

Tecnologie Cloud e Mobile

Lez. 07

AJAX con JQuery

Giuseppe Psaila

Università di Bergamo

giuseppe.psaila@unibg.it

JQuery e i Framework JavaScript

- JQuery è un **framework** JavaScript
- Un framework JavaScript viene sviluppato con lo scopo di
 - fornire funzionalità di alto livello
 - basate comunque sull'ambiente JavaScript
 - ma più sofisticate e complesse rispetto a quanto offerto dall'ambiente base.

Perché si Chiama JQuery?

- Perché nasce per interrogare il DOM
- Al fine di trasformarlo
- Con costrutti semplici e potenti
- Riducendo di molto lo sforzo per il programmatore

AJAX

- Tuttavia, JQuery è stato ulteriormente esteso
- Con funzionalità per gestire le chiamate AJAX in modo robusto
- Molto più robusto di quanto si possa fare con la funzione `makeAjaxRequest`

JQuery UI






- Un progetto parallelo a JQuery
- Per fornire i «gadget» o «widget» da usare nelle pagine web
- Ad esempio:
 - **Date picker**, per inserire le date senza scriverle
 - La **Progress Bar**, per indicare che il codice sta lavorando

Il Progetto JQuery


- La cosa migliore è guardare il sito ufficiale

<https://jquery.com/>


Il Progetto JQuery



PluginsContributeEventsSupportJS Foundation




write less, do more.



Your donations help fund the continued development and growth of jQuery.


[SUPPORT THE PROJECT](#)

[Download](#)[API Documentation](#)[Blog](#)[Plugins](#)[Browser Support](#)




Lightweight Footprint

Only 30kB minified and gzipped. Can also be included as an AMD module




CSS3 Compliant

Supports CSS3 selectors to find elements as well as in style property manipulation



Cross-Browser

[Chrome](#), [Edge](#), [Firefox](#), [IE](#), [Safari](#), [Android](#), [iOS](#), and [more](#)



Download jQuery

v3.4.1

The 1.x and 2.x branches no longer receive patches.

[View Source on GitHub →](#)
[How jQuery Works →](#)

La Libreria

- Per prima cosa, occorre scaricare la libreria sul server e includerla nella pagina HTML
- `<script type="text/javascript" src="jquery-2.0.0.js"></script>`
- Questa versione dell'elemento script carica un file che contiene il codice JavaScript

L'Oggetto/Funzione \$ o jquery

- Il cuore del framework è l'oggetto **jquery**
- Per comodità si usa l'alias **\$**
- È un oggetto, perché fornisce metodi (come per AJAX)
- È una funzione, che interroga il DOM
- Si sfrutta la doppia natura oggetti/funzioni di JavaScript

AJAX in JQuery

- La chiamata AJAX viene direttamente supportata da JQuery
- **La chiamata cross-domain è supportata direttamente**
- Possono essere gestiti molti formati di risposta
- Si può avere un controllo molto fine della chiamata AJAX

L'Oggetto jqXHR

L'oggetto jqXHR

- Incorpora ed estende l'oggetto XMLHttpRequest
- Consente di controllare lo stato della comunicazione (ma non è necessario)
- Memorizza i dati ricevuti
- Gestisce le funzioni di callback

L'Oggetto jqXHR

Metodi principali

- **done(funzione)**
specifica la funzione da chiamare in caso di successo della chiamata

L'Oggetto jqXHR

Metodi principali

- **fail(funzione)**
specifica la funzione da chiamare in caso di fallimento della chiamata

L'Oggetto jqXHR

Metodi principali

- **always(funzione)**
specifica la funzione da chiamare comunque al termine

Interfaccia Funzione done

**jqXHR.done(
function(data, textStatus, jqXHR) {...})**

- **data**: il dato ricevuto dalla risposta (in diversi formati)
- **textStatus**: testo che indica lo stato finale della chiamata HTTP
- **jqXHR**: l'oggetto che gestisce la sessione AJAX

Interfaccia Funzione fail

**jqXHR.fail(function(jqXHR, textStatus,
 errorThrown) {...})**

- **jqXHR**: l'oggetto che gestisce la sessione AJAX
- **textStatus**: testo che indica lo stato del processo
- **errorThrown**: l'errore restituito

