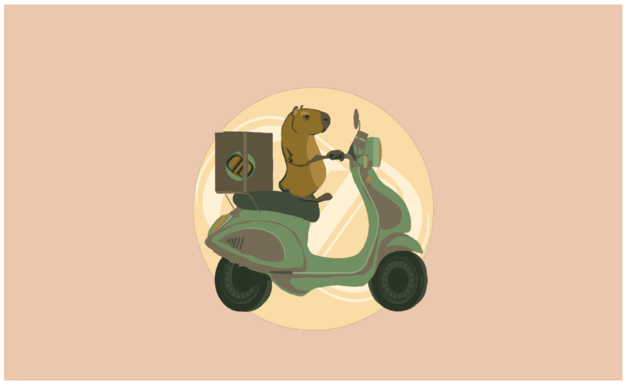


Disruptive Delivery

Design Document - Team 34

Bogdan Teodoru - S4791886
Nona Bulubasa - S4759192
Ieva Randyte - S4729315

Table of contents

I. Team Contractual Agreement

Communication Agreements

1. Due to personal schedule overlaps, most of the communication will take place on a group chat on WhatsApp, to which all 3 members of the team have access to and are encouraged to be active on.
   1.1. If responses are requested, each member is expected to respond within a day (24 hours).
2. Weekly physical meetings will take place in order to evaluate progress.
   2.1. The meetings will happen on Tuesday after the morning lecture (11 a.m.), in the Study Landscape in Bernoulli Borg.
   2.2. In the case that a member cannot attend a meeting they are required to announce their absence on the WhatsApp group chat at least 12 hours before the meeting (rare exceptions such as sickness will be excused).
   2.3. Depending on the workload for the week, extra physical meetings might be proposed, although presence will not be as strictly demanded.
3. Team members are expected to attend Problem Analysis and Software design (PASD) tutorials as these are crucial for the understanding of the project guidelines.

Goals of the project

1. The team will prioritise practicality and creativity so that the final form of the project will have an innovative and functional form that stands out from the standard delivery software.
2. It is acknowledged that time constrictions will limit the potential of the work input, however, members are to respect the team itself and take the project seriously.
   2.1. If disagreements with one of the team members' approach occurs, a discussion between all 3 will happen and, if needed, there will be actions taken.
3. The team will naturally strive for a perfect score, so the project will progress with a good grade taken into consideration.

Practical Agreements

1. A set of requirements to implement will be given to the team. It is agreed that this set is to be split equally on every assignment.
   1.1. All members should put in effort in a capacity as similar as possible, but if someone needs help with their part of the assignment, it is the team's responsibility to offer reasonable advice. This does not mean that requests for help should be abused.It is decided that it is abused once the time of the team is exploited and progress on the project is put at risk.
2. At each meeting, "to-do" lists will be established in order to keep the work organised.
3. Documents will be shared through email.
4. Alternatively, Google Docs will be used as a brainstorming platform during meetings and to keep track of log changes.

II. Vision

Small ideas deliver big

Disruptive Delivery's vision is to make a difference in the online shopping market by finding the ideal balance between cost & time-efficient shipping and conscious environmental choices.

III. Scope

A short description summing up the company's activities, functionalities and ambitions.

Disruptive Delivery is here to bring something fresh to the delivery table. Our solution brings forth simplicity for our users, one tap and everything is set to go, from the shipper to the receiver. We stay small so we can deliver fast and safe with no hassle, bringing the ultimate sustainable delivery option in the game. It also saves costs and is completely cashless.

Therefore, these are minimum technical functionalities expected:

- The seller only needs to input the formal information of the shipper and receiver, and with one button and the rest is up to us;
- Automatic payment from the retailer via the provided banking information;
- Upon the completion of an order by the seller, depending on the information provided, all parties involved will receive a confirmation email containing a tracking number;
- The ability for all parties involved to properly track the status of the order via our website;
- Consistently monitoring the warehouse capacity and storage, storing locations and providing information and updates about a package via generated QR codes;
- The option to generate informative reports that can be used to review monthly changes in trends for each department and further improve;
- Automatic generation of batches of orders to be delivered and path for delivery drivers to use for maximum efficiency;
- Real-time updates of an order for delivery drivers in case changes occur.

IV. Stakeholders

A stakeholder is a party that has an interest in a company and can either affect or be affected by the Disruptive Delivery business. They are ranked based on their importance to the company.

1. **Owner (CEO)** - main sponsor who is the one paying for the development of Disruptive Delivery. They have the final say in what functionality and scope the product should have in order to yield the optimal value.
2. **Sellers** - small businesses, grocery stores and online stores in the area of Groningen, Netherlands whose products are required to be delivered via Disruptive Delivery.
3. **Receivers** - individual people or companies that get products delivered to them.
    - Local shoppers
    - Remote shoppers
4. **Employees** - Internal people who work behind the scenes of the deliveries.
    - Delivery drivers
    - Warehouse employees
    - Marketing specialists
    - Accountants
5. **Other delivery companies** - Disruptive Delivery partners with other delivery companies for the packages that need to be delivered outside Groningen.
6. **Negative stakeholder**s - individuals/companies that do not want for the product of Disruptive Delivery to succeed.
    - Hackers
7. **Environmentalists** - people who have an interest in the sustainability factor of Disruptive Delivery products and actively offer improvement ideas.

