universidade de aveiro
departamento de eletrónica, telecomunicações e informática

HW1: Mid-term assignment report

João Nuno da Silva Luís [107403], 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.
The purpose of this work is to develop a full-stack application that allows users to search for trips between cities, and then, after a trip is selected, provide ways to “book” a ticket on that trip.

The developed application is composed by a frontend and backend. Despite dockerfile’s and Docker-Compose files are presented, running docker-compose instance is not working due to problems on node_modules on frontend. So, to run the application, it’s necessary to have 2 terminals open, running **mvn spring-boot:run** inside backend folder and **npm run dev** inside frontend folder.

Because the main focus of this assignment is on the test part, different test tools were used:

- Junit 5 for unit and repository tests;
- Mockito and AssertJ for service and repository tests;
- MockMvc, hamcrest and Mockito for the integration tests;
- Selenium + Cucumber for the frontend UI;
- Sonar Cloud for Static Code Quality checks (working on a GitHub action).

The Spring REST API is documented using OpenAPI 3.0 and is accessible at the following URL: http://localhost:8080/swagger-ui/index.html.

1.2 Current limitations

The basic features have all been implemented. However, in some fields, there isn’t the proper validation required for a production app. Additional feature G’ is also done.
The currency API implementation has little tests, because it was difficult to mock it.
The cache developed for this project is also a little bit rudimentary. I used a HashMap and then passed a variable that was the Time-To-Live, but in case of a bigger/production system, probably was going to be necessary another mechanism for cache.

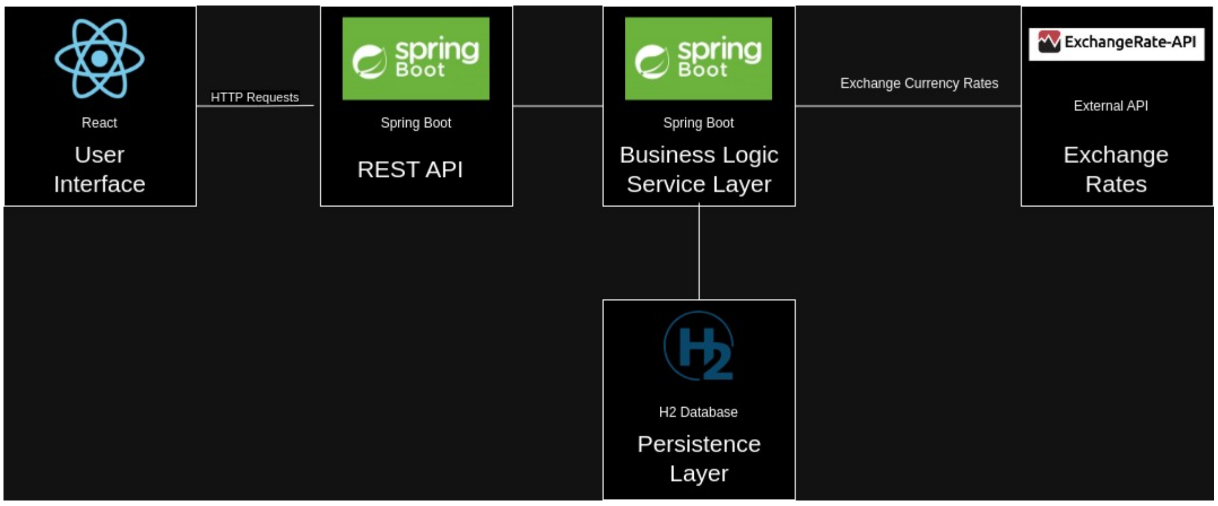
2 Product specification

2.1 Functional scope and supported interactions

The current application was designed for accommodating the needs of a user who intends to search for a ticket between two cities, list the bus trips available between them, book the desired trip with his information details and then see the confirmation.
There is only a single actor in this application, that is a a client that want to follow the behavior described above, because in this app, no administrative features were developed.

