Tổng Hợp Kiến Thức

JSX

- **JSX là gì?**: Cú pháp mở rộng của JavaScript cho phép viết HTML trực tiếp trong code JavaScript. Không phải HTML thực sự, mà là cú pháp để React hiểu và tạo DOM.
- **Quy tắc JSX cơ bản:**
  - **Một phần tử gốc:** Mỗi component JSX phải trả về **một** phần tử bao bọc duy nhất (ví dụ: `<div>`, `<React.Fragment>`, `<> </>`).
  - **Biểu thức JavaScript trong JSX:** Sử dụng dấu ngoặc nhọn `{}` để nhúng biểu thức JavaScript vào JSX.
  - **Thuộc tính HTML:** Sử dụng camelCase cho thuộc tính HTML (ví dụ: `className` thay vì `class`, `onClick` thay vì `onclick`).
  - **Tự đóng thẻ:** Các thẻ không có nội dung phải tự đóng (ví dụ: `<input />`, `<img />`).

```
function MyComponent() {
  const name = "React Developer";
  const age = 30;

  return (
    <div> {/* Phần tử gốc */}
      <h1>Xin chào, {name}!</h1> {/* Nhúng biến JavaScript */}
      <p>Bạn {age} tuổi.</p>
      <button onClick={() => alert('Button clicked!')}> {/* Thuộc tính sự kiện */}
        Click me
      </button>
      <input type="text" placeholder="Nhập tên" /> {/* Tự đóng thẻ */}
    </div>
  );
}
```

Component

- **Component là gì?**: Khối xây dựng cơ bản của giao diện người dùng trong React. Tái sử dụng, độc lập và có thể kết hợp với nhau.
- **Hai loại Component chính:**
  - **Function Component:** Component là một hàm JavaScript trả về JSX. (Ưu tiên sử dụng).
  - **Class Component:** Component là một class ES6 kế thừa từ `React.Component` và có phương thức `render()` trả về JSX. (Ít dùng hơn trong React hiện đại).
- **Ví dụ Function Component:**

```
function WelcomeMessage(props) { // Nhận props (tùy chọn)
  return <h1>Chào mừng, {props.name}!</h1>;
}
```

- **Sử dụng Component:** Gọi Component như một thẻ HTML trong JSX:

```
<WelcomeMessage name="Bạn" />  {/* Sử dụng Function Component */}
```

Style trong React

- **Inline Style:** Style trực tiếp trong thuộc tính `style` của JSX. Giá trị là một object JavaScript.

```
<div style={{ color: 'blue', fontSize: '16px' }}>...</div>
```

- **CSS Classes (CSS Truyền Thống):**
  1. Tạo file CSS (`.css`).
  2. Định nghĩa class rules trong file CSS.
  3. Import file CSS vào component (`import './MyComponent.css';`).
  4. Sử dụng thuộc tính `className` trong JSX để gán class (`<div className="container">`).
- **CSS Modules:** (Khuyến khích cho dự án lớn)
  1. Tạo file CSS Modules (`.module.css`).
  2. Định nghĩa class rules trong file CSS Modules.
  3. Import file CSS Modules vào component (`import styles from './MyComponent.module.css';`).
  4. Sử dụng thuộc tính `className` và `styles.className` để gán class (`<div className={styles.container}>`). Tên class trở thành thuộc tính của object `styles` .

Biến và Biểu thức JavaScript

- **Biến trong JSX:** Nhúng biến JavaScript vào JSX bằng dấu ngoặc nhọn `{}` .

```
function MyComponent() {
  const message = "Đây là một thông điệp.";
  return <p>{message}</p>;
}
```

- **Biểu thức JavaScript trong JSX:** Có thể nhúng bất kỳ biểu thức JavaScript hợp lệ nào vào JSX trong dấu ngoặc nhọn.

```
function MyComponent() {
  const number = 10;
  return <p>Giá trị gấp đôi là: {number * 2}</p>;
}
```

- **Chú ý:** Không thể nhúng câu lệnh JavaScript (ví dụ: `if` , `for`) trực tiếp vào JSX. Cần sử dụng biểu thức điều kiện (ternary operator) hoặc render có điều kiện (conditional rendering).

State (Trạng thái)

- **State là gì?**: Dữ liệu **bên trong** component có thể thay đổi theo thời gian và làm component re-render khi thay đổi. Quản lý dữ liệu động và tương tác.
- **Hook** `useState` **:** Hook để thêm state vào Function Component.
  - **Khai báo State:** `const [stateValue, setStateFunction] = useState(initialValue);`
    - `stateValue` : Biến state (giá trị hiện tại của state).
    - `setStateFunction` : Hàm để cập nhật state.
    - `initialValue` : Giá trị khởi tạo cho state.
- **Cập nhật State:** Sử dụng `setStateFunction` để cập nhật state. **Không bao giờ** sửa đổi trực tiếp `stateValue` .
  - **Ví dụ:** `setStateFunction(newValue);` hoặc `setStateFunction(prevState => newValueBasedOnPrevState);` (dùng callback khi cập nhật state dựa trên state trước đó).
- **Re-render:** Khi state được cập nhật bằng `setStateFunction` , React tự động re-render component để hiển thị giao diện mới.
- **Ví dụ** `useState` **:**

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

function Counter() {
  const [count, setCount] = useState(0); // Khởi tạo state 'count' = 0
```

