TQS: Product specification report

Daniel Francisco [98188], Henrique Sousa [98324], Nuno Fahla [97631], Tiago Costa [98601]
v2022-06-23

1 Introduction

1.1 Overview of the project

Our app is a generic delivery system that can connect to any store, which provides the stores with an easy and straightforward method of delivery. Together with this, we will be creating a musical instrument and vinyl store, to connect to the delivery system and showcase how it works. Our goal with this project is to learn more about TDD, software testing, and CI/CD tools, which will all be used during the project, as well as make a good system that could be used to manage deliveries.

1.2 Limitations

We planned to implement a mobile app for the Rider User by making use of an Angular module to create a progressive web app, but due to time and availability constraints, this feature was never fully implemented. For the Music Shop app, it was intended to exist as an Administrator type of user, for owners of the shop, which would serve as an accessible way to add or remove products. A transaction entity was also planned, it would contain attributes like date, state, method of payment, etc, but this feature was considered less important.

For both APIs we also had planned a feature to show the rider/user delivery history.

2 Product concept

2.1 Vision statement

Our Delivery System aims to provide stores with an easy and straightforward method to deliver their products right to the customer's door. Delivering products to the customers' door is something that is becoming more and more common nowadays, but that is difficult to be done by small stores which doesn't make sense to invest in a delivery area and that's where our system comes in handy, having multiple drivers working for our service will make it possible to deliver products from a lot of stores to the customer in a matter of minutes and the stores won't have to worry about that.

Our system is similar to the well-known UberEats app, but instead of just food and groceries, our system is open to any type of store that wants to deliver its products within minutes.

2.2 Personas

Name: Thomas William
Profession: Musician
Age: 39
Single
Thomas lived his life focusing mostly on music. Ever since he was young, he had a talent that developed into a great career over the years. Ever since he broke his flute last week, he has been trying to get a new one to fill his instrument collection.

Name: Willson Grey
Profession: Delivery driver
Age: 26
Wilson is married and his wife is pregnant. He was unemployed and started working as a delivery driver for our delivery service.

Name: Gary Johnson
Profession: Delivery App Manager
Age: 57
Gary is married and has 3 kids. He is the Manager of the Delivery App.

2.3 Main scenarios

Thomas buys a flute:

- Thomas opens the application and sees a list of products for sale. He searches for flutes and sees a list of flutes he can purchase. After deciding on a flute, he adds it to the cart and opens the cart. There he finishes the purchase and receives the details about the order.

Willson completes a delivery:

- Willson opens the delivery application and logins with his account. He then sees a list of deliveries that he could complete. He chooses the one he wants to do the most, drives to the shop to get the product and then to the destination to deliver it. After delivering the product, he marks the delivery as completed.

Gary manages the deliveries

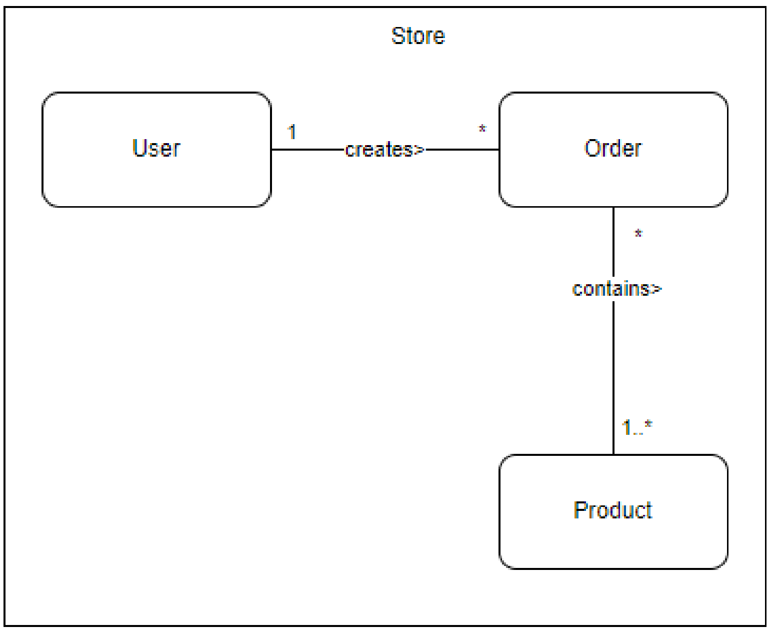
- Gary opens the delivery application and logs in with his account. He then sees a statistics page detailing how the delivery system has performed. On another page, he can find the currently active and delivered orders. Based on this information, Gary decides how to act to improve the deliveries.

