Rendre l'interface interactive.

Jusqu'ici, CinéTech affiche une interface complète et structurée.
Cependant, l'application reste totalement statique. En effet, elle ne réagit pas aux actions de l'utilisateur.

Pour aller plus loin, nous devons introduire une notion essentielle de React : le state ou en français l'état d'un composant.

Dans ce TP, nous allons aborder un concept propre à React qui permet de rendre une interface dynamique. Nous allons découvrir la gestion de l'état d'un composant et rendre CinéTech plus interactif.

1. Introduction aux hooks

Les hooks font partie du cœur de la logique de React.
C'est un choix fondamental qui le différencie d'autres bibliothèques.

Il existe de nombreux hooks intégrés à React et il est également possible de créer ses propres hooks. C'est d'ailleurs très courant dans des projets plus avancés afin d'éviter la répétition de logique et de centraliser des comportements réutilisables.

L'un des premiers hooks que nous allons utiliser est le useState().

useState() permet de gérer un état dynamique dans un composant.
C'est lui qui peut indiquer à React de re-rendre l'interface sans recharger la page.

Remarque :
Tous les hooks commencent par le mot-clé use, c'est une règle importante pour React.

Exemple avec un compteur interactif.

Créons d'abord un nouveau composant Counter puis importons le hook :

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

Dans le composant :

```
export default function Counter() {
  const [count, setCount] = useState(0);

  const increment = () => {
    setCount(count + 1);
  };

  return (
    <div className="bg-gray-100 rounded-xl p-4 shadow">
      <p>Compteur : {count}</p>
      <button
        className="bg-blue-500 hover:bg-blue-700 text-white font-bold px-2 py-1 rounded"
        onClick={increment}
      >
        +1
      </button>
    </div>
  );
}
```

Explications :

- « useState(0) » initialise l'état d'un composant, en l'occurrence ici « 0 ».
- La déstructuration permet de récupérer :
  - « count » la valeur actuelle.
  - « setCount » une fonction permettant de mettre à jour la valeur.
- L'attribut onClick permet de déclencher une action lors d'un clic.

Origine	Évènement React	Équivalent HTML
Souris	onClick, onDoubleClick, onContextMenu	onclick
Clavier	onKeyDown, onKeyUp	onkeydown
Formulaire	onChange, onSubmit, onFocus, onBlur	onchange
Média	onLoad, onError	onload
Fenêtre	onResize, onScroll	onresize
Drag & Drop	onDrag, onDrop	ondrag

Travail à réaliser :

- Tester le composant dans le navigateur
- Observer le fonctionnement du bouton.
- Implémenter un bouton qui permet de décrémenter le compteur.

La minute Git :
Envoyer votre code sur votre répertoire Github.
Nom du commit : « TP6 : decrementation du compteur »

Il est facile de voir qu'un compteur, en JavaScript natif, est plus verbeux.
Avec React, l'écriture devient beaucoup plus expressive et maintenable.

Un peu de pratique pour se faire la main.

Travail à réaliser :

- Ajouter un bouton rouge qui décrémente de -1.
- Permettre de configurer la valeur initiale du compteur.
- Ajouter la possibilité de personnaliser le pas des incrémentations et des décrémentation. Exemple : un bouton +5 et un bouton -5
- Créer un bouton qui réinitialiser le compteur.

La minute Git :
Envoyer votre code sur votre répertoire Github.
Nom du commit : « TP6 : decouverte du useState – compteur »

Nous savons maintenant modifier une valeur à l'écran sans recharger la page.
Passons à quelque chose de plus utile : manipuler la liste des films.

⇨ Retirer l'appel du composant Counter une fois terminé pour ne pas perturber l'affichage des films.

2. Trier les film grâce à la gestion d'état.

Nous allons ajouter un bouton permettant de trier les films par année (croissant / décroissant).

Déclarons un état dans MovieList.jsx :

```
//…

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

// …
```

Explications :

- Nous partons de la donnée reçu par le parent dans « moviesList ».
- Nous copions dans un nouvel état « movies ».
- « setMovies() » permettra de modifier l'affichage sans toucher aux données originales.

