Une image contenant texte, capture d’écran, Police, Site web

Description générée automatiquement

Une image contenant texte, capture d’écran, Site web, Page web

Description générée automatiquement

Pour obtenir un jeton d'API auprès de TMdb, voici les étapes à suivre :

1. Créez un compte sur le site de [TMdb](https://www.themoviedb.org/) en cliquant sur "Rejoindre TMdb" et en vous inscrivant avec votre adresse e-mail et un mot de passe.
2. Une fois connecté, accédez à votre compte, puis cliquez sur "Plus -> API".
3. Suivez le lien "API link" pour récupérer votre clé d'API.

Cette clé d'API sera incluse dans les requêtes que vous enverrez au serveur de TMdb pour récupérer des informations sur les films, telles que des listes de films, des images de posters, et des informations détaillées sur les films spécifiques​

<https://www.themoviedb.org/>

Une image contenant texte, capture d’écran, logiciel, Page web

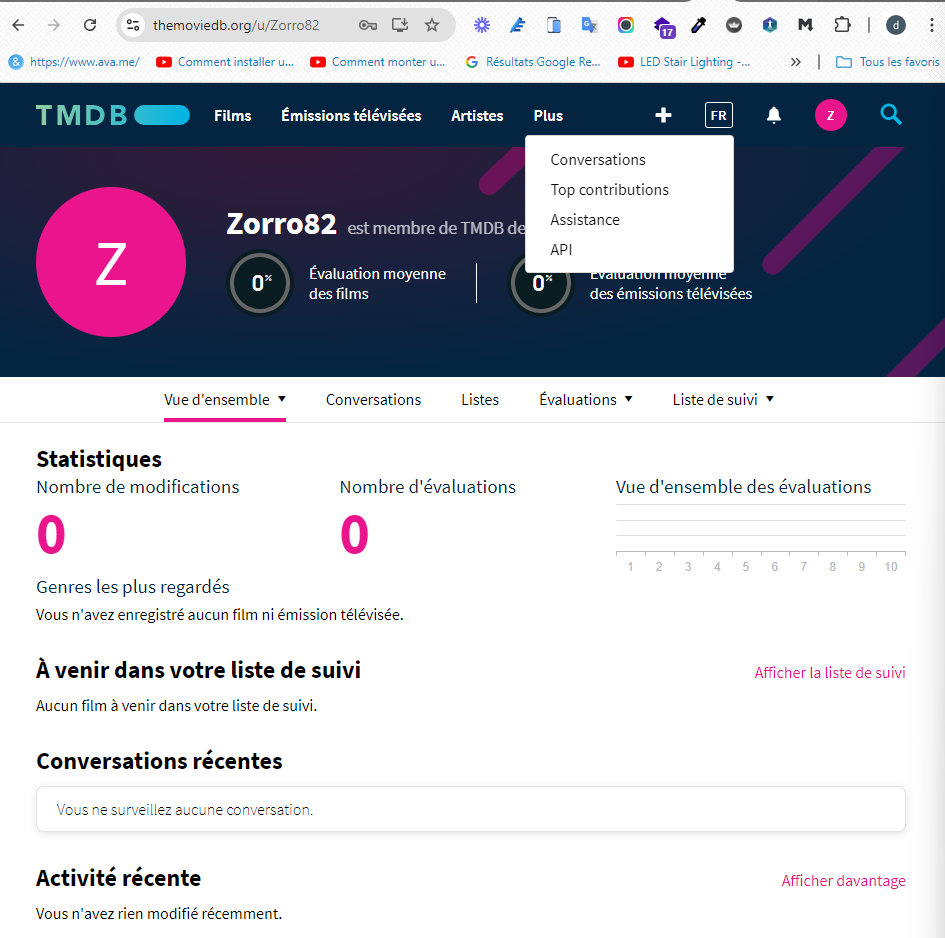
Description générée automatiquement

Une image contenant texte, Appareils électroniques, capture d’écran, logiciel

Description générée automatiquement

Une image contenant texte, capture d’écran, Site web, Page web

Description générée automatiquement



Une image contenant texte, capture d’écran, logiciel, Page web

Description générée automatiquement

Une image contenant texte, capture d’écran, logiciel, Page web

Description générée automatiquement

Une image contenant texte, capture d’écran, Page web, Site web

Description générée automatiquement

Une image contenant texte, capture d’écran, Page web, Site web

Description générée automatiquement

Une image contenant texte, capture d’écran, logiciel, Page web

Description générée automatiquement

**Clé d'API :** d806e23f6f54e2b429d698c25b89ef36

**Jeton d'accès en lecture à l'API :** eyJhbGciOiJIUzI1NiJ9.eyJhdWQiOiJkODA2ZTIzZjZmNTRlMmI0MjlkNjk4YzI1Yjg5ZWYzNiIsIm5iZiI6MTcyNzk2MzM5NS40NTI1MTMsInN1YiI6IjY2ZmU5ZDcyNmZjNzRlNTc1NmY3ZDQzZSIsInNjb3BlcyI6WyJhcGlfcmVhZCJdLCJ2ZXJzaW9uIjoxfQ.8TXf3YtkqeSJQRK9eV6ETQiDgQDTM0MJoZ4GRvUlWR8

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Une image contenant texte, capture d’écran, logiciel, Page web

Description générée automatiquement

Une image contenant texte, capture d’écran, logiciel, Système d’exploitation

Description générée automatiquement

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

Une image contenant texte, capture d’écran, logiciel, affichage

Description générée automatiquement

Si le module React-Dom n existe pas il faut l’installer :

Une image contenant capture d’écran, texte

Description générée automatiquement

Une image contenant texte, capture d’écran, logiciel, Site web

Description générée automatiquement

Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Description générée automatiquement

Une image contenant texte, capture d’écran, logiciel, affichage

Description générée automatiquement

Une image contenant texte, capture d’écran, logiciel, Police

Description générée automatiquement

Pour créer un composant "Film" en React qui affichera le résumé et le poster d'un film, suivez ces étapes :

1. Créez un fichier nommé Film.js pour définir le composant.
2. Importez le fichier CSS associé (Film.css) pour styliser le composant.
3. Déclarez le composant fonctionnel "Film", qui prendra les abstract et poster comme props.

Voici un exemple du fichier Film.js :

javascript

Copier le code

//\*\*\* fichier Film.js \*\*\*

import './Film.css';

function Film({abstract, poster}) {

return (

<div className="film">

<img src={poster} alt="poster du film" />

<p>{abstract}</p>

</div>

);

}

export default Film;

Le fichier Film.css associé peut contenir les styles suivants :

css

Copier le code

/\* fichier Film.css \*/

.film {

display: flex;

margin: 15px;

border: 1px solid black;

}

.film img {

width: 150px;

padding: 5px;

}

Ce composant affiche un poster d'un film à l'aide d'une balise img et affiche également le résumé (abstract) dans une balise <p>.

**Problème de ne pas pouvoir l’affichage cet image :**

Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Description générée automatiquement

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

L'image ne s'affiche pas parce que le chemin d'accès aux images peut être incorrect ou parce qu'il y a un problème avec la façon dont les images sont chargées. Voici quelques points à vérifier :

1. **Vérifier le chemin relatif des images** :
   * Assurez-vous que le chemin "../images/dwwm\_1.png" est correct. Ce chemin doit être relatif par rapport à l'emplacement du fichier List.js.
   * Vérifiez que le dossier images et le fichier dwwm\_1.png existent bien dans le répertoire attendu.
2. **Problème de casse dans le chemin** :
   * Assurez-vous que la casse des noms de dossiers et de fichiers correspond exactement (par exemple, Images vs images). En JavaScript, les chemins de fichiers sont sensibles à la casse.
3. **Utiliser un chemin absolu** :
   * Si le problème persiste, essayez d'utiliser un chemin absolu pour vérifier si l'image est trouvée. Par exemple :

javascript

Copier le code

{ poster: "/images/dwwm\_1.png", abstract: "Résumé du premier film" }

* + Cela signifie que le dossier images doit être dans le répertoire public de votre projet.

1. **Vérifier la présence du fichier dans le répertoire public** :
   * Il est souvent recommandé de placer les ressources statiques, comme les images, dans le répertoire public si vous voulez y accéder par un chemin direct (/images/dwwm\_1.png).
   * Si l'image est située dans public/images/dwwm\_1.png, le chemin absolu /images/dwwm\_1.png devrait fonctionner.
2. **Inspecter l'élément** :
   * Utilisez les outils de développement de votre navigateur (clic droit > Inspecter) pour vérifier la balise <img>. Cela vous permettra de voir l'URL exacte utilisée pour l'image et de vérifier s'il y a une erreur de chargement.

Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Description générée automatiquement

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

3

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Une image contenant capture d’écran, texte, affichage, Rectangle

Description générée automatiquement

Pour récupérer des données de l'API TMDB, voici les étapes à suivre en utilisant React et la fonction fetch :

1. **Créer une fonction pour interagir avec l'API TMDB** :

Vous allez créer un fichier TMDBApi.js dans le répertoire src de votre projet. Cette fonction permettra de faire des requêtes à l'API TMDB.

javascript

Copier le code

//\*\*\* fichier TMDBApi.js \*\*\*

const API\_TOKEN = "votre\_API\_Key\_TMDB";

// le paramètre text = le film recherché, page = le numéro de page voulu

export default function getFilmsFromTMdbWithText(text, page) {

const url = `https://api.themoviedb.org/3/search/movie?api\_key=${API\_TOKEN}&language=fr&query=${text}&page=${page}`;

return fetch(url)

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

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

}

* + Remplacez "votre\_API\_Key\_TMDB" par votre clé d'API obtenue sur le site de TMDB.
  + La fonction getFilmsFromTMdbWithText() retourne une promesse qui renvoie les résultats de la requête sous forme de tableau de films.

1. **Intégrer la récupération des films dans le composant List** :

Pour afficher les films récupérés, vous allez intégrer cette fonction dans votre composant List en utilisant le hook useEffect pour effectuer la requête lorsque le composant se charge ou lorsqu'une recherche est effectuée.

javascript

Copier le code

//\*\*\* fichier List.js \*\*\*

import React, { useState, useEffect } from 'react';

import { useParams } from 'react-router-dom';

import './List.css';

import Film from '../Film/Film';

import getFilmsFromTMdbWithText from '../TMDBApi'; // importer la fonction d'API

function List() {

const [films, setFilms] = useState([]);

const { data } = useParams(); // récupérer la recherche de l'utilisateur

useEffect(() => {

if (data) {

getFilmsFromTMdbWithText(data, 1) // rechercher le film avec le paramètre "data"

.then((response) => {

if (response && response.results) {

setFilms(response.results); // mettre à jour l'état des films

}

});

}

}, [data]); // exécuter à chaque changement du paramètre "data"

return (

<div className="container">

<h2>Liste des films</h2>

{films.map((film) => {

return (

<Film

key={film.id}

poster={`https://image.tmdb.org/t/p/w300${film.poster\_path}`}

abstract={film.overview}

/>

);

})}

</div>

);

}

export default List;

* + **useState** : Utilisé pour stocker la liste des films.
  + **useEffect** : Permet de déclencher la récupération des films à chaque fois que la variable data (paramètre de recherche) change.
  + La fonction getFilmsFromTMdbWithText(data, 1) est appelée pour récupérer les films correspondant au texte de recherche data.
  + Le poster\_path est utilisé avec l'URL de base https://image.tmdb.org/t/p/w300 pour afficher l'image.

1. **Vérifier le rendu des films** :

Une fois que vous avez intégré cette fonctionnalité, les films récupérés devraient s'afficher dans votre composant List, chacun avec son poster et son résumé.

