HW1: Mid-term assignment report

Rafael Sousa Vilaça [107476], 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 main purpose of the application is to:

- Search for bus connections (trips) between two cities.
- Book a Reservation for a passenger.

And it is possible to:

- Consider different types of seats (if a Seat is vip the price Multiplier is higher for example).
- Work with specific seats numbers (Individual Seats).
- See and use Seats booked/free per section in the route (Braga-Porto-Lisboa) in the section Braga-Porto the seat can be booked while it is free in Porto-Lisboa
- Cancel existing bookings (this deletes the reservation and frees the seat)

1.2 Current limitations

No data for travels in being used this means that all the routes are global.
Not a real problem since there would be a limit for routes by default anyway when the database was populated.
The data inserted is “hardcoded.”
Problems in creating Reservations by hand.

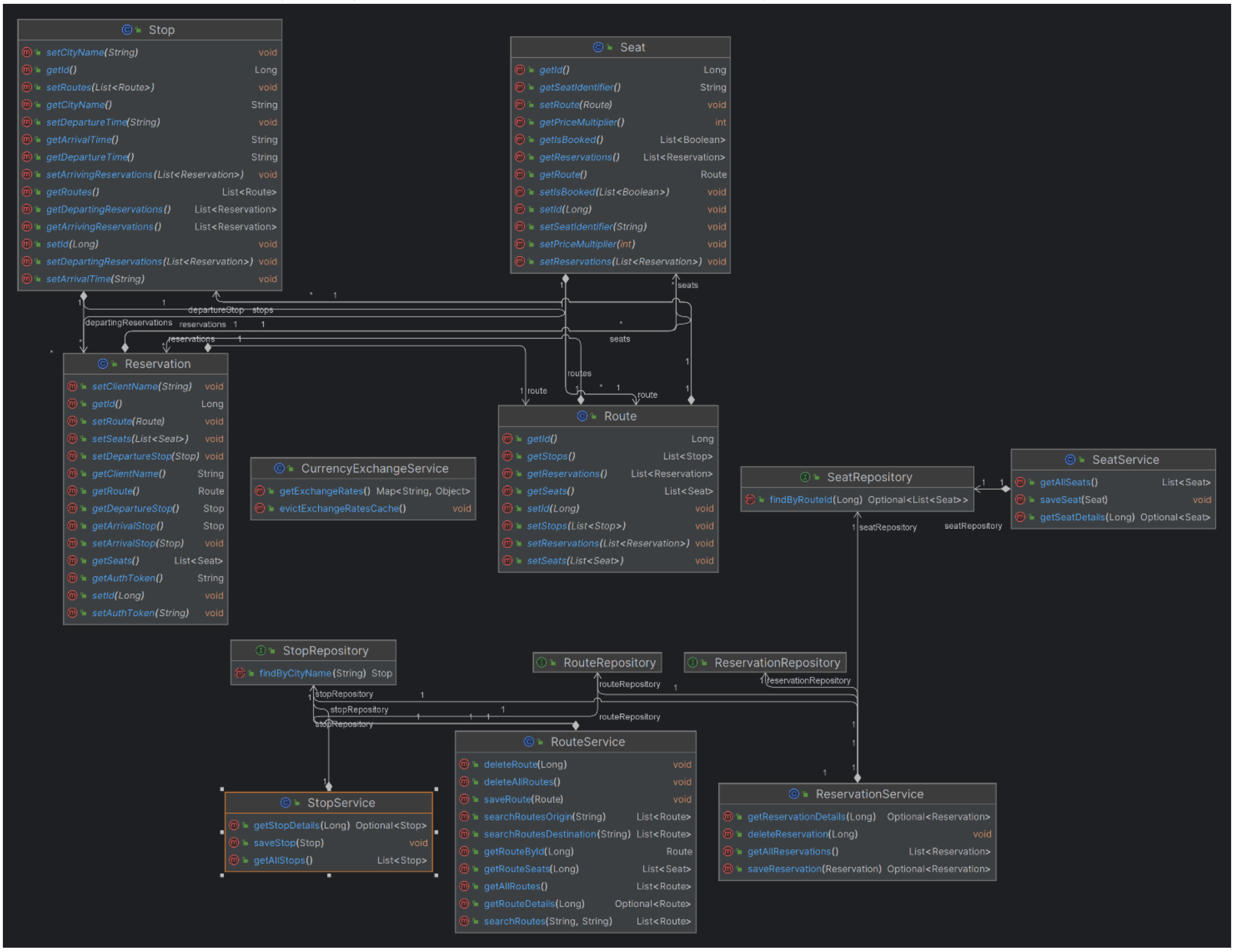
2 Product specification

2.1 Functional scope and supported interactions

The application I developed is specifically for client use. Clients can search for and book their desired travel arrangements, as well as manage their reservations by deleting or reviewing them.

I intentionally didn't include functionality for adding routes or stops, as they're not required for the primary use cases.

This is the classes Diagram (where all the methods are visible)


