deti universidade de aveiro
departamento de eletrónica, telecomunicações e informática

HW1: Mid-term assignment report

Tiago José Soares de Albuquerque [112901], v2025-03-26

1 Introdução

1.1 Visão geral do trabalho

Este relatório apresenta o Mid-term Assignment necessário para TQS, abrangendo tanto os recursos do produto de software quanto a estratégia de garantia de qualidade adotada.
A aplicação que desenvolvi chama-se **MealBooking** e consiste numa plataforma Web que permite que os utilizadores reservem refeições nas diversas cantinas do Campus, com antecedência. O sistema, para além de fornecer uma visão geral das refeições disponíveis por até 6 dias, incluindo o atual, partilha também uma previsão, diária, do tempo para apoiar um utilizadores em possíveis decisões.

1.2 Limitações atuais

Tendo em conta o atual estado de desenvolvimento, a aplicação MealBooking apresenta as seguintes limitações:

- Falta de autenticação dos utilizadores;
- Ausência de gestão da capacidade das cantinas - Não há limite para o número de reservas por refeição;
- Sistema de Notificações.

2 Especificações do produto

2.1 Escopo funcional e Interações suportadas

A aplicação **MealBooking** destina-se a alunos e funcionários da Universidade de Aveiro que pretendem reservar refeições nas cantinas disponíveis.

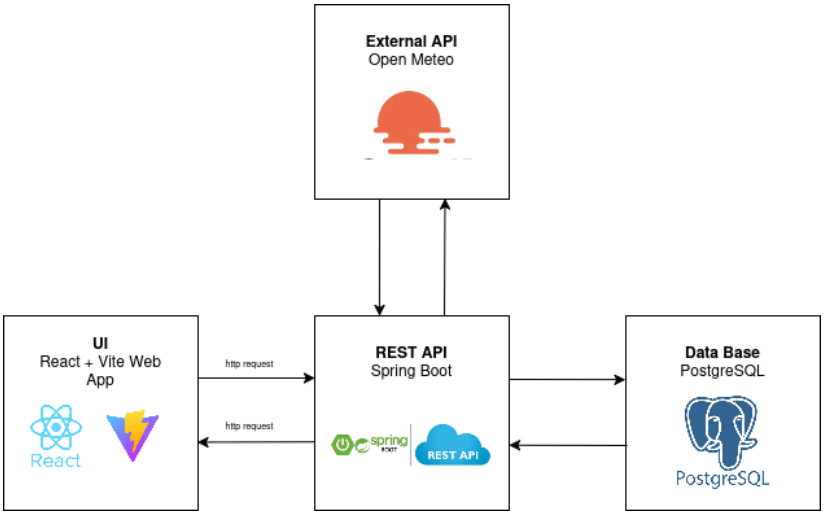
Os principais atores e as suas respectivas interações, são então:

- **Utilizador Comum** (Aluno, Funcionário, etc.):
  - Visualizar a lista de cantinas disponiveis;
  - Consultar as refeições disponiveis por cantina, por data e tipo (Almoço/Jantar), incluindo a previsão meterologica do dia;
  - Realizar reservas para refeições futuras, escolhendo a cantina, a data e o tipo da refeição;
  - Consultar a lista das reservas (ativas e anteriores);
  - Cancelar reservas ativas.
- **Staff da Cantina:**
  - Acesso a uma interface onde pode dar check-in à reserva do utilizador, com base no respetivo token de reservas.

Fluxo simples da utilização por parte do **Utilizador Comum:**

A página do staff apenas valida, ou não, a reserva efetuada pelo utilizador com base no token da mesma. Será partilhada no vídeo de demonstração, visto que, pela sua simplicidade, não achei que se justificasse a sua presença neste fluxo.

