UNIVERSIDADE DE SALVADOR

EDUARDO GONZAGA LIMA RA: 1272220388
VITOR PIO VIEIRA RA: 1272220376
MARCOS RIBEIRO RA: 12723116626
HERICLES SANDER DOS SANTOS CONCEICAO RA: 1272226965

RELATÓRIO DE DESENVOLVIMENTO DE PROJETO: APLICAÇÃO WEB PARA GERENCIAR CATÁLOGO DE JOGOS DE UM USUÁRIO

SALVADOR, BA

2023

SUMÁRIO

1. VISÃO GERAL

A aplicação WEB para gerenciar catálogo de jogos é uma solução para o problema de manter um registro de jogos eletrônicos. A aplicação é fácil de usar e permite que os usuários registrem seus jogos, adicione notas e categorias, e visualizem seus jogos.

2. OBJETIVOS

Mostrar como a aplicação WEB para gerenciar catálogo de jogos pode ajudar as pessoas a manter um registro de seus jogos.

- Descrever as principais funcionalidades da aplicação.
- Destacar a usabilidade da aplicação.

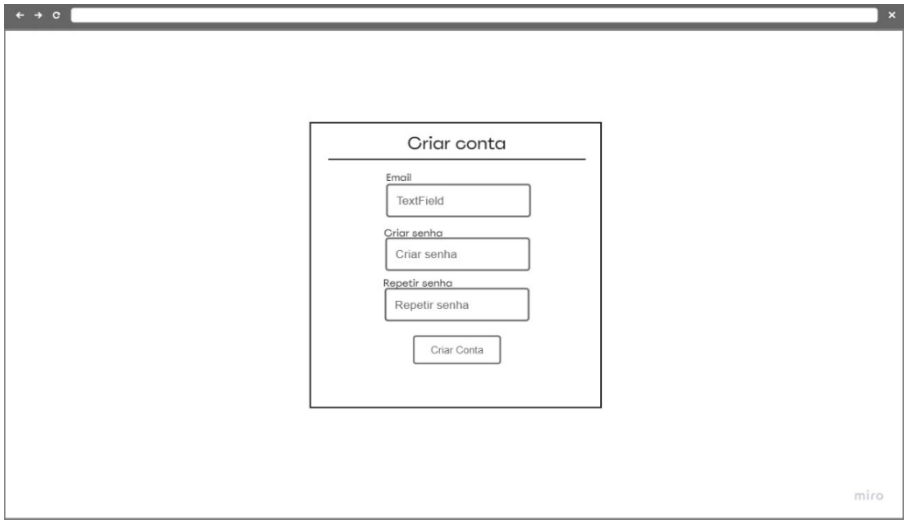
3. REQUISITOS DO PROJETO

Antes de iniciar o desenvolvimento, foram definidos os seguintes requisitos:

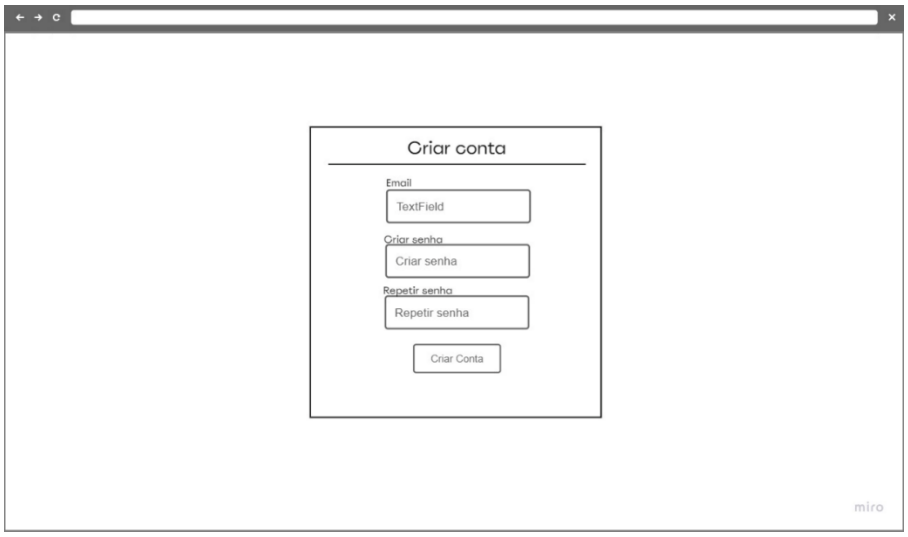
- Páginas Web: Desenvolver a página web com as seguintes especificações.
    - Página de login para acessar a aplicação WEB.
    - Página de cadastro na aplicação WEB.
    - Página principal de exibição da aplicação WEB.
    - Página da biblioteca onde mostra quais jogos ele possui.
    - Página de jogos para adicionar na aplicação WEB.
    - Página de acesso a plataforma principal onde estão os jogos.
    - Página de suporte ao usuário
- Design Responsivo: Garantir que o layout da aplicação seja responsivo para se adaptar a diferentes tamanhos de tela.
- Estilo e Layout: Utilizar CSS para estilizar as páginas de acordo com o design especificado.
- Funcionalidade Javascript: Implementar as seguintes funcionalidades usando Javascript:
    - Lista de funcionalidades, como validação de formulários, animações, etc.