Stop
setCityName(String) void
getId() Long
setRoutes(List<Route>) void
getCityName() String
setDepartureTime(String) void
getArrivalTime() String
getDepartureTime() String
setArrivingReservations(List<Reservation>) void
getRoutes() List<Route>
getDepartingReservations() List<Reservation>
getArrivingReservations() List<Reservation>
setId(Long) void
setDepartingReservations(List<Reservation>) void
setArrivalTime(String) void
Seat
getId() Long
getSeatIdentifier() String
setRoute(Route) void
getPriceMultiplier() int
getIsBooked() List<Boolean>
getReservations() List<Reservation>
getRoute() Route
setIsBooked(List<Boolean>) void
setId(Long) void
setSeatIdentifier(String) void
setPriceMultiplier(int) void
setReservations(List<Reservation>) void
departureStop
stops
departingReservations
reservations
reservations
seats
seats
routes
route
route
Reservation
setClientName(String) void
getId() Long
setRoute(Route) void
setSeats(List<Seat>) void
setDepartureStop(Stop) void
getClientName() String
getRoute() Route
getDepartureStop() Stop
getArrivalStop() Stop
setArrivalStop(Stop) void
getSeats() List<Seat>
getAuthToken() String
setId(Long) void
setAuthToken(String) void
CurrencyExchangeService
getExchangeRates() Map<String, Object>
evictExchangeRatesCache() void
Route
getId() Long
getStops() List<Stop>
getReservations() List<Reservation>
getSeats() List<Seat>
setId(Long) void
setStops(List<Stop>) void
setReservations(List<Reservation>) void
setSeats(List<Seat>) void
SeatRepository
findByRouteId(Long) Optional<List<Seat>>
SeatService
getAllSeats() List<Seat>
saveSeat(Seat) void
getSeatDetails(Long) Optional<Seat>
seatRepository
seatRepository
StopRepository
findByCityName(String) Stop
RouteRepository
ReservationRepository
stopRepository
stopRepository
stopRepository
routeRepository
routeRepository
reservationRepository
StopService
getStopDetails(Long) Optional<Stop>
saveStop(Stop) void
getAllStops() List<Stop>
RouteService
deleteRoute(Long) void
deleteAllRoutes() void
saveRoute(Route) void
searchRoutesOrigin(String) List<Route>
searchRoutesDestination(String) List<Route>
getRouteById(Long) Route
getRouteSeats(Long) List<Seat>
getAllRoutes() List<Route>
getRouteDetails(Long) Optional<Route>
searchRoutes(String, String) List<Route>
ReservationService
getReservationDetails(Long) Optional<Reservation>
deleteReservation(Long) void
getAllReservations() List<Reservation>
saveReservation(Reservation) Optional<Reservation>

2.2 System architecture

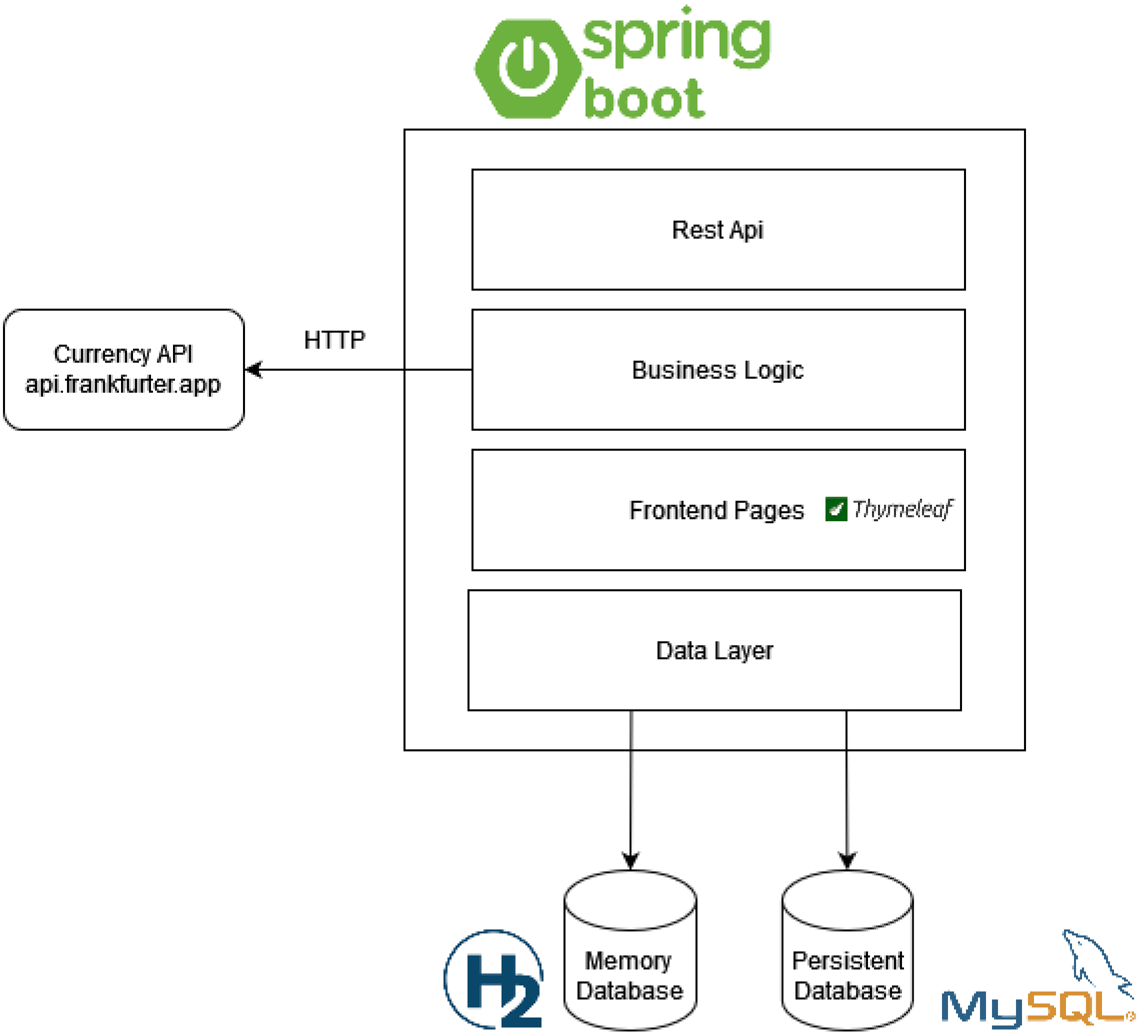
The system architecture is composed by:

-Thymeleaf – for rendering the html pages.

-External Api – Integrated to get current rates.

-H2 – used to store data for unit tests (of Repositories).

