implementación del método "Selenium" para la búsqueda del bus a monitorizar, ya que ahorramos todas las peticiones inútiles contra los buses que no están de servicio. Esta disminución en el número de peticiones nos permite incluir más itinerarios en los grupos de monitorización, obteniendo de esta forma más registros por día. Se puede observar la mejora al comparar la ratio peticiones/registros obtenidos; en dos días diferentes (siendo el mismo día de la semana) con el método de "fuerza bruta" necesitamos 7.000 peticiones para obtener 691 registros (10 peticiones/registro) mientras que con el método "Selenium" grabamos 1668 registros invirtiendo 6.700 peticiones (4 peticiones/registro).

Cálculo de las estimaciones de llegada

El sistema de autobuses únicamente nos informa de la hora a la que inician los distintos viajes de cada itinerario, pero no nos ofrece una estimación de la hora aproximada a la que suele llegar el autobús al resto de paradas que forman el itinerario. Para calcular esta estimación nos valdremos de los datos recopilados por la aplicación monitorizadora que indican a qué hora real llegaron los distintos buses del sistema a las paradas.

Para el cálculo de una estimación fiable deberemos filtrar los registros del log en que no sean relevantes, como por ejemplo los casos en que el conductor se haya retrasado un cuarto de hora o que por alguna avería haya tardado un tiempo anormal en llegar al destino. Para detectar estas anomalías partiremos de la premisa de que el bus sólo podrá retrasarse, pero nunca iniciar el viaje antes del tiempo indicado en su sistema. Como la expedición nunca comenzará antes de la hora indicada únicamente tendremos que obtener la marca del log más temprana y a desde ella establecer un intervalo dentro del que consideraremos los tiempos como "normales". Este intervalo se regulará con el uso de la aplicación, inicialmente se ha considerado adecuada una horquilla de 10 minutos respecto a la marca más rápida. Para regular el intervalo se tendrán en cuenta las estadísticas que se muestran tras la ejecución del cálculo de las estimaciones, que reflejan con un porcentaje el número de marcas que se han rechazado por estar fuera de los tiempos considerados como normales. Una vez tenemos la lista de tiempos dentro del intervalo, calculamos la diferencia temporal media entre la marca más temprana y el resto de las marcas para después sumarla a la hora tomada de referencia.

A continuación, se muestra una tabla con datos sobre la información obtenida al calcular la hora media a la que suelen pasar los buses. La variable N representa la longitud del intervalo en que aceptamos las llegadas como correctas, tomando como inicio del intervalo la llegada más temprana

registrada a la parada. En la tabla se muestra el total de registros de llegada disponibles por intervalo y la variación del número de registros rechazados con varios valores de N. Para el cálculo de las horas que se usarán en la aplicación se ha optado por un intervalo de 5 por considerar que un número más bajo rechazaría demasiados registros y el valor obtenido no sería tan fiable.

		N = 10		N = 7		N = 5		N = 3		N = 2	
Itinerario	**Total**	**# Rech.**	**% Rech.**	**# Rech.**	**% Rech.**	**# Rech.**	**% Rech.**	**# Rech.**	**% Rech.**	**# Rech.**	**% Rech.**
3	**3.014**	15	**0,50%**	29	**0,96%**	65	**2,16%**	253	**8,39%**	535	**17,75%**
4	**4.454**	11	**0,25%**	26	**0,58%**	76	**1,71%**	422	**9,47%**	846	**18,99%**
15	**685**	0	**0%**	0	**0%**	5	**0,73%**	89	**12,99%**	188	**27,45%**
16	**838**	0	**0%**	0	**0%**	13	**1,55%**	88	**10,5%**	157	**18,74%**
17	**31**	0	**0%**	0	**0%**	0	**0%**	0	**0%**	5	**16,13%**
18	**8**	0	**0%**	0	**0%**	0	**0%**	0	**0%**	0	**0%**
19	**39**	0	**0%**	0	**0%**	1	**2,56%**	5	**12,82%**	8	**20,51%**
20	**49**	0	**0%**	1	**2,04%**	1	**2,04%**	9	**18,37%**	14	**28,57%**
24	**1.120**	0	**0%**	17	**1,52%**	43	**3,84%**	194	**17,32%**	402	**35,89%**
25	**488**	0	**0%**	0	**0%**	2	**0,41%**	33	**6,76%**	91	**18,65%**
26	**84**	0	**0%**	0	**0%**	0	**0%**	3	**3,57%**	12	**14,29%**
27	**24**	0	**0%**	0	**0%**	0	**0%**	2	**8,33%**	5	**20,83%**
28	**207**	0	**0%**	2	**0,97%**	20	**9,66%**	43	**20,77%**	81	**39,13%**
30	**42**	0	**0%**	0	**0%**	0	**0%**	0	**0%**	2	**4,76%**
31	**36**	0	**0%**	1	**2,78%**	1	**2,78%**	4	**11,11%**	7	**19,44%**
53	**9.438**	140	**1,48%**	403	**4,27%**	764	**8,09%**	1635	**17,32%**	2500	**26,49%**
54	**8.830**	227	**2,57%**	552	**6,25%**	1108	**12,55%**	2084	**23,6%**	2904	**32,89%**
57	**1.446**	65	**4,5%**	73	**5,05%**	105	**7,26%**	171	**11,83%**	223	**15,42%**
58	**1.989**	0	**0%**	9	**0,45%**	59	**2,97%**	400	**20,11%**	742	**37,31%**
59	**989**	0	**0%**	0	**0%**	21	**2,12%**	135	**13,65%**	260	**26,29%**
60	**2.668**	0	**0%**	12	**0,45%**	54	**2,02%**	328	**12,29%**	681	**25,52%**
61	**956**	0	**0%**	0	**0%**	4	**0,42%**	46	**4,81%**	98	**10,25%**
62	**116**	0	**0%**	0	**0%**	0	**0%**	3	**2,59%**	4	**3,45%**
63	**72**	0	**0%**	0	**0%**	0	**0%**	0	**0%**	2	**2,78%**
70	**5.028**	47	**0,93%**	102	**2,03%**	255	**5,07%**	755	**15,02%**	1272	**25,3%**
71	**4.404**	4	**0,09%**	28	**0,64%**	91	**2,07%**	389	**8,83%**	725	**16,46%**
72	**384**	2	**0,52%**	12	**3,13%**	29	**7,55%**	80	**20,83%**	130	**33,85%**

10. Aplicación móvil: GuadaBus

10.1 Instalación

Para instalar la aplicación deberá disponer de un dispositivo Android con versión superior a la versión Debido a que aún no se ha publicado la app en la tienda de aplicaciones de Android para instalarla se deben activar en el menú de "Ajustes" del móvil la opción "Orígenes desconocidos" para permitir la instalación. Una vez activada esta opción bastara con copiar el instalador (GuadaBUS.apk) en la memoria de un dispositivo Android (Interna o tarjeta SD), y seleccionar el archivo desde el dispositivo. Se preguntará al usuario si quiere instalar la aplicación, seleccionamos la opción "Si" y esperamos a que confirme que la instalación se ha completado.

Cuando finalmente se publique la aplicación en la tienda oficial la instalación será tan sencilla como buscar en la Store de Google por el nombre "GuadaBus" y pulsar instalar.

10.2 Patrones software

Durante el desarrollo de ambas versiones de la aplicación móvil se han aplicado distintos patrones software que facilitan la mantenibilidad y eviten sorpresas cuando se avance en la codificación por un mal diseño del sistema:

Singleton

Patrón de creación que garantiza que una clase solo tenga una instancia y proporciona un punto de acceso global a ella. Todos los objetos que utilizan una instancia de esa clase usan la misma instancia. El resultado de aplicar este patrón es que solo puede existir al mismo tiempo una instancia de una clase, que será accedida a través de un método.

Este patrón se ha aplicado en la clase App ya que es la encargada de gestionar la cola global de peticiones HTTP, por lo que sólo debe existir una instancia simultáneamente. Para usar sus métodos bastará con llamar a "App.getInstance()", que nos devolverá la instancia ya existente de la clase.

```
public class App extends Application
{
    // Instancia Singleton de la aplicación
    private static App sInstancia;

    // Cola de peticiones Volley
    private RequestQueue mColaPeticiones;

    /**
     * @return la instancia singleton de la aplicación
     */
    public static App getInstance()
    {
        return sInstancia;
    }

    /**
     * Devuelve la cola de peticiones Volley para poder crear peticiones HTTP
     *
     * @return {@link com.android.volley.RequestQueue}
     */
    public RequestQueue getVolleyRequestQueue()
    {
        if (mColaPeticiones == null)
        {
            mColaPeticiones = Volley.newRequestQueue( context: this, new OkHttpStack(new OkHttpClient()));
        }
        return mColaPeticiones;
    }
```

Builder

Este patrón de creación simplifica la creación de objetos complejos definiendo una clase cuyo propósito es construir instancias de otra clase, separando la construcción de un objeto complejo de su representación, de forma que el mismo proceso de construcción pueda crear diferentes representaciones.

La creación de los AlertDialogs ("popups") mediante un Builder se debe a que el proceso de construcción debe permitir diferentes representaciones del objeto que está siendo construido, siendo adaptado para múltiples situaciones únicamente especificando distinto título o Views para el Dialog (indicar fallo de conexión, solicitar un dato al usuario… etc).

```
new android.app.AlertDialog.Builder(contexto)
        .setTitle("Introduzca un nombre para la parada")
        .setView(vistaDialog)
        .setPositiveButton( text: "Aceptar", (dialog, which) → {
                EditText input = (EditText) vistaDialog.findViewById(R.id.nombre_deseado_input);
                String textoIntroducido = String.valueOf(input.getText());
                if(textoIntroducido.trim().isEmpty()){
                    textoIntroducido = "<Sin nombre>";
                }
                actualizaParadaBBDDeInterfaz(textoIntroducido, favorita,tVNombrePersonalizado);
        })
        .setNegativeButton( text: "Cancelar", (dialog, which) → {
                //SI cancelar introducimos "<sin nombre>"
                actualizaParadaBBDDeInterfaz("<Sin nombre>", favorita,tVNombrePersonalizado);
                dialog.cancel();
        }).show();
```

Adapter

Este patrón estructural sirve de intermediario entre dos clases, convirtiendo las interfaces de una clase para que pueda ser utilizada por otra. Permite que cooperen clases que tienen interfaces incompatibles.

La aplicación utiliza múltiples adapters ya que por ejemplo la clase listview de Android por defecto contiene un TextView únicamente, por lo que si se quieren añadir una interfaz más compleja añadiendo vistas adicionales se debe crear un Adaptador personalizado, estableciendo en el las propiedades de las vistas adicionales con los argumentos que recibe. El siguiente ejemplo muestra el adapter de paradas usado para la ListView de la actividad "Líneas" y la actividad "Búsqueda de paradas".

```
public class ParadaAdapter extends ArrayAdapter<Parada> {

    private List<Parada> listaParadas;

    public ParadaAdapter(Context context, List<Parada> listaParadas) {
        super(context,  resource: 0, listaParadas);
        this.listaParadas = listaParadas;
    }

    @Override
    public View getDropDownView(int position, View cnvtView, @NonNull ViewGroup prnt)
    {
        return getView(position,cnvtView,prnt);
    }

    @NonNull
    @Override
    public View getView(final int position, View contentView, @NonNull ViewGroup parent)
    {
        // Comprueba si se está reutilizando una vista ya existente, si no es así la crea ("infla")
        if (contentView == null) {
            contentView = LayoutInflater.from(getContext()).inflate(R.layout.drawer_list_stop, parent,  attachToRoot: false);
        }

        final TextView tVCodigo = (TextView) contentView.findViewById(R.id.list_stop_cod);
        final TextView tVDescripcion = (TextView) contentView.findViewById(R.id.list_stop_description);

        tVCodigo.setText(String.valueOf(listaParadas.get(position).getCodStop()));
        tVDescripcion.setText(listaParadas.get(position).getDescription());
```

Proxy

El patrón estructural proxy proporciona un objeto (representante) con el que controlamos el acceso a otro objeto. Esto proporciona protección al objeto representado además de concentrar todas las operaciones sobre el mismo en una clase, haciendo más sencillo el control.

En la aplicación se ha aplicado para controlar el acceso a la base de datos a través de la clase "RouteDataSource". Esta clase contiene métodos para realizar consultas contra la base de datos SQLite de la aplicación, pudiendo así recuperar datos sobre los distintos elementos para mostrarlos en la interfaz.

```
private SQLiteDatabase database;

//Crea una conexión hacia la base de datos
public RouteDataSource(Context context) {
    DataHelper openHelper = new DataHelper(context);
    database = openHelper.getWritableDatabase();
}
```

A través de una instancia de RouteDataSource podemos realizar las peticiones a la base de datos, como se muestra a continuación:

```
rds = new RouteDataSource( context: this);
rutas = rds.getRutasActivas();
itinerariosDeRuta = rds.getItinerarios(rutas.get(0).getIdRuta(), ruta: true);
```

Observer

Este patrón de comportamiento permite definir dependencias uno-a-muchos de forma que los cambios en un objeto se comuniquen a los objetos que dependen de él, proporcionando a los componentes una forma flexible de enviar mensajes de difusión a los receptores interesados.

El paquete "network" contiene las clases necesarias para recibir la información de Internet. Estas peticiones se administran mediante una "cola de peticiones" del framework Volley; puesto que los tiempos de respuesta dependen de la conexión a internet, este sistema permite añadir una petición a la cola, y asignarle un listener y errorlistener de modo que cuando la operación de red haya finalizado (recibimos el JSON de la web, o recibimos un error de conexión) se notificará al listener y podremos actuar con los datos en consecuencia. Además, se pueden cancelar las peticiones mediante identificadores, permitiendo así cancelar las operaciones de red si el usuario ha cambiado de actividad en la aplicación. Una de las ventajas de este sistema es que la aplicación no es bloqueada mientras espera la respuesta del servidor por lo que puede continuar respondiendo a las acciones del usuario en la interfaz.

```
public static GsonGetRequest<Times> getTimes
        (
                @NonNull final int idParada,
                @NonNull final Response.Listener<Times> listener,
                @NonNull final Response.ErrorListener errorListener
        )
{
    StringBuilder urlBuilder = new StringBuilder(); //faster append
    final String url = urlBuilder.append(BuildConfig.apiDomainName).append("tiemposParada.json?idParada=").append(String.valueOf(idParada)).toString();
    System.out.println(url);
    GsonGetRequest<Times> gsonGetRequest = new GsonGetRequest<>
            (
                    url,
                    new TypeToken<Times>() {}.getType(),
                    gson_time,
                    listener,
                    errorListener
            );
    gsonGetRequest.setRetryPolicy(new DefaultRetryPolicy(TIMEOUT, DefaultRetryPolicy.DEFAULT_MAX_RETRIES, DefaultRetryPolicy.DEFAULT_BACKOFF_MULT));
    return gsonGetRequest;
}
```

Además, la relación array-adaptador usada en los elementos Spinner y ListView también aplica este patrón, donde el adaptador recibe las notificaciones de que el array o la lista ha sido modificando indicando que debe actualizar la interfaz con esta nueva información.

```
private void actualizaCeldasAMsg(String msg){
    //Actualizamos el mensaje de la celda de itinerario
    for(ItinerarioLista itinerarioLista:itinerariosDeParada){
        itinerarioLista.setCargando(false);
        itinerarioLista.setMsgTiempo(msg);
    }
    adaptadorItinerarios.notifyDataSetChanged();
}
```

10.3 Diferencias entre versiones

El desarrollo de dos versiones de la aplicación para el proyecto se debe principalmente a dos motivos. El primero es un cambio en los requisitos de la aplicación, añadiendo varios como las estimaciones de llegada y el seguimiento de buses y eliminando el de "alarma" por considerarse poco útil (más adelante se detallan los motivos). La segunda razón es que tras el desarrollo de la primera versión se expusieron nuevos datos en la página web de la empresa de buses, facilitando el seguimiento de buses y haciendo su seguimiento posible. A continuación, se detallan las principales diferencias entre ambas versiones:

1. **Evolución de los requisitos iniciales**

   Se eliminó el requisito de "Alarma de bus", mientras que se añadieron otros como el de "Obtener datos adicionales para el modo sin conexión" y el de "Mostrar la posición de los distintos buses de la línea en tiempo real".

