Capitolo 1

MVC 5

1.1 ASP.NET MVC 5

In questo capitolo è trattato lo sviluppo di CSBlog, un'applicazione web sviluppata tramite il framework ASP.NET MVC 5 e , l'implementazione della libreria di test di accettazione già mostrata nei precedenti capitoli utilizzando Specflow, una libreria per il BDD ispirata a Cucumber, e Coypu per automatizzare la navigazione del browser web.

In questo capitolo saranno descritte le particolarità del framework, l'implementazione delle diverse componenti e gli strumenti utilizzati. Dalla sezione 1.3 sono documentate le diverse funzionalità presenti nella libreria dei test di accettazione e descritto il processo di testing, includendo esempi e frammenti di codice.

1.1.1 Visual Studio 2013

Per lo sviluppo di CSBlog, dei test di accettazione e la gestione del modello è stato utilizzato Visual Studio 2013 Ultimate, VS, che integra le diverse funzionalità per lo sviluppo web, la configurazione della propria applicazione e l'implementazione delle diverse componenti del pattern.

Tutti i plugin necessari, come per Specflow, sono stati installati tramite NuGet, il gestore di pacchetti per il mondo .NET, direttamente tramite l'ambiente di sviluppo. VS è un ambiente di sviluppo con enormi potenzialità, articolato e complesso ma, nel corso delle prossime sezioni, con numerose funzionalità per il supporto alla configurazione e generazione del progetto, come sarà mostrato nelle prossime sezioni.

1.2 L'interpretazione di ASP.NET del pattern MVC

In questa sezione è descritto il processo di definizione dell'architettura di CSBlog, caratterizzato da un uso costante degli strumenti di VS per la definizione delle varie componenti. Al termine della sezione risulterà evidente come, nonostante il framework MVC5 e l'intero progetto ASP.NET sia estremamente articolato, lo sviluppo non risente delle grandi potenzialità messe a disposizione.

1.2.1 Il modello

Per definire il modello con RoR e Spring è stato scelto di definire prima le entità, rispettivamente tramite gli ActiveRecord e JPA, ed utilizzare le classi per effettuare la generazione dei database su PostgreSql. RoR genera di default tre database e le tabelle delle migrazioni, ed anche JPA implementa ad alto livello molte funzionalità dei database relazioni, come i vincoli di integrità. E' quindi molto conveniente per lo sviluppatore progettare la persistenza partendo dalle classi che rappresentano il dominio.

Per MVC5 è stato scelto invece di definire il modello in maniera "classica" utilizzando il DDL di SQL -come database è stato scelto SQL Server Express, la versione gratuita del database sviluppato da Microsoft-, la generazione guidata e gli strumenti grafici per la gestione disponibili per creare le tabelle "Post" ed "Autore" e la relazione presente fra le due entità in breve tempo.

Entity Framework

Per implementare l'interfaccia del modello di MVC5 è stato scelto l'Entity Framework, attualmente alla versione 6, che implementa l'astrazione del modello attraverso la tecnica ORM in maniera simile a quanto già visto per RoR e Spring. Il framework si integra con ADO.NET, responsabile dell'interazione con i sistemi di persistenza esistenti e del mantenimento della consistenza fra le entry e gli oggetti.

Le classi che descrivono le entità dell'EF estendono "DbContext", che fornisce le funzionalità per eseguire interrogazioni, tener traccia dei cambiamenti allo stato delle istanze ed invocare le operazioni di creazione, aggiornamento e cancellazione. Visual Studio include un wizard per la configurazione del modello, che provvede a definire la connessione al database, a generare le classi parziali[1] rappresentanti le entità e a configurare le risorse necessarie per il funzionamento dell'EF.

