Les effets de bord.

Jusqu'ici, CinéTech fonctionne uniquement avec des données statiques stockées dans le projet. C'est parfait pour découvrir JSX, les composants et la gestion d'état, mais ce n'est pas représentatif d'une application moderne.

Dans une vraie application React :

- Les données changent.
- Elles proviennent d'API.
- Elles peuvent mettre du temps à arriver.
- Elles peuvent générer des erreurs.
- Il faut parfois sauvegarder ou synchroniser des informations.

Concrètement, notre code doit réagir au monde extérieur, et non plus seulement aux interactions directes de l'utilisateur.

Pour cela, React met à disposition un hook essentiel : useEffect(), qui permet d'exécuter du code au bon moment, par exemple :

- Au chargement du composant.
- Après une mise à jour d'état.
- Lorsqu'une valeur change.
- Ou avant la destruction d'un composant (cleanup).

Dans ce TP, nous allons l'utiliser pour récupérer des films depuis une API, et conserver des données en local.

1. Introduction au hook : useEffect()

A quoi sert le useEffect() ?

Le useEffect() permet d'effectuer une action en dehors du rendu JSX, souvent appelée effet secondaire (ou side effect).

Cela comprend par exemple

- La récupération des données
- L'écoute d'un événement
- L'écriture dans le localStorage
- La synchronisation d'un état avec une ressource externe

C'est exactement ce qu'il nous faut pour remplacer nos données statiques.

Avant de partir à l'aventure, prenons un exemple introductif :

Créons un nouveau composant TestEffect.jsx :

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

export default function TestEffect() {
  const [message, setMessage] = useState('Chargement...');

  useEffect(() => {
    setTimeout(() => {
      setMessage('Données reçues !');
    }, 2000);
  }, []);

  return <p>{message}</p>;
}
```

Explications :

- « useEffect(() => { ... }, []) » s'exécute une seule fois au chargement du composant.
- Le tableau vide « [] » signifie : aucune dépendance ce qui implique, donc pas de réexécution.
- Au bout de 2 secondes, l'état est mis à jour et l'affichage du composant se réactualise.

Travail à réaliser :

- Intégrer « TestEffect » dans App.jsx
- Tester le comportement
- Modifier le délai (500ms, 3s)
- Observer le comportement

La minute Git :
Envoyer votre code sur votre répertoire Github.
Nom du commit : « TP7 : decouverte du useEffect »

Dans la prochaine partie, nous allons faire un grand pas : connecter CinéTech à une vraie API de films.

2. Création du compte TMDB

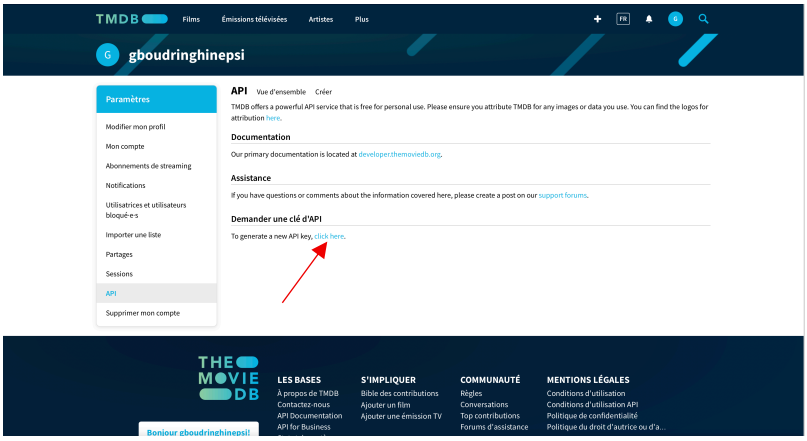
Pour l'application CinéTech, nous allons utiliser The Movie Database (TMDB) qui propose une API simple et gratuite.
Créer un compte développeur ici : https://www.themoviedb.org/signup

Informations :

- Utiliser l'adresse mail EPSI
- La validation du compte via un mail est obligatoire

Une fois connecté :

Étape 1 : Aller dans votre profil
Étape 2 : Cliquer sur "Demander une clé d'API"
Étape 3 : Indiquer que c'est un usage personnel
Étape 4 : Remplir le formulaire
Étape 5 : Récupérer la clé API et le token d'accès



Cette clé doit absolument être gardée secrète.

3. Connexion à l'API TMDB

Dans MovieList.jsx, remplaçons notre liste statique par une liste vide au départ :

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

Puis ajoutons l'appel API :

```
// …

export default function MovieList({ moviesList }) {
  const [movies, setMovies] = useState([]);
  const [moviesAsc, setMoviesAsc] = useState(true);

  useEffect(() => {
    fetch(

'https://api.themoviedb.org/3/movie/now_playing?api_key=YOUR_API_KEY&language=fr-FR&append_to_response=credits&page=1'
    )
      .then((response) => response.json())
      .then((data) => {
        console.log(data.results);
        setMovies(data.results) ;
      });
  }, []);

//…
```

Explications :

- « fetch() » récupère les données depuis l'API.
- « response.json() » transforme la réponse en objet JS.
- « data.results » contient la liste des films à l'affiche.
- « YOUR_API_KEY » est à remplacer par votre clé API.

Attention :
Le tableau [] est obligatoire, sinon le composant fera des appels API en boucle infinie.

Voici le lien isolé :

```
https://api.themoviedb.org/3/movie/now_playing?api_key=YOUR_API_KEY&language=fr-FR&append_to_response=credits&page=1
```

Il est possible d'utiliser d'autres points d'entrées de l'API. Pour cela, il faut se référer à la documentation.

La documentation de l'API est juste ici :
https://developer.themoviedb.org/reference/getting-started

Pourquoi notre page se casse ?

Parce que TMDB n'utilise pas le même format que notre tableau local.

Exemple de film renvoyé par TMDB :


