HW1: Mid-term assignment report

Bruno Tavares Meixedo [113372], v2025-04-09

1.1 Visão Geral 1
1.2 Limitações 1
2.1 Espetro funcional e interações suportadas 2
2.2 Arquitetura do sistema 2
2.3 Documentação da API 3
3.1 Estratégia de testes 4
3.2 Testes Unitários e de Integração 4
3.3 Testes Funcionais 4
3.4 Testes não funcionais 5
3.5 Analise da qualidade de código 7
3.6 Continuous integration pipeline [optional] 7

1 Introdução

1.1 Visão Geral

O relatório representa um projeto individual desenvolvido como parte da disciplina de TQS (Teste e Qualidade de Software). O meu projeto (CampusEmentas) tem como objetivo implementar um sistema de reservas de refeições que visa facilitar a escolha e reserva de pratos. O cliente pode selecionar o restaurante de sua preferência, escolher o prato desejado e efetuar a reserva da sua refeição. Após a reserva, o cliente receberá um código de token (reservationCode), que pode ser utilizado para consultar a reserva ou para que o funcionário do restaurante registe o "Check-in" da respetiva reserva.
A aplicação também tem um serviço que utiliza uma API externa para mostrar a previsão do tempo associada a cada prato, ajudando o cliente a escolher a refeição com base no clima. Para evitar fazer muitas chamadas à API, a aplicação utiliza um sistema de cache, tornando o processo mais rápido e eficiente.

1.2 Limitações

Uma das grandes limitações que houve foi que a API externa (OpenWeatherMap) só dá previsões de tempo até 8 dias (desde o dia presente), o que restringiu a capacidade de fornecer previsões mais longas para os users do sistema. Além disso, ao implementar a integração contínua com GitHub Actions, encontrei dificuldades com os testes automatizados de Selenium, que falharam devido à falta de uma interface gráfica no ambiente do GitHub Actions.

2 Especificações do Produto

2.1 Espetro funcional e interações suportadas

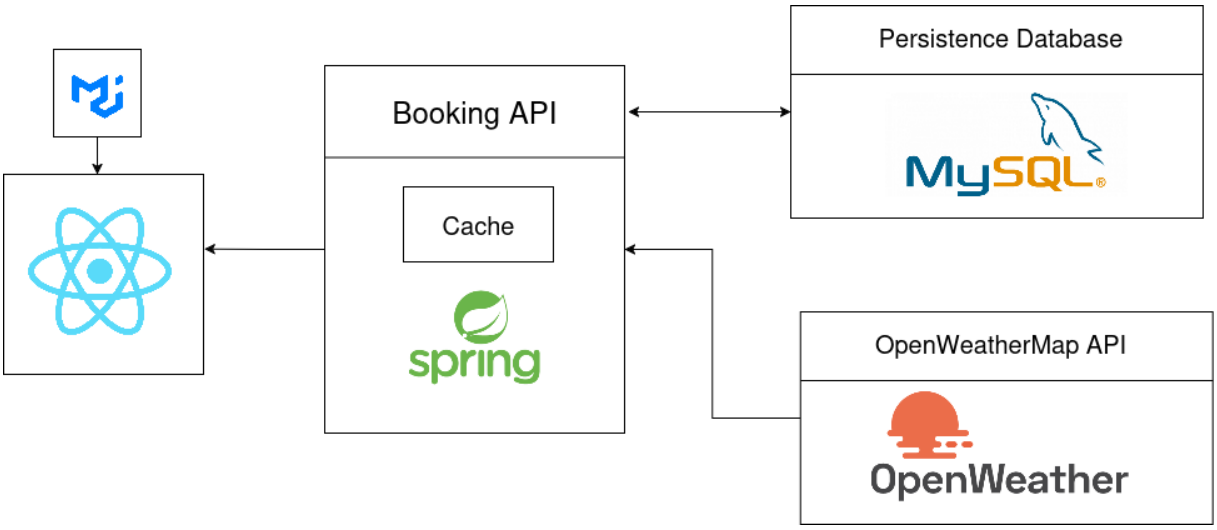
A aplicação web é destinada a dois tipos principais de utilizadores: os clientes, que a usam para fazer e consultar reservas em restaurantes, e os funcionários, que têm acesso à funcionalidade de check-in de reservas. Quando o utilizador entra na aplicação, é recebido por uma landing page que apresenta duas opções: fazer uma reserva ou consultar uma reserva existente.

