AJAX | Axios GET Request

Learning Goals

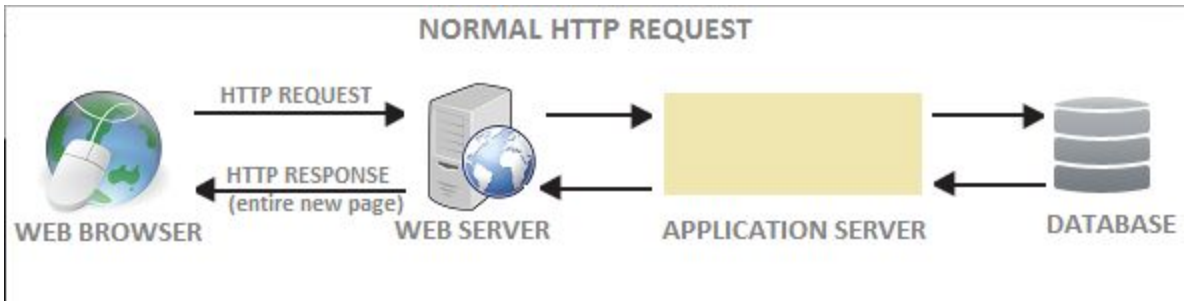
In this learning unit you will be able to:

- Understand what AXIOS is and what it can be used for
- Make AJAX GET requests to Web APIs
- Understand what Axios is and what it can be used for
- Use Axios to make AJAX GET requests to Web APIs
- Retrieve data from a Web API to use it in your own app

Introduction

There are a lot of resources on the Internet that we can use for our own applications, not only for retrieving data but for enabling functionalities too. Think of Google Maps, Mailchimp or many others.

On the other hand, sometimes we simply want to get information to store it in our application's database or as an input for our apps logic in the server.



The problem comes when we want to include that functionality in our application and use it without reloading the entire HTML every time we make a request to a remote server.

Here is where Axios saves our day.

Intro to Axios

Axios is a **Promise-based** HTTP client for JavaScript which can be used in your front-end application and in your Node.js backend.

By using Axios it's easy to send asynchronous HTTP request to REST endpoints and perform CRUD operations.

Now, this being said in human understandable laguage: AJAX allows **web pages** to be **updated asynchronously** by **exchanging data with a web server behind the scenes**. This means that it is possible to **update parts of a web page, without reloading the whole page**.

Axios is relying on AJAX technology. **AJAX** is a technique for accessing web servers from a web page and stands for *Asynchronous JavaScript And XML*.

Axios - Main Features

- Make XMLHttpRequests from the browser
- Make `HTTP` requests from Node.js
- Supports the Promise API
- Intercept requests and responses
- Transform requests and responses data
- Cancel requests
- Automatic transforms JSON into JS objects
- Client-side support for protecting against XSRF

Request method aliases

For convenience, the aliases have been provided for all supported request methods:

- `axios.request(config)`
- `axios.get(url[, config])`
- `axios.delete(url[, config])`
- `axios.head(url[, config])`
- `axios.options(url[, config])`
- `axios.post(url[, data[, config]])`
- `axios.put(url[, data[, config]])`

- `axios.patch(url[, data[, config]])`

When using the alias method `url`, the `data` and `config` properties don't need to be specified (that's why they are in the `[]`, which means they are optional).

Common Axios uses

Some use-cases when to use Axios calls could be:

- When updating data or part of the structure of an HTML
- In form validations
- When filtering data
- When making comments
- When reading files

Installing Axios

We can install Axios with **NPM** or we can add it through a CDN (easier and faster 😉).

Using CDN

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

Using NPM

```
$ npm install axios
```

API testing

The Internet is filled with free APIs available for developers. Google, YouTube, Yahoo, Instagram, Twitter, Flickr, LinkedIn and many more huge and famous applications provide a free API to integrate to our platform.

To demonstrate how AJAX works, we will use the `REST COUNTRIES API`.