2.4 Project epics and priorities

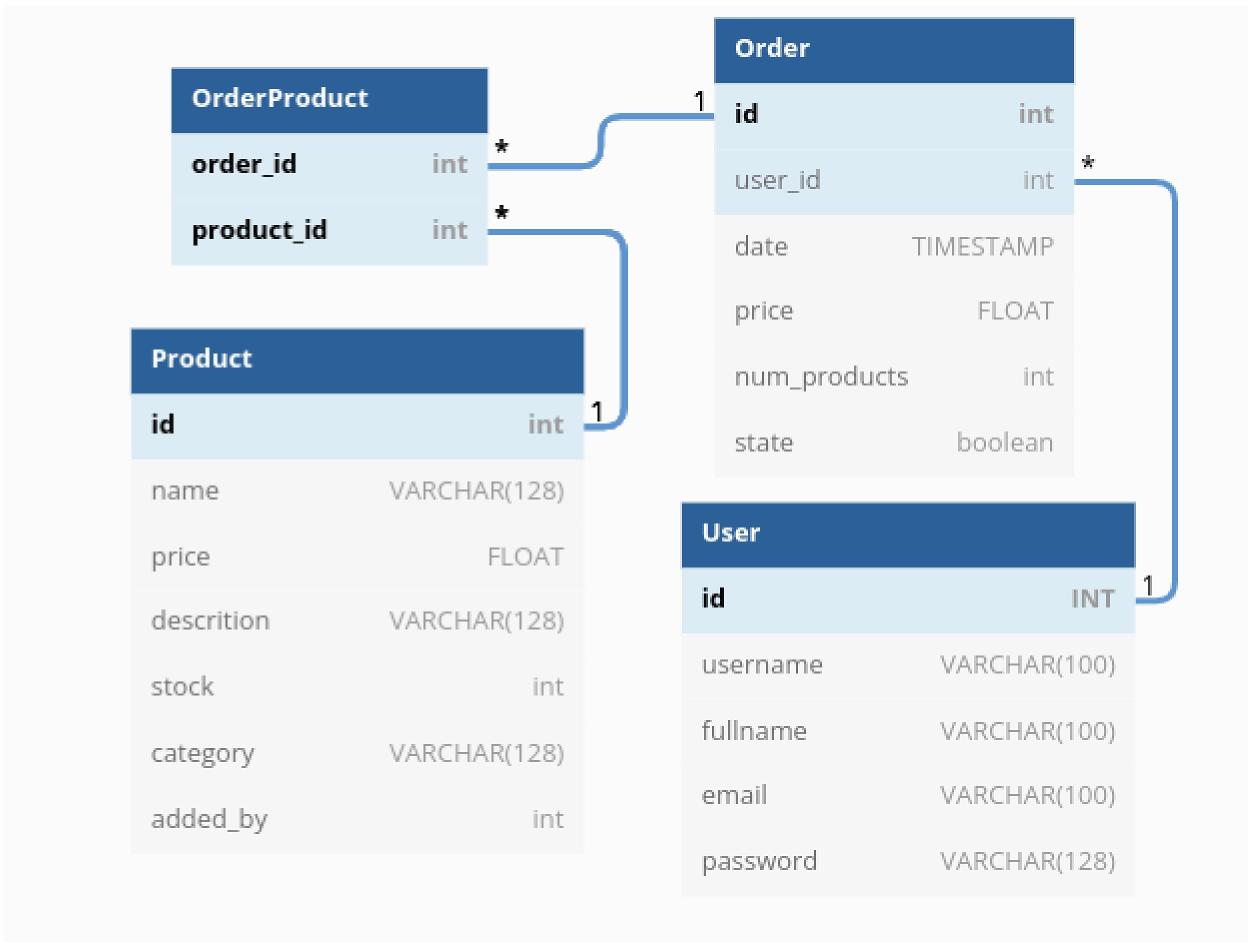
To implement our solution we used an agile methodology with weekly sprints. The project was concluded in 5 sprints.

