APLICACIÓN ANDROID FARFLY

APP MULTILENGUAJE

Se ha configurado la app para que soporte el inglés y el castellano

El cambio de idioma se ha implementado a nivel local por lo que es independiente del idioma del dispositivo y su gestión la lleva a cabo el usuario sin salir de la aplicación.

Este cambio de idioma puede realizarse desde el menú de opciones de la actividad PrincipalActivity.java y utiliza la clase Idioma.

ARRANQUE DE LA APP

Configuración del idioma en el arranque

Cuando arrancamos la app por primera vez, ésta nos dará la opción de configurar el idioma, para que el usuario no tenga que hacerlo manualmente después. Sin embargo el idioma puede cambiarse también más adelante mediante la opción de "Idioma" del menú de "Opciones" de PrincipalActivity.java

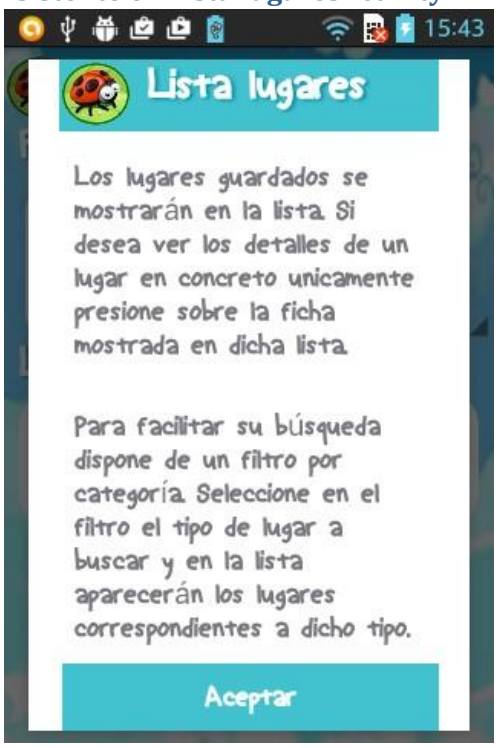
Asistentes de uso

La app incorpora una serie de asistentes para guiar al usuario en las diferentes secciones:

Asistente en MapaLugaresActivity

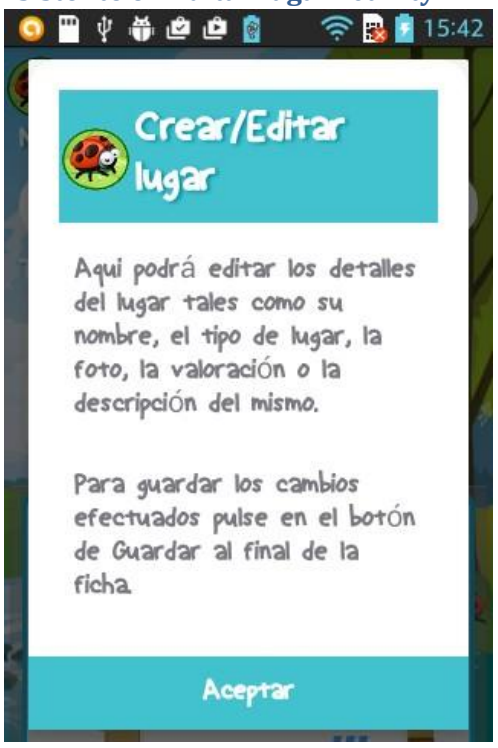
Algunos asistentes incorporan ScrollView para poder ver todo su contenido.

Asistente en ListaLugaresActivity



Asistente en MostrarLugarActivity

Asistente en EditarLugarActivity



APLICACIÓN ADAPTADA AL CAMBIO DE ORIENTACIÓN DE LA PANTALLA

Se ha implementado la app para que su interfaz se ajuste adecuadamente al cambio de orientación de pantalla y que ningún proceso quede interrumpido generando el correspondiente error cuando dicho cambio se produzca.

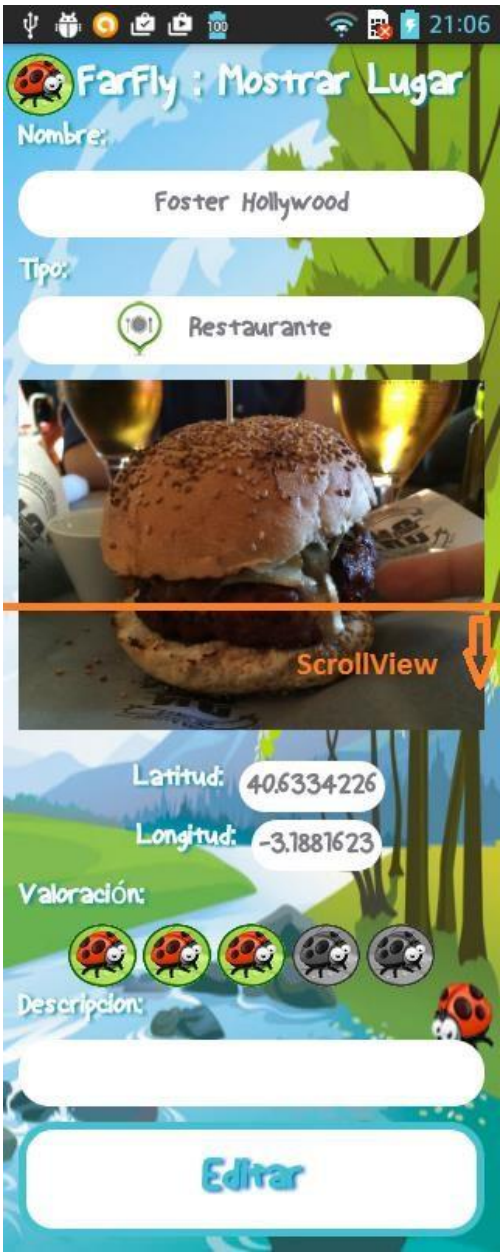
- **PrincipalActivity:**

- **MapaLugaresActivity:**

- **ListaLugaresActivity:**

- **MostrarLugarActivity**

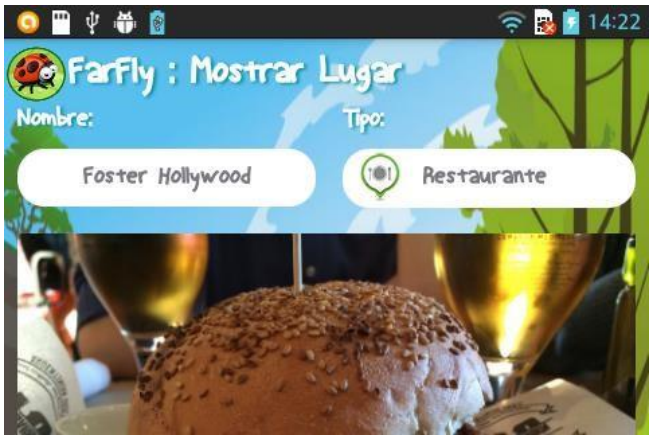
Con el dispositivo en orientación vertical o portrait la vista de la ficha quedaría de la siguiente manera:



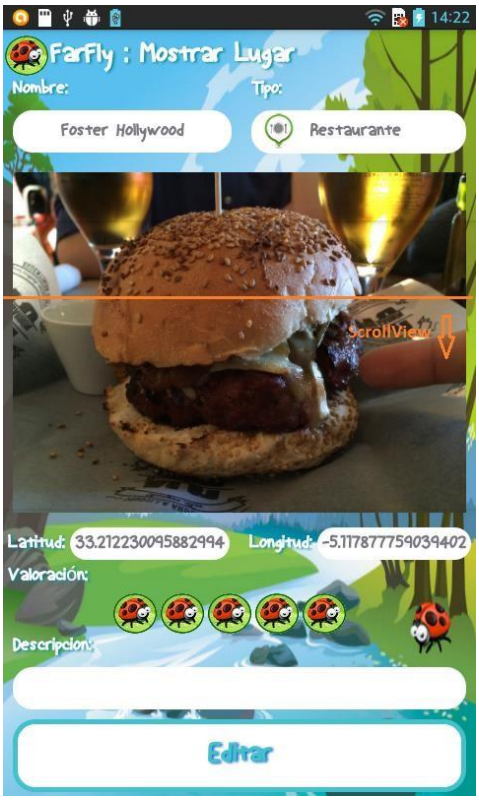
Debido a que la ficha no es visible en su totalidad en la pantalla del dispositivo, ha sido necesario implementar un ScrollView que nos permita desplazarnos hacia abajo tal y como se muestra en la imagen de la derecha.

El ScrollView comenzaría a ser efectivo a partir de la línea naranja que se puede ver en la imagen.

Si nos fijamos en la imagen con orientación de pantalla landscape (horizontal), ésta aparece medio cortada, sin embargo al igual que con la orientación vertical se ha implementado un ScrollView para poder desplazarnos hacia abajo y visualizar toda la ficha del lugar.

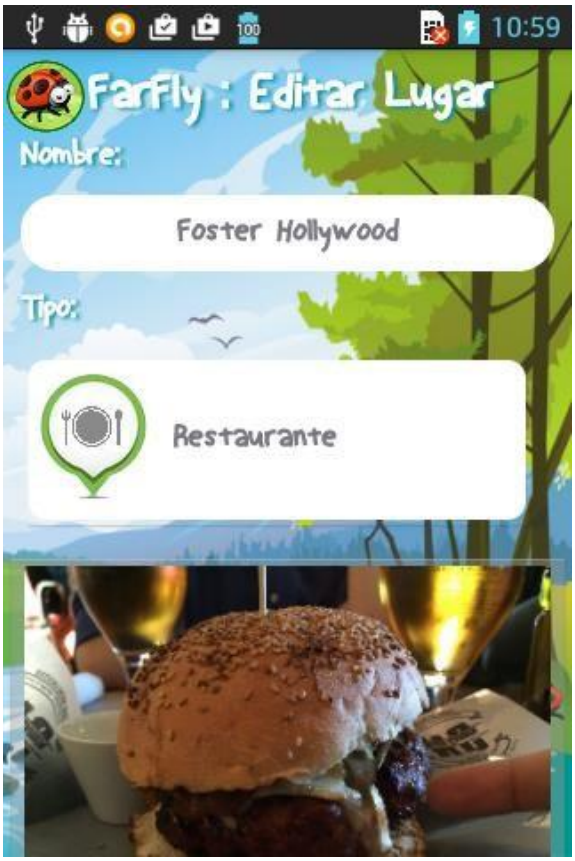


El ScrollView comenzaría a partir de la línea naranja como se puede apreciar en la imagen de la derecha.



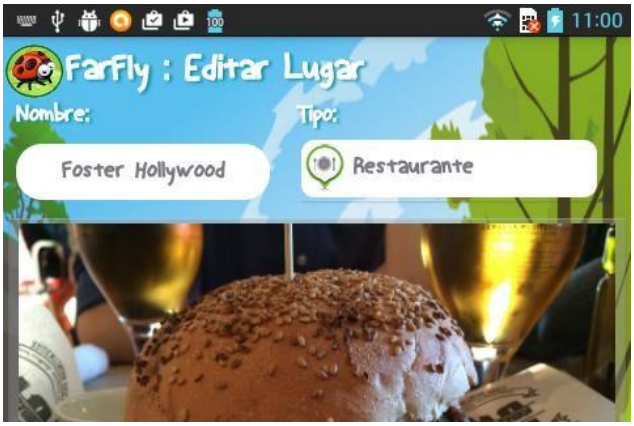
- **EditarLugarActivity**

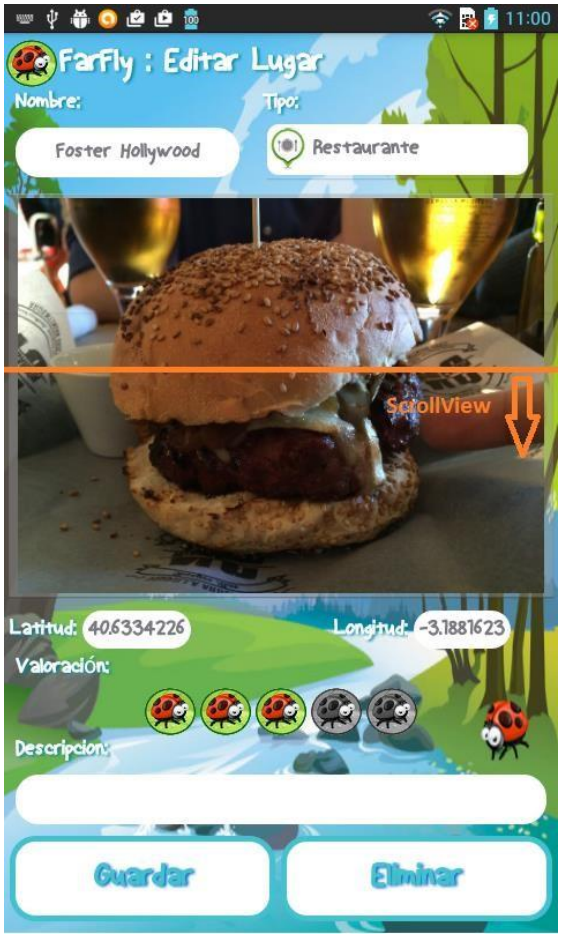
Si visualizamos la actividad con el dispositivo orientado en modo vertical o portrait.



