

# Object: oggetti in Javascript

Ogni cosa che non è un tipo primitivo (number, string, boolean null, undefined) in javascript è un **object**, un oggetto.

Per definire un oggetto abbiamo una sintassi molt semplice:

```
const car = {  
  }  
}
```

Ci sono molti altri modi per definire un **object** in Javascript ma per adesso a noi basta questo, più avanti vedremo altri modi per definirlo.

## Proprietà degli oggetti

Ogni oggetto può avere delle proprietà ed ognuna di loro è composta da una **label** (o anche **key**) associata ad un **valore (value)**:

```
const car = {  
  color: 'orange',  
  type: 'diesel';  
}
```

la **label (key)** potrebbe essere anche un valore tra apici esempio

```
const car = {  
  color: 'orange',  
  type: 'diesel';  
  'new color': 'red'  
}
```

Le **label** possono essere qualsiasi stringa, ma attenzione ai **caratteri speciali**: se volessi includere un carattere non valido come nome variabile nel nome della proprietà, avrei dovuto usare le “ ” intorno ad esso:

**I caratteri di nome variabile non validi** includono **spazi**, **trattini** e altri **caratteri speciali**.

Accedere ad una proprietà è semplice

- dot notation

```
car.color
```

- uso delle [ ]

```
car['color']
```

```
car['new color']
```

accedere ad una proprietà che non esiste ritorna un **undefined**

Un oggetto può avere oggetti innestati, esempio

```
const car = {  
  brand: {  
    name: 'Ford'  
  },  
  color: 'blue'  
}
```

e per accedere ad una sua proprietà doveri accedere

```
car.brand.name
```

oppure

```
car['brand'] ['name']
```

E' possibile aggiornare il valore di una proprietà di un oggetto in qualsiasi momento:

```
car.color = 'yellow'  
car['new color'] = 'green'
```

E' possibile anche aggiungere a run time nuove proprietà non inserite durante la sua definizione, esempio

```
car.model = 'Fiesta'
```

Infine è possibile eliminare una proprietà in maniera molto semplice

```
delete car.model
```

## I metodi degli oggetti

E' possibile aggiungere anche delle funzionalità ai nostri oggetti in maniera molto semplice:

```
const car = {  
  brand: 'Ford',  
  model: 'Fiesta',  
  start: function() {  
    console.log('Started')  
  }  
}
```

ed invocare il metodo semplicemente richiamandolo dall'oggetto stesso

```
car.start()
```

All'interno di un metodo definito usando una sintassi **function() {}** abbiamo accesso all'istanza oggetto facendo riferimento a **this**.

Nell'esempio seguente, abbiamo accesso ai valori delle proprietà del marchio e del modello utilizzando **this.brand** e **this.model**:

```
const car = {  
  brand: 'Ford',  
  model: 'Fiesta',  
  start: function() {  
    console.log('Started ${this.brand} ${this.model}')  
  }  
}  
car.start()
```

E' possibile anche utilizzare le **arrow function** dentro gli oggetti ma non è preferibile perché non potremmo utilizzare il **this**.

Infine possiamo anche passare dei dati alle nostre funzioni come facciamo con delle **regular function**:

```
const car = {  
  brand: 'Ford',  
  model: 'Fiesta',  
  goTo: function(destination) {  
    console.log('Going to ${destination}')  
  }  
}  
car.goTo('Rome')
```

# Json - Ajax - Fetch

Un **Json** è un **oggetto** che potremmo definire anche come un **protocollo** per l'archiviazione “momentanea” dei dati (**Non è un DataBase**).

**JSON** sta per **JavaScript Object Notation**, una notazione che facilita molto la programmazione soprattutto nella trasformazione di informazioni in **oggetti** più “**leggibili**” lato linguaggio di programmazione.

## JSON

### JavaScript Object Notation

Un file JSON può contenere:

stringhe

```
{ "nome": "Mario" }
```

numeri

```
{ "anni": 50 }
```

array

```
{ "studenti": [ "Mario", "Carla", "Pino" ] }
```

oggetti

```
{ "utente": { "nome": "Mario", "anni": 50,
```

operatori booleani

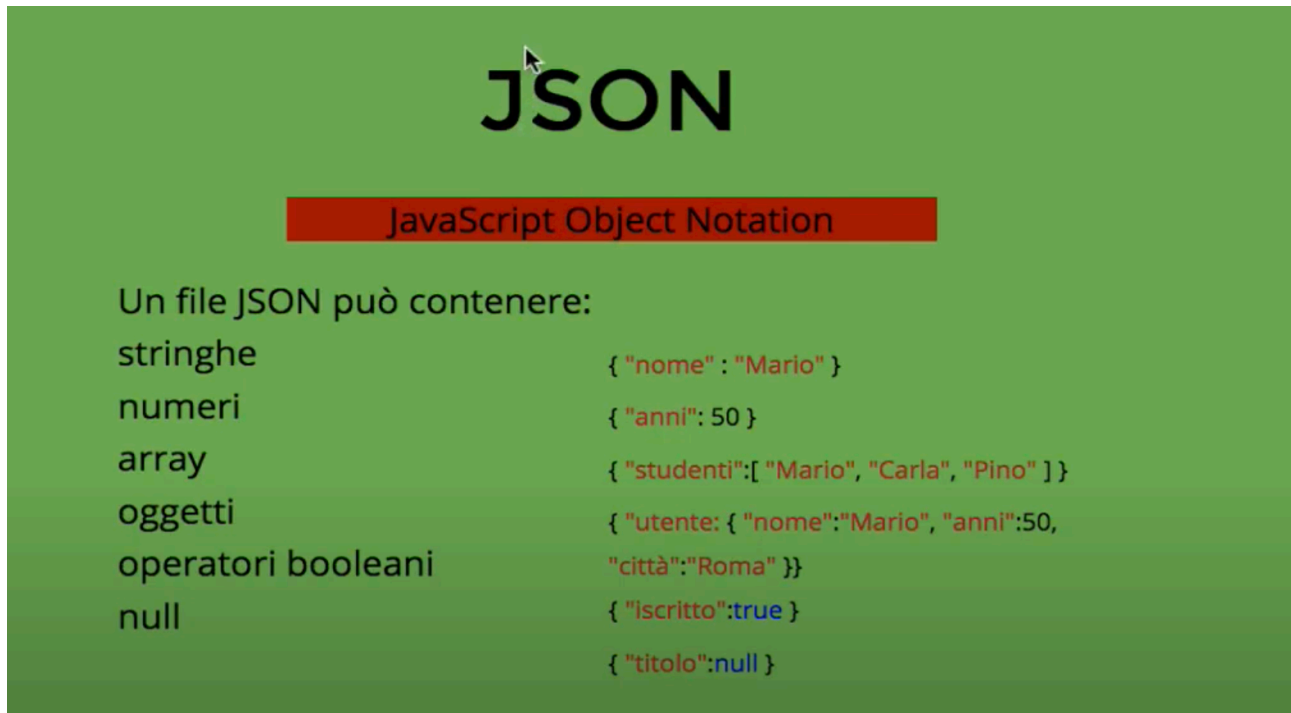
```
"città": "Roma" } }
```

null

```
{ "iscritto": true }
```

```
{ "titolo": null }
```

# Json più da vicino



**JSON**

JavaScript Object Notation

Un file JSON può contenere:

stringhe	<code>{ "nome": "Mario" }</code>
numeri	<code>{ "anni": 50 }</code>
array	<code>{ "studenti": [ "Mario", "Carla", "Pino" ] }</code>
oggetti	<code>{ "utente": { "nome": "Mario", "anni": 50,</code>
operatori booleani	<code>"città": "Roma" } }</code>
null	<code>{ "iscritto": true }</code>
	<code>{ "titolo": null }</code>

Un oggetto **JSON** è spesso utilizzato per la ricezione e l'invio di dati attraverso una tipica **formattazione** che adesso vedremo meglio e più nel dettaglio.

Quello che non abbiamo ancora visto è che **JSON** ci offre anche la possibilità di creare degli oggetti complessi con **oggetti** annidati al suo interno oppure **array di oggetti** che a loro volta possono avere altri array e/o oggetti innestati oppure oggetti con innestati altri array di oggetti.

Sembra complesso ma non lo è, tutto sta nel saperlo leggere, creare e validare quando serve.

Per la **creazione** e la **validazione** vedremo che sulla rete si trovano strumenti che ci possono aiutare.

# Creiamo un JSON complesso

Come abbiamo già visto un oggetto **JSON** ha una struttura che parte sempre da una `{` e termina con una `}` e sappiamo che gli array si definiscono a partire da `[]`

```
{
  "users": [
    {
      "id": 1,
      "name": "Valentino Romano",
      "age": "46",
      "email": "vr@cosmopolis.biz",
      "pets": [
        "cat",
        "dog"
      ]
    },
    {
      "id": 1,
      "name": "Leanne Graham",
      "age": "40",
      "email": "sincere@april.biz",
      "pets": [
        "rabbit",
        "dog"
      ]
    }
  ]
}
```

Una volta scritto questo semplice **Json** controllarlo e validarle è molto semplice ma se avessimo un Json molto più grande e complesso validarle potrebbe diventare un lavoro molto complesso, ecco perché ci aiuteremo con un servizio on-line <https://jsonlint.com/> che ci permetterà di controllare ed eventualmente correggere il json.

# Primo Esempio - JSON

Creiamo il nostro primo piccolissimo progetto e vediamo come **codificare e decodificare un oggetto ovvero passare da un oggetto ad un JSON e viceversa**.

