FPT Education

FPT POLYTECHNIC

cdio™

Conceive Design Implement Operate

THỰC HỌC – THỰC NGHIỆP

# FRONT-END FRAMEWORK

## LIFECYCLE TRONG VUE - SỬ DỤNG AXIOS CALL API

http://www.poly.edu.vn

# Mục tiêu

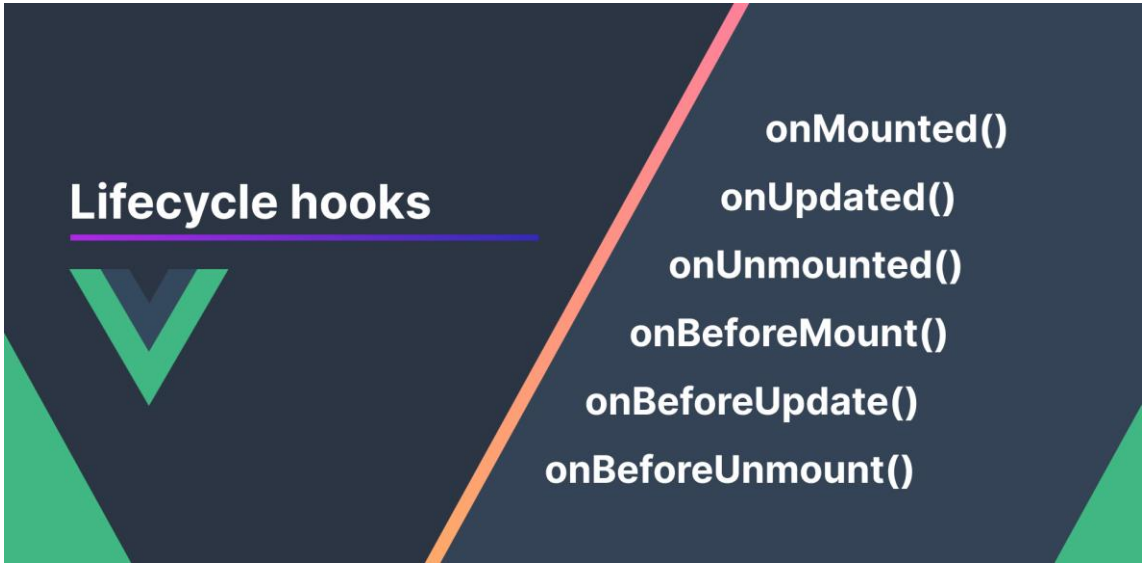
- Hiểu được khái niệm lifecycle
- Hiểu được cách sử dụng các hooks trong lifecycle component
- Nắm được cách cài đặt thư viện axios
- Hiểu được cách sử dụng thư viện axios trong Vue

# NỘI DUNG

- Phần 1: Lifecycle trong Vue
  - Tổng quan về lifecyle hooks
  - Sử dụng hooks cơ bản
  - onMounted()
  - onUpdate()
  - onUnmounted()
  - onBeforeMount()
  - onBeforeUpdate()
  - onBeforeUnmount()
- Phần 2:
  - Tổng quan Axios
  - Cách cài đặt và sử dụng các phương thức trong Axios

# Phần 1
# Lifecycle trong Vue

# LIFECYCLE COMPONENT LÀ GÌ?




- **Lifecycle component (vòng đời component )** trong Vue là chuỗi các giai đoạn mà một component trải qua từ khi được tạo ra cho đến khi bị hủy.
- Trong quá trình này, các hàm **lifecycle hooks (hook vòng đời)** được thực thi, cho phép lập trình viên thêm mã tùy chỉnh tại các giai đoạn quan trọng.

