deti universidade de aveiro departamento de eletrónica, telecomunicações e informática

HW1: Mid-term assignment report

João RIcardo Lopes Neto [113482], v2025-04-09

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 application enables users to make restaurant reservations across Portugal. Initially designed as a meal booking app for Moliceiro University campus, but because of my choice of the on-line API to get the weather forecast, I chose to have multiple restaurants across Portugal (so we can see different weather forecast in each restaurant). Besides normal reservations it was implemented an extra feature, group booking, this means it is possible to make a reservation for more than one person and have the correspondent number of meals.

1.2 Current limitations

The application does not include a user management system, as it was not required. This means that anyone with the reservation code can view it, and anyone who has access to the reservation can cancel it or check in. Additionally, no authorization is required to use the app's endpoints. Due to time constraints and the need to focus on multiple projects, I wasn't able to invest much time into developing an advanced frontend. Instead, I used Thymeleaf to

create a functional frontend that prioritizes the essential features needed for the app's intended purpose.

2 Product specification

2.1 Functional scope and supported interactions

Actors:

a) User
b) Restaurant Administrator

Scenarios:

c) Make a reservation
d) How does it work? The user opens the application, selects a restaurant, and is then redirected to a new page where they can enter the reservation details (meals, time, and number of people). Once the reservation is confirmed, a reservation code is displayed, which the user should save for future reference.
e) See reservation details
f) How does it work? The user can search for their reservation code, and if it exists, they will be redirected to a page displaying the reservation details.
g) Cancel a reservation
h) How does it work? The user can search for their reservation code, and if it exists, they will be redirected to a page displaying the reservation details. From there, they will have the option to cancel the reservation.
i) Check-in a reservation
j) How does it work? The restaurant administrator can search for a reservation code, and if it exists, they will be redirected to a page displaying the reservation details. From there, they will have the option to check-in the reservation.

