Group 25 Minor Project Documentation

Software Engineering 14:332:452:01

Project Problem #2: Lyfter

Table Of Contents

Sprint 1

Sprint 2

Sprint 1

Functional Requirements

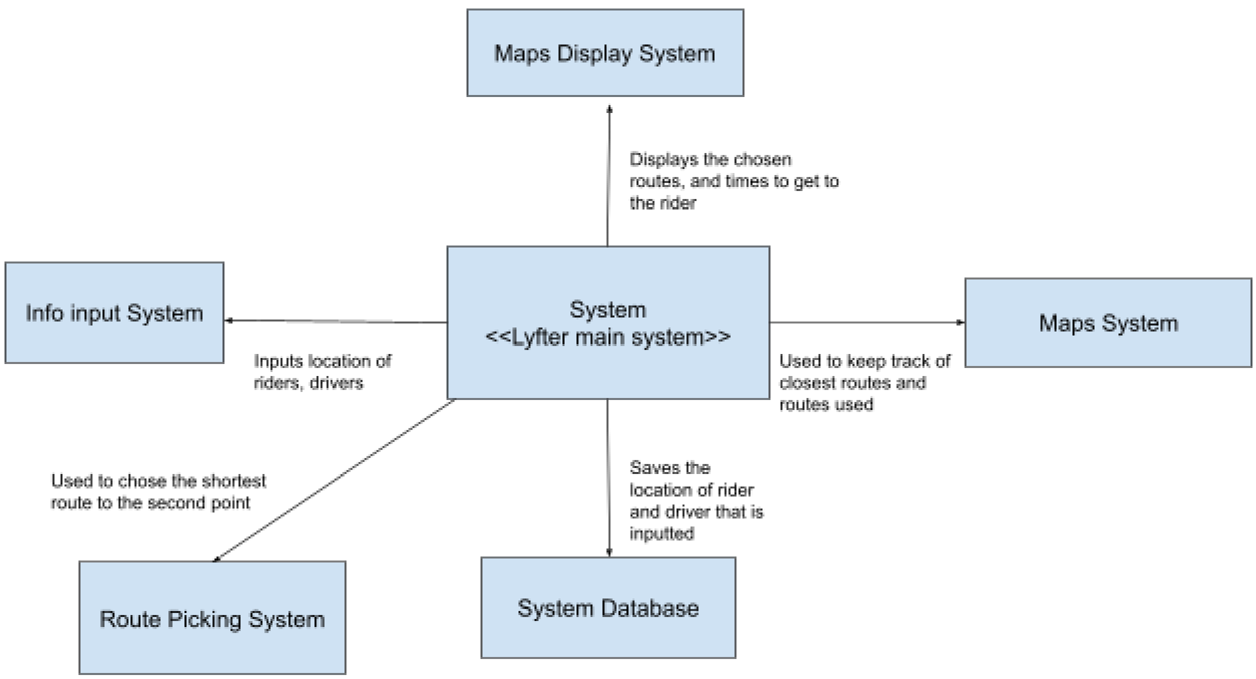
Requirement Number	Requirement Description
1	The software must be built so that it takes into account a specific place in the world that the program will be used in.
2	The software must be able to find the closest driver to a rider at all times.
3	The software should take less than 5 minutes for a driver to arrive, once a rider requests a ride.
4	The software should take into account real time traffic, and specific roads. It cannot be simply theoretical.
5	The software must be able to assign drivers to riders who are at specific locations using a csv file.
6	The software must be able to find the shortest route between 2 points.

