Construction de la première interface CinéTech.

Notre environnement de développement est désormais opérationnel : Node, Vite, VS Code et la structure du projet sont prêts.

Il est temps de passer à l'étape la plus concrète : commencer à construire l'interface de CinéTech, l'application qui nous accompagnera tout au long du module.

Dans ce TP, nous allons découvrir comment React affiche une interface dans la page, et apprendre à exploiter la syntaxe JSX pour structurer visuellement notre application.

Ce travail constitue la première pierre visuelle de CinéTech.

À la fin de ce TP, votre application aura enfin une forme : un titre, des sections, et les premiers films affichés à l'écran.

1. Comprendre la hiérarchie de React

Dans le chapitre précédent nous avons abordés la structure de base et les différents fichiers qui composent la base d'un projet React.
MP

Comme pour toute application web classique, le point de départ est le fichier « index.html ». Ouvrons-le :

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>CinéTech</title>
  </head>
  <body>
    <div id="root"></div>

    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

Explications :
Deux lignes sont particulièrement importantes :

- `<div id="root"></div>` : cette balise représente une zone vide, identifiée par root. C'est à l'intérieur de cet élément que toute l'interface de CinéTech sera affichée.
- `<script type="module" src="/src/main.jsx"></script>` : cette balise charge un fichier JavaScript (`/src/main.jsx`). Même si l'extension « .jsx » est spécifique, on peut pour l'instant le considérer comme du JavaScript.

Intéressons-nous maintenant au fichier `src/main.jsx` :

```
import { createRoot } from 'react-dom/client';

createRoot(document.getElementById('root')).render(
  <h1>CinéTech — Projet prêt</h1>
);
```

Explications :
Ce fichier joue un rôle de “chef d’orchestre” entre le HTML et le code React.

- il récupère l’élément HTML #root dans index.html grâce à document.getElementById('root')
- il appelle la méthode « render() » pour dessiner un contenu dans cet élément (ici : un titre <h1>CinéTech — Projet prêt</h1>).

En résumé, l’élément #root est une zone vide dans la page, et main.jsx se charge d’y injecter l’interface.
Le fichier main.jsx est donc le point d’entrée JavaScript de CinéTech : c’est à partir de lui que tout le reste sera chargé.

Grâce à ce principe, et avec l’aide du bundler Vite, il sera possible de répartir le code dans plusieurs fichiers sans perdre cette hiérarchie : au final, tout converge vers main.jsx, puis vers index.html.

```
index.html → #root → main.jsx → interface.
```

Dans la partie suivante, l’objectif sera de déplacer la logique d’affichage dans un fichier dédié pour mieux organiser le code.

2. Isoler l’interface principale

Pour l’instant, main.jsx affiche directement un titre :

```
createRoot(document.getElementById('root')).render(
  <h1>CinéTech — Projet prêt</h1>
);
```

Ce fonctionnement est suffisant pour un test rapide, mais il devient vite limité si l’interface grandit.
Une meilleure organisation consiste à laisser main.jsx gérer uniquement le démarrage, et déléguer l’affichage à une autre partie du code.

L’idée est la suivante :

- main.jsx : point d’entrée, configuration minimale ;
- App.jsx : point d’entrée visuel, qui décrit ce que CinéTech doit afficher.

Travail à réaliser :

- Créer un nouveau fichier src/App.jsx.
- Y placer le code suivant :

```
export default function App() {
  return (
    <h1>CinéTech</h1>
  );
}
```

Explications :

- le fichier déclare la fonction JavaScript App().
- la fonction retourne un titre HTML.
- export default permet de rendre cette fonction accessible depuis d'autres fichiers.

On peut maintenant demander à main.jsx d'afficher le résultat de App().

Travail à réaliser :

- Modifier main.jsx comme suit :

```
import { createRoot } from 'react-dom/client';
import App from './App.jsx';

createRoot(document.getElementById('root')).render(
  App()
);
```

- Vérifier que la page affiche toujours le titre CinéTech.

La fonction App() joue désormais le rôle de point d'entrée pour l'affichage : c'est elle qui décidera de la structure de l'interface de CinéTech.

