UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

FELIPE CESAR MACCARI

SISTEMA WEB PARA CONTROLE DE TORNEIO DE TÊNIS EM MODELO DE PIRÂMIDE PARA CLUBES NA CIDADE DE PATO BRANCO

PATO BRANCO

2022

FELIPE CESAR MACCARI

SISTEMA WEB PARA CONTROLE DE TORNEIO DE TÊNIS EM MODELO DE PIRÂMIDE PARA CLUBES NA CIDADE DE PATO BRANCO

Web Application to Control Tennis Tournament in Pyramid Model For Clubs In The City of Pato Branco

Trabalho de conclusão de curso de graduação apresentado como requisito para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas da Universidade Tecnológica Federal do Paraná (UTFPR).
Orientador(a): Profª. Andréia Scariot Beulke.

PATO BRANCO

2022

FELIPE CESAR MACCARI

SISTEMA WEB PARA CONTROLE DE TORNEIO DE TÊNIS EM MODELO DE PIRÂMIDE PARA CLUBES NA CIDADE DE PATO BRANCO

Trabalho de conclusão de curso de graduação apresentado como requisito para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas da Universidade Tecnológica Federal do Paraná (UTFPR).

Data de aprovação: Dia/mês por extenso/ano

Andréia Scariot Beulke
Mestra
Universidade Tecnológica Federal do Paraná

Rúbia Eliza de Oliveira Schultz Ascari
Doutora
Universidade Tecnológica Federal do Paraná

Vinicius Peorini
Mestre
Universidade Tecnológica Federal do Paraná

PATO BRANCO

2022

RESUMO

Os benefícios que a prática esportiva traz para os praticantes, sejam atletas casuais ou profissionais, são vários. O esporte além de aumentar a qualidade da saúde das pessoas também amplifica o senso de comunidade entre elas, e o tênis é conhecido como um esporte de referência para ensinar respeito e união, que são fatores essenciais para uma boa convivência em sociedade. Assim, o incentivo ao esporte deve ser reforçado de diversas maneiras possíveis para motivar aqueles que já praticam e trazer novos praticantes para as modalidades, e as competições esportivas podem ser grandes catalisadoras deste incentivo. Diante disso, neste trabalho é apresentado o desenvolvimento de um sistema web para controle de torneios de tênis no modelo de pirâmide para utilização em clubes de tênis na cidade de Pato Branco – Paraná, visando melhorar o controle dos administradores dos torneios e obter um engajamento cada vez maior para aqueles que participam deste tipo de torneio, por meio de uma interface interativa e funcional, exibindo diversas estatísticas de performance dos jogadores e incentivando a competitividade. Para a implementação do sistema, as principais tecnologias utilizadas foram a biblioteca NextJS, que é um *framework* baseado na biblioteca ReactJS, para a criação do lado cliente (*front-end*) e para a criação do lado servidor foi utilizada a biblioteca Express que permite a criação de API REST utilizando NodeJS e o banco de dados PostgresSQL.

Palavras-chave: tênis de campo; torneios esportivo; sistema web de controle de torneio esportivo; javascript; postgres.

ABSTRACT

The countless scientifically proven benefits that the practice of sports brings to practitioners, be they casual or professional athletes, are undeniable. Sport, in addition to increasing people's quality of health, also amplifies the sense of community among them, and tennis is known as a reference sport in teaching respect and unity, essential factors for a good coexistence in society. Therefore, the incentive to sport should be reinforced in several possible ways to motivate those who already practice and bring new practitioners to the modalities, and sports competitions can be great catalysts for this incentive. With this in mind, the present work reports the development of a web application to control tennis tournaments in the pyramid model for use in a tennis clubs in the city of Pato Branco – Paraná, and it is intended to provide better control by those who manage the tournaments and a growing engagement on the part of those who participate in this type of tournament, through an interactive and functional interface displaying various player performance statistics and encouraging competitiveness. For the implementation of the system, the main technologies used were the NextJS library, which is a framework based on the ReactJS library, for the creation of the client side (front-end) and for the creation of the server side, the Express library was used, which allows REST API creation using NodeJS and PostgresSQL database.

Keywords: lawn tennis; sports competitions; web applications to sports tournament management; javascript; postgres.

LISTA DE ILUSTRAÇÕES

Figura 1 - Exemplo de formato de torneios de tênis em pirâmide 11
Figura 2 - Demonstração do event-loop do motor V8 18
Figura 3 – Interface principal do controle do torneio por planilha 20
Figura 4 - Tabela de dados dos desafios do torneio 21
Figura 5 - Tabela de estatísticas dos desafios 21
Figura 6 - Diagrama de casos de uso da aplicação 25
Figura 7 - Diagrama de classes com métodos 30
Figura 8 – Diagrama de entidade-relacionamento 34
Figura 9 – Interface para login dos administradores 35
Figura 10 – Formulário de cadastro de um novo torneio 36
Figura 11 – Interface de cadastro de novos jogadores 36
Figura 12 – Exibição do torneio em formato de pirâmide 37
Figura 13 – Exibição das estatísticas iniciais do torneio vigente 38
Figura 14 – Tela de listagem dos jogadores cadastrados 39
Figura 15 – Interface de cadastro de um novo desafio 40
Figura 16 – Interface de listagem dos desafios cadastrados 41
Figura 17 – Interface para adicionar resultado de um desafio 42
Figura 18 – Exibição da pirâmide com desafios em andamento 42
Figura 19 – Exibição dos detalhes de um desafio na visualização de pirâmide 43
Figura 20 – Exibição da listagem dos desafios com resultados e *status*. 44
Figura 21– Estruturas de pastas do *backend* 45
Figura 22 – Estrutura de pastas do *frontend* 57
Figura 23 – Estrutura de arquivos de rotas do *frontend* 58

LISTA DE QUADROS

Quadro 1 - Lista de ferramentas e tecnologias 16
Quadro 2 - Lista de Requisitos Funcionais do sistema 23
Quadro 3 - Lista de Requisitos Não Funcionais do sistema 24
Quadro 4 - Caso de uso Manter Torneios 25
Quadro 5 - Caso de uso Manter Jogadores 25
Quadro 6 - Caso de uso Manter Desafios 26
Quadro 7 - Caso de uso Manter Resultados 26
Quadro 8 - Caso de uso Manter *Status* 26
Quadro 9 - Caso de uso Vincular Jogadores em Torneio 27
Quadro 10 - Caso de uso Visualizar Torneios 27
Quadro 11 - Caso de uso Visualizar Estatísticas 27
Quadro 12 - Caso de uso Visualizar Desafios 27
Quadro 13 - Caso de uso Controlar Acesso de Usuário 28
Quadro 14 - Caso de uso Sorteio das Posições Iniciais 28
Quadro 15 - Caso de uso Manter Posições dos Jogadores 28
Quadro 16 - Caso de uso Controlar Pontuação dos Jogadores 28
Quadro 17 - Caso de uso Calcular Estatísticas dos Jogadores 29
Quadro 18 - Descrição da classe User 30
Quadro 19 - Descrição da classe Tournament 31
Quadro 20 - Descrição da classe TournamentPlayer 31
Quadro 21 - Descrição da classe Player 32
Quadro 22 - Descrição da classe Challenge 32
Quadro 23 - Descrição da classe ChallengeResults 33

LISTA DE CÓDIGOS

Listagem 1 – Código de controle da pontuação por planilha 21
Listagem 2 – Configuração de agrupamento de rotas..... 46
Listagem 3 – Configuração de sub rotas do modulo *Tournament* 46
Listagem 4 – *Middleware* de rotas autenticadas..... 48
Listagem 5 – *Controller* de sorteio dos jogadores no torneio 49
Listagem 6 – *Usecase* de sorteio dos jogadores no torneio..... 49
Listagem 7 – Importações de dependências do *useCase* de resultados do desafio..... 50
Listagem 8 – Declarações de função do *useCase* de resultados do desafio 51
Listagem 9 – Validações de desafio do *useCase* de resultados do desafio..... 52
Listagem 10 – Definição de pontuações do *useCase* de resultados do desafio..... 54
Listagem 11 – Componente principal da aplicação cliente 55
Listagem 12 – Exemplo de componente de rota autenticada..... 56
Listagem 13 – Código do componente da pirâmide no cliente 59
Listagem 14 – Código do item da pirâmide que representa o jogador 60

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
ARO	*Army Research Office*
ATP	*Association of Tennis Professionals*
BSD	*Berkeley Software Distribution*
ECMA	*European Computer Manufacturers Association*
HTTP	*Hypertext Transfer Protocol*
IDE	*Integrated Development Environment*
ITF	*International Tennis Federation*
NSF	*National Science Foundation*
ORM	*Object Relational Mapping*
REST	*Representational State Transfer*
RF	Requisitos Funcionais
RNF	Requisitos Não Funcionais
SGBD	Sistema Gerenciador de Banco de Dados
SSR	*Server-Side Rendering*

SUMÁRIO

1 INTRODUÇÃO **9**
1.1 Considerações Iniciais **9**
1.2 Objetivos **11**
1.2.1. Objetivos Gerais 11
1.2.2. Objetivos Específicos 11
1.3 Justificativa **12**
1.4 Estrutura do Trabalho **12**
2 REFERENCIAL TEÓRICO **13**
3 MATERIAIS E MÉTODO **16**
3.1 Materiais **16**
3.1.1 PostgreSQL **16**
3.1.2 Typescript **17**
3.1.3 NodeJs **18**
3.1.4 Next.Js **19**
3.2 Método **19**
4 RESULTADOS **22**
4.1 Escopo do Sistema **22**
4.2 Modelagem do Sistema **22**
4.2.1. Requisitos Funcionais 23
4.2.2. Requisitos Não Funcionais 23
4.2.3. Casos de Uso 24
4.2.4. Diagrama de Classes 29
4.2.5. Diagrama Entidade-Relacionamento 34
4.3 Apresentação do Sistema **35**
4.4 Implementação do Sistema **44**
4.4.1. Servidor 44
4.4.2. Cliente 55
5 CONCLUSÃO **62**

1 INTRODUÇÃO

Neste capítulo são apresentadas as considerações iniciais, os objetivos, a justificativa da realização deste trabalho e a organização do texto por meio de uma breve descrição dos seus capítulos.

1.1 Considerações Iniciais

O tênis tem se desenvolvido muito no Brasil ao longo dos anos. Segundo pesquisa especializada no perfil, comportamento e hábitos dos fãs de esportes, em 2019 o país contava com cerca de 27 milhões de fãs desta modalidade (IBOPE REPUCOM, 2019).

Os registros históricos trazem para o esporte uma origem controversa por ser muito antigo. Contudo, diversas fontes o ligam a jogos populares da época do Império Romano e que sua popularização se deu no meio da aristocracia francesa e da nobreza inglesa, de onde, inclusive, existe o primeiro registro de um torneio oficial de tênis, que é o torneio de Wimbledon em 1877 (CUNHA; PINTO, 1998).

O estado do Rio de Janeiro foi o pioneiro no esporte no Brasil. No final da década de 1880, engenheiros britânicos chamados para trabalhar na melhoria da estrutura das cidades e estradas de ferro trouxeram além de seu conhecimento de construção, fatores culturais como a prática do *lawn tennis* (tênis de gramado) (GONÇALVES et al., 2018).

No Brasil o tênis também começou a se popularizar por grupos e famílias com maiores condições financeiras da sociedade, e historicamente sempre foi considerado um esporte praticado por pessoas com alto poder aquisitivo. Este é um fato que está mudando ao longo do tempo pois conforme o esporte torna-se mais popular entre pessoas de faixas etárias e classes sociais distintas, o custo de estruturas e equipamentos diminui de forma considerável no mercado.

A ascensão de alguns atletas brasileiros no ranking mundial da *Association of Tennis Professionals* (ATP) ajudou para que o esporte fosse divulgado amplamente no país, tendo como maiores referências até hoje o tenista Gustavo Kuerten, que alcançou a posição de número 1 do mundo no dia 03 de dezembro de 2000, e a tenista Maria Esther Bueno, que foi número 1 do mundo em 3 anos (1959, 1964 e 1966) (CARTA, 2004).

Alguns dados corroboram a influência dos atletas, conforme um estudo divulgado pela *International Tennis Federation* (ITF), mostrando que de 2001 até 2006, período do auge do tenista Gustavo Kurten, o número de praticantes de tênis cresceu em média 11% alcançando a marca de 1.457.715 pessoas (GRIZZO, 2007).

A divulgação de estudos e pesquisas que associam a prática esportiva a melhora da qualidade de vida das pessoas também é um grande incentivo. Segundo uma pesquisa realizada em 2013 pelo Ministério do Esporte, os motivos que levam as pessoas a praticarem esporte são: melhorar a qualidade de vida e bem-estar (36.3%), melhorar o desempenho físico (29,3%), relaxar no tempo livre (11,9%), melhorar a harmonia corporal (5,3%), competir (3,6%), indicações médicas, prêmios e bolsas (4,5%). Outros motivos são: para relaxar e poder passar um tempo com amigos, obter títulos de competições e escalada em *rankings* de federações ou dos próprios clubes e outros praticantes estão atrás de uma carreira no esporte tentando se profissionalizar (MINISTÉRIO DO ESPORTE, 2015).

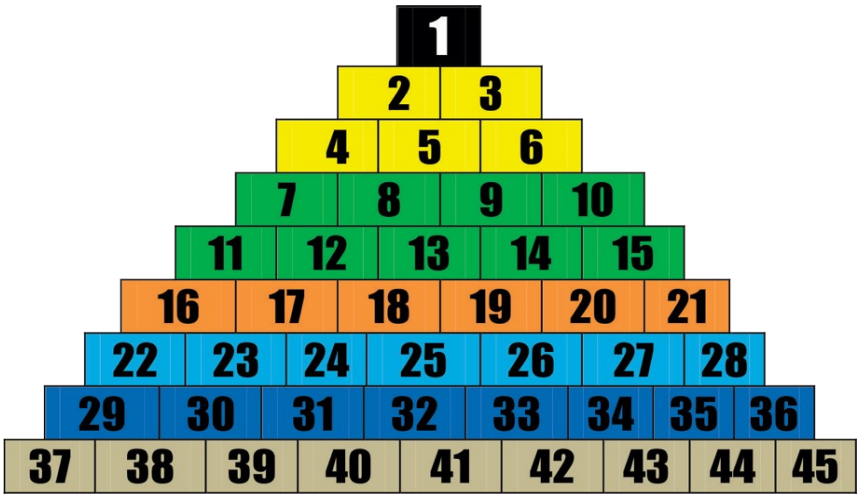
Os torneios esportivos motivam muito os atletas dentro dos clubes em cidades de todo Brasil e podem ter diversos formatos. Essa movimentação é importante para os clubes que conseguem manter os associados e angariar recursos para melhorar a estrutura do esporte e também para a integração entre os atletas, melhorando suas técnicas de jogo e conhecendo outras pessoas.

Na cidade de Pato Branco - Paraná, os torneios no formato de pirâmide estão se tornando muito populares pelo fácil entendimento, longa duração e grande integração entre jogadores. Esse tipo de torneio, que possui este nome devido ao formato piramidal das posições dos jogadores, é definido pelo objetivo principal de defender sua posição atual e desafiar outro jogador que esteja sempre na linha de cima para ocupar a posição deste. Os torneios em modelo de pirâmide são tradicionais em competições esportivas, principalmente aquelas em que o esporte é individual. O que difere este projeto são as regras que foram desenvolvidas ao longo do tempo como adaptação ao modelo tradicional de pirâmide, para que seja melhor apreciado pelos jogadores.

Diante do exposto, este trabalho propõe o desenvolvimento de um sistema web para que os clubes de tênis possam gerenciar os torneios do formato de pirâmide, controlando de forma automática a pontuação, posicionamento dos jogadores, prazos para os jogos e demais regras que se aplicam a este formato de torneio.

A Figura 1 apresenta um exemplo da pirâmide, em que os jogadores nas posições 2 e 3 só podem desafiar o jogador na posição 1, os jogadores nas posições 4, 5 e 6 só podem desafiar jogadores na posição 2 e 3 e, assim, sucessivamente.

Figura 1 - Exemplo de formato de torneios de tênis em pirâmide



Fonte: Cachoeira Tênis (2021)

1.2 Objetivos

A seguir são apresentados os objetivos do sistema proposto neste trabalho, sendo que o objetivo geral está relacionado com o resultado principal e os objetivos específicos ilustram funcionalidades do sistema e complementam o objetivo geral.

1.2.1 Objetivos Gerais

Desenvolver um sistema web para controle de torneios de tênis no modelo de pirâmide, segundo o regulamento criado pela Comissão dos Tenistas do clube Grêmio Industrial Patobranquense da cidade de Pato Branco - Paraná.

1.2.2 Objetivos Específicos

Proporcionar aos administradores do torneio de tênis no modelo de pirâmide no Grêmio Industrial Patobranquense a marcação dos desafios realizados entre os jogadores e os prazos para realização de jogos, desistências, recusas e pontuações.

Realizar de forma automática o controle de disponibilidade dos desafios segundo as regras impostas pelo regulamento.

Permitir que os jogadores possam acompanhar a movimentação dos jogos do torneio, a classificação em visualização de pirâmide e o ranqueamento em pontos de cada jogador.

1.3 Justificativa

A prática esportiva e o exercício físico são importantes para a saúde e qualidade de vida das pessoas. Inclusive é uma das estratégias adotadas na medicina esportiva para tratar, prevenir ou reabilitar pessoas com debilidades ou doenças (SILVA; OLIVEIRA, 2018).

Além de o esporte ser considerado benéfico para a saúde dos praticantes, a prática esportiva também se tornou importante na sociedade de consumo, sendo bastante difundida e atingindo pessoas de faixas etárias e culturas distintas.

O tênis é um esporte considerado muito importante para o desenvolvimento físico e mental dos praticantes, porque envolve movimentos distintos como correr, saltar e arremessar, atenção e concentração (GODTSFRIEDT; ANDRADE; VASCONCELLOS, 2014).

A necessidade que justifica o trabalho que será desenvolvido, é a modernização do controle dos jogos do torneio de tênis em formato de pirâmide no clube Grêmio Industrial Patobranquense em Pato Branco - Paraná, tendo em vista que atualmente todo o controle das regras do torneio é feito de forma manual em planilhas digitais, que gera um grande trabalho para todos aqueles que fazem o gerenciamento. Visa-se também evitar erros humanos em relação às regras que regem o torneio os quais podem afetar a pontuação e consecutivamente a classificação dos jogadores.

