**Tecnologias para a Web e Ambientes Móveis**

**Projeto da Disciplina / Fase de Implementação**

**Ano Letivo: 2022-2023 / 2º semestre / 2º Ano**

**Miguel Marmelete /20431**

1. **Introdução**

Neste relatório, será apresentado o enquadramento geral do projeto, bem como a identificação das quatro tarefas que foram analisadas e desenhadas na fase anterior.

O objetivo geral do projeto é desenvolver uma aplicação que gerencie uma lista de filmes, permitindo ao utilizador adicionar filmes à sua *watchlist*, remover filmes dessa lista, procurar informações sobre filmes específicos e atribuir pontuações aos filmes assistidos. Para alcançar esse objetivo, foi utilizada a API TMDB (*The Movie Database*).

A TMDB API é uma interface de programação de aplicativos que fornece acesso a um vasto conjunto de dados relacionados a filmes, incluindo informações detalhadas sobre filmes específicos, como títulos, sinopses, elenco, equipes de produção, datas de lançamento e avaliações de utilizadores.

**Tarefas Analisadas e Desenhadas:**

**Remover um filme da *watchlist***: Essa tarefa consiste em permitir que o utilizador remova um filme específico de sua *watchlist*. Será implementada uma funcionalidade na aplicação que exibirá a lista de filmes presentes na *watchlist* e permitirá ao utilizador selecionar um filme para ser removido.

**Adicionar um filme à *watchlist***: Nessa tarefa, o utilizador poderá adicionar filmes à sua *watchlist*. Será implementada uma função na aplicação que permitirá ao utilizador pesquisar um filme por título e, em seguida, exibir os resultados correspondentes. O utilizador poderá selecionar um filme dos resultados e adicioná-lo à sua *watchlist*.

**Procurar um filme**: Essa tarefa envolve permitir que o utilizador procure informações sobre um filme específico. Será implementada uma funcionalidade na aplicação que permitirá ao utilizador pesquisar um filme por título e exibir os detalhes relevantes do filme encontrado, como sinopse, elenco e avaliações.

**Pontuar um filme**: Nessa tarefa, o utilizador poderá atribuir uma pontuação a um filme que tenha assistido. Será implementada uma funcionalidade na aplicação que permitirá ao utilizador selecionar um filme da *watchlist* ou procurar um filme específico e, em seguida, atribuir uma pontuação a ele.

Das quatro tarefas analisadas e desenhadas, todas as 4 foram implementadas.

A implementação dessas tarefas será baseada nos recursos e dados fornecidos pela API TMDB. Os detalhes dos filmes, como títulos, sinopses, elenco e avaliações, serão obtidos por meio das chamadas adequadas à API e exibidos na interface da aplicação para o utilizador.

1. **Decisões Globais de Implementação**

Durante o desenvolvimento do sistema web, foram tomadas várias decisões importantes em relação às tecnologias utilizadas, à estética e aparência das interfaces gráficas, à responsividade em diferentes dispositivos e às considerações de acessibilidade para pessoas com necessidades especiais.

**Tecnologias de desenvolvimento**:

Para a implementação da aplicação *front-end*, a tecnologia escolhida foi a biblioteca *React*. O *React* é uma biblioteca JavaScript de código aberto que permite a criação de interfaces de utilizador interativas. A sua flexibilidade, eficiência e grande comunidade de suporte foram os principais motivos para sua escolha. A versão específica do *React* utilizada no projeto foi a mais recente disponível durante a implementação.

**Estética e aparência**:

No que diz respeito à estética e aparência das interfaces gráficas, não foram utilizadas tecnologias específicas além do próprio *React*. No entanto, outras bibliotecas e *frameworks* populares, como *Bootstrap*, poderiam ter sido empregues para facilitar o desenvolvimento de estilos e layouts consistentes. O *Bootstrap* é uma biblioteca CSS e JavaScript que oferece componentes e estilos pré-definidos, permitindo uma aparência mais profissional e responsiva às interfaces.

**Responsividade em diferentes dispositivos**:

Para garantir a responsividade da aplicação em diferentes dimensões e resoluções de dispositivos, foram tomadas decisões de implementação no CSS dos componentes. Ao definir as propriedades de largura (*width*) dos elementos, utilizou-se unidades de medida relativas, como percentagem (%), em vez de unidades absolutas, como pixéis (px). Isso permite que os componentes se ajustem automaticamente ao tamanho do ecrã, proporcionando uma experiência de utilizador consistente em dispositivos de diferentes tamanhos.