V. Functional requirements

A user story is a general explanation of a software feature written from the perspective of the stakeholder. These user stories are ranked from most important to least important.

High priority user stories

Main user stories which are necessary for our MVP.

1. *As a seller I want to receive an overview sheet about all the orders placed every month with their location so that I know the areas where my product gets delivered the most.*

    a. Verify that all deliveries are stored in the database according to their ZIP-code & shop name.
    b. Verify that every seller receives a correct document with their name and shop name.
    c. Verify that the number of deliveries is calculated correctly.
    d. Verify that the delivery number starts from 0 every month.

2. *As a seller I want to be able to leave feedback so that my opinion is taken into account.*

    a. Verify that the website displays a text box specifically for sellers to leave feedback.
    b. Verify that the text box has two separate sections for rating and an actual message.
    c. Verify that the text box content is stored in the database.

3. *As a seller I want to see feedback from other sellers so that I know whether I can trust the delivery company.*

    a. Verify that each feedback message is stored in the database.
    b. Verify that each feedback message is visible to a customer.
    c. Verify that each feedback message has a rating on a scale 1-5, an actual message and a date.
    d. Verify that messages are displayed ordered in descending order based on rating.

4. *As a delivery driver I want to receive the fastest route already created for me before my shift so that I can deliver as many packages as possible.*

    a. Verify that all the addresses that need to be reached within a shift are grouped together.
    b. Verify that the shortest path between each address is found to avoid repeating unnecessary routes.
    c. Verify that the fastest route between 2 points is chosen with the aid of Google Maps

5. *As a warehouse employee I want to be able to access information about every single package located in the warehouse by scanning the package's QR code so that I can find every package quickly.*

    a. Verify that each package is given a generated QR code.

    b. Verify that the database attaches to each QR code an exact warehouse location (ex: hall B section 1 shelf 4).

   c. Verify that similar products are grouped together and stored close to each other so that employees find them more easily.

6. *As a warehouse employee I want to be able to notify (via QR) that I have put a package out for delivery so that I do not miss any packages.*

   a. Verify that the QR scanning system is always up and running accordingly.
   b. Verify that the QR codes are generated correctly.
   c. Verify that every QR code generated is unique.
   d. Verify that the location information is accurate and associated with the right QR code.
   e. Verify the info/error messages when there is a fault with the system or code generation.

7. *As a shopper I want to be able to change the delivery time/location in case it is not convenient for me.*

   a. Verify that the initial delivery time has been sent accordingly and the information is not wrong.
   b. Verify that the option to change the delivery time/location is available.
   c. Verify that the options made available are proper.
   d. Verify that the recommendation algorithm provides valid solutions.

8. *As a shopper I want to be able to track my order so that I am informed about the delivery process in real time.*

   a. Verify that a unique tracking number is attributed to the order placed.
   b. Verify that the tracking number is attached to the QR code information of the packages from said order.
   c. Verify that the user can access status information about the delivery on the website using the tracking number.
   d. Verify that status is regularly updated according to package location.

9. *As a seller I want for the system to automatically subtract the required amount of money to pay for orders so that I do not have to transfer it myself.*

   a. Verify that the user is prompted to enter their payment information when they place an order.
   b. Verify that the total cost of the order based on the items selected are correctly calculated
   c. Verify that the correct amount of money is deducted from the user's account through Paypal.
   d. Verify that the seller's account balance is updated to reflect the transaction.

10. *As a seller I want to be able to choose the characteristics of my packages: size, weight, area of Groningen where it needs to be delivered, fragility, etc. so that my packages get handled accordingly.*

    a. Verify that the seller is allowed to specify the size, weight, and delivery area for each package they offer.
    b. Verify that the seller is able to indicate whether a package is fragile or requires special handling.

   c. Verify that the package characteristics are accurately displayed on the seller's product pages and in the checkout process.

11. *As a seller I want to receive a confirmation email when an order has been successfully placed so that I can start making preparations for shipping.*

    a. Verify that the individual's order details have been properly processed and stored.
    b. Verify that the system has generated and sent a correct confirmation email.
    c. Verify that the confirmation email includes the necessary information about the order, such as the customer's name, shipping address, and the items purchased.
    d. Verify that the confirmation email is sent to the correct email address for the seller.

12. *As a seller I want to place my products for delivery more than once a day so that I can respond to more clients within that day.*

    a. Verify that the seller is being asked how many times a day they want to place an order.
    b. Verify that answer is stored correctly in the database.
    c. Verify that the seller cannot choose to place orders more than 3 times a day.
    d. Verify that the seller can see available order placing times.

13. *As a warehouse employee I want to receive proper details of each batch of packages that have to be put out for delivery so that I can properly manage my time and be as efficient as possible in preparing the batches.*

    a. Verify that the batch generation algorithm is both time and memory efficient.
    b. Verify that the batch generation algorithm is working properly.
    c. Verify that the generated batch information is displayed correctly and easy to digest.