# ĐĂNG KÝ CÁC LIFECYCLE HOOKS

- Trong Vue, các lifecycle hooks **(hook vòng đời)** như onMounted được sử dụng để chạy mã sau khi component đã hoàn thành việc render ban đầu và tạo ra các node DOM.

```
<script setup>
import { onMounted } from'vue'
onMounted(() => {
        console.log(`the component is now mounted.`)
})
</script>
```

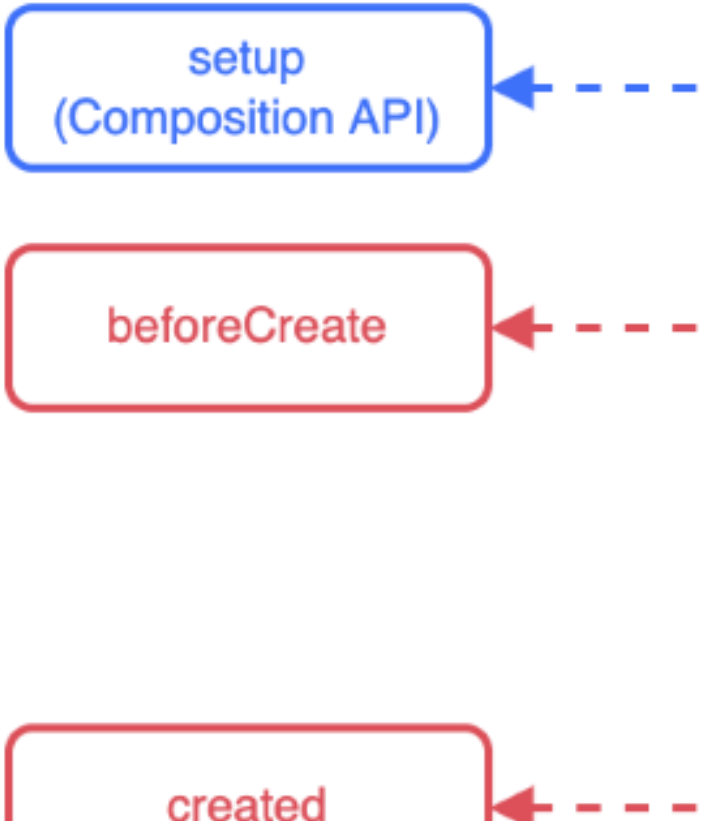
- Một vài hook thường dùng: **onMounted**, **onUpdated**, and **onUnmounted**.
- Các hooks phải được đăng ký đồng bộ trong quá trình thiết lập component, nên không nên sử dụng chúng trong các hàm bất đồng bộ như **setTimeout**.

```
setTimeout(() => {
        onMounted(() => {
        // Cách này sẽ không hoạt động.
         })
}, 100)
```

# Sơ đồ vòng đời

Các giai đoạn chính bao gồm:

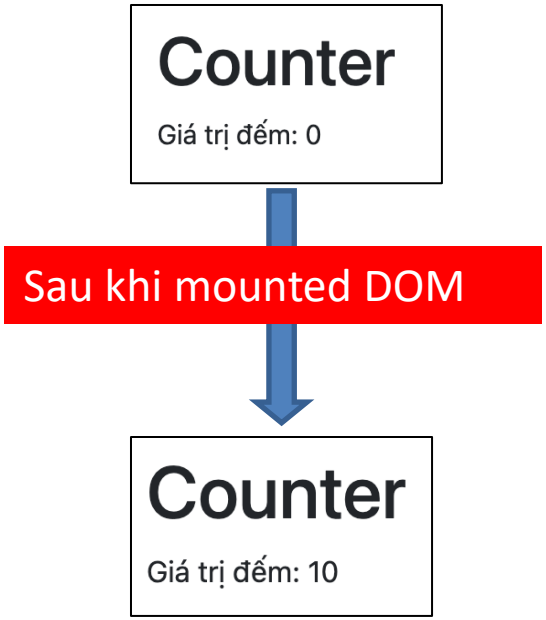
- **Khởi tạo:** Component được tạo và dữ liệu reactive được khởi tạo.
- **Gắn vào DOM (Mounting):** Component được gắn vào DOM.
- **Cập nhật (Updating):** Component được cập nhật khi dữ liệu thay đổi.
- **Hủy bỏ (Unmounting):** Component bị xóa khỏi DOM và bị hủy.



