TQS: Product specification report

Diogo Marto [108298], Tiago Pereira [108546], Miguel Pinto [107449]

V06-20224

1 Introduction

1.1 Overview of the project

This document serves as the comprehensive product specification report for the final project of the TQS (Teste e Qualidade de Software) course. This project's main objective is to demonstrate the application of software quality practices through developing a Minimum Viable Product (MVP). The project encapsulates various aspects of software engineering, including requirements gathering, system design, implementation, testing, and deployment, strongly emphasizing quality assurance.

The project, named ChuChu, is a train reservation and management system designed to streamline the process of booking train tickets, managing train schedules, and providing real-time information through digital signage. This system aims to address the common challenges faced by train travelers, such as inefficient booking processes, lack of real-time updates, and poor user experience.

ChuChu is built with a robust backend using Spring Boot, which implements three distinct APIs with varying access levels. These APIs support different functionalities and ensure secure and efficient communication between the server and client. The front end, developed using Ionic, provides a user-friendly interface for customers and administrators to interact with the system.

1.2 Limitations

Despite our efforts to deliver a comprehensive solution, there are some known limitations in the current version of ChuChu that we aim to address in subsequent iterations:

- Allow the user to choose seats & class. Already backend supported and tested, lacked time to develop front-end UI.
- Ticket Usage.
- Cancel Ticket.

2 Product concept and requirements

2.1 Vision statement

ChuChu is designed to optimize the train travel experience by providing a seamless and efficient platform for booking reservations, managing train schedules, and displaying real-time travel information. Our system addresses the high-level business problem of enhancing the railway sector's operational efficiency and user satisfaction. By integrating advanced digital signage and a user-friendly interface, ChuChu ensures that passengers have access to timely information and an easy booking process. The ultimate vision is to create a comprehensive ecosystem that supports travelers, train operators, and administrators in making train journeys more enjoyable, reliable, and hassle-free.

2.2 Personas and scenarios

Personas:

a Thompson):

: Marketing Manager

- Goals: Commutes daily for work, needs a reliable and quick way to book tickets

and check schedules, and a way to check previous trips.

- Pain Points: Dislikes long queues and outdated information, values efficiency and accuracy.

Occasional User (John Doe):

- Age: 45
- Occupation: Freelance Photographer
- Goals: Travels occasionally for assignments, prefers a straightforward and

hassle-free booking process.

- Pain Points: Finds it difficult to navigate complex booking systems, needs clear and simple options.

Admin (Sarah Lee):

- Age: 40
- Occupation: Railway Operations Manager
- Goals: Manages train schedules and ensures smooth operations, requires

efficient tools to monitor and update information.

- Pain Points: Needs reliable systems to avoid disruptions and ensure

passenger satisfaction.

Scenarios:

Emma Thompson (Regular User):
Scenario: Emma logs into ChuChu, searches for available trains on her regular route, selects a preferred seat, and completes her reservation within minutes. She receives confirmation and views her booking details.

John Doe (Occasional User):
Scenario: John opens ChuChu to book a train ticket for a photography assignment. He quickly finds the train he needs, selects a one-time ticket option, and completes the purchase without any complications. On the day of travel, he uses the digital signage module to locate his train and check its schedule.

Sarah Lee (Admin):
Scenario: Sarah uses the admin dashboard to create new train schedules and update existing ones. When an unexpected delay occurs, she updates the system, and the changes are instantly reflected on the digital signage and user interfaces.

2.3 Project epics and priorities

Epics:

User Authentication and Authorization (Auth):

- Priority: Medium
- Description: Implement secure login and registration processes to ensure that only authorized users can access certain parts of the system. This includes password encryption and user role management.

Reservation Management (Travel):

- Priority: High
- Description: Develop the core functionalities for searching train schedules, booking tickets, and managing reservations. This includes real-time seat availability, payment integration, and booking confirmation.

Digital Signage System (Digital Signage):

- Priority: Medium
- Description: Implement a robust digital signage module to display real-time information such as train schedules, delays, and other announcements. This module should be easy to update and integrate seamlessly with the backend.

Admin Tools and Schedule Management (Manage Connections - Admin):

- Priority: High
- Description: Develop tools for administrators to manage train schedules, monitor system performance, and ensure data accuracy. This includes interfaces for creating, updating, and deleting train schedules.