Non-functional Requirements

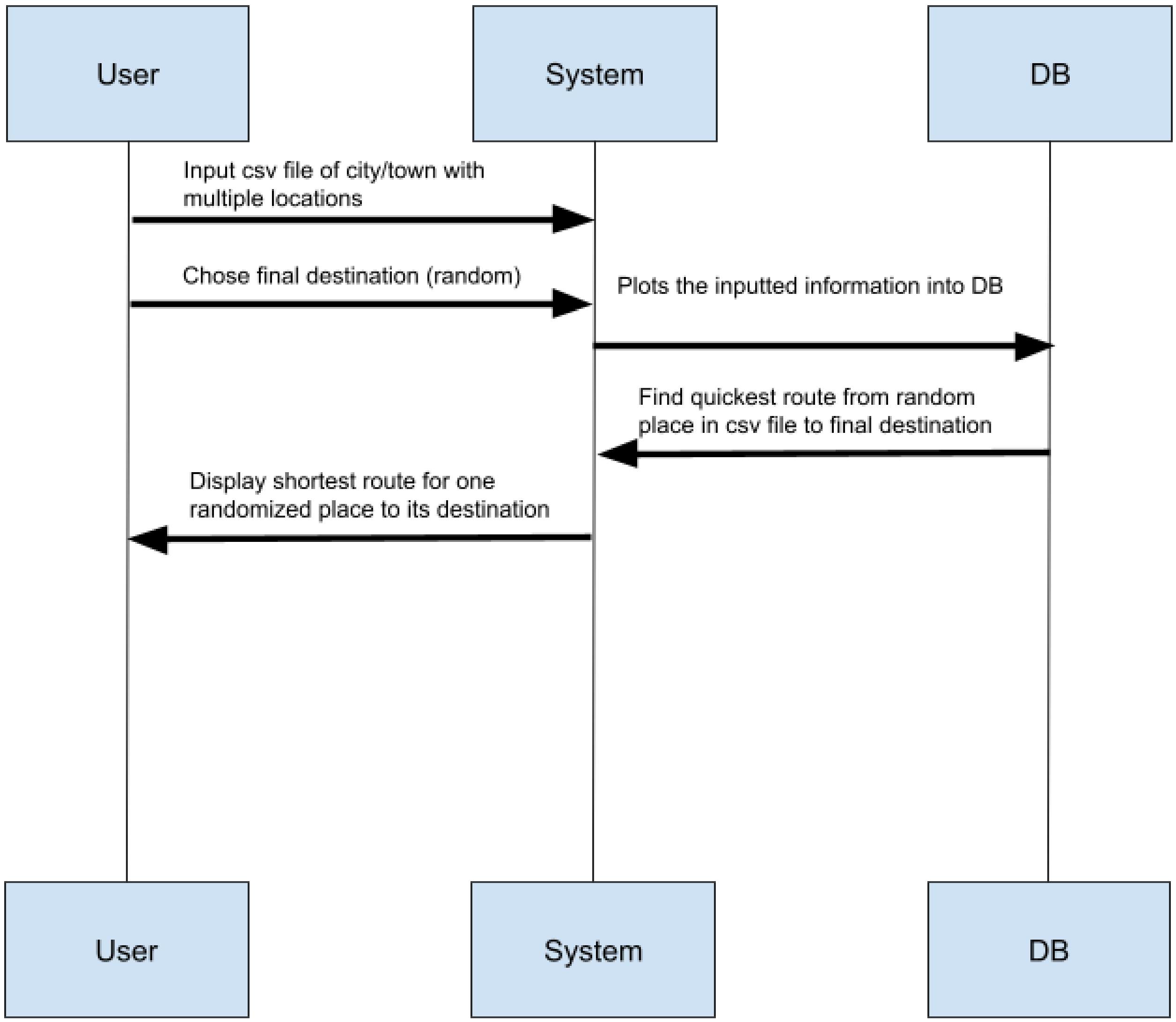
Requirement Number	Requirement Description
1	The software must be adjustable to fit any other location given specific coordinates
2	The software must be able to change modes and estimate delivery time, ex: bus vs car
3	The software must be adjustable with the number of riders at a given time.

