

API Layer là gì?

API Layer là **lớp trung gian** giúp ứng dụng frontend (React) giao tiếp với backend (REST API, GraphQL). Nó là nơi bạn định nghĩa:

- ☒ Các request đến backend (GET, POST, PUT,...)
 - ☒ Kiểm tra dữ liệu trả về có đúng định dạng không
 - ☒ Tái sử dụng logic gọi API trong toàn ứng dụng
-

1. Dùng 1 instance API client duy nhất

 **Không nên:**

```
ts
fetch("/api/users")
axios.get("/api/products")
```

→ Gọi trực tiếp lặp đi lặp lại → khó cấu hình header, token, baseUrl

☒ **Nên tạo 1 instance được cấu hình sẵn:**

```
ts
// apiClient.ts
import axios from 'axios';

export const apiClient = axios.create({
  baseUrl: 'https://api.example.com',
  headers: {
    Authorization: 'Bearer your_token',
  },
});
```

☒ **Ưu điểm:**

- Không lặp cấu hình

- Quản lý header, interceptor dễ dàng
- Tái sử dụng trong toàn ứng dụng

2. Tách riêng và xuất các API request


Không nên:

Gọi API ngay trong component:

```
ts
useEffect(() => {
  fetch("/api/users").then(...);
}, []);
```

Nên:

- Định nghĩa riêng từng API trong file `api/`
- Xuất ra một **fetcher function** + **hook**

 Ví dụ cấu trúc thư mục:

```
css
src/
├─ api/
│  └─ users/
│     ├── getUsers.ts  ← định nghĩa API
│     └─ useUsers.ts   ← custom hook để gọi API
```

3. Cấu trúc chuẩn cho 1 API request

```
src/
├─ api/
│  └─ users/
│     ├── types.ts      ← định nghĩa kiểu dữ liệu
│     ├── getUsers.ts   ← hàm gọi API (fetcher)
│     └─ useUsers.ts    ← hook React Query dùng trong UI
├─ lib/
│  └─ apiClient.ts      ← cấu hình instance axios
```

Mỗi API nên có 3 phần:

① Kiểu dữ liệu (Type + Schema)

```
ts
// types.ts
export type User = {
  id: string;
  name: string;
};
```

(nếu dùng Zod, Joi,... có thể kiểm tra luôn format dữ liệu trả về)

② Fetcher function (chỉ gọi API, không dùng trong UI trực tiếp)

```
ts
// getUsers.ts
import { apiClient } from '../client';
import { User } from './types';

export async function getUsers(): Promise<User[]> {
  const res = await apiClient.get('/users');
  return res.data;
}
```

③ Hook gọi fetcher (React Query / SWR / Apollo...)

```
ts
// useUsers.ts
import { useQuery } from '@tanstack/react-query';
import { getUsers } from './getUsers';

export function useUsers() {
  return useQuery({
    queryKey: ['users'],
    queryFn: getUsers,
  });
}
```



Bước 1: Tạo **apiClient** dùng Axios

```
ts
// src/lib/apiClient.ts
import axios from 'axios';

export const apiClient = axios.create({
  baseURL: 'https://api.example.com', // thay đổi tùy server bạn
  headers: {
    'Content-Type': 'application/json',
  },
});
```

Bước 2: Định nghĩa kiểu dữ liệu (types.ts)

```
ts
// src/api/users/types.ts
export type User = {
  id: string;
  name: string;
  email: string;
};
```

Bước 3: Viết hàm gọi API (fetcher function)

```
ts
// src/api/users/getUsers.ts
import { apiClient } from '@lib/apiClient';
import { User } from './types';

export async function getUsers(): Promise<User[]> {
  const res = await apiClient.get('/users');
  return res.data; // Giả sử API trả về mảng User
}
```

Bước 4: Viết custom hook dùng React Query

```
ts
// src/api/users/useUsers.ts
import { useQuery } from '@tanstack/react-query';
import { getUsers } from './getUsers';
```

```
import { User } from './types';

export function useUsers() {
  return useQuery<User[], Error>({
    queryKey: ['users'],
    queryFn: getUsers,
    staleTime: 1000 * 60 * 5, // 5 phút cache
  });
}
```

Bước 5: Sử dụng trong component UI




```
tsx
// src/pages/UserList.tsx
import { useUsers } from '@api/users/useUsers';

export default function UserList() {
  const { data: users, isLoading, error } = useUsers();

  if (isLoading) return <div>Đang tải...</div>;
  if (error) return <div>Lỗi: {error.message}</div>;

  return (
    <ul>
      {users?.map((user) => (
        <li key={user.id}>{user.name} ({user.email})</li>
      ))}
    </ul>
  );
}
```

Một số mẹo mở rộng

Kỹ thuật	Ví dụ
 Tự động refetch	<code>refetchInterval: 10000</code> trong <code>useQuery()</code>
 Dừng mutation	<code>useMutation()</code> cho POST, PUT, DELETE
 Hiển thị lỗi đẹp	Kết hợp <code>toast</code> , <code>react-toastify</code> , <code>notistack</code> ...

💬 Tùy chỉnh queryKey `['users', teamId]` để phân biệt cache

🔒 Sử dụng token Thêm vào `apiClient.defaults.headers`

✅ Tổng kết lại luồng sử dụng:

1. 📌 **apiClient.ts**: cấu hình Axios 1 lần duy nhất
2. 📄 **getUsers.ts**: viết hàm gọi API và trả dữ liệu về
3. 🔄 **useUsers.ts**: dùng React Query để cache, loading, error...
4. 🎨 **Component** chỉ cần gọi `useUsers()` là có dữ liệu dùng ngay

✨ Lợi ích khi làm API layer như trên:

✅ Lợi ích 💬 Giải thích

- 📁 Gọn gàng Code API tách biệt, dễ bảo trì
 - 🔄 Dễ reuse Gọi ở đâu cũng dùng được chung hook
 - 📄 Type-safe Dễ kiểm tra dữ liệu trả về có đúng không
 - 🔧 Dễ test Có thể test từng hàm fetcher riêng
 - ⚡ Hiệu năng Dễ cache, quản lý loading/error với React Query
-

🧠 Tổng kết

Nguyên tắc	Giải thích
📌 1 client duy nhất	Dùng <code>axios.create()</code> hoặc <code>GraphQLClient()</code> cấu hình sẵn
📄 Tách request ra file riêng	Mỗi API → types + fetcher + hook
🔗 Dùng thư viện quản lý dữ liệu	Dùng <code>react-query</code> , <code>swr</code> , <code>apollo</code> , <code>urql</code> ,...
🔒 Gỡ kiểu dữ liệu rõ ràng	Tránh lỗi runtime khi dữ liệu không khớp