deti universidade de aveiro departamento de eletrónica, telecomunicações e informática

TQS: Quality Assurance manual

Ricardo Rodriguez [98388], Pedro Sobral [98495], Daniel Figueiredo [98498], Eva Bartolomeu [98513]
v2022-05-23

1 Project management

1.1 Team and roles

Papel	Responsável	Responsabilidades
Team Coordinator	Pedro Sobral	Gerir a distribuição de tarefas entre os elementos da equipa; Garantir um bom planeamento no projeto.
Product owner	Eva Bartolomeu	Conhecimento do produto e do domínio da aplicação; Aceita os incrementos da solução.
QAEngineer	Ricardo Rodriguez	Sugere práticas de garantia de qualidade; Aplica ferramentas para medir a qualidade da implementação.
DevOps master	Daniel Figueiredo	Responsável pela infraestrutura de desenvolvimento e produção e configurações necessárias; Garante que o framework de desenvolvimento funcione corretamente.
Developer	Pedro Sobral, Eva Bartolomeu, Ricardo Rodriguez, Daniel Figueiredo	Desenvolvimento das tarefas.

1.2 Agile backlog management and work assignment

Por cada iteração, o grupo tem de se reunir para debater o que será feito, e para o Team Coordinator atribuir as tarefas da iteração por cada elemento do grupo. De forma a agilizar e persistir esta informação a mesma é mantida através do Jira. No Jira o trabalho foi dividido por user stories, criando assim um epic para cada uma, cada user story tem uma lista de tarefas a serem feitas, que são atribuídas aos elementos do grupo como mencionado no segmento anterior. No final de cada iteração a equipa tem de analisar o que foi feito, para concluir o que está feito dentro da iteração em que está.

2 Code quality management

2.1 Guidelines for contributors (coding style)

O código foi feito com o intuito de ser o mais simples possível, ou seja, ser de fácil compreensão por outros contribuidores que pudessem querer fazer algumas alterações. Deste modo, sempre que possível foi-se fazendo uma leitura de alguns excertos de código e caso possível fazer a reformatação do mesmo, também foram deixados alguns comentários ao longo do código.

Em termos de deployment do código, foi utilizado o Github e foram criados vários branches para se tratar de novas features, issues e fixs que poderiam aparecer. A nomenclatura padrão dos branches foi feita da seguinte maneira:

<feature/issue/fix>/<módulo que estamos a trabalhar>/<descrição>

Os testes possuem, também, um nome padrão que permite facilmente entender daquilo que tratam e o que vai ser testado, bem como o que se pretende atingir:

test<functionality>_then<expected outcome>

Em termos de nomenclatura de variáveis e afins, utilizamos as regras de estilo Java.

Desde o início, o grupo decidiu utilizar exceções que realmente dessem a entender a que se referiam e que identificassem logo o problema. Isto significa que vamos apanhar exceções específicas e não ignorá-las nem apanhar exceções genéricas.

Exceção	**Problema**
ConflictException	Gerada quando existe algum conflito, por exemplo, quando é adicionado um produto que já existe.
InvalidCredentialsException	Gerada quando são passadas credenciais erradas.
ResourceNotFoundException	Gerada quando o recurso a que se está a tentar aceder não é encontrado.
PersonNotFoundException	Gerada quando é feito um login de um rider que não existe.
InvalidReviewException	Gerada quando é feita uma review inválida.

2.2 Métricas de Qualidade de Código

Para termos uma análise estática do código por nós desenvolvido, utilizamos o SonarQube (integrado no GitHub Actions), uma vez que é uma ferramenta com capacidade de disponibilizar avaliações do código, sendo capaz de fazer a análise do mesmo e reportar code smells, vulnerabilidades, bugs, entre outras.

Metric	Operator	Value
Coverage	is less than	65.0%
Duplicated Lines (%)	is greater than	3.0%
Maintainability Rating	is worse than	A
Reliability Rating	is worse than	A
Security Rating	is worse than	A

3 Pipeline de Entrega Contínua (CI/CD)

3.1 Workflow de Desenvolvimento

Para que o trabalho no projeto seja feito de forma organizada e eficiente, é usada a plataforma *Jira* para seguir a metodologia Agile. Esta disponibiliza uma série de funcionalidades que a destaca em relação a outras plataformas como, por exemplo, a possibilidade de configurar um projeto consoante as suas necessidades, métricas personalizadas que avaliam o *workflow* de toda a equipa, monitorização e criação de tarefas de forma rápida e eficaz, adição de plug-ins, entre outros.

