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

Mục tiêu của bài học này là biến bạn từ việc chỉ biết chỉnh sửa một file duy nhất (App.js) thành một "kiến trúc sư" có thể tự tạo ra các "viên gạch" (Components) và lắp ráp chúng thành một giao diện hoàn chỉnh.

**Phần 1: Tìm hiểu về "Ngôn ngữ thiết kế" - JSX**

**Bước 1: JSX là gì? Tại sao nó tồn tại?**

Hãy mở file src/App.js của bạn. Bạn thấy code bên trong return() chứ? Nó trông rất giống HTML nhưng lại nằm trong một file JavaScript. Đó chính là **JSX**.

* **Định nghĩa:** JSX (JavaScript XML) là một phần mở rộng cú pháp cho JavaScript. Nó cho phép bạn viết code trông giống HTML ngay trong logic JavaScript của mình.
* **Analogy:** Tưởng tượng bạn đang viết một kịch bản phim. Thay vì viết lời thoại ở một nơi và chỉ dẫn hành động ở một nơi khác, JSX cho phép bạn viết cả hai chung với nhau. Điều này giúp code trở nên rất trực quan: bạn nhìn vào code là hình dung ra giao diện sẽ trông như thế nào.

// Đây là JSX

const myElement = <h1>Chào bạn, đây là một tiêu đề.</h1>;

**Bước 2: 3 quy tắc "vàng" của JSX cần nhớ**

Khi viết JSX, người mới bắt đầu thường gặp lỗi ở 3 điểm sau. Hãy ghi nhớ chúng:

1. **Phải trả về một thẻ cha duy nhất:** Một component không thể trả về nhiều thẻ nằm ngang hàng nhau. Chúng phải được bọc trong một thẻ cha chung.
   * **SAI ❌:**

return (

<h1>Tiêu đề</h1>

<p>Nội dung</p>

)

* + **ĐÚNG ✅ (Dùng <div> hoặc Fragment <>):**

return (

<div>

<h1>Tiêu đề</h1>

<p>Nội dung</p>

</div>

)

// Hoặc dùng Fragment để không tạo ra thẻ div thừa

return (

<>

<h1>Tiêu đề</h1>

<p>Nội dung</p>

</>

)

1. **Dùng className thay vì class:** Vì class là một từ khóa đã được sử dụng trong JavaScript để tạo lớp (class), nên trong JSX, để gán class CSS, bạn phải dùng className.
   * **SAI ❌:** <div class="container">
   * **ĐÚNG ✅:** <div className="container">
2. **Thuộc tính dùng camelCase:** Các thuộc tính HTML có hai từ (ví dụ: onclick, tabindex) sẽ được viết theo kiểu camelCase trong JSX.
   * **SAI ❌:** <button onclick="...">
   * **ĐÚNG ✅:** <button onClick="...">

**Phần 2: "Những viên gạch Lego" - Components**

**Bước 3: Hiểu về Component**

* **Analogy:** Nếu trang web của bạn là một ngôi nhà, thì **Components** chính là những viên gạch, cửa sổ, cửa ra vào... Chúng là những phần giao diện độc lập, có thể tái sử dụng. Thay vì xây cả bức tường lớn, bạn sẽ tạo ra từng viên gạch rồi ghép chúng lại.
* **Function Component:** Cách hiện đại và phổ biến nhất để tạo component. Nó đơn giản là một hàm JavaScript có những đặc điểm sau:
  + Tên hàm **luôn viết hoa chữ cái đầu** (ví dụ: Header, Button).
  + Hàm này **trả về JSX**.

**Phần 3: Thực hành - Xây dựng Layout đầu tiên**

Bây giờ, chúng ta sẽ áp dụng lý thuyết ở trên để chia trang web đơn giản của mình thành 3 component: Header, Content, và Footer.

**Bước 4: Dọn dẹp App.js**

Đầu tiên, hãy dọn dẹp file App.js để chuẩn bị "lắp ráp".

1. Mở file src/App.js.
2. Xóa dòng import './App.css'; (chúng ta chưa cần CSS lúc này).
3. Xóa hết nội dung bên trong return (...) và chỉ để lại một thẻ div rỗng.
4. Code của bạn bây giờ trông sẽ như thế này:

function App() {

return (

<div className="App">

{/\* Chúng ta sẽ lắp ráp các component vào đây \*/}

</div>

);

}

export default App;

**Bước 5: Tạo thư mục components**

Để giữ cho dự án gọn gàng, chúng ta nên tạo một thư mục riêng để chứa các component con.

1. Trong cây thư mục của VS Code, click chuột phải vào thư mục src.
2. Chọn **New Folder** và đặt tên là components.

**Bước 6: Tạo Component Header.js**

1. Click chuột phải vào thư mục components vừa tạo, chọn **New File**.
2. Đặt tên file là Header.js (Lưu ý viết hoa chữ H).
3. Bên trong file Header.js, gõ đoạn code sau:

function Header() {

// Nó trả về JSX mô tả phần đầu trang

return (

<header>

<h1>Chào mừng đến với Blog của tôi!</h1>

<nav>

<a href="#">Trang chủ</a> | <a href="#">Giới thiệu</a> | <a href="#">Liên hệ</a>

</nav>

</header>

);

}

// Dòng này rất quan trọng! Nó cho phép các file khác có thể "gọi" và sử dụng component Header.

export default Header;

**Bước 7: Tạo Component Content.js và Footer.js**

Làm tương tự như Bước 6 để tạo thêm 2 file nữa trong thư mục components.

1. Tạo file Content.js:

function Content() {

return (

<main>

<h2>Đây là bài viết đầu tiên</h2>

<p>Nội dung của bài viết sẽ được đặt ở đây. React rất thú vị!</p>

</main>

);

}

export default Content;

1. Tạo file Footer.js:

function Footer() {

return (

<footer>

<p>Bản quyền © 2025. Thực hiện bởi Sinh viên Đại học Vinh.</p>

</footer>

);

}

export default Footer;

**Bước 8: Lắp ráp các "viên gạch" lại với nhau**

Bây giờ chúng ta đã có 3 "viên gạch" Header, Content, Footer. Quay trở lại file src/App.js để lắp ráp chúng thành ngôi nhà hoàn chỉnh.

1. Mở lại file src/App.js.
2. Ở trên cùng, bên dưới import React..., hãy **import** 3 component bạn vừa tạo vào:

import Header from './components/Header';

import Content from './components/Content';

import Footer from './components/Footer';

1. Bên trong return của App, hãy đặt các component này vào như thể chúng là các thẻ HTML:

function App() {

return (

<div className="App">

<Header />

<Content />

<Footer />

</div>

);

}

Code đầy đủ của file App.js sẽ trông như sau:

import Header from './components/Header';

import Content from './components/Content';

import Footer from './components/Footer';

function App() {

return (

<div className="App">

<Header />

<Content />

<Footer />

</div>

);

}

export default App;

**Bước 9: Chiêm ngưỡng thành quả!**

Lưu tất cả các file lại. Bây giờ hãy nhìn vào trình duyệt của bạn. Bạn sẽ thấy nội dung từ cả 3 component đã được ghép lại với nhau một cách hoàn hảo!

**Chúc mừng bạn đã hoàn thành bài học thứ hai!** 🎉 Bạn đã học được khái niệm quan trọng nhất của React: chia nhỏ giao diện thành các **Component** độc lập và có thể tái sử dụng, sau đó kết hợp chúng lại với nhau. Đây là nền tảng cho mọi thứ phức tạp hơn sau này.