HW1: Mid-term assignment report

Filipe Miguel Neto Viseu [119192], v2025-11-05

1 Introduction 1
1.1 Overview of the work 1
1.2 Current implementation (faults & extras) 1

2 Product specification 2
2.1 Functional scope and supported interactions 2
2.2 System implementation architecture 3
2.3 API for developers 4

3 Quality assurance 4
3.1 Overall strategy for testing 4
3.2 Unit and integration testing 5
3.3 Acceptance testing 7
3.4 Non-functional testing 7
3.5 Code quality analysis 8
3.6 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.
ZeroMonos is a waste collection management system designed to simplify the scheduling and management of waste collection across Portuguese municipalities (Aveiro does something similar [“Retomado o serviço de recolha de volumosos e resíduos verdes”](#).
My application enables citizens to book waste collection appointments while providing staff with tools to monitor, filter and update booking status.

1.2 Current implementation (faults & extras)

Faults/Limitations:

- No authentication
- The CORS is set to [*] (couldn’t be if this went to production)

Extras:

- Modern and responsive UI
- Assign work to employees
- Operations dashbord

1.3 Use of generative AI

The use of AI in my work was mostly to make some of the frontend stuff.
I made the structure of the HTML code but I asked AI to make it prettier.
Also, in some tests I used AI to help me (for example in WebFunctionalTest where I was having trouble to put the date and I asked AI that gave me the option of “Javascript Executor”).
Besides that, the 100 tests I have weren’t all made by hand, I made 50 tests and asked AI to help me make the other 50 (Employee and WorkTask tests). This was very helpful but it wasn’t perfect, AI made a lot of mistakes that had to be corrected by me.

2 Product specification

2.1 Functional scope and supported interactions

Citizens:

- Create bookings
- Track booking status with token
- View booking history
- Cancel existing bookings

Staff:

- View all bookings
- Filter bookings by municipality and by status
- Update booking status

Employee:

- 4 functions: DRIVER, COLLECTOR, SUPERVISOR, ADMIN
- Can be assigned to a booking

Main interactions (Citizen):

- Access citizen portal from the “index” page
- Fill booking form (description, municipality and date)
- Receive token
- Track booking with token (In “My Bookings” tab)
- View status update and cancel booking if needed

Main interactions (Staff):

- Access staff portal from the “index” page
- Load and filter bookings by municipality or by status
- Update booking status
- Add employees to DB
- Assign booking to employees

2.2 System implementation architecture

My system follows a layered architecture:
It consists of a Maven project using Spring Boot named **backend** where all the backend stuff is and a folder named **frontend** with HTML, JS and CSS code.
Inside backend I chose to create 3 folders (representing layers):

- **Boundary** - contains the **BookingController** that exposes REST endpoints for operations & **WorkTaskController** & **EmployeeController** (same logic)
- **Service** - contains the **BookingService** that implements the business logic (functions used by BookingController endpoints) & **WorkTaskService** (same logic)
- **Data** - contains the **BookingRepository** (that extends JPARepository), **BookingRequest** an Entity mapped to the DB using JPA and **BookingHistoryRepository** & **BookingHistoryRequest** to keep track of states
  WorkTaskRepository & **WorkRequest** & **EmployeeRepository** & **EmployeeRequest**

In general, I used

- Spring Boot 3.5.6
- Spring Data JPA
- REST Assured 5.4.0
- PostgresSQL
- HTML, CSS, JS
- K6

2.3 API for developers

Endpoints	Methods	Description
/api/bookings	POST	Create new booking
/api/bookings	GET	Get all bookings
/api/bookings/{token}	GET	Get specific booking
/api/bookings/{token}/cancel	PUT	Update status to canceled
/api/bookings/{token}/status	PUT	Update booking status
/api/bookings/{token}/history	GET	Get booking status history

Example (Create Booking):

```
POST /api/bookings
{
  "description": "Large furniture disposal",
  "municipality": "Aveiro",
  "date": "2025-12-25T10:00:00"
}
```

3 Quality assurance

3.1 Overall strategy for testing

Multi-layered strategy.

Unit Tests

- **Tools**: JUnit 5, Mockito
- **Focus**: Testing business logic in isolation
- **Approach**:
    - Mocked repository dependencies to test service layer logic independently
    - Validated core business rules (e.g., booking limits, status transitions)
    - Fast execution, no external dependencies

Integration Tests

- **Tools**: REST-Assured, Spring Boot Test, JdbcTemplate
- **Focus**: Testing API endpoints with real database interactions
- **Approach:**
    - REST-Assured for clean, readable HTTP request/response validation
    - Database cleanup using JdbcTemplate
    - Verified complete request-response cycles including data persistence

Controller Tests

- **Tools:** MockMvc , MockitoBean
- **Focus & Approach:** Test controllers mocking services layers