Listato 1.1: Il contesto di CSBlog generato da VS.

```
1 public partial class CSBlogEntities : DbContext{
```

[1]Una classe parziale in C# rappresenta una parte di una classe intera. La definizione completa è suddivisa in più fine che saranno unite a tempo di compilazione.

```
2    public CSBlogEntities() : base("name=CSBlogEntities"){}
                protected override void OnModelCreating(
        DbModelBuilder modelBuilder){
3      throw new UnintentionalCodeFirstException();
4    }
5    public virtual DbSet<Author> Authors { get; set; }
6    public virtual DbSet<Post> Posts { get; set; }
7  }
```

Le proprietà "Authors" e "Posts" rappresentano le entità presenti nel database e sono utilizzate dall'EF per effettuare le query. Per implementare le interrogazioni necessarie per CSBlog è stato utilizzato LINQ, Language-Integrated Query.

Listato 1.2: Proiezione dei titoli dei post con LINQ.

```
1  CSBlogEntities db = new CSBlogEntities();
2  var postTitles = db.Posts.Select(p => p.title).ToList();
```

LINQ permette di definire le proprie query attraverso un insieme di metodi che rispecchiano le funzionalità di SQL e permettono di utilizzare il paradigma ad oggetti, ad esempio la proiezione per ottenere i titoli dei post esistenti è definita effettuato un accesso all'attributo "title" dell'entità "Post". Utilizzando una libreria che fornisce una tale astrazione del modello e delle entità è possibile definire le proprie interrogazioni sfruttando tutte le funzionalità esistenti per C# in VS, come l'auto-completamento e l'evidenziatura dei diversi elementi sintattici. Inoltre LINQ definisce query utilizzando metodi generici che permettono di verificare a compile-time il corretto utilizzo dei tipi e degli attributi presenti nelle interrogazioni.

In maniera simile all'interfaccia per la definizione delle query con gli Active Record di RoR, lo sviluppatore può sfruttare l'astrazione introdotta dall'uso dell'EF e definire le proprie query in maniera veloce ed intuitiva. Rispetto però alla controparte in Ruby, LINQ è più completo permettendo la definizione di interrogazioni anche su altri sistemi per la persistenza, come documenti XML, ed anche su collezioni di tipo "IEnumerable".

La semplice query di proiezione dell'esempio è suddivisibile in tre elementi: l'ottenimento del contesto tramite le classi dell'EF, rappresentato dall'oggetto "db", per specificare quale sia l'entità utilizzata nell'interrogazione, la specifica delle operazioni, come il metodo "Select", e l'esecuzione che coincide con l'invocazione del metodo "ToList"[2]. L'oggetto "postTitles" ha tipo IQueryable, interfaccia generica sul tipo dell'interrogazione che estende IEnumerable e dichiara le funzionalità per compiere la valutazione della query.

[2]In generale ogni invocazione che comporta un accesso esplicito agli oggetti coincide con l'esecuzione della query, come il metodo "ToList", le funzionalità di LINQ "First" o "Any" o l'iterazione tramite il costrutto "for".

Oltre alla generazione dell'interfaccia del modello, VS include un browser per visualizzare lo schema delle entità esistenti e le relazioni presenti. Tramite questo strumento, direttamente nell'ambiente di sviluppo, è possibile visualizzare e aggiornare la struttura del modello, sincronizzare gli schemi dei database utilizzati, ed anche gestire associazioni e relazioni di ereditarietà.

1.2.2 I controlli

Come per la definizione del modello, anche per i controlli e le relative azioni, sono stati utilizzati gli strumenti di VS per la generazione automatica delle componenti. Il wizard per la configurazione è ben strutturato e permette di indicare quali azioni sia necessario generare, ad esempio è possibile creare un controllo contenente le operazioni CRUD oppure vuoto, e se utilizzare l'EF per associare il nuovo controllo ad una delle entità e generare delle azioni opportunamente connesse al modello.

Durante il processo di configurazione del nuovo controllo è anche generato un'insieme di viste corrispondenti alle azioni create. In maniera simile a RubyMine, gli strumenti dell'ambiente di sviluppo facilitano lo sviluppo fornendo un'implementazione funzionante per le nuove applicazioni e preziose indicazioni sulla struttura del framework e di come le componenti interagiscano fra loro.

