## **Hướng dẫn Tích hợp API Backend cho Component CalendarSchedule**

Component CalendarSchedule hiện đang sử dụng dữ liệu giả lập (mockData) và các hàm API giả lập (mockApi). Để kết nối với backend thực tế, bạn cần thực hiện các bước sau ở phía frontend và định nghĩa các yêu cầu cho backend.

### **I. Các Bước Thực Hiện Phía Frontend**

1. **Quản lý State cho Dữ liệu:**
   * Thay vì import trực tiếp scheduleEvents và teachers từ mockData, hãy sử dụng useState để lưu trữ dữ liệu này.
   * Thêm state để quản lý trạng thái tải dữ liệu (loading) và lỗi (error).

| **import** React, { useState, useEffect } **from** "react"; // ... (các import khác) // Bỏ import scheduleEvents, teachers từ mockData // import { updateScheduleEvent, deleteScheduleEvent } from "@/lib/api"; // Bỏ import này nếu API thật dùng fetch/axios  // ... (trong component) **const** [events, setEvents] = useState<any[]>([]); // State cho danh sách sự kiện **const** [teacherList, setTeacherList] = useState<any[]>([]); // State cho danh sách giáo viên **const** [isLoading, setIsLoading] = useState(**true**); // State cho trạng thái loading **const** [error, setError] = useState<string | **null**>(**null**); // State cho lỗi // ... (các state khác: currentDate, selectedEvent, v.v.) |
| --- |

1. **Lấy Dữ liệu Ban đầu (useEffect):**
   * Sử dụng hook useEffect để gọi API lấy danh sách sự kiện (scheduleEvents) và danh sách giáo viên (teachers) khi component được mount lần đầu.
   * Cập nhật state events, teacherList, isLoading, và error dựa trên kết quả trả về từ API.

| useEffect(() => {  **const** fetchData = **async** () => {  setIsLoading(**true**);  setError(**null**);  **try** {  // Gọi API lấy danh sách giáo viên  **const** teacherResponse = **await** fetch('/api/teachers'); // Thay đổi URL API nếu cần  **if** (!teacherResponse.ok) **throw** **new** Error('Không thể tải danh sách giáo viên');  **const** teachersData = **await** teacherResponse.json();  setTeacherList(teachersData);   // Gọi API lấy danh sách lịch học (có thể cần truyền tham số tuần/ngày)  **const** eventResponse = **await** fetch('/api/schedule'); // Thay đổi URL API nếu cần  **if** (!eventResponse.ok) **throw** **new** Error('Không thể tải lịch học');  **const** eventsData = **await** eventResponse.json();  setEvents(eventsData);   } **catch** (err: any) {  setError(err.message);  console.error("Fetch data error:", err);  } **finally** {  setIsLoading(**false**);  }  };   fetchData(); }, []); // Mảng dependency rỗng để chỉ chạy 1 lần khi mount |
| --- |

1. **Thay thế Mock Data Bằng State:**
   * Trong phần JSX, thay thế tất cả các chỗ sử dụng scheduleEvents bằng events và teachers bằng teacherList.

| // Ví dụ trong phần render Teacher Legend {teacherList.map((teacher) => ( /\* ... \*/ ))}  // Ví dụ trong phần lọc eventsForSlot **const** eventsForSlot = events.filter( /\* ... \*/ );  // Ví dụ trong phần tìm teacher cho event **const** teacher = teacherList.find((t) => t.id === event.teacherId)!; |
| --- |