2.2 System implementation architecture

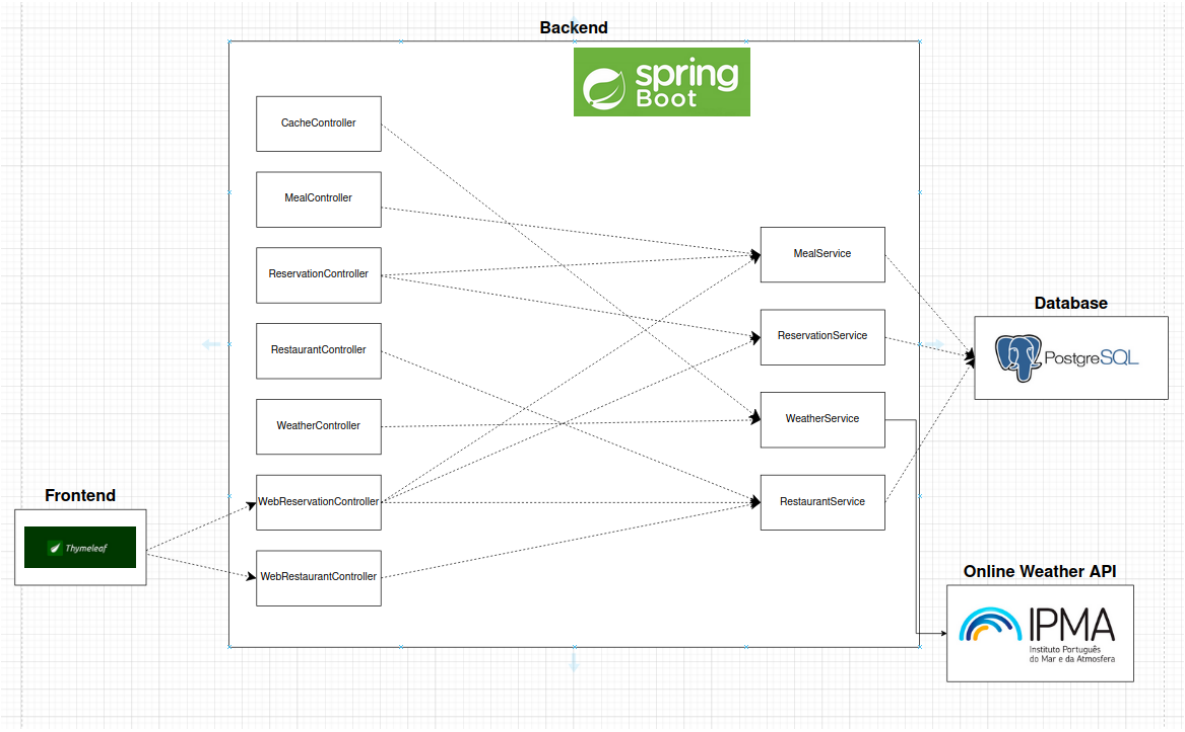

Fig.1 Architecture

2.3 API for developers

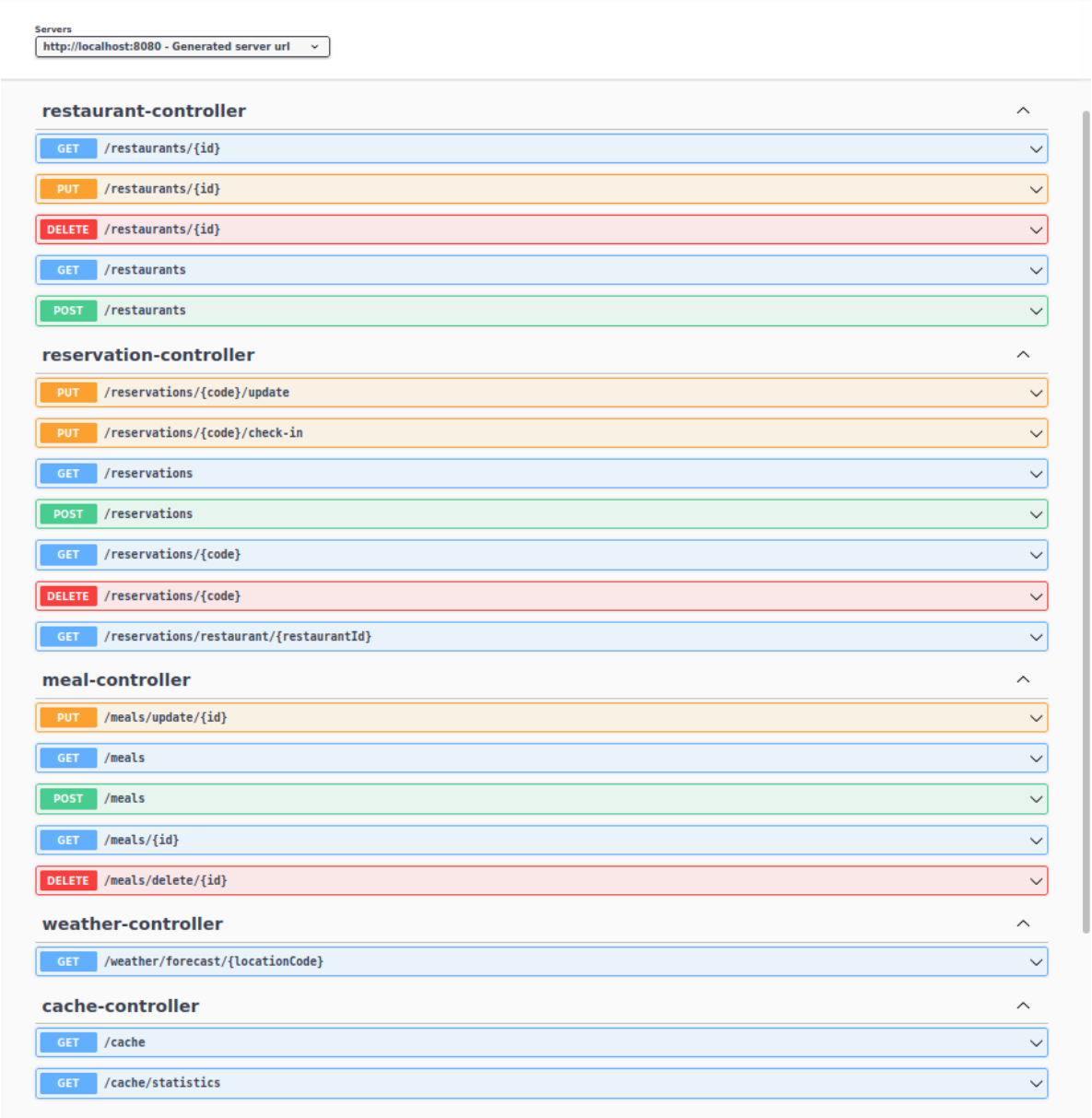

Fig.2 Endpoints (Swagger view)

restaurant-controller

k) GET /restaurants/{id} – Retrieves a specific restaurant by its ID.
l) PUT /restaurants/{id} – Updates an existing restaurant identified by its ID.
m) DELETE /restaurants/{id} – Deletes a restaurant with the given ID.
n) GET /restaurants – Retrieves a list of all restaurants.
o) POST /restaurants – Creates a new restaurant.

