HW1: Mid-term assignment report

Maria Rafaela Alves Abrunhosa [107658], v2024-04-09

1 Introduction **1**
1.1 Overview of the work 1
1.2 Current limitations 2
2 Product specification **2**
2.1 Functional scope and supported interactions 2
2.2 System architecture 3
2.3 API for developers 4
3 Quality assurance **5**
3.1 Overall strategy for testing 5
3.2 Unit and integration testing 5
3.3 Functional testing 8
3.4 Code quality analysis 11
4 References & resources **11**

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.

This is a full-stack web application developed in Spring Boot with access and connection to an external api to receive the bus trips. The external api used is the flixbus api, provided by the Rapid API Hub.
As it was asked for, this application allows a user to search for a bus trip, book it and make a reservation with his data.

The purpose of this application is to develop a small platform with an API(REST) to test it and see how much coverage do we have overall.
This homework is important for us to understand the positive impact of automated testing, for our applications to work how they should.

1.2 Current limitations

During the development of this application, some obstacles appeared which translated into some limitations for a complete development of it.

The user can make a reservation for a bus trip introducing his personal info however, even with the reservation associated with the booked trip, the reservation is not saved anywhere. The problem is that there is no database implemented due to some problems with time, errors and docker. The expected was to implement a database that would save the reservations and maybe some trips for the external api flixbus. The access to the database data would be implemented in the repository.

There is also no cache implementation as well due to lack of time.

About the tests, some tests needed to be commented on, for example the cucumber test with the chrome driver, because it is giving dependency problems.

The selenium tests generated by the selenium IDE were not being detected by the VSCode but they are implemented. The cucumber test was implemented in addition to the selenium frontend test.

One of the limitations was also the number of free requests per month to the external api.

2 Product specification

2.1 Functional scope and supported interactions

This application is useful to book a bus trip between two cities and make a reservation. The first web page provides a small form to search by a bus trip choosing the departure city, the arrival city, the date of the trip and the currency of the price.

The main actors interacting with this web application are all the users that want and need to search for a bus trip and book it.

User Stories:

- As a traveling person, I want to search for a bus trip so that I can go from one city to another.
- As a traveling person, I want to choose a bus trip so that I can book the trip and make a reservation in my name and my personal info.

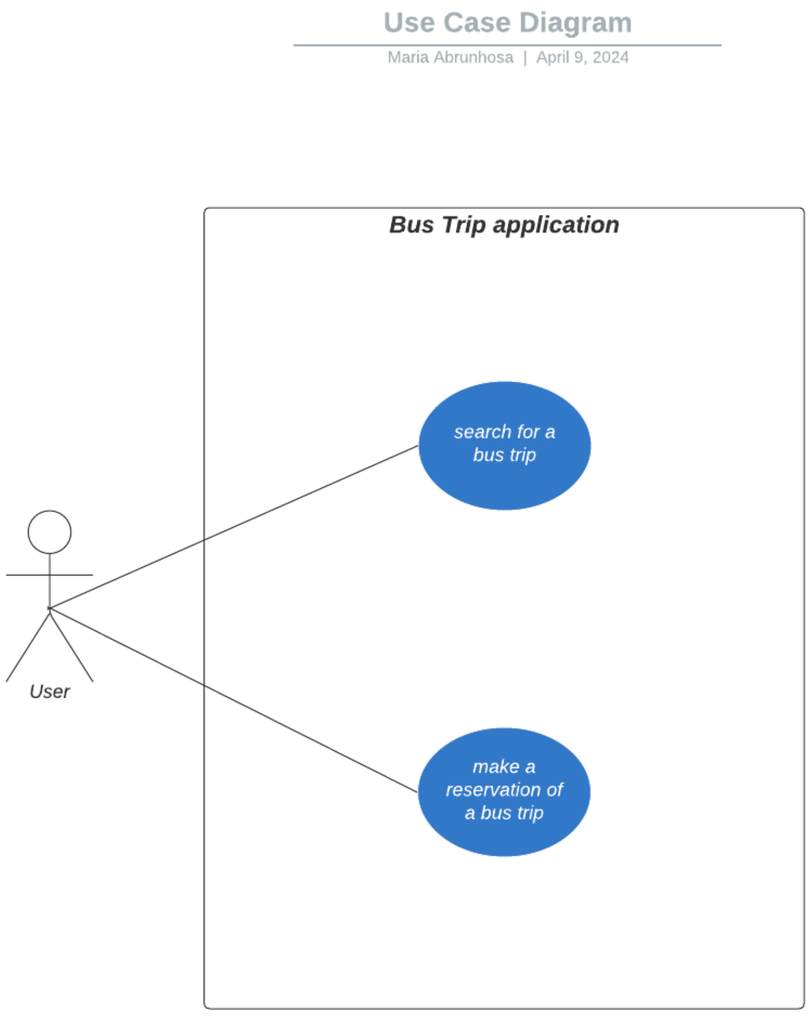

Use Case Diagram
Maria Abrunhosa | April 9, 2024
Bus Trip application
search for a bus trip
User
make a reservation of a bus trip

Image 1 - Bus Trip Application Use Case Diagram

2.2 System architecture

The system architecture can be divided by three main domains, the frontend, the backend and an external API.

The frontend was developed using HTML, CSS for styling, Javascript and AJAX to fetch the api data from the controller.

The backend was implemented with spring boot. The spring boot and its services deal with the clients requests, the requests to the external api flixbus and it was supposed to deal with the database and store the reservations.
The web application makes the requests to the REST API so it can retrieve the bus trips data and then, the REST API makes the request to the external api searching for bus trips.
The external api was provided by the Rapid API Hub.

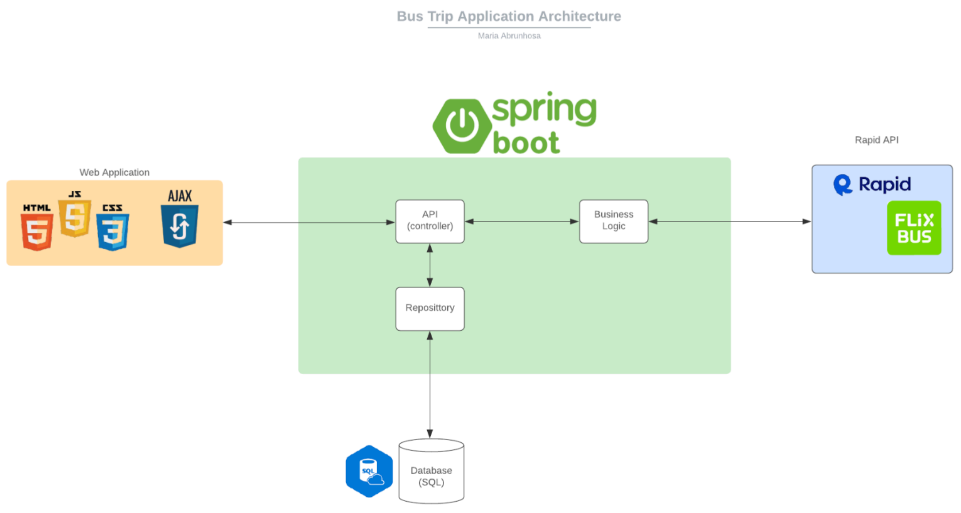


Image 2 - Bus Trip Application Architecture

2.3 API for developers

The REST API offers two main endpoints because the cache was not implemented.
The first one is the trips endpoint (api/trips?{parameters}) which returns a list of trips between two cities. The parameters are a from id, a to id, a date, the number of passengers and an optional parameter, the currency.
The second endpoint is the autocomplete endpoint (api/autocomplete?query=city) which returns all the information about a city including its id. The parameter is the name of the city we want to access the information.
The second endpoint is needed because the search in the index page is made with the name of the city or station and the fetch to the api to return the trips list just works with the id of the city or station.

