Modulo 2

Web Tier

JavaScript parte 2 – Day 3

Presentato da:
Vitale Esca
IBM Client Innovation Center - Italy

22/11/2023

IBM Client Innovation Center
Italy

Agenda Day 3

1. Form, a simple case!
2. Event listners in breve ed Exceptions
3. HTTP vs HTTPS
4. Cosa sono le API

IBM Client Innovation Center
Italy

Agenda Day 3

5 Chiamate REST

6 Request asincrone e sincrone

7 Da JSON a oggetto JS e viceversa

8 GET/POST con esempi

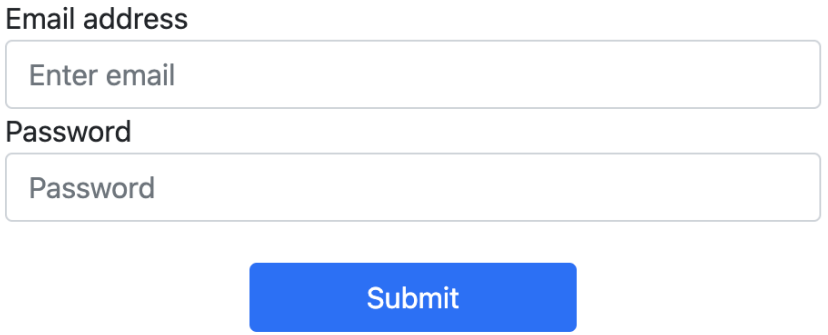
IBM Client Innovation Center
Italy

Day 3 – Per recuperare...

Il grande dimenticato... Il form

I form sono molto utili per la creazione di moduli di registrazione, moduli di contatto, sondaggi, questionari e molto altro ancora. Permettono, tramite controlli JavaScript di validare il contenuto del form ed eventualmente segnalare all'utente eventuali problemi in modo reattivo. Questo è un esempio usando bootsrap

```
<body>
  <div id="form-container" class="container">
    <div class="row">
      <div class="col">
        <form>
          <div class="form-group">
            <label for="exampleInputEmail1">Email address</label>
            <input type="email" class="form-control" id="exampleInputEmail1" placeholder="Enter email">
          </div>
          <div class="form-group">
            <label for="exampleInputPassword1">Password</label>
            <input type="password" class="form-control" id="exampleInputPassword1" placeholder="Password">
          </div>
          <div class="d-grid gap-2 col-6 mx-auto">
            <button id="submit" type="submit" class="btn btn-primary">Submit</button>
          </div>
        </form>
      </div>
    </div>
  </div>
<script src="./form.js"></script>
</body>
```



Per approfondimenti: https://getbootstrap.com/docs/5.0/forms/overview/

Il grande dimenticato... Il form

Lato JavaScript è necessario assegnare al pulsante con id submit (o ciò che avete definito in fase di scrittura del form) un eventListner in ascolto, specificando l'azione che deve scatenare la procedura del form e l'evento che ha scatenato il listner...

Nota bene la riga event.preventDefault(); utilizzata per impedire l'azione predefinita associata all'evento che si sta gestendo e gestirlo lato JavaScript.

```
document.getElementById("submit").addEventListener("click", function(event) {
    event.preventDefault(); // Previene l'invio del modulo
    let emailInput = document.getElementById("exampleInputEmail1").value;
    let passwordInput = document.getElementById("exampleInputPassword1").value;
    console.log("Email: " + emailInput);
    console.log("Password: " + passwordInput);
});
```

Per approfondimenti: https://getbootstrap.com/docs/5.0/forms/overview/

Day 3 – Per recuperare...

Cosa sono gli EventListner?

Gli event listner in JavaScript sono un modo per gestire gli eventi in modo dinamico e personalizzato.

Permettono di **eseguire una funzione** da noi definita **quando si verifica un evento specifico**, come ad esempio il **click** di un pulsante, il **caricamento di una pagina**, l'invio di un modulo ecc...
Gli event listner possono essere registrati su **moltissimi elementi diversi** a seconda del tipo di evento che si vuole gestire.

E' possibile registrare più event listner sullo stesso elemento.

Per approfondimenti: https://www.w3schools.com/js/js_htmldom_eventlistener.asp

Eccezioni in JavaScript

Anche in JavaScript, così come ad esempio in Java, è possible gestire le eccezioni sollevate nel codice.

Come in Java si utilizza il costrutto try/catch/finally.

