HW1: Mid-term assignment report

Sara Figueiredo Almeida [108796], v2024-04-09

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.

For this project we needed to create an API to support a simple bus ticket selling application that was able to give support to searching for bus connections between two cities and make a trip reservation for a passenger, using a generated token to identify the reservation. We were also required to implement a price logic where the user could choose the preferred currency in which to see the trip cost, using an external API.

Current limitations

As for the current project limitations, my cache is not implemented in a persistent way, which means that if the application goes down the data already requested to the external currency exchange API is not kept, new requests will need to be made. I also haven't implemented cache usage statistics.

As for the preferred currency the user can only choose them in the page where is shown the information of the buses available to make the connection between the cities he has chosen , but then for the rest of the reservation logic the price stays in EUR as they are in the database.

Besides that, I couldn't do some of the extras I wished for, such as making an available seat and cancelling reservations logics.

2 Product specification

2.1 Functional scope and supported interactions

This application is destined for anyone who wants to buy a bus ticket for a bus trip between two cities on a specific date.

Inicially, the user inserts the origin, destination and date of the trip they pretend to search for. Then, when clicking the *search* button, a table is presented with all the buses available for that specification. As for the buses, it is presented their number, which identifies them, the origin, destination, date, departure time, arrival time, price and capacity of that trip. In this page, the user can choose between a big selection of currencies to see the prices in the one they prefer (the same user can choose more than one currency and see the price changing each time). So, on this page, the user can choose the bus they want to travel in by clicking the *reserve* button.

After this, a big form is presented so the user can fill in their personal data, such as name, email and address, and payment data such as credit card number and cvv. When clicking *submit* the user is redirected to a page where the token associated to their reservation is showed. They can, then, copy it and click the *check reservation* button so they are led to a page where they can enter that token and see all the details associated to their reservation.

They can then come back to the home page through a button, as well as they can check their reservations from the home page, also through a button.

