# Chapitre 1

## 1. React est une bibliothèque JavaScript :

### Leçon :

React est une bibliothèque JavaScript qui permet de créer des interfaces utilisateur interactives et dynamiques. Contrairement à un framework, elle se concentre uniquement sur la création de l'interface utilisateur, laissant aux développeurs le choix d'autres outils pour gérer l'architecture complète d'une application.

### Exemple de code :

```
// Un simple composant en React
import React from 'react';

function Bonjour() {
  return <h1>Bonjour, React est une bibliothèque JavaScript !</h1>;
}

export default Bonjour;
```

## 2. React couvre la couche UI (Interface Utilisateur) :

### Leçon :

React se limite à la couche d'interface utilisateur (UI), qui représente tout ce qui est visible par l'utilisateur. Elle ne gère ni les données ni la logique métier de l'application, mais se concentre sur la manière dont les données sont affichées et interactives.

### *Exemple de code :*

```
function CarteProfil({ nom, age }) {
  return (
    <div>
      <h2>Nom: {nom}</h2>
      <p>Âge: {age}</p>
    </div>
  );
}

// Utilisation
function App() {
  return <CarteProfil nom="Alice" age={30} />;
}

export default App;
```

### 3. Utilisation du DOM Virtuel pour les performances

**Leçon :**

React utilise un DOM virtuel (Virtual DOM) pour améliorer les performances. Le DOM virtuel est une copie légère du DOM réel. React l'utilise pour détecter les changements et ne mettre à jour que les éléments modifiés dans le DOM réel, réduisant ainsi les temps de chargement.

**Exemple de code :**

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

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

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

export default Compteur;
```

### 4. React est basé sur les composants :

**Leçon :**

React utilise des composants pour construire les interfaces. Un composant est une partie autonome de l'interface, comme un bouton ou un formulaire. Chaque composant est réutilisable, ce qui facilite la gestion et la composition d'applications.

**Exemple de code :**

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

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

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

export default Compteur;
```

## 5. Organisation des composants dans un projet React :

**Leçon :**

Les composants sont généralement stockés dans un dossier components dans le répertoire src/ pour structurer le code proprement.

**Exemple de code :**

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

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

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

export default Compteur;
```

## 6. Un composant peut rendre plusieurs éléments:

**Leçon :**

Un composant React peut retourner plusieurs éléments à condition qu'ils soient encapsulés dans un conteneur parent, comme une div ou React.Fragment.

**Exemple de code :**

```jsx
function Liste() {
  return (
    <ul>
      <li>Élément 1</li>
      <li>Élément 2</li>
      <li>Élément 3</li>
    </ul>
  );
}

export default Liste;
```

## 7. Limites de React :

### Leçon :

React a certaines limites :

- Il gère seulement la couche vue, il faut donc combiner d'autres outils pour une application complète.
- L'utilisation de JSX peut être difficile à appréhender pour certains.
- La taille de la bibliothèque peut être un obstacle pour les petites applications.

### Exemple de code :

```jsx
// Exemple d'utilisation de JSX pour créer une interface simple
function Message() {
  return <p>React est puissant, mais il nécessite d'autres outils pour une application complète.</p>;
}

export default Message;

```

## 8. Accès à l'état d'un composant avec this.state

### Leçon :

Dans une classe React, l'état (state) est accessible via this.state. Dans les fonctions, on utilise le hook useState.

### Exemple de code (classe) :

```jsx
import React, { Component } from 'react';

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

  incrementer = () => {
    this.setState({ count: this.state.count + 1 });
  };

  render() {
    return (
      <div>
        <p>Compteur : {this.state.count}</p>
        <button onClick={this.incrementer}>Incrémenter</button>
      </div>
    );
  }
}
```

### 9. Les Props pour transmettre des données entre composants

**Leçon :**

Les props sont des données passées d'un composant parent à un composant enfant et ne peuvent être modifiées par l'enfant.

**Exemple de code :**

```jsx
function Enfant({ message }) {
  return <p>{message}</p>;
}

function Parent() {
  return <Enfant message="Hello depuis le parent !" />;
}

export default Parent;
```

10. **Le State comme stockage interne d'un composant**

**Leçon :**
Le state représente les données internes d'un composant, spécifiques à son fonctionnement. Contrairement aux props, le state peut être modifié par le composant lui-même.

**Exemple de code :**

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

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

  const incrementer = () => {
    setCount(count + 1);
  };

  return (
    <div>
      <p>Compteur : {count}</p>
      <button onClick={incrementer}>Incrémenter</button>
    </div>
  );
}

export default Compteur;
```