deti universidade de aveiro
departamento de eletrónica, telecomunicações e informática

TQS: Manual de Garantia de Qualidade

Manuel Diaz [103645], Pedro Coelho [104247], Diogo Silva [98644], Luca Pereira [97689]
v2023-06-01

1 Gestão do Projeto

1.1 Equipa e papéis

- **Team Coordinator:** Pedro Coelho
  Team Coordinator deve garantir uma distribuição justa das tarefas, tem de verificar regularmente se as tarefas estão a ser feitas dentro dos prazos estipulados e ainda resolver quaisquer problemas que aconteçam ao longo das iterações.

- **Product Owner:** Luca Pereira
  Tem de ter conhecimento geral acerca do projeto, tanto funcionalidades principais como desafios que se podem encontrar no futuro. Deve trazer ao de cima o ponto de vista dos *stakeholders* mas também informar os mesmos de atrasos e problemas encontrados no decorrer do projeto.

- **QA Engineer:** Diogo Silva
  Responsável por garantir a qualidade do projeto, garantindo a correta utilização das práticas de QA. Além disso, é responsável por utilizar ferramentas de análise de qualidade no projeto.

- **DevOps Master:** Manuel Diaz
  Tem como objetivo desenvolver a infraestrutura e respetivas configurações, garantido o funcionamento esperado.
  Deve ser o responsável pelo repositório garantindo o acesso a todos os membros ainda da preparação de outras tecnologias tais como a base de dados e outras infraestruturas.

- **Developer:** Pedro Coelho, Luca Pereira, Diogo Silva, Manuel Diaz
  Responsável pela produção e desenvolvimento do código.

1.2 Gestão do backlog ágil e divisão de tarefas

Para fazer o *backlog management* utilizamos a ferramenta JIRA, onde definimos as tarefas a serem realizadas através de *epics*.

A divisão do trabalho foi feita consoante o papel de cada um, indicado no ponto anterior, e distribuindo as restantes tarefas de igual forma por todos. À medida que as tarefas foram iniciadas ou terminadas, a sua informação foi atualizada.

2 Qualidade do código

2.1 Métricas de qualidade

De modo a garantir a qualidade do código utilizámos o SonarCloud para o analisar, para o qual definimos a seguinte *quality gate*:

- Percentagem de Linhas duplicadas inferior a 10%;
- Rating de Manutenibilidade com nota A;
- Rating de Confiabilidade com nota A;
- Cobertura dos testes superior a 80%

3 Pipeline de entrega contínua (CI/CD)

3.1 Workflow de desenvolvimento

O nosso workflow de desenvolvimento foi efectuado com recurso ao Github. Cada um dos diferentes módulos do nosso projeto foi desenvolvido numa *branch* diferente e à medida que as funcionalidades foram implementadas foram feitos *pull requests* para a *main*. Estes eram revistos pelo *QA Engineer* que os aprovava ou não.

3.2 CI/CD pipeline e ferramentas

A definição da pipeline CI/CD foi feita com recurso ao Github Actions.

Sempre que é efectuado um *push* para a *main* que afete a pasta da plataforma mailMover são efectuados os testes associados à mesma, assim como a análise de código do SonarCloud. Por sua vez, quando é feito um *push* para a *main* que afete a pasta do *frontend* da *store* são efectuados os testes funcionais ao seu *frontend*.

Uma vez que acabámos por não conseguir efetuar o deploy da nossa solução na VM, também não fizemos CD.

4 Testes de software

4.1 Estratégia de teste

A estratégia utilizada para o desenvolvimento da aplicação foi orientada a testes, seguindo a abordagem *Test-Driven Development* (TDD), o que significa que os testes foram desenvolvidos antes mesmo de se começar a desenvolver o código, com base nos requisitos e naquilo que queríamos para a nossa aplicação, desde início.

Este tipo de estratégia colocou a qualidade do software em primeiro plano desde o início do desenvolvimento do sistema, pois a equipa concentrou-se em definir claramente o comportamento desejado e garantir que os requisitos fossem atendidos.

Assim, conseguimos obter bastantes vantagens significativas, como:

- Deteção precoce de problemas;
- Maior confiança no código;
- Melhor compreensão daquilo que o sistema requer;
- Facilidade de manutenção
- Possibilidade de reestruturação de código segura

4.2 Testes funcionais

Os testes funcionais de ambos os *frontends* do admin e da página de ACP's foram feitos usando cypress. Ainda que o Selenium tenha sido a ferramenta usada para os testes funcionais durante as aulas práticas, a escolha de Cypress era a mais adequada visto ter uma integração nativa com React e permitir uma execução rápida e mais simples dos testes.

Os testes feitos com o Cypress percorreram todas as funcionalidades de cada página como o Login, filtros, adição de ACP e etc. Nestes testes também se aproveitou para fazer alguns testes unitários do frontend, como por exemplo, testar se o comportamento de um botão está de acordo com esperado, o que não foi possível fazer nos testes unitários específicos que vamos discutir mais adiante. A capacidade de realizar testes unitários foi também uma das razões da escolha de usar Cypress.

Também foram feitos testes funcionais para o *frontend* da *store*, em que utilizámos o Selenium, nomeadamente para testar o login, a navegação entre páginas e a compra de um produto, porém, não conseguimos com que estes testes sejam validados pela CI/CD *pipeline* da *store*, talvez porque a realização dos mesmos foi feita numa máquina que se encontra com os drivers do Chrome desatualizados.

4.3 Testes unitários

Os testes unitários foram utilizados para testar a mudança de estado das encomendas e garantir que não havia mudanças não aceitáveis e para testar a camada de persistência de dados (*repository*), sendo que para ambos utilizamos a ferramenta Junit.

Foram também utilizados para testar o *service*, onde tirámos partido da ferramenta Mockito de maneira a ser possível testar os métodos independentemente da camada de persistência, daí ter se utilizado Mocks.

Alguns testes unitários foram realizados no frontend utilizando a ferramenta nativa de testes do React, porém houve uma dificuldade em dar mock ao servidor para testar componentes para além da página de login. Como discutido acima, para colmatar esta falha foram implementados alguns testes unitários com Cypress sobre os componentes não testados pelo React Test.

4.4 Testes de sistema e integração

Para finalizar, no que toca aos testes do sistema e de integração, estes foram feitos com recurso ao *Mock WebMvc* de maneira a testar a camada do *Controller*, com e sem ligação direta à base de dados. Para tal, para cada endpoint foram realizados um conjunto de verificações para garantir que o funcionamento era o esperado.