4. WIREFRAMES

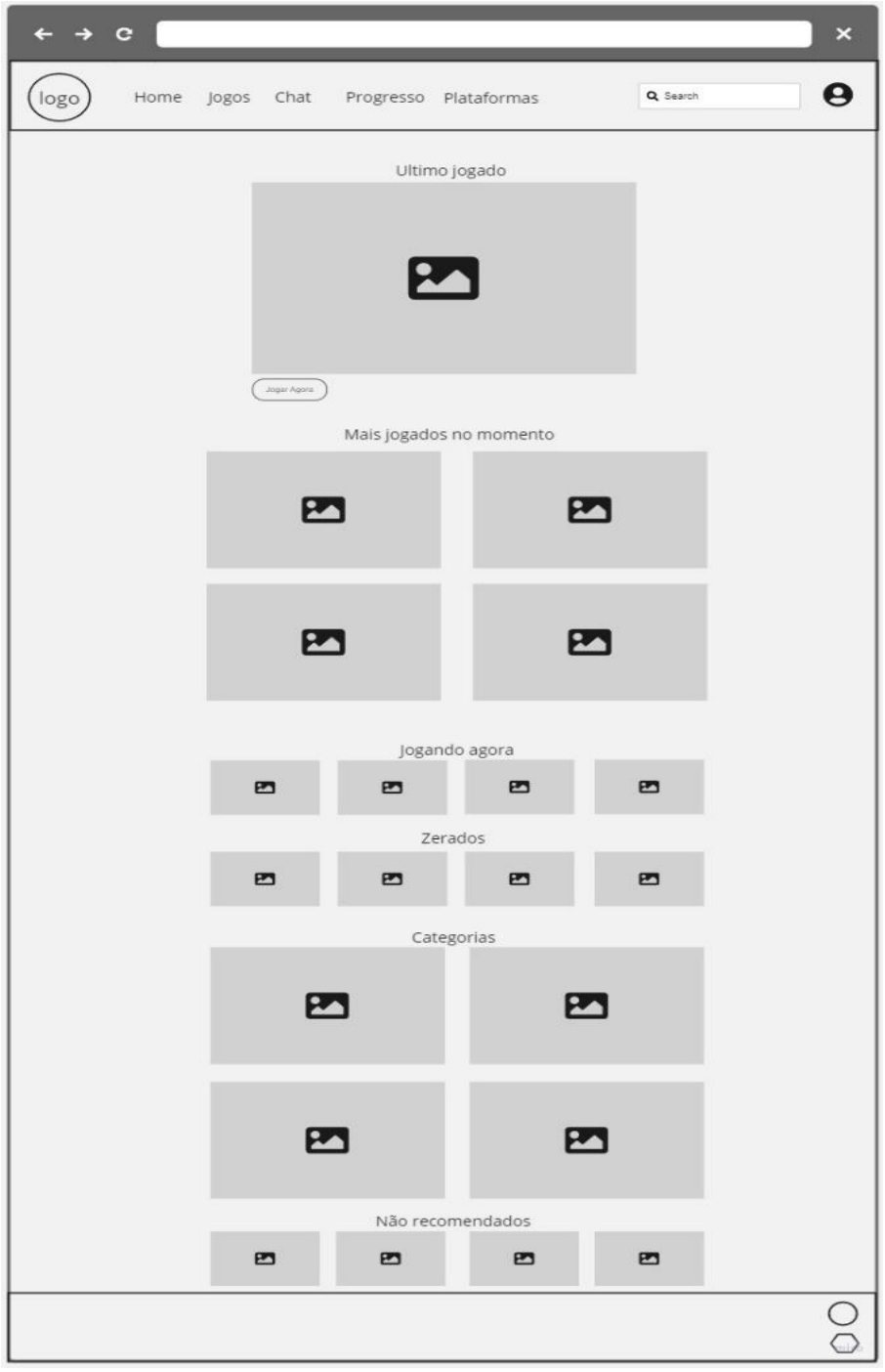
4.1 TELA DE LOGIN



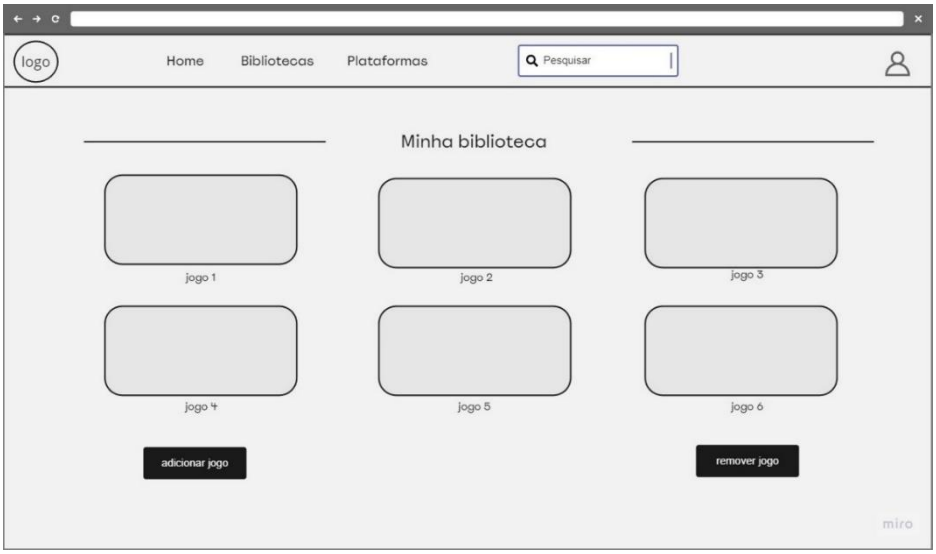
4.2 TELA DE CRIAR CONTA



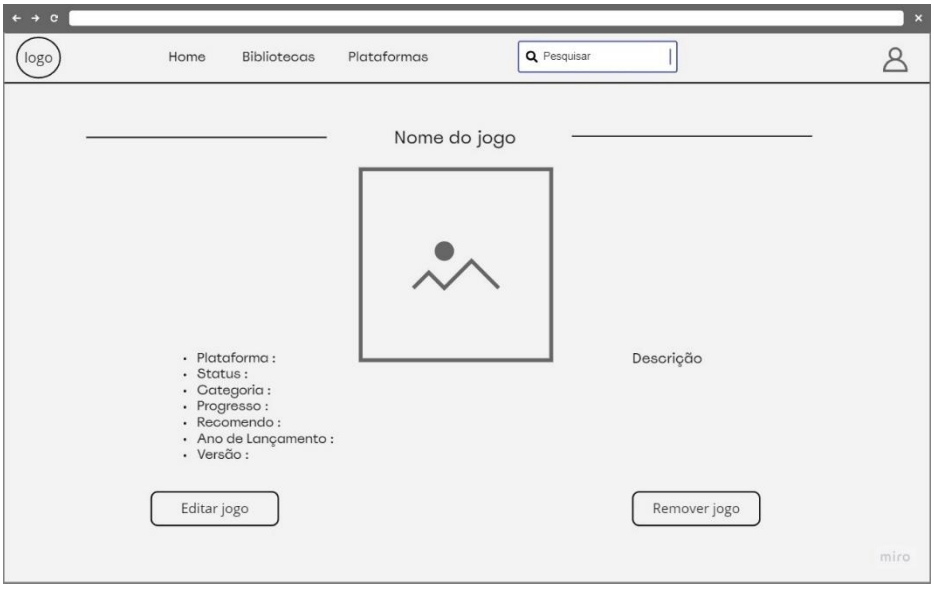
4.3 TELA INICIAL



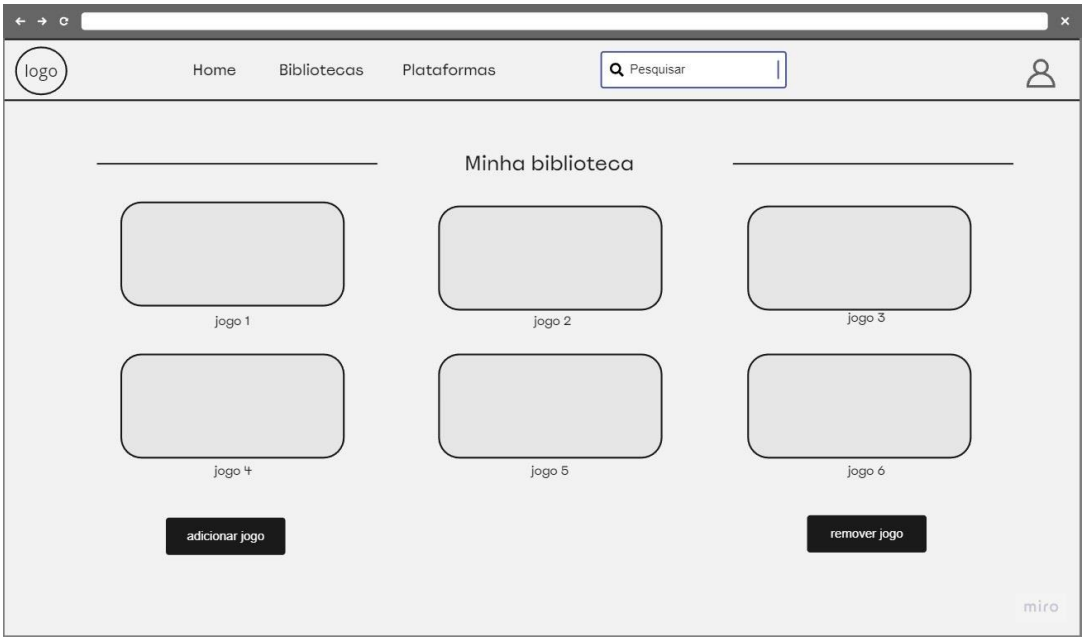
4.4 TELA BIBLIOTECA



4.5 TELA DO JOGO



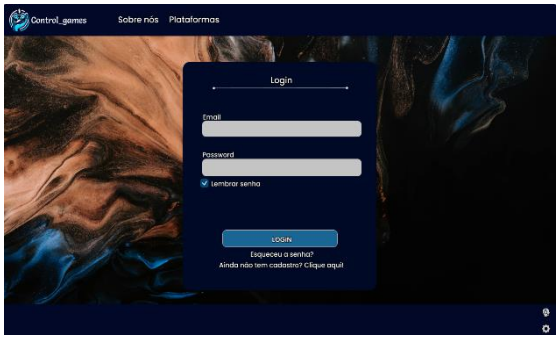
