|  |
| --- |
| **Università degli Studi di Salerno Corso di Ingegneria del Software** |

**Buy To Eat  
System Test Plan e Specifica dei Casi di Test  
Versione 1.0**

****

Data: 19/12/2017

Indice

1. Funzionalità testate e non…………………………………………………………...3

2. Criteri Pass/Failed……...…………………………………………………………...3

3. Approccio…………………………………………………………………………...3

* 1. Testing di Unità ………………………………………………………..4
  2. Testing d‘ Integrazione…………………………………………………4
  3. Testing di Sistema……………………………………………………...4

1. Sospensione e Ricerca …………………………………………………………….5
   1. Criteri di Sospensione………………………………………………….5
   2. Criteri di Ripristino…………………………………………………….5
2. Materiale per il Testing……………………………………………………………5
3. Casi di Test………………………………………………………………………...5
4. Pianificazione del Testing…………………………………………………………7
5. **Funzionalità Testate e Non**

Le funzionalità che verranno testate tra quelle implementate sono:

• **Modifica dati personali;**

• **Modifica residenza;**

• **Elimina account.**

Quelle che non verranno testate, sempre tra quelle implementate, sono invece:

* + **Registrazione utente;**
  + **Registrazione azienda;**
  + **Login;**
  + **Logout;**
  + **Aggiungi prodotto al carrello;**
  + **Modifica prodotto al carrello;**
  + **Elimina prodotto dal carrello;**
  + **Visualizza dettagli ordine;**
  + **Modifica password.**

1. **Criteri Pass/Failed**

Di seguito vengono definiti i criteri per stabilire quando un input avrà superato un test e quando, invece, risulterà fallito.

Si noti che i dati di input del test verranno suddivisi in classi di equivalenza, ovvero verranno raggruppati in insiemi dalle caratteristiche comuni, per i quali sarà sufficiente testare un solo elemento rappresentativo.

Un input avrà superato un test se l’output risultante sarà quello atteso, cioè quello che è stato specificato nell’oracolo del test case.

Si noti che per oracolo si intende la persona (sviluppatore o membro del team di testing) che conosce quale dovrebbe essere l’output corretto.

In questo modo l’intera classe di appartenenza verificherà la correttezza del test.

Ovviamente, la stessa classe non supererà il test se l’output risultante sarà diverso da quello atteso.

1. **Approccio**

Le tecniche di testing adottate riguarderanno inizialmente il testing di unità dei singoli componenti, in modo da testare nello specifico la correttezza di ciascuna unità.

Seguirà il testing d’integrazione, che focalizzerà l’attenzione principalmente sul test delle interfacce delle suddette unità.

Infine verrà eseguito il testing di sistema, che vedrà come oggetto di testing l’intero sistema assemblato nei suoi componenti.

Quest’ ultimo servirà soprattutto a verificare che il sistema soddisfi le richieste del committente.

* 1. **Testing di Unità**

Durante questa fase, verranno ricercate le condizioni di fallimento isolando i componenti ed usando test driver e test stub, cioè implementazioni parziali di componenti che dipendono o da cui dipendono le componenti da testare.

Al fine di minimizzare il numero dei test cases, i possibili input verranno partizionati in classi di equivalenza e per ogni classe verrà selezionato un test case.

La strategia utilizzata per il testing, si baserà esclusivamente sulla tecnica BlackBox , che si focalizza sul comportamento Input/Output, ignorando la struttura interna della componente.

Gli stati erronei scovati in questa, che comporteranno un fallimento del sistema dovranno essere tempestivamente comunicati agli sviluppatori al fine di correggerli e ripristinare il testing al più presto.

Si noti che le unità verranno anche testate utilizzando il modulo JUnit 3 messo a

disposizione dal linguaggio Java. Tuttavia, per ragioni di tempo , questa tipologia di

testing non verrà documentata.

