TQS: Project Specification Report

Team Coordinator - Tomé Carvalho 97939[1], Product Owner - Lucius Vinicius 96123[1], QA Engineer - Dinis Lei 98452[1], and DevOps Master - Camila Fonseca - 97880[1]

[1]DETI, University of Aveiro

June 23, 2022

Contents

1 Introduction

1.1 Overview of the project

In the context of the TQS subject, we aim to develop a viable software product, with functional specification, system architecture and implementation. Additionally, throughout our software engineering process, we will specify and enforce a *Software Quality Assurance* (SQA) strategy.

The project has two main objectives. The first one is to create a complete delivery engine (*Deliverize*), that captures the typical requirements of last-mile deliveries, featuring dynamic matchmaking of rides and riders. This engine should be usable by different end-user applications. It shall be used by businesses in order to submit, track and assess their delivery requests. The other objective is to create an implementation (*Zap*) of such a business.

1.2 Limitations

Even though most of the features were implemented there were still some that because of lack of time or being out of scope were left off.

In terms of business logic we have a very basic first come first served for choosing the riders, but we would like to have an algorithm to choose based on distance and other factors.

We would like to have a message broker to update the status of the delivery instead of having the Zap fetching the status.

And finally we wanted to have a better integration with the Rider's App.

2 Product concept

2.1 Vision statement

The *Zap* web app will be used by the business to manage the store's products and by users to place orders. *Zap* managers will be able to add new products and manage stock. Users will be able to place orders on products and keep track of them.

The *Deliverize* administration web app will be used for overview/analysis and management purposes. Managers will be able to access information such as riders' reputation and a performance dashboard. They will also have the ability to remove businesses from the engine.

The *Deliverize* Rider's app will be used by riders to accept orders placed through applications like *Zap*'s.

Zap is a store that sells electronic parts and tools such as cables, adapters, batteries, screws, screwdrivers, etc.

2.2 Requirements

2.2.1 Functional Requirements

The functional requirements of the *Zap* web app are:

- Managers can register Zap on the *Deliverize* engine.
- Managers can view and update the stock of items, as well as add new items.
- Users can search/filter items by name and category.
- Users can view the items' prices and order any available amount of them.
- Users can view the status of their deliveries. (Waiting for rider, in progress, delivered).

The functional requirements of *Deliverize* administration web app are.

- Managers can black companies from the Deliverize engine.
- Managers can access information about the riders' reputation and a performance dashboard.
- Riders can sign up.
- Riders can accept orders.

2.2.2 Non-Functional Requirements

Both Store and Rider side have the same non-functional requirements:

- The application must be user-friendly.

2.3 Personas

2.3.1 Caio Costela

Figure 1: Caio Costela

Caio Costela is a healthy 32 years old man who works as an electrical engineer. In the moment Caio is fixing a client's equipment, however he noticed that are missing adequate pieces to complete the process. Since Caio lives at considerably distance from the city's urban area, he can't buy those pieces somewhere close.

2.3.2 João Silva Pereira

Figure 2: João Silva Pereira

João Silva Pereira has 40 years old and he is the manager of Deliverize. He is the responsible for accepting companies on the app, and he uses an amount of the day to make analysis about the riders and the orders that are being made to verify if an company should or not go to the black list.

2.3.3 Diego Góis

Diego Góis is a 24 young adult who in the moment has problems to find some work, so he currently is making deliveries as a Deliverize's Rider. He waits until new orders in his region appears so that he could delivery the order from a company to a client.

Figure 3: Diego Góis

2.4 User Stories

2.4.1 Change Company Status

João Silva Pereira, the manager of Deliverize App, wants to check what companies are accepted, blacklisted or pended, and he wants to update the status for some of them. To realize that, he clicks on the "Companies" section on the navigation bar and then he can observe each Company status. Next, he wants to update a status from a specific Company, and he just clicks on the button right next to it.

