

# Matías Macaya L.

Estudiante de Ingeniería Civil Eléctrica, Universidad de Chile.  
keny92@gmail.com  
+569 9087 2550

## LeBike App

Revisión Técnica y Documentación

### 1. Resumen

En el contexto de SmartCities, una de las formas de crear la nueva “ciudad inteligente”, es a través de saber que está realmente pasando en la ciudad, en los espacios públicos. Muchas veces esto deriva en sensar que está ocurriendo en la ciudad a través de métodos subjetivos y concretos como sensores, y llenar así grandes bases de datos con esta información.

Se desea obtener entonces gran cantidad de datos de la ciudad para luego ser procesados e interpretados. Esta información permite encontrar entre otras cosas, dos principales: (1) fenómenos o comportamientos interpretables a simple vista pero antes desconocidos, y (2) a través de técnicas de análisis de datos develar patrones o relaciones antes escondidas dentro de la ciudad.

Pero SmartCities no es solo sensar y procesar, es darle a la gente acceso a esta información, acceso a expresar además su opinión sobre cómo ven y viven la ciudad. En específico el proyecto se enfoca en el ciclista urbano y en medir ahora su experiencia en la ciudad, poder recabar información subjetiva de cómo este se desarrolla en la ciudad y poder generar una plataforma donde esta información pueda ser utilizada por otros ciclistas para generar así un uso colectivo de la bicicleta en la ciudad más consciente y activa.

LeBike App se desarrolla como una plataforma que pretende medir la experiencia del ciclista en la ciudad. Trabaja en conjunto con un clicker bluetooth que permite al ciclista marcar como positivo o negativo el lugar por el que está transitando. La aplicación debe motivar por sí misma a los usuarios a utilizarla de manera que se distribuya su uso por la utilidad que entrega.

Además de esto, se convierte en el enlace del ciclista con la comunidad y la herramienta para plasmar su opinión e informar a los demás que está ocurriendo.

# Indice

<b>1. Resumen</b>	<b>1</b>
<b>2. Aplicación</b>	<b>4</b>
2.1. Aspectos Técnicos	4
2.2. Flujo de la aplicación	4
<b>3. Documentación</b>	<b>6</b>
3.1. Actividades	6
3.1.1. LeBikeActivity	6
3.1.1.1. ¿Qué hace?	6
3.1.1.2. Trabaja con	6
3.1.1.3. Ciclo de Vida	6
3.1.1.4. Métodos	7
3.1.1.5. To Do's	7
3.1.1.6. Observaciones	8
3.1.2. MisDestinosActivity	8
3.1.2.1. ¿Qué hace?	8
3.1.2.2. Trabaja con	8
3.1.2.3. Ciclo de Vida	8
3.1.2.4. Métodos Importantes	9
3.1.2.5. To Do's	10
3.1.2.6. Observaciones	10
3.1.3. MisAlertasActivity	10
3.1.3.1. ¿Qué hace?	11
3.1.3.2. Trabaja con	11
3.1.3.3. Ciclo de Vida	11
3.1.3.4. Métodos Importantes	12
3.1.3.5. To Do's	13
3.1.3.6. Observaciones	13
3.1.4. EnRutaActivity	13
3.1.4.1. ¿Qué hace?	14
3.1.4.2. Trabaja con	14
3.1.4.3. Ciclo de Vida	14
3.1.4.4. Métodos Importantes	16
3.1.4.5. To Do's	17
3.1.4.6. Observaciones	18
3.2. Fragments	18
3.2.1. DestinoEditDialog	19

3.2.1.1. Funciones	19
3.2.1.2 Detalles	19
3.2.2. AlertaEditDialog	19
3.2.2.1. Funciones	20
3.2.2.2. Detalles	20
3.2.3. MisAlertasCompletasTab	20
3.2.3.1. Funciones	20
3.2.3.2. Detalles	21
3.2.4. MisAlertasPendientesTab	21
3.2.5. MisAlertasTodasTab	21
3.2.6. EnRutaAuxiliaryBottomBar	21
3.2.6.1. Funciones	21
3.2.6.2. Detalles	21
3.3. Services	21
3.3.1. EnRutaTrackingService	21
3.3.1.1. Funciones	22
3.3.1.2. Funcionamiento	22
3.3.1.3. To Do's	22
3.4. Adapters	22
3.4.1. DondeVasAdapter	22
3.4.2. MisDestinosAdapter	23
3.4.3. MisAlertasAdapter	23
3.4.3.1. To Do's	23
3.4.4. MisAlertasPagerAdapter	23
3.5. Clases	23
3.5.1. Destino	24
3.5.1.1. Campos	24
3.5.1.2. Constructores	24
3.5.1.3. Métodos	24
3.5.2. Alerta	24
3.5.2.1. Campos	24
3.5.2.2. Constructores	25
3.5.2.3. Métodos	25
3.5.3. Ruta	25
3.5.3.1. Campos	25
3.5.3.2. Constructores	26
3.5.3.3. Métodos	26
3.6. Interfaces	26
3.6.1. MisAlertasInterfaces	26
3.7. Utils	26

3.7.1. DatabaseHandler	26
3.7.1.1. Definición de Tablas	27
3.7.1.2. Métodos de Escritura	28
3.7.1.2.1. Ruta (Puntos)	28
3.7.1.2.2. Destinos	28
3.7.1.2.3. Rutas	29
3.7.1.2.4. Alertas	29
3.7.1.3 Métodos de Lectura	29
3.7.1.3.1. Ruta (Puntos)	29
3.7.1.3.2. Destinos	29
3.7.1.3.3. Rutas	30
3.7.1.3.4. Alertas	30
3.7.2. GPX	30
3.7.2.1. Métodos	30
3.8. AndroidManifest	31
3.9. Build Gradle(Module: app)	31

## 2. Aplicación

### 2.1. Aspectos Técnicos

Desarrollada actualmente para Android a través del IDE oficial de Google para desarrollo de aplicaciones Android Studio y programada en lenguaje Java.

Versión mínima Android 4.3 (JellyBean) requerida. El 77% de los dispositivos Android actuales utiliza esta versión o una mayor. Puede correr en versiones previas como Android 4.0 (IceCreamSandwich), presente en el 97% de los dispositivos con Android, pero pierde las funcionalidades de conexión bluetooth con botonera.

### 2.2. Flujo de la aplicación

El flujo actual de la aplicación está basado en el concepto de “destinos”, estos destinos serán los lugares configurados que uno regularmente visita. Al seleccionar uno de estos destinos la aplicación y la información suministrada se ajusta a este destino. Esto hace más rápido el mostrar información relevante al usuario en base a lo que desea hacer y a la gran cantidad de información a filtrar.