Utilisez la console du navigateur pour vérifier que les données sont bien récupérées en ajoutant un console.log(response.results) dans la fonction then() de getFilmsFromTMdbWithText().

Si vous rencontrez des problèmes, assurez-vous que :

* Votre clé d'API est valide et bien insérée dans API\_TOKEN.
* Le format de la réponse est correctement vérifié (par exemple, response && response.results).
* L'URL utilisée pour récupérer les films est correcte et conforme à la documentation de TMDB.

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

Une image contenant texte, capture d’écran, logiciel, Page web

Description générée automatiquement

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Pour afficher les composants Film dans votre composant List, nous allons maintenant nous assurer que les films récupérés depuis l'API sont correctement rendus en tant que composants React. Nous avons déjà vu comment utiliser useEffect pour obtenir les données des films, et nous allons maintenant nous concentrer sur l'affichage de ces données via des composants Film.

**Étape 3 : Affichage des Composants « Film »**

1. **Déclarer un état films avec useState** : L'état films est utilisé pour stocker la liste des films obtenus à partir de l'API TMDB.

javascript

Copier le code

const [films, setFilms] = useState([]);

1. **Utiliser useEffect pour récupérer les films** : Avec useEffect, nous allons appeler getFilmsFromTMdbWithText() chaque fois que la variable data change. Nous mettrons à jour l'état films avec les résultats obtenus.

javascript

Copier le code

useEffect(() => {

if (data) {

getFilmsFromTMdbWithText(data, 1)

.then((response) => {

if (response && response.results) {

setFilms(response.results); // Mettre à jour l'état avec les films obtenus

}

})

.catch((error) => console.log(error)); // Gérer les erreurs

}

}, [data]);

1. **Utiliser .map() pour afficher chaque film** : Maintenant, nous allons utiliser films.map() pour parcourir la liste des films et rendre un composant Film pour chacun d'eux.

javascript

Copier le code

return (

<div className="container">

<h2>Liste des films</h2>

{films.length > 0 ? (

films.map((film) => (

<Film

key={film.id} // Utiliser l'ID unique du film comme clé

poster={`https://image.tmdb.org/t/p/w300${film.poster\_path}`} // URL complète de l'image du poster

abstract={film.overview} // Résumé du film

/>

))

) : (

<p>Aucun film trouvé pour la recherche : "{data}"</p> // Message si aucun film n'est trouvé

)}

</div>

);

1. **Définition du Composant Film** : Assurez-vous que le composant Film prend en charge les props poster et abstract. Voici un exemple du composant Film :

javascript

Copier le code

// Film.js

import React from 'react';

import './Film.css'; // Assurez-vous que le fichier CSS contient les styles nécessaires

function Film({ poster, abstract }) {

return (

<div className="film">

<img src={poster} alt="Poster du film" /> {/\* Affichage du poster \*/}

<p>{abstract}</p> {/\* Affichage du résumé \*/}

</div>

);

}

export default Film;

1. **CSS pour styliser les composants Film** (optionnel) : Vous pouvez ajouter du CSS pour améliorer l'affichage des composants Film. Par exemple :

css

Copier le code

/\* Film.css \*/

.film {

display: flex;

flex-direction: column;

align-items: center;

margin: 20px;

border: 1px solid #ddd;

padding: 10px;

border-radius: 5px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

}

.film img {

width: 150px;

height: auto;

margin-bottom: 10px;

}

.film p {

font-size: 14px;

color: #333;

}

**Résumé**

* **Étape 1** : Utilisez useState pour créer un état films qui stocke les données des films.
* **Étape 2** : Utilisez useEffect pour appeler l'API TMDB et mettre à jour l'état avec les films récupérés.
* **Étape 3** : Utilisez .map() pour afficher chaque film en utilisant le composant Film.
* **Étape 4** : Créez le composant Film qui prend en charge l'affichage du poster et du résumé.
* **Étape 5** : Stylisez les composants Film avec du CSS pour un rendu agréable.

En suivant ces étapes, vos films récupérés depuis l'API TMDB seront affichés sous forme de composants Film dans la page List.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Pour afficher correctement le poster de chaque film dans le composant Film, vous devez vous assurer que l'URL de l'image est bien formée, en utilisant la propriété poster\_path retournée par l'API TMDB. Voici les étapes détaillées pour vous assurer que le poster de chaque film s'affiche correctement :

**Étape 4 : Affichage du Poster de Chaque Film**

1. **URL de Base pour le Poster** : Le poster des films est accessible via l'URL de base de l'API TMDB. Vous devez ajouter la valeur poster\_path retournée par l'API à cette URL de base. L'URL de base pour les images TMDB est :

ruby

Copier le code

https://image.tmdb.org/t/p/w300

Ici, w300 correspond à la largeur de l'image en pixels. Vous pouvez utiliser d'autres tailles disponibles, par exemple w500 pour une meilleure qualité.

1. **Mettre à Jour le Composant Film** : Mettez à jour votre composant Film pour qu'il prenne en compte l'URL complète du poster, en combinant l'URL de base avec poster\_path.

javascript

Copier le code

// Film.js

import React from 'react';

import './Film.css';

function Film({ poster, abstract }) {

return (

<div className="film">

<img

src={`https://image.tmdb.org/t/p/w300${poster}`} // Construction de l'URL complète de l'image

alt="Poster du film" // Texte alternatif

/>

<p>{abstract}</p> {/\* Affichage du résumé du film \*/}

</div>

);

}

export default Film;

1. **Assurer la Correction dans le Composant List** : Dans votre composant List, lorsque vous mappez la liste des films pour afficher chaque composant Film, utilisez la propriété poster\_path :

javascript

Copier le code

return (

<div className="container">

<h2>Liste des films</h2>

{films.length > 0 ? (

films.map((film) => (

<Film

key={film.id} // Utiliser l'ID du film comme clé unique

poster={film.poster\_path} // Passer "poster\_path" comme prop au composant Film

abstract={film.overview} // Utiliser "overview" pour le résumé du film

/>

))

) : (

<p>Aucun film trouvé pour la recherche : "{data}"</p> // Message si aucun film n'est trouvé

)}

</div>

);

1. **Gérer les Postes Manquants** : Il est possible que certains films n'aient pas de poster\_path. Pour éviter d'afficher une image cassée, vous pouvez vérifier si poster\_path est disponible avant de construire l'URL ou fournir un poster par défaut.

javascript

Copier le code

function Film({ poster, abstract }) {

const posterUrl = poster

? `https://image.tmdb.org/t/p/w300${poster}`

: "https://via.placeholder.com/300x450?text=No+Image+Available"; // Image de remplacement si le poster est manquant

return (

<div className="film">

<img

src={posterUrl}

alt="Poster du film"

/>

<p>{abstract}</p>

</div>

);

}

1. **Ajouter du CSS pour Styliser le Poster** : Pour améliorer l'apparence des posters des films, vous pouvez ajouter du CSS au fichier Film.css :

css

Copier le code

/\* Film.css \*/

.film {

display: flex;

flex-direction: column;

align-items: center;

margin: 20px;

padding: 10px;

border: 1px solid #ddd;

border-radius: 5px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

}

.film img {

width: 150px;

height: auto;

margin-bottom: 10px;

border-radius: 5px;

}

.film p {

font-size: 14px;

color: #333;

text-align: center;

}

**Résumé**

* Utilisez l'URL de base https://image.tmdb.org/t/p/w300 pour former l'URL complète du poster en ajoutant poster\_path.
* Mettez à jour le composant Film pour qu'il reçoive le poster\_path via ses props et construise l'URL complète du poster.
* Gérer les cas où poster\_path est manquant en fournissant une image par défaut.
* Ajoutez du CSS pour styliser l'affichage des posters.

Ces étapes garantiront que chaque film affiche correctement son poster dans votre application React, offrant ainsi une meilleure expérience visuelle pour l'utilisateur.

Pour entrer votre clé API TMDB dans votre application React, vous pouvez la stocker dans un fichier séparé pour éviter qu'elle soit directement codée en dur dans vos composants. Voici plusieurs façons de procéder :

**1. Créer un Fichier de Configuration**

Vous pouvez créer un fichier de configuration où vous stockez votre clé API.

1. **Créer un fichier config.js dans votre répertoire src** :

javascript

Copier le code

// src/config.js

export const TMDB\_API\_KEY = 'votre\_clé\_API\_TMDB'; // Remplacez par votre clé API

1. **Importer la clé API dans TMDBApi.js** :

javascript

Copier le code

// TMDBApi.js

import { TMDB\_API\_KEY } from '../config'; // Importer la clé API depuis le fichier de configuration

export default function getFilmsFromTMdbWithText(text, page) {

// Construction de l'URL avec la clé API

const url = `https://api.themoviedb.org/3/search/movie?api\_key=${TMDB\_API\_KEY}&language=fr&query=${text}&page=${page}`;

// Faire la requête fetch

return fetch(url)

.then((response) => response.json()) // Transformer la réponse en JSON

.catch((error) => console.log(error)); // Gérer les erreurs

}

**2. Utiliser un Fichier .env pour les Variables d'Environnement**

Pour des raisons de sécurité, il est recommandé de stocker les clés API dans un fichier .env (variables d'environnement). Cela évite de laisser des informations sensibles dans votre code source.

1. **Créer un fichier .env dans le répertoire racine de votre projet** :

bash

Copier le code

REACT\_APP\_TMDB\_API\_KEY=votre\_clé\_API\_TMDB

**Remarque :** Le préfixe REACT\_APP\_ est obligatoire pour que les variables soient accessibles dans une application React.

1. **Ajouter .env au fichier .gitignore** : Cela permet d'éviter de commettre ce fichier dans votre dépôt git, car il contient des informations sensibles.
2. **Accéder à la clé API dans votre code** :

Dans votre fichier TMDBApi.js, vous pouvez accéder à la clé API à l'aide de process.env :

javascript

Copier le code

// TMDBApi.js

const API\_TOKEN = process.env.REACT\_APP\_TMDB\_API\_KEY; // Accéder à la clé API depuis les variables d'environnement

export default function getFilmsFromTMdbWithText(text, page) {

// Construction de l'URL avec la clé API

const url = `https://api.themoviedb.org/3/search/movie?api\_key=${API\_TOKEN}&language=fr&query=${text}&page=${page}`;

// Faire la requête fetch

return fetch(url)

.then((response) => response.json()) // Transformer la réponse en JSON

.catch((error) => console.log(error)); // Gérer les erreurs

}

1. **Redémarrer le Serveur de Développement** : Chaque fois que vous modifiez le fichier .env, vous devez redémarrer votre serveur de développement (npm start ou yarn start).

**Avantages d'Utiliser .env :**

* **Sécurité** : Ne pas exposer directement la clé API dans votre code source.
* **Flexibilité** : Permet de changer la clé API facilement sans toucher au code.

**Exemples d'Utilisation dans Votre Code**

Si vous suivez l'une des méthodes ci-dessus, l'utilisation de la clé API est simplifiée dans votre fichier TMDBApi.js et vous pouvez l'appeler chaque fois que vous avez besoin d'envoyer une requête à l'API TMDB.

**Remarque**

Bien que ces méthodes permettent de sécuriser la clé API dans le code source, si votre application est déployée côté client (comme une application React classique), la clé API reste accessible via les outils de développement du navigateur. Pour une meilleure sécurité, envisagez de créer un backend qui interagit avec l'API TMDB et cache la clé API côté serveur.

