Universidade de Brasília
INSTITUTO DE EXATAS - IE
DEPARTAMENTO DE CIÊNCIAS DA COMPUTAÇÃO - CIC
ENGENHARIA DE SOFTWARE 1/2025

DESCRIÇÃO DA ARQUITETURA DO SOFTWARE

Discentes: BRUNO EDUARDO DOS SANTOS - 211066249
GEILSON DOS SANTOS SA - 231006239
MARCOS ALEXANDRE DA SILVA NERES - 211055334
PEDRO FARIAS DE OLIVEIRA - 211055577
THELMA EVANGELISTA DOS SANTOS - 231003513
WESLEY HENRIQUE FERREIRA - 231021496

Sistema de Gestão de Feiras	
Descrição da Arquitetura do Software	Data: 16/06/2025

1. Metas Arquiteturais e Filosofia

A arquitetura do sistema seguirá uma estrutura modular inspirada nos princípios da SOA – Service-Oriented Architecture será construída com base no padrão MVC (Model-View-Controller)**.** Essa combinação visa promover separação de responsabilidades, coerência estrutural, e flexibilidade para manutenção e evolução. O foco está em garantir que o sistema seja escalável, seguro e sustentável a longo prazo.

Filosofia da Arquitetura

A filosofia adotada prioriza a modularidade lógica e a clara delimitação entre camadas, de forma que cada componente possa evoluir de maneira independente. Embora a implantação inicial utilize uma arquitetura monolítica, a estrutura modular adotada permite futura adaptação para microsserviços ou aplicações móveis/PWA), conforme a evolução das necessidades.

Além disso, o sistema será desenvolvido com atenção à segurança, devido à manipulação de dados sensíveis e autenticados, exigindo controles rigorosos de autorização, integridade referencial e rastreabilidade.

Questões que moldam a filosofia:

- **Segurança e controle de acesso**: devido às operações com usuários autenticados e permissões específicas, o sistema requer autenticação robusta (JWT) e verificação de autoria.
- **Manutenção e evolução**: o sistema será mantido por ciclos contínuos de melhoria, exigindo uma base de código limpa, testável e extensível.
- **Desempenho sob concorrência moderada**: funcionalidades como compra de ingressos e cadastro de produtos devem responder com agilidade, sem

comprometer a integridade dos dados.
- **Facilidade de migração futura**: a arquitetura visa permitir transição futura para arquiteturas mais distribuídas ou aplicações mobile/web progressivas, com esforço mínimo de reestruturação.

2. Premissas e Dependências

A definição da arquitetura do sistema foi guiada por uma série de premissas técnicas, operacionais e organizacionais que afetam diretamente as decisões de projeto. Esses fatores influenciam tanto a escolha das tecnologias quanto a estruturação das camadas e mecanismos de segurança do sistema.

Premissas:

- O sistema será acessado via navegador web, sendo necessário um servidor web para hospedar a aplicação.
- O backend será implementado em Node.js com o framework Express, utilizando ORM para abstração no acesso ao banco de dados relacional.
- A autenticação e autorização de usuários será feita com JSON Web Tokens (JWT).
- A implantação inicial será emarquitetura monolítica, com todos os componentes rodando em um único servidor local, embora com possibilidade de evolução futura para ambientes distribuídos.

Dependências e fatores críticos:

- **Dependência de tecnologias específicas:** A arquitetura assume familiaridade da equipe com JavaScript, Node.js, Express, React e PostgreSQL. Alterações nessas tecnologias exigiriam requalificação ou substituição da equipe.
- **Acesso a recursos de infraestrutura:** O desenvolvimento e os testes dependem da disponibilidade de ambientes com suporte a containers (por exemplo, Docker), especialmente para o banco de dados e possíveis serviços auxiliares.
- **Dados sensíveis e controle de acesso:** Como o sistema manipula informações de usuários autenticados, é fundamental garantir segurança nas camadas de autenticação e persistência, além de garantir a integridade das regras de negócio.
- **Simplicidade de manutenção inicial:** A escolha por uma arquitetura monolítica reduz a complexidade inicial de implantação e facilita a entrega rápida, mas exige atenção à modularidade interna para não comprometer a escalabilidade futura.
- **Ausência de legado a integrar:** O sistema será desenvolvido do zero, sem dependências de sistemas legados, o que permite maior liberdade de escolha tecnológica.
- **Equipe de desenvolvimento reduzida:** A arquitetura deve ser simples o suficiente para ser compreendida e mantida por uma equipe pequena, o que reforça a escolha por padrões conhecidos e tecnologias com ampla documentação.

