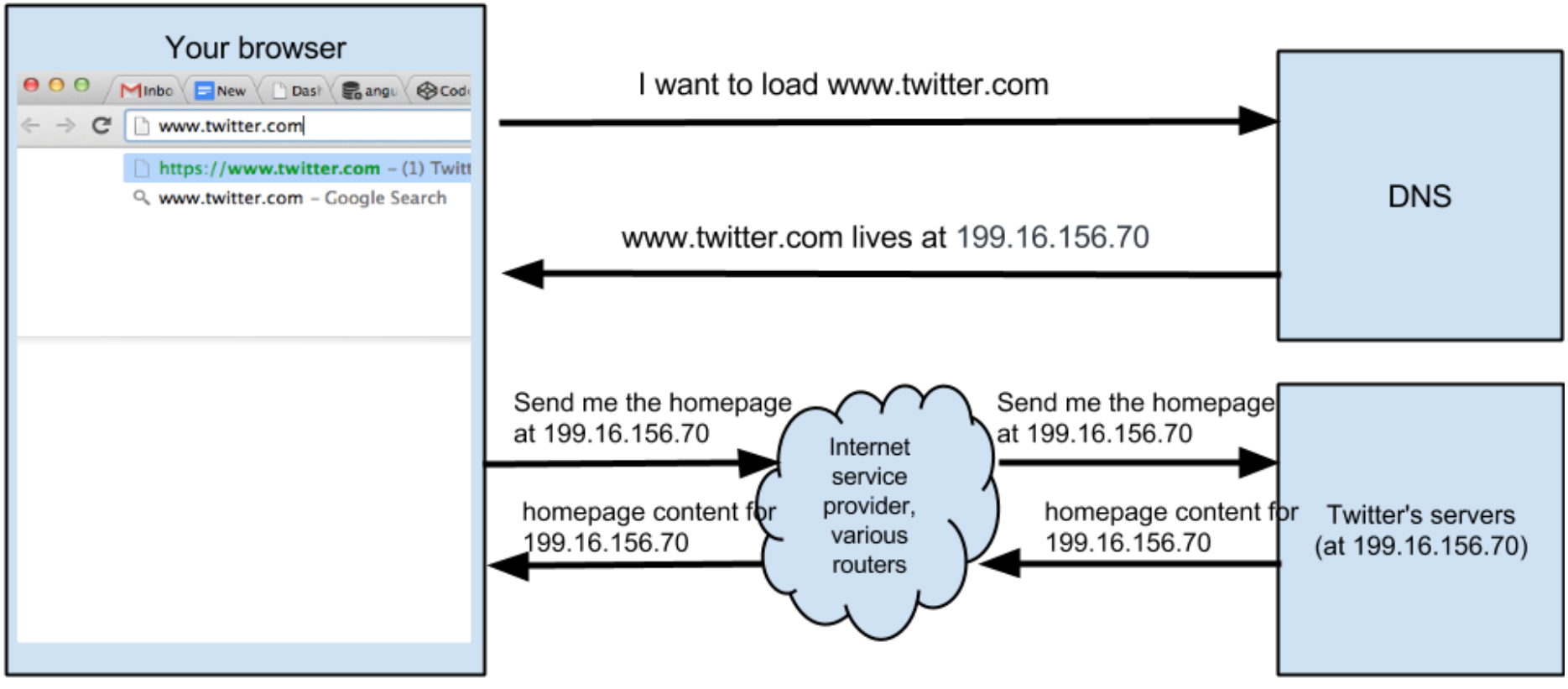
{POWER.CODERS}

AJAX and JSON

Agenda

Today we will look into

- Recap: HTTP / HTTPS
- JSON
- AJAX
- Promises



Communication between computers

- Computers communicate with each other with **HTTP/HTTPS**.

Communication between computers

- Computers communicate with each other with **HTTP/HTTPS**.
- Computers can be **clients** or **servers**.

Communication between computers

- Computers communicate with each other with **HTTP/HTTPS**.
- Computers can be **clients** or **servers**.
- The client sends a **request** to the server.

Communication between computers

- Computers communicate with each other with **HTTP/HTTPS**.
- Computers can be **clients** or **servers**.
- The client sends a **request** to the server.
- The server sends a **response** back to the client.

What is HTTP?

What is HTTP?

- **h**yper**t**ext **t**ransfer **p**rotocol

What is HTTP?

