UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS DE QUIXADÁ

**CURSO DE ENGENHARIA DE SOFTWARE**

# PLANO DE MEDIÇÃO - PIES


**Equipe:**

João Vitor Soares - 495328

Cleiton dos Santos Queiros - 477852

**Professor:**

Camilo Almendra

**SUMÁRIO**

**GLOSSÁRIO**

| Siglas | Definição |
|---|---|
| QA | Quality Assurance |
| | |
| | |
| | |
| | |

**HISTÓRICO DE REVISÕES**

| Data | Versão | Descrição | Responsável |
|---|---|---|---|
| 01/10/2022 | 1.0 | Criação do documento. | João Vitor Soares Furtado e Cleiton dos Santos Queiroz |
| 06/10/2021 | 1.1 | Preenchimento de toda a seção 1 (Introdução), seção 2 (Método) e seção 3 (Referências). | João Vitor Soares Furtado e Cleiton dos Santos Queiroz |
| 19/10/2022 | 1.2 | Revisão do documento | João Vitor Soares Furtado e Cleiton dos Santos Queiroz |

## 1. INTRODUÇÃO

O presente sistema na qual será abordado neste documento é uma aplicação desenvolvida a fim de digitalizar e desburocratizar o processo de adoção de animais em situação de rua ou situação de risco. O sistema se chama Petland, na sua versão mobile, está sendo desenvolvida em Kotlin, utilizando o Postgresql como solução de banco de dados e Nodejs como Back-end.

### 1.1. Descrição do produtos a ser avaliado

Este documento visa fornecer o plano de medição do aplicativo *"Petland".* O mesmo se trata de um sistema de adoção de pets. O intuito do sistema, é facilitar o sistema de adoção de pets por meio do aplicativo de maneira simples e menos burocrática. Por conta da forma como a arquitetura do sistema descrito foi construída, podemos escolher qual ambiente será analisado. Assim como foi descrito antes, iremos analisar a aplicação, que consiste na parte do back-end e front-end do mesmo.

Graças ao sistema de criação de tags do Github, podemos avaliar o presente sistema através de versões. As tags criadas são referências às versões na qual o sistema se encontra. A partir dessas versões, podemos analisar a qualidade do código através de métodos de avaliação que serão descritos mais à frente. Nosso objetivo é ver a evolução do sistema, podendo ver se o sistema melhorou ou não durante o período de desenvolvimento analisado.

### 1.2. Objetivos da avaliação

Temos como objetivo, avaliar métricas como Manutenibilidade, Eficiência e Usabilidade do sistema Petland.

| | |
|---|---|
| Analisar | Usabilidade |
| Para o propósito de | Analisar o sistema a fim de verificar se está funcionando dentro da conformidade. |
| Com respeito a | Operabilidade |
| Do ponto de vista | Dos usuários da aplicação |
| No contexto de | Da aplicação |

| | |
|---|---|
| Analisar | Eficiência |
| Para o propósito de | Analisar a eficiência da aplicação no quesito do tempo de resposta das ações dos usuários durante o uso da |

| | |
|---|---|
| | aplicação. Analisando o tempo na qual a aplicação leva para responder comandos executados pelo usuário. |
| Com respeito a | Tempo de Resposta |
| Do ponto de vista | Usuário |
| No contexto de | Da aplicação utilizando o driver Appium |

| | |
|---|---|
| Analisar | Manutenibilidade |
| Para o propósito de | Analisar a facilidade do software de ser modificado a fim de corrigir defeitos e realizar novas implementações e identificar quando ocorrem falhas e se elas são críticas para o sistema quando ocorrem. |
| Com respeito a | Testabilidade, Modularidade |
| Do ponto de vista | Dos desenvolvedores e QAs |
| No contexto de | Understand e Embold |

## 2. MÉTODO

Nos tópicos abaixo, abordamos quais as tarefas que devem ser realizadas pelos participantes, qual ambiente será testado e as ferramentas utilizadas. Além disso, foi realizada uma descrição de todo esse procedimento.

### 2.1. Ambiente de avaliação

- As medidas a serem coletadas foram :
  - quantidade de linhas
  - métodos testados, além de taxas de funções, e arquivos acima do permitido
  - tempo para conclusão de tarefa específica
  - taxa de consistência operacional em uso
- Teste de usabilidade será feito no aplicativo, instalado em um celular android.
- As ferramentas Embold e Understand para coletar métricas direto do código, Appium para coletar os resultados nos dispositivos.

### 2.2. Procedimentos da Avaliação

Os participantes foram informados que a manutenibilidade (testabilidade e modularidade), usabilidade (operabilidade) e eficiência (tempo de resposta) do aplicativo serão analisadas apenas para fins de coleta de dados para que possamos analisar a facilidade do software de ser modificado a fim de corrigir bugs e realizar novas implementações, entender a facilidade de uso do sistema e, finalmente, testar seu desempenho em uso. Dessa forma, os desenvolvedores e testadores são informados de que

este não é um teste especificamente para avaliar seu desempenho individual, mas busca entender melhor o código e fazer melhorias para otimizar o desenvolvimento posterior.

O avaliador então explica aos desenvolvedores e ao controle de qualidade como será a avaliação, mostrando o que deve ser coletado e as ferramentas que tornam o processo possível. Além disso, os participantes foram entrevistados para ver se tinham experiência com as ferramentas que usariam.

## 2.3. Medidas de Software

Abaixo foram descritas as medidas de software que serão coletadas, sendo elas Operabilidade, Tempo de resposta, Testabilidade e Modularidade.

