Group 28
Aras Aniulis - S4694996
Mare Smits - S4723732
Andrei Radu Smaranda - S4771974

Final Design Document

Table Of Contents

I Team Contract

Communication

The team communication will be done through whatsapp during work hours (09:00 to 17:00), a response will be given within 2 hours unless outside of work hours, where the response time will be dictated by the urgency of the situation. Emails will be used occasionally for file transfers and help requests from the TAs. If a certain team member is not available during a specific time or day, they should disclose this in the whatsapp group chat in advance.

Goals

It is the team's goal to strive for at least an 80% which will be achieved by presenting sophisticated documentation and a working proof of technology which follows our specifications.

Subtasks

Technical Design Document

- Draft version will be submitted before 12:00 on the 29th of November for peer review.
- Final version will be submitted before 23:59 on the 2nd of December.
- The work will be split evenly between the three group members. We will meet to discuss required changes and decide how to delegate the work accordingly.
- The frameworks and APIs we use will be decided together from our joint research and planning to ensure each team member is content working with at least one of them.
- One of the team members will be in charge of amending the diagrams and ensuring they are clear and cohesive. The other two team members will split the work when it comes to the written components ensuring an even workload.

Final Design Document

- Draft version will be submitted before 12:00 on the 20th of December.
- Final version will be submitted before 23:59 on the 23rd of December.
- To write the evaluation the team will have a discussion regarding the project and will decide upon one member to write the evaluation, while the other two augment the already existing document according to the feedback received.

Proof of Technology

- Must be finished by the 16th of January
- The team will split the programming evenly, meeting at least once a week (potentially digitally) to delegate work and ensure requirements and deadlines are being met.
- The code shall be worked on by each member using github to ensure ease of access.
- One of the members will be responsible for working with the API and its integration with the rest of the codebase.

Presentations

- For the presentations each team member is expected to state their availability at least a week in advance of the presentation date.
- The presentations will be divided equally among the team members depending upon which parts of the project they have worked on previously, to ensure they are well versed in the subject matter.
- Unless unfeasible, the whole team is expected to be present for the presentation.

Final project must be finished by 23:59 on the 27th of January.

II Stakeholders

Users

- Delivery Worker - The company employees in charge of delivering the products from point A to point B.
- Warehouse Employee - The company employees in charge of loading and unloading the mediums for product transportation and warehouse related logistics.

Customers of Disruptive Delivery[1]

- Businesses Consumer - The businesses using Disruptive Delivery, they want their business to operate without issues.
- Delivery Businesses - The businesses using Disruptive Delivery to deliver on their behalf or the business to which we outsource deliveries.
- Retail Business - A retail business which uses Disruptive Delivery to ship their products.
- Individual Consumer - A consumer who wants an order delivered to a certain location.

Non-User Stakeholders

- CEO - The head of the company; their interests reflect the success of the firm.
- Shareholders - Those financially invested in the company, are interested in financial returns.
- Software and Systems Developers - Those developing the software. They want the software to be maintainable and extendable.
- Other Company Employees - others employed by Disruptive Delivery.
- Storage Companies - Companies which provide storage as a service.
- Government - in charge of regulations such as carbon emissions.
- General Public - general public which does not use Disruptive Delivery. However, public opinion and public relations is still influential.

Negative Stakeholders

- Competing Delivery Companies - firms which include delivery as part of their business model.
- Local Stores - Stores which offer their own product delivery as a service.

III Scope

Disruptive delivery is an enhanced distribution company which will rapidly deliver products anywhere at any time. This service will be the “middle man” that facilitates the transportation from point A to the consumer, where point A is the retailer in Groningen. Even though Disruptive Delivery will have its own transportation services, they will have strategic partnerships with other delivery companies for strenuous locations to make sure that all consumers are in timely reach and to maximize usage of its own amenities. The CEO’s vision for the company is extensive, however to implement this within the allocated time constraints, there will be limitations in terms of scope.

- The system will be application based and have a UI with each product having a visual representation.
- The application will allow each user to have their own shopping cart with items which they have added. If no stock of the item exists the customer will be informed with a visual or textual representation.
- The system will have a backend and be able to check the warehouse, location range, opening hours and vehicle capacity to reject or approve customer orders.
- The program will calculate a possible route from point A (pickup) to point B (dropoff) for delivery drivers.

[1] Our model assumes that customers are a subset of users.

