deti universidade de aveiro
departamento de eletrónica, telecomunicações e informática

HW1: Mid-term assignment report

Francisco José Gomes Pinto [113763], v2025-04-09

1 Introduction

1.1 Overview of the work

This project is a restaurant reservation system that allows users to book meals at specific restaurants while also considering the weather conditions expected at the time of the meal. The main goal was to offer users a more informed and convenient reservation experience, combining basic restaurant booking features with real-time weather forecasting.While storing the weather in a cache system to minimize calls to external API.

1.2 Current limitations

The main features work as expected, but there are some limitations, particularly on the frontend, which was the last part I developed and remains quite simplistic. Currently, the creation of new restaurants and meals is only possible through Swagger, as a proper user interface for this functionality was not implemented. I also intended to improve the reservation viewing system by allowing clients to search using their email address, but this feature was not completed. Additionally, I encountered issues with SonarQube and was

unable to run it locally, so I couldn't extract the code coverage metrics. The CI/CD pipeline was also not implemented.

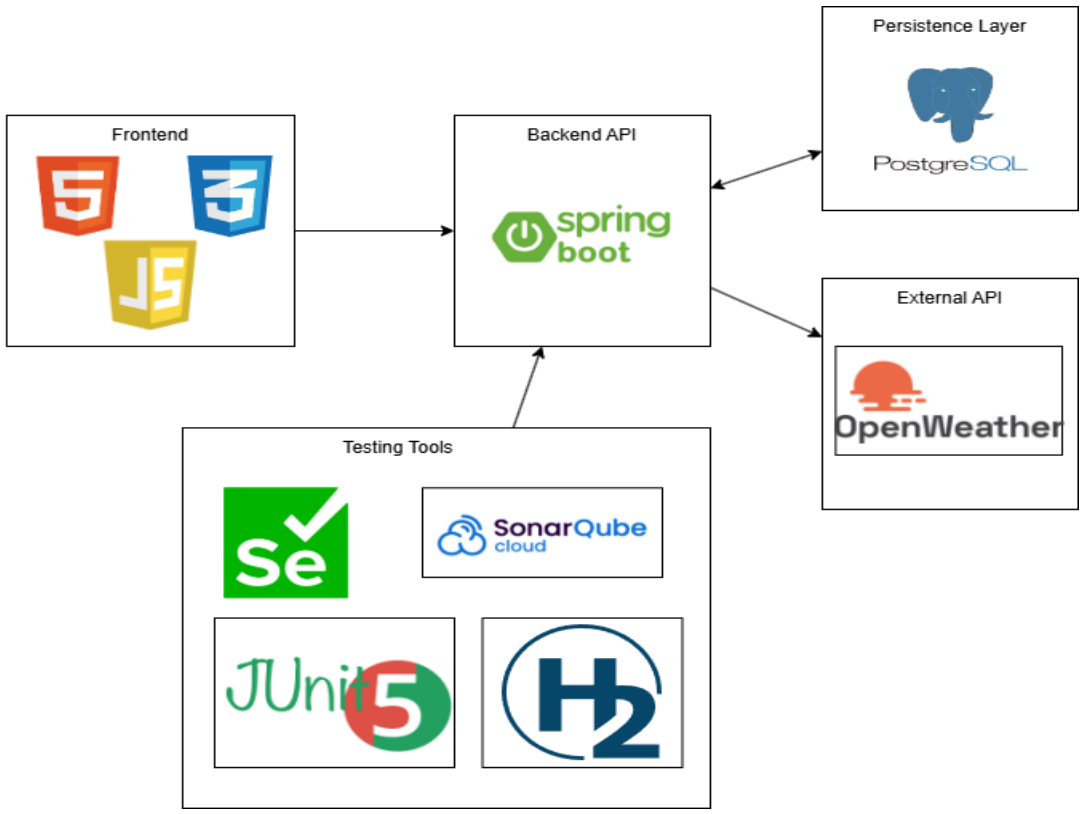
2 Product specification

2.1 Functional scope and supported interactions

The primary user of the system is someone who wants to make a reservation for a meal at a specific restaurant, while also being informed about the weather conditions expected at the time of the meal. After selecting the desired meal, the user receives a unique code that allows them to track the reservation and cancel it if needed, as long as it hasn't been confirmed yet. Each meal has a limited capacity, and once the number of non-canceled reservations reaches this limit, no further reservations can be made.

There is also a secondary user: the staff member. A staff member can confirm a reservation by entering the reservation code, but only if the reservation is still pending and has not yet been confirmed.