Ocurre lo mismo que en la actividad anterior y por ello implementamos el ScrollView.

Si cambiamos la orientación del dispositivo a landscape u horizontal tendremos la siguiente vista:





La imagen de la izquierda se corresponde a la vista obtenida del dispositivo sin aplicar el ScrollView, mientras que la vista de la derecha sería la visión completa de la ficha.

En la imagen de la derecha podemos ver donde comienza a tener efecto el ScrollView.

PRINCIPALACTIVITY

IMPLEMENTACION DEL BOTÓN "ATRÁS"

Cuando el usuario pulsa el botón "Atrás" del dispositivo, la aplicación lanza un Dialogo flotante con dos botones (aceptar y cancelar), donde le informa al usuario que está a punto de salir de la app. Si el usuario desea seguir adelante y pulsa "aceptar" cerraremos la actividad, si por el contrario pulsa "cancelar" cancelaremos el dialogo.

MENÚ DE OPCIONES

En la esquina superior derecha se ha colocado un icono de ajustes que nos permite abrir el menú de opciones.

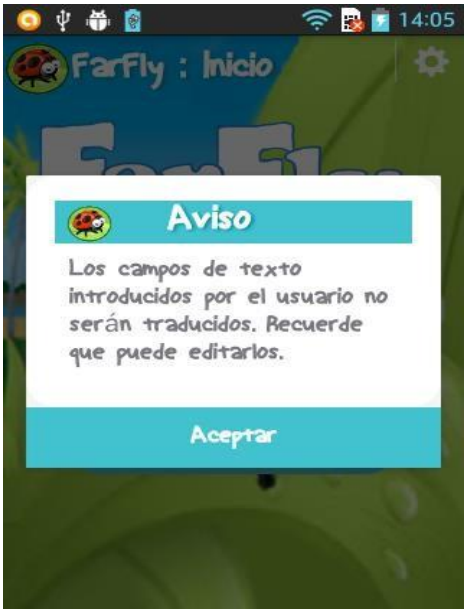
En este menú el usuario podrá acceder a las siguientes secciones:

- **Acerca de la App:** Se abre una ventana con información de la App, tal como, el nombre, la fecha de publicación, la versión mínima de Android que soporta la app, los idiomas disponibles. Esta ventana tendrá un botón de "aceptar" que cerrará el menú.

- **Contacto:** Se abre una ventana con información del desarrollador, tal como, su nombre, su titulación y el email de contacto. Esta ventana tendrá un botón de "aceptar" que cerrará el menú.

- **Mostrar Asistente:** Esta opción nos permite volver a visualizar todos los asistentes que saltan la primera vez que utilizamos la aplicación.

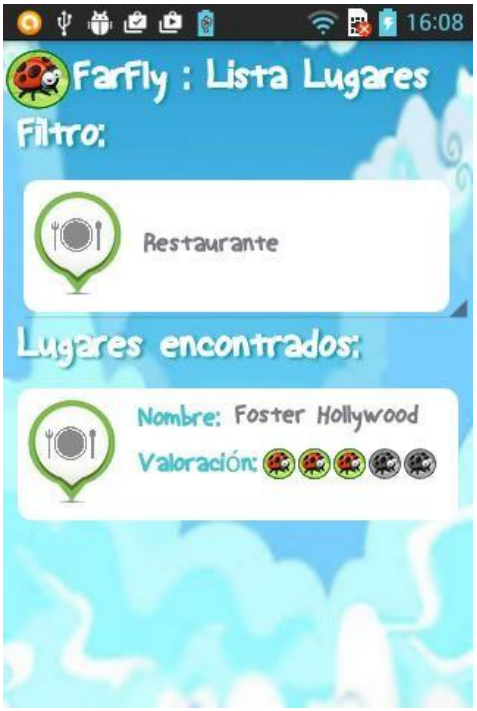
- **Idioma:** Se ha configurado la aplicación como una app multilenguaje que soporta el castellano y el inglés, al presionar en esta sección nos aparecerá un mensaje de aviso donde nos indica que al cambiar el idioma habrá campos de los lugares (que son los correspondientes a los introducidos por el usuario en modo texto) que no serán traducidos . Al presionar en el botón de "aceptar" del mensaje de aviso la app nos abrirá un menú con las dos opciones de idioma. Por ultimo una vez realizado el cambio, un toast personalizado nos indicará que el cambio se realizó con éxito o en caso contrario nos mostrará que hubo problemas.



LISTALUGARES ACTIVITY

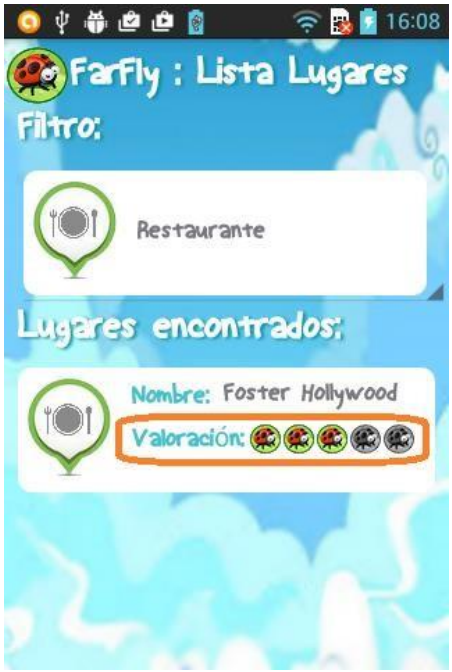
CAMBIO DEL CONTENIDO EN LA LISTA DE LUGARES

Según las exigencias del proyecto era necesario que en el listado de lugares apareciera el nombre del lugar y su descripción. En vez de esto he incorporado el nombre, un icono que identificará el tipo de lugar y un RatingBar que nos mostrará la valoración que le dimos al mismo. Para ello ha sido necesario implementar la clase LugaresAdapter.java



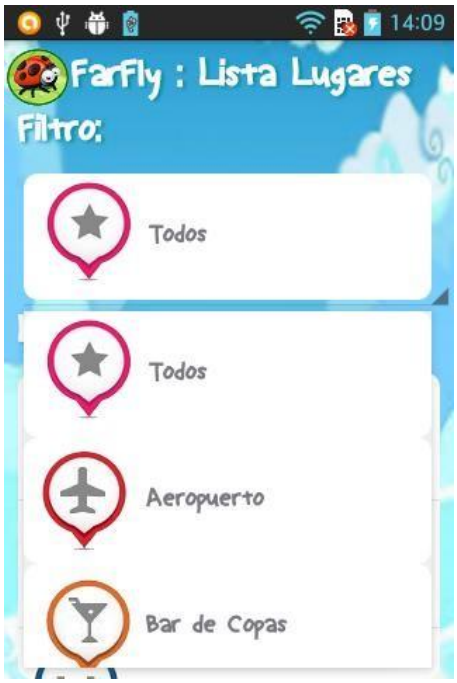
RATINGBAR PERSONALIZADO

El RatingBar de los ítems de la lista de lugares es personalizado y muestra el logo de la app como medida de valoración.



