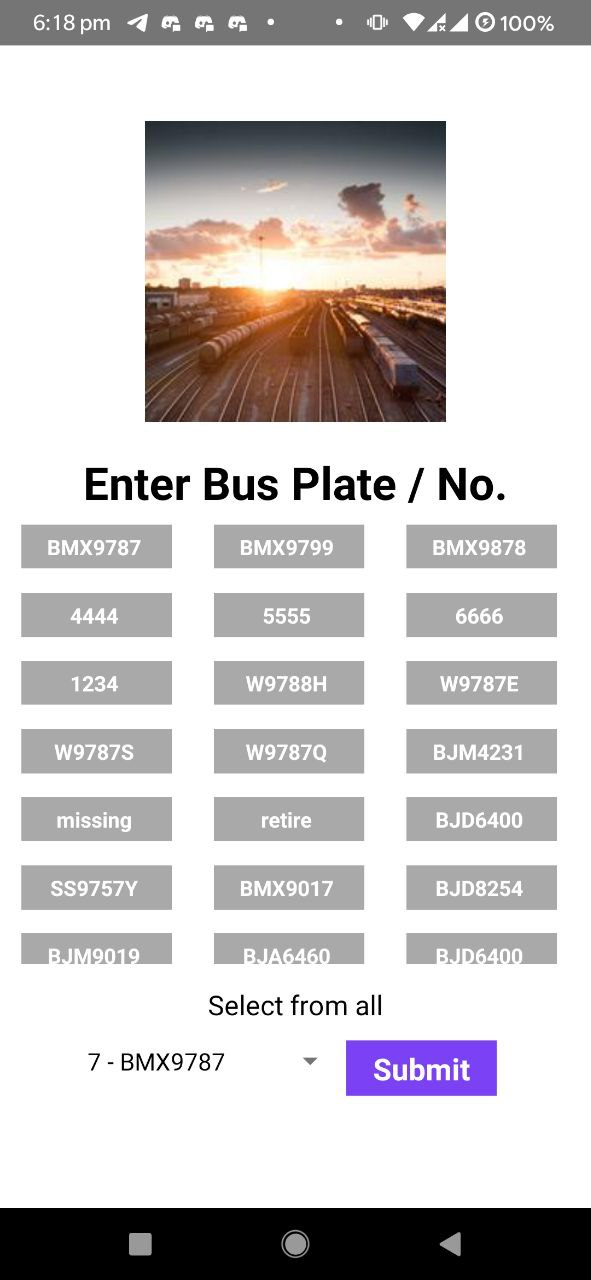
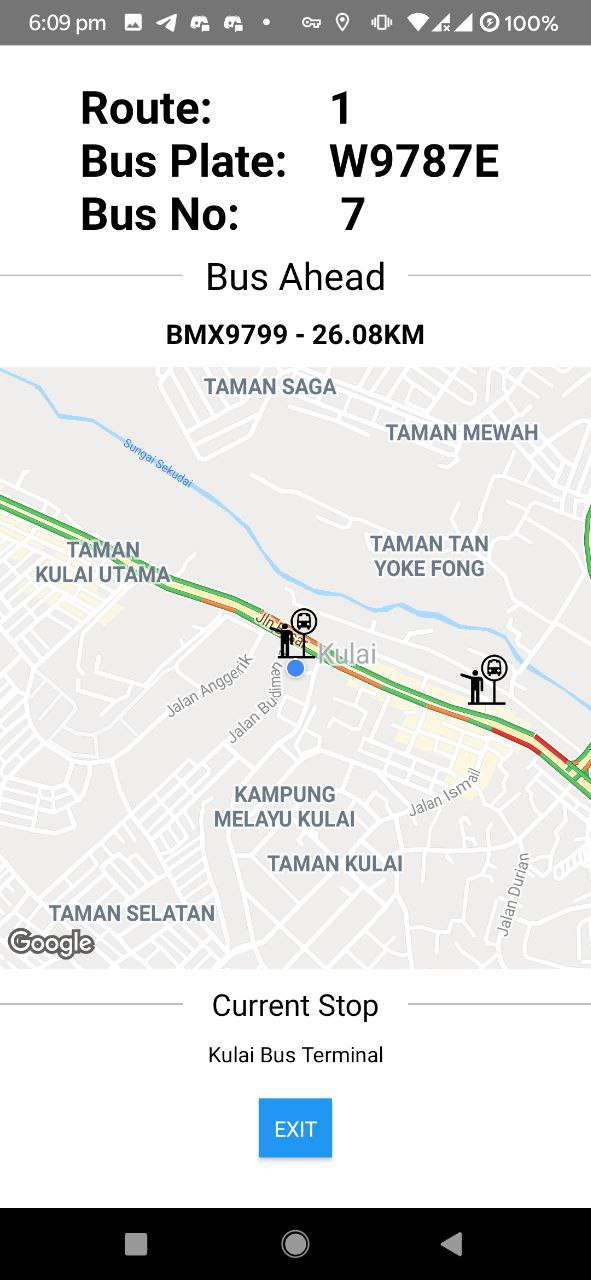
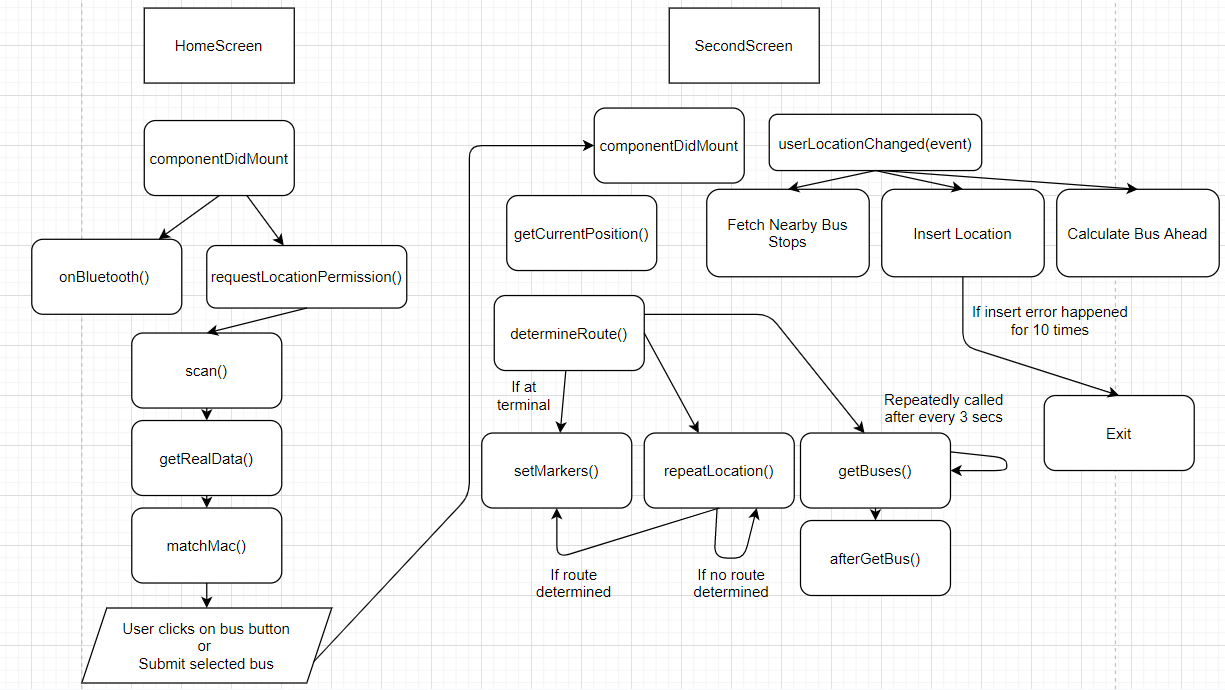