**Rappel :**

**Clé d'API :** d806e23f6f54e2b429d698c25b89ef36

**Jeton d'accès en lecture à l'API :** eyJhbGciOiJIUzI1NiJ9.eyJhdWQiOiJkODA2ZTIzZjZmNTRlMmI0MjlkNjk4YzI1Yjg5ZWYzNiIsIm5iZiI6MTcyNzk2MzM5NS40NTI1MTMsInN1YiI6IjY2ZmU5ZDcyNmZjNzRlNTc1NmY3ZDQzZSIsInNjb3BlcyI6WyJhcGlfcmVhZCJdLCJ2ZXJzaW9uIjoxfQ.8TXf3YtkqeSJQRK9eV6ETQiDgQDTM0MJoZ4GRvUlWR8

Une image contenant texte, capture d’écran, logiciel, Police

Description générée automatiquement

// fichier index.js

import React from 'react';

import ReactDOM from 'react-dom';

import { BrowserRouter } from 'react-router-dom'; // Importer BrowserRouter

import App from './components/App';

import 'bootstrap/dist/css/bootstrap.min.css';

ReactDOM.render(

  <BrowserRouter> {/\* Le seul Router devrait être ici \*/}

    <App />

  </BrowserRouter>,

  document.getElementById('root')

);

// App.js

import './App.css';

import Home from './Home/Home';

import About from './About/About'; // Ajout de l'import pour le composant About

import { Routes, Route } from 'react-router-dom';

// import { Link } from 'react-router-dom';

// import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';

import Layout from './Layout/Layout';

import Search from './Search/Search';

import List from './List/List';

import Film from './Film/Film';

function App() {

  return (

    // <Router>

      <div className="App">

        {/\* <nav>

          <Link to="/">Home</Link> | <Link to="/about">About</Link>

        </nav> \*/}

        <Routes>

          <Route path="/" element={<Layout />}>

            <Route path="/" element={<Home />} />

            <Route path="/about" element={<About />} /> {/\* Ajout de la route About \*/}

            <Route path="/search" element={<Search />} />

             {/\* Route pour accéder à la page List sans paramètre \*/}

            <Route path="/List" element={<List />} />

             {/\* Route pour accéder à la page List avec un paramètre \*/}

            <Route path="/List/:data" element={<List />} />

            <Route path="/Film" element={<Film />} />

          </Route>

        </Routes>

      </div>

    // </Router>

  );

}

export default App;

/\* App.css \*/

.nav-bar {

  background-color: #333;

  overflow: hidden;

}

.nav-bar a {

  float: left;

  display: block;

  color: white;

  text-align: center;

  padding: 14px 16px;

  text-decoration: none;

}

.nav-bar a:hover {

  background-color: #ddd;

  color: black;

}

.container {

  margin-top: 100px;

  text-align: center;

}

.App {

  text-align: center;

}

.App-logo {

  height: 40vmin;

  pointer-events: none;

}

@media (prefers-reduced-motion: no-preference) {

  .logo {

    animation: App-logo-spin infinite 20s linear;

  }

}

.App {

  background-color: #282c34;

  min-height: 100vh;

  display: flex;

  flex-direction: column;

  align-items: center;

  justify-content: center;

  font-size: calc(10px + 2vmin);

  color: white;

}

.App-button {

  color: blue;

  font-size: 1.5rem;

}

.App-link {

  color: #61dafb;

}

@keyframes App-logo-spin {

  from {

    transform: rotate(0deg);

  }

  to {

    transform: rotate(360deg);

  }

}

// Home.js

import React from 'react';

import { useState } from 'react';

import './Home.css';

import logo2 from '../images/dwwm\_1.png';

function Home() {

  const [animation, setAnimation] = useState(true);

  const toggleAnimation = () => {

    setAnimation(animation => !animation);

  };

  return (

    <div className="container text-center">

      <img

        src={logo2}

        className={`logo ${animation && "rotate"}`}

        alt="logo"

      />

      <h1>TP React Développeur Web et Web Mobile</h1>

      <p>Bienvenue sur la page d'accueil.</p>

      <button

        className="btn btn-primary mt-3"

        onClick={toggleAnimation}

      >

        Modifier l’animation

      </button>

    </div>

  );

}

export default Home;

/\* Home.css \*/

.nav-bar {

  background-color: #333;

  overflow: hidden;

}

.nav-bar a {

  float: left;

  display: block;

  color: white;

  text-align: center;

  padding: 14px 16px;

  text-decoration: none;

}

.nav-bar a:hover {

  background-color: #ddd;

  color: black;

}

.container {

  margin-top: 100px;

  text-align: center;

}

  .App {

    background-color: #282c34;

    min-height: 100vh;

    display: flex;

    flex-direction: column;

    align-items: center;

    justify-content: center;

    font-size: calc(10px + 2vmin);

    color: white;

  }

  .logo {

    height: 30vmin;

    pointer-events: none;

    margin-bottom: 50px;

  }

  @media (prefers-reduced-motion: no-preference) {

    .rotate {

    }

    /\* .logo {

      animation: rotate-spin infinite 20s linear;

    } \*/

}

  .App-link {

    color: #61dafb;

  }

  @keyframes rotate-spin {

    from {

      transform: rotate(0deg);

    }

    to {

      transform: rotate(360deg);

    }

  }

// About.js

import React from 'react';

import './About.css';

function About() {

  return (

    <div className="container">

      <h2>Page About</h2>

      <p>Ceci est la page à propos de notre application.</p>

    </div>

  );

}

export default About;

/\* About.css \*/

.nav-bar {

  background-color: #333;

  overflow: hidden;

}

.nav-bar a {

  float: left;

  display: block;

  color: white;

  text-align: center;

  padding: 14px 16px;

  text-decoration: none;

}

.nav-bar a:hover {

  background-color: #ddd;

  color: black;

}

.container {

    margin-top: 100px;

    text-align: center;

  }

// fichier Layout.js

import { Link, Outlet } from 'react-router-dom';

function Layout() {

  return (

    <div>

      <nav className="navbar navbar-expand-lg navbar-dark bg-primary">

        <div className="container-fluid">

          <div className="collapse navbar-collapse" id="navbarNav">

            <ul className="navbar-nav">

              <li className="nav-item">

                <Link to="/" className="nav-link">Home</Link>

              </li>

              <li className="nav-item">

                <Link to="/about" className="nav-link">About</Link>

              </li>

              <li>

                <Link to="/search" className="nav-link">Search</Link>

              </li>

              <li>

                <Link to="/List" className="nav-link">List</Link>

              </li>

              {/\* <li>

                <Link to="/Film" className="nav-link">Film</Link>

              </li> \*/}

            </ul>

          </div>

        </div>

      </nav>

      <div className="container mt-4">

        <Outlet />

      </div>

    </div>

  );

}

export default Layout;

//\*\*\* fichier Search.js \*\*\*

import React from 'react';

import { useState } from 'react';

import { useNavigate } from 'react-router-dom'; // Importer useNavigate

import './Search.css';

function Search() {

  const [search, setSearch] = useState('');

  const navigate = useNavigate(); // Initialiser useNavigate

  const handleChange = (e) => {

    setSearch(e.target.value);

  };

  const handleSubmit = (e) => {

    e.preventDefault();  // Empêche le comportement par défaut de soumission de formulaire

    // Redirection vers la page List avec la valeur saisie comme paramètre

    if (search.trim() !== '') {

      navigate(`/List/${search}`); // Navigue vers /List/<valeur\_saisie>

    }

  };

  return (

    <div className="search">

      <p>Recherchez ici vos films préférés.</p>

      <form onSubmit={handleSubmit}>

        <fieldset>

          <label htmlFor="idSearch">Votre recherche</label>

          <input

            id="idSearch"

            name="search"

            type="text"

            value={search}

            onChange={handleChange}

          />

        </fieldset>

        <input type="submit" value="Rechercher" />

      </form>

    </div>

  );

}

export default Search;

/\* fichier Search.css \*/

.container {

    margin-top: 100px;

    text-align: center;

  }

.search {

    padding: 5px 20px;

}

.search fieldset {

    margin: 20px 0px;

}

.search label {

    display: block;

}

.search input {

    Margin-left: 100px;

    Font-size: 1.4rem;

}

// // List.js

// import React from 'react';

import React, { useState, useEffect } from 'react'; // Importation du hook useEffect depuis React

import { useParams } from 'react-router-dom'; // Importation du hook useParams pour accéder aux paramètres de l'URL

import './List.css'; // Importation du fichier CSS pour styliser le composant List

import Film from '../Film/Film'; // Importation du composant Film pour afficher chaque film

import getFilmsFromTMdbWithText from '../TMDBApi/TMDBApi'; // Importation de la fonction pour interagir avec l'API TMDB

function List() {

  const [films, setFilms] = useState([]);

  const { data } = useParams(); // Récupération du paramètre "data" de l'URL

  useEffect(() => {

    if (data) {

      getFilmsFromTMdbWithText(data, 1)

        .then((response) => {

          if (response && response.results) {

            setFilms(response.results); // Mettre à jour l'état "films" avec les résultats

          }

        })

        .catch((error) => console.log(error)); // Gérer les erreurs

    }

  }, [data]); // Déclencher l'effet à chaque changement de "data"

  return (

    <div className="container">

      <h2>Liste des films</h2>

      <br />

      {films.length > 0 ? (

         films.map((film) => (

            <Film

            key={film.id} // Utiliser l'ID du film comme clé unique

            poster={film.poster\_path} // Passer "poster\_path" comme prop au composant Film

            abstract={film.overview} // Utiliser "overview" pour le résumé du film

           />

          ))

      ) : (

        <p>Aucun film trouvé pour la recherche : "{data}"</p> // Message si aucun film n'est trouvé

      )}

    </div>

  );

}

export default List;

/\* List.css \*/

.nav-bar {

  background-color: #333;

  overflow: hidden;

}

.nav-bar a {

  float: left;

  display: block;

  color: white;

  text-align: center;

  padding: 14px 16px;

  text-decoration: none;

}

.nav-bar a:hover {

  background-color: #ddd;

  color: black;

}

.container {

    margin-top: 100px;

    text-align: center;

  }

Voici comment améliorer le style de la navigation et de l'affichage des films en utilisant les fichiers que vous avez fournis :

**Étape 1 : Amélioration du Style des Pages Navigateur avec les Boutons Home, About, Search et List**

* **Modification dans Layout.js :** Vous avez déjà un menu de navigation dans le fichier Layout.js qui affiche les boutons "Home", "About", "Search" et "List". Vous pouvez améliorer le style de ces boutons en utilisant des classes CSS Bootstrap et en ajoutant une personnalisation.
* **Code Modifié dans Layout.js** :

// fichier Layout.js

import { Link, Outlet } from 'react-router-dom';

import './Layout.css'; // Ajout du CSS pour personnaliser le style de la navigation

function Layout() {

return (

<div>

<nav className="navbar navbar-expand-lg navbar-dark bg-primary">

<div className="container-fluid">

<div className="collapse navbar-collapse" id="navbarNav">

<ul className="navbar-nav">

<li className="nav-item">

<Link to="/" className="nav-link">Home</Link>

</li>

<li className="nav-item">

<Link to="/about" className="nav-link">About</Link>

</li>

<li className="nav-item">

<Link to="/search" className="nav-link">Search</Link>

</li>

<li className="nav-item">

<Link to="/List" className="nav-link">List</Link>

</li>

</ul>

</div>

</div>

</nav>

<div className="container mt-4">

<Outlet />

</div>

</div>

);

}

export default Layout;

* **Modification dans Layout.css :** Vous pouvez créer un fichier Layout.css pour améliorer le style des liens et des boutons.