```
  const handleIncrement = () => {
    setCount(count + 1); // Cập nhật state 'count'
  };

  return (
    <div>
      <p>Đếm: {count}</p>
      <button onClick={handleIncrement}>Tăng</button>
    </div>
  );
}
```

Sự Kiện (Events)

- **Xử lý sự kiện trong React:** Tương tự HTML, nhưng cú pháp JSX khác một chút.
  - Sử dụng **camelCase** cho tên sự kiện (ví dụ: `onClick`, `onChange`, `onSubmit`).
  - Truyền một **hàm JavaScript** (event handler) cho thuộc tính sự kiện.
- **Sự kiện phổ biến:** `onClick`, `onChange`, `onSubmit`, `onKeyDown`, `onMouseOver`, v.v.
- **Đối tượng sự kiện (event object):** Hàm xử lý sự kiện nhận một đối tượng sự kiện `event` (thường được gọi là `e`) chứa thông tin về sự kiện.
  - `e.target`: Phần tử HTML đã kích hoạt sự kiện.
  - `e.preventDefault()`: Ngăn chặn hành vi mặc định của sự kiện (ví dụ: submit form làm reload trang).
- **Ví dụ xử lý sự kiện `onClick`:**

```
function MyButton() {
  const handleClick = (event) => { // Hàm xử lý sự kiện
    console.log('Button clicked!', event.target);
    alert('Button đã được click!');
  };

  return (
    <button onClick={handleClick}>Click Me</button>  {/* Gán hàm xử lý sự kiện */}
  );
}
```

Cập nhật State với Callback (prevState)

- **Khi nào cần dùng Callback để cập nhật State?**: Khi giá trị state mới phụ thuộc vào giá trị state trước đó. Đảm bảo cập nhật chính xác trong các tình huống bất đồng bộ hoặc cập nhật liên tục.
- **Cú pháp Callback trong `setStateFunction`:** Truyền một **hàm** vào `setStateFunction`. Hàm này nhận vào `prevState` (state trước đó) và trả về giá trị state mới.

  ```
  setCount(prevState => prevState + 1); // Tăng count lên 1 dựa trên giá trị trước đó
  ```

- **Lợi ích của Callback:**
  - **Đảm bảo tính chính xác:** Tránh lỗi "state stale" khi cập nhật state dựa trên giá trị đã cũ.
  - **Cập nhật state bất đồng bộ:** React có thể gộp nhiều lần cập nhật state để tối ưu hiệu năng. Callback giúp đảm bảo các cập nhật state tuần tự và chính xác.
- **Ví dụ Callback trong Counter Component:**

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

```
  const handleIncrement = () => {
    setCount(prevState => prevState + 1); // Cập nhật state bằng callback
  };

  return (
    <div>
      <p>Đếm: {count}</p>
      <button onClick={handleIncrement}>Tăng</button>
    </div>
  );
}
```

Render Danh Sách (Lists)

- **Render danh sách động trong React:** Sử dụng phương thức `map()` của mảng để lặp qua các phần tử trong mảng và trả về JSX cho mỗi phần tử.
- `key` **Prop bắt buộc:** Khi render danh sách, **bắt buộc** phải thêm prop `key` vào phần tử **ngoài cùng** của mỗi item trong danh sách. `key` giúp React theo dõi các phần tử trong danh sách hiệu quả hơn khi có sự thay đổi. `key` phải là **duy nhất** và **ổn định** giữa các lần render.
- **Ví dụ Render Danh Sách:**

```
function ProductList(props) {
  const products = [
    { id: 1, name: 'Sản phẩm A', price: 100 },
    { id: 2, name: 'Sản phẩm B', price: 200 },
    { id: 3, name: 'Sản phẩm C', price: 150 }
  ];

  return (
    <ul>
      {products.map(product => (
        <li key={product.id}> {/* Sử dụng key prop - BẮT BUỘC */}
          {product.name} - Giá: ${product.price}
        </li>
      ))}
    </ul>
  );
}
```

9. Props (Properties - Thuộc tính)

- **Props là gì?**: Cơ chế để **truyền dữ liệu từ Component cha xuống Component con.** Giúp component con nhận dữ liệu từ bên ngoài và hiển thị động.
- **Props là Read-only (chỉ đọc):** Component con **không được phép** thay đổi Props mà nó nhận được. Dữ liệu một chiều (one-way data flow).
- **Truyền Props:** Giống như thuộc tính HTML khi sử dụng component.

```
<MyComponent data="Giá trị prop" count={123} isActive={true} />
```

- **Nhận Props trong Function Component:** Props được truyền vào Function Component như là **đối số đầu tiên (argument)** của hàm. Thường đặt tên là `props`.

```
function MyComponent(props) {
  console.log(props); // props là một object chứa tất cả props truyền vào
  return <p>Dữ liệu prop: {props.data}</p>; // Truy cập prop: props.tênProp
}
```

- **Destructuring Props (ES6 Destructuring):** Cách gọn hơn để truy cập props bằng cách "giải nén" trực tiếp các prop từ object `props` trong đối số hàm.

```
function MyComponent({ data, count, isActive }) { // Destructuring props
  return (
    <div>
      <p>Dữ liệu: {data}</p>
      <p>Count: {count}</p>
      <p>Active: {isActive ? 'Có' : 'Không'}</p>
    </div>
  );
}
```

- **Prop `children` đặc biệt:** Cho phép truyền **JSX (bao gồm cả các component khác)** làm nội dung con của một component. Nội dung con được truy cập qua `props.children`.

```
function Card(props) {
  return (
    <div style={{ border: '1px solid black', padding: '10px' }}>
      {props.children} {/* Hiển thị nội dung con */}
    </div>
  );
}

// Sử dụng Card component:
<Card>
  <h2>Tiêu đề Card</h2><p>Nội dung bên trong Card.</p>
</Card>
```