

Project Structure là gì?

Project Structure là cách bạn **tổ chức file và thư mục** trong một dự án phần mềm. Cấu trúc hợp lý giúp:

- **Dễ tìm kiếm**, bảo trì và mở rộng code
- Hỗ trợ **làm việc nhóm hiệu quả**
- Đảm bảo **tách biệt rõ ràng giữa các chức năng (features)**

Most of the code lives in the `src` folder and looks something like this:

```
src
|
+-- app           # application layer containing:
| |              # this folder might differ based on the meta framework used
| +-- routes      # application routes / can also be pages
| +-- app.tsx     # main application component
| +-- provider.tsx # application provider that wraps the entire application with different global providers - this might a
| +-- router.tsx  # application router configuration
+-- assets        # assets folder can contain all the static files such as images, fonts, etc.
|
+-- components    # shared components used across the entire application
|
+-- config        # global configurations, exported env variables etc.
|
+-- features      # feature based modules
|
+-- hooks         # shared hooks used across the entire application
|
+-- lib           # reusable libraries preconfigured for the application
|
+-- stores        # global state stores
|
+-- testing       # test utilities and mocks
|
+-- types         # shared types used across the application
|
+-- utils         # shared utility functions
```

Chi tiết một số thư mục chính

app/ – Lớp ứng dụng (Application Layer)

- Chứa cấu hình ứng dụng React (routing, provider...)
- Phụ thuộc framework bạn dùng (VD: Remix, Next.js, Vite SPA)

```
app/  
├─ app.tsx      # Component gốc toàn app  
├─ router.tsx   # Cấu hình react-router  
├─ provider.tsx # Bọc app với các context provider (theme, auth, v.v.)  
└─ routes/      # Các route (tùy framework)
```

features/ – Các tính năng tách biệt

Mỗi feature là **một module độc lập**, không phụ thuộc lẫn nhau.

Ví dụ:

```
features/  
├─ auth/  
├─ users/  
├─ discussions/  
├─ comments/  
└─ teams/
```

Bên trong mỗi feature (tùy nhu cầu):

```
awesome-feature/  
├─ api/      # API riêng cho feature này  
├─ assets/   # Ảnh, icon, logo riêng  
├─ components/ # Các component chỉ dùng trong feature này  
├─ hooks/    # Hooks riêng  
├─ stores/   # State riêng  
├─ types/    # Loại dữ liệu riêng  
└─ utils/    # Hàm tiện ích riêng
```

✓ **Chỉ tạo thư mục khi cần**, không cần đầy đủ nếu không dùng.

components/ – Các component dùng chung

components/
├── ui/ # Atom-level (Button, Input, Modal)
├── layout/ # Header, Footer, Sidebar
├── feedback/ # Alert, Toast, Spinner
└── form/ # FormField, Select, Checkbox

VD: `<Button />`, `<Modal />`, `<Avatar />`, v.v.

hooks/ – Hooks dùng chung

hooks/
├── useAuth.ts
├── useDebounce.ts
└── useOutsideClick.ts

VD: `useDarkMode`, `useDebounce`, `useMediaQuery`,...

lib/ – Thư viện nội bộ

lib/
├── axios.ts # Preconfigured axios instance
├── queryClient.ts # React Query client với cấu hình mặc định
├── i18n.ts # i18next config
└── stripe.ts # Stripe SDK wrapper

Ví dụ:

- `lib/axios.ts` → cấu hình sẵn Axios
 - `lib/auth.ts` → xử lý token, cookie...
-

stores/ – State toàn cục (global state)

stores/
├── authStore.ts
└── cartStore.ts

|— uiStore.ts

Dành cho các state dùng toàn hệ thống:

- `authStore.ts`
- `themeStore.ts`
- `notificationStore.ts`

`config/` – Cấu hình toàn hệ thống

`config/`

|— `env.ts` # Import biến môi trường từ `process.env` (hoặc `dotenv`)
|— `api.ts` # URL endpoint, version, timeout, etc.
|— `theme.ts` # Theme tokens hoặc cấu hình Tailwind

