



# KẾT NỐI FRONTEND VÀ BACKEND

Trình bày: Bùi Bá Nguyên

Ryomo Vietnam Solutions Co. Ltd

# Giới thiệu

- **Mục tiêu:** Kết nối Next.js với API Spring Boot qua Axios, xử lý CORS, kiểm tra CRUD.
- **Vai trò:**
  - Axios: Gọi API HTTP, xử lý JSON.
  - CORS: Cho phép frontend truy cập backend.
- **Quy trình:**
  - Frontend gửi request → Backend trả JSON → Cập nhật giao diện.

# Hiểu và cấu hình CORS

- **CORS là gì?**

- **CORS** (Cross-Origin Resource Sharing - Chia sẻ tài nguyên giữa các nguồn gốc khác nhau), nói một cách đơn giản CORS là một cơ chế bảo mật cho phép các trình duyệt web kiểm soát việc các ứng dụng web chạy trên một **nguồn gốc (origin)** này có thể truy cập tài nguyên từ một **nguồn gốc khác** hay không.

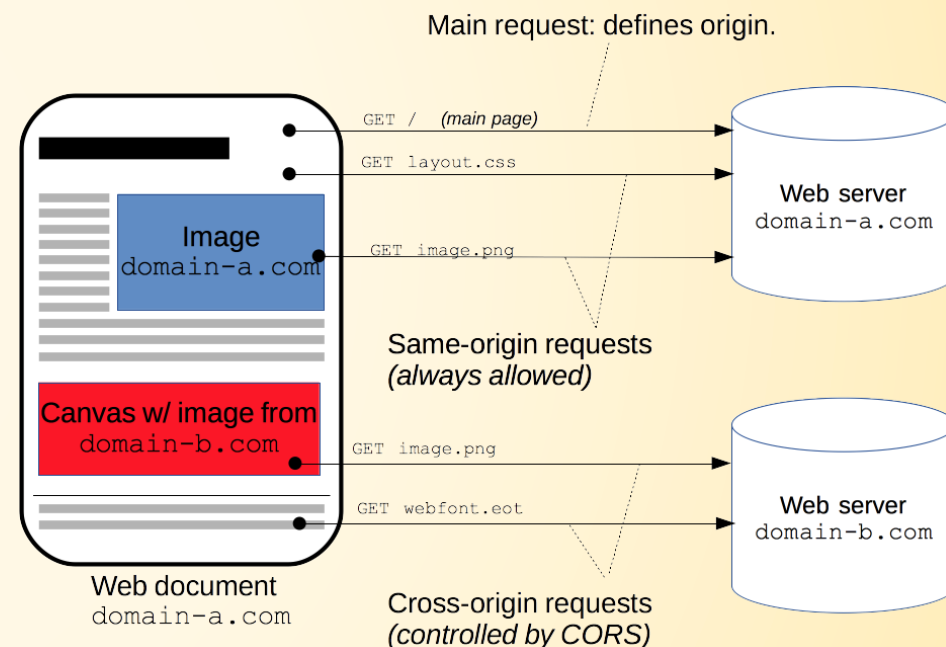
- **Tại sao chúng ta cần CORS?**

- Theo chính sách bảo mật mặc định được gọi là **Same-Origin Policy** (SOP), đây là một trong những chính sách an ninh quan trọng nhất trên các trình duyệt hiện đại được đưa ra bởi Netscape vào năm 1995. Về cơ bản thì đây là một chính sách quy định nội dung của một domain chỉ được đọc và thay đổi bởi một thành phần khác cũng nằm cùng domain đó, các yêu cầu từ các domain khác sẽ bị chặn lại.
- Nếu không có **Same-Origin Policy** thì sẽ thế nào ?
- SOP có hiệu quả trong việc ngăn chặn việc truy cập trái phép từ các domain khác nhau nhưng đồng thời nó cũng ngăn cản những tương tác hợp lệ giữa các server hoặc client đáng tin cậy. Ví dụ như api.example.com và image.example.com, hai domain này chắc chắn là không cùng nguồn gốc.