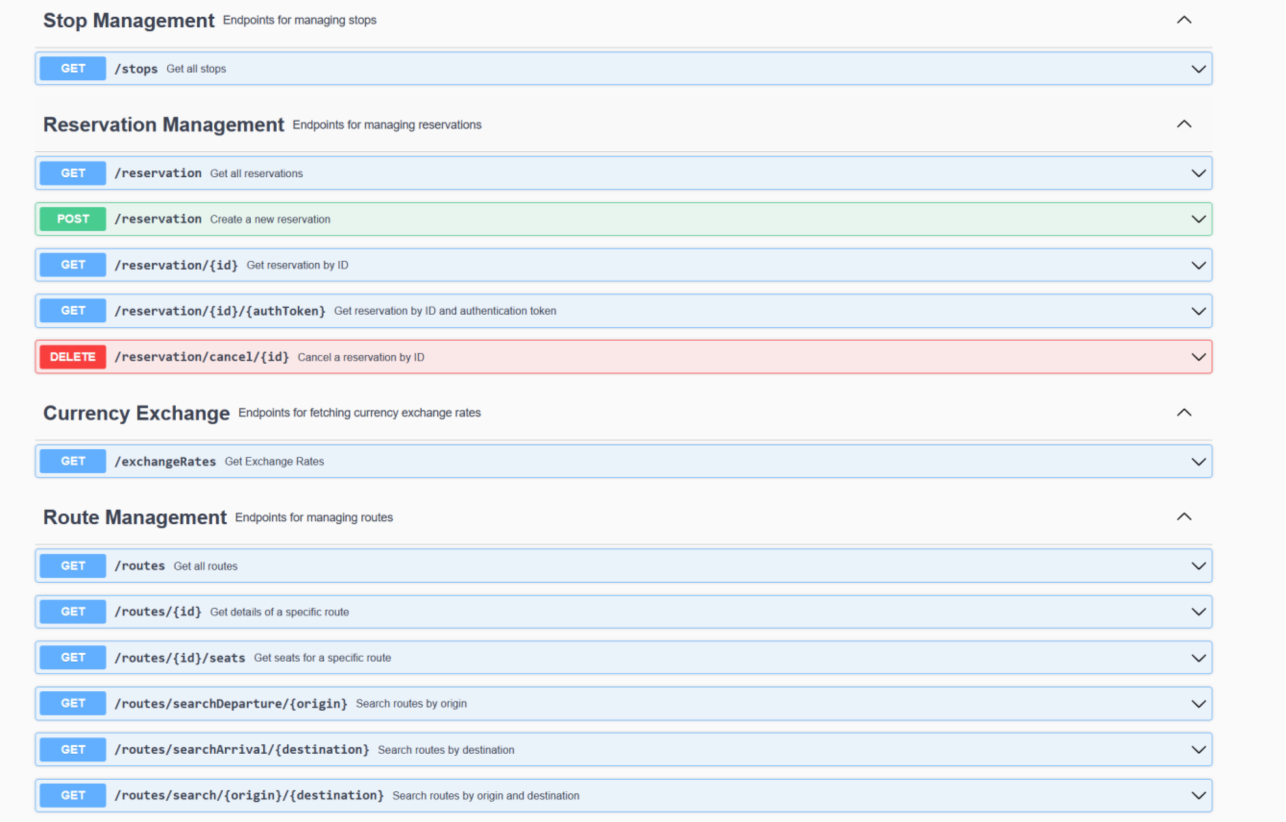
-Mysql – used to store data for IT tests etc.



2.3 API for developers

The documentation page can be accessed (with the application running) in http://localhost:8080/swagger-ui/index.html#/.

These are the Endpoints Available



Stop Management Endpoints for managing stops

GET `/stops` Get all stops

Reservation Management Endpoints for managing reservations

GET `/reservation` Get all reservations

POST `/reservation` Create a new reservation

GET `/reservation/{id}` Get reservation by ID

GET `/reservation/{id}/{authToken}` Get reservation by ID and authentication token

DELETE `/reservation/cancel/{id}` Cancel a reservation by ID

Currency Exchange Endpoints for fetching currency exchange rates

GET `/exchangeRates` Get Exchange Rates

Route Management Endpoints for managing routes

GET `/routes` Get all routes

GET `/routes/{id}` Get details of a specific route

GET `/routes/{id}/seats` Get seats for a specific route

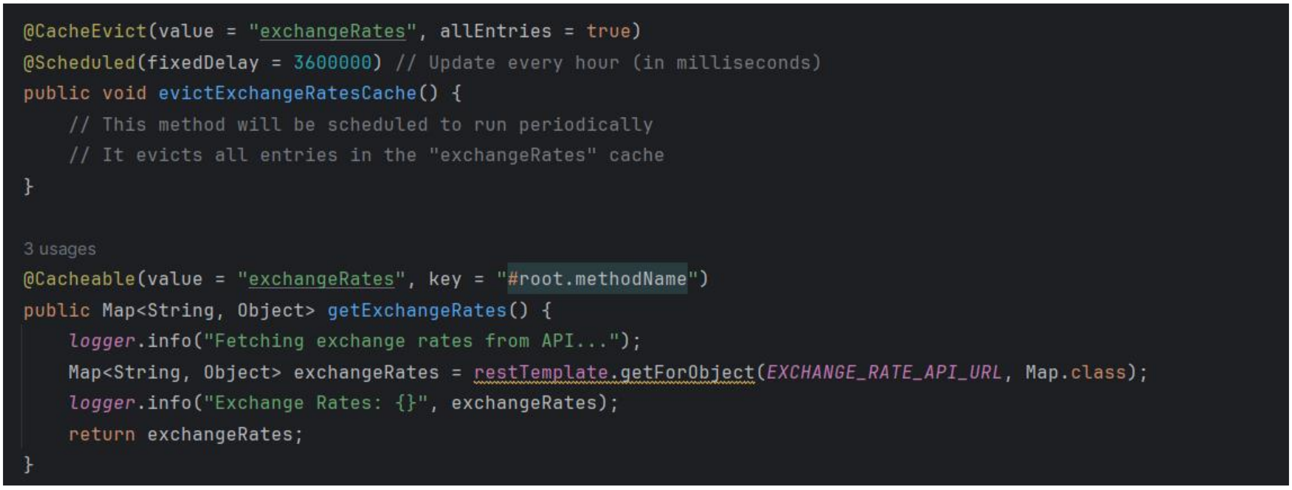
GET `/routes/searchDeparture/{origin}` Search routes by origin

GET `/routes/searchArrival/{destination}` Search routes by destination

GET `/routes/search/{origin}/{destination}` Search routes by origin and destination

No api cache hits or misses were taken in account because the cache is being hadled by the SpringBoot framework @Cacheble method.