1. **Cập nhật Hàm Xử lý Sự kiện (CRUD):**
   * **handleSaveClick** (Update):
     + Thay vì gọi updateScheduleEvent từ mock API, hãy sử dụng fetch (hoặc Axios) để gửi request PUT hoặc PATCH đến endpoint API cập nhật sự kiện với selectedEvent làm body.
     + Sau khi thành công, cập nhật lại state events (có thể bằng cách cập nhật trực tiếp phần tử đã sửa hoặc gọi lại API lấy danh sách sự kiện).
   * **handleDeleteClick** (Delete):
     + Thay vì gọi deleteScheduleEvent từ mock API, hãy sử dụng fetch (hoặc Axios) để gửi request DELETE đến endpoint API xóa sự kiện với selectedEvent.id.
     + Sau khi thành công, cập nhật lại state events bằng cách lọc bỏ sự kiện đã xóa.
   * **Xử lý Thêm sự kiện (Nếu có):** Tương tự, tạo hàm gửi request POST đến API để tạo sự kiện mới và cập nhật state events.

| // Ví dụ sửa đổi handleDeleteClick **const** handleDeleteClick = **async** () => {  **if** (!selectedEvent) **return**;  setIsLoading(**true**); // Có thể thêm loading cục bộ cho dialog  **try** {  **const** response = **await** fetch(`/api/schedule/${selectedEvent.id}`, { // Thay đổi URL API  method: 'DELETE',  });   **if** (!response.ok) {  **throw** **new** Error('Xóa lịch học thất bại');  }   // Cập nhật state events sau khi xóa thành công  setEvents(prevEvents => prevEvents.filter(event => event.id !== selectedEvent.id));   toast({ /\* ... \*/ });  setIsDialogOpen(**false**);  setSelectedEvent(**null**);  } **catch** (error: any) {  console.error("Error deleting event:", error);  toast({ variant: "destructive", /\* ... \*/ });  } **finally** {  setIsLoading(**false**); // Tắt loading cục bộ  } };  // Ví dụ sửa đổi handleSaveClick  **const** handleSaveClick = **async** () => {  **if** (!selectedEvent) **return**;  setIsLoading(**true**);  **try** {  **const** response = **await** fetch(`/api/schedule/${selectedEvent.id}`, { // Thay đổi URL API  method: 'PUT', // Hoặc PATCH  headers: {  'Content-Type': 'application/json',  },  body: JSON.stringify(selectedEvent), // Gửi dữ liệu đã chỉnh sửa  });   **if** (!response.ok) {  **throw** **new** Error('Cập nhật lịch học thất bại');  }   **const** updatedEvent = **await** response.json(); // Backend nên trả về event đã cập nhật   // Cập nhật state events  setEvents(prevEvents =>  prevEvents.map(event =>  event.id === updatedEvent.id ? updatedEvent : event  )  );   toast({ /\* ... \*/ });  setIsEditMode(**false**);  // Không cần đóng dialog ở đây nếu muốn xem lại thông tin vừa sửa  } **catch** (error: any) {  console.error("Error updating event:", error);  toast({ variant: "destructive", /\* ... \*/ });  } **finally** {  setIsLoading(**false**);  } }; |
| --- |

1. **Hiển thị Trạng thái Loading và Lỗi:**
   * Sử dụng state isLoading để hiển thị một spinner hoặc thông báo tải khi đang gọi API.
   * Sử dụng state error để hiển thị thông báo lỗi nếu API gặp sự cố.

| **if** (isLoading) {  **return** <**div**>Đang tải dữ liệu...</**div**>; // Hoặc một component Spinner đẹp hơn  }   **if** (error) {  **return** <**div** className="text-red-600">Lỗi: {error}</**div**>;  }   // Nếu không loading và không có lỗi, render nội dung chính  **return** (  <**div** className="space-y-4">  {/\* ... Phần còn lại của component ... \*/}  </**div**>  ); |
| --- |

### **II. Yêu Cầu Dành Cho Backend Developer**

Để component CalendarSchedule hoạt động với dữ liệu thực tế, backend cần cung cấp các API endpoints sau:

1. **Endpoint Lấy Danh Sách Giáo Viên:**
   * **Method:** GET
   * **URL:** /api/teachers (Ví dụ)
   * **Response Body (JSON):** Mảng các đối tượng giáo viên. Mỗi đối tượng cần có cấu trúc tương tự như trong mockData.teachers:

| [  {  "id": 1,  "name": "Nguyễn Văn A",  "subject": "Toán",  "avatar": "/path/to/avatar1.png", // URL hoặc đường dẫn tới ảnh  "color": "bg-blue-100 border-blue-300 text-blue-800" // Chuỗi class Tailwind (frontend sẽ dùng)  },  // ... more teachers ] |
| --- |

