HW1: Mid-term assignment report

*Lia Ribeiro de Lima Cardoso [107548]*, v2024-04-09

[1 Introduction 1](#_Toc130550537)

[1.1 Overview of the work 1](#_Toc130550538)

[1.2 Current limitations 1](#_Toc130550539)

[2 Product specification 2](#_Toc130550540)

[2.1 Functional scope and supported interactions 2](#_Toc130550541)

[2.2 System architecture 2](#_Toc130550542)

[2.3 API for developers 2](#_Toc130550543)

[3 Quality assurance 2](#_Toc130550544)

[3.1 Overall strategy for testing 2](#_Toc130550545)

[3.2 Unit and integration testing 2](#_Toc130550546)

[3.3 Functional testing 3](#_Toc130550547)

[3.4 Code quality analysis 3](#_Toc130550548)

[3.5 Continuous integration pipeline [optional] 3](#_Toc130550549)

[4 References & resources 3](#_Toc130550550)

# Introduction

## 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.

This project consists on a WebApp that enables users to search for bus trips between major cities on specific dates and make reservations for these trips. Additionally, the application allows clients to access their reservation information at any time, provided they have the reservation token.

## Current limitations

Currently, all the essential features that were requested for this system are working as expected. However, none of the additional functionalities that were proposed have been implemented yet. Although the BDD with Selenium (Cucumber) tests were implemented, they are not running for unknown reasons.

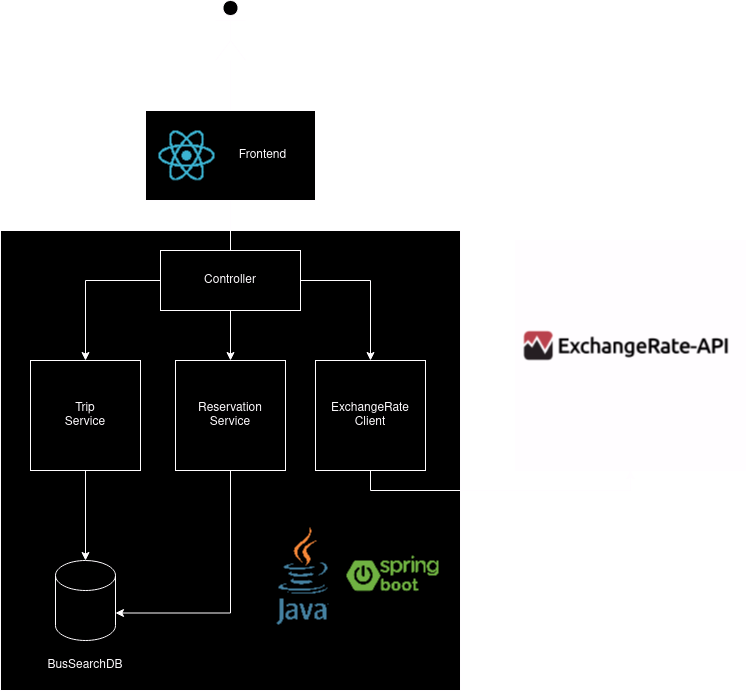
# Product specification

## Functional scope and supported interactions

This system was designed to assist individuals who frequently travel by bus. It offers an application that enables them to easily check information about bus trips and make reservations in an efficient manner.

As a traveler, you can conveniently search for and book your bus journey by specifying your starting point, destination, and travel date. You will be presented with a range of available trips, and once you have made your decision, you can enter your details to confirm your booking. You will receive a unique reservation token which you can use to access your booking information at a later time. For bus companies, my platform offers a robust interface to manage trip schedules, monitor seat availability, and keep track of bookings, all designed to optimize efficiency and enhance the overall travel experience.

## System architecture



The frontend for this project was done using ReactJs with Tailwind and DaisyUI, while the backend was done using Spring Boot and an H2 in-memory database.

## API for developers

## Endpoints

### Exchange Rates

* GET /exchange/
  + Description: Converts a given amount from one currency to another using the latest exchange rate.
  + Query Parameters:
    - amount (Double): The amount to be converted.
    - currency (String): The currency code to convert from.
  + Response: Double - The converted amount in the target currency.