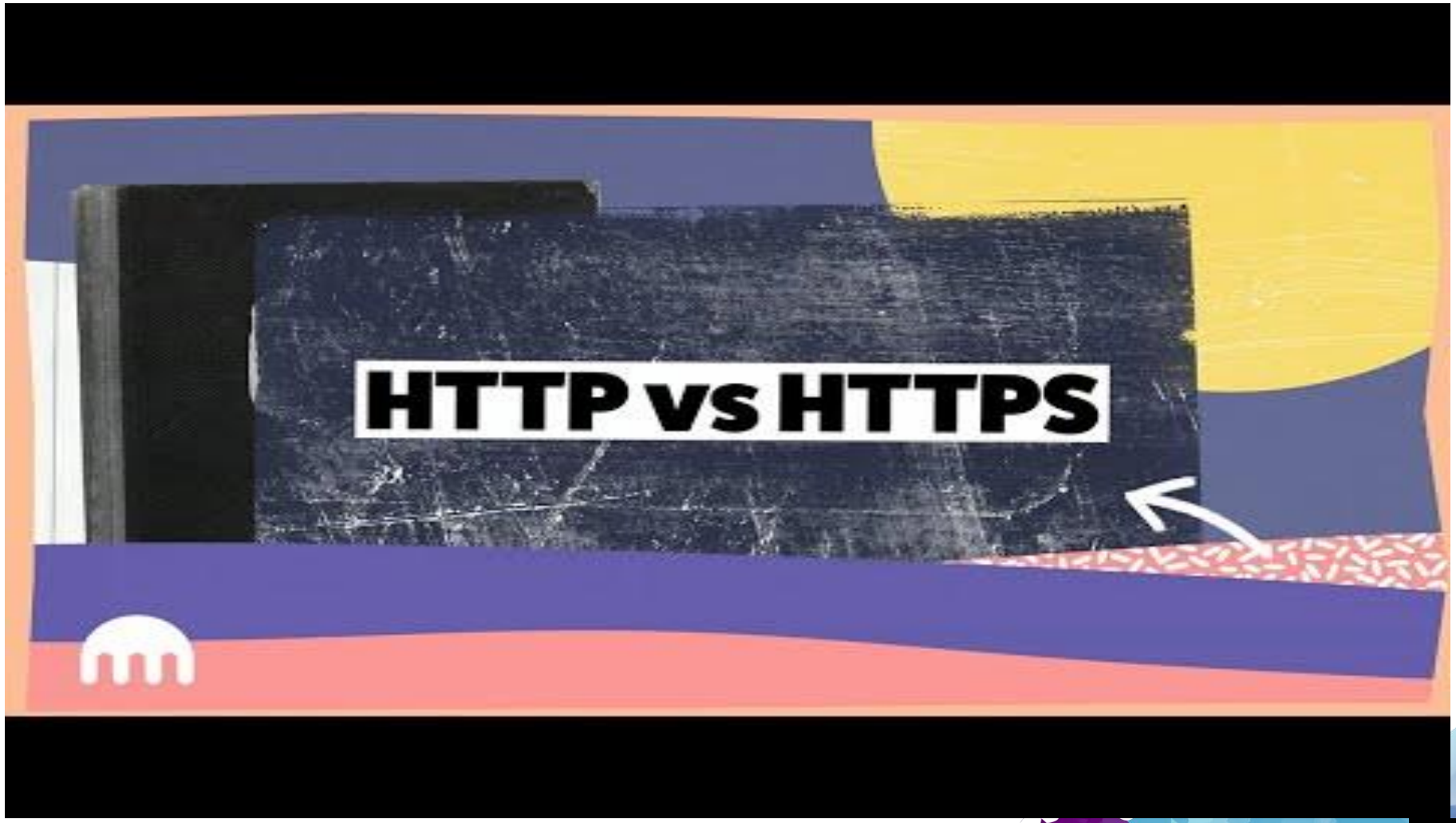
In questo esempio viene aperto un alert in caso di eccezione.

```
function haltAnException() {
    try {
      // Codice che potrebbe generare un'eccezione
      let x = y + 1; // Questa riga genera un'eccezione, poiché y non è definito
    } catch (error) {
      // Gestione dell'eccezione
      alert("Si è verificata un'eccezione: " + error.message);
    }
}
```

NB. Non bisogna dimenticarsi del finally, per eseguire delle istruzioni del blocco del codice, indipendentemente se vi si è verificata una exception o meno... come in Java!

Approfondimento: https://www.w3schools.com/js/js_errors.asp

HTTP vs HTTPS



IBM Client Innovation Center
Italy

Cosa sono le **A**pplication**P**rogrammin**I**nterface

Cosa sono i servizi REST?

REST è l'acronimo di **Re**presentational **S**tate **T**ransfer.

I servizi web REST offrono una connessione a delle risorse. L'output varia per contenuto e formato.

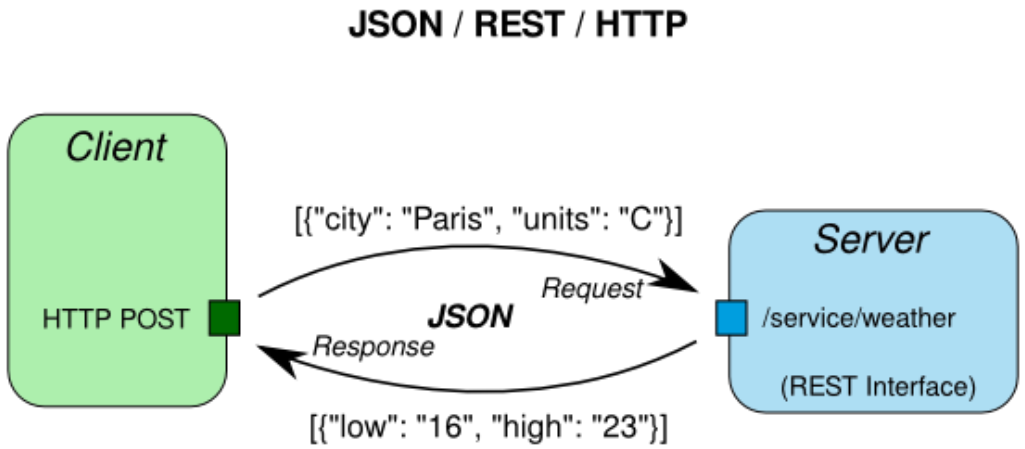
I servizi web REST sono identificati da **URI**.

