Università di Genova

JavaScript (5)

Marina Ribaudo, marina.ribaudo@unige.it

Ajax: modello di esecuzione

Browser

Richiesta HTTP

Risposta HTTP con
pagina HTML + JS

Web server

Database
Php script
Immagini
...

Ajax: modello di esecuzione

Browser

Richiesta HTTP

Risposta HTTP con
pagina HTML + JS

Web server

Database
Php script
Immagini
...

Applicazione
Browser

Richiesta HTTP

Dati
XML
Testo
JSON
Immagini
.....

Web server

Database
Php script
Immagini
...

Ajax: modello di esecuzione

Browser

Richiesta HTTP

Risposta HTTP con
pagina HTML + JS

Applicazione
Browser

Richiesta HTTP

Dati
XML
Testo
JSON
Immagini
.....

Database
Php script
Immagini
...

Web server

Database
Php script
Immagini
...

Web server

NON CAMBIA NIENTE

Ajax: modello di esecuzione

Browser

Richiesta HTTP

Risposta HTTP con
pagina HTML + JS

Database
Php script
Immagini
...

Web server

Applicazione
Browser

Richiesta HTTP

CAMBIA IL TIPO DI DATO

Dati
XML
Testo
JSON
Immagini
.....

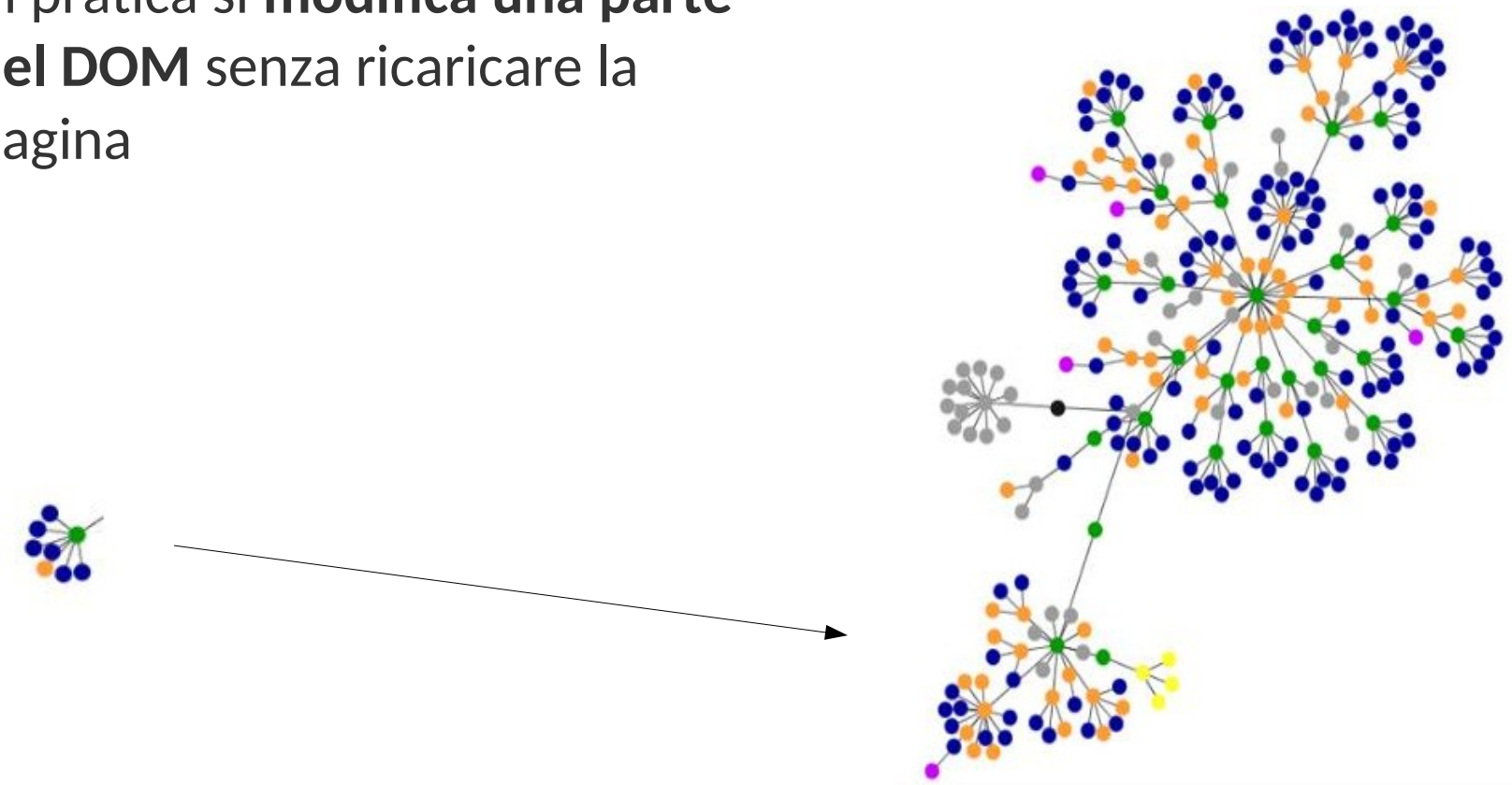
Database
Php script
Immagini
...

