

Để mỗi hình ảnh khi click vào sẽ dẫn đến một trang ứng dụng khác, bạn có thể cập nhật thuộc tính href trong JSON để chứa đường dẫn đến các trang tương ứng. Sau đó, sử dụng giá trị này trong thẻ <a> để điều hướng người dùng đến trang đúng khi họ nhấn vào hình ảnh.

**1. Cập nhật JSON**

Cập nhật file JSON của bạn để mỗi đối tượng chứa URL tương ứng với trang ứng dụng:

json

Copy code

[

{

"href": "/automotive", // URL của ứng dụng này

"p": "Automotive",

"img": "https://epicpadprinting.com/public/img/indus/Automotive.png"

},

{

"href": "/technology",

"p": "Technology",

"img": "https://example.com/img/technology.png"

},

{

"href": "/healthcare",

"p": "Healthcare",

"img": "https://example.com/img/healthcare.png"

}

]

**2. Cập nhật React Component**

Sử dụng giá trị href từ JSON để điều hướng khi người dùng click vào hình ảnh:

javascript

Copy code

import React, { useState, useEffect } from "react";

export default function Ungdung() {

const [data, setData] = useState([]);

useEffect(() => {

fetch("/application.json")

.then(response => {

if (!response.ok) {

throw new Error(`HTTP error! Status: ${response.status}`);

}

return response.json(); // Đọc dữ liệu JSON

})

.then(json => setData(json))

.catch(error => console.error("Có lỗi xảy ra khi fetch dữ liệu: ", error));

}, []);

return (

<section className="bg-white py-24 px-4 lg:px-16">

<div className="container mx-auto px-[12px] md:px-24 xl:px-12 max-w-[1300px] nanum2">

<h1 className="text-center text-5xl pb-12">Danh sách ứng dụng</h1>

<div className="grid grid-cols-1 lg:grid-cols-2 xl:grid-cols-3 gap-x-4 gap-y-28 lg:gap-y-16">

{data.map((item, index) => (

<div key={index} className="relative group h-48 flex flex-col rounded-xl bg-white bg-clip-border text-gray-700 shadow-md">

<a href={item.href} className="block">

<div className="h-28">

<div className="absolute -top-20 lg:top-[-10%] left-[5%] z-40 group-hover:top-[-40%] group-hover:opacity-[0.9] duration-300 w-[90%] h-48 bg-red-500 rounded-xl justify-items-center align-middle">

<img src={item.img} className="w-36 h-36 mt-6 m-auto" alt={item.p} title={item.p} loading="lazy" width="200" height="200"/>

</div>

</div>

<div className="p-6 z-10 w-full">

<p className="mb-2 inline-block text-tg text-center w-full text-xl font-sans font-semibold leading-snug tracking-normal antialiased">

{item.p}

</p>

</div>

</a>

</div>

))}

</div>

</div>

</section>

);

}

**3. Xử lý điều hướng trong React**

Nếu bạn muốn điều hướng trong ứng dụng React mà không tải lại trang, bạn có thể sử dụng react-router-dom. Dưới đây là cách sử dụng:

* **Cài đặt react-router-dom:**

bash

Copy code

npm install react-router-dom

* **Sử dụng Link thay vì thẻ <a>:**

javascript

Copy code

import React, { useState, useEffect } from "react";

import { Link } from "react-router-dom"; // Import Link từ react-router-dom

export default function Ungdung() {

const [data, setData] = useState([]);

useEffect(() => {

fetch("/application.json")

.then(response => {

if (!response.ok) {

throw new Error(`HTTP error! Status: ${response.status}`);

}

return response.json(); // Đọc dữ liệu JSON

})

.then(json => setData(json))

.catch(error => console.error("Có lỗi xảy ra khi fetch dữ liệu: ", error));

}, []);

return (

<section className="bg-white py-24 px-4 lg:px-16">

<div className="container mx-auto px-[12px] md:px-24 xl:px-12 max-w-[1300px] nanum2">

<h1 className="text-center text-5xl pb-12">Danh sách ứng dụng</h1>

<div className="grid grid-cols-1 lg:grid-cols-2 xl:grid-cols-3 gap-x-4 gap-y-28 lg:gap-y-16">

{data.map((item, index) => (

<div key={index} className="relative group h-48 flex flex-col rounded-xl bg-white bg-clip-border text-gray-700 shadow-md">

