**BÁO CÁO TỔNG HỢP**

**Chương 1. Giới thiệu về ReactJS**

1. **Khái niệm**
   * ReactJS là thư viện JavaScript mã nguồn mở do Facebook phát triển năm 2013.
   * Được thiết kế để xây dựng giao diện người dùng (UI) theo hướng component-based.
   * Khác với Angular (framework toàn diện), React chỉ tập trung vào “View” trong mô hình MVC**.**
2. **Lịch sử và sự phát triển**
   * 2011: React lần đầu xuất hiện trong hệ thống Facebook News Feed.
   * 2013: Facebook công bố ReactJS mã nguồn mở.
   * 2015: React Native ra đời, mở rộng React sang lập trình mobile.
   * Hiện nay: ReactJS là công nghệ frontend phổ biến nhất thế giới (theo khảo sát StackOverflow**).**
3. **Ưu điểm chính**
   * Hiệu năng cao nhờ Virtual DOM.
   * Component-based: dễ tái sử dụng.
   * Hỗ trợ server-side rendering: cải thiện SEO.
   * Cộng đồng lớn: nhiều thư viện bổ trợ (Redux, React Router, Formik…).
4. **Nhược điểm**
   * Không phải framework toàn diện → cần dùng thêm thư viện khác cho routing, quản lý state phức tạp.
   * Tài liệu thay đổi liên tục → dễ gây khó khăn cho người mới.

**Chương 2. Virtual DOM**

1. **DOM truyền thống**
   * Mỗi thay đổi nhỏ (ví dụ: tăng số đếm) sẽ làm trình duyệt render lại toàn bộ cây DOM.
   * Dẫn đến chậm, đặc biệt với ứng dụng nhiều dữ liệu.
2. **Cơ chế Virtual DOM trong React**
   * React lưu trữ một bản sao ảo của DOM thật.
   * Khi state/props thay đổi:
     1. Tạo Virtual DOM mới.
     2. So sánh (diffing) với Virtual DOM cũ.
     3. Chỉ cập nhật phần khác biệt lên DOM thật (reconciliation).
3. **Ưu điểm**
   * Tối ưu hiệu suất.
   * Tránh thao tác DOM không cần thiết.
   * Giúp trải nghiệm người dùng mượt mà**.**
4. **Ví dụ minh họa**

import React, { useState } from "react";

function Counter() {

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

return (

<div>

<p>Bạn đã click {count} lần</p>

<button onClick={() => setCount(count + 1)}>Tăng</button>

</div>

);

}

Chỉ <p> được cập nhật, thay vì toàn bộ trang.

**Chương 3. JSX (JavaScript XML)**

1. **Khái niệm**
   * JSX cho phép viết HTML trong JavaScript.
   * Giúp code trực quan, dễ hiểu.
2. **Cơ chế biên dịch**
   * JSX không được trình duyệt hiểu trực tiếp.
   * Babel sẽ biên dịch JSX thành React.createElement().
3. **Tính năng nổi bật**
   * Nhúng biểu thức JS bằng {}.
   * Cho phép viết HTML có logic điều kiện, vòng lặp.
4. **Ví dụ minh họa**

const user = "Nguyễn Văn A";

const element = <h1>Xin chào, {user}</h1>;

**Biên dịch thành:**

const element = React.createElement("h1", null, "Xin chào, ", user);

**Chương 4. Components trong React**

1. **Khái niệm**
   * Component là khối UI độc lập, có thể tái sử dụng.
   * Mỗi ứng dụng React được tạo từ nhiều component lồng nhau**.**
2. **Class Component**

import React, { Component } from "react";

class Welcome extends Component {

render() {

return <h1>Xin chào, {this.props.name}</h1>;

}

}

1. **Function Component**

function Welcome(props) {

return <h1>Xin chào, {props.name}</h1>;

}

1. **So sánh**
   * Class Component: đầy đủ lifecycle, nhưng cú pháp dài.
   * Function Component: ngắn gọn, hiện đại, kết hợp với Hooks để thay thế lifecycle.
2. **Ứng dụng thực tế**
   * Header, Footer, Menu, Form đăng nhập… đều là các component tái sử dụng.