Web server

NON CAMBIA NIENTE

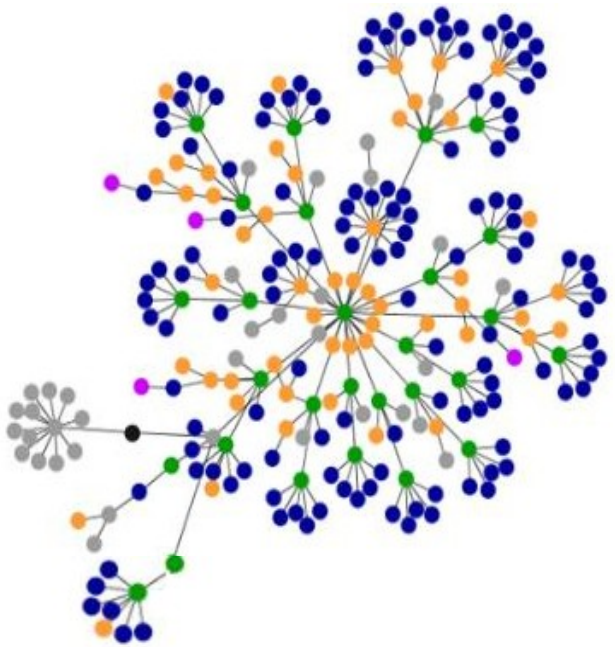
Ajax: com'è possibile?

In pratica si **modifica una parte del DOM** senza ricaricare la pagina



Ajax: com'è possibile?

In pratica si **modifica una parte del DOM** senza ricaricare la pagina



Idea di base

- In **HTTP** il modello classico è quello della **pagina** (statica o dinamica)

- Si può lavorare a livello di **porzione di pagina** usando JavaScript ma non si può interagire con dati remoti perché si lavora sul client

Idea di base

- Sfruttando il DOM di JavaScript e le **chiamate Ajax** si possono modificare in modo **asincrono** o **sincrono** piccole porzioni della pagina corrente senza ricaricarla completamente

- Veniva usato per esempio da Google Maps, Gmail, YouTube, …
- Nelle Google Maps nuove sezioni della mappa sono caricate dal server quando servono, senza richiedere un refresh della pagina

Idea di base


```
<script>
  function update(id) {
    let el = document.getElementById(id);
    el.innerHTML = <chiamata ad una funzione remota di un server>
  }
</script>
```

