UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências exatas e de tecnologia

Departamento de Computação

Trabalho de conclusão de curso - TCC

Ricardo Augusto Lopes de Faria

Desenvolvimento Seguro em Plataformas de Fantasy Game: InterREP Manager

São Carlos - São Paulo

2024

Ricardo Augusto Lopes de Faria

Desenvolvimento Seguro em Plataformas de Fantasy Game: InterREP Manager

Trabalho de Conclusão de Curso apresentado ao Departamento de Computação como parte dos requisitos para a conclusão da graduação em Engenharia de Computação.

Orientação Prof. Dr. Paulo Matias

São Carlos - São Paulo

2024

DEDICO

Agradecimentos

Com todo o meu coração, agradeço à minha família por me amparar nos momentos mais desafiadores e me inspirar a nunca desistir dos meus sonhos. Vocês são a base da minha força e o alicerce da minha esperança.

Estendo minha profunda gratidão aos meus amigos, companheiros de jornada, que caminharam ao meu lado nestes últimos anos, me apoiando, me motivando. Vocês são parte essencial da minha história e merecem toda a minha gratidão.

Agradeço também à minha república estudantil, lar que me acolheu por 6 anos e onde tive a oportunidade de conhecer pessoas incríveis que levarei para sempre no coração. Vocês transformaram minha experiência universitária em algo inesquecível.

"Não há fatos eternos, como não há verdades absolutas."
(Friedrich Nietzsche)

Resumo

O presente trabalho aborda o desenvolvimento de um aplicativo web front-end para o fantasy game do torneio InterREP, com foco na experiência do usuário e na segurança das informações. A interface do aplicativo foi projetada seguindo as Heurísticas de Nielsen, buscando a clareza, consistência e facilidade de uso. Além disso, foram implementadas medidas de segurança para proteger os dados dos usuários e garantir a integridade do sistema. O desenvolvimento do aplicativo utilizou tecnologias modernas, como o framework Next.js 14, o gerenciador de pacotes pnpm e a biblioteca de componentes Shadcn/UI, proporcionando uma experiência de desenvolvimento eficiente e um produto final de alta qualidade. Espera-se que o aplicativo aprimore a experiência dos participantes do InterREP, fortalecendo o torneio e promovendo a integração entre as repúblicas estudantis.

Palavras-chave: Fantasy Game, Desenvolvimento Web ui/ux, Segurança

Abstract

This work addresses the development of a front-end web application for the InterREP tournament's fantasy game, focusing on user experience and information security. The application interface was designed following Nielsen's Heuristics, seeking clarity, consistency, and ease of use. Furthermore, security measures were implemented to protect user data and ensure system integrity. The application development utilized modern technologies, such as the Next.js 14 framework, the pnpm package manager, and the Shadcn/UI component library, providing an efficient development experience and a high-quality final product. The application is expected to enhance the experience of InterREP participants, strengthening the tournament and promoting integration among student republics.

Keywords: Fantasy Game, Web Development, UI/UX, Security

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

A revolução digital transformou a sociedade, abrindo espaço para novas formas de entretenimento, como os Fantasy Games. Nesse cenário dinâmico, emergiram como um fenômeno global, conquistando fãs e transformando a maneira como as pessoas se relacionam com o esporte (CASAGRANDE; LAVARDA; SILVEIRA, 2021). A paixão pelo esporte, que antes se manifestava principalmente nas arquibancadas, agora encontra expressão também no mundo virtual, onde os torcedores podem testar seus conhecimentos em competições.

Fantasy Games permitem que jogadores criem times virtuais com atletas reais, combinando fantasia e estratégia. Competição, planejamento e interação social tornam os Fantasy Games uma experiência imersiva. A compreensão de símbolos e elementos visuais é fundamental para a prática desses jogos, que se popularizaram nas Américas e abrangem diversas modalidades esportivas, com destaque para o futebol. Através dos Fantasy Games, os torcedores apaixonados encontram uma maneira de demonstrar seus conhecimentos e paixões esportivas, que impulsionam ainda mais o crescimento desse mercado. A projeção de crescimento do mercado de Fantasy Games, que deve passar dos US$ 40 bilhões até 2026, evidencia a força e o potencial desse fenômeno. (EXAME, 2022)

No Brasil, o Cartola FC, Fantasy Game de futebol, lidera o cenário, conquistando milhões de jogadores e se tornando um fenômeno cultural. Inspirado no sucesso do Cartola FC, este projeto propõe um aplicativo web para o Fantasy Football do InterREP, com foco na experiência do usuário e segurança.

O InterREP, torneio de futebol society entre repúblicas estudantis de São Carlos, reúne alunos da USP, UFSCar e IFSP em três modalidades: Série A, Série B e Campeonato Feminino. Com mais de 300 usuários e 700 jogadores, o fantasy game do torneio busca não apenas aumentar a visibilidade da competição, mas também agregar valor à experiência dos participantes, proporcionando uma nova forma de interação e entretenimento. Atualmente, a aplicação está datada e não possui responsividade para aplicativos mobile.

Este trabalho se propõe a aprimorar a experiência do fantasy game do InterREP, InterREP Manager,desenvolvendo um aplicativo web front-end que ofereça uma interface intuitiva e funcionalidades inovadoras para o registro e acompanhamento do desempenho dos jogadores. Através de um layout moderno e responsivo e da análise de dados, o aplicativo visa otimizar a experiencia de usuário, dessa forma trazendo ainda mais jogadores para o fantasy game.

Para garantir uma experiência de usuário fluida e agradável, o design da interface seguirá heurísticas, buscando a clareza, a consistência e a facilidade de uso. Além disso,

medidas de segurança serão implementadas para proteger os dados dos usuários e garantir a integridade do sistema.

Com a implementação deste aplicativo, espera-se fortalecer o InterREP como um evento esportivo de destaque, promovendo a integração entre as repúblicas estudantis e fomentando a paixão pelo futebol. A otimização do registro de desempenho dos jogadores, aliada a uma interface intuitiva e segura, contribuirá para uma experiência mais completa e envolvente para os participantes e torcedores, consolidando o InterREP Manager como um componente essencial do torneio.

2 Objetivos

2.1 Objetivos Gerais

O presente trabalho visa analisar e aplicar técnicas modernas no desenvolvimento de um produto digital, especificamente a plataforma InterREP Manager. O foco principal reside na criação de uma interface segura, intuitiva e eficiente para os usuários, explorando ferramentas de design, bibliotecas e frameworks de desenvolvimento front-end.

2.2 Objetivos Específicos

Este trabalho tenta propor uma solução para o problema de refazer o produto já existente, conforme contextualizado na introdução capítulo 1, para isso, precisa-se realizar alguns passos, como a escolha de uma ferramenta para o desenvolvimento gráfico, escolha de heurísticas e criação de um protótipo, escolha de linguagem e framework para o desenvolvimento front-end, criação do website, e consumo de dados recebidos de uma API simulada (mock). Esses passos podem ser resumidos nas seguintes etapas:

1. Seleção de um software adequado para a criação da interface visual da plataforma, considerando fatores como usabilidade, recursos e compatibilidade com o projeto;
2. Definição de princípios e diretrizes que nortearão o design da interface, garantindo a consistência, a facilidade de uso e a experiência positiva do usuário;
3. Desenvolvimento de uma versão inicial da interface, permitindo a visualização e o teste das funcionalidades antes da implementação final;
4. Seleção das tecnologias mais adequadas para a construção da interface, considerando aspectos como desempenho, escalabilidade e segurança;
5. Implementação da interface final, integrando-a com a API mock do site MockAPI (2024) para simular o funcionamento real. A utilização de uma API mock se faz necessária devido à ausência de um back-end dedicado, o que impõe limitações na interação com dados reais e funcionalidades mais complexas.

2.3 Resultados Alcançados

o final deste projeto, alcançou-se a criação de uma plataforma web funcional e intuitiva, utilizando os dados fornecidos pela API mock, proporcionando aos usuários uma

experiência de jogo aprimorada e facilitando a interação com as informações do jogo. A plataforma foi criada com uma interface gráfica que segue as melhores práticas de design, oferecendo uma navegação clara, acesso rápido às funcionalidades e informações relevantes, além de uma experiência visualmente agradável.

Para demonstrar a funcionalidade e a usabilidade da plataforma, foi desenvolvido um protótipo interativo, permitindo ajustes antes da implementação final.

Este projeto contribuiu para o conhecimento na área de desenvolvimento de produtos digitais, através de uma análise aprofundada das técnicas modernas, com foco em ferramentas de design, bibliotecas e frameworks para front-end.

3 Tecnologias e teorias utilizadas

3.1 Ferramentas de Design

A ferramenta escolhida foi *Figma*. Sua escolha foi feita devido à sua consolidação como a principal ferramenta de UI Design em 2023 Geoco (2023), de acordo com a pesquisa UX Tools. Sua dominância no mercado é evidente, com uma ampla vantagem sobre seus concorrentes. A plataforma oferece uma combinação única de recursos que a tornam indispensável para designers, incluindo colaboração em tempo real, prototipagem interativa e facilidade de uso.

A colaboração em tempo real é um dos principais diferenciais, permitindo que equipes de design trabalhem juntas de forma eficiente, independentemente da localização. Essa funcionalidade agiliza o processo de criação e feedback, tornando-o mais ágil e colaborativo.

Além disso, a capacidade de criar protótipos interativos de alta fidelidade diretamente na ferramenta é outro fator crucial para seu sucesso. Essa funcionalidade permite que designers testem e validem suas ideias de forma rápida e eficiente, garantindo que a experiência do usuário seja considerada desde o início do processo de design.

A facilidade de uso também contribui para sua popularidade. A interface intuitiva e a curva de aprendizado suave tornam a ferramenta acessível tanto para designers experientes quanto para iniciantes, proporcionando a democratização do design de interfaces de maior qualidade.

Em suma, *Figma* se destaca como a ferramenta líder em UI Design em 2023, impulsionada por sua colaboração em tempo real, recursos de prototipagem e facilidade de uso. Sua posição dominante no mercado reflete sua capacidade de atender às necessidades dos designers modernos, oferecendo uma plataforma completa e eficiente para a criação de interfaces de usuário de alta qualidade.

Outras ferramentas importantes são *Skecth* e *Adobe XD*, o último pela integração de todo ecossistema do Adobe como *Photoshop* e *Illustrator*.(GEOCO, 2023)

3.2 Heurística de Nielsen

As Heurísticas de Nielsen,(NIELSEN, 1994), são um conjunto de 10 princípios fundamentais que norteiam o design de interfaces de usuário intuitivas e eficazes. Desenvolvidas por Jakob Nielsen em 1994, essas diretrizes oferecem um guia valioso para designers

e desenvolvedores, ajudando a criar sistemas que proporcionam uma experiência positiva e satisfatória para os usuários.

Ao aplicar as Heurísticas de Nielsen, os designers podem garantir que a interface seja clara, intuitiva e fácil de usar. A utilização de elementos visuais familiares, mensagens de feedback claras e opções de desfazer ações contribuem para uma experiência de usuário mais fluida e agradável. Além disso, a prevenção de erros e o suporte para resolução de problemas garantem que os usuários se sintam confiantes e seguros ao utilizar o sistema.

Essas heurísticas abordam aspectos cruciais da usabilidade.

3.2.1 Visibilidade do Status do Sistema

O usuário deve ser informado sobre o que está acontecendo no sistema a todo momento, por meio de feedback claro e oportuno. Barras de progresso, mensagens de confirmação e indicadores visuais são exemplos de como manter o usuário informado.

3.2.2 Compatibilidade do Sistema com o Mundo Real

A interface deve utilizar linguagem e conceitos familiares ao usuário, espelhando o mundo real. Ícones intuitivos, como a lixeira para excluir arquivos, e termos comuns, como “adicionar ao carrinho”, facilitam a compreensão e a interação.

3.2.3 Controle e Liberdade do Usuário

O usuário deve ter autonomia para navegar e desfazer ações facilmente. Funcionalidades como “cancelar”, “desfazer” e “refazer” são cruciais para evitar frustrações e erros.

3.2.4 Consistência e Padrões

A interface deve seguir padrões de design e manter a consistência em elementos visuais, linguagem e comportamento. Isso facilita o aprendizado e a utilização do sistema, pois o usuário pode aplicar conhecimentos prévios em novas situações.

3.2.5 Prevenção de Erros

O design deve antecipar e prevenir erros do usuário. Mensagens claras de erro, validação de dados em tempo real e confirmações antes de ações irreversíveis são medidas importantes para evitar problemas.

3.2.6 Reconhecimento ao invés de Memorização

O usuário não deve precisar memorizar informações para usar o sistema. Elementos visíveis, como menus e ícones, devem estar sempre disponíveis para auxiliar na navegação e na execução de tarefas.

3.2.7 Flexibilidade e Eficiência de Uso

O sistema deve atender tanto usuários novatos quanto experientes, oferecendo opções de personalização, atalhos e funcionalidades avançadas para agilizar o uso.

3.2.8 Estética e Design Minimalista

A interface deve ser visualmente agradável e organizada, evitando informações desnecessárias e distrações. Um design limpo e minimalista facilita a leitura e a compreensão das informações.

3.2.9 Suporte para Identificar, Diagnosticar e Corrigir Erros

Mensagens de erro claras e informativas, juntamente com sugestões de solução, auxiliam o usuário a resolver problemas e evitar frustrações.

3.2.10 Ajuda e Documentação

Mesmo com um design intuitivo, é importante fornecer ajuda e documentação acessíveis para auxiliar o usuário em caso de dúvidas. Tutoriais, FAQs e suporte ao cliente são recursos valiosos para garantir uma boa experiência.

3.3 Linguagem

O JavaScript, linguagem fundamental para a web, impulsionou a criação de páginas dinâmicas e interativas. No entanto, o crescimento da complexidade das aplicações web expôs suas limitações, especialmente em projetos de grande escala.

O TypeScript é uma das soluções para auxiliar o desenvolvimento, oferecendo tipagem estática e recursos avançados que aprimoram a escalabilidade, a manutenibilidade e a robustez do código.

3.3.1 JavaScript

Linguagem interpretada, dinamicamente tipada, ideal para adicionar interatividade à web. Sua flexibilidade, porém, pode levar a erros em projetos complexos.

3.3.2 TypeScript

TypeScript é um Superset do JavaScript, adiciona tipagem estática, interfaces, classes e módulos. Permite identificar erros em tempo de desenvolvimento, facilitando a refatoração e a manutenção.

3.3.3 Comparativo

TypeScript facilita o desenvolvimento JavaScript ao introduzir um sistema de tipos estático que transforma a experiência de codificação. Ao contrário do JavaScript, onde os tipos são inferidos em tempo de execução, o TypeScript verifica os tipos durante a compilação, detectando erros antes mesmo de o código ser executado. Essa verificação antecipada previne uma classe de erros comuns em JavaScript, como atribuir valores a variáveis de tipos incompatíveis. Além disso, a tipagem explícita do TypeScript atua como uma forma de documentação com o *intellisense*, tornando o código mais legível e facilitando a colaboração entre desenvolvedores.

Características	**JavaScript**	**TypeScript**
Tipagem	Dinâmica	Estática
Verificação de Tipos	Em tempo de execução	Em tempo de compilação
Detecção de Erros	Tardia	Antecipada
Documentação	Implícita	Explícita

Tabela 1 – Tabela Comparativa entre JavaScript e TypeScript.

3.4 Análise Comparativa de Ferramentas JavaScript/TypeScript para Desenvolvimento Web

Esta seção apresenta uma análise das três principais ferramentas, figura 1, JavaScript para desenvolvimento web KROTOFF (2023): React, Vue.js e Angular. Cada framework será examinado em relação às suas características, vantagens e desvantagens, com o objetivo de auxiliar na escolha da ferramenta mais adequada para diferentes cenários de projeto.

3.4.1 Angular

Angular, mantido pelo Google, é um framework completo e robusto, ideal para projetos de grande escala que demandam escalabilidade e organização. Sua arquitetura baseada em componentes, injeção de dependência e diretivas personalizadas proporcionam uma estrutura sólida e opiniões claras,decisões e convenções predefinidas, sobre a construção de aplicações, garantindo consistência e manutenibilidade a longo prazo. A utilização do TypeScript, com sua tipagem estática e recursos avançados, contribui para a qualidade do

código, facilita a identificação de erros e aumenta a produtividade da equipe. (ANGULAR, 2024)

Vantagens:

- Estrutura robusta e opiniões claras para projetos de grande escala.
- TypeScript para maior qualidade de código e produtividade.
- Ecossistema completo de ferramentas e suporte do Google.

Desvantagens:

- Curva de aprendizado inicial mais acentuada.

3.4.2 React

React, desenvolvido por um time do Facebook, é uma biblioteca JavaScript flexível para construção de interfaces de usuário, amplamente utilizada em projetos de todos os portes. Sua abordagem baseada em componentes reutilizáveis, gerenciamento de estado e Virtual DOM permite a criação de aplicações rápidas, eficientes e com excelente experiência do usuário. A sintaxe JSX/TSX, que combina HTML e JavaScript ou TypeScript, facilita a escrita de código declarativo e intuitivo, tornando o desenvolvimento mais ágil e produtivo.