4.6 TELA DE PLATAFORMAS

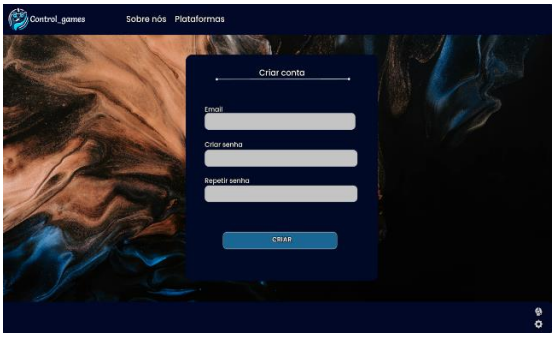


4.7 TELA DE AJUDA

5. FIGMA

5.1 TELA DE LOGIN E CADASTRO



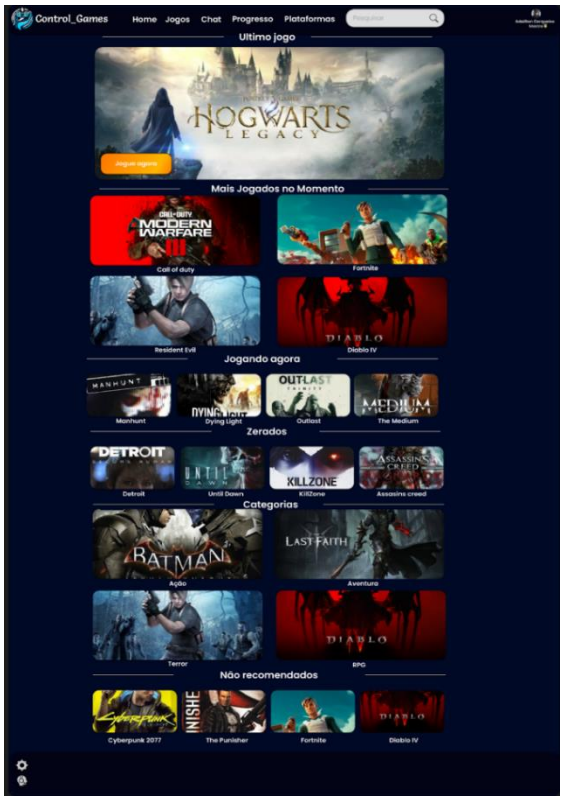


- Nas telas apresentadas foram utilizadas um background com uma temática escura para combinar com as cores da marca (azul marinho).