Endpoints of BusTripController class:

Endpoint: /api/trips?from_id={from_id}&to_id={to_id}&date={date}&adult={adult} (optional &currency={currency})

Method: GET
Explanation: Return a list of bus trips between two cities with four required parameters (from_id, to_id, date and adult) and one optional (currency).

Endpoint: /api/trips?query={cityName}
Method: GET
Explanation: Returns all the information about the city, including coordinates, id and others.

3 Quality assurance

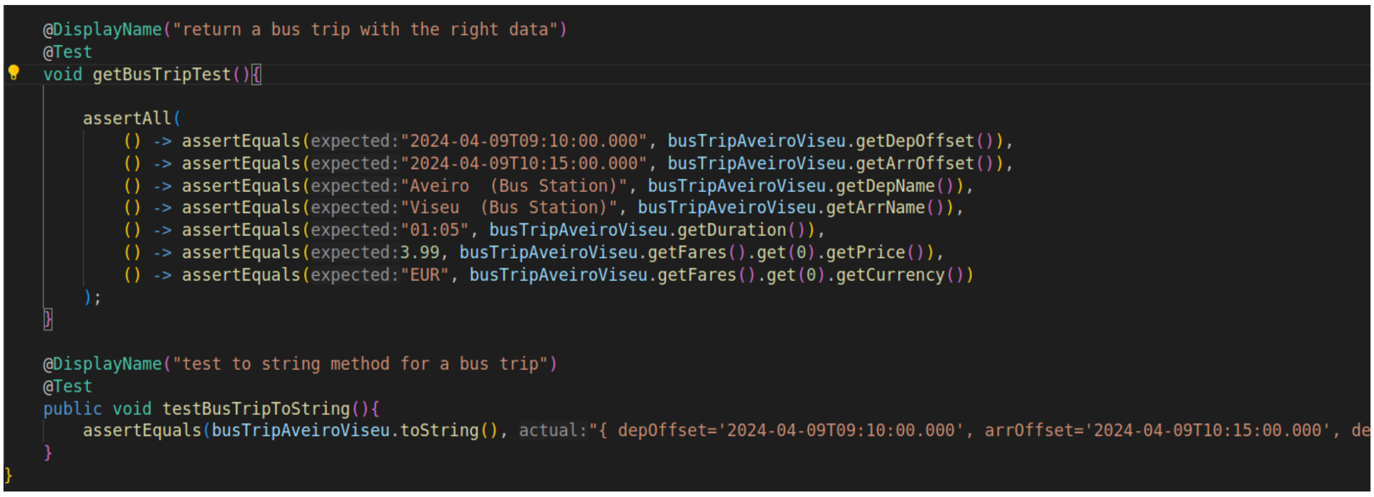
3.1 Overall strategy for testing

Ideally all test and application development would be TDD (Test Driven Development). Some tests were developed before the development of the features.
However, due to the agreement between the models and the api, some tests were finished after the development of the connection with the api, for example the integration tests.
The cucumber was used to test the frontend workflow using chrome driver.
They were implemented unit tests, integration tests, cucumber tests using steps for a trip search and the frontend and selenium tests using the Selenium IDE.

The TDD is extremely important to understand the structure needed to develop all the features idealized.