Ao escolher fazer uma reserva, o cliente é encaminhado para uma página onde pode visualizar todos os restaurantes disponíveis, podendo pesquisá-los pelo nome ou filtrá-los por localização. Depois de selecionar um restaurante, é levado para uma página de ementas, que apresenta os pratos da semana atual e da semana seguinte. Esta página inclui várias informações úteis, como a data e hora de cada prato, a previsão do tempo associada, o nome do prato e o respetivo preço.

As refeições estão divididas por período do dia: almoço, que decorre entre as 11:00 e as 15:00, e jantar, entre as 18:00 e as 23:00. O utilizador pode filtrar os pratos por estes períodos para facilitar a escolha. Após selecionar o prato desejado, basta inserir o nome para concluir a reserva.

Se o cliente quiser consultar a sua reserva, pode aceder a uma página específica para isso, onde só precisa de introduzir o código de reserva. Nesta página, é possível ver todos os detalhes da reserva e eliminá-la, caso deseje. Os funcionários também utilizam esta funcionalidade para realizar o check-in da reserva quando o cliente chega ao restaurante.

2.2 Arquitetura do sistema

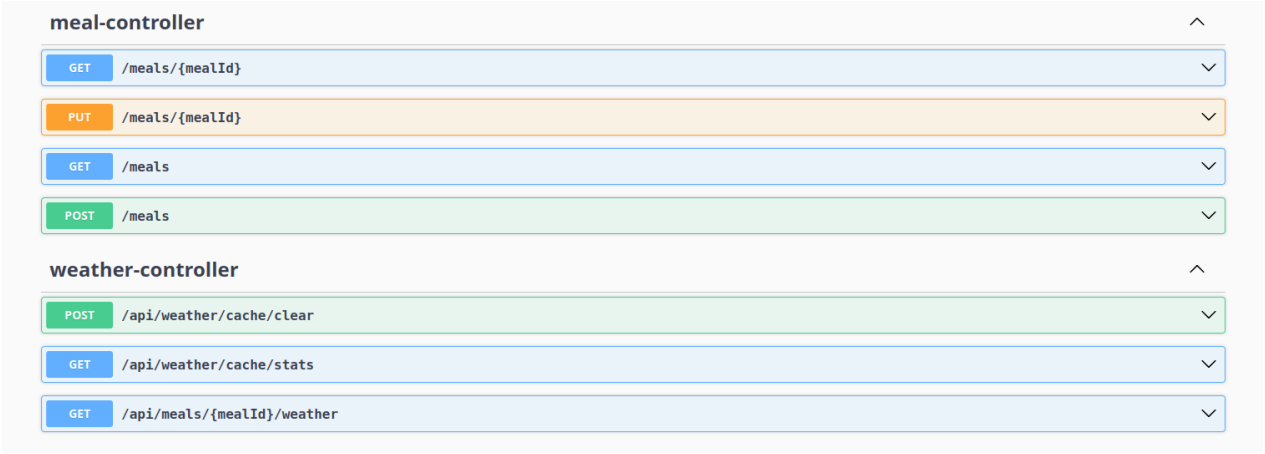


A Booking API, desenvolvida em Java com Spring Boot, gere a lógica de restaurantes, pratos e reservas. Utiliza serviços web RESTful e injeção de dependências para facilitar a

comunicação entre os componentes. A API consulta a OpenWeatherMap para obter previsões meteorológicas (dando o nome da cidade, data e hora) e usa uma cache manual atualizada a cada 15 minutos para reduzir chamadas externas. Os dados da Booking API são armazenados numa base de dados MySQL. O frontend, feito em React e Material UI (MUI), comunica com a Booking API através de fetch requests permitindo ao utilizador fazer ou consultar reservas.

