



Tutorial do projeto de Web mobile - Pichu

1. GRUPO

Nome: Moisés Santos
RA: 10419955

Nome: Pedro Moreno Campos
RA:10390807

Professora: Paula Leite

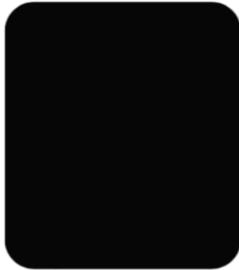
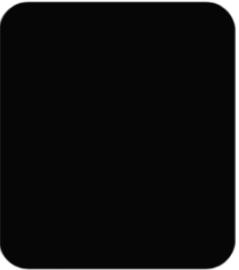


Introdução

A ideia surgiu a partir do amor que os integrantes possuem por filmes, então a ideia de criar um site com um tema de catálogo de filmes onde o usuário conseguirá encontrar diversos filmes, separados por categorias específicas, e a partir do momento que o mesmo clicar no filme desejado, ele será direcionado para uma página onde terá classificação indicativa, nota no metaquitic, descrição, e entre outras informações.

Segue abaixo uma exemplificação do que será o site:

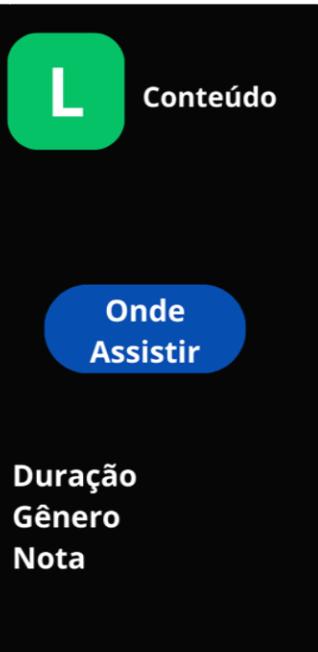
Catálogo de Filme



Nome
Descrição

Nome
Descrição

Nome
Descrição



Filme X

L Conteúdo

Onde Assistir

Duração
Gênero
Nota



Pré-requisitos

- Ter o Node.js instalado (versão recomendada LTS — 18.x ou superior).
- Ter um editor de código (recomendado: VSCode).

Primeiros Passos

Instalando o Node.js

1. Acesse: <https://nodejs.org/>
2. Baixe e instale a versão LTS (Long Term Support).

Criando um projeto Next.js

Crie uma pasta em um diretório. No terminal vá até essa pasta e execute o comando “npx create-next-app@latest”.

Dê um nome para o projeto (ou pressione Enter para usar o padrão).

Selecione:

- TypeScript? -> Yes
- ESLint? -> No
- Tailwind CSS? -> No
- src/ directory? -> No
- App Router (recommended)? -> No
- Customize import alias? -> No

Criando o arquivo de conexão com API

Configurando a chave da API

- Entre no site TheMovieDB.org, cadastre-se e gere um token para API
- Crie um arquivo chamado .env.local na raiz do projeto.
- Adicione sua chave da API TMDb:

```
TMDB_API_KEY=XXXXXXXXXXXXXXXXXXXXXX
```

Criando o arquivo api.ts dentro da pasta lib

```
const API_KEY = process.env.TMDB_API_KEY;
const BASE_URL = 'https://api.themoviedb.org/3';
```

- **API_KEY**: Pega a chave da API que você salvou no arquivo .env.local para não deixá-la exposta no código. A variável de ambiente TMDB_API_KEY guarda essa chave.
- **BASE_URL**: URL base da API do TMDb. Todos os endpoints começam com isso.

```
export interface Filme {
  id: number;
  titulo: string;
  descricao: string;
  ano: number;
  capa: string;
```



```
    classificacao: string;
    duracao: string;
    genero: string;
    nota: string;
}
```

