TQS: Product specification report

Pedro Monteiro [97484]*, Daniela Dias *[98039], Eduardo Fernandes [98512], Hugo Gonçalves [98497]
v2022-06-23

1 Introduction

1.1 Overview of the project

Objectives

In the scope of the TQS course, with this assignment we're asked to develop a viable software product (functional specification, system architecture and implementation) and apply specification and enforcement of a Software Quality Assurance (SQA) strategy throughout the software engineering process.

We're also asked to separate what is common to several businesses (deliveries/logistics services) and what is the specific area of the store (that uses the deliveries platform).

Overview of the application

Our application is an online food ordering and delivery platform named "SendASnack". While the scope of the deliveries platform will remain as general as possible, the client side application will be specific to a fast food restaurant.

1.2 Limitations

We've found some limitations in both modules.

Service

- Costumers are unable to review and rate the products/the store.

Core

- Costumers are unable to review and rate the riders.
- Businesses and riders are unable to change their profile.

2 Product concept

2.1 Vision statement

Our system will provide a core API that can be used by businesses to easily create a delivery structure for their products. This can be useful to small and medium businesses which do not have the necessary infrastructure to start their own delivery business but can use our already established system to comfortably create a delivery mechanism. The system will be focused on the food and other consumables, allowing restaurants and catering services, for example, to deliver their products.

Requirements

Deliveries platform requirements:

- Manage a workforce of riders.
- Accept orders.
- Manage orders.
- Manage riders' reputation.

Client application requirements:

- Food Ordering.
- Provide Contact Information for the Delivery Person.
- Search Filters
    - Search for items based on pricing, category, or rating.
- Order History.
- Profile Customization and Registration
    - Shipping addresses, payment options, order analytics.

Requirements Gathering

For requirements gathering and selection, we searched (conceptually similar) solutions already available in the current marketplace, such as *Uber Eats* and *Glovo*. We also researched blog articles discussing the most valuable features to include in food ordering apps.

https://www.netsolutions.com/insights/essential-features-food-ordering-apps/

Actors

Deliveries Platform

- Riders
- Clients (for example, Restaurants)

Client Application

- Restaurant Clients

Use Cases

Deliveries Platform

#	Use Case	Description
1.1	Rider Registration	Riders should be able to register an account with their personal details.
1.2	Rider Login	Riders should be able to log into their accounts to access all rider's related features.
1.3	Rider Status Update	Riders should be able to change their status (working, on pause, busy).
1.4	Accept/Reject Delivery	Riders should be able to accept or reject available deliveries.
1.5	Update Delivery Status	Riders should be able to update the status of their ongoing deliveries.
2.1	Create New Delivery	Clients should be able to create new deliveries.
2.2	Check Delivery Status	Clients should be able to track their deliveries.
2.3	Get Rider Account Details	Clients should be able to check the rider profile and account details.

Client Application

#	Use Case	Description
1.1	Client Registration	Clients should be able to register an account with their details.
1.2	Client Login	Clients should be able to log into their accounts to access all client's related features.
1.3	Customize Profile	Clients should be able to customize their profile and set their shipping address, and payment options.
1.4	Filter Products	Clients should be able to search and filter products by price, category, rating, etc.
1.5	Add Products to Cart	Clients should be able to add products to their cart and check the basket status at any time.
1.6	Order Products	Clients should be able to place orders for products previously added to their cart.
1.7	Obtain Rider's Contact Information	Clients should be able to obtain the contact information and other details (i.e. ratings) about the rider responsible for delivering their order.
1.8	View Order History	Clients should be able to view their order history.

2.2 Personas

Owner:

Maria Fernandes

Job Title
Ramona's Owner

Age
48

Highest Level of Education
9th grade

Goals or Objectives

Maria owns a restaurant and wants to deliver her burguers to Aveiro people.

Stories

- Register her restaurant on the platform.

Rider:

Joana Moreira

Job Title
Rider

Age
22 years

Highest Level of Education
12th grade

Goals or Objectives

Joana just got fired from her previous job so she wants to give an opportunity working as a rider for the deliveries system.

Stories

- Resgister as a rider and being able to receive delivery requests;
- Being able to select if she is avaiable for deliveries or not;
- Share her current location so she can be notified and select orders close to her location.

SendASnack's User:

Miguel Silva

Age
19

Highest Level of Education
12th grade

Goals or Objectives

Miguel wants to eat a burguer without having to leave his home, so he uses our application to order it.