CREACIÓN DE UN SPINNER PERSONALIZADO

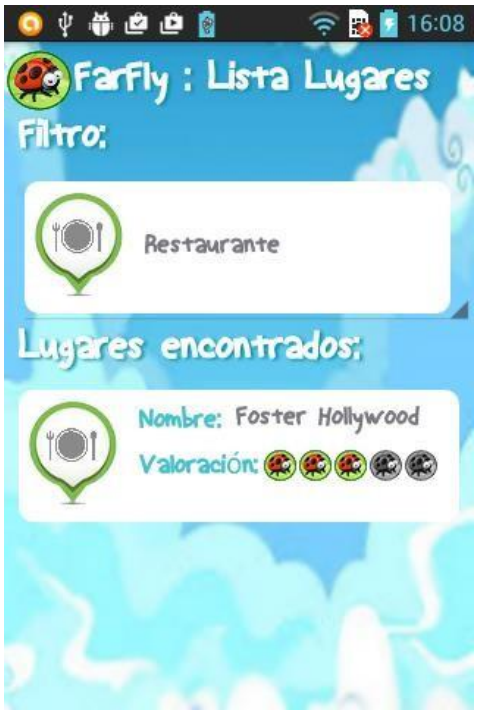
Para la realización del filtro se ha creado un spinner totalmente personalizado que consta de un icono personalizado y un texto. Ambos corresponden a los tipos de lugares que la app acepta. Para ello ha sido necesario implementar la clase SpinnerAdapter.java



He creado un filtro para la lista de lugares a través de un spinner personalizado.

El spinner contendrá un icono y el tipo de lugar que se corresponde con dicho icono (por ejemplo, restaurante u hotel). Se han introducido algunos tipos de lugares, la posibilidad del tipo “otros” por si el tipo de lugar deseado no estuviera en el listado del spinner y la posibilidad de “todos”.

Cuando se elija un tipo en el filtro en el listado de lugares solo aparecerán aquellos lugares que se correspondan con dicho tipo.



Si se selecciona la opción de "todos" se mostrarán todos los lugares almacenados en la app. Si no existiera ningún lugar almacenado se mostraría un texto indicando al usuario donde ir para crear un lugar.

Si no hubiera coincidencia con el tipo de lugar seleccionado en el filtro se mostraría un ítem especial que nos indicaría que no se han encontrado lugares de este tipo guardados y si el usuario decidiese pinchar en él, aparecería un toast indicándole que debe cambiar el tipo de lugar en el filtro.

La funcionalidad de dicho filtro se encuentra en la clase LugaresAdapter.java

EDITARLUGARACTIVITY

CREACIÓN DEL CAMPO TIPO A TRAVÉS DE UN SPINNER PERSONALIZADO

Para seleccionar el tipo de lugar se ha creado un spinner personalizado de las mismas características que en ListaLugaresActivity.

Este spinner será traducido cuando se produce un cambio de idioma, y permitirá al usuario seleccionar el tipo de lugar que está creando/editando. Además el icono del spinner será utilizado más adelante como marcador del lugar en el mapa.

La imagen ha sido mostrada del dispositivo con orientación landscape.

CREACIÓN DEL CAMPO VALORACIÓN

Se ha creado un campo nuevo a través de un RatingBar personalizado y de mayor tamaño que el que aparece en ListaLugaresActivity.java para que el usuario pueda calificar su nivel de satisfacción con el lugar creado. Teniendo como máxima valoración un 10 y correspondiendo a cada "mariquita" un valor de 2, el usuario podrá calificar con un 10 el lugar rellenando con el dedo las 5 mariquitas posibles que presenta el RatingBar.

La imagen ha sido mostrada del dispositivo con orientación landscape.

OBTENCIÓN DE IMAGEN

La app permite al usuario añadir una foto en la ficha del lugar. El usuario puede decidir no hacerlo de forma que la imagen que se guardará es una por defecto que indica al usuario que no hay foto con un texto que será traducido en función del idioma de la app.

Si por el contrario el usuario desea adjuntar una fotografía tendrá dos opciones:

- Tomarla con la cámara de fotos.
- Tomarla de la galería a través de uno de los proveedores de imagen del dispositivo.

Para ello se ha implementado un menú que saltará cuando presionemos en la imagen:

Si decidimos pulsar en "Ir a Cámara de Fotos" nos dirigirá a la cámara y si decidimos pulsar "Ir a Galería" nos permitirá elegir el proveedor de contenido y una vez elegido nos llevará a la Galería correspondiente:

Es necesario tratar la imagen obtenida antes de incorporarla a la ficha del lugar para evitar los problemas típicos de los Bitmap , para ello se ha implementado la clase ImagenSupport.java.

AVISOS

La app a la hora de crear/editar un lugar dispone de varios avisos de guía para el usuario:

- **Aviso datos incompletos:** Este aviso salta cuando el usuario presiona el botón de "Guardar" para comunicarle que hay campos del lugar que están sin rellenar y le pregunta que si es su deseo continuar creando/editando el lugar sin rellenarlo. Si es así el lugar se crea con dichos campos vacíos y el usuario no desea continuar se cierra el aviso y permite que este los rellena.

- **Aviso falta categoría:** Este aviso salta y no permite crear el lugar si el usuario no selecciona el tipo del mismo, ya que es necesario para aplicarle el icono en MapaLugaresActivity.java.

- **Aviso eliminar lugar:** Este aviso salta cuando el usuario presiona el botón de "Eliminar" para indicarle que está a punto de eliminar el lugar en cuestión y si ese es su deseo. Si el usuario sigue adelante el lugar se borrará y saltará un toast indicando que todo fue bien y que para crear otro lugar debe ir a la opción de "Mapas" o por el contrario si hubo algún problema lo notificará.