En la versión actual de la aplicación no se aprecia muy bien la potencialidad de esta modalidad, pero a futuro en función del destino seleccionado se pueden sugerir rutas automáticas al destino desde el lugar en que uno esté, pudiendo ordenar estas rutas en base a criterios de rapidez, distancia, o incluso la ruta más agradable, la ruta más óptima en función de la experiencia, la ruta más segura, etc.

La libertad de poder hacer una ruta que no tenga un destino predeterminado también es importante por lo que una futura opción de “ruta libre”, que permita guardar una ruta sin destino predeterminado será útil.

La vista más importante dentro de todas, es la del mapa, aquí se despliega la información acerca de qué está ocurriendo en la ciudad y como se percibe esta. Como dar a entender esto no es trivial y si bien una solución puede ser poder mostrar la mayor cantidad de información posible, pudiendo quizás filtrarla de ciertas maneras en función de lo que quiera el usuario, esto no se recomienda para esta plataforma dado que finalmente esto terminará por confundir al ciclista, pensando además en el reducido tamaño de la pantalla.

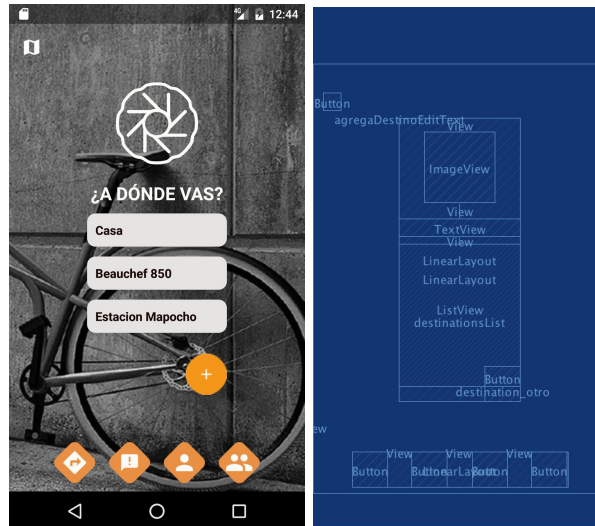
Debe existir un equilibrio entonces entre la simplicidad y minimalidad de la información que se muestra de manera que sea fácil de entender, y que no sea tan poca, o sea suficiente de manera que sea útil.

Pueden existir varias maneras de hacer esto, un mapa de calor, o filtros predeterminados, pero el enfoque de definición de destinos ayuda al usuario a acceder información útil para él en ese momento de manera más específica.

## 3. Documentación

### 3.1. Actividades

#### 3.1.1. LeBikeActivity



##### 3.1.1.1. ¿Qué hace?

- Primera actividad de entrada a la aplicación.
- Presenta una lista con los destinos (configurados por el usuario) para indicar dónde uno se dirige. Además un botón para agregar destinos directamente.
- Posee barra inferior de navegación para acceder a 'Mis Destinos' y 'Mis Alertas'.
- Permite ingresar de manera directa al mapa con el botón superior izquierdo.

##### 3.1.1.2. Trabaja con

- a. DondeVasAdapter
- b. DatabaseHandler

##### 3.1.1.3. Ciclo de Vida

- i. onCreate
  1. Se configura 'activity\_lebike.xml' como contenido de la actividad.
  2. Carga lista de destinos (List<Destino>) desde la base de datos (si es necesario).

- i. En esta versión genera 3 destinos de prueba: a. Casa, b. Beauchef 850 y c. Estación Mapocho
- ii. onStart
  - 1. Si no hay destinos en la base de datos despliega en lugar de la ListView un mensaje (TextView) para agregar uno.
  - 2. Configura el Adapter (*DondeVasAdapter*) de la ListView de los destinos, con la lista de destinos (*listaDestinos*).
  - 3. Configura el onItemClickListener de la lista de destinos. onItemClick:
    - i. Recupera objeto *Destino* correspondiente al item.
    - ii. Genera un Intent con
      - 1. *EnRutaActivity* como objetivo.
      - 2. El nombre del destino como extra.
    - iii. Se inicia una nueva actividad a través del Intent configurado.

#### 3.1.1.4. Métodos

- i. addDestination (onClick)
  - a. Método enlazado a botón para agregar destino directamente.
  - 1. Genera Intent con
    - i. *MisDestinosActivity* como objetivo.
    - ii. Extra que indica que se desea agregar nuevo destino.
  - 2. Se inicia la nueva actividad definida por el Intent.
- ii. misDestinos (onClick)
  - a. Método enlazado a botón de barra inferior de navegación.
  - 1. Inicia actividad *MisDestinosActivity*.
- iii. misAlertas (onClick)
  - a. Método enlazado a barra inferior de navegación.
  - 1. Inicia actividad *MisAlertasActivity*.
- ii. toMapActivity (onClick)
  - a. Método enlazado a botón que abre mapa directamente.
  - 1. Inicia actividad *EnRutaActivity*.

#### 3.1.1.5. To Do's

- 1. Pulir algunos elementos gráficos con mala resolución.
  - a. Botón de agregar nueva ruta.
- 2. Definir algunas keys como variables finales.



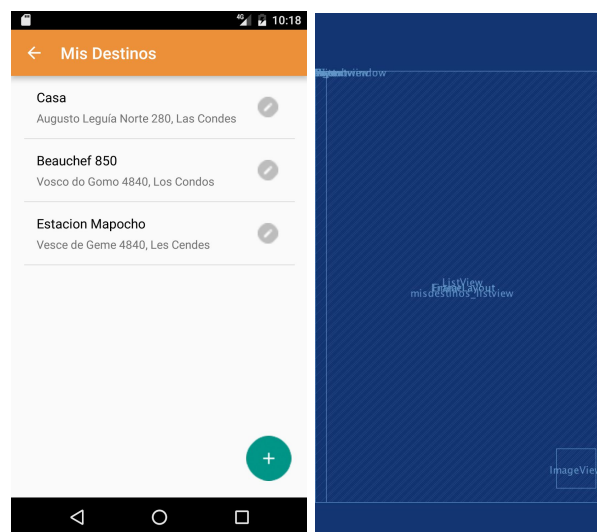
3. Agregar ruta libre, que permita hacer ruta sin un destino asociado.

Esto pensando en mantener el flujo actual de la aplicación. Se cree que este debe ser modificado.

### 3.1.1.6. Observaciones

LeBikeActivity utiliza un tema diferente al de las demás actividades. Esto con un fin estético, eliminando la ActionBar superior para una vista más amplia.

