# TQS: Product specification report

***João Borges [98155], MIguel Monteiro [98157], Vicente Samuel [98515], Vasco Regal [97636]***
v2022-05-13

**1 Introduction 1**
1.1 Overview of the project 1
1.2 Limitations 1

**2 Product concept 2**
2.1 Vision statement 2
2.2 Personas 2
2.3 Main scenarios 2
2.4 Project epics and priorities 2

**3 Domain model 2**
**4 Architecture notebook 3**
4.1 Key requirements and constraints 3
4.2 Architectural view 3
4.3 Deployment architecture 4

**5 API for developers 4**
**6 References and resources 4**

# 1 Introduction

### 1.1 Overview of the project

As a project for the TQS subject, we will explain the development of a viable software product, which will have the purpose of enforcing and enhancing the use of a software quality assurance strategy which should be applied in any software engineering project. Our system is a delivery service designed to deliver books to your home.

The system will have three different interfaces, one for clients that wish to request a book delivery, another is for the couriers that will carry the delivery to the user, and finally the admin interface for courier management and statistics.
To handle the backend operations, the software will have two APIs each one connected with a database. The Delivery API will be responsible for the delivery information and the creation and

management of courier, while the Service api will be responsible for the user and book information and to connect the user with the Delivery API.

### 1.2 Limitations

As for the limitations of the solution, we wished to implement another user, the book shop, so we could have an interface for the book shop owner to add books to the database. However, due to the short deadline, this feature was undervalued and wasn't implemented.

Also the Client's and the Rider's mobile application couldn't be finished. We tried a solution by converting react to an apk using cordova and android studio, however the apk didn't work. We also tried using … however it didn't work with the dockers

## 2 Product concept

### 2.1 Vision statement

This system was developed to help bookshops expand their business to more users giving the opportunity of a delivery approach. The system will be used to assign dynamic requests made from a client to a courier that will take responsibility to collect the book on the designated bookshop and the consequent delivery. It was also developed, an interface for the users to be able to create a delivery. This system is similar to other systems like UberEats, or Glovo.

Initially, we expected to deliver books from a library instead of a bookshop and, naturally, return the book to the library after the user's requested time limit. This way, the courier would have two types of jobs: take the book from the library and deliver it to the user or collect a book from a user and return it to the library. However, we decided to simplify the problem and change the bookshop to a library.