**Acessibilidade para pessoas com necessidades especiais**:

Considerações de acessibilidade foram aplicadas no projeto, principalmente no que diz respeito à interação com os elementos da interface. Ao passar o rato sobre um botão ou o poster de um filme, foram implementadas mudanças visuais, como alteração de cor, para fornecer feedback visual aos utilizadores com dificuldades de perceção. Essa abordagem visa tornar a aplicação mais acessível para pessoas com necessidades especiais, garantindo que as informações e funcionalidades possam ser compreendidas e utilizadas da melhor forma possível.

1. **Decisões de Implementação Específicas**

Em seguida são explicadas as principais decisões de implementação para a funcionalidade que suporta cada um dos seguintes elementos:

* + 1. **Homepage**

Em seguida é apresentada uma imagem da Página inicial. A página inicial contém dois componentes principais, uma “*navbar*” e um container onde são guardados os filmes. Dentro do container dos filmes, existem inúmeros componentes do tipo “*MovieCard*” e cada um representa um filme.

**Uma imagem com texto, Cara humana, pessoa, mulher

Descrição gerada automaticamente**

**Figura 1 - Homepage**

**Principais decisões de implementação:**

**Javascript**

Para poder utilizar a aplicação web, foi preciso criar um mecanismo de autenticação pois para utilizar os POSTS, além da API\_KEY é necessário um SESSION\_ID. Para tal, é necessário utilizar a API\_KEY para pedir um REQUEST\_TOKEN e depois com o REQUEST\_TOKEN pedir o SESSION\_ID.

A seguir é apresentado o código que trata a autenticação:

async function autentication() {

  const queryString = new URLSearchParams(window.location.search);

  const approved = queryString.get("approved");

  const request\_token = queryString.get("request\_token");

  console.log(approved);

  console.log(request\_token);

  localStorage.setItem("moderator\_token", "moderator");

  localStorage.setItem("moderator", "false");

  console.log("approved and request token");

  if (approved && request\_token) {

    console.log("approved and request token");

    const { session\_id } = await fetch(

      `https://api.themoviedb.org/3/authentication/session/new?api\_key=${urls.API\_KEY}`,

      {

        method: "POST",

        headers: {

          "Content-Type": "application/json",

        },

        body: JSON.stringify({

          request\_token: request\_token,

        }),

      }

    ).then((res) => res.json());

    localStorage.setItem("session\_id", session\_id);

    window.location = `${urls.CALLBACK}`;

    console.log("session id");

  } else if (localStorage.getItem("session\_id") == null) {

    // Initiates an authentication flow

    // clear session information

    localStorage.removeItem("session\_id");

    const { request\_token } = await fetch(

      `https://api.themoviedb.org/3/authentication/token/new?api\_key=${urls.API\_KEY}`

    ).then((r) => r.json());

    // redirect user

    window.location = `https://www.themoviedb.org/authenticate/${request\_token}?redirect\_to=${urls.CALLBACK}`;

      }}

Resumindo o funcionamento, usando a *queryString* o sistema procura os valores do *approved* e do *request\_token*, caso o *approved* seja *true* e o *request\_token* não seja **null**, irá executar um *fetch* para obter o session\_id. Caso contrário, o sistema vai pedir um *request\_token*. Após receber *o request\_token* o site redireciona para a parte de autenticação do TMDB para obter o *session\_id*.

Outra decisão importante de implementação na homepage foi o que exibir quando o utilizador entra do site. Neste caso decidi exibir os filmes com melhor pontuação. Em seguida mostro o excerto de código que obtém os filmes:

 const [movies, setMovies] = useState([]);

  const getTopRatedMovies = async (url) => {

    const res = await fetch(url);

    const data = await res.json();

    console.log(data);

    setMovies(data.results);

  };

  const movieCards = movies.map((movie) => {

    console.log(movie.id);

    return (

      <MovieCard key={movie.id} movieId={movie.id} setChange={setChange} />

    );

  });

Resumindo, faço um fetch usando o url disponibilizado pelo TMDB que me permite obter os filmes com melhor pontuação e guardo no useState usando a função set. Com os filmes no useState, é feito um map que cria vários “MovieCard” com os dados do filme.

**HTML**

Como referido, para a *HomePage*, foram criados dois componentes principais. A seguir encontra-se o código HTML dos dois.