Dans la partie suivante, nous allons nous intéresser de plus près à cette ligne mystérieuse qui est passée inaperçu.

3. Du "HTML" qui n'en est pas : introduction à JSX

Reprenons le contenu de App.jsx :

```
export default function App() {
  return <h1>CinéTech</h1>;
}
```

L'instruction surlignée ressemble clairement à du HTM. Pourtant elle ne peut pas être interprétée comme tel dans un fichier JavaScript classique.
Si ce code était copié tel quel dans un simple fichier .js, une erreur serait générée.

La différence vient de l'extension du fichier : App.jsx.
Ici, l'écriture <h1>CinéTech</h1> n'est pas du HTML, mais du JSX.

JSX est une extension de syntaxe de JavaScript utilisée par React.
Elle permet de décrire l'interface de manière plus expressive, ressemblant fortement à du HTML, tout en restant dans un contexte JavaScript.

JSX peut être utilisé de deux manières :

- avec les balises usuelles que nous connaissons déjà en HTML (<h1>, <p>, , etc.) ;
- avec nos propres fonctions, écrites en JavaScript, que React interprète comme des balises personnalisées.

Par exemple :

```
function MaFonction() {
  return "Hello world !";
}

// Appel avec la syntaxe JSX :
<MaFonction />
```

Points à noter :

- la fonction utilisée ainsi doit respecter une notation en PascalCase (première lettre en majuscule) ;
- la "balise" doit être correctement fermée (<MaFonction />).

Nous reviendrons dessus dans un prochain TP car cette notation cache en réalité d'autres notions très intéressantes.

Mais de manière à tester, nous pouvons considérer la fonction App() éligible à cette syntaxe JSX.

Travail à réaliser :

- Dans main.jsx, modifier le code pour appeler la fonction App() grâce à la syntaxe JSX.
- Vérifier que le programme fonctionne toujours.

La minute Git :
Afin de versionner progressivement l'application, vous réaliserez les actions nécessaire pour envoyer votre code sur votre répertoire Github.
Nom du commit : « TP3 : structure de base & jsx »

Liste des balises JSX courantes		
Catégorie	Balises	Rôle principal
Structure globale	, <main>, <section>, <article>, <header>, <footer>, <nav>	Structurer les grandes zones de la page
Titres	<h1>, <h2>, <h3>, <h4>, <h5>, <h6>	Hiérarchiser le contenu
Texte	<p>, , ,	Gérer les blocs et éléments de texte
Listes	, ,	Créer des listes à puces ou numérotées
Liens & boutons	<a>, <button>	Créer des listes à puces ou numérotées
Médias	<img />, <video>, <audio>	Afficher des images, vidéos, sons
Formulaires	<form>, <label>, <input />, <textarea>, <select>, <option>	Entrer ou sélectionner des données
Tableaux	<table>, <thead>, <tbody>, <tr>, <td>, <th>	Organiser des données tabulaires
Divers	<hr />, 	Séparations et retours à la ligne

Dans la partie suivante, l'objectif sera d'exploiter cette syntaxe JSX pour afficher une première vraie structure d'interface : la liste des films de CinéTech.

4. Afficher la première liste de films

L'environnement de base est en place, App() affiche un titre, et JSX est opérationnel.
Il est temps d'utiliser tout cela pour afficher quelque chose de plus concret : la liste des films.

Pour apprendre à manipuler JSX avec des données structurées, nous allons simuler la liste des films dans un fichier local. Nous utiliserons une véritable API plus tard au cours d'un prochain TP.

⇨ La liste des films est à retrouver dans le fichier annexe.

Travail à réaliser :

- Créer un nouveau fichier « src/data/moviesList.js ».
- Déclarer une constante « moviesList » contenant un tableau d'objets des films.
- Coller la liste fournie dans cette constante.
- Exporter la constante par défaut.

Nous pouvons maintenant importer la liste des films dans « App.jsx » :

```
import moviesList from './data/moviesList.js';

//…
```

Vérifions ensuite que les données sont bien chargées :

```
export default function App() {
  console.log(moviesList);

  return <h1>CinéTech</h1>;
}
```

Dans la console du navigateur, la liste des films devrait s'afficher.

