Cloud Computing Assignment 2: BusTracker  
User Guide

Sean Smyth

s3704987

RMIT UNIVERSITY

Table of Contents

[Summary: 3](#_Toc21188287)

[**Introduction:** 4](#_Toc21188288)

[**Implementation:** 5](#_Toc21188289)

[Setup: 5](#_Toc21188290)

[Search Function: 6](#_Toc21188291)

[Nearby Stops: 7](#_Toc21188292)

[Current Bus Location: 9](#_Toc21188293)

[Moving Marker Too Search: 10](#_Toc21188294)

[**User Guide:** 13](#_Toc21188296)

[**References** 14](#_Toc21188297)

Summary:

The purpose of this app was to help users traveling by bus see when and where their desired bus was. The app was based slightly on ‘tramTracker’, and another app called ‘Transit’. However, the aim of the application was to make a much leaner app that can be easily used by anyone very quickly. Buses are seldomly mentioned by PTV, getting little attention behind trains and trams. With that being said, buses service anyone living in major suburbs such as Doncaster, Templestowe and Bulleen (to name a few), as people living in those areas do not have access to any train or tram services nearby. This app should potential users in those areas navigate and find their nearest bus stop and to see when the next 5 services are. This application will also help tourists travelling by bus, as they might not know where their closest bus stop is or where it’s taking them. This application will also help users travelling by tram in the same way.

**Introduction:**

BusTracker is an android application which can tell a user their closest bus or tram stop, when said bus or tram will arrive to that stop, whether the user will make it in time to catch that service and if the user clicks on their preferred departure, the user will be able to see where that bus or tram is in real time. The original motivation behind this app was to make a main screen like Uber, where upon startup, a user can see all the buses moving around them in real time. However, this proved to be difficult to get done in the time allocated for this project. While that didn’t come to fruition, it is still a goal for a future project.

The application takes advantage of the PTV API [3], which gives real-time stop, route, departures and nearby stops information, Firebase, to access google cloud storage for the route suggestions and their real-time database to store user search queries and Google’s Map and Directions API, to access a user’s location and deliver walking time. At first the application was designed just for buses, however one change to an instance variable will change all the API calls to gather data for trams, therefore they were included. After accepting the permissions request, the app will take you to the main screen shown in figure 2. Figure 1 details the application framework, showing how the application interacts with the user and cloud services.

