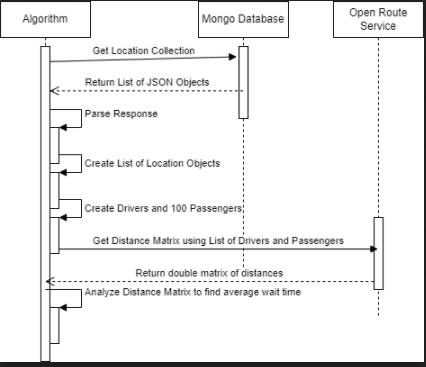
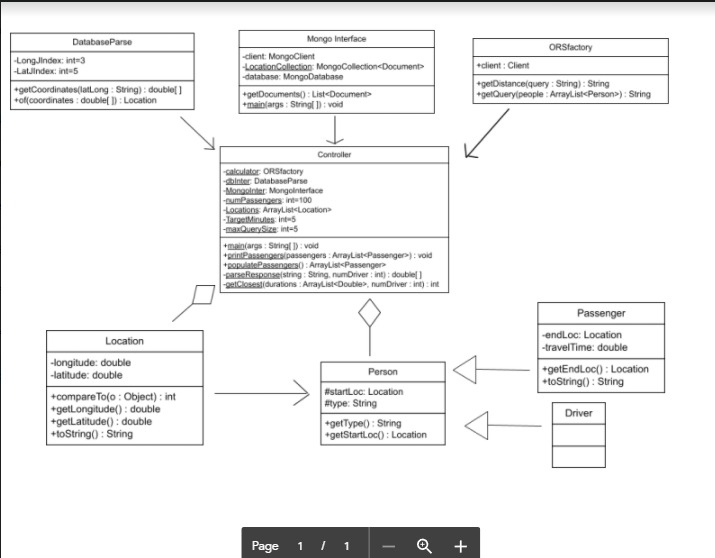
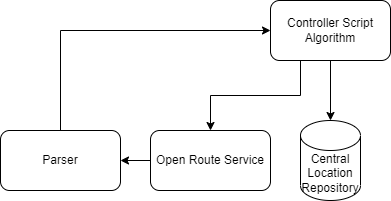
**System Modeling**

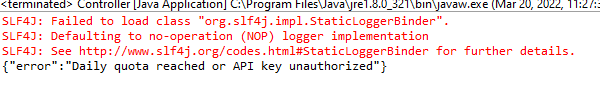
* Rob created a sequence diagram for a successful run of the Byfter algorithm interactions with the other components of the system. Assuming the dataset is pre curated and correctly uploaded, the Algorithm analysis will find and return the average wait time.
* 

**Architectural Design**

* Wicktor designed a class diagram for the Byfter project.
  + 
* Parser component had the name changed from DBInteface to DatabaseParse in order to better articulate its functionality.
* Rob updated Conceptual View Diagram.