Interfaccia Funzione **always**

**jqXHR.always(
function(data, textStatus,jqXHR) {...})**

- **data**: il dato ricevuto dalla risposta (in diversi formati)
- **textStatus**: testo che indica lo stato finale della chiamata HTTP
- **jqXHR**: l'oggetto che gestisce la sessione AJAX

Catene su jqXHR

- done, fail e always restituiscono ancora l'oggetto jqXHR
- Così si possono impostare catene di chiamate sull'oggetto

Esempio

```
jqReq.done( ... ).fail( .. ).always( ... )
```

L'Oggetto XMLHttpRequest

- L'oggetto XMLHttpRequest incorpora l'oggetto JavaScript nativo XMLHttpRequest
- Fornisce metodi per accedervi, se necessario
- In genere, non è necessario, ma per completezza riportiamo questi metodi e campi

L'Oggetto XmlHttpRequest

- **readyState**
- **status**
- **statusText**
- **responseXML e/o.responseText**

L'Oggetto XmlHttpRequest

- **setRequestHeader(name, value)**
- **getAllResponseHeaders()**
- **getResponseHeader()**
- **statusCode()**
- **abort()**

L'Oggetto XMLHttpRequest

- I campi e metodi appena visti sono «esattamente» gli stessi dell'oggetto XMLHttpRequest
- Questo suggerisce che il framework JQuery costruisca l'oggetto jqXHR per estensione dell'oggetto base, aggiungendo i suoi campi e i suoi metodi

Chiamata AJAX

L'oggetto **\$ (jquery)** fornisce il metodo **ajax**

- Questo metodo riceve un parametro con le impostazioni della chiamata
- È un oggetto di configurazione
- Molte opzioni possono essere specificate
- Il metodo ajax restituisce l'oggetto jqXHR

Oggetto di «Configurazione»

- Il metodo ajax riceve un oggetto di configurazione
- Questo oggetto contiene le informazioni per gestire la chiamata nel modo opportuno
- Il modo migliore per prepararlo è usare il terzo modo per creare oggetti in JavaScript, cioè prepararlo al volo (vedremo nell'esempio)

Oggetto di «Configurazione»

- **url**: l'url (anche cross-domain) della richiesta AJAX
- **type**: il metodo della richiesta
 - GET
 - POST
- **crss-domain**
 - false (default)
 - true

Oggetto di «Configurazione»

- **dataType**: il tipo di dato da ricevere
 - xml
 - html
 - json
 - jsonp
 - text

Oggetto di «Configurazione»

- **data**: il contenuto della richiesta da inviare (oggetto serializzabile o stringa)
- **Headers**: un oggetto con gli header della richiesta HTTP
- **contentType**: stringa con il MIME type della richiesta

Oggetto di «Configurazione»

- **username:** nome utente per l'autenticazione (opzionale)
- **password:** password per l'autenticazione (opzionale)

Chiamata AJAX

`$.ajax(`

```
{url: "/AJAX_00/calculate_all.jsp?v1=" +  
      n1 + "&v2=" + n2 + "&op=" + oper,  
  dataType: "xml", type: "GET"}
```

```
) .done(function(data, textStatus,  
             jqXHR) {risposta(data, "xml");})
```

```
.fail(function(jqXHR, textStatus,  
               errorThrown) {alert(errorThrown);})
```

Chiamata AJAX

- OK, così è corretta
- Ma è anche un po' contorta e difficile da leggere
- Separiamo la definizione delle funzioni dalla definizione dell'oggetto di configurazione dalla chiamata effettiva (ci servirà qualche slide)

Chiamata AJAX

```
var ajaxConf =  
{  
    url: "/AJAX_00/calculate_all.jsp?v1=" +  
        n1 + "&v2=" + n2 + "&op=" + oper,  
    dataType: "xml",  
    type: "GET"  
};
```

Chiamata AJAX

```
var success =  
    function(data, textStatus, jqXHR)  
    {  
        risposta(data, "xml");  
    }  
)
```


Chiamata AJAX

```
var failure =  
    function(jqXHR, textStatus,  
              errorThrown)  
    {  
        alert(errorThrown) ;  
    }
```

Chiamata AJAX

```
$.ajax( ajaxConf )  
    .done( success )  
    .fail( failure ) ;
```

Il Formato JSONP

- **JSONP: JSON with Padding**
- La risposta contiene una chiamata di funzione
- il cui parametro attuale è l'oggetto JSON