<Link to={item.href} className="block"> {/\* Sử dụng Link thay vì thẻ <a> \*/}

<div className="h-28">

<div className="absolute -top-20 lg:top-[-10%] left-[5%] z-40 group-hover:top-[-40%] group-hover:opacity-[0.9] duration-300 w-[90%] h-48 bg-red-500 rounded-xl justify-items-center align-middle">

<img src={item.img} className="w-36 h-36 mt-6 m-auto" alt={item.p} title={item.p} loading="lazy" width="200" height="200"/>

</div>

</div>

<div className="p-6 z-10 w-full">

<p className="mb-2 inline-block text-tg text-center w-full text-xl font-sans font-semibold leading-snug tracking-normal antialiased">

{item.p}

</p>

</div>

</Link>

</div>

))}

</div>

</div>

</section>

);

}

Với cách tiếp cận này, mỗi khi người dùng nhấp vào hình ảnh, họ sẽ được điều hướng đến trang ứng dụng tương ứng mà không cần phải tải lại toàn bộ trang web.

Nếu bạn sử dụng PostgreSQL thay vì file JSON, thì dữ liệu sẽ được lưu trữ và truy xuất từ cơ sở dữ liệu thay vì từ một file tĩnh. Dưới đây là cách bạn có thể chuyển đổi từ sử dụng file JSON sang PostgreSQL.

**1. Thiết kế bảng trong PostgreSQL**

Trước tiên, bạn cần tạo một bảng trong PostgreSQL để lưu trữ dữ liệu mà trước đây bạn lưu trong file JSON. Ví dụ:

sql

Copy code

CREATE TABLE applications (

id SERIAL PRIMARY KEY,

href VARCHAR(255) NOT NULL,

p VARCHAR(255) NOT NULL,

img VARCHAR(255) NOT NULL

);

**2. Chèn dữ liệu vào bảng**

Bạn có thể chèn dữ liệu vào bảng này bằng lệnh INSERT:

sql

Copy code

INSERT INTO applications (href, p, img)

VALUES

('/automotive', 'Automotive', 'https://epicpadprinting.com/public/img/indus/Automotive.png'),

('/technology', 'Technology', 'https://example.com/img/technology.png'),

('/healthcare', 'Healthcare', 'https://example.com/img/healthcare.png');

**3. Kết nối và truy xuất dữ liệu từ PostgreSQL trong React**

Trong ứng dụng ReactJS, bạn cần sử dụng một REST API để kết nối và truy xuất dữ liệu từ PostgreSQL. Bạn có thể sử dụng Node.js với Express để tạo một API.

**Ví dụ tạo API với Node.js và Express:**

* **Bước 1: Cài đặt các package cần thiết:**

bash

Copy code

npm install express pg cors

* **Bước 2: Tạo một API cơ bản để lấy dữ liệu từ PostgreSQL:**

javascript

Copy code

const express = require('express');

const cors = require('cors');

const { Pool } = require('pg');

const app = express();

const pool = new Pool({

user: 'your\_username',

host: 'localhost',

database: 'your\_database',

password: 'your\_password',

port: 5432,

});

app.use(cors());

app.get('/applications', async (req, res) => {

try {

const result = await pool.query('SELECT \* FROM applications');

res.json(result.rows);

} catch (error) {

console.error(error);

res.status(500).send('Server error');

}

});

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => console.log(`Server running on port ${PORT}`));

**4. Fetch dữ liệu từ API trong React**

Trong ứng dụng React của bạn, thay vì fetch dữ liệu từ file JSON, bạn sẽ fetch từ API:

javascript

Copy code

import React, { useState, useEffect } from "react";

export default function Ungdung() {

const [data, setData] = useState([]);

useEffect(() => {

fetch("http://localhost:5000/applications") // Địa chỉ API của bạn

.then(response => response.json())

.then(json => setData(json))

.catch(error => console.error("Có lỗi xảy ra khi fetch dữ liệu: ", error));

}, []);

return (

<section className="bg-white py-24 px-4 lg:px-16">

<div className="container mx-auto px-[12px] md:px-24 xl:px-12 max-w-[1300px] nanum2">

<h1 className="text-center text-5xl pb-12">Danh sách ứng dụng</h1>

<div className="grid grid-cols-1 lg:grid-cols-2 xl:grid-cols-3 gap-x-4 gap-y-28 lg:gap-y-16">

