1.5 Definizione del modello

Nella sezione sono descritte le componenti di MVC5 che permettono la gestione delle entità attraverso CSBlog e lo sviluppo dei relativi test di accettazione.

Listato 1.22: Funzionalità dell'iterazione.

```
1 Funzionalità: Gestione dei post
2   Come Autore
3   Vorrei poter inserire, leggere, modificare e rimuovere dei post
        su RBlog
4   Per poter documentare la tesi
```

1.5.1 Dipendenze

Anche per i test in CSBlog è necessario provvedere alla regressione del modello, rimuovendo dal database le entità create per verificare uno scenario. Come descritto nelle sezioni precedenti, con Specflow e Coypu sono stati definiti due hook per la creazione e rilascio della sessione del browser, che assicurano l'esecuzione dei test in nuove finestre e la rimozione di ogni cookie e sessione HTTP relativi a precedenti azioni.

Per completare il processo di regressione è necessario provvedere a rimuovere anche eventuali post creati negli scenari. La libreria di testing prevede l'utilizzo del tag "@clear" per individuare le funzionalità o gli scenari che introducono ma non eliminano nuovi contenuti sul blog.

Nel successivo listato è mostrato l'intera callback eseguita alla conclusione degli scenari, indipendentemente dal risultato delle asserzioni.

Listato 1.23: Callback relativa alla terminazione degli scenari.

```
1  [AfterScenario]
2  public void AfterScenario(){
3    if(ScenarioContext.Current.ScenarioInfo.Tags.Contains("clear") ||
         FeatureContext.Current.FeatureInfo.Tags.Contains("clear")){
4      login();
5      string LoremIpsumTitle = "Lorem Ipsum";
6      string xpathQuery = String.Format("//div[@class = 'post'][p/a[
           contains(text(),'{0}')]]", LoremIpsumTitle);
7      browser.Visit("/");
8      foreach (var post in browser.FindAllXPath(xpathQuery)) {
9        post.FindCss(".remove_post_button").Click();
10       browser.ClickButton("Confermi la rimozione?");
11     }
12   }
13   _browser.Dispose();
14 }
```

L'attributo "AfterScenario" accetta come parametri un numero variabile di stringhe rappresentanti i tag per specificare per quali scenari debba essere eseguito il metodo. In Cucumber l'ordine di esecuzione degli hook coincide con l'ordine di registrazione dei metodi, mentre in Cucumber-JVM, che sfrutta il

meccanismo delle annotazioni, è possibile definire un attributo per indicare la priorità di ciascuna callback.

Al contrario in Specflow, non è possibile specificare alcun ordinamento ed invece di definire due callback, rispettivamente per la rimozione di eventuali post presenti nel blog e la conclusione della sessione di test, è necessario all'interno di un generico hook verificare manualmente la presenza del tag "@clear" per la funzionalità o lo scenario corrente ed eventualmente effettuare la rimozione dei post.

Per implementare gli hook in SpecFlow è necessario utilizzare gli attributi per associare le callback agli eventi e implementare i metodi all'interno di classi annotate con l'attributo "Binding", come per la definizione delle classi contenenti i passi.

1.5.2 Gestione dei form

Nella sottosezione corrente è descritta l'implementazione delle componenti necessarie per l'implementare le funzionalità per la creazione di un nuovo post. Come specificato nell'introduzione, VS permette la generazione dei controlli, attraverso un wizard di configurazione, e delle viste corrispondenti.

Lo sviluppo si è svolto in maniera veloce, se paragonato a Spring, potendo sfruttare una struttura già esistente e dovendo esclusivamente personalizzarne il comportamento.

I controlli

Ad ogni componente è associato un controllo ed una collezione di azioni eseguibili, nei seguenti frammenti di codice sono mostrate le implementazione delle azioni per la creazione dei post.

Listato 1.24: Frammento di codice contente l'azione "Create" per la visualizzazione della relativa vista.

```
1 public class PostController : Controller {  private CSBlogEntities
      db = new CSBlogEntities();
2   public ActionResult Create(){
3     if (!isAuthorized())
4       return redirectToLoginPage();
5     return View();
6   }
7   /*..*/
```

Come in RoR, sono presenti due metodi "Create" per disaccoppiare la gestione della vista per la creazione di un nuovo post e la gestione della persistenza dell'oggetto definito attraverso l'interfaccia dell'applicazione nel modello.