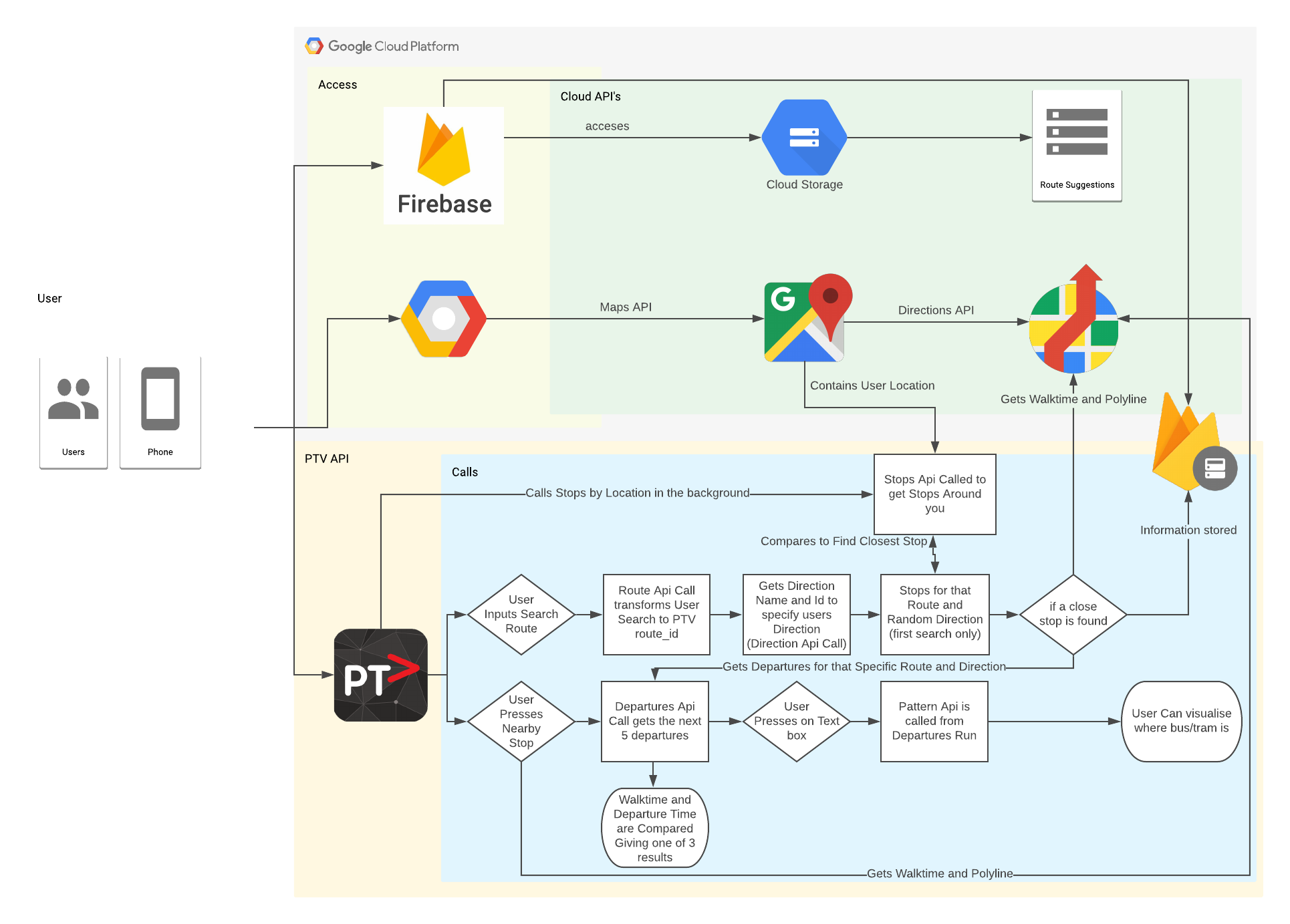


Figure 1: Application framework, detailing the how the cloud API’s and the PTV API calls interact with each other when a user uses a function on the app. Also details the process of how information is shown to the user.

This application can be used in real-life to get real time departures for a specific bus or tram, and to see where that bus or tram is on its route in real-time. It should also help users find where their closest stop is based on their location and should allow them to get an approximate location of that bus or tram.

**Implementation:**

## Setup:

* This app was built using Java in Android Studio. Download android studio and setup a new project.
* To access google maps and use their location services, get an API key from google cloud and then refer to [1]. Obtain the gradle implementations and sync the project.
* It’s recommended to make the first Activity be the user accepting that they are using an application which uses their location.
* Create a new Map Activity that will display the Google Maps API. This activity will be the main interaction the user will have with this application, as it will display all the data generated. Enable the Directions API so calls can be made to it.
* Deploy the app to Firebase. To do this just follow the guide in [2].
* Set up firebase storage with the suggestions text files (one for tram and bus) and then change the permissions on them to allow users to read them.
* Set up Firebase real-time database by changing the permissions to allow users to write to the database.
* Obtain the Firebase gradle implementations and sync the project.
* Initialize Firebase in the OnCreate method. This enables access to the storage and database from within the application.
* The application is now connected to the cloud. This allows visualizations of all the analytics that firebase provides (i.e. current users, demographics etc.).
* Acquire the PTV API. To do this a request must be made to attain a developer ID and API key from PTV [3].
* Follow their method to generate a HTTPS request and use the Swagger UI [4] to test certain calls. The API calls will return in JSON format, which can be parsed and turned into java objects. These can be either parsed onto another HTTPS request or displayed and shown to the user.

## Search Function:

The main function for this application is the search function. The user can enter their desired route, or if they don’t know the exact route, suggestions will be displayed based on what they’ve typed (figures 2 and 3). The route suggestions are sourced from google cloud storage. This method was chosen for a couple of reasons. One being that route names may change [5] or be discontinued entirely.

