HW1: Mid-term assignment report

José Pedro Santos Mendes [107188], v2024-04-09

1 Introduction **1**
1.1 Overview of the work 1
1.2 Current limitations 1

2 Product specification **2**
2.1 Functional scope and supported interactions 2
2.2 System architecture 2
2.3 API for developers 3

3 Quality assurance **4**
3.1 Overall strategy for testing 4
3.2 Unit and integration testing 4
3.3 Functional testing 6
3.4 Code quality analysis 7
3.5 Continuous integration pipeline [optional] 8

4 References & resources **9**

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 is called **TicketTrack Bus** and it was developed with the purpose of letting users easily buy bus tickets between some available cities in Portugal. Users can conveniently select their preferred currency for payment. Additionally, the app provides a feature for users to review their purchase history, allowing them to easily track previously acquired tickets.

1.2 Current limitations

- The application was developed locally, without deployment in a server. All the components that take part are containerized and the application can easily be launched by using the *docker compose* file.

2 Product specification

2.1 Functional scope and supported interactions

The developed application is intended for individuals seeking a streamlined way to purchase bus tickets between specific cities in Portugal. It also provides users with the ability to review their past ticket purchases.

Scenario	**Procedure**
Check the available bus trips between two cities	The user opens the application and introduces the origin and destination cities along with the pretended date for the Trip. Finally, the user clicks in the *Search* button and the available trips are shown.
Purchase a trip	In the trips page, the user selects the pretended trip and clicks the *Buy* button. Afterwards, the user can select the currency they want to purchase the trip. Finally, the user fills the form with his information and clicks the *Pay* button, receiving a token for the reservation.
Check previous purchased trips	In the reservations page, the user inserts his email along with the reservation token given in the moment of purchase and clicks the *Get Reservation* button. If the email and token are correct, the information about the reservation and trip will be displayed.

2.2 System architecture

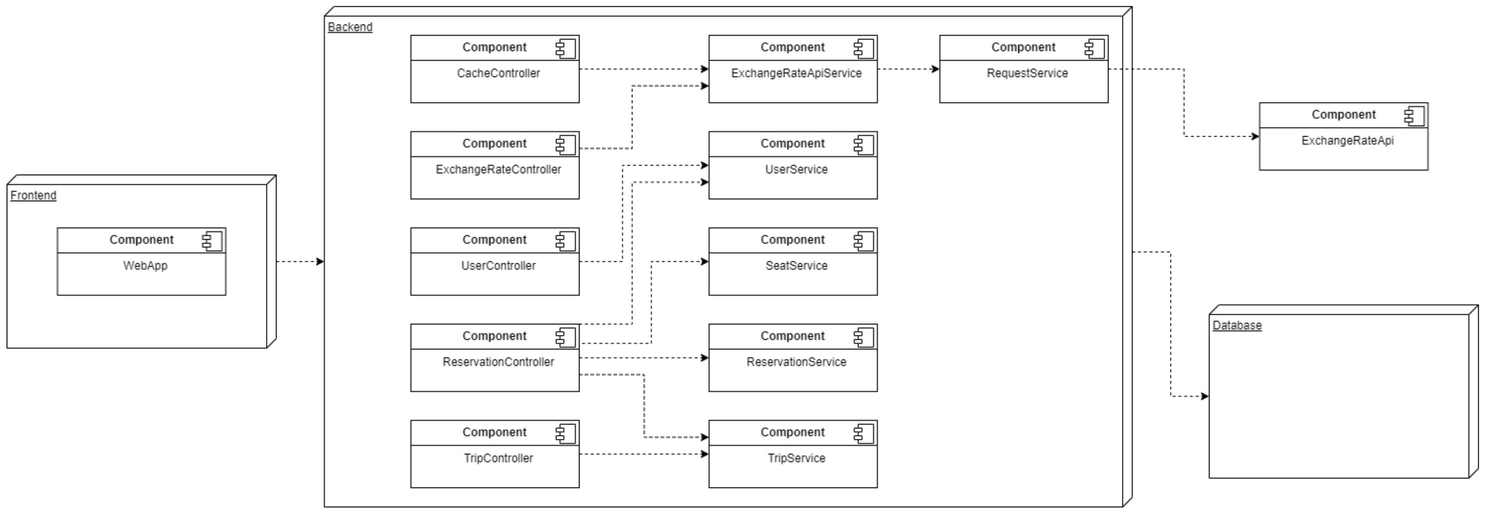


Figure 1 - System Architecture

The application was built using **Spring Boot** for the backend, allowing for the creation of a REST API. On the frontend, **React** and **Vite** were chosen along with **MaterialUI** for rapid development. **Axios** was used to simplify backend requests. For data persistence, **MySQL** was selected due to its seamless integration with Spring Boot, enabling straightforward communication between the application and the database. This tech stack ensured efficient development and reliable data management for the bus ticketing application.

