Minor Project - Team 4

Akanksha Arun - aa2013

Eshaan Mathur - em919

Eoin O'Hare - eso25

Sidhu Arakkal - ssa150

Mohammad Awais Zubair - maz106

Github Link: https://github.com/radical-teach/minor-project-team-4.git

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.

Sprint documentation must include:
Requirements engineering; System Modeling; Architectural Design; Design and Implementation; Software Testing (across all levels); and Evaluation. Especially in the final sprint.

Testing: **unit testing, integration testing, system testing.**

Sprint 1

Activities Emphasized

- Requirements Engineering
- System Modeling
    - Activity Diagram (layout for sequence diagram)
    - Conceptual Diagram
    - Sequence Diagram

The team planned on how the project would be laid out, starting off by deciding what front end language we would be using, deciding on Python.

This sprint involved a lot of brainstorming and planning.

Our main goal is to communicate ideas so everyone can be on the same page.
The next step was to take in consideration of the user requirements and specifications:

Requirements Engineering

Non-Functional Requirements:

- Reliability of the program
- Security isn't major priority
- Readable data (not just coordinates)
- Easy usability for customers

Functional Requirements:

- Find 'optimal route' for trucks with the minimum distance
- Allow for many pickup and delivery points
- Calculate gas cost for each route and the total price, given the gas price input
- Check whether available trucks meet the intended number of routes

Initial Plan Using UML

Use Cases

1. User logins in with credentials
2. System scans databases to check credentials
    a. If login is validated, then user enters application
    b. Otherwise, user must try to validate credentials again

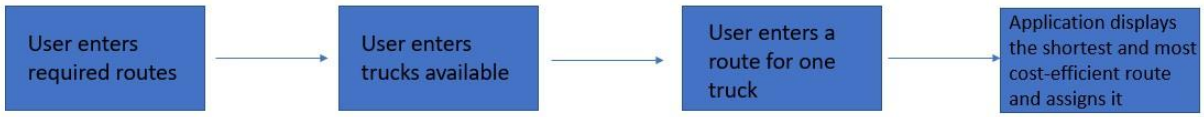
3. User chooses from a selection of pickup points
4. User enters the destinations required to be delivered to
5. User enters how many trucks are available
6. System checks if the amount of trucks available is sufficient for all deliveries
    a. If not, application notifies user and exits (maybe how many deliveries would be possible)
    b. If the amount of trucks are sufficient, the application continues
7. Based on destination point, direct truck to the closest pickup point in the shortest distance
8. Determine the best route for the truck to deliver the packages to the the destination points from the pickup point
9. Calculate the cost of gas for the trucks and the total miles traveled per truck

Next, we wanted to focus on the system and modeling.

System Modeling

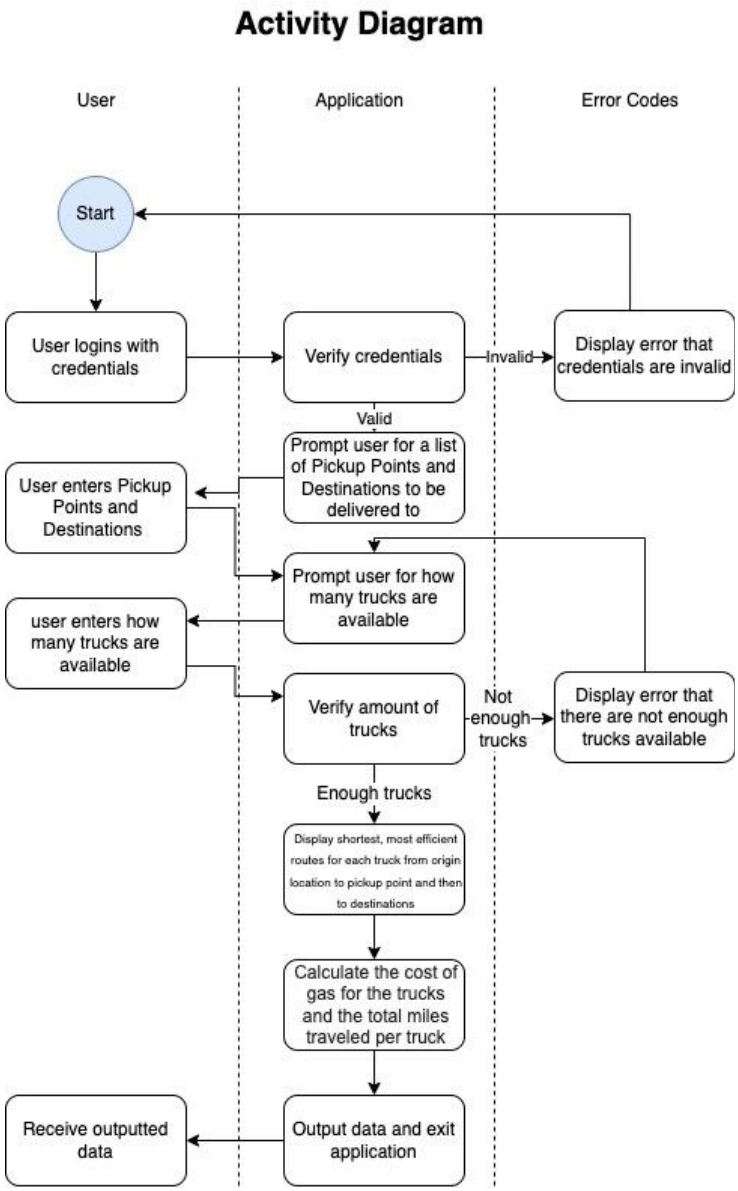
Activity diagram, sequence diagram, conceptual diagram and process modeling. We do this so that the client can get a better understanding on how we are trying to approach their requirements in an easy way so there would be no conflicts or misunderstandings.