# Hiểu và cấu hình CORS

## ▪ CORS hoạt động ra sao?

- Khi một trình duyệt gửi yêu cầu đến một tài nguyên ở một nguồn gốc khác, trình duyệt sẽ gửi kèm một tiêu đề HTTP đặc biệt là Origin trong yêu cầu. Tiêu đề này cho biết nguồn gốc của trang web đang thực hiện yêu cầu.
- Server nhận yêu cầu sẽ kiểm tra tiêu đề Origin. Nếu Server cho phép nguồn gốc đó truy cập tài nguyên, nó sẽ phản hồi lại với một header HTTP đặc biệt khác là Access-Control-Allow-Origin. Giá trị của tiêu đề này sẽ là nguồn gốc được phép hoặc \* (cho phép tất cả các nguồn gốc).
- Nếu header Access-Control-Allow-Origin không được trả về, hoặc nếu nguồn gốc của yêu cầu không được liệt kê trong tiêu đề đó, trình duyệt sẽ chặn phản hồi và báo lỗi CORS, ngăn không cho JavaScript của trang web truy cập vào tài nguyên đó.



# Hiểu và cấu hình CORS

- **Các loại yêu cầu CORS:**

- Có hai loại yêu cầu CORS chính:

- Yêu cầu đơn giản (Simple Requests): Đây là những yêu cầu HTTP đáp ứng một số tiêu chí nhất định (ví dụ: chỉ sử dụng các phương thức GET, HEAD, POST và không có các tiêu đề tùy chỉnh đặc biệt). Trong trường hợp này, trình duyệt sẽ gửi yêu cầu trực tiếp và máy chủ sẽ phản hồi với tiêu đề Access-Control-Allow-Origin nếu được phép.
    - Yêu cầu tiền kiểm tra (Preflight Requests): Đối với các yêu cầu phức tạp hơn (ví dụ: sử dụng các phương thức PUT, DELETE, hoặc có các tiêu đề HTTP tùy chỉnh), trình duyệt sẽ gửi một yêu cầu OPTIONS trước khi gửi yêu cầu chính. Yêu cầu OPTIONS này được gọi là yêu cầu "tiền kiểm tra" (preflight request). Máy chủ sẽ phản hồi yêu cầu tiền kiểm tra với các thông tin về các phương thức HTTP và tiêu đề được phép. Nếu yêu cầu tiền kiểm tra được chấp nhận, trình duyệt mới tiếp tục gửi yêu cầu chính.

# Hiểu và cấu hình CORS

- **Cấu hình Spring Boot:**

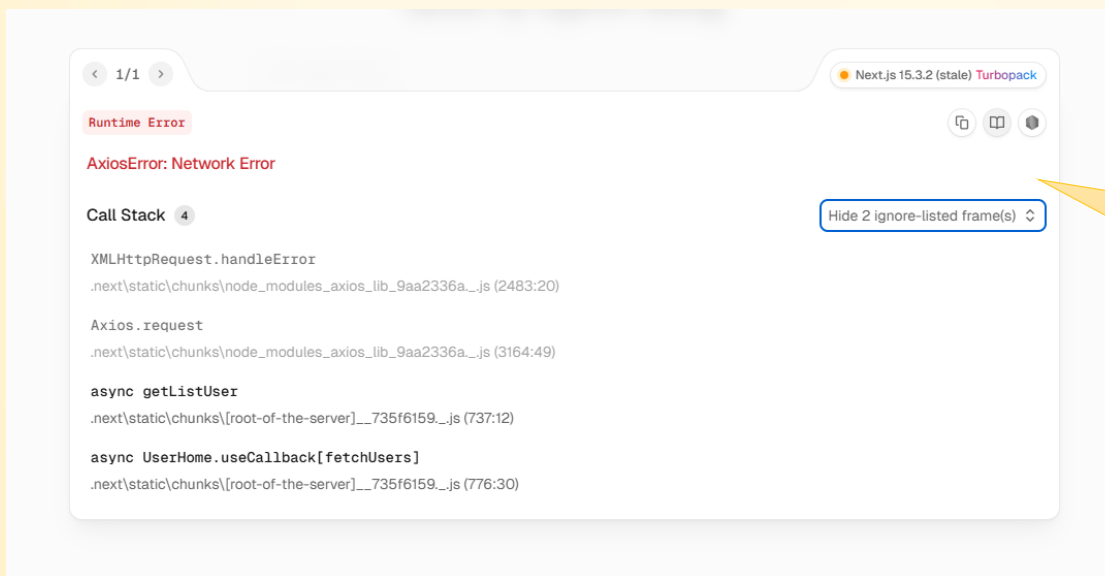
- Tạo CorsConfig.java (aots.rvsc.user.backend.config):

```
package aots.rvsc.user.backend.config;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
@Configuration
public class CorsConfig implements WebMvcConfigurer {
    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**")
            .allowedOrigins("http://localhost:3000")
            .allowedMethods("GET", "POST", "PUT", "DELETE")
            .allowedHeaders("*");
    }
}
```