{data.map((item, index) => (

<div key={index} className="relative group h-48 flex flex-col rounded-xl bg-white bg-clip-border text-gray-700 shadow-md">

<a href={item.href} className="block">

<div className="h-28">

<div className="absolute -top-20 lg:top-[-10%] left-[5%] z-40 group-hover:top-[-40%] group-hover:opacity-[0.9] duration-300 w-[90%] h-48 bg-red-500 rounded-xl justify-items-center align-middle">

<img src={item.img} className="w-36 h-36 mt-6 m-auto" alt={item.p} title={item.p} loading="lazy" width="200" height="200"/>

</div>

</div>

<div className="p-6 z-10 w-full">

<p className="mb-2 inline-block text-tg text-center w-full text-xl font-sans font-semibold leading-snug tracking-normal antialiased">

{item.p}

</p>

</div>

</a>

</div>

))}

</div>

</div>

</section>

);

}

**5. Lợi ích của việc chuyển sang PostgreSQL:**

* **Quản lý dữ liệu linh hoạt**: Dễ dàng thêm, sửa, xóa và truy xuất dữ liệu.
* **Tăng khả năng mở rộng**: Có thể xử lý lượng dữ liệu lớn và mở rộng ứng dụng dễ dàng.
* **Tích hợp với các dịch vụ khác**: Dễ dàng tích hợp với các công nghệ khác trong hệ sinh thái backend.

Bằng cách chuyển đổi như trên, bạn đã sẵn sàng để mở rộng ứng dụng với nhiều dữ liệu và tính năng hơn trong tương lai.

File App.js của bạn đã được cấu trúc khá tốt để quản lý điều hướng và xác thực người dùng. Để tích hợp các trang ứng dụng từ JSON như đã thảo luận trước đó, bạn cần đảm bảo rằng mỗi ứng dụng sẽ có một route tương ứng và việc điều hướng sẽ dựa trên các liên kết được tạo động từ dữ liệu JSON.

### Cập nhật App.js

Nếu bạn có thêm các trang ứng dụng khác như Automotive, Technology, Healthcare được tạo dựa trên JSON, bạn cần thêm các route tương ứng trong phần Routes.

Dưới đây là ví dụ cập nhật:

javascript

Copy code

import React, { useState } from "react";

import Layoutmenu from "./compoments/Layout/Layoutmenu";

import Headermain from "./compoments/Layout/Headermain";

import { BrowserRouter as Router, Routes, Route, Navigate } from "react-router-dom";

import Trangchu from "./compoments/Pages/Trangchu";

import Quidinh from "./compoments/Pages/quidinh";

import Huongdan from "./compoments/Pages/huongdan";

import Thongbao from "./compoments/Pages/thongbao";

import Ungdung from "./compoments/Pages/ungdung";

import Congdoan from "./compoments/Pages/congdoan";

import Hinhanh from "./compoments/Pages/hinhanh";

import Admin from "./compoments/Pages/admin";

import Login from "./compoments/Pages/login/login";

// Import các component trang ứng dụng khác

import Automotive from "./compoments/Pages/application/account/Automotive";

import Technology from "./compoments/Pages/application/account/Technology";

import Healthcare from "./compoments/Pages/application/account/Healthcare";

function App() {

const [isAuthenticated, setIsAuthenticated] = useState(false);

const handleLogin = () => {

setIsAuthenticated(true);

};

return (

<Router>

{isAuthenticated ? (

<div>

<Headermain />

<Layoutmenu />

<Routes>

<Route path="/" element={<Trangchu />} />

<Route path="/Quidinh" element={<Quidinh />} />

<Route path="/Huongdan" element={<Huongdan />} />

<Route path="/Thongbao" element={<Thongbao />} />

<Route path="/Ungdung" element={<Ungdung />} />

<Route path="/Congdoan" element={<Congdoan />} />

<Route path="/Hinhanh" element={<Hinhanh />} />

<Route path="/Admin" element={<Admin />} />

{/\* Các route ứng dụng khác \*/}

<Route path="/automotive" element={<Automotive />} />

<Route path="/technology" element={<Technology />} />

<Route path="/healthcare" element={<Healthcare />} />

</Routes>

</div>

) : (

<Routes>

<Route path="/\*" element={<Navigate to="/login" />} />

<Route path="/login" element={<Login onLogin={handleLogin} />} />

</Routes>

)}

</Router>

);