# onMounted()

- **onMounted()**: Được gọi ngay sau khi component được gắn vào DOM. Đây là hook phổ biến để thực hiện các tác vụ như gọi API hoặc thao tác DOM sau khi component đã render xong.
- ví dụ:

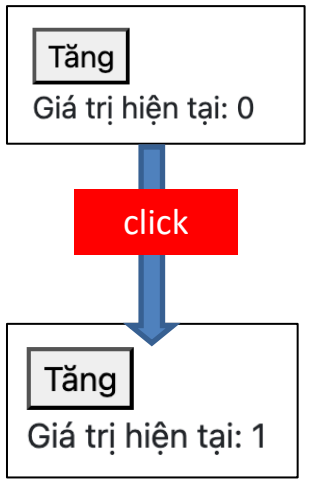
```
<template>
        <h1>Counter</h1>
        <p>Giá trị đếm: {{count}}</p>
</template>
<script setup>
import { ref, onMounted } from"vue";
// Tạo biến reactive cho giá trị đếm
const count = ref(0);
// Sử dụng onMounted để thực hiện hành động khi component được gắn vào DOM
onMounted(() => {
        console.log("Component đã được gắn vào DOM");
        // Tăng giá trị đếm lên 10 khi component được mount
        count.value = 10;
});
</script>
```



# onUpdated()

- **onMounted()**: Được gọi ngay sau khi component đã cập nhật cây DOM của nó do thay đổi state component
- **Lưu ý quan trọng:** Không nên thay đổi trạng thái của component trong onUpdated(), vì có thể dẫn đến vòng lặp cập nhật vô hạn.
- ví dụ:

```
<template>
        <button @click="increment">Tăng</button>
        <p id="count">Giá trị hiện tại: {{count}}</p>
</template>
<script setup>
import { ref, onUpdated } from"vue";
const count = ref(0);
// Hook được gọi sau khi component đã cập nhật
onUpdated(() => {
        console.log("Component đã cập nhật. Giá trị hiện tại:", count.value);
});
const increment = () => {
        count.value++;
};
</script>
```



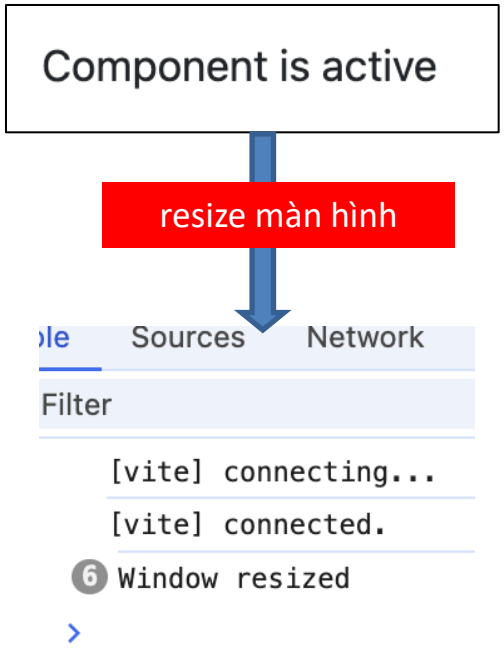

```
Component đã cập nhật. Giá trị hiện tại: 1
```

Trong ví dụ này, **onUpdated**() được sử dụng để truy cập vào nội dung văn bản của phần tử DOM sau khi giá trị state **count** thay đổi.

# onUnmounted()

- **onUnmounted()** là một hook cho phép thực hiện các hành động dọn dẹp khi một component bị gỡ bỏ khỏi DOM.
- Nó giúp đảm bảo rằng tài nguyên được giải phóng đúng cách, tránh rò rỉ bộ nhớ và giữ cho ứng dụng hoạt động hiệu quả.
- ví dụ:

```
<template>
        <div>
        <p>Component is active</p>
        </div>
</template>
<script setup>
import { onUnmounted } from "vue";
const handleResize = () => {
        console.log("Window resized");
};
// Đăng ký sự kiện resize khi component được mount
window.addEventListener("resize", handleResize);
// Sử dụng onUnmounted để gỡ bỏ sự kiện khi component bị hủy
onUnmounted(() => {
        window.removeEventListener("resize", handleResize);
});
</script>
```