- Serve para tipar os objetos de filme no TypeScript.
- Isso ajuda no autocomplete, validação de tipos e segurança no desenvolvimento.
- Define os dados que você irá trabalhar na aplicação.

```
export async function getFilmePorId(id: string): Promise<Filme | null> {
  const res = await
fetch(`https://api.themoviedb.org/3/movie/${id}?api_key=${API_KEY}&language=pt-BR`);

  if (!res.ok) {
    return null;
  }

  const data = await res.json();

  const filme: Filme = {
    id: data.id,
    titulo: data.title,
    descricao: data.overview,
    ano: parseInt(data.release_date?.slice(0, 4)) || 0,
    capa: data.poster_path ?
`https://image.tmdb.org/t/p/w500${data.poster_path}` :
'/images/placeholder.png',
    classificacao: data.adult ? '18+' : 'Livre',
    duracao: data.runtime ? `${data.runtime} min` : 'N/A',
    genero: data.genres?.map((g: any) => g.name).join(', ') || 'N/A',
    nota: data.vote_average ? data.vote_average.toFixed(1) : 'N/A',
  };

  return filme;
}
```

Busca os detalhes de um filme específico pelo ID.

- Faz uma requisição para /movie/{id}.
- Verifica se a resposta foi bem-sucedida.
- Organiza os dados (id, título, ano, capa, gênero, duração...).
- Retorna um objeto no formato Filme.

```
export async function getFilmesPopulares() {
  const res = await
fetch(`https://api.themoviedb.org/3/movie/popular?api_key=${API_KEY}&language=pt-BR`);
```



```
if (!res.ok) {
    throw new Error('Erro ao buscar filmes populares');
}

const json = await res.json();
return json.results;
}
```

Busca os filmes populares.

- Faz uma requisição para /movie/popular.
- Verifica se deu certo.
- Retorna a lista de filmes como vem da API (você pode tentar formatar antes)

Gerenciamento de favoritos

Arquivo contexto/FavoritosContext.tsx

```
import { createContext, useEffect, useState, ReactNode } from 'react';
```

Importa hooks e tipos do React.

```
interface FavoritosContextType {
    favoritos: number[];
    toggleFavorito: (id: number) => void;
    isFavorito: (id: number) => boolean;
}
```

Define quais dados e funções o contexto irá fornecer:

- Lista de favoritos (favoritos).
- Função para adicionar/remover (toggleFavorito).
- Função para verificar se está favoritado (isFavorito).

```
export const FavoritosContext = createContext<FavoritosContextType | undefined>(undefined);
```

Cria o contexto que será usado na aplicação.

```
export function FavoritosProvider({ children }: Props) {...}
```

Componente que encapsula a aplicação para fornecer acesso ao contexto.

```
const [favoritos, setFavoritos] = useState<number[]>([]);
```

Estado local para armazenar os IDs dos filmes favoritos.

```
return (
    <FavoritosContext.Provider value={{ favoritos, toggleFavorito, isFavorito }}>
        {children}
    </FavoritosContext.Provider>
```



);

Disponibiliza os dados e funções para todos os componentes filhos.

Arquivo hooks/useFavoritos.ts

```
import { useContext } from 'react';
import { FavoritosContext } from '../context/FavoritosContext';

export const useFavoritos = () => {
  const context = useContext(FavoritosContext);
  if (!context) {
    throw new Error('useFavoritos precisa estar dentro de
FavoritosProvider');
  }
  return context;
};
```

A função useFavoritos simplifica o acesso ao contexto de favoritos, garantindo que ele só seja usado onde o contexto está disponível, evitando erros difíceis de debugar.

Criação dos componentes

Arquivo componentes/FilmeCard.tsx

```
import Link from 'next/link';
import styles from '../styles/FilmeCard.module.css';
import { useFavoritos } from '../hooks/useFavoritos';
```

- **Link**: componente do Next.js que permite navegação interna entre páginas, otimizada para SPA.
- **styles**: importação dos estilos CSS módulos para estilizar o componente.
- **useFavoritos**: hook personalizado para acessar o contexto dos filmes favoritos.

```
interface FilmeCardProps {
  id: number;
  title: string;
  poster_path: string | null;
}
```

Definimos as propriedades que o componente recebe:

- **id**: número que identifica o filme.
- **title**: título do filme.
- **poster_path**: caminho para a imagem do pôster, que pode ser null se não existir imagem disponível.

```
export default function FilmeCard({ id, title, poster_path }: FilmeCardProps)
{
  const { isFavorito, toggleFavorito } = useFavoritos();
  ...
}
```