Conceptual Model



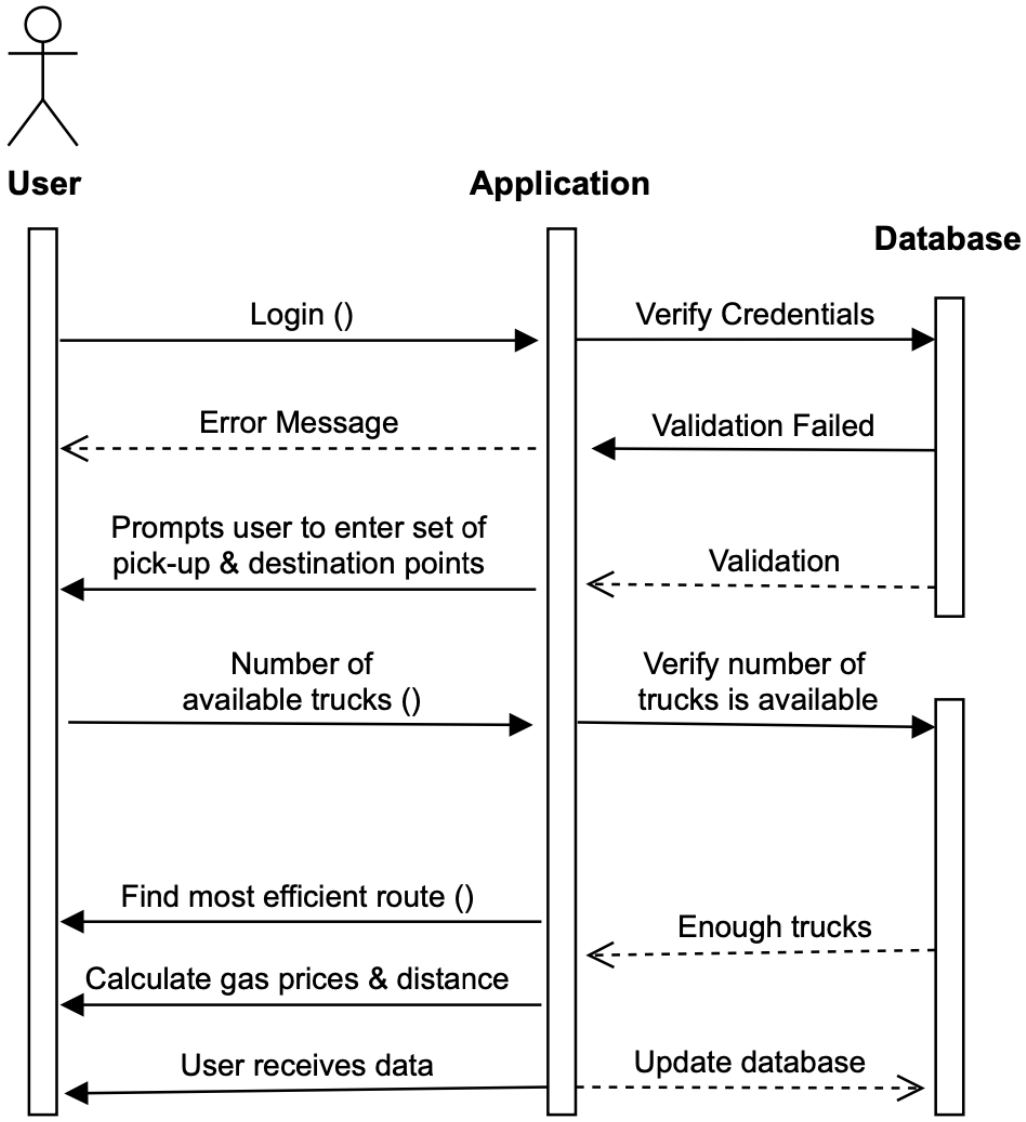
This shows the basic model on how to go about the program.

Activity Diagram



This is the activity diagram builds up on the conceptual model as it implements the use cases.

Sequence Diagram



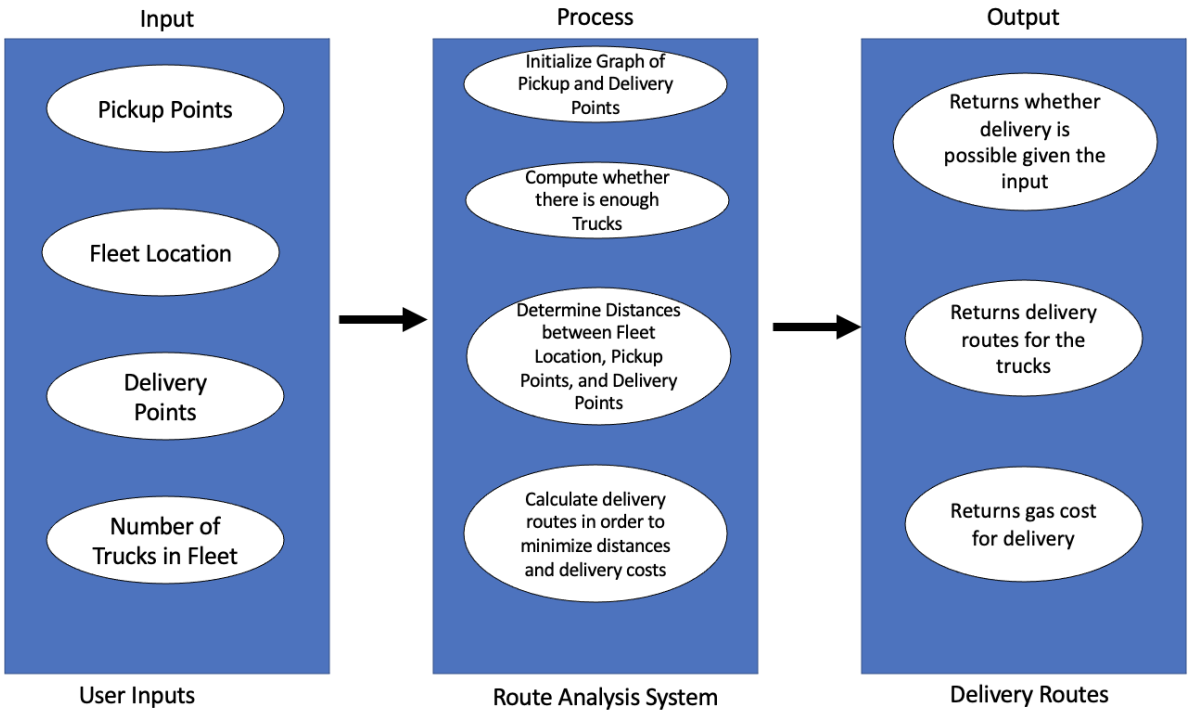
The ideology behind the sequence diagram stemmed from the activity diagram. This shows what possible errors occur and how it plays out. We were able to see the different routes that could be taken that involve the user, the application, and the database. This is useful for testing and defect testing to check how the program responds to errors.

Sprint 2

Activities Emphasized

- Architectural Design
- Design and Implementation

Architectural Design



We believe that the transaction model was the best use for this because it makes it easier to comprehend, spelling out the process of the code and the model to the clients in a way they could understand. This also shows the relationship of the use cases with each other.

Design and Implementation

We used an object oriented design process by developing a lot of models to help us in the previous sprint to give us an idea of how to analyze and implement the problem. Part of using agile methods is to be able to actively implement the designs and create a process that is friendly to changes and this is what sets it apart from plan driven approach.

Problem Analysis

From a very broad sense, the OptimalDelivery problem (based on our assumptions) is a modified version of the Travelling Salesman problem. The Traveling Salesman problem is a computer science problem where the objective is to find the shortest path between a set of locations that must be visited. In the regular Traveling Salesman problem, there is no specified starting point or ending point, thus the path returned is just optimized to select the starting and ending point that result in the shortest path. This distinction becomes much more relevant later in this section when it comes to the design of the shortest path implementation.

Libraries Utilized

mlrose-hiive

This library contains some of the most common optimization and search algorithms. In addition, it includes functionality to help solve some common computer science problems (knapsack problem, neural network weight problem, traveling salesman problem, etc).
Dependent Libraries: NumPy, SciPy, Scikit-Learn

PyInputPlus

This library assists with user input collection and input error handling. It is used when collecting all inputs except the initial password prompt.

bcrypt

This library is used to hash the inputted password and compare it against the stored password hash for authentication/

Passlib

This library assists with password input collection. This library is used when collecting the initial password prompt to prevent the password from being logged.

Pytest

This library assists with creating tests for functions, as it provides detail in which tests failed and how. This library is used to create the unit tests for our functions.

Command Line Interface

When we developed the Command Line Interface (CLI), we aimed to make a simple, user friendly interface that neatly organized information and allowed for easy navigation through the program. In order to achieve this, we needed to find a way to organize different messages by their context and implement error handling during input collection.

