Università di Roma
Tor Vergata

AJAX

AJAX

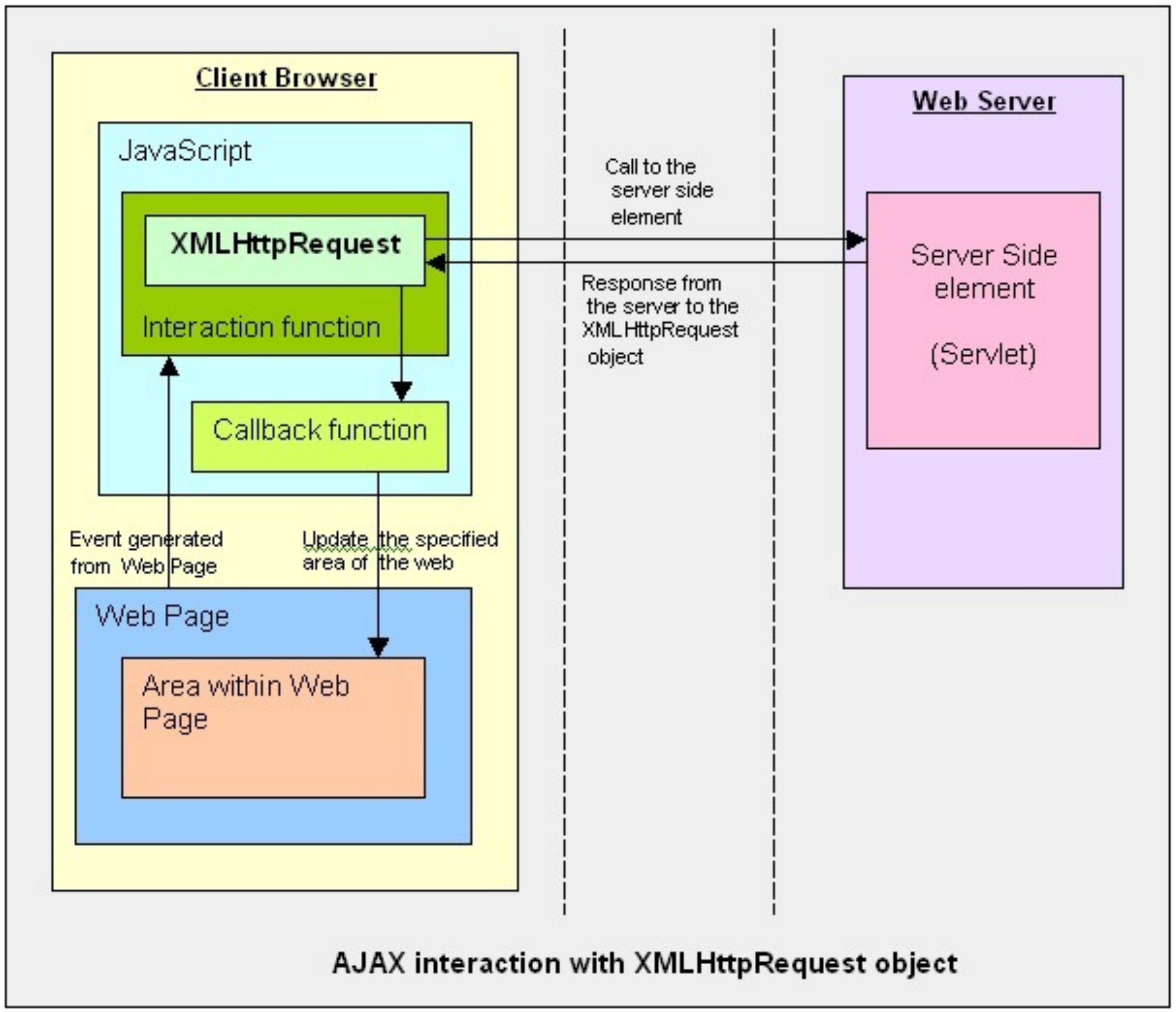
- **A**synchronous **J**avaScript **A**nd **X**ML.
- AJAX usa solo una combinazione di:
  - Un oggetto XMLHttpRequest incorporato nel browser (per richiedere dati da un server Web)
  - JavaScript e HTML DOM (per visualizzare o utilizzare i dati)
  -
- AJAX è il sogno di uno sviluppatore, perché puo:
  - Aggiornare una pagina Web senza ricaricare la pagina
  - Richiedere dati a un server - dopo che la pagina è stata caricata
  - Ricevere dati da un server - dopo che la pagina è stata caricata
  - Inviare dati a un server - in background

XMLHttpRequest

- Crea una richiesta web
- Metodi/attributi più utilizzati:
  - **open**('GET', 'http://www.uniroma2.it', false)
    - Il terzo parametro dice se la richiesta deve essere asincrona. Se async=true
  - **send**() - Invia la richiesta
  - **responseText** - La risposta (DOMString)

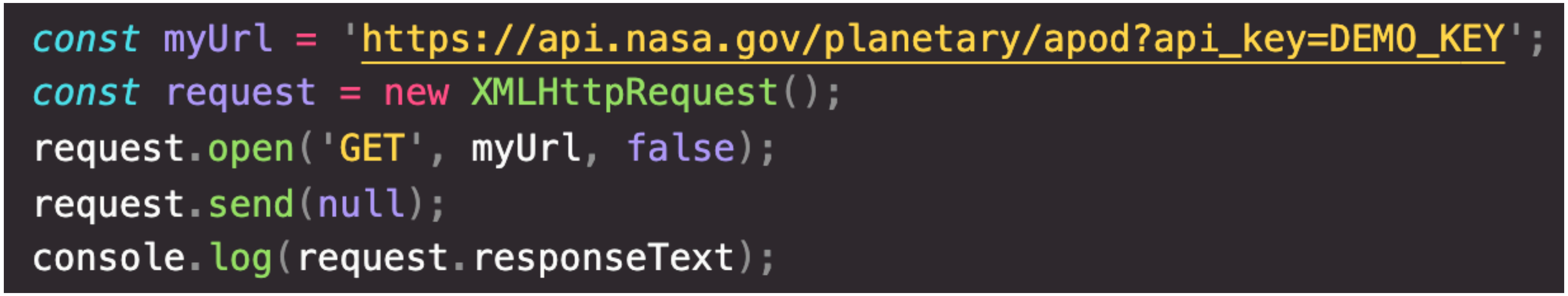
Guida: https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest

Architettura XMLHttpRequest


Client Browser
JavaScript
XMLHttpRequest
Interaction function
Callback function
Event generated from Web Page
Update the specified area of the web
Web Page
Area within Web Page
Call to the server side element
Response from the server to the XMLHttpRequest object
Web Server
Server Side element
(Servlet)

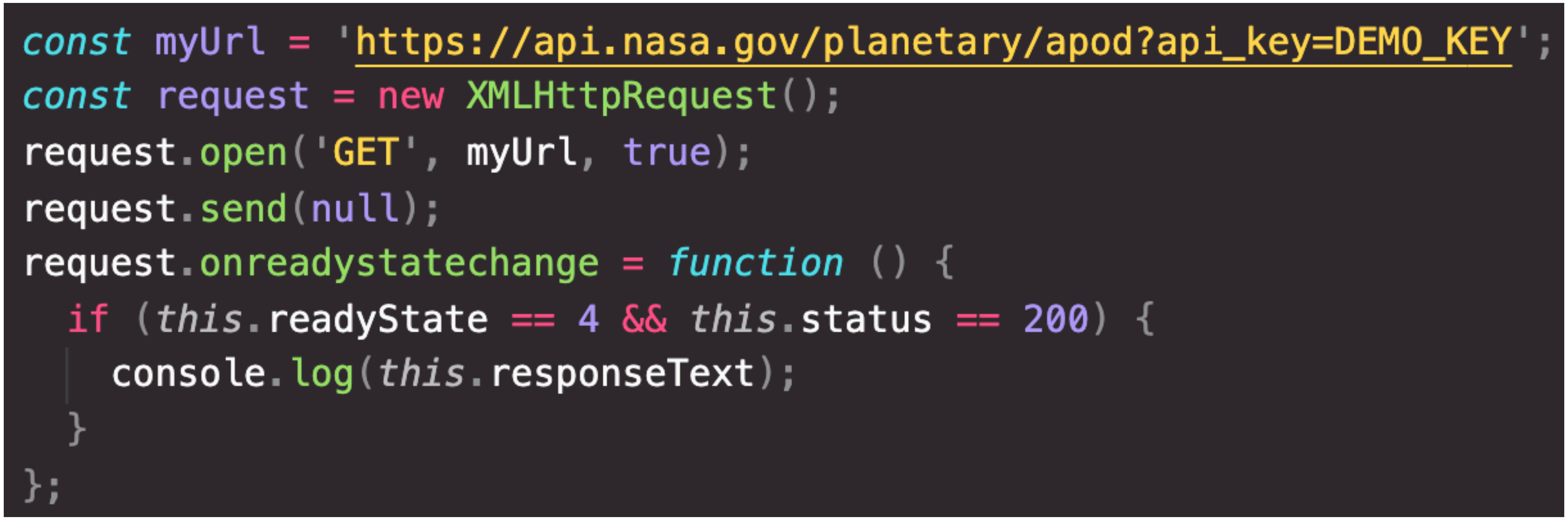
AJAX interaction with XMLHttpRequest object

