APIs & JSON

Application Programming Interface

JavaScript Object Notation

MDN wants to learn about developers like you: https://qsurvey.mozilla.com/s3/MDN-dev-survey

‹ Web technology for developers

Web APIs

When writing code for the Web with JavaScript, there are a great many APIs available. Below is a list of all the interfaces (that is, types of objects) that you may be able to use while developing your Web app or site.

A

ANGLE_
Abstra
Ambien
AmbientLightSensorReading
AnalyserNode
Animation
AnimationEffectReadOnly
Animat
Animat
Animat
Animat
Animat
AnimationTimeline
ArrayBufferView
Attr
AudioBuffer
AudioBufferSourceNode
AudioContext
AudioDestinationNode
AudioListener
AudioNode
AudioParam

GamepadButton
GamepadEvent
GamepadHapticActuator

H

HMDVRDevice
HTMLAnchorElement
HTMLAreaElement
HTMLAudioElement
HTMLBRElement
HTMLBaseElement
HTMLBaseFontElement
HTMLBodyElement
HTMLButtonElement

MimeTypeArray
MouseEvent
MouseScrollEvent
MouseWheelEvent
MutationEvent
MutationObserver
MutationRecord

NavigatorGeolocation
NavigatorID
NavigatorLanguage
NavigatorOnLine
NavigatorPlugins
NavigatorStorage
NetworkInformation
Node
NodeFilter
NodeIterator

SVGGeometryElement
SVGGlyphElement
SVGGlyphRefElement
SVGGradientElement
SVGGraphicsElement
SVGHKernElement
SVGImageElement
SVGLength
SVGLengthList
SVGLineElement
SVGLinearGradientElement
SVGMPathElement
SVGMaskElement
SVGMatrix
SVGMeshElement
SVGMetadataElement
SVGMissingGlyphElement
SVGNumber
SVGNumberList
SVGPathElement
SVGPatternElement
SVGPoint
SVGPolygonElement

API

- An API (**Application Programming Interface**) is a set of subroutine definitions, protocols, and tools for building application software.
- An **API** may be for a web-based system, operating system, database system, computer hardware or software library.
- A **web API** is an API for either a web server or a web browser.
- A **server-side** web API is a programmatic interface consisting of one or more publicly exposed **endpoints** to a defined request–response message system, typically expressed in **JSON** or **XML**, which is **exposed via the web**.

- Restaurant Petrograd's API: http://kea-alt-del.dk/t5/api/

JSON

- JSON is a syntax for storing and exchanging data.
- JSON is text written with **J**ava**S**cript **O**bject **N**otation.
- When exchanging data between a **browser** and a **server**, the data can only be text.
- We can convert any JavaScript object into JSON, and send JSON to the server.
- We can also convert any JSON received from a server into **JavaScript objects**.

https://www.w3schools.com/js/js_json_intro.asp

JSON Syntax Rules

- JSON syntax is derived from JavaScript object notation syntax:
  - 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!
  - In JavaScript, keys can be strings, numbers, or identifier names:
    { name: ”Jonas” }

The main difference between JSON and JS objects = quotes / no quotes in key names.

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
  - a date
  - undefined

AJAX, Fetch & promise

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.**

https://www.w3schools.com/xml/ajax_intro.asp

What is AJAX?

- AJAX stands for **A**synchronous **J**avaScript **A**nd **X**ML.
- It can send and receive information in various formats, including JSON, XML, HTML, and text files.
- AJAX's most appealing characteristic is its "asynchronous" nature, which means it can communicate with the server, exchange data, and update the page without having to refresh the page.

https://developer.mozilla.org/en-US/docs/AJAX/Getting_Started

What is AJAX?

