UNIVERSIDADE FEDERAL DE MINAS GERAIS
Instituto de Ciências Exatas – ICEx
Departamento de Ciência da Computação

Relatório Tecnológico Final da disciplina de Projeto Orientado em Computação II:

Plataforma de Gerenciamento de Torneios para Jogos Online

Michel Barros da Fonseca
Orientador: Pedro Olmo Stancioli Vaz De Melo

Belo Horizonte

2025

Sumário

1 Introdução

A cada dia, os jogos eletrônicos se tornam uma parte cada vez mais comum do tempo livre de muitas pessoas. A Internet desempenha um papel crucial ao facilitar a comunicação, conectando as pessoas por meio dos jogos multijogador e suas comunidades.

A competitividade é uma característica inerente à natureza humana, manifestando-se tanto em aspectos pessoais quanto em esportes tradicionais como futebol, vôlei e basquete. Essa mesma competitividade se reflete nos jogos eletrônicos e nos eSports, onde os jogadores competem em partidas amistosas ou em torneios online.

No entanto, a fragmentação de tais comunidades de jogos torna difícil a descoberta de novos torneios e a interação entre os membros da comunidade. Geralmente, os torneios organizados por essas comunidades carecem de uma estrutura adequada, limitando-se apenas ao essencial para o torneio aconteça de fato.

Dado isso, o tema desse relatório detalha uma possível solução: a criação de uma plataforma de gerenciamento de torneios para jogos online. Essa plataforma possui o objetivo de aproximar as comunidades de jogos eletrônicos e promover os torneios organizados por elas. A plataforma permite que os usuários criem, gerenciem, participem, customizem, revisitem e recordem torneios, facilitando a descoberta de novas comunidades e jogos, além de proporcionar momentos emocionantes a serem compartilhados.

2 O Sistema

O sistema é dividido em duas partes:

- O servidor: responsável por ser uma interface para realizar as operações lógicas do sistema como criação dos torneios, rodadas e times, acompanhar o andamento dos torneios e de suas partidas.
- O cliente: responsável pela interação do usuário final com o sistema, mostrando os dados e as operações no sistema de uma maneira fácil e amigável.

O foco desse relatório vai ser na integração entre o servidor e o cliente. O cliente desenvolvido foi uma aplicação web, disponível em https://th.nkrw.dev/. Ao acessar, o navegador do usuário faz download dos arquivos *JavaScript* necessários para a execução do cliente, que contém toda a lógica de exibição de conteúdo e operações.

2.1 Protocolo HTTP

Um dos protocolos de rede mais comuns hoje é o protocolo HTTP. Por meio desse protocolo é possível fazer requisições para um servidor na internet. Uma requisição HTTP parece algo como *GET http://catfact.ninja/fact*. Esse comando faz uma requisição no link *http://catfact.ninja/fact* com o método *GET* e retorna um JSON contendo algum fato sobre gatos. Hoje em dia, basicamente qualquer eletrônico conectado à internet possui suporte a HTTP.

A plataforma usa extensivamente as requisições HTTP para sincronizar e atualizar dados entre o cliente e o servidor. Algumas das requisições feitas do cliente ao servidor são:

- POST https://th.nkrw.dev/users/login: realiza o login de um usuário com o seu nome de usuário e senha e retorna um token de autenticação.
- POST https://th.nkrw.dev/tournaments: cria um torneio a partir dos dados passados no corpo da requisição.
- GET https://th.nkrw.dev/matches/{match_id}: retorna os dados de alguma partida com identificador *match_id*.
- GET https://th.nkrw.dev/rounds/{round_id}: retorna os dados de alguma rodada com identificador *round_id*

Cada uma dessas operações no contexto do servidor são chamados de rotas ou endpoints e possuem o propósito de implementar alguma operação específica.

Outro protocolo utilizado é o HTTPS, que é responsável por criptografar sua requisição HTTP de forma a transferir seus dados de uma maneira segura de uma ponta a outra sem a possibilidade de algum agente no meio do caminho interceptar e ler seus dados. Isso é essencial ao executar essa aplicação na Internet, já que dados sensíveis como senhas e tokens de autenticação são transmitidos entre o cliente e o servidor.

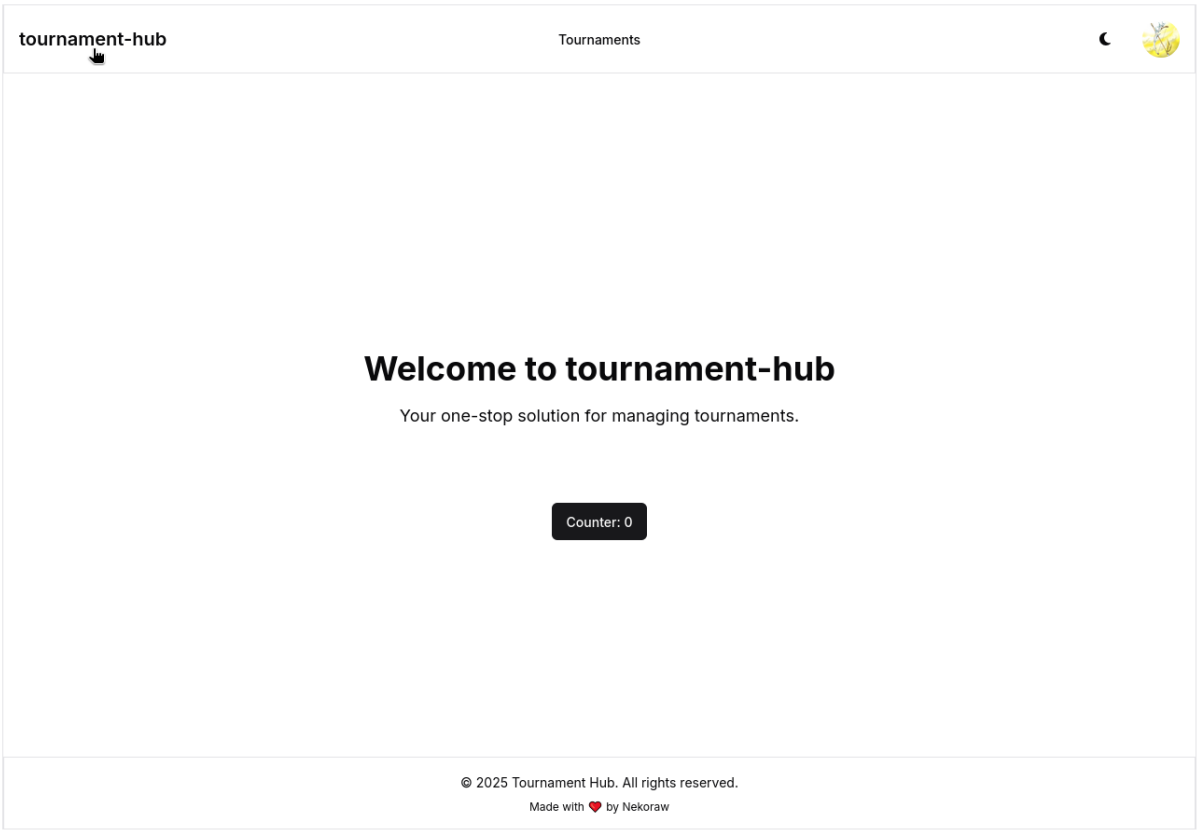

Figura 1 – A página inicial do cliente

3 Modelagem do Servidor

3.1 Entidades

O servidor possui vários membros que iremos chamar de entidades. Uma entidade é um modelo que representa um objeto bem definido usado para armazenar dados de um objeto ou conceito do mundo real.