To achieve the first task, we used color-coding to display the context of a message. For example, when an inputted password was incorrect, the error message would be colored red and when the inputted password was successful the response message would be green. This color coding allows users to differentiate between messages when the application prints multiple responses to the CLI.

To achieve the second task, we used a library called PyInputPlus in order to collect user input. This library had built-in error handling and helpful tools for efficient user input collection. For one, the library can be configured to only take one type of input. For example, when our application asks how many trucks are available the user input can only be an integer. If anything else is inputted, an error message will be displayed and the prompt will be repeated. This prevents unsupported data types from being used in the application helper functions and causing errors. Moreover, the library allows users to pick options from a list which is used in the route selection function.

Overall, the command line interface uses color coding, concise messages, and smart input collection in order to create a user-friendly CLI for interfacing with our application.

Shortest Path Implementation (Iteration 1)

The first iteration of the shortest path implementation involves the use of mlrose's coordinate grid system to find the shortest path. Once the user input has been converted into a list of coordinates to find the shortest path between, an optimization problem must be declared. Utilizing the TSPOpt function from mlrose we get an optimized route object. Then, by running a randomized optimization algorithm (genetic_alg) on the route object we should get a valid solution.

Issues with this Implementation

Upon implementing this method of solving the OptimalDelivery problem, the group noticed a glaring problem with the method when doing manual user testing. Since our problem statement is a modified Traveling Salesman problem, we needed the route to have a particular start and end point. Unfortunately, when using the genetic_alg function

with coordinates, there was no way to reorder the output in a way that provided the shortest path with a specified start/end point. Thus another method must be developed to account for having a specified start and end point.

The design and implementation of this sprint was the code prior to this commit (this sprint's MVP).

Sprint 3

Activities Emphasized

- Requirements Engineering
- System Modeling
- Architectural Design
- Design and Implementation
- Software Testing
- Evaluation

Requirements Engineering

Non-Functional Requirements:

- Reliability of the program
- Security isn't major priority
- Readable data (not just coordinates)
- Easy usability for customers

Functional Requirements

- Find 'optimal route' for trucks with the minimum distance
- Allow for many pickup and delivery points
- Calculate gas cost for each route and the total price, given the gas price input
- Check whether available trucks meet the intended number of routes

Initial Plan Using UML

Use Cases

1. User logins in with credentials
2. System scans databases to check credentials
    a. If login is validated, then user enters application
    b. Otherwise, user must try to validate credentials again
3. User enters how many trucks are available
4. User enters how many routes they have
5. System checks if the amount of trucks available is sufficient for all deliveries
    a. If not, application notifies user and exits (maybe how many deliveries would be possible)
    b. If the amount of trucks are sufficient, the application continues

6. User enters the destinations required to be delivered to for their respective truck
7. Determine the best route for the truck to deliver the packages to the the destination points from the pickup point
8. Calculate the cost of gas for the trucks and the total miles traveled per truck

Assumptions

- A trucks can carry unlimited amount of packages
- Each route only has one pickup point
- The first location of the route is the pickup point
- The last location of the route is always the Truck Depot
- We start calculating distance from the pickup point
- You can't drop off at the same place twice in a route (List to Set -> Set Size Comparison)
- Locations are on a coordinate grid

System Modeling

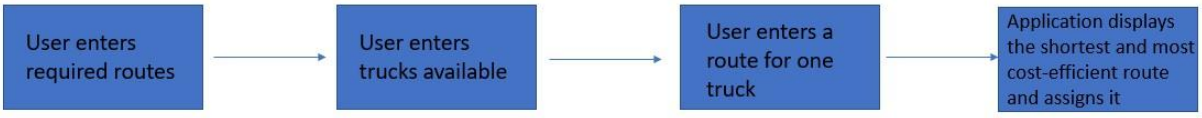
This meeting we focused on the modeling aspect where we model all the use cases and then focus on the system modeling and architectural design by implementing: activity diagram, sequence diagram, conceptual diagram and process modeling. We do this so that the client can get a better understanding on how we are trying to approach their requirements in an easy way so there would be no conflicts or misunderstandings.

We are assuming these factors before consideration:

- The starting point will always be the first location that the client inputs.
- The user will always end at the truck depot (the ending point is concrete).

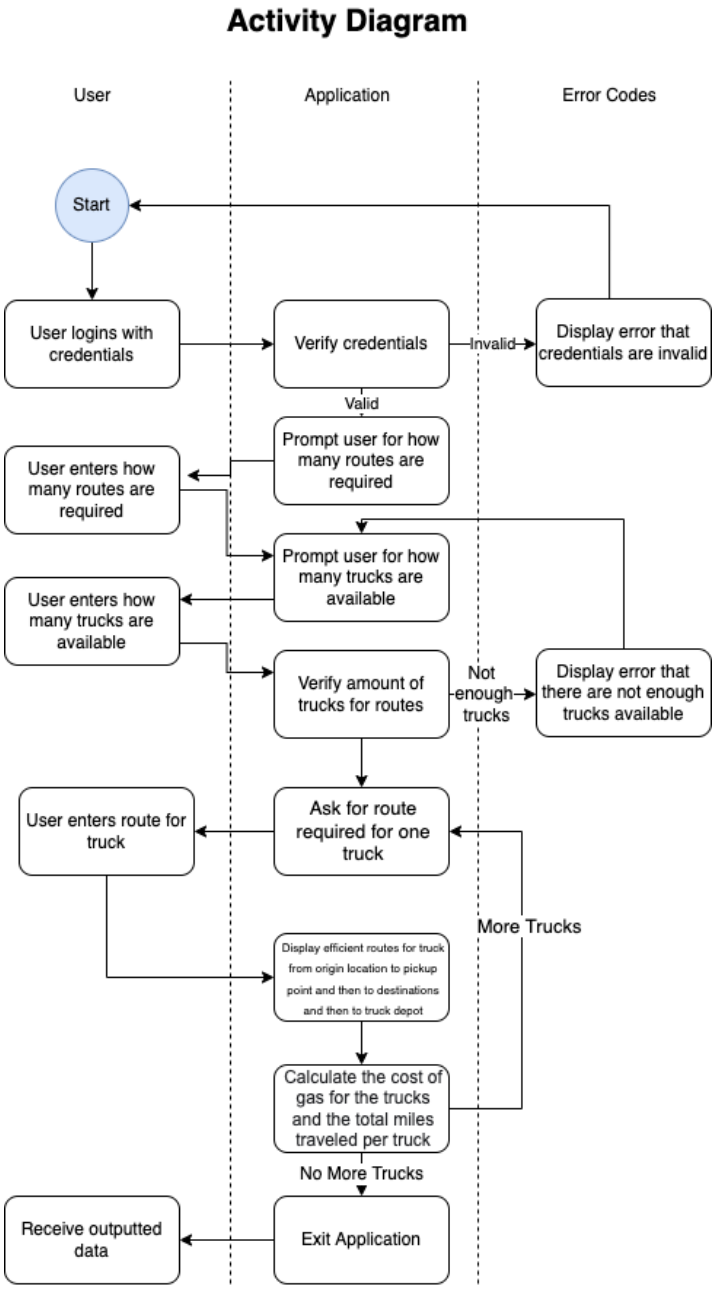
Here are the diagrams below:

Conceptual Model



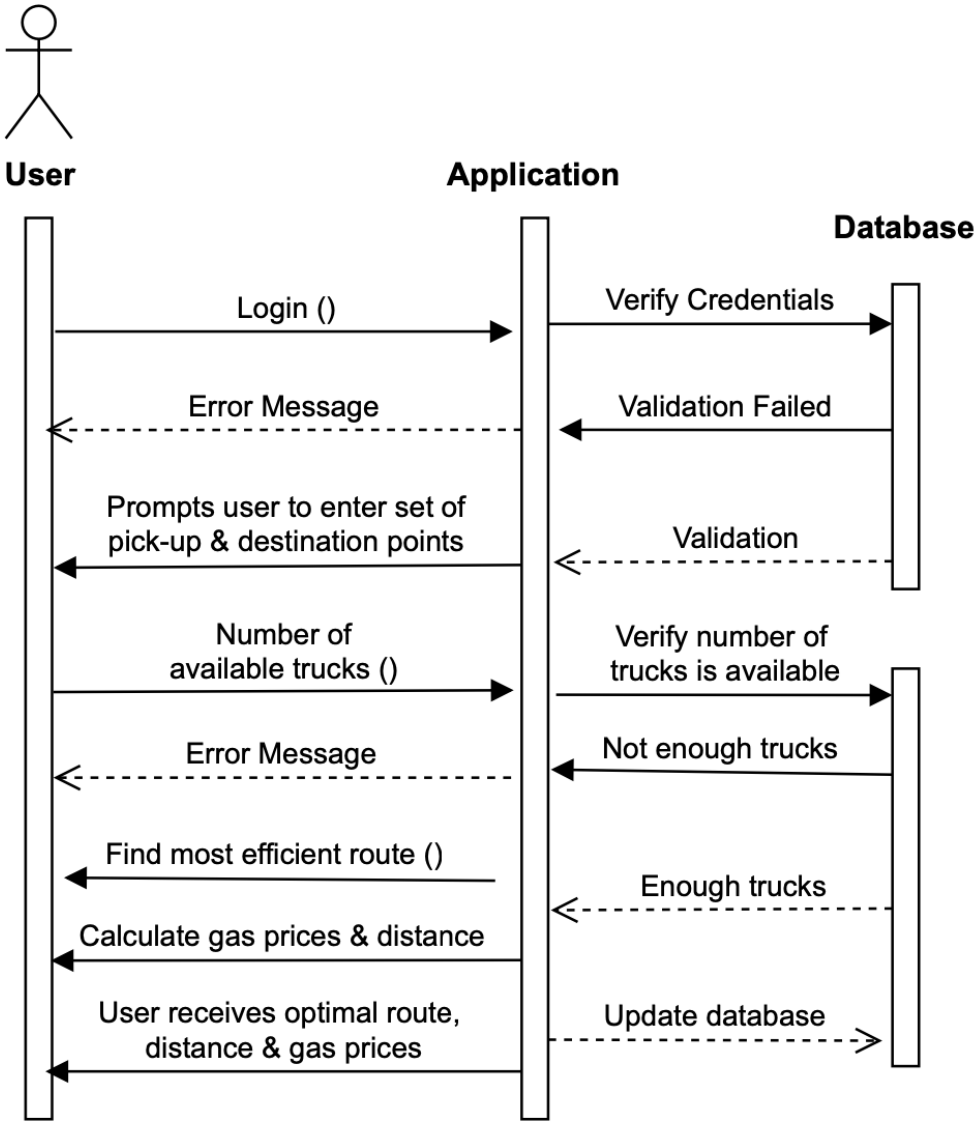
In the simplest way possible, this is what we want our program to carry out. The basic concept to help clients understand where exactly we are taking this. To say, the “best path” in approaching this issue and build on from here using the following diagrams listed below.

Activity Diagram



This shows the client an expanded version of the concept diagram and goes into the specific details by using the use cases and connecting them to each other. Here we can see how the basis of our program is going to look. This diagram added updates on adding more trucks for certain cases with less trucks than routes. This addition of more trucks is now classified as required when compared to the original version in Sprint 1 which did not have this constraint.

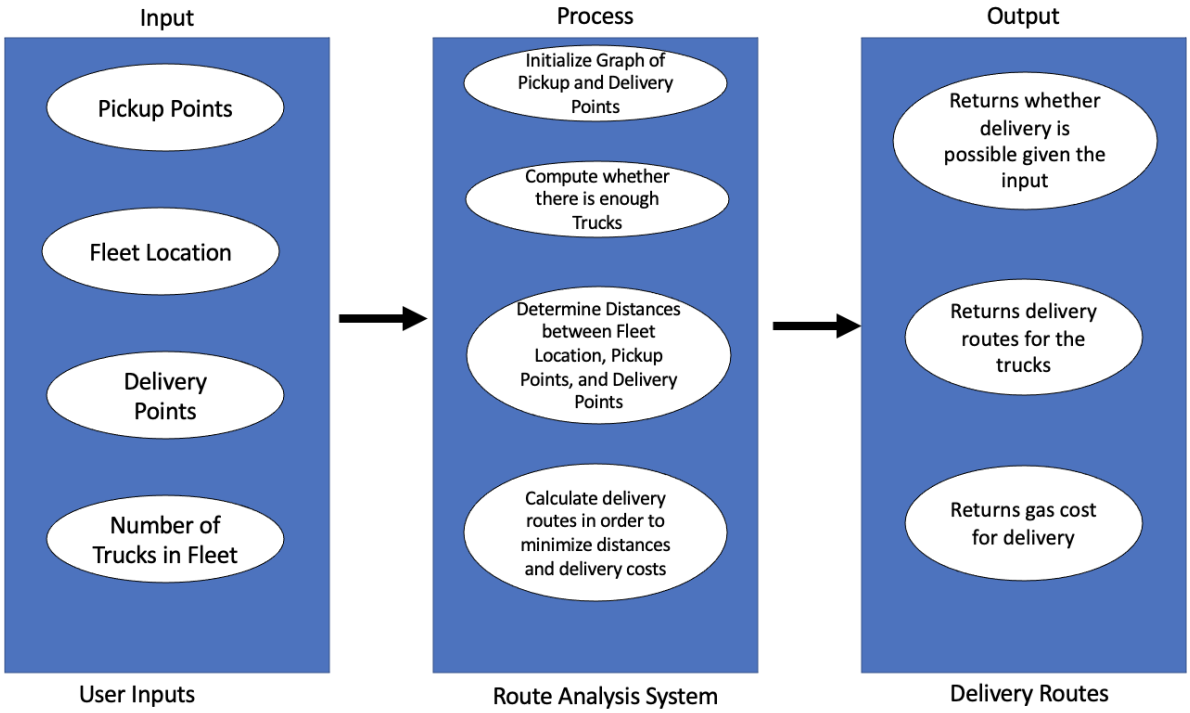
Sequence Diagram



This diagram is formatted the same as the diagram found in Sprint 1, however, it is more specific since we got an update from the activity diagram side of things. More requirements were mentioned for what would happen if there are not enough routes to support a specific number of trucks and the output to the user was specified as the optimal route, distance, and cost of gas for every route.

Architectural Design

Process Model



In our third sprint, we again revisited the architectural design to determine if our implementation still followed our initial transactional model, and it still followed the pattern of an input -> process -> output system. The input was still the user inputs, the process was our route analysis system, and the output was the delivery routes along with the respective costs and distances. This transaction model proved to be the best for this scenario as it made it easier to comprehend, spelling out the process of the code and the model to the clients in a way they could understand. This also elaborates on the use cases of the system.

