Projeto 2 – Pikachu

Web Mobile

André Lima 10410280

Alan Ribeiro do Carmo 10428496

Giovanni Suppo 10438719

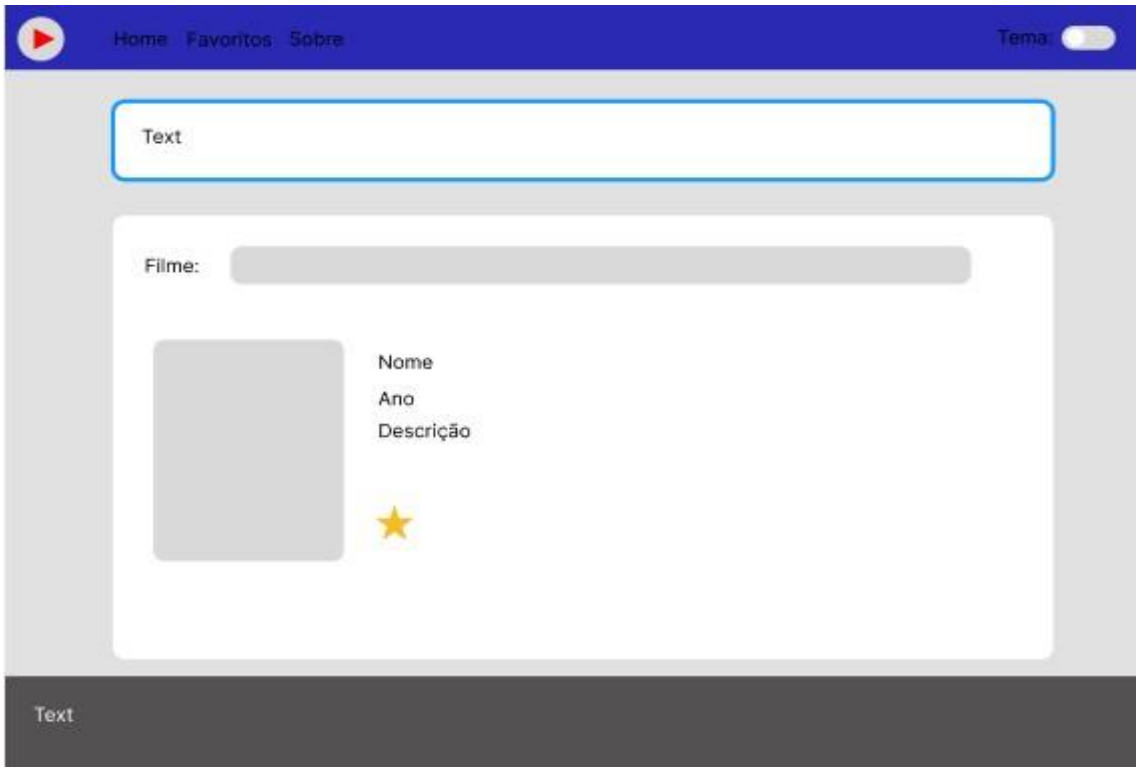
Henrique Durao 10438068

São Paulo, abril de 2024

Ideia:

Nossa ideia inicial do projeto é fazer um site onde o usuário pode pesquisar filmes e favoritá-los, adicionando-os numa lista de filmes favoritos. Para isso, iremos consumir uma API de filmes, que quando é pesquisado um filme, ele aparece na tela.

Protótipo da ideia inicial de como será o site:



Parte 1

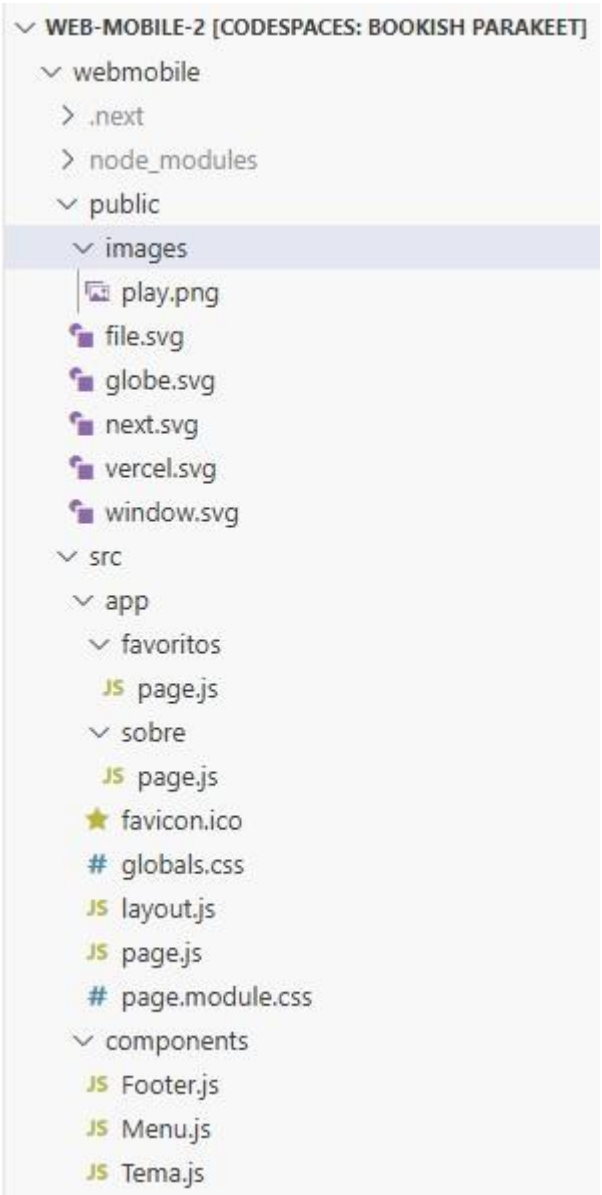
Primeiramente, criamos nosso projeto NextJS pelo codespace do Github, no terminal, foi inserido o código:

npx create-next-app@latest

E adicionamos as rotas, ESLint e diretório SRC como dependências.

