Structurer avec les composants.

Jusqu'ici, nous avons privilégié un découpage logique de notre application en séparant les fichiers selon leur rôle.

Cette approche nous a permis d'avancer progressivement. Cependant, avec le TP4, nous avons ajouté beaucoup d'éléments JSX dans un même fichier. La structure devient alors difficile à lire et de plus en plus compliquée à maintenir.

C'est un problème classique en React : si nous laissons grandir l'interface dans un seul fichier, l'application devient illisible.

Il est donc temps de découvrir une notion centrale de React : les composants.
Les composants permettent de découper l'interface en petites briques autonomes et réutilisables.

1. Les composants fonctionnels

Nous avons en réalité déjà manipulé un composant sans l'avoir nommé comme tel.
En effet, le fichier App.jsx est considéré par React comme un composant fonctionnel.

Ouvrons à nouveau ce fichier :

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

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

  return (
    // …
      <h1 className="text-4xl font-bold text-center my-6">CinéTech</h1>

      <main className="max-w-7xl mx-auto mt-6">
        // …
      </main>
    // …
  );
}
```

Explications :

- Le fichier contient une fonction JavaScript App().
- La fonction retourne du JSX.
- Pour l'utiliser ailleurs, nous pouvons écrire <App />.

Définition :
Un composant n'est rien d'autre qu'une fonction qui retourne du JSX.

Nous savons maintenant qu'un composant est simplement une fonction qui retourne une portion d'interface. Passons à la création de notre première "brique" visuelle.

2. Un premier composant : MovieList

Nous allons créer un composant dédié à l'affichage d'une liste de films : MovieList.jsx.

Créons le fichier src/components/MovieList.jsx puis insérons ce code :

```
export default function MovieList() {
  return (
    <section>
      <h2 className="text-2xl font-bold mb-4">Films à l'affiche</h2>
```

```
    <p>Ma liste de film à l'affiche</p>
   </section>
  );
}
```

Explications :

- MovieList() est une fonction classique qui retourne du JSX.
- Le dossier components contiendra l'ensemble des composants visuels de notre application.

Utilisons ce nouveau composant :

```
import MovieList from './components/MovieList.jsx';
import moviesList from './data/moviesList.js';

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

 return (
  <>
   <h1 className="text-4xl font-bold text-center my-6">CinéTech</h1>

   <main className="max-w-7xl mx-auto mt-6">
    // …

    <MovieList />

    <section>
     <h2 className="text-2xl font-bold mb-4">Films à l'affiche</h2>

     <ul className="grid grid-cols-1 sm:grid-cols-5 gap-6">
      // …
     </ul>
    </section>

    // …
   </main>
  </>
 );
}
```

Une nouvelle section apparaît.

Nous allons copier la logique autour du « map() » utilisé précédemment et le coller dans « MovieList.jsx », puis nous pouvons nettoyer App.jsx afin de ne plus faire apparaître les lignes déplacées dans notre composant.

Résultat attendu pour « MovieList.jsx » :

```
export default function MovieList() {
  return (
    <section>
      <h2 className="text-2xl font-bold mb-4">Films à l'affiche</h2>

      <ul className="grid grid-cols-1 sm:grid-cols-5 gap-6">
        {moviesList.map((movie) => (
          <li key={movie.id} className="bg-gray-100 rounded-xl p-4 shadow">
            <h3 className="text-xl font-semibold">{movie.title}</h3>
            <img
              src={movie.poster}
              alt={movie.title}
              className="w-full h-64 object-cover rounded"
            />
            <p className="text-sm text-gray-600 mt-2">
              {movie.year} — {movie.director}
            </p>
            <p className="mt-2">Note : {movie.rating}</p>
          </li>
        ))}
      </ul>
    </section>
  );
}
```

Rien ne s'affiche.
Problème : moviesList n'est pas défini dans ce composant.

Nous pourrions importer moviesList ici, mais ce serait dupliquer la source de données et perdre le contrôle.

Nous devons maintenant apprendre à faire circuler l'information entre les composants et c'est ce que nous allons faire dans la prochaine partie.

3. Découvertes des properties

Un composant ne connaît pas les variables définies ailleurs. Il doit recevoir les informations dont il a besoin.

C'est précisément le rôle des props (properties) : transmettre des données depuis un composant parent vers un composant enfant.

Étape 1 : Indiquer que « MovieList » recevra des données

```
export default function MovieList({ movies }) {
  return (
    <section>
      <h2 className="text-2xl font-bold mb-4">Films à l'affiche</h2>