The data is deleted every hour from the cache (Using @CacheEvict Method) and filled again when the respective endpoint is called.



```java
@CacheEvict(value = "exchangeRates", allEntries = true)
@Scheduled(fixedDelay = 3600000) // Update every hour (in milliseconds)
public void evictExchangeRatesCache() {
    // This method will be scheduled to run periodically
    // It evicts all entries in the "exchangeRates" cache
}

3 usages
@Cacheable(value = "exchangeRates", key = "#root.methodName")
public Map<String, Object> getExchangeRates() {
    logger.info("Fetching exchange rates from API...");
    Map<String, Object> exchangeRates = restTemplate.getForObject(EXCHANGE_RATE_API_URL, Map.class);
    logger.info("Exchange Rates: {}", exchangeRates);
    return exchangeRates;
}
```

3 Quality assurance

3.1 Overall strategy for testing

The overall test development strategy for the application primarily revolved around Test-Driven Development (TDD). This strategy helped the development process by focusing on requirements and ensuring that the code meets those requirements.

I started by building the skeleton of the classes I was implementing, returning null values and making sure the application compiled. Then, the tests were written, and only then the classes were actually implemented.

This allowed me to test my application while developing it, in order to find errors and bugs much earlier in development.

The API components were tested/developed in order, starting with the Controller, then the Service etc.

Worth noting that I don't love it when it is 1 Person development (lots of work) and in this specific case some frontend etc. was done to understand what was needed.

3.2 Unit and integration testing

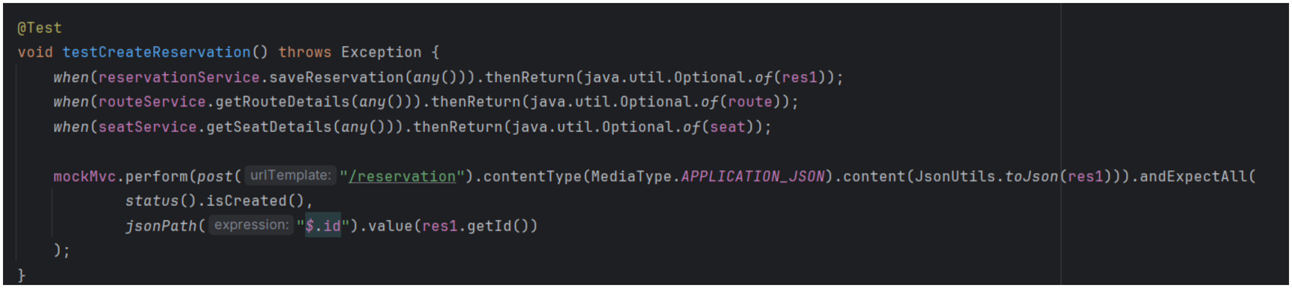
Each API component had their own unit tests written before being implemented. Most of these tests use Mockito to mock dependencies and isolate the test subject, like the service tests.

MockMvc was also used for the controller tests, in order to mock the requests being made to the controller.

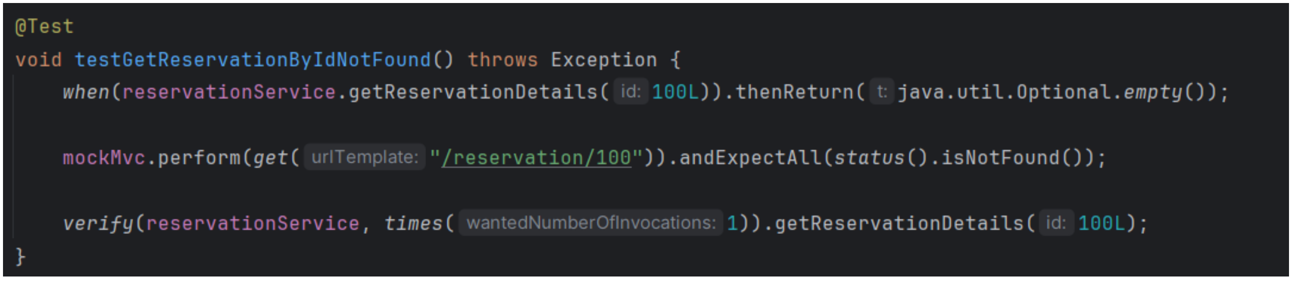
Here are some examples of dependency or request mocking using the specified libraries.

The first one shows a ReservationController test, where we mock the response from the Services needed and test the ReservationController object building functionality.

The second one tests the ReservationController class, by mocking a response to a bad id and expecting a NotFound code when an invalid id is provided.



```java
@Test
void testCreateReservation() throws Exception {
    when(reservationService.saveReservation(any())).thenReturn(java.util.Optional.of(res1));
    when(routeService.getRouteDetails(any())).thenReturn(java.util.Optional.of(route));
    when(seatService.getSeatDetails(any())).thenReturn(java.util.Optional.of(seat));

    mockMvc.perform(post( urlTemplate: "/reservation").contentType(MediaType.APPLICATION_JSON).content(JsonUtils.toJson(res1))).andExpectAll(
            status().isCreated(),
            jsonPath( expression: "$.id").value(res1.getId())
    );
}
```