2.4.2 Buy a product

Caio Costela wants to buy some missing pieces to fix his client's equipment. To accomplish that, he goes to the Zap website, does a log-in into his account, to see the list of products, he clicks on "Store" on the navigation bar. For each of his wanted pieces, he filters by its name, add the intended quantity and clicks on "Add to Cart".

After everything is bought, Caio goes to his cart and clicks on "Go to Checkout". In that page, he fill the form with his informations and clicks on "Check out". Caio then can see his order status in the website page's "Orders"

2.4.3 Deliver Package

Diego Góis wants to earn some money so he opens the app and checks the orders available. Then he checks the information about each order. After

some deliberation he clicks on the button accept to indicate that he will take on that order. He then proceeds to go to the store to pick up he items, and after confirming that every thing is correct he goes to the app and marks the request as 'picked up'.

He then arrives at the clients house and after delivering the items he inputs the clients code to confirm the success of the delivery.

2.5 Project epics and priorities

Each user story has an epic associated, and then divided in smaller more manageable tasks. Each one-week iteration tries to complete the tasks necessary to have the feature up and running. If the feature isn't completed in the iteration, it will be transferred to the next one and given an higher priority.

3 Domain model

The problem is divided in three main parts:

- The Zap part includes the ZAP API and the Zap Store frontend.
- Deliverize part includes the Deliverize Management and API.
- Lastly there is the Rider's part that has the Rider's mobile application

Each frontend communicates with its respective API to fetch the relevant information. The ZAP API sends requests to the Deliverize API to post Orders for Riders to pick up. The Rider app sends requests to Deliverize to get the Orders available

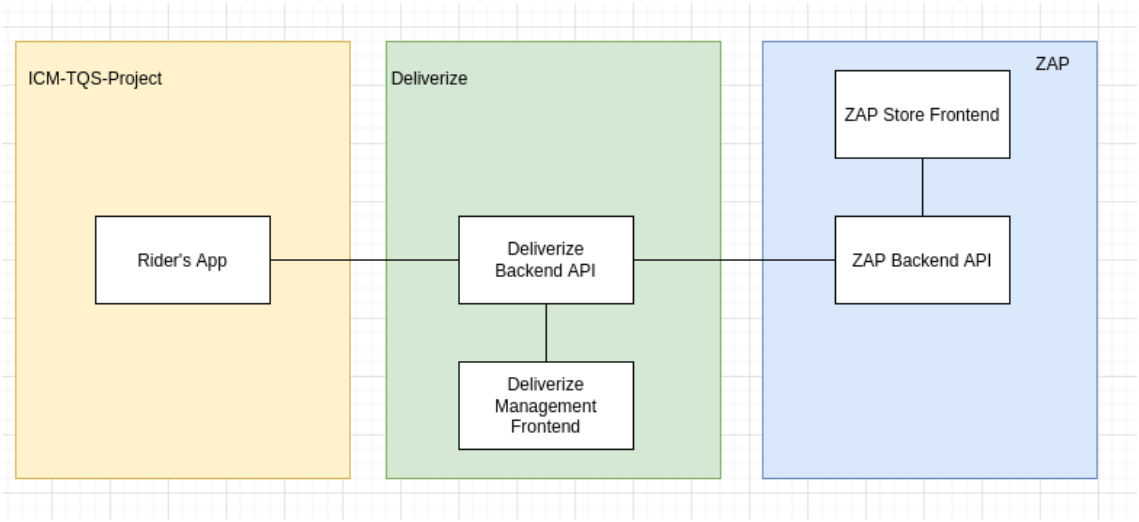

Figure 4: Domain Model

4 Architecture notebook

4.1 Architectural view

The main technology to develop the APIs was Spring Boot. Both Front Ends used the React Framework with MUI for customization.

