ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Universidad Politécnica de Cartagena

Trabajo~~Proyecto~~ Fin de Grado

Diseño e implementación de un sistema de publicación de rutas para compartir vehículo ~~*una aplicación cliente servidor para dispositivos móviles Android con el api de Google Maps*~~

etsit escuela técnica superior de ingeniería de telecomunicación

AUTOR: José Antonio Díaz Mateo
DIRECTOR: Juan Pedro Muñoz Gea

Septiembre / 2015

Autor	José Antonio Díaz Mateo
E-mail del Autor	joseantonio.diaz85@gmail.com
Director	Juan Pedro Muñoz Gea
E-mail del Director	juanp.gea@upct.es
Título del PFC	Diseño e implementación de un sistema de publicación de rutes para compartir vehículo ~~*una aplicación para dispositivos móviles Android con el API de Google Maps.*~~
Descriptores	Google Maps, Geolocalización, Servlets, Compartir vehiculo.

Resumen

Los smartphones, tablets y otros dispositivos de tamaño relativamente pequeño, han irrumpido desde hace unos años en los hogares convirtiéndose en herramientas de uso diario donde se satisfacen diversas necesidades de los usuarios, llegando a parecerse cada vez más a lo que sería un microordenador. Podemos encontrar una diversa variedad de aplicaciones para múltiples propósitos, como deporte, quemar calorías, sistemas de localización por GPS, etc. Esto hace de estos dispositivos un instrumento indispensable para el día a día.

Cada vez es más habitual que la gente tenga la necesidad de compartir los gastos derivados del transporte.~~,~~ Ésta es la principal motivación de este trabajo cuyo objetivo fundamental es la realización de una aplicación que satisfaga estas necesidades, creando un sistema que permita facilitar la comunicación entre los usuarios donde se puedan publicar itinerarios que a los que realizan de forma habitual los conductores, permitiendo así la suscripción de los pasajeros en función de las plazas disponibles que destina los conductores.

Titulación	Grado en Ingeniería Telemática~~Ingeniería Técnica de Telecomunicación, especialidad Telemática~~
Intensificación	
Departamento	Tecnologías de la Información y las Comunicaciones
Fecha de Presentación	Septiembre - 2015

Tabla de Contenidos

Índice de Figuras

1.Primer Capítulo
Consideraciones Iniciales

1.1. Introducción

Los smartphones, tablets y otros dispositivos de tamaño relativamente pequeño, han irrumpido desde hace unos años en los hogares convirtiéndose en herramientas de uso diario donde se satisfacen diversas necesidades de los usuarios, llegando a parecerse cada vez más a lo que sería un microordenador. Podemos encontrar una diversa variedad de aplicaciones para múltiples propósitos, como deporte, quemar calorías, sistemas de localización por GPS, etc. Esto hace de estos dispositivos un instrumento indispensable para el día a día.

Cada vez es más habitual que la gente tenga la necesidad de compartir los gastos derivados del transporte, creando un sistema que permita facilitar la comunicación entre los usuarios donde se puedan publicar itinerarios que a los que realizan de forma habitual los conductores, permitiendo así la suscripción de los pasajeros en función de las plazas disponibles que destina los conductores.

1.2. Objetivos

El objetivo principal de este proyecto es desarrollar una aplicación cliente – servidor que permita el intercambio y publicación en un sistema en la nube de información sobre itinerarios que se realizan en coche de forma habitual con el fin de minimizar el gasto derivado del uso de los vehículos personales. Para ello se llevarán a cabo las siguientes tareas:

Consideraciones Iniciales

Primeramente se estudiarán las diferentes tecnologías disponibles para el desarrollo de aplicaciones móviles, así como las diferentes opciones disponibles para el desarrollo de un sistema de comunicación entre la aplicación móvil y una base de datos en la nube.

Desarrollo Teórico

En este apartado, nos centraremos en la aplicación cliente – servidor. Se describirán con detalle los aspectos teóricos del desarrollo de la misma a partir de una serie de esquemas que mostrarán un punto de vista genérico de la aplicación a desarrollar, así como cada una de las partes por separado (cliente y servidor), ofreciendo una mejor comprensión de lo que se pretende realizar.

Implementación

En la implementación se describirán las herramientas utilizadas, además de las posibles opciones que se han descartado, tanto en términos de lenguajes de programación como en posibles herramientas externas. Además, se describirá con detalle cada uno de los módulos que componen la aplicación, atendiendo a los criterios teóricos anteriormente definidos.

Conclusiones

Aquí se exponen las conclusiones generales del proyecto, resumiendo brevemente todo lo que se ha hecho durante el mismo. Además, también se trata un pequeño apartado en el que se proponen posibles líneas futuras por las que puede encaminarse el proyecto.

1.3. Análisis general de las diferentes tecnologías y consideraciones para el desarrollo.

En primer lugar, se deben analizar una serie de conceptos básicos antes de empezar a describir las herramientas que se utilizarán para desarrollar nuestra aplicación móvil. Por ejemplo, se plantean cuestiones como las siguientes:

- ¿Cuál es el sistema más adecuado para el desarrollo de la aplicación cliente?
- ¿Qué tecnología es la más adecuada para el desarrollo del sistema servidor?
- ¿Cuál es la mejor forma de realizar el intercambio de información de itinerarios entre los dos sistemas?

1.3.1. ¿Cuál es el sistema más adecuado para el desarrollo de la aplicación cliente?

Nos encontramos ante un sistema cliente servidor donde el cliente es una plataforma móvil. En este caso disponemos de una gran variedad de sistemas operativos, pero para reducir las opciones, nos limitaremos a aquellos que tiene una mayor cuota de mercado. Siguiendo este criterio, existen tres sistemas operativos bajo los que desarrollar nuestra aplicación para móviles:

- IOS de Apple.
- Windows Phone de Microsoft
- Android de Google

Los tres sistemas son buenas opciones, sin embargo, atendiendo a la presencia de los diferentes sistemas operativos presentes en el mercado, Android e IOS, son los dos más utilizados por los usuarios de plataformas móviles, quedando definitivamente descartado el sistema de Microsoft.

Finalmente, de las opciones restantes, se ha elegido Android debido principalmente a su menor coste en cuanto a terminales móviles se refiere y a su gran comunidad que ofrece un enorme apoyo para el desarrollo de aplicaciones así como su mayor presencia en el mercado.

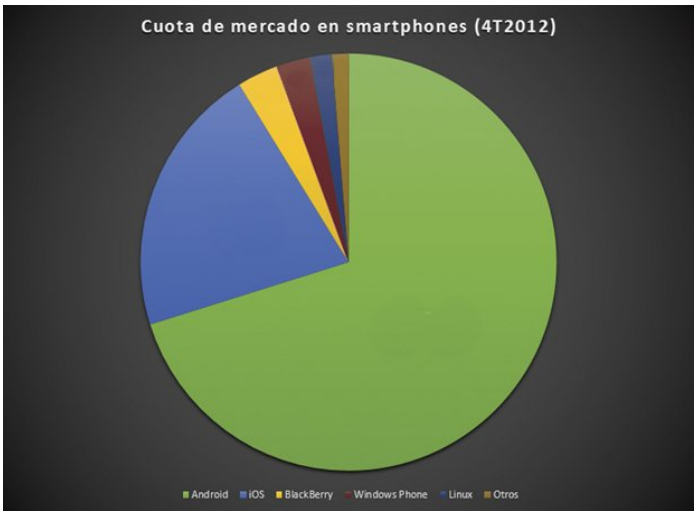

Figura 1.3.1. Cuota de mercado de los distintos sistemas operativos móviles.

En la figura 1.3.1 se muestra un gráfico de la cuota de mercado de los diferentes sistemas operativos. Como podemos observar, Android e IOS ~~,~~ocupan la mayor parte del mercado móvil actualmente, dejando a sistemas como Windows Phone o Black Berry OS en una posición de muy baja importancia, por lo que los argumentos anteriormente expuestos para la selección de la plataforma de desarrollo son los adecuados.

1.3.1.1. Versión de A~~a~~ndroid

Ahora que ya sabemos en qué sistema operativo móvil se va a trabajar debemos elegir la versión del mismo a partir de la cuál será compatible nuestra aplicación. Cuando se comenzó con el desarrollo de esta aplicación, Android se encontraba en su versión *Ice Cream Sandwich (4.0)*, por lo que aún era muy habitual encontrar terminales con la versión *Gingerbread (2.3.x)*. ~~,~~Sin embargo,~~pero~~ debido a la gran evolución del sistema de Google en con su reciente *Lolipop (5.x)*, la presencia de *Gingerbread* se ha visto desplazada por versiones superiores como *Ice Cream Sandwich* o *Jelly Bean (4.1.x)*, por lo que finalmente, se ha decidido emplear una versión mínima de Android *Ice Cream Sandwich (4.0)* para obtener acceso a herramientas de diseño más avanzadas en el entorno gráfico.

1.3.1.2. Adaptación del diseño de la interfaz a múltiples dispositivos

Otro problema subyacente al desarrollo en plataformas móviles es la heterogeneidad en cuanto a tamaños y resoluciones de pantalla existentes en el mercado, especialmente en Android que hay tanta diversidad de terminales y de especificaciones muy diferentes.

Para solventar este problema, Android dispone de una unidad de medida denominada *DPI* (Pixel Per Inch) que permite adaptar nuestra interfaz a cualquier pantalla independientemente de la resolución.

1.3.2. ¿Qué tecnología es la más adecuada para el desarrollo del sistema servidor?

Una vez que ya hemos seleccionado una plataforma de desarrollo para nuestra aplicación cliente, debemos comprobar las diferentes opciones que tenemos para desarrollar un sistema servidor que nos permita comunicarnos con una base de datos para el almacenamiento y consulta de la información sobre los itinerarios de los conductores y la suscripción de los pasajeros a los mismos en función de las plazas disponibles. Para ello se empleará un servidor web apache con accesos a una base de datos MySQL que se desarrollará en Java utilizando Servlets, descartando así, el uso de PHP, ya que por afinidad con Android, que también utiliza Java como lenguaje de programación, el tratamiento de las comunicaciones se realiza de forma análoga en ambos sistemas (Cliente y Servidor), facilitando en gran medida el desarrollo.

1.3.3. ¿Cuál es la mejor forma de realizar el intercambio de itinerarios entre los dos sistemas?

Otra de las cuestiones que se plantean cuando desarrollamos una aplicación cliente servidor es la forma en que los datos fluyen entre ambas. En nuestro caso no se trata de una aplicación que requiera de tiempo real para la transferencia de datos, ya que no se tratan flujos de datos muy pesados, como por ejemplo, contenido multimedia, por lo que se ha optado por un sistema de comunicación asíncrona.

Por otro lado, hay que tener en cuenta, que los dispositivos móviles son muy variados en especificaciones técnicas, por lo que si el sistema de intercambio de datos requiere de muchos recursos, estaríamos

limitando el número de dispositivos a los que podría llegar nuestra aplicación, por lo tanto se debe elegir un sistema de intercambio de datos ligero y que consuma pocos recursos. Para ello, se empleará el formato *JSON*, que nos permite intercambiar información en modo texto, codificando los datos propios de un lenguaje de programación (objetos) a una cadena de texto, lo que aligera mucho las comunicaciones en comparación con una transmisión basada en datos puros.

2.Segundo Capítulo
Desarrollo teórico

2.1. Introducción

Hasta ahora se han explicado las bases sobre las que se sustentará este proyecto, tales como el formato *JSON* para el intercambio de datos y las diferentes tecnologías para el desarrollo de la aplicación móvil.

Como ya se citó en la introducción de este documento, se pretende desarrollar una aplicación basada en tecnología móvil para la publicación de itinerarios a los que se podrán inscribir los pasajeros en función de las plazas ofrecidas por los conductores que los realizan. A continuación se explicarán los diferentes elementos que componen el sistema desde un punto de vista general, describiendo las funciones básicas.

