AJAX (Asynchronous JavaScript and XML)

You may be wondering what AJAX is and why you want to learn more about it, so let us take a look at how you can use AJAX with JavaScript to build dynamic web applications. Before we look at AJAX, let us talk about how a browser gets information to display a page. When a browser requests a page from a website, the browser in your machine is acting as a client. The website is being delivered through a machine that is serving up the information, so we call that a server. When you ask for a page, the client makes a request to the server and the server returns a page. Sometimes when you are looking at a page, you may want to ask for some additional information from the server.

Say for example that your page has thumbnails for a photo album and perhaps the album has more thumbnails than you can see in the current view. Without AJAX if you want to see more thumbnails, you would have to make a request back to the server and it would send you a whole new page with additional thumbnails. That means the server ends up sending a lot of information that you do not really need all over again. With AJAX, the server does not have to send you the whole page; it just sends the pieces you need-- in this case new thumbnails. So how does that work? AJAX is not a single technology, but a group of technologies working together to make that happen.

Technically, AJAX stands for Asynchronous JavaScript and XML. That is quite a mouthful, so let us break that down piece by piece:

- **Asynchronous** means that the client can request new pieces of information from the server at any time. It does not have to wait for a page to reload. A new request can be triggered by an event like clicking on a button, hovering over an image, or whatever. \
- The **J** in AJAX stands for **JavaScript**. JavaScript is where all this magic takes place. It handles the events that trigger a new request, makes the requests for new data to the server, and takes care of updating only the part of the document that needs to change.
- JavaScript talks to the server through a set of programming methods called an API and uses what has called an XHR or XML HTTP request. This is where the **X** in AJAX comes from. The XHR API lets the browser send and request data from a server. This can be a bit confusing because a lot of people think of XML as a language used to describe data that's a lot like HTML. Sometimes people assume that the X in AJAX means that the data from an AJAX request has to be in XML, but the data that is transferred to and from the server can be in any format, and it's usually either a text file, HTML, or a JavaScript object, like JSON.

AJAX is really just a fancy term for a technology that lets you build pages that update without requiring a page reload. JavaScript does most of the heavy lifting with AJAX, and it uses the XHR API to handle the communication between the client and the server.

With AJAX, you can:

- Update a web page without reloading the page
- Request data from a server - after the page has loaded
- Receive data from a server - after the page has loaded
- Send data to a server - in the background

How to check an API using online apps

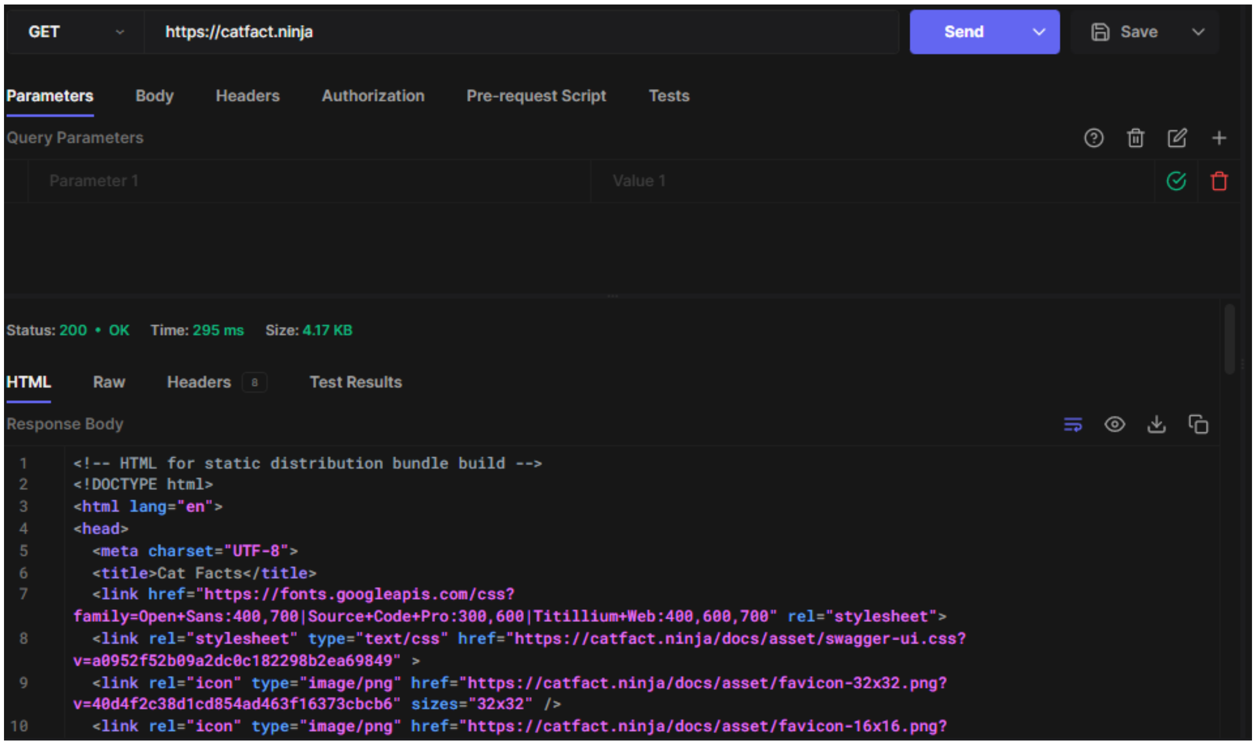
HOPP SCOTCH is a tool we use to make HTTP request to test out different APIs and to save requests. It helps us make API calls and test things out. You can always visit: https://hoppscotch.io/

