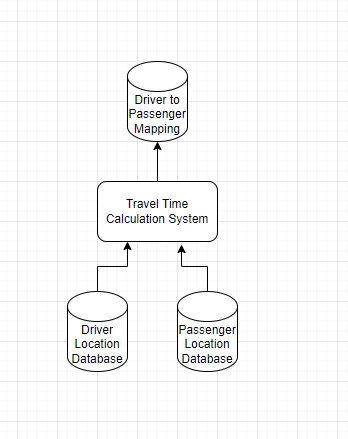
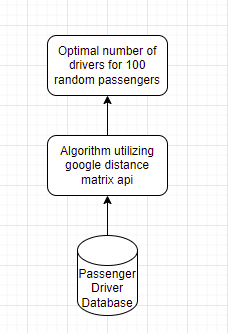
**System Modeling**

* Decided to model the system off of Rutgers New Brunswick Campus
  1. **To do**: Choose 100 random locations spread across the 4 subcampasses of Rutgers New Brunswick (Danny)
* Driver Data : their passenger driving time (in minutes, 0 if no passenger), driver start location
  1. **To do** : generate a sample set of potential driver start locations to be created (TBD)
* Passenger Data : Initial location, End location, Distance (in minutes)
* Algorithm will do the following (at the least)
  1. Start initially with 1 driver
  2. Step through the process of the driver taking the passenger (only if the wait time < 5 minutes else look for a new driver)
  3. Calculating the wait time for the next passenger to decide if the system passes the simulation ( wait time < 5 minutes)
     + Distance (minutes) = distanceFrom(driver start location, passenger start) + distanceFrom(passenger start, passenger end) + distanceFrom(passenger end, next passenger start)
     + At this point update the driver start to be passenger 1 end location to start the next process
  4. When system fails add 1 driver, and go to step 2
  5. When system passes, output number of drivers required so that all passengers have wait time < 5 minutes

**Architectural Design**

Potential Architectural Design

**v1****v2**

* Decided on using the google maps distance matrix api
  + <https://developers.google.com/maps/documentation/distance-matrix/start#maps_http_distancematrix_start-py>
  + Supports many languages but decided on using java
  + Useful API for http request and response returns json object with information that will be used for the algorithm
  + **To Do** : Test API by writing sample code for request and response from the api in java (Rob)
* Database
  + Set of locations based off of system model in google api format for 100 passenger locations
  + Set of locations based off of system model in google api format for a to be determined number of potential driver locations
  + Schema: rider (static latitude + longitude, rider ID and/or name, timestamp), driver (lat + long, driver ID and/or name, timestamp)
  + **To do** : Decide on database to use (Brian, Wiktor)
    - Firedb
    - Monodb
    - Mysql
  + 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**

* It may be required to add another passenger once a ride is completed, either through recycling this location to the “end of the queue” or otherwise adding more locations
  + Under this assumption the system has to be run for a length of time to be deemed verified.
* Identified the minimum verifiable product, a central repository for location storing with software that pulls from the database, and software to perform the relevant calculations.