- **addMapping:** đây là điểm khởi đầu để định nghĩa một cấu hình CORS mới. Nó chỉ định một hoặc nhiều URL patterns mà quy tắc CORS này sẽ áp dụng.
    - **allowedOrigins:** chỉ định các nguồn gốc (origins) được phép gửi yêu cầu cross-origin đến các tài nguyên khớp với pathPattern này.
    - **allowedMethods:** chỉ định các phương thức HTTP (GET, POST, PUT, DELETE, OPTIONS, PATCH, HEAD) được phép cho các yêu cầu cross-origin. Điều này ảnh hưởng đến cả yêu cầu thực tế và yêu cầu tiền kiểm tra (preflight requests).
    - **allowedHeaders:** Chỉ định các tiêu đề yêu cầu (request headers) mà client được phép gửi trong yêu cầu cross-origin. Các tiêu đề này được kiểm tra trong yêu cầu tiền kiểm tra.

# Demo CORS - Simple Requests

- Trước khi cấu hình:
  - Gửi GET `http://localhost:8080/api/users` từ frontend.
  - Kết quả: Lỗi CORS, request bị chặn.

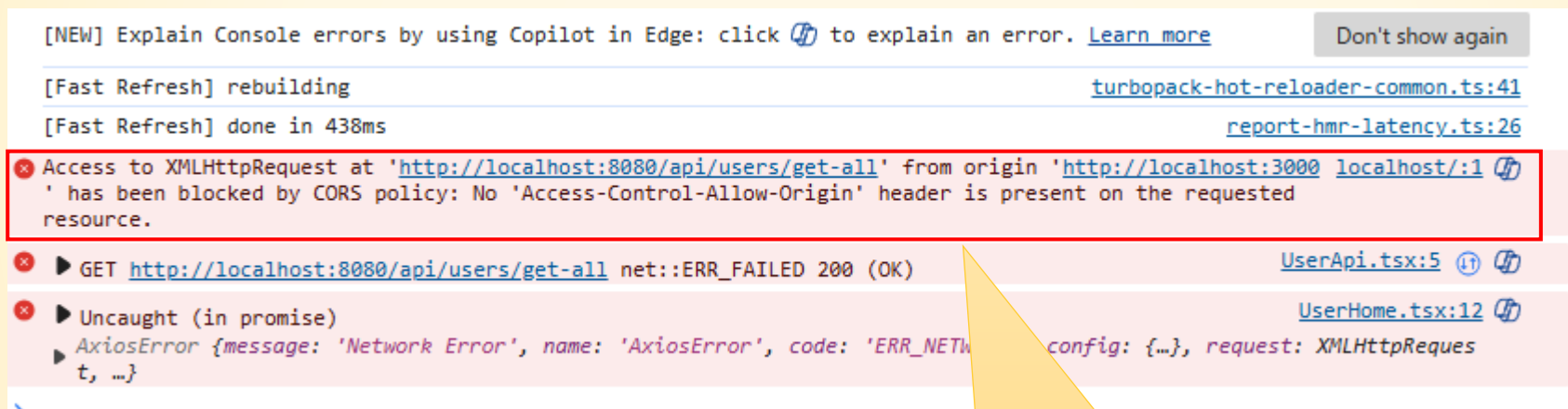


Trình duyệt hiện thông báo lỗi



# Demo CORS - Simple Requests

- Trước khi cấu hình:
  - Mở DevTools > Tab Console đang hiển thị lỗi CORS

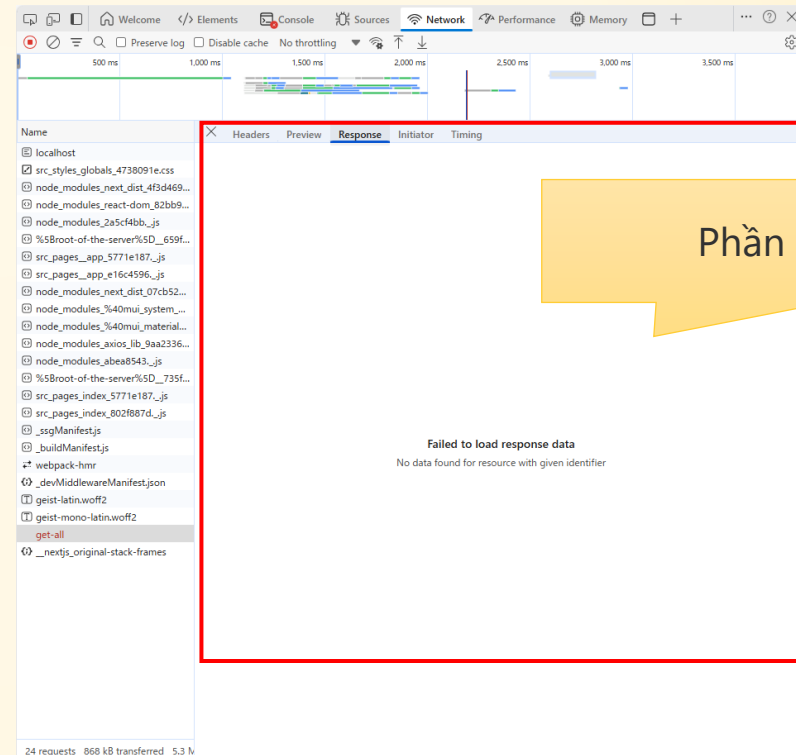
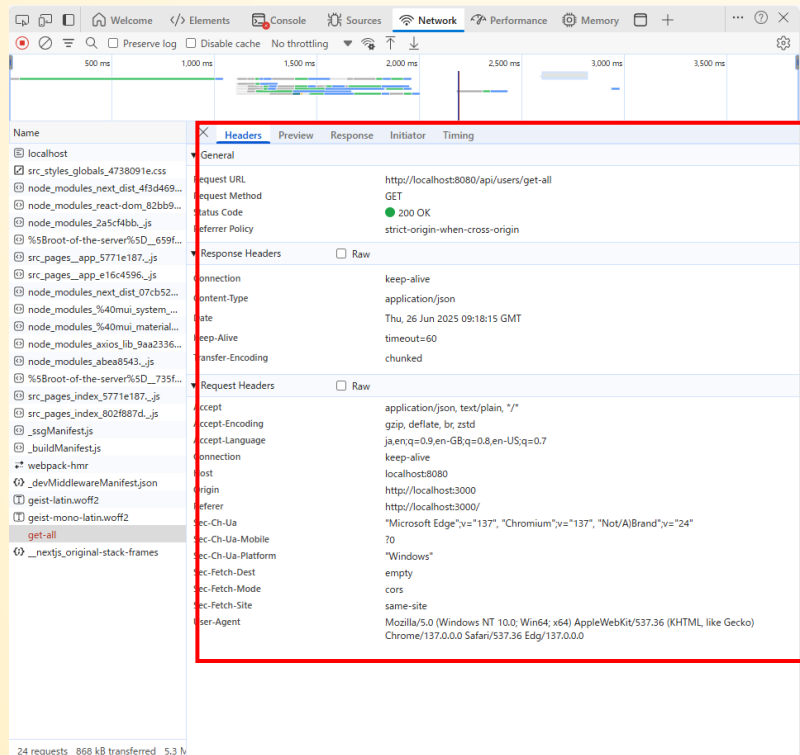