3. Requisitos Arquiteturalmente Significativos

Os seguintes requisitos, especificados no artefato de Requisitos Funcionais (RF) do sistema, impactam diretamente as decisões arquiteturais e moldam o design e a implementação da solução:

- **Implementação completa de operações CRUD** para as entidades: Feiras, Expositores, Produtos e Ingressos (*RF-002*, *RF-003*, *RF-004*, *RF-005*, *RF-011*, *RF-012*, *RF-013*, *RF-014*).
- **Controle de acesso com autenticação e autoria de registros**: apenas o criador pode editar ou excluir suas entidades (*RF-011*, *RF-012*, *RF-013*).
- **Restrições de exclusão com base em integridade referencial**: não permitir exclusão de feiras com expositores ou de expositores com produtos vinculados (*RF-012*, *RF-013*).
- **Acesso público para listagens e detalhes**, e acesso restrito às ações de modificação (*RF-006*, *RF-007*, *RF-008*, *RF-010*).
- **Validação de dados obrigatórios e unicidade de informações sensíveis**, como nome de feira (*RF-002*, *RF-011*).
- **Expiração de sessões após inatividade**, para garantir segurança (requisito implícito de segurança derivado do uso de JWT e práticas recomendadas).
- **Controle de concorrência em operações críticas**, como aquisição de ingressos (*RF-005*).
- **Geração e exportação de relatórios administrativos** em formatos estruturados (requisito funcional previsto, ainda não especificado formalmente).
- **Modularização suficiente para futura migração para microsserviços ou PWAs** (motivação arquitetural com base na filosofia de evolução).
- **Envio de notificações assíncronas por e-mail**, como confirmações ou atualizações de eventos (previsto como mecanismo de usabilidade e engajamento futuro).

4. Decisões, Restrições e Justificativas

Esta seção descreve as principais decisões arquiteturais tomadas e as restrições impostas ao desenvolvimento, com suas respectivas justificativas. Esses elementos devem ser observados por toda a equipe para garantir coerência técnica, manutenibilidade e alinhamento com os objetivos do projeto.

Decisões e Justificativas

- **Uso do padrão MVC**
  A arquitetura segue o padrão Model-View-Controller para separar claramente responsabilidades entre as camadas de apresentação, controle e lógica de negócio. Isso facilita testes, organização de código, manutenção e futura evolução do sistema.
- **Stack com Node.js + Express + ORM**
  A escolha por Node.js e Express visa agilidade de desenvolvimento, facilidade de integração com bibliotecas e compatibilidade com REST. O uso de ORM (como Sequelize ou Prisma) abstrai o acesso ao banco de dados relacional (PostgreSQL), garantindo consistência entre o modelo de domínio e as tabelas persistidas.

- **Validações obrigatórias no backend**
  Todas as regras de negócio e validações de integridade devem ser **r**eaplicadas no backend, mesmo que existam verificações no frontend. Isso evita manipulação maliciosa de dados via requisições diretas e assegura integridade no nível da aplicação.
- **Autenticação com JWT (JSON Web Tokens)**
  O sistema utiliza JWT para autenticação stateless, o que permite escalabilidade, sessões simultâneas e facilidade de integração com APIs RESTful, eliminando a necessidade de armazenamento de sessão no servidor.
- **Implantação com arquitetura monolítica (inicial)**
  Para reduzir a complexidade nas etapas iniciais, o sistema será desenvolvido e implantado como uma aplicação monolítica. No entanto, será estruturado com modularização interna suficiente para facilitar a migração futura para microsserviços ou arquitetura orientada a serviços.

Restrições e Recomendações (DOs and DON'Ts)

DOs (Boas práticas que devem ser seguidas):

- Estruture o código seguindo o padrão MVC, com diretórios separados para `models`, `controllers`, `services` e `routes`.
- Utilize o ORM para todas as operações de banco de dados (evite SQL direto).
- Implemente autenticação e autorização de forma centralizada, via middlewares.
- Reforce as validações de entrada e regras de negócio no backend, mesmo que o frontend valide.
- Planeje os módulos com acoplamento fraco e coesão alta para facilitar manutenção e testes.