Per semplificare la risoluzione delle richieste HTTP per l'applicazione, MVC5 definisce un file "RouteConfig.cs" all'interno della cartella "AppData".

Listato 1.3: Definizione del pattern per la risoluzione delle richieste.

```
1  public class RouteConfig{
2    public static void RegisterRoutes(RouteCollection routes){
3    /*...*/
4    routes.MapRoute(
5      name: "Default",
6      url: "{controller}/{action}/{id}",
7      defaults: new {
8        controller = "Post",
9        action = "Index",
10       id = UrlParameter.Optional
11       }
12     );
13   }
14 }
```

Il metodo "RegisterRoutes" si occupa di registrare la convenzione scelta per la risoluzione delle richieste effettuate dagli utenti. Come si può osservare dal metodo, il pattern scelto rispecchia l'organizzazione di un'architettura REST: ad esempio l'URL relativo "/Post/Details/id", per visualizzare il contenuto di un singolo post, è suddiviso in tre elementi, il nome del controllo, l'azione corrispondente e l'identificatore dell'elemento. Inoltre è specificato quale sia l'homepage tramite il parametro "default".

Listato 1.4: Visualizzazione di un singolo post.

```
1  namespace Blog.Controllers {
2    public class PostController : Controller     {
3      private CSBlogEntities db = new CSBlogEntities();
4      /*...*/
5      public ActionResult Details(Guid? id){
6        if (id == null){
7          return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
8        }
9        Post post = db.Posts.Find(id);
10       if (post == null){
11         return HttpNotFound();
12       }
13       return View(post);
14     }
15     /*...*/
16   }
17 }
```

Per implementare un nuovo controllo, è sufficiente creare una nuova classe pubblica all'interno del namespace del progetto ed estendere la classe astratta "Controller". Nel frammento della classe "PostController" è mostrata l'implementazione di una singola azione che, in funzione del parametro "id" carica tramite LINQ ed EF l'entità corrispondente e restituisce un riferimento alla vista "Details".

Ad ogni azione, se non specificato diversamente, corrisponde una vista con ugual nome; in particolare, dopo aver individuato il corretto "Post" all'interno del modello, il server web fornirà all'utente la vista corrispondente al file "Views/Details.cshtml". L'azione "Details" accetta richieste HTTP di qualsiasi tipo, per applicare delle limitazioni è sufficiente annotare il metodo con attributi come "HttpGet" o "HttpPost" presenti in MVC5.

Il tipo "ActionResult" utilizzato come risultato dell'azione ha il compito di rappresentare il risultato e permettere al framework fornire una risposta, nel frammento il metodo "View" della classe padre "Controller" istanzia un'oggetto "ViewResult" che specifica la vista da renderizzare.

Rispetto a Spring e RoR, non è prevista all'interno dell'architettura una componente con il ruolo di disaccoppiare i controlli dalla logica per l'accesso al modello, sta quindi all'utente introdurre il pattern che meglio si presta allo scopo ed implementarlo.

1.2.3 Le viste

Per l'implementazione delle viste di CSBlog è stato utilizzato Razor, un linguaggio ASP.NET per l'implementazione di pagine dinamiche che introduce un DSL basato su C# ed integrato in VS, ad esempio è previsto l'IntelliSense e la possibilità di creare nuove pagine sfruttando i wizard presenti nell'ambiente di sviluppo.

Razor rappresenta la scelta di default per lo sviluppo di una nuova applicazione web, introdotto dalla versione 4 del framework, introduce una sintassi per ottimizzare la generazione di pagine HTML, definendo un insieme di costrutti per permettere allo sviluppatore di distinguere facilmente le porzioni di codice dinamico dal resto del documento statico.

