

🧩 1. Colocate – Đặt mọi thứ gần nơi được dùng

👉 Ý tưởng:

Giữ **component, style, state, hàm xử lý**,... càng gần nơi sử dụng càng tốt.

✅ Lợi ích:

- Code **gọn hơn, dễ đọc**
- **Dễ bảo trì**
- Giảm **render lại không cần thiết** do prop/state đi vòng xa

🧠 Ví dụ: Nếu một hook chỉ dùng cho **CommentBox**, hãy để trong thư mục **CommentBox/** chứ không cần vớt vào **src/hooks**.

🗑️ 2. Tránh component quá lớn với nhiều hàm render bên trong

❌ Không nên:

tsx

```
function Component() {  
  function renderItem() {  
    return <ul>...</ul>;  
  }  
  return <div>{renderItem()}</div>;  
}
```

✅ Nên tách thành component con:

tsx

```
function Items() {  
  return <ul>...</ul>;  
}
```

```
function Component() {  
  return <div><Items /></div>;  
}
```

✅ Lợi ích:

- Code dễ đọc
 - Tái sử dụng được **Items**
 - Tách biệt trách nhiệm (separation of concerns)
-

3. Giữ phong cách viết code nhất quán

- Tên component nên dùng **PascalCase**
- Dùng Prettier và ESLint để **đảm bảo style giống nhau**
- Tên file: nên dùng **kebab-case** hoặc **PascalCase**, đừng trộn lẫn

! Độ nhất quán quan trọng hơn phong cách cụ thể. Nhất quán = chuyên nghiệp.

4. Hạn chế số lượng **props** trong component

Nếu component nhận quá nhiều props (8–10+), hãy cân nhắc:

- Tách ra nhiều component nhỏ hơn
- Dùng **children**, **slots** hoặc **context** để giảm số props

Lợi ích:

- Component dễ dùng, dễ đọc
 - Giảm nhầm lẫn khi truyền props
-

5. Trừu tượng hoá các component dùng chung (Shared Component Library)

Khi bạn thấy lặp đi lặp lại các UI giống nhau (ví dụ: button, modal, card...), hãy:

- Tạo thư viện component riêng: **src/components/**

- Đừng trừu tượng hoá quá sớm → chỉ tách khi thực sự có ít nhất 2 nơi dùng lại

6. Quấn (wrap) các component từ thư viện bên thứ 3

Mục tiêu:

Khi bạn dùng một thư viện UI lớn như Material UI (MUI), Ant Design (AntD), Chakra UI,... thì:

- Thay vì dùng component của họ trực tiếp khắp nơi (`<Button>`, `<Input>`, `<Modal>`,...),
- Bạn nên tạo một lớp "wrapper component" riêng cho từng component bạn cần (ví dụ: `MyButton`, `MyInput`, `MyModal`).

Vì sao nên quấn (wrap):

1. Dễ tùy chỉnh / thêm logic riêng

Bạn có thể:

- Thêm style mặc định cho toàn bộ hệ thống.
- Gắn thêm logic (tracking, validation, events, animation,...)
- Đồng bộ hoá theme hoặc hành vi ứng dụng.

Ví dụ:

tsx

```
// MyButton.tsx
```

```
import { Button } from '@mui/material';
```

```
export const MyButton = ({ children, ...props }) => {
```


```
return (  
  <Button  
    variant="contained"  
    color="primary"  
    sx={{ borderRadius: '12px', textTransform: 'none' }}  
    {...props}  
  >  
    {children}  
  </Button>  
);  
};
```

Sau đó bạn dùng `<MyButton />` thay vì `<Button />` khắp nơi.

2. Dễ thay thế thư viện sau này


Nếu bạn dùng `MUI` khắp nơi → sau này muốn đổi sang `ChakraUI` thì bạn phải sửa hàng trăm file.

Nhưng nếu bạn wrap một lần ở `MyButton.tsx`, `MyModal.tsx`,... thì bạn chỉ cần sửa một chỗ duy nhất trong file đó.

 Sai cách:

tsx

```
<Button variant="contained" /> // dùng khắp nơi
```

 Đúng cách:

tsx

`<MyButton />` // dùng khắp nơi → dễ thay đổi sau này

💡 Gợi ý cách tổ chức thư mục:

bash

src/

|— components/

| |— ui/ # nơi quản tất cả component từ thư viện ngoài

| | |— MyButton.tsx

| | |— MyInput.tsx

| | |— MyModal.tsx

✨ Bonus:

Bạn có thể thêm các logic như:

- Loading state
 - Accessibility
 - Animation
 - Theme tự động
 - Responsive mặc định
-

🏠 Tổng kết:

Lợi ích khi wrap

Giải thích



Tái sử dụng

Không cần lặp lại style / logic



Dễ bảo trì

Chỉ cần sửa 1 chỗ khi đổi thư viện UI



Thống nhất UI

Giao diện đồng bộ toàn hệ thống



Tùy biến dễ dàng

Dễ thêm logic riêng cho từng project

7. Sử dụng Component Libraries

Fully Featured (có sẵn style):


Tên	Mạnh về	Ghi chú
Chakra UI	Trải nghiệm tốt, prototyping nhanh	Rất dễ dùng
Ant Design (AntD)	Quản trị, bảng biểu	Hơi khó custom style
MUI	Phổ biến, nhiều components	Theo Material Design
Mantine	Rất nhiều component & hook	Tùy biến cao

Headless Component Libraries (không có style):

Tên	Dùng khi nào
Radix UI	Cần kiểm soát styling tuyệt đối
Headless UI	Kết hợp tốt với Tailwind
react-aria, Ark UI, Reakit	Hỗ trợ accessibility tốt

8. Các giải pháp styling nên dùng

Tên	Ưu điểm
Tailwind CSS	Tiện dụng, không cần viết CSS riêng
CSS Modules	Dễ hiểu, hỗ trợ tốt trong nhiều dự án
styled-components / emotion	Viết style bằng JS
vanilla-extract / Panda CSS	Tối ưu cho performance (zero-runtime)

 Với **React Server Components**, nên dùng **zero-runtime styling** (như Panda CSS, vanilla-extract).

9. Storybook – Tài liệu và phát triển component tách biệt





Storybook là công cụ giúp bạn:






- Xem trước từng component một cách riêng biệt
- Viết tài liệu cho các thành phần UI
- Test component theo nhiều trạng thái (hover, error, loading,...)

Lợi ích:




- Phát triển component **tách biệt khỏi ứng dụng**
- Dễ kiểm thử & demo cho team, PM, Designer

Tóm lại – 9 nguyên tắc vàng khi viết component React

Nguyên tắc	Giải thích
 Gắn nơi dùng (colocate)	Tránh để state/component xa vị trí dùng
 Tách nhỏ UI	Tránh lồng hàm render quá nhiều
 Style nhất quán	Dùng ESLint + Prettier
 Hạn chế props	Dùng composition hoặc tách nhỏ

 Trừu tượng shared component	Đảm bảo tái sử dụng, nhất quán
 Quán thư viện bên ngoài	Dễ thay đổi thư viện sau này
 Dùng component library	Tăng tốc, đỡ mất thời gian
 Styling hiện đại	Ưu tiên Tailwind, Panda, vanilla-extract
 Storybook	Phát triển & test component độc lập

1. Kebab-case (dấu gạch nối)

-  Dùng dấu **gạch ngang** - để nối các từ.
-  Chủ yếu dùng để đặt **tên file, folder**, đặc biệt trong frontend (HTML, CSS, JS).
-  Không được dùng cho tên biến hoặc hàm trong JavaScript vì không hợp lệ cú pháp.


♦ Ví dụ:

bash

product-detail-card.jsx

shopping-cart.ts

user-profile-form.css

 Ưu tiên dùng **kebab-case** cho:

- Tên file React component: **product-page.tsx**
- Tên CSS file: **checkout-form.module.css**
- Tên folder: **user-profile/**, **order-summary/**

2. PascalCase (viết hoa chữ cái đầu mỗi từ)

-  Mỗi từ đều **viết hoa chữ cái đầu**.

- 📁 Chủ yếu dùng để đặt **tên Component hoặc Class** trong JavaScript/TypeScript/React.

♦ **Ví dụ:**

tsx

```
// Component React
function UserProfileCard() {
  return <div>...</div>;
}
```

// Class

```
class ShoppingCartService {}
```

✅ Ưu tiên dùng **PascalCase** cho:

- Tên Component React: **HomePage.tsx**, **ProductCard.tsx**
- Tên class, interface: **UserService**, **ProductDTO**

* So sánh nhanh:

Kiểu đặt tên	Ví dụ	Dùng cho
kebab-case	product-detail.tsx	📁 file, folder
PascalCase	ProductDetail.tsx	🧱 component, class, interface
camelCase	getProductDetail()	🔧 biến, hàm (React hook, JS)

✅ **Gợi ý cho dự án React:**

- 📁 Folder: **user-profile**, **product-page**
- 📄 File component: **ProductCard.tsx**
- 🧱 Component name: **ProductCard**, **UserForm**
- ⚙️ Hook, biến: **useUserData**, **fetchProductList**

Nếu bạn đang xây 1 dự án e-commerce, nên thống nhất:

- 📁 Folder: `kebab-case`
- 📄 Component file: `PascalCase` (tên trùng với tên component)
- 📦 Component name: `PascalCase`
- 🧠 Biến/hàm/hook: `camelCase`

👉 Ví dụ:

```
cpp
src/
├── components/
│   ├── product-card/
│   │   ├── ProductCard.tsx
│   │   └── product-card.module.css
```