Ad esempio:

URL: https://www.example.com/book/add

Dominio: https://www.example.com

URI: /book/add



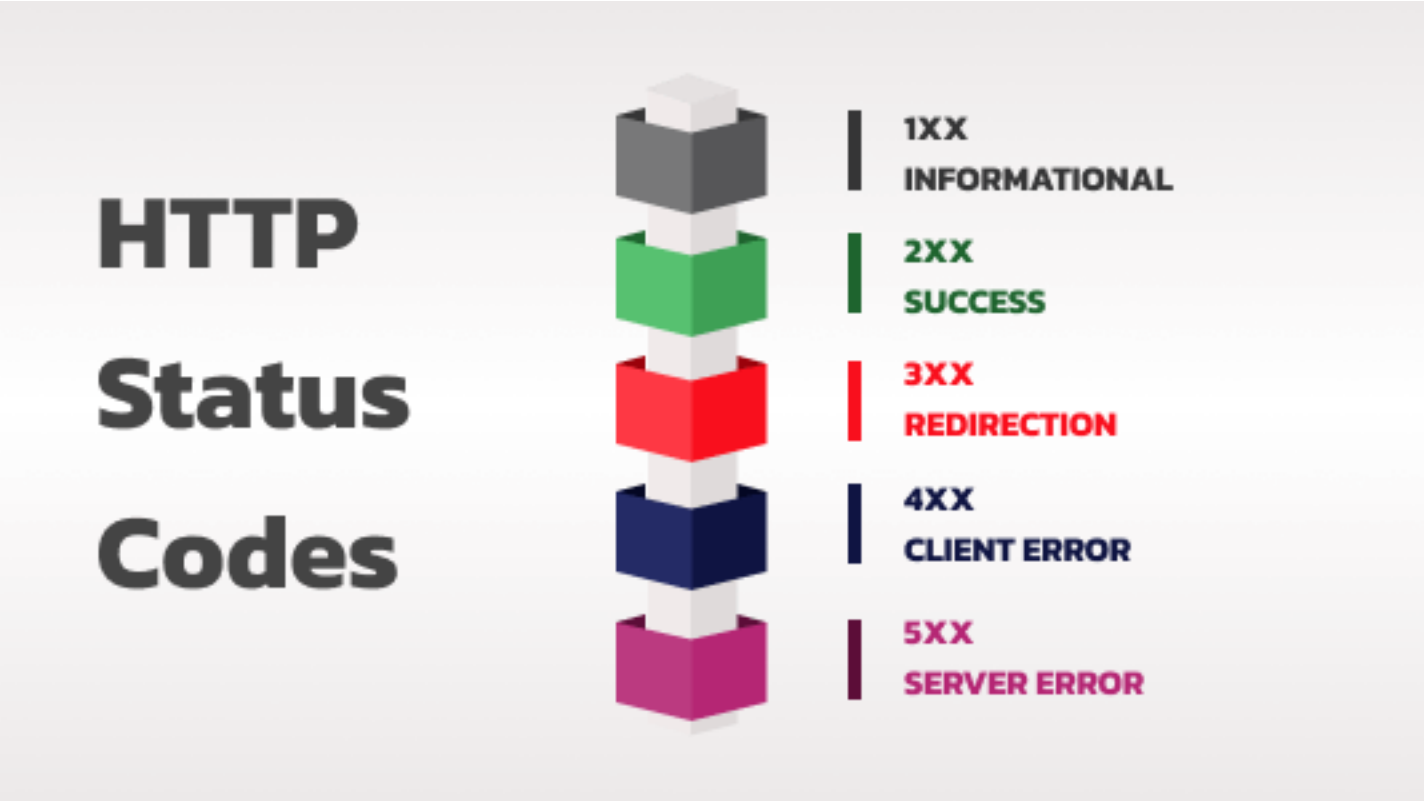
Per approfondimenti: https://www.ibm.com/it-it/cloud/learn/rest-apis

Metodi servizi REST

Nell'invocazione di un servizio REST è necessario definire il **metodo**, tra:

Metodo	Descrizione
GET	Utilizzato per recuperare le risorse
POST	Utilizzato per aggiungere/inviare delle risorse
DELETE	Utilizzato per eliminare le risorse
PUT	Utilizzato per aggiornare le risorse

