

HW1: Mid-term assignment report

Roberto Rolão de Castro [107133], v2024-04-09

1	Introduction	1
1.1	Overview of the work.....	1
1.2	Current limitations.....	1
2	Product specification.....	2
2.1	Functional scope and supported interactions.....	2
2.2	System architecture.....	2
2.3	API for developers	3
3	Quality assurance	3
3.1	Overall strategy for testing	3
3.2	Unit and integration testing.....	3
3.3	Functional testing.....	4
3.4	Code quality analysis.....	5
4	References & resources	5

1 Introduction

1.1 Overview of the work

This report presents the midterm individual project required for TQS, covering both the software product features and the adopted quality assurance strategy.

It involves developing a full-stack application named BusTickets - TQS, designed to streamline the process of purchasing bus tickets online. It leverages React for a dynamic front-end user experience and Maven for managing the RESTful API backend. This app allows users to search, buy tickets for various bus routes and then check the reservation by the reservation Id.

1.2 Current limitations

The known limitations of the project are:

- Continuous Integration/Continuous Deployment (CI/CD) pipelines are not implemented, which may result in manual testing and deployment processes, slowing down development cycles.
- The functionality for canceling reservations is missing, not allowing users to manage their bookings correctly.

These limitations impact the application's functionality, automation capabilities, and user experience. Addressing them will be essential for enhancing overall quality and usability.

2 Product specification

2.1 Functional scope and supported interactions

The website is created to offer an effective bus ticket booking process. The main user of this application is the Passenger, who utilizes it to meet their travel requirements.

Actor: Passenger

Primary Goal:

- Find and purchase bus tickets for their desired routes and see their reservations.

Interactions:

- Finding and Buying Tickets:** The passenger looks for bus tickets by specifying the departure city, the destination city, and the departure date. The web interface will list all the available bus routes that match the criteria. The passenger can then choose the best option and buy the ticket.
- See Their Reservation:** Once the ticket is bought, the passenger gets a reservation ID. They can check their reservation details at any time by entering this ID, allowing them to see the travel information.

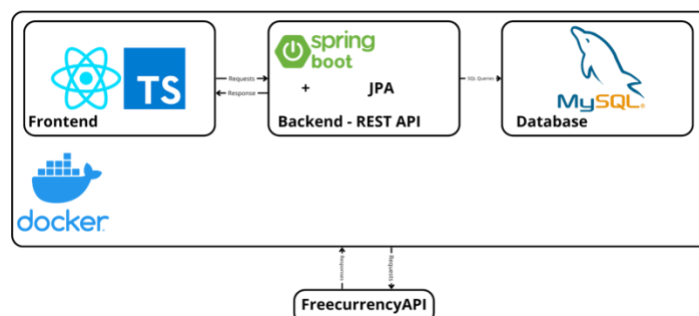
Main Use Case:

- A passenger is looking to book a bus ticket from Aveiro to Porto, on the April 10th 2024 and want to use BusTickets website. They input their travel data on the web, see the different bus options, and choose the best for them. Once they successfully book the ticket, they receive a reservation ID. This unique ID allows them to view their booking specifics, access their ticket details.