Stories

- Register in SendASnack application;
- Set personal info and delivery address;
- Order a burguer from a restaurant;
- Track an order;
- Rate drivers.

2.3 Main scenarios

Maria, Ramona's owner, registers her restaurant in the application

As the owner of a restaurant, Maria wants to innovate and implement a home delivery system, so Maria decides to register her restaurant in the application, so she can increase her profit. By joining the deliveries platform, customers do not need to leave their house to eat burgers.

Maria, Ramona's owner, checks the statistics around orders

As an admin, Maria needs to see the statistics concerning all orders and ongoing orders. So, she logs into the deliveries platform and checks overall / last week's statistics on the main dashboard.

Miguel places an order

Due to the high workload Miguel was unable to make his lunch, so he chose to order a hamburger using the SendASnack system, without having to leave his house.

Joana receives an order request

As a Rider, Joana just needs to log in into the system because she has already registered herself.

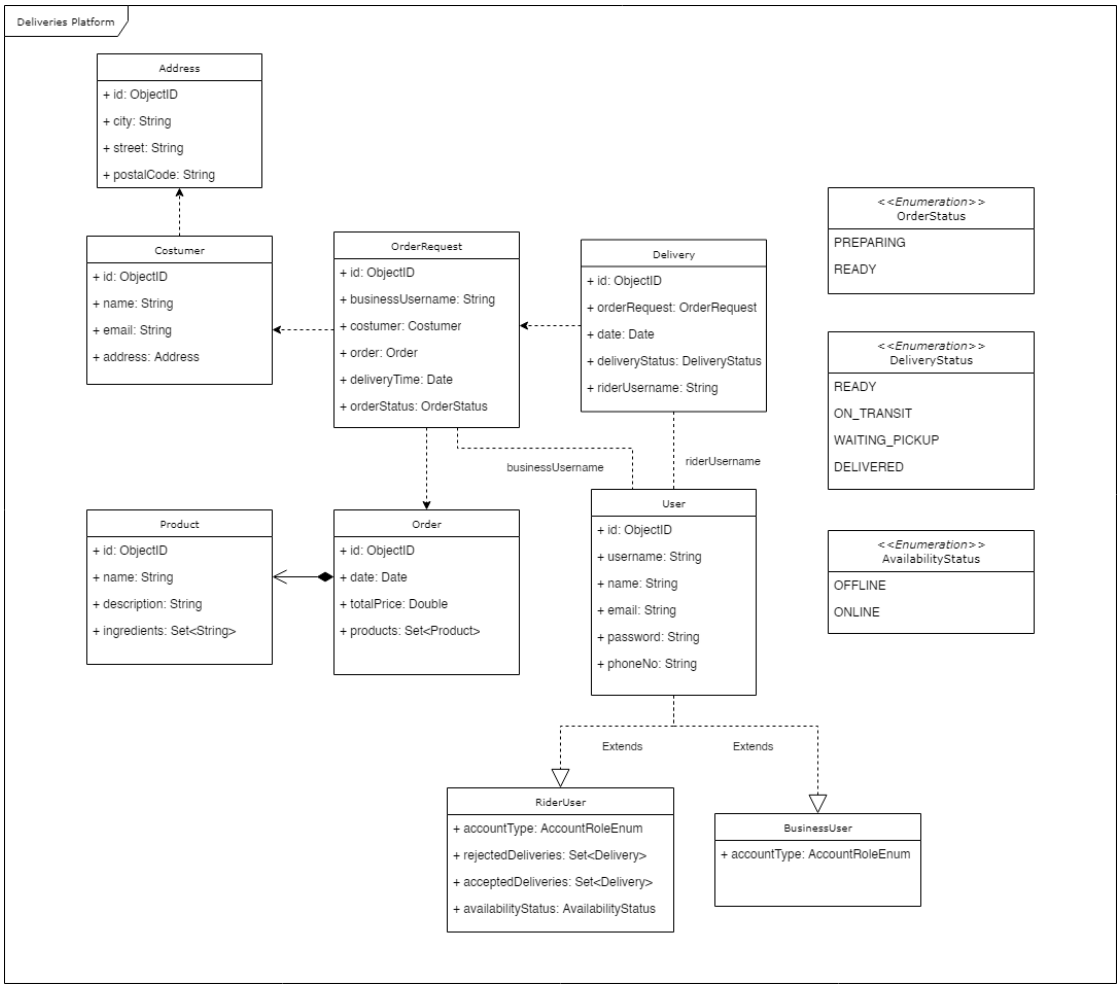
As she wants to make some money today, she first needs to change her availability so the application can recommend orders to her. For that, she needs to go to her account and change her status to active. After some time, she receives an order request and accepts it. Succeeding this event, she picks up the order from the store and delivers it to the customer.

