TQS: Product specification report

Bárbara Nóbrega Galiza [105937], Diana Miranda [107457], Diogo Falcão [108712], Rúben Garrido [107927]

v2024-04-19

1 Introduction

1.1 Overview of the project

This project is undertaken as part of the TQS course, with the goal of developing a MVP product while applying software enterprise architecture patterns, specifying and enforcing a Software Quality Assurance (SQA) strategy and applying Continuous Testing, Continuous Integration and Continuous Delivery practices. The chosen theme was a bus service, like the ones you found in common transportation terminals. This is a digital service which includes a customer portal, tools for staff to edit bus and trips information and a digital signage system. Our project aligns with these goals by focusing on the development of a user-centered online system for a public bus service, with the advantage of unifying these digital services into one.

Introducing CityConnect: CityConnect is a web-based application designed to streamline operations and enhance the user experience. It encompasses three key components:

- **Customer Portal:** This user-friendly interface empowers passengers to access real-time bus information, purchase tickets and plan their journeys.
- **Staff Management System:** This system equips bus operations staff with the tools they need to manage terminal operations.
- **Digital Signage:** Current buses status screen displayed in CityConnect terminals.

By creating a unified digital platform for both passengers and staff, CityConnect aims to improve efficiency, transparency, and overall satisfaction with the bus service.

1.2 Limitations

Due to the fact that the expected outcome of this project is a MVP (minimal viable product), we expect some features to remain unimplemented, like the self check-in system. However, the core functionality will be done, with as many features as possible.

2 Product concept and requirements

2.1 Vision statement

CityConnect is a platform designed to simplify the booking, control, and management of bus travel. It consists of three main components: a back-office platform for staff, an app for customers, and an informative display at bus stops.

The system aims to address the challenges faced by bus operators and customers using these services. Bus operators encounter difficulties in organizing and efficiently managing trips, controlling seat availability, and effectively communicating with customers about bus departures and arrivals. On the other hand, customers often struggle to find updated information on departure times, make reservations conveniently, and stay informed in real-time about changes in travel schedules.

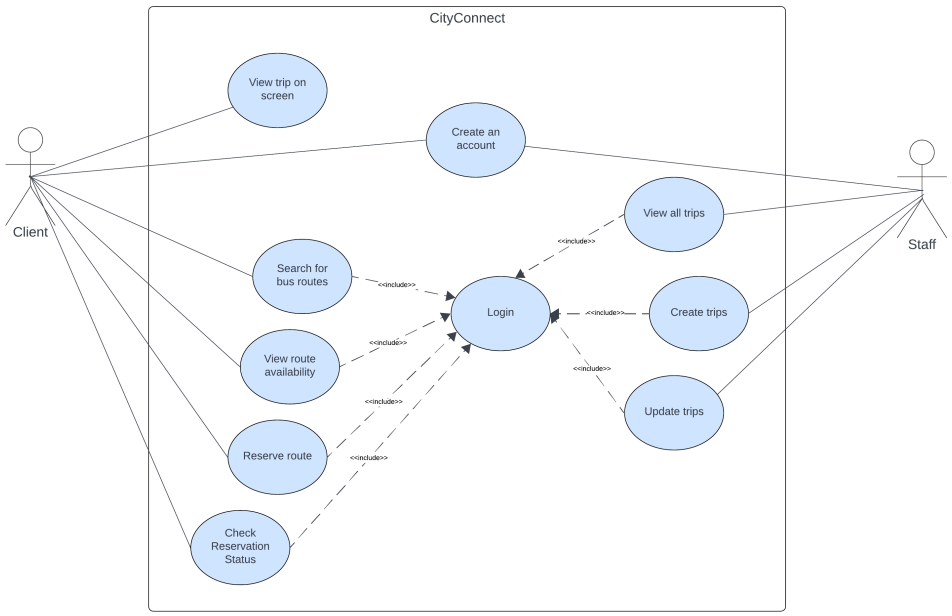

Figure 1: UML Use Case Diagram

2.2 Personas and scenarios

2.2.1 Personas

- **Luísa Almeida (Staff)**

  Gender: Female.

  Age: 38 years old.

  Occupation: Bus operations staff.

  Background: She works for CityConnect, since 2016.

  Needs: She needs an online system for defining trips, configuring its availability and editing its details

  Main motivations: She needs a highly available, efficient and easy to use online system for reduced manual tasks. She wants to work in a efficient manner.

- **Daniel Perreira (Client)**

  Gender: Male.

  Age: 52 years old.

  Occupation: Construction worker.

  Background: He travels every weekend back to his home city by bus.

  Needs: An easy-to-use app for purchasing tickets, managing reservations, checking available trips, and their schedules quickly and without waiting.