/\* Layout.css \*/

.nav-link {

font-weight: bold;

transition: background-color 0.3s ease, color 0.3s ease;

}

.nav-link:hover {

background-color: #ffffff;

color: #0056b3;

border-radius: 5px;

}

.navbar {

padding: 15px;

}

**Étape 2 : Amélioration de l'Affichage des Films dans une Grille (Page List)**

* Pour afficher les films en utilisant une grille de 3 colonnes et 3 lignes, nous allons modifier List.js et List.css.

1. **Modification dans List.js :** Vous avez déjà un composant List.js qui récupère les films depuis l'API TMDB et les affiche. Nous allons ajouter une classe pour styliser l'affichage en grille.

// List.js

import React, { useState, useEffect } from 'react';

import { useParams } from 'react-router-dom';

import './List.css';

import Film from '../Film/Film';

import getFilmsFromTMdbWithText from '../TMDBApi/TMDBApi';

function List() {

const [films, setFilms] = useState([]);

const { data } = useParams();

useEffect(() => {

if (data) {

getFilmsFromTMdbWithText(data, 1)

.then((response) => {

if (response && response.results) {

setFilms(response.results.slice(0, 9)); // Limiter à 9 films pour afficher une grille 3x3

}

})

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

}

}, [data]);

return (

<div className="container">

<h2>Liste des films</h2>

<div className="grid-container">

{films.length > 0 ? (

films.map((film) => (

<Film

key={film.id}

poster={film.poster\_path}

abstract={film.overview}

/>

))

) : (

<p>Aucun film trouvé pour la recherche : "{data}"</p>

)}

</div>

</div>

);

}

export default List;

1. **Modification dans List.css :** Ajoutez des styles pour créer une grille avec 3 colonnes et 3 lignes.

/\* List.css \*/

.grid-container {

display: grid;

grid-template-columns: repeat(3, 1fr); /\* 3 colonnes \*/

grid-gap: 20px; /\* Espace entre les éléments \*/

margin-top: 20px;

}

.container {

margin-top: 50px;

text-align: center;

}

**Étape 3 : Amélioration du Composant Film**

Pour le composant Film.js, vous pouvez améliorer le style des affiches et des descriptions.

* **Modification dans Film.js** :

// Film.js

import React from 'react';

import './Film.css';

function Film({ abstract, poster }) {

const posterUrl = poster

? `https://image.tmdb.org/t/p/w300${poster}`

: "https://via.placeholder.com/300x450?text=No+Image+Available";

return (

<div className="film">

<img

src={posterUrl}

alt="Poster du film"

/>

<p className="film-abstract">{abstract}</p>

</div>

);

}

export default Film;

* **Modification dans Film.css** :

/\* Film.css \*/

.film {

display: flex;

flex-direction: column;

align-items: center;

background-color: #f9f9f9;

border-radius: 8px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

padding: 20px;

transition: transform 0.3s ease;

}

.film img {

width: 200px;

height: auto;

margin-bottom: 15px;

border-radius: 8px;

}

.film:hover {

transform: scale(1.05);

}

.film-abstract {

font-size: 14px;

color: #333;

text-align: center;

padding: 10px;

}

**Résumé des Changements**

1. **Navigation** :
   * Amélioration du style de la barre de navigation avec des couleurs plus contrastées et des effets au survol pour les boutons Home, About, Search, et List (Layout.js et Layout.css).
2. **Affichage des Films dans une Grille** :
   * Utilisation de grid-template-columns: repeat(3, 1fr) pour organiser les films dans une grille de 3 colonnes et 3 lignes (List.js et List.css).
   * Limitation de l'affichage à 9 films pour s'assurer qu'ils s'affichent bien dans une grille 3x3.
3. **Composant Film** :
   * Ajout d'un style pour afficher les films avec une image et une description, et ajout d'un effet de survol pour rendre l'interface plus interactive (Film.js et Film.css).

Ces améliorations permettront d'avoir une interface plus esthétique et professionnelle, tout en facilitant l'exploration des films pour les utilisateurs.

Pour vous assurer que la barre de navigation reste en haut de l'écran, et pour recentrer l'affichage de la liste des films, vous pouvez faire quelques ajustements dans le CSS et les composants React. Voici les étapes pour corriger ce problème :

**Étape 1 : Fixer la Barre de Navigation en Haut de l'Écran**

Pour que la barre de navigation soit toujours en haut de l'écran, vous devez utiliser le CSS position: fixed. Cela garantit que la barre reste en place même lorsque vous faites défiler la page.

**Modification dans Layout.css** :

css

Copier le code

/\* Layout.css \*/

.navbar {

position: fixed;

top: 0;

left: 0;

width: 100%;

z-index: 1000; /\* S'assurer que la barre est toujours au-dessus des autres éléments \*/

background-color: #007bff;

}

.container {

margin-top: 100px; /\* Ajouter une marge supérieure pour compenser la barre de navigation fixe \*/

}

**Étape 2 : Ajuster l'Affichage de la Liste des Films pour le Centrer**

Pour centrer l'affichage de la liste des films, il est recommandé de travailler avec flexbox ou de recentrer la grille dans le conteneur principal.

**Modification dans List.css** :

* Mettez à jour le CSS pour centrer le contenu horizontalement, en utilisant margin: 0 auto et en définissant une largeur maximale.

css

Copier le code

/\* List.css \*/

.container {

max-width: 1200px;

margin: 100px auto; /\* Centrer horizontalement et ajouter de la marge en haut \*/

text-align: center;

padding: 20px;

}

.grid-container {

display: grid;

grid-template-columns: repeat(3, 1fr); /\* 3 colonnes \*/

gap: 20px; /\* Espace entre les éléments \*/

justify-items: center; /\* Centrer les items dans chaque colonne \*/

}

**Étape 3 : Vérification de la Page List**

Pour que les changements prennent effet, assurez-vous que les composants List et Layout importent les bons fichiers CSS et que List est rendu dans un conteneur correctement configuré.

* **List.js** : Vérifiez que le conteneur principal (div className="container") est bien utilisé pour appliquer le style mis à jour.

**Mise à Jour Complète dans List.js** :

javascript

Copier le code

// List.js

import React, { useState, useEffect } from 'react';

import { useParams } from 'react-router-dom';

import './List.css';

import Film from '../Film/Film';

import getFilmsFromTMdbWithText from '../TMDBApi/TMDBApi';

function List() {

const [films, setFilms] = useState([]);

const { data } = useParams();

useEffect(() => {

if (data) {

getFilmsFromTMdbWithText(data, 1)

.then((response) => {

if (response && response.results) {

setFilms(response.results.slice(0, 9)); // Limiter à 9 films pour afficher une grille 3x3

}

})

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

}

}, [data]);

return (

<div className="container">

<h2>Liste des films</h2>

<div className="grid-container">

{films.length > 0 ? (

films.map((film) => (

<Film

key={film.id}

poster={film.poster\_path}

abstract={film.overview}

/>

))

) : (

<p>Aucun film trouvé pour la recherche : "{data}"</p>

)}

</div>

</div>

);

}

export default List;

**Étape 4 : Assurez-vous d'Inclure les Fichiers CSS Correctement**

* **App.js** : Assurez-vous que Layout est correctement inclus et que le CSS est chargé.

**Exemple dans App.js** :

javascript

Copier le code

// App.js

import React from 'react';

import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';

import Layout from './components/Layout';

import Home from './components/Home';

import Search from './components/Search';

import List from './components/List/List';

import About from './components/About';

import './App.css';

function App() {

return (

<Router>

<Layout>

<Routes>

<Route path="/" element={<Home />} />

<Route path="/about" element={<About />} />

<Route path="/search" element={<Search />} />

<Route path="/list" element={<List />} />

</Routes>

</Layout>

</Router>

);

}

export default App;

**Résumé des Changements**

1. **Fixer la Barre de Navigation** :
   * Utiliser position: fixed pour la barre de navigation dans Layout.css.
   * Ajouter margin-top à .container pour éviter que le contenu ne soit masqué par la barre.
2. **Centrer l'Affichage de la Liste des Films** :
   * Utiliser margin: 0 auto pour centrer la .container.
   * Utiliser justify-items: center pour aligner les éléments dans la grille.

Avec ces modifications, la barre de navigation restera en haut de l'écran lorsque vous faites défiler la page, et la liste des films sera bien centrée, offrant une interface plus agréable à utiliser.

Pour centrer le texte "Aucun film trouvé pour la recherche" lorsque aucun résultat n'est disponible, vous pouvez ajuster le CSS du message dans List.css. Voici comment procéder :

**Étape 1 : Modifier le Texte de "Aucun Film Trouvé" pour le Centrer**

* Ajoutez une classe CSS au paragraphe qui contient le message "Aucun film trouvé pour la recherche" dans List.js.
* Utilisez ensuite le CSS pour centrer ce message à l'intérieur du conteneur.

**Modification dans List.js** :

javascript

Copier le code

// List.js

import React, { useState, useEffect } from 'react';

import { useParams } from 'react-router-dom';

import './List.css';

import Film from '../Film/Film';

import getFilmsFromTMdbWithText from '../TMDBApi/TMDBApi';

function List() {

const [films, setFilms] = useState([]);

const { data } = useParams();

useEffect(() => {

if (data) {

getFilmsFromTMdbWithText(data, 1)

.then((response) => {

if (response && response.results) {

setFilms(response.results.slice(0, 9)); // Limiter à 9 films pour afficher une grille 3x3

}

})

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

}

}, [data]);

return (

<div className="container">

<h2>Liste des films</h2>

<div className="grid-container">

{films.length > 0 ? (

films.map((film) => (

<Film

key={film.id}

poster={film.poster\_path}

abstract={film.overview}

/>

))

) : (

<p className="no-results">Aucun film trouvé pour la recherche : "{data}"</p>

)}

</div>

</div>

);

}

export default List;

**Étape 2 : Ajouter le CSS pour Centrer le Texte "Aucun Film Trouvé"**

* Ajoutez une classe .no-results dans le fichier List.css pour styliser le message "Aucun film trouvé" et le centrer.

**Modification dans List.css** :

css

Copier le code

/\* List.css \*/

.container {

max-width: 1200px;

margin: 100px auto;

text-align: center;

padding: 20px;

}

.grid-container {

display: grid;

grid-template-columns: repeat(3, 1fr);

gap: 20px;

justify-items: center;

margin-top: 20px;

}

.no-results {

grid-column: span 3; /\* Étendre sur les 3 colonnes de la grille \*/

font-size: 1.5em;

color: #666;

margin-top: 50px;

text-align: center;

}

**Explications**

1. **Classe .no-results dans List.css** :
   * **grid-column: span 3;** : Cette propriété permet de faire en sorte que le texte s'étende sur les 3 colonnes de la grille, le centrant automatiquement dans la disposition.
   * **font-size: 1.5em;** : Agrandit légèrement le texte pour le rendre plus visible.
   * **color: #666;** : Donne une couleur de texte légèrement grisée, ce qui le rend visible mais moins dominant.
   * **margin-top: 50px;** : Ajoute de l'espace au-dessus du texte pour séparer le message de la grille des films.

Avec ces modifications, le message "Aucun film trouvé pour la recherche" sera bien centré et clairement visible lorsque les résultats de recherche ne renverront aucun film.

Pour ajouter un titre avec le nom du film trouvé, le nombre d'affichages des films, et styliser l'arrière-plan du conteneur jusqu'à la fin de la grille, nous allons procéder avec les modifications suivantes :

**Étape 1 : Mise à Jour du Composant List.js**

Nous allons ajouter un titre avec le nom de la recherche, le nombre de films trouvés, et assurer la cohérence dans l'affichage de l'arrière-plan et des dimensions.