2.2. Desarrollo de una aplicación con tecnología de localización basada en Google Maps

La idea se basa en desarrollar una aplicación cliente – servidor que permita la comunicación entre una base de datos y un dispositivo móvil para la publicación de itinerarios que los conductores de vehículos realizan de forma habitual, ofreciendo un número de plazas determinadas, pudiendo suscribirse a esos itinerarios cualquier usuario como pasajero.

La aplicación consta de los siguientes elementos:

Servidor. Se encarga de recibir las peticiones de los usuarios para la publicación o consulta de itinerarios, así como, la suscripción de pasajeros a los itinerarios ya publicados. Además, se encarga de gestionar las consultas a la base de datos donde se aloja la información sobre usuarios, itinerarios, etc.

Sistema de Bases de Datos. Se encarga de almacenar toda la información de los usuarios registrados, los itinerarios, los pasajeros suscritos etc.

Cliente. Esta es la propia aplicación móvil. Esta aplicación tiene como cometido, la interacción con el usuario y la solicitud de las peticiones indicadas por éste al sistema servidor, mostrando los resultados de las mismas en el dispositivo.

En la Figura 2.1 se muestra un esquema que ilustra lo explicado de forma gráfica.

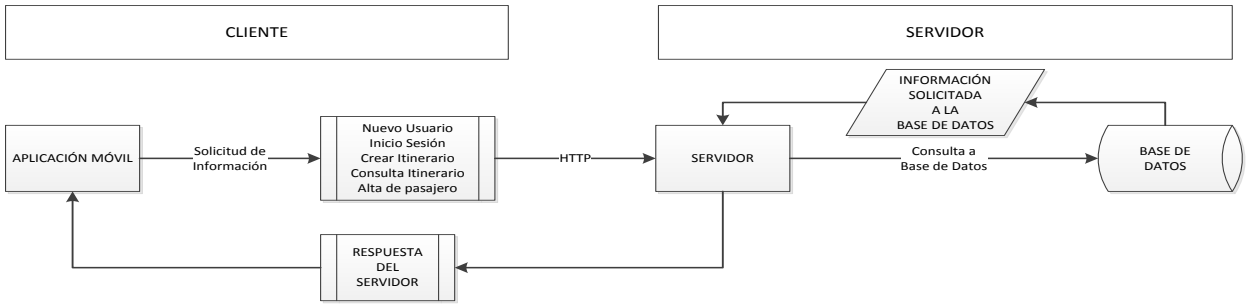

Figura 2.1.Esquema general de la aplicación

Otro punto a tratar a la hora de desarrollar una *aplicación de comunicaciones* como ésta, es saber de qué tipo de aplicación estamos hablando, es decir, se debe conocer lo que *requiere* nuestra aplicación para utilizar unos protocolos u otros a la hora de transmitir la información. Si se trata de una aplicación donde la fiabilidad sea imprescindible en la transmisión, donde se deba asegurar un orden correcto en la llegada de los paquetes se utilizará TCP como protocolo de transporte. En cambio, si se trata de una aplicación donde tiene prioridad la velocidad de transmisión frente a lo demás, se utilizará UDP.

En nuestro caso se pretende desarrollar una aplicación que transmite información sobre itinerarios tanto para almacenamiento como para consulta, por lo que estamos hablando de una aplicación donde el tiempo no es prioritario, así que para las partes de este proyecto donde se requiera la transmisión de cualquier tipo de información a través de la red, utilizaremos el protocolo HTTP que utiliza TCP como protocolo de transporte, puesto que no necesitamos una velocidad excesiva ni mantener un flujo constante en la comunicación, sino fiabilidad. A continuación se explicará con más detalle cada una de las partes de la aplicación tanto en el sistema servidor como en la aplicación móvil.

2.3. Esquema del sistema servidor

Ahora que ya se tiene una visión general sobre la aplicación a desarrollar, se puede tomar como base para describir con más detalle cada una de sus partes, teniendo así una referencia más sólida y concreta, pero sin dar detalles sobre implementación o herramientas utilizadas para su desarrollo ya que esto se abordará en futuros apartados del presente documento.

Cada una de las partes que componen el servidor resuelve un problema asociado a ella, por lo tanto, a continuación se expondrán qué problemas han ido surgiendo a la hora de plantear el desarrollo de la aplicación y cómo se han solventado con cada una de las partes de las que se compone.

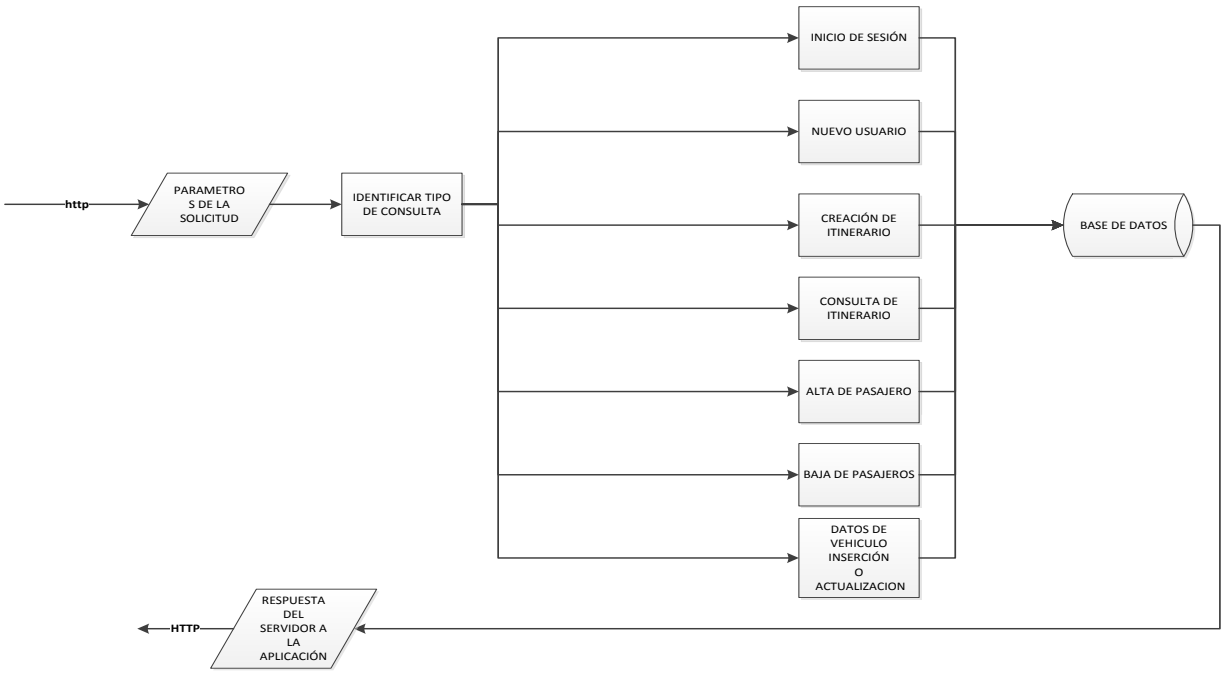

Figura 2.2 – Esquema del servidor.

El servidor se encarga de extraer o almacenar la información necesaria para la publicación o consulta de los itinerarios de los pasajeros, así como los datos de los vehículos de los usuarios, compuesto principalmente por tres módulos:

Sistema identificador. Esta parte del servidor se encarga de determinar qué operación se está solicitando, enviando la información contenida en la solicitud del cliente al sistema procesador en función del tipo de operación a realizar.

S~~Es s~~istema p~~P~~rocesador. Una vez que se ha identificado el tipo de operación a realizar, se tratan los datos de la solicitud HTTP ejecutando la operación indicada por el sistema identificador que puede ser por ejemplo, inicio de sesión, registro de un nuevo usuario, alta de un pasajero en un itinerario, etc.

Base de d~~D~~atos. Aquí se almacena toda la información asociada a los itinerarios publicados por los usuarios, así como los propios usuarios dados de alta en el sistema y los pasajeros suscritos a las diferentes rutas publicadas. Esta información es explotada por el servidor en base a los datos recibidos en la solicitud que definen el tipo de operación a realizar sobre ella, previamente determinada por el sistema identificador.

2.4. Esquema de la aplicación móvil

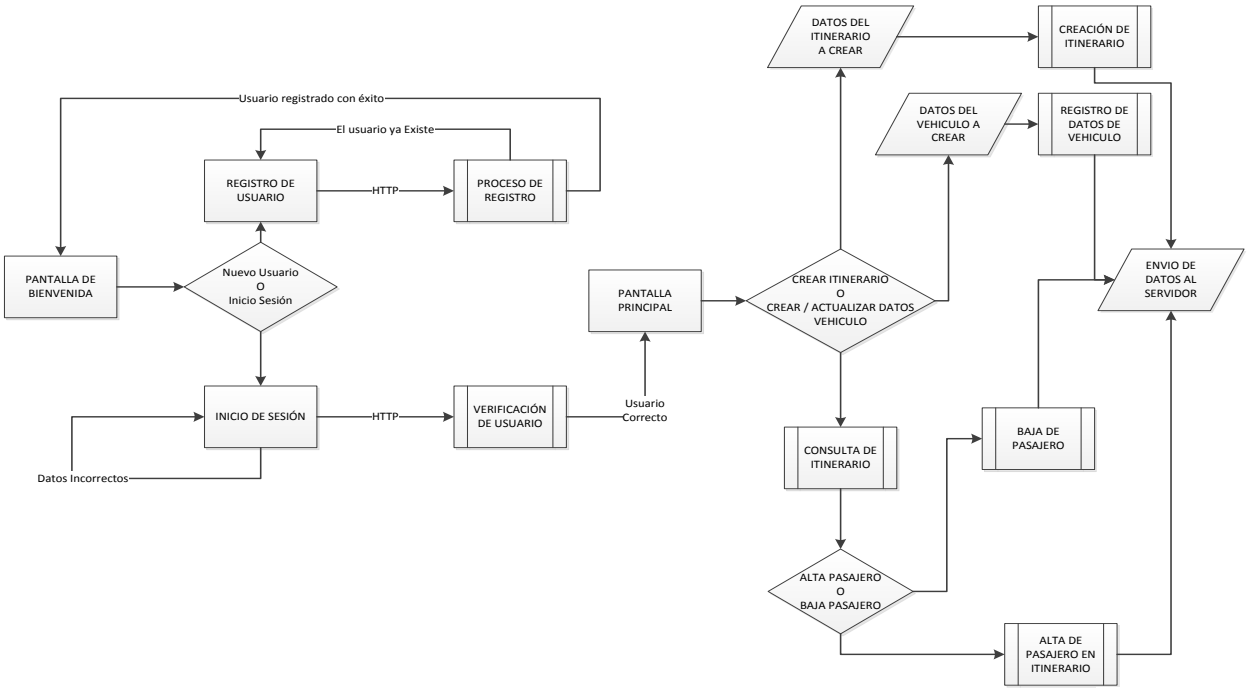

Figura 2.3 .Esquema de la aplicación

Una vez que ya se han definido las operaciones básicas y los módulos que componen el servidor es hora de ver el funcionamiento básico de la aplicación móvil.

Lo primero que se debe tener en cuenta en el sistema móvil es que sin un usuario no podremos realizar ninguna operación de consulta, publicación o suscripción, por lo que lo primero es registrar un nuevo usuario en el sistema, realizando la correspondiente solicitud de registro de usuario al servidor con los datos del mismo.

Una vez que ya disponemos de usuario en el sistema lo primero que debemos hacer es iniciar sesión con dicho usuario, lo que nos dará acceso total al sistema de publicación de itinerarios y al de altas y bajas de pasajeros para poder suscribirnos como pasajeros en itinerarios ya publicados. Además, también podemos crear los datos correspondientes al vehículo que conducimos en caso de que realicemos rutas como conductor.

Por último, podemos consultar las rutas ya publicadas y darnos de alta en cualquiera de ellas o darnos de baja si ya estamos suscritos en función de las plazas disponibles. El sistema de suscripción a rutas sólo estará accesible desde la consulta de itinerarios.