Um exemplo simples de entidade é o *User*:

- User
  - id: UUID
  - name: string
  - email: string
  - password_hash: string

Esses tipos de entidades são usadas em maior parte para definir um modelo de armazenamento dos dados no banco de dados usado, que é o MongoDB (MongoDB, Inc. (2025)). Elas não são suficientes ou adequadas para serem usadas como modelo de transmissão dos dados entre servidor e cliente. O que foi usado nesse caso, são os *Data Transfer Objects* (DTOs). Eles são uma modelagem mais simplista dos dados e são usados exclusivamente para a transferência de dados entre cliente e servidor. Veja:

- LoginUserDTO
  - username: string = Field(max_length=16, min_length=3)
  - password: string = Field(min_length=1)

Esse é um DTO usado para efetuar o log-in na plataforma. Não é necessário que o usuário forneça o e-mail para identificá-lo, somente o seu nome de usuário e a sua senha. Além disso, o FastAPI (Ramírez (2025)), o framework utilizado no servidor para conexão do servidor com a Internet, consegue identificar automaticamente propriedades do dado esperado como comprimento do username e rejeitar automaticamente quaisquer requisições que não obedeçam a essas regras.

É inseguro transmitir suas senhas diretamente por meio desse modelo em uma conexão HTTP pura, mas é seguro transmití-la por meio de HTTPS + TLS, já que a conexão é criptografada e é impossível de algum ator com más intenções consiga descobrir sua senha dessa forma. O website em que o cliente está hospedado usa HTTPS.

3.2 Regras especificas dos jogos

3.2.1 Genshin Impact

Genshin Impact é um jogo RPG de mundo aberto onde os jogadores podem explorar o mundo e derrotar monstros com times de até quatro personagens (`GenshinCharacter`).

Esses personagens podem ser personagens 4 estrelas ou personagens 5 estrelas, onde a quantidade de estrelas simboliza a raridade do personagem (`GenshinRarity`).

Uma das partes do equipamento dos personagens é a sua arma, como lança, espada, espadão, arco e catalisador. As armas podem ser de 1 estrela até 5 estrelas de raridade.

Além disso esses personagens podem usar ataques de algum elemento específico como Pyro, Hydro, Electro, Anemo, Dendro, Geo e Cryo (`GenshinElement`).

Os jogadores competem de modo assíncrono na última dificuldade do Abismo do Espiral. No último piso do Abismo do Espiral, os jogadores precisam de dois times completos e são desafiados a derrotar monstros em três batalhas consecutivas.

O jogador vencedor é o jogador que consegue derrotar todos os monstros na menor quantidade de tempo. O resultado dessa partida, como o tempo restante de cada jogador, personagens selecionados para cada time e os banimentos de personagens são salvos na entidade `GenshinMatchData`.

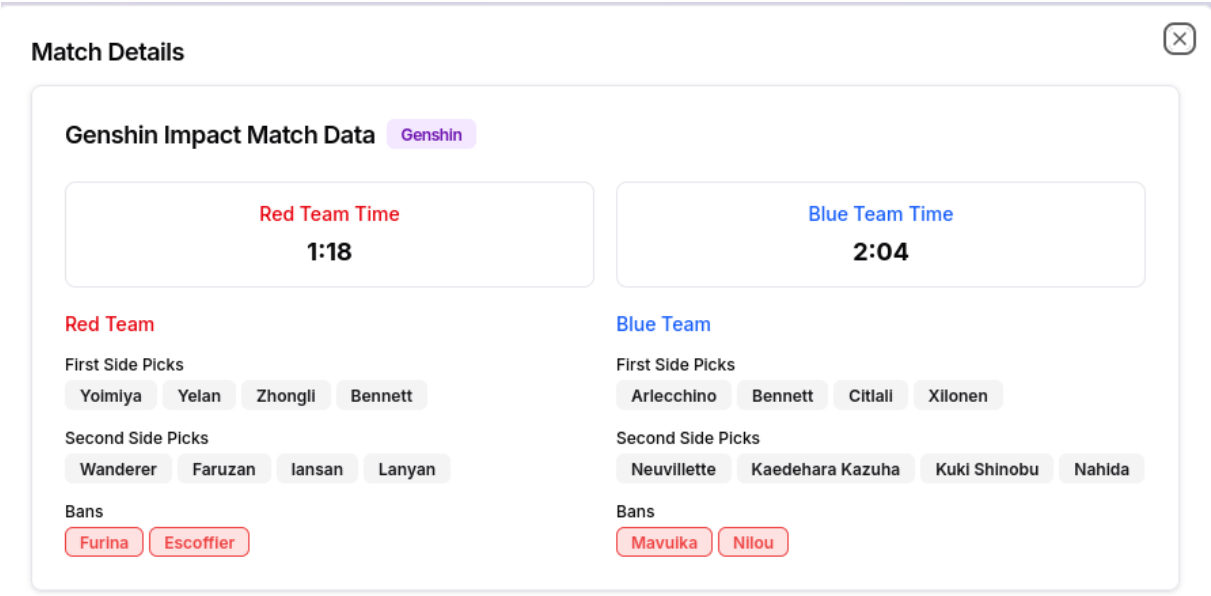

Figura 2 – Dados de uma partida de Genshin Impact

Existem algumas regras mais específicas que podem ser customizadas, como a quantidade de personagens e armas 5 estrelas, além da quantidade de tentativas para o jogador tentar obter o melhor tempo (`GenshinTournamentRules`).

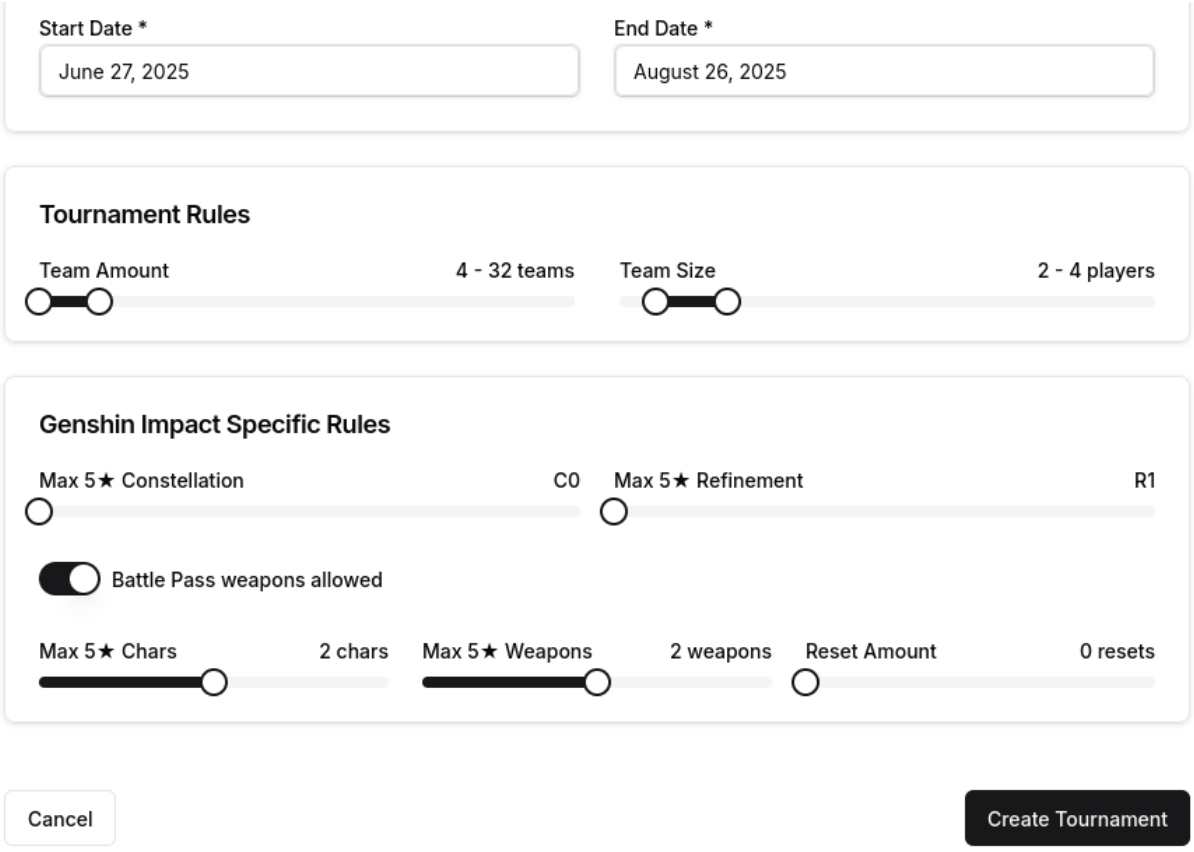

Figura 3 – Regras personalizáveis para o Genshin Impact

3.2.2 osu!

osu! é um jogo de ritmo onde o jogador clica em círculos no ritmo da música. *Beatmap* é uma coleção de círculos, sliders e spinners que representam uma música.