- AJAX is **not** a programming language.
- AJAX just uses a combination of:
  - A browser built-in **XMLHttpRequest** object to request data from a web server.
  - **JavaScript** and HTML **DOM manipulation** to display or use the received data.
- All modern browsers support the XMLHttpRequest object.

https://www.w3schools.com/xml/ajax_intro.asp

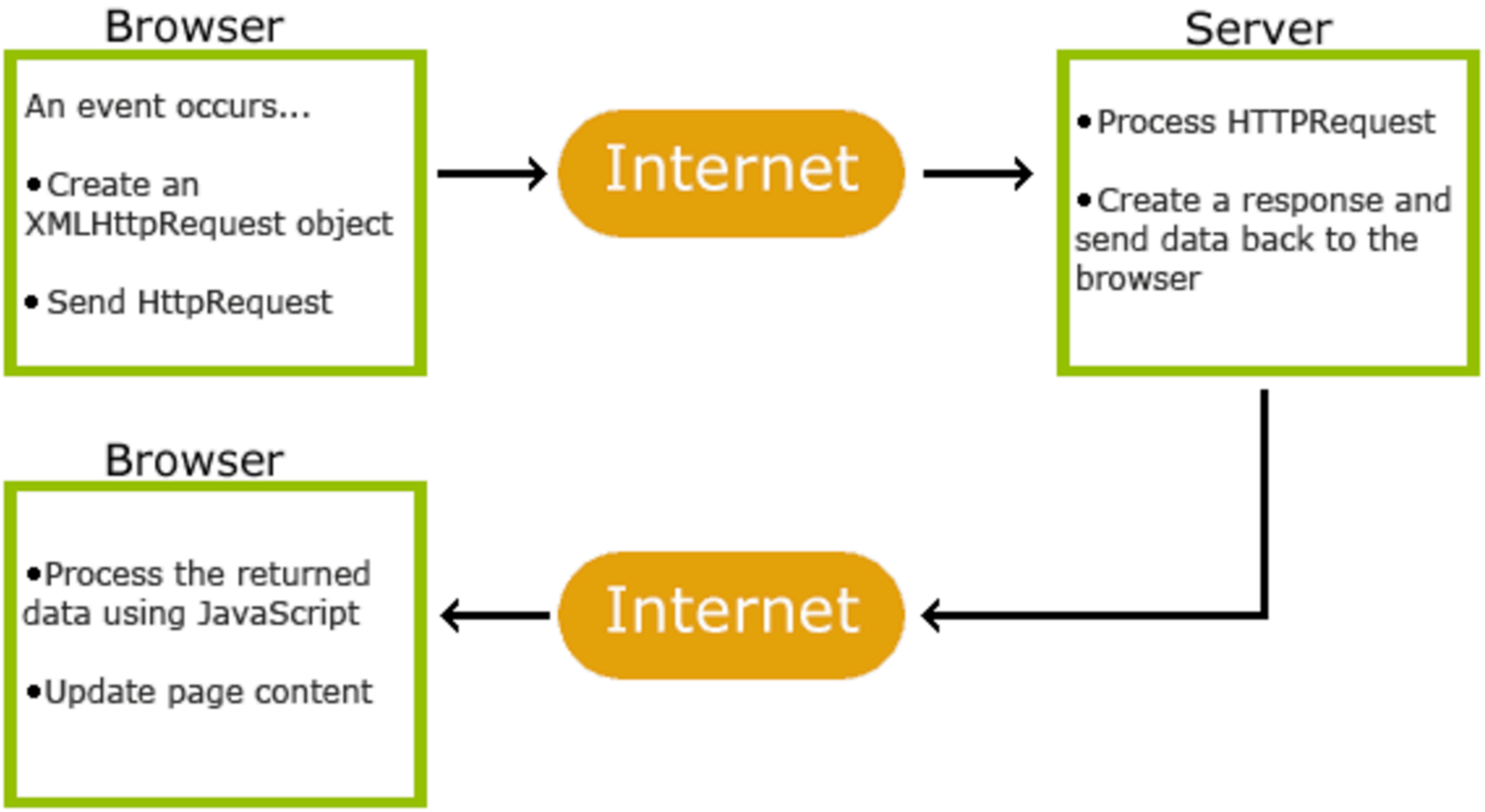
AJAX is a misleading name!