2. **Eliminar la opción de alarma**

   Uno de los requisitos del comienzo consistía en permitir que el usuario estableciera una alarma que se activaría cuando faltaran N segundos para la llegada del bus según la estimación online del sistema. Esta opción se descartó por la poca precisión de las estimaciones del sistema, que pueden fluctuar enormemente repentinamente, haciendo estas alarmas poco fiables ya que podrían activarse demasiado tarde para que al usuario le fueran útiles.

3. Desplazar los selectores de la "barra de app" al contenido de la pantalla

La aproximación inicial fue situar el selector de líneas en las pantallas que la necesitan (líneas y mapa) en la barra de App, de modo que ahorrábamos ese espacio en el propio contenido de la pantalla y así podíamos mostrar más datos (Figura 0.1). Lo que en principio parecía una buena idea terminó desembocando en una saturación de información excesiva en la pantalla, además de llevar a confusión sobre su funcionamiento. Para solucionarlo en la versión final (Figura 0.2) se ha desplazado este selector a la pantalla y se ha añadido una etiqueta indicando su significado. El impacto sobre el espacio libre de este desplazamiento ha sido mínimo ya que, además, se ha reubicado la funcionalidad de favoritos al menú de navegación y al menú inicial.

4. Menú principal

En la primera versión cuando se abre la aplicación se accedía directamente a la pantalla de las líneas de autobuses como se puede observar en la Figura 0.3. Para la segunda se creó un menú inicial para que, antes de bombardear al usuario con información, pueda elegir qué y cómo quiere consultar los datos (Figura 0.4).

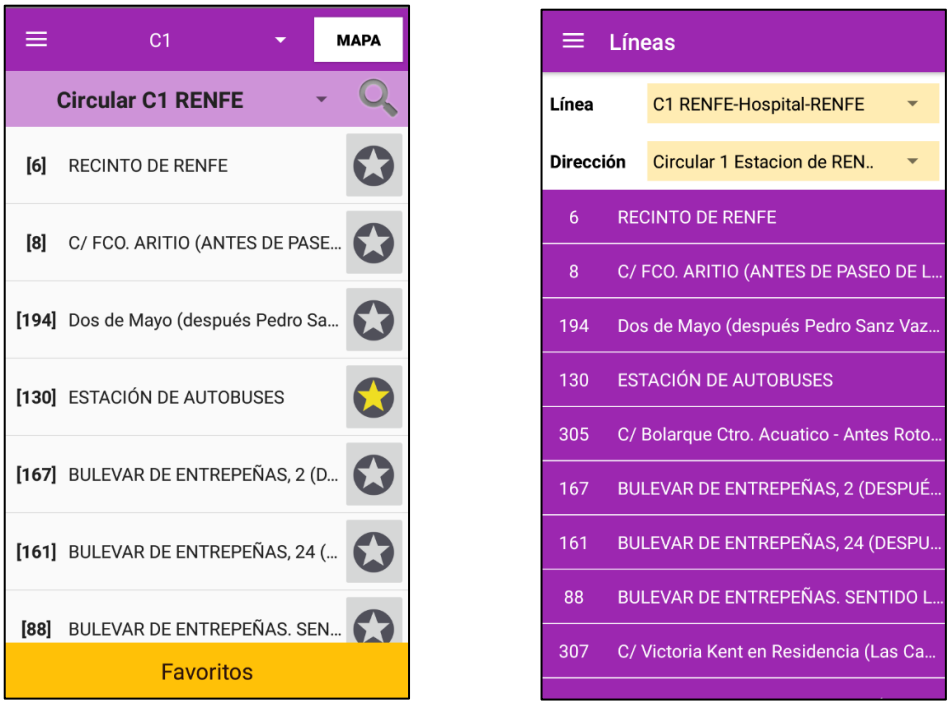


Fig 0.1, 0.2: Pantalla de líneas en versión 1 (0.1) y en la versión 2 (0.2)

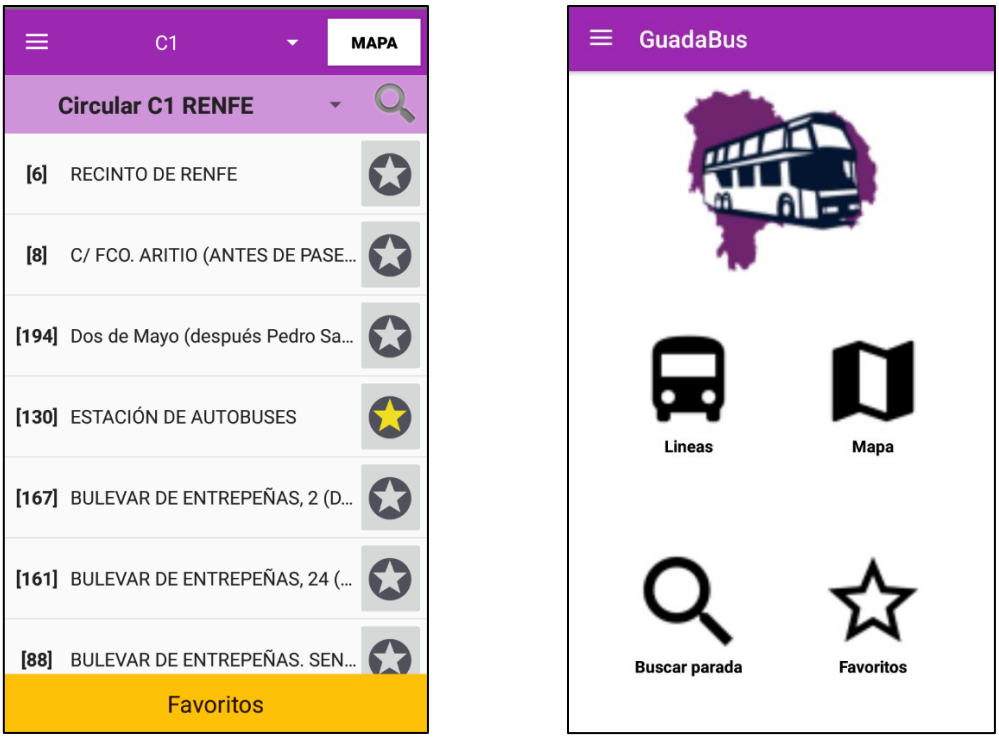


Fig 0.3, 0.4: Pantalla inicial en versión 1 (0.3) y en la versión 2 (0.4)

5. Carga de base de datos

En la versión inicial la base de datos la generaba la propia aplicación la primera vez que se iniciaba con conexión a Internet. Esto permitía que si se aplica algún cambio en el sistema al usuario le bastaría con pulsar la opción "Actualizar BBDD" para generarla de nuevo y obtener estos datos actualizados de la API.

En la versión final se pasó a incluir la base de datos como un "asset" externo ya precargado, principalmente debido a que se encontraron datos que proporciona la API que se han corregido manualmente (autobuses que inician su itinerario a las 99:99 por ejemplo). Para evitar mostrar estos datos erróneos al usuario se han purgado manualmente y han sido cargados en nuestra base de datos de modo que nosotros controlamos los datos que mostramos.

6. Base de datos

Para cumplir los nuevos requisitos se crearon nuevas tablas que no existían en la base de datos inicial, y se modificaron algunas columnas como la que almacena el dato de una parada cuando es favorita. A continuación, se muestran los esquemas de ambas aplicaciones a modo de comparativa. Para evitar incluir información innecesaria para la aplicación móvil sobre el sistema de rastreo al incluir esta base de datos se ejecuta el comando SQL "DROP TABLE" sobre las tablas Log_peticiones y BusHaceItinerario.

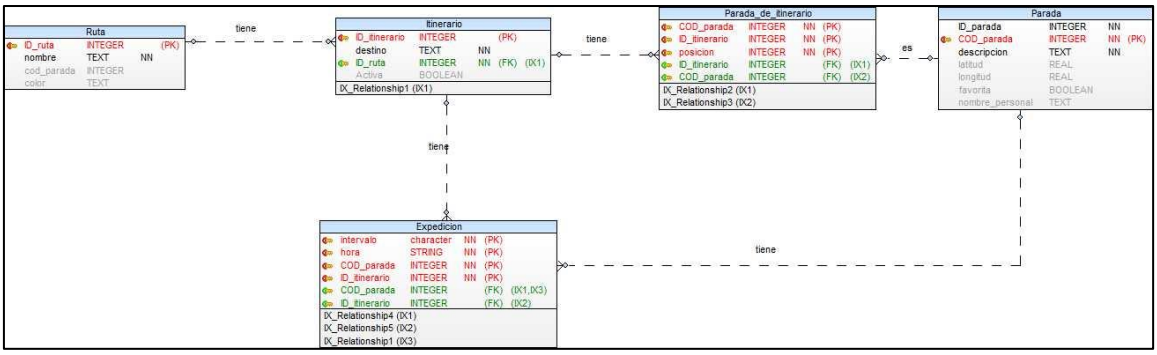


Fig 0.6: Esquema de la base de datos en la versión 1

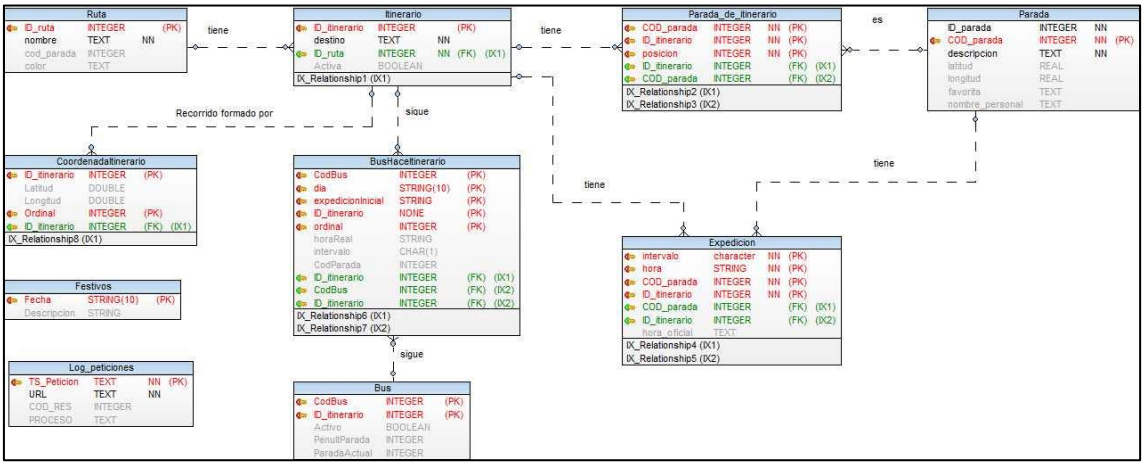


Fig 0.7: Esquema de la base de datos en la versión 2

7. Recorrido del itinerario sobre el mapa

El trayecto del itinerario que se dibuja sobre el en el mapa se realiza a través de la API de Google Maps, solicitando el camino en coche más rápido para ir de una parada a la siguiente. Se repetía esta operación con todas las paradas del itinerario seleccionado siguiendo el orden de parada.

Se optó por mostrar el recorrido obtenido de la API en lugar de calcularlo con Google Maps por varias razones. Para empezar la ruta no siempre era exactamente la que seguían los autobuses, probablemente debido a que debido a momentos puntuales de tráfico nos puede sugerir una óptima para el momento o porque deban. Además, cada vez que se quería consultar el mapa se debían solicitar de nuevo las coordenadas entre los dos mismos puntos que hace unos minutos, algo innecesario y que en ocasiones supone un malgasto de la conexión de datos móviles. Al guardarlo en nuestra propia tabla además evitamos el inconveniente del número limitado de peticiones para la API Key de Google Maps de nuestra aplicación, ya que podría llegar a suponer un problema si tuviéramos muchos usuarios conectados a la vez.

8. Estructura de la app

Inicialmente la estructura de la app se basaba en una actividad estática sobre la que se desplegaban Dialogs (una especie de Popup) con las distintas funcionalidades. De esta estructura se pasó a dividir las distintas funcionalidades en actividades individuales.

Esta decisión se debe a que muchas veces las pantallas estaban demasiado cargadas de información de modo que ocupaba prácticamente la totalidad de la pantalla. Se adaptó esta nueva estructura para aprovechar el espacio del dispositivo en su totalidad, además de debido a que este no se considera el uso correcto de los Dialogs. Como ejemplo de este cambio a continuación se puede ver cómo ha evolucionado la forma de mostrar los detalles de una parada, sus estimaciones y las expediciones de una versión a otra.

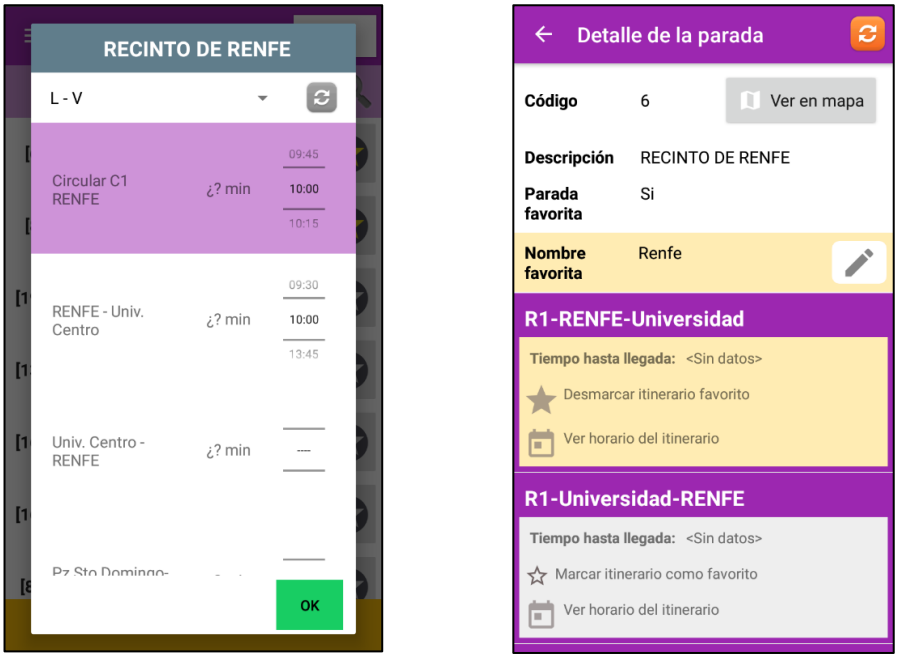


Fig 0.8, 0.9: Interfaz de "Detalle de parada" en versión 1 (0.8) y en versión 2 (0.9)

9. Favoritos

En la primera versión se marcaba únicamente la parada favorita y la lista se mostraba en un desplegable que se encontraba en todo momento en el borde inferior, que nos permitía clicar sobre las distintas opciones favoritas para que nos mostrase un Popup con los tiempos. En las siguientes capturas puede observarse el funcionamiento del desplegable.