Listato 1.5: Hello Razor!

```
1 @{
2   var helloRazor = "Benvenuto su CSBlog!";
3   var weekDay = DateTime.Now.DayOfWeek;
4   var greetingMessage = greeting + " Oggi è: " + weekDay;
5 }
6 <p>@helloRazor</p>
```

Il design delle istruzioni è essenziale ed è paragonabile ad ERB, sono infatti presenti anche in Razor dei delimitatori per includere all'interno delle viste codice. Nel frammento d'esempio è introdotto un blocco contente istruzioni C#, all'interno del quale è creato un messaggio di benvenuto. Tramite la notazione "@variabile" è possibile ottenere il valore della stringa definita nel blocco.

La distinzione fra la sintassi "@{...}" e l'operatore "@", che consente di accedere ai valori e di utilizzarne le espressioni per generare le viste dinamiche, permette di dichiarare viste in cui sono distinte in maniera chiara le porzioni di codice che includono l'implementazione della logica dalle istruzioni dalle istruzioni che esclusivamente utilizzano le funzionalità in lettura.

Listato 1.6: Il ciclo "for" in Razor.

```
1 <ul>
2 @for (int i = 0; i < 10; i++){
3   <li>@i</li>
4 }
5 </ul>
```

La sintassi di Razor include anche i più comuni costrutti sintattici, nell'esempio è mostrato l'utilizzo di un ciclo "for" le cui iterazioni generano nuovi elementi della lista "ul".

L'organizzazione e le convenzioni utilizzate da MVC5 semplifica la definizione dell'intera applicazione, ed anche per la definizione delle viste è previsto un meccanismo per fattorizzare al meglio il proprio codice.

Per ogni vista da processare e visualizzare in risposta ad una richiesta HTTP, viene ricercato il file "_ViewStart.cshtml"[3] all'interno della cartella "Views". In CSBlog è stato utilizzata questa risorsa per indicare quale sia il file Razor contenente il layout delle pagine.

Listato 1.7: Il contenuto della vista _ViewStart.

```
1 @{
2   Layout = "~/Views/Shared/_Layout.cshtml";
3 }
```

[3] "cshtml" è l'estensione delle viste in Razor.

La scelta del layout può essere effettuata dinamicamente, permettendo di aver maggior controllo sull'aspetto delle proprie pagine semplicemente cambiando il percorso della risorsa o aggiungendo della logica all'assegnazione. Ad esempio è possibile definire più versioni del proprio sito, ottimizzandone l'uso in funzione del tipo di device che sta effettuando la navigazione.

Listato 1.8: La struttura delle viste in CSBlog.

```
1  <!DOCTYPE html>
2  <html xmlns:th="http://www.thymeleaf.org">
3    @Html.Partial("_HeadPartial")
4    <body>
5      @Html.Partial("_HeaderPartial")
6      <div id="content">
7      @RenderBody()
8      </div>
9      @Html.Partial("_FooterPartial")
10   </body>
11 </html>
```

Per definire un layout è sufficiente creare una nuova vista con Razor ed utilizzare funzionalità come "Partial", per espandere il contenuto di una vista parziale, o "RenderBody" per espandere la vista restituita dall'azione che ha gestito la richiesta. Nell'esempio è mostrato com'è organizzata una pagina di CSBlog.

1.2.4 Peculiarità

Visual Studio, ASP.NET e l'EF sono software estremamente ricchi di funzionalità e professionali, ma il loro utilizzo è semplificato da VS che, alla creazione di una nuova applicazione, definisce un progetto nel quale sono utilizzate la maggior parte delle componenti presenti nel framework. Navigando fra i sorgenti appena creati è possibile approfondirne le funzionalità, intuirne le potenzialità e osservare le soluzioni proposte dal team di VS.

Oltre ad includere un esempio di applicazione funzionante per ogni nuovo progetto, VS fornisce agli sviluppatori numerosi wizard per generare e configurare le nuove funzionalità. Durante lo sviluppo non è mai stato necessario modificare manualmente i file XML per la configurazione dell'applicazione e l'introduzione di nuovi controlli, viste o entità non ha richiesto uno studio approfondito del framework per una singola funzionalità, come ad esempio per la definizione del modello con JPA.