AJAX applications might use XML to transport data, but it is equally common to transport data as plain text or **JSON** (thus also known as **AJAJ**)

How AJAX Works

1. An event occurs in a web page (the page is loaded, a button is clicked, typing into an input field…)
2. An XMLHttpRequest object is created by JavaScript.
3. The XMLHttpRequest object sends a request to a web server.
4. The server processes the request.
5. The server sends a response back to the web page.
6. The response is read by JavaScript.
7. Proper action is performed by JavaScript (without reloading the page)

How AJAX Works


Browser
An event occurs...
•Create an XMLHttpRequest object
•Send HttpRequest
Internet
Server
•Process HTTPRequest
•Create a response and send data back to the browser
Internet
Browser
•Process the returned data using JavaScript
•Update page content

AJAX(J) in action: fetch!

```
let myLink = "http://kea-alt-del.dk/t5/api/productlist";

function loadData(link){
    fetch(link).then(e=>e.json()).then(data=>show(data));
}
```

Fetch returns a *promise*. When the data is loaded it is interpreted as JSON (another *promise*). Then we can finally do stuff with it (like putting snippets of data into our HTML template at the right places)

Fetch

- The Fetch **API** provides an interface for fetching resources (also across the network)
- **Fetch** is a modern concept equivalent to XMLHttpRequest, but is designed to be more extensible and efficient.
- The **fetch()** method takes one mandatory argument, the **path** to the resource you want to fetch, e.g. JSON
- It returns a **promise** that resolves to the Response to that request, whether it is successful or not.

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

Promise

- A **Promise** is an object representing the eventual completion or failure of an asynchronous operation
  - called an **asynchronous function call**.
- **Promise** allows two or more asynchronous operations to execute back to back, where each subsequent operation **starts when the previous operation succeeds, with the result from the previous step**.
- Accomplished by creating a **promise chain**.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Using_promises

Promise in Kyle Simpson's words

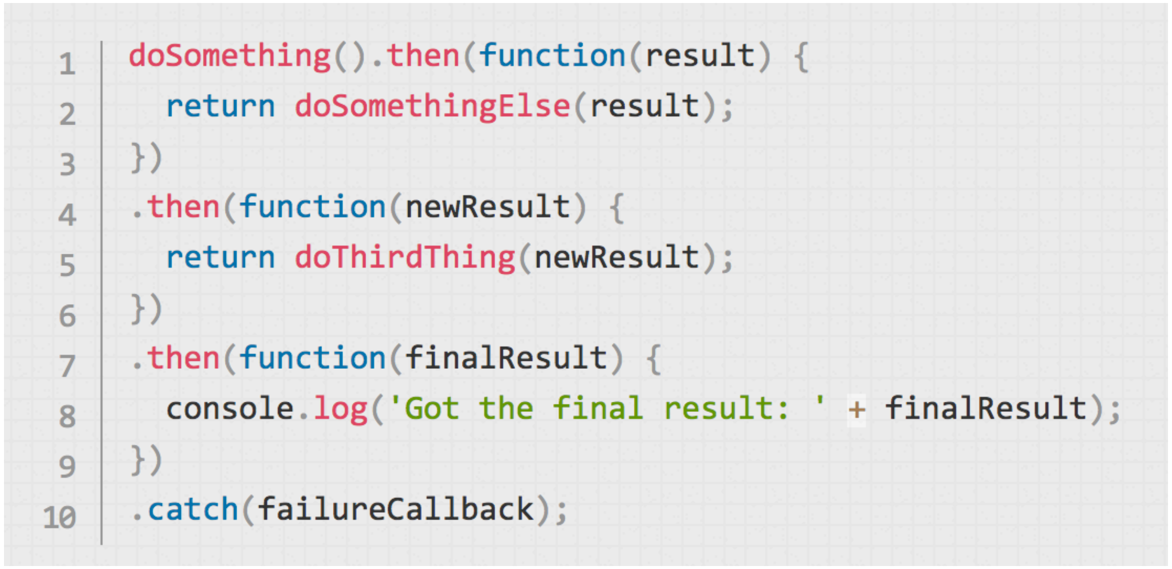
A Promise is a way to **reason about data that doesn't yet exist**, but you know it will. **It's like ordering food at a fast-food restaurant**:

1. Order your food.
2. Pay for your food and receive a ticket with an order number.
3. Wait for your food.
4. When your food is ready, they call your ticket number.
5. Receive the food.

- You may not be able to eat your food while you're waiting for it, but you can think about it, and you can prepare for it.
- You can proceed with your day knowing that food is going to come, even if you don't have it yet, because the food has been "promised" to you.
- That's all a Promise is: **An object that represents data that will eventually exist**.