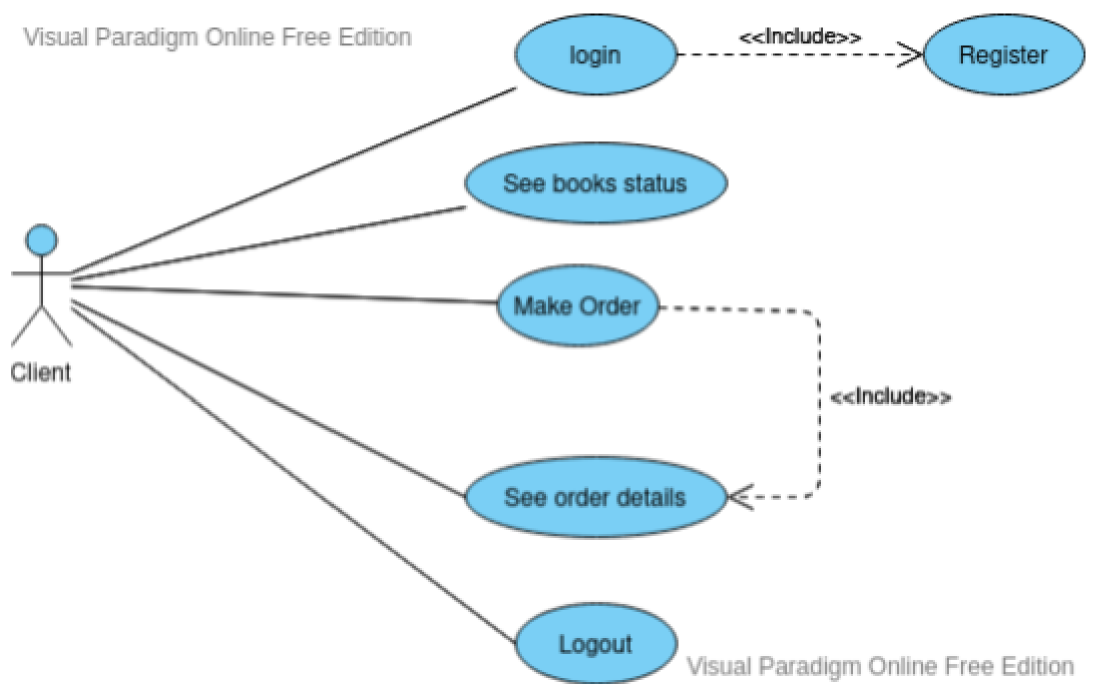


Fig. 1: Client use case Diagram

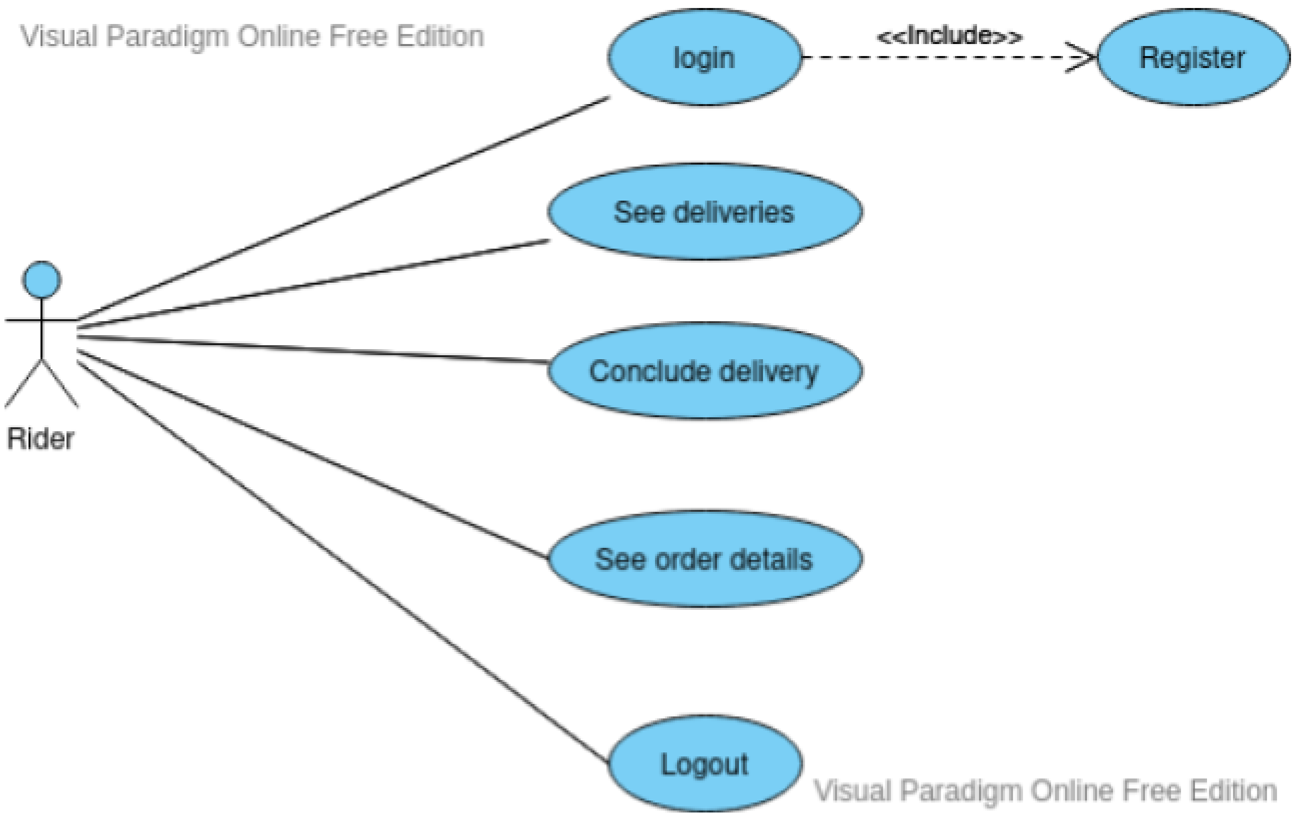


Fig. 2: Rider's use case Diagram

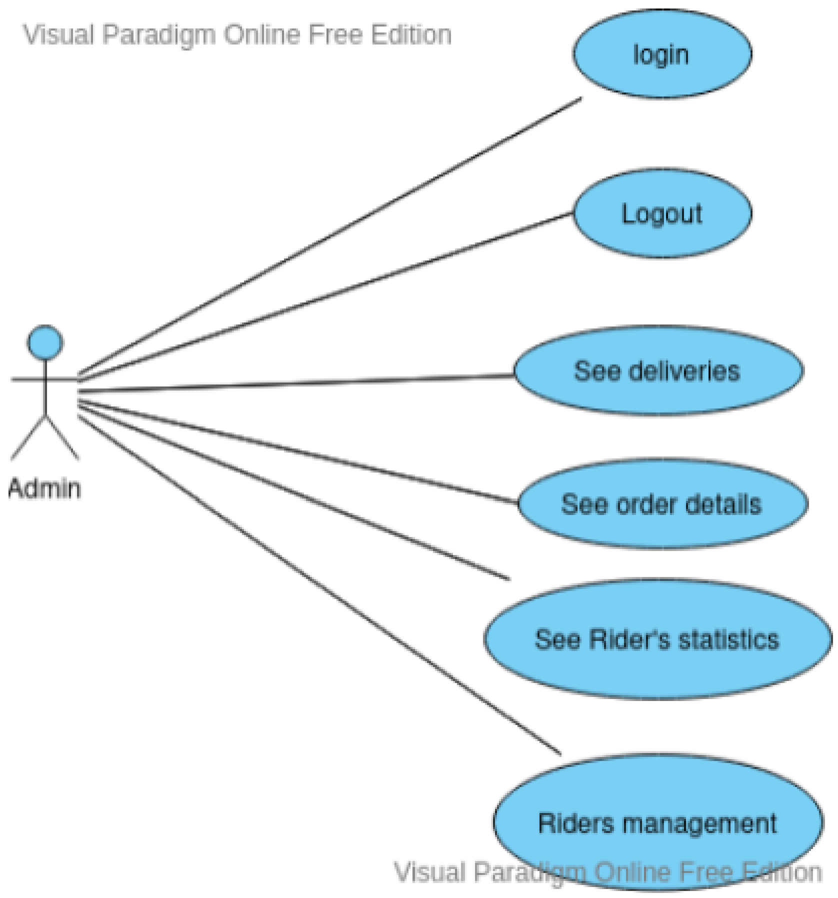
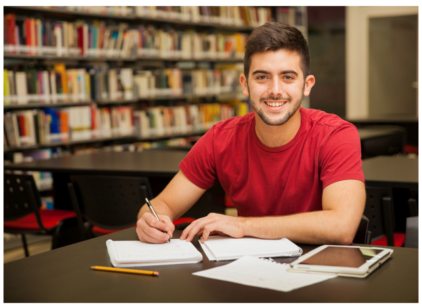


Fig. 3: Admin use case Diagram

## 2.2 Personas



**Persona 1**
Pedro is 20 years old and he is a Student in Aveiro's University. From one of his classes Pedro's professor asks him to read a book that fulfills the knowledge of class and will help for the exam. However, Pedro is very busy and going to a library will take a lot of time that Pedro doesn't wish to waste. So he wants a more efficient way to get the book he needs.

**Persona 2**
Maria is a 60 years old reformed lady. She loves to read new books, especially about cooking and gardening. However, the only time she is able to go to the library is at weekends when their son comes to visit her and she doesn't want to waste the time with their son this way. Since her mobility and their strength are not like the old days, she wishes for away that she could get the books she loves more frequently without need to dislocate to the library

**Persona 3**
Ernesto is a 35 year old and he is a librarian. Few and a few people are going to his library these days and he wishes that his books would reach more people in less time.

**Persona 4**
Barbara is 30 years old and has a motorcycle. She worked in a factory but she was tired of the routine so she quit. She wants a way to earn money with a flexible schedule and where she can ride her motorcycle.