### 3.1.2. MisDestinosActivity



#### 3.1.2.1. ¿Qué hace?

- Permite administrar destinos.
- Permite agregar nuevos destinos.

#### 3.1.2.2. Trabaja con

- a. MisDestinosAdapter
- b. DatabaseHandler
- c. DestinoEditDialog

#### 3.1.2.3. Ciclo de Vida

- i. onCreate
  1. Configura 'activity\_misdestinos.xml' como contenido de la actividad.

2. Se recupera Intent y si este indica que se debe agregar un nuevo destino ejecuta *nuevoDestino(null)*.
- ii. onStart
    1. Carga lista de destinos (List<Destino>) desde la base de datos en caso de no haberlo hecho.
    2. Si la base de datos no posee destinos despliega mensaje (TextView) indicandolo.
    3. Configura Adapter (*MisDestinosAdapter*) de la ListView de los destinos.
    4. Configura onItemClick de los elementos de la lista:
      - i. Recupera destino (*Destino*) del elemento cliqueado.
      - ii. Guarda ID del destino en una variable global *onEditDestinoID*
      - iii. Crea un Bundle que contiene todos los campos del destino (nombre, direccion, id, lat y lon)
      - iv. Ejecuta *editDestino(bundle)*
    5. En esta versión se agregan rutas de prueba en la base de datos: fablab\_casa (x2), fablab\_beauchef850 (x2) y fablab\_estacionmapocho(x2).

### 3.1.2.4. Métodos Importantes

- i. nuevoDestino [onClick]
  - a. Método enlazado a botón para agregar destino.
    1. Ejecuta *editDestino(null)*.
- ii. editDestino(Bundle)
  - a. Se encarga de abrir DialogFragment para edición de destino.
    1. Adjunta bundle con información del destino (en caso de estar editando destino) al dialog fragment.
    2. Configura actividad como Listener del fragmento para que se puedan comunicar.
    3. Despliega el DialogFragment
- ii. onCloseClick (DialogFragment: onClick)
  - a. Enlazado a través de un DialogListener a botón para cerrar diálogo.
- ii. onEliminarClick(int id) (DialogFragment:onClick)
  - a. Enlazado a través de un DialogListener a botón para eliminar destino de id respectivo.
- iii. onGuardarClick(nombre, dirección, id, lat, lon)
  - a. Enlazado a través de DialogListener a botón para agregar destino.

- iv. `showToast(text)`
  - a. Muestra toast's ejecutados desde dialog fragment.

#### 3.1.2.5. To Do's

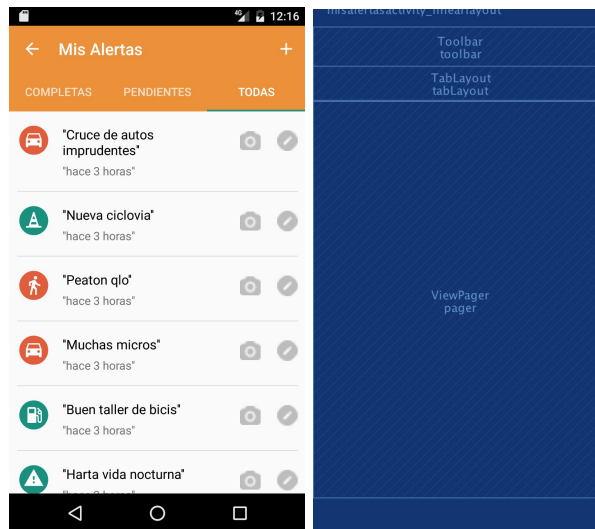
- Verificar que acciones de escritura y borrado de destinos a base de datos sean exitosas.
- Arreglar interfaz del DialogFragment. Ajustar tamaño de DialogFragment a tamaño predefinido, para diferentes tipos de pantalla se ve mal.
- Pulir elementos gráficos con mala resolución.
- Para versiones posteriores, en la ventana de edición del destino:
  - Verificar campos para que sean números o letras respectivamente.
  - Setear teclado especial para números.
  - Configurar mapa para editar localización del destino. O poder ingresar dirección y extraer localización desde ella.

#### 3.1.2.6. Observaciones

Se trabajó con DialogFragment para la ventana de edición del destino. Este es un Fragment que trabaja paralelo a la actividad con su propio ciclo de vida. El manejo del destino y su información entre ambas entidades se realizó de manera que toda la información de destino se entrega al fragmento (en caso de existir). El fragmento solo sirve de manera de ingresar y editar datos. Luego el fragmento entrega toda la nueva información del destino para que la actividad ejecute las acciones en base a esta información, no se guarda información del destino dentro de la actividad relacionado a su edición.

Para la comunicación hacia el fragmento, este recibe un bundle con datos del destino en caso de que se esté editando un destino. Para la comunicación desde el fragmento, se utilizó una Interface.

#### 3.1.3. MisAlertasActivity



### 3.1.3.1. ¿Qué hace?

- Despliega alertas con información acerca de estas.
- Separa alertas según su estado: completas o pendientes.
- Permite editar o agregar alertas.
- Trabaja con 3 ListFragments además de un DialogFragment y maneja el flujo entre los 4 y sí misma.

### 3.1.3.2. Trabaja con

- DatabaseHandler
- AlertaEditDialog
- MisAlertasInterfaces
- MisAlertasPagerAdapter

### 3.1.3.3. Ciclo de Vida

#### i. onCreate

1. Se configura '*activity\_misalertas.xml*' como el contenido de la actividad.
2. Se configura la ActionBar como Toolbar (android.support.v7.widget.Toolbar)\*
3. Se inicializa base de datos.
4. Se existe un Intent que indique la acción de nueva alerta se abre el diálogo de edición de alertas `openAlertasEditDialog(null, lat, lon, id_ruta)`.
5. Se configura el TabLayout, pestañas que permiten elegir entre alertas completas, pendientes o todas.

6. Se configura el ViewPager, páginas que permiten visualizar las listas de alertas completas, pendiente o todas.
- ii. onStart
  1. Se cargan las alertas a una lista (List<Alerta>) desde la base de datos.
    - i. En esta versión se genera un grupo de alertas de prueba.
  2. Se configura el adapter (*MisAlertasPagerAdapter*) del ViewPager.

