AJAX with Fetch

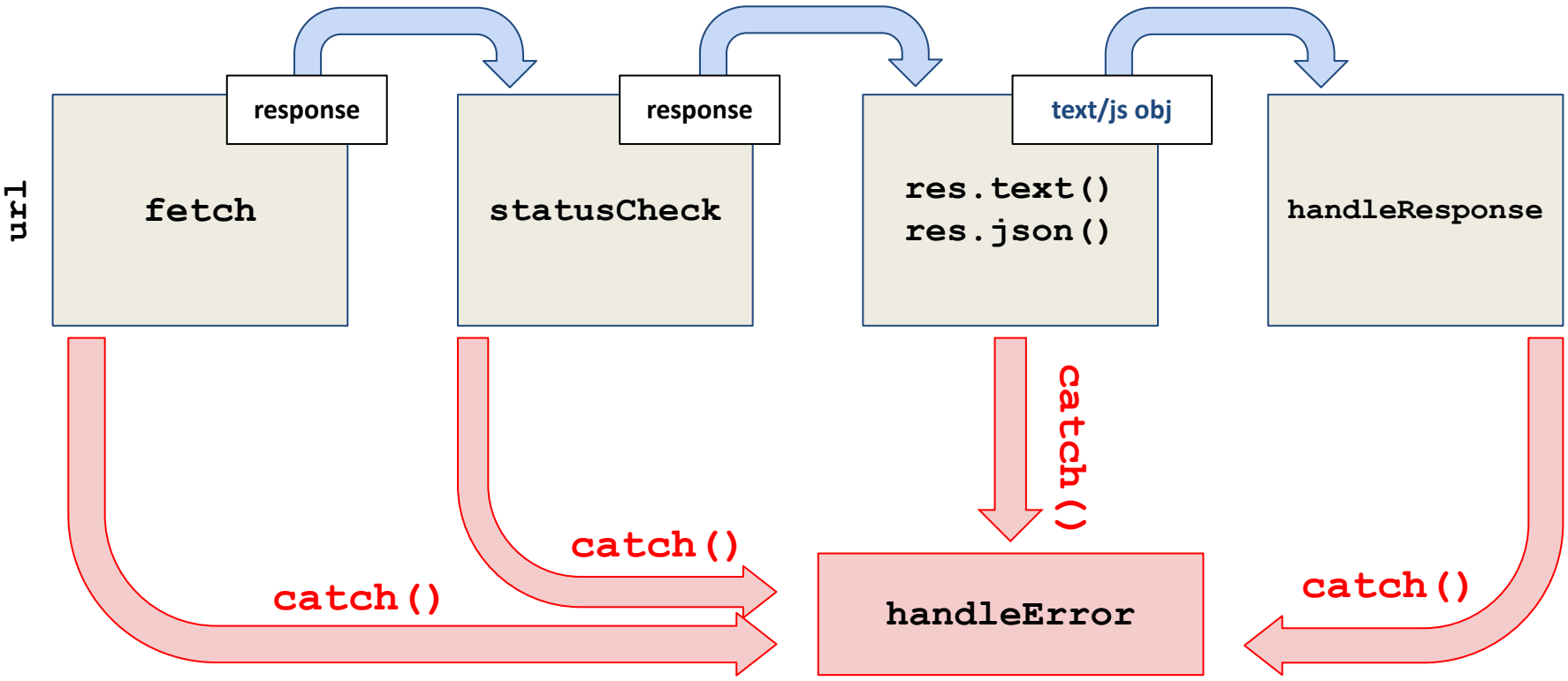
fetch API

- 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

```
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/

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));
```

Esempio GitHub

```
const url = `http://pw.netgroup.uniroma2.it/docenti/${docente}.json`
console.log(url)
fetch(url)
  .then(res => res.json())
  .then(json_data => {
    const url_user = `https://api.github.com/users/${json_data.name}`
    console.log(url_user)
    return fetch(url_user)
  })
  .then(res=>res.json())
  .then(user_data=>{
```

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();
```

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();
```

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')
```

Esempio GitHub con async/await

