HW1: Mid-term assignment report

Rodrigo Abreu [113626], 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.
My application allows users to visualize the scheduled meals of the restaurants that are part of the Moliceiro University.
Users can check each Restaurant both lunch and dinner scheduled meals and the restaurant location weather forecast, for the current and following 4 days.
To get the weather forecast information, each restaurant as an associated weather ID for its respective location, that is used to make API calls to the IPMA weather API.
Users can book meals, that gives them a reservation code that they should note to later check its reservation details. For this, the app has a search page where the user can put its reservation code and search for its reservation. Users have the option to cancel their reservation at any time.
Restaurant workers can also check each meal reservations by clicking on the meal card button “Check Reservations” where they can check the Meal Information and view the reservations that were made for this meal. Reservations can be validated by restaurant workers by clicking on the reservation card button “Validate”. Reservations can only be validated on the meal date and if the reservation status is valid.

1.2 Current limitations

Some limitations of this app may be on the reservation area, where if a reservation is not validated on the meal day it does not change its status to expired, it just continues with the status as valid. Another is on having more than 2 meal options for a day, the system is designed to allow it, but it is not tested.
The system has no authentication nor access control, so there is no way to differentiate between a regular user of a restaurant worker. Therefore, each user can validate or cancel any reservation.
The system has no dynamic data insertion program. So, if it does not get periodically updated with meal info, it just gets no useful information. It is designed to be updated through the API, except for reservations.
The app is not deployed and only the database is containerized. Therefore, to run the app you must run the database docker compose up and then the spring boot project

2 Product specification

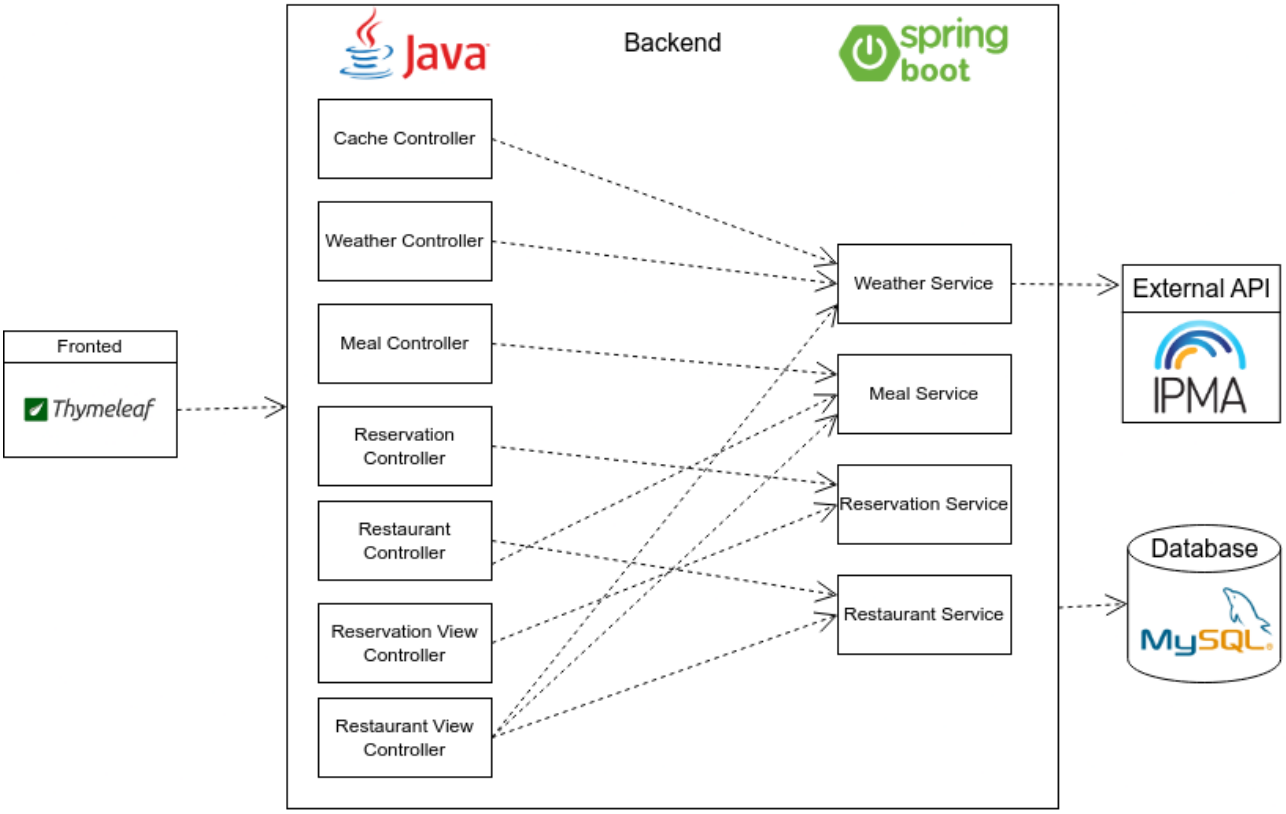
2.1 Functional scope and supported interactions