2.2 Arquitetura da implementação do Sistema

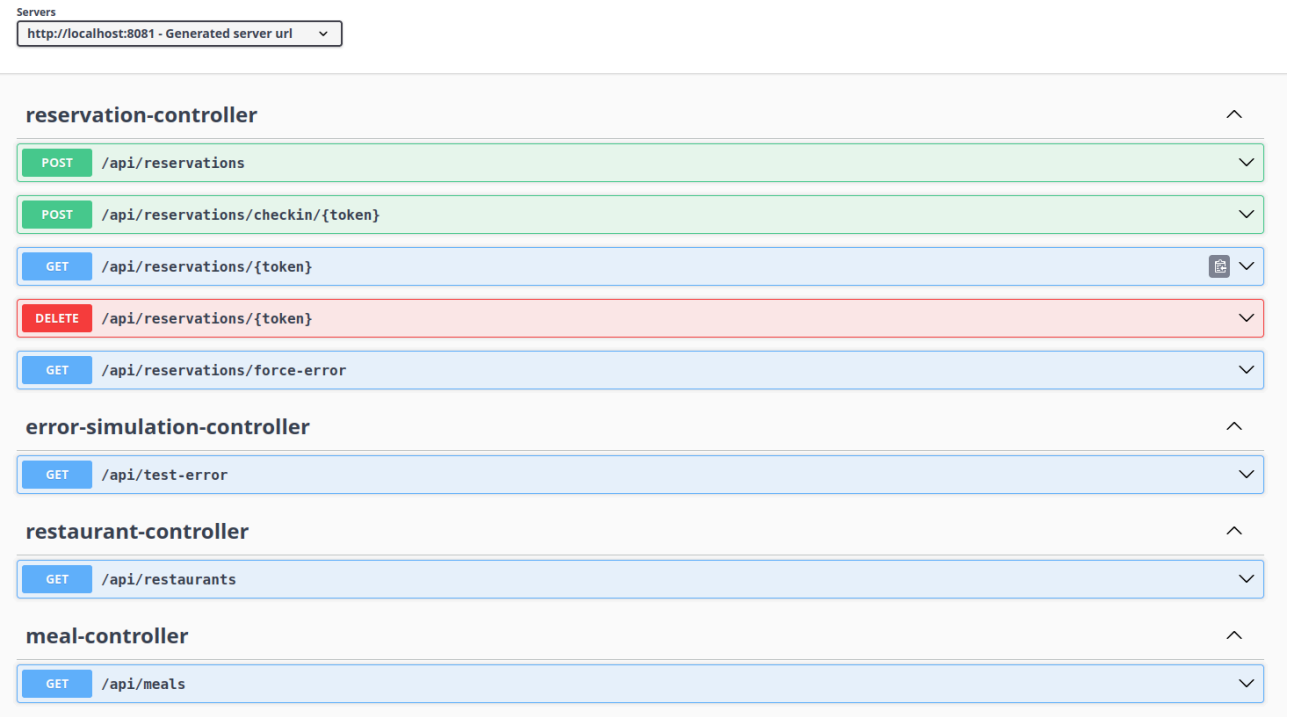


Conforme detalhado na imagem acima, a **arquitetura** da aplicação **MealBooking** segue uma abordagem baseada em **três camadas principais**:

- **Frontend**: Desenvolvido com *React* e *Vite*, é responsável pela interface do utilizador. Envia pedidos HTTP à API REST para consultar refeições, fazer reservas e visualizar as previsões meteorológicas.
- **Backend (API REST)**: Implementado com *Spring Boot*, atua como intermediário entre o frontend, a Base de Dados e a API externa.
- **Base de Dados**: Utiliza *PostgreSQL* para armazenar dados estruturados de restaurantes, refeições e reservas efetuadas pelos utilizadores.
- **API Externa (Open-Meteo)**: O backend consulta esta API para obter previsões meteorológicas diárias associadas às refeições.