Por exemplo, a música *My Love* de *Kuba Oms* possui três dificuldades (Oms (2014)): Fácil, Normal e Insano. Cada uma dessas dificuldades é um ‘Beatmap‘.

Uma partida de osu! acontece em uma sala multiplayer em tempo real, onde os jogadores podem escolher ou banir beatmaps de um conjunto de beatmaps definidos para aquela rodada (`OsuRoundRules`), chamado de `Mappool`. Se um jogador demorar muito para se juntar à partida (regra do torneio), pode receber alguma penalidade.

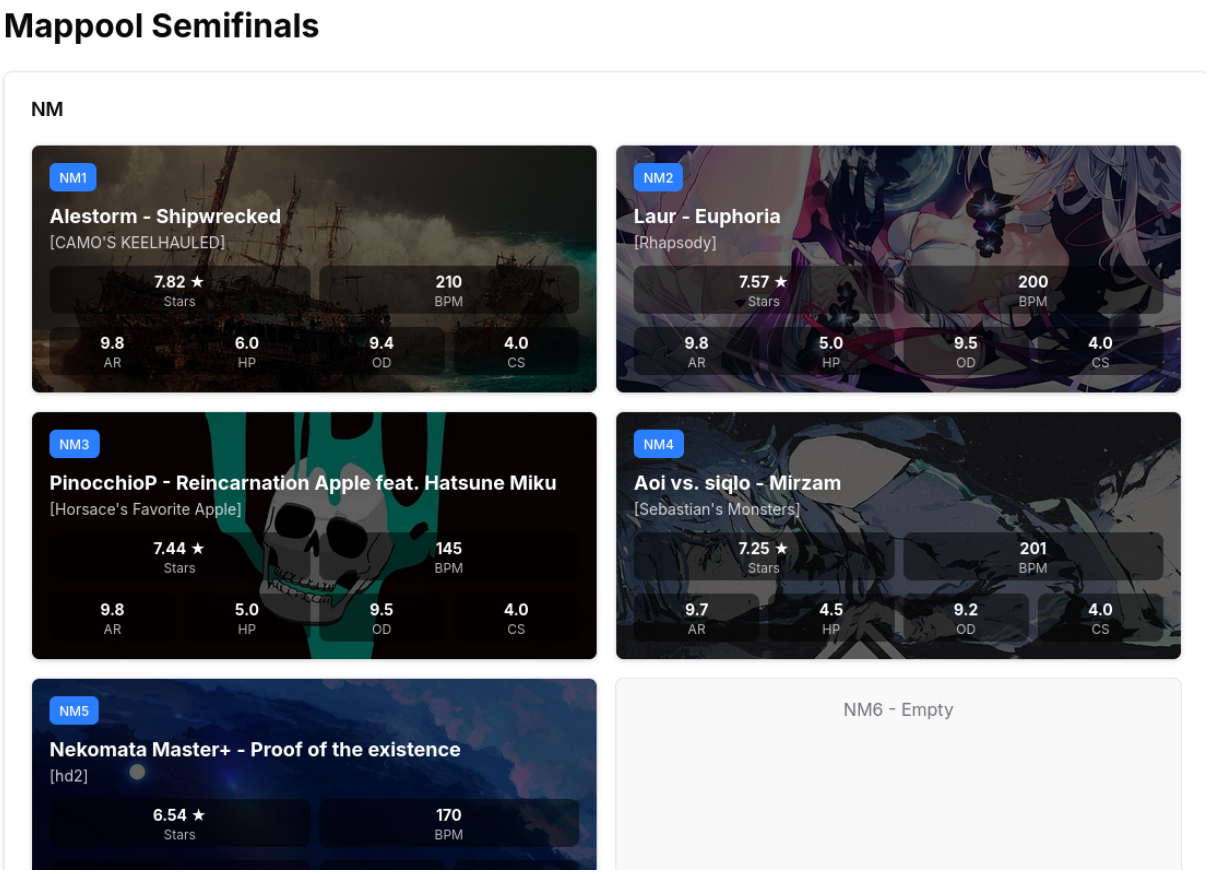

Figura 4 – Conjunto de Beatmaps de exemplo

Cada jogador tem uma certa quantidade de tempo para escolher ou banir um beatmap (regra do torneio), e quem fazer a maior pontuação no beatmap ganha um ponto. Quem fizer mais pontos durante a partida até o limite máximo (melhor de 5, ou melhor de 7 por exemplo) ganha a partida. Dados como os mapas selecionados e banidos são salvos na entidade `OsuMatchData` e as regras são definidas na entidade `OsuTournamentRules`.

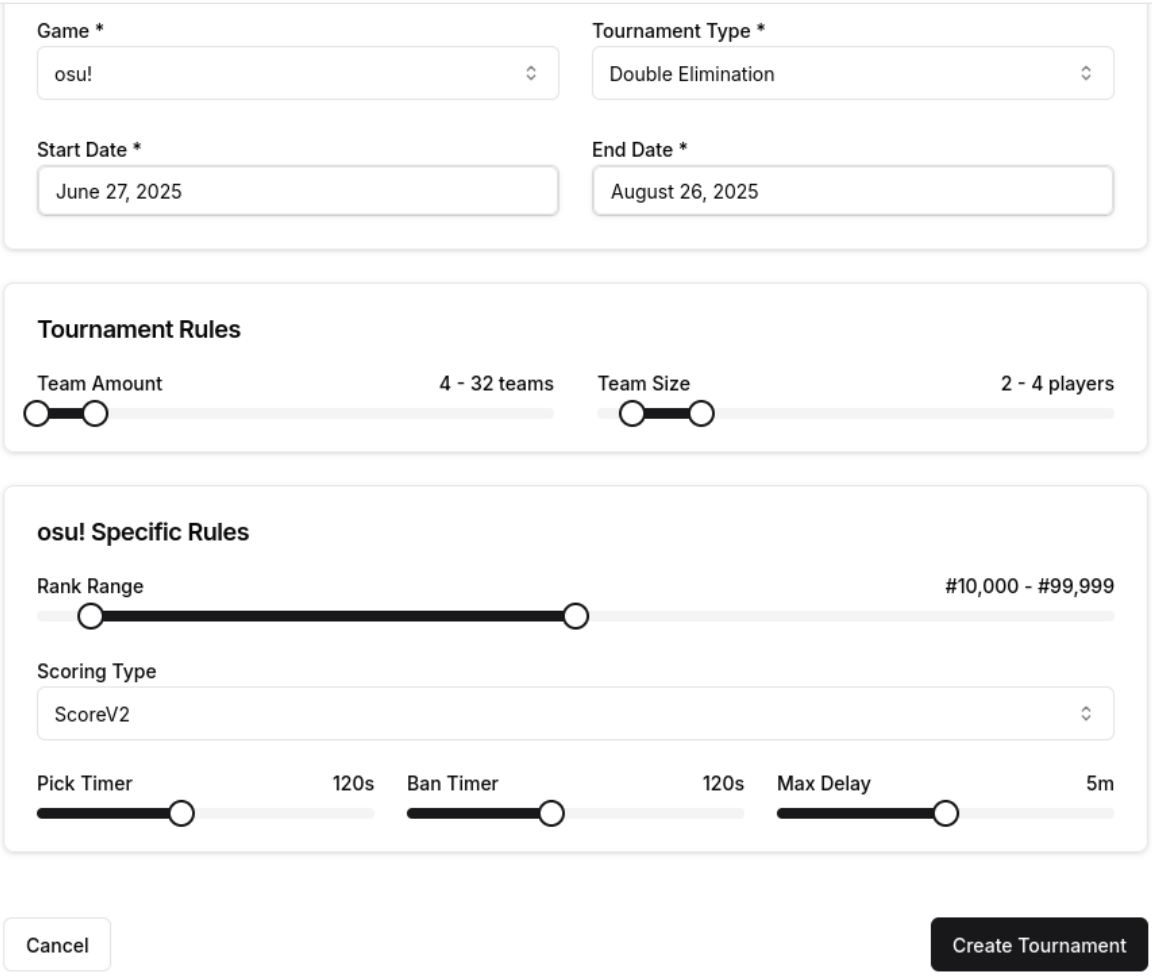

Figura 5 – Regras para o osu!

