

Sviluppare applicazioni Web 2.0 con Ajax*

Maurizio Cozzetto

11 Agosto 2009

Ajax

Definizione

Il termine *Ajax* è apparso la prima volta su Internet il 18 febbraio 2005 nell'articolo "*Ajax: A New Approach to Web Applications*" (<http://adaptivepath.com/ideas/essays/archives/000385.php>) scritto da da *Jesse James Garrett*, fondatore di *Adaptive Path* (<http://adaptivepath.com>) (fig. 1).

Garrett nel suo articolo scrive:

"Ajax isn't a technology. It's really several technologies, each flourishing in its own right, coming together in powerful new ways. Ajax incorporates:

- standards-based presentation using XHTML and CSS;
- dynamic display and interaction using the Document Object Model;
- data interchange and manipulation using XML and XSLT;
- asynchronous data retrieval using XMLHttpRequest;
- and JavaScript binding everything together."

Traduzione:

"Ajax non è una tecnologia. E' l'insieme di più tecnologie indipendenti che insieme danno vita a potenti prospettive. Ajax incorpora a se:

- presentazione standard utilizzando XHTML e CSS;
- layout dinamico con possibilità di interazione mediante il Document Object Model;
- scambio e manipolazione dati mediante l'utilizzo di XML e XSLT;
- recupero asincrono dei dati utilizzando XMLHttpRequest;
- ed infine Javascript che amalgama il tutto."

Il nome Ajax (che si dovrebbe scrivere *AJAX*), acronimo di "*Asynchronous JavaScript And XML*" rappresenta un modo moderno e avanzato di scrivere applicazioni web ed è usato in molte applicazioni di grande successo, come *Google Suggest*, *Google Maps*, *Google Mail*, *Flickr*, *Meebo* ecc (quelle che nel linguaggio comune vengono definite come *applicazioni Web 2.0*). Recentemente, come alternativa a XML e XSLT, comincia ad essere molto usato anche *JSON (JavaScript Object Notation)*, un *sottoinsieme* di Javascript basato sullo standard ECMA 262.

Sono basati su Ajax molti framework *lato client* come *Dojo*, *Prototype*, *Scriptaculous*, *jQuery*, *mootools*, *YUI (The Yahoo! User Interface Library)* ecc e framework *client-server* come *JSF (JavaServer Faces)*, *ICEFaces* e *RichFaces*

*Liberamente adattato da *Wikipedia* (<http://it.wikipedia.org/wiki/AJAX>), da *Ajax Tutorial* (<http://www.w3schools.com/Ajax/Default.Asp>), dall'articolo di Alessandro Lacava "*Sviluppare un framework Ajax*" (ioProgrammo, agosto 2009, n. 141) e dal libro di Ivan Venuti "*Programmazione con Javascript, dalle basi ad Ajax*", edizioni FAG Milano