Design and Implementation

The design and implementation of this sprint was done after the merge of this commit. The final MVP is the most up to date code in the main branch. The new method of determining the shortest path was minimally tested on a separate branch before being merged into the main branch to undergo rigorous testing.

Shortest Path Implementation (Iteration 2)

The second iteration of the shortest path implementation involved the creation of a weighted graph and a dummy node to find the shortest path given a specified start and end point. Once the user input was converted into a list of coordinates, a weighted graph was created from these inputs. This was stored as a tuple with three items: location 1, location 2, and distance between the two locations. For example, given three locations (0,1,2) the weighted graph would contain three tuples: (0,1,distance), (0,2,distance), and (1,2,distance).

Then a dummy node was inserted into the weighted graph, which had a large predetermined distance between all points except the starting and ending points, which had a predetermined small distance. Going back to the previous example, assuming the start point is 0, the end point is 2, and the dummy node is 3, the following tuples would be added to the weighted graph: (0,3,0.1), (1,3,100000), (2,3,0.1). The reason a dummy node is needed is to ensure that one of three scenarios occurred when using the mlrose library:

1. The shortest path list generated starts with the starting location and ends with the ending location, thus no reordering needs to be done.
2. The shortest path list generated starts with the ending location and ends with the starting location, thus the list needs to be returned backwards.
3. In any other case, the shortest path would return a list where somewhere in the list the order is: [..., starting point, dummy node, ending point, …] or [..., ending point, dummy node, starting point, …]. Then the reorder function would find the starting point and then reorder the list either forwards or backwards depending on whether the dummy node was before or after the starting point.

Since this implementation involves a weighted graph, before getting to the optimization and randomized optimization algorithm, a fitness function must be generated from the weighted graph. To do so, the TravellingSales function was utilized from the mlrose library. The output of the TravellingSales function was then used in the TSPOpt function to get the optimized route object, which had randomized optimization algorithms run on it using genetic_alg which generated the optimal route. This optimal route then had to be reordered as mentioned above, and the dummy node removed from the route before returning the route to the program. In this way, we were able to account for a specified starting and ending point and properly solve the OptimalDelivery problem.

Software Testing

Through the development phase, we have to go through all levels of testing. This is to show that the program does what it is intended to do and to discover any defects before the product is released. This is to give clients and stakeholders confidence and build the trust that you have given them a bug-free product or are able to take care of bugs quickly due to the agile process approach that favors any changes. This makes it cost and time efficient.

There are two main goals of testing:

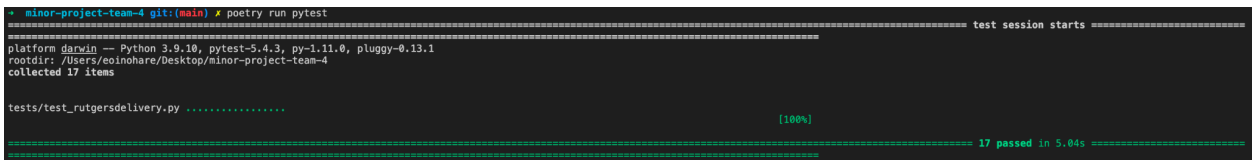
1) Validation testing
    a) This is where we check if the program meets its requirements and the software carries out as intended
2) Defect testing
    a) To discover faults and defects in the software
    b) Test cases are used to show where there are defects in the software

These types of testing fall under the development testing.

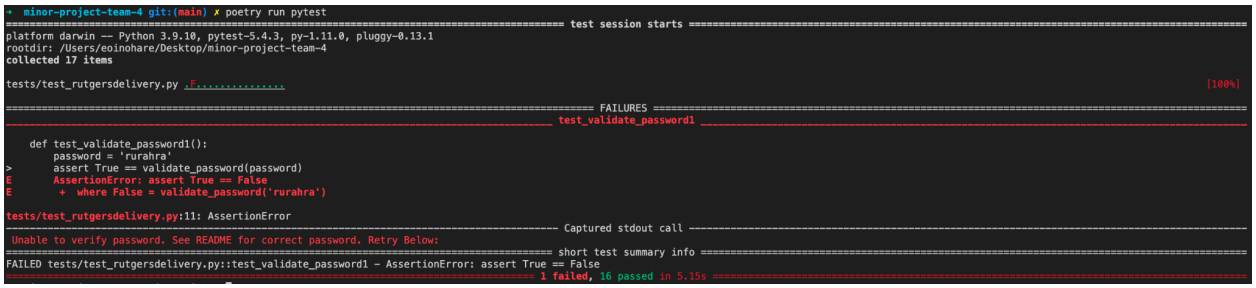
Unit Testing

The purpose of this testing is to check for any defect and falls under the defect testing aspect.

Our goal for unit testing was to create automated tests for all our functions individually, giving them a predetermined input and the expected output based on that input. If the expected output did not match the output generated from the function, it would indicate that the function's test failed. As the tests are calling the functions from the main code, the tests will always utilize the currently used functions and check whether any changes made to the code now creates an unintended output. Therefore, our unit tests also test for regression. We created a python script called test_rutgersdelivery.py and it is run through the “poetry run pytest” command. We have 17 total tests and as of submission, all of our tests passed, as shown in the image below.



An example of a failed test is shown below, where the test_validate_password1(): test is checking whether the user inputting the correct password would generate the intended response:



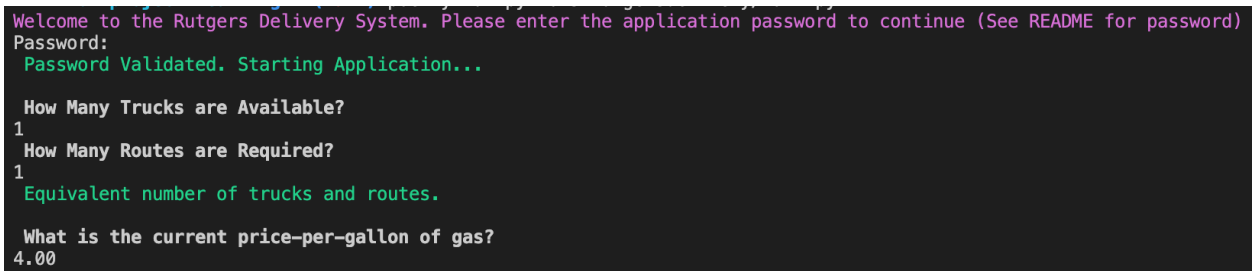
Our unit testing allowed us to check for any defects within our functions, and gave us confidence that our functions worked as intended. After completing all the unit tests and making sure they were done correctly and would update with changes to the functions' code, we moved onto integration testing.