}

- O componente recebe as props descritas.
- Usa o hook useFavoritos para acessar:
 - **isFavorito(id)**: função que verifica se o filme está na lista de favoritos.
 - **toggleFavorito(id)**: função que adiciona ou remove o filme dos favoritos.

```
return (
  <article className={styles.card}>
    <img
      src={poster_path ? `https://image.tmdb.org/t/p/w300${poster_path}` :
      '/images/placeholder.png'}
      alt={title}
      className={styles.poster}
    />
    <h2 className={styles.title}>{title}</h2>

    <footer className={styles.actions}>
      <Link href={`/filmes/${id}`}
        <button className={styles.button}>Ver detalhes</button>
      </Link>

      <button onClick={() => toggleFavorito(id)}
        className={styles.favoriteButton}>
        {isFavorito(id) ? '❤️' : '🤍'}
      </button>
    </footer>
  </article>
);
```

- **<article>**: container principal do cartão, com classe para estilização.
- ****:
 - Exibe o pôster do filme usando a URL base da API.
 - Se não houver poster_path, exibe uma imagem padrão (placeholder).
 - O atributo alt é o título do filme para acessibilidade.
- **<h2>**: mostra o título do filme.
- **<footer>**:
 - Botão "Ver detalhes" dentro do Link para navegar para a página específica do filme.
 - Botão para favoritar, que ao ser clicado chama toggleFavorito para adicionar/remover dos favoritos.
 - O ícone do botão muda dinamicamente:
 - Coração vermelho (❤️) se o filme já é favorito.
 - Coração branco (🤍) se não é favorito.

Arquivo componentes/DetalhesFilme.tsx

```
import Link from 'next/link';
import styles from '../styles/DetalhesFilme.module.css';
import { useFavoritos } from '../hooks/useFavoritos';
```



Imports parecidos com o do outro componente, com exceção do css.

```
interface Filme {  
    titulo: string;  
    descricao: string;  
    ano: number;  
    capa: string;  
    classificacao: string;  
    duracao: string;  
    genero: string;  
    nota: string;  
    id: number;  
}  
  
interface Props {  
    filme: Filme;  
}
```

- A interface Filme define as propriedades esperadas para o filme, com informações detalhadas.
- Props define que o componente recebe um objeto filme do tipo Filme.

```
export default function DetalhesFilme({ filme }: Props) {  
    const { isFavorito, toggleFavorito } = useFavoritos();  
    ...  
}
```

- Recebe a propriedade filme.
- Usa o hook useFavoritos para:
 - Verificar se o filme está nos favoritos (isFavorito).
 - Alternar o estado de favorito (toggleFavorito).

```
return (  
    <main className={styles.container}>  
        <button className={styles.favorito} onClick={() =>  
            toggleFavorito(filme.id)}>  
            {isFavorito(filme.id) ? '❤️' : '♥'}  
        </button>  
  
        <img  
            src={filme.capa}  
            alt={filme.titulo}  
            className={styles.capa}  
            onMouseEnter={e => (e.currentTarget.style.transform = 'scale(1.03)')}  
            onMouseLeave={e => (e.currentTarget.style.transform = 'scale(1)' )}  
        />  
        <h1 className={styles.titulo}>{filme.titulo}</h1>  
    </main>
```



```
<section className={styles.infoGrid}>
  <p>  <strong>Gênero:</strong> {filme.genero}</p>
  <p>  <strong>Duração:</strong> {filme.duracao}</p>
  <p>  <strong>Classificação:</strong> {filme.classificacao}</p>
  <p>  <strong>Ano:</strong> {filme.ano}</p>
  <p>  <strong>Nota:</strong> {filme.nota} / 10</p>
</section>

<p className={styles.descricao}>{filme.descricao}</p>

<Link href="/">
  <button
    className={styles.botaoVoltar}
    onMouseEnter={e => {
      e.currentTarget.style.backgroundColor = '#333';
      e.currentTarget.style.transform = 'scale(1.05)';
    }}
    onMouseLeave={e => {
      e.currentTarget.style.backgroundColor = '#4a4a4a';
      e.currentTarget.style.transform = 'scale(1)';
    }}
  >
    ← Voltar
  </button>
</Link>
</main>
);
```

