




# LẬP TRÌNH ECMASCRIPT

---

XỬ LÝ BẤT ĐỒNG BỘ

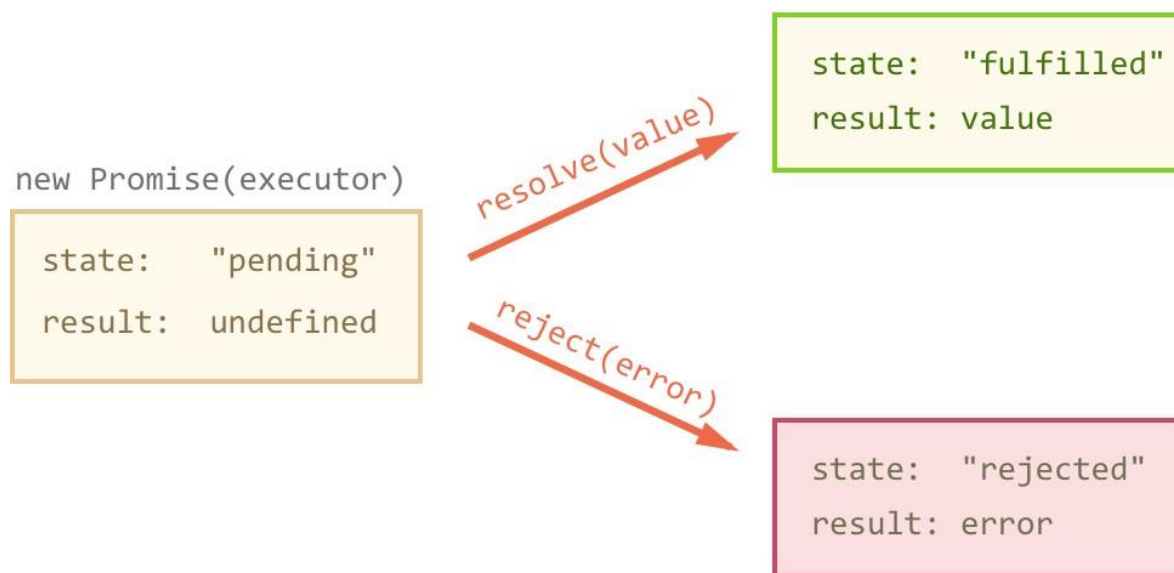
- Ôn tập lại các kiến thức cơ bản bất đồng bộ và promise
- ◎ Giới thiệu Async & Await
- ◎ Sử dụng Async với Axios

-  Ôn tập Promise
-  Tổng quan Async & Await
-  Hướng dẫn sử dụng Async với Axios

# PHẦN 1: ÔN TẬP LẠI PROMISE

# PROMISE LÀ GÌ?

- ❖ Promise là một cơ chế trong JavaScript giúp bạn thực thi các tác vụ bất đồng bộ mà không rơi vào **callback hell** ( là tình trạng các hàm callback lồng nhau ở quá nhiều tầng )
- ❖ Các tác vụ bất đồng bộ có thể là gửi AJAX request, gọi hàm bên trong setTimeout, setInterval hoặc requestAnimationFrame... Đây là một callback hell điển hình.



## ❖ Ví dụ

```
api.getListUser('fpoly', function (err, user) {  
  if (err) throw err  
  api.getPostsOfUser(user, function (err, posts) {  
    if (err) throw err  
    api.getCommentsOfPosts(posts, function (err, comments) {  
      // vân vân và mây mây...  
    })  
  })  
})
```

## ❖ Ví dụ trên khi được viết lại bằng Promise sẽ là:

```
api.getListUser('fpoly', function (err, user) {  
  if (err) throw err  
  api.getPostsOfUser(user, function (err, posts) {  
    if (err) throw err  
    api.getCommentsOfPosts(posts, function (err, comments) {  
      // vân vân và mây mây...  
    })  
  })  
})
```