Esempio XMLHttpRequest (sync)



```
const myUrl = 'https://api.nasa.gov/planetary/apod?api_key=DEMO_KEY';
const request = new XMLHttpRequest();
request.open('GET', myUrl, false);
request.send(null);
console.log(request.responseText);
```

Esempio XMLHttpRequest (async)

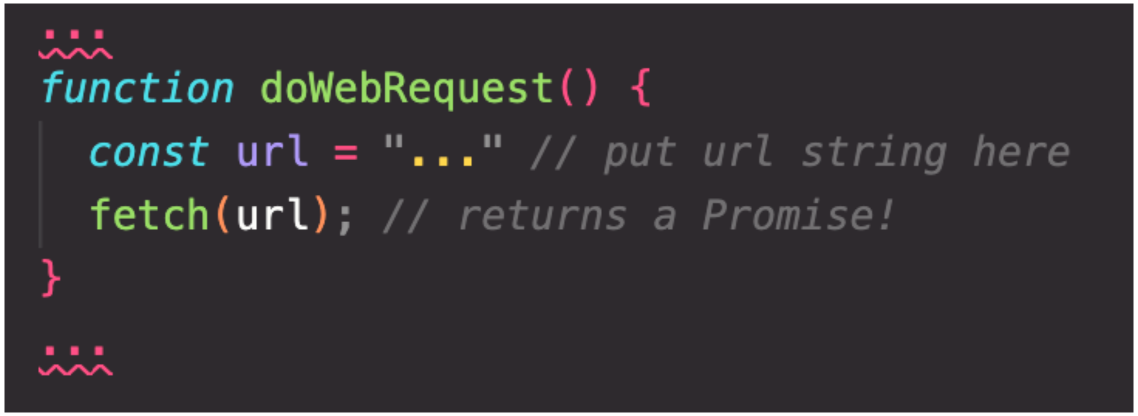


```javascript
const myUrl = 'https://api.nasa.gov/planetary/apod?api_key=DEMO_KEY';
const request = new XMLHttpRequest();
request.open('GET', myUrl, true);
request.send(null);
request.onreadystatechange = function () {
  if (this.readyState == 4 && this.status == 200) {
    console.log(this.responseText);
  }
};
```

AJAX with Fetch

fetch API

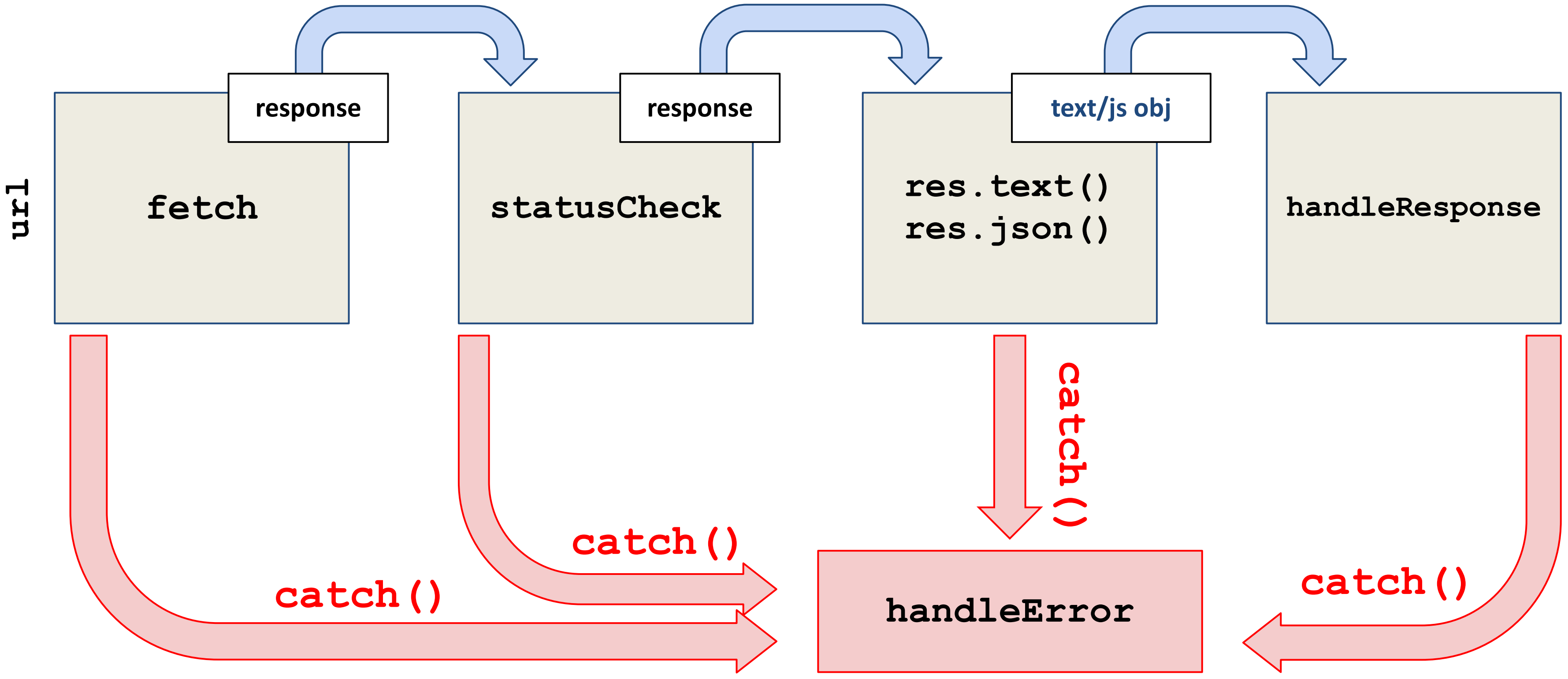
- Fetch
  - promise-based API for Ajax requests
  - replace **XMLHttpRequest**
  - now supported in all modern browsers



```
...
function doWebRequest() {
  const url = "..." // put url string here
  fetch(url); // returns a Promise!
}
...
```

https://www.digitalocean.com/community/tutorials/js-fetch-api

The Promise Pipeline

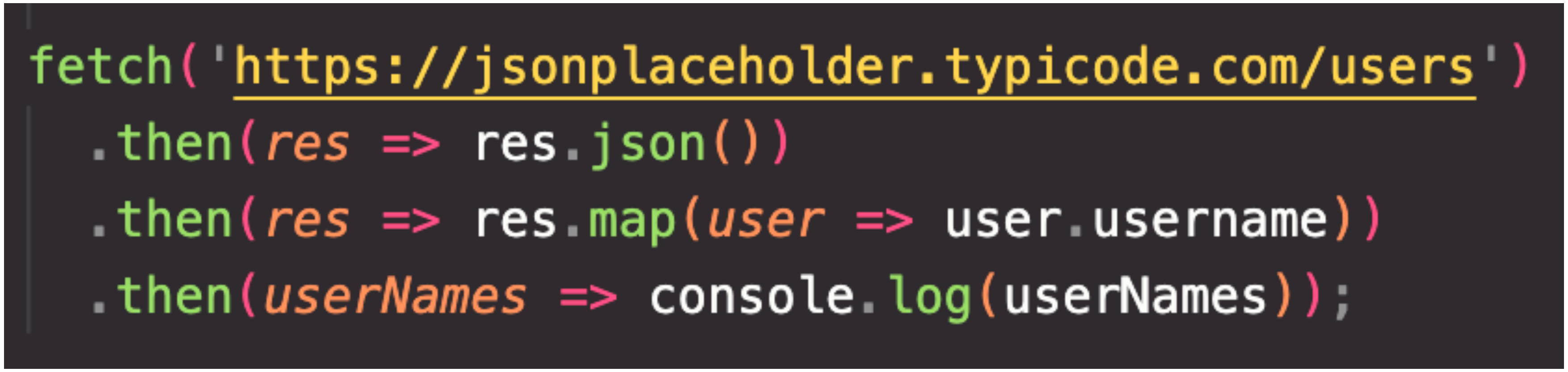


Esempio GET 1

```
<p id="demo">Fetch a file to change this text.</p>

<script>
    let file = "fetch_info.txt"
    fetch (file)
    .then(x => x.text())
    .then(y => document.getElementById("demo").innerHTML = y);
</script>
```

https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_api_fetch

Esempio GET Json



```
fetch('https://jsonplaceholder.typicode.com/users')
   .then(res => res.json())
   .then(res => res.map(user => user.username))
   .then(userNames => console.log(userNames));
```

Nota: sulla console funziona solo sulla pagina: https://jsonplaceholder.typicode.com/