```
const getInfoProf = async function(docente){
  try {

    const url = `http://pw.netgroup.uniroma2.it/docenti/${docente}.json`

    const res = await fetch(url);
    const json_data = await res.json()
    const url_user = `https://api.github.com/users/${json_data.name}`

    const res_gh = await fetch(url_user)
    const user_data = await res_gh.json()
```

CORS

CORS: cross-origin HTTP request

- un client richiede una risorsa di un differente dominio, protocollo o porta.

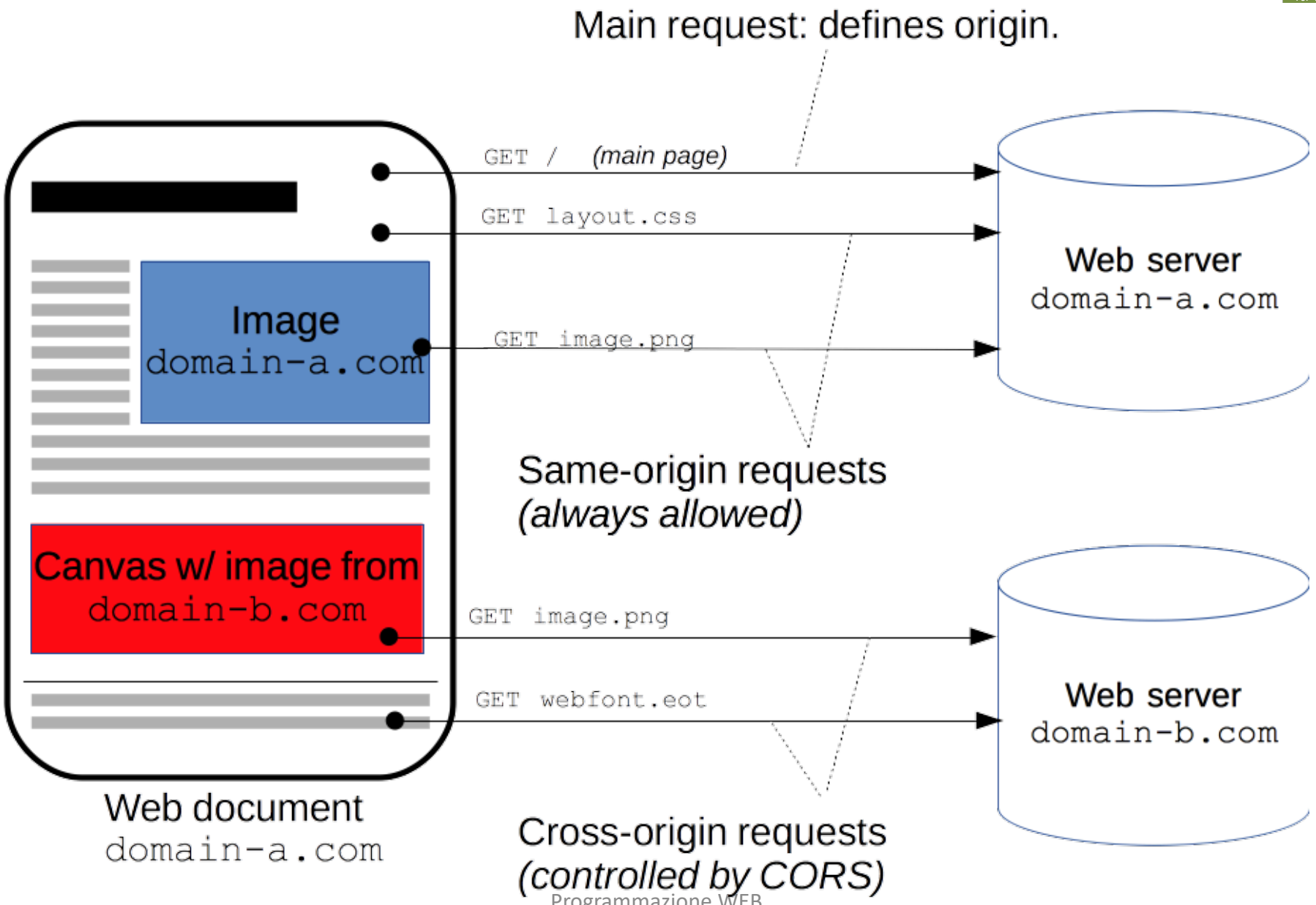
- Esempio
  - una web application con dominio X non può richiedere una risorsa ad un dominio Y tramite AJAX request se Y non ha abilitato il CORS.

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

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


Main request: defines origin.
GET / (main page)
GET layout.css
GET image.png
Image
domain-a.com
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)

Cross Origin Resource Sharing

- Come fare se vogliamo espressamente permettere il resource sharing tra due siti diversi? → CORS
  - Standard W3C per condividere risorse tra domini diversi
  - Prevede richiesta di autorizzazione (client) e autorizzazione (server)

- Viene implementato inviando degli header HTTP in req/resp
  - Richieste Semplici
  - Richieste Preflight

Simple request

- Metodi Ammessi
  - GET
  - HEAD
  - POST
- Header pemessi
  - Accept
  - Accept-Language
  - Content-Language
  - Content-Type (but note the additional requirements below)
  - …
- 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();
```

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>');
```

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>');
```