- ❖ Để tạo ra một promise object thì bạn dùng class Promise có sẵn trong trình duyệt như sau:

```
const doSomething = new Promise(  
  /* executor */ function(resolve, reject) {  
    // Thực thi các tác vụ bất đồng bộ ở đây  
    // và gọi `resolve()` khi tác vụ hoàn thành.  
    // Nếu xảy ra lỗi, gọi đến `reject(error)`.  
  },  
)
```

- ❖ Trong đó, **executor** là một hàm có hai tham số:
  - **resolve** là hàm sẽ được gọi khi promise hoàn thành
  - **reject** là hàm sẽ được gọi khi có lỗi xảy ra

## ❖ Ví dụ với một hàm lấy danh sách sản phẩm từ API

```
api.getProduct = function(id) {  
  // Hàm api.getProduct() trả về một promise object  
  return new Promise((resolve, reject) => {  
    // Gửi AJAX request  
    http.get(`/product/${id}`, (err, result) => {  
      // Nếu có lỗi bên trong callback, chúng ta gọi đến hàm `reject()`  
      if (err) return reject(err)  
  
      // Ngược lại, dùng `resolve()` để trả dữ liệu về cho `.then()`  
      resolve(result)  
    })  
  })  
}
```



- ❖ Hàm `api.getProduct()` sẽ trả về một promise object. Chúng ta có thể truy xuất đến kết quả trả về bằng phương thức **.then()** như sau:

```
function onSuccess(id) {  
  // Trả về giá trị là ID của Product  
  console.log('ID product: ', id);  
}  
function onError(err) {  
  console.error('lỗi: ', error);  
}  
api.getProduct(1).then(onSuccess, onError) // ví dụ 1 là ID của sản phẩm
```

- ❖ Phương thức **.then()** nhận vào 2 hàm là `onSuccess`, `onError`
  - **onSuccess** được gọi khi promise hoàn thành
  - **onError** được gọi khi có lỗi xảy ra.
- ❖ Bên trong tham số **onSuccess** bạn có thể trả về một giá trị đồng bộ, chẳng hạn như giá trị số, chuỗi, null, undefined, array hay object; hoặc một **promise object** khác.

- ❖ Các giá trị bất đồng bộ sẽ được bọc bên trong một Promise, cho phép bạn kết nối (chaining) nhiều promises lại với nhau.

```
getListProduct().then(function(result) {  
    return doSomethingElse(result);  
})  
.then(function(newResult) {  
    return doThirdThing(newResult);  
})  
.then(function(finalResult) {  
    console.log('Kết quả cuối cùng: ' + finalResult);  
})  
.catch(failureCallback);
```

- ❖ Phương thức **.catch()**. Phương thức này chỉ là *cú pháp bọc đường* (syntactic sugar) của `.then(null, onError)`

**▶ DEMO**

- ❑ Truy cập [liên kết](#), hàm showCircle() hiện tại đang sử dụng callback để trả về kết quả, yêu cầu sử dụng cú pháp promise để thay thế
- ❑ Ví dụ:

```
showCircle(150, 150, 100, div => {  
  div.classList.add('message-ball');  
  div.append("Hello, world!");  
});
```

- ❑ Truy cập [liên kết](#) để xem đầy đủ đáp án.
- ❑ Cách mới:

```
showCircle(150, 150, 100).then(div => {  
  div.classList.add('message-ball');  
  div.append("Hello, world!");  
});
```

## PHẦN 2: TỔNG QUAN ASYNC/AWAIT

## ASYNC/AWAIT LÀ GÌ?

- ❑ **Async / Await** là một tính năng JavaScript được dự đoán từ lâu, giúp làm việc với các chức năng không đồng bộ thú vị hơn và dễ hiểu hơn nhiều.
- ❑ Nó được xây dựng dựa trên Promise và tương thích với tất cả các API dựa trên Promise hiện có.
- ❑ Cú pháp :

```
async function name([param[, param[, ...param]]) { statements }
```

- **Name** Tên của function.
- **Param** Tên của một đối số được truyền vào function.
- **Statements** Các câu lệnh bao hàm phần thân của function.