Functional E2E Tests

- **Tools:** Selenium WebDriver
- **Focus:** Test complete workflows
- **Approach:** Simulated real user interactions, testing all system

Performance Tests

- **Tools:** K6
- **Focus:** Validate system behavior
- **Approach:** LoadTests, SpikeTests, StressTests

3.2 Unit and integration testing

Unit Tests were used on BookingServiceTest & WorkServiceTest:

- Tests business logic in isolation
- Uses Mockito to mock the repositorys
- Validates booking creation, token generation, booking limits, status history checking

```java
@Test
    void testCreateBookingThrowsExceptionWhenLimitReached() {
        LocalDateTime date = LocalDateTime.of(2025, 11, 1, 10, 0);
        List<BookingRequest> existingBookings = List.of(
            new BookingRequest("r1", "Aveiro", date, "T1"),
            new BookingRequest("r2", "Aveiro", date, "T2"),
            new BookingRequest("r3", "Aveiro", date, "T3"),
            new BookingRequest("r4", "Aveiro", date, "T4"),
            new BookingRequest("r5", "Aveiro", date, "T5")
        );
        when(repository.findByMunicipality("Aveiro")).thenReturn(existingBookings);
        assertThatThrownBy(() -> service.createBooking("room", "Aveiro", date))
            .isInstanceOf(IllegalStateException.class)
            .hasMessage("Booking limit reached for this municipality");
    }
```

Controller Tests were used on BookingControllerTest & EmployeeControlerTest & WorkControllerTest :

- Uses WebMvcTest and MockMvc
- Tests Endpoints, request validations and response formats
- Validate status codes (200, 400, etc.)

```java
@Test
    void givenManyBookings_whenGetBookings_thenReturnJsonArray() throws Exception {
        BookingRequest b1 = new BookingRequest("room", "Aveiro", LocalDateTime.of(2024, 6, 10, 10, 0), "T1");
        BookingRequest b2 = new BookingRequest("car", "Porto", LocalDateTime.of(2024, 6, 11, 11, 0), "T2");
        List<BookingRequest> allBookings = Arrays.asList(b1, b2);

        when(bookingService.getAllBookings()).thenReturn(allBookings);

        mvc.perform(
                get("/api/bookings").contentType(MediaType.APPLICATION_JSON)
        )
                .andExpect(status().isOk())
                .andExpect(jsonPath("$", hasSize(2)))
                .andExpect(jsonPath("$[0].description", is("room")))
                .andExpect(jsonPath("$[1].description", is("car")));

        verify(bookingService, times(1)).getAllBookings();
    }
```

Integration Tests were used on BookingControllerIT, EmployeeControllerIT, WorkControllerIT:

- Uses Rest Assured for full API testing
- Tests actual HTTP requests (no mock)
- Validates the complete “lifecycle”

```java
@Test
    void testCreateAndGetBooking() {
        // create booking
        String token =
            RestAssured.given()
                .contentType(ContentType.JSON)
                .body("{\"description\":\"Large furniture\",\"municipality\":\"Aveiro\",\"date\":\"2025-06-10T10:00:00\"}")
            .when()
                .post("/api/bookings")
            .then()
                .statusCode(201)
                .body("description", equalTo("Large furniture"))
                .body("municipality", equalTo("Aveiro"))
                .body("date", equalTo("2025-06-10T10:00:00"))
                .body("status", equalTo("RECEIVED"))
                .body("token", notNullValue())
                .extract()
                .path("token");

        // get booking by token
        RestAssured.given()
            .when()
                .get("/api/bookings/" + token)
            .then()
                .statusCode(200)
                .body("description", equalTo("Large furniture"))
                .body("municipality", equalTo("Aveiro"))
                .body("token", equalTo(token));
    }
```

3.3 Acceptance testing

Functional Tests were implemented in WebFunctionalTest:

- Implemented using Selenium WebDriver with @ExtendWith(SeleniumJupiter.class)
- Automatic driver management

Test Cases:

- testCreateBookingAsCititzen: Validates de booking creation workflow
- testSearchBooking: Tests the search functionality
- testStaffPortalAcess: Validates the staff interface and bookings loading
- testRoleSwitching: Switch between user and staff
- testMunicipalityFilter: Test the filtering option in staff

```
@Test
    void testStaffPortalAccess(ChromeDriver driver) {
        driver.get("http://localhost:8080/index.html");

        WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));

        WebElement staffBtn = wait.until(ExpectedConditions.elementToBeClickable(
            By.cssSelector("button.menu-btn.staff")
        ));
        staffBtn.click();
        wait.until(ExpectedConditions.presenceOfElementLocated(By.id("loadBtn")));
        assertThat(driver.getTitle()).contains("Staff Portal");

        WebElement loadBtn = driver.findElement(By.id("loadBtn"));
        loadBtn.click();

        wait.until(ExpectedConditions.presenceOfElementLocated(By.id("staffResult")));
        WebElement staffResult = driver.findElement(By.id("staffResult"));

        assertThat(staffResult.getText()).isNotEmpty();
    }
```

