TQS: Product specification report

Diogo Filipe Amaral Carvalho [92969]*, *Pedro Miguel Loureiro Amaral [93283], Rafael Ferreira Baptista [93367], Ricardo Saraiva Cruz [93118]

v2020-05-28

1 Introduction

1.1 Overview of the project

This is the final TQS course project in which the students are expected to deliver a "medium-sized project, comprising a multi-layered, enterprise web application and a Software Quality Assurance (SQA) strategy, applied throughout the software engineering process."

In this project we are encouraged to implement a "digital marketplace for "last-mile" deliveries" that should be composed by 2 main components:

- The deliveries platform includes the riders operations and management and should be reusable in different contexts.
- A specific application (business initiative) that should use the deliveries platform to deliver products and serve some purpose.

Having this proposal in mind, we decided to implement the safeDeliveries application which will be the correct choice for anyone that wants to deliver some product(s) to another person in his region.

1.2 Limitations

Due to the lack of time we had and since the main goal of the course was not towards the functionalities developed, we have not implemented some aspects that would add an extra value to the application. For example:

- Mobile application
- Tracking of rider location in real time

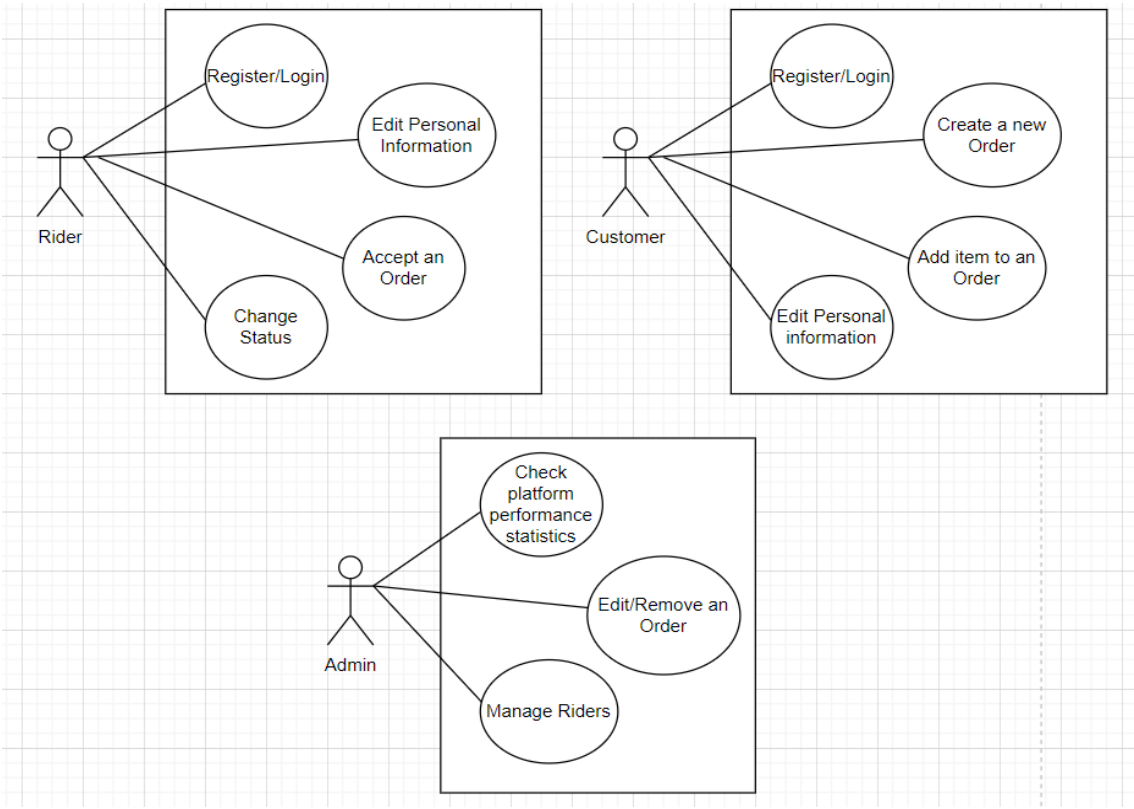
2 Product concept

2.1 Vision statement

safeDeliveries is an application designed for everyone that needs to deliver items to a friend, colleague, or anyone else. Sometimes we have products that we want to loan to a friend, or we do trades online, and we need a way to deliver the product to the buyer. Situations like these are covered by our app, which provides an easy, safe and fast mechanism to deliver these products between people.