2.2 System architecture

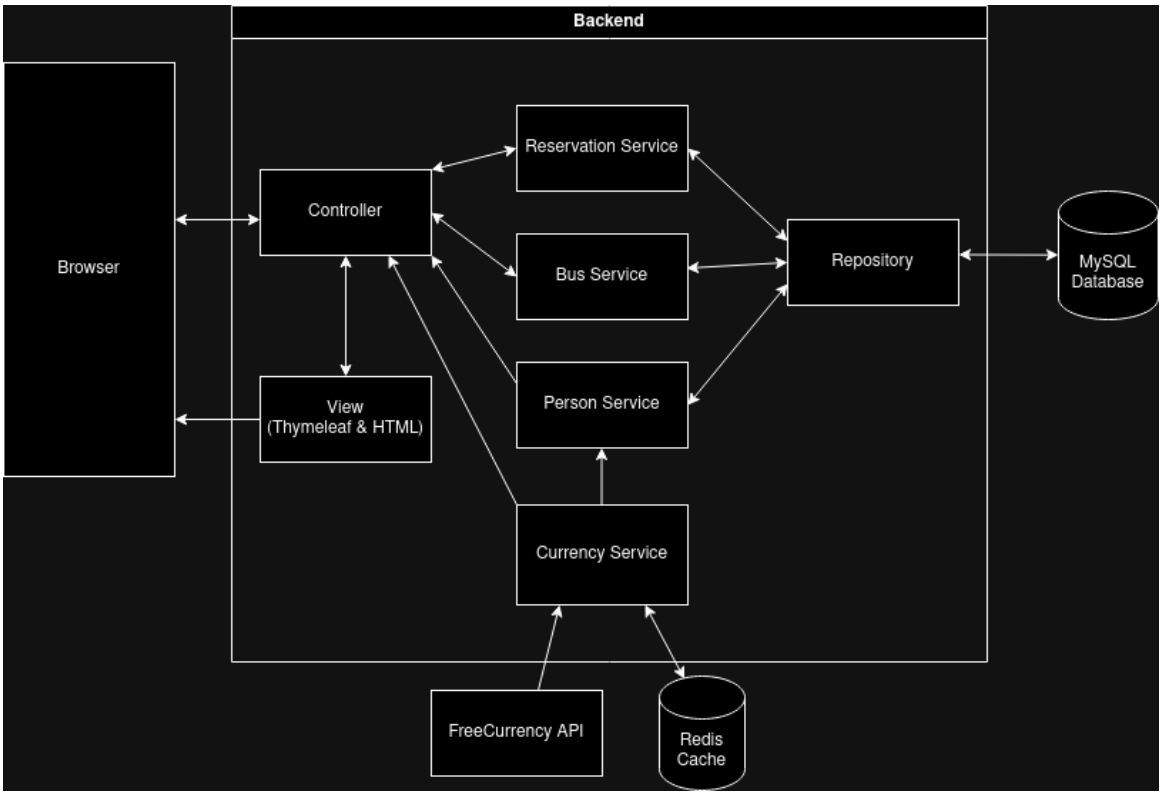

Figura 1. System Architecture

As for the whole backend of the application I used **Spring Boot** , using **Thymeleaf** templates for the views, as well as HTML, CSS and some JS for some flow features.
I used The **FreeCurrency API** for the implementation of the currency exchange feature and I used **redis** for caching the responses from the requests of that API.
As for the database, I used **MYSQL**.

2.3 API for developers

As for the endpoints:

- **GET** *"/"* : Returns a view which is the HTML initial page that has the form for the user to fill with the origin, destination and date of the wanted trip.

- **POST** *"/api/bus"* : Returns the HTML that dynamically present all the buses available for the data filled previously.
- **POST** *"/api/details"* : Returns an HTML with the information of the bus chosen by the user and a form for them to fill with their personal and payment data to proceed with the reservation.
- **POST** *"/api/reservation"* : Returns an HTML with the token that was generated to be associated with the reservation made. In this method, the data filled in before is associated with a new reservation.
- **POST** *"/api/reservation/check"* : Returns the HTML where the user can enter a token and see the reservation details associated with it, or not, if it doesn't have any.
- **GET** *"api/reservation/check"* : Receives the token entered by the user and gets the details of the reservation associated with it.

- **GET** *"/api/currencies"* : Used to retrieve all the currencies available in the currency exchange FreeCurrencyAPI
- **GET** *"/api/exchange"* : Used to retrieve from the API the exchange rate when changing from one currency to another.

3 Quality assurance

3.1 Overall strategy for testing

When it comes to the overall testing strategy, I made things side by side. This is, i started with some frontend and some endpoints in the controller to test the flow logic and data transmission. After that, tests were made along with the backend implementation, starting with unitary tests, followed by integration tests and finally some more functional tests, including frontend using Selenium and Cucumber.

3.2 Unit and integration testing

As for the unit tests, they were conducted mainly for testing the services, mocking the behaviour of the repositories, using Mockito.

```java
@Test
@DisplayName("Test getBuses for invalid origin, destination and date")
void testGetBusesForInvalidOriginDestinationAndDate() {
    List<Bus> buses = bus_service.getBuses(origin:"Lisboa", destination:"Aveiro", date:"2024-04-11");
    assertEquals(expected:null, buses);
}

@Test
@DisplayName("Test getBuses for invalid origin and destination with valid date")
void testGetBusesForInvalidOriginAndDestinationWithValidDate() {
    List<Bus> buses = bus_service.getBuses(origin:"Lisboa", destination:"Porto", date:"2024-04-12");
    assertEquals(expected:0, buses.size());
}

@Test
@DisplayName("Test getBuses for valid origin, destination and null date")
void testGetBusesForValidOriginDestinationAndNullDate() {
    List<Bus> buses = bus_service.getBuses(origin:"Viseu", destination:"Aveiro", date:null);
    assertNotNull(buses);
    assertEquals(expected:0, buses.size());
}
```

Figura 2.

