UNIVERSITÀ DEGLI STUDI
DI SALERNO

Corso di Ingegneria del Software
Test Plan
(TP)

Quattroℂℌi.it

Data: 22/01/2018

Coordinatori del progetto:

Nome
De Lucia Andrea
Francese Rita

Partecipanti:

Nome	Matricola
Palmiero Alberto	0512103454
Piccolo Luigi	0512103964
Reccia Luca	0512103736
Volpe Nicola	0512103556

Sommario

1. INTRODUZIONE 4

2. DOCUMENTI CORRELATI 4

2.1. RELAZIONI CON IL DOCUMENTO DI ANALISI DEI REQUISITI (RAD) 4

2.2. RELAZIONI CON IL SYSTEM DESIGN DOCUMENT (SDD) 4

2.3. RELAZIONI CON L'OBJECT DESIGN DOCUMENT (ODD) 4

3. PANORAMICA DEL SISTEMA 5

4. FUNZIONALITÀ DA TESTARE E DA NON TESTARE 5

5. CRITERI PASS/FAILED 6

6. L'APPROCCIO 6

6.1. TESTING DI UNITÀ 6

6.2. TESTING DI INTEGRAZIONE 6

6.3. TESTING DI SISTEMA 6

7. SOSPENSIONE E RIPRESA 7

7.1. CRITERI DI SOSPENSIONE 7

7.2. CRITERI DI RIPRESA 7

8. MATERIALE PER IL TESTING 7

9. TEST CASE 8

10. PIANIFICAZIONE DEL TESTING 8

10.1. DETERMINAZIONE DEI RUOLI 8

10.2. DETERMINAZIONE DEI RISCHI 8

10.3. ORGANIZZAZIONE DELLE ATTIVITÀ DI TESTING 8

1. Introduzione

Lo scopo di questo documento è quello di pianificare il testing dell'e-commerce Quattrocchi con l'obiettivo di verificare che non ci siano differenze tra il comportamento atteso e quello effettivo. L'obiettivo è quello di collezionare e risolvere quanti più errori, diminuendo le probabilità che questi si presentino durante l'effettivo utilizzo del sistema da parte degli utenti finali.
È da pianificare il testing per le tre classi di gestione (utenti, articoli e ordini). Tuttavia andranno ad essere testate solo le funzionalità implementate specificate di seguito e nell'ODD.

2. Documenti correlati

Di seguito sono riportate le relazioni tra il test plan e la documentazione precedente.

2.1. Relazioni con il documento di analisi dei requisiti (RAD)

La relazione tra test plan e RAD riguarda in particolare i requisiti funzionali e non funzionali del sistema. Pertanto ogni test eseguito controllerà la coerenza con le funzionalità espresse nel RAD e opportunamente anche con i requisiti funzionali.

2.2. Relazioni con il System Design Document (SDD)

Nel System Design Document si è suddiviso il sistema in sottosistemi e l'architettura in tre livelli: Presentation Layer, Application Layer e Storage Layer. Il test dei vari componenti sarà basato tenendo conto di questa architettura.

2.3. Relazioni con l'Object Design Document (ODD)

Il test d'integrazione farà quanto più riferimento possibile alle class interfaces definite nell'ODD.

3. Panoramica del sistema

Come già definitivo nel SDD la struttura del sistema è divisa secondo l'architettura three tiers. I tre livelli sono il Presentation Layer, l'Application Layer e lo Storage Layer.
L'obiettivo di questa suddivisione e quello di diminuire l'accoppiamento ed aumentare la coesione tra le classi del sistema. Il sistema è diviso anche in sottosistemi più piccoli:

1. Gestione utenti
2. Gestione articoli
3. Gestione ordini

Per ognuno di questi sottoinsiemi andranno testate le operazioni più usate.

4. Funzionalità da testare e da non testare

Di seguito saranno elencate per ogni gestione quali sono le funzionalità che saranno testate

Gestione utenti

- accesso alla piattaforma (RF1)
- visualizzazione profilo (RF7)
- aggiunta nuovo metodo di pagamento (RF8)
- aggiunta nuovo indirizzo di spedizione (RF9)
- visualizzazione storico ordini (RF10)

Gestione ordini

- aggiunta di prodotti al carrello (RF4)
- visualizzazione carrello (RF5)
- checkout (RF11)
- visualizzazione ordini da spedire, in corso e terminati (RF12)
- modifica stato di un ordine (RF13)
- inserimento del corriere e numero tracking (RF14)

Gestione articoli

- ricerca nel catalogo (RF2)
- visualizzazione catalogo (RF3)

5. Criteri Pass/Failed

I dati di input del test saranno 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 all'interno del test case.