System Modeling

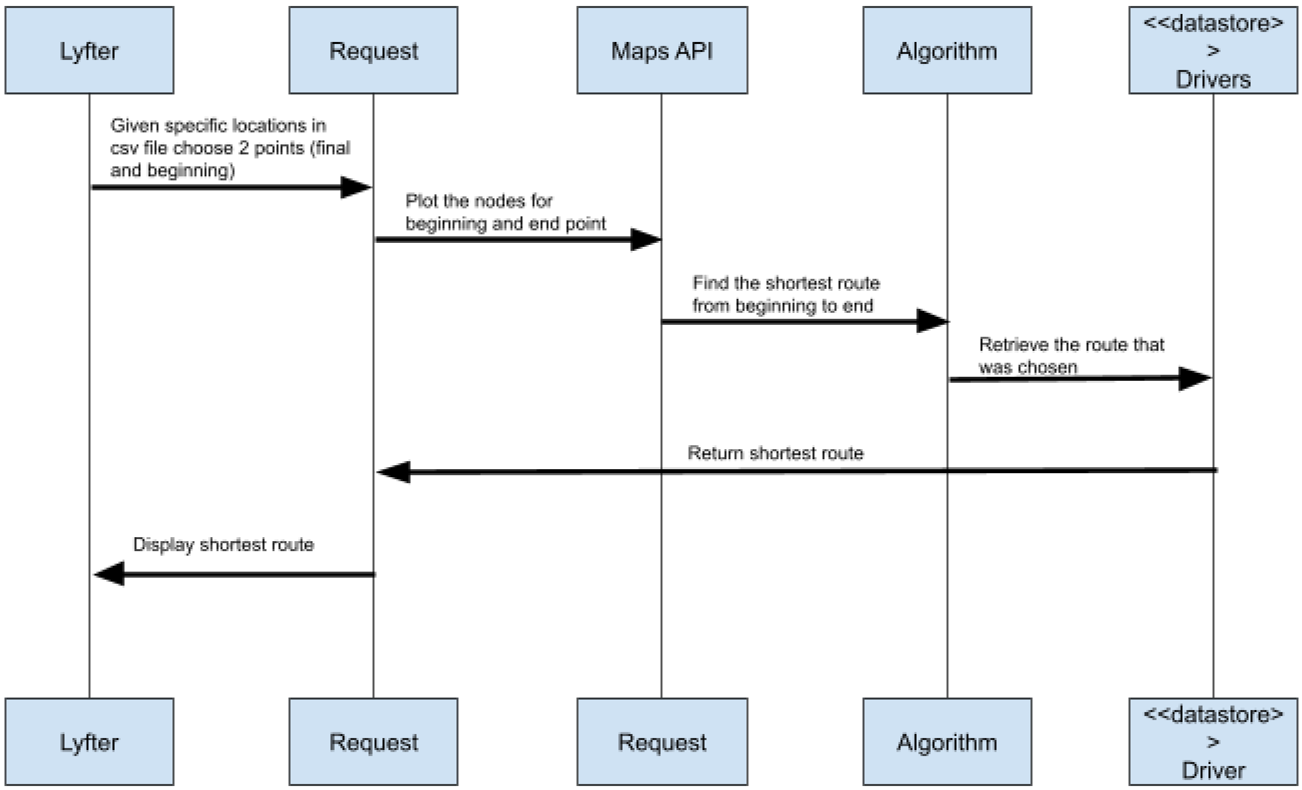
Context Model



Interaction Model



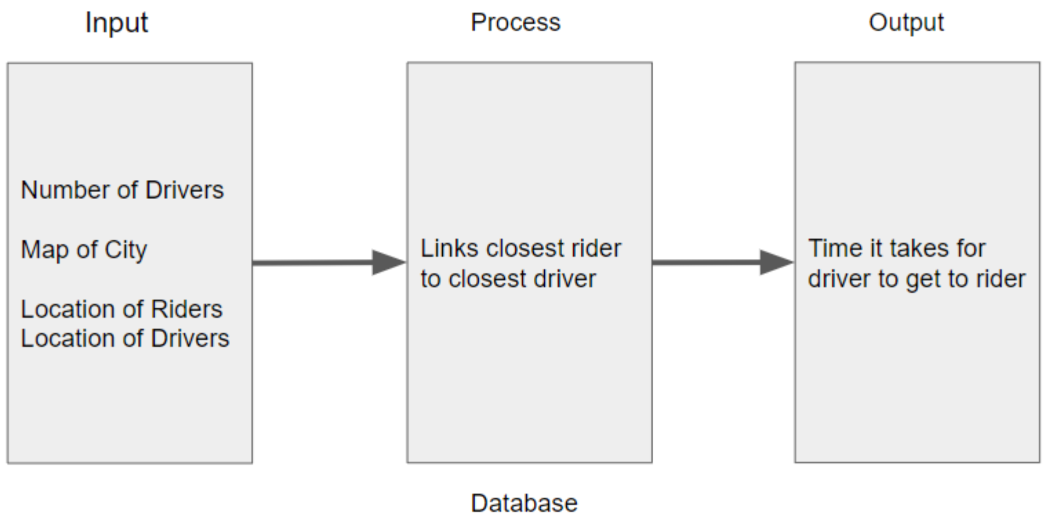
Behavioral Model



Structural Diagram

Group 25 opted to not have a class/structural diagram, since for our code we did not implement multiple classes or classes which relied upon each other for this sprint.