Vedremo l'uso di **Stringify()** e **ParseJson()** i due metodi che ci aiuteranno nel nostro intento. Questa cosa più avanti non servirà.

```
<body>
```

```
<h1>Hello devs</h1>
```

```
<div id="demo">Qui vedrai i tuoi dati Json</div>
```

```
<ul id="ajax">Dati ancora non pronti ....</ul>
```

```
<ul id="fetch">Dati in fetch ancora non pronti ....</ul>
```

```
<script>
```

```
//creiamo un oggetto Javascript
```

```
let user = { name: "Valentino", surname: "Romano", age: 46 };
```

```
// vediamo il metodo stringify
```

```
let userJson = JSON.stringify(user);
```

```
console.log("Stringify (mi crea un JSON)= " + userJson);
```

```
// abbiamo trasformato il nostro oggetto in un Json
```

```
// se adesso volessi fare al contrario ovvero cioè da Json ad un oggetto?
```

```
// prendo un oggetto JSON e lo parso con JSON.parse(obj)
```

```
let userObjParse = JSON.parse('{"name":"valentino","surname":"romano","age":46}');
```

```
console.log('Obj: userObjParse =', userObjParse);
```

```
// se partiamo dal nostro oggetto userJson reso JSON con stringify allora:
```

```
let userObj = JSON.parse(userJson);
```

```
console.log('Obj: userObj =', userObj);
```

```
console.log('Obj: userObj.name =', userObj.name);
```

```
// mostriamolo a video
```

```
document.getElementById("demo").innerHTML = `
```

```
    ${userObj.name} ${userObj.surname} , ${userObj.age} anni
```

```
`
```

```
</script>
```

```
</body>
```

# AJAX

## Cos'è AJAX

Ajax non è una libreria né un framework. Si tratta di un **pattern di programmazione** o anche tecnica di sviluppo per la realizzazione di applicazioni web interattive.

**AJAX** è l'acronimo di:

**A**syncrhonous  
**J**avascript  
**A**nd  
**X**ML

Partiamo dal concetto di chiamate

### Cosa sono le chiamate asincrone nel caso di AJAX?

Sono delle chiamate che sfruttano l'**HTTP design pattern REST-API** (**R**epresentetional **S**tate **T**ransfer ), quindi siamo in un' architettura **Client-Server** dove le chiamate ad un **server** sono fatte in modo tale da non dover attendere in modo **“frezzato”** la risposta.

Oltre all'aspetto Asincrono ci sono poi le operazioni di **“refresh”** dei dati appena arrivati dalla chiamata Asincrona, su una determinata porzione di della pagina HTML.

Una particolarità di **Ajax** sta nel fatto che ogni richiesta fatta al server o una modifica fatta al **DOM HTML** della nostra pagina, è fatta in modo da **evitare il refresh dell'intera pagina**.

**AJAX agisce su una porzione di pagina**, esattamente quella per cui chiediamo di effettuare una modifica che sia una modifica al DOM o un update dei dati da mostrare a video, ricevuti da un server remoto/esterno.

La tecnica AJAX utilizza una combinazione di:

- **HTML CSS** per il markup e lo stile
- **Javascript** per la manipolazione del **DOM** (**D**ocument **O**bjct **M**odel)
- L'oggetto **XMLHttpRequest** per le chiamate tra browser e web server;
- **XML /Json** come formato di scambio dei dati

Nel presentare **AJAX** abbiamo visto che il formato dei dati come interscambio è **XML** (la X di AJAX ) ma in realtà avrete molto più a che fare con un formato più semplice e leggibile che è **JSON**.