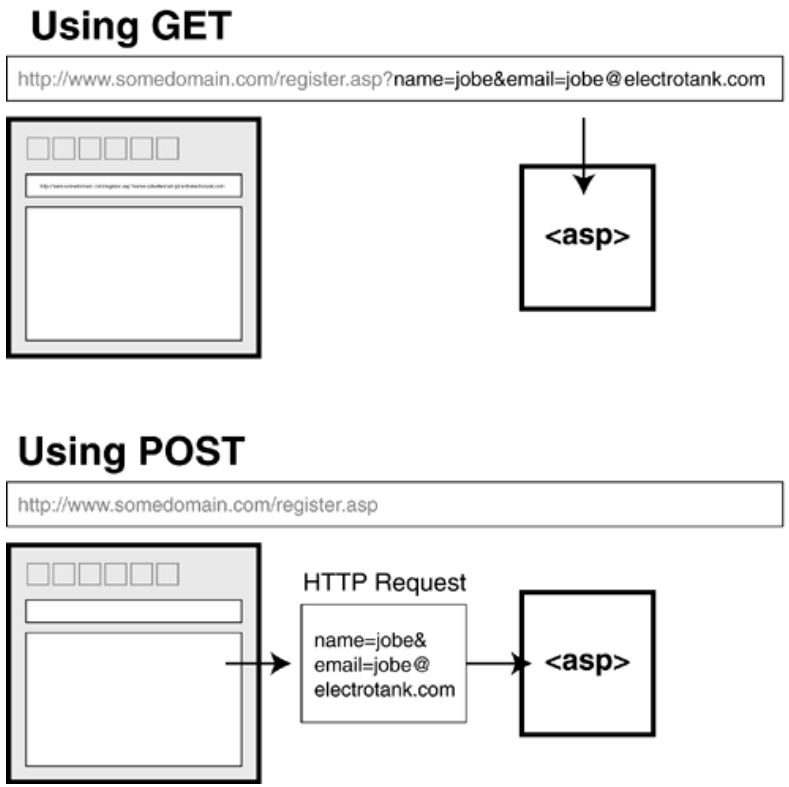
HTTP status codes

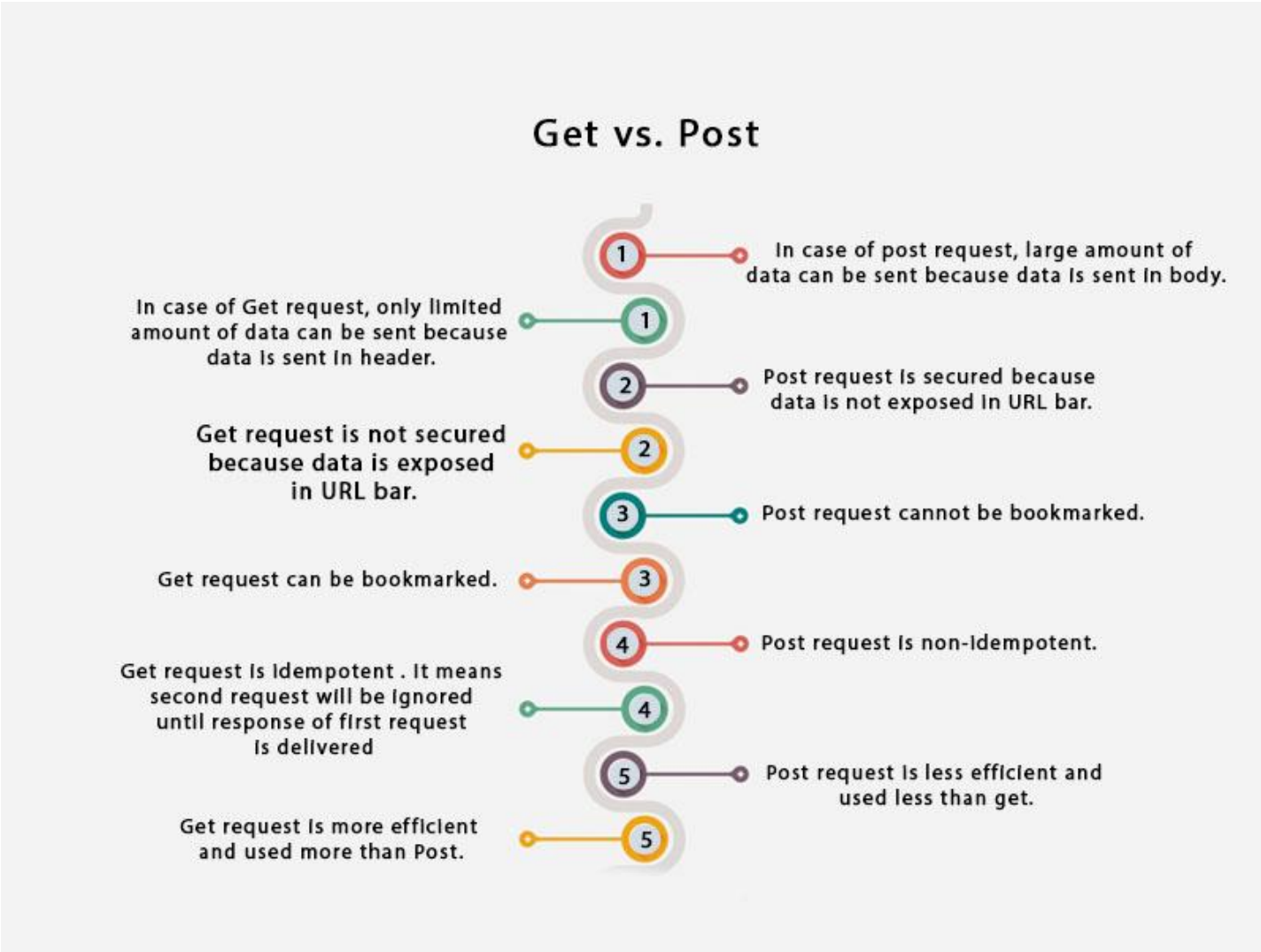


Approfondimenti: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

Metodi servizi REST

Questa distinzione è invalicabile? ***NO***





Servizi REST: un esempio

Documentazione:

https://imgflip.com/api

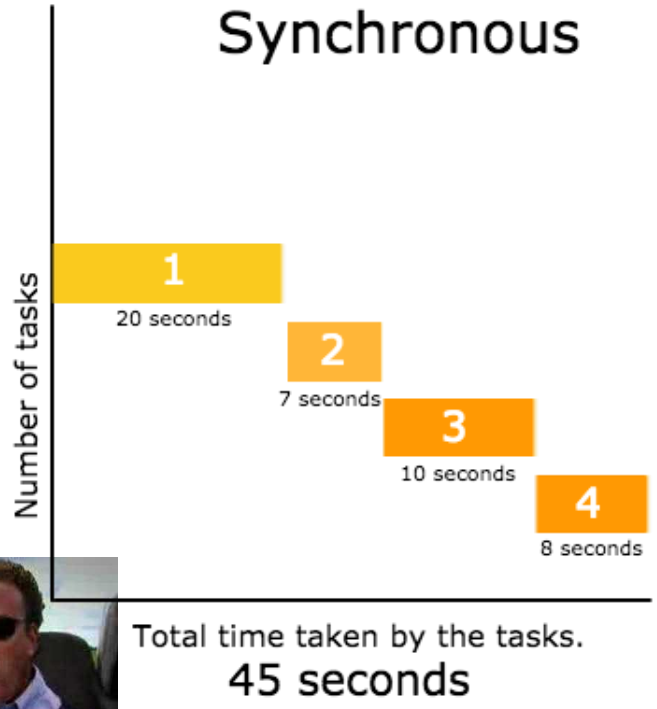
Queste API REST consentono di *generare meme*.

