DEVELOPER DOCUMENTATION

BUILD PROCESS

The app uses the build automation system Gradle. For the app to work all API keys must first be manually added before building for the app to work. API keys must be added to the APIConstants Java class and and the res/values/api_keys.xml file. This is intentionally left out to make sure that the source code can be open and publically available.

MAJOR PARTS

There are three major components in the application which all come together to fill the purpose of the app. These major components are score collection, score comparison and the bus map.

Score collection

The score collection is the most important part of the app. Its purpose is to determine when the user should get points, and how many points the user should get.

The score collection is done in a couple of steps:

1. The wireless network adapter on the device is scanning for WiFi. This scanning is OS-dependent and varies from device to device when it comes to how often it performs the scanning. If a desired MAC address (the MAC address of an ElectriCity connected bus) is found nearby, it triggers the app's WiFi background service. This is currently the only way the WiFi service is triggered but it can be launched from anywhere, for example via a button or a "swipe to refresh" gesture.
2. If a MAC-address is found nearby the user for a certain amount of time, the score collection will begin.
3. The WiFi background service is constantly checking the MAC address in and when it can't find the bus MAC address for a certain amount of time the point collection stops.
4. The two timestamps of when the user went on and then off the bus is used to check how far the bus went during that time. This becomes the user's added score. While the user is currently on the bus, the current timestamp is temporarily used as the end time to live update the score.

Score comparison

To enable the users to compare scores with each other the Facebook Scores API is used. Each user continuously uploads its score to Facebook every 15 minutes. This of course requires the user to connect to Facebook which is possible within the app's profile or highscore screen.

The user can then view its Facebook friends' scores and compare them with its own in the highscore fragment. The highscore is shown as a sorted list where the Facebook friend with the highest score is at the top and every entry in the list shows the Facebook friend's name, picture, place and score.

Bus map

Another big part of the application is the map that displays all of the busses and bus stations. The Google Maps API is used to show everything on a map. Pins are used to represent bus stations while arrows represents busses. The position and data of bus stations are hardcoded in the Constants class while the busses' information are fetched continuously, using the BusPositionUpdater task, from the Electricity API. When the user presses on a bus (a moving arrow) another screen is opened to show more detailed information for that bus.

DESIGN SOLUTIONS

Android API Level

The app's minimum supported API level is 15 since that covers about 95% of the Android devices currently in use. It also has all necessary functions for the current version of Bus Battle. However, since we were aiming for a modern design using the material design guidelines which support was added for in API level 21 some Android support libraries are used for backwards compatibility.