}

export default App;

### Lưu ý:

1. **Thêm Component mới:** Đảm bảo rằng bạn đã tạo các component như Automotive, Technology, và Healthcare trong thư mục src/compoments/Pages/application/account.
2. **JSON và Fetch Data:** Đối với các trang ứng dụng có thể tạo động từ dữ liệu JSON, bạn nên quản lý các route và nội dung từ dữ liệu JSON một cách động, hoặc thông qua cách fetch dữ liệu từ JSON hoặc cơ sở dữ liệu như PostgreSQL.

### Chuyển JSON thành Cấu trúc cho PostgreSQL:

Nếu bạn muốn lưu trữ dữ liệu này trong PostgreSQL thay vì JSON, bạn sẽ cần chuyển các trường từ JSON thành các cột trong bảng PostgreSQL. Ví dụ:

sql

Copy code

CREATE TABLE applications (

id SERIAL PRIMARY KEY,

href VARCHAR(255),

title VARCHAR(255),

img\_url VARCHAR(255)

);

Sau đó, bạn có thể lưu dữ liệu từ JSON vào bảng này và fetch dữ liệu từ PostgreSQL thông qua API NodeJS để sử dụng trong React.

Cấu trúc mà bạn đã thiết lập khá ổn cho một số lượng ứng dụng vừa phải, nhưng nếu bạn dự định có nhiều ứng dụng trong tương lai, bạn nên cân nhắc một số cải tiến sau:

### 1. ****Quản lý Route Động (Dynamic Routing):****

Thay vì xác định cứng các route trong App.js, bạn có thể quản lý các route động dựa trên dữ liệu lấy từ JSON hoặc cơ sở dữ liệu. Điều này sẽ giúp bạn dễ dàng mở rộng ứng dụng mà không cần phải chỉnh sửa mã nguồn mỗi khi thêm một ứng dụng mới.

javascript

Copy code

import React, { useState, useEffect } from "react";

import { BrowserRouter as Router, Routes, Route, Navigate } from "react-router-dom";

import Headermain from "./compoments/Layout/Headermain";

import Layoutmenu from "./compoments/Layout/Layoutmenu";

import Trangchu from "./compoments/Pages/Trangchu";

import Login from "./compoments/Pages/login/login";

import axios from "axios";

function App() {

const [isAuthenticated, setIsAuthenticated] = useState(false);

const [appRoutes, setAppRoutes] = useState([]);

const handleLogin = () => {

setIsAuthenticated(true);

};

useEffect(() => {

// Fetching the applications from JSON or API

axios.get('/path/to/your/applications.json')

.then(response => {

setAppRoutes(response.data);

})

.catch(error => {

console.error("Error fetching application routes:", error);

});

}, []);

return (

<Router>

{isAuthenticated ? (

<div>

<Headermain />

<Layoutmenu />

<Routes>

<Route path="/" element={<Trangchu />} />

{appRoutes.map(app => (

<Route key={app.href} path={app.href} element={<DynamicComponent componentName={app.p} />} />

))}

</Routes>

</div>

) : (

<Routes>

<Route path="/\*" element={<Navigate to="/login" />} />

<Route path="/login" element={<Login onLogin={handleLogin} />} />

</Routes>

)}

</Router>

);

}

function DynamicComponent({ componentName }) {

const Component = React.lazy(() => import(`./compoments/Pages/application/account/${componentName}`));

return (

<React.Suspense fallback={<div>Loading...</div>}>

<Component />

</React.Suspense>

);

}

export default App;

### 2. ****Sử dụng Cơ sở Dữ liệu:****

Nếu bạn chuyển sang sử dụng PostgreSQL hoặc một cơ sở dữ liệu khác, bạn có thể xây dựng một API với NodeJS để quản lý các ứng dụng này. Khi ứng dụng tăng lên, việc quản lý thông tin về các ứng dụng qua cơ sở dữ liệu sẽ giúp bạn duy trì và mở rộng một cách dễ dàng hơn.

### 3. ****Tối Ưu Hóa Hiệu Suất:****

Khi bạn có nhiều ứng dụng, bạn cũng nên cân nhắc đến việc tối ưu hóa tải trang bằng cách sử dụng React.lazy và React.Suspense để tải các component chỉ khi cần thiết.

