HW1: Mid-term assignment report

Tiago Filipe Gonçalves Pereira [108546], 2024-04-10

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. In this sense, I've developed a full stack application that allows for reservations of trips between european cities. Measures have been taken in order to boost comodity, such as expedition of return trips reservation, and currency exchange rates at user disposal.

1.2 Current limitations

All implemented features are working fine. However, due to time limitations, the seating arrangement is calculated, at the backend, meaning, no 2 reservations can choose the same seat in the same connection, but that decision is made by the backend, not the user, as there was no time to develop the needed front-end.

Product specification

1.3 Functional scope and supported interactions

The ticket reservation application facilitates the booking of tickets for buses. The main actors who will use the application are:

1. **Customers:** Users who want to book tickets for their travel needs.

Main Usage Scenario:

1. **Customer Reservation:** Customers can search for available transportation services based on origin, destination, and date. They can then select the desired service and proceed to book tickets with the expedition of their return trip, if needed.

1.4 System architecture

The system architecture follows a typical client-server model, with the application split into frontend and backend components.

Frontend:

- The frontend is developed using Thymeleaf, a server-side Java template engine for web and standalone environments.
- Thymeleaf templates are rendered on the server-side and served to the client browser.

Backend:

- The backend is built using Spring Boot, a popular Java framework for creating standalone, production-grade applications.
- It consists of a web controller responsible for rendering views using Thymeleaf and communicating with the REST API.
- The REST API handles requests from the frontend and interacts with a MySQL database to fetch or update ticket reservation information.

Technologies/Frameworks Used:

1. **Frontend:**

    - Thymeleaf: For server-side HTML rendering.
    - HTML/CSS/JavaScript: For frontend development and user interface interaction.

2. **Backend:**
    - Spring Boot: For creating RESTful APIs and handling HTTP requests.
    - Spring Data JPA: For interacting with the MySQL database.
    - Hibernate: As the ORM (Object-Relational Mapping) tool for database operations.
    - Java: As the primary programming language for backend development.
3. **Database:**
    - MySQL: Relational database management system for storing ticket reservation data running on docker.

```
+-----------------------+
|      Frontend         |
|   (Thymeleaf views)   |
+-----------+-----------+
            |
            v
+-----------------------+
|       Web Controller  |
|   (Spring Boot Backend) |
+-----------+-----------+
            |
            v
+-----------------------+
|         REST API      |
|   (Spring Boot Backend) |
+-----------+-----------+
            |
            v
+-----------------------+
|       MySQL Database  |
+-----------------------+
```

1.5 API for developers

1.5.1 API Documentation

1.5.1.1 Origin Cities

- **Description:** Retrieves a list of available origin cities.
- **URL:** `/api/v1/origins`
- **Method:** `GET`
- **Response:**
    - Status Code: `200 OK`
    - Content Type: `application/json`

    - Body: List of origin cities in JSON format.

Destination Cities

- **Description:** Retrieves a list of available destination cities.
- **URL:** `/api/v1/destinations`
- **Method:** `GET`
- **Response:**
    - Status Code: `200 OK`
    - Content Type: `application/json`
    - Body: List of destination cities in JSON format.

Terminals

- **Description:** Retrieves a list of terminals.
- **URL:** `/api/v1/terminals`
- **Method:** `GET`
- **Response:**
    - Status Code: `200 OK`
    - Content Type: `application/json`
    - Body: List of terminals in JSON format.

Connections

- **Description:** Retrieves connections between origin and destination cities.
- **URL:** `/api/v1/connections`
- **Method:** `GET`
- **Query Parameters:**
    - `origin`: Origin city name.
    - `destination`: Destination city name.
    - `fromDate` (optional): Start date for filtering connections.
    - `toDate` (optional): End date for filtering connections.
- **Response:**
    - Status Code: `200 OK`
    - Content Type: `application/json`
    - Body: List of connections in JSON format.

Tickets

- **Description:** Retrieves a ticket reservation by token.
- **URL:** `/api/v1/tickets`
- **Method:** `GET`

- **Query Parameters:**
  - `token`: Unique token for the ticket reservation.
- **Response:**
  - Status Code: `200 OK`
  - Content Type: `application/json`
  - Body: Ticket reservation details in JSON format if found, or empty response if not found.
- **Description:** Creates a new ticket reservation.
- **URL:** `/api/v1/tickets`
- **Method:** `POST`
- **Request Body:** JSON object containing purchase form details.
- **Response:**
  - Status Code: `200 OK` if successful, `500 Internal Server Error` if failed.
  - Content Type: `application/json`
  - Body: Ticket reservation details in JSON format if created successfully, or null if failed.