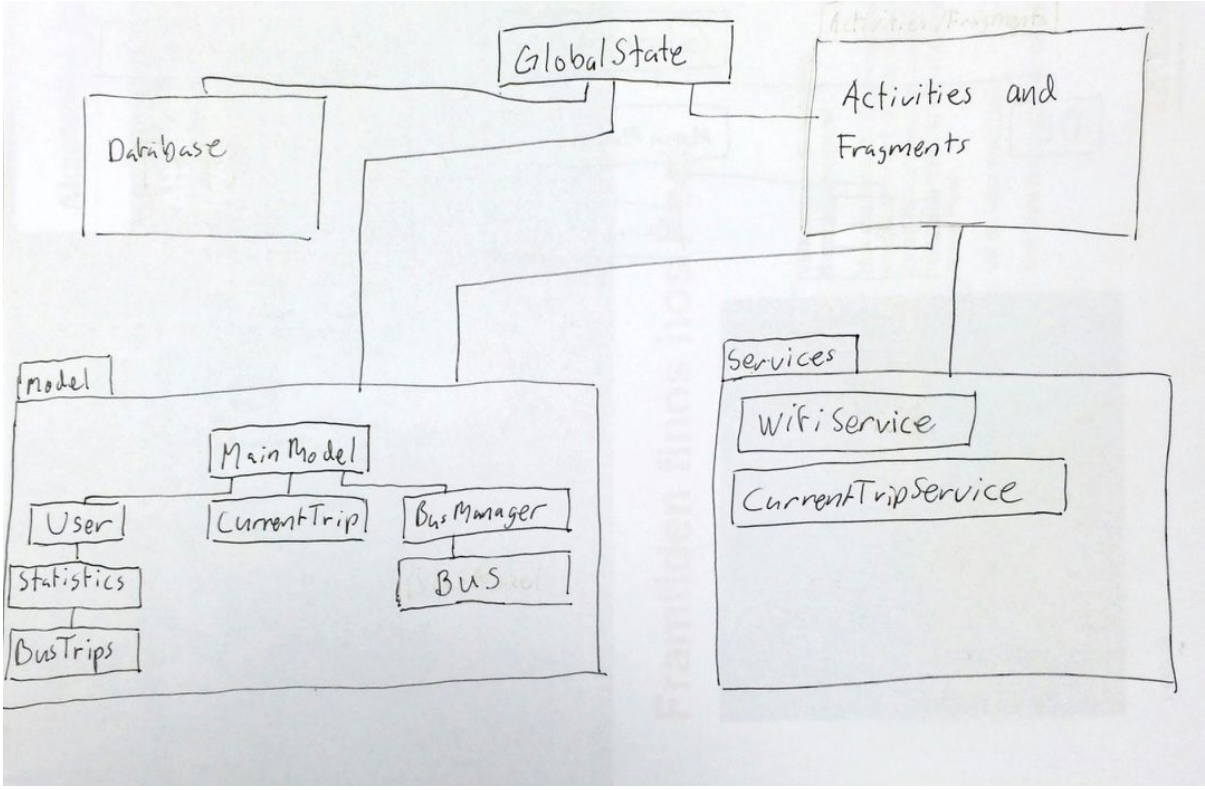
Model View Controller

The MVC is implemented by letting Android Activities and Fragments represent controllers and by seeing the view part as those view objects that the system *inflates* from the layout XML files. The model is implemented in a separate package and is accessed by extending the Android *Application* class and using it as a main controller.

Bus data

The data for identifying the busses (their VIN number and corresponding MAC address) is currently hard coded into the app. This is because there is no way of getting the busses' MAC addresses from the ElectriCity API. In a further development this data needs to be moved to an API accessible over the internet so that it can be updated remotely without rebuilding the application.

DOMAIN MODEL



GlobalState extends Android's Application class and acts as the "main" or "top level" controller in the application. Since GlobalState extends the Application class it is accessible in all Activities and Fragments, we therefore let GlobalState own the model so that the model then can be accessed where needed. Since GlobalState owns the model we also let GlobalState be responsible for loading and saving it from and to persistent data - in this app's case a local SQLLite database.

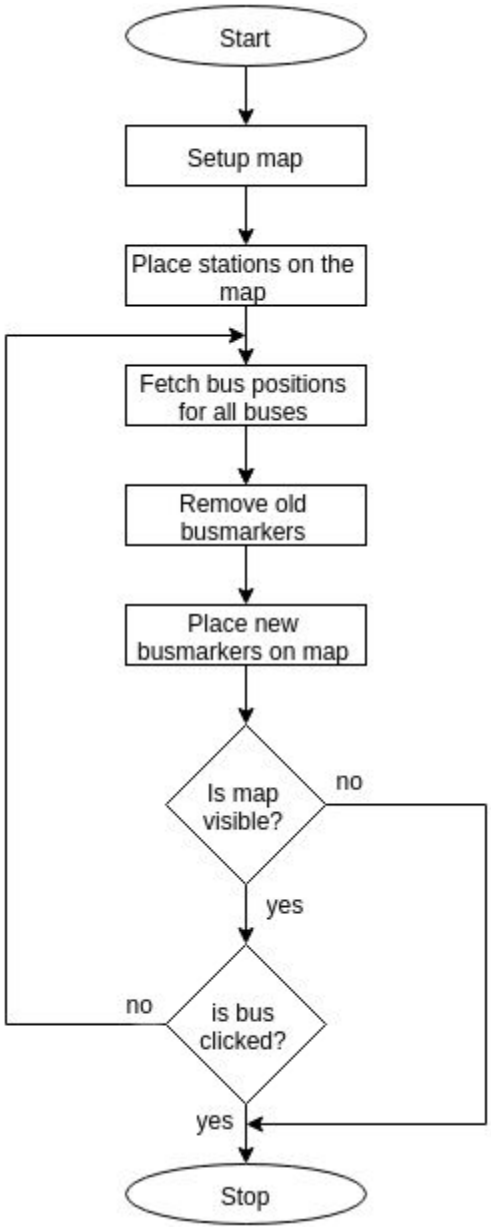
The views are updated using listeners that listens to changes in the model. Everytime the model is updated all listeners are notified.

We also have some services that perform tasks in the background. The WiFi service is responsible for determining if the user is on a bus or not and the CurrentTripService is then used to continuously update points based on the distance travelled while the user in on the bus.