Low priority user stories

Additional user stories which will be implemented in later versions of software.

14. *As a delivery driver I want to see whether a package is fragile after scanning its QR code so that I know how to treat it during delivery.*

15. *As a delivery driver I want to see how many orders I need to complete until I get my bonus so that I can control my salary.*

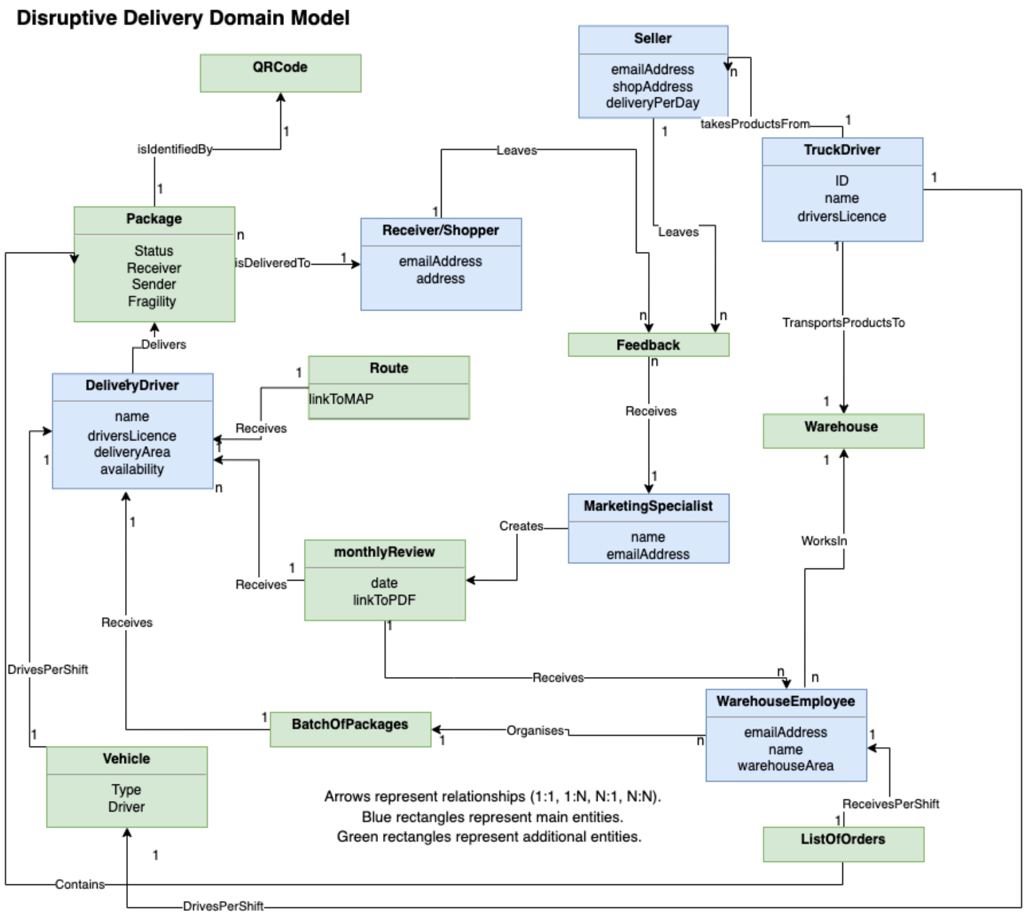
16. *As a receiver I want to choose between different sustainable packing options of my products so that I can be more environmentally friendly*

17. *As a receiver I want to be able to choose between more or less convenient delivery times (for the company) so that I can save up on either money or time.*

VI. Domain model

Entities and attributes from the domain model

1. *Drivers* are separated into two main parts: truck drivers and bike riders. Truck drivers are only responsible for delivering the goods from retailers to the warehouse and bike delivery drivers are responsible for delivering the packages for the buyers. They are identified by attributes such as name, drivers' licence and by the vehicle they are using.
2. *Sellers* are retailers that put trust into Disruptive Delivery's services to transfer their packages to the buyers. They have attributes such as email address, shop address, and how many orders they want to be delivered per day.
3. Buyers are the ones that are hoping for their package to be securely delivered by Disruptive Delivery. They are also identified by email and normal addresses, so they could be easily contacted.
4. *Warehouse employees* are a part of Disruptive Delivery that organises packages in the warehouses and prepares them for delivery. They are identified by email address and the area they work in the warehouse.
5. *Marketing specialists* are responsible for generating reports about the overall performance of Disruptive Delivery employees, retailers and buyers. They generate monthly reports so that Disruptive Delivery could know the areas where to improve.
6. *Vehicles* are either trucks or bikes and are used by drivers to transport goods.
7. Packages are the orders that have to be delivered.