- **h**yper**t**ext **t**ransfer **p**rotocol
- Protocol used for websites to transfer HTML, CSS, JS, images and text

HTTP REQUESTS

HTTP REQUESTS

- **GET**: I receive the Twitter feed with all tweets from today

HTTP REQUESTS

- **GET**: I receive the Twitter feed with all tweets from today
- **POST**: I create a new user which is added to the server

HTTP REQUESTS

- **GET**: I receive the Twitter feed with all tweets from today
- **POST**: I create a new user which is added to the server
- **PUT**: I edit a tweet I made before

HTTP REQUESTS

- **GET**: I receive the Twitter feed with all tweets from today
- **POST**: I create a new user which is added to the server
- **PUT**: I edit a tweet I made before
- **DELETE**: I delete a tweet or my user account

HTTP RESPONSES

HTTP RESPONSES

- **Status Code**, e.g. 200 (OK), 404 (Not found), 500 (Server error)

HTTP RESPONSES

- **Status Code**, e.g. 200 (OK), 404 (Not found), 500 (Server error)
- **Data**, e.g. HTML, CSS, JS, images and more

HTTP RESPONSES

- **Status Code**, e.g. 200 (OK), 404 (Not found), 500 (Server error)
- **Data**, e.g. HTML, CSS, JS, images and more

Have a look at the network tab in your code inspector.

HTTPS

HTTPS

- **h**yper**t**ext **t**ransfer **p**rotocol **s**ecure

HTTPS

- **h**yper**t**ext **t**ransfer **p**rotocol **s**ecure
- The data is encrypted between client and server using a secret key.

HTTPS

- **h**yper**t**ext **t**ransfer **p**rotocol **s**ecure
- The data is encrypted between client and server using a secret key.
- The technology used is today the transport layer security (TLS) and before was the secure sockets layer (SSL).

Data formats

Standard formats to send data over the internet and receive it

- Plain text
- HTML

Data formats

Standard formats to send data over the internet and receive it

- Plain text
- HTML
- XML

Data formats

Standard formats to send data over the internet and receive it

- Plain text
- HTML
- XML
- JSON

XML

e**X**tensible **M**arkup **L**anguage

XML

e**X**tensible **M**arkup **L**anguage

`XML` uses tags to define its structure similar to `HTML` and `SVG`.

XML

e**X**tensible **M**arkup **L**anguage

`XML` uses tags to define its structure similar to `HTML` and `SVG`.

Tags are not predefined, you can specify your own tags.

XML

e**X**tensible **M**arkup **L**anguage

`XML` uses tags to define its structure similar to `HTML` and `SVG`.

Tags are not predefined, you can specify your own tags.

Nested tags are possible.

XML EXAMPLE

```
<person>
  <name>Susanne</name>
  <surname>Koenig</surname>
  <nationality>German</nationality>
  <languages>
    <language>German</language>
    <language>English</language>
  </languages>
</person>
```

JSON

Java**S**cript **O**bject **N**otation

JSON

Java**S**cript **O**bject **N**otation

`JSON` is a lighweight, readable format for structuring data.

JSON

Java**S**cript **O**bject **N**otation

`JSON` is a lighweight, readable format for structuring data.

It is a modern alternative to **XML** to transmit data from server to client.

JSON

Java**S**cript **O**bject **N**otation

`JSON` is a lighweight, readable format for structuring data.

It is a modern alternative to **XML** to transmit data from server to client.

JSON is a string whose format very much resembles JavaScript objects.

JSON EXAMPLE

```
{
  "name"        : "Susanne",
  "surname"     : "Koenig",
  "nationality" : "German",
  "languages"   : ["German", "English"]
}
```

JSON EXAMPLE

```
{
  "name"        : "Susanne",
  "surname"     : "Koenig",
  "nationality" : "German",
  "languages"   : ["German", "English"]
}
```

Like a JS object JSON maps **keys** to **values**.

JSON EXAMPLE

```
{
  "name"        : "Susanne",
  "surname"     : "Koenig",
  "nationality" : "German",
  "languages"   : ["German", "English"]
}
```

Like a JS object JSON maps **keys** to **values**.

Always use **double quotes** `""` in JSON.

Why JSON?

- JSON is easier to parse and use with JS
- JSON saves bandwidth
- JSON improves the response time
- JSON is the standard today

JSON in JavaScript

```
const obj = JSON.parse('{"name": "Susanne","surname": "Koenig"}');

let myJSON = JSON.stringify(obj);
```

JSON in JavaScript

```
const obj = JSON.parse('{"name": "Susanne","surname": "Koenig"}');

let myJSON = JSON.stringify(obj);
```

`JSON.parse()` is a built-in JavaScript method to turn **JSON** into a **JavaScript object**.