BusService_UnitTest.java

When it comes to the integration tests, they were conducted to test the controllers, using Mockmvc, to make sure the backend was functioning correctly.

```java
@Test
@DisplayName("Test getReservations endpoint with valid token")
void testGetReservationsEndpointWithValidToken() throws Exception {
    Reservation reservation2 = new Reservation(
        creditCardNumber:1234567890123456L,
        creditCardMM:12L,
        creditCardYY:25L,
        creditCardCVV:123L,
        jose,
        bus3
    );
    reservation2.setToken(UUID.randomUUID().toString());

    reservation_repository.save(reservation2);

    String validToken = reservation2.getToken();
    mvc.perform(get(urlTemplate:"/api/reservation/check")
            .param(name:"token", validToken)
            .contentType(MediaType.APPLICATION_JSON))
            .andExpect(status().isOk())
            .andExpect(jsonPath(expression:"$.id", notNullValue()));
}
```

Figura 3.

ReservationControllerIT.java

3.3 Functional testing

For the user-facing tests, I used Selenium to perform the frontend testing while recording it and then implemented the Page Object Pattern in addition to using Cucumber to simulate the reservation booking flow as explained in the 2.1.Functional scope and supported interactions section. Here's a snippet of the .feature file and the Steps.java implemented:

```
@hw1_sample
Feature: HW1

    Scenario: Reserve Ticket
        Given the user is on the index page
        When the user writes 'São Pedro do Sul' on the from input
        And the user writes 'Aveiro' on the to input
        And the user writes '2024-04-12' on the date input
        And the user clicks on the Search button
        Then the user should be redirected to the page with the title 'Table'
        When the user the user selects the first available bus, with the bus_number 24 and price 7
        Then the user should be redirected to the page with the title 'Personal'
        When the user fills in the information needed:
            | name              | Sara                      |
            | surname           | Almeida                   |
            | email             | sarafalmeida@ua.pt        |
            | phoneNumber       | 123456789                 |
            | address           | Rua da Alegria            |
            | city              | Viseu                     |
            | postalCode        | 3660-231                  |
            | country           | Portugal                  |
            | creditCardNumber  | 21312332534653            |
```

Figura 4. HW1.feature

```java
public class SeleniumSteps {

  private WebDriver driver = new FirefoxDriver();
  private String token;

  @Given("the user is on the index page")
  public void on_index_page() {
    driver.get(url:"http://localhost:8080/");
    driver.manage().window().setSize(new Dimension(width:1472, height:764));
  }

  @When("the user writes {string} on the from input")
  public void write_from_input(String from) {
    driver.findElement(By.id(id:"from")).click();
    driver.findElement(By.id(id:"from")).sendKeys(from);
  }

  @And("the user writes {string} on the to input")
  public void write_to_input(String to) {
    driver.findElement(By.id(id:"to")).click();
    driver.findElement(By.id(id:"to")).sendKeys(to);
  }
```

Figura 5. SeleniumSteps.java

3.4 Code quality analysis

The main tool used for code analysis was the SonarCube, but I also used the jacoco reports in an initial fase. As for the coverage, I got 96.1% but my project presents some medium-level issues, mainly due to the logs implementation.

