main.js:

```
import { App, Back } from './components/App.js';
import { getMovies, getId } from './lib/tmdb.js';

const main = document.querySelector("#root");

getId().then((data) => { // Obtém os dados de todos os filmes
   return window.location.hash.slice(9); // retorna o ID do filme do hash da URL
})

getMovies().then((data) => { // Obtém os dados de todos os filmes
   document.getElementById('root').appendChild(App(data)); // adicionando o componente como filho do elemente root
})

const init = () => { // A função init agora recebe dados como argumento
   window.addEventListener("hashchange", () => {
      main.innerHTML = ""; // Limpa o conteúdo principal ao mudar o hash
         if (window.location.hash.startsWith("#details")) {
            console.log(window.location.hash.startsWith("#details"))
         }
        else {
            main.appendChild(App(data));
         }
   });
}

window.addEventListener('load', () => { // escutador, falta template // escutador de carregamento
  //  main.appendChild(Back())  // Adiciona o componente principal com os dados de todos os filmes
   init(); // Inicializa a aplicação passando os dados
});
```

1. Importando Módulos: No início, o código importa alguns módulos e funções de outros arquivos para usar neste arquivo.
2. Selecionando Elementos HTML: Ele seleciona um elemento no HTML com o ID "root" e o armazena na variável `main`.
3. Obtendo Dados de Filmes: Ele chama uma função chamada `getMovies()` para obter informações sobre filmes.
4. Iniciando a Aplicação: A função `init()` é chamada quando a página é carregada e ela escuta mudanças no hash da URL. Se o hash começar com "#details", ele limpa o conteúdo principal. Caso contrário, ele adiciona os dados dos filmes ao conteúdo principal.
5. Evento de Carregamento: Quando a página é totalmente carregada, ele executa a função `init()`.

Basicamente, este código é uma aplicação simples que exibe informações sobre filmes em uma página da web e responde a mudanças na URL. Ele usa JavaScript para fazer isso, manipulando o conteúdo da página dinamicamente.

tmdb.js:

```
const options = { // história de usuário 1
   method: 'GET', // indicando que a solicitação será uma solicitação de leitura de dados da API. Em uma solicitação GET, os parâmetros são enviados na URL da solicitação.
   headers: { // cabeçalho.
      accept: 'application/json', // Define o tipo de conteúdo que o cliente está disposto a aceitar da API. No caso JSON
      Authorization: 'Bearer eyJhbGciOiJIUzI1NiJ9.eyJhdWQiOiI1MDkzNjA4NGFiNjU3ZjA1OGMxMzczODk1MDI4MWVmNyIsInN1YiI6IjY2MGQ4YmMyZDQwMGYzMDE2NDA1ZDhjNCIsInNjb3BlcyI6WyJhcGlfcmVhZCJdLCJ2ZXJzaW9uIjoxfQ.bLbpeBECzSBZJHtkDKdfY9whOAQrFSDZBXGJVHwselc'
   }
};

export function getMovies() {

   // função fetch('url' e o options (variável que criei na linha 5)) faz solicitações http. Aqui enviei uma solicitação get para a url que peguei. A API. JSON Objeto javascript
   return fetch('https://api.themoviedb.org/3/discover/movie?include_adult=false&include_video=false&language=en-US&page=1&sort_by=popularity.desc', options)
      .then(response => response.json()) // função .then() é encadeada para manipular a resposta da solicitação. Neste caso, a resposta HTTP é convertida em formato JSON usando o método .json()
      .then((data) => {
         return data
      })
      .catch(error => console.error('Erro ao obter dados da API: ', error)); // para lidar com erros que ocorrem durante a execução da promessa
}

const secondPage = { // história de usuário 2
   method: 'GET',
   headers: {
     accept: 'application/json',
     Authorization: 'Bearer eyJhbGciOiJIUzI1NiJ9.eyJhdWQiOiI1MDkzNjA4NGFiNjU3ZjA1OGMxMzczODk1MDI4MWVmNyIsInN1YiI6IjY2MGQ4YmMyZDQwMGYzMDE2NDA1ZDhjNCIsInNjb3BlcyI6WyJhcGlfcmVhZCJdLCJ2ZXJzaW9uIjoxfQ.bLbpeBECzSBZJHtkDKdfY9whOAQrFSDZBXGJVHwselc'
   }
 };

 export function getId(movie_id) {

 return fetch(`https://api.themoviedb.org/3/movie/${movie_id}?language=en-US`, secondPage) // dinâmico
   .then(response => response.json())
   .then((data) => {       return data
   })
   .catch(error => console.error('Erro ao obter dados da API: ', error));
}
```

