Time Your Trip Design Document

Names: James Zhou, Yixin Zhao, Sirine Trigui,
Navjashan Singh, Johannes Gallmann

Introduction

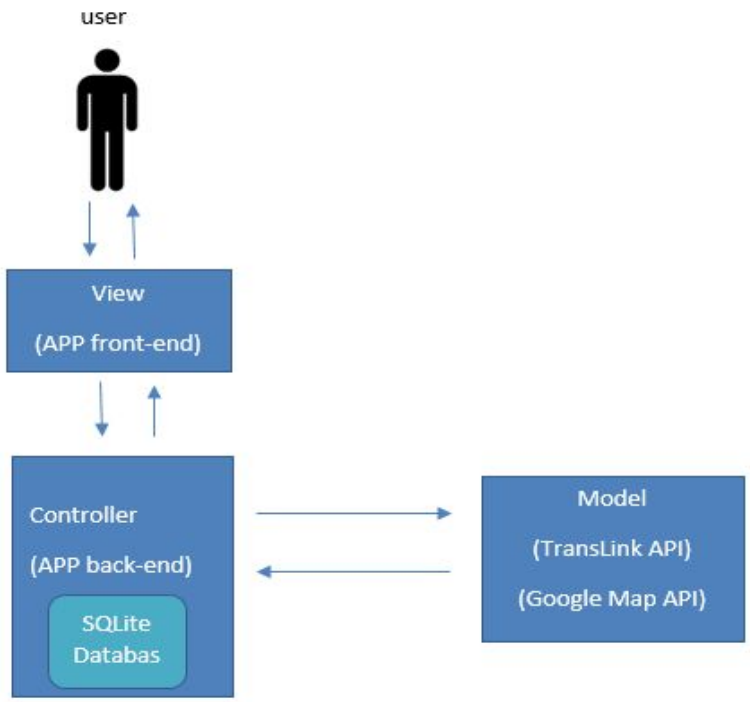
'Time your Trip' is an Android application which notifies the user when he or she has to get off the bus. To achieve this, the app will utilize the TransLink Open API to obtain real-time bus departure and arrival times, as well as the static information (ie. all bus stops for a particular route) to calculate an estimated time for a user to arrive from one stop to another for a chosen bus. The app will also be utilizing GPS coordinates and the Google Maps API to improve estimated time calculations. This document serves to describe the overall design of our product.

Architecture and Rationale

The overall application is composed of the following major components:

1. Android application that can send HTTP requests via TransLink API, and parse and display relevant information in JSON. It is also connected to an embedded SQLite database.
2. SQLite database that follows a relational model and contains tables of static General Transit Feed Specification (GTFS) data.

One of our system architecture is the Model-View-Controller:



- Data (Model): Translink API
- An interface to view and modify the data (View): App UI
- Operations that can be performed on the data (Controller): Back-end

The MVC pattern:

1. The model (Translink API) represents the data, and does nothing else. The model does NOT depend on the controller(Back-end) or the view(App UI).
2. The view (App UI) displays the model data(Translink API Data) in JSON format, and sends user actions (e.g. button clicks) to the controller. The view(App UI) in this case is independent of both the Translink API and the Back-end.
3. The back-end provides model data in JSON format to the UI, and interprets user actions such as button clicks. The controller depends on the view and the model.

Android Studio

We chose to use Android Studio IDE because it is the environment that will be able to support all of our features for the project. Android Studio utilizes Java as the programming language, a language that is robust and supports Object-Oriented design. Object-oriented design is crucial for the overall implementation of our system, and will be explained in more detail in the Detailed Design section.

Android Studio also utilizes XML to create UI, and we will be using this feature to make the GUI for our app that will be visually pleasing and will guide the user through a series of steps to achieve the overall goal (setting an alarm).

