Chapter 05: Quản lý State nâng cao với useReducer và useContext trong ReactJS

A. Mục tiêu

Sau bài học này, học viên sẽ:

- Hiểu khi nào `useState` không đủ và cần sử dụng giải pháp mạnh mẽ hơn.
- Thành thạo hook `useReducer` để quản lý state phức tạp và logic cập nhật state.
- Hiểu rõ vấn đề **prop drilling** và cách hook `useContext` giải quyết nó.
- Kết hợp `useReducer` và `useContext` để tạo pattern quản lý state toàn cục đơn giản.

B. Nội dung lý thuyết

1. Giới hạn của useState

Hook `useState` phù hợp cho state đơn giản, nhưng gặp khó khăn trong các trường hợp:

- **State phức tạp**: Khi state là object hoặc array lớn với nhiều thuộc tính cần quản lý.
- **Logic cập nhật phức tạp**: Khi logic cập nhật state phụ thuộc vào state trước đó hoặc có nhiều hành động khác nhau.
- **Tái sử dụng logic**: Khi cần tách biệt logic cập nhật state để dễ test và tái sử dụng.
- **Hiệu năng**: Khi nhiều thay đổi state dẫn đến render không cần thiết.

Trong các trường hợp này, `useReducer` cung cấp cách tiếp cận tập trung và dễ dự đoán hơn.

2. Hook useReducer - Lựa chọn thay thế cho useState

Cú pháp

```
const [state, dispatch] = useReducer(reducer, initialState);
```

- `reducer` : Hàm thuần túy `(state, action) ⇒ newState`, nhận state hiện tại và một **action** để trả về state mới.
- `action` : Object mô tả hành động, thường có thuộc tính `type` (chuỗi định danh) và `payload` (dữ liệu bổ sung).
- `dispatch` : Hàm gửi action đến reducer để cập nhật state.
- `initialState` : Giá trị khởi tạo của state.

Ví dụ:

```
const initialState = { count: 0 };

function reducer(state, action) {
  switch (action.type) {
    case 'increment':
      return { count: state.count + 1 };
    case 'decrement':
      return { count: state.count - 1 };
    default:
      return state;
  }
}

function Counter() {
  const [state, dispatch] = useReducer(reducer, initialState);
  return (
    <div>
      <p>Đếm: {state.count}</p>
      <button onClick={() ⇒ dispatch({ type: 'increment' })}>Tăng</button>
      <button onClick={() ⇒ dispatch({ type: 'decrement' })}>Giảm</button>
    </div>
  );
}
```

Ưu điểm

- Tách biệt logic cập nhật state ra khỏi component.
- Dễ dự đoán vì reducer là hàm thuần túy.
- Tối ưu hiệu năng vì `dispatch` không thay đổi giữa các lần render.
- Dễ test và tái sử dụng logic reducer.

3. Vấn đề Prop Drilling

Prop drilling xảy ra khi bạn phải truyền props qua nhiều cấp component trung gian để đưa dữ liệu hoặc hàm đến component con sâu trong cây component. Ví dụ:

```
function App() {
  const [data, setData] = useState('');
  return <Parent data={data} setData={setData} />;
}
function Parent({ data, setData }) {
  return <Child data={data} setData={setData} />;
}
function Child({ data, setData }) {
  return <input value={data} onChange={(e) => setData(e.target.value)} />;
}
```

Điều này làm code:

- Khó đọc và bảo trì.
- Dễ gây lỗi khi cấu trúc ứng dụng thay đổi.
- Tăng độ phức tạp khi truyền nhiều props.

4. Hook useContext - Giải pháp cho Prop Drilling

Hook `useContext` cho phép chia sẻ dữ liệu toàn cục mà không cần truyền props qua từng cấp component.

Các bước sử dụng

1. **Tạo Context:**

```
import { createContext } from 'react';
const MyContext = createContext(defaultValue);
```

2. **Cung cấp giá trị qua Provider:**

```
<MyContext.Provider value={data}>
  <Child />
</MyContext.Provider>
```

3. **Sử dụng trong component con:**

```
import { useContext } from 'react';
function Child() {
  const data = useContext(MyContext);
  return <p>{data}</p>;
}
```

5. Pattern kết hợp useReducer và useContext

Kết hợp `useReducer` và `useContext` tạo ra hệ thống quản lý state toàn cục mạnh mẽ, phù hợp cho ứng dụng vừa và nhỏ:

- Sử dụng `useReducer` để quản lý state và logic.
- Đặt `state` và `dispatch` vào Context để các component con truy cập.
- Các component con dùng `useContext` để đọc state hoặc gửi action qua `dispatch`.