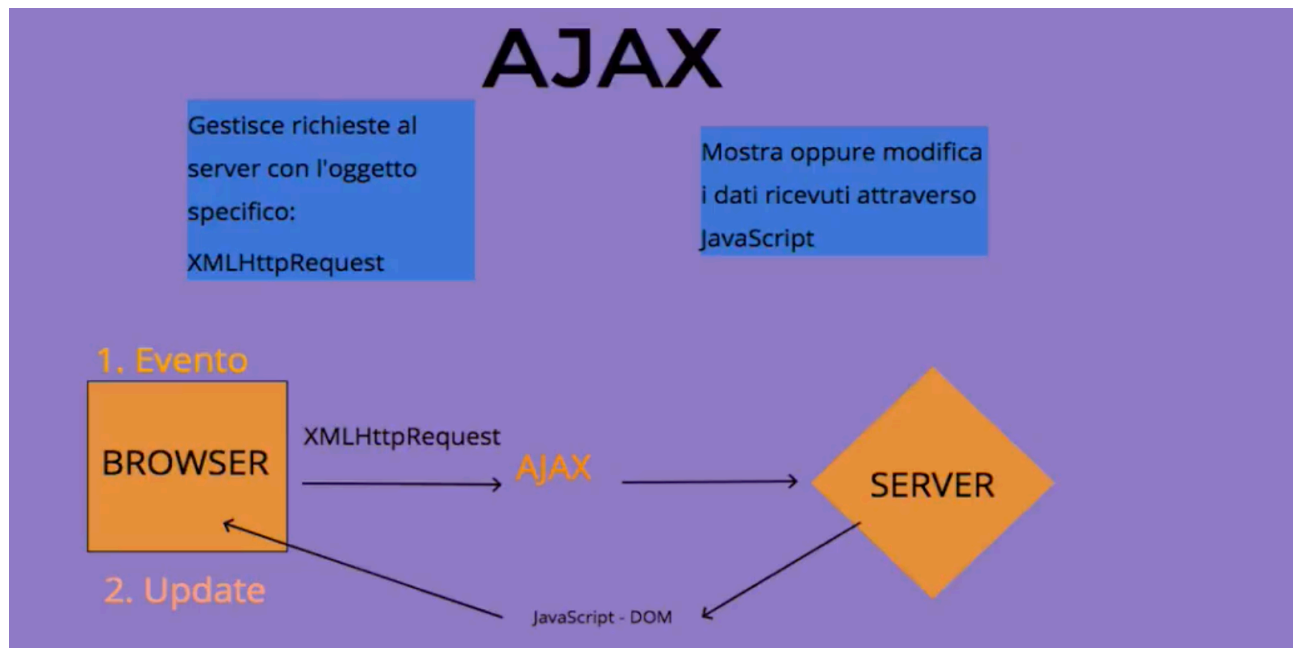


## Come funziona

Di seguito un breve schema del suo funzionamento.

Abbiamo un **Client** che fa una richiesta, grazie ad un oggetto

**XMLHttpRequest**, ad un **Server** che, quando sarà pronto (aspetto **Async**), risponderà e la risposta del server produrrà l'aggiornamento di una porzione di **HTML** senza **refresh** grazie al **Javascript** (con uso di **JQuery**, **Js vanilla** o altra **libreria**).



## Nota:

Quando lavoriamo con i **json** possiamo imbatterci in **json** semplici, complessi, **array di json** oppure **json** che dentro ha un array di oggetti. Ebbene sapere cosa fare ogni volta.

Ad esempio potremmo trovare un oggetto json con una **chiave** “**users**” che ha dentro un **array** di **oggetti** **{ users: [ {}, {}, {}, {} ] }**

Oppure potremmo trovarci davanti a qualcosa di simile un array di object **[ {}, {}, {}, {} ]**

```
{
  "users": [
    {
      "id": 1,
      "name": "Valentino Romano",
      "age": "46",
      "email": "vr@cosmopolis.biz",
      "pets": [
        "cat",
        "dog"
      ]
    },
    {
      "id": 1,
      "name": "Leanne Graham",
      "age": "40",
      "email": "sincere@april.biz",
      "pets": [
        "rabbit",
        "dog"
      ]
    }
  ]
}

[
  {
    "id": 1,
    "name": "Valentino Romano",
    "age": "46",
    "email": "vr@eht.biz",
    "pets": [
      "cat",
      "dog"
    ]
  },
  {
    "id": 1,
    "name": "Leanne Graham",
    "age": "40",
    "email": "sincere@april.biz",
    "pets": [
      "rabbit",
      "dog"
    ]
  }
]
```

Sembra non esserci differenza ma al momento di doverci lavorare la differenza è sostanziale.

# 1°Esempio: AJAX- Local Request

Creiamo il nostro primo piccolissimo progetto. **first-example.html**

Per farlo possiamo utilizzare un webserver (**MAMP, XAMP WAMP LAMP**) oppure Visual Studio code con Live Server.

## Cosa sono i Local Server Web

Per chi non lo sapesse, **XAMP** o anche **MAMP, WAMP** e **LAMP** sono **stack implementativi** che ci offrono la possibilità di avere un Server Web in locale senza scomodarci a pagare un server web remoto.

Sono costituiti da **Apache HTTP server**, il **Database** (MySQL/MariaDB/altro) e un linguaggio di programmazione lato server tipo **Php**.

La “X” di **Xamp** sta per **multiplatform**

Link WAMPServer: <https://wampserver.it.uptodown.com/windows/download>

**MA..... prima di piombarvi scaricare un Local Web Serve.....**

Lavorando con **Visual Studio Code** e possiamo scaricare una particolare **Estensione (Live server)**, così non servirà nessun **Local Web Server**. ;-)

**Live server**, inserirà per noi un piccolo script per avviare un **Local Web Server**.

**Link utile:** [https://www.w3schools.com/js/js\\_ajax\\_http.asp](https://www.w3schools.com/js/js_ajax_http.asp)

## Per il momento vediamo come creare un oggetto Json

### Vediamo un esempio completo

esempio **Corso Javascript-typescript-6-json-Ajax-XMLRequest (first-example)**,

```
<!DOCTYPE html>
<html>

<head>
  <style>
    #demo{
      width: 200px;
      height: 200px;
      background-color: orange;
      margin: 20px auto;
    }
  </style>
  <title>Page Title</title>
</head>

<body>
  <div id="demo">Non siamo ancora attivi</div>

  <script>
    function showAjaxInAction(){

      // Oggetto XMLHttpRequest
      let request = new XMLHttpRequest();
      //      method      url      async
      request.open("GET", "user.json", true) // Creazione
      // Invio richiesta
      request.send();

      // Callback: verrà eseguita quando la risposta sarà pronta
      request.onload = function(){
        document.getElementById("demo").innerHTML= request.response
      }
    }
  </script>
</body>
</html>
```

Al momento il nostro codice mostra solo una pagina con un div con sfondo Arancio anche se abbiamo già inserito il codice JS per la chiamata **Ajax**.

