



THỰC HỌC – THỰC NGHIỆP

JAVASCRIPT NÂNG CAO

JSON OBJECT & AJAX



- ⦿ Giải thích được khái niệm JSON Object
- ⦿ Giải thích được khái niệm Asynchronous JavaScript and XML (Ajax)
- ⦿ Thực hiện được việc gửi request tới server bằng javascript ajax (sử dụng fetch)
- ⦿ Sử dụng được thư viện bên thứ 3 (axios)

- 📖 Khái niệm JSON Object
- 📖 Khái niệm Asynchronous JavaScript and XML (Ajax)
- 📖 Sử dụng fetch để gửi request tới server
- 📖 Sử dụng thư viện bên thứ 3 (axios)



- 📖 Giải thích được khái niệm Callback Hell
- 📖 Biết các xử lý code để tránh gây ra Callback Hell
- 📖 Giải thích được khái niệm Promise
- 📖 Sử dụng được Promise trong xử lý bất đồng bộ



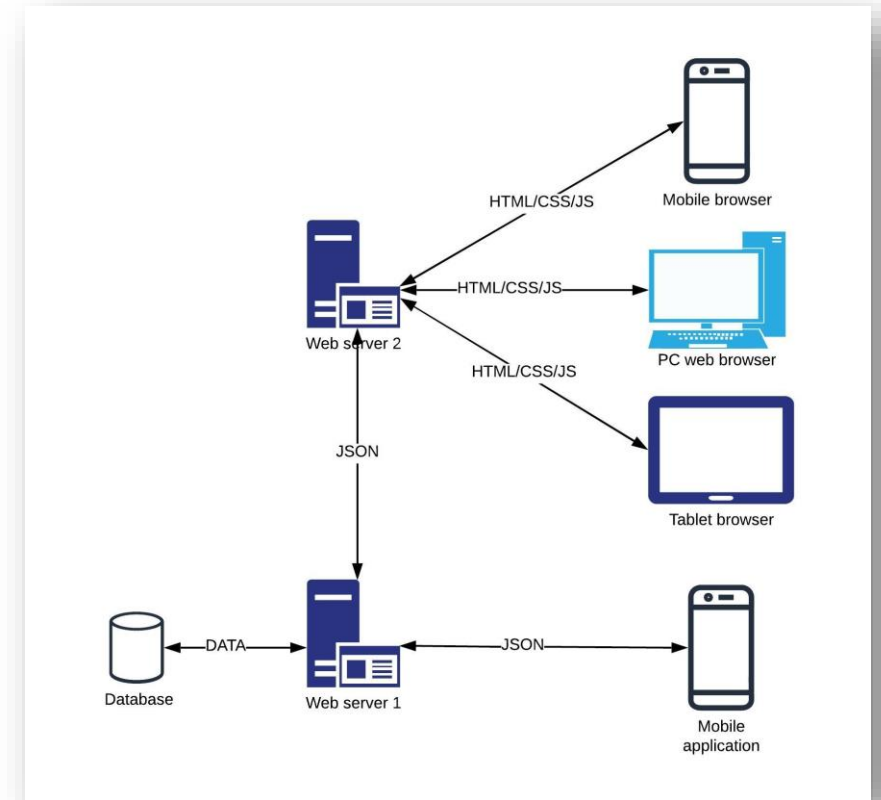


PHẦN 1: KHÁI NIỆM JSON OBJECT

❑ **JavaScript Object Notation** (thường được viết tắt là **JSON**) là một kiểu dữ liệu mở trong Javascript. Kiểu dữ liệu này bao gồm chủ yếu là text, có thể đọc được theo dạng cặp "thuộc tính - giá trị". Về cấu trúc, nó mô tả một vật thể bằng cách bọc những vật thể con trong vật thể lớn hơn trong dấu ngoặc nhọn (**{ }**). JSON là một kiểu dữ liệu trung gian, chủ yếu được dùng để vận chuyển thông tin giữa các thành phần của một chương trình.

TẠI SAO LẠI LỰA CHỌN JSON?