### Trips

* GET /trips/{id}
  + Description: Retrieves a specific trip by its ID.
  + Path Parameters:
    - id (int): The ID of the trip.
  + Response: Trip object containing trip details.
* GET /trips/all\_for\_search
  + Description: Searches for trips based on origin city, destination city, and date.
  + Query Parameters:
    - fromCity (String): The city of departure.
    - toCity (String): The destination city.
    - dateTrip (String): The date of the trip.
  + Response: List of Trip objects matching the search criteria.

### Reservations

* GET /reservations/{token}
  + Description: Retrieves a specific reservation by its token.
  + Path Parameters:
    - token (String): The unique token of the reservation.
  + Response: Reservation object containing reservation details.
* POST /reservations/save
  + Description: Creates a new reservation for a trip.
  + Request Parameters:
    - tripId (int): The ID of the trip to book.
    - firstName (String): The first name of the passenger.
    - lastName (String): The last name of the passenger.
    - email (String): The email address of the passenger.
  + Response: Map containing the reservation object and the token.

The provided API documentation outlines essential features of the application, covering trip planning, booking, and currency conversion. The Exchange Rates endpoint allows for real-time currency conversions, crucial for international travelers. Trips endpoints enable users to search for and retrieve specific trip details, facilitating efficient travel planning. The Reservations section handles booking processes, from viewing existing bookings to creating new ones. This API forms the backbone of the application, ensuring a smooth and integrated user experience for trip management and planning tasks.

# Quality assurance

## Overall strategy for testing

In this project, was adopted a multi-dimensional testing strategy that seamlessly integrates various testing methodologies to ensure the overall quality and reliability of the application. I started with unit testing, which allowed me to scrutinize the behavior of individual components in isolation, ensuring each part functions correctly on its own. To complement this, I integrated testing to evaluate the interactions between components, verifying that they work together as expected to deliver the desired outcomes. While there wasn’t explicitly an use of a Behavior-Driven Development (BDD) framework, my testing approach was heavily influenced by BDD principles, focusing on tests that describe the behavior of the application in a readable and requirement-aligned manner. Recognizing the critical role of external services in our application, it was also incorporated external API testing, ensuring robust and seamless integration with third-party systems. Additionally, UI testing was an integral part of our strategy, aimed at ensuring that the user interface is intuitive and error-free, thereby enhancing the user experience. By blending these diverse testing approaches, the strategy covered all critical facets of the application, from core functionality and integration to user interaction and external service compatibility, ensuring a comprehensive assessment of the software's quality.

## Unit and integration testing

In the development of the project, a strategic approach was taken to apply unit and integration tests across different layers to ensure robustness and reliability.

### Unit Testing

Unit tests were employed primarily in the service and repository layers to validate the functionality of individual components in isolation from the rest of the system. This approach facilitated granular issue identification, ensuring each component of the application performed its intended function correctly. In the service layer, unit tests focused on methods that executed specific business operations, verifying their correct behavior under various scenarios, including normal operations, edge cases, and error conditions. In the repository layer, emphasis was placed on data retrieval and manipulation methods, ensuring accurate interaction with the database.

### Integration Testing

Integration tests were utilized to evaluate the interactions between components, with a particular focus on the communication between controllers and services, as well as between services and repositories. The objective was to verify the data flow and logic across multiple layers, ensuring seamless operation of components in unison to fulfill business requirements. An example of this involved testing controllers by making HTTP requests and asserting responses, which included checking status codes, response bodies, and database side effects, thereby simulating real-world application usage.

### Implementation Strategy

A bottom-up approach was adopted for the implementation strategy, starting with unit tests for the lowest layers (repositories) and progressively advancing to services and controllers. This strategy ensured a stable foundation, with lower-level components being thoroughly tested before their integration into higher-level tests.