- **Sprint 1 (19/5 to 26/5)**
  - Definition of the product architecture
  - UI prototyping using angular
  - Backlog management using JIRA
  - Outline the SQE tools and practices
- **Sprint 2 (26/5 to 2/6)**
  - **(EPIC)** The music store client can view and add a product to cart on the web app
  - **(EPIC)** The delivery app client can check deliveries on the web app.
- **Sprint 3 (2/6 to 9/6)**
  - **(EPIC)** Create CI Pipeline
  - **(EPIC)** Create CD Pipeline
  - Create Deliveries Backend
  - Create Music Store Backend
- **Sprint 4 (9/6 to 16/6)**
  - **(EPIC)** Dockerize Applications
  - **(EPIC)** Communication between services from RabbitMQ
  - Finalize frontend implementations with backends
- **Sprint 5 (16/6 to 23/6)**
  - **(EPIC)** The rider can change the delivery status
  - The admin web app can log in an account
  - **(EPIC)** The admin can check all deliveries information
  - **(EPIC)** A user has the most recent information about the delivery status

3 Domain model

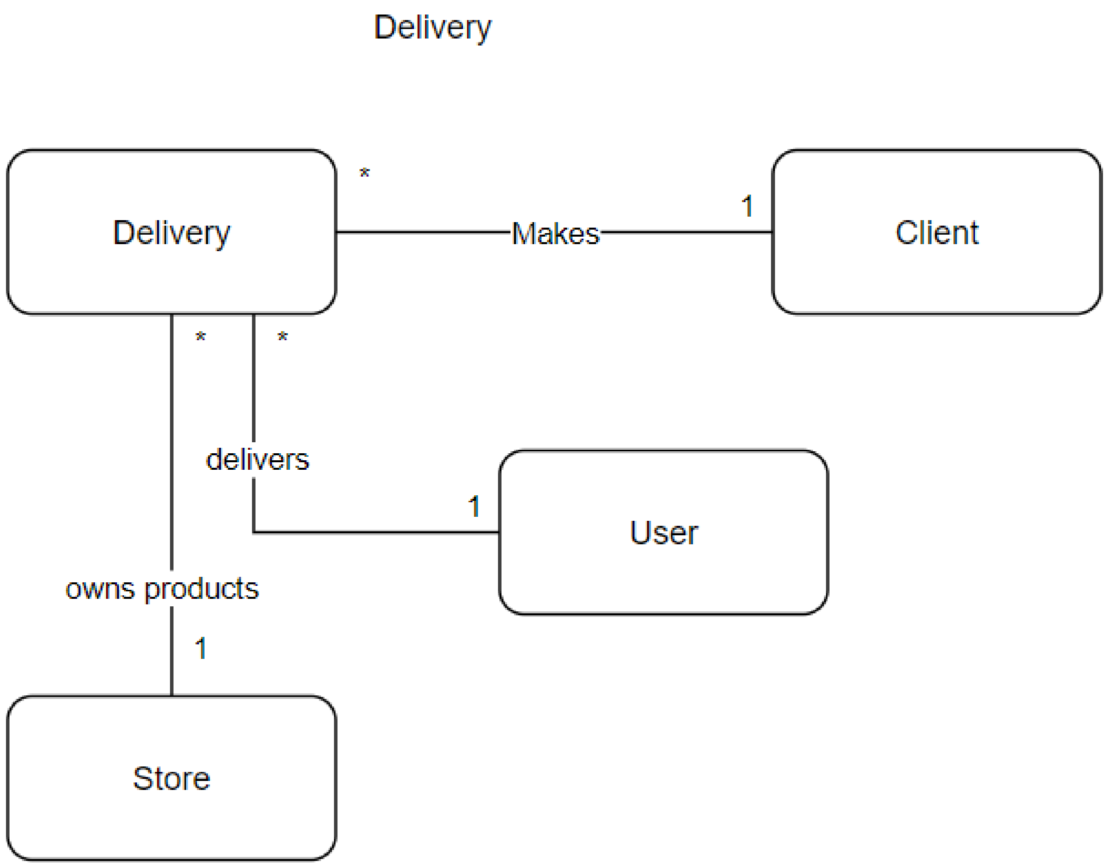


Music Store



In our project, we will be dealing with 2 domains, with the first one being the store domain. In this domain, we will have Users that can Order a set of Products. When a User finishes an Order, a Delivery is generated on our other domain, the delivery domain.