### 4. ****Cấu trúc thư mục****:

Với số lượng ứng dụng lớn, bạn có thể sắp xếp các component ứng dụng thành các thư mục con dựa trên các danh mục hoặc nhóm tính năng, giúp quản lý và tìm kiếm dễ dàng hơn.

Bằng cách tổ chức route động và sử dụng cơ sở dữ liệu, bạn sẽ dễ dàng mở rộng và quản lý một lượng lớn ứng dụng mà không cần thay đổi cấu trúc chính của mã nguồn.

NodeJS backend

#### Khởi tạo file package.json

Chạy lệnh dưới đây để khởi tạo file package.json cho dự án NodeJS của bạn. File này sẽ quản lý các gói và cấu hình của dự án:

npm init -y

Điều này sẽ tạo ra file package.json với cấu hình mặc định. Bạn có thể chỉnh sửa file này sau nếu cần.

#### Cài đặt các gói cần thiết

Một số gói phổ biến thường được cài đặt để xây dựng backend NodeJS, chẳng hạn như:

* **Express.js**: Một framework phổ biến để xây dựng API và ứng dụng web.
* **dotenv**: Quản lý biến môi trường.
* **nodemon**: Công cụ giám sát thay đổi file và tự động restart server trong quá trình phát triển.

Cài đặt các gói này:

npm install express dotenv

npm install --save-dev nodemon

npm install express pg // Thư viện postgres

Nếu bạn đã có thư mục **frontend** riêng để chạy ReactJS, cấu trúc dự án sẽ tách biệt giữa **backend** và **frontend**. Điều này giúp dễ quản lý và phát triển khi cả hai phần hoạt động độc lập với nhau.

Dưới đây là gợi ý về cách sắp xếp thư mục **backend** và **frontend** cho ứng dụng của bạn.

### Cấu trúc tổng thể dự án

bash

Copy code

/project-root

├── /backend # Thư mục backend (API server, database, authentication, etc.)

│ ├── /config # Cấu hình server và database

│ ├── /controllers # Controller để xử lý logic

│ ├── /models # Mô hình dữ liệu (ORM models)

│ ├── /routes # Định nghĩa API routes

│ ├── /services # Business logic

│ ├── /middleware # Middleware dùng chung

│ ├── /utils # Các tiện ích chung

│ ├── /tests # Unit và integration test

│ ├── app.js # File khởi tạo ứng dụng Express

│ └── package.json # Package quản lý cho backend

├── /frontend # Thư mục frontend (ReactJS)

│ ├── /public # Static files như ảnh, favicon, etc.

│ ├── /src # Source code của React

│ │ ├── /components # Các component ReactJS

│ │ ├── /pages # Các trang ReactJS

│ │ ├── /utils # Các tiện ích cho frontend

│ │ ├── App.js # Entry point của ReactJS

│ │ ├── index.js # Main file của React

│ └── package.json # Package quản lý cho frontend

└── README.md # Hướng dẫn dự án

### Cách hoạt động

1. **Backend**:
   * **Backend** sẽ chịu trách nhiệm cung cấp API cho **frontend** thông qua các route RESTful. Bạn sẽ sử dụng thư viện như **Express** để xây dựng API trong NodeJS.
   * Phần kết nối với PostgreSQL và các thao tác với cơ sở dữ liệu như tb\_menu sẽ được thực hiện tại đây.
2. **Frontend**:
   * **Frontend** với **ReactJS** sẽ gửi yêu cầu HTTP đến **backend** để lấy dữ liệu từ PostgreSQL (qua API).
   * **Frontend** sẽ hiển thị các dữ liệu như menu được cung cấp từ **backend** (thay thế cho file JSON tĩnh).

