Dynamic Assignment of Trucks to Delivery Requests

Final Report Document

Group 32
Client & Advisor: Dr. Goce Trajcevski

https://sdmay22-32.sd.ece.iastate.edu/
sdmay22-32@iastate.edu

Revised April 28, 2022

Bernard Fay
Asma Gesalla
Joshua Heroldt
Matthew Medley
Indrajeet Roy
Nolan Slimp

Executive Summary

This document contains the design and implementation of our solution to dynamically assigning trucks to delivery requests with a focus on rerouting trucks in the case of a truck breaking down.

Development Standards & Practices Used

For software development:

- Scrum methodology:
  - https://scrumguides.org/scrum-guide.html
  - https://www.scrum.org/resources/professional-scrum-developer-glossary
- IEEE 610.12, Standard Glossary of Software Engineering Terminology
- IEEE 1540: Software Risk Management

For software testing:

- IEEE 1012: A standard for Software Verification and Validation.
- IEEE 1061: A methodology for establishing quality requirements
- IEEE 1008: Unit testing standard

For working with coordinate systems:

- GPS Coordinates will use the UTM or WGS84 format for representing geolocated points
- Mercator projection and Map matching have no agreed-upon standards, so we will follow conventional projection formulas

Summary of Requirements

- Set of trucks and delivery requests
- For each truck
  - Initial location
  - Delivery location (target)
  - Goods being transported
  - Capacity of the truck (weight of goods that can be carried)
  - Load (amount of goods being carried/transported on the truck)
- Generate route for each truck
- Cater to the dynamic updates:
  - Broken truck at any given time
- Reassign the rest of the trucks from the fleet as a result of the dynamic updates

- UI requirements
  - Desktop application to display routes based on fleet
  - Intuitive design
- Constraints
  - Response time (Seconds to a minute of response time for dynamic updates)
  - Assuming the availability of road network maps and other traffic distribution data (traffic density) -> Needed for any assignment (both initial and dynamic)
  - Economics:
    - Minimize delivery delay as a result of a dynamic update
    - Minimize idle time of trucks

    - Resource requirements
        - Need a server to be running constantly to host the database and requests as well as running the assignment algorithm
        - Visualization tools/frameworks

Applicable Courses from Iowa State University Curriculum

- COM S 228
- COM S 309
- COM S 311
- COM S 363
- S E 319
- S E 329

Skills/Knowledge Acquired Not Taught in Courses

- Angular
- Typescript
- Mapbox

Contents

List of Tables and Figures

Figures

Tables

1 The Team

1.1 Team Members

- Bernard Fay
- Asma Gesalla
- Joshua Heroldt
- Matthew Medley
- Inrdajeet Roy
- Nolan Slimp

1.2 Skill Sets Covered by the Team

- Databases
    - Bernard Fay
    - Matthew Medley
    - Nolan Slimp
- APIs
    - Bernard Fay
    - Joshua Heroldt
    - Matthew Medley
- Web Development
    - Joshua Heroldt
    - Matthew Medley
    - Nolan Slimp
- Inter-process Communication
    - Bernard Fay
    - Asma Gesalla
    - Indrajeet Roy
- Program Efficiency
    - Bernard Fay
    - Joshua Heroldt

2 Introduction

2.1 Acknowledgement

We would like to express our gratitude to Dr. Goce Trajcevski for meeting with us on a biweekly basis and then a weekly basis as the deadline drew closer. His guidance and willingness to assist us in whatever way we needed did not go unnoticed.

2.2 Problem Statement

Develop a system that will enable users to participate in route assignments for delivery trucks. The fundamental algorithm question we are trying to solve is how to assign a truck from a given fleet to a new request, or how to re-assign truck(s) to respond to dynamic changes in traffic or if a truck breaks down. The rest of the tasks will involve implementing the algorithmic solutions, the user interface, and integrating map data with request data to generate routes.

Truck transportation is primarily used when large packages need to be transported from one location to another. Trucks can be used to transport goods to retailers, personal items to new locations, cars to dealers, or a number of other necessary reasons for moving items. It is an essential to almost every aspect of industry and life in the entire world nowadays. Approximately 70% of all shipments in the United States are shipped by truck. Since trucks play a large role in construction, this also meant faster and more efficient construction could take place.

Any successful business depends on customer satisfaction, and what a customer will look at when it comes to delivery services is speed, a customer would love to receive their orders on the expected time. That's what led our team to focus on solving common issues about truck transportation. We aimed to solve the issue of broken trucks and try to find the closest truck that can take care of the load.

2.3 Requirements & Constraints

In the system, there will be a set of trucks and delivery requests that base all of our constraints. Each truck will have an initial location, delivery location, goods being transported, capacity of the truck, and current load. Then, the system must generate the optimal route for each truck. It needs to update if any truck breaks down on route at any given time (constraint). Based on these constraints, dynamic updates will be made to reassign the rest of the trucks from a given fleet. These dynamic updates must be made in less than a minute response time (constraint). The user interface will consist of a dispatcher web application to display routes to users. The system will also be efficient, and must minimize route time for dynamic updates, idle time of a truck, and initial assignments (constraint). This project assumes we have access to road

network maps and other traffic information; we will rely on this data and it is necessary to assign trucks (constraint).

- Set of trucks and delivery requests
- For each truck
    - Initial location
    - Delivery location (target)
    - Goods being transported
    - Capacity of the truck (weight of goods that can be carried)
    - Load (amount of goods being carried/transported on the truck)
- Generate route for each truck
- Cater to the dynamic updates:
    - Broken truck at any given time
- Reassign the rest of the trucks from the fleet as a result of the dynamic updates
- UI requirements
    - Dispatcher (Desktop) UI
    - Intuitive design for the UI
- Constraints
    - Response time (Seconds to a minute of response time for dynamic updates)
    - Assuming the availability of road network maps and other traffic distribution data (traffic density) -> Needed for any assignment (both initial and dynamic)
    - Economics:
        - Minimize delivery delay as a result of a dynamic update
        - Minimize idle time of trucks
    - Resource requirements
        - Need a server to be running constantly to host the database and requests as well as running the assignment algorithm
        - Android mobile device
        - Visualization tools/frameworks

2.4 Engineering Standards

For software development:

- Scrum methodology:
    - https://scrumguides.org/scrum-guide.html
    - https://www.scrum.org/resources/professional-scrum-developer-glossary
- IEEE 610.12, Standard Glossary of Software Engineering Terminology
- IEEE 1540: Software Risk Management

For software testing:

- IEEE 1012: A standard for Software Verification and Validation.
- IEEE 1061: A methodology for establishing quality requirements
- IEEE 1008: Unit testing standard

For working with coordinate systems:

- GPS Coordinates will use the UTM or WGS84 format for representing geolocated points

- Mercator projection and Map matching have no agreed-upon standards, so we will follow conventional projection formulas

2.5 Intended Users and Uses

The primary beneficiaries of our project are customers, truck dispatchers, and truck drivers. While not directly used by the customers, the project will have the general impact of ensuring timely deliveries. The project will be more directly used by dispatchers as it will aid them in deciding initial routes for trucks as well as making decisions and adjustments in the case of changing circumstances (traffic, new orders, truck breakdowns, etc.). This will benefit the dispatcher by reducing the stress of unpredictable circumstances and having to make quick decisions. Lastly, the project will be used by truck drivers to receive their assignments and any changes that occur throughout the day due to traffic, new orders, breakdowns, etc. This will benefit truck drivers by reducing the amount of time they spend making deliveries by optimizing routes, reducing waiting times for updated assignments, and minimizing the amount of time spent making deliveries.

2.5.1 Base Use Case

The base use case is formulated on the assumptions that all actors (Truck dispatchers and customers) are involved, the order route is between the cargo origin point and a single destination point (in comparison to multiple destination points) and external factors such as traffic, vehicle malfunction are not present.

A customer's input order is allocated to a dispatcher via the allocation algorithm and displayed based on which truck it was assigned to. The truck route from the order origin pick up point to the destination point will be determined by the routing algorithm and done initially for all trucks in a given fleet. Post order delivery to destination, the truck will move back to the warehouse and await new orders.

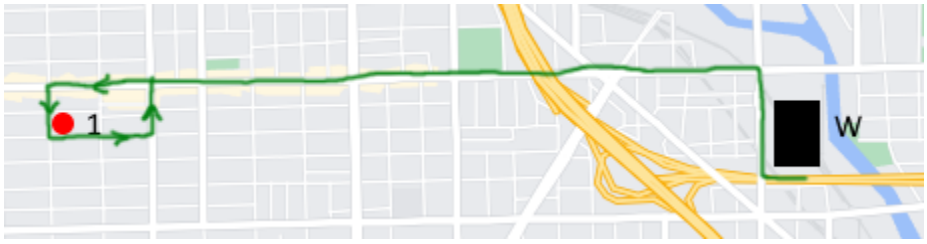

Figure 2.5.1.1. Single Destination Use Case

Given warehouse W and order 1, the algorithm will generate the depicted route and present it to the dispatcher. The truck will then take the route starting at the warehouse and go to the order location.

2.5.2 Multiple Destinations Use Case

The multiple destination use case is formulated on the assumptions that all actors (Truck dispatchers and customers) are involved, the order route is between the cargo origin point and multiple destination points and external factors such as traffic, vehicle malfunction are not present.

A customer's input order is allocated to a dispatcher via the allocation algorithm. The truck driver's route from the warehouse to each of the delivery locations will be determined by the routing algorithm and displayed to the dispatcher. Post order delivery to the final destination point the customer will be notified that the order has been successfully delivered to the destination point, as per the customer's order input.

