We all remember the dreaded XMLHttpRequest we used back in the day to make requests, it involved some really messy code, it didn't give us promises and let's just be honest, it wasn't pretty JavaScript, right? Maybe if you were using jQuery, you used the cleaner syntax with `jQuery.ajax()` .

Well JavaScript has it's own built-in clean way now. Along comes the Fetch API a new standard to make server request jam-packed with promises and all those things we learned to love over the years.

How do we use the Fetch API?

In a very simple manner all you really do is call fetch with the URL you want, by default the Fetch API uses the GET method, so a very simple call would be like this:

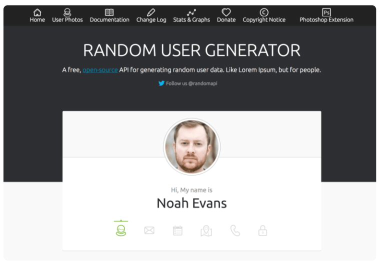
JAVASCRIPT

```
fetch(url) // Call th
.then(function() {
    // Your code for 
})
.catch(function() {
    // This is where 
});
```

Looks pretty simple right? So let's starting using it...

Using fetch to get data from an API

We are now going to build a simple GET request, in this case, I will use the Random User API (https://randomuser.me) and we will get 10 users and show them on the page using vanilla JavaScript.



Sara Vieira

FOLLOW @NIKKITAFTW(HTTPS://TWITTER.COM/IN

Sara Vieira is a Front-End Developer for Mindera in Portugal with a passion for everything front end related. She is also a drummer and a big time TV Show addict.

VIEW MY 5 POSTS

Let's get started with the HTML, all we really need is a heading and an unordered list:

HTML

```html
<h1>Authors</h1>
<ul id="authors"></u
```

The idea is to get all the data from the Random User API and display it in list items inside the author's list.

The first step is to actually set the URL we need and also the list we are gonna put the data in, so in the Javascript we write:

JAVASCRIPT

```javascript
const ul = document
const url = 'https:/
```

I have set these to consts so you don't risk changing these in the future and these two are meant to be constants through all the project. Now we get into actual Fetch API:

JAVASCRIPT

```
fetch(url)
  .then(function(data
    // Here you get t
    })
  })
  .catch(function(err
    // If there is any
  });
```

Let's review this code, shall we? So first we are calling the Fetch API and passing it the URL we defined as a constant above and since no more parameters are set this is a simple GET request. Then we get a response but the response we get is not JSON but an object with a series of methods we can use depending on what we want to do with the information, these methods include:

- » **clone()** - As the method implies this method creates a clone of the response.
- » **redirect()** - This method creates a new response but with a different URL.
- » **arrayBuffer()** - In here we return a promise that resolves with an ArrayBuffer.
- » **formData()** - Also returns a promise but one that resolves with FormData object.

- » **blob()** - This is one resolves with a Blob.
- » **text()** - In this case it resolves with a string.
- » **json()** - Lastly we have the method to that resolves the promise with JSON.

Looking at all these methods the one we want is the JSON one because we want to handle our data as a JSON object so we add:

JAVASCRIPT

```
fetch(url)
.then((resp) => resp
.then(function(data
  // Create and appe
  })
})
```

Now let's get to the part we create the list items, for that, I created two helper functions at the top of my file just to make the code simpler down the line:

JAVASCRIPT

```
function createNode
  return document.c
}

function append(pare
  return parent.appe
}
```

All these functions do is append and create elements as you can see. Once this is done we can move on to the resolution of our promise and add the code we need to append these list items to our unordered list:

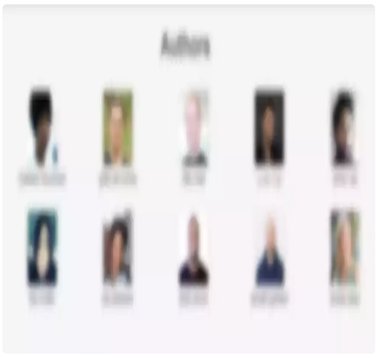
JAVASCRIPT

```
then(function(data) {
    let authors = dat
    return authors.ma
      let li = create
          img = creat
          span = crea
      img.src = autho
      span.innerHTML
      append(li, img)
      append(li, span
      append(ul, li);
  })
})
```

So first we define authors as the response we get from the request then we map over all the authors and for each we create a list item, a span, and an image. In the image source, we place the picture of the user, the HTML of the span will be the first and last name interpolated and then all we need to do is append this to their rightful parents and voilá, our HTTP request in vanilla JavaScript is done and returning something to the HTML.

(https://scotch.io/giveaways/scotch-launch)

CHECK IT OUT



To handle our catch all I will do is console log the error as we get it but you can do whatever you want with the error such as append it to the HTML with the functions we created. This is the full code of our little request:

JAVASCRIPT

```
function createNode
    return document
}

function append(pare
  return parent.appe
}

const ul = document
const url = 'https:/
fetch(url)
.then((resp) => resp
.then(function(data)
  let authors = data
  return authors.map
    let li = createN
        img = create
        span = creat
    img.src = author
    span.innerHTML =
    append(li, img)
    append(li, span
    append(ul, li);
  })
})
.catch(function(err
  console.log(error
});
```

Handling more requests like POST

So this is a GET request, the default one for the fetch function but of course we can do all other types of requests and also change the headers and off course send data, all we need for this is to set our object and pass it as the second argument of the fetch function:

JAVASCRIPT

```
const url = 'https://
// The data we are go
let data = {
    name: 'Sara'
}
// The parameters we a
let fetchData = {
    method: 'POST',
    body: data,
    headers: new Head
}
fetch(url, fetchData)
.then(function() {
    // Handle respons
});
```

You can also define cache, mode and all those things you are used to defining in your POST requests.

To create our object and use the fetch function we also have another option and that is to use the request constructor to create our request object, so instead of defining the object in the function itself we do this:

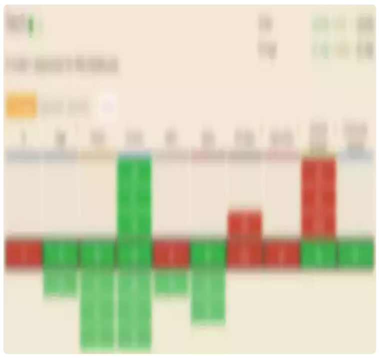
JAVASCRIPT

```
const url = 'https://
// The data we are go
let data = {
    name: 'Sara'
}
// Create our request
var request = new Req
    method: 'POST',
    body: data,
    headers: new Head
});

fetch(request)
.then(function() {
    // Handle respons
})
```

You can use the way you are most comfortable with to build your request object.

Final Thoughts



While the Fetch API is not yet supported by all the browsers (currently Safari does not support it) it is the beautiful replacement for XMLHttpRequest we desperately needed in our life. There are also polyfills if you really want to use it in more professional projects.