  Main motivations: He values his time and seeks a hassle-free experience when buying bus tickets. He wants to book his tickets quickly and easily, avoiding long lines or complicated processes. His priority is a convenient and efficient solution to streamline his travel experience.

2.2.2 Scenarios

Scenario 1: Book a trip

Daniel Perreira is going on a bus trip back home next weekend. He needs to book a bus trip from Aveiro to his hometown, in a suitable date and hour. Then, he searches for available trips. He looks closely at the various trips to find one that meets his needs. He considers price, date and hour through his options. Finally, he selects a trip and books a reservation.

Scenario 2: Check trip state at call screen

Daniel Perreira is going on a bus trip back home today. He gets to the bus station, and searches for the call screens. After finding a screen, he reads the list, finds his trip and checks if and where his bus is on-boarding. After reading it is now boarding, he goes on to the boarding place and gets inisde the bus.

Scenario 3: Create trip

Luísa, tasked with adding a new route from a partner company, logs into CityConnect's system. She navigates to "Connections" and adds a new trip, defining route details, schedule, bus type, and seats. After reviewing, she creates the trip.

Scenario 4: Update trip details

There was an issue with one of the buses, which is now scheduled to depart 10 minutes late. To inform customers about this change, Luísa needs to input this information into the system. This will ensure that the update is reflected both on the display screen and in the application, allowing passengers to receive this information about the delay. So she looks for the corresponding trip in the trips list, chooses it, and update its details.

2.3 Project epics and priorities

2.3.1 Project epics

Epic 1: Authentication for user and staff

The persona must be able to create an account and to login.

Goal functionality: Authentication logic for both user and staff, protect endpoints to be used only by logged-in users.

Priority: High

Epic 2: Staff trip management (Staff portal)

The staff must be able to create and update trips/routes.

Goal functionality: view list of trips, view trip, edit trip, add trip.

Priority: High

Epic 3: Book and check reservations (Client portal)

The user must be able to book trips and check reservation status.

Goal functionality: remaining functionality related to booking trips and checking reservation status. A part of it was already done from the midterm project.

Priority: Medium

Epic 4: Digital signage

The user must be able to view the call screen information, and the staff updates must be reflected on it.

Goal functionality: Call screen receives and displays updated information, in a user-friendly manner.

Priority: Low

3 Domain model

This system contains several information regarding buses, cities, trips and users.

The **Bus** class contains the number of seats, and the **City** one keeps track of a city's name.

Each **Trip** has an associated bus and two cities (i.e., the departure and the arrival ones). Besides that, it stores the departure and arrival times, as well as the price (in EUR) and the number of free seats.

The **User** class stores a user's name, email, password, address and phone number.

A trip **Reservation** holds information regarding a ticket, with associated Trip and User, as well as the price and the number of seats booked.

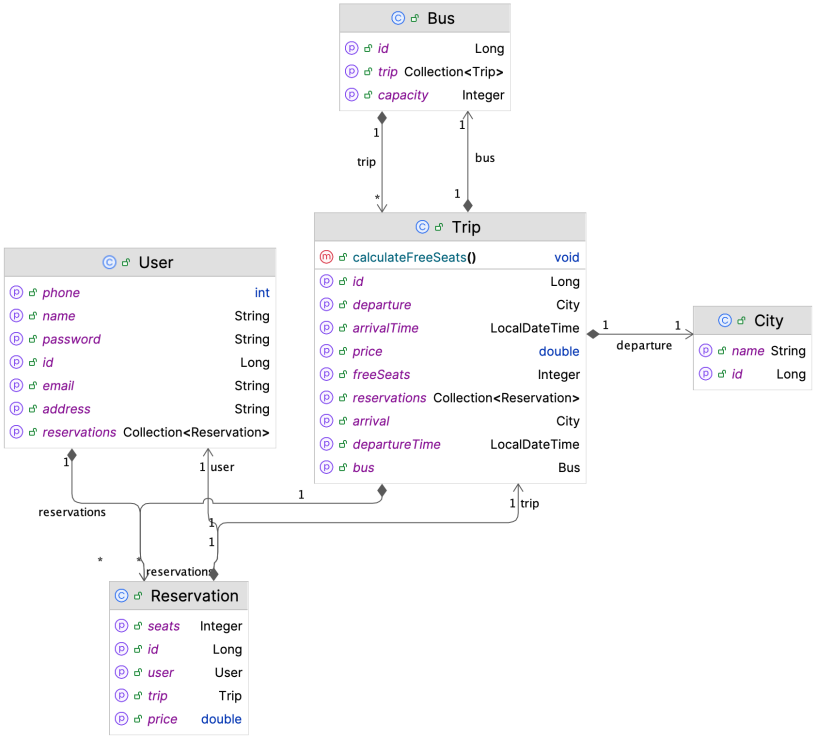

Figure 2: UML classes diagram

4 Architecture notebook

4.1 Key requirements and constrains

This system has multi user-interfacing platforms, from a mobile app to a big call screen. Also, it's created from scratch, which means that there are no legacy systems to deal with, so we can design it using modern technologies, and no data is adaptable.