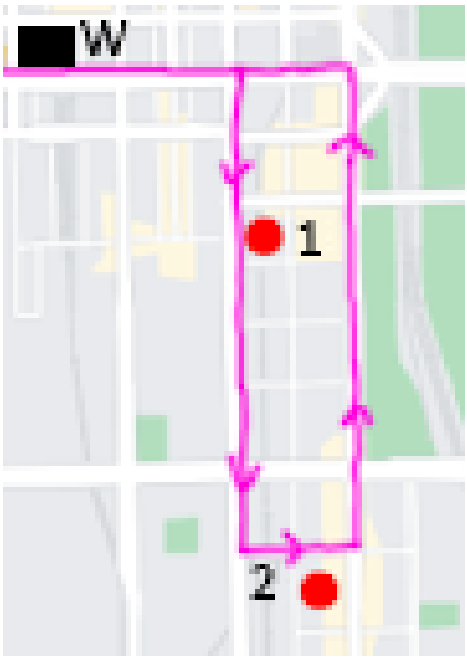

Figure 2.5.2.1. Multiple Destinations Use Case

Given warehouse W, orders 1 and 2 at the locations depicted, and one truck, the algorithm would output the above route given the locations of the orders. The route with all points will be displayed to the dispatcher.

2.5.3 Multiple Destinations with Multiple Routes Use Case

The multiple destination use case is formulated on the assumptions that all actors (Truck dispatchers and customers) are involved, the order route is between the cargo origin point and multiple destination points and external factors such as traffic, vehicle malfunction are not present.

A customer's input order is allocated to a dispatcher via the allocation algorithm. The truck driver's route from the warehouse to each of the delivery locations will be determined by the routing algorithm and displayed to the truck dispatcher. Post order delivery to the final destination point the customer will be notified that the order has been successfully delivered to the destination point, as per the customer's order input.

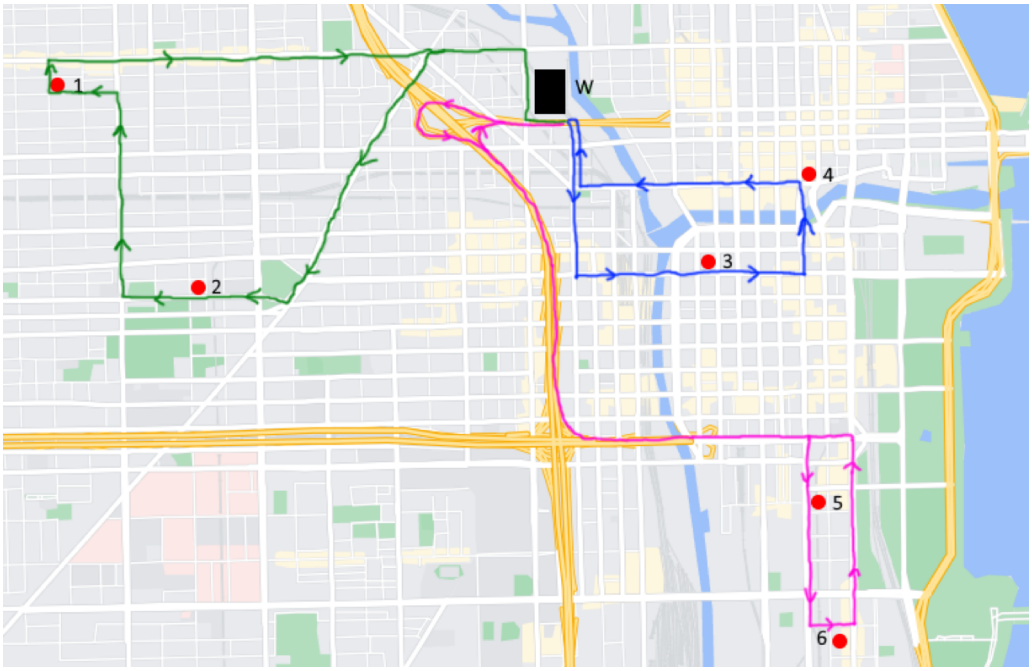

Figure 2.5.3.1. Multiple Destinations with Multiple Routes Use Case

Given warehouse W, orders 1, 2, 3, 4, 5, and 6 at the locations depicted, and three trucks, the algorithm would output three routes, green, blue, and pink, given the proximity of the orders, expected delivery times, and load balancing. Each of the three trucks would be assigned to one of the routes for the day, starting and ending at the warehouse.

2.5.4 Route Reallocation Use Case

The route reallocation use case is formulated on the assumptions that all actors (Truck dispatchers and customers) are involved, the order route is between the cargo origin point and destination points and external route blocking or route inefficiency factors such as traffic, road construction, route obstacles are present.

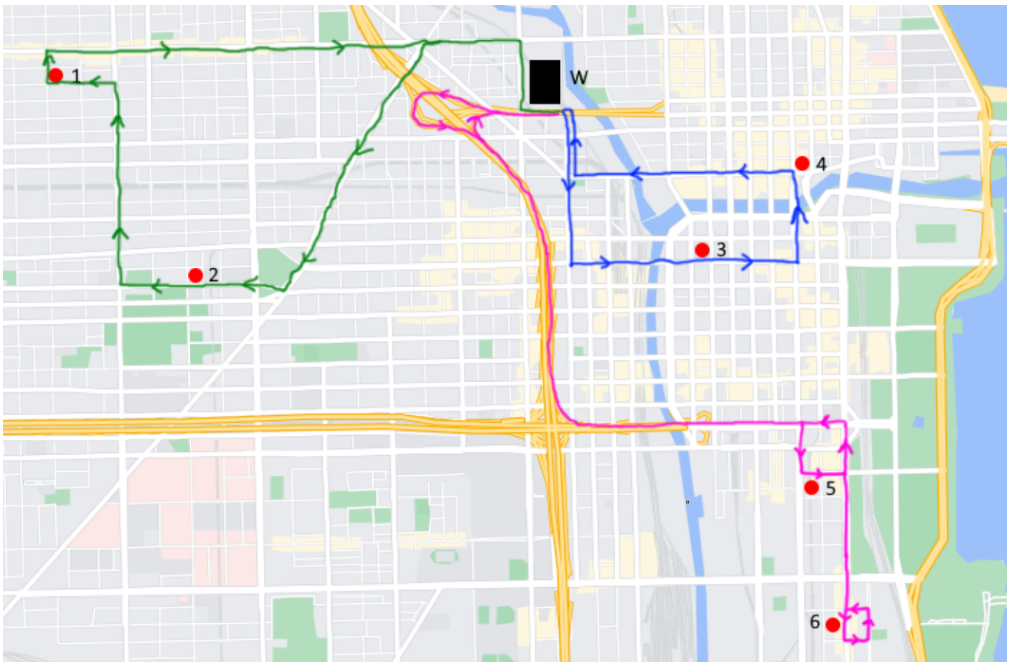

Figure 2.5.4.1. Route Recalculation Use Case

This example is similar to the example presented in the previous use case. However, upon delivering order 5, a backup is reported along the initial planned route to deliver order 6. The route allocation algorithm recalculates the route to ensure timely delivery of the customer's order and reduce the amount of time spent delivering orders for the truck driver. This recalculated route will be displayed to the dispatcher as the update occurs.

3 Project Plan

3.1 Project Management & Tracking Procedures

We are using SCRUM as a framework for our project's development and weekly lifecycle. Following SCRUM and the agile methodologies, met weekly to complete status update. Furthermore, each week after our client meeting, the team went over issues/impediments from the previous week and addressed them. Development and planning work is expected to be expressed clearly in the weekly status reports and in standup after each client meeting, and the work for development should be reflected in a story on the team's Trello board. We chose SCRUM because it is the most familiar project management style, and fits closely with our goals and objectives as developers to complete the project and facilitate communication.

Our project is using Gitlab for version control. We also are using Trello to help track work, progress, and aid in our standup meetings. Our primary means of communication is Discord, and secondarily we are using email when a group member is needed.

3.2 Task Decomposition

The tasks to be completed were decomposed in the following manner:

Task	**Description**
Implement Visualization Tool Front-End	Create a user interface for dispatchers and drivers to view routes and stops
Develop UI for Web App	Create a user interface for dispatchers to view orders and routes, communicate with drivers, and input changes manually
Develop REST API microservices	Design and implement application functions into multiple microservices which communicate with the frontend, db and external services.
Setup application DB	Setup db configs and communication with the API.
Setup application server	Setup server to host API and config changes.
Final Application Testing	Testing individual components of the web app, mobile application, and the microservices.

Table 3.2.1. Task Decomposition

Subtasks

Implement Visualization Tool Front-End

- Display external map (Mapbox)
- Overlay routes (based on initial coordinates)
- Overlay stops on routes (different icons for warehouses and stop locations)

Develop UI for Web App

- Create home page
- Create visualization page
- Create communication page
- Create order overview page
- Create route update/modify page

Develop API microservices

- Create communication service
- Create truck allocation service and route allocation service
- Create user account (login, registration, settings) service
- Create user-order service. (New user order)
- Create user order tracking service

Setup application DB

- Setup db connection to microservices
- Setup db configs (SQL dialect, security configs, data handling configs)

Setup application server

- Setup server to host API services
- Setup server configs

Final Application Testing

- Test the web app and the mobile application for navigation between views and other inconsistencies.
- Test the web app for correctly receiving data from the backend
- Test the algorithm using real time metrics.
- Test the individual microservices for their respective functions.
- Setup a testing environment to see if all the components communicate with each other flawlessly and achieve desired results.