Ci mancano ancora dei passaggi:

Dobbiamo creare la risorsa “**user.json**”, capire cos’è l’oggetto **XMLHttpRequest** ed infine **postare a video il risultato**,

Per creare “**user.json**” basta creare un file dentro la cartella del progetto alla stesso livello e popolare il file con un **oggetto JSON** come segue :

```
{
  "id": 1,
  "name": "Valentino Romano",
  "age": "46",
  "email": "vr@eht.biz",
  "pets": [
    "cat",
    "dog"
  ]
}
```

## L’oggetto XMLHttpRequest

Vediamo di capire cosa è una richiesta eseguita con **XMLHttpRequest()**

- **new XMLHttpRequest()** crea un oggetto **XMLHttpRequest** per la richiesta;
- **abort()** cancella la richiesta
- **open()** costruisce le specifiche della richiesta con i seguenti parametri (alcuni opzionali\*):
  - **metodo : GET/POST**
  - **url** : percorso della richiesta (in locale o sul server)
  - **async. : true / false**
  - **\*username:** permette di inserire uno username di accesso al server
  - **\*password:** permette di inserire un password di accesso al server
- **send()** invia la richiesta

### Eventi:

- **onload:** definisce una funzione di **CallBack** che è chiamata quando la risposta del server sarà arrivata;
- **onreadystatechange** = ci dice in quale stato ci troviamo dopo la richiesta. **Attenzione viene richiamata diverse volte;**
  - **readyState** = indica lo stato della nostra richiesta

- 0 = richiesta non ancora inviata;
- 1 = connessione al server stabilita;
- 2 = la richiesta è stata ricevuta;
- 3 = la richiesta è in fase di processamento;
- 4 = la richiesta è conclusa e la response è pronta;
- **response**: la risposta del server;
- **responseText**: la risposta in formato **stringa**
- **responseXML**: la risposta in formato **XML**
- **status**: restituisce un messaggio relativo alla richiesta, esempio:
  - **200** = ok
  - **403** = accesso non autorizzato
  - **404** = risorsa non trovata

**Nota** : I vari codici di response html li trovate a questo link: [https://www.w3schools.com/tags/ref\\_httpmessages.asp](https://www.w3schools.com/tags/ref_httpmessages.asp)

1. Informational responses (100–199)
2. Successful responses (200–299)
3. Redirection messages (300–399)
4. Client error responses (400–499)
5. Server error responses (500–599)

**Facciamolo funzionare:** Inseriamo un **button** per poter chiamare la nostra funzione e postare a video il risultato. **(first-example completo)**

```
<!DOCTYPE html>
<html>

<head>
  <style>
    #demo{
      width: 300px;
      height: 200px;
      background-color: orange;
      margin: 20px auto;
    }
  </style>
  <title>Page Title</title>
</head>

<body>
  <div id="demo">Non siamo ancora attivi</div>
  <button id="simpleButton" onclick="showAjaxInAction();">Click me</button>

  <script>
    function showAjaxInAction() {
      // Oggetto XMLHttpRequest
      let request = new XMLHttpRequest();

      // Callback: verrà eseguita quando la risposta sarà pronta
      request.onload = function () {
        //document.getElementById("demo").innerHTML = request.response
        let user = JSON.parse(request.response)
        console.log("response: user = ", user)
        let formattedUser = `<li> ${user.id} - ${user.name} age ${user.age}`
        document.getElementById("demo").innerHTML = formattedUser
      }
      //      method    url      asyn
      request.open("GET", "user.json", true) // Creazione e invio
      // Invio richiesta
      request.send();
    }
  </script>
</body>
</html>
```

Da notare `request.open("GET", "user.json", true)` il **true** alla fine ci dà la possibilità di generare una chiamata **Asincrona**. In false sarebbe Sincrona .

