

Introduction à React et Setup du Projet

Objectif

L'objectif de cette première semaine est de comprendre les bases de React, installer un projet fonctionnel et commencer à manipuler les composants.

1. Introduction à React

Qu'est-ce que React ?

React est une **bibliothèque JavaScript** développée par Facebook, utilisée pour construire des interfaces utilisateur interactives. Elle permet de créer des applications web dynamiques en utilisant des composants réutilisables.

Pourquoi utiliser React ?

- **Performances accrues** : React utilise un **Virtual DOM**, qui optimise les mises à jour de l'interface et améliore la rapidité de rendu.
- **Composants réutilisables et modulaires** : Le code est structuré en petits blocs indépendants, facilitant la maintenance et la réutilisation.
- **Facilite la gestion de l'interface utilisateur et de l'état** : Grâce à des outils comme les **Hooks** et **Redux**, il devient plus simple de gérer l'état de l'application.
- **Utilisation massive dans l'industrie** : De nombreuses entreprises (Facebook, Instagram, Airbnb, Netflix) utilisent React, ce qui en fait une compétence très recherchée.

React utilise un **Virtual DOM** (DOM virtuel), ce qui améliore les performances et optimise les mises à jour de l'interface utilisateur. Voici une explication détaillée avec un exemple :

◆ Qu'est-ce que le Virtual DOM ?

Le **DOM (Document Object Model)** représente la structure HTML d'une page web sous forme d'arborescence. Lorsqu'un changement se produit dans une page, le navigateur met à jour le DOM, ce qui peut être coûteux en performances.

Le **Virtual DOM** est une copie légère du DOM réel, gérée par React. Lorsqu'un état change, React met à jour uniquement les éléments nécessaires dans le DOM réel au lieu de le reconstruire entièrement.

◆ Comment fonctionne le Virtual DOM ?

1. Lorsqu'un état change, React met à jour une copie du DOM appelée **Virtual DOM**.
2. Il compare l'ancienne version et la nouvelle version grâce à un processus appelé **diffing**.

3. Seules les modifications détectées sont appliquées au **DOM réel**, au lieu de re-render toute la page.

◆ Exemple concret : Mise à jour d'un compteur

Imaginons une application React avec un bouton qui incrémente un compteur :

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

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

  return (
    <div>
      <h1>Compteur : {count}</h1>
      <button onClick={() => setCount(count + 1)}>Incrémenter</button>
    </div>
  );
}