2.3 Documentação da API

A API do projeto tem 4 controllers (restaurant, reservation, meal, weather). Cada um destes controllers tem alguns endpoints mas nem todos são usados diretamente no frontend. Sobre o restaurant-controller, temos a opção de criar e chamar restaurantes, mas se houver um request com o restaurantId devolve menus e reservas. No reservation-controller temos também a opção de receber e criar reservas, se enviarmos o reservationCode há a opção de editar, eliminar e de dar check in a essa reserva especifica. No meal-controller são operações basicas de “get” e "post”, e por fim no weather-controller tenho um dos principais endpoints que chama a api externa dando os dados do prato (através da mealId) e assim conseguindo a resposta da api externa, tambem criei dois endpoints de cache para conseguir ter uma boa visualização deles no frontend.



3 Garantia de Qualidade

3.1 Estratégia de testes

Quando estava a começar a fazer o projeto tentei focar me em fazer uma abordagem em TDD, mas depois de vários erros e dificuldades desisti da ideia e acabei por fazer a aplicação, onde a cada "feature" implementada ia escrevendo os seus testes e assim conseguindo corrigir as falhas na aplicação (nos services, controllers...). Usei BDD apenas uma vez quando fiz os testes em "Selenium". Outra estratégia de testes que tive foi implementar o SonarQube a meio do processo, que me ajudou a corrigir vários erros, principalmente os de segurança.

3.2 Testes Unitários e de Integração

Foram realizados testes unitários para avaliar componentes isolados, como serviços, cache e inicializadores de dados, utilizando o Mockito para simular dependências. Já os testes de integração verificaram a interação entre os componentes, como repositórios e controladores REST, usando as anotações @DataJpaTest e @WebMvcTest. O MockMVC foi usado para testar os controladores, simulando requisições HTTP.

3.3 Testes Funcionais

Usei BDD com o Selenium Web driver.

```
You, há 4 horas | 1 author (You)
Feature: Reservar e fazer check-in em um restaurante      You, há 4 h
  Como um usuário
  Quero fazer uma reserva em um restaurante
  E realizar o check-in
  Para garantir minha mesa

  Scenario: Fazer uma reserva e realizar check-in
    Given o usuário acessa a aplicação em "http://localhost:3000/"
    When o usuário navega até a página de reserva
    And clica na imagem do restaurante
    And seleciona o primeiro horário disponível
    And preenche "Bruno" como nome do cliente
    And pressiona Enter
    And confirma a reserva
    And navega para consultar reserva
    And clica na reserva
    And clica no botão de confirmar
    And clica no botão de sucesso
    Then o status deve ser "Checked In"
```

