# HW1: Mid-term assignment report

Miguel Aido Miragaia [108317], 09/04/2024

# 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 application serves as a platform for users to search for available bus connections, make reservations and view reservations by their ID, given after making the purchase.

## 1.2 Current limitations

**Code Coverage: 64% ->** because of the low implementation of test for the BusConnectionController and for the model Reservation there is a code coverage % below the expected.
**Endpoints ->** several endpoints are created, but not implemented at the frontend (deleteBusConnectionbyId , updateBusConnectionbyId, deleteReservationbyId, updateReservationbyId).
**Selenium ->** lack of tests with Selenium

# 2 Product specification

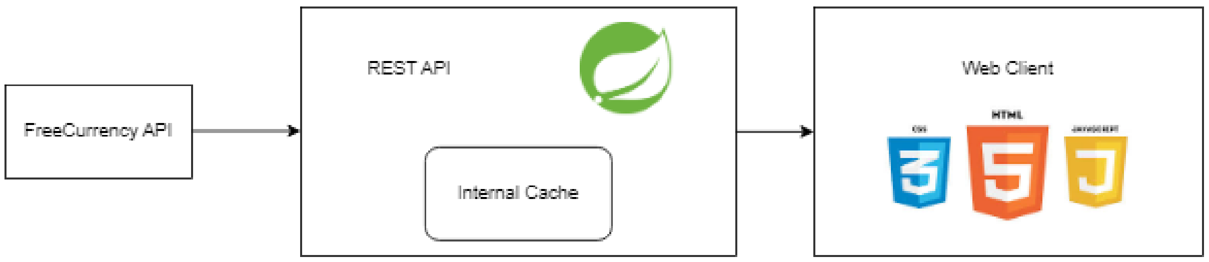
## 2.1 Functional scope and supported interactions

The application is designed to facilitate the booking of bus tickets for users who wish to travel between cities.
The **actors** are these customers, that will interact with the app by searching for Bus Connections, making the Reservation of a Bus Ticket after selecting the Bus Connection, and finally they can search for past made Reservations.
**Usage Scenario:**

- **Search for Bus Connections:** Customers can search for available bus connections based on origin, destination, and departure date. They can view a list of available trips that match their criteria.
- **Make Reservation:** After selecting the preferred Bus Connection based on the criteria presented, customers can proceed to make a reservation by providing their personal information, such as name, contact details, and payment information.
- **Manage Reservations:** After making a reservation, customers are presented with a goodbye message that includes the ID of the Reservation they just made. Customers can search for the reservation ID and be presented with the Reservation and Bus Connection information.

