Descrição da Arquitetura do Software (Architecture Notebook)

Data: 24 de Junho de 2025

Versão: 1.1

Autor: Henrique Valente Lima (211055380)

1. Introdução

Este documento detalha a arquitetura do Sistema de Gestão de Feiras, descrevendo seus componentes essenciais, os relacionamentos entre eles e as tecnologias que fundamentam sua estrutura. A arquitetura proposta foi concebida para assegurar modularidade, manutenção, segurança e escalabilidade, alinhando-se com os requisitos não funcionais do sistema.

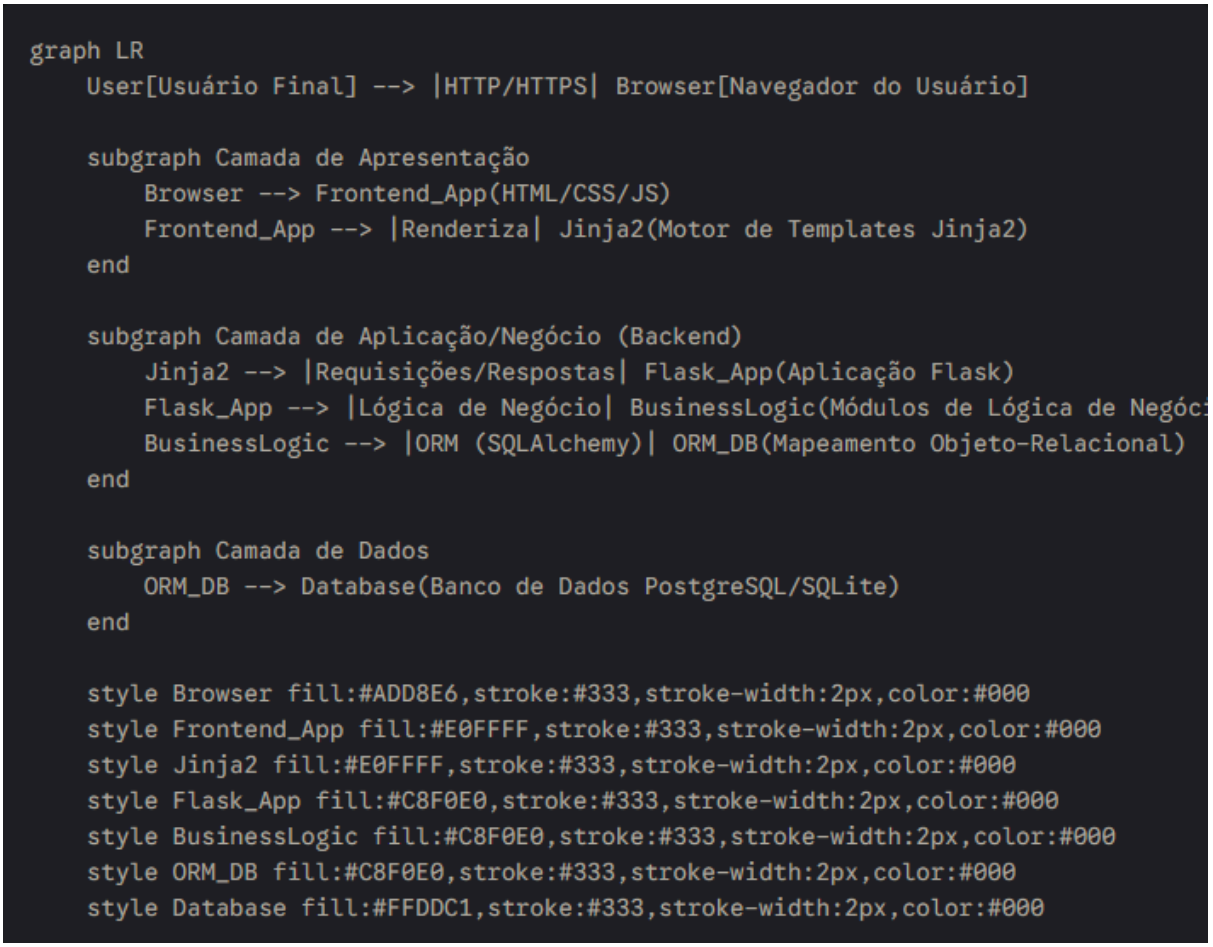
2. Visão Geral da Arquitetura

O Sistema de Gestão de Feiras será desenvolvido com base em uma **arquitetura em camadas (N-tier)**, um padrão comprovado e robusto para aplicações web. Esta abordagem é crucial para promover a clara separação de responsabilidades, o que facilita significativamente o desenvolvimento, a manutenção e a evolução contínua do sistema. A arquitetura será estruturada em três camadas principais:

1. **Camada de Apresentação (Frontend):** Responsável pela interface do usuário e pela interação direta no navegador do cliente.
2. **Camada de Aplicação/Negócio (Backend):** Engloba a lógica de negócios, gerencia o fluxo de dados e processa as requisições provenientes do frontend.
3. **Camada de Dados:** Encarregada do armazenamento e da gestão persistente de todos os dados do sistema.