SQLite

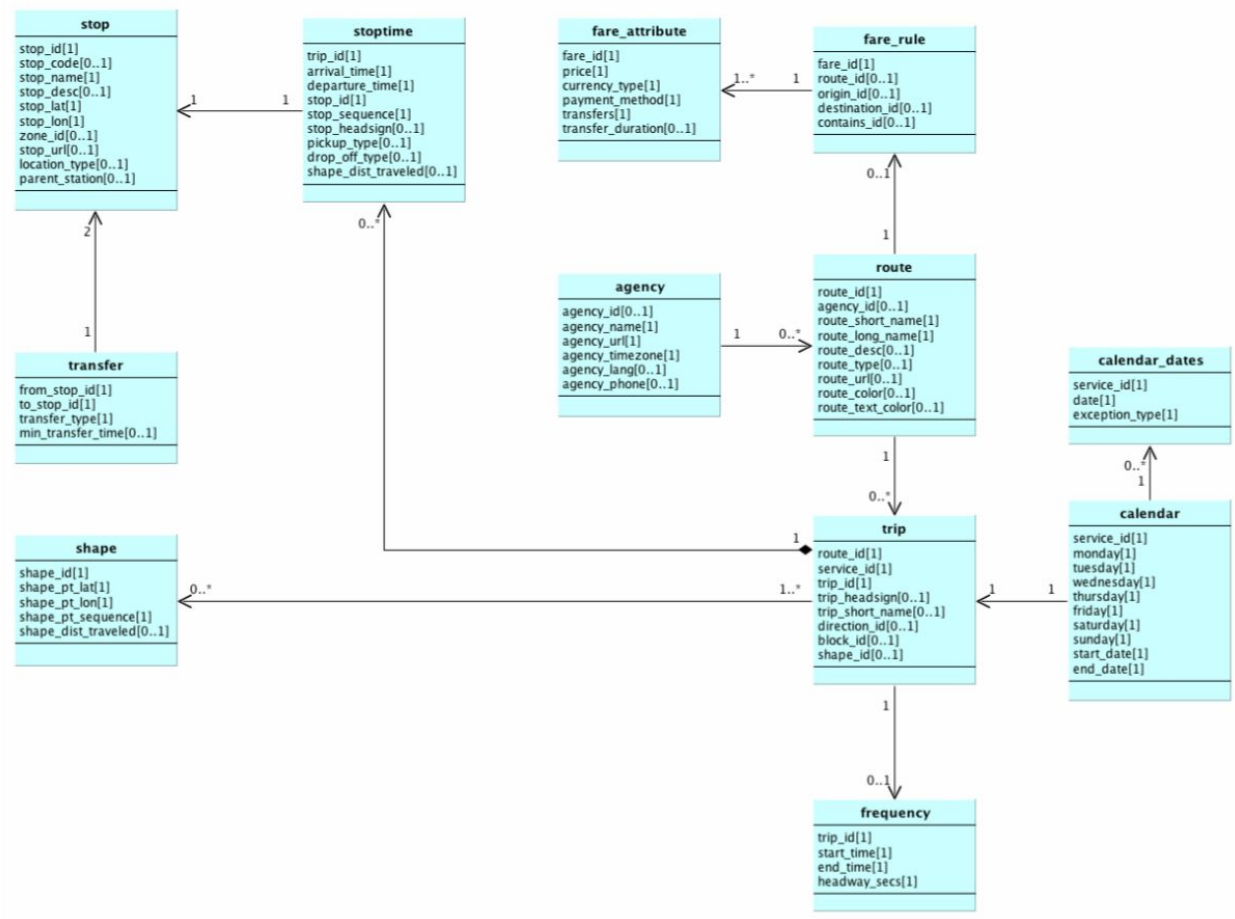
Our group will utilize an embedded SQLite database to store static Transit data. SQLite will allow us to build relational databases, and this level of organization will be beneficial for us, as our app will be sending queries to, and retrieving information from, this database. SQLite is also supported by the Android Framework through APIs, allowing it to be easily integrated into our overall project. Lastly, SQLite does not require an additional server. Rather, it is embedded into the actual mobile application. This will allow us to add an offline feature for our app, where users don't require constant internet access.

Data

We will be using the SQLite database to store GTFS (General Transit Feed Specification) data, an open data format for public transportation schedules and associated geographic information. This static feed is available for download on the TransLink Open API website, and is updated every two weeks with renewed transit info for all buses. The GTFS static feed contains 12 files, and relationship between these files are

detailed in the diagram below. Once our app downloads this feed, it will organize the contents into a relational database as shown in the diagram.