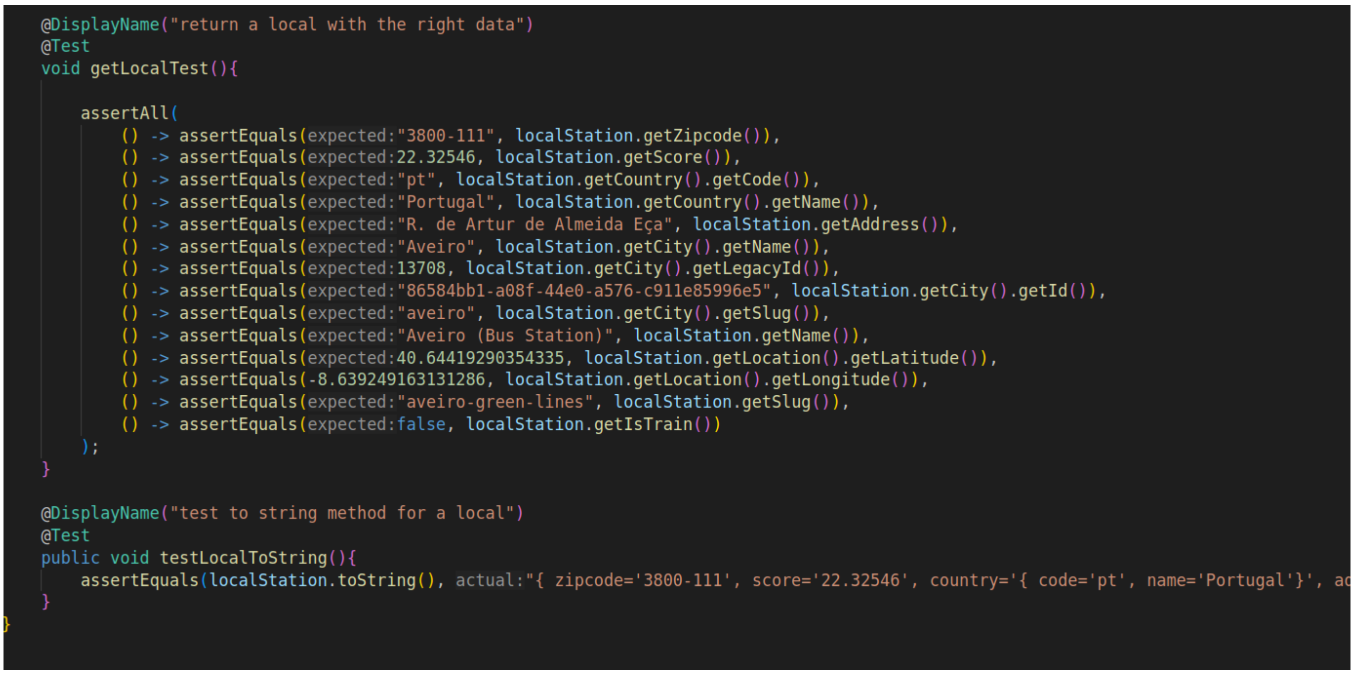
3.2 Unit and integration testing

Unit tests were used to test the BusTrip and Local classes. The BusTrip class represents a bus trip and the Local class represents all the information about a specific city. These tests validate the data introduced.



```
    @DisplayName("return a bus trip with the right data")
    @Test
    void getBusTripTest(){

        assertAll(
            () -> assertEquals(expected:"2024-04-09T09:10:00.000", busTripAveiroViseu.getDepOffset()),
            () -> assertEquals(expected:"2024-04-09T10:15:00.000", busTripAveiroViseu.getArrOffset()),
            () -> assertEquals(expected:"Aveiro  (Bus Station)", busTripAveiroViseu.getDepName()),
            () -> assertEquals(expected:"Viseu  (Bus Station)", busTripAveiroViseu.getArrName()),
            () -> assertEquals(expected:"01:05", busTripAveiroViseu.getDuration()),
            () -> assertEquals(expected:3.99, busTripAveiroViseu.getFares().get(0).getPrice()),
            () -> assertEquals(expected:"EUR", busTripAveiroViseu.getFares().get(0).getCurrency())
        );
    }

    @DisplayName("test to string method for a bus trip")
    @Test
    public void testBusTripToString(){
        assertEquals(busTripAveiroViseu.toString(), actual:"{ depOffset='2024-04-09T09:10:00.000', arrOffset='2024-04-09T10:15:00.000', de
    }
}
```

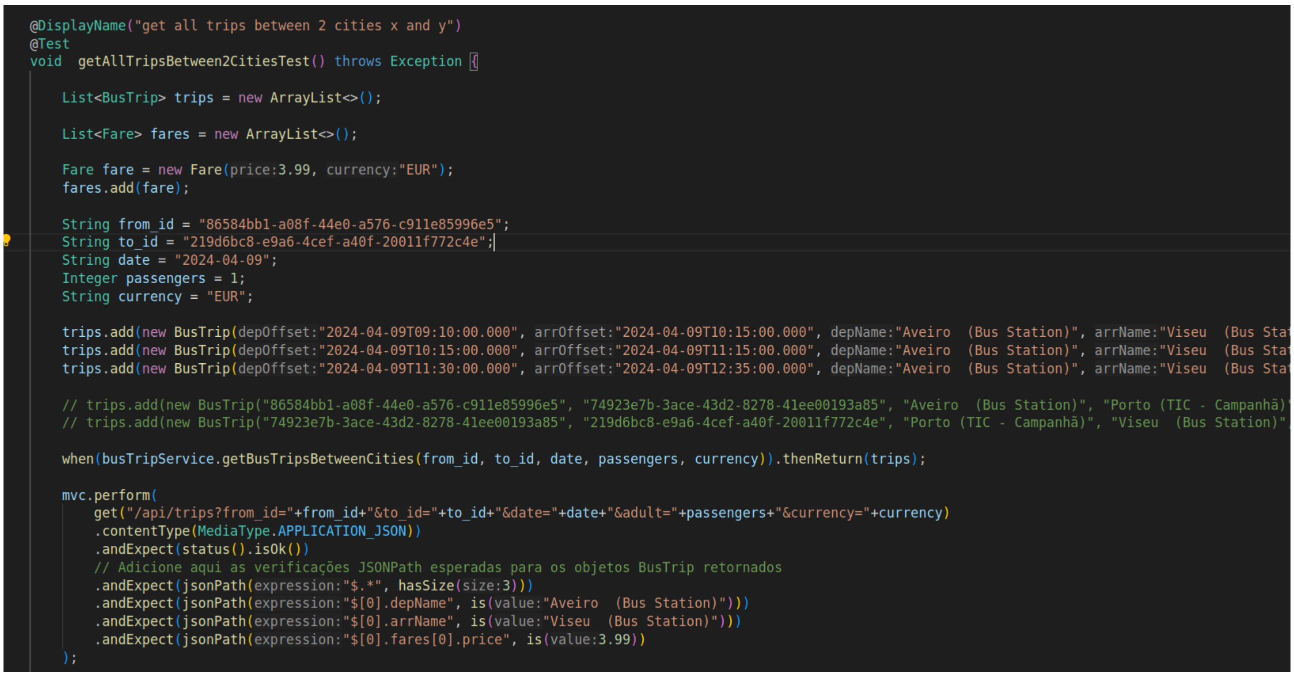
Image 3 - Unit test for a bus trip



```java
    @DisplayName("return a local with the right data")
    @Test
    void getLocalTest(){

        assertAll(
            () -> assertEquals(expected:"3800-111", localStation.getZipcode()),
            () -> assertEquals(expected:22.32546, localStation.getScore()),
            () -> assertEquals(expected:"pt", localStation.getCountry().getCode()),
            () -> assertEquals(expected:"Portugal", localStation.getCountry().getName()),
            () -> assertEquals(expected:"R. de Artur de Almeida Eça", localStation.getAddress()),
            () -> assertEquals(expected:"Aveiro", localStation.getCity().getName()),
            () -> assertEquals(expected:13708, localStation.getCity().getLegacyId()),
            () -> assertEquals(expected:"86584bb1-a08f-44e0-a576-c911e85996e5", localStation.getCity().getId()),
            () -> assertEquals(expected:"aveiro", localStation.getCity().getSlug()),
            () -> assertEquals(expected:"Aveiro (Bus Station)", localStation.getName()),
            () -> assertEquals(expected:40.64419290354335, localStation.getLocation().getLatitude()),
            () -> assertEquals(-8.639249163131286, localStation.getLocation().getLongitude()),
            () -> assertEquals(expected:"aveiro-green-lines", localStation.getSlug()),
            () -> assertEquals(expected:false, localStation.getIsTrain())
        );
    }

    @DisplayName("test to string method for a local")
    @Test
    public void testLocalToString(){
        assertEquals(localStation.toString(), actual:"{ zipcode='3800-111', score='22.32546', country='{ code='pt', name='Portugal'}', ac
    }
}
```