There are two main actors:

a) Clients: University students, professors or workers that have their meals in the university restaurants and want to keep informed about each restaurant meal and its respective weather forecast and schedules. Also to book reservations and manage it in a simple and efficient way.
b) Restaurant worker: A user that needs to manage daily meal reservations for the restaurant he works at.


View University Restaurants
View Restaurant Weather Forecast
View Restaurant Scheduled Meals
Manage Meal Reservations
<<extend>>
<<include>>
Client
Book a Meal Reservation
Restaurant Worker
Validate Meal Reservation
<<extend>>
Cancel Reservation
<<extend>>
VIew Reservation Details

2.2 System implementation architecture



Main technologies used:

c) Maven: For project structure and dependency management.

d) Spring Boot: Backend of the Web App allowed the creation of a REST API.

e) Thymeleaf: Technology used for the frontend of the webapp due to its easy integration with Spring Boot.

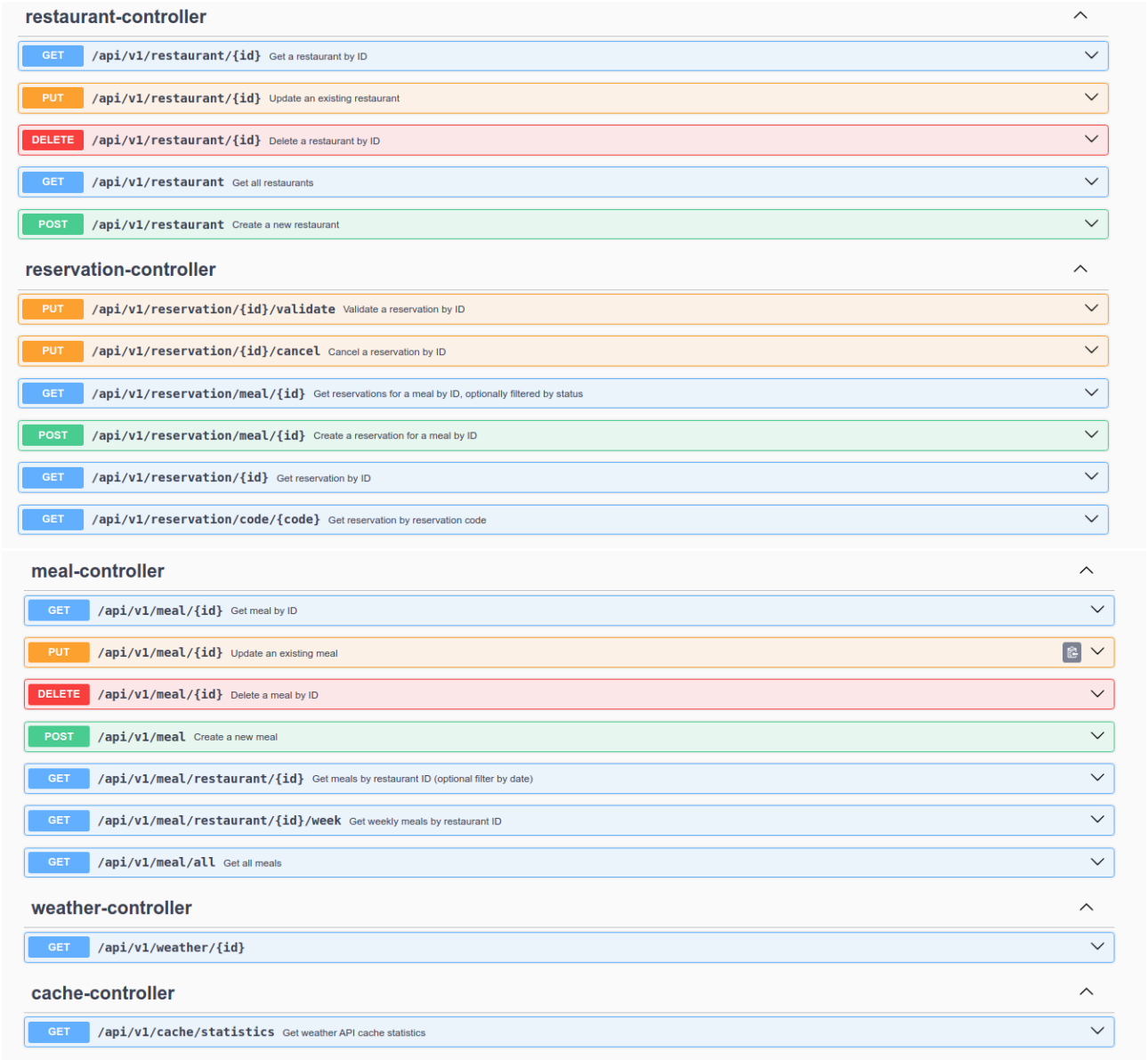
f) MySQL: This data persistence database was selected due to its seamless integration with Spring Boot.