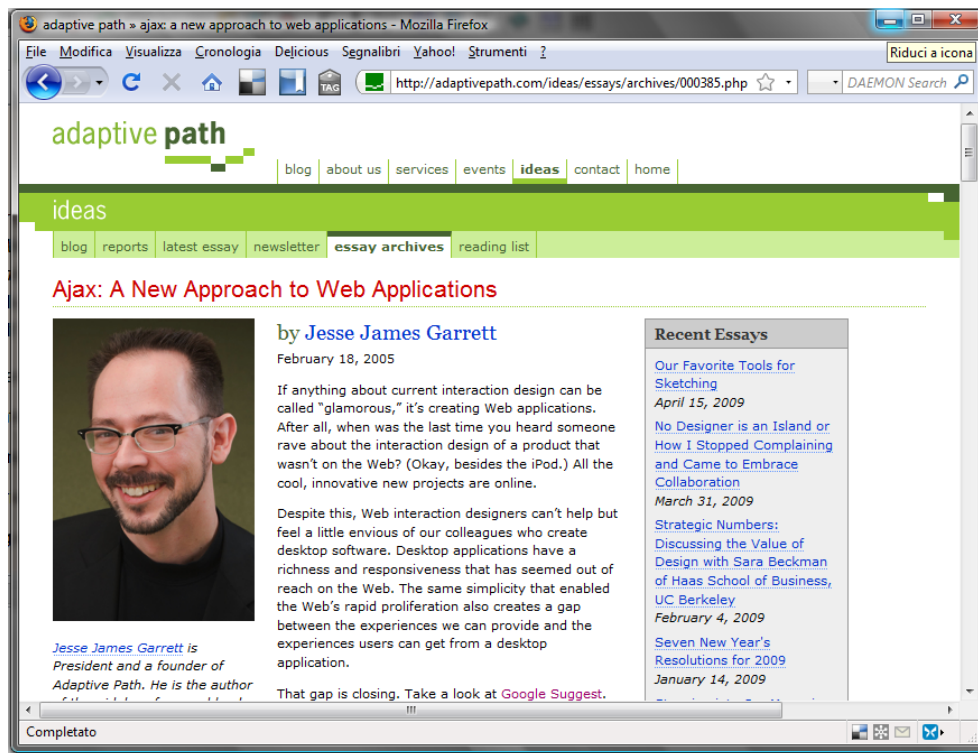


Figura 1: La pagina web relativa all'articolo scritto da *Jesse James Garrett*

(basati su JSF) oppure *DWR (Direct Web Remoting)* e *GWT (Google Web Toolkit)* non basati su JSF ecc. Ajax tuttavia non è esente da difetti (ricordiamo il problema del *bookmarking* e il problema del bottone *back*) e la scrittura di un'applicazione Ajax richiede alcuni accorgimenti.

Le applicazioni oggi

Fondamentalmente abbiamo tre scelte di base quando scriviamo applicazioni:

- Applicazioni *desktop*
- Applicazioni *web*
- Rich Internet Application (*RIA*)

Applicazioni desktop

Sono le usuali applicazioni, richiedono un CD di installazione (o una connessione a Internet per il download) e si installano completamente sul computer, sono molto veloci e hanno interfacce grafiche a volte anche molto complesse e "ricche".

Applicazioni web

Non necessitano di alcuna installazione, girano in un browser web (di solito già installato sul pc dell'utente), forniscono servizi impensabili in una applicazione desktop tradizionale (come i servizi forniti da *Amazon*, *eBay* ecc), ma richiedono lunghi tempi di attesa della risposta del server o del refresh o della generazione di una pagina web.

Rich Internet Application

Ajax tenta di colmare il *gap* tra applicazioni tradizionali e applicazioni web, consentendo lo sviluppo delle cosiddette cosiddette *RIA (Rich Internet Application)*, migliorando l'esperienza dell'utente e garantendo maggiore interattività, velocità ed usabilità. Una RIA basata su Ajax è un'applicazione dinamica e interattiva che gira in un browser, non

