Prática de Teste em OSS 1 – Estratégia de Testes

FGA 0238 - Testes de Software			
Turma	01	**Semestre**	2023.2
Equipe	Testeiros de Jeová		

Nome	**Matrícula**
Lucas Andrade Lobão	202023888
Bruno Tarquinio Silva	211061494
Leandro Souza da Silva	211031109
Guilherme Soares Rocha	211039789
João Pedro Rodrigues Morbeck	202063300
João Lucas Miranda de Sousa	202016604
Maykon Júnio dos Santos Soares	211030863

1 Projeto

O que nós escolhemos para contribuir é o MEC Energia, o sistema é projetado para apoiar as instituições de ensino superior (IES) na gestão e avaliação da conformidade dos contratos de fornecimento de energia elétrica. Isso é feito através do registro das faturas mensais de energia e da geração de relatórios contendo recomendações de ajustes contratuais, com o objetivo de promover a economia de recursos.

https://gitlab.com/lappis-unb/projects/mec-energia

https://lappis-unb.gitlab.io/projects/mec-energia/documentacao/

2 Tipos e Níveis de Teste

Os níveis de testes identificados no projeto são testes de unidade e testes de integração, não é aplicada a pirâmide de testes por não haver testes de sistema e de aceitação, de fato na documentação oficial não é citado nenhum tipo de teste, os únicos testes identificados são os que são rodados através de integração contínua.

Os testes de unidade estão relacionados às unidades mais básicas do software, como a validação de métodos, funções e partes do código que podem ser isoladas sem a interferência de outros artefatos, já os testes de integração estão relacionados à conexão com o banco de dados.

Poucos são os testes de segurança, o principal e mais importante é o teste de autenticação, que permite que partes sensíveis do software estejam restritas aos usuário com permissão suficiente, a arquitetura já é suficiente para trabalhar a segurança, mas é importante testá-la para confirmar sua eficácia.

3 Automação de Testes

A equipe de desenvolvimento optou pela utilização de testes automatizados durante as fases de Push e Merge nos repositórios da API e do Front-end. Com isso, a equipe estabeleceu testes para avaliar a qualidade da escrita, instalação e compilação dos códigos em pipelines definidas em workflows presentes na pasta raiz dos repositórios.

4 Ciclo de Vida das Issues

O ciclo de atividades de uma issue pode variar dependendo do contexto em que ela é usada. Podem haver variações de acordo com o tipo de projeto que está sendo trabalhado. Um ciclo de issue pode incluir as seguintes etapas:

1. **Criação:** É criado uma issue para uma determinada tarefa, podendo ser para correção de bugs, uma melhoria de uma funcionalidade ou qualquer atividade que precise ser abordada.

2. **Atribuição:** Após ser criada essa issue, ela será atribuída para algum desenvolvedor do projeto, com a intenção de resolvê-la.
3. **Desenvolvimento:** O membro da equipe passa a trabalhar na tarefa. Isso pode envolver codificação, correções ou outra ação necessária para resolver o problema ou completar a tarefa.
4. **Teste de Unidade:** Após a conclusão do desenvolvimento, o autor da "issue" realiza testes de unidade para verificar se a funcionalidade ou correção atende aos requisitos e não introduziu novos bugs.
5. **Testes de Integração:** Após a revisão do código e a aprovação, a "issue" é integrada ao ambiente de desenvolvimento para testes de integração, onde a funcionalidade é testada em conjunto com outros componentes do sistema.
6. **Encerramento:** Uma vez que a "issue" tenha sido resolvida, testada e aprovada, ela é fechada. Isso indica que a tarefa foi concluída com sucesso e não requer mais ação.

5 Papéis e Responsabilidades

Os papéis e as responsabilidades dentro do projeto MEC Energia são distribuídos da seguinte maneira:

- **Desenvolvedor** : Responsável pela criação e aprimoramento de funcionalidades, bem como pela correção de erros no software.
- **Testador**: Encarregado de realizar testes para assegurar o funcionamento correto do software.
- **Revisor** : Responsável por examinar o código-fonte e os testes para assegurar que estejam em conformidade com os requisitos especificados.

6 Ferramentas de Teste

GitLab CI/CD - Integração contínua

Pytest - API

Jest - Front-end

7 Métricas de Teste

- Cobertura de testes:

Utilizar métricas de cobertura de testes, como cobertura de instruções, ramificações ou caminhos para garantir que diferentes partes do código sejam testadas adequadamente. Isso ajuda a identificar áreas do código que precisam de testes adicionais

- Taxa de erros de teste:

Usar a taxa de erros de teste para monitorar como está a eficiência dos testes. Se a taxa de erros de teste for alta, provavelmente é necessário revisar e melhorar os casos de teste

- Tempo de execução de testes:

Registrar o tempo de execução dos testes para avaliar o desempenho e a eficiência dos testes automatizados. Isso ajuda a visualizar possíveis otimizações nos testes.

8 Necessidade de Testes

Testar um sistema é fundamental por várias razões, que incluem garantir a qualidade do software, reduzir riscos, melhorar a confiabilidade e desempenho, entre outras coisas.

Nesse sentido, com base em uma inspeção nas pastas do projeto, tanto da parte Web quanto na própria API foi possível identificar alguns pontos que podem sim receber mais testes (Lembrando que com uma análise mais detalhada, pode ser que encontraremos muitos outros pontos para testar). Dito isso, os principais pontos observados e que em nossa visão precisam de mais teste são:

1. Toda a parte de lançar uma fatura, mas especialmente os campos de **Demanda** e **Consumo**.
2. A funcionalidade de editar uma unidade, mais especificamente o campo de "**tensão contratada**"
3. Editar distribuidora, também pode receber alguns teste, em especial o campo de **CNPJ** da empresa
4. Toda a parte de cadastro de uma nova distribuidora

Futuramente, alguns testes a mais podem ser implementados para a parte de login e usuários, mas no momento atual do projeto, acreditamos que não seja tão necessário.

9 Análise da Estratégia e Situação dos Testes do Projeto

Análise:

A estratégia atual de testes do projeto apresenta aspectos positivos e áreas que requerem atenção. A utilização de testes de unidade e integração é fundamental para verificar a funcionalidade e a interação das partes do sistema, o que contribui para a qualidade do código. A automação de testes durante as fases de Push e Merge é uma prática que promove a consistência e a rapidez na identificação de problemas.

O ciclo de vida das issues está bem definido, o que facilita o acompanhamento das atividades de desenvolvimento e teste. As métricas de teste, como a cobertura de testes e a

taxa de erros, são monitoradas, proporcionando insights valiosos para a melhoria contínua da estratégia de testes.

Riscos e Preocupações:

Os principais riscos e preocupações para a execução de testes no projeto são os seguintes:

1. **Dependência da Integração Contínua:** A estratégia atual depende fortemente da integração contínua para execução de testes. Qualquer falha no ambiente de CI/CD pode afetar a qualidade dos testes e a estabilidade do projeto. É importante considerar alternativas ou redundâncias para mitigar esse risco.

2. **Falta de cobertura de testes:** A cobertura de testes é uma métrica que indica a proporção do código que é testada. Uma cobertura de testes baixa pode indicar que existem áreas do código que não estão sendo testadas adequadamente, o que pode aumentar o risco de bugs.

Recomendações:

- **Medir a cobertura de testes:** Isso pode ajudar a identificar áreas do código que precisam de mais testes.
- **Otimizar os testes automatizados:** Isso pode ser feito identificando testes que são lentos ou que falham com frequência.

A implementação dessas recomendações ajudará a melhorar a qualidade do software e a reduzir os riscos associados à execução dos testes.