Esempio:

```
Process ( { "result" : 5 } )
```

Il Formato JSONP

- La funzione deve essere presente nel codice JavaScript
- Tipicamente, le chiamate in modalità GET specificano la funzione con cui fare il padding con il parametro jsonp
- Secondo me è rischioso, ma sembra che nelle chiamate AJAX cross-domain sia l'unica accettata da alcuni browser

Altri Metodi AJAX

- Il metodo ajax dell'oggetto \$ (jQuery) va benissimo, perché
- attraverso l'oggetto di configurazione si può impostare tutto quello che serve
- Tuttavia, esistono altri metodi, per fare chiamate AJAX in casi specifici

Altri Metodi AJAX

- **\$.get** chiamata con metodo GET
- **\$.getJSON** chiamata GET che ottiene JSON
- **\$.getScript** chiamata GET che riceve uno script
- **\$.post** chiamata con metodo POST

Accesso al DOM

Approccio

- Quando l'oggetto \$ è usato come funzione
- il parametro contiene dei «selettori» dei nodi del DOM
- Tipicamente, si usano i «selettori CSS»

Esempio

```
<div id="mioBlocco">
```

CIAO

```
</div>
```

L'elemento viene selezionato come

```
$ ( ' #mioblocco ' )
```

DOM Esteso

- Gli oggetti restituiti sono estensioni del DOM proprie di JQuery
- Quindi, forniscono metodi per poterli modificare
- Per esempio, il metodo `css` modifica lo stile css associato

```
$ ( ' #mioblocco ' )  
  .css ( "border" , "2px solid green" ) ;
```

Classi CSS Come Selettori

- Anche le classi di stile CSS possono essere usate
- Per esempio, si cercano gli elementi DOM con stile `miaclasse`

```
$ ( ' .miaclasse ' )
```

Selezioni più Sofisticate

- Tutti gli elementi in un blocco

```
$ ( ' #mioblocco * ' )
```

- Tutti gli elementi di un certo tipo in un blocco (esempio, list items)

```
$ ( ' #mioblocco li ' )
```

- Solo i figli diretti

```
$ ( ' #mioblocco > li ' )
```

Creazione di Blocchi DOM

Si possono anche creare blocchi DOM

```
var blocco =  
$( "<div><p>Esempio</p></div>" );
```

Che poi può essere appeso a un altro elemento
(con `appendTo`)

```
blocco.appendTo( document.body );  
blocco.appendTo( '#mioblocco' );
```

Creazione di Blocchi DOM

Dato il nuovo blocco DOM

```
var blocco =
```

```
$ ( "<div><p>Esempio</p></div>" ) ;
```

Può essere appeso in modo alternativo

```
$ ( ' body ' ) . append ( blocco ) ;
```

```
$ ( ' #mioblocco ' ) . append ( blocco ) ;
```

Liste di Nodi

- La funzione `$` (o **jquery**) restituisce «liste di nodi»
- Sfruttando la programmazione a oggetti, si possono creare catene di azioni su queste liste di nodi
- Sfruttando i metodi forniti

Liste di Nodi

- Dimensione della selezione

```
$ ( ' #menu li ' ) .size ( )
```

```
$ ( ' #menu li ' ) .length
```


Rappresentazione DOM Base

- Per ottenere la rappresentazione DOM base di JavaScript, invece di quella estesa di JQuery
- Metodo **get()** (restituisce un NodeList di DOM)

```
$ ( ' #menu li ' ) .get ( )
```

Rappresentazione DOM Base

- Metodo **get(indice)**
fornisce l'elemento (in DOM puro) nella
posizione indicata
- Si ottiene un **HTMLElement** o un **Text**

```
$ ( ' #menu li ' ) .get (0)
```

Rappresentazione DOM JQuery

- Metodo **eq(indice)**
fornisce l'elemento (in DOM JQuery) nella
posizione indicata

```
$ ( ' #menu li ' ) .eq ( 0 )
```