**Persona 5**
Alexandre is 25 year old and has recently finished graduation and wants a job related to data analysis. He feels comfortable managing a team and working with the information of the company. He is used to technology and he is assertive, which makes him an independent and competent manager.

## 2.3 Main scenarios

**Pedro ask for a book delivery** - Pedro's teacher asks for the students to read a book that he recommends. Pedro wants to read this book but the travel to the bookshop and the search for the book that may not be in that store is not worth the short time available that he has. So he decides to install a mobile application so the book comes to him and he can work on other classes.

**Barbara sees the order details**- A client asked for a book delivery and Barbara is in charge of delivering that book. Barbara goes to the application to check the order details.

**Alexandre wants to manage riders**- Barbara wants to go to the gym and to the pool in the morning, so she would like to work in the afternoon. Alexandre goes to the application and sets the working schedule of Barbara to the afternoon.

## 2.4 Project epics and priorities

To implement our solution we used an agile methodology with weekly sprints. The project was concluded in 4 sprints and a half.

- **Sprint 1 (20/5 to 27/5)**
  - Definition of the product architecture
  - UI prototyping using figma
  - Backlog management using Jira
  - Outline the SQE tools and practices
- **Sprint 2 (27/5 to 3/6)**
  - **(EPIC)** The mobile App client can create an account and log in
  - **(EPIC)** The mobile App client can watch which books are available to make an order and see the books details
- **Sprint 3 (3/6 to 10/6)**
  - **(EPIC)** The mobile App client can order a book
  - **(EPIC)** The mobile App client can search for a book
  - **(Epic)** The rider has a functional mobile application
  - **(EPIC)** The mobile App rider can create an account and log in
- **Sprint 4 (10/6 to 17/6)**
  - **(EPIC)** The mobile App rider can see which delivery request exists and see the details
  - **(EPIC)** The mobile App rider can accept a delivery and conclude it
  - **(EPIC)** The web application admin can log in and see the delivery details
- **Sprint 5 (17/6 to 21/6)**
  - Clone the rider and client web application into a mobile application
  - **(EPIC)** The rider will notify the user when the delivery is accepted and is conclude
  - **(EPIC)** The admin web app can log in an account
  - **(EPIC)** The admin can see all the information of the system

In this sprint was also concluded this issues:

1. **(Issue)** As a Rider I want to get the order with the Store Nearest To Me

2. **(Issue)** As a user I want to be able to access the application even if there is a big amount of traffic, so that I can use it normally without problems or delays.

## 3 Domain model

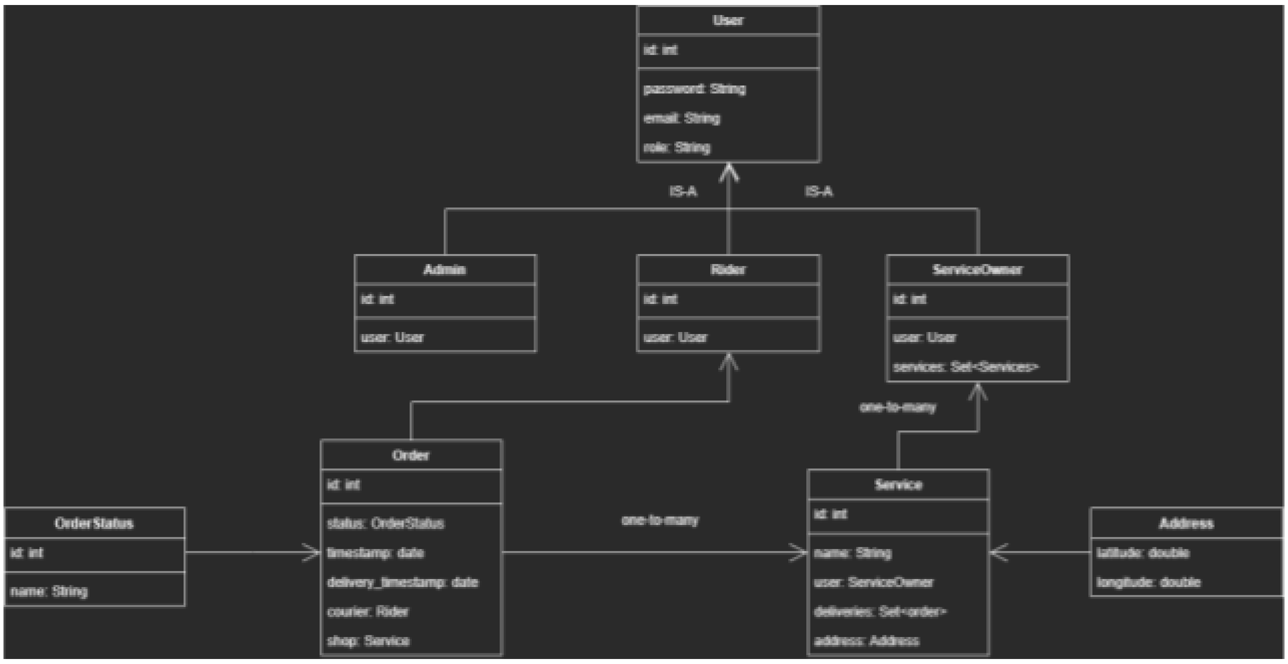


Fig. 3: Deployment diagram of the delivery engine

As we can see in the figure 3, the delivery engine has three types of users, the Admin, the Rider and the ServiceOwner, each one with an id, an email, a password and a role that distinguish them from each other. Besides, the ServiceOwner has a set of Services. The Service has an id, a name, a ServiceOwner, an Address, which is composed of the latitude and the longitude, and a set of Orders. Finally, an Order has an Id, a status, which has an Id and a name, a timestamp, a delivery timestamp, a courier and a shop.