***Navbar***:

<div className="navbar\_container">

      <div className="logo">

        <Link to={`/`} style={{ color: "#000", textDecoration: "none" }}>

          Movie

          <br />

          List

        </Link>

      </div>

      <div className="navbar\_search">

        <input

          type="text"

          placeholder="Search Movie"

          className="navbar\_search-input"

          value={inputValue}

          onChange={(e) => setInputValue(e.target.value)}

        ></input>

        <button className="navbar\_search-button" onClick={handleSearch}>

          Search

        </button>

      </div>

      <div>

        <button className="navbar\_rate-button" onClick={handlePermissions}>

          Mod

        </button>

        <button className="watchlist\_button">

          <Link

            to="/watchlist"

            style={{ color: "#FFF", textDecoration: "none" }}

          >

            Watchlist

          </Link>

        </button>

      </div>

    </div>

A *navbar* é composta pelos seguintes elementos: Um texto que representa o logo do site, uma barra de pesquisa, um botão acionar a pesquisa e dois botões, um “*mod*” e outro “*watchlist*”

Container dos filmes:

<div className="movies\_container">{movieCards}</div>

Cada *MovieCard* apresenta o seguinte HTML:

<div className="movie\_card">

      <Link to={{ pathname: `/movie/${movieId}` }}>

        <img

          className="movie\_card\_image"

          src={`https://image.tmdb.org/t/p/w400${movie.poster\_path}`}

          alt={movie.title}

        />

      </Link>

      <div className="movie\_card\_info">

        <h2 className="movie\_title">{movie.title}</h2>

        <p className="rating">

          <FaStar /> {movie.vote\_average}

        </p>

        <div>

          <button className="add\_remove\_button" onClick={handleAddRemoveClick}>

            {isItemInList ? "Remove" : "Add"}

          </button>

          <button className="add\_remove\_button" onClick={handleRate}>

            Rate

          </button>

        </div>

      </div>

    </div>

**CSS**

Para o CSS as principais decisões tomadas foram o alinhamento dos itens da navbar e do container dos filmes. A seguir apresento os excertos do código pela ordem referida:

.navbar\_container {

  display: flex;

  background-color: rgb(59, 156, 197);

  justify-content: space-between;

  height: 80px;

  align-items: center;

  font-weight: bold;

  width: auto;

  overflow: hidden;

}

.logo {

  margin: 10px 20px;

  width: auto;

  font-size: x-large;

}

.navbar\_search-input {

  width: auto;

  height: 30px;

}

.navbar\_search-button {

  height: 35px;

}

.watchlist\_button,

.navbar\_rate-button {

  margin-right: 10px;

  width: auto;

  height: 40%;

  font-size: larger;

  display: inline-block;

  padding: 5px 10px;

  font-size: 16px;

  border: none;

  border-radius: 4px;

  background-color: #000000;

  color: #fff;

  text-align: center;

  cursor: pointer;

  transition: background-color 0.3s ease;

}

.watchlist\_button:hover,

.navbar\_rate-button:hover {

  background-color: #6e7777;

}

**Acessibilidade**

Para a acessibilidade foi implementado no CSS o “*hover*”, ao passar por cima dos botões ou do poster do filme, a cor o mesmo muda. Esta implementação facilita a localização do cursor.

* + 1. **Tarefa 1 (pesquisar filme)**

Abaixo podemos observar o resultado da tarefa 1 (procurar filme). Apesar de não mostrar na barra “*search*”, uma vez que ao pesquisar, a barra fica vazia, pesquisei por “inter” na barra de pesquisa. Os filmes apresentados são as correspondências da pesquisa. Como demonstrado, não é preciso colocar o nome completo do filme para fazer uma pesquisa.

Uma imagem com texto, captura de ecrã, Software de multimédia, pessoa

Descrição gerada automaticamente

**Figura 2 - Procurar Filme**

**Principais decisões de implementação:**

**Javascript**

Para fazer a pesquisa de um filme é necessário colocar o nome na barra de pesquisa e carregar no botão “*search*”. Em seguida é possível observar o código javascript:

const queryString = new URLSearchParams(window.location.search);

  const [searchParams, setSearchParams] = useSearchParams();

  const [inputValue, setInputValue] = useState("");

  const handleSearch = () => {

    setSearchParams({ movieName: inputValue });

    queryString.set("movieName", searchParams);

    setInputValue("");

  };

Cada vez que o utilizador adiciona um caracter á barra de *search*, o valor do input é atualizado. Ao carregar no botão *search*, a função *handleSearch* vai executar, colocando na *queryString* o nome do filme.