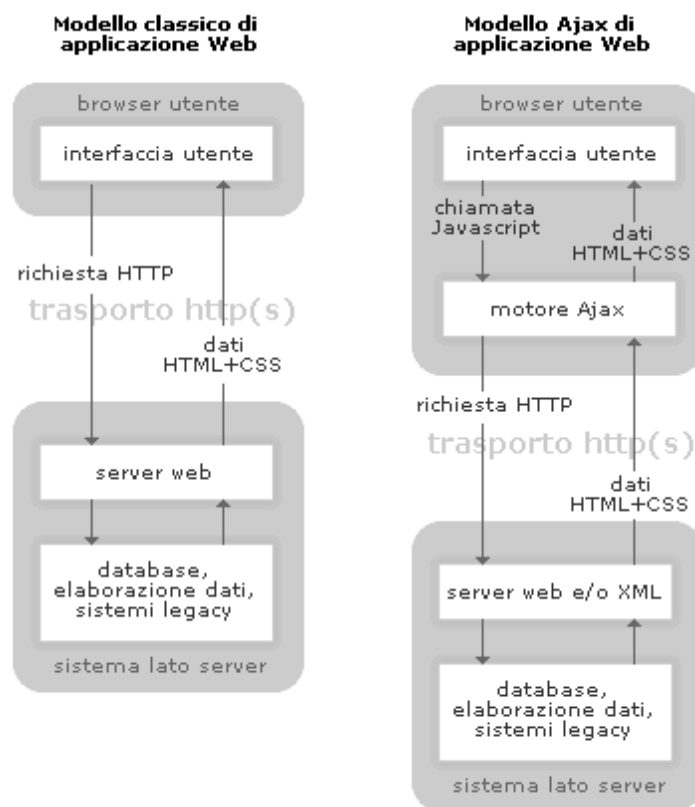


Figura 2: Modello classico e modello Ajax a confronto

richiede plug-in, mantiene tempi di caricamento accettabili, è leggera e mantiene l'indipendenza dalla piattaforma. Ajax non è tuttavia l'unica tecnologia con queste prerogative: altre tecnologie nate prima di Ajax sono le *Applet Java*, *JavaFX*, *Java WebStart*, le *ActiveX* e *Silverlight* di Microsoft, *Air*, *Flash* e *Flex* di Adobe, *Apollo*, *OpenLaszlo*, *XUL*, *XAML*, *SVG* ecc. Tecnologie non basate su Ajax come Flash, le Applet ecc possono richiedere il download dei corrispondenti plug-in.

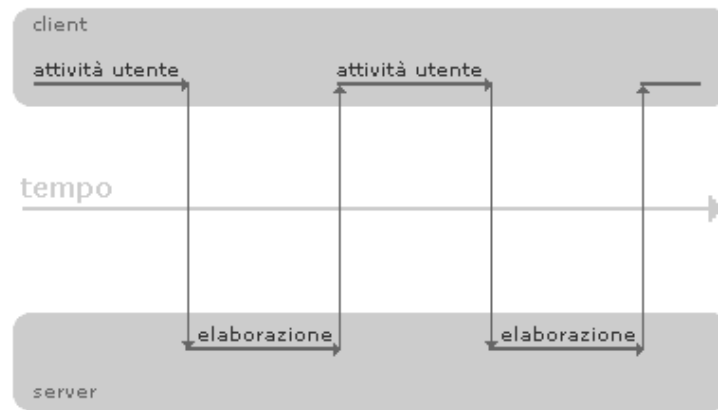
Modello classico

Il modello classico (*fig. 2, parte di sinistra*) funziona in questo modo: il cliente inoltra una richiesta (in genere una pagina web, statica o dinamica) al server web (un server web è un applicativo - per esempio *Apache* - in ascolto su una porta, di solito la porta 80). Se la pagina esiste ed è statica viene copiata e inoltrata al client. Se la pagina è dinamica, il motore di scripting lato server provvede a generare la sua parte di contenuto e a unirlo al codice HTML già disponibile e poi comunque la inoltra al client. Il modello classico soffre tuttavia di un grave difetto: se l'utente interagisce ancora col client (per esempio cliccando su un link) viene inoltrata al server una nuova richiesta e le parti della pagina, quali i menu, i riquadri di testo e le immagini, che sono gli stessi della pagina precedente (magari generati dinamicamente) vanno di nuovo riprocessati e ricaricati. Questo comporta uno spreco di banda e allunga i tempi di caricamento, come si può immaginare.

Modello Ajax

Nel modello Ajax (*fig. 2, parte di destra*), le applicazioni, d'altra parte, possono inviare richieste al web server per ottenere *solo* i dati che sono necessari grazie alla possibilità di effettuare chiamate *asincrone*. Come risultato si ottengono applicazioni più veloci dato che la quantità di dati interscambiati si riduce come si riduce anche il tempo di elaborazione da parte del server web visto che molti dati sono stati già inoltrati ed elaborati (*fig. 3*).

Modello classico di applicazione Web (sincrono)



Modello Ajax di applicazione Web (asincrono)



Figura 3: Modello sincrono e modello asincrono a confronto

Primo esempio

Un esempio concreto: consideriamo un elenco (ordinato per *codice*) di articoli di un sito di e-commerce. Se vogliamo ordinare l'elenco per *descrizione*, con un'applicazione web tradizionale, l'utente potrebbe cliccare su un link in corrispondenza dell'*intestazione* di una colonna di una tabella. Questa azione allora causerebbe l'invio di una nuova *query SQL* al server di database il quale provvederebbe a ordinare i dati e a rimandarli al server web. Il server web successivamente potrebbe ricostruire da zero la pagina web inoltrandola integralmente all'utente. Usando le tecnologie Ajax, questo evento invece potrebbe preferibilmente essere eseguito con un *JavaScript* lato client che produrrebbe dinamicamente l'ordinamento richiesto (mediante *DHTML*).