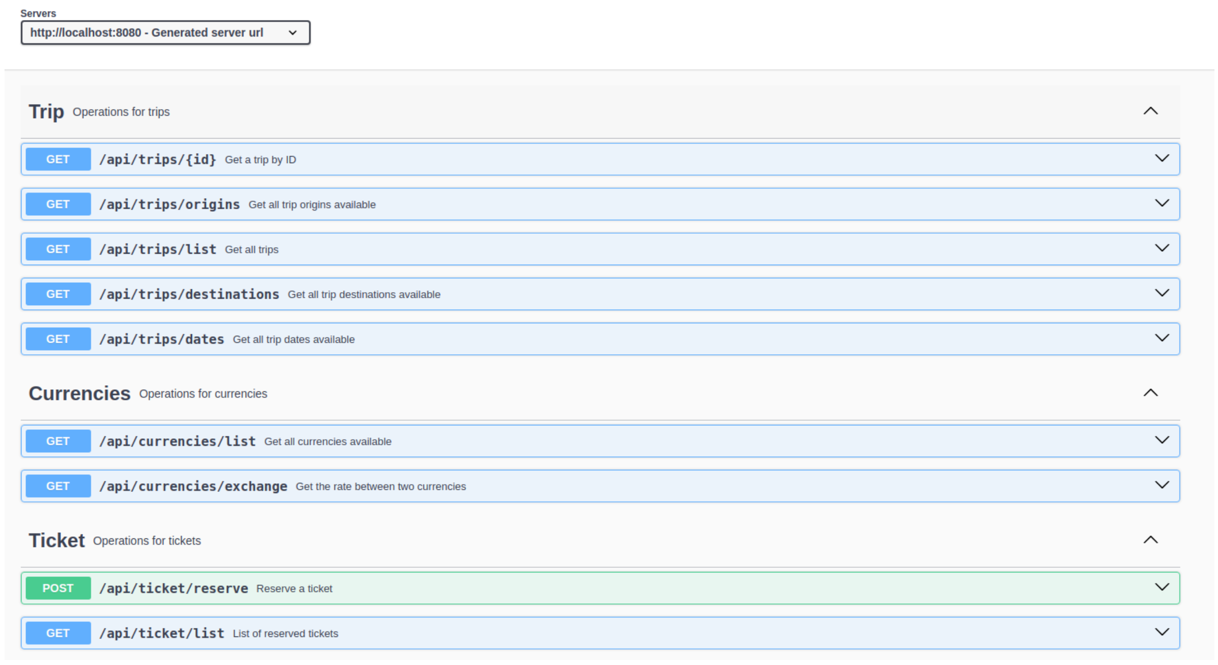
Therefore, the user is going to follow this Scenario/Steps:
1. User opens the app and selects the origin city, the date, the destination city and currency of the wanted trip;
2. The system retrieves available bus trips and some info about them;
3. Input user information (First and Last Names, City and email), as well as previous informations about the trip and ticket cost;
4. Click on the button “Purchase Ticket”;
5. Check all details related to the bought ticket on a confirmation page.

2.2 System architecture



2.3 API for developers

Endpoints developed for this assignment:


Servers

http://localhost:8080 - Generated server url

Trip Operations for trips

GET /api/trips/{id} Get a trip by ID

GET /api/trips/origins Get all trip origins available

GET /api/trips/list Get all trips

GET /api/trips/destinations Get all trip destinations available

GET /api/trips/dates Get all trip dates available

Currencies Operations for currencies

GET /api/currencies/list Get all currencies available

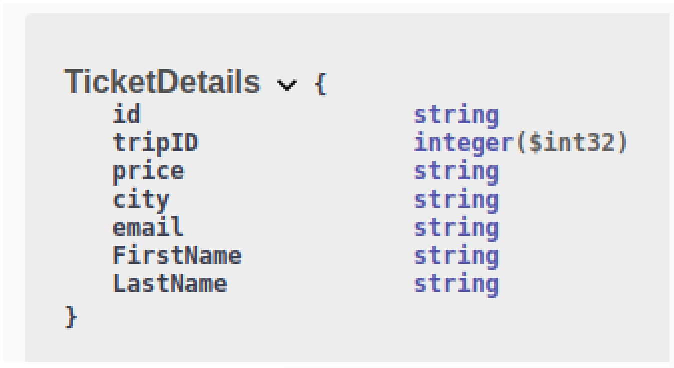
GET /api/currencies/exchange Get the rate between two currencies