3.3 Project Milestones, Metrics, and Evaluation Criteria

- Metrics of interest:
    - UI and visualization tool usability
    - Algorithm update speed (in response to dynamic changes)
    - General algorithm efficiency
- Evaluation criteria:

    - UI and visualization tool usability
        - Create questionnaire for users
        - Responsiveness of UI
        - Accuracy of visualization tool
    - Algorithm update speed
        - Time for changes to be returned
    - General algorithm efficiency
        - Compare miles traveled, time driving, packages delivered between a brute force algorithm and our implementation
    - Algorithm scalability
        - Ease of adding new drivers or dispatchers (measured in change in efficiency values and update speed as more drivers/dispatchers are added)
- Milestones:
    - Baseline functional UI
    - Alpha UI (first round of user feedback)
        - UI responds to input in under 500 ms
        - Visualization tool at least 75% accurate
    - Beta UI (second round of user feedback)
        - UI responds to input in under 250 ms
        - Visualization tool at least 90% accurate
    - Polished UI
        - UI responds to input in under 100 ms
        - Visualization tool at least 98% accurate
    - Algorithm speed improvements
        - Algorithm can make updates in under 20 seconds
        - Algorithm can make updates in under 10 seconds
        - Algorithm can make updates in under 5 seconds
        - Algorithm can make updates in under 1 second
    - Algorithm efficiency is better than brute force approach
        - Algorithm is 15% more efficient than brute force approach
        - Algorithm is 25% more efficient than brute force approach
        - Algorithm is 50% more efficient than brute force approach
    - Scaling the input space provides minimal decrease in performance
        - Doubling the number of drivers/dispatchers reduces miles traveled and time driving by 10%
        - Doubling the number of drivers/dispatchers reduces miles traveled and time driving by 25%
        - Doubling the number of drivers/dispatchers reduces miles traveled and time driving by 50%

3.4 Project Timeline

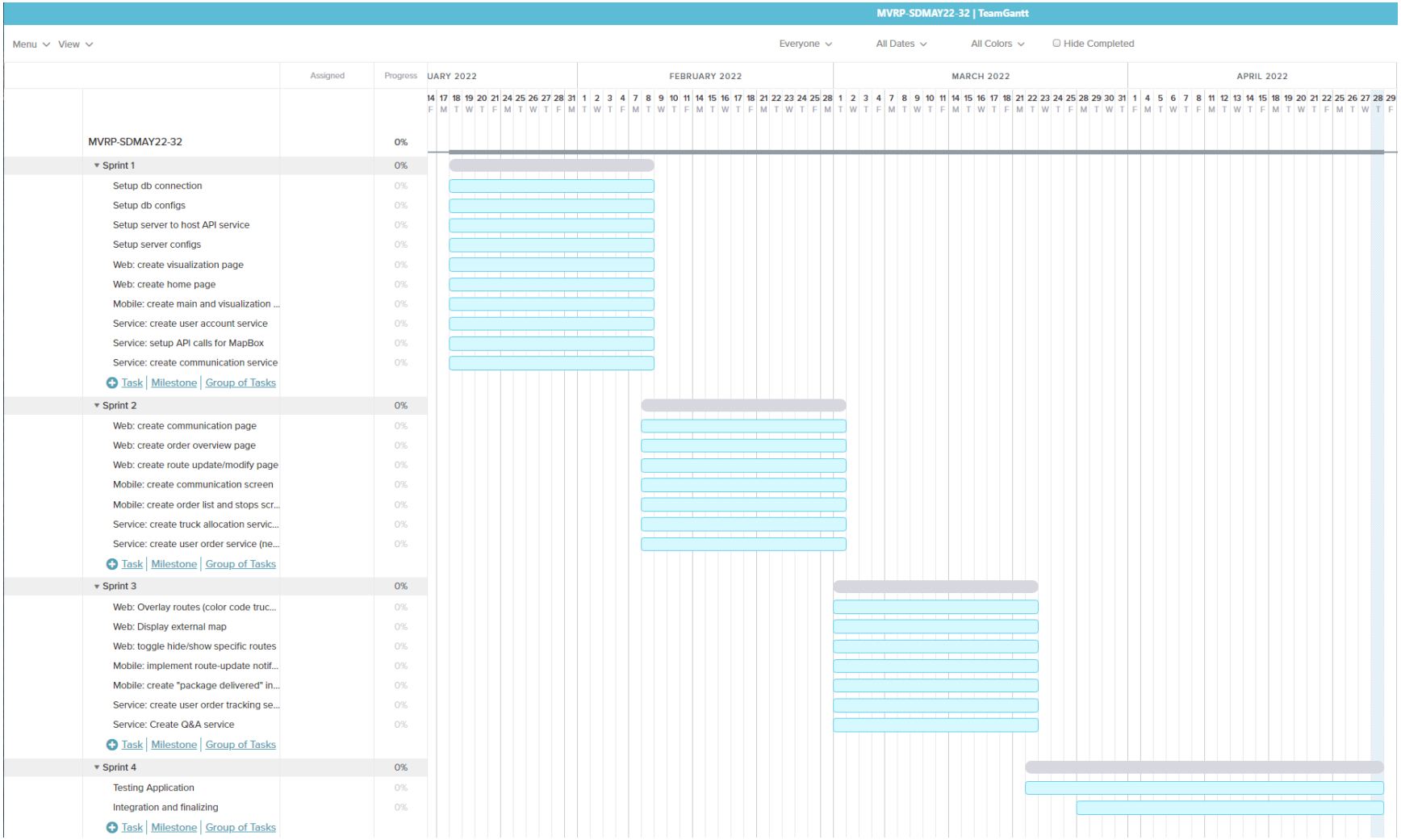

Figure 3.4.1. Gantt Chart

- Each Sprint will be 3 weeks with weekly standup meetings
- Integration testing and improvements will be made after the completion of this development schedule
- Sprint 1: January 18th - February 8th
- Sprint 2: February 8th - March 1st
- Sprint 3: March 1st - March 22nd
- Sprint 4: March 22nd - April 28th

3.5 Risks and Risk Management/Mitigation

Risks for each task:

- Implement Visualization Tool on Front-End
    - The biggest risk here is that we struggle to correctly implement the Vehicle Routing algorithm into the application, or that it takes longer than originally anticipated. This could take some time to re-plan and evaluate how to handle the situation. However, we have team members that are confident in figuring out the algorithm so this is not likely to happen. Risk probability: 0.3

- Develop UI for Web App
    - The biggest risk here is correctly retrieving data from the backend. This shouldn't be a big issue because we have members experienced of Angular and React.
        - Risk probability: 0.3
    - Another risk is that we simply run into defects in the UI that cause us to re-plan. Even with team members experienced in UI development for a particular framework, weird defects can always arise, though most of them won't take long to solve.
        - Risk probability: 0.2

- Develop REST API microservices
    - Since backend development is the bridge between frontend and the DB, the biggest risk/challenge is that frontend/backend communication or backend/DB communication is not working. This would set the team back some time to focus on the issue and get it resolved, may require some re-planning.
        - Risk probability: 0.4

- Setup application DB
    - The only risk here is the decision on whether or not to use a SQL database or no-SQL. This decision depends on the experience of the team and will require time and effort to consider. We have members that are experienced working with SQL, so this should likely not be an issue.
        - Risk probability: 0.1

- Setup application server
    - There's hardly any risk to just setting up the application server. The biggest risk is that the server can't run for whatever reason, which may require us to take a deeper look simply using online resources or discussion.
        - Risk probability: 0.1

3.6 Personnel Effort Requirements

Task	Description	Time (person-hours)
Implement Visualization Tool Front-End	Create a user interface for dispatchers and drivers to view routes and stops	70
Develop REST API microservices	Design and implement application functions into multiple microservices which communicate with the frontend, db and external services.	40
Setup application DB	Setup db configs and communication with the API.	5
Setup application server	Setup server to host API and config changes.	5
Development	Working on the actual algorithms for the project	80
Learn new language syntax and framework	This depends on the individual experience with the chosen language or framework that has been used.	50
Debugging	Fixing final errors before testing	15
Final application testing	Testing individual components of the web app, mobile application, and the microservices.	40

Table 3.6.1. Task-Effort Decomposition

3.7 Other Resource Requirements

No other resources were required due to the software-centric nature of this project.

4 Design

4.1 Design Context

4.1.1 Broader Context

The broader context is situated in the domain of transportation and delivery. Specifically, we consider a fleet of delivery trucks, a set of orders for goods from stores with given locations and a set of warehouses where those goods are stored and loaded into the trucks. Given an information about a road network (map) with corresponding distances/travel times and the capacity of load for each truck in the fleet, we

A. Develop novel solutions to the problem of having one of the trucks break down(re-distributing its load)

B. Implement an interactive system that can manage users and assignments.

Relevant considerations related to our project:

Area	Description	Examples
Public health, safety, and welfare	This project uses algorithmic efficiency to benefit the public health by cutting down the time trucks and other fleet vehicles are on the road.	Tries to optimize the efficient delivery of goods(welfare). However, the findings can be indirectly used in public health for routing ambulances. Additionally with efficiently routing trucks we can lessen the probability of trucks coming into contact with pedestrians so public safety will increase.
Global, cultural, and social	Countries that rely on heavy amounts of consumerism will be allowed to continue with this practice more through the use of our product.	Our project is not affected by any specific societal context such as nationality, ethnicity and so on. However, it does reflect an impact on the efficiency of transportation and delivery.