**Modification dans List.js** :

* Ajoutez un titre pour afficher le nom du film recherché.
* Ajoutez également le nombre de films trouvés.

javascript

Copier le code

// List.js

import React, { useState, useEffect } from 'react';

import { useParams } from 'react-router-dom';

import './List.css';

import Film from '../Film/Film';

import getFilmsFromTMdbWithText from '../TMDBApi/TMDBApi';

function List() {

const [films, setFilms] = useState([]);

const { data } = useParams();

useEffect(() => {

if (data) {

getFilmsFromTMdbWithText(data, 1)

.then((response) => {

if (response && response.results) {

setFilms(response.results.slice(0, 9)); // Limiter à 9 films pour afficher une grille 3x3

}

})

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

}

}, [data]);

return (

<div className="list-page">

<h2>Résultats de recherche pour : "{data}"</h2>

<p className="film-count">{films.length} films trouvés</p>

<div className="grid-container">

{films.length > 0 ? (

films.map((film) => (

<Film

key={film.id}

poster={film.poster\_path}

abstract={film.overview}

/>

))

) : (

<p className="no-results">Aucun film trouvé pour la recherche : "{data}"</p>

)}

</div>

</div>

);

}

export default List;

**Étape 2 : Ajouter le CSS pour Harmoniser l'Affichage et Styliser la Page List**

Pour avoir un arrière-plan homogène, une dimension cohérente, et un affichage aligné, nous allons mettre à jour List.css.

**Modification dans List.css** :

* Ajoutez un style pour toute la page List (.list-page) afin d'assurer que l'arrière-plan couvre toute la section de la grille.
* Assurez-vous que les films ont la même dimension en utilisant des styles cohérents.

css

Copier le code

/\* List.css \*/

/\* Arrière-plan et alignement de la page entière \*/

.list-page {

background-color: #f0f0f0; /\* Couleur d'arrière-plan pour toute la page List \*/

padding: 40px;

min-height: 100vh; /\* S'assure que la page couvre au moins toute la hauteur de la fenêtre \*/

display: flex;

flex-direction: column;

align-items: center;

}

h2 {

color: #333;

font-size: 2em;

margin-bottom: 10px;

}

.film-count {

color: #555;

font-size: 1.2em;

margin-bottom: 20px;

}

.grid-container {

display: grid;

grid-template-columns: repeat(3, 1fr);

gap: 20px;

justify-items: center;

width: 100%;

max-width: 1200px;

margin: 0 auto;

padding-bottom: 40px; /\* Ajoute de l'espace au bas de la grille \*/

}

/\* Film card styles pour des dimensions homogènes \*/

.film {

background-color: #fff;

border-radius: 8px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

transition: transform 0.3s ease;

overflow: hidden;

width: 100%;

max-width: 300px;

}

.film img {

width: 100%;

height: 450px; /\* Hauteur fixe pour garder les dimensions homogènes \*/

object-fit: cover;

}

.film:hover {

transform: scale(1.05);

}

.film-abstract {

padding: 15px;

font-size: 0.9em;

color: #666;

text-align: justify;

}

**Explications des Changements**

1. **Ajout du Titre et du Compteur de Films** :
   * **Titre** (<h2>) : Affiche le nom du film recherché (paramètre data) pour indiquer clairement les résultats de la recherche.
   * **Compteur de Films** (<p className="film-count">) : Affiche le nombre de films trouvés grâce à la recherche.
2. **Style de la Page List (.list-page)** :
   * **background-color: #f0f0f0** : Définit une couleur d'arrière-plan pour l'ensemble de la page List.
   * **min-height: 100vh** : Assure que la page couvre toute la hauteur de l'écran, même s'il y a peu de contenu.
3. **Grille des Films (.grid-container)** :
   * **grid-template-columns: repeat(3, 1fr)** : Définit une grille de 3 colonnes.
   * **gap: 20px** : Ajoute un espacement uniforme entre les éléments de la grille.
   * **max-width: 1200px** et **width: 100%** : Assure que la grille est bien centrée et n'excède pas une largeur définie.
4. **Dimensions Homogènes pour les Films (.film)** :
   * **max-width: 300px** et **height: 450px** : Fixe la largeur et la hauteur des affiches pour assurer une apparence homogène.
   * **object-fit: cover** pour l'image : Assure que l'affiche du film couvre entièrement l'espace défini sans déformation.

Avec ces ajustements, la page List affichera le titre de la recherche, le nombre de films trouvés, un arrière-plan uniforme, et les cartes de films auront une dimension cohérente, ce qui rendra l'affichage plus harmonieux et professionnel.

Pour résoudre les problèmes suivants :

1. **Le titre et le nombre de films trouvés ne sont pas visibles correctement.**
2. **Les films sont décalés trop vers le haut.**
3. **Conserver la barre de navigation avec les boutons "Home", "About", "Search", "List".**
4. **Créer une fenêtre modale qui affiche les détails du film sélectionné.**

Je vais vous guider pour faire les corrections nécessaires.

**Étape 1 : Corriger le Positionnement et la Visibilité des Éléments**

1. **Fixer la Position de la Navigation et Ajouter de la Marge au Contenu**

* Assurez-vous que la barre de navigation ne cache pas les éléments en haut de la page en ajoutant une marge au conteneur principal.

**Modification dans Layout.css** :

css

Copier le code

/\* Layout.css \*/

.navbar {

position: fixed;

top: 0;

left: 0;

width: 100%;

z-index: 1000; /\* Assure que la barre est au-dessus des autres éléments \*/

background-color: #007bff;

}

.container {

margin-top: 120px; /\* Ajouter de la marge pour compenser la barre de navigation fixe \*/

}

1. **Marge Supplémentaire sur la Page List**

Assurez-vous que la page List a une marge suffisante pour que les éléments ne soient pas décalés.

**Modification dans List.css** :

css

Copier le code

/\* List.css \*/

.list-page {

background-color: #f0f0f0;

padding: 40px;

min-height: 100vh;

display: flex;

flex-direction: column;

align-items: center;

margin-top: 60px; /\* Ajoute de la marge supplémentaire pour éviter le chevauchement \*/

}

**Étape 2 : Créer une Fenêtre Modale pour Afficher les Détails du Film Sélectionné**

Pour créer une fenêtre modale qui s'ouvre lorsque vous cliquez sur un film, nous devons :

1. Ajouter un état dans List.js pour gérer l'ouverture et la fermeture de la modale.
2. Créer un composant Modal qui affichera les détails du film.
3. **Ajoutez l'État de la Modale et le Composant dans List.js** :

* Ajoutez un état pour la sélection du film, ainsi que les fonctions nécessaires pour ouvrir et fermer la modale.

**Modification dans List.js** :

javascript

Copier le code

// List.js

import React, { useState, useEffect } from 'react';

import { useParams } from 'react-router-dom';

import './List.css';

import Film from '../Film/Film';

import Modal from '../Modal/Modal'; // Nouveau composant pour la fenêtre modale

import getFilmsFromTMdbWithText from '../TMDBApi/TMDBApi';

function List() {

const [films, setFilms] = useState([]);

const [selectedFilm, setSelectedFilm] = useState(null); // État pour le film sélectionné

const { data } = useParams();

useEffect(() => {

if (data) {

getFilmsFromTMdbWithText(data, 1)

.then((response) => {

if (response && response.results) {

setFilms(response.results.slice(0, 9)); // Limiter à 9 films pour afficher une grille 3x3

}

})

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

}

}, [data]);

// Fonction pour ouvrir la modale

const openModal = (film) => {

setSelectedFilm(film);

};

// Fonction pour fermer la modale

const closeModal = () => {

setSelectedFilm(null);

};

return (

<div className="list-page">

<h2>Résultats de recherche pour : "{data}"</h2>

<p className="film-count">{films.length} films trouvés</p>

<div className="grid-container">

{films.length > 0 ? (

films.map((film) => (

<div key={film.id} onClick={() => openModal(film)}>

<Film

poster={film.poster\_path}

abstract={film.overview}

/>

</div>

))

) : (

<p className="no-results">Aucun film trouvé pour la recherche : "{data}"</p>

)}

</div>

{selectedFilm && (

<Modal film={selectedFilm} closeModal={closeModal} />

)}

</div>

);

}

export default List;

1. **Créer un Composant Modal Pour Afficher les Détails du Film** :

Créez un nouveau composant Modal.js qui sera utilisé pour afficher les détails du film sélectionné.

**Nouveau fichier : Modal.js** :

javascript

Copier le code

// Modal.js

import React from 'react';

import './Modal.css';

function Modal({ film, closeModal }) {

const posterUrl = film.poster\_path

? `https://image.tmdb.org/t/p/w300${film.poster\_path}`

: "https://via.placeholder.com/300x450?text=No+Image+Available";

return (

<div className="modal-overlay">

<div className="modal-content">

<button className="close-button" onClick={closeModal}>X</button>

<img src={posterUrl} alt={`Poster de ${film.title}`} />

<h2>{film.title}</h2>

<p>{film.overview}</p>

</div>

</div>

);

}

export default Modal;

1. **Styles pour la Fenêtre Modale (Modal.css)** :

Ajoutez un fichier Modal.css pour styliser la fenêtre modale.

**Nouveau fichier : Modal.css** :

css

Copier le code

/\* Modal.css \*/

.modal-overlay {

position: fixed;

top: 0;

left: 0;

width: 100%;

height: 100%;

background: rgba(0, 0, 0, 0.7);

display: flex;

justify-content: center;

align-items: center;

z-index: 1000;

}

.modal-content {

background: white;

padding: 20px;

border-radius: 8px;

width: 80%;

max-width: 500px;

text-align: center;

position: relative;

}

.close-button {

position: absolute;

top: 10px;

right: 10px;

background: red;

color: white;

border: none;

border-radius: 50%;

width: 30px;

height: 30px;

cursor: pointer;

}

.modal-content img {

width: 100%;

height: auto;

border-radius: 8px;

margin-bottom: 15px;

}

.modal-content h2 {

margin-bottom: 10px;

color: #333;

}

.modal-content p {

color: #666;

text-align: justify;

}

**Explications des Modifications**

1. **Gestion de la Modale dans List.js** :
   * Ajout de l'état selectedFilm pour garder une référence au film sélectionné.
   * Utilisation des fonctions openModal et closeModal pour gérer l'ouverture et la fermeture de la modale.
2. **Création du Composant Modal** :
   * **Modal.js** affiche les détails du film sélectionné, tels que le titre, l'affiche, et le résumé.
   * Ajoutez un bouton de fermeture (X) pour que l'utilisateur puisse fermer la modale.
3. **Stylisation de la Modale (Modal.css)** :
   * La classe .modal-overlay ajoute un arrière-plan semi-transparent couvrant toute la page.
   * La classe .modal-content contient les détails du film, avec une bordure arrondie et un fond blanc pour se détacher du reste de la page.

Avec ces modifications, la page "List" affichera correctement le titre et le nombre de films trouvés. La barre de navigation sera toujours visible, et vous pourrez ouvrir une fenêtre modale pour afficher les détails du film sélectionné, offrant une expérience utilisateur plus interactive et agréable.

Pour assurer que tous les conteneurs de film aient la même dimension, même lorsque le film est vide ou s'il n'y a pas de texte, vous pouvez définir explicitement des dimensions fixes et utiliser des valeurs par défaut pour l'affiche et le texte. Voici comment régler ce problème :

**Étape 1 : Assurez-vous que Tous les Conteneurs de Films Ont une Dimension Fixe**

* Ajoutez une largeur et une hauteur fixes au conteneur .film pour que tous les films aient la même taille, quelle que soit la quantité de contenu.
* Ajoutez également une image par défaut et un texte par défaut si les données sont manquantes.