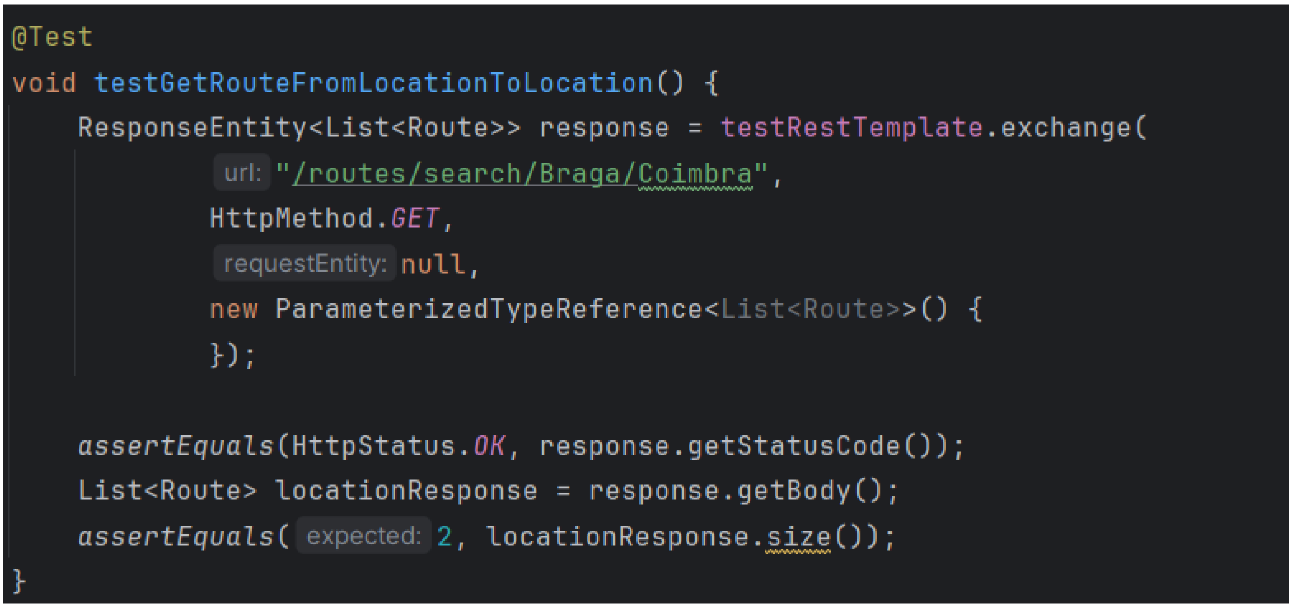


```java
@Test
void testGetReservationByIdNotFound() throws Exception {
    when(reservationService.getReservationDetails( id: 100L)).thenReturn( t: java.util.Optional.empty());

    mockMvc.perform(get( urlTemplate: "/reservation/100")).andExpectAll(status().isNotFound());

    verify(reservationService, times( wantedNumberOfInvocations: 1)).getReservationDetails( id: 100L);
}
```

After developing the components, we tested their interactions with each other using integration testing, making sure the application behaves correctly as a whole. For this, no mocking was used, and instead real calls were made to the respective components, expecting them to reply correctly.

For this purpose, the REST Assured library was used, to make actual requests to the controller.

The following snippet demonstrates one of the integration tests run - asking for the travels for today from Braga to Coimbra and expecting a valid response with 2 as the number of routes returned.

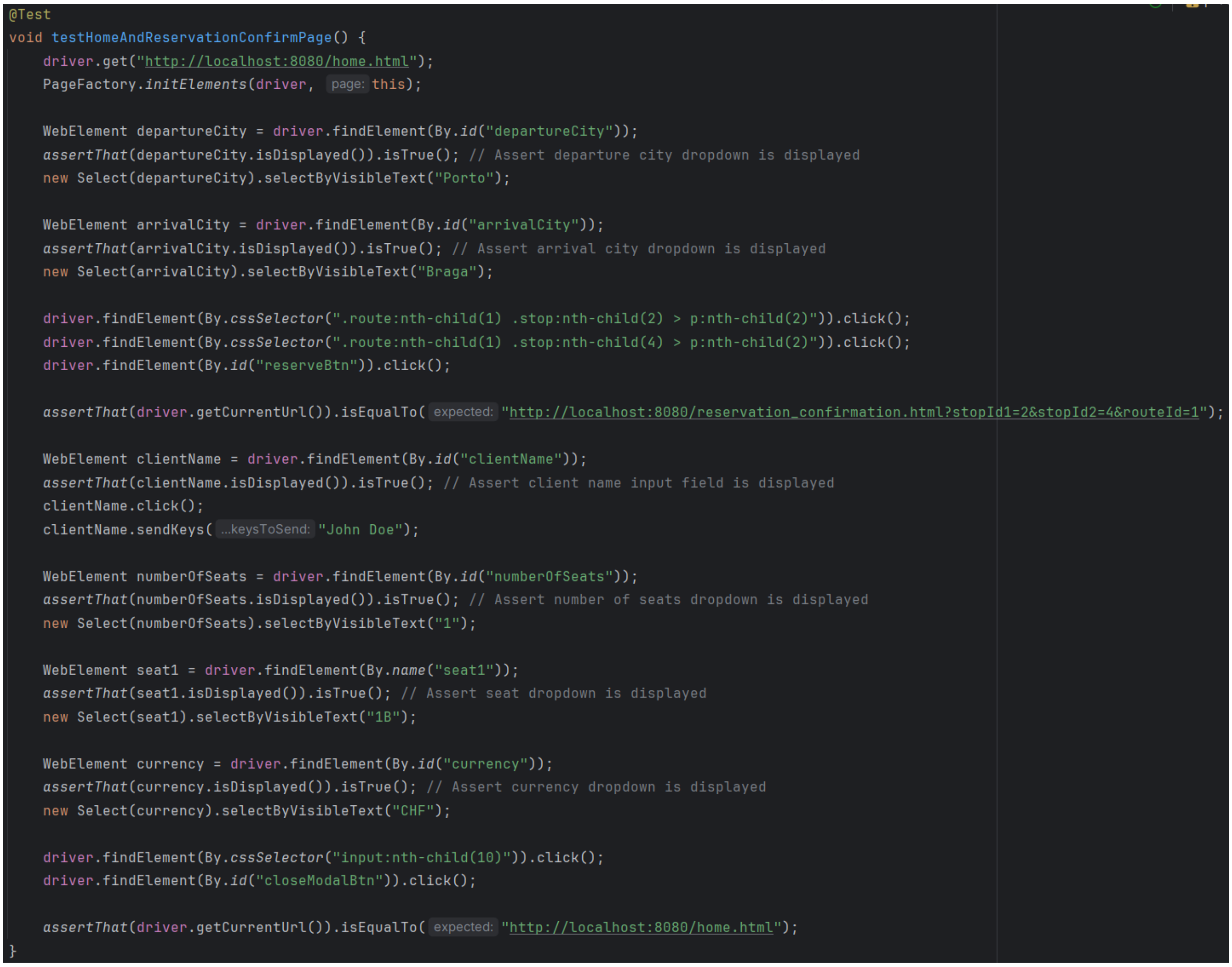


```
@Test
void testGetRouteFromLocationToLocation() {
    ResponseEntity<List<Route>> response = testRestTemplate.exchange(
            url: "/routes/search/Braga/Coimbra",
            HttpMethod.GET,
            requestEntity: null,
            new ParameterizedTypeReference<List<Route>>() {
            });

    assertEquals(HttpStatus.OK, response.getStatusCode());
    List<Route> locationResponse = response.getBody();
    assertEquals( expected: 2, locationResponse.size());
}
```