Secondo esempio

Un altro esempio potrebbe essere la scelta del *nickname* in fase di creazione di un account su un sito web. Nel modello classico, dovremmo compilare prima tutto il modulo e inviarlo al server. Ci accorgiamo, solo dopo aver atteso il caricamento della pagina di conferma, che il nome è già esistente e che dobbiamo cambiarlo. Con Ajax invece può essere introdotto un controllo sull'evento *onChange* o addirittura *onKeyUp* della casella di testo che ci può informare tempestivamente che il nome inserito non è valido, magari evidenziando il testo in rosso (mediante *CSS* e *Javascript*).

```

1 // Contenuto del file ajax.js
2
3 // variabile globale
4 var xmlhttp
5
6 function getXmlHttpRequest() {
7     // variabile locale
8     var xhr;
9
10    if (window.XMLHttpRequest) {
11        // compatibilità con IE7+, Firefox, Chrome, Opera, Safari
12        try {
13            // tento di istanziare l'oggetto
14            xhr = new XMLHttpRequest();
15            return xhr;
16        } catch(e) {
17            xhr = null;
18        }
19    } // fine if
20
21    if (window.ActiveXObject) {
22        // compatibilità con IE6, IE5
23        try {
24            // in questo caso si tratta di un ActiveX
25            xhr = new ActiveXObject("Microsoft.XMLHTTP");
26            return xhr;
27        } catch (e) {
28            xhr = null;
29        }
30    } // fine if
31
32    return xhr;
33 }

```

Figura 4: File ajax.js (applicativo AjaxSuggestPhpApp)

L'oggetto XMLHttpRequest (Javascript)

Alla base della tecnologia Ajax, vi è sempre un oggetto di classe *XMLHttpRequest*. In tutti i moderni browser, infatti la classe (o l'oggetto) *XMLHttpRequest* (in Javascript, i termini *oggetto* e *classe* sono sinonimi) è implementata all'interno dell'oggetto *window* di Javascript e una delle sue peculiarità è che essa è in grado di effettuare chiamate *http* (di tipo *sincrono* o *asincrono*). Nella versione di *IE7* (e succ.), in *Firefox*, in *Opera* ecc, tale oggetto è *nativo*, mentre nelle versioni *precedenti* di *IE*, l'oggetto invece è implementato tramite un *ActiveX*. Predisponiamo allora una funzione Javascript (*getXmlHttpRequest()*), da includere in tutti i nostri progetti Ajax (*file ajax.js*, *fig. 4*) per differenziare i due casi.

La funzione javascript *getXmlHttpRequest()* provvederà quindi, a seconda del browser, a istanziare un oggetto di classe *XMLHttpRequest* (*riga 14*) o un oggetto *ActiveX* (*riga 25*), mentre la variabile globale *xmlhttp* riceverà il valore della chiamata alla funzione.

Proprietà dell'oggetto XMLHttpRequest

La proprietà *readyState*

Dopo la sua creazione, l'oggetto *XMLHttpRequest* può trovarsi in 5 stati distinti, individuati dalla proprietà *readyState*, illustrati nella *Tabella 1*.

Valore di readyState	Descrizione
0	UNINITIALIZED. L'oggetto XHR è stato creato ma il metodo <i>open()</i> non è stato ancora chiamato.
1	LOADING. Il metodo <i>send()</i> non è stato ancora chiamato.
2	LOADED. Il metodo <i>send()</i> è stato chiamato, ma la risposta non è ancora disponibile
3	INTERACTIVE. La risposta è in caricamento e la proprietà <i>responseText</i> contiene dati parziali.
4	COMPLETED. La risposta è stata caricata e la richiesta è completa.

Tabella 1: I 5 stati dell'oggetto XMLHttpRequest (per brevità XHR)

La proprietà onreadystatechange

Quando il valore della proprietà *readyState* cambia, il motore Ajax richiama un *event listener*, ossia una funzione di *callback* invocata a ogni cambio di stato, che è nostro compito scrivere.

La proprietà.responseText

Questa proprietà contiene la risposta *testuale* del server nel caso in cui *readyState* valga 3 o 4. Nel primo caso, il response è incompleto ma ci sarà. Per valori diversi, *responseText* sarà la stringa nulla.

La proprietà.responseXML

La proprietà *responseXML* contiene un oggetto di tipo *Document* se e solo se si verificano le seguenti condizioni:

- il server risponde con un apposito Content-Type nell'header;
- il valore di *readyState* è 4;
- il response è un documento XML su cui si può effettuare il *parsing* (ossia è un documento *well-formed* ecc).