3.3 Modelo Completo

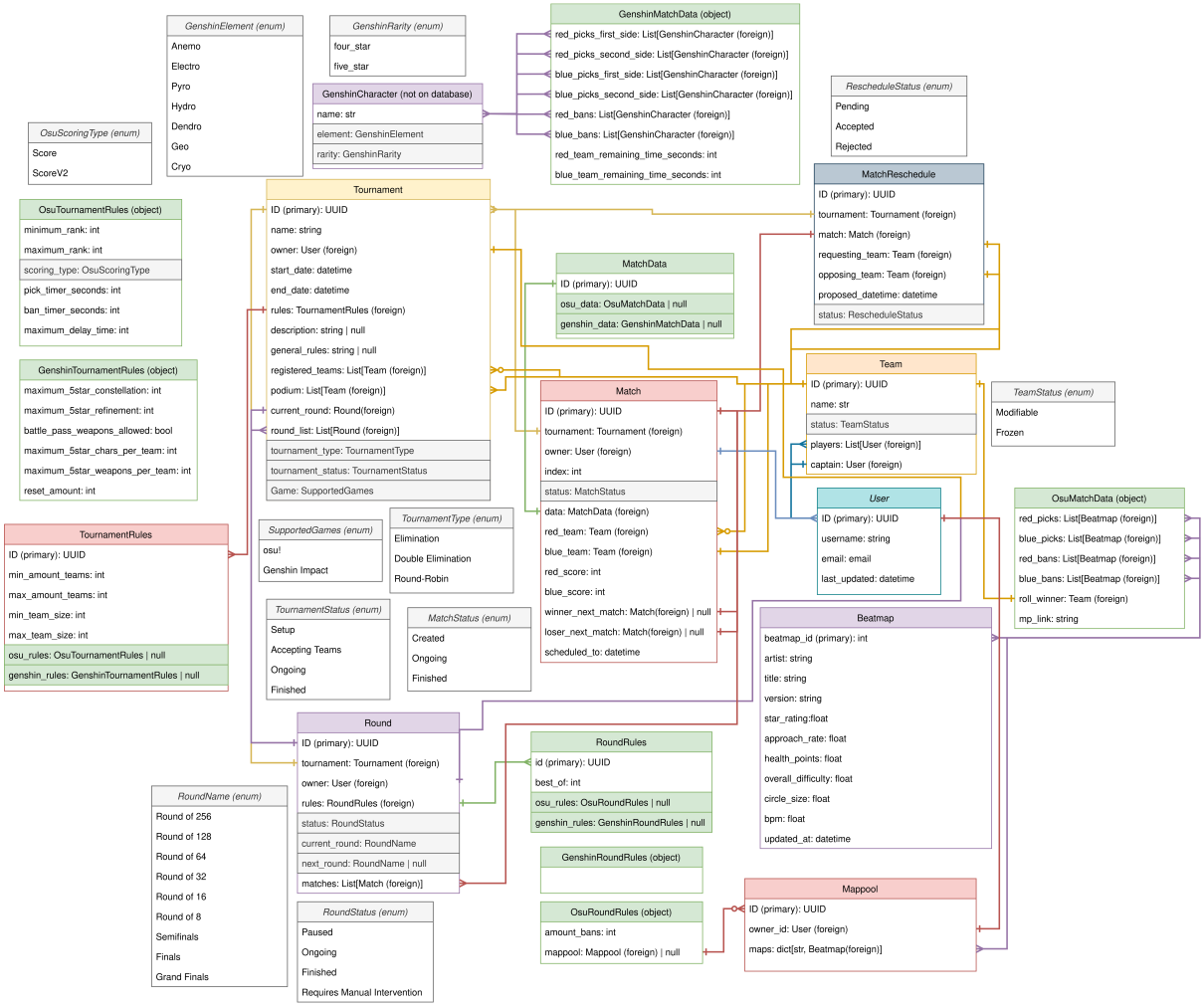

Figura 6 – Modelo de Entidades

Acima está o modelo completo de entidades. Todas as entidades com o campo de ID são uma collection diferente no MongoDB (MongoDB, Inc. (2025)).

3.4 Funcionalidades implementadas

Uma série de funcionalidades foram implementadas no servidor e no cliente:

- Os usuários conseguem criar novos torneios, visualizar torneios e editar suas regras conforme desejado.

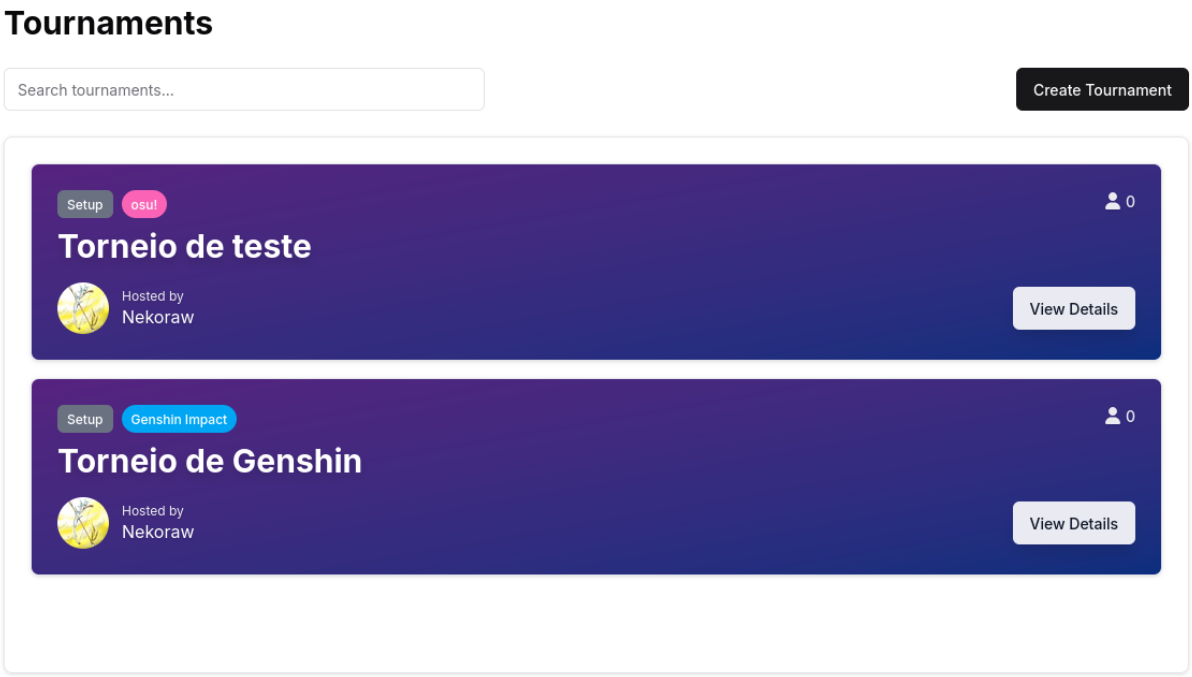

Figura 7 – Lista de torneios e botão para criação

- Regras específicas para cada jogo: É possível customizar as regras dos jogos suportados para qualquer torneio ou rodada.
- Suporte inicial para dois jogos: Genshin Impact e osu!
- Suporte a estruturas personalizadas para os jogos suportados como `GenshinCharacters` e `Mappool`.
- Preenchimento e criação automática das partidas dado a quantidade máxima de times por torneio e os times inscritos.

