HW1: Mid-term Assignment Report

Teste e Qualidade de Software

José Maria Mendes [114429]

v2025-03-26

Contents

1 Introdução

1.1 Visão geral do trabalho

Este relatório apresenta o Mid-term Assignment necessário para a unidade curricular de TQS, abordando tanto os recursos do produto de software quanto a estratégia de garantia de qualidade adotada. A aplicação que desenvolvi chama-se Cantinas Universitárias e consiste numa plataforma Web que permite aos utilizadores reservar refeições nas diversas cantinas do Campus, com antecedência. O sistema, além de fornecer uma visão geral das refeições disponíveis para até 6 dias, incluindo o atual, disponibiliza também uma previsão diária do estado do tempo, de modo a apoiar os utilizadores nas suas decisões.

1.2 Limitações atuais

A aplicação apresenta as seguintes limitações:

- Não tem autenticação nem homepage
- Não é possivel controlar o número de reservas
- Só pode escolher Almoço ou Jantar no mesmo dia

2 Especificação do produto

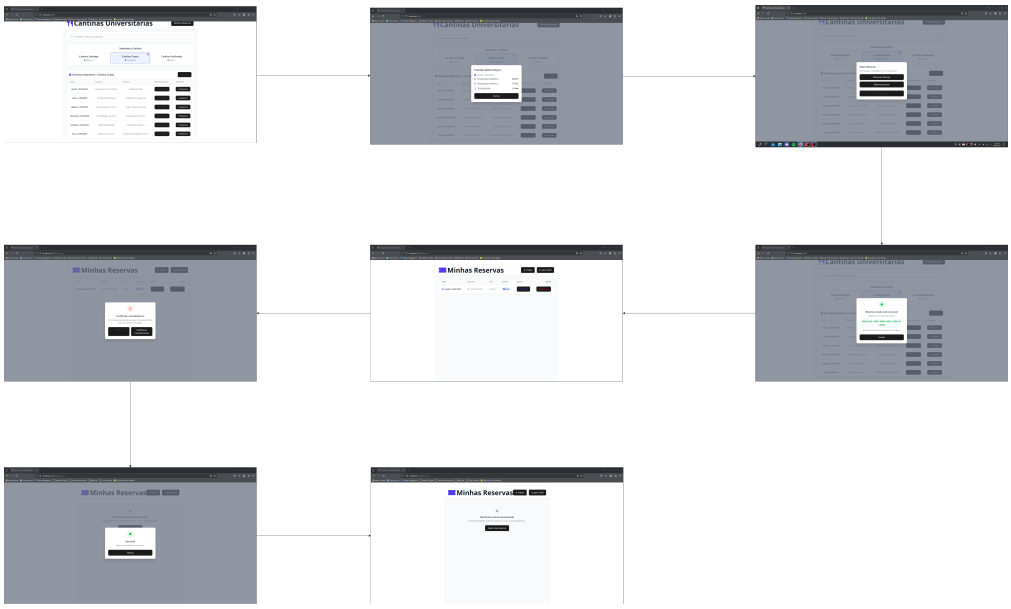
2.1 Escopo funcional e Interações suportadas

A aplicação é destinada aos alunos e funcionários da Universidade de Aveiro que pertendem reservar refeições com antecedência nas cantinas disponíveis.

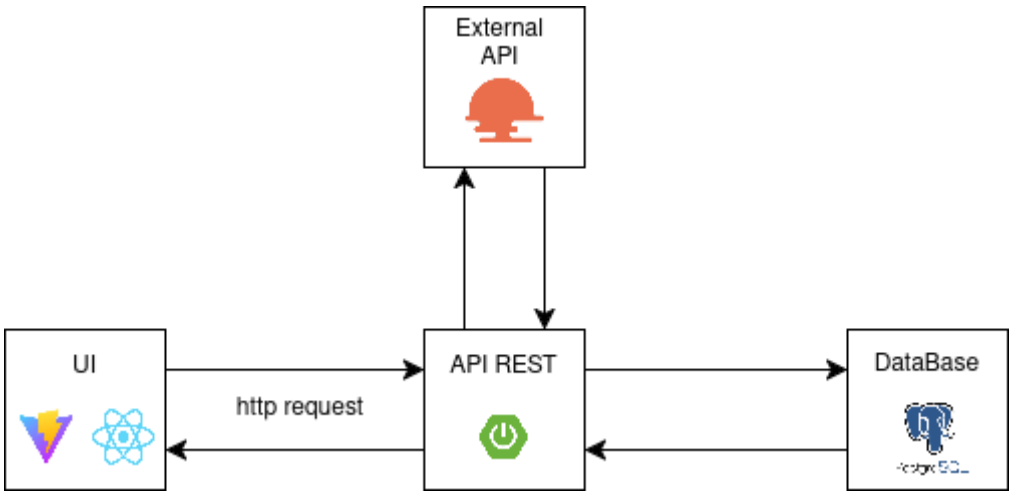
Os principais atores são:

- Utilizador Comum:
  - Ver as cantinas disponíveis e refeições disponíveis;
  - Realizar uma reserva para uma refeição futura;
  - Consultar as reservas feitas;
  - Cancelar as reservas feitas.
- Staff da cantina
  - Realizar o check-in das reservas dos utilizadores por um token de reserva.

A imagem abaixo mostra um fluxo simples da utilização da aplicação pelo utilizador:



2.2 Arquitetura do sisema

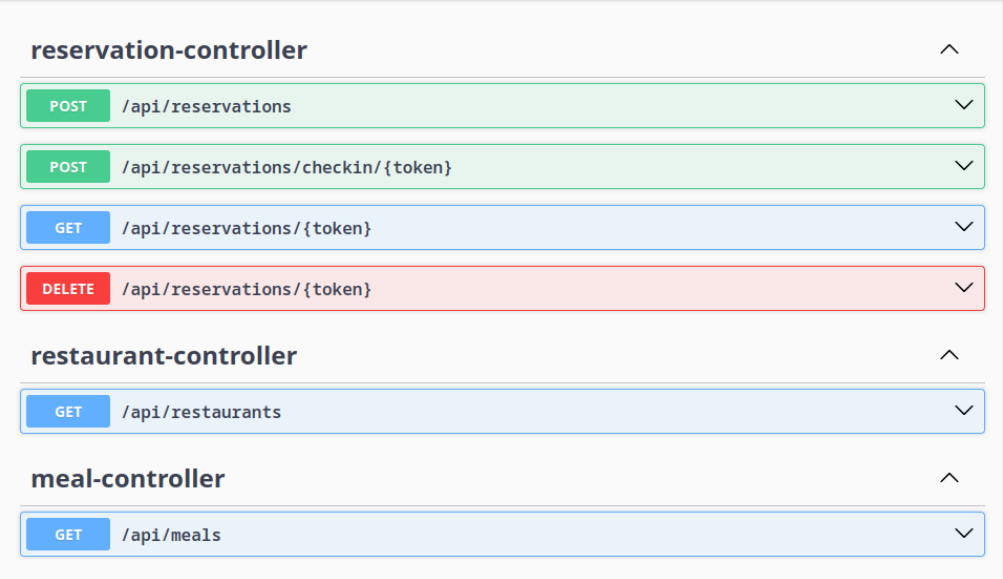


Como é possível ver acima, a arquitetura da aplicação Cantinas Universitárias está dividida em três camadas principais:

- Frontend: Desenvolvido em React e Vite, este é responsável pela interface. Envia pedidos HTTP à API REST.
- Backend: Implementado com Spring Boot, faz a ligação entre o frontend, a database e a external API
- DataBase: Utiliza o PostgreSQL para armazenar os dados dos restaurantes, refeições e reservas feitas.
- API Externa: É usada o Open-Meteo para obter as previsões metereológicas diárias.

2.3 API para desenvolvedores

A API foi projetada de modo a permitir a interação com o sistema de reservas de forma simples e eficiente. Na imagem abaixo, é possível ver os endpoints disponíveis:



Lista dos Endpoints:

- POST /api/reservations
  - Criação de uma nova reserva.
- POST /api/reservations/checkin/token
  - Realiza o check-in de uma reserva usando o token da mesma.
- GET /api/reservations/token
  - Pegar os detalhes de uma reserva pelo token.
- DELETE /api/reservations/token
  - Cancelar uma reserva pelo token.
- GET /api/restaurants
  - Devolve uma lista dos restaurantes disponíveis para reserva.
- GET /api/meals
  - Retorna as refeições disponíveis de um restaurante.

