**Bài 1: "Hello World!" & Môi trường phát triển 🚀**

* **Mục tiêu:** Hiểu React là gì và thiết lập thành công môi trường để bắt đầu.
* **Lý thuyết:**
  + **React là gì?**: Một thư viện JavaScript để xây dựng giao diện người dùng (UI), không phải là một framework hoàn chỉnh.
  + **Virtual DOM**: Cơ chế giúp React cập nhật UI một cách hiệu quả.
  + **Node.js & npm/yarn**: Môi trường chạy JavaScript phía server và công cụ quản lý thư viện.
* **Thực hành:**
  1. Cài đặt **Node.js** và **npm** (hoặc **yarn**).
  2. Sử dụng lệnh npx create-react-app my-react-app để tạo dự án đầu tiên.
  3. Chạy dự án với npm start và xem kết quả trên trình duyệt.
  4. **Khám phá cấu trúc thư mục**: Tìm hiểu vai trò của các file và thư mục quan trọng như public/index.html, src/index.js, src/App.js.

**Bài 2: JSX và Components - Viên gạch xây dựng UI 🧱**

* **Mục tiêu:** Hiểu cách React sử dụng JSX để mô tả UI và cách chia nhỏ UI thành các Components.
* **Lý thuyết:**
  + **JSX (JavaScript XML)**: Cú pháp cho phép viết HTML trong JavaScript.
  + **Components**: Các hàm JavaScript độc lập, tái sử dụng, trả về JSX để mô tả một phần của UI. Chúng ta sẽ tập trung vào **Function Components**.
* **Thực hành:**
  1. Trong App.js, xóa code mặc định và viết một thẻ <h1>Hello, [Tên của bạn]!</h1>.
  2. Tạo các component con đơn giản như Header.js, Content.js, Footer.js trong thư mục src/components.
  3. Import và lồng các component này vào App.js để xây dựng một layout cơ bản.

**Bài 3: Props & State - "Hơi thở" của Component ❤️‍🔥**

* **Mục tiêu:** Hiểu cách truyền dữ liệu và quản lý trạng thái bên trong một component.
* **Lý thuyết:**
  + **Props (Properties)**: Dữ liệu được truyền từ component cha xuống component con. **Props là bất biến (immutable)**.
  + **State**: Dữ liệu nội tại của một component, có thể thay đổi và khi thay đổi sẽ khiến component render lại.
  + **Hook useState**: Công cụ cơ bản nhất để tạo và quản lý state trong Function Component.
* **Thực hành:**
  1. Truyền một prop title="My Blog" từ App.js xuống Header.js và hiển thị nó.
  2. Tạo một component Counter.js. Sử dụng useState để tạo một state count. Hiển thị giá trị count và tạo một nút để tăng giá trị này mỗi khi click.

**Bài 4: Sự kiện, Vòng đời và useEffect 🔄**

* **Mục tiêu:** Xử lý tương tác người dùng và thực hiện các "side effect" (như gọi API, tương tác DOM).
* **Lý thuyết:**
  + **Handling Events**: Cách gắn các hàm xử lý sự kiện như onClick, onChange vào các phần tử JSX.
  + **Component Lifecycle**: Các giai đoạn của một component (mounting, updating, unmounting).
  + **Hook useEffect**: Cho phép thực thi code ở các giai đoạn khác nhau trong vòng đời của component.
* **Thực hành:**
  1. Tạo một ô input. Mỗi khi người dùng gõ, lấy giá trị và console.log ra.
  2. Sử dụng useEffect(() => { ... }, []) để thay đổi tiêu đề của trang (document.title) một lần duy nhất khi component được render.
  3. Tạo một component hiển thị thời gian thực, cập nhật mỗi giây bằng cách sử dụng useEffect kết hợp với setInterval.

**Bài 5: Conditional Rendering & Lists 📋**

* **Mục tiêu:** Hiển thị UI một cách linh hoạt dựa trên điều kiện và render danh sách dữ liệu.
* **Lý thuyết:**
  + **Conditional Rendering**: Sử dụng toán tử ba ngôi (condition ? <A /> : <B />) hoặc toán tử && (condition && <A />) để quyết định hiển thị component nào.
  + **Lists & Keys**: Dùng hàm .map() của mảng để render một danh sách các component. Thuộc tính key là bắt buộc và phải là duy nhất để React nhận diện các phần tử.
* **Thực hành:**
  1. Tạo một state isLoggedIn. Nếu true, hiển thị "Welcome back!"; nếu false, hiển thị nút "Login".
  2. Tạo một mảng dữ liệu (ví dụ: const todos = [{id: 1, text: 'Learn React'}, ...]).
  3. Sử dụng .map() để render danh sách này ra thành các thẻ <li>, đừng quên thêm key={todo.id}.

**Giai đoạn 2: Quản lý State chuyên nghiệp & Xây dựng CRUD (Tuần 3-4)**

Mục tiêu của giai đoạn này là áp dụng kiến thức đã học để xây dựng một ứng dụng hoàn chỉnh và quản lý state tập trung với Redux.

