DOCUMENTAÇÃO TÉCNICA DO PROJETO – API BIBLIOTECA
Título do Projeto
Sistema de Gerenciamento de Times

Grupo
Gabriel Favalessa Ferreira, Nicolas Toaldo Golemba, Matheus Belniak Mendes

1. Objetivo do Projeto

O objetivo deste projeto é desenvolver uma API RESTful para gerenciamento de times de futebol, aplicando conceitos de modelagem de dados, rotas HTTP e integração com banco de dados SQLite via Entity Framework Core. A API permite operações básicas de listagem, consulta por ID, cadastro e exclusão tanto de times quanto de jogadores, além da busca de jogadores pelo nome.

A aplicação visa facilitar o controle e consulta dos times da Série A do Campeonato Brasileiro, incluindo a gestão de jogadores com suas estatísticas (número de jogos e títulos). Este sistema pode ser utilizado para fins acadêmicos e como base para aplicações esportivas.

2. Passo a Passo Lógico para a Resolução

Definição das Entidades:
Foram definidas duas classes principais — Time e Jogador.

Time representa o clube de futebol, com propriedades como nome, técnico, ano de fundação e categoria.

Jogador representa um atleta, contendo nome, número, posição, estatísticas (jogos e títulos) e a referência ao time.

Configuração do Banco de Dados:
Utilizamos o Entity Framework Core para configurar o contexto AppDbContext que gerencia as entidades e persiste os dados em um banco SQLite (times.db).

População Inicial do Banco:
Criamos um inicializador (DbInitializer) que adiciona ao banco 8 times da Série A, cada um com 22 jogadores fictícios e suas estatísticas simuladas para testes.

Desenvolvimento da API:
Foram criados dois controladores principais:

TimesController com rotas para listar, buscar por ID, adicionar e deletar times.

JogadoresController com rotas similares para jogadores, além de uma rota extra para busca de jogadores por nome, retornando todos os que correspondem à pesquisa.

Separação das Responsabilidades:
O projeto está organizado com separação clara entre modelos, dados e controladores, facilitando manutenção e futuras expansões.

Otimização de Consulta:
O uso de .Include() no Entity Framework garante que os jogadores associados a cada time sejam carregados junto, evitando consultas múltiplas e garantindo melhor desempenho.

3. Estruturas e Comandos Utilizados

Modelagem com Classes C#:
Definimos as entidades Time e Jogador com atributos e relacionamentos (one-to-many).

Entity Framework Core:
Configurado via AppDbContext para SQLite. Comandos como context.Times.Add(), context.SaveChangesAsync(), e context.Times.Include() são usados para manipulação dos dados.

API REST com ASP.NET Core:

[ApiController] e [Route] para definição das rotas.

Métodos HTTP: GET, POST, DELETE.

Async/Await para operações assíncronas e melhor resposta.

Tratamento de respostas com NotFound(), CreatedAtAction(), e NoContent() para status HTTP corretos.

Rotas específicas:

GET /api/times lista todos os times com jogadores.

GET /api/times/{id} busca time por ID.

POST /api/times adiciona novo time.

DELETE /api/times/{id} exclui time.

GET /api/jogadores lista jogadores.

GET /api/jogadores/{id} busca jogador por ID.

GET /api/jogadores/por-nome/{nome} busca jogadores pelo nome (contendo string).

POST /api/jogadores adiciona jogador.

DELETE /api/jogadores/{id} exclui jogador.

4. Justificativas e Métodos Adotados
Modelagem em duas entidades:
O relacionamento entre time e jogador é natural e permite consultas eficientes e relacionamentos via EF Core.

Banco SQLite:
Escolha prática para um projeto acadêmico simples, leve e fácil de configurar sem necessidade de servidor.

Entity Framework Core:
Facilita o trabalho com banco de dados via LINQ, abstrai comandos SQL e mantém código limpo e legível.

Async/Await:
Melhora a performance da API ao evitar bloqueio de threads durante operações de banco, tornando a aplicação responsiva.

Inclusão de dados inicializados:
Facilita a demonstração das funcionalidades e testes, garantindo que a API tenha dados reais para operações.

Estrutura modular:
Separação clara entre models, dados e controladores, além de rotas dedicadas para cada entidade, promove organização e escalabilidade.

Busca por nome com filtro:
Permite a funcionalidade extra requisitada de localizar jogadores por nome com busca parcial, aumentando a usabilidade da API.

Uso de Swagger:
Para documentação automática da API e facilitar testes pelo navegador durante desenvolvimento.

5. Conclusão
O projeto implementa uma API RESTful robusta e organizada para gerenciamento de times e jogadores, contemplando requisitos fundamentais como CRUD, relacionamento entre entidades e busca específica. O uso de boas práticas, padrões do ASP.NET Core e EF Core, além da inicialização automatizada de dados, garante funcionalidade, manutenção facilitada e desempenho adequado para a proposta.