Desarrollo en dispositivos móviles

“Aplicación accesible de guiado para invidentes en el transporte público”

Development on mobile devices

Iradiel García Pérez
Ingeniería Informática
Escuela Técnica Superior de Ingeniería Informática

Trabajo de Fin de Grado

La Laguna, 09 de julio de 2014

D. **Alejandro Pérez Nava**, con N.I.F. 43.821.179-S profesor Asociado de Universidad adscrito al Departamento de Ingeniería Informática de la Universidad de La Laguna y

D. **Fernando Pérez Nava**, con N.I.F. 42.091.420-V profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática de la Universidad de La Laguna

C E R T I F I C A N

Que la presente memoria titulada:

"Desarrollo en dispositivos móviles. Aplicación accesible de guiado para invidentes en el transporte público"

ha sido realizada bajo su dirección por D. **Iradiel García Pérez**, con N.I.F. 78.629.508-Y.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 9 de julio de 2014

Agradecimientos

Quiero agradecer a toda mi familia, en especial a mis padres y hermana, por saber aguantarme en estos meses y apoyarme durante el transcurso de toda la carrera.

También agradecer a mis compañeros, una nueva familia en la que apoyarte dentro de lo que se ha convertido estos años como en una segunda casa, la facultad, y que espero permanecerán siendo una inestimable amistad en el futuro.

Agradecer a todos los profesores que me han instruido para ayudarme a convertirme en un ingeniero, por sus enseñanzas y sus consejos invaluables. Y sobre todo agradecer a mi director de proyecto por saber guiarme a lo largo de estos meses, dejándome libertad para elegir en ocasiones y estando pendiente de mí recordándome que el tiempo pasaba si me retrasaba en algún plazo.

Y por último, agradecer a la directiva por el gran trabajo realizado en el proceso de cambio del antiguo plan al actual.

Resumen

El objetivo de este trabajo ha sido desarrollar una aplicación móvil accesible para el sistema operativo Android que, por medio de voz, avise a usuarios invidentes cual será la próxima parada de una línea de autobús. El trabajo de la aplicación consiste en descargar los datos de las líneas de cualquier compañía del mundo que tenga disponibles sus datos en GoogleTransit y por medio de la geolocalización estimar en qué punto de la ruta se encuentra el usuario y ofrecer un mensaje de voz para informarle tanto de la próxima parada como de alguna información extra del tipo informativo sobre dicha parada. Para la realización del proyecto se han utilizado como ejemplo los datos de TITSA para mostrar el funcionamiento de la aplicación.

Palabras Clave

Android, Accesibilidad, Geolocalización, GoogleTransit, Paradas de autobús

Abstract

The objective of this work has been to develop an accessible Android application. This application must notify blind users the next bus stop in their route using voice notification. The work of the app is to download data trips from any agency in the world that has them available in GoogleTransit. Then, through geolocation, it estimates the point of the route where the user is and voice-reports the next stop with some extra information about it. For the presentation of the project, data from TITSA has been used as an example to show how the application works.

Keywords

Android, Accessibility, Geolocation, GoogleTransit, Bus stop

Índice

Capítulo 1. Análisis ... 1
1.1 Introducción ... 1
1.2 Antecedentes ... 2
1.3 Requisitos ... 3
1.3.1 Requisitos funcionales ... 3
1.3.2 Requisitos no funcionales ... 4
Capítulo 2. Phonegap versus Aplicaciones nativas ... 5
2.1 Introducción ... 5
2.2 Creación del proyecto ... 5
2.3 Plugins y librerías utilizados ... 8
2.4 Problemas con Phonegap ... 9
Capítulo 3. Aplicación servidor ... 10
3.1 Introducción ... 10
3.2 Obtención de los datos ... 11
3.3 Creación de la Base de Datos ... 14
3.4 Conversión de los datos ... 16
3.4.1 Insertar datos de compañía, rutas y paradas ... 16
3.4.2 Insertar datos de trips ... 17
Capítulo 4. Conexión de la aplicación móvil con el servidor ... 20
4.1 Introducción ... 20
4.2 REST vs SOAP ... 21
4.3 REST API ... 22
4.4 Implementación en el servidor ... 26
Capítulo 5. Base de datos y servicio para actualizarla ... 28
5.1 Introducción ... 28

5.2 Almacenamiento de los datos en el móvil ... 28
5.3 Tareas asíncronas ... 32
5.4 Servicio de actualización periódica ... 35
Capítulo 6. Geolocalización y proximidad ... 38
6.1 Introducción ... 38
6.2 Obtener la ubicación del dispositivo ... 38
6.3 Seguimiento de una ubicación ... 41
Capítulo 7. Desarrollo accesible en Android ... 43
7.1 Introducción ... 43
7.2 Pautas para un desarrollo accesible ... 43
7.3 Diseño de la aplicación ... 48
Capítulo 8. Portal web ... 53
8.1 Introducción ... 53
8.2 Herramientas utilizadas ... 54
8.3 Diseño del portal ... 55
Capítulo 9. Conclusiones y Trabajos Futuros ... 58
9.1 Conclusiones ... 58
9.2 Trabajos futuros ... 59
Capítulo 10. Summary and Conclusions ... 60
10.1 Conclusions ... 60
10.1 Future works ... 61
Capítulo 11. Presupuesto ... 62
11.1 Horas dedicadas ... 62
11.2 Herramientas utilizadas ... 62
11.3 Equipo necesario ... 63
11.4 Total ... 63
Bibliografía ... 64

Índice de figuras

Ilustración 1. Esquema de arquitectura del proyecto 1
Ilustración 2 - Estructura de un proyecto phonegap en eclipse 6
Ilustración 3. Primera aplicación con Phonegap 7
Ilustración 4. Soporte de pantallas Android 7
Ilustración 5. Permisos de uso en Android 8
Ilustración 6. Android configChanges 8
Ilustración 7. Esquema de arquitectura dl servidor 10
Ilustración 8. Estructura de un fichero GoogleTransit 11
Ilustración 9. Esquema relacional de GoogleTransit 14
Ilustración 10. Esquema relacional de la Base de Datos del proyecto 15
Ilustración 11. Código PHP para descargar zip de TITSA de GoogleTransit 16
Ilustración 12. Código PHP para descomprimir archivos 16
Ilustración 13. Código PHP para obtener los datos de un fichero txt 17
Ilustración 14. Código PHP para obtener los datos de los trips 18
Ilustración 15. Código PHP para comprobar si un trip ya está almacenado 18
Ilustración 16. Código PHP para recuperar datos de stop_times.txt 19
Ilustración 17. Código PHP para relacionar los ficheros trips y stop_times 19
Ilustración 18. Esquema de conexión de la aplicación con el servidor 20
Ilustración 19. Niveles para conseguir REST según el modelo Richardson Maturity Model 22
Ilustración 20. Resultado de una petición sin hipermedia 25
Ilustración 21. Resultado de una petición con hipermedia incorrecta 25
Ilustración 22. Resultado de una petición con hipermedia correcta 26
Ilustración 23. Instalación del framework Slim 26
Ilustración 24. Instanciar y ejecutar clase Slim 27
Ilustración 25. Definir rutas con Slim 27
Ilustración 26. Consultas para crear las tablas de la BD 29
Ilustración 27. Clase SQLiteOpenHelper 30
Ilustración 28. Apertura y cierre de la BD 31
Ilustración 29. Inserción y consulta de datos de rutas 31
Ilustración 30. Creación de un método que se ejecuta en un hilo aparte 32
Ilustración 31. Método doInBackground de la clase AsyncTask 33
Ilustración 32. Método onPostExecute de la clase AsyncTask 34

Ilustración 33. Creación del servicio 35
Ilustración 34. Realizar tareas de forma repetida 36
Ilustración 35. Obtener ubicación del dispositivo 39
Ilustración 36. Clase LocationListener 39
Ilustración 37. Método requestLocationUpdates( ) 40
Ilustración 38. Añadir alerta de proximidad 41
Ilustración 39. Implementar el receptor de la alerta de proximidad 42
Ilustración 40. Eliminar alerta de proximidad 42
Ilustración 41. Atributo contentDescription 45
Ilustración 42. Atributo focusable 45
Ilustración 43. Teclado direccional de Eye-Free Keyboard 46
Ilustración 44. Habilitar TalkBack 46
Ilustración 45. Habilitar exploración táctil 47
Ilustración 46. Tamaño accesible de los controles 47
Ilustración 47. Notificación mediante pDialog 48
Ilustración 48. Notificación mediante Toast 48
Ilustración 49. Menú de la aplicación 49
Ilustración 50. Mapa navegacional del menú de la aplicación 49
Ilustración 51. Mapa navegacional ejemplo de uso de la aplicación 50
Ilustración 52. Mapa navegacional general de la aplicación 51
Ilustración 53. Icono de la aplicación 52
Ilustración 54. Esquema de conexión del portal web con el servidor 53
Ilustración 55. Esquema de la base de datos con usuarios 56
Ilustración 56. Portal web. Inicio/Login 57
Ilustración 57. Portal web. Vista de los datos 57

Índice de tablas

Tabla 1. Ejemplo de fichero agency.txt 11
Tabla 2. Ejemplo de fichero calendar.txt 12
Tabla 3. Ejemplo de fichero calendar_dates.txt 12
Tabla 4. Ejemplo de fichero routes.txt 12
Tabla 5. Ejemplo de fichero stop_times.txt 13
Tabla 6. Ejemplo de fichero stops.txt 13
Tabla 7. Ejemplo de fichero trips.txt 13
Tabla 8. Presupuesto por horas dedicadas 62
Tabla 9. Presupuesto por licencias de frameworks y librerías 62
Tabla 10. Presupuesto por licencias de software 63
Tabla 11. Presupuesto por equipo utilizado 63
Tabla 12. Presupuesto total 63

Capítulo 1. Análisis

1.1 Introducción

En este capítulo veremos de forma más extendida los detalles de la aplicación a desarrollar. Empezaremos con una búsqueda de antecedentes de aplicaciones similares en el mercado y luego hablaremos sobre los requisitos que ha de tener nuestra aplicación.

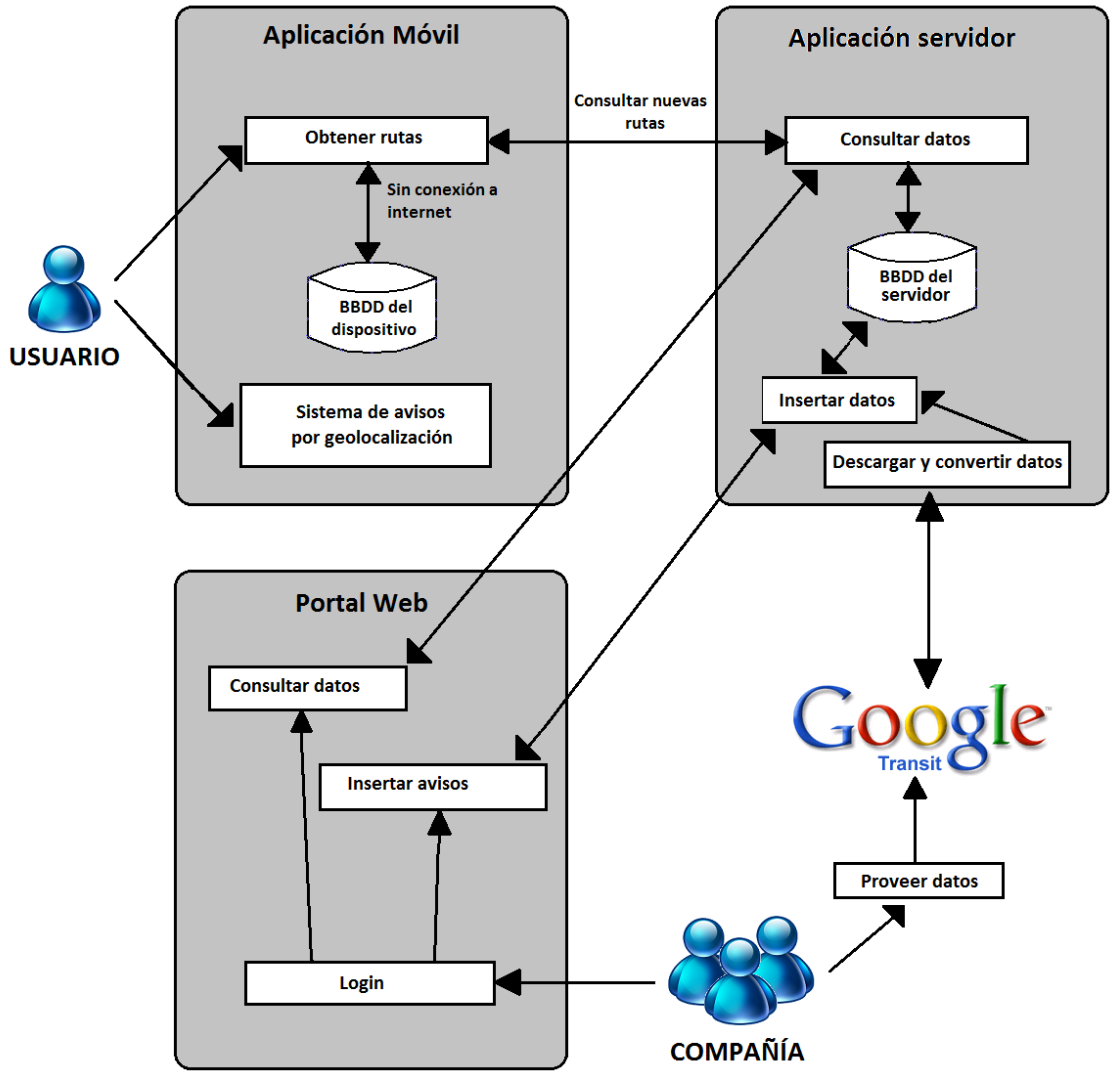


Ilustración 1. Esquema de arquitectura del proyecto

1.2 Antecedentes

Realizando una búsqueda de aplicaciones o servicios similares en la red nos encontramos con aplicaciones como OnTheBus y TorBus.

TorBus es una aplicación desarrollada por un equipo científico de la Universidad de Almería. Utilizando la geolocalización y el sistema VoiceOver la app ofrece al usuario invidente funcionalidades tales como "¿Dónde estoy?" que indica la posición exacta, "Todas las paradas", que señala la distancia en metros de las distintas paradas del autobús, "Todas las líneas", con las diversas líneas de autobús operativas tanto de ida como de vuelta, "Modo BUS" te informa del origen, destino, línea, paradas y distancia hasta la parada final, "Guíame", que utiliza la brújula del iPhone para dirigir a la persona hasta la parada de bus más próxima, e "Ir a casa" una vez establecida nuestra casa en el mapa el smartphone nos ayudará a llegar a casa dándonos indicaciones. De momento la app solo está para el sistema operativo IOS y está disponible a nivel local.

OnTheBus es una aplicación móvil desarrollada por MASS Factory una Spin-Off del grupo GABiTAP de la Escuela de Ingeniería de la Universitat Autònoma de Barcelona. Se trata de una aplicación para dispositivos móviles con sistema operativo Android que, al igual que TorBus ofrece una amplia variedad de servicios. Dado un destino, la aplicación ofrece un conjunto de rutas óptimas a elegir. Al seleccionar una de ellas, la aplicación guía al usuario desde el punto donde se encuentre hasta la parada de autobús. En este punto el sistema informa al usuario del tiempo que queda para que llegue el trasporte. Una vez en el autobús, se recibe información de las paradas por las que se va pasando y avisa cuándo se debe pulsar el timbre para apearse en la siguiente parada. Cuando el usuario ha bajado, se le guía hasta el punto de destino. En caso de no necesitar autobús, se realizaría el guiado a pie. La app también cuenta con cuatro perfiles accesibles "Limitación Visual", "Limitación Cognitiva", "Limitación Auditiva" y "Limitación de Movilidad". De momento la aplicación solo está disponible para algunas compañías en Barcelona, Madrid, Laval y Vitoria-Gasteiz.

1.3 Requisitos

Dividiremos los requisitos del proyecto en funcionales y no funcionales. Para ello hay que tener en cuenta que el proyecto está, a su vez, dividido en tres partes; por un lado necesitaremos una aplicación servidor que se encargará de descargar los datos necesarios para la aplicación móvil, luego será necesario un portal web en el que las compañías de transporte puedan modificar notificaciones de alerta o información sobre las rutas o paradas propias y por último la propia aplicación móvil.

1.3.1 Requisitos funcionales

Aplicación servidor:

- Tendrá que ser capaz de acceder a los datos de una empresa alojados en GoogleTransit y descargarlos.
- Deberá convertir los datos al formato necesario y almacenarlos en la base de datos propia.
- Deberá facilitar el acceso a los datos a la aplicación móvil.

Portal web:

- Gestión de usuarios.
- Posibilidad de añadir información sobre rutas o paradas.
- Conexión con la app servidor.

Aplicación móvil:

- Poder acceder a los datos de las compañías y rutas facilitados por la app servidor y almacenarlos en la base de datos del dispositivo.
- Servicio para actualizar datos de forma periódica.
- Obtener la localización actual del usuario.

- Servicio para obtener cambios en la posición del usuario y notificar de la proximidad a una localización.
- Conexión con la aplicación del servidor.