1. Opções de Requisição: No início, ele define algumas opções para as solicitações HTTP, incluindo o método de solicitação e os cabeçalhos, como o tipo de conteúdo aceito e a autorização.
2. Função `getMovies()`: Esta função faz uma solicitação GET para a API do TMDB para obter informações sobre filmes. Ele usa as opções definidas anteriormente para configurar a solicitação. Ele retorna os dados dos filmes obtidos da API.
3. Função `getId(movie_id)`: Esta função recebe um ID de filme como parâmetro e faz uma solicitação GET para a API do TMDB para obter informações sobre esse filme específico. Ele usa o ID do filme na URL da solicitação. Retorna os dados do filme com o ID fornecido.

Ambas as funções utilizam o método `fetch()` para fazer as solicitações HTTP para a API do TMDB e usam `.then()` para lidar com a resposta da solicitação, convertendo-a para JSON e retornando os dados. Se houver algum erro durante a solicitação, ele é capturado e registrado no console.

app.js:

```
export const App = (data) => {

  // aqui crio um elemento de div
  const el = document.createElement('div');

  // Adiciona uma classe ao elemento principal para estilização
  el.classList.add('poster-container');

  data.results.forEach((item) => {
    el.innerHTML += `
    <div class="poster-container">
    <dl itemscope itemtype="Filmes" class="content__card">
      <dt><a href="/#details-${item.id}"><img
src="https://image.tmdb.org/t/p/w500/${item.poster_path}"
alt="Imagem do Filme" class="image__card"></a></dt>
      <h2 itemprop="title" class="title">${item.title}</h2>
      <dd itemprop="release_date"
class="releaseDate">${item.release_date}</dd>
    </dl>
  </div>
    `;
  });
  return el;
}

export const Back = (data) => {
  const details = document.createElement('div');
  details.classList.add('poster-container');

  data((item) => {
    details.innerHTML += `
    <div class="poster-container">
    <h2 itemprop="title" class="title">${data.title}</h2>
    <dl itemscope itemtype="Filmes" class="content__card">
      <dt><img
src="https://image.tmdb.org/t/p/w500/${data.poster_path}"
alt="Imagem do Filme" class="image__card"></dt>
      <dd itemprop="overview"
class="overview">${data.overview}</dd>
      <dd itemprop="release_date"
class="releaseDate">${data.release_date}</dd>
      <dd itemprop="vote_average"
class="voteAverage">${data.vote_average}</dd>
    </dl>
</div>
`;
  });
  return details;
}
```

1. Função `App(data)`: Esta função cria e retorna um elemento HTML `<div>` que contém cartões de filme. Ela recebe um objeto `data` como argumento, que contém informações sobre os filmes.

- Para cada filme no objeto `data`, ele cria um cartão de filme dentro do elemento `<div>`. Cada cartão contém uma imagem do filme, o título e a data de lançamento.

2. Função `Back(data)`: Esta função cria e retorna um elemento HTML `<div>` que contém detalhes de um filme específico. Ela recebe um objeto `data` como argumento, que contém informações sobre o filme específico.

- Ele cria um cartão de detalhes para o filme dentro do elemento `<div>`. Este cartão contém o título do filme, uma imagem, uma visão geral do filme, a data de lançamento e a classificação média do filme.
- No entanto, há um pequeno problema na função `Back(data)`, onde `data` está sendo usada como uma função, mas deveria ser apenas um objeto de dados. Para corrigir isso, você pode remover os parênteses em torno de `data` na linha `data((item) => {...})` para que seja apenas `data` e não uma função.