- Escolhemos colocar a tela de login e cadastro em páginas diferentes ao invés de deixar na mesma página, a fim de entregar um layout mais clean e familiar para os usuários novos.

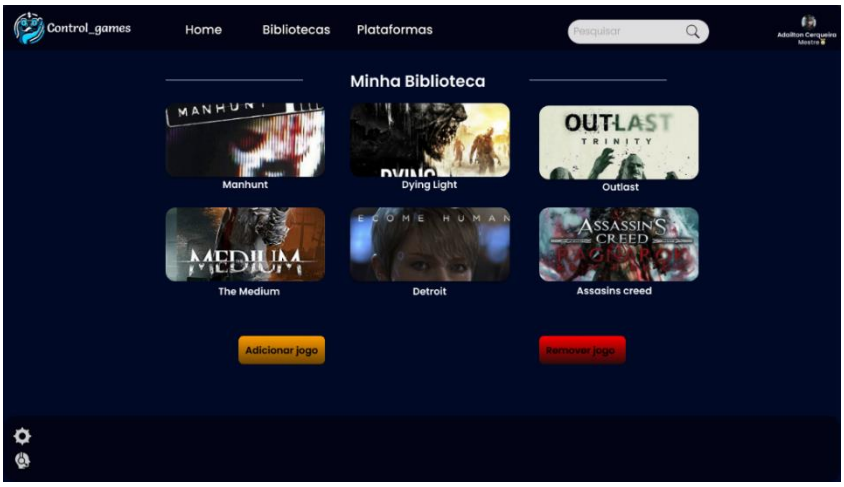
- Foram utilizadas simbolos com objetos do mundo real como o ícone de configuração e ajuda para acoplar a funcionalidade do botão a imagem.

5.2 TELA INICIAL



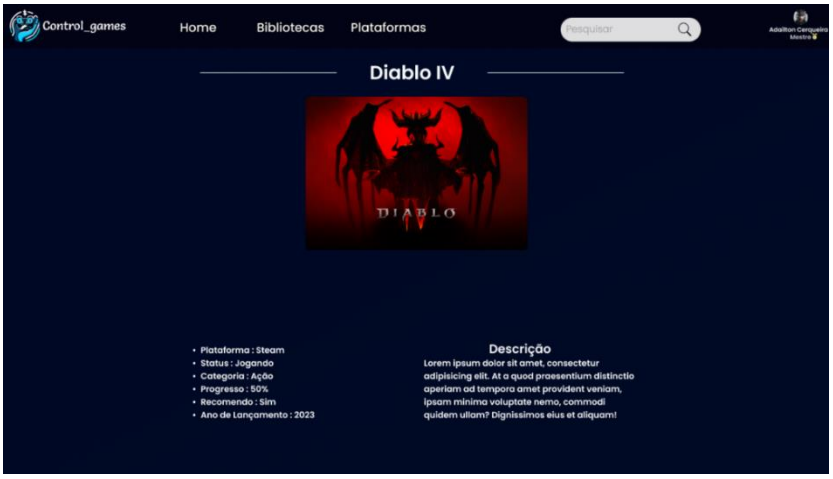
- Na tela inicial escolhemos as mesmas cores da tela de cadastro e login para manter fiel as características da marca ja apresentadas.
- No header e no footer utilizamos a cor preta para separar a parte principal da página da área de transição.
- A cor alaranjada do botão está relacionada a psicologia das cores. Bastante utilizada em lojas digitais, como a amazon, para sinalizar ao usuário uma decisão importante, que é a de compra.
- Colocamos o quadrante do “Último jogo” como a parte em destaque na parte superior e de maior escala na página para facilitar ao usuário o acesso do jogo que ele está jogando atualmente.
- A barra de navegação esta alocada na parte superior para deixar espaço para o conteúdo no body, deixando a parte principal mais clean.
- Sinalizamos as sessões em diferentes tamanhos e seus respectivos nomes para facilitar a navegação dos usuários novos.

5.3 TELA DA BIBLIOTECA



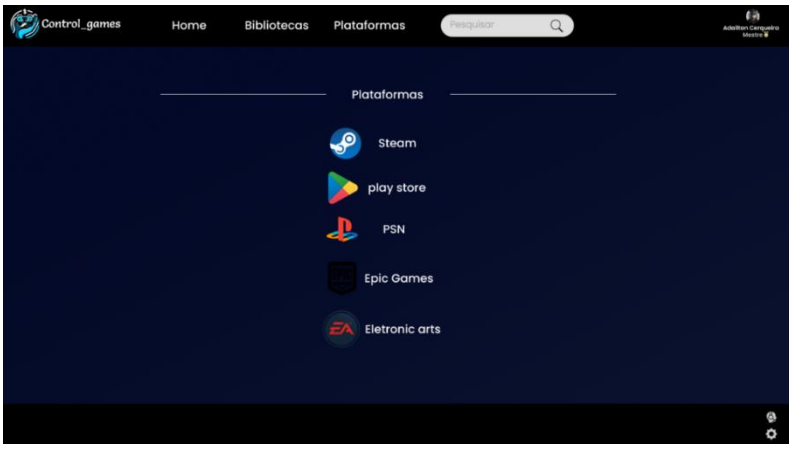
- Na tela de biblioteca usamos o mesmo sistema de cores para não causar desconfoto para o usuário.
- Colocamos um layout simples para melhor visualização do conteudo.
- Colocamos botões para adicionar e remover o jogo da biblioteca com avisos de confirmação de cada ação tomada.