2.4 Project epics and priorities

Sprint	Epics	Tasks
Sprint 1 (19 to 26 Mai)	Documentation	Define system architecture. Define system requirements. Create Main Scenarios. Create Personas. Create Domain Model.
Sprint 2 (26 Mai to 2 June)	Documentation DevOps Core Backend Core Frontend Service Backend Service Frontend	Create documentation for core and client APIs. Configure CI for the different projects. Draft the QA manual. Implement riders' register. Implement riders' login. Implement clients' register. Implements clients' login. Implement clients' profile (address, name, etc.).
Sprint 3 (2 to 9 June)	Documentation Core Backend Core Frontend Service Backend Service Frontend	Configure CD pipeline. Finish the QA manual. Implement 'Create New Delivery' Implement 'Check Delivery Status' Implement 'Update Delivery Status' Implement 'Add Products to Cart' Implement 'Order Products' Implement 'Filter Products'
Sprint 4 (9 to 16 June)	Core Backend Core Frontend Service Backend Service Frontend	Implement 'Push Notifications' Implement 'Rider Status Update' Implement 'Accept/Reject Delivery' Implement 'Track Order' Implement 'View Order History'
Sprint 5 (16 to 23 June)	Core Backend Core Frontend Service Backend Service Frontend	Finish Product Specification report. Implement 'Add/view ratings to products/riders' Implement 'Rate Riders' Implement 'Obtain Rider's Contact Information' Implement 'Rate Riders'