The REST COUNTRIES API

The restcountries is a REST API created by Faydel Florez and acquired by apilayer.

REST COUNTRIES provides a simple API for getting information about the world's nations via REST calls. These calls allow users to retrieve all available countries or to retrieve a given country's currency, capital city, calling code, region, sub-region, ISO 639-1 language, name, or country code.

If we visit restcountries end points we can see which REST endpoints are available for us to use to search for countries.

Some examples are:

- search by country name: `https://restcountries.eu/rest/v2/name/{name}`

- search by capital city: `https://restcountries.eu/rest/v2/capital/{capital}`

- Search by ISO 4217 currency code:
`https://restcountries.eu/rest/v2/currency/{currency}`

But let's see it in practice.

API testing through browser's navigation bar

If you copy any of the links into the browser's navigation bar and replace the part within `{}` with real value, you can test any end point.

After inputting this URL: https://restcountries.eu/rest/v2/name/spain, you should see something like this:

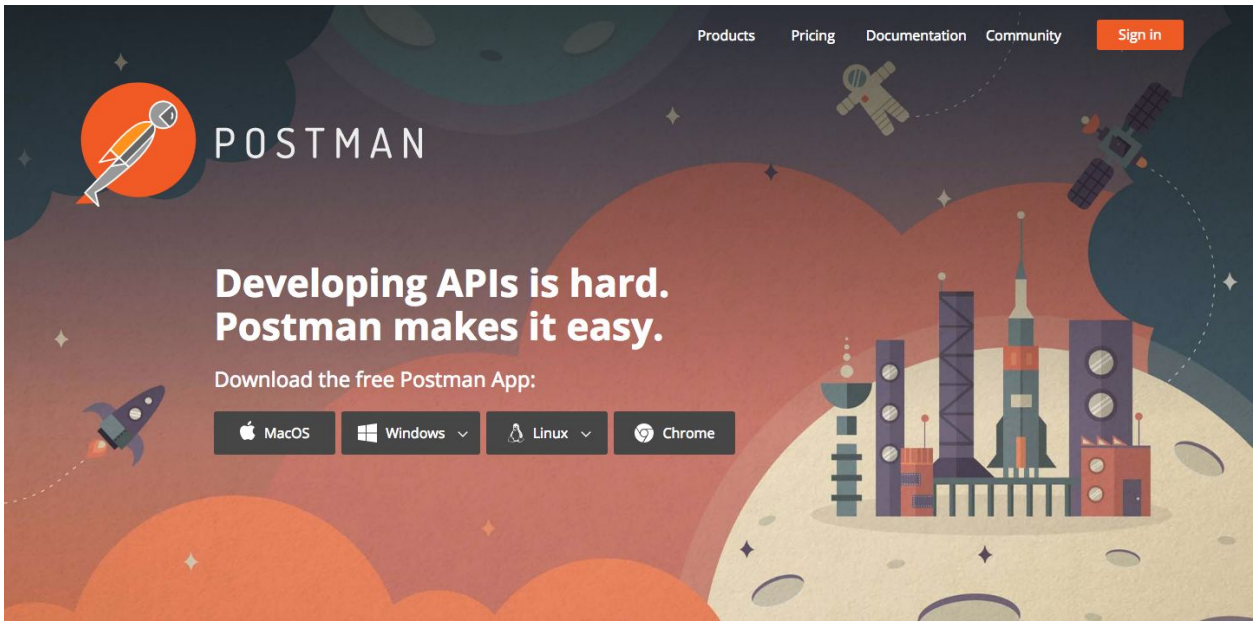
It is possible that your browser shows a different and unformatted object. If that happens, try installing the **JSON Viewer** extension.**

Postman

Postman allows us to test Web API requests from our system. We could set the desired parameters as if we were in our own code.

We can use any HTTP verb for the request and we can use any amount of parameters.