1.3.2 Requisitos no funcionales

Aplicación servidor:

- Disponibilidad: deberá estar disponible las 24h.
- Seguridad: deberá tener implementados métodos de seguridad para evitar acciones malintencionadas como la modificación de los datos.
- Escalabilidad: el servidor deberá estar capacitado para soportar grandes cantidades de información.

Portal web:

- Seguridad: deberá contar con seguridad puesto que se tendrá que gestionar usuarios.

Aplicación móvil:

- Accesibilidad: deberá tener una interfaz accesible para el perfil de usuario que nos interesa, los usuarios invidentes.
- Usabilidad: Hay que tener en cuenta que los usuarios para los que está diseñada la aplicación cuentan con una dificultad adicional, por lo que deberá ser lo más sencilla posible.
- Disponible en Android.
- Permita ser utilizado correctamente con TalkBack.

Capítulo 2. Phonegap versus Aplicaciones nativas

2.1 Introducción

Phonegap (Adobe-Systems) es un framework para el desarrollo de aplicaciones móviles que permite a los programadores diseñar las aplicaciones utilizando lenguajes de programación como HTML5, JavaScript y CSS3. El resultado son aplicaciones híbridas, es decir que no son nativas del dispositivo pero tampoco son aplicaciones web, que se renderizan mediante vistas web en lugar de con interfaces gráficas específicas del dispositivo.

Como veremos más adelante, tratar las aplicaciones como vistas web dará problemas a la herramienta TalkBack de Android para acceder al contenido web de estas vistas y poder reproducirlo correctamente.

Para comenzar la creación de la aplicación es necesario instalar el software necesario para poder desarrollar la misma. El entorno de desarrollo elegido es eclipse, utilizando Android SDK y Android Virtual Device. Una vez tengamos configurado eclipse con los plugins necesarios para desarrollar aplicaciones android, descargamos el software de phonegap.

2.2 Creación del proyecto

En el eclipse, creamos un nuevo proyecto de aplicación android (Alonsojpd). Para habilitar phonegap en nuestro proyecto debemos realizar los siguientes pasos:

*Nota: Los archivos que se mencionan a continuación y que deben ser alojados dentro de nuestro proyecto deben estar comprimidos en el zip de phonegap que se ha descargado con anterioridad.

- En la carpeta “assets” de nuestro proyecto creamos un nuevo directorio de nombre “www”.
- Dentro de dicho directorio debemos copiar el archivo “cordova.js”.

- Copiamos el archivo “cordova-{{versión}}.jar” en el directorio libs de nuestro proyecto.
- Copiamos el directorio “xml” del zip de phonegap dentro del directorio “res” de nuestro proyecto.

La estructura de nuestro proyecto debe quedar así:

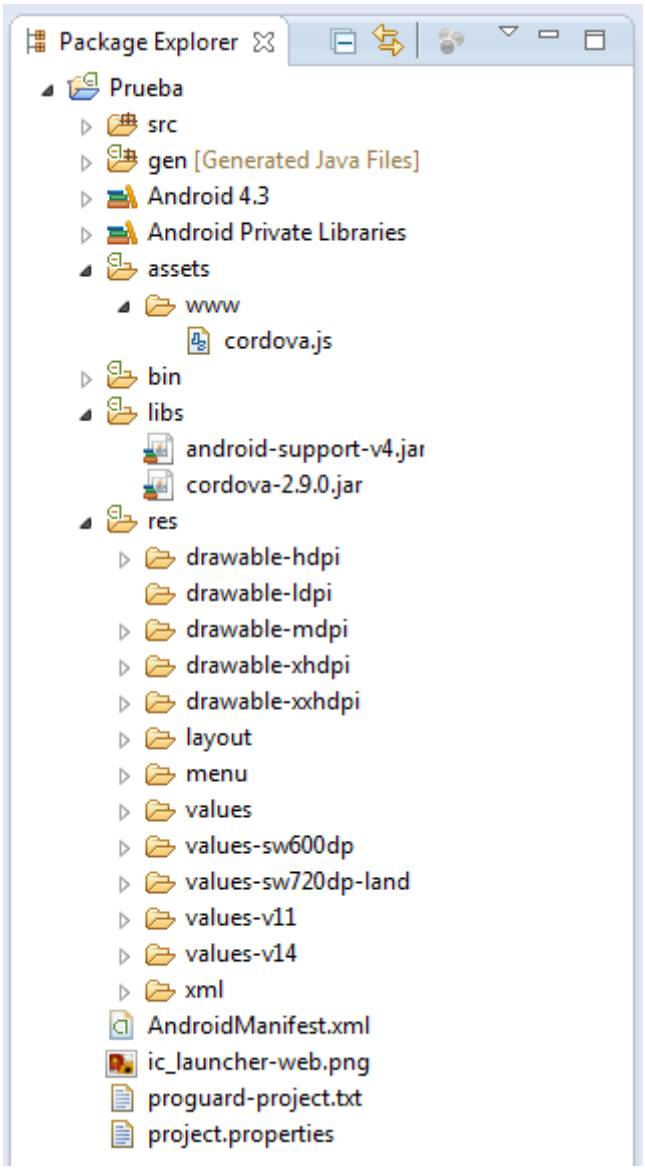


Ilustración 2 - Estructura de un proyecto phonegap en eclipse

Ahora creamos un fichero HTML dentro del directorio “www” y le damos de nombre index (será nuestra pantalla inicial de la app).

Modificamos el fichero “MainActivity.java”:

- Importamos la librería de cordova:

- import org.apache.cordova.*;

- Cambiamos la clase de la que extiende MainActivity por “DroidGap”.
- Eliminamos el segundo método (onCreateOptionsMenu)
- Cambiamos la línea “setContentView…” por esta otra “super.loadUrl(Config.getStartUrl());” para que detecte y cargue el index.html como principal de la app.
- Cambiamos el método “protected” por “public”.

El fichero deberá quedar de la siguiente forma:

```
package com.example.prueba;

import android.os.Bundle;
import org.apache.cordova.*;

public class MainActivity extends DroidGap {

  @Override
  public void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     super.loadUrl(Config.getStartUrl());
  }

}
```

Ilustración 3. Primera aplicación con Phonegap

Modificamos como texto el fichero “AndroidManifest.xml”:

- Añadimos soporte de pantallas:

```
<supports-screens
        android:largeScreens="true"
        android:normalScreens="true"
        android:smallScreens="true"
        android:xlargeScreens="true"
        android:resizeable="true"
        android:anyDensity="true" />
```

Ilustración 4. Soporte de pantallas Android

- Añadimos permisos de uso:

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.INTERNET" />
```

Ilustración 5. Permisos de uso en Android

- Añadimos la línea configChanges:

```
<activity
        android:name="com.example.prueba.MainActivity"
        android:label="@string/app_name" >
    android:configChanges="orientation|keyboardHidden|keyboard|screenSize|locale">

        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
```

Ilustración 6. Android configChanges

Guardamos todo y ya podemos ejecutar el proyecto.

2.3 Plugins y librerías utilizados

Durante el desarrollo del proyecto en Phonegap se utilizaron diferentes librerías y plugins para facilitar algunas actividades dentro de la aplicación. Recogeremos en este apartado el uso que se dio de cada una de ellas.

- App Framework, JQuery mobile (Intel): es una librería construida para móviles que permite hacer consultas de los selectores de forma rápida.
- PGSQLite (Brody): librería utilizada para permitir la gestión de bases de datos SQLite en Phonegap.

- TTS Plugin, Text to Speech (Macdonst): este plugin permite la conversión de texto en voz, necesario para las notificaciones a los usuarios.

2.4 Problemas con Phonegap

Una vez estudiados los plugins a utilizar en la aplicación y realizado un pequeño prototipo, probamos que funcionase correctamente la herramienta TalkBack de Android con nuestra aplicación.

El resultado fue que TalkBack solamente detectaba contenido web en la interfaz de la aplicación y no los elementos que contenía la aplicación. La herramienta TalkBack no estaba preparada para aplicaciones diseñadas mediante Phonegap en su versión más reciente (PhoneGap discussion group).

Después de analizar el problema, se nos plantearon dos alternativas, migrar todo el proyecto a una versión de Phonegap más antigua o empezar de cero utilizando el lenguaje nativo de Android. Se descartó la primera opción puesto que no se aseguraba que fuera a funcionar TalkBack y para evitar otros posibles errores con la comunicación entre esta herramienta y la aplicación.

Capítulo 3. Aplicación servidor

3.1 Introducción

Como hemos comentado con anterioridad, es importante que una parte del proyecto se encargue de descargar los datos de las líneas de las compañías y los mantenga en una base de datos propia para que luego puedan ser consultados por el usuario con sencillez.

Los datos que nos provee GoogleTransit (Google Inc) contienen información sobre los horarios de los transportes, hora de llegada a cada parada, tipos de rutas (días laborales, festivos,...). El uso que le vamos a dar a nuestra aplicación no necesita toda esta sobrecarga de datos puesto que nuestro sistema se basará en la geolocalización del usuario para avisar de la proximidad a las paradas. Veremos en este capítulo el trabajo realizado por la aplicación servidor, desarrollada en PHP, para dejar estos datos accesibles a los usuarios.

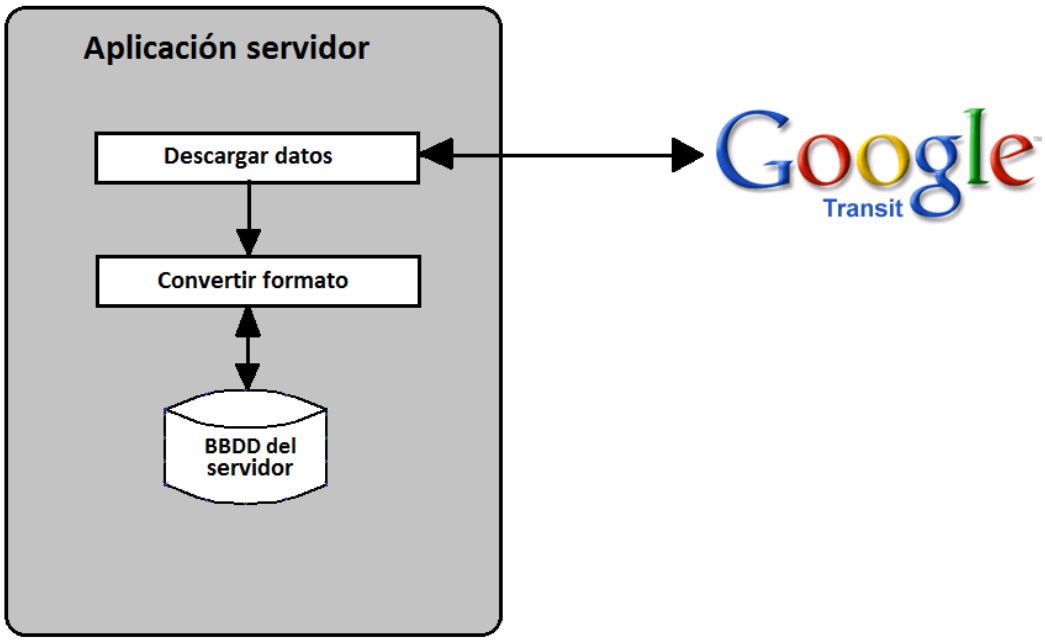
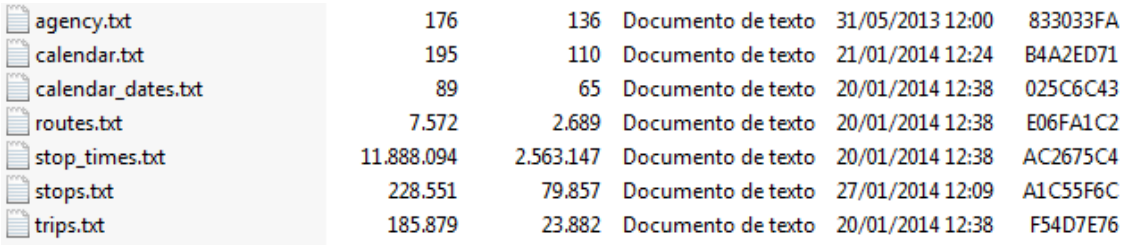


Ilustración 7. Esquema de arquitectura dl servidor

3.2 Obtención de los datos

Para obtener los datos, GoogleTransit nos facilita un link (Google Inc) en el que se encuentran los feeds de muchas compañías que utilizan el servicio. La aplicación servidor recibe este feed, que es una url que descargará el zip con los datos de la compañía elegida y comenzará el proceso de almacenado de datos en la base de datos.



agency.txt	176	136	Documento de texto	31/05/2013 12:00	833033FA
calendar.txt	195	110	Documento de texto	21/01/2014 12:24	B4A2ED71
calendar_dates.txt	89	65	Documento de texto	20/01/2014 12:38	025C6C43
routes.txt	7.572	2.689	Documento de texto	20/01/2014 12:38	E06FA1C2
stop_times.txt	11.888.094	2.563.147	Documento de texto	20/01/2014 12:38	AC2675C4
stops.txt	228.551	79.857	Documento de texto	27/01/2014 12:09	A1C55F6C
trips.txt	185.879	23.882	Documento de texto	20/01/2014 12:38	F54D7E76

Ilustración 8. Estructura de un fichero GoogleTransit

Para entender las transformaciones necesarias para eliminar los datos que no son relevantes en nuestro proyecto se mostrarán tablas de datos de ejemplo que contendría cada uno de los archivos txt que contiene el zip de GoogleTransit.

El fichero agency.txt contiene información sobre la compañía de transporte que proporciona los datos.

agency_id	agency_name	agency_url	agency_timezone	agency_lang	agency_phone
TITSA	Transportes Interurbanos de Tenerife S.L.	http://www.titsa.com	Atlantic/Canary	Es	+34922531300

Tabla 1. Ejemplo de fichero agency.txt

En calendar.txt encontramos los diferentes tipos de servicio, indicando que días se llevan a cabo. En este ejemplo vemos que hay tres tipos de servicio: uno para los días laborales, otro los sábados y otro los domingos.

service_id	mo	tu	we	th	fri	sa	su	start_date	end_date
1	1	1	1	1	1	0	0	20130701	20130930
2	0	0	0	0	0	1	0	20130701	20130930
3	0	0	0	0	0	0	1	20130701	20130930

Tabla 2. Ejemplo de fichero calendar.txt[1]

Aunque haya distinción de servicios, hay días festivos que no se realizará el servicio que le corresponde según la tabla anterior y que no se tienen en cuenta en esta. Por eso es necesario el siguiente fichero, para indicar estos días y que servicio se realizará.

service_id	date	exception_type
1	20130106	2
1	20130212	2
1	20130221	2

Tabla 3. Ejemplo de fichero calendar_dates.txt

En routes.txt obtenemos los datos de las rutas. Es importante distinguir entre la ruta y el viaje o "trip" que se realiza varias veces al día y diariamente de una misma ruta. Para explicarlo mejor pondremos un ejemplo, en esta compañía la ruta 012 va desde La Laguna hasta El Sauzal pasando por una serie de paradas concretas, pero esta ruta se realiza alrededor de 20 veces en un mismo día y cada uno de estos viajes o "trips" también está registrado en los datos que proporciona GoogleTransit.

route_id	route_short_name	route_long_name	route_type
11	011	LA LAGUNA-GUAMASA-NARANJEROS-CALVARIO-S CATALINA-SAUZAL	3
12	012	LA LAGUNA-NARANJEROS-TACORONTE-EMP. EL SAUZAL-EL SAUZAL	3
14	014	S/C--LA CUESTA-LA LAGUNA	3

Tabla 4. Ejemplo de fichero routes.txt

El fichero stop_times.txt contiene los horarios en los que un viaje o "trip" pasa por cada una de las paradas de su ruta y el orden en el que las recorre.

[1] Los encabezados de la tabla mo, tu, we,… hacen referencia a los días de la semana Monday, Tuesday, Wednesday. Se han acortado en la tabla para ahorrar espacio y poder mostrar todos los encabezados en una sola línea.

trip_id	arrival_time	departure_time	stop_id	stop_seq
2308826	07:30:00	07:30:00	2625	0
2308826	07:31:04	07:31:04	2549	1
2308826	07:32:40	07:32:40	1723	2
2308826	07:33:36	07:33:36	1183	3

Tabla 5. Ejemplo de fichero stop_times.txt

En el siguiente fichero se almacena la información sobre cada una de las paradas que tiene registradas la compañía.

stop_id	stop_code	stop_name	stop_lat	stop_long
1100	1100	ACORAN PARADA N 1	28.418338	-16.303946
1101	1101	ACORAN PARADA N 2	28.417573	-16.306626
1102	1102	URB. ACORAN	28.416302	-16.308758
1103	1103	COLEGIO ADONAY	28.415496	-16.310959

Tabla 6. Ejemplo de fichero stops.txt

Por último tenemos el fichero que nos permite relacionar las rutas con cada uno de sus viajes o "trips". Podemos observar por ejemplo que la ruta con id 11 tiene al menos 4 viajes con el tipo de servicio realizado en días laborales y todos en la misma dirección.

route_id	service_id	trip_id	direction_id
11	1	2308826	0
11	1	2308827	0
11	1	2308828	0
11	1	2308829	0

Tabla 7. Ejemplo de fichero trips.txt

Si ampliásemos la información de esta última tabla veríamos que la misma ruta se puede realizar en dos direcciones.

Para comprender mejor la estructura de los datos descargados de GoogleTransit se ha elaborado el siguiente esquema relacional que muestra las conexiones entre tablas.

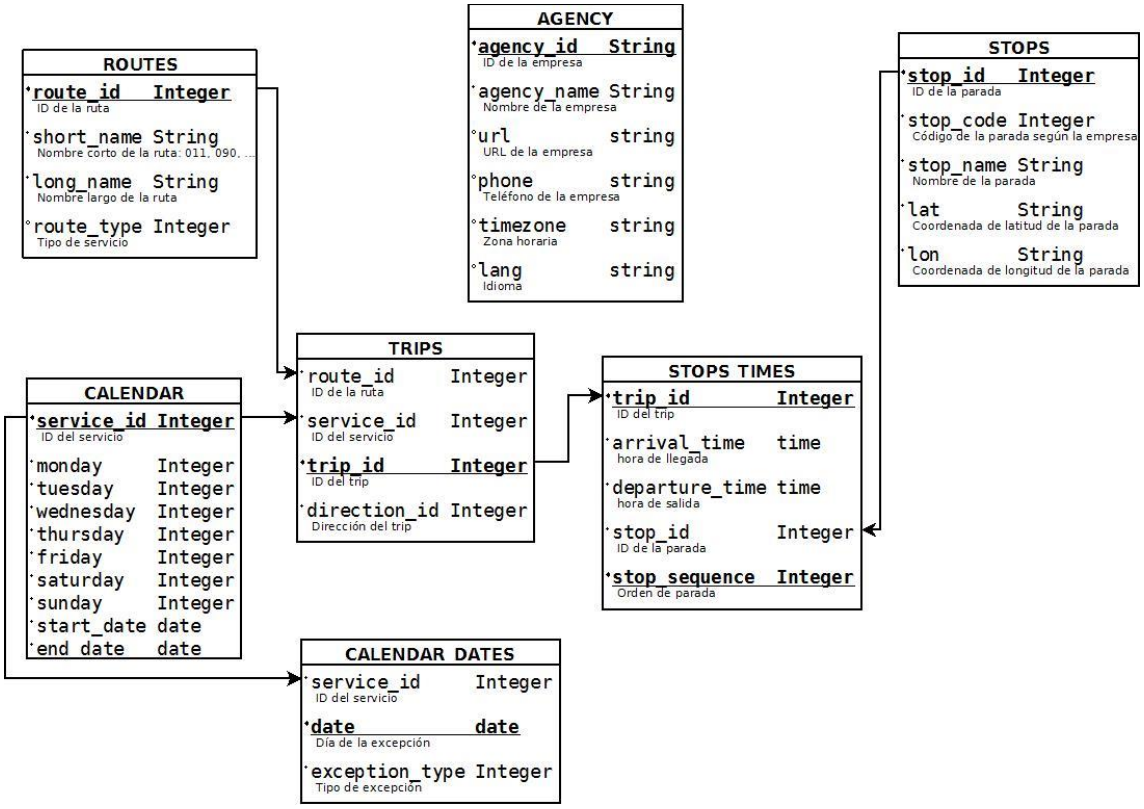


Ilustración 9. Esquema relacional de GoogleTransit

3.3 Creación de la Base de Datos

Como la aplicación que se va a realizar será un sistema de avisos de paradas basado en la posición GPS del usuario no es necesario que mantengamos información con horarios en nuestra base de datos, es por eso que podríamos prescindir de los siguientes datos:

- La tabla calendar diferencia entre tipos de servicios para saber los horarios de los viajes así que no es necesaria.
- Lo mismo ocurre con la tabla calendar_dates, no necesitamos saber las excepciones de los servicios en días festivos.
- En la tabla stop_times podemos obviar los horarios de llegada y salida de cada parada.

También hay que tener en cuenta que nuestra base de datos debe estar diseñada para poder almacenar datos de más de una compañía, por lo que

tendremos que añadir un atributo que nos permita relacionar la tabla agency con las demás.

Una vez eliminados los datos innecesarios, el esquema relacional de nuestra base de datos queda de la siguiente forma.

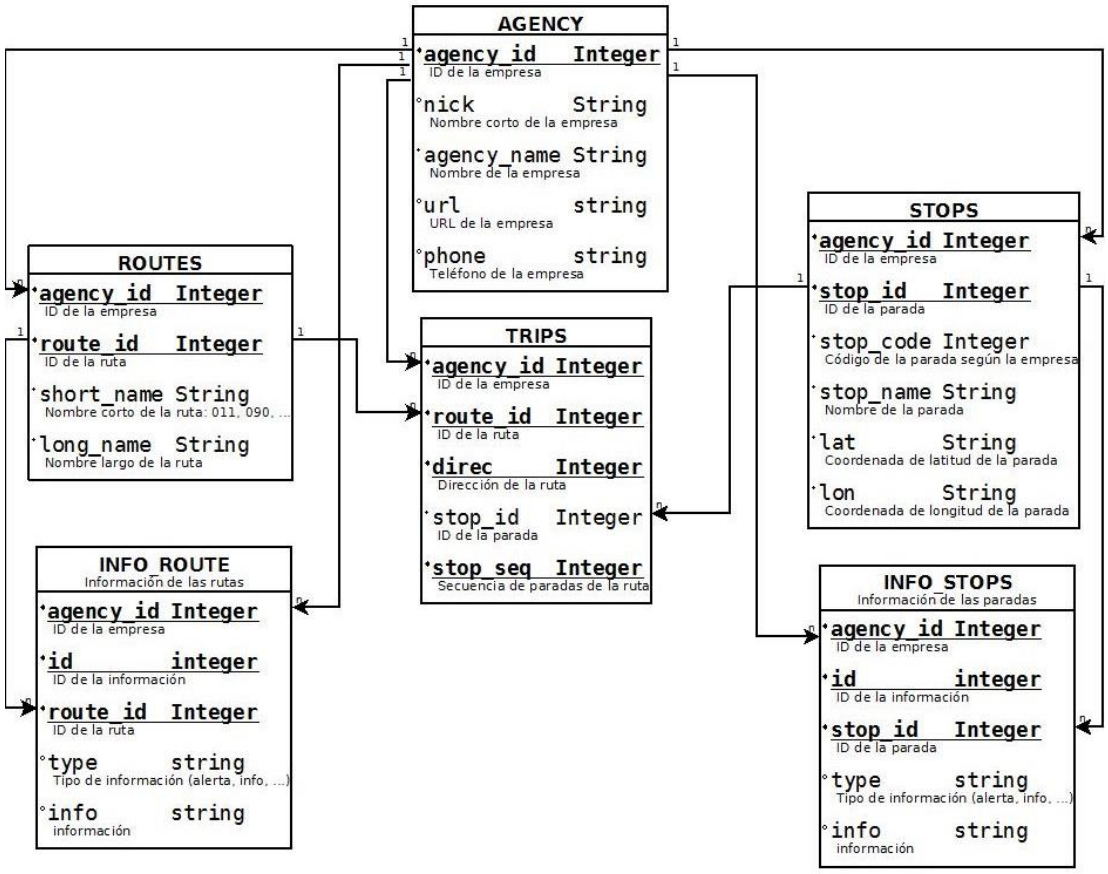


Ilustración 10. Esquema relacional de la Base de Datos del proyecto

También se han añadido dos tablas extra para añadir información adicional sobre las rutas o paradas respectivamente, como por ejemplo avisos por obras, traslado de paradas, etc.

De esta forma tendremos en la tabla agency los datos de todas las compañías de transporte que estén en GoogleTransit, indexados por un ID único. Las rutas y paradas también tendrán su propio ID único para cada compañía, por lo que la clave primaria para estas tablas será ese ID y el de la compañía.

En la tabla trip almacenaremos la secuencia de paradas correspondiente a una ruta. Hay que tener en cuenta que una ruta puede ser recorrida en dos direcciones y no necesariamente deben tener las mismas paradas en los dos

sentidos. Es por eso que la clave primaria en esta tabla es el ID de la compañía, el ID de la ruta, la dirección del viaje y la secuencia de la parada. Se ha tomado como parte de la clave primaria la secuencia de la parada en lugar del ID de la parada porque se puede dar, y de hecho se da en los datos de TITSA, que una ruta pase por una misma parada dos veces en un mismo viaje.

Por último, las tablas de información adicional almacenarán el tipo de la información relacionada a la ruta o parada y el mensaje informativo.

3.4 Conversión de los datos

Lo primero que debe ser capaz de hacer nuestra aplicación servidor es descargar el ZIP con los datos de la compañía, descomprimir los archivos txt contenidos en el ZIP y acceder a los datos.

Para descargar los datos de TITSA desde nuestro servidor PHP, tenemos que proporcionarle la URL donde se encuentra el ZIP con la información.