Il ritorno di tipo "View", istanziato senza specificare parametri, risolve la richiesta visualizzando la vista "/Post/Create.cshtml", il cui nome coincide con l'identificatore dell'azione.

Listato 1.25: Frammento di codice contente l'azione "Create" persistenza del nuovo oggetto creato.

```
1  // POST: /Post/Create
2  [HttpPost]
3  [ValidateAntiForgeryToken]
4  public ActionResult Create([Bind(Include = "title,body")] Post post
       ){
5    if (!isAuthorized())
6      return redirectToLoginPage();
7    if (ModelState.IsValid){
8      post.id = Guid.NewGuid();
9      post.createdAt = post.updatedAt = DateTime.Now;
10     post.authorId = (Guid)Session["author_id"];
11     db.Posts.Add(post);
12     db.SaveChanges();
13     TempData["notice"] = String.Format("Il post '{0}' è stato
           creato con successo.", post.title);
14     return RedirectToAction("Details", new { post.id });
15   }
16   return View(post);
17 }
```

Nonostante la creazione di nuovi post sia stata suddivisa in due azioni, il metodo "Create" è piuttosto articolato. Di seguito sono descritte le principali funzionalità:

- l'espressione booleana "ModelState.IsValid" verifica che i vincoli definiti a livello di logica dell'entità siano soddisfatti e si possa procedere nell'esecuzione delle istruzione per rendere persistenti i dati inseriti dall'utente;

- l'attributo "ValidateAntiForgeryToken" protegge l'applicazione, ed in particolare le pagine che permettono la definizione di nuovi contenuti, da attacchi di over-posting e dalla forgiatura di richieste cross-site. Il processo per implementare la sicurezza dei form è abbastanza semplice e consiste con la generazione di un valore unico all'interno del form durante la visita della pagina e la copia del valore all'interno di un cookie HTTP creato ad-hoc, al momento dell'invio dei dati inseriti è verificato da MVC5 l'uguaglianza dei due valori eventualmente permettendo l'operazione;

- l'attributo "Bind" del parametro "post" specifica quali campi del form devono essere utilizzati per inizializzare l'oggetto, gli attributi della classe "Post" comprendono anche altre informazioni che sono però inizializzate secondo la logica dell'applicazione;

- la proprietà "Session" permette l'accesso ai valori presenti nella corrente sessione HTTP, in cui è mantenuto un riferimento all'identificatore dell'utente che ha effettuato l'autenticazione;

- "db.Posts.Add" e "db.SaveChanges" aggiungono l'oggetto al modello ed effettuano la persistenza delle modifiche.

- il metodo "RedirectToAction" restituisce un'istanza di tipo "RedirectToRouteResult" per effettuare la ridirezione della richiesta dopo aver creato il post. Combinando l'uso della ridirezione con il dizionario "TempData" è possibile fornire alla vista, che verrà visualizzata al termine dell'azione corrente, delle espressioni il cui valore sarà mantenuto per una sola richiesta HTTP all'interno della sessione;

- Nel caso in cui la validazione degli attributi del post non sia andata a buon fine, il post restituisce un riferimento alla vista "Create" fornendo lo stesso parametro ricevuto in input per inizializzare i campi del form;

Come in Spring, in cui è possibile sfruttare i parametri di tipo "Model" per passare degli oggetti alla vista, in MVC5 è disponibile la proprietà "dynamic" "ViewBag" presente nella super classe "Controller". Il tipo "dynamic" in C# posticipa le operazioni di type-checking normalmente eseguite a compile time al momento dell'esecuzione, semplificando l'utilizzo delle proprietà come in questo caso.

Listato 1.26: Esempio di utilizzo della proprietà "ViewBag".

```
1 ViewBag.message = "Hello ViewBag";
```

Nel listato è inizializzato con una stringa l'attributo dinamico "message", che sarà utilizzabile nelle viste in Razor.