1. **Endpoint Lấy Danh Sách Lịch Học (Sự Kiện):**
   * **Method:** GET
   * **URL:** /api/schedule (Ví dụ)
   * **Query Parameters (Tùy chọn):** Có thể cần thêm tham số để lọc theo tuần, tháng hoặc ngày cụ thể (ví dụ: ?week=15&year=2024). Frontend sẽ cần truyền các tham số này dựa trên currentDate.
   * **Response Body (JSON):** Mảng các đối tượng sự kiện. Mỗi đối tượng cần có cấu trúc tương tự mockData.scheduleEvents:

| [  {  "id": 101,  "title": "Toán Cao Cấp",  "teacherId": 1, // ID liên kết với giáo viên  "room": "A101",  "day": 1, // 0=CN, 1=T2, ..., 6=T7  "startTime": "07:00", // Định dạng HH:MM  "endTime": "08:30" // Định dạng HH:MM  },  // ... more events for the specified period ] |
| --- |

1. **Endpoint Cập Nhật Lịch Học:**
   * **Method:** PUT (hoặc PATCH)
   * **URL:** /api/schedule/{eventId} (Ví dụ, với {eventId} là ID của sự kiện cần cập nhật)
   * **Request Body (JSON):** Đối tượng sự kiện chứa các thông tin đã được chỉnh sửa từ form. Cấu trúc tương tự như một phần tử trong mảng trả về của API lấy danh sách lịch học.

| {  "title": "Toán Cao Cấp (Đã sửa)",  "teacherId": 2,  "room": "B202",  "day": 1,  "startTime": "07:15",  "endTime": "08:45"  // ID thường không cần gửi trong body khi dùng PUT/PATCH với ID trong URL } |
| --- |

* + **Response Body (JSON):** Đối tượng sự kiện đã được cập nhật thành công (bao gồm cả ID).
  + **Status Codes:** 200 OK (thành công), 404 Not Found (không tìm thấy event), 400 Bad Request (dữ liệu không hợp lệ), 500 Internal Server Error.

1. **Endpoint Xóa Lịch Học:**
   * **Method:** DELETE
   * **URL:** /api/schedule/{eventId} (Ví dụ, với {eventId} là ID của sự kiện cần xóa)
   * **Response Body:** Thường không cần nội dung (No Content).
   * **Status Codes:** 204 No Content (thành công), 404 Not Found (không tìm thấy event), 500 Internal Server Error.
2. **Endpoint Thêm Lịch Học (Nếu cần):**
   * **Method:** POST
   * **URL:** /api/schedule (Ví dụ)
   * **Request Body (JSON):** Đối tượng sự kiện mới cần tạo (không cần id).
   * **Response Body (JSON):** Đối tượng sự kiện vừa được tạo (bao gồm id do backend sinh ra).
   * **Status Codes:** 201 Created (thành công), 400 Bad Request, 500 Internal Server Error.

**Lưu ý cho Backend:**

* Đảm bảo các kiểu dữ liệu trả về khớp với yêu cầu của frontend (đặc biệt là id, day, startTime, endTime).
* Xử lý lỗi và trả về các mã trạng thái HTTP phù hợp.
* Xem xét việc phân trang hoặc lọc dữ liệu nếu số lượng sự kiện/giáo viên lớn.
* Cân nhắc về vấn đề xác thực (authentication) và ủy quyền (authorization) để đảm bảo chỉ người dùng phù hợp mới có thể xem/sửa/xóa lịch học.

Bằng cách thực hiện các bước trên ở frontend và backend cung cấp các API cần thiết, bạn có thể tích hợp thành công component CalendarSchedule với dữ liệu thực tế.