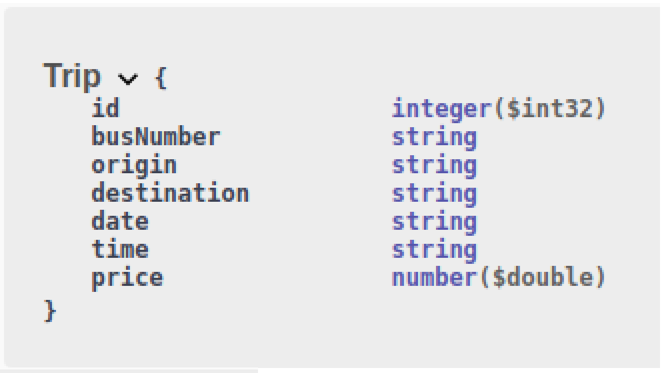
Ticket Operations for tickets

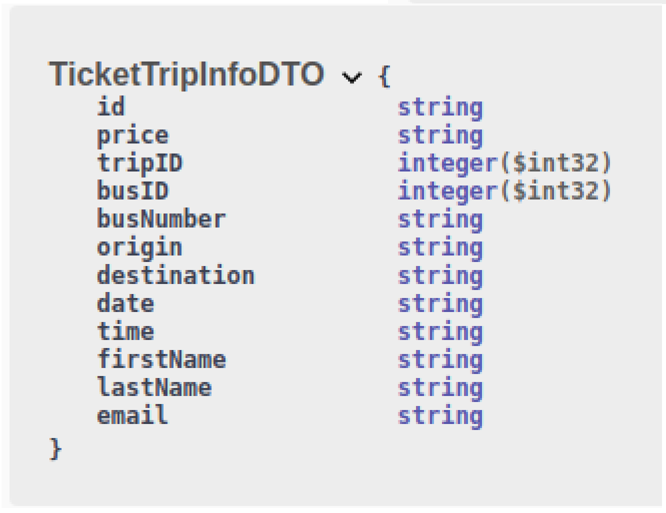
POST /api/ticket/reserve Reserve a ticket

GET /api/ticket/list List of reserved tickets

The following schemas were also developed:


```
TicketDetails ⌄ {
    id              string
    tripID          integer($int32)
    price           string
    city            string
    email           string
    FirstName       string
    LastName        string
}
```


```
Trip ⌄ {
    id              integer($int32)
    busNumber       string
    origin          string
    destination     string
    date            string
    time            string
    price           number($double)
}
```


```
TicketTripInfoDTO ⌄ {
    id              string
    price           string
    tripID          integer($int32)
    busID           integer($int32)
    busNumber       string
    origin          string
    destination     string
    date            string
    time            string
    firstName       string
    lastName        string
    email           string
}
```