#### 3.1.3.4. Métodos Importantes

- i. onCreateOptionsMenu
  1. Configura '*menu\_misalertas.xml*' como menú. Agrega sus elementos (solo 'nueva alerta') en esquina superior izquierda.
- ii. onOptionsItemSelected
  1. Reacciona a click sobre elemento del menú.
  2. Si es nueva alerta abre la actividad *EnRutaActivity* con un intent indicando que se está agregando una nueva alerta.
- iii. openAlertasEditDialog(alerta, lat, lon, id\_ruta)
  1. Abre diálogo de edición de alertas entregando un bundle.
  2. Por default, la alerta puede o ser nueva o antigua. Se reacciona entonces solo a estos casos.
  3. Se define que si argumento alerta es *null* se está editando una nueva alerta. Siempre que se defina una nueva alerta se debe proveer latitud y longitud. El *id\_ruta* puede no quedar definido, el valor -1 indica que no está definido (equivalente a null).
  4. En caso de proveer un argumento alerta definido, no se toman en cuenta los demás argumentos.
- iv. onAlertasListClick(alerta) (*MisAlertas*[Completas|Pendiente|Todas]Tab: onClick)
  1. Función llamada por medio de *MisAlertasTabListener* Interface desde alguna de las listas de alertas al seleccionar un item de la lista, una alerta.
  2. Abre un diálogo de edición de la alerta correspondiente.
- v. onCloseClick (AlertaEditDialog: onClick)
  1. Llamada para cerrar diálogo de edición de alerta.
- vi. onMostrarMapa (AlertaEditDialog: onClick)
  1. Llamada cuando se presiona opción de 'ver en el mapa' en diálogo de edición de alertas.
  2. Abre *EnRutaActivity* con acción que indica que se apunte a la respectiva alerta.

- vii. `onAgregarAlerta(alerta, action)` (`AlertaEditDialog: onClick`)
  - 1. Toma objeto *alerta* entregado y lo actualiza( o crea) en la base de datos en función de lo que indique *action*.
- viii. `onEliminarAlerta(alerta_id)` (`AlertaEditDialog: onClick`)
  - 1. Elimina alerta correspondiente a *alerta\_id* de la base de datos.

### 3.1.3.5. To Do's

- Terminar de implementar `onMostrarMapa()`, en esta actividad debe agregar al Intent información acerca de que alerta. Lo demás debe ser implementado en `EnRutaActivity`.
- Lo siguiente funciona, pero puede ser evaluado en pos de un mejor desempeño:
  - Ahora cada `ListFragment` hace acceso a la base de datos y carga las alerta. Evaluar si es posible, y mejor, hacer que solo `MisAlertasActivity` cargue alertas de la base de datos y las entregue a cada fragmento.
  - En `onAgregarAlerta()` se debe evaluar si crear o actualizar alerta en función del id de la alerta recibida, como en los destinos, o a través de la *action*.

### 3.1.3.6. Observaciones

`MisAlertasActivity` utiliza otro tema, diferente al de las demás actividades, este no posee `action bar` y permite utilizar `tabs` (pestañas).

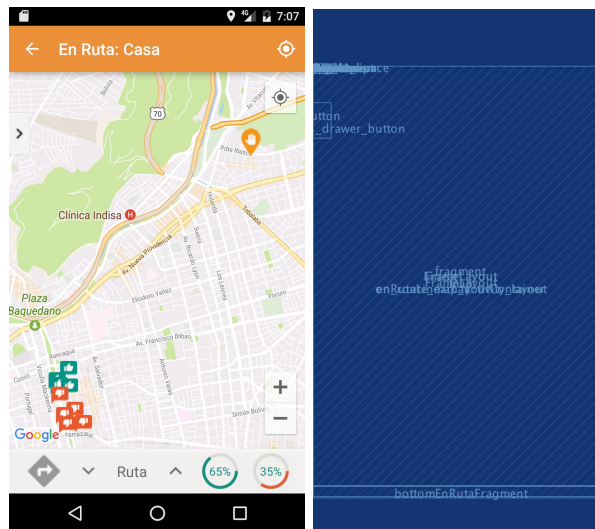
`MisAlertasActivity` debe manejar los 3 `ListFragment` de alertas y un `DialogFragment`.

Los tres `ListFragment` son manejados por `ViewPager` de similar manera que un `ListView` con sus items, a través de un `Adapter` en este caso llamado *MisAlertasPagerAdapter*. En este se definen y cargan los tres `ListFragment`.

Se utiliza la misma interface para manejar el click de cualquiera de las alertas de las tres listas.

El manejo de la información de una alerta desde la actividad al diálogo de edición, se da a través de un `bundle` hacia el dialogo. El diálogo evalúa la información contenida en este y ajusta su interfaz acordeamente. Al guardar la alerta se arma un objeto `Alerta` con información extraída de del estado de los elementos gráficos y se envía a la actividad a través de un interfaz.

### 3.1.4. EnRutaActivity



#### 3.1.4.1. ¿Qué hace?

- Actividad más importante de la aplicación.
- Despliega el mapa con información de rutas y alertas al usuario.
- Permite iniciar el tracking de una ruta.
- Permite conectarse por bluetooth a cliquer.
- Permite filtrar alertas en base a si clase.
- Permite agregar alertas en el mapa.

#### 3.1.4.2. Trabaja con

- EnRutaTrackingService
- DatabaseHandler
- EnRutaAuxiliaryBottomBar
- GPX

#### 3.1.4.3. Ciclo de Vida

- onCreate
  1. Configura 'activity\_enruta.xml' como contenido de la actividad.
  2. Inicializa base de datos.
  3. Lee Intent y carga
    - i. Id\_destino
    - ii. Intent\_action.
  4. Inicializa GoogleApiClient.

5. Inicializa barra auxiliar inferior.
- ii. onStart
    1. Evalúa si existe un destino a desplegar o no.
      - a. Si. Cambia título de la actividad y despliega barra inferior.
      - b. No. Configura título a 'Mapa' y vista queda abierta sin un destino específico.
    2. Conecta GoogleApiClient.
    3. Detecta si dispositivo es compatible con Bluetooth y despliega mensaje de resultado.
  - ii. onStop
    1. Desconecta GoogleApiClient
  - iii. onResume
    1. Configura Adapter y ListView de dispositivos Bluetooth en ventana de conexión con clicker.
    2. Configura onItemClick de dispositivo bluetooth:
      - i. Obtiene nombre del dispositivo bluetooth.
      - ii. Cancela scan de nuevos dispositivos.
      - iii. Ejecuta beanElementClicked(bean), lo conecta.
  - iv. onConnected (GoogleApiClient Connection Callback)
    1. Carga alertas de la base de datos en arreglo *alertas*.
    2. Carga rutas hacia destino especificado (en caso de haber un destino y no ser mapa libre) en *rutasADestino*.
    3. Detecta si es posible obtener la localización del dispositivo y la guarda.
    4. Ejecuta pedido para desplegar el mapa.
  - v. onRequestPermissionsResult (Location Request Permission Response Callback)
    1. Nada por ahora
  - vi. onMapReady (Map Request Ready Callback)
    1. Configure Map options.
    2. Realiza zoom en función de (1) posición actual usuario y (2) posición destino disponible:
      - a. (1) si y (2) si : enfoca ambas posiciones.
      - b. (1) si y (2) no : enfoca solo al usuario
      - c. (1) no y (2) si : enfoca solo al destino
      - d. (1) no y (2) no: enfoca localización por defecto.