3.4 Non-functional testing

I tested the performance of my system using K6 (studied in TP classes)
LoadTest:

- Gradual ramp-up to 20 users
- Sustain 20 users for 30 seconds
- Test operations
- Thresholds

SpikeTest:

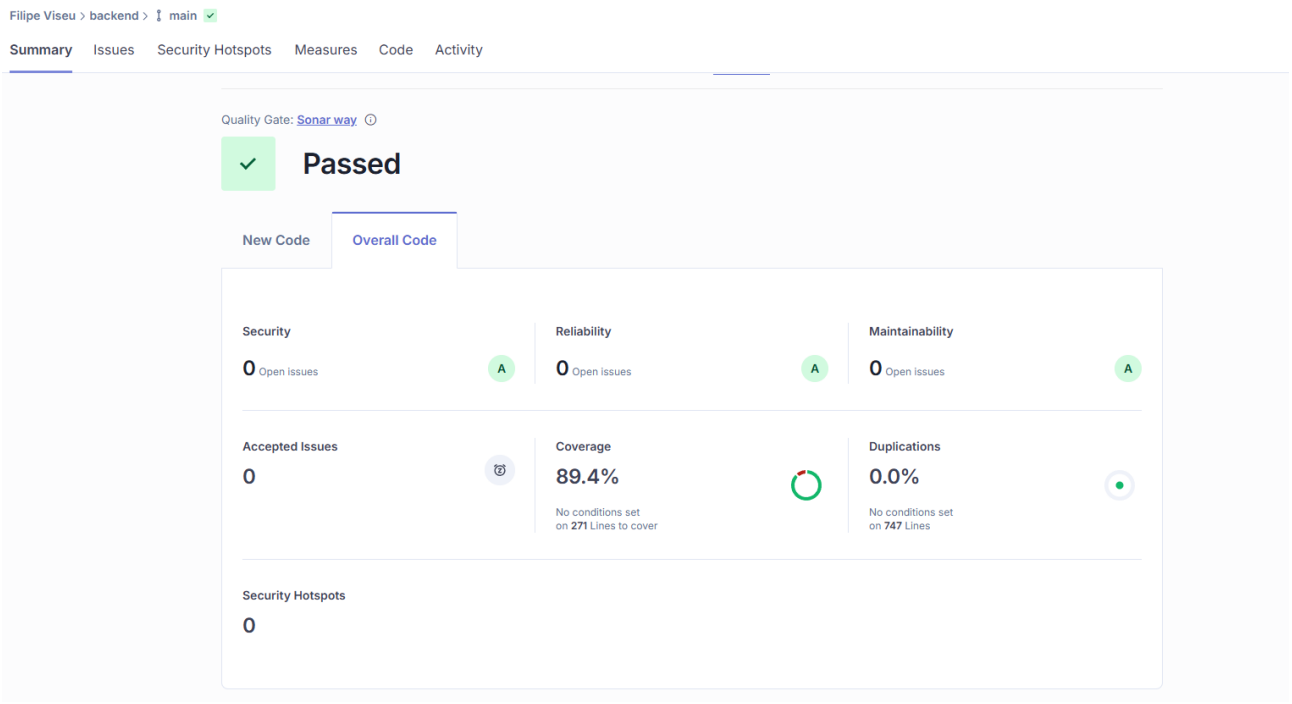
- Sudden spike from 10 to 100 users
- Test the behavior with that amount of users

StressTest:

- Incremental load ; Breaking point ; System stability

3.5 Code quality analysis

I used sonarqube, local and web (with a public github project).
In this analysis I saw problems like CORS for example that in this case it isn't a real problem but if this application goes to production, then it turns a problem.
Other problems were security ones in logs where user input were being logged directly, creating a potential log injection vulnerability.



3.6 Continuous integration pipeline [optional]

I used github actions and sonarqube.

1. **Automated Testing**
   - Runs on every push to `main` branch and pull requests
   - Executes unit tests, integration tests (Selenium tests excluded in CI)
   - PostgreSQL service container for database-dependent tests

2. **Code Quality Analysis**
   - SonarCloud integration for code quality metrics
   - Code coverage tracking with JaCoCo
   - Automated quality gate checks

4 References & resources

Project resources

Resource:	URL/location:
Video demo	docs/ZeroMonosDemo
QA dashboard (online)	https://sonarcloud.io/summary/overall?id=FilipeNV1_ZeroMonos119192&branch=main
CI/CD pipeline	https://github.com/FilipeNV1/ZeroMonos119192/actions

Reference materials

Geo API: https://geoapi.pt/docs/
REST Assured: https://www.geeksforgeeks.org/software-testing/how-to-test-api-with-rest-assured/