3 Quality assurance

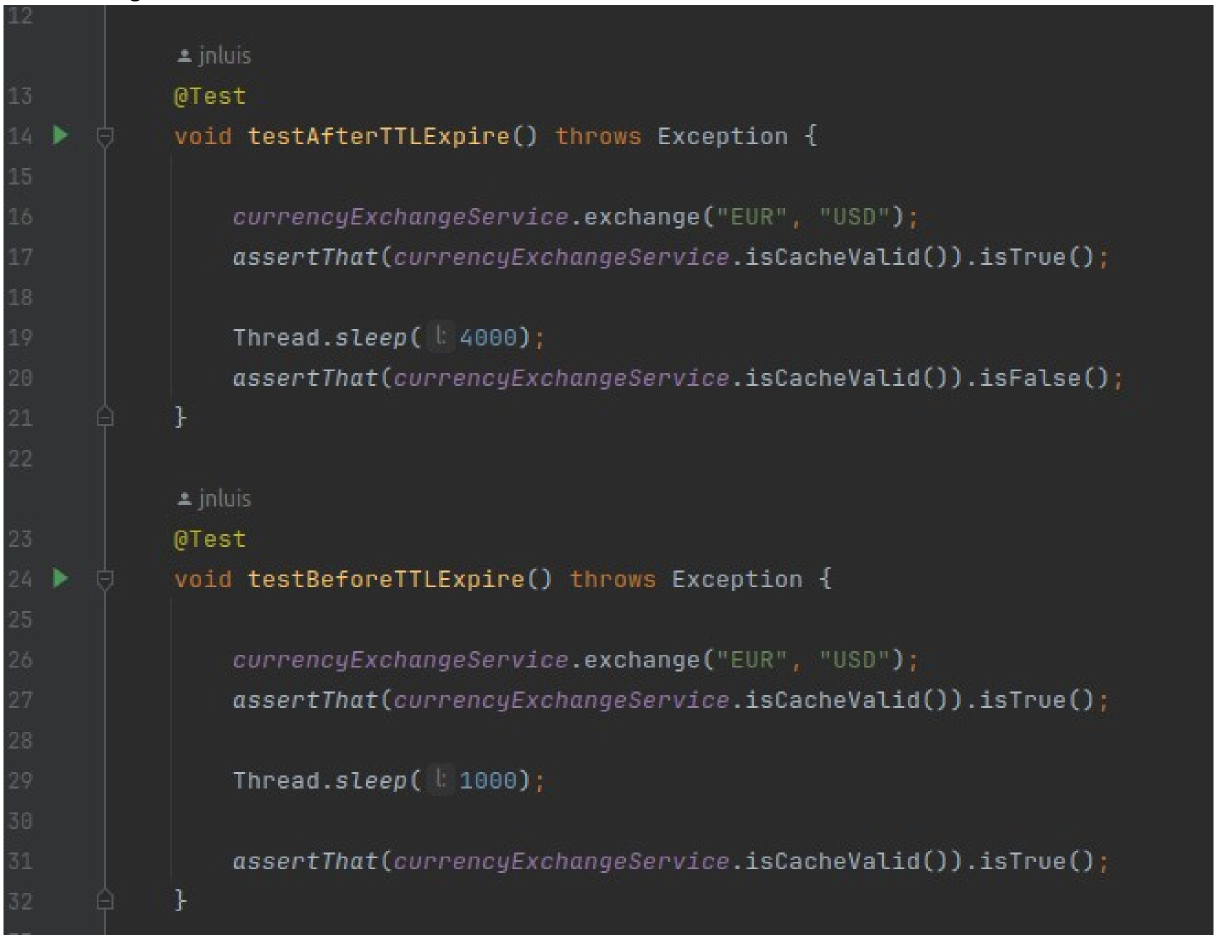
3.1 Overall strategy for testing

To start the project, I started by creating the models I was thinking that were going to be necessary, alongside the creation of the frontend, because this visualization helped me to understand which attributes were going to be necessary. Also, the most part of the application was based on Blaze Demo from the labs exercises.
After that, I implemented the overall logic of the application creating controllers, repositories, and services.
I implemented integration with the External API only after this, and some tests to this parts.
Then I implemented the cache system.
The Selenium+Cucumber tests were left to the final part, and due to lack of time, are incomplete.

Because I started using SonarCloud early on the project, I was correcting some issues that he would give me alongside the development of the project, instead of do that just when the full application was done.