Miglioriamo la parte del **json** per mostrare a video informazioni che abbiamo un senso per l'utente utilizzando il **JSON.parse()**:

```
request.onload = function () {  
    //document.getElementById("demo").innerHTML = request.response  
    let user = JSON.parse(request.response)  
    console.log("response: user = ", user)  
    let formattedUser = `- ${user.id} - ${user.name} age ${user.age}`  
    document.getElementById("demo").innerHTML = formattedUser  
}

```



## Secondo esempio

Nel secondo esempio mostreremo come lavorare con un oggetto **json** con una **chiave** “**users**” che ha dentro un **array** di **oggetti** { users: [ {}, {}, {}, {} ] }

ovvero:

```
{
  "users": [
    {
      "id": 1,
      "name": "Valentino Romano",
      "age": "46",
      "email": "vr@cosmopolis.biz",
      "pets": [
        "cat",
        "dog"
      ]
    },
    {
      "id": 1,
      "name": "Leanne Graham",
      "age": "40",
      "email": "sincere@april.biz",
      "pets": [
        "rabbit",
        "dog"
      ]
    }
  ]
}
```

Questa volta però utilizzeremo altre opzioni della chiamata **XMLHttpRequest** come ad esempio il controllo se il server ha inviato una risposta, controlleremo lo stato della richiesta e se lo status è 200 o meno.

`request.onreadystatechange`

Se vi ricordate l'oggetto **XMLHttpRequest** ci permette di vedere queste proprietà ed anche altre e che sono molto utili per gestire le nostre richieste al server e controllare di volta in volta se effettivamente la risposta è arrivata o meno ed in caso gestire il tutto nel migliore dei modi (spinner in caso di caricamento, page 404 in caso di risorsa non trovata, page 500 in caso di server in avaria e via dicendo)

Vediamo un esempio sotto

## Codice esempio

```
<head>
  <style>
    #lista {
      width: 400px;
      height: 100px;
      color: white;
      background-color: orange;
      margin: 20px auto;
    }

    #simpleButton {
      width: 200px;
      height: 50px;
      color: white;
      background-color: orange;
      border: 1px;
      border-radius: 8px;
      margin: 0 auto;
      display: block;
    }
  </style>
</head>

<body>
  <div="lista">Non siamo ancora attivi</div>
  <button id="simpleButton" onclick="showUsers();">Click me</button>
  <script>
    function showUsers() {
      // aggiungo lo spinner
      //spin.className = "loader"
      // meglio se uso un toggle su classList
      spin.classList.toggle("loader")
      let request = new XMLHttpRequest();
      request.onreadystatechange = function () {
        if (request.readyState == 4 && request.status == 200) {
          let myResponse = JSON.parse(request.responseText)
          console.log(myResponse);
          console.log(myResponse.users);
          // occhio qui, devo prendere l'array che si trova dentro la chiave users
          const users = myResponse["users"]
          //const users = myResponse.users // adesso anche così
          let mostra = "";
          for (let i = 0; i < users.length; i++) {
            mostra += '<li>' + users[i].name + '</li>';
            console.log("mostra = " + mostra);
          }
          document.getElementById("lista").innerHTML = mostra
        } else if (request.readyState == 4 && request.status == 404) {
          alert("Ops ... qualcosa è andato storto")
        }
      }
      request.open("GET", "users.json", true)
      request.send();
    }
  </script>
</body>
```

A questo punto non ci resta che fare un esempio con un array di users

**[ {}, {}, {} ]** come il seguente:

```
[
  {
    "id": 1,
    "name": "Leanne Graham",
    "age": "40",
    "email": "sincere@april.biz",
    "pets": [
      "rabbit",
      "dog"
    ]
  },
  {
    "id": 1,
    "name": "Valentino Romano",
    "age": "46",
    "email": "vr@cosmopolis.biz",
    "pets": [
      "cat",
      "dog"
    ]
  }
]
```

Creiamo la risorsa allo **stesso livello** del nostro file e chiamiamolo **users\_array.json**

Dopodiché possiamo scrivere la funzione che mostrerà il nostro nuovo json come **array di json**:

```
function showArray() {
    let request = new XMLHttpRequest();

    request.onreadystatechange = function () {
        if (request.readyState == 4 && request.status == 200) {
            // Occhio qui, l'array è già la risposta diversante da prima che dovevo accedere al
            contenuto relativo alla chiave di "users"
            let myResponseArray = JSON.parse(request.responseText)
            console.log(myResponseArray);
            console.log(myResponseArray.length);
            let mostra = "";
            for (let i = 0; i < myResponseArray.length; i++) {
                console.log(i);
                mostra += `<li> ${myResponseArray[i].name} </li>`;
                console.log("mostra = " + mostra);
            }
            document.getElementById('lista').innerHTML = mostra

        } else if (request.readyState == 4 && request.status == 404) {
            alert("Ops ... qualcosa è andato storto")
        }
    }
    request.open("GET", "users_array.json", true)
    request.send();
}
```

Come potete vedere la gestione è leggermente diversa ma probabilmente vi ritroverete a gestire questo tipo di problema molto spesso.

## Esempio Js & JSON on SERVER Array - AJAX - 3

Fino ad ora abbiamo chiamato risorse locali quindi non abbiamo visto davvero il funzionamento di una chiamata di callback in Asincrono. Vediamo di creare un progetto che utilizzi una chiamata **REST** ad un server on line.

Per farlo ci serviremo di un servizio che ci offre dei **Json** già pronti.