Vantagens:

- Flexibilidade e ecossistema rico em ferramentas.
- Sintaxe JSX/TSX para código declarativo e intuitivo.
- Amplamente utilizado em projetos de todos os portes.

Desvantagens:

- Exige decisões sobre arquitetura e ferramentas, o que pode ser desafiador para iniciantes.

3.4.3 Vue.js

Vue.js é um framework progressivo que une a simplicidade e flexibilidade com a estrutura e organização. Sua curva de aprendizado suave e a sintaxe intuitiva, baseada em templates HTML e diretivas personalizadas, facilitam a adoção por desenvolvedores de todos os níveis. Vue.js oferece grande flexibilidade, permitindo a escolha das ferramentas e bibliotecas mais adequadas às necessidades do projeto, como Vue Router para gerenciamento de rotas ou o Vuetify para a criação de componentes.

Vantagens:

- Simplicidade, flexibilidade e curva de aprendizado suave.
- Comunidade com crescente número de recursos.
- Adequado para projetos de diferentes portes e complexidades.

Desvantagens:

- Menor número de opiniões em relação a Angular, o que pode exigir mais decisões por parte dos desenvolvedores.

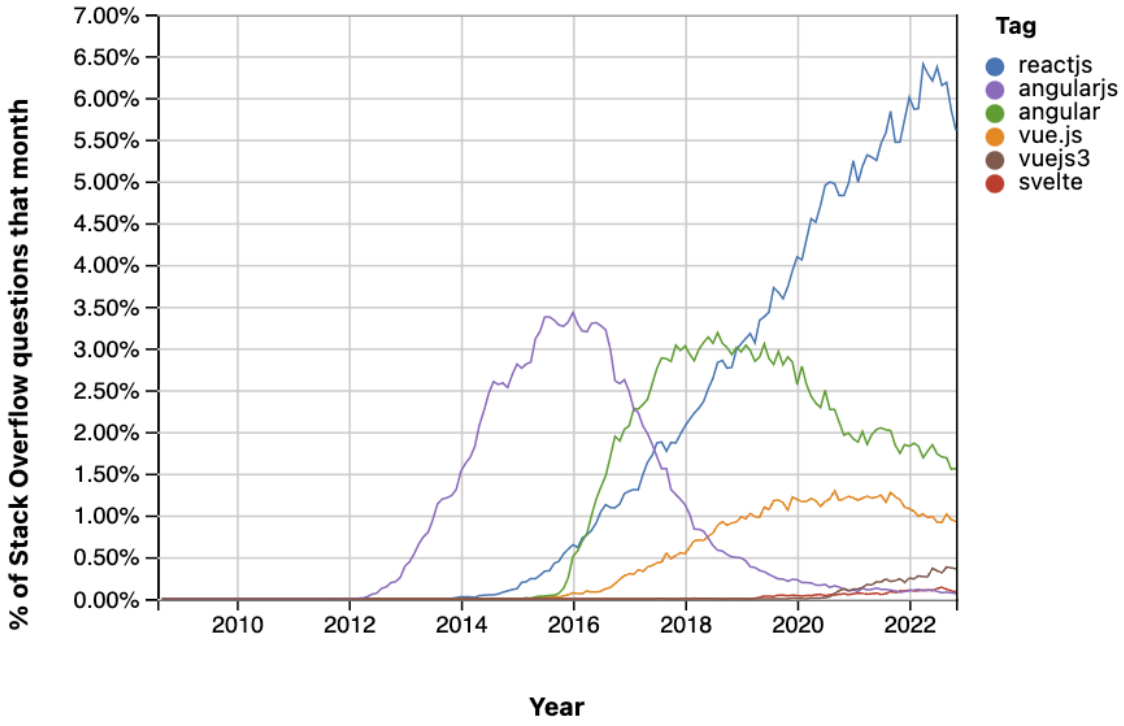

Figura 1 – Frameworks TypeScript/JavaScript. Fonte: StackOverflow

3.5 Renderização

A renderização de páginas da web é um aspecto crucial para a experiência do usuário e o desempenho do site. Existem diversas técnicas de renderização, cada uma com suas próprias características e casos de uso ideais. As duas abordagens principais são a renderização do lado do servidor (SSR) e a renderização do lado do cliente (CSR). No SSR, o servidor processa a página completa e envia o HTML finalizado para o navegador do cliente, enquanto no CSR, o navegador recebe um esqueleto da página e um arquivo JavaScript, que busca os dados e renderiza o conteúdo dinamicamente(NORDSTRöM; DIXELIUS, 2023).

Outras técnicas, como a Geração de Site Estático (SSG), também desempenham um papel importante, especialmente para sites com conteúdo estático ou que muda com pouca frequência. A escolha da técnica de renderização adequada depende de vários fatores, incluindo o tipo de conteúdo do site, a necessidade de interatividade e as condições de rede dos usuários.(NORDSTRöM; DIXELIUS, 2023)

3.5.1 Server-Side Rendering (SSR)

No SSR, o servidor renderiza a página da web completa, incluindo todos os componentes e dados, antes de enviar o HTML finalizado para o navegador do cliente. Isso significa que o navegador recebe uma página totalmente funcional e pronta para ser exibida, sem a necessidade de executar JavaScript para renderizar o conteúdo. O SSR é vantajoso para melhorar o desempenho percebido, especialmente em conexões mais lentas, pois o usuário vê o conteúdo inicial mais rapidamente. Além disso, o SSR é benéfico para SEO (*Search Engine Optimization*), pois os mecanismos de busca podem facilmente rastrear e indexar o conteúdo da página, já que ele está presente no HTML inicial. No entanto, o SSR pode aumentar a carga no servidor, especialmente em sites com alto tráfego, e pode não ser ideal para aplicações altamente interativas, onde atualizações frequentes do lado do cliente são necessárias.

3.5.2 Client-Side Rendering (CSR)

No CSR, o servidor envia um documento HTML básico com pouco ou nenhum conteúdo e um arquivo JavaScript para o navegador do cliente. O JavaScript, então, busca os dados necessários da API e renderiza dinamicamente o conteúdo da página no navegador. O CSR é vantajoso para criar aplicações web altamente interativas e dinâmicas, com atualizações de conteúdo em tempo real, sem a necessidade de recarregar toda a página. Além disso, pode reduzir a carga no servidor, pois a renderização é feita no lado do cliente. No entanto, pode levar a um tempo de carregamento inicial mais lento, especialmente em conexões mais lentas, pois o navegador precisa baixar e executar o JavaScript antes de renderizar o conteúdo. Além disso, essa técnica de renderização pode apresentar desafios para SEO, pois os mecanismos de busca podem ter dificuldade em rastrear e indexar o conteúdo renderizado dinamicamente pelo JavaScript.

3.5.3 Static Site Generation (SSG)

O SSG é um método de renderização que gera todo o site em tempo de compilação. Isso significa que todas as páginas são pré-renderizadas em arquivos HTML estáticos, que são armazenados em um servidor ou CDN (*Content Delivery Network*). Quando um usuário solicita uma página, o servidor simplesmente envia o arquivo HTML estático correspondente, resultando em tempos de carregamento extremamente rápidos. O SSG é

ideal para sites com conteúdo estático ou que muda com pouca frequência, como blogs, portfólios e sites de documentação. No entanto, o SSG não é adequado para sites com conteúdo altamente dinâmico ou personalizado, pois cada alteração no conteúdo exigiria uma reconstrução completa do site.

3.5.4 React Server Components

React Server Components (RSCs) são um novo paradigma no desenvolvimento React que permite a renderização de componentes no servidor, combinando os benefícios da renderização no lado do servidor (SSR) com a flexibilidade do React. React (2024b) Com os RSCs, o servidor processa e renderiza os componentes, buscando dados de bancos de dados ou APIs, e envia o HTML resultante para o cliente em partes, usando o conceito de streaming. Isso permite que o conteúdo seja exibido progressivamente, melhorando a experiência do usuário, especialmente em conexões mais lentas. No lado do cliente, um pacote JavaScript é responsável por "hidratar" os RSCs, tornando-os interativos.(VERCEL, 2023)

O padrão Island permite a coexistência de Server Components e Client Components em uma mesma aplicação. Essa arquitetura divide os componentes em estáticos (sem interatividade, renderizados no servidor) e dinâmicos (com interatividade, hidratados no cliente com JavaScript) (ISLAND, 2024). Essa divisão otimiza o desempenho, priorizando o carregamento rápido dos componentes estáticos e a interatividade sob demanda dos dinâmicos. Como mostrado na figura 2

Server Components são renderizados no servidor e não têm acesso direto ao DOM, enquanto Client Components são renderizados no cliente e podem gerenciar estado e interatividade. Ao isolá-los, o Client Component e seus filhos dentro de uma "ilha" de interatividade, ou seja, o usuário recebe JavaScript e bote interagir com a página. Essa abordagem combina os benefícios de ambas as técnicas de renderização, otimizando o desempenho, o SEO e a experiência do usuário.

Essa abordagem traz diversas vantagens, como melhor performance devido à renderização no servidor e ao streaming, SEO aprimorado pela disponibilidade do conteúdo no HTML inicial, menor uso de JavaScript no cliente, resultando em tempos de carregamento mais rápidos, e acesso direto a dados e APIs do servidor pelos RSCs. Vercel (2023) No entanto, também apresentam algumas desvantagens. A introdução de um novo paradigma pode exigir adaptação dos desenvolvedores, e a ausência de estado no cliente pode demandar soluções alternativas para gerenciar interações complexas. Além disso, a renderização no servidor pode limitar a personalização e a interatividade em tempo real.

Apesar das limitações, os RSCs são uma ferramenta poderosa para o desenvolvimento de aplicações React, especialmente em cenários onde o desempenho, o SEO e o

acesso a dados do servidor são cruciais. Eles são particularmente adequados para páginas com conteúdo dinâmico, que depende de dados externos, e para páginas com pouca interatividade, como blogs e páginas de destino.

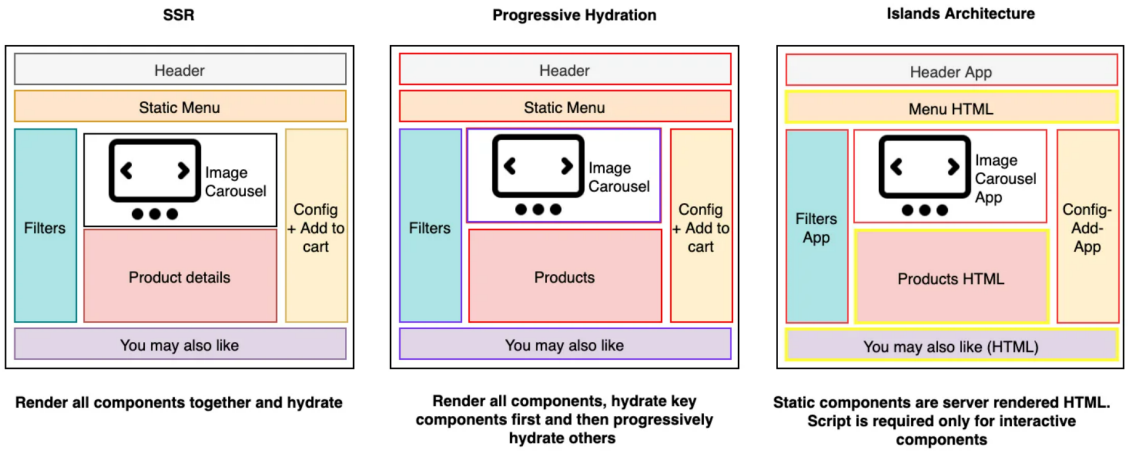

Figura 2 – Padrão Island. Fonte: patterns.dev

3.6 Gerenciadores de Pacotes

Um gerenciador de pacotes é uma ferramenta que automatiza o processo de instalação, atualização, configuração e remoção de softwares e bibliotecas em um projeto. Ele simplifica o gerenciamento de dependências, garantindo que todas as bibliotecas necessárias estejam presentes e com as versões corretas. NPM, Yarn, pnpm e Bun são gerenciadores de pacotes para JavaScript.

3.6.1 NPM

O NPM (*Node Package Manager*), sendo o gerenciador de pacotes padrão do Node.js, conquistou uma posição de destaque na comunidade JavaScript. Sua popularidade se deve, em grande parte, à sua extensa biblioteca de pacotes, que oferece uma ampla gama de soluções prontas para diversas necessidades de desenvolvimento. Essa vasta coleção de recursos agiliza o processo de desenvolvimento, permitindo que os desenvolvedores incorporem funcionalidades complexas em seus projetos com facilidade.

Outro fator que contribui para a popularidade do NPM é sua interface familiar e intuitiva. A curva de aprendizado relativamente suave, o que torna o NPM acessível tanto para iniciantes que estão começando no desenvolvimento JavaScript quanto para desenvolvedores experientes que buscam eficiência e produtividade.

No entanto, a vasta biblioteca de pacotes do NPM também apresenta um desafio significativo: a segurança. A grande quantidade de pacotes disponíveis aumenta a probabilidade de que alguns deles contenham vulnerabilidades que podem ser exploradas

por agentes maliciosos. Essas vulnerabilidades podem variar desde falhas de lógica que permitem acesso não autorizado a dados até problemas de execução de código remoto que podem comprometer a integridade da aplicação. Para mitigar esses riscos, o NPM oferece recursos de auditoria de segurança que auxiliam na identificação de vulnerabilidades em pacotes instalados. O comando *npm audit* realiza uma varredura nas dependências do projeto, comparando-as com um banco de dados de vulnerabilidades conhecidas. Caso alguma vulnerabilidade seja encontrada, o NPM fornece informações sobre o problema e sugere possíveis soluções, como atualizar para uma versão mais segura do pacote.

3.6.2 Yarn

O Yarn, gerenciador de pacotes desenvolvido pelo Facebook, foi desenvolvido devido às demandas por maior velocidade e confiabilidade no gerenciamento de dependências, especialmente em projetos de grande escala. Sua arquitetura e recursos inovadores o posicionam como uma alternativa atraente ao NPM, oferecendo vantagens significativas em termos de desempenho e segurança.

A principal inovação do Yarn reside em sua abordagem de cache e instalação paralela. O cache inteligente do Yarn armazena localmente os pacotes já baixados, eliminando a necessidade de repetidos downloads e acelerando o processo de instalação. Além disso, a instalação paralela permite que o Yarn baixe e instale múltiplos pacotes simultaneamente, otimizando ainda mais o tempo de configuração do projeto. Essa combinação de recursos resulta em um processo de gerenciamento de pacotes notavelmente mais rápido, crucial para projetos complexos com inúmeras dependências.

Além da velocidade, o Yarn também se destaca por seus recursos de segurança. A verificação de integridade garante que os pacotes instalados não foram corrompidos ou alterados durante o download, protegendo o projeto contra a inclusão de código malicioso. Adicionalmente, a instalação offline permite que o Yarn utilize pacotes armazenados em cache mesmo sem conexão à internet, garantindo a continuidade do desenvolvimento em situações adversas e reduzindo a dependência de fontes externas.

3.6.3 pnpm

O pnpm se destaca no cenário de gerenciadores de pacotes por sua abordagem inovadora que visa tanto a otimização de projetos quanto a segurança. Um dos seus principais diferenciais é a economia de espaço em disco, que é alcançada através do uso inteligente de links simbólicos, como mostra em seu site Pnpm (2024a). Em vez de copiar pacotes repetidos para cada projeto que os utiliza, o pnpm cria links simbólicos que apontam para um único local centralizado onde o pacote é armazenado. Essa estratégia evita o desperdício de espaço em disco, especialmente em projetos que utilizam muitas

dependências, tornando-o uma excelente opção para otimizar o uso de recursos.

Além da economia de espaço, o pnpm também se destaca por sua estrutura de pastas não plana. Diferentemente de outros gerenciadores de pacotes que criam uma estrutura de pastas aninhadas complexa, o pnpm organiza as dependências de forma mais eficiente, resultando em um projeto mais organizado e fácil de navegar. Essa estrutura também contribui para um melhor desempenho, pois o pnpm pode realizar operações de instalação e atualização de forma mais rápida e eficiente.

No que diz respeito à segurança, o pnpm também apresenta vantagens significativas. Sua estrutura de dependências, que evita a duplicação de pacotes e centraliza o armazenamento, reduz a "superfície de ataque" da aplicação. Com menos cópias de pacotes espalhadas pelo sistema de arquivos, há menos pontos de entrada para potenciais vulnerabilidades.

Essa característica, combinada com a capacidade de gerar um lockfile confiável que garante a instalação consistente das dependências, contribui para um ambiente de desenvolvimento mais seguro e protegido contra ameaças.