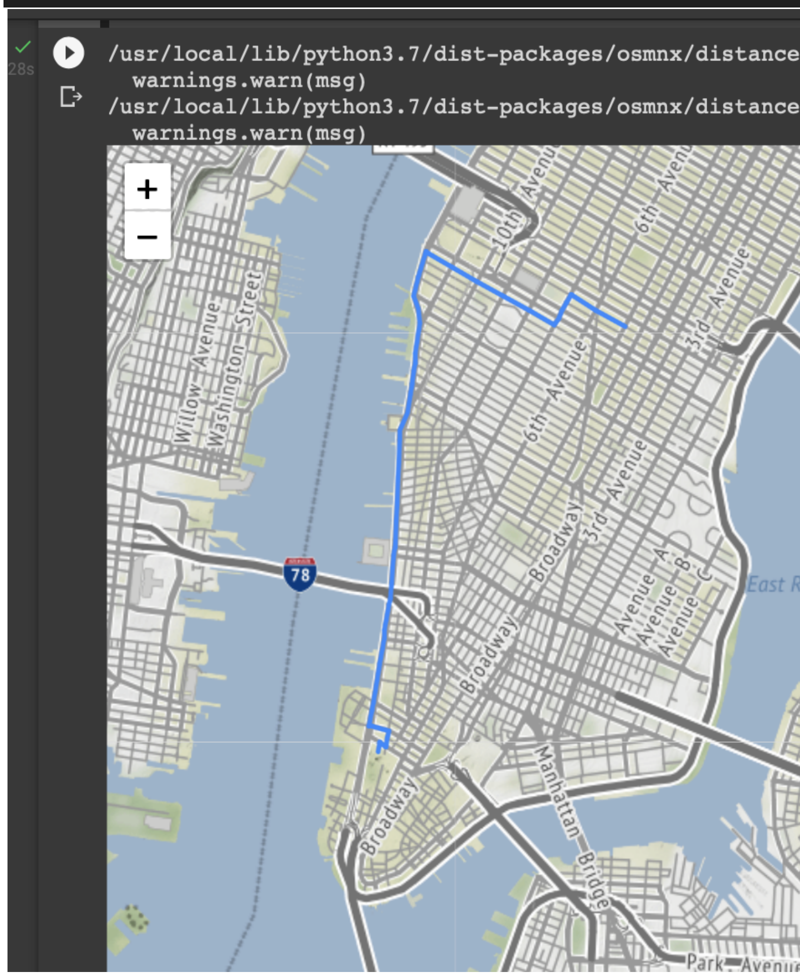
Architectural Design



Design and Implementation

```
pip install osmnx
pip install geopy
import osmnx as ox #to be able to graph
import networkx as nx #to be able to graph
import pandas as pd #to be able to read csv files
import random #to be able to choose random locations
import csv #to be able to read csv file
ox.config(log_console=True, use_cache=True)
from geopy.geocoders import Nominatim #import geo encoders to be able to
use regualr names for locations
locator = Nominatim(user_agent = "MinorProject") #set name of project

df = pd.read_csv('/nyc.csv') #read the csv file named nyc with names of
nyc locations to choose from

for i in range(2): #run loop twice
 start_location = df.sample() #choose random location from pdf for
starting
 start_latlng = locator.geocode(start_location).point #set starting
location by finding the name of location in the geopy file using the
locations regular name

 end_location = df.sample() #choose random location from pdf for ending
 end_latlng = locator.geocode(end_location).point #set ending location by
finding the name of location in the geopy file using the locations regular
name

 place     = 'Manhattan, New York City, New York, United States'
 mode      = 'drive'
 optimizer = 'time'

 graph = ox.graph_from_place(place, network_type = mode) #make graph from
chosen place (NYC) and choose form as drive
 orig_node = ox.get_nearest_node(graph, start_latlng) #find the closest
node to chosen starting location
```

```
 dest_node = ox.get_nearest_node(graph, end_latlng) #find the closest node
to chosen ending location
 shortest_route = nx.shortest_path(graph, orig_node, dest_node,
weight=optimizer) #calculate the shortest route from original and final
nodes, optimize by time

shortest_route_map = ox.plot_route_folium(graph, shortest_route,
tiles='Stamen Terrain') #form map from graph, use shortest route, and type
map is stamen
shortest_route_map #print map
```



Screenshot of output of sprint 1; the code randomly picks two locations from a csv file, in this case the empire state building and the one world trade center and finds the shortest route between both of them and outputs a graph of it.

Sprint 1: https://github.com/radical-teach/minor-project-group25/tree/main/Sprint1

Software Testing

Requirement Index	Test description	Test Case	Expected Result	Priority	P or F (pass/fail)
1	Communicate with a database for rider and driver info.	Request data	Data is retrieved	Very High	Pass
2	Communicate with API to find locations	Find locations	Locations retrieved	Very High	Pass
3	Find the shortest route to chosen locations	Find locations and manually check if shortest route	Shortest route received to destination	Very High	Pass
4	Software must run very quickly	Find shortest route between 2 locations	Shortest route received in very short time	Med	Pass

Sprint 2

Functional Requirements

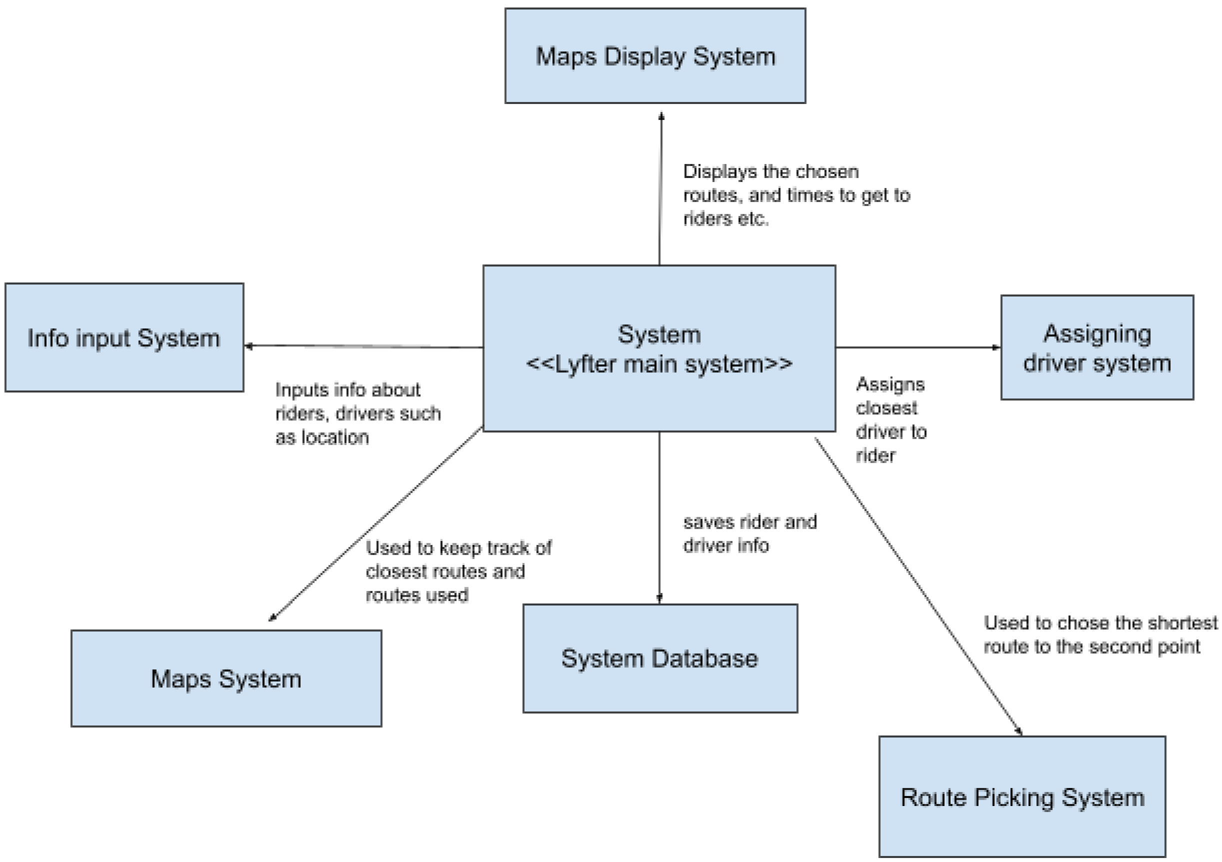
Requirement Number	Requirement Description
1	The software must be built so that it takes into account a specific place in the world that the program will be used in.
2	The software must be able to find the closest driver to a rider at all times.
3	The software should take less than 5 minutes for a driver to arrive, once a rider requests a ride.
4	The software should take into account real-time traffic, and specific roads. It cannot be simply theoretical.
5	The software created must take into account specific coordinates where to pick up the riders.
6	The software must take specific coordinates as inputs instead of simply location names