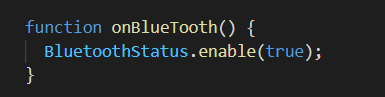
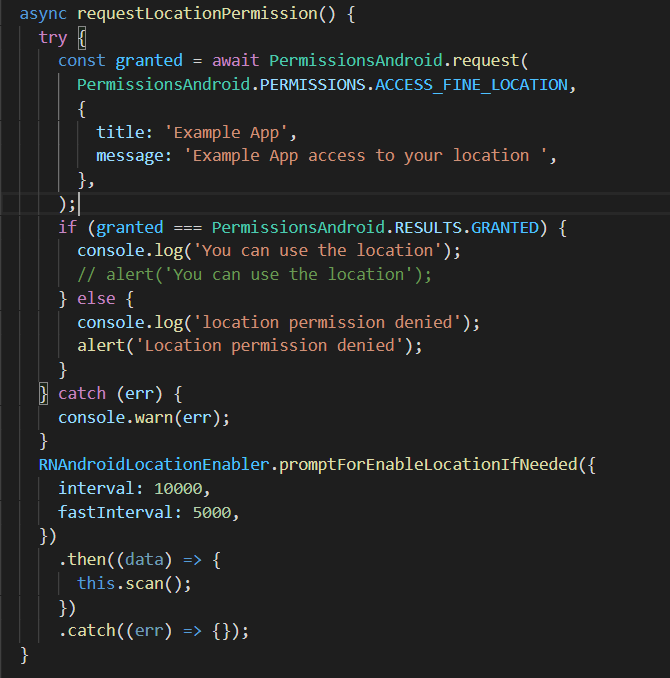
**Bus Driver App**