Depois de alterar a *queryString*, essa alteração é detetada pelo *useEffect* no container dos filmes. Ao detetar a mudança, é feito um *fetch* com base no nome do input do utilizador.

useEffect(() => {

    if (movieName) {

      async function searchedMovies() {

        const getMovie = await getMovieByName(movieName);

        setMovies(getMovie);

      }

      searchedMovies();

    } else {

      if (movieName) {

      }

      getTopRatedMovies(urls.TOP\_RATED\_MOVIES\_URL);

    }

  }, [location.search]);

A seguir apresento o código do *fetch*:

async function getMovieByName(movieName) {

  const options = {

    method: "GET",

    headers: {

      accept: "application/json",

      "content-type": "application/json",

      Authorization:

        "Bearer eyJhbGciOiJIUzI1NiJ9.eyJhdWQiOiI5ZmZhZjAxZDhlNDNkMDFlMmJhY2QxNjdjYmJlNTVjYyIsInN1YiI6IjY0MDYxYTczMzgzZGYyMDBhOTU1ZTZhZCIsInNjb3BlcyI6WyJhcGlfcmVhZCJdLCJ2ZXJzaW9uIjoxfQ.go6AtD8ijivF0367MCqzWRUeIp4nnPeJJVKKg8qJY4s",

    },

  };

  const response = await fetch(

    `https://api.themoviedb.org/3/search/movie?query=${movieName}&api\_key=${urls.API\_KEY}`,

    options

  );

  const movieData = await response.json();

  return movieData.results;

}

**HTML**

O HTML usado é o mesmo usado anteriormente, pelo “*MovieCard*”, o *container* e a *navbar*.

**CSS**

Os elementos presentes na execução desta tarefa utilizam o mesmo CSS já referido anteriormente

* + 1. **Tarefa 2 (adicionar filme á watchlist)**

Abaixo podemos observar a *watchlist*, depois de adicionar 3 filmes. Os componentes usados são os mesmos referidos anteriormente

Uma imagem com texto, captura de ecrã, a esquiar, design gráfico

Descrição gerada automaticamente

**Figura 3 - Watchlist**

**Javascript**

A seguir é apresentado o troço de código que implementa esta funcionalidade.

async function addMovie({ movieId }) {

  const list\_id = localStorage.getItem("list\_id");

  const session\_id = localStorage.getItem("session\_id");

  const success\_status\_code = 12;

  const options = {

    method: "POST",

    headers: {

      accept: "application/json",

      "content-type": "application/json",

      Authorization:

        "Bearer eyJhbGciOiJIUzI1NiJ9.eyJhdWQiOiI5ZmZhZjAxZDhlNDNkMDFlMmJhY2QxNjdjYmJlNTVjYyIsInN1YiI6IjY0MDYxYTczMzgzZGYyMDBhOTU1ZTZhZCIsInNjb3BlcyI6WyJhcGlfcmVhZCJdLCJ2ZXJzaW9uIjoxfQ.go6AtD8ijivF0367MCqzWRUeIp4nnPeJJVKKg8qJY4s",

    },

    body: JSON.stringify({ media\_id: movieId }),

  };

  const response = await fetch(

    `https://api.themoviedb.org/3/list/${list\_id}/add\_item?session\_id=${session\_id}`,

    options

  );

  const movieData = await response.json();

  if (movieData.status\_code === success\_status\_code) {

    alert("Movie added to watchlist");

  } else {

    alert("Could not add movie to watchlist, try again");

  }

}

Resumindo, a função recebe o ID do filme que o utilizador pretende adicionar. O site vai buscar o *session\_id* e o id da lista, para poder adicionar o filme que o utilizador escolheu. Após a operação, é verificado se o “*status\_code*” é o código de sucesso, se sim, o utilizador é notificado que o filme foi adicionado, caso contrário é notificado que não foi possível adicionar o filme á lista.

Este código vai executar quando o utilizador pressionar no botão “*add*” de um filme.

**HTML**

O HTML usado nesta tarefa já foi referido anteriormente.

**CSS**

O CSS usado nesta tarefa já foi referido anteriormente.

* + 1. **Tarefa 3 (remover filme da watchlist)**

Quando o utilizador pressiona o botão “remove”, aparece um *prompt* que pede a confirmação do utilizador.