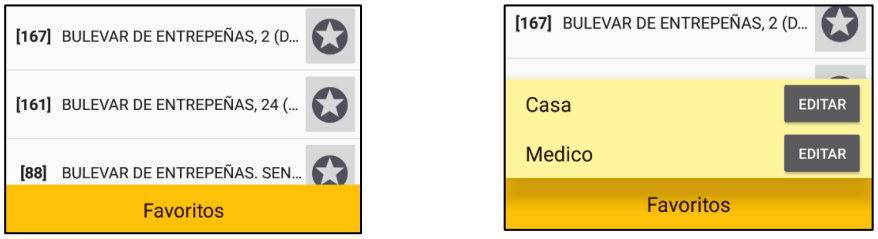


Fig 0.10,0.11: Desplegable de favoritos en la versión 1 contraido (0.10) y desplegado (0.11)

Este funcionamiento presentaba dos desventajas: al usuario se le muestran las estimaciones de todos los itinerarios de la parada cuando probablemente sólo le interese uno, y, además, antes tiene que clicar la favorita que desea ver para obtener la información. El problema de no poder remarcar únicamente un itinerario se acentúa más si cabe en las paradas más importantes como puede ser el Hospital o la Estación de buses, ya que por ellas transcurre un gran número de los itinerarios del sistema.

Para subsanar estos dos problemas se replanteó la funcionalidad de favoritos, pasando de un desplegable a una actividad completa (Figura 0.12). Esto permite al usuario ver en el momento en que selecciona favoritas todas sus paradas favoritas simultáneamente. Además, al seleccionar un par parada-itinerario se le muestra únicamente el itinerario que le interesa, eliminando así el ruido visual que generaba el resto. Pero esta decisión nos lleva a otra pregunta, ¿permitir introducir un nombre personalizado por cada itinerario de la parada? ¿o en su lugar usar un nombre común y que el usuario elija varios itinerarios que mostraremos en el mismo registro?

Al final se optó por el mismo nombre de parada y, en consecuencia, varios itinerarios en el mismo registro. Esta elección se debe a que probablemente los usuarios elijan un nombre únicamente para la parada, como por ejemplo "Casa de pepe" en lugar de usar varios nombres para la misma (por ejemplo, en la misma parada en un sentido "parada para ir a supermercado" mientras que en el otro "a casa de juan"). Además, si tuviéramos varios nombres para la misma parada se podría confundir al usuario haciéndole creer que son paradas distintas.

Esta decisión supuso cambios de la base de datos en el atributo "favorita" de la entidad Parada, que pasó de ser un booleano a una lista de itinerarios favoritos de la parada.

10. Estimaciones de llegada

Inicialmente no se pretendía calcular estimaciones medias de llegada a las distintas paradas de los itinerarios por lo que esta función no está presente en la primera versión.

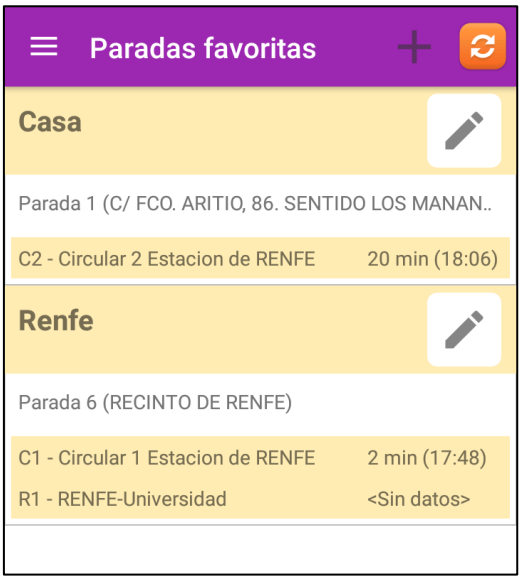


Fig 0.12: Actividad favoritos en la versión 2

11. Buses

Al comenzar el proyecto ni siquiera se planteó la posibilidad de mostrar información de los distintos buses por la poca información que se disponía del sistema y por la falta de una función en la API móvil que nos informara sobre los buses activos. No será hasta la segunda versión que encontraremos referencias a los vehículos y sus distintas posiciones gracias a la información obtenida por el sistema de monitorización. Con los códigos que obtiene la aplicación de escritorio podemos consultar la información sobre los distintos buses que estén activos, haciendo posible una de las funcionalidades más útiles de la aplicación: conocer la posición en tiempo real de los buses de un itinerario.

12. Reglamento

El reglamento en la versión inicial se mostraba usando un enlace de Google Docs al que se había subido el archivo PDF con la información del reglamento. Debido a que este sistema tiene varias desventajas, como la necesidad de disponer de conexión a Internet y que el enlace puede ser borrado en cualquier momento se decidió incluir este archivo en la propia aplicación como Asset adicional. Para la toma de esta decisión se tuvo en cuenta el tamaño del archivo (unos 88Kb) por si esto pudiera provocara que la aplicación ocupara demasiado espacio y repercutiera negativamente. Como es un fichero bastante pequeño se decidió incluirlo al considerar que el impacto sobre la aplicación es mínimo.

13. Búsqueda de parada

Inicialmente sólo se permitía la búsqueda de paradas por código de parada exacto (Figura 0.13). Esta funcionalidad me pareció insuficiente, por lo que para la siguiente versión se incluyó en la búsqueda un filtrado simultáneo por descripción de la parada, itinerario al que pertenecía y código. En esta nueva versión el filtro se aplica instantáneamente conforme el usuario introduce las letras del filtro. Además, el código de parada no se filtra por exactitud, sino usando en la consulta SQL un LIKE 'CODIGO%'. Gracias a esto si el usuario no recuerda el número exacto, sino que recuerda únicamente los primeros dígitos, el sistema le ayuda mostrándole las paradas cuyo código comience por la cifra que ha introducido, enseñando primero las que coincidan exactamente. Un ejemplo de esta ventaja sería si el usuario recuerda que el código de la parada a consultar es un "veintialgo"; el usuario entonces tendría que introducir el carácter 2, y la aplicación instantáneamente le mostraría el resultado con todas las paradas que comienzan por 2. Esta funcionalidad puede observarse en la Figura 0.14.

Además, se planteó también incluir un filtrado de paradas por favorita (si/no), pero se descartó al considerar que si el usuario quisiera encontrar una parada favorita accedería a la opción de "Favoritas" donde las puede encontrar directamente.

14. Festivos

En la segunda versión en la pantalla de "Expediciones de un itinerario" se querían mostrar por defecto las expediciones del día actual sin conexión. Para ello no nos basta con saber qué día de la semana es, sino también si es día festivo o no en Guadalajara. Para ello se creó una tabla nueva que guarde todos los distintos días festivos de la ciudad.

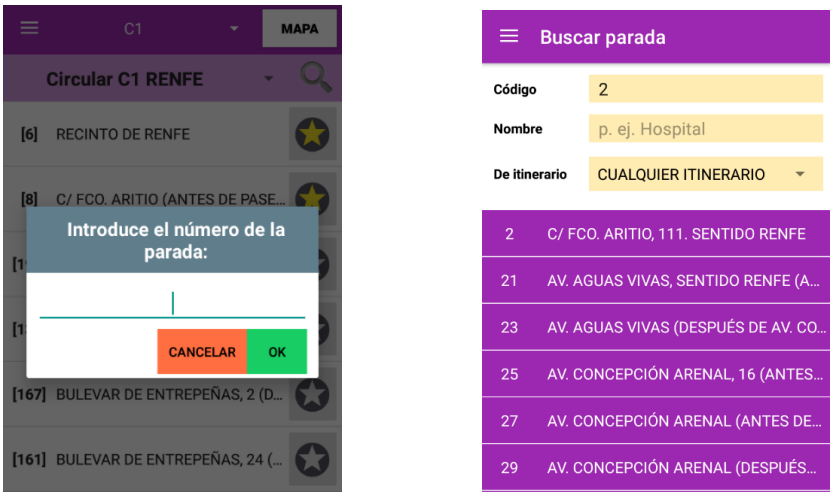


Fig 0.13, 0.14: Búsqueda de paradas – versión 1 (0.13) y versión 2 (0.14)

15. Colores de la interfaz

Para la versión final se cambiaron los colores anteriores por unos similares que son sugeridos en la guía de diseño Materials Design.

16. Tamaño de los desplegables

Como en la versión inicial se daban situaciones en las que un desplegable acababa ocupando gran parte de la pantalla debido a que contiene muchos elementos se modificó el código para que únicamente se ocupara el espacio necesario para mostrar N registros y medio, tomando N = 7 para la versión final. El medio registro adicional se añade para transmitir al usuario que aún hay más elementos ocultos al ver que hay medio elemento visible.

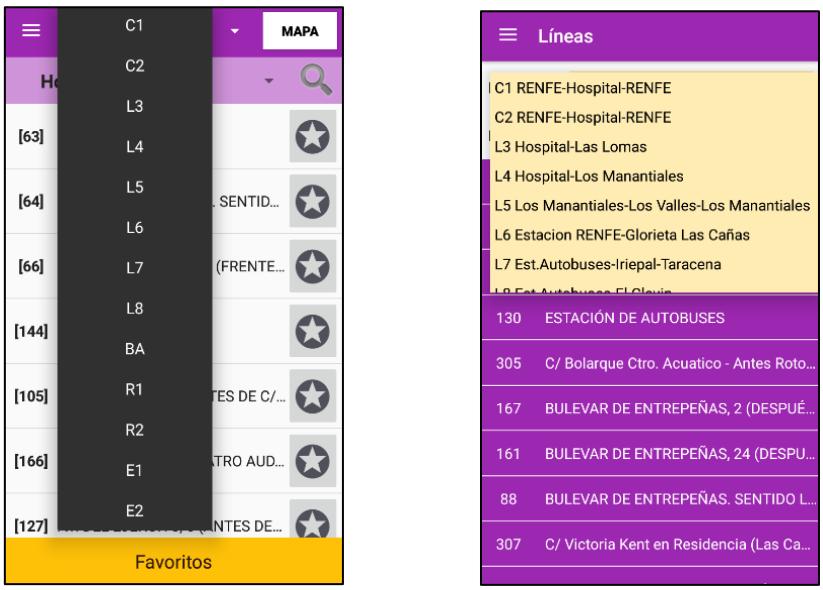


Fig 0.14, 0.15: Desplegable de líneas en versión 1 (0.14) y limitada en la versión 2 (0.15)

10.4 GuadaBus 1.0

Al abrir la aplicación se lleva al usuario directamente a la actividad principal. Esto se debe a que esta actividad incluye las funcionalidades que más usarán los usuarios, como son consultar la información de las distintas líneas, el mapa con su trayecto y consultar y añadir paradas favoritas, como se puede observar en la siguiente sección. Además, gracias al menú de navegación el usuario puede acceder rápidamente al resto de funcionalidades.

Fig 1.1: Actividad principal

10.4.1 Actividad principal

La actividad principal, como se ha indicado anteriormente, concentra prácticamente todas las funcionalidades que más uso darán los usuarios. A continuación, analizaremos las distintas partes que componen la interfaz: La barra de app, el desplegable de itinerarios, el botón de búsqueda, el área de paradas de itinerario, la barra de favoritos y el menú de navegación (este último presente en todas las actividades).

1. **Barra de app**

   Compuesta por el **botón del menú de navegación**, la **barra de líneas** y el **botón de modo**. Al pulsar el botón de navegación se muestra al usuario el menú de navegación, que permite acceder al resto de las actividades haciendo clic sobre ellas como se puede observar en la sección del menú de navegación.

a. **Desplegable de líneas**: Desplegable que al ser pulsado muestra las distintas líneas de la base de datos. Al seleccionar una línea las paradas de la lista (o el mapa si estamos en modo mapa) se actualizarán con los valores correspondientes a la línea seleccionada y por defecto al primer itinerario de la misma.

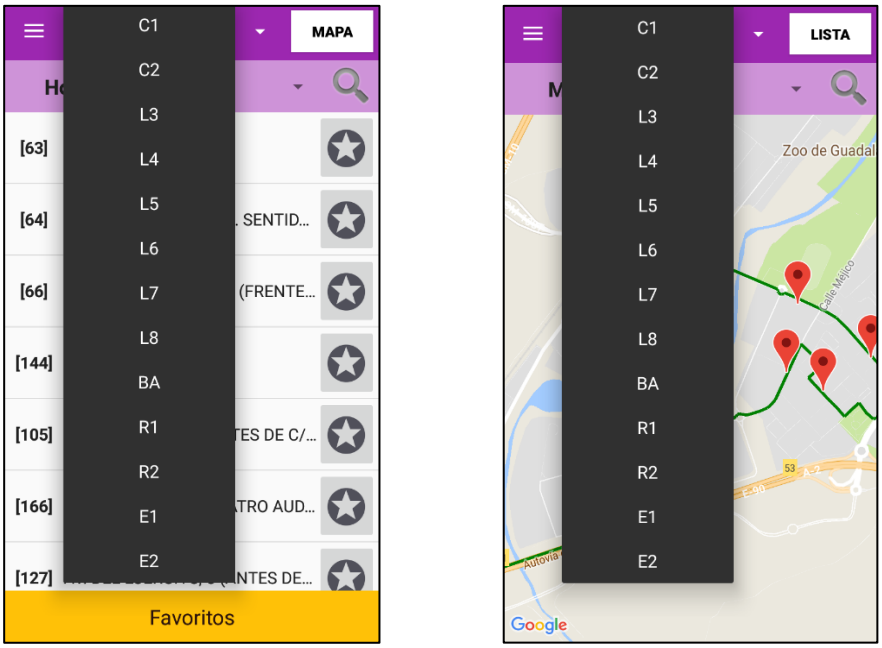


Fig 1.2, 1,3: Actividad principal - desplegable de líneas en modo líneas y mapa

b. Botón de modo: Permite alternar entre los dos modos de la actividad: "Lista" y "Mapa".

En el modo "Lista" se muestra una lista de cada parada de la línea e itinerario seleccionados (Figura 1.4). Se puede ver el código de la parada, su descripción y un botón con forma de estrella. Este botón indica si la parada es favorita o no, mostrándose la estrella de color amarillo cuando es favorita y gris cuando no lo es. Este botón puede ser pulsado para marcar esa parada como favorita.

En el modo "Mapa" se muestran las distintas paradas del itinerario seleccionado sobre Google Maps (Figura 1.5 y 1.6), unidas por un trazo del color de la línea (en la tabla de la base de datos hay una columna que define el color de cada línea).

Fig 1.4: Actividad principal en modo lista

Fig 1.5, 1.6: Actividad principal en modo mapa

2. Desplegable de itinerarios

El selector color rosa bajo la barra de app donde se muestran los distintos itinerarios de la línea seleccionada. Como cada línea puede tener varios itinerarios (de "A" a "B", y de "B" a "A" por ejemplo) es necesario que el usuario especifique cuál de ellos quiere consultar. Al hacer clic sobre ella se desplegarán una lista para seleccionar el itinerario deseado.

Fig 1.7: Actividad principal con desplegable de itinerarios desplegado

3. Botón búsqueda de parada

El botón de búsqueda de parada está situado a la derecha del desplegable de itinerarios y se muestra con el icono de una lupa. Este elemento nos permite acceder a la funcionalidad de búsqueda de paradas. Esta funcionalidad se ha implementado con un popup que se despliega cuando el usuario pulsa sobre el icono de la lupa. Los detalles de este popup se especifican más adelante en la sección "Búsqueda de parada".

Fig 1.8: Botón de búsqueda de parada.