Le viste

Dopo aver mostrato i controlli per la creazione di nuovi post, è ora descritta la vista per la creazione di nuovi post.

```
1  @model Blog.Models.Post
2  @{
3    ViewBag.PageTitle = ViewBag.Title = "Crea un post";
4  }
5  @using (Html.BeginForm()){
6  @Html.AntiForgeryToken()
7  <div class="form-horizontal">
8    @Html.ValidationSummary(true)
9    <div class="form_actions">
10     <input alt="Scrivi un nuovo post" id="submit" src="@Url.Content
           ("~/Images/save_48.png")"                    type="image"
           value="Create" />
11   </div>
12   <p id="form_title">
13     Scrivi un nuovo post
14   </p>
15   <div class="form-group">
16     @Html.LabelFor(model => model.title, new { @class = "control-
           label" })
17     <div>
18       @Html.EditorFor(model => model.title)
19       <div class="error_explanation">
20         @Html.ValidationMessageFor(model => model.title)
```

```
21          </div>
22        </div>
23      </div>
24      <!-- .. -->
```

In MVC, tramite le funzionalità di Razor è possibile tipare le proprie viste indicando, tramite la sintassi "@model", quale l'entità rappresentata dalla pagina corrente. Nella classe "Controller" sono presenti le funzionalità per istanziare degli oggetti di tipo "ViewResult" fornendo un istanza di "object" che rappresenterà il modello, come mostrato nel listato 1.25 in caso di fallimento della validazione.

Nella vista sono utilizzate diverse funzionalità di Razor:

- nella parte iniziale della vista sono inizializzati alcuni valori della proprietà "ViewBag" utilizzati all'interno della vista rappresentante il layout per impostare il titolo della pagina e dell'intestazione di CSBlog;
- il metodo "BeginForm", della classe "FormExtensions" di Razor, genera il codice HTML relativo all'elemento "form" e permette di utilizzare altri metodi per la definizione delle varie componenti del form;
- il metodo "ValidationSummary" restituisce le informazioni sulla validazione, nel caso in cui la vista attuale sia visualizzata come conseguenza del fallimento delle operazioni di persistenza;
- i metodi "LabelFor", "EditorFor" e "TextAreaFor" generano gli elementi di input del form e i relativi "label" HTML;
- il metodo "ValidationMessageFor" ritorna i messaggi d'errore che descrivono i problemi riscontrati durante la validazione dell'entità, opportunamente formattati in HTML;

Gestione degli errori nei form

Utilizzando gli attributi dell'EF è possibile definire dei vincoli sugli attributi delle entità. Nel successivo listato è mostrato un frammento di codice relativo alla proprietà "title" dell'entità "Post".

```
1 [Required(ErrorMessage = "Titolo mancante.")]
2 [StringLength(100, MinimumLength = 5, ErrorMessage = "Il titolo
      deve essere compreso fra 5 e 100 caratteri.")]
3 [Remote("CheckForDuplication", "Post", AdditionalFields = "id")]
4 [Display(Name = "Titolo", Description = "Inserisci in questo campo
      il titolo che vuoi dare al tuo articolo.")]
5 public string title { get; set; }
```

Gli attributi "Required" e "StringLength" effettuano la verifica del valore della stringa e invalidano l'oggetto nel caso il titolo sia mancante, troppo corto o troppo lungo; è anche possibile associare a ciascun attributo un messaggio d'errore personalizzato che rappresenta il testo mostrato all'interno della vista.

Per effettuare operazioni di validazione come la verifica d'unicità del titolo, non è possibile sfruttare gli attributi già definiti in EF. Per verificare la presenza di altri post con il titolo simile all'interno del blog è effettuata una richiesta HTTP asincrona tramite JQuery, definita tramite l'attributo "Remote" ed i parametri "CheckForDuplication", "Post" e "id" che rispettivamente rappresentano l'azione e il controllo che gestiranno la richiesta e il parametro aggiuntivo "id"[5].

Nel listato successivo, utilizzando LINQ per accedere al modello e JSon per definire il risultato dell'azione, è verificato in funzione dei parametri la presenza di eventuali post con titolo simili.

Listato 1.27: Azione per la verifica dell'unicità del titolo dei post.

```
1  [HttpGet]
2  public JsonResult CheckForDuplication(Guid? id, string title){
3    var post = db.Posts.Where(p => p.title.Equals(title,
         StringComparison.CurrentCultureIgnoreCase)).FirstOrDefault();
4    if (post != null && (id == null || !post.id.Equals(id)))
5      return Json("Il titolo è già presente.", JsonRequestBehavior.
           AllowGet);
6    }
7    return Json(true, JsonRequestBehavior.AllowGet);
8  }
```

L'azione restituisce un valore serializzato tramite JSon, una stringa contenente il messaggio in caso di fallimento della validazione, oppure il valore "true" se il titolo è utilizzabile.