Dynamisons l'affichage :

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

  const sortMoviesByDate = () => {
   setMovies([...movies].sort((a, b) =>
     b.year - a.year
   ));
  };

  return (
    <Section title={"Films à l'affiche"}>
      <div className="flex bg-gray-100 rounded-xl p-4 shadow sm:p-6 border border-
white/10">
       <button
         className="ml-auto bg-blue-500 hover:bg-blue-700 text-white font-bold px-2 py-1
rounded"
         onClick={sortMoviesByDate}
```

```
      >
        Tri par année
      </button>
    </div>

    <ul className="grid grid-cols-1 sm:grid-cols-5 gap-6 mt-4">
      {movies.map((movie) => (
        <MovieCard key={movie.id} movie={movie} />
      ))}
    </ul>
  </Section>
 );
}
```

Explications :

- Nous avons ajouté une « div » qui affiche une barre grise et qui contiendra l'ensemble des élément pour filtrer et trier les films.
- Le bouton déclenche la fonction « sortMovies » lorsqu'il est cliqué.
- Plusieurs choses sur la fonction « sortMovies »
  - « […] » permet de copier le tableau de film sans le modifier.
  - « .sort() » permet de trier les films.
  - « b.year - a.year » permet un ordre croissant.

Notre triage s'effectue correctement cependant une fois que le bouton est cliqué, il n'est plus possible de changer le tri.

Ajoutons la possibilité d'afficher les films par ordre de date décroissante :

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

 const sortMoviesByDate = () => {
  const sorted = [...movies].sort((a, b) =>
   moviesAsc ? a.year - b.year : b.year - a.year
  );

  setMovies(sorted);
  setMoviesAsc(!moviesAsc);
 };

 return (
  <Section title={"Films à l'affiche"}>
   <div className="flex bg-gray-100 rounded-xl p-4 shadow sm:p-6 border border-
white/10">
```

```
      <button
        className="ml-auto bg-blue-500 hover:bg-blue-700 text-white font-bold px-2 py-1
rounded"
        onClick={sortMoviesByDate}
      >
        Tri par année
      </button>
    </div>

    <ul className="grid grid-cols-1 sm:grid-cols-5 gap-6 mt-4">
      {movies.map((movie) => (
        <MovieCard key={movie.id} movie={movie} />
      ))}
    </ul>
  </Section>
 );
}
```

Explications :

- Nous devons ajouter un état supplémentaire pour savoir si le tri se faire par date croissante ou décroissante. Si la valeur est « true » l'affichage est croissant si c'est « false » c'est décroissant.
- La deuxième modification est une condition ternaire qui permet d'afficher le calcul pour l'affichage croissant ou décroissant en fonction de l'état d'affichage « moviesAsc ».
- Enfin, les vues sont mises à jour en modifiant les états de la liste de film et de l'ordre de tri.

Nous pouvons maintenant trier notre liste de film. Le premier clique permet d'afficher les films par ordre de date croissante et le deuxième clique par ordre de date décroissante. L'opération peu se répéter à en boucle.

La minute Git :
Envoyer votre code sur votre répertoire Github.
Nom du commit : « TP6 : tri date croissante decroissante »

Malgré cela il reste un dernier soucis : la liste ne peut pas revenir à son état initiale.

Travail à réaliser

- Ajouter la possibilité de réinitialiser le tri (plusieurs façons de faire).

La minute Git :
Envoyer votre code sur votre répertoire Github.
Nom du commit : « TP6 : reinitialisation tri par date »

Nous savons maintenant utiliser le state pour modifier l'affichage en fonction d'une action. Dans la prochaine partie, nous allons découvrir comment il est possible de conditionner l'affichage.

3. Conditionner l'affichage grâce au state

Nous allons utiliser les états pour afficher des éléments visuels différents en fonction d'actions utilisateur.

Par exemple, essayons de changer l'icône du bouton selon l'ordre du tri.

Installons la bibliothèque d'icone « React Icons » grâce à la commande :

```
> npm install react-icons --save
```

Pour choisir notre icône nous pouvons nous rendre sur le site :
https://react-icons.github.io/react-icons/

Ici nous allons utiliser les icones de FontAwesome6.