2.2 System architecture

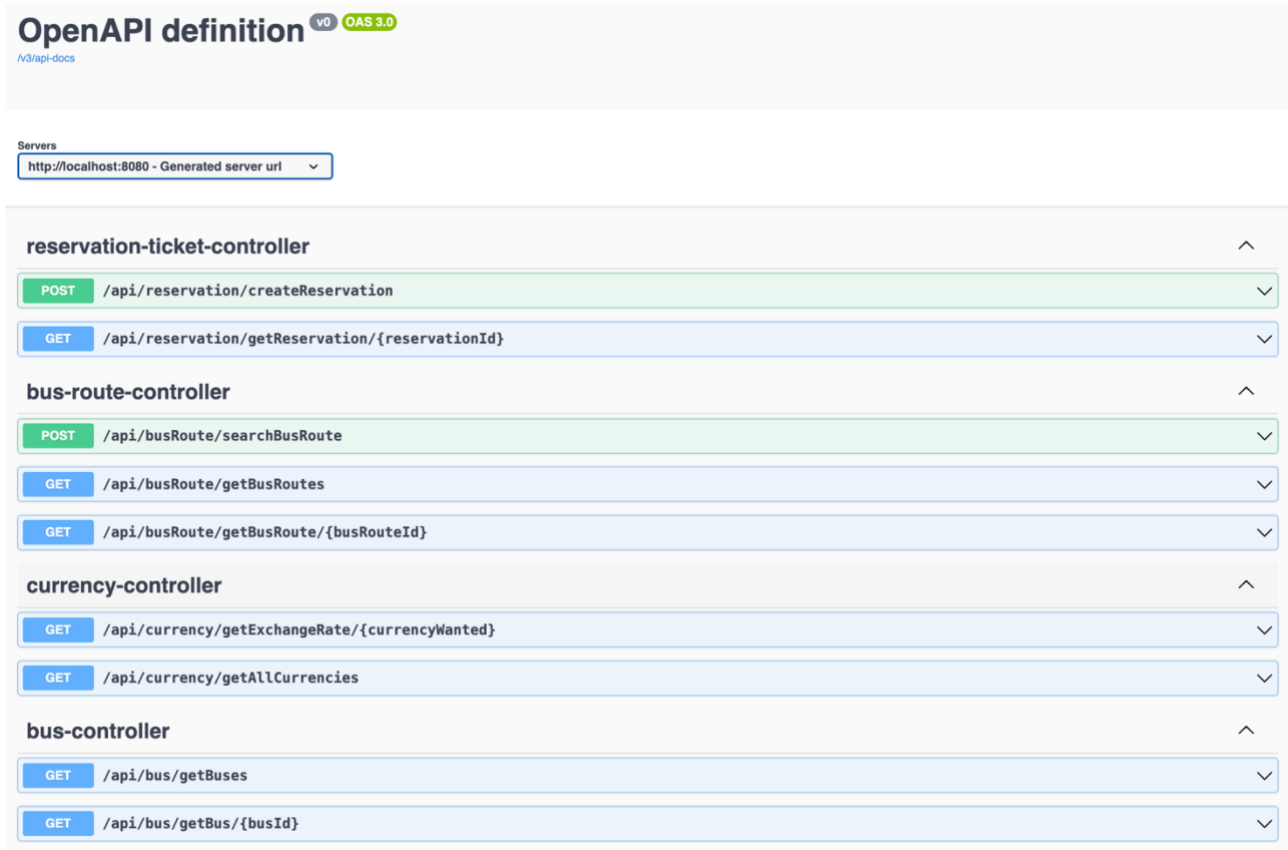
The architecture is divided into three main components: the front end, the back end, and the database, all encapsulated within Docker containers for ease of deployment and isolation.

For the frontend the technology used is React, using *TypeScript*. For the backend it uses Maven with Spring Boot and Java Persistence API (JPA), and it calls an external API for the exchanges rates and to get all the currencies. And for the database I use MySQL.



2.3 API for developer

The documentation for the REST API was created using Swagger Open API 3, integrated with Maven, and is available through the /swagger-ui.html endpoint of the API, accessible at localhost:8080/swagger-ui.html.



OpenAPI definition v0 **OAS 3.0**
/v3/api-docs

Servers
http://localhost:8080 - Generated server url

reservation-ticket-controller

- POST** /api/reservation/createReservation
- GET** /api/reservation/getReservation/{reservationId}

bus-route-controller

- POST** /api/busRoute/searchBusRoute
- GET** /api/busRoute/getBusRoutes
- GET** /api/busRoute/getBusRoute/{busRouteId}

currency-controller

- GET** /api/currency/getExchangeRate/{currencyWanted}
- GET** /api/currency/getAllCurrencies

bus-controller

- GET** /api/bus/getBuses
- GET** /api/bus/getBus/{busId}

3 Quality assurance

3.1 Overall strategy for testing

The testing strategy for BusTickets - TQS followed a post-development approach, instead of using TDD, focusing on testing after the application code was completed. This strategy included unit tests for entities and DTOs, using mocks for the service tests (which allows to isolate the verification of business logic), integration tests were applied to controllers to confirm correct handling of and responses and for the web interface testing it was BDD with cucumber.











3.2 Unit and integration testing

The unit tests were used to test the different entities and the DTOs validate their functionality. In my project I used Mockito for isolating the service layer from its dependencies, ensuring business logic correctness.

For the integration tests, they were concentrated on controllers, allowing to test the integration between the different layers. I also used Jacoco to help me keep a track of the tests coverage.

HW1-BusTicketSelling

HW1-BusTicketSelling

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
ua.deti.tqs.hw1busticketselling.entity		100%		100%	0	87	0	58	0	86	0	4
ua.deti.tqs.hw1busticketselling.service		100%		100%	0	26	0	93	0	23	0	7
ua.deti.tqs.hw1busticketselling.config		100%		100%	0	13	0	60	0	9	0	3
ua.deti.tqs.hw1busticketselling.controller		100%		100%	0	20	0	45	0	17	0	4
ua.deti.tqs.hw1busticketselling.dto		100%		n/a	0	38	0	23	0	38	0	2
ua.deti.tqs.hw1busticketselling		100%		n/a	0	2	0	3	0	2	0	1
Total	0 of 1 480	100%	0 of 22	100%	0	186	0	282	0	175	0	21