Name: Kwang Qi Rong Student ID: 1901829 GUID: 2508435K

App Description  
Bus drivers are travelling either route 1 or 2.   
They would use the app:  
1) To gauge how far is the bus ahead  
2) Interact with map for information — Other buses, Bus Stops along the same route  
3) App would repeatedly insert location in the backend while bus is moving   
 for other bus drivers to view.  
  
Features Summary  
Bluetooth Scanning  
Permissions API to request for bluetooth and location  
Geolocation library to always retrieve current location  
Fetching and sending using server API to communicate with DB  
Dynamically fill and update React Native UI  
Repeatedly call function using setInterval  
Interacting with Map, setting of markers and displaying.  
  
General Flow  
**When app is started, at the Homescreen:**  
1) App will start scanning for beacons and identify the closest few beacons for user to select.   
(Identify your own bus instance)  
2) If more than 1 beacon detected, show the detected few.   
 If none detected, show all bus instances for user to select   
 If only 1 detected, show only the one.  
3) Once user clicks on bus button or submit using selection picker, app will navigate to second screen.  
 **On the second screen:**4) Bus instance information will be displayed on the UI (Route, Bus Number, Bus Plate  
5) App will repeatedly retrieve all bus locations from DB.   
6) While bus is moving:  
 App will always retrieve and store bus current location   
 App will calculate and display bus ahead from retrieved bus locations  
 App will calculate and display nearest bus stop.  
 App will insert bus location to DB, if inserting not successful for 10 times, App will exit.  
6) App will use buses recorded locations to calculate route  
7) Map will be loaded with route buses and bus stops once route is detected.

Flow Diagram

Functions:  
**HomeScreen.js**  onBlueTooth(): Enables the phone’s Bluetooth.

requestLocationPermission()

Checks if location’s permission is given. Next, prompt to enable location if location is off.  
Once location is on, scan() function is called.

scan():  
If BLE scanner is powered on, an event listener is attached — startDeviceScan function.

Whenever a device is detected, the device’s MAC address and RSSI will be added into the Map “devices”.  
  
Once the detected devices has reached more than 20, the scanning would be stopped.  
  
The Map is then stored in the state scannedMacArray.  
  
The function GetRealData() will be called at the end.



GetRealData():   
Fetch all the buses stored in the database.

Once fetched, the buses’ MAC address is stored in bMacArray array.   
  
The fetched buses and bMacArray is stored in state.   
  
matchMac() will be called with bMacArray in params.   
This function would be to match the scanned MACs and the MACs from database.

scannedMacArray and busValues is retrieved from state.

scannedMacArray and busValues will be double looped to compare each other’s MAC address attribute. If both are matched, the mac address and RSSI is pushed into matchBeacon[] array.

(It is currently commented out so app can show all buses in the database on home screen)  
  
If only 1 beacon matched,   
use the matched beacons mac address to filter/search in the busValues array for the specific bus with that mac address.   
  
The filteredArray will be stored in busValues to replace it.

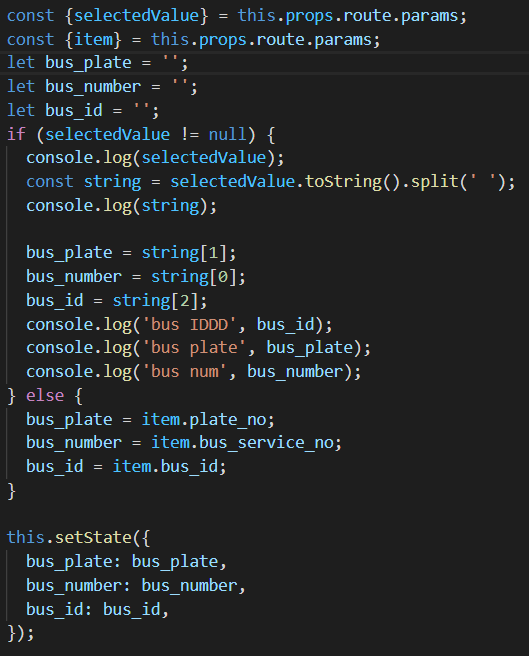
If more than 1 beacon matched:  
Loop through the matchedBeacons array and filter/search in the busValues array for the specific bus, then push the bus instance into multipleFilteredArray.

multipleFilteredArray will then stored in busValues state to replace it.

