ARRAYS & FOREACH

JOFH@KEA.DK

AGENDA

- Introduction to API's
- JSON
- Exercise: Superhero info
- links in ressources
- AJAX
- The Product API
- Making our product site dynamic

INTRODUCTION TO API'S

APPLICATION PROGRAMMING INTERFACES

In computer programming, an application programming interface (API) is a set of subroutine definitions, communication protocols, and tools for building software. In general terms, it is a set of clearly defined methods of communication among various components. A good API makes it easier to develop a computer

IN PLAINER ENGLISH

- It's a service, someone built we can interact with
- Commonly, it's a database system
- Or some code others built, that we can use
- It's also what they call a lot of browser features

https://developer.mozilla.org/en-US/docs/Web/API

IN PROGRAMMING

- When you use a library someone else wrote (jQuery, React etc)
- They have "exposed" a set of functions/methods that you can use (interface with)
- That's the API

IN DATABASES

- The authors/owners of the DB will have exposed certain parts of the data
- Using various techniques (often `fetch`) you can query the database
- Most data-API's will allow you to READ data, few will allow you to POST data

EXAMPLE, JS LIBRARY

Let's look at what we cloned the other day

JSON

- JSON is a syntax for storing and exchanging data.
- JSON is text written with JavaScript Object Notation.
- Used to send data between different API's
- We can convert any JavaScript object into JSON, and send JSON to the server.
- We can also convert any JSON received from a

JSON SYNTAX RULES

- JSON syntax is derived from JavaScript object notation
  - Data is in name/value pairs
  - Data is separated by commas
  - Curly braces hold objects
  - Square brackets hold arrays
- A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value:

```
{ "name": "Jonas" }
```

- In JSON, keys must be strings, written with double quotes!

JSON SYNTAX RULES

- In JSON, values must be one of the following data types:
  - a string
  - a number
  - an object (JSON object)
  - an array
  - a boolean
  - null
- In JavaScript values can be all of the above, plus any other valid JavaScript expression, including:
  - a function

EXERCISE:

SUPERHERO INFO

TWO PARTS

1. First **we** create an object
2. Then you do it

So let's model a superhero

YOUR TURN

1. Create an object following our model
2. Save it as `my-superhero-name.json`
   (e.g. `lord-destroyer-van-der-doom.json`)
3. Upload it to fronter
   Plans > Activities > 2. Superhero Info, part 1
4. Then take a break

fetch

JOFH@KEA.DK

ONE SLIDE

TO RULE THEM ALL

This is the full, advanced version

```
fetch("SOME-API-ENDPOINT")
  .then(response => {
    if (!response.ok) {
      throw Error(response.statusText);
    }
    return response.json();
  })
  .then(data => {
    //We have the data
    console.log(data);
  })
  .catch(e => {
    //Woops, something went wrong
    console.error("An error occured:", e.message);
  });
```

We probably won't get there

AJAX

A SYNCHRONOUS JAVASCRIPT AND XML

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.

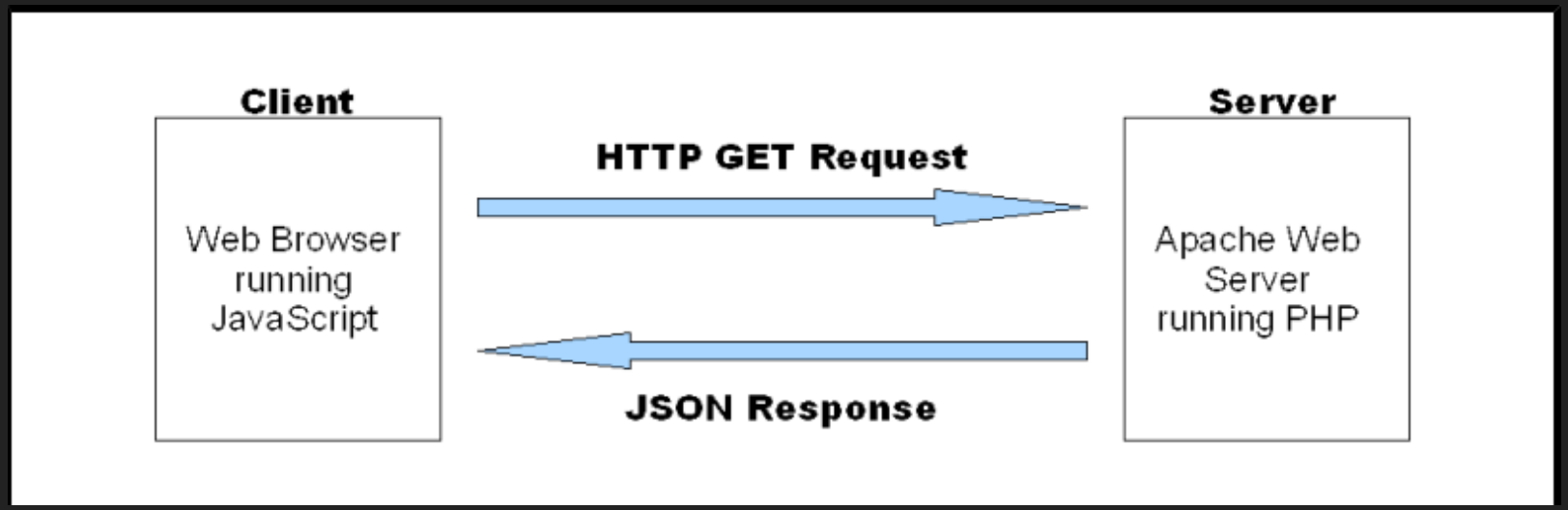
AJAX - THE DEVELOPER'S DREAM

because 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.