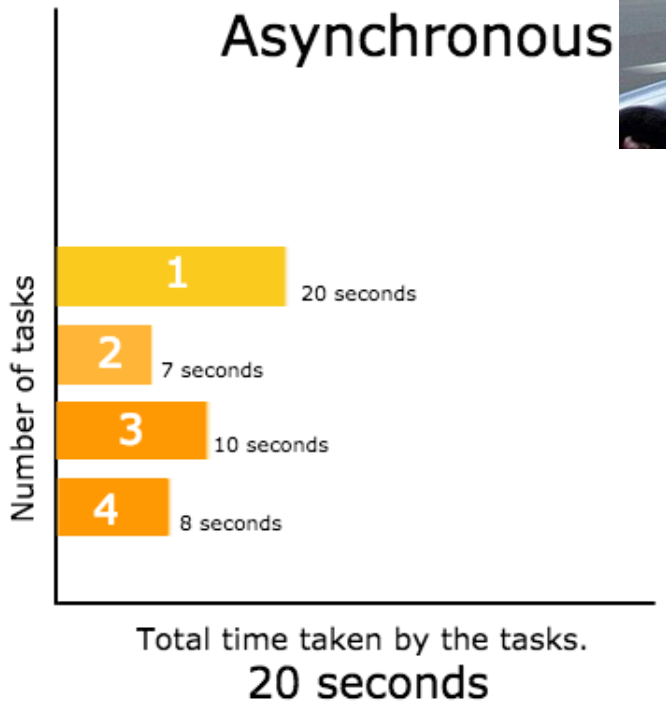
Con un browser o postman possiamo provarle in modo semplice

Scaricare qui Postman:
https://www.postman.com/

IBM Client Innovation Center
Italy

Request sincrone vs asincrone





Request sincrona

```
var request = new XMLHttpRequest()
request.open('GET', 'https://api.imgflip.com/get_memes', false) //"false" = chiamata sincrona!
request.send(null)

if (request.status === 200) {
  console.log(request.response)
}
```

XMLHttpRequest() è un oggetto in JavaScript che viene utilizzato per effettuare richieste HTTP(s). Può essere utilizzato sia in modo sincrono che asincrono.

La differenza principale tra una richiesta sincrona e una asincrona sta nel modo in cui viene gestito il flusso di esecuzione del codice durante l'elaborazione della richiesta.

In una richiesta XMLHttpRequest sincrona, il codice viene bloccato fino a quando la richiesta non viene completata. Questo significa che il codice successivo non verrà eseguito fino a quando la richiesta non viene completata, il che può causare un blocco dell'interfaccia utente se la richiesta impiega molto tempo.

Per approfondimenti: https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/Synchronous_and_Asynchronous_Requests

Funzione asincrona

```
var xhr = new XMLHttpRequest()
xhr.open('GET', 'https://api.imgflip.com/get_memes', true) //"true" = chiamata asincrona!
xhr.onload = function (e) {
  if (xhr.readyState === 4) {
    if (xhr.status === 200) {
      console.log(xhr.response)
    } else {
      console.error(xhr.statusText)
    }
  }
}
xhr.onerror = function (e) {
  console.error(xhr.statusText)
}
xhr.send(null)
```

Per approfondimenti: https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/Synchronous_and_Asynchronous_Requests

Da stringa JSON a Oggetto JS

Quando si invoca un'API che restituisce un JSON è possibile operare sulla response trasformando il JSON in un oggetto JavaScript tramite la funzione ***JSON.parse****(variabileJson)*.

In questo modo sarà possibile accedere ad ogni singola variabile del JSON ricevuto:

```
> let responseObj = JSON.parse('{"id":"181913649","name":"Drake Hotline Bling","url":"https://i.imgflip.com/30b1gx.jpg","width":1200,"height":1200,"box_count":2,"captions":0}')
< undefined
> responseObj
< ▼{id: '181913649', name: 'Drake Hotline Bling', url: 'https://i.imgflip.com/30b1gx.jpg', width: 1200, height: 1200, …}
    box_count: 2
    captions: 0
    height: 1200
    id: "181913649"
    name: "Drake Hotline Bling"
    url: "https://i.imgflip.com/30b1gx.jpg"
    width: 1200
  ▶[[Prototype]]: Object
> JSON.stringify(responseObj)
< '{"id":"181913649","name":"Drake Hotline Bling","url":"https://i.imgflip.com/30b1gx.jpg","width":1200,"height":1200,"box_count":2,"captions":0}'
```

E per trasformare un oggetto JS in JSON? Basta scrivere JSON.stringify(object)

Approfondimento su JSON.parse():
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/parse

Come ottenere dati tramite GET

Per ottenere dati da un endpoint REST di tipo GET **non parametrizzato**, è sufficiente invocare il servizio senza parametri:

```
function get() {
  try{
    let request = new XMLHttpRequest();
    request.open('GET', 'https://api.imgflip.com/get_memes', false);
    request.send(null)

    if(request.status === 200){
      let memeArray = JSON.parse(request.response).data.memes;
    }else{
      throw new Error('Response status != 200: ' + request.status);
    }
  }catch(error){
    alert('Endpoint remoto - GET - /get_meme non disponibile');
  }
}
```