Ademais, desenvolver um sistema que seja confiável e sólido com as regras do torneio, trará um maior engajamento por parte dos jogadores, e consequentemente benefícios pessoais em questão de saúde e para o esporte como um todo que se desenvolve mais rápido com uma quantidade maior de adeptos.

1.4 Estrutura do Trabalho

As informações a seguir estão organizadas na forma de capítulos. O Capítulo 2 apresenta o referencial teórico sobre conceitos abordados neste trabalho. O Capítulo 3 mostra os materiais e o método previsto no desenvolvimento do trabalho. Por fim, o Capítulo 4 apresenta a modelagem do sistema e o desenvolvimento das interfaces e da implementação do sistema. Por fim, está a conclusão e as referências utilizadas no trabalho.

2 REFERENCIAL TEÓRICO

Este capítulo apresenta o referencial teórico necessário para o desenvolvimento deste estudo. As seções a seguir, apresentam conceitos para melhor entendimento do trabalho e do sistema proposto. A seção 2.1 integraliza as regras básicas da pontuação de partidas de tênis para compreender as funções do sistema. A seção 2.2 apresenta uma descrição detalhada de como funcionam os ranqueamentos em formato de pirâmide para torneios esportivos com foco no regulamento criado pela Comissão dos Tenistas do clube Grêmio Industrial Patobranquense da cidade de Pato Branco - Paraná.

2.1 Regras Básicas de Pontuação do Tênis

A pontuação das partidas de tênis é dividida em três partes complementares, são elas: pontos, *games* e *sets*. Quando um jogador marca quatro pontos, um *game* é marcado. Já um *set* é marcado quando um jogador marca seis *games.*

Os pontos não são contados de forma sequencial, no tênis conta-se 15, 30, 40 e o último ponto não é descrito pois ele encerra a disputa de um *game*. “De onde veio 15, 30, 40? Do relógio. Mas então não seria 45 ao invés de 40? Sim, mas, seguramente, com o passar do tempo, as pessoas adotaram 40, pois seria mais fácil de dizer do que 45”. (GRIZZO, 2017).

Dependendo do torneio, é definida a quantidade de *sets* que precisam ser marcados para um jogador ganhar a partida. De modo geral, é comum que sejam disputados três *sets,* sendo que quem vencer dois *sets* garante a vitória. Esse modo é conhecido como “melhor de três”.

Alguns fatos devem ser considerados em relação à pontuação. O primeiro é que caso ambos os jogadores marquem três pontos, é preciso que um deles marque dois pontos de vantagem para garantir a vitória do *game* disputado. Outro fato em relação à disputa dos *sets* é que quando ambos os jogadores marcam cinco *games* deve-se abrir dois *games* de vantagem, ou seja, o *set* acaba quando um dos jogadores marcar sete *games*. Caso o *set* empate em seis a seis *games*, deve-se disputar um *set* especial chamado de *tiebreak* ou desempate.

Os *tiebreaks* são *sets* especiais disputados em situação de empate, eles são contados com pontos sequenciais. Quando é para o desempate que vai definir um *set* o *tiebreak* é disputado até sete, e quando é um *tiebreak* para definir quem vai ganhar a partida, ou seja, definição de empate em *sets*, ganha o jogador que marcar dez pontos primeiro. É importante reforçar que o *tiebreak* só pode ser finalizado com um jogador tendo dois pontos de diferença do outro, mesmo que essa pontuação seja maior que sete ou dez como citado anteriormente.

2.2 Sistema de Torneio em Formato de Pirâmide

O sistema de torneio em formato de pirâmide é comum em competições esportivas, principalmente para esportes individuais. Esse formato é muito utilizado em clubes e associações esportivas pelo fácil entendimento, alto volume de jogos e grande integração entre todos os atletas participantes.

O objetivo principal do jogador é defender sua posição atual e desafiar outro atleta que esteja uma linha acima da sua atual para ocupar a posição deste visando alcançar o topo da pirâmide.

O desenvolvimento deste trabalho está embasado no Regulamento da 4ª Pirâmide do Tênis do clube Grêmio Industrial Patobranquense. Esse documento foi desenvolvido pela Comissão dos Tenistas do referido clube, que traz todas as regras básicas dos torneios em formato de pirâmide e as adaptações realizadas ao longo dos anos em todas as edições do torneio.

As regras descritas a seguir são importantes, pois algumas tornaram-se requisitos funcionais do sistema proposto, tendo em vista que é nesses requisitos que o controle do projeto é fundamentado. Essas regras são:

a) A posição inicial dos tenistas é definida por sorteio com todos os interessados;

b) Novos tenistas podem entrar na disputa a qualquer momento. Estes iniciam sempre na última posição não ocupada na base da pirâmide;

c) Cada tenista só poderá desafiar um atleta da linha acima da sua;

d) O prazo para que o jogo aconteça é de sete dias;

e) Os prazos podem ser aumentados, caso no dia marcado para o jogo, as quadras não estejam em condições de jogo (chuvas fortes, manutenções ou qualquer outro bloqueio). Nestes casos, o desafio ganha mais 7 dias de prazo partindo da data do jogo não ocorrido;

f) Motivos como lesão, viagem, estudo e trabalho não dão direito a um maior prazo para o jogo. Neste caso os dois atletas devem chegar a um acordo para decidir quem foi o vencedor;

g) Atletas que já estão com um desafio em andamento, ficarão identificados com a cor vermelha na visualização da pirâmide. Estes não podem ser desafiados;

h) Um desafio é considerado finalizado quando o jogo é realizado, ou caso exista desistência por qualquer motivo de um dos jogadores;

i) Após ser desafiado por alguém, quando o desafio é finalizado, o atleta não pode receber novos desafios por dois dias. Esta regra garante que o jogador possa realizar desafios neste período e na visualização da pirâmide este ficará identificado com a cor preta;

j) Os jogadores podem, ao longo de todo o torneio, recusar um desafio recebido sem perder pontos ou posição;

k) Após dois jogadores se enfrentarem, um próximo confronto direto só é possível depois de sete dias corridos;

l) O desafio deve ser anunciado em grupo de WhatsApp, controlado pela secretaria de esportes e pela comissão dos tenistas do clube;

m) Os jogos devem ser obrigatoriamente realizados nas quadras do clube Grêmio Industrial Patobranquense;

n) Devido à disponibilidade dos horários do clube, cada jogador inicia com dois *games* em cada *set*. O primeiro que ganhar dois *sets* ganha a partida, tendo um *tiebreak* como desempate em caso de uma vitória em *set* para cada lado;

o) Cada vitória por dois *sets* a zero contabiliza 40 pontos ao vencedor e 10 pontos ao perdedor. Uma vitória por dois *sets* a um contabiliza 30 pontos ao vencedor e 20 pontos ao perdedor;

p) Caso o prazo da partida expire sem desistência de nenhum dos jogadores, ambos perdem 50 pontos;

q) No caso de desistências, os jogadores perdem 50 pontos para cada jogo que desistiram a partir da segunda desistência.

As regras descritas são adaptações realizadas ao longo de seis torneios realizados na cidade, nas quais os jogadores junto com a Comissão dos Tenistas estiveram propondo ideias e melhorias para criar um torneio mais divertido, com mais engajamento e competitividade.

3 MATERIAIS E MÉTODO

A seguir são elencados os materiais e descrito o método utilizado para a implementação do sistema que foi obtido como resultado da realização deste trabalho.

3.1 Materiais

O Quadro 1 apresenta a lista de ferramentas e tecnologias que foram utilizadas para o desenvolvimento deste trabalho.

Quadro 1 - Lista de ferramentas e tecnologias

Ferramenta / Tecnologia	Versão	Finalidade
Axios	0.24	Biblioteca para gerenciamento de requisições HTTP
Chakra UI	1.6	Biblioteca de componentes personalizáveis
ExpressJS	4.17	Framework para criação da API com NodeJS
Lucidchart	1.0	Criação do diagrama de classe e de entidade-relacionamento
Nextjs	12	*Framework* para criação do *front-end*
NodeJS	17.0	Runtime Javascript para *back-end*
PostgreSQL	13.4	Banco de dados
React-Query	3.32	Biblioteca para acompanhamento de requisições para API REST no *front-end*
TypeORM	0.2.38	Biblioteca ORM - Mapeamento Objeto-Relacional
Typescript	4.3	Linguagem de Programação – *back-end* e *front-end*
Visual Studio Code	1.62	IDE - *Integrated Development Environment* (Ambiente de Desenvolvimento Integrado)
Whimsical	1.0	Criação de fluxogramas e diagrama de caso de uso

Fonte: Autoria própria (2022)

A seguir são apresentadas algumas informações importantes sobre alguns dos principais materiais que serão utilizados no desenvolvimento do projeto.

3.1.1 PostgreSQL

O PostgreSQL é um sistema gerenciador de banco de dados objeto-relacional que possui os recursos dos sistemas de banco de dados proprietários tradicionais, que possui licença *Berkeley Software Distribution* (BSD) ou seja, ela é de código aberto e seu código fonte completo está disponível para todos.

Em sua documentação oficial a equipe de desenvolvimento é descrita como desenvolvedores voluntários espalhados por todo o mundo que se comunicam pela internet. É um projeto controlado por uma comunidade e não por uma empresa privada, o que traz uma grande confiabilidade já que é remota a possibilidade de que o projeto seja encerrado em algum momento, como acontece com alguns softwares e bibliotecas mantidas por empresas privadas, gerando um gasto de recursos para migração desses sistemas.

De acordo com Milani (2008, p. 23) "o PostgreSQL é uma ótima opção entre os sistemas SGDB disponíveis, pois se trata de um sistema de grande potencial e confiabilidade com todas as características dos demais sistemas utilizados tradicionalmente no mercado". Ainda, segundo o autor, sua origem se dá a um projeto chamado *POSTGRES* patrocinado pela Universidade de Berkeley, na Califórnia (EUA), em 1986. A ideia inicial era criar um modelo para um novo sistema de armazenamento de dados com o apoio de órgãos como o *Army Research Office* (ARO) e o *National Science Foundation* (NSF) (MILANI, 2008, p. 26).

Algumas especificações interessantes descritas em sua documentação, é que o PostgreSQL não possui nenhum tipo de limitação para o tamanho dos bancos de dados, sendo limitado apenas pelo *hardware* da máquina que o hospeda. As limitações se dão a nível de tabela com um limite máximo de 32TB cada uma, a nível de linhas (registros) com 1.6TB cada uma e a nível de campos 1GB cada um.

3.1.2 Typescript

O Typescript é uma linguagem de programação fortemente tipada, que tem como propósito adicionar sintaxes adicionais para o Javascript, linguagem na qual foi construída como base, fornecendo às equipes de desenvolvimento o poder da inferência de tipos na dinamicidade do Javascript.

Desenvolvida e mantida pela empresa Microsoft, o Typescript é uma linguagem de código aberto e, sintaticamente, pode ser chamada de *superset*[1] do *ECMAScript*, que é uma linguagem de programação baseada em *scripts* padronizada pela companhia *ECMA International*. Entre as vantagens de se usar o Typescript, pode-se citar: tipagem estática, orientação a objetos, *namespaces*[2] e *decorators*[3] (TYPESCRIPT, 2022).

---

[1] Um conjunto que inclui outro conjunto. Uma extensão.

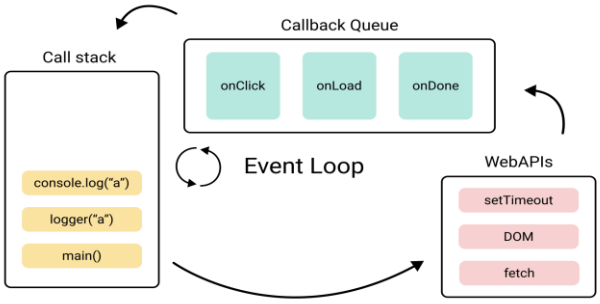
[2] Paradigma de organização de código, que deixa variáveis, funções, interfaces e classes agrupadas em um escopo único.

[3] Segundo a documentação oficial do Typescript, são um tipo especial de declaração que podem ser anexadas a uma declaração de classe, método, propriedade ou parâmetro.

3.1.3 NodeJs

O NodeJS é um *runtime*[4] para aplicações Javascript criado com base no interpretador de código aberto V8 criado pela Google, ele permite a execução de códigos Javascript fora de um navegador web com uma alta performance, fornecendo a liberdade de usar a dinamicidade da linguagem em sistemas isolados, inclusive para *back-end*. O NodeJS utiliza uma arquitetura orientada a eventos e um modelo de lidar com instruções de entrada e saída de uma forma não bloqueante, tornando a execução leve e eficiente e resolvendo problemas de tráfego intenso de rede e aplicações em tempo real, que são cada vez mais frequentes (MORAES, 2021). A forma de lidar com instruções de entrada e saída de forma não bloqueante, é possível devido a um recurso do motor V8 chamado *event-loop* (laço de eventos), que faz com que tudo seja processado em uma única *thread*[5] em vez de abrir uma nova para cada requisição. A Figura 2 demonstra de forma visual como o *event-loop* lida com as requisições.

Figura 2 - Demonstração do event-loop do motor V8



Fonte: Web Devolution (2021)

As instruções são chamadas pelas WebAPIs e enviadas para a fila de execução, ou *callback queue*, e então enviadas para *call stack* que vai resolver as instruções. Uma vez que a *call stack* tem seu espaço preenchido na memória, ocorre o evento conhecido como *stackoverflow* ou estouro de pilha que resulta em falha. Cada plataforma que roda as aplicações Javascript pode tratar a falha de formas diferentes, sendo mais comum os navegadores encerrarem o processo da aplicação.

[4] Ambiente de tempo de execução. Faz a conexão das chamadas da linguagem com a linguagem de máquina.
[5] Conhecida também como fio de execução, é a forma e a ordem de um processo ser executado pelo processador

3.1.4 Next.Js

Next.js é um *framework*[6] para desenvolvimento de aplicações web, que utiliza como base o React (biblioteca Javascript para a construção de interfaces web de forma declarativa e baseada em componentização).

Uma das funcionalidades de destaque que justificam a escolha do Next.js como o *framework* para a criação da interface do projeto é a possibilidade de se trabalhar com renderização estática pelo lado do servidor, recurso conhecido como *Server-Side Rendering* (SSR). Para isso, o Next.js utiliza um servidor NodeJS para gerar as páginas e entregar para o cliente a interface já renderizada, tirando a responsabilidade do cliente de utilizar a memória local para esta função. Outro benefício que o SSR traz é uma melhor indexação das páginas pelos motores de busca de conteúdo *on-line*, tendo em vista que o servidor já entrega todo o conteúdo pronto quando solicitado.

O Next.js também disponibiliza de forma nativa formas de armazenar em memória *cache,* algumas ou todas as páginas da aplicação, trazendo uma performance ainda maior para quem consome.

3.2 Método

Durante a concepção da ideia de um sistema web para o gerenciamento dos torneios em formato de pirâmide, foi efetuada uma pesquisa simples a fim de encontrar referenciais de funcionalidade com o projeto proposto.

Nesta pesquisa foram encontrados alguns sistemas com funcionalidades similares com a proposta deste projeto. Alguns desses sistemas eram voltados ao gerenciamento de clubes esportivos em sua totalidade, contendo controle de associados, controles financeiros, controles de estoque, diversos modelos de torneios disponíveis para criação, entre outras funcionalidades. Entretanto, nenhum dos sistemas encontrados demonstrou ter uma especialidade no controle de torneios em modelo de pirâmide, principalmente comparando com as regras do regulamento citado no Capítulo 2. Alguns dos sistemas encontrados na pesquisa, não possuíam uma interface com a visão clara do desenho do formato de pirâmide, o que neste projeto é uma das ideias principais para que se alcance um engajamento maior pelos atletas.

Entre os sistemas com finalidades similares encontrados, destacam-se:

---

[6] Em desenvolvimento de software, refere-se a um conjunto de ferramentas pré-programadas para utilização no desenvolvimento de aplicações, garantindo ganho de qualidade e produtividade para o projeto.

a) LetzPlay - Ranking e Torneios de Tênis: uma aplicação web paga, com aplicativos para Android e iOS. O foco desse sistema é o gerenciamento de clubes, academias, ligas e condomínios, por meio do gerenciamento de aulas e professores, locações de quadras, *rankings*, torneios e controle financeiro contando com a emissão de boletos para sócios. Disponibilizam *rankings* de formatos diferenciados. Alguns com modelos internacionais similares aos torneios oficiais da ATP e, também, o *ranking* em modelo de pirâmide similar ao proposto por este trabalho, porém sem uma visualização clara das posições em formato de pirâmide.

b) Leegues: um sistema web gratuito para gerenciamento de ligas, *rankings*, torneios, e competições esportivas de várias modalidades. Por ser uma plataforma mais generalista, disponibiliza os tipos mais básicos de ranqueamento de jogadores que são aplicados para diferentes modalidades esportivas, não abrangendo particularidades que o tênis pode requerer.

Até a data da conclusão deste trabalho, o controle dos torneios do modelo de pirâmide do clube Grêmio Industrial Patobranquense é realizado por meio de planilhas no Google Docs (sistema *on-line* que permite a criação de planilhas digitais), e controlado por fórmulas que calculam a pontuação dos jogadores conforme os valores dos *games* e *sets* informados para cada jogo. Esta forma de controle foi utilizada como referência para a aplicação desenvolvida. A Figura 3 mostra a interface principal deste controle, na qual pode-se observar os jogadores que estão com desafios em andamento (células com a cor de fundo vermelha) e jogadores que estão bloqueados e não podem receber desafios (células com a cor de fundo preta).

Figura 3 – Interface principal do controle do torneio por planilha

Fonte: Autoria própria (2022)

No controle por planilhas, também existe a interface na qual são registrados todos os dados dos desafios, como as datas iniciais e de prazo final, as identificações de desistências, prazo expirado e desafio recusado. A Figura 4 mostra a interface que exibe a listagem de desafios com os dados supracitados.