Ví dụ:

```jsx
const MyContext = createContext();

function reducer(state, action) {
  switch (action.type) {
    case 'increment':
      return { count: state.count + 1 };
    default:
      return state;
  }
}

function App() {
  const [state, dispatch] = useReducer(reducer, { count: 0 });
  return (
    <MyContext.Provider value={{ state, dispatch }}>
      <Child />
    </MyContext.Provider>
  );
}

function Child() {
  const { state, dispatch } = useContext(MyContext);
  return (
    <div>
      <p>Đếm: {state.count}</p>
      <button onClick={() => dispatch({ type: 'increment' })}>Tăng</button>
    </div>
  );
}
```

C. Bài tập thực hành

Bài 1: Refactor component Counter với useReducer

- **Yêu cầu**: Chuyển component `Counter` từ bài 1 sang sử dụng `useReducer` với các action `increment` và `decrement`.
- **Hướng dẫn**:

1. Tạo `src/components/Counter.js` :

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

const initialState = { count: 0 };

function reducer(state, action) {
  switch (action.type) {
    case 'increment':
      return { count: state.count + 1 };
    case 'decrement':
      return { count: state.count - 1 };
    default:
      return state;
  }
}

function Counter() {
  const [state, dispatch] = useReducer(reducer, initialState);
  return (
    <div>
      <h2>Đếm: {state.count}</h2>
      <button onClick={() => dispatch({ type: 'increment' })}>Tăng</
      <button onClick={() => dispatch({ type: 'decrement' })}>Giảm</
    </div>
  );
}
export default Counter;
```

2. Sử dụng trong `App.jsx` :

```
import Counter from './components/Counter';

function App() {
  return <Counter />;
}
export default App;
```

Bài 2: Refactor ứng dụng Todo List với useReducer

- **Yêu cầu**: Quản lý state `todos` bằng `useReducer` với các action: `ADD_TODO`, `DELETE_TODO`, `TOGGLE_TODO`.
- **Hướng dẫn**:
  1. Tạo `src/components/TodoList.js`:

```
function TodoList({ todos, dispatch }) {
  return (
    <ul>
      {todos.map(todo ⇒ (
        <li key={todo.id} style={{ textDecoration: todo.completed ?
          {todo.text}
          <button onClick={() ⇒ dispatch({ type: 'TOGGLE_TODO', pay
            {todo.completed ? 'Hoàn tác' : 'Hoàn thành'}
          </button>
          <button onClick={() ⇒ dispatch({ type: 'DELETE_TODO', pay
        </li>
      ))}
    </ul>
  );
}
export default TodoList;
```

  2. Tạo `src/components/TodoForm.js`:

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

function TodoForm({ dispatch }) {
  const [text, setText] = useState('');

  function handleSubmit(e) {
    e.preventDefault();
    if (text.trim()) {
      dispatch({ type: 'ADD_TODO', payload: { id: Date.now(), text,
      setText('');
    }
```

```
  }

  return (
    <div>
      <input
        type="text"
        value={text}
        onChange={(e) => setText(e.target.value)}
        placeholder="Thêm công việc"
      />
      <button onClick={handleSubmit}>Thêm</button>
    </div>
  );
}
export default TodoForm;
```

3. Cập nhật `src/App.jsx`:

```
import { useReducer } from 'react';
import TodoForm from './components/TodoForm';
import TodoList from './components/TodoList';

const initialState = { todos: [] };

function reducer(state, action) {
  switch (action.type) {
    case 'ADD_TODO':
      return { todos: [...state.todos, action.payload] };
    case 'DELETE_TODO':
      return { todos: state.todos.filter(todo => todo.id !== action.
    case 'TOGGLE_TODO':
      return {
        todos: state.todos.map(todo =>
          todo.id === action.payload ? { ...todo, completed: !todo.c
        )
      };
    default:
      return state;
  }
}
```

```
function App() {
  const [state, dispatch] = useReducer(reducer, initialState);
  return (
    <div>
      <h1>Todo List</h1>
      <TodoForm dispatch={dispatch} />
      <TodoList todos={state.todos} dispatch={dispatch} />
    </div>
  );
}
export default App;
```