Uma imagem com texto, a esquiar, captura de ecrã, design gráfico

Descrição gerada automaticamente

**Figura 4 - Remover filme**

**Javascript**

A implementação desta tarefa é semelhante á de adicionar os filmes á *watchlist*. Em seguida é mostrado o código.

async function removeItem({ movieId }) {

  const list\_id = localStorage.getItem("list\_id");

  const session\_id = localStorage.getItem("session\_id");

  const success\_status\_code = 13;

  const options = {

    method: "POST",

    headers: {

      accept: "application/json",

      "content-type": "application/json",

      Authorization:

        "Bearer eyJhbGciOiJIUzI1NiJ9.eyJhdWQiOiI5ZmZhZjAxZDhlNDNkMDFlMmJhY2QxNjdjYmJlNTVjYyIsInN1YiI6IjY0MDYxYTczMzgzZGYyMDBhOTU1ZTZhZCIsInNjb3BlcyI6WyJhcGlfcmVhZCJdLCJ2ZXJzaW9uIjoxfQ.go6AtD8ijivF0367MCqzWRUeIp4nnPeJJVKKg8qJY4s",

    },

    body: JSON.stringify({ media\_id: movieId }),

  };

  const response = await fetch(

    `https://api.themoviedb.org/3/list/${list\_id}/remove\_item?session\_id=${session\_id}`,

    options

  );

  const data = await response.json();

  if (data.status\_code === success\_status\_code) {

    alert("Movie removed from watchlist");

  } else {

    alert("Could not add movie to watchlist, try again");

  }

}

Para perceber melhor como o website sabe quando adicionar ou remover o filme:

const listId = localStorage.getItem("list\_id");

const [isItemInList, setIsItemInList] = useState(false);

  useEffect(() => {

    // Check if the item is in the list whenever the component is rendered

    const checkItemInList = async () => {

      const itemInList = await checkIfItemInList(movieId, listId);

      setIsItemInList(itemInList);

    };

    checkItemInList();

  }, [isItemInList]); // Run this effect whenever movieId or listId changes

  const handleAddRemoveClick = async () => {

    if (isItemInList) {

      const confirmed = window.confirm(

        "Are you sure you want to remove this movie?"

      );

      if (confirmed) {

        await removeItem({ movieId });

        setChange(true);

        setIsItemInList(false);

      }

    } else {

      await addMovie({ movieId });

      setIsItemInList(true);

    }

  };

Quando o botão “*add*”/”*remove*” é premido, é feita a verificação se o filme pertence á lista ou não. Se pertencer é removido caso contrário é adicionado. O código que verifica se o item pertence á lista é o seguinte:

async function checkIfItemInList(movieId, listId) {

  const options = { method: "GET", headers: { accept: "application/json" } };

  const response = await fetch(

    `https://api.themoviedb.org/3/list/${listId}/item\_status?language=en-US&movie\_id=${movieId}&api\_key=${urls.API\_KEY}`,

    options

  );

  const data = await response.json();

  return data.item\_present;

}

**HTML**

O HTML usado nesta tarefa já foi referido anteriormente.

**CSS**

O CSS usado nesta tarefa já foi referido anteriormente.

* + 1. **Tarefa 4 (avaliar filme)**

Para poder utilizar esta funcionalidade, o utilizador precisa ser um moderador. Para tal, precisa de selecionar o botão “*mod*” e introduzir o *token* correto.

Em seguida apresento a sequencia normal que um utilizador com o *token* poderá executar:

Primeiro o utilizador pressiona o botão “rate” sem ser moderador e por isso, é notificado pelo site.

Uma imagem com texto, captura de ecrã, desenho, design gráfico

Descrição gerada automaticamente

**Figura 5 - Notificação de moderador**

Em seguida, o utilizador pressiona o botão “*mod*” e é lhe pedido um input

Uma imagem com texto, captura de ecrã, design gráfico, desenho

Descrição gerada automaticamente

**Figura 6 - Pedido do token de moderador**

Se o utilizador colocar o *token* correto, é notificado com a mensagem “*You are now a moderator*”, caso contrário, é notificado com a mensagem “*wrong token, try again*”.

