Plataforma de Análise de Dados financeiros do IPEA

Documento de arquitetura

Grupo 9 - 2025.1

Versão 1.0

Histórico de revisões:

Data	Versão	Descrição	Autores
23/04/2025	1.0	Considerações Iniciais	Pedro Rocha Ferreira Lima
08/05/2025	2.0	Adição de tópicos, diagramas e tecnologias novas	Pedro Rocha Ferreira Lima

Autores:

Matrícula	Nome	Função
222034270	Pedro Rocha Ferreira Lima	Arquiteto de Software
200014226	Ana Luiza Borba de Abrantes	Project Owner
231034064	Arthur Henrique Vieira	Dev
222006857	João Vitor Sales Ibiapina	Dev
232014057	Kauã Vale Leão	DevOps
232014567	Saied Muhamad Yacoub Falaneh	Scrum Master

Sumário

1.0 INTRODUÇÃO

1.1 Propósito

Este documento tem como objetivo descrever a arquitetura da Plataforma de Análise de Dados Financeiros do IPEA, desenvolvida no contexto da disciplina de Métodos de Desenvolvimento de Software (2025.1). Ele visa garantir a clareza na construção, manutenção e evolução do sistema, orientando as decisões técnicas e organizacionais da equipe.

1.2 Escopo

O sistema permitirá a coleta, análise e visualização de dados financeiros públicos, auxiliando pesquisadores e gestores do IPEA na formulação de políticas públicas baseadas em dados. Os usuários poderão construir painéis interativos para visualizar os dados financeiros em tempo real, e contar com a geração automática de textos com base na análise de dados financeiros e resumos automatizados de tendências financeiras e alertas para gestores públicos.

2.0 REPRESENTAÇÃO ARQUITETURAL

2.1 Definições

Adotaremos a arquitetura baseada no padrão MVC (Model-View-Controller), adaptado ao padrão MVT (Model-View-Template) do Django, onde o sistema apresenta três componentes e cada camada é responsável por uma função específica no sistema.

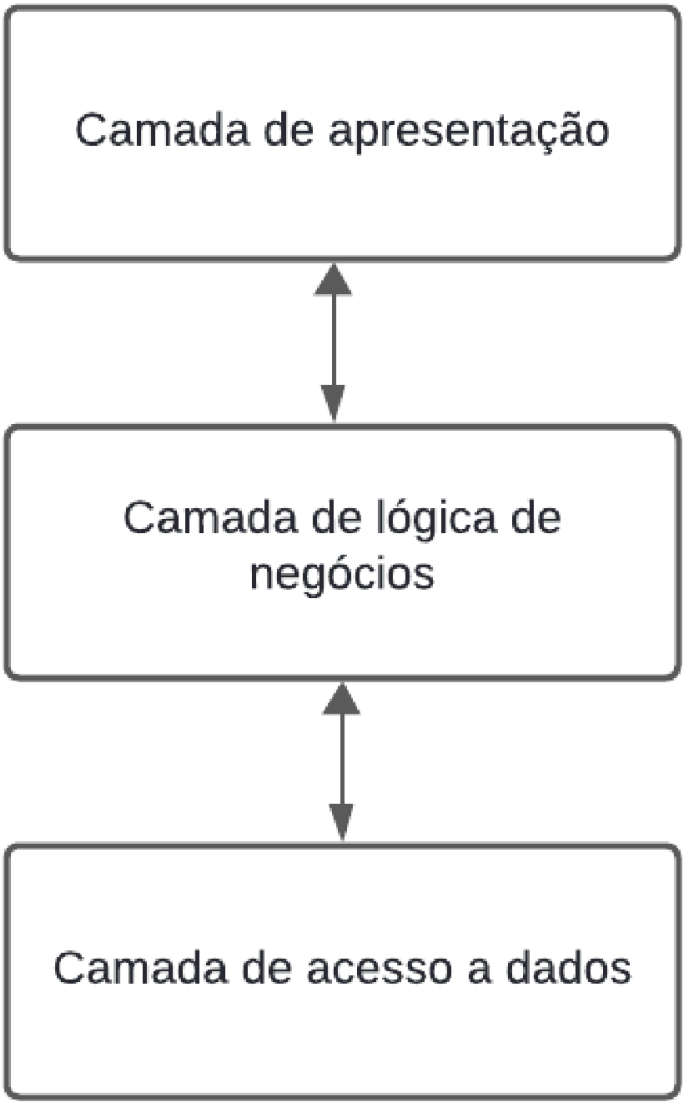

Figura 1.