Após isso, foi utilizado o código “cd webmobile” para abrir a pasta do projeto e npm run dev para rodá-lo.

Assim foram organizadas as pastas:



Na pasta public serão adicionadas as imagens, na app estão, além da página principal com o css global, estão as pastas das outras páginas (favoritos e sobre).

Na pasta componentes foram adicionados algumas partes do site que serão implementadas em todas as páginas, como o tema (claro e escuro), o menu e o rodapé.

No arquivo layout.js, as únicas mudanças foram o idioma da página para “pt-br”, o título e a descrição e na função foram colocados o <Menu /> antes da children e o <Footer /> após a children, pois essa {children} representa o conteúdo dessa página e ele ficará entre o cabeçalho e rodapé.

```
16  export const metadata = {
17    title: "Web Mobile",
18    description: "Trabalho de Web",
19  };
20
21  export default function RootLayout({ children }) {
22    return (
23      <html lang="pt-br">
24        <body className={`${geistSans.variable} ${geistMono.variable}`}>
25          <Menu />
26          {children}
27          <Footer />
28        </body>
29      </html>
30    );
31  }
```

Para o arquivo do menu fizemos da seguinte maneira:

webmobile > src > components > JS Menu.js > ...

```
1   import Link from "next/link";
2   import Tema from "./Tema";
3   import Image from "next/image";
4
5   export default function Menu() {
6       return (
7           <nav>
8               <section className = "logomenu">
9                   <Image
10                  src = "/images/play.png"
11                  width = {50}
12                  height = {50}
13                  alt = "Logo"
14                  className="imagem"
15                  />
16                  <ul>
17                      <li><Link href = "/" className="links">Home</Link></li>
18                      <li><Link href = "/favoritos" className="links">Favoritos</Link></li>
19                      <li><Link href = "/sobre" className="links">Sobre</Link></li>
20                  </ul>
21              </section>
22              <Tema />
23          </nav>
24      );
25  }
```

Criamos a função default, nela, criamos o css com a <nav> para o menu, dentro dela, criamos uma <section> para separar as informações e colocamos o <Image>, que pelo NextJS a imagem deve ser adicionada dessa forma, com o src linkando a imagem (que

está na public), depois é necessário colocar a altura e largura (mas que podem ser alteradas no CSS pela className e o alt com a descrição.

Então, adicionamos a <ul> com as <li> que estão os links das outras páginas. O link deve estar numa tag <Link>, que referencia o nome da pasta onde estão.

Parte 2

Além de alguns ajustes no CSS para melhor responsividade e visualização, a principal mudança para essa segunda parte são as rotas dinâmicas, para isso, criamos dentro da pasta filmes, uma pasta chamada [filmes] e criamos um page.js dentro dela, para que seja possível pesquisar o filme de acordo com o código dele.

No [filmes] → page.js foi implementado dessa forma:

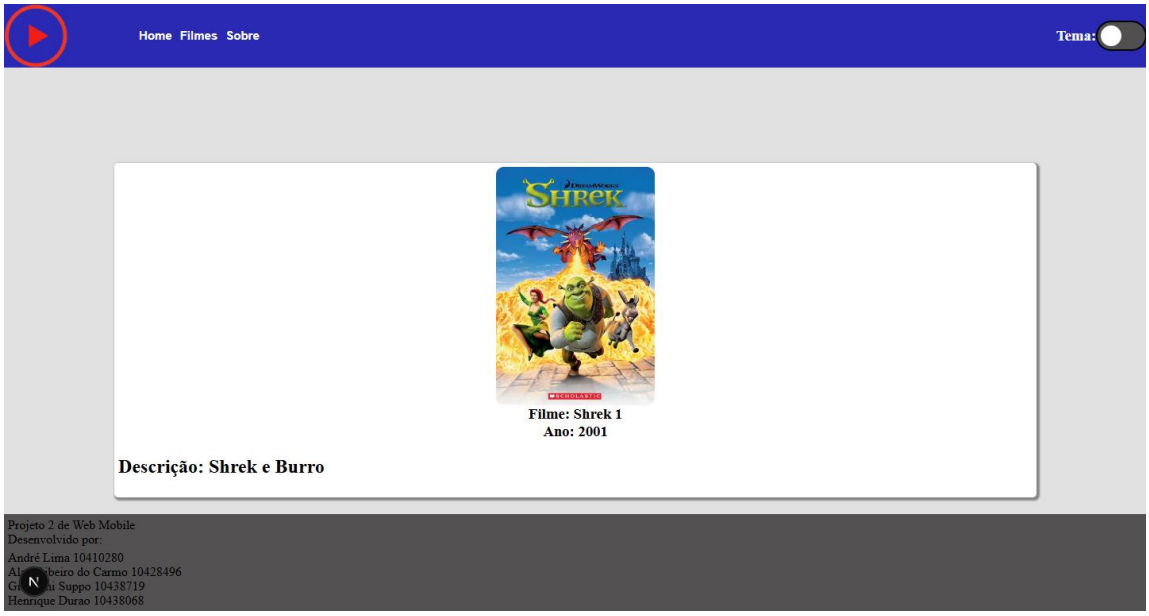
webmobile > src > app > filmes > [filme] > JS page.js > Filmes

```
1  import Image from "next/image";
2
3  export default async function Filmes ({params}) {
4      const {filme} = await params;
5
6      const codigoFilme = Number(filme);
7
8      const filmes = [
9          {codigo: 1, nome: "Shrek 1", ano: 2001, descricao: "Shrek e Burro", capa: "/images/shrek.jpg"},
10         {codigo: 2, nome: "Velozes e Furiosos 1", ano: 2001, descricao: "carro", capa: "/images/velozes.jpg"},
11         {codigo: 3, nome: "EuroTrip", ano: 2004, descricao: "besteirol", capa: "/images/eurotrip.jpg"}
12     ]
13
14     const resposta = filmes.filter((f) => f.codigo === codigoFilme);
15
16     if(resposta.length === 0){
17         return <h1>Página não encontrada</h1>
18     }
19
```