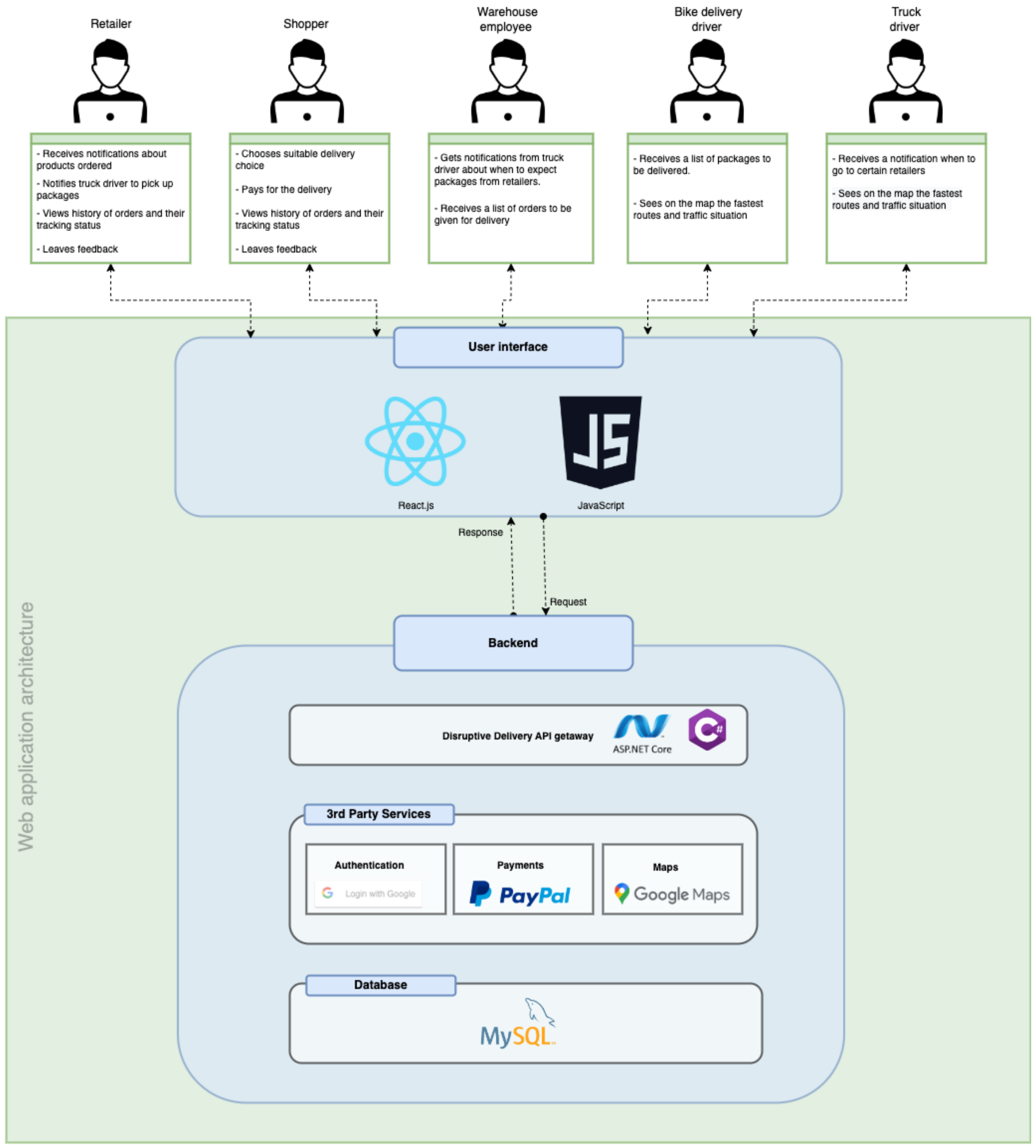
VII. Changes 1.0

After receiving feedback from our CEO we made some changes regarding the last functional requirements document. The most important ones are:

1. Changed the scope description and some user stories so that they consist of functionalities and not of technical terms.
2. Started each section with a description of what will be considered in that part of the document so that it is clear for the reader.
3. Removed some out of scope user stories since we are building a software system which does not consist of any hardware or physical tools.
4. For the 9th user story we added a few more characteristics for a package. (size, weight, priority, area to which it needs to be delivered)
5. Added user stories for accepting payments and inserting information of ordered packages to the system.

VIII. Technical Design

The diagram below shows the overall architecture of Disruptive Delivery's software. Different users will be authenticated so we can provide different User Interfaces for each of them. All components are covered and explained in the following sections. The main difference between bike drivers' and truck drivers' interfaces is that truck drivers can also see warehouse state and communicate with its workers since they only deliver packages from the retailers to the warehouse.