**Bài 6: Khởi tạo Redux - Trung tâm quản lý State 🏛️**

* **Mục tiêu:** Hiểu tại sao cần Redux và nắm được luồng dữ liệu cơ bản của nó.
* **Lý thuyết:**
  + **Vấn đề:** Quản lý state phức tạp, truyền props qua nhiều cấp (prop drilling).
  + **Kiến trúc Redux:**
    - **Store**: Nơi duy nhất chứa toàn bộ state của ứng dụng.
    - **Actions**: Các object mô tả sự kiện xảy ra (e.g., { type: 'todos/addTodo', payload: ... }).
    - **Reducers**: Các hàm thuần túy nhận vào state hiện tại và action, trả về state mới.
* **Thực hành:**
  1. Cài đặt các thư viện: redux, react-redux, @reduxjs/toolkit (khuyến khích dùng Toolkit vì nó đơn giản hóa rất nhiều thứ).
  2. Tạo một "slice" bằng createSlice cho ứng dụng Counter (bao gồm state, reducers, và actions).
  3. Cấu hình Store bằng configureStore.
  4. Kết nối Store với ứng dụng React bằng component <Provider>.
  5. Sử dụng hook useSelector để đọc state và useDispatch để gửi action từ component Counter đã tạo ở Bài 3.

**Bài 7: Lắp ráp ứng dụng CRUD - Xây dựng Giao diện 🎨**

* **Mục tiêu:** Xây dựng giao diện tĩnh cho một ứng dụng quản lý công việc (Todo App) hoàn chỉnh.
* **Thực hành:**
  1. **Cấu trúc thư mục**: Sắp xếp lại project theo cấu trúc pages, components, features (hoặc redux).
  2. Tạo trang HomePage.js.
  3. Tạo các component tái sử dụng: Button.js, Input.js, TodoItem.js.
  4. Dựng giao diện gồm: một form để thêm công việc mới và một danh sách để hiển thị các công việc.

**Bài 8: Redux và Tác vụ bất đồng bộ (Async) 📞**

* **Mục tiêu:** Gọi API (giả lập) để lấy và cập nhật dữ liệu bằng Redux.
* **Lý thuyết:**
  + **Redux Thunk**: Một middleware cho phép viết các action creators có thể trả về một hàm (thay vì object), nơi chúng ta thực hiện logic bất đồng bộ.
  + **createAsyncThunk**: Một hàm từ Redux Toolkit giúp tạo ra các thunks xử lý API và tự động dispatch các actions cho các trạng thái pending, fulfilled, rejected.
* **Thực hành:**
  1. Tạo một slice todosSlice cho việc quản lý công việc.
  2. Sử dụng createAsyncThunk để tạo các thunks cho việc fetchTodos, addTodo, deleteTodo.
  3. Trong extraReducers của slice, xử lý 3 trạng thái của mỗi thunk để cập nhật state (loading, success, error).
  4. Trong component, dispatch các async thunks này để tương tác với "API".

**Bài 9: Hoàn thiện CRUD & Form Validation ✅**

* **Mục tiêu:** Kết nối tất cả lại với nhau và thêm validation cho form.
* **Thực hành:**
  1. **Create**: Kết nối form thêm công việc để dispatch action addTodo.
  2. **Read**: Sử dụng useSelector để lấy danh sách công việc từ store và hiển thị ra TodoItem. Dispatch fetchTodos khi component được mount.
  3. **Update**: Thêm chức năng đánh dấu hoàn thành công việc.
  4. **Delete**: Thêm nút xóa và dispatch action deleteTodo.
  5. **Validation**: Thêm logic để không cho phép thêm công việc nếu input rỗng.
  6. **(Nâng cao)**: Tìm hiểu và tích hợp thư viện react-hook-form để quản lý form chuyên nghiệp hơn.

**Bài 10: Tối ưu hóa với Reselect và React.memo ⚡**

* **Mục tiêu:** Ngăn chặn các lần render không cần thiết để cải thiện hiệu năng.
* **Lý thuyết:**
  + **React.memo**: Một HOC (Higher-Order Component) giúp component chỉ render lại khi props của nó thay đổi.
  + **Reselect**: Tạo ra các hàm selector được "ghi nhớ" (memoized), chỉ tính toán lại khi phần state mà nó phụ thuộc thay đổi.
* **Thực hành:**
  1. Bọc component TodoItem trong React.memo để nó không bị render lại khi các công việc khác thay đổi.
  2. Sử dụng thư viện reselect để tạo một selector tính toán số công việc đã hoàn thành. Sử dụng selector này trong UI và quan sát hiệu năng qua React DevTools.

Chúc bạn học tốt! Hãy đi từng bước một, thực hành thật nhiều và đừng ngần ngại đọc thêm tài liệu chính thức nhé.