Non-functional Requirements

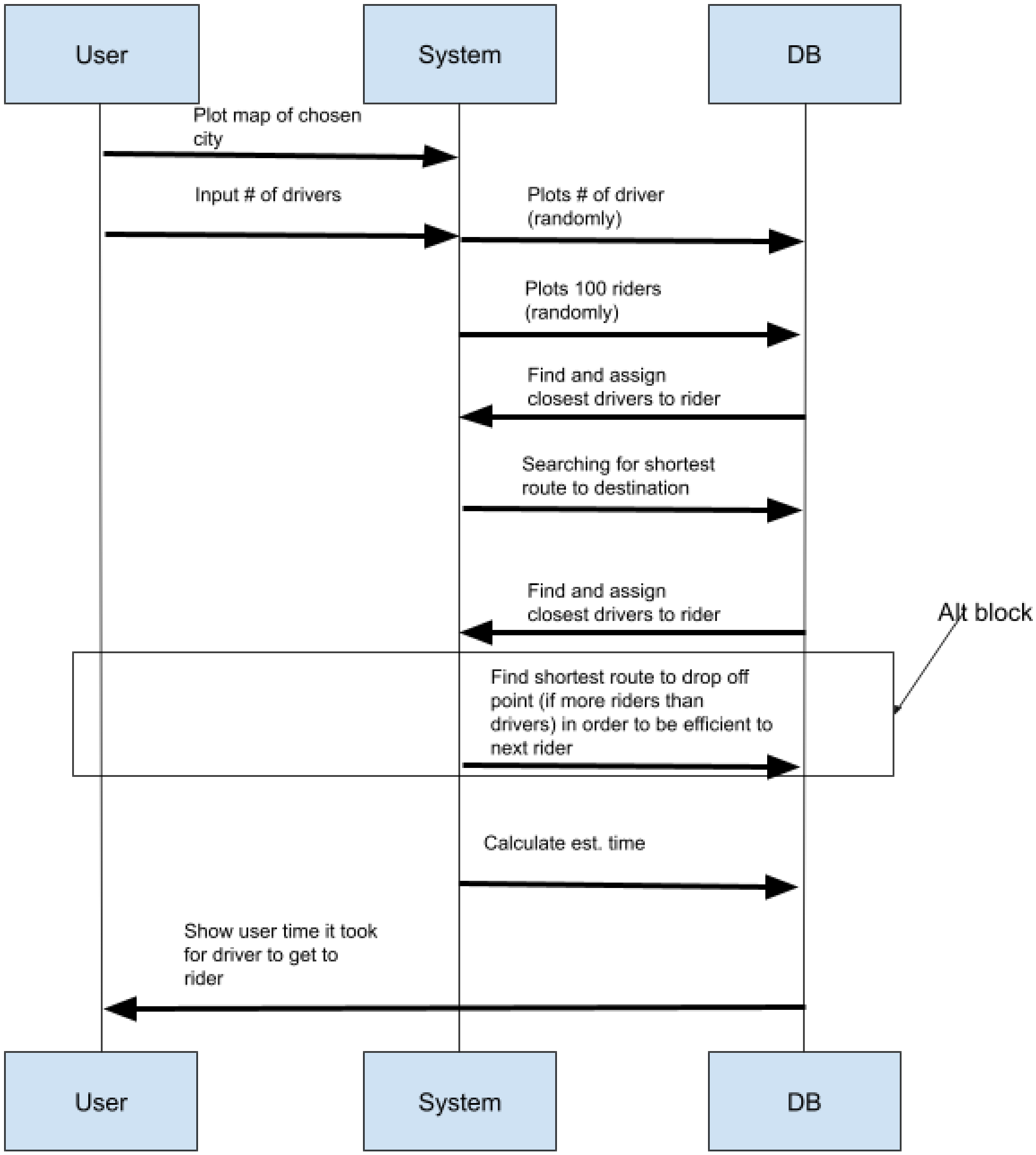
Requirement Number	Requirement Description
1	The software must be adjustable to fit any other location given specific coordinates
2	The software must be able to change modes and estimate delivery time, ex: bus vs car
3	The software written must be efficient in assigning drivers to riders, but written efficiently so that the time it takes to run for many riders must stay low.
4	The software must be adjustable with the number of drivers/riders at a given time
5	The software must be able to handle multiple drivers and riders (seemingly) at the same time, opposed to coordinating one at a time.