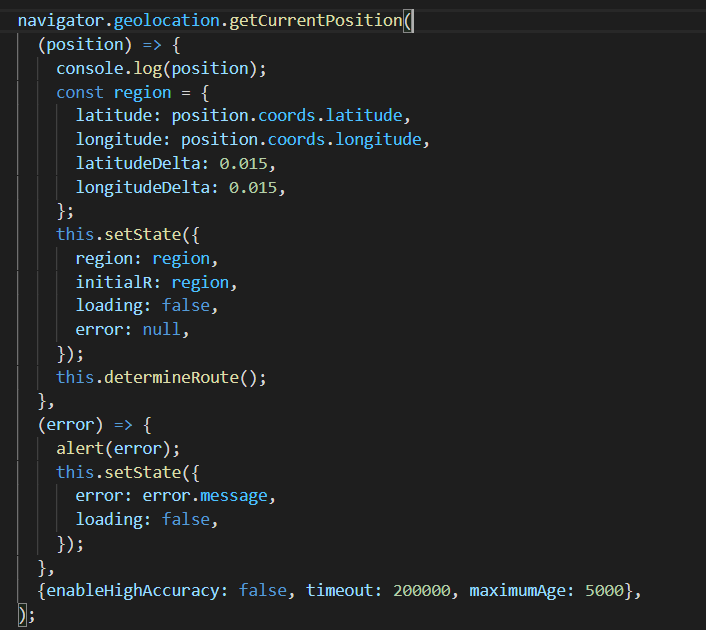
If there aren’t any matched beacons, do nothing. busValues will still be all the buses fetched from database. User will then select from all the buses.

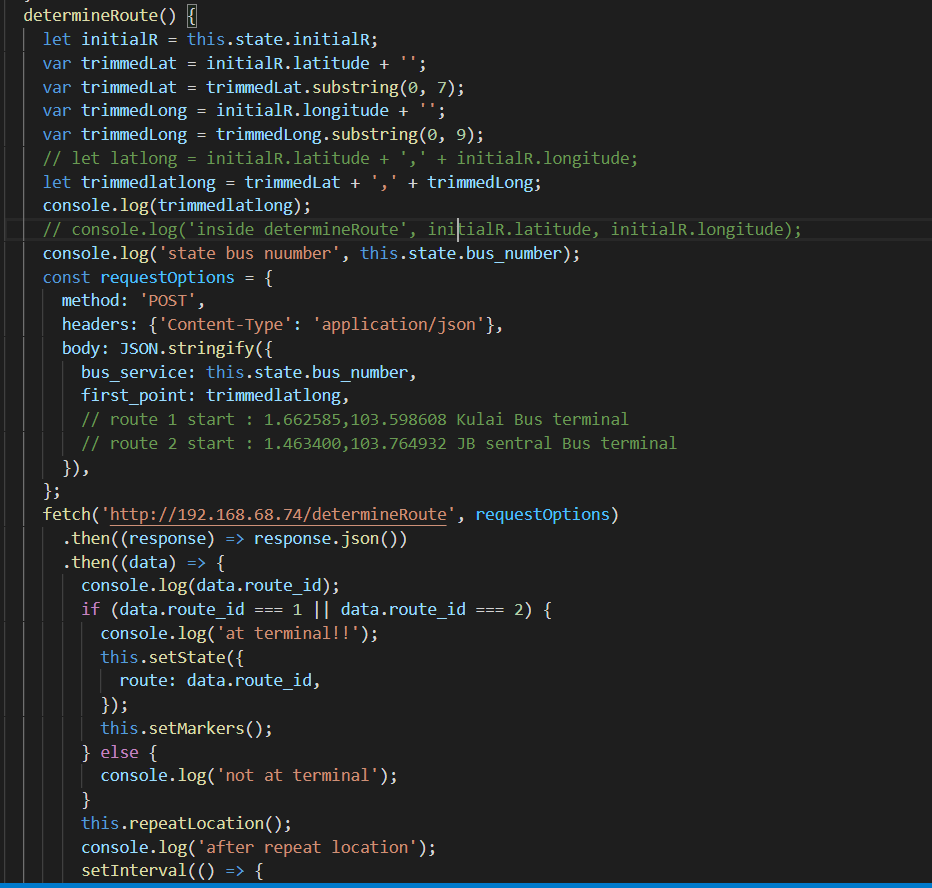
The last function getRealData() is to replace the busValues which was retrieved from DB.   
busValues is then used to populate the picker selection and the buttons for the user to select.  
Once user selects one of the buses, the bus object will be passed into the next screen with it.

