HW1: Mid-term assignment report

Victor Milhomem Losada Lopez [128660], v2025-03-26

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.
Consisting of a full-stack web application for managing meal reservations at the Moliceiro University campus. The system allows users (students) to view available meals at different restaurants, make reservations, and access weather forecasts to help decide where to eat. Restaurant staff can validate and mark reservations as used. The backend was developed using Spring Boot with Thymeleaf for the frontend and H2 for in-memory database testing.

1.2 Current limitations

- No user authentication was implemented (not required by the assignment).
- No group bookings or meal-specific availability limits.
- Weather forecast only shows summary and temperature; no icons or extended forecast.

2 Product specification

2.1 Functional scope and supported interactions

The application supports the following actors:

- **Students**: can view meals per restaurant, check the weather, and book a reservation.
- **Staff**: can check the validity of a reservation using the provided token and mark it as used.

Main user interactions:

- Select a restaurant and view meals for upcoming days.
- View weather forecast for each meal.
- Book a meal and receive a reservation token.
- Enter a token on the check-in screen to validate attendance.

2.2 System implementation architecture

- **Backend**: Spring Boot + Spring Data JPA
- **Database**: H2 in-memory (for tests); supports PostgreSQL/MySQL for production
- **Frontend**: Thymeleaf (HTML templates), CSS for styling
- **External API**: OpenWeather for 5-day forecast
- **Cache**: In-memory cache with TTL (2h) for weather forecasts

2.3 API for developers

Meals & Reservations:

- GET /api/meals?restaurant=Rest A – List meals for a restaurant
- POST /api/reservations – Book a meal reservation
- GET /api/reservations/{token} – Get reservation details
- DELETE /api/reservations/{token} – Cancel reservation
- POST /api/reservations/{token}/check-in – Mark reservation as used

Weather:

- GET /api/weather?location=Aveiro&date=2025-04-10 – Get weather forecast for location/date
- GET /api/weather/cache-stats – View cache hits/misses

3 Quality assurance

3.1 Overall strategy for testing

We adopted a TDD-inspired approach. Business logic in services was tested via unit tests, while endpoints were tested using Spring's MockMvc for integration. Test data was loaded via @Sql for repository tests.

3.2 Unit and integration testing

The unit and integration testing was used in every service, controller and repository from the application.

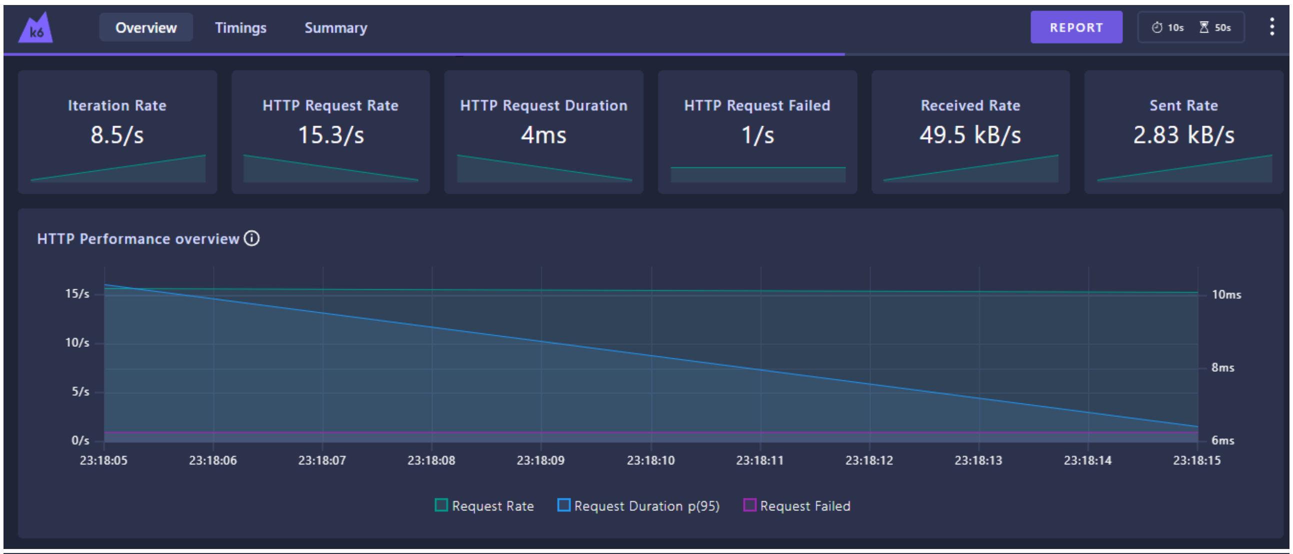
```
@WebMvcTest(RestaurantController.class)
public class RestaurantControllerTest {
   @Autowired
   private MockMvc mockMvc;

   @MockBean
   private RestaurantRepository restaurantRepository;

   @Test
   void getAllRestaurants_ReturnsRestaurants() throws Exception {
      // Mock repository response
      List<Restaurant> mockRestaurants = List.of(
            new Restaurant(1L, "Campus Cafe", "Building A"),
            new Restaurant(2L, "University Diner", "Building B")
      );
      when(restaurantRepository.findAll()).thenReturn(mockRestaurants);

      mockMvc.perform(get("/api/restaurants"))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$[0].name").value("Campus Cafe"))
            .andExpect(jsonPath("$[1].location").value("Building B"));

      verify(restaurantRepository, times(1)).findAll();
   }
}
```