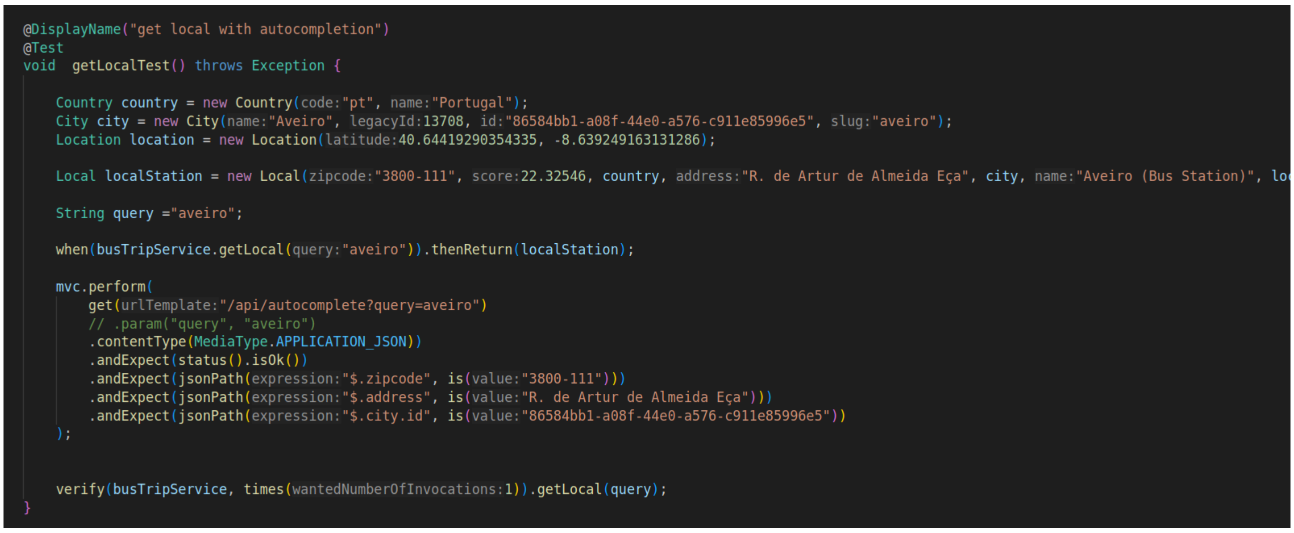
Image 4 - Unit test for a local

For integration tests, the connection and access to the external api and the controller api was tested. The strategy chosen was MockMvc provided by SpringBoot to test every enpoint of the controller. These tests verify if the information returned equals the expected one.



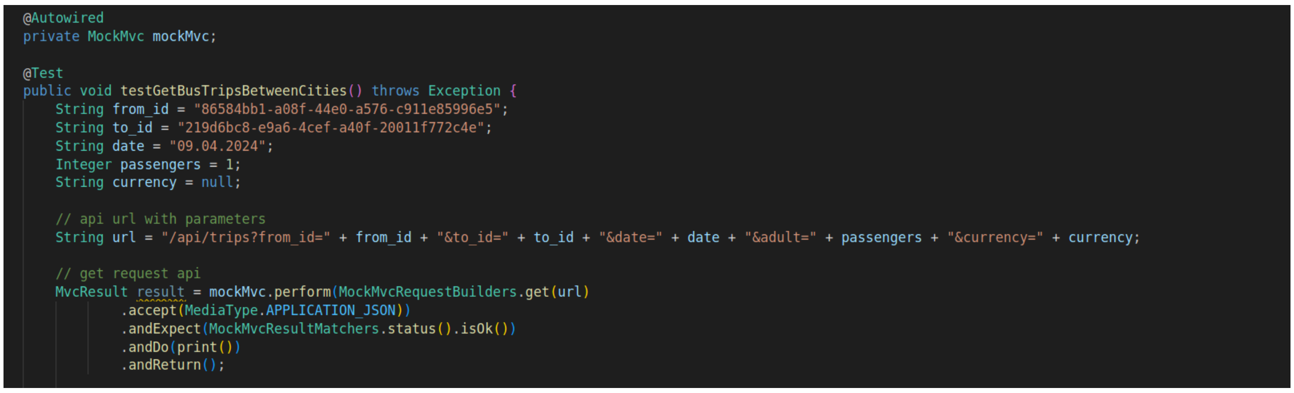
```java
    @DisplayName("get all trips between 2 cities x and y")
    @Test
    void  getAllTripsBetween2CitiesTest() throws Exception {

        List<BusTrip> trips = new ArrayList<>();

        List<Fare> fares = new ArrayList<>();

        Fare fare = new Fare(price:3.99, currency:"EUR");
        fares.add(fare);

        String from_id = "86584bb1-a08f-44e0-a576-c911e85996e5";
        String to_id = "219d6bc8-e9a6-4cef-a40f-20011f772c4e";
        String date = "2024-04-09";
        Integer passengers = 1;
        String currency = "EUR";

        trips.add(new BusTrip(depOffset:"2024-04-09T09:10:00.000", arrOffset:"2024-04-09T10:15:00.000", depName:"Aveiro  (Bus Station)", arrName:"Viseu  (Bus Sta
        trips.add(new BusTrip(depOffset:"2024-04-09T10:15:00.000", arrOffset:"2024-04-09T11:15:00.000", depName:"Aveiro  (Bus Station)", arrName:"Viseu  (Bus Sta
        trips.add(new BusTrip(depOffset:"2024-04-09T11:30:00.000", arrOffset:"2024-04-09T12:35:00.000", depName:"Aveiro  (Bus Station)", arrName:"Viseu  (Bus Sta

        // trips.add(new BusTrip("86584bb1-a08f-44e0-a576-c911e85996e5", "74923e7b-3ace-43d2-8278-41ee00193a85", "Aveiro  (Bus Station)", "Porto (TIC - Campanhã)"
        // trips.add(new BusTrip("74923e7b-3ace-43d2-8278-41ee00193a85", "219d6bc8-e9a6-4cef-a40f-20011f772c4e", "Porto (TIC - Campanhã)", "Viseu  (Bus Station)",

        when(busTripService.getBusTripsBetweenCities(from_id, to_id, date, passengers, currency)).thenReturn(trips);

        mvc.perform(
            get("/api/trips?from_id="+from_id+"&to_id="+to_id+"&date="+date+"&adult="+passengers+"&currency="+currency)
            .contentType(MediaType.APPLICATION_JSON))
            .andExpect(status().isOk())
            // Adicione aqui as verificações JSONPath esperadas para os objetos BusTrip retornados
            .andExpect(jsonPath(expression:"$.*", hasSize(size:3)))
            .andExpect(jsonPath(expression:"$[0].depName", is(value:"Aveiro  (Bus Station)")))
            .andExpect(jsonPath(expression:"$[0].arrName", is(value:"Viseu  (Bus Station)")))
            .andExpect(jsonPath(expression:"$[0].fares[0].price", is(value:3.99)))
        );
```