3.6.4 Bun

Bun é um novo gerenciador de pacotes, construído com foco em velocidade e desempenho. Sua arquitetura e recursos integrados, como transpilador e servidor de desenvolvimento, o tornam uma opção promissora para projetos que buscam agilidade. (BUN, 2024)

Em relação à segurança, o Bun ainda está em desenvolvimento ativo e sua comunidade está crescendo, o que significa que a identificação e correção de vulnerabilidades podem ser mais rápidas. No entanto, é importante ter cautela e acompanhar o desenvolvimento, devido a sua pouca idade do projeto para garantir a segurança do seu código.

3.6.5 Comparativos

Em 2024,Pnpm (2024b), o cenário dos gerenciadores de pacotes JavaScript se mostra dinâmico e competitivo, com Yarn, pnpm e npm disputando a preferência dos desenvolvedores. Cada um possui características próprias, mas o pnpm tem se destacado como uma opção atraente, especialmente para quem busca desempenho, eficiência e segurança.

Enquanto o npm, o gerenciador padrão do Node.js, ainda é amplamente utilizado, sua performance em termos de velocidade de instalação e gerenciamento de dependências pode ser superada pelo Yarn e, principalmente, pelo pnpm. O Yarn, por sua vez, oferece recursos interessantes como o cache offline e a instalação paralela de pacotes, mas o pnpm

se sobressai com seu sistema de cache ainda mais eficiente e o uso de links físicos, que economizam espaço em disco e aceleram as instalações.

Além da velocidade, o pnpm se destaca pelo gerenciamento inteligente de dependências e por ser mais seguro. Ao evitar a duplicação de pacotes e criar uma estrutura de node_modules mais enxuta, ele contribui para um projeto mais organizado e com menor consumo de espaço em disco. Além disso, o pnpm garante que cada pacote acesse apenas suas próprias dependências, reduzindo o risco de execução de código não autorizado e vulnerabilidades. Recursos como o modo "workspace" para gerenciar múltiplos projetos e o filtro de dependências para instalar apenas o necessário também agregam valor ao pnpm.

Embora o npm conte com a maior comunidade e o Yarn também possua uma base sólida de usuários, a comunidade do pnpm vem crescendo rapidamente, com documentação abrangente e suporte ativo nos fóruns online. Isso garante que os desenvolvedores que optarem pelo pnpm terão acesso a recursos e ajuda quando necessário.

Essa superioridade do pnpm em relação aos seus concorrentes fica evidente no gráfico comparativo, figura 3, presente na documentação oficial do pnpm. O gráfico compara o desempenho dos gerenciadores de pacotes npm, Yarn e pnpm em diversos cenários. Em quase todos os testes, o pnpm se destaca com tempos de execução significativamente menores, especialmente em instalações "clean" e repetidas, demonstrando sua eficiência em lidar com o cache e a resolução de dependências. Em alguns casos, o pnpm é até 10 vezes mais rápido que o npm e consideravelmente mais veloz que o Yarn. Essa performance superior é um dos principais motivos para a crescente adoção do pnpm em 2024, tornando-o uma escolha atraente para projetos que buscam agilidade e otimização no gerenciamento de dependências.

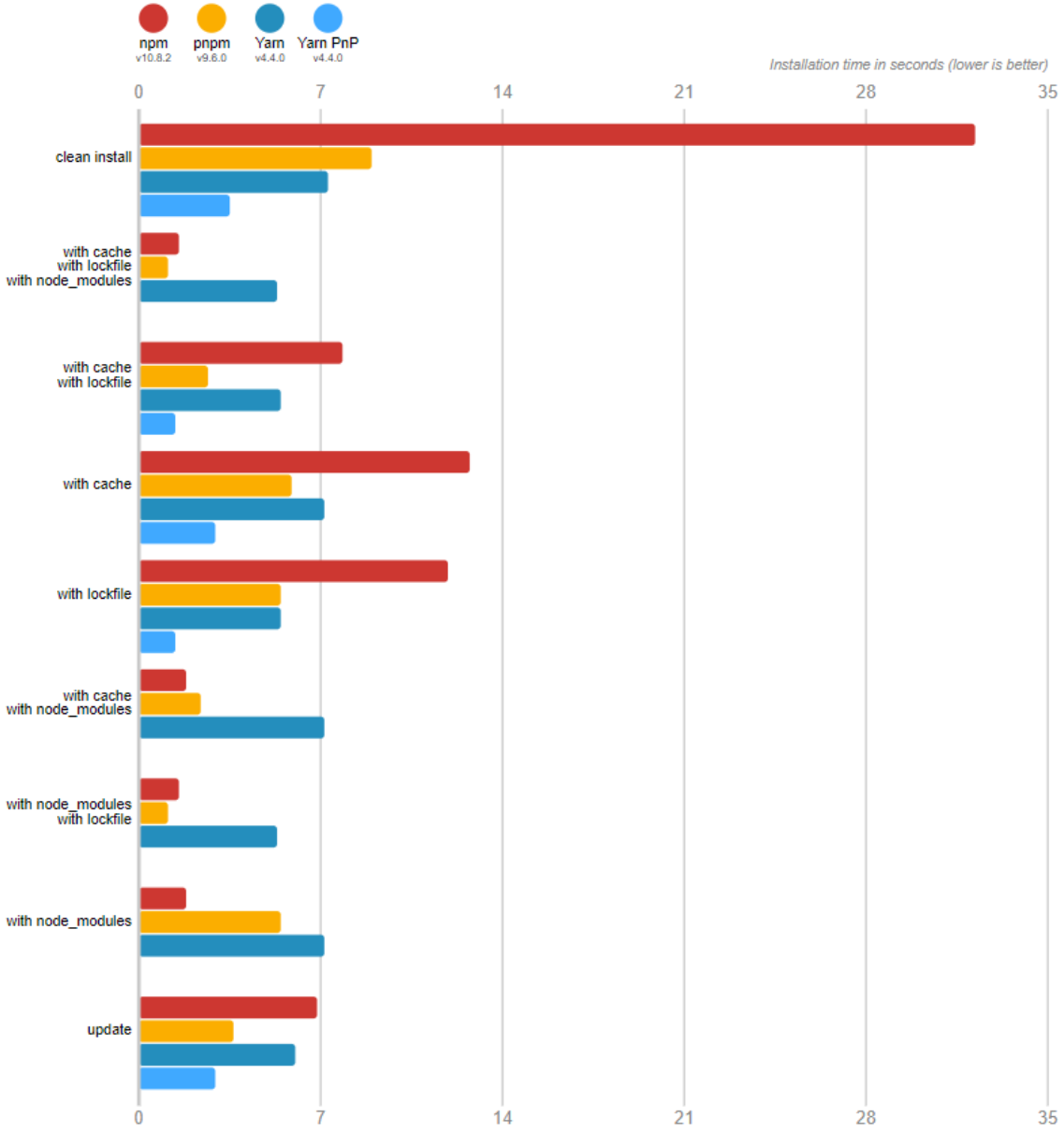

Figura 3 – Benchmark dos gerenciadores. Fonte: pnpm docs

3.7 Create React App

Create-React-App (CRA) é uma ferramenta popular para criar aplicativos React. Ele fornece um ambiente pré-configurado com tudo necessário para começar a desenvolver um aplicativo React, incluindo um servidor de desenvolvimento, um sistema de construção e um conjunto de ferramentas básicas.

Vantagens

- Fácil de se usar
- Facilita a criação de projetos pequenos.

Desvantagens

- CRA é um pacote grande, o que pode tornar o desenvolvimento lento.
- Conhecida por ser lento, especialmente durante o processo de desenvolvimento.
- CRA se tornou depreciado, o que pode levar a problemas de segurança e compatibilidade.
- Pacotes desatualizados podem levar a vulnerabilidades de segurança.
- Pode ter problemas com alguns *plugins*, o que pode dificultar o desenvolvimento de aplicativos complexos.

3.8 Conclusão

Em resumo, este capítulo explorou as tecnologias e teorias que serão empregadas no desenvolvimento deste projeto. A partir da análise comparativa, optou-se pela utilização do Next.js 14, um framework react sugerido em sua documentação oficial. O gerenciador de pacotes pnpm será adotado para garantir a eficiência e organização do projeto, aproveitando seus recursos de economia de espaço e velocidade. O protótipo será elaborado utilizando Figma seguindo as heurísticas de Nielsen.

Com a base teórica e tecnológica definida, o próximo capítulo detalhará a implementação prática do projeto, demonstrando como as ferramentas e conceitos aqui apresentados serão aplicados para alcançar os objetivos propostos.

4 Elaboração do Design

Este capítulo detalha o processo de elaboração do projeto de design da nova interface do InterREP Manager. O objetivo principal foi criar uma experiência de usuário mais intuitiva, eficiente e esteticamente agradável, alinhada com as heurísticas de Nielsen, 3.2, para usabilidade. A pesquisa de referências, a definição de cores e a criação das telas foram etapas cruciais para alcançar esse objetivo.

4.1 Pesquisa de Referência

Para buscar e identificar boas práticas de design, foram analisadas duas referências principais: A primeira referência foi a da versão original do produto disponibilizado em seu site.

Conforme a figura 4 que apesar do projeto obedecer algumas das heurísticas, capítulo 3.2, como a compatibilidade com o mundo real, ele não é muito bom em manter um design minimalista e não evita erros devido ao seu design.

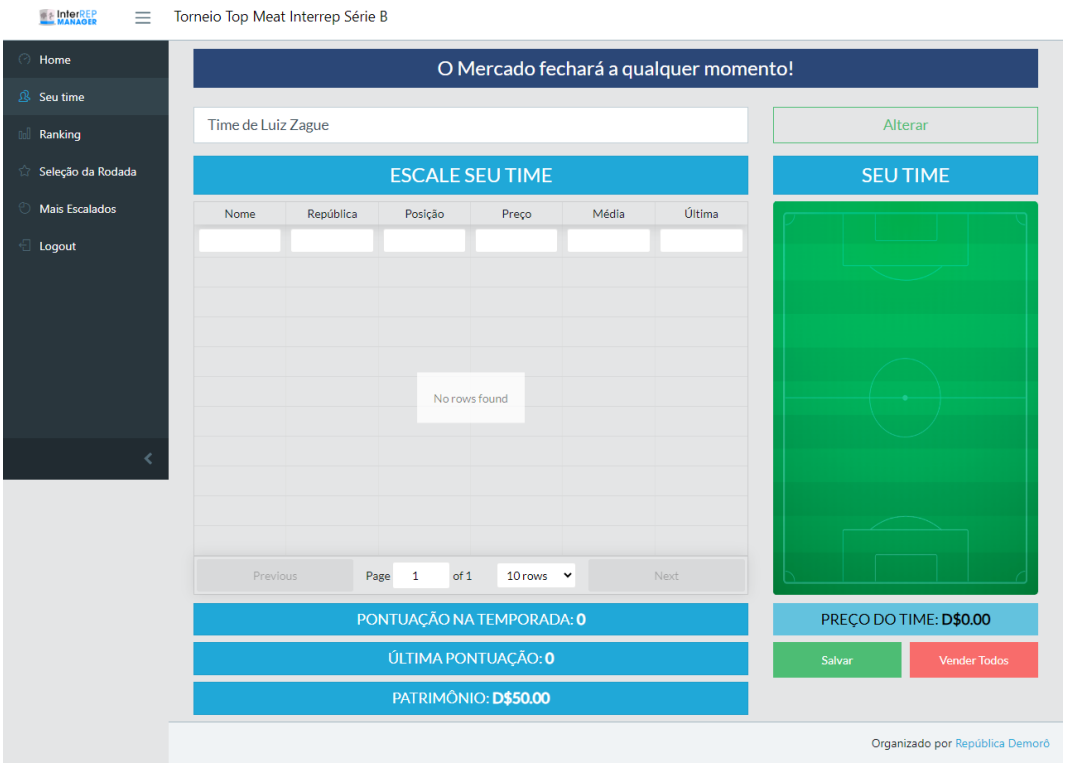

Figura 4 – Tela inicial do projeto original. Fonte: projeto inicial

A segunda referência utilizada foi o Cartola do grupo Globo. O maior fantasy game brasileiro da atualidade. Na figura 5 pode-se perceber que um estilo mais limpo e intuitivo.

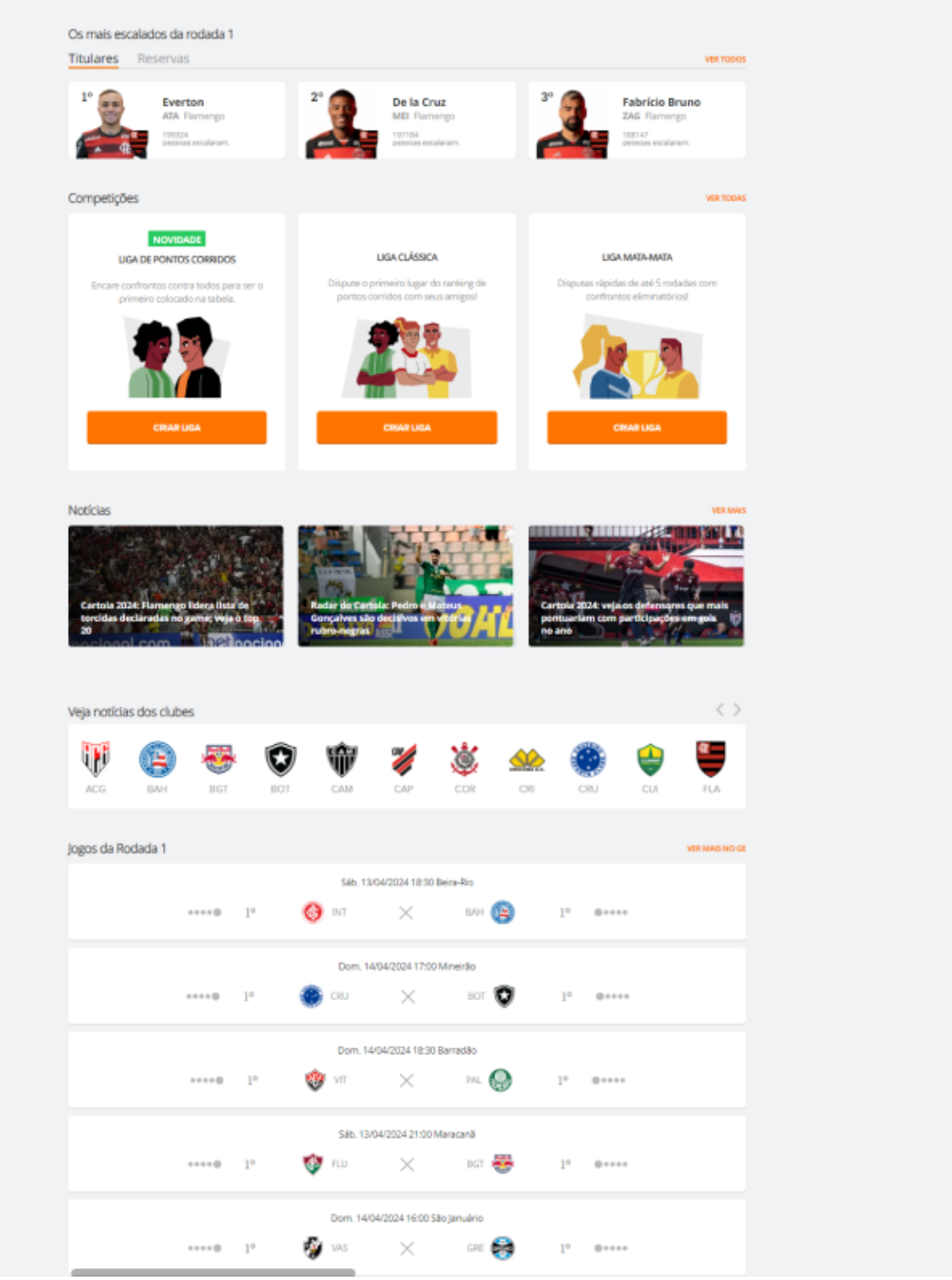

Figura 5 – Tela inicial do cartola. Fonte: Cartola Fc

4.1.1 Analise da versão original

A versão original do InterREP Manager infringia algumas heurísticas e boas práticas e por esse motivo houve a necessidade de um novo design. Entre os problemas encontrados foram:

- Tela de Login:
  - A falta de responsividade resultava em campos de entrada pequenos e sobreposição de elementos, dificultando o uso em diferentes dispositivos.
  - A ausência de feedback durante a criação de um usuário violava a heurística de "Visibilidade do Status do Sistema", deixando o usuário sem saber se a ação foi bem-sucedida.
- Tela Inicial de Escalação:
  - A impossibilidade de remover um jogador escalado e a falta de um botão de confirmação feriam as heurísticas de "Controle e Liberdade do Usuário" e "Prevenção de Erros".
  - A sobreposição de elementos no cabeçalho da página prejudicava a estética e a clareza da interface.
- Navegação:
  - A navegação confusa entre torneios violava a heurística de "Reconhecimento ao invés de Memorização", dificultando a localização de informações.
- Área do Administrador:
  - Os mesmos problemas de usabilidade encontrados nas telas dos usuários comuns também estavam presentes na área do administrador.
  - A presença de um botão visível para a área do administrador na interface do usuário comum representava um risco de segurança, possibilitando acessos não autorizados.

