
## PROGRAMMAZIONE WEB

# LA TECNOLOGIA AJAX

**Prof. Ada Bagozi**
ada.bagozi@unibs.it

# Un nuovo modello


- ✓ L'utilizzo di DHTML (JavaScript/Eventi + DOM + CSS) delinea un nuovo modello per le applicazioni Web
- ✓ In pratica ci troviamo di fronte ad un modello ad eventi simile a quello delle applicazioni tradizionali
- ✓ Abbiamo però due livelli di eventi:
  - ✓ eventi locali che sono gestiti da event handler codificati in Javascript e che portano ad un cambiamento locale della pagina
  - ✓ eventi remoti che sono gestiti tramite ricaricamento della pagina che viene modificata sul lato server in base ai parametri passati tramite HTTP
- ✓ Il ricaricamento di una pagina per rispondere ad un evento remoto prende il nome di postback

# Esempio di evento remoto (I)


- ✓ Consideriamo un modulo in cui compaiono due menu a tendina che servono a selezionare il comune di nascita di una persona
  - ✓ un menu con le province
  - ✓ un menu con i comuni
- ✓ Si vuole fare in modo che, scegliendo la provincia nel primo menu a tendina, appaiano nel secondo menu solo i comuni di quella provincia

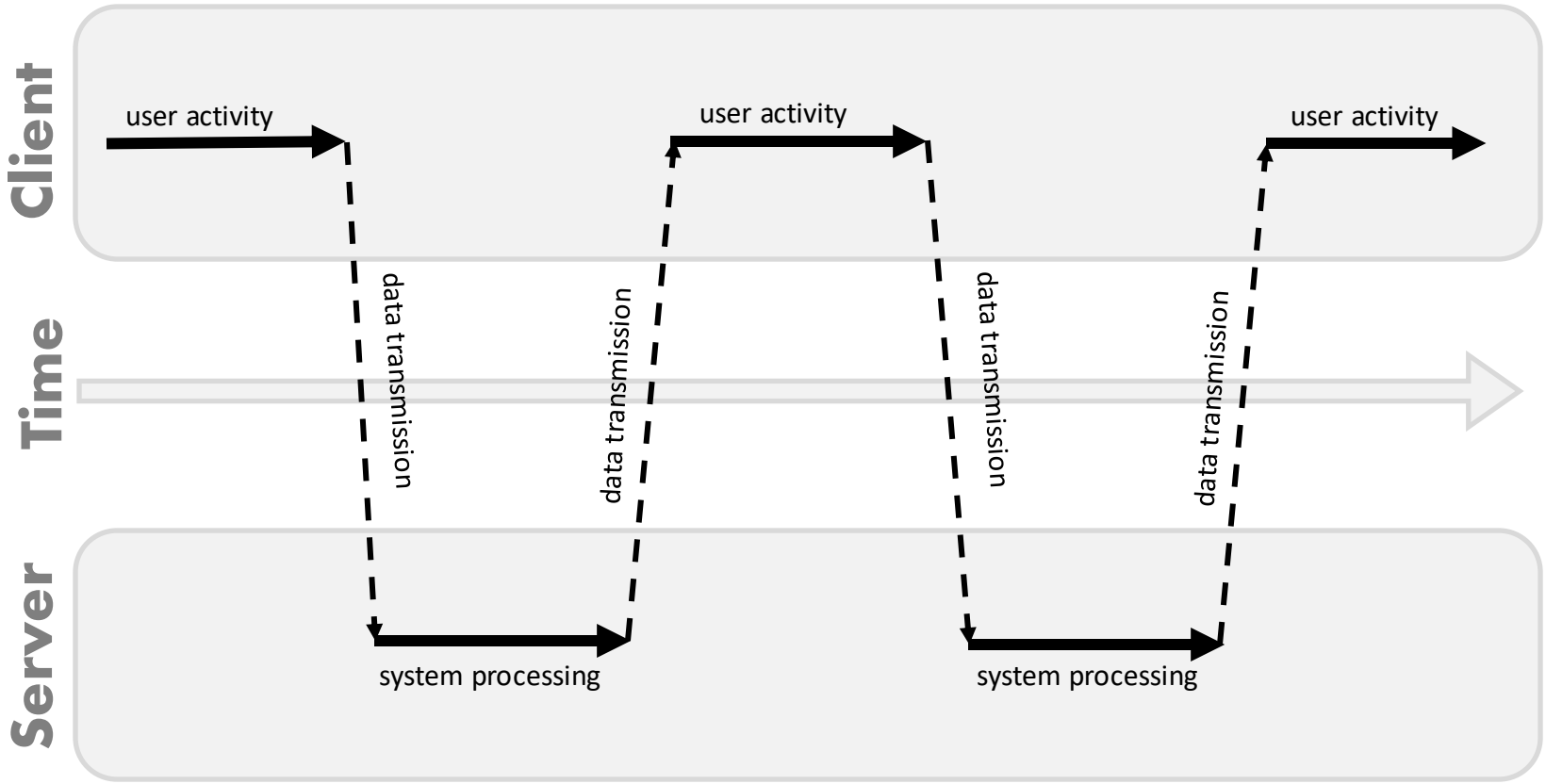
# Esempio di evento remoto (II)


