PLANO DE TESTE PARA "*Crater*"

1. INTRODUÇÃO

Este plano de teste define as estratégias de teste, processos, fluxos de trabalho e metodologias que serão utilizados no projeto *Crater*. O objetivo é garantir que o software atenda aos requisitos funcionais e não funcionais definidos. O *Crater* é um sistema de gestão financeira voltado para pequenas e médias empresas e freelancers, permitindo a criação, gerenciamento e envio de faturas, orçamentos e estimativas. O sistema também facilita a gestão de despesas, recebimentos e clientes, além de oferecer funcionalidades para monitorar o fluxo de caixa e gerar relatórios financeiros detalhados.

1.1 ESCOPO

1.1.1 No escopo

- Geração de relatórios financeiros detalhados.
- Personalização de templates de faturas e orçamentos.
- Configurações de impostos, moeda e idioma.
- Integração com gateways de pagamento.
- Sistema de autenticação para proteção de dados.
- Backup automático e manual dos dados financeiros.

1.1.2 Fora do escopo

- Testes de desempenho em escala massiva.
- Funcionalidades que não são parte do MVP (Produto Mínimo Viável).

1.2 OBJETIVOS DE QUALIDADE

Os objetivos do nosso projeto de teste incluem:

- Garantir que o Crater esteja em conformidade com os requisitos funcionais e não funcionais.
- Assegurar que o Crater atenda às especificações de qualidade definidas pelos clientes.
- Identificar e corrigir defeitos antes do software entrar em operação.

1.3 PAPÉIS E RESPONSABILIDADES

- Melhorar e aumentar os casos de testes unitários, isolando suas dependências
- Implementar casos de teste de integração
- Indicação das medidas de cada atributo de qualidade da ISO 25010 seguindo uma escala. Justificar as decisões para indicação das medidas.
- Implementar testes de sistema considerando requisitos funcionais E pelo menos um atributo de qualidade (último opcional)
- Projetar e melhorar o conjunto de casos de teste, utilizando as técnicas:
    - Funcional
    - Estrutural (ao menos 80% de cobertura no critério todas-arestas das classes não CRUD da entrega 1)
    - Baseada em Defeitos (80% de escore de mutação nas mesmas classes do teste estrutural das classes não CRUD da entrega 1)

Os papéis de cada um foram divididos dentre os cinco membros do grupo, havendo apoio em todos os requisitos do trabalho atribuído, como dito durante a apresentação.

2. METODOLOGIA DE TESTE

2.1 Visão Geral

A metodologia de teste selecionada para o projeto será Ágil, permitindo adaptações rápidas conforme o desenvolvimento progride.

2.2 Fases de Teste

- Teste Unitário: Garantir que cada unidade do software funcione corretamente.

- Teste de Integração: Verificar a interação entre diferentes módulos do sistema.
- Teste de Sistema: Validar o sistema como um todo.
- Teste de Aceitação: Confirmar que o sistema atende aos requisitos definidos pelo cliente.

2.3 Triagem de Erros

- Definir a resolução para cada erro.
- Priorizar os erros e definir cronogramas para correção.

2.4 Critérios de Suspensão e Requisitos de Retomada

- Testes serão suspensos se erros críticos impedirem a continuidade dos testes.
- Testes serão retomados após a correção dos erros críticos.

2.5 Completude do Teste

Os testes serão considerados completos quando:

- 100% dos casos de teste forem executados.
- Todos os erros críticos forem corrigidos.
- A cobertura de código atingir um nível satisfatório.
- O score de mutação igualar, ou superar, 80%.
- O teste estrutural igualar ou superar 80% de cobertura no critério todas-arestas das classes não CRUD da primeira entrega.

3. ENTREGÁVEIS DE TESTE

- Plano de Teste
- Casos de Teste
- Relatórios de Erros
- Métricas de Teste
- Documentação de Estratégia de Teste

4. NECESSIDADES DE RECURSOS E AMBIENTE

4.1 Ferramentas de Teste

- Repositório de código: GitHub, onde fizemos um fork do projeto original e commitamos as alterações.

- Comunicação e colaboração: Discord, WhatsApp e reuniões presenciais.
- Ferramentas de automação de teste:
- PHPUnit para testes de unidade do backend em PHP
- Laravel Dusk para testes de browser end-to-end
- Mockery para mocks em testes de unidade
- Infestation para testes de mutação
- Testlink para gerenciamento de casos de teste
- Ferramentas de integração contínua: Não foi utilizado.

4.2 Ambiente de Teste

- Backend: Desenvolvido em PHP utilizando o framework Laravel.
- Frontend: Interface de usuário desenvolvida com Vue.js.
- Banco de Dados: MariaDB.
- Servidores: A aplicação é executada em contêineres Docker, com cada serviço isolado em contêineres distintos.
    - Aplicativo Crater: Rodando em um contêiner próprio.
    - Banco de Dados MariaDB: Executado em um contêiner separado, facilitando a gestão e backup dos dados.
    - Servidor Nginx: Utilizado para servir a aplicação e lidar com requisições HTTP.
- Requisitos mínimos de hardware: Computadores com processadores i3 10gen ou superior, 8GB de RAM.

5. TERMOS / ACRÔNIMOS

- API: Application Program Interface
- MVP: Minimum Viable Product
- Ágil: Metodologia de desenvolvimento ágil
- PHPUnit: Framework para criação de testes unitários em PHP
- Laravel Dusk: Ferramenta para testes de browser end-to-end em Laravel
- Mockery: Biblioteca para criação de mocks em testes de unidade
- Infestation: Ferramenta para testes de mutação
- Testlink: Ferramenta para gerenciamento de casos de teste