Laurea Magistrale in informatica-Università di Salerno
Corso di *Gestione dei Progetti Software*- Prof.ssa F.Ferrucci

Test Plan

MyBomber

Riferimento	
Versione	1.0
Data	11/01/2022
Destinatario	Prof.ssa F. Ferrucci
Presentato da	Gaetano Mauro
Approvato da	

Revision History

Data	Versione	Descrizione	Autori
04/01/2022	0.1	Prima stesura	Gaetano Mauro
11/01/2022	1.0	Revisione	Gaetano Mauro

Sommario

Revision History 2
1. Introduzione 4
1.1 Definizioni, acronimi, e abbreviazioni 4
1.1.1 Definizioni 4
1.1.2 Acronimi e Abbreviazioni 4
1.1.3 Riferimenti 4
2. Documenti correlate 5
2.1 Relazione con il RAD 5
2.2 Relazione con il SDD 5
2.3 Relazione con l'ODD 5
2.4 Relazione con il SOW 5
3. Panoramica del Sistema 5
4. Funzionalità da Testare 6
5. Funzionalita da non Testare 7
6. Approcci Utilizzati 7
6.1 Testing di unità 7
6.1.1 Approccio Scelto 7
6.2 Testing di Integrazione 7
6.2.1 Approccio Scelto 7
6.3 Testing di Sistema 7
6.3.1 Approccio Scelto 8
7 Criteri di pass/fail 8
8. Criteri di sospensione e ripresa 8
8.1 Criteri di sospensione 8
8.2 Criteri di ripresa 8
9. Test Deliverables 8
10. Materiale per effettuare Testing 9
11. Attività di training per lo staff 9
12. Responsabilità 9
13. Glossario 9

1. Introduzione

1.1 Definizioni, acronimi, e abbreviazioni

1.1.1 Definizioni

- Branch Coverage: tecnica adoperata durante la fase di testing, che prevede l'esecuzione di tutti i rami del programma almeno una volta durante la fase di testing.
- Failure: mancata o scorretta azione di un determinato servizio atteso.
- Fault: causa che ha generato una failure.
- Model View Control: è un metodo architetturale che prevede la divisione dell'applicazione di tre parti. Tale divisione viene effettuata per separate la rappresentazione delle informazioni interne del sistema dal meccanismo in cui le informazioni sono presentate all'utente

1.1.2 Acronimi e Abbreviazioni

- RAD: Requirement Analysis Document.
- SDD: System Design Document.
- ODD: Object Design Document.
- SOW: Statement of work.
- TP: Test plan.
- MVC: Model View Controller.
- DB: Database.
- API: Application Programming Interface.
- GUI: Graphical User Interface.

1.1.3 Riferimenti

- Object-Oriented Software Engineering (Using UML, Patterns, and Java) Third Edition. Autori: Bernd Bruegge & Allen H. Dutoit
- PMBOK Guide and Software Extention to the PMBOK Guide, Fifth Ed, Project Management Institute, 2013;

- Documentazione del progetto:
  - 2021_RAD_C11_MyBomber_Mauro versione 2.0
  - 2021_SDD_C11_MyBomber_Mauro versione 2.0
  - 2021_ODD_C11_MyBomber_Mauro versione 2.0
  - 2021_SOW_C11_MyBomber_Mauro versione 1.0

2. Documenti correlate

Questo document si basa su tutti gli altri precedentemente realizzati, quindi verrà mano mano aggiornato ogni volta che viene apportata una modifica anche sugli altri documenti.

2.1 Relazione con il RAD

I test case sono basati sulle funzionalità del sistema individuate nel RAD. Inoltre, gli scenari, use case, state chart e tutti gli altri diagrammi sono d'aiuto per comprendere al meglio ogni singolo requisito del sistema.

2.2 Relazione con il SDD

In questo documento sono riportate le informazioni di progettazione della base dati, dell'architettura usata e delle configurazioni scelte per eseguire il sistema.

2.3 Relazione con l'ODD

Contiene i package e le classi del sistema.

2.4 Relazione con il SOW

Nello statement of work, nella sezione 'criteri di accettazione', è stato stabilio che la branch coverage deve essere pari almeno al 75%. Quindi i test case saranno sviluppati in modo da rispettare questo criterio.

3. Panoramica del Sistema

Come definito nel System Design Document, il sistema avrà una struttura a tre livelli. La componente fondamentale di questo approccio è il controller che si occuperà della logica esecutiva di ogni

sottosistema, nel model verranno indicate le entità persistente del DB, infine nella view verranno mostrate le interfacce utente.

4. Funzionalità da Testare

Nell'attività di testing si andranno a testare quasi tutte le classi del sistema MyBomber, proprio per ridurrre al minimo il margine di errore.

Per quanto riguarda i model, si andranno a testare soltanto i DAO (dato che i Bean contengono getter e setter e considerando che sono stati autogenerati, riteniamo non importante la realizzazione delle classi di test su questi ultimi e saranno testati SOLO nel caso in cui non si riesca a raggiungere il 75% di branch coverage).