**Chương 5. Props và State**

1. **Props**
   * Dùng để truyền dữ liệu từ cha → con.
   * Không thể thay đổi.

function Hello(props) {

return <h2>Xin chào, {props.name}</h2>;

}

1. **State**
   * Lưu trữ dữ liệu nội bộ.
   * Có thể thay đổi bằng this.setState hoặc useState.

function Counter() {

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

return <button onClick={() => setCount(count + 1)}>Đếm: {count}</button>;

}

1. **So sánh**
   * Props = dữ liệu từ ngoài vào.
   * State = dữ liệu bên trong component**.**

**Chương 6. React Lifecycle**

1. **Class Component Lifecycle**
   * Mounting: constructor, render, componentDidMount.
   * Updating: shouldComponentUpdate, componentDidUpdate.
   * Unmounting: componentWillUnmount.
2. **Ví dụ**

class Demo extends React.Component {

componentDidMount() {

console.log("Component được mount");

}

componentWillUnmount() {

console.log("Component bị gỡ");

}

render() {

return <h1>Hello React</h1>;

}

}

1. **Function Component Lifecycle với Hooks**

* Dùng useEffect() thay thế.

useEffect(() => {

console.log("Mounted");

return () => console.log("Unmounted");

}, []);

**Chương 7. Event trong React**

1. **Khái niệm**
   * Event là hành động người dùng: click, nhập liệu, di chuột…
2. **Cú pháp**
   * Viết bằng camelCase: onClick, onChange.
   * Truyền hàm callback thay vì chuỗi.
3. **Ví dụ**

function App() {

const handleClick = () => alert("Bạn vừa click!");

return <button onClick={handleClick}>Click me</button>;

}

**Chương 8. Form trong ReactJS**

1. **Controlled Component**

* Input được quản lý bởi state.

function MyForm() {

const [value, setValue] = React.useState("");

const handleSubmit = e => {

e.preventDefault();

alert("Bạn nhập: " + value);

};

return (

<form onSubmit={handleSubmit}>

<input value={value} onChange={e => setValue(e.target.value)} />

<button type="submit">Gửi</button>

</form>

);

}

1. **Uncontrolled Component**

* Lấy dữ liệu từ DOM qua ref.

function MyForm() {

const inputRef = React.useRef();

const handleSubmit = e => {

e.preventDefault();

alert("Bạn nhập: " + inputRef.current.value);

};

return (

<form onSubmit={handleSubmit}>

<input ref={inputRef} />

<button type="submit">Gửi</button>

</form>

);

}

1. **Thư viện hỗ trợ**
   * Formik, React Hook Form: giúp validate dữ liệu dễ hơn.

**Chương 9. Hooks cơ bản (bổ sung để báo cáo dài hơn)**

1. **useState –** Quản lý state trong Function Component**.**
2. **useEffect –** Thay thế lifecycle**.**
3. **useContext –** Quản lý state toàn cục mà không cần Redux.
4. **useReducer –** Quản lý state phức tạp (giống Redux thu nhỏ).

**Ví dụ useState + useEffect:**

function Timer() {

const [time, setTime] = useState(0);

useEffect(() => {

const timer = setInterval(() => setTime(t => t + 1), 1000);

return () => clearInterval(timer);

}, []);

return <p>Thời gian: {time}s</p>;

}

**Chương 10. Kết luận**

* ReactJS là thư viện frontend mạnh mẽ, phổ biến nhất hiện nay.
* Các khái niệm Virtual DOM, JSX, Component, Props, State, Lifecycle, Event, Form là nền tảng để học nâng cao.
* Với Hooks, React hiện đại hơn, dễ dùng hơn.
* Ứng dụng: từ website thương mại điện tử (Shopee, Tiki) đến mạng xã hội (Facebook, Instagram).

**Chương 11. React Hook: useState**

**1. Khái niệm**

* useState là Hook cơ bản nhất trong React, cho phép quản lý **state trong Function Component**.
* Trước đây, state chỉ có trong **Class Component**, nhưng với useState, function component cũng có thể lưu trữ và thay đổi dữ liệu.

**2. Cú pháp**

const [state, setState] = useState(initialValue);

* state: giá trị hiện tại.
* setState: hàm dùng để cập nhật state.
* initialValue: giá trị khởi tạo (số, chuỗi, object, array...).

**3. Ví dụ cơ bản**

import React, { useState } from "react";

function Counter() {

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

return (

<div>

<p>Bạn đã click {count} lần</p>

<button onClick={() => setCount(count + 1)}>Tăng</button>

</div>

);

}

Khi click, setCount tăng giá trị → giao diện render lại.

**4. Lưu ý khi dùng useState**

* Không thay đổi state trực tiếp (count++ ).
* Cập nhật dựa trên state cũ:

setCount(prev => prev + 1);

* Có thể khởi tạo state bằng function → chỉ tính toán 1 lần khi render đầu tiên.

**5. useState với Object và Array**

**Object:**

const [user, setUser] = useState({ name: "An", age: 20 });

setUser(prev => ({ ...prev, age: 21 }));

**Array:**

const [list, setList] = useState([1, 2, 3]);

setList(prev => [...prev, 4]); // thêm

setList(prev => prev.filter(x => x!==2)); // xóa

**6. Ưu điểm**

* Ngắn gọn, dễ dùng hơn class component.
* Trực quan, đồng bộ UI với dữ liệu.
* Là nền tảng để kết hợp với các Hook khác (useEffect, useReducer, …).