2.3 API for developers

The API endpoint documentation is available in the endpoint ***/swagger-ui/index.html***, and it was achieved using Swagger.

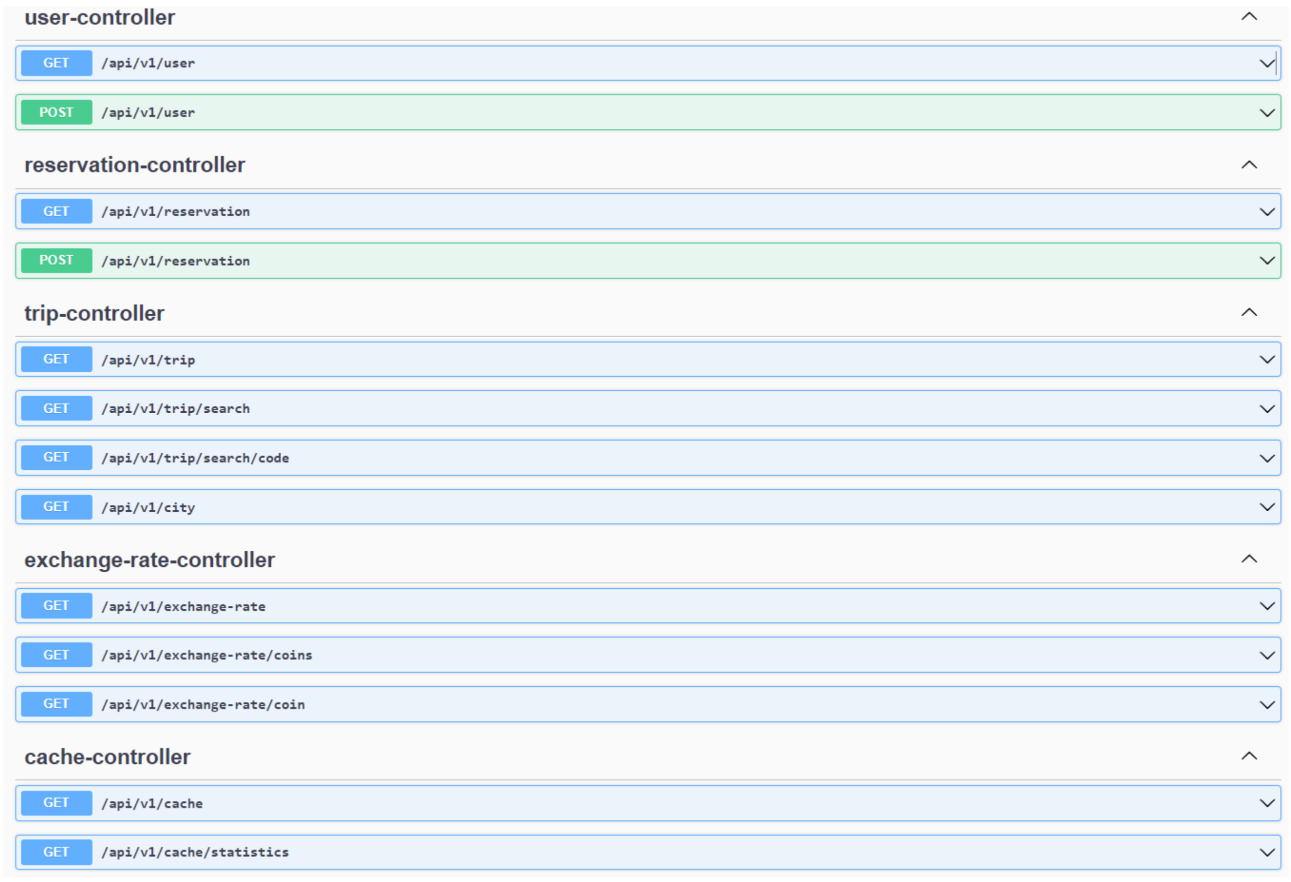

user-controller
GET /api/v1/user
POST /api/v1/user
reservation-controller
GET /api/v1/reservation
POST /api/v1/reservation
trip-controller
GET /api/v1/trip
GET /api/v1/trip/search
GET /api/v1/trip/search/code
GET /api/v1/city
exchange-rate-controller
GET /api/v1/exchange-rate
GET /api/v1/exchange-rate/coins
GET /api/v1/exchange-rate/coin
cache-controller
GET /api/v1/cache
GET /api/v1/cache/statistics

Figure 2 - Swagger API Documentation

- **GET *api/v1/user*** – Get a user by passing the email as a parameter;
- **POST *api/v1/user*** – Saves a user in the database;
- **GET *api/v1/reservation*** – Get a reservation by passing the reservation token as a parameter;
- **POST *api/v1/reservation*** – Saves a reservation in the database by passing the reservation token, date and user as parameters;
- **GET *api/v1/trip*** – Get all the trips available in the database;
- **GET *api/v1/trip/search*** – Get all the trips available in the database filtering them by origin and destination cities passed as parameters;
- **GET *api/trip/search/code*** – Get a trip by passing the trip code as a parameter;