Figure 2.1: GTFS Feed Diagram

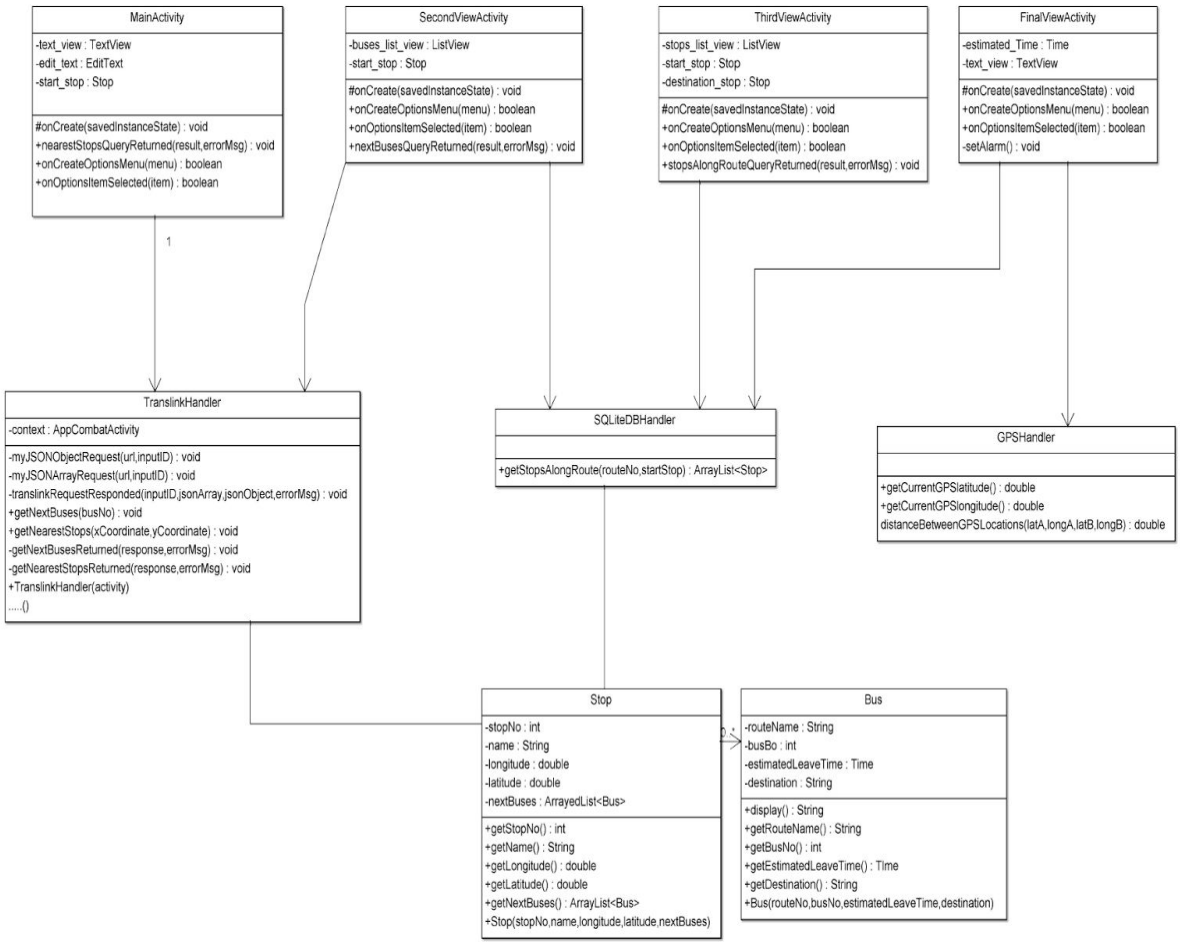


Source: https://developers.google.com/transit/gtfs/

Detailed Design

Our design will utilize Object-Oriented principles. Our overall software is composed of multiple interacting class, where each class is only responsible for one particular task.The project will have the 4 different Views that can be shown to the user. Every view has a corresponding View-Controller class which manages all the UI details as well as the user inputs. The View-Controller classes communicate with the three underlying model classes. The first model class is the *TraslinkHandler* class which is responsible to create queries for the online Translink database and parse the retrieved JSON objects. The second Model class is called *SQLiteDBHandler* and it is used to access the locally stored Database for queries that don't need the real-time database of Translink. The last model class (*GPSHandler*) simply provides

the View-Controller classes with GPS-Coordinate related functions such as getting the devices current GPS location or calculating the distance between two points. Last but not least we're going to use the two custom objects *Stop* and *Bus* to nicely store all relevant data related to a Stop or Bus in one object to pass it between the View-Controllers. The class diagram that represents this description is shown below.