### Clarification with Code Snippets

For further clarification, a simplified code snippet from a unit test for the ReservationService is provided below:

@Test

public void testSaveReservation() {

Reservation reservation = new Reservation("token123", "Aveiro", "Porto", "2021-03-01", "10:00", "John", "Doe", "john.doe@example.com");

when(reservationRepository.save(any(Reservation.class))).thenReturn(reservation);

Reservation savedReservation = reservationService.saveReservation(reservation);

assertEquals("token123", savedReservation.getToken());

}

This unit test verifies the saveReservation method in the ReservationService, ensuring it properly saves a reservation and returns the expected token.

In addition, a simplified snippet from an integration test for the ReservationController is as follows:

@Test

public void whenSaveReservationWithValidInput\_thenCreateReservation() throws Exception {

when(tripService.findTripById(1)).thenReturn(mockTrip);

when(reservationService.saveReservation(any(Reservation.class))).thenReturn(mockReservation);

mvc.perform(post("/api/reservations/save")

.param("tripId", "1")

.param("firstName", "John")

.contentType(MediaType.APPLICATION\_JSON))

.andExpect(status().isOk())

.andExpect(jsonPath("$.token", is("token123")));

}

This integration test simulates a POST request to save a reservation, asserting the response is as expected. It demonstrates the controller, service, and repository layers working in concert.

Through this structured approach to testing, the project aimed to ensure both the individual components and their integrations functioned as intended, contributing to the stability and reliability of the application.

## Functional testing

User-facing test cases were designed to reflect real-world scenarios that end-users might encounter while interacting with the application. These test cases focused on key functionalities such as trip search, reservation booking, and currency conversion, ensuring that the application met user expectations and requirements.

### User-Facing Test Cases

1. Trip Search: Test cases were developed to simulate users searching for trips between two cities on a specific date. These tests verified that the application returned a list of available trips matching the search criteria, including details like departure and arrival times, prices, and available seats.
2. Reservation Booking: Test scenarios were created to cover the process of booking a reservation for a selected trip. This included tests for successful booking when seats were available and error handling for cases such as invalid trip IDs, fully booked trips, or incomplete user information.
3. Currency Conversion: Given the international nature of travel, test cases were also considered for currency conversion features, ensuring that users could view trip prices in different currencies based on real-time exchange rates.

### Implementation Strategy

The implementation of these user-facing test cases was carried out using a combination of unit, integration, and UI tests:

* Unit Tests: Validated the underlying logic and algorithms responsible for functionalities like search filters, reservation logic, and currency calculations.
* Integration Tests: Ensured that the controllers correctly handled HTTP requests for trip searches, bookings, and currency conversions, returning the expected responses.
* UI Tests: Utilized Selenium WebDriver to automate interactions with the web application's interface, simulating user actions like filling out search forms, clicking on search buttons, selecting trips, and completing reservation forms.

### Code Snippets

Below is a simplified code snippet from a UI test using Selenium WebDriver for the trip search functionality:

@Test

public void testTripSearch() {

driver.get("http://localhost:8080/");

WebElement fromCityInput = driver.findElement(By.id("fromCity"));

WebElement toCityInput = driver.findElement(By.id("toCity"));

WebElement searchButton = driver.findElement(By.id("searchButton"));

fromCityInput.sendKeys("Aveiro");

toCityInput.sendKeys("Lisbon");

searchButton.click();

List<WebElement> searchResults = driver.findElements(By.className("searchResult"));

