Documentação Técnica - Sistema de Gerenciamento de Quadras de Tênis

Sumário

Introdução

Este documento descreve a implementação técnica do Sistema de Gerenciamento de Quadras de Tênis, um CRUD (Create, Read, Update, Delete) completo que permite cadastrar, listar, editar e excluir quadras de tênis, além de gerenciar sua disponibilidade.

O sistema foi desenvolvido como parte de um desafio técnico para demonstrar habilidades em desenvolvimento full-stack, incluindo modelagem de banco de dados, criação de API RESTful e implementação de frontend.

Arquitetura do Sistema

O sistema segue uma arquitetura cliente-servidor com três camadas principais:

1. **Banco de Dados**: PostgreSQL para armazenamento persistente dos dados.
2. **Backend**: API RESTful desenvolvida com Node.js, Express e Sequelize ORM.
3. **Frontend**: Interface de usuário desenvolvida com React e Bootstrap.

A comunicação entre o frontend e o backend é realizada através de requisições HTTP, seguindo os princípios REST.

Banco de Dados

Modelo de Dados

O banco de dados consiste em uma única tabela `courts` com a seguinte estrutura:

```
CREATE TABLE courts (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    location VARCHAR(255) NOT NULL,
    available BOOLEAN DEFAULT true,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT
CURRENT_TIMESTAMP,
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT
CURRENT_TIMESTAMP
);
```

Campos

- **id**: Identificador único da quadra (chave primária, auto-incremento)
- **name**: Nome da quadra
- **location**: Endereço/localização da quadra
- **available**: Indicador de disponibilidade da quadra (booleano)
- **created_at**: Data e hora de criação do registro
- **updated_at**: Data e hora da última atualização do registro

Trigger para Atualização Automática

Um trigger foi implementado para atualizar automaticamente o campo `updated_at` sempre que um registro é modificado:

```
CREATE OR REPLACE FUNCTION update_updated_at_column()
RETURNS TRIGGER AS $$
BEGIN
   NEW.updated_at = NOW();
   RETURN NEW;
END;
$$ language 'plpgsql';

CREATE TRIGGER update_courts_updated_at
BEFORE UPDATE ON courts
FOR EACH ROW
EXECUTE FUNCTION update_updated_at_column();
```

Backend (API RESTful)

Tecnologias Utilizadas

- **Node.js**: Ambiente de execução JavaScript
- **Express**: Framework web para Node.js
- **Sequelize**: ORM (Object-Relational Mapping) para interação com o banco de dados
- **PostgreSQL**: Sistema de gerenciamento de banco de dados relacional
- **Swagger**: Documentação da API
- **Jest**: Framework de testes

Estrutura de Diretórios

```
backend/
├── src/
│   ├── config/
│   │   ├── database.js
│   │   └── swagger.js
│   ├── controllers/
│   │   └── courtController.js
│   ├── middlewares/
│   │   └── validation.js
│   ├── models/
│   │   └── Court.js
│   ├── routes/
│   │   └── courtRoutes.js
│   └── app.js
├── tests/
│   └── court.test.js
├── database.sql
└── server.js
```

Endpoints da API

Método	Endpoint	Descrição
POST	/api/courts	Criar uma nova quadra
GET	/api/courts	Listar todas as quadras (com filtro opcional de disponibilidade)
GET	/api/courts/:id	Buscar uma quadra por ID
PUT	/api/courts/:id	Editar uma quadra

Método	Endpoint	Descrição
PATCH	/api/courts/:id/availability	Alterar a disponibilidade de uma quadra
DELETE	/api/courts/:id	Excluir uma quadra

Validação e Tratamento de Erros

Foram implementados middlewares para validação de dados e tratamento de erros:

- **validateCourtData**: Valida os campos obrigatórios (name, location) ao criar ou atualizar uma quadra.
- **validateAvailability**: Valida o campo available ao atualizar a disponibilidade de uma quadra.
- **errorHandler**: Middleware global para tratamento de erros, garantindo respostas padronizadas.

Frontend

Tecnologias Utilizadas

- **React**: Biblioteca JavaScript para construção de interfaces
- **React Router**: Roteamento no lado do cliente
- **Axios**: Cliente HTTP para comunicação com a API
- **Bootstrap**: Framework CSS para estilização
- **React Testing Library**: Biblioteca para testes de componentes React

Estrutura de Diretórios

```
frontend/
├── public/
├── src/
│   ├── components/
│   │   ├── CourtForm.js
│   │   ├── CourtList.js
│   │   └── Header.js
│   ├── services/
│   │   └── courtService.js
│   ├── tests/
│   │   └── CourtList.test.js
│   ├── App.js
```

```
│   └── index.js
└── package.json
```

Componentes Principais

- **CourtList**: Exibe a lista de quadras com opções para filtrar, editar, excluir e alterar disponibilidade.
- **CourtForm**: Formulário para criar e editar quadras.
- **Header**: Barra de navegação superior.

Serviços

O arquivo `courtService.js` encapsula todas as chamadas à API, facilitando a manutenção e reutilização do código.

Recursos Adicionais

Documentação Swagger

A API é documentada usando Swagger, acessível através do endpoint `/api-docs`. A documentação inclui:

- Descrição de todos os endpoints
- Parâmetros esperados
- Exemplos de requisições e respostas
- Códigos de status HTTP

Middlewares Avançados

Foram implementados middlewares para:

- Validação de dados de entrada
- Tratamento centralizado de erros
- Formatação consistente de respostas

Testes

Testes de Backend

Os testes do backend utilizam Jest e Supertest para testar os endpoints da API. Eles verificam:

- Listagem de quadras (com e sem filtro)
- Busca de quadra por ID
- Criação de quadra
- Atualização de quadra
- Atualização de disponibilidade
- Exclusão de quadra
- Tratamento de erros

Testes de Frontend

Os testes do frontend utilizam React Testing Library para testar os componentes React. Eles verificam:

- Renderização da lista de quadras
- Filtragem de quadras disponíveis
- Alternância de disponibilidade
- Exclusão de quadra

Instruções de Instalação e Execução

Pré-requisitos

- Node.js (v14 ou superior)
- npm ou yarn
- PostgreSQL

Configuração do Banco de Dados

1. Crie um banco de dados PostgreSQL: `sql CREATE DATABASE tennis_courts;`
2. Execute o script SQL para criar a tabela e o trigger: `bash psql -d tennis_courts -f database.sql`

Instalação e Execução do Backend

1. Navegue até o diretório do backend: `bash cd backend`
2. Instale as dependências: `bash npm install`
3. Configure a conexão com o banco de dados em `src/config/database.js`.
4. Inicie o servidor: `bash npm start`

O servidor estará disponível em `http://localhost:3001`.

Instalação e Execução do Frontend

1. Navegue até o diretório do frontend: `bash cd frontend`
2. Instale as dependências: `bash npm install`
3. Inicie o servidor de desenvolvimento: `bash npm start`

A aplicação estará disponível em `http://localhost:3000`.

Considerações Finais

Este projeto demonstra a implementação de um CRUD completo seguindo boas práticas de desenvolvimento, incluindo:

- Código limpo e organizado
- Arquitetura em camadas
- Validação de dados
- Tratamento de erros
- Documentação da API
- Testes automatizados

O sistema pode ser expandido no futuro com recursos adicionais como autenticação JWT, agendamento de horários para as quadras, integração com sistemas de pagamento, entre outros.