- ❑ Trong quá trình xây dựng 1 hệ thống sẽ có các thành phần khác nhau tham gia vào hệ thống, các thành phần này có thể được xây dựng bằng nhiều ngôn ngữ và nền tảng khác nhau (Java, C#, PHP, Android, Swift,...)
- ❑ Cần có một ngôn ngữ chung để các thành phần của hệ thống có thể trao đổi thông tin với nhau
- ❑ XML và sau này là JSON ra đời để thực hiện nhiệm vụ này



- ❑ JSON nhẹ hơn so với XML do đó có tốc độ truyền tải nhanh hơn. Điều này hợp lý cho những ứng dụng realtime (đạt được mức gần với thời gian thực)
- ❑ Cú pháp của JSON ngắn gọn hơn, dễ đọc và tạo đối tượng hơn so với XML
- ❑ JSON có thể được phân tích và xử lý dữ liệu nhanh hơn so với XML

- ❑ JSON là ngôn ngữ độc lập có nguồn gốc từ JavaScript.
- ❑ Chuỗi JSON được bao bọc bởi cặp dấu ngoặc nhọn {}
- ❑ Các key và value của JSON bắt buộc phải đặt trong cặp dấu nháy kép ""
- ❑ Dùng dấu phẩy "," để ngăn cách các cặp giá trị key:value
- ❑ Các key của JSON nên đặt tên giống như quy tắc đặt tên của biến (bao gồm chữ cái, chữ số, dấu gạch dưới, không bao gồm khoảng trắng và các ký tự đặc biệt, ký tự đầu tiên không phải là số.)

- ❑ Chuỗi (string): phải đặt trong cặp dấu ngoặc kép ""
- ❑ Số (Number): là một số nguyên hoặc là một số thực
- ❑ Đối tượng (đối tượng JSON)
- ❑ Mảng: được bao trong cặp dấu ngoặc vuông []
- ❑ Boolean
- ❑ NULL

❑ Ví dụ 1: Mô tả lại 1 sinh viên bằng JSON

```
{
  "mssv": "PH01679",
  "full_name": "Nguyễn Thành Nhân",
  "birth_date": "2001-03-08",
  "home_town": "Lào Cai",
  "id_number": 174280979,
  "is_married": false,
  "gender": "male"
}
```

❑ Ví dụ 2: Mô tả lại 2 sinh viên bằng JSON

```
[
  {
    "mssv": "PH01679",
    "full_name": "Nguyễn Thành Nhân",
    "birth_date": "2001-03-08",
    "home_town": "Lào Cai",
    "id_number": 174280979,
    "is_married": false,
    "gender": "male"
  },
  {
    "mssv": "PH09996",
    "full_name": "Nguyễn Thùy Linh",
    "birth_date": "2005-03-08",
    "home_town": "Yên Bái",
    "id_number": 175250374,
    "is_married": false,
    "gender": "female"
  }
]
```

❑ Ví dụ 3: Mô tả lại các đầu điểm của 3 bạn sinh viên trong lớp

```
{
  "pt15111_web": {
    "ph15427": {
      "com107": 7.5,
      "com107": 9,
      "web1013": 6
    },
    "ph15434": {
      "com107": 8,
      "com107": 7,
      "web1013": 6.5
    },
    "ph15488": {
      "com107": 10,
      "com107": 9,
      "web1013": 9
    }
  }
}
```

- ❑ Hiểu được định dạng JSON người học sẽ khá dễ dàng bắt đầu với những ứng dụng quản trị cơ sở dữ liệu dạng NoSQL Database



Amazon DynamoDB

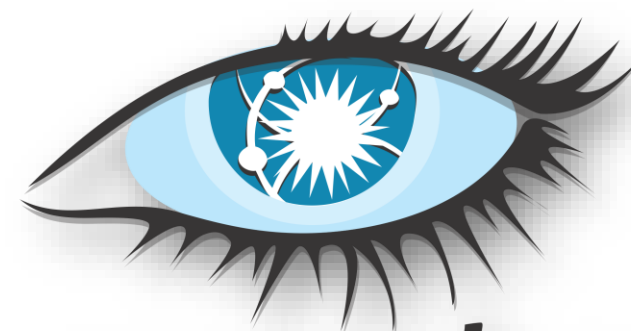
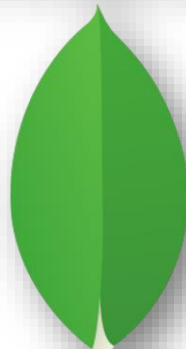