Integration testing:

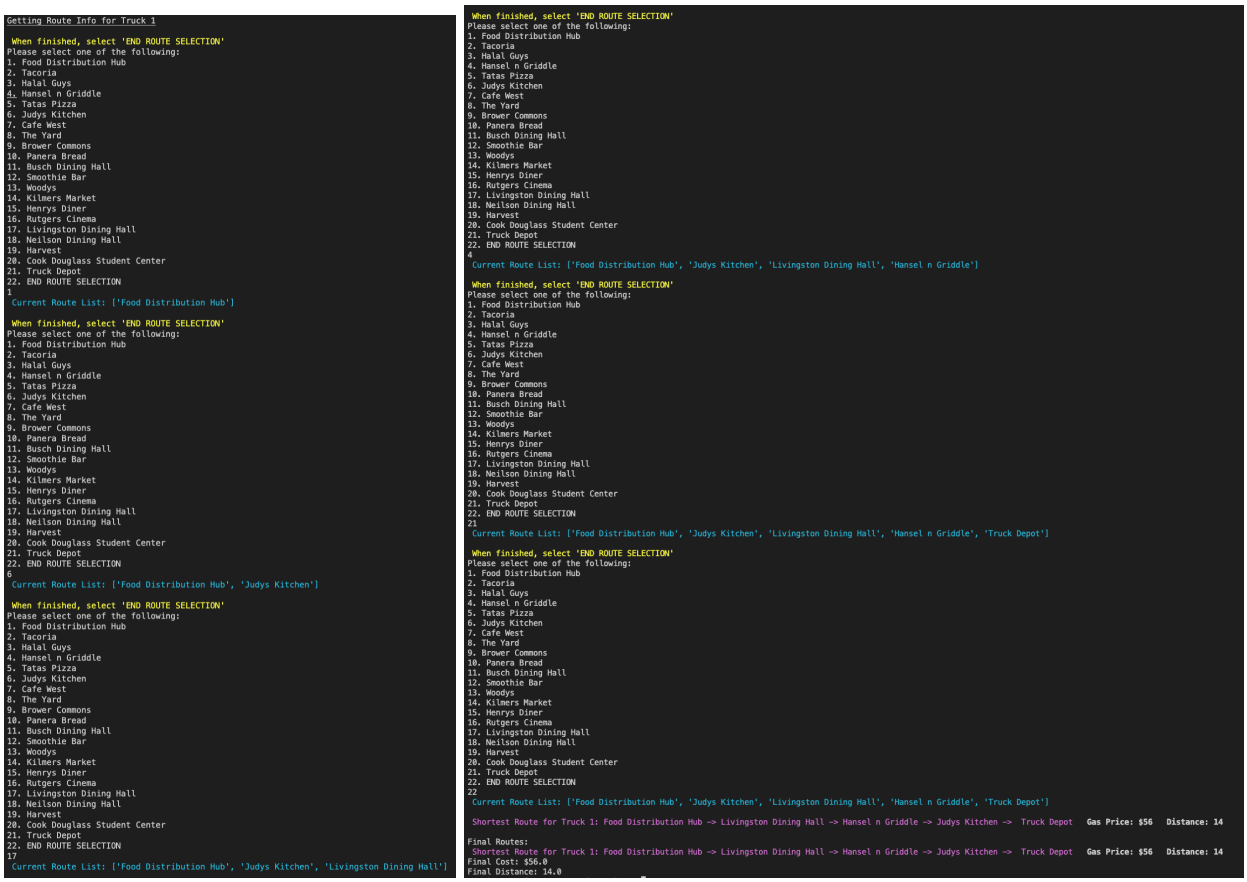
The purpose of this test is to test the interfaces between modules and is a way of defect testing as this exposes any faults or defects.

We utilized integration testing to view the user input interface, function components, and output interface to identify any faults or defects in the different sections of our code. The input interface incorporates user input for the password, the passwordValidated function, and user inputs for the number of trucks, pickup points, and delivery points. After using our automated unit testing to ensure that each function in the user input component of the system worked separately, we tested the edge cases of inputs in order to test that the user input component works as a whole.

Our first test was selecting the route in a scenario where only one truck was available and only one truck was needed.



As shown in the picture above, the processes for inputting the password, verification, gas price, and inputting the trucks available and required worked as intended. The next step in the user input interface was inputting the points for the truck's route.

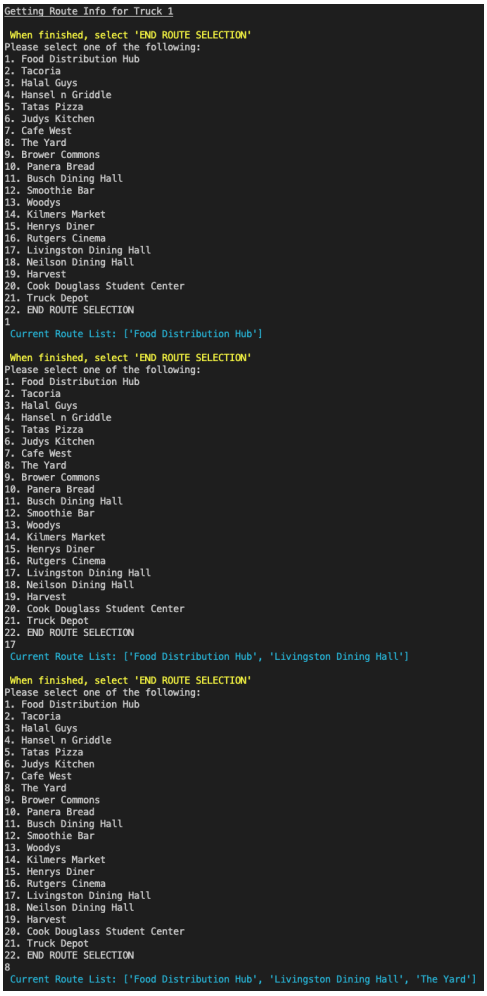


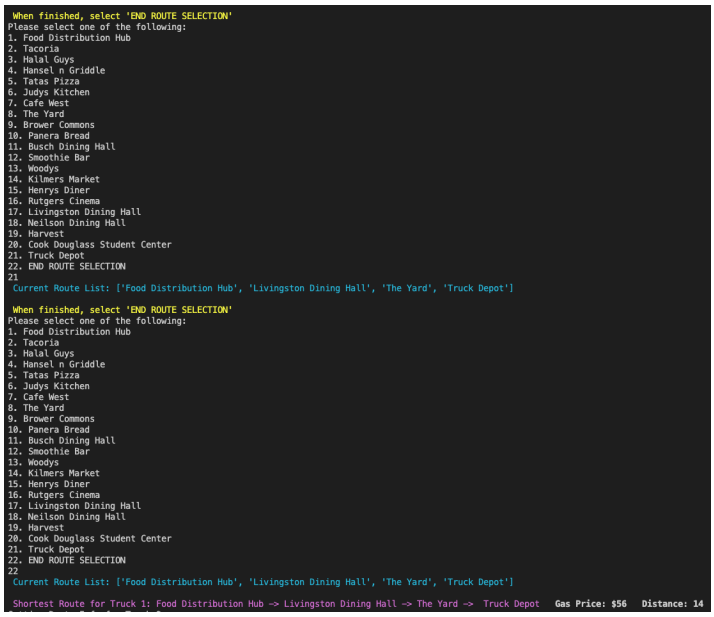
As shown in the picture above, the user interface was able to correctly process the input and add it to the current route list, and then allow the user to continue inputting points until they selected the "END ROUTE SELECTION" option.

