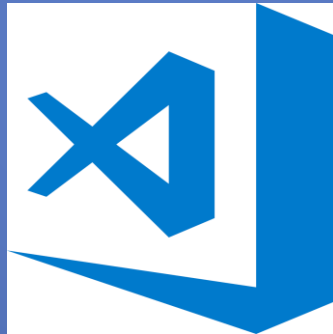


# JAVASCRIPT2

## Install

- Visual Studio Code  
<https://code.visualstudio.com/>



- Node.js  
<https://nodejs.org/en/>



# JAVASCRIPT2

## Subjects / Code examples

- ECMAScript
- var / const / let
- Classes / Inheritance
- Arrow functions
- Modules
- Arrays
- Callback functions
- Promises
- Async / Await
- Closures
- Module pattern
- Scope
- This

# JAVASCRIPT2

## JavaScript Projects

- SinglePageApplication

Frontend project

<https://github.com> - code\_simple\_SPA

- JSONServer

Backend REST project

<https://github.com> - code\_jsonserver\_with\_errors

# JAVASCRIPT2

## JavaScript Array Filter Callback Function

Create arrays

```
const names1 = ["Franco", "Alberto", "Miguel", "Jose", "Fernando", "Pepe", "Manuel"];  
const names2 = ["Franco", "Alberto", "Miguel", "Jose", "Fernando", "Pepe"];
```

Implement myFilter(array, callback)

Implement filter callback function

Return element

If name is longer than 4 letters OR name position in array is even OR number of names in array is even

Test myFilter with array and filter callback function / Test array.filter(callback)

Correct output should be...

Names1 ---> [ 'Franco', 'Alberto', 'Miguel', 'Fernando', 'Manuel' ]

Names2 ---> [ 'Franco', 'Alberto', 'Miguel', 'Jose', 'Fernando', 'Pepe' ]

# JAVASCRIPT2

## JavaScript Array Filter Callback Function

```
const names1 = ["Franco", "Alberto", "Miguel", "Jose", "Fernando", "Pepe", "Manuel"];
const names2 = ["Franco", "Alberto", "Miguel", "Jose", "Fernando", "Pepe"];
function myFilter(array, callback)
{
    let arrayFiltered = [];
    for(let index = 0; index < array.length; index++)
    {
        if(callback(array[index],index,array))
        {
            arrayFiltered.push(array[index]);
        }
    }
    return arrayFiltered;
}
const filter1 = (element, index, array) => {
    if (element.length > 4 || index % 2 == 0 || array.length % 2 == 0)
    {
        return true;
    }
    else
    {
        return false;
    }
};
console.log(myFilter(names1, filter1));
console.log(names1.filter(filter1));
console.log(myFilter(names2, filter1));
console.log(names2.filter(filter1));
```

# JSON

## JavaScript Object Notation

- JSON stands for **JavaScript Object Notation**
- JSON is a lightweight data-interchange format
- JSON is language independent \*
- JSON is "self-describing" and easy to understand
- JSON evaluates to JavaScript Object

```
var text = '{"name":"John Johnson","street":"Oslo West 16","phone":"555 1234567"}';  
var obj = JSON.parse(text);  
document.getElementById("Person").innerHTML = obj.name;
```



# JSON / XML

## Differences / Similarities

### **Much Like XML Because**

- Both JSON and XML is "self describing" (human readable)
- Both JSON and XML is hierarchical (values within values)
- Both JSON and XML can be parsed and used by lots of programming languages
- Both JSON and XML can be fetched with an XMLHttpRequest

### **Much Unlike XML Because**

- JSON doesn't use end tag
- JSON is shorter
- JSON is quicker to read and write
- JSON can use arrays
- The biggest difference is:
- XML has to be parsed with an XML parser, JSON can be parsed by a standard JavaScript function.