Bài 3: Chuyển đổi giao diện Sáng/Tối (Light/Dark Theme)

- **Yêu cầu**: Sử dụng `useContext` để quản lý theme toàn cục (`light` / `dark`).
- **Hướng dẫn**:
  1. Tạo `src/context/ThemeContext.js` :

```
import { createContext } from 'react';

const ThemeContext = createContext();
export default ThemeContext;
```

  2. Tạo `src/components/ThemeToggleButton.js` :

```
import { useContext } from 'react';
import ThemeContext from '../context/ThemeContext';

function ThemeToggleButton() {
  const { toggleTheme } = useContext(ThemeContext);
  return <button onClick={toggleTheme}>Chuyển đổi Theme</button>;
}
export default ThemeToggleButton;
```

3. Tạo `src/components/HomePage.js`:

```
import { useContext } from 'react';
import ThemeContext from '../context/ThemeContext';

function HomePage() {
  const { theme } = useContext(ThemeContext);
  return (
    <div style={{ background: theme === 'light' ? '#fff' : '#333', c
      <h2>Trang chủ</h2>
      <p>Theme hiện tại: {theme}</p>
    </div>
  );
}
export default HomePage;
```

4. Cập nhật `src/App.jsx`:

```
import { useState } from 'react';
import ThemeContext from './context/ThemeContext';
import ThemeToggleButton from './components/ThemeToggleButton';
import HomePage from './components/HomePage';

function App() {
  const [theme, setTheme] = useState('light');
  const toggleTheme = () => setTheme(theme === 'light' ? 'dark' : 'l

  return (
    <ThemeContext.Provider value={{ theme, toggleTheme }}>
      <HomePage />
      <ThemeToggleButton />
    </ThemeContext.Provider>
  );
}
export default App;
```

Bài 4: Nâng cấp Todo List với useReducer và useContext

- **Yêu cầu**: Sử dụng `useReducer` và `useContext` để quản lý state `todos` mà không cần truyền props.
- **Hướng dẫn**:
  1. Tạo `src/context/TodoContext.js`:

```
import { createContext } from 'react';

const TodoContext = createContext();
export default TodoContext;
```

  2. Cập nhật `src/components/TodoForm.js`:

```
import { useState, useContext } from 'react';
import TodoContext from '../context/TodoContext';

function TodoForm() {
  const { dispatch } = useContext(TodoContext);
  const [text, setText] = useState('');

  function handleSubmit(e) {
    e.preventDefault();
    if (text.trim()) {
      dispatch({ type: 'ADD_TODO', payload: { id: Date.now(), text,
      setText('');
    }
  }

  return (
    <div>
      <input
        type="text"
        value={text}
        onChange={(e) => setText(e.target.value)}
        placeholder="Thêm công việc"
      />
```

```
      <button onClick={handleSubmit}>Thêm</button>
    </div>
  );
}
export default TodoForm;
```

3. Cập nhật `src/components/TodoList.js` :

```
import { useContext } from 'react';
import TodoContext from '../context/TodoContext';

function TodoList() {
  const { state, dispatch } = useContext(TodoContext);
  return (
    <ul>
      {state.todos.map(todo => (
        <li key={todo.id} style={{ textDecoration: todo.completed ?
          {todo.text}
          <button onClick={() => dispatch({ type: 'TOGGLE_TODO', pay
            {todo.completed ? 'Hoàn tác' : 'Hoàn thành'}
          </button>
          <button onClick={() => dispatch({ type: 'DELETE_TODO', pay
        </li>
      ))}
    </ul>
  );
}
export default TodoList;
```