2.2 Justificativa

2.2.1 Separação de responsabilidade

A arquitetura em camadas garante que cada componente tenha responsabilidades bem definidas, facilitando a manutenção e testes.

2.2.2 Reutilização de Componentes

O uso de Django e Pandas permite criar componentes reutilizáveis, como filtros e gráficos interativos.

2.2.3 Facilidade de Teste

A estrutura modular favorece a criação de testes unitários e de integração por camada.

2.2.4 Desenvolvimento paralelo

Com as camadas desacopladas, diferentes membros da equipe podem trabalhar simultaneamente no frontend (Streamlit) e backend (Django).

2.2.5 Escalabilidade e Adaptação

A integração com Docker e AWS permite escalar os serviços conforme a necessidade.

2.2.6 Alinhamento às Necessidades

A arquitetura foi escolhida considerando os requisitos funcionais do IPEA, a facilidade de uso para analistas e o tempo disponível para entrega.

2.3 Detalhamento

1. Model: É a parte que gerencia os dados e a lógica de negócios da aplicação. Em nossa aplicação será usado no sistema da Plataforma de Análise de Dados Financeiros do IPEA, o Model será responsável por representar a estrutura dos dados e interagir com o banco de dados (SQLite) por meio do ORM do Django.

Ele definirá entidades como:

- Conjunto de Dados: arquivos CSV carregados pelo usuário;
- Análises: parâmetros e resultados de análises realizadas com Pandas;
- Usuários: perfis de acesso e preferências de visualização;
- **Logs de Acesso**: registros de uso da plataforma.

Essas classes serão fundamentais para persistência, integridade e validação dos dados.

2. Template (View): O template é a parte visual do código, responsável pela renderização e o conteúdo dinâmico para o usuário. Neste sistema, os templates serão utilizados principalmente para:

- Exibir visualizações interativas com Streamlit (ou HTML/CSS, se necessário);

- Apresentar dashboards, gráficos e tabelas resultantes da análise de dados;
- Exibir mensagens de erro, loading, e feedback de ações do usuário.

O Figma será usado para planejar esses templates, e a renderização será feita com Streamlit + HTML estilizado.

3. View (Controller): Será o sistema que gerencia as requisições, lógica de negócio e interações entre as camadas, utilizando Django. As Views em Django:

- Recebem as requisições do usuário;
- Chamam funções de análise (com Pandas);
- Validam os dados de entrada;
- Recuperam dados do banco de dados;
- Retornam os resultados renderizados nos templates (Streamlit ou HTML).

Exemplo: ao carregar um CSV, a View processa o arquivo com Pandas, salva as informações no banco (Model), e envia os dados tratados para o Template.

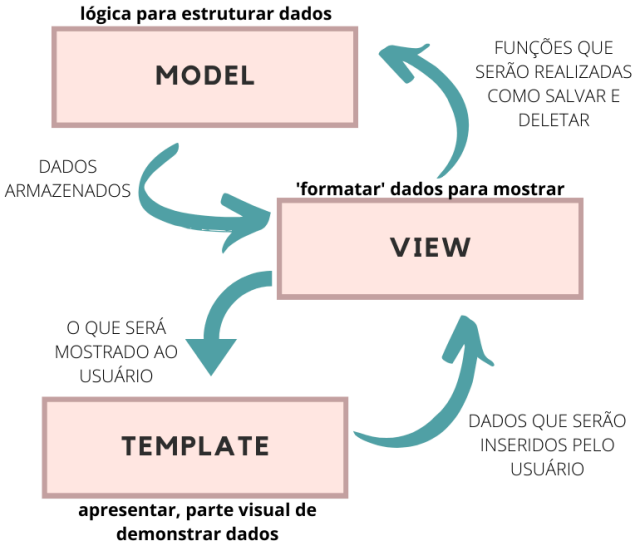

Figura 2.