4.2 Criação das telas

Antes de iniciar o processo de design, foi fundamental entender as necessidades dos usuários e as funcionalidades esperadas para o InterREP Manager. Foram definidos dois perfis de usuário.

Para isso foram realizadas reuniões quinzenais com os organizadores durante toda a elaboração do novo design.

4.2.1 Definição de funcionalidade

Para o usuário padrão, ele poderá:

1. Na tela inicial, ver campanhas, jogos e seu time principal;
2. Escalar times para as três divisões, masculino série A e B e feminino.
3. Ver sua pontuação geral
4. Criar ligas com seus amigos a fim de criar rankings privados
5. Editar informações do seu perfil

Para o usuário administrador, ele poderá:

1. Abrir o mercado;
2. Criar, editar e visualizar todas as partidas.
3. Configurar as repúblicas participantes
4. Colocar cada *scouts* de cada jogador para que seja feito o cálculo de sua pontuação
5. Criar, editar e visualizar todos jogadores
6. Criar, editar e visualizar todas as campanhas
7. **Importante:** o usuário administrador não poderá escalar jogadores como o usuário comum,

4.2.2 Definição de cores

A paleta de cores foi definida em conjunto com os organizadores do torneio optando-se por utilizar as cores do sistema Tailwind CSS em conjunto com o Moon Design System, figura 6.

A paleta resultante prioriza o contraste e a acessibilidade, com o neutral 50 como cor de fundo principal, o branco como cor secundária e o blue 800 para botões e elementos de destaque. Os textos são exibidos em neutral 950 para garantir legibilidade sobre o fundo claro.

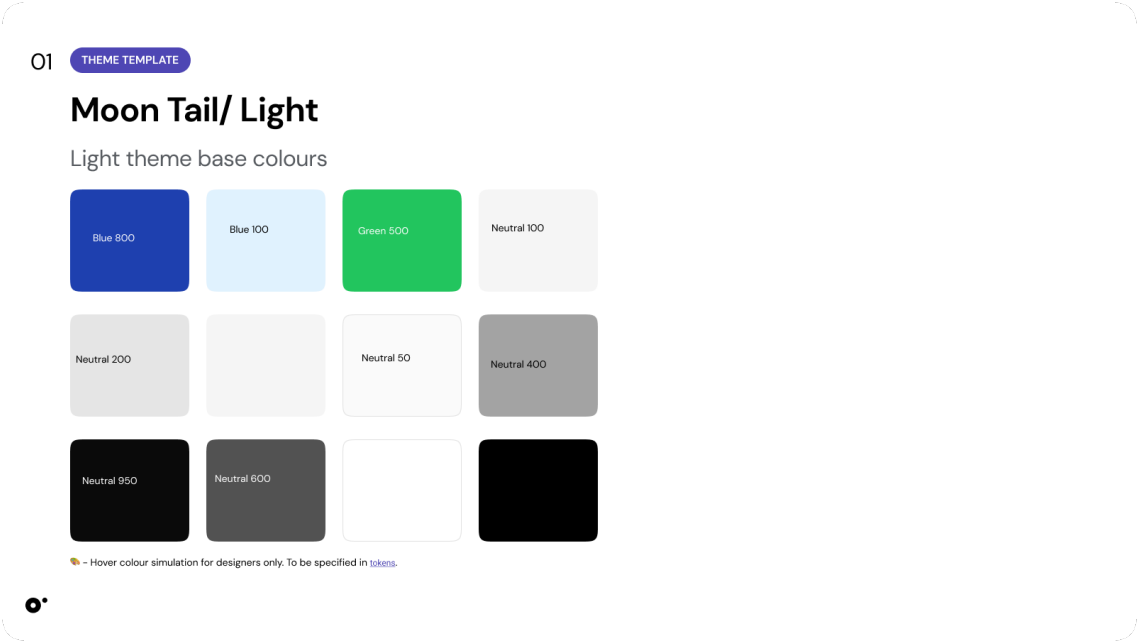

Figura 6 – Paletas de Cores. Fonte: autor

Figura 7 – Componente do Figma. Fonte: autor

4.2.3 Reutilização de componentes

Com a definição da usabilidade, pesquisa de referências e paleta de cores, iniciou-se o processo de criação das telas Figma. Para garantir a consistência e eficiência do design, foram utilizados componentes reutilizáveis, como o componente de jogos apresentado na figura 7, facilitando assim a quarta e sexta heurística de Nielsen descritos em 3.2

A figura 7 mostra o componente criado para mostrar as partidas marcadas dentro do sistema. Dessa forma, é possível replicar em todo projeto sem a necessidade de recriar do zero o mesmo elemento.

4.2.4 Prototipação

Concluído o design com os principais componentes utilizados, foi iniciada a prototipação das telas no Figma. Para que seja possível visualizar o fluxo a ser seguido durante o desenvolvimento.

Ambas jornadas começam iguais, através do login e é redirecionado para a Home, ambos possuem um side bar e um footer. No entanto, a diferença está no que foi descrito em 4.2.1.

4.2.5 Comparativo com o Original

A figura 8 mostra a nova página inicial da aplicação. O design mais limpo permite que o usuário tenha uma experiência melhor na navegação.

Nela, são mostradas campanhas criadas a pedido de fornecedores e parceiros do torneio, seu principal time e os próximos jogos.

O usuário consegue navegar tanto pelo menu de navegação, que contem ícones respeitando a heurísticas discutidas anteriormente, quanto pelo clicando em ver todos afim de ver os próximos jogos ou os seus times escalados.

Na figura 4 o usuário escalava dentro da tela inicial. No novo projeto, essa tela mudou para uma tela própria, figura 9

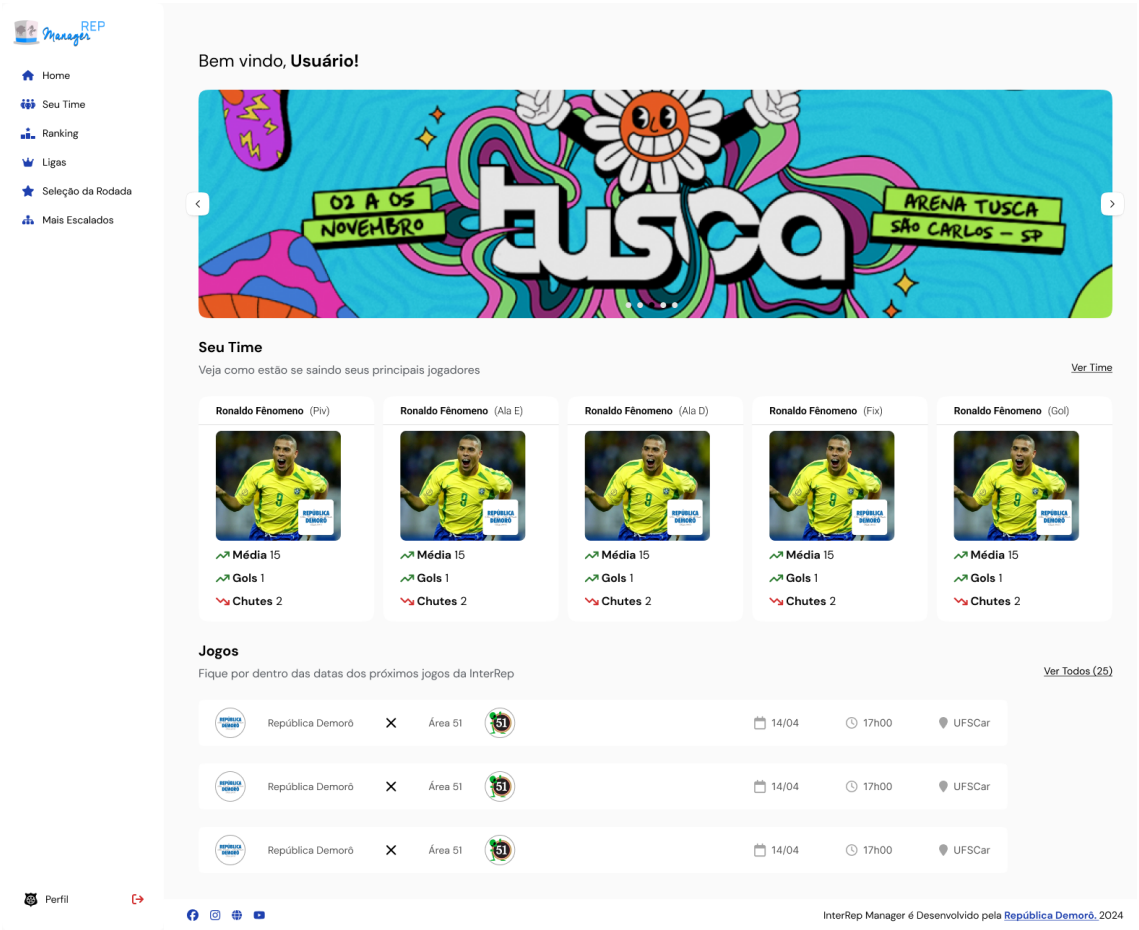

Figura 8 – Nova tela inicial. Fonte: autor

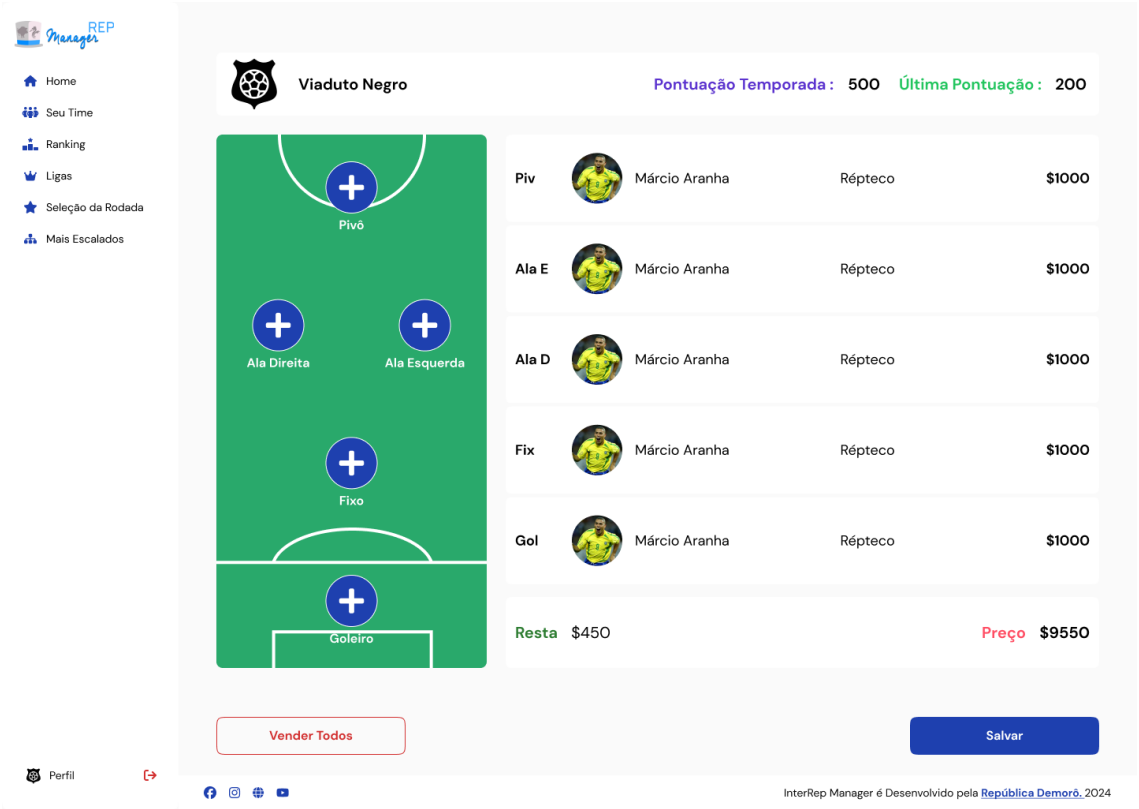

Figura 9 – Nova tela de escalação. Fonte: autor

Com ambas figuras é possível perceber um melhor emprego das heurísticas de Nielsen. Além de ambos os designs serem mais limpos a criação de uma página dedicada a navegação de torneio ajuda no "Reconhecimento ao invés de Memorização" a presença de botões de confirmação e ajuda na prevenção de erros corrigindo os problemas com as heurísticas citadas em 4.1.1.

Assim como o modelo original é utilizado botões com símbolos auxilia "Compatibilidade do Sistema com o Mundo Real"

4.2.6 Tabela

A tabela 2 sintetiza o método utilizado para a analise do antigo design

Critério de Avaliação	**Antigo Modelo**	**Novo Design**	**Melhoria**
Visibilidade do Status do Sistema	Violação na criação de usuário (sem feedback)	Feedback claro durante o cadastro	Sim
Compatibilidade com o Mundo Real	Adequado (uso de ícones e termos comuns)	Mantido e aprimorado	Sim
Controle e Liberdade do Usuário	Violação na escalação (sem remoção ou confirmação)	Botões de ação e confirmação claros	Sim
Consistência e Padrões	Problemas de inconsistência e navegação confusa	Padrões de design e navegação aprimorados	Sim
Prevenção de Erros	Violação na escalação (sem confirmação)	Botões de confirmação e validação de dados	Sim
Reconhecimento ao invés de Memorização	Violação na navegação entre torneios	Navegação clara e intuitiva	Sim
Flexibilidade e Eficiência de Uso	Problemas de usabilidade em diferentes perfil	Design responsivo e adaptado a cada perfil	Sim
Estética e Design Minimalista	Problemas de sobreposição e clareza	Layout limpo e organizado	Sim
Suporte para Erros	Não abordado explicitamente	Mensagens de erro claras e informativas (em desenvolvimento)	Sim
Ajuda e Documentação	Não abordado explicitamente	Pode ser aprimorado em versões futuras	Potencial de melhoria
Segurança	Botão de acesso de administrador visível. Login simples	Login mais robusto e melhor proteção da área de administrador	Sim
Responsividade	Ausente em algumas páginas, causando problemas de usabilidade	Design responsivo para diferentes dispositivos	Sim

Tabela 2 – Benchmark

O novo design apresenta melhorias significativas em relação ao antigo modelo em quase todos os critérios de avaliação, demonstrando uma forte adesão às Heurísticas de Nielsen e boas práticas de design. A usabilidade, a estética e a segurança foram aprimoradas, proporcionando uma experiência de usuário mais intuitiva, eficiente e agradável.

4.3 Autenticação e autorização

No desenvolvimento de qualquer aplicação, a segurança é um pilar fundamental que garante a proteção dos dados dos usuários e a integridade do sistema. No contexto do InterREP Manager, a implementação de medidas robustas de segurança é essencial para garantir a confiabilidade da plataforma e a privacidade das informações dos usuários.

A autenticação e a autorização são os primeiros guardiões da segurança do sistema. A tela inicial do projeto é a tela de login. Para acessar o aplicativo é preciso que o usuário coloque sua senha e seu e-mail, conforme mostra a figura 10. Nela o campo de senha deve conter um estado que possa permitir que o usuário possa controlar a visibilidade de sua senha, dessa forma protegendo-a contra o roubo e permitindo que o usuário possa verificar se a digitou corretamente. Ademais, nessa tela existe a possibilidade de troca de senha e de cadastrar um novo usuário.

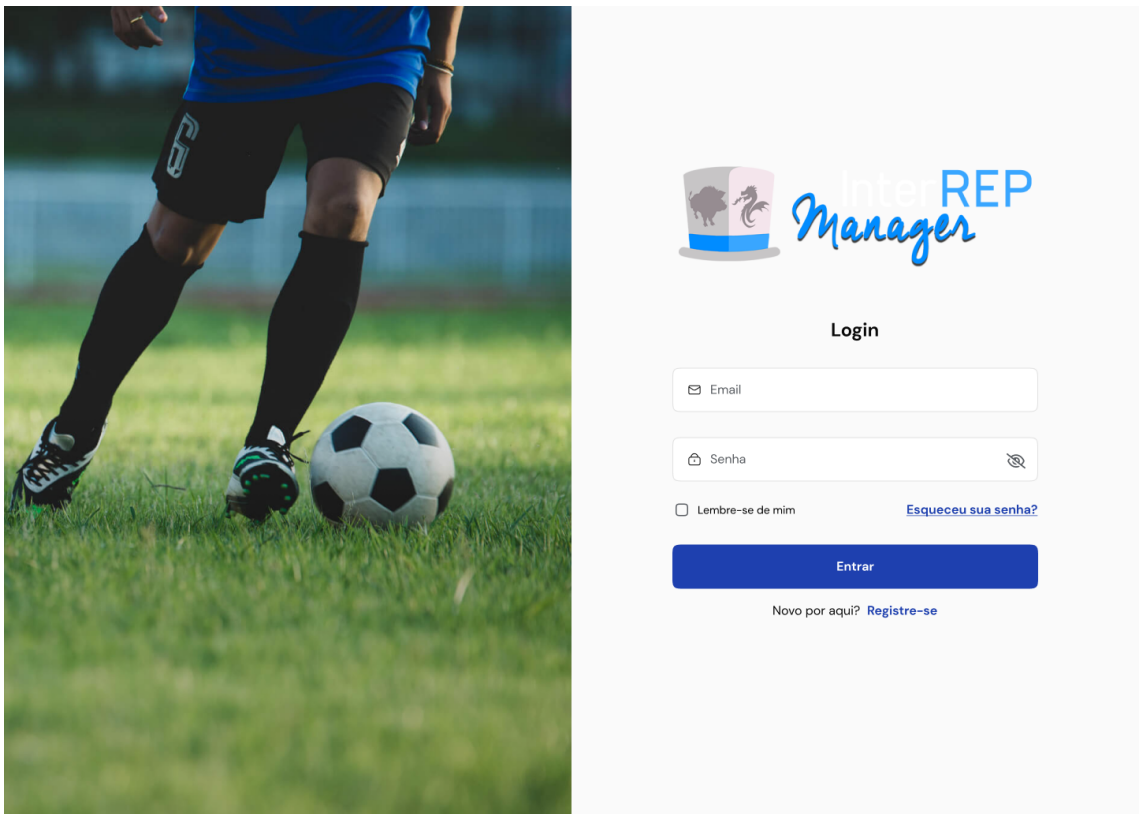

Figura 10 – Tela de Login. Fonte: autor

As figuras 11 e 12 são as etapas de troca de senha para um usuário. Nela é preciso que o usuário digite e-mail dele para receber um link de prazo curto para troca de senha.

