Software Engineering

Major Project

Project Problem #1: OptimalDelivery: OptimalDelivery is a logistics company that wants to move shipments from Producer to Consumer in a distance-optimal manner. They approach your team to help them design a route planning system that enables their fleet of T trucks to pick up from P (predetermined) pickup points and deliver to D receiving points, so as to incur minimum gas costs, i.e., the fleet has to drive minimum distance collectively.

By Group 33: Devan Patel, Mark Rezk and David Banyamin

Sprint 1

Requirement Engineering:

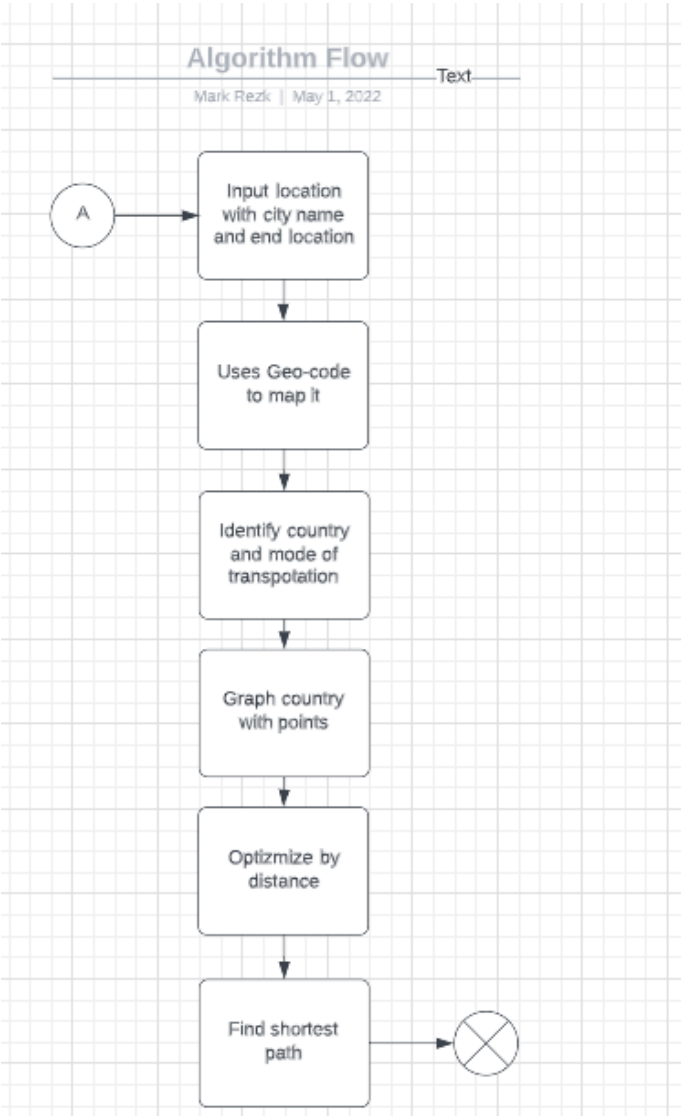
Requirement ID:	Requirement:	Reasoning:
R1	System needs to be able to link the shortest and strongest path together	We need to find the shortest distance between the two points
R2	System needs to be able to identify chosen country and mode of transportation	To be able to map out chosen location and choose mode of transportation for trucks (Which is driving)
R3	Systems reads location of pickup station and trucks	To be able to map out the points
R4	System uses Geo-code to map	To be able to plot map
R5	System needs to know the location of destination points	To be able to find the shortest path
R6	System needs to be able to get the shortest distance for a truck to go to pick up station and destination	To find the shortest path
R7	Calculate distance the truck traveled	To check and see to minimize gas costs

Non-Functional Requirements

Requirement ID:	Requirement:	Reasoning:
R1	You will need to connect to the internet to use this application	Need to be able to access google api
R2	Should be able to work at least 95% of the time	So that it is reasonably reliable
R3	Route choosing algorithm must be accurate in choosing path to destination	This is so that the software which was engineered is reliable and accurate

System Modeling:

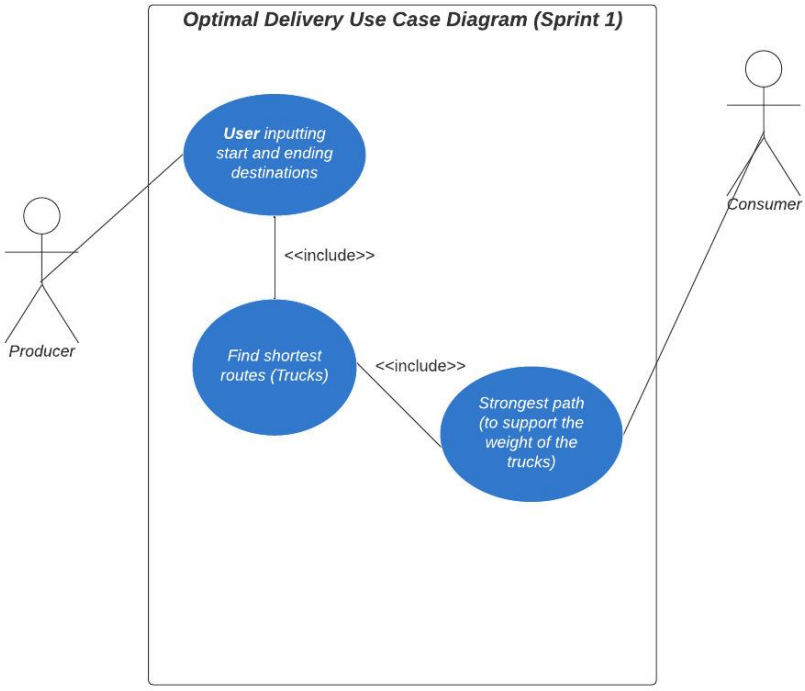
Activity Diagram:



Use Case Model:

Use case diagram

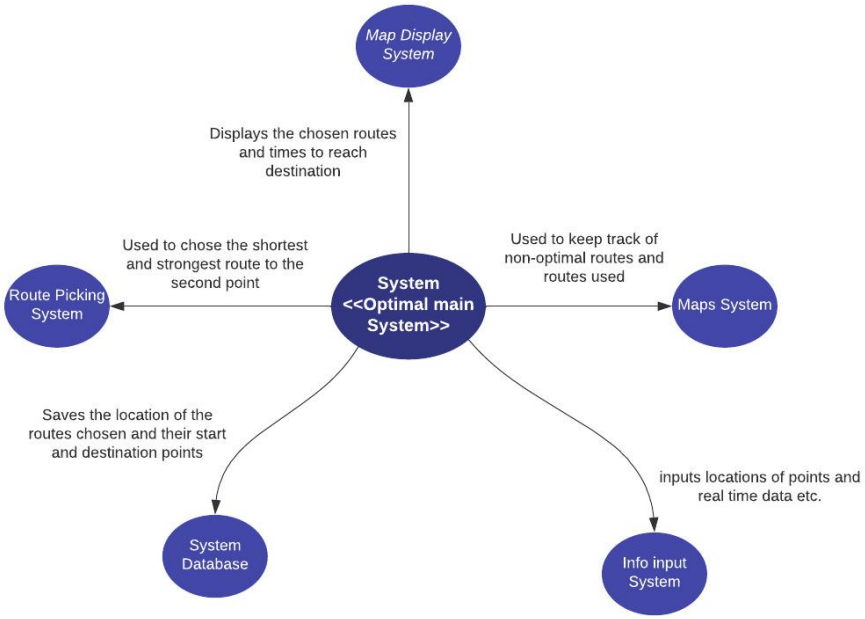
Devan | April 30, 2022



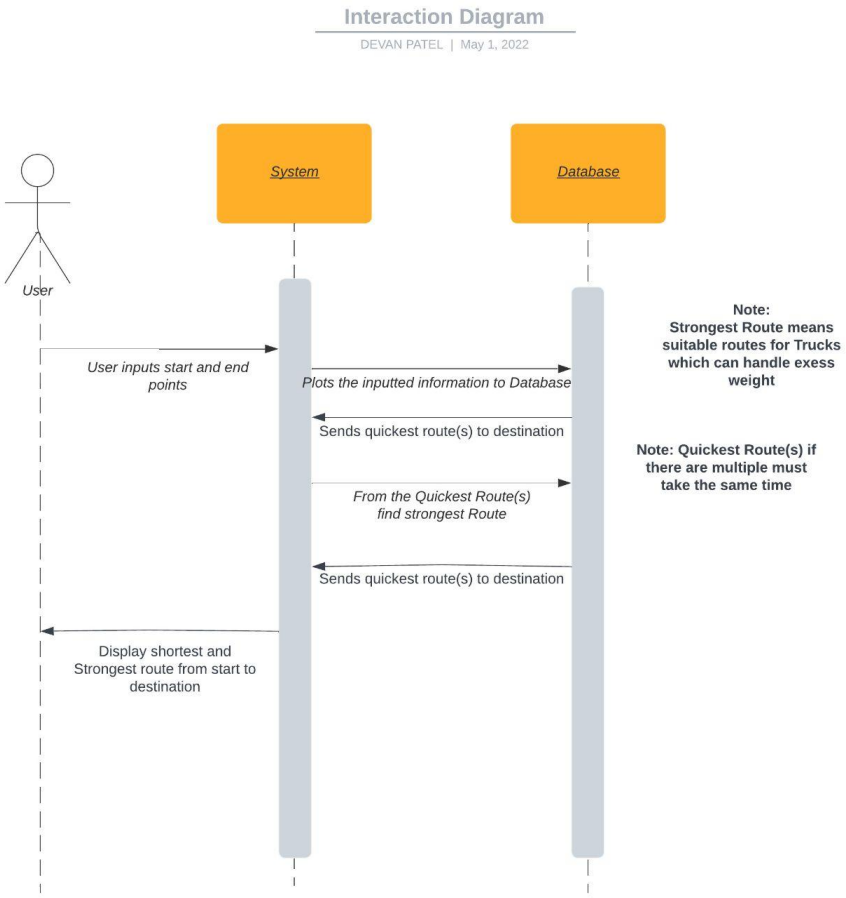
Context Model:

Spider concept map

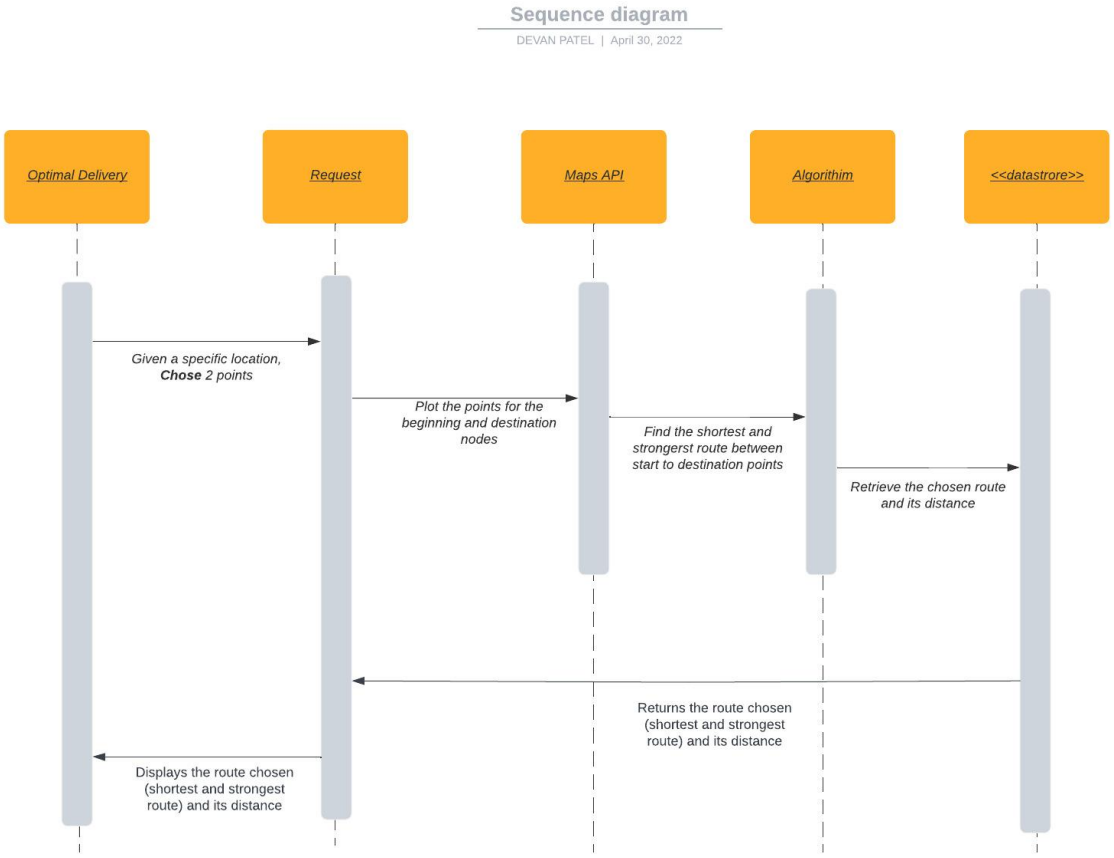
DEVAN PATEL | April 28, 2022



Interaction Diagram:



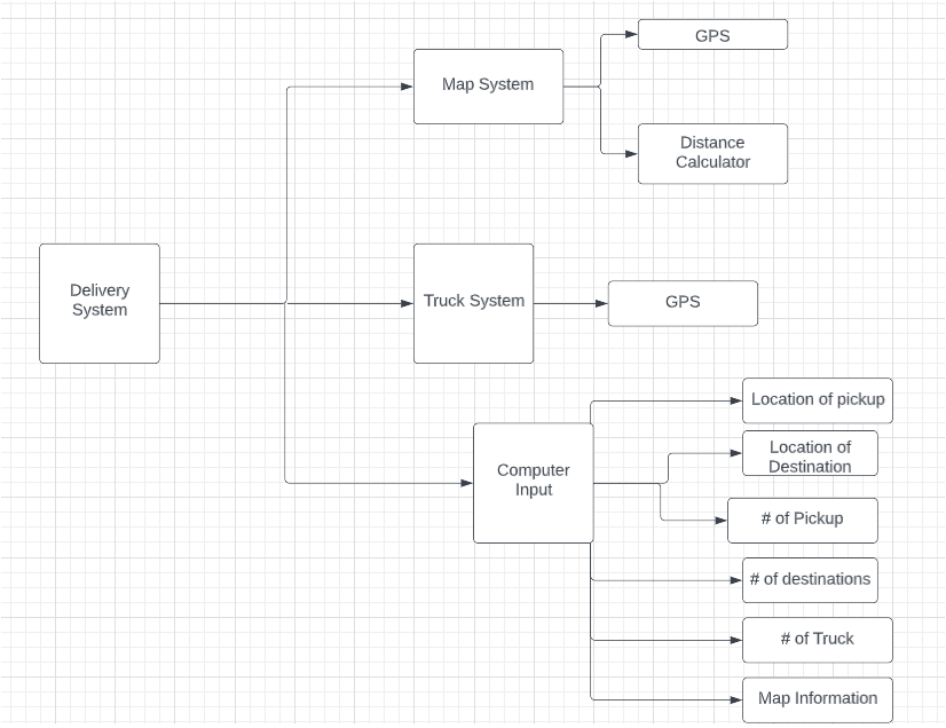
Behavioral Model:



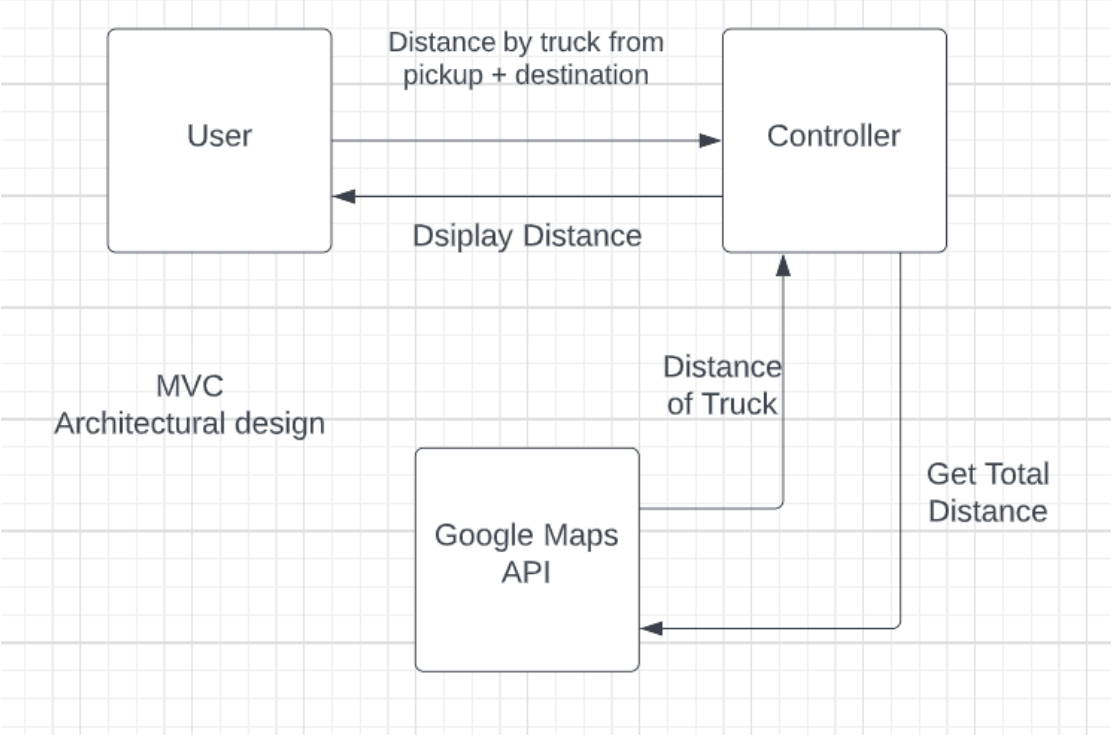
Class Diagram:

Group 33 opted not to have a Class/Structural Diagram since our software which we engineered did not have more than one class nor any classes which were reliant on each other.

Architectural Design:



(Multiple Architectural views)



(User is User Interface)

Specifications:

Function	Computes total distance in meters of a single route traveled.
Description	Computes the total distance traveled between two cities by a truck that the user chooses in a given place.
Inputs	Names of Two cities.
Source	Cities that the truck will be going from and to, to deliver a package.
Outputs	The total distance that will be traveled in meters.
Destination	Google API
Action	The program will take the name of the starting location city and geocode it to get its location on OSMNX using the google api then does the same for the end location and using networkx it finds the shortest route using real streets while optimizing distance over time.
Requires	Two cities in the same vicinity or place chosen.
Precondition	Those two places can be traveled to by drive method.
Postcondition	None
Side Effects	None

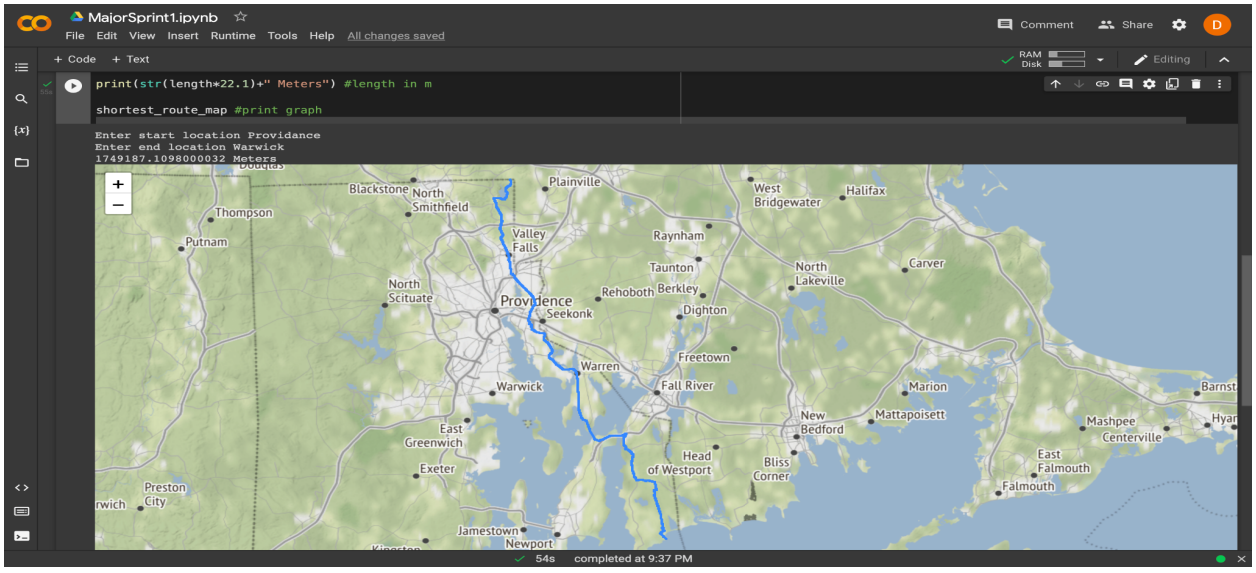
System Design and Implementation:

1. The code uses and imports osmnx to take snippets from google maps and lays out the shortest route in the real world on real roads. It also uses and imports networkx to be able to calculate the length between that found shortest distance.
2. Here as well we imported warnings because it sent out a useless warning about a different syntax that was unnecessary and we just wanted to get rid of it to clean up the output.
3. The code then asks the user for the start location and end location of one single route.
4. It uses the transportation mode as drive (other options as walk, bike, train, etc.), and it is optimized by length as opposed to time.
5. Then the code uses osmnx to graph a map of New Jersey this time using the variables stated before.
6. The start and end location is the two cities the user chooses in New jersey.
7. The code is able to use normal city names because of geopy, another package that needs to be downloaded which geocodes these regular names to preset locations that it has.
8. Then the start node and end node are found and using networkx the shortest path is mapped out, and then length is found and printed. And in the output it shows all of the packages loaded and the total distance traveled.