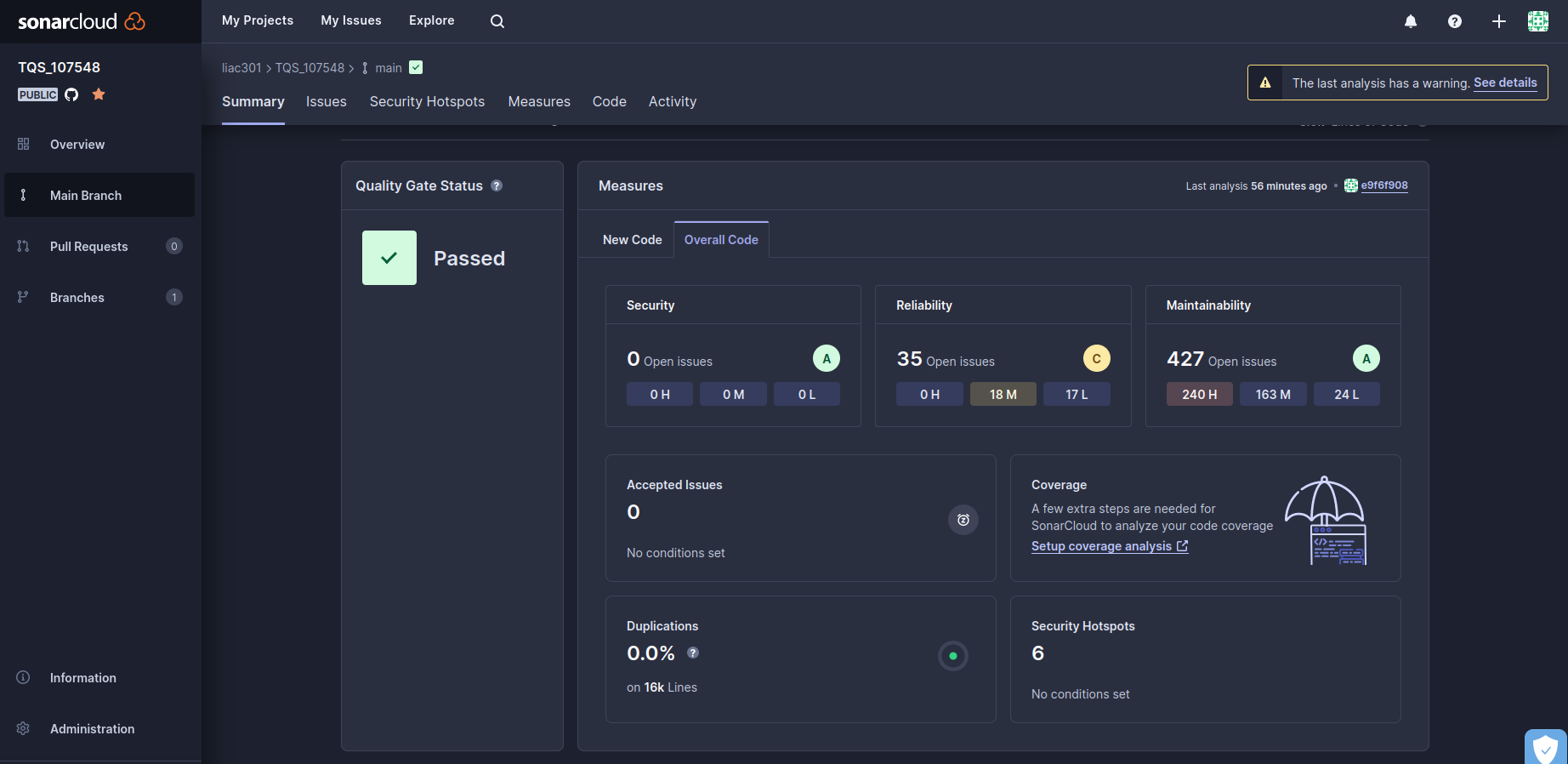
assertFalse(searchResults.isEmpty());