3. Verifica si la actividad se está recargando para no desplegar elementos gráficos de nuevo. Para esto evalúa la lista de markers de alertas para ver si ya posee elementos cargados, si si, termina el método.
4. Posiciona un marker en el destino.
5. Muestra un `SnackBar` con instrucciones en caso de que `intent_action` del principio indique nueva alerta.
6. Posiciona markers en las alertas contenidas en array *alertas*.
  - a. Guarda los marker en *mapAlertas*.
  - b. Guarda el par {marker, alerta} en el `HashMap` *angelaMerkel*.
7. Dibuja rutas contenidas en *rutasADestino*.
8. Configura clicks sobre el mapa
  - a. `onMapLongClickListener`
    1. Dibuja marker de nueva alerta (borra el anterior si es que existía) y lo guarda en *addAlertaMarker*, variable `Marker` única.
    2. Muestra instrucciones en `SnackBar`.
  - b. `onMarkerClickListener`
    1. Si es *addAlertaMarker* obtiene posición de este e inicia *MisAlertasActivity* con la información de la nueva alerta.
  - c. `onMapClickListener`
    1. Borra *addAlertaMarker*.

#### 3.1.4.4. Métodos Importantes

- i. `startTrack(trackingState)` (*EnRutaAuxiliaryBottomBar*: `onClick`)
  1. En función de si `tracking state` es *true*:
    - a. No, si la ubicación está disponible:
      - i. Si, comienza el `tracking service` entregando el id de la ruta.
      - ii. No, no comienza.
    - b. Si,
      - i. Detiene el servicio
      - ii. Obtiene la lista de puntos que componen la ruta.
      - iii. Si es una ruta real (más de 2 puntos), guarda un archivo de la ruta formato `id_ruta.gpx` con nombre del id de la ruta. Si no, elimina la ruta inicializada de la base de datos.
      - iv. Borra puntos de la ruta de la base de datos.
  2. Informa a la barra auxiliar inferior de lo ocurrido para que actualice su estado.

- ii. `isLocationAvailableAndStoreIt`
  - 1. Checkea si existen permisos para leer ubicación del dispositivo.
  - 2. Despliega mensaje en función del tipo de ubicación disponible,
  - 3. Guarda ubicación en caso de estar disponible.
- iii. `latLng2CorrectBounds(latLng1, latLng2)`
  - 1. Devuelve esquinas sur-oeste y nor-este que definen los puntos ingresados.
  - 2. Funciona solo en hemisferio sur.
- iv. `loc2LL`
  - 1. Pasa de location a LatLng
- v. `bring/hideAlertasDrawer` (2 métodos)
  - 1. Muestra/Esconde barra lateral de filtrado de alertas.
- vi. `cancelScann`
  - 1. Detiene el scanner de dispositivos bluetooth
- vii. `beanElementClicked(bean)`
  - 1. Conecta el bean (clicker) al celular.
  - 2. Configura Callbacks de la conexión bluetooth
    - a. `onConnected`
      - i. Actualizo interfaz acordeamente.
    - b. `onDisconnected`
      - i. Actualizo interfaz acordeamente.
    - c. `onSerialMessageReceived(byte[] data)`
      - i. Recupera si fue un click positivo o negativo y lo despliega en un `SnackBar`.
      - ii. Se obtiene última localización del dispositivo.
      - iii. Se guarda una nueva alerta.
      - iv. Dibuja el marker correspondiente.
- viii. `beanOptionsClick`
  - 1. Si es que el bluetooth está habilitado
    - a. Se comienza/reinicia scann de dispositivos bluetooth.
    - b. Se configuran Callbacks
      - i. `onBeanDiscovered`
      - ii. `onDiscoveryComplete`

### 3.1.4.5. To Do's

En la actividad se encuentran con el mismo índice que aquí para poder identificarlos mejor.


1. Al cargar la actividad, en `onStart()`, si se está grabando un track el mapa siempre debe mostrar ese destino. Puede extrapolarse también a desplegar la información de que se está grabando un track en las demás actividades.
2. Ver qué hacer, en `onStart()`, si el destino especificado no está definido en la base de datos. Esto puede ser un error en la aplicación y se debe actuar.
3. Al iniciar el servicio de tracking, implementar mensaje desde el servicio confirmando el correcto funcionamiento.
4. Ir chequeando a medida que se capturan ubicaciones del tracking si el acceso a la localización es interrumpido.
5. Manejar bien la solicitud de permiso de ubicación.
6. Desplegar menú de acceso para activar localización.
7. Agregar este marker a array list `mapAlertas`.
8. Mostrar notificación para acceder a configuración de bluetooth.
  - Separar manejo del bluetooth en un fragment diferente. Actualmente la conexión al dispositivo bluetooth se maneja desde dentro de la actividad, y el layout es parte del layout de la actividad solo que es encuentra oculto. Llevar esto a un fragment externo.
  - Llevar el manejo del tracking a fragmento de la barra inferior.
  - Llevar el filtro de alertas a un fragment separado de la actividad.
  - Implementar creación y manejo de elementos gráficos:
    - Reimplementar ventanas de información de alertas. Existe una implementación antigua en la carpeta `Deprecated`.
    - Mejorar el manejo en el click de alertas.
    - Implementar el filtro de alertas según su tipo. Esto no es trivial e idealmente debe hacerse de manera eficiente.
    - Implementar agrupación de alertas al alejarse del mapa.
  - Llevar comunicación bluetooth a un service. Actualmente la comunicación bluetooth solo sirve al estar la aplicación activa. Si se desea poder presionar el clicker andando en bicicleta, el bluetooth de la aplicación debe estar funcionando en el background.

#### 3.1.4.6. Observaciones

- Para conexión bluetooth con LightBlue Bean se utiliza librería bluetooth proporcionada por los fabricantes de este dispositivo (`PunchThrough`).