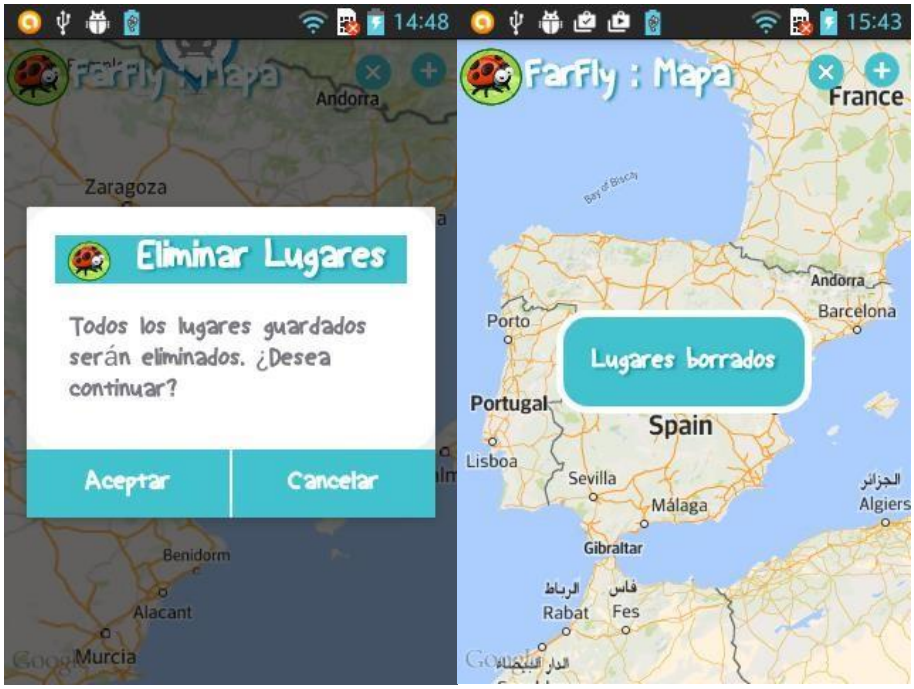
MAPALUGARESACTIVITY

MARCADORES

Los marcadores que indican el lugar en el mapa serán los iconos correspondientes con el tipo lugar que el usuario selecciono en el spinner cuando lo creo. Además si el usuario pulsa sobre dicho marcador podrá ver el nombre del lugar y su id en la base de datos.

BOTON ELIMINAR TODOS LOS LUGARES

Se ha creado una especie de action bar que indica que nos encontramos en la opción de mapa. En esta cabecera se ha implementado un botón con el icono “x”, este botón nos permite borrar todos los lugares guardados en la base de datos. Para lo cual la app primero comprueba que haya lugares, de no haber se lo indica al usuario. Si hay lugares salta un aviso para indicar al usuario que va a borrar todos los lugares preguntándole si ese es su deseo. Si acepta los lugares serán borrados de la base de datos y se mostrara un toast de confirmación. Si cancela se cerrará el aviso y no se seguirá adelante con el borrado.



BOTÓN CREAR LUGAR CON LA UBICACIÓN ACTUAL

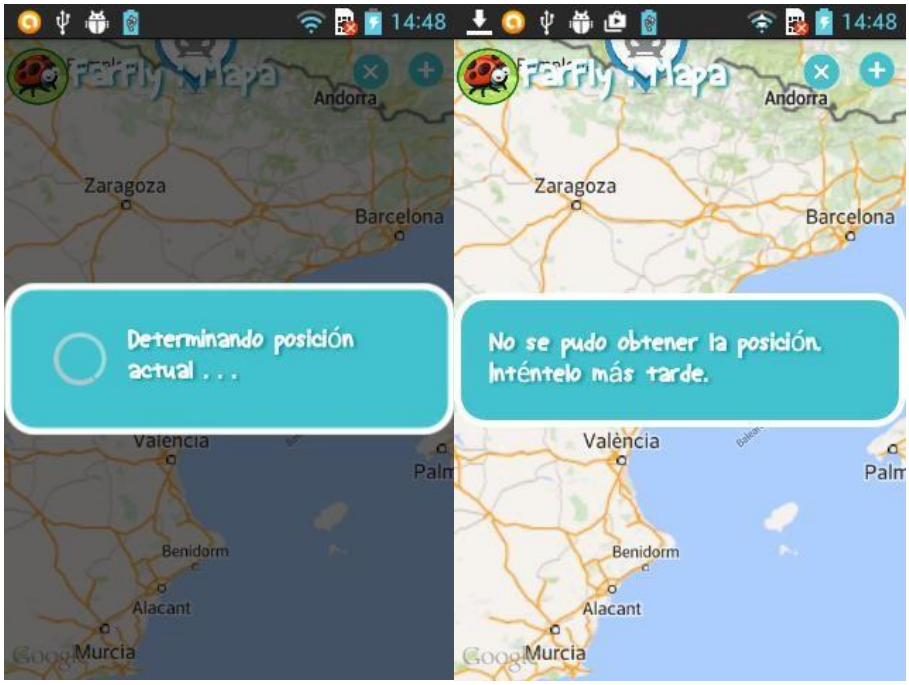
En la misma cabecera que el anterior botón tenemos otro con el icono "+", con este botón el usuario puede crear un lugar con las coordenadas actuales del dispositivo.

La app lo primero que hace es comprobar que existen proveedores de ubicación activos. De no ser así se lo comunica al usuario y le da la opción de activarlos mediante un aviso.

Si pulsamos en el botón de "Aceptar" nos llevará a la parte de ajuste del dispositivo donde podemos activar los proveedores de ubicación.

Por el contrario si presionamos “Cancelar” el aviso se cerrará.

Una vez que están activos y el usuario sigue queriendo crear el lugar se procede a la obtención de la ubicación. Para no tener al usuario esperando a ciegas se ha implementado una tarea asíncrona que gestiona el tiempo de búsqueda y muestra una barra de progreso personalizada al usuario. Si en aproximadamente 10s no se obtuvo localización se cierra el proceso de búsqueda, se le comunica al usuario que no se pudo obtener la localización y que pruebe a intentarlo más tarde. Si por el contrario en ese tiempo se obtiene localización se lanza EditarLugarActivity.java con las coordenadas obtenidas para crear el lugar.



La obtención de la localización se ha implementado de tal manera que puede ser cancelada por el usuario antes de esos 10s, para lo cual únicamente tendrá que presionar el botón “Atrás” de su dispositivo. Esto cerrará la barra de progreso, terminará la tarea asíncrona y detendrá la búsqueda.

