Exercice 7 - Custom Hook : useToggle

Objectif

Créer son premier hook personnalisé pour réutiliser de la logique.

Concepts abordés

- Création de custom hooks
- Convention de nommage (use...)
- Réutilisation de logique stateful
- Composition de hooks

Qu'est-ce qu'un Custom Hook ?

Un custom hook est une **fonction JavaScript** qui :

1. Commence par `use` (convention obligatoire)
2. Peut utiliser d'autres hooks (useState, useEffect, etc.)
3. Permet de **réutiliser de la logique** entre composants

```
// Ce n'est PAS un composant, c'est une fonction qui utilise des hooks
function useMonHook() {
  const [state, setState] = useState(...);

  // Logique réutilisable...

  return { state, ... };
}
```

Énoncé

Partie 1 : useToggle

Créer un hook `useToggle` qui :

- Gère un état booléen
- Retourne la valeur et des fonctions pour la manipuler

```
const [isOpen, toggle, setOpen, setClose] = useToggle(false);
```

Partie 2 : useLocalStorage

Créer un hook `useLocalStorage` qui :

- Fonctionne comme useState
- Persiste la valeur dans localStorage
- Initialise depuis localStorage si une valeur existe

Correction

useToggle

```
// hooks/useToggle.js
import { useState, useCallback } from 'react';

/**
 * Hook pour gérer un état booléen avec des helpers
 * @param {boolean} initialValue - Valeur initiale (false par défaut)
 * @returns {[boolean, function, function, function]}
 */
function useToggle(initialValue = false) {
  const [value, setValue] = useState(initialValue);

  // useCallback pour stabiliser les références des fonctions
  const toggle = useCallback(() => setValue(v => !v), []);
  const setTrue = useCallback(() => setValue(true), []);
  const setFalse = useCallback(() => setValue(false), []);

  return [value, toggle, setTrue, setFalse];
}

export default useToggle;
```

Utilisation de useToggle

```
import useToggle from './hooks/useToggle';

function Modal() {
  const [isOpen, toggleModal, openModal, closeModal] = useToggle(false);

  return (
    <div>
      <button onClick={openModal}>Ouvrir</button>

      {isOpen && (
        <div className="modal">
          <h2>Ma Modal</h2>
          <button onClick={closeModal}>Fermer</button>
        </div>
      )}
    </div>
  );
}
```

```
function DarkModeToggle() {
  const [isDark, toggleDark] = useToggle(false);

  return (
    <button onClick={toggleDark}>
      {isDark ? '🌙 Dark' : '☀️ Light'}
    </button>
  );
}
```

useLocalStorage

```
// hooks/useLocalStorage.js
import { useState, useEffect } from 'react';

/**
 * Hook qui synchronise un state avec localStorage
 * @param {string} key - Clé localStorage
 * @param {any} initialValue - Valeur par défaut si rien en localStorage
 */
function useLocalStorage(key, initialValue) {
  // Initialiser le state depuis localStorage
  const [value, setValue] = useState(() => {
    if (typeof window === 'undefined') {
      return initialValue;
    }

    try {
      const item = localStorage.getItem(key);
      return item ? JSON.parse(item) : initialValue;
    } catch (error) {
      console.warn(`Erreur lecture localStorage "${key}":`, error);
      return initialValue;
    }
  });

  // Synchroniser avec localStorage à chaque changement
  useEffect(() => {
    try {
      localStorage.setItem(key, JSON.stringify(value));
    } catch (error) {
      console.warn(`Erreur écriture localStorage "${key}":`, error);
    }
  }, [key, value]);

  return [value, setValue];
}

export default useLocalStorage;
```

Utilisation de useLocalStorage

```
import useLocalStorage from './hooks/useLocalStorage';

function ThemeSelector() {
  // Le thème sera sauvegardé et restauré automatiquement
  const [theme, setTheme] = useLocalStorage('theme', 'light');

  return (
    <select value={theme} onChange={e => setTheme(e.target.value)}>
      <option value="light">Clair</option>
      <option value="dark">Sombre</option>
      <option value="system">Système</option>
    </select>
  );
}

function UserPreferences() {
  const [prefs, setPrefs] = useLocalStorage('userPrefs', {
    notifications: true,
    language: 'fr'
  });

  const toggleNotifications = () => {
    setPrefs(prev => ({
      ...prev,
      notifications: !prev.notifications
    }));
  };

  return (
    <div>
      <label>
        <input
          type="checkbox"
          checked={prefs.notifications}
          onChange={toggleNotifications}
        />
        Activer les notifications
      </label>
    </div>
  );
}
```

Exercice bonus : useCounter

```
// hooks/useCounter.js
import { useState, useCallback } from 'react';
```

```jsx
function useCounter(initialValue = 0, { min = -Infinity, max = Infinity, step = 1
} = {}) {
  const [count, setCount] = useState(initialValue);

  const increment = useCallback(() => {
    setCount(c => Math.min(c + step, max));
  }, [step, max]);

  const decrement = useCallback(() => {
    setCount(c => Math.max(c - step, min));
  }, [step, min]);

  const reset = useCallback(() => {
    setCount(initialValue);
  }, [initialValue]);

  const set = useCallback((value) => {
    setCount(Math.min(Math.max(value, min), max));
  }, [min, max]);

  return { count, increment, decrement, reset, set };
}

// Utilisation
function QuantitySelector() {
  const { count, increment, decrement, reset } = useCounter(1, {
    min: 1,
    max: 10
  });

  return (
    <div>
      <button onClick={decrement}>-</button>
      <span>{count}</span>
      <button onClick={increment}>+</button>
      <button onClick={reset}>Reset</button>
    </div>
  );
}
```

Règles des Custom Hooks

1. **Nom commençant par `use`** - Obligatoire pour que React détecte les hooks
2. **Peuvent appeler d'autres hooks** - useState, useEffect, autres custom hooks
3. **Retournent ce qu'ils veulent** - Array, object, valeur simple...
4. **Chaque appel a son propre state** - Pas de partage entre composants