Ao acessar este link,o usuário será encaminhado para a tela da figura 12 e digite sua nova senha e a confirme

Figura 11 – Tela de envio de e-mail para troca de senha. Fonte: autor

Figura 12 – Tela de nova senha. Fonte: autor

A última tela relacionada a autenticação é o de cadastro que, conforme a figura 13, segue o mesmo padrão das outras de proteger a senha.

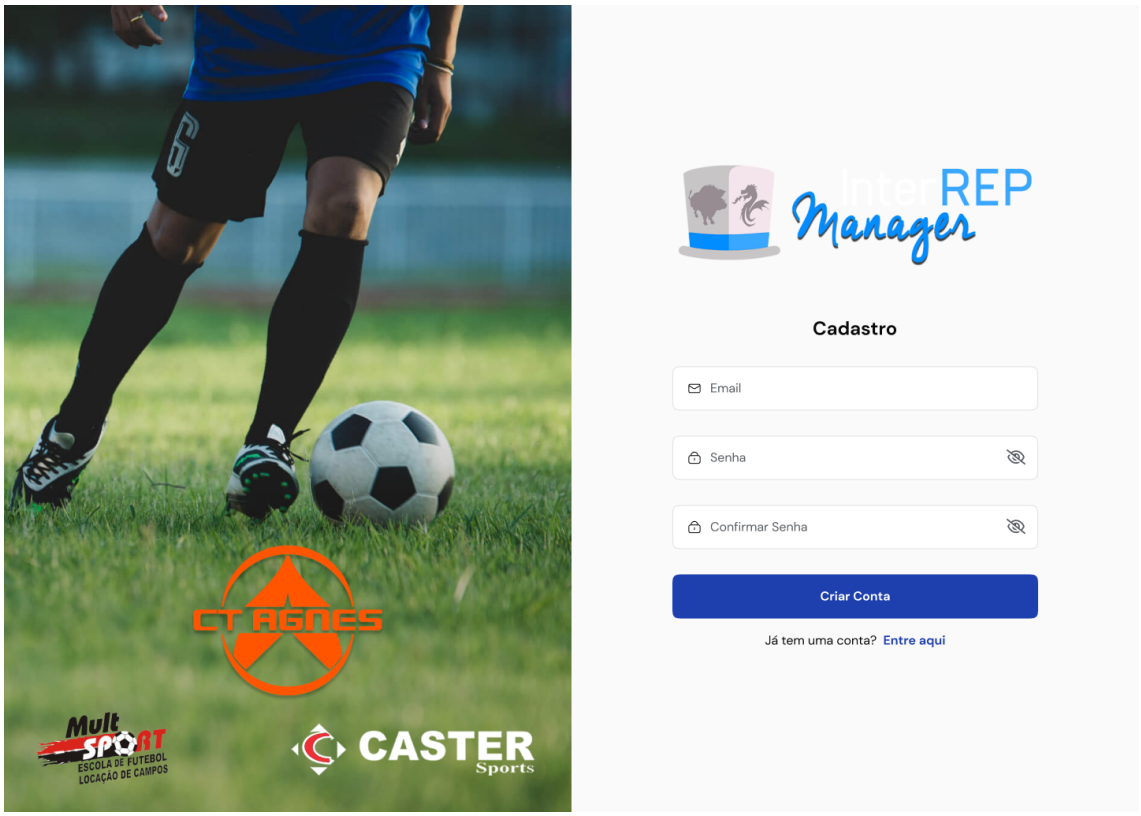

Figura 13 – Tela de cadastro. Fonte: autor

4.3.1 Responsabilidades

A definição de diferentes perfis de usuário, como “usuário comum” e “administrador”, é crucial para garantir a segurança e a integridade do sistema. Cada perfil deve ter acesso apenas às funcionalidades e informações relevantes para suas atribuições, como discutido em 4.2.1. O administrador tem acesso privilegiado as funcionalidades do sistema, incluindo gerenciamento de usuários, configuração de times, partidas e campanhas, e acesso a dados sensíveis. No entanto, não poderá participar do torneio, a fim de manter a integridade. O administrador tem acesso restrito às suas próprias informações e ações relacionadas ao jogo, como escalação de times, participação em ligas e visualização de rankings.

5 Criação do front end

Antes de começar a discutir a implementação é preciso discutir a arquitetura do framework escolhido em 3.8, além das bibliotecas utilizadas para auxiliar o desenvolvimento.

5.1 Next.js

Next.js é um framework React para construir aplicações web completas e com maior performance. Ele oferece diversos recursos que facilitam a experiência de desenvolvimento, desde a criação de rotas e renderização até a otimização de imagens e estilos. Algumas das características são:

- Sistema de roteamento baseado no app router que suporta layouts, roteamento aninhado, estados de carregamento e muito mais.
- Renderização do lado do servidor (SSR) e renderização estática (SSG) para otimizar o desempenho e a experiência do usuário.
- Suporte a diversos métodos de estilização, incluindo CSS Modules, Tailwind CSS e CSS-in-JS.
- Otimizações para imagens, fontes e scripts para melhorar as métricas essenciais da web e a experiência do usuário.
- Suporte aprimorado para TypeScript com melhor verificação de tipos e compilação mais eficiente.

5.1.1 Server Components

Os Server Components (RSCs) são um recurso no Next.js que permite a renderização de componentes no servidor, combinando os benefícios da renderização do lado do servidor (SSR) com a flexibilidade do React como explicado em 3.5.4.

O padrão Island no Next.js 14 permite a coexistência de Server Components e Client Components em uma mesma aplicação. Para utilizar componentes que precisam ser executados no navegador, como aqueles que lidam com interatividade ou gerenciam estado local, o Next.js introduziu a diretiva “use client”. Essa diretiva é utilizada para indicar explicitamente que um componente deve ser tratado como um Client Component, isolando-o e seus filhos. Ao adicionar no topo de um arquivo de componente, indica-se ao Next.js que ele deve ser tratado como um Client Component e, portanto, renderizado e

executado no navegador do usuário, figura 14. Essa flexibilidade permite que se combine as funcionalidades dos Server Components com a interatividade dos Client Components.

5.1.2 Cache

Next.js 14 introduz um sistema de cache híbrido, combinando o cache do lado do servidor (armazenando resultados de dados, rotas completas e memorização de requisições) com o cache do lado do cliente (armazenando payloads de Server Components e permitindo regeneração incremental) Vercel (2024). Essa abordagem otimiza o desempenho, reduzindo a carga no servidor e melhorando a velocidade de carregamento e a responsividade para o usuário. O sistema é configurável, permitindo ajustar tempos de expiração, tags de cache e revalidação sob demanda.

O Next.js 14 oferece um sistema de cache flexível para otimizar a busca de dados em APIs. A opção no-cache, exemplificada no código da figura 15, garante que os dados de usuários sejam sempre buscados diretamente da API, ignorando qualquer cache existente. Isso é útil para garantir a atualização constante de informações sensíveis.

Além disso, o Next.js oferece outras estratégias de cache: force-cache armazena indefinidamente a resposta, ideal para dados estáticos; revalidate revalida o cache em segundo plano após um tempo definido, útil para dados que mudam com frequência; e no-store nunca armazena a resposta, essencial para dados altamente sensíveis.

A escolha da estratégia e do tempo de expiração do cache permite equilibrar a velocidade de carregamento com a precisão dos dados. O tempo de expiração padrão é definido pelo cabeçalho Cache-Control da API, mas pode ser personalizado globalmente ou por rota no Next.js.

5.1.3 App Router

App Router, uma inovação do Next.js 14, redefine a construção de aplicações web, priorizando desempenho, experiência do usuário e flexibilidade. Diferentemente de Pages Router, baseado em arquivos e pastas para definir rotas, App Router introduz um sistema de componentes interconectados para estruturar a aplicação. Essa mudança de paradigma permite um controle mais granular sobre roteamento, *layouts* e carregamento de dados, viabilizando recursos avançados que otimizam performance e experiência do usuário. (VERCEL, 2024)

Um pilar de App Router são os Server Components, que possibilitam a renderização de componentes diretamente no servidor. Também suporta o Streaming com Suspense, permitindo o carregamento progressivo do conteúdo da página. Enquanto os dados de um componente são buscados ou processados, o App Router exibe um estado de carregamento ou conteúdo parcial, proporcionando uma experiência mais fluida.

```tsx
"use client";
/**
 * imports
*/
const createCampaingSchema = z.object({
  name: z.string().min(1, "Nome do Time é obrigatório"),
  link: z.string().url(),
  photo: z.string(),
  beginDate: z.date({
    required_error: "A data de ínicio é necessária",
  }),
  endDate: z.date({
    required_error: "A data de fim é necessária",
  }),
});

type CreateCampaingData = z.infer<typeof createCampaingSchema>;

export default function CreateCampaingForm() {
  const methods = useForm<CreateCampaingData>({
    resolver: zodResolver(createCampaingSchema),
  });
  const {
    register,
    control,
    handleSubmit,
    formState: { errors },
    getValues,
  } = methods;

  const onSubmit = (data: CreateCampaingData) => {
    const formDataToSend = new FormData();
    for (const [key, value] of Object.entries(data)) {
      formDataToSend.append(key, value as string | Blob);
    }
    console.log(formDataToSend);
    const fileInput = document.getElementById(
      "photo"
    ) as HTMLInputElement | null;
    if (fileInput?.files && fileInput.files.length > 0) {
      formDataToSend.append("photo", fileInput.files[0]);
    }
    for (const [key, value] of formDataToSend.entries()) {
      console.log(key, value);
    }
  };
  const handlePhotoChange = (event: React.ChangeEvent<HTMLInputElement>) => {
    const files = event.target.files;
    if (files && files.length > 0) {
      const selectedFile = files[0];
      console.log("Selected File:", selectedFile);
    }
  };
  const { field } = useController({
    name: "photo", control, defaultValue: "",
  });

  return (
    <FormProvider {...methods}>
      <form onSubmit={handleSubmit(onSubmit)} className="space-y-6">
        <div className="flex flex-col items-center justify-center gap-4">
          <div className="w-full flex flex-col gap-1">
            <label htmlFor="name" className="self-start">
              Nome
            </label>
```

Figura 14 – Exemplo de utilização de um Client Component. Fonte: autor

```
async function getCampaign(): Promise<Campaign[]> {
  try {
    const res = await fetch(
      `https://667e1d1d297972455f6723ea.mockapi.io/campaign`,
      {
        cache: "no-cache",
        // next: { revalidate: 259200 },
      }
    );

    if (!res.ok) {
      throw new Error(`Failed to fetch team data. Status: ${res.status}`);
    }

    const data = await res.json();
    return data;
  } catch (error) {
    console.error("Error fetching team:", error);
    return [];
  }
}
```

Figura 15 – Exemplo de cache na requisição de um fetch. Fonte: autor

Além disso, o App Router suporta Server Actions, funções executadas no servidor em resposta a eventos do usuário. Essa funcionalidade possibilita a criação de interações complexas e personalizadas, como formulários de contato, sistemas de comentários e outras ações que exigem processamento no lado do servidor.

5.1.3.1 page.tsx

Em uma aplicação Next.js 14 com App Router, o arquivo page.tsx representa uma rota individual e define o conteúdo principal exibido em cada página. Ele é responsável por buscar dados, se necessário, e renderizar o conteúdo específico daquela rota. Cada arquivo page.tsx corresponde a uma URL única e é o ponto de entrada para a lógica e interface de cada página da aplicação.

5.1.3.2 layout.tsx

O arquivo layout.tsx define o layout compartilhado por várias páginas, como cabeçalho, rodapé e elementos de navegação. Ele envolve o componente da página (page.tsx) e fornece uma estrutura consistente para toda a aplicação. Cada arquivo layout.tsx é renderizado apenas uma vez e persiste entre as navegações, garantindo uma transição suave e mantendo o estado dos elementos do layout.

5.1.3.3 loading.tsx

O arquivo loading.tsx é responsável por exibir um estado de carregamento enquanto os dados da página estão sendo buscados ou processados. Ele melhora a experiência do usuário, fornecendo feedback visual durante o carregamento, evitando que a página fique

em branco ou pareça quebrada. O loading.tsx é renderizado automaticamente pelo App Router quando um componente da página (page.tsx) está sendo carregado.

5.1.3.4 not-found.tsx

O arquivo not-found.tsx entra em ação quando uma rota solicitada pelo usuário não corresponde a nenhuma página existente na aplicação. Ele permite que o desenvolvedor personalize a página de erro 404, exibindo uma mensagem informativa, sugestões de navegação ou qualquer outro conteúdo relevante para orientar o usuário e evitar que ele abandone o site.

Na figura 16 foi criada uma página customizada para os erros 404. Dessa forma, criando um experiência melhor para o usuário.

```
1  import { Logo } from "@/components/logo";
2  import Link from "next/link";
3
4  export default function NotFound() {
5    return (
6      <div className="flex justify-center h-screen">
7        <div className="flex flex-col m-auto space-y-5 text-center">
8          <Logo />
9          <h1 className="text-3xl font-bold ">Não Encontrado</h1>
10         <p>Parece que a página não existe</p>
11         <Link href="/manager/" className="text-blue-800">
12           Volte para página inicial
13         </Link>
14       </div>
15     </div>
16   );
17 }
```

Figura 16 – Página not-found customizada. Fonte: autor

5.1.3.5 error.tsx

O arquivo error.tsx é responsável por lidar com erros que ocorrem durante a renderização de uma página. Esses erros podem ser causados por falhas na busca de dados, problemas de lógica no componente ou outras exceções inesperadas. O error.tsx permite que o desenvolvedor personalize a página de erro, exibindo informações sobre o erro (em ambiente de desenvolvimento) ou uma mensagem genérica e amigável ao usuário (em ambiente de produção), evitando que a aplicação quebre e oferecendo uma experiência mais agradável mesmo em situações de erro.

5.1.3.6 Server Actions

No contexto do Next.js, “Server Actions” são funções que rodam exclusivamente no servidor. Elas permitem que execute lógicas sensíveis ou que requerem acesso a recursos do servidor, como bancos de dados ou sistemas de arquivos, de forma segura e eficiente.

```
1  "use server";
2
3  import { cookies } from 'next/headers';
4  import { redirect } from 'next/navigation';
5
6  export async function logoutAction() {
7    const cookiesList = cookies();
8    cookiesList.delete('authToken');
9    cookiesList.delete('isAdmin');
10   redirect('/login');
11 }
```

Figura 17 – Exemplo de Server Action. Fonte: autor

A diretiva “use server”; no início do código da figura 17, indica ao Next.js que a função logoutAction é uma Server Action. Isso garante que ela seja executada apenas no servidor, protegendo informações confidenciais e evitando que o código seja exposto no navegador do cliente.

No exemplo fornecido, a logoutAction é responsável por encerrar a sessão do usuário. Ela acessa os cookies do navegador, remove os cookies de autenticação (authToken e isAdmin), e redireciona o usuário para a página de login. Essa lógica é crucial para garantir a segurança da aplicação, pois lida com informações sensíveis relacionadas à identidade do usuário.

5.1.4 Metadata

No Next.js 14, metadados são informações cruciais que descrevem e otimizam o comportamento de uma página web. Eles são definidos em um componente especial chamado <Metadata>, figura 18, e podem incluir elementos como título da página, descrição, palavras-chave para SEO, links para fontes e folhas de estilo, e outras tags relevantes.