reservation-controller

p) PUT /reservations/{code}/update – Updates an existing reservation identified by its code.
q) PUT /reservations/{code}/check-in – Performs a check-in for a reservation using its code.

r) GET /reservations – Retrieves a list of all reservations.
s) POST /reservations – Creates a new reservation.
t) GET /reservations/{code} – Retrieves a specific reservation by its code.
u) DELETE /reservations/{code} – Cancels a reservation with the given code.
v) GET /reservations/restaurant/{restaurantId} – Retrieves all reservations for a specific restaurant identified by its ID.

meal-controller

w) PUT /meals/update/{id} – Updates an existing meal identified by its ID.
x) GET /meals – Retrieves a list of all meals.
y) POST /meals – Creates a new meal.
z) GET /meals/{id} – Retrieves a specific meal by its ID.
aa) DELETE /meals/delete/{id} – Deletes a meal with the given ID.

weather-controller

bb) GET /weather/forecast/{locationCode} – Retrieves the weather forecast for a specific location identified by its code.

cache-controller

cc) GET /cache – Retrieves the cached data.
dd) GET /cache/statistics – Retrieves statistics about the cache.

3 Quality assurance

3.1 Overall strategy for testing

Initially, I adopted a Test-Driven Development (TDD) approach. However, due to time constraints, I gradually shifted away from it. Instead, I focused on implementing the features I anticipated needing and performed the necessary testing afterward.
For backend testing, I utilized a variety of tools, including JUnit 5, Spring Boot's MockMvc, Mockito, and TestContainers. Flyway was also used to populate the database during tests. On the frontend side, I chose to use Selenium IDE for testing. While more advanced options like Selenium WebDriver or Cucumber were available, I opted for Selenium IDE to save time and streamline the process.

3.2 Unit and integration testing

Unit tests were written to verify cache behavior and to test the service layer. Integration tests were implemented at the controller level using TestContainers to simulate real-world scenarios.
Later, while reviewing the code with SonarQube, I noticed a discrepancy: despite seeing high test coverage in VSCode, Sonar reported only around 40% coverage, with no coverage detected for the controller layer. To address this, I added unit tests specifically for the controllers to improve overall coverage and accuracy.

```java
public class InMemoryCacheTest {

    private InMemoryCache<String, String> cache;

    @BeforeEach
    public void setUp() {
        // Initialize the cache before each test
        cache = new InMemoryCache<>();
    }

    @Test
    public void testPutAndGet() {
        cache.put(key:"key1", value:"value1");
        String value = cache.get(key:"key1");
        assertNotNull(value, message:"The value should not be null.");
        assertEquals(expected:"value1", value, message:"The value should be 'value1'.");
    }
```

Fig.3 Unit Test - Cache

```java
@ExtendWith(MockitoExtension.class)
class RestaurantServiceTest {

    @Mock
    private RestaurantRepository restaurantRepository;

    @Mock
    private ReservationRepository reservationRepository;

    @Mock
    private MealRepository mealRepository;

    @InjectMocks
    private RestaurantService restaurantService;

    @Test
    void whenGetAllRestaurants_thenReturnListOfRestaurants() {
        Restaurant r1 = new Restaurant(name:"Sushi House", location:"Lisbon", locationCode:1010500, capacity:50, opening…"10:00", "23:00");
        Restaurant r2 = new Restaurant(name:"Pasta Palace", location:"Porto", locationCode:4000000, capacity:40, opening…"11:00", "22:00");
        when(restaurantRepository.findAll()).thenReturn(List.of(r1, r2));
                                                    List<Restaurant> org.tqs.deti.ua.homework.service.RestaurantService.getAllRestaurants
        List<Restaurant> restaurants = restaurantService.getAllRestaurants();

        assertEquals(expected:2, restaurants.size());
        verify(restaurantRepository, times(wantedNumberOfInvocations:1)).findAll();
    }
```