- The system will also be able to delegate orders to other companies if such order is outside our range of operation.
- The system will have visual representations of the calculated routes for both customers and delivery drivers to be informed during the delivery process.
- The system will delegate deliveries to specific mediums of transport to most effectively deliver the products.
- Management of the application will be taken care of by the development team. This application will not have to be supervised 24/7 by the development team but rather AI will take over during off hours. If there is a problem with the software, they will take care of it within 2-3 business days.

IV User Stories and Acceptance Tests

1. High Priority User Stories[2]

"As a business, I want the ability to order at least 100 items to my location, so that I may have stock for my business operations."

Acceptance Test:

- Verify the user is able to add items to their cart using a button or other input type.
- Verify there is a visual indication of how many items are inside their cart.
- Verify the app checks whether Disruptive Delivery can deliver to their location and whether Disruptive Delivery has the capability to move the amount of items ordered.

"As an individual consumer, I want the ability to order 1 to 20 products directly to my home, so that I would not need to leave my home to shop or buy them myself."

Acceptance Test:

- Verify the user is able to add items to their cart until our maximum delivery capacity has been reached.
- Verify once the user reaches our maximum capacity, there is a visual indication that they may not add anymore items to their cart.
- Verify the application disables buttons which add items to their cart for that specific delivery.

"As a business, I want products sent from my warehouse to the customer, so that I can avoid the additional logistical encumbrance of my own distribution network."

Acceptance Test:

- Verify the user is able to select a delivery and pick up location using a text input field or other input type.
- Verify the location is a valid location on the map as well as within our scope of delivery.
- Verify the application checks whether there is an available route from the pick up location to the client location.

"As a delivery worker, I want access to an interface that will show me a possible route, so that I can deliver packages on schedule."

Acceptance Test:

- Verify the application is able to display the calculated route on a map as well as display textual instructions of how to reach the destination.

[2] The MVP for our system is composed of the user stories in the high priority category only.

- Verify The calculated route is a real possible route and should be displayed on a map.

“As an individual consumer, I want an application with images or visual representations of the products I may order, so that I know what's available.”

Acceptance Test:

- Verify the application provides a visual representation of every item available to be ordered.
- Verify the application should not allow for any items without images to be possible to be ordered.

“As an individual consumer, I want an application with a shopping cart and a main menu, so that I am able to see what I may order, or have already placed inside my current order.”

Acceptance Test:

- Verify the cart is accessible from anywhere within the UI of the application.
- Verify the application should have a page which allows the user to view all products which may be ordered.
- Verify there is also a page to view orders which were already placed.

“As a delivery worker, I want the application to show me which method of delivery I should use, so that I can deliver the product within the designated time frame.”

Acceptance Test:

- Verify that based on the capacity of the order, the application indicates the method of transportation the delivery worker should take.
- Verify there is a visual and textual representation of which delivery method to use. The application should see whether the order exceeds the maximum capacity of our different delivery methods, and choose the most appropriate delivery method.

“As a warehouse employee, I want access to an interface that informs me of what packages to move and when they are supposed to be picked up, so that I can make sure the packages arrive where they need to.”

Acceptance Test:

- Verify the application is able to display the ID of the vehicle or location in which the delivery should be loaded.

“As an individual consumer, I want to be able to schedule when my package will arrive, so that I know when it will arrive and manage my time accordingly.”

Acceptance Test:

- Verify the customer has the option to receive the package as soon as possible, in which case the order will be added to the delivery queue after the higher priority orders, or to select a timestamp when the package should arrive, in which case the order is added to the queue according to the date given.

“As a business, I want the ability to schedule deliveries at specific times, sometimes a month in advance, so that I can ensure the correct flow of goods.”

Acceptance Test:

- Verify the application allows the user to select the date and time of delivery through a specific field.
- Verify there exists a visual indication of the selected delivery time and date
- Verify the application checks the delivery date is possible, meaning there are available delivery drivers at that time.

“As a business, I want the ability to order using express delivery to have the product within 2 hours in case the customer needs it, so that I may fulfill urgent orders.”

Acceptance Test:

- Verify that after the user indicates the delivery locations, and once the application checks whether a delivery within 2 hours is possible (we have free delivery drivers and the delivery location is within 2 hours of travel), the client is able to select a delivery option using a button for the “express delivery” option.
- Verify there is a visual indication whether the user selected “standard delivery” or “express delivery”.

2. Medium Priority User Stories

“As a business, I want products delivered on schedule, so that I can ensure late deliveries do not delay the flow of other products. The deliveries should not be more than 5 minutes late.”