# JSON / XML

## Advantages

### Why JSON?

For AJAX applications, JSON is faster and easier than XML:

### Using XML

- Fetch an XML document
- Use the XML DOM to loop through the document
- Extract values and store in variables

### Using JSON

- Fetch a JSON string
- JSON.Parse the JSON string



# JSON Rules

## Syntax rules

### 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

# JSON Conversion

## String from Object

```
const employees = { employees: [  
  {firstName:"John",lastName:"Doe"},  
  {firstName:"Anna",lastName:"Smith"},  
  {firstName:"Peter",lastName:"Jones"},  
]};  
const text = JSON.stringify(employees);  
console.log(text);
```

## Object from string

```
const text = '{ "employees" : [' +  
  '{ "firstName":"John" , "lastName":"Doe" },' +  
  '{ "firstName":"Anna" , "lastName":"Smith" },' +  
  '{ "firstName":"Peter" , "lastName":"Jones" } ]}';  
const obj = JSON.parse(text);  
console.log(obj.employees[1].firstName + " " + obj.employees[1].lastName);
```

## Older browsers / No JSON.parse() support

```
const text = '{ "employees" : [' +  
  '{ "firstName":"John" , "lastName":"Doe" },' +  
  '{ "firstName":"Anna" , "lastName":"Smith" },' +  
  '{ "firstName":"Peter" , "lastName":"Jones" } ]}';  
const obj = eval ("(" + text + ")");  
console.log(obj.employees[1].firstName + " " + obj.employees[1].lastName);
```

# AJAX

## XML

- AJAX = Asynchronous JavaScript and XML.
- AJAX is not a programming language, but a way to use existing standards.
- AJAX is the art of exchanging data with a server, and updating parts of a web page - without reloading the whole page.
- AJAX is a technique for creating fast and dynamic web pages.
- AJAX allows web pages to be updated asynchronously by exchanging small amounts of data with the server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.
- The XMLHttpRequest object is used to exchange data with a server.

# XMLHttpRequest object

## Asynchronous

### TRUE

- Execute other scripts while waiting for server response
- Deal with the response when the response ready

### FALSE

- Will NOT continue to execute, until the server response is ready. If the server is busy or slow, the application will hang or stop.

# Server response

## **responseText**

- get the response data as a string

## **responseXML**

- get the response data as XML data

# Events

## onreadystatechange

The onreadystatechange event is triggered every time the readyState changes. The readyState property holds the status of the XMLHttpRequest.

Property	Description
onreadystatechange	Stores a function (or the name of a function) to be called automatically each time the readyState property changes
readyState	Holds the status of the XMLHttpRequest. Changes from 0 to 4: 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready
status	200: "OK" 404: Page not found



# AJAX Basic

```
const ajaxBasic = () => {  
  const request = new XMLHttpRequest();  
  request.onreadystatechange = () => {  
    console.log('State: ' + request.readyState + ' Status: ' + request.status);  
    if (request.readyState === 4 && request.status === 200) {  
      console.log(JSON.parse(request.responseText));  
    }  
  }  
  request.open('GET', 'http://localhost:3333/api/users', true);  
  request.send();  
}  
ajaxBasic();
```

# AJAX General

```
const ajaxGeneral = (url, cb) => {  
  const request = new XMLHttpRequest();  
  request.onreadystatechange = () => {  
    if (request.readyState == 4 && request.status == 200) {  
      cb(JSON.parse(request.responseText));  
    }  
  }  
  request.open("GET", url, true);  
  request.send();  
}  
ajaxGeneral('http://localhost:3333/api/users', (json) => {  
  console.log(json);  
});
```

# FETCH Basic

```
const fetchBasic = () => {  
  fetch('http://localhost:3333/api/users')  
    .then(function (response) {  
      return response.json();  
    })  
    .then(function (json) {  
      console.log(json);  
    });  
}  
fetchBasic();
```

# FETCH General

```
const fetchGeneral = (url, cb) => {  
  fetch(url)  
    .then(function (response) {  
      return response.json();  
    }).then(function (json) {  
      cb(json);  
    }).catch(function (error) {  
      console.log('Error: ' + error);  
    });  
}  
fetchGeneral('http://localhost:3333/api/users', json  
=> {  
  console.log(json);  
});
```

