1. Introdução

Este documento fornece uma visão geral do sistema de gerenciamento de torneios. O objetivo do sistema é simular e gerenciar torneios de mata-mata, facilitando a administração de competições onde equipes competem até a determinação de um vencedor.

Objetivo: O sistema oferece uma solução web eficiente para a administração de torneios, permitindo a gestão de equipes e partidas, bem como a determinação do vencedor final. O sistema visa ser intuitivo e robusto, atendendo às necessidades de gerenciamento de torneios de forma clara e organizada.

---

2. Tecnologias Utilizadas

- **ASP.NET Core 8.0: Framework para desenvolvimento web, escolhido pela sua robustez e escalabilidade.**
- **Entity Framework Core 8.0: ORM para interação com o banco de dados, facilitando a manipulação de dados com um modelo orientado a entidades.**
- **MySQL: Banco de dados relacional para armazenamento de dados, escolhido por sua confiabilidade e suporte amplo.**
- **AutoMapper: Biblioteca para mapeamento automático entre objetos, essencial para a conversão eficiente de dados entre camadas.**
- **Swashbuckle.AspNetCore: Gera documentação Swagger para APIs RESTful, permitindo a visualização e teste dos endpoints de forma interativa.**
- **Pomelo.EntityFrameworkCore.MySql: Provedor que integra o Entity Framework Core com o MySQL, garantindo compatibilidade e desempenho.**
- **Moq: Framework para criação de mocks, facilitando a realização de testes unitários.**
- **xUnit: Framework de testes utilizado para criar e executar testes automatizados.**
- **Microsoft.EntityFrameworkCore.InMemory: Provedor de banco de dados em memória usado para testes e desenvolvimento, permitindo testes rápidos e isolados.**
- **Microsoft.EntityFrameworkCore.Tools: Ferramentas para migrações e scaffolding, facilitando o desenvolvimento e manutenção do esquema do banco de dados.**
- **HTML, CSS e JS: Tecnologias de front-end usadas para criar a interface do usuário, garantindo uma experiência visual e interativa.**

---

3. Requisitos Não Funcionais

- **Segurança: O sistema protege utilizando do sistema de CORS**
- **Usabilidade: A interface deve ser intuitiva, garantindo que os usuários consigam realizar operações sem dificuldades.**
- **Escalabilidade: A arquitetura deve suportar a adição de novos recursos e usuários sem comprometer a performance.**

- **Performance: O sistema deve responder rapidamente mesmo com grandes volumes de dados.**

---

4. Visão Geral da Arquitetura

- **Descrição dos Componentes: O sistema é composto por um front-end em HTML, CSS e JS, que se comunica com um back-end desenvolvido em ASP.NET Core 8.0. O back-end utiliza Entity Framework Core para interagir com o banco de dados MySQL.**

---

5. Detalhes de Implementação

- **Configurações do Ambiente: O projeto requer variáveis de ambiente para conexão com o banco de dados e configuração de serviços externos. Exemplos incluem:**

DB_CONNECTION_STRING=Server=localhost;Database=tournament_db;User=root;Password=yourpassword;

- **Execução do Projeto:**
    - **Para iniciar o projeto localmente, execute dotnet run na raiz do projeto após instalar as dependências listadas no arquivo README.md.**
    - **Executar Migrations: Para aplicar as migrations e atualizar o banco de dados, use os seguintes comandos:**

dotnet ef migrations add InitialCreate

dotnet ef database update

Estes comandos criarão a estrutura inicial do banco de dados e aplicarão as migrações necessárias.

---

6. Endpoints

- **MatchController**
    - **Criar uma Partida**
        - **Endpoint: http://localhost:5000/api/match**
        - **Descrição: Cria uma partida com os dados fornecidos.**
        - **Exemplo de Requisição JSON:**

```
{
    "teamAId": 1,
    "teamBId": 2,
    "tournamentId": 1
}
```

- **Obter uma Partida por ID**
  - **Endpoint: GET http://localhost:5000/api/match/{id}**
  - **Descrição: Retorna detalhes de uma partida específica.**
- **Obter Todas as Partidas**
  - **Endpoint: GET http://localhost:5000/api/match**
  - **Descrição: Retorna uma lista de todas as partidas.**
- **Atualizar uma Partida por ID**
  - **Endpoint: PUT http://localhost:5000/api/match/1**
  - **Descrição: Atualiza uma partida existente com dados fornecidos.**
- **Excluir uma Partida por ID**
  - **Endpoint: DELETE http://localhost:5000/api/match/1**
  - **Descrição: Remove uma partida específica.**

- **TeamController**
  - **Criar um Novo Time**
    - **Endpoint: POST /api/team**
    - **Descrição: Cria um time com os dados fornecidos.**
    - **Exemplo de Requisição JSON:**

```
{
    "name": "Noxus",
    "region": "Runeterra Central"
}
```