**Chương 12. React Hooks nâng cao**

**1. Tổng quan về Hooks**

* **Hooks** được giới thiệu từ React 16.8 (2019).
* Trước đây: chỉ có *Class Component* mới có state và lifecycle.
* Với Hooks: *Function Component* cũng có thể quản lý state, lifecycle, context…
* Lợi ích:
  + Cú pháp gọn gàng, dễ đọc.
  + Tách logic tái sử dụng (custom hooks).
  + Thống nhất cách viết component (chỉ cần function).

**2. useEffect**

**2.1. Khái niệm**

* Dùng để quản lý **side effects** trong function component (gọi API, thao tác DOM, setInterval…).
* Thay thế các lifecycle như componentDidMount, componentDidUpdate, componentWillUnmount.

**2.2. Cú pháp**

useEffect(() => {

// logic chạy sau khi render

return () => {

// cleanup khi unmount

};

}, [dependencies]);

* dependencies: mảng phụ thuộc, quyết định khi nào useEffect chạy lại:
  + []: chỉ chạy 1 lần khi mount.
  + [state]: chạy lại khi state thay đổi.
  + Không có: chạy mỗi lần render.

**2.3. Ví dụ**

import React, { useState, useEffect } from "react";

function Timer() {

const [time, setTime] = useState(0);

useEffect(() => {

const t = setInterval(() => setTime(prev => prev + 1), 1000);

return () => clearInterval(t); // cleanup

}, []);

return <p>Thời gian: {time}s</p>;

}

Ứng dụng thực tế: **đồng hồ đếm giờ, gọi API khi load trang, theo dõi sự kiện bàn phím/chuột**.

**3. useContext**

**3.1. Khái niệm**

* Giúp truyền dữ liệu qua nhiều cấp component **mà không cần props drilling** (truyền props lồng nhau).

**3.2. Các bước sử dụng**

1. Tạo context bằng React.createContext.
2. Dùng <Context.Provider> để bọc các component con.
3. Dùng useContext(Context) để lấy dữ liệu.

**3.3. Ví dụ**

import React, { createContext, useContext } from "react";

const UserContext = createContext();

function Child() {

const user = useContext(UserContext);

return <h2>Xin chào {user}</h2>;

}

function App() {

return (

<UserContext.Provider value="Nguyễn Văn A">

<Child />

</UserContext.Provider>

);

}

Thường dùng cho **theme, ngôn ngữ, trạng thái đăng nhập**.

**4. useReducer**

**4.1. Khái niệm**

* Quản lý state phức tạp (nhiều điều kiện, nhiều bước cập nhật).
* Giống Redux thu nhỏ, nhưng dùng ngay trong React.

**4.2. Cú pháp**

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

function reducer(state, action) {

switch (action.type) {

case "increment":

return { count: state.count + 1 };

case "decrement":

return { count: state.count - 1 };

default:

return state;

}

}

**4.3. Ví dụ**

import React, { useReducer } from "react";

function Counter() {

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;

}

}

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

return (

<div>

<p>Số đếm: {state.count}</p>

<button onClick={() => dispatch({ type: "increment" })}>Tăng</button>

<button onClick={() => dispatch({ type: "decrement" })}>Giảm</button>

</div>

);

}

Dùng khi state có nhiều trường, nhiều logic (giỏ hàng, form phức tạp).

**5. Custom Hooks**

**5.1. Khái niệm**

* Người dùng có thể tự viết Hook để tái sử dụng logic.
* Quy tắc: tên Hook phải bắt đầu bằng **use**.

**5.2. Ví dụ – custom hook đếm thời gian**

import { useState, useEffect } from "react";

function useTimer() {

const [time, setTime] = useState(0);

useEffect(() => {

const t = setInterval(() => setTime(prev => prev + 1), 1000);

return () => clearInterval(t);

}, []);

return time;

}

// Component sử dụng

function App() {

const time = useTimer();

return <h1>Thời gian: {time}s</h1>;

}

**6. Tổng kết về Hooks**

* **useState**: quản lý state cơ bản.
* **useEffect**: quản lý side effects.
* **useContext**: chia sẻ dữ liệu toàn cục.
* **useReducer**: quản lý state phức tạp.
* **Custom Hook**: tái sử dụng logic.

Hooks làm React **dễ học hơn, code gọn hơn, logic rõ ràng hơn**, giúp function component thay thế gần như hoàn toàn class component.