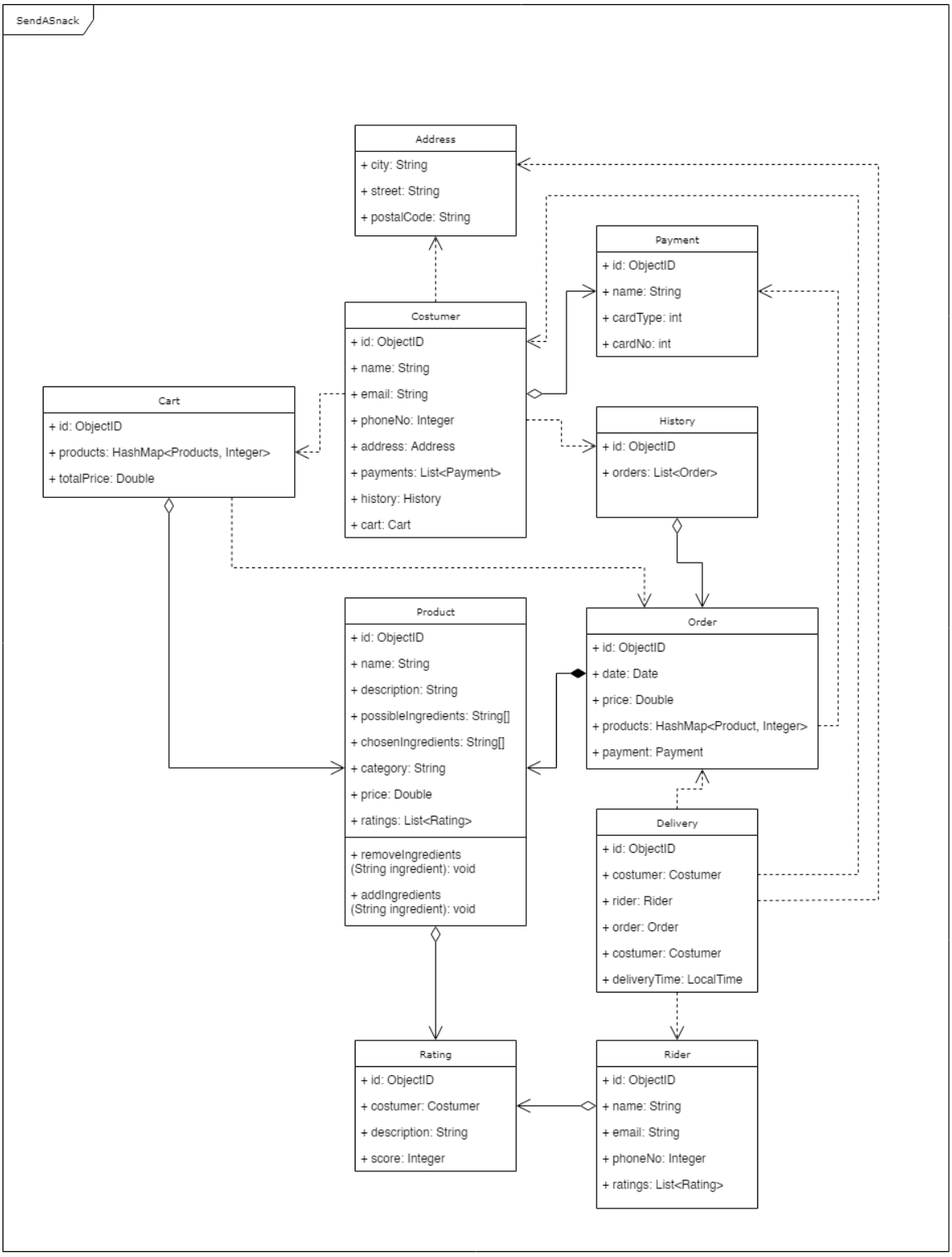
3 Domain model

3.1 Deliveries Platform

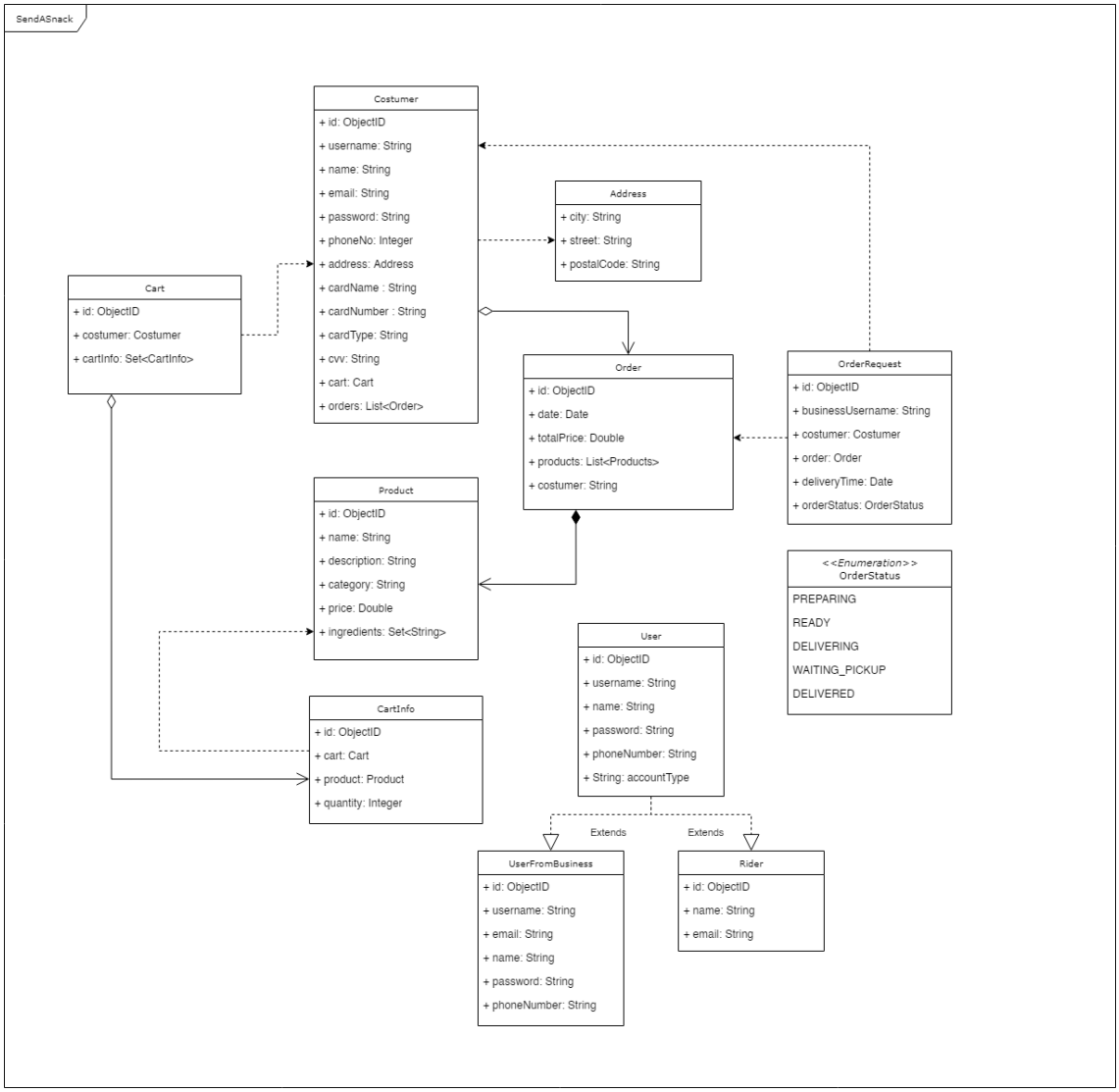


3.2 Client Application

First Version



Final Version



4 Architecture notebook

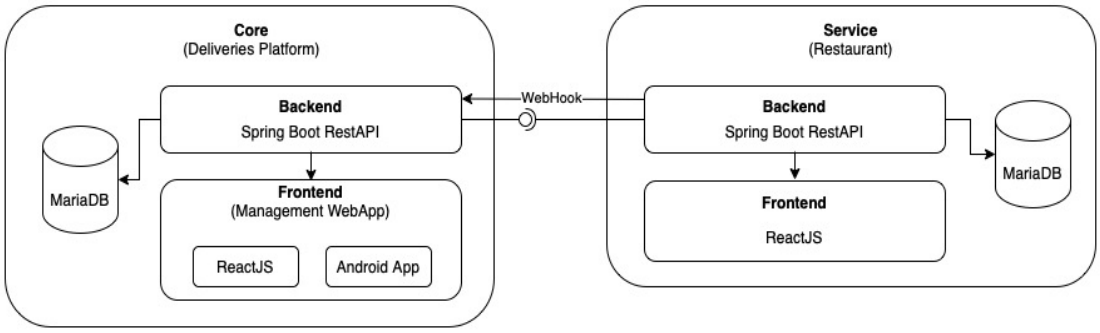
4.1 Key requirements and constrains

One of the main requirements for our system is the ability to provide a general API that can be used by several small or medium businesses to provide deliveries to their clients. This means our API needs to be scalable and be able to support the load generated by all the partner businesses.

To manage the deliveries and provide the registered riders a way to accept/reject orders and work delivering those orders we will create a web application and a native mobile application (in Android) that will provide the riders a better way to track their deliveries. Once the delivery business is very active and dynamic, i.e., there are a lot of events always occurring, we need to keep our client apps up to date all the time, which will require some extra planning and engineering architecture wise.