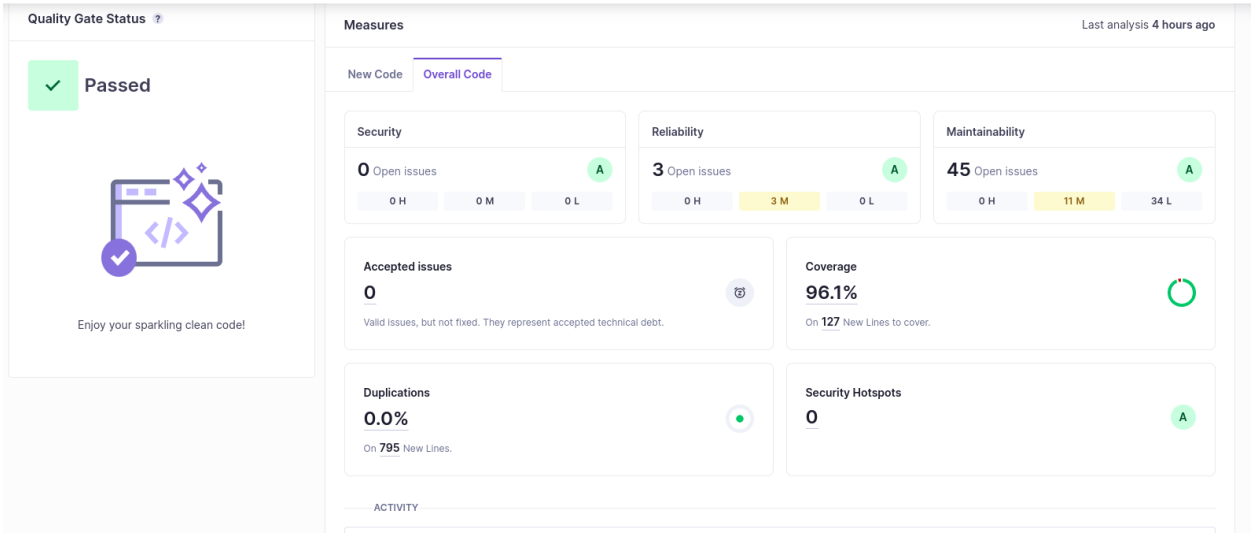

Figura 6. SonarQube Overview

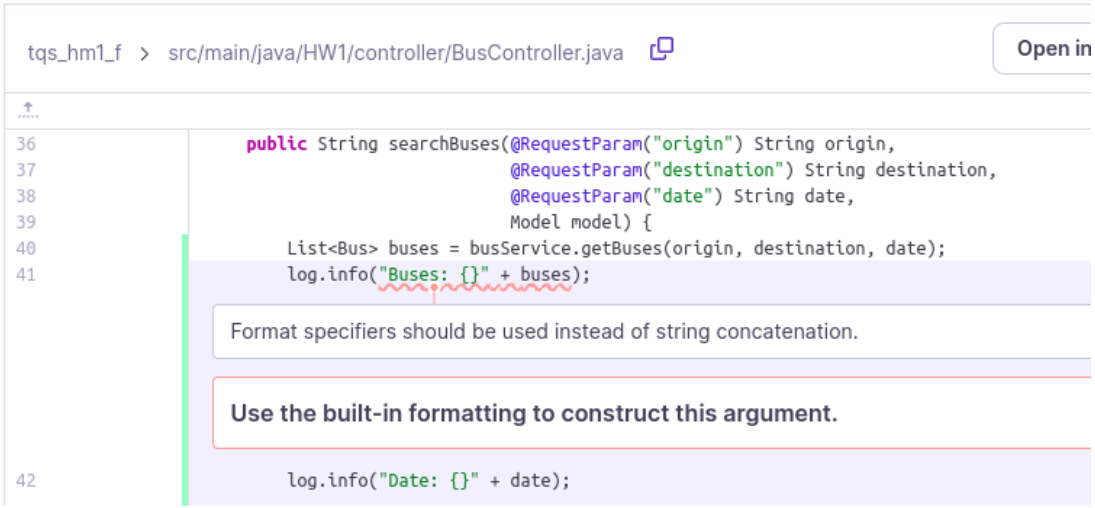

Figura 7. Exemple of SonarQube found issue

3.5 Continuous integration pipeline [optional]

I was not able to implement the pipeline.

4 References & resources

Project resources

Resource:	URL/location:
Git repository	https://github.com/SardinhaAlmeida/TQS_108796/tree/main/HW1
Video demo	https://github.com/SardinhaAlmeida/TQS_108796/tree/main/HW1/handouts

QA dashboard (online)	---
CI/CD pipeline	---
Deployment ready to use	---

Reference materials

https://www.thymeleaf.org/doc/articles/layouts.html
https://www.baeldung.com/spring-cache-tutorial
https://app.freecurrencyapi.com/request-playground
https://mvnrepository.com/
https://www.learninjava.com/4-ways-to-test-webclient-mocking/