- **Botão de favorito:**
 - Ícone muda entre coração cheio () e vazio () conforme o estado do filme nos favoritos.
 - Ao clicar, alterna o estado de favorito.
- **Imagen da capa:**
 - Mostra a imagem principal do filme.
 - Aplica efeito de zoom ao passar o mouse (scale(1.03)).
- **Título:**
 - Exibe o título do filme em destaque.
- **Seção de informações:**
 - Exibe gênero, duração, classificação indicativa, ano e nota do filme, com ícones para facilitar a leitura visual.
- **Descrição:**
 - Texto que apresenta a sinopse ou descrição do filme.
- **Botão Voltar:**
 - Link para voltar à página principal.
 - Ao passar o mouse, muda a cor e aumenta levemente o tamanho para efeito visual.

Criando o arquivo pages/_app.tsx



```
import type { AppProps } from 'next/app';
import { FavoritosProvider } from '../context/FavoritosContext';
```

- **AppProps:** tipo fornecido pelo Next.js que define as props padrão para o componente principal da aplicação.
- **FavoritosProvider:** componente que provê o contexto de favoritos para toda a aplicação.

```
export default function App({ Component, pageProps }: AppProps) {
  return (
    <FavoritosProvider>
      <Component {...pageProps} />
    </FavoritosProvider>
  );
}
```

- O componente App recebe:
 - **Component:** o componente da página que deve ser renderizado (exemplo: FilmeCard, DetalhesFilme ou qualquer outra página).
 - **pageProps:** propriedades específicas da página.
- Todo o conteúdo da aplicação é envolvido pelo FavoritosProvider, garantindo que o contexto de favoritos esteja disponível em qualquer componente dentro da árvore.
- Isso permite usar o hook useFavoritos em qualquer componente, sem precisar passar props manualmente.

Criando a página inicial (pages/index.tsx)

```
import { GetServerSideProps } from 'next';
import { getFilmesPopulares } from '../lib/api';
import FilmeCard from '../components/FilmeCard';
import { useFavoritos } from '../hooks/useFavoritos';
import styles from '../styles/Home.module.css';
```

- **GetServerSideProps:** tipo do Next.js para usar renderização do lado do servidor.
- **getFilmesPopulares:** função que busca filmes populares (provavelmente via API externa).
- **FilmeCard:** componente que mostra cada filme individualmente.
- **useFavoritos:** hook para acessar favoritos.
- **styles:** estilos para a página.

```
export const getServerSideProps: GetServerSideProps = async () => {
  const filmes = await getFilmesPopulares();
  return { props: { filmes } };
};
```

- Essa função roda no servidor a cada requisição à página.
- Busca a lista de filmes populares via API (getFilmesPopulares).
- Passa essa lista para o componente como propriedade filmes.
- Garante que o conteúdo está sempre atualizado.

```
export default function Home({ filmes }: { filmes: any[] }) {
  const { favoritos } = useFavoritos();
```



```
const filmesFavoritos = filmes.filter(filme =>
favoritos.includes(filme.id));
const filmesRestantes = filmes.filter(filme =>
!favoritos.includes(filme.id));
```

- Recebe a lista de filmes via props.
- Pega a lista de filmes favoritos do contexto com useFavoritos.
- Separa os filmes em dois grupos:
 - **filmesFavoritos**: os que estão na lista de favoritos.
 - **filmesRestantes**: os que não estão favoritados.

```
return (
  <main className={styles.container}>
    <h1 className={styles.title}>🎬 Catálogo de Filmes</h1>

    {filmesFavoritos.length > 0 && (
      <>
        <h2 className={styles.subtitle}>❤️ Filmes Favoritados</h2>
        <section className={styles.grid}>
          {filmesFavoritos.map(({ id, title, poster_path }) => (
            <FilmeCard key={id} id={id} title={title}
poster_path={poster_path} />
          )));
        </section>
      </>
    )}
  </main>
);
```

- Mostra um título geral.
- Se houver filmes favoritados:
 - Exibe uma seção separada com eles.
 - Usa o componente FilmeCard para renderizar cada filme.
- Depois, exibe todos os filmes populares que não estão favoritados em outra seção.
- As duas listas são renderizadas em grades (styles.grid).