- ✓ Per realizzare questa interazione si crea uno script lato server (per esempio, in PHP) che inserisce nel menu a tendina dei comuni l'elenco di quelli che appartengono alla provincia passata come parametro
  1. si definisce un evento `onChange` collegato all'elemento `select` delle province che forza il ricaricamento della pagina (postback) quando una provincia viene selezionata
  2. l'utente sceglie una provincia
  3. viene invocato lo script lato server con il parametro della provincia impostato al valore scelto dall'utente
  4. la pagina restituita contiene nel menu a tendina dei comuni l'elenco di quelli che appartengono alla provincia scelta

# Limiti del modello


- ✓ Quando lavoriamo con applicazioni desktop siamo abituati ad un elevato livello di interattività:
  - ✓ le applicazioni reagiscono in modo rapido ed intuitivo ai comandi
- ✓ Le applicazioni Web tradizionali espongono invece un modello di interazione rigido
  - ✓ modello “Click, wait and refresh”
  - ✓ è necessario il refresh della pagina da parte del server per la gestione di qualunque evento remoto (sottomissione di dati tramite form, visita di un link per ottenere informazioni di interesse, …)
- ✓ È ancora un modello sincrono: l’utente effettua una richiesta e deve attendere la risposta da parte del server

# Modello di interazione classico



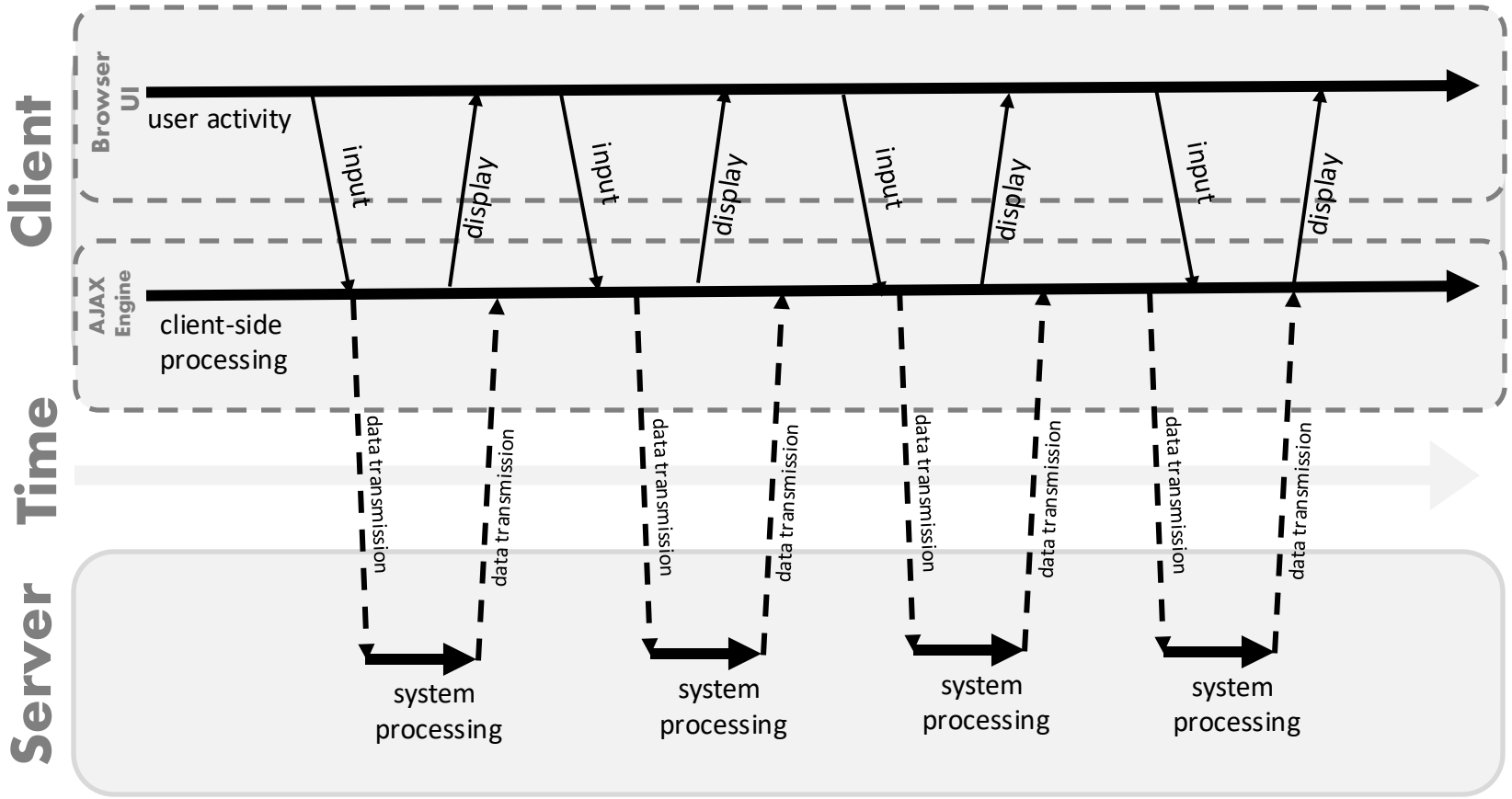
# AJAX (I)