```
file_put_contents("files/Tmpfile.zip", fopen("http://www.titsa.com/Google_transit.zip", "r"));
$zip = new ZipArchive;
$res = $zip->open('files/Tmpfile.zip');
```

Ilustración 11. Código PHP para descargar zip de TITSA de GoogleTransit

Una vez alojado el archivo en nuestro servidor, descomprimiremos uno a uno los ficheros que contienen los datos de la compañía y almacenaremos la información en nuestra base de datos.

3.4.1 Insertar datos de compañía, rutas y paradas

```
$zip->extractTo('files/', array('agency.txt'));
$gestor = @fopen("files/agency.txt", "r");
```

Ilustración 12. Código PHP para descomprimir archivos

Para leer el fichero txt, tenemos que recuperar la información línea a línea y separar los datos detectando el carácter limitador del fichero, una coma.

```
$datos;
$cabec = array(1 => 'nick', 2 => 'name', 3 => 'url', 4 => 'zone', 5 => 'lang', 6 => 'phone');
while (($bufer = fgets($gestor, 4096)) !== false) {
    if ($cont != 0){
        $ini = 0;
        $fin = strpos($bufer, ',', $ini);
        $num = 1;
        while ($fin !== false){
            $subst = substr($bufer, $ini, $fin - $ini);
            $datos[$cabec[$num]] = $subst;
            $ini = $fin + 1;
            $fin = strpos($bufer, ',', $ini);
            $num++;
        }
        $datos[$cabec[$num]] = substr($bufer, $ini);
```

Ilustración 13. Código PHP para obtener los datos de un fichero txt

Para terminar, hay que almacenar esta información en la base de datos. Esto es tan sencillo como conectar con la base de datos y realizar una consulta SQL.

Recuperar la información de las rutas y paradas no tiene mayor complicación, se utiliza el mismo método que para almacenar los datos de la compañía. El único cambio son las cabeceras de la información que leemos de los ficheros txt y hay que tener en cuenta que debemos almacenar también el ID de la compañía para relacionar las tablas de datos.

3.4.2 Insertar datos de trips

Con el cambio realizado en la base de datos del servidor con respecto a la de GoogleTransit, rescatar la información de los trips se convierte en una tarea más ardua que las anteriores, puesto que tendremos que leer dos ficheros txt para poder obtener todos los datos necesarios antes de introducirlos en la base de datos.

```php
$zip->extractTo('files/', array('trips.txt'));
$gestor = @fopen("files/trips.txt", "r");
$cont = 0;
$datos["agency_id"] = $agency_id;
$all = array();
$cabec = array(1 => 'route_id', 2 => 'service_id', 3 => 'trip_id', 4 => 'direction');
while (($bufer = fgets($gestor, 4096)) !== false) {
    if ($cont != 0){
        $ini = 0;
        $fin = strpos($bufer, ',', $ini);
        $num = 1;
        while ($fin !== false){
            $subst = substr($bufer, $ini, $fin - $ini);
            $datos[$cabec[$num]] = $subst;
            $ini = $fin + 1;
            $fin = strpos($bufer, ',', $ini);
            $num++;
        }
        $subst = substr($bufer, $ini, 1);
        $datos[$cabec[$num]] = $subst;
```

Ilustración 14. Código PHP para obtener los datos de los trips

Hasta aquí todo es igual que en los casos anteriores, leemos línea a línea y vamos recuperando los datos tomando la coma como carácter delimitador al separar la información del string. Sin embargo, recordemos que en GoogleTransit una ruta tiene asociados múltiples trips, cada uno con su ID y sus horarios, y que esto no era relevante para nuestra aplicación. Solo necesitamos un trip en cada dirección para cada una las rutas, el resto de información podemos desecharla.

```php
$enc = false;
foreach ($all as $value) {
    if ($value["route_id"] == $datos["route_id"]){
        if ($value["direction"] == $datos["direction"]){
            $enc = true;
            break;
        }
    }
}
if (!$enc)
    array_push ($all, $datos);
```

Ilustración 15. Código PHP para comprobar si un trip ya está almacenado

Con el código anterior nos aseguramos de que un trip con una dirección determinada ya está recuperado y desechamos los demás que tengan los mismos datos. Puesto que estamos obteniendo la información de un txt, necesitamos leer todas las líneas ya que no sabemos cuáles nos ofrecen información nueva o repetida.

Ya tenemos un trip asociado a una ruta y a una dirección, pero nos falta obtener las paradas de ese trip y el orden en el que se encuentran estas paradas. Para ello necesitamos recuperar esta información de un nuevo fichero, "stop_times.txt".

```
$gestor = @fopen("files/stop_times.txt", "r");
$all2 = array();
$cont = 0;
$cabec = array(1 => 'trip_id', 2 => 'arrival', 3 => 'departure', 4 => 'stop_id', 5 => "stop_seq");
while (($bufer = fgets($gestor, 4096)) !== false) {
    if ($cont != 0){
        $ini = 0;
        $fin = strpos($bufer, ',', $ini);
        $num = 1;
        while ($fin !== false){
            $subst = substr($bufer, $ini, $fin - $ini);
            $datos[$cabec[$num]] = $subst;
            $ini = $fin + 1;
            $fin = strpos($bufer, ',', $ini);
            $num++;
        }
        $subst = substr($bufer, $ini);
        $datos[$cabec[$num]] = $subst;
```

Ilustración 16. Código PHP para recuperar datos de stop_times.txt

Una vez recuperada la información de este fichero hay que relacionarla con la que se había obtenido anteriormente del fichero "trips.txt".