2.4 Metas e Restrições Arquiteturais

Para definir as metas de arquitetura do software, são consideráveis os seguintes aspectos:

- **Escalabilidade**: O sistema precisa ser capaz de crescer para suportar um número cada vez maior de usuários e dados ao longo do tempo. É essencial que a infraestrutura consiga lidar com essa expansão sem comprometer o desempenho.
- **Desempenho**: O software deve oferecer respostas rápidas e manter um desempenho eficiente, garantindo uma experiência satisfatória para os usuários.
- **Manutenibilidade:** A estrutura do software deve ser projetada para facilitar sua manutenção e evolução ao longo do tempo, possibilitando a adição de novas funcionalidades, correções de erros e melhorias contínuas de maneira ágil e eficiente.
- **Segurança:** É fundamental proteger o sistema contra acessos não autorizados, assegurando a integridade e a confidencialidade das informações.

Ao definir as restrições arquiteturais para o software de divulgação e gerenciamento, é importante considerar os seguintes aspectos:

- **Compatibilidade:** O sistema precisa ser funcional em todas as máquinas.

Esses objetivos e restrições arquiteturais devem ser avaliados com atenção e incorporados ao desenvolvimento do sistema, de forma a atender plenamente às necessidades da equipe de competição e garantir um funcionamento seguro e eficiente.

2.5 Visão de Casos de Uso (escopo do produto)

A arquitetura escolhida para o sistema segue o padrão Model-View-Template (MVT), utilizando os framework Django para o backend, que foi escolhido pela facilidade que se comunica com o frontend em CSS e HTML e pela forma que consegue atender com perfeição as necessidades do projeto. Usaremos também a biblioteca do Python Pandas que irá auxiliar na criação de gráficos e a biblioteca Scikit-learn para o sistema de aprendizado de máquina.

As linguagens escolhidas para o frontend foram HTML e CSS, que se integram e se comunicam com facilidade ao framework Django, e pela capacidade de criar o escopo do site de acordo com as necessidades do projeto. O banco de dados adotado foi o SQLite devido a uma maior afinidade dos membros do grupo.

No que se diz a respeito do escopo geral do sistema, ele pode ser dividido em partes que têm funcionalidades como:

- Tela inicial com menu de navegação.

- Construção de painéis interativos para visualizar os dados financeiros em tempo real.
- Geração automática de textos e relatórios com base na análise de dados financeiros.
- Previsões utilizando Machine Learning
- Aplicar filtros e critérios de análise.
- Carregar datasets financeiros.