A seconda della documentazione del servizio, **la response potrebbe essere un elemento solo o una lista di elementi**
Tramite la funzione JSON.parse() possiamo trasformare la risposta JSON in oggetti JS

IBM Client Innovation Center
Italy

Come ottenere dati tramite GET

Per ottenere dati da un endpoint REST di tipo GET **parametrizzato**, è necessario costuirsi l'url parametrizzato:

```
function getParametrizzata() {
  try{
    let request = new XMLHttpRequest();
    let requestParams = 'template_id=131087935&username=test_test_test&password=test_test_test&text0=ciao&text1=mondo';
    request.open('GET', 'https://api.imgflip.com/caption_image?' + requestParams, false);
    request.send();

    if(request.status === 200){
      let memeArray = JSON.parse(request.response).data.memes;
    }else{
      throw new Error('Response status != 200: ' + request.status);
    }
  }catch(error){
    alert('Endpoint remoto - GET - /get_meme non disponibile');
  }
}
```

Ovviamente l'url può essere costruito tramite una funzione da noi definita, oppure passando gli elementi come parametri della function

IBM Client Innovation Center
Italy

Esempi di POST

Per inviare una request POST ad un edpoint, è necessario specificare il tipo di request come POST e passare **l'oggetto JS trasformato in JSON** come request:

```
function postRequestJson() {
  try{
    let request = new XMLHttpRequest();
    request.open('POST', 'https://api.imgflip.com/caption_image', false);

    let requestObject = {
      template_id : '131087935',
      username : 'test_test_test',
      password : 'test_test_test',
      text0 : 'ciao',
      text1 : 'mondo',
    };

    request.send(JSON.stringify(requestObject))

    if(request.status === 200){
      let memeArray = JSON.parse(request.response).data.memes;
    }else{
      throw new Error('Response status != 200: ' + request.status);
    }
  }catch(error){
    alert('Endpoint remoto - POST - /caption_image non disponibile');
  }
}
```

E' il metodo **più utilizzato e sicuro per le chiamate POST**, ovviamente è necessario consultare la **documentazione** del servizio per capire quali parametri accetta!
Ad esempio l'API in oggetto non accetterà questo tipo di chiamata!!

IBM Client Innovation Center
Italy

Esempi di POST – Un caso particolare...

Alcuni servizi, anche se POST, potrebbero non accettare un JSON come ad esempio il nostro servizio /capturing_image, il quale accetta come servizio POST i parametri nell'Header:

```
function postRequestHeader() {
  try{
    let request = new XMLHttpRequest();
    request.open('POST', 'https://api.imgflip.com/caption_image', false);
    let requestParams = 'template_id=131087935&username=test_test_test&password=test_test_test&text0=ciao&text1=mondo';
    request.setRequestHeader('Content-type', 'application/x-www-form-urlencoded');
    request.send(requestParams);

    if(request.status === 200){
      let memeArray = JSON.parse(request.response).data.memes;
    }else{
      throw new Error('Response status != 200: ' + request.status);
    }
  }catch(error){
    alert('Endpoint remoto - GET - /get_meme non disponibile');
  }
}
```

Come già detto, è possibile parametrizzare dinamicamente l'url di richiesta rendendo ad esempio la funzione postRequestHeader, parametrizzata!

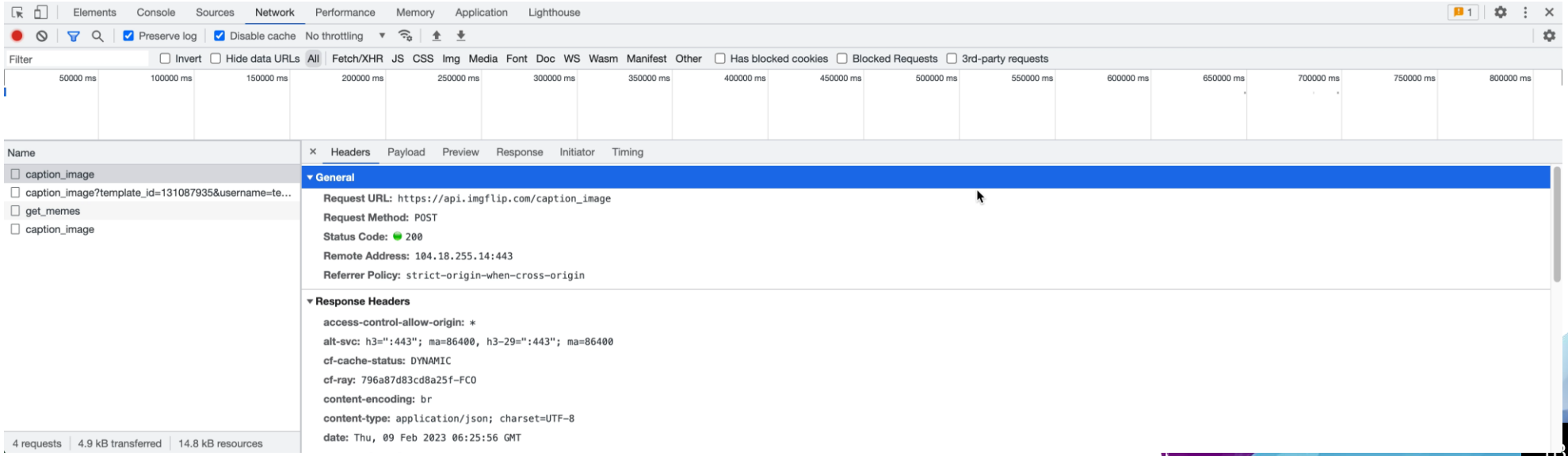
IBM Client Innovation Center
Italy

Debug delle chiamate REST con il nostro browser

Per poter capire se il nostro servizio sta invocando correttamente il servizio e come, è sufficiente utilizzare la funzione “ispeziona” del browser. Come mostrato nei scorsi giorni, basta andare nella pagina web del nostro servizio, premere il tasto destro e cliccare su “ispeziona”.