Software Testing

Test 1: Providence to Warwick

Reason: Tested from Providence to Warwick as an example of using major cities in an area such as Rhode Island, given that these two are the largest cities there. The distance is 1749187 Meters.

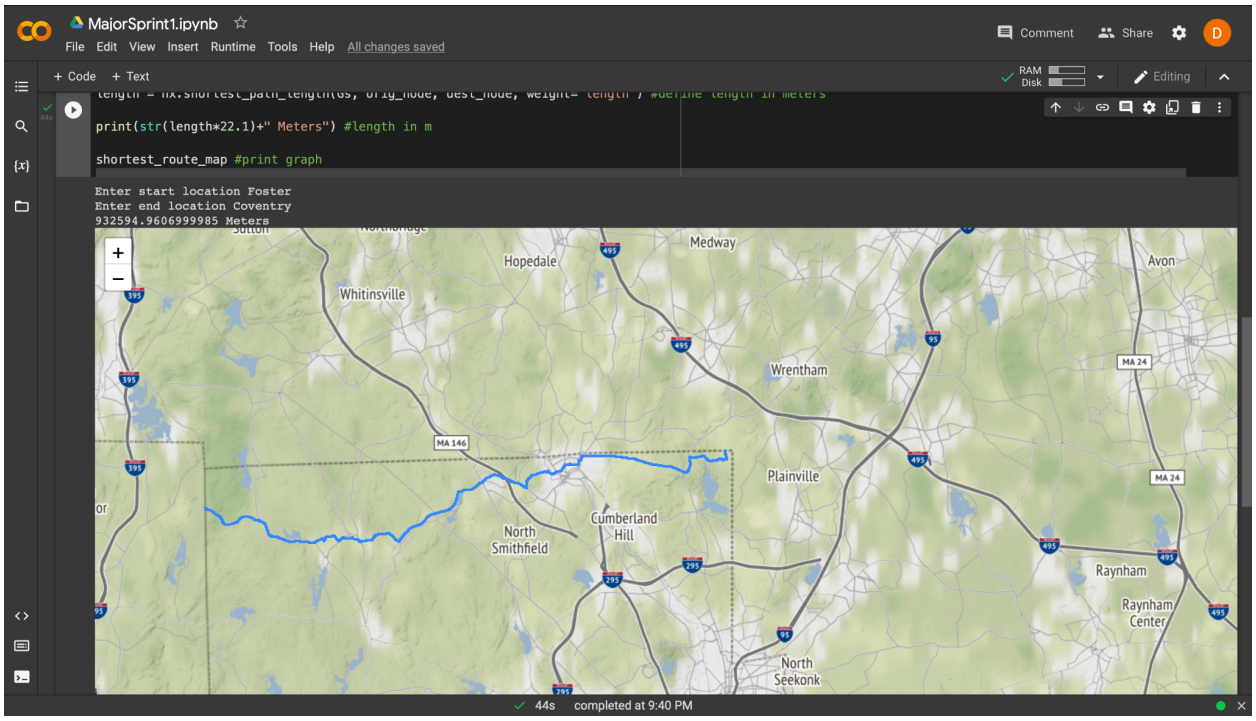


(If tested in console, rather than Google Colab then user will receive the total distance their chosen path will take, rather than the entire map)

Test 2: Foster to Coventry

```
2022-05-02 21:47:24 Removed 10358 nodes outside polygon
2022-05-02 21:47:32 Removed 0 isolated nodes
2022-05-02 21:47:48 Got largest weakly connected component (380390 of 386930 total nodes)
2022-05-02 21:47:49 Truncated graph by polygon
2022-05-02 21:47:57 Counted undirected street segments incident on each node
2022-05-02 21:47:57 graph_from_polygon returned graph with 380390 nodes and 745418 edges
2022-05-02 21:47:58 graph_from_place returned graph with 380390 nodes and 745418 edges
Enter start location: Providence
Enter end location: Warwick
2022-05-02 21:50:37 Created nodes GeoDataFrame from graph
2022-05-02 21:50:40 Created nodes GeoDataFrame from graph
2022-05-02 21:51:00 Got largest strongly connected component (379432 of 380390 total nodes)
2022-05-02 21:51:01 Created edges GeoDataFrame from graph
25305.166999999972 Meters
(base) Davids-MacBook-Pro:~ dbanyamin$ python3 sprint1.py
2022-05-02 21:51:32 Configured OSMnx 1.1.2
2022-05-02 21:51:32 HTTP response caching is on
2022-05-02 21:51:32 Retrieved response from cache file "cache/67c17f0a3d881f2f814b0eba48e9437f0a81208c.json"
/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages/osmnx/geocoder.py:110: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  gdf = gdf.append(_geocode_query_to_gdf(q, wr, by_osmid))
2022-05-02 21:51:33 Created GeoDataFrame with 1 rows from 1 queries
2022-05-02 21:51:33 Constructed place geometry polygon(s) to query API
2022-05-02 21:51:33 Projected GeoDataFrame to +proj=utm +zone=19 +ellps=WGS84 +datum=WGS84 +units=m +no_defs +type=crs
2022-05-02 21:51:33 Projected GeoDataFrame to epsg:4326
2022-05-02 21:51:33 Projected GeoDataFrame to +proj=utm +zone=19 +ellps=WGS84 +datum=WGS84 +units=m +no_defs +type=crs
2022-05-02 21:51:33 Projected GeoDataFrame to epsg:4326
2022-05-02 21:51:33 Requesting data within polygon from API in 6 request(s)
2022-05-02 21:51:33 Retrieved response from cache file "cache/a86cf54bbc4bf145847be85592d76bd2b6f337c1.json"
2022-05-02 21:51:33 Retrieved response from cache file "cache/beee6ed7e964ae6a18dc70e889d3403ace942d37.json"
2022-05-02 21:51:33 Retrieved response from cache file "cache/e58740f71ca9e35bfb33ee8529dee06e339b5c66.json"
2022-05-02 21:51:33 Retrieved response from cache file "cache/dec6c98ca49b88ac5a49a99f5b90b1cf81266.json"
2022-05-02 21:51:34 Retrieved response from cache file "cache/8cd1a77490e28a912ba9d8686fcec1d63338a8cb.json"
2022-05-02 21:51:34 Retrieved response from cache file "cache/5ad3e71c2caf656a8910c174fb81705be7d9b379.json"
2022-05-02 21:51:34 Got all network data within polygon from API in 6 request(s)
2022-05-02 21:51:34 Creating graph from downloaded OSM data...
2022-05-02 21:51:42 Created graph with 451129 nodes and 882561 edges
2022-05-02 21:51:47 Added length attributes to graph edges
2022-05-02 21:51:47 Identifying all nodes that lie outside the polygon...
2022-05-02 21:52:04 Created nodes GeoDataFrame from graph
2022-05-02 21:52:18 Created r-tree spatial index for 451129 geometries
2022-05-02 21:52:22 Identified 396967 geometries inside polygon
2022-05-02 21:52:30 Removed 53841 nodes outside polygon
2022-05-02 21:52:30 Truncated graph by polygon
2022-05-02 21:52:32 Identifying all nodes that lie outside the polygon...
2022-05-02 21:52:50 Created nodes GeoDataFrame from graph
2022-05-02 21:53:00 Created r-tree spatial index for 397288 geometries
2022-05-02 21:53:03 Identified 386700 geometries inside polygon
2022-05-02 21:53:09 Removed 10358 nodes outside polygon
2022-05-02 21:53:17 Removed 0 isolated nodes
2022-05-02 21:53:31 Got largest weakly connected component (380390 of 386930 total nodes)
2022-05-02 21:53:32 Truncated graph by polygon
2022-05-02 21:53:36 Counted undirected street segments incident on each node
2022-05-02 21:53:37 graph_from_polygon returned graph with 380390 nodes and 745418 edges
2022-05-02 21:53:37 graph_from_place returned graph with 380390 nodes and 745418 edges
Enter start location: Foster
Enter end location: Coventry
2022-05-02 21:55:02 Created nodes GeoDataFrame from graph
2022-05-02 21:55:04 Created nodes GeoDataFrame from graph
2022-05-02 21:55:24 Got largest strongly connected component (379432 of 380390 total nodes)
2022-05-02 21:55:26 Created edges GeoDataFrame from graph
42198.86699999993 Meters
(base) Davids-MacBook-Pro:~ dbanyamin$
```

Reason: Tested from Foster to Coventry to see if less major cities would be able to read by the program or if that would cause any issues and not be read however it was fine and the distance was 932594 Meters.