Attributi degli Elementi

Dato un elemento DOM JQuery, si possono manipolare i suoi attributi

- **.attr(nomeattr)**

Restituisce il valore dell'attributo

- **.attr(nomeattr, valore)**

Imposta il valore dell'attributo

- **.removeAttr(nomeattr)**

Rimuove l'attributo dall'elemento

Contenuto degli Elementi

Dato un elemento DOM JQuery, si può accedere al suo contenuto testuale

- **.text()**

Restituisce il testo del paragrafo

- **.text(stringa)**

Imposta il valore del testo del paragrafo

Contenuto degli Elementi

Dato un elemento DOM JQuery, si può accedere al suo contenuto HTML

- **.html()**

Restituisce il contenuto HTML del nodo

- **.html(stringa)**

Imposta il contenuto HTML del nodo

Classi di Stile degli Elementi

Dato un elemento DOM JQuery, si possono gestire le sue classi di stile

- **.hasClass(classe)**
booleana, dice se l'elemento ha la classe indicata
- **.addClass(classe)**
Aggiunge la classe di stile
- **.removeClass(classe)**
Rimuove la classe di stile

Processare le Liste

- Processare ogni singolo elemento di una selezione
.each()

```
$ (" #menu li" ) .each (function (i ,el)  
                                { ... } ) ;
```

- **i** è la posizione nella lista della selezione
- **el** è l'elemento (DOM puro)

Processare le Liste

```
var processs= function (i,el)
{ var index = i; //0,1,2 etc
  var id = el.id; //id dell'elemento
  alert(" (index,id)=( "+index+" , "+
        id+" ) " );
};

$("#menu li").each(process);
```

Processare le Liste

La funzione richiamata da **each** riceve il parametro **el**

- Questo è il nodo DOM puro (con tutte le specializzazioni del caso, fino ad `HTMLElement`)
- Per applicare i metodi di JQuery, occorre modificare la funzione

Processare le Liste

```
var processs= function (i,el)
var index = i; //0,1,2 etc
    var id = $(el).attr("id");
alert(" (index,id)=( "+index+" , "+
        id+" ) ");
};

$("#menu li").each(process);
```

Campi delle form

- Come processare i campi delle form?
- Attraverso i selettori css si seleziona il campo di interesse
- A quel punto, si possono usare dei metodi per accedere al valore del campo

Campi delle form

- **.val()**

Restituisce il valore del campo (singolo o array)

- **.val(valoresingolo)**

Imposta il valore del campo (se valore singolo)

- **.val(array)**

Imposta l'array di valori (se valori multipli ammessi)

Eventi

- Si possono impostare eventi e gestori di evento
- Con il metodo **.bind(evento, funzione)**
- La funzione di callback riceve il parametro **event**
- **event** ha il campo **target**, che è il nodo DOM puro su cui l'evento è avvenuto

Eventi: Esempio

```
$ ("a").bind("click",  
    function (event)  
    {  
        var target = event.target;  
        alert(target.tagName);  
        //il nome del tag su cui abbiamo  
        //cliccato  
    }  
);
```

Eventi

Metodi e proprietà utili:

- **event.preventDefault()**
Inibisce l'azione di default dell'evento
- **event.type**
tipo dell'evento (es. "click")
- **event.target**
l'elemento oggetto dell'evento (DOM puro)

Funzione di Inizializzazione

- Il codice

```
$ (function ()  
    { alert("DOM caricato!"); } ) ;
```

- Esegue la funzione con il DOM pronto, ma prima che gli stili e le immagini siano state caricate

