Introduction à ReactJS

Qu'est-ce que ReactJS ?

ReactJS est une bibliothèque JavaScript open-source développée par Facebook. Elle est utilisée pour construire des interfaces utilisateur (UI) pour des applications web à page unique (SPA - Single Page Application).

SPA (Single-Page Application)

Une application web monopage (SPA), ou single-page application en anglais, est une application web accessible via une seule page web. L'objectif principal des SPAs est d'améliorer l'expérience utilisateur en évitant le chargement d'une nouvelle page à chaque action demandée. Cela a plusieurs avantages :

- Réduction de la quantité de données à télécharger.
- Mise à jour plus rapide de la page, car seule une partie de la page est actualisée.
- Meilleure fluidité dans la navigation de l'utilisateur.

Le terme SPA a été introduit par Steve Yen en 2005. Les ressources nécessaires au fonctionnement de l'application peuvent être chargées dès l'ouverture de l'application ou récupérées dynamiquement au besoin, en fonction des actions de l'utilisateur. De nombreuses bibliothèques JavaScript, telles qu'AngularJS, Vue.js et React, sont spécialisées dans la création d'applications web monopages.

Pourquoi utiliser ReactJS ?

1. **Composants réutilisables** : React vous permet de créer des composants, des éléments d'interface utilisateur personnalisés que vous pouvez réutiliser dans toute votre application. Cela rend votre code plus lisible, plus facile à maintenir et plus efficace.
2. **Performances** : React utilise un DOM virtuel (Document Object Model virtuel) pour minimiser les mises à jour directes du DOM, ce qui améliore considérablement les performances de l'application.
3. **Communauté** : React a une grande communauté de développeurs et une multitude de ressources disponibles, ce qui facilite l'apprentissage et la résolution de problèmes.
4. **Prise en charge par Facebook** : React est développé et maintenu par Facebook, ce qui signifie qu'il est régulièrement mis à jour et amélioré.

Ces avantages font de React un choix populaire pour le développement d'applications web modernes.

Comment fonctionne ReactJS ?

1. **Composants React** : React vous permet de construire des interfaces utilisateur en utilisant des composants, qui sont des éléments d'interface utilisateur personnalisés et réutilisables. Chaque composant a son propre état et ses propres propriétés.
2. **DOM virtuel** : React utilise un DOM virtuel pour améliorer les performances. Le DOM virtuel est une représentation en mémoire du DOM réel. Lorsqu'un changement est apporté à l'interface utilisateur, React met à jour le DOM virtuel en premier, puis compare la version mise à jour avec la version précédente pour déterminer les modifications minimales nécessaires dans le DOM réel.
3. **JSX** : JSX est une syntaxe qui est utilisée pour décrire la structure des composants React. Bien que l'utilisation de JSX ne soit pas obligatoire avec React, elle est recommandée car elle rend le code plus lisible et plus facile à écrire.

ReactJS est une bibliothèque puissante et flexible pour construire des interfaces utilisateur interactives et dynamiques. Avec sa capacité à créer des composants réutilisables, ses performances optimisées grâce à l'utilisation du DOM virtuel, et le soutien d'une grande communauté, React est un excellent choix pour les développeurs de tous niveaux.

Installation et configuration

Installation de Node.js et npm

Tout d'abord, vous devez installer Node.js et npm sur votre ordinateur. Node.js est une plateforme logicielle qui permet d'exécuter du code JavaScript côté serveur, tandis que npm est un gestionnaire de paquets pour Node.js.

1. Allez sur le site officiel de Node.js et téléchargez la dernière version stable de Node.js.
2. Exécutez le fichier d'installation et suivez les étapes de l'assistant d'installation.
3. Pour vérifier que Node.js et npm sont correctement installés, ouvrez un terminal et exécutez les commandes suivantes :

```
node -v
npm -v
```

Si les commandes renvoient des numéros de version, cela signifie que Node.js et npm sont correctement installés.

Création d'une application React avec Create React App

Create React App est un outil qui permet de créer une application React avec une configuration de base.

Pour créer une application React avec Create React App, suivez ces étapes :

1. Ouvrez un terminal.
2. Exécutez la commande suivante pour créer une nouvelle application (remplacez "my-app" par le nom que vous souhaitez donner à votre application) :
   ```
   npx create-react-app my-app
   ```
3. Accédez au répertoire de votre nouvelle application :
   ```
   cd my-app
   ```
4. Démarrez le serveur de développement local et ouvrez votre application dans le navigateur par défaut avec la commande :
   ```
   npm start
   ```

Ces étapes vous permettront de créer rapidement une nouvelle application React.