Regarding deployment and different server conditions, the project should work in the same way, no matter the server it is hosted on. Also, it should have remote access, so that clients can buy trips from anywhere in the world.

The backend should require authentication for all data-sensitive operations, so that authorization is applied and thus each person gets only access to the data they are meant to get or modify.

An external currency-conversion API should be used for making this project available in multiple currencies.

4.2 Architecture view

This project is defined by a micro-service architecture, on a Docker environment. It creates a unified experience across multiple environments, which means that the whole project is decoupled from the server(s) it's hosted on. Also, it makes it easy to enable remote access, with a simple port forwarding needed, since the port exposing is part of the Docker solution and not the host.

The backend consists of a Spring Boot app that is connected to a PostgreSQL database through the JDBC (Java Database Connectivity) protocol. The database is connected to a separate Docker network, that only the backend and the database can access, which means that no other container can communicate with it. Regarding the external currency-conversion API, its results are cached in memory by the backend, with periodic fetching, for faster loading times and reduced API costs.

There are three frontend services: the staff, the client and the digital signage. All of them are implemented in React, using the Vite building tool. For the client portal, a mobile app is required, so Capacitor is used to create a mobile web app based on the corresponding website.

All these containers have no exported ports due to security and CORS blocking. To circumvent this, a Traefik-powered reverse proxy exposes ports 80 and 8080 (only in development builds), and maps requests to the respective containers. It allows further access control, as well as secure protection to the upstream servers.

Backend and frontend communicate through a REST API and WebSockets, both using HTTP underneath. WebSockets are used for real-time data fetching, which is useful for some environments, such as the data signage user interface.

CI/CD workflows are used for automatic testing, integration and deployment through GitHub Actions, which leads to better code quality and lower downtime. Check section 4.3 for more regarding deployment.

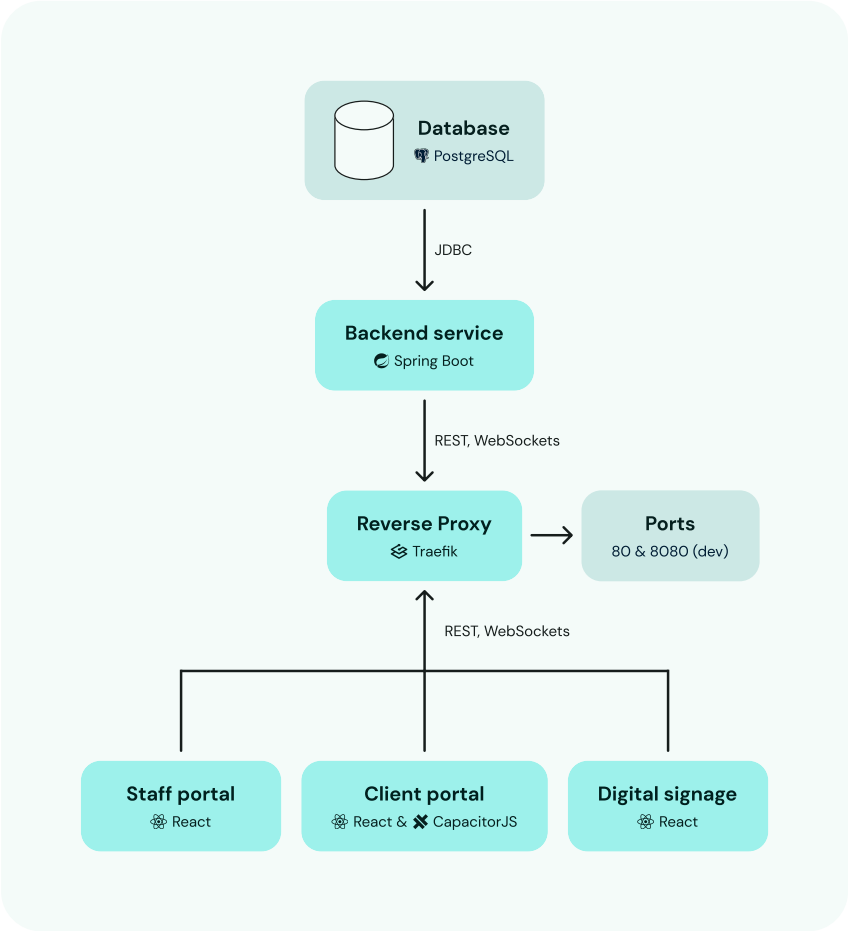

Figure 3: Architecture diagram

4.3 Deployment architecture

The deployment architecture is pretty much the same as the development one, since both use Docker containers in a similar environment.