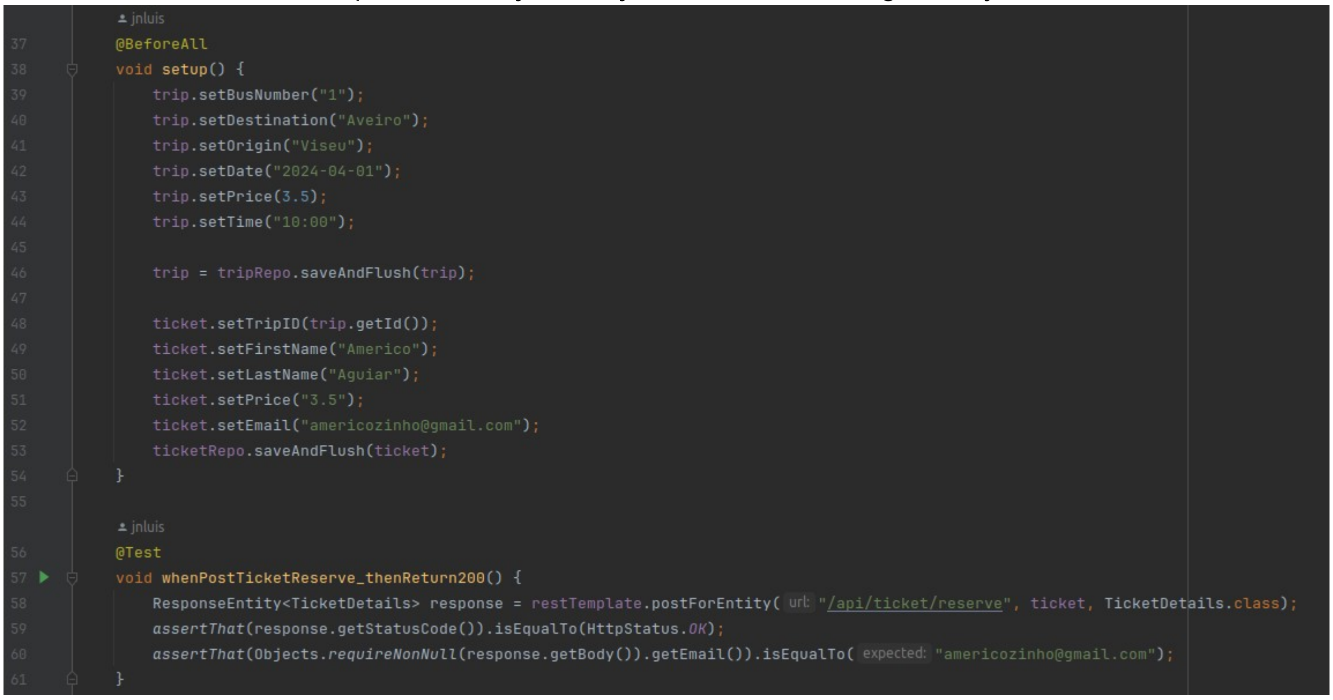
3.2 Unit and integration testing

The idea of the this UnitTests were check if the TTL component in the cache system was working or not.



```java
    @Test
    void testAfterTTLExpire() throws Exception {

        currencyExchangeService.exchange("EUR", "USD");
        assertThat(currencyExchangeService.isCacheValid()).isTrue();

        Thread.sleep( l: 4000);
        assertThat(currencyExchangeService.isCacheValid()).isFalse();
    }

    @Test
    void testBeforeTTLExpire() throws Exception {

        currencyExchangeService.exchange("EUR", "USD");
        assertThat(currencyExchangeService.isCacheValid()).isTrue();

        Thread.sleep( l: 1000);

        assertThat(currencyExchangeService.isCacheValid()).isTrue();
    }
```

I used TestRestTemplate to verify if all layers were functioning as they should.

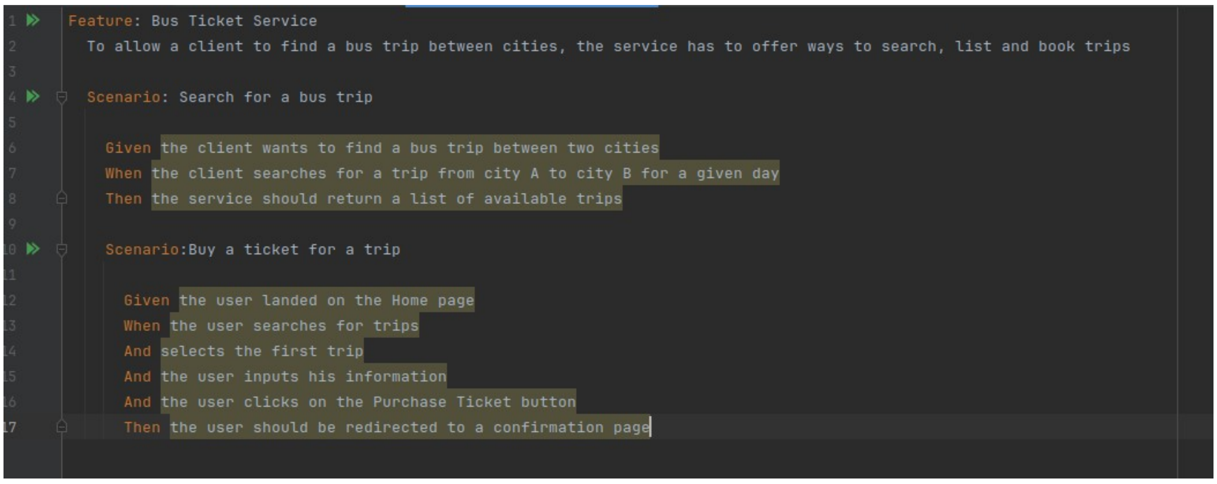


```java
    jnluis
    @BeforeAll
    void setup() {
        trip.setBusNumber("1");
        trip.setDestination("Aveiro");
        trip.setOrigin("Viseu");
        trip.setDate("2024-04-01");
        trip.setPrice(3.5);
        trip.setTime("10:00");

        trip = tripRepo.saveAndFlush(trip);

        ticket.setTripID(trip.getId());
        ticket.setFirstName("Americo");
        ticket.setLastName("Aguiar");
        ticket.setPrice("3.5");
        ticket.setEmail("americozinho@gmail.com");
        ticketRepo.saveAndFlush(ticket);
    }

    jnluis
    @Test
    void whenPostTicketReserve_thenReturn200() {
        ResponseEntity<TicketDetails> response = restTemplate.postForEntity( url: "/api/ticket/reserve", ticket, TicketDetails.class);
        assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
        assertThat(Objects.requireNonNull(response.getBody()).getEmail()).isEqualTo( expected: "americozinho@gmail.com");
    }
```

3.3 Functional testing

For the functional testing, Cucumber and Selenium Web Driver were the way to go.

