HW1: Mid-term assignment report

Cristiano Antunes Nicolau [108536], v2024-04-09

1 Introdução

1.1 Visão Geral

O presente relatório descreve o projeto individual desenvolvido como parte da disciplina de TQS (Teste e Qualidade de Software). O projeto consiste na implementação de um sistema de reserva de viagens de autocarros entre cidades, com a possibilidade de selecionar viagens de ida ou ida e volta. Além disso, o sistema inclui uma funcionalidade de cache para lidar com a taxa de câmbio aplicada aos preços dos bilhetes.

1.2 Limitações

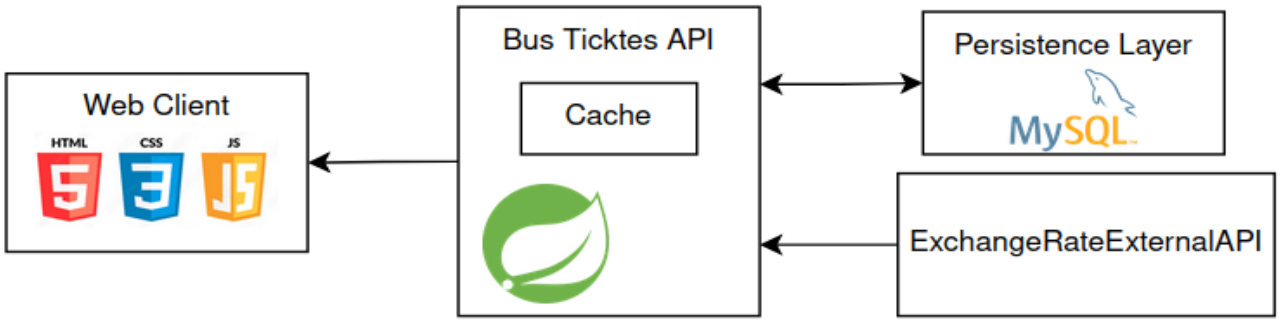
Uma das principais limitações é referente aos testes dos *repository* que deveria usar a base dados em memória do java, para lidar com dados de forma controlada e sem afetar a base dados de produção mas devido a vários erros não foi possível sendo assim utilizada a base dados de produção. Outra limitação é a não implementação de um CI pipeline, que seria algo a fazer como próxima tarefa. Além disso, se houvesse mais algum tempo poderiam-se implementar mais algumas features e ainda mais alguns testes para lidar com as mesmas.

2 Especificações do Produto

2.1 Espectro funcional e Interações suportadas

Ao utilizar a aplicação web, os utilizadores têm a possibilidade de selecionar um local de partida, um local de chegada e a data de partida para sua viagem. No caso de uma viagem de ida e volta, é ainda possível escolher a hora de chegada. Após essa seleção inicial, os usuários são redirecionados para a página de escolha de bilhetes, onde podem definir detalhes adicionais, como a moeda em que desejam visualizar o bilhete. Ao alterar a moeda, o valor da taxa de câmbio é armazenado em cache por um período de 20 minutos, reduzindo assim o número de solicitações à API externa. Por fim, os utilizadores concluem a transação inserindo os seus dados pessoais e escolhendo o seu lugar de viagem, adicionando por fim o bilhete à base de dados.

2.2 Arquitetura do Sistema



O sistema utiliza o *freecurrencyapi* para obter as taxas de câmbio entre duas moedas diferentes (a moeda base e a moeda de destino). Esses valores são armazenados em cache assim que são obtidos da API com um timer de 20 minutos, para que posteriormente não seja necessário buscar os dados novamente. O cliente web foi construído com HTML/CSS/JS. O backend foi desenvolvido em Java com o framework Spring Boot, que disponibiliza funcionalidades como serviços web RESTful e injeção de dependências.

2.3 Documentação API

A API possui trens endpoints principais: *trip, mark, exchange.* Cada um destes endpoints oferece métodos específicos para interagir com recursos, mas nem todos são utilizados no front-end. No *endpoint trip* é possível criar e dar delete de novas viagens, é ainda possível procurar através de id, origem e destino, e até por data. Em relação ao *endpoint mark*, também é possível criar e dar delete de novas marcações e é possível também procurar por id e tripid. Por fim, em relação ao *endpoint exchange* é usado para obter a taxa de câmbio entre as moedas selecionadas.

