Exercice 9 - Context API : Thème global

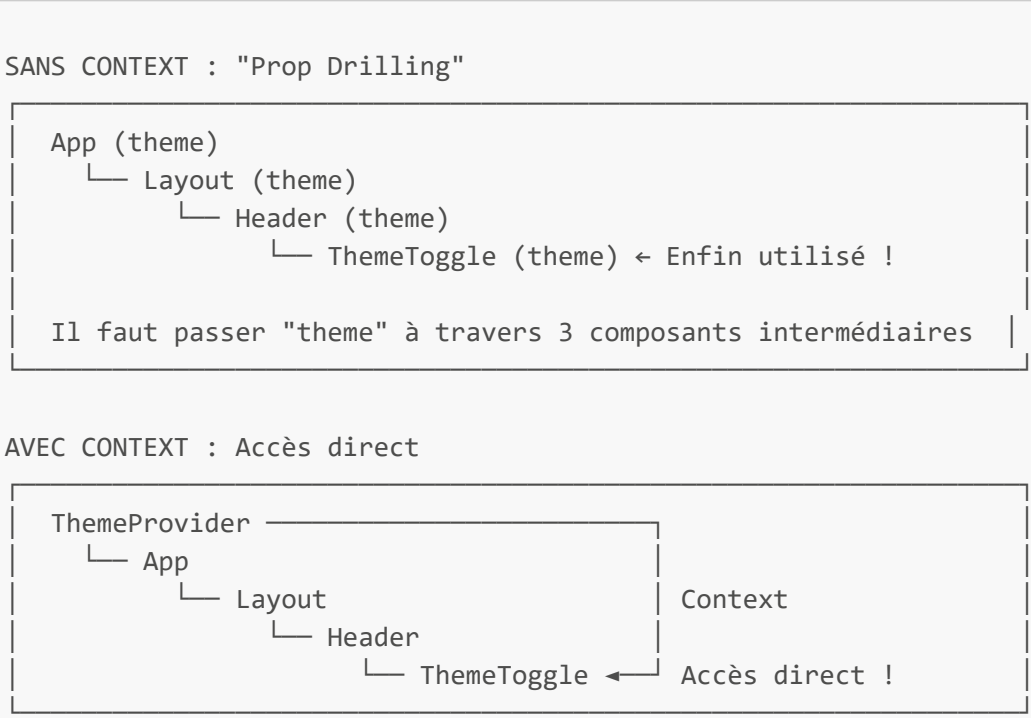
Objectif

Implémenter un système de thème clair/sombre avec Context API.

Concepts abordés

- createContext
- Context.Provider
- useContext
- Pattern Provider + Hook personnalisé

Problème résolu par Context



Énoncé

Créer un système de thème complet :

1. Un contexte `ThemeContext`
2. Un provider `ThemeProvider`
3. Un hook `useTheme` pour consommer le contexte
4. Persister le choix dans localStorage
5. Appliquer le thème via CSS (variables CSS)