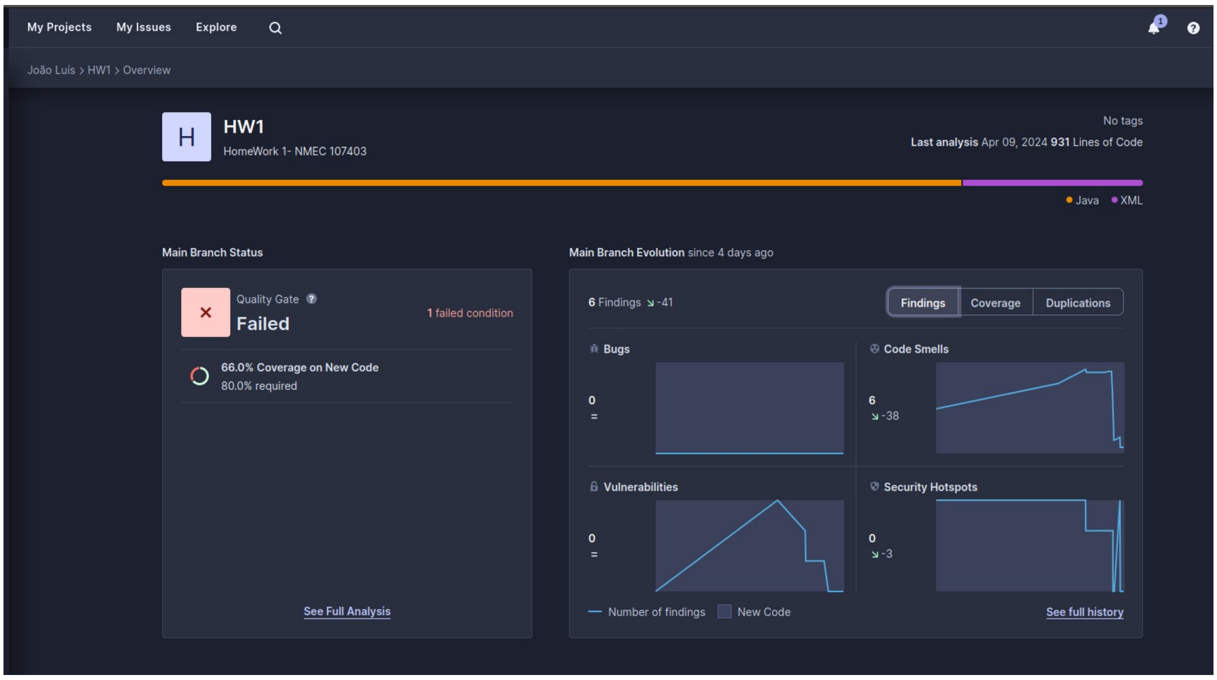
I used Cucumber to represent the workflow of the user, and the next step would be write the tests using Selenium.



```gherkin
Feature: Bus Ticket Service
  To allow a client to find a bus trip between cities, the service has to offer ways to search, list and book trips

  Scenario: Search for a bus trip

    Given the client wants to find a bus trip between two cities
    When the client searches for a trip from city A to city B for a given day
    Then the service should return a list of available trips

    Scenario:Buy a ticket for a trip

      Given the user landed on the Home page
      When the user searches for trips
      And selects the first trip
      And the user inputs his information
      And the user clicks on the Purchase Ticket button
      Then the user should be redirected to a confirmation page
```