Implementazione dei test

Oltre alle funzionalità per la navigazione, la classe "BrowserSession" di Coypu fornisce le funzionalità per la gestione dei form e la selezione degli elementi del DOM in funzione del tipo e del testo mostrato.

Listato 1.28: Login su CSBlog tramite Coypu.

```
1  [Given(@"mi autentico come ""(.*)""")]
2  [When(@"mi autentico come ""(.*)""")]
3  public void DatoMiAutenticoCome(string email){
4    string password = "password";
5    browser.ClickLink("Login");
6    browser.FillIn("Email").With(email);
7    browser.FillIn("Password").With(password);
8    browser.ClickButton("Login");
9  }
```

Rispetto a Selenium, il DSL di Coypu è più semplice e facilmente utilizzabile; inoltre è evidente la propensione della libreria a favorire la definizione di selettori in funzione del testo mostrato dai vari elementi.

Queste pratiche permettono la definizione di test leggibili e di semplice comprensione, semplificando la manutenzione del codice.

[5]La verifica dell'unicità del titolo del post è effettuata sia alla creazione che alla modifica di un post, il parametro aggiuntivo "id" può assumere valore nullo ed è utilizzato per associare il titolo ad un eventuale post esistente.

Listato 1.29: Definizione di un selettore tramite i metodi di Coypu.

```
1 protected ElementScope findPostByTitle(String title) {
2   return browser.FindAllCss(".post").First(p => p.FindLink(title,
        new Options { TextPrecision = TextPrecision.Substring }).
        Exists());
3 }
```

Il metodo del listato precedente definisce una funzionalità ausiliaria per ricercare all'interno del DOM un post, contenuto all'interno di un "div" con attributo di "class" uguale a "post", il cui titolo, rappresentato da un collegamento, contenga parte del testo indicato dal parametro.

Il metodo è una funzionalità utile per individuare il l'elemento HTML che contiene le informazioni visualizzate per un certo post, il cui risultato è utilizzato per effettuare ulteriori accessi all'HTML interno.

Nelle versioni in Ruby e Java, lo stesso metodo è stato definito utilizzando un selettore ed un'espressione XPath. Con Coypu e le funzionalità dell'interfaccia "IEnumerable", che permettono l'uso di funzioni, è possibile definire query articolate ma allo stesso tempo comprensibili. Inoltre la definizione di selettori attraverso una concatenazione di metodi riduce la fragilità dei metodi semplificando le manutenibilità del codice e permette la verifica durante la compilazione rispetto ad una query XPath rappresentata in una stringa.

Selettori

In Coypu sono implementati una buona varietà di metodi per l'implementazione di funzionalità per la definizione di selettori ed in generale effettuare ricerche all'interno del DOM. Oltre i metodi specifici per un tipo di elemento, come "FindLink" già mostrato in precedenza, sono presenti anche funzionalità più simili a quelle viste per Selenium.

Di seguito sono proposti alcuni esempi e frammenti dei passi contenti alcuni dei metodi esistenti nella libreria.

Listato 1.30: Esempio del metodo "FindCss".

```
1 browser.FindCss(".post_title a", text: title).Exists();
```

In maniera simile a Capybara, ed in particolare ai metodi del modulo "Finders", nel metodo "FindCSS" è possibile specificare delle stringhe o delle espressioni regolari oltre alle regole del CSS.

Listato 1.31: Esempio del metodo "FindId".

```
1 protected ElementScope header {get{return browser.FindId("header")
    ;}}
```

Oltre alle funzionalità classiche delle librerie di automazione web, come il metodo "FindId", Coypu cerca di fornire delle funzionalità ulteriori tramite parametri opzionali o implementando metodi derivati come "FindIdEndingWith".

Listato 1.32: Esempio del metodo "FindAllCss".

```
1 public void DatoIlPostNonELeggibileSuCSBlog(string title){
2   browser.Visit("/");
3   var posts = browser.FindAllCss(".post");
4   foreach (var post in posts) {
5     Assert.That(!post.FindLink(title).Exists());
6   }
7 }
```

A differenza di Capybara, Coypu non restituisce errore nel caso in cui la venga utilizzato un metodo per individuare un singolo elemento ed al contrario siano presenti più potenziali risultati, ma restituisce il primo elemento incontrato.

Per gestire collezioni di elementi sono disponibili metodi come "FindAllCss"; rispetto ai metodi che restituiscono un singolo risultato, i metodi che ricercano tutti i potenziali riscontri all'interno della pagina non applicano strategie di attesa per eventuali elementi asincroni, bensì restituiscono la collezione di elementi presenti al momento dell'invocazione; per modificare il comportamento è possibile definire un predicato per descrivere lo stato atteso.