g) IPMA API: Simple API to get weather forecast based on location Id. Abstracted the complexity of using real coords.

h) Docker: To containerize the MySQL database.

2.3 API for developers

I used Swagger to document my REST API, it has all the endpoints developers need to maintain the Web App updated. It also as a endpoint to check cache statistics.



3 Quality assurance

3.1 Overall strategy for testing

For the development of the small web application, I began by creating the backend components, which included entities, repositories, services, and controllers. To ensure reliability and correctness, I applied unit tests to each component, striving to achieve comprehensive coverage of their functionalities. Integration tests were also implemented for the weather API, resulting in successful validation of the API's integration.

Once the backend was functioning as expected, I proceeded to develop the frontend using Thymeleaf. During this stage, significant restructuring of the backend was required to ensure seamless interaction between the frontend and backend. This restructuring necessitated the refactoring of the backend tests, which turned out to be time-consuming and inefficient. However, after addressing these issues, I achieved stable backend testing results.

To validate the principal features of the web application from an end-user perspective, I applied Behavior-Driven Development (BDD) techniques using Cucumber and Selenium. These tools allowed me to test the application's core functionalities effectively.

Given more time, I would have adopted the Test-Driven Development (TDD) approach for the backend, as it emphasizes writing tests before code implementation, leading to better design and maintainability. However, due to time constraints, I opted for a different approach, prioritizing efficiency and meeting project deadlines.

3.2 Unit and integration testing

I started by implementing unit tests on all Entities, to ensure they were well structured, and its methods were working well. Then I used Mockito to make unitary tests on each service except for the Weather Service where I decided to apply an integration test, because I thought it is important to use the real API, instead if the mocked one.

3.3 Functional testing

I applied two functional tests on the principal features of the app:

i) View the scheduled meals for a restaurant.

j) Book a meal reservation.