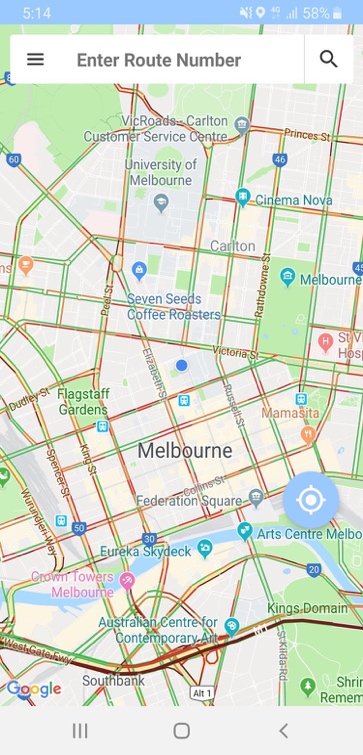
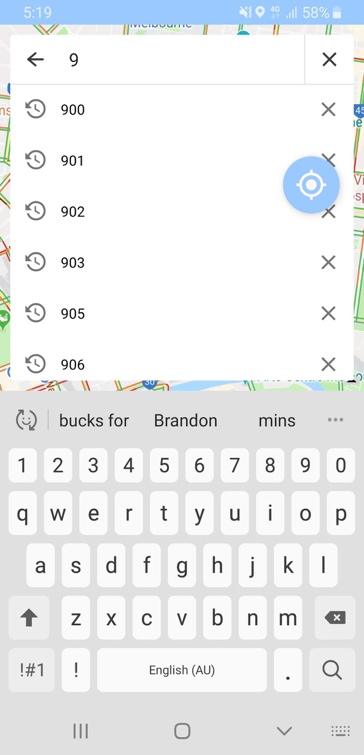


Figure 2: App Main Screen (After Accepting Permission Request)

Figure 3: Suggestions from Firebase Storage. These suggestions will change depending on whether they are searching for tram or bus.

If this occurs, a change in the suggestion text files in Firebase is all that is needed to update the suggestions. The other reason being that this application needs to be as light weight as possible. Once a user has entered in a route, their query is passed through to the PTV REST API. A series of HTTP requests is then sent off to generate the actual route ID, direction ID/name and Departures. The nearest stops list is then generated by using the location services provided by the Google Maps API in conjunction with PTV’s nearby Stops Request. This list of stops is then compared to the route and direction of the stops the user has searched for. The stop with the shortest distance in the direction specified is then returned. The Stop information (route name, direction name, stop ID, latitude and longitude) is then stored in Firebase’s Real Time Data Base (figure 4).

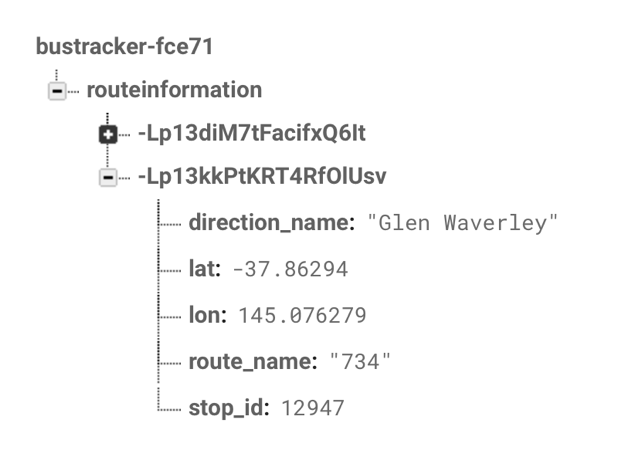


Figure 4: (left) Search call being updated in Firebase’s real time database. (right) Search call information that was recorded.

The position of the stop is then marked, and the Google Directions API will return the distance and a polyline showing the directions and walking time to that stop (figure 5 and 6).

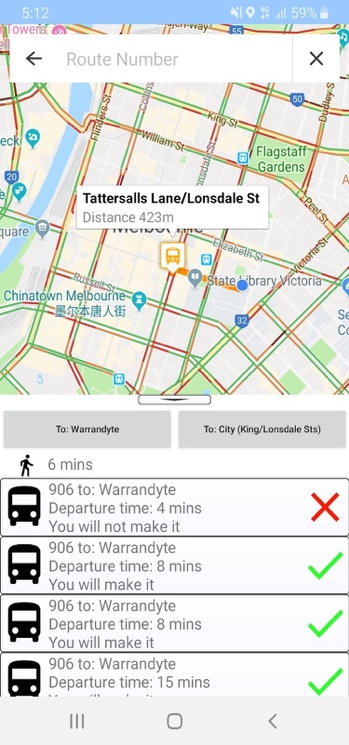
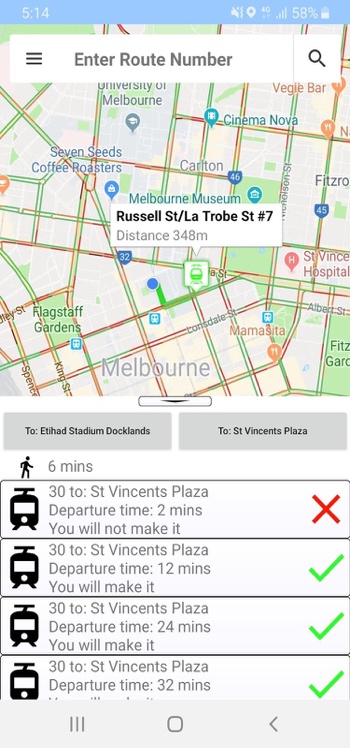


Figure 5: Bus Search Result

Figure 6: Tram Search Result

The walking time is then compared to the departure time and a message will be displayed according to whether or a user will make a certain Bus or Tram. The departures displayed run on an internal timer, which should give accurate updates to the time to arrival. The user can refresh the departures by either swiping down on the view or re-pressing the direction button. A summary of all the calls to the PTV API is listed in table 1.