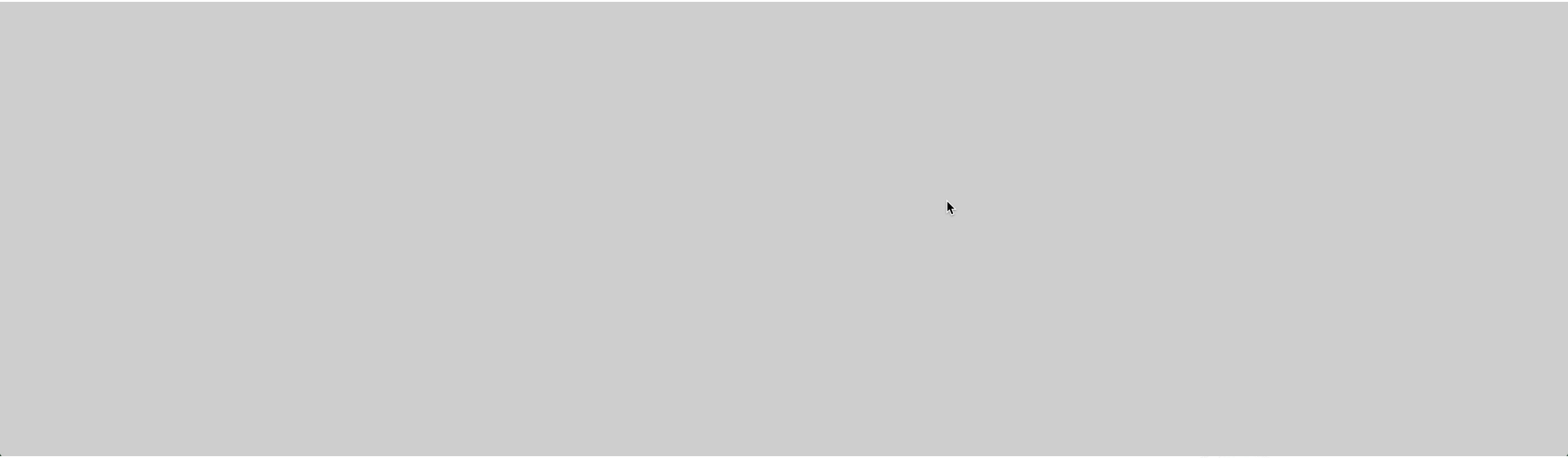
Nella sezione network è possible visualizzare tutte le chiamate effettuate ai servizi remote, con I dettagli quali ad esempio:

- *Url chiamato*
- *Status code ricevuto*
- *Dati di request inviati*
- *Dati di response ricevuti*



Debug delle chiamate REST con il nostro browser

Esempio video:



Promise

Analizziamo le seguenti tre entità:

1.Codice Produttore: che fa qualcosa e richiede tempo. Ad esempio, del codice che carica i dati su una rete.

2.Codice Consumatore: che vuole il risultato del “codice produttore” una volta pronto.

3.Una ***promise*** è uno speciale oggetto JavaScript che collega insieme il codice produttore e il codice consumatore. Una promise impiega tutto il tempo necessario per produrre il risultato promesso e la poi lo rende disponibile a tutto il codice sottoscritto quando è pronto.

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

// resolve runs the first function in .then
promise.then(
  result => alert(result), // shows "done!" after 1 second
  error => alert(error) // doesn't run
);
```

IBM Client Innovation Center
Italy

Fetch API

Le API Fetch sono una versione semplificata e facile da usare di XMLHttpRequest per recuperare risorso in modo asincrono. La differenza principale è che Fetch utilizza le promise.

```
fetch('https://api.github.com/users/manishmshiva', {
  method: "GET",
  headers: {"Content-type": "application/json;charset=UTF-8"}
})
.then(response => response.json())
.then(json => console.log(json));
.catch(err => console.log(err));
```

ES6 - ECMAScript 2015

Con il rilascio della versione ECMAScript 2015, conosciuta come ES6, vengono lanciate importanti novità che rivoluzionano il linguaggio.

Per approfondimenti: https://www.w3schools.com/react/react_es6.asp

ES6 - ECMAScript 2015

•**Keyword “let”**, permette di modificare il valore di una variabile all’interno di uno spazio di codice delimitato per poi dare nuovamente alla variabile il valore precedente.

•**Keyword “const”**, molto simile alla keyword “let” ma il valore assegnato alla variabile non può cambiare all’interno delle parentesi graffe in cui è dichiarato. Esternamente ottiene di nuovo il valore iniziale.

•Nuovi metodi sugli array: **Map()** e **Filter();**

Per approfondimenti: https://www.w3schools.com/react/react_es6.asp

IBM Client Innovation Center
Italy

ES6 - ECMAScript 2015

•Le nuove **arrow functions** che permettono la dichiarazione di funzioni in modo più efficace risparmiando righe di codice.

Before:

```
hello = function() {
  return "Hello World!";
}
```

With Arrow Function:

```
hello = () => {
  return "Hello World!";
}
```

Per approfondimenti: https://www.w3schools.com/react/react_es6.asp

ES6 - ECMAScript 2015

•**Spread operator** per copiare rapidamente il contenuto di un array o oggetto esistente in un altro array o oggetto;

```
const numbersOne = [1, 2, 3];
const numbersTwo = [4, 5, 6];
const numbersCombined = [...numbersOne, ...numbersTwo];
```

Per approfondimenti: https://www.w3schools.com/react/react_es6.asp

IBM Client Innovation Center
Italy

ES6 - ECMAScript 2015

- **Desctructuring**, per estrarre solo quello di cui abbiamo bisogno da un array o un oggetto.