Criamos uma função assíncrona que pode receber algum parâmetro, facilitando a próxima entrega. Foi implementado um array com o código, nome, ano, descrição e imagem do filme (como nessa entrega ainda não é obrigatória a implementação da API, simulamos como se tivesse a API).
Foi criada uma constante codigoFilme, de forma que o código do filme sempre precisará ser um número. Depois foi implementado um filtro para os códigos do filme.

```
20
21      return(
22          <main className="mainFilme">
23              <article className="article1">
24                  <section className="centralizado">
25                      <Image
26                          width= {200}
27                          height= {300}
28                          src = {resposta[0].capa}
29                          alt = {resposta[0].nome}
30                          className="capa"
31                      />
32                      <h1 className="nome filme">Filme: {resposta[0].nome}</h1>
33                      <h1> Ano: {resposta[0].ano}</h1>
34                  </section>
35                  <h1>Descrição: {resposta[0].descricao}</h1>
36              </article>
37          </main>
38      )
39  }
```

A página sempre vai retornar o filme buscado. Nela aparecerá a capa do filme com o nome e ano abaixo, seguido da descrição.
A página ficará assim:



Na pasta filmes → page.js, implementamos dessa maneira, novamente simulando a lista.

webmobile > src > app > filmes > JS page.js > Filme > filmes.map() callback

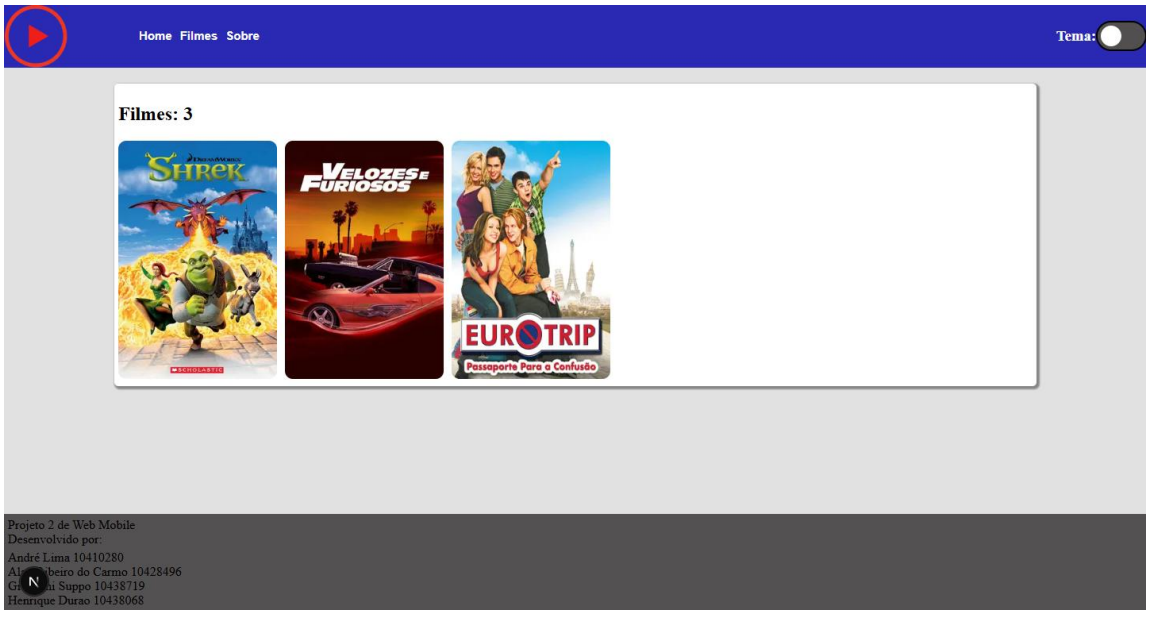
```
1   import Image from "next/image";
2   import Link from "next/link";
3
4   export default function Filme() {
5
6       const filmes = [
7           {codigo: 1, nome: "Shrek 1", ano: 2001, descricao: "Shrek e Burro", capa: "/images/shrek.jpg"},
8           {codigo: 2, nome: "Velozes e Furiosos 1", ano: 2001, descricao: "carro", capa: "/images/velozes.jpg"},
9           {codigo: 3, nome: "EuroTrip", ano: 2004, descricao: "besteirol", capa: "/images/eurotrip.jpg"}
10      ]
11
12      const qtdFilmes = filmes.reduce((acumulador) => {
13          return acumulador + 1;
14      },0);
15
```

Há um reduce que o acumulador soma a quantidade de filmes da lista.