3.Tercer Capítulo
Implementación

3.1. Introducción

Este capítulo trata de abordar los diferentes problemas que se expusieron en el anterior, tratando las diferentes opciones disponibles para resolver dichos problemas y escogiendo la que es más acertada para lo que se debe implementar o la que más ventajas ofrezca en cuanto a determinados requisitos que se tienen en cuenta en la aplicación, como pueden ser, manejo de la memoria, disponibilidad de herramientas de desarrollo o documentación, etc.

Por otra parte, una vez que se haya decidido sobre qué herramientas son las más adecuadas para el desarrollo de nuestra aplicación, se detallarán los diferentes módulos, atendiendo a las herramientas seleccionadas.

3.2. Lenguajes y herramientas

Ahora que ya conocemos qué tipo de aplicación queremos desarrollar y cuáles son los problemas que ésta nos propone solucionar, debemos escoger las diferentes herramientas con las que llevar a cabo su implementación.

La primera cuestión que nos debemos plantear, tal y como ya se indicó en el apartado de *consideraciones iniciales*, es el sistema operativo móvil sobre el que se va a trabajar. Podemos utilizar un sistema operativo que disponga de herramientas para la programación gratuitas y sea de código abierto como Android o por el contrario, sistemas como IOS de Apple o Windows Phone de Microsoft que son de código cerrado y dispone de una serie de herramientas de pago para el desarrollo de aplicaciones.

En nuestro caso, para la realización de este proyecto, se ha optado por el sistema Android, debido principalmente a su amplia documentación y su gran presencia en el mercado, lo que nos permite abarcar una gran cantidad de dispositivos.

Por otra parte, Android ofrece un acceso directo a las librerías de la API de Google Maps, que nos servirán para la presentación de los itinerarios en la interfaz de usuario mediante el dibujado sobre mapas en diferentes perspectivas.

Finalmente, dentro de la API de Google disponemos de la Google Direction API, que nos permite obtener los datos de ubicación e indicaciones de los itinerarios que representaremos sobre los mapas, basados en diferentes parámetros como los peajes, la región, el origen, el destino, etc.

3.2.1.Java (Sun Microsystems)

Java es un lenguaje de programación orientado a objetos que nos permite modelar el mundo real a través de unas entidades que dan nombre a su paradigma, los objetos. Los objetos nos pueden servir para modelar el mundo real de forma muy representativa, como por ejemplo, un coche, una casa o una calle.
La entidad de Java que representa a un objeto es lo que se conoce como clase. Las clases son la base de todo programa Java, ya que representan con unas variables de estado llamadas variables ejemplares o de instancia, y unas funciones llamadas métodos las características y comportamiento de un objeto.

Estas líneas vendrían a ser un pequeño resumen de lo que principalmente nos permite Java como lenguaje, pero atendiendo a cuestiones más prácticas, lo que nos interesa saber, es por qué se ha elegido Java en lugar de otras opciones como podría ser C.

Las principales cuestiones por las que se suele elegir Java en lugar de otros lenguajes como C, suelen ser la gestión automática de memoria, la mejor abstracción pudiendo ver los diferentes módulos de la aplicación como objetos en lugar de como un conjunto de ficheros o la gran cantidad de documentación con la que cuenta su api. De hecho, además de lo anteriormente mencionado, el motivo principal por el que se ha escogido Java para este proyecto son las herramientas de desarrollo. Entornos de desarrollo muy completos que permiten al programador crear código de forma muy sencilla y corregir errores de manera muy dinámica, sobre todo aquellos asociados a la compilación o a la importación de librerías, además de contar con aplicaciones de depuración de programas para así poder realizar un mejor seguimiento de la ejecución de los mismos. Su documentación se puede encontrar en la página oficial de Sun Microsystems: http://java.sun.com/javase/reference/index.jsp.

3.2.2. Google Maps API V2

La *API* de *Google Maps*, tal como se explicaba en el apartado anterior, nos permite representar ubicaciones e itinerarios de forma gráfica con mapas en diferentes vistas, satélite, carretera, hibrida, etc. Esta API nos ofrece un completo repertorio de clases para su tratamiento. A continuación, se exponen algunas de las principales funciones que permite utilizar la versión dos de la API, así como su configuración básica y los requisitos necesarios para poder emplearla en nuestro proyecto.

3.2.2.1. Configuración básica del proyecto para el correcto funcionamiento de Google Maps API V2

Primeramente, debemos disponer de un dispositivo con *Google Play* instalado para poder ejecutar el proyecto que utiliza *Google Maps API*, de lo contrario no podremos ejecutar nuestra aplicación. Además, nuestro proyecto deberá incluir la librería del SDK de Android *Google Play Services* en su última versión para obtener acceso a los mapas y poder trabajar con ellos a nivel de programación.

Hecho lo anterior, lo siguiente es obtener una *API Key* para poder descargar los mapas en aquellas interfaces de la aplicación que los utilicen. Los pasos a seguir podemos encontrarlos en la documentación de la API en https://developers.google.com/maps/documentation/android/?hl=es .

Por último, debemos configurar una serie de datos y permisos en la aplicación que utiliza la API de *Google Maps*. Esta información se configura en el fichero ***AndroidManifest.xml*** de nuestro proyecto. A continuación se explican brevemente los datos que se deberán indicar.

Versión de la librería de Google Play Services. Se trata de metadatos que resultan útiles a nivel informativo para el sistema operativo a la hora de ejecutar la aplicación con *Tecnología Google Maps*.

API Key. En este apartado indicamos la Clave que se ha obtenido para poder emplear los servicios de mapas de Google.

Los permisos a configurar en el fichero *manifest* son los siguientes:

- *Android.permission.INTERNET*. Este permiso nos otorga acceso a la red, lo que nos permite descargar los mapas de los servidores de Google, así como el acceso al sistema de bases de datos a través del servidor de la aplicación.
- *Android.permission.ACCESS_NETWORK_STATE*. Permite comprobar el estado de la red a la API para saber si es posible o no descargar los mapas.
- *Android.permision.WRITE_EXTERNAL_STORAGE*. Permite almacenar información en un sistema de almacenamiento externo, lo que hace posible que la *API* de *Google Maps* utilice este almacenamiento como cache de información de mapas.

- *Android.permision.ACCESS_COARSE_LOCATION*. Este permiso nos ofrece la posibilidad de utilizar los datos de ubicación basados en las redes WIFI y las redes móviles, lo que nos dará una localización menos precisa basada en las células.
- *Android.permission.ACCESS_FINE_LOCATION*. Si queremos alta precisión en los datos de localización obtenidos de la *API* de *Google Maps*, este es el permiso imprescindible a utilizar, ya que otorgará acceso al sistema GPS (Global Position System) del dispositivo para una obtención de localización basada en los satélites geoestacionarios.

Otro requisito para a configurar en el fichero *manifest* para que los mapas funcionen correctamente es el acceso a gráficos *OpenGL* para el dibujado de los mapas, de lo contrario, es posible que no podamos cargar el mapa.

3.2.2.2. Obtener API Key para Google Maps y Registrar el Proyecto Android

Hasta ahora se ha hablado de la configuración básica necesaria para el correcto funcionamiento de *Google Maps*. Entre otras cuestiones, una fundamental, como se comentó en el apartado anterior, es la obtención del *API Key de Google Maps* y registrar el proyecto que la va a utilizar en la *Consola de Desarrollo de Google*.

Lo primero que se debe hacer es obtener el certificado digital *SHA1*. Para ello debemos acceder al almacén de claves con la herramienta *KeyTool* proporcionada por el *SDK de Android*. Existen dos tipos de certificados, el de desarrollo (*Debug*) y el de aplicación final para la venta (*release*). Para obtener el certificado, tendremos que utilizar el almacén de claves correspondiente (*Key store*). En la figura 3.2.1 podemos observar un ejemplo de certificado de desarrollo con la clave SHA1 correspondiente.

```
Alias name: androiddebugkey
Creation date: Jan 01, 2013
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=Android Debug, O=Android, C=US
Issuer: CN=Android Debug, O=Android, C=US
Serial number: 4aa9b300
Valid from: Mon Jan 01 08:04:04 UTC 2013 until: Mon Jan 01 18:04:04 PST 2033
Certificate fingerprints:
     MD5:  AE:9F:95:D0:A6:86:89:BC:A8:70:BA:34:FF:6A:AC:F9
     SHA1: BB:0D:AC:74:D3:21:E1:43:07:71:9B:62:90:AF:A1:66:6E:44:5D:75
     Signature algorithm name: SHA1withRSA
     Version: 3
```

Figura 3.2.1. Almacén de claves para un certificado de desarrollo.

Una vez que ya se ha obtenido la clave *SHA1* que nos servirá como certificado digital, debemos registrar nuestro proyecto en la consola para desarrolladores de Google (*Google Developer's Console*). Para ello, accedemos a nuestra cuenta de Google y a continuación a la consola de desarrolladores https://console.developers.google.com/. Una vez dentro, nos aparecerá una pantalla como la mostrada en la figura 3.2.2~~-A~~.

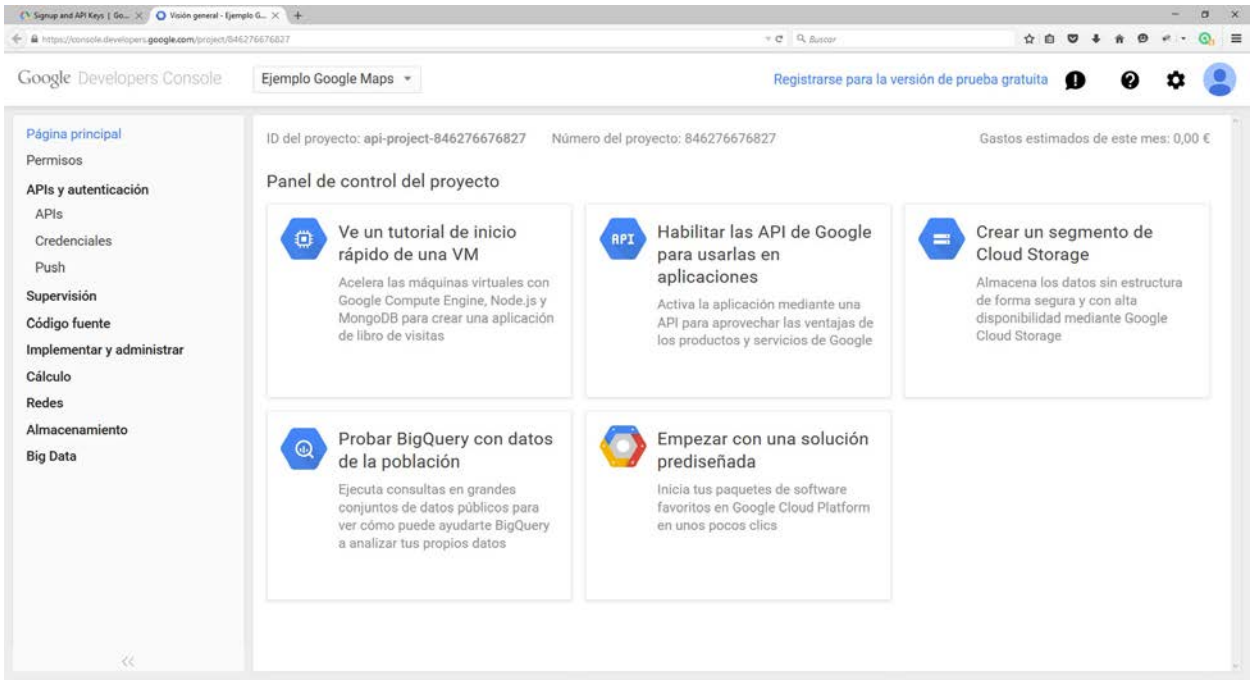

Figura 3.2.2 ~~A~~. Pantalla principal de la Consola para desarrolladores de Google.