2.2 System implementation architecture



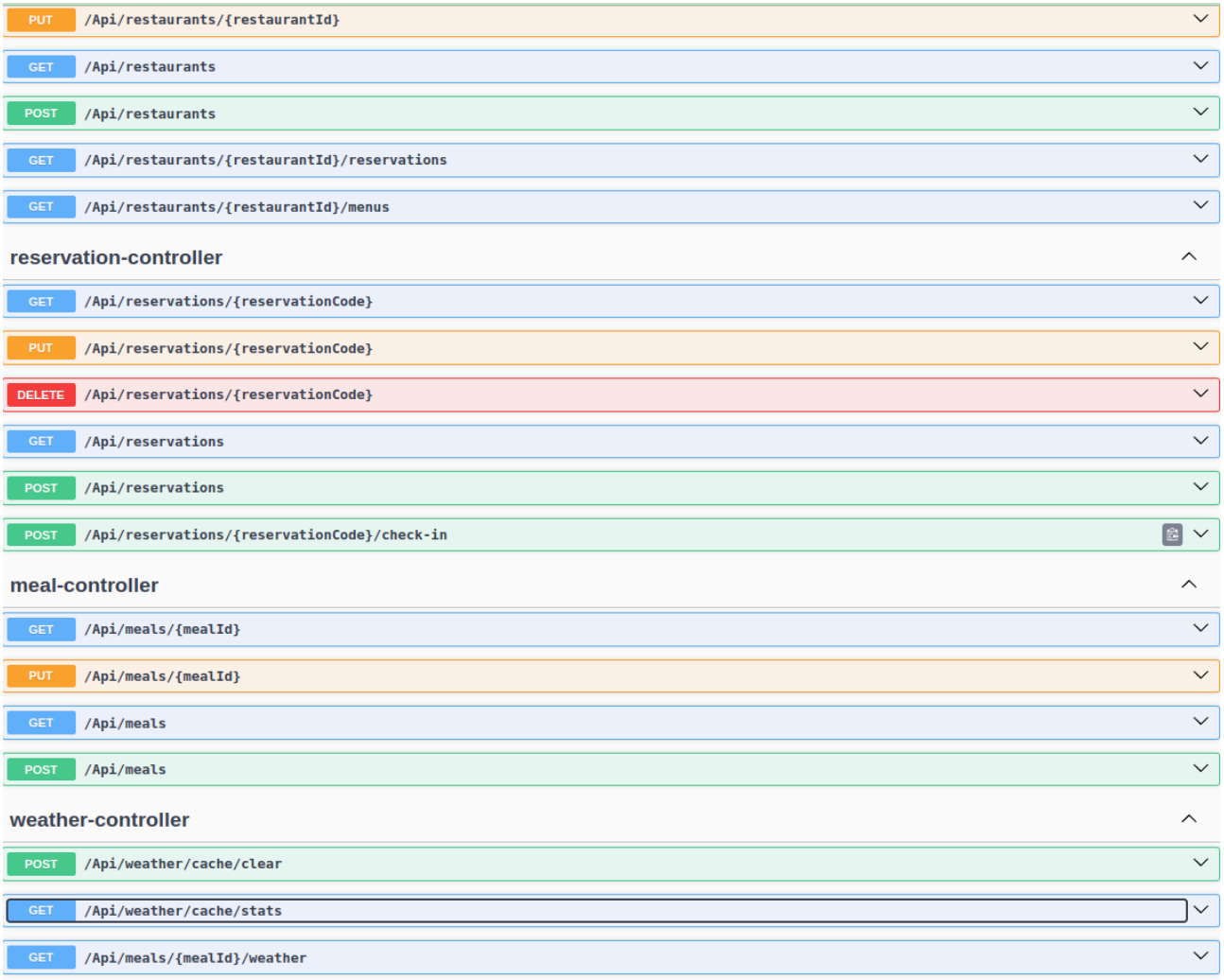
To provide weather information, I integrated the OpenWeatherAPI, which retrieves weather conditions for the location of the restaurant and the specific time the meal will take place. To minimize the number of external API calls—especially due to usage limitations—the weather data is cached and refreshed every 30 minutes.

The backend was developed using Spring Boot, and the persistent database was implemented with PostgreSQL. For testing, I used a variety of tools to cover different aspects of the system, including unit, integration, and frontend testing.

The frontend was developed using basic HTML, JavaScript, and CSS, with a focus on simplicity and core functionality.

2.3 API for developers

The Endpoints were separated into 4 groups, Meals, Restaurants, Reservations and Cache. These endpoints offer ways to interact with the system, but not all were used in the front-end, somewhere only made for testing purposes.