I implemented this by describing the scenarios on the cucumber and combining it with the Selenium web driver.
Here is an example of the second one:

```
Feature: Book Meal Reservation

    Scenario: Book a meal reservation

      Given the user is on the Home page
      When the user clicks the button View Meals for "BurgerUA"
      And cliks the reservation button of a valid meal
      Then the user should get an alert with its reservation code
```

```java
@ExtendWith(SeleniumJupiter.class)
public class BookReservationSteps {
    WebDriver driver;

    @Given("the user is on the Home page")
    public void setup() {
        WebDriverManager.chromedriver().setup();
        driver = new ChromeDriver();
        driver.get(url:"http://localhost:8080/restaurants");
    }

    @When("the user clicks the button View Meals for {string}")
    public void clickViewMealsButtonForRestaurant(String restaurantName) {
        // Find all restaurant cards
        List<WebElement> restaurantCards = driver.findElements(By.cssSelector(cssSelector:".restaurant-card"));

        for (WebElement card : restaurantCards) {
            // Find the restaurant name inside the card
            WebElement title = card.findElement(By.cssSelector(cssSelector:".card-title"));

            if (title.getText().equalsIgnoreCase(restaurantName)) {
                // Click the 'View Meals' button in this card
                WebElement viewMealsButton = card.findElement(By.cssSelector(cssSelector:"a.btn.btn-success"));
                viewMealsButton.click();
                return;
            }
        }

        throw new RuntimeException("Restaurant with name '" + restaurantName + "' not found.");
    }

    @And("cliks the reservation button of a valid meal")
    public void clicksMealReserveButton() {
        WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(2));
        try {
            // Wait for all reserve buttons to be present and clickable
            List<WebElement> reserveButtons = wait.until(
                ExpectedConditions.presenceOfAllElementsLocatedBy(By.cssSelector(cssSelector:"button.btn.btn-success.btn-sm"))
            );

            // Skip the first two buttons and click the third one
            if (reserveButtons.size() > 2) {
                reserveButtons.get(2).click();  // Click the third button
            } else {
                throw new AssertionError("Not enough valid meal reservation buttons found.");
            }
        } catch (TimeoutException e) {
            throw new AssertionError("No valid meal reservation button found.");
        }
    }
```

```java
    @Then("the user should get an alert with its reservation code")
    public void getResult() {
        WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(2));
        try {
            // W
            WebE
                ExpectedConditions.visibilityOfElementLocated(By.cssSelector(cssSelector:".modal-title"))
            );
            assertEquals(expected:"Reservation Confirmation", modalTitle.getText(), message:"Modal title is incorrect.");

            // Check confirmation message
            WebElement modalMessage = driver.findElement(By.id(id:"modalMessage"));
            assertEquals(expected:"Reservation confirmed!", modalMessage.getText(), message:"Modal message is incorrect.");

            // Ensure the code block is displayed
            WebElement codeBlock = driver.findElement(By.id(id:"highlightCode"));
            assertTrue(codeBlock.isDisplayed(), message:"Reservation code block is not displayed.");

        } catch (TimeoutException e) {
            throw new AssertionError("Reservation modal did not appear.");
        }
    }
}
```

org.openqa.selenium.support.ui.ExpectedConditions

Canned ExpectedConditions which are generally useful within webdriver tests.

3.4 Non functional testing

I applied lighthouse tests and the results can be checked on my repository. Overall it reported the following scores:

k) Performance: 99
l) Accessibility: 66
m) Best Practices: 89
n) SEO: 91

I can conclude that accessibility is a problem to address in future work. To improve it, I would start by applying the framework recommendations:

o) Give buttons an accessible name.
p) And apply more contrat between the font and the background, etc.

I did not apply load tests.

3.5 Code quality analysis

I tried to use SonarQube for the static code analysis but had problems with its installation. It occupies a lot of disk space, and my computer can't have that space for now. I am working to fix it.

3.6 Continuous integration pipeline [optional]

Not implemented.

4 References & resources

Project resources

Resource:	URL/location:
Git repository	https://github.com/rodrigoabreu22/TQS_113626/tree/main/HW1
Video demo	https://youtu.be/hfZvDBbodD4
QA dashboard (online)	Not impemented.
CI/CD pipeline	Not implemented.
Deployment ready to use	Not implemented