Prática de Teste em OSS 1 – Estratégia de Testes

FGA 0238 - Testes de Software			
Turma	01	**Semestre**	<2024.1>
Equipe	Nevermind		

Nome	**Matrícula**
Elias Faria de Oliveira	221007706
Yan Lucas Souza Guimarães	222006220
Daniel dos Santos Barros de Sousa	211030980
João Eduardo Pereira Rabelo	180053299

1 Projeto

Descrição:

Projeto: MEC – Energia

O sistema MEC-Energia tem por objetivo auxiliar as instituições de ensino superior (IES) a gerenciar e avaliar a adequação de contratos de conta de energia elétrica a partir do registro das faturas mensais de energia, gerando relatórios de recomendações de ajustes nos contratos visando economia de recursos. Documentação sobre os testes e os testes já realizados pelos desenvolvedores.

2 Tipos e Níveis de Teste

Foi criado o 'tests_utils', um conjunto de módulos que contêm funções dedicadas à criação de objetos de teste para os diversos modelos do sistema. Cada função dentro desses módulos é responsável por criar e salvar um objeto no banco

de dados, utilizando os dados fornecidos em dicionários que representam os diferentes cenários de teste para as várias partes do sistema. Ao agrupar esses dados e as funções de criação em um módulo separado, facilita-se a reutilização e a manutenção dos testes em todo o projeto.

Os testes funcionais utilizam desse módulo para configurar o ambiente de teste antes de executar os testes em si. Isso ajuda a manter os testes independentes e isolados uns dos outros. Todos os testes utilizados nesse projeto são automatizados, o que é facilitado pelo uso da biblioteca pytest e pytest_django.

A aplicação da pirâmide de testes esta presente no projeto, com uma base sólida de testes de unidade, seguida por testes de integração e sistema, embora a presença limitada de testes de aceitação possa ser uma área a ser considerada para uma cobertura de teste mais completa.

Os testes não funcionais utlizados no projeto são todos na parte de prevenção de bugs e estabilidade. Focando na segurança, desempenho, manutenibilidade e confiabilidade do sistema.

3 Automação de Testes

Foi utilizado a pirâmide de testes, o que implica a presença de testes unitários, de integração e de sistema. Todos esses testes foram automatizados, aproveitando as capacidades da biblioteca pytest e pytest-django. A automatização de testes se da pelo sistema de pipeline do gitlab, onde toda vez que um PR é criado é executado um script que realiza os testes sobre a versão da qual foi requisitado de se fazer o merge.

4 Ciclo de Vida das Issues

O projeto Mec - Energia possui um template de issue a ser usado, mas é possível notar que nem sempre é utilizado dentro da especificação das issues, sendo possível observar que algumas issues possuem uma documentação clara do que se deve fazer, até mesmo com os requisitos de aceitação.

Diante disso, o que difere as issues entre si são as tags atribuídas a cada uma delas, onde através delas nota-se a diferenciação do tipo de issue, se é

desenvolvimento, bug etc. Como também da equipe responsável por ela. Porém, mesmo com as diferenciações atribuídas pelas tags, é possível notar que todas as issues seguem um ciclo padrão

Com respeito aos testes, em issues bem documentadas, é descrito que se deve realizar os testes durante a issue, e em issues que o foco são os testes, ocorre o mesmo padrão de ciclo de vida de issues normais.

5 Papeis e Responsabilidades

De acordo com as commits e issues gerados no projeto, foram identificados os seguintes papeis no desenvolvimento:

Owner:

O "Owner", ou proprietário, possui o mais alto nível de permissões, tendo controle absoluto sobre o projeto. Eles podem realizar uma variedade de ações administrativas, como adicionar ou remover membros, gerenciar permissões e até mesmo transferir a propriedade do projeto.

Maintainer:

Os "Maintainers" são responsáveis pela edição e gerenciamento do projeto, podendo adicionar, modificar e remover arquivos, além de gerenciar issues e solicitações de mesclagem. Embora possuam um amplo poder de edição, não têm autoridade administrativa completa.

Guest:

Os "Guests", ou convidados, têm permissões restritas, permitindo-lhes apenas acesso de visualização ao projeto. Eles podem visualizar o código fonte e outros recursos, mas não podem fazer alterações, criar issues ou solicitações de mesclagem.

6 Ferramentas de Teste

Os testes da API foram feitos com Pytest como base para testes e pytest-django para acesso ao banco de dados. Como suporte para a execução de testes, é utilizado o GitLab CI/CD, que automatiza a execução dos testes toda vez que um PR é criado.

7 Métricas de Teste

Tendo em vista métricas disponibilizadas na documentação do projeto no GitLab, vemos que o projeto MEC Energia está com uma cobertura do código em 67% com a criação e utilização para os testes utilizando a biblioteca Pytest do Python, que possibilita gerar um relatório quantificando e denotando o percentual de cobertura de testes no projeto.

Nota-se que falta na documentação dom projeto algumas métricas importantes como a taxa de falhas dos testes e alguns outros como tempo médio para detectar e corrigir os defeitos.

8 Necessidade de Testes

Uma área que requer maior cobertura é a de testes de aceitação. Os testes de aceitação são essenciais para garantir que o sistema atenda aos requisitos do usuário e funcione conforme o esperado em cenários do mundo real. Portanto, é importante investir na criação e execução de testes de aceitação para complementar os testes existentes e garantir uma cobertura abrangente de todos os aspectos do sistema.

9 Análise da Estratégia e Situação dos Testes do Projeto

A estratégia de testes do projeto MEC-Energia é abrangente e automatizada, seguindo a pirâmide de testes com os testes de unidade, integração e sistema. Os testes são executados automaticamente pelo GitLab coma aplicação da prática CI/CD, garantindo a detecção precoce de falhas e mantendo a estabilidade do sistema. No entanto, há uma necessidade identificada de melhorar a cobertura de testes de aceitação para garantir que o sistema atenda adequadamente às expectativas dos usuários em diferentes cenários. A cobertura de código atual é de 67%, refletindo um bom nível de testes implementados, mas deve ser continuamente monitorada e melhorada ao longo do tempo.

10 Recomendações

Para melhorar a qualidade do software e o processo de desenvolvimento do projeto MEC-Energia, algumas recomendações podem ser implementadas. É fundamental expandir a cobertura de testes de aceitação para garantir que o sistema atenda às expectativas dos usuários em diferentes cenários. Além disso, é importante implementar testes de regressão para evitar a introdução de falhas em funcionalidades existentes durante o desenvolvimento ágil, melhorar a documentação das issues, enfatizar a colaboração entre desenvolvedores e testadores e monitorar continuamente a cobertura de código são passos cruciais. Também é recomendado automatizar testes de desempenho e segurança e promover a reutilização de testes para otimizar a eficiência e a manutenção. Essas ações contribuirão para um desenvolvimento mais robusto e confiável do sistema MEC-Energia.