Na figura 18, são configurados metadados específicos relacionada ao InterREP Manager. O título padrão é InterREP Manager, mas um modelo permite criar títulos dinâmicos, inserindo conteúdo específico da página no lugar de %s. A descrição detalhada destaca os atrativos do fantasy game, enquanto as palavras-chave auxiliam os mecanismos de busca a entender o tema da página. Adicionalmente, definiu-se o formato de exibição do card para o Twitter.

Ao realizar essa inserção de metadados no layout na raiz do projeto é garantido que todas páginas estejam configuradas. É possível sobrescrever certas informações, como feito na página inicial de administrador em que o *title* recebe admin ficando admin | InterREP Manager.

Esses metadados são renderizados no cabeçalho HTML da página, tornando-se visíveis para mecanismos de busca e navegadores. Isso melhora a indexação do site e a

```tsx
export const metadata: Metadata = {
  title: {
    default: "InterREP Manager ",
    template: " %s | InterREP Manager",
  },
  description:
    "Mergulhe no mundo do fantasy game do InterREP!" +
    " Escale seu time dos sonhos com os craques do torneio, " +
    "acompanhe as partidas em tempo real e dispute o topo do ranking com seus amigos."+
    "Viva a emoção do InterREP como nunca antes!",
  keywords:
    "InterREP, fantasy game, futebol society, São Carlos, USP, UFSCar, IFSP",
  twitter: {
    card: "summary_large_image",
  },
  openGraph: {
    images: '/opengraph-image.png',
  },
  metadataBase: new URL('http://localhost:3000')
};
```

Figura 18 – Exemplo Metadata. Fonte: autor

exibição de informações relevantes nos resultados de pesquisa. Além disso, o Next.js 14 oferece flexibilidade para personalizar e controlar os metadados em cada página, permitindo uma otimização granular para diferentes seções do site.

5.1.5 Robots e sitemap

O Next.js 14 oferece recursos para gerenciar o acesso de robôs de busca ao site e fornecer um mapa detalhado do seu conteúdo, otimizando ainda mais os resultados de busca online. O arquivo robots.ts permite controlar quais partes do seu site os robôs podem acessar, enquanto o sitemap.ts fornece um índice estruturado das suas páginas, facilitando a indexação e a descoberta do seu conteúdo pelos mecanismos de busca.

Nas Figuras 19 e 20, Next.js 14 possibilita a criação dinâmica desses arquivos essenciais. Através de funções específicas, pode-se gerar robots.txt e sitemap.xml em tempo real, garantindo que eles estejam sempre atualizados com a estrutura e o conteúdo do seu site, mesmo em aplicações com rotas dinâmicas ou conteúdo gerado em tempo de execução. Essa abordagem dinâmica elimina a necessidade de atualizar manualmente esses arquivos sempre que houver alterações no seu site, simplificando o processo e garantindo a máxima eficiência na comunicação com os robôs de busca.

```
import { MetadataRoute } from "next";

export default function robots(): MetadataRoute.Robots {
  return {
    rules: [
      {
        userAgent: "*",
        allow: "/",
        disallow: ["/manager/admin"],
      },
    ],
    sitemap: `${process.env.NEXT_PUBLIC_BASE_URL}/sitemap.xml`,
  };
}
```

Figura 19 – Arquivo robots. Fonte: autor

```
import { MetadataRoute } from "next";

export default async function sitemap(): Promise<MetadataRoute.Sitemap> {
  return [
    {
      url: `${process.env.NEXT_PUBLIC_BASE_URL}/register`,
    },
    {
      url: `${process.env.NEXT_PUBLIC_BASE_URL}/login`,
    },
    {
      url: `${process.env.NEXT_PUBLIC_BASE_URL}/manager`,
    },
    {
      url: `${process.env.NEXT_PUBLIC_BASE_URL}/leagues`,
    },
    {
      url: `${process.env.NEXT_PUBLIC_BASE_URL}/ranking`,
    },
  ];
```

Figura 20 – Arquivo sitemap. Fonte: autor

5.2 React Hook Form

React Hook Form se destaca como uma biblioteca leve e eficiente para gerenciamento de formulários em React. Sua popularidade se deve à sua API intuitiva e flexível, que simplifica a criação e validação de formulários complexos. Com React Hook Form, desenvolvedores podem registrar campos de formulário, aplicar regras de validação personalizadas, gerenciar erros de forma eficiente e enviar dados para o servidor com facilidade.

Uma das principais vantagens do React Hook Form é sua abordagem de validação baseada em registro. Ao invés de gerenciar o estado de cada campo individualmente, o React Hook Form permite que registre os campos e aplique as regras de validação de forma declarativa. Isso reduz a quantidade de código boilerplate, seções de código que são repetidas, e torna o formulário mais fácil de manter. (FORM, 2024)

Além disso, o React Hook Form se integra com outras bibliotecas populares do

ecossistema React através de resolvers, como Yup e Zod. Essa integração permite que se crie formulários robustos e com validação avançada, garantindo a integridade dos dados e melhorando a experiência do usuário.

Outra característica importante dele é seu desempenho otimizado. Ao utilizar uma abordagem não controlada para gerenciar o estado do formulário, a biblioteca minimiza o número de renderizações desnecessárias, resultando em um formulário mais rápido e responsivo. Adicionalmente, React Hook Form oferece recursos nativos para a prevenção de ataques Cross-Site Request Forgery (CSRF), uma vulnerabilidade comum em aplicações web. Através da geração e validação de tokens CSRF, React Hook Form garante que as requisições enviadas pelo formulário sejam originadas de fontes confiáveis, protegendo a aplicação contra ataques maliciosos.

A figura 21 evidencia um exemplo de utilização dentro do projeto, nele é feito o formulário para o preenchimento do perfil. O hook useForm é inicializado com o zodResolver, que permite a integração com o Zod para a validação dos dados do formulário. O hook também fornece funções como register, handleSubmit e formState para controlar o estado do formulário, registrar os campos e lidar com o envio dos dados.

```tsx
export default function PerfilForm({
  name,
  email,
  password,
  team,
  avatar,
}: PerfilProps) {
  const [showPassword, setShowPassword] = useState(false);
  const {
    register,
    handleSubmit,
    control,
    formState: { errors },
  } = useForm<ProfileData>({
    resolver: zodResolver(profileSchema),
    defaultValues: {
      user: name,
      email: email,
      password: password,
      team: team,
      foto: "",
    },
  });

  const onSubmit = (data: ProfileData) => {
    const formDataToSend = new FormData();
    for (const [key, value] of Object.entries(data)) {
      formDataToSend.append(key, value as string | Blob);
    }
    console.log(formDataToSend);
    const fileInput = document.getElementById(
      "foto"
    ) as HTMLInputElement | null;
    if (fileInput?.files && fileInput.files.length > 0) {
      formDataToSend.append("foto", fileInput.files[0]);
    }
    for (const [key, value] of formDataToSend.entries()) {
      console.log(key, value);
    }
  };
  const handleFotoChange = (event: React.ChangeEvent<HTMLInputElement>) => {
    const files = event.target.files;
    if (files && files.length > 0) {
      const selectedFile = files[0];
      console.log("Selected File:", selectedFile);
    }
  };
  const { field } = useController({
    name: "foto",
    control,
    defaultValue: "",
  });
  return (
    <form onSubmit={handleSubmit(onSubmit)} className="flex flex-col gap-20">
      <div className="flex flex-col gap-4">
        <div className="w-full lg:w-1/2 flex flex-col gap-1">
          <label className="" htmlFor="user">
            Nome do Perfil
          </label>
          <Input type="text" id="user" {...register("user")} />
          {errors.user && <span>{errors.user.message}</span>}
        </div>
```

Figura 21 – Exemplo de formulário criado pelo react-hook-form. Fonte: autor

5.2.1 Funcionalidades do hook

As principais funcionalidades, são:

- **register**: A função register é utilizada para conectar cada campo do formulário, permitindo que a biblioteca gerencie o estado e a validação dos campos.
- **handleSubmit**: A função handleSubmit é chamada quando o usuário envia o formulário. Ela executa a função onSubmit, que lida com o envio dos dados para o servidor.
- **control**: Objeto que contém informações sobre o estado do formulário, como se ele está sendo enviado, se há erros de validação e quais são esses erros.
- **formState**: A função handleSubmit é chamada quando o usuário envia o formulário. Ela executa a função onSubmit, que lida com o envio dos dados para o servidor.
- **errors**: A propriedade errors do objeto formState é usada para acessar os erros de validação dos campos. Esses erros são exibidos ao usuário como feedback em tempo real.
- **resolvers**: atua como um adaptador entre a biblioteca e uma ferramenta de validação externa. Ele traduz as regras de validação do formulário para um formato compreensível pela ferramenta, permitindo a validação dos dados de forma integrada e eficiente.

5.3 Zod

Zod é uma biblioteca de validação de esquema TypeScript projetada para ser fácil de usar. Permite que o desenvolvedor defina esquemas para os dados e os valide antes de usá-los. Isso ajuda a garantir que os dados estejam sempre em um formato consistente e correto.

Zod oferece uma série de vantagens que otimizam o desenvolvimento em TypeScript. Ao garantir a consistência e correção dos dados através de esquemas de validação rigorosos, a biblioteca previne bugs e potenciais vulnerabilidades de segurança. Adicionalmente, a sintaxe clara e intuitiva dos esquemas Zod promove a legibilidade do código, facilitando a manutenção e colaboração em projetos. Essa combinação de segurança e clareza resulta em um aumento significativo na produtividade, permitindo que os desenvolvedores escrevam código mais rápido e com menos erros (ZOD, 2024).

5.3.1 Características

Suas principais características. são:

- **Inferência de tipo estática**: infere automaticamente os tipos TypeScript de seus esquemas, o que ajuda a evitar erros de tipo em tempo de compilação.
- **Validação de primitivos**: fornece validação para tipos primitivos como string, número, booleano, data e muito mais.
- **Validação de objetos**: permite que se valide a estrutura e os tipos de propriedades em objetos.
- **Validação de arrays**: permite que se valide o conteúdo e a estrutura de arrays.
- **Uniões e tipos discriminados**: suporta uniões de tipos e tipos discriminados, o que permite validar dados com diferentes estruturas.
- **Refinamentos personalizados**: permite que se crie suas próprias funções de validação para casos de uso específicos.
- **Validação assíncrona**: suporta validação assíncrona para cenários em que se precisa buscar dados de outras fontes.

```
1  const profileSchema = z.object({
2    user: z.string().min(3, "Nome do perfil deve ter no mínimo 3 caracteres"),
3    password: z
4      .string()
5      .min(8, "A senha deve ter no mínimo 8 caracteres")
6      .refine(
7        (value) => /[A-Z]/.test(value) && /[a-z]/.test(value) && /\d/.test(value),
8        "A senha deve conter pelo menos uma letra maiúscula, uma letra minúscula e um número"
9      ),
10   email: z.string().email("Email inválido"),
11   team: z.string().optional(),
12   foto: z.string().optional(),
13 });
14
15 export type ProfileData = z.infer<typeof profileSchema>;
```

Figura 22 – Exemplo de criação de um zod schema. Fonte: autor

5.4 Shadcn/ui

Shadcn/UI se destaca no cenário de desenvolvimento de interfaces de usuário em Next.js por sua flexibilidade e modularidade, utilizando o Radix UI como base para muitos de seus componentes. Um dos principais atrativos dessa biblioteca é a liberdade que ela oferece aos desenvolvedores na escolha dos componentes que desejam integrar em seus projetos. Diferentemente de outras bibliotecas que exigem a instalação de um pacote completo, o Shadcn/UI permite a instalação individual de cada componente, resultando em diversas vantagens (SHADCN/UI, 2024)

Essa abordagem modular reduz significativamente o tamanho do bundle da aplicação, pois apenas os componentes necessários são instalados, otimizando o tempo de carregamento e a performance geral. Além disso, a instalação individual permite a customização de cada componente de acordo com as necessidades específicas do projeto, sem a necessidade de modificar o código fonte da biblioteca.

A modularidade de Shadcn/UI também facilita a manutenção do código, pois as atualizações e correções de bugs podem ser aplicadas individualmente a cada componente, sem afetar o restante da aplicação. Essa flexibilidade na escolha dos componentes permite que os desenvolvedores utilizem apenas as funcionalidades que realmente precisam, evitando a inclusão de código desnecessário e tornando o Shadcn/UI uma opção atraente para projetos de todos os tamanhos e complexidades.

Um dos principais diferenciais de Shadcn/UI é sua abordagem de estilização baseada em Tailwind CSS. Essa poderosa ferramenta permite a personalização granular dos componentes, adaptando-os facilmente às necessidades específicas de cada projeto. A combinação do Shadcn/UI com o Tailwind CSS proporciona aos desenvolvedores um alto grau de controle sobre a aparência da interface, sem a necessidade de escrever CSS personalizado. Ao instalar cada componente, são criados dentro da pasta "components" na sub pasta ui criada pela própria biblioteca. A figura 23 mostra todos componentes utilizados no projeto.

Cada um dos aquivos .tsx dentro da pasta ui é um componente do próprio Shadcn/ui. Eles servem de base para a criação de componentes específicos como demonstra a figura 24 que cria um Dialog para mostrar uma tabela com cada um dos jogadores que podem ser escaldos no time do usuário.

Apesar da criação de componentes modificáveis pode ocorrer um problema na segurança pois a alteração destes se torna responsabilidade do desenvolvedor, por se tratar de códigos simples esse problema é altamente mitigado.

Além de sua versatilidade e facilidade de uso, o Shadcn/UI também se destaca pela sua integração perfeita com o ecossistema Next.js. A biblioteca oferece suporte completo para recursos como roteamento, otimização de imagens e renderização do lado do servidor, garantindo um desempenho otimizado e uma experiência de usuário fluida.

O Shadcn/UI, embora ofereça um componente <Table> básico, não inclui uma solução completa para DataTables complexas. Para construir DataTables com recursos avançados, como paginação, ordenação, filtragem e seleção de linhas, o Shadcn/UI recomenda a integração com a biblioteca TanStack Table, (*@tanstack/react-table*), apresentado em 5.4.1.

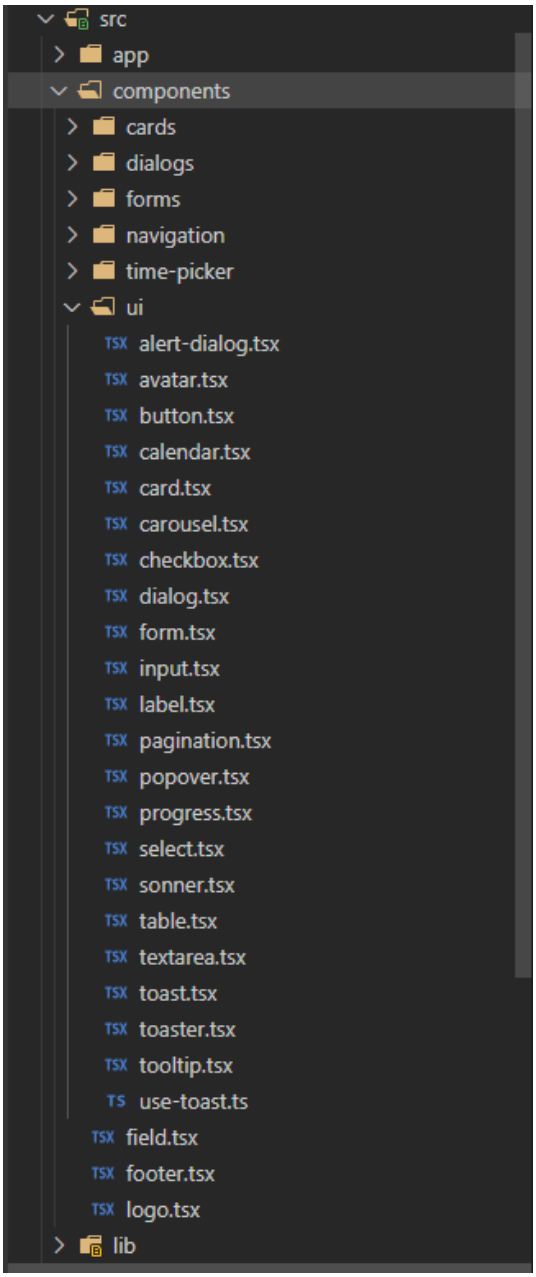

Figura 23 – Arquivos de componentes utilizados no projeto. Fonte: autor

```
export async function Dialogtable() {
  const players = await getTeamPlayers("1", "1");

  return (
    <Dialog>
     <DialogTrigger asChild>
      <Button
        className="border-blue-800 text-blue-800 rounded-lg border font-semibold p-3 w-[100px]"
          variant="outline"
        >
          Escalar
        </Button>
      </DialogTrigger>
      <DialogContent className="flex flex-col  justify-center items-center">
        <DialogHeader className="hidden">
          <DialogTitle>Table Dialog</DialogTitle>
          <DialogDescription>Table with all the players</DialogDescription>
        </DialogHeader>
        <div className="w-full">
          <DataTable columns={columns} data={players} />
        </div>
      </DialogContent>
    </Dialog>
  );
}

```

Figura 24 – Exemplo de um dialog customizado com base no Shadcn/ui. Fonte: autor

5.4.1 TanStack Table

TanStack Table é uma biblioteca "headless", o que significa que não impõe uma interface visual específica. Em vez disso, ele fornece a lógica e as ferramentas necessárias para construir DataTables altamente customizáveis. Essa abordagem "headless" permite que os desenvolvedores tenham total controle sobre a aparência e o comportamento da DataTable, adaptando-a perfeitamente ao design e às necessidades do projeto.