5.4 TELA DO JOGO



- Na tela do jogo usamos um layout conciso para exibição das informações do jogo contendo: Plataforma, Status (jogando ou não), Gênero, Progresso, Recomendação, Ano de lançamento e Descrição.
- O layout foi projetado para mostrar as informações mais importantes do jogo

5.5 TELA DE PLATAFORMAS



- Na página de plataformas contém ícones para fácil reconhecimento das plataformas quais ele possui conta vinculada.

5.6 TELA DE AJUDA

- A página de ajuda foi criada para sanar as dúvidas de como usar o site.
- Na tela de ajuda contém os passos iniciais para usar a plataforma como: se cadastrar e acessar o site, como conectar os jogos na plataforma para pode ver o progresso no nosso site.

6. PROGRESSO DO DESENVOLVIMENTO

Durante o período de desenvolvimento, a equipe progrediu da seguinte forma:

- Criação das páginas web de acordo com os requisitos.
- Implementação do layout esponsive.
- Estilização das páginas usando CSS.
- Adição de funcionalidades JavaScript conforme especificado nos requisitos.

7. PRINCIPIOS DE USABILIDADDE UTILIZADAS

Correspondência entre o Sistema e o Mundo Real: Utilizamos linguagem e conceitos familiares aos usuários, como ícones e rótulos de botões, que refletiam claramente as ações a serem executadas.

Controle e Liberdade do Usuário: Incluímos botões de “Voltar” e a capacidade de cancelar ações para dar aos usuários controle sobre suas interações.

Consistência e Padrões: Mantivemos a consistência na aparência e no comportamento dos elementos da interface em todo o sistema, seguindo padrões de design web estabelecidos.

Prevenção de Erros: Implementamos confirmações e validações para evitar que os usuários cometessem erros, como excluir informações importantes sem confirmação.

Reconhecimento ao Invés de Lembrança: Projetamos a interface para que os usuários não precisassem lembrar informações ou comandos. Tudo o que eles precisavam estava visível e acessível.

Estética e Design Minimalista: Seguimos um design limpo e minimalista para tornar a aplicação visualmente atraente e facilitar a compreensão da informação.

Ajuda e Documentação: Incluímos dicas de ferramentas e informações de ajuda contextual sempre que necessário, para fornecer orientações adicionais aos usuários.

Escolha de cores: Ela foi escolhida pois transmite uma sensação de confiança, credibilidade e segurança. É uma cor que passa uma imagem de marca profissional e confiável.

O azul marinho também é associado à inovação e à inteligência. É uma cor que é frequentemente usada por startups e empresas de tecnologia que estão procurando se posicionar como líderes inovadores no mercado.

Aqui estão alguns dos significados específicos da cor azul marinho na tecnologia:

- Confiança: O azul marinho é uma cor que é frequentemente associada à confiança e à credibilidade. É uma cor que é frequentemente usada por empresas de tecnologia para criar uma imagem de marca profissional e confiável.

- Credibilidade: O azul marinho é uma cor que é frequentemente associada à credibilidade. É uma cor que é frequentemente usada por empresas de tecnologia para criar uma imagem de marca que é confiável

- Segurança: O azul marinho é uma cor que é frequentemente associada à segurança. É uma cor que é frequentemente usada por empresas de tecnologia para criar uma imagem de marca que é segura e confiável.

- Inovação: O azul marinho é uma cor que é frequentemente associada à inovação. É uma cor que é frequentemente usada por startups e empresas de tecnologia que estão procurando se posicionar como líderes inovadores no mercado.

- Inteligência: O azul marinho é uma cor que é frequentemente associada à inteligência. É uma cor que é frequentemente usada por empresas de tecnologia que estão procurando se posicionar como líderes inteligentes e inovadores no mercado.

8. TÉCNOLOGIAS USADAS NO PROJETO

8.1. FRONT-END

React:

- Uma biblioteca JavaScript para construir interfaces de usuário. React permite criar componentes reutilizáveis e construir interfaces de usuário eficientes.

Javascript:

- Uma linguagem de programação amplamente utilizada no desenvolvimento web para adicionar interatividade às páginas.

HTML e CSS:

- Linguagens fundamentais para o desenvolvimento web. HTML é usado para estruturar o conteúdo da página, e CSS é usado para estilizar e formatar esse conteúdo.

8.2. BACK-END

Node.js

- Um ambiente de execução JavaScript no lado do servidor que permite a construção de aplicativos escaláveis e de alta performance.

SQL

- Uma linguagem padrão para manipulação de bancos de dados relacionais. No contexto do back-end, SQL geralmente é usado para consultar e manipular dados em um banco de dados.

8.3. BIBLIOTECAS

- React (useState e useEffect): Hooks do React que permitem adicionar estado e efeitos a componentes funcionais. useState é usado para gerenciar estado, e useEffect é usado para realizar ações lado do cliente em determinados pontos do ciclo de vida do componente.

- Axios: Uma biblioteca para fazer requisições HTTP. No contexto do front-end, é comumente usado para interagir com APIs.

- React-router-dom (router, routes, Route, link e useNavigate): Uma biblioteca para navegação entre componentes React. Fornece meios para criar rotas e links em aplicativos de página única (SPA).

- LocalServerUrl: Essa URL completa é então exportada para ser utilizada em outros módulos ou arquivos do projeto. Configurar a comunicação entre o front-end e o back-end da aplicação

8.4. BACK-END

- Express: Um framework web para Node.js que simplifica a criação de APIs RESTful e aplicativos web.

- Sqlite3: Um banco de dados SQLite é utilizado. SQLite é um banco de dados relacional leve e fácil de incorporar em aplicativos.

- Cors: Um middleware utilizado para habilitar o controle de acesso a recursos de uma página web de um domínio diferente do que o site está sendo executado.