## Nearby Stops:

The other way a user can get stop information is by pressing the menu icon and subsequently pressing on the show stops menu item (figure 7). This will then display (initially) the 20 closest stops for a tram or bus to the user (figure 8). If the user wants to change how many nearby stops are displayed, they can press the change stops menu item and use the slider to limit or increase the number of stops that they can display and press on, or they can search (figure 9).

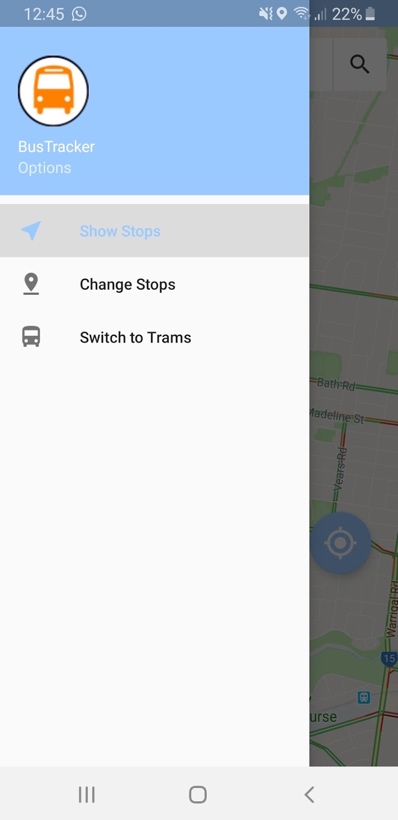
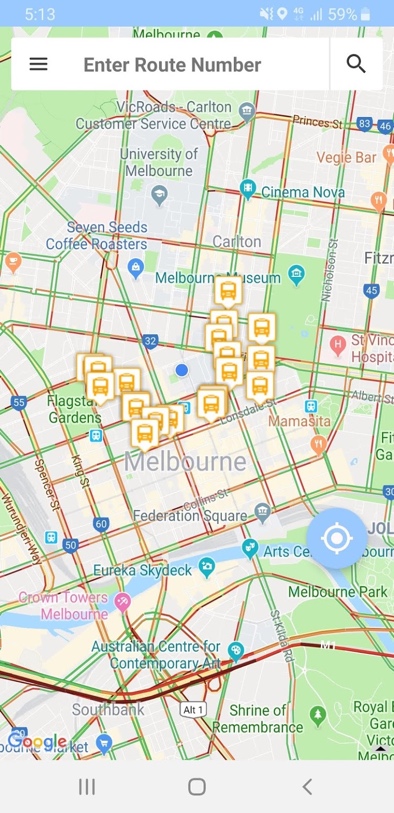
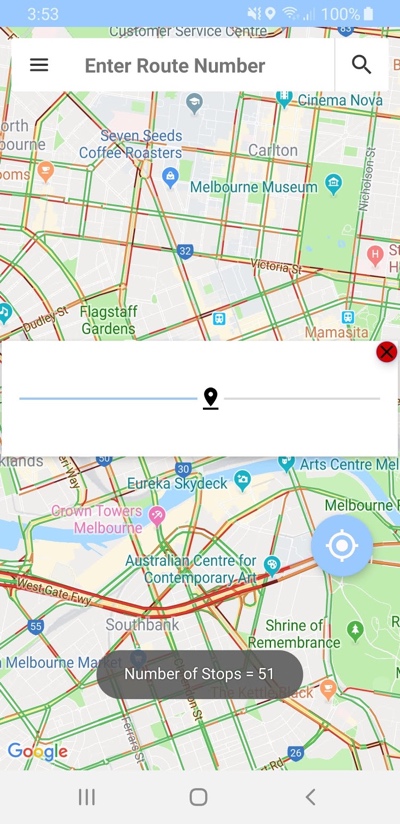
  

Figure 7: Show stops and change stops menu items. Also shows the switch to trams button

Figure 9: Change number of nearby stops

Figure 8: Show nearby Bus Stops.