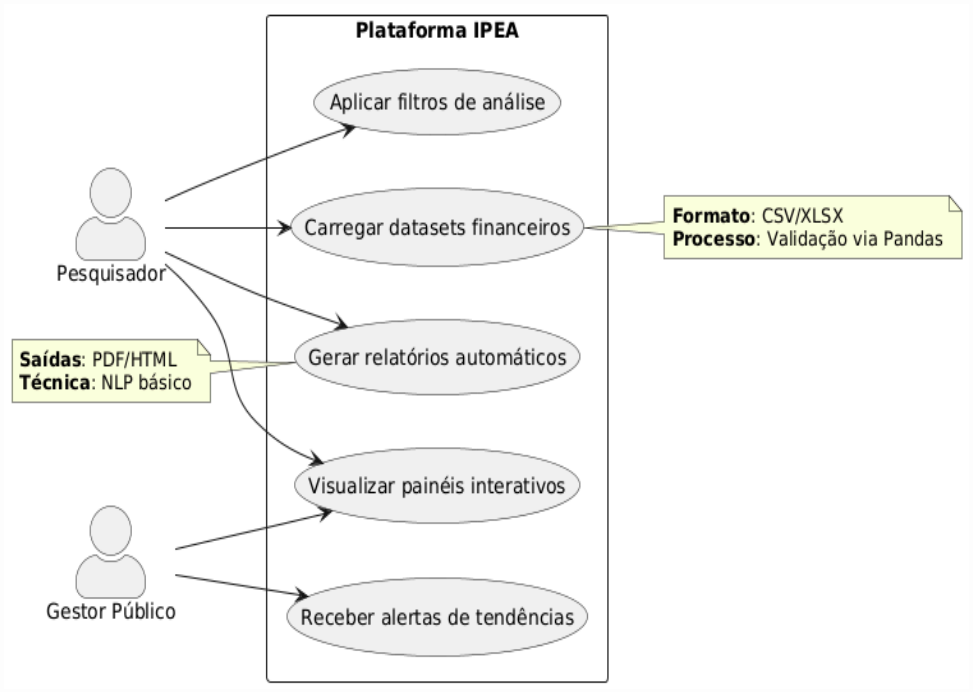

Figura 3. Diagrama de casos de uso.

2.6 Visão Lógica

A visão lógica descreve os principais componentes do sistema, suas responsabilidades e suas interações, com base no padrão **Model-View-Template (MVT)** do Django.

Componentes principais:

- **Model (Modelo):** Representa os dados e regras de negócio da aplicação. Inclui classes como:
    - **ConjuntoDeDados**: representa arquivos CSV carregados pelos usuários.

    - **AnaliseFinanceira:** armazena os parâmetros de análise e os resultados processados com Pandas.
    - **Usuario**: controla autenticação, permissões e preferências de visualização.
    - **LogAcesso**: registra as ações executadas pelos usuários.

- **View (Controlador - lógica de controle):** Responsável por processar as requisições, executar a lógica de negócio e intermediar a comunicação entre o modelo e a camada de apresentação.
    - Exemplo: Ao carregar um dataset, a View valida e processa os dados com Pandas, salva no modelo e envia para renderização.
- **Template (Apresentação):** Interface com o usuário. Será construído com **Streamlit**, HTML e CSS para exibir dashboards, gráficos interativos, tabelas e mensagens ao usuário.

Fluxo Lógico Geral:

1. O usuário interage com a interface (template).
2. A requisição é processada pelas Views no Django.
3. As Views acessam ou modificam os dados no Model.
4. Os dados tratados são enviados de volta ao template para exibição.

Figura 4. Diagrama de Estados da Aplicação.