Le proprietà.status e.statusText

Le proprietà *status* e *statusText* rappresentano rispettivamente il *codice* e la *descrizione* del response HTTP del server. Ad esempio, *status* vale 200 per un response che indica un *successo* (ovviamente per valori di *readyState* pari a 3 o 4). Per valori minori di 3, è sollevata un'eccezione.

Metodi dell'oggetto XMLHttpRequest

Metodo open

Il metodo *open()* dispone dei seguenti parametri: *method*, *url*, *sync*, *username* e *password*. Gli unici parametri obbligatori sono *method* e *url*. Il primo parametro rappresenta il *metodo* da usare per la richiesta: quelli disponibili sono i classici metodi HTTP *GET*, *POST*, *PUT*, *DELETE* o *HEAD*. Di solito si usano *GET* e *POST*. Il secondo parametro (*url*) invece è la *URL* a cui postare la richiesta, di solito una pagina web *dinamica* (uno script *Php*, una *servlet*, una pagina *ASP.NET*, una *JSP* ecc ma volendo anche una *pagina statica HTML* o un *file di testo*). Il terzo parametro *sync* vale *true* o *false* nel caso rispettivamente di una richiesta *asincrona* o *sincrona* (per noi ovviamente saranno più interessanti le richieste *asincrone*). I parametri *username* e *password* si possono usare nel caso in cui il server richieda un'autenticazione.

Metodo.setRequestHeader

Questo metodo è richiamato solo dopo l'invocazione di una *open()*. *setRequestHeader* dispone di due parametri, *header* e *value*. Essi sono due valori stringa e rappresentano la coppia *chiave/valore* relativa a un determinato header.

Metodo send

Invia la richiesta vera e propria al server. Ha un solo parametro (*data*) che è una stringa in formato *querystring* in caso di richiesta di tipo *POST*, *null* in caso di richiesta di tipo *GET*.

Metodo getResponseHeader

Quando *readyState* vale 3 o 4, è possibile usare il metodo *getResponseHeader()* seguito dal parametro *header* di cui si vuole recuperare il valore. Per valori di *readyState* differenti, restituisce una stringa vuota.

Metodo getAllResponseHeaders

Il metodo *getAllResponseHeaders()* restituisce una *singola* stringa costituita da *diverse* righe, dove ogni riga rappresenta un *header* differente. Il formato è

```
header1:valore1  
header2:valore2
```

....

Per esempio:

```
Server:Apache/1.3.31(Unix)  
Keep-Alive:timeout=15, max=99
```

....

Metodo abort

Provvede a *killare* la richiesta pendente, se esiste, e riporta l'oggetto XMLHttpRequest allo stato iniziale (*readyState=0*).

Gli applicativi proposti

Presenteremo dapprima il miniapplicativo *AjaxSuggestPhpApp* in *Ajax* e *Php* che simula il comportamento di *Google Suggest* e successivamente il miniapplicativo *AjaxAutocompleterPhpApp2* che fa uso del framework *Scriptaculous* (<http://script.aculo.us>). Dal momento che *Ajax* è indipendente dal linguaggio di scripting lato server utilizzato, al posto di *Php*, avremmo potuto scegliere qualsiasi altro linguaggio lato server (come *Asp*, *Asp.NET*, *Perl*, *Python*, *Jsp* ecc).

AjaxSuggestPhpApp

L'applicativo *AjaxSuggestPhpApp* è composto dai seguenti file: *ajax.js* e *clientHint.js* (due file Javascript), la pagina *index.html* (un form) e la pagina dinamica *hint.php*.

Il file clientHint.js

Il file *clientHint.js* (fig. 7) contiene le due funzioni javascript *showHint()* e *stateChanged()*. La funzione *showHint()* riceve in ingresso il parametro *str* che rappresenta la stringa inserita dall'utente nel form (*riga 1*), provvede a creare l'oggetto XMLHttpRequest (memorizzandolo nella variabile globale *xmlHttp*, *riga 5*), sempre che questo sia possibile, e prepara la *url* da inviare al server (*righe 10-11*). La *url* comprende il riferimento alla pagina dinamica *hint.php* e alla *querystring* costituita dal parametro *q* seguito dal valore della variabile *str* e dal parametro *sid* seguito da un valore casuale (*riga 15*). Provvediamo poi a registrare la funzione di *callback* da usare quando il server restituisce il response completo (*riga 19*). Successivamente apriamo la connessione (*riga 22*) specificando una richiesta *asincrona* (valore *true* del terzo parametro) e inoltriamo la richiesta (*riga 25*). La funzione di callback *stateChanged()* visualizzerà la risposta solo in caso di response completo (*riga 33*).