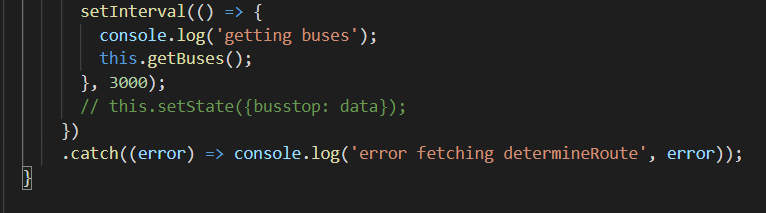
­\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

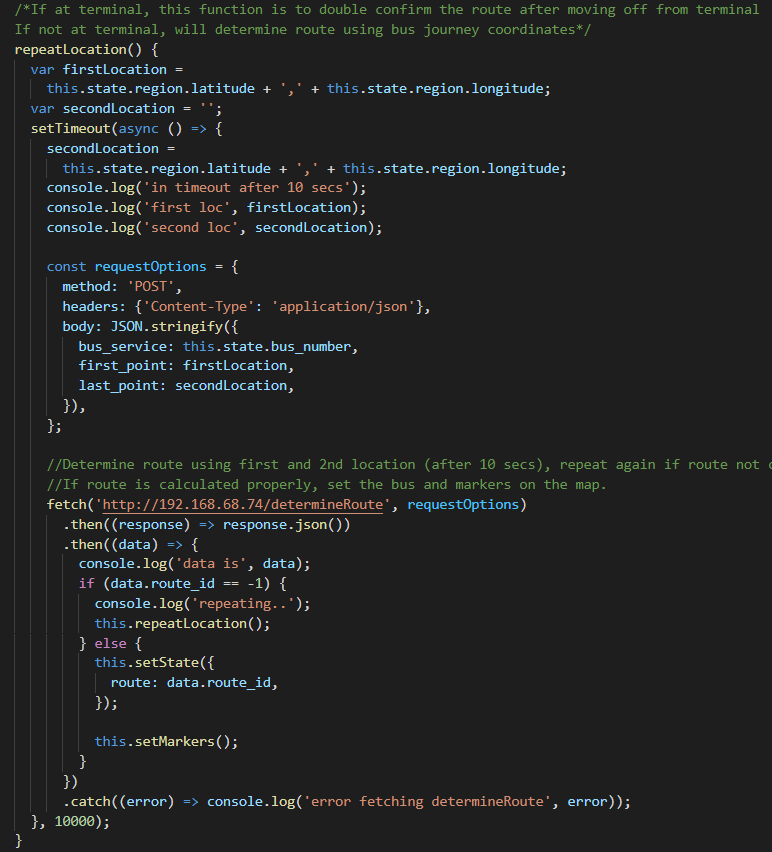
**SecondScreen.js**  
main (componentDidMount()):

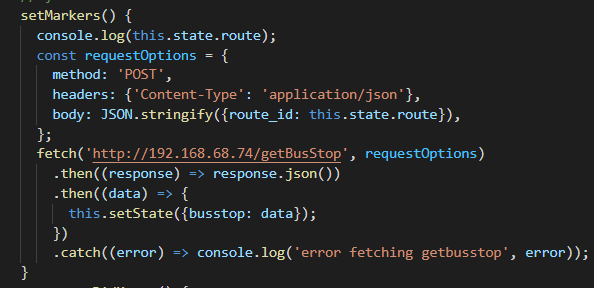
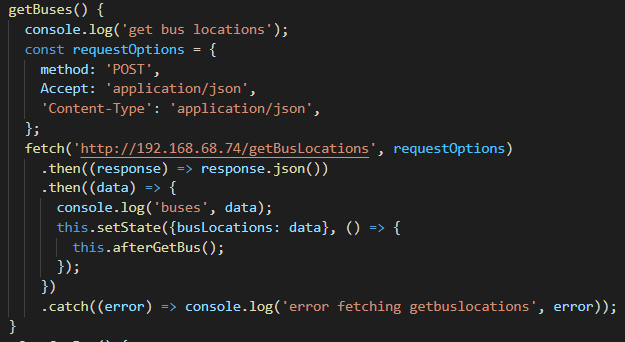
These codes are to retrieved the passed bus item value from previous page whether from the submit button or the Flatlist button.   
  