Listato 1.33: Esempio del metodo "FindXPath".

```
1 [Then(@"l_intestazione è posizionata all_inizio")]
2 public void AlloraLInizio(){
3   var first = browser.FindXPath("//body/*[1]");
4   Assert.True(first.Exists());
5   Assert.True(header.Exists());
6   Assert.AreEqual(first.Id, header.Id);
7 }
```

Nonostante la libreria propenda per l'utilizzo di funzionalità ad alto livello, per favorire la leggibilità dei test, è anche possibile definire i seletori tramite espressioni XPath.

```
1 protected ElementScope FindNotice(string noticeMessage){
2   var notice = browser.FindId("notice", new Options { TextPrecision
        = TextPrecision.Exact});
3   Assert.That(notice.Exists());
4   return notice;
5 }
```

In alcune parti della libreria si può notare la forte ispirazione che Capybara ha dato al progetto, ad esempio i metodi permettono l'uso di un parametro "Options," che ricorda l'uso del dizionario come parametro opzionale in Ruby.

Le istanze della classe "Options" permettono di modificare il comportamento dei metodi, specificando la strategia di attesa e di polling, il tipo di confronto sulle stringhe da utilizzare, la cardinalità attesa e molti altri parametri.

1.5.3 Debug con Specflow

Durante lo sviluppo di CSBlog con VS sono state individuate alcune difficoltà nell'applicare la tecnica dell'ATDD a causa dell'impossibilità di eseguire il

debug dei test. All'interno dell'ambiente di sviluppo è possibile eseguire la propria applicazione sia in modalità debug classica, in cui VS monitora l'esecuzione ed eventualmente gestisce e segnala gli errori, oppure è possibile procedere all'esecuzione semplice.

Dai tentativi fatti su VS, per qualsiasi sia il tipo di esecuzione dell'applicazione sembra che non sia possibile eseguire i propri test di accettazione in modalità debug, perché è consentito che venga eseguito al più un solo processo per ogni istanza di VS.

Creando un secondo progetto contenente esclusivamente i test ed aprendo due istanze di VS, una per lo sviluppo ed una per il testing, si dovrebbe evitare i conflitti durante l'esecuzione, introducendo però possibili rallentamenti nel sistema.

1.6 Login & Autorizzazione

Listato 1.34: Autenticazione in CSBlog.

```
1  @cap6
2  Funzionalità: Autenticazione su SBlog
3    Come Autore di SBlog
4    Vorrei che alcune operazioni sensibili siano permesse previa
         autenticazione
5    Per poter garantire l'autenticità dei contenuti
```

Come per le funzionalità riguardanti l'analisi del CSS e delle proprietà estetiche dei nodi dell'HTML, Coypu non implementa metodi specifici per la gestione dei cookie e delle sessioni.

E' comunque possibile utilizzare le funzionalità native per effettuare operazioni sui cookie e le sessioni HTTP come già descritto nel capitolo su SBlog.

1.6.1 Accesso ai singoli attributi

La classe "ElementScope" di Coypu ridefinisce il comportamento dell'operatore "[ ]": come in Capybara, è possibile accedere alle singole proprietà indicando tramite una stringa l'identificatore desiderato. Accedere alle proprietà di un elemento HTML come se fosse un array associativo presenta alcuni vantaggi: la sintassi è facilmente leggibile e comprensibile, inoltre, dal punto di vista dell'implementazione, è facilmente estendibile nel caso il W3C prevedesse delle variazioni nell'HTML, come è ad esempio accaduto per la versione HTML5.

1.7 Asincronia

Coypu gestisce gli elementi della pagina con lo stesso principio di Capybara, ogni elemento può potenzialmente essere asincrono, preferendo la definizione di una strategia di ricerca globale tramite attese e polling rispetto alla definizione di asserzioni sullo stato degli elementi, perché difficili da esprimere, implementare e mantenere.

