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

**GAP  
Test Plan**

****

Data: 24/11/2021

**Partecipanti:**

|  |  |
| --- | --- |
| Nome | Matricola |
| Giammarino Emanuele | 0512108088 |
| Adinolfi Giacinto | 0512107764 |

**Indice**

1. Dominio del problema ……………………………………………………………………4
2. Requisiti Funzionali…………………………………………………………………………5
3. Requisiti non funzionali…………………………………………………………………..6
4. Target Environement………………………………………………………………………7
5. Casi d’uso e diagrammi…………………………………………………………………10
6. Sequence Diagram………………………………………………………………………..32
7. Class Diagram.………………………………………………………………………………38
8. Activity Diagram……………………………………………………………………………39
9. Mockup…………………………………………………………………………………………40

**1. Introduzione**

Il Test Plan Document di GAP include e tiene traccia delle informazioni necessarie per definire l’approccio che deve essere usato nella fase di testing del progetto. Lo scopo di questo documento, quindi, è quello di gestire lo sviluppo, le attività di test di GAP e trovare eventuali errori all’interno del codice realizzato. I risultati dei test saranno poi utilizzati per mettere in risalto eventuali correzioni da apportare al codice al fine di risolvere i problemi per migliorare il sistema.

**2. Relazione con altri documenti**

Il test plan è legato ad altri documenti quali:

* RAD: dove verranno testati requisiti funzionali, requisiti non funzionali e casi d’uso.
* SDD: comprendente la definizione del sistema in sottosistemi e i controllo di accesso e sicurezza del sito web.
* ODD: verranno testate le classi implementate.

**3. Panoramica del sistema**

Il nostro sistema, come già detto nei documenti precedenti, è basato su architettura MVC. Le componenti principali sono il model che gestisce i dati e il control che gestisce la logica dell’applicazione e che quando accetta l’input lo converte in comandi per il modello e/o la vista. La view, invece, gestisce l’output delle informazioni e le interfacce.

**4. Funzionalità da testare**

Di seguito verranno riportate le funzionalità del sistema che saranno sottoposte a test:

**- Gestione Utente**

* Registrazione
* Login
* Logout
* Modifica password

**- Gestione Prodotto**

* Aggiungi un prodotto
* Rimuovi un prodotto
* Modifica un prodotto
* Ricerca un prodotto

**- Gestione Ordine**

* Inserisci carta e completa acquisto.

**- Gestione carrello**

* Aggiungi prodotto al carrello
* Rimuovi prodotto dal carrello
* Aumenta quantità di un prodotto nel carrello
* Diminuisci quantità di un prodotto nel carrello

In forse: visualizzazione ordini, visualizzazione prodotti appartenenti ad un ordine, visualizzazione pagina prodotto, controllo recensione di un prodotto.

**5. Criteri di successo/insuccesso**

La buona riuscita di un testing si ha quando trova una failure nel programma. Ciò, ovviamente, significa che la fase di testing ha successo nel caso in cui individuerà una o più failure. In tal caso, quest’ultima verrà analizzata e corretta.

La failure rappresenta uno stato in cui viene indirizzato il sistema, ma che non rispetta l’oracolo previsto.

**6. Approccio**

Il testing che svolgeremo su questo sistema si compone di tre fasi:

* Testing di unità, con lo scopo di testare le componenti del sistema singolarmente;
* Testing di integrazione, che servirà a testare le funzionalità dei vari componenti del sistema;
* Testing di sistema, che mira a verificare che l’intero sottosistema soddisfi le richieste del cliente.

**6.1 Test di unità**

Per realizzare il testing di ogni singola componente verrà utilizzata la tecnica ***Black-Box***.

Andremo ad esaminare le funzionalità dell’applicazione ed il comportamento input/output delle singole componenti senza tener conto della loro struttura interna.

Essendo quasi impossibile generare tutti i possibili input, verranno create ***classi di equivalenza*** che raggrupperanno i Test Cases per ridurre la ridondanza e rendere il test più efficiente. I risultati del testing verranno analizzati e usati successivamente per correggere eventuali errori del sistema.

**6.2 Test di integrazione**

(Il professore ci consigliò di testare bottom up le classi DAO, testare top down le classi JSP e poi mettere assieme e testare il tutto)

Il test di integrazione serve a testare l’intero sistema e si sviluppa dopo il test di unità, quando ogni parte testata viene integrata con il resto del sistema.

Per realizzare il test di integrazione utilizzeremo la strategia sandwitch: essa combina le strategie bottm-up e top-down, cercando di sfruttare al meglio entrambe.

**6.3 Test di sistema**

Il test di sistema ha lo scopo di testare tutte le funzionalità più importanti e usate maggiormente. Trattandosi di un sistema web-based verrà utilizzato il tool Selenium che si occupa di simulare l’interazione con il sistema dal punto di vista dell’utente.

**7. Sospensione e ripresa**

La fase di testing del sistema verrà sospesa quando non si otterranno i risultati attesi in accordo con i tempi di sviluppo previsti. La fase di testing poi potrà riprendere, in seguito, dopo aver effettuato le modifiche e/o eventuali correzioni che generano errori o fallimenti. I test cases, quindi, verranno sottoposti nuovamente al sistema assicurandosi così di aver risolto effettivamente il problema

**8. Materiali di prova (requisiti hardware/software)**

Gli strumenti utilizzati nella fase di testing saranno:

* Il software ***MySQL*** per la gestione del database.
* Il server ***Apache Tomcat***, per il caricamento in locale del sito web.
* ***JUnit, DBUnit e Mokito*** per il test di unità e il test di integrazione.
* ***Selenium*** per il test di sistema.

**9. Test Cases**

Capire cosa ci va qui

La sezione 9, il nucleo del piano di test, elenca i casi di test che sono usati durante i test. Ogni caso di test è descritto in dettaglio in un documento separato Test Case Specification. Ogni esecuzione di questi test sarà documentata in un documento Test Incident Report. Descriviamo questi documenti in maggior dettaglio più avanti in questa sezione.