Retailer
- Receives notifications about products ordered
- Notifies truck driver to pick up packages
- Views history of orders and their tracking status
- Leaves feedback
Shopper
- Chooses suitable delivery choice
- Pays for the delivery
- Views history of orders and their tracking status
- Leaves feedback
Warehouse employee
- Gets notifications from truck driver about when to expect packages from retailers.
- Receives a list of orders to be given for delivery
Bike delivery driver
- Receives a list of packages to be delivered.
- Sees on the map the fastest routes and traffic situation
Truck driver
- Receives a notification when to go to certain retailers
- Sees on the map the fastest routes and traffic situation
Web application architecture
User interface
React.js
JavaScript
Response
Request
Backend
Disruptive Delivery API getaway
ASP.NET Core
3rd Party Services
Authentication
Login with Google
Payments
PayPal
Maps
Google Maps
Database
MySQL

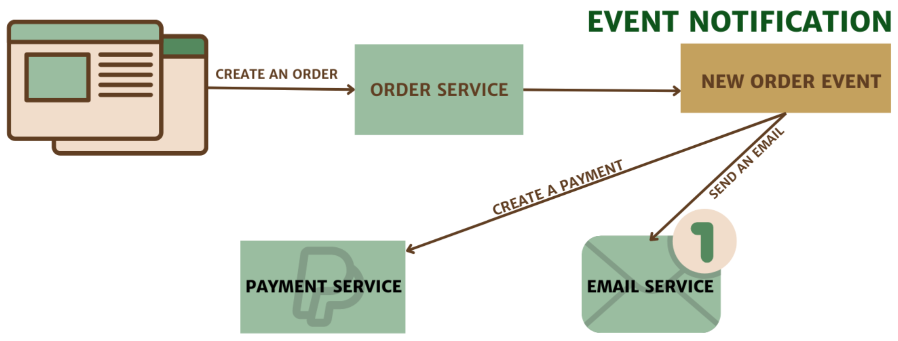
Architecture choice

There are many different approaches to designing a delivery software, and the appropriate architecture will depend on the specific requirements and constraints of the project. Some of the factors that may influence the choice of architecture include the scale of the system, the types of deliveries being made, the number and distribution of users, and the availability of resources such as hardware and personnel.

One common approach to building a delivery software is to use a microservices architecture, in which the system is decomposed into a set of independent, modular services that can be developed, deployed, and scaled independently. This can make it easier to manage the complexity of a large system and allows different parts of the system to evolve over time without requiring a complete overhaul.

Other architectural patterns that might be suitable for a delivery software include event-driven architectures, which are well-suited to systems that need to react to real-time events such as the completion of a delivery, and serverless architectures, which can be cost-effective and easy to scale for systems that experience variable or unpredictable workloads.

Given the options and requirements for our project we have decided to settle for the event-driven architecture as we believe that it is the best fit with our ideas. The way it is implemented can be seen in the diagram above.



An event-driven architecture is a design pattern in which the components of a system communicate with each other by generating and reacting to events. In an event-driven system, an event is a signal that something has happened or changed within the system. When an event occurs, it is broadcast to any interested parties, who can then respond by taking some action or performing some processing.

One of the key benefits of an event-driven architecture is that it allows different parts of the system to be loosely coupled, meaning that they can operate independently and do not need to be aware of each other's internal details. This can make it easier to develop and maintain the system, as it allows different parts of the system to evolve over time without requiring a complete overhaul.

In our specific case, the event-driven architecture fits our needs as the system needs to react to real-time events. For example, a customer places an order with us to which the system reacts and does all the magic required to proceed with the order. Another great example would be the completion of a delivery. When a delivery is marked as completed, the system then responds to this and notifies the involved parties about the event as well as archiving the entire information about the delivery and updates the path provided to the delivery driver.

Deployment

Since we are expecting large volumes of requests and changes in the system the best deployment solution is a scalable hosting solution. This means that we will be hosting our web application on a cloud hosting platform, as these platforms scale depending on demand.

For an optimal deployment we have chosen to containerize our application using Docker and rely on an AWS option such as the Amazon Elastic Container Service (ECS) as it provides a complete package with a handful of developer tools, powerful infrastructure and extensive services needed for us to deploy and host the Disruptive Delivery software.

As of why we have chosen containerization, here are 2 very straightforward reasons:

- Containerization makes it easier to deploy and manage the web application, since the application and its dependencies are bundled together in a single package. This makes it easier to deploy the application in different environments, since the containerized application can be run consistently across different hosts.

- Containerization also helps in improving the security and stability of the application, since the application and its dependencies are isolated from the host system. This helps in preventing conflicts between the application and other software on the host, and can also help to prevent security vulnerabilities in the host system from affecting the application.

Technologies and frameworks

Software type

Our team has chosen to deliver the solution for Disruptive Delivery as a web application. There are 3 reasons for this:

1. Web applications are generally easier to create and maintain than normal applications. This is because websites are built using web technologies such as HTML, CSS, and JavaScript, which are widely known and supported by a large community of developers. In contrast, creating an application typically requires more specialised knowledge and expertise.
2. Web applications are typically more accessible than applications. This is because most people already have a web browser installed on their device, so they can access a website without needing to download and install any additional software. In contrast, applications often require users to download and install them before they can be used, which can be a barrier to adoption.
3. Web applications are generally easier to share and distribute than applications. This is because websites have a unique web address (URL) that can be easily shared with others, whereas applications typically need to be downloaded from an app store or other distribution platform.

Best fit: Web application. Our customers should not be expected to download additional software since web browsers are already so convenient. Disruptive Delivery prioritises ease of use.