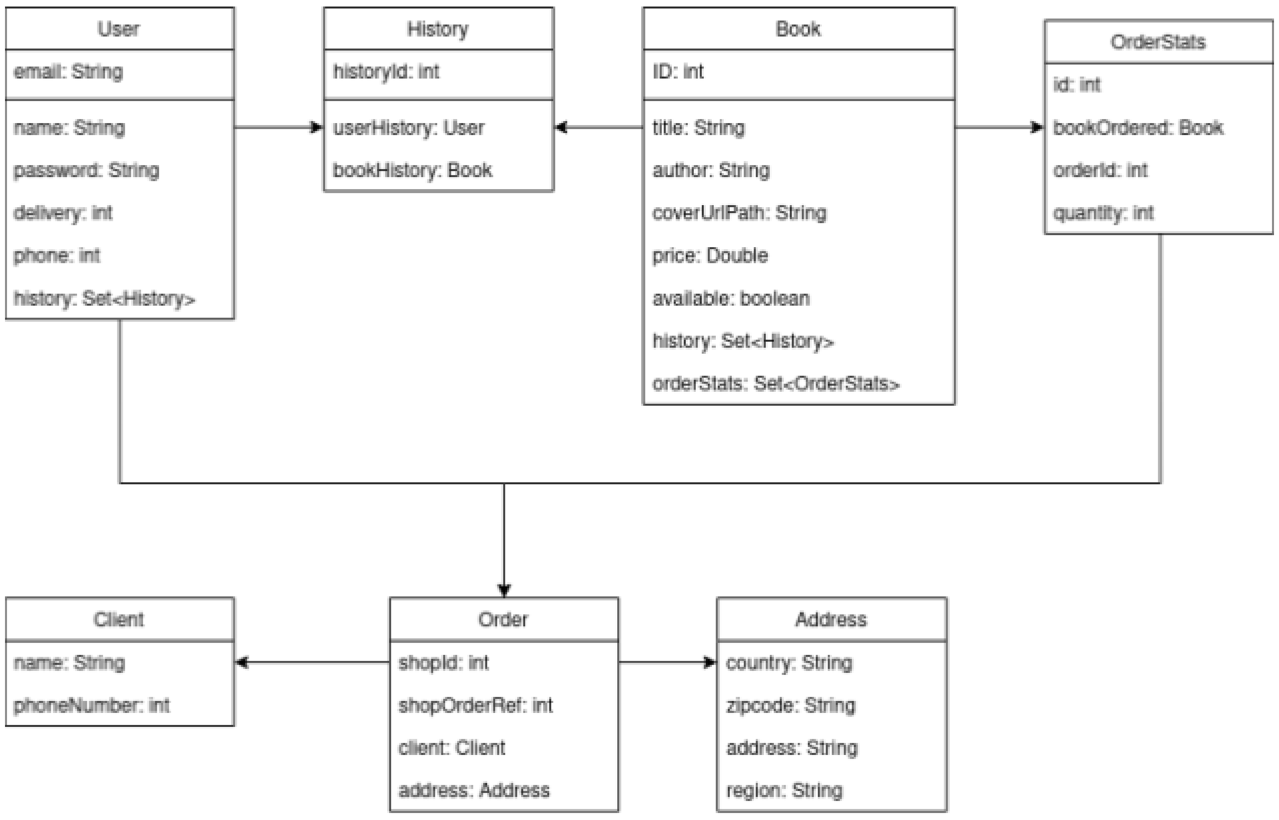


Fig. 4: Service API Domain Modal

In the Service API, as we can see in the figure 4, we have 3 core entities: the user, the book and the order. The User has an email, a name, a password, a phone, a delivery id and a history of the books purchased. The Book has an id, a title, an author, an url path, a price, a boolean to check if it is available, a history and order stats. An Order has a shopId, an order ref, an address and a client. The client is an entity that is used to create an order and it has a name and a phone, and the address has a country, a zip code, an address and a region. Finally the order stats has an id, a book, a quantity and an order id.

# 4 Architecture notebook

## 4.1 Key requirements and constraints

Non-Functional Requirements

- Scalability - The application should support many users using it at the same time
- Reliability - It shouldn't frequently crash
- Availability - Always available to any user
- Usability - Intuitive application

Functional Requirements

- The deliveries engine and the Business initiative should be hardware isolated and communicate through API
- Should register a Client
- Should register a Rider
- Should register a Library
- A client should be able to ask for a delivery
- The Rider and the library should be notify when a order occurs and be able to see it details
- A library should be able to add books to the system database and a client should be able to ask for a delivery of that book
- The library and the client should be able to see the book status (not available, available)
- A client should be able to see the delivery status (pending, on going, arrived)
- The system must ensure complete protection of data from unauthorized access. All remote accesses are subject to user identification and password control.
- The system will be implemented as a client-server system. The system must operate on Pcs and Smartphones.