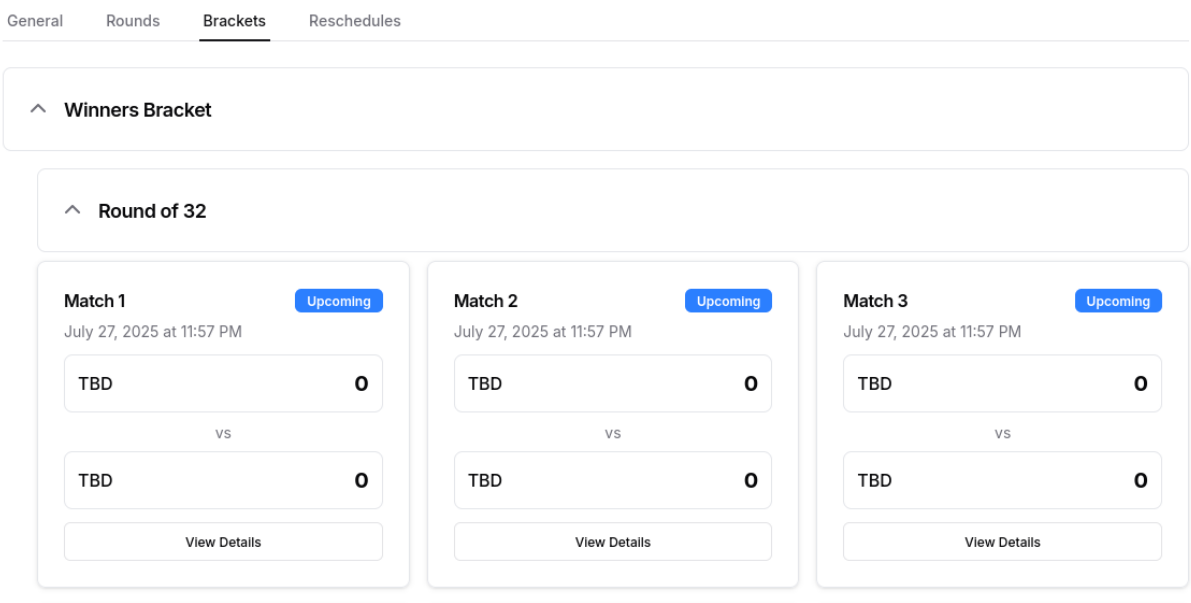

Figura 8 – Criação automática das partidas

- Suporte para vários tipos de formato de torneio: O usuário pode optar por criar um torneio de Eliminação, Eliminação dupla ou *Round Robin*.
- Definição dos resultados da partida (pontuação e regras para cada jogo)

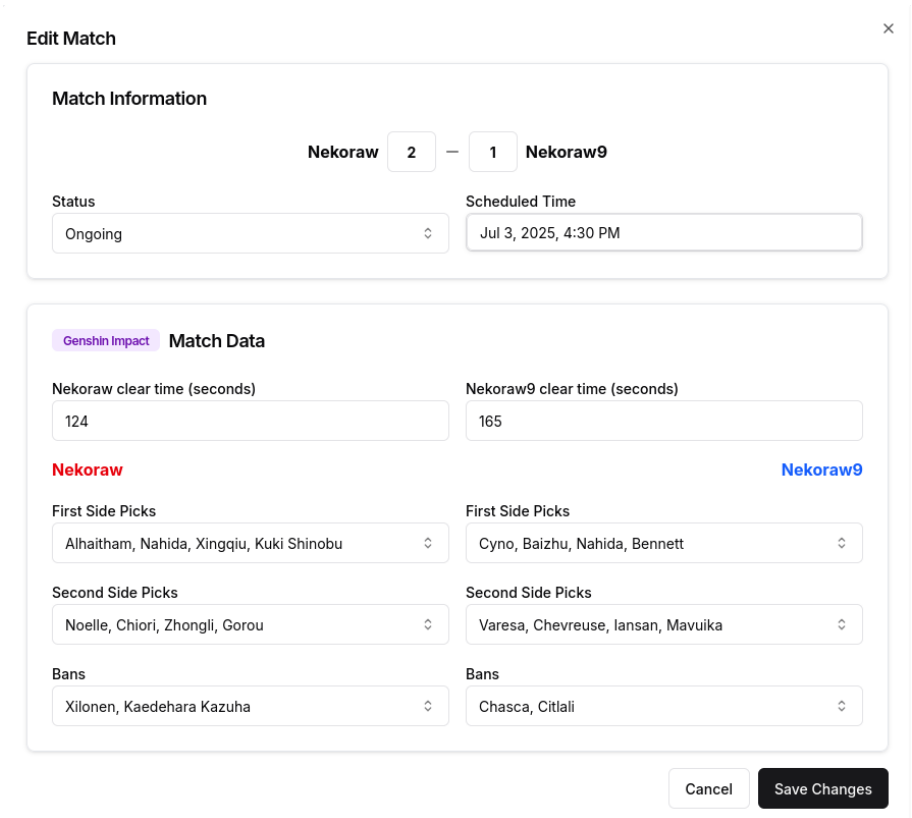

Figura 9 – Definição do resultado de uma partida de Genshin

- Definição automática dos vencedores do torneio após término das partidas.
- Criação e gerenciamento de times para participar dos torneios.
- Registro dos times nos torneios por parte dos jogadores, facilitando a vida do anfitrião do torneio.

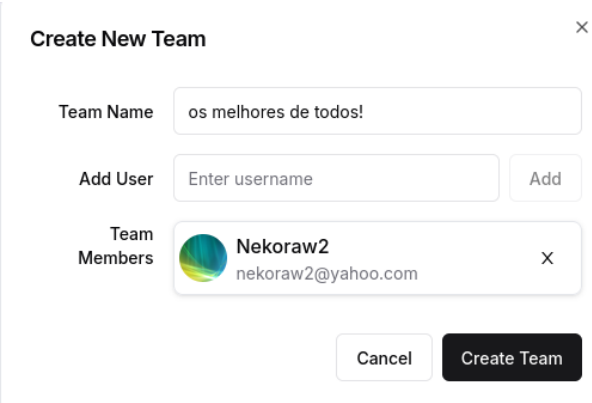

Figura 10 – Tela de criação de times

- Suporte para reagendamento de partidas automático desde que os capitães dos times aceitem o novo horário.

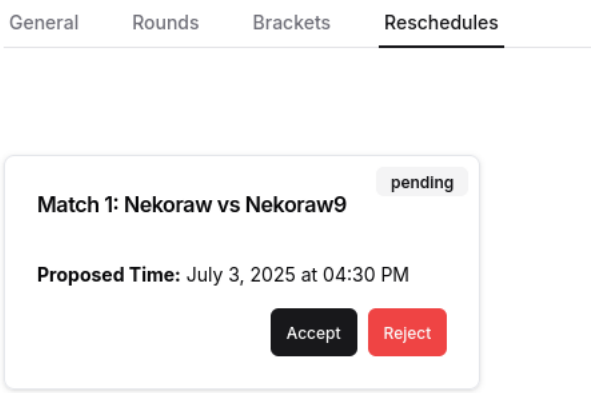

Figura 11 – Reagendamento de partidas por parte dos times

4 Principais Objetivos Técnicos

4.1 Organização do código

Para que um software seja capaz de viver por vários anos, é necessário que seja fácil mantê-lo. Essa facilidade vem de fatores como organização do código, arquitetura limpa, baixa quantidade de *code smells*. Devido a isso, resolvi usar alguns padrões de arquitetura de código no desenvolvimento do servidor.

Separation of concerns é um padrão de arquitetura de código que idealiza o fato de que cada parte do código deve ser desenvolvida para fazer somente uma coisa, mas fazer isso bem. Na implementação do servidor existem várias separações importantes:

- Entidades: Responsáveis por representar dados que existem no mundo real ou somente em conceito. É o principal usado na modelagem do servidor.
- Data Transfer Objects (DTOs): Responsáveis por ser um tipo de entidade usada somente para transferir dados entre a Internet e o ponto de entrada do servidor.
- Constantes: Reponsáveis por ser um tipo de dado específico que só admite certos valores exatos.
- Roteadores: São a porta de entrada da comunicação entre o servidor e a Internet. São responsáveis por identificar qual rota recebeu um comando e encaminhar para a execução da função correta.
- Serviços: São o cérebro do servidor. É onde as operações lógicas e regras de negócio são executadas. Existe um serviço para cada parte vital do sistema, como o serviço de torneios, que é responsável desde criar o torneio até definir o vencedor do torneio.
- Repositórios: São uma forma de controlar o fluxo de dados usados nos serviços por meio de injeção de dependência. Ou seja, todas as operações de entrada e saída para armazenamento permanente do serviço passam por um repositório. Devido à forma que estão implementados atualmente, É simples trocar o Banco de Dados usado no sistema por um completamente diferente, desde que sua interface correspondente seja implementada corretamente.