To check an API response in HTML, we can type the API that we want to check, for example, if we want to test the following APIs

https://catfact.ninja/

https://swapi.dev/documentation#base

Let us try **catfacts.ninja**



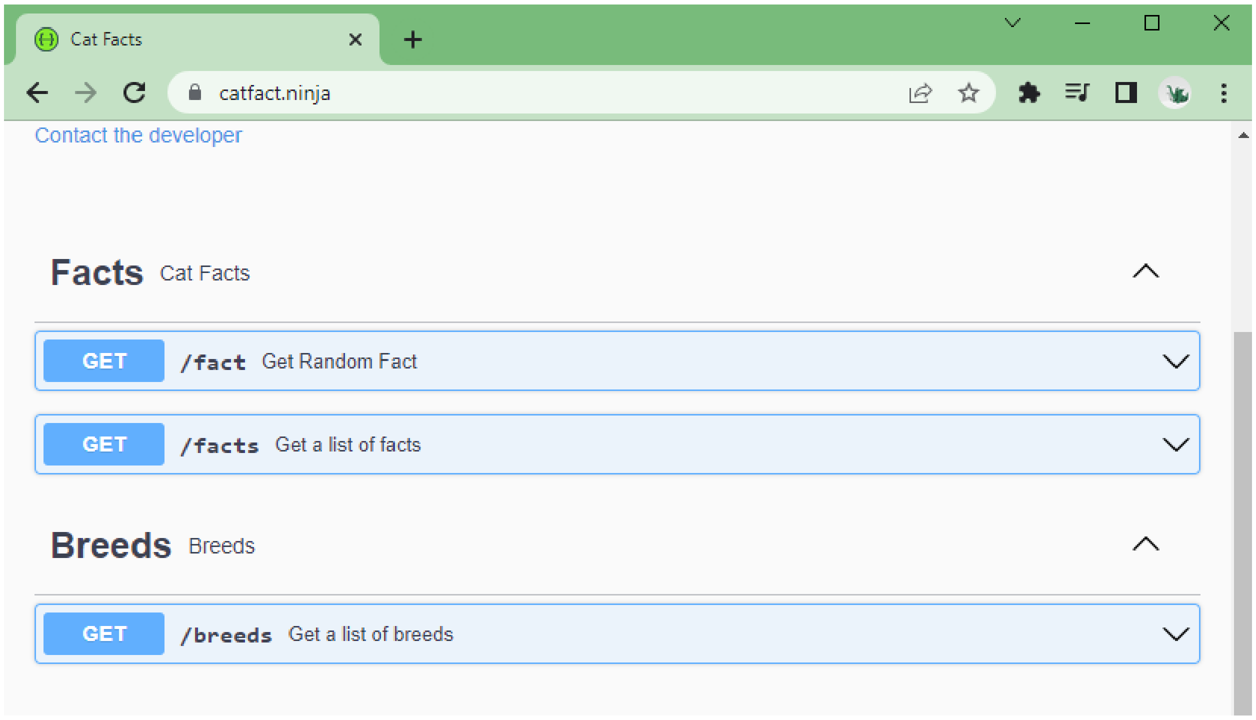
Status: 200 • OK   Time: 295 ms   Size: 4.17 KB

```
<!-- HTML for static distribution bundle build -->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Cat Facts</title>
  <link href="https://fonts.googleapis.com/css?
family=Open+Sans:400,700|Source+Code+Pro:300,600|Titillium+Web:400,600,700" rel="stylesheet">
  <link rel="stylesheet" type="text/css" href="https://catfact.ninja/docs/asset/swagger-ui.css?
v=a0952f52b09a2dc0c182298b2ea69849" >
  <link rel="icon" type="image/png" href="https://catfact.ninja/docs/asset/favicon-32x32.png?
v=40d4f2c38d1cd854ad463f16373cbcb6" sizes="32x32" />
  <link rel="icon" type="image/png" href="https://catfact.ninja/docs/asset/favicon-16x16.png?
```

It will open the HTML file.

Note: If you are running HOPP SCOTCH for the first time, it might open a pop-up window asking to selector the environment where you want to open the request, you can select **proxy**.

If we visit the website https://catfact.ninja/ , we can see how to use and navigate the information of *catfact's* API.