Risultato di una chiamata che modifica l'elemento **id** nella pagina corrente

XMLHttpRequest

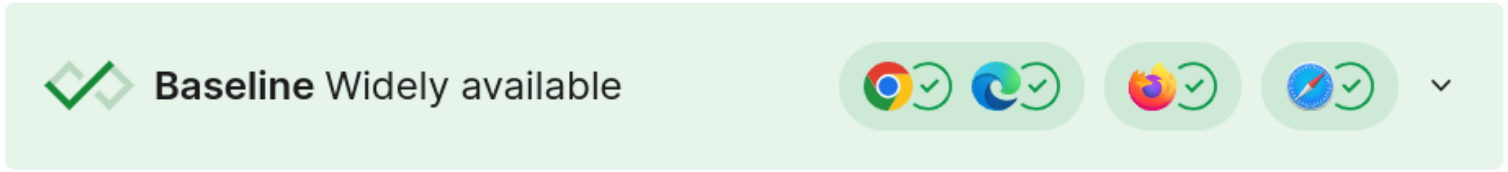
- **XMLHttpRequest** è l'**oggetto JavaScript** messo a disposizione dal browser per **Ajax**
- Permette di eseguire **chiamate HTTP** tramite JavaScript
- Nonostante il nome, non si limita ad essere usato con XML ma accetta anche altri formati, per es. **JSON**

Vedi:
https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest
http://en.wikipedia.org/wiki/XMLHttpRequest

API Fetch

- Oggi Ajax è ancora usato, ma esiste una API più moderna per le chiamate remote

Fetch API



Note: This feature is available in Web Workers.

The Fetch API provides an interface for fetching resources (including across the network). It is a more powerful and flexible replacement for `XMLHttpRequest`.

API Fetch

- “JavaScript is a **single-threaded programming language** which means only one thing can happen at a time. ... That's where asynchronous JavaScript comes into play. Using asynchronous JavaScript (such as callbacks, promises, and async/await), you can perform long network requests without blocking the main thread.”

API Fetch

- L'API Fetch permette di fare **chiamate HTTP** (asincrone) verso endpoint remoti
- Fornisce il metodo **fetch()** che permette di richiedere una risorsa dalla rete, e ritorna una **Promise** che verrà “realizzata” quando la risorsa è disponibile

Promise

- Una **Promise** è un oggetto che rappresenta il risultato di un'operazione asincrona
- Può trovarsi in uno dei tre stati
  - **pending** (in attesa)
  - **fulfilled** (completata con successo)
  - **rejected** (completata con errore)

Promise

- Se la Promise viene **completata con successo** viene restituito un oggetto di tipo **Response** che contiene il **risultato**, che può essere
  - la **risorsa cercata**
  - la **ragione per cui tale risorsa non viene restituita** (per esempio c'è stato un errore a livello di rete oppure la risorsa non esiste)

    Nota: una Promise ha successo anche se non restituisce la risorsa cercata, bisogna controllare i valori ritornati, per esempio Response.**ok**

Promise

- Se la Promise **fallisce** viene restituito un oggetto di tipo **Error** che rappresenta l'errore che si è verificato durante l'esecuzione dell'operazione asincrona
- Si può capire l’errore che si è verificato andando a leggere il valore di Error.**message**

API Fetch: richiesta

- Per la **richiesta** si usa il metodo

  fetch(url [, options])

  - **url**, unico parametro obbligatorio, contiene l'indirizzo della risorsa cercata
  - **options** è un oggetto opzionale che consente di configurare la richiesta specificando il metodo usato, gli header della richiesta, l'eventuale body

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

API Fetch: risposta

- Quando viene ricevuta la risposta si possono usare proprietà e metodi dell'oggetto Response
  - Response.**ok**

    true/false (true se lo stato è nell'intervallo 200-299, altrimenti false)
  - Response.**status**

    codice della risposta (200 in caso di successo)
  - Response.**statusText**

    testo corrispondente al codice (per esempio OK per il codice 200)

API Fetch: risposta

- Response.Body.**text()**
  - Legge uno stream di dati fino alla fine e **ritorna una Promise** che conterrà una **stringa**
- Response.Body.**json()**
  - Legge uno stream di dati fino alla fine e **ritorna una Promise** che conterrà un **oggetto JSON**
- Response.Body.**blob()**
  - Legge uno stream di dati fino alla fine e **ritorna una Promise** che conterrà un **blob**