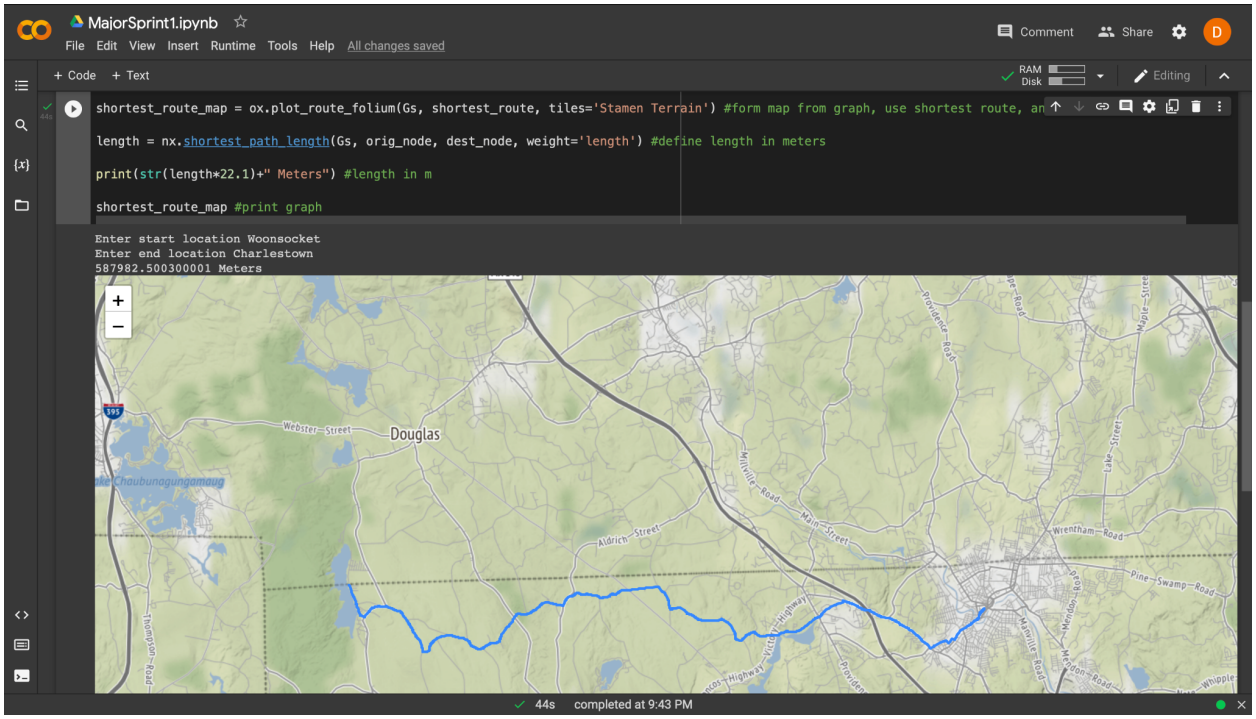


(If tested in console, rather than Google Colab then user will receive the total distance their chosen path will take, rather than the entire map)

Test 3: Woonsocket to Charlestown

```
2022-05-02 21:53:09 Removed 10358 nodes outside polygon
2022-05-02 21:53:17 Removed 0 isolated nodes
2022-05-02 21:53:31 Got largest weakly connected component (380390 of 386930 total nodes)
2022-05-02 21:53:32 Truncated graph by polygon
2022-05-02 21:53:36 Counted undirected street segments incident on each node
2022-05-02 21:53:37 graph_from_polygon returned graph with 380390 nodes and 745418 edges
2022-05-02 21:53:37 graph_from_place returned graph with 380390 nodes and 745418 edges
Enter start location: Foster
Enter end location: Coventry
2022-05-02 21:55:02 Created nodes GeoDataFrame from graph
2022-05-02 21:55:04 Created nodes GeoDataFrame from graph
2022-05-02 21:55:24 Got largest strongly connected component (379432 of 380390 total nodes)
2022-05-02 21:55:26 Created edges GeoDataFrame from graph
42198.866999999993 Meters
(base) Davids-MacBook-Pro:~ dbanyamin$ python3 sprint1.py
2022-05-02 21:57:35 Configured OSMnx 1.1.2
2022-05-02 21:57:35 HTTP response caching is on
2022-05-02 21:57:35 Retrieved response from cache file "cache/67c17f0a3d881f2f814b0eba48e9437f0a81208c.json"
/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages/osmnx/geocoder.py:110: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  gdf = gdf.append(_geocode_query_to_gdf(q, wr, by_osmid))
2022-05-02 21:57:35 Created GeoDataFrame with 1 rows from 1 queries
2022-05-02 21:57:36 Constructed place geometry polygon(s) to query API
2022-05-02 21:57:36 Projected GeoDataFrame to +proj=utm +zone=19 +ellps=WGS84 +datum=WGS84 +units=m +no_defs +type=crs
2022-05-02 21:57:36 Projected GeoDataFrame to epsg:4326
2022-05-02 21:57:36 Projected GeoDataFrame to +proj=utm +zone=19 +ellps=WGS84 +datum=WGS84 +units=m +no_defs +type=crs
2022-05-02 21:57:36 Projected GeoDataFrame to epsg:4326
2022-05-02 21:57:36 Requesting data within polygon from API in 6 request(s)
2022-05-02 21:57:36 Retrieved response from cache file "cache/e86cf540bc6bf14584 7be85692d76bd2b6f337c1.json"
2022-05-02 21:57:36 Retrieved response from cache file "cache/bee4ed7e944ae4a180dc70e089d3403acbe942d37.json"
2022-05-02 21:57:36 Retrieved response from cache file "cache/e58748f71ca9e3bfb33ee8529dee80a339b5c66.json"
2022-05-02 21:57:36 Retrieved response from cache file "cache/dec6c98ca698488ac5da49ea9bf569bb1cf81266.json"
2022-05-02 21:57:37 Retrieved response from cache file "cache/8cd1a774980c28a912ba9d8686fcec1d63336a8cb.json"
2022-05-02 21:57:37 Retrieved response from cache file "cache/5ad3e71c2caf556a8910c174fb81705be7d9b379.json"
2022-05-02 21:57:37 Got all network data within polygon from API in 6 request(s)
2022-05-02 21:57:37 Creating graph from downloaded OSM data...
2022-05-02 21:57:46 Created graph with 451129 nodes and 882561 edges
2022-05-02 21:57:51 Added length attributes to graph edges
2022-05-02 21:57:51 Identifying all nodes that lie outside the polygon...
2022-05-02 21:58:09 Created nodes GeoDataFrame from graph
2022-05-02 21:58:24 Created r-tree spatial index for 451129 geometries
2022-05-02 21:58:27 Identified 396967 geometries inside polygon
2022-05-02 21:58:35 Removed 63841 nodes outside polygon
2022-05-02 21:58:35 Truncated graph by polygon
2022-05-02 21:58:37 Identifying all nodes that lie outside the polygon...
2022-05-02 21:58:55 Created nodes GeoDataFrame from graph
2022-05-02 21:59:06 Created r-tree spatial index for 397288 geometries
2022-05-02 21:59:09 Identified 386700 geometries inside polygon
2022-05-02 21:59:16 Removed 10358 nodes outside polygon
2022-05-02 21:59:24 Removed 0 isolated nodes
2022-05-02 21:59:40 Got largest weakly connected component (380390 of 386930 total nodes)
2022-05-02 21:59:41 Truncated graph by polygon
2022-05-02 21:59:46 Counted undirected street segments incident on each node
2022-05-02 21:59:46 graph_from_polygon returned graph with 380390 nodes and 745418 edges
2022-05-02 21:59:47 graph_from_place returned graph with 380390 nodes and 745418 edges
Enter start location: Woonsocket
Enter end location: Charlestown
2022-05-02 22:01:48 Created nodes GeoDataFrame from graph
2022-05-02 22:01:50 Created nodes GeoDataFrame from graph
2022-05-02 22:02:09 Got largest strongly connected component (379432 of 380390 total nodes)
2022-05-02 22:02:10 Created edges GeoDataFrame from graph
26605.543000000045 Meters
(base) Davids-MacBook-Pro:~ dbanyamin$
```

Reason: Tested Woonsocket to Charlestown to see how far the edge of the program extends to or if being by the ocean causes any issues to the program but it does not and the distance was 587928 Meters.



(If tested in console, rather than Google Colab then user will receive the total distance their chosen path will take, rather than the entire map)

Evaluation

Regarding our testing we tested scenarios where the pick up and drop of points were in large or small cities as well as near the ocean to see how it would affect our route finding system. We were able to tell that our system would work in many different scenarios and conditions. For this sprint we wanted to see how far the boundaries of the API would reach and how accurate it would be. To do this we first tested major cities to check if it does in fact work and we verified the distance that we received by checking the best routes between certain cities on other mapping services such as Apple maps or Google maps. Then we also tried to make sure that the program does not exclude smaller cities, so we focused our second test on small towns in Rhode Island and it was able to read them as well, and after we got the output for the distance we once again verified this on other mapping services. For the third test in sprint 1 we tried to see boundaries and how cities near water would affect the calculations, so we took cities by the water boundary of Rhode Island and a city near the border of Rhode Island and it was once again able to successfully calculate the distance once again. For this sprint we met the requirements by making sure the system linked the shortest and strongest paths using chosen area and mode of transportation, use geocode for the starting and end location inputted by the user and able to plot out the map and display the total distance traveled by the truck.

Dependable Attribute:

- **System Security:**
    - We did not create our own maps API since this would have taken a large amount of time to develop as well as resources. Additionally if we had developed our own API, then it would easily be prone to external attacks. However we ended up using a Google Maps API which helps keep the system secure, since Google is very well versed with security protocols for their applications/sites. Because we used Google Maps API, we can keep our system secure as well as the location information safe because it is stored with Google.

Project Management Concept: Post-Facto Risk-Management

Description	Impact	Probability	Severity	Mitigation Strategy
The team would not have enough money to be able to afford resources such as a mapping program to map out entire world ourselves	Without this the project would really not be able to be started and would really just be an imagination	Low	Very Severe	To mitigate this we decided to use Google Maps API which was free and very secure making it so that the team has enough resources on this project.
Team members may not be very experienced for certain portions of the project	Team members would have to become more competent before beginning tasks they were assigned	Moderate	Severe	To mitigate this, the group leader heard out what each member is confident in doing and assigned tasks based on this, and was done very early so that this could be completed in advance so team members would have time to prepare
The software which was engineers has many bugs or does not run properly	This would make it so that the team would have to work harder to do extra work or to move back the deadlines causing delays	High	Very Severe	To mitigate this, every week all group members took time to look at and to test the code which was implemented to see if it accomplished the goals we set forth

				successfully
Tasks may not have been completed by team members for this sprint on time	This would mean that portions of the project would not be completed in time, deadlines may have to be pushed, group members will have to work much harder to try to finish the project in time	Moderate	Severe	To Mitigate this, the group leader would continue to check up on the work done by everyone weekly to ensure the project will be completed in time.
The Group Leader spreads out the time unevenly or the team uses too much time on one portion and makes an incorrect call on the time they have left in the project	This would mean that the team would later have to rush on other portions which could lead to a lack of quality on important parts, or could cause delays	Moderate	Somewhat Severe	This can easily be mitigated by setting periods of time for group members who are tasked with portions and checking in during those periods of time in order to ensure things are going smoothly. If not the group leader must change how things are working in order to get the team back on track

Advanced SWE Topic - Software Reuse

Since this project (Optimal Delivery) was somewhat similar to our minor project (Lyfter), we decided to reuse portions of the code which we had in the previous project. For example, in the Lyfter project's sprint 1 we found the shortest distance between 2 points in a given location. So we decided to reuse this portion for sprint 1 of this project along with some additional changes such as also looking for the strongest path. Since we also used Google maps API for the minor project, we were able to resume a portion of the software which we previously used since it was very accurate, and suited many of our needs.