**Modification dans Film.js** :

* Ajoutez des valeurs par défaut pour le poster et l'abstract si ces données sont manquantes.

javascript

Copier le code

// Film.js

import React from 'react';

import './Film.css';

function Film({ abstract, poster }) {

const posterUrl = poster

? `https://image.tmdb.org/t/p/w300${poster}`

: "https://via.placeholder.com/300x450?text=No+Image+Available";

const filmAbstract = abstract || "Description non disponible."; // Valeur par défaut si le texte est vide

return (

<div className="film">

<img src={posterUrl} alt="Poster du film" />

<p className="film-abstract">{filmAbstract}</p>

</div>

);

}

export default Film;

**Étape 2 : Styliser le Composant Film pour Assurer des Dimensions Homogènes**

* Définissez des dimensions fixes pour le conteneur du film et l'image.
* Assurez-vous que même sans texte, l'espace est réservé pour éviter les différences de taille.

**Modification dans Film.css** :

css

Copier le code

/\* Film.css \*/

.film {

display: flex;

flex-direction: column;

align-items: center;

justify-content: space-between; /\* Pour espacer uniformément l'image et le texte \*/

background-color: #fff;

border-radius: 8px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

transition: transform 0.3s ease;

overflow: hidden;

width: 100%;

max-width: 300px;

height: 550px; /\* Hauteur fixe pour que tous les films aient la même taille \*/

padding: 15px;

}

.film img {

width: 100%;

height: 400px; /\* Hauteur fixe pour que toutes les images soient uniformes \*/

object-fit: cover;

margin-bottom: 10px;

}

.film-abstract {

font-size: 0.9em;

color: #666;

text-align: justify;

min-height: 50px; /\* Fixe la hauteur minimum pour le texte \*/

}

**Explications des Changements**

1. **Valeur Par Défaut pour l'Affiche et le Texte** :
   * **posterUrl** : Si le poster n'est pas disponible, utilisez une image par défaut (https://via.placeholder.com/300x450?text=No+Image+Available).
   * **filmAbstract** : Si le texte est vide, utilisez un texte par défaut ("Description non disponible.").
2. **Dimensions Fixes pour le Conteneur de Film (.film)** :
   * **max-width: 300px et height: 550px** : Assurez-vous que chaque film ait la même largeur et la même hauteur.
   * **padding: 15px** : Ajoutez du rembourrage autour du contenu pour le rendre plus agréable à regarder.
3. **Dimensions Fixes pour l'Image (.film img)** :
   * **height: 400px** : Assurez-vous que toutes les images ont la même hauteur.
   * **object-fit: cover** : Assurez-vous que l'image couvre tout l'espace défini sans se déformer.
4. **Hauteur Minimum pour le Texte (.film-abstract)** :
   * **min-height: 50px** : Réservez de l'espace pour le texte, même s'il est absent. Cela garantit que tous les conteneurs ont la même dimension, qu'il y ait du texte ou non.

Ces modifications garantissent que les conteneurs des films ont une apparence homogène et que tous les films s'affichent de manière uniforme, même si certaines informations sont manquantes. Cela permet également d'améliorer la mise en page générale et de rendre l'application plus agréable visuellement.

Pour résoudre le problème de texte qui semble être tronqué dans chaque carte de film, vous pouvez ajuster le CSS pour limiter la hauteur du texte de la description et afficher une indication visuelle (comme "..." pour indiquer que le texte est coupé). Voici comment faire cela :

**Étape 1 : Ajuster la Longueur du Texte de la Description**

* Utilisez overflow: hidden pour masquer tout texte qui dépasse une certaine hauteur.
* Ajoutez text-overflow: ellipsis pour indiquer qu'il y a plus de texte que ce qui est visible.

**Modification dans Film.css** :

css

Copier le code

/\* Film.css \*/

.film {

display: flex;

flex-direction: column;

align-items: center;

justify-content: space-between; /\* Pour espacer uniformément l'image et le texte \*/

background-color: #fff;

border-radius: 8px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

transition: transform 0.3s ease;

overflow: hidden;

width: 100%;

max-width: 300px;

height: 550px; /\* Hauteur fixe pour que tous les films aient la même taille \*/

padding: 15px;

}

.film img {

width: 100%;

height: 400px; /\* Hauteur fixe pour que toutes les images soient uniformes \*/

object-fit: cover;

margin-bottom: 10px;

}

.film-abstract {

font-size: 0.9em;

color: #666;

text-align: justify;

min-height: 50px; /\* Hauteur minimum pour garantir de l'espace pour le texte \*/

max-height: 70px; /\* Limite la hauteur de la description \*/

overflow: hidden; /\* Masque le texte qui dépasse \*/

text-overflow: ellipsis; /\* Affiche "..." si le texte est coupé \*/

line-height: 1.2em; /\* Hauteur de ligne pour s'assurer que les lignes sont uniformes \*/

display: -webkit-box;

-webkit-line-clamp: 3; /\* Limite le nombre de lignes à 3 \*/

-webkit-box-orient: vertical;

}

**Étape 2 : Explications des Changements**

1. **Limiter la Hauteur de la Description (.film-abstract)** :
   * **max-height: 70px** : Limite la hauteur de la description afin qu'elle ne dépasse pas une certaine taille. Cela garantit que toutes les descriptions ont une hauteur uniforme.
   * **overflow: hidden** et **text-overflow: ellipsis** : Masque tout texte qui dépasse et ajoute des points de suspension ("...") pour indiquer que la description est coupée.
2. **Clamp pour Limiter le Nombre de Lignes** :
   * **-webkit-line-clamp: 3** : Utilisé avec -webkit-box-orient: vertical, cela permet de limiter la description à 3 lignes maximum, quelle que soit sa longueur.

**Résultat Attendu**

Avec ces modifications, la description des films sera limitée à trois lignes, et toute information supplémentaire sera coupée avec des points de suspension ("..."). Cela garantira que toutes les cartes de film ont une apparence uniforme et évitera les textes trop longs qui déséquilibrent la grille.

N'oubliez pas de recompiler et de redémarrer votre serveur pour voir les modifications prises en compte dans le navigateur.

Pour conserver le texte complet dans le conteneur du film sans le couper, il est possible de permettre au conteneur de s'ajuster à la taille du texte tout en maintenant une dimension homogène des autres éléments. Voici comment vous pouvez ajuster les styles afin que la carte du film puisse afficher le texte entier.

**Étape 1 : Ajuster la Hauteur du Conteneur de Film**

Pour éviter de couper le texte, vous pouvez laisser la hauteur de la carte s'adapter dynamiquement au contenu tout en limitant uniquement la hauteur de l'image pour maintenir une cohérence visuelle.

**Modification dans Film.css** :

css

Copier le code

/\* Film.css \*/

.film {

display: flex;

flex-direction: column;

align-items: center;

justify-content: flex-start; /\* Aligner le contenu du haut vers le bas \*/

background-color: #fff;

border-radius: 8px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

transition: transform 0.3s ease;

overflow: hidden;

width: 100%;

max-width: 300px;

padding: 15px;

margin-bottom: 20px; /\* Ajouter de l'espace entre les cartes \*/

}

.film img {

width: 100%;

height: 400px; /\* Hauteur fixe pour l'image \*/

object-fit: cover; /\* Garder l'image proportionnelle \*/

margin-bottom: 10px;

}

.film-abstract {

font-size: 0.9em;

color: #666;

text-align: justify;

padding: 10px 0; /\* Espacement pour le texte \*/

height: auto; /\* Laisser le texte s'étendre dynamiquement \*/

}

**Étape 2 : Assurez-vous que Tous les Films Ont une Dimension Homogène**

Avec ces ajustements, la hauteur de chaque conteneur de film dépendra de la longueur du texte. Les images auront toujours la même hauteur, assurant ainsi une cohérence visuelle pour les affiches, mais les descriptions seront complètement affichées sans être coupées.

**Explications des Changements** :

1. **Ajustement de la Hauteur du Conteneur (.film)** :
   * **justify-content: flex-start** : Pour aligner tout le contenu (image et texte) en commençant du haut.
   * **overflow: hidden** et **height: auto** : Laissez la hauteur du conteneur s'ajuster dynamiquement au texte.
2. **Hauteur Fixe pour l'Image (.film img)** :
   * La hauteur de l'image est fixée à 400px pour garder la cohérence visuelle entre les différentes cartes de films, même si la longueur des descriptions varie.
3. **Texte (.film-abstract)** :
   * **height: auto** : Laissez le texte s'étendre sans être limité en hauteur.
   * **padding: 10px 0** : Ajoutez de l'espace pour rendre le texte plus lisible.

**Résultat Attendu**

Avec ces modifications, la carte de chaque film aura une image de taille fixe et le texte de description pourra s'étendre dynamiquement sans être coupé. Cela permet d'afficher tout le contenu de la description sans perdre d'informations tout en maintenant une apparence cohérente pour les affiches.

D'accord, si vous constatez que le texte de la description est toujours coupé, il est probable que la hauteur de chaque conteneur n’est pas ajustée dynamiquement en fonction de son contenu. Nous allons donc adapter la structure du CSS pour s'assurer que le texte entier soit toujours visible.

Je vous propose une autre approche où l'on s'assure que le conteneur s'adapte dynamiquement à la taille du contenu et que l'image et la description sont correctement alignées. Voici les étapes à suivre :

**Étape 1 : Supprimer les Limites de Hauteur et Laisser le Conteneur S'étendre**

Assurez-vous que chaque conteneur de film a la possibilité de s'adapter à la hauteur totale du contenu, y compris l'image et la description.

**Modification dans Film.css :**

css

Copier le code

/\* Film.css \*/

.film {

display: flex;

flex-direction: column;

align-items: center;

justify-content: flex-start;

background-color: #fff;

border-radius: 8px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

transition: transform 0.3s ease;

overflow: hidden;

width: 100%;

max-width: 300px;

padding: 15px;

margin-bottom: 20px; /\* Ajouter de l'espace entre les cartes \*/

height: auto; /\* Laisser la hauteur du conteneur s'adapter au contenu \*/

}

.film img {

width: 100%;

max-height: 400px; /\* Fixe la hauteur maximale de l'image \*/

object-fit: cover; /\* Assure que l'image s'adapte au conteneur \*/

margin-bottom: 10px;

}

.film-abstract {

font-size: 0.9em;

color: #666;

text-align: justify;

padding: 10px 0;

height: auto; /\* Laisser la hauteur du texte s'ajuster au contenu \*/

}

**Étape 2 : S'assurer que Chaque Élément s'Affiche Correctement**

1. **Hauteur Auto du Conteneur .film** :
   * La hauteur du conteneur .film est maintenant réglée sur auto pour permettre au conteneur de s'adapter à la quantité de contenu (image + texte).
2. **Limiter la Hauteur de l'Image** :
   * En définissant max-height: 400px sur l'image, vous vous assurez que toutes les images restent dans une taille raisonnable, mais sans limiter le conteneur en fonction du texte.
3. **Pas de Limitation pour le Texte (.film-abstract)** :
   * La hauteur du texte est réglée sur auto pour permettre à la description de s'afficher intégralement.

**Étape 3 : Tests et Ajustements**

* **Vérifiez le Comportement** : Une fois que vous avez effectué ces modifications, testez l'affichage sur votre navigateur pour vérifier que la description des films n'est plus coupée.
* **Adaptation aux Différents Scénarios** : Selon la longueur des descriptions, vous devrez peut-être ajuster d'autres aspects du style, comme l'espacement entre les films.

Ces changements devraient permettre à chaque conteneur de film de s'étendre dynamiquement pour inclure tout le contenu, que le texte soit long ou court, garantissant ainsi une apparence uniforme dans l'interface.