Think Facebook / Google docs / auto complete / any

So, how does clients & servers actually talk?



What is a

AJAX IN ACTION - fetch

`fetch`

`fetch` is one of the ways we can work with AJAX in JS

It is the easiest, if we can accept a bit of "black boxing"

The process could be worse

1. We send a request for something
2. We get a response (headers)
3. We extract the JSON from the response
4. We get the data

(if all goes well)

I PROMISE YOU SOMETHING

The hard part comes as the process is asynchronous

It can take time be fore we get the response

It can take time before we get the JSON out

Which gives us a new syntax to learn

SAMPLE

Which file do we load?

```
1  fetch("https://somesite.com/api/...")
2    .then(function (initialResponse) {
3      //could also be .text();
4      return initialResponse.json();
5    })
6    .then(function (data) {
7      buildIt(data);
8    });
9  function buildIt(data) {
10   //Ready to go
11   //clone the template add data from JSON
12 }
```

What kind of data do we need?

What to do once we get it

FETCH - ES6

```
1 fetch("https://somesite.com/api/...")
2   .then((e) => e.json())
3   .then(buildIt);
4
5 function buildIt(data) {
6   console.log(data);
7 }
```

(TOO) MANY WAYS TO USE A CALLBACK

Let's start simple

```
1 const button = document.querySelector("button");
2
3 button.addEventListener("click", buttonClicked);
4
5 function buttonClicked() {
6   console.log("hi");
7 }
```

We can do it without creating a variable for the button

```
1 document.querySelector("button").addEventListener("click", butt
2
3 function buttonClicked() {
4   console.log("hi");
5 }
```

We can define the function directly in the callback

```
1 const button = document.querySelector("button");
2
3 button.addEventListener("click", function buttonClicked() {
4   console.log("hi");
5 });
```

When used like that, we don't need the name

```
1 const button = document.querySelector("button");
2
3 button.addEventListener("click", function () {
4   console.log("hi");
5 });
```

And since "ES6" we can use arrow functions

```
1 const button = document.querySelector("button");
2
3 button.addEventListener("click", () => {
4   console.log("hi");
5 });
```

MAKING

OUR SITE DYNAMIC

This is our API

https://kea-alt-del.dk/t7/api/

This is my repo

https://github.com/jonasholbech/static-site-v2

EXERCISE

Fronter: "3. The product page, v1"

arrays & forEach

An array is a special variable, which can hold more than one value at a time.

DECLARING ARRAYS

```
1 const animals = ["Monkey", "Chicken"];
2
3 const placesVisitedIn2020 = [];
4 const placesVisitedIn2021 = [];
5
6 const kmToWorkThisWeek = [0, 11, 0, 0, 11];
```

DECLARING ARRAYS, ADVANCED

```
const kids = [
  {
    name: "Storm",
    age: 12,
  },
  {
    name: "Birk",
    age: 10,
  },
  {
    name: "Liv",
    age: 5,
  },
];
```

USING ARRAYS

We access arrays using a "zero based index"

```
1 const animals = ["Monkey", "Chicken"];
2 const firstAnimal = animals[0];
3 const secondAnimal = animals[1];
```

USING ARRAYS

or by "looping" through them one at a time

```
1  const ninjas = ["Kai", "Jay", "Lloyd", "Cole", "Zane"];
2
3  //For each ninja, call the function doKarate
4  ninjas.forEach(doKarate);
5
6  /* The ninjas are passed to the function as
7  'oneNinja' one at a time */
8  function doKarate(oneNinja) {
9    console.log(`${oneNinja} does some karate`);
10 }
```

developer.mozilla.org/.../Array/forEach

Other syntaxes exists

What if it was an array of objects?

Easy, peasy

```
const ninjas = [
  {
    name: "Cole",
    power: "earth"
  },
  {
    name: "Zane",
    power: "ice"
  }
];

//For each ninja, call the function doYourThing
ninjas.forEach(doYourThing);

/* The ninjas are passed to the function as
'nin' one at a time */
function doYourThing(nin) {
  console.log(`${nin.name} has a special power: ${nin.power}`)
}
```

ARRAYS, OBJECTS & THE <template>

We have all the pieces

1. Create a template
2. Loop through the data
3. `forEach` item in the data:
    1. Grab the template
    2. clone the template
    3. change some content
    4. grap the parent

```
const items = [
  {
    name: "Jonas",
    pets: 6,
  },
  {
    name: "Lasse",
    pets: 0,
  },
];

items.forEach(showProduct);

function showProduct(item) {
  const template = document.querySelector("template").content;
  const clone = template.cloneNode(true);
  clone.querySelector(".name").textContent = item.name;
  clone.querySelector(".pets").textContent = item.pets;
  const parent = document.querySelector("main");
  parent.appendChild(clone);
}
```

EXERCISE

Fronter: "4. Superhero list, #2"

BONUS EXERCISE

Aren't you lucky

- If you forked before you cloned you can now `add => commit => push` directly
- DO IT
- Since it's now YOUR repository, we can turn on GitHub Pages
- Follow the instructions in this link