Acceptance Test:

- Verify the application displays an estimated time of arrival to the client and delivery driver, as well as a calculated route to ensure the delivery is possible before the agreed upon time.

3. Low Priority User Stories

“As a business, I want the ability to deliver shipments more than 10 kilometers away, so that my customers may be far from my location”

Acceptance Test:

- Verify the application is able to delegate all orders further than our maximum range away to other delivery companies which can better accommodate the clients.

“As a delivery business, I want access to the location of Disruptive Delivery’s warehouse, so that I can send packages through them.”

Acceptance test:

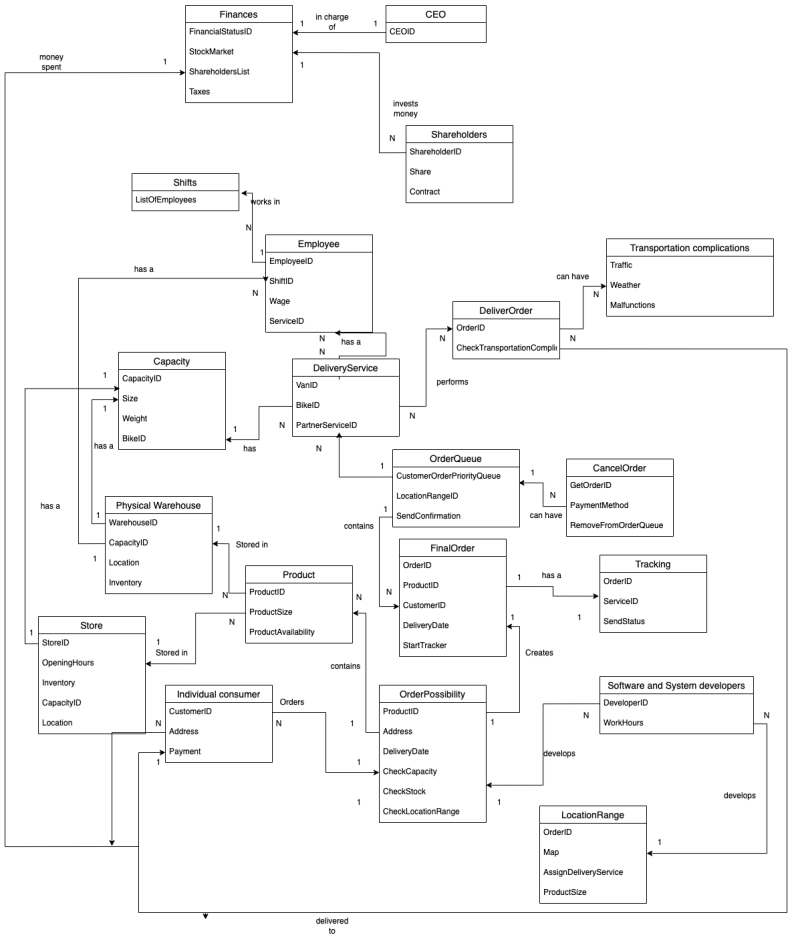
- Verify The application allows the delivery to be sent to our location, giving a pre-configured address if someone wishes to send to our location.

“As a delivery business, I want a guarantee that the package I forwarded arrived at its destination, so that I’m not forced to send another one.”

Acceptance Test:

- Verify there is a visual or textual indication that the package arrived at its destination. The indication should be shown once the delivery driver has selected that they delivered the order.

V Domain Model



5.1 This is the domain model which shows the interactions between the different components of our delivery system

- A Ceo is in charge of the companies finances
- Shareholders invests money into the financial situation of the company
- An employee works in shifts
- Delivery service has an employee, performs a delivery, has a capacity
- Delivery order is delivered to an individual consumer and can have transportation complications
- Physical warehouse has a capacity and is filled with employees
- Product is stored in a physical warehouse or is stored in a store
- Orderqueue impact the delivery service and contains a final order
- Cancel order can be initialized in the order queue
- Final order contains an order queue and has a tracker code
- Store has a capacity
- Individual consumer spendings impact the financial success of company and the order places is checked in possibility logic
- Order possibility contains a product and creates a final order
- Software and system developers develops the logic behind the order possibility check and the locations range

VI Requirement Changes

There have not been any substantial changes made to the requirements document, rather only minor corrections to ensure the contents of the document are clear and concise. Primarily, the domain model was altered to ensure that there were no vague attributes such as “ID” and that the system structure was clear.