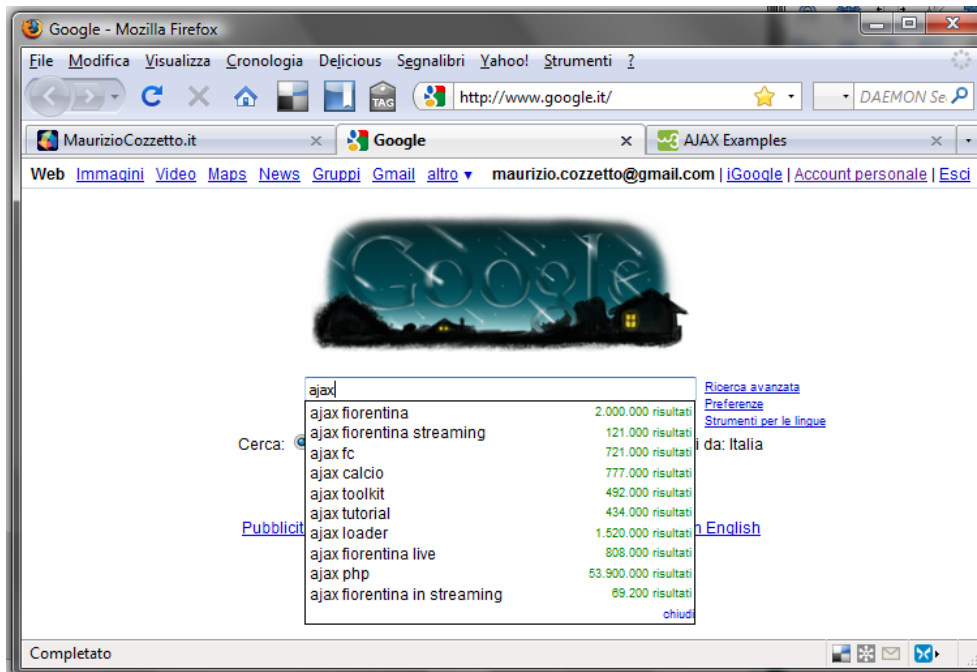


Figura 5: I suggerimenti forniti dal servizio *Google Suggest*

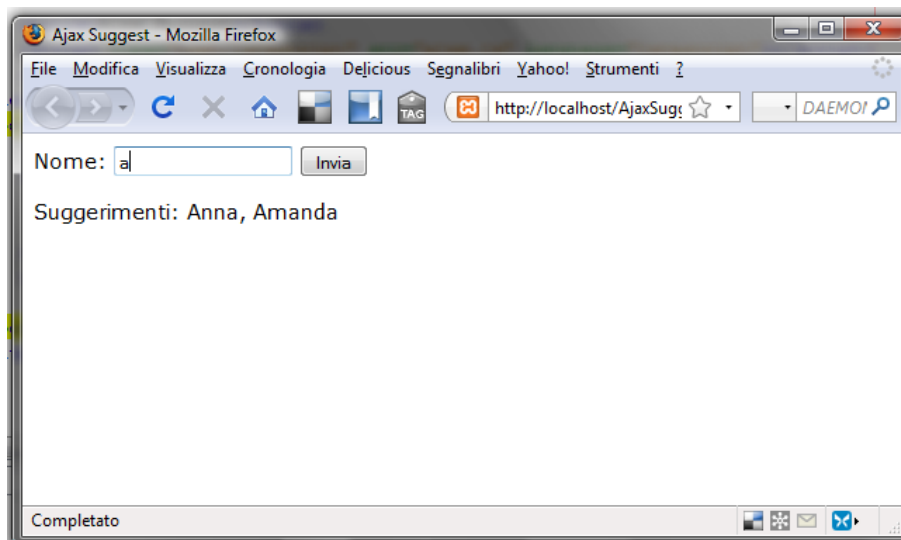


Figura 6: Il nostro miniapplicativo *AjaxSuggestPhpApp* in esecuzione nel nostro browser (*Firefox*)

```

1 function showHint(str) {
2     ...
3
4     // crea l'oggetto XHR e lo memorizza nella variabile globale xmlHttp
5     xmlHttp=getXmlHttpRequestObject();
6
7     ...
8
9     // creo la url
10    var url="hint.php";
11    url=url+"?q="+str;
12
13    // aggiunge in coda alla url un valore causale per impedire
14    // al server di usare la cache
15    url=url+"&sid="+Math.random();
16
17    // registrazione della funzione javascript di callback
18    // quando il server restituisce il response completo
19    xmlHttp.onreadystatechange=stateChanged;
20
21    // apro la connessione
22    xmlHttp.open("GET",url,true);
23
24    // inoltra la richiesta
25    xmlHttp.send(null);
26
27 } // fine della funzione showHint
28
29 function stateChanged() {
30     // se la risposta è completa
31     if (xmlHttp.readyState==4) {
32         // visualizza la risposta nel div txtHint
33         document.getElementById("txtHint").innerHTML = xmlHttp.responseText;
34     }
35 } // fine della funzione stateChanged

```

Figura 7: Il file clientHint.js (applicativo AjaxSuggestPhpApp)

```

1 <html>
2   <head>
3     <title>Ajax Suggest</title>
4     <script src="javascript/ajax.js"></script>
5     <script src="javascript/clientHint.js"></script>
6   </head>
7   <body>
8     <form>
9       <label>Nome: </label>
10      <input type="text" id="txtNome" onKeyUp="showHint(this.value)"/>
11      <input type="button" value="Invia">
12    </form>
13    <div id="txtHint">Suggerimenti:</div>