Incremental Implementation Plan:

Spring 0): *(Not tracked on Jira)*

- Define the product concept.
- Team resources setup
- Start backlog

Sprint 1):

- Basic user authentication and registration.
- Basic understanding of domain and requirements.
- Setting up CI pipeline.
- Agreeing of pratices to ensure quality.
- Defining architeture

Sprint 2):

- Implementation of managing and creating trips,
- Search for trips.
- Non-functional tests and systems observability.

Sprint 3):

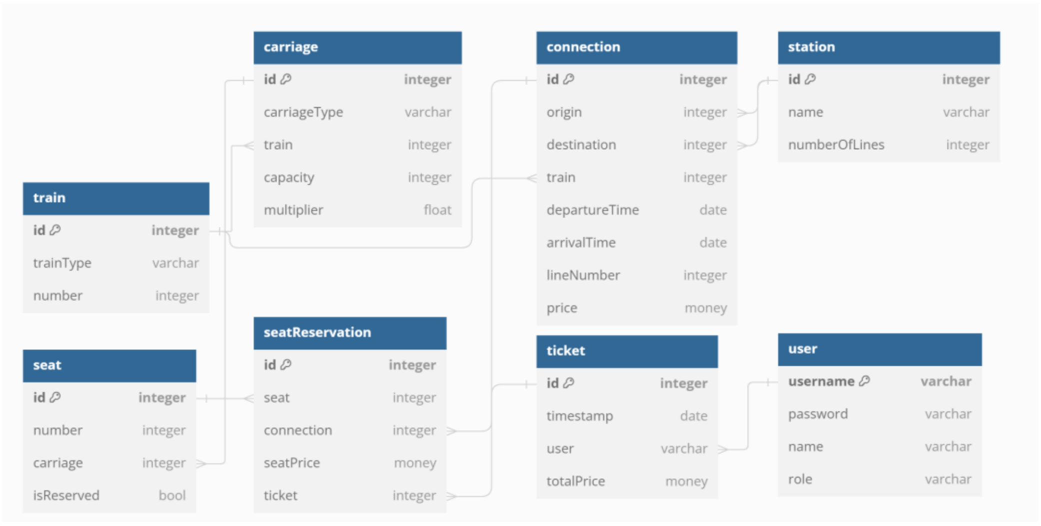
- Implementation of the digital signage module.
- Finish buy ticket flow.
- API documentation
- Initial deployment of VM.

Sprint 4):

- Deployment via CD
- Edit ticket

Goal of a having a MVP by 3rd of june.

3 Domain model



Train:

- id (integer): Unique identifier for the train.
- trainType (varchar): Type of the train.
- number (integer): Train number.

Carriage:

- id (integer): Unique identifier for the carriage.
- carriageType (varchar): Type of carriage.
- train (integer): Foreign key linking to the train entity.
- capacity (integer): Number of seats in the carriage.
- multiplier (float): Pricing multiplier based on the carriage type.

Seat:

- id (integer): Unique identifier for the seat.

- number (integer): Seat number within the carriage.
- carriage (integer): Foreign key linking to the carriage entity.
- isReserved (bool): Indicates if the seat is reserved.

Connection:

- id (integer): Unique identifier for the connection.
- origin (integer): Foreign key linking to the station entity for the origin station.
- destination (integer): Foreign key linking to the station entity for the destination station.
- train (integer): Foreign key linking to the train entity.
- departureTime (date): Departure time of the train.
- arrivalTime (date): Arrival time of the train.
- lineNumber (integer): Line number of the connection.
- price (money): Price for the connection.

Station:

- id (integer): Unique identifier for the station.
- name (varchar): Name of the station.
- numberOfLines (integer): Number of lines at the station.

SeatReservation:

- id (integer): Unique identifier for the seat reservation.
- seat (integer): Foreign key linking to the seat entity.
- connection (integer): Foreign key linking to the connection entity.
- seatPrice (money): Price of the reserved seat.
- ticket (integer): Foreign key linking to the ticket entity.

Ticket:

- id (integer): Unique identifier for the ticket.
- timestamp (date): Timestamp when the ticket was issued.
- user (varchar): Foreign key linking to the user entity (username).
- totalPrice (money): Total price of the ticket.

User:

- username (varchar): Unique username for the user.
- password (varchar): Password for the user account.
- name (varchar): Full name of the user.
- role (varchar): Role of the user (e.g., regular, admin).

If the role of User is “ADMIN” he has increased privileges and can for example create new connections.