Uma imagem com texto, captura de ecrã, desenho, design gráfico

Descrição gerada automaticamente

**Figura 7 - Mensagem de sucesso ou insucesso**

Após se tornar moderador, o utilizador pode avaliar os filmes

Uma imagem com texto, captura de ecrã, design gráfico

Descrição gerada automaticamente

**Figura 8 - Avaliar filme**

**Javascript**

A seguir é mostrado o script que trata da avaliação do filme

function rateMovie(rate, movieId) {

  const session\_id = localStorage.getItem("session\_id");

  const options = {

    method: "POST",

    headers: {

      accept: "application/json",

      "Content-Type": "application/json;charset=utf-8",

    },

    body: `{"value":${rate}}`,

  };

  fetch(

    `https://api.themoviedb.org/3/movie/${movieId}/rating?session\_id=${session\_id}&api\_key=${urls.API\_KEY}`,

    options

  )

    .then((response) => response.json())

    .then((response) => {

      console.log(response);

      if (response.success) {

        alert("Success");

      } else {

        alert("Something went wrong");

      }

    })

    .catch((err) => console.error(err));

}

Este código é executado quando o botão rate é selecionado.

1. **Avaliação das Interfaces / Testes com Utilizadores**

Em seguida são explicadas as metodologias e os procedimentos executados nos testes com utilizadores que permitiram avaliar o grau de usabilidade do sistema Web, nas versões desktop e mobile

Para a execução dos testes o local escolhido foi a casa do desenvolvedor uma vez que é um local sossegado e com os recursos necessários á realização dos testes. Para a realização dos testes, foi pedido aos utilizadores que realizassem as tarefas que iam sendo pedidas. Os dados analisados foram o tempo de execução de cada tarefa.

Os testes foram realizados com 3 utilizadores. Para a execução dos testes, como software foi utilizado o OBS, o Visual Studio Code e o Brave. O primeiro foi usado para gravar a voz e o ecrã, o segundo para iniciar o programa e o terceiro corre o programa e permite utilizar a aplicação. Em relação ao hardware, foi utilizado um computador portátil, rato e teclado.

**Resultados**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | T1 | T2 | T3 | T4 |
| Utilizador 1 | 13,97 | 13,96 | 12,46 | 35,10 |
| Utilizador 2 | 13,43 | 14,83 | 9,28 | 27,31 |
| Utilizador 3 | 15,59 | 3,4 | 5,87 | 25,64 |

Após analisar os resultados, é possível concluir que as primeiras três tarefas são fáceis de realizar. No entanto, a terceira tarefa, foi mais difícil de executar para os utilizadores, não só por ter mais passos, mas notou-se durante a execução ds testes que os utilizadores por vezes não sabiam como se tornarem moderadores.

Links para os testes:

<https://drive.google.com/file/d/1ToOJGe0hoIRSpo3OFRPEtrfHe_CsITzY/view?usp=sharing>

<https://drive.google.com/file/d/17PYbCxUqkzScoExSjURzjtDLS84I6ngq/view?usp=sharing>

<https://drive.google.com/file/d/1lix5C8WvJt2khu3SohS6UIcsrlEmwJk_/view?usp=sharing>

1. **Conclusões Finais**

Este relatório destaca o desenvolvimento de um trabalho front-end que utilizou a TMDB API como fonte de dados. A aplicação web resultante apresentou diversas funcionalidades úteis para os usuários, incluindo a pesquisa de filmes, a adição de filmes a uma watchlist, a remoção de filmes da watchlist e a possibilidade de avaliar filmes.

A integração bem-sucedida com a TMDB API permitiu que os usuários acessassem informações atualizadas sobre uma vasta coleção de filmes. A capacidade de pesquisar filmes facilitou a localização de títulos específicos com base em critérios como nome. Além disso, a inclusão de uma watchlist ofereceu aos usuários a conveniência de salvar filmes interessantes para assistir mais tarde.

A funcionalidade de adicionar e remover filmes da watchlist garantiu uma experiência personalizada, permitindo que os usuários organizassem suas preferências e mantivessem um registo de filmes desejados. A capacidade de avaliar filmes adicionou uma dimensão interativa à aplicação, permitindo que os usuários compartilhassem suas opiniões sobre filmes específicos e fornecessem feedback útil para outros usuários.

No geral, este trabalho front-end demonstrou a aplicação prática de habilidades de desenvolvimento web, aproveitando uma API robusta como a TMDB API para criar uma aplicação web intuitiva e funcional. Através da pesquisa de filmes, adição à watchlist, remoção de filmes da watchlist e avaliação de filmes, os usuários foram capazes de explorar, organizar e compartilhar sua paixão por filmes de forma eficiente.