- **GET** ***/api/v1/city*** – Get all the cities that have trips associated to them;
- **GET** ***/api/v1/exchange-rate*** – Get the exchange rate relative to the euro;
- **GET** ***/api/v1/exchange-rate/coins*** – Get all the coins that exist in the exchange rate;
- **GET** ***/api/v1/exchange-rate/coin*** – Get the total price from the original in euro to the wanted coin by passing the original price and the pretended coin;
- **GET** ***/api/v1/cache*** – Get the cached data that is being cached from the exchange rate API;
- **GET** ***/api/v1/cache/statistics*** – Get the statistics of the cached data, retrieving the hits, misses and puts of the cache;

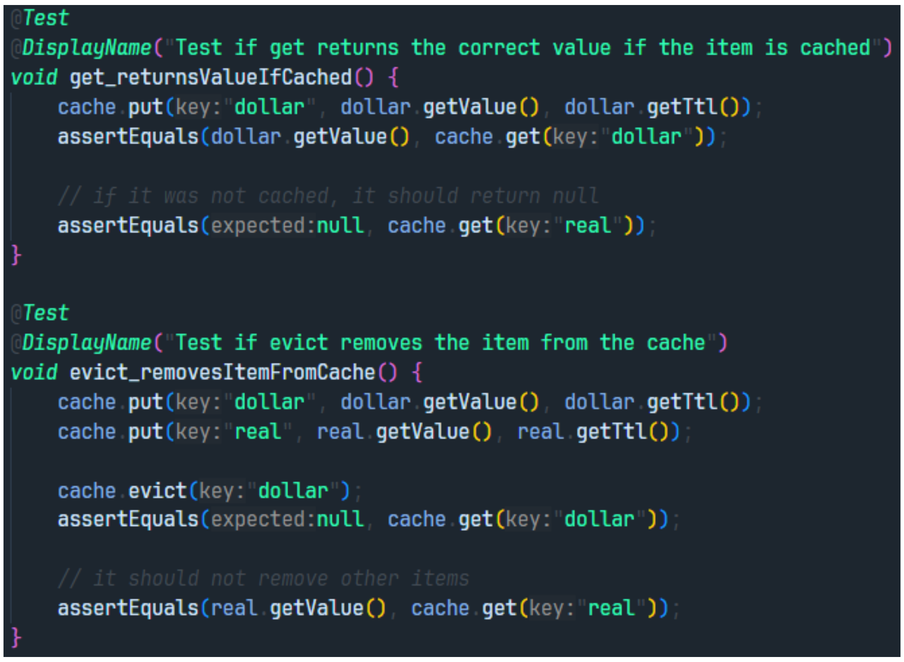
3 Quality assurance

3.1 Overall strategy for testing

In order to test the implementation and the functionalities of the application, an initial development strategy was Test-Driven Development (TDD), chosen for its effectiveness in ensuring software quality. However, a significant portion of the backend was developed before initiating the tests, with test development beginning after establishing the backend skeleton.
The tools used in the test development were presented and lectured in the course. Tools such as **JUnit5** for Unit Testing, **Mockito** and **SpringBoot MockMvc** for testing the services along with Integration Testing. For the frontend tests, **Selenium** was used, allowing for a Behavior-Driven Development (BDD).