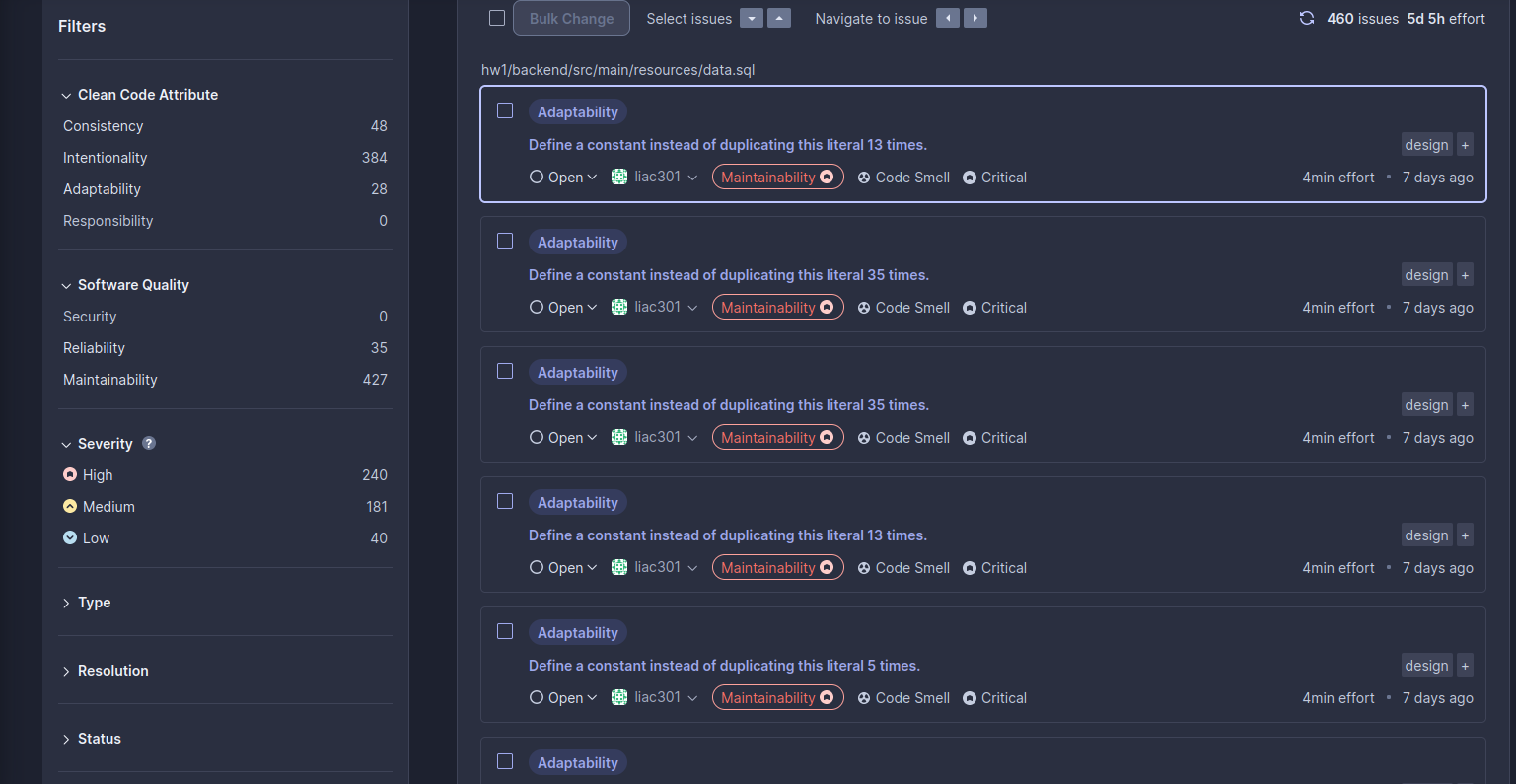
}

This UI test automates the process of entering search criteria (from and to cities) and validates that the search results are displayed as expected.

## Code quality analysis

For static code analysis in the project, SonarCloud was selected as the primary tool. SonarCloud, a cloud-based service, offers comprehensive analysis capabilities, making it highly effective for scrutinizing code for potential issues, such as bugs, vulnerabilities, code smells, and technical debt, while also providing metrics on code complexity and duplication. I’ve only run this tool one time, at the end of the project.





The code has multiple code smells and a technical debt of 5 days and 5 hours, therefore there is a lot to be improved in this project. Unfortunatelly, I haven’t been able to fix this issues.

# References & resources

Project resources

|  |  |
| --- | --- |
| **Resource:** | **URL/location:** |
| Git repository | https://github.com/liacr301/TQS\_107548 |
| Video demo | <https://github.com/liacr301/TQS_107548/blob/main/hw1/Demo_part1.webm>→ Part 1  <https://github.com/liacr301/TQS_107548/blob/main/hw1/Demo_part2.webm> → Part 2 |
| QA dashboard (online) | https://sonarcloud.io/summary/new\_code?id=liacr301\_TQS\_107548 |

Reference materials

External API docs:  
https://www.exchangerate-api.com/docs/standard-requests