Cat Facts

catfact.ninja

Contact the developer

Facts Cat Facts

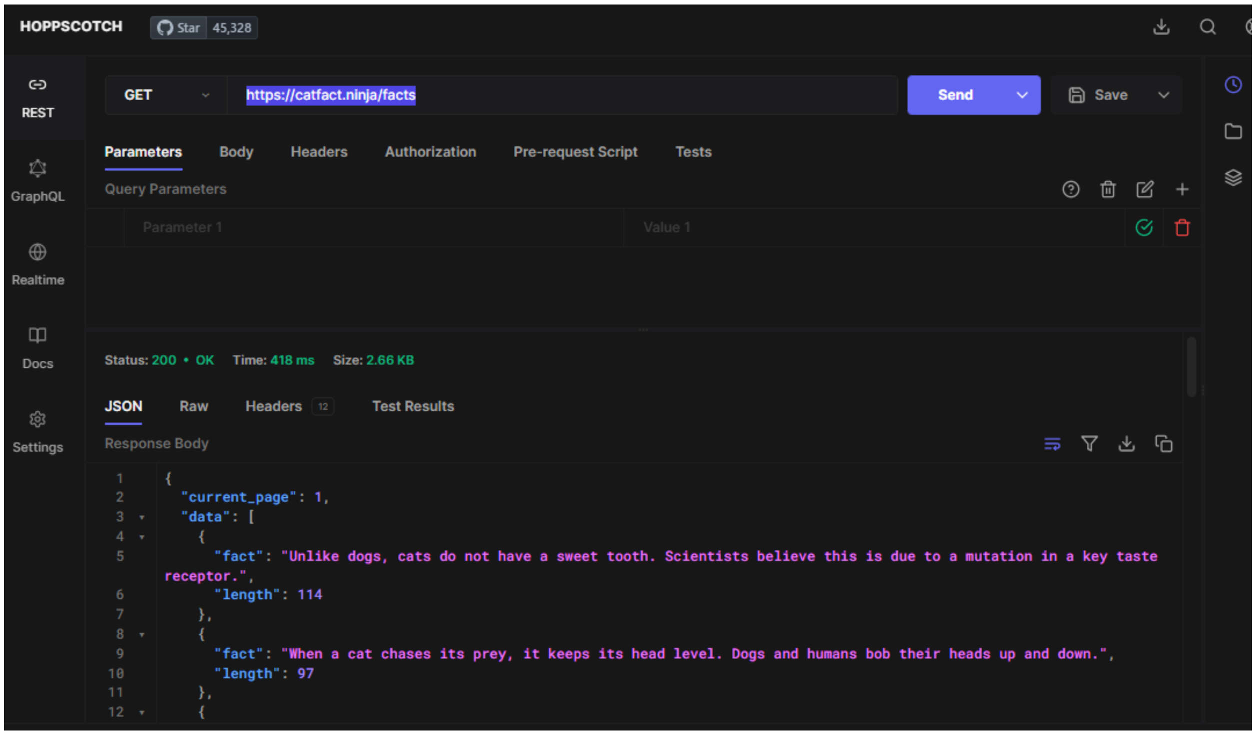
GET /fact Get Random Fact

GET /facts Get a list of facts

Breeds Breeds

GET /breeds Get a list of breeds

If we want to check some of the information about cats, let us say **facts**, we can type https://catfact.ninja/facts and click **Send**


HOPPSCOTCH Star 45,328

GET https://catfact.ninja/facts Send Save

Parameters Body Headers Authorization Pre-request Script Tests

Query Parameters

Parameter 1 Value 1

Status: 200 • OK Time: 418 ms Size: 2.66 KB

JSON Raw Headers 12 Test Results

Response Body

```
{
  "current_page": 1,
  "data": [
    {
      "fact": "Unlike dogs, cats do not have a sweet tooth. Scientists believe this is due to a mutation in a key taste receptor.",
      "length": 114
    },
    {
      "fact": "When a cat chases its prey, it keeps its head level. Dogs and humans bob their heads up and down.",
      "length": 97
    },
    {
```

In return, it will show us the JSON file about cat facts.

Data formats

The response to an AJAX request usually comes in one of the three formats:

- **HTML:** it is the simplest way to get data into a page.
- **XML**: it looks similar to HTML, but the tag names are different because they describe the data that they contain. The syntax is also more strict that HTML.
- **JSON:** JavaScript Object Notation uses a similar syntax to object literal notation in order to represent data.

XML: Extensible Markup Language

XML looks a lot like HTML, but the tags contain different words. The purpose of the tags is to describe the kind of data that they hold. Below is an example of a XML file:

```
<?xml version="1.0" encoding ="utf-8" ?>
<events>
  <event>
    <location>New York City</location>
    <date>May 13, 2020</date>
    <map>img/NYC_map.png</map>
  </event>
  <event>
    <location>Boston, MA</location>
    <date>May 5, 2020</date>
    <map>img/MA_map.png</map>
  </event>
</events>
```

JSON: JavaScript Object Notation

JavaScript JSON object can turn JSON data into a JavaScript object. It can also convert a JavaScript object into a string. It looks very similar to object literals syntax, but it is not object. It is just plain text data. It is like a dictionary that uses key to identify values. The XML code above in JSON text will be:

```
{
   "events":{
      {
        "location":"New York City",
        "date":"May 13, 2020",
        "map": "img/NYC_map.png"
      },
      {
      "location":"Boston, MA",
      "date":"May 5, 2020",
      "map": "img/MA_map.png"
      }
   }
}
```

The keys and the values of JSON have to be enclosed in a **double-quotation mark**. The keys are located at the left and the values at the right. Each key is separated by a comma.

$$\underbrace{\texttt{"location"}}_{\text{key}}\texttt{:}\underbrace{\texttt{"Boston, MA"}}_{\text{value}}$$

<u>How to access API in AJAX</u>

There are different ways that developers can access data in APIs. The following will introduce the earliest to the latest method to access and use data of API

<u>Handling AJAX requests and responses</u>

Using a synchronous XHR

The first step in working with AJAX is to learn about the API browsers provide for sending and retrieving information. The way you access the API is by using a XML HTTP request or XHR object.