If the riders use the web app to track their deliveries instead of the mobile app, then some features such as location (GPS), etc. won't be available. This means our system needs to be prepared to handle this kind of variation in the client conditions and act accordingly (attributing a general payment to riders, for example).

4.2 Architectural view

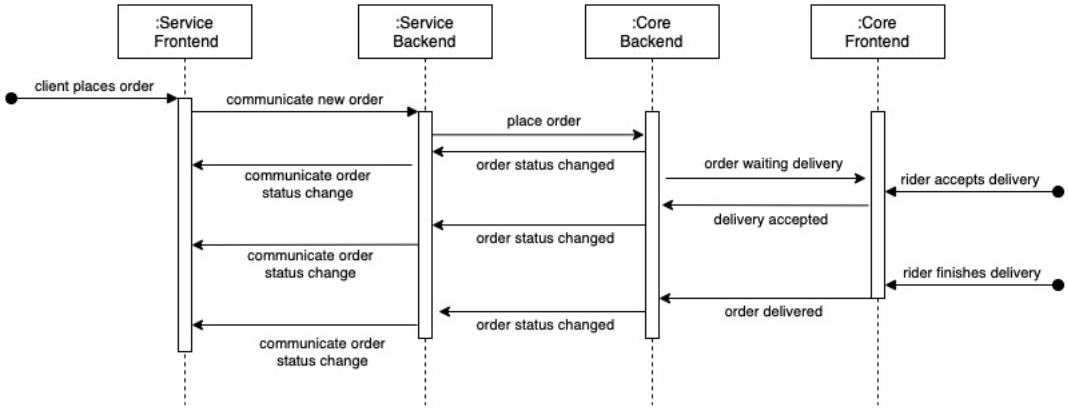


The architecture of our system will be fairly simple. It will be separated in two main modules. One of those modules, called as **Core** module, will be the one that will implement the Delivery Platform, i.e., the central API that will provide delivering services to other businesses. The second module, also called as **Service** is the representation of a business that has its own architecture but wants to use our **Core API** to start providing delivery services to their customers.