Figura 4 - Tabela de dados dos desafios do torneio

	JOGADORES		INFORMAÇÕES DO DESAFIO			W.O OU RECUSA			
	DESAFIADOR (1)	DESAFIADO (2)	DATA DESAFIO	PRAZO DESAFIO	DATA JOGO	W.O (1)	W.O (2)	EXPIRADO	RECUSA
3	Chicoski	Bruno	21/04/2022	28/04/2022	26/04/2022	Não	Não	Não	Não
4	Emerson Polzin	Daniel Sens	21/04/2022	28/04/2022	25/04/2022	Não	Não	Não	Não
5	Joelmir	Adriana Manarin	21/04/2022	28/04/2022	27/04/2022	Não	Não	Não	Não
6	André Babinski	Otávio	21/04/2022	28/04/2022	25/04/2022	Não	Não	Não	Não
7	Rodrigo Del Sent	Marcos Sato	21/04/2022	28/04/2022	27/04/2022	Não	Não	Não	Não
8	Dayane	Cobra	21/04/2022	28/04/2022		Não	Sim	Não	Não
9	Álvaro Piccinin	Eduardo Tonet	21/04/2022	28/04/2022	23/04/2022	Não	Não	Não	Não
10	João Pedro	Elisandro	21/04/2022	28/04/2022	28/04/2022	Não	Não	Não	Não

Fonte: Autoria própria (2022)

Nesta mesma interface são exibidas as estatísticas de cada desafio, as quais são compostas pelas pontuações de cada um dos *sets* e *tiebreak* (quando disputado) e a pontuação final de cada jogador. A Figura 5 mostra as estatísticas supracitadas.

Figura 5 - Tabela de estatísticas dos desafios

	ESTATÍSTICAS DO JOGO						PONTUAÇÃO			
	PRIMEIRO SET (1)	PRIMEIRO SET (2)	SEGUNDO SET (1)	SEGUNDO SET (2)	TIE BREAK (1)	TIE BREAK (2)	PONTOS (1)	PONTOS (2)	WIN (1)	WIN (2)
	6	3	4	**6**	**12**	10	30	20	Sim	Não
	6	2	**6**	2			40	10	Sim	Não
	4	**6**	3	**6**			10	40	Não	Sim
	6	3	**6**	4			40	10	Sim	Não
	7	5	**6**	3			40	10	Sim	Não
							0	0	Não	Não
	6	2	**6**	2			40	10	Sim	Não
	7	5	4	**6**	**10**	7	30	20	Sim	Não

Fonte: Autoria própria (2022)

Os sistemas de planilhas eletrônicas permitem que sejam criados algoritmos para automatizar certas funções e, no caso deste documento, os valores das colunas de pontuação estão automatizados tendo as outras colunas do desafio como parâmetros principais. Na Listagem 1 é possível ver o código criado para definir os valores de pontuação de uma célula.

Listagem 1 – Código de controle da pontuação por planilha

```
=IFS(
  H3 = "Sim"; -50;
  AND(J3>K3; L3>M3); 40;
  OR(AND(J3>K3; N3>O3); AND(L3>M3; N3>O3)); 30;
  OR(AND(J3>K3; L3<M3; N3<O3); AND(J3<K3; L3>M3; N3<O3)); 20;
  AND(J3<K3; L3<M3); 10; AND(J3=K3; L3=M3; N3=O3);
  0 )
```

Fonte: Autoria própria (2022)

A análise do controle por meio das planilhas agrega muito no conhecimento sobre as regras do regulamento e sobre as funcionalidades que foram desenvolvidas neste projeto.

4 RESULTADOS

Este capítulo apresenta na seção 4.1 o escopo do sistema proposto. A seção 4.2 contém a modelagem do sistema exibindo os requisitos funcionais, os requisitos não funcionais, os casos de uso, o diagrama de classes e o diagrama de entidade-relacionamento.

4.1 Escopo do Sistema

O sistema permite que a Comissão dos Tenistas do Grêmio Industrial Patobranquense possa gerenciar o torneio de modelo de pirâmide. Ademais, conta com formas de acesso para dois tipos de usuários: jogadores e administradores.

Aos jogadores, o sistema disponibiliza, de forma pública, a exibição da pirâmide com suas posições, a tabela de jogos, as pontuações e as estatísticas. Para os administradores, existe uma área restrita com acesso por meio de nome de usuário e senha onde é realizado o gerenciamento dos desafios entre os jogadores.

Os administradores podem fazer a inclusão de novos desafios, informando quais são os jogadores e a data do desafio.

Além disso, é possível ao administrador adicionar uma informação caso algum jogador desista da partida, recuse o desafio ou se o prazo expirar sem o jogo acontecer. Em todos esses casos, o sistema já calcula a pontuação dos jogadores conforme o regulamento.

No painel administrativo, o sistema permite que seja adicionada a pontuação final dos *games* de cada *set* disputado e realiza automaticamente o cálculo da pontuação conforme o regulamento.

O sistema também permite que os administradores façam o encerramento dos torneios bloqueando inclusões de novos jogos e a criação de um novo torneio, quando necessário. Aos jogadores, existe o acesso aos torneios passados para fins de consulta histórica.

O sistema possui um painel que exibe de forma pública algumas estatísticas em relação aos jogos e jogadores, são elas: desafios realizados, desafios recebidos, pontuação como desafiado, pontuação como desafiador, pontuação total, vitórias como jogador desafiado e vitórias como jogador que desafia.

4.2 Modelagem do Sistema

A apresentação da modelagem foi realizada como forma de facilitar o entendimento da proposta de intervenção no contexto abordado com a solução pretendida.

4.2.1 Requisitos Funcionais

No Quadro 2 estão elencados os Requisitos Funcionais (RF) identificados para o sistema.

Quadro 2 - Lista de Requisitos Funcionais do sistema

Identificação	Nome	Descrição
RF01	Manter torneios	Permitir que o usuário com perfil de administrador possa criar novos torneios, editar e encerrar torneios já criados. Exclusões podem ser feitas caso não exista nenhum desafio cadastrado para um torneio.
RF02	Manter jogadores	O usuário com perfil de administrador poderá cadastrar novos jogadores, editar e inativar jogadores cadastrados. Exclusões podem ser feitas caso não exista nenhum desafio cadastrado em algum torneio para o jogador.
RF03	Manter desafios	O usuário com perfil de administrador poderá registrar, editar e remover desafios realizados entre os jogadores, adicionando a data do desafio.
RF04	Manter resultados	O usuário com perfil de administrador poderá adicionar, editar e remover o resultado em *games* de cada jogador no desafio.
RF05	Manter *status*	O usuário com perfil de administrador poderá manipular o *status* de cada desafio, informando se houve desistência de algum dos jogadores, se houve recusa por parte do jogador desafiado ou se o prazo do jogo expirou.
RF06	Vincular jogadores no torneio	O administrador poderá vincular novos jogadores em um torneio em aberto.
RF07	Visualizar Torneios	Ambos atletas e administradores devem ter acesso a visualização das posições do torneio no formato de pirâmide.
RF08	Visualizar Estatísticas	O atleta deve ter acesso a visualização das estatísticas do torneio. As seguintes estatísticas serão apresentadas: desafios realizados, desafios recebidos, pontuação como desafiado, pontuação como desafiador, pontuação total, vitórias como jogador desafiado, vitórias como jogador que desafia. Tais estatísticas serão exibidas em formato de tabela para melhor acompanhamento da performance dos competidores.
RF09	Visualizar Desafios	O atleta deve ter acesso a visualização de todos os dados dos desafios de todo o torneio.

Fonte: Autoria própria (2022)

4.2.2 Requisitos Não Funcionais

O Quadro 3 apresenta os Requisitos Não Funcionais (RNF) identificados para o sistema.

Quadro 3 - Lista de Requisitos Não Funcionais do sistema

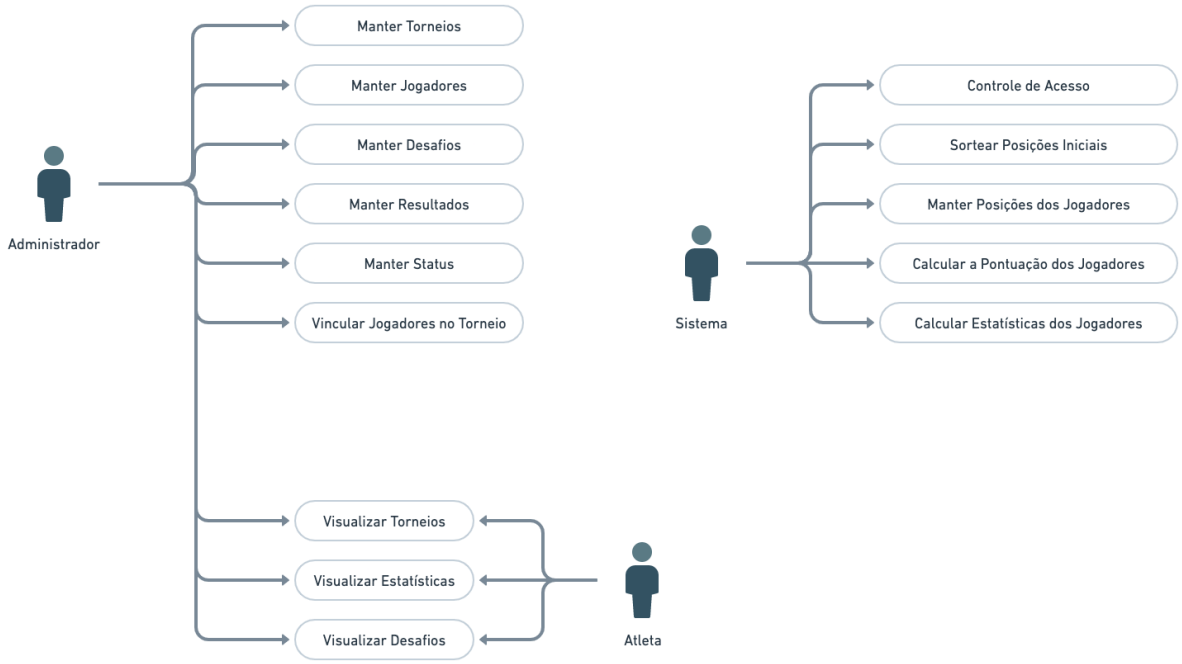
Identificação	Nome	Descrição
RNF01	Controle de Acesso	O acesso por parte dos administradores para o gerenciamento dos torneios se dará através de um nome de usuário e senha.
RNF02	Sortear posições iniciais	O sistema deve realizar o sorteio das posições iniciais do torneio com os jogadores já vinculados.
RNF03	Manter posições dos jogadores	O sistema deve fazer o controle automático das posições da pirâmide conforme o resultado dos desafios.
RNF04	Calcular a pontuação dos jogadores	Dados os resultados dos *sets* de cada jogador, o sistema deve calcular a pontuação final de cada jogador segundo descrito pelo regulamento.
RNF05	Calcular estatísticas dos jogadores	O sistema deve disponibilizar em tempo real as estatísticas dos jogadores, em formato de tabelas, já calculadas conforme o resultado dos jogos. As seguintes estatísticas serão apresentadas: desafios realizados, desafios recebidos, pontuação como desafiado, pontuação como desafiador, pontuação, total, vitórias como jogador desafiado e vitórias como jogador que desafia.

Fonte: Autoria própria (2022)

4.2.3 Casos de Uso

O diagrama de casos de uso, apresentado na Figura 6, ilustra as funcionalidades essenciais do sistema para os três tipos de atores do sistema, sendo: Atletas, Administradores e Sistema. O ator Administrador será o único a ter funcionalidades que necessitam de autenticação, como, por exemplo, o cadastro de desafios que é uma das principais funcionalidades do sistema. O ator Atleta visualiza os torneios, as estatísticas e os desafios, e o ator Sistema representa todas as funcionalidades que são automáticas dentro da aplicação e acontecem como efeito de ações do ator Administrador.

Figura 6 - Diagrama de casos de uso da aplicação



Fonte: Autoria própria (2022)

Os casos de uso da Figura 3 estão descritos a seguir. No Quadro 4 está a descrição do caso de uso denominado ‘Manter Torneios’.

Quadro 4 - Caso de uso Manter Torneios

Caso de Uso: Manter Torneios
Descrição: Realizar o cadastro de novos torneios, bem como o encerramento de torneios já finalizados, a edição de informações básicas cadastrais do torneio e a possível remoção de torneios que ainda não possuem desafios cadastrados.
Evento Iniciador: Início de um novo torneio no clube.
Ator: Administrador.
Pré-Condição: Dados básicos do torneio: descrição, data de início, data de término e jogadores participantes já cadastrados.
Pós-Condição: Torneio cadastrado com os jogadores sorteados em suas posições iniciais.

Fonte: Autoria própria (2022)

No Quadro 5 está a descrição do caso de uso denominado ‘Manter Jogadores’.

Quadro 5 - Caso de uso Manter Jogadores

Caso de Uso: Manter Jogadores
Descrição: Realizar o cadastro e a edição de jogadores e inativar jogadores que não participam mais dos torneios. Será também possível remover jogadores que não participaram de nenhum torneio previamente.
Evento Iniciador: Entrada ou saída de um jogador dos torneios.

Ator: Administrador.
Pré-Condição: Dados básicos do jogador: Nome completo, telefone para contato.
Pós-Condição: Jogador cadastrado.

Fonte: Autoria própria (2022)

No Quadro 6 está a descrição do caso de uso denominado ‘Manter Desafios’.

Quadro 6 - Caso de uso Manter Desafios

Caso de Uso: Manter Desafios.
Descrição: Criar e editar o registro de um novo desafio, informando quem foi o jogador que desafiou, qual jogador foi desafiado e qual a data que o desafio foi feito.
Evento Iniciador: Um desafio feito de um jogador para o outro.
Ator: Administrador.
Pré-Condição: Jogadores envolvidos no desafio e a data do desafio.
Pós-Condição: Desafio registrado e os jogadores sinalizados na visualização do torneio que ambos estão em desafio corrente.

Fonte: Autoria própria (2022)

No Quadro 7 está a descrição do caso de uso ‘Manter Resultados’.

Quadro 7 - Caso de uso Manter Resultados

Caso de Uso: Manter Resultados.
Descrição: Adicionar os resultados dos *games* e *sets* dos desafios após concluídos.
Evento Iniciador: Desafio concluído pelos jogadores, e sinalizado o resultado para o administrador.
Ator: Administrador.
Pré-Condição: Desafio concluído pelos jogadores.
Pós-Condição: Resultados registrados na listagem de jogos, posições atualizadas na visualização do torneio e a pontuação adequada para cada jogador atualizada na listagem de jogos e na visualização das estatísticas.

Fonte: Autoria própria (2022)

No Quadro 8 está a descrição do caso de uso ‘Manter *Status’*.

Quadro 8 - Caso de uso Manter *Status*

Caso de Uso: Manter *Status*.
Descrição: Manter o *status* de cada jogo do torneio. Os *status* disponíveis são: desistência de um dos jogadores, recusa do jogador desafiado e jogo com prazo expirado. O valor de cada *status* pode ser sim ou não.
Evento Iniciador: Desafio concluído pelos jogadores.
Ator: Administrador.
Pré-Condição: Resultado do jogo repassado ao administrador pelos jogadores.
Pós-Condição: Os resultados dos sets são todos zerados, e a pontuação atualizada para cada jogador neste jogo conforme regras do regulamento.

Fonte: Autoria própria (2022)

No Quadro 9 está a descrição do caso de uso ‘Vincular Jogadores em Torneios’.

Quadro 9 - Caso de uso Vincular Jogadores em Torneio

Caso de Uso: Vincular Jogadores em Torneios.
Descrição: Vincular um jogador já cadastrado no sistema em um torneio para que possa participar na edição do torneio.
Evento Iniciador: Jogadores iniciando ou encerrando a participação em um torneio.
Ator: Administrador.
Pré-Condição: Jogador cadastrado no sistema.
Pós-Condição: Posição do jogador adicionada ao torneio.

Fonte: Autoria própria (2022)

No Quadro 10 está a descrição do caso de uso ‘Visualizar Torneios’.

Quadro 10 - Caso de uso Visualizar Torneios

Caso de Uso: Visualizar Torneios.
Descrição: Visualizar os torneios em formato de pirâmide, tanto os em andamento quanto os que já encerraram.
Evento Iniciador: Necessidade de informações.
Ator: Administrador, Atleta.
Pré-Condição: Dados atualizados dos desafios, atletas cadastrados e vinculados a um torneio.
Pós-Condição: Torneio visualizado.

Fonte: Autoria própria (2022)

No Quadro 11 está a descrição do caso de uso ‘Visualizar Estatísticas’.

Quadro 11 - Caso de uso Visualizar Estatísticas

Caso de Uso: Visualizar Estatísticas.
Descrição: Visualizar as estatísticas provenientes dos desafios já concluídos.
Evento Iniciador: Necessidade de informações.
Ator: Administrador, Atleta.
Pré-Condição: Dados atualizados dos desafios.
Pós-Condição: Estatísticas visualizadas.

Fonte: Autoria própria (2022)

No Quadro 12 está a descrição do caso de uso ‘Visualizar Desafios’.

Quadro 12 - Caso de uso Visualizar Desafios

Caso de Uso: Visualizar Desafios.
Descrição: Visualizar os desafios, tanto pendentes quanto já encerrados, com suas pontuações quando houver, valores de *games* e *sets* já jogados, bem como os prazos iniciais e finais dos desafios.
Evento Iniciador: Necessidade de informações, acompanhamento dos prazos dos desafios.
Ator: Administrador, Atleta.
Pré-Condição: Dados atualizados dos desafios.

Pós-Condição: Desafios visualizados.

Fonte: Autoria própria (2022)

No Quadro 13 está a descrição do caso de uso ‘Controlar Acesso de Usuário’.

Quadro 13 - Caso de uso Controlar Acesso de Usuário

Caso de Uso: Controlar Acesso de Usuário.
Descrição: Controle de acesso dos usuários administradores dos torneios.
Evento Iniciador: Acesso de um usuário ao sistema.
Ator: Sistema.
Pré-Condição: Acesso de um usuário ao sistema.
Pós-Condições: Usuário com acesso liberado ao sistema com ações de gerenciamento do torneio disponíveis.