9. DETALHAMENTO DO BACKEND

9.1 FUNCIONAMENTO DO SERVIDOR

Configuração:

- Const app para receber o express()

- const port para receber a porta de funcionamento do servidor
- const db para criar um novo banco de dados
- criação de tabelas
- foi utilizado o db.run (CREATE TABLE) para cada tabela existente no projeto

Manutenção das tabelas

- app.post: inserir novos dados nas tabelas (INSERT INTO)
- app.get: puxar os dados das tabelas (SELECT)
- app.put: atualizar os dados das tabelas (UPDATE)
- app.delete: deletar os dados (DELETE)

9.2 FUNCIONAMENTO DO LOGIN

```
const handleFazerLogin = async () => {
    try {
        const response = await axios.post(`http://${LocalServerUrl}/usuarios/login`, { nickname, senha });
        alert(`Login bem-sucedido! Bem vindo: ${response.data.nickname}, você será redirecionado para o Control Games`);
        localStorage.setItem('usuarioLogado', response.data.nickname)
        console.log('Usuario Logado:', localStorage.getItem('usuarioLogado'))

        window.location.href = "/home";
        setNickname('');
        setSenha('');

    } catch (error) {
        alert(`Erro ao fazer login: Usuario ou Senha incorretos`);
        setSenha('');
    }
};
```

- Usa a biblioteca Axios para fazer uma solicitação HTTP POST para a URL http://${LocalServerUrl}/usuarios/login, passando um objeto com as propriedades nickname e senha. O resultado da solicitação é armazenado na constante response.

- localStorage.setItem('usuarioLogado', response.data.nickname): Armazena o nickname do usuário que acabou de fazer login no localStorage do navegador com a chave 'usuarioLogado'. Esse usuarioLogado será chamado durante todo o projeto nas requisições do back-end

- depois de fazer o login bem sucedido será redirecionado para a página /home

9.3 FUNCIONAMENTO DO CADASTRO DO USUARIO

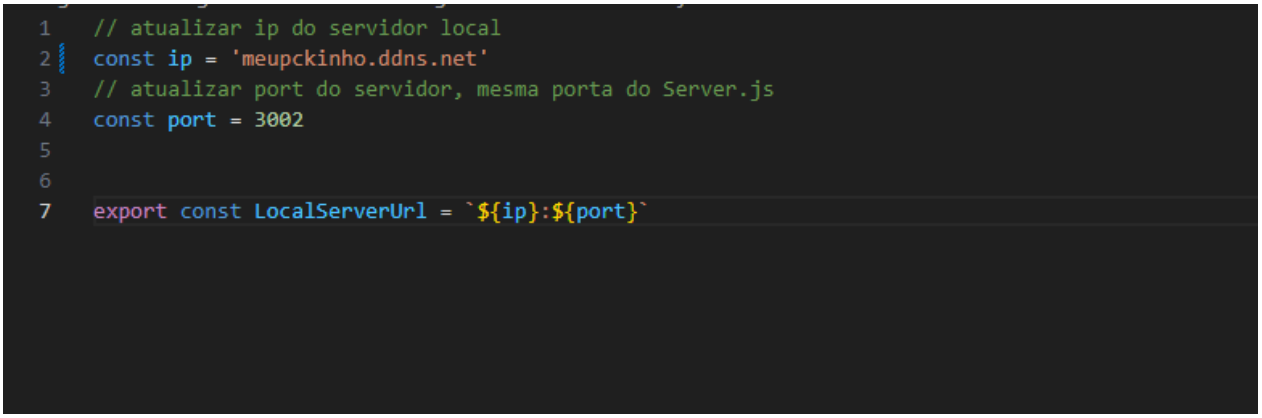
```
const handleCriarUsuario = async () => {
    try {

        if (nickname === '' || email === '' || senha === '' || confirmSenha === '' || avatar === '') {
            alert('Todos os campos são obrigatórios');
            return;
        } else if (senha !== confirmSenha) {
            alert('As senhas não coincidem. Por favor, digite novamente.');
            return;
        }

        await axios.post(`http://${LocalServerUrl}/usuarios`, { nickname, email, senha, avatar });
        alert(`Usuario cadastrado com sucesso você será redirecionado para a tela de login`);

        setNickname('');
        setEmail('');
        setSenha('');
        setAvatar('');
        setConfirmSenha('');
        navigate('/login')

    } catch (error) {
        alert(`Erro ao criar Usuario, Usuario já existe`);
        setSenha('');
        setConfirmSenha('')
    }
};
```

A função recebe os parâmetros nickname, email, senha e confirmSenha e realiza uma solicitação HTTP POST para a rota /users/create. A rota /users/create verifica se os dados do usuário são válidos e não existem no banco de dados. Se os dados forem válidos, a rota criará um novo usuário no banco de dados.

A função handleCriarUsuario() primeiro verifica se todos os campos obrigatórios estão preenchidos. Se algum campo obrigatório estiver vazio, a função exibe uma mensagem de erro.

Em seguida, a função verifica se as senhas senha e confirmSenha são iguais. Se as senhas não forem iguais, a função exibe uma mensagem de erro.

9.4 FUNCIONAMENTO DO LOCAL SERVER



```
// atualizar ip do servidor local
const ip = 'meupckinho.ddns.net'
// atualizar port do servidor, mesma porta do Server.js
const port = 3002


export const LocalServerUrl = `${ip}:${port}`
```

Esse trecho de código em Javascript tem como objetivo definir a URL do servidor local que será utilizada em uma aplicação. Vamos explicar cada parte:

const ip = 'meupckinho.ddns.net': Aqui, é atribuído um valor à constante ip. O valor é um endereço de domínio dinâmico (DDNS) chamado "meupckinho.ddns.net". Esse DDNS provavelmente é utilizado para mapear dinamicamente o endereço IP público atribuído ao seu roteador ou servidor local.

const port = 3002: Nesta linha, é atribuído um valor à constante port. O valor é o número da porta, que é 3002. A porta é um mecanismo utilizado para diferenciar diferentes serviços ou processos em um mesmo endereço IP.

export const LocalServerUrl = `${ip}:${port}`: Aqui, uma URL completa para o servidor local é criada utilizando a template string. A variável LocalServerUrl é exportada e armazena a combinação do endereço IP (ip) e o número da porta (port). Isso resulta em algo como "meupckinho.ddns.net:3002", que representa a URL completa do servidor local.