Each one of these modules will have submodules. The first submodule of the Core module is the **Backend**. This module will expose a Rest API that will have the responsibility of handling all the business logic for the delivery platform. The second submodule is the **Frontend**; this submodule will contain the web application, developed in React, and the mobile app. Both integrations will be used by riders to accept/reject orders and start delivering the customers orders. These integrations will connect to the Backend technology and get the information they need using the exposed Rest API.

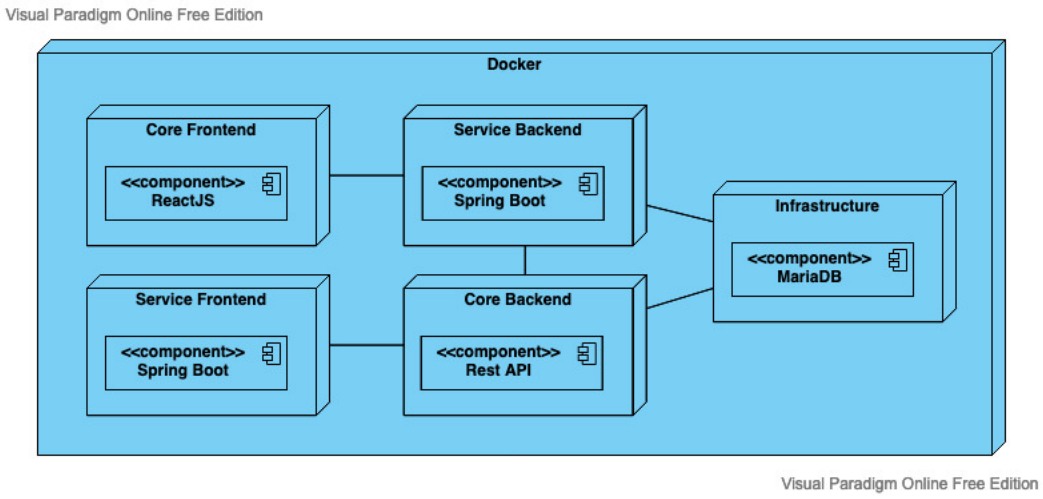
The second module (**Service**) which may be represented by an ordinary Restaurant, for example, that wants to start delivering its products will have a similar architecture. A submodule called **Backend** will be implemented using Spring Boot and will expose a Rest API that will be consumed by the second submodule (**Frontend**). The **Frontend** will contain a Web app, developed using ReactJS, that will be used by the restaurant clients to order their food.

Both the **Core** module and the **Service** module will use MariaDB as database. The **Core** module will also expose a Web Hook component in its **Backend** that will allow the restaurants to register web hooks that must be called when certain actions are executed (push notifications). Some examples of these 'actions' are for example, the change in the status of a certain order.



4.3 Deployment architecture

As already discussed previously, our entire project is divided into 4 different modules. Each module was deployed in its own Docker container and the database was also deployed in its own container – the "Infrastructure" one. The Backend services communicate with the database by using a Docker network with the name "infrastructure". The Frontend Technologies communicate with the Backend as expected through the implemented Rest APIs.



5 API for developers

The Core API is our backend implementation for the Deliveries Platform service.

This Rest API will allow developers that, as a business can create new orders that will be delivered by the riders' task force. It will also allow the integration with the Rider's frontend application, that will allow riders to accept, reject and change the state of our deliveries.

By using our Core API, a developer can interact with the delivery service and, as a business, create new orders that will be delivered by the riders' task force.

Our Service API is the implementation of an API that would be owned by a business and would, therefore allow customers to place their orders and choose the products they want to buy. This Service API will communicate with the Core API to place the orders of their costumers making them deliverable by the riders' task force.

Core Backend API Documentation:
https://documenter.getpostman.com/view/16743908/Uz5Dobw4

Service Backend API Documentation:
https://documenter.getpostman.com/view/16743908/Uz5DocAK

6 Demo

To display all the implemented features and the main use cases we've made the following demonstration videos for each one of the modules of our system (**Core** and **Store**). To better understand the flow of the system, the Store video should be viewed before the Core video.

SendASnack Store: https://youtu.be/tj7jsxZRDIY
SendASnack Core: https://youtu.be/FI_5qCqIxd0

7 References and resources

What is deployment diagram?
https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-deployment-diagram/

Documenting your API
https://learning.postman.com/docs/publishing-your-api/documenting-your-api/

What are WebHooks?
https://zapier.com/blog/what-are-webhooks/

React Router testing library
https://testing-library.com/docs/example-react-router/

Make my persona
https://www.hubspot.com/make-my-persona