Fonte: Autoria própria (2022)

No Quadro 14 está a descrição do caso de uso ‘Sorteio das Posições Iniciais’.

Quadro 14 - Caso de uso Sorteio das Posições Iniciais

Caso de Uso: Sorteio das Posições Iniciais.
Descrição: Posições iniciais dos jogadores no torneio definidas por sorteio.
Evento Iniciador: Ao salvar um novo torneio, o sorteio é realizado automaticamente.
Ator: Sistema.
Pré-Condição: Jogadores cadastrados e vinculados ao torneio.
Pós-Condição: Posições iniciais dos jogadores sorteadas.

Fonte: Autoria própria (2022)

No Quadro 15 está a descrição do caso de uso ‘Manter Posições dos Jogadores’.

Quadro 15 - Caso de uso Manter Posições dos Jogadores

Caso de Uso: Manter Posições dos Jogadores.
Descrição: As posições dos jogadores devem ser atualizadas conforme o resultado dos desafios diretos.
Evento Iniciador: Atualização do resultado de um desafio.
Ator: Sistema.
Pré-Condição: Desafio registrado no torneio.
Pós-Condição: Posições dos jogadores atualizadas conforme parciais do desafio.

Fonte: Autoria própria (2022)

No Quadro 16 está a descrição do caso de uso ‘Calcular Pontuação dos Jogadores’.

Quadro 16 - Caso de uso Controlar Pontuação dos Jogadores

Caso de Uso: Calcular Pontuação dos Jogadores.
Descrição: Controle da pontuação dos jogadores conforme o resultado dos sets registrados.
Evento Iniciador: Dados do jogo inseridos no sistema.
Ator: Sistema.

Pré-Condição: Jogo realizado e resultados repassados ao administrador.
Pós-Condição: Pontuação dos jogadores atualizada no jogo e nas estatísticas.

Fonte: Autoria própria (2022)

No Quadro 17 está a descrição do caso de uso ‘Calcular Estatísticas dos Jogadores’.

Quadro 17 - Caso de uso Calcular Estatísticas dos Jogadores

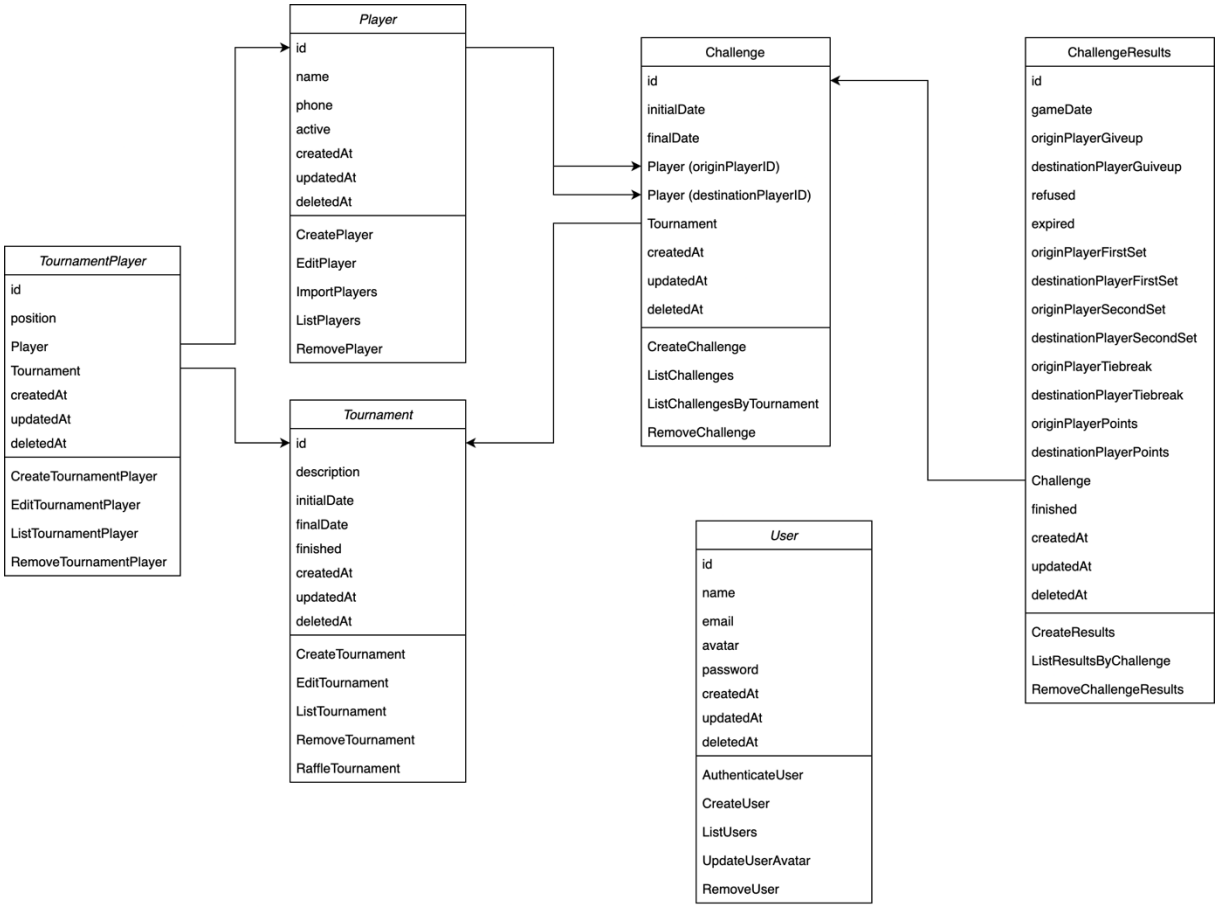
Caso de Uso: Calcular Estatísticas dos Jogadores.
Descrição: Cálculo das estatísticas dos jogadores após o resultado dos sets adicionados no jogo no sistema.
Evento Iniciador: Jogo executado e resultados repassados ao administrador.
Ator: Sistema.
Pré-Condição: Desafio realizado entre dois jogadores e repassado ao administrador.
Pós-Condição: Estatísticas individuais atualizadas.

Fonte: Autoria própria (2022)

4.2.4 Diagrama de Classes

A Figura 7 ilustra o diagrama de classes do sistema. Tanto as classes quanto os métodos de cada uma delas já estão descritos no diagrama com a nomenclatura correta na qual foram implementados. Também é possível observar as ligações chave entre as classes desenvolvidas.

Figura 7 - Diagrama de classes com métodos



Fonte: Autoria própria (2022)

As classes do diagrama da Figura 7, estão descritas a seguir nos quadros numerados de 18 a 23.

A classe User, descrita no Quadro 18, irá registrar todos os dados necessários para que os administradores tenham acesso ao sistema. Cada administrador terá um registro desta classe, tendo os campos e-mail e senha como as informações de acesso.

Quadro 18 - Descrição da classe User

Identificação:	User
Descrição:	Registro de usuários para acesso ao sistema.
Atributos:	- id (string): identificação única do usuário; - nome (string): nome de identificação do usuário; - email: (string): email para acesso do usuário no sistema; - avatar: (string): imagem do usuário em *base64*; - password: (string): armazenamento da senha criptografada de acesso ao sistema; - createdAt: (timestamp): data e hora de criação do registro;

	- updatedAt: (timestamp): data e hora de atualização do registro; - deletedAt: (timestamp): data e hora de remoção lógica do registro;
Métodos:	AuthenticateUser(), CreateUser(), ListUser(), UpdateUserAvatar(), RemoveUser()
Requisitos Envolvidos:	RF01, RF02, RF03, RF04, RF05, RF06, RF07, RF08, RF09, RNF01

Fonte: Autoria própria (2022)

A classe *Tournament,* apresentada no Quadro 19, é a responsável pelos métodos de manutenção dos torneios na aplicação. Cada torneio vai ter datas de início e fim e que são parâmetros fundamentais para a validação da inclusão de novos jogos.

Quadro 19 - Descrição da classe Tournament

Identificação:	*Tournament*
Descrição:	Registro dos torneios de jogos de tênis
Atributos:	- id (string): identificação única do torneio; - description (string): descrição do torneio; - initialDate: (timestamp): data inicial do torneio; - finalDate: (timestamp): prazo final do torneio; - finished: (boolean): indicador de torneio encerrado ou aberto; - createdAt: (timestamp): data e hora de criação do registro; - updatedAt: (timestamp): data e hora de atualização do registro; - deletedAt: (timestamp): data e hora de remoção lógica do registro;
Métodos:	CreateTournament(), EditTournament(), ListTournament(), RemoveTournament(), RaffleTournament()
Requisitos Envolvidos:	RF01, RF03, RF04, RF05, RF06, RF07, RF08, RF09, RNF02, RNF03, RNF04, RNF05

Fonte: Autoria própria (2022)

A classe *TournamentPlayer,* apresentada no Quadro 20, é a responsável pelo registro dos jogadores que estão participando do torneio, bem como suas posições no mesmo.

Quadro 20 - Descrição da classe TournamentPlayer

Identificação:	*TournamentPlayer*
Descrição:	Registro dos jogadores participantes de um torneio
Atributos:	- id (string): identificação única do jogador no torneio; - position (integer): posição do jogador no torneio; - Player: (string): identificação única do jogador; - Tournament: (string): identificação única do torneio; - createdAt: (timestamp): data e hora de criação do registro; - updatedAt: (timestamp): data e hora de atualização do registro; - deletedAt: (timestamp): data e hora de remoção lógica do registro;
Métodos:	CreateTournamentPlayer(), EditTournamentPlayer(), ListTournamentPlayer(), RemoveTournamentPlayer()

Requisitos Envolvidos:	RF03, RF04, RF05, RF06, RF07, RF08, RF09, RNF01, RNF02, RNF03, RNF04, RNF05

Fonte: Autoria própria (2022)

A classe *Player,* apresentada no Quadro 21, controla os parâmetros básicos para os registros dos jogadores no sistema, como o nome e o telefone para contato, uma informação útil para que os jogos possam ser marcados entre os atletas.

Quadro 21 - Descrição da classe Player

Identificação:	*Player*
Descrição:	Registro dos jogadores que participaram dos torneios
Atributos:	- id (string): identificação única do jogador; - name (string): descrição/nome do jogador; - phone: (integer): telefone para contato do jogador; - active: (boolean): indicador de jogador ativo ou inativo; - createdAt: (timestamp): data e hora de criação do registro; - updatedAt: (timestamp): data e hora de atualização do registro; - deletedAt: (timestamp): data e hora de remoção lógica do registro;
Métodos:	CreatePlayer(), EditPlayer(), ImportPlayer(), ListPlayer(), RemovePlayer();
Requisitos Envolvidos:	RF02, RF03, RF04, RF05, RF06, RF07, RF08, RF09, RNF01, RNF02, RNF03, RNF04, RNF05

Fonte: Autoria própria (2022)

A classe *Challenge,* apresentada no Quadro 22, é a principal do sistema, já que todos os dados básicos dos jogos do torneio são produtos desta classe.

Quadro 22 - Descrição da classe Challenge

Identificação:	*Challenge*
Descrição:	Registro dos desafios entre os jogadores
Atributos:	- id (string): identificação única do desafio; - initialDate (timestamp): data inicial do desafio; - finalDate (timestamp): data do prazo final do desafio; - Player (originPlayerID) (string): identificação do jogador que originou o desafio; - Player (destinationPlayerID) (string): identificação do jogador no qual é destinado o desafio; - Tournament (string): identificação do torneio; - createdAt: (timestamp): data e hora de criação do registro; - updatedAt: (timestamp): data e hora de atualização do registro; - deletedAt: (timestamp): data e hora de remoção lógica do registro;
Métodos:	CreateChallenge(), ListChallenge(), ListChallengesByTournament(), RemoveChallenge()

Requisitos Envolvidos:	RF03, RF04, RF05, RF07, RF08, RF09, RNF01, RNF02, RNF03, RNF04, RNF05

Fonte: Autoria própria (2022)

A classe *ChallengeResults,* apresentada no Quadro 13, contém todas as informações adicionais dos resultados dos jogos do torneio.

Quadro 23 - Descrição da classe ChallengeResults

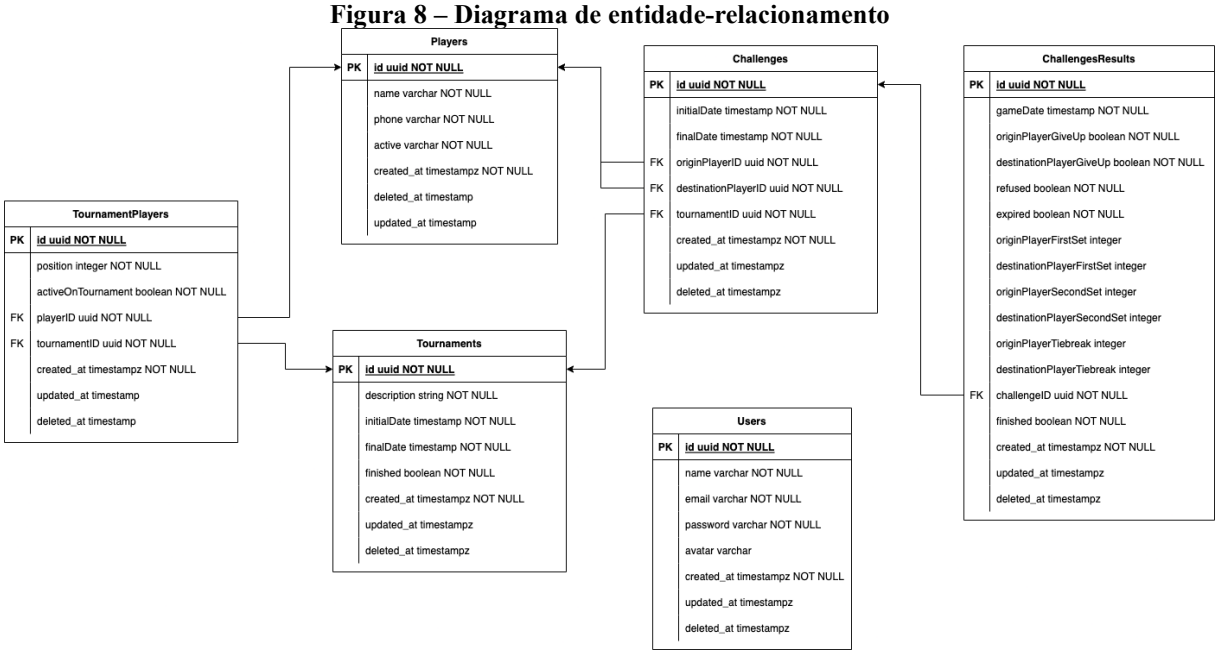
Identificação:	*ChallengeResults*
Descrição:	Registro dos resultados dos desafios entre os jogadores
Atributos:	- id (string): identificação única do resultado de desafio; - gameDate (timestamp): data da realização do jogo; - originPlayerGiveup (boolean): informação se o jogador desafiador desistiu; - destinationPlayerGiveup (boolean): informação se o jogador desafiado desistiu; - refused (boolean): informação se o jogador desafiado recusou o desafio; - expired (boolean): informação se o jogo expirou sem acontecer; - originPlayerFirstSet (integer): valor do primeiro *set* para o jogador desafiador - destinationPlayerFirstSet (integer): valor do primeiro *set* para o jogador desafiado - originPlayerSecondSet (integer): valor do segundo *set* para o jogador desafiador - destinationPlayerSecondSet (integer): valor do segundo *set* para o jogador desafiado - originPlayerTiebreak (integer): valor do terceiro *set* para o jogador desafiador (*tiebreak*) - destinationPlayerTiebreak (integer): valor do terceiro *set* para o jogador desafiado (*tiebreak*) - originPlayerPoints (integer): pontuação final para o jogador desafiador - destinationPlayerPoints (integer): pontuação final para o jogador desafiado - Challenge (string): identificação única do desafio; - finished (boolean): identificação se o desafio está encerrado; - createdAt: (timestamp): data e hora de criação do registro; - updatedAt: (timestamp): data e hora de atualização do registro; - deletedAt: (timestamp): data e hora de remoção lógica do registro;
Métodos:	CreateResults(), ListResultsByChallenge(), RemoveChallengeResults()
Requisitos Envolvidos:	RF04, RF05, RF07, RF08, RF09, RNF01, RNF02, RNF03, RNF04, RNF05

Fonte: Autoria própria (2022)

4.2.5 Diagrama Entidade-Relacionamento

A Figura 8 ilustra o diagrama de entidade-relacionamento que representa o banco de dados para a manutenção das informações do sistema proposto, exibindo todas as tabelas que foram criadas e as relações que possuem.