A página retorna a lista de filmes com a quantidade de filmes que estão inseridos, a partir do map que mapeia os filmes do array e a partir dos códigos, apresenta na página a capa dos filmes. Caso o usuário clique na página do filme, ele é redirecionado para a /filmes/[filme], onde verá mais informações do filme escolhido, que foi apresentado anteriormente.

```
17      return(
18          <main className="mainPrincipal">
19              <article className="article1">
20                  <h1>Filmes: {qtdFilmes}</h1>
21                  <section className="capaFilmes">
22                          {
23                              filmes.map((f) => {
24                                  return(
25                                      <div key = {f.codigo} className="divFilmes">
26                                          <Link href = {'\\filmes\\'+f.codigo}>
27                                              <Image
28                                                  width= {200}
29                                                  height= {300}
30                                                  src = {f.capa}
31                                                  alt = {f.nome}
32                                                  className="capa"
33                                              />
34                                          </Link>
35                                      </div>
36
37
38                                  )
39                              })
40                          }
41                  </section>
42              </article>
43          </main>
44      );
45  }
```

A página fica assim:



Nessa entrega, não vamos apresentar a parte da API como algo já implementado, por conta que está no início, por conta de ser algo mais trabalhoso. No momento o código está assim:

webmobile > src > components > Pesquisa.jsx > Pesquisa > buscar

```jsx
export default function Pesquisa(){
    const [nomes, setNomes] = useState("");
    const [filmes, setFilmes] = useState([]);
    const [erro, setErro] = useState(null);
    const [ano, setAno] = useState("");
    const [descricao, setDescricao] = useState("");


    const buscar = async() => {
        try{
            const resp = await fetch("https://api.themoviedb.org/3/search/movie?api_key=d3bb84ede80dc92a552c391613321c4b&query="+nomes);
            const json = await resp.json();
            if(json.error){
                setErro(json.error);
                setFilmes([]);
            }else{
                setFilmes(json.title);
                setErro(null);
            }
        }catch(error){
            setErro("Filme não encontrado")
        }
    }
```

```jsx
    useEffect(() => {
        buscar();
    }, []);

    const filmesJSX = filmes.map((f) => {
        return (
            <div>

            </div>
        )
    })
}
```

Foram criadas várias constantes com useState para mudar o estado da página conforme forem utilizadas. A função buscar é assíncrona e tudo que é feito dentro dela está entre try{ }catch{ }, de forma que os erros serão tratados. É feito um fetch que busca as informações da API e apresenta uma mensagem de erro se for o caso.

Parte 3

Para essa entrega, foram adicionadas ou modificadas muitas coisas no projeto, então, alguns componentes iremos relembrar do zero como serão montados.

Nessa parte do projeto, implementamos a API gratuita do The Movie Database (TMDb), em que será feita a busca pelos filmes. Para isso, foi necessário fazer uma conta no site oficial deles para conseguir uma key e entender como fazer as pesquisas dentro da API.

Primeiramente, criamos o componente Busca.jsx, em que ele irá pegar as informações da API, ficou dessa maneira:

webmobile > src > components > Busca.jsx > Busca

```jsx
export async function Busca (query){
    try {
      const url = `https://api.themoviedb.org/3/search/movie?api_key=d3bb84ede80dc92a552c391613321c4b&language=pt-br&query=${encodeURIComponent(query)}`;

      const resp = await fetch(url);
      const json = await resp.json();

      if (json.errors || json.success === false) {
        return {filmes : [], erro: "Filme não encontrado."};
      } else {
        return {filmes: json.results, erro: null};
      }
    }catch (error) {
      return {filmes: [], erro: "Erro na busca"};
    }
}

```

Primeiro, é necessário criar uma export async function, para que essa função seja exportada em outro componente e é necessário o async por conta de ser uma função assíncrona que irá aguardar (await) uma promessa, que será o link da API.
O código está dentro de um try{}catch{} para tratar exceções, então quando foi capturado um erro (error), retornará uma lista vazia e a frase “Erro na busca”.
Foram criadas 3 constantes, a primeira com o link da API, em que foi colocada a chave fornecida pelo site no “key= ...”, “language=pt-br” para indicar que pegará as informações em português e query=${encodeURIComponent(query)} insere o termo de busca na URL, codificando caracteres especiais (como espaços).

Nas linhas abaixo,

const resp = await fetch(url);
const json = await resp.json();

foi criada uma constante em que aguarda (await) a requisição HTTPS GET (fetch) da url, enquanto a const json converte a chamada da API num JSON.

Depois disso, há uma verificação se a chamada deu erro ou sucesso, se json.errors, retorna uma lista vazia e uma mensagem de erro, se não, retorna os resultados e o erro é nulo.

Após isso, foi criada a Pesquisa.jsx na pasta components, que será usada para fazer a pesquisa dos filmes na tela inicial do site.

webmobile > src > components > Pesquisa.jsx > Pesquisa

```jsx
'use client';

import Link from 'next/link';
import { useState } from 'react';
import { Busca } from './Busca';

export default function Pesquisa() {
  const [query, setQuery] = useState('');
  const [filmes, setFilmes] = useState([]);
  const [erro, setErro] = useState(null);

  const handleBuscar = async () => {
    const {filmes, erro} = await Busca(query);
    setFilmes(filmes);
    setErro(erro);
  };
```

O ‘use cliente’ significa que será renderizado no lado do cliente (navegador), e não do servidor.

Foram importados o Link e o useState, além da função Busca.

Após isso, se inicia a função, criando useState para a query, os filmes e o erro, todos iniciando de uma forma vazia ou nula.

Após isso, a função handleBuscar é assíncrona, para aguardar a promessa da Busca, que será feita com a palavra digitada (query), após isso, atualiza o estado de filmes e erro, com novos filmes ou novos erros.

Após isso, começa o retorno de nossa função:

```jsx
  return (
    <section className="secaoBarra">
      <section className= "caixaPesquisa">
        <input
          type="text"
          value={query}
          onChange={(e) => setQuery(e.target.value)}
          placeholder="Digite o nome do filme"
          className="barraPesquisa"
        />
        <button onClick={handleBuscar} className="ml-2 p-2 bg-blue-500 text-white">
          Buscar
        </button>

        {erro && <p className="text-red-500 mt-2">{erro}</p>}

      </section>
```

Ela retorna uma section que contém um input para o usuário escrever o nome do filme, onde o valor dentro dele será a query, que será mudada quando clicar para buscas, no button, que quando clica, chama a função handleBuscar.
Após isso, há uma verificação de erro, que quando não for nulo, exibe a mensagem de erro.