# onBeforeMount()

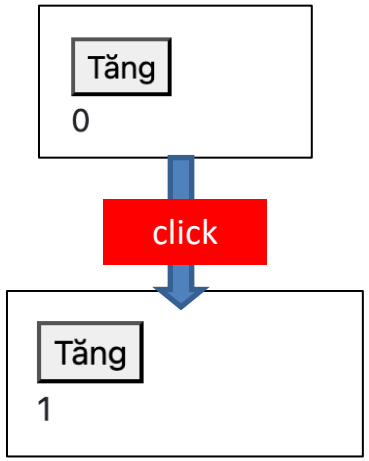
- **onBeforeMount**() là một hook được gọi ngay trước khi component được mount vào DOM.
- Thường sử dụng cho việc thiết lập dữ liệu và thực hiện các hành động cần thiết trước khi component hiển thị, nhưng không thể tương tác với DOM trong hook này.
- Ví dụ:

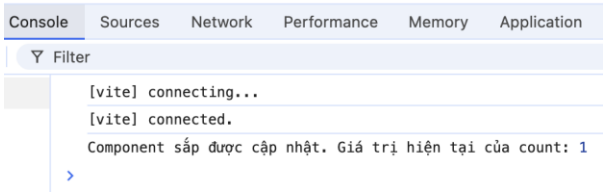
```
<template>
        <div>
                <h1>{{message}}</h1>
        </div>
</template>
<script setup>
import { ref, onBeforeMount } from 'vue';
const message = ref('');
// Thiết lập giá trị cho message trước khi component được mount
onBeforeMount(() => {
        message.value = 'Hello, Vue 3!';
});
</script>
```

# onBeforeUpdate()

- **onBeforeUpdate()** là một hook được gọi ngay trước khi component được cập nhật do thay đổi trạng thái.
- Thường sử dụng để thực hiện các hành động chuẩn bị trước khi cập nhật, nhưng bạn không thể truy cập DOM đã cập nhật trong hook này.
- Ví dụ:

```
<template>
        <button @click="tang">Tăng</button>
        <p>{{count}}</p>
</template>
<script setup>
import { ref, onBeforeUpdate } from "vue";
const count = ref(0);
// Hook được gọi trước khi component cập nhật
onBeforeUpdate(() => {
        console.log("Component sắp được cập nhật. Giá trị hiện tại
của count:", count.value);
});
const tang = () => {
        count.value++;
};
</script>
```

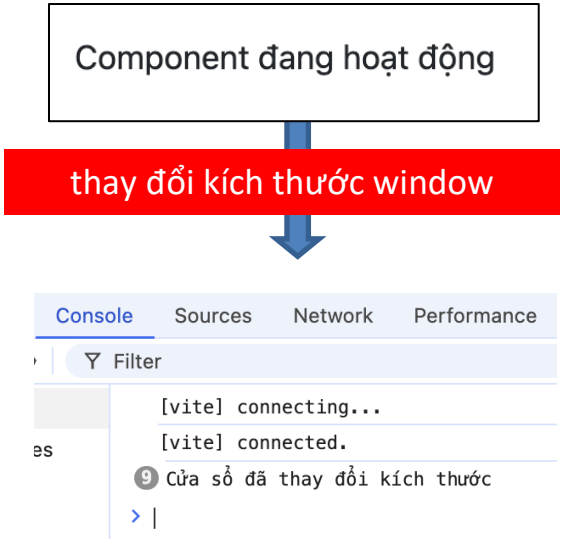


# onBeforeUnmount()

- **onBeforeUnmount()** là một hook được gọi ngay trước khi một component bị hủy bỏ (unmounted) khỏi DOM.
- Ví dụ:

```
<template>
<div>
<p>Component đang hoạt động</p>
</div>
</template>
<script setup>
import { onBeforeUnmount } from "vue";
const handleResize = () => {
console.log("Cửa sổ đã thay đổi kích thước");
};
// Đăng ký sự kiện resize khi component được mount
window.addEventListener("resize", handleResize);
// Sử dụng onBeforeUnmount để gỡ bỏ sự kiện khi component bị hủy
onBeforeUnmount(() => {
window.removeEventListener("resize", handleResize);
console.log("Component sắp bị hủy, đã gỡ bỏ sự kiện resize");
});
</script>
```