3.3 Functional testing

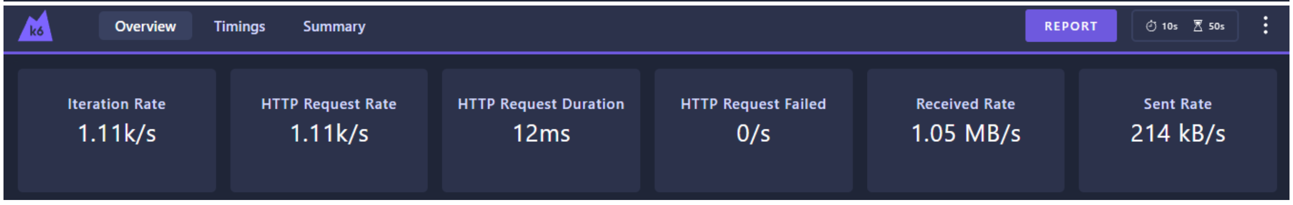
For the functional testing we used the Selenium Web IDE, for the booking and check-in working flows.

3.4 Non functional testing

For the Non functional testing we used the K6 tool. Simulating concurrent requests for meal reservations and check-ins.



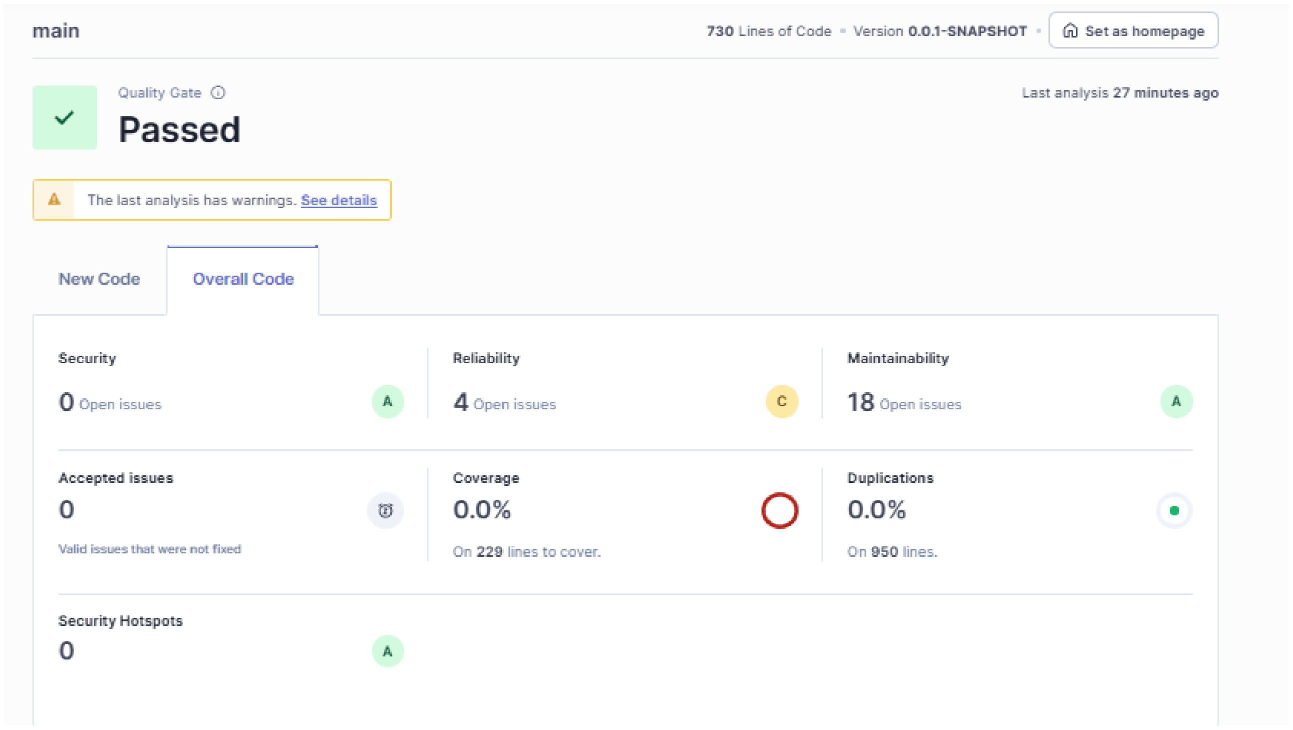
Iteration Rate	HTTP Request Rate	HTTP Request Duration	HTTP Request Failed	Received Rate	Sent Rate
8.5/s	15.3/s	4ms	1/s	49.5 kB/s	2.83 kB/s



Iteration Rate	HTTP Request Rate	HTTP Request Duration	HTTP Request Failed	Received Rate	Sent Rate
1.11k/s	1.11k/s	12ms	0/s	1.05 MB/s	214 kB/s

3.5 Code quality analysis

Code was analyzed using **SonarQube** for code quality metrics and **Lighthouse** for basic frontend performance and accessibility audits.



main

730 Lines of Code · Version 0.0.1-SNAPSHOT

Quality Gate

Passed

Last analysis 27 minutes ago

The last analysis has warnings. See details

Metric	Value	Detail	Rating
Security	0	Open issues	A
Reliability	4	Open issues	C
Maintainability	18	Open issues	A
Accepted issues	0	Valid issues that were not fixed	
Coverage	0.0%	On 229 lines to cover.	
Duplications	0.0%	On 950 lines.	
Security Hotspots	0		A

3.6 Continuous integration pipeline [optional]

No CI/CD pipeline was defined.

4 References & resources

Project resources

Resource:	**URL/location:**
Git repository	TQS_128660/HW1-128660 at main · VictorMilhomem/TQS_128660
Video demo	The Video Demo is included in the docs folder
QA dashboard (online)	[**optional**; if you have a quality dashboard available online (e.g.: sonarcloud), place the URL here]
CI/CD pipeline	[**optional**; if you have th CI pipeline definition in a server, place the URL here]
Deployment ready to use	[**optional**; if you have the solution deployed and running in a server, place the URL here]

Reference materials

- OpenWeather API docs
- Spring Boot documentation
- Baeldung tutorials on MockMvc, caching, and Thymeleaf
- H2 documentation for in-memory SQL use