Backend

Main language - C#

For our backend, we have decided to go with C# as it is a popular and powerful programming language that is often used developing web applications, and it has built-in support for event-driven programming.
As such, we can use the built-in event handling features of the C# language and the .NET framework. This includes features such as delegates, which allow us to define a method that can be called in response to an event, and events, which allow us to define custom events that can be raised and handled within our web application.
There is also some experience laying dormant within us, making C# the perfect choice for the development of our web application. However, it is worthy to note that we had also considered using JavaScript, but we humbly rejected it after the results of a comprehensive comparison in which C# has overwhelmingly won the battle in all categories shown in a comparison table below.

	JavaScript	C#
Code support	World-famous.	Extensive.
Error detection	JS cannot detect errors until coding is complete and the program is executed.	In C# you can detect an error in your code and change it at any time.
Versatility	Vast.	Vast.
Code maintenance	Because JS is dynamically typed, code maintenance is a hassle, as all the code must be pulled out and examined to update or fix bugs.	Statically typed C# code is easy to maintain, just as it is easy to find and fix bugs.

Main framework -ASP.NET

As for the framework, we have settled with ASP.NET, a very powerful framework for building C# web applications and services. It offers a number of advantages over other frameworks, including the following:

1. It is easy to learn and use, even for developers who are new to web development.
2. It is built on top of the popular .NET framework, which means that it has access to a large and active community of developers who can provide support and resources.
3. It is highly scalable and can be used to build applications that can handle large amounts of traffic and data.
4. It has a number of built-in features that can save you time and effort, such as authentication and authorization, caching, and data access.
5. It is supported by Microsoft, which means that it is constantly being updated and improved to ensure that it remains a top choice for web development.

Best fit: C# & ASP.NET. Disruptive Delivery needs a code that is easy to maintain and reliably consistent since a company of our kind demands a stable system more than anything .

Database

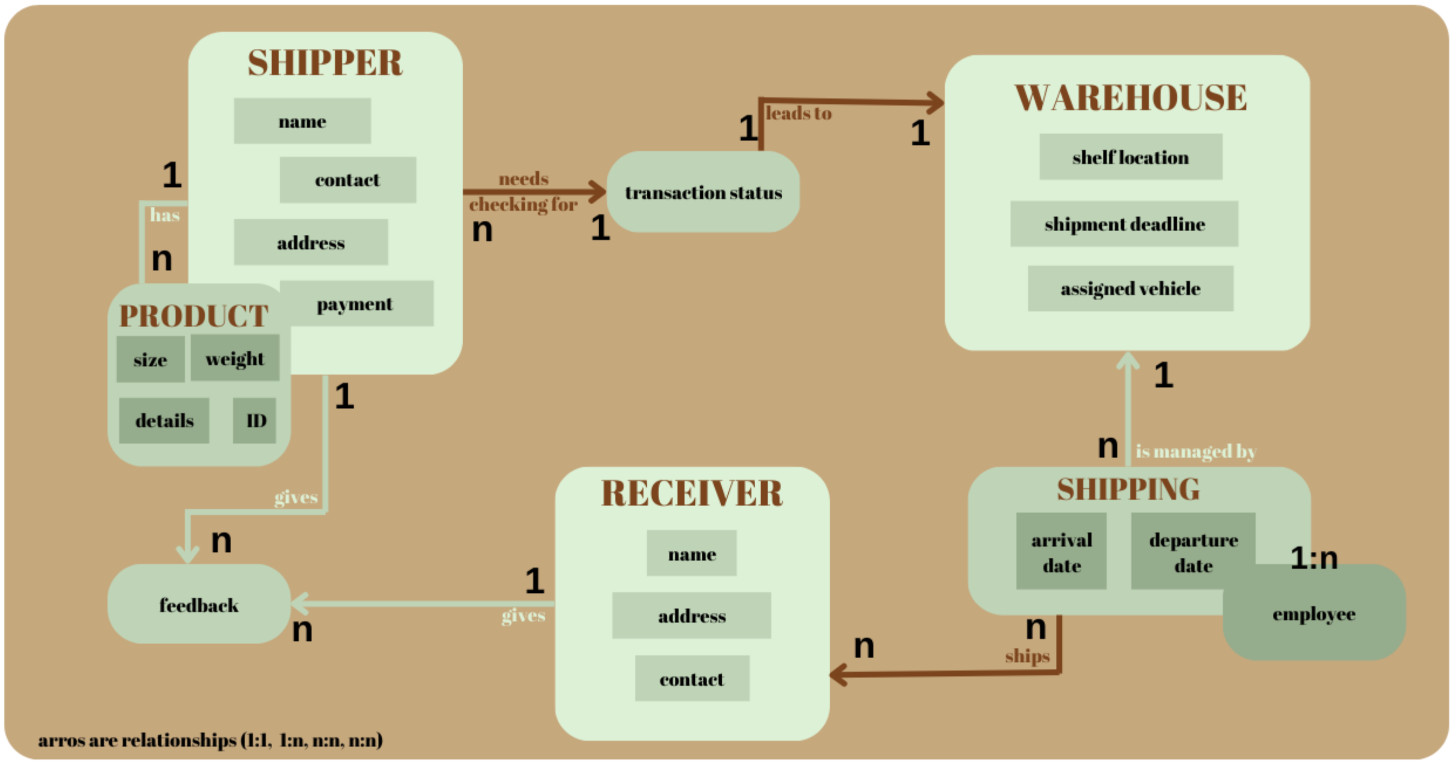
Choosing a correct database management system that meets the needs of the company is just as relevant as it can impact the performance, scalability, security, and cost of the project. There are a lot of DBMS, like Oracle, the Microsoft Server, Cassandra etc. but we narrowed our options down to *MongoDB and MySQL* since they are both widely-used and high-performance. Both systems are able to handle large volumes of data and to support high levels of concurrency, but they are designed for different use cases.

