Descrição da Arquitetura do Software (Architecture Notebook)

Data: 20 de Maio de 2025

Versão: 1.0

Autor: Henrique Valente Lima

1. Introdução

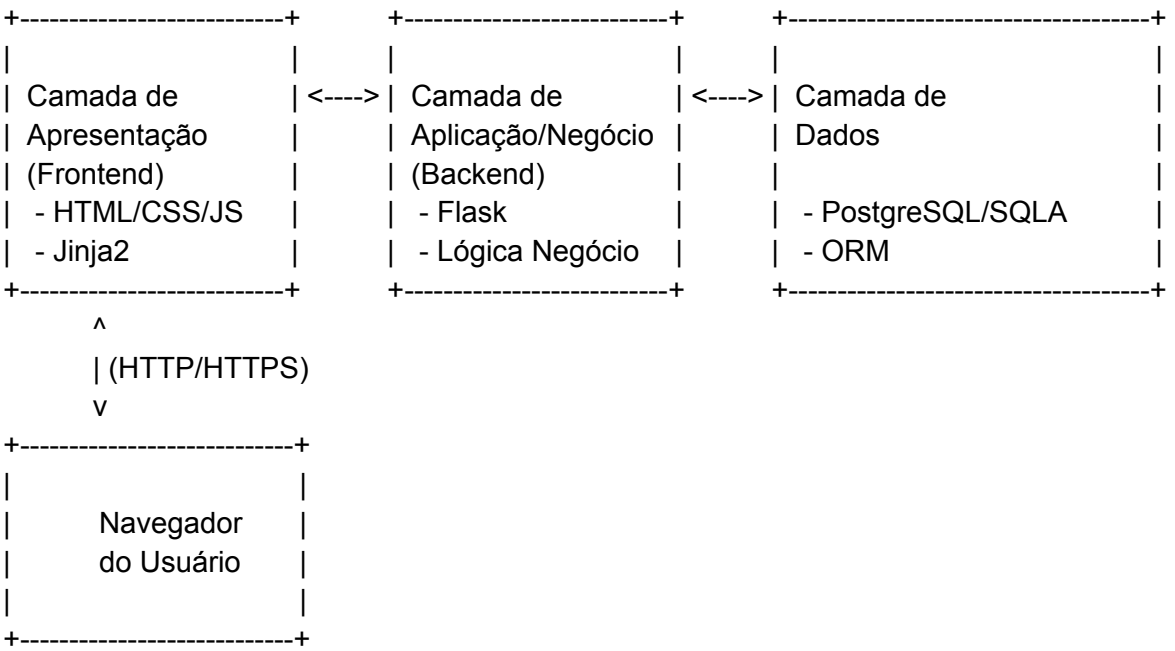
Este documento descreve a arquitetura do Sistema de Gestão de Feiras, detalhando seus componentes principais, os relacionamentos entre eles e as tecnologias que influenciam sua estrutura. A arquitetura proposta visa garantir modularidade, manutenibilidade, segurança e escalabilidade, conforme os requisitos não funcionais.

2. Visão Geral da Arquitetura

O sistema será construído seguindo uma **arquitetura em camadas (ou N-tier)**, que é um padrão comum e robusto para aplicações web. Essa abordagem promove a separação de responsabilidades, facilitando o desenvolvimento, a manutenção e a evolução do sistema. A arquitetura será dividida em três camadas principais:

1. **Camada de Apresentação (Frontend):** Responsável pela interface do usuário e pela interação direta com o navegador do cliente.
2. **Camada de Aplicação/Negócio (Backend):** Contém a lógica de negócios, controla o fluxo de dados e processa as requisições do frontend.
3. **Camada de Dados:** Responsável pelo armazenamento e gerenciamento persistente dos dados do sistema.

3. Diagrama da Arquitetura (Conceitual)



4. Descrição das Camadas e Elementos

4.1. Camada de Apresentação (Frontend)

Esta camada é o que o usuário final vê e interage diretamente.

- **Tecnologias Principais:**
    - **HTML5:** Estrutura das páginas web.
    - **CSS3:** Estilização e layout da interface, visando uma aparência limpa e responsiva.
    - **JavaScript:** Interatividade no lado do cliente (e.g., validações de formulário, requisições assíncronas se necessário para uma experiência mais dinâmica, embora o foco inicial seja mais baseado em renderização de templates).
    - **Jinja2:** Motor de templates padrão do Flask. Permite a criação de templates HTML dinâmicos, que são preenchidos com dados do backend antes de serem enviados ao navegador. Isso evita a necessidade de um framework JavaScript complexo para a renderização inicial do frontend.
- **Responsabilidades:**
    - Renderizar as páginas web que serão exibidas ao usuário.
    - Coletar entradas do usuário através de formulários.
    - Exibir os dados recebidos do backend.
    - Lidar com a navegação do usuário.