Desde esta pantalla principal, debemos acceder al menú de credenciales, donde encontraremos todas las claves ya registradas para el proyecto seleccionado. En esta pantalla crearemos la asociación entre la aplicación móvil y Google para que cuando ésta solicite cualquier información sobre mapas, tenga autorización para hacerlo. Para establecer este vínculo debemos crear una nueva credencial con la clave SHA1 que obtuvimos en el paso anterior y el nombre del paquete de nuestra aplicación. En la figura 3.2.3~~2 B~~ podemos ver un ejemplo la creación de una *API Key*.

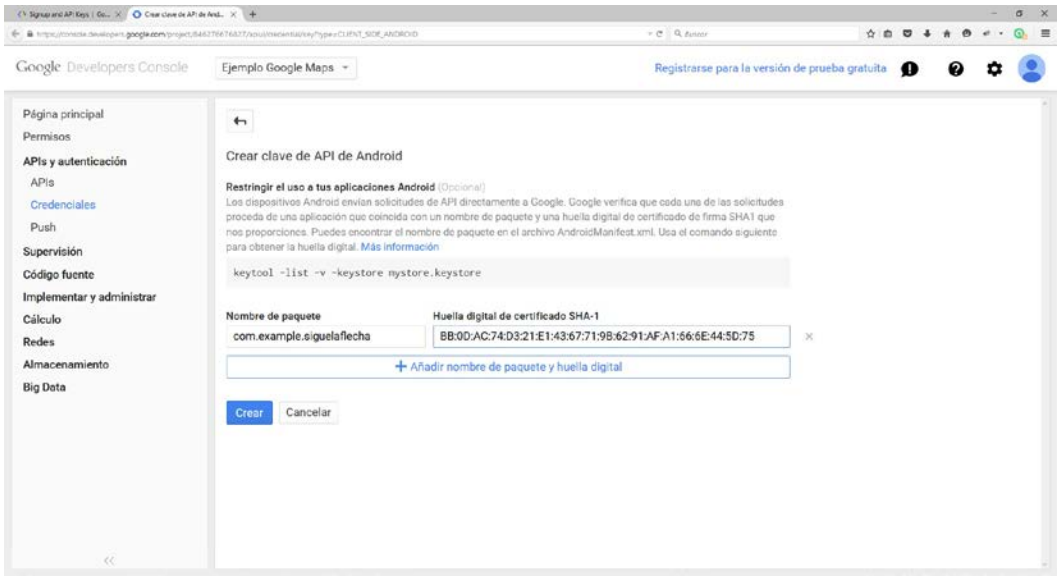

Figura 3.2.3~~2 B~~. Creando una API Key.

3.2.3.Google Directions API

Este servicio nos ofrece la posibilidad de obtener datos de ruta en función de un origen y un destino, pudiendo configurar diversos parámetros adicionales como por ejemplo, si la ruta evita peajes o no, establecer rutas que no utilicen un ferri siempre que sea posible, etc.

Al igual que en el caso de *Google Maps*, necesitaremos crear una *API key* para obtener acceso a este servicio, solo que el tipo de clave será de servidor, ya que se trata de un servicio web que funciona bajo el protocolo *HTTP (Hyper Text Transfer Protocol)*. El proceso de creación de una API Key de servidor se realiza desde la *Consola de Desarrollo de Google* al igual que para el caso de *Google Maps*.

Una vez que disponemos de la *API Key* de servidor para obtener acceso al servicio, podemos ver algunas de las bases de su funcionamiento. El sistema funciona utilizando el paso de parámetros por *GET* mediante el protocolo *HTTP* al igual que una página web. ~~En la figura 3.2.3~~A continuación se muestra un ejemplo de url para obtener los datos de una ruta que va desde La Unión (Murcia) a Madrid.

https://maps.googleapis.com/maps/api/directions/json?origin=La+Uniond+%28Murcia%29&destination=Madrid&mode=driving&key=MIAPIKEY

~~Figura 3.2.3. Ejemplo de ruta desde La Unión a Madrid utilizando la API de direcciones de Google.~~

Como podemos ver, la dirección tiene una serie de parámetros que definen el origen, el destino, la forma de realizar el itinerario, etc. A continuación pasamos explicar para que sirve cada uno de los parámetros que aparecen.

Origin. Define el punto de partida de la ruta a realizar. Este valor se puede indicar tanto en texto~~, como es el caso del ejemplo de la figura 3.2.3,~~ como con~~o~~ mediante coordenadas.
Destination. Define la ubicación de destino de nuestro itinerario. Al igual que el parámetro anterior, se puede indicar un valor en texto o en coordenadas según el caso.
Mode. Este parámetro define la forma en que el usuario realizará la ruta. Dependiendo de la forma en que el usuario realice la ruta, el sistema nos devolverá rutas distintas. Por ejemplo, obtendremos diferentes resultados si utilizamos un coche, un transporte público, una bici o simplemente, vamos andando. Los valores que puede tomar este parámetro son los siguientes:

- *Driving*. En caso de que no se indique el parámetro mode en la url, este será el modo por defecto, ya que el sistema presupone que las rutas se realizan normalmente en coche, una suposición heredada de que generalmente los sistemas GPS se emplean para realizar rutas en coche.

- *Walking*. Este modo le indica al API que se va a ir a pie, por lo que las rutas que devuelve el sistema están basadas en zonas peatonales o arcenes por los que puedan caminar los transeuntes, siempre que sea posible.

- *Bicycling*. Este modo devuelve rutas que utilizan carriles bici o similares para circular siempre que sea posible.

- *Transit*. Elegiremos este modo si queremos utilizar el transporte público para viajar. Adicionalmente, podemos elegir la hora de salida y de llegada para el transporte que se va a utilizar (tren, autobús, etc).

Key. Aquí es donde indicaremos la *API Key* obtenida en la consola de desarrollo para que el servicio nos devuelva correctamente los datos, ya que es esta clave la que nos da autorización para comunicarnos con los servidores.

JSON o XML. Por último, hay que destacar el primer parámetro que se le pasa a la URL. Este primer parámetro indica al servicio web quée formato debe emplear para devolver la información. Como ya se dijo anteriormente, para nuestro desarrollo se empleará el formato JSON, debido a la sencillez de tratamiento que nos ofrece Java.

3.2.3.1. Ejemplos de resultados obtenidos desde la Google Directions API.

Hemos visto que la API de direcciones de Google tiene dos formatos de salida: JSON y XML. Para el desarrollo del presente proyecto, se ha empleado el formato JSON para tratar la salida que se nos ofrece como respuesta a las solicitudes de itinerarios, como ya se ha visto durante la descripción de las funciones básicas de la API, debido al hecho de que es menos pesado que XML. A continuación se puede ver un ejemplo sobre la salida resultante en formato JSON, explicando las partes del contenido más importantes. La salida en formato XML es equivalente.

En primer lugar, se debe comprender la estructura básica de la respuesta devuelta por los servidores de Google. En la figura 3.2.3.1 podemos observar la estructura básica de la respuesta JSON que Google nos devuelve con los datos de la ruta solicitada y sus posibles recorridos. La estructura básica se sustenta en los siguientes componentes:

Gocoded_waypoints. Contiene información básica sobre la geo codificación de la ruta (estado, localidad, política o el identificador de la ruta solicitada).
Routes. Contiene toda la información referente a la ruta y las indicaciones necesarias para llegar al destino desde el origen indicado.
Status. Define la información devuelta por Google en la respuesta con diferentes resultados, como por ejemplo, OK, ZERO_RESULTS, etc.

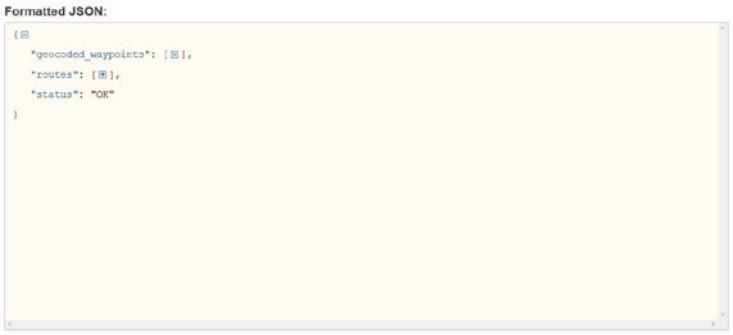

Figura 3.2.3.1 A. Estructura básica de la respuesta JSON de los datos de ruta devuelta por la Google Directions API.

El array Routes
Esta sección de la respuesta devuelta por Google Directions API, contiene todos los datos referentes a los posibles recorridos de la ruta solicitada entre un origen y un destino dado. A continuación, se detalla el contenido de los datos más importantes.

Bounds. Contiene la ubicación en términos de latitud y longitud, las fronteras que delimitan con la ruta solicitada.

Legs. Este array contiene todos los posibles recorridos que se pueden tomar para realizar la ruta entre origen y destino, así como las indicaciones para cada uno de ellos. Los parámetros que contiene cada una de estas rutas los vemos en los siguientes campos:

Start_address. Contiene la dirección establecida como punto de partida del recorrido en formato texto. Por ejemplo: “La Unión (Murcia)”.

Start_location. Contiene las coordenadas de la ubicación establecida como punto de partida de la ruta a realizar.

End_address. Contiene la dirección del punto de destino del recorrido formato texto.

End_location. Contiene las coordenadas de la ubicación asociada al punto de destino del recorrido a realizar

Distance. Contiene el valor de la distancia entre la ubicación origen y la ubicación destino en cada uno de los recorridos posibles.

Duration. Contine la duración del viaje desde el origen al destino en cada uno de los recorridos posibles.

Steps. En este array se almacenan todos los puntos por los que se debe pasar en cada recorrido y las indicaciones GPS correspondientes.

Way_points. Contiene todos los puntos intermedios que se tiene en cuenta para el cálculo de los diferentes recorridos.

En la figura 3.2.3.1 B, podemos ver un ejemplo del contenido del array legs en formato JSON.

Figura 3.2.3.1 B. Ejemplo del campo legs y su contenido.

3.2.4.La librería GSON

JSON (JavaScript Object Notation) es un formato de intercambio de información que está basado en estructuras de pares clave-valor. Es un formato mucho más ligero que *XML* y más indicado que éste en determinados escenarios.

Para el desarrollo de la aplicación que nos ocupa, este formato se empleará para el intercambio de información entre la aplicación móvil y el servidor, como datos de la ruta a insertar, el usuario y la contraseña a validar al hacer log in, etc. Además, el formato *JSON* será el empleado para recibir la respuesta por parte de los servidores de Google sobre los datos de los itinerarios a consultar.

Otro de los motivos por los que se utiliza JSON es porque nos permite de una forma muy sencilla tratar objetos Java como arrays asociativos enviados como texto, lo que hace mucho más ligero el procesado y el envío de la información por la red. Para este fin, utilizaremos la librería ***GSON***, que nos ofrece herramientas que facilitan mucho este procesado de objetos tanto en el origen como en el destino.

Es posible descargar esta librería desde su página web oficial https://sites.google.com/site/gson/, de forma gratuita. En este proyecto se ha empleado el Objeto *GSON* y el Objeto *GSONBUILDER* como principales bazas para el traspaso de datos entre aplicación y servidor.

3.2.5.La Clase Gson

Esta clase es la encargada de transformar un objeto Java a formato JSON y viceversa. Con el método *toJson()*, podemos transformar cualquier objeto a formato JSON. Este proceso se conoce como serialización de objetos, lo que nos permite transmitir objetos a través de la red de forma totalmente transparente al tipo de datos del que se trate, ya que para el sistema, se transmite como una cadena de texto.

Por otra parte, para recibir información en formato JSON, podemos utilizar el método *fromJson()* que requiere dos parámetros:

- *La cadena en formato JSON*. Este parámetro es la información a pasar de JSON a objeto Java.
- *La clase contenedora*. Este parámetro tiene como fin, determinar el tipo de datos al que se va a convertir la información recibida en formato JSON. Puede ser un objeto, un array una variable, etc.