Correction

1. ThemeContext.jsx

```jsx
import { createContext, useContext, useState, useEffect } from 'react';

// ═══════════════════════════════════════════════════════════════════
// ÉTAPE 1 : Créer le contexte
// La valeur par défaut est utilisée si pas de Provider parent
// ═══════════════════════════════════════════════════════════════════
const ThemeContext = createContext(null);

// ═══════════════════════════════════════════════════════════════════
// ÉTAPE 2 : Créer le Provider
// C'est un composant qui va envelopper l'application
// ═══════════════════════════════════════════════════════════════════
export function ThemeProvider({ children }) {
  // Initialiser depuis localStorage ou défaut 'light'
  const [theme, setTheme] = useState(() => {
    if (typeof window !== 'undefined') {
      return localStorage.getItem('theme') || 'light';
    }
    return 'light';
  });

  // Synchroniser avec localStorage et le DOM
  useEffect(() => {
    // Sauvegarder dans localStorage
    localStorage.setItem('theme', theme);

    // Appliquer au document (pour les CSS variables)
    document.documentElement.setAttribute('data-theme', theme);

    // Ou ajouter/retirer une classe
    document.body.classList.remove('light', 'dark');
    document.body.classList.add(theme);
  }, [theme]);

  // Fonctions helper
  function toggleTheme() {
    setTheme(prev => prev === 'light' ? 'dark' : 'light');
  }

  function setLightTheme() {
    setTheme('light');
  }

  function setDarkTheme() {
    setTheme('dark');
  }

  // Valeur fournie au contexte
```

```
  const value = {
    theme,
    isDark: theme === 'dark',
    isLight: theme === 'light',
    toggleTheme,
    setLightTheme,
    setDarkTheme,
    setTheme
  };

  return (
    <ThemeContext.Provider value={value}>
      {children}
    </ThemeContext.Provider>
  );
}

// ═══════════════════════════════════════════════════════════════════
// ÉTAPE 3 : Hook personnalisé pour consommer le contexte
// Avantages : vérifie que le Provider existe, plus simple à utiliser
// ═══════════════════════════════════════════════════════════════════
export function useTheme() {
  const context = useContext(ThemeContext);

  if (context === null) {
    throw new Error(
      'useTheme doit être utilisé à l\'intérieur d\'un <ThemeProvider>. ' +
      'Assurez-vous d\'envelopper votre application avec <ThemeProvider>.'
    );
  }

  return context;
}

export default ThemeContext;
```

2. Utilisation dans l'application

```
// main.jsx ou App.jsx
import { ThemeProvider } from './contexts/ThemeContext';
import App from './App';

// Envelopper l'application avec le Provider
ReactDOM.createRoot(document.getElementById('root')).render(
  <ThemeProvider>
    <App />
  </ThemeProvider>
);
```

3. Composant ThemeToggle

```jsx
import { useTheme } from './contexts/ThemeContext';
import './ThemeToggle.css';

function ThemeToggle() {
  const { theme, toggleTheme, isDark } = useTheme();

  return (
    <button
      onClick={toggleTheme}
      className={`theme-toggle ${theme}`}
      aria-label={`Passer en mode ${isDark ? 'clair' : 'sombre'}`}
    >
      <span className="icon">{isDark ? '🌙' : '☀️'}</span>
      <span className="label">{isDark ? 'Sombre' : 'Clair'}</span>
    </button>
  );
}

export default ThemeToggle;
```

4. Composant utilisant le thème

```jsx
import { useTheme } from './contexts/ThemeContext';

function Card({ title, children }) {
  const { isDark } = useTheme();

  return (
    <div className={`card ${isDark ? 'card-dark' : 'card-light'}`}>
      <h3>{title}</h3>
      {children}
    </div>
  );
}
```

5. CSS avec variables

```css
/* styles/theme.css */

/* Variables par défaut (thème clair) */
:root {
  --bg-primary: #ffffff;
  --bg-secondary: #f3f4f6;
  --text-primary: #1f2937;
  --text-secondary: #4b5563;
  --border-color: #e5e7eb;
  --shadow: 0 2px 8px rgba(0, 0, 0, 0.1);
}
```

```css
/* Variables thème sombre */
[data-theme="dark"] {
  --bg-primary: #1f2937;
  --bg-secondary: #111827;
  --text-primary: #f9fafb;
  --text-secondary: #d1d5db;
  --border-color: #374151;
  --shadow: 0 2px 8px rgba(0, 0, 0, 0.3);
}

/* Application des variables */
body {
  background-color: var(--bg-secondary);
  color: var(--text-primary);
  transition: background-color 0.3s, color 0.3s;
}

.card {
  background-color: var(--bg-primary);
  border: 1px solid var(--border-color);
  box-shadow: var(--shadow);
  padding: 1.5rem;
  border-radius: 8px;
}

/* ThemeToggle */
.theme-toggle {
  display: flex;
  align-items: center;
  gap: 0.5rem;
  padding: 0.5rem 1rem;
  border: 2px solid var(--border-color);
  background: var(--bg-primary);
  color: var(--text-primary);
  border-radius: 999px;
  cursor: pointer;
  transition: all 0.2s;
}

.theme-toggle:hover {
  border-color: var(--text-secondary);
}

.theme-toggle .icon {
  font-size: 1.25rem;
}

.theme-toggle .label {
  font-weight: 500;
}
```

Schéma du flux de données