Travail à réaliser :

- Vérifier que la liste s'affiche correctement avant de continuer.

Nous pouvons maintenant utiliser JSX et les fonctionnalités de JavaScript avancée pour afficher les différents films.

```
// …

export default function App() {
  console.log(moviesList);

  return (
    <h1>CinéTech</h1>

    <main>
      <h2>Films à l'affiche</h2>

      <ul>
        {moviesList.map((movie) => (
          <li key={movie.id}>
            <h3>{movie.title}</h3>
            <p>
              {movie.year} — {movie.director}
            </p>
            <p>Note : {movie.rating}</p>
          </li>
        ))}
      </ul>
    </main>
```

```
  );
}
```

Explications :

- L'instruction {moviesList.map((movie) => (...))} permet de parcourir le tableau moviesList et de générer un <li> par film
- Les accolades {} permettent d'insérer des expressions JavaScript dans le rendu ({movie.title}, {movie.year}, etc.), c'est l'interpolation.
- « key={movie.id} » est un attribut qui doit être obligatoirement inclus lors de la création d'une liste d'éléments.

A ce stade notre code est complètement cassé et l'éditeur de code nous affiche une erreur assez explicite mais qui mérite quelques précisions.

```
JSX expressions must have one parent element.
```

Cette erreur signifie que le JSX retourné par la fonction contient plusieurs éléments racine. C'est-à-dire qu'une fonction ne doit retourner qu'un seul élément parent.

En analysant notre code, nous pouvons facilement remarquer qu'il y a deux nœuds JSX au même niveau :

- Le titre « h1 » contenant « Films à l'affiche ».
- L'élément « main » qui contient la liste des films.

Deux solutions possibles :

- Utiliser un élément classique tel qu'une « div » pour entourer tout le code. Cette solution affichera la « div » dans l'arborescence HTML final.
- Utiliser un fragment qui agit comme un conteneur invisible. Il permet d'avoir un seul parent JSX sans ajouter de balise supplémentaire dans le HTML final.

Pour insérer un fragment la syntaxe est relativement simple :

```
<>
   // Code JSX
</>
```

Travail à réaliser :

- Corriger l'erreur de parent unique selon la méthode qui paraît la plus adaptée.
- Vérifier que la liste des films s'affiche correctement dans la page.

La minute Git :

Envoyer votre code sur votre répertoire Github.

Nom du commit : « TP3 : liste des films »

La structure de base de CinéTech commence à prendre forme.
Dans la prochaine partie nous allons pratiquer un petit peu et continuer la structuration du contenu de notre application en lui ajoutant quelques possibilités.

5. Améliorations de l'interface

L'objectif va être d'apporter quelques autres améliorations à l'application CinéTech afin de la rendre plus complète.

La première amélioration consiste à afficher la liste des films « coups de cœur ».

Travail à réaliser :

- Ajouter une nouvelle section « Coups de cœur » sous la section « Films à l'affiche ».
- Afficher uniquement les films dont la note est supérieure ou égale à 8.
- Vérifier que seule une partie des films s'affiche.

La minute Git :

Envoyer votre code sur votre répertoire Github.

Nom du commit : « TP3 : films coups de cœur »

La deuxième amélioration consiste à créer une nouvelle section ayant le critère super-héros.

Travail à réaliser :

- Ajouter une nouvelle section « Films de super-héros » sous la section « Coups de cœur »
- Afficher les films selon un critère permettant de déterminer sa catégorie « super-héros »
- Vérifier que seule une partie des films s'affiche correctement.

La minute Git :

Envoyer votre code sur votre répertoire Github.

Nom du commit : « TP3 : films de super-heros »

6. Conclusion du TP

CinéTech dispose désormais d'une première ébauche de page :
Un titre, des sections, une liste de films, et même des sélections spécifiques ("Coups de cœur", "Films de super-héros").

Pour l'instant, tout cela reste brut : aucun style, aucune mise en forme visuelle.
Le prochain TP sera donc entièrement consacré à la mise en forme CSS de cette interface