Currency

- **Description:** Changes the currency used for ticket reservation.
- **URL:** `/api/v1/currency`
- **Method:** `POST`
- **Request Body:** String containing the new currency code.
- **Response:**
  - Status Code: `200 OK`
  - Content Type: `application/json`
  - Body: Exchange rate for the new currency in JSON format.

Quality assurance

1.6 Overall strategy for testing

Initially did TDD. Then, due to time reasons, switched to TAD. Used Rest Assured to test the API, and have a commented test using cucumber. Didn't use it in the

final submission as it was having some problems at testing time that I didn't had time to resolve.

1.7 Unit and integration testing

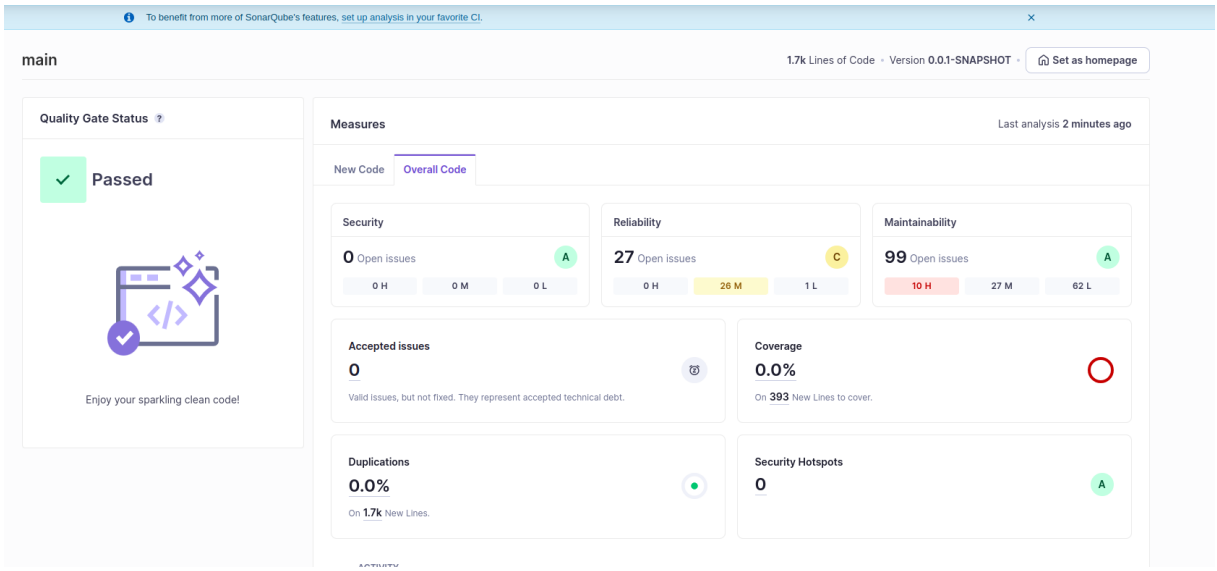
Tested the cache behaviour, Util functions, such as calculate distance between two cities, and tested multi-services functions. This was to make sure the main parts of the code worked just as expected.
To implement this, I mocked the necessary dependencies with mockito.

1.8 Functional testing

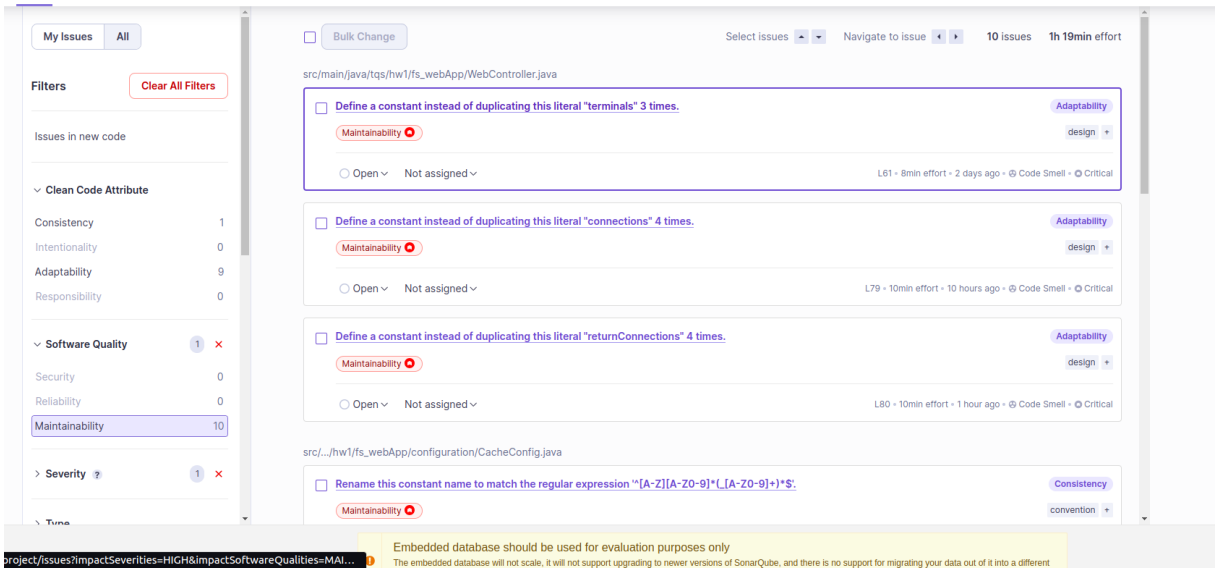
Used Testing with PageObjects and WebDrivers. Tested the main useCases: List Terminals, Buy Oneway Trip, Buy Two Way Trip. Only was left to test the check ticket after reservation use case.