When a server responds to any request, it should send back a status message, to indicate if it completed the request. The values can be:

200 ➔ The server has responded and all is ok

304 ➔ Not modified

404 ➔ Page not found

500 ➔ Internal error of the server

Remember that you get a server status property if you are working in a server. Otherwise, if the data is done locally, you will not get a server status property. You can check on w3schools website for more information about HTTP status messages.

<u>*XMLHttpRequest (XHR) objects*</u>

Use `XMLHttpRequest` (XHR) objects to interact with servers. You can retrieve data from a URL without having to do a full page refresh. This enables a Web page to update just part of a page without disrupting what the user is doing.

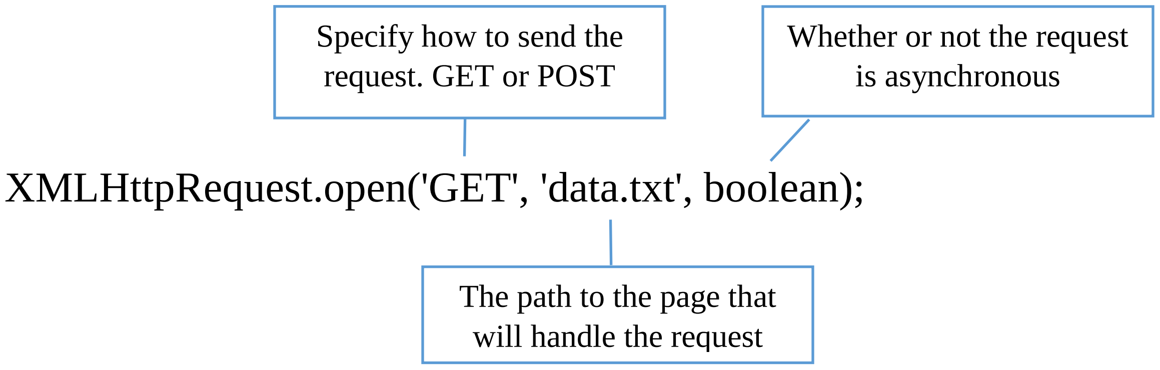
<u>*XMLHttpRequest constructor*</u>

The XMLHttpRequest Standard defines an API that provides scripted client functionality for transferring data between a client and a server.

Handling AJAX request and responses using XMLHttpRequest

This interface also inherits properties of *`XMLHttpRequestEventTarget`* and of *`EventTarget`*.

The XMLHttpRequest object's **open( )** method prepares the request. It has three parameters:


Specify how to send the request. GET or POST
Whether or not the request is asynchronous
XMLHttpRequest.open('GET', 'data.txt', boolean);
The path to the page that will handle the request

To create an AJAX request, browser use the **`XMLHttpRequest`** object. When the server responds to the browser's request, the same **`XMLHttpRequest`** object will process the result.

The request looks as the following

```
let xhr = new XMLHttpRequest();
xhr.open('GET', 'data/test.json', true);
xhr.send('null');
```

1. An instance of XMLHttpRequest object is constructor notation. It uses the **new** keyword and stores the object in a variable. The variable name **xhr** is short for **XMLHttpRequest**
2. The XMLHttpRequest object's **open()** method prepares the request. It has three parameter ('*the HTTP method*', '*the URL of the page that will handle the request*', *a Boolean indicating if it should be asynchronous*)
3. The **send()** method is the one that sends the prepared request to the server. Extra information can be passed to the server in the parentheses. If no extra information is sent, you may see the keyword **null** used

Example 1) create a request to obtain the first person in API https://swapi.dev

```
  <script type="text/javascript">
      const req = new XMLHttpRequest();

      req.onload = function(){
        console.log("LOADED!");
        console.log(this);
      }

      req.onerror = function(){
        console.log("ERROR!");
        console.log(this);
      }

      req.open("GET", https://swapi.dev/api/people/1/);
      req.send();
    </script>
```

If we want specific information from the request, we can create an object of the response and request specific properties:

```
const req = new XMLHttpRequest();

  req.onload = function(){
    console.log("LOADED!");
    console.log(this.response);
      const person = JSON.parse(this.response)
      console.log(`NAME: ${person.name}, HEIGHT: ${person.height}`)
  }

  req.onerror = function(){
    console.log("ERROR!");
    console.log(this);
  }

  req.open("GET", "https://swapi.dev/api/people/1/");
  req.send();
```

fetch

fetch is a JavaScript interface for making AJAX request. It is implemented widely in modern browsers and is used to call an API. Fetch returns a promise with a Response object. Promises enables a simpler and cleaner API, avoiding callback and having to remember the complex API of XMLHttpRequest.

Example 2) send the same request as in example 1 using fetch

```
fetch("https://swapi.dev/api/people/1/")
.then((res) =>{
  console.log("RESOLVED!", res);
  res.json().then((data)=>
  console.log("JSON RESULT", data));
});
```

It is recommended to complete one request at the time. Requests are independent, they don’t depend on one of another. However, sometimes you might have a request that you need to make first before you can make a second one.