Furthermore, there were slight alterations made to the user stories and acceptance tests to ensure consistency in the formatting of each point. Each of the user stories now follows the “As a ___, I want ___, so that___.” format. This helps ensure clarity and consistency when identifying the key requirements and reasoning for the design.

Finally, the user stories’ importance was reevaluated and reorganized to ensure the system would be the MVP. This reorganization will help facilitate a more uniform understanding of the goals and criteria of the system.

VII Scope of the Design

The original goal for the system remains unchanged however there have been several changes in the scope of the design to further clarify the type of system which will be developed and delivered. The primary changes were in regards to the form factor in which the system will be delivered. Majority of the defined functionality remains, as well as a few extensions to the functionality discussed in the original scope.

Included In Scope

- The system will be a web app and have a UI with each product having a visual representation.
- The web app will allow each user to have their own shopping cart with items which they have added. If no stock of the item exists the customer will be informed with a visual or textual representation.
- The web app will have a backend and be able to check the warehouse, location range, opening hours and vehicle capacity to reject or approve customer orders.
- The web app will calculate a possible route from point A (pickup) to point B (dropoff) for delivery drivers.
- The web app will also be able to delegate orders to other companies if such order is outside our range of operation.
- The web app will have visual representations of the calculated routes for both customers and delivery drivers to be informed during the delivery process.
- The web app will delegate deliveries to specific mediums of transport to most effectively deliver the products.
- The web app will be able to handle payments from the user, to ensure the goods are being delivered only after payment has been fulfilled.
- The customer can log in to the app, register the location they want the product to be delivered to, select a store and add their products to the cart.
- A minimum delivery time needed estimate will be displayed to the user upon completing the order within the web app.

Excluded From Scope

- Management of the application will be taken care of by the development team. This application will not have to be supervised 24/7 by the development team but rather AI will take over during off hours. If there is a problem with the software, they will take care of it within 2-3 business days.

  This has been excluded because it is more business oriented rather than in relation to the system being developed.

- The individual user application, business user application and delivery user application will allow for different functionality depending on user type.

This has been excluded because of time constraints and difficulty of development. The web app will integrate the functionality for each user into one platform

VIII Overview of the System Architecture

The web app will consist of several key components which will interact with one another; the frontend, backend and the persistence layer. The system will also follow the layers architectural pattern to help decouple the components from one another and decrease dependencies. The frontend will have several manifestations; one for individual users, one for business users and one for delivery workers. Each user will be able to login and depending on their account type, they will have access to certain functionality.

The information regarding customer orders will be hosted on the persistence layer, where that data will be accessible to the delivery workers' frontend. The backend will take care of individual client information (login credentials, user type, delivery information etc.), add deliveries to a queue to organize the order of the deliveries by their date/importance, and send back to the client the relevant information (information about products, estimation of delivery arrival time etc.). The client has the role to send requests to the server, requesting lists of retailers, lists of products, information regarding products and delivery information. The purpose of using this kind of architecture is to facilitate information exchange between the user and our database, while also helping the developers maintain a certain tidiness to the code and enhancing performance.

Three Tier Layered Architecture

1. **Frontend and User Layer**

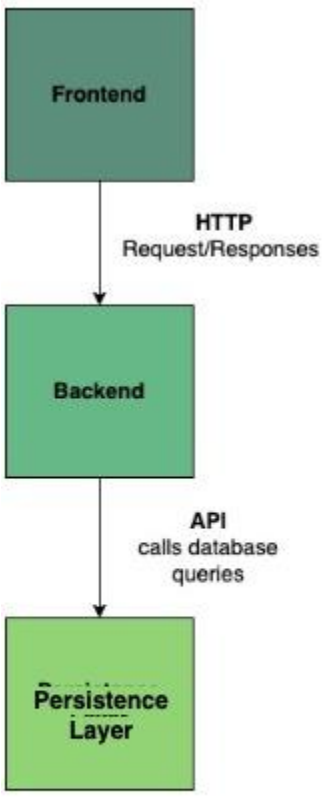
   This component will oversee all user interactions with the system. It will be the UI of the web app and allow for users to interact with different elements such as buttons and images to allow them to access the web app's functionality. This part of the system will communicate directly with the backend and processing layer, sending requests when it requires any data from the backend or the persistence. This layer will also help split the different types of users by allowing them to log in. Once logged in they will be able to access different functionalities of the web app. This layer will interact by sending HTTP requests using a custom API designed for frontend and backend communication.