A flexibilidade se estende além da aparência. A biblioteca oferece uma ampla gama de recursos, como manipulação de dados, gerenciamento de estado, eventos e *hooks* , que permitem aos desenvolvedores criar *DataTables* com funcionalidades avançadas e interativas (TANSTACK, 2024).

5.5 Boas Práticas

5.5.1 Separação de Preocupações

A arquitetura do projeto, focada na separação de responsabilidades e na organização modular de diretórios, impulsiona a reutilização de código e a escalabilidade do sistema. Componentes como o DialogCreateLeague, encapsulados em suas respectivas pastas, tornam-se blocos de construção reutilizáveis em diferentes partes da aplicação. Essa abordagem reduz a redundância e acelera o desenvolvimento, garantindo consistência na interface e no comportamento.

A estrutura modular também simplifica a manutenção e a evolução do projeto. Novas funcionalidades podem ser adicionadas de forma isolada e controlada, sem afetar o funcionamento de outros módulos. A escalabilidade é beneficiada pela flexibilidade da arquitetura, permitindo que o sistema se adapte facilmente a crescentes demandas e complexidades.

5.5.2 Componentes Reutilizáveis

A implementação de componentes reutilizáveis, como GameCard, PlayerCard, TeamCard e CampaignCard, desempenha um papel crucial na otimização do desenvolvimento da interface do usuário. Ao compartilhar elementos visuais e funcionais em diferentes seções da aplicação, essa abordagem não apenas minimiza a redundância de código, mas também assegura uma experiência visual e interativa coesa.

O componente GameCard, por exemplo, ilustra essa versatilidade ao ser utilizado tanto na página inicial para apresentar os próximos jogos, quanto na área administrativa para gerenciar as partidas. Essa reutilização estratégica promove a manutenibilidade do código, facilitando a atualização e aprimoramento da interface em um único ponto, impactando positivamente toda a aplicação.

Além disso, a adoção de componentes reutilizáveis contribui para a escalabilidade do projeto, permitindo a incorporação de novas funcionalidades e seções sem comprometer a estrutura e a organização do código. Essa modularidade simplifica a colaboração entre desenvolvedores, agilizando o processo de desenvolvimento e reduzindo a probabilidade de erros.

5.5.3 Renderização Condicional

A renderização condicional, baseada em variáveis de *estado* estado ou *props* (como hasLeague e mercadoAberto), possibilita a exibição dinâmica de conteúdo e elementos da interface. Essa técnica é importante para adaptar a interface do usuário de acordo com o estado da aplicação ou as interações do usuário. Por exemplo, o componente LeaguePage, que exibe informações sobre uma liga existente se hasLeague for verdadeiro, ou um prompt para criar uma liga caso contrário.

5.5.4 Acessibilidade

Para garantir a acessibilidade da aplicação, foram implementadas medidas que visam a inclusão de pessoas com deficiências. Dentre essas medidas, destacam-se a utilização de atributos *alt* em imagens, que fornecem descrições textuais para leitores de tela, e atributos aria-label em elementos interativos, que melhoram a navegação para usuários com dificuldades visuais ou motoras.

Um exemplo concreto dessa prática pode ser observado no componente Field, onde o atributo alt=“Field” na imagem correspondente auxilia usuários com deficiência visual a compreender o contexto da imagem e sua relação com o restante da interface.

5.5.5 Responsividade

A utilização de classes do Tailwind CSS, como flex, flex-col, flex-row e os utilitários responsivos (exemplificado por lg:w-1/2), permitiu criar layouts fluidos e adaptáveis. Estes layouts se ajustam de forma inteligente e harmoniosa a uma ampla gama de tamanhos de tela, abrangendo desde monitores de desktop espaçosos até os compactos displays de smartphones e tabletes. Essa abordagem garante não apenas uma experiência de usuário consistente, mas também intuitiva em qualquer dispositivo utilizado para acessar a aplicação.

Figura 25 – Tela de escalação mobile. Fonte: autor

Na figura 25 foi criada a tela de escalção mobile. Nela foi retirado o campo e os elementos foram modificados para ficar mais claro em resolução menores. O componente de navegação (Nav) serve como um exemplo prático da implementação dessa estratégia. Através do uso dessas classes, o componente se transforma, apresentando um menu hambúrguer compacto e funcional em telas menores, otimizando o espaço limitado, enquanto em telas maiores se expande para uma barra de navegação completa, oferecendo acesso rápido e intuitivo a todas as funcionalidades.

5.5.6 Context e prop drilling

No desenvolvimento de aplicações React, a passagem de dados entre componentes é um aspecto fundamental. Duas abordagens comuns para lidar com essa tarefa são o prop drilling e a Context API. O prop drilling, embora simples em sua implementação, pode se tornar problemático em estruturas de componentes complexas, onde os dados precisam ser passados por múltiplos níveis. A Context API, por outro lado, oferece uma solução mais legível e melhor para compartilhar dados globalmente, evitando a necessidade de passar props explicitamente por cada componente intermediário. (REACT, 2024a)

Características	**Prop Drilling**	**Context**
Passagem de dados	Através de props em cada nível	Através de um provedor de contexto
Manutenibilidade	Mais difícil de manter, especialmente em estruturas complexas	Mais fácil de manter, pois a mudança é feita apenas no provedor
Legibilidade	Código menos legível devido à repetição de props	Código mais legível e organizado
Flexibilidade	Dados acessíveis apenas pelos componentes que recebem as props	Dados acessíveis por qualquer componente abaixo do provedor

Tabela 3 – Tabela Comparativa entre Prop Drilling e Context API.

No contexto do projeto InterREP Manager, ambas as abordagens foram empregadas de forma estratégica. O *prop drilling*, figura 26, foi utilizado em cenários mais simples, como na página de perfil, onde a passagem de dados ocorre apenas entre um componente pai e seu filho imediato. Essa escolha se justifica pela simplicidade e baixo *overhead* do *prop drilling* em casos como esse.

Entretanto, em situações mais complexas, como na edição de times dentro de um modal, a Context API se mostrou mais adequada. A criação de um contexto específico, como o TeamContext da figura 27, permitiu que os dados da página fossem compartilhados diretamente com o formulário de edição, eliminando a necessidade de passar props por cada componente intermediário, figuras 28 e 29. Essa abordagem resultou em um código

```tsx
export default async function PerfilPage() {
  const perfil = await getLoggedUserinformation("1");
  const perfilFormData = {
    name: perfil?.name,
    email: perfil?.email,
    password: perfil?.password,
    team: perfil?.team,
    avatar: perfil?.avatar,
  };
  return (
    <div className="min-h-screen flex flex-col gap-8">
      <p className="text-2xl font-bold self-start">Configurações</p>

      <PerfilForm {...perfilFormData}></PerfilForm>
    </div>
  );
}

```

Figura 26 – Exemplo de prop drilling. Fonte: autor

mais limpo, organizado e fácil de manter, especialmente considerando a estrutura aninhada de componentes dentro do modal.

```tsx
"use client";
import { Team } from "@/types/teams";
import { createContext, useContext, ReactNode } from "react";

interface TeamContextType {
  teamData: Team | null;
}

const TeamContext = createContext<TeamContextType | undefined>(undefined);

export const useTeamContext = () => {
  const context = useContext(TeamContext);
  if (!context) {
    throw new Error("useTeamContext must be used within a TeamProvider");
  }
  return context;
};

interface TeamProviderProps {
  teamData: Team | null;
  children: ReactNode;
}

export const TeamProvider: React.FC<TeamProviderProps> = ({
  teamData,
  children,
}) => {
  return (
    <TeamContext.Provider value={{ teamData }}>{children}</TeamContext.Provider>
  );
};

```

Figura 27 – TeamContext. Fonte: autor

```tsx
export const metadata: Metadata = {
  title: "Team",
};

export default async function AdminTeamPage({ params }: TeamProps) {
  const team = await getTeam(params.teamId);
  let players: Player[] = [];
  if (team) {
    players = await getPlayers(team.id, team.tournamentId);
  }
  const games = await getGames();
  return (
    <div className="flex flex-col gap-6 items-center justify-center">
      <div
      className="bg-white flex flex-col lg:flex-row
      gap-3 self-start w-full justify-between px-4 py-2 items-center h-fit lg:h-20">
        <div className="w-full flex gap-4 items-center">
          <Image
            src={team?.avatar || ""}
            width={60}
            height={60}
            alt="foto do jogador"
            className="rounded-full"
          />
          <span className="text-lg lg:text-xl font-semibold w-full">
            {team?.name}
          </span>
        </div>
        <Suspense fallback={<p>lendo...</p>}>
          <TeamProvider teamData={team}>
            <DialogEditTeam></DialogEditTeam>
          </TeamProvider>
        </Suspense>

```

Figura 28 – Criação do TeamContext na página de administrador. Fonte: autor

A decisão de utilizar o prop drilling ou a Context API depende da complexidade da estrutura de componentes e da necessidade de compartilhar dados globalmente. No projeto InterREP Manager, a combinação estratégica dessas duas abordagens garantiu a eficiência na passagem de dados, a manutenibilidade do código e a otimização do desempenho da aplicação. A Context API, em particular, se mostrou valiosa para lidar com a complexidade da edição de times dentro de um modal, demonstrando sua capacidade de simplificar o compartilhamento de dados em estruturas de componentes aninhadas.

```
type editTeamData = z.infer<typeof editTeamSchema>;

export default function EditTeamForm() {
  const { teamData } = useTeamContext();
  const {
    register,
    handleSubmit,
    formState: { errors },
  } = useForm<editTeamData>({
    resolver: zodResolver(editTeamSchema),
    defaultValues: {
      teamName: teamData?.name || "",
      teamAvatar: teamData?.avatar || "",
    },
  });

  const onSubmit = (data: editTeamData) => {
    console.log(data);
  };

  return (
    <form onSubmit={handleSubmit(onSubmit)} className="space-y-6">
      <div className="flex flex-col items-center justify-center gap-4">
        <Image
          src={teamData?.avatar || "/demoro.png"}
          width={250}
          height={250}
          alt="foto do jogador"
          className="rounded-2xl"
        />
        <div className="w-full flex flex-col gap-1">
          <label htmlFor="teamName" className="self-start">
            Nome da Time:

```

Figura 29 – Utilização do contexto na edição do time. Fonte: autor

5.5.7 Cookies

Nesta aplicação, foi decidido utilizar os cookies da biblioteca next/headers. Usá-los oferece maior segurança em comparação ao localStorage, principalmente devido à sua natureza HTTP-only. Isso significa que os cookies são acessíveis apenas pelo servidor, protegendo-os de potenciais ataques de Cross-Site Scripting (XSS), onde scripts maliciosos podem tentar roubar dados sensíveis do localStorage no lado do cliente. Além disso, os cookies permitem um controle mais granular sobre a expiração e o escopo, garantindo que os dados sejam armazenados apenas pelo tempo necessário e acessíveis apenas às partes relevantes da aplicação. A figura 30 demonstra a criação dos cookies para autorização.

Com o objetivo de reforçar a segurança da aplicação, são implementadas flags específicas na criação dos cookies. A flag httpOnly garante que os cookies sejam acessíveis exclusivamente pelo servidor, blindando-os contra potenciais ataques XSS que poderiam tentar capturar informações sensíveis no lado do cliente. Adicionalmente, a flag secure assegura que a transmissão dos cookies ocorra apenas em conexões HTTPS, adicionando uma camada extra de proteção. Por fim, a flag sameSite é empregada para restringir o envio de cookies em requisições de terceiros, prevenindo ataques CSRF que exploram a

confiança estabelecida entre o navegador e o servidor. No capítulo 5.1.3.6 é possível ver que ao realizar o logout os cookies setados são deletados.

```
    if (response.ok) {
      const { token, isAdmin } = await response.json();

      const cookieStore = cookies();
      cookieStore.set("authToken", token,{ httpOnly: true, secure: true, sameSite:true });
      cookieStore.set("isAdmin", isAdmin, { httpOnly: true, secure: true, sameSite:true  });

      return NextResponse.json({ success: true, isAdmin });
    } else {
      const errorData = await response.json();
      return NextResponse.json({ error: errorData.error || "Unknown error" }, { status: 400 });
    }
  } catch (error) {
    return NextResponse.json({ error: "Server error" }, { status: 500 });
```

Figura 30 – Cookie de autorização. Fonte: autor

5.5.8 Middleware

O middleware no Next.js 14 é uma ferramenta essencial para proteger rotas e controlar o fluxo de requisições em seu aplicativo. Ele atua como um intermediário entre as requisições do usuário e as respostas do servidor, permitindo executar código antes que uma requisição seja concluída. Isso possibilita diversas ações, como redirecionamentos, modificação de cabeçalhos e até mesmo respostas diretas.

O código da figura 31 demonstra como o middleware pode ser utilizado para proteger rotas administrativas. Ele verifica se o usuário possui um cookie de autenticação válido e, caso contrário, redireciona-o para a página de login. Além disso, o middleware verifica se o usuário tem permissão de administrador para acessar determinadas rotas, restringindo o acesso a áreas sensíveis apenas para usuários autorizados. Como esse middleware foi implementado ao mesmo nível da pasta app, todas as rotas dentro do projeto são protegidas por ele.

Um middleware permite implementar autenticação e autorização, controlando o acesso a áreas restritas. Também possibilita redirecionamentos condicionais com base em informações da requisição, como o agente de usuário ou cookies. O middleware também pode ser usado para manipular cabeçalhos de requisição ou resposta, permitindo personalizar o comportamento do servidor. Além disso, ele pode ser utilizado para implementar tratamento de erros centralizado e registrar informações sobre as requisições, auxiliando na análise e depuração do aplicativo.

```
1  import { NextResponse, NextRequest } from "next/server";
2  import { cookies } from "next/headers";
3
4  export function authMiddleware(request: NextRequest): NextResponse {
5    console.log("Middleware triggered");
6    try {
7      const cookieStore = cookies();
8      const tokenCookie = cookieStore.get("authToken");
9
10     if (!tokenCookie) {
11       return NextResponse.redirect(new URL("/login", request.url));
12     }
13
14     const isAdminCookie = cookieStore.get("isAdmin");
15     console.log("isAdmin cookie:", isAdminCookie?.value);
16
17     if (request.nextUrl.pathname === "/manager/admin") {
18       if (!isAdminCookie || isAdminCookie.value !== "true") {
19         console.log(
20           "Redirecting to /manager due to missing/invalid isAdmin cookie"
21         );
22         return NextResponse.redirect(new URL("/manager", request.url));
23       } else {
24         return NextResponse.next();
25       }
26     }
27
28     if (request.nextUrl.pathname.startsWith("/manager/")) {
29       return NextResponse.next();
30     }
31
32     return NextResponse.next();
33   } catch (error) {
34     console.error("Error in middleware:", error);
35     return NextResponse.next();
36   }
37 }
38 export const config = {
39   matcher: ["/manager/:path*"],
40 };
41
```

Figura 31 – Middleware. Fonte: autor

6 Segurança

A segurança em aplicações web é um desafio constante, com ameaças em constante evolução que buscam explorar vulnerabilidades e comprometer a integridade dos sistemas. O InterREP Manager, como qualquer plataforma online, está exposto a esses riscos. Para garantir a proteção dos dados dos usuários e a confiabilidade do sistema, foram implementadas diversas medidas de segurança que abordam os principais vetores de ataque.

6.1 DDoS (Distributed Denial of Service)

Um ataque DDoS é uma tentativa maliciosa de tornar um servidor, serviço ou rede indisponível para usuários legítimos, inundando-o com um volume massivo de tráfego proveniente de várias fontes (computadores, dispositivos IoT comprometidos). O objetivo é sobrecarregar os recursos do alvo, como largura de banda, CPU ou memória, até que ele fique lento, inoperante ou completamente inacessível. Existem diferentes tipos de ataques DDoS, como ataques volumétricos (inundam o alvo com tráfego), ataques de protocolo (exploram vulnerabilidades em protocolos de rede) e ataques de aplicação (direcionados a aplicações específicas).

Para mitigar esse risco, o uso de cache, como implementado no projeto, armazena cópias de páginas e recursos estáticos, reduzindo a carga no servidor durante um ataque na camada 7 e mantendo a plataforma acessível, mesmo sob intenso tráfego malicioso. Durante um ataque o cache pode servir essas cópias diretamente aos usuários, reduzindo a carga no servidor e mantendo a plataforma acessível.