```
foreach ($all as $value) {
    if ($value["trip_id"] == $datos["trip_id"]){
        $value["stop_id"] = $datos["stop_id"];
        $value["stop_seq"] = $datos["stop_seq"];
        array_push($all2, $value);
        break;
    }
}
```

Ilustración 17. Código PHP para relacionar los ficheros trips y stop_times

Ya tenemos la información necesaria para comenzar a introducir los datos en la base de datos y terminar con ello la obtención de los datos de una compañía.

Capítulo 4. Conexión de la aplicación móvil con el servidor

4.1 Introducción

Por un lado tenemos el servidor, por otro la aplicación que se ejecutará desde los dispositivos móviles de los usuarios. Cuando el usuario abra la aplicación querrá consultar las compañías que están registradas en el servicio para luego buscar la ruta que va a realizar, pero ¿cómo va a acceder a esta información que está alojada en el servidor?

Para comunicar nuestra aplicación móvil con la aplicación servidor necesitaremos algún tipo de tecnología que, por medio de protocolos y estándares, permita intercambiar datos entre aplicaciones a través de la red, esto es conocido como servicio web (Wikipedia, Servicio web).

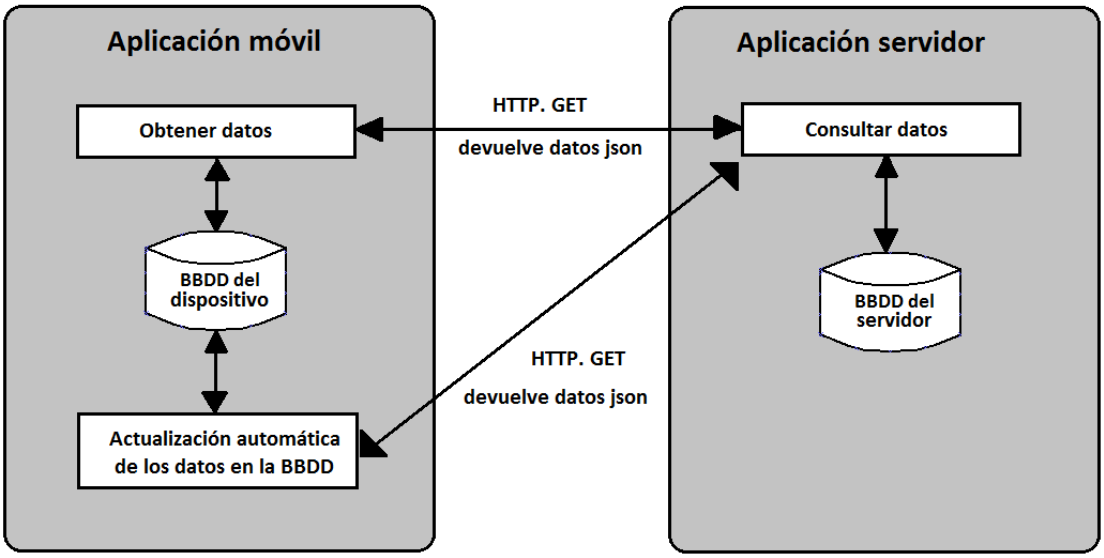


Ilustración 18. Esquema de conexión de la aplicación con el servidor

4.2 REST vs SOAP

No podemos hablar de servicios web sin tener que, como mínimo, nombrar SOAP y REST (StartCapps), dos técnicas de arquitectura utilizadas para intercambiar información en la implementación de estos servicios.

SOAP, Simple Object Access Protocol (Wikipedia, SOAP), define cómo dos aplicaciones pueden comunicarse por medio de intercambios de datos en formato XML y no permite ningún otro formato. SOAP puede ser utilizado en diferentes protocolos de transporte como HTTP, TCP o SMTP. Cabe destacar que SOAP depende de WSDL, Web Services Description Language (Wikipedia, WSDL), un formato XML utilizado para describir servicios web y que especifica la interfaz abstracta a través de la cual un cliente puede acceder al servicio y como debe utilizarlo.

REST, REpresentational State Tranfer (Marqués), es una arquitectura sencilla que se centra en el uso del estándar HTTP para la transmisión de datos. Las operaciones se solicitan mediante los métodos GET, POST, PUT y DELETE[2] y el cliente accede a los recursos utilizando las URIs únicas para cada proceso, recibiendo la información representada en formato JSON o XML.

Puesto que REST permite utilizar JSON (Inusual), que es un formato más ligero, como contenedor de los datos, parece ser la opción más adecuada para nuestro proyecto que buscaría un alto rendimiento y no utilizar muchos recursos ya que los clientes serán dispositivos móviles.

[2] Siguiendo el modelo CRUD, Create, Read, Update and Delete, que hace referencia a las operaciones básicas en una base de datos (Wikipedia, CRUD).

4.3 REST API

Como explican Asier Marqués[3] y Martin Fowler[4] en sus respectivos artículos, según el modelo Richardson Maturity Model, desarrollado por Leonard Richardson, existen tres niveles de calidad en la implementación de REST.

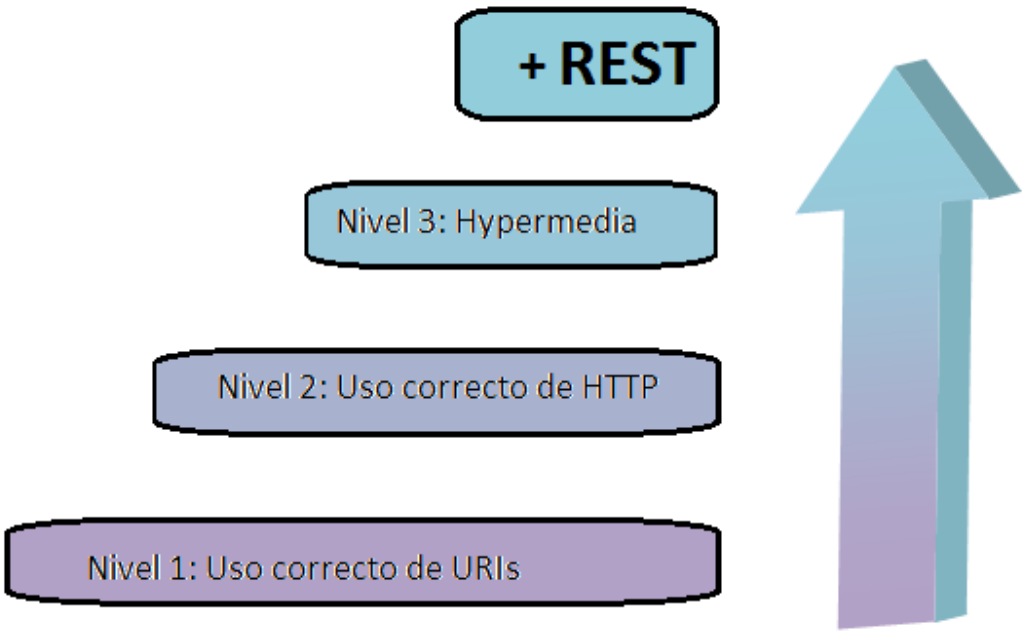


Ilustración 19. Niveles para conseguir REST según el modelo Richardson Maturity Model

El primer nivel se alcanza haciendo un uso correcto de las URIs. Existen unas reglas básicas para crear estas URIs de forma adecuada:

- No deben referirse a una acción. Por ejemplo, */server/getagencys* sería incorrecta. La forma adecuada debería ser */server/agencys*, independientemente de si vamos a consultar, editar, crear o eliminar.
- Deben ser únicas, es decir, una misma URI no debe identificar a más de un recurso.
- Deben ser independientes de formato, es decir, las URIs no pueden contener ninguna extensión de formato como por ejemplo

[3] Asier Marqués, «Conceptos sobre APIs REST». (Marqués)
[4] Martin Fowler, «Richardson Maturity Model». (Fowler)

/server/agencys.txt. Independientemente del formato en el que queramos consultar la información la URI debe ser la misma siempre, */server/agencys*.

- Deben tener un orden jerárquico lógico. Si queremos consultar la ruta 380 de la compañía 0 la URI adecuada sería */agencys/0/routes/380*.
- Si queremos filtrar la información, nunca debemos incluir este filtro como parte del nombre en la URI. Por ejemplo, si quisiéramos filtrar las rutas ordenándolas de forma descendente no utilizaríamos la URI */agencys/0/routes/orden/desc* puesto que el recurso al que se accedería sería el mismo pero utilizando una nueva URI para indicar el filtrado. Para realizar este tipo de filtrado podemos utilizar parámetros HTTP, la URI adecuada sería */agencys/0/routes?orden=DESC*.

Para alcanzar el nivel 2 tenemos que conocer los estados que devuelve el protocolo HTTP y saber utilizar correctamente los métodos GET, POST, PUT y DELETE.

Vemos ahora como podemos acceder a la información de un recurso con el mismo URI, independientemente de que queramos consultar, editar, añadir o borrar, utilizando para ello los diferentes métodos que nos provee HTTP.

GET */agencys* obtiene los datos de las compañías.

POST */agencys* crea una nueva compañía.

GET */agencys/0* obtiene los datos de la compañía 0.

PUT */agencys/0* modifica la información de la compañía 0.

DELETE */agencys/0* elimina la compañía 0.

El siguiente punto a controlar en este segundo nivel de calidad son los códigos de estado de HTTP. Hay múltiples códigos HTTP para cubrir todas las posibles situaciones que podrían ocurrir al solicitar un recurso, pero las más usuales y las que, como mínimo, debería utilizar nuestra REST API son:

Códigos 2xx. Éxito.

- **200 OK:** La petición al servidor ha tenido éxito.
- **201 Created:** La petición ha sido completada y ha resultado en la creación de un nuevo recurso.
- **204 No Content:** La petición se ha realizado correctamente pero no necesita devolver ningún contenido. Por ejemplo al eliminar un registro.

Códigos 4xx. Error en el cliente.

- **400 Bad Request:** El servidor no puede entender la petición. Puede haber un error de sintaxis.
- **401 Unauthorized:** La petición requiere autenticación de usuario.
- **403 Forbidden:** La petición ha sido legal, pero el servidor prohíbe el acceso a la información. A diferencia del estado 401, no se podría acceder aunque se autentificara el usuario.
- **404 Not Found:** El servidor no encuentra el recurso requerido.

Códigos 5xx. Error en el servidor.

- **500 Internal Server Error:** El servidor encontró un problema al procesar la petición. Mensaje de error genérico que se da cuando no hay mensajes más específicos para el problema encontrado.

Por lo tanto sería un error tratar cualquiera de estas situaciones enviando un código de estado 200 indicando que la petición ha sido procesada correctamente y realizar nuestra propia corrección enviando, en lugar del recurso solicitado, mensajes que informen del error. Esto hace necesario que el cliente que se conecte a este servidor deba conocer perfectamente el funcionamiento del mismo para gestionar los errores, lo que no es siempre una posibilidad.

El tercer y último nivel de calidad consiste en el uso de controles hipermedia (Wikipedia, Hipermedia), entendiendo hipermedia como el conjunto de métodos o procedimientos para escribir, diseñar o componer contenidos que integren soportes tales como texto, imagen, video, audio, ..., y que además, el resultado obtenido, tenga la posibilidad de interactuar con los usuarios.

HATEOAS (Charlie don't code), abreviación de Hypermedia As The Engine Of Application State, es una restricción de la arquitectura REST. Este principio se basa en que el cliente debe recibir, junto con el resultado de sus peticiones, alguna información sobre cómo acceder al enlace de otro recurso con el que esté asociado el primero (feedback).

De esta forma, el cliente tiene la posibilidad de conocer cómo acceder a todos los recursos de la API sin la necesidad de conocerla previamente. Por ejemplo, cuando solicitamos los datos de una compañía recibimos una respuesta como esta.

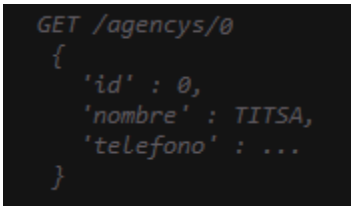


Ilustración 20. Resultado de una petición sin hipermedia

Sin embargo, si implementásemos hipermedia en nuestro servidor el resultado podría proveernos, además de la información sobre la compañía, el enlace necesario para obtener datos sobre las rutas o paradas de esa compañía.

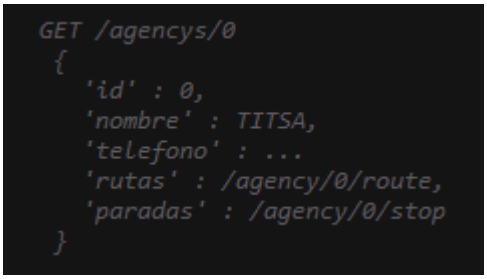


Ilustración 21. Resultado de una petición con hipermedia incorrecta

Aunque la representación anterior no es del todo correcta. ¿Cómo sabe el cliente que la información contenida en los campos *rutas* y *paradas* provee un enlace a otros recursos que podría solicitar? Para ello podemos mostrar los contenidos hipermedia dentro de un mismo campo que indique que los datos van a ser enlaces a otros recursos de la API.

```
GET /agencys/0
 {
   'id' : 0,
   'nombre' : "TITSA",
   'telefono' : ...
   '_links' :
    [
        {
            "rel" : "Rutas"
            "href" : "/agency/0/route"
        },
        {
            "rel" : "Paradas"
            "href" : "/agency/0/stop"
        }
    ]
 }
```

Ilustración 22. Resultado de una petición con hipermedia correcta

Cubriendo estos tres niveles podemos decir que nuestra aplicación sigue la estructura API REST.

4.4 Implementación en el servidor

Ahora que ya conocemos la teoría toca sumergirse en la práctica. Como hemos comentado, para decidir la operación que el cliente desea solicitar lo haremos teniendo en cuenta el método HTTP con el que se realiza la petición.

Recordemos que nuestra aplicación servidor esta implementada en PHP. Existe un framework llamado Slim (Statamic) que nos facilita la gestión de una API REST, por lo que es ideal para nuestro propósito (Tamada, How to create REST API for Android app using PHP, Slim and MySQL).

Para utilizar Slim (Muñoz), descargamos el framework y lo alojamos en el directorio de nuestro servidor. En nuestro *index.php* tenemos que incluir el paquete y registrar el autoloader.

```php
<?php
require 'Slim/Slim.php';
\Slim\Slim::registerAutoloader();
```

Ilustración 23. Instalación del framework Slim

Una vez hecho esto solo tenemos que instanciar la clase Slim y ejecutarla.

```
$app = new \Slim\Slim();

  ...

$app->run();
```

Ilustración 24. Instanciar y ejecutar clase Slim

Pero esto simplemente no hace nada, es necesario que definamos las URIs con las que se va a acceder a nuestros recursos e indicar con que métodos se accede. Slim nos ofrece una manera sencilla de gestionar estos accesos.

```
$app->get('/hello/:name', function ($name) {
    echo "Hello, $name";
});
```

Ilustración 25. Definir rutas con Slim

Basta con indicar que tipo de método es el que accede, pasándole como parámetros la URI que se utiliza para acceder al recurso y la función que ejecutará el servidor ante dicha petición.

Estos son los pasos básicos para comenzar a utilizar este framework. Para obtener más detalles sobre el uso de Slim se puede consultar el enlace a la documentación que se adjunta en la bibliografía (Statamic).

Capítulo 5. Base de datos y servicio para actualizarla

5.1 Introducción

Cuando el usuario va a utilizar la aplicación, esta tiene que conectarse con el servidor para obtener las compañías disponibles para luego volver a realizar otra petición al servidor solicitando la lista de rutas que ofrece la compañía seleccionada por el usuario y una vez que el usuario haya seleccionado la ruta deseada, nuevamente se pedirá al servidor información, esta vez sobre la ruta.

Cada vez que el usuario coja el transporte tendría que realizar estas tres peticiones al servidor consumiendo recursos y demorando el tiempo necesario por el servidor para ejecutar estas consultas y por el dispositivo móvil para interpretar los datos recibidos. Y si el usuario suele realizar con regularidad una ruta concreta, peor eficiencia tendrá nuestra aplicación puesto que en todas y cada una de ellas tendría la necesidad de realizar las consultas al servidor.

Para solucionar este problema, la aplicación debería almacenar la información de las paradas que el usuario suele utilizar con normalidad para poder acceder a ellas con mayor rapidez.

5.2 Almacenamiento de los datos en el móvil

Las bases de datos (Salvador Gómez, Bases de Datos en Android (I): Primeros pasos) son una herramienta de gran utilidad en la creación de aplicaciones informáticas que permiten almacenar información para su uso posteriormente. Android cuenta con todas las herramientas necesarias para crear y gestionar bases de datos SQLite, por lo que utilizaremos este tipo de base de datos para contener los datos de las paradas que el usuario seleccione.

Siguiendo un tutorial del libro (Lee W.-M. , 2011), lo primero que tenemos que hacer es realizar las consultas que crearán las tablas de nuestra base de datos. Recordemos que la base de datos de nuestro proyecto cuenta con seis tablas, la que contiene las compañías, una que contiene las rutas, otra que almacena las paradas, la que mantiene una relación entre las rutas y las paradas y dos tablas de información para rutas y paradas respectivamente.

```
static final String DATABASE_CREATE_AGENCY =
        "CREATE TABLE IF NOT EXISTS `agency` (`id` int(11) NOT NULL," +
                "`nick` varchar(40) NOT NULL," +
                "`name` varchar(140) NOT NULL," +
                "`url` varchar(140) DEFAULT NULL," +
                "`phone` varchar(12) DEFAULT NULL," +
                "PRIMARY KEY (`id`));";
static final String DATABASE_CREATE_ROUTES =
        "CREATE TABLE IF NOT EXISTS `routes` (`agency_id` int(11) NOT NULL," +
                "`route_id` int(11) NOT NULL," +
                "`short_name` varchar(140) NOT NULL," +
                "`long_name` varchar(260) NOT NULL," +
                "PRIMARY KEY (`agency_id`,`route_id`));";
static final String DATABASE_CREATE_STOPS =
        "CREATE TABLE IF NOT EXISTS `stops` (`agency_id` int(11) NOT NULL," +
                "`stop_id` int(11) NOT NULL," +
                "`stop_code` int(11) NOT NULL," +
                "`stop_name` varchar(140) NOT NULL," +
                "`lat` double NOT NULL," +
                "`lon` double NOT NULL," +
                "PRIMARY KEY (`agency_id`,`stop_id`));";
static final String DATABASE_CREATE_TRIPS =
        "CREATE TABLE IF NOT EXISTS `trips` (`agency_id` int(11) NOT NULL," +
                "`route_id` int(11) NOT NULL," +
                "`direction` tinyint(1) NOT NULL," +
                "`stop_seq` int(3) NOT NULL," +
                "`stop_id` int(11) NOT NULL," +
                "PRIMARY KEY (`route_id`,`direction`,`stop_seq`,`agency_id`));";
static final String DATABASE_CREATE_INFO_ROUTE =
        "CREATE TABLE IF NOT EXISTS `info_route` (`agency_id` int(11) NOT NULL," +
                "`route_id` int(11) NOT NULL," +
                "`info_id` int(11) NOT NULL," +
                "`type` varchar(30) NOT NULL," +
                "`info` varchar(260) NOT NULL," +
                "PRIMARY KEY (`route_id`,`info_id`,`agency_id`));";
static final String DATABASE_CREATE_INFO_STOP =
        "CREATE TABLE IF NOT EXISTS `info_stop` (`agency_id` int(11) NOT NULL," +
                "`stop_id` int(11) NOT NULL," +
                "`info_id` int(11) NOT NULL," +
                "`type` varchar(30) NOT NULL," +
                "`info` varchar(260) NOT NULL," +
                "PRIMARY KEY (`agency_id`,`stop_id`,`info_id`));";
```

Ilustración 26. Consultas para crear las tablas de la BD

Como vemos, cada una de las variables de la imagen contiene la sentencia SQL adecuada para crear la tabla de la BD. Para la creación de la base de datos nos apoyamos en la clase de Android SQLiteOpenHelper (Android Developers) que nos facilita la gestión y creación de bases de datos y nos ofrece un control

de versiones de la misma. Pero no basta solo con extender nuestra clase de esta otra, también tenemos que reemplazar los métodos *onCreate( )* y *onUpgrade( ).*

```
private static class DatabaseHelper extends SQLiteOpenHelper {
    DatabaseHelper(Context context){
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db){
        try {
            Log.d("DataBase", "Creando Database!");
            db.execSQL(DATABASE_CREATE_AGENCY);
            db.execSQL(DATABASE_CREATE_ROUTES);
            db.execSQL(DATABASE_CREATE_STOPS);
            db.execSQL(DATABASE_CREATE_TRIPS);
            db.execSQL(DATABASE_CREATE_INFO_ROUTE);
            db.execSQL(DATABASE_CREATE_INFO_STOP);
        } catch (SQLException e){
            e.printStackTrace();
        }
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion){
        Log.d(TAG, "Actualizando base de datos desde version " + oldVersion +
                " a " + newVersion + ", lo que destruirá toda la información antigua.");
        db.execSQL("DROP TABLE IF EXISTS agency");
        db.execSQL("DROP TABLE IF EXISTS routes");
        db.execSQL("DROP TABLE IF EXISTS stops");
        db.execSQL("DROP TABLE IF EXISTS trips");
        db.execSQL("DROP TABLE IF EXISTS info_route");
        db.execSQL("DROP TABLE IF EXISTS info_stop");
        onCreate(db);
    }
}
```

Ilustración 27. Clase SQLiteOpenHelper

El método *onCreate( )* crea la base de datos si no existe con anterioridad y el método *onUpgrade( )* se llama cuando se detecta que la variable *DATABASE_VERSION* ha cambiado y elimina todas las tablas de la BD para crearlas de nuevo.

Con esto tendríamos la base de datos creada, ahora creamos los métodos para abrirla y cerrarla.