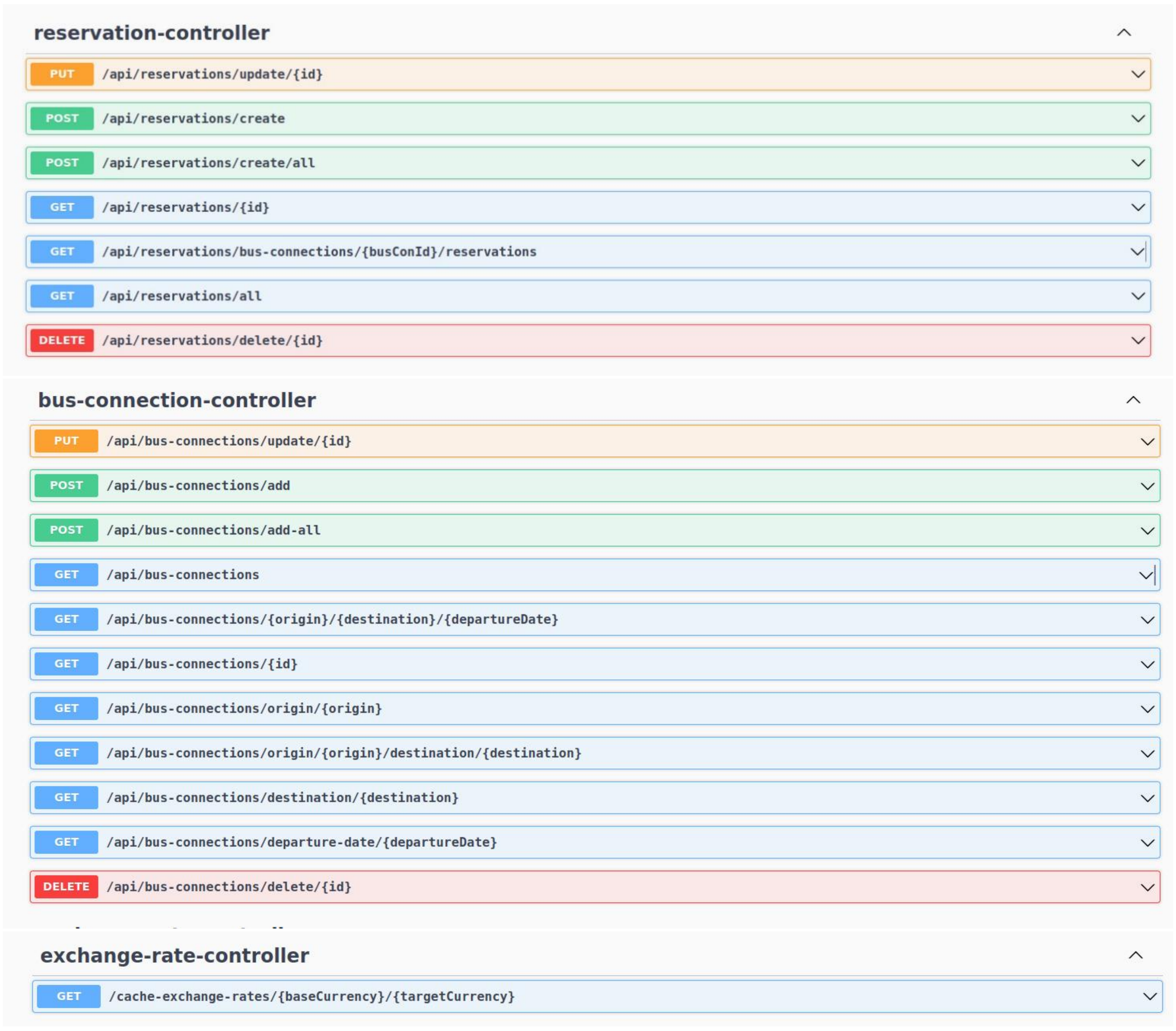
## 2.2 System architecture

The application follows a **client-server architecture**, where the client-side interacts with server-side to perform various actions.
It makes use of **“freecurrencyapi”** to get the exchange rates of two different currencies given (the base currency and the target currency). These values are stored in **cache** as they are fetched from the API, so later it will not exist the need of fetching the data from the API again.
The web client was built with **HTML/CSS/JS**
The backend was built using **Java with the Spring Boot framework**, which provides features such as RESTful web services and dependency injection.
For the architecture of the **REST API**, it was used the standard approach of Controller – Service – Repository pattern. The controller handles request mapping, and invokes the service methods, that deal with data fetching and cache management, Repository provides an abstraction layer for data access and manipulation (CRUD operations).



## 2.3 API for developers

The REST API was documented with Swagger for Maven. It can be accessed at http://localhost:8080/swagger-ui/index.html



**reservation-controller**

PUT /api/reservations/update/{id}
POST /api/reservations/create
POST /api/reservations/create/all
GET /api/reservations/{id}
GET /api/reservations/bus-connections/{busConId}/reservations
GET /api/reservations/all
DELETE /api/reservations/delete/{id}

**bus-connection-controller**

PUT /api/bus-connections/update/{id}
POST /api/bus-connections/add
POST /api/bus-connections/add-all
GET /api/bus-connections
GET /api/bus-connections/{origin}/{destination}/{departureDate}
GET /api/bus-connections/{id}
GET /api/bus-connections/origin/{origin}
GET /api/bus-connections/origin/{origin}/destination/{destination}
GET /api/bus-connections/destination/{destination}
GET /api/bus-connections/departure-date/{departureDate}
DELETE /api/bus-connections/delete/{id}

**exchange-rate-controller**

GET /cache-exchange-rates/{baseCurrency}/{targetCurrency}

# 3 Quality assurance

## 3.1 Overall strategy for testing

Firstly, started developing the unit tests for Cache, API Response and Data Initializer.
The rest of the test development was started with the Controller, then Services and finally the Repository.

## 3.2 Unit and integration testing

In my testing strategy, I prioritize the development of comprehensive unit and integration tests to ensure the reliability and correctness of my application components.