Trình duyệt đã chặn yêu cầu do chính sách CORS.



# Demo CORS - Simple Requests

- Trước khi cấu hình:
  - Mở DevTools > Tab Network phần response sẽ không lấy được dữ liệu



Phần Response trống

# Demo CORS - Simple Requests

- Sau khi cấu hình:
  - Thêm CorsConfig.java.
  - Gửi lại request → Nhận JSON thành công.





localhost:3000

### Quản lý người dùng

Tên người dùng

Email

THÊM NGƯỜI DÙNG

Tên	Email	Hoạt động	Hành động
user1	user@local.com	<input checked="" type="checkbox"/>	
user2	user@local.com	<input checked="" type="checkbox"/>	
user3	user3@local.com	<input checked="" type="checkbox"/>	
Nguyen Van A	nguyenvana@local.com	<input type="checkbox"/>	

Màn hình hiển thị dữ liệu thành công

# Demo CORS - Simple Requests

- Sau khi cấu hình:
  - Mở DevTools > Tab Console không hiện thông báo lỗi

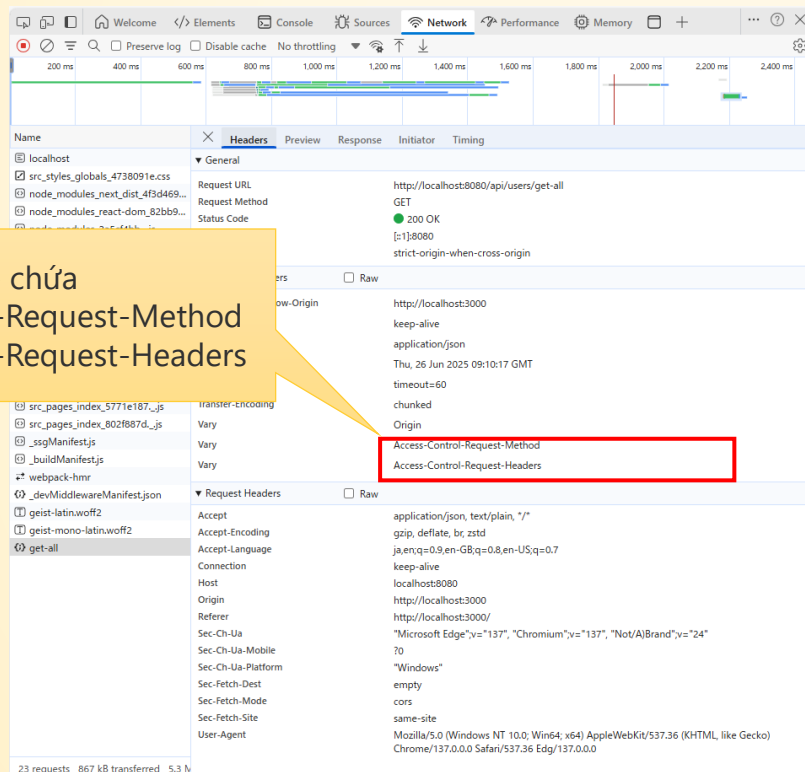
```
Download the React DevTools for a better development experience: https://react.dev/link/react-devtools react-dom-client.development.js:24868  
[HMR] connected websocket.ts:32  
[Fast Refresh] done in NaNms report-hmr-latency.ts:26  
[Fast Refresh] rebuilding turbopack-hot-reloader-common.ts:41  
[Fast Refresh] done in 135ms report-hmr-latency.ts:26  
[Fast Refresh] rebuilding turbopack-hot-reloader-common.ts:41  
[Fast Refresh] done in 127ms report-hmr-latency.ts:26
```