```
//--- abre la base de datos ---
public DBAdapter open() throws SQLException {
    db = DBHelper.getWritableDatabase();
    return this;
}

//--- cierra la base de datos ---
public void close() {
    DBHelper.close();
}
```

Ilustración 28. Apertura y cierre de la BD

Ya solo necesitaríamos establecer las consultas SQL para insertar los datos y para consultarlos luego. Mostraremos solo las sentencias para insertar y consultar información de las rutas, el resto de tablas seguirían la misma lógica.

```
//--- insertar una ruta ---
public long insertRoute(int agency_id, int route_id, String short_name, String long_name){
    ContentValues initialValues = new ContentValues();
    initialValues.put(KEY_AGENCYID, agency_id);
    initialValues.put(KEY_ROUTEID, route_id);
    initialValues.put(KEY_SHORT_NAME, short_name);
    initialValues.put(KEY_LONG_NAME, long_name);
    return db.insert(DATABASE_TABLE_ROUTES, null, initialValues);
}

//--- actualizar una ruta ---
public boolean updateRoute(int agency_id, int route_id, String short_name, String long_name){
    ContentValues args = new ContentValues();
    args.put(KEY_SHORT_NAME, short_name);
    args.put(KEY_LONG_NAME, long_name);
    return db.update(DATABASE_TABLE_ROUTES, args, KEY_AGENCYID + "=" + agency_id + " AND " +
            KEY_ROUTEID + "=" + route_id, null) > 0;
}

//--- recuperar rutas ---
public Cursor getAllRoutes(){
    return db.query(DATABASE_TABLE_ROUTES, new String[] {KEY_AGENCYID,
            KEY_ROUTEID, KEY_SHORT_NAME, KEY_LONG_NAME}, null, null, null, null, null);
}

//--- recuperar rutas de una agencia ---
public Cursor getRoutesFromAgency(int agency_id){
    Cursor cursor = db.query(true, DATABASE_TABLE_ROUTES, new String[] {KEY_AGENCYID,
            KEY_ROUTEID, KEY_SHORT_NAME, KEY_LONG_NAME}, KEY_AGENCYID + "=" + agency_id,
            null, null, null, null, null);
    //if (cursor != null)
        //cursor.moveToFirst();
    return cursor;
}
```

Ilustración 29. Inserción y consulta de datos de rutas

Como podemos observar, Android utiliza la clase Cursor (Alcalde) para devolver los resultados de las consultas realizadas a la base de datos. Un Cursor es como un puntero al resultado devuelto por la base de datos, contiene una colección de filas lo que nos permite gestionarlas de forma más eficiente.

Con la base de datos creada y las consultas realizadas, lo único que tenemos que hacer ahora es que cuando el usuario consulte una ruta determinada, toda la información recibida por el servidor quede almacenada en la base de datos del dispositivo. A partir de ahora, cuando se le solicite a la aplicación la lista de rutas de una compañía primero comprobará las rutas alojadas en la base de datos del dispositivo, si la que el usuario está buscando no se encuentra en esta lista tendrá la opción de buscar más rutas en la base de datos del servidor. De esta forma evitamos realizar consultas al servidor con tanta frecuencia, mejorando los tiempos de respuesta de la aplicación móvil.

5.3 Tareas asíncronas

Como las tareas en las que se tiene que consultar el servidor o descargar datos del mismo pueden demorar tiempo en realizarse, sería conveniente ejecutar estos procesos en hilos aparte mientras el hilo principal ofrece algún aviso al usuario del trabajo que está realizando la aplicación en segundo plano.

Esto se consigue creando un método que amplíe de la clase *AsyncTask* (Lee W.-M. , Desarrollar servicios Android, 2011). Esta clase permite llevar a cabo una tarea en segundo plano sin la necesidad de gestionar manualmente los hilos de ejecución.

```
class GetRoutes extends AsyncTask<String, Void, Integer> {

    /**
     * Before starting background thread Show Progress Dialog
     * */
    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        if(pDialog != null)
            pDialog.dismiss();
        pDialog = new ProgressDialog(ViewRoutesActivity.this);
        pDialog.setMessage("Cargando rutas. Por favor, espere...");
        pDialog.setIndeterminate(false);
        pDialog.setCancelable(true);
        pDialog.show();
    }
```

Ilustración 30. Creación de un método que se ejecuta en un hilo aparte

Al ampliar la clase *AsyncTask* se debe especificar tres tipos genéricos que especifican el tipo de datos utilizados por los métodos *doInBackground( )*, *onProgressUpdate( )* y *onPostExecute( )* respectivamente. Nosotros solo utilizaremos los hilos para descargar los datos en segundo plano mientras se muestra un aviso al usuario y no necesitamos mostrar el progreso de la descarga. Es por este motivo que prescindiremos de utilizar el método *onProgressUpdate( )*, pero aun así estamos obligados a definir el tipo de dato al ampliar la clase.

En lugar de avisar del progreso, utilizaremos el método *onPreExecute( )* para crear un aviso que se mostrará en la pantalla del dispositivo mientras este descarga los datos.

```
protected Integer doInBackground(String... args) {

    int success = 0;
    try {
        // Building Parameters
        List<NameValuePair> params = new ArrayList<NameValuePair>();

        JSONObject json = jParser.makeHttpRequest(
                url_routes + "/" + agencyId, "GET", params);

        success = json.getInt(TAG_SUCCESS);
        if (success == 1) {


            ...


        }
    } catch (JSONException e) {
        e.printStackTrace();
    }

    return success;
}
```

Ilustración 31. Método doInBackground de la clase AsyncTask

El método *doInBackground( )* será el que se ejecute en segundo plano por lo que es donde se situará el código que realice la petición al servidor e interprete los datos devueltos en formato JSON. Como se puede observar, la petición al servidor se realiza por medio de una petición HTTP en la que se especifica la URL del recurso que queremos solicitar y los parámetros necesarios para procesar dicha petición. En este ejemplo no necesitamos especificar ningún parámetro adicional, pero cuando el usuario seleccione que ruta desea si será

imprescindible que se especifique el id de la ruta como parámetro para poder acceder a sus datos.

jParser es un objeto de la clase JSONParser creada en el proyecto. Esta clase es responsable de gestionar las peticiones HTTP y obtener sus respuestas. En la bibliografía de la memoria hay un enlace con un tutorial que explica cómo crear esta clase (Tamada, Android JSON Parsing Tutorial).

Tras obtener la respuesta del servidor en formato JSON, es necesario interpretar estos datos para poder mostrarlos correctamente en el dispositivo y que el usuario pueda entenderlos.

Por último, el método *doInBackground( )* devuelve un entero, que es el tercer tipo especificado en la ampliación de la clase *AsyncTask*, que indica si la descarga de los datos se ha realizado correctamente y que será utilizado por el método *onPostExecute( )*.

```
    protected void onPostExecute(Integer result) {
        // dismiss the dialog after getting all products
        if(pDialog != null)
            pDialog.dismiss();
        // updating UI from Background Thread
        if (result == 1){

          ...

        }
        else{
            Toast toast = Toast.makeText(getApplicationContext(),
                    "Error al descargar los datos desde el servidor", Toast.LENGTH_LONG);
            toast.show();
        }
    }
}
```

Ilustración 32. Método onPostExecute de la clase AsyncTask

El método *onPostExecute( )* se ejecuta en el hilo de la interfaz del usuario y se llama cuando el método *doInBackground( )* ha terminado de ejecutarse. Es en este método donde se localiza el código necesario para mostrar en el dispositivo los datos descargados desde el servidor. Lo primero que tenemos que realizar es eliminar el mensaje que avisaba al usuario de la descarga de los datos para luego comprobar que dicha descarga se ha realizado correctamente, en cuyo caso pasamos a mostrar la información. En caso de error en la descarga mostramos un pequeño mensaje para avisar al usuario.

5.4 Servicio de actualización periódica

Tal y como hemos diseñado la aplicación hasta el momento, una vez que el usuario se descargue los datos de una ruta, esta no volverá a ser consultada en el servidor y puede que haya actualizaciones de información en estas rutas que no quedarían reflejadas en los resultados devueltos por la aplicación móvil. Imaginemos que la compañía de una ruta añade una parada nueva o elimina una que ya tenía de su recorrido, la información que se tiene en el dispositivo sería errónea y llevaría a equivocación al usuario que utiliza la aplicación.

Para solventar este problema, se ha decidido crear un servicio que, periódicamente, se descargue del servidor los datos de las rutas que tiene alojados en el dispositivo para mantener esta información lo más actualizada posible.

Para crear nuestro servicio (Lee W.-M. , Desarrollar servicios Android, 2011) definimos una nueva clase que extienda de la clase *Service* y comenzamos implementando el método *onStartCommand( ).*

```
public class MyService extends Service {

  @Override
  public int onStartCommand(Intent intent, int flags, int startId) {
      // Abrir db
      db = new DBAdapter(this);

      doSomethingRepeatedly();

      return START_STICKY;
  }

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

      if (timer != null){
          timer.cancel();
      }

      Toast.makeText(this, "Service Destroyed", Toast.LENGTH_LONG).show();
  }
}
```

Ilustración 33. Creación del servicio

El método *onStartCommand( )* es llamado cuando se inicia el servicio. Este método es el comienzo del servicio y devuelve la constante *START_STICKY* para indicar que el servicio continuará ejecutándose hasta que este se detenga explícitamente.

El método *onDestroy( )* se llama para detener el servicio y limpiar los recursos utilizados por el mismo. Como veremos a continuación, utilizaremos la clase *Timer* para correr nuestro servicio y es conveniente que, cuando destruyamos el mismo, detengamos también el uso de esta clase.

Vimos en la creación del método *onStartCommand( )* que se realizaba una llamada a la función *doSomethingRepeatedly( )*, pero no comentamos el motivo. Recordemos que nuestro servicio debe ser definido para que se ejecute cada X tiempo, actualizando la información de la base de datos del dispositivo para mantenerla al orden del día.

```
static final int UPDATE_INTERVAL = 1000 * 60 * 60 * 24;
private Timer timer = new Timer();
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
private void doSomethingRepeatedly() {
    timer.scheduleAtFixedRate( new TimerTask() {
        public void run() {

            new DoBackgroundTask().execute();

        }
    }, 0, UPDATE_INTERVAL);
}
```

Ilustración 34. Realizar tareas de forma repetida

Para conseguir realizar una tarea repetidamente en un intervalo de tiempo específico, podemos utilizar la clase *Timer*. Lo primero que debemos hacer es crear un objeto *Timer*, y definimos un intervalo de tiempo, en milisegundos, en el que queremos que se ejecute nuestro servicio. Nosotros hemos definido que el servicio se ejecute cada 24 horas.

Llamamos al método *scheduleAtFixedRate( )* de la clase *Timer* pasándole una instancia de la clase *TimerTask*, de modo que todo el código localizado dentro del método *run( )* de esta clase será el que se ejecute reiteradamente. Los otros dos parámetros que se le pasan al método *scheduleAtFixedRate( )* son la cantidad de tiempo, en milisegundos, a esperar hasta que se ejecute la primera

iteración del servicio y el intervalo de tiempo, también en milisegundos, entre las siguientes ejecuciones.

Capítulo 6. Geolocalización y proximidad

6.1 Introducción

Quizás la parte más interesante del proyecto es la posibilidad de avisar al usuario sobre las próximas paradas a medida que se va acercando a ellas teniendo como única referencia la geolocalización actual del dispositivo en todo momento.

Necesitaremos por tanto un servicio (Lee W.-M. , Servicios de localización, 2011) que se ejecute y obtenga la posición actual del usuario, calcule la parada más próxima del recorrido seleccionado para saber en qué punto del recorrido se encuentra y que, además, se mantenga en ejecución y verifique constantemente dicha posición para cotejarla con la geolocalización de la siguiente parada en el recorrido.

6.2 Obtener la ubicación del dispositivo

Lo primero que debemos hacer es localizar la posición del usuario. Actualmente, los dispositivos móviles cuentan con varias formas para hacer esto. La primera opción es utilizando el receptor GPS, sin embargo esto requiere un cielo despejado y no siempre funcionará en interiores o lugares que los satélites no puedan alcanzar. Otra forma de obtener la geolocalización es por medio de triangulación de torres móviles, no tan precisa como el GPS pero funciona en interiores. Por último, también se puede localizar el dispositivo basándose en la triangulación Wi-Fi. De los tres métodos descritos, la triangulación Wi-Fi es el menos preciso.

Veamos cómo hacer que nuestra aplicación determine la ubicación del dispositivo (Salvador Gómez, Localización geográfica en Android (II)). Para ello Android nos proporciona una clase llamada *LocationManager* (Android Developers).

```
//Obtenemos una referencia al LocationManager
locManager =
    (LocationManager)getSystemService(Context.LOCATION_SERVICE);

// getting GPS status
boolean isGPSEnabled = locManager
        .isProviderEnabled(LocationManager.GPS_PROVIDER);

// getting network status
boolean isNetworkEnabled = locManager
        .isProviderEnabled(LocationManager.NETWORK_PROVIDER);

if (!isGPSEnabled && !isNetworkEnabled) {
    // no network provider is enabled
} else {
```

Ilustración 35. Obtener ubicación del dispositivo

Para empezar, hay que obtener una referencia a la clase *LocationManager* por medio del método *getSystemService( )*. Lo siguiente que hacemos es comprobar si los proveedores están activados en cuyo caso procedemos a obtener la ubicación.

```
locListener = new LocationListener() {
    public void onLocationChanged(Location location) {
        loc = location;
        Log.d("LocListener", "location changed");
    }
    public void onProviderDisabled(String provider){
        Log.d("LocListener", "Provider Disabled");
    }
    public void onProviderEnabled(String provider){
        Log.d("LocListener", "Provider Enabled");
    }
    public void onStatusChanged(String provider, int status, Bundle extras){
        Log.d("LocListener", "Provider Status: " + status);
    }
};
```

Ilustración 36. Clase LocationListener

Android nos facilita la clase *LocationListener* para recibir notificaciones de la clase *LocationManager* cuando la posición del usuario ha cambiado. Para implementar esta clase debemos definir cuatro métodos. Cuando se detecta un cambio de ubicación *LocationListener* llamará al método *onLocationChanged( )*. Si se activa o desactiva el proveedor se hará una llamada a los métodos *onProviderEnabled( )* y *onProviderDisabled( )* respectivamente. El último método, *onStatusChanged( )*, se lanzará cuando se detecte un cambio en el estado del proveedor.

Lo que realmente nos interesa es lo que sucede cuando el usuario cambia de ubicación, por lo que nos centraremos en el método *onLocationChanged( )*. Cuando se detecta el cambio almacenamos la nueva posición, más adelante veremos que uso hacemos de esta variable.

Veamos lo que tenemos hasta ahora. Hemos obtenido una referencia a *LocationManager* y hemos implementado la clase *LocationListener* para detectar los cambios de ubicación. Ya solo queda registrar una petición para que nos notifique de estos cambios. Esto se consigue por medio del método *requestLocationUpdates( )*.

```
Location location = null;
if (isGPSEnabled) {
    locManager.requestLocationUpdates(
            LocationManager.GPS_PROVIDER,
            1000 * 5,
            50, locListener);
    Log.d("GPS", "GPS Enabled");
    if (locManager != null) {
        location = locManager
                .getLastKnownLocation(LocationManager.GPS_PROVIDER);
    }
}
if (isNetworkEnabled) {
    if (location == null){
        locManager.requestLocationUpdates(
                LocationManager.NETWORK_PROVIDER,
                0,
                0, locListener);
        Log.d("Network", "Network Enabled");
        if (locManager != null) {
            location = locManager
                    .getLastKnownLocation(LocationManager.NETWORK_PROVIDER);
        }
    }
}
```

Ilustración 37. Método requestLocationUpdates( )

El método *requestLocationUpdates( )* recibe cuatro argumentos. El primero es el proveedor con el que se registra, en nuestro caso tratamos de obtener la posición del usuario por medio del GPS, pero en caso de no conseguirla lanzamos el mismo método pero esta vez utilizando el proveedor de red. El segundo y tercer argumento son los intervalos de tiempo, en milisegundos, y distancia, en metros, mínimos para que el sistema notifique los cambios de ubicación. Por último el método necesita una referencia al objeto *listener* que recibirá las notificaciones.

De esta forma ya tendríamos un "oyente" registrando cada cambio de posición que se produce en el dispositivo. Con esto conseguimos tener controlada la ubicación del usuario en todo momento y podemos definir por primera vez la parada actual en la que se encuentra el usuario. Solo tenemos que comprobar las coordenadas de latitud y longitud de todas las paradas de la ruta que ha seleccionado el usuario y, por medio de una función que provee Android para la clase *location*, obtener la distancia entre cada parada y la posición actual del usuario, quedándonos con la que obtenga el mínimo valor.

6.3 Seguimiento de una ubicación

Ya sabemos en qué parada está localizado nuestro usuario, y puesto que tenemos un campo de la tabla para indicar la secuencia ordenada de paradas de una ruta conocemos cuál será la siguiente parada del trayecto. Solo faltaría añadir algún tipo de aviso que nos permita notificar al usuario cuando esté próximo a la localización que nos interesa. La clase *LocationManager* posee un método con esta finalidad, *addProximityAlert( )* (androidmyway).