Delivery Service



In the delivery domain, we have Deliveries that are made by Clients that finished their order in the Store. This Delivery is going to be assigned to a rider User, who will retrieve and deliver the products.

4 Architecture notebook

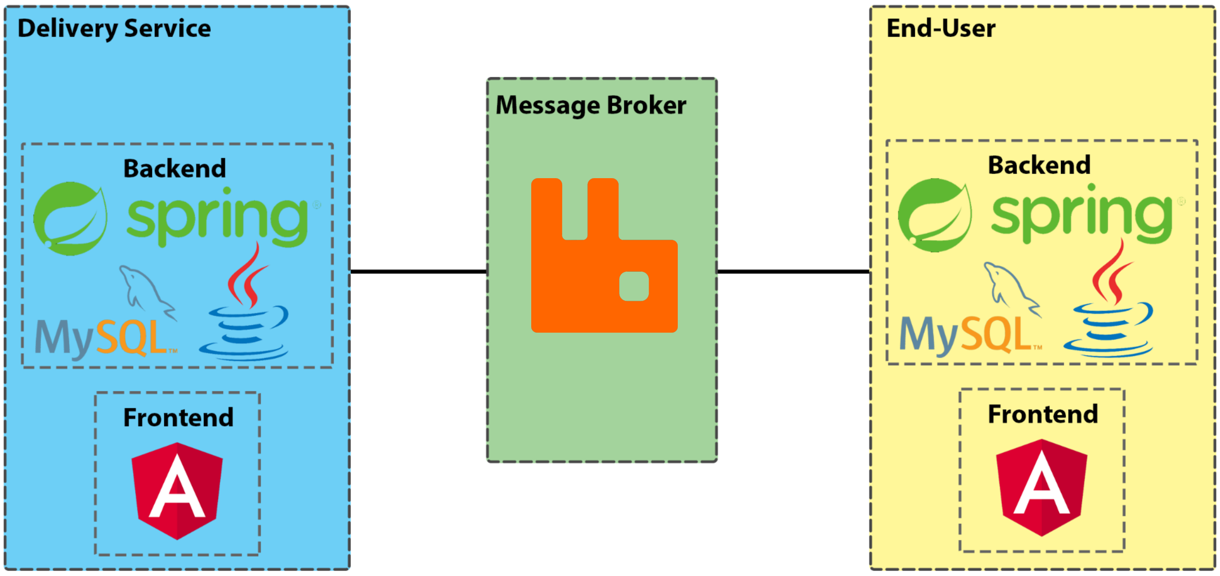
4.1 Key requirements and constraints

While analyzing the idea for the system, we found these key requirements and constraints that significantly impact our architecture:

- The system must be able to connect to any store, no matter the product they sell or the architecture they use on their applications.
- The system must be accessible anywhere in the world through a browser, for all riders and managers.

- The system must be easy to maintain and update, so it doesn't disrupt the deliveries when it's updated.
- The system must be able to handle hundreds of requests per second.