Fig.4 Unit Test - Services

```java
@SpringBootTest
@AutoConfigureMockMvc
@ExtendWith(SpringExtension.class)
@Testcontainers
@TestMethodOrder(OrderAnnotation.class)
class RestaurantControllerIT {

    @Container
    static PostgreSQLContainer<?> postgresContainer = new PostgreSQLContainer<>(dockerImageName:"postgres:15")
            .withDatabaseName(databaseName:"testdb")
            .withUsername(username:"testuser")
            .withPassword(password:"testpass");

    @DynamicPropertySource
    static void overrideProps(DynamicPropertyRegistry registry) {
        registry.add(name:"spring.datasource.url", postgresContainer::getJdbcUrl);
        registry.add(name:"spring.datasource.username", postgresContainer::getUsername);
        registry.add(name:"spring.datasource.password", postgresContainer::getPassword);
        registry.add(name:"spring.flyway.enabled", () -> "true");
    }

    @Autowired
    private MockMvc mockMvc;

    @Test
    @Order(1)
    void testGetAllRestaurants() throws Exception {
        mockMvc.perform(MockMvcRequestBuilders.get(uriTemplate:"/restaurants"))
                .andExpect(status().isOk())
                .andExpect(jsonPath(expression:"$", not(empty())))
                .andExpect(jsonPath(expression:"$[0].name").value(expectedValue:"Campus Cafeteria"))
                .andDo(print());
    }
```

Fig.5 Integration Test - Controllers

3.3 Functional testing

As mentioned earlier, the only functional testing performed was using Selenium IDE.

1	✓ *open*	/web/restaurants	
2	✓ *set window size*	1858x1053	
3	✓ *assert title*	Restaurants	
4	✓ *click*	linkText=Guimarães Tavern	
5	✓ *assert title*	Make a Reservation	
6	✓ *assert text*	xpath=//h2/span	Guimarães Tavern
7	✓ *mouse down at*	css=.meal-scroll-container	681.61669921875,285.28334045410156
8	✓ *mouse move at*	css=.meal-scroll-container	681.61669921875,285.28334045410156
9	✓ *mouse up at*	css=.meal-scroll-container	681.61669921875,285.28334045410156
10	✓ *mouse down at*	css=.meal-scroll-container	595.61669921875,267.28334045410156
11	✓ *mouse move at*	css=.meal-scroll-container	595.61669921875,267.28334045410156
12	✓ *mouse up at*	css=.meal-scroll-container	595.61669921875,267.28334045410156
13	✓ *click*	id=meal-21	
14	✓ *mouse down at*	name=quantities[21]	-8.04998779296875,16.883331298828125
15	✓ *mouse move at*	name=quantities[21]	-8.04998779296875,16.883331298828125

Fig.6 Selenium IDE

After completing the main features, I'll see if there's enough time before the deadline to implement additional tests using Selenium WebDriver, if i have time to properly implemented and can't add the proper documentation to this report i will talk about it in the oral presentation.

3.4 Non functional testing

Using the K6 workshop from Lab 7, I was able to perform load testing on few key checkpoints.