export default Counter;
```

💡 Ce qui se passe en arrière-plan avec le Virtual DOM :

- Lorsque l'utilisateur clique sur le bouton, **seul le <h1> affichant le compteur est mis à jour**.
- React **ne recharge pas** tout le DOM, il met uniquement à jour la partie du Virtual DOM qui a changé.
- Grâce à cela, les performances sont améliorées, évitant un re-rendu complet de la page.

◆ Comparaison avec le DOM classique

Dans un site traditionnel avec JavaScript natif, chaque mise à jour du DOM entraîne un rechargement plus coûteux. En revanche, avec React et son Virtual DOM, seules les parties réellement affectées sont mises à jour, rendant l'expérience utilisateur plus fluide et rapide.

En résumé, **le Virtual DOM permet d'éviter les mises à jour inutiles du DOM réel**, réduisant ainsi le nombre d'opérations coûteuses et améliorant la performance de l'application React. 🚀

Différence entre une page statique et une SPA (Single Page Application)

Une page statique se recharge entièrement à chaque action, alors qu'une SPA permet de naviguer sans recharger la page, offrant une expérience fluide.

2. Installation de React avec Vite

Pourquoi Vite ?

Vite est plus rapide et plus léger que Create React App, facilitant le développement.

Vite est souvent préféré à Create React App (CRA) pour plusieurs raisons, principalement en raison de sa rapidité et de sa légèreté. Voici pourquoi :

1. Temps de démarrage ultra-rapide

Vite utilise **ES Modules (ESM) natifs** et un **serveur de développement optimisé** qui ne recompile pas tout le projet à chaque changement. Contrairement à CRA (qui utilise Webpack et reconstruit l'ensemble du bundle), Vite ne traite que les fichiers modifiés, ce qui réduit considérablement le temps de démarrage et de rechargement.

2. Hot Module Replacement (HMR) ultra-rapide

Vite offre un **HMR instantané** grâce à son architecture basée sur ESM, alors que CRA peut devenir lent en raison de Webpack et de la nécessité de reconstruire de gros fichiers.

3. Bundler plus efficace (Esbuild + Rollup vs Webpack)

- **Vite utilise Esbuild** (écrit en Go) pour le pré-traitement des fichiers, ce qui est **beaucoup plus rapide** que Babel/Webpack.
- Pour la production, Vite utilise **Rollup**, qui génère des bundles plus optimisés et performants que Webpack.

4. Moins de configuration et de dépendances

- CRA installe beaucoup de dépendances inutiles (Webpack, Babel, etc.), ce qui alourdit le projet.
- Vite est **plus léger**, avec une configuration minimale, tout en permettant des personnalisations avancées.


5. Support natif TypeScript et JSX sans transpilation lourde

Vite supporte **TypeScript et JSX sans nécessiter Babel**, alors que CRA dépend de Babel, ce qui peut ralentir la compilation.

6. Meilleure expérience en développement et production

- **En dev**, Vite charge uniquement les fichiers demandés, ce qui réduit la consommation mémoire.
- **En prod**, le build avec Rollup génère un code plus optimisé et mieux divisé en chunks.

Conclusion

Vite est **plus rapide, plus léger et plus efficace** que Create React App, surtout pour les **grands projets** ou ceux qui nécessitent un développement rapide. CRA reste un bon choix pour ceux qui veulent un projet React clé en main, mais Vite est clairement **l'avenir du développement frontend**. 

Installation de React avec Vite

Ouvrez un terminal et exécutez les commandes suivantes :

```
npm create vite@latest my-app --template react  
cd my-app  
npm install  
npm run dev
```

Exercice :

1. Créez un projet React avec Vite.
2. Exécutez npm run dev et ouvrez le projet dans le navigateur.

3. Structure du projet

Principaux fichiers et dossiers dans un projet React :

- index.html : Page principale
- main.jsx : Point d'entrée de l'application
- App.jsx : Composant principal
- src/ : Contient les composants et assets

Exercice 3 :

1. Ouvrez App.jsx
2. Remplacez son contenu par :

```
function App() {  
  return (  
    <div>  
      <h1>Bienvenue dans mon premier projet React 🚀 </h1>  
    </div>  
  );  
}  
export default App;
```

3. Enregistrez et observez le résultat dans le navigateur.

4. Création et utilisation de composants

Un composant est une fonction qui retourne du JSX. Il permet de réutiliser du code facilement.

Création d'un composant Message.jsx

Créez un fichier Message.jsx dans le dossier src/ et ajoutez le code suivant :

```
function Message() {  
  return <p>Ceci est un composant React !</p>;  
}  
  
export default Message;
```

Importation et affichage du composant

Dans App.jsx, importez et utilisez Message :

```
import Message from "./Message";  
  
function App() {  
  return (  
    <div>  
      <h1>Bienvenue dans mon premier projet React 🚀</h1>  
      <Message />  
    </div>  
  );  
}  
  
export default App;
```

Exercice 4 :

1. Créez un composant Footer.jsx affichant "© 2025 - Mon projet React"
2. Importez et affichez Footer dans App.jsx

5. Bonnes pratiques et outils de débogage

- **React Developer Tools** : Extension pour inspecter les composants React
- **Console et debug** : Utiliser console.log() pour voir les données
- **Erreur courantes** : Problèmes avec JSX, oublis d'importations, erreurs de syntaxe

Exercice 5 :

1. Installer **React Developer Tools** (Chrome/Firefox)
2. Inspecter les composants dans le navigateur

6. Conclusion et devoirs

À retenir :

- React est basé sur des **composants** réutilisables
- JSX permet de mélanger HTML et JavaScript
- L'installation avec **Vite** permet de démarrer rapidement

Exercice à la maison :

1. Créer un composant **Header.jsx** avec un titre
2. L'afficher dans App.jsx
3. Ajouter du style CSS basique

 Prochaine étape : Composants et JSX avancés ! 