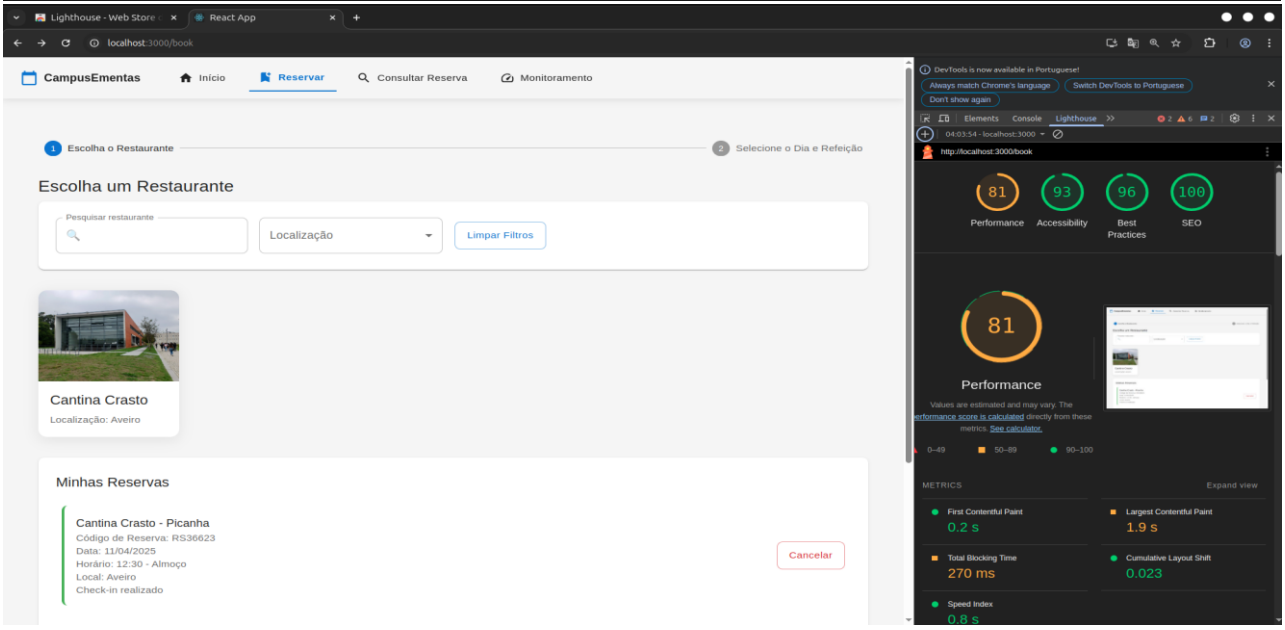
```java
You, há 4 horas | 1 author (You)
public class ReservaSteps {
    private WebDriver driver;
    private WebDriverWait wait;
    private Map<String, Object> vars;
    private JavascriptExecutor js;

    @Before
    public void setUp() {
        WebDriverManager.chromedriver().setup();
        ChromeOptions options = new ChromeOptions();
        driver = new ChromeDriver(options);
        js = (JavascriptExecutor) driver;
        vars = new HashMap<String, Object>();

        driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(15));

        wait = new WebDriverWait(driver, Duration.ofSeconds(15));      You, há 20 horas • mid assigment
    }

    @Given("o usuário acessa a aplicação em {string}")
    public void oUsuarioAcessaAplicacao(String url) {
        driver.get(url);
        driver.manage().window().setSize(new Dimension(1920, 1048));
    }

    @When("o usuário navega até a página de reserva")
    public void oUsuarioNavegaParaReserva() {
        WebElement botaoReservar = driver.findElement(By.linkText("Reservar"));
        botaoReservar.click();
    }

    @And("clica na imagem do restaurante")
    public void clicaNaImagemDoRestaurante() {
        WebElement imagemRestaurante = driver.findElement(By.cssSelector(".MuiGrid-root:nth-child(1) .MuiCardMedia-root"));
        imagemRestaurante.click();
    }

    @And("seleciona o primeiro horário disponivel")
```