DON'Ts (Práticas proibidas ou desaconselhadas):

- Não insira lógica de negócio diretamente nos controladores.
- Não acesse o banco de dados diretamente sem passar pelos modelos e serviços.
- Não armazene dados de sessão no servidor (deve ser stateless via JWT).
- Não misture responsabilidades entre camadas (ex: validações de regra de negócio no frontend apenas).
- Evite estruturas rígidas que impeçam a migração para uma arquitetura modular no futuro.

6. Mecanismos Arquiteturais

A seguir, são listados os mecanismos arquiteturais definidos até o momento no sistema. Cada item apresenta uma descrição inicial e o estado atual de implementação ou planejamento. Esses mecanismos evoluirão até se consolidarem como colaborações ou padrões aplicáveis ao design.

Autenticação

- **Descrição**: Responsável por identificar e validar usuários autenticados.
- **Estado atual:** Implementado com JSON Web Tokens (JWT), em modelo stateless, com expiração automática configurada.

Autorização e Controle de Acesso

- **Descrição:** Restringe ações de modificação e exclusão a usuários com permissão adequada.
- **Estado atual:** Baseado no campo `usuario_criador_id`, já validado na lógica de negócio.

Controle de Autoria

- **Descrição:** Assegura a vinculação de registros ao usuário que os criou.
- **Estado atual:** Presente em todas as entidades relevantes e aplicado nas validações de edição e exclusão.

Integridade Referencial

- **Descrição:** Mantém a consistência dos vínculos entre entidades relacionadas.
- **Estado atual:** Garantida por constraints no banco de dados e reforçada por validações no backend.

Regras de Exclusão Condicional

- **Descrição:** Impede a exclusão de registros vinculados a outros recursos.
- **Estado atual:** Aplicada na lógica de negócio para casos como feiras com expositores ativos.

Persistência de Dados

- **Descrição:** Gerencia o armazenamento e recuperação dos dados da aplicação.
- **Estado atual:** Implementada com ORM (Sequelize), mapeando as entidades ao banco PostgreSQL.

Validação de Dados

- **Descrição:** Garante que os dados inseridos atendam aos critérios de completude, formato e unicidade.
- **Estado atual:** Validações já ativas no backend, abrangendo campos obrigatórios e dados sensíveis como CPF, CNPJ e nomes únicos.

Sessões de Usuário (Stateless)

- **Descrição:** Controla a sessão do usuário sem armazenamento no servidor.
- **Estado atual:** Gerenciada via JWT; a expiração do token controla a validade da sessão.

Concorrência e Transações

- **Descrição:** Previne conflitos em operações simultâneas de leitura e escrita.
- **Estado atual:** Em fase de análise, com foco em cenários críticos como aquisição de ingressos.

Exportação de Relatórios

- **Descrição:** Permite a geração de relatórios administrativos em formatos estruturados.
- **Estado atual:** Planejado para etapas futuras; será implementado de forma modular, com suporte a PDF e CSV.

Notificações Assíncronas

- **Descrição:** Envia alertas e confirmações por e-mail de forma automática.
- **Estado atual:** Previsto para implementação posterior, via integração com serviços externos como SMTP ou APIs de terceiros.

6. Mecanismos Arquiteturais 1: Autenticação, Autorização e Sessão

Esse mecanismo é responsável pela identificação, autorização e controle de sessão dos usuários autenticados no sistema. Ele assegura que apenas usuários válidos e autorizados possam executar operações sensíveis, mantendo o estado da sessão de forma segura.

Propósito:

Proteger endpoints e dados sensíveis, validando quem acessa o sistema e garantindo que só os responsáveis modifiquem seus próprios dados.

Atributos:

- Implementação com JWT (JSON Web Tokens).
- Sessões são stateless — não armazenadas no servidor.
- Tokens possuem expiração configurada, forçando revalidação após inatividade.
- Controle de autoria baseado no campo `usuario_criador_id`.

Funções:

- Validar identidade do usuário no login.
- Proteger rotas privadas com middleware JWT.
- Garantir que ações (como edição/exclusão) só sejam executadas por quem criou o recurso.
- Bloquear acesso indevido e evitar vazamento de informações.