2. **Backend and Processing Layer**

   The backend and processing layer will largely consist of API's and supplementary processing software which will delegate frontend requests to the persistence layer or to be further processed before being redirected again. This layer will only be accessible to the users by the predetermined requests set by our custom API which will prevent illegal access of resources or functionality which should be inaccessible to a particular user. The backend will be able to convert the HTTP requests from the frontend to SQL which will be able to be understood by the persistence layer.

3. **Persistence Layer**

   The persistence layer oversees all data storage in relation to the users and their orders. This layer will communicate directly with the backend and processing layer. It will never interact directly with the frontend and user layer and will only use the backend and processing layer as a middleman. This is to ensure security and cohesion when it comes to the frontend user requests. Once an individual user or a business user make an order, the data regarding their order will be placed into the persistence layer where this data may again be sent to the frontend, but only for the delivery workers.



Multi-Page Design

The difference between users revolves around the functionality which they may access in our web app. Therefore, there needs to be a way to differentiate between them in our UI. Therefore, to both declutter the UI layout and decouple the different user types, the page will allow the users to sign in and then display different components depending on the type of user it is.

IX Components and Frameworks

Frontend

1. **React JS**

   The primary framework/library which we will be using for the development of the frontend will be React JS. It has several advantages over other frameworks and libraries such as AngularJS; one being that it is a library and not a full framework. This means that React is easier to integrate with other external resources. Furthermore, React also features a virtual DOM which allows for the frontend to develop in real time and with increased efficiency. React is also an industry standard making it easy to find resources and documentation which may help alleviate some of the load when it comes to creating a brand new system from scratch. React also allows for the creation of different components, which allows the system to be more organized and decrease boilerplate code while at the same time increasing reusability. React is also written in Javascript which is a widely used language for frontend development which allows it to be easy to learn and familiar to many developers, us included. This also gives us the flexibility to be able swap to coding in Typescript if we run into issues in the future. Our main alternatives included Vue JS and Javascript. We decided to incorporate a framework to decrease the amount of code and work required, however we disregarded Vue JS primarily because react is more readily used throughout the industry.

2. **CSS & Bootstrap**

   CSS is vital for constructing websites as it allows us to control the themes and aesthetics of the frontend to a greater degree. Standard styling from HTML elements is not seen as very good looking, however, manually setting colors for each of them is very time consuming and requires a lot of boilerplate code. However, by using CSS it is possible to create a system wide theme for any elements greatly reducing the time taken to create an aesthetically pleasing frontend UI.

   Bootstrap is advantageous in a similar sense, but rather than help by making colors and themes easier to use, it helps by giving a standard set of pre-styled element templates which may be slightly augmented and improved to our specifications. This again helps greatly reduce our workload when designing the frontend. We additionally considered using Tailwind CSS for its ability to help make utility objects. In the end we concluded that it was unnecessary to introduce more complexity in terms of custom utility objects for the proof of technology.

Backend

1. **Custom API**

   This API will be developed specifically to facilitate the communication between the frontend and backend of the web application. This component will receive HTTP requests from the frontend and convert them into Java which can be worked with the backend. The API will also return JSON data when the frontend requires a response. Furthermore, the API will only allow for certain requests to be made so that users cannot access the full database and the data stored within. The ability to access particular data will be key in differentiating different users with different permissions within the system.

2. **Java**

   Java is a widely used programming language for backend because of its reliability, scalability and workability. Java is an object oriented programming language meaning that it uses classes to represent different objects within the code. This has its advantages such as being able to be easy to work on after

longer periods of time, or when the project's scale increases beyond a certain extent. Proper naming conventions and comments combined with classes for different objects makes Java one of the most scalable and workable backend languages. This additionally allows for the extendability of the system through addition of more classes. Object oriented programming languages follow the Open-Closed principle meaning, that code should be open for extension but closed for modification, thereby properly written java code is very scalable and reusable. We considered using NodeJs, however we were most concerned with demonstrating the key functionalities of our system and since our team had little to no experience with NodeJS, we concluded that we would avoid the additional workload of learning new languages and frameworks to save time and resources.

Persistence

1. **JDBC & SQL**

   JDBC is a Java API that allows our Java program to connect and interact with databases. The benefits of working with this API in our java environment is that it is designed to be efficient and perform well when working with large amounts of data. The drivers are available for many different databases, so Java programs can use JDBC to connect to a wide variety of databases, in our case this is SQL. The reason that we chose SQL is that it has a simple and intuitive syntax making it easy to use. Also SQL supports features such as transaction and constraints, which can help ensure the integrity and consistency of data stored in the database. How it will work is that when the user signs up for an account, the account is added to a database through JDBC into an SQL database. If the user logs in, the database is accessed via SQL and the user details are loaded into the app with the help of JDBC. When the user presses the button to change details about the account, the database is accessed and the details are modified accordingly. We considered other database solutions such as MongoDB, however we decided that a locally hosted solution would be more optimal to work with and troubleshoot issues with.