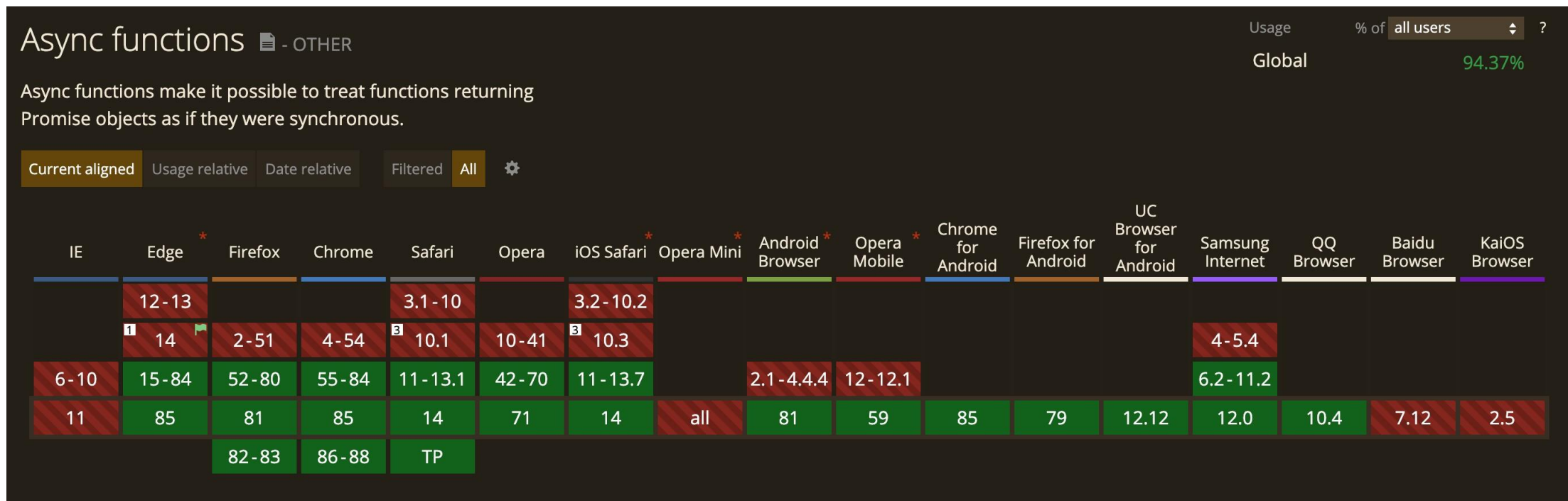
## ASYNC/AWAIT LÀ GÌ?

---

- ❑ Nếu Promises là tương tự như các callback có cấu trúc, async/await là tương tự với kết hợp các [generators](#) và promises.
- ❑ Một hàm async có thể baoMục đích của async/await là để đơn giản hóa việc sử dụng các promises một cách đồng bộ, và để triển khai một số hoạt động trên một nhóm của các Promises.
- ❑ gồm một biểu thức [await](#)
- ❑ Từ khóa **Await** - tạm dừng thực thi các chức năng không đồng bộ và chỉ có hiệu lực bên trong hàm async



- ❑ Hầu hết các browser hiện tại đã support async/await nhưng vẫn còn IE và Opera Mini là không support. Bạn có thể tham khảo hình dưới:



## ❑ Ví dụ sử dụng async/await

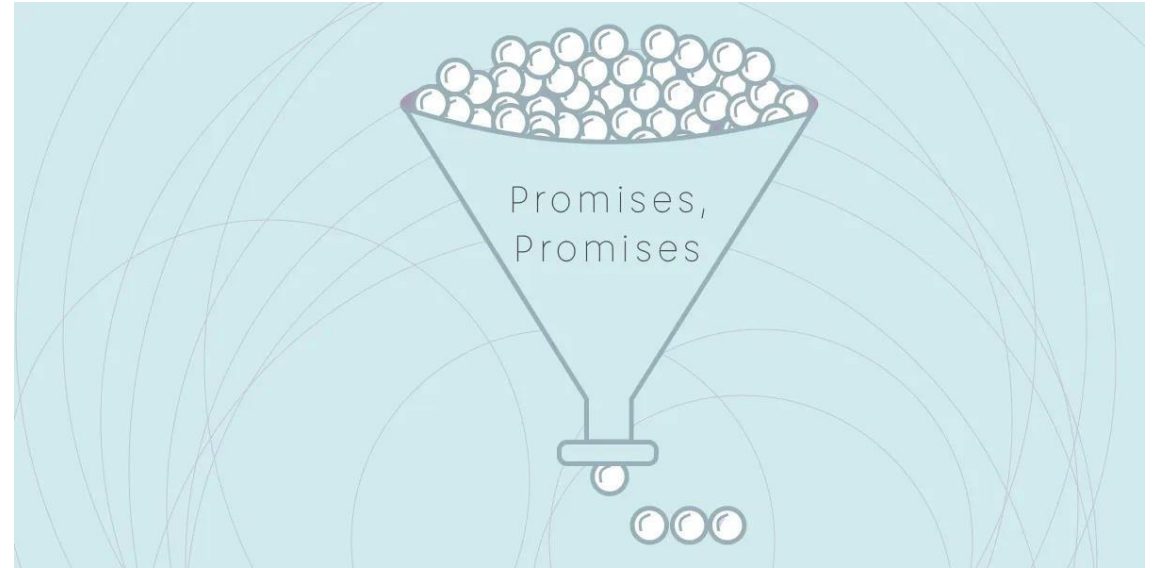
```
async function getProduct() {  
  try {  
    const idProduct = await api.getProduct(1)  
    const quantity = await api.getQuantityOfProduct(idProduct)  
  
    console.log(quantity)  
  } catch (err) {  
    console.log(err)  
  }  
}
```

❑ Cần lưu ý là kết quả trả về của async function luôn là một Promise.

## MỘT SỐ TRƯỜNG HỢP CẦN LƯU Ý

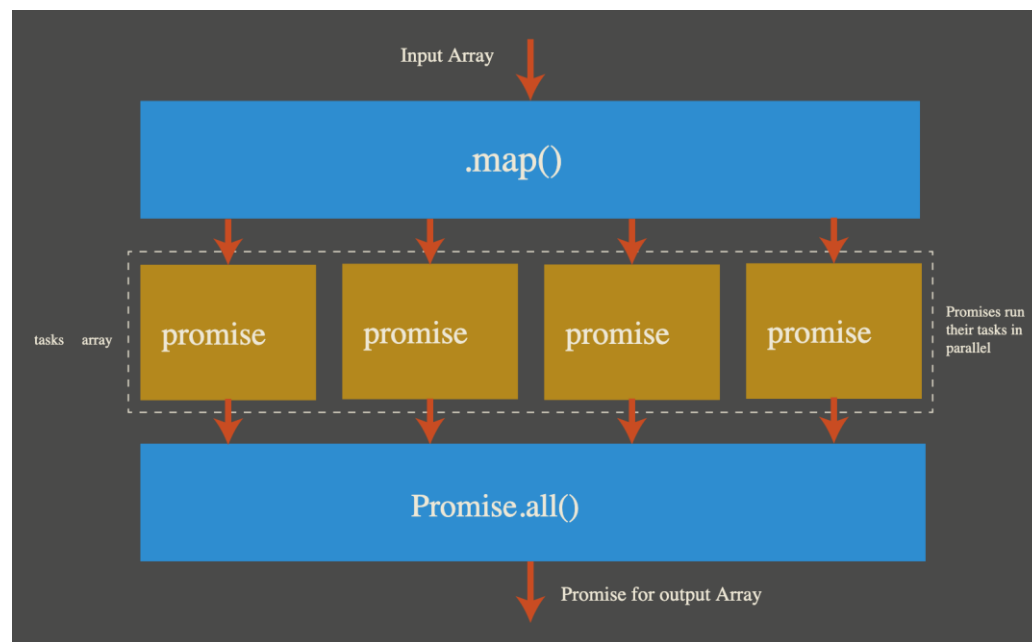
- ❖ Trong trường hợp muốn chạy các promises một cách tuần tự như ảnh bên, bạn có thể sử dụng cú pháp **Async/await**

```
async function() {  
  const res1 = await promise1()  
  const res2 = await promise2(res1)  
  const res3 = await promise3(res2)  
}
```