Per quanto riguarda il WhiteBox Testing, la sua realizzazione è stata effettuata

durante la fase implementativa in modo totalmente implicito da parte degli sviluppatori. Pertanto, la sua documentazione non viene riportata nel TestPlanning.

* 1. **Testing d’Integrazione**

Si procederà all’integrazione delle componenti di una funzionalità che verranno testate nel complesso attraverso una strategia di tipo Sandwich, derivante dalla combinazione delle tecniche Bottom-Up e Top-Down.

Si passerà, poi, alla funzionalità successiva, fino ad esaurire le funzionalità implementate.

Quest’approccio mira principalmente a ridurre le dipendenze tra funzionalità differenti e a facilitare la ricerca di errori nelle interfacce di comunicazione tra sottosistemi.

* 1. **Testing di Sistema**

Lo scopo di questa fase del testing è quello di dimostrare che il sistema soddisfi

effettivamente i requisiti richiesti e che sia pronto all’uso.

Come per il testing di unità, si cercherà di testare le funzionalità più importanti per

l’utente e quelle che hanno una maggiore probabilità di fallimento.

Si noti che, come per lo Unit Testing, si procederà attraverso la tecnica BlackBox.

1. **Sospensione e Ricerca**

**Criteri di sospensione**

Il testing sarà sospeso quando saranno state testate tutte le classi scelte per ogni

possibile input e il risultato sarà quello atteso e sperato.

Nel momento in cui il testing rivelerà un errore si dovrà passare alla fase di correzione che dovrà essere seguita da una ripetizione dell’intero testing per rilevare la presenza di errori introdotti dalle correzioni stesse.

**Criteri di ripristino**

La ripresa in seguito alla fase di correzione avverrà ripetendo l’intero testing per rilevare la presenza di errori introdotti dalle correzioni.

1. **Materiale per il Testing**

L’hardware necessario per l’attività di test è un PC, o possibilmente uno per ogni soggetto incaricato del testing, su cui sia installato il DBMS MySQL, un web server ApacheTomcat, un ambiente java Eclipse eun browser.

1. **Casi di Test**

La funzionalità da testare riguarda la ***modifica dei dati*** dell’utente registrato

**Classi di equivalenza**

Gli input del test sono codice fiscale, data nascita dell’utente registrato.

|  |  |  |
| --- | --- | --- |
| Input | Codice fiscale, data nascita | |
| Classe valida | C\_001 | Stringa alfanumerica di 16 caratteri e carattere di controllo valido. |
| Classe valida | C\_002 | Stringa alfanumerica di 10 caratteri. |
| Classi non valide | C\_003  C\_004 | Stringa maggiore o minore di 16 caratteri.  Stringa maggiore o minore di 10 caratteri |

**Identificazione test case**

|  |  |  |
| --- | --- | --- |
| Test case: **TC\_Modifica dati\_ 01** |  |  |
| **INPUT** | **CLASSE** | **MEMBRO** |
| Codice fiscale | **C\_001** | GLNGPP96E07A399E |
| Data nascita | **C\_002** | 07/05/1996 |

|  |  |  |
| --- | --- | --- |
| Test case: **TC\_Modifica dati\_ 02** |  | |
| **INPUT** | **CLASSE** | **MEMBRO** |
| Codice fiscale | **C\_001** | **CPRLSS96D25A783U** |
| Data nascita | **C\_004** | **123/4/80** |

|  |  |  |
| --- | --- | --- |
| Test case: **TC\_Modifica dati\_ 03** |  | |
| **INPUT** | **CLASSE** | **MEMBRO** |
| Codice fiscale | **C\_003** | **CPRLSS96D25A783Z** |
| Data nascita | **C\_004** | **1234/5645/2134** |

La funzionalità da testare riguarda la ***modifica residenza*** dell’utente registrato

**Classi di equivalenza**

L’input del test è solo il CAP dell’utente registrato.