Environmental	This project will have a deal of impact on any industry that deals with the production of gasoline or diesel fuel. This is due to the nature of efficiency that our project aims to create for a fleet of trucks.	Minimizing fuel consumption and the emission of a fleet of trucks. This will allow for less greenhouse gasses overall to be released companywide for whoever utilizes our product.
Economic	Economics benefits from our project can fall under two categories. The first is the savings on fuel cost that businesses can experience due to our product. The second is the maximization of profits that businesses can generate from their current fleet.	System implementation will be deployable either on a standard desktop with minimum installation overhead. This product will allow any company that uses it to optimize costs of delivery and balance it with reassignment of deliveries.

Table 4.1.1.1. Responsibility Considerations

4.1.2 User Needs

- Warehouses/Dispatchers
    - Each warehouse/dispatcher needs to be able to summarize all the requests from the customers and determine the assignment of trucks to delivery locations so that they can keep track of and be on top of orders that are coming in from customers.

4.1.3 Prior Work/Solutions

The VRP has been written about a lot. Broad nature or specific characteristics of the problem. Previous research includes that done by Nasser A. El-Sherbeny [4], Wang, et. al. [5], Cappanera, Requejo, and Scuttelà [3], and Bektas [2].

In his research, El-Sherbeny focused on the various exact methods, heuristics, and metaheuristics that can be used to solve a VRP with time windows (VRPTW) [4]. While he investigates a large variety of methods for finding an optimal solution to a VRPTW, the methods are approached from a theoretical perspective rather than a practical one. As a result, the true optimality of the methods presented may not be accurate when transitioned to a real-world application.

In their research, Wang, et. al. expand upon the VRPTW problem by including simultaneous delivery and pickup and optimizing their solution based on a 5 part multiobjective function (MO-VRPSDPTW) [5]. The two methods they focus on are local search and a memetic

algorithm. While the research comes from a theoretical perspective, they openly acknowledge that the optimality of the two methods may not translate well when implemented in a practical solution.

In a variant of the VRP, Cappanera, Requejo, and Scuttelà focus on the skill VRP, an extension of the traditional VRP that focuses on delivering human services instead of goods [3]. What makes their research interesting is their specific focus on situations where one individual does not have all of the skills necessary to complete a job. As a result, multiple “deliveries” must be made to the same customer.

Lastly, Bektas examines the existing approaches to solving the Multiple Traveling Salesman problem [2]. Based on the limited research in this area and the complexity of implementing such solutions, this approach to our problem was not the best choice for our application given our time constraints. Additionally, rerouting is not something that was discussed in this paper.

In comparison, what separates this project is the setting that it considers, which is reacting to the breakdown of a truck and properly executing the reassignment of the routes to the rest of the fleet so that:

A. The goods from the broken truck(s) are delivered to their destinations, and
B. It is done in an optimal manner.

As a result, the focus of the above research on the VRPTW problem does not address the same problem as we seek to address. While many of the aspects are shared between our project and previous research, the end goals diverge greatly when problem constraints are considered. This proves to be an advantage to us as customer-specific time windows for delivery increase the difficulty of finding an optimal solution. Additionally, our focus on the situation where a truck breaks down and routes need to be recalculated part way through is not addressed in any of the above research. This is a shortcoming of the above research and will likely prove to be a disadvantage to us if the solution isn’t immediately obvious.

4.1.4 Technical Capacity

There are three kinds of novelties in this project:

1. Algorithmic: We will solve the very specific problem of reassignment of trucks. This is being done through an existing application called Mapbox. Mapbox will be performing a MultiVehicle Routing Algorithm that we can use for the assignment of trucks.

2. System-wide: We will develop, implement, and deliver a system for managing assignments of trucks to delivery locations which can be accessed and used by all kinds of entity classes that participate in this scenario(customers and warehouses). Across our project, we will do integration testing to ensure all components are functioning properly.

3. Technical complexity: Defining proper test cases for an “optimal route” and evaluation procedures. Our project also focuses on routing trucks with a capacity constraint, and the potential for vehicles to break. This creates additional complexity as a rerouting will have to occur for broken vehicles. Additionally, a new vehicle must have the capacity to be able to assist a broken vehicle with deliveries.

4.2 Design Exploration

4.2.1 Design Decisions

Use the use cases to see the initial system architecture.

1. Location - Ames
2. Traffic Density and map - handled by Mapbox
3. DBMS - MySQL
4. Front end Framework - Angular
5. Backend communication - Spring
6. Database Management System - MySQL Workbench

4.2.2 Decision-Making and Trade-Off

Because of the relational nature of MySQL workbench and its ability to be used easily on backend systems with the use of its specialized drivers we are selecting MySQL workbench for this project. It also has the added benefit of being able to be run and accessed from multiple different systems without having data loss. Additionally, this project does not require anything more complex. Should the complexity increase in the future, we will re-evaluate this decision

4.3 Proposed Design

4.3.1 Design Visual and Description

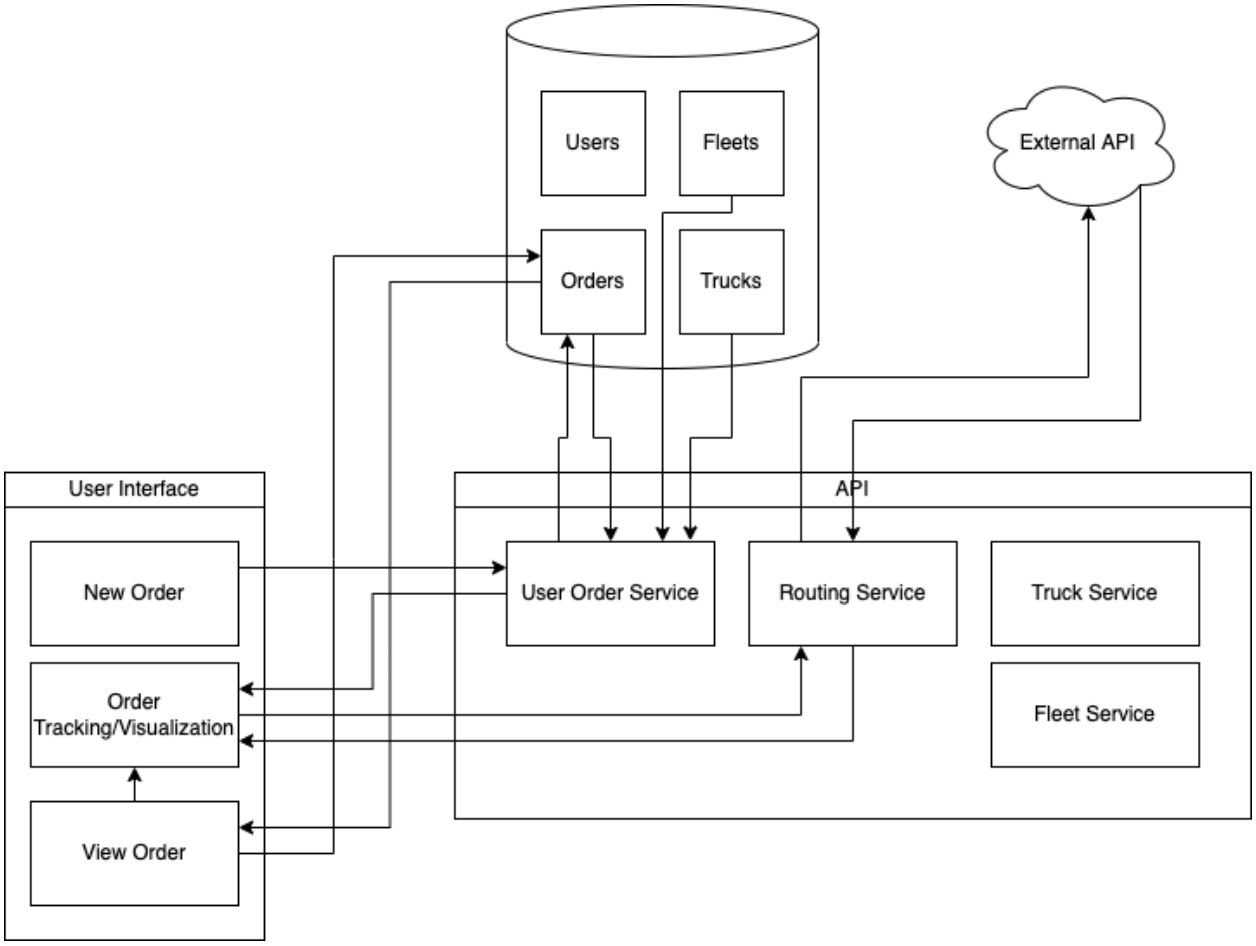

Figure 4.3.1.1. System Architecture

Components

- User interface (New Order, Order Tracking/Visualization, View Order)
- DB (User, Truck, Fleet and Order tables)
- API (User Order Service, Routing Service, Truck Service, Fleet Service)
- External API services

1. A new order is placed via the UI New order page.
1.1 Frontend call to API to change address to Lat, long
1.1 Order is saved to DB via CRUD operation.

2. User orders can be viewed via the UI View Order page.
2.1 Frontend call to API to perform CRUD operations and fetch all user orders from the DB.

3. User clicks on a displayed order to go to the order tracking/visualization page.
3.1 Frontend call to API to get the number of trucks in a fleet.
3.2 Frontend call to API to get the Lat, long coordinates per truck for the stops on its route.
3.3 Truck routes are calculated and displayed on the visualization page.
3.4 Once every route is calculated they are put into a JSON object and sent to the API.