O uso dessas separações facilita a integração de novas funcionalidades ao servidor e a sua manutenção eventual.

4.2 Validação de Dados

Ao chamar uma rota do servidor, talvez seja necessário passar alguns dados para a rota: ao criar um torneio é necessário algumas informações como o nome do torneio.

Entretanto, é muito importante verificar os dados que são recebidos no servidor para que nenhum comportamento inesperado aconteça, mesmo que esses modelos estejam bem definidos como DTOs.

Para resolver esse problema, o próprio FastAPI (Ramírez (2025)), o framework utilizado para conexão do servidor com a Internet, consegue verificar os tipos dos dados recebidos na rota e disparar um erro HTTP caso ele não esteja dentro do padrão. O FastAPI faz isso por meio de modelos do Pydantic (Pydantic Contributors (2025)), que é uma biblioteca de validação de dados em Python.

O Pydantic permite a criação de modelos de dados extremamente robustos, com validação de dados desde a criação até durante a mudanças de valores da estrutura. Essa robustez é extremamente importante para que a aplicação funcione corretamente, principalmente em Python que não possui um sistema de tipagem estático.

Devido a isso, o Pydantic foi usado na criação de todas as entidades do modelo, garantindo que todos os modelos sejam consistentes e robustos com a validação automática e suas restrições customizadas para cada campo de uma entidade.

4.3 Testes Automatizados

Uma das formas de certificar que o comportamento do código escrito está correto, é por meio de testes do código. Entretanto, testar manualmente todos os comportamentos do código na mão é cansativo.

Para resolver esse problema, vários testes automatizados foram criados para verificar o funcionamento adequado dessa lógica do servidor backend. Atualmente, 57% do código está coberto e com testes que passaram. O código foi testado usando o unittest (Python Software Foundation (2025)) e a cobertura do código foi calculada usando o coverage.py (Batchelder (2025).

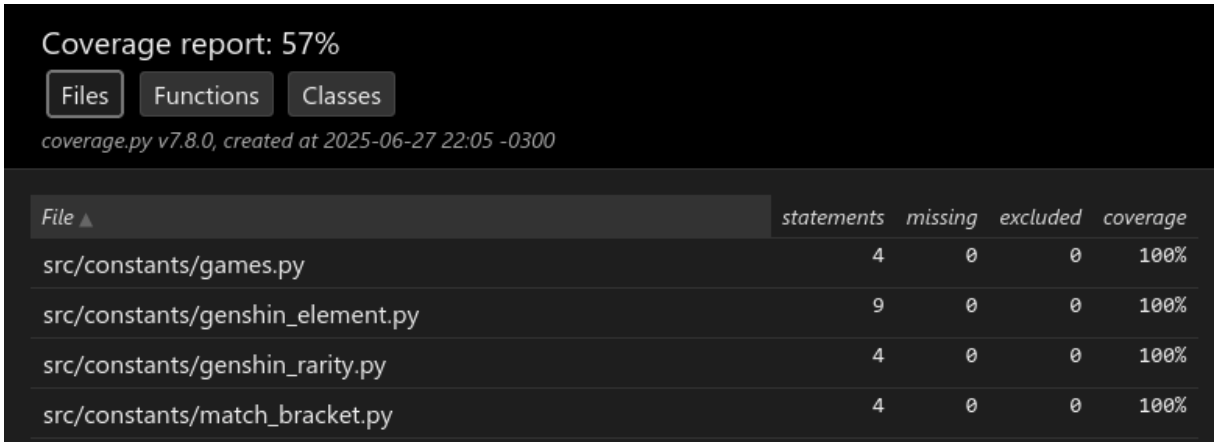

Figura 12 – Cobertura do código do servidor (backend)

4.4 Autenticação

Para garantir que certas operações no servidor sejam realizadas somente pelos usuários autorizados e pelos fins de identificação, um sistema de autenticação foi implementado. Os usuários conseguem criar uma conta no servidor e, após log-in, recebem um *token* de autorização usado em todas consultas posteriores que modifiquem dados no servidor. O formato do token usado é o *JSON Web Token* (JWT), que consegue armazenar dados JSON validados criptograficamente sem a necessidade de ser armazenado no banco de dados do servidor.

4.5 Documentação

A documentação é uma parte crucial de qualquer software, já que definem as instruções de como o software deve ser usado. Entretanto é algo extremamente difícil de se fazer dependendo do tamanho da aplicação desenvolvida. Esse foi um dos fatores que contribuiu na escolha do FastAPI como framework backend, já que ele é capaz de gerar a documentação de todas as rotas automaticamente no formato OpenAPI (OpenAPI Initiative (2025)) e as exibe em um formato amigável em uma rota usando o Swagger (Smartbear Software (2025)).

5 Arquitetura do Servidor

5.1 Ambiente de execução

O cliente está disponível para uso em https://th.nkrw.dev/. O servidor HTTP responsável por enviar o código *JavaScript*, junto com o servidor em si, o banco de dados MongoDB estão rodando em um contâiner Docker (Docker Inc. (2025a)).

O contêiner Docker é um ambiente de execução separado do resto do sistema que é usado para rodar um serviço de cada vez. Os contâineres de cada serviço não conseguem interferir nos outros serviços, o que é bom para a segurança dos dados, já que caso algum serviço for comprometido, o vetor de ataque se torna menor.

Todos os serviços estão sendo hospedados em um VPS (Virtual Private Server) com acesso direto à internet. Os serviços são:

- Servidor backend: responsável por receber requisições e processar a lógica do servidor
- Cliente Web: responsável por enviar ao navegador o *JavaScript* necessário para o funcionamento da plataforma
- MongoDB: banco de dados de objetos para armazenamento de dados de forma persistente
- Anubis (Techaro (2025)): responsável por bloquear o acesso de agentes de IA ou crawlers a algum serviço (atualmente usado para bloquear o acesso ao cliente web)
- Caddy (ZeroSSL. (2025)): proxy reverso responsável por redirecionar corretamente as requisições HTTP do cliente web, do Anubis e do servidor backend a partir da URL acessada no navegador
- Watchtower (Watchtower Contributors (2025)): serviço executado em conjunto para atualizar automaticamente os contêineres do cliente web e do servidor backend após mudanças no código

Para sincronizar o funcionamento de todos esses serviços, foi usado o Docker Compose (Docker Inc. (2025b)) que lê um arquivo que declara a forma em que cada um desses contêineres executarão em conjunto.

O código está hospedado em um repositório do GitHub que está configurado para gerar automaticamente novas imagens do servidor backend e do cliente web após receber atualização de código por meio do Github Actions (GitHub, Inc. (2025)). Após a geração dessas imagens,

o serviço Watchtower verifica que existe uma atualização nova e atualiza os contêineres do backend e do cliente web.

5.2 Detalhes de Implementação do Cliente

O cliente foi desenvolvido em *TypeScript* (Microsoft (2025)), que é uma linguagem intermediária com suporte à tipagem estática de dados e que é transformada em *JavaScript* antes de ser usada pelo navegador.

A biblioteca *openapi-typescript* (ts (2025)) foi responsável por criar automaticamente um cliente para fazer requisições HTTP para o servidor backend, adicionando a compatibildade das entidades e dos DTOs usados no backend diretamente com o cliente.

Para melhorar a performance de requisições ao backend no cliente web, foi usado o TanStack Query (TanStack LLC (2025)), que é responsável por lembrar das requisições já feitas para o servidor e guardar os resultados em cache até que os dados sejam invalidados.