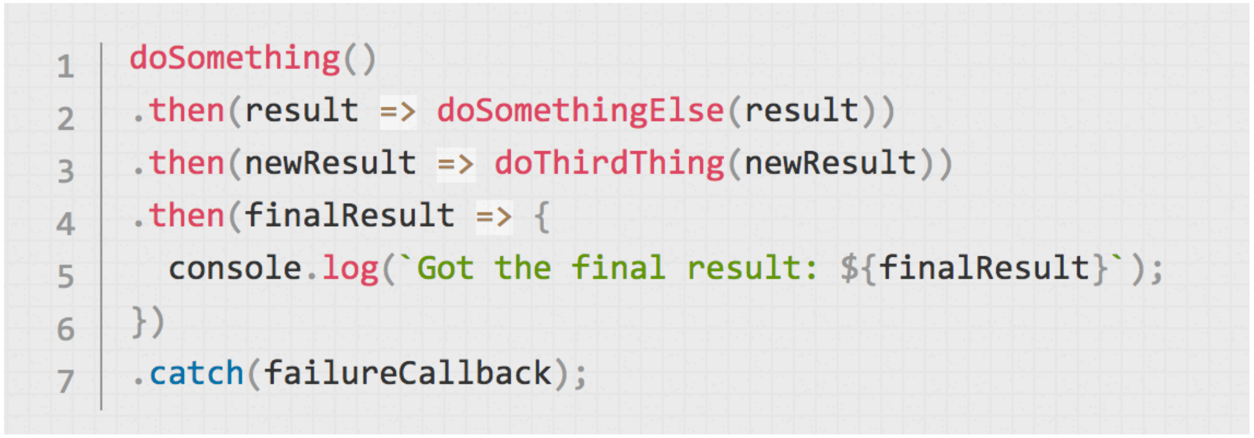
https://css-tricks.com/using-es2017-async-functions/

Promise chain



```
doSomething().then(function(result) {
  return doSomethingElse(result);
})
.then(function(newResult) {
  return doThirdThing(newResult);
})
.then(function(finalResult) {
  console.log('Got the final result: ' + finalResult);
})
.catch(failureCallback);
```

Same chain written with *arrow* syntax



```
doSomething()
.then(result => doSomethingElse(result))
.then(newResult => doThirdThing(newResult))
.then(finalResult => {
  console.log(`Got the final result: ${finalResult}`);
})
.catch(failureCallback);
```

Fetch in praxis (generic example)

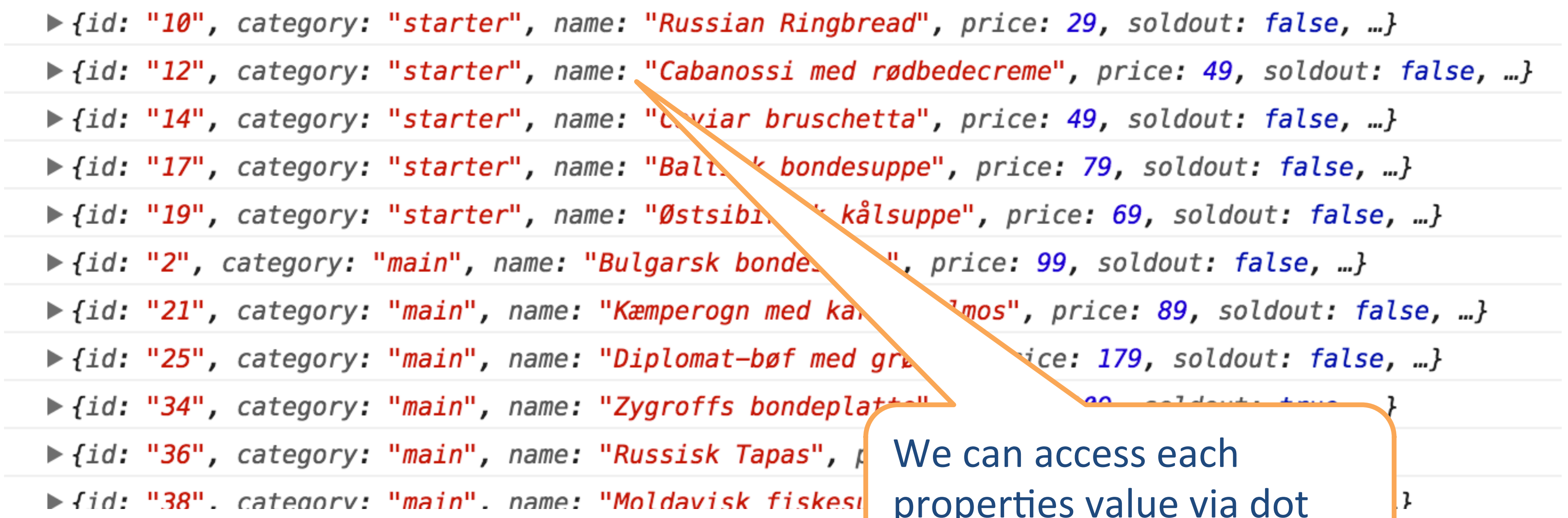
```
let myLink = "http://kea-alt-del.dk/t5/api/productlist";

function loadData(link){
    fetch(link).then(e=>e.json()).then(data=>show(data));
}

function show(data){
    data.forEach(object => console.log(object.property));
}

loadData(myLink);
```