System Modeling

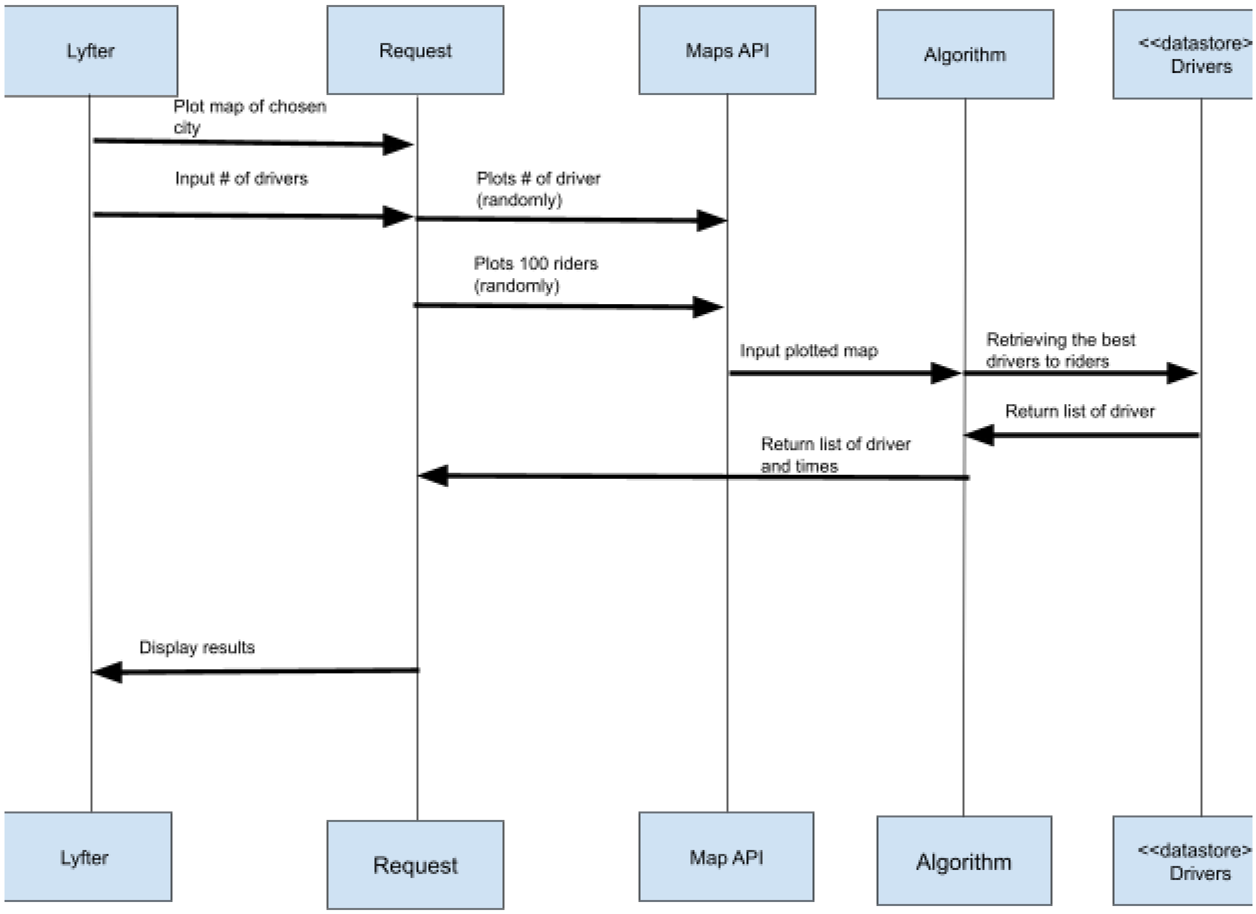
Context Model



Interaction Model



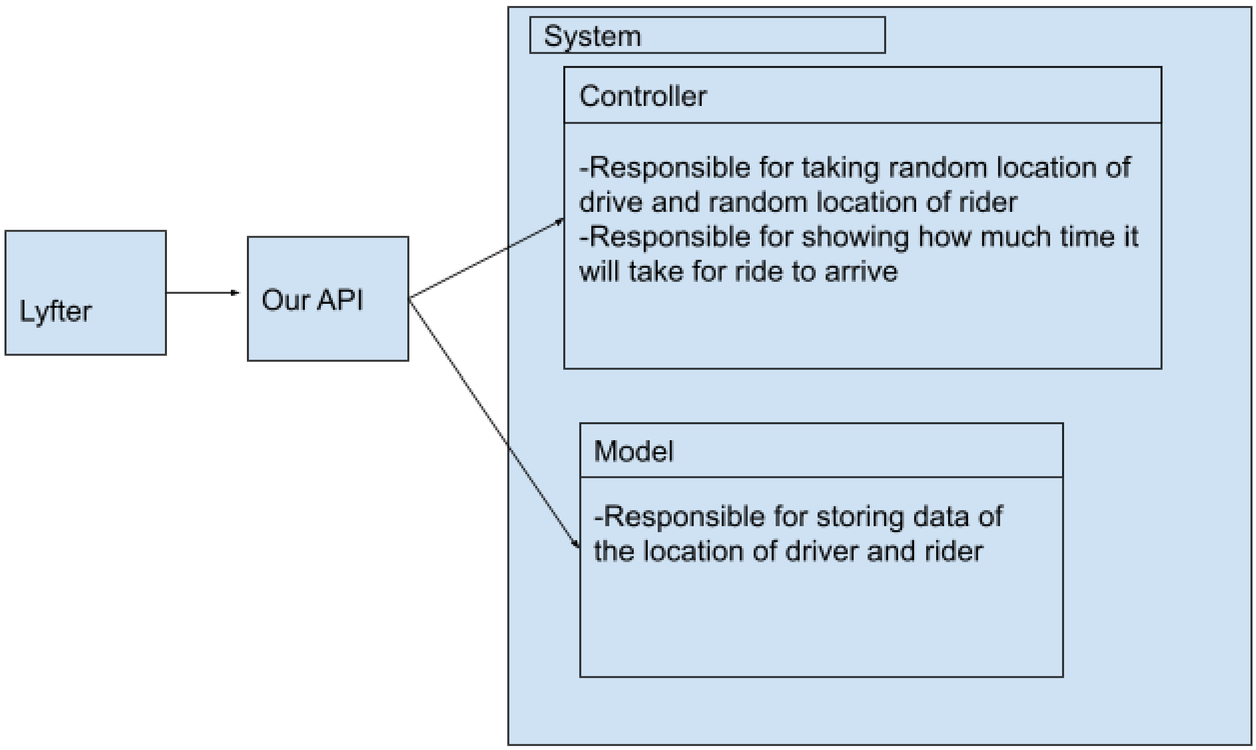
Behavioral Model



Structural Model

Group 25 opted to not have a class/structural diagram, since for our code we did not implement multiple classes or classes which relied upon each other for this sprint.