      <ul className="grid grid-cols-1 sm:grid-cols-5 gap-6">
        {movies.map((movie) => (
          // …
        ))}
      </ul>
    </section>
  );
}
```

Explications :

- Les accolades « { } » permettent de déstructurer l'objet de props.
- Nous remplaçons « moviesList » par « movies ».

Étape 2 : Envoyer la liste depuis App.jsx

```
import MovieList from './components/MovieList.jsx';
import moviesList from './data/moviesList.js';

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

  return (
    <>
      <h1 className="text-4xl font-bold text-center my-6">CinéTech</h1>

      <main className="max-w-7xl mx-auto mt-6">
        <MovieList movies={moviesList} />
      </main>
    </>
  );
}
```

La liste s'affiche à nouveau.

La minute Git :
Envoyer votre code sur votre répertoire Github.
Nom du commit : « TP5 : composant MovieList »

Nous savons maintenant faire circuler des données depuis un parent vers un enfant. Passons à l'étape suivante : isoler l'affichage d'une carte.

4. Séparer le style et la structure

Actuellement, MovieList contient un long bloc JSX pour chaque film.
C'est lisible pour 3 films, mais compliqué pour 50.

A vous de jouer !

Travail à réaliser :

- Créer un nouveau composant « MovieCard »
  - Il doit recevoir un film en paramètre
  - Il doit afficher les mêmes informations (titre, photo, année, réalisateur, note)
  - Il doit retourner un seul élément « li »
- Adapter le composant « MovieList » pour faire en sorte d'utiliser le nouveau composant.

Attention !
N'adaptez pas les autres sections que vous avez précédemment créée, nous devons avant tout aborder une autre notion.

La minute Git :
Envoyer votre code sur votre répertoire Github.
Nom du commit : « TP5 : composant MovieCard »

5. Le composant Section

Prenons un autre cas un peu plus spécifique.

Nous répétons à plusieurs reprise une structure de code :

```
<section>
    <h2 className="text-2xl font-bold mb-4">Film à l'affiche</h2>

   // Affichage de la liste des films selon un critère.
   // Exemple : MovieList
</section>
```

Au lieu de répéter ce bloc, nous allons créer un composant plus générique que nous pourrons réutiliser pour nos prochaines section.

Le nouveau composant « src/components/Section.jsx » :

```
export default function Section({ title }) {
  return (
    <section className="my-10">
      <h2 className="text-2xl font-bold mb-4">{title}</h2>
    </section>
  );
}
```

Intégrons maintenant le composant « Section » dans « MovieList.jsx » :

```
import MovieCard from './MovieCard';
import Section from './Section';

export default function MovieList({ movies }) {
  return (
    <Section title={"Films à l'affiche"}>
      <ul className="grid grid-cols-1 sm:grid-cols-5 gap-6">
        {movies.map((movie) => (
          <MovieCard movie={movie} />
        ))}
      </ul>
    </Section>
  );
}
```

L'affichage du titre est bien effective, cependant la liste des films n'est plus affichée.
Le composant n'est en réalité pas capable d'interpréter nativement les éléments JSX que nous lui passons entre deux balises.

Exemple :

```
<Composant>
  <p>Je suis du JSX</p>
</Composant>
```

Ce composant ne peut pas afficher le paragraphe.
Pour que le composant puisse afficher, nous devons lui indiquer grâce au concept du « children ».

Définition :
« Children » est une « props » spéciale, automatiquement fournie par React, qui représente tout le contenu placé entre les deux balises du composant.

Dans notre fichier « Section.jsx » nous devons simplement ajuster le code :

```
export default function Section({ title, children }) {
  return (
    <section className="my-10">
      <h2 className="text-2xl font-bold mb-4">{title}</h2>

      {children}
    </section>
  );
}
```

Grâce à cette approche, nous centralisons la logique dans un composant unique : il devient alors beaucoup plus simple d'ajuster la structure de l'interface, sans avoir à modifier le code dans plusieurs fichiers séparés.

La minute Git :
Envoyer votre code sur votre répertoire Github.
Nom du commit : « TP5 : composant Section »

Nous savons maintenant composer l'interface en regroupant structure et contenu dans des composants réutilisables. L'application devient lisible, modulaire et beaucoup plus facile à faire évoluer.

Travail à réaliser :

- Adapter le code pour toutes les sections créées.
- Permettre au composant MovieList d'afficher des films selon un critère configurable grâce à un props.

La minute Git :
Envoyer votre code sur votre répertoire Github.
Nom du commit : « TP5 : tout en composant »

Dans le prochain TP, nous allons apprendre à rendre notre application plus interactive et dynamique.