Installing Postman



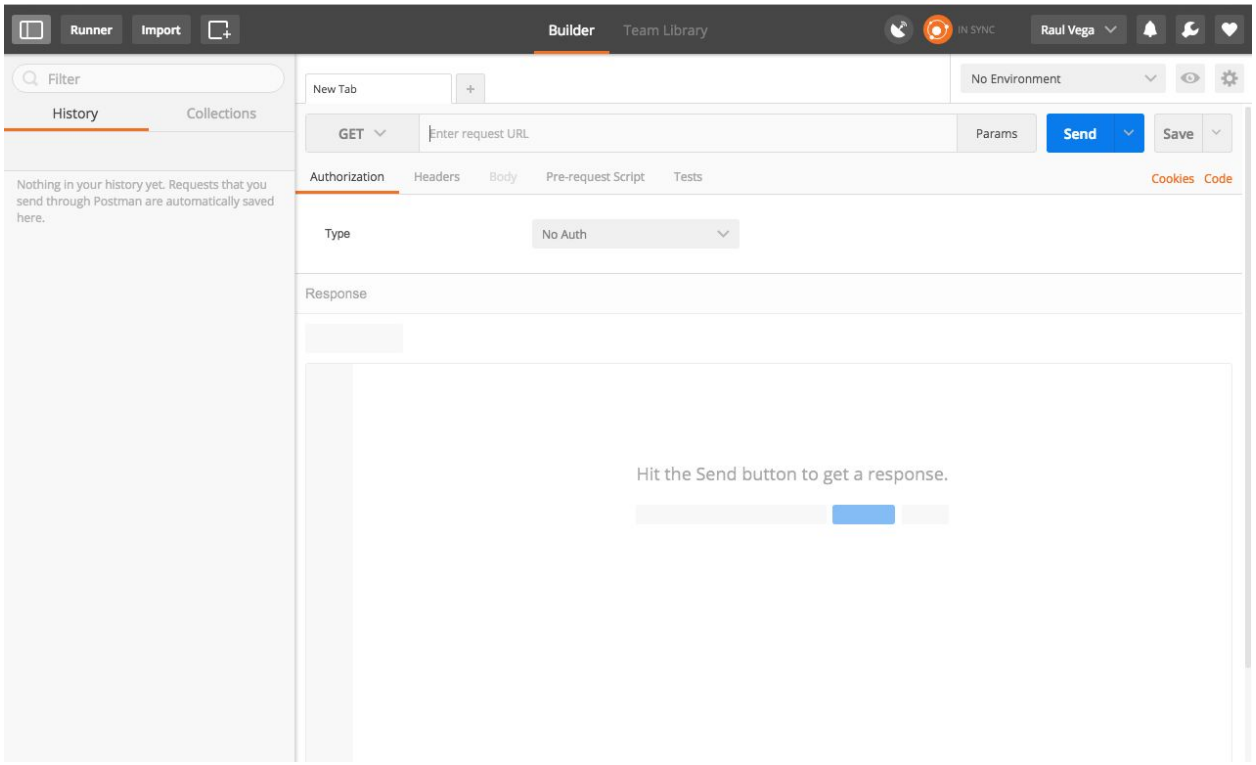
Go to the official website and download Postman. Be careful to select the right Operating System for your computer.

We can use Postman as a Google Chrome extension too, but we will use the GUI version since is more intuitive.

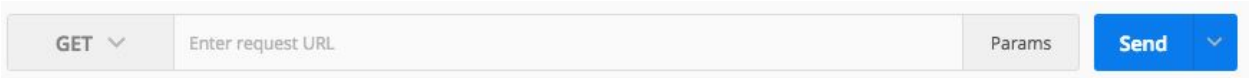
You can also use Insomnia which has similar features.