Esercizio Dog Image

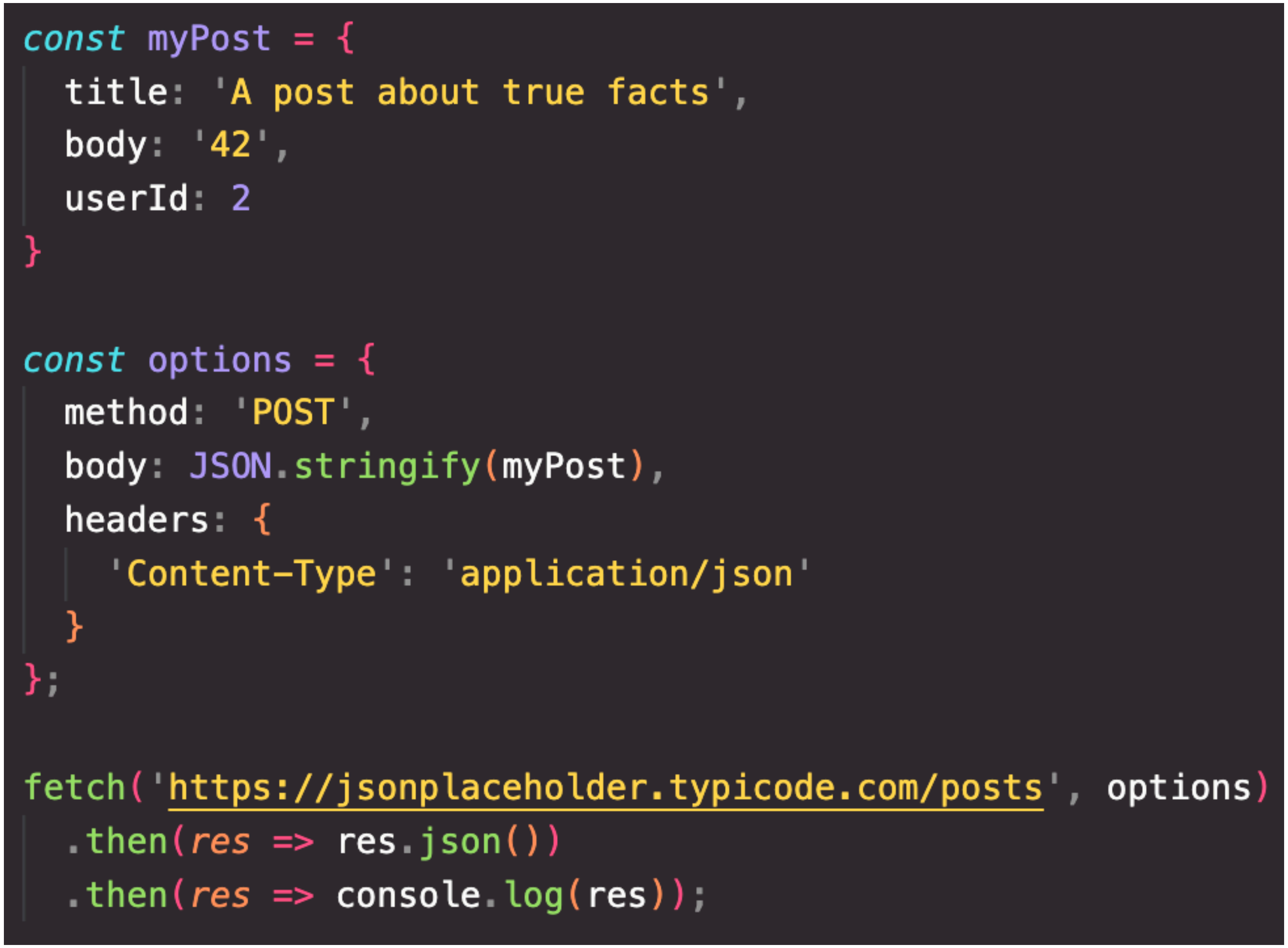
- https://dog.ceo/dog-api/

- Realizzare una pagina con un bottone che cliccato mostra un immagine casuale di un cane presa dal sito dog.ceo
  - API: https://dog.ceo/api/breeds/image/random

```
{
    "message": "https://images.dog.ceo/breeds/leonberg/n02111129_4435.jpg",
    "status": "success"
}
```

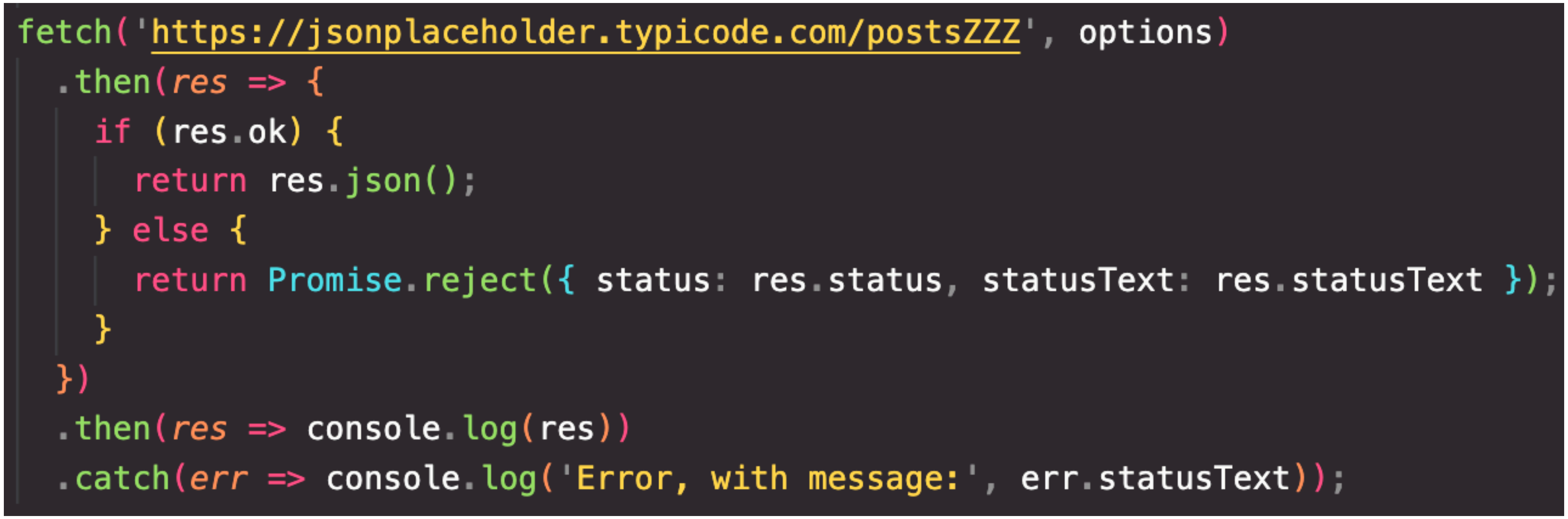
  - Nota: creare l'elemento image nella pagina

Esempio POST



```
const myPost = {
  title: 'A post about true facts',
  body: '42',
  userId: 2
}

const options = {
  method: 'POST',
  body: JSON.stringify(myPost),
  headers: {
    'Content-Type': 'application/json'
  }
};

fetch('https://jsonplaceholder.typicode.com/posts', options)
  .then(res => res.json())
  .then(res => console.log(res));
```

Gestione dell'Errore



```
fetch('https://jsonplaceholder.typicode.com/postsZZZ', options)
  .then(res => {
    if (res.ok) {
      return res.json();
    } else {
      return Promise.reject({ status: res.status, statusText: res.statusText });
    }
  })
  .then(res => console.log(res))
  .catch(err => console.log('Error, with message:', err.statusText));
```

Università di Roma
Tor Vergata

ASYNC AWAIT

async

```
async function f() {
  return 1;
}
```

- async before a function means that a function always returns a promise

async

```
async function f() {
  return 1;
}
```

- async before a function means that a function always returns a promise

```
async function f() {
  return 1;
}

f().then(alert); // 1
```

async

```
async function f() {
  return 1;
}
```

- async before a function means that a function always returns a promise

```
async function f() {
  return 1;
}

f().then(alert); // 1
```

```
async function f() {
  return Promise.resolve(1);
}

f().then(alert); // 1
```

await

- Rende il codice asincrono ed aspetta la risposta

```
// works only inside async functions
let value = await promise;
```