External Connections

Google Maps API

The Google Maps API is a tool that will be used to help calculate the fastest delivery route and display this to the user. The key benefit of the Google Maps API is that it is well-documented and easy to use. It is also designed to be efficient and perform well when working with large amounts of data. To Integrate this API into the web application we will first need to obtain an API key from Google and load the Google Maps JavaScript library. Then using this library we will need to implement the necessary logic to find the fastest routes which we can do by using the Direction Service which calculates the directions between locations. When an order is placed the Google Maps code will be run which will determine the ETA is for the order, which transportation method can be used, and with which other order batches it will be delivered for optimal use of time.

Google Pay API

We chose the Google Pay API as our payment platform that will allow the customers to purchase their orders. Google Pay uses strong encryption and other security measures to protect user information. It is also well-documented and easy to integrate into our web application. When a user selects the Google Pay payment option the web application sends a request to Google Pay API which includes the amount to be charged and the payment method. The API then processes the payment and returns a transaction ID. This communication will be frontend-based, however once the transaction is completed the user will be notified and the backend system will be sent the information regarding the transaction. This will be the method by which our system is able to tell which orders may now be processed and ordered.

Disruptive Delivery API

In addition to the aforementioned external APIs we were additionally required to facilitate communications with the Disruptive Delivery API. This is to allow Disruptive Delivery to be able to send orders and delivery to our system from their servers. Being able to demonstrate the functionality of this API is necessary to ensure that our system is capable of communicating with the client.

X System Flow and Interactions

User Interaction

1. **Individual User**

   The individual user differentiates itself from the business user in the sense that they are not able to schedule deliveries more than a day in advance. Additionally, they are limited by a lower maximum amount of products which they may have in their cart. Therefore certain UI elements will not be present for the individual user.
   The user will be presented with a login page or registration page, once they have logged and/or created an account, they will be redirected to a web page which displays items available for purchase. The user is then able to add the items to their cart. Once the user has completed adding products to their cart they may select the checkout option and will be redirected to a new page which will prompt them for a delivery address and payment method. On this page the user will also be able to select a time for their delivery. Once the payment process is complete the user will be redirected back to the home page, but there will be an icon for their most recent purchase.
   By clicking on the icon, the user will be redirected to another page where they will be able to view the calculated delivery route of their order and the estimated time of delivery.

2. **Business User**

   The business user is able to schedule their delivery 30 days in advance, choose different addresses for billing and delivering and order in much larger quantities than the individual user. Therefore there will be additional UI elements not present in the individual user's UI.
   The business user will have the same flow as the individual user except in the checkout screen where they will be able to select from a larger selection of dates and times at which the delivery should be completed.

3. **Delivery Worker**

   The delivery worker will have a different homepage than the other users. They will not be able to select products to add to a cart, and their homepage will be occupied by instructions of which medium of transport to take and the time to do so. In addition, they will be able to see the map of the route that was calculated.
   Once the delivery worker completed a delivery they will be able to click the end delivery button which will prompt them to their next destination and route.

API Communication

Once the order has been placed, the frontend of the application will communicate with the backend of the system, by sending an HTTP request to store the data of the order in the persistence layer. This communication will be facilitated by the frontend-backend API and will ensure that the requests are stored correctly without allowing the users to store anything or make changes to the database outside of the submitted order. This process will be entirely invisible to the users.

Backend and Processing

The backend of the system organizes the delivery with the rest of the active[3] orders according to the time of delivery. Once the order has been placed in the correct order among the other orders, the backend will send the information to the persistence layer where the order of the deliveries will be preserved, as well as its contents. This communication will be done by the backend using SQL queries to make the changes necessary within the database.

