**Entrega 2**

**Introdução**

Neste documento, detalhamos o design de software, diagramas UML, padrões de projeto e arquiteturas aplicadas no desenvolvimento do sistema de análise de dados de interrupções e reclamações de energia elétrica. O objetivo é fornecer uma solução tecnológica robusta e escalável que permita gerenciar, analisar e visualizar dados de maneira eficiente, facilitando a tomada de decisões estratégicas e melhorando o entendimento das causas e impactos no setor elétrico.

**Design de Software**

O design de software do projeto foi desenvolvido com base em princípios de modularidade, separação de preocupações e boas práticas de engenharia de software. A arquitetura escolhida segue o padrão **MVC (Model-View-Controller)**, garantindo que o sistema seja organizado e eficiente.

**Arquitetura Geral**

A arquitetura é dividida em três camadas principais:

1. **Model (Modelo)**: Responsável por gerenciar a lógica de negócios e a interação com o banco de dados SQLite. Inclui as entidades Interrupcao e Reclamacao.
2. **View (Visão)**: Implementada por meio de dashboards interativos que apresentam visualizações gráficas e relatórios.
3. **Controller (Controlador)**: A API desenvolvida em Python que serve de intermediária entre a camada de modelo e as visualizações, processando as requisições e fornecendo os dados necessários.

**Camadas do Sistema**

1. **Backend**:
   * Implementação da API RESTful utilizando Flask ou FastAPI.
   * Camada de lógica de negócios e gerenciador de banco de dados para a persistência e consulta de dados.
2. **Frontend**:
   * Dashboards interativos que mostram insights e estatísticas, utilizando bibliotecas como Plotly/Dash ou Seaborn.
3. **Data Processing**:
   * Manipulação de dados com Pandas e análise preditiva usando Scikit-Learn, para enriquecer as visualizações e fornecer insights.

**Diagramas UML**

**1. Diagrama de Casos de Uso**

**Descrição**: Representa as principais funcionalidades do sistema, destacando as interações do usuário (administrador ou analista de dados) com a aplicação.

* **Ator Principal**: Administrador ou Analista de Dados.
* **Casos de Uso**:
  + Consultar dados de interrupções.
  + Consultar dados de reclamações.
  + Gerar gráficos de análise.
  + Exportar relatórios.
  + Atualizar banco de dados.

**2. Diagrama de Classes**

**Descrição**: Define as principais classes e suas relações, incluindo atributos e métodos.

* **Classes**:
  + Interrupcao: Representa dados de interrupções, como data, duração e causa.
  + Reclamacao: Contém informações sobre reclamações por município.
  + Dashboard: Classe para gerar gráficos interativos.
  + APIController: Gerencia requisições e respostas da API.
  + DatabaseManager: Classe singleton que controla a conexão com o banco de dados SQLite.
* **Relações**:
  + Dashboard interage com APIController para buscar dados.
  + APIController se comunica com DatabaseManager para operações de CRUD (Create, Read, Update, Delete).

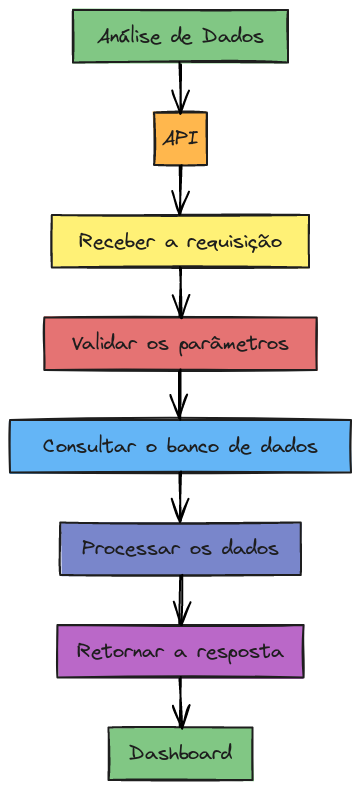
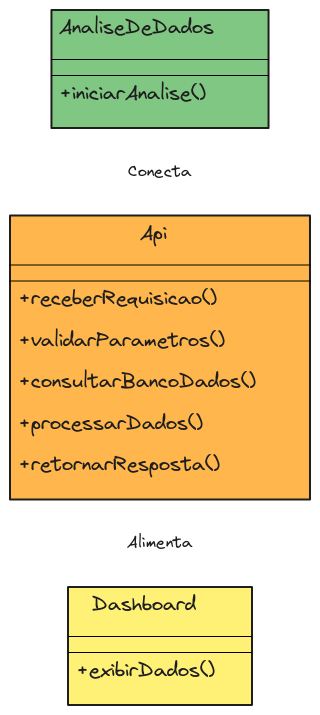
**3. Diagrama de Sequência**

**Descrição**: Mostra o fluxo de interação para a consulta de dados de interrupções e a atualização do dashboard.

1. O usuário solicita uma consulta de dados.
2. APIController processa a solicitação e interage com DatabaseManager.
3. DatabaseManager retorna os dados solicitados.
4. APIController envia os dados ao Dashboard, que renderiza a visualização.

**4. Diagrama de Atividade**

**Descrição**: Representa o fluxo de atividades para processar uma requisição de consulta.

**Padrões de Projeto**

1. **Singleton**: Implementado em DatabaseManager para garantir uma única instância da conexão com o banco de dados.
2. **Factory Method**: Aplicado na criação de objetos complexos, como gráficos de visualização.
3. **Observer Pattern**: Utilizado para atualizar a interface do usuário quando há alterações significativas nos dados.

**Arquitetura de Sistema**

* **RESTful API**: A API segue os princípios REST, fornecendo endpoints para consultas, filtragens e atualizações dos dados de interrupções e reclamações.

**Tecnologias Utilizadas**

* **Linguagem de Programação**: Python.
* **Frameworks**: Flask para a API, Pandas para análise de dados, Plotly/Dash para visualizações.
* **Banco de Dados**: SQLite.
* **Bibliotecas de Machine Learning**: Scikit-Learn para análise preditiva.
* **Ferramentas de Visualização**: Seaborn e Plotly.

**Conclusão**

O design proposto garante que o sistema seja eficiente, escalável e fácil de manter. A aplicação das técnicas de engenharia de software, como o uso de padrões de projeto e a separação de preocupações, oferece uma base sólida para futuras expansões. Além disso, os diagramas UML fornecem uma representação visual clara das interações e fluxos do sistema, facilitando a compreensão por todos os stakeholders.

A solução proposta ajuda a abordar a dor do mercado relacionada à necessidade de análise detalhada e visualizações de interrupções e reclamações. O sistema oferece aos gestores de concessionárias uma maneira de entender melhor os problemas, tomar decisões informadas e comunicar os dados de forma eficaz.