Depois temos a section onde ficará os resultados da pesquisa.

```
36          <section className="capaFilmes">
37            {filmes.map((f) => (
38              <section key={f.id} className="sectionFilmes">
39                {f.poster_path ? (
40                  <img
41                    src={`https://image.tmdb.org/t/p/w200${f.poster_path}`}
42                    alt={f.title}
43                    className="capa"
44                  />
45                ) : (
46                  <p>Sem imagem</p>
47                )}
48                <h2>{f.title}</h2>
49                <p>{f.release_date || 'Data desconhecida'}</p>
50                <p>{f.overview ? f.overview : 'Sem descrição'}</p>
51                <Link href = {`/filmes/${f.id}`}>
52                  <button>Ver Mais</button>
53                </Link>
54              </section>
55            ))}
56          </section>
57        </section>
58      );
59    }
60
```

Ele mapeia o “filmes”, que está guardado o resultado da pesquisa (query), e busca na API a imagem da capa, o título da obra, a data de lançamento e a descrição. Além disso, há um botão “Ver Mais”, que quando o usuário clica, redireciona para o /filmes/id, que apresenta o filme mais detalhado, é a página onde o usuário poderá favoritar os filmes.

Outro componente criado foi o BuscaID, que pegará os resultados de um filme específico de acordo com o ID, ele é necessário para apresentar o filme individualmente no /filmes/id.

webmobile > src > components > BuscaID.jsx > BuscaFilmePorId

```jsx
export async function BuscaFilmePorId(id) {
  try {
    const url = `https://api.themoviedb.org/3/movie/${id}?api_key=d3bb84ede80dc92a552c391613321c4b&language=pt-BR`;
    const resp = await fetch(url);
    const json = await resp.json();

    if (json.success === false || json.status_code) {
      return null;
    }

    return json;
  } catch (error) {
    return null;
  }
}
```

Foi criada uma função assíncrona, que esperará pelo resultado da API (assim como nos componentes anteriores), porém, dessa vez buscará um único filme a partir de seu ID, então quando o usuário pesquisar o filme na página inicial e clicar em “Ver Mais”, fará uma busca apenas desse filme em específico a partir desse componente e apresentará na página /filmes/id (veremos posteriormente). Se a busca der certo, apresenta o resultado do json, caso não, apresenta uma mensagem de erro e retorna nulo.

Após isso, criamos o componente Estrela, que será a responsável por salvar o filme na lista dos filmes salvos.

webmobile > src > components > Estrela.jsx > ...

```jsx
'use client'

import { useEffect, useState } from "react";

export default function Estrela({filme}){
    const [curtido, setCurtido] = useState(false);

    useEffect(() => {
        const filmesCurtidos = JSON.parse(localStorage.getItem('filmesCurtidos') || '[]');
        const jaCurtido = filmesCurtidos.some(f => f.id === filme.id);
        setCurtido(jaCurtido);
    }, [filme.id]);

    const salvarFilme = () => {
    const filmesCurtidos = JSON.parse(localStorage.getItem('filmesCurtidos') || '[]');

    if (!curtido) {
      filmesCurtidos.push(filme);
      localStorage.setItem('filmesCurtidos', JSON.stringify(filmesCurtidos));
    } else {
      const filtrados = filmesCurtidos.filter(f => f.id !== filme.id);
      localStorage.setItem('filmesCurtidos', JSON.stringify(filtrados));
    }

    setCurtido(!curtido);
  };
```

Primeiramente, o “use cliente” para renderizar do lado do cliente (navegador). Foram importados o useState e useEffect.
A função recebe como props o {filme}, que foi buscado. Ela inicia o estado curtido como falso, para que o estado mude quando o usuário clicar na estrela.
O useEffect verificará se o filme está curtido, olhando no localStorage (armazenamento local) os filmes curtidos, depois verificando se esse filme do id escolhido já foi curtido e mudará o estado curtido.

A função salvarFilme verificará novamente os filmes curtidos no localStorage e caso não esteja curtido, ele adiciona o filme na lista quando ele for curtido e caso ele já foi curtido, ele removerá do localStorage quando clicar na estrela.
Após isso, alterna o estado curtido com setCurtido(!curtido);.

Após isso, temos o retorno:

```
return (
    <svg
        onClick={salvarFilme}
        xmlns="http://www.w3.org/2000/svg"
        fill= {curtido ? "yellow" : "white"}
        viewBox="0 0 24 24"
        strokeWidth={1.5}
        stroke="currentColor"
        className="estrelaCurtir">
        <path strokeLinecap="round" strokeLinejoin="round" d="M11.48 3.499a.562.562 0 0 1 1.04 0l2.125 5.111a.563.563 0 0 0 .475.345l5.518.442c.499.04.701.663.321.988
    </svg>
)
};
```

Ele tem um svg importado a partir do <path> de um site de sprites para sites. Há um onClick, para quando o usuário clicar na imagem, executa a função salvarFilme e ela muda a cor para branco ou amarelo, de acordo se foi curtido ou não.