1.9 Code quality analysis

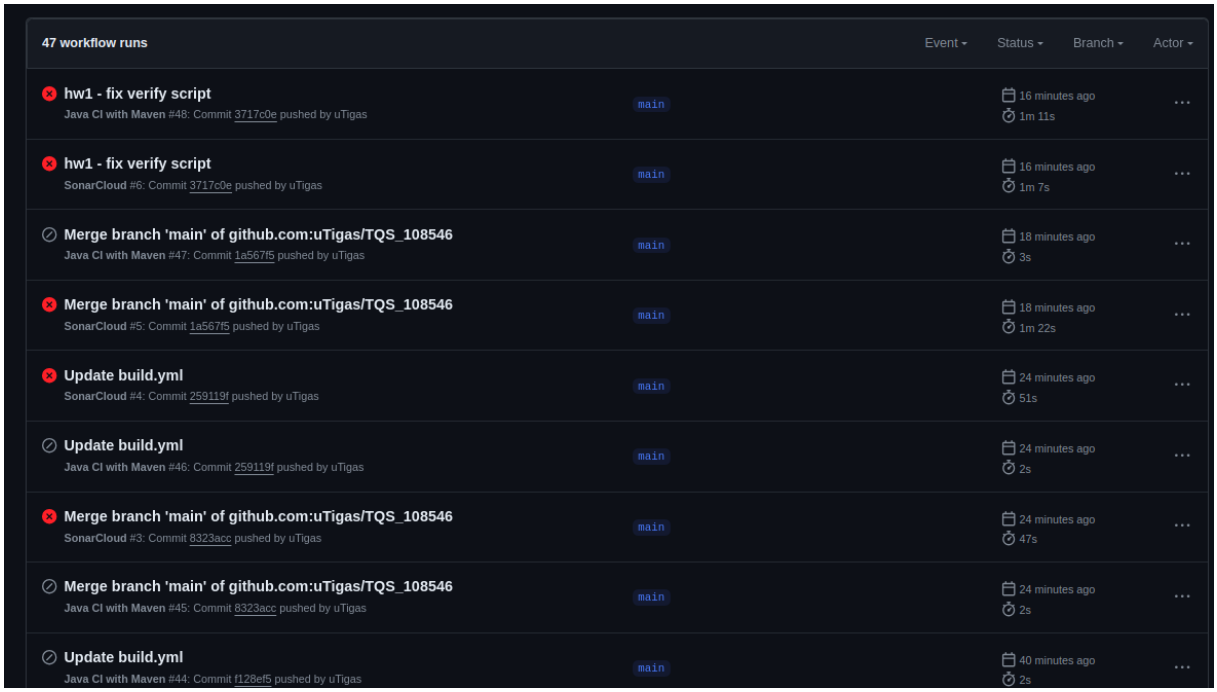
Tried to use sonar cloud, but due to tests not executing on Github Actions, had do resort to local scan.
Realised my coding stile makes bad use of efficiently declaring variables... No other significant issues arose.