```
locManager.addProximityAlert(nextstop.latitud, nextstop.longitud, 60, -1, pending);

IntentFilter filter = new IntentFilter("com.example.sisapi.ProximityAlert");
proximityIntent = new ProximityIntentReceiver();
registerReceiver(proximityIntent, filter);
```

Ilustración 38. Añadir alerta de proximidad

El método *addProximityAlert( )* recibe cinco argumentos. Los dos primeros son las coordenadas de la ubicación a la que queremos realizarle un seguimiento. El segundo argumento es el radio en metros, tomando como punto central la posición que se está siguiendo. Cuando el usuario entre en este radio saltará el aviso. El tercer argumento es el tiempo, en milisegundos, durante el cual la alarma es válida, pasándole el valor -1 conseguimos que la alerta no caduque. El último argumento es un objeto de la clase *PendingIntent* que se lanzará cuando el usuario entre o salga del radio de detección de la alerta.

La alarma está definida, pero también tenemos que registrar el receptor de la alerta de proximidad. Pero no basta solo con registrarlo, también tenemos que definir el comportamiento de este receptor.

```
public class ProximityIntentReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        ...
    }
}
```

Ilustración 39. Implementar el receptor de la alerta de proximidad

Cuando la alerta de proximidad se ejecute, el programa debería enviar una notificación al usuario y recargar la actividad anterior, dejando esta vez como parada actual a la que le estábamos realizando el seguimiento y creando una nueva alerta para la próxima parada del recorrido. Pero antes de hacer esto debemos eliminar la alerta anterior.

```
if (loc != null) {
    unregisterReceiver(proximityIntent);
```

Ilustración 40. Eliminar alerta de proximidad

Con esto ya tenemos una aplicación Android que por medio de la geolocalización, obteniendo la ubicación por medio del GPS o del proveedor de red, estima en que parada de la ruta seleccionada por el usuario se encuentra, tomando la que se encuentra más cerca del usuario en el momento en el que ejecuta la aplicación. Tras esto comienza a realizar un seguimiento de la posición actual y notifica al usuario de la proximidad a la siguiente parada.

Capítulo 7. Desarrollo accesible en Android

7.1 Introducción

En este capítulo abordaremos el diseño de la interfaz que se ha realizado para la aplicación móvil teniendo en cuenta el perfil de usuario para el que está pensada, personas con discapacidad visual. Empezaremos hablando de dos conceptos clave en el ámbito de la informática para el desarrollo adecuado de este tipo de interfaces.

Cuando hablamos de usabilidad nos referimos a la facilidad con la que las personas pueden utilizar una herramienta para alcanzar un objetivo concreto con efectividad y satisfacción.

La accesibilidad se define como el grado en el que todas las personas pueden utilizar un servicio independientemente de sus capacidades técnicas, cognitivas o físicas. Decimos que una aplicación es accesible (Informática y Accesibilidad para Todos) cuando cualquier usuario puede utilizarla en su dispositivo móvil de forma satisfactoria con su sistema de acceso habitual.

Debido a que los potenciales usuarios de esta aplicación serán invidentes, es necesario que todo el contenido que se muestre se pueda reproducir de alguna forma. Para facilitar este trabajo Android cuenta con una herramienta de accesibilidad llamada TalkBack.

7.2 Pautas para un desarrollo accesible

A la hora de realizar un diseño accesible podemos tener en cuenta algunas técnicas (Hidalgo Reina) que nos guiarán en el proceso de creación de nuestra interfaz y que enumeraremos a continuación.

1. **Conocer las barreras y necesidades de nuestros usuarios.** Como hemos dicho, nuestros principales usuarios tienen

discapacidad visual así es que de poco servirá todo el contenido que mostremos si no es posible reproducirlo.

2. **No centrarse en un único perfil.** Aunque tengamos un perfil de usuario principal, no deberíamos centrarnos únicamente en este. Existen más grupos de personas con otro tipo de discapacidades y que podríamos tener en cuenta.
3. **No basarse únicamente en el color para dar información y diseñar con el suficiente contraste.** Si hay poco contraste en las combinaciones de color de la interfaz algunos grupos de usuarios podrían tener problemas para visualizar el contenido.
4. **No abusar del lenguaje técnico.** Cuanto más sencillo sea, más fácil de comprender y más satisfactorio para el usuario será.
5. **No saturar las pantallas de elementos e información.** Si tratamos de eliminar el contenido prescindible ganaremos simplicidad en la navegación de la aplicación.
6. **Probar, testear.** Una parte importante de todo diseño es verificar que todo funciona correctamente una vez acabado.
7. **Interacción con campos.** Si disponemos de formularios en nuestra aplicación, tendremos que asociar correctamente las etiquetas con los controles.
8. **Alternativa al contenido no textual.** Si ofrecemos contenido no textual en la interfaz debemos ofrecer su equivalente textual.

Aparte de estas técnicas generales para cualquier desarrollo accesible, también hay que conocer las pautas de accesibilidad que hay sobre Android. En la web de Android Developers (Android Developers) encontramos una lista de buenas prácticas para hacer más accesible nuestra aplicación utilizando el framework de Android.

Primero tenemos que asegurarnos de que nuestra aplicación es accesible realizando los siguientes pasos:

1. **Etiquetar los elementos de la interfaz.** Algunos elementos de la interfaz indican su significado o su uso por medio de alguna imagen. Estos elementos no podrán ser interpretados por personas con discapacidad visual, lo que los hace inservibles para ellos. Podemos hacer más accesible estos elementos utilizando el atributo

android:contentDescription. Este texto no aparecerá en la interfaz, pero si el usuario tiene activada las opciones de accesibilidad que proporcionan mensajes de voz, cuando el usuario navegue sobre estos elementos el texto se reproducirá.

```
<CheckBox android:id="@+id/checkbox_service"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/ActiveService"
    android:onClick="onCheckboxClicked"
    android:textSize="24sp"
    android:layout_marginTop="10dip"
    android:paddingTop="12dip"
    android:paddingBottom="12dip"
    android:contentDescription="@string/ActiveService" />
```

Ilustración 41. Atributo contentDescription

2. **Habilitar la navegación por foco.** Si el usuario necesita un teclado direccional para navegar por la interfaz, los elementos de esta deberían poder captar el foco. Hay elementos de la interfaz que, de forma predeterminada, no son navegables de esta forma. Supongamos que tenemos un texto en la interfaz y que el usuario utiliza un teclado direccional con TalkBack habilitado, el usuario nunca podrá escuchar este texto puesto que nunca podrá poner el foco en él. Para solucionar esto basta con utilizar el atributo *android:focusable*. Estableciendo a *true* este atributo conseguiremos que el usuario pueda poner el foco en los elementos deseados e interactuar con ellos.

```
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/ServiceExp"
    android:textSize="18sp"
    android:paddingTop="6dip"
    android:focusable="true" />
```

Ilustración 42. Atributo focusable

Lo próximo que te recomiendan en Android Developers es que realices las siguientes pruebas para asegurar un mínimo nivel de accesibilidad en la aplicación:

1. **Control direccional.** Verificar que la aplicación es navegable sin utilizar la pantalla táctil. La aplicación debe ser navegable

utilizando solamente un control direccional. Para testear esto podemos descargar de PlayStore una aplicación que simula este tipo de teclado, Eyes-Free Keyboard (Eyes-Free Project).

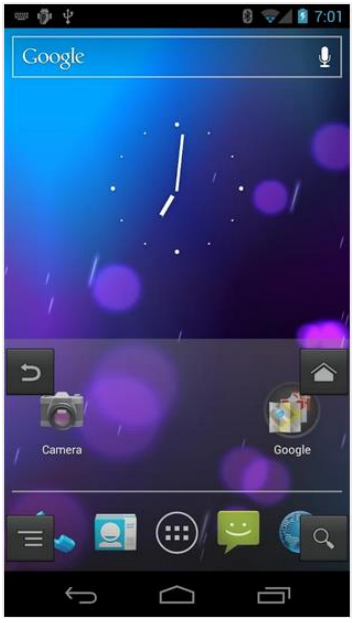


Ilustración 43. Teclado direccional de Eye-Free Keyboard

2. **TalkBack.** Verificar que todos los elementos de la interfaz ofrecen una descripción de audio adecuada cuando TalkBack está habilitado. Para habilitar TalkBack hay que ir a ajustes->accesibilidad->TalkBack y activarlo.

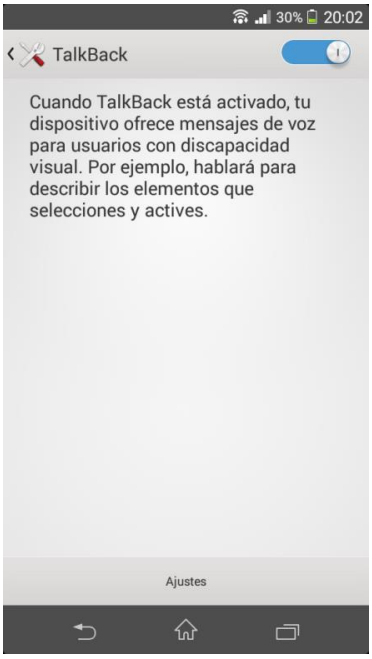


Ilustración 44. Habilitar TalkBack

3. **Exploración táctil.** Verificar que todos los elementos de la interfaz proveen una descripción de audio cuando la exploración táctil está activada. Para habilitar esta opción hay que activar TalkBack primero y luego en sus ajustes activar la exploración táctil.

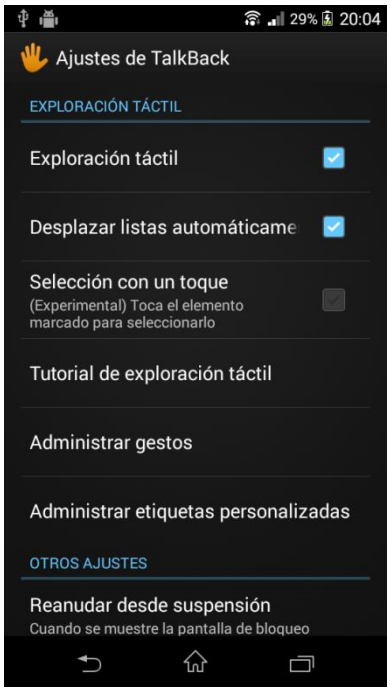


Ilustración 45. Habilitar exploración táctil

4. **Tamaño de los controles.** Todos los controles que el usuario pueda seleccionar deben tener un tamaño mínimo de 48dip (aproximadamente 9mm) en ancho y alto.