Após isso, fizemos as alterações na page Filmes:

webmobile > src > app > filmes > JS page.js > ...

```
'use client'

import Image from "next/image";
import Link from "next/link";
import { useEffect, useState } from "react";

export default function Filme() {
    const [filmes, setFilmes] = useState([]);


    useEffect(() => {
        const filmesCurtidos = JSON.parse(localStorage.getItem('filmesCurtidos') || '[]');
        setFilmes(filmesCurtidos);
    }, []);

```

Inicialmente foi adicionado o use client com os imports, após isso, criamos a função e nela está o estado filmes, que inicialmente está vazio, após isso, tem o useEffect que verificará o filmes curtidos que estão no localStorage e mudará o estado do filmes, adicionando os filmes que estão curtidos e alocados no localStorage.

```
    return(
        <main className="mainPrincipal">
            <article className="article1">
                <h1>Filmes: {filmes.length}</h1>
                <section className="capaFilmes">
                        {
                            filmes.map((f) => {
                                return(
                                    <section key = {f.id} className="sectionFilmes">
                                        <Link href = {'\\filmes\\'+f.id}>
                                            <Image
                                                width= {200}
                                                height= {300}
                                                src = {`https://image.tmdb.org/t/p/w200${f.poster_path}`}
                                                alt = {f.title}
                                                className="capaLista"
                                            />
                                        </Link>
                                    </section>


                                )
                            })
                        }
                </section>
            </article>
        </main>
    );
}
```

Após isso, há o retorno do conteúdo da página, primeiro ela apresentará o tamanho da lista (a quantidade de filmes), após isso, mapeia os filmes e apresenta a imagem da capa deles e tem um <Link> vinculado à imagem, que quando o usuário clica, redireciona para a página /filmes/id, que apresenta mais informações do filme.

Após isso, fizemos as alterações no /filmes/id, a página em que conterá o componente Estrela para salvar os filmes e mais conteúdo sobre o filme.

webmobile > src > app > filmes > [filme] > JS page.js > Filmes

```
import { BuscaFilmePorId } from "@/components/BuscaID";
import Estrela from "@/components/Estrela";
import Image from "next/image";

export default async function Filmes ({params}) {
    const {filme} = await params;
    const dados = await BuscaFilmePorId(filme);

    if(!dados) {
        return <h1>Filme não encontrado</h1>
    }
```

Inicialmente há os imports, e criamos a função assíncrona que aguarda os parâmetros e os dados do filme a partir da busca pelo ID.
Se os dados forem nulos ou errados, ele apresenta uma mensagem de filme não encontrado.

E o return:

```
    return(
        <main className="mainFilme">
            <article className="article1">
                <section className="centralizado">
                    {dados.poster_path ? (
                       <Image
                        width= {200}
                        height= {300}
                        src = {`https://image.tmdb.org/t/p/w300${dados.poster_path}`}
                        alt = {dados.title}
                        className="capa"
                    />
                    ) : (
                        <h1>Sem imagem</h1>
                    )}
                    <h1 className="nome filme">Filme: {dados.title}</h1>
                    <h1> Ano: {dados.release_date?.split('-')[0] || "Desconhecido"}</h1>
                    <Estrela filme = {dados}/>
                </section>
                <p>Descrição: {dados.overview || "Sem descrição."}</p>
            </article>
        </main>
    )
}
```

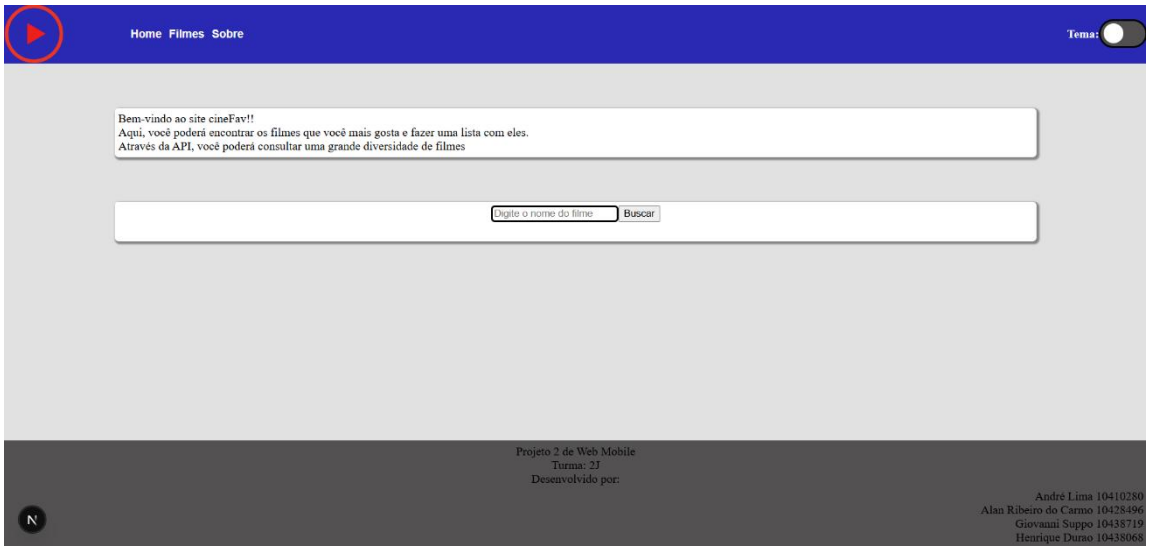
No return, ele verifica se o filme tem uma imagem, e apresenta ela a partir da imagem que a API fornece, caso não tenha, ele apresenta uma mensagem.
Após isso, ele apresenta o título, a data de lançamento, a descrição e a estrela para curtir.
Além disso, para não dar erro na busca da imagem dos filmes, no arquivo next.config.mjs, colocamos o domínio da API para as imagens:

webmobile > JS next.config.mjs > default

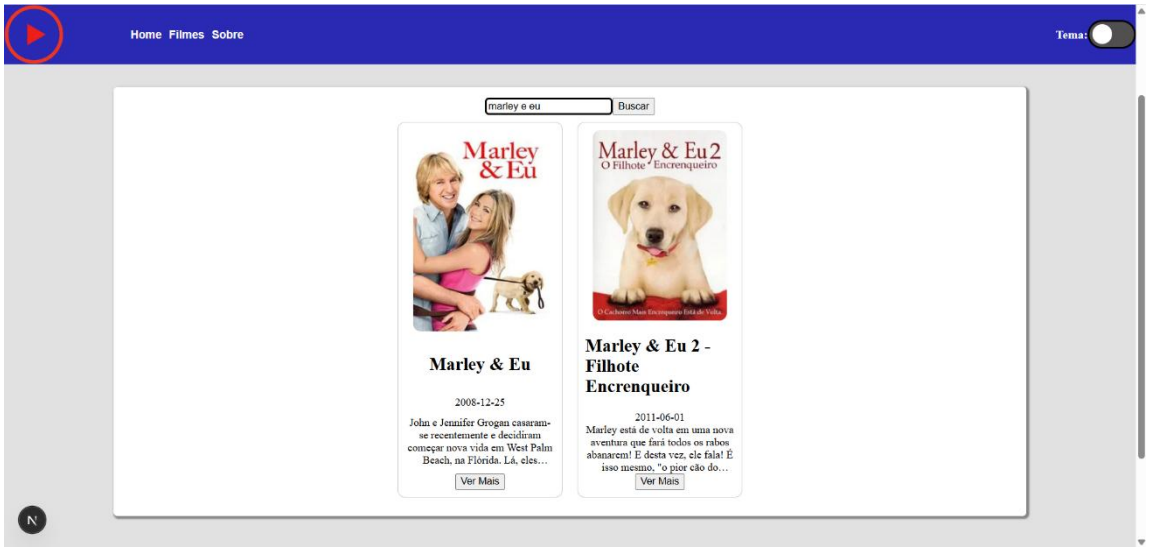
```
/** @type {import('next').NextConfig} */
const nextConfig = {
    images: {
    domains: ['image.tmdb.org'],
  },
};

export default nextConfig;

```

Como o site está?

Página inicial:



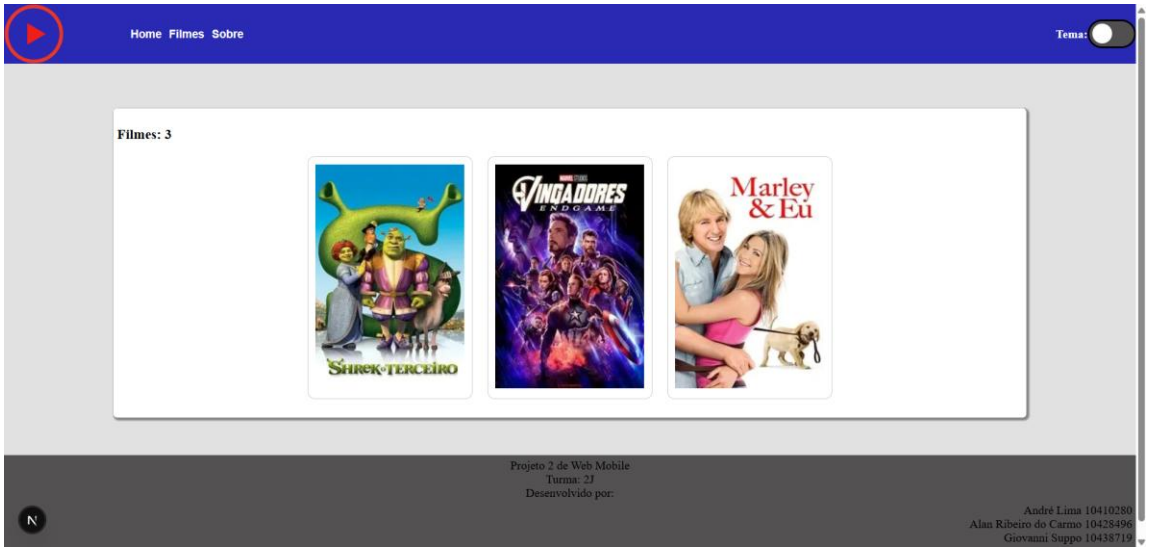
Quando o usuário pesquisa um filme:



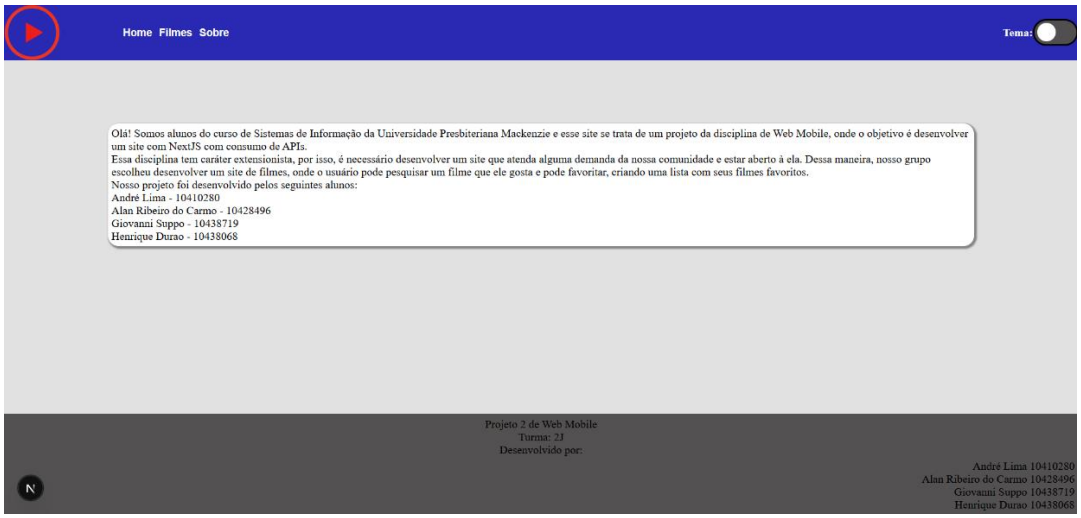
Quando o usuário clica em “Ver Mais”, abre o filmes/id, com mais informações do filme e é a página onde ele pode favoritar esses filmes.