Il servizio di chiama <https://jsonplaceholder.typicode.com>

Ritorna un array di oggetti Json (esattamente come l'ultimo esempio fatto) motivo per cui l'unica cosa che dobbiamo fare per avere una call ad una risorse esterna è modificare la nostra richiesta ovvero cambiare

```
request.open("GET","usersArray.json", true)
```

con

```
request.open("GET","https://jsonplaceholder.typicode.com/users", true)
```

Il risultato non cambia, avremo sempre un array di oggetti **user** pronti all'uso ;-)

**Nota:** se proviamo a modificare l'URL della risorsa inserendo ad esempio un errore come ad esempio "**https://jsonplaceholder.typicode.com/usersxxxx**"

vedremo apparire un alert relativo ad:

```
else if (request.readyState == 4 && request.status == 404) {  
    alert("Ops ... qualcosa è andato storto")  
}
```

## Aggiungiamo un loader con classList.toggle

Possiamo migliorare sul codice aggiungendo un load durante le operazioni in background.

Per il loader usiamo il solo css così non impattiamo sulla cpu dell'utente: aggiungiamo il seguente spinner preso da [w3cScool](https://w3cschool.com/)

```
<style>
.loader {
  border: 6px solid #f3f3f3;
  border-radius: 50%;
  border-top: 6px solid #3498db;
  position: absolute; // posizioniamolo sopra tutto
  top: 2%;
  left: 50%;
  width: 20px;
  height: 20px;
  -webkit-animation: spin 2s linear infinite;
  /* Safari */
  animation: spin 2s linear infinite;
}

/* Safari */
@-webkit-keyframes spin {
  0% {-webkit-transform: rotate(0deg);}
  100% {-webkit-transform: rotate(360deg);}
}

@keyframes spin {
  0% { transform: rotate(0deg); }
  100% { transform: rotate(360deg); }
}
</style>
```

Aggiungiamo un segnaposto per il nostro loader ovvero un div anche se a fare la posizione sarà sempre il css da noi impostato

```
<body>
.....
<button id="simpleButton" onclick="showArray();">Click me to show usersArray.json </
button>
<div id="spin" class="over"></div>

<script>
function showUsers() {
  // aggiungo lo spinner
  //spin.className = "loader"
  // meglio se uso un toggle su classList
  spin.classList.toggle("loader")
  let request = new XMLHttpRequest();

  request.onreadystatechange = function () {
    if (request.readyState == 4 && request.status == 200) {
      let myResponse = JSON.parse(request.responseText)
```

```

    console.log("myResponse = ", myResponse);
    console.log("myResponse.users = ", myResponse.users);
    const users = myResponse["users"]
    let mostra = "";
    for (let i = 0; i < users.length; i++) {
        console.log(i);
        mostra += `<li> ${users[i].name} </li>`;
        console.log("mostra = " + mostra);
    }
    // elimino la classe dello spinner
    //spin.className = ""
    // meglio se uso un toggle su classList
    spin.classList.toggle("loader")

    document.getElementById('lista').innerHTML = mostra

    } else if (request.readyState == 4 && request.status == 404) {
        alert("Ops ... qualcosa è andato storto")
    }
    }
    request.open("GET", "users.json", true)
    request.send();
}

```

Potremmo anche provare ad allungare il tempo di aggiornamento della lista inserendo un `setTimeout` che simula un tempo più lungo della ricezione della risposta

```

setTimeout(() => {
    console.log('setTimeout')
    spin.classList.toggle("loader")
    document.getElementById('lista').innerHTML = mostra
}, 2000)

```

## Esempio AJAX & JQuery: risorsa in locale No JSON : AJAX-4

JQuery è una **libreria**.

E' stata creata nel **2006** da **John Resig**. È stata progettato per gestire le incompatibilità del browser e per semplificare la manipolazione **HTML DOM**, la gestione degli eventi, le animazioni e le chiamate **Ajax**.

Per molt anni , jQuery è stata la libreria JavaScript più **popolare al mondo**.

Tuttavia, dopo JavaScript Versione 5 (2009), la maggior parte delle utility jQuery può essere risolta con alcune righe di **JavaScript** standard.

### Esempio:

Finding HTML Element by Id

Return the element with id="id01":

#### JQuery:

```
myElement = $("#id01");
```

#### Js:

```
myElement = document.getElementById("id01");
```

#### Js shortmode

uso il suo reference variable dato dal suo id : id01

La **sintassi jQuery** è fatta su misura per la selezione di elementi HTML e per eseguire qualche azione sull'elemento.