4. When a truck breaks down, the API calculates and returns locations of all trucks including the broken truck.
4.1 Frontend recalls the optimization and navigation API using a dummy route, including the location of the broken truck to find out which routes are optimal (also using load).
4.2 Truck routes are updated and displayed on the visualization page.

4.3.2 Functionality

Functional requirements:

1. Truck drivers should pick-up locations.
2. Truck drivers should be able to deliver to the picked location.
3. Find the nearest truck in case of any breaks.
4. Find the set of closest trucks that can take care of the load that the truck has
5. Set of orders and trucks w/ given capacity.

Non-functional requirements:

1. App should have access to all truck drivers databases.
2. App should have enough information about all warehouses locations, type of loads and capacity.

Our design functions by taking the orders in the system and the associated delivery locations (red dots in the below figures) and generating an optimal set of routes (denoted by green, pink, orange, and blue lines in the below figures) based on the available number of trucks and warehouse locations (denoted by gray boxes in the below figures). Routes will start and end at the same warehouse and trucks will only deliver goods from a single warehouse. The result of this step is shown in Figure 4.3.2.1. Once the routes have been decided and assigned, the truck drivers will be notified via the UI system of their route for that day. In our specific case, we are addressing the instance in which a truck breaks down. Suppose the pink truck breaks down at the location depicted in Figure 4.3.2.2. The UI system will be used to notify the central system that the pink truck has broken down Based on the locations of the other trucks, their load

capacity and the remaining delivery locations, two trucks are assigned to take over the remaining deliveries for the pink truck and their routes are recalculated. In this instance, the blue and orange trucks are identified as the optimal choices and new routes for these two trucks are generated as depicted in Figure 4.3.2.3. The orange and blue truck drivers are then notified of their updated routes via the UI system. This system meets all of the listed functional and non-functional requirements sufficiently.

