FRONTEND PROGRAM

3- **React hooks**

ISMAIL BOUROUBA

React hooks

What is React hooks?

Les hooks React sont une fonctionnalité puissante dans React qui vous permet d'ajouter des états et d'autres fonctionnalités aux composants fonctionnels.

1: Hook useState :

Le hook useState vous permet d'ajouter un état à un composant fonctionnel. Il prend une valeur initiale en tant qu'argument et renvoie un tableau avec deux éléments : la valeur actuelle de l'état et une fonction pour la mettre à jour.

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

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

  const increment= () => {
    setCount(count + 1);
  }
const decrement = () => {
    setCount(count - 1);
  }

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={increment}>Increment</button>
      <button onClick={decrement}>decrement</button>
    </div>
  );
}
```

2: Hook useEffect

Le hook useEffect vous permet d'effectuer des effets secondaires dans un composant fonctionnel. Les effets secondaires incluent des actions telles que la récupération de données depuis une API, la mise à jour du DOM ou l'abonnement à un événement.

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

function DataFetcher() {
  const [data, setData] = useState([]);

  useEffect(() => {
    fetch('https://api.example.com/data')
      .then(response => response.json())
      .then(data => setData(data));
  }, []);

  return (
    <ul>
      {data.map(item => (
        <li key={item.id}>{item.name}</li>
      ))}
    </ul>
  );
}
```

3: Le Hook useContext :

Le Hook useContext vous permet d'accéder à un objet de contexte dans un composant fonctionnel. Le contexte est une manière de faire passer des données dans l'arborescence des composants sans avoir à transmettre manuellement des propriétés (props).

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

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

function ThemeButton() {
  const theme = useContext(ThemeContext);

  return (
    <button style={{ background: theme === 'dark' ? 'black' : 'white',
color: theme === 'dark' ? 'white' : 'black' }}>
      Toggle Theme
    </button>
  );
}
```

4: Le Hook useReducer :

Le Hook useReducer vous permet de gérer un état complexe dans un composant fonctionnel. Il est similaire au Hook useState, mais au lieu d'une valeur simple, il prend une fonction de réduction (reducer) et un état initial.

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

function cartReducer(state, action) {
  switch (action.type) {
    case 'add':
      return [...state, action.payload];
    case 'remove':
      return state.filter(item => item.id !== action.payload.id);
    default:
      return state;
  }
}

function ShoppingCart() {
  const [cart, dispatch] = useReducer(cartReducer, []);

  const addItem = (item) => {
    dispatch({ type: 'add', payload: item });
  }

  const removeItem = (item) => {
    dispatch({ type: 'remove', payload: item });
  }

  return (
    <div>
      <h2>Shopping Cart</h2>
      <ul>
        {cart.map(item => (
          <li key={item.id}>
            {item.name} - ${item.price}
            <button onClick={() => removeItem(item)}>Remove</button>
          </li>
        ))}
      </ul>
      <button onClick={() => addItem({ id: 1, name: 'Item 1', price: 9.99
})}>Add Item</button>
    </div>
  );
}
```

5: Le Hook useCallback :

Le crochet useCallback vous permet de mémoriser une fonction afin qu'elle ne soit recréée que lorsque ses dépendances changent. Cela peut contribuer à améliorer les performances en évitant des ré-renderisations inutiles.

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

function SearchBar({ onSearch }) {
  const [query, setQuery] = useState('');

  const handleQueryChange = useCallback(event => {
    setQuery(event.target.value);
    onSearch(event.target.value);
  }, [onSearch]);

  return (
    <input type="text" value={query} onChange={handleQueryChange} />
  );
}
```

6: Le Hook useMemo:

Hook useMemo vous permet de mémoriser une valeur afin qu'elle ne soit recalculée que lorsque ses dépendances changent. Cela peut contribuer à améliorer les performances en évitant des recalculs inutiles.

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

function ExpensiveCalculation({ a, b }) {
  const result = useMemo(() => {
    console.log('Calculating...');
    return a * b;
  }, [a, b]);

  return <p>Result: {result}</p>;
}
```