Being a digital marketplace for “last-mile” deliveries, safeDeliveries has some common aspects with other applications like Glovo and UberEats. However, unlike these applications that are made for people who want to buy items from a shop or retailer, we are designed to operate in situations where the items are being moved between people and no shops are involved.

In order to have a better vision about our application, we are going to present a UML Use Case diagram to support our explanation (Figure 1).



2.2 Personas

Rui is a young man of twenty-four years, born in Ílhavo, in the region of Aveiro. He completed his degree two months ago in the area of Computer Engineering at the University of Minho. In addition to work, Renato practices futsal and likes to take his girlfriend to a movie night.

Motivation: Rui who represents the riders, as a recent graduate, wants to obtain a source of extra income that allows him better financial comfort, so he looks for our application to work as a delivery man.

Sofia is a fifty-six-year-old woman who lives in Lisbon and belongs to the safeDeliveries project management team. At the end of a long day at work, Sofia likes to walk her cat and loves to solve a good sudoku.

Motivation: Sofia, who represents the administrators, wants to consult detailed information on the number of requests that were made through our application today, in order to obtain a better analysis of the evolution of the application.

Fátima is a young woman of twenty-seven years old, inhabitant of the city of Braga and works in a supermarket near her home. Graduated in social communication, Fátima is still looking for a job in her field. Fátima likes to read and write long novels in her spare time.

Motivation: Fátima, who represents the customers, intends to use our application in order to be able to deliver products to people without having to make the trips themselves. Even today, she plans to hand over a necklace to a friend as a loan, as they are both preparing to have a festive night.

2.3 Main scenarios

Rui changes his status to “waiting” and receives an order

Rui, as a rider, needs to get some money since he wants to take his girlfriend to dinner. Since he already has an account, he logs in and changes his status to “waiting”. As soon as a suitable order arrives, the system notifies him. Now he is ready to check the order’s details where he can find he pick up and deliver locations to perform the request.

Fátima creates a new order

Fátima, as a customer, wants to send a pair of jeans to her best friend that is hanging out with her tonight. In order to accomplish that, she registers herself in our application, goes to the page where she is able to create a new order, adds an item to the order (in this case, the pair of jeans), fills in the required fields and concludes.

Sofia verifies app statistics

Sofia, as a member of safeDeliveries administration board, wants to verify the usage of our system to see how the application is growing up. Therefore, she logs in with an admin account, goes to admin panel and check the statistics.

2.4 Project epics and priorities

Sprint	Epic
1	• User interface prototype
2	• Rider’s Registry. • Customer’s Registry • Development of Riders’ API.
3	• Create orders in customer application • Automate assignment of orders • Development of Customers’ API
4	• Development of statistics regarding rider’s app • Customers being able to rate Riders

3 Domain model

In our data model, we will have 5 entities organized in 5 tables.

Starting with the User entity, it will have an ID, a first and last name, an email that will be unique for each user, a password, and an account type to differentiate between normal users and administrator users.