After which is to format it into proper string and then store it into the state for UI components to use and display.



When the second screen loads, it will use Geolocation to detect the bus initial location and store it.   
  
determineRoute() will then be called.  
  
  
  
  
  
  
  
  
  
  
  
determineRoute():



In determineRoute():  
initial region will be retrieved from state, and determineRoute server API will be called using fetch. This will use 1 location only to see if the initial location is at any of the bus terminal.  
If it is at terminal, we can already know which route it is at. Therefore, I stored the route in the state it is once returned from the fetch. Once it is stored, I call the setMarkers() function that will render the bus stop markers for that certain route.   
  
Even after finding out the route using the determineRoute, I would still call the function repeatLocation() which will use 2 locations which the bus is traveling through to confirm the route again. This is to confirm the route once again.  
  
Afterwards, I have a function getBuses() which is called to always fetch all bus locations in the database and it is being repeatedly called using setInterval function.  
  
repeatLocation():  
In here, the bus current location will be stored in firstLocation variable.   
After a time delay of 10 seconds, the location after 10 seconds will be stored in the variable secondLocation. These 2 variables will then be used to fetch determineRoute server API using 2 locations in the parameter. If route can’t be determined, it will recursive call repeatLocation() again until route is determined. Once it is determined, route is then set in the state.   
After which, setMarkers() will be called, filling the map with bus stop markers according to the route.

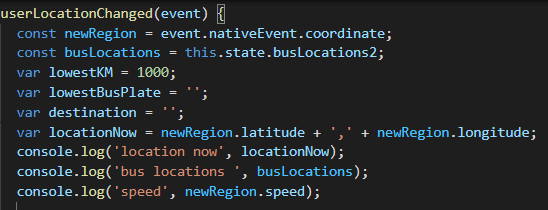
setMarkers():  
This is basically a function that gets called once route is determined.  
It will use the route stored in state to fetch getBusStop server API which returns a list of bus stops.  
The list will be stored for the UI component to retrieve to set the markers.

getBuses() & afterGetBus():  
These 2 functions helps facilitate the setting of bus markers. afterGetBus() is to add an attribute of the distance between each bus and the destination (terminal) using testGetKM server API.