GET request

Open Postman. You will see this:

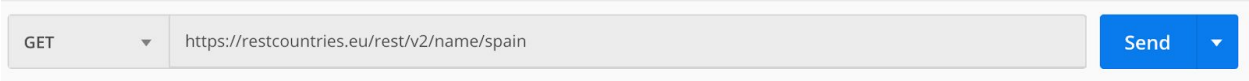


For now, we will focus on the navigation bar:

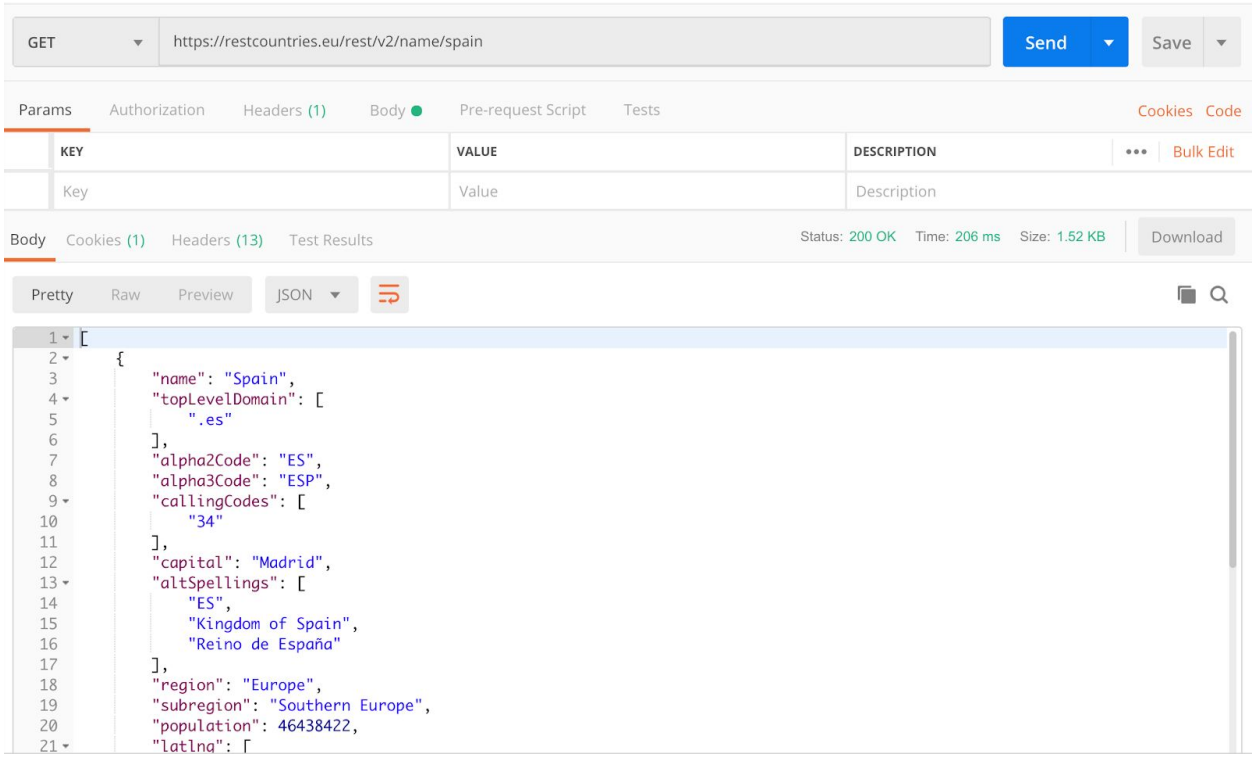


- **The navigation bar** simulates the browser's navigation bar
- **The `select` input** on the left indicates the HTTP verb you want to use in the request
- **The *Params* button** on the right is for specifying parameters in the request. If the HTTP verb used is *GET*, it will add the parameters in the URL's *query string*
- **The *SAVE* button** allows you to save the request, in case you want to repeat the same request. This is useful when you are introducing the same data repeatedly.
- **The *SEND* button** will create the request and send it.

Let's input the request to search by country name:



When the *SEND* button is clicked, Postman will display the information related to the response in the panel below.