3.3 Functional testing

The web client testing was written using Selenium. The tests written give the app a location as input, through the text input present, and expect the app to change between pages and return some data.

Here is an illustrative code snippet from the functional testing class:

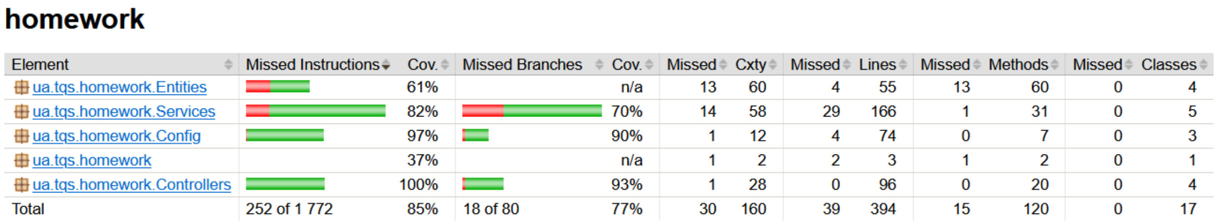


```java
@Test
void testHomeAndReservationConfirmPage() {
    driver.get("http://localhost:8080/home.html");
    PageFactory.initElements(driver,  page: this);

    WebElement departureCity = driver.findElement(By.id("departureCity"));
    assertThat(departureCity.isDisplayed()).isTrue(); // Assert departure city dropdown is displayed
    new Select(departureCity).selectByVisibleText("Porto");

    WebElement arrivalCity = driver.findElement(By.id("arrivalCity"));
    assertThat(arrivalCity.isDisplayed()).isTrue(); // Assert arrival city dropdown is displayed
    new Select(arrivalCity).selectByVisibleText("Braga");

    driver.findElement(By.cssSelector(".route:nth-child(1) .stop:nth-child(2) > p:nth-child(2)")).click();
    driver.findElement(By.cssSelector(".route:nth-child(1) .stop:nth-child(4) > p:nth-child(2)")).click();
    driver.findElement(By.id("reserveBtn")).click();

    assertThat(driver.getCurrentUrl()).isEqualTo( expected: "http://localhost:8080/reservation_confirmation.html?stopId1=2&stopId2=4&routeId=1");

    WebElement clientName = driver.findElement(By.id("clientName"));
    assertThat(clientName.isDisplayed()).isTrue(); // Assert client name input field is displayed
    clientName.click();
    clientName.sendKeys( ...keysToSend: "John Doe");

    WebElement numberOfSeats = driver.findElement(By.id("numberOfSeats"));
    assertThat(numberOfSeats.isDisplayed()).isTrue(); // Assert number of seats dropdown is displayed
    new Select(numberOfSeats).selectByVisibleText("1");

    WebElement seat1 = driver.findElement(By.name("seat1"));
    assertThat(seat1.isDisplayed()).isTrue(); // Assert seat dropdown is displayed
    new Select(seat1).selectByVisibleText("1B");

    WebElement currency = driver.findElement(By.id("currency"));
    assertThat(currency.isDisplayed()).isTrue(); // Assert currency dropdown is displayed
    new Select(currency).selectByVisibleText("CHF");

    driver.findElement(By.cssSelector("input:nth-child(10)")).click();
    driver.findElement(By.id("closeModalBtn")).click();

    assertThat(driver.getCurrentUrl()).isEqualTo( expected: "http://localhost:8080/home.html");
}
```

3.4 Code quality analysis

To analyze code quality, I used:

Jacoco



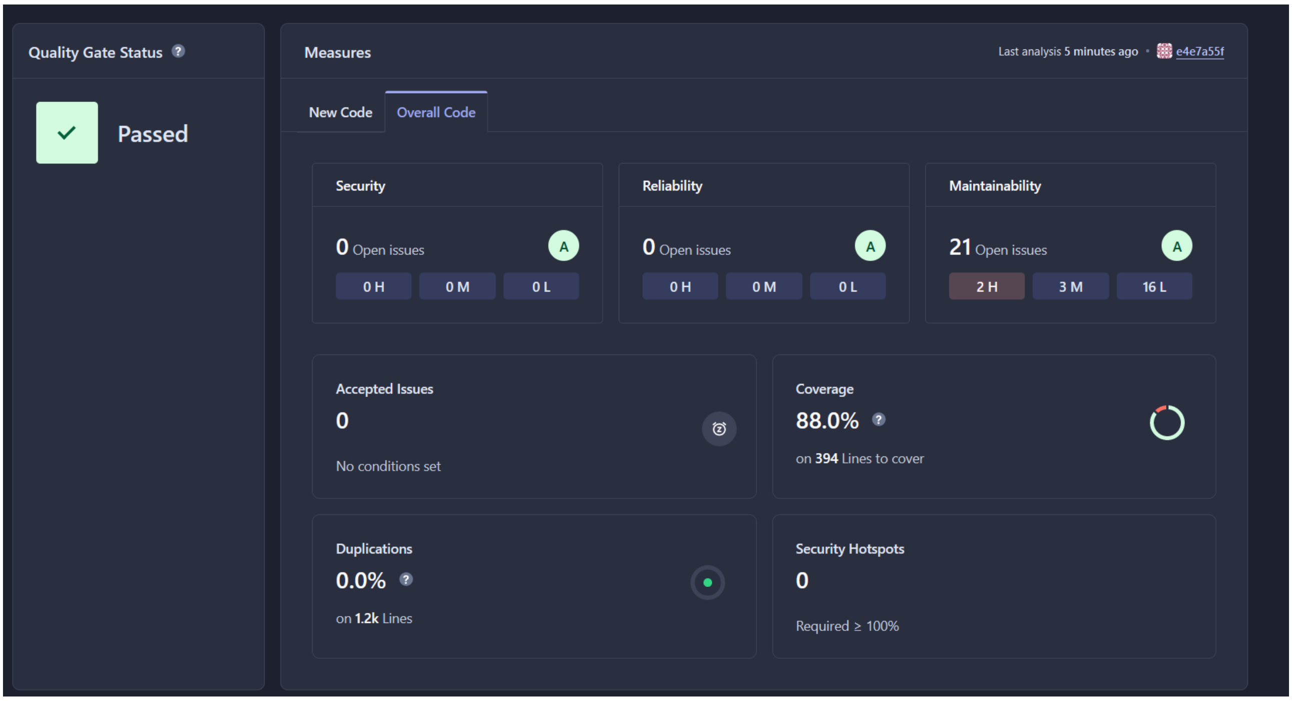
homework

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
ua.tqs.homework.Entities	🟥🟩	61%		n/a	13	60	4	55	13	60	0	4
ua.tqs.homework.Services	🟥🟩	82%	🟥🟩	70%	14	58	29	166	1	31	0	5
ua.tqs.homework.Config	🟩	97%	🟥🟩	90%	1	12	4	74	0	7	0	3
ua.tqs.homework		37%		n/a	1	2	2	3	1	2	0	1
ua.tqs.homework.Controllers	🟩	100%	🟥🟩	93%	1	28	0	96	0	20	0	4
Total	252 of 1 772	85%	18 of 80	77%	30	160	39	394	15	120	0	17

Global average of instructions at 85% and branches at 77% these results are very good and show that most of the instructions are being tested.

SonarCloud

These were the results.



Quality Gate Status

✓ Passed

Measures

Last analysis 5 minutes ago · e4e7a55f

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

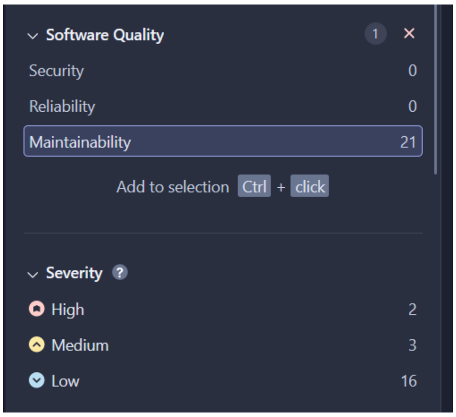
Maintainability
21 Open issues A
2 H | 3 M | 16 L

Accepted Issues
0
No conditions set

Coverage
88.0%
on 394 Lines to cover

Duplications
0.0%
on 1.2k Lines

Security Hotspots
0
Required ≥ 100%



Software Quality 1 ×

Security	0
Reliability	0
Maintainability	21

Add to selection Ctrl + click

Severity

High	2
Medium	3
Low	16

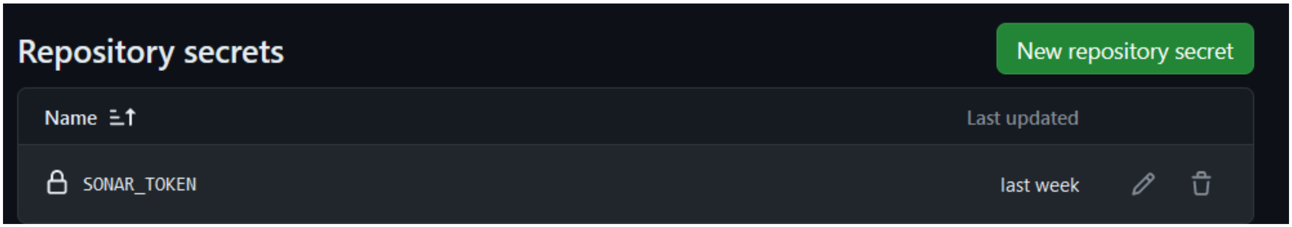
In the end the coverage is a good, no duplication, no security hotspots finnaly I still have some issues that in my opinion aren't real issues. All the low severity ones are a problem with the package name, the medium ones I don't know how to fix, and the high ones are ok (not real problems in my opinion). In general, both tools helped me find and fix some problems besides that I didn't encounter any difficult code smell to fix using any of them.

3.5 Continuous integration pipeline

A simple CI Pipeline was implemented using GitHub Actions. I created a workflow that runs all unit and integration tests when a push is made to the application folder (in this case, the hw1 folder in the repository).

When tests are successful, the application is also packaged, setting up the production deployment. At this point, a CD pipeline is also trivial to implement, but was not done due to the fact that the project is not currently deployed.