- On one hand, MongoDB uses a NoSQL data model, which is very good for unstructured data. Scalability-wise, MongoDB makes it easy to distribute across multiple servers while the data layout is JSON-like and document-oriented.

- On the other hand, MySQL is better suited for structured data that follows a clear, predefined schema. MySQL also supports transactions and ensures data integrity and consistency. Another important factor to consider is that MySQL has been around for longer than MongoDB has, which means that it has extensive testing and stability, making it easy to find solutions for potential problems.

Best fit: MySQL. Our vision for now is small-scale which works in our advantage as far as data organisation goes.

Below there is a basic design of what our database will register during a shipping.



Frontend

Main language - JavaScript

After considering a few languages for frontend our team decided to use **JavaScript**. One of the best things about this language is its *versatility* and *flexibility*. It allows us to build a wide variety of applications, so for example if in the future we would decide to build an application with similar functionality as Disruptive Delivery's web application, we could easily do that because of JavaScript.

Additionally, JavaScript has excellent cross-platform compatibility and is supported by all modern web browsers, so the users of Disruptive Delivery software do not have to worry about buying specific operating systems gadgets or use particular browsers just to use our system. Because JavaScript websites can run on any device or platform it makes it easy and accessible to a wider audience.

Main framework - React.js

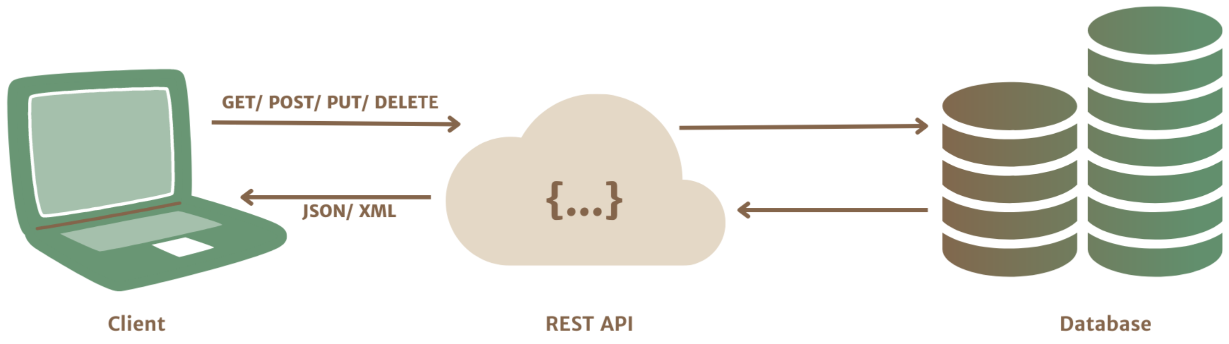
React.js is a popular and powerful JavaScript library for building user interfaces, which makes it easy to create dynamic and interactive user experiences using menus, forms, buttons, managing state and dataflow of our web application. This is a perfect tool to make a website intuitive and most efficient to use for Disruptive Delivery employees and Groningen shoppers.

In addition, it allows for modular and reusable code, which makes it easy to build and maintain large and complex applications. This means that the frontend of Disruptive Delivery website will be broken down into smaller, independent components, which can be developed, tested, and updated independently. This will save our time and effort making it easier to maintain overtime.

Best fit: JavaScript & React.js. We want Disruptive Delivery to be reachable from anywhere and we like to build keeping in mind expandability for the future of our company.

APIs

An API is a small piece of software that allows two different programs or systems to communicate with each other so we will use this to incorporate extra functionality to our advantage.



Disruptive Delivery WebShop API

What for?
Will be mostly used by Disruptive Delivery to get orders from the stores and to place offers for their delivery. It will also provide us with the delivery information, such as status, receiver, seller, fragility, etc. Also, it will store labels for the deliveries.

Why?
This API was provided by the Disruptive Delivery CEO.

Google Maps API

What for?
Will be mostly used by Disruptive Delivery drivers to search for locations, see traffic information. They will be able to see all deliveries that need to be delivered, click on the address and be redirected to the fastest route from their current location because of Google Maps API.