4. Área de paradas de itinerario

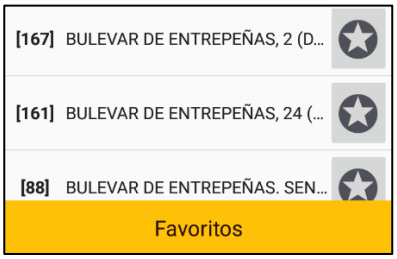
En esta área de la interfaz es donde se muestran las distintas paradas del itinerario seleccionado en el orden natural en que las transita el bus, siendo la primera parada mostrada la primera parada por la que pasará el vehículo. Si el usuario quiere más información de la parada debe pulsar sobre ella; aparecerá un indicador de carga circular que nos indica que la aplicación está tratando de conectar a través de internet para conseguir las estimaciones. Si la conexión con los servidores tiene éxito pasaremos a la vista de "detalles de parada" que se explica más adelante.

Fig 1.9: Área de paradas de itinerario

5. Barra de favoritos

La barra de favoritos se sitúa en la parte inferior de la interfaz. Al pulsar sobre ella se despliega la lista de favoritas (las paradas que hayamos seleccionado con el botón de la estrella). El usuario puede pulsar sobre cualquiera de las paradas favoritas para desplegar el popup de "Detalle de parada" y así obtener más información de la parada, como el nombre real de la parada o los minutos que el sistema estima que faltan para que el siguiente bus llegue a ella.

Además, cada favorita tiene un botón de editar para cambiar el nombre del favorito para que sea más fácil de recordar ("casa" por ejemplo). Para borrar el favorito deberemos pulsar de nuevo sobre el botón-imagen con forma de estrella.



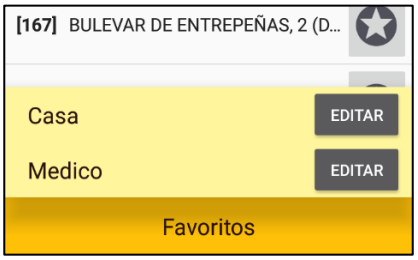


Fig 1.10, 1.11: Barra de favoritos contraida (1.10) y desplegada (1.11)

Cuando se pulsa sobre el botón editar se muestra el siguiente popup para que el usuario edite el nombre personalizado que ha elegido para la parada.

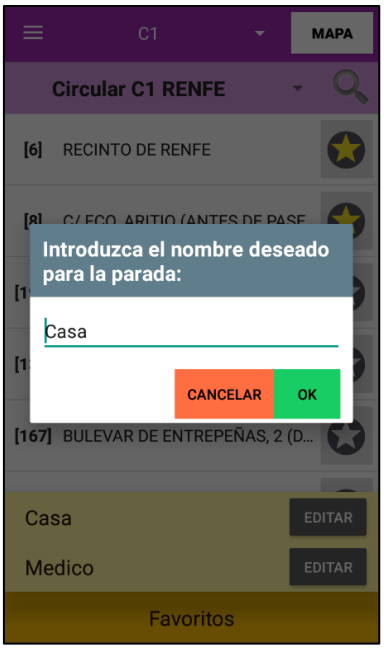


Fig 1.12: Edición del nombre de la parada favorita

10.4.2 Búsqueda de parada

Cuando el usuario pulsa el botón de búsqueda (icono de la lupa) se despliega el popup de búsqueda de paradas. En este popup el usuario deberá introducir el código de la parada que quiere consultar. Si este código corresponde a una parada del sistema se le mostrará directamente el popup de "Detalle de parada". Si por el contrario no existe, o no ha introducido ningún código se le mostrará un mensaje de error.

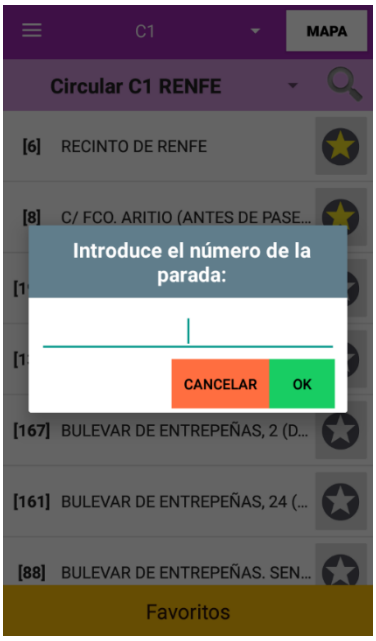


Fig 1.13: Búsqueda de parada

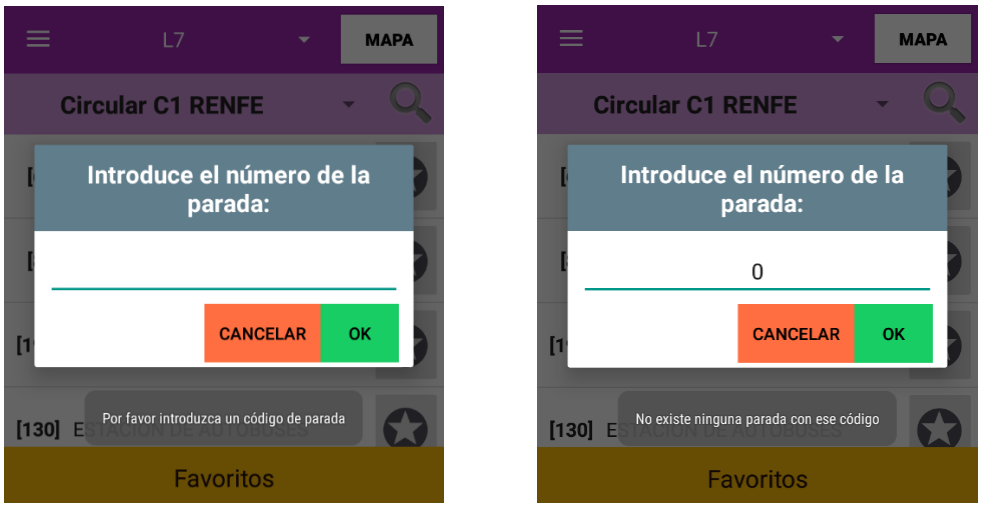


Fig 1.14,1.15: Búsqueda de parada – mensajes de error

10.4.3 Detalles de parada

La interfaz popup de detalles de parada muestra **el nombre real completo de la parada** y **la lista de itinerarios que pasan por ella**, junto a información sobre estos como los minutos en tiempo real que faltan para que el siguiente bus alcance la parada o las horas programadas para las expediciones. Este popup se muestra cuando el usuario selecciona una parada dentro de la actividad principal, ya sea un registro de la lista en el "modo líneas" o un icono de parada en el "modo mapa". En la lista de itinerarios de la parada se resaltará en color lila el itinerario a partir del cual hayamos accedido a el popup (es decir, el que esté seleccionado en el desplegable de itinerarios) ya que se considera de más interés que el resto de los itinerarios.

La información individual de cada itinerario se muestra en el siguiente orden:

- **Nombre del itinerario**
- **Minutos en tiempo real** para que el siguiente bus alcance la parada. Además de los minutos, para ayudar al usuario, se calcula la hora "absoluta" sumando estos minutos a la hora del momento. Cuando no se disponga de conexión a internet se mostrarán iconos de interrogación para indicar que no se dispone de estimación.
- Selector con las **horas de salida desde el inicio** del itinerario programadas **para el día seleccionado en el desplegable**. Como en esta versión no teníamos una hora estimada de llegada, únicamente se mostraban las horas cuando la parada era inicio de parada.

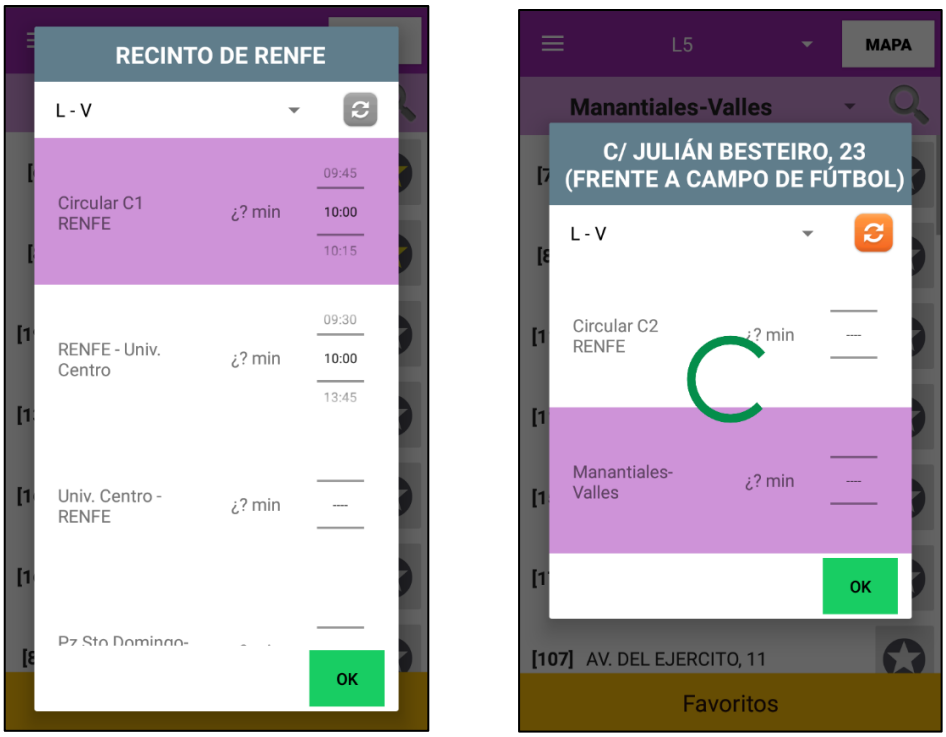


Fig 1.16, 1.17: Detalle de parada – sin conexión (1.16) y cargando estimaciones (1.17).

Bajo el nombre real de la parada encontramos dos elementos que nos permiten interactuar con los detalles: El **selector de día** y el **botón de "refrescar automáticamente"**.

- El **botón de refrescar automáticamente** indica si se desea que las estimaciones de llegada se actualicen constantemente cada cierto tiempo (inicialmente se ha establecido en 10 segundos). Cuando está activado se muestra ligeramente más grande y de color naranja, mientras que al estar desactivado es más pequeño y gris. En caso de perder la conexión el botón se desactiva automáticamente y se muestra al usuario un mensaje indicándole que no dispone de conexión a Internet o los servidores están caídos.
- El **selector de día** nos permite elegir el día para el que queremos que se nos muestre. Como las expediciones siempre eran las mismas entre semana o en día festivo/fin de semana se decidió incluir dos opciones: "L-V" (lunes a viernes) o "Festivo".

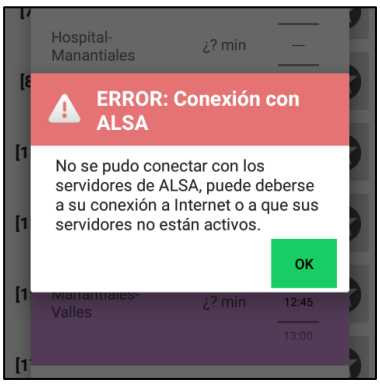


Fig 1.18: Detalle de parada – Error de conexión en el refresco de estimaciones.

Fig 1.19, 1.20: Detalle de parada – selector de día desplegado (1.19) y botón de refresco automático pulsado (1.20).

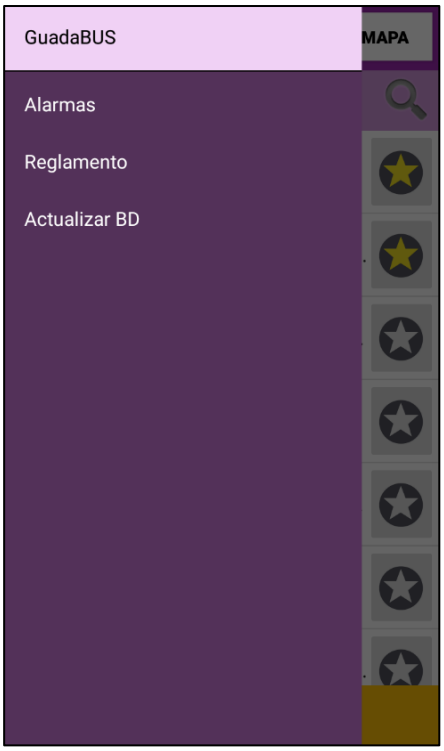
10.4.4 Menú de navegación

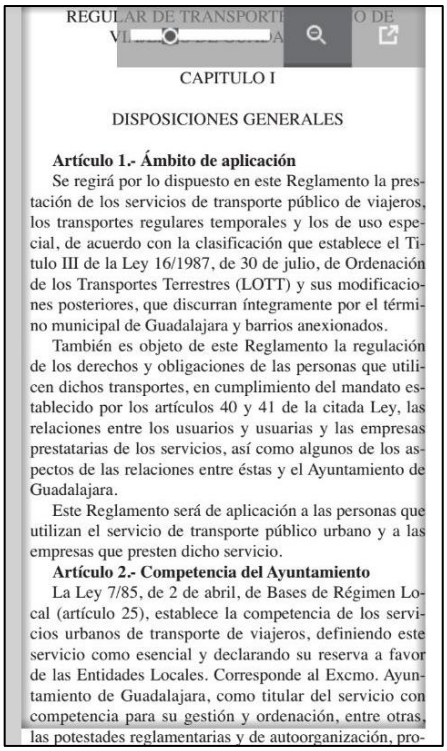
Se puede acceder a este menú haciendo clic sobre el icono de la esquina superior izquierda (icono de hamburguesa) o haciendo un gesto de deslizamiento de izquierda a derecha desde el borde izquierdo de la interfaz en cualquiera de las actividades. En este menú se muestran los accesos a las distintas actividades de forma que el usuario puede moverse de una a otra en cualquier momento (Figura 1.21). La opción de "Alarmas" no se llegó a desarrollar debido al cambio del rumbo del proyecto como se ha indicado en la anterior sección "Diferencias entre versiones".

10.4.5 Actividad reglamento

La segunda actividad es la actividad de reglamento, se puede acceder a ella desplegando el menú de navegación pulsando el botón "hamburguesa" (esquina superior izquierda) o realizando un gesto de arrastre hacia la derecha desde el borde izquierdo de la pantalla. Una vez desplegado el menú el usuario deberá pulsar sobre "Reglamento".

En esta actividad (Figura 1.22) se muestra el reglamento oficial de la empresa encargada de los buses urbanos de Guadalajara a través de un enlace que usaba la aplicación oficial a un documento en Google Docs, por lo que requiere de conexión a Internet. Al pulsar se nos muestra un archivo PDF con toda la información que ha hecho pública la empresa respecto a derechos y obligaciones del usuario del servicio. Se pueden ampliar las distintas partes del documento para poder leer el documento sin problemas gracias a la interfaz que proporciona la plataforma de Google.





REGULAR DE TRANSPORTE ... O DE VIAJEROS ... DE GUADALAJARA

CAPITULO I

DISPOSICIONES GENERALES

Artículo 1.- Ámbito de aplicación

Se regirá por lo dispuesto en este Reglamento la prestación de los servicios de transporte público de viajeros, los transportes regulares temporales y los de uso especial, de acuerdo con la clasificación que establece el Título III de la Ley 16/1987, de 30 de julio, de Ordenación de los Transportes Terrestres (LOTT) y sus modificaciones posteriores, que discurran íntegramente por el término municipal de Guadalajara y barrios anexionados.

También es objeto de este Reglamento la regulación de los derechos y obligaciones de las personas que utilicen dichos transportes, en cumplimiento del mandato establecido por los artículos 40 y 41 de la citada Ley, las relaciones entre los usuarios y usuarias y las empresas prestatarias de los servicios, así como algunos de los aspectos de las relaciones entre éstas y el Ayuntamiento de Guadalajara.