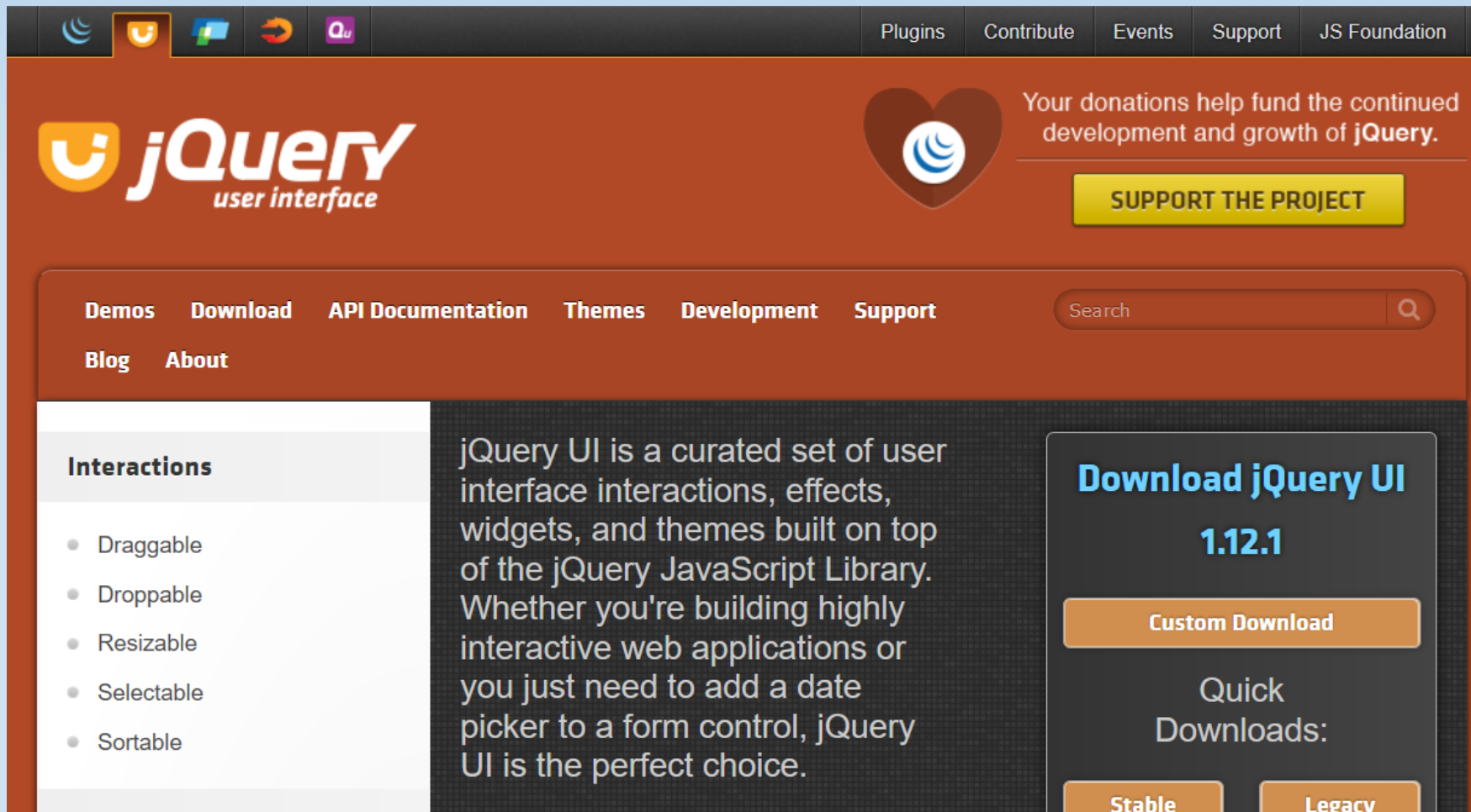
JQuery UI

JQuery UI

- Si tratta di un sotto-progetto di JQuery
- O meglio, è un progetto di contorno
- Focalizzato sul fornire Widget da inserire nelle pagine HTML
- I Widgget sono accompagnati da codice JavaScript che lavora su stili e DOM
- Il programmatore può sfruttare queste funzionalità