3.4 Code quality analysis

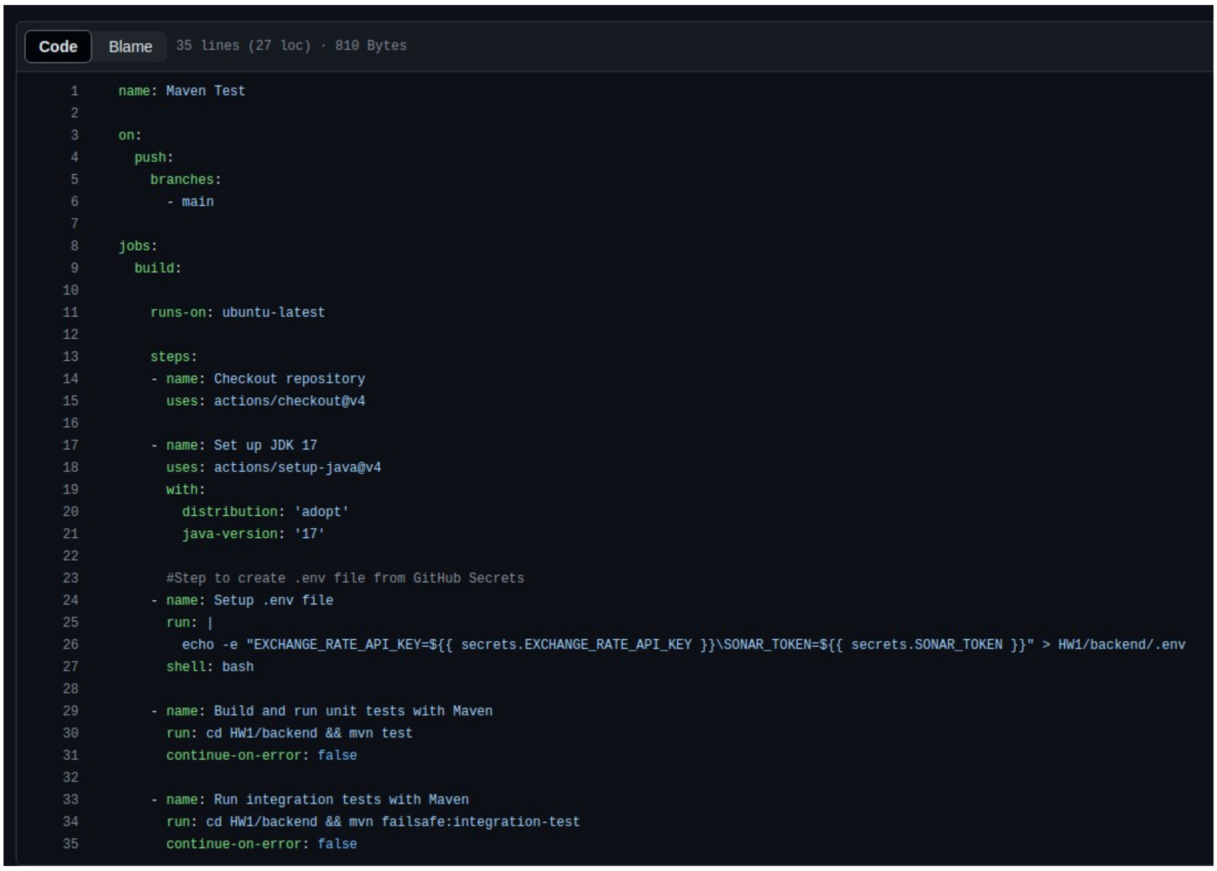
For the static code quality analysis, I used SonarCloud with Jacoco for the coverage reports. The Jacoco part was only added because we used it on labs, and I wanted to see the percentages, but I did not waste much time looking to Jacoco.
With SonarCloud, I really found some things really interesting, specially on security side, and SSRF. Most of the issues presented were easy to adress, and most of them I did not know that were a problem. What I found most strange, was the fact that as I worked out on the gived issues, the coverage actually dropped, ended below 80% (it was my objective to comply to)



3.5 Continuous integration pipeline [optional]

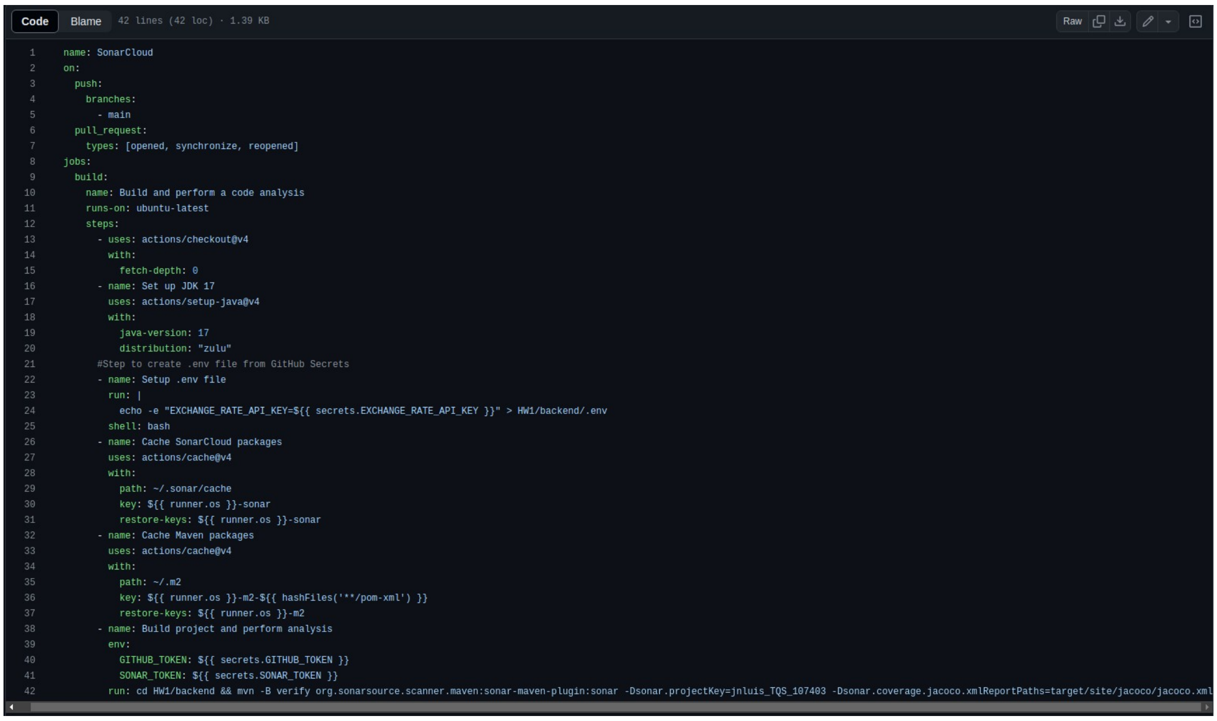
A CI pipeline was implemented for both automatic code testing and static analysis using Github Actions.
It consists of two actions with the following jobs:
- Code testing -> runs mvn test and mvn failsafe integration-test.
- Static code analysis -> static analysis using SonarCloud + Jacoco.