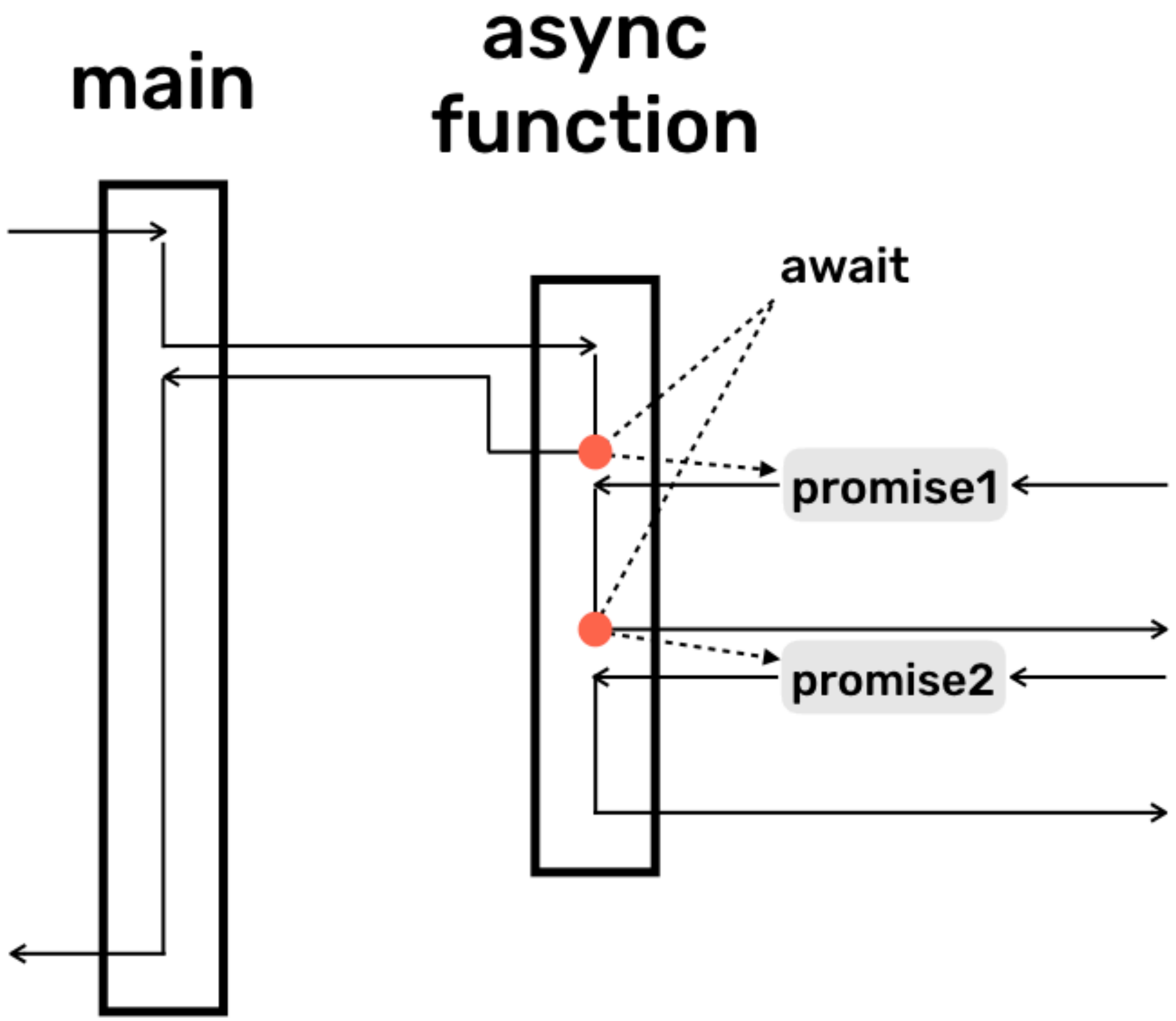
await

- Rende il codice asincrono ed aspetta la risposta futura

```
// works only inside async functions
let value = await promise;
```

```
async function f() {

  let promise = new Promise((resolve, reject) => {
    setTimeout(() => resolve("done!"), 1000)
  });

  let result = await promise; // wait until the promise resolves (*)

  alert(result); // "done!"
}

f();
```

Control flow of async/await



https://dmitrykandalov.com/async-await

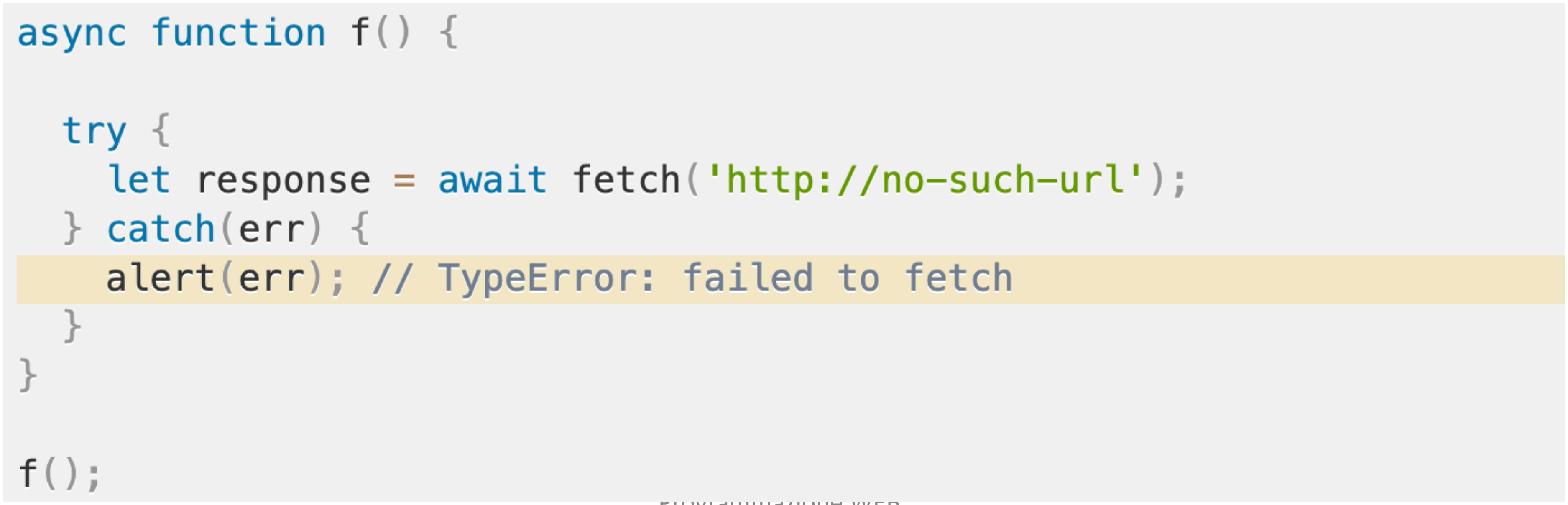
error handling: try catch

- in the case of a rejection a promise throws the error
  - as if there were a throw statement at that line

error handling: try catch

- in the case of a rejection a promise throws the error
  - as if there were a throw statement at that line

- can catch that error using try..catch

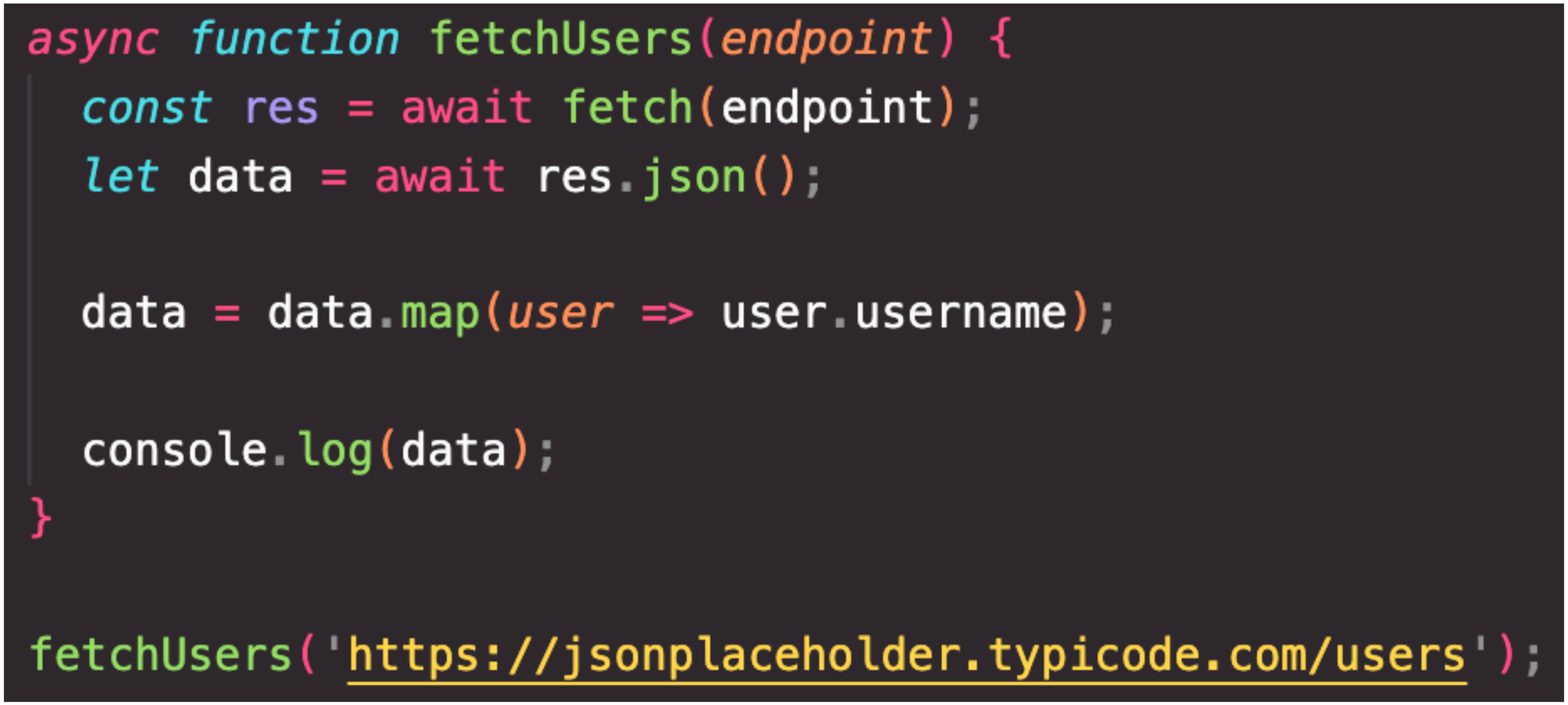


```
async function f() {

  try {
    let response = await fetch('http://no-such-url');
  } catch(err) {
    alert(err); // TypeError: failed to fetch
  }
}

f();
```