Màn hình console không hiển thị lỗi

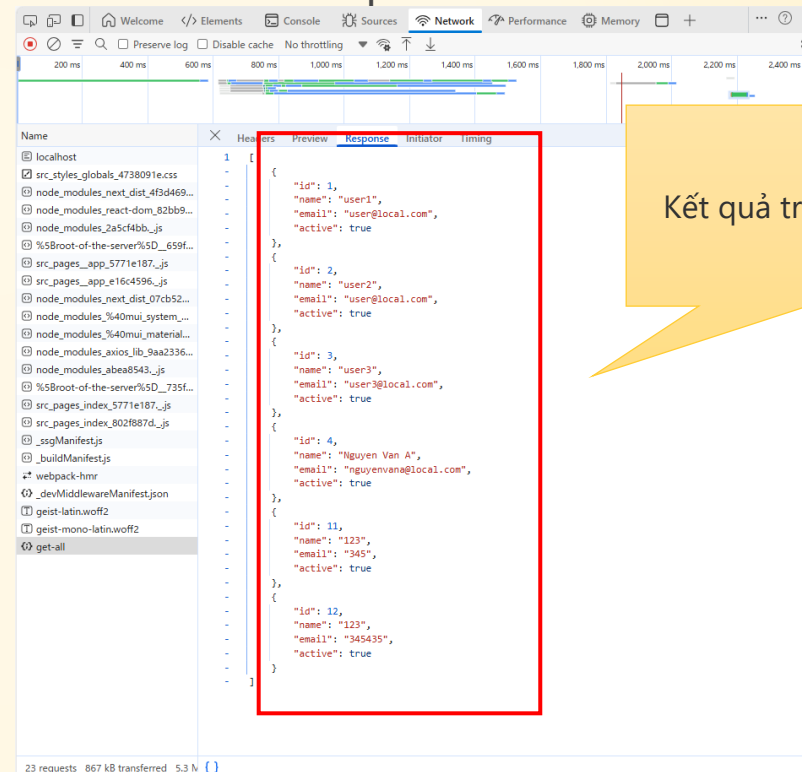
# Demo CORS - Simple Requests

- Sau khi cấu hình:
- Mở DevTools > Tab Network kiểm tra phần Headers và Response

Phần Header sẽ chứa  
Access-Control-Request-Method  
Access-Control-Request-Headers