3.3 Functional testing

The functional tests were made with a focus on user-facing scenarios to ensure the application's behavior aligned with user expectations. This was achieved through Behavior-Driven Development (BDD) using Cucumber, which allowed for the creation of test cases in a human-readable format.

```
Feature: BusTicket Book Bus

Background:
  Given I am on the BusTicket homepage on "http://localhost:5173"

Scenario: Book a bus
  When I write "Aveiro" as the departure city and "Porto" as the arrival city
  And I select the date "11-04-2024" for the departure
  And I click on the Search Buses button
  And I select the Currency "¥ - JPY - Japanese Yen"
  And I choose the bus with Bus Number "R045"
  And I fill in the form with the following information:
    | Name | Roberto |
    | Surname | Castro |
    | Email | robertorcastr@ua.pt |
    | Address | Caminho da Alegria 5 |
    | Postal Code | 4950-750 |
    | City | Monção |
    | Country | Portugal |
    | Phone Number | 916162223 |
    | Credit Card Number | 1111222233334444 |
    | Expiration Date | 12/2024 |
    | CVV | 123 |
  And I click on the Confirm Button
  And I capture the confirmation code
  Then I should get the "Booking Confirmed!!"

Scenario: Check Reservation
  When I click on the Check Reservation button
  And I write the confirmation code
  And I click on the Search Button
  Then I should get the "Booking Details" and the Reservation Status "Confirmed"
```

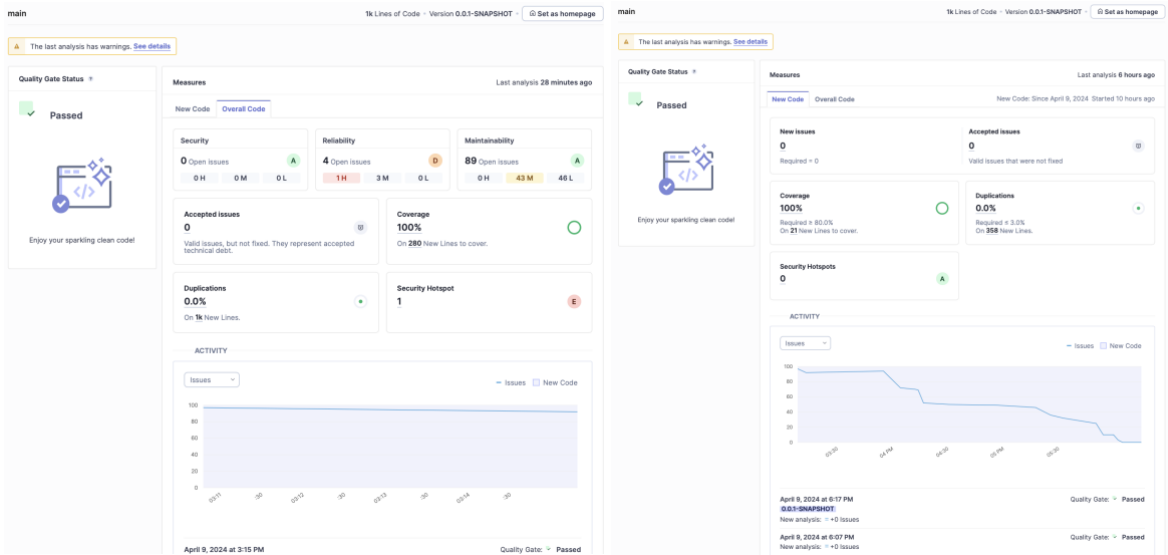
```
@And("I select the date (string) for the departure")
public void i_select_the_departure_date(String depDate) {
    logger.info(format("Departure Date is: {} ", depDate));
    WebElement element = driver.findElement(By.id("date"));
    element.click();
    element.sendKeys(depDate);
}

@And("I click on the Search Buses button")
public void i_click_on_the_SearchBuses() {
    WebElement element = driver.findElement(By.cssSelector(cssSelector:"btn-primary"));
    logger.info(msg:"Click on the Search Buses button");
    element.click();
}

@And("I select the Currency (string)")
public void i_select_the_currency(String currency) {
    WebElement element = driver.findElement(By.id(id:"dropdown-variants-Currency"));
    element.click();
    logger.info(format("Choosing the currency: {} ", currency));
    WebElement currencyElement = driver.findElement(By.xpath(xpathExpression:"/html/body/div/nav/div/div/div/div/div/a[3]"));
    currencyElement.click();
}
```

3.4 Code quality analysis

For code quality analysis in BusTickets project, SonarQube was employed to perform static code analysis. This tool provided valuable insights into code quality, identifying potential bugs, vulnerabilities, and code smells. In the first scan I got some code smells as it can be seen in the screenshot. And after correcting them I got them to zero.



4 References & resources

Project resources

Resource:	URL/location:
Git repository	https://github.com/RobertoCastro391/TQS_107133/tree/main/HW1
Video demo	https://github.com/RobertoCastro391/TQS_107133/blob/main/HW1/Sreencast%20HW1.mp4 and https://youtu.be/WYv8exDVRkM
QA dashboard (online)	I did not use

Reference materials

- <https://freecurrencyapi.com> – External API for the currencies