Each object in the json array has one or more properties, e.g. “name”, “price” etc. (equal to the column names in the database (spreadsheet))

Fetch in praxis (Petrograd example)

```
function show(data){
    data.forEach(object => console.log(object));
}
```

The functions will output:



```
▶{id: "10", category: "starter", name: "Russian Ringbread", price: 29, soldout: false, …}
▶{id: "12", category: "starter", name: "Cabanossi med rødbedecreme", price: 49, soldout: false, …}
▶{id: "14", category: "starter", name: "C viar bruschetta", price: 49, soldout: false, …}
▶{id: "17", category: "starter", name: "Balt k bondesuppe", price: 79, soldout: false, …}
▶{id: "19", category: "starter", name: "Østsibi k kålsuppe", price: 69, soldout: false, …}
▶{id: "2", category: "main", name: "Bulgarsk bonde ", price: 99, soldout: false, …}
▶{id: "21", category: "main", name: "Kæmperogn med ka mos", price: 89, soldout: false, …}
▶{id: "25", category: "main", name: "Diplomat-bøf med gr ice: 179, soldout: false, …}
▶{id: "34", category: "main", name: "Zygroffs bondeplat }
▶{id: "36", category: "main", name: "Russisk Tapas", p
▶{id: "38", category: "main", name: "Moldavisk fiskesu }
```

We can access each properties value via dot notation: e.g. object.name

Petrograd product <template>

```
<template class="product">
    <div class="list-product">
        <a href="data-big-image">
            <img class="product-small-img" src="data-small-img" alt="data-caption">
        </a>
        <h3 class="name">data-name</h3>
        <p class="category">data-category</p>
        <h4 class="price"><span>data-price</span> kr.</h4>
        <button>details</button>
    </div>
</template>
```

The **text content** will be replaced with the text that is fetched from the database (spreadsheet)

Show all dishes script (.then syntax)

```
let productlist_link = "http://kea-alt-del.dk/t5/api/productlist";
let image_path = "http://kea-alt-del.dk/t5/site/imgs/small/";
let main = document.querySelector('main');
let template = document.querySelector('.product');

function loadData(link){
    fetch(link).then(e=>e.json()).then(data=>show(data));
}

function show(data){
    data.forEach(element => {
        let clone = template.cloneNode(true).content;
        clone.querySelector('.product-small-img').src = image_path + element.image + "-sm.jpg";
        clone.querySelector('.name').textContent = element.name;
        clone.querySelector('.category').textContent = element.category;
        clone.querySelector('.price span').textContent = element.price;
        main.appendChild(clone);
    });
}

loadData(productlist_link);
```

The fetched data is “injected” to the clone in the relevant places

Each new clone is appended to <main> when the data is in place

Resources

- https://en.wikipedia.org/wiki/Application_programming_interface#Web_APIs
- https://www.w3schools.com/js/js_json_syntax.asp
- https://developer.mozilla.org/en-US/docs/AJAX
- https://www.w3schools.com/xml/ajax_intro.asp
- https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Using_promises