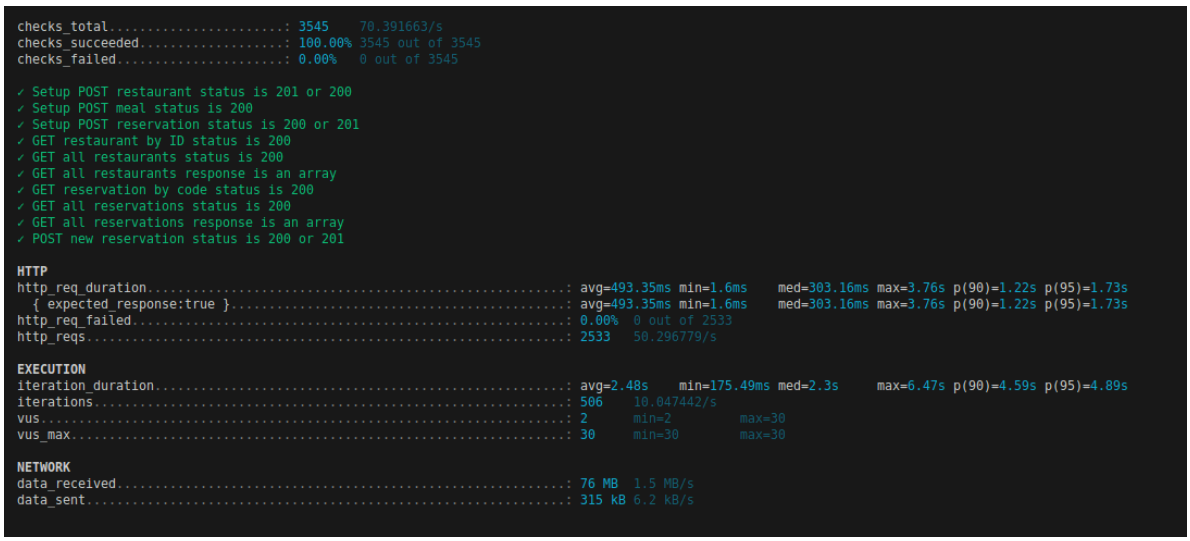

Fig.6 Load Tests

3.5 Code quality analysis

I used the SonarQube container from Lab 8 for code analysis. It proved challenging, as it didn't capture coverage from the integration tests, as mentioned earlier. Additionally, it flagged several issues—some of which, in my opinion, weren't issues. To move forward with the analysis, I chose to mark those issues as "Accepted" and continue with the process.

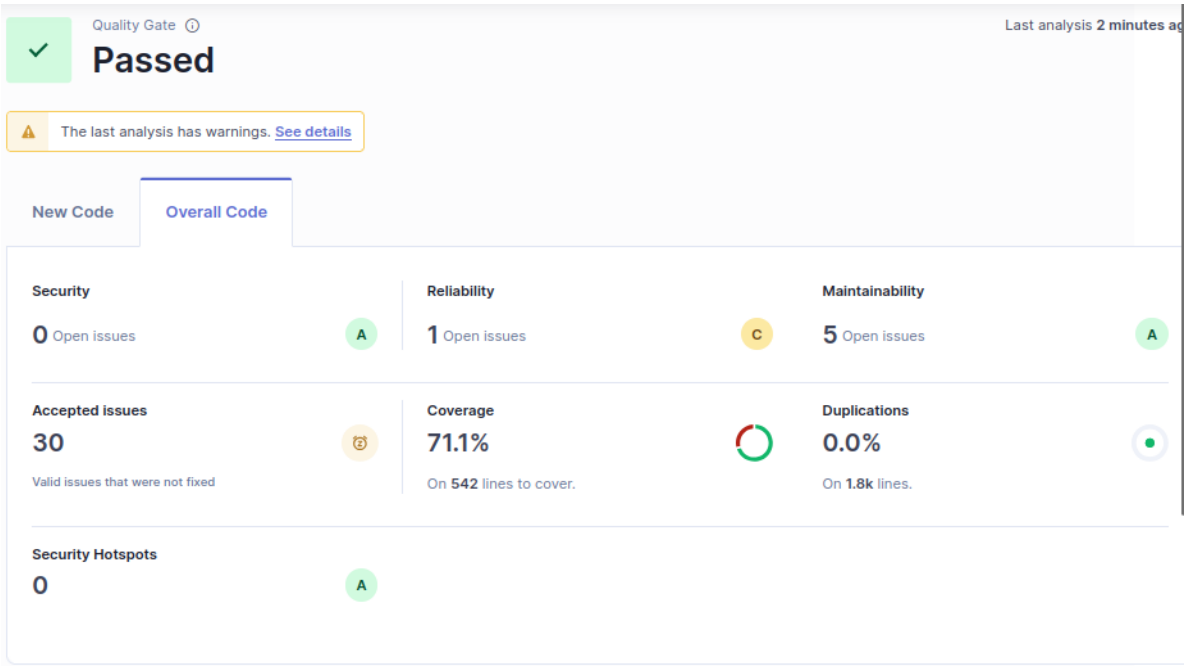

Fig.7 SonarQube Analysis

4 References & resources

Project resources

Resource:	URL/location:

Git repository	https://github.com/FunnyJoaoneto/TQS_113482
Video demo	https://youtu.be/u8nA8Ndqado
QA dashboard (online)	Not used
CI/CD pipeline	Not used
Deployment ready to use	Not used

Reference materials

ee) Weather forecast api

ff) https://api.ipma.pt/open-data/forecast/meteorology/cities/daily/