Structure d'un projet React

Lorsque vous ouvrez le dossier de votre application dans votre éditeur de texte préféré, vous verrez plusieurs dossiers et fichiers importants :

```
crivant comment utiliser ou exécuter l'application.
│
├── node_modules/    # Dossier contenant toutes les dépendances et
bibliothèques nécessaires pour votre projet. Ces dépendances sont
installées via npm ou yarn.
│
├── package.json     # Fichier contenant les métadonnées du projet,
comme les dépendances, les scripts, la version, etc.
│
├── public/          # Dossier contenant les fichiers publics qui ne
seront pas traités par Webpack.
│   │
│   ├── index.html   # Page HTML principale de votre application. C'est
le point d'entrée de votre app React.
│   │
│   └── favicon.ico  # Icône de favori (ou "favicon") qui s'affiche
dans l'onglet du navigateur.
│
└── src/             # Dossier contenant le code source de votre
application React.
    │
    ├── App.css      # Fichier CSS pour le style de votre composant
App.
    │
    ├── App.js       # Fichier principal du composant App.
    │
```

```
    ├── App.test.js  # Fichier de tests pour votre composant App.
    │
    ├── index.css    # Fichier CSS global pour des styles généraux.
    │
    ├── index.js     # Point d'entrée JavaScript de votre application.
C'est ici que ReactDOM rend votre composant App.
    │
    └── logo.svg     # Logo de l'application, souvent le logo React par
défaut pour les nouvelles applications CRA.
```

React Developer Tools

React Developer Tools est une extension de navigateur qui vous aide à inspecter et à déboguer vos applications React :

- Vous pouvez voir l'arborescence des composants React telle qu'elle est rendue sur la page. Cela vous permet de naviguer dans la hiérarchie des composants et de voir leurs props, leur état et d'autres informations.
- React Developer Tools prend également en charge l'inspection des Hooks, vous permettant de voir les valeurs actuelles des Hooks comme useState, useEffect, etc.

Pour Chrome : Vous pouvez installer React Developer Tools depuis le Chrome Web Store.

Pour Firefox : L'extension est disponible sur le Firefox Add-ons Marketplace.

Après l'installation, vous verrez une icône React (un atome) dans la barre d'outils de votre navigateur.

Lorsque vous visitez une page qui utilise React, cette icône sera activée, vous permettant d'ouvrir les outils de développement et d'accéder à l'onglet React.

2. Composants et Props

Les composants sont les éléments de base de ReactJS. Ils permettent de découper l'interface utilisateur en morceaux réutilisables et indépendants.

Composant

Un composant est une partie de l'interface utilisateur qui peut être réutilisée dans différentes parties de l'application. Les composants peuvent être des éléments

simples comme des boutons ou des champs de texte, ou des éléments plus complexes comme des formulaires ou des tableaux. En ReactJS, il existe deux types de composants : les composants fonctionnels et les composants de classe.

- **Composants fonctionnels** : Ce sont des fonctions JavaScript qui retournent un élément React. Par exemple, voici un composant fonctionnel qui affiche un titre :

```
function Titre(props) {
  return <h1>{props.texte}</h1>;
}
```

- **Composants de classe** : Ce sont des classes JavaScript qui étendent la classe **React.Component**. Par exemple, voici un composant de classe qui affiche un compteur :

```
class Compteur extends React.Component {
  constructor(props) {
    super(props);
    this.state = { compteur: 0 };
  }

  render() {
    return <div>{this.state.compteur}</div>;
  }
}
```

La majuscule est obligatoire pour les noms de composants. React considère les composants commençant par des lettres minuscules comme des balises DOM. Par exemple, `<div />` représente une balise HTML div, mais `<Welcome />` représente un composant et exige que l'identifiant `Welcome` existe dans la portée courante.

Passage de données avec les props

Les props (abréviation de "properties") sont des données que l'on peut passer à un composant pour le configurer ou lui fournir des données.

Les props sont passées en tant qu'attributs HTML dans le JSX. Par exemple, pour passer une chaîne de caractères à un composant, on peut utiliser la syntaxe suivante :

```
<Titre texte="Mon titre" />
```

Dans le composant `Titre`, vous pouvez accéder aux props en utilisant `this.props` pour les composants de classe, ou en utilisant le paramètre `props` pour les composants fonctionnels.

Le State en React

Le state en React est un objet JavaScript qui permet de stocker des données qui peuvent être modifiées dans un composant. Contrairement aux props, le state est interne à un composant et peut être modifié par ce dernier.