Image 5 - Integration test for controller endpoint /trips



```java
@DisplayName("get local with autocompletion")
@Test
void  getLocalTest() throws Exception {

    Country country = new Country(code:"pt", name:"Portugal");
    City city = new City(name:"Aveiro", legacyId:13708, id:"86584bb1-a08f-44e0-a576-c911e85996e5", slug:"aveiro");
    Location location = new Location(latitude:40.64419290354335, -8.639249163131286);

    Local localStation = new Local(zipcode:"3800-111", score:22.32546, country, address:"R. de Artur de Almeida Eça", city, name:"Aveiro (Bus Station)", lo

    String query ="aveiro";

    when(busTripService.getLocal(query:"aveiro")).thenReturn(localStation);

    mvc.perform(
        get(urlTemplate:"/api/autocomplete?query=aveiro")
        // .param("query", "aveiro")
        .contentType(MediaType.APPLICATION_JSON))
        .andExpect(status().isOk())
        .andExpect(jsonPath(expression:"$.zipcode", is(value:"3800-111")))
        .andExpect(jsonPath(expression:"$.address", is(value:"R. de Artur de Almeida Eça")))
        .andExpect(jsonPath(expression:"$.city.id", is(value:"86584bb1-a08f-44e0-a576-c911e85996e5"))
    );


    verify(busTripService, times(wantedNumberOfInvocations:1)).getLocal(query);
}
```

Image 6 - Integration test for controller endpoint /autocomplete

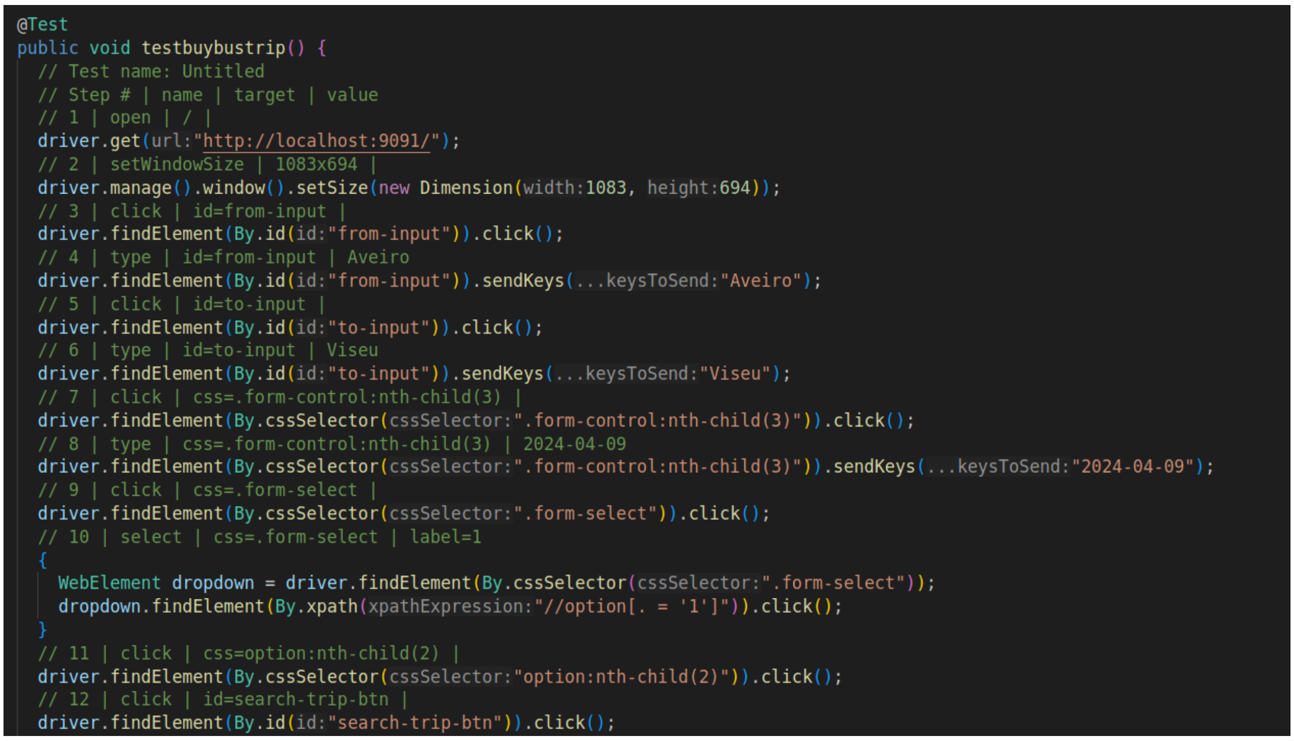


```java
@Autowired
private MockMvc mockMvc;

@Test
public void testGetBusTripsBetweenCities() throws Exception {
    String from_id = "86584bb1-a08f-44e0-a576-c911e85996e5";
    String to_id = "219d6bc8-e9a6-4cef-a40f-20011f772c4e";
    String date = "09.04.2024";
    Integer passengers = 1;
    String currency = null;

    // api url with parameters
    String url = "/api/trips?from_id=" + from_id + "&to_id=" + to_id + "&date=" + date + "&adult=" + passengers + "&currency=" + currency;

    // get request api
    MvcResult result = mockMvc.perform(MockMvcRequestBuilders.get(url)
            .accept(MediaType.APPLICATION_JSON))
            .andExpect(MockMvcResultMatchers.status().isOk())
            .andDo(print())
            .andReturn();
```