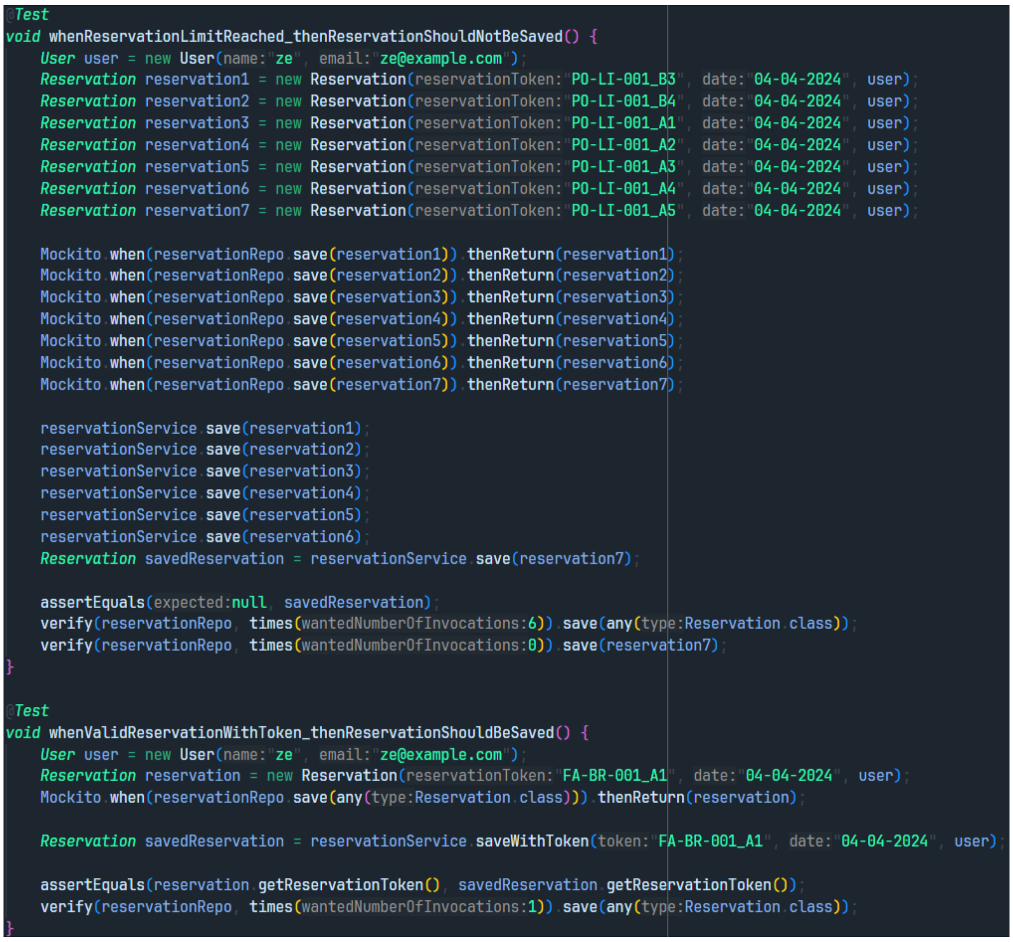
3.2 Unit and integration testing

Unit tests were mainly used to ensure that the behavior of the entities and cache was as expected, a great example is the *CacheUnitTest* test class.
Secondly, Service level tests with dependency isolation were implemented, allowing to decouple the services from the repository connected to the database or from the exchange rate API.
Finally, the integration tests were implemented to test the behavior of the controllers, and were achieved by using *SpringBoot MockMvc*. Each endpoint as a test associated with it.



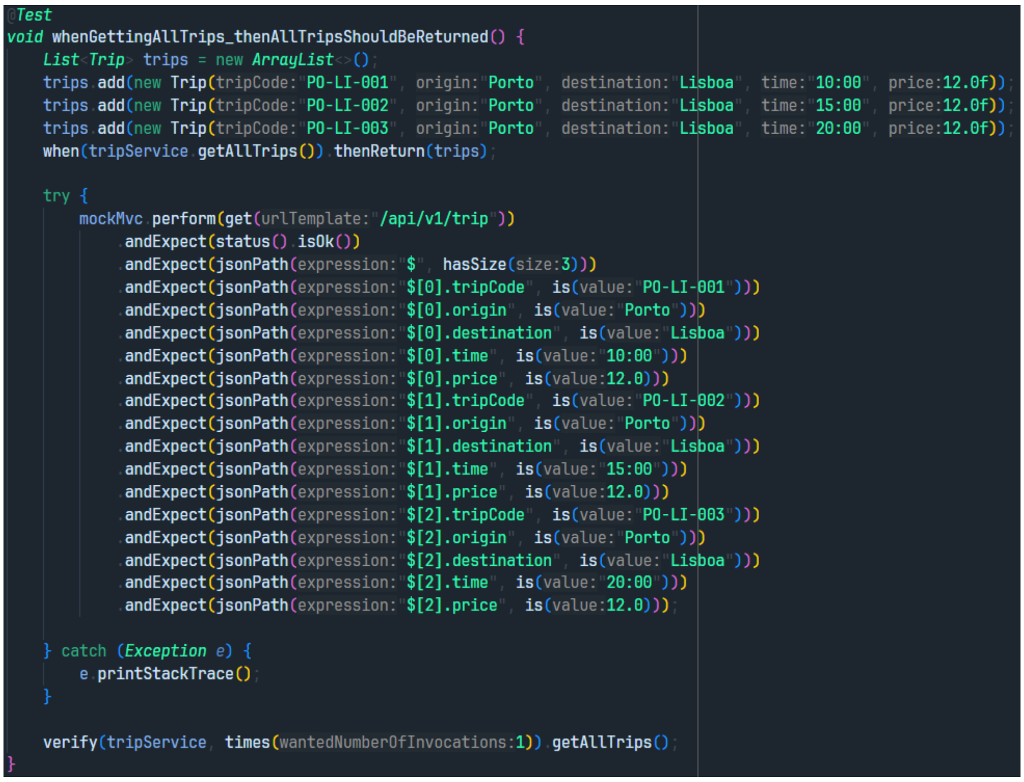
```java
@Test
@DisplayName("Test if get returns the correct value if the item is cached")
void get_returnsValueIfCached() {
    cache.put(key:"dollar", dollar.getValue(), dollar.getTtl());
    assertEquals(dollar.getValue(), cache.get(key:"dollar"));

    // if it was not cached, it should return null
    assertEquals(expected:null, cache.get(key:"real"));
}

@Test
@DisplayName("Test if evict removes the item from the cache")
void evict_removesItemFromCache() {
    cache.put(key:"dollar", dollar.getValue(), dollar.getTtl());
    cache.put(key:"real", real.getValue(), real.getTtl());

    cache.evict(key:"dollar");
    assertEquals(expected:null, cache.get(key:"dollar"));

    // it should not remove other items
    assertEquals(real.getValue(), cache.get(key:"real"));
}
```