## 4.2 Architectural view

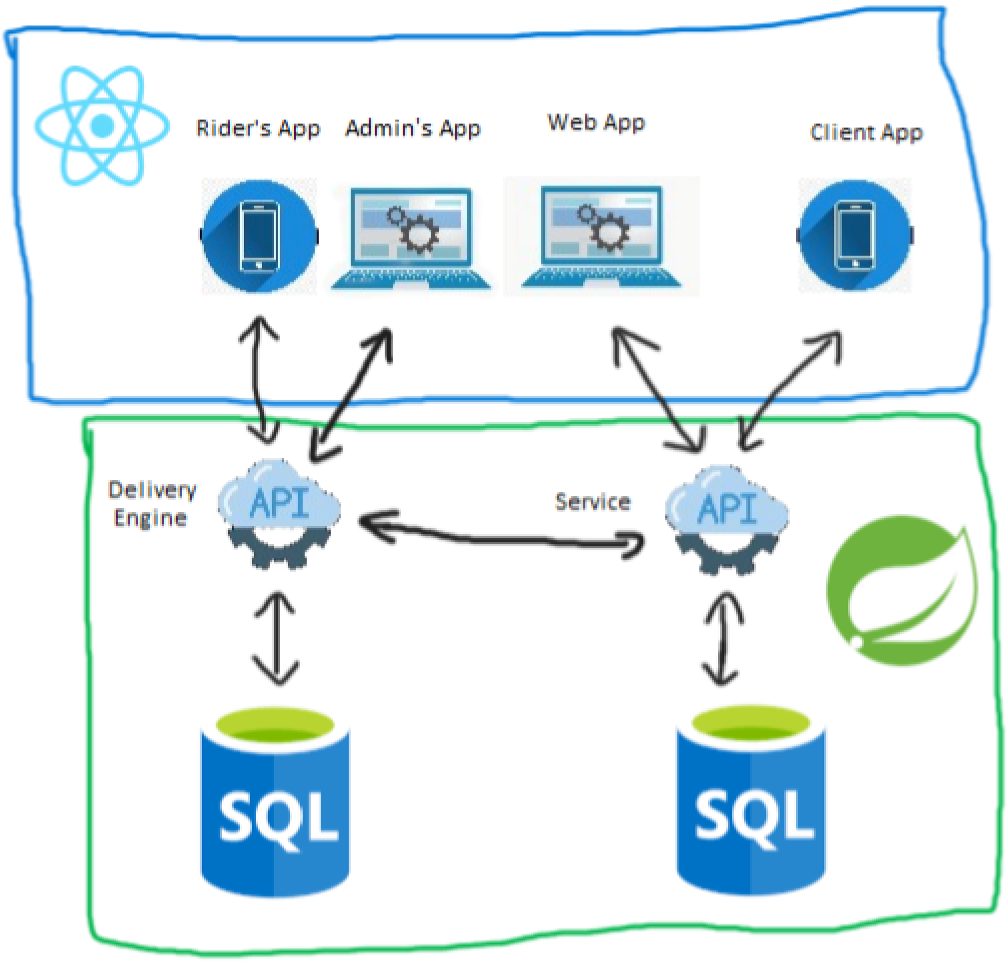


Fig. 5: Architecture diagram

As we can see in the figure 5, the system will be centered on the APIs that will be built using spring-boot for both of them. The UI will be built using React for the Web Apps and React Native for the Mobile App and the Rider's App. For the data storage, we'll be using SQL.

The Service API will manage the User's and the Book's information, so will handle the Mobile App and the Web App, while the Delivery Engine will handle the Admin Web App and the riders app hence, will manage the Rider's and delivery's information.

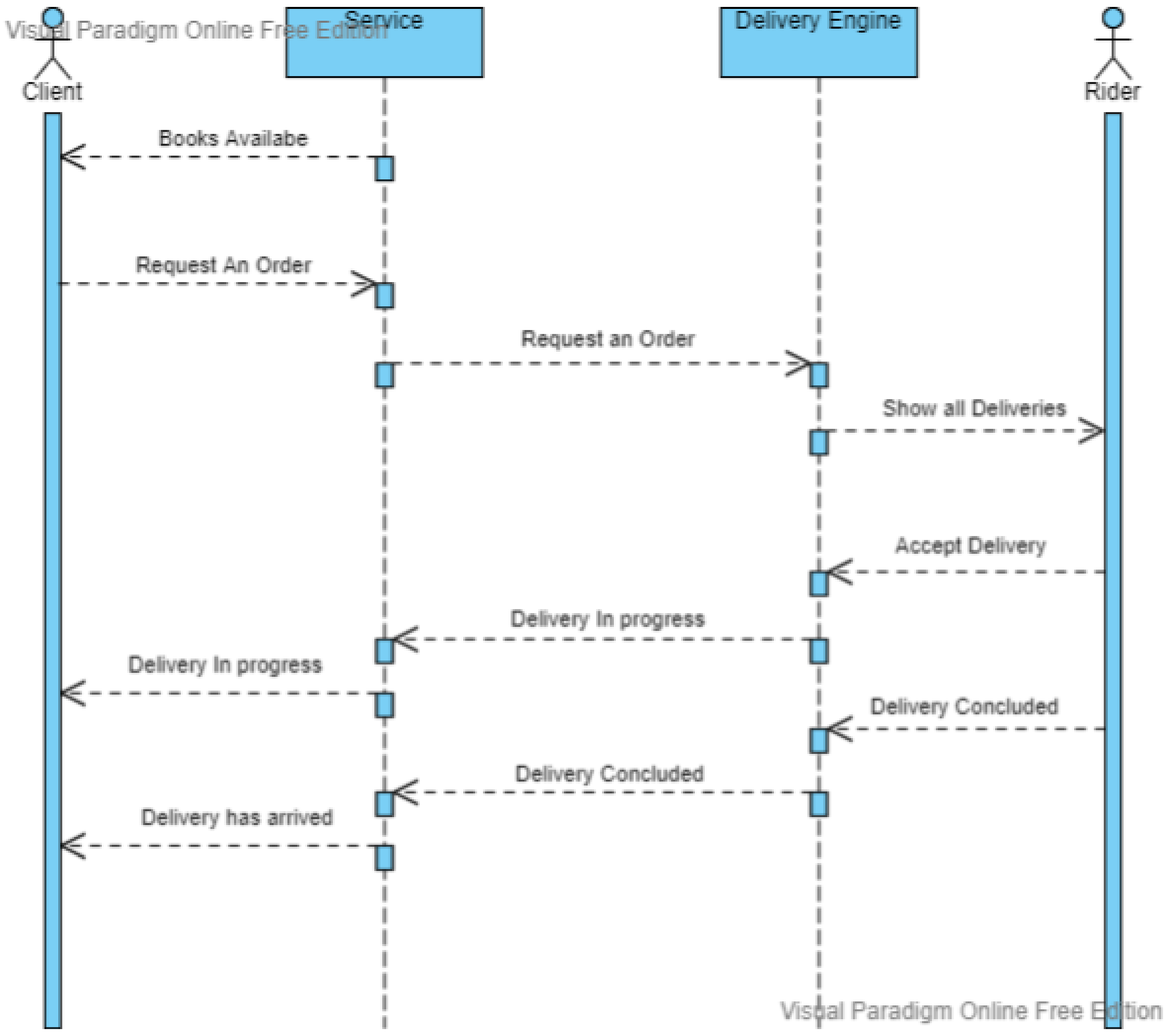


Fig. 6: Sequence Diagram of interaction between Client and Rider

As we can see in Figure 6, the service api will start by sending the client every product available in the database. If a user likes a book and requests a delivery order, the Service API will send this request to the Delivery Engine and, consequently, will send it to a nearby Rider. The Rider will accept the delivery and the delivery engine will notify the Client that his delivery is in progress. When the Rider arrives the system will also notify the Client.