Architectural Design



Design and Implementation

Sprint 2:
https://github.com/radical-teach/minor-project-group25/blob/238038e8511dea61276f685296dfbce760ecfc3a/Sprint2/code

```
pip install osmnx
import osmnx as ox #Osmnx takes snippet from google maps, lays out a shortest distance in realworld roads
import networkx as nx #to be able to use the functions for the graph
import random #to randomize the choice of coordinates
import numpy as np #calculating the distances

ox.config(log_console=True, use_cache=True)



#place layout
place     = 'San Jose, California, United States'
mode      = 'drive' #transportation mode
optimizer = 'time' #optimization for time

# create graph from OSM within the boundaries of some geocodable place(s)
graph = ox.graph_from_place(place, network_type = mode, retain_all=False, truncate_by_edge=True, simplify=False) #graph of layout of san Jose

import warnings #to be able to work with warnings

drivers = [] #create array for drivers
riders = [] #create array for riders

#random coordinates for 10 riders using coordinates in San Jose
for i in range(100): #run loop 100 times
 x_rider = round(random.uniform(37.26864, 37.32493),5) #choose a random x from given range as lag for rider, round to 5th decmial
```

```
 y_rider = round(random.uniform(-121.91585, -121.97973),5) #choose a random y from given
range as lat for rider, round to 5th decmial

 start_latlng = (x_rider,y_rider) #set start latlang using found x and y
 riders.append(start_latlng) #to keep going to the loop and appending another coordinating thru
the loop

for i in range(150): #run loop 150 times
 x_driver = round(random.uniform(37.26864, 37.32493),5) #choose a random x from given
range as lag for driver, round to 5th decmial
 y_driver = round(random.uniform(-121.91585, -121.97973),5) #choose a random y from given
range as lat for driver, round to 5th decmial

 end_latlng = (x_driver,y_driver) #set end latlang using found x and y
 drivers.append(end_latlng) #to keep going to the loop and appending another coordinating thru
the loop

#finds closest driver to rider
for rider in riders: #iterate thru all riders
 A = np.array(drivers) #check all elements in array drivers
 B = np.array(rider) #choose element from array riders
 distances = np.linalg.norm(A-B, axis=1) #find closest driver to choosen rider
 min_index = np.argmin(distances) #set choosen driver to rider

 warnings.filterwarnings("ignore") #ignore warning given (the warning is just a diff way of
using"ox.get_nearest_node")

 Gs = ox.utils_graph.get_largest_component(graph, strongly=True) #choose "strong" main
roads

 driver_node = ox.get_nearest_node(Gs, (A[min_index])) #set driver node as the choosen index
  rider_node = ox.get_nearest_node(Gs, rider) #set the first checked rider node
  shortest_route = nx.shortest_path(Gs, driver_node, rider_node, weight=optimizer) #calculate
the shortest route from original and final nodes, optimize by time

 shortest_route_map = ox.plot_route_folium(Gs, shortest_route, tiles='Stamen Terrain') #form
map from graph, use shortest route, and type map is stamen
```

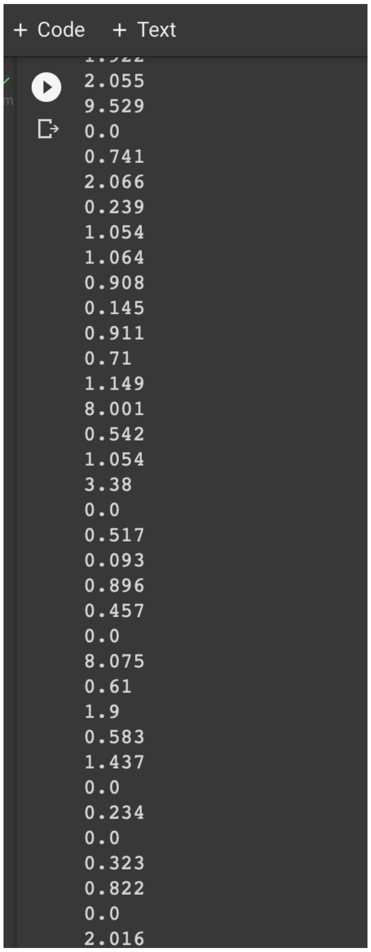
```
 length = nx.shortest_path_length(Gs, driver_node, rider_node, weight='length') #define length
in meters
 print(round(length*0.0015, 3)) #length in m, divide by speed to find time, convert to mins
shortest_route_map #print graph
```

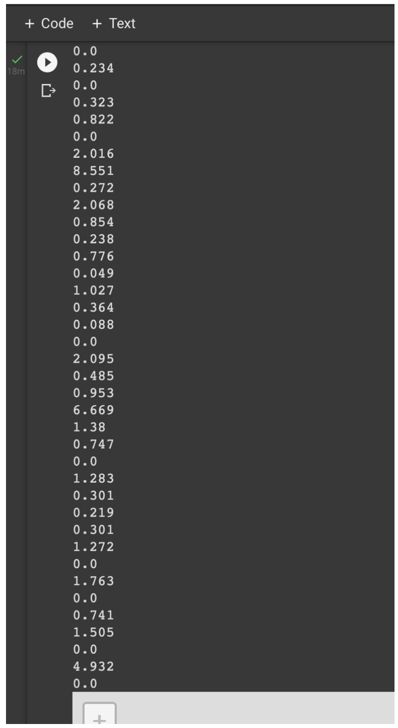
Despite the software completing the tasks we would have liked it to, it did however take some time to complete the program.