Inoltre VS integra all'interno dell'ambiente di sviluppo tutte le diverse categorie di strumenti e linguaggi necessari per lo sviluppo di un'applicazione web. Oltre ai classici HTML, CSS e Javascript, è supportato di default lo sviluppo a tecnologie innovative come Sass e Boostrap[4].

[4]Bootstrap è un'interessante framework, sviluppato originariamente da Twitter, per la definizione di pagine web responsive. E' particolarmente apprezzato per la definizione di siti

1.3 Hello CSBlog!

Nella sezione sono introdotti gli strumenti per effettuare la definizione dei test di accettazione automatici per CSBlog. Nella prima parte è descritto il processo di configurazione di Coypu, la libreria per l'automazione scelta per .NET, e di SpecFlow, il framework per il BDD. Successivamente sono introdotti i primi test e le funzionalità utilizzate per l'implementazione.

1.3.1 SpecFlow

SpecFlow è un framework per .NET che sfrutta alcune componenti di Cucumber come il DSL e il relativo parser, entrambe presenti pubblicamente su GitHub. Nonostante utilizzi Gherkin, non è un porting dell'intero framework in C#, al momento disponibile solo nelle versioni Java, Ruby e JavaScript.

Le funzionalità

Listato 1.9: La prima feature di SBlog

```
1 @cap1
2 Funzionalità: Hello RBlog!
3 Per leggere i post e visitare il blog
4 Come Lettore
5 Vorrei che RBlog permettesse la navigazione
```

Grazie all'utilizzo di SpecFlow è possibile riutilizzare la stessa libreria utilizzata per lo sviluppo di RBlog e SBlog. Nell'importazione e nello sviluppo dei post non sono state rilevate problemi legati al linguaggio ed ai costrutti sintattici utilizzati, sono però presenti delle differenze nell'implementazione, come verrà descritto nelle successive sezioni.

Supporto a Specflow in VS Per utilizzare SpecFlow in VS è sufficiente installare tramite il gestore dei pacchetti il relativo plugin. Per eseguire le funzionalità e gli scenari tramite il runner di NUnit è necessario anche un pacchetto aggiuntivo, installabile tramite la console di NuGet utilizzando il comando seguente.

Listato 1.10: Installazione del plugin per SpecFlow in VS.

```
1 Install-Package SpecFlow.NUnit
```

Completata l'installazione del software non è necessario compiere alcuna operazione di configurazione. Il plugin estende il funzionamento di VS aggiungendo il completo supporto a Gherkin, evidenziando la sintassi nelle diverse lingue esistenti, introducendo l'auto-completamento all'interno dei file ".feature", permettendo la navigazione da passo ad implementazione, permettendo la generazione automatica degli stub rappresentanti i passi e aggiungendo le funzionalità in Gherkin nei menù rapidi per la creazione delle risorse.

facilmente navigabili da device mobile in quanto non prevede l'utilizzo di fogli di stile per definire la disposizione degli elementi nella pagina.

Per l'esecuzione delle funzionalità è possibile selezionare direttamente i file ".feature", eventualmente utilizzando il tag "@ignore" per saltare alcuni scenari, oppure sfruttare l'integrazione con NUnit per scegliere quale sotto-insieme di scenari eseguire, è possibile utilizzare i diversi ordinamenti di NUnit per eseguire gli scenari in funzione di durata di esecuzione, risultato del test, tag utilizzati o namespace.

1.3.2 Coypu

I parametri utilizzati per scegliere la libreria per l'automazione della navigazione via browser favoriscono strumenti preferibilmente scritti nel linguaggio utilizzato per implementare l'applicazione web, il cui sviluppo sia attivo, che possibilmente siano open-source e che abbiano una comunità attiva per poter avere un riscontro in caso di difficoltà.