Figura 8 – Diagrama de entidade-relacionamento



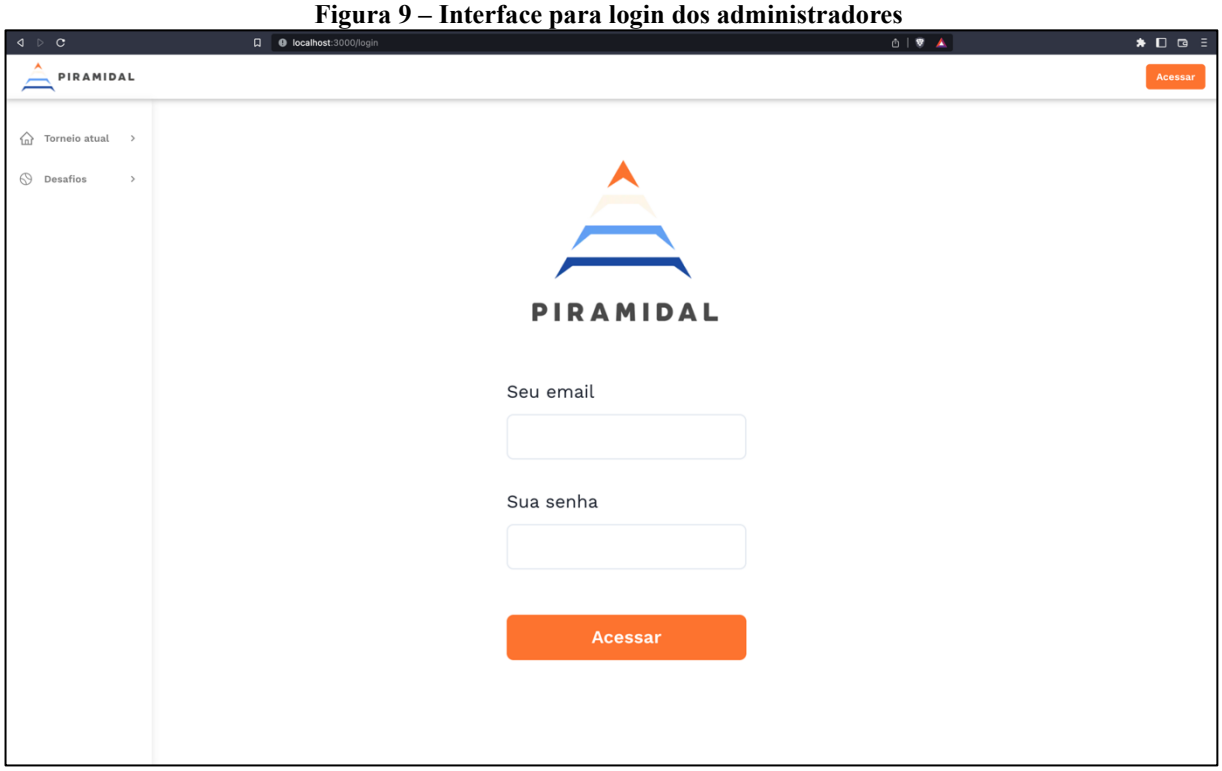
Fonte: Autoria própria (2022)

A entidade denominada "*Users*" vai armazenar todos os usuários que possuem acesso ao sistema e poderão gerenciar os torneios e desafios.A entidade denominada "*Tournaments*" armazena as informações básicas dos torneios criados pelos usuários, com uma descrição para identificação, a data inicial e a data final que é parâmetro para bloquear a inclusão de novos jogos. A entidade denominada "*TournamentsPlayers*" armazena a relação de todos os jogadores que participam dos torneios, e quais suas posições nestes. A entidade denominada "*Players*" vai armazenar apenas o nome de identificação do jogador, e seu número de telefone que pode ser útil durante os desafios. A entidade denominada "Challenges" é a principal do projeto. Serão armazenados quais são os jogadores envolvidos em um desafio, a qual torneio esse desafio pertence e o prazo limite. Por fim, na entidade denominada "*ChallengesResults*" são armazenadas as informações sobre os resultados de um desafio, incluindo a data do jogo, se houve desistência de algum jogador, quais são os resultados dos *sets* e qual a pontuação que vai ser calculada automaticamente pelo sistema.

4.3 Apresentação do Sistema

O acesso ao sistema ocorrerá pelo fornecimento das credenciais para os usuários com perfil de administrador, tendo em vista que neste primeiro momento não haverá uma interface para cadastro dos usuários. O acesso será realizado pela interface de *login* conforme exibida na Figura 9.

Figura 9 – Interface para login dos administradores

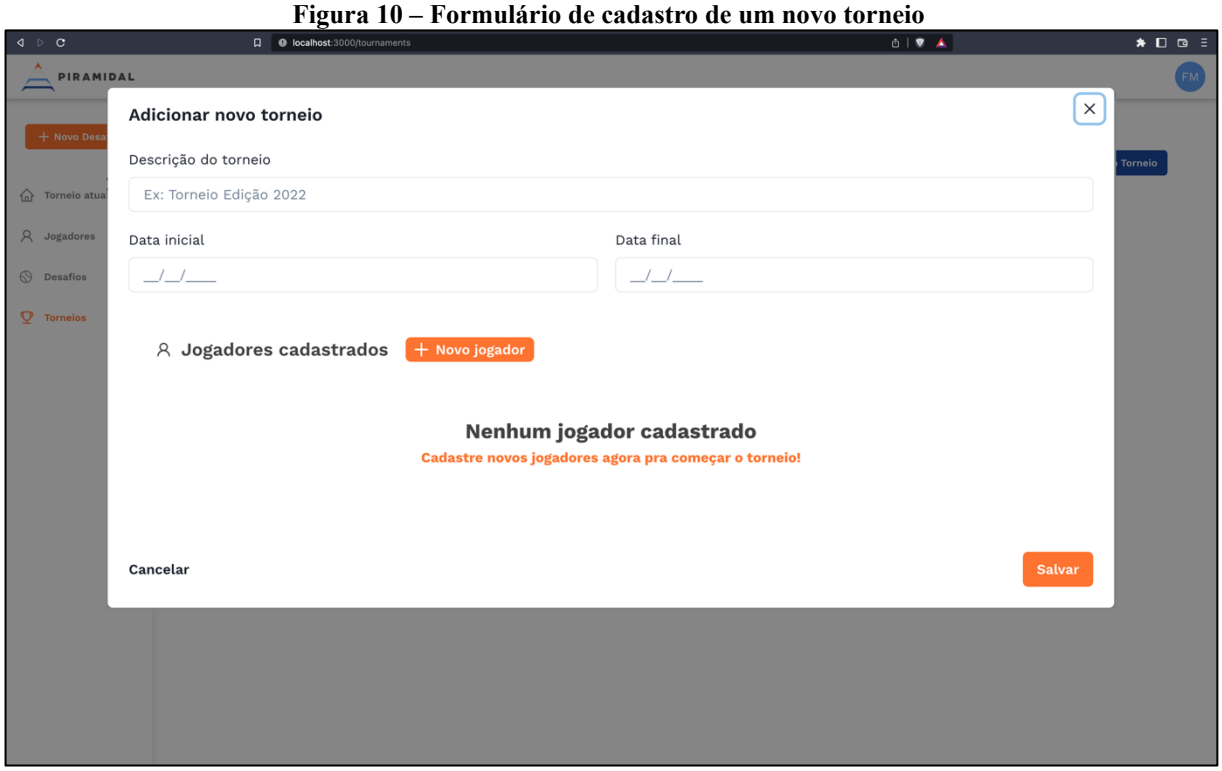


Fonte: Autoria própria (2022)

A partir do acesso com suas credenciais, o administrador poderá realizar o cadastro dos torneios na aplicação. Ao efetuar esses cadastros, é possível cadastrar os jogadores que vão participar do torneio, sendo possível alterar essas informações a qualquer momento.

A Figura 10 mostra o formulário de cadastro de um novo torneio e a Figura 11 mostra a tela que permite a inclusão de novos jogadores para o torneio.

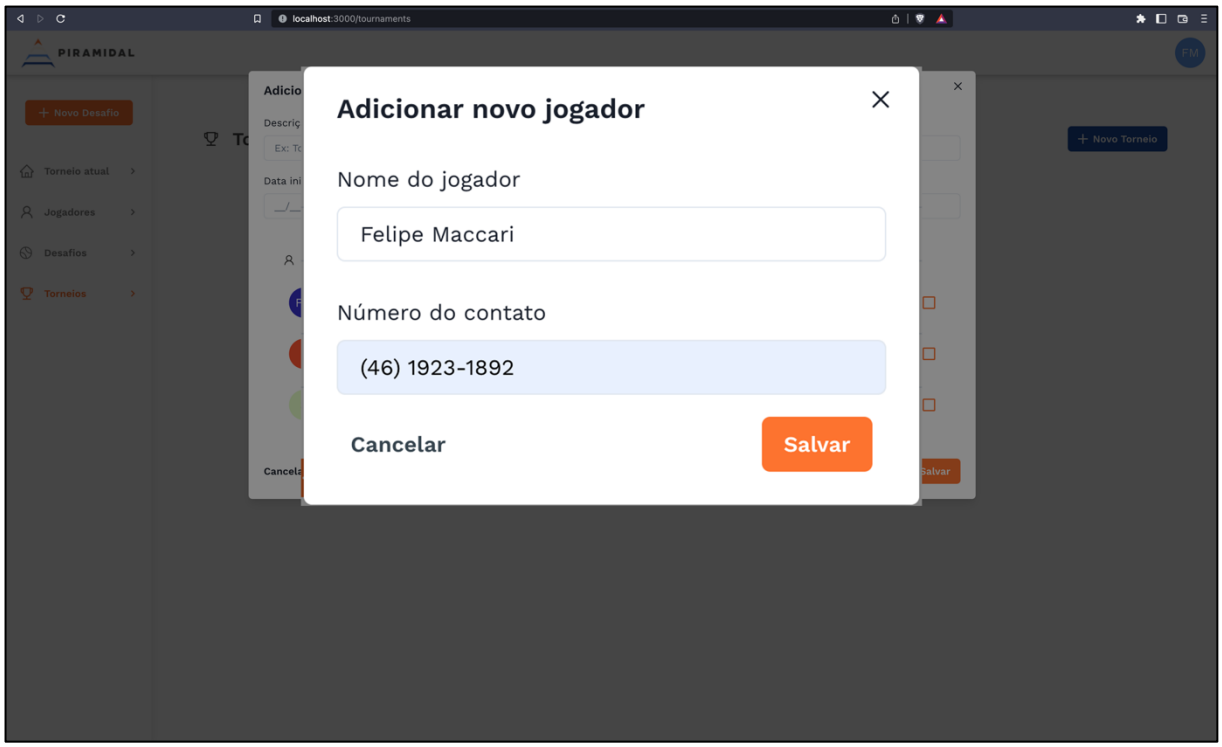
Figura 10 – Formulário de cadastro de um novo torneio



Fonte: Autoria própria (2022)

Como facilidade na usabilidade do projeto, é possível cadastrar novos jogadores por meio do botão “Novo Jogador” na interface de criação do torneio. A Figura 11 exibe a tela de cadastro de novos jogadores.

Figura 11 – Interface de cadastro de novos jogadores

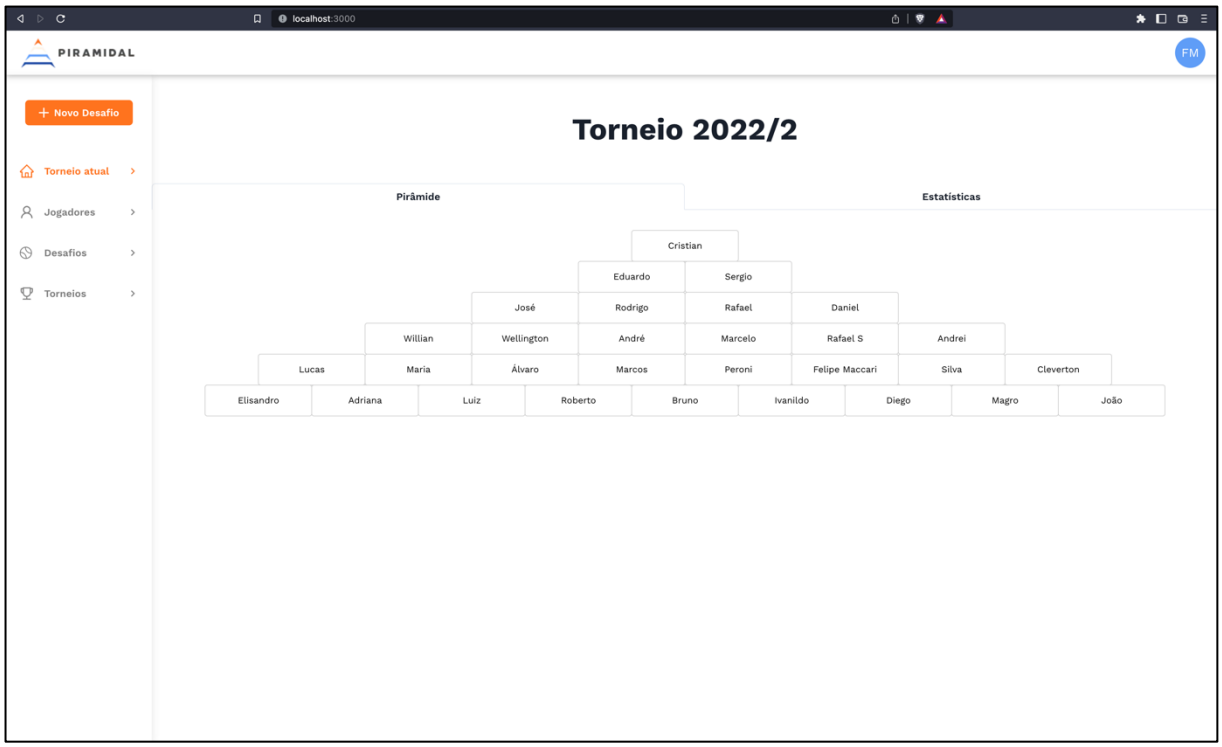


Fonte: Autoria própria (2022)

No momento em que o torneio é criado, o sistema realiza o sorteio das posições dos jogadores no formato de pirâmide e cria as tabelas de estatísticas para acompanhamento das pontuações. Com isso na tela inicial do sistema já é possível acompanhar o torneio. Essa interface de exibição é pública para todos os jogadores e não necessita nenhum tipo de autenticação.

A Figura 12 mostra a tela inicial do sistema com as posições dos jogadores já sorteadas em seu estado inicial de torneio, enquanto nenhum desafio foi realizado, por consequência sem marcação visual em nenhum jogador. Essa tela é dividida em duas abas, sendo, “Pirâmide” e “Estatísticas” que, ao serem selecionadas, exibem as informações correspondentes à pirâmide e às estatísticas de cada jogador.

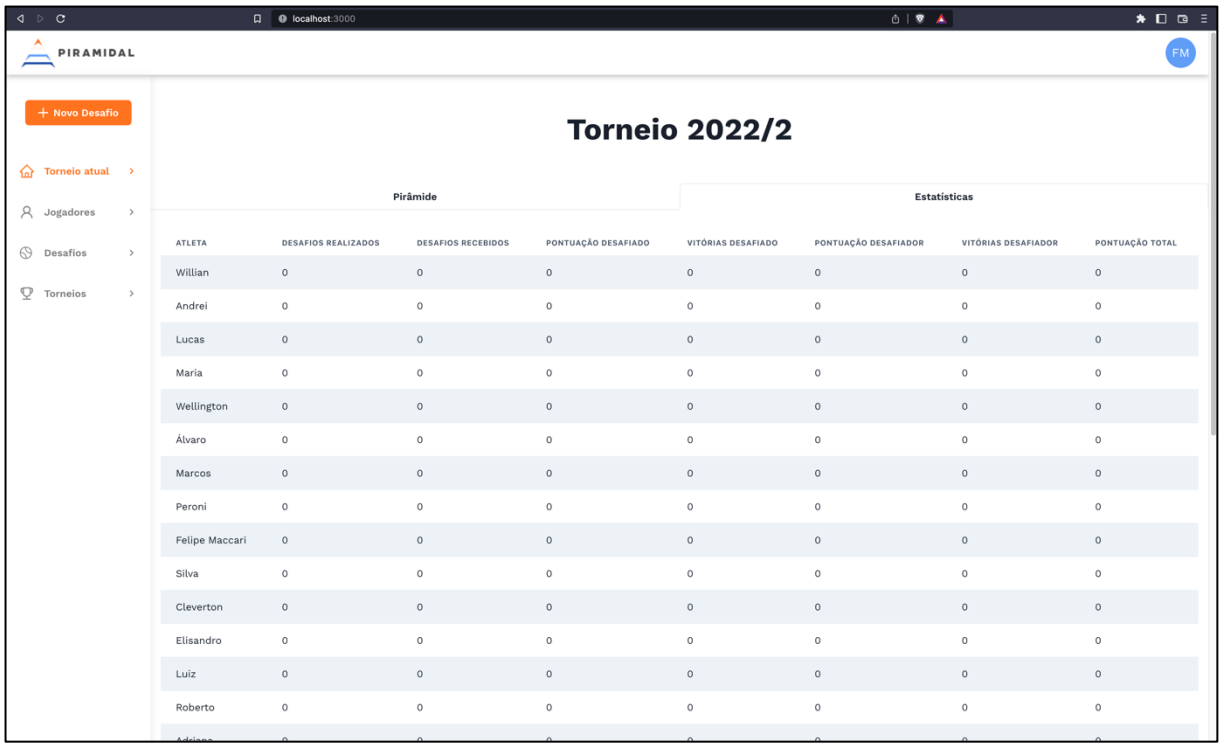
Figura 12 – Exibição do torneio em formato de pirâmide



Fonte: Autoria própria (2022)

Na Figura 13 é possível visualizar as estatísticas de cada jogador correspondente aos desafios realizados e recebidos, as pontuações do jogador desafiado e do desafiador e a pontuação total.

Figura 13 – Exibição das estatísticas iniciais do torneio vigente



Fonte: Autoria própria (2022)

É possível realizar a manutenção de jogadores cadastrados ou cadastrar novos jogadores, mesmo que não vinculados em um torneio específico, por meio de uma interface de gerenciamento específica, apresentada na Figura 14.

Figura 14 – Tela de listagem dos jogadores cadastrados

Fonte: Autoria própria (2022)

O sistema disponibiliza também a interface para criação dos desafios e o registro de seus respectivos resultados sempre que necessário.