Para implementar a interface do cliente em si, foi usado o Solid.js (SolidJS Contributors (2025), uma biblioteca feita para desenhar e criar interfaces reativas e velozes. Junto com o SolidUI (Essiet (2025)), a interface visual do cliente web ficou simples e moderna.

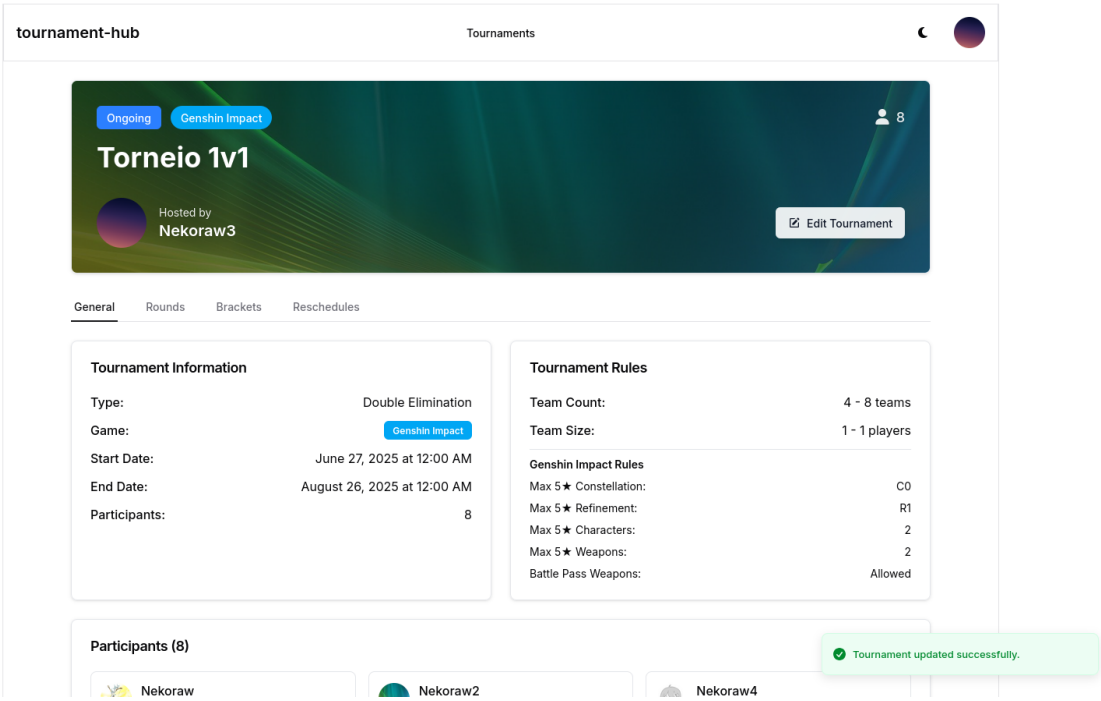

Figura 13 – Interface do Sistema

6 Exemplo de uso

Todas as rotas do servidor estão documentadas e disponíveis para uso temporariamente aqui. Os exemplos assumem que os comandos serão executados por meio do cliente web.

6.1 Criação de conta

Primeiramente, é necessário que o usuário crie uma conta no servidor para acessar outras partes do sistema. Isso pode ser feito por meio da tela de login:

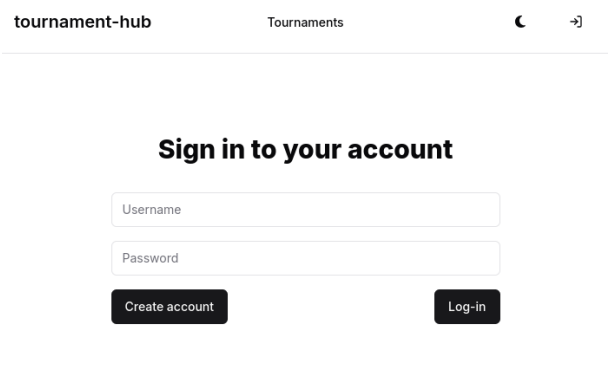

Figura 14 – Tela de Login

Caso o usuário não tiver uma conta, ela pode ser criada por meio da tela de registro:

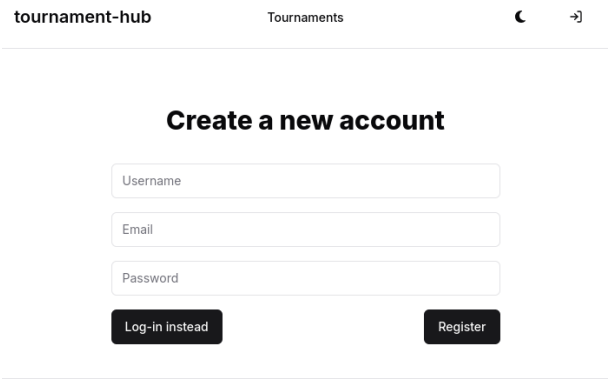

Figura 15 – Tela de Registro

6.2 Criação do torneio

Após criar a conta, o usuário deve ir para a página de torneios acessando o link no topo da página e clicar no botão de criar torneio:

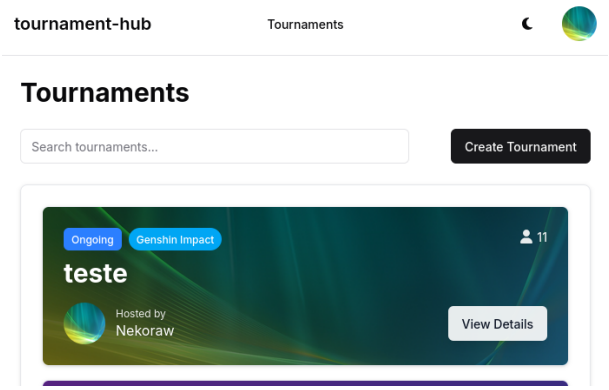

Figura 16 – Página de torneios

Após isso, o usuário pode definir as regras do torneio:

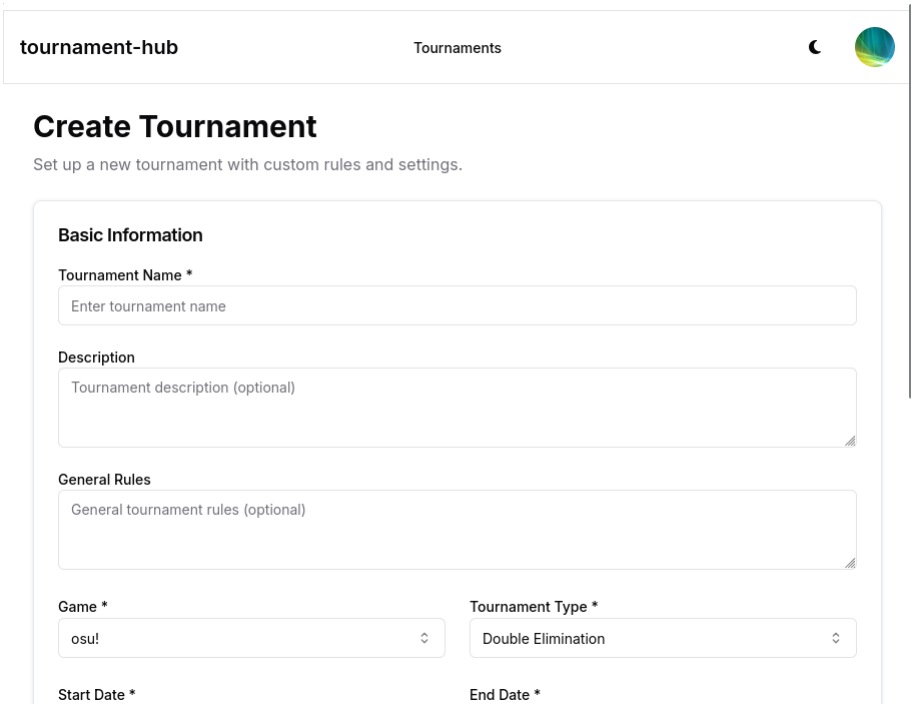

Figura 17 – Tela de criação de torneio

6.3 Registro de jogadores

Após definir todos os dados descritos acima, é necessário definir as regras das rodadas do torneio na aba *Rounds* na tela de torneio:

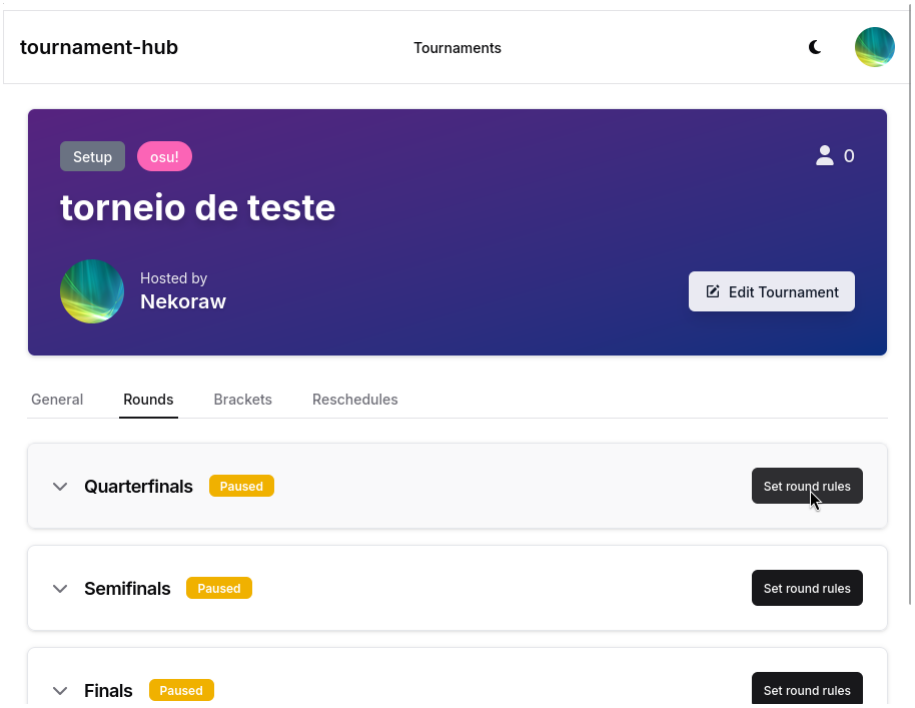

Figura 18 – Definindo as regras de torneio

Após isso, clique no botão de editar torneio e clique no botão para começar a aceitar registros para o torneio:

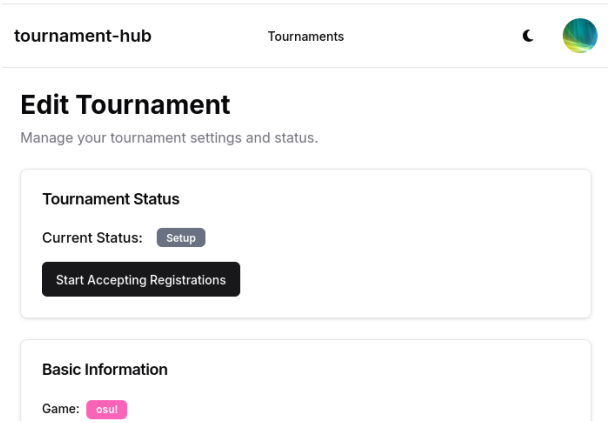

Figura 19 – Configurações do torneio

6.4 Fluxo do torneio

Após os times se registrarem no torneio, é possível começar o torneio em si por meio do mesmo botão que começou os registros. Após isso, a rodada irá começar e, após definir o resultado de todas as partidas, é possível avançar o torneio para o próximo estágio por meio do mesmo botão.

Após avançar o torneio por todos os estágios, o torneio será finalizado.

7 Conclusão

A criação dessa uma plataforma de gerenciamento de torneios para jogos online resolve um grande problema nas comunidades dos jogos online: a falta de estrutura e acessibilidade para organização de torneios. Esse relatório detalhou o design, a modelagem e as funcionalidades do servidor, que constitui o núcleo dessa solução.

A plataforma oferece suporte a dois jogos e formatos de torneio diferentes, garantindo flexibilidade e escalabilidade. A arquitetura organizada do servidor e a validação robusta dos dados asseguram que o sistema seja confiável, eficiente e fácil de manter. Além disso, a documentação automatizada facilita o desenvolvimento futuro e a integração com outros clientes.

O cliente web implementado em si é uma ótima maneira de interagir com a plataforma, por meio de uma interface simples e moderna, criar, gerenciar e participar de torneios de várias comunidades nunca foi tão fácil.

Referências

BATCHELDER, N. *Coverage.py*. [S.l.], 2025. Disponível em: <https://coverage.readthedocs.io/en/7.9.1/>. Acesso em: 27 jun. 2025. Citado na página 16.

DOCKER INC. *Docker*. [S.l.], 2025. Disponível em: <https://www.docker.com/>. Acesso em: 23 abr. 2025. Citado na página 18.

DOCKER INC. *Docker Compose*. [S.l.], 2025. Disponível em: <https://docs.docker.com/compose/>. Acesso em: 23 abr. 2025. Citado na página 18.

ESSIET shadcn tremorlabs . S. E.-K. . M. *SolidUI*. [S.l.], 2025. Disponível em: <https://www.solid-ui.com/>. Acesso em: 27 jun. 2025. Citado na página 19.

GITHUB, INC. *GitHub Actions*. [S.l.], 2025. Disponível em: <https://github.com/features/actions>. Acesso em: 23 abr. 2025. Citado na página 18.

MICROSOFT. *TypeScript Language*. [S.l.], 2025. Disponível em: <https://www.typescriptlang.org/>. Acesso em: 23 abr. 2025. Citado na página 19.

MONGODB, INC. *MongoDB*. [S.l.], 2025. Disponível em: <https://www.mongodb.com/>. Acesso em: 23 abr. 2025. Citado 2 vezes nas páginas 6 e 11.

OMS, K. *My Love*. [S.l.], 2014. Disponível em: <https://osu.ppy.sh/beatmapsets/163112>. Citado na página 8.

OPENAPI INITIATIVE. *OpenAPI Specifdication*. [S.l.], 2025. Disponível em: <https://www.openapis.org/>. Citado na página 17.

PYDANTIC CONTRIBUTORS. *Pydantic Library*. [S.l.], 2025. Disponível em: <https://docs.pydantic.dev/latest/>. Acesso em: 23 abr. 2025. Citado na página 16.

PYTHON SOFTWARE FOUNDATION. *unittest*. [S.l.], 2025. Disponível em: <https://docs.python.org/3/library/unittest.html>. Acesso em: 27 jun. 2025. Citado na página 16.

RAMÍREZ, S. *Framework FastAPI*. [S.l.], 2025. Disponível em: <https://fastapi.tiangolo.com/pt/>. Acesso em: 23 abr. 2025. Citado 2 vezes nas páginas 6 e 16.

SMARTBEAR SOFTWARE. *Swagger*. [S.l.], 2025. Disponível em: <https://swagger.io/>. Citado na página 17.

SOLIDJS CONTRIBUTORS. *SolidJS*. [S.l.], 2025. Disponível em: <https://www.solidjs.com/>. Acesso em: 23 abr. 2025. Citado na página 19.

TANSTACK LLC. *TanStack Query*. [S.l.], 2025. Disponível em: <https://www.openapis.org/>. Citado na página 19.

TECHARO. *Anubis*. [S.l.], 2025. Disponível em: <https://anubis.techaro.lol/>. Acesso em: 23 abr. 2025. Citado na página 18.

TS openapi. *openapi-typescript*. [S.l.], 2025. Disponível em: <https://openapi-ts.dev/>. Acesso em: 27 jun. 2025. Citado na página 19.

WATCHTOWER CONTRIBUTORS. *Watchtower*. [S.l.], 2025. Disponível em: <https://containrrr.dev/watchtower/>. Acesso em: 23 abr. 2025. Citado na página 18.

ZEROSSL. *Caddy*. [S.l.], 2025. Disponível em: <https://caddyserver.com/>. Acesso em: 23 abr. 2025. Citado na página 18.