Notes and Domino



Firebase

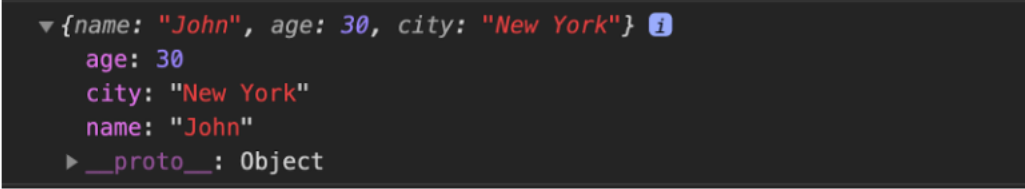
mongoDB®



cassandra

- ❑ JSON trong javascript về cơ bản cũng là 1 loại đối tượng, do đó khi tương tác với các giá trị trong đối tượng cũng áp dụng các cú pháp như học ở Slide 3: **"Lập trình hướng đối tượng trong Javascript"**

- ❑ Để sử dụng các hàm trong javascript tương tác với JSON

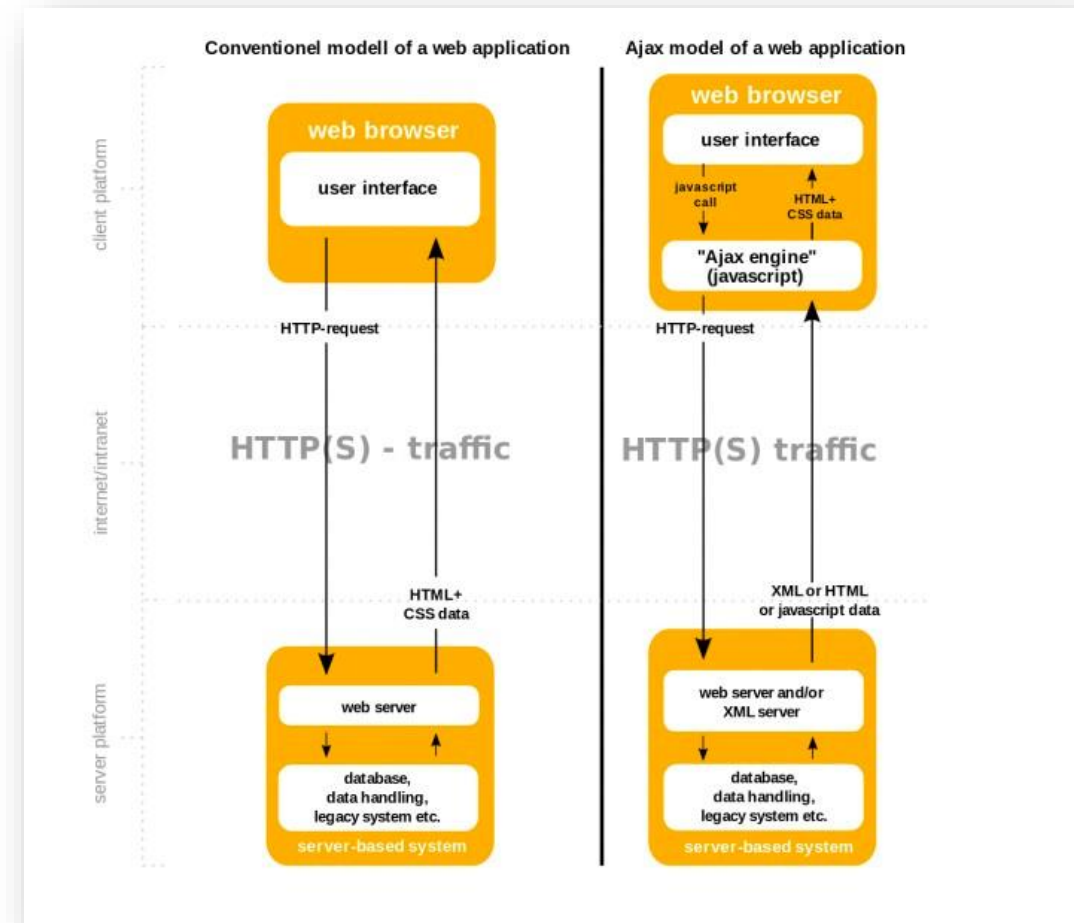
Phương thức	Mô tả	Ví dụ
JSON.parse(str)	Đưa 1 chuỗi (string) về đối tượng JSON	<pre>> var obj = JSON.parse('{ "name":"John", "age":30, "city":"New York"}'); console.log(obj);</pre> 
JSON.stringify(obj)	Đưa 1 đối tượng JSON về dạng chuỗi	<pre>> var obj = { name: "John", age: 30, city: "New York" }; var myJSON = JSON.stringify(obj); console.log(myJSON); typeof(myJSON);</pre> <pre>{"name":"John","age":30,"city":"New York"}</pre> <pre>< "string"</pre>