14  </body>
15 </html>

```

Figura 8: Il file index.html

Il file index.html

Il file *index.html* (fig. 8) è costituito da un *form* HTML e da un blocco *div*. Viene richiamata la funzione *showHint()* nel momento in cui l'utente digita un carattere (evento *onKeyUp*).

Il file hint.php

Riportiamo il codice nella fig. 9 senza aggiungere ulteriori commenti, essendo il codice già sufficientemente commentato.

AjaxAutocompleterPhpApp2 (Scriptaculous)

Innanzitutto è necessario scaricare le *librerie* Javascript che costituiscono il framework *Scriptaculous* (si tratta di diversi file che hanno delle dipendenze reciproche per cui consigliamo per semplicità di scaricarli tutti e di includerli nel progetto in un'apposita cartella, ad esempio *javascript*). L'applicativo consisterà di due file: il primo, il file *index.html* (fig. 10), contiene il *form* di inserimento dati e il secondo, il file *hint.php* (fig. 11), provvede a recuperare le informazioni dalla "base di dati" sul server (in realtà non usiamo un vero e proprio database, per semplicità).

Il file index.html

Il file *index.html* deve contenere l'inclusione delle librerie del framework (righe 6-7), pena il non funzionamento dell'applicativo. Nel file *index.html*, predisponiamo la casella di input per l'utente (riga 12) e istanziamo un oggetto di tipo *Autocompleter* (riga 19). Il costruttore è seguito da diversi parametri: il primo è il nome della casella testuale, il secondo è il nome del blocco nel quale vogliamo visualizzare i risultati (formattato opportunamente con uno stile apposito), il terzo è il file php da richiamare sul server e il quarto fa riferimento alla *querystring* contenente il nome della casella di input.