# onBeforeUnmount()

- **Ngoài ra còn các Lifecycle Hook khác như:**

| Lifecycle Hook | Mô Tả |
|---|---|
| **onErrorCaptured** | Giúp bắt và xử lý lỗi trong component con, cho phép bạn quyết định cách ứng xử khi lỗi xảy ra. |
| **onRenderTracked** | Cung cấp thông tin về các phụ thuộc phản ứng mà component đang theo dõi, hữu ích cho việc tối ưu hóa. |
| **onRenderTriggered** | Giúp theo dõi khi nào component được render lại, có thể được sử dụng để ghi lại hoặc phân tích hiệu suất. |
| **onActivated** | Cho phép bạn thực hiện các hành động khi component được kích hoạt lại, rất hữu ích khi làm việc với keep-alive. |
| **onDeactivated** | Cung cấp cơ hội để thực hiện dọn dẹp hoặc lưu trữ trạng thái khi component không còn hiển thị. |
| **onServerPrefetch** | Hữu ích trong môi trường server-side rendering, cho phép bạn lấy dữ liệu trước khi gửi HTML đến client. |


Ví dụ sử dụng kiến thức về lifecycle hooks cho phép người dùng lấy dữ liệu từ API, hiển thị nó, và tương tác với giao diện. Các thông tin về lifecycle hooks sẽ được ghi lại trong console để kiểm tra hoạt động của chúng.

## ❑ Bước 1: Xây dựng giao diện

```
<template>
<div>
        <h1>Demo Lifecycle Hooks with API</h1>
        <button @click="fetchData">Lấy dữ liệu</button>
        <pv-if="loading">Đang tải dữ liệu...</p>
        <pv-if="error">{{error}}</p>
        <ul>
                <li v-for="item in items" :key="item.id">{{item.title}}</li>
        </ul>
        <button @click="clearItems">Xóa dữ liệu</button>
</div>
</template>
```

## ❑ Bước 2: Code javascript

```
<script setup>
import { ref, onMounted, onUpdated, onUnmounted, onBeforeMount,
onBeforeUpdate, onBeforeUnmount} from"vue";
const items = ref([]);
const loading = ref(false);
const error = ref(null);
// Hook gọi trước khi component được mount
onBeforeMount(() => {
console.log("Component sắp được mount.");
});
// Hook gọi khi component được mount
onMounted(async () => {
console.log("Component đã được mount.");
            // Lấy dữ liệu khi component được mount
            await fetchData(); });
// Hook gọi trước khi component cập nhật
onBeforeUpdate(() => {
console.log("Component sắp được cập nhật.");
});
// Hook gọi khi component được cập nhật
onUpdated(() => {
console.log("Component đã được cập nhật.");
});
// Hook gọi trước khi component bị hủy
onBeforeUnmount(() => {
console.log("Component sắp bị hủy.");
});
// Hook gọi khi component bị hủy
onUnmounted(() => {
console.log("Component đã bị hủy.");
```

```
// Hàm lấy dữ liệu từ API
const fetchData = async () => {
loading.value = true;
error.value = null;
try {
const response = await
fetch("https://jsonplaceholder.typicode.com/todos");
const data = awaitresponse.json();
items.value = data.slice(0, 10); // Chỉ lấy 10 item đầu
 } catch (err) {
error.value = "Có lỗi xảy ra khi lấy dữ liệu!";
 } finally {
loading.value = false;
 }
};
// Hàm xóa dữ liệu
const clearItems = () => {
            items.value = [];
};
</script>
```

# Phần 2: Sử dụng Axios Call API

# GIỚI THIỆU VỀ AXIOS


- Axios là một thư viện HTTP client cho JavaScript.
- Dễ dàng sử dụng với các Promise.
- Hỗ trợ cho việc gọi API từ frontend.
- Cung cấp các tính năng như interceptors, tự động chuyển đổi JSON, và nhiều hơn nữa.

