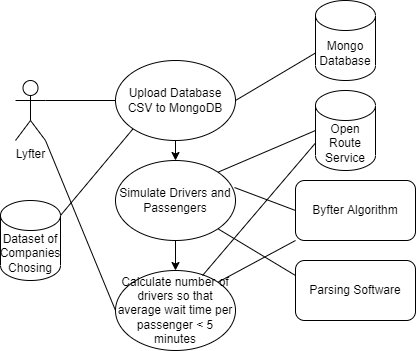
**System Modeling**

* To do : Rob will create a class diagram and use case diagram for Byfter project
* Rob created use case diagram for Byfter project



**Architectural Design**

* Database
  + Team chose MongoDB
  + Team started writing parsing software for database location records
* Brian and Wiktor decided to use java to design the parsing software.
  + Precondition to the software is that the pulling from the database results in a java string object.

**Requirements Analysis**

* 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.
* Kiernan created a requirements document to formally outline functional and not functional requirements.

**Design and Implementation**

* Rob updated Byfter code to include helper methods for testing
* Rob to implement code for input MongoDB api key.
* Rob taught Danny about Junit and helped him put together a list of 20 distances to check against google maps
* Brian and wiktor wrote code to parse out the database and uploaded it to github
  + Kiernan’s code pulls from the database and returns a string, Brian’s code parses that to create Location objects as a factory class.
* Kiernan created a monogDB and uploaded Danny’s database
* Kiernan started writing code to pull from the database.
* Rob demonstrated functionality by uploading a certain amount of locations simply to a file and then uploaded into the code manually to confirm the ORSfactory works.

**Software Testing**

* Danny created spreadsheet to test Rob’s ORSfactory class to see if the distances are accurate
* Kiernan uploaded Danny’s spreadsheet to the database, ensured that MongoDB supports uploading from CSV (exported from excel)

**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.
  + This total wait time could be divided by 100 in order to be considered the wait time per passenger. In the case that we create at least x many drivers, this will yield a sufficient answer to the question, how many drivers are required for 100 passengers in order to have an average wait time of less than 5 minutes.