3 Garantia de Qualidade

3.1 Estratégia geral para testes

A estratégia de testes adotada para este projeto baseou-se numa abordagem incremental, orientada a testes unitários e de integração, com foco nas principais funcionalidades do back-end.

Desde o início, os testes foram desenvolvidos em paralelo com a implementação do serviço de reservas, de modo a assegurar que cada caso de uso principal fosse testado isoladamente, utilizando JUnit e Mockito, em conformidade com a abordagem TDD (Test-Driven Development).

Para validar a correta integração entre as camadas da aplicação (controller, services e repositories), foram criados testes de integração com Spring Boot e MockMvc, recorrendo a uma base de dados em memória (H2) para simular requisições reais à API.

Por fim, foram realizados testes de desempenho com k6, com o objetivo de avaliar a robustez da aplicação sob condições de carga intensiva.

Esta combinação de estratégias permitiu alcançar uma cobertura ampla e fiável da aplicação.

Testes Realizados

A seguir são descritos os testes implementados para garantir a robustez, correção e desempenho da aplicação:

- **Testes Unitários**
  - `MealWithWeatherDTOTest.java`: Verifica a correta criação do objeto `MealWithWeatherDTO`.
  - `ReservationResponseDTOTest.java`: Testa a criação e integridade do objeto `ReservationResponseDTO`.
  - `ReservationServiceTest.java`: Avalia a lógica do serviço de reservas, incluindo criação, consulta, check-in e cancelamento de reservas.
  - `WeatherServiceTest.java`: Valida a obtenção da previsão meteorológica para uma data específica.
- **Testes de Integração**
  - `ReservationControllerTest.java`: Testa a integração da API de reservas, cobrindo criação, cancelamento e check-in.
  - `RestaurantControllerTest.java`: Garante que o controller de restaurantes retorna corretamente as informações esperadas.
- **Testes de Desempenho**
  - `PerformanceGetTest.js`: Avalia o tempo de resposta da API para requisições do tipo GET, simulando múltiplos acessos simultâneos.
  - `PerformancePostTest.js`: Testa a capacidade da API em lidar com múltiplos utilizadores a realizar reservas em simultâneo.

3.2 Testes unitários e de integração

A estratégia de testes adotada baseou-se na separação clara entre Testes Unitários e Testes de Integração.

Os Testes Unitários concentraram-se na validação da lógica interna da aplicação, com especial foco no serviço `ReservationService`. Para isso, foram utilizadas dependências simuladas (mocks) com o auxílio do Mockito, permitindo isolar o comportamento de componentes externos, como os repositórios. Esses testes permitiram verificar o cumprimento de regras específicas, como a prevenção de reservas duplicadas e a impossibilidade de realizar check-in em reservas já canceladas.

Já os Testes de Integração foram desenvolvidos com Spring Boot Test e MockMvc, e tiveram como objetivo validar a comunicação entre os *Controllers*, *Services* e a base de dados em memória (H2). Nessa abordagem, simulam-se chamadas reais à API REST, cobrindo cenários completos de uso, como a criação, consulta, cancelamento e check-in de reservas.

Essa combinação de estratégias garantiu que tanto a lógica de negócio quanto os fluxos completos da aplicação funcionassem corretamente em diferentes contextos e condições reais de utilização.

3.3 Testes não funcionais

Os testes não funcionais tiveram como foco principal a avaliação do desempenho da API de reservas. Para isso, foi utilizada a ferramenta `k6`, que permitiu simular cenários de carga e monitorar o comportamento da aplicação durante essas situações.

O principal objetivo desses testes foi analisar a robustez e a escalabilidade da API, assegurando que ela é capaz de lidar com múltiplos pedidos simultâneos sem comprometer significativamente o tempo de resposta.

3.4 Testes de Desempenho (Não Funcionais)

Os testes não funcionais incidiram sobre a avaliação do desempenho da API de reservas, recorrendo à ferramenta **k6** para simular cenários de carga e analisar o comportamento da aplicação sob pressão.