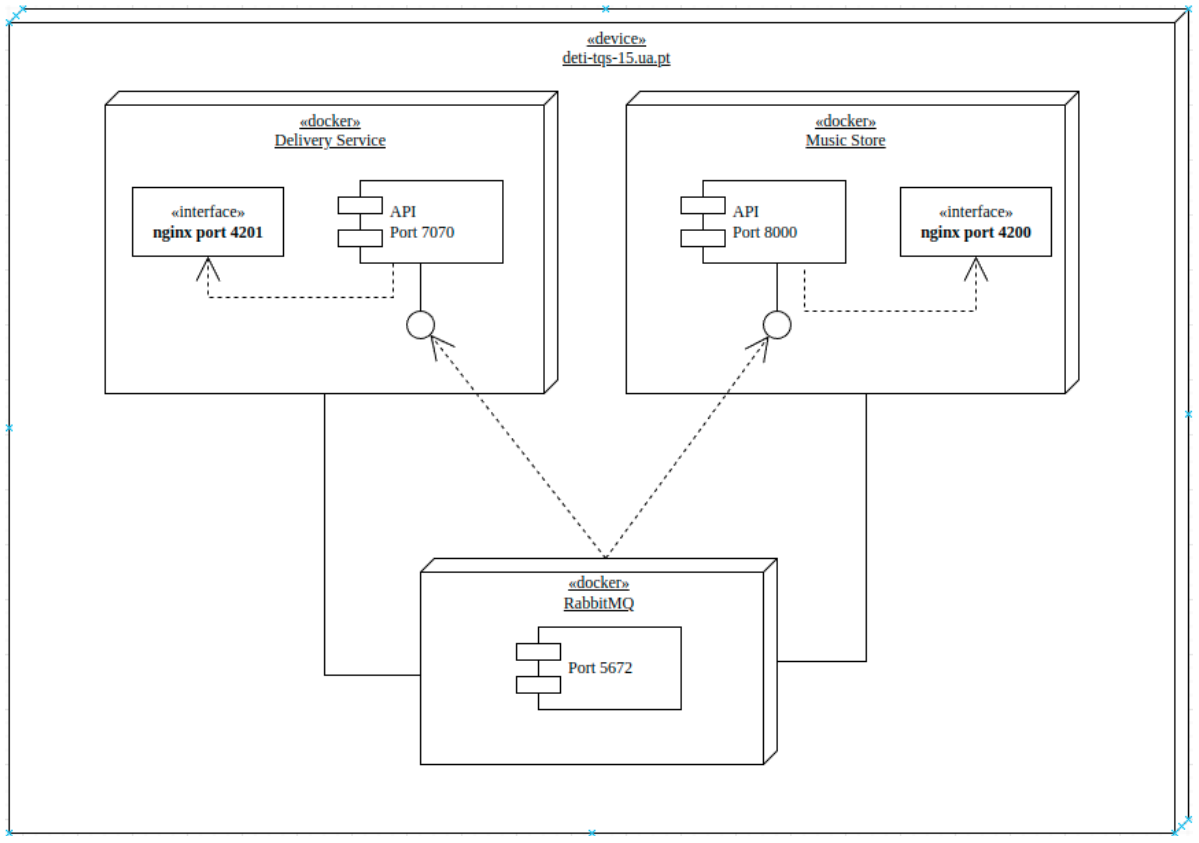
4.2 Architectural view



Our project is going to have 2 applications, both made with similar technologies, connected with a message broker. Both backends will use Spring Boot, a MySQL database, and java as their programming language, while the frontends will both be made with Angular. The message broker will be made with RabbitMQ.

The end-user system is going to send messages through the message broker to the delivery service, to start a new delivery, cancel a delivery, ask for updates on the driver's location, and more. The opposite will also happen, the delivery service will be able to send messages to the end user system, for more urgent situations, like updating deadlines if there's been an accident or change in driver.