A este último proceso, se le denomina *deserialización de objetos*, ya que se trata de la conversión del contenido JSON a un tipo definido por el lenguaje de programación que nos permitirá tratar la información de manera directa.

3.2.6.Librería View Pager Indicator

La librería ***View Pager Indicator*** nos permite crear un sistema de pestañas muy similar al que podemos encontrar hoy en día en muchas aplicaciones modernas, como por ejemplo la propia tienda de Google. Este sistema nos permite tener accesible todo el contenido principal de nuestra aplicación con un sencillo gesto de deslizamiento de un dedo, además de estar muy bien agrupado por secciones. Con este sistema, se ha diseñado la pantalla principal de la aplicación que nos permite la manipulación de la información general de rutas, vehículos y mapas.

3.3. Módulos y clases de la aplicación

A continuación nos centraremos en los diferentes módulos de los que se compone nuestra aplicación, ~~y~~ explicando cuál es el funcionamiento de cada uno de estos y sus distintos componentes.

3.3.1.El Servidor

A continuación se va a abordar todos los detalles de implementación sobre el servidor que se vio en el desarrollo teórico agrupando las diferentes clases que podemos encontrar en el paquete Servidor de la aplicación.

La Clase ClienteMYSQL

public class **ClienteMySQL**
Es la clase encargada de realizar todas las operaciones sobre la base de datos MYSQL que almacena la información sobre los itinerarios, usuarios del sistema, y pasajeros suscritos a los itinerarios publicados.

public **ClienteMySQL()**
Crea el objeto para la interacción con la base de datos MYSQL.

public void **conectar()**
Establece la conexión con la base de datos y permite ejecutar operaciones sobre ella.

public ResultSet **consultarDatos**(String consulta)
Este método devuelve un conjunto de datos con la información solicitada a la base de datos. Solamente válido para consultas de selección. En las consultas de actualización, inserción o eliminación, devolverá un conjunto de datos vacío que no nos aporta información.
Parámetros:
Consulta. Cadena de texto que contiene la instrucción sql a ejecutar.
Tipo devuelto Resulset. Conjunto de datos devuelto como resultado de la operación sobre la base de datos.

public int **actualizarDatos**(String sql)
Permite realizar operaciones de actualización, inserción o eliminación sobre la base de datos.
Devuelve un entero que indica el número de filas afectadas por la operación.
Parámetros:
sql. Cadena de texto que contiene la instrucción sql a ejecutar.
Tipo devuelto int. Devuelve un número entero donde se indican las filas afectadas por la operación sql.

public void **desconectar**()
Este método permite cerrar una conexión abierta previamente con el método conectar() con la base de datos.

La Clase Procesador

public class **Procesador**
La clase Procesador tiene como misión atender todas las peticiones derivadas de los clientes, identificarlas y procesarlas de la forma adecuada en función del tipo de consulta que realiza el usuario.

public **Procesador**()
Crea el objeto que recibe las peticiones de los usuarios y las identifica para su procesamiento.

protected void **processRequest**(javax.servlet.http.HttpServletRequest request,
javax.servlet.http.HttpServletResponse response)
throws javax.servlet.ServletException,
java.io.IOException
Se encarga de procesar las solicitudes y respuestas en el servidor para los métodos GET y POST de HTTP.
Parámetros:
Request. Solicitud realizada por un cliente al servidor.
Response. Almacena los datos de la respuesta derivada del procesamiento de la solicitud al servidor.

protected void **doGet**(javax.servlet.http.HttpServletRequest request,
javax.servlet.http.HttpServletResponse response)
throws javax.servlet.ServletException,
java.io.IOException
Se encarga del manejo de las solicitudes realizadas por el método GET de HTTP ofreciendo una respuesta adecuada a ellas.
Parámetros:
Request. Solicitud realizada mediante el Método GET.
Response. Respuesta resultante del procesamiento de la información recibida en la solicitud.

protected void **doPost**(javax.servlet.http.HttpServletRequest request,
javax.servlet.http.HttpServletResponse response)
throws javax.servlet.ServletException,
java.io.IOException
Es equivalente al método doGet(), pero utilizando el método POST de HTTP.

Métodos de procesado de peticiones

private String **iniciarSesion**(HttpServletRequest solicitud)
Se encarga de realizar la verificación del usuario que intenta acceder al sistema.
Parámetros:
Solicitud. Información de usuario y contraseña a validar
Tipo devuelto String. Devuelve una cadena en formato JSON con la validación del usuario permitiendo o denegando el acceso en función de si el usuario es correcto o no.

private String **registrarUsuario**(HttpServletRequest solicitud)
Es el método encargado de realizar las operaciones necesarias para el registro de los nuevos usuarios en el sistema.
Parámetros:
Solicitud. Contiene los datos de registro necesarios para dar de alta al usuario en el sistema.
Tipo devuelto String. Devuelve una confirmación para verificar que el usuario ha sido registrado correctamente.

private String **registrarVehiculo**(HttpServletRequest solicitud)
Este método permite registrar los datos del vehículo que el conductor utilizará cuando realice los itinerarios que tiene publicados en el sistema.
Parámetros:
Solicitud. Contiene los datos del vehículo a registrar o actualizar.
Tipo devuelto String. Devuelve una confirmación de que el vehículo ha sido registrado correctamente.

private String **obtenerDatosVehiculo**(HttpServletRequest solicitud)
Se encarga de enviar los datos del vehículo solicitado por el usuario.
Solicitud. Contiene el identificador del vehículo del que se desean conocer sus datos.
Tipo devuelto String. Devuelve los datos del vehículo solicitado en formato JSON.

private String **crearRuta**(HttpServletRequest solicitud)
Da de alta una nueva ruta en el sistema con los datos ofrecidos en la solicitud del usuario.
Parámetros:
Solicitud. Datos de la ruta a crear en el sistema.
Tipo devuelto String. Devuelve una confirmación para verificar la correcta creación de la ruta.

public **getResultadosBusqueda**(http.HttpServletRequest solicitud)
Se encarga de procesar la petición de búsqueda de itinerarios, devolviendo la información de la búsqueda solicitada en formato JSON a partir de la solicitud recibida.

Parámetros:
solicitud. Solicitud de búsqueda realizada por la aplicación móvil.
Tipo devuelto String. Devuelve los datos de la búsqueda solicitada en formato JSON.

public String **registrarPasajero**(HttpServletRequest solicitud)
Este método realiza las operaciones necesarias para suscribir a un usuario como pasajero de un itinerario determinado.
Parámetros:
solicitud. Solicitud de registro del usuario como pasajero de una ruta dada.
String. Confirmación de que el usuario ha sido dado de alta correctamente.

public String **anularRegistroPasaajero**(HttpServletRequest solicitud)
Se encarga de realizar la operación de baja de un usuario como pasajero en una ruta dada.
Parámetros:
Request. Solicitud de baja en la ruta indicada por parte del usuario.
Tipo devuelto String. Devuelve la confirmación de que el usuario ha sido dado de baja de la ruta dada correctamente.

La Clase RespuestaDatosRuta

public class **RespuestaDatosRuta**
Esta clase actúa como contendor de los datos de un itinerario, permitiendo la transferencia de los mismos entre cliente y servidor a través de la serialización de objetos.

public **RespuestaDatosRuta**()
Crea un objeto que contiene la información completa sobre la una ruta concreta.

Métodos de Obtención de información del Itinerario.
Estos métodos nos permiten obtener cualquier dato del itinerario contenido en el objeto RespuestaDatosRuta.

public String **getDireccionOrigen**()
Devuelve la dirección origen del itinerario.

public String **getDireccionDestino**()
Devuelve la dirección de destino del itinerario.

public String **getDistanciaTexto**()
Devuelve la distancia entre el origen y el destino en un formato adecuado para presentar en pantalla.

public double **getDistanciaValor**()
Devuelve el valor numérico de la distancia existente entre el origen y el destino del itinerario.

public String **getDuracionTexto**()
Devuelve el tiempo que se tarda en llegar desde el origen al destino en un formato adecuado para presentar en pantalla.

public double **getDuracionValor**()
Devuelve el valor de tiempo que se tarda en llegar desde el origen al destino del itinerario.

public String **getEstado**()
Devuelve un estado que determina si los datos alojados en el objeto son correctos o no.

public String **getFecha**()
Devuelve la fecha en que se realizará el itinerario

public String **getHora**()
Devuelve la hora de salida del itinerario.

public double **getDesviacionMaxima**()
Devuelve la desviación máxima en kilómetros permitida por el conductor durante el trayecto para la recogida de pasajeros.

public double **getRetardoMaximo**()
Devuelve el tiempo máximo que está dispuesto a esperar el conductor durante el trayecto para recoger pasajeros.

public int **getAsientosOfrecidos**()
Devuelve los asientos ofrecidos como plazas para suscribir pasajeros a un itinerario concreto.

public int **getIdUsuario**()
Devuelve el identificador del usuario que creó el itinerario.

public String **getEvitaPeajes**()
Devuelve un valor que indica si el itinerario evita peajes o no durante todo su recorrido.

public List<List<HashMap<String, String>>> **getIndicaciones**()
Devuelve un objeto HashMap con las indicaciones necesarias para realizar el recorrido dado.

Métodos de modificación de información del itinerario
Estos métodos nos permiten rellenar la información sobre el itinerario obtenido a través de la Google Direction API en formato JSON.

public void **setDireccionOrigen**(String direccionOrigenL)
Permite modificar la dirección de origen para un itinerario dado.

public void **setDireccionDestino**(String direccionDestinoL)
Permite modificar la dirección de destino del itinerario.

public void **setDistanciaTexto**(String distanciaTextoL)
Permite modificar el valor formateado de la distancia en kilómetros que nos permite visualizar la distancia en un formato adecuado para la visualización en pantalla.

public void **setDistanciaValor**(double distanciaValorL)
Permite modificar el valor de la distancia del itinerario. Este valor es el devuelto por google maps Direction API en formato decimal con máxima precisión.

public void **setDuracionTexto**(String duracionTextoL)
Permite la modificación del tiempo que se tarda en llegar desde el origen y el destino que se muestra en formato más adecuado para la visualización en pantalla.

public void **setDuracionValor**(double duracionValorL)
Este método nos ofrece la posibilidad de modificar el valor de duración entre origen y destino que podemos emplear para realizar cálculos de tiempo a nivel de programación.

public void **setEstado**(String estadoL)
Permite modificar el estado asociado a la respuesta dada con los datos del itinerario. Este estado define si los datos han sido correctamente obtenidos o no.

public void **setIndicaciones**(List<List<HashMap<String, String>>> datosRutaL)
Este método permite insertar en el objeto de datos de ruta las indicaciones asociadas a un itinerario dado.

public void **setFecha**(String fecha)
Modifica la fecha de salida del itinerario.

public void **setHora**(String hora){
Modifica la hora a la que se inicia el itinerario.

public void **setDesviacionMaxima**(double desviacionMaxima){
Modifica la desviación máxima permitida para la recogida de pasajeros.

public void **setRetardoMaximo**(double retardoMaximo)
Modifica el tiempo máximo que está dispuesto el conductor a esperar para la recogida de pasajeros.

public void **setAsientosOfrecidos**(int asientosOfrecidos)
Modifica el número de plazas disponibles en un itinerario dado.

public void **setIdUsuario**(int idUsuario)
Modifica el usuario creador de la ruta.

public void **setEvitaPeajes**(String evitaPeajes)
Modifica el flag que indica si la ruta permitirá el paso por peaje durante el recorrido en caso de que los haya o no.

La clase RespuestaDatosVehiculo

public class **RespuestaDatosVehiculo**
Es la clase contendora de los datos de un vehículo dado. Esta clase nos proporciaona la interfaz de comunicación entre la aplicación Android y el servidor para la creación y actualización de los datos asociados a un vehículo.

public **RespuestaDatosVehiculo**()
Crea un objeto contenedor de los datos de un vehículo que actua como interfaz de comunicación entre aplicación y servidor.