Configuração dos Testes

- **Ferramenta usada:** k6
- **Simulações realizadas:**
  - 50 *Virtual Users* (VUs) durante 10 segundos para os testes de `GET`
  - 30 *Virtual Users* (VUs) durante 15 segundos para os testes de `POST`
- **Endpoints testados:**
  - `GET /api/restaurants` – Consulta da lista de restaurantes
  - `GET /api/meals?restaurantId=X` – Consulta das refeições de um restaurante (IDs de 1 a 3)

- `POST /api/reservations` – Criação de reservas com variações de tipo de refeição (Almoço/Jantar) e datas distintas

Resultados Obtidos

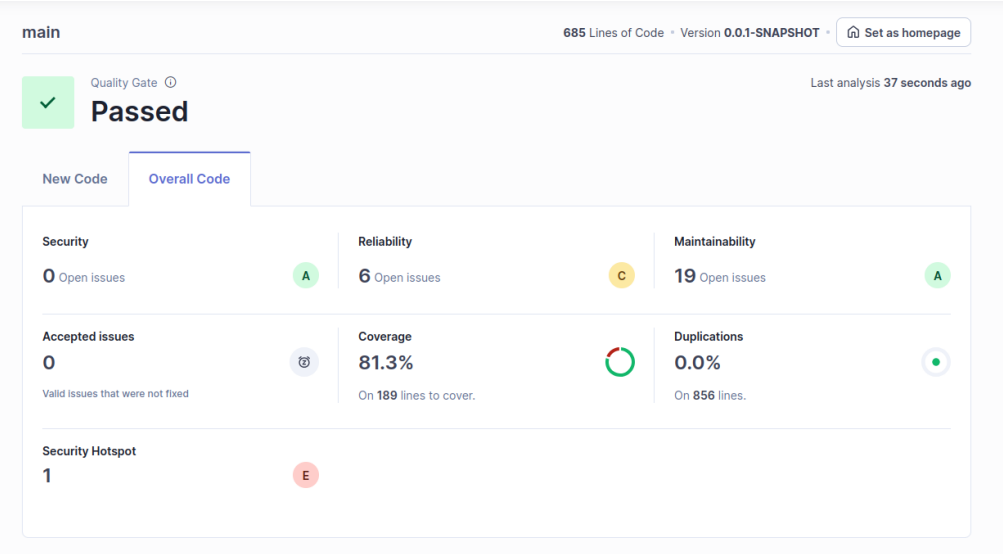
- **GET /api/restaurants e GET /api/meals?restaurantId=X**
  - Total de requisições: **4900**
  - Taxa de sucesso: **100%**
  - Tempo médio de resposta: **1.97 ms**
  - Tempo máximo de resposta: **14.49 ms**
  - Percentis:
    - P90: 3.77 ms
    - P95: 4.93 ms
- **POST /api/reservations**
  - Total de requisições: **18,218**
  - Taxa de sucesso: **98.29%**
  - Taxa de falhas: **1.70%**
  - Tempo médio de resposta: **33.63 ms**
  - Tempo máximo de resposta: **6.42 ms**
  - Percentis:
    - P90: 11.78 ms
    - P95: 14.47 ms

Conclusão

A API demonstrou uma **excelente capacidade de resposta** nos testes de leitura (GET), com tempos de resposta extremamente baixos mesmo sob carga. Os testes de criação (POST) apresentaram ligeiras falhas devido ao maior volume de interações com a base de dados, mas mantiveram uma taxa de sucesso elevada. No geral, o sistema mostrou-se **escalável e robusto**, conseguindo lidar com múltiplos pedidos concorrentes com **tempos de resposta satisfatórios**.

3.5 Análise da qualidade do código

A análise da qualidade do código foi feita no SonarQube, que fornece informações sobre confiabilidade, manutenibilidade, segurança e cobertura de testes. A cobertura de testes está fixa nos 81,3%, verificando que está num bom nível de verificação das funcionalidades.



4 References & Resources

4.1 Project resources

- **Git repository:** `https://github.com/jmendes418/TQS_114429/tree/main/HW1-114429`

4.2 Referências

`https://open-meteo.com/`
`https://site.mockito.org/`