However, some environment variables are changed, such as secrets, and production builds are used instead of development ones. This is defined in production-specific Dockerfile and Docker Compose schemas, with a deployment as simple as running `docker compose -f compose.prod.yaml up -d` in the terminal.

5 API for developers

The API supports five entities: Bus, City, Trip, Reservation, and User. All of them have endpoints for the basic CRUD operations.

Regarding relationships, a Trip contains one Bus and two Cities, while a Reservation has one Trip and one User. Therefore, there are two additional endpoints:

- Get reservations associated to an user
- Get reservations associated to a trip

There is also an endpoint that shows statistics about the currency API service and its cache.

Documentation regarding the API can be reached through a Swagger instance that is embedded in the Spring Boot app and implements the OpenAPI protocol.

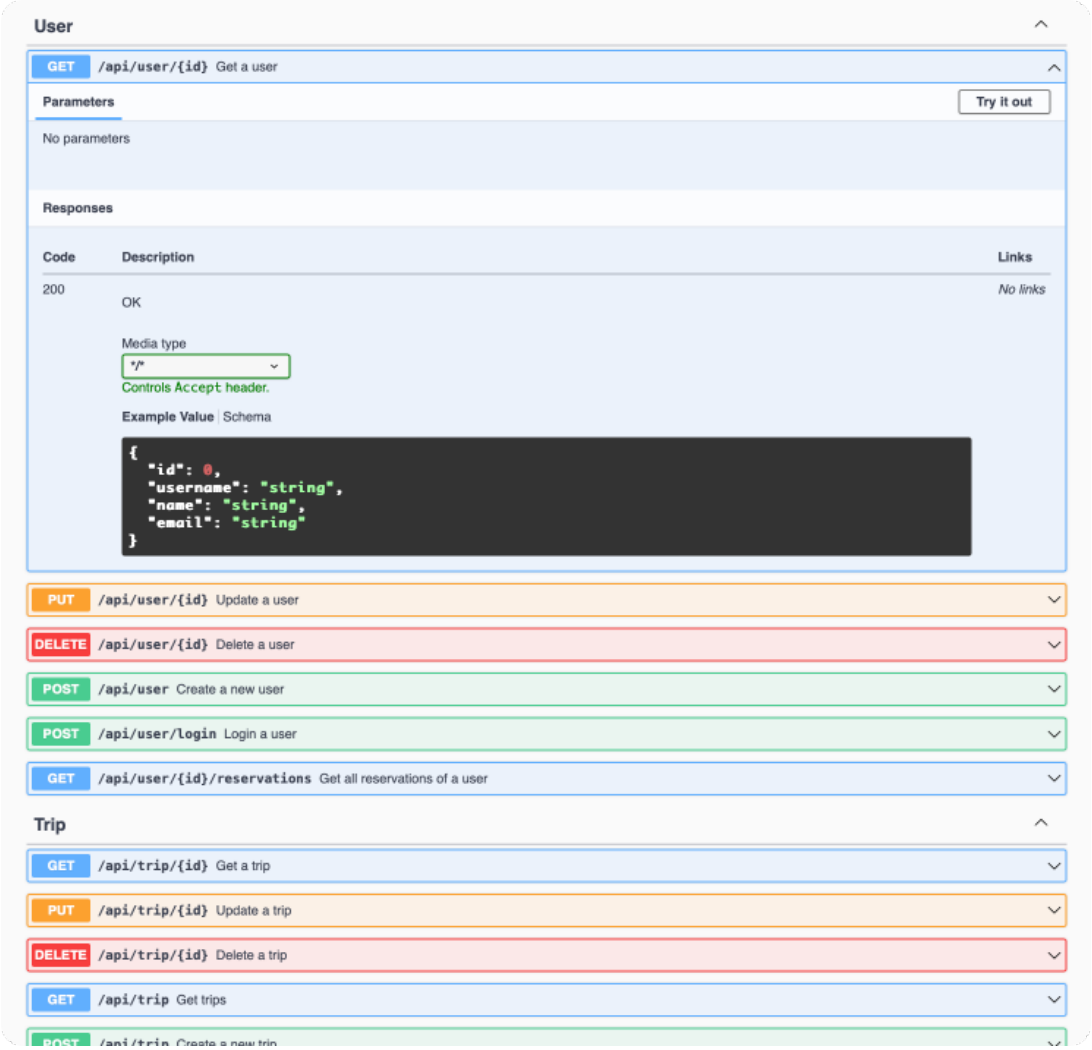

Figure 4: Swagger UI

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¿