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

API Fetch: esempio


```
fetch(url)
 .then( function(response) { response.json(); })
 .then( function(data) { console.log(data); })
 .catch( function(error) { console.log(error) });
```

API Fetch: esempio


```
fetch(url)
 .then(response => response.json())
 .then(data => console.log(data))
 .catch(error =>  console.log(error));
```

Arrow function

API Fetch: esempio

- Si possono mandare dati in POST

```
fetch(url,
{   method: "post",
    headers: { "Content-type": "application/x-www-form-urlencoded" },
    body: "email=" + usermail
})
```

API Fetch: esempio

- Si possono mandare dati in POST

```
fetch(url,
{   method: "post",
    headers: { "Content-type": "application/x-www-form-urlencoded" },
    body: "email=" + usermail
}).then(function (response) {
    /* code for Response object*/
    return response.text();
})
```

API Fetch: esempio

- Si possono mandare dati in POST

```
fetch(url,
{   method: "post",
    headers: { "Content-type": "application/x-www-form-urlencoded" },
    body: "email=" + usermail
}).then(function (response) {
    /* code for Response object*/
    return response.text();
}).then(function (data) {
    /* code for data */
});
```

API Fetch: esempio

- Si possono anche mandare dati in JSON

```
fetch(url,
{   method: "post",
    headers: { "Content-type": "application/json" },
    body: JSON.stringify({
      email: “m.r@mail.it”
})
}).then(function (response) {
    /* code for Response object*/
    return response.text();
}).then(function (data) {
    /* code for data */
});
```

REST

- Con **Representational State Transfer** (REST) si intende un tipo di architettura software per i **sistemi distribuiti**

- Il termine è stato introdotto nel 2000 nella tesi di dottorato di Roy Fielding, uno dei principali autori delle specifiche del protocollo HTTP

- http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

REST

“...The World Wide Web represents the largest implementation of a system conforming to the REST architectural style.

REST-style architectures conventionally consist of clients and servers. Clients initiate requests to servers; servers process requests and return appropriate responses. Requests and responses are built around the transfer of representations of resources.”

REST

- I servizi vengono invocati usando i **metodi** del protocollo **HTTP**
- Generalmente i risultati sono restituiti in **XML** o **JSON**
- Il **parsing** dei risultati può essere fatto lato client con JavaScript oppure lato server