6. L'approccio

La fase di testing inizierà con il testing di unità dei singoli componenti, in tal modo da verificare la correttezza delle unità prima di andarle a comporre.
Il testing di integrazione sarà il successivo. Si controlla il comportamento dei componenti quando ne vengono aggiunti di nuovi.
Al termine di questi test seguirà il testing di sistema in cui verrà testato tutto il sistema implementato.

6.1. Testing di unità

Durante questa fase di testing verranno isolate le varie componenti. Usando driver e stub, ovvero implementazioni parziali di componenti dipendenti o da cui dipende il componente testato sarà possibile testare il comportamento su un ristretto insieme di input. Verrà usata la tecnica del Black Box la quale si limita al controllo dell'output avendo definito l'input ed ignora i comportamenti interni del componente. I possibili input saranno partizionati in classi di equivalenza e per ognuna identificata verrà scelto un test case. Gli errori saranno comunicati agli sviluppatori tramite un test incident report.

6.2. Testing di integrazione

Durante questa fase di testing si procede al testing delle componenti singolarmente testate e successivamente integrate secondo una strategia Bottom-up. Questo processo sarà iterato per tutte le funzionalità implementate. Questo metodo serve a ridurre le dipendenze tra funzionalità differenti e ad agevolare la ricerca di errori nelle interfacce di comunicazione dei sottosistemi.

6.3. Testing di sistema

Questa fase servirà a dimostrare che il sistema risponde a tutti i requisiti richiesti dal committente. Si ha come obiettivo di testare tutte le funzionalità implementate nel primo rilascio. Come per il testing di unità verrà utilizzato un approccio Black-Box.

7. Sospensione e ripresa

7.1. Criteri di sospensione

Il testing di un sottosistema o di una funzionalità dovrà essere interrotto in caso di sottosistemi esterni dipendenti non ancora disponibili, di errori riscontrati che dovranno prima esse re risolti e nel caso in cui la deadline della consegna del progetto sia imminente.
Aree del sistema non in dipendenza dal sottosistema che presenta un errore riscontrato nella fase di testing potranno continuare ad essere testate senza aspettare la fix da parte dello sviluppatore.

7.2. Criteri di ripresa

Il testing di un sottosistema o di una funzionalità potrà essere ripreso quando i sottosistemi dipendenti saranno disponibili, sarà stato comunicato dagli sviluppatori di aver risolto gli errori precedentemente riscontrati o venga spostata la deadline della consegna in una data che consenta di effettuare un test più accurato.
È necessario notare che in caso di retesting di un sottosistema o funzionalità che precedentemente ha riscontrato errori non è possibile riprendere il test dove interrotto ma dovrà essere rieseguito per verificare l'eventuale presenza di nuovi errori.

8. Materiale per il testing

L'hardware necessario per l'attività di test è un pc con browser che supporti javascript ed una connessione ad Internet.

9. Test case

I test case sono specificati nel documento “Test Case Specification”.

10. Pianificazione del testing

Il team di testing sarà costituito da persone che hanno una completa conoscenza del sistema, dei documenti e delle tecniche di testing. Date le dimensioni ridotte del team i tester avranno anche implementato parte del sistema. Tuttavia si cercherà di distribuire i compiti in modo tale che ogni membro del gruppo possa testare un componente non implementato da lui stesso.
Dopo aver eseguito un test ed aver riscontrato un errore il tester deve compilare il Test Incident Report. Al termine del lavoro dello sviluppatore il test dovrà essere rieseguito, non solo per verificare l’effettiva rimozione dell’errore, ma per controllare che la correzione non ne abbia creati di nuovi.

10.1. Determinazione dei ruoli

Il team dedicato all’attività di testing di sistema sarà composto da: Palmiero Alberto, Piccolo Luigi, Reccia Luca e Volpe Nicola. A ognuno di essi è in più richiesto il testing di unità delle componenti da loro sviluppate.

10.2. Determinazione dei rischi

I rischi di un completo fallimento verranno minimizzati effettuando una pianificazione verticale delle attività di testing funzionale. SI procederà prima col test di unità per le componenti del DB, poi per le classi manager, per infine passare al test di integrazione e sistema. Tale approccio ridurrà notevolmente l’utilizzo di driver e stub, evitando l’introduzione di nuovi errori dovuti all’implementazione di tali componenti.

10.3. Organizzazione delle attività di testing

Il testing sarà organizzato secondo uno schema che effettuerà la divisione di tipo verticale. Questo per evitare che nonostante ritardi dovuti e numerosi errori presenti in un sottosistema, sia comunque possibile rilasciare il sistema con meno componenti, ma testate accuratamente.