3. Diagrama da Arquitetura (Conceitual Detalhado)

Para uma melhor compreensão visual, o diagrama abaixo ilustra a interação entre as camadas e os principais componentes tecnológicos envolvidos:



4. Descrição das Camadas e Elementos

4.1. Camada de Apresentação (Frontend)

Esta camada constitui a interface direta com a qual o usuário final interage.

- **Tecnologias Principais:**
    - **HTML5:** Utilizado para estruturar as páginas web.
    - **CSS3:** Empregado para estilização e layout da interface, buscando uma aparência limpa e responsiva.
    - **JavaScript:** Responsável pela interatividade no lado do cliente, como validações de formulário e, se necessário para uma experiência mais dinâmica, requisições assíncronas. O foco inicial, no entanto, é a renderização de templates no servidor.
    - **Jinja2:** Atua como o motor de templates padrão do Flask. Ele permite a criação de templates HTML dinâmicos, que são preenchidos com dados do backend antes de serem enviados ao navegador. Essa abordagem minimiza a necessidade de um framework JavaScript complexo para a renderização inicial do frontend.
- **Responsabilidades:**
    - Renderizar as páginas web que serão exibidas ao usuário.
    - Coletar as entradas do usuário por meio de formulários.

- Exibir os dados recebidos do backend.
- Gerenciar a navegação do usuário.

4.2. Camada de Aplicação/Negócio (Backend)

Esta camada representa a "inteligência" do sistema, onde a lógica de negócio é efetivamente executada.

- **Tecnologias Principais:**
  - **Python 3.x:** A linguagem de programação principal para o desenvolvimento do backend.
  - **Flask:** Um microframework web leve e flexível que serve como pilar para o desenvolvimento da API e das rotas do backend. Sua natureza modular permite construir funcionalidades de forma incremental.
  - **Flask-SQLAlchemy:** Uma extensão do Flask que facilita a integração com bancos de dados relacionais utilizando o ORM SQLAlchemy.
  - **Flask-Login:** Extensão dedicada ao gerenciamento de sessões de usuário e autenticação, simplificando o controle de acesso.
  - **Werkzeug/Jinja2 (integrados ao Flask):** Utilizados para roteamento, manipulação de requisições/respostas e renderização de templates.
- **Estrutura Lógica (Módulos/Componentes):**
  - app.py (Módulo Principal): Responsável pela inicialização da aplicação Flask, configuração básica e registro de blueprints.
  - **Módulos de Rotas (Blueprints):** Agrupam rotas relacionadas por funcionalidade (e.g., auth.py para autenticação, feira.py para gestão de feiras, expositor.py para gestão de expositores). Isso promove a modularidade e a organização do código.
  - **Módulos de Lógica de Negócio (Serviços):** Funções que implementam regras de negócio complexas e orquestram operações entre o banco de dados e as requisições. Exemplos incluem feira_service.py e expositor_service.py.
  - **Módulos de Modelo de Dados (Models):** Contêm as definições das classes que representam as entidades do banco de dados (Feira, Expositor, Produto, Ingresso, Usuário) e seus relacionamentos, utilizando o ORM SQLAlchemy.
- **Responsabilidades:**
  - Receber requisições HTTP do frontend.
  - Validar os dados de entrada.
  - Implementar toda a lógica de negócio, incluindo operações CRUD (Criar, Ler, Atualizar, Excluir) e regras de autorização.
  - Interagir com a Camada de Dados para consultas, inserções, atualizações e exclusões.
  - Gerenciar a autenticação e as sessões de usuário.
  - Preparar os dados e renderizar os templates HTML para a resposta ao frontend.

4.3. Camada de Dados

Esta camada é dedicada à persistência e recuperação dos dados do sistema.