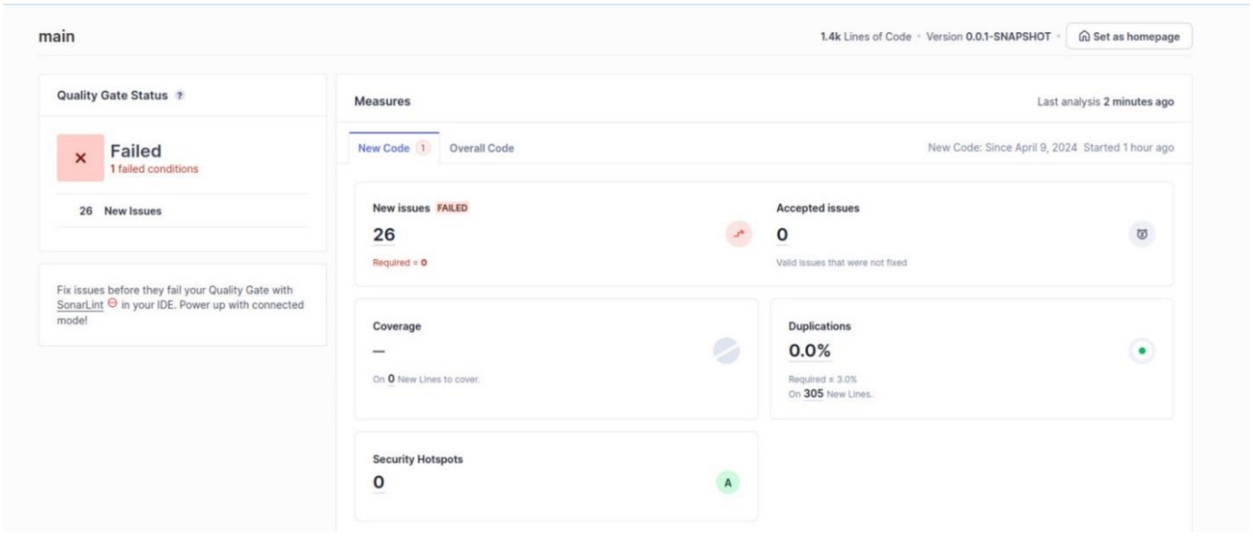
Unit Tests for Cache, API Response handling and Data Initializer
Integration Tests for Controllers, Services and Repositories ensuring that the system functions correctly as a whole.

## 3.3 Functional testing

Selenium was the selected framework to test the Web Client but there was not enough time to complete the tests.

## 3.4 Code quality analysis

SonarQube was deployed locally and used for code analysis, initially the was not passing the Quality Gate but after some refactoring to the code it was possible to.



main

1.4k Lines of Code · Version 0.0.1-SNAPSHOT · Set as homepage

Quality Gate Status

Failed
1 failed conditions

26 New Issues

Fix issues before they fail your Quality Gate with SonarLint in your IDE. Power up with connected mode!