4.2. Camada de Aplicação/Negócio (Backend)

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

- **Tecnologias Principais:**
    - **Python 3.x:** Linguagem de programação principal para o desenvolvimento do backend.
    - **Flask:** Microframework web que servirá como o pilar para o desenvolvimento da API e das rotas do backend. Sua leveza e flexibilidade são ideais para este projeto, permitindo construir as funcionalidades de forma incremental.
    - **Flask-SQLAlchemy (ou similar):** Extensão do Flask que facilita a integração com bancos de dados relacionais utilizando o ORM SQLAlchemy.
    - **Flask-Login (ou similar):** Extensão para gerenciamento de sessões de usuário e autenticação, simplificando o controle de acesso.
    - **Werkzeug/Jinja2 (integrados ao Flask):** Para roteamento, requisições/respostas e renderização de templates.
- **Estrutura Lógica (Módulos/Componentes):**
    - **app.py (Módulo Principal):** Inicialização da aplicação Flask, configuração básica e registro de blueprints.
    - **Módulos de Rotas (Blueprints):** Agruparão rotas relacionadas (e.g., auth.py para autenticação, feira.py para gestão de feiras, expositor.py para gestão de expositores, etc.). Isso promove a modularidade e a organização do código.

    - **Módulos de Lógica de Negócio (Serviços):** Funções que implementam as regras de negócio complexas e orquestram operações entre o banco de dados e as requisições. Ex: feira_service.py, expositor_service.py.
    - **Módulos de Modelo de Dados (Models):** Definições das classes que representam as entidades do banco de dados (Feira, Expositor, Produto, Ingresso, Usuário) e seus relacionamentos, utilizando um ORM (e.g., SQLAlchemy).
- **Responsabilidades:**
    - Receber requisições HTTP do frontend.
    - Validar dados de entrada.
    - Implementar toda a lógica de negócio (CRUD, regras de autorização).
    - Interagir com a Camada de Dados (consultas, inserções, atualizações, exclusões).
    - Gerenciar autenticação e sessões de usuário.
    - Preparar os dados e renderizar os templates HTML para resposta ao frontend.

4.3. Camada de Dados

Esta camada é responsável pela 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. Alternativamente, **SQLite** pode ser usado para o desenvolvimento local por sua simplicidade e ausência de configuração, embora o foco seja em PostgreSQL para produção.
    - **SQLAlchemy (via Flask-SQLAlchemy):** Biblioteca ORM (Object-Relational Mapper) que abstrai as operações diretas em SQL, permitindo interagir com o banco de dados usando objetos Python. Isso melhora a legibilidade do código e reduz a chance de erros de SQL.
- **Responsabilidades:**
    - Armazenar e recuperar todos os dados do sistema (feiras, expositores, produtos, ingressos, usuários).
    - Garantir a integridade referencial dos dados através 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 via **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 pela ORM (SQLAlchemy), que traduz operações Python em comandos SQL e vice-versa.

6. Impacto das Ferramentas Usadas na Arquitetura

As ferramentas escolhidas influenciam a arquitetura da seguinte forma:

- **Flask:** Sua natureza de microframework incentiva uma arquitetura modular, onde diferentes partes da aplicação (e.g., autenticação, feiras, expositores) podem ser organizadas em **Blueprints** separados. Isso facilita a organização do código e a separação de responsabilidades. A escolha do Flask implica que a renderização da maior parte da interface do usuário será feita no servidor (Server-Side Rendering) através de templates Jinja2.
- **SQLAlchemy (e Flask-SQLAlchemy):** Abstrai a complexidade do SQL, permitindo que a Camada de Aplicação interaja com o banco de dados usando objetos Python. Isso melhora a produtividade e a portabilidade do código de banco de dados, tornando a mudança de SGBD (ex: de SQLite para PostgreSQL) menos traumática.
- **Jinja2:** Fortalece o padrão de renderização do lado do servidor, onde o Flask compila as páginas HTML dinamicamente. Isso simplifica o frontend, pois não exige um framework JavaScript pesado 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) promovem um código consistente e limpo. Isso diretamente impacta a **manutenibilidade** do software, tornando-o mais fácil de ler, entender e depurar.

7. Considerações de Escalabilidade e Segurança

- A separação em camadas permite que cada camada seja potencialmente escalada de forma independente. Por exemplo, a camada de aplicação pode ser escalada horizontalmente (adicionando mais instâncias do Flask) sem afetar diretamente a camada de dados, desde que o banco de dados seja configurado para suportar essa carga.
- A segurança é abordada em cada camada:
    - **Frontend:** Validações básicas (pré-submissão), embora a validação crítica seja no backend.
    - **Backend:** Validação robusta de entradas, gerenciamento seguro de sessões (Flask-Login), hashing de senhas, controle de autorização baseado no criador do registro.
    - **Banco de Dados:** Integridade referencial, e as operações são mediadas pela Camada de Aplicação, protegendo contra acesso direto não autorizado.
    - **Comunicação:** HTTPS para cifrar o tráfego entre cliente e servidor.

8. Próximos Passos (Aprofundamento)

- Definir os modelos de dados detalhadamente (classes Python para Feira, Expositor, etc.).
- Criar diagramas de sequência ou fluxo para interações críticas (e.g., login, criação de feira).

- Especificar bibliotecas Python adicionais que serão utilizadas para tarefas específicas (e.g., validação de email, manipulação de datas).