Why?
Google Maps API allows us to integrate maps, locations, and other geographical information into our web application. This is useful for a Disruptive Delivery since it provides users with interactive maps, displaying locations of businesses or other points of interest, or creating custom map overlays. Additionally, the API provides access to a wide range of data, including street and satellite imagery, traffic data, and more. This way we do not need to implement maps and routing ourselves.

PayPal Checkout API

What for?

Will be mainly utilised by users to pay for our services. Retailers and buyers will be able to quickly and safely pay for their deliveries since PayPal is highly secure, has a wide range of payment options, has a customizable checkout experience and can be easily integrated into Disruptive Delivery's web application.

Why?
PayPal's payment system is highly secure and compliant with industry standards, which can help to protect both your business and your customers from fraud and other security threats. In addition, it allows users to make payments using a variety of funding sources, including PayPal balance, bank accounts, credit and debit cards, and more. This can help to increase the number of successful transactions and provide a better user experience. We will also use it because PayPal's reputation for secure and convenient online payments can help to increase customer trust and loyalty in our application.

Google Sign In API

What for?
Will be used to authenticate different users and provide different user experience depending on their role. This is useful since for example we do not want Disruptive Delivery's drivers' UI to be crowded with data that is only useful to warehouse employees, buyers or retailers.

Why?
Disruptive Delivery users can easily sign in to our application using their existing Google account, which can simplify the login process and make it more convenient. In addition, Google Sign In API uses OAuth 2.0, which is a widely used and secure authorization protocol. It allows users to grant your application access to their Google account without sharing their password. By using Google Sign In API, we can access user's basic profile information, such as their name and email address, which can be useful for personalization and user management and this way we can store their account information in our MongoDB database.

Google Chart API

Why?
Google Chart API is a free service provided by Google that allows developers to create a variety of charts, including bar charts, line charts, and pie charts, for use in web applications and websites. The API generates charts as images, which can be easily embedded in HTML pages and dynamically updated with data from a variety of sources. The API supports a wide range of chart types and configuration options, and can be used with a variety of programming languages and frameworks.

What for?
The Google Chart API allows us to generate QR codes by creating a URL with specific parameters. These parameters include the data to be encoded in the QR code, the size of the QR code, and the error correction level.
When a package is added to our system, we can generate a unique QR code and link it to the package information in our database. Customers can also use the web application to track their packages by scanning the QR code or entering the package's tracking number, which were provided to them by email. Additionally, delivery personnel could update the package's status by scanning the QR code at

each step of the delivery process, such as when the package is picked up, delivered, or returned. In this case, the system can also notify the customers via email or text message whenever the package's status is updated.

X. Insights from PoT

Our PoT consisted of a primitive version of our Disruptive Delivery software which was a web application that connects to provided Web Shop API, 3rd party Google Maps API & provides the company with deliveries, their statuses and lets it post offers towards specific orders.

The code is in: https://github.com/ievaaR/pasd.git

Technical learnings

We stayed confident about our technical decisions about using C# and ASP.NET for backend and React.js for frontend, however we changed our mind and used MongoDB instead of regular SQLite as it proved to be easier to manage for such a small scale operation, and also easier to implement in this case as we did not need too many functionalities regarding the database.

One of the most important ideas that we learnt was about the whole architecture of web applications. We non-directly divided our application into 3 layers: presentation, application and resource which each served a different purpose in our project. Presentation layer is mostly used to exchange data between this and application layer, as well as present User Interface for the customer. In the application layer we had our business logic about the orders, deliveries and customers, so it was the backend part and it would send responses through the REST api to the User Interface. Last but not least the resource layer helped us incorporate MongoDB into our system and this way store data.

As we were implementing the backend, by working with C# and ASP.NET, we realised that extracting data from the body/header of a request is done in a rather overcomplicated manner and it can get very tedious. However, with the right tools it can be done, however in this specific case we feel that it could have been handled better in a language such as JavaScript.

As for the frontend we were content with choosing JavaScript because it allowed us to divide User Interface into several components and this way work on them concurrently. We learnt a lot about how states, hooks and effects work in JavaScript and also how to fetch data from 3rd party APIs and send requests to them. It also allowed us to make our User Interface modern because of Bootstrap library, which provides us with easy CSS & HTML templates so that our forms, buttons and other components would be not only functional but also easy and intuitive to use.

Teamwork learnings & Future Plans

We think that establishing a contract with fixed meetings really helped the team to keep matters organised. We evaluated what each member is experienced with and what skills each wants to expand upon and we divided our tasks accordingly. As previously mentioned, changes were made: not all of our past design decisions turned out to be the right ones for Disruptive Delivery, but the team has been able to modify its modus operandi so that our product satisfies the expectations. The biggest setback we encountered was probably the time constraint. We needed to scope down because of hard deadlines, but our ambitious visions still stand as we make plans for the future of Disruptive Delivery.

We want the company to present two separate interfaces, one for the clients and one for the employees, with different functionalities and usages catered to different stakeholders. We would also like to add the QR scanning system to the packages for faster updates and the authentication system for the employees. In the next development phase we will also provide the drivers with a "fastest-route" schema, warehouse information and, of course, a feedback box for the clients, so that Disruptive Delivery can constantly improve itself.