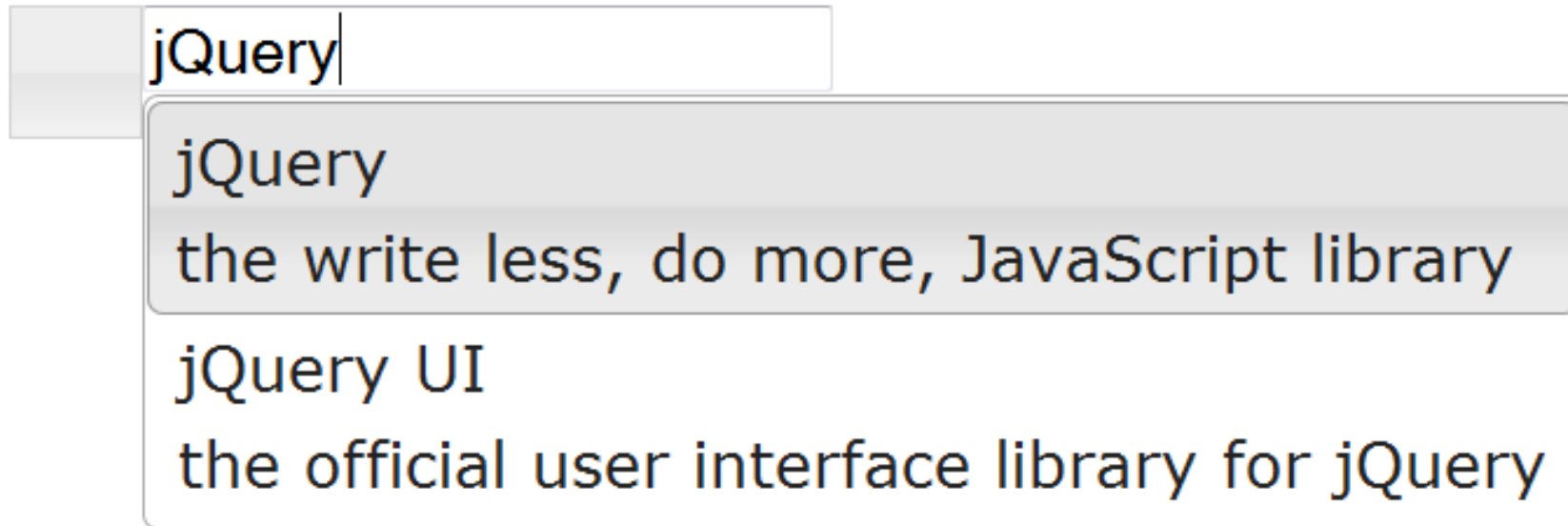
jQuery UI

- Sito Web: www.jqueryui.com



Widget: Autocomplete

Select a project (type "j" for a start):



The image shows a web form with a text input field containing the text "jQuery". Below the input field is a dropdown menu with three items. The first item is "jQuery" with the description "the write less, do more, JavaScript library" below it. The second item is "jQuery UI" with the description "the official user interface library for jQuery" below it. The third item is empty.

Project Name	Description
jQuery	the write less, do more, JavaScript library
jQuery UI	the official user interface library for jQuery

Widget: Menu

Aberdeen

Ada

Adamsville





Addyston

Delphi >





Saarland

Salzburg >

Amesville

-  Save
-  Zoom In
-  Zoom Out
-  Print...