Pour créer une navigation avec le logo de l'AFPA à gauche et les liens de navigation (Home, About, Search, List) à droite, et ajouter une signature de l'équipe en pied de page, voici comment vous pouvez procéder :

**Étape 1 : Mettre à Jour le Composant Layout.js**

* Ajoutez le logo de l'AFPA à gauche de la barre de navigation.
* Alignez les éléments de navigation à droite.

**Modification dans Layout.js** :

// Layout.js

import { Link, Outlet } from 'react-router-dom';

import './Layout.css';

import logo from '../images/logo.png'; // Assurez-vous d'avoir un logo d'AFPA

function Layout() {

return (

<div>

<nav className="navbar">

<div className="navbar-container">

<div className="navbar-logo">

<img src={logo} alt="Logo AFPA" />

</div>

<ul className="navbar-links">

<li className="nav-item">

<Link to="/" className="nav-link">Home</Link>

</li>

<li className="nav-item">

<Link to="/about" className="nav-link">About</Link>

</li>

<li className="nav-item">

<Link to="/search" className="nav-link">Search</Link>

</li>

<li className="nav-item">

<Link to="/list" className="nav-link">List</Link>

</li>

</ul>

</div>

</nav>

<div className="container mt-4">

<Outlet />

</div>

<footer className="footer">

<p>Projet réalisé par l'équipe: Jean Dupont, Marie Durant, Sophie Martin, etc.</p>

</footer>

</div>

);

}

export default Layout;

**Étape 2 : Styliser le Composant Layout.css**

Pour avoir un logo aligné à gauche et les éléments de navigation alignés à droite, utilisez flexbox pour organiser la barre de navigation. De plus, créez un pied de page avec la signature de l'équipe.

**Modification dans Layout.css** :

css

Copier le code

/\* Layout.css \*/

/\* Navigation Bar \*/

.navbar {

display: flex;

justify-content: space-between;

align-items: center;

background-color: #007bff; /\* Couleur de fond \*/

padding: 15px;

position: fixed;

top: 0;

left: 0;

width: 100%;

z-index: 1000;

box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);

}

.navbar-container {

display: flex;

justify-content: space-between;

width: 100%;

}

.navbar-logo img {

height: 40px; /\* Ajuster la taille du logo \*/

}

.navbar-links {

list-style: none;

display: flex;

gap: 20px;

margin-right: 20px;

}

.nav-item {

display: inline-block;

}

.nav-link {

color: #ffffff;

text-decoration: none;

font-weight: bold;

transition: color 0.3s ease;

}

.nav-link:hover {

color: #cccccc;

}

/\* Footer \*/

.footer {

background-color: #007bff; /\* Couleur de fond correspondant à la barre de navigation \*/

color: #ffffff;

text-align: center;

padding: 15px 0;

margin-top: 40px;

position: relative;

bottom: 0;

width: 100%;

}

.container {

margin-top: 100px; /\* Compense la hauteur de la barre de navigation fixe \*/

padding: 20px;

}

**Explications des Modifications**

1. **Navigation avec Logo et Liens Alignés** :
   * **navbar** : Utilisez flexbox avec justify-content: space-between pour aligner le logo à gauche et les liens à droite.
   * **Logo (navbar-logo)** : Ajoutez une image de logo et définissez sa hauteur pour éviter qu'il ne prenne trop de place.
   * **Liens de Navigation (navbar-links)** : Utilisez flexbox pour aligner les éléments de navigation horizontalement et ajoutez un espace entre eux avec gap: 20px.
2. **Pied de Page avec Signature de l'Équipe** :
   * **footer** : Créez un pied de page en bas de la page avec la signature de l'équipe du projet.
   * **position: relative** et bottom: 0 : Pour s'assurer que le pied de page est toujours en bas lorsque le contenu est trop court.

**Résultat Attendu**

Avec ces modifications :

* Le logo AFPA sera aligné à gauche dans la barre de navigation.
* Les liens de navigation "Home", "About", "Search", "List" seront alignés à droite.
* La barre de navigation restera en haut de l'écran grâce à position: fixed.
* En bas de la page, vous verrez un pied de page avec la signature de l'équipe du projet.

Ces ajustements amélioreront la présentation globale de votre application en rendant la navigation plus professionnelle et le pied de page informatif.

oici une explication détaillée de chaque fichier JavaScript et CSS que vous avez partagé, ainsi que la relation entre eux :

**Explications de chaque fichier**

1. **TMDBApi.js** [API de films]
   * Ce fichier gère l'appel à l'API de TMDB pour récupérer des informations sur les films en fonction d'un texte de recherche et d'un numéro de page.
   * La fonction getFilmsFromTMdbWithText() construit une URL et utilise fetch() pour faire la requête, transformant la réponse en JSON et gérant les erreurs via .catch()​(TMDBApi).
2. **Search.css** [Style de la barre de recherche]
   * Contient le style pour la barre de recherche, notamment les marges, la taille du texte, et l'apparence des champs de saisie.
   * .container détermine le style général de la zone de recherche, .search gère le style des éléments à l'intérieur (par ex., les champs de saisie et les légendes)​(Search).
3. **Search.js** [Composant de recherche]
   * Ce composant React gère la saisie de l'utilisateur et la navigation. Il inclut un champ de recherche et un bouton de soumission.
   * handleChange met à jour l'état de la recherche, et handleSubmit redirige l'utilisateur vers la page des résultats de recherche (/List/<valeur\_saisie>) en utilisant useNavigate()​(Search).
4. **App.css** [Style global de l'application]
   * Définit les styles de base pour l'application, incluant les couleurs de fond et des animations.
   * La classe .App stylise le conteneur principal de l'application avec des propriétés comme background-color et la mise en page en flex​(App).
5. **App.js** [Composant principal de l'application]
   * Le composant principal qui définit la structure de l'application et les routes disponibles.
   * Il utilise Routes et Route de react-router-dom pour définir la navigation entre différentes pages (par exemple, /, /about, /search, etc.)​(App).
6. **List.css** [Style de la liste de films]
   * Gère la présentation de la liste de films sous forme de grille (3 colonnes).
   * .container et .grid-container définissent la disposition générale, notamment en centrant le contenu et en créant une grille pour afficher les films​(List).
7. **List.js** [Composant de la liste des films]
   * Ce composant récupère la liste des films à partir de TMDBApi.js selon la recherche de l'utilisateur.
   * useParams() est utilisé pour extraire le paramètre de l'URL et faire appel à l'API pour obtenir les films correspondants. Les films sont affichés sous forme de grille en utilisant le composant Film​(List).
8. **Layout.css** [Style de la disposition générale]
   * Contient le style pour la barre de navigation, y compris les transitions lors du survol.
   * .navbar est définie pour être fixe en haut de la page, avec des couleurs spécifiques et une disposition fluide pour s'assurer qu'elle reste visible​(Layout).
9. **Layout.js** [Composant de la mise en page]
   * Ce composant crée la structure de navigation principale de l'application avec des liens vers différentes pages (Home, About, Search, List).
   * Il utilise Link pour une navigation interne fluide entre les composants, et Outlet pour afficher le contenu des sous-composants en fonction de la route active​(Layout).

**Relation entre les fichiers JavaScript et CSS**

* **App.js** est le point d'entrée principal, où la structure de l'application est définie à l'aide des composants **Layout**, **Search**, **List**, etc. Chaque route définit quel composant sera chargé (par exemple, /search charge **Search.js**).
* **Layout.js** fournit une barre de navigation constante pour l'application, stylisée par **Layout.css**. Tous les composants chargés sont inclus dans la disposition créée par **Layout**.
* **Search.js** et **List.js** sont des composants spécifiques à certaines fonctionnalités. **Search.js** permet à l'utilisateur d'effectuer une recherche, tandis que **List.js** affiche les résultats de cette recherche, en utilisant les styles définis dans **Search.css** et **List.css** respectivement.
* **TMDBApi.js** est utilisé par **List.js** pour récupérer les données nécessaires à partir de l'API de films.
* **App.css** définit des styles généraux pour l'ensemble de l'application, tels que le fond, la taille du texte et d'autres éléments de mise en page globale, affectant ainsi **App.js**.

**Diagramme de la relation entre les fichiers JavaScript**

Voici une représentation textuelle de la relation entre les fichiers :

mathematica

Copier le code

App.js

├── Layout.js

│ └── Layout.css

│

├── Search.js

│ └── Search.css

│

├── List.js

│ ├── List.css

│ └── TMDBApi.js

│

└── autres composants (Home, About, Film, etc.)

**Description des relations** :

* **App.js** est le conteneur principal qui gère les routes vers tous les autres composants.
* **Layout.js** est le composant de navigation global, tandis que **Search.js** et **List.js** sont responsables de la recherche et de l'affichage des résultats.
* **TMDBApi.js** est utilisé par **List.js** pour interagir avec l'API externe.

Si vous avez besoin d'une illustration graphique de ce diagramme, je peux vous la créer.

4o

**Vous avez dit :**

Home.css

Fichier

Home.js

JavaScript

Film.css

Fichier

Film.js

JavaScript

About.css

Fichier

About.js

JavaScript

config.js

JavaScript

PAreil en premier et ces pièces jointes ajoutants. J'ai besoin d'une illustration graphique de ce diagramme, vous pouvez vous la créer.

ChatGPT

**Explications des fichiers supplémentaires**

1. **Home.css** [Style de la page d'accueil]
   * Gère le style de la page d'accueil, notamment le conteneur .App, qui définit l'apparence générale avec un arrière-plan sombre et des animations pour le logo​(Home).
2. **Home.js** [Composant de la page d'accueil]
   * Ce composant React affiche une image (logo), un titre, un paragraphe de bienvenue, et un bouton interactif pour déclencher ou arrêter l'animation.
   * Il utilise useState pour gérer l'animation du logo, permettant à l'utilisateur de la démarrer ou l'arrêter en cliquant sur le bouton​(Home).
3. **Film.css** [Style du composant Film]
   * Définit le style pour chaque carte de film affichée dans la liste des résultats.
   * Utilise des effets de survol pour ajouter une légère animation de mise à l'échelle lorsque l'utilisateur passe la souris sur un film​(Film).
4. **Film.js** [Composant Film]
   * Ce composant reçoit deux props : abstract (le résumé du film) et poster (l'image du film).
   * Il construit l'URL complète pour l'image du film et la rend visible. S'il n'y a pas d'image, un placeholder est affiché à la place​(Film).
5. **About.css** [Style de la page "À propos"]
   * Contient les styles de la page à propos, mais ceux-ci ont été mis en commentaires. Il contient essentiellement des styles similaires aux autres fichiers CSS pour une mise en page standard​(About).
6. **About.js** [Composant "À propos"]
   * Ce composant React affiche un titre et un paragraphe expliquant la page à propos de l'application​(About).
7. **config.js** [Fichier de configuration]
   * Contient la clé API utilisée pour les requêtes à l'API de TMDB. Ce fichier est essentiel pour accéder aux données des films via l'API​(config).

**Relation entre tous les fichiers JavaScript et CSS**

* **App.js** reste le point d'entrée principal, intégrant **Home**, **About**, **Search**, **List**, et **Film**.
* **Home.js** est utilisé pour la page d'accueil avec **Home.css** pour le style.
* **About.js** est la page "À propos", stylisée avec **About.css**.
* **Search.js** est utilisé pour rechercher des films, et **Search.css** est utilisé pour le style de la zone de recherche.
* **List.js** est utilisé pour afficher une liste des films recherchés, et **List.css** pour le style des films affichés sous forme de grille.
* **Film.js** affiche les détails individuels d'un film dans la liste, en utilisant **Film.css** pour le style de chaque carte de film.
* **Layout.js** gère la structure de navigation, et **Layout.css** définit les styles de la barre de navigation.
* **TMDBApi.js** est la fonction d'API qui interagit avec l'API de films, tandis que **config.js** contient la clé API nécessaire.