Now, you can try to get the data from some other API endpoint. It's easy right? 💯

Axios calls

Axios GET call

A standard Axios call for any HTTP request is a function `axios()` that receives a JavaScript object that specifies different options of the request.

Since **Axios** is Promise based, we need to add our `.then()` to process the response, and the `.catch()` method in case there is an error on the response.

These are the most common options:

```
axios({
  method: "The HTTP method (verb) we are going to use",
  url: "The url the server is going to receive.",
  params: "URL parameters to be sent with the request" ,
})
.then(response => {
  // Here we can do something with the response object
})
```

```
.catch(err => {
  // Here we catch the error and display it
})
```

There are many others options we can configure in our **Axios** requests. You can check all of them in the Axios Documentation

- **`baseURL`**: will be prepended to `url`
- **`responseType`**: indicates the type of data that the server will respond. Options are `arraybuffer`, `blob`, `document`, `json`, `text`, `stream`.
- **`maxContentLength`**:defines the max size of the http response content allowed.

👀 **Check the official documentation to see the full list of available options.**

Making a request to a REST API

We will practice making `axios get` requests. To start, create folder `rest-countries` and inside create two files: `index.html` and `index.js`.

```
$ mkdir rest-countries
$ cd rest-countries
$ touch index.html index.js
```

Now add this code to `index.html`.

```
<!-- index.html -->

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Rest Countries API</title>
</head>
<body>
  <h1>Countries Info</h1>
<!--  use input's id to grab the value from the input field -->
  <input id="theInput" type="text">
  <button id="theButton">Get the country</button>
```

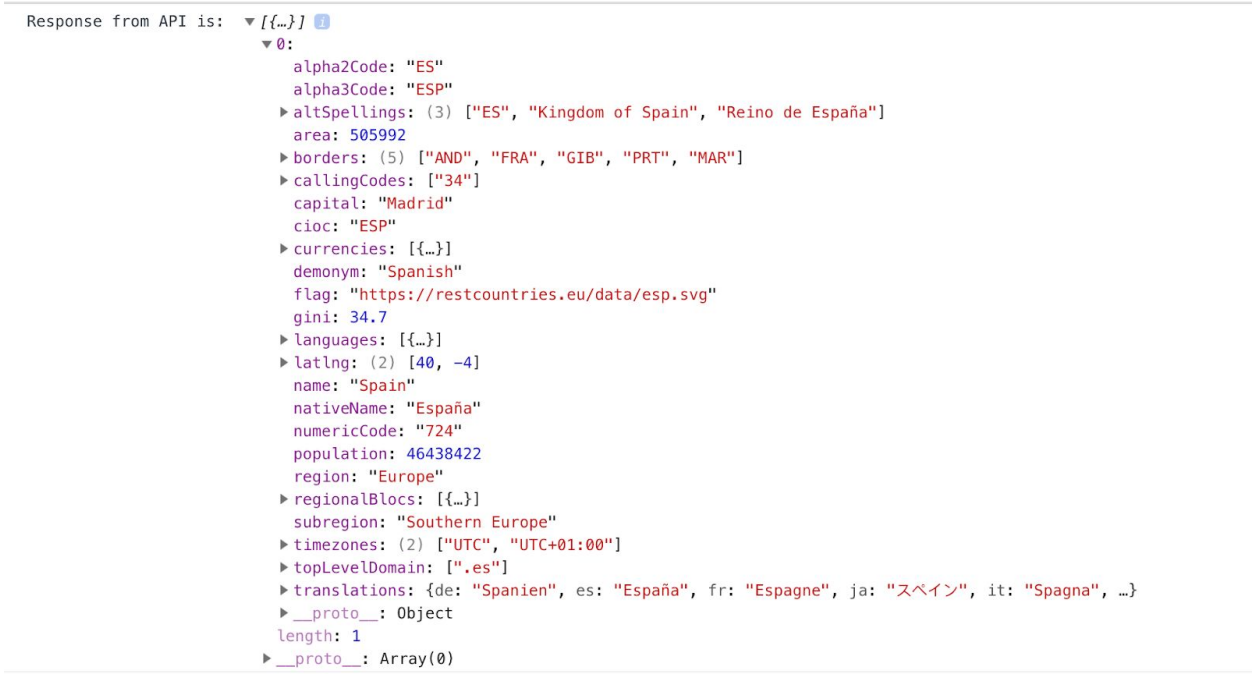
```
<!-- the data we get from the API will be populated in here:   -->
  <h2 id="countryName"></h2>
  <p id="countryCapital"></p>

  <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
  <script type="text/javascript" src="index.js"></script>
</body>
</html>
```