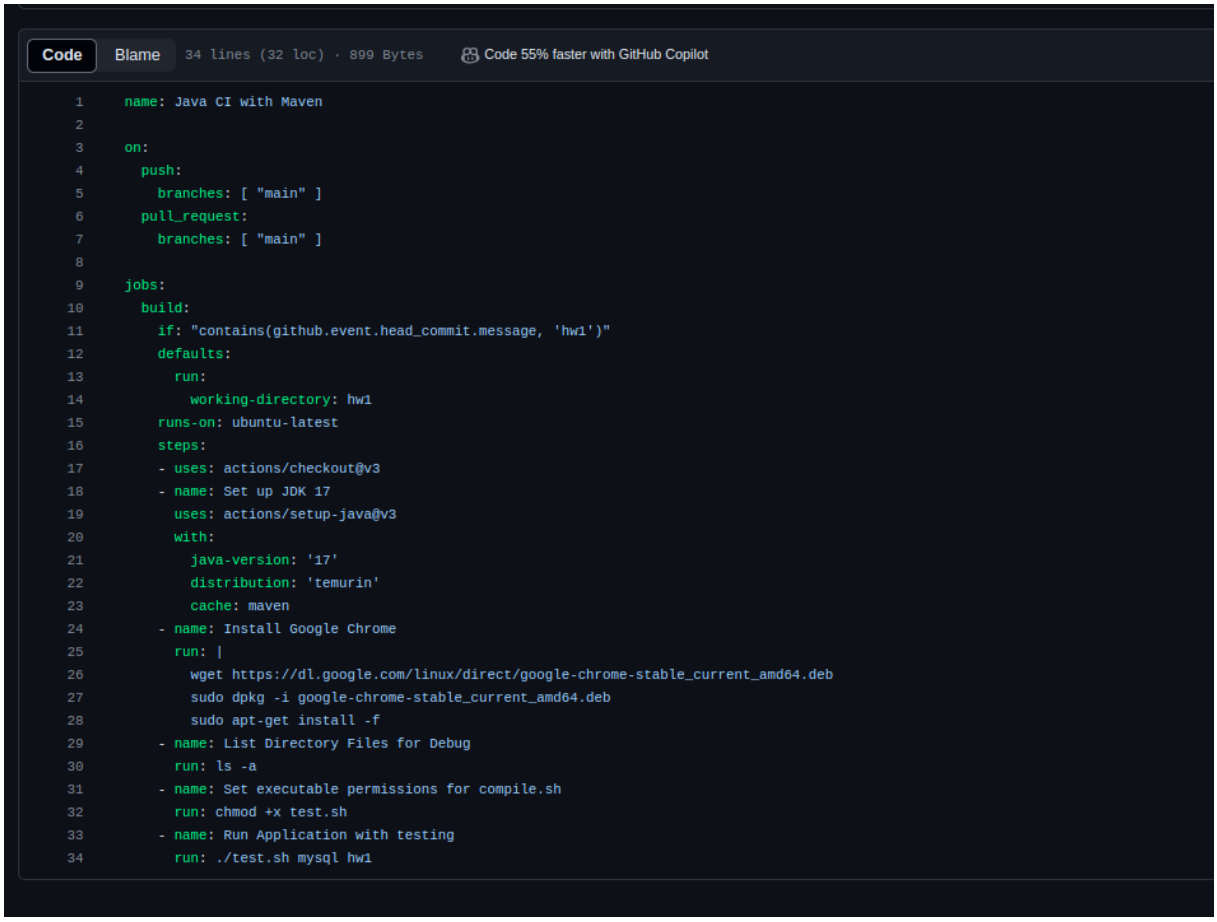


1.10 Continuous integration pipeline [optional]



Used Github Actions. Had some problems with all test succeding on the actions environment which didn't allow for me to use sonar cloud.






```yaml
name: Java CI with Maven

on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]

jobs:
  build:
    if: "contains(github.event.head_commit.message, 'hw1')"
    defaults:
      run:
        working-directory: hw1
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v3
    - name: Set up JDK 17
      uses: actions/setup-java@v3
      with:
        java-version: '17'
        distribution: 'temurin'
        cache: maven
    - name: Install Google Chrome
      run: |
        wget https://dl.google.com/linux/direct/google-chrome-stable_current_amd64.deb
        sudo dpkg -i google-chrome-stable_current_amd64.deb
        sudo apt-get install -f
    - name: List Directory Files for Debug
      run: ls -a
    - name: Set executable permissions for compile.sh
      run: chmod +x test.sh
    - name: Run Application with testing
      run: ./test.sh mysql hw1
```

References & resources

Project resources

Resource:	URL/location:

Git repository	https://github.com/uTigas/TQS_108546
Video demo	https://youtu.be/aijHXUw5FDg
QA dashboard (online)	[**optional**; if you have a quality dashboard available online (e.g.: sonarcloud), place the URL here]
CI/CD pipeline	https://github.com/uTigas/TQS_108546/actions
Deployment ready to use	[**optional**; if you have the solution deployed and running in a server, place the URL here]

Reference materials