- **Tecnologias Principais:**
    - **Banco de Dados Relacional: PostgreSQL** é a escolha preferencial devido à sua robustez, escalabilidade e suporte a recursos avançados. Para desenvolvimento local, **SQLite** pode ser utilizado por sua simplicidade e ausência de configuração, mantendo o foco em PostgreSQL para produção.
    - **SQLAlchemy (via Flask-SQLAlchemy):** Uma biblioteca ORM (Object-Relational Mapper) que abstrai as operações SQL diretas, permitindo a interação com o banco de dados usando objetos Python. Isso aprimora a legibilidade do código e minimiza erros de SQL.
- **Responsabilidades:**
    - Armazenar e recuperar todos os dados do sistema, incluindo feiras, expositores, produtos, ingressos e usuários.
    - Garantir a integridade referencial dos dados por meio de chaves primárias e estrangeiras.
    - Executar transações de banco de dados.

5. Fluxo de Dados e Comunicação

A comunicação entre o Navegador do Usuário e a Camada de Apresentação (Flask/Jinja2 renderizando HTML) ocorre por meio de **HTTP/HTTPS**. A Camada de Apresentação (Flask) recebe as requisições, processa-as através da Camada de Aplicação/Negócio, que por sua vez interage com a Camada de Dados. As respostas seguem o caminho inverso, com a Camada de Aplicação renderizando o HTML (via Jinja2) e enviando-o de volta ao navegador. A comunicação entre a Camada de Aplicação/Negócio e a Camada de Dados é gerenciada pelo ORM (SQLAlchemy), que traduz operações Python em comandos SQL e vice-versa.

6. Impacto das Ferramentas Usadas na Arquitetura

As ferramentas selecionadas exercem uma influência significativa na arquitetura do sistema:

- **Flask:** Sua natureza de microframework fomenta uma arquitetura modular, permitindo que diferentes partes da aplicação (como autenticação, feiras e expositores) sejam organizadas em **Blueprints** separados. Isso facilita a organização do código e a separação de responsabilidades. A escolha do Flask implica que a maior parte da interface do usuário será renderizada no servidor (Server-Side Rendering) por meio de templates Jinja2.
- **SQLAlchemy (e Flask-SQLAlchemy):** Abstrai a complexidade do SQL, capacitando a Camada de Aplicação a interagir com o banco de dados utilizando objetos Python. Isso resulta em maior produtividade e portabilidade do código do banco de dados, tornando a transição entre SGBDs (ex: de SQLite para PostgreSQL) menos complexa.
- **Jinja2:** Reforça o padrão de renderização do lado do servidor, onde o Flask compila as páginas HTML dinamicamente. Isso simplifica o frontend, pois dispensa a

necessidade de um framework JavaScript robusto para a construção de SPAs (Single Page Applications) complexas, adequando-se ao escopo do projeto.
- **PEP 8 e Ferramentas de Linting:** A adesão a padrões como PEP 8 e o uso de linters (e.g., Flake8) e formatadores (e.g., Black) promove um código consistente e limpo. Isso impacta diretamente a **manutenibilidade** do software, tornando-o mais fácil de ler, entender e depurar.

7. Considerações de Escalabilidade e Segurança

- A separação em camadas confere à arquitetura a capacidade de escalar cada camada de forma independente. Por exemplo, a camada de aplicação pode ser escalada horizontalmente (adicionando mais instâncias do Flask) sem impactar diretamente a camada de dados, desde que o banco de dados seja configurado para suportar essa carga.
- A segurança é abordada em cada camada do sistema:
    - **Frontend:** Inclui validações básicas (pré-submissão), embora a validação crítica seja sempre realizada no backend.
    - **Backend:** Realiza validação robusta de entradas, gerenciamento seguro de sessões (utilizando Flask-Login), hashing de senhas e controle de autorização baseado no criador do registro.
    - **Banco de Dados:** Garante a integridade referencial, e todas as operações são mediadas pela Camada de Aplicação, protegendo contra acessos diretos não autorizados.
    - **Comunicação:** Utiliza **HTTPS** para cifrar o tráfego de dados entre o cliente e o servidor.

8. Próximos Passos (Aprofundamento)

Para as próximas fases do projeto, serão necessários os seguintes aprofundamentos:

- Definir os modelos de dados detalhadamente, incluindo as classes Python para Feira, Expositor, etc..
- Criar diagramas de sequência ou fluxo para interações críticas, como o processo de login ou a criação de uma feira.
- Especificar bibliotecas Python adicionais que serão empregadas para tarefas específicas, como validação de e-mail ou manipulação de datas.