If a user presses on a stop, 5 departures for that given stop are then returned to the user. Therefore, it can display different routes that use that stop (figure 10). Both the nearby stops function and the search function call the same methods to update the times until departure.

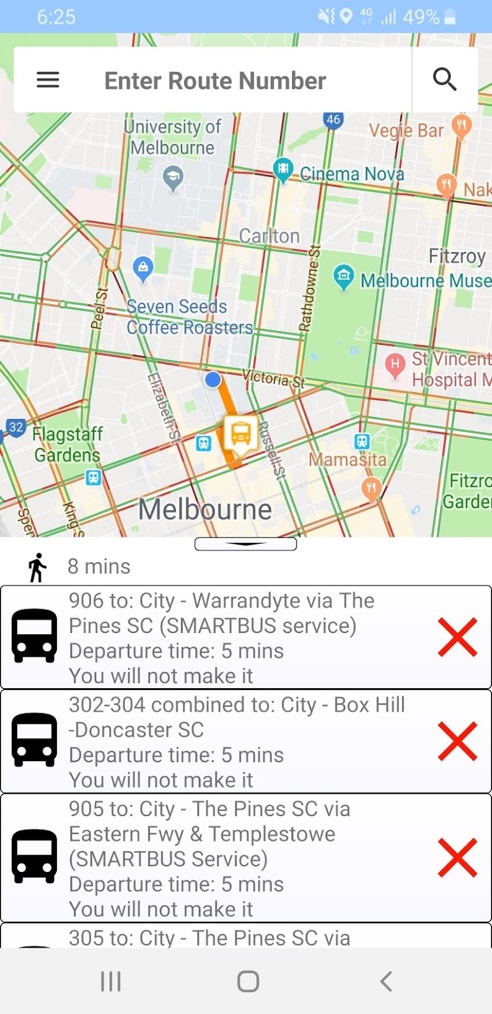
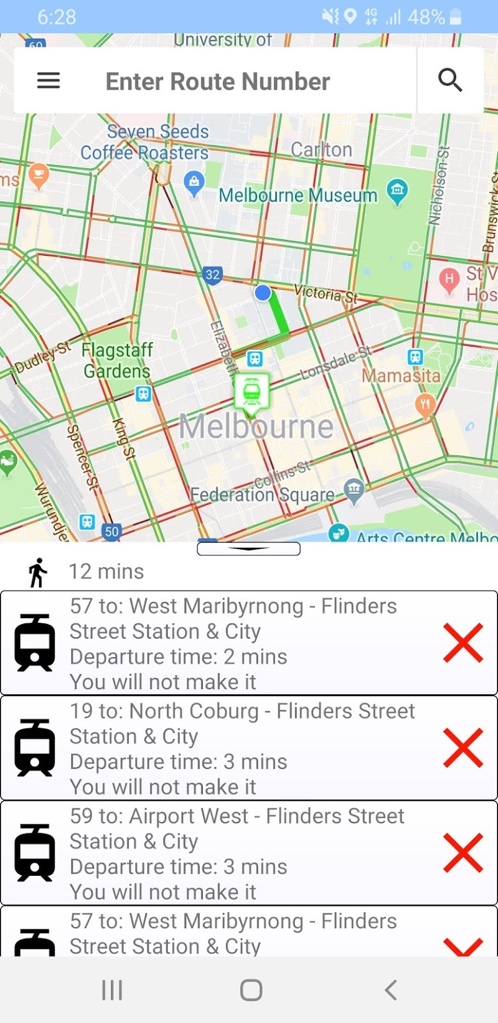
 

Figure 10: Show stops marker click result for both trams and buses. Different routes are displayed as you’re not searching for a specific route.

## Current Bus Location:

Bus Location is based on the Pattern HTTPS request (see [4] for schema), after the list of times is sorted, a calculation of to gather the minutes for each stop is parsed into one of two linked lists (priorstops or currentstops). If the minutes calculated is less than 0 it will be parsed into the priorstops list, and vis versa for the currentstops list. The bus location is the last ‘LatLng’ in the priorstops list and the map camera will move to wherever the bus or trams approximate location is (figure 11). The Patterns HTTPS request needs a ‘run ID’ in order for it to display all its information. The run ID’s are taken from the departures which was called prior. Like the departures, this function also runs on a timer, however timer runs every 30 seconds as it calls the Pattern request each time.