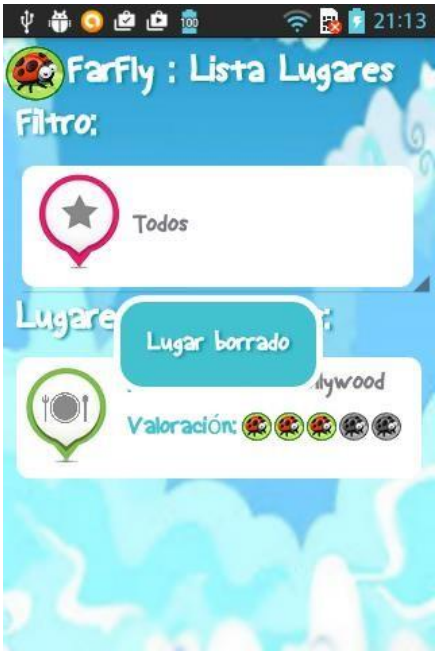
Para llevar a cabo la obtención del posición actual se utiliza la clase LocationHelper y la tarea asíncrona LocationControl que se encuentra en la actividad MapaLugaresActivity.java.

AVISOS SIN CONEXIÓN DE RED

Cuando el usuario accede a la opción de mapas lo primero que hace la app es comprobar si el dispositivo posee conexión a internet, de no ser así los mapas no se cargarán adecuadamente por lo que se mostrará un aviso personalizado al usuario indicándole que no posee conexión. Para saber si poseo conexión de red se utiliza la clase InternetConexion.

DISEÑO

- **Toast Personalizado:** Todos los Toast que aparecen en la app son personalizados manteniendo la estética de la app.



- **Menús Personalizados:** Todos los menús de la app siguen un estándar de estilo personalizado como el que se muestra a continuación.

- **Avisos Personalizados:** Todos los avisos de la app siguen un estándar de estilo personalizado como el que se muestra a continuación.

- **Creación de una action bar personalizada:** Creación de cabecera indicando en que parte de la app nos encontramos.

- **Letra**

personalizada en todos los textos de la app: Para ello se ha importado una letra personalizada a través del archivo Animated.ttf que se ha incluido en la carpeta *assets/fonts* de la aplicación.

- **Clases botón, texto y edit propias:** Se han implementado 3 clases para generar TextView, EditText y Button personalizados.
  Estas clases son: MiButton, MiEditText y MiTextView.

- **Botones con cambio de color al pulsar:** Si nos fijamos en la foto, los botones de toda la app presentarán la siguiente imagen, siendo el botón de la derecha la vista del botón sin pulsar y la de la izquierda la del botón pulsado.

- **Creación de Logo de la app:** Se ha creado un logo personalizado para la aplicación.

OPTIMIZACIONES Y CLASES

Clase Soporte:

Esta clase se utiliza para dar soporte a toda la app, gracias a ella podemos reutilizar mucho código e unificar funciones.

En ella se han implementado métodos muy dispares que cumplen funciones muy distintas:

Métodos para las Preferencias compartidas:

- `public boolean leerPreferencia(String flag)`

  Este método permite leer las preferencias guardadas en el archivo MisPreferencias.xml que contiene los flags de todos los asistentes de la app.

  Como parámetro de entrada recibe el nombre del asistente al que se desea acceder.

  Devuelve un true en caso de estar activado el asistente y false en caso contrario.

- `public void guardarPreferencia(String flag,boolean valor)`

  Este método permite guardar las preferencias en el archivo MisPreferencias.xml que contiene los flags de todos los asistentes de la app.

  Como parámetro de entrada recibe el nombre del asistente al que se desea modificar y su valor.

Métodos para la gestión de la imagen.

- `public ImagenSupport getImagenSuport()`

  Este método permite obtener la clase ImagenSupport que se encarga de gestionar la foto del lugar y que detallamos más adelante.

- `public String getUri_foto_cargar()`

  Este método se utiliza para obtener el String correspondiente a la uri de la foto que se va a cargar. Se llama en el método LeerCampos() de la actividad EditarLugarActivity.java

- `public void setUri_foto_cargar(String uri_foto_cargar)`

  Este método se utiliza para guardar el string de la uri de la foto a cargar obtenida de la consulta a la base de datos. Se utiliza dentro del método ConsultaBD de la clase Soporte.

Métodos para el estilo y los toast:

- `public void setEstilo(TextView texto,Button boton, Typeface letra,int size, int color)`

  Este método se encarga de establecer el estilo al texto de forma dinámica, esto es en tiempo de ejecución, tanto para el texto en si como para el texto de botones. Se utiliza en la clase LugaresAdapter para cambiar el estilo a la ficha cuando no se encontraron lugares tras aplicar el filtro y para devolver la apariencia original a la ficha.

  También se usa en la clase DialogoInfo para personalizar la apariencia de los avisos.

  Como parámetros de entrada tenemos:

  - **TextView texto**: Objeto de tipo TextView al que se le va a dar estilo.En caso de querer customizar un botón este campo será null.
  - Button boton: Objeto de tipo Button al que se le va a dar estilo.En caso de querer customizar un texto este campo será null.
  - **Typeface letra**: El tipo de letra personalizada a utilizer para dar estilo.
  - **int size**: Tamaño seleccionado para la letra.
  - **int color**: Identificador del color seleccionado para dar estilo.

- `public void CrearToast(String texto)`

  Este método se utiliza para crear y mostrar por pantalla toasts personalizados de acuerdo a la estética de la app.
  Como argumento de entrada se le pasa el texto que mostrará el toast.

Métodos para gestionar el spinner:

- `public void AñadirItemsSpinner(String tipo,Spinner spinner_tipo)`

  Este método se utiliza para añadir los elementos al spinner “tipo”. Este método es llamado por la actividad EditarLugarActivity.java y

ListaLugaresActivity.java.
Como parámetros de entrada tenemos:

- **String desde**: Indicará desde donde es llamado el método, podrá tomar los valores de "editar" si es llamado desde EditarLugarActivity.java o "listar" si es llamado desde ListaLugaresActivity.java.
- **Spinner spinner_tipo**: El spinner al que se le añadirán los elementos.

- `public void setListaTipo(ArrayList<String> customList)`

Este método se utiliza para guardar la lista de posibles tipos de lugares que soporta la app. Se utiliza en el método anterior dentro de la clase Soporte.

- `public ArrayList<String> getListaTipo()`

Este método se utiliza para obtener el listado de los posibles tipos de lugar que soporta la app. Se utiliza en la actividad EditarLugarActivity.java para crear el objeto Layout correspondiente.

- `public int SelectorDeLogo(String categoria, String tamanyo)`

Este método se utiliza para asignar el logo correspondiente al tipo de lugar en el spinner. Es por ello que este método se utiliza en la clase SpinnerAdapter.

