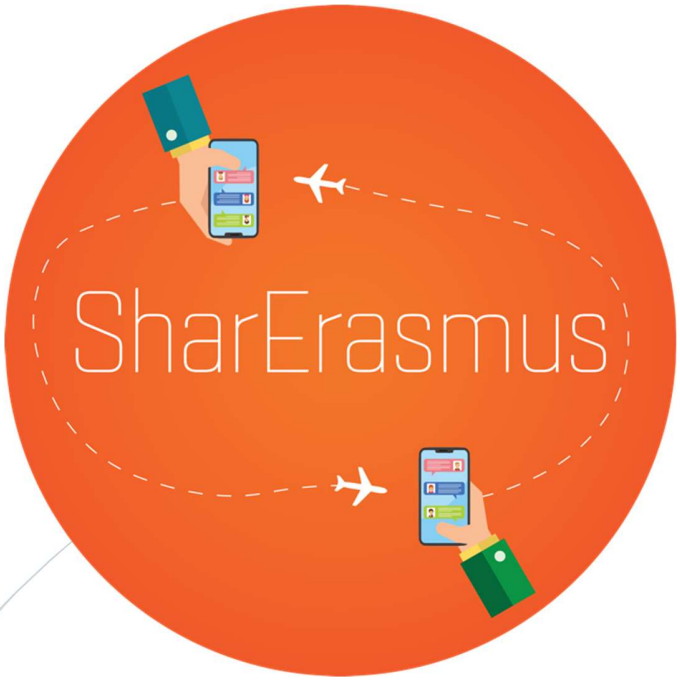
28/11/2018

Test Plan ShareErasmus

Share your Erasmus experience



Riferimento	
Versione	1.0
Data	28/11/2018
Destinatario	Prof.ssa F. Ferrucci
Proposto da	Federico Vitale, Francesco Vicidomini
Approvato da	

A.A. 2018/2019

Revision History

Data	Versione	Descrizione	Autori
28/11/18	1.0	Prima stesusra	Federico Vitale, Francesco Vicidomini

Team composition

Ruolo	Nome	Posizione	Contatti
Top Manager	Filomena Ferrucci	Rappresentante del cliente	f.ferrucci@unisa.it
Project Manager	Federico Vitale	Project Manager	f.vicidomini14@studenti.unisa.it
Project Manager	Francesco Vicidomini	Project Manager	f.vitale40@studenti.unisa.it
Team Member	Alfonso Ruggiero		a.ruggiero114@studenti.unisa.it
Team Member	Davide Bottiglieri		d.bottiglieri4@studenti.unisa.it
Team Member	Francesco Breve		f.breve@studenti.unisa.it
Team Member	Giuseppe Cavaliere		g.cavaliere10@studenti.unisa.it
Team Member	Paolo Cantarella		p.cantarella1@studenti.unisa.it
Team Member	Rosaria Iorio		r.iorio11@studenti.unisa.it
Team Member	Silvio Corso		s.corso1@studenti.unisa.it
Team Member	Vincenzo Sabato		v.sabato1@studenti.unisa.it

Indice

1. Introduzione

In questo documento verranno definiti gli approcci e le attività di testing riguardanti la piattaforma web SharErasmus. Verranno identificati gli elementi da testare e da non testare, gli approcci e gli strumenti usati per l'attività di testing. Lo scopo del testing è quello di rilevare fault e failure presenti nel sistema, infatti, diremo che il testing avrà avuto successo se è riuscito ad individuare dei fault o una failure. Questa attività ci consentirà di scoprire degli errori prima della messa in esercizio del sistema, così facendo, potremo porre rimedio a questi errori e fare in modo che non si rivelino durante la messa in esercizio del sistema.

Le gestioni che andremo ad analizzare, sono le gestioni che contengo requisiti con priorità media o alta, quindi analizzeremo:

- gestione profilo utente;
- gestione forum;
- gestione coordinatori.

1.1 Definizioni, Acronimi e Abbreviazioni

1.1.1 Definizioni

- **Branch Coverage:** metodo che richiede che tutti i rami del programma o gli stati condizionali vengano testati almeno una volta durante un processo di test;
- **Failure**: mancata prestazione di un servizio atteso;
- **Fault**: causa di un failure, insieme di informazioni che quando processate generano un fallimento;
- **Model View Control**: modello architetturale comunemente usato per lo sviluppo di interfacce utente che dividono un'applicazione in tre parti interconnesse. Questo viene fatto per separare le rappresentazioni interne di informazioni dai modi in cui le informazioni sono presentate e accettate dall'utente;
- **REpresentional State Transfer**: definisce un insieme di vincoli da utilizzare per la creazione di servizi Web;

1.1.2 Acronimi e Abbreviazioni

- **SE_TP_Vers.1.0:** Utilizzata per indicare il Test Plan;
- **RAD**: Abbreviazione utilizzata per indicare il Requirement Analysis Document;
- **SDD**: Abbreviazione utilizzata per indicare il System Design Document;
- **ODD**: Abbreviazione utilizzata per indicare il Object Design Document;
- **SOW**: Abbreviazione utilizzata per indicare lo Statement Of Work;
- **BC**: Abbreviazione utilizzata per indicare il Business Case;

- **TP**: Abbreviazione utilizzata per indicare il Test Plan;
- **STC**: Abbreviazione utilizzata per indicare il System Test Case;
- **MVC**: Abbreviazione utilizzata per indicare l'architettura Model View Controller;
- **REST**: Abbreviazione utilizzata per indicare il REpresentional State Transfer;
- **DB:** Abbreviazione utilizzata per indicare il Database;
- **API:** Abbreviazione utilizzata per indicare le Application Programming Interface.
- **SE_CP_Vers.1.0:** Abbreviazione utilizzata per indicare il documento riguardante il Category Partition