Example 3) make three question using example 2

```
fetch("https://swapi.dev/api/people/1/")
.then((res) =>{
  console.log("1st REQUEST RESOLVED!", res);
  res.json().then((data)=>
  console.log("JSON RESULT REQ1", data));
});
fetch("https://swapi.dev/api/people/2/")
.then((res) =>{
  console.log("2nd REQUEST RESOLVED!", res);
  res.json().then((data)=>
  console.log("JSON RESULT REQ2", data));
});
fetch("https://swapi.dev/api/people/3/")
.then((res) =>{
  console.log("3rd REQUEST RESOLVED!", res);
  res.json().then((data)=>
  console.log("JSON RESULT REQ3", data));
});
```

If we need more than one request, we can use fetch in one variable function:

```
const loadPeople = async()=>{
  const res = await fetch("https://swapi.dev/api/people/1/")
  const data = await res.json()
  console.log(data)
  const res2 = await fetch("https://swapi.dev/api/people/2/")
  const data2 = await res2.json()
  console.log(data2)
}
loadPeople();
```

We catch an error by implementing the **try** and **catch** statement

```
const loadPeople = async()=>{
  try {
    const res = await fetch("https://swapi.dev/api/people/1/")
    const data = await res.json()
    console.log(data)
    const res2 = await fetch("https://swapi.dev/api/people/2/")
    const data2 = await res2.json()
    console.log(data2)
    } catch (e) {
      console.log("ERROR!",e)
    }
  }
loadPeople();
```

There is also a better way to request API by using **axios**

axios

axios is a library for making HTTP request. It uses fetch in the browser, but since it is not built on top of JavaScript, we have to go and import it or add it to our code so we can use it. Here is instruction on how to install AXIOS

https://github.com/axios/axios#installing

Instead of download axios and use it as an external file, we can use one library that will make requests from the browser, and at the same library will make request from the server side. For this, we can include a script in our HTML file:

Using jsDelivr CDN:

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

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

AXIOS has a couple of methods on its own, we can always review those methods by scrolling down on the axios github.

How to return data from axios API with JavaScript?

To return data from axios API with JavaScript, we can use the return statement.

For instance, we use the following syntax:

```
const axiosTest = async () => {
  const response = await axios.get(url);
  return response.data;
};
```

- To define the **axiosTest** function that returns the response data from the **axios.get** method.
- We call **axios.get** with **url** to return a promise with the response from making a get request to the url.
- Then we return **response.data** to return the data.

Next, in an **async** function, we write:

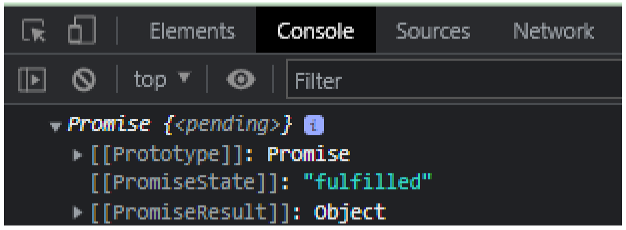
```
const data = await axiosTest();
```

to get the response data from **axiosTest**.[1]

Example 4) use axios to perform the same operation that example 3

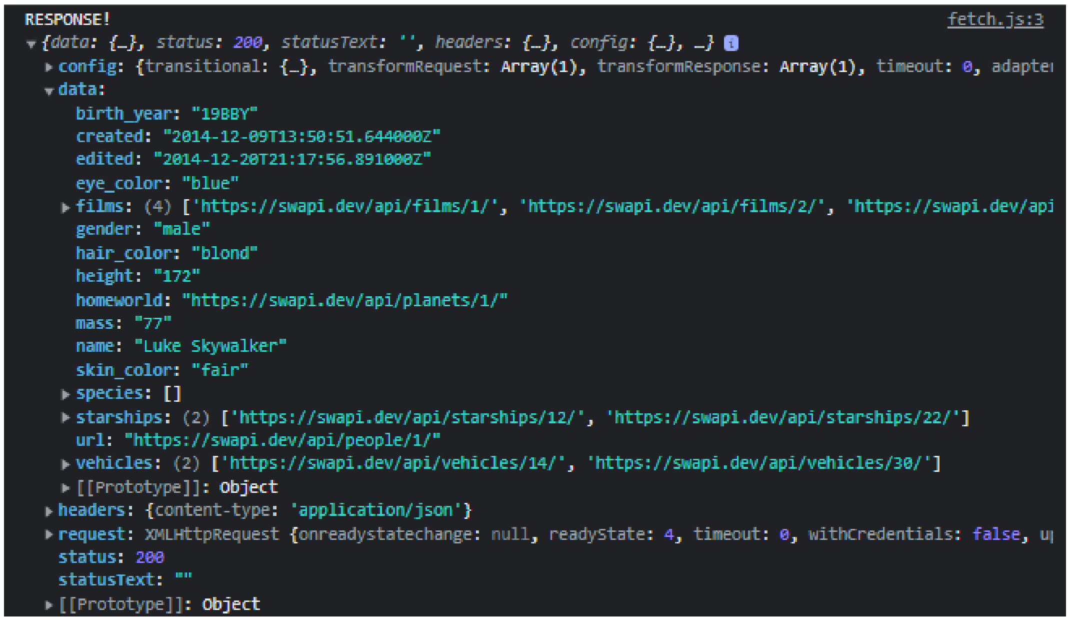
First of all, let us check how axios return looks like:

```
let axio = axios.get("https://swapi.dev/api/people/1/")
console.log(axio)
```



```
▾ Promise {<pending>}
  ▸ [[Prototype]]: Promise
    [[PromiseState]]: "fulfilled"
  ▸ [[PromiseResult]]: Object
```

```
let axio = axios.get("https://swapi.dev/api/people/1/")
.then(res =>{
    console.log("RESPONSE!", res)
})
```



```
RESPONSE!                                                                                   fetch.js:3
▾ {data: {…}, status: 200, statusText: '', headers: {…}, config: {…}, …}
  ▸ config: {transitional: {…}, transformRequest: Array(1), transformResponse: Array(1), timeout: 0, adapter
  ▾ data:
      birth_year: "19BBY"
      created: "2014-12-09T13:50:51.644000Z"
      edited: "2014-12-20T21:17:56.891000Z"
      eye_color: "blue"
    ▸ films: (4) ['https://swapi.dev/api/films/1/', 'https://swapi.dev/api/films/2/', 'https://swapi.dev/api
      gender: "male"
      hair_color: "blond"
      height: "172"
      homeworld: "https://swapi.dev/api/planets/1/"
      mass: "77"
      name: "Luke Skywalker"
      skin_color: "fair"
    ▸ species: []
    ▸ starships: (2) ['https://swapi.dev/api/starships/12/', 'https://swapi.dev/api/starships/22/']
      url: "https://swapi.dev/api/people/1/"
    ▸ vehicles: (2) ['https://swapi.dev/api/vehicles/14/', 'https://swapi.dev/api/vehicles/30/']
    ▸ [[Prototype]]: Object
  ▸ headers: {content-type: 'application/json'}
  ▸ request: XMLHttpRequest {onreadystatechange: null, readyState: 4, timeout: 0, withCredentials: false, u
    status: 200
    statusText: ""
  ▸ [[Prototype]]: Object
```