## 3.2. Fragments

### 3.2.1. DestinoEditDialog

A dialog box titled 'DestinoEditDialog' with a close button (X) in the top right corner. It contains three text input fields: the first is labeled 'Casa', the second is labeled 'Augusto Leguía Norte 280, Las Condes', and the third is labeled with coordinates '-33.4129' and '-70.597636'. At the bottom, there are two buttons: 'GUARDAR' and 'ELIMINAR'.

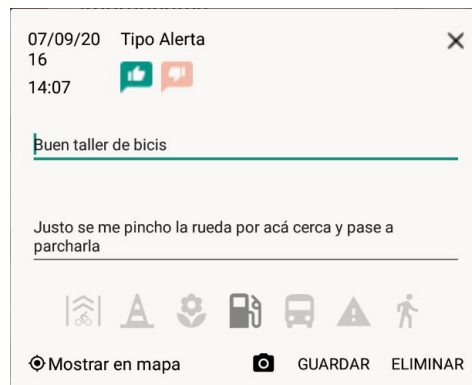
#### 3.2.1.1. Funciones

- Desplegar información de un destino existente (en caso de que uno haya sido provisto al momento de su creación)
- Permite editar información acerca de un destino nuevo o existente.
- Reaccionar a eventos de cerrar, eliminar o guardar. Enviar callbacks a actividad padre a través de interfaces, que además lleven la información necesaria para ejecutar la acción correspondiente.

#### 3.2.1.2 Detalles

- Corresponde a un DialogFragment
- Utiliza layout 'fragmentdialog\_edit\_destino.xml'

### 3.2.2. AlertaEditDialog

A dialog box titled 'AlertaEditDialog' with a close button (X) in the top right corner. It displays a date '07/09/20' and a time '14:07'. Below this, there are two icons: a green thumbs up and a red thumbs down. The main text area contains the title 'Buen taller de bicis' and the description 'Justo se me pincho la rueda por acá cerca y pase a parcharla'. At the bottom, there is a row of icons representing different activities: a house, a person, a gear, a fuel pump, a bus, a warning triangle, and a person walking. Below these icons, there are three buttons: 'Mostrar en mapa', 'GUARDAR', and 'ELIMINAR'.

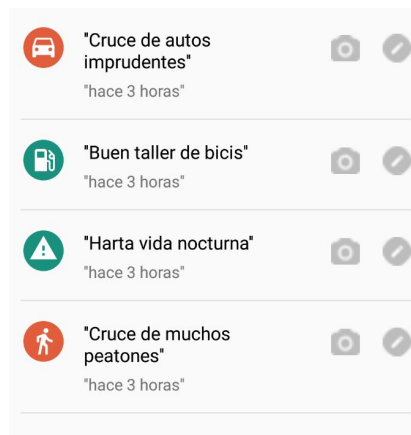
### 3.2.2.1. Funciones

- Desplegar información de una alerta ya existente si es el caso, o en edición. Si existe información de una alerta que desplegar, toda la información que define esta alerta será provista por la actividad padre al momento de la creación del diálogo.
- Permite modificar una alerta (nueva o existente) a través de la edición de sus campos.
- Ante llamados a la actividad principal, revisa si la información ingresada es suficiente y/ correcta, y la empaqueta para ser enviada a la actividad.
- Permite ejecutar ciertas acciones: cerrar ventana, guardar o eliminar alerta y mostrar alerta en el mapa, a través de callbacks a la actividad padre que realiza la acción en cuestión.

### 3.2.2.2. Detalles

- Este diálogo en realidad no sabe si la alerta es nueva o ya existente, ya que en realidad siempre se le entrega una alerta existente. Lo que cambia es solo la cantidad de información (campos definidos) que esta lleva.
- Corresponde a un DialogFragment.
- Utiliza layout 'fragmentdialog\_edit\_alerta.xml'

### 3.2.3. MisAlertasCompletasTab



### 3.2.3.1. Funciones

- Cargar alertas del estado correspondiente de la base de datos.
- Desplegar una lista de las alertas.
- Reaccionar ante clicks sobre la lista (items) y desplegar AlertaEditDialog.

- Administrar Callbacks generados por ventana de edición de alertas.

### 3.2.3.2. Detalles

- Corresponde a un ListFragment.
- Utiliza adaptador MisAlertasAdapter.
- Utiliza layout 'fragmenttab\_misalertas\_completas.xml'

### 3.2.4. MisAlertasPendientesTab

- Idem a 3.2.3.

### 3.2.5. MisAlertasTodasTab

- Idem a 3.2.3.

### 3.2.6. EnRutaAuxiliaryBottomBar



#### 3.2.6.1. Funciones

- Aparece en caso de que se haya ingresado a un destino. Para visión de mapa libre la barra permanece oculta.
- Permite seleccionar una ruta en específico de las desplegadas en el mapa.
- Despliega información de esta ruta, indicando porcentaje de votos positivos y negativos.
- Contiene botón de inicio de track.

#### 3.2.6.2. Detalles

- Corresponde a un Fragment
- Manejo del botón de tracking no es realizado aquí.
- Todos, manera bien selección de rutas y despliegue de información.
- Todos, realizar manejo del tracking en este fragmento.

## 3.3. Services

### 3.3.1. EnRutaTrackingService

#### 3.3.1.1. Funciones

- Trabaja como un proceso paralelo, en el background de la aplicación. Se mantiene activo aún con el celular bloqueado o la aplicación cerrada.
- Accede a la ubicación del usuario de manera periódica y la guarda.
- Al terminar el track la aplicación cierra el servicio y accede a todas los puntos guardados generando una ruta.

#### 3.3.1.2. Funcionamiento

- Al ser creado el servicio, guarda una nueva ruta en la base de datos, esta no contiene toda la información, pero al ser terminada la ruta se completan los campos faltantes, o se elimina la ruta si no es efectivo guardarla.
- Se crea un GoogleApiClient y se conecta.
- Una vez conectado, se inicia el servicio de LocationServices.FusedLocationApi.requestLocationUpdates.
- Se guardan los puntos generados por los updates de localización periodicos provistos por el callback onLocationChanged.

#### 3.3.1.3. To Do's

- Guardar la ruta generada con nombre correspondiente.
- Implementar un broadcast para recibirlo en EnRutaActivity e ir dibujando la ruta en tiempo real.
- Reaccionar a denegación de acceso al GPS en la ruta.

## 3.4. Adapters

### 3.4.1. DondeVasAdapter

- Asociado a lista de destinos en LeBikeActivity.
- ArrayAdapter.
- Maneja objetos de clase Destino.
- Despliega layout de un TextView simple definido en 'listitem\_dondevas.xml'.