Quando o usuário favorita o filme, a estrela fica amarela:

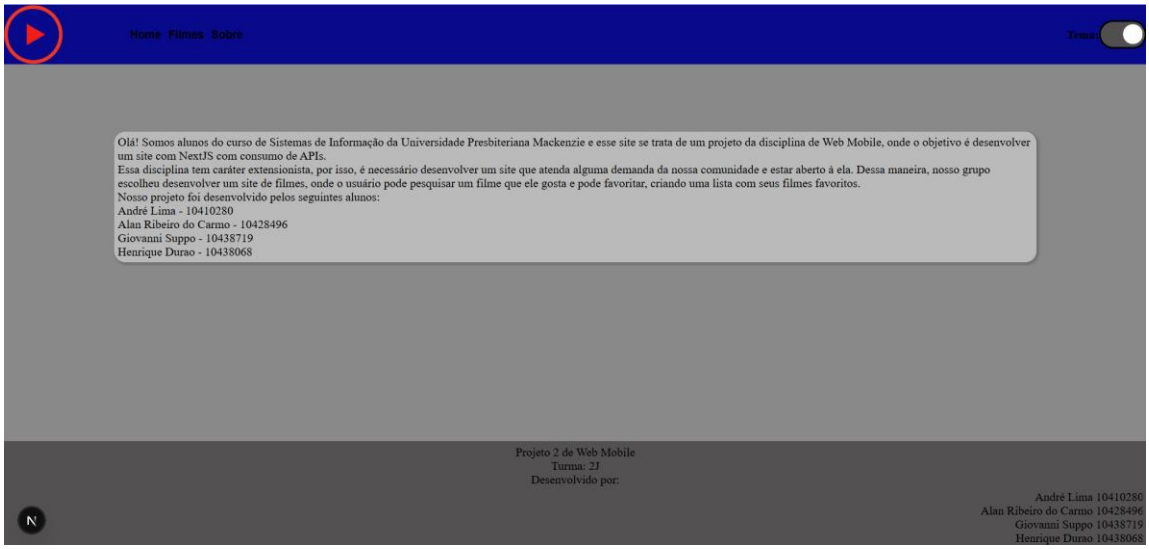
Então, na página filmes, ficam os filmes favoritados e a quantidade de filmes. Para desfavoritar, o usuário precisa clicar na imagem do filme, que abrirá novamente a página dele e deverá clicar novamente na estrela, ela ficará branca e tirará o filme da lista de curtidos.

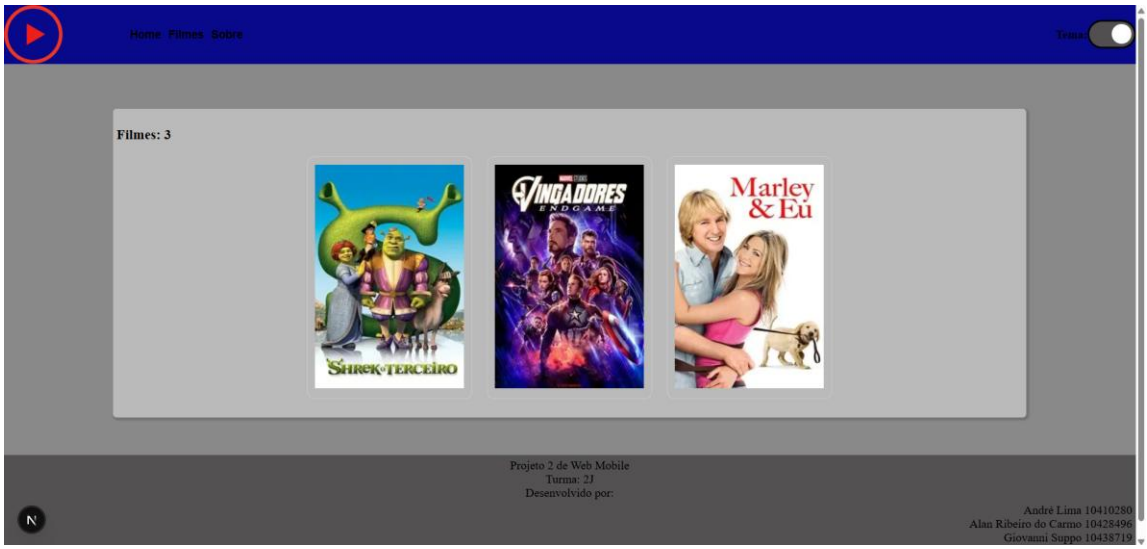


A página “sobre”:



As páginas também podem ficar com tema escuro quando é clicado no botão que há no canto direito do menu. Exemplos:





O que aprendemos com esse projeto?

Com as aulas de Web Mobile desse semestre e o desenvolvimento desse projeto, aprendemos a criar sites mais incrementados, com novas funcionalidades, utilizando o Next JS para desenvolver o front end e o back end ao mesmo tempo, utilizando componentes do React. Tivemos muito conteúdo e aprendemos muito nesse semestre e com certeza iremos continuar aprimorando nosso conhecimento sobre web.