```
{
    "adult": false,
    "backdrop_path": "/lZYMXx74pWmbj5Q5jp1QaMvmuuR.jpg",
    "genre_ids": [
      28,
      14,
      53
    ],
    "id": 1180831,
    "original_language": "no",
    "original_title": "Troll 2",
    "overview": "Lorsqu'un nouveau troll sème la dévastation dans le pays, Nora, Andreas et le major Kris se lancent dans la plus périlleuse des missions.",
    "popularity": 530.1827,
    "poster_path": "/5s33v65FEzVtt9X6G5EuHP9dbwx.jpg",
    "release_date": "2025-11-30",
    "title": "Troll 2",
    "video": false,
    "vote_average": 6.8,
    "vote_count": 202
}
```


Remarques :

- « poster » devient « poster_path »
- « year » devient « release_date »
- « director » n'existe pas dans cette réponse
- « rating » correspond à « vote_average »

Ces petites différences vont imposer quelques modifications dans notre affichage.

Travail à réaliser :

- Adapter « MovieCard » aux nouvelles données.
- Réadapter les filtres pour fonctionner avec TMDB.
- Modifier les sections si nécessaire.
- Vérifier que l'application fonctionne entièrement.

La minute Git :
Envoyer votre code sur votre répertoire Github.
Attention : ne jamais commit votre clé API.
Nom du commit : « TP7 : decouverte du useState – compteur »

4. Gérer l'état de l'API

Appeler une API apporte deux notions spécifiques :

- Le temps d'appel à l'API qui génère une certaine latence.
- Les erreurs lié à l'API qui doivent être traitée pour ne pas planter l'application.

Nous devons donc gérer deux nouveaux états : un état de chargement et un état d'erreur.

Ajoutons-les dans « MovieList » :

```
const [isLoading, setLoading] = useState(true);
const [error, setError] = useState(null);
```

Ajuster le « useEffect() » pour prendre en compte les deux états :

```
// …

useEffect(() => {
  fetch("URL_API")
    .then((res) => {
      if (!res.ok) throw new Error("Erreur réseau");
      return res.json();
    })
    .then((data) => {
      setMovies(data.results);
    })
    .catch((err) => {
      setError(err.message);
    })
    .finally(() => {
      setLoading(false);
    });
}, []);

// …
```

Adapter ensuite l'affichage :

```
// …

const IconMoviesByDate = moviesAsc
   ? FaArrowDownWideShort
   : FaArrowUpWideShort;

  if (isLoading) return <p>Chargement en cours</p>;
  if (error) return <p>{erreur}</p>;

  return (
    // …
  )
}
```

Travail à réaliser :

- Styliser les messages d'erreur et de chargement avec Tailwind.

La minute Git :
Envoyer votre code sur votre répertoire Github.
Attention : ne jamais commit votre clé API.
Nom du commit : « TP7 : api latence & erreur »

5. Persister dans le localstorage

Pour l'instant, dès que l'on actualise CinéTech, tout repart de zéro : les filtres reviennent à leur état initial et les favoris disparaissent. C'est logique puisque notre application ne conserve rien entre deux sessions. Pourtant, il serait très utile que CinéTech se souvienne de certains choix de l'utilisateur.

Le navigateur fournit un mécanisme simple pour conserver des données : le localStorage.
En le combinant avec useEffect, il devient extrêmement facile d'enregistrer une information et de la recharger plus tard.

Commençons par voir comment enregistrer une valeur.
Si nous souhaitons mémoriser la valeur d'un compteur, il suffit d'utiliser un useEffect qui s'exécute à chaque mise à jour de la variable :

```
useEffect(() => {
   localStorage.setItem("countValue", count);
  }, [count]);
```

Explication :

- « localStorage.setItem("countValue", count) » écrit la valeur actuelle dans le stockage du navigateur.
- Le « useEffect » se déclenche automatiquement dès que count change grâce au tableau de dépendances contenant uniquement cette variable.

Pour lire une valeur enregistrée, nous pouvons utiliser un autre useEffect :

```
useEffect(() => {
   const saved = localStorage.getItem("countValue");
   console.log("Valeur sauvegardée :", saved);
 }, []);
```

Explications :

- « localStorage.getItem("countValue") » récupère la valeur courante stocké dans le navigateur.
- Le « useEffect » n'est exécuté qu'une seule fois, car son tableau de dépendances est vide « [] ».

Travail à réaliser :

- Persister la liste des films favoris dans le localStorage.
- Afficher dynamiquement cette liste en la récupérant dans le localstorage.

La minute Git :
Envoyer votre code sur votre répertoire Github.
Attention : ne jamais commit votre clé API.
Nom du commit : « TP7 : favoris localstorage »

6. Challenge final du TP

Pour terminer ce TP, nous allons enrichir CinéTech en exploitant pleinement ce que nous avons appris : les appels API, la gestion du state, les effets, et l'adaptation de l'interface à partir de données externes.

L'objectif est d'ajouter de nouvelles fonctionnalités inspirées d'une vraie plateforme de filmographie.

Travail à réaliser :

- Ajouter la possibilité de lister des séries parmi les films.
- Afficher la ou les catégories du film sur sa carte.
- Indiquer si c'est la carte est un film ou une série grâce à une étiquette coloré.
- Ajouter une nouvelle section « Prochaines sorties » qui affiche les 8 prochains films/série prêt à sortir.

Dans le prochain TP, nous allons franchir une étape importante : CinéTech deviendra une application multi-pages.
Nous introduirons alors la navigation grâce à React Router, ce qui permettra d'organiser l'application autour d'un véritable parcours utilisateur.