- Items poseen forma rectangular redondeada dada por el drawable 'shape\_item\_dondevas.xml'.



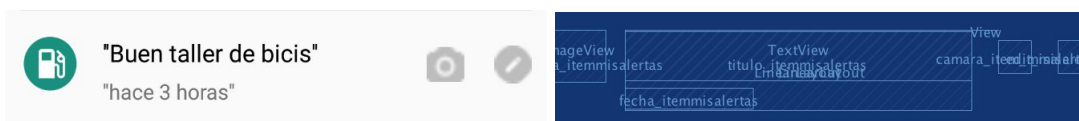
### 3.4.2. MisDestinosAdapter

- Asociado a lista de destinos en MisDestinosActivity.
- ArrayAdapter.
- Maneja objetos de clase Destino.
- Layout definido por 'listitem\_dondevas.xml'.



### 3.4.3. MisAlertasAdapter

- Asociado a lista de alertas en los ListFragment (MisAlertas<Completas /Pendientes /Todas>Tab]) de MisAlertasActivity.
- ArrayAdapter.
- Maneja objetos de clase Alerta.
- Layout definido por 'listitem\_misalertas.xml'.
- Debe en función del tipo de alerta, desplegar la imagen correcta.



#### 3.4.3.1. To Do's

- Mejorar la forma de asignación de imagen del tipo de alerta.

### 3.4.4. MisAlertasPagerAdapter

- Asociado a ViewPager de fragmentos en MisAlertasActivity.
- FragmentPagerAdapter.
- Crea y administra los tres fragmentos para desplegar alertas completas, pendientes o todas.



- Configura el Interface que maneja el click sobre los elementos de la lista de los tres fragmentos.

## 3.5. Clases

### 3.5.1. Destino

#### 3.5.1.1. Campos

Clase bastante simple que agrupa los elementos que definen un destino:

- Nombre
- Dirección
- Id
- Latitud
- Longitud.

Estos son definidos como campos privados de la clase

#### 3.5.1.2. Constructores

Posee constructores en base a definición directa de sus campos, o a través de un Cursor. Este último para facilidad de trabajo con la base de datos dado que es la estructura de datos que devuelve un query SQL.

#### 3.5.1.3. Métodos

Los métodos existentes sirven para leer los campos.

### 3.5.2. Alerta

#### 3.5.2.1. Campos

Campos definidos de manera privada y definen totalmente el concepto de alerta trabajado:

- Id
- posneg (positiva o negativa)
- Latitud
- Longitud
- Tipo de Alerta
  - Tipo de alerta entre las 7 opciones definidas. Se utilizan Strings de largo 4.
    - cicl(ovía), vias, vege(tación), mant(ención), auto(s), peat(ones) y otro(s)
- Hora
- Fecha

- Título
- Descripción
- Id Ruta
- Versión
- Estado

### 3.5.2.2. Constructores

Posee constructores en base a variables directas, Cursor, utilizado en trabajo con base de datos, y Bundle, usado en comunicación con fragmentos.

### 3.5.2.3. Métodos

Además de métodos para leer campos.

1. setEstado
  - a. Configura estado de la alerta.
2. isComplete
  - a. Indica si todos los campos se encuentran definidos. Esto la define como alerta pendiente o completa.
3. toBundle
  - a. Devuelve una estructura Bundle equivalente.
  - b. Define un conjunto de keys como variables utilizadas en el bundle.

## 3.5.3. Ruta

### 3.5.3.1. Campos

Campos definidos como privados de la instancia.

- ID
- ID Destino
- Name
- Num Positive
- Num Negative
- Hora
- Fecha
- Duración (en segundos)
- Distancia (en metros)

### 3.5.3.2. Constructores

Constructores en base a variables directas y a Cursor, este último para facilitar trabajo con base de datos.

### 3.5.3.3. Métodos

Además de los métodos para leer los campos, el otro método público es:

1. `getTrack(context, routeID)`
  - a. Devuelve una lista `List<LatLng>` con todos los puntos de una ruta.
  - b. Estos puntos se encuentran almacenados en el archivo *nombre\_ruta.gpx* creado al terminar una ruta.

## 3.6. Interfaces

### 3.6.1. MisAlertasInterfaces

Se define la interfaz `MisAlertasTabListener`. Definida de manera externa a los `ListFragments` de `MisAlertasActivity` ya que es compartida por los tres de ellos. Esto permite a `MisAlertasActivity` manejar un click en cualquiera de las tres listas a través de un solo `CallBack`.

Se aprovecha entonces de definir también la interfaz `AlertaDialogListener` para que todas las interfaces relacionadas a alertas queden aquí.

## 3.7. Base de Datos

### 3.7.1. DatabaseHandler

Clase encargada de crear y administrar la base de datos de la aplicación. Android provee herramientas para trabajar con `SQLite` que es la base de datos utilizada.

En completo la clase se divide en 3 partes:

1. Definición y creación de tablas SQL.
2. Métodos de Escritura en la base de datos
3. Métodos de Lectura en la base de datos.

### 3.7.1.1. Definición de Tablas

La información relevante a guardar en esta etapa son: (1) primero y más importante, las alertas, (2) las rutas realizadas y (3) los destinos configurados. No es de extrañar que sean las mismas definiciones para las cuales se definió una clase.

Notar que el guardar las rutas realizadas, en realidad consiste de dos partes. Primero la ruta en sí, es decir, el conjunto de puntos geográficos que define el camino seguido. Y segundo, la ruta como definición, es decir, el nombre de ruta, hacia qué destino se dirige, su identificador, la hora y fecha de realización, etc.

Para almacenar la ruta como camino en la memoria, no se utilizará una tabla SQL para cada ruta, sino que se guardarán como archivos *.gpx*, formato estándar a la hora de generar archivos con caminos geolocalizados. De todas maneras, si existirá una tabla para guardar la ruta como camino, pero esta será utilizada mientras se realiza un camino, se irá llenando a medida que se realiza una ruta, y terminada la información se guarda en un archivo gpx y la tabla se borra.


Son cuatro entonces el total de tablas a utilizar:

- Destinos
- Ruta (Puntos)
- Rutas
- Alertas

en la siguiente figura un detalle de su estructura.