### 2.3.1 Testabilidade

A Testabilidade examina as diferentes probabilidades e características comportamentais que levam o código a falhar se alguma coisa estiver incorreta. Ou seja, a capacidade de testar um sistema modificado, tanto quanto as novas funcionalidades quanto as não afetadas diretamente pela modificação.

| **Nome** | **Descrição** | **Função de Medição** | **Método** |
|---|---|---|---|
| Taxa de **linhas** de código testadas. | Qual a **taxa de linhas** de código que foram chamadas pelo menos uma vez durante os testes unitários ? | X = ( LT/T ) * 100<br>LT = Número de linhas testadas<br>T = Número total de linhas existentes na classe | Análise de código usando a ferramenta Embold. |
| Taxa de **métodos** testados. | Qual a **taxa de métodos** que foram chamadas pelo menos uma vez em uma classe durante os testes unitários ? | X = ( MT/T ) * 100<br>MT = Número de métodos testados<br>T = Número total de métodos existentes na classe | Análise de código usando a ferramenta Embold. |
| Taxa de **declarações** testadas. | Qual a **taxa de declarações** “If” que foram cumpridas pelo menos uma vez em uma classe durante os testes unitários ? | X = ( DT/T ) * 100<br>DT = Número de declarações testadas<br>T = Número total de declarações existentes na classe | Análise de código usando a ferramenta Embold. |

### 2.3.2 Modularidade

As medidas de modularidade abaixo, foram utilizadas para avaliar o grau em que as mudanças em um componente podem impactar em outros componentes, buscando garantir o menor impacto possível.

Para realizar a análise dessa subcaracterística, levamos em consideração os argumentos apresentados por Martin Fowler sobre características que levam a um código de qualidade. Dentre suas análises, destacamos seu comentário sobre a quantidade de linhas em funções e o impacto na complexidade do projeto. A partir desse estudo, desenvolvemos métricas visando a qualidade de código do projeto.

| **Nome** | **Descrição** | **Função de Medição** | **Método** |
|---|---|---|---|
| Taxa de **funções** com tamanho acima do permitido. | Quantas **funções** passam do **limite** de **linhas** estipulado pela equipe? | X = ( NF / NT ) * 100<br>NF = Número de funções que passam do limite de linhas por função.<br>NT = Número total de funções. | Análise de código usando a ferramenta Understand. |
| Taxa de **classes** com tamanho acima do permitido. | Quantas **classes** passam do **limite** de **linhas** estipulado pela equipe? | X = ( NC / NT ) * 100<br>NC = Número de classes que passam do limite de linhas por classes.<br>NT = Número total de classes. | Análise de código usando a ferramenta Understand. |

### 2.2.3 Operabilidade

A operabilidade vai medir o grau de facilidade com a qual aplicativo é operado e/ou controlado pelo usuário. Aqui ela será utilizada para analisar se o app está de acordo com o propósito pelo qual ele foi desenvolvido através de uma análise da consistência dos componentes de interface do usuário.

| | | | |
|---|---|---|---|
| Taxa de consistência operacional em uso | Quão consistentes são os componentes da interface do usuário? | Número de operações que o usuário considerou inaceitavelmente inconsistentes com a expectativa do usuário | Análise de resultados com teste de usabilidade com o usuário |

### 2.2.4 Tempo de resposta

É a métrica que mede o tempo em que uma requisição leva para responder. Essa medição se inicia no momento que o usuário faz a solicitação até o momento

em que recebe uma resposta do servidor. Seus resultados podem ser variados e questões de rede, infraestrutura, banco de dados e tantas outras influenciam diretamente em seus valores.

| Nome | Descrição | Função de Medição | Método |
|---|---|---|---|
| Tempo para conclusão de uma tarefa específica | Qual é o tempo necessário para concluir uma tarefa específica? | T = Tf - Ti<br>Tf = Tempo para obter o resultado<br>Ti = Tempo da entrada do comando finalizada | Análise do desempenho utilizando o Appium para coletar o resultado nos dispositivos |

### 2.4. Procedimentos de Interpretação

Através das métricas coletadas das ferramentas citadas neste documento, podemos concluir qual o nível de maturidade que o nosso sistema se encontra, quais partes devem ser refatoradas ou não. Temos, como objetivo, usar essas métricas para melhorar o sistema nos aspectos de Manutenibilidade, Eficiência e Usabilidade do sistema Petland.

- Com base na decisão da equipe, julgamos que no máximo 50 linhas por função é o ideal, pelo fato de ser uma quantidade de linhas que geralmente preenche a tela de um desenvolvedor. Para tamanho de classes, julgamos 150 linhas o número ideal de linhas em uma única classe.
- Como representação gráfica, serão utilizados gráficos de barras da ferramenta de compreensão e gráficos e relatórios do Embold.
- Para operacionalidade, definimos consistentemente o nível como menos de 02 operações inconsistentes para bom, entre 03 e 6 operações inconsistentes para regular e mais de 7 operações inconsistentes para ruim.
- Quanto ao tempo de resposta, a equipe julgou ser o menor tempo de resposta possível para as operações realizadas. No melhor caso até 2 segundos, regular entre 3 a 8 segundos, ruim de 8 a 15 segundos e péssimo acima de 15 segundos.

## 3. REFERÊNCIAS

ISO/IEC 25000. Software Engineering - Software Product Quality Requirements and Evaluation (SQuaRE) – Guide to SQuaRE. v. 2005, 2005.

ISO/IEC 9126. Software Engineering – Product Quality – Part 1. 2001.

ISO/IEC 9126-2. Software Engineering – Product Quality – Part 2: External Metrics. 2001.