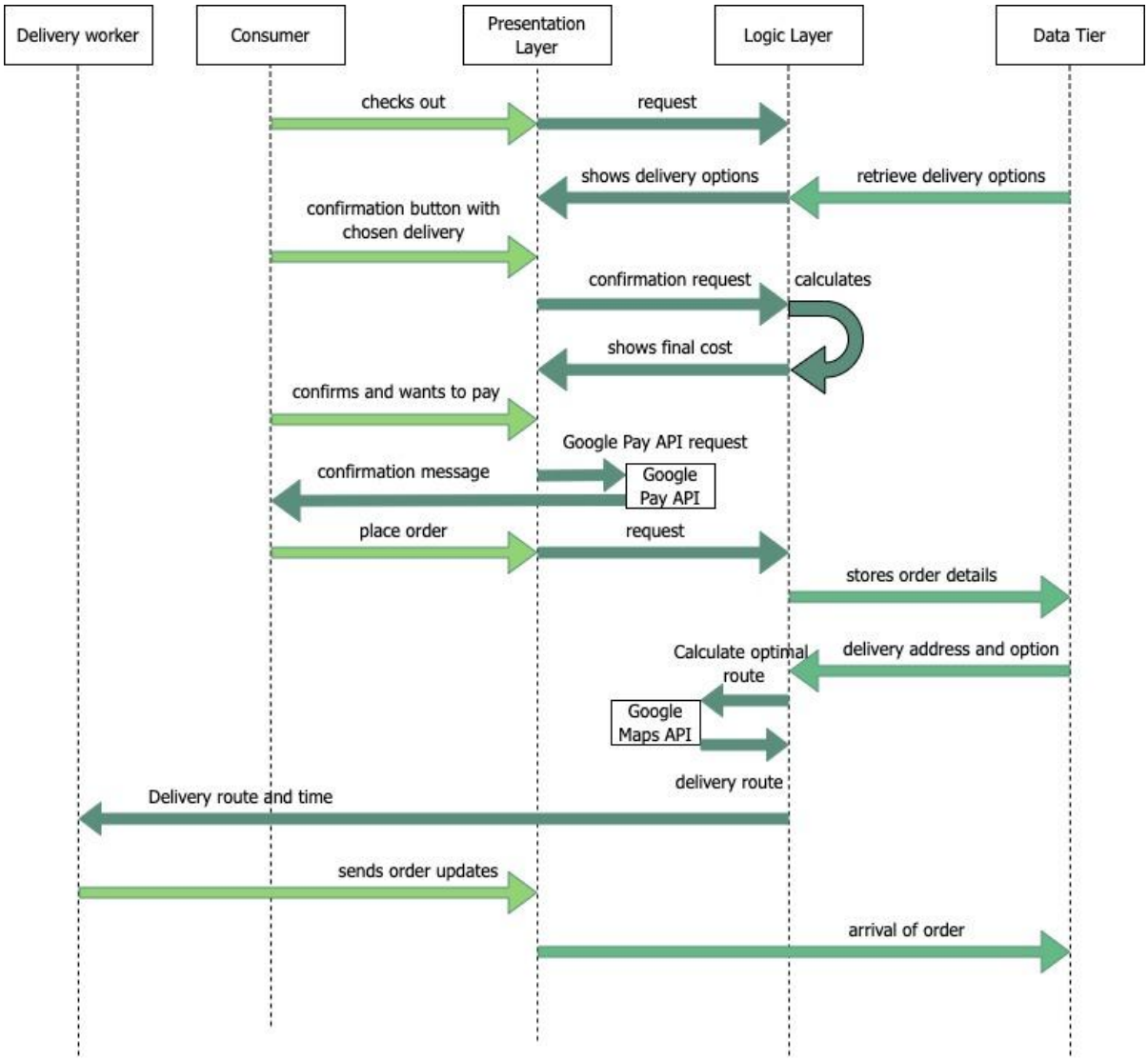
However, when users need to be shown a map of their delivery, the backend will query the database for the calculated order of delivery, as well as the addresses of each delivery the driver will make. The backend will then return this data with an HTTP request to the frontend, which will then communicate with the Google Maps API allowing the users to view the calculated route the delivery driver will take.

Persistence and Data Storage

The data storage and persistence layer will constantly be updated with new information by the backend. This will allow the system to keep track of all of the orders, as well as organize the information for delivery workers when they will need to deliver the items purchased. By all of the data going through the backend queries, it establishes a layer of security which prevents unauthorized or illegal changes to the database, ensuring that all of the orders are fulfilled as required. The users will never have direct access to the persistence layer, they will only be able to be displayed certain information returned to them through the backend communication.

[3] An active order has not yet been fulfilled, but is scheduled to be completed on the same business day.

XI Sequence Diagram



11.1 Sequence Diagram

This diagram represents the sequential flow of interactions and messages between our three tier layer and the client server model, including the API components.