Este Reglamento será de aplicación a las personas que utilizan el servicio de transporte público urbano y a las empresas que presten dicho servicio.

Artículo 2.- Competencia del Ayuntamiento

La Ley 7/85, de 2 de abril, de Bases de Régimen Local (artículo 25), establece la competencia de los servicios urbanos de transporte de viajeros, definiendo este servicio como esencial y declarando su reserva a favor de las Entidades Locales. Corresponde al Excmo. Ayuntamiento de Guadalajara, como titular del servicio con competencia para su gestión y ordenación, entre otras, las potestades reglamentarias y de autoorganización, pro-

Fig 1.21, 1.22: Menú de navegación desplegado (1.21) y actividad reglamento (1.22)

10.4.6 Actualizar BD

Al seleccionar esta opción del menú de navegación se descarga de nuevo de la API toda la información del sistema de autobuses (líneas, paradas, expediciones programadas…) siguiendo el algoritmo descrito en el punto "9.1 Carga de la base de datos" y, después de borrar las tablas, se insertan en la base de datos. Para evitar perder los datos por un posible error se realizan ambas operaciones dentro de la misma transacción. Una vez finalizada la descarga de los datos del sistema se muestra al usuario un mensaje emergente (Toast) indicándole que los datos han sido actualizados correctamente.

10.5 GuadaBus 2.0

Las mejoras respecto a la versión inicial se detallan en profundidad en la sección "Diferencias entre versiones". A continuación, se hace un resumen general de estas modificaciones:

- Interfaz remodelada para facilitar el uso
- Incluidas "estimaciones" de llegada a las paradas de los buses sin conexión.
- Mostrar los autobuses en el mapa en tiempo real además de las paradas.

Al abrir la aplicación se lleva al usuario directamente al menú principal que se detalla a continuación.

10.5.1 Menú principal

El menú contiene cuatro iconos que llevan a las funcionalidades que más usarán los usuarios, como buscar una parada, consultar la información de las distintas líneas, el mapa con sus trayectos y consultar y añadir paradas favoritas, como se puede observar en la siguiente sección. El resto de opciones son accesibles a través del menú de navegación que se despliega al hacer clic en el icono de "hamburguesa".

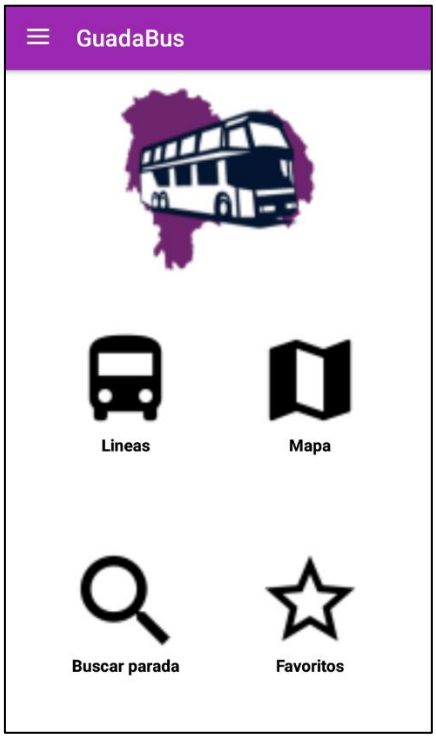


Fig 2.1: Actividad menú principal

10.5.2 Menú de navegación

Este menú se despliega cuando el usuario pulsa el icono de "hamburguesa" o cuando realiza un gesto horizontal de arrastre desde el lado izquierdo hacia el derecho. El menú es accesible desde todas las actividades que aparecen en el mismo y desde el menú principal para que el usuario pueda navegar entre funcionalidades fácilmente desde cualquier punto de la aplicación. Para el resto de pantallas, a las que se accede a través de estas actividades (como puede ser el detalle de una parada) se muestra una flecha que devuelve al usuario a la actividad anterior.

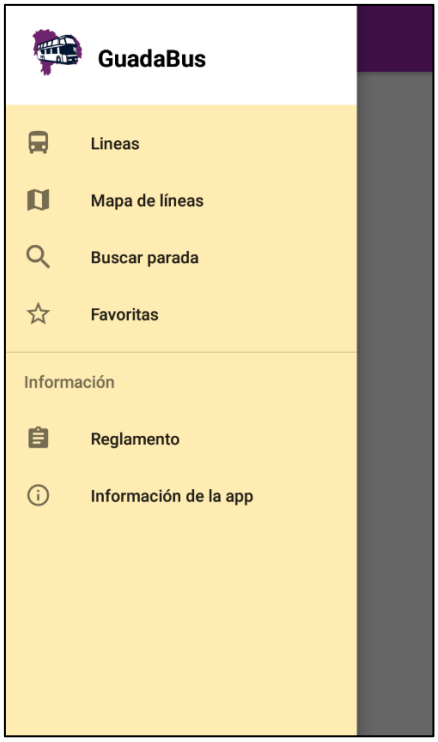


Fig 2.2: Menú de navegación desplegado

Fig 2.3, 2.4: Actividad con icono de hamburguesa (2.3) y actividad con icono de retorno (2.4)

10.5.3 Actividad buscar parada

Esta actividad permite buscar entre las paradas del sistema sin necesidad de disponer de conexión a internet. Al iniciar la actividad se muestran todas las paradas del sistema, y conforme el usuario introduce datos en los filtros estos se aplican automáticamente en la interfaz. Si el usuario quiere

consultar más datos de una de las paradas solo tiene que pulsar sobre la fila deseada y se le dirigirá a la actividad "Detalle de parada".

Dispone de tres filtros:

- **Código**: El código de la parada. Para facilitar la búsqueda se aplica el filtro por similitud, es decir, que si el usuario introduce un "2" le aparecerá la parada número 2 y todas las que comiencen por 2 (21,22…). La similitud se aplica únicamente contra el inicio del texto introducido porque si se aplicara contra la totalidad los resultados podrían ser confusos, siguiendo el ejemplo anterior además de los mencionados aparecerían como resultados 12,32…etc.
- **Nombre**: El nombre que le asignó la empresa a la parada. Suele corresponderse con un lugar interés que esté cercano a la misma o con la calle en la que está situada.
- **Itinerario**: Si el usuario conoce algún itinerario al que pertenezca la parada con este filtro podrá acotar bastante la búsqueda.

Fig 2.5, 2.6: Búsqueda sin filtros (2.5) y con filtro de código y descripción (2.6)

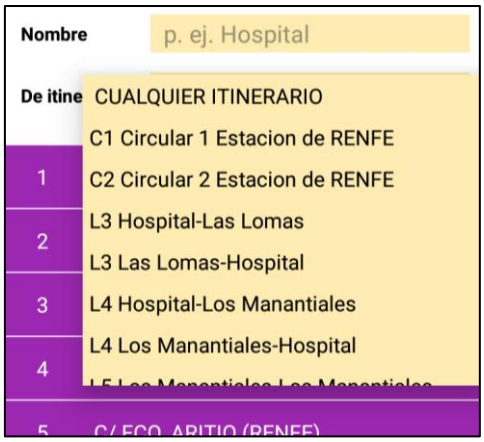


Fig 2.7: Búsqueda de paradas - filtro de itinerario desplegado

10.5.4 Actividad detalle de parada

A esta actividad se llega dirigido desde alguna las cuatro actividades principales: "Búsqueda de parada", "Líneas", "Mapa" y "Favoritas". Pulsando el botón de la esquina superior izquierda se devuelve al usuario a la actividad que lo dirigió a esta pantalla.

En la interfaz se muestra:

- **Código** de la parada
- **Descripción** de la parada
- **Si es parada favorita o no**.
- **Si es parada favorita**, se muestra **el nombre personalizado** que le ha asignado el usuario. A la derecha del nombre hay un botón de edición por si el usuario quiere modificar el nombre que le asignó a la parada favorita.
- La lista de **itinerarios de la parada**. Para cada itinerario se muestran (si está activado el refresco de estimaciones) los **minutos que quedan para que el siguiente autobús** llegue a la parada. Si no está activado el refresco se muestra el literal "<desactivado>".
  Además, junto a la estimación se encuentra un **icono de estrella (para marcar el itinerario como favorito)** y un **icono de calendario (para ver las expediciones del itinerario programadas en el día para esa parada)**. Si el usuario selecciona "Marcar itinerario favorito" se comprueba si la parada ya tenía algún itinerario favorito o no. Si aún no era favorita significa que no disponemos de un nombre personalizado por lo que se pedirá al usuario que introduzca uno.

Pulsando el botón "**Ver en mapa**" el usuario puede consultar sobre el mapa de Google Maps la posición exacta de la parada, e incluso como llegar redirigiéndole a la aplicación oficial.

El botón de la esquina superior derecha es el **botón de "refresco automático"**. Si está activado se mostrará naranja y grande, mientras que si está desactivado su color cambia a gris y su tamaño disminuye. Cuando está activado la estimación de llegada de la interfaz se actualizará con la obtenida del sistema de autobuses en tiempo real.

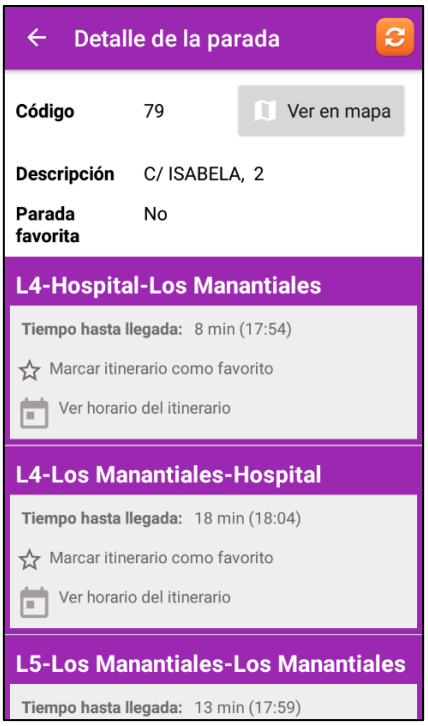


Fig 2.8, 2.9: Detalles de parada – parada NO favorita (2.8) y parada favorita (2.9)

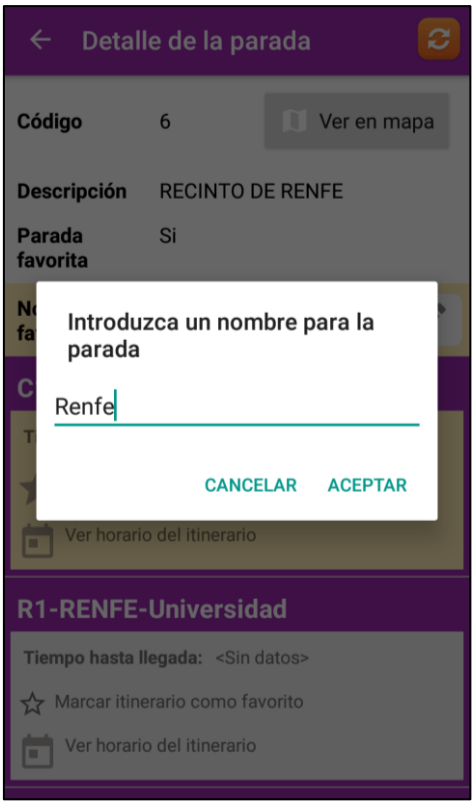


Fig 2.10: Detalles de parada – Pop up para asignarle o editar el nombre de la parada favorita

10.5.5 Actividad expediciones de itinerario en parada

Muestra las expediciones programadas para un itinerario en una parada en el día seleccionado. Las expediciones se indican en la tabla de la mitad inferior, divida en dos columnas. En la primera columna se muestran las horas exactas oficiales de salida desde el inicio del itinerario, las que la empresa se ha comprometido a cumplir. En la segunda columna se muestran las expediciones calculadas con el sistema de monitorización, es decir, la hora media a la que suele pasar el autobús por la parada consultada. Este dato debe usarse solo a nivel orientativo, por lo que si se desea saber el tiempo de llegada para el siguiente bus se debe consultar la estimación en tiempo real en la pantalla detalle ya que esta se actualiza si hay una incidencia, además de que (debería) ser más precisa que la media que hemos calculado.

El usuario puede modificar el día consultado con el desplegable de día. Por defecto este desplegable tiene el valor del día de la consulta, detectando si es festivo gracias a la tabla "Festivos" de la base de datos.

Si la parada consultada es la que inicia el itinerario no tiene sentido que mostremos una hora media ya que se supone que la expedición iniciará exactamente a la hora indicada por el sistema (Hora de salida de inicio de línea). Por ello en este caso en la columna de "Hora media estimada" mostrará el mensaje "[INICIO DE ITINERARIO]".

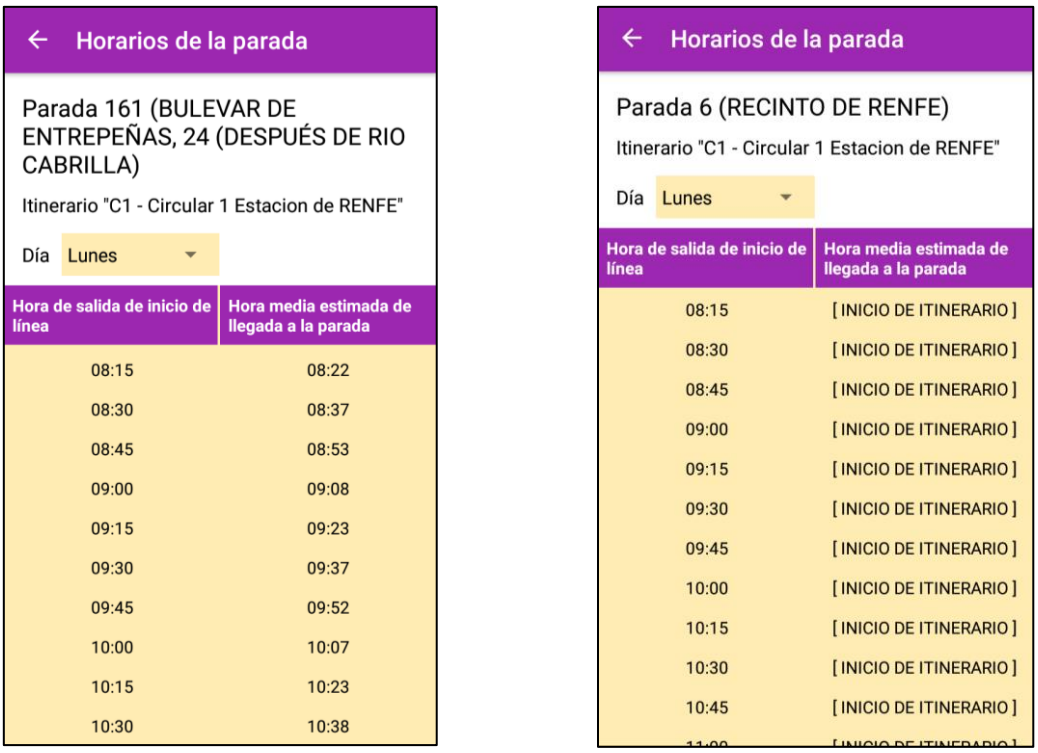


Fig 2.11, 2.12: Expediciones de itinerario en parada – parada no inicial (2.11) y parada inicial (2.12)

10.5.6 Actividad mapa

En esta actividad se puede consultar el recorrido de los itinerarios y las posiciones de los buses que lo están realizando en tiempo real. La aplicación solicita permiso para conocer la ubicación del usuario, si se acepta la solicitud se mostrará la posición del usuario sobre el mapa con un símbolo circular azul.