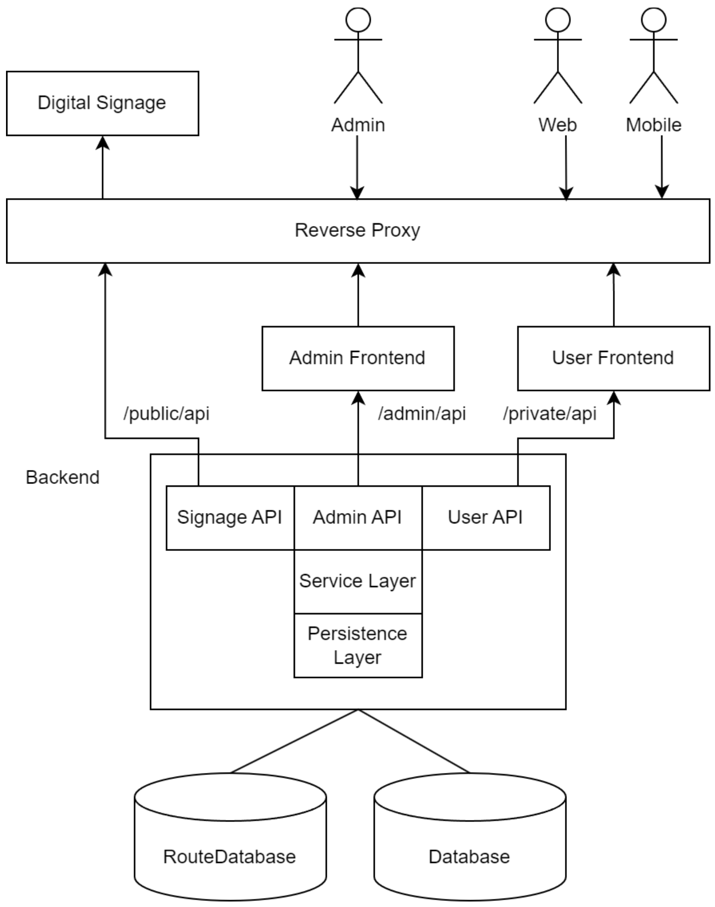
4 Architecture notebook

4.1 Key requirements and constrains

The ChuChu train reservation and management system is designed to address several key requirements and constraints to ensure a seamless and efficient user experience:

- The system must provide real-time responses for booking queries and updates to train schedules.
- The digital signage module must display real-time information without delays.
- The system must implement secure login processes and encrypt sensitive data.
- Admin functionalities must be secured to prevent unauthorized access.
- The system will be accessed through multiple user-facing platforms, including web, mobile devices, and digital signage.
- The system should be designed for easy maintenance and updates, with modular components.
- The system must ensure data integrity and consistency across all modules.
- All external interfaces and APIs must adhere to standard protocols.
- The system components must be containerized using Docker for consistent deployment.
- The system must include comprehensive monitoring and logging capabilities.
- Non-functional testing and observability tools should be integrated into the development pipeline.
- The user interfaces should be intuitive, easy to navigate, and accessible to users with disabilities.
- The system should be able to calculate routes between station A and B even if A and B arent directly connected.

4.2 Architecture view



Components:

- Reverse Proxy: Acts as the entry point for all incoming traffic. It routes requests to the appropriate backend services or frontend applications based on the request path.
- Frontend Applications:
  - Admin Frontend: A web interface designed for administrative tasks such as content management, user management, and system configuration.
  - User Frontend: Provides the interface for users to interact with the digital signage content via web and mobile applications.
- Backend:

APIs: The backend exposes a set of APIs to handle requests from the frontends and the digital signage displays. These APIs include:

- Signage API: Handles requests related to displaying content on the digital signage screens this API is public.
- Admin API: Handles requests for administrative tasks. This API is private and only selected users can access it.
- User API: Handles requests from user applications coming from the frontend. This API is internal and only used to serve frontend.