## MỘT SỐ TRƯỜNG HỢP CẦN LƯU Ý

- ❖ Trường hợp muốn lấy nhiều giá trị promise cùng một lúc, ngoài cách sử dụng **promise.all()** bạn có thể sử dụng cú pháp **async/await** như sau:



```
async function getUsers() {  
  const userIds = [1, 2, 3, 4]  
  const [user1, user2, user3, user4] = await Promise.all(usersIds.map(api.getUser))  
}
```

**▶ DEMO**

- 
- ❑ Yêu cầu: Viết lại ví dụ phía dưới từ cú pháp Promise Chaining, sử dụng `async/await` để thay thế `.then/catch`

```
function loadJson(url) {  
    return fetch(url)  
        .then(response => {  
            if (response.status == 200) {  
                return response.json();  
            } else {  
                throw new Error(response.status);  
            }  
        })  
}
```

```
loadJson('no-such-user.json')  
    .catch(alert); // Error: 404
```

- 
1. Khai báo hàm loadJson() thành **async**
  2. Tất cả .then bên trong thay thế bằng await
  3. Chờ đợi kết quả json trả về và gán vào biến json
  4. Nếu lỗi thì thông báo

```
async function loadJson(url) { // (1)
  let response = await fetch(url); // (2)

  if (response.status == 200) {
    let json = await response.json(); // (3)
    return json;
  }

  throw new Error(response.status);
}

loadJson('no-such-user.json')
  .catch(alert); // Error: 404 (4)
```

## PHẦN 3: SỬ DỤNG ASYNC VỚI AXIOS



- ❑ Axios là một thư viện JavaScript dựa trên Promised được sử dụng để gửi các yêu cầu HTTP. Bạn có thể coi nó như một sự thay thế cho hàm **fetch()** của JavaScript.
- ❑ Axios nó là một mã nguồn mở và được cung cấp miễn phí, bạn có thể truy cập [Github Respository](#) này để tìm hiểu nhiều hơn về Axios
- ❑ Do các tính năng và tính dễ sử dụng của nó, nó trở thành một lựa chọn phổ biến cho các nhà phát triển JavaScript khi làm việc với HTTP request.

- ❑ Trước khi bắt đầu bạn cần tạo một folder và cài đặt NPM

```
$ mkdir axios-tutorial  
$ cd axios-tutorial  
$ npm init -y
```

- ❑ Tiếp theo, sử dụng NPM để cài đặt axios

```
$ npm i --save axios
```

- ❑ Tạo một file mới đặt tên **getRequestPromise.js** và thực hiện theo đoạn mã dưới:

```
const axios = require('axios').default;

axios.get('https://jsonplaceholder.typicode.com/posts')
  .then(resp => {
    console.log(resp.data);
  })
  .catch(err => {
    // Handle Error Here
    console.error(err);
  });
```

- ❑ Nếu bạn chạy đoạn code trên với câu lệnh **node getRequestPromise.js** bạn sẽ nhìn thấy kết quả

```
[ { userId: 1,
  id: 1,
  title:
    'sunt aut facere repellat provident occaecati excepturi optio reprehenderit',
  body:
    'quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit molestiae ut ut quas totam\nnostrum rerum est autem sunt rem eveniet architecto' },
  { userId: 1,
    id: 2,
    title: 'qui est esse',
    body:
      'est rerum tempore vitae\nsequi sint nihil reprehenderit dolor beatae ea dolores neque\nfugiat blanditiis voluptate porro vel nihil molestiae ut reiciendis\nqui aperiam non debitis possimus qui neque nisi nulla' },
  ...
```