Figure 3 - Cache Unit Test



```java
@Test
void whenReservationLimitReached_thenReservationShouldNotBeSaved() {
    User user = new User(name:"ze", email:"ze@example.com");
    Reservation reservation1 = new Reservation(reservationToken:"PO-LI-001_B3", date:"04-04-2024", user);
    Reservation reservation2 = new Reservation(reservationToken:"PO-LI-001_B4", date:"04-04-2024", user);
    Reservation reservation3 = new Reservation(reservationToken:"PO-LI-001_A1", date:"04-04-2024", user);
    Reservation reservation4 = new Reservation(reservationToken:"PO-LI-001_A2", date:"04-04-2024", user);
    Reservation reservation5 = new Reservation(reservationToken:"PO-LI-001_A3", date:"04-04-2024", user);
    Reservation reservation6 = new Reservation(reservationToken:"PO-LI-001_A4", date:"04-04-2024", user);
    Reservation reservation7 = new Reservation(reservationToken:"PO-LI-001_A5", date:"04-04-2024", user);

    Mockito.when(reservationRepo.save(reservation1)).thenReturn(reservation1);
    Mockito.when(reservationRepo.save(reservation2)).thenReturn(reservation2);
    Mockito.when(reservationRepo.save(reservation3)).thenReturn(reservation3);
    Mockito.when(reservationRepo.save(reservation4)).thenReturn(reservation4);
    Mockito.when(reservationRepo.save(reservation5)).thenReturn(reservation5);
    Mockito.when(reservationRepo.save(reservation6)).thenReturn(reservation6);
    Mockito.when(reservationRepo.save(reservation7)).thenReturn(reservation7);

    reservationService.save(reservation1);
    reservationService.save(reservation2);
    reservationService.save(reservation3);
    reservationService.save(reservation4);
    reservationService.save(reservation5);
    reservationService.save(reservation6);
    Reservation savedReservation = reservationService.save(reservation7);

    assertEquals(expected:null, savedReservation);
    verify(reservationRepo, times(wantedNumberOfInvocations:6)).save(any(type:Reservation.class));
    verify(reservationRepo, times(wantedNumberOfInvocations:0)).save(reservation7);
}

@Test
void whenValidReservationWithToken_thenReservationShouldBeSaved() {
    User user = new User(name:"ze", email:"ze@example.com");
    Reservation reservation = new Reservation(reservationToken:"FA-BR-001_A1", date:"04-04-2024", user);
    Mockito.when(reservationRepo.save(any(type:Reservation.class))).thenReturn(reservation);

    Reservation savedReservation = reservationService.saveWithToken(token:"FA-BR-001_A1", date:"04-04-2024", user);

    assertEquals(reservation.getReservationToken(), savedReservation.getReservationToken());
    verify(reservationRepo, times(wantedNumberOfInvocations:1)).save(any(type:Reservation.class));
}
```