1.7.1 JavaScript

Listato 1.35: Introduzione di un breve script Javascript.

```
1  Funzionalità: Easter Egging
2    Come Sviluppatore
3    Vorrei che nel blog fosse presente un mio logo
4    Per firmare il mio lavoro
5
6    Contesto:
7    Dato apro CSBlog
8    Scenario: EasterEgg
9    Dato non è presente il logo nell'intestazione
10   Quando clicco sull'area del pié di pagina
11   Allora è presente il logo
```

Nei successi listati è mostrata l'implementazione dello scenario "EasterEgg". Rispetto alle attese esplicite di Selenium e a Capybara che sfrutta il metodo "synchronize" per verificare il completamento delle operazioni asincrone, l'implementazione è nettamente più compatta e leggibile; inoltre non è presente alcuna particolarità rispetto ai test descritti nelle precedenti sessioni che suggerisca che il piè di pagina implementi un comportamento asincrono.

Listato 1.36: Pre-condizione.

```
1  [Given(@"non è presente il logo nell'intestazione")]
2  public void DatoNonEPresenteIlLogoNellIntestazione(){
3    Assert.That(footer, Shows.No.Css("img"));
4  }
```

Listato 1.37: Evento.

```
1  [When(@"clicco sull'area del pié di pagina")]
2  public void QuandoCliccoSullAreaDelPieDiPagina(){
3    footer.Click();
4  }
```

Listato 1.38: Asserzione.

```
1  [Then(@"è presente il logo")]
2  public void AlloraEPresenteIlLogo(){
3    var footer = base.footer;
4    Assert.That(footer, Shows.Css("img"));
5    Assert.That(footer.FindId("woodstock").Exists());
6  }
```

1.7.2 Scenari sull'auto-completamento con JQuery UI

Listato 1.39: Scenario sulla ricerca nei post.

```
1  @cap5
2  Funzionalità: Ricerca fra i post
3    Come Lettore
4    Vorrei poter ricercare i post su RBlog
5    Per poter navigare fra i contenuti più velocemente
```

Nella funzionalità corrente è verificato il widget per l'auto-complemento del menu della ricerca, implementato tramite JQuery UI ed una chiamata AJAX.

Listato 1.40: Scenario riguardante l'auto-completamento della ricerca.

```
1  Scenario: Autocompletamento della ricerca
2    Dato nell'intestazione è presente la barra di ricerca
3    Dato il post "Lorem Ipsum" esiste
4    Quando inserisco il testo "lor" da ricercare
5    Allora viene proposto il post "Lorem Ipsum"
6    Quando inserisco il testo "xyz" da ricercare
7    Allora non è proposto alcun post
8    ...
```

Nei successivi listati sono incluse le implementazioni dei passi che verificano la presenza del menù a tendina per il campo "input" della ricerca.

Listato 1.41: Verifica della presenza di un post nei suggerimenti.

```
1  [Then(@"viene proposto il post ""(.*)""")]
2  public void AlloraVienePropostoIlPost(string title){
3    var suggestion = browser.FindCss(".ui-menu-item", title);
4    Assert.That(suggestion.Exists());
5  }
```

Coypu anche per uno scenario più complesso, in quanto questa operazione asincrona ha maggior latenza rispetto alla risoluzione della chiamata Javascript dello scenario "EasterEgg", non necessità di introdurre istruzioni ad-hoc. Inoltre Coypu è coerente nell'implementazione delle proprie funzionalità e restituisce solo elementi effettivamente visibili nella pagina[6], al contrario utilizzando Selenium per questo stesso test è stato necessario verificare l'effettiva visualizzazione dell'elemento "ui-menu-item" tramite il metodo "isDisplayed" dell'interfaccia "WebElement" a causa del comportamento particolare del widget.

Listato 1.42: Verifica dell'assenza di suggerimenti.

```
1  [Then(@"non è proposto alcun post")]
2  public void AlloraNonEPropostoAlcunPost(){
3    Assert.That(browser, Shows.No.Css(".ui-menu-item"));
4  }
```

[6]Tramite la configurazione della sessione di navigazione, è possibile richiedere che siano individuati anche gli elementi non visibili.

Il widget per l'auto-completamento, probabilmente per minimizzare il numero di operazioni di modifica del DOM, applica un meccanismo paragonabile al caching. Invece di rimuovere le opzioni proposte dalla pagina, che corrispondono ai punti di una lista numerata in HTML, quando il focus è spostato su un altro elemento della pagina, JQuery UI rende non visibile l'intera lista e ne mantiene gli elementi nel DOM.

Quando sarà effettuata una nuova ricerca all'interno della pagina, il widget provvederà a correggere la cardinalità della lista e sostituire il testo degli elementi con i nuovi valori.