7. Mecanismos Arquiteturais 2: Camadas da Arquitetura

A estrutura do sistema segue o padrão Model-View-Controller (MVC), adaptado ao desenvolvimento web com separação clara entre responsabilidades, visando manutenção, escalabilidade e modularidade.

Camada de Apresentação (View)

- **Propósito:** Exibir dados e interagir com o usuário.
- **Atributos:** Desenvolvida com HTML, CSS, JavaScript e React.

- **Função:** Enviar requisições e apresentar respostas visuais da aplicação.

Camada de Controle (Controller)

- **Propósito:** Fazer a ponte entre o frontend e a lógica de negócio.
- **Atributos:** Endpoints RESTful via Express.js.
- **Função:** Interpretar requisições, chamar serviços e devolver respostas HTTP.

Camada de Negócio (Service/Model)

- **Propósito:** Centralizar lógica e regras da aplicação.
- **Atributos:** Organizada por entidades com funções reutilizáveis.
- **Função:** Validar permissões, aplicar regras de negócio e interagir com a camada de dados.

Camada de Persistência (Repository)

- **Propósito:** Gerenciar acesso ao banco de dados.
- **Atributos:** Implementada com ORM (Sequelize/Prisma) e PostgreSQL.
- **Função:** Executar operações CRUD com consistência e segurança.

8. Visões Arquiteturais

As visões arquiteturais fornecem diferentes perspectivas sobre a estrutura e o funcionamento do sistema, com base no modelo 4+1 Views. Cada visão representa um mecanismo arquitetural com propósito específico no processo de análise, projeto e comunicação do sistema.

Visão Lógica (Logical View)

- **Propósito:** Representar a estrutura estática do sistema, evidenciando os principais elementos de domínio e suas interações.
- **Atributos:** Diagramas de classes e pacotes, mostrando entidades como Feira, Expositor, Produto, Ingresso e Usuário, além de seus relacionamentos (associações, composições, heranças) e atributos relevantes.
- **Função:** Auxiliar no entendimento da lógica de negócio, na definição da estrutura de dados persistente e na organização dos componentes de software durante o desenvolvimento.

Visão de Processo (Process View)

- **Propósito:** Representar os aspectos dinâmicos do sistema, como concorrência, paralelismo e comunicação entre processos ou componentes em tempo de execução.
- **Atributos:** Controle de sessões simultâneas por usuários autenticados; uso de middleware (Express) para roteamento e controle de requisições; implementação de middlewares assíncronos para autenticação (JWT) e controle de acesso.
- **Função:** Apoiar decisões relacionadas ao desempenho, controle de concorrência e integridade de transações, como na aquisição de ingressos ou expiração de sessões.

Visão de Desenvolvimento (Development View)

- **Propósito:** Descrever a organização do software no ambiente de desenvolvimento, evidenciando a modularidade, reutilização e estruturação do código-fonte.
- **Atributos:** Organização do sistema em módulos seguindo o padrão MVC, com separação de camadas em diretórios (`controllers`, `models`, `routes`, `services`, etc.); uso de ferramentas como Node.js, ORM (Sequelize) e React.
- **Função:** Facilitar a manutenção, testes e evolução do sistema, promovendo a reutilização de componentes e a separação clara de responsabilidades no código.

Visão de Implantação (Deployment View)

- **Propósito:** Especificar a estrutura física de execução do sistema, incluindo os componentes de hardware e as unidades de software implantadas.
- **Atributos:** Aplicação monolítica hospedada em um único servidor local, com o banco de dados PostgreSQL executando em um serviço separado (container ou processo independente).
- **Função:** Guiar decisões de infraestrutura, identificar pontos de comunicação entre processos e orientar a configuração do ambiente de produção e testes.

Visão de Casos de Uso (Use Case View)

- **Propósito:** Demonstrar o comportamento funcional do sistema a partir da perspectiva dos usuários e suas interações com as funcionalidades oferecidas.
- **Atributos:** Casos de uso como cadastro de feiras, visualização pública de eventos, criação de produtos por expositores, aquisição de ingressos e autenticação de usuários.
- **Função:** Servir como base para a definição dos requisitos funcionais, validação do escopo do sistema e mapeamento dos fluxos de controle que serão implementados na camada de lógica.