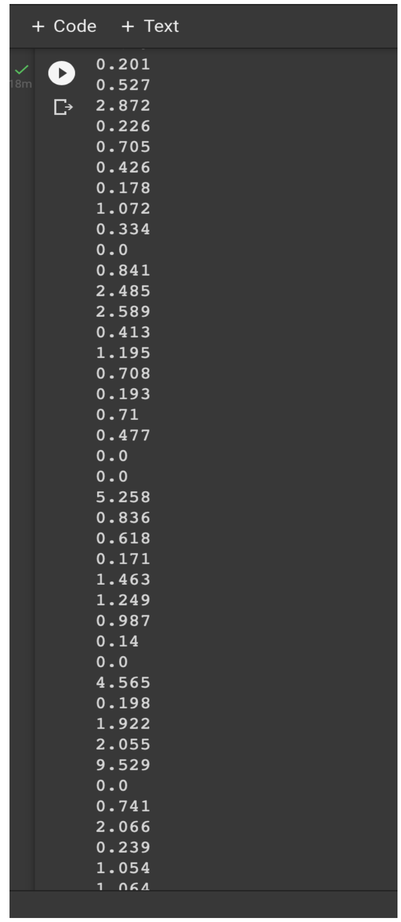
Software testing

Requirement Index	Test description	Test Case	Expected Result	Priority	P or F (pass/fail)
1	Communicate with a database for rider and driver info.	Request data	Data is retrieved	Very High	Pass
2	Communicate with API to find locations	Find locations	Locations retrieved	Very High	Pass
3	Locate closest driver to rider	Match driver and rider	Driver and rider matched	Very High	Pass
4	Application calculates time driver takes to get to rider	Driver should take less than 5 minutes	Drivers took less than 5 minutes	Very High	Pass
5	Code is able to run relatively quickly	The code takes a short time to run for all 100 riders	Expected results received within one minute	Med	Fail
6	The software must find routes for drivers to riders within 5 minutes	Run the code for all 100 riders, checking if driver to rider are all under 5 minutes	Assigned driver to every rider must be under 5 minutes	Very High	Pass
7	The software is able to handle multiple drivers at the same time	Code is able to receive multiple outputs	The times to get to riders, where there are multiple riders	High	Pass

Outputs


+ Code + Text
2.055
9.529
0.0
0.741
2.066
0.239
1.054
1.064
0.908
0.145
0.911
0.71
1.149
8.001
0.542
1.054
3.38
0.0
0.517
0.093
0.896
0.457
0.0
8.075
0.61
1.9
0.583
1.437
0.0
0.234
0.0
0.323
0.822
0.0
2.016


+ Code + Text
0.0
0.234
0.0
0.323
0.822
0.0
2.016
8.551
0.272
2.068
0.854
0.238
0.776
0.049
1.027
0.364
0.088
0.0
2.095
0.485
0.953
6.669
1.38
0.747
0.0
1.283
0.301
0.219
0.301
1.272
0.0
1.763
0.0
0.741
1.505
0.0
4.932
0.0


+ Code + Text
0.201
0.527
2.872
0.226
0.705
0.426
0.178
1.072
0.334
0.0
0.841
2.485
2.589
0.413
1.195
0.708
0.193
0.71
0.477
0.0
0.0
5.258
0.836
0.618
0.171
1.463
1.249
0.987
0.14
0.0
4.565
0.198
1.922
2.055
9.529
0.0
0.741
2.066
0.239
1.054

Evaluation

- MVP can get the total time it takes for each driver to reach its designated rider.
  - Lyfter includes an algorithm which takes the distance of the driver and the rider which is divided by the speed to find the average time. After it gives the hours it converts it to minutes and also converts kph to mph
  - Speed in the specific city is always 25 mph which is why this value is constant.
- This basic algorithm keeps the door open for multiple opportunities for using different data sets or going anywhere in the world just by changing the name of the location of the city. And the input can be changed as well such as using specific names, or using geopy to write names of location normally, or using coordinates such as what was used in sprint 2.
- Our Algorithm met our specifications which were provided in the project guidelines document and did indeed work. However our project does take some time to run, and may not be the most efficient software which we can create.
- In our first version of software, we attempted to figure out the shortest route between two points, we just at first used two coordinates and used the API using OSMNX to calculate the shortest distance, then after that was successful we used geopy to be able to uses regular location names instead of specific coordinates. To add a little more and make it choose randomly, we made a short csv file and had it pick 2 random locations from the file and find the shortest distance between them.
- In sprint 2 we needed to now make it on a bigger scale with 100 riders and much more other drivers and match up the closest driver to rider, and add as many drivers as needed to have the time be less than 5 mins between each, which the program outputs the time at the end of the loop. However since it was unrealistic for us to make a 150 location csv in a random city, we set a selection of random coordinates but within a specific location range.

- Additionally we also implemented some of the non-functional requirements as we believed that this could make the program more feasible, the software being scalable or there being different transportation modes if for example lyfter might offer a bus service, or rickshaws, motorbikes etc.