PHẦN 2: KHÁI NIỆM ASYNCHRONOUS JAVASCRIPT AND XML (AJAX)

- ❑ **AJAX** (Asynchronous JavaScript and XML) là một nhóm các công nghệ sử dụng để tạo các ứng dụng web động hay các ứng dụng giàu tính internet (*rich Internet application*).
- ❑ Từ *Ajax* được ông **Jesse James Garrett** đưa ra và dùng lần đầu tiên vào tháng 2 năm 2005 để chỉ kỹ thuật này, mặc dù các hỗ trợ cho *Ajax* đã có trên các chương trình duyệt từ 10 năm trước. Ajax là một kỹ thuật phát triển web có tính tương tác cao bằng cách kết hợp các ngôn ngữ:
 - ❖ HTML (hoặc XHTML) với CSS trong việc hiển thị thông tin
 - ❖ Mô hình DOM (Document Object Model), được thực hiện thông qua JavaScript, nhằm hiển thị thông tin động và tương tác với những thông tin được hiển thị
 - ❖ Đối tượng XMLHttpRequest để trao đổi dữ liệu một cách không đồng bộ với máy chủ web

- ❑ Ajax tự nó không phải là một công nghệ mà là một thuật ngữ mô tả việc sử dụng kết hợp một nhóm nhiều công nghệ với nhau.
- ❑ Về mô hình hoạt động của ajax, có thể thấy nó sử dụng javascript như bộ đệm để xử lý việc thay đổi các thành phần (hay thậm chí toàn bộ) trên màn hình giao diện. Bởi vì thực hiện bằng javascript cho nên người dùng sẽ không cảm thấy giao diện bị tải lại, việc này đem đến cho người dùng trải nghiệm tốt hơn.



- ❑ Bước 1: Thu thập dữ liệu, lên kịch bản cho sự kiện
- ❑ Bước 2: Khi sự kiện xảy ra, sử dụng javascript để gửi 1 request lên server (có thể sử dụng request GET, POST, PUT, PATCH,... tùy theo API)
- ❑ Bước 3: Server nhận dữ liệu, xử lý và trả về dữ liệu (thường là JSON)
- ❑ Bước 4: Javascript nhận dữ liệu gửi về, parse dữ liệu sang định dạng JSON sau đó bóc tách và có hành động thay đổi 1 phần (hoặc toàn bộ) dữ liệu đang có trên màn hình
- ❑ Do các bước trên được trình duyệt thực hiện ngầm dựa theo code javascript, do đó người dùng sẽ không cần phải tải lại trang theo cách truyền thống để gửi dữ liệu lên server



PHẦN 3: THỰC HÀNH SỬ DỤNG HÀM FETCH

- ❑ Trước đây (javascript ES5) để thực hiện gửi request từ js lên server để tương tác với dữ liệu bắt buộc phải sử dụng XMLHttpRequest hoặc ActiveXObject (đối với trình duyệt IE5, 6)
- ❑ Code ra hàm gửi request rất vất vả và hạn chế dần khi xu hướng công nghệ thay đổi, IE đã được khai tử bởi Microsoft.