- API endpoint, base URL
- Tên app, default language
- Import biến từ `.env`

`types/` – TypeScript types dùng toàn app

`types/`

|— `user.ts`
|— `product.ts`
|— `order.ts`
|— `index.d.ts` # Nếu cần global type declaration

VD:

- `User.ts`
- `Team.ts`
- `ApiResponse.ts`

utils/ – Hàm tiện ích

utils/

```
|— formatDate.ts  
|— formatPrice.ts  
|— debounce.ts  
|— validateEmail.ts
```

📌 Không chứa side effect (axios, window, dom), thuần logic

VD:

- `formatDate.ts`
- `capitalize.ts`
- `downloadFile.ts`

testing/ – Mocks và test helpers

bash

testing/

```
|— mocks/  
|— utils/  
|— testWrapper.tsx # Wrap test với Providers nếu cần
```

📌 Dùng cho Jest / Vitest / React Testing Library

🚫 **Tránh import chéo giữa các feature**

❌ **Ví dụ không nên:**

ts

// Sai: Feature discussions gọi sang feature comments

```
import { CommentItem } from "@features/comments/components/CommentItem";
```

✅ Giải pháp:

Chỉ combine các feature ở tầng **app/** hoặc **pages/**, không để feature A gọi feature B.

Cách chặn bằng ESLint:

```
js
'import/no-restricted-paths': [
  'error',
  {
    zones: [
      {
        target: './src/features/comments',
        from: './src/features',
        except: ['./comments'],
      },
      {
        target: './src/features/discussions',
        from: './src/features',
        except: ['./discussions'],
      },
    ],
  },
],
```

📦 Về việc dùng Barrel File (index.ts)

- Trước đây thường gộp export vào **index.ts**:

```
ts
export * from "./Form";
export * from "./Card";
```

- Tuy nhiên với **Vite**, việc này gây khó khăn cho tree-shaking → nên **import trực tiếp** từng file bạn cần.

📝 **NOTE:** Không cần tạo tất cả các thư mục cho mỗi feature

✅ Ý nghĩa:

Bạn **không bắt buộc** phải tạo đủ các thư mục như `api/`, `hooks/`, `components/`, `types/`... trong từng feature.

Chỉ nên tạo khi cần

Ví dụ:

- Nếu feature **chỉ có 1 component đơn giản**, không cần tạo `hooks/`, `utils/`, `types/` làm gì cho nặng nề.
 - Nếu feature có xử lý gọi API → **mới nên có `api/`**
 - Tư tưởng: **Lean structure** – cấu trúc càng nhẹ càng tốt, mở rộng dần khi phức tạp hơn.
-

Có thể để API riêng ở ngoài `features/` nếu được dùng chung

Vì sao?

- Nếu bạn có **nhiều API dùng chung** giữa các feature, việc để API trong từng feature sẽ dẫn đến **code trùng lặp** hoặc **phụ thuộc chéo**.

Giải pháp:

Tạo thư mục riêng:

bash

`src/api/`

|— `auth.api.ts`

|— `user.api.ts`

|— `discussion.api.ts`

→ Import dùng ở nhiều feature mà vẫn đảm bảo **độc lập giữa các feature**.

Tránh dùng Barrel File (`index.ts`) trong features

Barrel file là gì?

Là file `index.ts` dùng để gom export:

```
ts
// src/features/users/index.ts
export * from './UserList';
export * from './UserDetail';
```

⚠ Vấn đề:

- Vite không tree-shake tốt với barrel files → file build bị lớn
- Dẫn đến **hiệu năng kém**, tải nhiều code không dùng

✅ Giải pháp:

Import **trực tiếp** từ file:

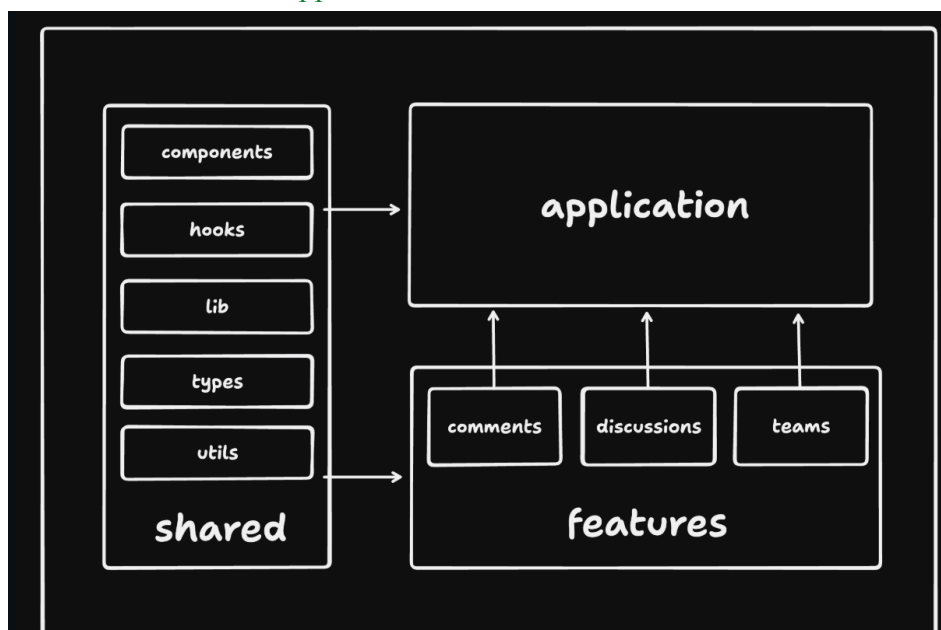
```
ts
import { UserList } from '@features/users/UserList';
```

🔄 Tuân thủ kiến trúc hướng một chiều (Unidirectional)

💡 Ý tưởng:

Dòng chảy import chỉ đi theo một hướng:

vbnet
shared → features → app



Nghĩa là:

- **app/** có thể import từ **features/** & **shared/**
- **features/** chỉ được import từ **shared/**
- **shared/** (hooks, components, utils, ...) **không bao giờ import ngược lại**






Lợi ích:

- Tránh vòng lặp phụ thuộc
- Rõ ràng luồng dữ liệu
- Dễ tái sử dụng hoặc scale từng phần riêng biệt

Cấu hình ESLint để ép buộc:

```
js
{
  target: './src/features',
  from: './src/app',
},
{
  target: ['./src/components', './src/hooks', './src/utils', './src/types'],
  from: ['./src/features', './src/app'],
}
```

Tại sao nên tổ chức theo feature?

Lợi ích	Giải thích
 Tách biệt rõ ràng	Mỗi tính năng nằm riêng, không lẫn lộn
 Dễ mở rộng	Thêm tính năng mới không ảnh hưởng cái cũ
 Dễ phân công team	Mỗi dev/team phụ trách 1 feature
 Dễ bảo trì & refactor	Ít bị ảnh hưởng đến phần khác khi sửa
 Gắn với thực tế dự án lớn	Được dùng nhiều trong kiến trúc micro-frontend hoặc monorepo

Ví dụ về cấu trúc một feature

bash

src/features/comments/

```
|— components/    # CommentItem, CommentList,...
|— hooks/         # useCommentForm, useCommentFetch,...
|— api/           # comment.api.ts
|— stores/        # commentStore.ts
|— types/         # Comment.ts
|— utils/         # formatCommentDate.ts
```

Tóm lại:

- **Feature** là một **module tính năng**, đại diện cho một phần riêng biệt trong ứng dụng.
- Việc tổ chức code theo **feature-based structure** giúp:
 - Dự án **gọn gàng, rõ ràng**
 - **Dễ scale**, dễ chia việc
 - **Ít lỗi** do giới hạn import chéo