Além destas caraterísticas mencionadas anteriormente, é possível integrar o projeto do Jira com o repositório GitHub, permitindo um *workflow* fluído entre a plataforma de monitorização e a de desenvolvimento. Assim, é possível sincronizar *issues*, *commits, pull requests*, associar *issues* a Epics e muitas outras ações que facilitam o desenvolvimento do projeto.

O mapeamento das *user stories* com o *workflow* de desenvolvimento é feito através das *issues*, uma vez que ambas estão diretamente associadas. Através do Jira, é possível criar, ou associar, novos branches ou commits relativos a um *issue*.

Para evitar a descontextualização interna da equipa em relação ao projeto e para uma identificação fácil dos erros no desenvolvimento, todos os *pull requests* realizados por um membro terão de ser revistos por um membro da equipa. Este terá de ser analisado pelos *reviewers* de forma cautelosa e poderá ter comentários precisos que dão feedback construtivo. Da mesma forma, os pull requests devem, também, ter uma descrição detalhada das funcionalidades implementadas.

Para as *issues* serem resolvidas, estas têm de cumprir uma série de requisitos estabelecidos previamente que indicam quando é que a mesma pode ser registada como concluída. Desta forma, as *user stories* têm um DOD (Definition Of Done), que podem incluir critérios como, por exemplo, a conclusão dos testes unitários para essa funcionalidade, a

documentação feita, entre outros.

3.2 CI/CD pipeline and tools

Pipeline CI:
- Corre testes spring boot engine service
- Corre testes spring boot specific service
- Corre o analizador de código SonarQube

Pipeline CD:
- Deployment da parte do specific
- Deployment da parte do engine

A pipeline CD é realizada com recurso a Docker containers.

Pipeline CI:

A pipeline de CI executa os testes dos serviços Spring Boot, bem como faz uma avaliação do código através do SonarQube. O CI é realizado através do Github Actions, onde é definido que após commits com pull requests abertos no GitHub são efetuados testes, de forma a verificar se os requisitos são cumpridos.

Pipeline CD:

O CD é realizado também através do Github Actions. Aqui definimos que depois de um commit para a branch main são executados comandos que fazem build dos containers Docker e executam os containers.

4 Software testing

4.1 Overall strategy for testing

A garantia de qualidade no desenvolvimento do produto espera, sempre que possível, seguir uma estratégia de BDD e de TDD. Primeiro, espera-se que os detalhes das *user-stories* sejam descritos através de cenários, o que precisa de uma identificação precisa dos requisitos do sistema e melhora o conhecimento geral da equipa sobre o projeto. Quando as *user-stories* e as respectivas *features* estiverem bem definidas, deve-se desenvolver os testes que vão assegurar o bom funcionamento das features a um nível técnico e de desenvolvimento.

As ferramentas de testagem usadas foram: JUnit, AssertJ, Mockito, Selenium, Jacoco, REST-Assured e o SonarQube.

4.2 Functional testing/acceptance

Por motivos de tempo, infelizmente, não conseguimos implementar testes funcionais, embora saibamos que estes apresentam grande importância para a testagem da aplicação web.

4.3 Unit tests

Os testes unitários têm como objetivo testar uma parte isolada do sistema, de modo a garantir que estas partes, mesmo sozinhas, estão a funcionar corretamente. Deste modo, fizemos testes que cobrissem o comportamento dos repositórios, serviços e controllers.

Para testar os repositórios, foi utilizada a anotação @DataJpaTest, com a base de dados em Testcontainers, de modo a permitir persistência. Fizeram-se testes para todos os métodos implementados por nós que se encontram presentes nas classes dos repositórios e também foram testados vários casos de erro.

Para os testes dos serviços, utilizou-se a extensão Mockito, para se conseguir testar esses métodos independentemente da camada de persistência. Mais uma vez, foram testados os nossos métodos mais alguns casos de erro.

Por fim, quanto aos testes dos controllers utilizamos RestAssuredMockMvc, permitindo utilizar mocks no lugar dos serviços, isolando o comportamento dos controllers. Tal como nos repositórios e serviços, testamos os testes que cobrem os métodos mais vários casos de erro.

4.4 System and integration testing

A realização de testes de integração permite entender se todos os componentes que envolvem a resposta a um pedido feito, estão de acordo com o esperado, ou seja, se não dão nenhum problema. Posto isto, temos o Spring Mvc que mapeia os pedidos http para o controller respectivo ao pedido.

É necessário a confirmação que todos os pedidos que um controller venha a receber, fazem o aquilo que é suposto fazerem de acordo com o objetivo de cada endpoint.

Para cada controller temos um teste com a terminologia TemplateIT, onde são analisadas e verificadas as diferentes formas de execução, sendo portanto verificado se o pretendido realmente foi alcançado.