Rachel Hart

TripTipper Project Description

The mobile application, TripTipper, is so named because it tips off the user at various points in their trip. As a frequent rider of the MBTA subway, I have noticed almost daily that, as the train rolls into a station, someone will look around frantically for some indication as to whether this station is their station, often only slipping out of the car just as the doors are closing. I, myself, have had this experience many times, and have missed my stop by several stations more often than I’d like. When I did a search for something that could help me in the google play store, I was surprised to find nothing for the MBTA. In fact, the only app giving train station notifications was for a train line in India, and the next closest thing I could find was a general GPS notification app. Since there is nothing on the market to solve this problem for me and other inattentive riders of the MBTA, I decided to use that idea for my project. While designing the app, I tried to keep in mind that it could one day be used by people with impaired vision, which drove the design to be high contrast, simple, and well-labelled.

In doing this project, I got to learn about many of the functionalities used in the app we tested out in homework 4. Some parts of my app are based on my study of that example, particularly the handling of locational data. My design is tab-based, with one tab for setting destinations, and the other for displaying the station name. To switch between the tabs, the user can tap on the tab itself, or swipe the screen. I was originally going to use a button system that when pressed would switch to the other fragment, but when we were introduced to the tab system, it was clear that it was much more user friendly to be able to just swipe between tabs. This is particularly true if the app were used by the sight-impaired because it removed the need to find those particular buttons. I also kept my colours darker because bright schemes are a drain on battery. Energy saving is important for my app because anyone using it is definitely away from home and is likely to not have access to a charger for a while. I included a gpx file to simulate a short trip from North Station to State.

I currently have a limited selection of stations supported as proof of concept. They are selected with switch toggles in a scroll view on the menu tab, which makes selection very easy, but if all stations from every line were added, I would probably need to add a button for each line that would open a scroll view for the stations just in that line. That’s something I would have implemented if I were working with a partner. Swiping will open the trip tab, which initially just displays “Station”, but when the region of a station is entered, it will display the name of that station. As the station is departed, the display will change to indicate so. It is also meant to change to a yellow background from a navy blue background and vibrate when a destination is reached, however my issues with that will be addressed later in this discussion.

The MainActivity class handles the tab function, while the actual functionality of the app is handled in fragment classes for each tab. MainActivity extends AppCompatActivity which allows for the use of fragments in the tabs. The UI design of the tabs is set programmatically in this class to go with an overall darker navy/grey/white theme. The tabs use a page adapter to manage switching between fragments on different tabs. Here, the page adapter is invoked, but it is implemented in another class.

The PageAdapter class does the logistical work for the tabs. A switch statement is used to invoke the class for its respective tab when that tab is the one selected. Tab1 references the MenuFragment class, which is where destinations are selected, and tab2 references the TripFragment class, which displays the station. For a simple two-tab design the implementation is quite straight-forward.

In the MenuFragment class, the main function is to gather the information from a series of switch toggles. There is a toggle for each station supported by the app, and they are added in the menu\_fragment xml file, the view for which is inflated in this class. A reference to each toggle is retrieved using findViewById, and for each reference a setOnCheckedChange listener is invoked. An onCheckChanged function uses a switch statement to check which of the toggles was changed. It then creates a JSON object corresponding to the toggled station information, which is put into a jsonArray so the toString method can be called on it. That string is supposed to be written to a file as specified in the ListItem class. This part of the program is not functioning properly.

Writing to and reading from a file is handled in the ListItem class. It has a function to check if external storage is readable and another to check if it is writable. It also consolidates Items into a list, which is used to store data as items after reading from a file. It uses inputStream and outputStream for reading from and writing to a file called destinations.txt in external storage.

Reading from the file is initiated in the TripFragment class, which does the heavy lifting for making the app function. Its associated xml file is basically just a textView which is manipulated in the TripFragment class to change as stations are entered and exited. The stations that are supported by the app are hard-coded in a function called initializeStations(), which is called in onCreateView(). Since the stations that a user might pass through are not going to change or be changed by the user, it is not an issue for these to be hard-coded and in fact should not be easily accessible to the user. Each station is made into an Item object and saved in an arrayList. The class implements LocationListner so it can create a LocationManager and request location updates every second. The onLocationChanged() function will then be called any time a change in the user’s GPS location is returned from location updates. On update, an attempt is made to read in the destinations.txt file using ListItems, and a simple JSONParser class is used to decode that data into a usable list of Items. However, I found when testing that it appears the file is not being written to. The display of each station name as it is reached is working since those are just hardcoded and kept in an ArrayList. At each change in location, the list of stations is iterated and if it is within a radius of 150m of the current location, the name of that station is displayed on the tab fragment by calling setText on the textView in the fragment. A check is also done to detect when the user is leaving the vicinity of a station, and the textView will be changed to indicate that the user has departed from that station. There is also a section of code for iterating through the stations saved to the destinations.txt file, but this functionality is prevented from working because the file is never written to. However, I have hardcoded an example to show what would happen if that file was written to. If Haymarket is entered, the screen will turn yellow with black text and the phone will vibrate to get the users attention.

This project was certainly an exercise in trial and error. I had many first attempts, for example I was initially going to use the Google Maps API to get the locations, I tried using intent to communicate the destinations across the fragments, and had several different layouts. As is so often the case when I’m learning something new, when I make some minor oversight it can turn into hours of research and debugging because I assume the problem is with the way I understand the implementation of a concept. I had quite a bit of trouble getting the change listeners for the switch toggles to work, but it turned out than I hadn’t initialized the ListItems where I was trying to store the destination Items. In the process I streamlined my code and may have fixed other issues that were there, so it wasn’t completely wasted time. I also experienced a computer crash while testing my app and when I rebooted, my most recent changes were gone and nothing I did after that would make the app work again. I eventually tried some of the example apps and they didn’t work either. It turned out that the problem wasn’t with my app at all, but with Android Studio because my app worked again after I reinstalled Android Studio.

The problem with getting the program to write to a file is the only thing not working properly, although it impairs a good portion of the functionality. I could hardcode my own destination stations for my personal use, but that would make the app very inflexible, so I left the framework for future changes. A hardcoded example for Haymarket stands as a demonstration of what would happen at a user’s destination. At first, I thought the vibration feature was not working as well, but when I did some research, I found that the AVD just doesn’t give any indication of vibration. I have tested on multiple AVDs, and while the write functionality has not worked on any, I do wonder if it might be an issue with Android Studio’s support for writing to external storage since I’m still having persistent problems with the emulator. If I had more time or a partner, I would have wanted to add more stations and have a better menu that makes it easier to navigate through and select one of what would be a great many station choices.