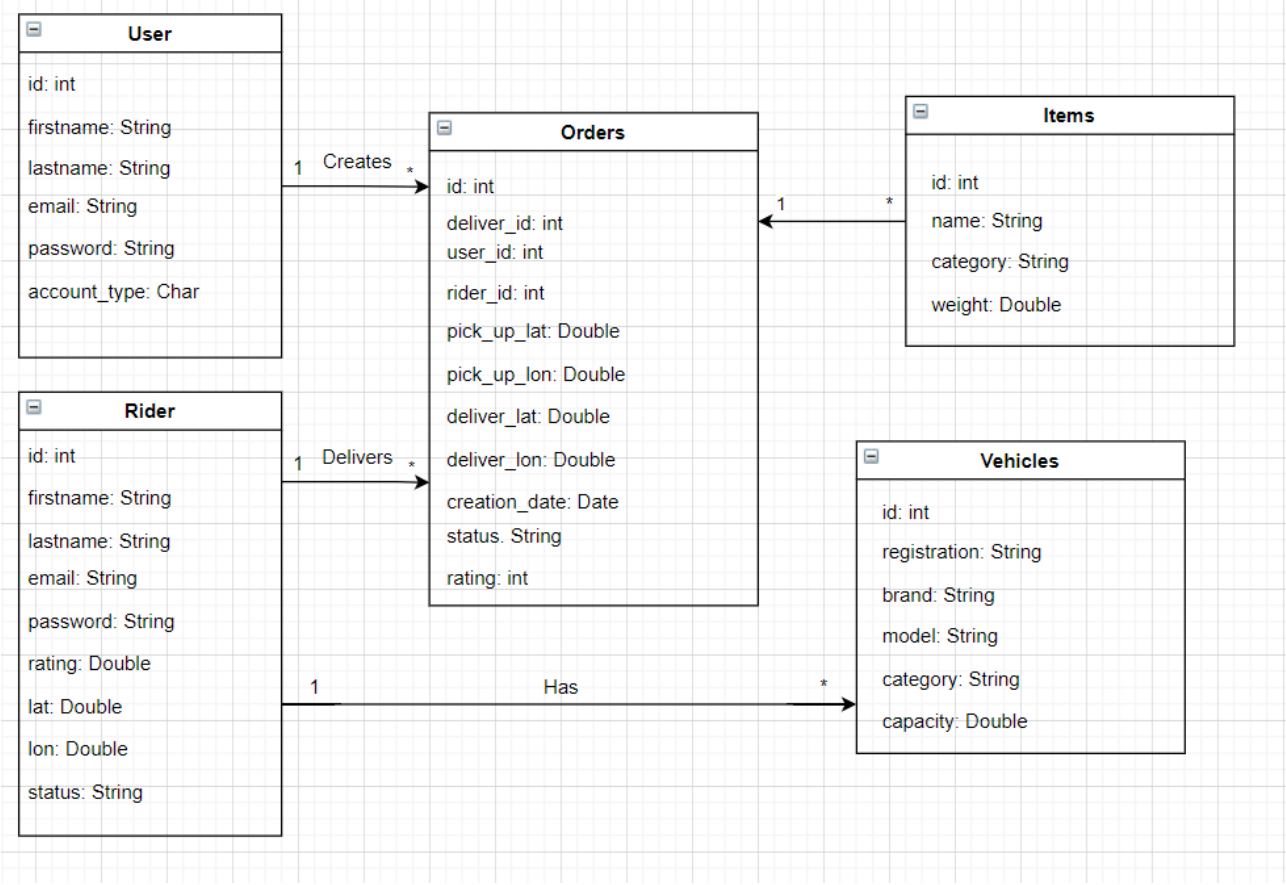
We also have the Rider entity, which is intended to store information about the riders who will have to deliver orders made by normal users. In addition to storing the same information as in the User table, it is necessary to store the vehicles that each Rider has, and for that, a relation 1 to N is made for the Vehicles entity that has an ID, a registration, the brand and model of the vehicle, the maximum weight capacity, and a category to know what type of vehicle it is.

To store information related to orders, there is an Orders entity that has an ID, a creator (the User who made the order), a deliver (the Rider who accept the order and will deliver it), the address for the Rider to pick up the items, the address for the Rider to deliver the items, a date when the order was created, a status (for example, pending, delivered, ...), and a rating that the customer will give to the Rider.

Since orders can have more than one item, the relation of orders to items is 1 to N.

The Items entity will have an ID, the name of the product, the product category, and its weight.

This information can be seen in the following diagram:



4 Architecture notebook

4.1 Key requirements and constraints

Our order application aims to receive orders made by normal users through a web page in a simple way. Our delivery application aims to allow users, in this case, users who are going to make deliveries, to choose the orders they want to deliver. To achieve these goals, we have to comply with the following requirements:

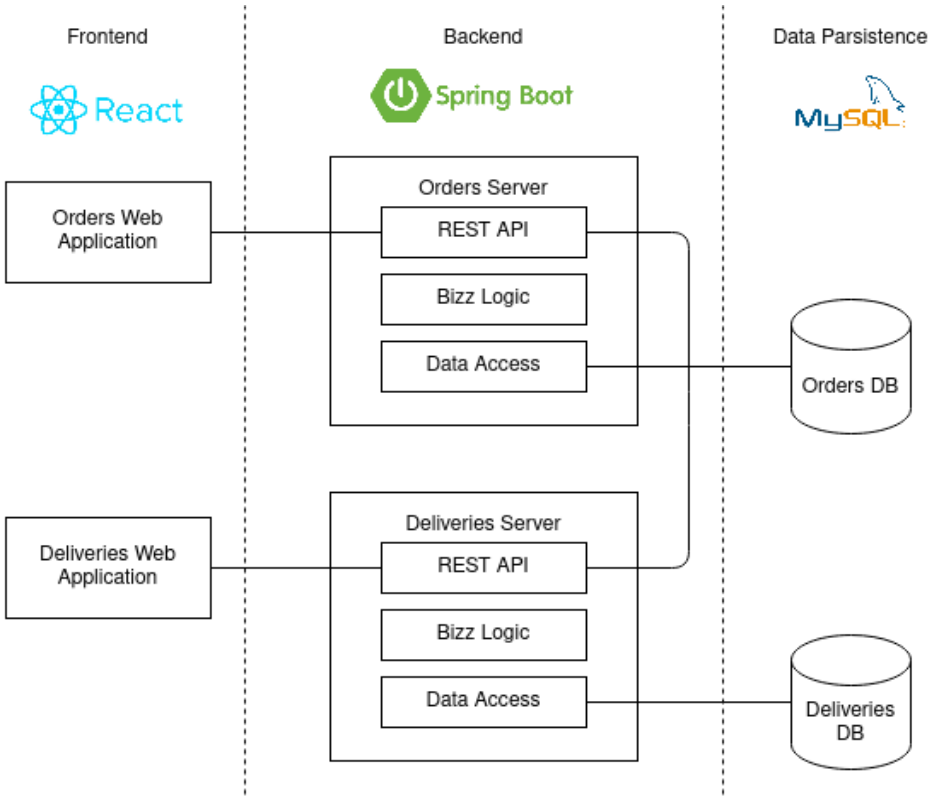
- The delivery application must be generic, it must support several order applications simultaneously.
- Both applications must process the information received and store it to guarantee data persistence.
- The two applications must have web interfaces, to which they must expose the data and allow to perform the operations destined for each application in a simple way.
- Both applications must have an API that provides data and that allows saving and updating data.

4.2 Architectural view

The core of our two applications will be composed of two applications developed in Spring Boot that will work as a data processing and business logic layer.

Through Spring's JPA module, each application will make a connection to a data persistence layer that will consist of two MySQL databases (one for the order application and the other for the delivery application).

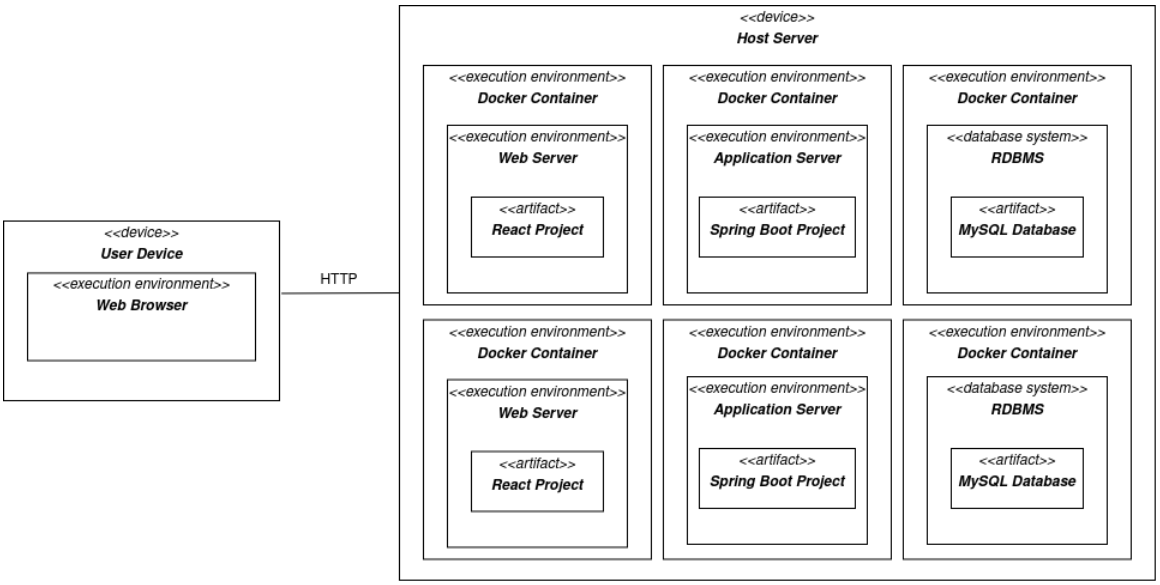
We will also have two RestFul APIs, one for each application, which will provide the endpoints that are going to be used by the two presentation layers to perform the CRUD operations towards our data. These presentation layers will be developed using the React framework.