And now add some code to the `index.js`:

```
// index.js

const restCountriesApi = axios.create({
    baseURL: 'https://restcountries.eu/rest/v2/name/'
});

function getCountryInfo(theName) {
    restCountriesApi.get(theName)
    .then(responseFromAPI => {
        console.log('Response from API is: ', responseFromAPI.data);
})
.catch(err => {
    console.log('Error is: ', err);
    })
}

document.getElementById("theButton").onclick = function(){
    const country = document.getElementById("theInput").value;
    getCountryInfo(country);
}
```

Most of the times we use the syntax `axios.get(URL, body)` since it is more readable!

If you input **spain** and click the button, you will see this array of object(s) in the browser's console:



The key concept of Axios requests: **there is no page reload!** If you keep changing the countries. ypu'll get the data without refreshing the page.

Remember that normally, after every HTTP request, the page is refreshed with a full new page to render. If that were the case, the console would be empty.

Let's do extra step and display the data in the browser, not just in the console. Update the code in `index.js` with the following:

```
// index.js

let errDiv;

const restCountriesApi = axios.create({
    baseURL: 'https://restcountries.eu/rest/v2/name/'
});

function getCountryInfo(theName) {
    restCountriesApi.get(theName)
    .then(responseFromAPI => {
    removeErrDiv();
    const countryName = responseFromAPI.data[0].name;
    const countryCapital = responseFromAPI.data[0].capital;

// instead in the console, show data in the browser using JS DOM manipulation:
    document.getElementById("countryName").innerHTML = countryName;
```

```
    document.getElementById("countryCapital").innerHTML = "Capital: " +
countryCapital;
})
.catch(err => {
        if(err.response.status === 404){
            removeCountryInfo();
            createDiv();
            const theErr = document.createTextNode(`What the heck is
${theName}? 🤔`);
            errDiv.appendChild(theErr);
        } else {
            console.log('err => ', err)
        }
    })
}

function createDiv(){
    errDiv = document.createElement("div");
    errDiv.setAttribute("id", "error");
    document.body.appendChild(errDiv);
}

function removeErrDiv(){
    if(document.getElementById("error")){
        const error = document.getElementById("error");
        error.parentNode.removeChild(error);
    }
}

function removeCountryInfo(){
    document.getElementById("countryName").innerHTML = "";
    document.getElementById("countryCapital").innerHTML = "";
}

function checkInput() {
    removeErrDiv();
    if(document.getElementById("theInput").value === "") {
        document.getElementById('theButton').disabled = true;
        removeCountryInfo();
        createDiv();
        const theErr = document.createTextNode(`Wanna input something? 🤪`);
        errDiv.appendChild(theErr);
    } else {
        document.getElementById('theButton').disabled = false;
    }
}

document.getElementById("theButton").onclick = function(){
```

```
    removeErrDiv();
    const country = document.getElementById("theInput").value;
    getCountryInfo(country);
}
```

And there's one update in the `index.html`:

```
// index.html
...

<input id="theInput" type="text" onkeyup="checkInput()">

...
```

At this point, we won't go in details since these are the basics of DOM manipulation we covered in Module 1, but this could be very good recap exercise in case you started forgetting it 😉

Your turn

Try to get the information hitting the endpoint that returns response based on the `currency` that's officially being used.