- ✓ Il modello AJAX è nato per superare queste limitazioni
- ✓ AJAX non è un acronimo ma spesso viene interpretato come Asynchronous Javascript And XML
- ✓ È basato su tecnologie standard:
  - ✓ JavaScript (e jQuery)
  - ✓ DOM
  - ✓ XML/JSON
  - ✓ HTML
  - ✓ CSS

# AJAX (II)


- ✓ AJAX punta a supportare applicazioni user friendly con un'interattività elevata
- ✓ Per tali applicazioni, si usa spesso il termine RIA (Rich Internet Application)
- ✓ L'idea alla base di AJAX è quella di consentire agli script JavaScript di interagire direttamente con il server
  - ✓ ottenere dati dal server senza la necessità di ricaricare l'intera pagina
  - ✓ introdurre anche una *comunicazione asincrona* fra client e server: il client non interrompe l'interazione con l'utente anche quando è in attesa di risposte dal server

# Modello di interazione classico



# Tipica sequenza AJAX


- ✓ Si verifica un evento determinato dall'interazione fra l'utente e la pagina Web
- ✓ L'evento comporta l'esecuzione di una funzione JavaScript in cui:
  - ✓ si prepara l'invio ad un URL sul server, predisponendo delle funzioni di *callback*, da eseguire all'arrivo della risposta del server
  - ✓ si effettua una chiamata al server
- ✓ Il server elabora la richiesta e risponde al client
- ✓ Il browser invoca l'azione asincrona che:
  - ✓ elabora il risultato
  - ✓ aggiorna il DOM della pagina per mostrare i risultati dell'elaborazione

# La chiamata AJAX


- ✓ La chiamata AJAX effettua la richiesta di una risorsa via HTTP ad un server Web
  - ✓ non sostituisce l'URI della propria richiesta all'URI corrente
  - ✓ non provoca un cambio di pagina
  - ✓ può inviare eventuali informazioni (parametri) sotto forma di variabili (come un modulo)
  - ✓ può effettuare sia richieste GET che POST
- ✓ Le richieste possono essere di tipo
  - ✓ *sincrono*: blocca il flusso di esecuzione del codice Javascript (ci interessa poco)
  - ✓ *asincrono*: non interrompe il flusso di esecuzione del codice Javascript né le operazioni dell'utente sulla pagina

# Gli ingredienti di AJAX


- ✓ L'URL della risorsa remota di invocare
  - ✓ tipicamente uno script, che ritorna (parte di) una pagina HTML, ma anche del semplice testo (plain, JSON, XML)
- ✓ Gli eventuali dati da passare allo script eseguito in remoto
- ✓ Il metodo HTTP da utilizzare (e, se necessario, gli header HTTP da impostare)
- ✓ La modalità (*asincrona*, sincrona)
- ✓ Il tempo che si è disposti ad aspettare per l'arrivo della risposta
- ✓ Le funzioni (*callback*) da invocare, in caso di successo o insuccesso della richiesta, di errore etc.