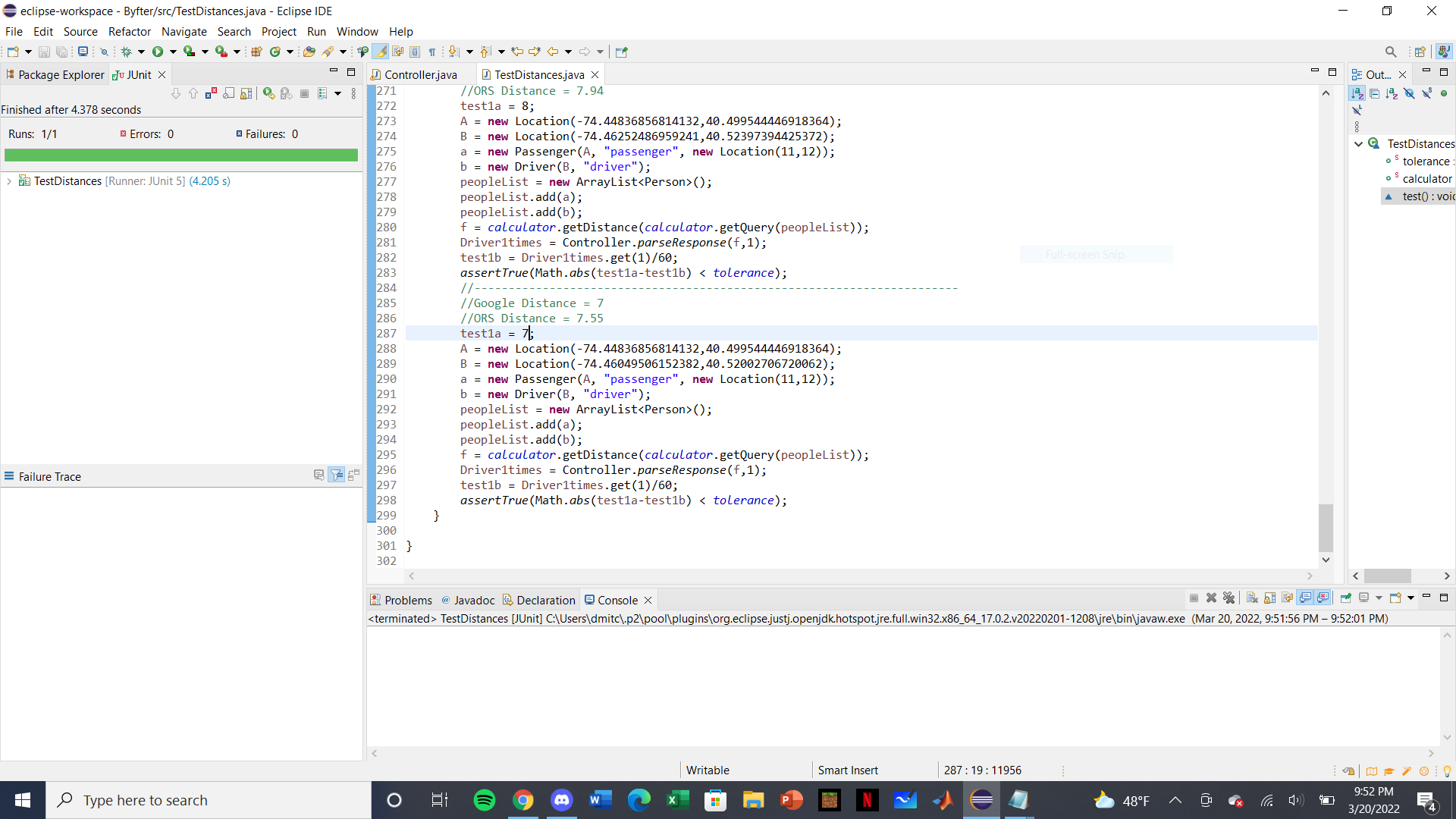
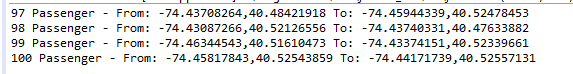
**Requirements Engineering**

* Rob discovered constraints on ORSfactory due to open route service maximum of 40 queries per minute.
  + Fixed in his code to query more often for smaller sets leading to the program being able to only be able to run a certain amount of times per day.
  + Possible fix, switching to a paid distance matrix API
    - Google distance matrix API is $10 a year and can handle this amount of traffic. Their business page is seemingly unlimited for paid customers without talking to a representative.
  + 
  + Lyfter requires their own dataset in the future.
  + Lyfter would be required to format their database based on the formatting document that kiernan made.
  + In order for this algorithm to work more effectively we need a distance API that can handle that kind of traffic.

**Design and Implementation**

* Rob wrote the final algorithm code in the Controller class of the Byfter Project and uploaded it to github. Running with a much smaller sample size of 1\* passengers were able to successfully figure out that the average time is simply the time to get there.
* 
  + Note this says 100 because Rob did not change the system.out.println
* Kiernan wrote the MongoInterface class which takes the API key hardcoded in the controller class as a constructor which allows an interface to pull the specified collection into the program.
  + Tested with Danny’s database was able to output the location once properly parsed from Brian and Wiktors database parsing class.
* Kiernan made a database formatting document for future datasets to be able to be used with the Byfter Algorithm.
* Kiernan generated a javadoc for the Byfter project.

**Software Testing**

* Danny created an excel spreadsheet to test Google maps distances and made sure it matched with the ORSfactory distances..
* Danny constructed JUnit tests from the spreadsheet using Java JUnit framework, found in Byfter project application package.
* 
* Brian tested his DBinterface class with string convention set by Danny’s location database successfully.
* Kiernan pulled the location database from MongoDB to Json arraylist.
  + Below is the result from utilizing both the functions of DatabaseParse and MongoInterface classes.
  + 

**Evaluation**

* MVP is able to get the total time it takes for a number of drivers to run through the whole list of passengers.
  + Example, the Byfter project includes methods and an algorithm that finds the closest passenger to each driver and then splits the list into two halves, and adds up the wait times between passengers, giving a total wait time.
* Product stands to be abstract enough to allow Lyfter company to update the dataset through MongoDB using the document format information guide, their theoretical number of passengers, and the target minute metric they are trying to meet in the program in order to output the average wait time for all passengers and the number of drivers to meet this metric.
  + Product also has fields available to interface with different packages of Open Route Service. If Lyfter decides to invest, they can modify the query limit and make the product more effective without investing in another team to make a distance routing system, or changing the distance routing system from Open Route Service.