FECAP 2025

MODELAGEM DE SOFTWARE 3 - ARQUITETURA COMPLETA

Integrantes do Projeto:

Vitor Santos - RA: 23025502

Gustavo Henrique - RA: 24026874

Gustavo Roberto - RA: 24026770

Lucas Alves Bernardo - RA: 24026768

Eriane S O Dias - RA: 24026678

Sumário

FECAP 2025

Engenharia e Arquitetura de Software — TechTeam

1. Visão Geral

O sistema tem como objetivo permitir que qualquer usuário visualize eventos disponíveis, realize cadastro, login e se inscreva em eventos. A solução busca ser intuitiva para o usuário final, e modular e escalável para o time de desenvolvimento.

A arquitetura adotada segue o padrão em camadas, promovendo separação de responsabilidades, manutenibilidade e reutilização de código.

2. Arquitetura em Camadas

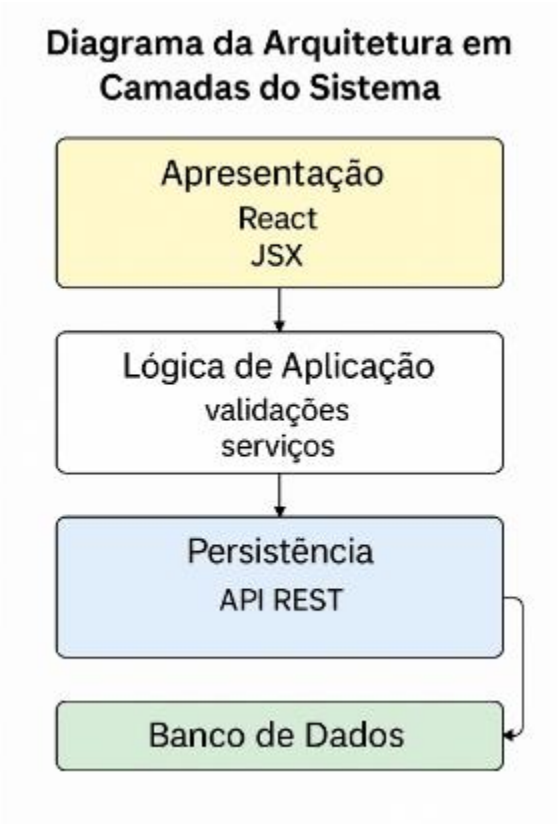

Figura 1 – Diagrama da Arquitetura em Camadas do Sistema

2.1. Camada de Apresentação

A camada de apresentação é responsável pela interface com o usuário. Todas as interações visuais, formulários, botões e páginas fazem parte desta camada. As funcionalidades presentes incluem:

- Visualização de eventos sem necessidade de login
- Cadastro e autenticação de usuários

- Preenchimento de formulários
- Inscrição em eventos
- Acesso a áreas internas por usuários autenticados

A aplicação foi desenvolvida com React, utilizando JavaScript, CSS e ferramentas de build como Vite.js. A estrutura está organizada com base em diretórios que seguem a lógica de páginas e componentes, promovendo reutilização e clareza no desenvolvimento.

Pastas principais na camada de apresentação:
- components: componentes reutilizáveis de UI
- pages: páginas do site (Homepage, Login, Signup, Eventos, etc.)
- App.jsx: ponto central da aplicação com configuração de rotas
- index.jsx: ponto de entrada que renderiza o app

Esta estrutura reflete uma clara separação entre interface, layout e lógica, facilitando a evolução futura da aplicação.

2.2. Camada de Negócio (Lógica de Aplicação)

A camada de negócio é responsável por centralizar as regras da aplicação, garantindo que as operações sigam a lógica correta antes de serem executadas ou enviadas para a persistência de dados.

Responsabilidades da camada de negócio:
- Validação de dados de entrada
- Controle de fluxos, como inscrição em eventos
- Processamento de regras específicas do domínio
- Comunicação com a API (camada de persistência)
- Controle de autenticação (login, acesso autorizado)
- Gerenciamento de estado intermediário da aplicação

A arquitetura sugere que esta lógica seja isolada em arquivos ou módulos específicos (como serviços), promovendo o desacoplamento entre a lógica e a interface. Isso facilita a manutenção, testes e reutilização da lógica em diferentes partes da aplicação.

2.3. Camada de Persistência (API / Banco de Dados)

A camada de persistência do sistema é responsável por armazenar, consultar e manter os dados de forma estruturada e segura. O projeto utiliza o banco de dados relacional MySQL, amplamente adotado em aplicações web, em conjunto com a biblioteca Sequelize, que atua como ORM (Object-Relational Mapping).

O back-end é implementado em Node.js com o framework Express, e utiliza o driver mysql2 para se conectar diretamente ao banco. As rotas de comunicação são desenvolvidas em JavaScript e organizadas no servidor, permitindo a integração entre o front-end (React) e a

camada de dados.