Figure 4 - ReservationService Mock Test

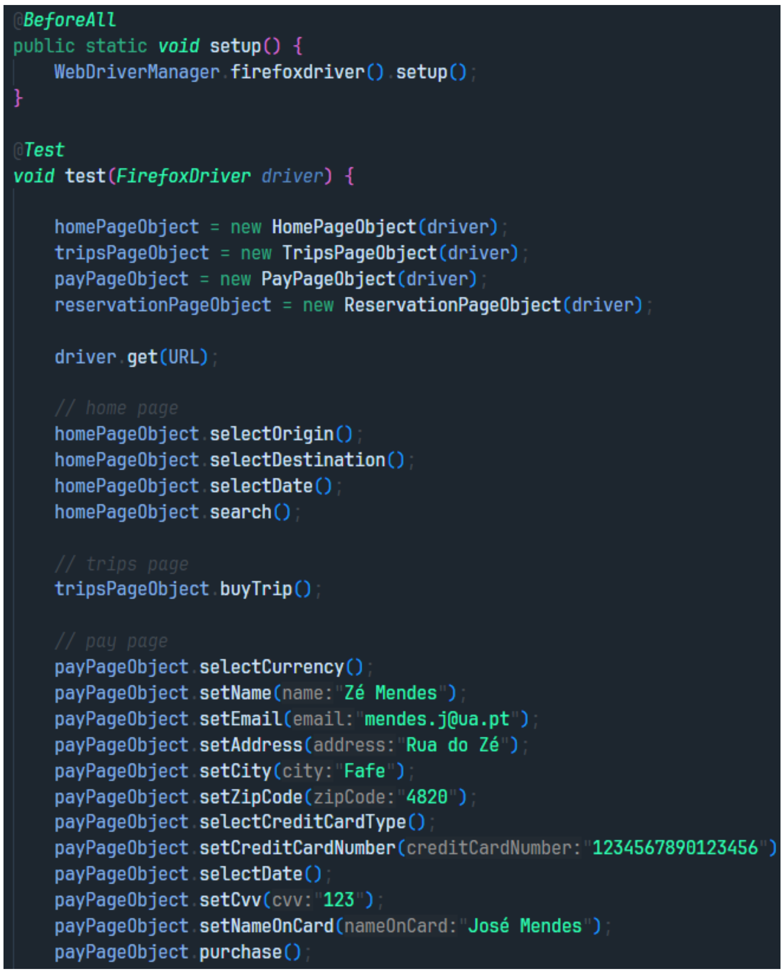


```java
@Test
void whenGettingAllTrips_thenAllTripsShouldBeReturned() {
    List<Trip> trips = new ArrayList<>();
    trips.add(new Trip(tripCode: "PO-LI-001", origin: "Porto", destination: "Lisboa", time: "10:00", price: 12.0f));
    trips.add(new Trip(tripCode: "PO-LI-002", origin: "Porto", destination: "Lisboa", time: "15:00", price: 12.0f));
    trips.add(new Trip(tripCode: "PO-LI-003", origin: "Porto", destination: "Lisboa", time: "20:00", price: 12.0f));
    when(tripService.getAllTrips()).thenReturn(trips);

    try {
        mockMvc.perform(get(urlTemplate: "/api/v1/trip"))
                .andExpect(status().isOk())
                .andExpect(jsonPath(expression: "$", hasSize(size: 3)))
                .andExpect(jsonPath(expression: "$[0].tripCode", is(value: "PO-LI-001")))
                .andExpect(jsonPath(expression: "$[0].origin", is(value: "Porto")))
                .andExpect(jsonPath(expression: "$[0].destination", is(value: "Lisboa")))
                .andExpect(jsonPath(expression: "$[0].time", is(value: "10:00")))
                .andExpect(jsonPath(expression: "$[0].price", is(value: 12.0)))
                .andExpect(jsonPath(expression: "$[1].tripCode", is(value: "PO-LI-002")))
                .andExpect(jsonPath(expression: "$[1].origin", is(value: "Porto")))
                .andExpect(jsonPath(expression: "$[1].destination", is(value: "Lisboa")))
                .andExpect(jsonPath(expression: "$[1].time", is(value: "15:00")))
                .andExpect(jsonPath(expression: "$[1].price", is(value: 12.0)))
                .andExpect(jsonPath(expression: "$[2].tripCode", is(value: "PO-LI-003")))
                .andExpect(jsonPath(expression: "$[2].origin", is(value: "Porto")))
                .andExpect(jsonPath(expression: "$[2].destination", is(value: "Lisboa")))
                .andExpect(jsonPath(expression: "$[2].time", is(value: "20:00")))
                .andExpect(jsonPath(expression: "$[2].price", is(value: 12.0)));

    } catch (Exception e) {
        e.printStackTrace();
    }

    verify(tripService, times(wantedNumberOfInvocations: 1)).getAllTrips();
}
```

Figure 5 - TripController Integration Test

3.3 Functional testing

For the functional tests, a Behavior-Driven Development (BDD) was achieved by using the **Selenium WebDriver**, allowing to test the product itself. There were some issues faced with the Web Drivers which were eventually mitigated.
To develop this, a Page Object Pattern was utilized, organizing the test in a better way.