Icône pour l'état croissant :

```
// Import
import { FaArrowUpWideShort } from "react-icons/fa6";

// Utilisation
<FaArrowUpWideShort />
```

Icône pour l'état décroissant :

```
// Import
import { FaArrowDownWideShort } from "react-icons/fa6";

// Utilisation
<FaArrowDownWideShort />
```

Modifions le rendu de notre composant comme ceci :

```
// …

return (
   <Section title={"Films à l'affiche"}>
     <div className="flex bg-gray-100 rounded-xl p-4 shadow sm:p-6 border border-
white/10">
       <button
```

```
        className="ml-auto flex items-center gap-1 bg-blue-500 hover:bg-blue-700 text-
white font-bold px-2 py-1 rounded"
        onClick={sortMoviesByDate}
      >
        Tri par année
        {moviesAsc ? <FaArrowDownWideShort /> : <FaArrowUpWideShort />}
      </button>
    </div>

    <ul className="grid grid-cols-1 sm:grid-cols-5 gap-6 mt-4">
      {movies.map((movie) => (
        <MovieCard key={movie.id} movie={movie} />
      ))}
    </ul>
  </Section>
 );
```

Explications :

- La première modification permet simplement d'ajouter les classes CSS nécessaires pour afficher correctement les icônes.
- La deuxième modification est une condition ternaire qui permet d'afficher l'icône cohérente en fonction de l'état « moviesAsc ».

Cette écriture parait correcte dans le fond mais sur la forme ce n'est pas forcément recommandé. Sur le long terme la visibilité devient plus compliquée. Nous pouvons simplifier cela grâce à un micro composant :

```
const IconMoviesByDate = moviesAsc ? FaArrowDownWideShort : FaArrowUpWideShort;
```

Ce qui nous permet d'obtenir le résultat suivant :

```
// …

const IconMoviesByDate = moviesAsc
   ? FaArrowDownWideShort
   : FaArrowUpWideShort;

 return (
  <Section title={"Films à l'affiche"}>
   <div className="flex bg-gray-100 rounded-xl p-4 shadow sm:p-6 border border-
white/10">
     <button
       className="ml-auto flex items-center gap-1 bg-blue-500 hover:bg-blue-700 text-
white font-bold px-2 py-1 rounded"
       onClick={sortMoviesByDate}
```

```
      >
        Tri par année
        <IconMoviesByDate />
      </button>
    </div>

    <ul className="grid grid-cols-1 sm:grid-cols-5 gap-6 mt-4">
      {movies.map((movie) => (
        <MovieCard key={movie.id} movie={movie} />
      ))}
    </ul>
  </Section>
);
```

Cela est clairement plus lisible et limite l'utilisation d'expression JavaScript dans le rendu où seul le JSX doit être présent.

La minute Git :
Envoyer votre code sur votre répertoire Github.
Nom du commit : « TP6 : icone tri date croissante decroissante »

Travail à réaliser :

- Ajouter une icône pour l'état "sans tri".
- Ajouter le tri par ordre alphabétique du titre
- Ajouter le tri par note croissante / décroissante

La minute Git :
Envoyer votre code sur votre répertoire Github.
Nom du commit : « TP6 : tri avancé »

4. Challenge final du TP

L'objectif va être de créer un système simple d'ajout de film en favoris.

Travail à réaliser :

- Ajouter un bouton (icône cœur) sur les films.
- Ajouter / retirer des films en favoris.
- Ajouter un style spécifique pour les films favoris.
- Afficher un compteur de favoris.
- BONUS : Ajouter une section qui affiche la liste des films en favoris. Si aucun film est en favoris afficher le message « Aucun film en favori pour le moment. »

La minute Git :
Envoyer votre code sur votre répertoire Github.
Nom du commit : « TP6 : favoris »

5. Conclusion du TP

Nous avons introduit l'un des concepts majeurs de React : le state.
Il permet d'obtenir une interface réellement interactive et agréable à utiliser.

Dans le prochain TP, nous allons poursuivre la dynamique en découvrant les effets secondaires avec le hook useEffect().

Nous pourrons notamment communiquer avec une API et stocker des données dans le navigateur.