Measures

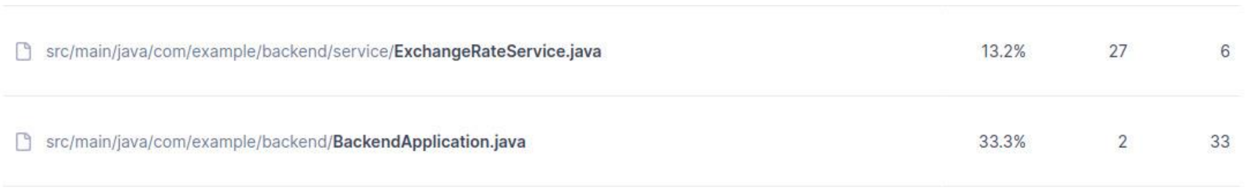
Last analysis 2 minutes ago

New Code 1 | Overall Code

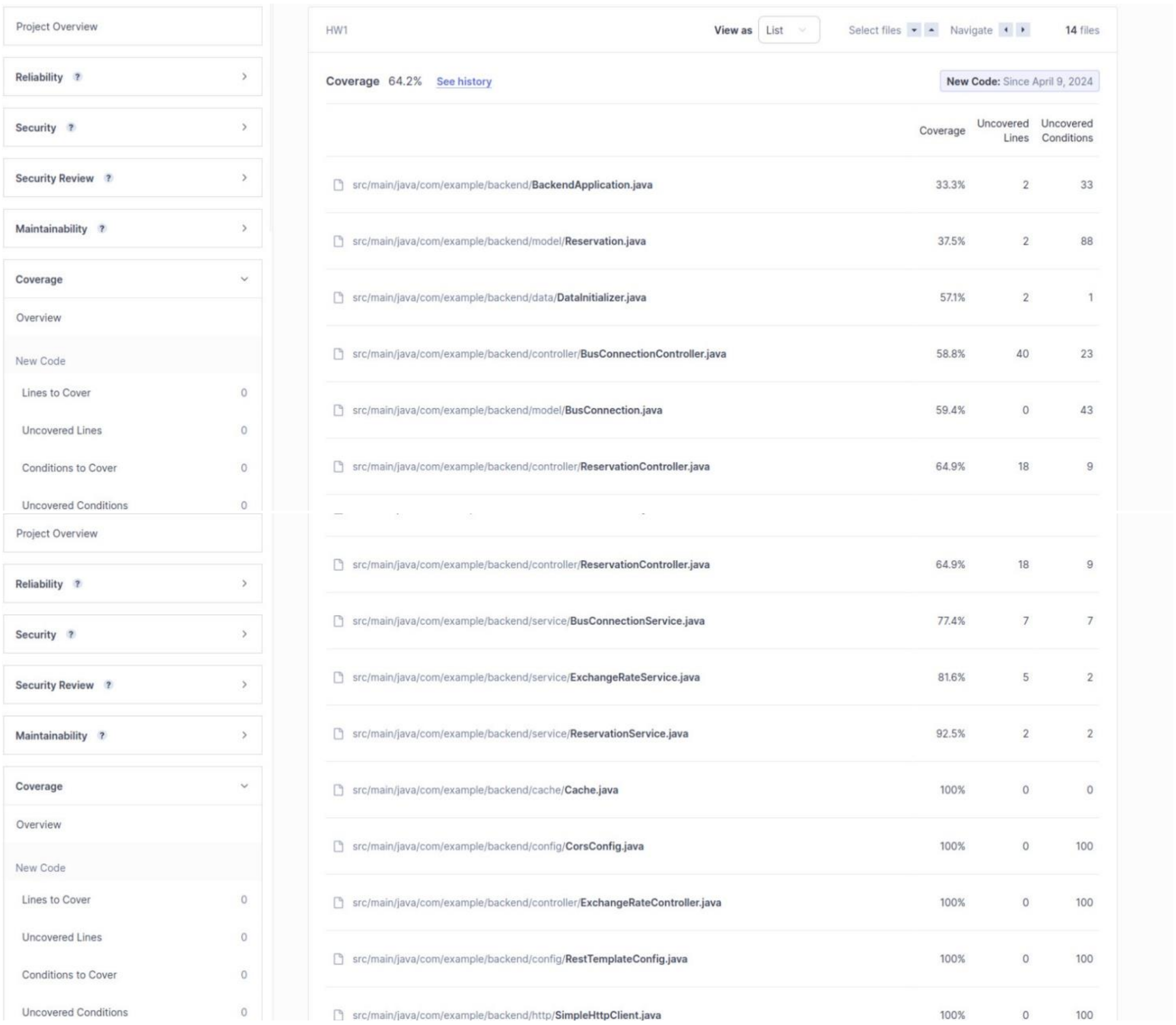
New Code: Since April 9, 2024 Started 1 hour ago

| | |
|---|---|
| New issues FAILED<br>26<br>Required = 0 | Accepted issues<br>0<br>Valid issues that were not fixed |
| Coverage<br>—<br>On 0 New Lines to cover. | Duplications<br>0.0%<br>Required ≤ 3.0%<br>On 305 New Lines. |
| Security Hotspots<br>0<br>A | |

By the analyses I was able to identify 2 security hotspots (API key at code, and Cors Config), due to time constraints I was not able to deal with them.
Code Coverage was very useful to keep the development of the tests and identify the main components that lacked tests. As you can see by the increase of % for the ExchangeRateService Component.



| | | | |
|---|---|---|---|
| src/main/java/com/example/backend/service/ExchangeRateService.java | 13.2% | 27 | 6 |
| src/main/java/com/example/backend/BackendApplication.java | 33.3% | 2 | 33 |