Università di Roma
Tor Vergata

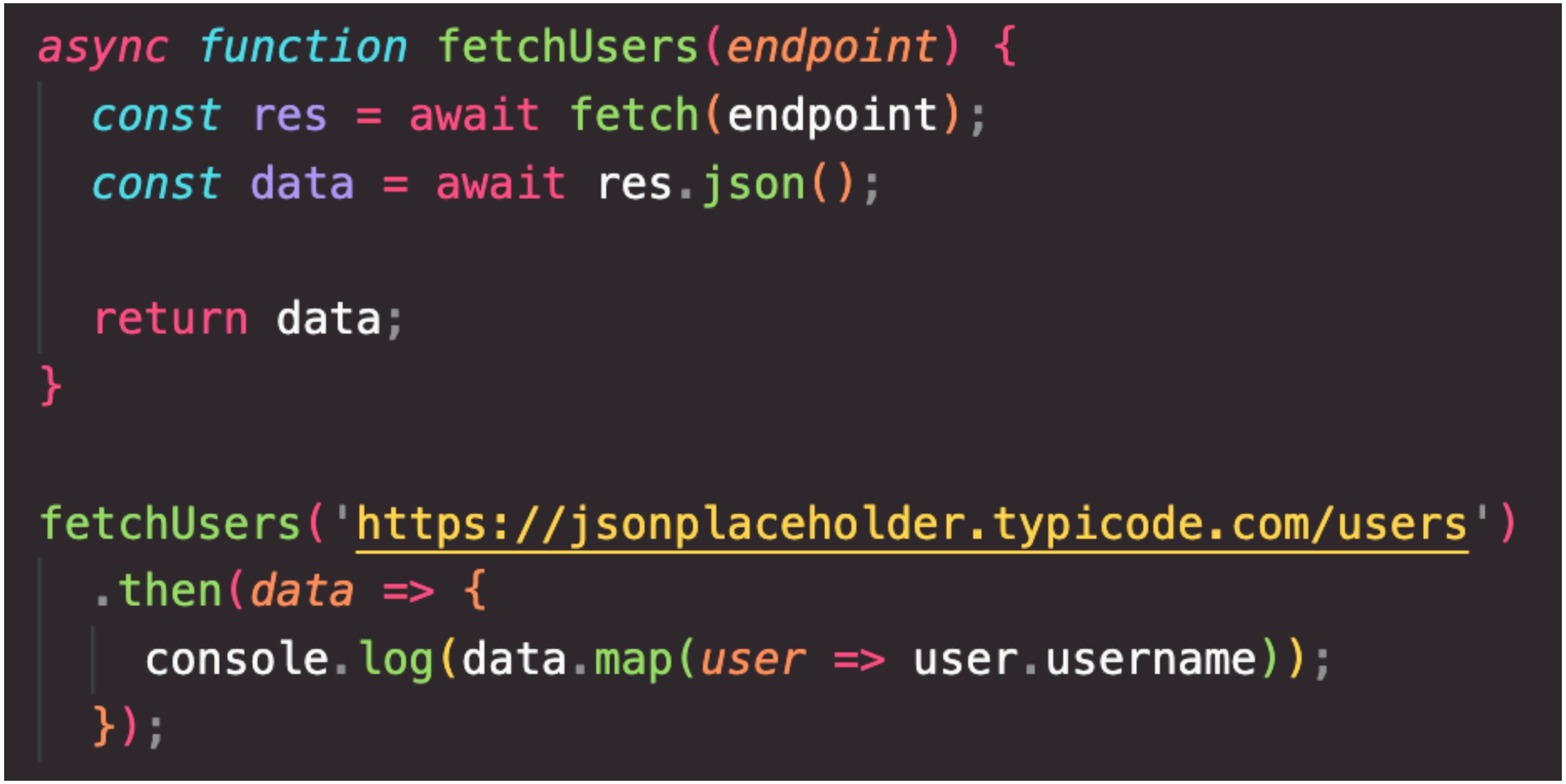
Fetch with Async/Await

GET



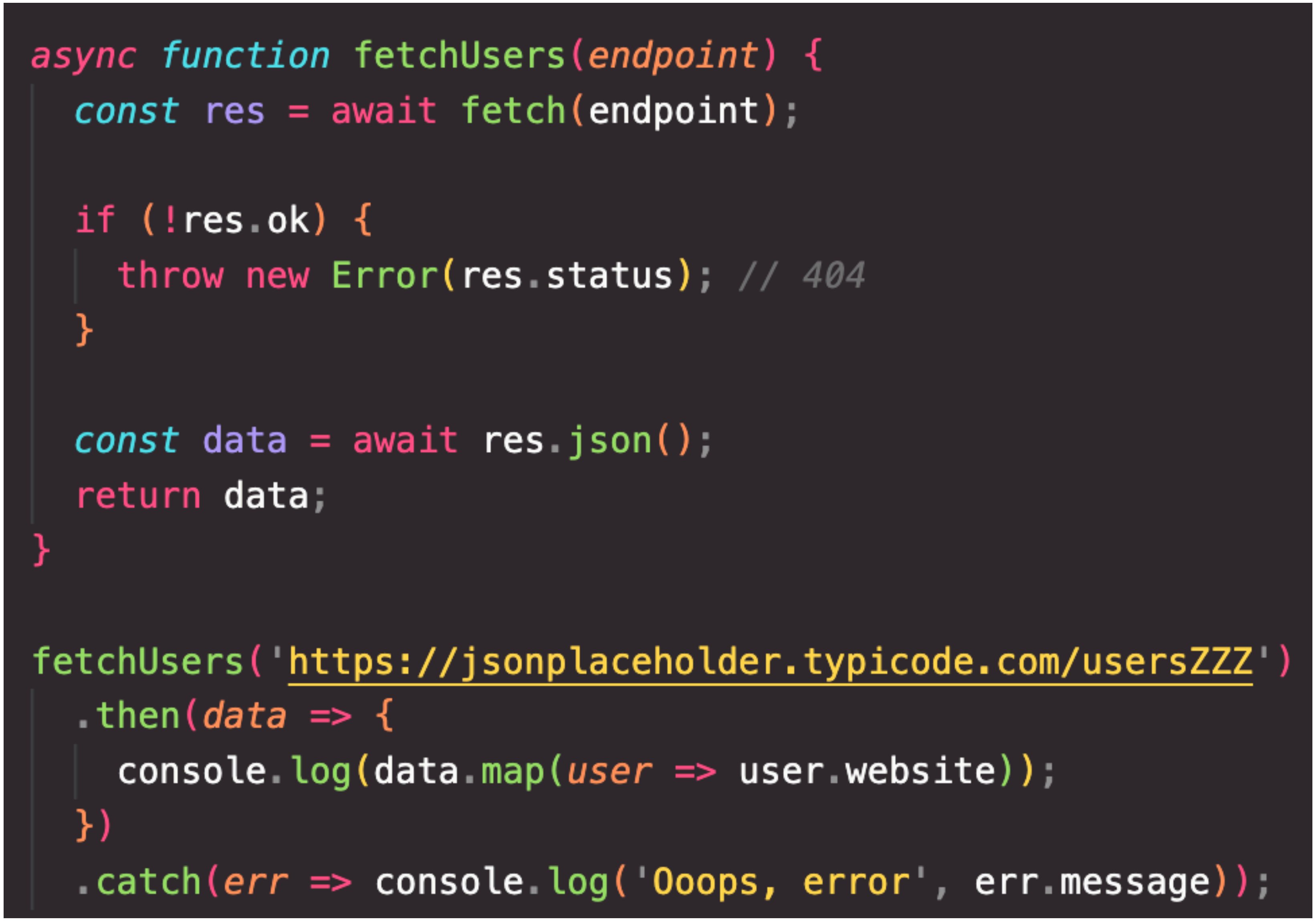
```
async function fetchUsers(endpoint) {
  const res = await fetch(endpoint);
  let data = await res.json();

  data = data.map(user => user.username);

  console.log(data);
}

fetchUsers('https://jsonplaceholder.typicode.com/users');
```