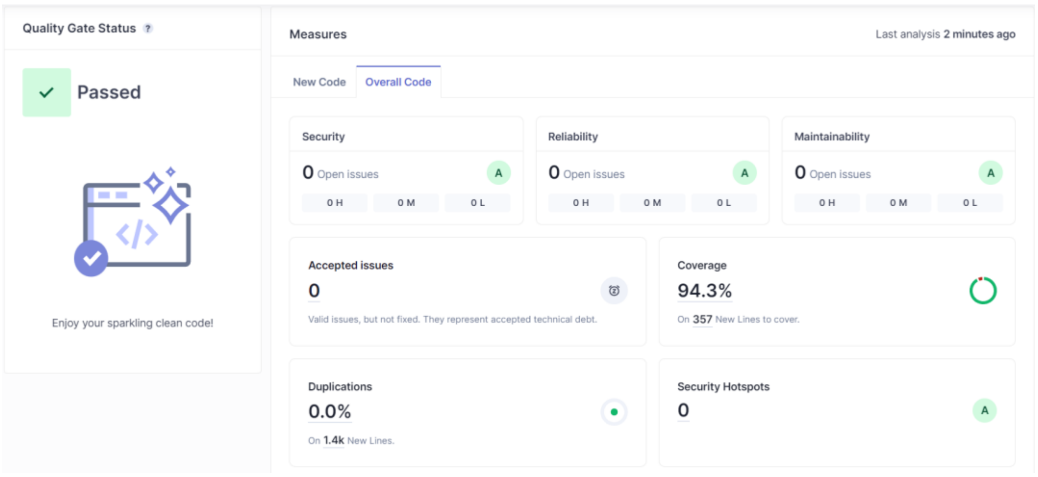


```
@BeforeAll
public static void setup() {
    WebDriverManager.firefoxdriver().setup();
}

@Test
void test(FirefoxDriver driver) {

    homePageObject = new HomePageObject(driver);
    tripsPageObject = new TripsPageObject(driver);
    payPageObject = new PayPageObject(driver);
    reservationPageObject = new ReservationPageObject(driver);

    driver.get(URL);

    // home page
    homePageObject.selectOrigin();
    homePageObject.selectDestination();
    homePageObject.selectDate();
    homePageObject.search();

    // trips page
    tripsPageObject.buyTrip();

    // pay page
    payPageObject.selectCurrency();
    payPageObject.setName(name: "Zé Mendes");
    payPageObject.setEmail(email: "mendes.j@ua.pt");
    payPageObject.setAddress(address: "Rua do Zé");
    payPageObject.setCity(city: "Fafe");
    payPageObject.setZipCode(zipCode: "4820");
    payPageObject.selectCreditCardType();
    payPageObject.setCreditCardNumber(creditCardNumber: "1234567890123456")
    payPageObject.selectDate();
    payPageObject.setCvv(cvv: "123");
    payPageObject.setNameOnCard(nameOnCard: "José Mendes");
    payPageObject.purchase();
```

Figure 6 - Selenium Test

3.4 Code quality analysis

For the code quality analysis, a *docker* container of **SonarQube** was used in order to inspect the code for Code Smells, Security Hotspots and Percentage of Code Coverage.



Quality Gate Status

Passed

Enjoy your sparkling clean code!

Measures

Last analysis 2 minutes ago

New Code | Overall Code

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

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

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

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

Coverage: 94.3% — On 357 New Lines to cover.

Duplications: 0.0% — On 1.4k New Lines.

Security Hotspots: 0 — A

Figure 7 - SonarQube Dashboard

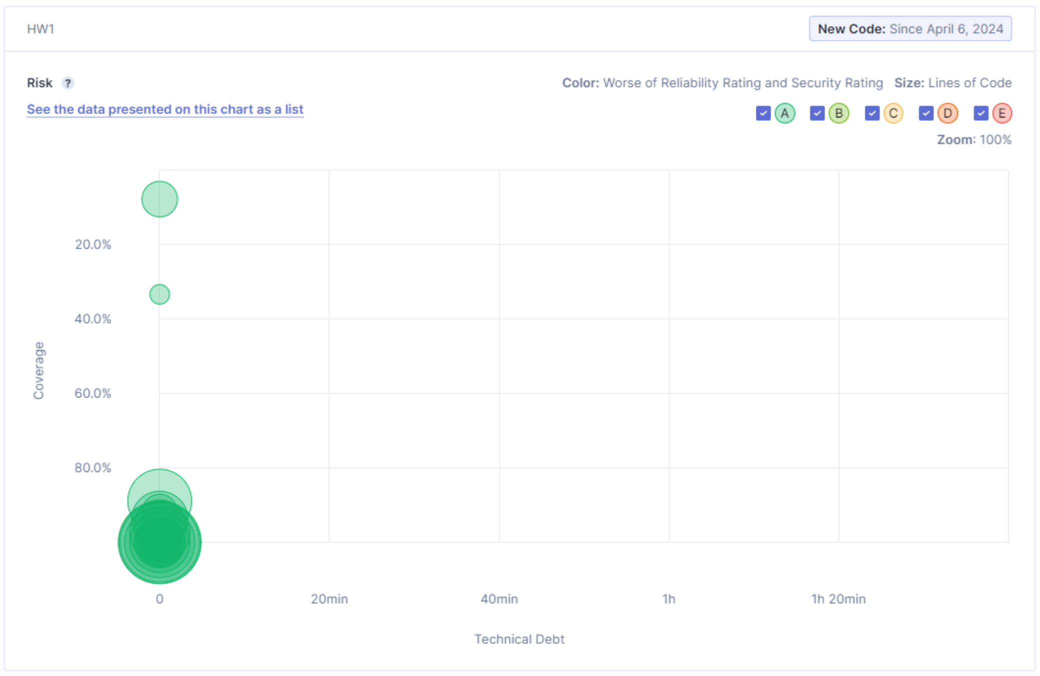
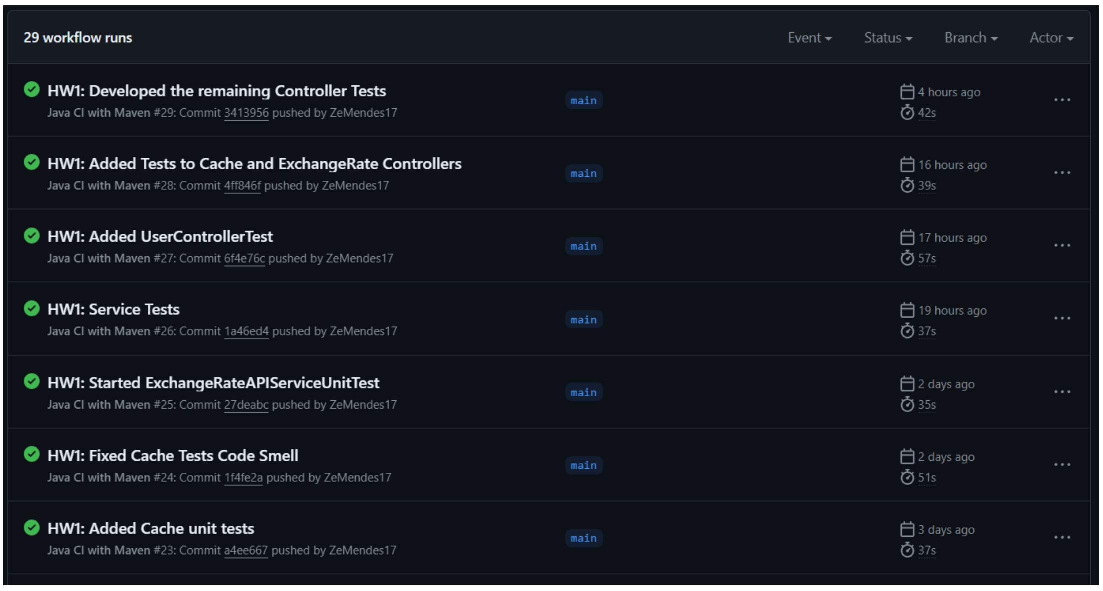