JSON in JavaScript

```
const obj = JSON.parse('{"name": "Susanne","surname": "Koenig"}');

let myJSON = JSON.stringify(obj);
```

`JSON.parse()` is a built-in JavaScript method to turn **JSON** into a **JavaScript object**.

`JSON.stringify()` is a built-in JavaScript method to turn a **JavaScript object** into a **JSON**.

AJAX

AJAX

HTTP can also fetch only one part of documents to update web pages on demand. Without reloading.

AJAX

HTTP can also fetch only one part of documents to update web pages on demand. Without reloading.

Google came up with that in 2006.

AJAX

Asynchronous **J**avacript **a**nd **X**ML

AJAX

Asynchronous **J**avacript **a**nd **X**ML

It combines a group of existing technologies: HTML, CSS, JavaScript, XML, JSON, etc.

AJAX

Asynchronous **J**avacript **a**nd **X**ML

It combines a group of existing technologies: HTML, CSS, JavaScript, XML, JSON, etc.

Together this group can build modern applications.

JavaScript

- initiates the AJAX request
- parses the AJAX response
- updates the DOM

The old way

`XMLHttpRequest`

Also known as **XHR API** is used to make a request to a server.

`XMLHttpRequest`

Also known as **XHR API** is used to make a request to a server.

API: Application Programming Interface is a set of methods which specify the rules of communication between two interested parties.

`XMLHttpRequest`

Also known as **XHR API** is used to make a request to a server.

API: Application Programming Interface is a set of methods which specify the rules of communication between two interested parties.

Note: The incoming data does not need to be in `XML`.

XHR EXAMPLE

```
var request = new XMLHttpRequest();

request.open('GET', '/path/to/api', true);
request.setRequestHeader('Content-type', 'application/json'); // Not for text + HTML

request.onload = function() {
  if (request.status >= 200 && request.status < 400) {
    console.log(JSON.parse(request.responseText));
  } else {
    // We reached our target server, but it returned an error
  }
});

request.onerror = function() {
  // There was a connection error of some sort
};
```

The new way

FETCH API

```
fetch("https://jsonplaceholder.typicode.com/todos/1")
  .then(response => response.json())
  .then(data => console.log(data));
```

fetch

`fetch`

1. returns a **promise**: I promise to let you know when the response of my request is returned

`fetch`

1. returns a **promise**: I promise to let you know when the response of my request is returned
2. then it returns a **response**: with its own method `json()` to parse the response

`fetch`

1. returns a **promise**: I promise to let you know when the response of my request is returned
2. then it returns a **response**: with its own method `json()` to parse the response
3. then it returns a **object**: with the data of your AJAX call

Promises

new in ES6

A **promise** is an **object** that may produce a single value some time **in the future**. Either a **resolve** value, or a reason why it's not resolved (**rejected**).

CALLBACKS

```
button.addEventListener("click", submitForm);
```

Once the button is clicked, the function submitForm will be called.

Nested callbacks

```
movePlayer(100, "left", function(){
  movePlayer(200, "right", function(){
    movePlayer(400, "left", function(){

    });
  });
});
```

Once the player moved 100 steps to the left and that is done, the player should move 200 steps to the reight and then this move is done, the player moves 400 steps to the left again.

Nested callbacks

```
movePlayer(100, "left", function(){
  movePlayer(200, "right", function(){
    movePlayer(400, "left", function(){

    });
  });
});
```

Once the player moved 100 steps to the left and that is done, the player should move 200 steps to the reight and then this move is done, the player moves 400 steps to the left again.

= pyramid of doom (repetition of code, difficult to read)

MOVEPLAYER WITH PROMISES

```
movePlayer(100, "left")
  .then(() => movePlayer(200, "right"))
  .then(() => movePlayer(400, "left"));
```

MOVEPLAYER WITH PROMISES

```
movePlayer(100, "left")
  .then(() => movePlayer(200, "right"))
  .then(() => movePlayer(400, "left"));
```

Better, but still hard to read

Different syntax

```
const promise = new Promise((resolve, reject) => {
  if(true){
    resolve('Stuff worked');
  } else {
    reject('Error, it broke');
  }
})

promise.then(result => console.log(result))
```

Error handling

```
promise
  .then(result => result + "!")
  .then(result2 => result2 + "?")
  .then(result3 => {
    throw Error;
    console.log(result3)
  })
  .catch(() => console.log("error!"));
```

Error handling

```
promise
  .then(result => result + "!")
  .then(result2 => result2 + "?")
  .then(result3 => {
    throw Error;
    console.log(result3)
  })
  .catch(() => console.log("error!"));
```

Promises are used for **asynchronous** JavaScript.