```
<Button android:id="@+id/btnViewAgencys"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/ViewAgencys"
    android:layout_marginTop="25dip"
    android:paddingTop="15dip"
    android:paddingBottom="15dip"
    android:textSize="30sp" />
```

Ilustración 46. Tamaño accesible de los controles

En este botón podemos ver que contamos con dos atributos de relleno de 15dip cada uno, sumados al tamaño del texto que se muestra en el control superan los 48dip requeridos.

5. **Correcto funcionamiento de los controles con TalkBack.** Verificar que los gestos necesarios para navegar por la aplicación como el scroll en las listas, listas despegables y demás, siguen funcionando adecuadamente cuando TalkBack está activado.
6. **No utilizar feedback únicamente de audio.** Los mensajes y notificaciones al usuario que se realicen mediante audio deben tener una alternativa para los usuarios con problemas auditivos. Por ejemplo, un sonido de alerta para avisar al usuario de algún evento debe ir acompañado de una notificación del sistema o alguna otra representación visual.

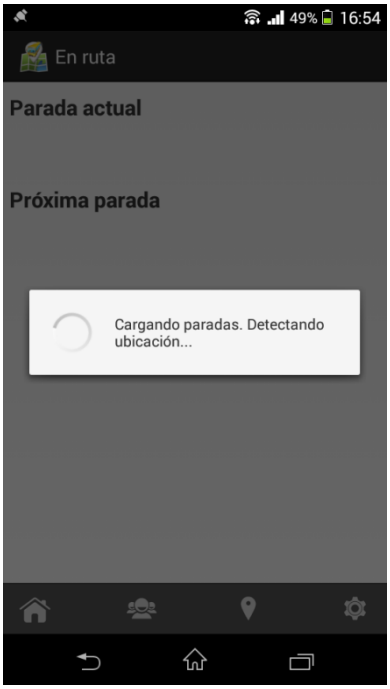


Ilustración 47. Notificación mediante pDialog

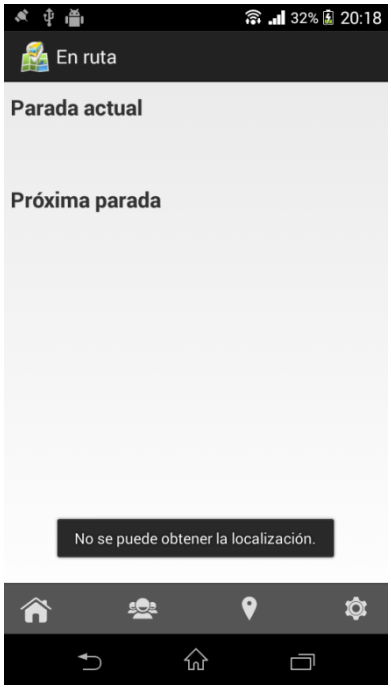


Ilustración 48. Notificación mediante Toast

7.3 Diseño de la aplicación

Estudiadas todas las pautas y prácticas recomendadas de accesibilidad hemos apostado por un diseño sencillo y minimalista, tratando de evitar el exceso de elementos e información en la interfaz y con el contraste necesario en la combinación de colores utilizados (negro, blanco y gris).

En todas las pantallas de la aplicación contaremos con un menú con opciones de navegación para llegar a las pantallas más importantes así como al menú de configuración de la aplicación.

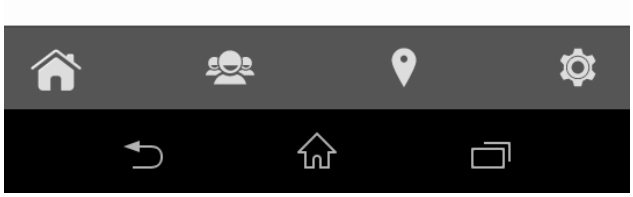


Ilustración 49. Menú de la aplicación

El primero de los botones situará al usuario en la pantalla inicial de la aplicación. El segundo muestra las compañías que el usuario ya ha consultado y se mantienen en la base de datos de su dispositivo móvil, en caso de no haber consultado ninguna aún realizará la consulta al servidor y mostrará todas las compañías. Para utilizar el tercer control primero se tiene que haber seleccionado una compañía como favorita en el menú de ajustes, después la opción mostrará las rutas de dicha compañía que ya han sido consultadas por el usuario. Por último, el cuarto elemento muestra las opciones de configuración de la aplicación.

Suponiendo que el usuario ya ha consultado algunas rutas de una compañía, este sería un mapa navegacional del menú de la aplicación.

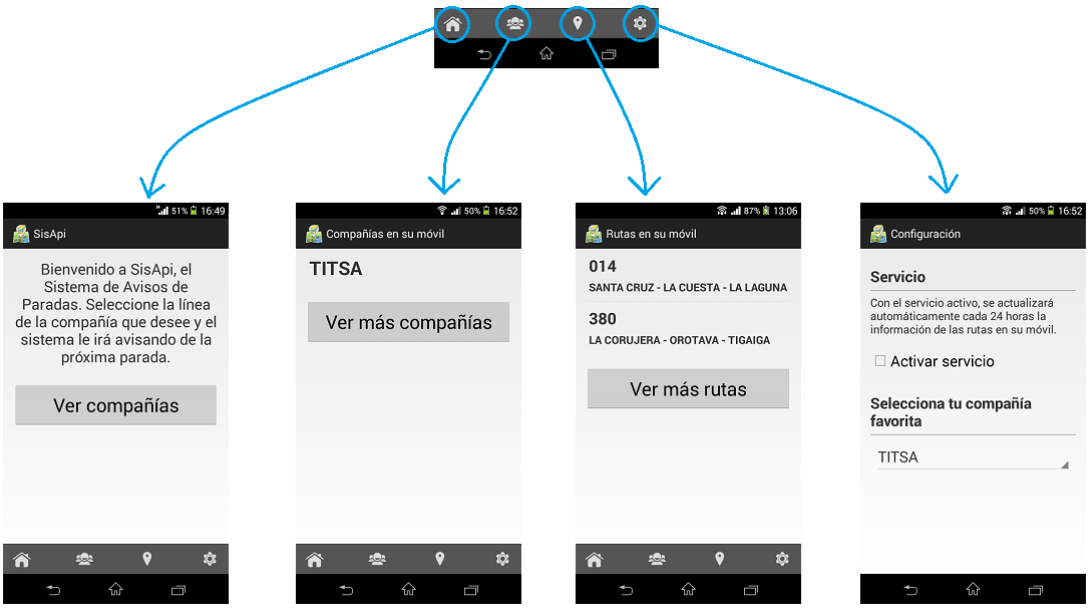


Ilustración 50. Mapa navegacional del menú de la aplicación

Si es la primera vez que el usuario utiliza la aplicación y comienza a navegar por ella, este sería un ejemplo de mapa navegacional en el que el usuario selecciona una ruta de una compañía y la aplicación comienza a detectar su ubicación y avisarle de las próximas paradas en la ruta elegida.

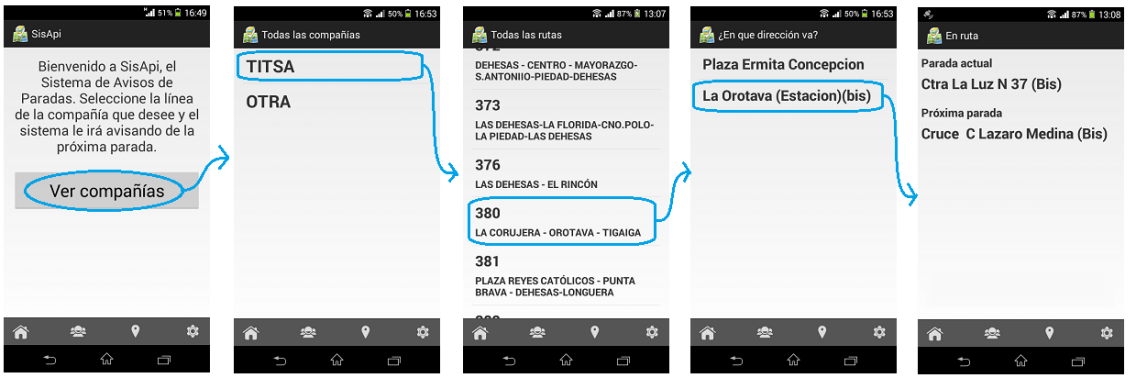


Ilustración 51. Mapa navegacional ejemplo de uso de la aplicación

Este es un mapa más general sobre el uso de la aplicación teniendo en cuenta la información que puede o no estar almacenada ya en la base de datos del dispositivo móvil.

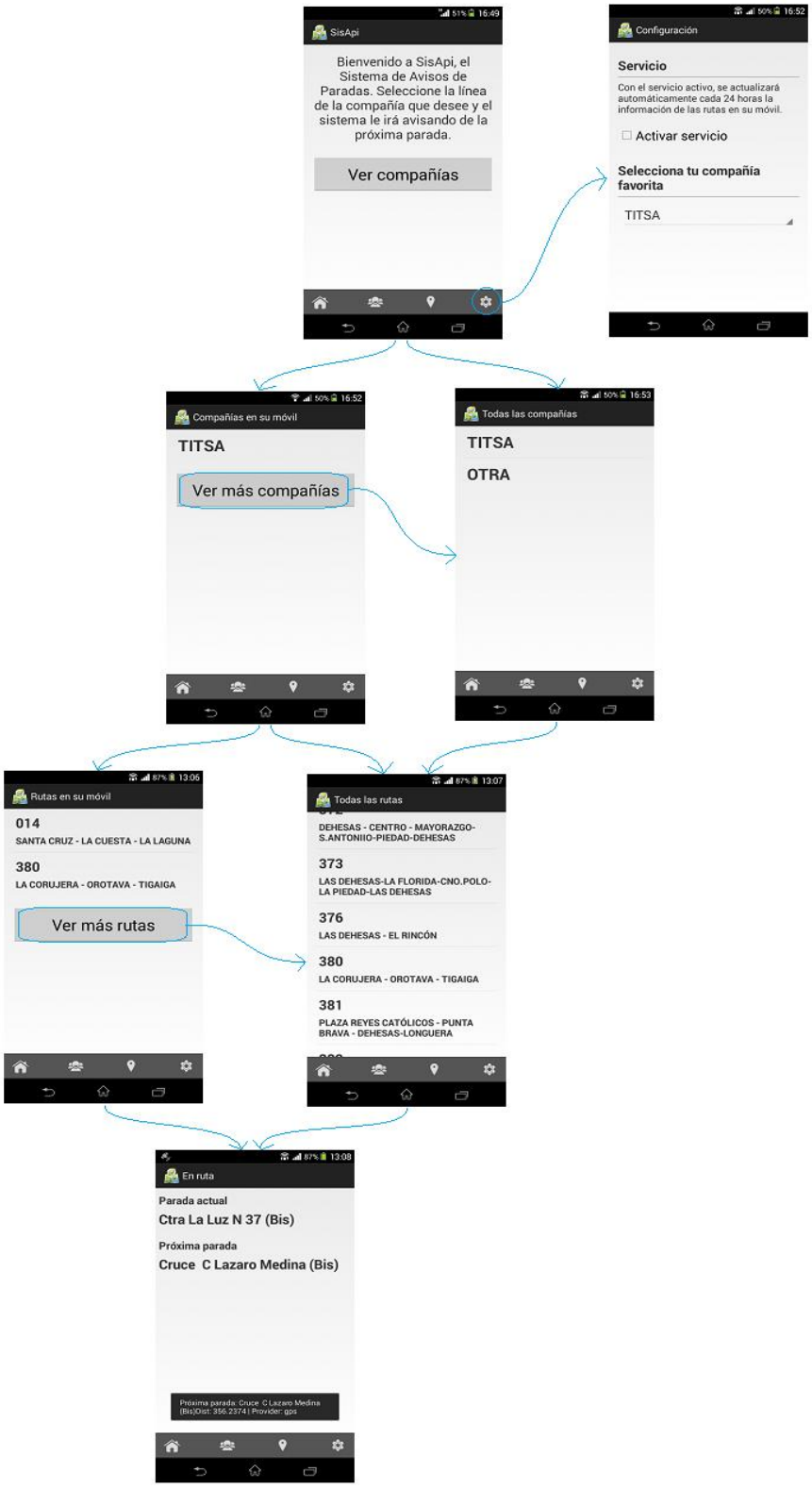


Ilustración 52. Mapa navegacional general de la aplicación

Para terminar el capítulo mostramos el icono diseñado para la aplicación.

Ilustración 53. Icono de la aplicación

Como vemos, el icono está formado por un vehículo de transporte público localizado sobre un mapa, combinando de esta forma los dos elementos principales de la aplicación.

Capítulo 8. Portal web

8.1 Introducción

Como vimos en el capítulo 2 de esta memoria, la estructura de los datos de GoogleTransit no contempla la posibilidad de añadir información adicional sobre las rutas o paradas de las compañías. Cuando hablamos de información adicional nos referimos a avisos que las compañías podrían querer transmitir a sus clientes, como por ejemplo el traslado de alguna parada de un punto a otro, obras en una parada, desvío de alguna ruta, etc.

Queremos que nuestra aplicación sea capaz de tener en cuenta estos avisos y transmitirlos al usuario. Ya vimos cómo se definió la base de datos del proyecto para poder almacenar estos datos, pero si GoogleTransit no provee estos datos ¿de dónde los vamos a sacar? Para este propósito hemos decidido desarrollar un portal web en el que las compañías puedan registrarse y añadir esta información adicional.

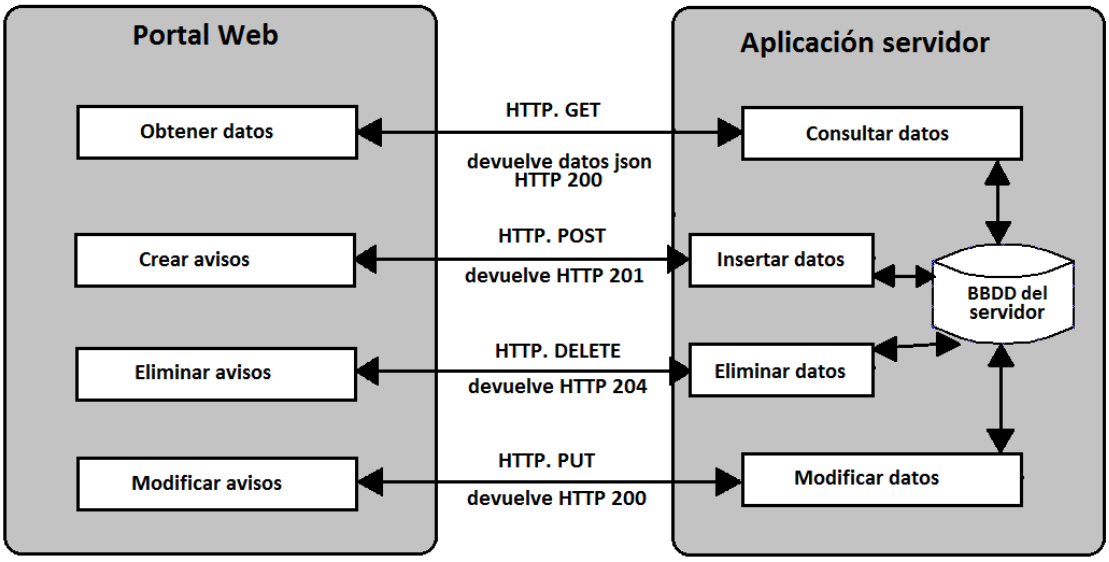


Ilustración 54. Esquema de conexión del portal web con el servidor

8.2 Herramientas utilizadas

Antes de hablar del desarrollo del portal, es importante mencionar el modelo-vista-controlador (Wikipedia, Modelo–vista–controlador). Este es un patrón de arquitectura de software que permite separar las operaciones entre el usuario y la aplicación en tres grupos: Modelo, Vista y Controlador.

- Modelo: Representa la información con la que opera el sistema, gestionando todos los accesos a dicha información. Provee a la Vista de la información que solicita para ser mostrada. Las solicitudes al Modelo se realizan a través del Controlador.
- Vista: Recibe la información del Modelo y la presenta al usuario.
- Controlador: Es el encargado de conectar el Modelo con la Vista. Gestiona las acciones que el usuario realiza en la Vista y realiza las peticiones al Modelo.

Para gestionar los accesos al servidor desde el portal web utilizamos una librería llamada Backbone.js (Backbonejs.org), que nos permite estructurar la web siguiendo el patrón MVC. De esta forma nuestro modelo contendría las URIs de nuestro servidor a las que se debería enviar la petición para obtener los recursos deseados, las vistas serán los templates en HTML con los que interactuará el usuario y el controlador en Backbonejs, que decidirá qué acción realizará en base a la URL solicitada, se gestiona por medio de los *routers*.

En la bibliografía hay un tutorial que te indica paso a paso como implementar esta librería en tu proyecto (Marín).

La única dependencia que tiene Backbone.js es Underscore.js (Underscorejs.org), una librería javascript que proporciona multitud de funciones y utilidades creada por los mismos autores de Backbone.js.

También se han utilizado la librería JQuery (JQuery Foundation)y el framework Bootstrap (Otto & Thornton) para el diseño y desarrollo de la plataforma web.

Cuantas más librerías usamos y teniendo en cuenta que estas pueden tener dependencias unas de otras, como en el caso de Backbone y Underscore, resulta útil algo que nos permita gestionar estas librerías y relaciones entre ellas. Este

es trabajo de Require.js (Chung), otra librería que se encarga de gestionar la carga de módulos.

8.3 Diseño del portal

En la base de datos que teníamos hasta ahora no contemplábamos el perfil de los usuarios que podrán acceder a la plataforma web. Necesitamos definir un rol de usuario/compañía con un email y una clave asociado a las compañías que ya están registradas en nuestra base de datos para que puedan añadir, modificar o eliminar la información de las rutas y paradas propias. Para ello es necesario añadir una nueva tabla para almacenar esta información. El nuevo esquema de la base de datos quedará de la siguiente forma.

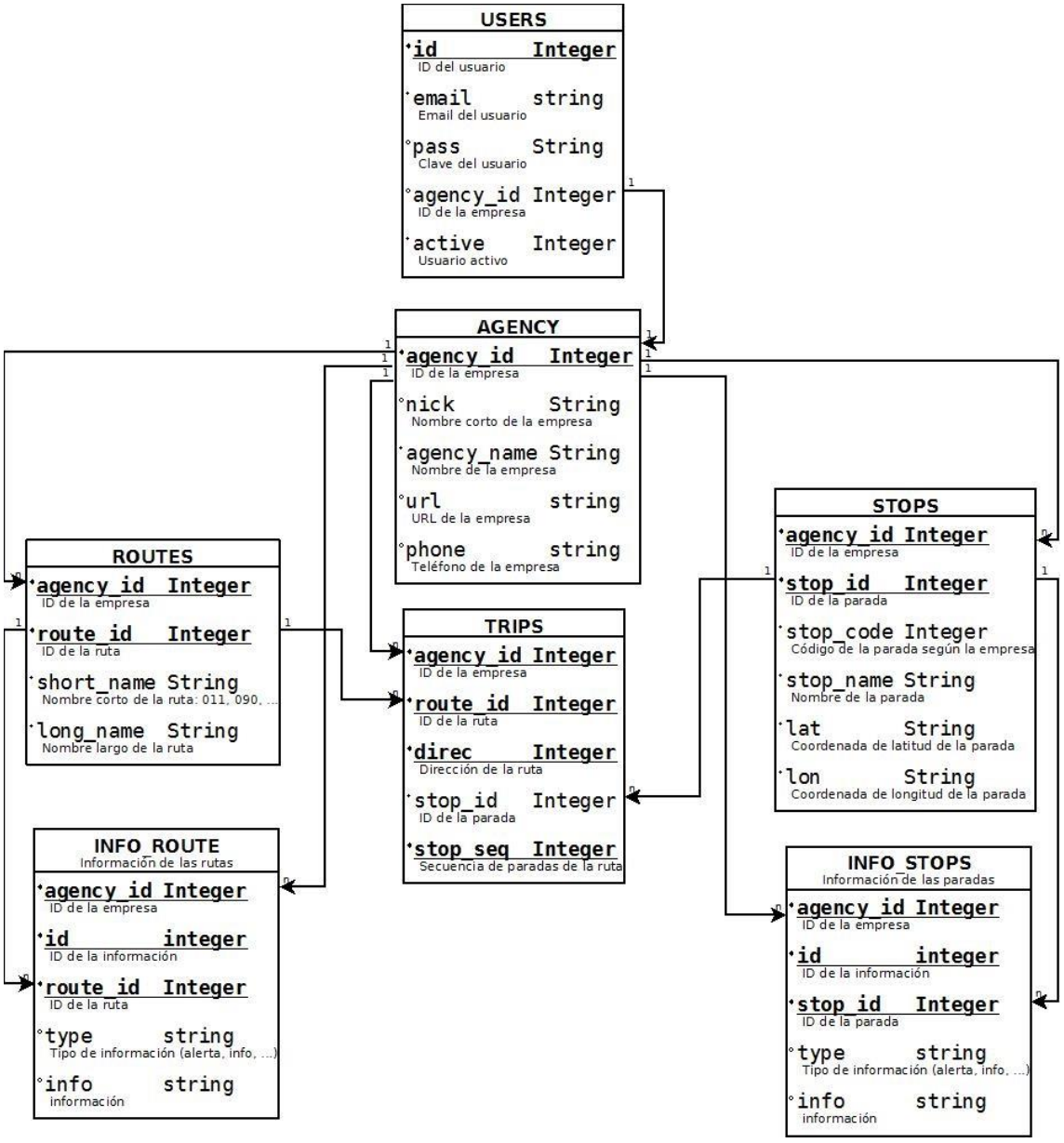


Ilustración 55. Esquema de la base de datos con usuarios

De momento contamos con un diseño bastante sencillo con una ventana de login para las empresas. Actualmente no se permite el registro por medidas de seguridad, si una empresa quiere acceder a su plataforma deberá ponerse en contacto a través del correo que se facilitará en la web para obtener su usuario y clave.

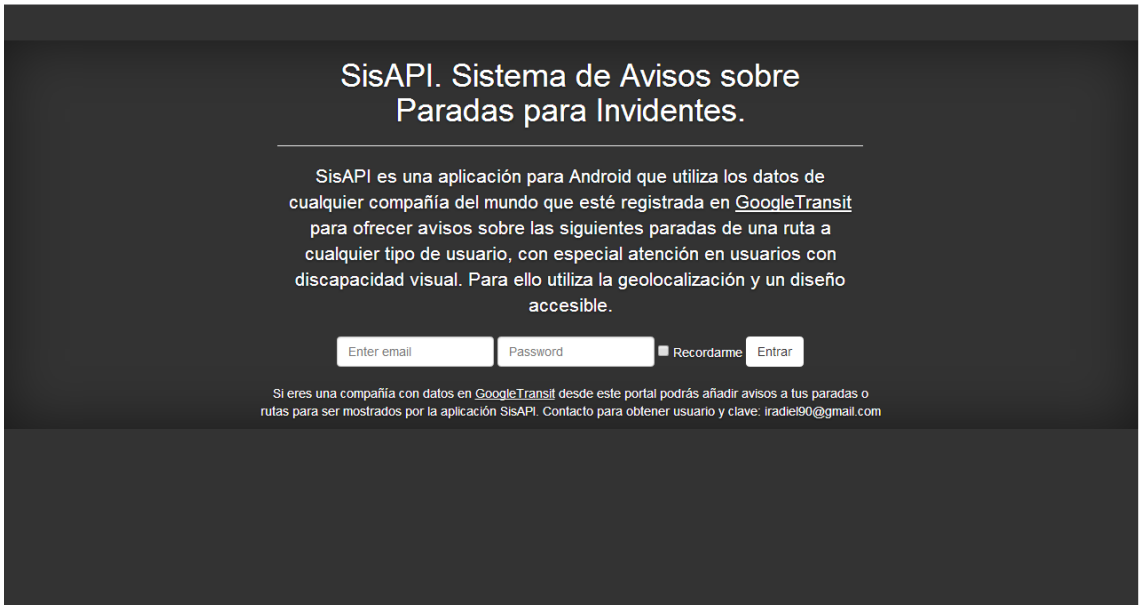


Ilustración 56. Portal web. Inicio/Login

Una vez que el usuario se ha identificado pasará a una vista de los datos que están actualmente en la base de datos del servidor del proyecto asociados a su compañía. Con un menú lateral en el que podrá acceder a la información de las diferentes tablas.

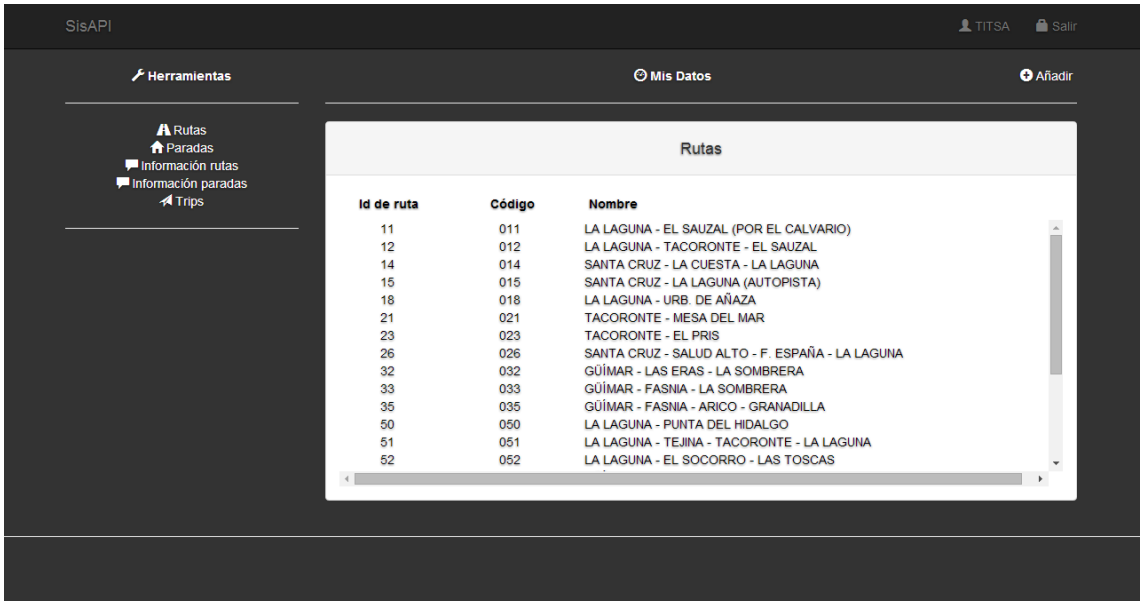


Ilustración 57. Portal web. Vista de los datos

Desde aquí las compañías podrán añadir los avisos que deseen para las rutas o paradas, esta información se alojará en la base de datos del servidor y estará disponible para los usuarios que utilicen la aplicación móvil.

Capítulo 9. Conclusiones y Trabajos Futuros

9.1 Conclusiones

Durante el desarrollo del proyecto se ha investigado sobre tecnologías como Phonegap y algunos plugins útiles para trabajar con bases de datos y convertir texto en voz. Aunque al final se tuvo que dejar de utilizar Phonegap y pasar al diseño en Android nativo, la experiencia y conocimientos adquiridos sobre esta tecnología podrían ser de utilidad en un futuro. En el proyecto se utilizaron funciones y métodos de Android para gestionar bases de datos, acceder a la localización del dispositivo, crear y gestionar servicios asíncronos y se estudió como realizar un diseño usable y accesible para esta plataforma.

También se elaboró una aplicación servidor en lenguaje PHP utilizando el framework Slim para facilitar la creación de una API REST. Así mismo, se tuvo que realizar una tarea de investigación para comprender el funcionamiento de la arquitectura REST y HATEOAS para diseñar una aplicación siendo lo más fiel posible a esta arquitectura, pasando por los niveles de calidad definidos por el modelo Richardson Maturity Model.

Por último, para la implementación del portal web, se utilizaron librerías como Backbone.js, Underscore.js y Require.js para implementar un modelo-vista-controlador en una web, además del framework Bootstrap y la librería JQuery para facilitar el diseño de la misma.

La aplicación móvil ha sido desarrollada y terminada de manera adecuada y satisfactoria. La lista de requisitos funcionales especificados ha sido cumplida en su totalidad. No obstante, algún aspecto que quizás se puede mejorar un poco más es la interfaz gráfica.

Según la especificación de requisitos y el modelo de casos de uso descritos en el capítulo primero, la aplicación permite al usuario final:

- Ser avisado por voz y texto de la próxima parada del recorrido.
- Obtener información adicional, de haberla, sobre las rutas o paradas.

9.2 Trabajos futuros

Como trabajos futuros se plantea:

- Mejorar la plataforma web para que las compañías puedan gestionar todos sus datos desde ella, no solo los avisos como se ofrece actualmente.