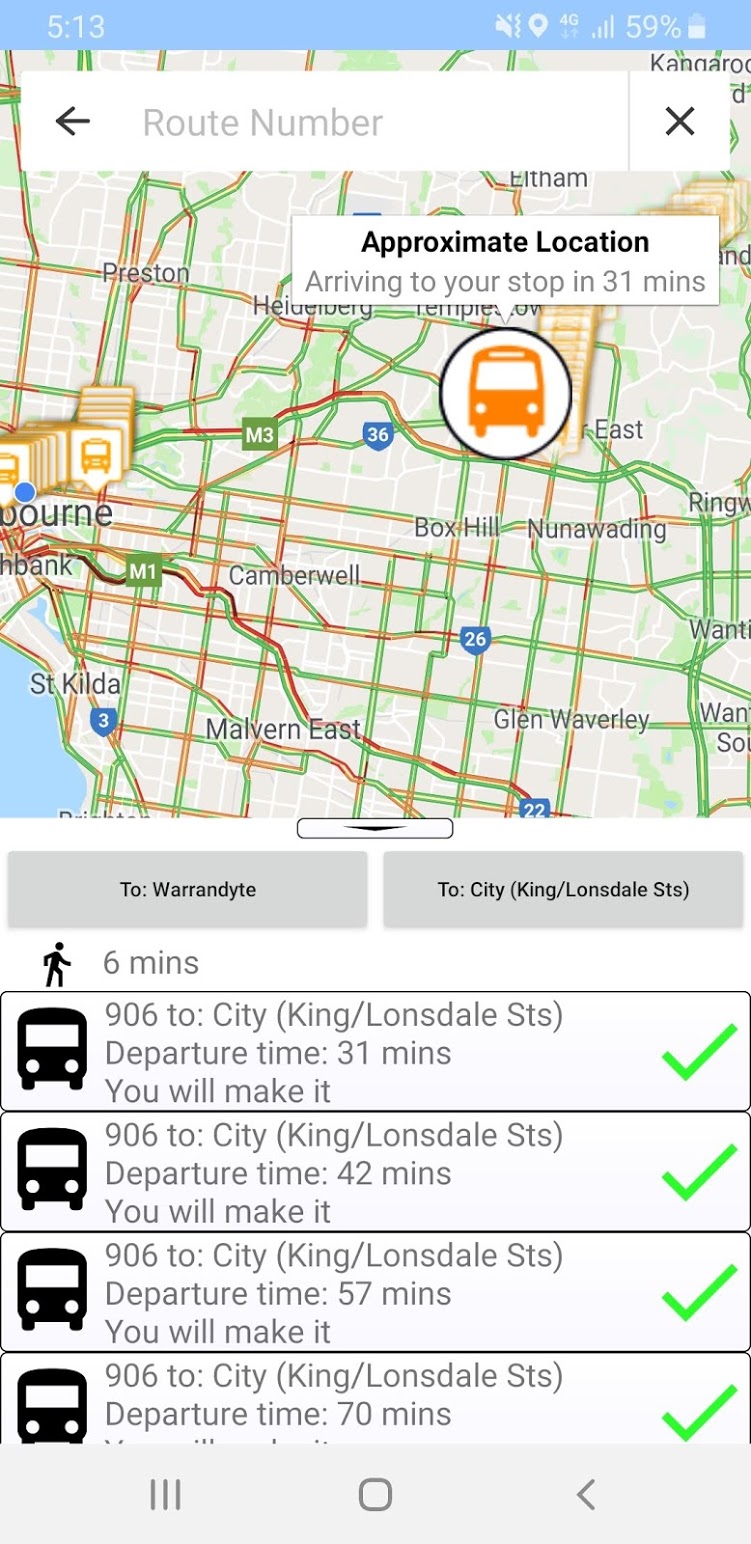