Al pulsar sobre un icono de autobús (azul) se muestra el **código del autobús**, el **código de la parada a la que se dirige** y **el nombre del itinerario que está realizando** (este dato será útil en un futuro si queremos mostrar varios itinerarios simultáneamente).

Cuando se pulsa el icono de una parada (rojo) se muestra el **código de parada** y la **descripción de la parada**. Si se desean conocer más detalles de la parada como por ejemplo las estimaciones de tiempo se puede pulsar de nuevo en la información para que nos dirija a la actividad "Detalle de parada".

Una de las ventajas de almacenar las coordenadas en la base de datos es que, si antes se ha consultado esta región del mapa, se puede consultar **sin conexión** ya que la zona estará cacheada y no hay que hacer consultas para obtener la ruta.

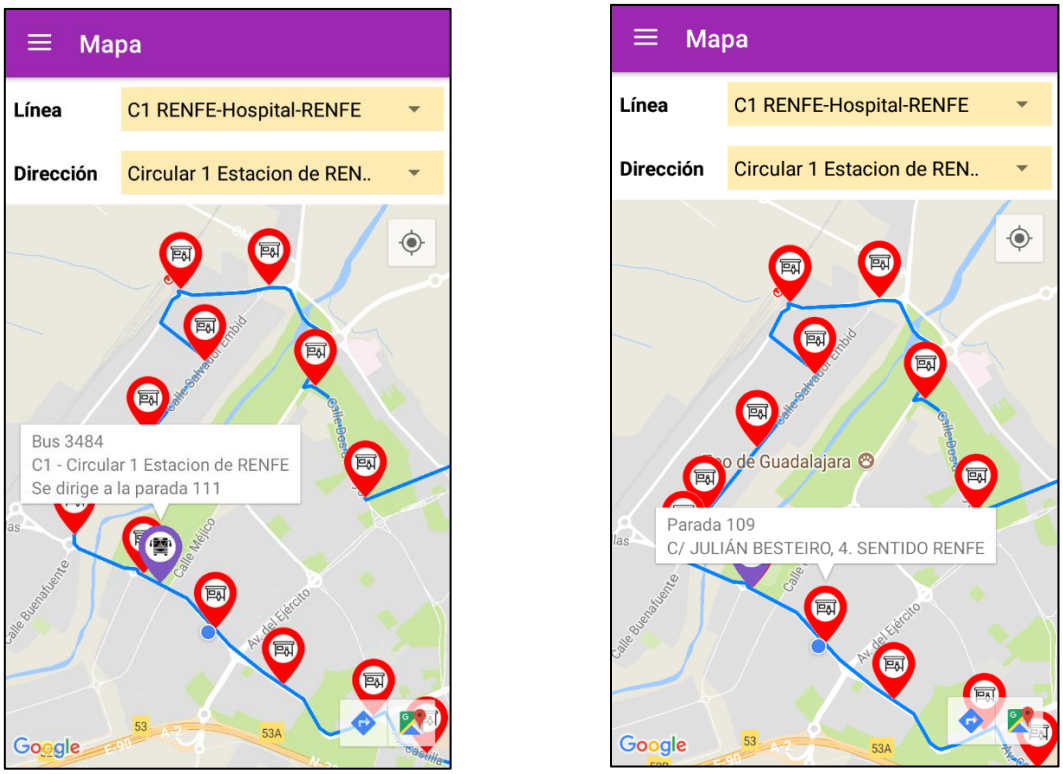


Fig 2.13, 2.14: Mapa – Bus seleccionado (2.13) y parada seleccionada (2.14)

10.5.7 Actividad favoritas

Uno de los objetivos de la aplicación, el de facilitar la consulta sin conocer los códigos de las paradas, supone que esta actividad debería ser la usada con más frecuencia. Lo que se busca con esta App es facilitar el día a día de las personas que usan el servicio, por lo que esta actividad muestra únicamente los datos que le puedan interesar: **Nombre personalizado** y **estimación de llegada**. Si se desea más información de alguna de las paradas favoritas basta con pulsar sobre la fila de la parada para que se nos dirija a la actividad "Detalle de una parada".

El primer elemento es el **nombre personalizado** que el usuario la haya asignado junto a un botón que permite editarlo, seguido del **nombre real de la parada** (en un tamaño más pequeño al ser menos relevante) y a continuación la **estimación en tiempo real de los itinerarios** que ha marcado como **favoritos** de la parada.

El **botón de añadir favorito** (símbolo del "mas") nos dirige a la actividad de búsqueda, para que el usuario pueda encontrar y marcar como favorita la parada deseada seleccionando el/los itinerario/s en que esté interesado.

El botón de la esquina superior derecha es el **botón de "refresco automático"**. Si está activado se mostrará naranja y grande, mientras que si está desactivado su color cambia a gris y su tamaño disminuye. Cuando está activado la estimación de llegada de la interfaz se actualizará con la obtenida del sistema de autobuses en tiempo real.

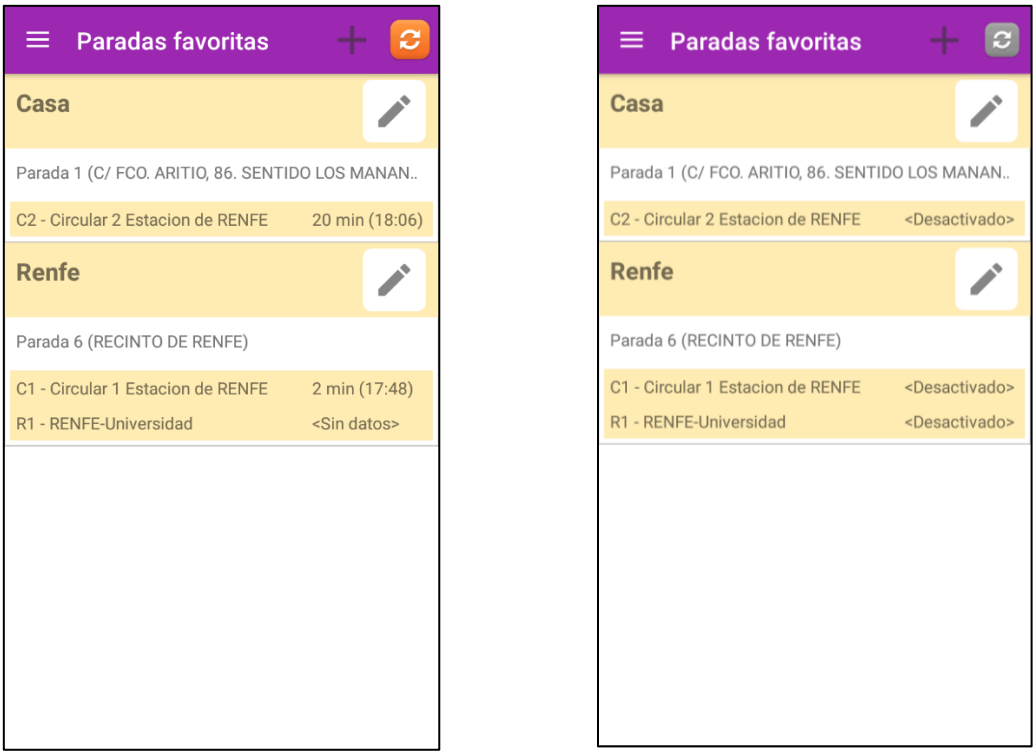
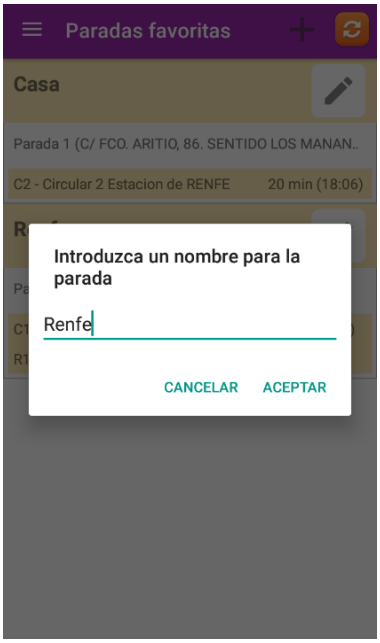


Fig 2.15, 2.16: Favoritas – Auto refresco activado (2.15) y auto refresco desactivado (2.16)

10.5.8 Actividad líneas

La actividad "líneas" (Figura 2.18) muestra las paradas que componen el itinerario seleccionado ordenadas de inicio a final del recorrido. El usuario puede seleccionar la ruta que quiere consultar con el desplegable "Línea" y dentro de la propia línea puede seleccionar los distintos itinerarios de esta. De cada parada se muestran su código y la descripción que le puso la empresa. Si se desea más información de alguna parada basta con pulsar sobre la fila de la parada para que se nos dirija a la actividad "Detalle de una parada".



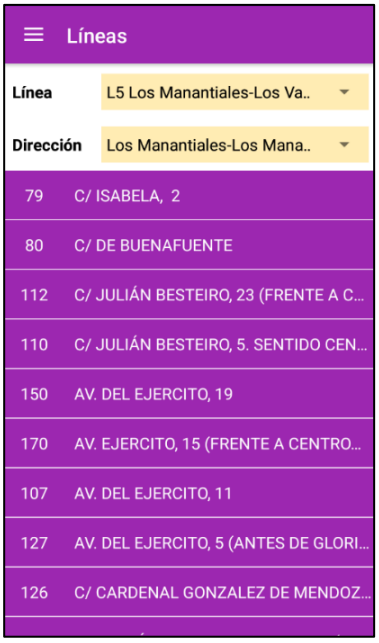


Fig 2.17,2.18: Favoritas – Edición de nombre personalizado (2.17) y actividad Líneas (2.18)

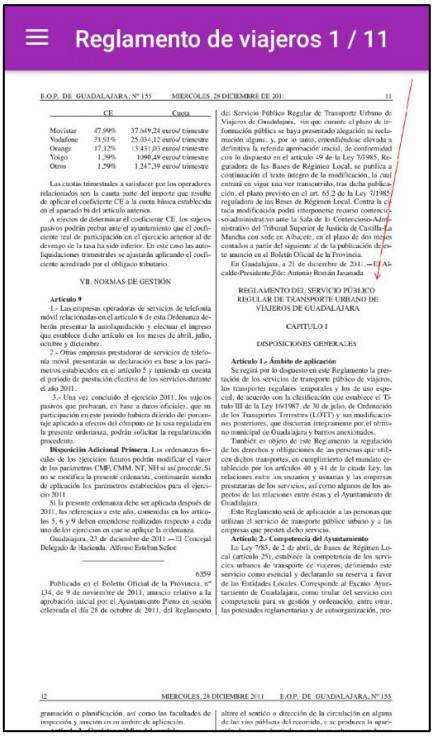
10.5.9 Actividad reglamento

En esta actividad (Figura 2.19) se muestra el reglamento oficial de la empresa encargada de los buses urbanos de Guadalajara. Al pulsar se nos muestra el archivo PDF incluido como Asset en la aplicación con toda la información presente en el BOP (Boletín Oficial de la Provincia) de Guadalajara respecto a derechos y obligaciones del usuario del servicio. Gracias a la librería "Android PDF Viewer" se puede interactuar con el documento pudiendo ampliar o desplazarse entre páginas de forma fluida. Al estar incluida como Asset esta información se puede consultar sin que sea necesario disponer de conexión a Internet.

10.5.10 Actividad información de la app

Esta actividad (Figura 2.20) tiene los siguientes cometidos:

- Informar al usuario de la vía de comunicación de errores o sugerencias con el desarrollador de la aplicación. Todo feedback de los usuarios que recibamos nos ayudará a mejorar la aplicación.
- Desligar el desarrollo de la aplicación de la empresa que ofrece el servicio.
- Agradecer al usuario que use nuestra aplicación.



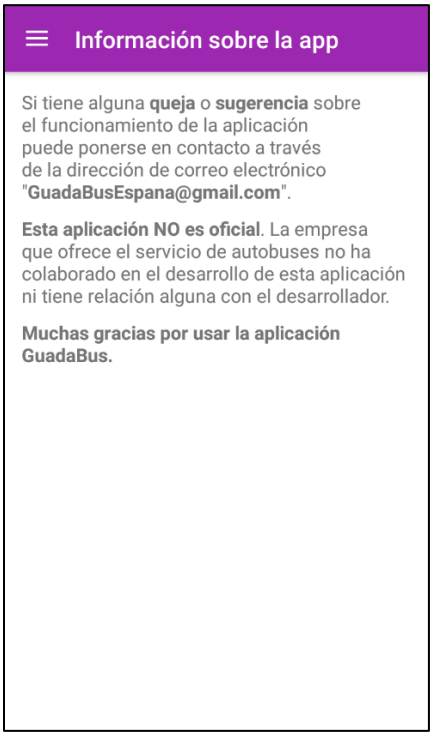


Fig 2.19, 2.20: Actividad reglamento (2.19) y actividad información sobre la App (2.20)

11. Presupuesto

Coste de material

Material	Precio	Duración	Uso	TOTAL
Ordenador portátil	500€	3 años	3 meses	41,70€
Smartphone Android	120€	3 años	2,5 meses	8,4 €
Conexión a Internet	32 €/mes	-	3 meses	96,00€
Cuenta de desarrollador de Google Play	21,24€	-	-	21,24€

Coste total del material: 167,34 euros.

Coste por tiempo de trabajo

Función	Horas totales	Precio/hora	TOTAL
Programador Android Junior	450 horas	8,5€/hora	3.825€

Coste total de los recursos humanos: 3.825 euros.

Coste total de ejecución material

Es la suma de los importes de los costes de material y el coste por tiempo de trabajo (mano de obra).

Coste de material	167,34 €
Coste por tiempo de trabajo	3.825,00 €
Coste total de ejecución material	3.992,34 €

Gastos generales y beneficio industrial

Esta sección incluye los gastos en instalaciones y otros gastos adicionales producto del proyecto. Los gastos generales y beneficio industrial se obtienen aplicando un el 20% del coste total de ejecución material. Los gastos generales y el beneficio industrial de nuestro proyecto sería 3.992,34 * 0,2 = 798,46 €.

Presupuesto de ejecución por contrata

Este equivale a la suma de los gastos generales y el coste total de ejecución material. Para nuestro proyecto 4.790,80 €.

Honorarios

Los honorarios se calculan de acuerdo con las tarifas de los ingenieros en trabajos particulares dictadas por el Colegio Oficial de Ingenieros Técnicos de Telecomunicación.

Los derechos del visado se calculan multiplicando el presupuesto total de ejecución material por el coeficiente reductor multiplicado por el porcentaje que nos indican. Como el presupuesto de ejecución material es inferior a 5 millones se usará un porcentaje del 7% y un coeficiente reductor de 1. Con estos valores obtendremos el total de los honorarios del proyecto.

0,07 * 1 * 3.992,34 = 279,46€.

Coste total del proyecto

El coste total del proyecto se calcula sumando el presupuesto de ejecución por contrata a los honorarios, y una vez calculado aplicando el impuesto de valor añadido:

Presupuesto de ejecución por contrata	4.790,80 €
Honorarios	279,46 €
TOTAL SIN IMPUESTOS	**5.070,26 €**
IVA (21%)	1.064,75 €
TOTAL	**6.135,01 €**

El coste total del proyecto asciende a SEIS MIL CIENTO TREINTA Y CINCO EUROS CON UN CÉNTIMO.