Because I am using both SONAR_TOKEN and External API TOKEN on a .env file, it was necessary to put them as secrets on repository and then do the 26 line trick (on first photo).
Here is the code for code testing workflow:



Code Blame 35 lines (27 loc) · 810 Bytes

```yaml
name: Maven Test

on:
  push:
    branches:
      - main

jobs:
  build:

    runs-on: ubuntu-latest

    steps:
    - name: Checkout repository
      uses: actions/checkout@v4

    - name: Set up JDK 17
      uses: actions/setup-java@v4
      with:
        distribution: 'adopt'
        java-version: '17'

      #Step to create .env file from GitHub Secrets
    - name: Setup .env file
      run: |
        echo -e "EXCHANGE_RATE_API_KEY=${{ secrets.EXCHANGE_RATE_API_KEY }}\SONAR_TOKEN=${{ secrets.SONAR_TOKEN }}" > HW1/backend/.env
      shell: bash

    - name: Build and run unit tests with Maven
      run: cd HW1/backend && mvn test
      continue-on-error: false

    - name: Run integration tests with Maven
      run: cd HW1/backend && mvn failsafe:integration-test
      continue-on-error: false
```



Code Blame 42 lines (42 loc) · 1.39 KB

```yaml
name: SonarCloud
on:
  push:
    branches:
      - main
  pull_request:
    types: [opened, synchronize, reopened]
jobs:
  build:
    name: Build and perform a code analysis
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
        with:
          fetch-depth: 0
      - name: Set up JDK 17
        uses: actions/setup-java@v4
        with:
          java-version: 17
          distribution: "zulu"
      #Step to create .env file from GitHub Secrets
      - name: Setup .env file
        run: |
          echo -e "EXCHANGE_RATE_API_KEY=${{ secrets.EXCHANGE_RATE_API_KEY }}" > HW1/backend/.env
        shell: bash
      - name: Cache SonarCloud packages
        uses: actions/cache@v4
        with:
          path: ~/.sonar/cache
          key: ${{ runner.os }}-sonar
          restore-keys: ${{ runner.os }}-sonar
      - name: Cache Maven packages
        uses: actions/cache@v4
        with:
          path: ~/.m2
          key: ${{ runner.os }}-m2-${{ hashFiles('**/pom-xml') }}
          restore-keys: ${{ runner.os }}-m2
      - name: Build project and perform analysis
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
          SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}
        run: cd HW1/backend && mvn -B verify org.sonarsource.scanner.maven:sonar-maven-plugin:sonar -Dsonar.projectKey=jnluis_TQS_107403 -Dsonar.coverage.jacoco.xmlReportPaths=target/site/jacoco/jacoco.xml
```

4 References & resources

Project resources

Resource:	**URL/location:**
Git repository	https://github.com/jnluis/TQS_107403
Video demo	Video Available under reports/ directory
QA dashboard (online)	https://sonarcloud.io/summary/overall?id=jnluis_TQS_107403https://github.com/jnluis/TQS_107403/tree/main/.github/workflows
CI/CD pipeline	https://sonarcloud.io/summary/overall?id=tqs-hw_road-roamhttps://github.com/jnluis/TQS_107403/tree/main/.github/workflows
Deployment ready to use	Not done

Reference materials

Currency API:

- https://www.exchangerate-api.com/

API Document Generation

- https://www.baeldung.com/spring-rest-openapi-documentation

CI Github SonarCloud:

- https://github.com/SonarSource/sonarcloud-github-action

Sonar Cloud Website:

https://sonarcloud.io/