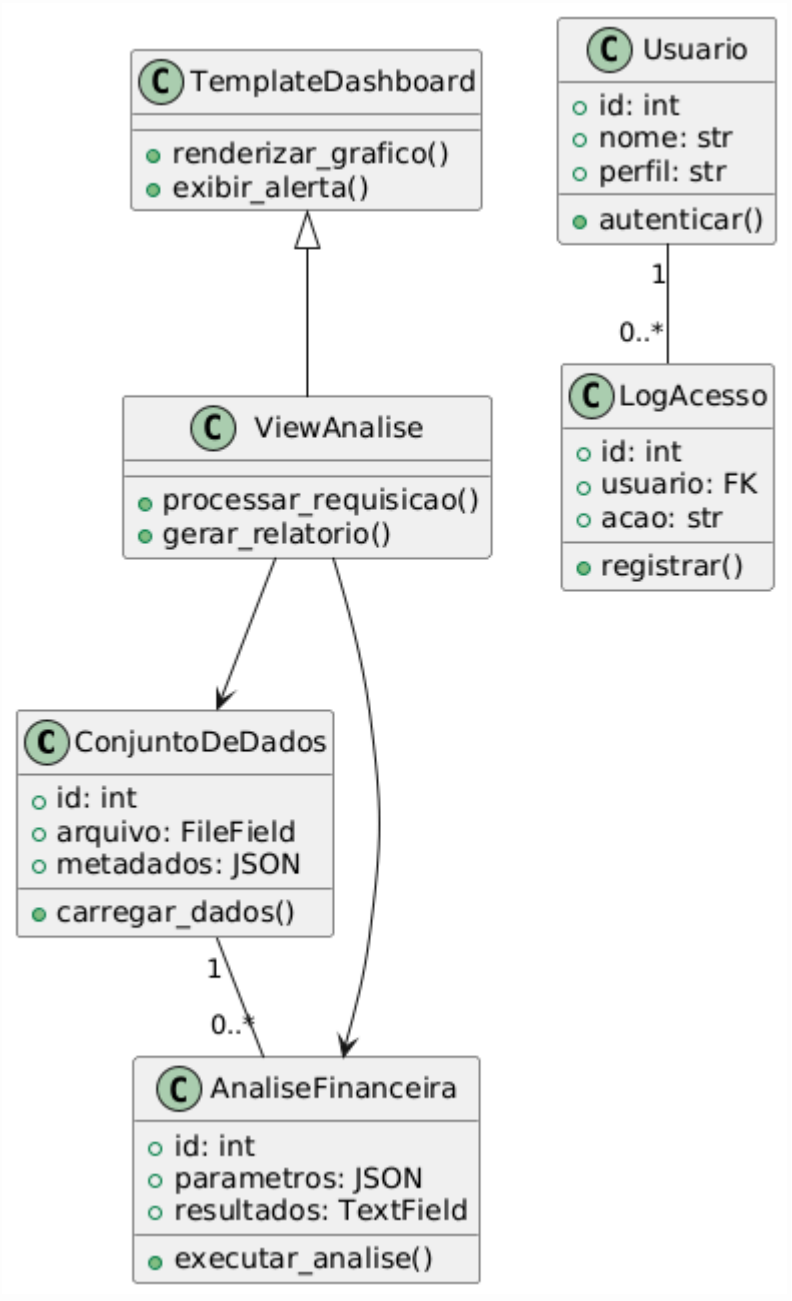

Figura 5. Diagrama de Classes

2.7 Visão de Implementação

A visão de implementação detalha a organização do sistema no nível de código e estrutura dos arquivos. A aplicação é organizada em camadas e pacotes modulares para facilitar o desenvolvimento, testes e manutenção.

Bibliotecas e frameworks:

- Django: estrutura principal da aplicação (backend).
- Scikit-Learn: aprendizado de máquina.
- Pandas: análise e manipulação de dados.
- Streamlit: renderização de dashboards interativos.
- HTML/CSS: estilização e estrutura visual adicional.