|  |  |  |
| --- | --- | --- |
| Input | CAP | |
| Classi valide | C\_001 | Stringa numerica di 5 caratteri. |
| Classi non valide | C\_002 | Stringa numerica maggiore o minore di 5 caratteri. |

**Identificazione test case**

|  |  |  |
| --- | --- | --- |
| Test case: **TC\_Modifica residenza\_01** |  | |
| **INPUT** | **CLASSE** | **MEMBRO** |
| CAP | **C\_001** | **83050** |

|  |  |  |
| --- | --- | --- |
| Test case: **TC\_Modifica residenza\_03** |  | |
| **INPUT** | **CLASSE** | **MEMBRO** |
| CAP | **C\_002** | **11111111** |

|  |  |  |
| --- | --- | --- |
| Test case: **TC\_Modifica residenza\_03** |  | |
| **INPUT** | **CLASSE** | **MEMBRO** |
| CAP | **C\_002** | **1** |

La funzionalità da testare riguarda l’***elimina account*** dell’utente registrato

**Classi di equivalenza**

L’input del test è la password dell’utente registrato.

|  |  |  |
| --- | --- | --- |
| Input |  | |
| Classi valide | C\_001 | Stringa alfanumerica da 8 a 30 caratteri. |
| Classi non valide | C\_002 | Stringa alfanumerica minore di 8 o maggiore di 30. |

**Identificazione test case**

|  |  |  |
| --- | --- | --- |
| Test case: **TC\_Elimina account\_01** |  | |
| **INPUT** | **CLASSE** | **MEMBRO** |
| Password | **C\_001** | **12galante96** |

|  |  |  |
| --- | --- | --- |
| Test case: **TC\_Elimina account\_02** |  | |
| **INPUT** | **CLASSE** | **MEMBRO** |
| Password | **C\_002** | **Ale96** |

|  |  |  |
| --- | --- | --- |
| Test case: **TC\_Elimina account\_03** |  | |
| **INPUT** | **CLASSE** | **MEMBRO** |
| Password | **C\_002** | **GiuseppeGalante\_AlessioCaporale\_** |

1. **Pianificazione del Testing**

Il team che effettuerà il testing deve essere composto da persone che abbiano una conoscenza almeno discreta del sistema e delle tecniche di testing.

Tali tecniche dovranno essere applicate nel rispetto dei vincoli di tempo, budget e qualità stabiliti.

Per quanto riguarda il tempo, il team avrà a disposizione non più di tre giorni, mentre, essendo un progetto a scopi prettamente didattici, non ci saranno limitazioni a livello di budget.

Rispetto alla qualità, invece, si cercherà di scovare il più alto numero possibile di errori, sempre nei limiti di tempo stabiliti.

Normalmente i componenti del team di testing sono persone che non siano state coinvolte nella fase di sviluppo, in quanto è statisticamente provato che uno sviluppatore è molto restio a modificare una sua “creatura”.

Nel caso specifico, tuttavia, tale approccio non potrà essere rispettato poiché il numero dei componenti del team, sei per la precisione, è insufficiente per coprire interamente i task di sviluppo ed anche perché, dati i fini didattici del progetto, si preferisce rendere partecipe ad ogni fase dello sviluppo il più alto numero possibile di componenti del team.

Faranno parte del team di testing, quindi, gli sviluppatori che per primi completeranno l’implementazione, in particolare quelli che non hanno partecipato allo sviluppo delle interfacce.

Per quanto riguarda la documentazione, gli eventuali fault di una componente verranno notificati agli sviluppatori della componente stessa, per consentire la correzione del sistema.

Dopo la correzione e la revisione, il sistema verrà nuovamente testato per verificare se le modifiche apportate per risolvere gli errori individuati non ne abbiano introdotti nuovi.

Data l’importanza del testing è fondamentale una buona schedulazione di esso.

|  |  |
| --- | --- |
| **Scritto da:** | Caporale – Galante |