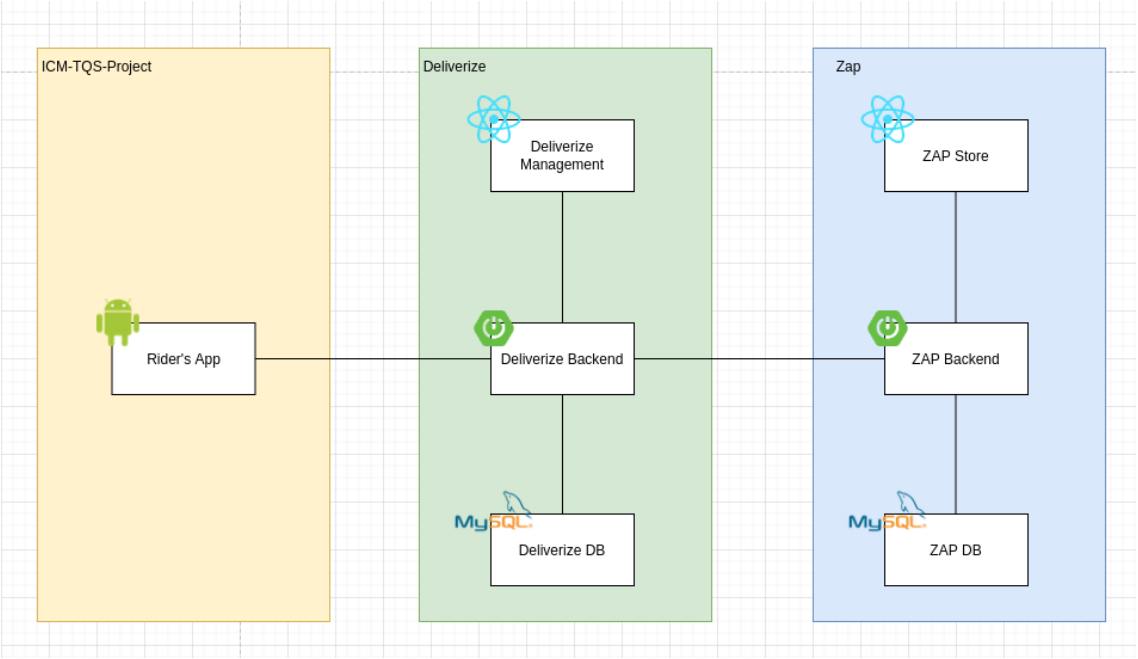

Figure 5: Architecture Diagram

In figure 6 we can see how the Clients and Riders interact with the system and in figure 7 how the Managers interact.