3.4 Testes não funcionais

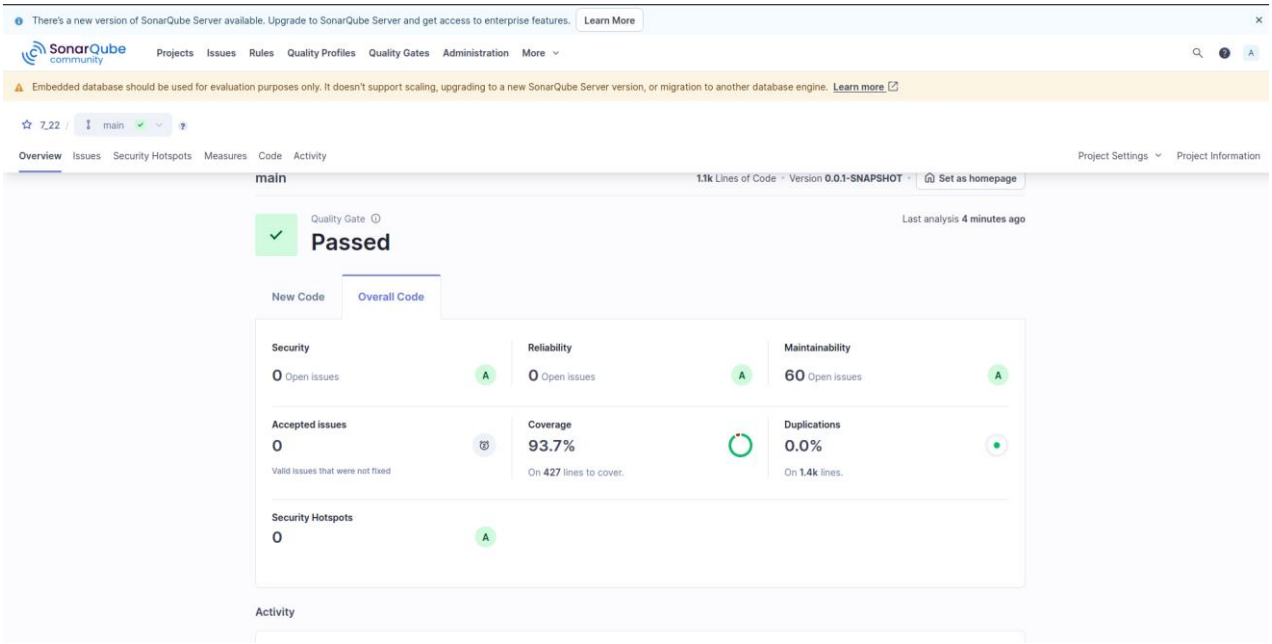
Usei K6 testes e lighthouse.

```
TOTAL RESULTS

checks_total.....................: 6       5.910713/s
checks_succeeded.................: 100.00% 6 out of 6
checks_failed....................: 0.00%   0 out of 6

✓ status é 200 para /restaurants
✓ duração da resposta < 500ms para /restaurants
✓ status é 200 para /reservations
✓ duração da resposta < 500ms para /reservations
✓ status é 200 para /meals
✓ duração da resposta < 500ms para /meals

HTTP
http_req_duration.....................................................: avg=4.4ms min=2.92ms med=4.13ms max=6.15ms p(90)=5.74ms p(95)=5.94ms
  { expected_response:true }...........................................: avg=4.4ms min=2.92ms med=4.13ms max=6.15ms p(90)=5.74ms p(95)=5.94ms
http_req_failed......................................................: 0.00% 0 out of 3
http_reqs............................................................: 3     2.955357/s

EXECUTION
iteration_duration...................................................: avg=1.01s min=1.01s  med=1.01s  max=1.01s  p(90)=1.01s  p(95)=1.01s
iterations...........................................................: 1     0.985119/s
vus..................................................................: 1     min=1      max=1
vus_max..............................................................: 1     min=1      max=1

NETWORK
data_received........................................................: 12 kB 12 kB/s
data_sent............................................................: 277 B 273 B/s
```

```
TOTAL RESULTS

checks_total.....................: 4       3.886929/s
checks_succeeded.................: 100.00% 4 out of 4
checks_failed....................: 0.00%   0 out of 4

✓ status é 201 para POST /restaurants
✓ duração da resposta < 500ms para /restaurants
✓ status é 201 para POST /meals
✓ duração da resposta < 500ms para /meals

HTTP
http_req_duration.....................................................: avg=13.59ms min=12.98ms med=13.59ms max=14.21ms p(90)=14.09ms p(95)=14.15ms
  { expected_response:true }...........................................: avg=13.59ms min=12.98ms med=13.59ms max=14.21ms p(90)=14.09ms p(95)=14.15ms
http_req_failed......................................................: 0.00% 0 out of 2
http_reqs............................................................: 2     1.943464/s

EXECUTION
iteration_duration...................................................: avg=1.02s  min=1.02s  med=1.02s  max=1.02s  p(90)=1.02s  p(95)=1.02s
iterations...........................................................: 1     0.971732/s
vus..................................................................: 1     min=1      max=1
vus_max..............................................................: 1     min=1      max=1

NETWORK
data_received........................................................: 645 B 627 B/s
data_sent............................................................: 487 B 473 B/s
```