Criando a página de detalhes do filme (pages/[id].tsx)

```
import { GetServerSideProps } from 'next';
import DetalhesFilme from '../../components/DetalhesFilme';
import { getFilmePorId, Filme } from '../../lib/api';
```



- **GetServerSideProps**: permite renderizar a página no servidor a cada requisição.
- **DetalhesFilme**: componente que exibe as informações completas do filme.
- **getFilmePorId**: função que busca os dados do filme pela API usando o id.
- **Filme**: tipo/interface que representa a estrutura dos dados do filme.

```
interface Props {  
  filme?: Filme;  
}
```

- Define o tipo das propriedades que o componente da página irá receber.
- O filme é opcional, pois pode não ser encontrado.

```
export default function PaginaDetalhes({ filme }: Props) {  
  if (!filme) return <p style={{ textAlign: 'center', marginTop: '2rem' }}>Filme não encontrado.</p>;  
  
  return <DetalhesFilme filme={filme} />;  
}
```

Verifica se o filme foi encontrado:

- Se não, exibe uma mensagem simples no centro da página: “Filme não encontrado.”
- Se sim, renderiza o componente DetalhesFilme passando o objeto filme como prop.

```
export const getServerSideProps: GetServerSideProps = async (context) => {  
  const { id } = context.params!;  
  
  const filme = await getFilmePorId(id as string);  
  
  return {  
    props: { filme },  
  };  
};
```

- Essa função roda no servidor a cada requisição.
- Extrai o parâmetro id da URL via context.params.
- Busca os dados do filme usando a função getFilmePorId com o id.
- Passa os dados do filme para o componente via props.
- Caso não encontre o filme, filme será undefined, o que ativa a mensagem de erro no componente.

Criação dos arquivos CSS

Arquivo styles/Home.module.css

```
.container {  
  padding: 2rem;  
  font-family: 'Segoe UI', sans-serif;  
  background-color: #f5f5f5;  
  min-height: 100vh;  
}
```



- **padding: 2rem** — Espaço interno em todas as direções, para afastar o conteúdo das bordas da página.
- **font-family: 'Segoe UI', sans-serif** — Define a fonte da página, com fallback para fontes sans-serif.
- **background-color: #f5f5f5** — Cor de fundo clara, um cinza muito suave.
- **min-height: 100vh** — Garante que o container tenha ao menos a altura total da tela, para preencher o viewport verticalmente.

```
.title {  
    font-size: 2.5rem;  
    text-align: center;  
    color: #333;  
    margin-bottom: 2rem;  
}
```

- **font-size: 2.5rem** — Texto grande para destaque do título principal.
- **text-align: center** — Centraliza o texto horizontalmente.
- **color: #333** — Cor do texto em um cinza escuro, para boa legibilidade.
- **margin-bottom: 2rem** — Espaço abaixo do título para separar dos próximos elementos.

```
.grid {  
    display: grid;  
    grid-template-columns: repeat(4, 1fr);  
    gap: 1.5rem;  
}
```

- **display: grid** — Ativa o layout em grade para os elementos filhos.
- **grid-template-columns: repeat(4, 1fr)** — Define 4 colunas de tamanhos iguais, cada uma ocupando uma fração igual do espaço disponível.
- **gap: 1.5rem** — Espaçamento uniforme entre linhas e colunas da grade.

Arquivo styles/FilmeCard.module.css

```
.card {  
    border: 1px solid #ddd;  
    border-radius: 12px;  
    box-shadow: 0 2px 8px rgba(0,0,0,0.1);  
    background: #fff;  
    overflow: hidden;  
    text-align: center;  
    padding: 1rem;  
    transition: transform 0.2s;  
    cursor: pointer;  
}
```

- **Borda e arredondamento:** `1px solid #ddd` cria uma borda cinza clara; `border-radius: 12px` arredonda os cantos do card.
- **Sombra:** `box-shadow` adiciona sombra suave para dar profundidade.
- **Fundo branco:** `background: #fff` para destacar o card.
- **Overflow oculto:** evita que conteúdo ultrapasse os limites arredondados.
- **Texto centralizado:** `text-align: center`.