apps/web/

```
|— public/        # Static files (logo, fonts, etc.)
|— src/
|   |— assets/     # Hình ảnh, SVG, v.v.
|   |— components/ # UI component (Button, Modal, Header, etc.)
|   |— features/   # Mỗi domain (cart, auth, product, order...) là 1 folder
|   |   |— cart/
|   |       |— api/      # Tách các request API dùng react-query
|   |       |— components/ # Component nội bộ
|   |       |— hooks/     # Custom hook: useCart()
|   |       |— cartSlice.ts # Redux hoặc Zustand/Jotai slice (nếu cần)
|   |— layouts/    # Các layout chính: GuestLayout, AuthLayout, etc.
|   |— pages/      # React Router pages: Home, Product, Checkout, etc.
|   |— routes/     # Route config + code splitting
|   |— services/   # API clients hoặc shared logic
|   |— stores/     # Global state (Zustand, Redux, etc.)
|   |— hooks/      # useAuth, useDebounce, useBreakpoint, etc.
|   |— lib/        # Helper: formatPrice, buildQueryParams, etc.
```

```

|   |   | styles/      # Tailwind config, globals.css
|   |   | types/      # TypeScript types riêng frontend
|   |   | main.tsx
|   | index.html
|   vite.config.ts

```

```

apps/server/
|   |   | src/
|   |   | |   | api/
|   |   | |   | |   | controllers/  # Business logic của từng API route
|   |   | |   | |   | routes/      # Express routes phân theo domain
|   |   | |   | |   | middlewares/  # Auth, logging, validation, etc.
|   |   | |   | |   | validators/   # Zod/JOI schemas validate request
|   |   | |   | services/          # Service layer: xử lý dữ liệu trước/sau controller
|   |   | |   | models/            # Các model custom (nếu không dùng ORM)
|   |   | |   | prisma/            # Prisma Client (nối với thư mục gốc /prisma)
|   |   | |   | utils/             # JWT, hash password, logger, etc.
|   |   | |   | jobs/              # Cron jobs: xử lý đơn hàng, email, etc.
|   |   | |   | config/            # Cấu hình app: DB, auth, CORS, mailer
|   |   | |   | index.ts           # Khởi chạy app
|   |   | |   | server.ts          # Tạo app Express
|   |   | tests/                   # Unit + integration test (Jest hoặc Vitest)
|   |   | .env
|   |   | tsconfig.json

```

✓ 1. Nếu bạn dùng React Vite / CRA (không có file-based routing):

👉 Các page nên được đặt trong thư mục **app/routes** như sau:

```

css
src/
|   |   | app/
|   |   | |   | routes/
|   |   | |   | |   | HomePage.tsx
|   |   | |   | |   | ProductPage.tsx
|   |   | |   | |   | CartPage.tsx
|   |   | |   | |   | NotFoundPage.tsx
|   |   | |   | app.tsx
|   |   | |   | provider.tsx
|   |   | |   | router.tsx  ← nơi cấu hình route với React Router

```

Trong `router.tsx`, bạn sẽ dùng `react-router-dom` để map đường dẫn tới các page:

```
import { Routes, Route } from 'react-router-dom';
import HomePage from './routes/HomePage';
import ProductPage from './routes/ProductPage';

export default function AppRouter() {
  return (
    <Routes>
      <Route path="/" element={<HomePage />} />
      <Route path="/products/:id" element={<ProductPage />} />
    </Routes>
  );
}
```

✅ 2. Nếu bạn dùng Next.js / Remix (file-based routing):

👉 Các page nên được đặt trong thư mục đặc biệt:

- Next.js: `src/pages/`
- Remix: `src/routes/`

Ví dụ:

```
bash
src/
├── pages/          ← Next.js tự hiểu đây là file-based routing
│   ├── index.tsx   → route /
│   ├── products.tsx → route /products
│   └── products/[id].tsx → dynamic route /products/:id
```

Lúc này, bạn **không cần viết `router.tsx`** vì framework tự làm việc đó.

✅ Tổng kết vị trí đặt Page:

Framework	Folder chứa page	Ghi chú
React (Vite, CRA)	<code>src/app/routes/</code>	Dùng <code>react-router-dom</code> , cấu hình router thủ công

Next.js	<code>src/pages/</code> hoặc <code>app/</code>	Sử dụng file-based routing
Remix	<code>src/routes/</code>	Cũng dùng file-based routing