## THIẾT LẬP GET REQUEST

- Bây giờ, chúng ta sẽ viết lại code javascript thiết lập GET request sử dụng async/await

```
const axios = require('axios');

const sendGetRequest = async () => {
  try {
    const resp = await axios.get('https://jsonplaceholder.typicode.com/posts');
    console.log(resp.data);
  } catch (err) {
    // Handle Error Here
    console.error(err);
  }
};

sendGetRequest();
```

## THIẾT LẬP POST REQUEST

- ❑ Chúng ta cũng có thể thiết lập POST request để tạo tài nguyên mới trong REST API thông qua **async/await**.
- ❑ Tạo một file **postRequest.js** và thực hiện theo code bên dưới:

```
const axios = require('axios').default;

const newPost = {
  userId: 1,
  title: 'A new post',
  body: 'This is the body of the new post'
};

const sendPostRequest = async () => {
  try {
    const resp = await axios.post('https://jsonplaceholder.typicode.com/posts', newPost);
    console.log(resp.data);
  } catch (err) {
    // Handle Error Here
    console.error(err);
  }
};

sendPostRequest();
```

- ❑ Nếu bạn chạy đoạn code trên với câu lệnh **node postRequest.js**, trong terminal sẽ có kết quả:

```
{ userId: 1,  
  title: 'A new post',  
  body: 'This is the body of the new post',  
  id: 101 }
```

## THIẾT LẬP PUT REQUEST

- ❑ **PUT** request được sử dụng để thay đổi dữ liệu. Bạn có thể sử dụng phương thức **axios.put()**
- ❑ Tạo một file mới đặt tên **putRequest.js** và thực hiện theo đoạn mã sau:

```
const axios = require('axios').default;

const updatedPost = {
  id: 1,
  userId: 1,
  title: 'A new title',
  body: 'Update this post'
};

const sendPutRequest = async () => {
  try {
    const resp = await axios.put('https://jsonplaceholder.typicode.com/posts/1', updatedPost);
    console.log(resp.data);
  } catch (err) {
    // Handle Error Here
    console.error(err);
  }
};

sendPutRequest();
```



## THIẾT LẬP POST REQUEST

---

- ❑ Giống như POST request, khi PUT request được thực thi dữ liệu đã được thay đổi.
- ❑ Chạy câu lệnh **node putRequest** trong terminal sẽ có kết quả, bạn sẽ thấy kết quả trả về.

```
{ id: 1, userId: 1, title: 'A new title', body: 'Update this post' }
```

## THIẾT LẬP DELETE REQUEST

- ❑ Bạn có thể gửi yêu cầu HTTP **DELETE** bằng phương thức **axios.delete()** để xóa dữ liệu khỏi API RESTful.
- ❑ Tạo một file mới đặt tên **deleteRequest.js** và thực hiện theo đoạn mã sau đây:

```
const axios = require('axios').default;

const sendDeleteRequest = async () => {
  try {
    const resp = await axios.delete('https://jsonplaceholder.typicode.com/posts/1')
    console.log(resp.data);
  } catch (err) {
    // Handle Error Here
    console.error(err);
  }
};

sendDeleteRequest();
```

## THIẾT LẬP DELETE REQUEST

---

- ❑ Hàm **axios.delete()** chỉ cần URL của tài nguyên mà chúng ta muốn xóa. Thực thi chương trình này với câu lệnh **node putRequest.js** trong terminal sẽ có kết quả:

```
{}
```

- ☑ Ôn tập Promise
- ☑ Tổng quan Async & Await
- ☑ Hướng dẫn sử dụng Async với Axios

A rustic, handmade-style image featuring a rectangular piece of light brown cardboard or kraft paper. The words "Thank you!" are written on the tag in a black, cursive script. A black string is tied through a hole on the left side of the tag. Three white daisies with bright yellow centers are scattered around the tag: one is in sharp focus to the right of the tag, while two others are in the background, slightly out of focus. The entire scene is set against a background of light-colored, weathered wood with visible grain and texture.

Thank  
you!