- **Padding:** espaçamento interno confortável.
- **Transição:** transition: transform 0.2s para animações suaves no hover.
- **Cursor pointer:** indica que o card é interativo (aparece a mãozinha ao passar o mouse).

```
.card:hover {  
    transform: scale(1.02);  
}
```

- Ao passar o mouse sobre o card, ele aumenta ligeiramente de tamanho (scale(1.02)), dando um efeito de "zoom" sutil.

```
.poster {  
    width: 100%;  
    border-radius: 8px;  
}  
  
.title {  
    margin-top: 0.5rem;  
}
```

- A imagem do poster ocupa toda a largura do card.
- Cantos arredondados (8px) para suavizar a aparência.
- Pequeno espaçamento acima do título para separar da imagem.

```
.button {  
    background-color: #0070f3;  
    color: #fff;  
    border: none;  
    padding: 0.5rem 1rem;  
    border-radius: 8px;  
    cursor: pointer;  
    margin-top: 0.5rem;  
    font-weight: bold;  
    transition: background-color 0.3s;  
}  
  
.button:hover {  
    background-color: #0059c1;  
}
```

- Botão com fundo azul vibrante (#0070f3) e texto branco.
- Sem bordas visíveis.
- Espaçamento interno confortável.
- Cantos arredondados para estilo moderno.
- Cursor pointer para indicar clicável.
- Espaço acima para separar do título.
- Texto em negrito.
- Transição suave para mudança de cor no hover.
- Quando o botão é focado pelo mouse, a cor de fundo muda para um azul mais escuro, dando feedback visual.



```
.actions {  
  display: flex;  
  gap: 10px;  
  align-items: center;  
  justify-content: center;  
}
```

- Container para os botões do card (ver detalhes e favoritar).
- Layout flexível horizontal.
- Espaçamento de 10px entre os botões.
- Alinhamento vertical centralizado.
- Centraliza os botões horizontalmente dentro do card.

```
.favoriteButton {  
  background: none;  
  border: none;  
  cursor: pointer;  
  font-size: 1.5rem;  
  transition: transform 0.2s;  
}  
  
.favoriteButton:hover {  
  transform: scale(1.2);  
}
```

- Botão de favoritar sem fundo nem borda para parecer um ícone limpo.
- Cursor pointer para indicar clicável.
- Fonte maior para destaque (ícones de coração).
- Transição para efeito visual suave.
- Ao passar o mouse, o botão aumenta de tamanho (20%) para dar feedback visual e indicar que está ativo.

Arquivo styles/DetalhesFilme.module.css

```
.container {  
  padding: 2rem;  
  font-family: 'Segoe UI', sans-serif;  
  background-color: #f0f2f5;  
  min-height: 100vh;  
  max-width: 700px;  
  margin: 0 auto;  
  text-align: center;  
}
```

- **Padding:** Espaçamento interno confortável ao redor do conteúdo.
- **Fonte:** 'Segoe UI', uma fonte limpa e moderna.
- **Fundo:** Cinza claro (#f0f2f5) para suavizar o visual.
- Altura mínima: 100vh garante que a página ocupe toda a altura da tela.
- **Largura máxima:** Limita o conteúdo para 700px, para melhor leitura em telas grandes.



- **Centralização:** margin: 0 auto centraliza o container horizontalmente.
- **Texto centralizado:** Alinha o texto ao centro da página, ideal para títulos e imagens.

```
.capa {  
    width: 100%;  
    max-width: 400px;  
    border-radius: 10px;  
    margin-bottom: 1.5rem;  
    box-shadow: 0 6px 12px rgba(0, 0, 0, 0.2);  
    transition: transform 0.3s;  
}
```