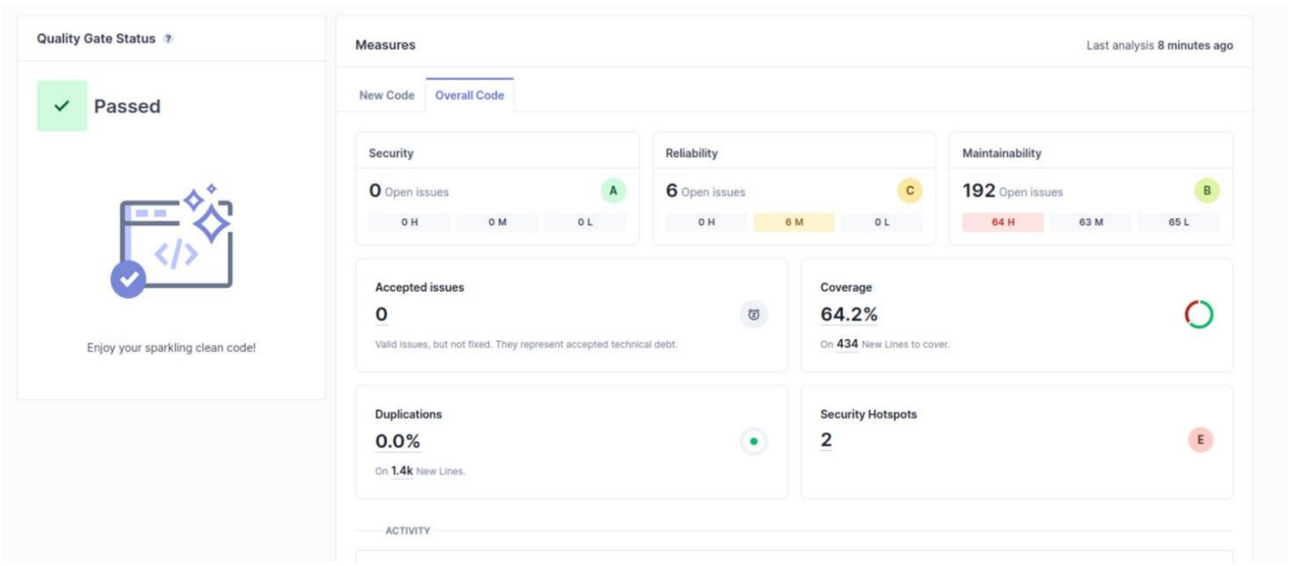


HW1 View as List Select files Navigate 14 files

Coverage 64.2% See history New Code: Since April 9, 2024

| | Coverage | Uncovered Lines | Uncovered Conditions |
|---|---|---|---|
| src/main/java/com/example/backend/BackendApplication.java | 33.3% | 2 | 33 |
| src/main/java/com/example/backend/model/Reservation.java | 37.5% | 2 | 88 |
| src/main/java/com/example/backend/data/DataInitializer.java | 57.1% | 2 | 1 |
| src/main/java/com/example/backend/controller/BusConnectionController.java | 58.8% | 40 | 23 |
| src/main/java/com/example/backend/model/BusConnection.java | 59.4% | 0 | 43 |
| src/main/java/com/example/backend/controller/ReservationController.java | 64.9% | 18 | 9 |
| src/main/java/com/example/backend/service/BusConnectionService.java | 77.4% | 7 | 7 |
| src/main/java/com/example/backend/service/ExchangeRateService.java | 81.6% | 5 | 2 |
| src/main/java/com/example/backend/service/ReservationService.java | 92.5% | 2 | 2 |
| src/main/java/com/example/backend/cache/Cache.java | 100% | 0 | 0 |
| src/main/java/com/example/backend/config/CorsConfig.java | 100% | 0 | 100 |
| src/main/java/com/example/backend/controller/ExchangeRateController.java | 100% | 0 | 100 |
| src/main/java/com/example/backend/config/RestTemplateConfig.java | 100% | 0 | 100 |
| src/main/java/com/example/backend/http/SimpleHttpClient.java | 100% | 0 | 100 |

This is the final Dashboard:



Security: 0 Open issues — A — 0 H, 0 M, 0 L

Reliability: 6 Open issues — C — 0 H, 6 M, 0 L

Maintainability: 192 Open issues — B — 64 H, 63 M, 65 L

Accepted issues: 0 — Valid issues, but not fixed. They represent accepted technical debt.

Coverage: 64.2% — On 434 New Lines to cover.

Duplications: 0.0% — On 1.4k New Lines.

Security Hotspots: 2 — E

### 3.5 Continuous integration pipeline [optional]

Not done

## 4 References & resources

**Project resources**

| **Resource:** | **URL/location:** |
|---|---|
| Git repository | https://github.com/Miragaia/TQS_108317 |
| Video demo | https://youtu.be/848-KVE-34Y |
| QA dashboard (online) | Local |
| CI/CD pipeline | Not done |
| Deployment ready to use | Not done |

**Reference materials**

Currency API - https://freecurrencyapi.com/