❑ Ví dụ 4: sử dụng code để gửi request bằng XMLHttpRequest:

```
function load_ajax(url, method){
    // Tạo một biến lưu trữ đối tượng XML HTTP. Đối tượng này
    // tùy thuộc vào trình duyệt browser ta sử dụng nên phải kiểm
    // tra như bước bên dưới
    var xmlhttp;

    // Nếu trình duyệt là IE7+, Firefox, Chrome, Opera, Safari
    if (window.XMLHttpRequest)
    {
        xmlhttp = new XMLHttpRequest();
    }
    // Nếu trình duyệt là IE6, IE5
    else
    {
        xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
    }

    // Khởi tạo một hàm gửi ajax
    xmlhttp.onreadystatechange = function()
    {
        // Nếu đối tượng XML HTTP trả về với hai thông số bên dưới thì mọi chuyện
        // coi như thành công
        if (xmlhttp.readyState == 4 && xmlhttp.status == 200)
        {
            // Sau khi thành công tiến hành thay đổi nội dung của thẻ div, nội dung
            // ở đây chính là
            console.log(xmlhttp.responseText);
        }
    };

    // Khai báo với phương thức GET, và url chính là file result.php
    xmlhttp.open(method, url, true);

    // Cuối cùng là Gửi ajax, sau khi gọi hàm send thì function vừa tạo ở
    // trên (onreadystatechange) sẽ được chạy
    xmlhttp.send();

    // tải danh sách user từ api của my json server
    var url = "https://my-json-server.typicode.com/thienth/js-nang-cao/users";
    load_ajax(url, "GET"); // gọi hàm để lấy dữ liệu
}
```



```
[
  {
    "id": 1,
    "username": "admin",
    "password": "123456",
    "avatar": "https://loempixel.com/640/480/people/?78623",
    "role": "admin"
  },
  {
    "id": 2,
    "username": "member 1",
    "password": "123456",
    "avatar": "https://loempixel.com/640/480/people/?78624",
    "role": "member"
  },
  {
    "id": 3,
    "username": "member 3",
    "password": "123456",
    "avatar": "https://loempixel.com/640/480/people/?78625",
    "role": "member"
  },
  {
    "id": 4,
    "username": "member 4",
    "password": "123456",
    "avatar": "https://loempixel.com/640/480/people/?78626",
    "role": "member"
  }
]
```

- ❑ Khi chuẩn ECMAScript 2015 (ES6) ra đời, đi kèm với khái niệm **Promise**, javascript đồng thời bổ sung hàm **fetch()** giúp gửi request bằng javascript tới server để tương tác với dữ liệu đơn giản và ngắn gọn hơn rất nhiều.
- ❑ **fetch()** được viết trên nền của promise nên sẽ tránh được trạng thái Callback Hell, code dễ đọc và sáng sửa hơn.
- ❑ **fetch()** được nâng cấp theo các chuẩn mới của javascript. ECMAScript 2017 (ES8) fetch() tích hợp được cả với các tính năng mới như async/await

❑ Ví dụ 5: gửi request lên cùng url với ví dụ 4 nhưng sử dụng **fetch()**

```
var url = "https://my-json-server.typicode.com/thienth/js-nang-cao/users";

fetch(url, {
  method: "GET"
}).then(function(response){
  // chuyển dữ liệu trả về thành json
  return response.json();
}).then(function(data){
  console.log(data);
});
```



```
▼ (4) [{...}, {...}, {...}, {...}] ⓘ
  ▶ 0: {id: 1, username: "admin", password: "123456", avatar: "https://lorempixel.com/640/480/people/?78623", role: "admin"}
  ▶ 1: {id: 2, username: "member 1", password: "123456", avatar: "https://lorempixel.com/640/480/people/?78624", role: "member"}
  ▶ 2: {id: 3, username: "member 3", password: "123456", avatar: "https://lorempixel.com/640/480/people/?78625", role: "member"}
  ▶ 3: {id: 4, username: "member 4", password: "123456", avatar: "https://lorempixel.com/640/480/people/?78626", role: "member"}
    length: 4
  ▶ __proto__: Array(0)
```

❑ Hàm `fetch(url[, options])` – ý nghĩa các tham số:

- ❖ url: Đường dẫn gửi request
- ❖ Options: là 1 json object để config các thành phần method, body, headers,... cho request
 - Có thể có hoặc không, nếu không có thì method mặc định của request là dạng **GET**
 - method: "GET"|"POST" : thuộc tính quy định kiểu request
 - headers: { "Content-type": "application/x-www-form-urlencoded; charset=UTF-8" }: config cho header của request (<https://www.w3.org/TR/cors/>)
 - body: {data}: dữ liệu cần gửi lên server