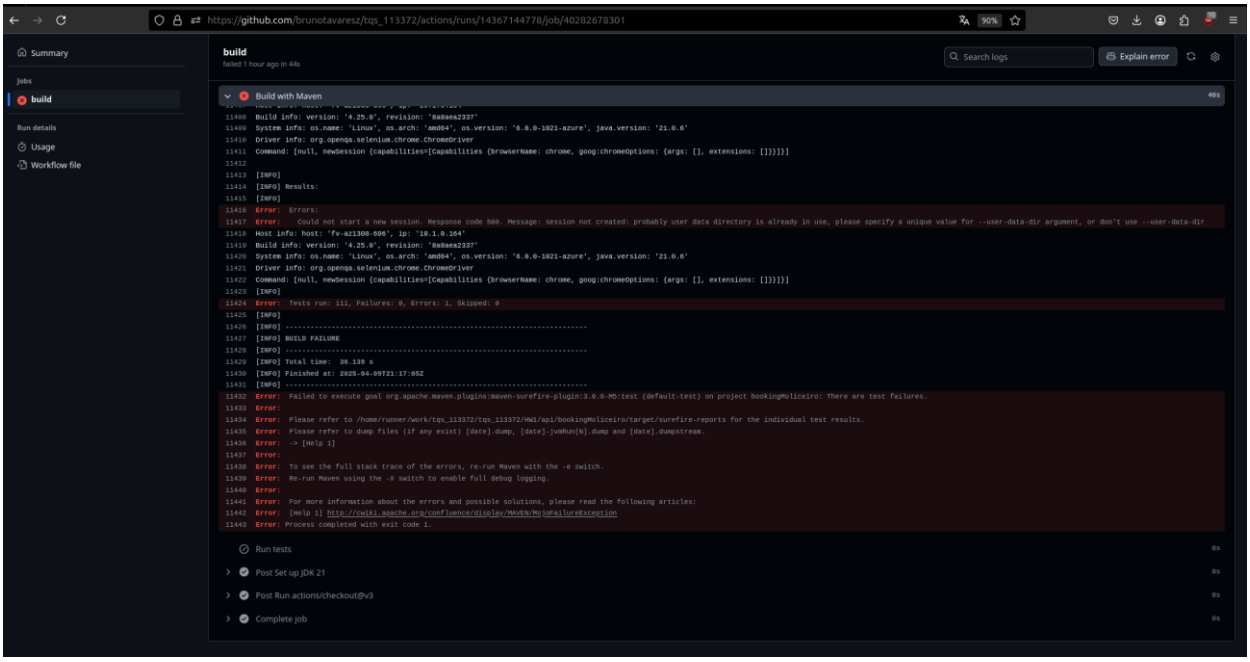
3.5 Analise da qualidade de código



Como implementei o sonarQube enquanto estava a fazer a aplicação ajudou-me a corrigir vários erros principalmente de segurança, um deles foi o uso de " new Random (); Make sure that using this pseudorandom number generator is safe here.” que consegui resolver usando uma ferramenta chamada “Secure Random”

3.6 Continuous integration pipeline [optional]

Implementei a pipeline (criando um workflow e o ficheiro .yml) mas como disse nas limitações deu erro no teste de selenium, algo que não costuma dar fora da pipeline.



4 References & resources

Project resources

Resource:	URL/location:
Git repository	https://github.com/brunotavaresz/tqs_113372
Video demo	https://youtu.be/-vSvMmShQv4
QA dashboard (online)	Usei SonarQube

Reference materials

External API: https://openweathermap.org/api
Guiões dos Labs de TQS