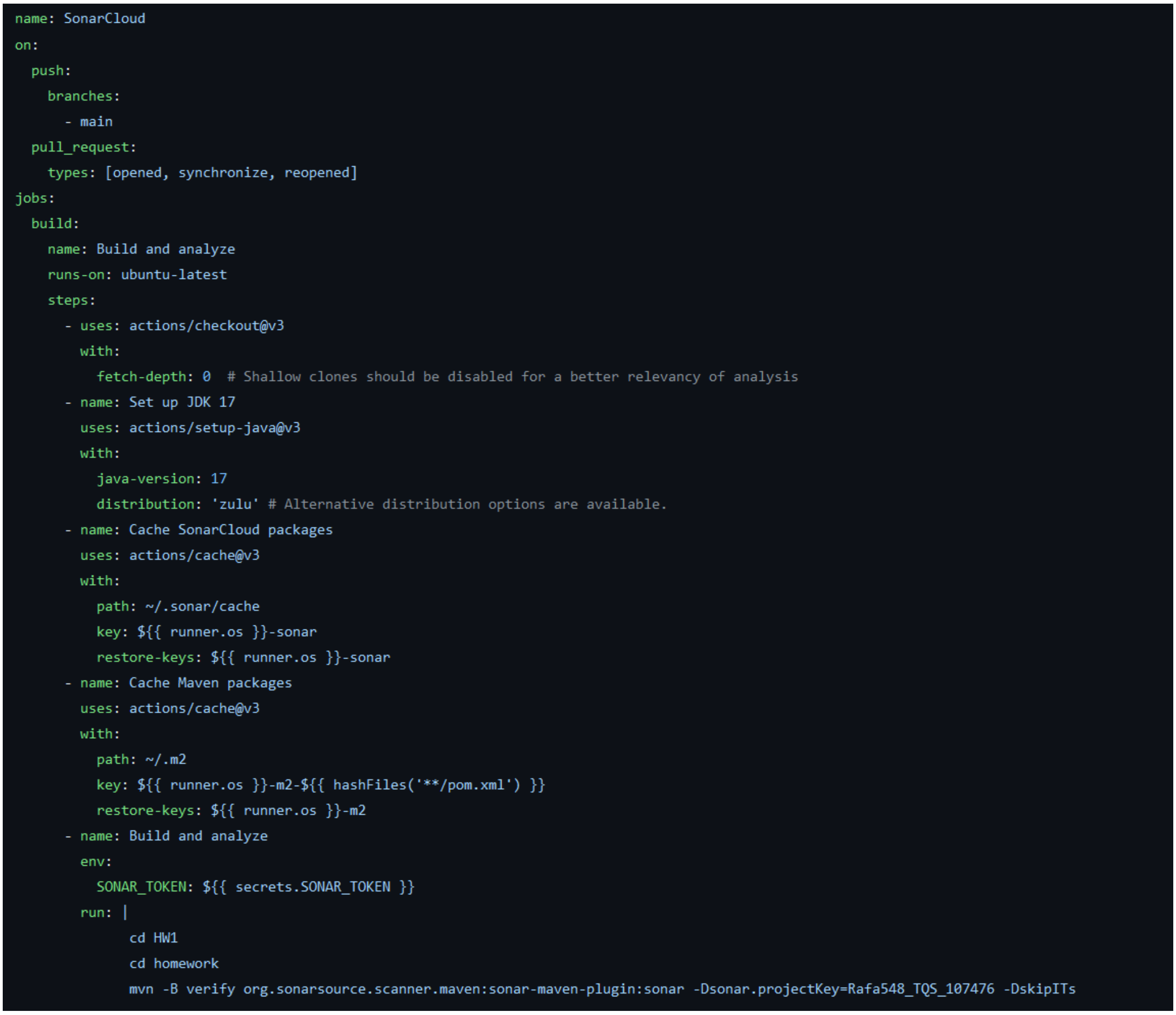
The respective token was configurated.



Repository secrets

Name	Last updated
SONAR_TOKEN	last week

Here is the GitHub Actions workflow responsible for continuously testing and building the application:



```
name: SonarCloud
on:
  push:
    branches:
      - main
  pull_request:
    types: [opened, synchronize, reopened]
jobs:
  build:
    name: Build and analyze
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
        with:
          fetch-depth: 0  # Shallow clones should be disabled for a better relevancy of analysis
      - name: Set up JDK 17
        uses: actions/setup-java@v3
        with:
          java-version: 17
          distribution: 'zulu' # Alternative distribution options are available.
      - name: Cache SonarCloud packages
        uses: actions/cache@v3
        with:
          path: ~/.sonar/cache
          key: ${{ runner.os }}-sonar
          restore-keys: ${{ runner.os }}-sonar
      - name: Cache Maven packages
        uses: actions/cache@v3
        with:
          path: ~/.m2
          key: ${{ runner.os }}-m2-${{ hashFiles('**/pom.xml') }}
          restore-keys: ${{ runner.os }}-m2
      - name: Build and analyze
        env:
          SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}
        run: |
              cd HW1
              cd homework
              mvn -B verify org.sonarsource.scanner.maven:sonar-maven-plugin:sonar -Dsonar.projectKey=Rafa548_TQS_107476 -DskipITs
```

Integration tests were skipped (using -DskipITs) since no sql database was provided

4 References & resources

Project resources

Resource:	**URL/location:**
Git repository	https://github.com/Rafa548/TQS_107476/tree/main/HW1/homework
Video demo	https://uapt33090-my.sharepoint.com/:f:/g/personal/rafael_vilaca_ua_pt/El_4VCv3DTJGqBWccLKZFKABXMIeNonwwT26dvV9EkU2ow?e=05i8MB
QA dashboard (online)	https://sonarcloud.io/project/overview?id=Rafa548_TQS_107476
CI/CD pipeline	GitHub actions
Deployment ready to use	Not done (ReadMe in GitHub page explains the configuration/execution)

Reference materials

Swagger Configuration -> https://www.baeldung.com/spring-rest-openapi-documentation
Open-Source API used -> https://github.com/hakanensari/frankfurter