Error handling

```
promise
  .then(result => result + "!")
  .then(result2 => result2 + "?")
  .then(result3 => {
    throw Error;
    console.log(result3)
  })
  .catch(() => console.log("error!"));
```

Promises are used for **asynchronous** JavaScript.

The dot syntax here is called **chaining**

Asynchronous JavaScript?

To understand **asynchronous** JavaScript, we need to understand **synchronous** JavaScript first.

Asynchronous JavaScript?

To understand **asynchronous** JavaScript, we need to understand **synchronous** JavaScript first.

Until now all our JS was synchronous: you run some code and the result will bet returned as soon as the browser can do it.

Synchronous JavaScript

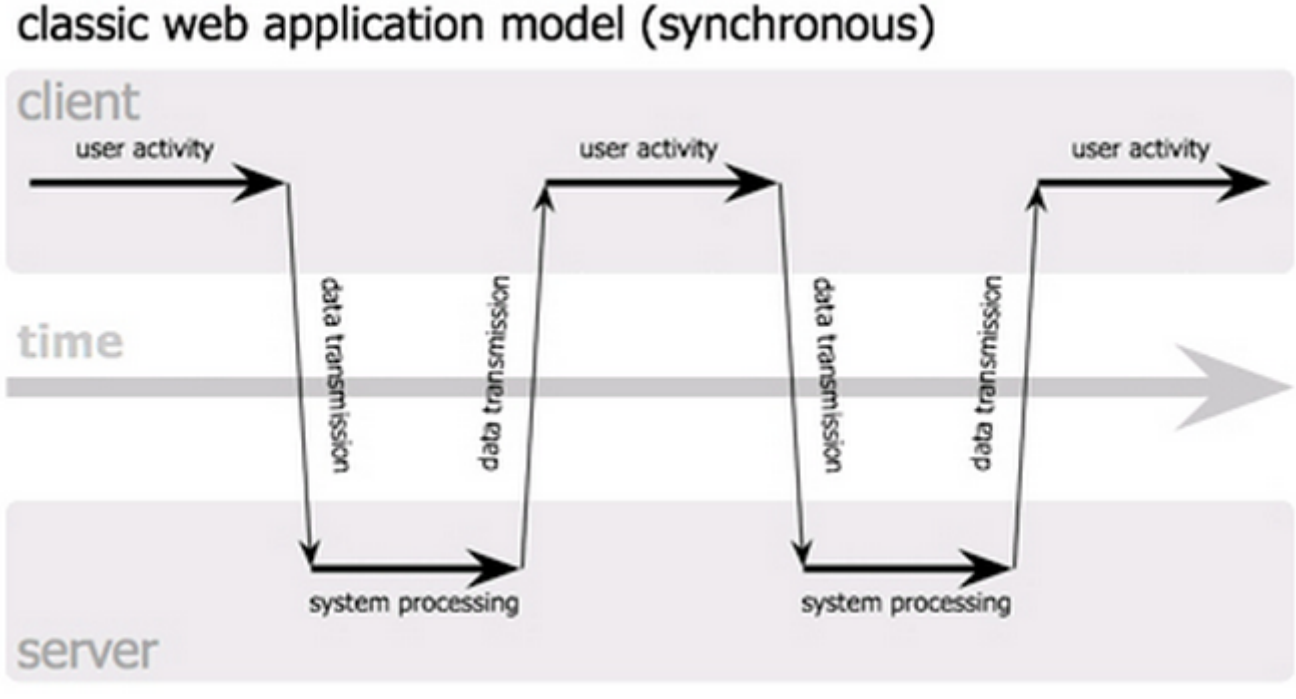
While an operation is processed (calling a function for example), nothing else can happen - remember `alert`?

Synchronous JavaScript

While an operation is processed (calling a function for example), nothing else can happen - remember `alert`?

Reason for that is that JS is **single-threaded**. All tasks run on the main thread like pearls on a necklace.