L'individuazione di una libreria per .NET che soddisfacesse questi parametri non è stato semplice. Oltre a Selenium, già utilizzato per Java, sono stati valutati Watin, il cui sviluppo è fermo dal 2011, e Telerick Testing Framework, che per quanto sia un progetto attivo e ben documentato, sembra non essere utilizzato ed essere privo di una propria comunità di utenti.

Viste le diverse difficoltà nella scelta di una libreria è stata effettuata una scelta sperimentale utilizzando Coypu, un wrapper di Selenium Web Driver, scritto in C# che nella propria implementazione si ispira al DSL di Capybara.

Il progetto è stato rilasciato la prima volta 2011, 2 anni prima della versione beta di Selenium 2.0, ed è attualmente sviluppato, anche se il supporto è principalmente effettuato dal un singolo autore. Nonostante si tratti di una libreria open-source utilizzata da una ridotta comunità, presenta un'interessante prospettiva dell'automazione della navigazione web che merita di essere approfondita.

I principali obbiettivi di Coypu riguardano la semplificazione e razionalizzazione delle funzionalità della libreria di Selenium e la definizione di una libreria che, come Capybara, permetta di descrivere i propri test nella maniera più vicina possibile a come un utente descriverebbe le proprie azioni sul browser.

Configurazione di una sessione di testing

Come effettuato per Selenium in SBlog, anche per Coypu è stato scelto di utilizzare un nuovo ambiente di testing per ogni scenario, aprendo una nuova finestra prima prima dell'esecuzione e rilasciando le risorse al termine.

Listato 1.11: Creazione della sessione.

```
1 [BeforeScenario]
2 public void Before(){
3   var sessionConfiguration = new SessionConfiguration
                          {
4     Port = 1448,
5     Driver = typeof(SeleniumWebDriver),
```

```
6       Browser = Coypu.Drivers.Browser.PhantomJS,
7       Timeout = TimeSpan.FromSeconds(5)
8     };
9
10    _browser = new BrowserSession(sessionConfiguration);
                        _browser.MaximiseWindow();
          _objectContainer.RegisterInstanceAs(_browser);
11  }
```

Anche in SpecFlow è possibile definire delle callback, nell'esempio è definito un hook "BeforeScenario" da eseguire prima di ogni scenario, contenente le istruzioni per la definizione di una sessione. In particolare tramite la classe "SessionConfigurazion" è impostata la porta del server web, il driver ed il browser utilizzato ed il tempo massimo di esecuzione di ricerca per un selettore all'interno della pagina.

Listato 1.12: Chiusura del browser.

```
1  [AfterScenario]
2  public void AfterScenario(){
3    /*...*/
4    _browser.Dispose();
5  }
```

Per rilasciare la sessione di testing è invocato il metodo "dispose" al termine di ogni scenario.

PhantomJS

Per utilizzare PhantomJS per come browser per i propri test, è necessario mantenere l'eseguibile all'interno del "path" del proprio sistema o all'interno della cartella "bin" del progetto in VS. Per installare il browser localmente è possibile utilizzare la console di Nuget ed eseguire il seguente comando.

Listato 1.13: Installazione locale di PhantomJS.

```
1  install-package phantomjs.exe
```

1.3.3 Implementazione dei passi

Listato 1.14: Implementazione del passo "apro SBlog".

```
1  [Given(@"apro CSBlog")]
2  public void DatoAproCSBlog(){
3    /*..*/
4  }
```

In maniera paragonabile a Cucumber e Cucumber-JVM, l'implementazione di un passo coincide con la definizione di un metodo pubblico annotato con un attributo, come Given nell'esempio, per indicare tramite l'espressione regolare quale sia il passo implementato. I passi devono essere inseriti all'interno di classi annotate con l'attributo "Binding".

Rispetto ai framework per il BDD utilizzati finora, Specflow è più rigido per quanto riguarda l'implementazione dei passi, è infatti obbligatorio che il tipo dichiarato in Gherkin coincida con l'attributo utilizzato.