Métodos de Obtención de datos del vehículo
Estos métodos nos permiten obtener cualquier dato sobre el vehículo contenedor.

public String **getMatricula**()
Devuelve la matrícula del vehículo asociado.

public String **getMarca**()
Devuelve el dato de la empresa fabricante del vehículo.

public String **getModelo**()
Devuelve el modelo del vehículo asociado. Este modelo depende del fabricante.

public String **getColor**()
Nos indica el color del vehículo asociado. Esto facilita el reconocimiento a los pasajeros que se suscriben a rutas

public int **getIDUsuario**()
Devuelve el identificador del usuario al que pertenece el vehículo.

public int **getIDVehiculo**()
Devuelve el identificador del vehículo contenido.

Métodos de modificación de la información del vehículo.
Estos métodos permiten rellenar los datos del contenedor que actua como interfaz entre cliente y servidor para transmitir la información sobre un vehículo dado.

public void **setMatricula**(String matricula)
Modifica la matrícula del vehículo asociado.

public void **setMarca**(String marca)
Modifica la marca del fabricante del vehículo.

public void **setModelo**(String modelo)
Modifica el modelo del vehículo asociado.

public void **setColor**(java.lang.String color)
Modifica el color de la pintura del vehículo.

public void **setIDUsuario**(int idUsuario)
Modifica el usuario que creó el vehículo contenedor.

public void **setiDVehiculo**(int idVehiculo)
Modifica el identificador del ehículo a asociar en el objeto contenedor.

La clase RespuestaInicioSesion

public class **RespuestaInicioSesion**
Esta clase actúa como interfaz entre cliente y servidor para la operación de verificación de inicio de se sión del usuario en el sistema.

public **RespuestaInicioSesion**(String estado_login)
Crea un objeto contenedor de los datos necesarios para verificar el inicio de sesión de un usuario en el sistema.

public String **getEstadoLogin**()
Permite saber si el usuario que intenta iniciar sesión en el sistema tiene o no permisos de acceso

public int **getIDUsuario**()
Permite obtener el identificador del usuario que ha iniciado sesión si existe en el sistema y no ha habido errores.

public void **setIDUsuario**(int idUsuario)
Modifica el valor del id de usuario que intenta iniciar sesión en el sistema.

La clase RespuestaInsertarVehiculo

public class **RespuestaInsertarVehiculo**
Es la interfaz que nos indica si los datos del vehículo del usuario han sido insertados con éxito.

public **RespuestaInsertarVehiculo**(String estado_insertar, int idVehiculo)
Crea un objeto contenedor de la verificación del resultado de la operación de inserción.

public String **getEstadoInsertar**()
Devuelve el estado de la operación de inserción del vehículo, permitiendo comprobar si ha sido satisfactoria o no.

public int **getIDVehiculo**()
Devuelve el identificador del vehículo insertado, en caso de éxito, en caso contrario devuelve cero.

La clase RespuestaRegistro

public class **RespuestaRegistro**
Es equivalente a la clase RespuestaInsertarVehiculo, pero para la verificación del registro del usuario.

La clase RespuestaRegistrarPasajero

public class **RespuestaRegistrarPasajero**
Esta clase permite verificar que un pasajero ha sido suscrito o dado de baja de un itinerario concreto.

public **getEstado**() y public **setEstado**()
Estos métodos permiten asignar u obtener el valor del resultado de la operación de suscripción o baja de un pasajero.

La clase RespuestaResultados

public class **RespuestaResultados**
Esta clase se encarga de devolver la información de los resultados de búsqueda sobre los itinerarios que el usuario ha solicitado.

public **RespuestaResultados**()
Crea un objeto contenedor de los resultados de búsqueda en función del criterio dado por el usuario que realiza la solicitud.

public List<java.util.HashMap<java.lang.String,java.lang.String>> **getRutasCabecera**()
Devuelve los resultados de la búsqueda realizada en un conjunto de pares clave valor utilizando la clase HashMap.

public void **setRutasCabecera**(List<java.util.HashMap<String, String>> rutasCabecera)
Rellena los datos correspondientes a los resultados de la búsqueda asociada.

public List<java.util.HashMap< String, String>> **getRutaCompleta**()
Obtiene los resultados de búsqueda de forma detallada incluyendo las indicaciones para llegar del origne al destino del itinerario.

public void **setRutaCompleta**(List< HashMap< String, String>> rutaCompleta)
Rellena los resultados de búsqueda detallados incluyendo los datos sobre indicaciones sobre cómo llegar del origen al destino.

3.3.2.La Aplicación Sigue la Flecha

En esta sección se va a detallar el funcionamiento básico y algunos datos importantes sobre la implementación de la comunicación con el servidor (servlet) y la transferencia de datos.

3.3.2.1. La interfaz gráfica

En este apartado se va a realizar una descripción sobre las diferentes pantallas que corresponden a la interfaz de usuario y a su diseño, así como la funcionalidad de cada una.

La pantalla de Bienvenida
Esta es la interfaz de usuario principal de la aplicación. Dentro de esta pantalla tenemos acceso a tres opciones básicas:

- *Registro de Usuario*. Permite el acceso a la pantalla de registro de nuevos usuarios.
- *Iniciar Sesión*. Permite el acceso a la pantalla de inicio de sesión de la aplicación.

- *Botón de menú*. Permite el acceso al menú contextual de la interfaz. Este menú permite configurar la dirección IP o dominio en el que se aloja el servidor que recibe las peticiones de la aplicación. Por ejemplo: http://localhost:8080

En la figura 3.3.2.1 A podemos ver cómo quedaría la pantalla.

Figura 3.3.2.1 A. Pantalla de Bienvenida

La pantalla de Inicio de Sesión

Esta pantalla nos permite acceder a la actividad principal de la aplicación indicando un usuario que tengamos previamente registrado en el sistema. En caso de que los datos sean incorrectos nos avisará de que el usuario no existe o no es válido.

A continuación en la figura 3.3.2.1 B, podemos ver una muestra de la pantalla de inicio de sesión.

Figura 3.3.2.1 B. Pantalla de inicio de sesión.

Como podemos observar, se nos piden los datos del usuario que desea acceder al sistema. El primer parámetro es la dirección de correo electrónico que indicamos al registrar nuestro usuario, la cual servirá para que los pasajeros se comuniquen con nosotros cuando se suscriben a los itinerarios publicados.

La pantalla de Creación de Nuevo Usuario

Esta pantalla nos solicita los datos necesarios para dar de alta a un nuevo usuario en el sistema, el cual nos permitirá acceder a todas las funcionalidades de la aplicación.

Figura 3.3.2.1 C. Pantalla de registro de nuevos usuarios

Como se puede observar, en esta pantalla se nos pide la información básica a almacenar en el sistema sobre el usuario a registrar. Todos los datos son obligatorios, pero los más importantes que se deben recordar son el email y la contraseña, que son los que se emplearán para el inicio de sesión. Además, en caso de que no rellenemos algún campo el sistema lo marcará en rojo y lanzará el aviso correspondiente.

La pantalla Principal

Esta es la base de toda la aplicación. Aquí residen todas las funciones que el usuario puede utilizar dentro de la aplicación. Se trata de una pantalla realizada con un sistema por pestañas basado en la librería ***View Pager*** que nos permite mostrar una interfaz similar a la que vemos en *Google Play Store* donde múltiples pantallas se encuentran en la misma actividad. A continuación, en la figura 3.3.2.1 D se muestra una serie de imágenes donde podemos ver las diferentes pestañas y su funcionamiento.

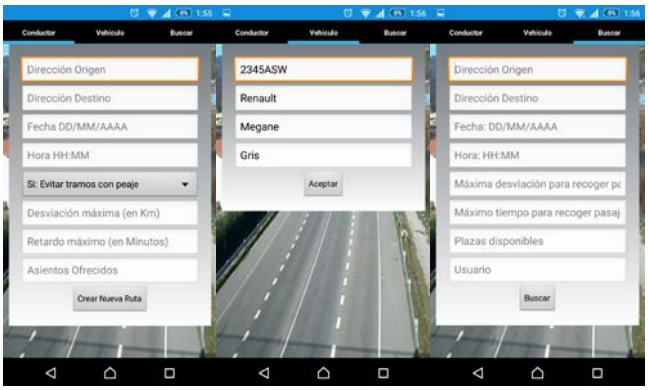

Figura 3.3.2.1 D. Pantalla principal que muestra las funcionalidades de inserción de rutas, creación o actualización de datos de vehículo y búsqueda de resultados.

La pantalla de Mapa

Esta pantalla nos muestra el recorrido de la ruta que hemos insertado en función del origen y el destino, indicando de forma visual en el mapa la información de las indicaciones que debemos seguir para ir del origen al destino permitiendo realizar zoom sobre el mapa para ver con mayor o menor detalle dichos datos. En la figura 3.3.2.1 E, podemos ver un ejemplo de ruta y sobre cómo generarla.

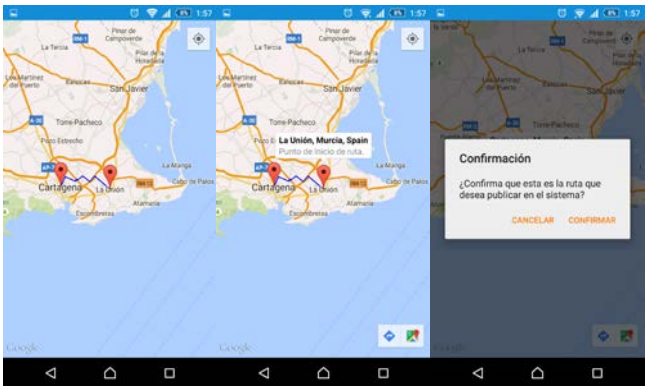

Figura 3.3.2.1 E Ejemplo de representación de ruta sobre el Mapa y confirmación de creación.

Como se puede observar en la figura 3.3.2.1 E, lo primero que nos muestra es la ruta en un plano general, pudiendo seleccionar cualquiera de los dos puntos, origen o destino, para que nos muestre información sobre éste. Una vez que sabemos con seguridad que es esa la ruta que queremos crear, al pulsar sobre el cuadro de información del origen o del destino nos aparecerá un dialogo de confirmación que permitirá confirmar la creación de la ruta, lo que enviará una solicitud al servidor para almacenar los datos de la misma en el sistema de bases de datos.

La pantalla de Resultados de Búsqueda

En esta pantalla podemos visualizar información sobre rutas publicadas en el sistema, en función de filtros como por ejemplo, origen, destino, fecha y hora de salida, usuario que la creó, plazas disponibles, etc.

Figura 3.3.2.1 F. Ejemplo de búsqueda y confirmación de suscripción o baja de pasajeros.

Como se puede observar en la figura 3.3.2.1 F, el sistema permite seleccionar cualquier ruta y suscribirse o eliminar dicha suscripción si ya nos habíamos suscrito con anterioridad. Una vez suscritos a la ruta en cuestión, el proceso de negociación sobre el lugar de recogida y la hora se realizara de forma manual entre los usuarios participantes en el itinerario.

3.3.2.2. Clases que otorgan vida a la interfaz gráfica de la aplicación

Hasta ahora, hemos visto la funcionalidad básica de la aplicación Sigue la Flecha desde el punto de vista de la interfaz de usuario. En este apartado se realizará una descripción detallada de las diferentes clases que permiten que la aplicación funcione, tanto de u sistema de intercambio de datos, como de las clases asociadas a las actividades que ejecuta el sistema Android para mostrar las distintas pantallas.

La Clase Bienvenida

public class **Bienvenida** extends Activity
Esta clase se encarga de dar funcionalidad a la pantalla de bienvenida de la aplicación, dándonos acceso al registro de nuevos usuarios, y al inicio de sesión, así como a la configuración del menú de preferencias para la url a la que debe acceder el sistema al solicitar cualquier información al servidor.