Sprint 2

Requirements Engineering:

This will be the same as Sprint 1 as most of the requirements are the same. The only thing that is different between sprint 1 and sprint 2 is now the code asks the user to input this information. Before, this information was inputted into the code prior but now it asks the user to input this information

Requirement ID:	Requirement:	Reasoning:
R1	System reads # of trucks and asks user to input this amount	So the system can know how many points to there are for T (trucks)
R2	System need to be able to identify chosen country and mode of transportation	To be able to map out chosen location and choose mode of transportation for trucks (Which is driving)
R3	System reads # of pickup stations (asks user for this information)	So the system can know how many points to there are for P (pickup)
R4	Systems reads location of pickup station and trucks	To be able to map out the points
R5	System uses Geo-code to map	To be able to plot map
R6	System needs the # of destination points (asks user for this amount)	To be able to find the shortest path
R7	System needs to know the location of destination points	To be able to find the shortest path

R8	System needs to be able to get the shortest and strongest distance for a truck to go to pick up station and destination	To find the shortest and strongest path suitable to the customers' transportation needs.
R9	Calculate distances of all trucks combined, as well as total gasoline usage	To check and see to minimize gas costs

Non-Functional Requirements:

Non-functional requirements will be the same as sprint 1 as it still applies

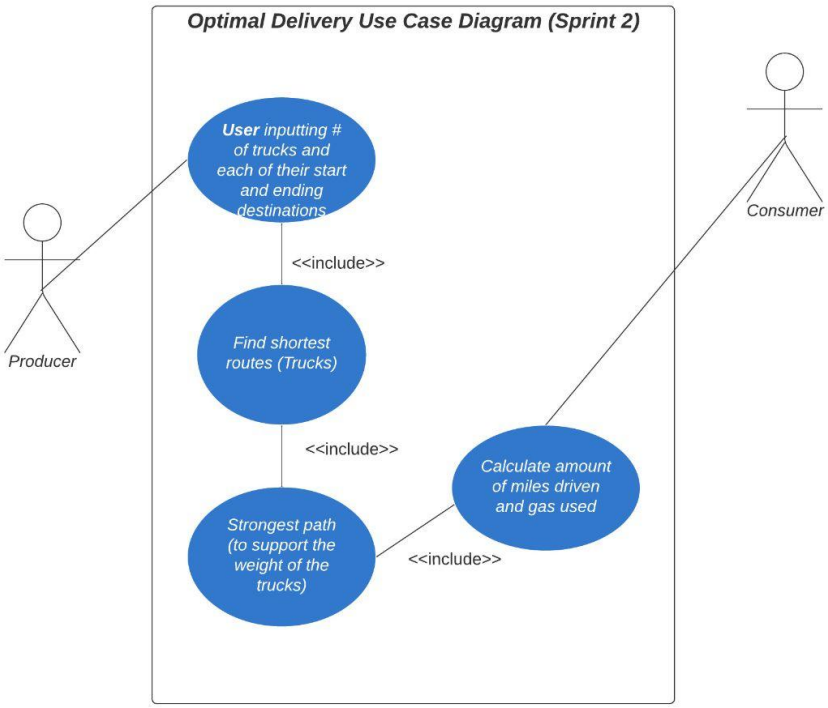
Requirement ID:	**Requirement:**	**Reasoning:**
R1	You will to connect to the internet to use this application	Need to be able to access google API
R2	Should be able to work at least 95% of the time	So that it is reasonably reliable
R3	Route choosing algorithm must be **very** accurate in choosing path to destination	This is so that the software which was engineered is reliable and accurate, to a very high degree
R4	Must be easy to access and use	Allows for a good customer experience, and easy usage would make it more customer friendly making it more popular

System Modeling:

Use Case Model:

Use case diagram

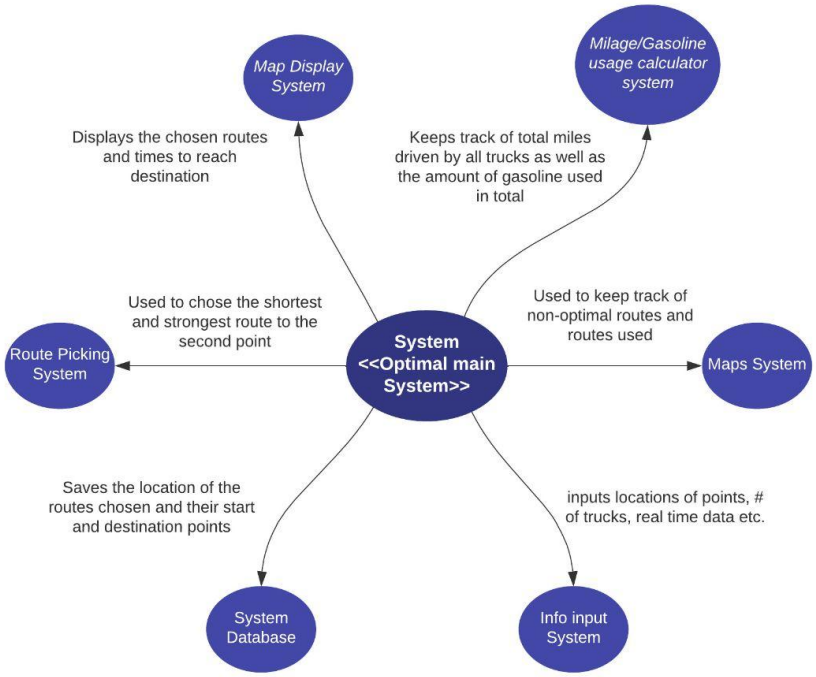
Devan | April 30, 2022



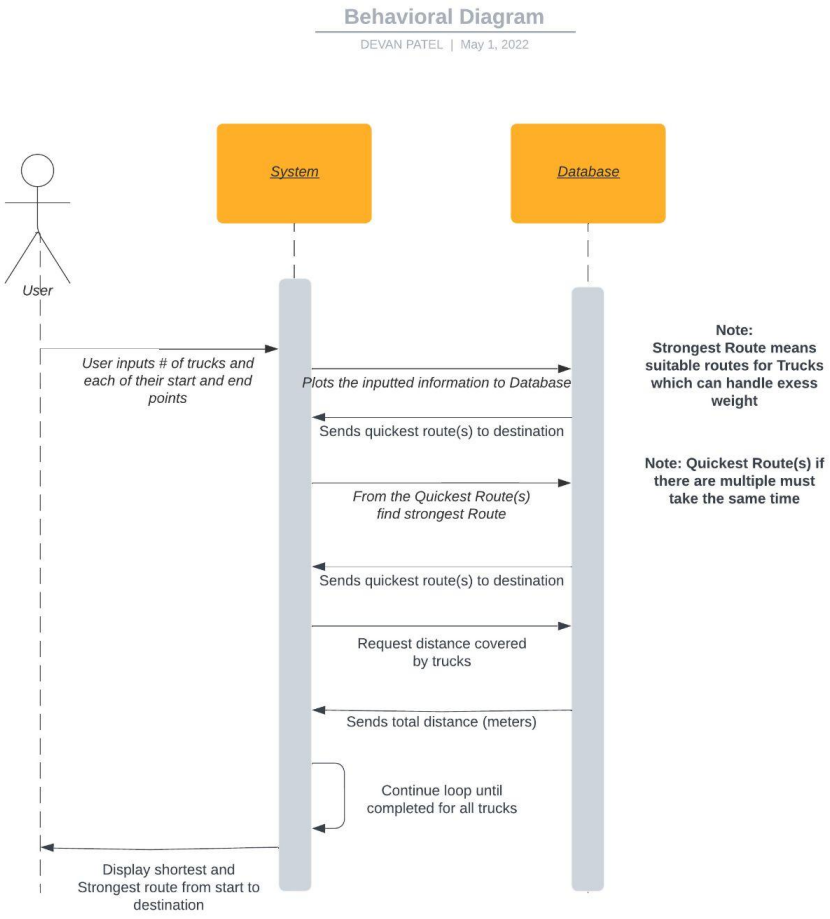
Context Model:

Spider concept map

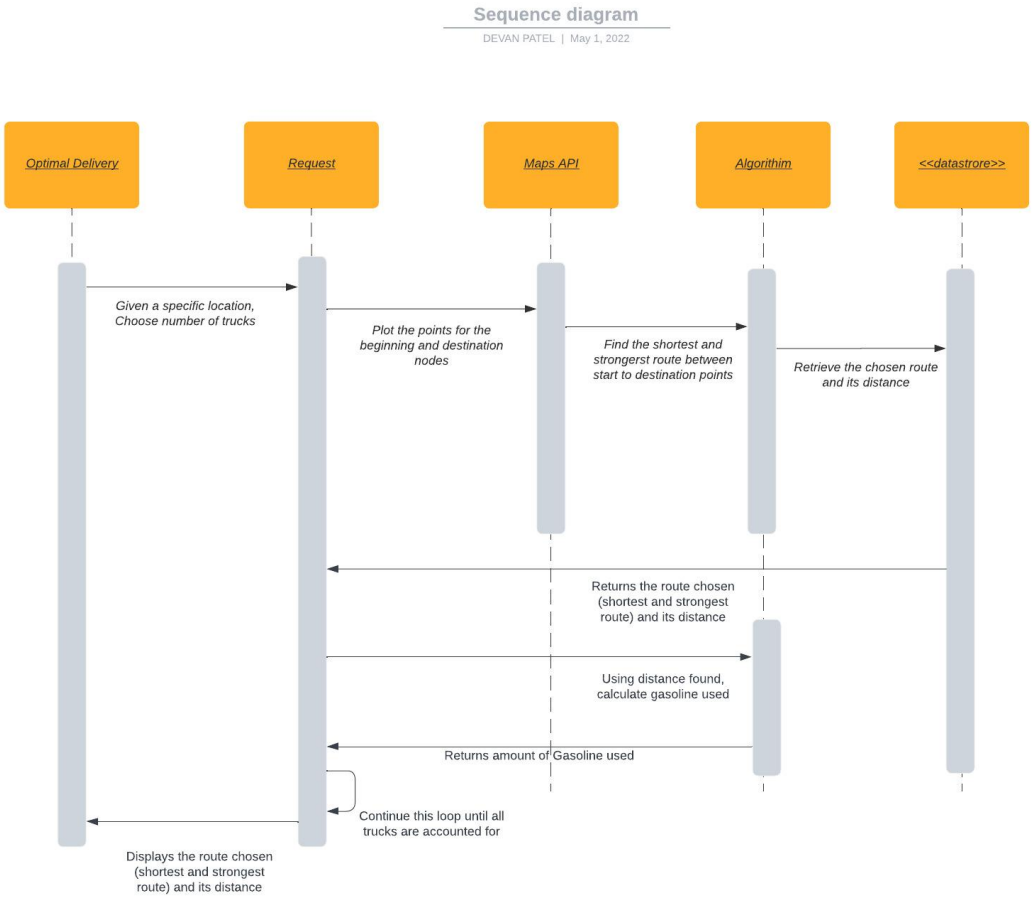
DEVAN PATEL | April 30, 2022



Behavioral Diagram:



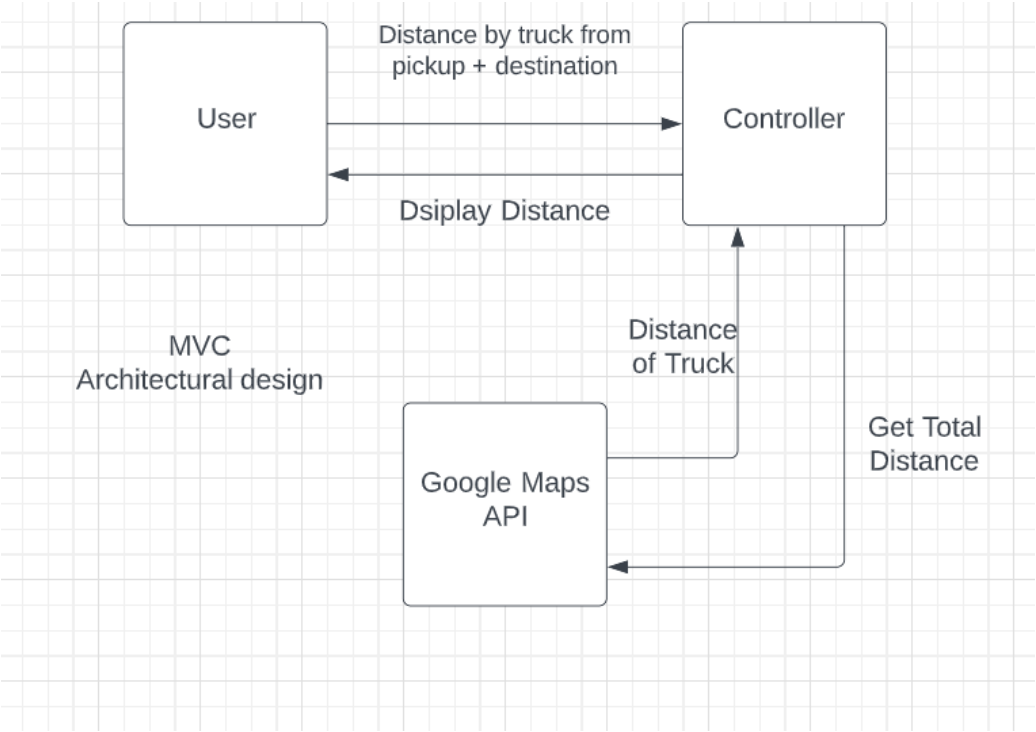
Interaction Diagram:



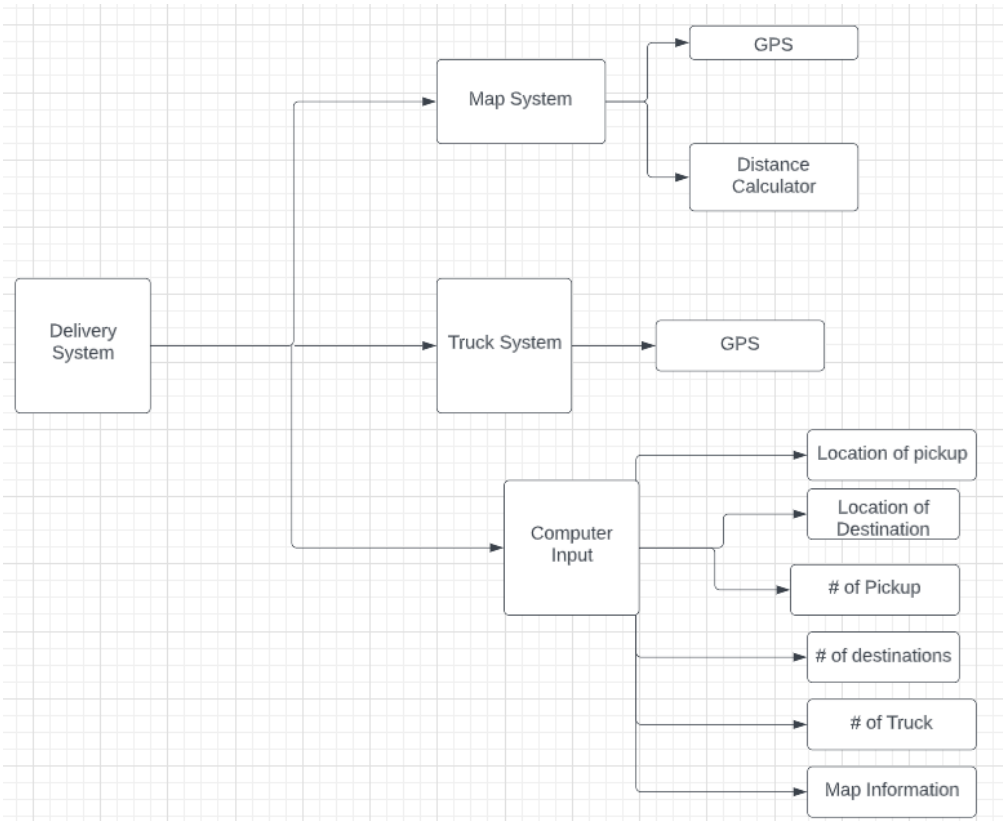
Class Diagram:

Group 33 opted not to have a Class/Structural Diagram since our software which we engineered did not have more than one class nor any classes which were reliant on each other.

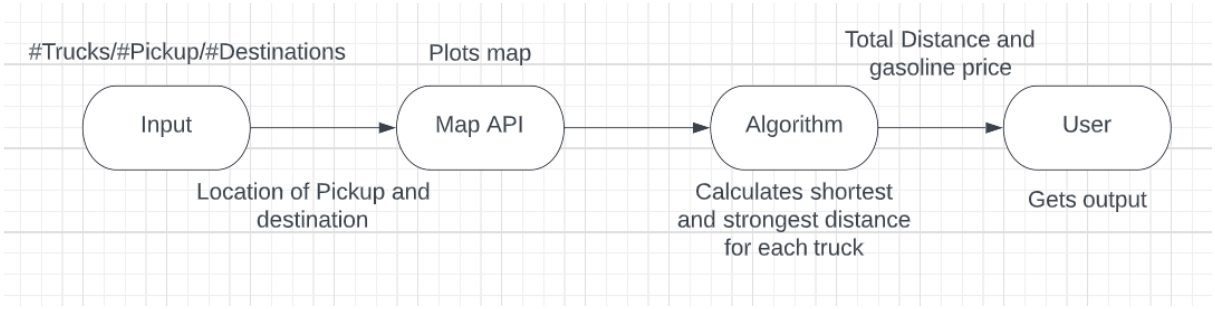
Architectural Design:



Additional Architectural Diagram View



Process Pipeline Process Structure View



(User = User interface)

Specifications:

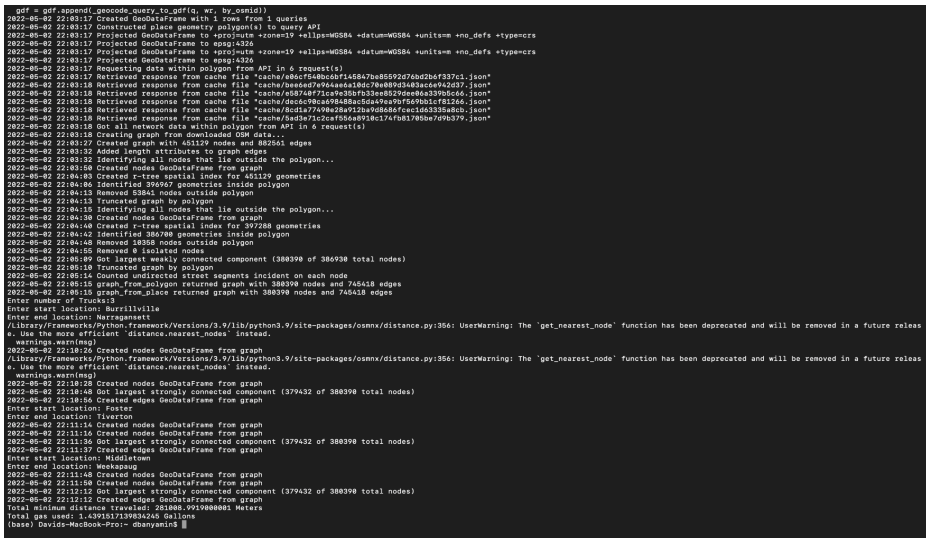
Function	Computes total distance in meters of all trucks and all routes traveled and gallons of gas used.
Description	Computes all of the total distances traveled between cities by each truck that the user chooses in a given place.
Inputs	Number of trucks/routes and names of Two cities for each truck.
Source	Cities that the truck will be going from and to, to deliver a package.
Outputs	The total distance that will be traveled in meters by all trucks and total gallons of gas used by all combined.
Destination	Google API
Action	The program will take the number of trucks first and based on that then name of each starting location city and geocode it to get its location on OSMNX using the google api then does the same for the end location and using networkx it finds the shortest route using real streets while optimizing distance over time, and repeats that for each truck.
Requires	Two cities in the same vicinity or place chosen.
Precondition	Those two places can be traveled to by drive method.
Postcondition	none
Side Effects	None

System Design and Implementation:

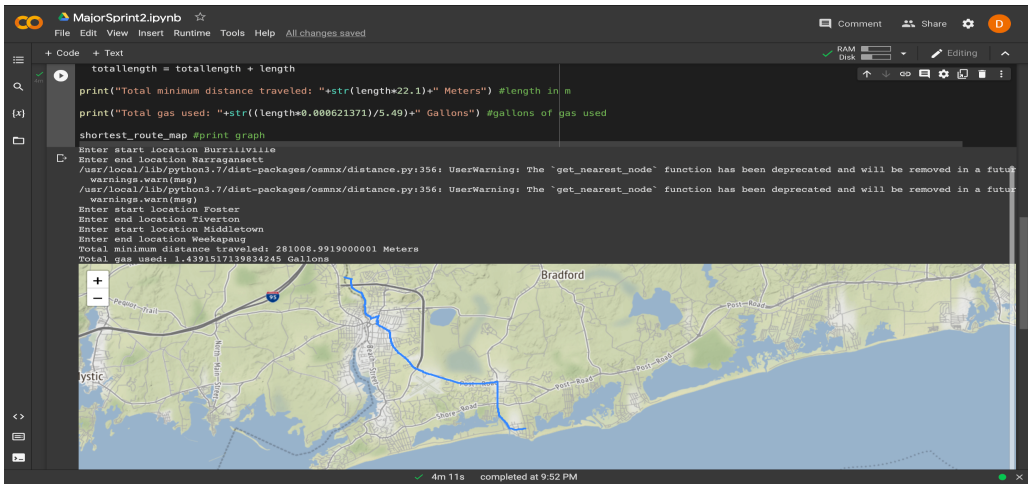
1. The first thing that the system needs to accomplish is to be able to read how many trucks and pickup stations as well as the destinations there are. To implement this on a map we will use geo-code to map out a chosen place which will also choose the mode of transportation which in this case will be driving. Once we have all these things then we will be able to calculate the shortest distance or route for each truck
2. Now that the system can read this, the user can input the number of trucks as well as the pickup stations and the location of the destination.
3. The number of trucks and pick up stations have to equal each other, the code will loop over and ask the user for a new start and end location for each truck.
4. All the pickup stations will have just 1 truck arriving and picking up to then go to a specific dropoff location
5. The code will continue to use and import osmnx to take snippets from google maps and lay out the shortest route in the real world on real roads. It also uses and imports networkx to be able to calculate the length between that found shortest distance.
9. Then the code uses osmnx to graph a map of New Jersey this time using the variables stated before.
10. The start and end location is the two cities the user chooses in New Jersey.
11. The code is able to use normal city names because of geopy, another package that needs to be downloaded which geocodes these regular names to preset locations that it has.
12. Then the start node and end node are found and using networkx the shortest path is mapped out, and then length is found and printed. And in the output it shows all of the packages loaded and the total distance traveled, as well as gas gallons used.

Software Testing

Test 1: Burrillville to Narragansett, Foster to Tiverton, Middletown to Weekapaug



Reason: We chose cities that are far from each other to see if the length of the trip would have any negative effects on the program, so the expected distance would be very large and that was what was received, 281009 Meters and 143 gallons of gas.

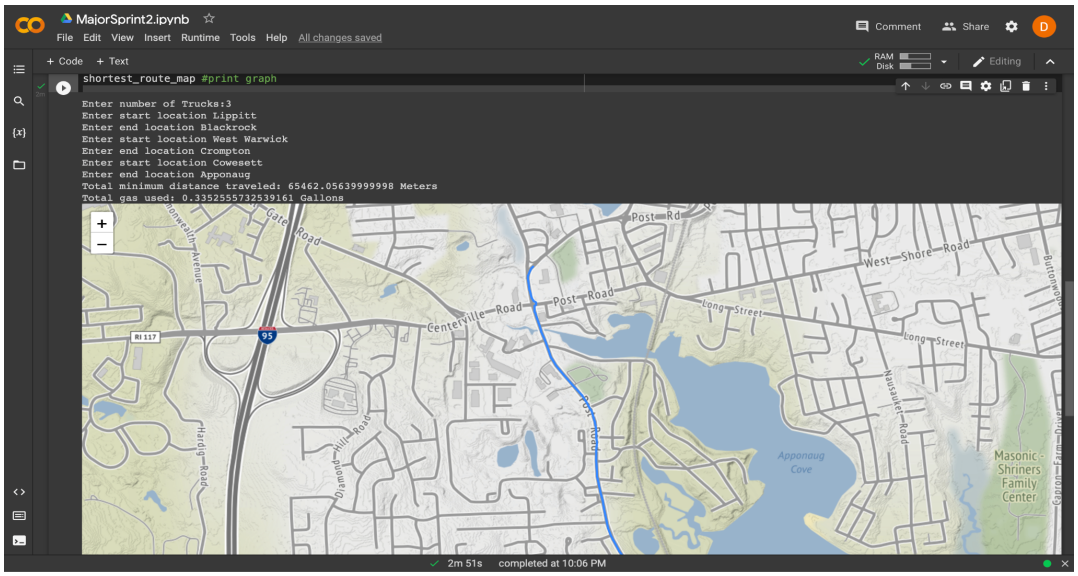


(If tested in console, rather than Google Colab then user will receive the total distance their chosen path will take rather than the entire map)

Test 2: Lippitt to Blackrock, West Warwick to Crompton, Cowesett to Apponaug

```
  gdf = gdf.append(_geocode_query_to_gdf(q, wr, by_osmid))
2022-05-02 22:14:56 Created GeoDataFrame with 1 rows from 1 queries
2022-05-02 22:14:56 Constructed place geometry polygon(s) to query API
2022-05-02 22:14:56 Projected GeoDataFrame to +proj=utm +zone=19 +ellps=WGS84 +datum=WGS84 +units=m +no_defs +type=crs
2022-05-02 22:14:56 Projected GeoDataFrame to epsg:4326
2022-05-02 22:14:56 Projected GeoDataFrame to +proj=utm +zone=19 +ellps=WGS84 +datum=WGS84 +units=m +no_defs +type=crs
2022-05-02 22:14:56 Projected GeoDataFrame to epsg:4326
2022-05-02 22:14:56 Requesting data within polygon from API in 6 request(s)
2022-05-02 22:14:56 Retrieved response from cache file "cache/e06cf540bc6bf145847be85592d76bd2b6f337c1.json"
2022-05-02 22:14:56 Retrieved response from cache file "cache/be66ed7e964a6a10dc70e8892d403ac6e942d37.json"
2022-05-02 22:14:56 Retrieved response from cache file "cache/e58740f71ca9e35bfb33ee8529dee86a339b5c66.json"
2022-05-02 22:14:57 Retrieved response from cache file "cache/dec6c98ca6984d8ac5da49ea9bf569bb1cf81266.json"
2022-05-02 22:14:57 Retrieved response from cache file "cache/8cd1a77490e28a912be9d8686fcec1d63335a8cb.json"
2022-05-02 22:14:57 Retrieved response from cache file "cache/5ad3e71c2caf556a8910c174fb81705be7d9b379.json"
2022-05-02 22:14:57 Got all network data within polygon from API in 6 request(s)
2022-05-02 22:14:57 Creating graph from downloaded OSM data...
2022-05-02 22:15:07 Created graph with 451129 nodes and 882561 edges
2022-05-02 22:15:12 Added length attributes to graph edges
2022-05-02 22:15:12 Identifying all nodes that lie outside the polygon...
2022-05-02 22:15:30 Created nodes GeoDataFrame from graph
2022-05-02 22:15:44 Created r-tree spatial index for 451129 geometries
2022-05-02 22:15:47 Identified 396967 geometries inside polygon
2022-05-02 22:15:55 Removed 53841 nodes outside polygon
2022-05-02 22:15:55 Truncated graph by polygon
2022-05-02 22:15:57 Identifying all nodes that lie outside the polygon...
2022-05-02 22:16:13 Created nodes GeoDataFrame from graph
2022-05-02 22:16:25 Created r-tree spatial index for 397288 geometries
2022-05-02 22:16:28 Identified 386700 geometries inside polygon
2022-05-02 22:16:34 Removed 10358 nodes outside polygon
2022-05-02 22:16:43 Removed 0 isolated nodes
2022-05-02 22:16:59 Got largest weakly connected component (380390 of 386930 total nodes)
2022-05-02 22:16:59 Truncated graph by polygon
2022-05-02 22:17:07 Counted undirected street segments incident on each node
2022-05-02 22:17:08 graph_from_polygon returned graph with 380390 nodes and 745418 edges
2022-05-02 22:17:09 graph_from_place returned graph with 380390 nodes and 745418 edges
Enter number of Trucks:3
Enter start location: Lippitt
Enter end location: Blackrock
/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages/osmnx/distance.py:356: UserWarning: The `get_nearest_node` function has been deprecated and will be removed in a future releas
e. Use the more efficient `distance.nearest_nodes` instead.
  warnings.warn(msg)
2022-05-02 22:19:09 Created nodes GeoDataFrame from graph
/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages/osmnx/distance.py:356: UserWarning: The `get_nearest_node` function has been deprecated and will be removed in a future releas
e. Use the more efficient `distance.nearest_nodes` instead.
  warnings.warn(msg)
2022-05-02 22:19:11 Created nodes GeoDataFrame from graph
2022-05-02 22:19:28 Got largest strongly connected component (379432 of 380390 total nodes)
2022-05-02 22:19:32 Created edges GeoDataFrame from graph
Enter start location: West Warwick
Enter end location: Crompton
2022-05-02 22:20:21 Created nodes GeoDataFrame from graph
2022-05-02 22:20:23 Created nodes GeoDataFrame from graph
2022-05-02 22:20:41 Got largest strongly connected component (379432 of 380390 total nodes)
2022-05-02 22:20:41 Created edges GeoDataFrame from graph
Enter start location: Cowesett
Enter end location: Apponaug
2022-05-02 22:21:19 Created nodes GeoDataFrame from graph
2022-05-02 22:21:21 Created nodes GeoDataFrame from graph
2022-05-02 22:21:37 Got largest strongly connected component (379432 of 380390 total nodes)
2022-05-02 22:21:37 Created edges GeoDataFrame from graph
Total minimum distance traveled: 65462.05639999998 Meters
Total gas used: 0.3352555732539161 Gallons
(base) Davids-MacBook-Pro:~ dbanyamin$
```

Reason: The opposite of the last test was done in that all of these cities were right next to each other, just neighboring towns to see how drastically the numbers would change and in this case the total distance traveled was 65426 Meters and used 33.5 gallons of gas.