Cabe recalcar que, si la empresa hubiera colaborado con información sobre el funcionamiento de su API, el modelo de datos que usan y hubieran facilitado información sobre tiempos medios que probablemente tengan se estima que se hubiera ahorrado en total unos 15 días de trabajo a jornada completa, reduciendo el coste total en unos 1234,20 euros.

12. Problemas encontrados

A continuación, se detallan algunos de los problemas que surgieron durante el desarrollo del proyecto y la solución que se ha aplicado para evitarlos.

- **Latencia en SQLite**. Como la aplicación de monitorización trabaja con varios hilos simultáneamente estos deben terminar su tarea lo más rápido posible para liberar la conexión para el resto de los hilos que puedan necesitarla. Durante el desarrollo de la aplicación se observó que estos tardaban un tiempo anormalmente largo para operaciones tan simples como UPDATEs e INSERTs de un solo registro. Investigando un poco en foros de internet se descubrió que la configuración por defecto de la base de datos tenía el modo "autocommit" activado, por lo que para realizar N operaciones había que añadir el tiempo de los N commit.

  Para evitar este problema se desactivó el autocommit por defecto, delegando esta tarea en el propio hilo y además se programaron para que compartieran la conexión a base de datos.

- **Gestión de permisos de Android**. Al inicio del proyecto se testeaba usando un smartphone con el sistema operativo Android 4.4, pero más adelante se cambió a un dispositivo con la versión 6.0.1. Con este cambio de dispositivo la aplicación dejó de funcionar, dando un error de ejecución al iniciar. Después de investigar un poco descubrimos que a partir de la versión de Android 6.0 se había cambiado el funcionamiento de los permisos, y que ya no bastaba con solicitarlos al instalar la aplicación, sino que debíamos solicitarlos cada vez que quisiéramos hacer uso de ellos. Como la aplicación intentaba realizar las acciones sin pedir permiso explícito al usuario fallaba irremediablemente.
  Se evitó este problema modificando el código para que la aplicación solicite los permisos antes de realizar acciones como, por ejemplo, solicitar la posición GPS del usuario.

- **Errores de la API de autobuses**

  - **Líneas circulares**: Cuando consultamos las paradas de un itinerario circular la lista de paradas que nos devuelve la API no contiene la última parada (que a su vez es la primera al ser circular). Para solventar esto se han editado manualmente esos registros en nuestra base de datos local.

  - **Festivos**: La API no informa de si el día es festivo como tal. Para solucionarlo se han cargado las fechas de los días festivos en Guadalajara de los siguientes 5 años en la base de datos. Lo ideal sería un servicio externo del que obtener la información, pero no se ha encontrado ninguno.

  - **Líneas Especiales:** No se han podido obtener datos de algunas líneas debido a que tienen lugar en situaciones muy concretas, como las líneas de Emergencia E1 y E2, algunos servicios bajo demanda…etc. No se ha encontrado forma de solventar este problema, por lo que estas líneas no se muestran a los usuarios.

- **Días de los servicios nocturnos:** Para el sistema de autobuses una expedición de un autobús nocturno que sale a las 02:00 AM la madrugada del sábado en realidad no transcurre en sábado sino en viernes. Inicialmente la orientación del proyecto fue seguir los datos a rajatabla según indican los calendarios, pero finalmente para facilitar los cálculos se terminó adoptando el criterio del sistema. Esto provoca que el final efectivo del día tenga lugar a partir de las 05:00 AM.

- **Errores de datos de la API:** Los datos de la API muchas veces parecen ser erróneos, a continuación, se enumeran algunos de los que hemos observado:

    - Expediciones duplicadas: En ocasiones las expediciones aparecen repetidas. Una posible explicación es que salgan dos vehículos a la vez a esa misma hora, aunque en mi opinión esto no tiene mucho sentido ya que es una hora a la que se da un servicio, no un vehículo en particular.

    - Expediciones erróneas: La aplicación de monitorización ha obtenido en varias ocasiones en la línea Circular 1 una expedición que tiene lugar a las "99:99".

    - Paradas de itinerario que al consultarlas "no existen": Se ha observado que tras solicitar las paradas de un itinerario e iterar por ellas en ocasiones la API nos informa de que el estado de la parada es "NO DESC". En el año 2017 se actualizaron los códigos de las paradas y esto puede haber producido algún error en los datos, ya que este problema solo se ha observado en algunas de las paradas modificadas.

    - Líneas fantasmas: No se ha conseguido obtener las paradas de algunas líneas como por ejemplo el "Búho C", ni aun usando los datos que proporcionan del servicio en su página web.

    - Parámetros y atributos sin utilidad aparente: En la respuesta de la mayoría de las funciones de la API los datos que obtenemos contienen muchos atributos con valor "null" en múltiples campos cuya finalidad se desconoce. Además, en algunas funciones hay parámetros obligatorios que no afectan en absoluto al resultado, como por ejemplo en la función datosVehiculo el parámetro "idEmpresa". Lo único que cambia cuando modificamos este parámetro es el atributo "idEmpresa" de la respuesta, atributo que, por supuesto, no nos sirve en absoluto para la aplicación.

    - Coordenadas suministradas por la API poco precisas. A la hora de pintar los itinerarios sobre el mapa en las curvas se observa una sucesión de rectas que, en muchas ocasiones, se salen de la carretera. La solución propuesta para este problema se detalla en la sección "Trabajo futuro".

13. Conclusiones

Para empezar, me gustaría resaltar las desastrosas consecuencias de no haber realizado la fase inicial de análisis de los requerimientos correctamente y así empezar un proyecto rumbo a un destino que nadie desea. Esta situación deberían haberla subsanado una vez detectado este problema, pero ya van más de cuatro años en que se desoyen sistemáticamente las quejas de los usuarios, que multitudinariamente señalan las carencias de la aplicación que nos ofrecen. Con este proyecto he intentado, y espero haberlo conseguido, llenar los vacíos que tiene el proyecto oficial.

Centrándome en la parte positiva, gracias a este proyecto he adquirido bastante conocimiento de los sistemas Android. Cada nueva funcionalidad que empezaba a codificar conllevaba alguna lección desconocida a la que había que adaptarse. El ciclo de vida de una actividad y como tratar los distintos estados por los que puede pasar una aplicación. El sistema de permisos y la evolución lógica que ha tenido en sus progresivas versiones. La importancia de tener en cuenta las distintas versiones de Android, y como intentar abarcar todas ellas puede suponer un gran trabajo en algunas situaciones. Las facilidades y el tiempo de codificación que ahorra el uso de librerías como GSON para tratar estándares JSON. La gran libertad que ofrece la plataforma para interactuar con la inmensa cantidad de elementos que pone a nuestra disposición un smartphone, complementada con una documentación oficial y una comunidad de colaboradores que evita que el programador novel se pierda en el abismal abanico de posibilidades que ofrece.

Curiosidades como las buenas prácticas que fuerza el propio sistema como lanzar una excepción en tiempo de ejecución para evitar que en el hilo principal se realicen peticiones que dependan de la red (es decir, que probablemente vayan a tardar un tiempo considerable) y así impedir que se deje la aplicación "congelada" durante ese periodo.

En cuanto al diseño valoro la gran ayuda que supone disponer de herramientas como Material Design para los programadores sin sentido alguno de la estética, invitándonos a combinar colores usando sus complementarios y mejorando enormemente la apariencia de la aplicación. El funcionamiento de los "pesos", "gravedades", alturas y anchuras para diseñar una interfaz que se adapte a cualquier pantalla en un mundo en el que cada dispositivo tiene distinta resolución. La importancia de transmitir más información con menos espacio al tener "los pixeles contados" debido a los dispositivos con pantallas más minúsculas. La reafirmación de conocimientos anteriores como el informar al usuario constantemente de los posibles tiempos de carga para que no se ponga nervioso y cierre la aplicación a los dos segundos de estatismo.

Como reflexión final me gustaría señalar la vital importancia que tiene ofrecer a los usuarios unos servicios mínimos de información en un sistema de transporte como es el de autobuses urbanos. Ya que disponemos de los medios físicos necesarios lo mínimo que podemos hacer es usar la información que estos nos proporcionan y hacerla útil para el mayor número de usuarios, en lugar de almacenarla en un cajón sin uso.

14. Trabajo futuro

El proyecto todavía puede continuar desarrollándose en distintos aspectos, a continuación, describo algunas de las ideas para futuros desarrollos:

- **Horarios de temporada baja**: Como la API no nos devuelve los horarios de la temporada baja hasta que no llegue la fecha de inicio de la misma (10 de julio), tendremos que obtener de nuevo todas las expediciones, añadiendo un nuevo campo que indique si son expediciones de temporada baja o no.

- Configurar la aplicación de **monitorización** para que sea **totalmente autónoma** y cargarla en una Raspberry Pi para evitar depender del portátil usado para el proyecto.

- Incluir los distintos **puntos de recarga de la tarjeta** de autobuses en el mapa.

- Invitar a los usuarios a que nos envíen feedback sobre la aplicación mediante un popup cuando la hayan usado un tiempo medio. Actualmente la única vía de comunicación es que nos escriban un correo electrónico a la dirección indicada en la pantalla "Información de la app", y probablemente muchos usuarios no estén dispuestos a tomarse la molestia de escribirlo. En ese mismo popup también se podría **enlazar a la tienda de aplicaciones** para invitar al usuario a que puntúe la aplicación

- **Recalcular usando la API de Google Maps las coordenadas de ruta** proporcionados por el sistema de buses. El problema se debe a que un porcentaje de los puntos geográficos no son precisos, por lo que en ciertas situaciones el trazado de la ruta se desvía de la carretera como se muestra en el siguiente ejemplo. Con la API de Google Maps para rutas en modo vehículo podemos obtener las coordenadas reales

- **Optimización de la monitorización y la consulta de los buses de una línea**: Solicitar a la empresa que se incluya en el sistema una función nueva para la API que informe de qué buses están activos, para así evitar tener que encontrarlos simulando ser un usuario o por fuerza bruta.

- Incluir una función que indique **la parada más cercana al usuario para llegar a un punto**. Como versión inicial básica se podría habilitar un botón que mostrase en el mapa todas las paradas del sistema sin que sea necesario seleccionar una línea, de modo que el usuario podría elegir la parada manualmente sin necesidad de conocer que línea es la adecuada

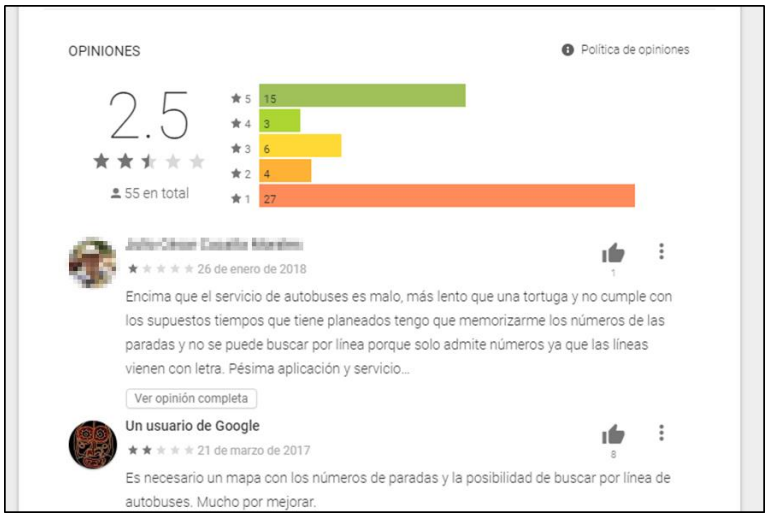
15. Bibliografía

Enlace a la aplicación oficial del servicio de autobuses urbanos de Guadalajara

Obtenido de https://play.google.com/store/apps/details?id=com.indra.alsaqr&hl=es_419

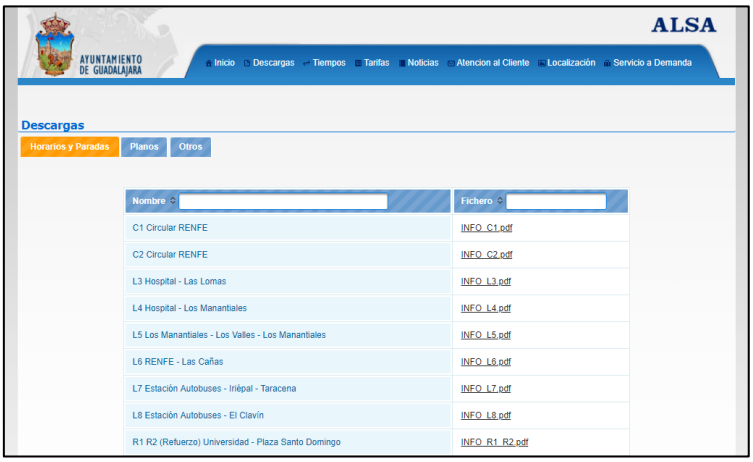
Opiniones negativas en la sección de comentarios de la aplicación oficial del servicio de autobuses urbanos de Guadalajara

Obtenido de https://play.google.com/store/apps/details?id=com.indra.alsaqr&hl=es_419



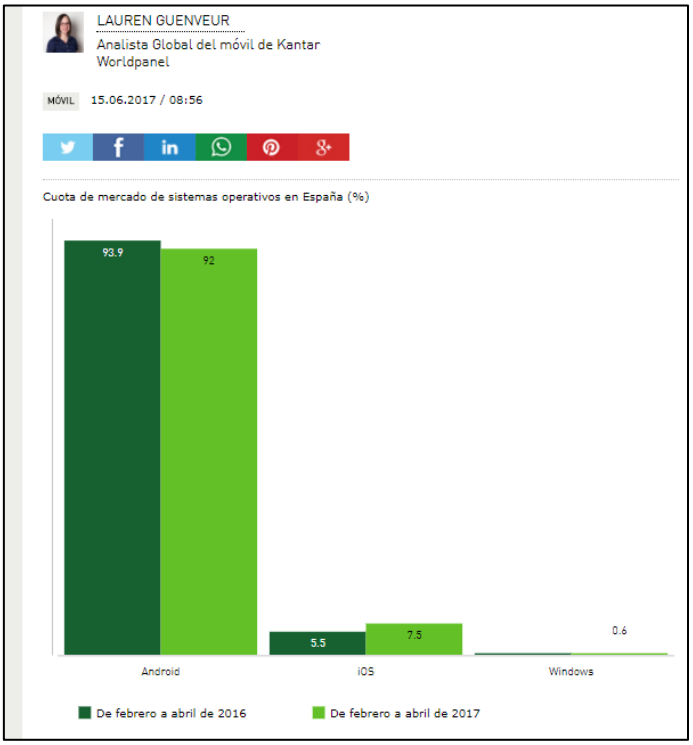
Web oficial con información de las líneas

Obtenido de http://urbanos.guadalajara.es/descargas/descargas.xhtml



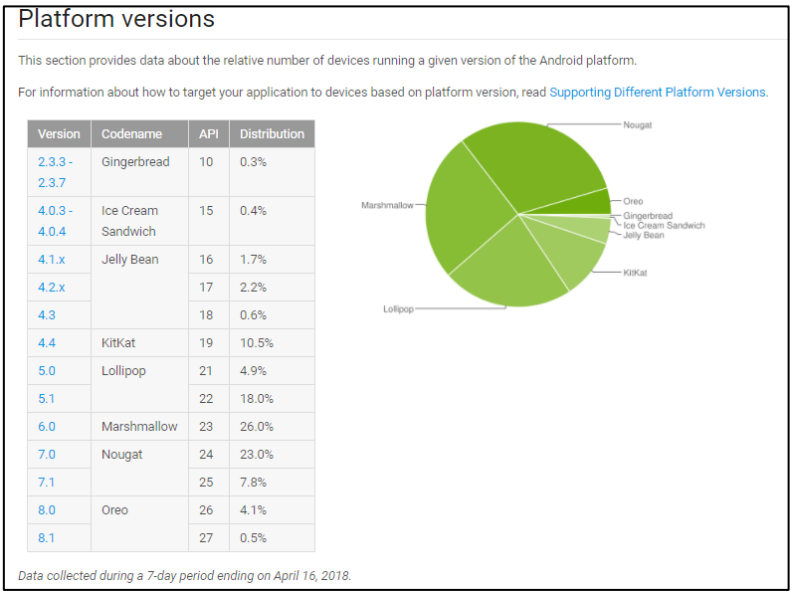
Cuota de mercado de SSOO de smartphone en España (inicio 2017)

Obtenido de https://es.kantar.com/tech/m%C3%B3vil/2017/junio-2017-cuota-de-mercado-de-smartphones-en-espa%C3%B1a-2017/



Distribución de uso de las distintas versiones de Android

Obtenido de https://developer.android.com/about/dashboards/index.html



Platform versions

This section provides data about the relative number of devices running a given version of the Android platform.