@Override
protected void **onCreate**(Bundle savedInstanceState)
Se encarga de la creación de la interfaz de bienvenida y de la inicialización y activación de la funcionalidad de los controles gráficos.
Parámetros
SavedInstanceState. Este parámetro contiene la información más reciente de la última vez que se suspendió la actividad, lo que nos permite regenerar la actividad con la información previa almacenada en dicho parámetro Bundle.

@Override
public boolean **onCreateOptionsMenu**(Menu menu)
Este método se encarga de la creación de los menús de preferencias que permiten configurar la url a la que apunta la aplicación para las peticiones de cualquier información.
Parámetros:
Menú. Contiene el fichero xml con las opciones que se mostrarán en el menú contextual.
Tipo devuelto Boolean. Devuelve true o false en función de si se desea mostrar o no el menú contextual.

@Override
public boolean **onCreateOptionsMenu**(Menu menu)
Es el método encargado de generar la interfaz gráfica del menú contextual que muestra las diferentes opciones de menú a las que tenemos acceso.
Parámetros:
Menú. Contiene el fichero xml con las opciones que se mostrarán en el menú contextual.

@Override
public boolean **onOptionsItemSelected**(MenuItem item)
Es el evento encargado de manejar las acciones a realizar cuando se selecciona un ítem del menú contextual.
Parámetros:
Ítem. Es el parámetro que permite obtener información sobre el ítem seleccionado para realizar la acción indicada en el manejo del evento.
Tipo devuelto Boolean. Devuelve true o false en función de si el evento asociado al ítem seleccionado debe ejecutarse o no.

public void **lanzarInicioSesion**(View v)
Se encarga de ejecutar la actividad que carga la pantalla de inicio de sesión.
Parámetros:
v. Este parámetro indica la vista que ha provocado la ejecución de la actividad de inicio de sesión.

public void **lanzarRegistro**(View v)
Lanza la actividad que carga la pantalla de registro de nuevos usuarios.
Parámetros:
v. Es la vista que ha provocado la ejecución de la actividad que carga la pantalla de registro.

public void **lanzarConfiguracion**(View v)
Ejecuta la pantalla de preferencias de la aplicación a través del botón de configuración del menú contextual.
Parametros:
v. Es la vista que ha causado la ejecución de la pantalla de preferencias.

La clase InicioSesion

public class **InicioSesion** extends Activity
Esta clase se encarga de mostrar y dar funcionalidad a la pantalla de inicio de sesión, ejecutando el proceso que nos permite verificar los datos de usuario para acceder a la pantalla principal de la aplicación.

@Override
protected void **onCreate**(Bundle savedInstanceState)
Se encarga de inicializar la actividad de inicio de sesión mostrando la pantalla correspondiente con la funcionalidad de verificación de datos de usuario.

public void **lanzarPantallaPrincipal**(View v)
Ejecuta la actividad que muestra la pantalla principal de nuestra aplicación.
v. almacena la vista causante de la ejecución de la actividad que muestra la pantalla principal.

La clase PantallaPrincipal extends Activity

public class **PantallaPrincipal**
Esta es la clase que contiene la funcionalidad más compleja dentro de la aplicación. Se encarga de ejecutar la actividad que carga la pantalla principal donde el usuario realiza todas las acciones como, consulta de rutas, creación de nuevas rutas, creación o actualización de la información sobre el vehículo que se utiliza como conductor, etc.

@Override
protected void **onCreate**(Bundle arg0)
Inicializa la actividad y carga la vista ViewPager que nos muestra las diferentes pantallas en base a un sistema de pestañas.

@ Override
public void **onBackPressed**()
Evento que gestiona la pulsación del botón físico o táctil que permite ir a la actividad anterior para que no salga directamente de la actividad de pantalla principal, llevándonos a la pestaña anterior siempre que no estemos en la primera de ellas.

El sistema de pestañas de la pantalla principal
Las pestañas que contiene la pantalla principal están manejadas por componentes independientes que se ensamblan mediante un adaptador con el sistema de pestañas de la pantalla principal. Cada uno de estos componentes gestiona la funcionalidad de forma independiente. A continuación se describirá con detalle cada uno de los componentes.

La clase RutaConductor

public class **RutaConductor** extends Fragment
Esta clase se encarga de crear el componente del sistema de pestañas que permite mostrar la pantalla de creación de nuevas rutas de conductor.

public static **RutaConductor** newInstance(Activity actividad)
Inicializa la instancia del componente que carga la interfaz que permite crear rutas de conductor.
Parámetros:

Actividad. Es el contexto de la actividad que genera el componente.

@Override
public View **onCreateView**(LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState)
Se encarga de inicilizar el componente y crear el layout que mostrará la pestaña de creación de rutas de conductor.

public void **lanzarMapa**(View v)
Es el método que ejecuta la actividad que carga el mapa con la representación visual de la ruta que se desea crear.
Parámetros:
v. Es la vista que ha provocado la ejecución de la actividad.

La clase InsertaVehiculo

public class **InsertaVehiculo** extends Fragment
Esta clase se encarga de crear el componente del sistema de pestañas que permite mostrar la pantalla de creación de nuevas rutas de conductor.

public static **RutaConductor** newInstance(Activity actividad)
Inicializa la instancia del componente que carga la interfaz que permite crear rutas de conductor.
Parámetros:
Actividad. Es el contexto de la actividad que genera el componente.

@Override
public View **onCreateView**(LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState)
Se encarga de inicializar el componente y crear el layout que mostrará la pestaña que gestiona la creación y actualización de los datos del vehículo que el conductor utiliza para realizar las diferentes rutas que ha publicado.

La clase BuscaRuta

public class **BuscaRuta** extends Fragment
Esta clase se encarga de crear el componente del sistema de pestañas que permite mostrar la pantalla de búsqueda de rutas en el sistema servidor.

public static **RutaConductor** newInstance(Activity actividad)
Inicializa la instancia del componente que carga la interfaz que permite buscar rutas de conductor.
Parámetros:
Actividad. Es el contexto de la actividad que genera el componente.

@Override
public View **onCreateView**(LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState)
Se encarga de inicilizar el componente y crear el layout que mostrará la pestaña de búsqueda de rutas de conductor.

public void **lanzarResultados**(View v)
Es el método que ejecuta la actividad que carga la pantalla con los resultados de búsqueda en función de los parámetros indicados por el usuario.
Parámetros:
v. Es la vista que ha provocado la ejecución de la actividad.

La Clase Mapa

public class **Mapa** extends FragmentActivity implements OnMapReadyCallback
Esta es la clase que gestiona todo lo relacionado con la visualización de la ruta a crear en un mapa de carreteras obtenido con la API de Google Maps.

@Override
protected void **onCreate**(Bundle savedInstanceState)
Crea lo necesario para mostrar la interfaz con el mapa y la representación de las indicaciones para llegar del origen al destino.

@Override
public void **onMapReady**(GoogleMap map)
Es el evento que gestiona la inicialización del mapa en pantalla y la representación de la ruta que se desea crear, garantizando que no se crea ningún objeto sobre el mapa hasta que no se haya cargado por completo.

private void **enviarDatosRuta**(View v)
Es el método encargado de preparar los datos de la ruta que se va a crear para enviarlos al servidor en una solicitud HTTP.

La Clase Resultados

public class **Resultados** extends ListActivity
Esta clase se encarga de mostrar la pantalla que contiene los resultados de búsqueda de rutas emplenanto un sistema de listado de resultados basado en un control ListView.

@Override
protected void **onCreate**(Bundle savedInstanceState)
Inicializa la actividad que muestra la pantalla de resultados con el listado de resultados de búsqueda.

@Override
protected void **onListItemClick**(ListView l, View v, int position, long id)
Se trata del evento que gestiona el click sobre uno de los ítems del listado de resultados de búsqueda, lo que nos permite suscribirnos o darnos de baja en cualquier ruta de las presentes en el listado de resultados.

3.3.2.3. La Clase Async Task

Una vez que ya se ha visto la base que compone las actividades de la aplicación, el siguiente paso es explicar la estructura básica de las clases que se encargan del procesamiento de las distintas solicitudes que se realizan al servidor para obtener la información necesaria para las diferentes operaciones (inicio de sesión, registro de usuarios, creación de rutas, etc).

La clase ***Async Task*** nos permite realizar procesos pesados en una aplicación Android empleando un hilo de ejecución secundario independiente de la interfaz gráfica, lo que nos permite evitar que el sistema operativo muestre el fatídico mensaje que indica que la aplicación no responde, lo que puede causar que se cierre inesperadamente.

Las clases *consultaXX*, donde *XX* corresponde al tipo de operación a realizar (inicio de sesión, creación de ruta, etc) tiene todas una estructura basada en la clase ***AsyncTask***. A continuación se detalla su estructura básica.

Las Clases ConsultaXX

public class **ConsultaXX** extends AsyncTask
Cada una de estas clases, se encarga de realizar la una operación concreta (inicio de sesión, registro de usuarios, creación de rutas, etc.).

public **ConsultaXX**(Activity actividad)
Crea el objeto que prepara la aplicación para enviar los datos de la solicitud HTTP que define la operación que debe realizar el servidor y trata la respuesta resultante de dicha operación.
Parámetros:
Actividad. Este parámetro define la actividad que está ejecutando la tarea asíncrona, lo que permite obtener los datos necesarios para realizar la operación adecuada en el contexto correcto.

@Override
protected void **onPreExecute**()
Este método se encarga de validar que todos los datos que se han pasado a través de la interfaz de usuario son correctos.

@Override
protected String **doInBackground**(String... urls)
Se encarga de la ejecución de la parte pesada de la operación a realizar , enviando la solicitud con la información correcta y recibiendo la respuesta con el resultado.
Parámetros:
Urls. Este parámetro almacena el enlace que contiene la dirección a laque debe conectarse la aplicación con el tipo de operación a realizar.

@Override
protected void **onPostExecute**(String output)
Es el método que muestra en pantalla la información asociada a los resultados de la ejecución de la tarea asíncrona.
Parámetros:
Output. Almacena la cadena de texto que contiene el resultado de la operación realizada.

private String **getOutputFromUrl**(String url)
Realiza la petición de información al servidor y devuelve el resultado de la respuesta recibida desde el servidor en función del tipo de operación que se desea realizar.
Parámetros:
url. Define el tipo de operación que se solicita al servidor.

3.3.2.4. Las Clases de configuración y tratamiento de la información

En este apartado, se van a tratar aquellas clases que ayudan al tratamiento de los datos dentro de la aplicación, así como la configuración de la aplicación.

La clase ConversorJSONRuta

public class **ConversorJSONRuta**
Esta clase contiene los métodos necesarios para realizar la conversión del formato json devuelto por Google Maps a un objeto legible dentro del lenguaje de programación.

public List<List<HashMap<String, String>>> **parse**(JSONObject jObject)
Este método se encarga de tratar la información devuelta por la API de direcciones de Google sobre las posibles rutas en formato JSON devolviendo un objeto que contiene los datos transformados de manera que sean legibles para la representación en el mapa.
Parámetros:
jObject. Datos en formato JSON de la respuesta de la API de direcciones de Google.
Tipo devuelto <List<List<HashMap<String, String>>>>. Devuelve los datos de Google Maps en un objeto Java que nos permite tratar la información de la ruta de forma sencilla para representarla en el mapa.

private List<LatLng> **decodePoly**(String encoded)
Transforma los datos de latitud y longitud de una localización dada en formato JSON a un objeto Java que nos permite utilizar los valores de forma normal en la programa-ción.
Parámetros:
Encoded. Cadena JSON que contiene los datos sobre latitud y longitud de las indicaciones de una ruta dada desde el origen al destino.
Tipo devuelto List<LatLng>. Devuelve el listado de objetos de tipo LatLng que representan cada una de las indicaciones que nos permiten llegar de origen a destino en una ruta dada.