## Base de Datos - App

Destinos	*Ruta	Rutas	Alertas
id	seq_num	id	id
nombre	tiempo	id_destino	pos_neg
direccion	lat	nombre	lat
lat	lon	num_pos	lon
lon		num_neg	hora
		hora	fecha
		fecha	tipo_alerta
		duracion	id_ruta
		distancia	autor
			titulo
			descripcion
			estado
			version



\* Esta tabla existe durante la grabación de un track, al terminar se guarda como archivo .gpx

### 3.7.1.2. Métodos de Escritura

#### 3.7.1.2.1. Ruta (Puntos)

- `addTrackingPoint(sequence number, tiempo, latitude, longitude)`
  - Añade un nuevo punto a la ruta.
- `eraseTrackPoints()`
  - Borra todos los elementos de la tabla

#### 3.7.1.2.2. Destinos

- `newDestiny(destino)`
  - Añade un destino a la tabla.
- `updateDestino(nombre, direccion, id, lat, lon)`

- a. Actualiza el destino especificado en el id con la información entregada.
- iii. deleteDestino(id)
  - a. Borra destino correspondiente a la id.

#### 3.7.1.2.3. Rutas

- i. startRoute(id, dest\_id, name, hora, fecha)
  - a. Añade nueva ruta (incompleta) con datos disponibles.
- ii. endRoute(id, num\_pos, num\_neg, duracion, distancia)
  - a. Actualiza ruta incompleta con información suministrada.
- iii. newRuta(ruta)
  - a. Añade nueva ruta (completa).
- iv. deleteRoute(id)
  - a. Borra ruta especificada por su Id.

#### 3.7.1.2.4. Alertas

- i. newAlerta(alerta)
  - i. Añade alerta a la tabla.
- ii. updateAlerta(alerta)
  - i. Actualiza alerta caracterizada por su id.
- iii. deleteAlerta(id)
  - i. Borra alerta.

### 3.7.1.3 Métodos de Lectura

#### 3.7.1.3.1. Ruta (Puntos)

- i. getRoutePoints
  - a. Devuelve una lista List<Location> con todos los puntos de la tabla.
- ii. getLastSeqNum
  - a. Devuelve el valor del mayor sequence number de algún punto en la tabla.

#### 3.7.1.3.2. Destinos

- i. getDestinationById(id)
  - a. Devuelve Destino correspondiente.

- ii. `getDestinations`
  - a. Devuelve una lista `List<Destino>` con todos los destinos configurados.
- iii. `getLastDestinationId`
  - a. Devuelve el mayor id de algún destino en la tabla.

#### 3.7.1.3.3. Rutas

- i. `getRutas`
  - a. Devuelve una lista `List<Ruta>` con todas las rutas realizadas.
- ii. `getRoutesByDestinationId(dest_id)`
  - a. Devuelve una lista `List<Ruta>` con todas las rutas hechas al destino definido por `dest_id`.
- iii. `getLastRutasId`
  - a. Devuelve el mayor id de alguna de las rutas en la tabla.

#### 3.7.1.3.4. Alertas

- i. `getAlertas`
  - a. Devuelve una lista `List<Alerta>` con todas las alertas disponibles.
- ii. `getAlertasByEstado(tipo_estado)`
  - a. Devuelve una lista `List<Alerta>` con todas las alertas disponibles que coincidan con el estado especificado.
- iii. `getLastAlertId`
  - a. Devuelve el mayor id de alguna de las alertas en la tabla.

## 3.8. Server

### 3.8.1. To Do's

Falta realizar toda la sincronización de la base de datos de la aplicación con el servidor. Esto abarca primero definir qué cosas se subirán en esta etapa: ¿Datos de destinos?, ¿Rutas?, ¿Alerta?, probablemente.

Luego definir en qué momento se realizarán las subidas de datos y descargas de información a y desde el servidor. ¿Se mantendrá la base de datos de la aplicación como para acceder a los datos sin conexión a internet? Esto puede parecer lógico pero no mantener una base de datos tan grande ayuda a hacer la aplicación más liviana. Quizás algunos datos pueden mantenerse y otros preguntarse en su medida.

## 3.9. Utils

### 3.9.1. GPX

Esta clase contiene las funcionalidades que ayudan a llevar una lista de puntos `List<Location>` a un archivo GPX.

#### 3.9.1.1. Métodos

1. `writePath(file, n, points)`
  - a. Escribe la lista de puntos *points* al archivo *gpx file*.

#### 3.9.1.2. To Do's

- Escribir un archivo más completo incluyendo información del tiempo de cada punto.

## 3.10. AndroidManifest

- Se solicitan permisos de
  - `ACCESS_FINE_LOCATION`
  - `ACCESS_COARSE_LOCATION`
  - `BLUETOOTH`
  - `BLUETOOTH_ADMIN`
  - `WRITE_EXTERNAL_STORAGE`
  - `READ_EXTERNAL_STORAGE`
- Se configura, como meta-data, la `api_key` para poder utilizar el mapa.
- Se declaran todas las actividades y servicios.

## 3.11. Build Gradle(Module: app)

- Versión mínima de sdk de android: 18.
- Se utilizan librerías:
  - `support:appcompat-v7:23.1.1`
  - `support:design:23.1.1`
  - `gms:play-services:9.4.0`
  - `Com.punchthrough.bean.sdk:sdk:2.0.1`



## 4. To Do's

Como resumen y recopilación de los aspectos pendientes más importantes de la aplicación:

1. Implementar enlace bluetooth al dispositivo en el background de la aplicación.
  - a. Agregar alertas a la base de datos.
  - b. Agregar punto de ruta a base de datos.
  - c. Mantener conexión en background.
2. Implementar correctamente administración gráfica y en estructura de datos de elementos del mapa:
  - a. Mostrar ventanas de información (snippets) de alertas.
  - b. Filtrar alertas con barra lateral izquierda.
  - c. Agrupar zonas de mucha densidad de alertas en función del zoom del mapa.
  - d. Etc.
3. Terminar manejo y selección de rutas por parte de barra inferior auxiliar en mapa.
4. Arreglar dimensiones de DialogFragments de edición de alertas y destinos.
5. Implementar sincronización con el servidor.
  - a. Definir qué datos, qué cantidad de esos datos.
  - b. Definir momentos de sincronización.
  - c. etc.

## 5. Referencias

[1] Android Developer: <https://developer.android.com/index.html>

[2] PunchThrough Android SDK: <https://github.com/PunchThrough/bean-sdk-android>

[3] LeBike App Repository: <https://github.com/stereo92/LeBikeApp>

[4] Versión On Line de este documento:

[https://docs.google.com/document/d/1tj5XWWkZl6yYla2KdJ\\_TAJxOzzAXLcROQgWsxjEi7dA/edit?usp=sharing](https://docs.google.com/document/d/1tj5XWWkZl6yYla2KdJ_TAJxOzzAXLcROQgWsxjEi7dA/edit?usp=sharing)