Em resumo, esse código define uma URL para um servidor local, indicando o endereço de domínio dinâmico e a porta a ser utilizada. A exportação da constante LocalServerUrl permite que outros módulos ou arquivos na aplicação possam acessar essa informação.

10. INTEGRAÇÃO COM O BACKEND

Funcionamento Base para carregar os dados nas páginas:
Exemplo 1: Página Home

```
import { Headerusuario } from '../componentes/Header'
import { Footer_interno } from '../componentes/Footer'
import { Botao_Jogaragora } from '../componentes/Botao'
import './stilos/Paginahome.css'
import { Link, useNavigate } from 'react-router-dom';
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import { LocalServerUrl } from '../configuracao/LocalServer';
```

Aqui é feita a importação dos componentes Header, e Footer que são iguais em todo o projeto para o usuário logado e demais importações para que o a página funcione corretamente.

Temos aqui também um importante que é feito em todo o projeto logado que é o local Server, que como já foi mencionado anteriormente mostra o local onde o back-end está.

```
const Paginahome = () => {

    const [jogos, setJogos] = useState([]);


    const usuarioLogado = localStorage.getItem('usuarioLogado')

    useEffect(() => {
        axios.get(`http://${LocalServerUrl}/jogos`)
            .then(response => {
                setJogos(response.data);
                console.log(response.data)
            })
            .catch(error => {
                console.error('Erro ao buscar Jogos:', error);
            });
    }, []);
```

const Paginahome = () => { ... }: Define um componente funcional chamado Paginahome. Este componente não possui propriedades (props) e é construído como uma função.

const [jogos, setJogos] = useState([]); Utiliza o hook useState para criar um estado chamado jogos. jogos é um array que será utilizado para armazenar os dados dos jogos. O setJogos é uma função que será utilizada para atualizar o estado jogos.

const usuarioLogado = localStorage.getItem('usuarioLogado'): Obtém o valor armazenado na chave 'usuarioLogado' no armazenamento local (localStorage). Essa variável armazenará o usuário logado.

useEffect(() => { ... }, []);: Utiliza o hook useEffect para realizar operações assíncronas ou efeitos colaterais. Neste caso, a função dentro do useEffect é executada uma vez, pois o array de dependências está vazio ([]). A função dentro do useEffect faz uma requisição GET usando o Axios para a URL do servidor local (http://${LocalServerUrl}/jogos). Quando a requisição é bem-sucedida, os dados dos jogos são armazenados no estado jogos usando a função setJogos. Se houver algum erro na requisição, uma mensagem de erro é exibida no console.

Resumindo, este componente React realiza uma solicitação assíncrona para obter dados de jogos do servidor local quando é montado. Os dados são armazenados no estado jogos, e o componente pode usar esses dados para renderizar a interface do usuário. O usuário logado também é obtido do armazenamento local.

```
return (
    <>
        <div id='bodyhome'>
            <div>
                <Headerusuario />
            </div>

            <div id='mainhome'>
                <section id="ultimo_jogo">
                    <div className="divisao">
                        <div className="linha"></div>
                        <p>Ultimo Jogo Adicionado</p>
                        <div className="linha"></div>
                    </div>

                    {
                        jogos
                            .filter((jogo) => jogo.usuario === usuarioLogado)
                            .slice(-1)
                            .map((jogo) => (
                                (
                                    <a key={jogo.id} >
                                        <Link to={`/game/${jogo.nomeJogo}`}>
                                            <img  src={jogo.urlCapaJogo} alt={jogo.nomeJogo} ></img>
                                        </Link>
                                    </a>
                                )
                            ))
                    }
                    {
                        jogos
                            .filter((jogo) => jogo.usuario === usuarioLogado).length === 0
                            ? <p className='retorno0'>Nenhum jogo encontrado.</p>
                            : null
                    }
```

Neste ponto, efetuamos uma filtragem na variável "jogo" para identificar o usuário logado e utilizamos o método slice para obter o último jogo adicionado. Logo abaixo, há um código com uma estrutura semelhante para lidar com a situação em que nenhum jogo é encontrado.

```
<div className="Imagem_mais_jogados">
    {
        jogos
            .filter((jogo) => jogo.usuario === usuarioLogado && jogo.status === 'Jogando')
            .slice(0, 4)
            .sort(() => Math.random() - 0.5)
            .map((jogo) => (
                (
                    <a className='Mais_jogados' key={jogo.id} >
                        <Link to={`/game/${jogo.nomeJogo}`}>
                            <img className='link_img_jogo' src={jogo.urlCapaJogo} alt={jogo.nomeJogo} ></img>
                        </Link>
                        <a className="link_jogo" >{jogo.nomeJogo}</a>

                    </a>
                )
            ))
    }
    {
        jogos
            .filter((jogo) => jogo.usuario === usuarioLogado && jogo.status === 'Jogando').length === 0
            ? <p className='retorno0'>Nenhum jogo encontrado.</p>
            : null
    }
```

Seguindo a mesma lógica, encontramos a seção de "Mais Jogados", onde utilizo o método filter(jogo) para filtrar os jogos do usuário logado. Em seguida, aplico slice para selecionar apenas quatro jogos dessa categoria. Adicionalmente, incorporo um sort() para randomizar a ordem de aparição desses jogos. Dessa forma, sempre que o usuário atualizar a página, teremos uma configuração diferente.