3 Garantia de Qualidade

3.1 Estrategia de Testes

No início, a intenção era adotar a estratégia de Desenvolvimento Orientado a Testes (TDD), onde eu começaria criando um esqueleto das classes que seriam implementadas. No entanto, devido a alguns erros relacionados à base de dados que surgiram, optei por desenvolver a aplicação primeiro e, em seguida, escrever os testes à medida que ia testando e realizando as correções necessárias na aplicação, nos services, controllers etc. Além disso, utilizei o SonarQube durante o desenvolvimento para facilitar a detecção de possíveis erros graves e melhorias no código.

Os componentes da API foram desenvolvidos e testados na seguinte ordem: cache, controller, service e repository.

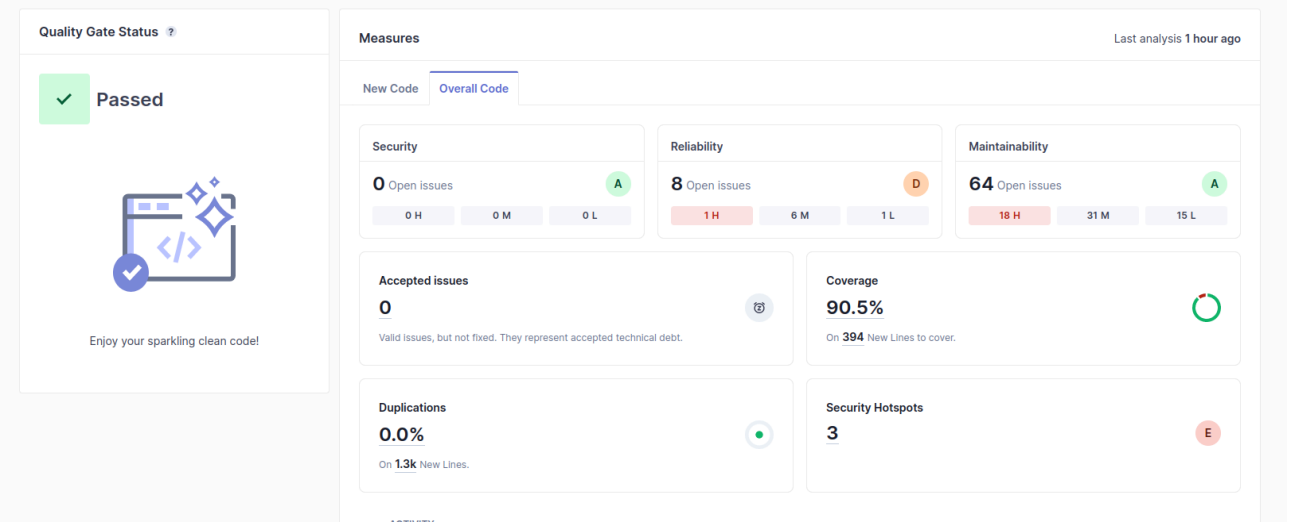
3.2 Testes Unitários e de integração

Para os testes e de integração foram usados o Junit 5, mockito e o MockMVC. Cada componente da API tem o seu próprio teste unitário. Na maior parte é usado o Mockito para isolar o objeto em testes, como acontece nos services. O MockMVC é usado para os testes do controller, de forma a controlar os pedidos do do controller.

3.3 Testes funcionais

Apesar de não estar funcional, testes foram escritos usando o Selenium e o Page Object Pattern de forma a testar os inputs e todo o front end.

3.4 Análise qualidade de código



De forma a analisar a qualidade do código, foi usado o SonarQube durante o desenvolvimento do projeto, de forma a acompanhar os bugs e possíveis problemas presentes na aplicação.

3.5 Continuous integration pipeline [optional]

Não aplicável.

4 Referencias & recursos

Recursos Projeto

Recursos:	URL:
Git repository	https://github.com/cristiano-nicolau/TQS_108536/tree/main/hw1
Video demo	https://youtu.be/aQj6cqA_YdA

Materiais usados

External API: https://app.freecurrencyapi.com/

Guiões Práticos

Materiais de apoio da UC