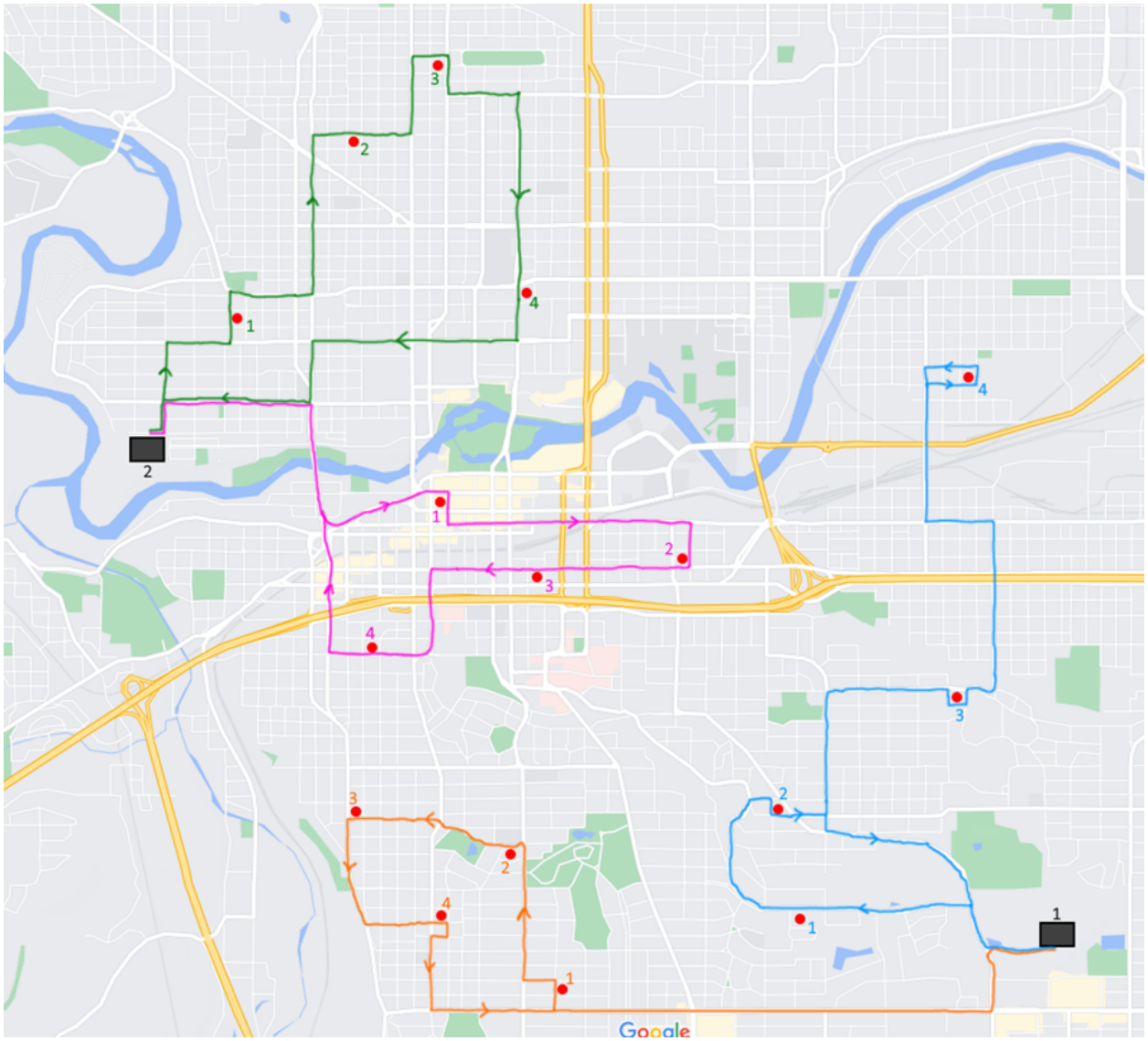

Figure 4.3.2.1. Initial Routes Calculated by Algorithm

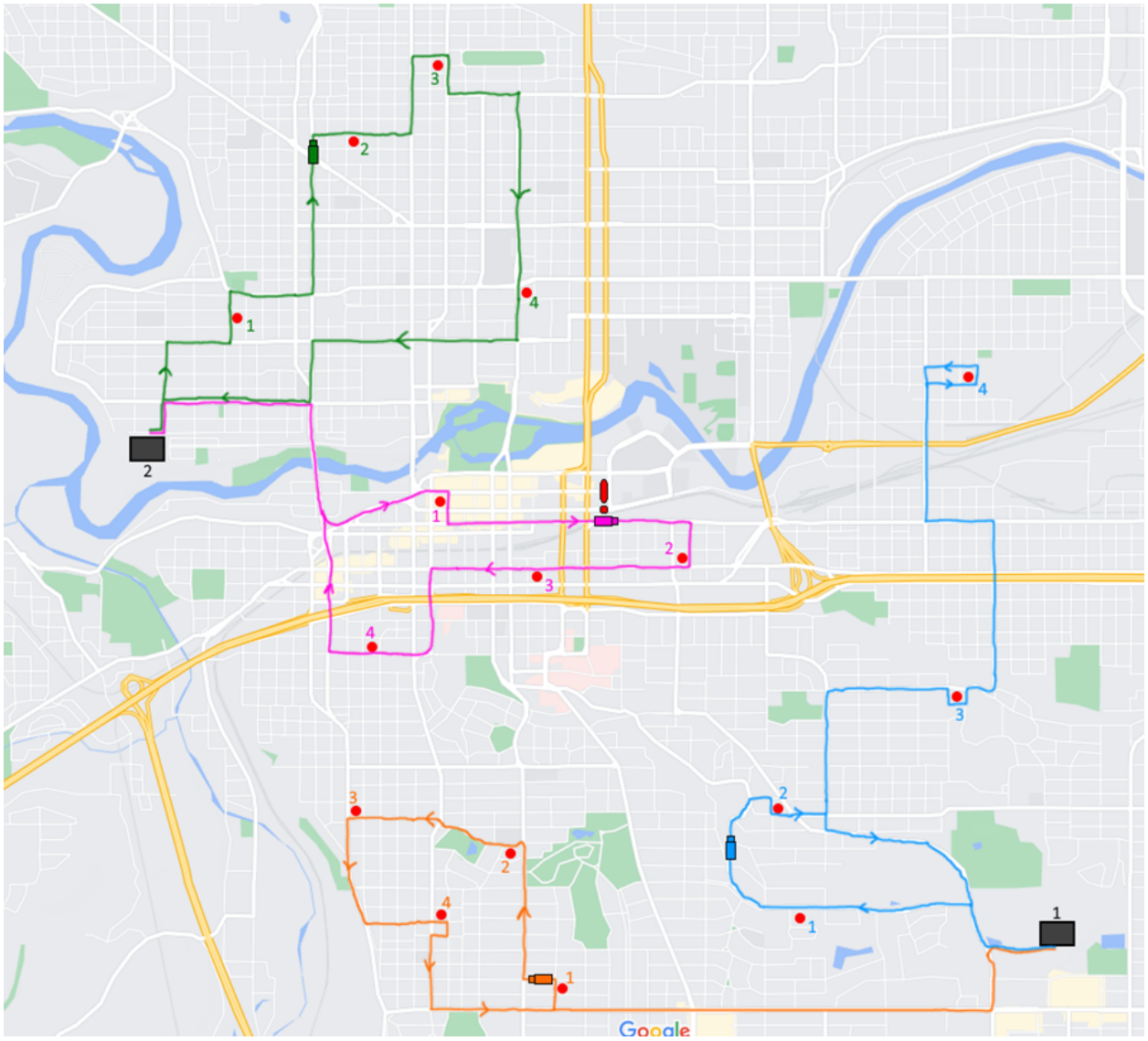

Figure 4.3.2.2. Pink Truck Breaks Down

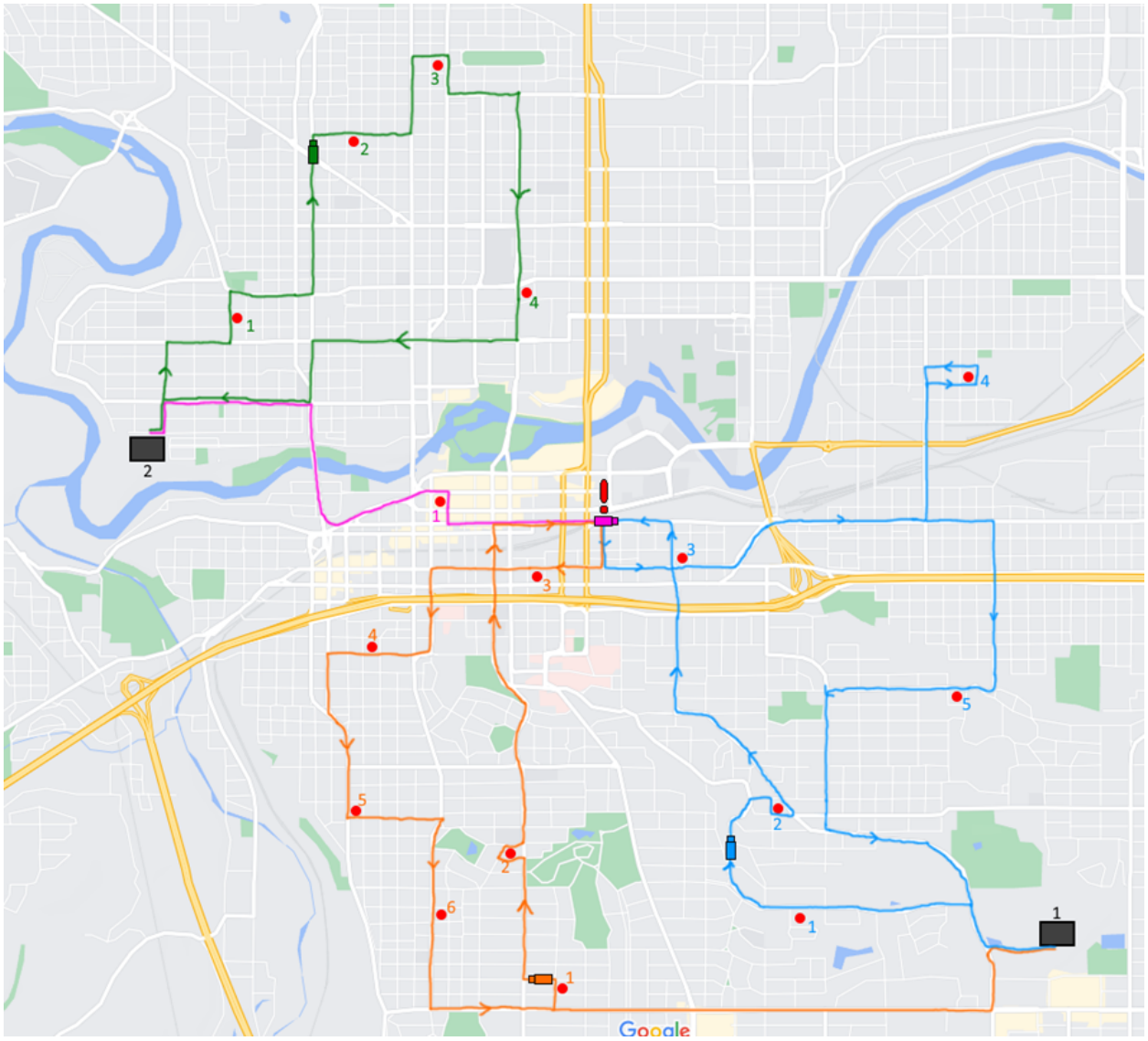

Figure 4.3.2.3. Load Divided Between Blue and Orange Trucks. Routes Recalculated Accordingly.

4.3.3 Areas of Concern and Development

Primary concerns for delivering a product/system that addresses requirements and meets user and client needs:

1. For our current design, we have a better sense of how we want to structure our backend services as opposed to our frontend UIs. This lack of description in the UI design can raise a possible concern because it can leave multiple interpretations to team members, making the project structure less organized. In the upcoming week, this issue will be addressed by the team.

2. Another possible concern is the usage of Mapbox external API. Mapbox will be crucial for our applications success and is needed for visualization and vehicle routing.

Immediate plans for developing the solution to address those concerns:

1. We plan to hold multiple team meetings to tackle this concern, so everyone in the team could get a single and clear understanding of how the UI design should be implemented, so it is not open to interpretation.
2. We plan to tackle this problem by collectively sharing our understandings of the Mapbox APIs. We plan to use widely available resources on the internet to educate ourselves about the API and use tutorials to get a basic simulation working as soon as possible.

4.4 Design Analysis

Our design underwent significant changes due to a variety of factors. The scope of the project was continuously reduced throughout the semester as a result of a team member dropping the course, lack of communication and contribution from other team members, and unforeseen difficulties encountered in implementing our initial design.

The changes changes to our initial design are as follows:

- The initial design included a mobile application that would also interface with the backend to provide updates to truck drivers. However, one of the team members who was assigned to focus on the development of that portion of the project dropped the course shortly after it started. After discussing the situation with our client/adviser, we decided to remove that aspect of the design and focus on developing a web application and backend.
- The initial design was intended to be hosted on a university server. However, lack of communication from the team members assigned to that aspect of the project at the beginning of the semester required us to reevaluate and shift the hosting of the frontend and backend onto the same machine for demonstration purposes.
- A similar lack of communication significantly delayed backend development. As a result, we had to reduce the number of use cases for our project. The initial design of having multiple warehouses and considering weight capacity for the trucks had to be cut in order to have a demo-able project by the end of the semester.
- Lastly, we discovered early on that accessing Mapbox requires a special key. Since the frontend already had a key for developing the UI, all interaction with Mapbox were shifted to the frontend, greatly shifting the work distribution from being backend-heavy to being frontend heavy.

4.5 Development Process

For the design process we followed a scrum methodology, which is an offshoot of agile that focuses on team interaction and sprint planning. This allowed for an iterative approach to our development process where we were able to change and adjust to new challenges as they presented themselves in the sprints. This was easier to do since the scope of the project was reduced from the planning phase of the project. This meant that changes to communication methods and full stack decisions could be made more easily as fewer factors needed to be considered by the team when talking about adjustments to the project implementation. The use of scrum and agile allowed for these changes to be made dynamically and quickly.

4.6 Design Plan

The following design plan does not entirely encompass the entire structure of the project architecture, but rather abstracts the various design decisions that were made into components that specific teams of people worked on.

Algorithms:
What we have done:

- Researched how to properly optimize initial route allocation
- Researched possible ways to reallocate trucks
- Decided on existing implementations for algorithmically assigning initial truck routes
- Decided on how to implement truck reallocation
- Implemented both algorithms in program code

Use-case/Requirement fulfillment:

- Base use case
- Multiple destination use case
- Multiple destination with multiple route use case
- Route reallocation use case

Backend:
What we have done:

- Created endpoints for the following types of data:
    - Trucks
    - Routes
    - Fleets
    - Orders
    - Users
- Set up initial infrastructure and database
- Created the interface between the frontend and backend

Use-case/Requirement fulfillment:

- Data for each truck
- Set of trucks and delivery requests
- Routes per truck
- Decide on broken trucks
- Reassign trucks

Frontend:

What we have done:

- Created visualization page
- Created order review page
- Created order submission page
- Utilized mapbox API to initially route trucks
- Formatted routes for backend

Use-case/Requirement fulfillment:

- Base use case
- Multiple destination use case
- Multiple destination with multiple route use case
- Route reallocation use case

5 Testing

For our project we aim to test a multitude of different scenarios that connect with our use cases and requirements. This covers not only the database and the external APIs, but also the backend connectivity as a whole. These cases allow us to be sure that as a whole the backend for the system is functioning properly and is able to run the algorithm and generate routes from given data both from the APIs and the database. While these backend unit and system tests are occurring, the frontend web application will also be tested. These tests align with use cases regarding the various types of users that our program can have and how they would go about using the program.

5.1 Testing Implementation Process

The testing process our group followed was per implementation. Important methods took priority, and unit tests were written for breaking and allocating trucks. Unit tests were primarily written before our group did integration between the front-end web application and SpringBoot back-end service. Many of our methods were also tested manually by using the tool Postman to create API calls from a local running application.

5.2 Unit Testing

The primary units that exist within our application's backend are contained within the routing interface. The units in the routing interface were individually tested for correctness prior to any integration attempts with the frontend. This allowed us to ensure that any issues we encountered during integration were not stemming from faulty code on the backend.

The database was tested in a few different ways. Firstly we ran a series of queries and checked the results in an attempt to see what commonly used queries do to the data within the database. These queries followed the flow of use cases that were developed and simulate the data needed in order to run the assignment algorithm. These tests and queries were run through MySQL workbench.

The backend algorithms were tested by following use cases with static data numbers. Standard expectations for the algorithm were developed such that we had expected results that could be verified for the tested assignment scenario. Additionally the APIs were tested to ensure that we were able to reliably use them to collect data and information that was needed for the assignment algorithm. These unit tests follow the standard unit testing framework and are done in Junit or the equivalent testing environment.