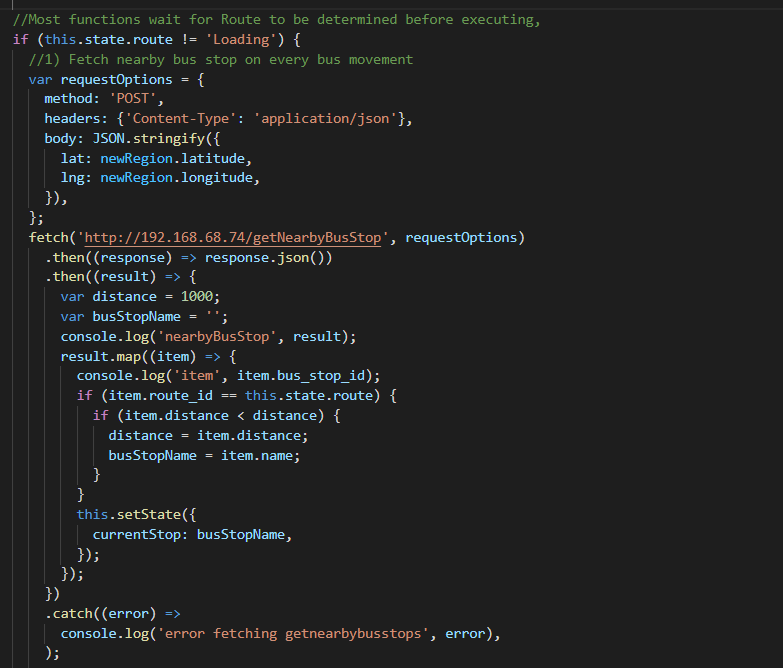
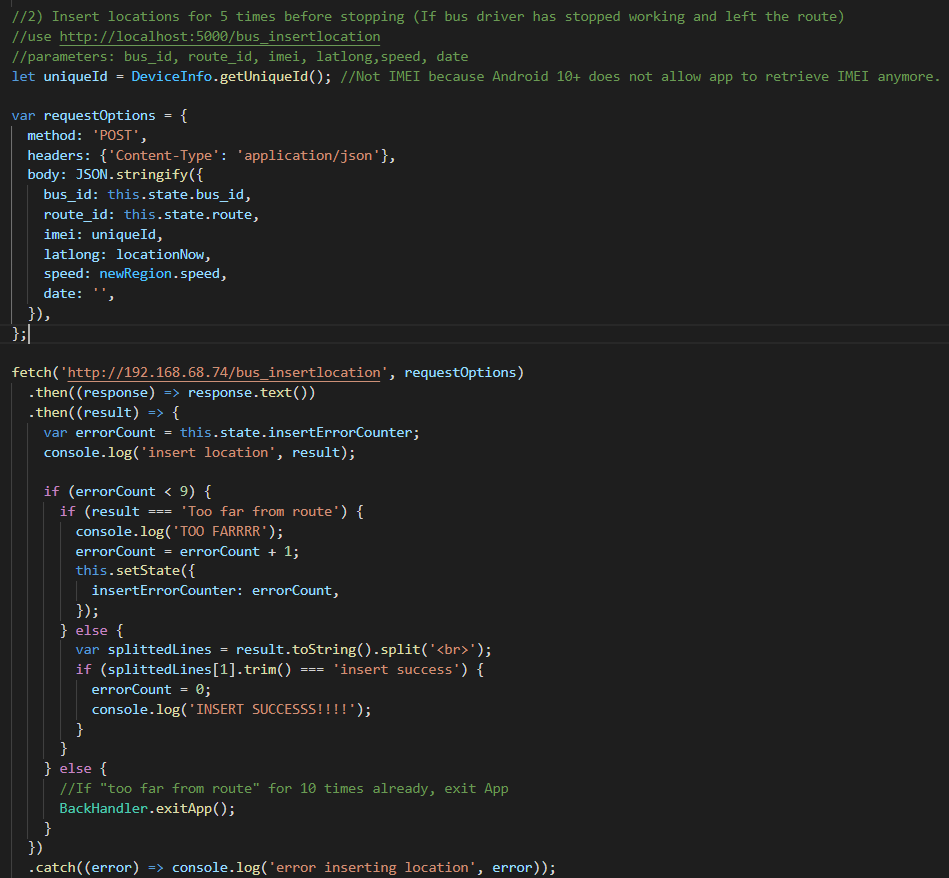
getBuses() is repeatedly getting called.   
In getBuses(), it will fetch all the buses’ location in the database and store the list in the state. afterGetBus() will then be called.

In afterGetBus(),  
the busLocations that were stored will be retrieved and iterated through its values.

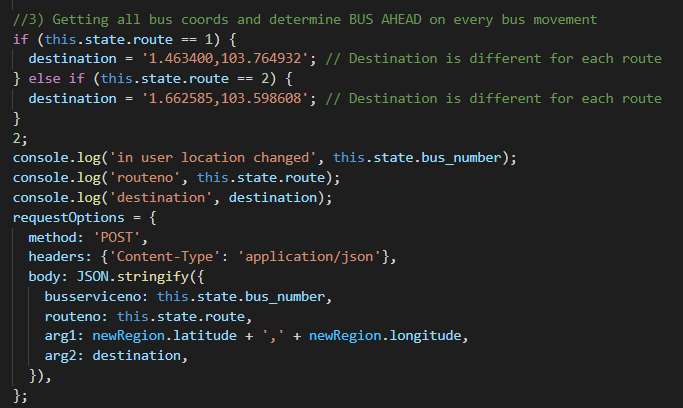
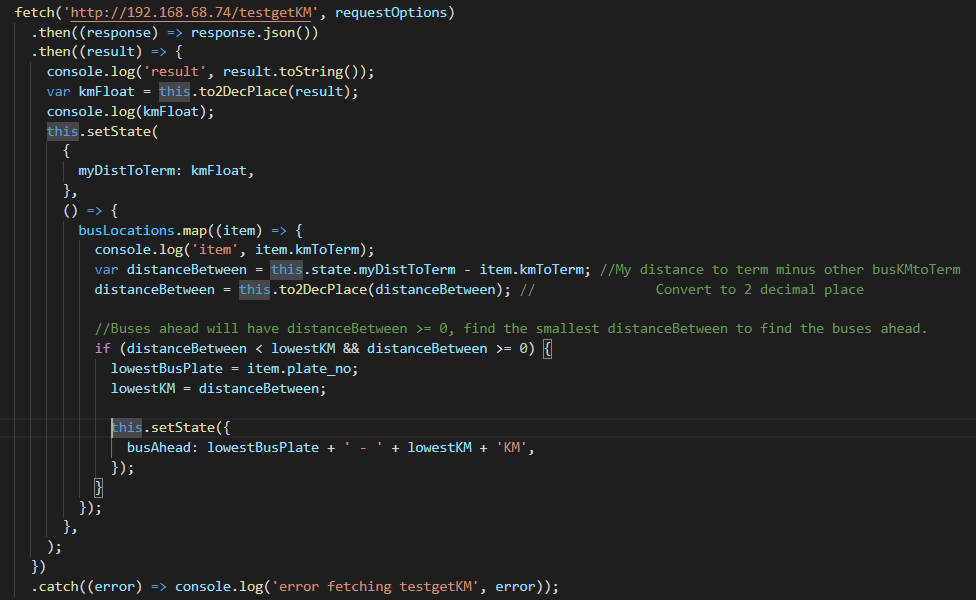
In each iteration, I will find the buses that is matching to my own bus route. If it does match — I will use my route, current location and destination (depending on route 1 or 2) to fetch the amount of distance in KM to the terminal.