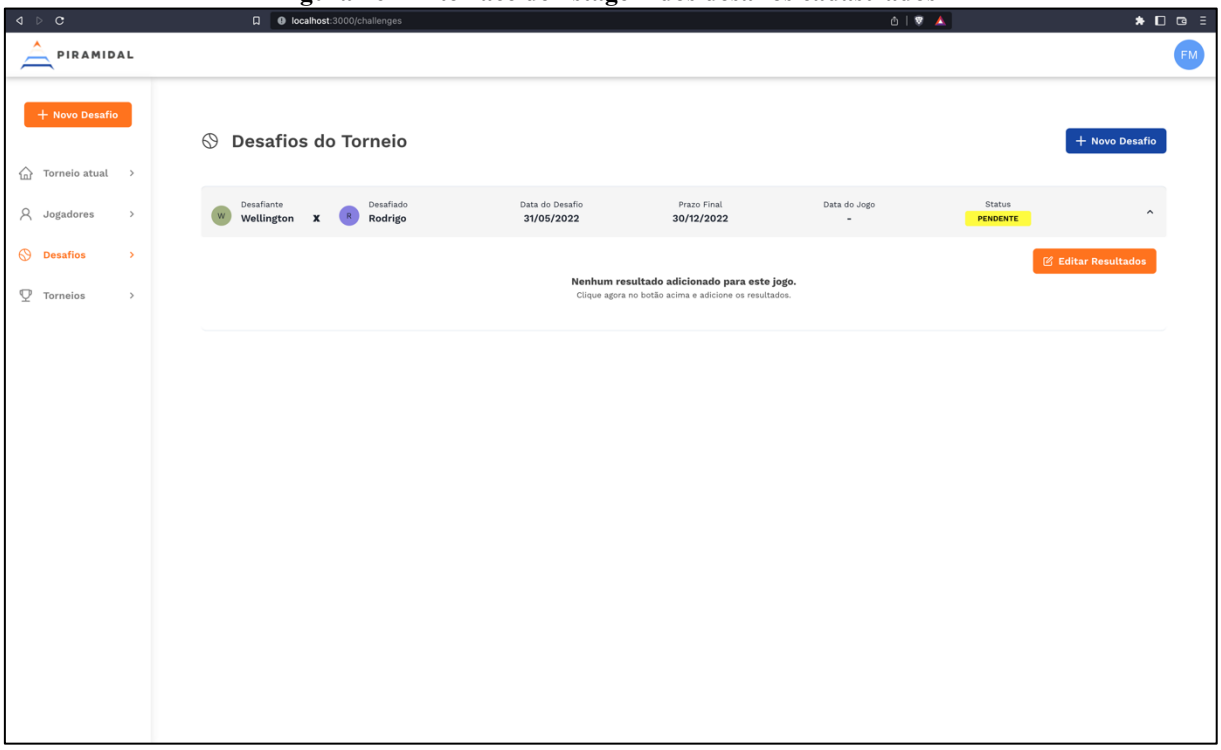
Para cadastrar um novo desafio é necessário informar os dados referentes à data de início, o prazo máximo que esse desafio pode ocorrer e os jogadores que irão participar do desafio, primeiro o jogador que realizou o desafio e o jogador que foi desafiado. A Figura 15 mostra a tela de cadastro de um novo desafio.

Figura 15 – Interface de cadastro de um novo desafio

Fonte: Autoria própria (2022)

Após a inclusão, o desafio é exibido na listagem conforme mostra a Figura 16. Como aspectos importantes pode-se notar a exibição do prazo inicial, prazo final, data de jogo (quando houver) e, também, o *status* do desafio, que pode ter três valores, sendo: pendente, concluído ou expirado.

Figura 16 – Interface de listagem dos desafios cadastrados

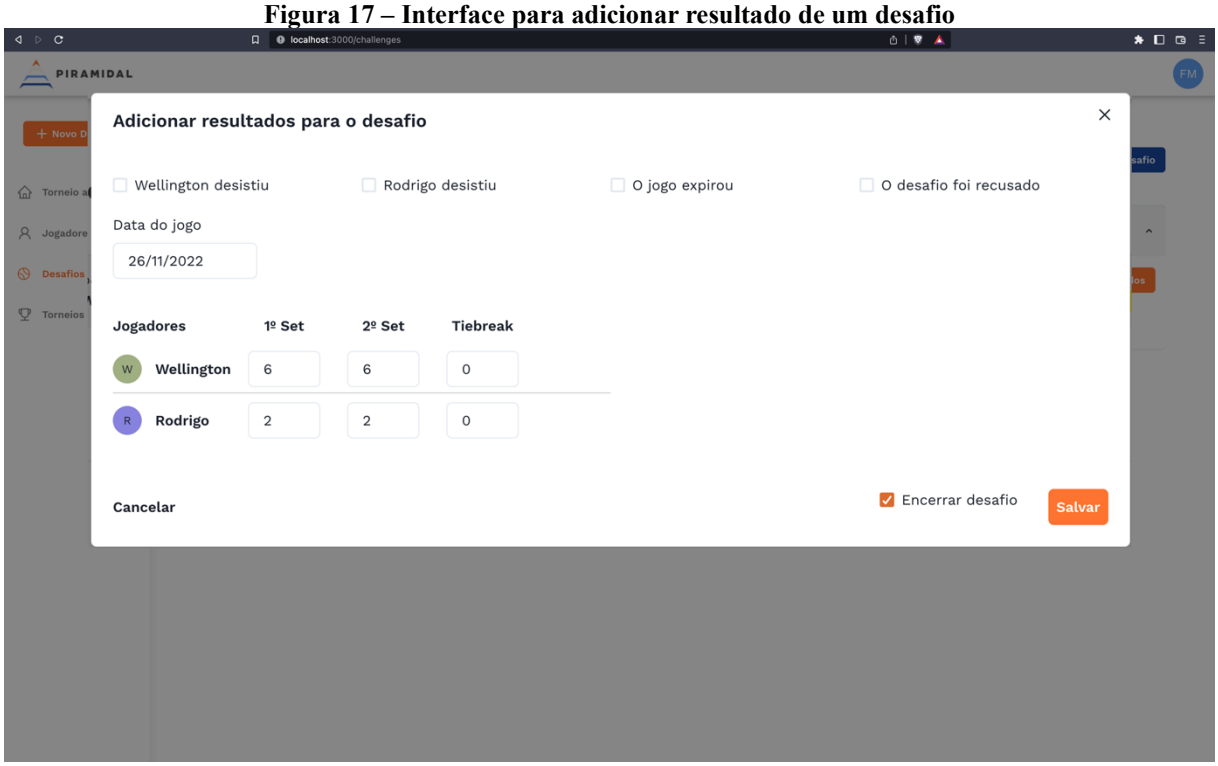


Fonte: Autoria própria (2022)

Na tela apresentada na Figura 16, nota-se que o desafio cadastrado está como pendente, pois é o primeiro *status* exibido após o seu cadastro. Além disso, quando o componente de listagem expande, é possível visualizar a mensagem que informa que ainda não existem resultados cadastrados e o botão de ação para o registro dos mesmos.

Quando os jogadores finalizam o desafio e repassam a informação ao administrador, ele poderá incluir os resultados informando se o jogo não ocorreu por algum dos motivos possíveis, caso contrário pode preencher os resultados dos *sets* disputados e a data do jogo, como mostra a tela apresentada na Figura 17.

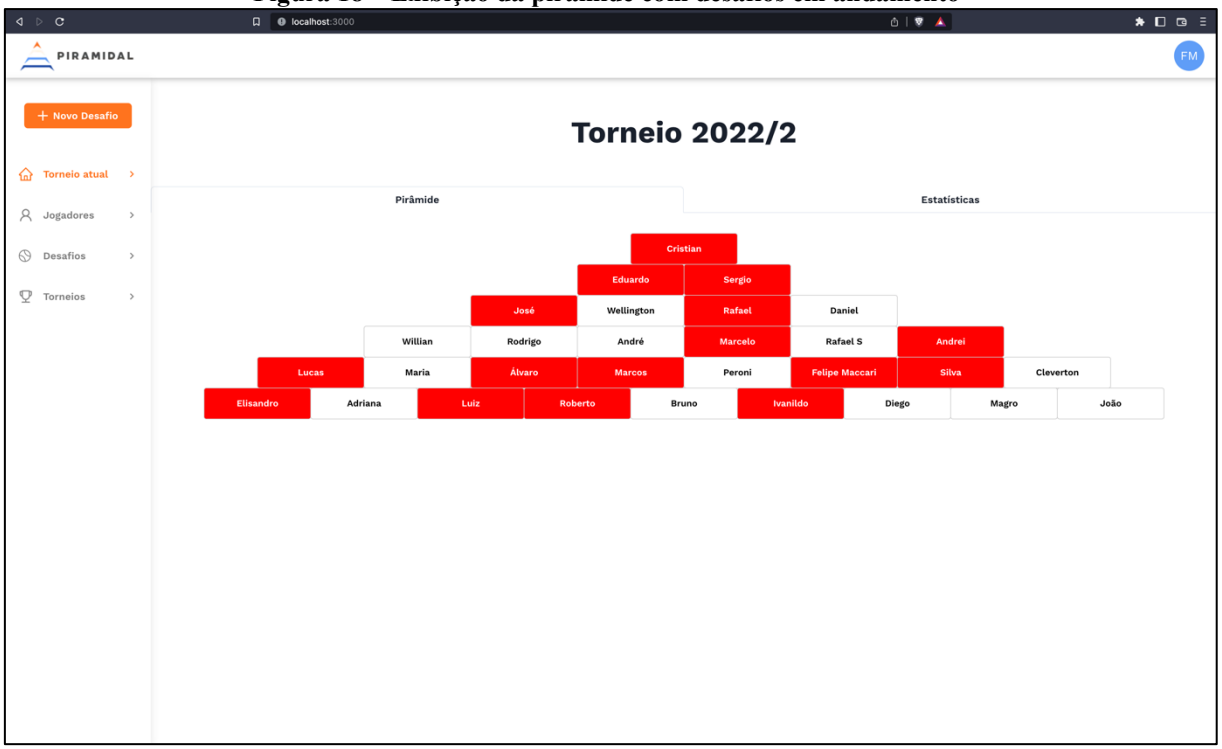
Figura 17 – Interface para adicionar resultado de um desafio



Fonte: Autoria própria (2022)

Enquanto os desafios estão pendentes, o sistema exibirá os jogadores envolvidos com um destaque em cor vermelha na exibição da pirâmide do torneio, como mostra a Figura 18, com uma série de desafios cadastrados.

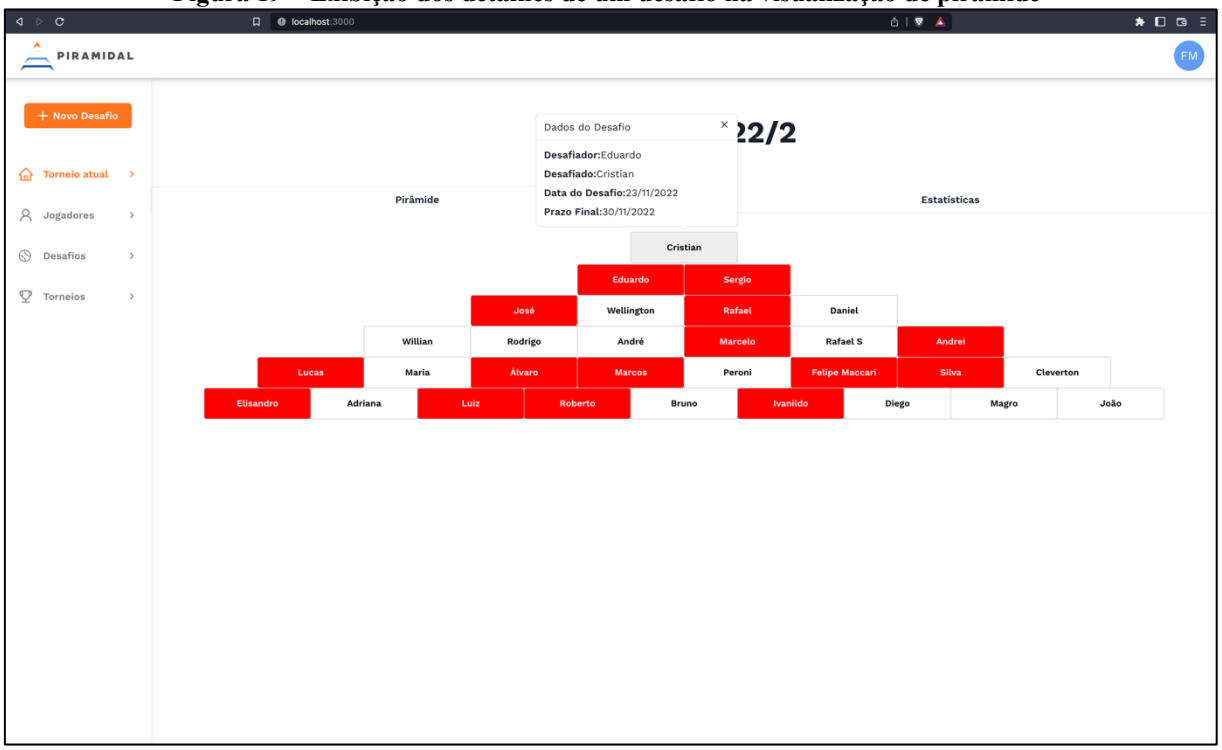
Figura 18 – Exibição da pirâmide com desafios em andamento



Fonte: Autoria própria (2022)

Para facilitar a leitura dos dados dos desafios do torneio, é possível visualizar informações sobre o desafio em que cada jogador está envolvido ao passar o mouse em cima do nome do jogador, conforme mostra a Figura 19.

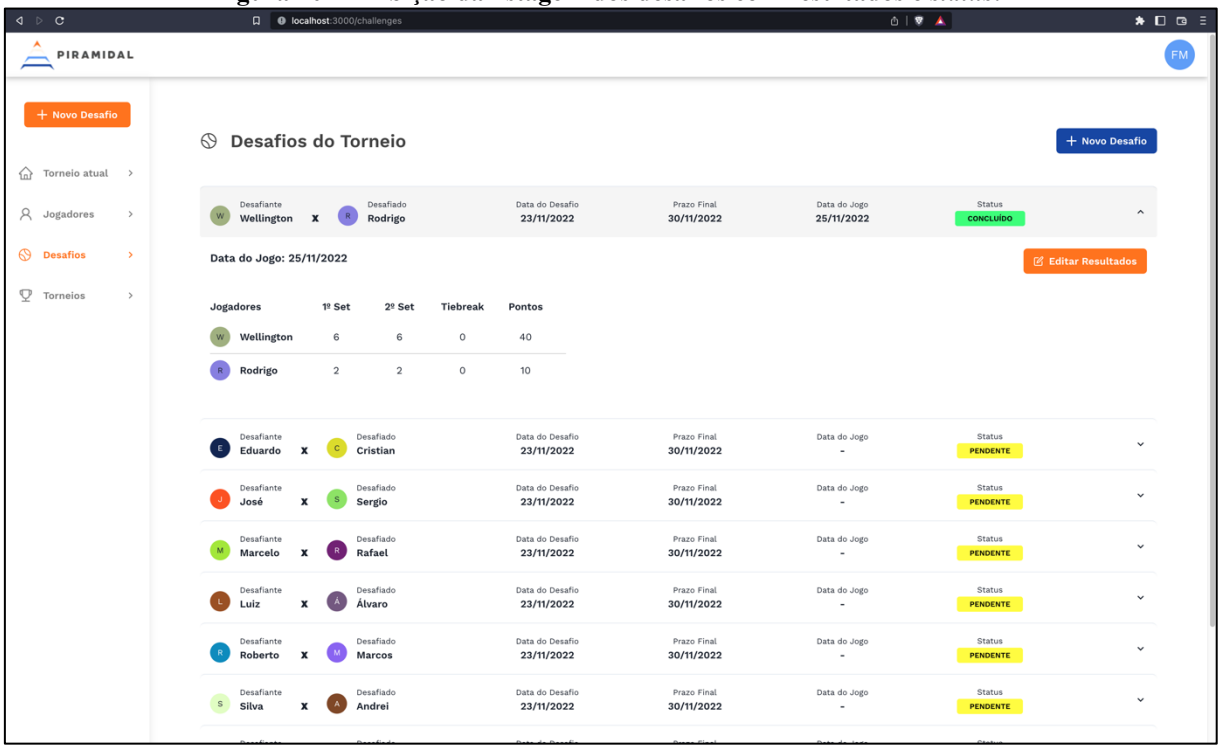
Figura 19 – Exibição dos detalhes de um desafio na visualização de pirâmide



Fonte: Autoria própria (2022)

Os usuários do sistema, sempre que necessário, poderão verificar o resultado e *status* dos desafios do torneio por meio da tela de desafios apresentada na Figura 20.

Figura 20 – Exibição da listagem dos desafios com resultados e *status*.



Fonte: Autoria própria (2022)

A implementação das telas apresentadas traz uma série de melhorias visuais e de usabilidade tanto para os administradores terem maior segurança e facilidade ao gerenciar os jogos, quanto para os jogadores que poderão ter um melhor acompanhamento da classificação e de seu desempenho ao longo do torneio, visando motivar a prática esportiva.

4.4 Implementação do Sistema

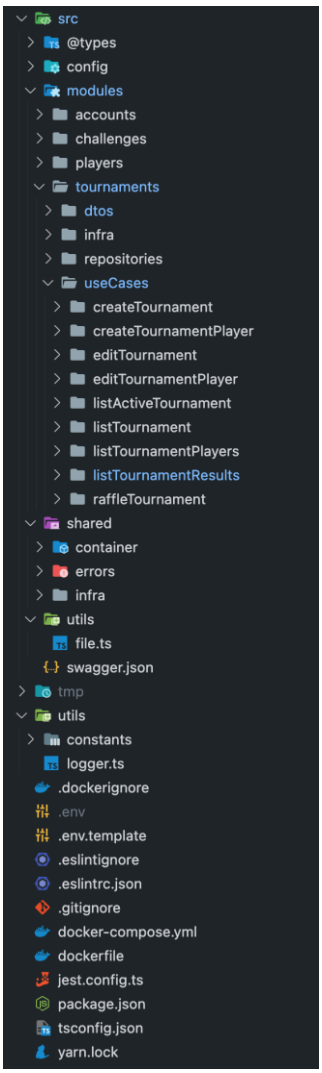
A estrutura do projeto se divide entre cliente e servidor, executados em projetos diferentes com portas separadas, podendo ser hospedados em servidores com arquiteturas únicas e desassociadas.

4.4.1 Servidor

Para a criação do servidor foi utilizado o *Express*, que é um *framework* para desenvolvimento de serviços de *back-end*, e a linguagem utilizada foi o TypeScript.

A Figura 21 demonstra a estrutura de pastas do projeto no *back-end*.

Figura 21– Estruturas de pastas do *backend*



Fonte: Autoria própria (2022)

Desta estrutura, é interessante destacar dois diretórios principais: *modules* e *shared*. A pasta *modules* contém as pastas com os contextos das rotinas do sistema e, dentro de cada contexto, estão os arquivos com todo o gerenciamento das requisições. Esse gerenciamento é separado nos seguintes diretórios:

– dtos - Interfaces declarativas dos objetos de transferência de dados.

– infra - Estrutura que contém as entidades e os repositórios que foram criados com dependência na biblioteca de ORM: Typeorm. Assim, caso seja necessária a substituição desta ferramenta, o código é facilmente desacoplado.

– *repositories* - Interfaces para que seja obrigatória a implementação de métodos pelos repositórios.