❑ Ví dụ 6: gửi request dạng POST với dữ liệu lên server

```
const url = 'https://reqres.in/api/users';

// post body data
const user = {
  first_name: 'John',
  last_name: 'Doe',
  job_title: 'Blogger'
};

// request options
const options = {
  method: 'POST',
  body: JSON.stringify(user),
  headers: {
    'Content-Type': 'application/json'
  }
}

// send POST request
fetch(url, options)
  .then(res => res.json())
  .then(res => console.log(res));
```

- ❑ response là một object được trả về, chúng ta có thể truy xuất rất nhiều metadata dễ dàng từ object này.
 - ❖ response.status – http status ví dụ: 200, 404, 403, 500,...
 - ❖ response.ok – trả về true/false – trạng thái gửi request có thành công hay không
 - ❖ response.type – có 3 loại type:
 - basic: có nghĩa là không có giới hạn ai có thể xem response này
 - cors: nguồn gốc của request không nằm cùng nguồn của server, và server trả về response có kèm cùng CORS header
 - opaque: các response không có CORS headers đến từ server có nguồn gốc, theo spec thì response này bạn không thể truy cập bất cứ thông tin gì, và vì thế không thể biết được response có thành công hay không, hay bạn không có quyền truy xuất resource này

- ❑ Response cũng cung cấp các phương thức để trả về dữ liệu rất thuận tiện
 - ❖ `response.text()` – trả về dữ liệu được chuyển đổi sang dạng text
 - ❖ `response.json()` – trả về dữ liệu được chuyển đổi sang JSON object
 - ❖ `response.formData()` – trả về dữ liệu được chuyển sang dạng formData (<https://javascript.info/formdata>)
 - ❖ `response.blob()` – trả về định dạng blob (dữ liệu nhị phân có xác định kiểu)
 - ❖ `response.arrayBuffer()` – trả về dưới dạng ArrayBuffer (biểu diễn dữ liệu nhị phân ở mức độ thấp)

- ❑ Hàm fetch() được các đa số các trình duyệt hiện đại hiện nay hỗ trợ đầy đủ

Browser compatibility

[Update compatibility data on GitHub](#)

		Desktop						Mobile					
		Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	Android webview	Chrome for Android	Firefox for Android	Opera for Android	Safari on iOS	Samsung Internet
fetch	⚠	42	14	39	No	29	10.1	42	42	39	29	10.3	?
Support for blob: and data:	⚠	48	No	?	No	?	?	43	48	?	?	?	?
referrerPolicy		52	No	52	No	39	11.1	52	52	52	41	No	?
signal	⚠	66	16	57	No	53	11.1	66	66	57	47	11.3	No
Streaming response body	⚠	43	14	Yes 🚩	No	29	10.1	43	43	No	No	10.3	?

What are we missing?



Full support



No support



Compatibility unknown



Experimental. Expect behavior to change in the future.



See implementation notes.



User must explicitly enable this feature.



PHẦN 4: SỬ DỤNG THƯ VIỆN AXIOS

- ❑ Axios là một thư viện HTTP Client dựa trên Promise. Cơ bản thì nó cung cấp một API cho việc xử lý XHR (XMLHttpRequests)

The image shows the Axios logo, which consists of the word "AXIOS" in a bold, sans-serif font. The letter "A" is stylized with a blue diagonal bar on its left side, while the remaining letters "XIOS" are in black. The logo is centered within a light blue rectangular box that has a subtle drop shadow.