Our second test was to test more than one truck in the user interface component:

```
Welcome to the Rutgers Delivery System. Please enter the application password to continue (See README for password)
Password:
 Unable to verify password. See README for correct password. Retry Below:
Password:
 Password Validated. Starting Application...

 How Many Trucks are Available?
3
 How Many Routes are Required?
5
 There are not enough trucks for the required routes. Please make sure the amount of trucks is equal to or more than the required routes.
 Please input valid values of trucks and routes to continue

 How Many Trucks are Available?
5
 How Many Routes are Required?
3
 Sufficient amount of trucks supplied. 2 trucks will not be used.

 What is the current price-per-gallon of gas?
3.00
```

Again, the processes for the user inputting the password, password verification, gas price, truck inputs for numbers available and required, and the verification of those numbers worked as intended and enabled the user input interface to work as intended. Next, the user input interface for multiple routes was tested:





```
Getting Route Info for Truck 2

 When finished, select 'END ROUTE SELECTION'
Please select one of the following:
1. Food Distribution Hub
2. Tacoria
3. Halal Guys
4. Hansel n Griddle
5. Tatas Pizza
6. Judys Kitchen
7. Cafe West
8. The Yard
9. Brower Commons
10. Panera Bread
11. Busch Dining Hall
12. Smoothie Bar
13. Woodys
14. Kilmers Market
15. Henrys Diner
16. Rutgers Cinema
17. Livingston Dining Hall
18. Neilson Dining Hall
19. Harvest
20. Cook Douglass Student Center
21. Truck Depot
22. END ROUTE SELECTION
1
 Current Route List: ['Food Distribution Hub']

 When finished, select 'END ROUTE SELECTION'
Please select one of the following:
1. Food Distribution Hub
2. Tacoria
3. Halal Guys
4. Hansel n Griddle
5. Tatas Pizza
6. Judys Kitchen
7. Cafe West
8. The Yard
9. Brower Commons
10. Panera Bread
11. Busch Dining Hall
12. Smoothie Bar
13. Woodys
14. Kilmers Market
15. Henrys Diner
16. Rutgers Cinema
17. Livingston Dining Hall
18. Neilson Dining Hall
19. Harvest
20. Cook Douglass Student Center
21. Truck Depot
22. END ROUTE SELECTION
19
 Current Route List: ['Food Distribution Hub', 'Harvest']
```

```
 When finished, select 'END ROUTE SELECTION'
Please select one of the following:
1. Food Distribution Hub
2. Tacoria
3. Halal Guys
4. Hansel n Griddle
5. Tatas Pizza
6. Judys Kitchen
7. Cafe West
8. The Yard
9. Brower Commons
10. Panera Bread
11. Busch Dining Hall
12. Smoothie Bar
13. Woodys
14. Kilmers Market
15. Henrys Diner
16. Rutgers Cinema
17. Livingston Dining Hall
18. Neilson Dining Hall
19. Harvest
20. Cook Douglass Student Center
21. Truck Depot
22. END ROUTE SELECTION
21
 Current Route List: ['Food Distribution Hub', 'Harvest', 'Truck Depot']

 When finished, select 'END ROUTE SELECTION'
Please select one of the following:
1. Food Distribution Hub
2. Tacoria
3. Halal Guys
4. Hansel n Griddle
5. Tatas Pizza
6. Judys Kitchen
7. Cafe West
8. The Yard
9. Brower Commons
10. Panera Bread
11. Busch Dining Hall
12. Smoothie Bar
13. Woodys
14. Kilmers Market
15. Henrys Diner
16. Rutgers Cinema
17. Livingston Dining Hall
18. Neilson Dining Hall
19. Harvest
20. Cook Douglass Student Center
21. Truck Depot
22. END ROUTE SELECTION
22
 Current Route List: ['Food Distribution Hub', 'Harvest', 'Truck Depot']

 Shortest Route for Truck 2: Food Distribution Hub -> Harvest ->  Truck Depot   Gas Price: $48   Distance: 12
```

```
Getting Route Info for Truck 3

 When finished, select 'END ROUTE SELECTION'
Please select one of the following:
1. Food Distribution Hub
2. Tacoria
3. Halal Guys
4. Hansel n Griddle
5. Tatas Pizza
6. Judys Kitchen
7. Cafe West
8. The Yard
9. Brower Commons
10. Panera Bread
11. Busch Dining Hall
12. Smoothie Bar
13. Woodys
14. Kilmers Market
15. Henrys Diner
16. Rutgers Cinema
17. Livingston Dining Hall
18. Neilson Dining Hall
19. Harvest
20. Cook Douglass Student Center
21. Truck Depot
22. END ROUTE SELECTION
19
 Current Route List: ['Harvest']

 When finished, select 'END ROUTE SELECTION'
Please select one of the following:
1. Food Distribution Hub
2. Tacoria
3. Halal Guys
4. Hansel n Griddle
5. Tatas Pizza
6. Judys Kitchen
7. Cafe West
8. The Yard
9. Brower Commons
10. Panera Bread
11. Busch Dining Hall
12. Smoothie Bar
13. Woodys
14. Kilmers Market
15. Henrys Diner
16. Rutgers Cinema
17. Livingston Dining Hall
18. Neilson Dining Hall
19. Harvest
20. Cook Douglass Student Center
21. Truck Depot
22. END ROUTE SELECTION
11
 Current Route List: ['Harvest', 'Busch Dining Hall']

 When finished, select 'END ROUTE SELECTION'
Please select one of the following:
1. Food Distribution Hub
2. Tacoria
3. Halal Guys
4. Hansel n Griddle
5. Tatas Pizza
6. Judys Kitchen
7. Cafe West
8. The Yard
9. Brower Commons
10. Panera Bread
11. Busch Dining Hall
12. Smoothie Bar
13. Woodys
14. Kilmers Market
15. Henrys Diner
16. Rutgers Cinema
17. Livingston Dining Hall
18. Neilson Dining Hall
19. Harvest
20. Cook Douglass Student Center
21. Truck Depot
22. END ROUTE SELECTION
21
 Current Route List: ['Harvest', 'Busch Dining Hall', 'Truck Depot']
```

```
 When finished, select 'END ROUTE SELECTION'
Please select one of the following:
1. Food Distribution Hub
2. Tacoria
3. Halal Guys
4. Hansel n Griddle
5. Tatas Pizza
6. Judys Kitchen
7. Cafe West
8. The Yard
9. Brower Commons
10. Panera Bread
11. Busch Dining Hall
12. Smoothie Bar
13. Woodys
14. Kilmers Market
15. Henrys Diner
16. Rutgers Cinema
17. Livingston Dining Hall
18. Neilson Dining Hall
19. Harvest
20. Cook Douglass Student Center
21. Truck Depot
22. END ROUTE SELECTION
22
 Current Route List: ['Harvest', 'Busch Dining Hall', 'Truck Depot']

 Shortest Route for Truck 3: Harvest -> Busch Dining Hall ->  Truck Depot   Gas Price: $64   Distance: 16

Final Routes:
 Shortest Route for Truck 1: Food Distribution Hub -> Livingston Dining Hall -> The Yard ->  Truck Depot   Gas Price: $56   Distance: 14
 Shortest Route for Truck 2: Food Distribution Hub -> Harvest ->  Truck Depot   Gas Price: $48   Distance: 12
 Shortest Route for Truck 3: Harvest -> Busch Dining Hall ->  Truck Depot   Gas Price: $64   Distance: 16
Final Cost: $168.0
Final Distance: 42.0
```

Through this test, we were able to determine that the user input interface incorporates the different modules of user input (password, truck information, and then route information) together correctly, allowing the user once authenticated to specify the number of routes needed, and then determine the pickup and delivery points on each route.

The output interface incorporates the processes for converting the user inputted data into usable information, then computes the route that utilizes the least distance possible in order to minimize the cost of gas. The user sees the output of the correct order of points and then the distance and cost of gas. The output interface correctly displays this information after each route, working as intended and allowing the user to easily identify the route of each truck. Then the interface displays all the routes in a list with each gas price and distance, and then the total price and distance. The output interface also indicated to us that the functions for calculating the shortest route also worked correctly, as we were able to verify that the outputted route was the shortest.