Image 7 - Integration test for bus trip service

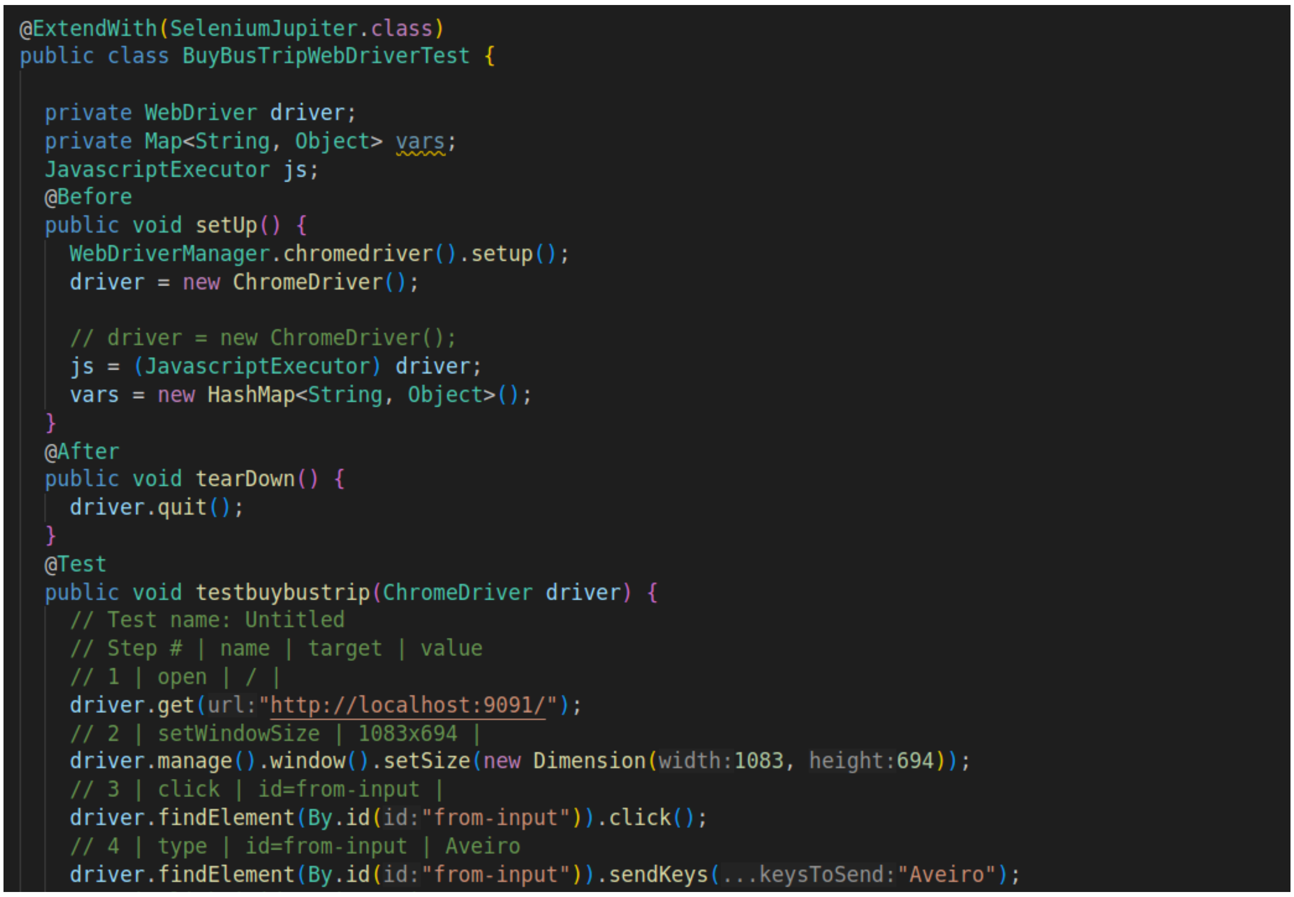
3.3 Functional testing

For the functional testing the tests were developed using the Selenium extension who created the file bustrip.side. This type of testing tests the functionalities of the bus trip application against some specifications.



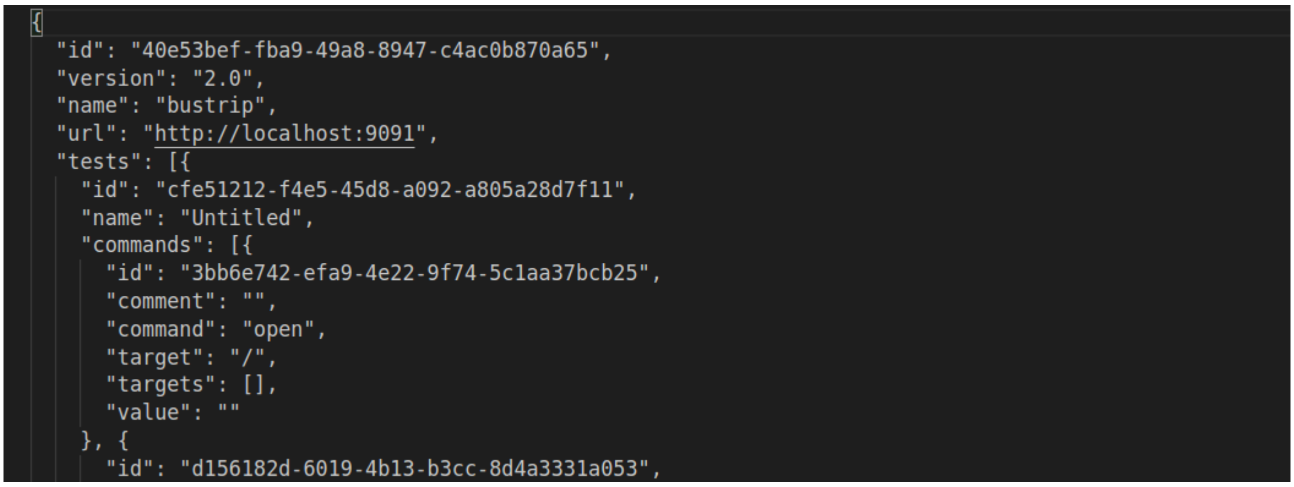
```java
@Test
public void testbuybustrip() {
  // Test name: Untitled
  // Step # | name | target | value
  // 1 | open | / |
  driver.get(url:"http://localhost:9091/");
  // 2 | setWindowSize | 1083x694 |
  driver.manage().window().setSize(new Dimension(width:1083, height:694));
  // 3 | click | id=from-input |
  driver.findElement(By.id(id:"from-input")).click();
  // 4 | type | id=from-input | Aveiro
  driver.findElement(By.id(id:"from-input")).sendKeys(...keysToSend:"Aveiro");
  // 5 | click | id=to-input |
  driver.findElement(By.id(id:"to-input")).click();
  // 6 | type | id=to-input | Viseu
  driver.findElement(By.id(id:"to-input")).sendKeys(...keysToSend:"Viseu");
  // 7 | click | css=.form-control:nth-child(3) |
  driver.findElement(By.cssSelector(cssSelector:".form-control:nth-child(3)")).click();
  // 8 | type | css=.form-control:nth-child(3) | 2024-04-09
  driver.findElement(By.cssSelector(cssSelector:".form-control:nth-child(3)")).sendKeys(...keysToSend:"2024-04-09");
  // 9 | click | css=.form-select |
  driver.findElement(By.cssSelector(cssSelector:".form-select")).click();
  // 10 | select | css=.form-select | label=1
  {
    WebElement dropdown = driver.findElement(By.cssSelector(cssSelector:".form-select"));
    dropdown.findElement(By.xpath(xpathExpression:"//option[. = '1']")).click();
  }
  // 11 | click | css=option:nth-child(2) |
  driver.findElement(By.cssSelector(cssSelector:"option:nth-child(2)")).click();
  // 12 | click | id=search-trip-btn |
  driver.findElement(By.id(id:"search-trip-btn")).click();
```

Image 8 - Functional test with selenium



```java
@ExtendWith(SeleniumJupiter.class)
public class BuyBusTripWebDriverTest {

  private WebDriver driver;
  private Map<String, Object> vars;
  JavascriptExecutor js;
  @Before
  public void setUp() {
    WebDriverManager.chromedriver().setup();
    driver = new ChromeDriver();

    // driver = new ChromeDriver();
    js = (JavascriptExecutor) driver;
    vars = new HashMap<String, Object>();
  }
  @After
  public void tearDown() {
    driver.quit();
  }
  @Test
  public void testbuybustrip(ChromeDriver driver) {
    // Test name: Untitled
    // Step # | name | target | value
    // 1 | open | / |
    driver.get(url:"http://localhost:9091/");
    // 2 | setWindowSize | 1083x694 |
    driver.manage().window().setSize(new Dimension(width:1083, height:694));
    // 3 | click | id=from-input |
    driver.findElement(By.id(id:"from-input")).click();
    // 4 | type | id=from-input | Aveiro
    driver.findElement(By.id(id:"from-input")).sendKeys(...keysToSend:"Aveiro");
```