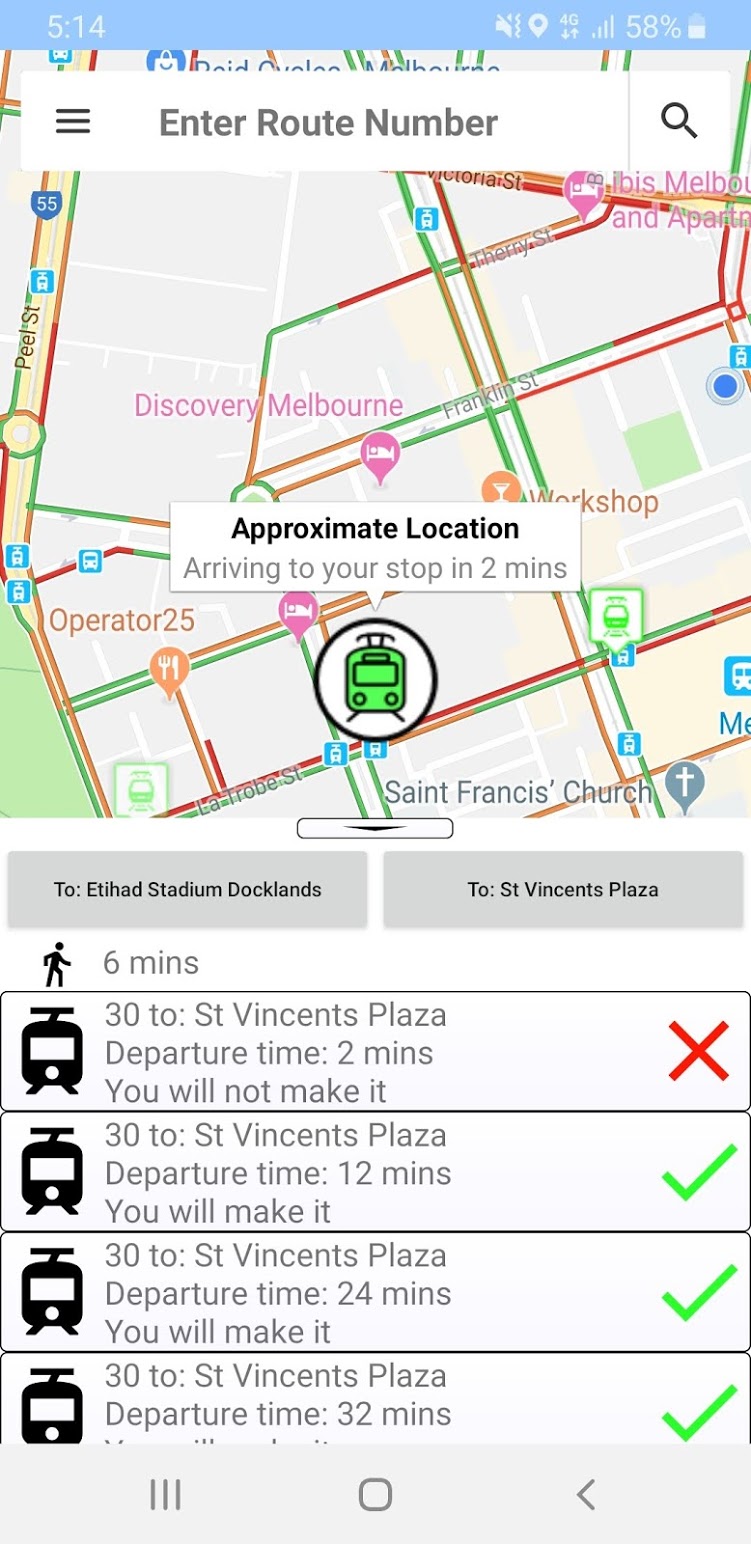
 

