Developer’s guide

1. Build process

The project uses gradle as a buildtool (http://gradle.org/). For more information see the readme file found in the repository.

2. Dependencies

The application uses a variety of different libraries, each listed below.

2.1. Lombok

A library that helps you create simple methods such as getters, setters as well as equals and hashcode (https://projectlombok.org/).

2.2. Volley

A library created by Google that simplifies and speeds up sending requests to REST APIs (https://github.com/mcxiaoke/android-volley).

2.3. Android support library

The project uses two Android support libraries, as well as the design library.

- v.4 support library is used for its ViewPager class.
- v.13 support library is used for its FragmentPagerAdapter, which acts as a bridge between v.4 fragments and regular fragments.
- Android design support library is used for its floating action buttons.

2.4. PagerSlidingTabStrip

The project uses astuetz's PagerSlidingTabStrip (https://github.com/astuetz/PagerSlidingTabStrip) to create its tab layout. This will be exchanged for Android's version of tab layouts in the future.

2.5. Google Map

Google's library is used for drawing a map in your application (https://developers.google.com/maps/documentation/android-api/).

2.6. Google Places

Google's library is used for searching for places (https://developers.google.com/places/android-api/).

2.7. Pmd, Findbugs and Lint

Code analysis libraries that detects bugs and bad practises in your code. For more information see the readme file found in the repository.
(https://pmd.github.io/)
(http://findbugs.sourceforge.net/)
(http://developer.android.com/tools/help/lint.html)

2.8. Stan

STAN (http://stan4j.com/) provides code structure analysis. It determines dependencies in and between packages and serve to remove cyclic dependencies in the application.

3. Version Control

Git is used as version control tool. In this project its main purpose is to administrate when new functionality is delivered, as well as a back-up feature.

3.1. Workflow

The project's git workflow consists of two main branches: the Master branch and the Develop branch. On the Master branch lies the current version of the product, which is always ready for delivery. The Develop branch works as a staging area for all functionality which should be included in the next delivery. Every large feature is branched from develop and merged back when completed, leaving the functionality branch visible in the history, whilst smaller changes are rebased and represented as a linear history in the branch they were made.

4. Design decisions

4.1. Android API level

The project’s target API-level is 22 and its minimum API-level 16. The levels are chosen so that the application is usable on roughly 90% of Android devices.

4.2. Extendability

It is vital to be able to further add functionality to the application as the applications goal has not yet been reached and possible future changes. Thus the project strives for a extendable design through the dependency inversion principle.

4.3. Portability (decoupled model)

Many of the implementations in the application is coupled to the Android platform but communication between the different classes is usually abstracted through interfaces. As such, most of the structure may be preserved if the application were to be ported to another system.

5. Components

Discussed below are the major components of the project.

5.1. GUI

The application uses a standard Android user interface. It consists of one main activity and four fragment tabs with an extra fragment for creating new destinations and dialogs. The application uses a custom theme for its appearance and the Android design library for some of its graphical components.

5.2 Floating action buttons and points data

The floating action buttons located in the map screen currently uses static data for points. This is a temporary solution when they should in fact dynamically query information from sources which provide the data that is needed. The IMapData interface exists to hide the method with which the point information are acquired and facilitates the change from static data into dynamic.

5.3. Destinations

Destinations are immutable objects in order to make them easier to manage. The SavedDestinations class manage all the instances of the destinations. It provides a single point of entry observer pattern for any object interested in changes on the list of destinations saved by the user.

The application provides the ability to save destinations. This is done using an SQLite database, which is the standard solution for saving data on an android device, in the AndroidDestinationSaver class. In order to make this functionality portable to other platforms the functionality is implemented behind the IDestinationSaver interface in the SavedDestinations class. Thus an alternative solution for other platforms can be easily implemented by sending an alternative destination saver when creating the SavedDestinations object. The SavedDestinations object will automatically save all changes in itself async to the database.

5.4. Guide

The guide functionality is a complex feature that consist of several components. The GuideHandler class manage all these components and provides a single point of entry for creating, starting and stopping a guide and all it's components.

The first step is the OnWhichBusIdentifier class, which will regularly scan the WiFi to identify the ElectriCity wifi in order to identify which ElectriCity bus the user is currently on. This class is hard coded for ElectriCity, but implements the IOnWhichBusIdentifier interface. In order to expand the application to work on other public transport networks an alternative identifier may be created and will work in the GuideHandler if it implements the IOnWhichBusIdentifier interface. Although it can be reused for other platforms as it does not contain any android specific functionality.

When a bus is identified, a Route is created with the bus as an argument. The route will regularly ask the IGetNextStopAPI interface for the next stop for the bus. This information is then used to identify and store Points of Interest in the route for future reference. The class is made portable through the use of interfaces and pure java code.

The route is provided when creating the Guide object, which ask the route which is the next point of interest each time the guide wishes to play a new part of the guided tour. In order to make the guide itself portable and modular, an abstract super class called AbstractGuide has been created. For the android specific solution an AndroidGuide has been created which handle both the voice and text guide when provided with a route.

5.5. MapScreen

The fragment containing the Google map uses the IMap interface to decouple itself from those who uses it.

5.6. API's

All queries to external APIs and similar functionality (AndroidDevice) is hidden behind the ApiFactory class as well as the IApiFactory interface. This Factory is reached globally as a Singleton and provides different parts of the application with access to the different API's used in the project.

5.6.1. Västtrafik API

The application uses Västtrafik's official API to determine a route to a destination for the user. The API provides the ability to customise request. The customisability isn't used, though, since the application only presents the first result to the user. As such this API could be changed to for example Google's Destinations API as it might provide easier and faster interaction.

5.6.2. ElectriCity API

The Electricity API is used for fetching the next stop for a specified bus. The bus is provided by the OnWhichBusIdentifier and the stop is used in route. The ElectriCity API currently provide more functionality than is in use in the application as a foundation for future development.

5.6.3 Onboard bus API

Each bus provides a local onboard WiFi which contain an API. This API has been used in the ElectriCityWiFiSystemIDAPI in order or get the system id of the local router. The ElectriCity assets has provided a list of all the ElectriCity busses, which contain this information. The OnWhichBussIdentifier compare the system ID from this API and compare it with the provided list to identify the bus.

5.6.4. Google's API

The project takes advantage of the Google Locations API in order to gain better geopositioning. It uses the Google Places API to be able to turn a search for a place into a latitude and longitude position. Finally it uses the Google Maps API to present to the user a high quality map with rich features. All use of these APIs are as carried out as recommended by Google on their guide pages. While the Google Places and Google Maps API can be found directly in the code where they are used the Google Locations API can be accessed through the common API façade.

6. Further development

A link to the backlog is provided in the README. Adding further API calls or expanding the queries to an existing API should follow these guidelines.

6.1. Adding further APIs

To add new APIs for the application to call you need to define the the queries in a interface. Let the ApiFactory instanciate your querier and provide it through IApiFactory.If you need to send a request to a REST API you can let ApiFactory give you access to the Request queue used by other queriers.

6.2. Adding queries to existing APIs

To add queries you either create a new interface for the query and let ApiFactory provide it to the application or add it to an existing interface, which the ApiFactory already provide. The query should follow the same callback structure used by existing queries.