Durante o processo de desenvolvimento, também são utilizadas bibliotecas complementares como:
- bcryptjs: para criptografia de senhas
- jsonwebtoken: para autenticação via tokens JWT
- multer: para upload de arquivos
- dotenv (via nodemon): para carregar variáveis de ambiente de forma segura

3. Organização Modular

A aplicação está organizada de forma modular, com pastas e arquivos divididos por responsabilidade. Cada página ou funcionalidade tem seus componentes e lógica separados, o que facilita a manutenção e escalabilidade do projeto.

Pontos positivos da organização:
- Separação entre páginas, componentes e lógica
- Componentes reutilizáveis promovem DRY (Don’t Repeat Yourself)
- Preparação clara para integração com API externa
- Adaptação facilitada a novos requisitos ou melhorias

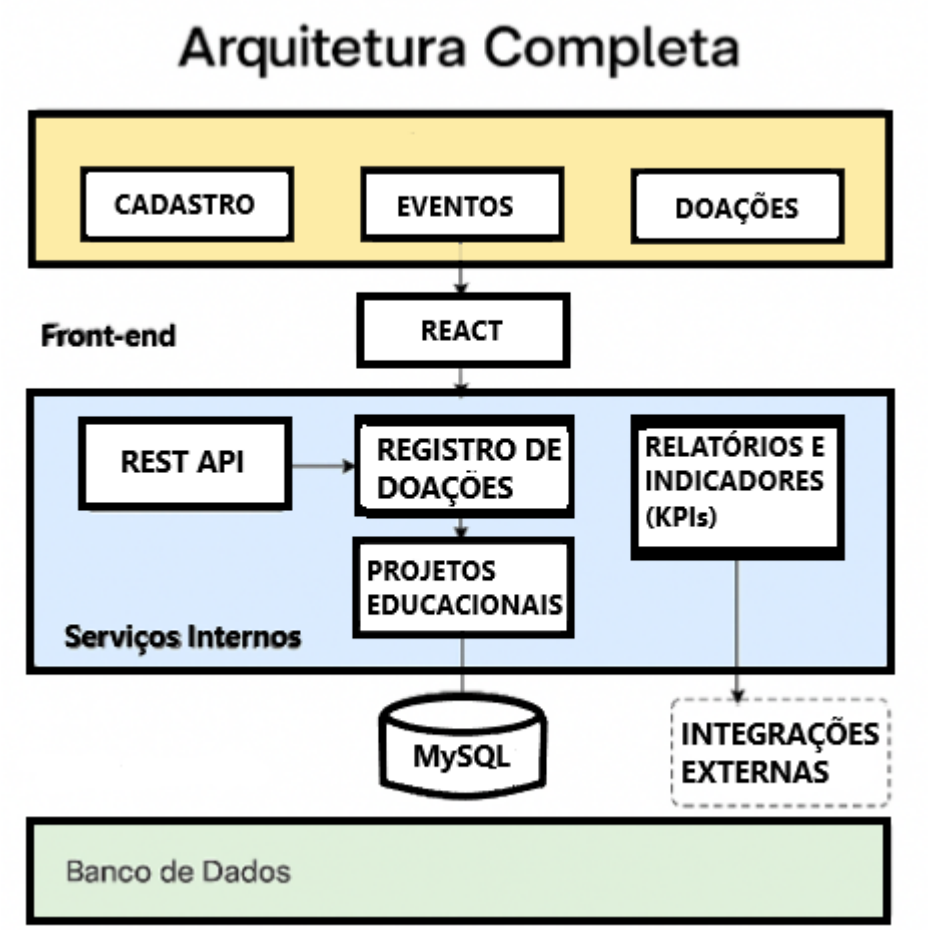

Figura 2 – Visão Funcional da Arquitetura do Sistema

4. Tecnologias Utilizadas

Camada de Apresentação: React, JavaScript, CSS, Vite.js

Camada de Lógica de Aplicação: JavaScript (serviços e validações)

Camada de Persistência (API): A definir com o grupo

5. Comunicação entre Camadas – Front-end e Back-end

A comunicação entre o front-end (React com Vite) e o back-end (Node.js com Express) ocorre por meio de requisições HTTP, utilizando a biblioteca Axios. No back-end, as rotas são organizadas com Express.js e configuradas para tratar requisições RESTful, fazendo uso do middleware CORS para permitir acesso de origens diferentes.

As requisições feitas no front-end usam Axios para interagir com as rotas criadas no back-end, permitindo operações como login, cadastro de usuários, visualização e inscrição em

eventos, entre outras funcionalidades. Essas rotas interagem com o banco de dados MySQL por meio do Sequelize, que facilita a manipulação dos dados.

Fluxo geral:
- Front-end React envia requisições com Axios
- Back-end Express recebe, processa e valida os dados
- Sequelize executa comandos SQL no MySQL
- Resposta é devolvida ao front-end

Essa arquitetura garante modularidade, segurança e facilidade de manutenção.

5. Conclusão

A arquitetura proposta atende aos princípios de modularidade, escalabilidade e manutenibilidade. Com a separação clara entre as camadas, a aplicação pode ser facilmente estendida e adaptada a novas necessidades, além de facilitar o trabalho em equipe e o desenvolvimento paralelo entre front-end e back-end.