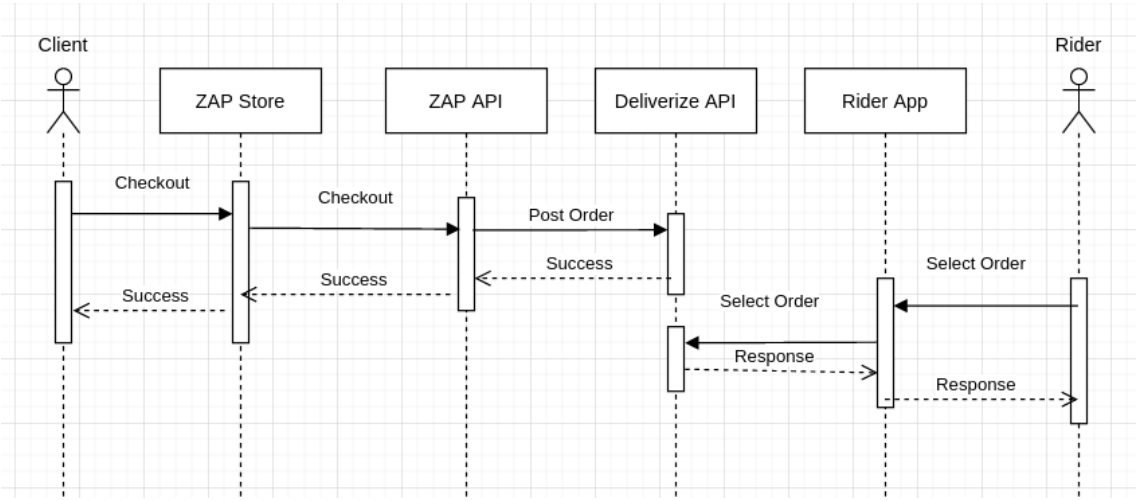

Figure 6: Checkout and Pick Order interactions

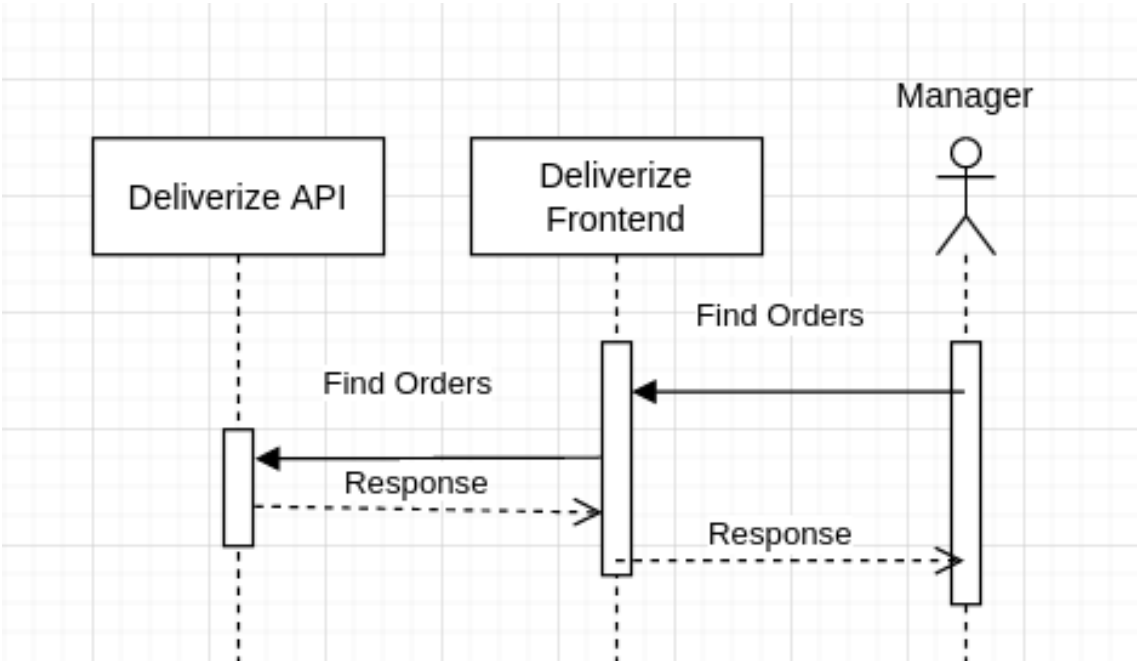

Figure 7: Find Order interaction

4.2 Deployment architecture

The deployment was made into an Azure Virtual Machine running several docker containers for each different component of the architecture. Continuous deployment was made using GitHub's Actions to push the new features directly to the VM. In figure 8 we can see the deployment architecture.