# CÀI ĐẶT AXIOS

- Cài đặt Axios qua npm:

```
npm i axios
```

```
},
"dependencies": {
  "axios": "^1.7.7",
  "bootstrap": "^5.3.3",
  "vue": "^3.4.31"
},
```

package.json

- Hoặc bạn có thể sử dụng CDN trong HTML:

```
<script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>
```

- Import Axios trong component Vue:

```
<script setup>
import axios from'axios';
</script>
```

# CẤU TRÚC MỘT YÊU CẦU API CƠ BẢN

- Cách thực hiện yêu cầu GET đơn giản:

```
<script setup>
import axios from"axios";
axios
        .get("https://api.example.com/data")
        .then((response) => {
        console.log(response.data); // Xử lý dữ liệu nhận được
 })
.catch((error) => {
        console.error(error); // Xử lý lỗi
 });
</script>
```

**Trong đó**

- **.get()** là phương thức dùng để gọi API.
- **.then()** nhận phản hồi thành công.
- **.catch()** để xử lý các lỗi phát sinh trong quá trình gọi API.

# SỬ DỤNG AXIOS TRONG VUE

## ❑ Ví dụ thực hiện phương thức GET:

```
<template>
<h1>Dữ liệu từ API</h1>
<ul>
<li v-for="todo in todos" :key="todo.id">{{todo.title}}</li>
</ul>
</template>
<script setup>
import { ref, onMounted } from "vue";
import axios from "axios";
const todos = ref([]);
// Gọi API khi component được mount
onMounted(async () => {
try {
const response = await axios.get("https://jsonplaceholder.typicode.com/todos");
todos.value = response.data; // Cập nhật dữ liệu vào state
 } catch (error) {
            console.error("Lỗi khi gọi API:", error);
 }
});
</script>
```

**Dữ liệu từ API**

- delectus aut autem
- quis ut nam facilis et officia qui
- fugiat veniam minus
- et porro tempora
- laboriosam mollitia et enim quasi adipisci quia provident illum
- qui ullam ratione quibusdam voluptatem quia omnis
- illo expedita consequatur quia in
- quo adipisci enim quam ut ab
- molestiae perspiciatis ipsa
- illo est ratione doloremque quia maiores aut
- vero rerum temporibus dolor
- ipsa repellendus fugit nisi
- et doloremque nulla
- repellendus sunt dolores architecto voluptatum
- ab voluptatum amet voluptas
- accusamus eos facilis sint et aut voluptatem
- quo laboriosam deleniti aut qui
- dolorum est consequatur ea mollitia in culpa
- molestiae ipsa aut voluptatibus pariatur dolor nihil
- ullam nobis libero sapiente ad optio sint
- suscipit repellat esse quibusdam voluptatem incidunt
- distinctio vitae autem nihil ut molestias quo
- et itaque necessitatibus maxime molestiae qui quas velit
- adipisci non ad dicta qui amet quaerat doloribus ea
- voluptas quo tenetur perspiciatis explicabo natus
- aliquam aut quasi
- veritatis pariatur delectus
- nesciunt totam sit blanditiis sit
- laborum aut in quam
- nemo perspiciatis repellat ut dolor libero commodi blanditiis omnis