❑ Các tính năng nổi bật của axios

- ❖ Khi sử dụng Fetch, nếu khi server trả về các mã lỗi 4xx hay 5xx, thì hàm catch() của bạn sẽ không được gọi đến và người lập trình viên sẽ có nhiệm vụ phải tự kiểm tra trạng thái của mã trả về để xác định xem liệu request đó có thành công hay không. Trong khi đó, **Axios** sẽ reject tất cả các promise của request nếu một trong các mã lỗi trên được trả về.
- ❖ **Fetch** không tự động gửi trả cookies về cho server khi tạo một request. Ta sẽ cần phải truyền một cách trực tiếp các option để cho cookies có thể được include. Còn với **Axios** thì bạn không hề phải lo về vấn đề này.
- ❖ **Axios** được xây dựng dựa trên các XHR API cũ hơn, bạn có thể khai báo các hàm callback cho onUploadProgress và onDownloadProgress để hiển thị phần trăm thành công tại giao diện cho app

❑ Cài đặt:

- ❖ Nhúng mã `<script src="https://unpkg.com/axios/dist/axios.min.js"></script>` vào code html

```
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
<script>

  axios({
    method: "GET",
    url: 'https://reqres.in/api/users',
    responseType: 'json'
  })
  .then(function(response){
    console.log(response.data);
  })
  .catch(function (error) {
    // kiểm soát lỗi tại đây
    console.log(error);
  })
  .finally(function () {
    // các lệnh sẽ luôn chạy
    // bất kể request gửi có thành công hay không
  });

</script>
```

```
▼ {page: 1, per_page: 6, total: 12, total_pages: 2, data: Array(6)} ⓘ
  ▼ data: Array(6)
    ▼ 0:
      avatar: "https://s3.amazonaws.com/uifaces/faces/twitter/calebogden/128.jpg"
      email: "george.bluth@reqres.in"
      first_name: "George"
      id: 1
      last_name: "Bluth"
      ► __proto__: Object
    ► 1: {id: 2, email: "janet.weaver@reqres.in", first_name: "Janet", last_name: "We"}
    ► 2: {id: 3, email: "emma.wong@reqres.in", first_name: "Emma", last_name: "Wong",
    ► 3: {id: 4, email: "eve.holt@reqres.in", first_name: "Eve", last_name: "Holt", a
    ► 4: {id: 5, email: "charles.morris@reqres.in", first_name: "Charles", last_name:
    ► 5: {id: 6, email: "tracey.ramos@reqres.in", first_name: "Tracey", last_name: "R
      length: 6
      ► __proto__: Array(0)
    page: 1
    per_page: 6
    total: 12
    total_pages: 2
    ► __proto__: Object
```


❑ Sử dụng API của axios (config json object ở axios({config here}))

```
// `url` là đích đến của request
url: '/user',

// `method` là phương thức được sử dụng để thực hiện request
method: 'get', // mặc định là GET

// `baseUrl` sẽ được gán vào trước url khi url là đường dẫn tương đối.
baseUrl: 'https://some-domain.com/api/',
```

```
// `transformRequest` cho phép thay đổi dữ liệu trước khi gửi lên server
// Option này chỉ khả dụng cho các request có phương thức là 'PUT', 'POST', và 'PATCH'
// Hàm cuối cùng phải trả về một thể hiện của Buffer hoặc ArrayBuffer hoặc FormData hoặc Stream
// Bạn cũng có thể thay đổi header của request ở đây.
transformRequest: [function (data, headers) {
  // Thực hiện thay đổi dữ liệu

  return data;
}],

// `transformResponse` cho phép thay đổi dữ liệu trả về trước khi vào hàm xử lý trong then/catch
transformResponse: [function (data) {
  // Thực hiện việc thay đổi dữ liệu

  return data;
}],
```

❑ Sử dụng API của axios

```
// `headers` là các header được đặt lại trước khi gửi lên server
headers: {'X-Requested-With': 'XMLHttpRequest'},

// `params` là các tham số URL sẽ được gửi lên cùng request
// Giá trị của nó phải là một object thuần hoặc là một đối tượng URLSearchParams
params: {
  ID: 12345
},

// `paramsSerializer` là một hàm tùy chọn, có nhiệm vụ là serialize `params`
paramsSerializer: function(params) {
  return Qs.stringify(params, {arrayFormat: 'brackets'})
},

// `data` là dữ liệu sẽ được gửi theo body của request
// Chỉ khả dụng cho các request có phương thức là 'PUT', 'POST', và 'PATCH'
// Khi không cài đặt `transformRequest`, data phải thuộc một trong các kiểu sau:
// - Chuỗi, object thuần, ArrayBuffer, ArrayBufferView, URLSearchParams, FormData, File, Blob, Stream, Buffer
data: {
  firstName: 'Fred'
},
```