– *useCases* - Separação dos *controllers* e *services* em diretórios por função específica do sistema. Os *controllers* são as funções responsáveis por receber a requisição, recuperar qualquer parâmetro e enviar ao *use case* que contém todas as regras de negócio de cada módulo. O *use case* pode ser também chamado de *service* em alguns padrões de projetos comuns que fazem utilização da arquitetura API REST.

A pasta *shared* contém os arquivos que são compartilhados entre todos os módulos como a configuração da injeção de dependências, tratamento de erros, arquivos para manipulação do banco de dados, configuração de rotas e *middlewares*.

Para a configuração das rotas do projeto foi criado um arquivo de direcionamento das requisições agrupadas por módulo. A Listagem 2 exibe esses agrupamentos.

Listagem 2 – Configuração de agrupamento de rotas

```
import { Router } from "express";
import authenticateRoutes from "./authenticate.routes";
import challengesRoutes from "./challenges.routes";
import playersRoutes from "./players.routes";
import tournamentsRoutes from "./tournaments.routes";
import usersRoutes from "./users.routes";

const router = Router();

router.use("/players", playersRoutes);
router.use("/users", usersRoutes);
router.use("/challenges", challengesRoutes);
router.use("/tournaments", tournamentsRoutes);
router.use(authenticateRoutes);

export default router;
```

Fonte: Autoria própria (2022)

Cada um dos módulos importados e repassados como o segundo parâmetro da declaração da rota contém todas sub rotas e para qual *controller* cada requisição é enviada. A Listagem 3 exibe o exemplo de todas as rotas relacionadas com a rota principal das requisições dos torneios, a rota ‘*/tournaments’*.

Listagem 3 – Configuração de sub rotas do modulo *Tournament*

```
import { Router } from "express";
import CreateTournamentController from
"@modules/tournaments/useCases/createTournament/CreateTournamentController";
import CreateTournamentPlayerController from
"@modules/tournaments/useCases/createTournamentPlayer/CreateTournamentPlayerCont
roller";
import EditTournamentController from
"@modules/tournaments/useCases/editTournament/EditTournamentController";
```

```
import EditTournamentPlayerController from
"@modules/tournaments/useCases/editTournamentPlayer/EditTournamentPlayerControll
er";
import ListActiveTournamentController from
"@modules/tournaments/useCases/listActiveTournament/ListActiveTournamentControll
er";
import ListTournamentController from
"@modules/tournaments/useCases/listTournament/ListTournamentController";
import ListTournamentPlayersController from
"@modules/tournaments/useCases/listTournamentPlayers/ListTournamentPlayersContro
ller";
import ListTournamentResultsController from
"@modules/tournaments/useCases/listTournamentResults/ListTournamentResultsContro
ller";
import RaffleTournamentController from
"@modules/tournaments/useCases/raffleTournament/RaffleTournamentController";
import ensureAuthenticated from
"@shared/infra/http/middlewares/ensureAuthenticated";

const tournamentsRoutes = Router();

const createTournamentController = new CreateTournamentController();
const createTournamentPlayerController = new CreateTournamentPlayerController();
const editTournamentController = new EditTournamentController();
const editTournamentPlayerController = new EditTournamentPlayerController();
const listTournamentController = new ListTournamentController();
const listTournamentPlayersController = new ListTournamentPlayersController();
const raffleTournamentController = new RaffleTournamentController();
const listActiveTournamentController = new ListActiveTournamentController();
const listTournamentResultsController = new ListTournamentResultsController();

tournamentsRoutes.post("/", ensureAuthenticated,
createTournamentController.handle);
tournamentsRoutes.post( "/:tournamentID/raffle", ensureAuthenticated,
raffleTournamentController.handle);
tournamentsRoutes.post( "/player/:tournamentID", ensureAuthenticated,
createTournamentPlayerController.handle);
tournamentsRoutes.get("/", listTournamentController.handle);
tournamentsRoutes.get("/active", listActiveTournamentController.handle);
tournamentsRoutes.get("/:tournamentID/results",
listTournamentResultsController.handle);
tournamentsRoutes.get("/:tournamentID", listTournamentPlayersController.handle);
tournamentsRoutes.put("/:tournamentID", ensureAuthenticated,
editTournamentController.handle);
tournamentsRoutes.put("/player/:tournamentPlayerID", ensureAuthenticated,
editTournamentPlayerController.handle);

export default tournamentsRoutes;
```

Fonte: Autoria própria (2022)

Nota-se que algumas rotas possuem além do primeiro parâmetro que descreve a sub rota, e o último que é a função *handle* de cada *controller* (separados por *use cases* dentro do módulo), uma função de *middleware*, que intercepta a rota e é executada antes da requisição chegar até o *controller*. Neste projeto foi desenvolvido apenas um *middleware* que é chamado

de *ensureAuthenticated* e tem o papel de verificar se a requisição possui no seu cabeçalho a autorização de sessão do usuário. Caso não possua, essa função encerra o ciclo da requisição retornando uma mensagem de erro ao cliente, caso contrário, o fluxo é executado normalmente. A Listagem 4 mostra o código do *middleware* supracitado.

Listagem 4 – *Middleware* de rotas autenticadas

```
import { NextFunction, Request, Response } from "express";
import { verify } from "jsonwebtoken";
import UsersRepository from
"@modules/accounts/infra/typeorm/repositories/UsersRepository";
import AppError from "@shared/errors/AppError";

interface IPayload {
  sub: string;
}

const ensureAuthenticated = async (request: Request, response: Response, next:
NextFunction) => {
  const authHeader = request.headers.authorization;

  if (!authHeader) {
    throw new AppError("Token missing", 401);
  }

  const [, token] = authHeader.split(" ");

  try {
    const { sub } = verify(token, "") as IPayload;
    const usersRepository = new UsersRepository();
    const user = await usersRepository.findByID(sub);

    if (!user) {
      throw new AppError("User does not exists", 401);
    }

    request.user = {id: user.id};

    next();
  } catch {
    throw new AppError("Invalid token", 401);
  }
};

export default ensureAuthenticated;
```

Fonte: Autoria própria (2022)

Ainda usando como exemplo o módulo de torneios, o próximo passo da requisição é a função *handle* na classe do *controller*, na qual todas as validações referentes à estrutura da requisição serão realizadas. Os *controllers* estão separados por *useCases* dentro dos módulos e, para esse exemplo, pode-se analisar o caso de uso responsável pelos sorteios dos jogadores do

torneio chamado de *RaffleTournamentController*. A Listagem 5 exibe o comportamento deste *controller*.

Listagem 5 – *Controller* de sorteio dos jogadores no torneio

```
import { Request, Response } from "express";
import { container } from "tsyringe";
import RaffleTournamentUseCase from "./RaffleTournamentUseCase";

class RaffleTournamentController {
  async handle(request: Request, response: Response): Promise<Response> {
    const { tournamentID } = request.params;
    const raffleTournamentUseCase = container.resolve(RaffleTournamentUseCase);
    const raffledPlayers = await raffleTournamentUseCase.execute(tournamentID);

    return response.status(201).json(raffledPlayers);
  }
}

export default RaffleTournamentController;
```

Fonte: Autoria própria (2022)

Como citado anteriormente, o *controller* recebe os parâmetros das rotas, realiza qualquer tipo de validação e repassa os parâmetros para o *useCase* por meio de injeção de dependência (na qual para este projeto a biblioteca *tsyringe* é a responsável). Também é função do *controller* realizar o retorno de um *status* HTTP para o cliente.

O *useCase* irá centralizar toda a regra de negócio necessária para o caso de uso da rota chamada. Neste exemplo, validações dos dados necessários para a execução da regra e a regra em si, estão descritas no arquivo *RaffleTournamentUseCase* como mostra a Listagem 6.

Listagem 6 – *Usecase* de sorteio dos jogadores no torneio

```
import { inject, injectable } from "tsyringe";
import TournamentPlayer from
"@modules/tournaments/infra/typeorm/entities/TournamentPlayer";
import ITournamentsPlayersRepository from
"@modules/tournaments/repositories/ITournamentsPlayersRepository";
import ITournamentsRepository from
"@modules/tournaments/repositories/ITournamentsRepository";
import AppError from "@shared/errors/AppError";

@injectable()
class RaffleTournamentUseCase {
  constructor(
    @inject("TournamentsRepository")
    private tournamentsRepository: ITournamentsRepository,
    @inject("TournamentsPlayersRepository")
    private tournamentsPlayersRepository: ITournamentsPlayersRepository
  ) {}

  async execute(tournamentID: string): Promise<TournamentPlayer[]> {
    const tournament = await this.tournamentsRepository.findByID(tournamentID);
```

```
    if (!tournament) {
      throw new AppError("Tournament not found");
    }

    const tournamentPlayers = await
this.tournamentsPlayersRepository.findByTournamentID(tournamentID);

    const raffledPlayers = tournamentPlayers.map((player) => ({player,sortIndex:
Math.random(),}))
      .sort((a, b) => a.sortIndex - b.sortIndex)
      .map(({ player }) => player);

    await raffledPlayers.forEach(async (player, index) => {
      await this.tournamentsPlayersRepository.editTournamentPlayer({
        id: player.id,
        position: index,
      });
    });

    return raffledPlayers;
  }
}

export default RaffleTournamentUseCase;
```

Fonte: Autoria própria (2022)

Em um primeiro momento é realizada a validação para verificar se o parâmetro necessário é válido. Nesse caso, é verificado se o torneio existe na base de dados para que os jogadores possam ser selecionados por meio do algoritmo de sorteio aleatório. O resultado é retornado para o *controller* que devolve para o cliente da requisição.

Dos demais *useCases* criados no projeto, é interessante destacar aquele que contém o controle do resultado dos desafios e todas as regras da pontuação de cada jogador que são definidas baseado no placar final do jogo. O arquivo se chama *CreateChallengeResultsUseCase* e faz parte dos *useCases* do módulo *challenges* responsável pelo contexto de desafios do sistema.

A Listagem 7 exibe as importações de arquivos dependentes para o *useCase* supracitado.

Listagem 7 – Importações de dependências do *useCase* de resultados do desafio

```
import { inject, injectable } from "tsyringe";
import ICreateChallengeResultsDTO from
"@modules/challenges/dtos/ICreateChallengeResultsDTO";
import ChallengeResults from
"@modules/challenges/infra/typeorm/entities/ChallengeResults";
import { IChallengesRepository } from
"@modules/challenges/repositories/IChallengesRepository";
import{ IChallengesResultsRepository } from
"@modules/challenges/repositories/IChallengesResultsRepository";
```

```
import ITournamentsPlayersRepository from
"@modules/tournaments/repositories/ITournamentsPlayersRepository";
import AppError from "@shared/errors/AppError";
import AppDataSource from "@shared/infra/typeorm";

import {
 PLAYER_POINTS_NOTHING_TWO,
 PLAYER_POINTS_ONE_TWO,
 PLAYER_POINTS_TWO_NOTHING,
 PLAYER_POINTS_TWO_ONE,
 EXPIRED_GAME_POINTS,
 GUIVEN_UP_EXCEED_GAME_POINTS,
} from "@utils/constants";
```

Fonte: Autoria própria (2022)

As dependências deste *useCase* são as entidades e repositórios que serão utilizadas para validações, o módulo responsável por manipular os erros da aplicação (*AppError*), funções da biblioteca *tsyringe* para tratamento de injeção de dependências e algumas constantes que tem o papel de deixar o código mais limpo e consistente por meio da centralização de valores, criados com nomes declarativos facilitando o entendimento.

A Listagem 8 mostra a parte do *useCase* onde são declaradas as propriedades e a função da classe que vão receber o valor do resultado do desafio e estabelecer qual a pontuação final para cada jogador.

Listagem 8 – Declarações de função do *useCase* de resultados do desafio

```
@injectable()
class CreateChallengeResultsUseCase {
 private originPlayerPoints: number;
 private destinationPlayerPoints: number;

 constructor(
   @inject("ChallengesResultsRepository")
   private challengeResultsRepository: IChallengesResultsRepository,
   @inject("ChallengesRepository")
   private challengesRepository: IChallengesRepository,
   @inject("TournamentsPlayersRepository")
   private tournamentsPlayersRepository: ITournamentsPlayersRepository
 ) {
   this.originPlayerPoints = 0;
   this.destinationPlayerPoints = 0;
 }

 handlePlayersPontuationBySets = ({
   originPlayerFirstSet,
   destinationPlayerFirstSet,
   originPlayerSecondSet,
   destinationPlayerSecondSet,
   originPlayerTiebreak,
   destinationPlayerTiebreak,
 }: ICreateChallengeResultsDTO) => {
   if (originPlayerTiebreak > destinationPlayerTiebreak) {
```

```
      this.originPlayerPoints = PLAYER_POINTS_TWO_ONE;
      this.destinationPlayerPoints = PLAYER_POINTS_ONE_TWO;
      return;
    }

    if (originPlayerTiebreak < destinationPlayerTiebreak) {
      this.originPlayerPoints = PLAYER_POINTS_ONE_TWO;
      this.destinationPlayerPoints = PLAYER_POINTS_TWO_ONE;
      return;
    }

    if (originPlayerFirstSet > destinationPlayerFirstSet && originPlayerSecondSet
> destinationPlayerSecondSet) {
      this.originPlayerPoints = PLAYER_POINTS_TWO_NOTHING;
      this.destinationPlayerPoints = PLAYER_POINTS_NOTHING_TWO;
      return;
    }

    if (originPlayerFirstSet < destinationPlayerFirstSet && originPlayerSecondSet
< destinationPlayerSecondSet) {
      this.originPlayerPoints = PLAYER_POINTS_NOTHING_TWO;
      this.destinationPlayerPoints = PLAYER_POINTS_TWO_NOTHING;
    }
  };
```

Fonte: Autoria própria (2022)

Quando é utilizada a biblioteca *tsyringe*, a declaração da classe deve receber obrigatoriamente a anotação *'@injectable()'*.

Na sequencia observa-se a declaração da função *execute*, que é a principal das classes de *useCases* e contém toda a regra de negócio. A Listagem 9 exibe o trecho desta função onde são feitas as validações das propriedades que declaram um jogo que não ocorreu, seja por desistência de algum dos jogadores, o jogador desafiado que usou a sua recusa (se possível) e quantos pontos cada um recebe segundo as regras.

Listagem 9 – Validações de desafio do *useCase* de resultados do desafio

```
async execute(data: ICreateChallengeResultsDTO): Promise<void> {
    const { challengeID } = data;

    const challengeExists = await this.challengesRepository.findByID(
      challengeID
    );

    if (!challengeExists) {
      throw new AppError("Challenge does not exists");
    }

    if (data.refused) {
      const refusedChallengesByDestinationPlayer =
        await AppDataSource.getRepository(ChallengeResults)
          .createQueryBuilder("challengesResults")
          .leftJoinAndSelect("challengesResults.challengeID", "challenges")
```

```
        .where("challenges.destinationPlayerID = :destinationPlayerID", {
          destinationPlayerID: challengeExists.destinationPlayerID,
        })
        .andWhere("challengesResults.refused = true")
        .getMany();

    if (refusedChallengesByDestinationPlayer.length > 0) {
      throw new AppError("This player already has refused an challenge");
    }
  }

  if (data.destinationPlayerGiveup) {
    const givenUpChallengesByDestinationPlayer =
      await AppDataSource.getRepository(ChallengeResults)
        .createQueryBuilder("challengesResults")
        .leftJoinAndSelect("challengesResults.challengeID", "challenges")
        .where("challenges.destinationPlayerID = :destinationPlayerID", {
          destinationPlayerID: challengeExists.destinationPlayerID,
        })
        .andWhere("challengesResults.destinationPlayerGiveup = true")
        .getMany();

    if (givenUpChallengesByDestinationPlayer.length > 1) {
      this.destinationPlayerPoints = GUIVEN_UP_EXCEED_GAME_POINTS;
    }
  }

  if (data.originPlayerGiveup) {
    const challenges = await this.challengesRepository.list();
    const challengeResults = await this.challengeResultsRepository.list();

    const originPlayerChallenges = challenges
      .filter(
        (challenge) =>
          challenge.originPlayerID === challengeExists.originPlayerID ||
          challenge.destinationPlayerID === challengeExists.originPlayerID
      )
      .map((challenge) => challenge.id);

    const originPlayerResults = challengeResults.filter((results) =>
      originPlayerChallenges.includes(results.challengeID)
    );

    if (originPlayerResults.length > 1) {
      this.originPlayerPoints = GUIVEN_UP_EXCEED_GAME_POINTS;
    }
  }

  if (data.expired) {
    this.originPlayerPoints = EXPIRED_GAME_POINTS;
    this.destinationPlayerPoints = EXPIRED_GAME_POINTS;
    return;
  }

  if (this.originPlayerPoints === 0 && this.destinationPlayerPoints === 0) {
    this.handlePlayersPontuationBySets(data);
  }
```

Fonte: Autoria própria (2022)

Nas funções que definem as pontuações é importante ressaltar o uso das constantes para a facilidade de leitura e manutenção de código. Caso algum dos valores seja alterado devido à novas regras, a alteração ocorre apenas na declaração das constantes e não no código todo.

A Listagem 10 exibe a etapa final do fluxo de criação do resultado de um desafio, que é a definição das posições finais dos jogadores dentro do torneio.

Listagem 10 – Definição de pontuações do *useCase* de resultados do desafio

```
  if (this.originPlayerPoints > this.destinationPlayerPoints) {
      const tournamentPositions =
        await this.tournamentsPlayersRepository.findByTournamentID(
          challengeExists.tournamentID
        );

      const challengePlayersPositions = tournamentPositions.filter(
        (position) =>
          position.playerID === challengeExists.destinationPlayerID ||
          position.playerID === challengeExists.originPlayerID
      );

      const newPositions = [
        {
          ...challengePlayersPositions[0],
          position: challengePlayersPositions[1].position,
        },
        {
          ...challengePlayersPositions[1],
          position: challengePlayersPositions[0].position,
        },
      ];

      await this.tournamentsPlayersRepository.editTournamentPlayer(
        newPositions[0]
      );

      await this.tournamentsPlayersRepository.editTournamentPlayer(
        newPositions[1]
      );
    }

    await this.challengeResultsRepository.create({
      ...data,
      originPlayerPoints: this.originPlayerPoints,
      destinationPlayerPoints: this.destinationPlayerPoints,
    });
  }
}

export default CreateChallengeResultsUseCase;
```

Fonte: Autoria própria (2022)

Caso o jogador que realizou o desafio obtenha uma pontuação maior, as posições de ambos são alternadas, caso contrário, se mantém as mesmas.