Como parámetros de entrada tenemos:

- **String categoría**: Es el texto correspondiente al tipo de lugar.
- **String tamanyo**: Puede tomar como valores "normal" y "small" para indicar que el drawable que debe elegirse como icono o logo debe ser el correspondiente al tamaño "normal" o "pequeño" respectivamente

Devuelve un entero correspondiente al identificador del drawable que se corresponde con el tipo o categoría y tamaño que se pasan como argumentos.

- `public int buscarTraducción(String texto)`

  Este método se utiliza para traducir el texto del tipo de lugar obtenido en la consulta a la base de datos para que al cargar el valor en el spinner no se produzca un error si es que el lugar se guardó con una configuración de idioma diferente a la de la app en el momento de la consulta. Este método se utiliza en el método ConsultaBD.

  Como parámetro de entrada tenemos un String texto que se corresponderá con el tipo de lugar que se va a traducir.

  Como parámetro de salida tendremos el entero que se corresponde con el identificador de texto de dicho tipo.

Métodos y clase interna para mostrar/editar un lugar

- `public boolean MostrarLugar(String accion,int id, Layout layout)`

  Este método se utiliza para mostrar un lugar. Es compartido por las actividades MostrarLugarActivity.java y EditarLugarActivity.java (esta última lo utiliza cuando se quiere editar el lugar, para cargar los datos que estaban guardados y evitarle al usuario rellenar todos los datos nuevamente)

  Como parámetros de entrada tendremos:

  - **String accion**: cuyos valores podrán ser "mostrar" cuando ejecutemos MostrarLugarActivity.java o "editar" cuando ejecutemos EditarLugarActivity.java.
  - **int id**: Indentificador correspondiente al lugar.
  - **Layout layout**: El layout donde se deberán cargar los datos que será diferente según la actividad que se esté ejecutando.

  Devuelve "true" en caso de que se pueda mostrar correctamente el lugar y un "false" en caso contrario.

- `public Consulta ConsultaBD(String seleccion,int id)`

  Este método se utiliza para hacer consultas al proveedor de contenido sobre los lugares guardados. Las consultas podrán realizarse de todos los lugares o de un lugar en concreto. Se utiliza dentro del método "MostrarLugar".

Como parámetros de entrada:

- **String seleccion**: Será "todos" cuando se quieran obtener los datos de todos los lugares guardados o "uno" cuando se quieran obtener los datos de un lugar en concreto.
- **int id**: Será -1 cuando queramos hacer una consulta de todos los lugares o el identificador del lugar cuando se quiera consultar un lugar en concreto.

Devuelve una clase de tipo Consulta que explicamos más adelante.

- `public Lugar ObtenerImagen(Lugar lugar)`

Este método se encarga de rellenar el campo Imagen del objeto de tipo Lugar que se le pasa como argumento. El campo que rellena se corresponde con la imagen del lugar ya tratada y en formato Bitmap. Este método se utiliza dentro del método "MostrarLugar".

Como parámetro de entrada se le pasa un objeto de tipo Lugar con todos los datos del lugar obtenidos en la consulta salvo el campo Imagen.

Devuelve el objeto pasado como argumento completo con la imagen del lugar.

- `public void CargarValoresLayout(String accion,Lugar lugar,Layout layout)`

Este método se encarga de cargar todos los datos que conforman el objeto Lugar pasado como argumento en el Layout que también se pasa como argumento.Se utiliza en el método "MostrarLugar".

Como parámetros de entrada tenemos:

- **String accion**: Toma como valores "mostrar" o "editar" en función de si hemos llamado al método desde MostrarLugarActivity.java o desde EditarLugarActivity.java. Es necesario porque dependiendo de la actividad desde donde se llame los elementos del layout a rellenar son diferentes.
- **Lugar lugar**: Es el objeto de tipo Lugar que contiene toda la información del lugar que va a ser cargado.
- **Layout layout**: Es el layout donde se cargarán todos los datos del lugar y que variará en función de la actividad desde donde se llamó al método "MostrarLugar".

- `public class Consulta`

Cuando llamamos al método ConsultaBD este devuelve un objeto de este tipo. Es una clase interna que consta de dos elementos:

- **ArrayList<Lugar> lugares**: Este elemento consiste en un ArrayList de objetos de tipo Lugar y se utiliza cuando realizamos una consulta de todos los lugares guardados en la base de datos. La consulta a la base de datos devolverá este tipo de dato relleno con todos los lugares y el elemento Lugar vacío.

- **Lugar lugar**: Este elemento es un objeto Lugar que se utiliza cuando queremos consultar los datos de un único lugar. La consulta a la base de datos devolverá este tipo de dato relleno con los datos del lugar y el ArrayList vacío.

Además de estos dos elementos la clase incluye los getters and setters de ambos elementos.

Método para eliminar uno o todos los lugares de la base de datos:

- `public void EliminarLugares(String seleccion,String desde,int id)`

Este método se utiliza en dos lugares diferentes de la app:

Opción 1: Se encarga de borrar un lugar determinado desde EditarLugarActivity.java

Como parámetros de entrada:

- selección=”uno” indicando que solo borraremos un lugar.
- desde=”lista” para indicar que se borra desde EditarLugarActivity.java que fue llamado desde ListaLugaresActivity.java, de esta manera la app sabe que deberá mostrar el toast de “Si quiere crear nuevos lugares vaya a la opción de mapa”.
- id=”entero” siendo “entero” el id del lugar que se desea borrar.

Opción 2: Se encarga de borrar todos los lugares de la base de datos desde MapaLugaresActivity.java

Como parámetros de entrada:

- selección=”todos” indicando que borraremos todos los lugares.
- desde=”mapa” para indicar que se borra desde MapaLugaresActivity.java
- id=-1 indicando que este parámetro aquí no es válido.
-

Método para la creación de la clase DialogoInfo:

- `public DialogoInfo nuevoDialogo(int opcion)`

  Este método es el constructor personalizado de la clase DialogInfo (que explicamos más adelante), esta clase se encarga de mostrar los asistentes de uso y muchos de los avisos de la app.

  Como parámetro de entrada tenemos el entero opción que indicará que tipo de aviso queremos mostrar.

  Devuelve la clase DialogoInfo creada.

Asistente de ListaLugaresActivity.java:

- `public void AsistenteListar()`

  Este método se encarga de mostrar el asistente correspondiente a la opción de listar lugares.

  Se crea en soporte y no a través de la clase DialogoInfo como el resto de asistentes debido a que la actividad ListaLugaresActivity.java extiende de ListActivity y no de FragmentActivity y no soporta la clase DialogoInfo. Esto es debido a que esta última extiende de DialogFragment y solo se puede mostrar en actividades extendidas de FragmentActivity, es por ello que es necesario crearlo aparte.