Listato 1.15: Implementazione valida per una pre-condizione ed un evento.

```
[Binding]
public class Constraints : BaseStep     {
  [Given(@"mi autentico come ""(.*)""")]
  [When(@"mi autentico come ""(.*)""")]
  public void DatoMiAutenticoCome(string email){
    /*..*/
  }
  /*..*/
```

Inoltre è possibile definire lo scope per ogni metodo o classe che implementi le funzionalità scritte in Gherkin tramite l'attributo "Scope". E' possibile impostare la visibilità dei passi in funzione del diverso posizionamento dell'attributo, annotando l'intera classe o il singolo metodo. Il codice del prossimo listato vincola la visibilità esprimendo le diverse condizioni disponibili.

Listato 1.16: Definizione dello scope per

```
[Scope(Tag = "xyz", Feature = "Funzionalità", Scenario = "Prova")]
```

Navigare all'interno del sito

```
[Given(@"apro CSBlog")]
public void DatoAproCSBlog(){
  browser.Visit("/");
  Assert.AreEqual("CSBlog", browser.Title);
}
```

Come in Selenium, Coypu fornisce all'utente diverse funzionalità per navigare all'interno del browser. Tramite l'istanza di "BrowserSession", che include le funzionalità per la gestione del browser come il ridimensionamento della finestra, il refresh della pagina e la navigazione attraverso la cronologia, è possibile navigare esplicitamente ad un URL passato come parametro, nell'esempio è mostrato come sia anche possibile utilizzare un percorso relativo.

Listato 1.17: Alcune delle funzionalità presenti nella classe "BrowserWindow".

```
public void AcceptModalDialog(Options options = null);
public void CancelModalDialog(Options options = null);
public bool HasDialog(string withText, Options options = null);
public bool HasNoDialog(string withText, Options options = null);
```

Approfondendo le funzionalità incluse nella classe "BrowserWindow", si intuisce la propensione di Coypu per la definizione di metodi che rappresentino le normali iterazioni di un'utente con il browser durante la navigazione web e siano il più intuitivi possibili.

```
[When(@"navigo verso ""(.*)""")]
public void QuandoNavigoVerso(string pageName){
```

```
3     Assert.True(browser.FindLink(pageName).Exists());
          browser.ClickLink(pageName);
4 }
```

Inoltre Coypu organizza la libreria in maniera che le funzionalità utilizzate di frequente siano facilmente accessibili. Ad esempio per utilizzare un collegamento presente nella pagina per navigare all'interno del sito è sufficiente utilizzare il metodo "ClickLink" esistente all'interno della classe "BrowserWindow" che rappresenta la sessione di navigazione ed è quindi già istanziata al momento dell'esecuzione dei test.

Inoltre il metodo "ClickLink" effettua il click dell'elemento in funzione del testo che appare e non di un selettore CSS o XPath, facilitando la definizione di test che mantengono la stessa prospettiva di un utente.

Il metodo "FindLink" permette la ricerca all'interno di un DOM di un collegamento in funzione del testo fornito come parametro. Il metodo restituisce un'istanza di "ElementScope" che rappresenta un tag presente nel DOM; in maniera simile a Capybara, sono presenti numerose proprietà per ottenere informazioni sull'elemento ed i metodi per eseguire le azioni.

Al fine di semplificare la definizione dei test, Coypu include in tutte le funzionalità che analizzano il DOM un meccanismo di attesa implicito. Assumere che ogni elemento possa essere il risultato di un'operazione asincrona, permette all'utente di definire test più semplici e mantenibili. Come definito nel metodo 1.11, che rappresenta la creazione e configurazione di una nuova sessione per la verifica di uno scenario, è possibile impostare sia il valore di attesa massimo che la frequenza di pooling desiderata.

Una carenza riscontrata nelle prime fasi di sviluppo è l'assenza di una documentazione nei sorgenti che chiarisca il funzionamento della libreria. Nonostante in Coypu sia facile intuire il funzionamento della libreria leggendo la segnatura dei metodi, la definizione di qualche esempio ne semplificherebbe l'utilizzo.