FLOWCHARTS

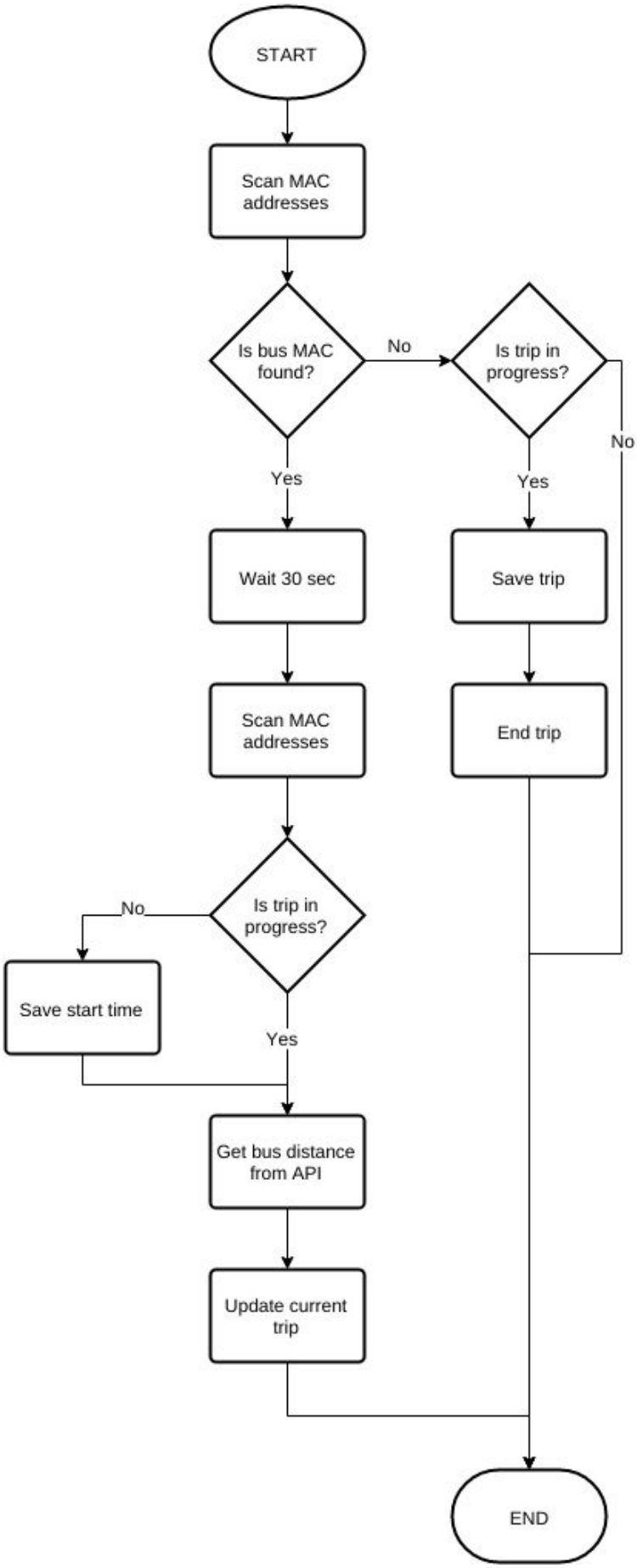
Live updated bus map

The flowchart to the right represents the bus map screen in the application. It shows the whole life cycle of the screen from its start until the user navigates back or clicks on a bus icon to get more information.



Updating user score

The flowchart to the right represents the process of determining if the user is on a bus and if it is; updates the user's score. This flow will be ran continuously, and does not handle it's own looping.



EXTERNAL DEPENDENCIES

Google maps API

The Google Maps API is part of the Google Play Services API for Android and is used in the application for showing the map of bus stations and buses.

ElectriCity API

An API for getting information from all ElectriCity connected buses. This is the core API for the app and is used to get all resources from the buses.

Facebook API

The Facebook API is used to get information about the user, such as its profile picture and name but also the user's friends. In particular we use a part of the Facebook API called the Facebook Scores API. It is used to associate a Facebook user with a score and save it in the cloud. This enables getting the score of your friends. Using Facebook for saving scores is only a temporary solution. If this project were to be further developed we would advise creating an own online database with scores and users, using Facebook as an optional feature. Then it would be possible to allow for other social networks as well, such as Google or Twitter.

Android support libraries

We use multiple Android support libraries both for getting backwards compatibility (since we want to support a minimum of API level 15) and for getting some pre made official view components which would be very complex to implement manually.