# JQuery e AJAX (I)


JQuery rende l'uso di AJAX estremamente rapido e agevole

**`$.ajax(url[,settings])`** esegue una chiamata AJAX all'indirizzo **`url`**, configurandola opportunamente attraverso l'insieme di coppie *chiave,valore* contenute nell'oggetto **`settings`**

- **`async`** – un parametro booleano per stabilire se la chiamata AJAX è sincrona (bloccante) o asincrona (opzione di default – **`{async: true}`**)
- **`data`** – i dati da inviare lato server per processare la richiesta (*query string*); a sua volta, questo parametro è impostato come un insieme di coppie *chiave,valore*
- **`headers`** – un insieme di coppie *chiave,valore* per settare l'header della richiesta HTTP che si sta inviando al server
- **`method`** – il metodo HTTP da utilizzare per inviare la richiesta al server («get», «post») – il valore di default è «get»

# JQuery e AJAX (II)


- **`dataType`** – il formato atteso della risposta HTTP da parte del server («xml», «html», «json», «text»)
- **`statusCode`** – un oggetto contenente i codici HTTP della risposta (p.e., «200» OK) e una funzione per ciascun codice da eseguirsi al verificarsi del codice stesso

```
1 $.ajax({
2   statusCode: {
3     404: function() {
4       alert( "page not found" );
5     }
6   }
7 });
```

- **`timeout`** – un valore espresso in millisecondi per fermare la gestione della richiesta

# JQuery e AJAX – Callback functions


- **`success`** – una funzione che viene invocata quando la richiesta è terminata con successo; gli argomenti della funzione sono il contenuto della risposta, formattata secondo il **`dataType`**, una stringa che rappresenta lo stato della risposta e un oggetto di tipo **`xmlHttpRequest`**
- **`error`** – una funzione che viene invocata quando la richiesta fallisce; gli argomenti della funzione sono un oggetto di tipo **`xmlHttpRequest`**, una stringa che rappresenta lo stato della risposta e una stringa contenente il messaggio di errore
- **`complete`** – una funzione che viene invocata quando la richiesta è stata gestita (dopo l'esecuzione di **`success`** e **`error`**); gli argomenti della funzione sono un oggetto di tipo **`xmlHttpRequest`** e una stringa che rappresenta lo stato della risposta («success», «nocontent», «error», «timeout»)

**Deprecation notice** – Da jQuery3.0 **`complete`**, **`error`** e **`success`** sono sostituite rispettivamente da **`always`**, **`fail`** e **`done`**

# Proprietà di XMLHttpRequest


- ✓ Stato e risultati della richiesta vengono memorizzati dall'interprete Javascript all'interno dell'oggetto **`XMLHttpRequest`** durante la sua esecuzione
- ✓ Per compatibilità all'indietro, **`XMLHttpRequest`** presenta le proprietà seguenti:

  **`readyState`**

  **`status`**

  **`statusText`**

  **`statusCode`**

  **`responseText`**

  **`responseXML`**

# Proprietà `ReadyState`


- ✓ Proprietà in sola lettura di tipo intero che consente di leggere in ogni momento lo stato della richiesta; ammette 5 valori:

`0: uninitialized` - l'oggetto esiste, ma non è stato invocato il metodo `open()`

`1: open` - è stato invocato il metodo `open()`, ma `send()` non ha ancora effettuato l'invio dati

`2: sent` - il metodo `send()` è stato eseguito ed ha inviato la richiesta

`3: receiving` – la risposta ha iniziato ad arrivare

`4: loaded` - l'operazione è stata completata

- ✓ Attenzione:
  - ✓ questo ordine non è sempre identico e non è sfruttabile allo stesso modo su tutti i browser
  - ✓ l'unico stato supportato da tutti i browser è il 4

# Metodi di XMLHttpRequest


I metodi `getAllResponseHeaders()` e `getResponseHeader(name)` consentono di leggere gli header HTTP che descrivono la risposta del server

- ✓ sono utilizzabili solo in caso di ricezione della risposta (`readystate>=3`)

Il metodo `setRequestHeader(nomeheader,valore)` consente di impostare gli header HTTP della richiesta da inviare

- ✓ viene invocata più volte, una per ogni header da impostare
- ✓ per una richiesta GET gli header sono opzionali
- ✓ sono invece necessari per impostare la codifica utilizzata nelle richieste POST