3 Quality assurance

3.1 Overall strategy for testing

For this project, I adopted a hybrid testing strategy that combined elements of both Test-Driven Development (TDD) and Behavior-Driven Development (BDD) to ensure code quality and reliability throughout the development process.

TDD was primarily applied to the backend logic, where I wrote unit tests before implementing the actual code. This approach helped enforce a modular design and facilitated early bug detection.

BDD principles guided the definition of user-facing behavior, making sure that the system fulfilled the intended business requirements from the user's perspective.

Tools and Technologies Used:
JUnit – for unit testing the business logic and service layers of the backend
Selenium IDE – for basic end-to-end testing of the frontend functionality
Mockito – for mocking dependencies and isolating components during unit and integration testing
H2 (In-memory database) – used for fast, isolated testing of data persistence logic
k6 – for load and performance testing, simulating concurrent users and assessing backend performance
Lighthouse – for auditing frontend performance, accessibility, and best practices

SonarQube – for static code analysis and quality assurance .

This multi-layered testing strategy helped validate the system's correctness, performance, and maintainability.

3.2 Unit and integration testing

To ensure code reliability and maintainability, I implemented both unit and integration testing throughout development. Unit tests validated individual components and core logic in isolation, ensuring that each function behaved as expected. Integration tests verified the correct interaction between modules, particularly where the system communicated with external APIs or the database, ensuring overall system coherence.

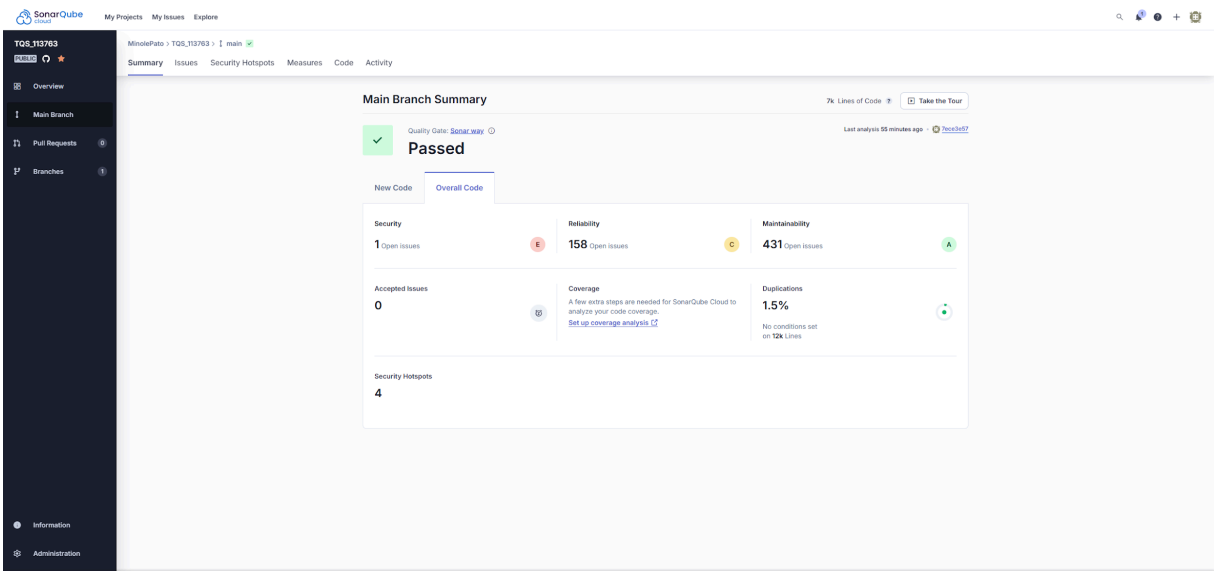
3.3 Functional testing

I used Selenium to test the workflows that the users have to do to accomplish their tasks.

3.4 Non functional testing

I Used lighthouse to test the front-end part of the project, the score is very high because the web-site is very small and directed to the use cases. Also used K6 for performance in the communication of the API.

3.5 Code quality analysis



3.6 Continuous integration pipeline [optional]

Did not do.

4 References & resources

Project resources

Resource:	URL/location:
Git repository	https://github.com/MinolePato/TQS_113763/tree/main/HW1-113763
Video demo	https://github.com/MinolePato/TQS_113763/blob/main/HW1-113763/prints/Screencast%20from%2009-04-2025%2021_33_41.webm
QA dashboard (online)	https://sonarcloud.io/summary/overall?id=MinolePato_TQS_113763&branch=main

Reference materials

External API:https://openweathermap.org/