# FETCH Options

```
function makeOptions(requestType, body) {  
  return {  
    method: requestType,  
    headers: {  
      'Accept': 'application/json',  
      'Content-Type': 'application/json'  
    },  
    body: JSON.stringify(body)  
  };  
}
```

# FETCH Pattern

```
function handleHttpErrors(response) {  
  if (!response.ok) {  
    return Promise.reject({ status: response.status, error: response.json() })  
  }  
  return response.json();  
}  
  
const url1 = 'http://localhost:3333/api/users';  
const data1 = { age: 34, name: "Anton Benson", gender: "female", email: "ab@ab.com" };  
const options1 = makeOptions("POST", data1);  
fetch(url1, options1)  
  .then(handleHttpErrors)  
  .then(json => console.log(json))  
  .catch(error => {  
    if (error.status)  
    {  
      error.error.then(e => console.log(e))  
    }  
    else  
    {  
      console.log("Network error");  
    }  
  })  
});
```



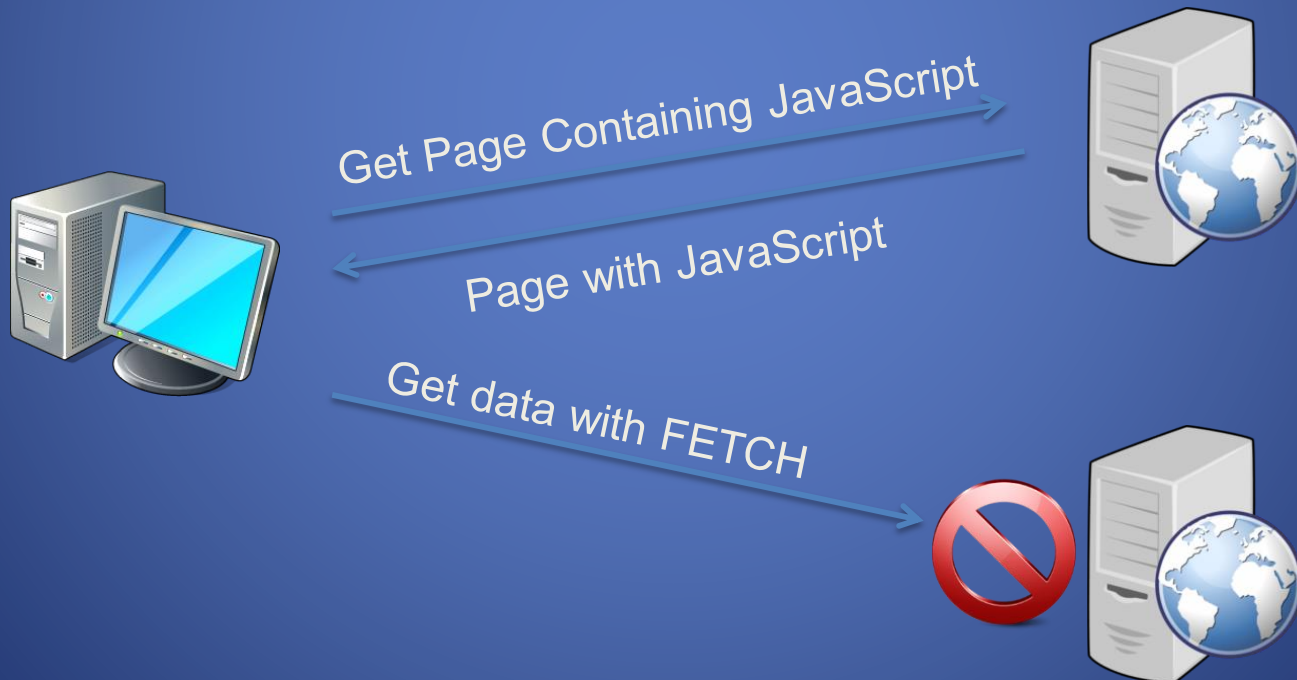
# FETCH Final

```
const fetchJson = async (url, options) => {
  try {
    const response = await fetch(url, options);
    const json = await response.json();
    if (!response.ok) {
      return Promise.reject({ status: response.status, json: json });
    }
    return Promise.resolve({ json: json });
  }
  catch (error) {
    return Promise.reject({ error: 'Network error' });
  }
}

const url2 = 'http://localhost:3333/api/users';
const data2 = { age: 34, name: "Anton Benson", gender: "male", email: "ab@ab.com" };
const options2 = makeOptions("POST", data2);
fetchJson(url2, options2)
  .then((resolve) => console.log(resolve.json))
  .catch((reject) => console.log(reject.json));
```