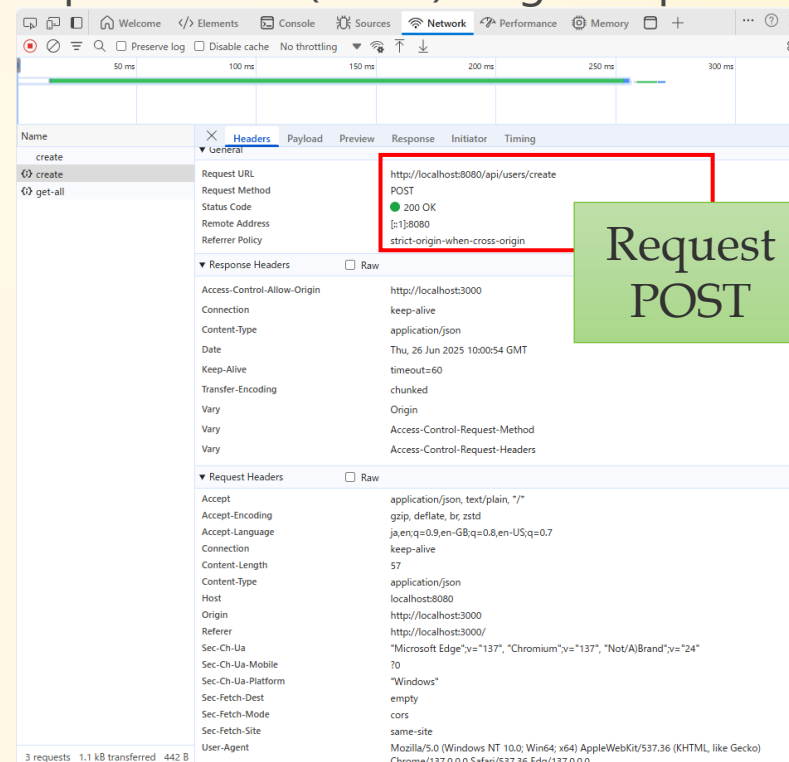
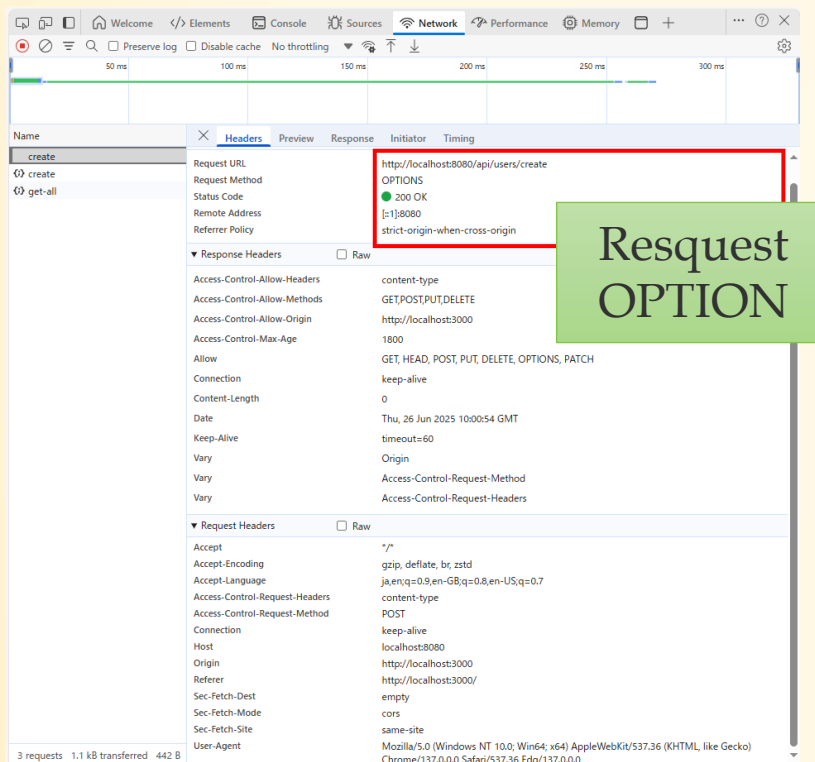


Kết quả trả về dạng JSON



# Demo CORS - Preflight Requests

- Sau khi cấu hình:
  - Mở DevTools > Tab Network trước khi gửi request chính (POST) sẽ gửi request OPTION



# Thư viện Axios

- **Axios là gì?**

- Là một **thư viện JavaScript phổ biến** được sử dụng để **thực hiện các yêu cầu HTTP**. Nó hoạt động cả ở phía **trình duyệt (web browsers)** và **Node.js (server)**.
- **CRUD** viết tắt của **Create, Read, Update, Delete** là một thuật ngữ lập trình nói đến 4 phương thức quen thuộc khi làm việc với kho dữ liệu.
- Axios cung cấp một cách giao tiếp đơn giản với các API web (backend servers).