2.3 API para desenvolvedores

A API foi projetada para permitir a interação com o sistema de reservas de refeições de forma simples e eficiente. Abaixo, estão os principais endpoints disponíveis:



Endpoints da API - Detalhados:

- **POST /api/reservations**
  - **Descrição**: Criação de uma nova reserva.
- **POST /api/reservations/checkin/{token}**
  - **Descrição**: Realiza o check-in para uma reserva utilizando o token de reserva.
- **GET /api/reservations/{token}**
  - **Descrição**: Recupera os detalhes de uma reserva utilizando o token de reserva.
- **DELETE /api/reservations/{token}**
  - **Descrição**: Cancela uma reserva utilizando o token da reserva.
- **GET /api/reservations/force-error**
  - **Descrição**: Endpoint utilizado para testar erros na aplicação (simula falhas).
- **GET /api/restaurants**
  - **Descrição**: Retorna a lista de restaurantes disponíveis para reservas.
- **GET /api/meals**
  - **Descrição**: Retorna as refeições disponíveis para um restaurante específico.

3 Garantia de Qualidade

3.1 Estratégia geral para testes

A estratégia de testes adotada para este projeto baseou-se, primeiramente, numa abordagem **incremental e orientada a testes unitários e de integração**, cobrindo as principais funcionalidades do backend.

Inicialmente, os testes foram desenvolvidos **em paralelo com a implementação do serviço de reservas**, para tentar assegurar que cada caso de uso principal era testado de forma isolada, com recurso ao **JUnit** e **Mockito** (seguindo o referido **TDD – Test-Driven Development**).

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

Seguidamente, foram desenvolvidos **testes funcionais com Selenium WebDriver**, para testar o frontend completo, e, por último, **testes de desempenho com k6**, para avaliar a robustez da aplicação sob alguma carga intensiva.

Esta combinação de abordagens permitiu garantir uma cobertura alargada e sólida da aplicação.

Organização dos testes:

- **Testes Unitários**:
  - Focados em testar a lógica isolada de métodos e funções específicas sem qualquer dependência externa.
  - São eles:
    - **MealWithWeatherDTOTest.java** -> Testa a criação do *MealWithWeatherDTO*;
    - **GlobalExceptionHandlerTest.java** -> Testa o tratamento de exceções globais;
    - **ReservationServiceTest.java** -> Testa a criação das reservas, consulta, cancelamento e check-in das mesmas;
    - **MealControllerTest.java** -> Valida a resposta do método `getMealsWithWeather` quando as refeições e as previsões meteorológicas são retornadas corretamente.
    - **WeatherServiceTest.java**: Testa a obtenção da previsão do tempo para uma data específica no serviço `WeatherService`.

- **Testes de Integração**:
  - Focados na interação entre as diferentes camadas da aplicação, garantindo que elas se comunicam de forma correta.
  - São eles:
    - **ReservationControllerIT.java**: Testa a criação, o cancelamento e o check-in de reservas, pela API;
    - **MealControllerIT.java**: Verifica a integração do controller de refeições com o serviço de previsões meteorológicas.
    - **ReservationControllerMockTest.java**: Testa o controller `ReservationController`, garantindo que os endpoints da API para criação, consulta, cancelamento e check-in de reservas respondem corretamente para os diferentes cenários, incluindo erros e sucesso.
- **Testes Funcionais**:
  - Teste completo da aplicação, com a UI e pelos fluxos de interação do usuário.
  - São eles:
    - **FunctionalTestHW1Test.java**: Verifica a interação com a interface web, incluindo a reserva de refeições.
- **Testes de Desempenho**:
  - Avaliam a performance da aplicação sob carga, verificando a escalabilidade e a eficiência da API.
  - São eles:
    - **PerformancePostTest.js**: Testa a criação de reservas com múltiplos utilizadores simultâneos;
    - **PerformanceGetTest.js**: Avalia o tempo de resposta para a obtenção de informações sobre refeições.
- **Testes de Aplicação**:
  - Validam a inicialização e o carregamento do contexto da aplicação, garantindo que a aplicação funciona corretamente.
  - São eles:
    - **BackendApplicationTest.java**: Testa a inicialização do Spring Boot.