Afterwards, I check if my own bus distance to terminal has been loaded. If it has, I get the variable distanceBetween (distance between my bus and the other bus).  
  
I use this and add it to the Object busLocations and other various attributes, notably kmToTerm and distanceBetween.  
  
  
I would need the distanceBetween attribute to know which specific bus is ahead of me.  
  
Afterwards, the Object busLocations is pushed into busLocArray. busLocArray is then set into state as busLocations2.  
  
**Once user location has moved (always repeating):**  
Once Route is detected, for every bus movement..  
1) Check for the nearest bus stop and display on UI.  
2) Send my bus location to DB  
3) fetch the distance between terminal and my own bus, then use it to compare with other buses to check for the bus ahead  
4) Keep updating the state with the updated location. Let map animate to updated current location.

Inside userLocationChanged(),   
newRegion variable is storing the newly updated current location. busLocations variable is stored with the state busLocations2.  
Below are the rest of the variables needed for the function to run.

First, I will check if route is loaded. Then (1), (2) and (3) will function.  
  
(1)  
Next, I am fetching the nearest bus stop with respect to my current location. Once fetched, I filter it by route and find out the one with the smallest distance. If it is the smallest distance, then I record the distance and bus stop name.  
After which, I set the state currentStop to that closest busStopName, so I can display it on the UI later.  
  
(2) Inserting of location.   
First, I get the uniqueID of the device.

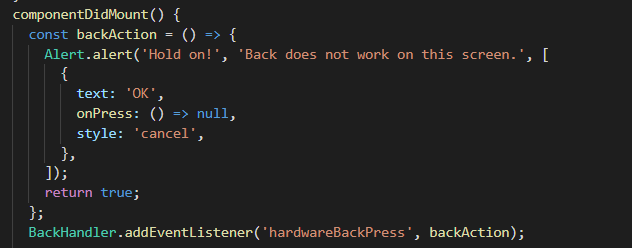
Then I call the server API bus\_insertlocation  
, sending in the current bus id, route, location, and speed.

After which, it will return “insert success” on the second line of the result if inserted correctly.   
  
Here, I check for the errorCount which is stored in the state. If errorCount is below 10, it can continue on. Afterwards, I check if the result is “Too far from route”, if it is, I will add 1 to the errorCount.   
If it is not, I check if the 2nd line is “insert success” which means it is successfully inserted. Once it is successfully inserted, I set the errorCount back to 0.   
  
Once errorCount has reached more than 10, the app will exit. This is to make sure the app exits once the driver has left the route / went home.  
  
(3)  
This is to find the bus ahead upon every bus movement because it’s always changing.  
  
I would set the destination based on my own route (1/2).  


The current location and destination will then be passed in to fetch my own distance to terminal. Once that is retrieved, I set it to myDistToTerm in the state. Afterwards, I will iterate through the busLocations. While iterating, I will calculate the distance between myself and the other buses and store it in a variable distanceBetween.   
After doing that, I will filter out and find the bus with the smallest distanceBetween and set it to busAhead, storing its bus plate and distance between the bus and my bus.

(4)

The new current location will be stored in the state region when updating.  
  
Afterwards, the map will animate to that new current region.

**Misc Functions:**InsidecomponentDidMount (main function), whenever the user clicks on back, it will alert the user by saying that back does not work on the second screen.