4.4.2 Cliente

Para a criação do cliente (*front-end*) foi utilizado o *framework* NextJS, junto com a biblioteca ChakraUI que, fornece uma gama de componentes visuais acessíveis para a construção das interfaces.

O gerenciamento dos pacotes foi realizado com o Yarn Package Manager, que auxilia o projeto a controlar as versões e interdependências de todas as bibliotecas usadas no *front-end*.

Projetos criados com NextJS possuem um arquivo principal chamado *_app.tsx* onde algumas configurações globais podem ser realizadas. Para este projeto foi criado neste arquivo um controle de componentes autenticados, para garantir que existe uma sessão do usuário ativa antes que o mesmo seja renderizado em tela. A Listagem 11 exibe código do arquivo citado.

Listagem 11 – Componente principal da aplicação cliente

```
import { useEffect } from 'react'
import type { AppProps } from 'next/app'
import { signIn, useSession } from 'next-auth/react'
import Providers from 'providers'
import { NextPageWithLayout } from 'types/page'
import MainLayout from 'layouts/MainLayout'
import 'service/index'
import 'styles/globals.css'

type AppPropsWithLayout = AppProps & {
  Component: NextPageWithLayout & { auth?: boolean }
}

export default function MyApp({
  Component,
  pageProps: { session, ...pageProps }
}: AppPropsWithLayout) {
  const getLayout = Component.getLayout ?? (page => page)

  return getLayout(
    <Providers pageProps={pageProps} session={session}>
      <MainLayout>
        {Component.auth ? (
          <Auth>
            <Component {...pageProps} />
          </Auth>
        ) : (
          <Component {...pageProps} />
        )}
      </MainLayout>
    </Providers>
  )
}

function Auth({ children }: any) {
  const { data: session, status } = useSession()
  const isUser = !!session?.user
```

```
  useEffect(() => {
    if (status === 'loading') return
    if (!isUser) signIn()
  }, [isUser, status])

  if (isUser) {
    return children
  }
  return <div>Loading...</div>
}
```

Fonte: Autoria própria (2022)

No código apresentado na Listagem 11, o componente *Providers* contém todas as configurações de serviços globais da aplicação. O *MainLayout* traz outros componentes que são exibidos em todas as telas da aplicação. Como por exemplo as barras de navegação e uma condicional para saber se o componente que vai ser renderizado nas telas possui a propriedade chamada *auth*, declarando que este é um componente de rota privada. Para casos positivos, neste projeto foi utilizada a biblioteca NextAuth para verificação da sessão ativa do usuário.

Um exemplo de declaração de componente de rota privada pode ser visto na Listagem 12 com o arquivo de rota *tournaments.tsx.* Em sua declaração, o componente recebe a propriedade *auth* indicando que é uma renderização privada.

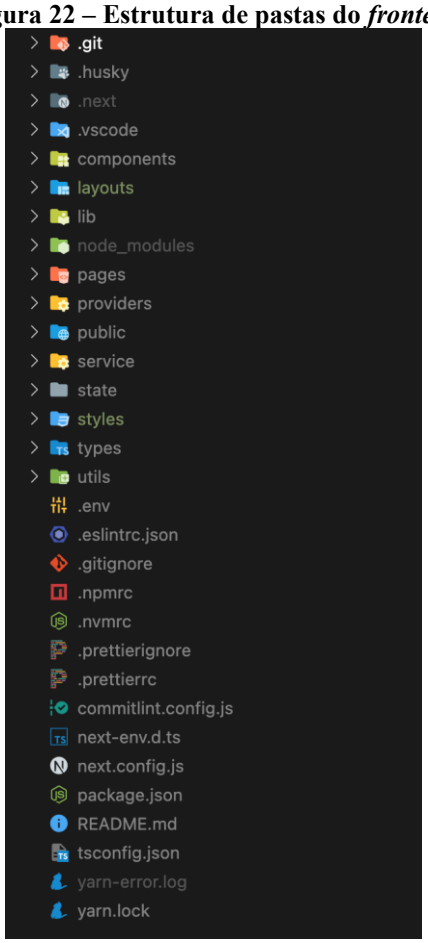
Listagem 12 – Exemplo de componente de rota autenticada

```
import LayoutTournaments from 'layouts/Tournaments/LayoutTournaments'
import { NextPageWithLayout } from 'types/page'

const Tournaments: NextPageWithLayout = () => <LayoutTournaments />

Tournaments.auth = true

export default Tournaments

Tournaments.getLayout = page => {
  return <>{page}</>
}
```

Fonte: Autoria própria (2022)

A Figura 22 mostra a estrutura completa de pastas criada para o projeto do cliente.

Figura 22 – Estrutura de pastas do *frontend*



Fonte: Autoria própria (2022)

A pasta *pages* possui todos os arquivos e diretórios que formam as rotas do projeto. O *NextJS* faz automaticamente o controle de rotas e sub rotas para cada arquivo criado dentro deste diretório. A função de cada página é fazer qualquer gerenciamento que possa existir em relação a renderização inicial dos componentes. A Figura 23 exibe todos os arquivos que definem as rotas do projeto.

Figura 23 – Estrutura de arquivos de rotas do *frontend*

Fonte: Autoria própria (2022)

Neste projeto não existem sub rotas, mas caso necessário, estas são organizadas por pastas sendo o nome da pasta a rota principal.

O diretório *providers* contém todos os arquivos que necessitam alguma configuração ou parâmetro que serão utilizados de forma global na aplicação, como temas e estilos, chamadas para a API e controle de rotas autenticadas. Já na pasta *layouts* estão os componentes principais de cada página e geralmente são responsáveis pelas requisições para o servidor e o gerenciamento das informações retornadas da API. Os componentes que possuem potencial para serem reutilizados na aplicação como um todo, são encontrados na pasta *components*. A pasta *service* contém todo o gerenciamento e funções que fazem comunicação com o servidor.

Na árvore de diretórios há, também, a pasta chamada *.husky* que armazena arquivos de configuração para que alguns comandos sejam automaticamente executados antes de um código ser enviado para o repositório (para que seja feita a verificação das mensagens do registro de alterações dentro de um formato específico e para rodar *scripts* que fazem a manutenção dos estilos do código, padronizando o repositório e facilitando o entendimento do código).

O projeto também conta com as bibliotecas *eslint* e *prettier*, que mantém um padrão de estilo de código (indentação, tabulação, entre outros). As configurações se encontram nos arquivos *.prettierrc* e *.eslintrc.json*. E por fim na pasta *utils* estão funções utilitárias (formatação de data por exemplo) que são compartilhadas com o projeto todo.

Dos componentes criados no projeto, pode-se destacar aquele que faz a montagem do desenho de pirâmide com os jogadores no torneio, sendo uma das funções principais do projeto. Na Listagem 13 está o código completo desta página.

Listagem 13 – Código do componente da pirâmide no cliente

```
import { useEffect, useState } from 'react'

import { Flex } from '@chakra-ui/react'
import { ActiveTournamentPlayers } from 'service/tournaments'
import PyramidItem from './PyramidItem'

type PyramidProps = {
 tournamentPlayers: Array<ActiveTournamentPlayers>
}

const Pyramid = ({ tournamentPlayers }: PyramidProps) => {
 const [playerLines, setPlayerLines] =
useState<Array<ActiveTournamentPlayers>>([])

 useEffect(() => {
   if (tournamentPlayers) {
     let pyramidItems: any[] = []

     let countIterator = 0
     let quantity = 1
     let playerPerLine = 2

     while (countIterator < tournamentPlayers.length) {
       pyramidItems = [
         ...pyramidItems,
         tournamentPlayers.slice(countIterator, quantity)
       ]
       countIterator = quantity
       quantity = quantity + playerPerLine
       playerPerLine = playerPerLine + 2
     }
     setPlayerLines(pyramidItems)
   }
 }, [tournamentPlayers])


 return (
   <Flex direction="column">
     {playerLines.map((item: any, index: any) => (
       <Flex key={index} justify="center">
         {item.map((item: any, index: any) => (
           <PyramidItem key={item.player.id} tournamentPlayer={item} />
         ))}
       </Flex>
     ))}
   </Flex>
 )
}

export default Pyramid
```

Fonte: Autoria própria (2022)

O componente apresentado no código da Listagem 13, recebe uma propriedade chamada *tournamentPlayers* que é requisitada no servidor e repassada por meio do componente pai que

o renderiza, e então executa a lógica utilizada para separar os jogadores por linhas e colunas da pirâmide, considerando a posição recebida do servidor. Cada jogador é renderizado em um componente secundário chamado *PyramidItem*, que faz a utilização da estrutura de componente *Popover* da biblioteca ChakraUI. A Listagem 14 exibe o código supracitado.

Listagem 14 – Código do item da pirâmide que representa o jogador

```
import { Flex, Popover, PopoverArrow, PopoverBody, PopoverCloseButton,
PopoverContent, PopoverHeader, PopoverTrigger, Text } from '@chakra-ui/react'
import { format, parseISO } from 'date-fns'
import { ActiveTournamentPlayers } from 'service/tournaments'

type PyramidItemProps = {
  tournamentPlayer: ActiveTournamentPlayers
}

const PyramidItem = ({ tournamentPlayer }: PyramidItemProps) => {
  const initialParsed = parseISO(tournamentPlayer.activeChallenge?.initialDate)
  const finalParsed = parseISO(tournamentPlayer.activeChallenge?.finalDate)

  return (
    <Popover trigger="hover" placement="top-end">
      <PopoverTrigger>
        <Flex mb="-1px" mr="-1px" border="1px solid #ccc" borderRadius="4px"
            justify="center" width="170px" align="center" height="50px"
            background={tournamentPlayer.activeChallenge ? 'red' : 'white'}
            color={tournamentPlayer.activeChallenge ? 'white' : 'black'}
          _hover={{
            background: '#eee',
            cursor: 'pointer',
            color: 'black',
            transition: ' transform 300ms',
            transform: 'translateY(-2px)'
          }}
        >
          <Text textAlign="center" fontWeight={500} fontSize="0.9rem">
            {tournamentPlayer.player.name}
          </Text>
        </Flex>
      </PopoverTrigger>

      <PopoverContent>
        <PopoverArrow />
        <PopoverCloseButton />
        <PopoverHeader>Dados do Desafio</PopoverHeader>
        <PopoverBody>
          {!tournamentPlayer.activeChallenge && (
            <Text fontWeight={500}>Esse jogador pode ser desafiado!</Text>
          )}

          {tournamentPlayer.activeChallenge && (
            <Flex direction="column">
              <Flex my="3px">
                <Text fontWeight={600}>Desafiador:</Text>
                <Text>
                  {tournamentPlayer.activeChallenge?.originPlayerName}
```

```
                </Text>
              </Flex>

              <Flex my="3px">
                <Text fontWeight={600}>Desafiado:</Text>
                <Text>
                  {tournamentPlayer.activeChallenge?.destinationPlayerName}
                </Text>
              </Flex>

              <Flex my="3px">
                <Text fontWeight={600}>Data do Desafio:</Text>
                <Text>{format(initialParsed, 'dd/MM/yyyy')}</Text>
              </Flex>

              <Flex my="3px">
                <Text fontWeight={600}>Prazo Final:</Text>
                <Text>{format(finalParsed, 'dd/MM/yyyy')}</Text>
              </Flex>
            </Flex>
          )}
        </PopoverBody>
      </PopoverContent>
    </Popover>
  )
}

export default PyramidItem
```

Fonte: Autoria própria (2022)

O componente apresentado no código da Listagem 14, recebe as informações do jogador em sua posição correta e o componente *Popover* é responsável por exibir as informações adicionais do desafio, quando houver, e a mensagem de permissão para novos desafios para um jogador, quando possível.

5 CONCLUSÃO

Neste trabalho foi desenvolvida uma aplicação para controle de tênis no modelo de pirâmide segundo as regras descritas no Regulamento da 4ª Pirâmide do Tênis do clube Grêmio Industrial Patobranquense, para ser utilizada no referido clube.

Para a otimização deste objetivo foram desenvolvidas duas partes desacopladas sendo uma cliente (*front-end*) e a outra o servidor (*back-end*), ambas com *frameworks*, bibliotecas e arquiteturas que possibilitaram um desenvolvimento mais fácil e ágil, permitindo que o foco seja maior na elaboração das funcionalidades para o usuário e não tanto em complexidade de código.

A parte do *front-end* foi criada com algumas bibliotecas e *frameworks*, sendo os principais: o *framework* NextJS, a biblioteca ChakraUI e a linguagem Typescript, o que possibilita um desenvolvimento rápido, prático e seguro. Já no *back-end,* o *framework Express* e a biblioteca de ORM Typeorm foram utilizadas criando uma estrutura robusta, modular e escalável junto com o nosso banco de dados *PostgreSQL*.

Algumas dificuldades foram encontradas ao longo do desenvolvimento da aplicação. Com relação ao desenvolvimento do lado servidor, as dificuldades se deram devido à pouca experiência com a tecnologia utilizada e à complexidade da rota de criação dos resultados dos desafios e das posições dos jogadores, na qual a criação modular das funções, que definem as regras, forneceu um melhor entendimento. No lado cliente, a maior dificuldade encontrada foi a criação da interface de exibição do torneio em formato de pirâmide, por possuir uma lógica mais difícil para ser desenvolvida, para a qual foi criada uma estrutura de repetição com o cálculo de linhas e colunas respeitando a posição correta de cada jogador. Desta forma, ao exceder as dificuldades apresentadas, o desenvolvimento do projeto resulta em grandes aprendizados.

Como trabalhos futuros sugere-se a implementação de acesso para os jogadores, para que possam criar seus desafios individualmente, automatização de rotinas para bloqueio dos jogos quando estiverem com prazos expirados, uma opção para duplicar torneios e facilitar a criação por parte dos administradores, gerenciamento de novos usuários também por parte dos administradores, e a possibilidade de que mais clubes possam utilizar o sistema.

REFERÊNCIAS

CARTA, Gianni. **O Tênis no Brasil: de Maria Esther Bueno a Gustavo Kuerten**. 1 Edição. Editora Códex. 2004.

CACHOEIRA TÊNIS, 2014. Disponível em: http://www.cachoeirastenis.com.br/regulamentos_ranking_2014_Anexo.html. Acesso em: 10 nov. 2021.

COMISSÃO DOS TENISTAS DO GRÊMIO INDUSTRIAL PATOBRANQUENSE. **Regulamento da 4ª Pirâmide do Tênis – Junho a Dezembro/2021.** 2021.

GODTSFRIEDT, Jonas; ANDRADE, Alexandro. Treinamento Mental no Tênis: Revisão Sistemática da Literatura. **Rev. Bras. Ciênc. Esporte**, Florianópolis, v. 36, n. 2, p. 577-586, 2014. Disponível em: https://www.scielo.br/j/rbce/a/n9cfsYHPrrCTpHBbJRq5qJH/?format=pdf&lang=pt. Acesso em 10 nov. 2021.

GONÇALVES, G. H. T., ASSMANN, A. B., GINCIENE, G., BALBINOTTI, C. A. A. y MAZO, J. Z. Uma história do tênis no Brasil: apontamentos sobre os clubes esportivos e seus métodos de ensino. **Educación Física y Ciencia**, 2018, 20(3), e057. Disponível em: https://doi.org/10.24215/23142561e057. Acesso em: 14 de nov. 2021.

GRIZZO, Arnaldo, 2007. Mercado brasileiro está se desenvolvendo. **Revista Tênis.** Edição 46. Disponível em: https://revistatenis.uol.com.br/artigo/mercado-brasileiro-esta-se-desenvolvendo_362.html. Acesso em: 27 de nov. 2021.

GRIZZO, Arnaldo, 2017. Entenda a contagem do placar no tênis. **Revista Tênis.** Edição 90. Disponível em: https://revistatenis.uol.com.br/artigo/conte-certo_6024.html. Acesso em: 24 de nov. 2021.

IBOPE REPUCOM, 2019. **Brasil tem 27 milhões de fãs de Tênis.** Disponível em: https://www.iboperepucom.com/br/noticias/rio-open-e-brasil-open-estao-entre-os-eventos-com-maior-interesse-da-populacao-conectada/. Acesso em: 27 de nov. 2021.

MILANI, André. **PostgreSQL - Guia do Programador.** 1 Edição. Editora Novatec. 2008.

MINISTERIO DO ESPORTE, 2015. **A Prática do Esporte no Brasil**. Disponível em: http://arquivo.esporte.gov.br/diesporte/2.html. Acesso em: 11 nov. 2021.

MORAES, William Bruno. **Construindo Aplicações com NodeJS.** 3 Edição. Editora Novatec. 2021.

PINTO, José Alberto; CUNHA, Flávio Henrique Gomes. **O TÊNIS COMO ALTERNATIVA NO CURRÍCULO ESCOLAR PARA CRIANÇAS ENTRE 8 E 12 ANOS.** MOTRIZ – Volume 4, Número 1, junho/1998. Disponível em: http://www.rc.unesp.br/ib/efisica/motriz/04n1/4n1_ART04.pdf. Acesso em: 14 de nov. 2021.

SILVA, Alan Fialho; OLIVEIRA, Leônidas. Os Benefícios da Prática Esportiva e do Exercício para a Saúde e Qualidade de Vida do Indivíduo. **XVI Encontro Latino Americano de Iniciação Científica e XII Encontro Latino Americano de Pós-Graduação –**

Universidade do Vale do Paraíba, 2018. Disponível em: http://www.inicepg.univap.br/cd/INIC_2012/anais/arquivos/0621_0243_01. Acesso em: 11 nov. 2021.

TYPESCRIPT. **Typescript Handbook - Typescript for the New Programmer.** Disponível em: https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html. Acesso em 21 nov. 2022.

WEB DEVOLUTION, **Javascript Event Loop Explained.** 2021. Disponível em: https://www.webdevolution.com/blog/Javascript-Event-Loop-Explained. Acesso em: 10 nov. 2021.