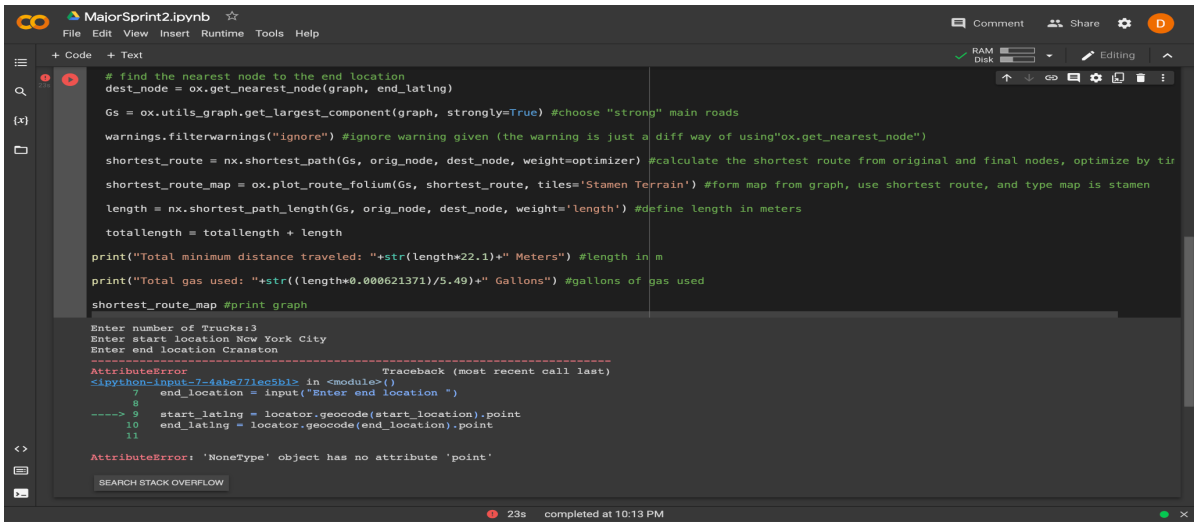


(If tested in console, rather than Google Colab then user will receive the total distance their chosen path will take, rather than the entire map)

Test 3: New York City to Cranston, Philadelphia to Bristol, Newark to Pawtucket

```
n. Use pandas.concat instead.
  gdf = gdf.append(_geocode_query_to_gdf(q, wr, by_osmid))
2022-05-02 22:21:48 Created GeoDataFrame with 1 rows from 1 queries
2022-05-02 22:21:48 Constructed place geometry polygon(s) to query API
2022-05-02 22:21:48 Projected GeoDataFrame to +proj=utm +zone=19 +ellps=WGS84 +datum=WGS84 +units=m +no_defs +type=crs
2022-05-02 22:21:48 Projected GeoDataFrame to epsg:4326
2022-05-02 22:21:48 Projected GeoDataFrame to +proj=utm +zone=19 +ellps=WGS84 +datum=WGS84 +units=m +no_defs +type=crs
2022-05-02 22:21:48 Projected GeoDataFrame to epsg:4326
2022-05-02 22:21:48 Requesting data within polygon from API in 6 request(s)
2022-05-02 22:21:48 Retrieved response from cache file "cache/e06cf540bc6bf145847be85592d76bd2b6f337c1.json"
2022-05-02 22:21:48 Retrieved response from cache file "cache/bee6ed7e964aee6a10dc70e089d3483ac6e942d37.json"
2022-05-02 22:21:48 Retrieved response from cache file "cache/e58740f71ca9e35bfb33ee8529dee86a339b5c66.json"
2022-05-02 22:21:49 Retrieved response from cache file "cache/dec6c90ca698488ac5da49ea9bf569bb1cf81266.json"
2022-05-02 22:21:49 Retrieved response from cache file "cache/8cd1a77490e28a912ba9d8686fcec1d6338a8cb.json"
2022-05-02 22:21:49 Retrieved response from cache file "cache/5ad15e71c2caf556a8910c174fb81705be769b379.json"
2022-05-02 22:21:49 Got all network data within polygon from API in 6 request(s)
2022-05-02 22:21:49 Creating graph from downloaded OSM data...
2022-05-02 22:21:58 Created graph with 451129 nodes and 882561 edges
2022-05-02 22:22:03 Added length attributes to graph edges
2022-05-02 22:22:03 Identifying all nodes that lie outside the polygon...
2022-05-02 22:22:18 Created nodes GeoDataFrame from graph
2022-05-02 22:22:29 Created r-tree spatial index for 451129 geometries
2022-05-02 22:22:32 Identified 396967 geometries inside polygon
2022-05-02 22:22:39 Removed 53841 nodes outside polygon
2022-05-02 22:22:39 Truncated graph by polygon
2022-05-02 22:22:40 Identifying all nodes that lie outside the polygon...
2022-05-02 22:22:56 Created nodes GeoDataFrame from graph
2022-05-02 22:23:06 Created r-tree spatial index for 397288 geometries
2022-05-02 22:23:09 Identified 386700 geometries inside polygon
2022-05-02 22:23:14 Removed 10358 nodes outside polygon
2022-05-02 22:23:22 Removed 0 isolated nodes
2022-05-02 22:23:37 Got largest weakly connected component (380390 of 386930 total nodes)
2022-05-02 22:23:37 Truncated graph by polygon
2022-05-02 22:23:42 Counted undirected street segments incident on each node
2022-05-02 22:23:42 graph_from_polygon returned graph with 380390 nodes and 745418 edges
2022-05-02 22:23:43 graph_from_place returned graph with 380390 nodes and 745418 edges
Enter number of Trucks:3
Enter start location: New York City
Enter end location: Cranston
/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages/osmnx/distance.py:356: UserWarning: The `get_nearest_node` function has been deprecated and will be removed in a future releas
e. Use the more efficient `distance.nearest_nodes` instead.
  warnings.warn(msg)
2022-05-02 22:24:25 Created nodes GeoDataFrame from graph
/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages/osmnx/distance.py:356: UserWarning: The `get_nearest_node` function has been deprecated and will be removed in a future releas
e. Use the more efficient `distance.nearest_nodes` instead.
  warnings.warn(msg)
2022-05-02 22:24:27 Created nodes GeoDataFrame from graph
2022-05-02 22:24:44 Got largest strongly connected component (379432 of 380390 total nodes)
2022-05-02 22:24:49 Created edges GeoDataFrame from graph
Enter start location: Philadelpha
Enter end location: Bristol
2022-05-02 22:25:16 Created nodes GeoDataFrame from graph
2022-05-02 22:25:18 Created nodes GeoDataFrame from graph
2022-05-02 22:25:35 Got largest strongly connected component (379432 of 380390 total nodes)
Traceback (most recent call last):
  File "/Users/dbanyamin/sprint2.py", line 38, in <module>
    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
  File "/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages/osmnx/folium.py", line 137, in plot_route_folium
    gdf_edges = utils_graph.graph_to_gdfs(G.subgraph(route), nodes=False).loc[uvk]
  File "/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages/osmnx/utils_graph.py", line 65, in graph_to_gdfs
    raise ValueError("graph contains no edges")
ValueError: graph contains no edges
(base) Davids-MacBook-Pro:~ dbanyamin$
```

Reason: We did this final test using cities outside the defined area of Rhode Island, using cities such as NYC, Philadelphia, and Newark, and immediately the program outputted “networkx.exception.NodeNotFound: Either source 4168253339 or target 103999154 is not in G” Which is exactly what it was supposed to do.



(If tested in console, rather than Google Colab then user will receive the total distance their chosen path will take rather than the entire map)

Evaluation

Regarding our test cases for sprint two, we first tested cities that were very far from each other in order to see if adding much larger distances together would affect the program in any way. So, we went ahead and chose cities which were very far from each other. We chose 3 pairs of cities far away from each other, the program was able to execute properly and brought up expected or similar results to other mapping systems, it was also able to calculate the total gallons of gas used in total by the trucks. For our next test we chose cities centered around a body of water and they were much closer to each other to see if there would be any limitations to our software to find the optimal route as expected. For the final test case we chose cities that were outside of the boundaries that were in place (Rhode Island) and chose cities such as New York, Philadelphia and Newark to use as pickup/destination points and as expected we got an error since these cities were not located in the original specified location which was Rhode Island. Additionally for the successful tests, we were able to handle multiple trucks and find the optimal routes to each of their individual drop off points which could handle the trucks and would have them use the least amount of gas (which was one of the main requirements).

When testing the system we were able to check that the system would have an input feature where it would receive the number of trucks from the user as well as chosen start and destination points for each of them. The system is also able to identify the chosen area for the routes and mode of transportation. We can also confirm that the system reads the number of pick up stations and the location of them as well as the locations of the trucks. Similar to sprint 1 it once again uses geocode to map. It also is able to read the number of destination points and their destinations. Once again the system was also able to find the shortest and strongest route between any two points in the given area. Finally the most important portion was that the system kept track of the amount of total millage the trucks covered and gasoline used and would try to minimize it as much as possible.

Dependable Attribute:

- **System Security:**
    - We did not create our own maps API since this would have taken a large amount of time to develop as well as resources. Additionally if we had developed our own API, then it would easily be prone to external attacks. However we ended up using a Google Maps API which helps keep the system secure, since Google is very well versed with security protocols for their applications/sites. Because we used Google Maps API, we can keep our system secure as well as the location information safe because it is stored with Google. **Once again, we used Google's API in order to keep the mapping and location data that is being used by our system safe from external or malicious attacks. Because we used Google's API the information is kept very secure and would be using their security making our system secure as well.**

Project Management Concept: Gantt Chart (Sprint 2 timeline)

Gantt chart Sprint 2

Devan | May 2, 2022

	4/24	4/25	4/26	4/27	4/28	4/29	4/30	5/1	5/2
Requirement Engineering (documentation)									
System modeling									
Architectural Design									
System Testing									
Implementation (documentation)									
System specifications									
Dependable attribute (documentation)									
Advanced SWE Topic (documentation)									
Software implementation									

Legend: Devan Patel; David Banyamin; Mark Rezk

Advanced SWE Topic - Software Reuse

Since this project (Optimal Delivery) was somewhat similar to our minor project (Lyfter), we decided to reuse portions of the code which we had in the previous project. For example, in the Lyfter project's sprint 1 we found the shortest distance between 2 points in a given location. So we decided to reuse this portion for sprint 1 of this project along with some additional changes such as also looking for the strongest path and the use of multiple trucks. Since we also used Google maps API for the minor project, we were able to resume a portion of the software which we previously used since it was very accurate, and suited many of our needs. **In this sprint we continued from sprint 1, and followed up by using the same API and additional portions of the code from our minor project's sprint two, in order to take into account of the number of trucks this time (rather than number of drivers/riders previously for Lyfter project)**