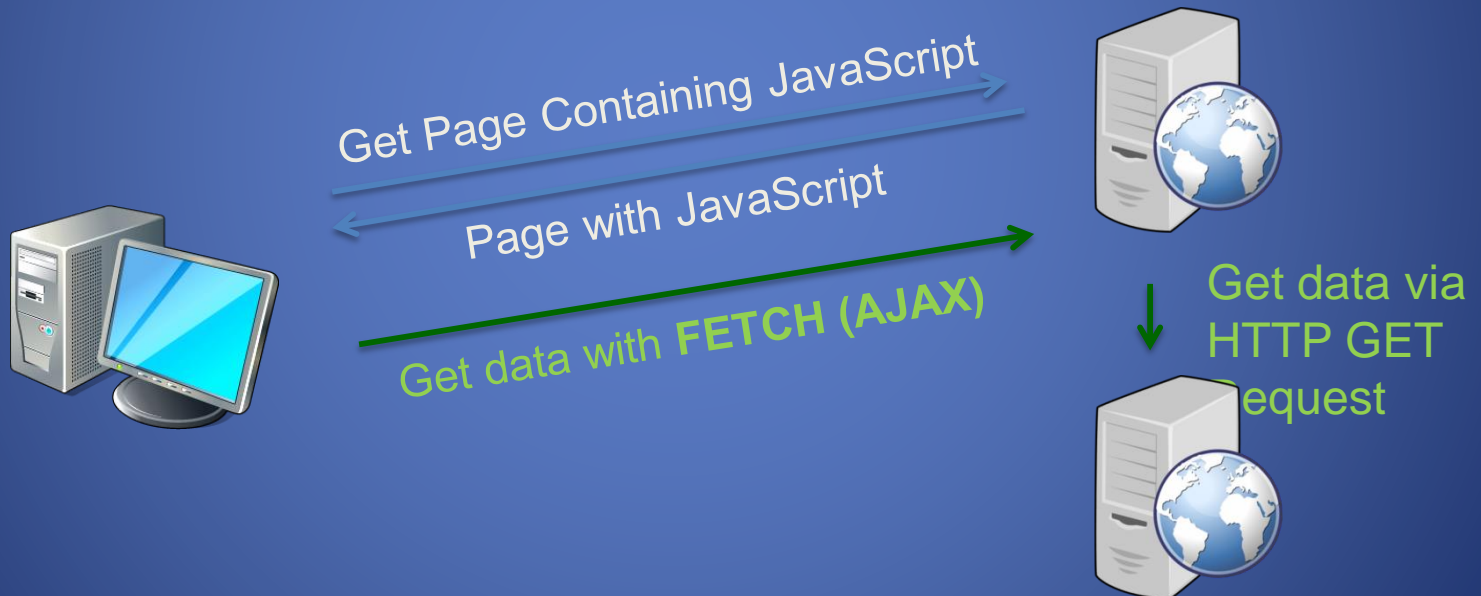
# SOP

**Same Origin Policy**  
**Problem: External servers disallowed**



# SOP

Same Origin Policy  
Solution. Proxy server



# CORS

## Cross Origin resource sharing

Technique for relaxing the same-origin policy

```
Response.ok(json).header("Access-Control-Allow-Origin", "*").build();
```