HW1: Mid-term assignment report

André Vasques Dora [113613], v2025-04-09

1 Introduction

1.1 Overview of the work

The product developed is Moliceiro University, an application designed to manage meal reservations within a university campus. The primary purpose of this application is to allow students and staff to easily browse available meals at campus restaurants, make reservations and view meal menus.
In addition, the application includes features like weather forecasts to help users plan their visits based on the weather and the ability to cancel reservations as needed.

1.2 Current limitations

There are several known limitations in the current implementation of the Moliceiro University application. These limitations are either features that have not been fully implemented or issues that are expected to be addressed in future updates. The key limitations include:

Token Management for Reservations: The automatic generation and management of tokens for reservations are not functioning as intended. Currently, tokens are not generated automatically when a reservation is made, which limits the functionality of the reservation system. This feature was intended to provide a unique identifier for each reservation, but it is not fully integrated.

Lack of Frontend Testing: Although backend testing has been implemented extensively, including tests for services, models, and controllers, there are no tests for the frontend. This means that user interactions with the frontend, such as making reservations, viewing menus, and interacting with the user interface, have not been validated through automated testing.

Missing Reservation States: Currently, the reservations lack the ability to track different states (such as "pending," "confirmed"). Reservations can only be canceled, but there is no mechanism to monitor or manage the progression of reservations over time.

2 Product specification

2.1 Functional scope and supported interactions

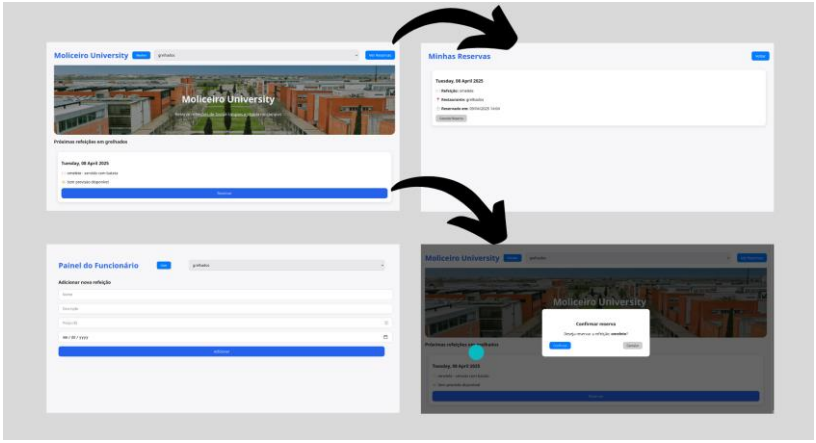
Actors and Interactions:
Students - Main Functions:

- View daily menus from restaurants.
- Reserve meals with a user token.
- Cancel reservations.
- Check reservation status.

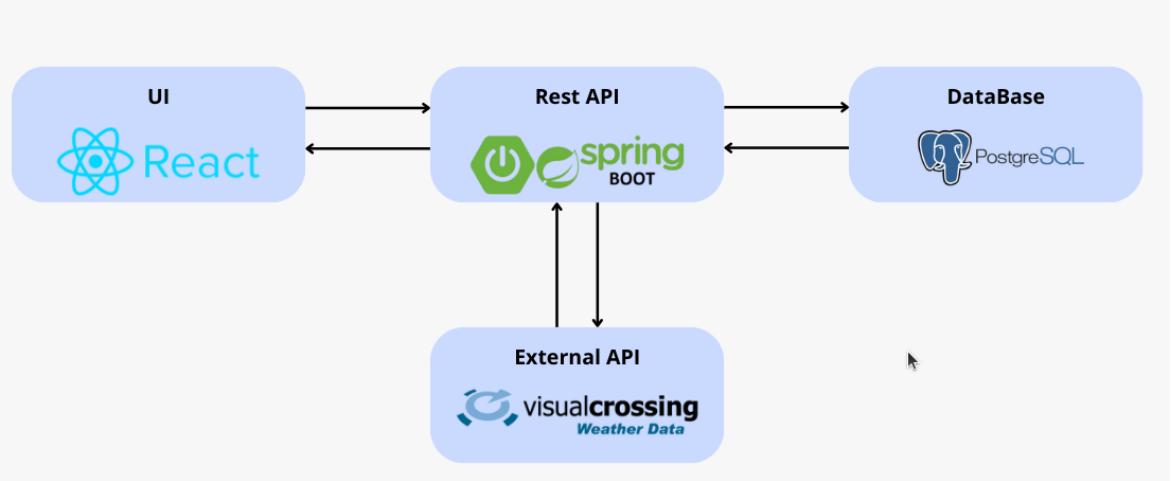
Restaurant Workers - Main Functions:

- Add new meals to the restaurant menu.
- View and manage reservations.

Summary of User Experience:
Home Page: Displays a list of restaurants with their menus.
Reservation Process: Users choose meals and confirm reservations.
Worker Dashboard: Allows restaurant staff to manage meal.



2.2 System implementation architecture



2.3 API for developers

3 Quality assurance

3.1 Overall strategy for testing

The testing strategy for the project focused on Behavior Driven Development (BDD), leveraging tools like Cucumber, JUnit, and Mockito, among others. Test Driven Development (TDD) was not used in this project.

3.2 Unit testing

During the development of the Moliceiro University application, unit testing were employed to ensure the application functions as expected and remains reliable as it evolves.

Unit tests were developed to validate critical components, such as reservation creation, input validation, and caching behavior. These tests are designed to verify the execution of individual units of code in isolation, ensuring that each part of the application functions as expected without relying on external systems.

```java
@Test
void testCheckWeather_cacheHit() {
    ReflectionTestUtils.setField(weatherService, "apiUrl", "http://dummy.url");

    Map<String, String> cache = (Map<String, String>) ReflectionTestUtils.getField(weatherService, "cache");
    cache.put("weather", "mocked weather");

    String result = weatherService.checkWeather();

    assertEquals("mocked weather", result);
}

@Test
void testGetCacheStats() {
    ReflectionTestUtils.setField(weatherService, "apiUrl", "http://dummy.url");

    weatherService.checkWeather();

    ReflectionTestUtils.setField(weatherService, "cache", Map.of("weather", "fake data"));

    weatherService.checkWeather();

    Map<String, Integer> stats = weatherService.getCacheStats();

    assertEquals(2, stats.get("totalRequests"));
    assertEquals(1, stats.get("cacheHits"));
    assertEquals(1, stats.get("cacheMisses"));
}
```

```java
@Test
void testAddMeal() {
    Meal meal = new Meal(1L, null, "Meal 1", "Description", 10.0, null, "Restaurante");
    when(mealRepository.save(meal)).thenReturn(meal);

    Meal result = mealService.addMeal(meal);

    assertEquals(meal, result);
    verify(mealRepository, times(1)).save(meal);
}
```

```java
@Test
void testCreateRestaurant() throws Exception {
    Restaurant restaurant = new Restaurant(1, "cantina de Santiago", "Aveiro", "9:00 - 22:00", LocalDate.now());
    when(restaurantService.addRestaurant(any(Restaurant.class))).thenReturn(restaurant);

    mockMvc.perform(post("/api/add_restaurant")
            .contentType("application/json")
            .content(
                    "{\"name\": \"cantina de Santiago\", \"location\": \"Aveiro\", \"openingHours\": \"9:00 - 22:00\",
            .andExpect(status().isCreated())
            .andExpect(jsonPath("$.name").value("cantina de Santiago"))
            .andExpect(jsonPath("$.location").value("Aveiro"));
}
```

4 References & resources

Project resources

Resource:	URL/location:
Git repository	https://github.com/andredora/TQS_113613
QA dashboard (online)	Ind_Project/docs