Definire asserzioni con NUnit NUnit è una libreria per la definizione di unit-test in C# nata dal porting di JUnit, attualmente alla versione 2.6 e il cui sviluppo è svolto in maniera separata dalla versione Java.

Nonostante i due progetti non siano più sviluppati dallo stesso team, utilizzando le asserzioni di NUnit è facile riconoscere l'origine comune. Ad esempio nel successivo passo, sono utilizzati i metodi "AssertThat" e "AreEqual", che rispettivamente verificano la presenza di un elemento con identificatore HTML "logo" ed effettuano un confronto sul titolo della pagina.

```
1 [Then(@"posso tornare alla pagina iniziale")]
2 public void AlloraPossoTornareAllaPaginaIniziale(){
3   var logoLink = browser.FindId("logo");
4   Assert.That(logoLink.Exists());
5   logoLink.Click();
6   Assert.AreEqual("CSBlog", browser.Title);
```

```
7 }
```

Uno degli obbiettivi degli autori di NUnit è di definire dei metodi che, attraverso la propria segnatura, descrivano il proprio funzionamento e siano facilmente leggibili.

Listato 1.18: Un possibile uso dell'asserzione "That".

```
1 Assert.That(myString, Is.EqualTo("Hello"));
```

Il metodo "That" nell'esempio è utilizzato per verificare un valore booleano, in generale però è utilizzato per la sua caratteristica di accettare parametri "IConstraint". L'interfaccia definisce un insieme di metodi per rappresentare una condizione e, oltre alle classi esistenti nella libreria come mostrato nell'esempio, permette la definizione di classi personalizzate. In Coypu è presente una classe "Shows" che implementa diverse condizioni e aumenta la leggibilità delle asserzioni.

Listato 1.19: Uso del tipo "Shows" di Coypu.

```
1 Assert.That(footer, Shows.Css("img"));
```

1.4 Introduzione del CSS

I metodi presenti nella libreria di Coypu non includono le funzionalità per leggere le proprietà di stile degli elementi presenti nel DOM di una pagina.

Listato 1.20: Seconda funzionalità per CSBlog

```
1 Feature: Introducendo il CSS
2 Per rendere l'esperienza di navigazione gradevole
3 Come Lettore
4 Vorrei che il sito esponesse una grafica omogenea
```

Nella funzionalità corrente dovrebbero essere verificati alcuni scenari relativi all'aspetto della pagina, in particolare gli effetti cromatici descritti nei precedenti capitoli e altre proprietà legate alla corretta visualizzazione degli elementi.

Nonostante gli sforzi dell'autore e degli sviluppatori, che frequentemente propongono le loro pull-request su GitHub, Coypu non può mantenere il livello di sviluppo di Selenium. Per fornire comunque pieno supporto ai propri utenti è possibile sfruttare la libreria nativa, utilizzata dalle stesse funzionalità di Coypu, come mostrato nel passo del listato 1.21.

Listato 1.21: Uso delle funzionalità native di Selenium.

```
1 [Given(@"l'intestazione ha un colore di sfondo")]
2 public void DatoLIntestazioneHaUnColoreDiSfondo(){
3   var selenium = ((OpenQA.Selenium.Remote.RemoteWebDriver)browser.
        Native);
4   var color = selenium.FindElementById("header").GetCssValue("
        background-color");
5   Assert.NotNull(color);
6   Assert.AreEqual("rgba(46, 47, 48, 1)", color);
7 }
```

Nel metodo è descritto l'utilizzo delle funzionalità di Selenium ed in particolare l'invocazione del metodo "GetCssValue" per ottenere la rappresentazione "rgba" del colore di sfondo dell'intestazione.

La versione per C# di Selenium è estremamente simile a quella per Java, sintassi permettendo; per questo motivo non è stata completata l'implementazione degli scenari per la funzionalità corrente.