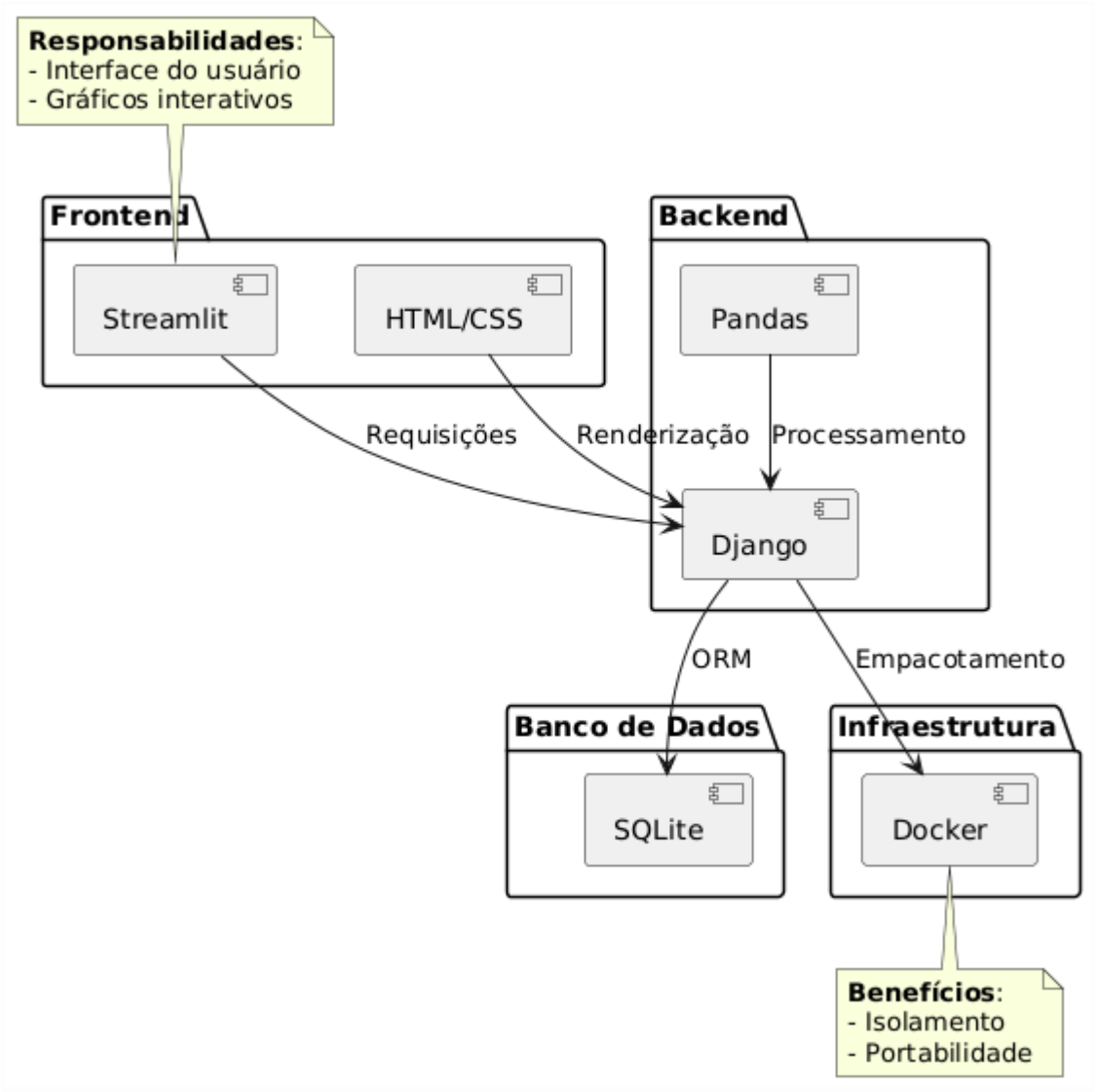

Figura 6. Diagrama de visão de Implementação.

2.8 Visão de Implantação

O software será implantado em ambiente desktop com suporte a execução via navegador, adotando uma abordagem moderna baseada em servidor web local. Para o front-end, será utilizada a biblioteca Streamlit, integrada com HTML e CSS para estruturar e estilizar as páginas, permitindo a criação de interfaces interativas e intuitivas.

No back-end, será utilizado o framework Django, que adota a arquitetura MVT (Model-View-Template), facilitando a separação das responsabilidades entre as camadas da aplicação. O banco de dados utilizado será o SQLite, devido à sua leveza, fácil configuração e integração nativa com o Django.

A implantação será feita utilizando contêineres Docker, o que permitirá o empacotamento da aplicação e suas dependências de forma padronizada, promovendo portabilidade e facilitando a replicação do ambiente em diferentes máquinas. Além disso, a aplicação será preparada para futura hospedagem em serviços de nuvem, como a AWS, garantindo escalabilidade e facilidade de acesso remoto, conforme a necessidade de crescimento da plataforma.