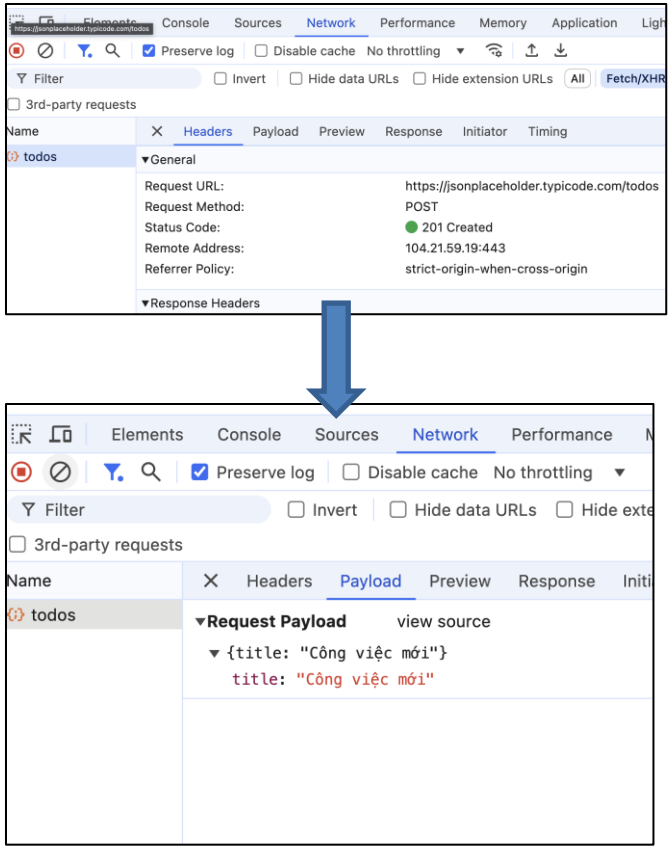
**Trong đó:**

- Sử dụng onMounted để gọi API ngay khi component được tải.
- Dữ liệu từ API được lưu vào biến items, và hiển thị trong danh sách.

# Gửi Dữ Liệu với POST

- Ví dụ gửi dữ liệu tới API với phương thức POST:

```
<template>
<h1>Dữ liệu từ API</h1>
<ul>
<li v-for="todo in todos" :key="todo.id">{{todo.title}}</li>
</ul>
</template>
<script setup>
import { ref, onMounted } from "vue";
import axios from "axios";
const todos = ref([]);
// Gọi API khi component được mount
onMounted(async () => {
try {
// Gửi dữ liệu mới bằng phương thức POST
const postResponse = await axios.post("https://jsonplaceholder.typicode.com/todos", {
title:"Công việc mới",
 });
console.log("Dữ liệu đã được gửi:", postResponse.data);
 } catch (error) {
console.error("Lỗi khi gọi API:", error);
 }
});
</script>
```



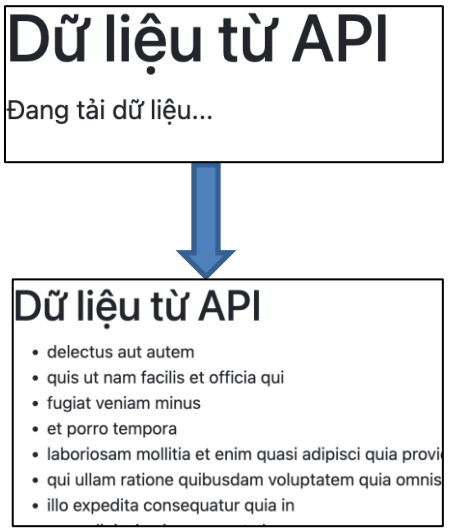

**Trong đó:**

- Thực hiện một yêu cầu POST để gửi dữ liệu đến server.
- Dữ liệu được truyền dưới dạng một đối tượng JSON.

# QUẢN LÝ TRẠNG THÁI TẢI

- Để quản lý trạng thái tải dữ liệu, bạn có thể sử dụng một biến trạng thái như sau:

```
<template>
<h1>Dữ liệu từ API</h1>
<p v-if="loading">Đang tải dữ liệu...</p>
<ul>
<li v-for="todo in todos" :key="todo.id">{{todo.title}}</li>
</ul>
</template>
<script setup>
import { ref, onMounted } from "vue";
import axios from "axios";
const todos = ref([]);
const loading = ref(false); // Biến trạng thái
onMounted(async () => {
        loading.value = true; // Bắt đầu tải dữ liệu
try {
const response = await axios.get("https://jsonplaceholder.typicode.com/todos");
todos.value = response.data; // Cập nhật dữ liệu
 } catch (error) {
console.error("Lỗi:", error);
 } finally {
loading.value = false; // Kết thúc tải dữ liệu
 }
});
</script>
```




**Trong đó:**

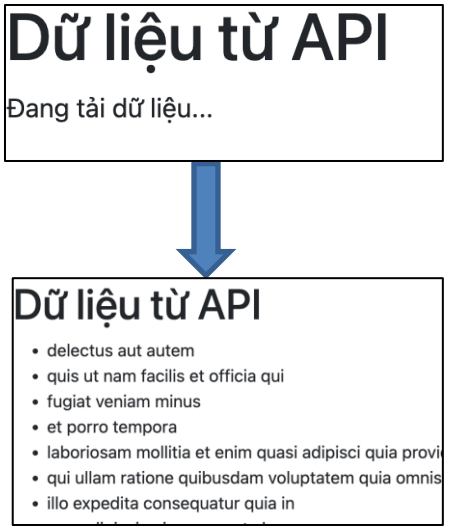
- Biến loading được sử dụng để hiển thị thông báo khi dữ liệu đang được tải.