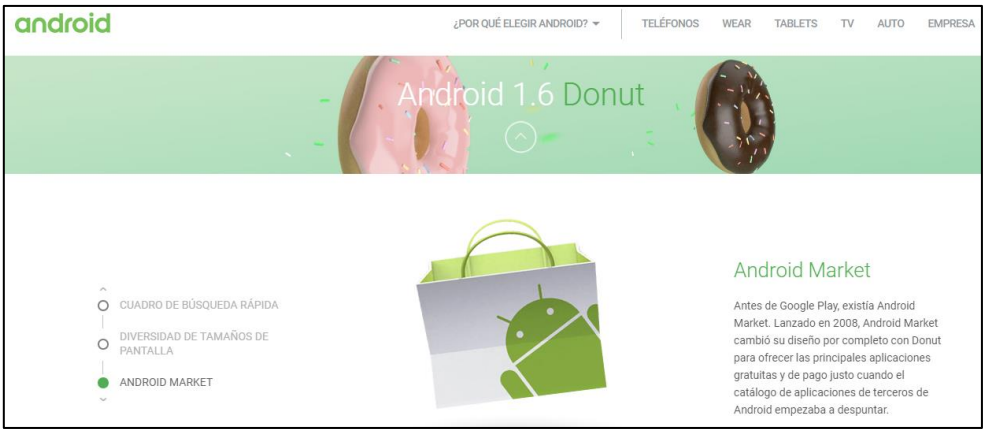
For information about how to target your application to devices based on platform version, read Supporting Different Platform Versions.

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.3%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.4%
4.1.x	Jelly Bean	16	1.7%
4.2.x		17	2.2%
4.3		18	0.6%
4.4	KitKat	19	10.5%
5.0	Lollipop	21	4.9%
5.1		22	18.0%
6.0	Marshmallow	23	26.0%
7.0	Nougat	24	23.0%
7.1		25	7.8%
8.0	Oreo	26	4.1%
8.1		27	0.5%

Data collected during a 7-day period ending on April 16, 2018.

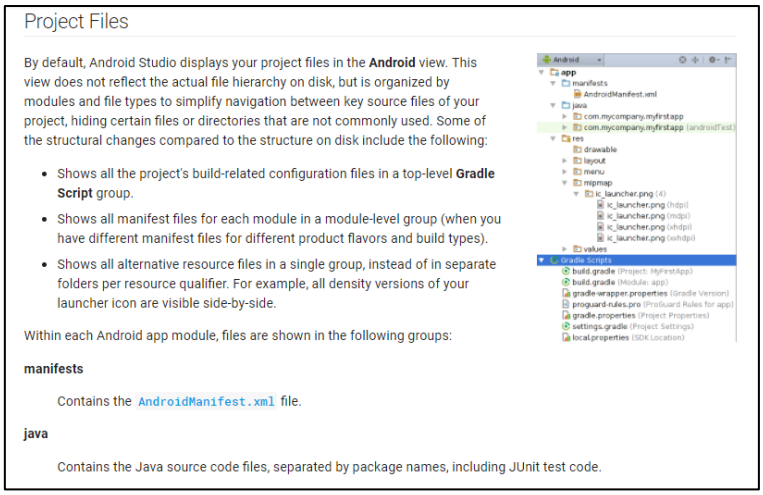
Mejoras de las distintas versiones de Android

Obtenido de https://www.android.com/intl/es_es/history/#/donut



Estructura de una aplicación Android

Obtenido de https://developer.android.com/studio/projects/#ProjectFiles



Project Files

By default, Android Studio displays your project files in the **Android** view. This view does not reflect the actual file hierarchy on disk, but is organized by modules and file types to simplify navigation between key source files of your project, hiding certain files or directories that are not commonly used. Some of the structural changes compared to the structure on disk include the following:

- Shows all the project's build-related configuration files in a top-level **Gradle Script** group.
- Shows all manifest files for each module in a module-level group (when you have different manifest files for different product flavors and build types).
- Shows all alternative resource files in a single group, instead of in separate folders per resource qualifier. For example, all density versions of your launcher icon are visible side-by-side.

Within each Android app module, files are shown in the following groups:

manifests

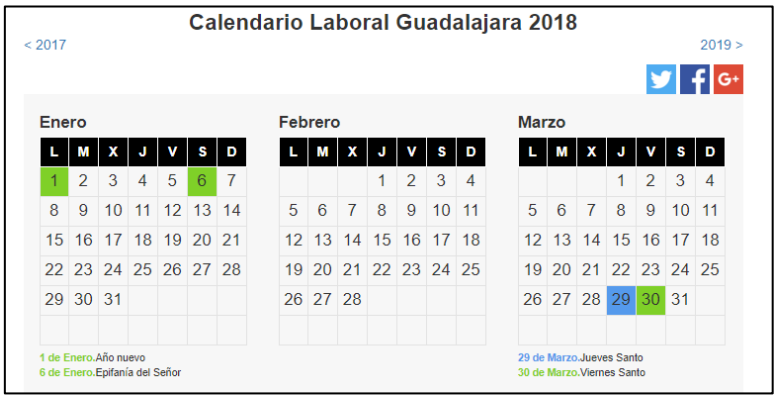
Contains the `AndroidManifest.xml` file.

java

Contains the Java source code files, separated by package names, including JUnit test code.

Días festivos en Guadalajara (España)

Obtenido de http://www.calendarioslaborales.com/calendario-laboral-guadalajara-2018.htm



Calendario Laboral Guadalajara 2018

< 2017 2019 >

Enero

L	M	X	J	V	S	D
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

Febrero

L	M	X	J	V	S	D
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28				

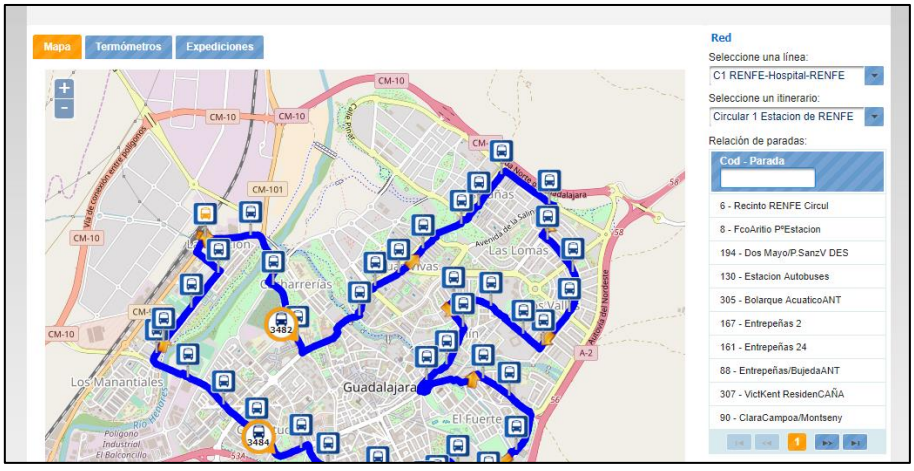
Marzo

L	M	X	J	V	S	D
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

1 de Enero.Año nuevo
6 de Enero.Epifanía del Señor
29 de Marzo.Jueves Santo
30 de Marzo.Viernes Santo

Web con información sobre los buses activos

Obtenido de http://urbanos.guadalajara.es/localizacion/localizacion.xhtml



16. Anexos

Anexo 1: Tabla de intervalos

Carácter	Significado
H	Hoy (el día de la consulta)
F	Festivo
L	Lunes
M	Martes
X	Miércoles
J	Jueves
V	Viernes
S	Sábado
D	Domingo

Anexo 2: Tabla de tipos de coordenadas

"Tipo" de coordenada	**Significado**
0	Coordenada de un punto en la ruta.
1	Coordenada de una parada intermedia de un itinerario.
3	Coordenada de la parada de inicio o final de itinerario.

Anexo 3: Estructura de la respuesta de la función "Información de la posición de un bus activo"

```
{
 "type": "object",
 "properties": {
  "estado": {
   "type": "string"
   "values": {"DATOS", "NO DATOS"}
  },
  "vehiculo": {
   "type": "object",
   "properties": {
    "id": {
     "type": "string"
    },
    "cod": {
     "type": "string"
    },
    "idEmpresa": {
     "type": "string"
    },
    "descEmpresa": {
     "type": "string"
    },
    "pos": {
     "type": "object",
     "properties": {
```

```
        "lon": {
          "type": "number"
        },
        "lat": {
          "type": "number"
        }
      }
    }
  }
},
"idParadaActual": {
  "type": "string"
},
"idItinerarioActual": {
  "type": "string"
},
"itinerarios": {
  "type": "array",
  "items": [
    {
      "type": "object",
      "properties": {
        "id": {
          "type": "string"
        },
        "cod": {
          "type": "string"
        },
        "destino": {
          "type": "null"
        },
        "nombre": {
          "type": "null"
        },
        "linea": {
          "type": "object",
          "properties": {
            "id": {
              "type": "string"
            },
            "cod": {
              "type": "string"
            },
            "empresa": {
              "type": "null"
            },
            "nombre": {
              "type": "null"
            },
            "color": {
              "type": "integer"
            }
          }
        },
        "actual": {
          "type": "boolean"
```

```
            },
            "puntos": {
              "type": "array",
              "items": [
                {
                  "type": "object",
                  "properties": {
                    "pos": {
                      "type": "object",
                      "properties": {
                        "lon": {
                          "type": "number"
                        },
                        "lat": {
                          "type": "number"
                        }
                      }
                    },
                    "tipo": {
                      "type": "integer"
                    },
                    "parada": {
                      "type": "object",
                      "properties": {
                        "id": {
                          "type": "string"
                        },
                        "cod": {
                          "type": "string"
                        },
                        "desc": {
                          "type": "string"
                        },
                        "ordenParada": {
                          "type": "integer"
                        }
                      }
                    },
                    "ordenPunto": {
                      "type": "integer"
                    }
                  }
                },...
              ]
            }
          }
        },...
      ]
    }
  }
}
```

Anexo 4: Estructura de la respuesta de la función "Líneas que transcurren por una parada"

```
{
 "type": "object",
 "properties": {
  "estado": {
    "type": "string"
    "values": {"DATOS", "NO LINEAS", "ERROR"}
  },
  "lista": {
    "type": "array",
    "items": [
      {
       "type": "object",
       "properties": {
        "id": {
          "type": "string"
        },
        "cod": {
          "type": "string"
        },
        "empresa": {
          "type": "string"
        },
        "nombre": {
          "type": "string"
        },
        "color": {
          "type": "integer"
        }
       }
      },...
    ]
  }
 }
}
```

Anexo 5: Estructura de la respuesta de la función "Tiempo de llegada a una parada"

```
{
 "type": "object",
 "properties": {
  "estado": {
    "type": "string"
    "values": {"DATOS", "SIN_SERVICIO", "NO DESC" , "ERROR"}
  },
  "parada": {
    "type": "object",
    "properties": {
     "id": {
       "type": "string"
```

```
      },
      "cod": {
        "type": "string"
      },
      "desc": {
        "type": "string"
      },
      "ordenParada": {
        "type": "integer"
      }
    }
  },
  "tiempos": {
    "type": "array",
    "items": [
      {
        "type": "object",
        "properties": {
          "itinerario": {
            "type": "object",
            "properties": {
              "id": {
                "type": "string"
              },
              "cod": {
                "type": "string"
              },
              "destino": {
                "type": "null"
              },
              "nombre": {
                "type": "string"
              },
              "linea": {
                "type": "object",
                "properties": {
                  "id": {
                    "type": "string"
                  },
                  "cod": {
                    "type": "string"
                  },
                  "empresa": {
                    "type": "null"
                  },
                  "nombre": {
                    "type": "string"
                  },
                  "color": {
                    "type": "integer"
                  }
                }
              },
              "actual": {
                "type": "boolean"
              },
```

```
            "puntos": {
              "type": "null"
            }
          }
        },
        "creacion": {
          "type": "string"
        },
        "minutos": {
          "type": "integer"
        },
        "cabecera": {
          "type": "boolean"
        },
        "tipo": {
          "type": "integer"
        }
      }
    },...
   ]
  }
 }
}
```

Anexo 6: Estructura de la respuesta de la función "Expediciones de itinerario"

```
{
 "type": "object",
 "properties": {
  "estado": {
   "type": "string"
   "values": {"DATOS", "NO EXPEDICIONES ITI" , "ERROR"}
  },
  "itinerario": {
   "type": "object",
   "properties": {
    "id": {
     "type": "string"
    },
    "cod": {
     "type": "string"
    },
    "destino": {
     "type": "string"
    },
    "nombre": {
     "type": "null"
    },
    "linea": {
     "type": "object",
     "properties": {
      "id": {
       "type": "string"
      },
```

```
          "cod": {
            "type": "string"
          },
          "empresa": {
            "type": "null"
          },
          "nombre": {
            "type": "null"
          },
          "color": {
            "type": "integer"
          }
        }
      },
      "actual": {
        "type": "boolean"
      },
      "puntos": {
        "type": "null"
      }
    }
  },
  "horas": {
    "type": "array",
    "items": [
      {
        "type": "string"
      },...
    ]
  }
 }
}
```

Anexo 7: Estructura de la respuesta de la función "Itinerarios de línea"

```
{
 "type": "object",
 "properties": {
   "estado": {
     "type": "string"
     "values": {"DATOS", "NO ITINERARIOS" , "ERROR"}
   },
   "linea": {
     "type": "object",
     "properties": {
       "id": {
         "type": "string"
       },
       "cod": {
         "type": "string"
       },
       "empresa": {
         "type": "null"
       },
```

```
      "nombre": {
        "type": "null"
      },
      "color": {
        "type": "integer"
      }
    }
  },
  "lista": {
    "type": "array",
    "items": [
      {
        "type": "object",
        "properties": {
          "id": {
            "type": "string"
          },
          "cod": {
            "type": "string"
          },
          "destino": {
            "type": "string"
          },
          "nombre": {
            "type": "string"
          },
          "linea": {
            "type": "object",
            "properties": {
              "id": {
                "type": "string"
              },
              "cod": {
                "type": "string"
              },
              "empresa": {
                "type": "string"
              },
              "nombre": {
                "type": "string"
              },
              "color": {
                "type": "integer"
              }
            }
          },
          "actual": {
            "type": "boolean"
          },
          "puntos": {
            "type": "array"
          }
        }
      },...
    ]
  }
```

```
  }
}
```

Universidad de Alcalá

Escuela Politécnica Superior

ESCUELA POLITECNICA SUPERIOR