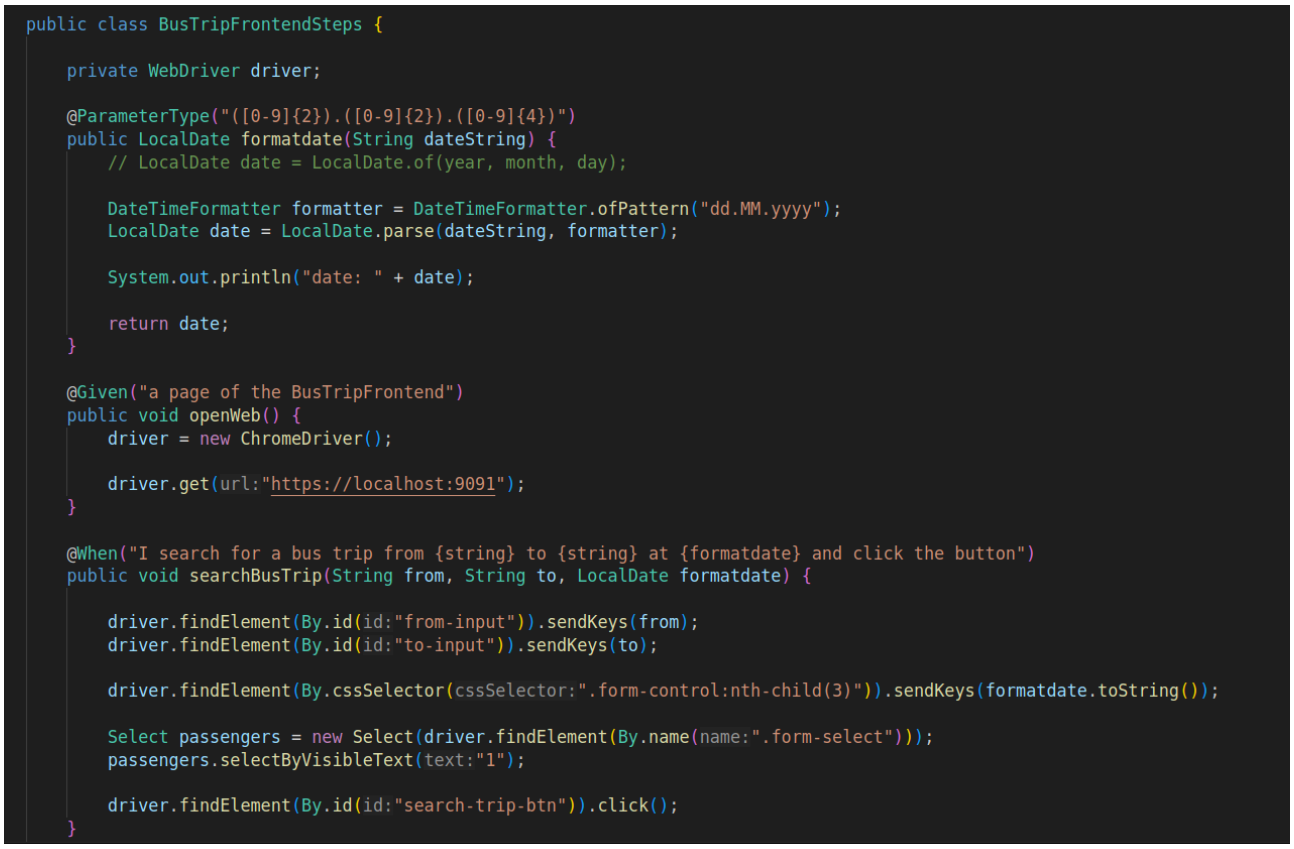
Image 9 - Functional test with selenium and webdriver chrome



```json
{
  "id": "40e53bef-fba9-49a8-8947-c4ac0b870a65",
  "version": "2.0",
  "name": "bustrip",
  "url": "http://localhost:9091",
  "tests": [{
    "id": "cfe51212-f4e5-45d8-a092-a805a28d7f11",
    "name": "Untitled",
    "commands": [{
      "id": "3bb6e742-efa9-4e22-9f74-5c1aa37bcb25",
      "comment": "",
      "command": "open",
      "target": "/",
      "targets": [],
      "value": ""
    }, {
      "id": "d156182d-6019-4b13-b3cc-8d4a3331a053",
```