  - **Obter um Time por ID**
    - **Endpoint: GET http://localhost:5000/api/team/{id}**
    - **Descrição: Retorna detalhes de um time específico.**
  - **Obter Todos os Times**
    - **Endpoint: GET http://localhost:5000/api/team**
    - **Descrição: Retorna uma lista de todos os times.**
  - **Atualizar um Time Existente**
    - **Endpoint: PUT http://localhost:5000/api/team/{id}**
    - **Descrição: Atualiza um time existente com dados fornecidos.**

        - **Exemplo de Requisição JSON:**

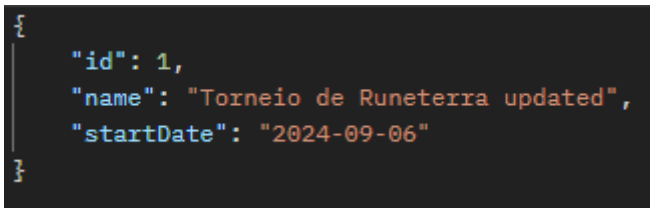
```
{
    "id": 1,
    "name": "Noxus updated",
    "region": "Runeterra Central"
}
```

    - **Excluir um Time por ID**
        - **Endpoint: DELETE http://localhost:5000/api/team/{id}**
        - **Descrição: Remove um time específico.**

- **TournamentController**
    - **Criar um Novo Torneio**
        - **Endpoint: POST http://localhost:5000/api/tournament**
        - **Descrição: Cria um torneio e retorna o ID do torneio criado.**
        - **Exemplo de Requisição JSON:**

```
{
    "name": "Torneio de Runeterra",
    "startDate": "2024-09-06"
}
```

    - **Obter um Torneio por ID**
        - **Endpoint: GET http://localhost:5000/api/tournament{id}**
        - **Descrição: Retorna detalhes de um torneio específico.**
    - **Obter um vencedor do torneio por ID**
        - **Endpoint: GET http://localhost:5000/api/tournament/{id}/winner**
        - **Descrição: Retorna o id do vencedor do torneio específico.**
    - **Obter Todos os Torneios**
        - **Endpoint: GET http://localhost:5000/api/tournament**
        - **Descrição: Retorna uma lista de todos os torneios.**
    - **Atualizar um Torneio Existente**
        - **Endpoint: PUT http://localhost:5000/api/tournament/{id}**
        - **Descrição: Atualiza um torneio existente com dados fornecidos.**
        - **Exemplo de Requisição JSON:**



- **Excluir um Torneio por ID**
  - **Endpoint: DELETE http://localhost:5000/api/tournament/1**
  - **Descrição: Remove um torneio específico.**

---

7. Acesso à Documentação da API

Para acessar a documentação interativa da API utilizando Swagger UI:

1. **Certifique-se de que o servidor backend está em execução.**
   - **Inicie o projeto com o comando:**

dotnet run

2. **Abra seu navegador e vá para:**

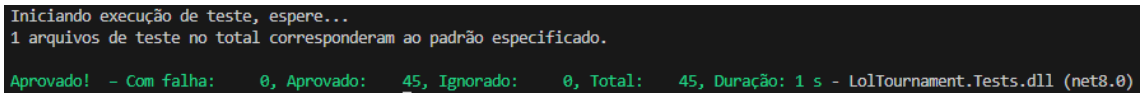
http://localhost:5000

O Swagger UI será exibido automaticamente e permitirá explorar e testar os endpoints da API.

---

8. Testes

Os testes são essenciais para garantir a qualidade e confiabilidade do sistema. Foram implementados testes unitários e de integração para validar o funcionamento correto das funcionalidades e componentes do sistema.



Cobertura dos Testes:

- **Validação de Regras de Negócio: Verifica a aplicação correta das regras de negócio.**
- **Operações CRUD: Confirma que operações de criação, leitura, atualização e exclusão funcionam como esperado.**
- **Tratamento de Exceções: Assegura que exceções são tratadas adequadamente e mensagens de erro corretas são retornadas.**

Manter uma suíte de testes robusta é crucial para identificar e corrigir problemas rapidamente, além de garantir que mudanças no código não introduzam novos erros.

9. Execução do Front-End

- **Configuração do Ambiente: Para rodar o front-end e consumir a API, você pode usar a extensão Live Server no Visual Studio Code (VSCode) para criar um servidor de desenvolvimento local.**
- **Passos para executar o Front-End:**
    1. **Instale a extensão Live Server no VSCode: Vá até a aba de extensões e busque por "Live Server". Instale a extensão desenvolvida por Ritwick Dey.**
    2. **Abra o Projeto Front-End: Navegue até a pasta onde está localizado o código do front-end.**
    3. **Inicie o Live Server: Clique com o botão direito no arquivo index.html (ou no arquivo principal do front-end) e selecione "Open with Live Server". Isso abrirá o front-end em um servidor local e permitirá que ele consuma a API exposta pelo back-end.**