Il file hint.php

Il file *hint.php* esegue l'elaborazione lato server sul "database" dei nomi sfruttando la funzione php *ereg()* che permette tra le altre cose di controllare se una stringa è presente all'interno di un'altra stringa (riga 14). L'output deve essere restituito in forma di lista non ordinata.

```

1 <?php
2 // riempie l'array a con dei nomi
3 $a[]="Anna";
4 $a[]="Brittany";
5 $a[]="Cinderella";
6 ...
7
8 // ottiene il valore di q
9 $q=$_GET["q"];
10
11 ...
12
13 //cerca tutti i suggerimenti nell'array a
14 if (strlen($q) > 0) {
15     $hint=null;
16     for($i=0; $i<count($a); $i++) {
17         // confronta le iniziali dei nomi
18         //dopo averle convertite in minuscolo
19         if (strtolower($q)==strtolower(substr($a[$i],0,strlen($q)))) {
20             if (strlen($hint)==0) {
21                 $hint=$a[$i];
22             } else {
23                 $hint=$hint.", ".$a[$i];
24             } // fine if interna
25         } // fine if esterna
26     } // fine for
27 } // fine if esterna
28
29 // Se non vi sono suggerimenti, imposta un semplice testo
30 // altrimenti imposta il suggerimento corretto
31 if (strlen($hint) == 0){
32     $response="Nessun suggerimento";
33 } else {
34     $response=$hint;
35 }
36
37 // manda in output la risposta
38 echo "Suggerimenti: ".$response;
39 ?>

```

Figura 9: Il file hint.php (applicativo AjaxSuggestPhpApp)

```

1 <!DOCTYPE HTML PUBLIC "-//W3C//DTD_HTML_4.01_Transitional//EN">
2 <html>7
3   <head>
4     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
5     <title>Autocompleter test</title>
6     <script src="javascript/prototype.js" type="text/javascript"></script>
7     <script src="javascript/scriptaculous.js" type="text/javascript"></script>
8   </head>
9   <body>
10    <form action=".">
11      <label>Nome:</label>
12      <input type="text" id="txtNome" name="txtNome" value="" />
13      <input type="button" id="btnInvia" name="btnInvia" value="Invia" />
14    </form>
15
16    <div id="txtHint"></div>
17
18    <script type="text/javascript">
19      new Ajax.Autocompleter(
20        'txtNome', 'txtHint', 'hint.php', { paramName: 'txtNome' }
21      );
22    </script>
23  </body>
24 </html>

```

Figura 10: Il file index.html (applicativo *AjaxAutocompleterPhpApp2*)

```

1 <?php
2   $a=array();
3   // queste informazioni possono essere prelevate da un database
4   $a[]="Anna";
5   $a[]="Brittany";
6   $a[]="Cinderella";
7   ...
8
9   $q=$_POST["txtNome"];
10
11   echo("<ul>");
12   if (strlen($q)!=0) {
13     for ($i=0; $i<count($a); $i++) {
14       if (ereg($q,$a[$i])) echo("<li>". $a[$i]. "</li>");
15     } // fine for
16   } // fine if
17
18   echo("</ul>");
19 ?>

```

Figura 11: Il file hint.php (applicativo *AjaxAutocompleterPhpApp2*)

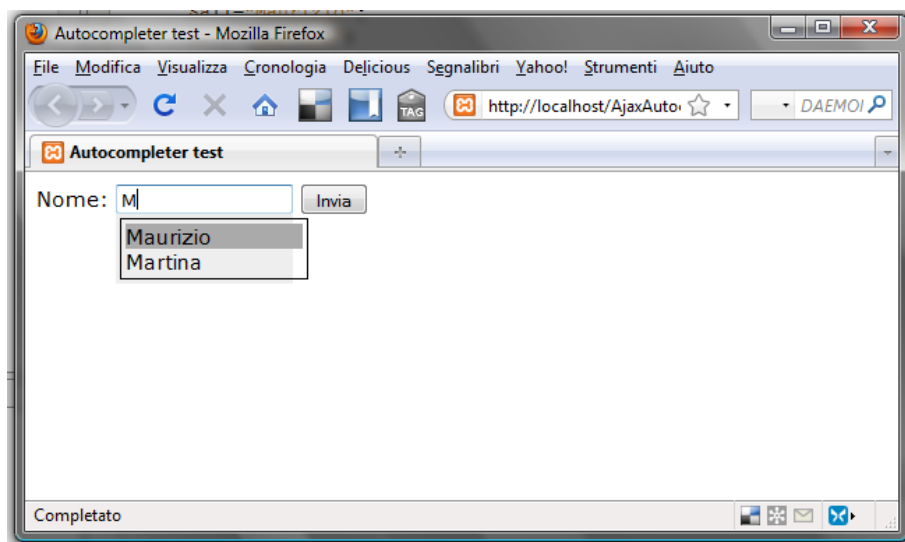


Figura 12: L'applicativo *AjaxAutocompleterPhpApp2* in esecuzione nel browser Firefox