Descrição da Arquitetura do Software: Sistema de Divulgação e Gestão de Feiras

1. Visão Geral

A arquitetura do "Sistema de Divulgação e Gestão de Feiras" foi projetada como uma aplicação web monolítica, utilizando o framework **Django** em Python. Essa escolha se baseia na necessidade de um desenvolvimento rápido, seguro e escalável, alinhado aos requisitos funcionais e não funcionais do projeto.

A arquitetura segue o padrão **Model-View-Template (MVT)**, uma variação do conhecido padrão Model-View-Controller (MVC), que promove uma clara separação de responsabilidades, atendendo diretamente ao requisito de manutenibilidade e modularidade do código.

2. Padrão Arquitetural: Model-View-Template (MVT)

O MVT organiza a aplicação em três componentes principais que trabalham em conjunto com o despachante de URLs do Django:

- **Model (Modelo):** Representa a estrutura dos dados da aplicação e a lógica de negócios fundamental. É a única camada que interage diretamente com o banco de dados.
    - **Implementação:** Em nosso sistema, os modelos Feira, Expositor, Produto e, futuramente, Ingresso definem os dados e seus relacionamentos (ex: um expositor pode participar de várias feiras). Eles são responsáveis por garantir a integridade e as regras de negócio dos dados.

- **View (Visão):** É a camada de lógica que processa as requisições do usuário. A View recebe a requisição, interage com os Models para buscar ou manipular dados e, em seguida, seleciona um Template para renderizar a resposta.
    - **Implementação:** Nossas funções em views.py (como pagina_inicial, detalhes_feira) atuam como as Views. Elas orquestram a busca de dados e os enviam para a camada de apresentação, além de controlarem o acesso com base no tipo de usuário, conforme o requisito de autenticação.

- **Template (Gabarito):** É a camada de apresentação, responsável por renderizar a interface do usuário (UI). Consiste em arquivos HTML com uma sintaxe especial (Django Template Language) que permite exibir os dados preparados pela View de forma dinâmica.
    - **Implementação:** Nossos arquivos .html (como pagina_inicial.html) são os Templates. Eles garantem que a interface seja responsiva e acessível em diferentes dispositivos, cumprindo o requisito da HU23.

- **URL Dispatcher (Despachante de URLs):** Embora não seja parte formal do MVT, é um componente central do Django. Ele analisa a URL da requisição e a direciona para a View correspondente, desacoplando as URLs da lógica da aplicação.
    - **Implementação:** Nossos arquivos urls.py definem os padrões de URL do sistema.

3. Componentes da Arquitetura e Tecnologias

- **Backend:**
    - **Framework:** Django (Python)
    - **Linguagem:** Python 3
- **Frontend:**
    - **Tecnologias:** HTML5, CSS3, Javascript. A interface é renderizada no lado do servidor (Server-Side Rendering) pelo Django, o que simplifica o desenvolvimento inicial e garante bom desempenho.
- **Banco de Dados:**
    - **Desenvolvimento:** SQLite (padrão do Django), pela sua simplicidade e por não exigir configuração.
    - **Produção (Proposta):** PostgreSQL, devido à sua robustez, escalabilidade e recursos avançados, garantindo a confiabilidade e a capacidade de recuperação do sistema.

4. Visão de Implantação (Proposta para Produção)

Para garantir a eficiência e a disponibilidade do sistema, a arquitetura de implantação proposta é a seguinte:

1. **Cliente (Navegador):** O usuário acessa o sistema através de um navegador web em seu desktop ou dispositivo móvel.
2. **Servidor Web (Nginx):** Atua como proxy reverso e servidor de arquivos estáticos (CSS, JS, imagens). Ele recebe as requisições e as encaminha para o servidor de aplicação.
3. **Servidor de Aplicação (Gunicorn):** É responsável por executar a aplicação Django, gerenciando múltiplos processos para lidar com requisições concorrentes.
4. **Aplicação Django:** Onde a lógica do sistema (MVT) é executada.
5. **Banco de Dados (PostgreSQL):** Onde os dados são persistidos de forma segura.

Essa estrutura é um padrão de mercado para aplicações Django, garantindo escalabilidade e separação de responsabilidades no ambiente de produção, e está preparada para uma futura migração para serviços em nuvem.

5. Visão de Dados: Elementos e Relacionamentos

Esta seção detalha a estrutura de dados da aplicação, que foi implementada utilizando o ORM (Object-Relational Mapper) do framework Django. Os dados são organizados em modelos que representam as entidades centrais do sistema.