**Illustration graphique du diagramme de relation entre les fichiers**

Je vais maintenant créer un diagramme pour visualiser la relation entre ces différents fichiers.

Analysé

python

Toujours afficher les détails

Copier le code

import matplotlib.pyplot as plt

import networkx as nx

# Créer un graphe dirigé

G = nx.DiGraph()

# Ajouter les nœuds (composants JavaScript et CSS)

nodes = [

"App.js", "Home.js", "Home.css", "About.js", "About.css", "Search.js", "Search.css",

"List.js", "List.css", "Film.js", "Film.css", "Layout.js", "Layout.css", "TMDBApi.js", "config.js"

]

G.add\_nodes\_from(nodes)

# Ajouter des arêtes (relations entre les fichiers)

edges = [

("App.js", "Home.js"), ("App.js", "About.js"), ("App.js", "Search.js"),

("App.js", "List.js"), ("App.js", "Layout.js"), ("List.js", "Film.js"),

("Search.js", "Search.css"), ("Home.js", "Home.css"), ("About.js", "About.css"),

("List.js", "List.css"), ("Film.js", "Film.css"), ("Layout.js", "Layout.css"),

("List.js", "TMDBApi.js"), ("TMDBApi.js", "config.js")

]

G.add\_edges\_from(edges)

# Définir la position des nœuds pour une meilleure visualisation

pos = nx.spring\_layout(G, seed=42)

# Dessiner le graphe

plt.figure(figsize=(12, 8))

nx.draw(G, pos, with\_labels=True, node\_size=3000, node\_color="skyblue", font\_size=10, font\_weight="bold", arrowsize=20)

plt.title("Diagramme des relations entre les fichiers JavaScript et CSS du projet React")

plt.show()

Diagramme des relations entre les fichiers JavaScript et CSS du projet React

Voici le diagramme représentant les relations entre les différents fichiers JavaScript et CSS de votre projet React. Chaque nœud représente un fichier, et les flèches indiquent les dépendances entre ces fichiers. Cela vous montre comment les composants et les fichiers CSS interagissent dans l'application.

Si vous avez besoin de plus de détails ou d'une autre illustration, n'hésitez pas à le demander ! ​​

4o

**Vous avez dit :**

How does Layout.js handle routing? en français

2/2

**ChatGPT a dit :**

ChatGPT

Dans **Layout.js**, le routage est géré à l'aide du composant Outlet de react-router-dom. Voici comment cela fonctionne :

* **Layout.js** définit une barre de navigation (nav) avec plusieurs liens (Link) vers différentes pages, comme Home, About, Search, et List. Ces liens permettent une navigation interne sans recharger la page.
* Le composant Outlet est utilisé pour rendre le contenu des routes enfants. Cela signifie que, lorsque l'utilisateur clique sur un lien, la partie spécifique de la page qui correspond à cette route sera affichée à l'endroit où se trouve Outlet. Ainsi, **Layout.js** sert de structure générale, notamment pour la barre de navigation, tandis que Outlet affiche les différents composants en fonction de la route active.

En résumé, **Layout.js** fournit la structure commune, telle que la barre de navigation, et Outlet gère le rendu dynamique du contenu des pages selon la route sélectionnée.

4o

**Vous avez dit :**

Comment le fichier config.js est-il utilisé?

**ChatGPT a dit :**

ChatGPT

Le fichier **config.js** est utilisé pour stocker la clé API nécessaire pour interagir avec l'API de TMDB.

Voici comment cela fonctionne dans le projet :

* **config.js** contient une constante TMDB\_API\_KEY qui stocke la clé API (export const TMDB\_API\_KEY = 'votre\_clé\_API';)​(config). Cette clé est essentielle pour accéder aux services de l'API The Movie Database (TMDB).
* Ce fichier est importé dans **TMDBApi.js** pour récupérer la clé API et l'utiliser dans les requêtes. En l'intégrant via une importation (par exemple import { TMDB\_API\_KEY } from '../Config/config';), cela permet de centraliser la configuration et de facilement mettre à jour la clé API en un seul endroit sans devoir la modifier dans tout le code​(TMDBApi).

Cette approche rend le code plus modulable, sécurisé, et facile à maintenir, car les informations sensibles (comme les clés API) sont séparées de la logique principale.

4o

**Vous avez dit :**

Comment l'API Film est-elle appelée?

**ChatGPT a dit :**

ChatGPT

L'appel à l'API pour récupérer des informations sur les films se fait principalement dans le fichier **List.js**, en utilisant la fonction définie dans **TMDBApi.js**.

Voici comment cela fonctionne :

1. **Définition de la fonction dans TMDBApi.js** :
   * Dans **TMDBApi.js**, la fonction getFilmsFromTMdbWithText(text, page) est définie pour construire une URL avec la clé API, un texte de recherche, et un numéro de page​(TMDBApi).
   * Cette fonction fait une requête fetch() à l'API de TMDB pour récupérer des films qui correspondent au texte de recherche. La réponse est ensuite transformée en JSON.
2. **Utilisation de l'API dans List.js** :
   * Dans **List.js**, la fonction getFilmsFromTMdbWithText() est appelée dans un useEffect(), lorsque le composant est rendu ou lorsque le paramètre data change (c'est-à-dire lorsqu'une nouvelle recherche est effectuée).
   * La fonction getFilmsFromTMdbWithText(data, 1) est appelée avec data, qui est le texte de recherche passé en tant que paramètre via l'URL (utilisant useParams()). Elle récupère les films de la première page de résultats​(List).
   * La réponse est ensuite utilisée pour mettre à jour l'état films en utilisant setFilms(response.results.slice(0, 9)) afin de limiter l'affichage à 9 films dans une grille.

En résumé, **List.js** appelle l'API via **TMDBApi.js** en fonction de la recherche de l'utilisateur. La fonction getFilmsFromTMdbWithText() envoie une requête à l'API TMDB, et les films obtenus sont affichés dans une grille de résultats.

4o

**Vous avez dit :**

Comment fonctionne l'interaction entre Home et Layout ?

**ChatGPT a dit :**

ChatGPT

L'interaction entre **Home** et **Layout** se fait principalement par le biais du système de routage défini dans **App.js** et via l'utilisation du composant Outlet dans **Layout.js**.

Voici comment cette interaction fonctionne :

1. **App.js : Définition des routes** :
   * **App.js** utilise Routes et Route pour définir les différentes pages de l'application. Le composant **Layout** est défini comme le conteneur principal de ces routes.
   * La route principale ("/") est associée à **Layout**, et à l'intérieur de **Layout**, il y a une sous-route pour la page d'accueil ("/") qui charge le composant **Home**. Cela signifie que **Layout** est toujours présent autour des composants enfants, y compris **Home**​(App).
2. **Layout.js : Structure de la page avec Outlet** :
   * **Layout.js** sert de cadre général pour l'application. Il comprend une barre de navigation (nav) qui est affichée sur toutes les pages.
   * **Layout** utilise Outlet pour rendre dynamiquement les composants enfants définis par les routes dans **App.js**.
   * Lorsqu'un utilisateur navigue vers la route racine ("/"), **Layout** est chargé avec la barre de navigation, et le contenu de la page d'accueil (Home) est affiché dans la zone spécifiée par Outlet​(Layout).
3. **Affichage du composant Home via Layout** :
   * Lorsqu'un utilisateur accède à la page d'accueil ("/"), **Layout** est chargé en premier, ce qui assure l'affichage de la barre de navigation.
   * Ensuite, le composant **Home** est affiché sous la barre de navigation dans la zone définie par Outlet. Cela permet de garder une structure cohérente, avec **Layout** comme contenant principal et **Home** comme contenu de la page.

En résumé, **Home** est affiché dans le cadre défini par **Layout** grâce au routage défini dans **App.js**. **Layout** gère la structure commune (comme la barre de navigation) tandis que Outlet permet de charger dynamiquement le contenu de la page appropriée, ici le composant **Home**. Cela permet une interface utilisateur homogène et bien organisée.

4o

**Vous avez dit :**

Comment styliser la barre de navigation?

**ChatGPT a dit :**

ChatGPT

Pour styliser la barre de navigation dans votre projet React, vous pouvez utiliser le fichier **Layout.css** en combinaison avec le composant **Layout.js**. Voici une explication détaillée sur la façon dont vous pouvez personnaliser la barre de navigation :

**Étapes pour styliser la barre de navigation**

1. **Définir la structure de la barre de navigation dans Layout.js** :
   * **Layout.js** comprend une barre de navigation (<nav>). Les liens de navigation (<Link>) sont définis pour les différentes sections de l'application (par exemple, Home, About, Search, List)​(Layout).
   * Ces éléments sont stylisés en ajoutant des classes CSS telles que "navbar", "nav-link", etc.
2. **Utiliser Layout.css pour appliquer des styles** :
   * Dans **Layout.css**, vous pouvez personnaliser l'apparence de la barre de navigation en utilisant des sélecteurs CSS qui correspondent aux classes attribuées dans **Layout.js**​(Layout).
   * Par exemple :

css

Copier le code

.navbar {

position: fixed;

top: 0;

left: 0;

width: 100%;

z-index: 1000; /\* Pour s'assurer que la barre de navigation est au-dessus des autres éléments \*/

background-color: #007bff; /\* Couleur de fond bleue \*/

padding: 10px 0;

}

.nav-link {

font-weight: bold;

color: white; /\* Couleur de texte blanche \*/

text-decoration: none;

transition: background-color 0.3s ease, color 0.3s ease;

padding: 10px 15px;

}

.nav-link:hover {

background-color: #ffffff; /\* Fond blanc au survol \*/

color: #0056b3; /\* Texte bleu au survol \*/

border-radius: 5px; /\* Ajout de coins arrondis \*/

}

1. **Explication des styles appliqués** :
   * .navbar : Définit le style général de la barre de navigation.
     + position: fixed : La barre de navigation reste fixée en haut, même lors du défilement.
     + width: 100% : S'étend sur toute la largeur de la page.
     + background-color: #007bff : Utilise une couleur de fond bleue.
     + z-index: 1000 : Assure que la barre de navigation est visible au-dessus des autres éléments.
   * .nav-link : Définir le style des liens de navigation.
     + font-weight: bold : Rend le texte en gras.
     + transition : Ajoute une transition douce lors du changement de couleur ou de fond.
     + padding : Ajoute de l'espace autour du texte pour agrandir la zone cliquable.
   * .nav-link:hover : Appliquer des styles lorsque l'utilisateur passe la souris sur un lien.
     + background-color: #ffffff et color: #0056b3 : Change la couleur de fond et de texte pour donner un retour visuel à l'utilisateur.
     + border-radius: 5px : Ajoute des coins arrondis pour un effet plus moderne.

**Suggestions supplémentaires pour améliorer le style :**

* **Responsivité** : Pour améliorer la compatibilité sur les appareils mobiles, ajoutez des règles @media pour ajuster la disposition des éléments de la barre de navigation lorsque la largeur de l'écran est réduite.
* **Effet actif** : Pour indiquer la page active, vous pouvez ajouter une classe .active au lien correspondant à la route actuelle et la styliser comme suit :

css

Copier le code

.nav-link.active {

background-color: #0056b3; /\* Couleur différente pour le lien actif \*/

color: #ffffff;

}

En appliquant ces styles dans **Layout.css** et en les liant correctement avec les classes attribuées dans **Layout.js**, vous pouvez créer une barre de navigation visuellement attrayante et facile à utiliser dans votre application React.