Playback >

-  Prev
-  Stop
-  Play
-  Next

Widget: Progress Bar



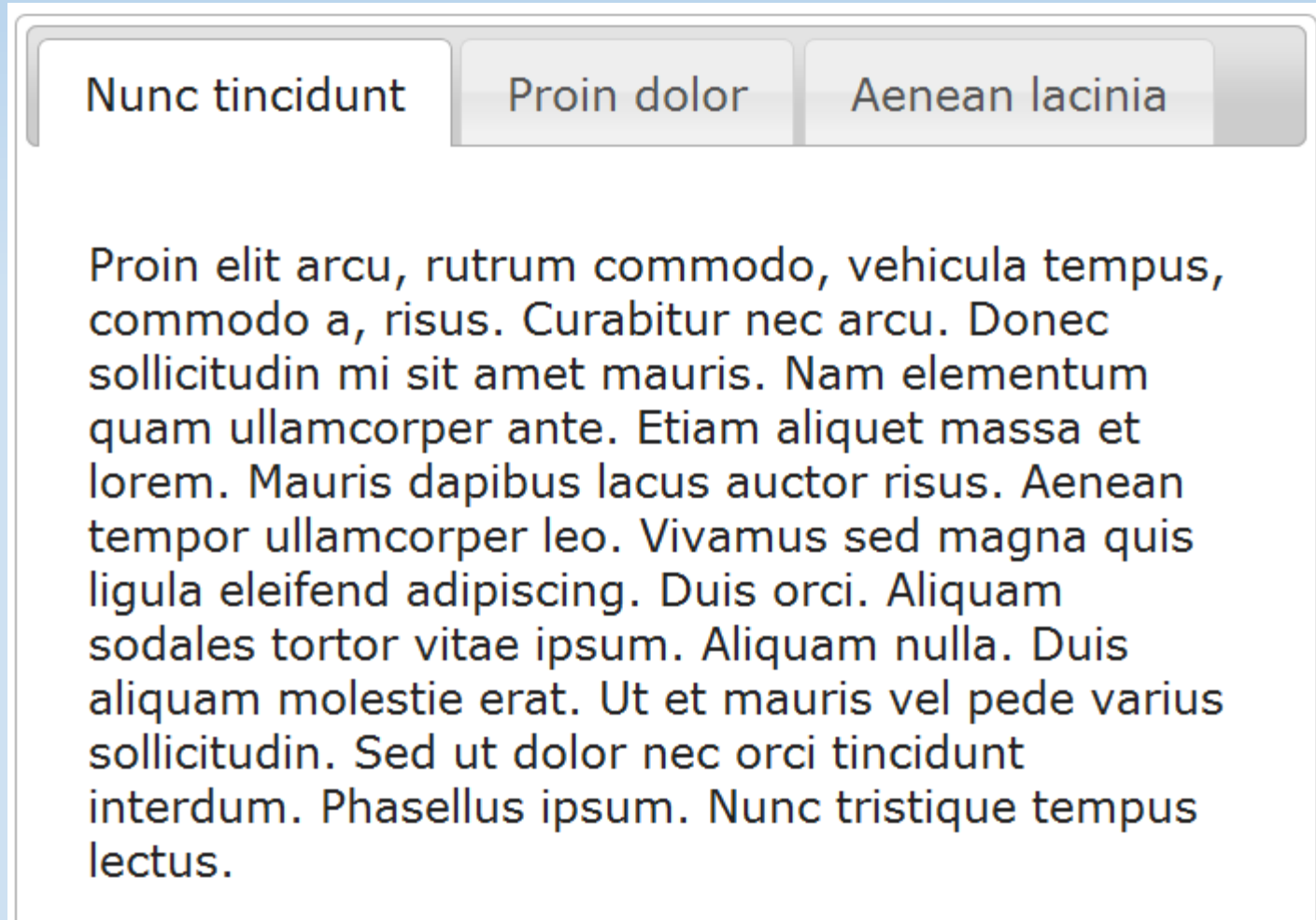
Widget: Spinner

Select a value:

5



Widiget: Tabs



Widget: Date Picker

Date:

May 2013

Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

Come si Usano i Widget

- I widget hanno normalmente una impostazione di default
- In genere, basta attivarne uno sull'elemento DOM esteso selezionato
- Volendo, si possono impostare delle proprietà

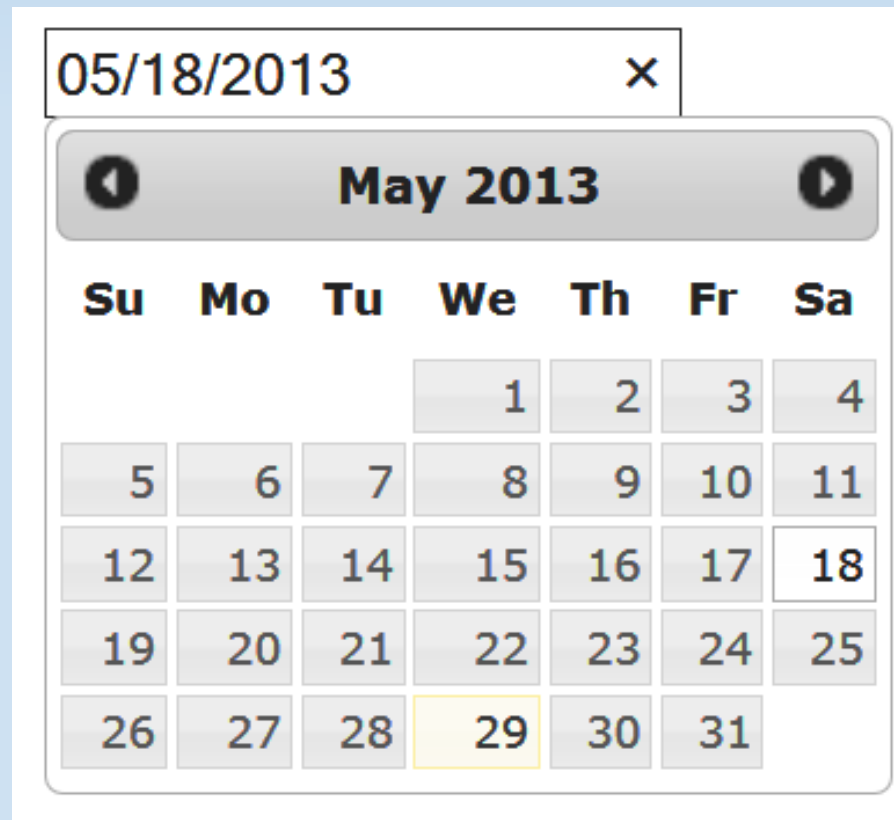
Esempio: DatePicker

```
$ (function() {  
    $ ( ' #mydate ' ) .datepicker ( ) ;  
} ) ;
```

dove `mydate` è l'id di un campo di input di tipo text

Esempio: DatePicker

- Cliccando sul campo, il DatePicker compare, ma il formato della data a noi non va bene