Descrição dos Elementos (Modelos)

Os modelos são as representações diretas das entidades de negócio. Para este sistema, os seguintes modelos foram definidos:

- **User (Modelo Nativo do Django):** É a base do sistema de autenticação. Armazena informações essenciais de login, como nome de usuário, email e senha criptografada. Ele serve como pilar para os diferentes papéis de usuário na plataforma.
- **Organizador:** Um perfil que estende o modelo User através de uma relação OneToOneField. Representa um usuário com permissões para criar e gerenciar eventos (Feira). Contém informações adicionais como o nome da empresa organizadora.
- **Expositor:** Outro perfil que estende o User via OneToOneField. Representa um usuário que pode participar de feiras para divulgar e vender seus produtos. Contém descricao e contato próprios.
- **Feira:** A entidade central do sistema, representando um evento. Armazena todas as informações pertinentes a uma feira, como nome, localidade e datas.
- **Produto:** Representa um item que pode ser vendido ou exposto por um Expositor. Cada produto está estritamente associado a um único expositor.
- **Ingresso:** Atua como uma entidade de ligação, representando a participação de um User (visitante) em uma Feira específica. Registra a data de emissão.

Descrição dos Relacionamentos

As conexões entre os modelos são cruciais para a lógica de negócios e são gerenciadas pelo Django ORM. Os principais relacionamentos são:

- **User ↔ Perfis (Organizador e Expositor):**
  - **Tipo:** Um-para-Um (OneToOneField).
  - **Descrição:** Cada User pode ser, no máximo, um Organizador ou um Expositor. Esta relação permite especializar o usuário padrão do Django, adicionando campos e comportamentos específicos para cada papel sem modificar o modelo de autenticação principal.
- **Organizador → Feira:**
  - **Tipo:** Um-para-Muitos (ForeignKey de Feira para Organizador).
  - **Descrição:** Um Organizador pode criar e ser responsável por várias Feiras, mas cada Feira possui apenas um Organizador.

- **Expositor → Produto:**
    - **Tipo:** Um-para-Muitos (ForeignKey de Produto para Expositor).
    - **Descrição:** Um Expositor pode cadastrar múltiplos Produtos, mas cada Produto pertence a um único Expositor.
- **Feira ↔ Expositor:**
    - **Tipo:** Muitos-para-Muitos (ManyToManyField), implementado com duas listas.
    - **Descrição:** Este é o relacionamento mais complexo, gerenciando o fluxo de inscrição e participação. Foi implementado com dois campos ManyToManyField no modelo Feira: expositores_aprovados e expositores_pendentes. Essa abordagem permite que o Organizador gerencie as inscrições (movendo um expositor da lista de pendentes para a de aprovados) sem a necessidade de um modelo intermediário (through model), simplificando a lógica da aplicação.
- **Ingresso (Ligação entre User e Feira):**
    - **Tipo:** O modelo Ingresso possui duas relações Um-para-Muitos (ForeignKey para User e ForeignKey para Feira).
    - **Descrição:** Ele funciona como uma tabela de associação que registra que um User específico emitiu um ingresso para uma Feira específica, criando uma relação Muitos-para-Muitos efetiva entre visitantes e eventos.

6. Impacto das Ferramentas (Django ORM e Migrations)

A escolha de utilizar o Django e seu ORM teve um impacto direto e positivo na arquitetura e no desenvolvimento:

- **Abstração e Produtividade:** O ORM permitiu que toda a lógica de banco de dados fosse escrita em Python, abstraindo a complexidade da linguagem SQL. Isso acelerou o desenvolvimento e reduziu a probabilidade de erros, como ataques de SQL Injection.
- **Integridade Referencial:** O uso de ForeignKey e on_delete=models.CASCADE garante que o banco de dados mantenha sua integridade. Por exemplo, ao deletar um User, seu perfil de Expositor e todos os seus Produtos associados são automaticamente removidos, evitando dados órfãos.
- **Versionamento do Schema:** O sistema de **migrações** do Django (makemigrations e migrate) atuou como um sistema de controle de versão para a estrutura do banco de dados. Cada alteração nos modelos foi registrada em um arquivo de migração, permitindo que a estrutura do banco evoluísse de forma segura e rastreável junto com o código da aplicação, o que é fundamental para a manutenibilidade.

7. Atendimento aos Requisitos Não Funcionais

A arquitetura escolhida atende diretamente aos requisitos não funcionais definidos no documento:

- **Eficiência de Desempenho:** O uso de um servidor de aplicação como Gunicorn e um servidor web como Nginx em produção otimiza o uso de recursos. O ORM do Django também oferece mecanismos de otimização de consultas.
- **Compatibilidade e Interoperabilidade:** A arquitetura baseada em Django permite que a aplicação coexista com outros serviços no mesmo ambiente de hospedagem. Além disso, o padrão modular do Django facilita a criação de APIs (Interfaces de Programação de Aplicação) para futuras integrações.
- **Usabilidade e Confiabilidade:** O Django é um framework maduro e estável, o que contribui para a confiabilidade do sistema. A renderização no lado do servidor permite a construção de interfaces limpas e intuitivas, conforme solicitado.
- **Segurança:** O Django fornece um robusto sistema de segurança embarcado, que inclui:
    - Proteção contra ataques comuns (XSS, CSRF, SQL Injection).
    - Um sistema de autenticação e controle de permissões por tipo de usuário.
    - Gerenciamento seguro de sessões.
- **Manutenibilidade:** A separação de responsabilidades do padrão MVT e o uso de "apps" no Django garantem a modularidade do código, facilitando sua análise, modificação e teste.