Finally, the admin web app will show all information of the system to the manager, including users and riders accounts, books and order details and shop information.

## 4.3 Deployment architecture

[Explicar a organização prevista da solução em termos configuração de produção (deployment). Modelar num diagrama de deployment]

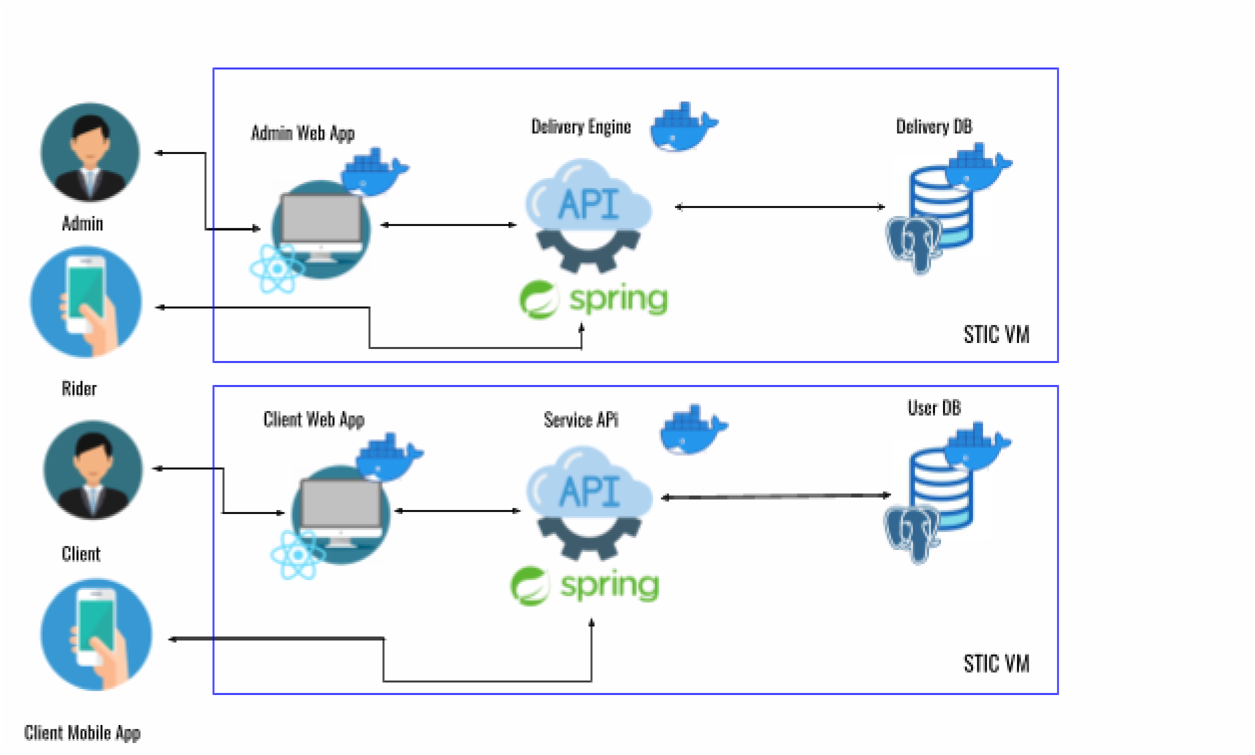
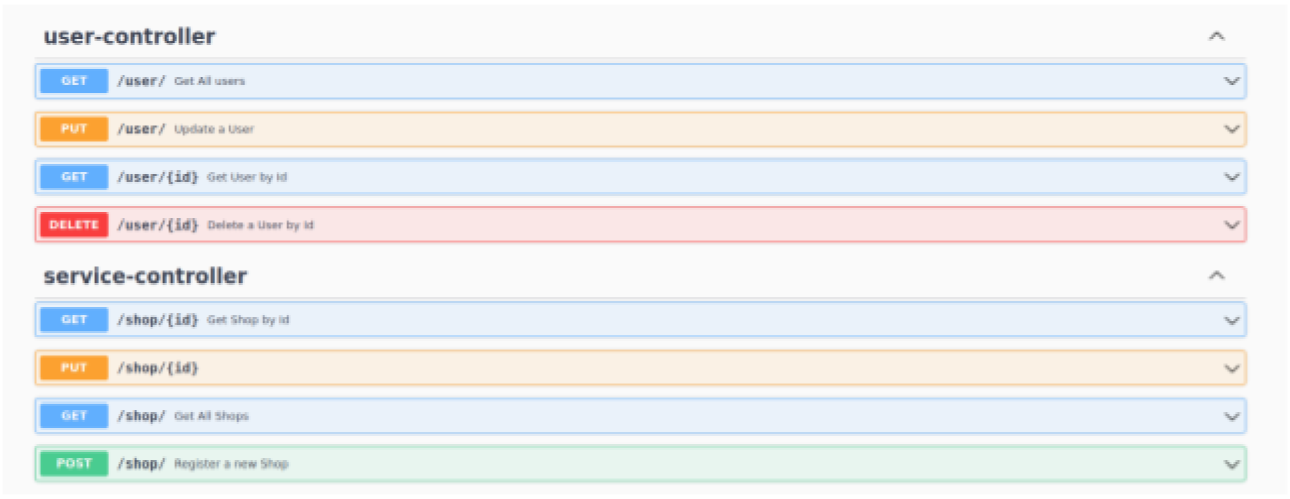


Fig. 8: Deployment Diagram

As demonstrated in figure 8, the deployment was made with a VM of STIC (Serviços de Tecnologias de Informação e Comunicação) from Aveiro's University of which were executed the following services: database, web application with help from docker containers and APIs. This way, the initial deployment as well as continuous deployment were facilitated, since we just need to rebuild the services changed by starting them again in a container with the new changes.