GET V2



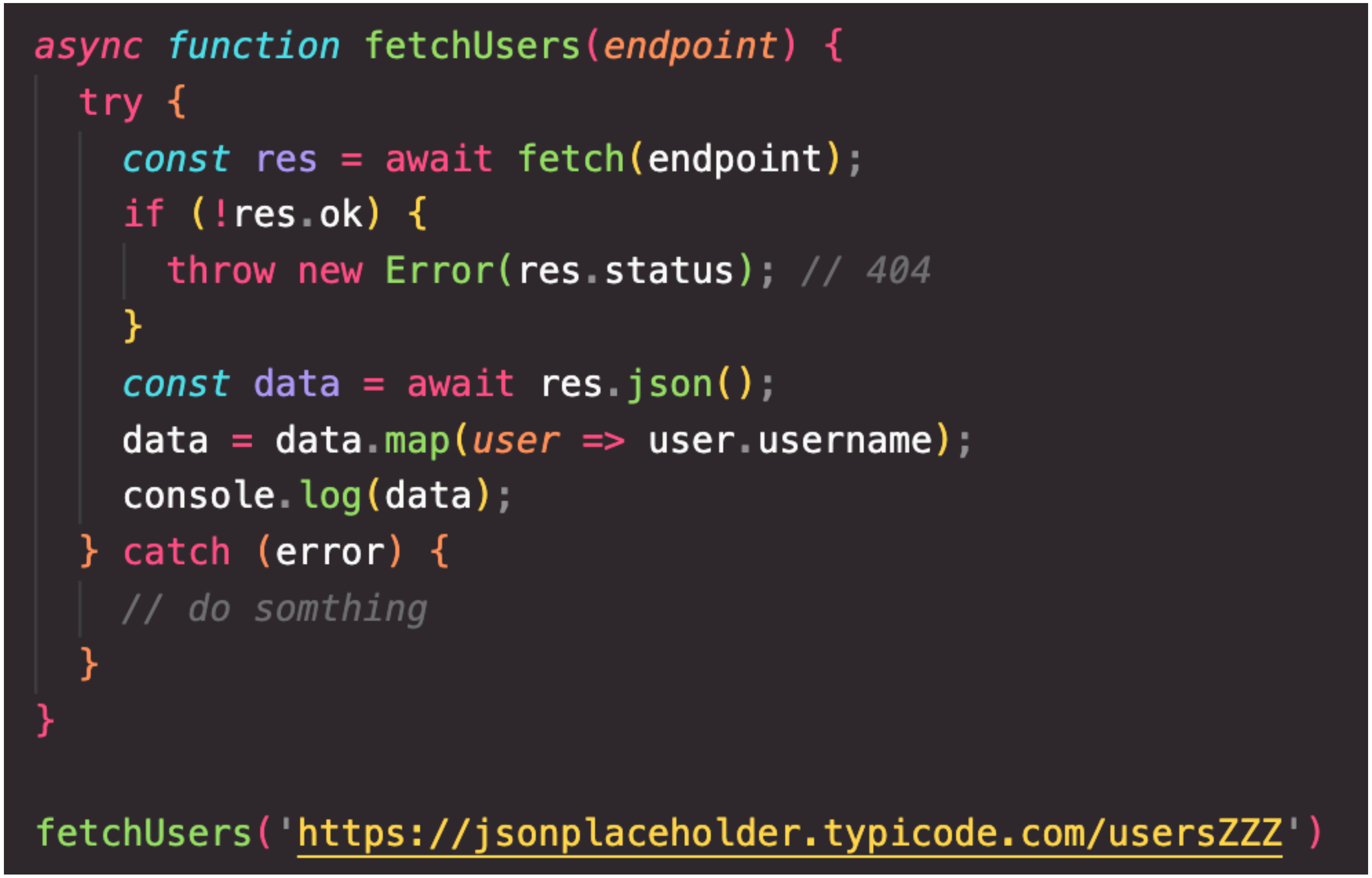
```
async function fetchUsers(endpoint) {
  const res = await fetch(endpoint);
  const data = await res.json();

  return data;
}

fetchUsers('https://jsonplaceholder.typicode.com/users')
  .then(data => {
    console.log(data.map(user => user.username));
  });
```

Errors



```
async function fetchUsers(endpoint) {
  const res = await fetch(endpoint);

  if (!res.ok) {
    throw new Error(res.status); // 404
  }

  const data = await res.json();
  return data;
}

fetchUsers('https://jsonplaceholder.typicode.com/usersZZZ')
  .then(data => {
    console.log(data.map(user => user.website));
  })
  .catch(err => console.log('Ooops, error', err.message));
```

Errors V2



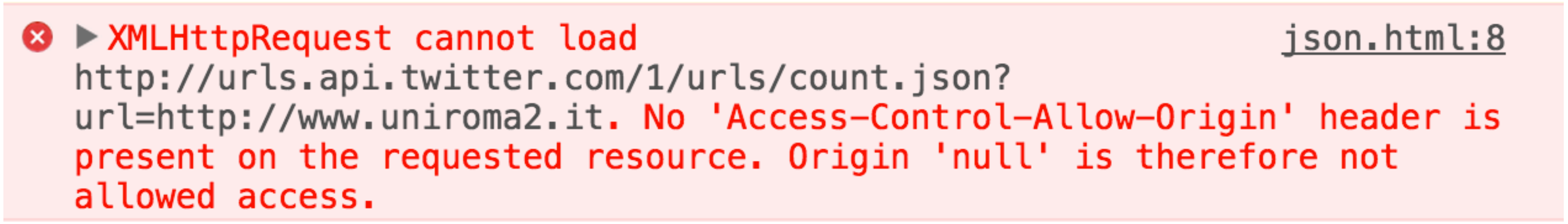
```
async function fetchUsers(endpoint) {
  try {
    const res = await fetch(endpoint);
    if (!res.ok) {
      throw new Error(res.status); // 404
    }
    const data = await res.json();
    data = data.map(user => user.username);
    console.log(data);
  } catch (error) {
    // do somthing
  }
}

fetchUsers('https://jsonplaceholder.typicode.com/usersZZZ')
```

Università di Roma
Tor Vergata

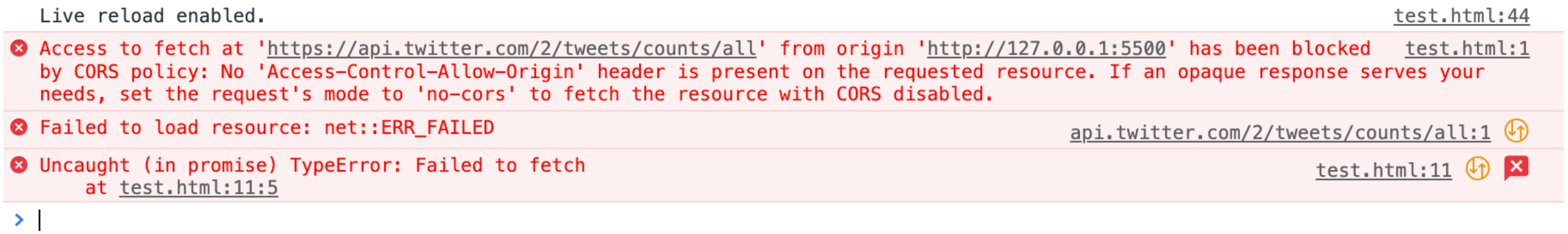
CORS

Same origin policy

- Un browser permette agli script contenuti in una pagina web di accedere ai dati contenuti in un’altra risorsa web (altra pagina web, json ecc) solo se entrambe le pagine hanno la stessa origine



```
▶XMLHttpRequest cannot load                                          json.html:8
http://urls.api.twitter.com/1/urls/count.json?
url=http://www.uniroma2.it. No 'Access-Control-Allow-Origin' header is
present on the requested resource. Origin 'null' is therefore not
allowed access.
```



```
Live reload enabled.                                                                                    test.html:44
Access to fetch at 'https://api.twitter.com/2/tweets/counts/all' from origin 'http://127.0.0.1:5500' has been blocked   test.html:1
by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource. If an opaque response serves your
needs, set the request's mode to 'no-cors' to fetch the resource with CORS disabled.
Failed to load resource: net::ERR_FAILED                                        api.twitter.com/2/tweets/counts/all:1
Uncaught (in promise) TypeError: Failed to fetch                                                        test.html:11
    at test.html:11:5
>
```

CORS: cross-origin HTTP request

- Uno script js fa una chiamata http ad un differente dominio, protocollo o porta!!!!