Rotas:

```
import React from "react";
import { BrowserRouter as Router, Routes, Route } from 'react-router-d
import { useState, useEffect } from "react";
import { LocalServerUrl } from "../configuracao/LocalServer";
import axios from "axios";
import Paginahome from "./paginahome"
import Paginabiblioteca from "./paginabiblioteca";
import Paginaplataformas from "./paginaplataformas";
import Paginasuporte from './paginasuporte'
import Paginaconfiguracao from './paginaconfiguracao'
import Paginalogin from './paginalogin'
import Paginacadastro from './paginacadastro'
import Paginacadastrojogo from './paginacasdastrojogo'
import Paginagame from './paginagame'
import Paginacadastroplataforma from './paginacadastroplataformas'
import Paginasuporteexterno from "./paginasuporteexterno";
import Paginaatualizajogo from './paginaatualizajogo'
import Paginacadastrarcategoria from "./paginacadastrarcategoria";
```

Importação dos componentes para o funcionamento da página rotas:

```
(const Rotas = () => {
   const usuarioLogado = localStorage.getItem('usuarioLogado')
   const [jogos, setJogos] = useState([]);
   useEffect(() => {
      axios.get(`http://${LocalServerUrl}/jogos`)
         .then(response => {
            setJogos(response.data);
            console.log(response.data)
         })
         .catch(error => {
            console.error('Erro ao buscar Jogos:', error);
         });
   }, []);

return (
      <Router>
         <Routes>
            <Route exact path={"/home"} element={<Paginahome />} />
            <Route exact path={"/biblioteca"} element={<Paginabiblioteca />} />
            <Route exact path={"/plataformas"} element={<Paginaplataformas />} />
            <Route exact path={"/suporte"} element={<Paginasuporte />} />
```

```
            <Route exact path={"/suporteexterno"} element={<Paginasuporteexterno />} />
            <Route exact path={"/configuracao"} element={<Paginaconfiguracao />} />
            <Route exact path={"/login"} element={<Paginalogin />} />
            <Route exact path={"/"} element={<Paginalogin />} />
            <Route exact path={"/cadastro"} element={<Paginacadastro />} />
            <Route exact path={"/cadastrojogo"} element={<Paginacadastrojogo />} />
            {/* <Route exact path={"/game"} element={<Paginagame />} /> */}
            <Route exact path={"/plataformascadastro"} element={<Paginacadastroplataforma />} />
            <Route exact path={"/paginaatualizajogo"} element={<Paginaatualizajogo />} />
            <Route exact path={"/cadastrocategoria"} element={<Paginacadastrarcategoria/>} />
          </Routes>
          {
            jogos
            .filter((jogo) => jogo.usuario === usuarioLogado)
            .map(jogo => (
                  <Routes>
                    <Route exact path={`/game/${jogo.nomeJogo}`} element={<Paginagame/>}>
                    </Route>
                  </Routes>
              ))
          }
          {
            jogos
              .map(jogo => (
                  <Routes>
                    <Route exact path={`/paginaatualizajogo/${jogo.nomeJogo}`}
element={<Paginaatualizajogo/>}>
                    </Route>
                  </Routes>
              ))
          }
        </Router>
    )
}
```

Esse código é um componente React chamado Rotas, que lida com a navegação dentro da aplicação usando o React Router.

const usuarioLogado = localStorage.getItem('usuarioLogado'): Obtém o valor armazenado na chave 'usuarioLogado' do localStorage, indica se um usuário está logado ou não para as rotas poderem funcionar.

const [jogos, setJogos] = useState([]);: Define um estado jogos utilizando o hook useState. Ele será utilizado para armazenar os dados dos jogos obtidos através de uma requisição AJAX.

useEffect(() => { ... }, []);: Utiliza o hook useEffect para executar operações assíncronas ou efeitos colaterais quando o componente é montado. Neste caso, faz uma requisição GET para obter dados de jogos do servidor local usando o Axios. Os dados são armazenados no estado jogos.

Roteamento com React Router:

Router, Routes, e Route são componentes do React Router que ajudam a definir as rotas da aplicação.

Route é utilizado para mapear URLs para componentes React específicos.

As rotas estão definidas para várias páginas, como home, biblioteca, plataformas, suporte, configuração, login, cadastro, etc.

MAPEAMENTO DINÂMICO DE ROTAS:

Após a definição estática de rotas, há um mapeamento dinâmico usando a função map nos jogos.

Para cada jogo, cria uma nova rota utilizando o componente Route. Isso cria dinamicamente rotas para cada jogo individual, por exemplo, /game/NomeDoJogo.

<Paginagame/> e <Paginaatualizajogo/>: São componentes React que são renderizados quando as rotas correspondentes são acessadas. Presumivelmente, essas páginas estão relacionadas à exibição de detalhes do jogo e à atualização de informações do jogo.

export default Rotas: Exporta o componente Rotas para que possa ser importado e utilizado em outros arquivos.

Usuários Pré-cadastrados conforme pedido:

Usuários já cadastrados:

id	nickname	email	senha
1	Victor-pio	victor.tuy@hotmail.com	3313
2	Kinho360ka	kinho360ka@example.com	3314
3	Lopes-brasil	lopes.brasil@example.com	5678
4	Eduardo-brasil	eduardo.brasil@example.com	9876
5	Victor-brasil	victor.brasil@example.com	4321
6	Hericles-lopes	hericles.lopes@example.com	6543
7	Victoria-valentina	victoria.valentina@example.com	8765
8	Mary-tuy	mary.tuy@example.com	2109
9	Juan-Medeiros	juan.medeiros@example.com	1852
10	Deison-santos	deison.santos@example.com	3498

Regras do Control-games:

Os usuários têm a capacidade de alterar sua senha, e-mail e avatar. Além disso, podem cadastrar novas categorias e plataformas. Entretanto, apenas o usuário administrador possui permissão para excluir plataformas ou categorias.

Como Inicializar o Projeto:

Rodando o projeto

- Abrir um terminal na pasta control-games-react e executar o comando: npm start
- Abrir outro terminal e na pasta control-games-server executar o comando: node server.js

Outras formas de Inicializar o Projeto:

O início do Projeto também pode ser realizado usando o endereço controlgames.ddns.net

Realizei um redirecionamento para um servidor local que possuo em casa, e o sistema estará operacional.

Isso só é possível pois temos um arquivo chamado LocalServer.js que mostra para o front end através de um dns dinâmico no no-ip onde o back-end se encontra.

A possibilidade mencionada é proporcionada pelo arquivo LocalServer.js. Esse arquivo informa ao front-end, por meio de um DNS dinâmico no No-IP, a localização do back-end. Se houver o desejo de publicar em outro servidor, basta realizar a alteração das informações contidas nesse arquivo