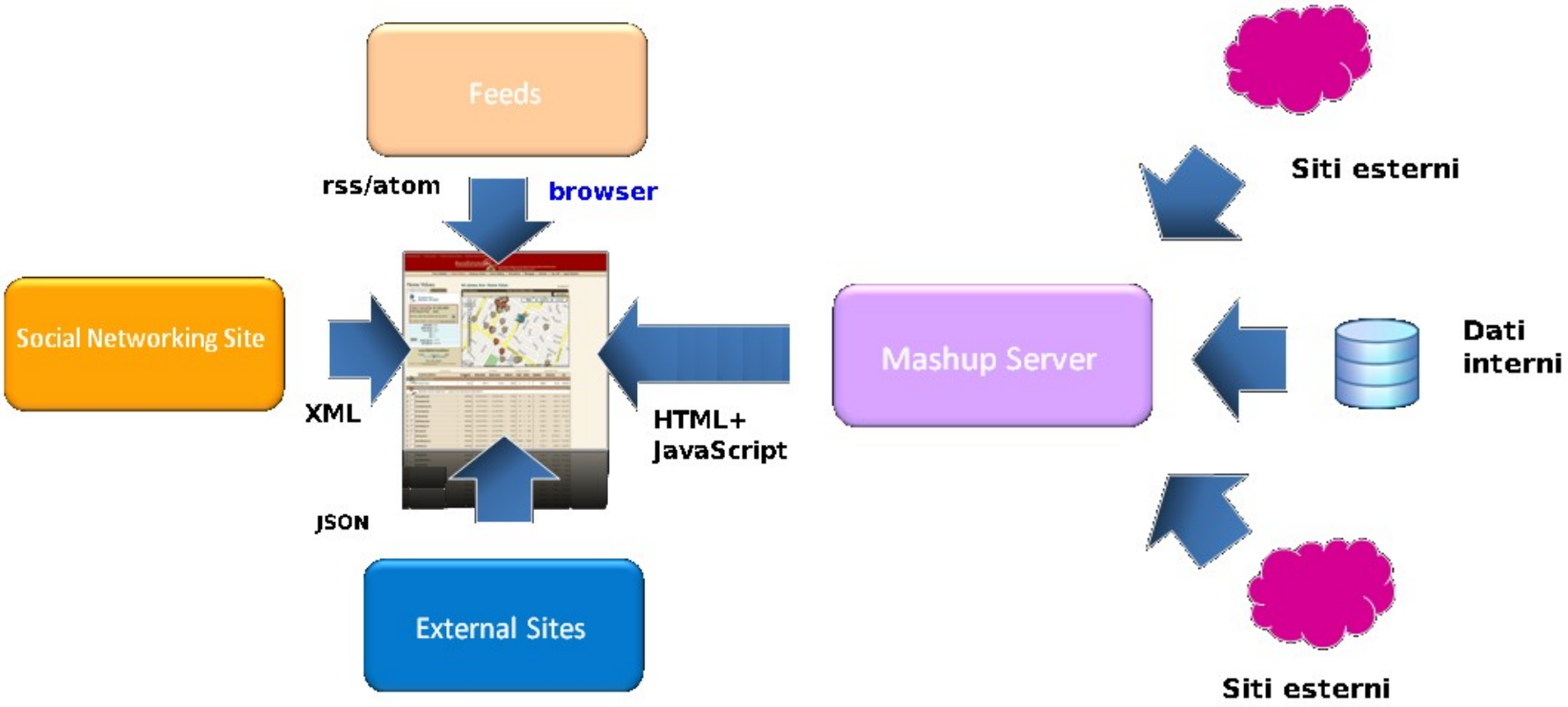
Mashup

- L'uso di servizi REST permette di realizzare dei **mashup**

  "... A mashup, in web development, is a web page, or web application, that uses and combines data, presentation or functionality from two or more sources to create new services. The term implies easy, fast integration, frequently using open application programming interfaces (API) and data sources to produce enriched results that were not necessarily the original reason for producing the raw source data."

Fonte: http://en.wikipedia.org/wiki/Mashup_web_application_hybrid

Mashup


Feeds
rss/atom
browser
Social Networking Site
XML
HTML+
JavaScript
JSON
External Sites
Mashup Server
Siti esterni
Dati
interni
Siti esterni