- Esempio:

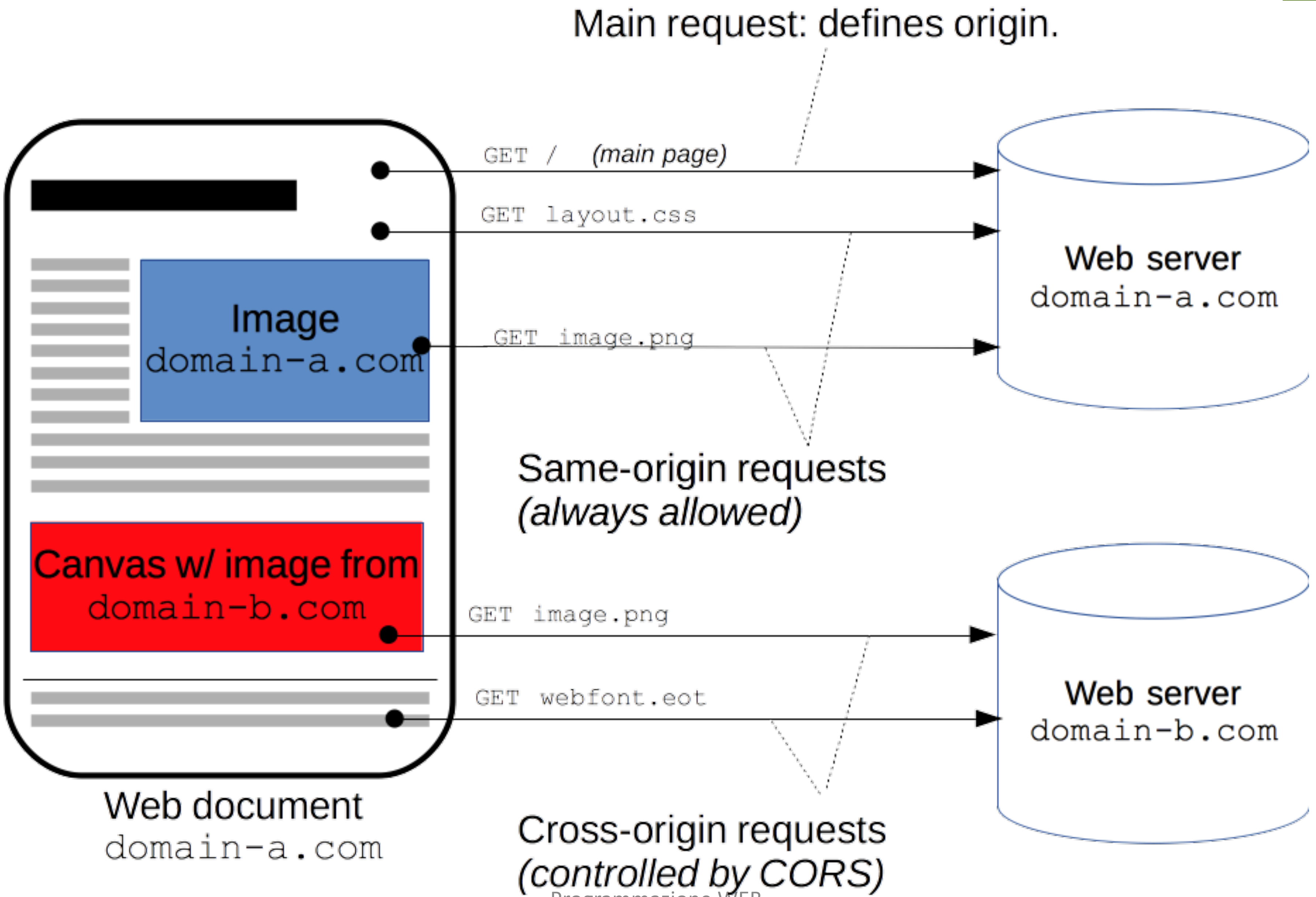
Pagina principale: https://www.miosito.com

Chiamate CORS

  - https://api.altrosito.com/data
  - http://www.miosito.com/data
  - https://www.miosito.com:3000/data

https://italiancoders.it/cors-in-dettaglio/

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


Main request: defines origin.
GET / (main page)
GET layout.css
Image
domain-a.com
GET image.png
Web server
domain-a.com
Same-origin requests
(always allowed)
Canvas w/ image from
domain-b.com
GET image.png
GET webfont.eot
Web server
domain-b.com
Web document
domain-a.com
Cross-origin requests
(controlled by CORS)

Condivisione di Risorse tra Domini

- **CORS (Cross-Origin Resource Sharing)?**
  - È uno **standard W3C** che permette a un sito web di accedere a risorse ospitate su un dominio diverso.

- Viene implementato inviando degli Header HTTP in req/resp

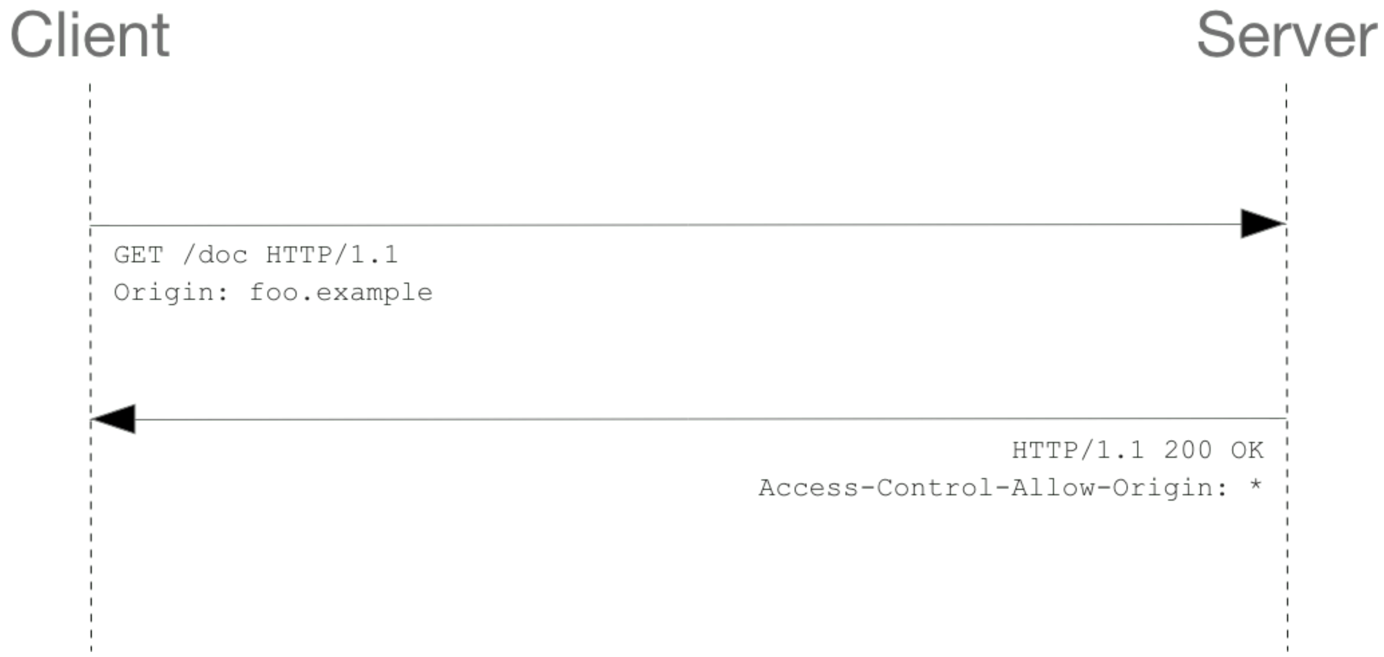
Tipo di Richiesta	Quando Avviene
Semplice	Metodo: GET, POST, HEAD + header standard
Preflight	Per metodi PUT, DELETE, header custom, ecc.