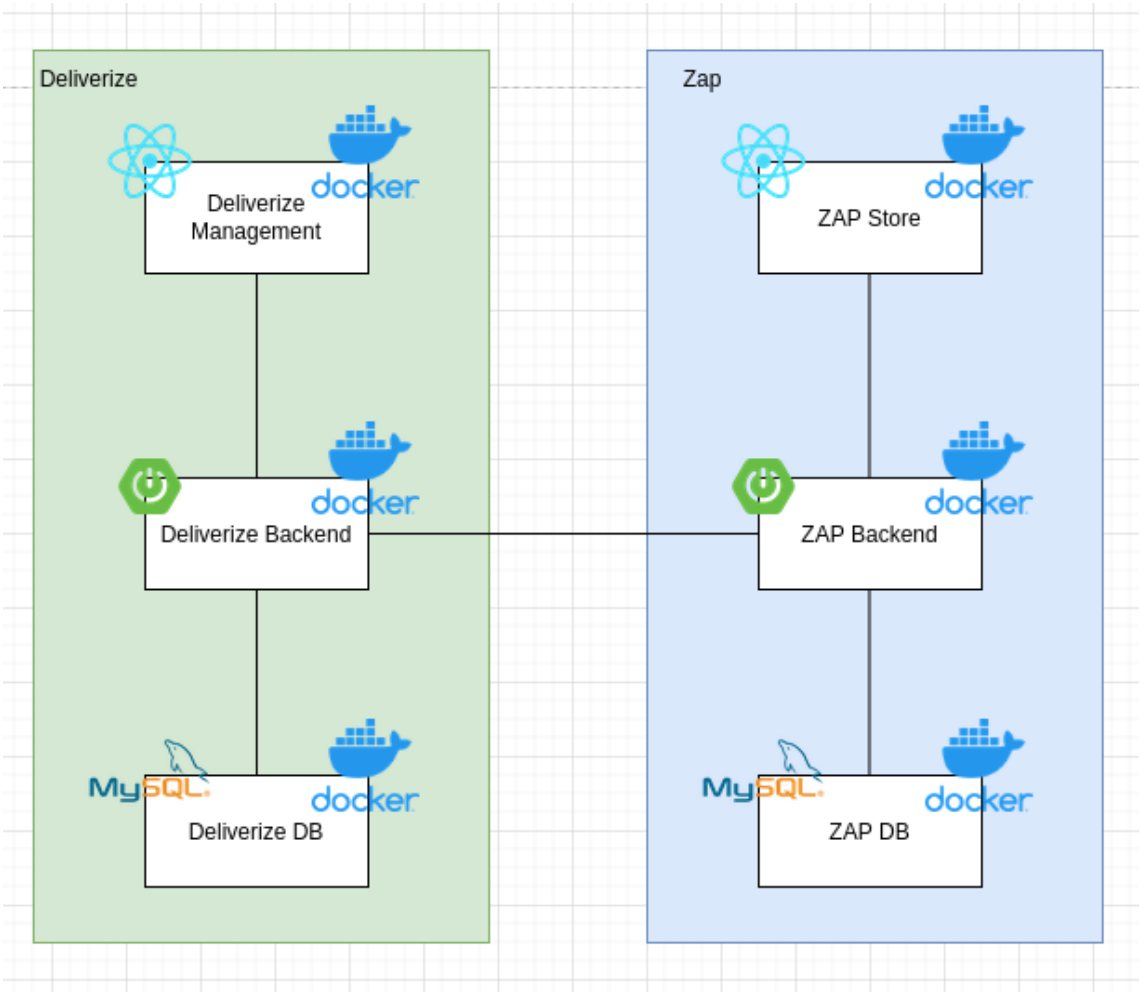

Figure 8: Deployment Architecture Diagram

5 API for developers

5.1 ZAP REST API

The API is divided in two parts, the User Authentication and Management (UAM) and the Store Logic (SL).

The UAM manages all the authentication and authorization on the website and has the following endpoints:

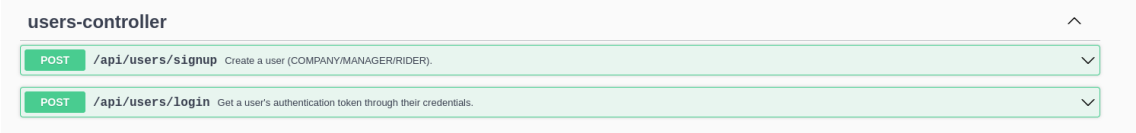

Figure 9: Architecture Diagram

The SL manages all the store business needs like fetching products or checking out the cart.