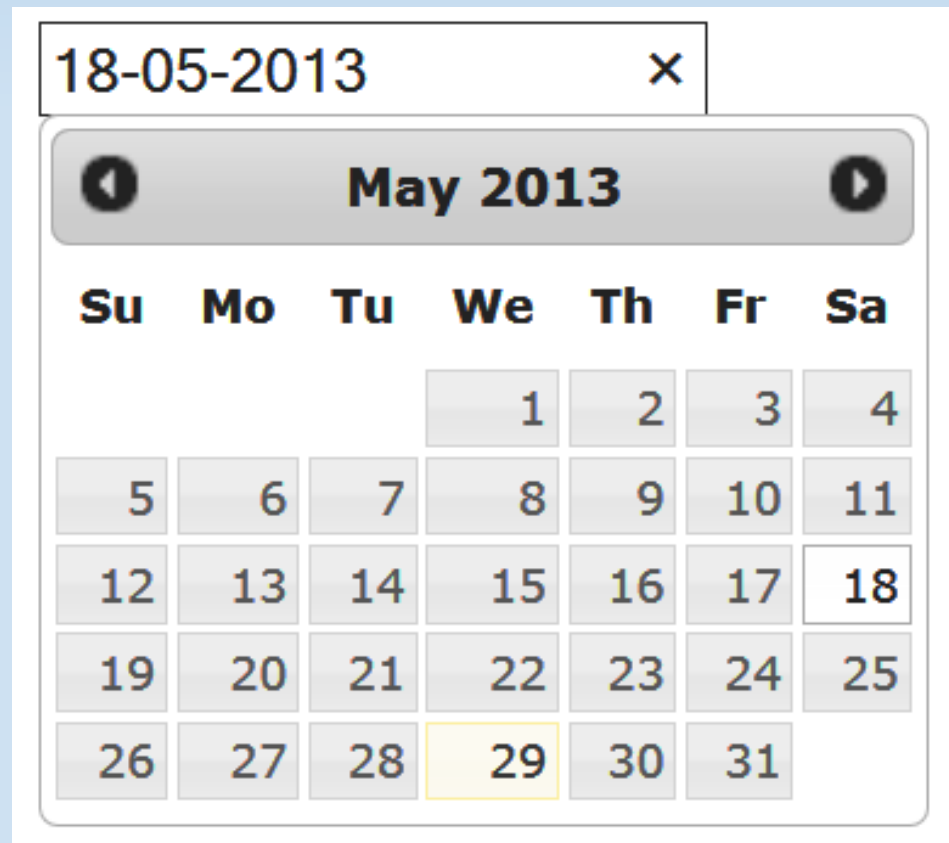
Specificare il Formato della Data

```
$ (function() {  
    $ ( ' #mydate ' ) .datepicker (  
        { dateFormat: "dd-mm-yyyy" } ) ;  
    } ) ;
```

Si crea un oggetto di configurazione al volo

Esempio: DatePicker

- Cliccando sul campo, il DatePicker compare



Altri Framework

Bootstrap

- Lo scopo di Bootstrap è gestire lo stile della pagina in modo uniforme
- Rendendo la pagina «Responsive»
- Responsive vuol dire che l'organizzazione dei contenuti si adatta automaticamente alla dimensione dello schermo e al livello di zoom

Bootstrap

- È basato su JavaScript
- Ma non va programmato, perché il codice JavaScript lavora sotto-traccia
- Si devono impostare stili e regole di posizionamento usando attributi aggiuntivi non standard degli elementi
- Questi attributi vengono usati da Bootstrap per gestire la visualizzazione

AngularJS

- È un framework molto popolare
- Si occupa di gestire la struttura dell'interfaccia
- Ma soprattutto introduce astrazioni di programmazione che dovrebbero semplificare la scrittura di azioni complesse
- Per essere efficace, va conosciuto molto bene
- Fare qualche prova, a tempo perso, non fa male