  Se podría haber implementado dentro de la propia actividad y no de la clase Soporte pero me parecía más limpio hacerlo de esta manera.

Clase DialogoInfo:

Esta clase se utiliza para reutilizar código a la hora de mostrar avisos de tipo informativo que únicamente necesitan del botón “aceptar”.

Para ello se ha modificado su constructor que se encuentra en la clase Soporte añadiéndole un entero como parámetro de entrada que nos indicará que tipo de aviso es el que queremos construir. En función de este parámetro “opción” se selecciona un layout u otro en la construcción del aviso y se muestra.

De esta manera nos evitamos tener que estar repitiendo código para cada aviso que creemos.

Clase ImgenSupport:

Esta clase se utiliza para el tratamiento de las imágenes y se llama a través de la clase Soporte. Dispone de 3 métodos:

- `public Bitmap decodeUri(Uri uri)throws FileNotFoundException`

  Devuelve el Bitmap de la foto escalada y ajustada a la pantalla del dispositivo. Este método se encarga de obtener la imagen del lugar escalada y ajustada a la pantalla, evitando errores del tipo ErrorOutOfMemory muy típicos en el uso de Bitmap, debido al poco espacio de memoria del que disponen los dispositivos para gestionar imágenes.

  Como parámetro de entrada recibe la Uri de la foto correspondiente al lugar que guardamos como un String en la base de datos

  Devuelve el Bitmap de la foto escalada y ajustada a la pantalla del dispositivo.

- `private Bitmap scaleImage(int anchura, Bitmap imagen)`

  Este método que es llamado en el método anterior es el encargado de realizar el escalado de la imagen para que se ajuste a la pantalla, lo que se ha hecho es fijar como referencia el ancho de la pantalla del dispositivo y escalar la altura en función de dicho ancho para evitar deformar la imagen y permitir así una correcta visualización de la imagen en la app.

  Como parámetros de entrada recibe la imagen Bitmap decodificada a partir de la uri y la anchura de la pantalla del dispositivo.

  Devuelve la imagen del lugar en formato Bitmap escalada y ajustada a la pantalla.

- `private int dpToPx(int dp)`

  Este método lo utiliza el método anterior para pasar la densidad física de la pantalla a píxeles.

  Recibe como parámetro de entrada los dp de la pantalla y devuelve los píxeles correspondientes a esos dp.

Clase Layout:

Esta clase se utiliza para guardar todos los elementos del layout que componen MostrarLugarActivity y EditarLugarActivity. Posee dos constructores uno para cada una de las actividades anteriores. El objetivo de esta clase es la optimización de código.

Cuando mostramos un lugar debemos mostrar todos sus datos al igual que cuando editamos dicho lugar es necesario mostrar los valores que había guardados por si el usuario no desea cambiarlos todos. Al crear una única función que nos muestra el lugar para las dos actividades es necesario tener un clase Layout que se le pasara como argumento a la función "MostrarLugar" de la clase Soporte indicando en que layout debemos mostrar los datos. Este layout variará en función de si estamos en la opción de mostrar el lugar o de editarlo.

Clase Lugar:

Esta clase se utiliza para guardar los datos obtenidos de la consulta a la base de datos a través del proveedor de contenido, al igual que la clase anterior se utiliza para optimizar. Al guardar los datos del lugar en esta clase evitamos tener que realizar una consulta a la base de datos cada vez que necesitemos algún dato del lugar.

Clases MiButton, MiEditText y MiTextView

Estas tres clases han sido implementadas para poder dar un utilizar un estilo de letra personalizado a través de un archivo importado del tipo ttf en los botones, los textos y los textos editables, respectivamente, a través de etiquetas xml, evitándonos tener que hacerlo en tiempo de ejecución y de esa manera mucho código repetitivo.

De la misma forma para implementar la clase MiButton ha sido necesario implementar otras dos clases:

- **CustomFontHelper**: Esta clase sirve para aliviar de código a la clase MiButton realizando la asignación de letra y las comprobaciones necesarias para que se ejecute adecuadamente y dejando que esta únicamente posea dos constructores.
- **FontCache**: Se utiliza para reducir el uso de memoria en dispositivos antiguos, ya que estos no poseen demasiada y podemos encontrarnos con algún error de memoria. Se utiliza dentro de la clase anterior.

NOTAS ACERCA DE LAS IMÁGENES UTILIZADAS EN LA APP

En cuanto a las imágenes utilizadas para dar estilo a la aplicación, la idea original fue crearlas personalmente mediante programas del tipo Adobe Illustrator o Adobe Photoshop.

Sin embargo y debido a que el proyecto no pretende demostrar mis conocimientos con estos programas y a que la elaboración de las mismas puede llevarme tiempo, tomé la decisión de utilizar imágenes que se aproximarán a la idea que yo tenía de como quería que la app se viera, pero que por motivos de copyright no puedo utilizar si subo la aplicación al PlayStore, ya que algunas de ellas, como digo, están bajo derechos de autor.

Debido a esto informó que si decido, subir la aplicación al PlayStore crearé mis propias imágenes similares a las que están y las sustituiré por aquellas que no podría utilizar.

NOTAS ACERCA DEL DISPOSITIVO DE PRUEBA

Toda la aplicación ha sido probada en un dispositivo móvil LG E610 OPTIMUS L5

Adjunto las especificaciones básicas y de pantalla del dispositivo:

LG OPTIMUS L5, SMARTPHONE COMPACTO, PANTALLA TÁCTIL DE 4,0'' Y GRAN DISEÑO

E610 Optimus L5

EXPANDIR TODAS LAS ESPECIFICACIONES

ESPECIFICACIONES BÁSICAS

Diseño	Full Touch Screen
Frecuencia	HSDPA 7.2 Mbps/GSM Quad/3G Dual
Dimensiones alto x Ancho x Profundidad (mm)	114 x 67 x 9.6
Sistema Operativo	Android 4.0 Ice Cream Sandwich
Chipset	MSM7225A (Cortex A5)
Velocidad del procesador	800MHz
Batería	1540mAh Li-Ion (typ) / 1500mAh (min)
Peso (incluida batería standard) (gr)	123

PANTALLA

Táctil	Sí
Tipo de pantalla	TFT LCD, Capacitive Touch HVGA
Tamaño de pantalla	4"
Resolución de pantalla (px)	HVGA (320 x 480) – 146ppi
Nº Colores de pantalla	16M

Por último reseñar que todas las imágenes presentadas en el documento han sido tomadas con la aplicación ejecutándose en este dispositivo y para asegurarnos que en versión Froyo la aplicación se veía correctamente se ha utilizado un emulador virtual de Android Studio (aunque con este la parte de mapas no ha podido probarse).