4.3 Deployment architecture

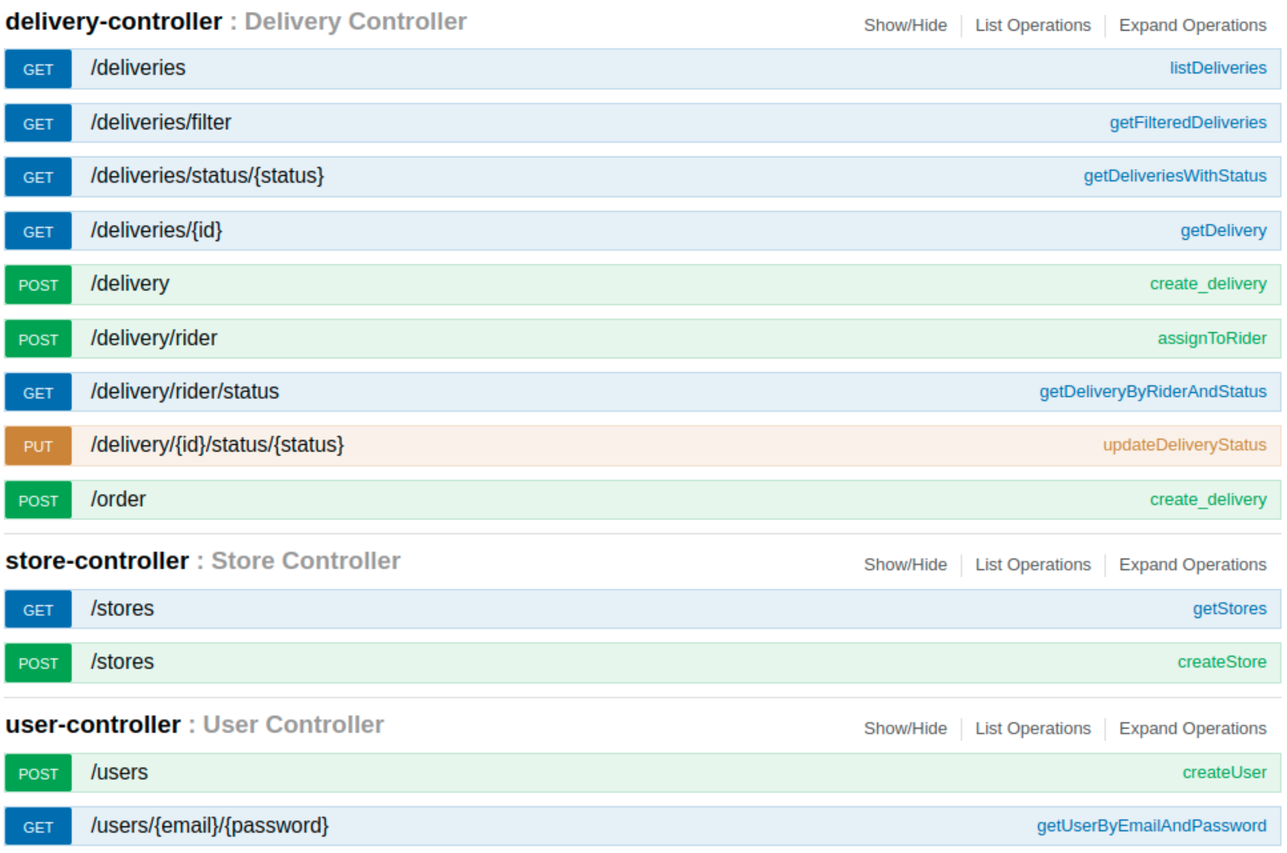


5 API for developers

Delivery Service API

https://app.swaggerhub.com/apis-docs/sousaUA/delivery-service-api/1.0

The API for the Delivery Service was divided into three controllers: one for the deliveries, one for the Stores and the last for the users. The delivery controller allows users to get deliveries, filter them, get their status, lets the users create a delivery and assign deliveries to a specific rider as well as update the delivery status both on the deliveries app and on the store where the order was made. They can add and get stores with the store controller. For the user operations, there are endpoints to create and get users.



delivery-controller : Delivery Controller — Show/Hide | List Operations | Expand Operations

GET	/deliveries	listDeliveries
GET	/deliveries/filter	getFilteredDeliveries
GET	/deliveries/status/{status}	getDeliveriesWithStatus
GET	/deliveries/{id}	getDelivery
POST	/delivery	create_delivery
POST	/delivery/rider	assignToRider
GET	/delivery/rider/status	getDeliveryByRiderAndStatus
PUT	/delivery/{id}/status/{status}	updateDeliveryStatus
POST	/order	create_delivery

store-controller : Store Controller — Show/Hide | List Operations | Expand Operations