Définition du state

Pour définir un state dans un composant de classe, vous devez utiliser le constructeur de la classe. Par exemple :

```
// Importation de la bibliothèque React (si ce n'est pas déjà fait)
import React from 'react';

// Définition d'une classe "MyComponent" qui étend la classe "Component" de
React
class MyComponent extends React.Component {
    // Le constructeur est appelé lors de la création d'une instance de ce
composant
    constructor(props) {
      // Appel du constructeur de la classe parent (React.Component) avec les
props
      super(props);

      // Initialisation du state du composant avec un objet ayant une
propriété "count" initialisée à 0
      this.state = { count: 0 };
    }

    // La méthode render est appelée à chaque fois que React décide de re-
rendre ce composant
    render() {
      // Retourne un élément div qui affiche la valeur actuelle de "count"
depuis le state
      return <div>{this.state.count}</div>;
    }
}

// Exportation de la classe "MyComponent" pour qu'elle puisse être utilisée
dans d'autres fichiers
export default MyComponent;
```

Dans cet exemple, le state est initialisé à **`{ count: 0 }`**. Le composant affiche ensuite la valeur de **`this.state.count`**.

Modification du state

Pour modifier le state dans un composant de classe, vous devez utiliser la méthode **`setState`**. Par exemple :

```
// Importation de la bibliothèque React (si ce n'est pas déjà fait)
import React from 'react';

// Définition d'une classe "MyComponent" qui étend la classe "Component" de
React
class MyComponent extends React.Component {
    // Le constructeur est appelé lors de la création d'une instance de ce
composant
    constructor(props) {
      // Appel du constructeur de la classe parent (React.Component) avec les
props
      super(props);

      // Initialisation du state du composant avec un objet ayant une
propriété "count" initialisée à 0
      this.state = { count: 0 };
    }

    // Méthode pour gérer le clic sur le bouton
    handleClick() {
      // Mise à jour du state en augmentant la valeur de "count" de 1
      this.setState({ count: this.state.count + 1 });
    }

    // La méthode render est appelée à chaque fois que React décide de re-
rendre ce composant
    render() {
      // Retourne un élément div contenant :
      // - Un div qui affiche la valeur actuelle de "count" depuis le state
      // - Un bouton qui, lorsqu'il est cliqué, appelle la méthode
"handleClick" pour incrémenter le compteur
      return (
        <div>
          <div>{this.state.count}</div>
          <button onClick={() => this.handleClick()}>Incrémenter</button>
        </div>
      );
    }
}

// Exportation de la classe "MyComponent" pour qu'elle puisse être utilisée
dans d'autres fichiers
export default MyComponent;
```

Dans cet exemple, la méthode **`handleClick`** utilise **`setState`** pour modifier la valeur de **`count`** dans le state. Le composant affiche ensuite la valeur de **`this.state.count`** et un bouton qui appelle **`handleClick`** lorsque l'utilisateur clique dessus.

Conclusion

Le state en React est un objet JavaScript qui permet de stocker des données qui peuvent être modifiées dans un composant. Le state est interne à un composant et peut être modifié par ce dernier. Pour définir un state, vous devez utiliser le constructeur de la classe. Pour modifier le state, vous devez utiliser la méthode **`setState`** dans un composant de classe.

Les Hooks

Les Hooks sont une fonctionnalité introduite dans React 16.8 qui d'utiliser des fonctionnalités de React telles que l'état et les effets sans avoir à écrire de classes.

Les Hooks sont des fonctions qui permettent de "s'accrocher" aux fonctionnalités de l'état et du cycle de vie de React à partir de composants fonctionnels.

Ils ne fonctionnent pas à l'intérieur des classes, mais permettent d'utiliser React sans classes.

useState :

Avant l'introduction des Hooks dans React, l'utilisation de l'état local était principalement limitée aux composants de classe.

Cependant, avec useState, les composants fonctionnels peuvent également avoir un état.

C'est un hook de React qui permet de définir et de mettre à jour l'état d'un composant fonctionnel.

Il est utilisé pour stocker des données qui peuvent être modifiées et qui doivent être conservées entre les rendus.

La syntaxe de useState est la suivante :

```
const [state, setState] = useState(initialState);
```

state est la variable qui contient l'état actuel,

setState est la fonction qui permet de mettre à jour l'état.

initialState est la valeur initiale de l'état.

Voici un exemple d'utilisation de useState :

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

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

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>Click me</button>
    </div>
  );
}
```

useState peut également être utilisé avec des objets :

```
const [id, setId] = useState({ name: '', age: 0 });
```

Pour mettre à jour un objet, il est important de se rappeler que setState ne fusionne pas automatiquement l'objet. Vous devez le faire manuellement :

```
setId(prevState => ({ ...prevState, name: 'Alice' }));
```

Il est tout à fait possible d'utiliser plusieurs appels useState dans un seul composant :

```
const [age, setAge] = useState(0);
const [fruit, setFruit] = useState('banane');
const [todos, setTodos] = useState([{ text: 'Apprendre les Hooks' }]);
```

Les composants fonctionnels sont devenus de plus en plus populaires en raison de leur syntaxe plus concise et de leur facilité de développement, de compréhension et de test. De plus, l'équipe React continue d'introduire davantage de Hooks pour les composants fonctionnels, ce qui les rend encore plus puissants.