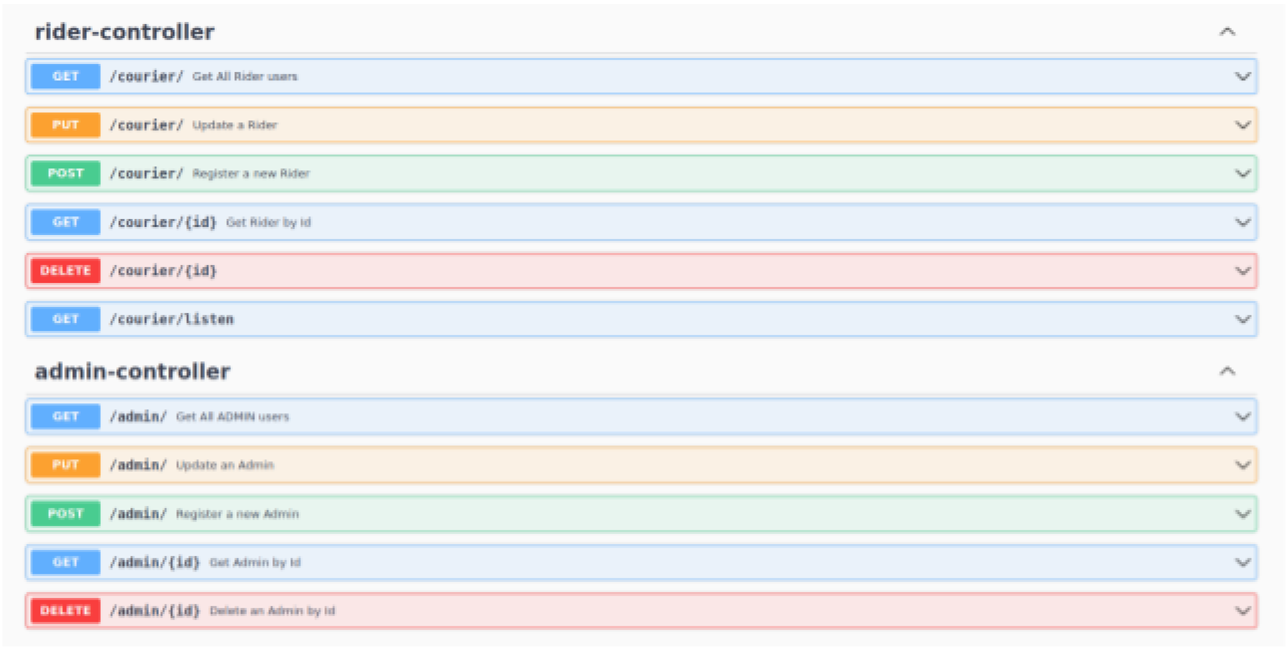
# 5 API for developers


user-controller
GET /user/ Get All users
PUT /user/ Update a User
GET /user/{id} Get User by Id
DELETE /user/{id} Delete a User by Id

service-controller
GET /shop/{id} Get Shop by Id
PUT /shop/{id}
GET /shop/ Get All Shops
POST /shop/ Register a new Shop

Fig. 9: Delivery engine API endpoints for users and shops

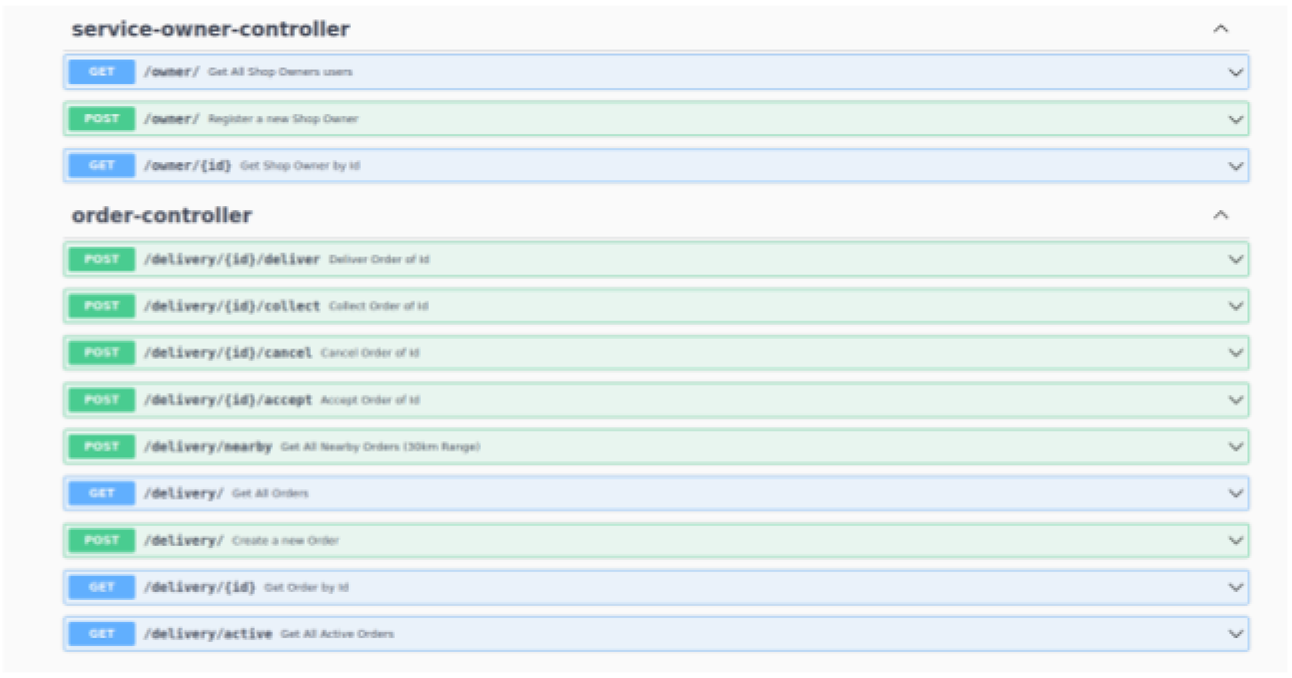
As we can see in the figure 9, the Delivery Engine API has four endpoints very basic and self-explanatory for users and shops. An endpoint to get all users, to update a user, get a user and delete a user by id. For the shop endpoints, we also have an endpoint to get a shop by id, update a shop, get all shops and create a new one.


rider-controller
GET /courier/ Get All Rider users
PUT /courier/ Update a Rider
POST /courier/ Register a new Rider
GET /courier/{id} Get Rider by Id
DELETE /courier/{id}
GET /courier/listen

admin-controller
GET /admin/ Get All ADMIN users
PUT /admin/ Update an Admin
POST /admin/ Register a new Admin
GET /admin/{id} Get Admin by Id
DELETE /admin/{id} Delete an Admin by Id