4.3 Deployment architecture

Regarding the deployment configuration, all servers will be hosted on a single machine. Inside the machine, we will follow a Containers-Based deployment technique using Docker, where each application will be located inside its own Docker container as well as each database, ending with a total of 6 containers.

In the following image, we present the UML Deployment Diagram.

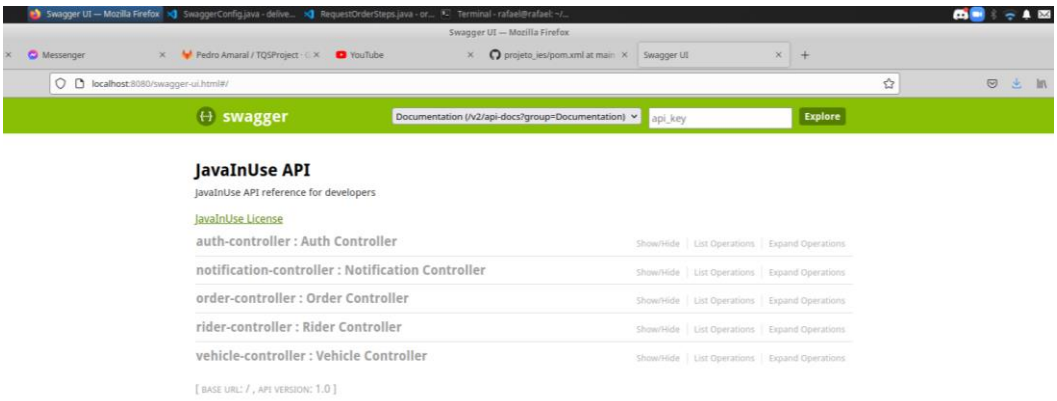


5 API for developers

Both of our applications have a REST API. We have configured swagger in both servers to get a documentation regarding our API's.

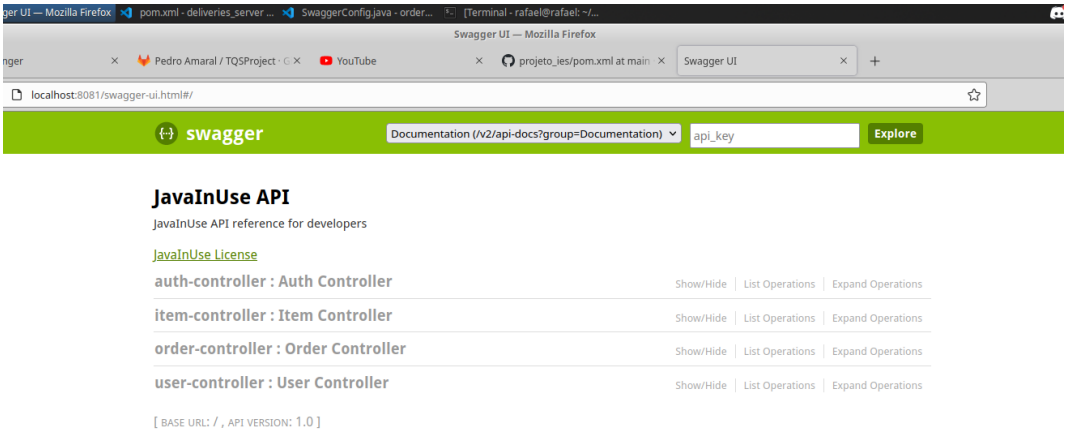
The API of Rider's Application has the following structure:

- AuthController – Register and Login operation endpoints
- Notification Controller – Controller with the endpoints required to handle operations towards rider's notification. Notification are the requests that rider receives for a new work. He can accept or refuse.
- OrderController - Controller with the endpoints required to handle operations towards orders entity. Orders are created in the client's application.
- RiderController – Controller used to find and update riders. Statistics regarding the rider entity that are displayed in the admin side are exposed here too.
- VehicleControler – Allow riders to update their vehicles' list.



The API of Order's Application has the following structure:

- AuthController – Register and Login operation endpoints
- ItemController – Controller with the endpoints required to handle operations towards item entity.
- OrderController - Controller with the endpoints required to handle operations towards orders entity.
- UserController – Controller that contains endpoint used to return all users by admin.



6 References and resources

- https://www.javainuse.com/spring/boot_swagger
- https://junit.org/junit5/docs/current/api/
- https://cucumber.io/docs/cucumber/
- https://www.selenium.dev/documentation/en/
- https://dev.to/hitjethva/how-to-install-apache-jmeter-on-ubuntu-20-04-2di9