4. Cập nhật `src/App.jsx` :

```
import { useReducer } from 'react';
import TodoContext from './context/TodoContext';
import TodoForm from './components/TodoForm';
import TodoList from './components/TodoList';

const initialState = { todos: [] };
```

```
function reducer(state, action) {
  switch (action.type) {
    case 'ADD_TODO':
      return { todos: [...state.todos, action.payload] };
    case 'DELETE_TODO':
      return { todos: state.todos.filter(todo => todo.id !== action.
    case 'TOGGLE_TODO':
      return {
        todos: state.todos.map(todo =>
          todo.id === action.payload ? { ...todo, completed: !todo.c
        )
      };
    default:
      return state;
  }
}

function App() {
  const [state, dispatch] = useReducer(reducer, initialState);
  return (
    <TodoContext.Provider value={{ state, dispatch }}>
      <h1>Todo List</h1>
      <TodoForm />
      <TodoList />
    </TodoContext.Provider>
  );
}
export default App;
```

Bài 5: Xây dựng giỏ hàng đơn giản

- **Yêu cầu**: Sử dụng `useReducer` và `useContext` để quản lý giỏ hàng với các action: `ADD_TO_CART`, `REMOVE_FROM_CART`, `INCREASE_QUANTITY`, `DECREASE_QUANTITY`.
- **Hướng dẫn**:
  1. Tạo `src/context/CartContext.js`:

```
import { createContext } from 'react';
```

```
const CartContext = createContext();
export default CartContext;
```

2. Tạo `src/components/ProductList.js`:

```
import { useContext } from 'react';
import CartContext from '../context/CartContext';

const products = [
  { id: 1, name: 'Laptop', price: 999 },
  { id: 2, name: 'Điện thoại', price: 499 },
  { id: 3, name: 'Tai nghe', price: 99 },
];

function ProductList() {
  const { dispatch } = useContext(CartContext);
  return (
    <div>
      <h2>Sản phẩm</h2>
      <ul>
        {products.map(product => (
          <li key={product.id}>
            {product.name} - ${product.price}
            <button onClick={() => dispatch({ type: 'ADD_TO_CART', p
              Thêm vào giỏ
            </button>
          </li>
        ))}
      </ul>
    </div>
  );
}
export default ProductList;
```

3. Tạo `src/components/ShoppingCart.js`:

```
import { useContext } from 'react';
import CartContext from '../context/CartContext';
```

```
function ShoppingCart() {
  const { state, dispatch } = useContext(CartContext);
  const total = state.cart.reduce((sum, item) => sum + item.price * 

  return (
    <div>
      <h2>Giỏ hàng</h2>
      <ul>
        {state.cart.map(item => (
          <li key={item.id}>
            {item.name} - ${item.price} x {item.quantity}
            <button onClick={() => dispatch({ type: 'INCREASE_QUANTI
            <button onClick={() => dispatch({ type: 'DECREASE_QUANTI
            <button onClick={() => dispatch({ type: 'REMOVE_FROM_CAR
          </li>
        ))}
      </ul>
      <p>Tổng tiền: ${total}</p>
    </div>
  );
}
export default ShoppingCart;
```

4. Cập nhật `src/App.jsx`:

```
import { useReducer } from 'react';
import CartContext from './context/CartContext';
import ProductList from './components/ProductList';
import ShoppingCart from './components/ShoppingCart';

const initialState = { cart: [] };

function reducer(state, action) {
  switch (action.type) {
    case 'ADD_TO_CART':
      const existingItem = state.cart.find(item => item.id === actio
      if (existingItem) {
        return {
          cart: state.cart.map(item =>
            item.id === action.payload.id ? { ...item, quantity: ite
```

```
          )
        };
      }
      return { cart: [...state.cart, action.payload] };
    case 'REMOVE_FROM_CART':
      return { cart: state.cart.filter(item => item.id !== action.pa
    case 'INCREASE_QUANTITY':
      return {
        cart: state.cart.map(item =>
          item.id === action.payload ? { ...item, quantity: item.qua
        )
      };
    case 'DECREASE_QUANTITY':
      return {
        cart: state.cart.map(item =>
          item.id === action.payload && item.quantity > 1
            ? { ...item, quantity: item.quantity - 1 }
            : item
        )
      };
    default:
      return state;
  }
}

function App() {
  const [state, dispatch] = useReducer(reducer, initialState);
  return (
    <CartContext.Provider value={{ state, dispatch }}>
      <h1>Giỏ hàng</h1>
      <ProductList />
      <ShoppingCart />
    </CartContext.Provider>
  );
}
export default App;
```