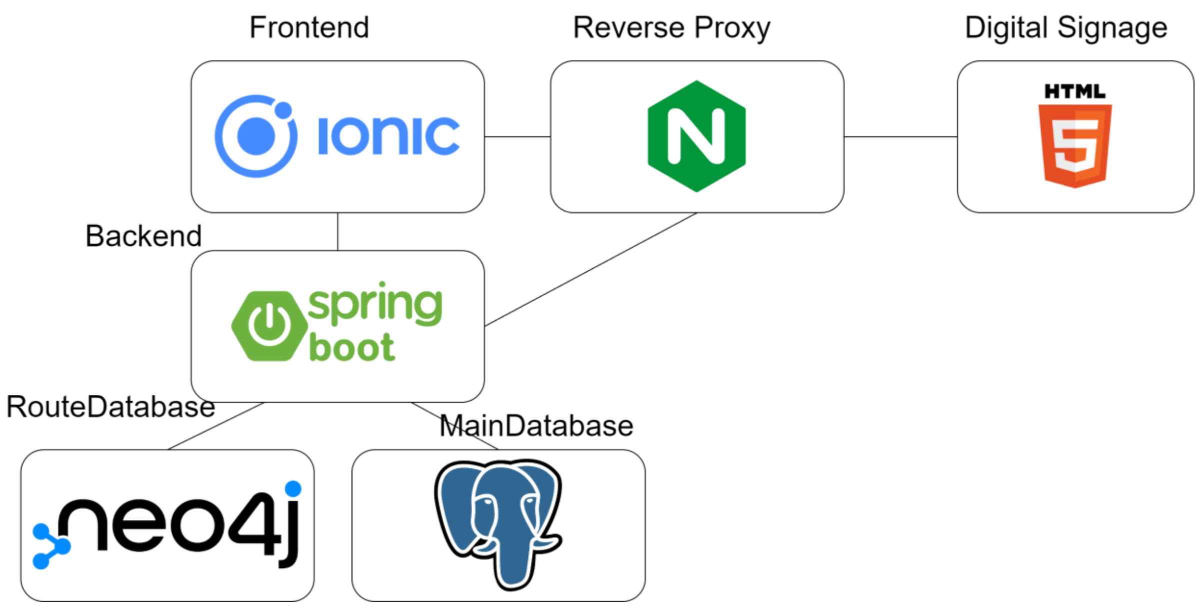
- Service Layer: Implements the business logic of the system, interacting with the persistence layer and handling data processing.
- Persistence Layer: Manages the storage of data in databases.

- RouteDatabase: Stores information related to the routing of trains.
- Database: A general-purpose database that stores other data.

Overall Flow:

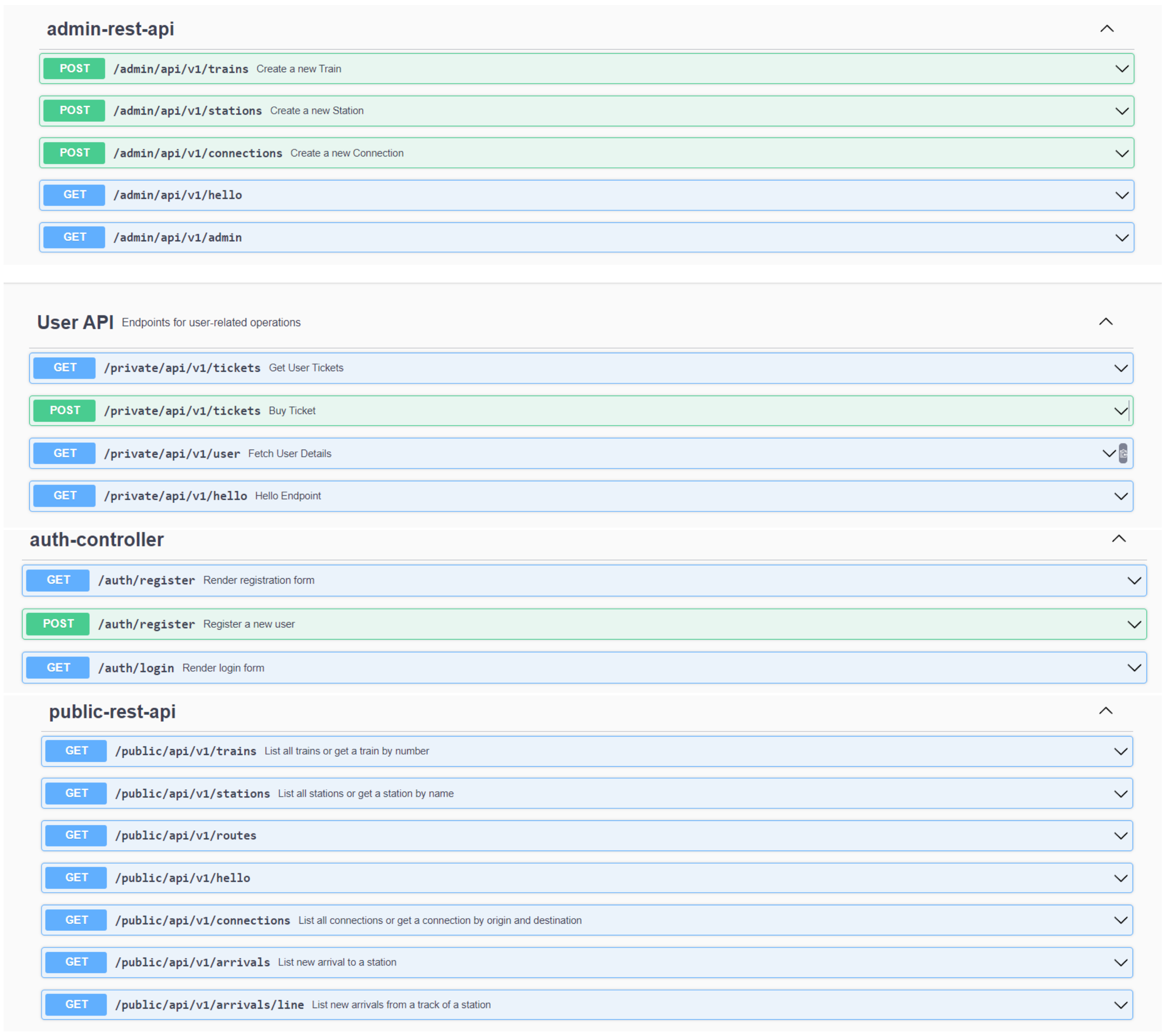
Users access the system through the web or mobile applications (User Frontend). The User Frontend communicates with the User API to fetch content and perform user-related actions. Administrative users access the Admin Frontend for administrative tasks. The Admin Frontend communicates with the Admin API to perform tasks such as content management and user management. The Signage API provides content to the digital signage displays based on predefined stations and lines. The Service Layer handles the business logic, processing data and interacting with the persistence layer. Data is stored in the RouteDatabase and the general-purpose Database via the persistence layer. Additionally, the RouteDatabase calculates routes between two specified points.