# Esempio (I)


```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <title>.: Un semplice esempio AJAX :.</title>
    <script type="text/javascript" src="https://code.jquery.com/jquery.js"></script>
  </head>
  <body>
    <h1>Ricerca nome</h1>
    <form>
      <p>Cerca un nome per il tuo bambino: noi abbiamo centinaia di suggerimenti da darti!<p>
      <input type="text">
    </form>

    <div id="risultati"></div>
    <script>
      $(document).ready(function(){
        $("input[type=text]").keyup(function(){
          $.ajax("ajax.php",{data: {nome: $(this).val()}, success: function(result) {
            $("#risultati").html(result);
          }});
        });
      });
    </script>
  </body>
</html>
```

**ajax.php?nome=...**

```php
<?php
    $nomi[0] = "Alessandro";
    $nomi[1] = "Alessio";
    $nomi[2] = "Claudio";
    $nomi[3] = "Davide";
    $nomi[4] = "Dario";
    $nomi[5] = "Francesco";
    $nomi[6] = "Giancarlo";
    $nomi[7] = "Luca";
    $nomi[8] = "Luigi";

    $nome = $_GET["nome"];
    $risultato = "";

    if (strlen($nome) > 0) {
        for ($i = 0; $i < count($nomi); $i++) {
            if (strtoupper($nome) == strtoupper(substr($nomi[$i], 0, strlen($nome)))) {
                if ($risultato == "") {
                    $risultato = $nomi[$i];
                }
                else {
                    $risultato .= ", " . $nomi[$i];
                }
            }
        }
    }
    if ($risultato == "") {
        echo "Nessun risultato...";
    }
    else {
        echo $risultato;
    }
?>
```

# AJAX e XML


- ✓ Spesso i dati scambiati fra client e server sono codificati in XML
- ✓ AJAX come abbiamo visto è in grado di ricevere documenti XML
- ✓ In particolare è possibile elaborare i documenti XML ricevuti utilizzando le API W3C DOM
- ✓ Per visualizzare i contenuti ricevuti modificchiamo il DOM della pagina HTML

# AJAX e JSON (I)


- ✓ JSON è tuttavia il formato più utilizzato per lavorare con AJAX
- ✓ Sul lato client:
  - ✓ si crea un oggetto JavaScript e si riempiono le sue proprietà con le informazioni necessarie
  - ✓ si usa `JSON.stringify()` per convertire l'oggetto in una stringa JSON
  - ✓ si manda la stringa al server mediante `XMLHttpRequest` (la stringa viene passata come variabile con GET o POST)

# AJAX e JSON (II)


- ✓ Sul lato server:
  - ✓ si decodifica la stringa JSON e la si trasforma utilizzando un apposito parser (si trova sempre su **www.json.org**)
  - ✓ si elabora l'oggetto
  - ✓ si crea un nuovo oggetto JSON che contiene i dati della risposta
  - ✓ si trasmette la stringa JSON al client nel corpo della risposta HTTP

# AJAX e JSON (III)


- ✓ Sul lato client:
  - ✓ si converte la stringa JSON in un oggetto Javascript usando `JSON.parse()`
  - ✓ si usa liberamente l'oggetto per gli scopi desiderati

# AJAX e JQuery - Shortcuts


`$.get(url [, data] [, success], [, dataType])` esegue una chiamata al server asincrona di tipo **GET**

`$.post(url [, data] [, success], [, dataType])` esegue una chiamata al server asincrona di tipo **POST**

`$.getJSON(url [, data] [, success])` esegue una chiamata al server asincrona di tipo **GET** e restituisce un risultato in formato **JSON**

https://api.jquery.com/jquery.ajax/

https://api.jquery.com/category/ajax/shorthand-methods/

```
<head>
  <title>Esempio nell'uso di JSON</title>
  <script type="text/javascript" src="lib/json2.js"></script>
  <script type="text/javascript" src="https://code.jquery.com/jquery.js"></script>
</head>

<body>
  <form name="personal" action="" method="POST">
    Nome <input type="text" name="firstname"><br>
    Email <input type="text" name="email"><br>
    Hobby <input type="checkbox" name="hobby" value="sport"> Sport
      <input type="checkbox" name="hobby" value="lettura"> Lettura
      <input type="checkbox" name="hobby" value="musica"> Musica
    <input type="button" name="valid" value="Validate">
    <script>
      $(document).ready(function(){
        $("input[type=button]").click(function() {
          var JSONObject = new Object;
          JSONObject.firstname = $("input[name=firstname]").val();
          JSONObject.email = $("input[name=email]").val();
          JSONObject.hobby = new Array;

          $("input[type=checkbox]:checked").each(function(i){
            JSONObject.hobby[i] = new Object;
            JSONObject.hobby[i].hobbyName = $(this).val();
          });

          JSONstring = JSON.stringify(JSONObject);
          alert(JSONstring);
          $.ajax("parser.php",{data: {json: JSONstring}, success: function(result){
            alert(result);
          }});
        });
      });
    </script>
  </form>
</body>
```