The user input interface, output interface, and modules to determine the shortest route fulfill all of the functional requirements.

The functional requirements of our system were:

- Find ‘optimal route’ for trucks with the minimum distance
- Allow for many pickup and delivery points
- Calculate gas cost for each route and the total price, given the gas price input
- Check whether available trucks meet the intended number of routes

The user input interface allows for many pickup and delivery points, and then allows the user to specify the pickup point and delivery points of each route. The user input interface also checks whether there are enough trucks to meet the intended number of routes, where the user then re-inputs the size of fleet and/or number of routes. The modules to determine the shortest route successfully find the ‘optimal route’ for each truck where the given route has the minimum distance possible. It also calculates the price of gas for each route. The output interface then outputs the ‘optimal’ routes, costs, and distances.

Therefore, after unit testing confirmed to us that each function worked as intended individually, integration testing allowed us to confirm that the user interface, route calculation modules, and output interface each all worked as intended. Our next step in the testing process was to verify that the system as a whole met all the requirements and worked as a whole as intended.

System Testing

The purpose of this section is to check that the software meets all the non-functional requirements that have been agreed within the users and does what it is supposed to do.

The non-functional requirements of our system were

- Reliability of the program
- Security isn't major priority
- Readable data (not just coordinates)
- Easy usability for customers

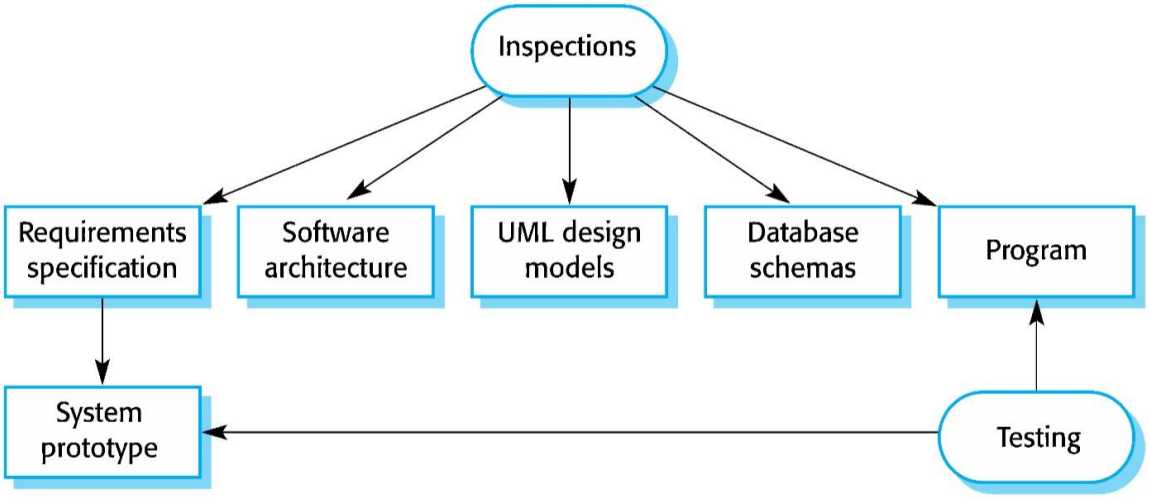
Through our unit and integration testing, we were able to confirm that our program is currently correct, and the tests would be able to check if any later changes would cause defects. The route calculations were also correct, providing reliable data to the users based on the user's truck fleet, pickup points, and delivery points. As security was not a main priority of this system, our simple password verification module and not having a username/password module suffices and meets our requirements. As shown in the integration testing, the outputted data is usable to the user, providing the distances, cost of gas for each route and the total cost, and the routes are provided with the names of the points (instead of the coordinates of the points) to allow for easily used data. The user input interface checked whether the number of trucks were sufficient, and allowed the user to re-input the number of trucks and intended routes if the number was insufficient. The system is very user-friendly and meets the usability requirement, as the user only has to input the correct password, truck fleet size, intended pickup and delivery points in order to receive the data on the shortest route possible. All the calculations are handled on the back-end and are not visible to the user. Therefore, all of our non-functional requirements were met, and our unit and integration tests allowed us to determine this.

Final Conclusion and Evaluation

After conducting the defect and validation testing, in this last sprint, we have to evaluate our entire product and the process of how we got here. The main ways to evaluate this would be using the verification and validation of the software as a whole.

The main way that we got around this was through inspection to catch any bugs or defects earlier on in every single sprint so it would not catch up to us later in the process.

Inspecting the software at each stage to find any anomalies and defects was done using the guide of the figure below: each stage our team discussed possible changes and bugs along the way. This was most helpful with the design and implementation aspect of the software as it clearly told us what kind of product we should be creating, what the software must carry out, what requirements it should meet, what is the most cost efficient and client friendly way, and how to make this program flexible to changes. This is the main advantage of agile methodology since everyone gets to be on the same page and is cohesive.



Now moving onto the evaluation.

To reiterate our use cases:

Use Cases

1. User logins in with credentials
2. System scans databases to check credentials
    a. If login is validated, then user enters application
    b. Otherwise, user must try to validate credentials again
3. User enters how many trucks are available
4. User enters how many routes they have

5. System checks if the amount of trucks available is sufficient for all deliveries
    a. If not, application notifies user and exits (maybe how many deliveries would be possible)
    b. If the amount of trucks are sufficient, the application continues
6. User enters the destinations required to be delivered to for their respective truck
7. Determine the best route for the truck to deliver the packages to the the destination points from the pickup point
8. Calculate the cost of gas for the trucks and the total miles traveled per truck

To reiterate our requirements which we followed throughout this sprint:

Non-Functional Requirements:

- Reliability of the program
- Security isn't major priority
- Readable data (not just coordinates)
- Easy usability for customers

Functional Requirements:

- Find 'optimal route' for trucks with the minimum distance
- Allow for many pickup and delivery points
- Calculate gas cost for each route and the total price, given the gas price input
- Check whether available trucks meet the intended number of routes

Validation

Are we building the right product?

The main prompt of this product was to produce a route planning system that gave the output of the most optimal cost efficient route to the trucks from their pickup point to the destination. Looking back at the integration and unit testing, we can conclude that our program, indeed, is the right product and carries out all the requirements and gives the user the most optimal route for the trucks as the prompt asked us to.

Verification

Are we building the product right?

Throughout the entire process of scrum calls and working on the program we used static verification.

This was a way to verify the correctness of the program. Using the sequence diagram and the use cases, we were able to verify where we could get the errors of the program and in what use cases the program flowed smoothly as we wanted it to. If the program did not carry out the requirements correctly, then we would know that we were not building the program correctly. All the tests: unit, integration and system testing prove to us that we built the product right and that it meets all the requirements.

Our team carried out their roles and responsibilities equally and contributed to the end product since everyone was on the same page and applied agile methodology & the agile manifesto, making it easier on all of us to give a splendid presentation.

Future Planning:

If clients require any updates or changes, we would be more than happy to implement the changes as software keeps evolving and changing. We have built a product that can easily handle any changes and through agile methodology this would make changing and upgrading components easier. Some possible future features include allowing the user the ability to specify the end point, as well as making a route with multiple pickup points.