Concepts sur React.js

Introduction

React est une bibliothèque JavaScript open-source développée par Facebook pour construire des interfaces utilisateurs (UI) de manière efficace et modulaire. Elle est particulière appréciée pour sa performance, sa flexibilité et sa communauté active.

Pourquoi utiliser React ?

Composants réutilisables : React permet de créer des composants encapsulés qui gèrent leur propre état, ce qui facilite la réutilisation du code.

DOM Virtuel : React utilise un DOM virtuel pour optimiser les mises à jour de l'interface, ce qui améliore les performances.

Ecosystème riche : Avec des outils comme React Router, Redux et Next.js, React offre une solution complète pour le développement d'applications modernes.

Communauté active : React bénéficie d'une grande communauté et d'une documentation exhaustive.

Concepts de base de React

1 – JSX (JavaScript XML)

JSX est une syntaxe qui permet d'écrire de HTML dans du JavaScript. Bien que ce ne soit pas obligatoire, il est largement utilisé pour faciliter la création de composants.

```
const element = <h1>Bonjour, monde !</h1>;
```

2 – Composants

Les composants sont les blocs de construction de React. Ils peuvent être de types :

- Composants fonctionnels : ce sont des fonctions JavaScript qui retournement du JSX.
- Composants de classe : ce sont des classes ES6 qui étendent React.Composant.

Exemple de composant fonctionnel :

```
function Welcome(props) {
```

```
  return <h1>Bonjour, {props.name}</h1>;
}
```

Exemple de composant de classe :

```
class Welcome extends React.Component {
  render() {
    return <h1>Bonjour, {this.props.name}</h1>;
  }
}
```

3 – Props et State

Props : les props (propriétés) sont des arguments passés à un composant. Elles sont immuables.

State : le state (état) est un objet qui contient des données qui peuvent changer au fi du temps. Il est géré à l'intérieur du composant.

Exemple d'utilisation du state :

```
class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = { date: new Date() };
  }

  render() {
    return (
      <div>
        <h1>Il est {this.state.date.toLocaleTimeString()}.</h1>
      </div>
    );
  }
```

```
}
```

4 – Gestion des événements

React utilise une syntaxe similaire au HTML pour gérer événements, mais en camelCase.

```
<button onClick={handleClick}>Cliquez-moi</button>
```

5 – Cycle de vie des composants

Les composants de classe ont des méthodes de cycle de vie comme *componentDidMount*, *componentDidUpdate*, et *componentWillUnmount* pour gérer les effets de bord.

Fonctionnalités avancées

1 – Hooks

Introduits dans React 16.8, les hooks permettent d'utiliser des fonctionnalités comme le state et le cycle de vie dans des composants fonctionnels.

Exemple avec useState :

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


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


  return (
    <div>
      <p>Vous avez cliqué {count} fois</p>
      <button onClick={() => setCount(count + 1)}>Cliquez ici</button>
    </div>
  );
```

```
}
```

2 – Context API

La Context API permet de partager des données entre composants sans avoir à passer explicitement des props à chaque niveau.

Exemple :

```
const ThemeContext = React.createContext('light');


function App() {
  return (
    <ThemeContext.Provider value="dark">
      <Toolbar />
    </ThemeContext.Provider>
  );
}


function Toolbar() {
  return (
    <div>
      <ThemedButton />
    </div>
  );
}


function ThemedButton() {
  const theme = useContext(ThemeContext);
  return <button style={{ background: theme === 'dark' ? '#333' : '#FFF' }}>Button</button>;
```

```
}
```

3 – React Router

React Router est une bibliothèque populaire pour gérer la navigation dans les applications React.

Exemple :

```
import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';


function App() {
  return (
    <Router>
      <Switch>
        <Route path="/about" component={About} />
        <Route path="/" component={Home} />
      </Switch>
    </Router>
  );
}
```

4 – Redux

Redux est une bibliothèque de gestion d'état souvent utilisée avec React pour gérer des états globaux complexes.

Bonnes pratiques

1) Découpage en petits composants : divisez votre application en petits composants réutilisables.

2) Utilisation des hooks : préférez les hooks pour gérer le state et les effets de bords dans les composants fonctionnels.
3) Optimisation des performances : utilisez React.memo, userMemo, et useCallback pour éviter les rendus inutiles.
4) Tests : utilisez des outils comme Jest et React Testing Library pour tester vos composants.

Conclusion

React est une bibliothèque puissante et flexible pour construire des interfaces utilisateur modernes. Avec ses concepts de composants, de state, et de hooks, React permet de créer des applications performantes et maintenables. En suivant les bonnes pratiques et en explorant son écosystème riche, vous pouvez maîtriser React et l'utiluser des projets de toutes tailles.