- Mejorar el servicio de actualización automática de la base de datos del dispositivo móvil para que se realice únicamente cuando ha habido cambios en los datos del servidor.
- Implementación de la aplicación móvil para las distintas plataformas.
- Añadir nuevos servicios a la aplicación:
    - ¿Dónde estoy?
    - ¿Cuánto queda para la próxima parada?
    - Servicio guiado a pie hasta la parada y hasta destino después de bajarse del transporte.
    - ¿Cómo llegar a? Para que la aplicación indique que líneas debe coger para llegar a un sitio determinado.
    - Ir a casa.

Capítulo 10. Summary and Conclusions

10.1 Conclusions

During the development of the project has researched technologies like Phonegap and some useful for working with databases and convert text to speech plugins. Although in the end had to stop using Phonegap and move to Android native design, experience and knowledge about this technology could be useful in the future. In the project, Android functions and methods were used to manage databases, access the device's location, building and managing asynchronous services and studied how to make a usable and accessible design for this platform.

A server application in PHP was also developed using the Slim framework to facilitate the creation of a REST API. Likewise, to design an application to be as faithful as possible to the REST and HATEOAS architectures, a research task was performed to understand the functionality of these architectures. The application had to pass quality standards defined by the Richardson Maturity Model.

Finally, for the implementation of the web, Backbone, Underscore and Require libraries were used to implement a model-view-controller on the website. In addition, Bootstrap framework and JQuery library were used to facilitate the design.

The mobile application has been developed and completed properly and and satisfactorily. The list of specified functional requirements have been fulfilled. However, one aspect that might be improved a little more is the graphical interface.

According to the requirements specification and the use cases models described in the first chapter, the application allows the end user to:

- Be notified by voice and text of the next bus stop.
- Obtain additional information, if any, of the routes and stops.

10.1 Future works

As future work is proposed:

- Improve the web platform for companies to manage all their data from it.
- Improve database automatic update service to perform only when there have been changes to the server data.
- Implementation of mobile application for all the platforms.
- Add new services to the application:
  - Where am I?
  - How much is left to the next stop?
  - Walk guided service to the bus stop and to the destination after exiting the bus.
  - How to get to? The application indicates which lines should take to reach a particular site.
  - Go home.

Capítulo 11. Presupuesto

11.1 Horas dedicadas

Se estima que se han dedicado unas 271 horas a la elaboración del trabajo de fin de grado. Las cuáles se pueden desglosar de la siguiente manera:

Tarea	Horas	Precio por hora	Presupuesto
ANÁLISIS	41	50 €	2050 €
DISEÑO	68	50 €	3400 €
PROGRAMACIÓN	122	36 €	4392 €
PRUEBAS	40	30 €	1200 €

Tabla 8. Presupuesto por horas dedicadas

11.2 Herramientas utilizadas

Por un lado tenemos el presupuesto invertido en las licencias de frameworks y librerías utilizadas en la elaboración del trabajo de fin de grado.

Frameworks y Librerías	Licencia	Presupuesto
SLIM	GRATUITA	0 €
BOOTSTRAP	GRATUITA	0 €
BACKBONE.JS	GRATUITA	0 €
UNDERSCORE.JS	GRATUITA	0 €
REQUIRE.JS	GRATUITA	0 €
JQUERY	GRATUITA	0 €

Tabla 9. Presupuesto por licencias de frameworks y librerías

Por otro tenemos el precio de las licencias del software empleado para diseñar y elaborar el proyecto.

Software	Descripción	Licencia	Presupuesto
EYES-FREE KEYBOARD	Comprobar accesibilidad navegacional de aplicaciones Android	GRATUITA	0 €
SUBLIME-TEXT	Editor de texto	GRATUITA	0 €
APTANA STUDIO	Entorno de desarrollo, utilizado para crear la aplicación servidor	GRATUITA	0 €
ECLIPSE	Entorno de desarrollo, utilizado para crear la aplicación Android	GRATUITA	0 €
XAMPP	Servidor independiente para realizar pruebas en local	GRATUITA	0 €
FILEZILLA	Cliente FTP	GRATUITA	0 €

Tabla 10. Presupuesto por licencias de software

11.3 Equipo necesario

En este apartado mostramos el presupuesto necesario por el equipo utilizado durante la elaboración del trabajo de fin de grado.

Equipo	Presupuesto
SERVIDOR	1200 €/AÑO

Tabla 11. Presupuesto por equipo utilizado

11.4 Total

El presupuesto total del trabajo de fin de grado asciende a:

TOTAL
11042 € + 1200 € anuales

Tabla 12. Presupuesto total

Bibliografía

Adobe-Systems. (s.f.). *Phonegap*. Recuperado el diciembre de 2013, de http://phonegap.com/

Alcalde, A. (s.f.). *PROGRAMACIÓN ANDROID: USANDO CURSORES*. Recuperado el febrero de 2014, de http://elbauldelprogramador.com/programacion-android-usando-cursores/

Alonsojpd. (s.f.). *AJPD soft*. Recuperado el diciembre de 2013, de http://www.ajpdsoft.com/modules.php?name=News&file=article&sid=660

Android Developers. (s.f.). *Implementing Accessibility*. Recuperado el junio de 2014, de http://developer.android.com/intl/es/training/accessibility/index.html

Android Developers. (s.f.). *LocationManager*. Recuperado el marzo de 2014, de http://developer.android.com/intl/es/reference/android/location/LocationManager.html

Android Developers. (s.f.). *SQLiteOpenHelper*. Recuperado el febrero de 2014, de http://developer.android.com/intl/es/reference/android/database/sqlite/SQLiteOpenHelper.html

androidmyway. (s.f.). *Proximity Alert In Android*. Recuperado el abril de 2014, de http://androidmyway.wordpress.com/2012/08/07/proximity-alert-in-android/

Backbonejs.org. (s.f.). *Backbone.js*. Recuperado el junio de 2014, de http://backbonejs.org/

Brody, C. (s.f.). *PGSQLitePlugin*. Recuperado el diciembre de 2013, de https://github.com/brodysoft/Cordova-SQLitePlugin

Charlie don't code. (s.f.). *HATEOAS: ¿qué y por qué?* Recuperado el febrero de 2014, de http://charliedontcode.com/rest/2012/09/27/rest-apis-hateoas.html

Chung, A. (s.f.). *Require.js*. Recuperado el junio de 2014, de http://requirejs.org/

Eyes-Free Project. (s.f.). *Eyes-Free Keyboard*. Recuperado el junio de 2014, de https://play.google.com/store/apps/details?id=com.googlecode.eyesfree.inputmethod.latin&hl=es

Fowler, M. (s.f.). *Richardson Maturity Model*. Recuperado el febrero de 2014, de http://martinfowler.com/articles/richardsonMaturityModel.html

Google Inc. (s.f.). *GoogleTransit*. Recuperado el diciembre de 2013, de http://maps.google.com/intl/es/landing/transit/

Google Inc. (s.f.). *GoogleTransitDataFeed*. Recuperado el diciembre de 2013, de https://code.google.com/p/googletransitdatafeed/wiki/PublicFeeds

Hidalgo Reina, J. (s.f.). *Para crear aplicaciones moviles accesibles, sentido común*. Recuperado el junio de 2014, de http://www.juanhidalgoreina.com/2011/12/aplicaciones-para-moviles-accesibles/

Informática y Accesibilidad para Todos. (s.f.). *Cómo Desarrollar Apps Móviles Accesibles para Dispositivos Android y No Fracasar en el Intento*. Recuperado el junio de 2014, de http://infoaccess4all.wordpress.com/2013/12/24/como-desarrollar-apps-moviles-accesibles-para-dispositivos-android-y-no-fracasar-en-el-intento/

Intel. (s.f.). *App-framework*. Recuperado el diciembre de 2013, de http://app-framework-software.intel.com/

Inusual. (s.f.). *REST vs SOAP al servicio de la web*. Recuperado el febrero de 2014, de http://inusual.com/comunicacion/rest-vs-soap-al-servicio-de-la-web/

JQuery Foundation. (s.f.). *JQuery*. Recuperado el junio de 2014, de http://jquery.com/

Lee, W.-M. (2011). Desarrollar servicios Android. En *Android 4 Desarrollo de aplicaciones.* Anaya.

Lee, W.-M. (2011). Persistencia de datos. En *Android 4 Desarrollo de aplicaciones.* Anaya.

Lee, W.-M. (2011). Servicios de localización. En *Android 4 Desarrollo de aplicaciones.* Anaya.

Macdonst. (s.f.). *TTS Plugin*. Recuperado el diciembre de 2013, de https://gist.github.com/macdonst/962784

Marín, A. (s.f.). *Tutorial Backbone.js*. Recuperado el junio de 2014, de http://alfonsomarin.com/desarrollo-web/tutoriales/backbonejs

Marqués, A. (s.f.). *Conceptos sobre APIs REST*. Recuperado el enero de 2014, de http://asiermarques.com/2013/conceptos-sobre-apis-rest/

Muñoz, N. (s.f.). *Slim, mini framework REST para PHP*. Recuperado el enero de 2014, de http://www.elserver.com/slim-mini-framework-rest-para-php/

Otto, M., & Thornton, J. (s.f.). *Bootstrap*. Recuperado el junio de 2014, de http://getbootstrap.com/

PhoneGap discussion group. (s.f.). *PhoneGap discussion group*. Recuperado el enero de 2014, de http://comments.gmane.org/gmane.comp.handhelds.phonegap/54875

PHP Group. (s.f.). *MySQLi PHP*. Recuperado el enero de 2014, de https://php.net/manual/es/book.mysqli.php

Salvador Gómez, O. (s.f.). *Bases de Datos en Android (I): Primeros pasos*. Recuperado el febrero de 2014, de http://www.sgoliver.net/blog/?p=1611

Salvador Gómez, O. (s.f.). *Localización geográfica en Android (II)*. Recuperado el marzo de 2014, de http://www.sgoliver.net/blog/?p=1932

StartCapps. (s.f.). *Web Services – REST vs SOAP*. Recuperado el febrero de 2014, de http://www.startcapps.com/blog/web-services-rest-vs-soap/

Statamic. (s.f.). *Slim Framework*. Recuperado el enero de 2014, de http://www.slimframework.com/

Statamic. (s.f.). *Slim Framework Documentation*. Recuperado el febrero de 2014, de http://docs.slimframework.com/

Tamada, R. (s.f.). *Android JSON Parsing Tutorial*. Recuperado el febrero de 2014, de http://www.androidhive.info/2012/01/android-json-parsing-tutorial/

Tamada, R. (s.f.). *How to create REST API for Android app using PHP, Slim and MySQL*. Recuperado el febrero de 2014, de http://www.androidhive.info/2014/01/how-to-create-rest-api-for-android-app-using-php-slim-and-mysql-day-12-2/

Underscorejs.org. (s.f.). *Underscore.js*. Recuperado el junio de 2014, de http://underscorejs.org/

Wikipedia. (s.f.). *CRUD*. Recuperado el enero de 2014, de http://es.wikipedia.org/wiki/CRUD

Wikipedia. (s.f.). *Hipermedia*. Recuperado el marzo de 2014, de http://es.wikipedia.org/wiki/Hipermedia

Wikipedia. (s.f.). *Modelo–vista–controlador*. Recuperado el junio de 2014, de http://es.wikipedia.org/wiki/Modelo_Vista_Controlador

Wikipedia. (s.f.). *Servicio web*. Recuperado el enero de 2014, de http://es.wikipedia.org/wiki/Servicio_web

Wikipedia. (s.f.). *SOAP*. Recuperado el enero de 2014, de http://es.wikipedia.org/wiki/Simple_Object_Access_Protocol

Wikipedia. (s.f.). *WSDL*. Recuperado el enero de 2014, de http://es.wikipedia.org/wiki/WSDL