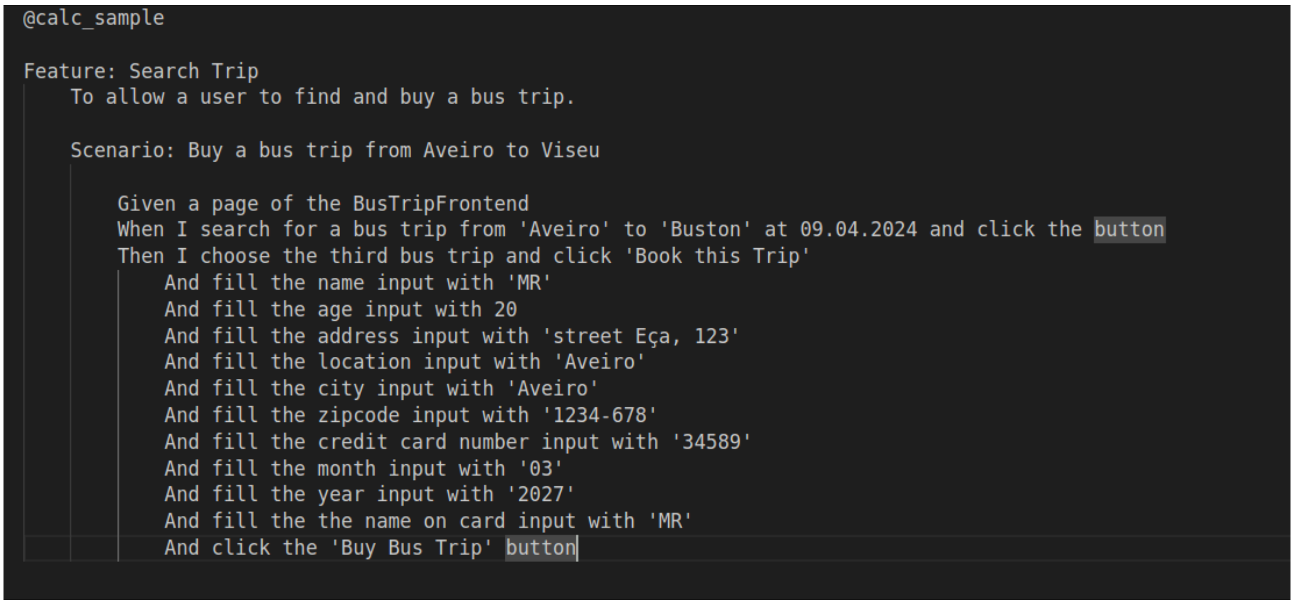
Image 10 - bustrip.side file

On top of this, a cucumber test was developed to evaluate the application steps. However, this test is commented on because of dependency errors.



```java
public class BusTripFrontendSteps {

    private WebDriver driver;

    @ParameterType("([0-9]{2}).([0-9]{2}).([0-9]{4})")
    public LocalDate formatdate(String dateString) {
        // LocalDate date = LocalDate.of(year, month, day);

        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd.MM.yyyy");
        LocalDate date = LocalDate.parse(dateString, formatter);

        System.out.println("date: " + date);

        return date;
    }

    @Given("a page of the BusTripFrontend")
    public void openWeb() {
        driver = new ChromeDriver();

        driver.get(url:"https://localhost:9091");
    }

    @When("I search for a bus trip from {string} to {string} at {formatdate} and click the button")
    public void searchBusTrip(String from, String to, LocalDate formatdate) {

        driver.findElement(By.id(id:"from-input")).sendKeys(from);
        driver.findElement(By.id(id:"to-input")).sendKeys(to);

        driver.findElement(By.cssSelector(cssSelector:".form-control:nth-child(3)")).sendKeys(formatdate.toString());

        Select passengers = new Select(driver.findElement(By.name(name:".form-select")));
        passengers.selectByVisibleText(text:"1");

        driver.findElement(By.id(id:"search-trip-btn")).click();
    }
```

Image 11 - Cucumber Test Steps for the frontend



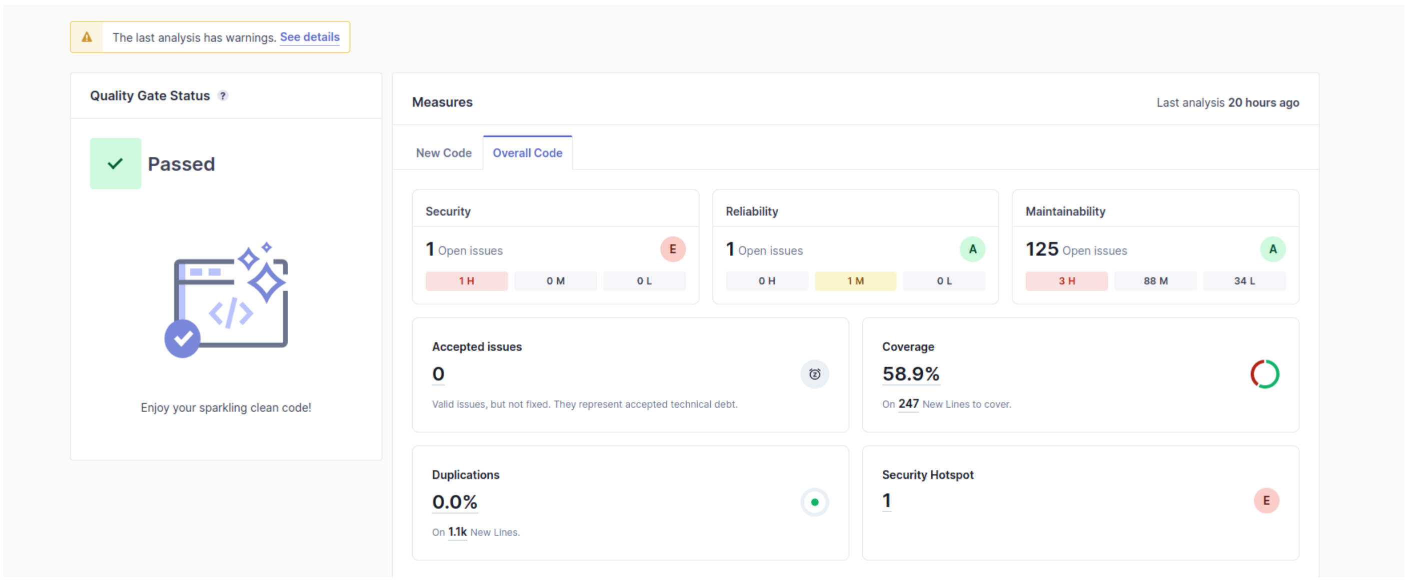
```gherkin
@calc_sample

Feature: Search Trip
    To allow a user to find and buy a bus trip.

    Scenario: Buy a bus trip from Aveiro to Viseu

        Given a page of the BusTripFrontend
        When I search for a bus trip from 'Aveiro' to 'Buston' at 09.04.2024 and click the button
        Then I choose the third bus trip and click 'Book this Trip'
            And fill the name input with 'MR'
            And fill the age input with 20
            And fill the address input with 'street Eça, 123'
            And fill the location input with 'Aveiro'
            And fill the city input with 'Aveiro'
            And fill the zipcode input with '1234-678'
            And fill the credit card number input with '34589'
            And fill the month input with '03'
            And fill the year input with '2027'
            And fill the the name on card input with 'MR'
            And click the 'Buy Bus Trip' button
```

Image 12 - Cucumber Test .feature file for the frontend

3.4 Code quality analysis

To code analysis it was used the SonarQube.

The average coverage is 58.9 per cent, which is not the best, but due to the time needed to solve some problems, it was as good as it could be.
The majority of the code smells are about comments in the code that should be deleted and some variables that are in snake_case instead of camelCase.


The last analysis has warnings. See details

Quality Gate Status

Passed

Enjoy your sparkling clean code!

Measures

Last analysis 20 hours ago

New Code | Overall Code

Security	Reliability	Maintainability
1 Open issues E	1 Open issues A	125 Open issues A
1 H · 0 M · 0 L	0 H · 1 M · 0 L	3 H · 88 M · 34 L

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

Coverage
58.9%
On 247 New Lines to cover.

Duplications
0.0%
On 1.1k New Lines.

Security Hotspot
1 E

Image 13 - Code Analysis with SonarQube

The first coverage was 52 per cent because of the existence of the class Reservation and some related classes. They could not be covered since the database and the repository was not implemented.

4 References & resources

Project resources

Resource:	URL/location:
Git repository	https://github.com/mariaabr/TQS_107658
Video demo	https://github.com/mariaabr/TQS_107658/tree/main/HW1/bus_trips/demo

Reference materials

https://rapidapi.com/3b-data-3b-data-default/api/flixbus2
https://swagger.io/