The following sequence diagram provides a dynamic description of our system, and describes a typical sequence of function calls and user interactions with the application.

GUI

The user interface is built using Layout Editor of Android Studio, which provides developers with preview of their design.
(https://developer.android.com/studio/write/layout-editor.html)

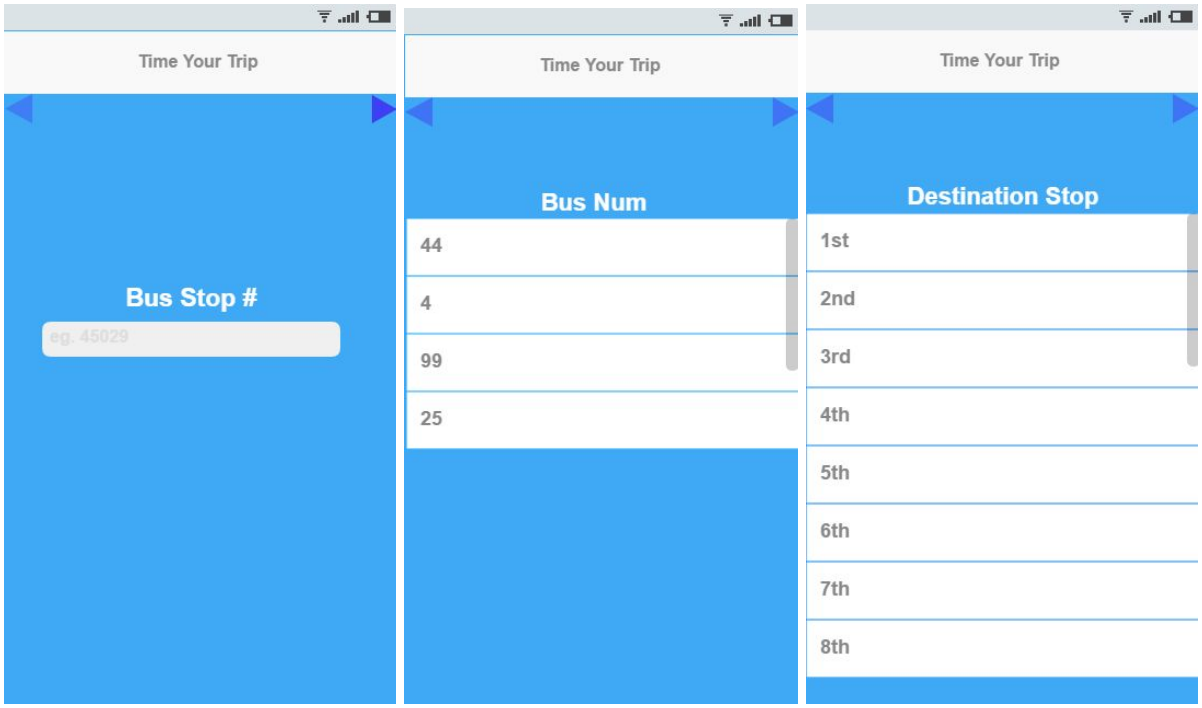
Validation

The UI validation is a continuous process in which the client was involved. The selected theme was discussed with the client and user in the process of the project and was approved. The design progress is presented on a weekly basis and client feedback taken into account to design the user interface as user friendly as possible. Changing requirements get incorporated into the design if applicable.

Launcher Page and Inputs:
1.As soon as the user launches the App, the user is going to be asked for providing the bus stop #, where he is at.

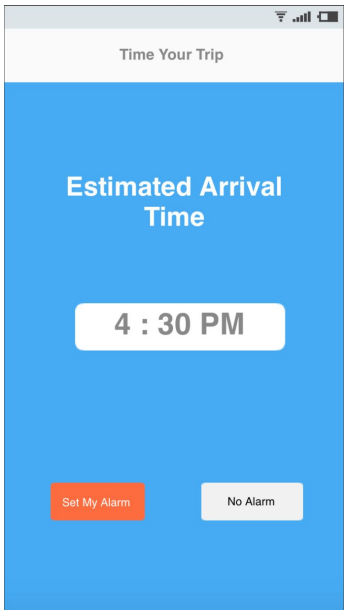
2.Providing a valid Bus Stop # and having clicked on “>” button, the user is going to be directed to a new window where they will be provided by a ListView from where they can select the bus number they would like to take.