Simple request

- Metodi Ammessi
  - GET, HEAD, POST
- Header pemessi
  - Accept
  - Accept-Language
  - Content-Language
  - Content-Type (solo con determinati valori, vedi sotto)
  - …
- Valori ammessi per header **Content-Type**:
  - application/x-www-form-urlencoded
  - multipart/form-data
  - text/plain

Simple request

```
const xhr = new XMLHttpRequest();
const url = 'https://bar.other/resources/public-data/';

xhr.open('GET', url);
xhr.onreadystatechange = someHandler;
xhr.send();
```



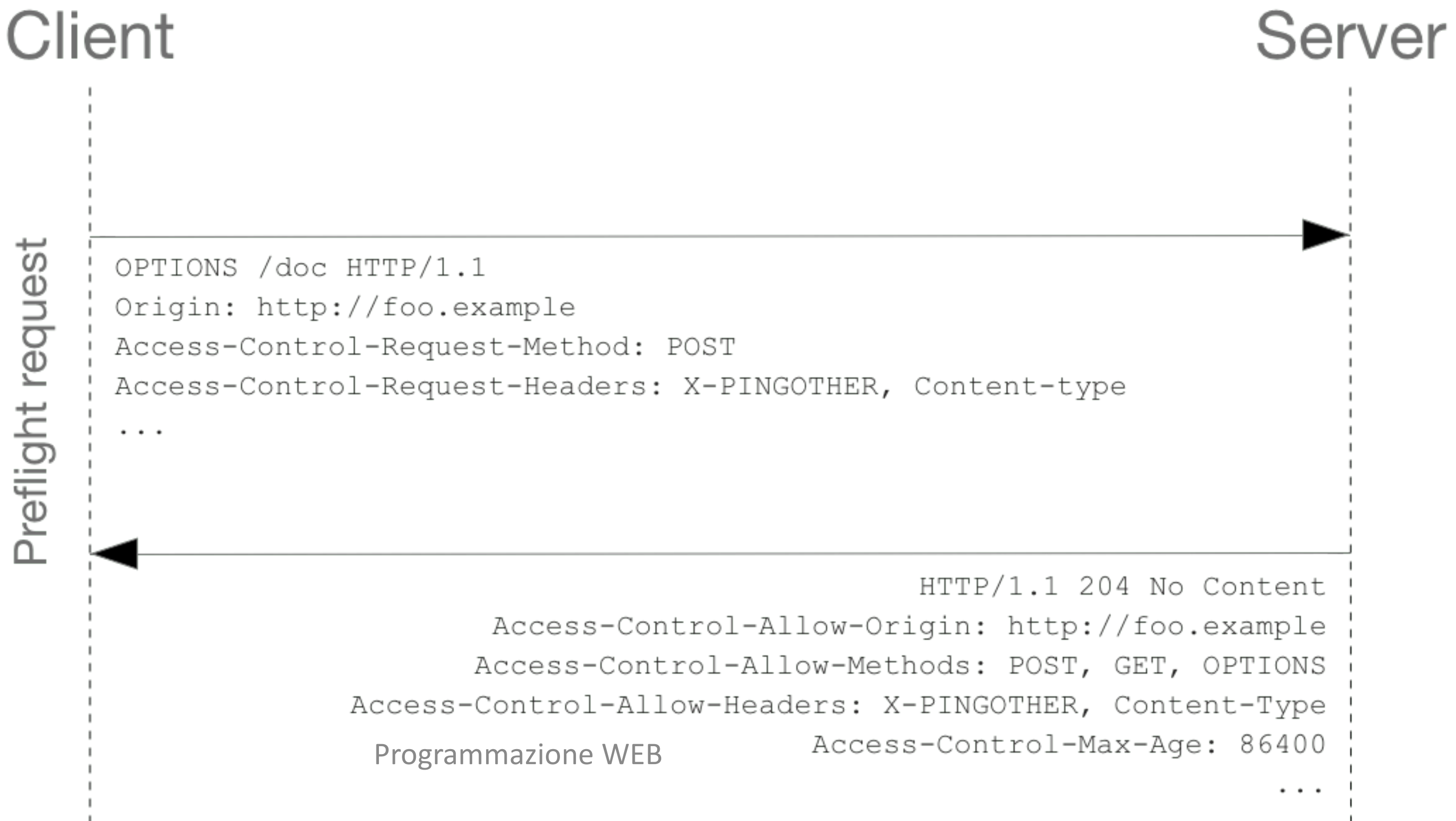
```
GET /doc HTTP/1.1
Origin: foo.example
```

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
```

Access-Control-Allow-Origin: * means that the resource can be accessed by **any** origin.

Pre-flight request

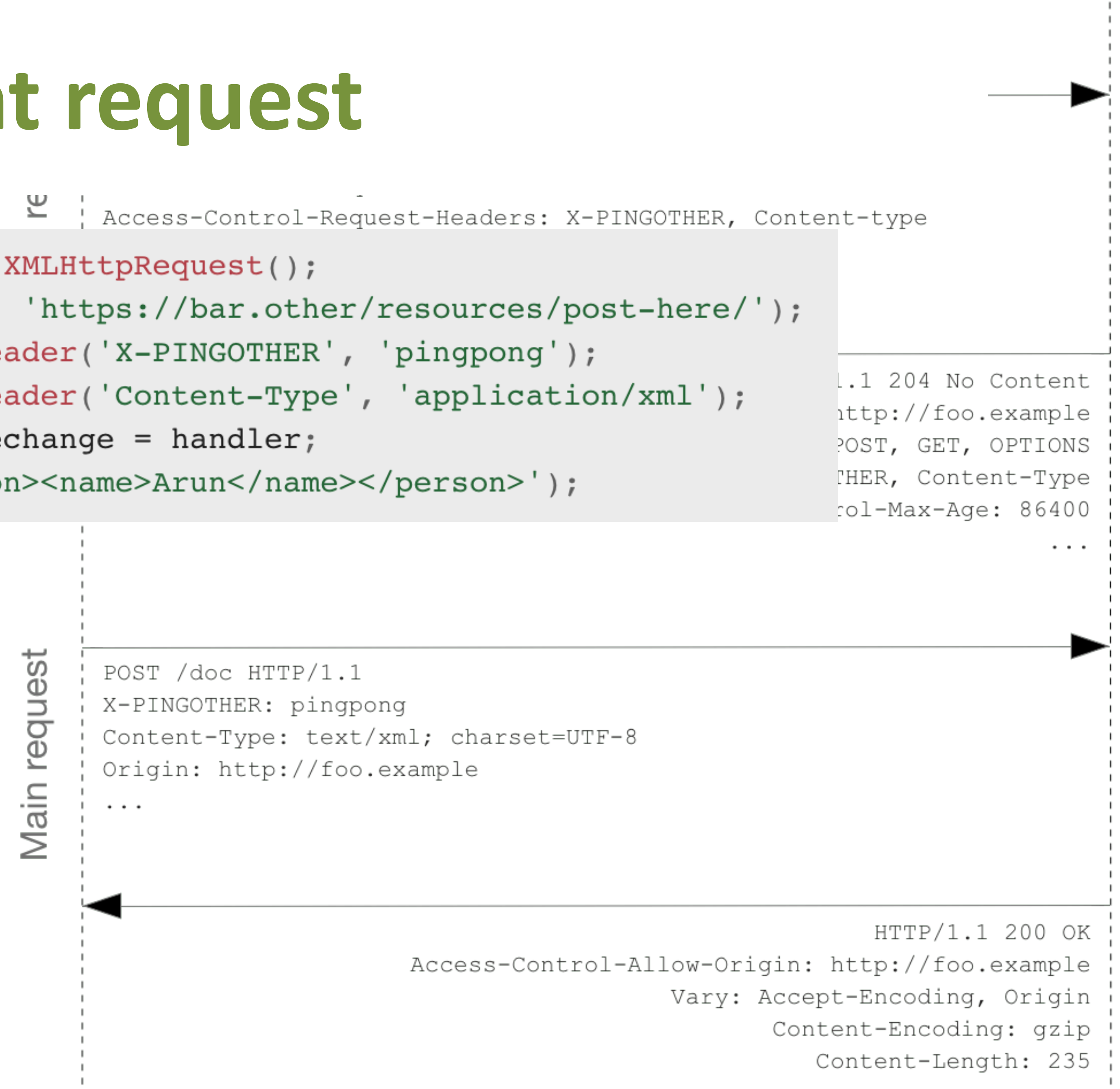
```
const xhr = new XMLHttpRequest();
xhr.open('POST', 'https://bar.other/resources/post-here/');
xhr.setRequestHeader('X-PINGOTHER', 'pingpong');
xhr.setRequestHeader('Content-Type', 'application/xml');
xhr.onreadystatechange = handler;
xhr.send('<person><name>Arun</name></person>');
```



```
OPTIONS /doc HTTP/1.1
Origin: http://foo.example
Access-Control-Request-Method: POST
Access-Control-Request-Headers: X-PINGOTHER, Content-type
...
```

```
HTTP/1.1 204 No Content
Access-Control-Allow-Origin: http://foo.example
Access-Control-Allow-Methods: POST, GET, OPTIONS
Access-Control-Allow-Headers: X-PINGOTHER, Content-Type
Access-Control-Max-Age: 86400
...
```

Pre-flight request



```
Access-Control-Request-Headers: X-PINGOTHER, Content-type
```

```
const xhr = new XMLHttpRequest();
xhr.open('POST', 'https://bar.other/resources/post-here/');
xhr.setRequestHeader('X-PINGOTHER', 'pingpong');
xhr.setRequestHeader('Content-Type', 'application/xml');
xhr.onreadystatechange = handler;
xhr.send('<person><name>Arun</name></person>');
```

```
.1 204 No Content
ttp://foo.example
OST, GET, OPTIONS
HER, Content-Type
ol-Max-Age: 86400
...
```

```
POST /doc HTTP/1.1
X-PINGOTHER: pingpong
Content-Type: text/xml; charset=UTF-8
Origin: http://foo.example
...
```

```
                         HTTP/1.1 200 OK
Access-Control-Allow-Origin: http://foo.example
          Vary: Accept-Encoding, Origin
                  Content-Encoding: gzip
                     Content-Length: 235
```