FPT Education
FPT POLYTECHNIC

# CẬP NHẬT DỮ LIỆU VỚI PUT

- Để cập nhật dữ liệu, bạn cần sử dụng phương thức PUT hoặc PATCH
- Địa chỉ URL cần phải chỉ định rõ ràng tài nguyên cần cập nhật. ví dụ như ID

```
<template>
        <h1>Dữ liệu từ API</h1>
        <ul>
        <li v-for="todo in          todos" :key="todo.id">{{todo.title}}</li>
        </ul>
</template>
<script setup>
import { ref, onMounted } from "vue";
import axios from "axios";
const todos = ref([]);
// Gọi API khi component được mount
onMounted(async () => {
try {
// Gửi dữ liệu mới bằng phương thức PUT
const putResponse = await axios.put("https://jsonplaceholder.typicode.com/todos/1", {
title:"Công việc đã cập nhật",
completed:true,
 });
console.log("Dữ liệu đã được cập nhật:", putResponse.data);
 } catch (error) {
console.error("Lỗi khi gọi API:", error);
 }
});
</script>
```



- Cần đảm bảo xử lý lỗi một cách rõ ràng:

```
<template>
<h1>Dữ liệu từ API</h1>
<ul>
<li v-for="todo in todos" :key="todo.id">{{todo.title}}</li>
</ul>
</template>
<script setup>
import { ref, onMounted } from "vue";
import axios from "axios";
const todos = ref([]);
onMounted(async () => {
try {
const response = await axios.get("https://jsonplaceholder.typicode.com/todos");
todos.value = response.data; // Cập nhật dữ liệu vào biến todos
console.log("Dữ liệu:", response.data);
 } catch (error) {
if (error.response) {
// Phản hồi từ server
console.error("Lỗi từ server:", error.response.status);
 } else if (error.request) {
// Yêu cầu đã được gửi nhưng không có phản hồi
console.error("Không có phản hồi từ server:", error.request);
 } else {
// Lỗi khác
console.error("Lỗi:", error.message);
 }
 }
});
</script>
```

# SỬ DỤNG INTERCEPTORS

- **Interceptors** cho phép bạn can thiệp vào yêu cầu hoặc phản hồi trước khi chúng được xử lý. Giúp dễ dàng quản lý và theo dõi các yêu cầu API, thực hiện các thao tác như thêm header hoặc xử lý token.

```
import { ref } from "vue";
import axios from "axios";
const items = ref([]);
const loading = ref(false);
// Thiết lập interceptor cho yêu cầu
axios.interceptors.request.use((config) => {
        // Thêm header vào yêu cầu
        config.headers["X-Custom-Header"] = "HelloWorld"; // Thêm một header tùy chỉnh
        console.log("Yêu cầu đang được gửi với thông tin:", config);
        return config;
 },(error) => {
        console.error("Lỗi yêu cầu:", error);
        return Promise.reject(error);
 }
);
const fetchData = async () => {
        loading.value = true; // Bắt đầu tải dữ liệu
try {
        const response = await axios.get("https://jsonplaceholder.typicode.com/posts");
        items.value = response.data; // Cập nhật dữ liệu vào biến items
} catch (error) {
        console.error("Lỗi khi gọi API:", error);
 } finally {
        loading.value = false; // Kết thúc tải dữ liệu
 }
};
```


# THỰC HIỆN LẠI DEMO CÁC NỘI DUNG TRÊN

# SUMARRY

- ☑ Phần 1: Lifecycle trong Vue
  - ❖ Tổng quan về lifecyle hooks
  - ❖ Sử dụng hooks cơ bản
  - ❖ onMounted()
  - ❖ onUpdate()
  - ❖ onUnmounted()
  - ❖ onBeforeMount()
  - ❖ onBeforeUpdate()
  - ❖ onBeforeUnmount()
- ☑ Phần 2:
  - ❖ Tổng quan Axios
  - ❖ Cách cài đặt và sử dụng các phương thức trong Axios


thank
you!