XII Important Decisions

Reasoning Behind Our Web Application

Our initial idea for our Disruptive Delivery company was to present our service into an app. This idea soon flipped when we started thinking about Proof Of Technology because of the complications and restrictions that became present when working with a mobile app. Mobile applications must be developed for each operating system and must also be published through corresponding app stores. They are also developed using platform-specific languages/tools and must be tested on a variety of physician devices, which can be time consuming. In contrast, web applications can be accessed from any device with a web browser, regardless of the operating system. They can be developed using a wide range of programming languages and frameworks which allows for more flexibility and convenience. Unlike mobile app stores, web applications can be hosted on a web server and accessed through a web

browser and can be tested in a variety of web browsers and devices. Therefore, once we made the decision to switch our application type from mobile to web, our creative flexibility increased tremendously and it made it easier to find a way to implement all of our three tier layers.

Reasoning Behind The Choice In APIs

When it comes to the Proof of Technology, it is necessary to be able to demonstrate the minimum viable product which satisfies the key user stories. In our case, the system and the referenced user stories involve aspects of payment and geo-location/pathfinding which all need to be implemented. There were several ways considered to add this functionality. Firstly we decided to use the Google Maps API for navigation processing and displaying the maps since it is a very reusable and functional API, as well as is used all across the industry for the functionally which we were implementing. Since it is such a popular API it is easy to find resources and documentation to help us further develop the system. We then needed a way to process payments in our system, we originally considered using the PayPal API, but since we had already decided to use a Google API, we transitioned to the Google Pay API as well. Although both are industry standards, Google Pay sees more use than PayPal outside of the United States.

We did not consider building these aspects of the system from scratch because of the complexity of the project and time constraints. We also did not add other types of APIs because our minimum viable product did not need additional functionality which necessitated the introduction of another API.

XIII Learnings from making the PoT

The proof of technology presented several challenges we needed to overcome and brought to our attention several issues with our initial specifications. In terms of the implementation of the MVP we were not able to sufficiently meet each criteria we specified. In the end we did not have a website which fluidly communicated with the backend system, but rather they were separate entities functioning on their own. This was the issue which prevented us from achieving most of our MVP goals. This outcome led to some realizations; such as our focus on the aesthetics of the system rather than the functionality. We were misguided by the fact that we assumed our product would work fluidly and without issue when we finished implementing it and therefore dedicated too much time to make it visually appealing. Additionally, since some of the group members were not experienced with React or JavaScript, it required them to allocate a substantial amount of time to learning how to work with the language and framework.

In terms of 3rd party API connections, we also learned we misjudged the time required to implement 3d party APIs such as the google maps connection. We found there were sufficient resources, however, regarding our project structure it was not as simple because not all of our design patterns were easy to incorporate with the google maps API. This led to difficulties in the development phase of the Proof of Technology, decreasing the amount of time we could allocate for other APIs and implementation strategies. Furthermore, we should have assigned a team member with experience using javascript to save even more time.

For the Proof of Technology project we discussed implementing a backend in our technical design specifications, however, this resulted in a very heavily increased workload. This meant that in addition to facilitating communications with the DD API given to us, we would also have to implement our own API and handling to actively be able to send and receive requests from both of the sources. Although this allowed us to incorporate a database which we had direct control over; it allowed us to choose which data we wanted to use and how it would be stored, it also meant we had to create a way for the data to be manipulated by the backend. This was a massive increase to the workload. However, we decided to continue as different types of users and different types of data for each user was a key criteria in our MVP. In hindsight, although Java is a very capable and a scalable language, it requires a lot of features to be flushed out by the user when creating a backend similar to what we have envisioned. In the end, although the backend was created with little error, it was not able to send all of the required requests taking away from our potential functionality. Furthermore, it was built without being tested by the frontend and the assumption that sending json using the backend would seamlessly be read by the frontend. In the end, the frontend and backend communication was not facilitated due to lack of time to troubleshoot certain connection issues

between the two parts of the system. For such projects, it would be ideal to introduce more code to test the existing implementations, however that too would take away from the time available for other parts of the project.

During the implementation of the database we had issues with SQL. We did not want a database which ran separately from our backend server as that would add additional bulk to the project and likely increase the workload as we would have to manage another running application. Therefore we opted to use SQLite3 because we easily store the source on our backend server and thereby more tightly integrate it with the system. In addition to being more easy to work with, it was also more compatible with different systems, as some of groupmates' machines could not run the SQL database applications and would have made the process of working with them near impossible decreasing our efficiency.