3.2 Testes unitários e de integração

A estratégia, inicialmente pensada e posteriormente adotada, passou por separar, claramente, os **Testes Unitários** dos **Testes de Integração**.
Os **Testes Unitários** focaram-se na lógica interna da aplicação, em particular no serviço *ReservationService*, recorrendo a **mocks** (com *Mockito*) para isolar o comportamento de dependências como os repositórios. Estes testes permitiram validar algumas regras impostas, como a impossibilidade de criar reservas duplicadas ou realizar check-in em reservas canceladas.

Por outro lado, os **Testes de Integração** foram desenvolvidos com **Spring Boot Test** e **MockMvc**, e validam a interação entre os Controllers, Services e a Base de Dados (*H2* em memória). Nestes testes, simula-se o comportamento real de um utilizador ao chamar os endpoints da API REST, testando a aplicação de forma mais completa. Esses testes garantem que os fluxos da aplicação, como **criação, consulta, cancelamento e check-in de reservas**, estejam operacionais sob condições de uso reais.

Assim, esta combinação permitiu garantir que tanto a lógica interna como os fluxos completos da aplicação funcionam corretamente sob diferentes cenários.

3.3 Testes Funcionais

Os **Testes Funcionais** foram implementados com recurso ao **Selenium WebDriver**, simulando a interação real de um utilizador com a interface web da aplicação. O objetivo foi validar os principais fluxos da experiência do utilizador, desde a navegação até à realização de uma reserva e apresentação do token:

- Selecionar uma cantina e visualizar as refeições disponíveis;
- Realizar uma reserva para uma refeição (Almoço ou Jantar);
- Cancelar uma reserva na página respetiva;
- Aceder à página do staff e realizar check-in com um token válido;
- Verificar mensagens de erro ou sucesso (em situações como tentar reservar uma refeição já reservada).

Algumas **estratégias** usadas no teste:

- **WebDriverWait** para evitar falhas em elementos que não estão visíveis e/ou clicáveis;
- Tratamento de alertas inesperados com **handleOptionalAlert()**, assegurando maior robustez em cenários de falha;
- Utilização de **scrollIntoView** e **safeClick()** para evitar exceções como **ElementClickInterceptedException**.

Exemplo das **estratégias** referidas no código:

```
// 16 | click | css=.token-button |
handleOptionalAlert();
WebElement tokenButton = new WebDriverWait(driver, Duration.ofSeconds(10))
        .until(ExpectedConditions.elementToBeClickable(By.cssSelector(".token-button")));
tokenButton.click();
```

Este tipo de teste garante que a aplicação funciona como esperado do ponto de vista do utilizador final, cobrindo funcionalidades reais e fluxos de navegação importantes.

3.4 Testes não funcionais

Os **Testes não funcionais** focaram-se, principalmente, na avaliação de desempenho da API de reservas, recorrendo à ferramenta **k6** para simular carga e analisar o comportamento da aplicação sob a mesma.

Objetivo:

- Avaliar a robustez e a escalabilidade da API, garantindo que consegue lidar com múltiplos pedidos simultâneos sem prejuízo acentuado no tempo de resposta.

Configuração do teste de carga:

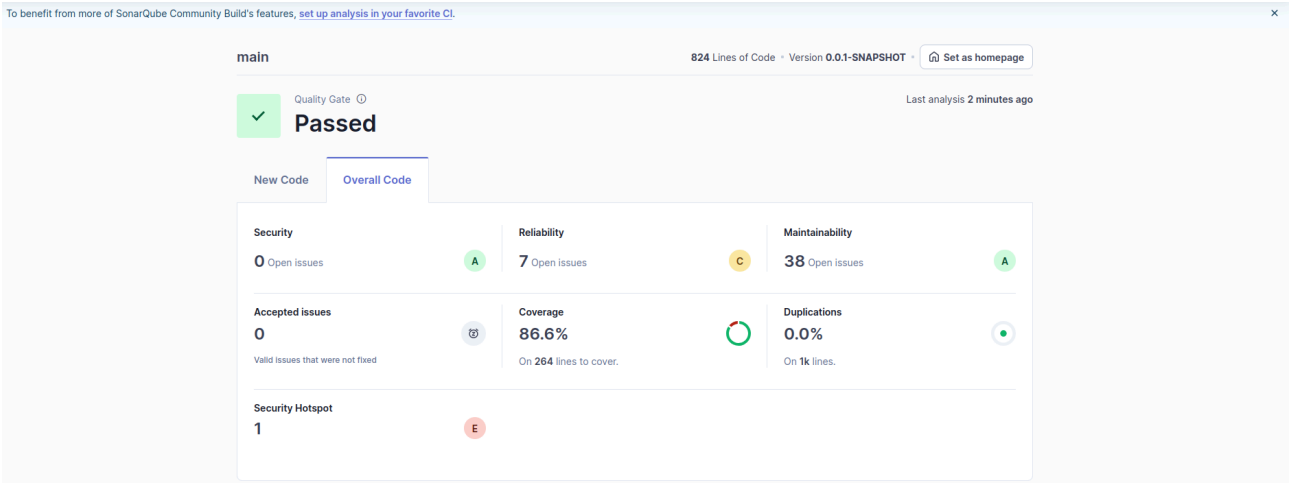
- ➔ **Ferramenta usada**: k6;
- ➔ **Simulação**: 100 utilizadores virtuais (VUs) durante 5 segundos para o teste dos **GET's**, e 20 VUs durante 10 segundos para o teste dos **POST's**.
- ➔ **Testes realizados**:
  - ◆ **GET /api/restaurants** -> Simulação de consultas para obter a lista de restaurantes.
  - ◆ **GET /api/meals?restaurantId=X** -> Simulação de consultas para obter as refeições de um restaurante específico, utilizando os IDs de 1 a 3.
  - ◆ **POST /api/reservations** -> Simulação de reservas com variações de tipo de refeição (Almoço/Jantar) e datas diferentes para evitar duplicação de reservas.
- ➔ **Resultados dos Testes**:
  - ◆ **POST /api/reservations:**
    - **Total de requisições realizadas**: 28,788
    - **Taxa de sucesso**: 81.45%
    - **Tempo médio de resposta**: ~44ms
  - ◆ **GET /api/restaurants** e **GET /api/meals?restaurantId=X:**
    - **Taxa de sucesso**: 99.94%
    - **Tempo médio de resposta**: <50ms
    - **Observações**: O sistema demonstrou uma excelente capacidade de resposta, com tempos de resposta bastante baixos, mesmo com múltiplos utilizadores a realizar consultas simultâneas.

Conclusão:

O sistema demonstrou uma **boa capacidade de resposta e escalabilidade**, com tempos de resposta baixos mesmo sob carga.

3.5 Análise da qualidade do código

A análise da qualidade do código foi realizada utilizando o **SonarQube**, que forneceu informações sobre **confiabilidade**, **manutenibilidade**, **segurança** e **cobertura de testes**. A **cobertura de testes** está fixa nos **86,6%**, refletindo, assim, um bom nível de verificação das funcionalidades.



4 Referências e Recursos

Recursos do Projeto

Resource:	URL/location:
Git repository	https://github.com/TiagoAlb12/TQS_112901/tree/main/HW1
Video demo	https://github.com/TiagoAlb12/TQS_112901/tree/main/HW1/demo
QA dashboard (online)	Print no tópico acima
CI/CD pipeline	Nada a referir
Deployment ready to use	Nada a referir

Referências

Sites utilizados para retirar ideias:

- https://mysas.ua.pt/Home/Index
- https://www.ipma.pt/pt/otempo/prev.localidade.hora/