La Clase GestorPreferencias

public class **GestorPreferencias**
Esta clase permite gestionar todo lo relacionado con la información de la pantalla de configuración de la aplicación.

public **GestorPreferencias**(Activity actividad)
Crea un objeto que permite leer, escribir y restablecer los valores de las preferencias en la pantalla de configuración.
Parámetros:
Actividad. Define la actividad que carga la pantalla de configuración para poder manejar los recursos del sistema a la hora de tratar algún valor en las preferencias.

public String **leerPreferencias**()
Este método se encarga de leer los datos contenidos en el fichero de preferencias y nos la devuelve en una cadena de caracteres.
Tipo devuelto String. Devuelve la cadena de la url a la que apunta la aplicación para solicitar la información al servidor de publicación de rutas.

public void **cargarValoresPorDefecto**()
Se encarga de restablecer la configuración de la aplicación a sus valores por defecto.

public void **guardarPreferencias**(String url)
Modifica la url que se emplea para conectar con el servidor de publicación de rutas.
Parámetros:
url. Dirección a la que se conecta la aplicación para realizar las peticiones al servidor de publicación de rutas.

La clase MetodosGenerales

public class **MetodosGenerales**
Esta clase permite la utilización de ciertos parámetros y métodos que realizan funciones básicas de propósito general totalmente independientes al contexto en que se ejecutan, como por ejemplo validación de fechas, números, etc.

Variables de uso General

public static final String ***URL*** = URL_SERVIDOR;
Variable global que determina la la url del servidor al que conectar la aplicación móvil por defecto.
public static final String ***URLMAPS*** = "https://maps.googleapis.com/maps/api/directions/json?";
Define la url de acceso a la API de direcciones de Google y el formato de salida en que devuelve los datos a la aplicación.
public static final String ***keyDirectionsApi*** = API KEY;
Clave de acceso que permite utilizar la API de direcciones de Google.

Meses del año
Estas variables representan un valor numérico para los meses del año, lo que nos permite emplearlas para validar fechas en formato DD/MM/AAAA.
public static final int ***ENERO*** = 1;
public static final int ***FEBRERO*** = 2;
public static final int ***MARZO*** = 3;
public static final int ***ABRIL*** = 4;
public static final int ***MAYO*** = 5;
public static final int ***JUNIO*** = 6;
public static final int ***JULIO*** = 7;
public static final int ***AGOSTO*** = 8;
public static final int ***SEPTIEMBRE*** = 9;
public static final int ***OCTUBRE*** = 10;
public static final int ***NOVIEMBRE*** = 11;
public static final int ***DICIEMBRE*** = 12;

public static boolean **isEmailValid**(String email)
Método global que comprueba si una dirección de correo electrónico es correcta en su formato o no. No se valida contra servidor, por lo que no es posible saber si la dirección indicada ha sido creada en el dominio al que hace referencia.
Parámetros:
Email. Es la dirección de correo electrónico a validar.
Tipo devuelto Boolean. Devuelve true si la dirección de correo ha sido validada con éxito.

public static boolean **isTelefono**(String numero)
Comprueba si el número de teléfono es correcto.
Parámetros:
Numero. Contiene el número de teléfono a comprabar.
Tipo devuelto Boolean. Si la validación del teléfono es correcta devuelve true.

public static boolean **validarFecha**(String dateToValidate, String dateFromat)
Comprueba si la cadena de texto dada, respeta el formato de fecha indicado.
Parámetros:
dateToValidate. Fecha a validar en formato de cadena de caracteres.
dateFormat. Formato con el que debe concordar la cadena dada en el parámetro dateTovalidatee. Por ejemplo: DD/MM/AAAA, AAAA/DD/MM, AAAA-MM-DD, etc.
Tipo devuelto Boolean. Devuelve true en caso de que la fecha concuerde con el formato dado.

public static boolean **validarHora**(String hora)
Comprueba si una cadena de texto coincide con la hora en formato 24h. Por ejemplo: 14:00.
Hora. Cadena que contiene la hora a validar.
Tipo devuelto Boolean. Devuelve true si la cadena dada corresponde a una hora en formato 24h.

public static boolean **isNumeric**(String str)
Comprueba si la cadena dada tiene un formato numérico correcto.
Parámetros:
Str. Este parámetro contiene la cadena de texto cuyo formato se va a comprobar.
Tipo devuelto Boolean. Si la cadena dada por el parámetro str es un número, devuelve true.

4. Cuarto Capítulo
Instalación y Configuración

4.1. Introducción

Hasta el momento se ha tratado lo referente a la forma teórico-práctica de llevar a cabo el desarrollo de una aplicación que nos permitiera publicar itinerarios donde los pasajeros se puedan suscribir y así poder compartir gastos derivados del combustible. Hemos visto un planteamiento teórico que definía los diferentes módulos de los que constaba el sistema, así como su implementación. En el presente capítulo se abordará lo necesario para instalar la aplicación móvil y montar un servidor en un dominio local para la solicitud de información y publicación sobre itinerarios.

En el fichero zip denominado aplicación, encontramos dos carpetas que corresponden a cliente y servidor. En el cliente encontramos lo siguiente: El fichero SigueLaFlecha.apk, que nos permitirá instalar la aplicación Sigue la Flecha en nuestro dispositivo móvil.

En el caso del servidor nos encontramos con lo siguiente: El fichero servidor.war. Este es el fichero que tendremos que alojar en el servidor Apache Tomcat para que se puedan realizar las peticiones desde la aplicación móvil sobre itinerarios, registro de usuarios, inicio de sesión, etc.

4.2. Instalación del servlet utilizando el servidor Apache Tomcat

Lo primero es instalar Tomcat en el equipo local o en un servidor remoto, dependiendo del ámbito en que queramos que se ejecute el servlet que se encarga de procesar las peticiones desde la aplicación móvil.

Podemos instalar la última versión del servidor *Apache Tomcat* desde su página oficial http://tomcat.apache.org/. Durante la instalación se nos pide una serie de parámetros como la contraseña de acceso al servidor *Apache Tomcat*.

Una vez instalado, debemos pasar a configurar algunos parámetros en el equipo que va a contener nuestro servlet. El primero de ellos es el puerto por el que el servidor escuchará las peticiones de los usuarios, que en caso de tratarse de un servidor remoto con un dominio público, será el puerto 80, mientras que si se trata de una instalación local como en nuestro caso, se configurará algún puerto de los que haya libre en el equipo local, como por ejemplo el 8080 por similitud con el puerto habitual para este tipo de sistemas. El segundo paso es configurar el puerto elegido como excepción en el Firewall para evitar que el sistema operativo bloquee las peticiones entrantes. Finalmente, en el fichero de configuración de usuarios del servidor *Apache Tomcat*, debemos asegurarnos de configurar correctamente los roles y permisos de usuario. En la figura 4.2, podemos ver un ejemplo de cómo quedaría la configuración del servidor apache Tomcat en cuanto a los usuarios y los roles que desempeñan.

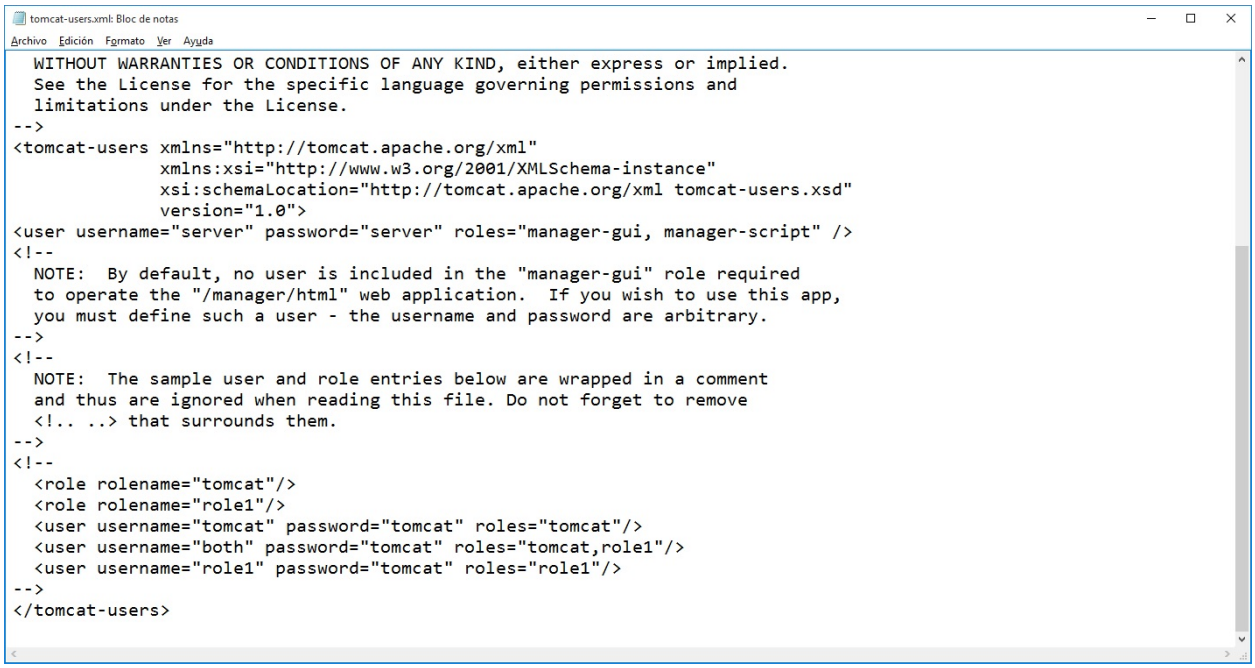

Figura 4.2. Fichero de configuración Tomcat Users.

4.3. Instalación y configuración de la aplicación móvil

En el caso del cliente, la instalación es muy sencilla. Para instalar el fichero Sigue la Flecha.apk, debemos activar la opción “orígenes desconocidos” de la sección de seguridad de nuestro dispositivo móvil, lo que permitirá instalar software aunque no esté firmado, solo en caso de que nos indique algún aviso.

Una vez instalada la aplicación, antes empezar a utilizarla debemos acceder al menú de configuración y establecer la dirección del servidor que se encargará de otorgarnos el acceso a toda la funcionalidad de la aplicación.

5.Quinto Capítulo
Conclusiones

5.1. Conclusiones

Finalmente. a modo de resumen global de todo lo que se ha tratado hasta ahora se pueden dividir las tareas que se han llevado a cabo en este proyecto en tres grandes grupos bien diferenciados.

Estudio sobre los diferentes sistemas operativos móviles y las tecnologías disponibles para el sistema servidor. Aquí se han estudiado los diferentes sistemas operativos móviles y su presencia en el mercado para poder decidir cuál es la mejor opción para garantizar que nuestra aplicación llegue al mayor número de usuarios. Además, también se han tenido en cuenta las restricciones provocadas por el uso de un sistema operativo móvil de código abierto o uno corporativo.

Estudio teórico para llevar a cabo el *Desarrollo de una Aplicación Cliente – Servidor* que utilice las API'S de Google para la obtención de datos sobre rutas y representación visual de éstas. También se ha llevado a cabo un estudio sobre cuál es la mejor forma de realizar el intercambio de datos sobre rutas entre el cliente y el servidor para minimizar el volumen de datos enviado al servidor, lo que mejora el rendimiento de la aplicación y disminuye la cantidad de tráfico en la red.

Búsqueda de las Herramientas Adecuadas para llevar a cabo su implementación. Se han estudiado las diferentes posibilidades que nos ofrece el mundo de la programación y las herramientas externas, cuya combinación ha dado como resultado una aplicación funcional.

Bibliografía y Referencias

Tutorial básico sobre el diseño e implementación de aplicaciones en Android
www.androidcurso.com

Instalación y Configuración del servidor Apache Tomcat.
http://tomcat.apache.org/

Instalación y configuración del entorno de desarrollo Android
http://developer.android.com/sdk/installing/index.html?pkg=tools

Documentación de las API de Google Maps
https://developers.google.com/maps/documentation/android-api/

Documentación de la API de direcciones de Google
https://developers.google.com/maps/documentation/directions/intro

Librería GSON
http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=GsonJavaJSON#038

Librería ViewPager
http://amatellanes.wordpress.com/2013/05/25/android-ejemplo-de-viewpager-en-android-parte-1/