Fig. 10: Delivery Engine API endpoints for Riders and admins

In figure 10, the admin endpoints are also very basic and self-explanatory (create, get all, get one, update and delete an admin) and the rider has similar endpoints with the same goal, except for the GET endpoint /courier/listen that has the feature of creating a web socket to communicate with the user the status of the delivery (Not Implemented, was part of an initial solution).


service-owner-controller

GET /owner/ Get All Shop Owners users

POST /owner/ Register a new Shop Owner

GET /owner/{id} Get Shop Owner by Id

order-controller

POST /delivery/{id}/deliver Deliver Order of Id

POST /delivery/{id}/collect Collect Order of Id

POST /delivery/{id}/cancel Cancel Order of Id

POST /delivery/{id}/accept Accept Order of Id

POST /delivery/nearby Get All Nearby Orders (30km Range)

GET /delivery/ Get All Orders

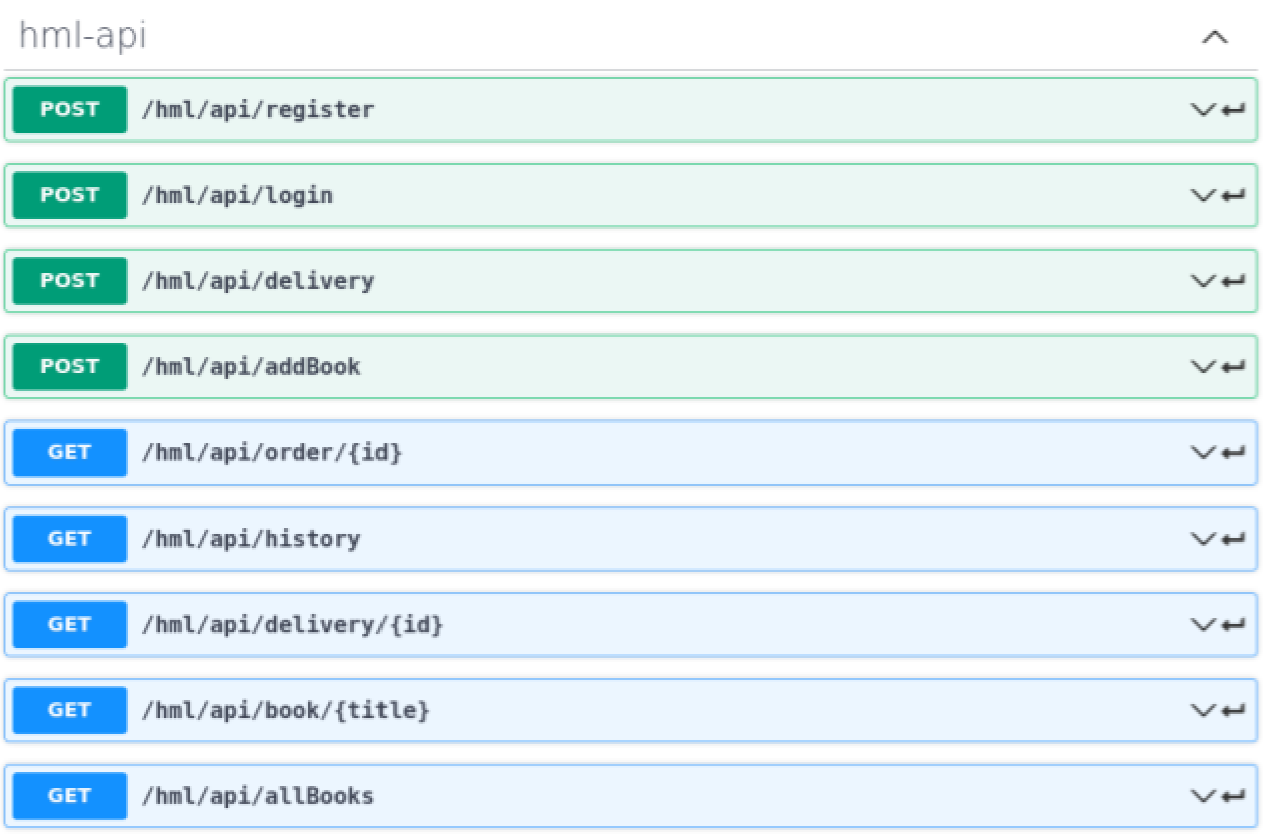
POST /delivery/ Create a new Order

GET /delivery/{id} Get Order by Id

GET /delivery/active Get All Active Orders

Fig. 11: Delivery Engine API for shop owners and delivery orders

In figure 11, the order controller has several endpoints. More in detail, an endpoint to get all orders (GET /delivery/) and get all orders in a range of 30km (POST /delivery/nearby), querying the postionstack [2] API. This endpoint, despite being more suitable to be a GET request, is a POST request due the rider's need to send their coordinates and we couldn't accomplish that feature with a GET request, so we changed to POST. Moreover this controller has an endpoint to create an order, get an order by id and get all active orders. Finally, it has endpoints for a rider to accept and deliver a delivery order and for a user to cancel and collect an order by the order's id.


hml-api

POST /hml/api/register

POST /hml/api/login

POST /hml/api/delivery

POST /hml/api/addBook

GET /hml/api/order/{id}

GET /hml/api/history

GET /hml/api/delivery/{id}

GET /hml/api/book/{title}

GET /hml/api/allBooks

Fig. 12: Service API endpoints

Lastly, the Service API, as we can see in figure 12, has 9 endpoints. The POST endpoints are the login and the register of users, the addition of a new book and the creation of a delivery order. The Get endpoints are to get all books, get a book details by his title, get a delivery by id and get the history of the books purchased by a user.

## 6 References and resources

<document the key components (e.g.: libraries, web services) or key references (e.g.: blog post) used that were really helpful and certainly would help other students pursuing a similar work>

[1] Swagger (software) https://en.wikipedia.org/wiki/Swagger_(software), disponível Online, acedido a 19 de junho, 2022.

[2] PositionStack (service) https://positionstack.com/