BookSales UniBG: iterazione 1

Del Prete Giovanni, Ghilardi Nicola, Polver Marco

August 20, 2019

Contents

0.1 Obiettivi dell'iterazione

Di seguito vengono riportate le funzionalità implementate durante l'iterazione 1:

1. Registrazione nuovo utente
2. Login
3. Logout
4. Ricerca utenti
5. Ricerca annunci

Per l'implementazione di queste funzionalità si sono resi necessari due ulteriori passi:

1. *Realizzazione dei modelli dei dati*: in Django i modelli sono delle classi Python che rappresentano le tabelle che devono essere realizzate all'interno del database.
2. *Realizzazione di uno script Python per lo riempimento del database*: questo passo si è reso necessario per poter avere un numero consistente di dati su cui testare le diverse funzionalità dell'applicazione.

0.1.1 Note sul processo di implementazione

L'obiettivo iniziale era quello di realizzare questo progetto tramite un processo di Test Driven Development puro, tuttavia è stato necessario modificare questo processo permettendoci di testare alcune componenti del software solo dopo la loro implementazione.
Questo va contro i principi del TDD, per cui bisognerebbe prima ideare una serie di test e solo dopo realizzare un codice sufficiente per passarli, ma si è reso necessario per due motivi:

1. *Il nostro progetto è una applicazione web*: il testing delle applicazioni web risulta essere diverso rispetto al testing di applicazioni desktop e a volte i test risultano difficili anche solo da pensare prima di un'implementazione anche solo parziale della funzionalità da testare.
2. *Inesperienza con Django*: per tutti e tre questo progetto ha rappresentato l'occasione di provare Django per la prima volta. Questo framework è stato scelto in quanto particolarmente utilizzato al giorno d'oggi, ma nessuno di noi l'aveva utilizzato prima di questo progetto. Questo ha reso difficile, soprattutto all'inizio, l'applicazione del TDD, in quanto la realizzazione di un test presuppone una buona conoscenza degli strumenti utilizzati. I risultati ottenuti, tuttavia, sono ottimi e anche la nostra confidenza nell'utilizzo di Django è cresciuta notevolmente con il tempo.

0.1.2 Note sui test effettuati

Per la realizzazione dei test sono stati utilizzati due strumenti:

- La classe *TestCase* fornita da Django, utilizzata per l'esecuzione di test strutturali.
- *Selenium WebDriver*, una famosa API che permette di automatizzare l'utilizzo di un qualsiasi browser, molto utile per l'esecuzione di test funzionali.

Test strutturali

Come già è stato detto in precedenza, i test strutturali sono stati effettuati utilizzando la classe TestCse che viene fornita da Django.
L'esecuzione di test all'interno del framework Django risulta particolarmente comoda in quanto permette di utilizzare un vero e proprio "database di test", evitando quindi di sporcare il database normalmnte utilizzato dall'applicazione con dati utili solo in fase di test.
I test strutturali effettuati per le varie funzionalità dell'applicazione verificano in genere i seguenti punti:

- Utilizzo del template HTML corretto in ogni pagina.
- Restituzione dello status code corretto (quasi sempre 200 o 404) a fronte della richiesta di una pagina web con determinati dati.
- Comportamento corretto dell'applicazione a seguito della somministrazione di dati corretti ed errati.

Test funzionali

In uno script Python è stato utilizzato Selenium WebDriver per effettuare test funzionali.
Le differenze tra i test che si possono eseguire con la classe TestCase e con il WebDriver sono notevoli:

- TestCase facilita l'utilizzo di richieste GET e POST. Selenium WebDriver è invece un puro sostituto dell'uomo, pertanto va istruito su quali tasti cliccare, i testi da inserire nelle varie textbox, ecc.
- Selenium WebDriver non fa parte del framework Django, pertanto le azioni compiute da esso andranno a scrivere nuovi dati nel vero database dell'applicazione e non in un database di prova.

I test funzionali sono stati realizzati basandosi sui *casi d'uso*, pertanto non testano una singola funzionalità dell'applicazione ma la successione delle azioni compiute normalmente da un utente con un certo fine.

Copertura

Solo in poche occasioni la copertura dei test sui file Python presenti all'interno dell'applicazione risulta essere del 100%, questo a causa dei seguenti motivi:

- All'interno dei file di un'app Django sono spesso presenti porzioni di codice di default che potrebbero non essere coperte.
- Alcune funzioni utilizzano la libreria "random", la quale rende il comportamento di alcune porzioni di codice imprevedibile; non è quindi possibile avere una copertura del 100% in certe funzioni.