Figure 11: Bus and Tram approximate locations. The zoom level updates depending on how close the tram or bus is to the stop you specify or searched for.

## Moving Marker Too Search:

A user can also search in different locations rather than their own. They can do this by dragging the marker to a location and then using any of the previous functions. This method allows for more flexibility for each function. This method works by calling the map marker drag listener [1]. Once the user places the marker, the latitude and longitude of that point is then parsed into the nearby stops HTTPS request from the PTV API. A user can then use the prior functions to search for trams or buses in that area using the same logic (figure 12). To deploy the app to google play refer to [7].

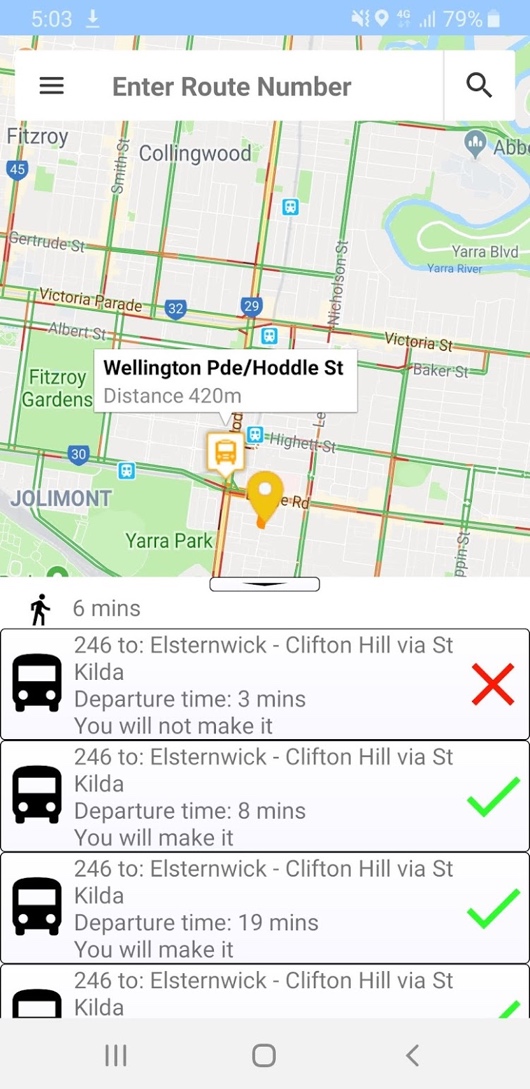


Figure 12: Nearby Stops Function used when user has moved the marker to their desired location.

Upload to make it live:

Table 1: Summary of all Calls to PTV API

|  |  |  |  |
| --- | --- | --- | --- |
| PTV API Calls | Have | Retrieves | Need |
| Stops (Location) | User Latitude and Longitude | List of Nearby Stops | Route ID, Direction ID, Nearest Stop ID (for specified Route), List of Departures, Run ID. |
| Routes | User Inputted Route Name | Route ID | Direction ID, Nearest Stop ID (for specified Route), List of Departures, Run ID. |
| Directions | Route ID | Direction ID and Name | Nearest Stop ID (for specified Route), List of Departures, Run ID. |
| Stops (For Route) | Route ID, Direction ID | List of All Stops on Route for a Direction | List of Departures, Run ID. |
| Departures | Nearest Stop ID (for specified Route) after comparing to List of Nearby Stop ID, Route ID | List of Departures for a given route, Time of departure, Run ID |  |
| Patterns | Run ID | All Stops (with locations), Time of departure, direction name (nearby stops function only) for a Run |  |

**User Guide:**

* Accept the permissions on the first activity. This will allow access to the main map activity (see figure 2)
* A user can either search for a route or pick a close by stop.
* Once a nearby stop is picked a user has searched for a desired route, it will either return with the closest stop (see search) or return ‘closest stop not found’.
* If a nearby stop is found (after search) or a user has selected a stop location, a list of departures will be generated. If a user wants to see get an approximate location on their desired mode of transport in real time, press on a departure and the approximate location of the bus or tram will be shown to the user.
* If the bus or tram is at the end of the route or there is no information, the application will display the bus or trams point of origin.
* If a user wants search for trams, open the menu and press on the ‘switch to trams’ menu item.
* If a user wants to see and search more stops, open the menu and press on the ‘change stops’ icon. This will open a slider. Move the slider to change the total amount of stops shown.
* If a user wants to search a different area, drag the marker to a desired location and the previous functions will work for that location.

**References**

[1]"Project Configuration  |  Maps SDK for Android  |  Google Developers", *Google Developers*, 2019. [Online]. Available: https://developers.google.com/maps/documentation/android-sdk/config. [Accessed: 24- Sep- 2019]

[2]"Add Firebase to your Android project  |  Firebase", *Firebase*, 2019. [Online]. Available: https://firebase.google.com/docs/android/setup. [Accessed: 24- Sep- 2019]

[3]"PTV Timetable API", *Public Transport Victoria*, 2019. [Online]. Available: https://www.ptv.vic.gov.au/footer/data-and-reporting/datasets/ptv-timetable-api/. [Accessed: 24- Sep- 2019]

[4]"Swagger UI", *Timetableapi.ptv.vic.gov.au*, 2019. [Online]. Available: http://timetableapi.ptv.vic.gov.au/swagger/ui/index. [Accessed: 24- Sep- 2019]

[5]"List of Melbourne tram routes", *En.wikipedia.org*, 2019. [Online]. Available: https://en.wikipedia.org/wiki/List\_of\_Melbourne\_tram\_routes. [Accessed: 24- Sep- 2019]

[6]"List of bus routes in Melbourne", *En.wikipedia.org*, 2019. [Online]. Available: https://en.wikipedia.org/wiki/List\_of\_bus\_routes\_in\_Melbourne. [Accessed: 24- Sep- 2019]

[7]"Publish your app  | Android Developers", *Android Developers*, 2019. [Online]. Available: https://developer.android.com/studio/publish. [Accessed: 24- Sep- 2019]