❑ Sử dụng API của axios

```
// `timeout` chỉ định số mili giây khi request vượt quá thời gian truy cập và bị hủy bỏ
timeout: 1000,

// `withCredentials` chỉ định có thực hiện các request cross-site Access-Control sử dụng credential hay không
withCredentials: false, // mặc định là false

// `responseType` chỉ định kiểu dữ liệu mà server sẽ trả về
// có thể là 'arraybuffer', 'blob', 'document', 'json', 'text', 'stream'
responseType: 'json', // default

// `xsrifCookieName` là tên của cookie được sử dụng như giá trị của xsrf token
xsrifCookieName: 'XSRF-TOKEN', // mặc định là 'XSRF-TOKEN'

// `xsrifHeaderName` là tên của header mang giá trị của xsrf token
xsrifHeaderName: 'X-XSRF-TOKEN', // mặc định là 'X-XSRF-TOKEN'

// `onUploadProgress` cho phép xử lý quá trình upload
onUploadProgress: function (progressEvent) {
  // Thực hiện việc thao tác với sự kiện progress
},

// `onDownloadProgress` cho phép xử lý quá trình download
onDownloadProgress: function (progressEvent) {
  // Thực hiện việc thao tác với sự kiện progress
},

// `maxContentLength` chỉ định độ dài tối đa mà response được trả về
maxContentLength: 2000,
```

❑ Sử dụng API của axios

```
// `maxLength` chỉ định độ dài tối đa mà response được trả về
maxLength: 2000,

// `validateStatus` chỉ định việc xử lý hay từ chối promise với HTTP response status được đưa ra
validateStatus: function (status) {
  return status >= 200 && status < 300; // trả về true hay null hay undefined thì sẽ xử lý, không thì sẽ từ chối
},

// `cancelToken` chỉ định một cancel token được dùng để hủy request
cancelToken: new CancelToken(function (cancel) {

})
```

- ❑ API của axios cung cấp đầy đủ các thiết lập để thực hiện được hầu hết các chức năng của một thư viện gửi request lên API để tương tác với dữ liệu.
- ❑ Axios cung cấp công cụ giống như Promise.all, giúp hỗ trợ việc thực hiện đồng thời nhiều request cùng 1 lúc, sau đó chờ và thu thập tất cả các dữ liệu nhận được từ request thì sẽ tiếp tục xử lý.

❑ Ví dụ 7: thực hiện nhiều request đồng thời

```
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
<script>

  function getListUser() {
    return axios({
      url: 'https://reqres.in/api/users/2',
      method: "GET"
    });
  }

  function getListProduct() {
    return axios({
      url: 'https://reqres.in/api/product/2',
      method: "GET"
    });
  }

  axios.all([getListUser(), getListProduct()])
    .then(axios.spread(function (users, products) {
      console.log(users, products);
    }));

</script>
```

```
▼ {data: {...}, status: 200, statusText: "", headers: {...}, config: {...}, ...} ⓘ
  ► config: {url: "https://reqres.in/api/users/2", method: "get", headers: {...}}
  ► data: {data: {...}}
  ► headers: {cache-control: "max-age=14400", content-type: "application/json"}
  ► request: XMLHttpRequest {readyState: 4, timeout: 0, withCredentials: false}
  status: 200
  statusText: ""
  ► __proto__: Object
▼ {data: {...}, status: 200, statusText: "", headers: {...}, config: {...}, ...} ⓘ
  ► config: {url: "https://reqres.in/api/product/2", method: "get", headers: {...}}
  ► data: {data: {...}}
  ► headers: {cache-control: "max-age=14400", content-type: "application/json"}
  ► request: XMLHttpRequest {readyState: 4, timeout: 0, withCredentials: false}
  status: 200
  statusText: ""
  ► __proto__: Object
```

- ☑ Giải thích được khái niệm JSON Object
- ☑ Giải thích được khái niệm Asynchronous JavaScript and XML (Ajax)
- ☑ Thực hiện được việc gửi request tới server bằng javascript ajax (sử dụng fetch)
- ☑ Sử dụng được thư viện bên thứ 3 (axios)





thank
you!