[1] *How to return data from axios API with JavaScript*, The Web Dev, https://thewebdev.info/2022/05/15/how-to-return-data-from-axios-api-with-javascript/, retrieved July 2022

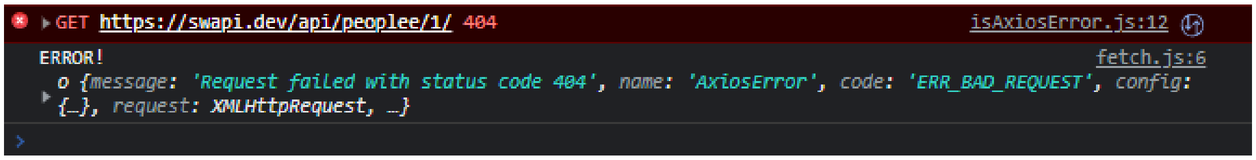
We can just add a **`catch()`** method after the response to handle any error:

```
.catch(e=>{
  console.log("ERROR!", e)
})
```

For example, if we made a mistake in the URL link:

```
let axio = axios.get("https://swapi.dev/api/peoplee/1/")
```

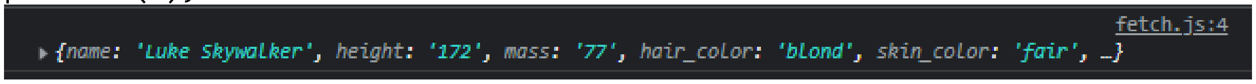
the console will display:



```
▸ GET https://swapi.dev/api/peoplee/1/ 404                    isAxiosError.js:12
  ERROR!                                                               fetch.js:6
▸ o {message: 'Request failed with status code 404', name: 'AxiosError', code: 'ERR_BAD_REQUEST', config:
  {…}, request: XMLHttpRequest, …}
>
```

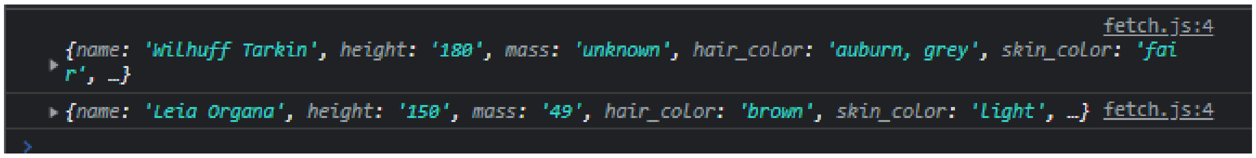
We can also get specific response from the same API:

```
const personID = async(id)=>{
  const res = await axios.get(`https://swapi.dev/api/people/${id}/`)
  console.log(res.data);
}
personID(1);
```



```
                                                                       fetch.js:4
▸ {name: 'Luke Skywalker', height: '172', mass: '77', hair_color: 'blond', skin_color: 'fair', …}
```

If we pick the 5th and 12th person, the response will be:



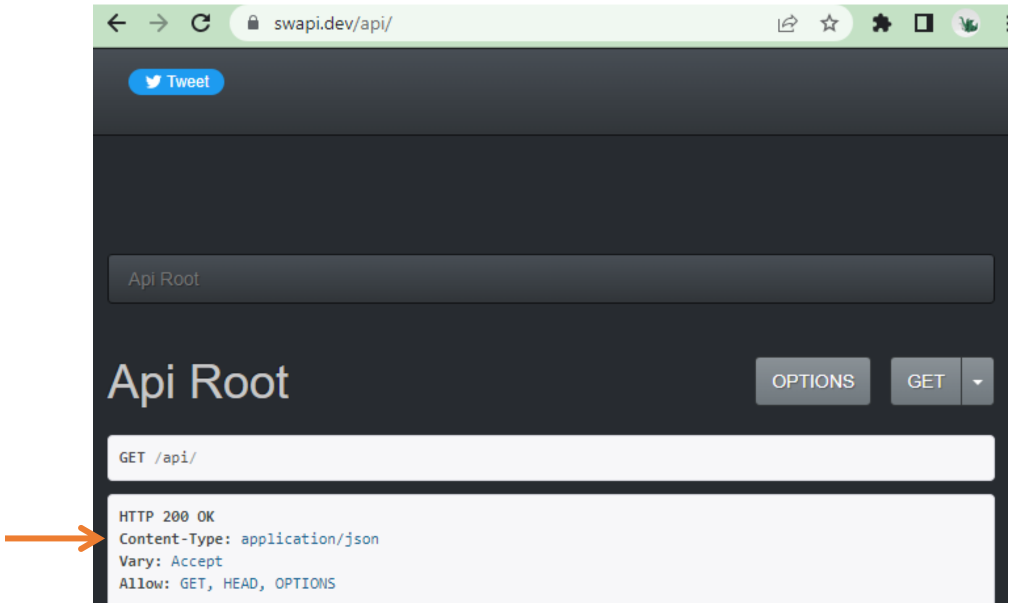
```
                                                                       fetch.js:4
▸ {name: 'Wilhuff Tarkin', height: '180', mass: 'unknown', hair_color: 'auburn, grey', skin_color: 'fai
  r', …}
▸ {name: 'Leia Organa', height: '150', mass: '49', hair_color: 'brown', skin_color: 'light', …} fetch.js:4
>
```

After it, we can include **try** and **catch** statement to handle error

```
const personID = async(id)=>{
  try {
    const res = await axios.get(`https://swapi.dev/api/people/${id}/`)
    console.log(res.data);
  } catch (e) {
    console.log("ERROR", e)
  }
}
personID(5);
personID(12);
```

How to get a specific data from the response

We can start checking of the header of the API to see what type of application API accepts. For this, you can go to the API website and check the headers:



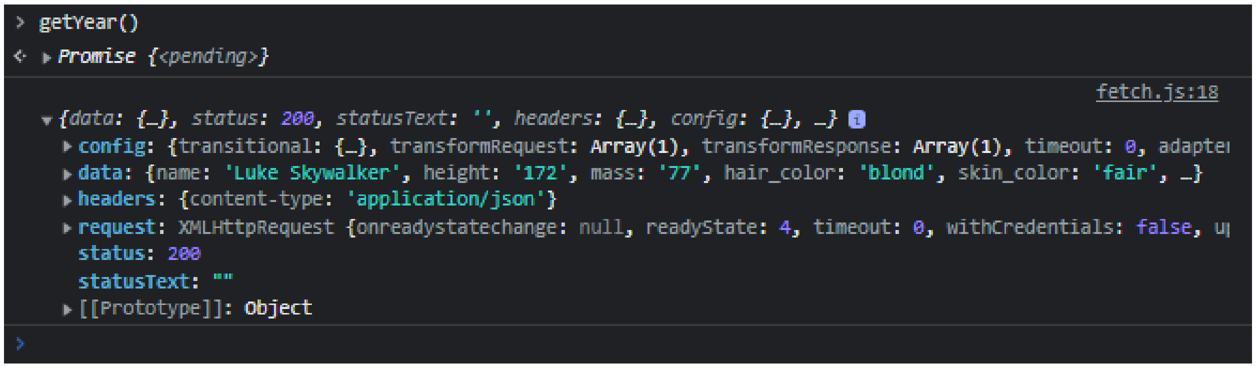
```
GET /api/

HTTP 200 OK
Content-Type: application/json
Vary: Accept
Allow: GET, HEAD, OPTIONS
```

Once we know the Content-Type, we can start writing our request:

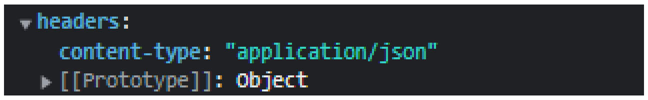
```
const getYear = async ()=>{
  const config = {headers: {Accept:'application/json'}}
  const res = await axios.get('https://swapi.dev/api/people/1/')
  console.log(res)
}
```

Now, if we call the function in the console:



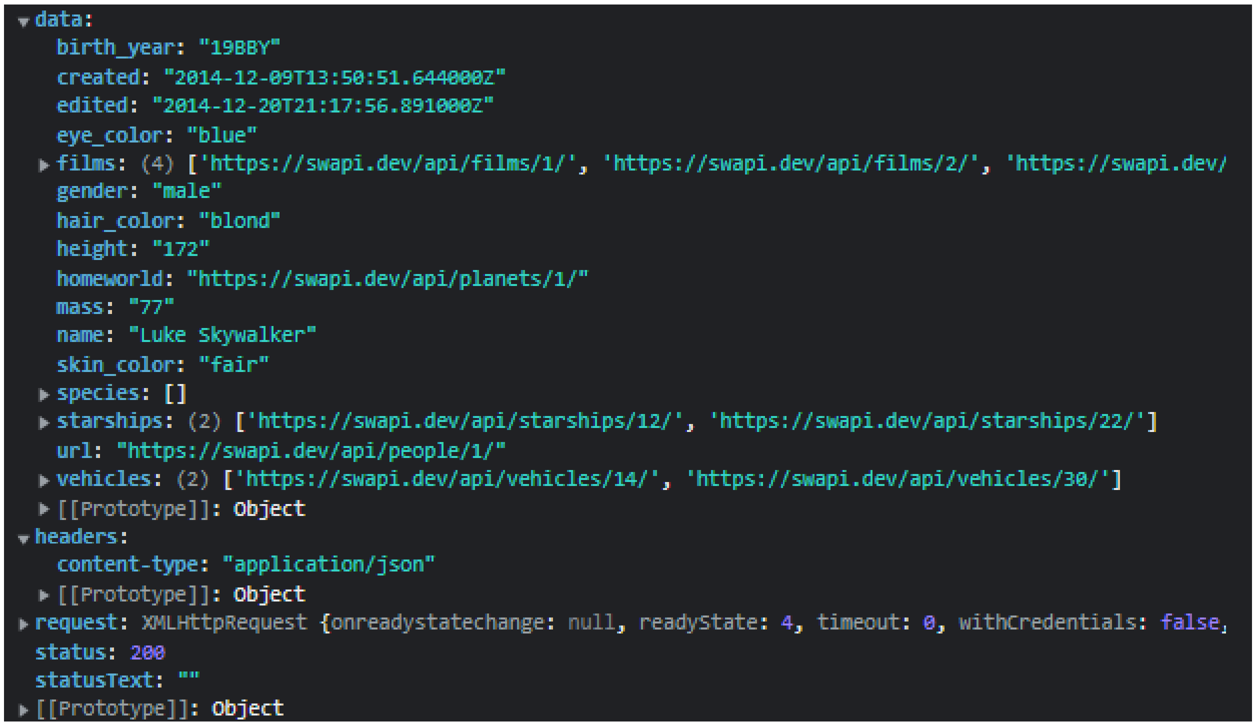
```
> getYear()
<· ▸ Promise {<pending>}
                                                                         fetch.js:18
  ▾{data: {…}, status: 200, statusText: '', headers: {…}, config: {…}, …}
    ▸config: {transitional: {…}, transformRequest: Array(1), transformResponse: Array(1), timeout: 0, adapte
    ▸data: {name: 'Luke Skywalker', height: '172', mass: '77', hair_color: 'blond', skin_color: 'fair', …}
    ▸headers: {content-type: 'application/json'}
    ▸request: XMLHttpRequest {onreadystatechange: null, readyState: 4, timeout: 0, withCredentials: false, u
     status: 200
     statusText: ""
    ▸[[Prototype]]: Object
>
```

If we expand the headers, we can see that it is a **JSON** file in response



```
▾headers:
   content-type: "application/json"
  ▸[[Prototype]]: Object
```

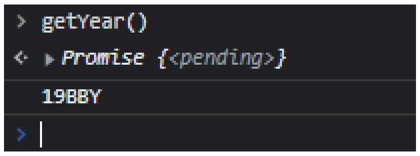
Then, we can expand the **data** object to check the properties that we can use:



```
▾ data:
    birth_year: "19BBY"
    created: "2014-12-09T13:50:51.644000Z"
    edited: "2014-12-20T21:17:56.891000Z"
    eye_color: "blue"
  ▸ films: (4) ['https://swapi.dev/api/films/1/', 'https://swapi.dev/api/films/2/', 'https://swapi.dev/
    gender: "male"
    hair_color: "blond"
    height: "172"
    homeworld: "https://swapi.dev/api/planets/1/"
    mass: "77"
    name: "Luke Skywalker"
    skin_color: "fair"
  ▸ species: []
  ▸ starships: (2) ['https://swapi.dev/api/starships/12/', 'https://swapi.dev/api/starships/22/']
    url: "https://swapi.dev/api/people/1/"
  ▸ vehicles: (2) ['https://swapi.dev/api/vehicles/14/', 'https://swapi.dev/api/vehicles/30/']
  ▸ [[Prototype]]: Object
▾ headers:
    content-type: "application/json"
  ▸ [[Prototype]]: Object
▸ request: XMLHttpRequest {onreadystatechange: null, readyState: 4, timeout: 0, withCredentials: false,
  status: 200
  statusText: ""
▸ [[Prototype]]: Object
```

Let us say that we want only the year of birth of the first person. For this, we change the response line to:

```
const getYear = async ()=>{
  const config = {headers: {Accept:'application/json'}}
  const res = await axios.get('https://swapi.dev/api/people/1/')
  console.log(res.data.birth_year)
}
```



```
> getYear()
<· ▸ Promise {<pending>}
  19BBY
> |
```

Supporting Material

What is AJAX, https://youtu.be/zibNBOBsDZk (video)

HTTP status codes, https://developer.mozilla.org/en-US/docs/Web/HTTP/Status (Website*)*

Introduction to Flask and handling AJAX API calls, https://youtu.be/r61j81dWyPk (video)