6.2 XSS(Cross-Site Scripting)

Reflected XSS ocorre quando um aplicativo web inclui dados não confiáveis (como parâmetros de consulta na URL ou dados de formulário) na resposta HTTP sem sanitização ou codificação adequada. Um invasor pode criar um link malicioso contendo código JavaScript e induzir a vítima a clicar nele. O servidor, sem validação adequada, processa a entrada maliciosa e a reflete de volta para o navegador da vítima. O navegador, acreditando que o script veio do servidor confiável, o executa, permitindo que o invasor roube dados sensíveis (como cookies de sessão), realize ações em nome do usuário ou até mesmo redirecione a vítima para outro site malicioso. (OWASP, 2024a)

Stored XSS ocorre quando dados fornecidos por um usuário são armazenados no servidor (banco de dados, sistema de arquivos) sem sanitização adequada e posteriormente

exibidos a outros usuários sem codificação de saída. Um invasor pode injetar código JavaScript malicioso em campos de entrada, como comentários em um fórum ou mensagens em um chat. Quando outros usuários visualizam o conteúdo contaminado, o script é executado em seus navegadores, permitindo que o invasor roube dados, realize ações em nome das vítimas ou comprometa seus sistemas. A sanitização rigorosa de todas as entradas do usuário antes de serem enviadas ao servidor, utilizando codificação contextual. (OWASP, 2024a)

A Codificação de Saída Contextual é uma técnica essencial para proteger aplicações web contra ataques XSS (Cross-Site Scripting), que se aproveitam da exibição de dados não confiáveis em páginas web. Em vez de aplicar uma codificação genérica, essa técnica adapta a codificação dos dados ao contexto específico em que serão inseridos, garantindo uma proteção mais eficaz.

Por exemplo, ao inserir dados diretamente no corpo do HTML, caracteres como "<", ">" e "&" são convertidos em suas respectivas entidades HTML, impedindo que o navegador os interprete como tags e execute código malicioso. Se os dados forem inseridos em atributos HTML, a codificação é diferente, focando em caracteres como aspas para evitar que o invasor feche o atributo e insira código adicional. (OWASP, 2024b)

Ela também se aplica a outros contextos, como blocos de código JavaScript, URLs e folhas de estilo CSS. Em cada caso, a codificação é adaptada para garantir que os dados sejam tratados corretamente e não permitam a execução de código malicioso.

Na figura 32, a segurança dos dados é priorizada na validação do nome da equipe, utilizando uma combinação robusta de tecnologias. O esquema Zod garante que apenas dados válidos e seguros sejam aceitos, aplicando regras como tipo de dado, comprimento e caracteres permitidos.

Simultaneamente, o DOMPurify remove potenciais scripts maliciosos, protegendo contra ataques XSS e garantindo a integridade dos dados. Essa abordagem é integrada ao React Hook Form, proporcionando feedback instantâneo ao usuário sobre erros de validação, o que melhora a experiência e guia o preenchimento correto do formulário. Por fim, o ambiente estruturado do Next.js complementa essa estratégia, resultando em uma aplicação web segura, confiável e moderna.

A fim de proteger ainda mais contra esses tipos de ataques foi criado um middleware como exemplificado anteriormente mas aplicando uma solução para uma política de segurança de conteúdo (CSP). O código da figura 33 utiliza um nonce, uma string aleatória e única, gerada para ser usada uma única vez.

Essa política define regras como permitir carregar recursos apenas da mesma origem "self" e permitir scripts e estilos inline somente se eles contiverem o nonce correto. As flags "strict-dynamic" e "unsafe-inline" modificam o comportamento padrão da CSP, permitindo

```
const editTeamSchema = z.object({
   teamAvatar: z.string().url().optional(),
   teamName: z
     .string()
     .trim()
     .min(1, "O nome da equipe é obrigatório")
     .max(30, "O nome da equipe deve ter no máximo 30 caracteres")
     .regex(/^[a-zA-Z0-9_-]+$/,
     "O nome da equipe pode conter apenas letras, números, hífens e underlines"
     )
     .transform((value) => {
      try {
        const sanitizedValue = DOMPurify.sanitize(value);
        if (sanitizedValue !== value) {
          throw new Error("O nome da equipe contém caracteres inválidos.");
        }
        return sanitizedValue;
      } catch (error) {
        throw new Error(
        "Ocorreu um erro ao validar o nome da equipe. Por favor, tente novamente."
        );
      }
    }),
 });
```

Figura 32 – Sanitização com o DOMPurify. Fonte: autor

o carregamento dinâmico de scripts e a execução de scripts inline legados, respectivamente, embora com riscos de segurança associados.

Além de configurar o cabeçalho CSP, o middleware também adiciona o nonce como um cabeçalho personalizado na resposta, permitindo que o utilize no lado do cliente para marcar scripts e estilos inline legítimos. Por fim, o middleware é configurado garantindo que a proteção da CSP seja aplicada onde é mais crítica.

6.3 CSRF (Cross-Site Request Forgery)

Um ataque CSRF explora a confiança que um aplicativo web tem no navegador de um usuário autenticado. O invasor cria uma página maliciosa ou envia um e-mail com um link/imagem oculta que, quando clicado/carregado pela vítima, força o navegador a enviar uma solicitação forjada para o aplicativo web alvo. Essa solicitação maliciosa pode realizar ações indesejadas, como alterar a senha da vítima, fazer compras ou transferir fundos, sem o conhecimento ou consentimento do usuário. O ataque se baseia no fato de que o navegador da vítima automaticamente inclui cookies de autenticação em todas as solicitações para o aplicativo, mesmo que a solicitação se origine de um site malicioso.

O uso de tokens anti-CSRF, gerados pelo servidor e incluídos em formulários e solicitações, ajuda a prevenir ataques CSRF. O servidor verifica se o token recebido corresponde ao token armazenado na sessão do usuário, garantindo que a solicitação seja legítima. Os tokens devem ser únicos por sessão e por usuário.

```
async function generateNonce(): Promise<string> {
  const array = new Uint8Array(16);
  crypto.getRandomValues(array);
  return btoa(String.fromCharCode(...array));
}

export async function cspMiddleware(request: NextRequest) {
  const nonce = await generateNonce();
  const cspHeader = `
    default-src 'self';
    script-src 'self' 'nonce-${nonce}' 'strict-dynamic';
    style-src 'self' 'nonce-${nonce}';
    img-src 'self' blob: data:;

    font-src 'self';
    object-src 'none';
    base-uri 'self';
    form-action 'self';
    frame-ancestors 'none';

    upgrade-insecure-requests;
  `;
  const contentSecurityPolicyHeaderValue = cspHeader
    .replace(/\s{2,}/g, " ")
    .trim();

  const requestHeaders = new Headers(request.headers);
  requestHeaders.set("x-nonce", nonce);
  requestHeaders.set(
    "Content-Security-Policy",

    contentSecurityPolicyHeaderValue
  );

  const response = NextResponse.next({
    request: {
      headers: requestHeaders,
    },
  });
  response.headers.set(
    "Content-Security-Policy",
    contentSecurityPolicyHeaderValue
  );

  return response;
}
```

Figura 33 – Política de csp. Fonte: autor

A configuração SameSite em cookies ajuda a restringir o envio de cookies em solicitações de origens diferentes, dificultando ataques CSRF.

A política criada na figura 33 também protege contra esse tipo de ataques ao definir as diretivas:

- **form-action 'self':** restringe o envio de formulários apenas para a mesma origem (o próprio site), impedindo que formulários sejam enviados para sites maliciosos em um ataque CSRF.
- **frame-ancestors 'none':** impede que o site seja incorporado em um iframe de outro site. Isso dificulta ataques CSRF que tentam enganar o usuário a realizar ações em um iframe oculto.

6.4 Medidas de Segurança em Níveis de Back-End e Infraestrutura

A segurança de uma aplicação vai muito além do front-end. O back-end, que inclui a API e o banco de dados, também exige medidas de proteção robustas. No back-end, sanitizar e validar os dados antes de armazená-los ou exibi-los aos usuários garante que apenas informações seguras sejam processadas, garantindo que apenas dados seguros sejam processados. Bibliotecas como o class-transformer do NestJS podem auxiliar nesse processo. Adicionalmente, a implementação de serviços de proteção DDoS em nuvem, como Cloudflare ou AWS Shield, é crucial para filtrar o tráfego malicioso e garantir a disponibilidade da aplicação.

Ameaças como a injeção de SQL, que permite a execução de comandos maliciosos no banco de dados, e o acesso não autorizado à API, que pode levar à manipulação ou roubo de dados, devem ser abordadas e prevenidas com soluções como o uso de frameworks de segurança, validação e sanitização de entradas, controle de acesso baseado em funções e criptografia de dados sensíveis.

Outras medidas de segurança importantes incluem a proteção contra ataques de força bruta, que tentam adivinhar senhas, através da limitação de tentativas de login. O uso de HTTPS para garantir a criptografia dos dados em trânsito também é fundamental para proteger as informações contra interceptação e espionagem.

É crucial lembrar que um ataque bem-sucedido pode ter consequências graves, como o vazamento de dados dos estudantes universitários de São Carlos. Portanto, é essencial integrar práticas de segurança em todas as etapas do desenvolvimento, garantindo a proteção dos dados e a confiança dos usuários na aplicação.

7 Conclusões finais

O desenvolvimento da plataforma de Fantasy Game InterREP Manager apresentou resultados positivos ao atingir os objetivos principais: proporcionar uma experiência de usuário aprimorada e garantir a segurança dos dados. A escolha de tecnologias modernas, como o Next.js 14 e o pnpm, demonstrou-se acertada, permitindo uma aplicação eficiente tanto em performance quanto em organização do código.

A implementação seguiu padrões de usabilidade estabelecidos, fundamentados nas Heurísticas de Nielsen, assegurando que o usuário final pudesse interagir com a plataforma de forma intuitiva e eficiente. Além disso, as medidas de segurança adotadas, como o uso de cookies HTTP-only e a configuração de middlewares para proteção de rotas, fortaleceram a proteção contra vulnerabilidades comuns, como XSS, CSRF e DDoS.

Este trabalho evidenciou a importância de aliar uma interface amigável a uma robusta estrutura de segurança, especialmente em plataformas que lidam com dados sensíveis. A aplicação do conhecimento adquirido durante o curso de Engenharia de Computação foi fundamental para o sucesso do projeto, possibilitando a entrega de um produto que não só atende às necessidades dos usuários, mas também segue as melhores práticas do desenvolvimento seguro.

O aplicativo web desenvolvido tem o potencial de aprimorar a experiência dos participantes do InterREP, fortalecendo o torneio e promovendo a integração entre as repúblicas estudantis. Espera-se que este trabalho sirva como referência para futuros projetos na área de desenvolvimento de fantasy games e aplicações web em geral, incentivando a busca por soluções inovadoras e centradas no usuário.

7.1 Trabalhos Futuros

Embora este trabalho tenha se concentrado no desenvolvimento do front-end da aplicação, há várias áreas de expansão e aprimoramento que podem ser exploradas em estudos futuros para garantir que o sistema atinja todo o seu potencial.

1. **Desenvolvimento do Back-end:** Um dos principais desafios para o futuro é a implementação de um back-end robusto. Esta camada será responsável por gerenciar os dados dos usuários, lidar com a lógica de negócios e garantir a comunicação eficaz com o banco de dados. Com um back-end funcional, os usuários poderão realizar ações como cadastro, login, gerenciamento de times e participar de ligas, o que tornará a aplicação não apenas interativa, mas também altamente dinâmica.

2. **Integração com Serviços de Nuvem:** Implementar a aplicação em uma infraestrutura de nuvem, como AWS, Azure ou Google Cloud, é uma etapa crucial. Esta abordagem não só garantirá a disponibilidade e acessibilidade do sistema a qualquer hora e de qualquer lugar, mas também facilitará a escalabilidade e a manutenção contínua da plataforma. A adoção de serviços de nuvem permitirá atualizações e a introdução de novas funcionalidades de maneira ágil e eficiente, atendendo às demandas crescentes dos usuários.

3. **Expansão de Funcionalidades:** O sistema pode ser aprimorado com a introdução de novas funcionalidades que melhorem a experiência do usuário e ampliem as possibilidades de interação. Exemplos incluem a criação das páginas de jogadores mais escalados dos craques da rodada, análises estatísticas gerais do torneio, e a integração com redes sociais para aumentar o engajamento e a divulgação.

4. **Testes de Usabilidade e Segurança:** Continuar a realizar testes rigorosos de usabilidade e segurança será essencial para garantir que o sistema permaneça eficiente e seguro à medida que cresce. Isso inclui não só a revisão contínua das heurísticas de usabilidade aplicadas, mas também a implementação de novas medidas de segurança para proteger os dados dos usuários contra possíveis ameaças. Como testes End-to-End e unitários

5. **Análise de Impacto e Feedback dos Usuários:** Por fim, é recomendável que seja criado mecanismos para coletar feedback dos usuários. Isso ajudará a adaptar a aplicação às necessidades e preferências dos usuários finais, garantindo que ela permaneça relevante e eficaz.

Essas direções futuras não apenas complementarão o trabalho realizado até agora, mas também contribuirão para o desenvolvimento de uma plataforma completa, robusta e voltada para o usuário, consolidando a aplicação como uma ferramenta indispensável para o torneio InterREP e possivelmente para outras competições similares no futuro.

Referências

ANGULAR. *Angular*. 2024. <https://v17.angular.io/docs>. Angular Official Documentation. Citado na página 20.

BUN. *Bun*. 2024. <https://bun.sh/docs>. Bun docs. Citado na página 26.

CASAGRANDE, M. C.; LAVARDA, S. D. L.; SILVEIRA, A. C. M. Fantasy game e reconfiguração da cultura desportiva. *Intexto*, n. 52, p. 95133, maio 14 2021. Citado na página 12.

EXAME. *Febre dos fantasy games impulsiona crescimento do Arena 22*. 2022. <https://exame.com/bussola/febre-dos-fantasy-games-impulsiona-crescimento-do-arena-22/>. Acesso em: 22 abril. 2024. Citado na página 12.

FORM, R. H. *React Hook Form Docs*. 2024. <https://react-hook-form.com/get-started>. React Hook Form Documentation. Citado na página 48.

GEOCO, T. *UI Design Tools Survey 2023*. 2023. <https://uxtools.co/survey/2023/ui-design>. UI Design Tools Survey by UX Tools. Citado na página 16.

ISLAND. 2024. <https://www.patterns.dev/vanilla/islands-architecture/>. Island architeture. Citado na página 23.

KROTOFF, T. *Stack Overflow trends*. 2023. <https://gist.github.com/tkrotoff/b1caa4c3a185629299ec234d2314e190>. Gist no GitHub comparando diferentes frameworks javascript. Citado na página 19.

MOCKAPI. 2024. <https://mockapi.io/>. Plataforma para criar api mock. Citado na página 14.

NIELSEN, J. Heuristic evaluation. In: NIELSEN, J.; MACK, R. L. (Ed.). *Usability Inspection Methods*. New York: John Wiley & Sons, 1994. p. 25–62. Citado na página 16.

NORDSTRöM, C.; DIXELIUS, A. *Comparisons of Server-side Rendering and Client-side Rendering for Web Pages*. Dissertação (Mestrado) — Uppsala University, Department of Information Technology, 2023. Citado 2 vezes nas páginas 21 e 22.

OWASP. *Types of Cross-Site Scripting (XSS)*. 2024. <https://owasp.org/www-community/Types_of_Cross-Site_Scripting>. Article about different types of Cross-Site Scripting (XSS) vulnerabilities. Citado 2 vezes nas páginas 64 e 65.

OWASP. *Types of Cross-Site Scripting (XSS)*. 2024. <https://owasp.org/www-project-web-security-testing-guide/v41/4-Web_Application_Security_Testing/07-Input_Validation_Testing/01-Testing_for_Reflected_Cross_Site_Scripting.html>. Article about Testing for Reflected Cross Site Scripting. Citado na página 65.

PNPM. *Pnpm*. 2024. <https://pnpm.io/motivation>. Why pnpm? - Motivation. Citado na página 25.

PNPM. *Pnpm Benchmarks*. 2024. <https://pnpm.io/benchmarks>. Benchmarks of JavaScript Package Managers. Citado na página 26.

REACT. *Context Api*. 2024. <https://react.dev/learn/passing-data-deeply-with-context>. React Context Api Reference. Citado na página 58.

REACT. *Server Components*. 2024. <https://react.dev/reference/rsc/server-components>. React Server Components Reference. Citado na página 23.

SHADCN/UI. *shadcn/ui*. 2024. <https://ui.shadcn.com/docs>. Beautifully designed components built with Radix UI and Tailwind CSS. Citado na página 52.

TANSTACK. *TanStack Table*. 2024. <https://tanstack.com/table/latest/docs/introduction>. Headless UI for building powerful tables & datagrids. Citado na página 55.

VERCEL. *Understanding React Server Components*. 2023. <https://vercel.com/blog/understanding-react-server-components>. Artigo do blog da Vercel explicando React Server Components. Citado na página 23.

VERCEL. *Next*. 2024. <https://nextjs.org/docs>. Next.js Official Documentation. Citado na página 42.

ZOD. *Zod Docs*. 2024. <https://zod.dev/>. Zod Documentation. Citado na página 51.