```
const vehicleOne = {
  brand: 'Ford',
  model: 'Mustang',
  type: 'car',
  year: 2021,
  color: 'red',
  registration: {
    city: 'Houston',
    state: 'Texas',
    country: 'USA'
  }
}

const { model, registration: { state } } = vehicleOne
const message = 'My ' + model + ' is registered in ' + state + '.';
```

Per approfondimenti: https://www.w3schools.com/react/react_es6.asp

IBM Client Innovation Center
Italy

ES6 - ECMAScript 2015

- **Operatore ternario**, un modo compatto di scrivere un if/else

```
let age = 18
let text = (age < 18) ? "Minorenne":"Maggiorenne";
```

Per approfondimenti: https://www.w3schools.com/react/react_es6.asp

IBM Client Innovation Center
Italy

Cos’è TypeScript?

TypeScript è un superset di JavaScript open-source sviluppato da Microsoft che aggiunge tipi, classi, interfacce e moduli al JavaScript tradizionale.

Essendo un superset, TypeScript risulta essere totalmente compatibile con la semantica e la sintassi JavaScript.

Il vantaggio principale dell’utilizzo di TypeScript è sicuramente la tipizzazione che permette di documentare meglio la forma di un oggetto e di verificarne il corretto funzionamento.


```typescript
interface Account {
  id: number
  displayName: string
  version: 1
}

function welcome(user: Account) {
  console.log(user.id)
}
```

```typescript
type Result = "pass" | "fail"

function verify(result: Result) {
  if (result === "pass") {
    console.log("Passed")
  } else {
    console.log("Failed")
  }
}
```

Per approfondimenti: https://www.w3schools.com/typescript/

Una breve guida al debugging con DevTools

Per approfondimenti: https://www.w3schools.com/howto/howto_js_redirect_webpage.asp

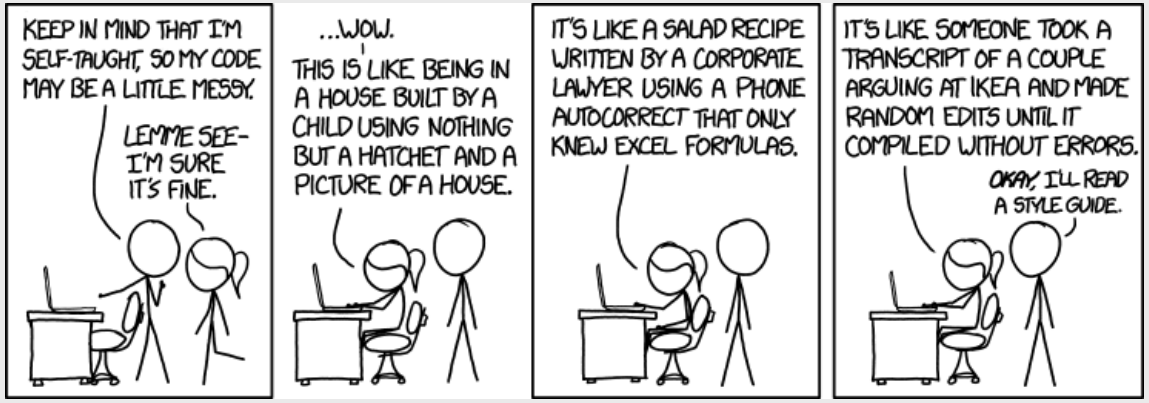
Ecco alcuni repository da consultare...

33 concetti di JavaScript

https://github.com/leonardomso/33-js-concepts

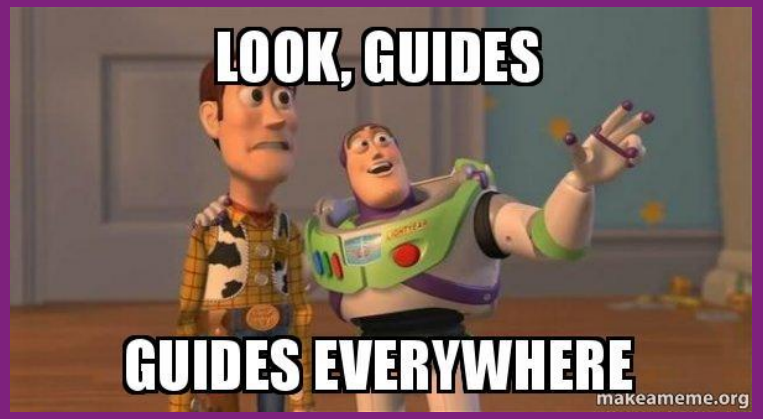
Clean code

https://github.com/ryanmcdermott/clean-code-javascript



JavaScript spiegato da Airbnb

https://github.com/airbnb/javascript




QUIZ TIME
SPOILER: NON È LA CIPOLLA

Experience.
Create.
Inspire.

Thank You

Vitale Esca

IBM Client Innovation Center
Italy