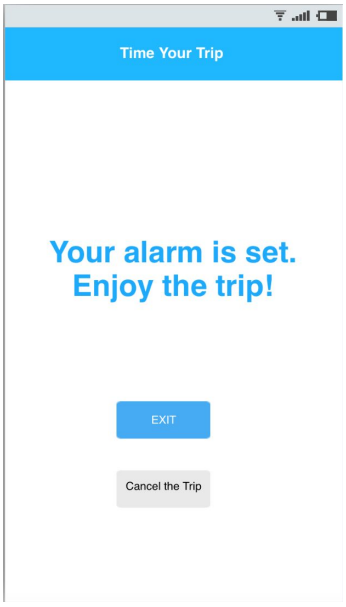
3.Having clicked on the “>” button, a new ListView of upcoming bus stops will be provided so that the user would be able to select their destination stop .

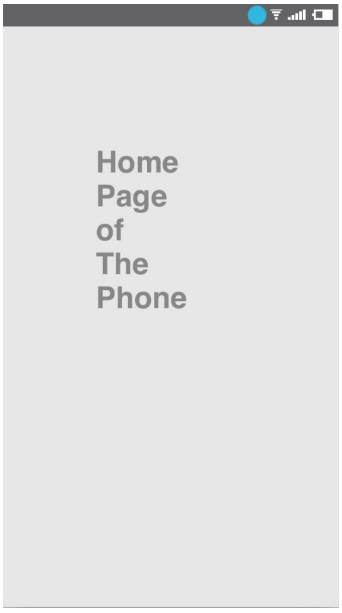


Having clicked on the “>” button, the user will be directed to a window where the estimated arrival time will be displayed with an option of clicking on “Set My Alarm” button if they want to or “No alarm” if they do not want to turn their alarm on.

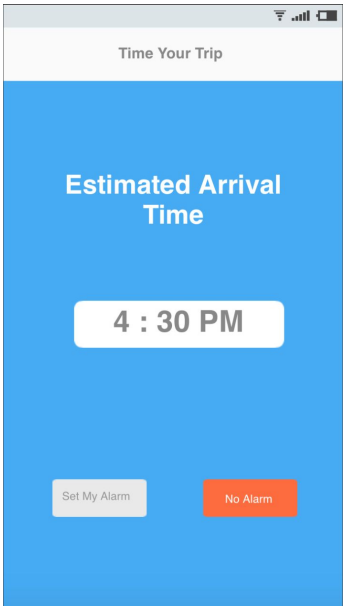
1.a) In case the user clicked on “Set My Alarm”:
Then the user will be directed to a window to be notified that their alarm is set and that they can exit the App, or choose to cancel this trip, which will lead the user to the starting page of the APP. The blue circle in the third picture is simply notifying the user that the app is running in the background.

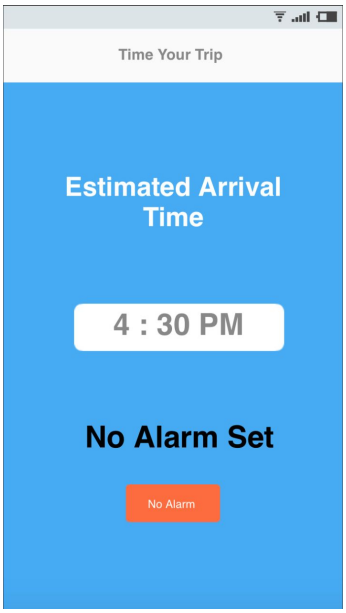






1.b) In case the user clicked on “No Alarm”:
Then the user will be directed to a window where the estimated arrival time still displayed plus a “No Alarm Set” notification.





Invalid Input Notification:
This picture is a sample example where the user is being prompted with an invalid input notification.


Time Your Trip
Bus Stop #
aaa
Invalid Stop Number