Figure 8 - SonarQube Technical Debt

By analyzing the Dashboard we can see that there aren't any Issues or Security Hotspots, no Duplications and the tests achieve a total of 94.3% coverage.

3.5 Continuous integration pipeline [optional]

A simple Continuous Integration (CI) pipeline was implemented utilizing **Gihub Actions**, and it runs all the tests with the exception of the Integration Tests. Whenever new code is pushed to the repository, tests run, verifying if every test is successful.



29 workflow runs — Event, Status, Branch, Actor

Workflow run	Branch	When	Duration
✓ HW1: Developed the remaining Controller Tests Java CI with Maven #29: Commit 3413956 pushed by ZeMendes17	main	4 hours ago	42s
✓ HW1: Added Tests to Cache and ExchangeRate Controllers Java CI with Maven #28: Commit 4ff846f pushed by ZeMendes17	main	16 hours ago	39s
✓ HW1: Added UserControllerTest Java CI with Maven #27: Commit 6f4e76c pushed by ZeMendes17	main	17 hours ago	57s
✓ HW1: Service Tests Java CI with Maven #26: Commit 1a46ed4 pushed by ZeMendes17	main	19 hours ago	37s
✓ HW1: Started ExchangeRateAPIServiceUnitTest Java CI with Maven #25: Commit 27deabc pushed by ZeMendes17	main	2 days ago	35s
✓ HW1: Fixed Cache Tests Code Smell Java CI with Maven #24: Commit 1f4fe2a pushed by ZeMendes17	main	2 days ago	51s
✓ HW1: Added Cache unit tests Java CI with Maven #23: Commit a4ee667 pushed by ZeMendes17	main	3 days ago	37s

Figure 9 - Github Action CI pipeline

4 References & resources

Project resources

Resource:	URL/location:
Git repository	https://github.com/ZeMendes17/TQS_107188
Video demo	https://github.com/ZeMendes17/TQS_107188/blob/main/HW1/media/demo.mp4
QA dashboard (online)	Not Applicable
CI/CD pipeline	https://github.com/ZeMendes17/TQS_107188/actions
Deployment ready to use	Not Applicable

Reference materials

- External Exchange Rates API - https://app.exchangerate-api.com
- Fady Kuzman, “Medium”, 29/01/2023 [Online]. Available: https://medium.com/@f.s.a.kuzman/using-swagger-3-in-spring-boot-3-c11a483ea6dc [Acedido a 08/04/2024]
- Eric Cabrel TIOGO, “Tericcabrel”, 17/07/2023 [Online]. Available: https://blog.tericcabrel.com/springboot-github-actions-ci-cd/ [Acedido a 03/04/2024]