# Esempio


```php
<?php
// decodifica della stringa JSON in un oggetto PHP
$decoded = json_decode($_GET['json']);

// creazione della risposta sotto forma di oggetto
$json = array();
$json['name'] = $decoded->firstname;
$json['email'] = $decoded->email;
$json['hobbies'] = array();
for($i=0; $i<count($decoded->hobby); $i++)
{
   $json['hobbies'][] = $decoded->hobby[$i]->hobbyName;
}

// codifica dell'array $json in una stringa JSON
$encoded = json_encode($json);

// invio della risposta e fine dello script
die($encoded);
?>
```

**parser.php?json=…**

# Vantaggi e svantaggi di AJAX


- ✓ Si guadagna in espressività, ma si perde la linearità dell'interazione
- ✓ Mentre l'utente è all'interno della stessa pagina le richieste sul server possono essere numerose e indipendenti

- ❖ Il tempo di attesa passa in secondo piano o non è avvertito affatto
- ❖ Possibili criticità sia per l'utente che per lo sviluppatore
  - ❖ percezione che non stia accadendo nulla (sito che non risponde)
  - ❖ problemi nel gestire un modello di elaborazione che ha bisogno di aspettare i risultati delle richieste precedenti

# Criticità nell'interazione con l'utente


- ✓ Le richieste AJAX permettono all'utente di continuare a interagire con la pagina
- ✓ Ma non necessariamente lo informano di cosa sta succedendo e possono durare troppo!
- ✓ L'effetto è un disorientamento dell'utente
- ✓ Dobbiamo quindi agire su due fronti:
  - ✓ rendere visibile in qualche modo l'andamento della chiamata (barre di scorrimento ecc.)
  - ✓ interrompere le richieste che non terminano in tempo utile per sovraccarichi del server o momentanei problemi di rete (timeout)

# Aspetti critici per il programmatore


- ✓ È accresciuta la complessità delle applicazioni Web
- ✓ La logica di presentazione è ripartita fra client-side e server-side
- ✓ Le applicazioni AJAX pongono problemi di debug, test e mantenimento
- ✓ Il test di codice JavaScript è complesso
- ✓ Il codice JavaScript ha problemi di modularità
- ✓ I toolkit AJAX sono molteplici e solo recentemente hanno raggiunto una discreta maturità
- ✓ Mancanza di standardizzazione nei vecchi browser

# FETCH


È l'API nativa di JavaScript per effettuare richieste HTTP.

- Sostituisce metodi più vecchi come XMLHttpRequest e $.ajax() di jQuery
- Basata su Promesse, quindi si integra perfettamente con async/await
- Supportata da tutti i browser moderni (tranne Internet Explorer)

# FETCH vs AJAX


| Feature | $.ajax() (jQuery) | fetch() (Native) |
|---|---|---|
| Built-in | ❌ Needs jQuery | ✅ Yes, in modern browsers |
| Promise-based | ❌ (callback-based) | ✅ Native support |
| Automatic JSON parse | ✅ (with dataType) | ❌ You must call .json() |
| HTTP error thrown | ✅ Triggers error | ❌ Must manually check response.ok |
| Cancel requests | ✅ With .abort() | ✅ With AbortController |
| CORS cookies | ✅ Easier | ❌ Must use credentials flag |

# FETCH Examples


## Esempio Base

```
fetch('/api/dati')
        .then(response => response.json())
        .then(data => console.log(data))
        .catch(error => console.error('Errore:', error));
```

## Esempio con async/await

```
async function caricaDati() {
        try {
                const risposta = await fetch('/api/dati');
                const dati = await risposta.json();
                console.log(dati);
        } catch (errore) {
                console.error('Errore nella richiesta:', errore);
        }
}
```


## PROGRAMMAZIONE WEB

# LA TECNOLOGIA AJAX

**Prof. Ada Bagozi**
ada.bagozi@unibs.it