4.3 Deployment architecture



The backend is made in SpringBoot, the routeDatabse is in neo4j, the mainDatabase is in Postgres, both the user frontend and admin frontend are made in ionic that can compile to a website or a mobile app, the reverse proxy is nginx, and the digital signage is made using html and javascript to periodically query the public API. All these components are containerized using Docker and can all be deployed via a Docker Compose except for the Digital Signage module since its deployment is on the stations and not on a server.

5 API for developers



admin-rest-api

POST /admin/api/v1/trains Create a new Train

POST /admin/api/v1/stations Create a new Station

POST /admin/api/v1/connections Create a new Connection

GET /admin/api/v1/hello

GET /admin/api/v1/admin

User API Endpoints for user-related operations

GET /private/api/v1/tickets Get User Tickets

POST /private/api/v1/tickets Buy Ticket

GET /private/api/v1/user Fetch User Details

GET /private/api/v1/hello Hello Endpoint

auth-controller

GET /auth/register Render registration form

POST /auth/register Register a new user

GET /auth/login Render login form

public-rest-api

GET /public/api/v1/trains List all trains or get a train by number

GET /public/api/v1/stations List all stations or get a station by name

GET /public/api/v1/routes

GET /public/api/v1/hello

GET /public/api/v1/connections List all connections or get a connection by origin and destination

GET /public/api/v1/arrivals List new arrival to a station

GET /public/api/v1/arrivals/line List new arrivals from a track of a station

Our API is available at http://deti-tqs-17.ua.pt:9090/swagger-ui/index.html , if you are running the project yourself it is going to be at *localhost:9090/swagger-ui/index.html.* The API uses annotations on springboot which automatically generates the swagger file. The API is divided into 4 parts: auth, admin-rest-api, rest-api, and public. Auth provides authentication methods, admin-rest-api feeds the frontend for the admin interface it's meant to be an internal API, and rest-feeds the regular user interface its mean meant to be an internal API. The public API is meant to be an API that we provide to other developers to integrate without system, namely we can create using a digital signage module.

6 References and resources

- https://ionicframework.com/docs
- https://www.atlassian.com/devops/testing-tutorials/jira-xray-integration-manage-test-cases
- https://docs.github.com/en/actions