All unit tests were required to pass before integration could begin. The requirement was put in place to reduce the pain of integration testing. Looking back this requirement probably saved us several hours that we needed to finish up integration.

5.3 Interface Testing

The primary interfaces present in the current iteration of the application architecture are the routing interface and the order interface. The routing interface is used for initially allocating the trucks and reassigning orders in the instance a truck breaks down mid-route. The order interface is used for inputting orders into our system for use in the routing interface.

Interface testing was based on the black box testing model. Black box testing was the best suited testing method because of its aims to not dig deep into the code, but to interact with the UI, test the end user functionality, and make sure that every input and output of the system meets the specified requirements.

- Interface testing scenarios that were verified for correctness:
  - Web UI actively allows user input such as mouse clicks and scrolls, testing functional status of UI components such as buttons on click listeners and scroll listeners.
  - Messaging page actively updates and persists data.
  - Order tracking page responds to dispatcher order information requests with correct data, testing backend service functionality and frontend json response handling and formatting.
  - Route allocation page displays up-to-date and accurate route information on the map component on the page. Calls to the Mapbox API and page map component updation were tested.

5.4 Integration Testing

Integration testing occurred once both frontend and backend components were verified to be working correctly in isolated environments. Integration testing was both the most important and lengthiest part of the testing process because of the nature of incorporating two previously independent “systems”. The testing was primarily done through loading the desired webpage that made calls to the backend and checking that the results returned were correct and formatted properly. Once the correctness of the API request was checked, we verified the frontend was able to parse the result and utilize it as necessary.

5.5 System Testing

Our system testing strategy was to focus on interaction between all parts of the system as much as possible; individual tests should be less plentiful and more general. Starting with unit tests, they were still important for system level testing but weren’t as plentiful or specific. For example, a set of unit tests in the routing was narrowed down to a single test that covers the general functionality of that file. In terms of interface testing scenarios, it was most important to zoom in on scenarios that cover as much of the entire system as possible. For example, running through initial route allocation on the frontend was a good system test because it incorporated testing the various UI elements on the frontend, communication between the frontend and backend, the initial allocation itself, and the various interactions between backend components. No new tools

were added for these tests, but rather a combination of the same tools that are used for unit and interface and integration testing.

5.6 Acceptance Testing

Our approach to acceptance testing was to incorporate it into our regular test regimen. This allowed us to ensure that acceptance testing was being done on a regular basis and the project was not deviating from the initial vision. When possible, we verified functional requirements were being met through dry end-to-end testing as stated in the requirements section. As for the rest of the functional and the non-functional requirements, those were verified through regular meetings with the client. Consistent communication with the client ensured that we remained on track and the final round of acceptance testing went smoothly.

6 Implementation

The specifics of the major implementation components will now be described in detail.

6.1 DB implementation

- The SQL dialect that was used is MySQL. The tables that persist data are orders, users and trucks.
- The relationship between users and orders is one-to-many(1:M), as customer users can have multiple orders, but each order can only have a single user assigned as the order owner. Similarly, driver users can be assigned to multiple orders, but each order can only have a single driver user assigned.
- The relation between trucks and orders is one-to-many(1:M), as trucks can have multiple orders, but an order can only be assigned to a single truck.
- The relation between driver users and trucks is one-to-many(1:M), as a driver user can drive multiple trucks, but a truck can only be assigned to a single driver.
- The relation between dispatcher users and trucks is many-to-many(M:M), as an available truck can be assigned to multiple dispatcher users and a dispatcher user can manage multiple trucks.

6.2 API implementation

- The API consists of multiple microservices. The API was developed using Spring Boot.
- The API implementation is based on the MVC (Model-View-Controller) architecture. The implementation is broken down into models/entities, controllers and services.

6.3 UI Implementation

- The UI for the web app was implemented using React (HTML, Javascript, CSS)
- The UI uses consistent CSS styling TODO what css?
- Utilizes data binding with frontend components
- Communicates with the backend APIs mentioned above

6.4 Entities

Each User entity consists of the data attributes:

1. id
2. name
3. address
4. location
5. role

Each Truck entity consists of the data attributes:

1. id
2. make
3. color
4. weight
5. fuel capacity
6. cargo capacity
7. availability
8. driver
9. dispatcher
10. license plate no.

Each Truck entity consists of the data attributes:

1. id
2. weight
3. end location
4. truck

Each order entity consists of the data attributes:

1. cargo pickup origin location
2. cargo destination location
3. weight
4. number of items(if applicable)
5. type of cargo
6. order owner
7. assigned truck
8. route reallocated(if applicable)
9. truck reallocated(if applicable)

6.5 Services

The services are implemented using the Spring framework and they perform multiple functions such as DB CRUD operations and algorithm execution, with the exception of few services which are implemented using external APIs such as route mapping and communication.

6.5.1 User Service

The user service was implemented to store a registered user's credentials and retrieve them for login. This was intended to be used for dispatchers on the web application to view their current fleet of trucks and orders.

6.5.2 Truck Service

The truck service was used to add and store trucks to a database. This service can create many trucks at once. Once there are many trucks created, a Fleet object can be created to store a set number of trucks. The trucks are assigned at random to a set number of three fleets. For example, 15 trucks would be assigned to either fleet 1, fleet 2, or fleet 3.

6.5.3 Order Service

The order service was used to add and store orders to a database. The service has the ability to create many orders at once, and assign them to Trucks that have already been created.

6.5.4 Route Service

Initial Allocation - The initial allocation of the trucks utilizes a specialized version of the k-means algorithm typically used in data science applications. There are a couple reasons we chose to use this algorithm to initially allocate the trucks. The first reason is that the optimal method required researching and implementing the Multiple Traveling Salesman problem. This is a very complex problem and the true optimal solution is still being researched. Because of this complexity and uncertainty, we decided to go with a less-optimal, but more straightforward algorithm. The algorithm works by taking in the set of orders and the number of trucks in a fleet. The algorithm initializes a number of clusters equal to the number of trucks. The initial cluster centers are assigned to a random order. The algorithm then loops through assigning points to the cluster and adjusting that cluster's center point based on the locations of the orders. This process is repeated until all of the centers remain the same for two iterations, indicating that the clusters have entered a stable state. The orders are then formatted into a JSON array for use by the frontend.

Figure 6.5.4.1. Initial Allocation Web Page Example

Broken Truck simulation - For our web application, we used Mapbox to provide static maps on our visualization page and Mapbox's Navigation API for routes. Since the scope of our project was redefined to not include a mobile application, we do not have drivers on a mobile app to simulate route progress. This led to an interesting issue where we decided to simulate a truck breaking in progress on its route, but we were limited to the data provided from Mapbox. What happens in this method is, our backend application receives a very large JSON object of each displayed truck's routes, which Mapbox breaks into legs and steps. Each leg and step of a route has a duration and a distance. To simulate a truck breaking, this method randomly picks a truck that would break. The broken truck then is randomly chosen to break at a random duration along its route. The method then iterates through every truck's route and figures out where each truck is currently located at this random time, thus simulating a snapshot in time when a truck has broken. This allows us to then display this snapshot moment in time and decide which truck will pick up the broken truck's orders.

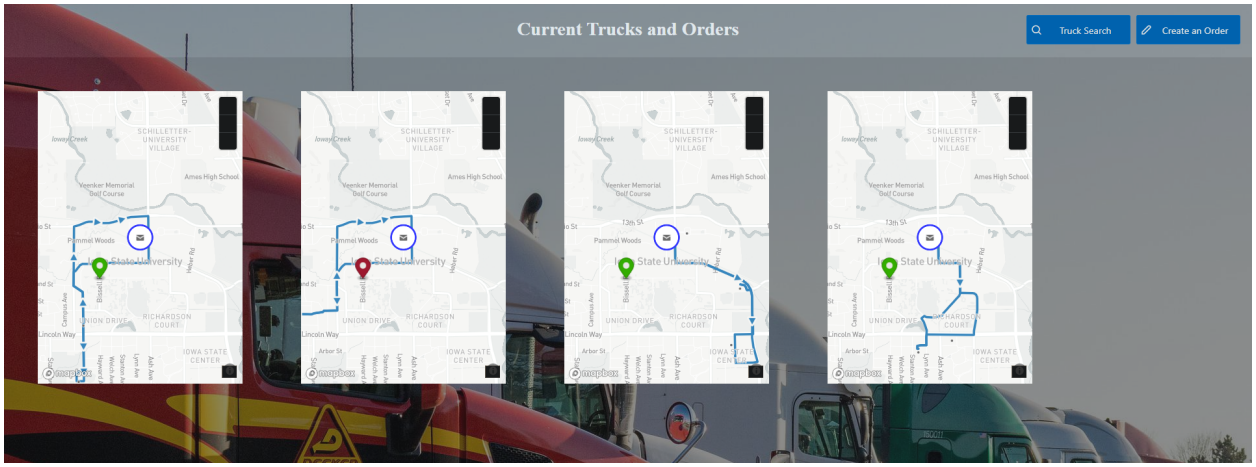

Figure 6.5.4.2. Broken Truck Web Page Example

6.5.5 Controllers

Each service has its own respective controller services http requests from clients with JSON data responses. This allows our front end web application to make calls to our backend service that is running on our server.

6.6 Security

Security, while not a large influence on the application, was necessary in certain situations. The following sections describe those situations. Additionally, no physical security was considered as our project is entirely software based.