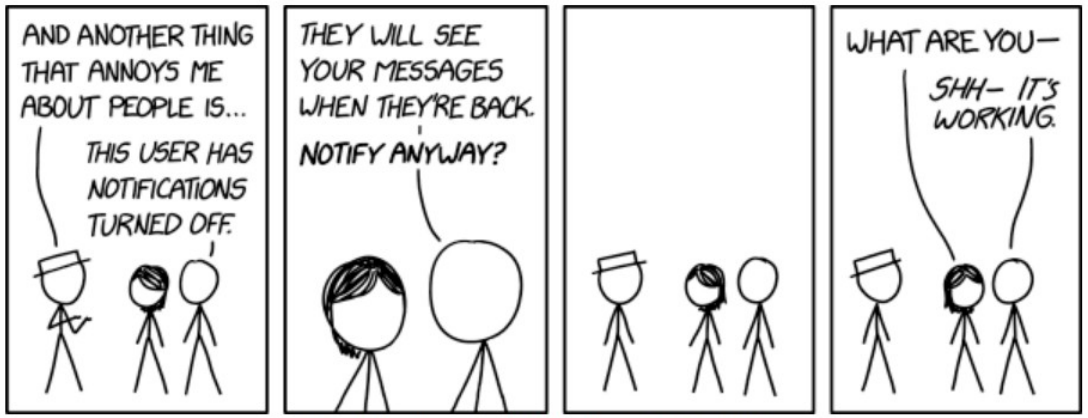
Esempio di servizio REST

- https://xkcd.com/json.html

Click on green buttons to see xkcd comics

Get last xkcd comic | Get random xkcd comic

more info at https://xkcd.com/json.html

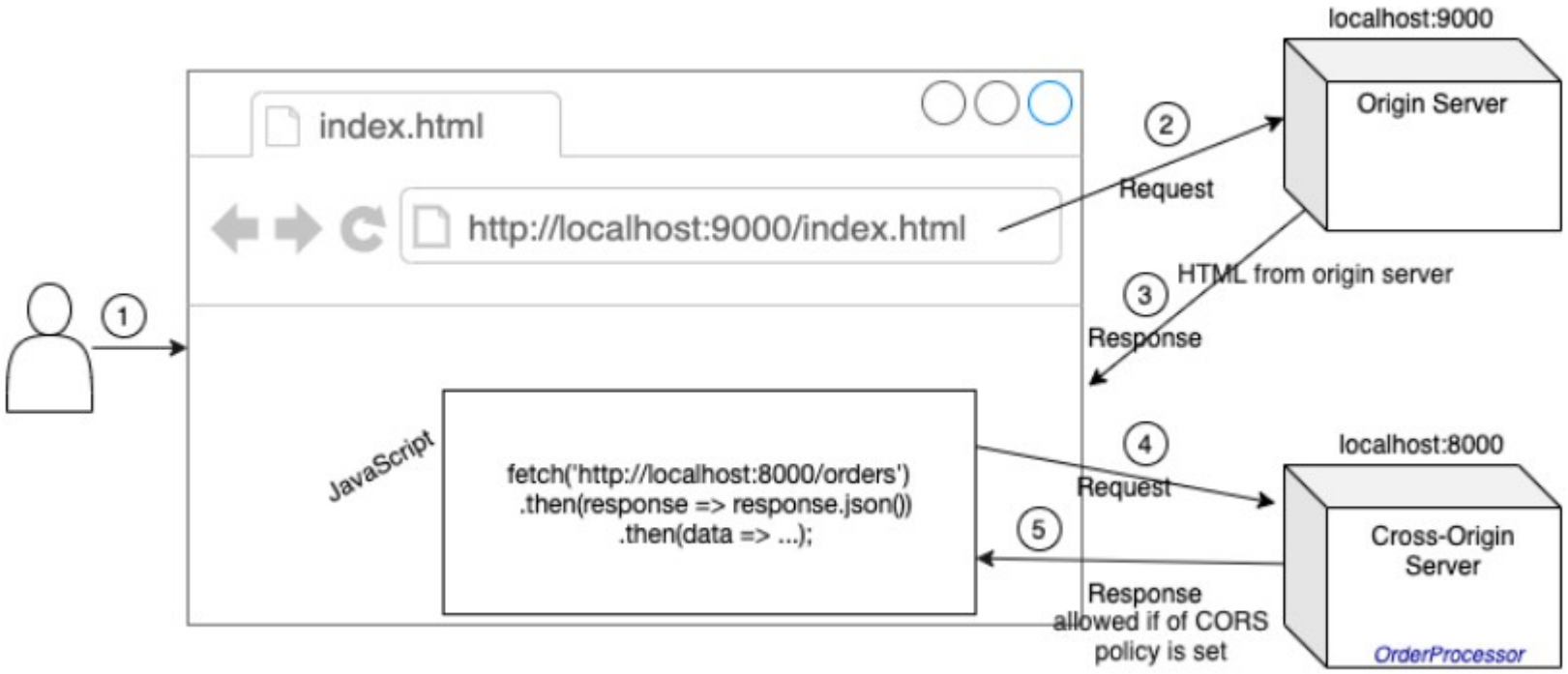


Notifications (published on 15-12-2021)

Cross-Origin Resource Sharing

- *Cross-Origin Resource Sharing (CORS) is a **protocol** that enables scripts running on a browser client to interact with resources from a different origin*
- *This is useful because, thanks to the **same-origin policy** followed by XMLHttpRequest and Fetch, **JavaScript can only make calls to URLs that live on the same origin as the location where the script is running***
- *For example, if a JavaScript app wishes to make a Fetch call to an API running on a different domain, it would be blocked from doing so thanks to the same-origin policy*