- **Cài đặt Axios**

- Thư viện axios có thể cài đặt thông qua công cụ npm: `npm install axios`

```
C:\workspace_aots\user-frontend>npm install axios
up to date, audited 428 packages in 14s
143 packages are looking for funding
  run `npm fund` for details
1 low severity vulnerability
To address all issues, run:
  npm audit fix
Run `npm audit` for details.
```

# Thư viện Axios

- **Cách sử dụng Axios cơ bản**

- Các phương thức HTTP thường sử dụng nhiều nhất để khai thác dữ liệu là GET và POST. Axios có hai phương thức để thực hiện GET và POST dữ liệu.

- **Promise**

- Là Object để sử dụng cho điều khiển dựa trên kết quả trả về của những xử lý bất đồng bộ. (Sau khi xử lý thành công or thất bại)

- **Phương thức GET**

```
export async function getListUser() {  
  return await axios.get('http://localhost:8080/api/users/get-all');  
}
```

- **Phương thức POST**

```
export async function addUser(user: User) {  
  return await axios.post('http://localhost:8080/api/users/create', user);  
}
```



# Tích hợp API

## ▪ Cập nhật pages/index.tsx:

```
export default function UserHome() {
  const [users, setUsers] = useState<User[]>([]);

  const fetchUsers = useCallback(async () => {
    const response = await userApi.getListUser();
    setUsers(response.data);
  }, []);

  useEffect(() => {
    fetchUsers();
  }, [fetchUsers]);

  const addUser = useCallback(async (name: string, email: string) => {
    const user = { name, email, active: true };
    await userApi.addUser(user);
    fetchUsers();
  }, [fetchUsers]);

  const toggleUser = useCallback(async (id: number | undefined) => {
    if (!id) return;
    const user = users.find((u) => u.id === id);
    if (user) {
      user.active = !user.active;
      await userApi.updateUser(user);
      fetchUsers();
    }
  }, [users, fetchUsers]);

  const deleteUser = useCallback(async (id: number | undefined) => {
    if (!id) return;
    const user = users.find((u) => u.id === id);
    if (user) {
      await userApi.deleteUser(user);
    }
  }, [users, fetchUsers]);
}
```

Axios dùng GET

Axios dùng POST

## ▪ useCallback

- Là một trong những React Hooks giúp bạn tối ưu hóa hiệu suất ứng dụng bằng cách ghi nhớ (memoize) các hàm
- Được sử dụng chủ yếu để ngăn chặn việc re-render không cần thiết của các component con (đặc biệt là các component con được bọc bởi React.memo).

# Tích hợp API - Component

```
return (  
  <div>  
    <h1 style={{ textAlign: 'center', marginTop: '2rem' }}>Quản lý người dùng</h1>  
    <UserForm onAdd={addUser} />  
    <UserList users={users} onDelete={deleteUser} onToggle={toggleUser} />  
  </div>  
);
```

- Theo đoạn source trên ta đã tích hợp các API vào 2 component là UserForm và UserList

- **Parameter Component:**

- Truyền hàm qua props (VD: onAdd trong UserForm).
- Parameter nên khai báo tường minh:

```
interface UserFormProps {  
  onAdd: (name: string, email: string) => void;  
}
```

# Thực hành

- **Nhiệm vụ (45 phút):**

- Chạy backend (Eclipse) và frontend (npm run dev).
- Kiểm tra:
  - Thêm người dùng mới (POST).
  - Xem danh sách (GET).
- Debug lỗi: CORS, API không phản hồi (mở DevTools).

- **Lưu ý:**

- Đảm bảo backend chạy trước (localhost:8080).
- Kiểm tra request/response trong DevTools.

# Tổng kết

- **Tổng kết:**

- Tích hợp full-stack với Axios.
- Xử lý CORS trong Spring Boot.
- Sử dụng useCallback để tối ưu.

- **Bài tập:**

- Thêm tính năng check active (cập nhật DB):
  - Thêm checkbox trong UserList.tsx.
  - Gọi API PUT để cập nhật active.
- Tìm hiểu Axios interceptors.

# Q&A