Basic syntax is: **\$(selector).action()**

- il **\$** viene utilizzato per accedere agli elementi del DOM
- Il **(selector)** serve a fare la "query per trovare l'elemento HTML
- **action()** è l'azione che JQuery deve compiere sull'elemento

Esempi:

**\$(this).hide()** - nasconde l'**elemento** corrente

**\$("p").hide()** - nasconde **tutti gli elementi/tag <p>**

**\$(".test").hide()** - nasconde **tutti gli elementi/tag** con **class="test"**.

**\$("#test").hide()** - nasconde l'**elemento** con **id="test"**.

Vedrete spesso qualcosa di questo tipo

```
$(document).ready(function(){  
    // jQuery methods go here...  
});
```

Serve ad evitare di fare operazioni prima che l'intero DOM sia caricato.  
in shortmode avremo

```
$(function(){  
    // jQuery methods go here...  
});
```

Altri esempi

`$("p")` seleziona tutti gli elementi **<p>**

`$("#test")` seleziona l'elemento con **id="test"**

`$(".test")` seleziona tutti gli elementi con **class="test"**

`$("p.test")` seleziona tutti gli elementi **<p>** con **class="test"**

`$("p:first")` seleziona il primo elemento/tag **<p>** trovato

`$("ul li:first")` seleziona il primo elemento/tag **<li>** del primo tag **<ul>**

Altri esempi li trovate su [w3cSchool](http://www.w3cSchool.com) o sul sito ufficiale di [jQuery](http://jquery.com)

Vediamo un esempio di utilizzo di **AJAX** con **jQuery**.

Riprendiamo il primo esercizio (**first-example**), per comodità e, partendo dallo stesso obiettivo, effettuiamo le modifiche opportune per poter far lavorare **AJAX** con **jQuery**.



```

<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>

<body>
  <div id="demo">Non siamo ancora attivi</div>
  <div id="bs">Avviso Da beforeSend</div>
  <button id="simple">Click me (jQuery example)</button>
  <button id="complete">Click me (jQuery example)</button>

  <!--Link CDN Google alla Libreria di JQuery -->
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.4/jquery.min.js"></script>
  <script>
    $(document).ready(function () {
      $('#simple').click(function () {
        $.ajax('user.json')
          .done(function (res) {
            console.log('res = ', res)
            //$('#demo').text(res);
            $('#demo').empty()
            $('#bs').empty()
            $('#demo').append('<div>${res.name}</div>');
          })
      })
    });

    $(document).ready(function () {
      $('#complete').click(function () {
        $.ajax('user.json', {
          beforeSend: function () {
            $('#bs').text('Sto caricando i dati ... parte lo spinner');
          }
        })
          .done(function (res) {
            console.log('res = ', res)
            //$('#demo').text(res);
            $('#demo').empty()
            $('#demo').append('<div>${res.name} ${res.email}</div>');
          })
          .always(function () {
            $('#bs').text('Dati caricati!');
          })
          .fail(function (errorType, errorMessage) {
            alert(errorMessage)
            console.log(errorType)
          })
      })
    });
  </script>

```

```
});
```

```
</script>
```

```
</body>
```

```
</html>
```

## Collegare JQuery ai nostri Progetti

La prima cosa da fare è collegare la risorsa JQuery al nostro progetto. Per farlo utilizzeremo il **link alla CDN Google di JQuery**:

<https://developers.google.com/speed/libraries>

Prima cosa inseriamo la nostra nuova libreria **JQuery** e poi, come già sapete, i nostri script.

Riprendiamo il primo esercizio (**first-example**), per comodità e, partendo dallo stesso obiettivo, effettuiamo le modifiche opportune per poter far lavorare **AJAX** con **JQuery**.

```
<head>
  <style>
    #demo{
      width: 200px; height: 200px;
      color: white; background-color: orange;
      margin: 20px auto;
    }
    #bs{
      width: 200px; height: 50px;
      color: white; background-color: orange;
      margin: 20px auto;
    }
    #simpleButton{
      width: 200px; height: 50px;
      color: white; background-color: orange;
      border: 1px; border-radius: 8px;
      margin: 0 auto;
      display: block;
    }
  </style>
  <title>Page Title</title>
</head>

<body>
  <div id="demo">Non siamo ancora attivi</div>
  <div id="bs">Avviso Da beforeSend</div>
  <button id= "simpleButton">Click me (jQuery example)</button>

  <!--Link CDN Google alla Libreria di JQuery-->
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.4/jquery.min.js"></script>
```

**<!-- i nostri script vanno inseriti subito dopo lo script che incorpora JQuery-->**

```
<script>
$(document).ready(function(){
    $('button').click(function(){
        $.ajax('user.json')
        .done(function(res){
            $('#demo').html(res);
        })
    })
});
```

// test con **BeforeSend** always & fail

```
$(document).ready(function(){
    $('button').click(function(){
        $.ajax('user.json',{
            beforeSend: function(){
                $('#bs').text('Sto caricando i dati ... parte lo spinner');
            }
        })

        .done(function(res){
            $('#demo').html(res);
        })

        .always(function(){
            $('#bs').text('Dati caricati!');
        })

        .fail(function(errorType, errorMessage){
            alert(errorMessage)
            console.log(errorType)
        })
    })
});
```

```
</script>
```

```
</body>
```

```
</html>
```