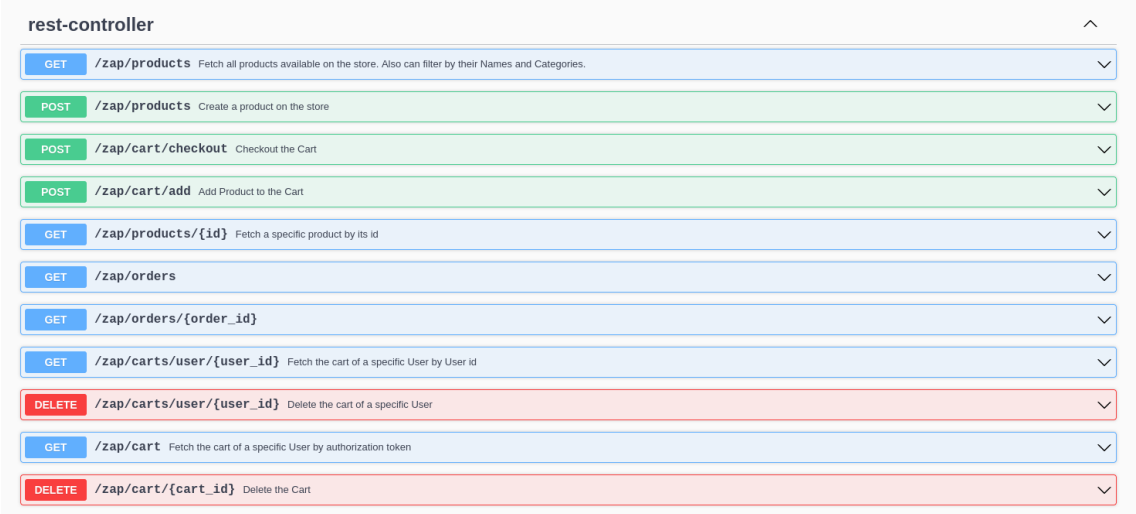

Figure 10: Architecture Diagram

5.2 Deliverize API

The API is also divided in two parts, the User Authentication and Management (UAM) and the Deliverize Orders (DO).

The UAM manages all the authentication and authorization on the website and has the following endpoints:

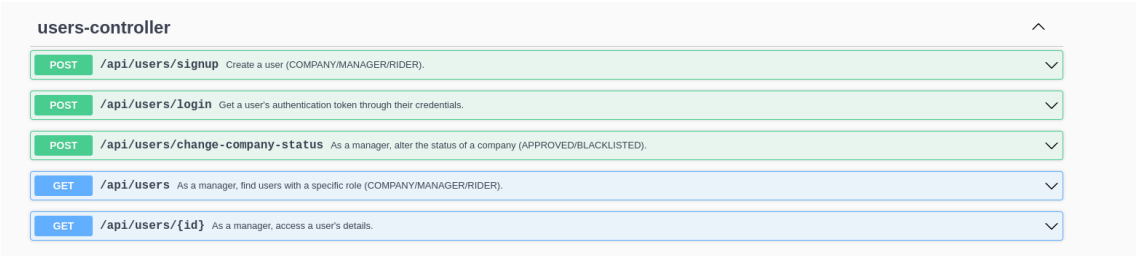

Figure 11: Architecture Diagram

The DO manages all the internal delivery and management requests.

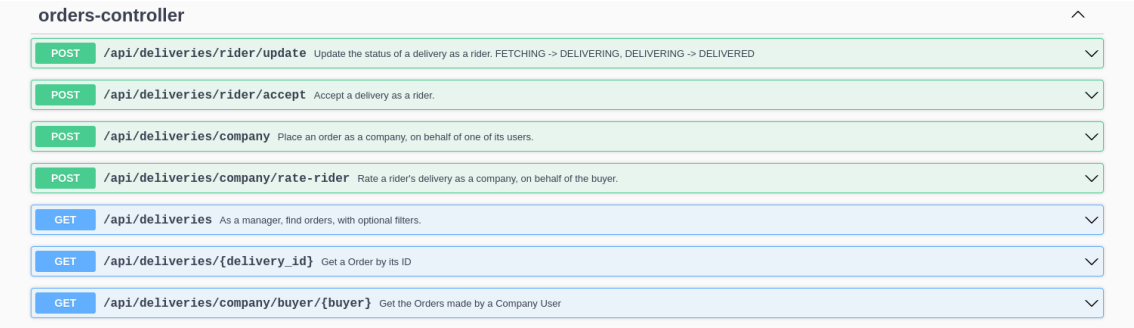

Figure 12: Architecture Diagram