Synchronous workflow



Asynchronous JavaScript

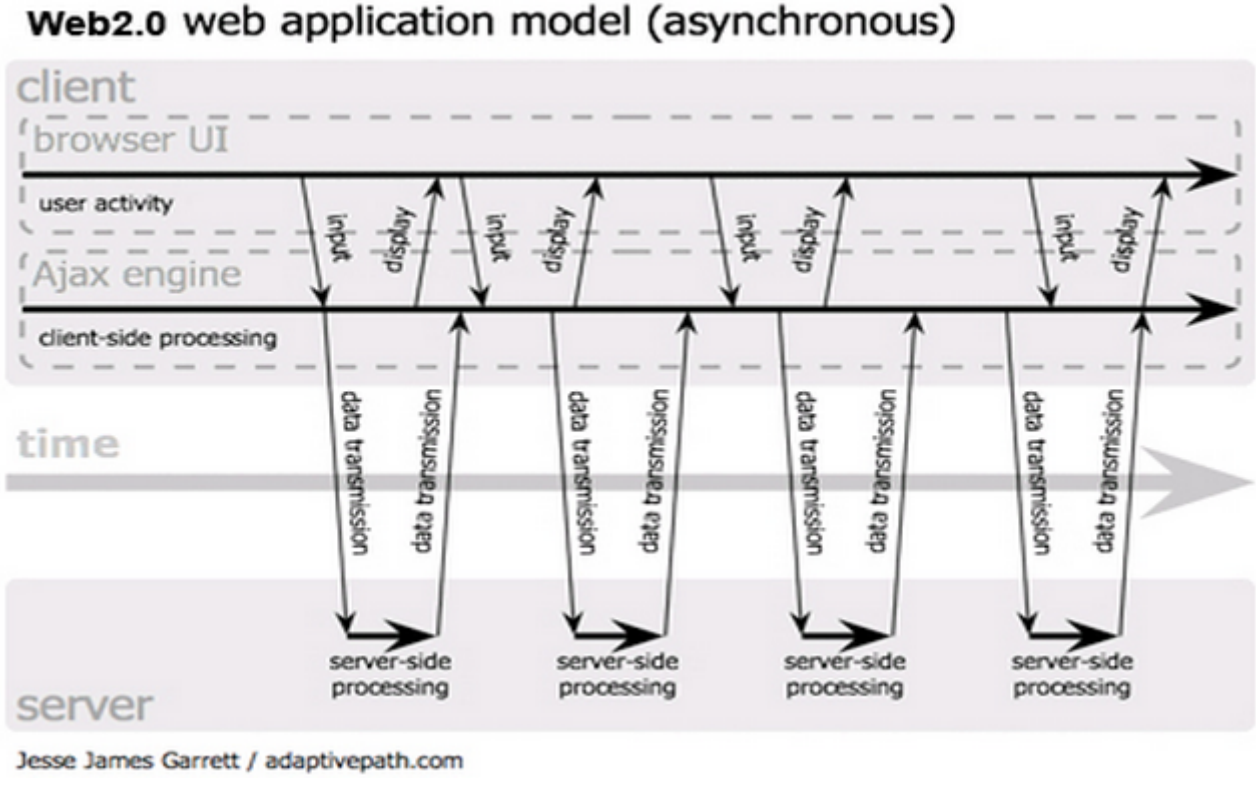
Often you need to wait inside a function for sth. to happen, for a **response**, before you can move on.

Asynchronous JavaScript

Often you need to wait inside a function for sth. to happen, for a **response**, before you can move on.

If you `fetch` an image from a server via JS for example, you cannot use that image right away - it has not been downloaded yet.

Asynchronous workflow



Online ressources

- HTTP messages
- JSON placeholders
- JSON View extension
- Arrow functions
- When to use arrow functions
- Using fetch
- Fetch API and promises
- All you need to know about Promise.all()
- Async await (ES8)

EXERCISES

Convert arabic numbers to Roman

Write a function to convert from arabic (normal) numbers to Roman Numerals. The Romans wrote numbers using letters: I, V, X, L, C, D, M. There is no need to be able to convert numbers larger than about 3000.

Example: `7` returns `VII`

Create a background gradient generator

1. Create a simple HTML file with the tags needed (see PDF of final generator)
2. Add the basic CSS file style.css and edit it to add a default gradient in the background
3. Add a JavaScript file
4. For each color of the gradient change the background gradient style with the new colors
5. Additionally output the gradient style to the HTML (see PDF of final generator)

`Promise.all()`

Look up how `Promise.all()` works and then call several JSON APIs (see JSON placeholder website in online ressources) and log the results.