Cross-Origin Resource Sharing



Cross-Origin Resource Sharing

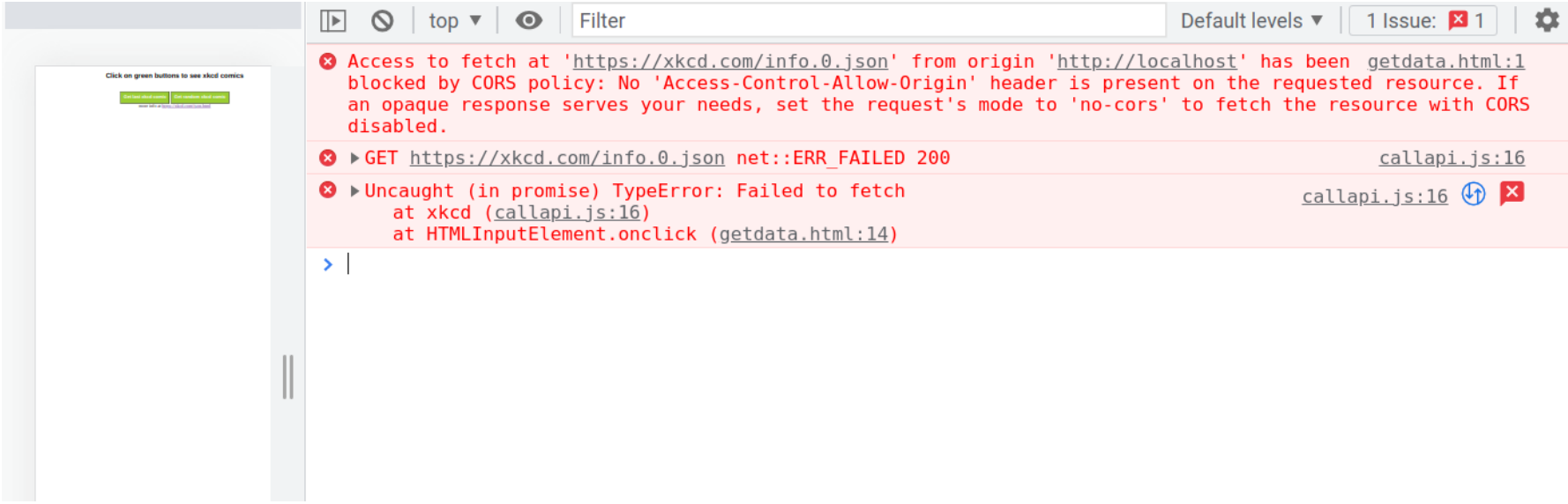
- Origin: [scheme]://[hostname]:[port]

URLs being Matched	Same-Origin or Cross-Origin	Reason
http://www.mydomain.com/targetPage.html	Same-Origin	same scheme, host, and port
http://www.mydomain.com/subpage/targetPage.html	Same-Origin	same scheme, host, and port
https://www.mydomain.com/targetPage.html	Cross-Origin	same host but different scheme and port
http://pg.mydomain.com/targetPage.html	Cross-Origin	different host
http://www.mydomain.com:8080/targetPage.html	Cross-Origin	different port

Cross-Origin Resource Sharing

- Per superare questo problema il **server** dovrebbe ritornare indietro speciali header tra cui il più importante è
  - **Access-Control-Allow-Origin: *** oppure
  - Access-Control-Allow-Origin: https://example.com
- I browser usano questo header per capire se la chiamata fatta con JavaScript può continuare o meno

Cross-Origin Resource Sharing



Se non controlli il server...

- ...e non puoi abilitare header lato server, puoi scrivere un programma “proxy” che
  - Riceve la chiamata Fetch
  - La inoltra al servizio remoto
  - Riceve la risposta
  - La restituisce al browser che ha fatto la richiesta

Fetch e REST