TQS: Quality Assurance manual

Marcel de Araújo Santos Souza - 101043
v2023-05-22

1 Project management

1.1 Team and roles

Todo o projeto foi realizado pelo aluno Marcel Santos Souza.

1.2 Agile backlog management and work assignment

Como foi um projeto de realização individual, o BackLog não foi implementado e os princípios de spaced-delivery do AGILE não puderam ser implementados também, uma vez que são necessários pelo menos 2 integrantes para tal.

2 Code quality management

2.1 Guidelines for contributors (coding style)

O projeto utiliza-se de linters para a organização das files que foram rodados antes de serem commitados na branch. Como foi realizado por apenas um estudante, não foi necessário realizar o trabalho com “branch model” como prevê a metodologia AGILE.

3 Continuous delivery pipeline (CI/CD)

3.1 Development workflow

Mais uma vez, infelizmente, como o projeto foi desenvolvido por apenas uma pessoa, nao julguei, como desenvolvedor do projeto,necessário criar mais do que uma branch para o porpósito. Sendo assim o workflow constituido apenas da branch main com commits que utilizam mais de uma linha para explicar o que foi desenvolvido na branch.

A definição de finalizado para o projeto foi: uma vez que a tarefa foi completada, revisada, passou pelo “lint” e por todos os testes, foi adicionado ao pipeline de CI e CD. Ou seja, uma vez que toda a tarefa foi finalizada e pode ser acessada pelo utilizador final.

3.2 CI/CD pipeline and tools

A ferramenta utilizada para CI foi o Github Actions. A partir deste, é possível isolar o projeto ReactJS e o SpringBoot. Estes rodam um build todas as vezes que são enviados para a branch online do Github por meio do git. O resultado dos testes em SpringBoot, tais como quaisquer erros de utilização e instalação de bibliotecas tanto no ReactJS quanto no SpringBoot são enviados ao email do utilizador responsável pelo commit.

A ferramenta para CD utilizada foi o Docker (Compose). Por meio deste, o deployment é facilitado e pode-se trabalhar com isolamento dos serviços. Assim, Database MySql, ReactJS e o SpringBoot rodam em containers distintos e autônamos, no entanto estão 100% integrados no que diz respeito a sua utilização como aplicação completa.

4 Software testing

4.1 Overall strategy for testing

A estratégia geral de desenvolvimento de testes para o projeto segue uma combinação de abordagens de teste unitário e teste de integração. Os testes unitários são focados em testar componentes individuais ou unidades de código de forma isolada. Já os testes de integração verificam a interação entre diferentes componentes e sua integração no sistema. A estratégia de teste inclui o uso de frameworks de mock, como o Mockito, para criar substitutos de teste para as dependências e garantir o isolamento durante os testes. Os testes são escritos usando o framework JUnit 5 e fazem uso de anotações como @Test, @BeforeEach e @Mock para definir e configurar os casos de teste. A estratégia adota uma abordagem de desenvolvimento orientado a testes (TDD), em que os casos de teste são usados para orientar a implementação dos recursos.

4.2 Functional testing/acceptance

Os testes funcionais são escritos como testes unitários para a classe MeetingController. Esses testes simulam interações do usuário e verificam o comportamento esperado do sistema. Eles cobrem cenários como criação de reuniões, recuperação de reuniões por e-mail, atualização de reuniões e exclusão de reuniões. Os testes usam a classe MockMvc do Spring Framework para realizar solicitações HTTP e verificar as respostas. Eles validam os códigos de status HTTP e o conteúdo das respostas para garantir o correto funcionamento do sistema do ponto de vista do usuário.

4.3 Unit tests

4.4 Os testes unitários são usados para validar o comportamento de unidades individuais ou componentes específicos do código. Eles são escritos usando o framework JUnit 5 e fazem uso de anotações como @Test e @Mock. O Mockito é utilizado para criar mocks das dependências e isolar as unidades em teste. Os testes unitários abrangem diferentes cenários, como criação de reuniões, recuperação de reuniões, atualização de reuniões e exclusão de reuniões. Eles fazem asserções comparando os resultados reais com os valores esperados ou usando asserções fornecidas pelo framework de teste.

4.5 System and integration testing

Os testes de sistema e integração verificam a interação entre diferentes componentes e a correta integração do sistema. Esses testes são escritos como testes unitários para a classe UserController. Eles utilizam frameworks de mock, como o Mockito, para simular as dependências e se concentram em testar o comportamento do controlador em conjunto com o UserRepository. Os testes simulam solicitações HTTP e verificam as respostas para garantir a correta integração dos componentes do sistema. Eles abrangem cenários como recuperação de todos os usuários, recuperação de um usuário por ID, exclusão de um usuário, recuperação de contatos, adição de um contato e remoção de um contato.