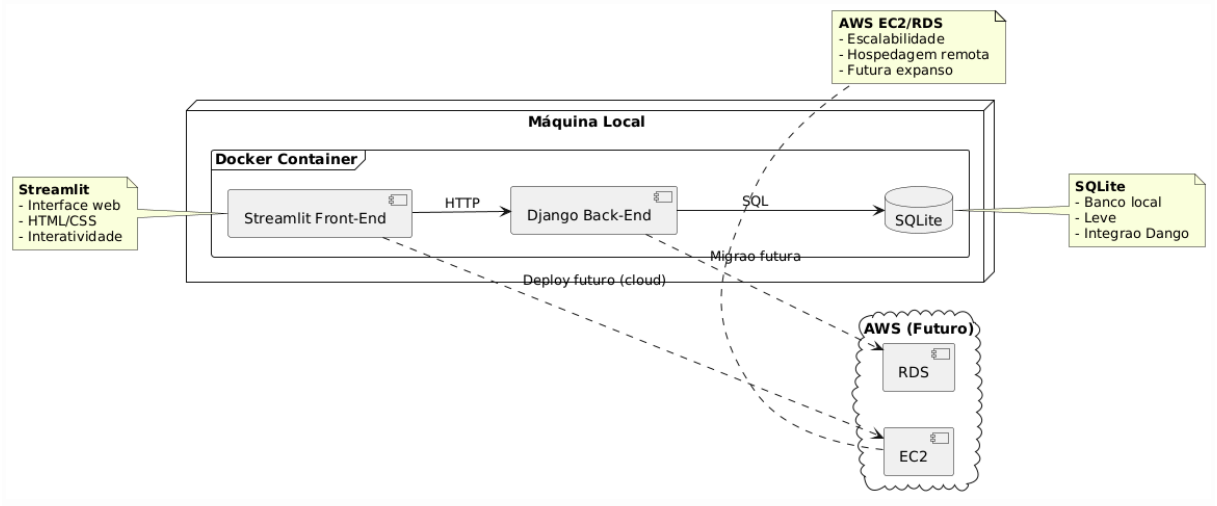

Figura 7. Diagrama de visão de Implantação

2.9 Restrições adicionais

O software possui restrições definidas para garantir que atenda os requisitos negociais e de qualidade, que serão levados extremamente a sério pela equipe de desenvolvimento para garantir seu funcionamento e proporcionar a experiência exigida pelo cliente.

- O site deverá ser acessível para todos os públicos de forma que consigam ter uma visão inicial sobre o produto antes de qualquer cadastro.
- Algumas funções serão restritas apenas a membros cadastrados e logados, garantindo que somente usuários possam acessar as funcionalidades originais do site.

Alguns requisitos de qualidade devem ser essenciais para alcançar o funcionamento desejado do sistema.

- **Usabilidade:** A interface do sistema deve ser intuitiva e de fácil navegação para os usuários não terem dificuldade no uso, trazendo assim uma experiência agradável permitindo que com facilidade utilizem de todas as funções ofertadas.
- **Portabilidade:** O site deve poder ser acessado por todos sistemas operacionais, navegadores e dispositivos, de forma que não haja mudança em nenhuma das opções escolhidas, assim democratizando o acesso para o público geral.
- **Segurança:** Deve haver proteção contra acessos não autorizados, assegurando a integridade dos dados e a privacidade das informações de usuário. Medidas de seguranças são essenciais para evitar ameaças e ataques.
- **Manutenibilidade:** O código deverá ser bem documentado seguindo os padrões de desenvolvimento, para facilitar correções de erros e adições futuras ao sistema.
- **Escalabilidade:** O sistema deve ser desenvolvido pensando em um futuro crescimento e aumento de fluxo de dados, permitindo a expansão conforme a necessidade e evitando assim problemas futuros em seu desempenho.

3. BIBLIOGRAFIA

Arquitetura MVC: entendendo o modelo-visão-controlador. DIO.me, 2024. Disponível em: https://www.dio.me/articles/arquitetura-mvc-entendendo-o-modelo-visao-controlador. Acesso em: 23 abril de 2025.