GET	/stores	getStores
POST	/stores	createStore

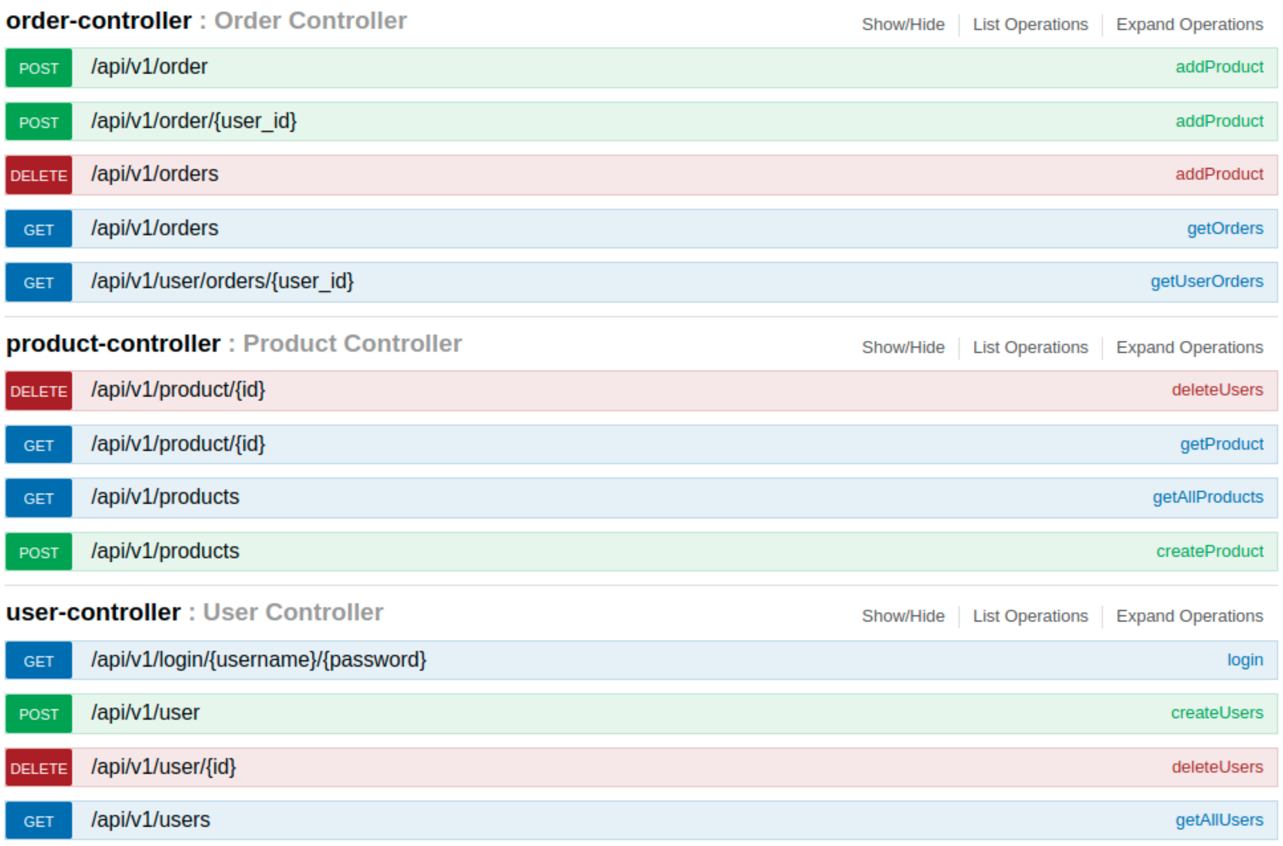
user-controller : User Controller — Show/Hide | List Operations | Expand Operations

POST	/users	createUser
GET	/users/{email}/{password}	getUserByEmailAndPassword

Music Store API

https://app.swaggerhub.com/apis-docs/sousaUA/music-store-api/1.0

The API for the Music Store was divided into three controllers: one for the orders, one for the products and the last for the users. The orders allow a user to add an order by sending a JSON body with an optional user id in the path, they can delete orders too and get all orders, with a filter for specific user ids. They can delete and add products, get a list with all the products or get a product by its id. For the user operations, there are endpoints to create, delete and get users and an endpoint that allows a user to check if the login details are correct.



order-controller : Order Controller — Show/Hide | List Operations | Expand Operations

POST	/api/v1/order	addProduct
POST	/api/v1/order/{user_id}	addProduct
DELETE	/api/v1/orders	addProduct
GET	/api/v1/orders	getOrders
GET	/api/v1/user/orders/{user_id}	getUserOrders

product-controller : Product Controller — Show/Hide | List Operations | Expand Operations

DELETE	/api/v1/product/{id}	deleteUsers
GET	/api/v1/product/{id}	getProduct
GET	/api/v1/products	getAllProducts
POST	/api/v1/products	createProduct

user-controller : User Controller — Show/Hide | List Operations | Expand Operations

GET	/api/v1/login/{username}/{password}	login
POST	/api/v1/user	createUsers
DELETE	/api/v1/user/{id}	deleteUsers
GET	/api/v1/users	getAllUsers

6 References and resources

https://www.rabbitmq.com/documentation.html
https://www.jenkins.io/doc/book/pipeline/syntax/
https://www.jenkins.io/doc/tutorials/build-a-java-app-with-maven/
https://issues.jenkins.io/browse/JENKINS-39482?page=com.atlassian.jira.plugin.system.issuetabpanels%3Acomment-tabpanel&showAll=true