6.6.1 User Login Security

One important aspect of ensuring security is through user login. Upon launching the app, a user is required to create an account (or login to their already existing account). While the current implementation currently functions as a placeholder and does not verify credentials,

implementing such features would be fairly straightforward. Despite being fairly simple, we had to cut implementing this feature because of the personnel and time crunch discussed previously.

6.6.2 Database Security

All important information used by the application is stored in a password protected database. This serves to prevent cyber criminals from simply taking the file and having direct access to the information. Given more time, additional security features such as encryption could be added.

6.6.3 Other Security Considerations

As mentioned above, security was not a top priority during development as our end result is meant more as a prototype rather than a full-fledged production application. However, we did consider security at various points throughout the project. Given more time with the end goal of a business-ready product, security would definitely be a top priority to protect both the company and users' data.

7 Closing Material

7.1 Conclusion

In conclusion, our initial goal for this project was to develop a truck delivery application that enables users to participate in route assignments for delivery trucks. We believe that we have accomplished the goals set in place at the beginning of last semester to the best of our ability. We have our truck delivery app functional and we were able to re-assign a truck in response to a truck breaking down. Additionally, we were also able to demonstrate a completed system of the truck delivery service from start to finish.

7.2 References

[1] Association for Computing Machinery, “Code of Ethics,” *Code of Ethics*, 2018. [Online]. Available: https://www.acm.org/code-of-ethics. [Accessed: 05-Dec-2021].

[2] Bektas, T., “The multiple Traveling Salesman Problem: An Overview of Formulations and Solution Procedures”, *Omega*, vol. 34, no. 3, pp. 209-219, doi.org/10.1016/j.omega.2004.10.004

[3] Cappanera, P., Requejo, C., & Scutellà, M. G., “Temporal constraints and device management for the Skill VRP: Mathematical model and lower bounding techniques”, *Computers & Operations Research*, vol. 1024, no.1, Dec. 2020, doi:10.1016/j.cor.2020.105054

[4] El-Sherbeny, N. A., “Vehicle routing with time windows: An overview of exact, heuristic and metaheuristic methods”, *Journal of King Saud University - Science*, vol. 22, no.3, pp. 123-131, doi:10.1016/j.jksus.2010.03.002

[5] Wang, J., Zhou, Y., Wang, Y., Zhang, J., Chen, C. L., & Zheng, Z., “Multiobjective Vehicle Routing Problems With Simultaneous Delivery and Pickup and Time Windows: Formulation, Instances, and Algorithms”, *IEEE Transactions on Cybernetics*, vol. 46, no. 3, pp. 582-594, doi:10.1109/tcyb.2015.2409837

7.1 Appendices

7.1.1 Code Repository

https://git.ece.iastate.edu/sd/sdmay22-32/

7.1.2 Important Resources

Mapbox API:
https://docs.mapbox.com/api/navigation/directions/
https://docs.mapbox.com/api/navigation/map-matching/
https://docs.mapbox.com/help/glossary/geocoding/
https://docs.mapbox.com/api/navigation/

8 Appendix I - Operation Manual

8.1 Overview

The following manual outlines how to both setup and use the delivery route optimization software. It will be divided into sections as listed below:

- Frontend: Angular Setup
- Backend (Server): Spring API Setup
- Backend (Database): MySQL Setup

8.2 Frontend: Angular Setup

The frontend of the application uses Angular, a javascript framework. To run this, follow the steps below.

1) Copy the group repository (sdmay22-32) to the directory of your choice. This can be done using either a command line or the GitLab user interface.
2) Open the locally saved repository folder using an Integrated Development Environment (IDE). For reference, both IntelliJ Idea and Microsoft Visual Studio Code were used during development and testing.
3) Before you can run the frontend, you have to install Angular and a few packages onto your machine. The packages and versions are listed below, and can be installed using npm on the command line. Be sure this is done within the project folder (e.g. **npm install @angular/cli@11.2**).
    - angular/cli version 11.2
    - primeng version 11.4.2
    - primeicons version 5.0
    - package.json
4) To run the application type the command "**npm run ng serve**". This will compile and then create an instance of the program. You will know the application is properly running if you see a screen similar to this:

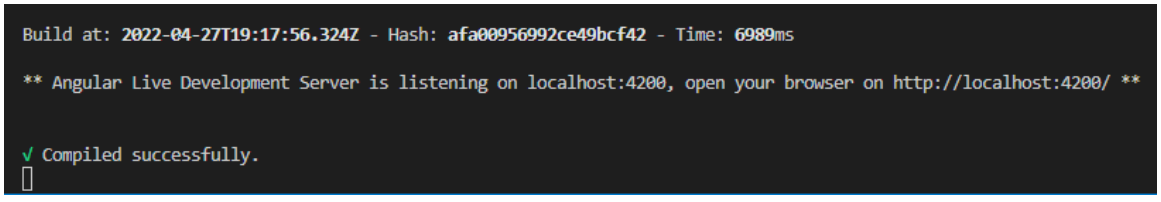

Figure 8.2.1. Frontend Compilation Success

5) The program can be viewed at the URL: http://localhost:4200/

8.3 Backend: SpringBoot Setup

The backend application uses Springboot.To run this, follow the steps below.

1. Copy the group repository (sdmay22-32) to the directory of your choice. This can be done using either a command line or the GitLab user interface. Then, checkout the branch BackendMaster.

2. Using your IDE of choice, run the “DemoApplication” Spring application file.

3. Edit application.properties to be set up for your own MySQL instance by editing the spring.datasource.url.

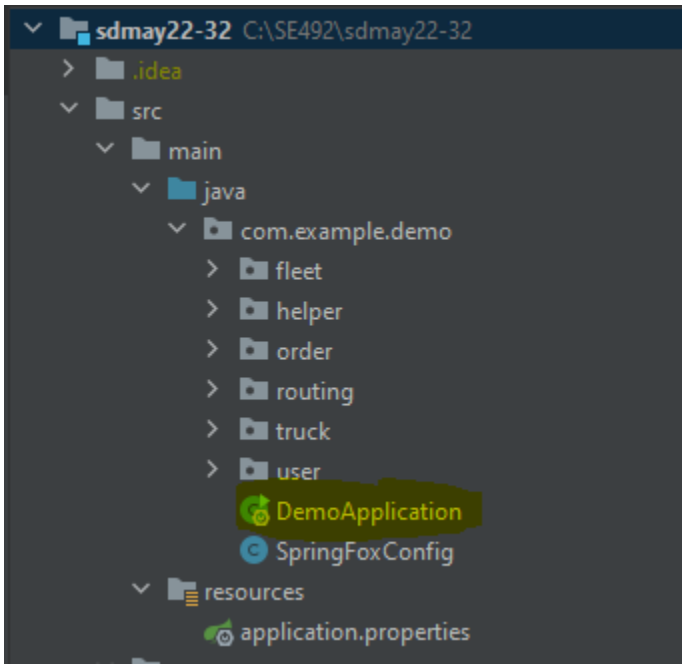

Figure 8.3.1. Backend Structure

8.4 Other steps

Starting the application

1. Navigating to the homepage
    a. Enter the link http://localhost:4200/ into your browser to bring up the application’s homepage. It should look like this:

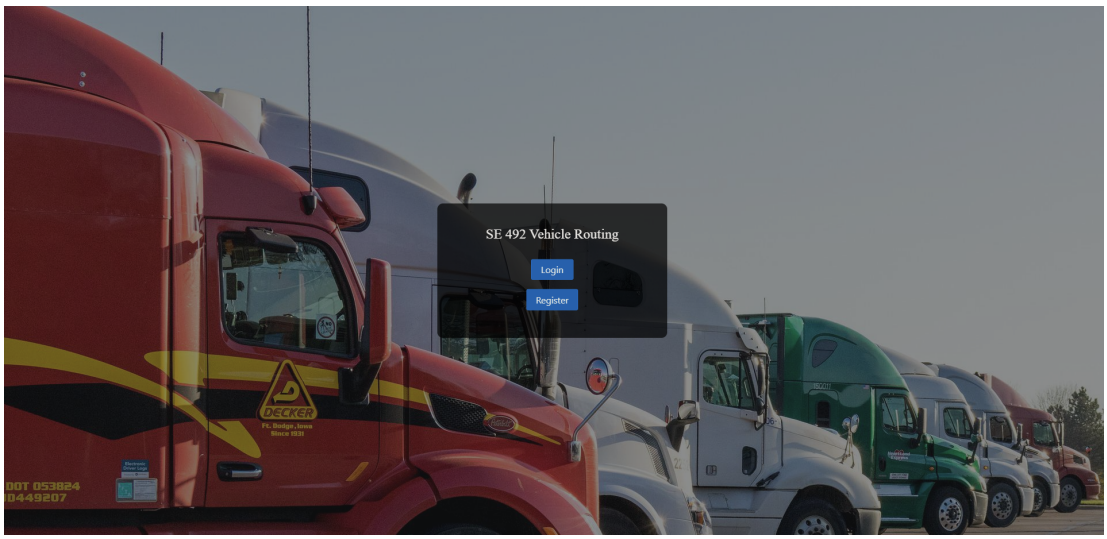

Figure 8.4.1. Home Page

2. Click on the register button to make a new account if you don’t already have one.
    a. Enter information into the fields on the registration form
    b. Click register
3. You should now be at the login page. Enter your username and password. If correct it will take you to the visualization page that displays the current trucks and their routes.

Figure 8.4.2. Visualization Page

4. After a few seconds a truck breaking will be simulated and the trucks will be reallocated.