### Triển khai backend

#### 1. **Cập nhật** menuController.js **để lấy dữ liệu từ cơ sở dữ liệu PostgreSQL**:

js

Copy code

const menuService = require("../services/menuService");

exports.getAllMenus = async (req, res) => {

try {

const menus = await menuService.getMenus();

res.json(menus); // Trả về JSON chứa danh sách menu

} catch (error) {

res.status(500).json({ error: "Failed to fetch menus" });

}

};

#### 2. **Dịch vụ** menuService.js **kết nối với PostgreSQL**:

js

Copy code

const db = require("../config/db");

exports.getMenus = async () => {

const result = await db.query("SELECT \* FROM tb\_menu");

return result.rows;

};

#### 3. **Định nghĩa route trong** menuRoutes.js:

js

Copy code

const express = require("express");

const menuController = require("../controllers/menuController");

const router = express.Router();

router.get("/", menuController.getAllMenus); // API endpoint: /api/menu

module.exports = router;

### Triển khai frontend

#### 1. **Cập nhật** Layoutmenu.js **để lấy dữ liệu từ API thay vì JSON tĩnh**:

js

Copy code

import React, { useEffect, useState } from "react";

import { Disclosure, DisclosureButton, DisclosurePanel } from "@headlessui/react";

import { Bars3Icon, XMarkIcon } from "@heroicons/react/24/outline";

import { Link } from "react-router-dom";

import axios from "axios";

function classNames(...classes) {

return classes.filter(Boolean).join(" ");

}

export default function Layoutmenu() {

const [navigation, setNavigation] = useState([]);

useEffect(() => {

// Lấy dữ liệu từ API backend thay vì file JSON

axios

.get("http://localhost:3001/api/menu") // API endpoint của backend

.then((response) => setNavigation(response.data))

.catch((error) => console.error("Error loading menu data:", error));

}, []);

return (

<Disclosure as="nav" className="bg-gray-800 items-center">

<div className="mx-auto max-w-7xl px-2 sm:px-6 lg:px-8">

<div className="relative flex h-16 items-center justify-between">

<div className="absolute inset-y-0 left-0 flex items-center sm:hidden">

{/\* Mobile menu button \*/}

<DisclosureButton className="group relative inline-flex items-center justify-center rounded-md p-2 text-gray-400 hover:bg-gray-700 hover:text-white focus:outline-none focus:ring-2 focus:ring-inset focus:ring-white">

<span className="sr-only">Open main menu</span>

<Bars3Icon

aria-hidden="true"

className="block h-6 w-6 group-data-[open]:hidden"

/>

<XMarkIcon

aria-hidden="true"

className="hidden h-6 w-6 group-data-[open]:block"

/>

</DisclosureButton>

</div>

<div className="flex flex-1 items-center justify-center sm:items-stretch sm:justify-start">

<div className="hidden sm:ml-6 sm:block">

<div className="flex space-x-4">

{navigation.map((item) => (

<Link

key={item.id\_menu}

to={item.href}

className={classNames(

item.current\_menu

? "bg-gray-900 text-white"

: "text-gray-300 hover:bg-gray-700 hover:text-white",

"rounded-md px-3 py-2 text-sm font-medium"

)}

>

{item.name}

</Link>

))}

</div>

</div>

</div>

</div>

</div>

<DisclosurePanel className="sm:hidden">

<div className="space-y-1 px-2 pb-3 pt-2">

{navigation.map((item) => (

<DisclosureButton

key={item.id\_menu}

as={Link}

to={item.href}

aria-current={item.current\_menu ? "page" : undefined}

className={classNames(

item.current\_menu

? "bg-gray-900 text-white"

: "text-gray-300 hover:bg-gray-700 hover:text-white",

"block rounded-md px-3 py-2 text-base font-medium"

)}

>

{item.name}

</DisclosureButton>

))}

</div>

</DisclosurePanel>

</Disclosure>

);

}

### Các bước triển khai cụ thể:

1. **Backend**:
   * Đảm bảo rằng API backend hoạt động đúng với các route đã thiết lập (ví dụ: /api/menu).
   * Chạy backend trên một cổng như 3001.
2. **Frontend**:
   * Thay vì lấy dữ liệu từ file JSON tĩnh, frontend sẽ gửi yêu cầu đến API (http://localhost:3001/api/menu) để lấy dữ liệu từ PostgreSQL.
   * Bạn có thể chạy frontend ReactJS bằng cách cài đặt npm và khởi động với lệnh npm start.

### Kết luận:

* Với cấu trúc **frontend** và **backend** tách biệt, bạn có thể dễ dàng mở rộng và phát triển hệ thống mà không ảnh hưởng đến các phần khác.
* Mọi dữ liệu từ PostgreSQL đều được cung cấp qua API, frontend chỉ cần gọi API để lấy dữ liệu, giúp việc bảo trì và cập nhật dữ liệu linh hoạt hơn.