I DAO che si andranno a testare sono i seguenti:

- EventoDAO
- GestoreDAO
- GiocatoreDAO
- PartecipazioneDAO
- RecensioneDAO
- StrutturaDAO

I controller (ovvero le Servlet) saranno testati tutti:

- AreaUtenteServlet
- CreaEventoServlet
- CronologiaEventiServlet
- EventiRecentiServlet
- LoginServlet
- LogoutServlet
- PartecipaEventiServlet
- RecensioneServlet
- RegistrazioneServlet
- RichiesteEventiServlet
- StrutturaServlet

Sarà, inoltre, testato anche il DriverManagerConnectionPool.

5. Funzionalita da non Testare

Come anticipato nel capitolo precedente, i Bean saranno testati solo nel caso in cui non si raggiunga una branch coverage del 75%. È stato ritenuto superfluo testare i Bean in quanto contengono solo i metodi getter e setter che sono stati autogenerati tramite l'IDE Eclipse.

6. Approcci Utilizzati

6.1 Testing di unità

Lo scopo del testing di unità è quello di testare la funzionalità isolandola da tutte le altre componenti con cui coopera.

6.1.1 Approccio Scelto

L'approccio scelto è di tipo white-box, quindi il sistema sarà testato da team members che conoscono il funzionamento interno. Sarà utilizzato JUnit per questa fase.

6.2 Testing di Integrazione

Lo scopo del testing di integrazione è quello di testare le componenti integrate tra loro, che precedentemente erano state testate in unità.

6.2.1 Approccio Scelto

Per effettuare il testing di integrazione si è scelto di adoperare un approccio bottom-up. Il vantaggio fondamentale di questa tipologia di testing è quello della riusabilità del codice. Questo tipo di approccio prevede però la costruzione driver per simulare l'ambiente chiamante. È stato scelto quindi questo tipo di approccio perché sembra quello più intuitivo e semplice.

6.3 Testing di Sistema

Lo scopo del testing di sistema è quello di verificare che i requisiti richiesti dal cliente siano stati effettivamente rispettati e che il cliente risulti soddisfatto del sistema stesso. In questo tipo di testing si

vanno a verificare le funzionalità utilizzate più spesso da parte del cliente e quelle che risultano più "critiche".

6.3.1 Approccio Scelto

Trattandosi di una piattaforma web il tool scelto per il testing di sistema è "Selenium" che si occuperà di simulare le interazioni con il sistema stesso come se le stesse svolgendo l'utente.

7 Criteri di pass/fail

Dopo aver individuato tutti i dati di input del sistema, quest'ultimi verranno raggruppati insieme in base alle caratteristiche in comune. Questa tecnica ci servirà per poter diminuire il numero di test da dover effettuare. Diremo che la fase di test ha successo se viene individuata effettivamente una failure all'interno del sistema, cioè l'output atteso per quel determinato input non è lo stesso previsto dall'oracolo. Successivamente la failure sarà analizzata e si passerà eventualmente alla sua correzione e verranno eseguiti nuovamente tutti i test necessari per verificare l'impatto che la modifica ha avuto sull'intero sistema. Diremo invece che il testing fallirà se l'output mostrato dal sistema coincide con quello previsto dall'oracolo.

8. Criteri di sospensione e ripresa

8.1 Criteri di sospensione

La fase di testing verrà sospesa nel momento in cui saranno raggiunti i risultati previsti in accordo con quello che è il budget a disposizione.

8.2 Criteri di ripresa

Tutte le attività di testing riprenderanno nel momento in cui verranno effettuate modifiche all'interno del sistema.

9. Test Deliverables

I documenti che saranno prodotti durante questa fase sono i seguenti:

- Test Plan
- Test Case Specification

- Test Execution Report
- Test Incident Report
- Test Summary Report

10. Materiale per effettuare Testing

Le risorse che vengono utilizzate dalle attività di testing comprendono i documenti di progetto Requirement Analysis Document, System Design Document ed Object Design Document, a partire dai quali vengono individuate le componenti da testare, rispettivamente, nel testing di sistema, nel testing di integrazione e nel testing d'unità. Per l'esecuzione di queste attività, invece, vengono utilizzati gli strumenti Selenium e JUnit su Eclipse, insieme ad altri tool qualora sia necessario.

11. Attività di training per lo staff

Sarà effettuata una sessione di training da parte del PM per l'utilizzo di JUnit, dei mock di Mockito per il testing di unità e Selenium.

12. Responsabilità

Ogni team member sarà responsabile della fase di testing. Non è stata prevista una suddivisione tra sviluppatori e tester in quanto ogni team member deve svolgere tutte le mansioni.

13. Glossario

Testing: procedimento utilizzato per individuare le carenze di correttezza, completezza e affidabilità dei componenti software nel corso dello sviluppo.
Tool: strumento software utilizzato per ottenere un determinato risultato.