- A imagem da capa do filme ocupa 100% da largura do container, até no máximo 400px.
- Cantos arredondados para suavidade visual.
- Espaço abaixo da imagem para separar do título e informações.
- Sombra forte para destacar a imagem.
- Transição suave para animações (caso queira adicionar algum efeito).

```
.titulo {  
    color: #222;  
    margin-bottom: 1rem;  
    text-shadow: 1px 1px 2px rgba(0, 0, 0, 0.1);  
}
```

- Cor do texto é quase preta para boa leitura.
- Espaço abaixo para separar do próximo conteúdo.
- Sombra discreta no texto para dar profundidade e suavizar.

```
.infoGrid {  
    display: grid;  
    gap: 0.6rem;  
    font-size: 1rem;  
    line-height: 1.6;  
    margin-bottom: 1.5rem;  
    text-align: left;  
    max-width: 500px;  
    margin-left: auto;  
    margin-right: auto;  
}
```

- Usa grid CSS para organizar as informações do filme (diretor, gênero, duração etc.).
- Espaço vertical entre linhas para facilitar leitura.
- Tamanho de fonte padrão, com espaçamento entre linhas confortável.
- Alinha o texto à esquerda para melhor leitura.
- Centraliza horizontalmente o bloco de informações, limitando a largura em 500px para evitar linhas muito longas.

```
.descricao {  
    font-style: italic;  
    margin-bottom: 2rem;
```



```
padding: 0 1rem;  
}
```

- O texto da descrição fica em itálico, destacando-se do restante.
- Espaço grande abaixo para separar do botão.
- Padding horizontal para que o texto não encoste nas bordas.

```
.botaoVoltar {  
background-color: #4a4a4a;  
color: #fff;  
border: none;  
padding: 0.8rem 1.6rem;  
border-radius: 8px;  
cursor: pointer;  
font-weight: bold;  
font-size: 1rem;  
transition: background-color 0.3s, transform 0.2s;  
}
```

- Botão escuro com texto branco para contraste forte.
- Sem borda para visual moderno.
- Espaçamento interno confortável para clique fácil.
- Cantos arredondados para suavidade visual.
- Cursor pointer para indicar clicável.
- Texto em negrito e tamanho legível.
- Transições para mudança suave de cor e possíveis animações de escala.

```
.favorito {  
background: none;  
border: none;  
padding: 0;  
margin: 0;  
cursor: pointer;  
font-size: 2rem;  
position: absolute;  
top: 1rem;  
right: 1rem;  
transition: transform 0.2s;  
}  
  
.favorito:hover {  
transform: scale(1.2);  
}
```

- Botão do ícone de favorito sem fundo nem borda, para parecer um ícone flutuante.
- Cursor pointer para indicar interatividade.
- Tamanho grande do ícone para destaque visual.
- Posicionado no canto superior direito da página (posição absoluta).
- Transição para animação suave.



- Ao passar o mouse sobre o botão favorito, ele aumenta 20% de tamanho, dando feedback visual para o usuário.

Considerações finais

Este projeto representou uma oportunidade valiosa para aplicar conhecimentos práticos em desenvolvimento web utilizando Next.js, TypeScript e consumo de APIs externas. A criação do catálogo de filmes não só permitiu a integração com a API do The Movie Database, mas também o aprofundamento no uso de recursos avançados do Next.js, como a renderização do lado servidor (SSR) para melhorar a experiência do usuário e SEO.

O gerenciamento de favoritos, implementado via Context API e hooks customizados, mostrou-se fundamental para proporcionar uma interação dinâmica e personalizada para os usuários, mantendo o estado global de forma simples e eficaz. Além disso, a organização modular do código, separando lógica de negócio, componentes e estilos via CSS Modules, facilitou a manutenção e escalabilidade da aplicação.

Durante o desenvolvimento, a preocupação com a tipagem em TypeScript auxiliou na detecção precoce de erros, garantindo maior segurança e qualidade no código. A experiência com as requisições assíncronas e tratamento dos dados da API reforçou a importância de lidar adequadamente com respostas e erros em aplicações reais.

Por fim, este trabalho não apenas reforçou conceitos técnicos, mas também destacou a importância do planejamento e organização no desenvolvimento de aplicações web modernas, desde a configuração inicial até a entrega de funcionalidades relevantes e intuitivas para o usuário final.