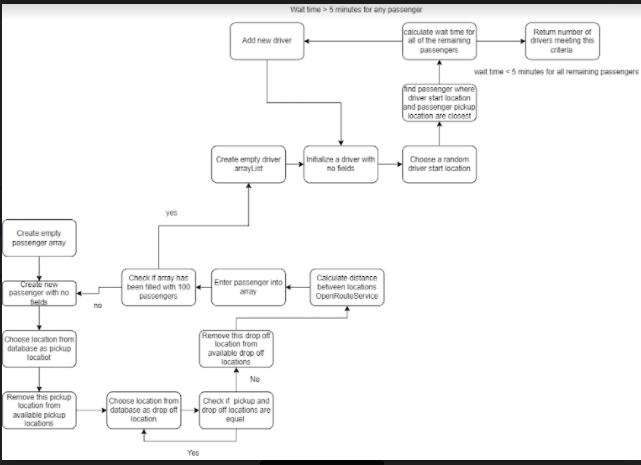
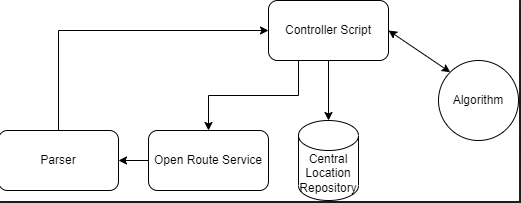
**System Modeling**

* Decided to model the system off of Rutgers New Brunswick Campus
  1. Danny has created a database (excel for now) of over 80 latitude longitude pairs encompassing Rutgers New Brunswick campus
     + Danny will finish this.
       - NOTE : Rob’s code should reflect that it is latitude longitude not longitude latitude : **To do**
* Rob created : algorithm workflow diagram.



**Architectural Design**

Rob created new architecture design to reflect repository architecture model



* Switched from using Google Maps distance matrix api to using free Open Route service
  + ORS has an additional constraint of a maximum of 500 queries per 24 hour period
* Database
  + **To do** : Write code for pulling from database (Kiernan, Rob, Danny)
    - Likely MongoDB
  + **To do** : Write code for parsing out danny's database to upload to mongodb (Brian, Wiktor)
    - Requires reading from file.
  + Software
    - Java code for pulling data from the database and giving it to the algorithm in a data structure that makes it efficient to parse for the algorithm

**Requirements Analysis**

* System model will now calculate the number of drivers it takes to run through the 100 passengers with each passenger having less than 5 minutes wait.
  + Differs from the past because before we were running a different model requiring the use of recycling and perhaps some AI we are not equipped to write.
* Identified the minimum verifiable product, a central repository for location storing with software that pulls from the database, and software that connects to the open route service api and parses out the http response, and then an algorithm to perform the relevant calculations.

**Design and Implementation**

* Rob created Github branch : Robs Branch
  + Created new file Byfter -
    - Contains java code for utilizing open route service
      * Description: Allows input of driver and passenger object array to ORSfactory class which builds a query to pass to the Open Route Service API. The Main Controller class then parses out the response to give a single driver’s distance in minutes to all 100 passengers. All in a single query. Option method created in the controller class for returning the closest person to each driver. Given the driver number (their index in the array), or the order rides are placed hypothetically.
* Danny created an excel spreadsheet with over 80 locations in lat, long pairs.
  + Rob was able to utilize these pairs in the open service to get the distance in minutes
    - Note : **To do** : Rob fix the latitude longitude pairs in your location class. They are switched. Or when pulling from database make the switch.

**Software Testing**

* Rob testing : **Open Route Service** and his Byfter project structure including ORSfactory and Controller class and some of danny’s coordinate pairs which yielded results in minutes
  + Was able to map multiple drivers to 100 passengers
  + Simply : DriverNum assigned to each driver as they are put into an array of Person objects. Drivers come first in this array. The array returned from ORS is a matrix of distances in minutes from location to location. Ie for 101 Person objects there will be 10201 distances, or 101 arrays of 101 distances each.
    - Now starting at the number of drivers input in the system. We can pick any driver, since this number is known, from the top of the Person array, and then loop through the individual rows of the matrix, starting at the DriverNum
      * The result of this algorithm is that it returns, for each driver, an array of distances from that driver, to each of the 100 passengers, excluding the driver-driver distances which are returned by the matrix. Simply put we are drawing a diagonal.
      * This is returned all in one query, meaning that the 500 query constraint from ORS should theoretically not be reached in a day unless extensive testing is taken place.

https://docs.google.com/document/d/1EzxD1Z\_v5mx7djQwvJwP8qV6f-BOZgwx4Yd3QGlu3k4/edit?usp=sharing