1.2 Riferimenti

- Kathy Schwalbe, "Information Technology Project Management", International Edition 7E, Cengage Learning, 2014;
- Bernd Bruegge, Allen H. Dutoit, "Object-Oriented Software Engineering Using UML, Patterns and Java", Third Ed., Pearson, 2010;
- Sommerville, "Software Engineering", Addison Wesley;
- PMBOK ® Guide and Software Extention to the PMBOK® Guide, Fifth Ed., Project Management Institute, 2013
- Documentazione di Progetto:
  - SE_RAD_Vers.1.0;
  - SE_SDD_Vers.1.0;
  - SE_STC_Vers.1.0.
  - SE_SOW_Vers.1

2. Documenti Correlati

Il presente documento è in stretta relazione con i documenti prodotti fino al rilascio della versione 1.0 del Test Plan, esso è in forte relazione anche con documenti che verranno sviluppati e rilasciati in futuro quindi sarà soggetto a modifiche ed aggiornamenti.

I test case sono basati sulle funzionalità individuate nel documento di raccolta ed analisi dei requisiti.

2.1 Relazione con il documento di raccolta ed analisi dei requisiti (RAD)

La relazione fra TP e RAD riguarda i requisiti funzionali e non funzionali del sistema, in quanto esso contiene la descrizione dettagliata delle funzionalità con scenari, use case, diagrammi e mockup e inoltre vi è indicata anche la priorità dei requisiti.

2.2 Relazione con il System Design Document (SDD)

Nell' SDD è presente la architettura del sistema (MVC), la struttura dei dati e i servizi dei sottosistemi.

2.3 Relazione con l'Object Design Document (ODD)

Nell'ODD (ancora non sviluppato al momento del rilascio dell'SE_TP_Vers.1.0) sono contenuti i package e le classi del sistema.

2.4 Relazione con lo Statement Of Work (SOW)

Nello Statement Of Work uno dei criteri di premialità, è quello di ottenere una branch coverage del 75%, i test case verranno strutturati in modo tale da poter soddisfare tale criterio.

3. Panoramica del Sistema

Come definito nel System Design Document la struttura del nostro sistema segue l'architettura MVC (Model View Controller) che espone i servizi utilizzando un approccio REST.

La componente fondamentale di questa architettura è il controller. Per ogni sottosistema ci sarà un controller che si occuperà della logica di business della specifica gestione.

Nel model verranno mappate le entità persistenti sul DB come oggetti.

La view si occuperà di mostrare le interfacce utente.

L'approccio REST verrà implementato utilizzando l'oggetto *router*. Esisterà un router per ogni sottosistema, il quale definirà una specifica URL sfruttando il nome del sottosistema e la firma di un metodo. Presso tale URL verranno esposti i risultati di una interrogazione al DB.

I sottosistemi sono:

- gestione profilo utente
- gestione forum
- gestione coordinatori
- gestione chat

4. Possibili Rischi

ID	Rischio	Risvolto	Probabilità	Impatto
R1	Pianificazione delle attività di testing non adeguata	Negativo	Bassa	Alto
R2	Il team non segue l'approccio definito	Negativo	Bassa	Alto
R3	Il team trova difficoltà nello specificare i casi di test	Negativo	Media	Alto
R4	Il team trova difficoltà nello usare i tool definiti per svolgere la attività di testing	Variabile	Media	Medio
R5	Le attività di testing proseguono oltre i tempi previsti	Variabile	Media	Medio/Alto
R6	I casi di test definiti non riescono a portare una branch coverage del 75%	Negativo	Media	Medio

5. Funzionalità da Testare

Come indicato in precedenza andranno testati solo i requisiti che presentano una priorità media o alta, di seguito l'elenco dei requisiti da testare per ogni gestione:

- Gestione profilo utente
  - Login
  - Logout
  - Recupero password
  - Cancellazione account
  - Registrazione nuovo utente
  - Modifica dati di accesso
  - Inserimento dati personali
  - Modifica dati personali
  - Visualizza dati personali
  - Timeline studenti
- Gestione forum
  - Pubblicazione post
  - Visualizzazione post
  - Rispondere post
  - Visualizzazione post raggruppati
  - Sistema di rating
  - Profili certificati
  - Bacheca avvisi
  - Sistema di notifiche
- Gestione coordinatori
  - Visualizzazione lista studenti
  - Aggiungere studente
  - Mappa globale studente
  - Tabella ECTS
  - Voto tradotto
  - Maching esami
  - Upload documenti
  - Timeline
- Gestione chat
  - Invio/ricezione messaggi
  - Blocca utente

6. Funzionalità da Non Testare

Di seguito sono riportate le funzionalità non candidate al testing perché di priorità bassa:

- Gestione chat
  - Invio file
  - Chat di gruppo

Un'altra delle funzionalità non candidate al testing è la funzionalità appartenente alla gestione forum riguardante il sistema di notifiche, questa non andrà testata dato che per la gestione delle notifiche useremo le API di Firebase.

7. Approccio

7.1 Testing di unità

In questa fase andremo a testare ogni singola funzione degli oggetti creati. Questa rappresenterà la nostra unità.

Verrà utilizzato un approccio black-box, ovvero non sarà basato sulla conoscenza dell'architettura e del funzionamento interno di un componente ma sulle sue funzionalità esternamente esposte.

Per tale fase utilizzeremo i tool Mocha, framework di test per NodeJS, al quale abbineremo Chai, una libreria per redigere asserzioni.

7.2 Testing di integrazione

In questa fase le singole unità vengono combinate e testate come gruppo.

Per poter effettuare l'integration test è stata scelta la strategia bottom-up, in quanto consente di poter iniziare l'attività di testing non appena il primo modulo è stato specificato. Questo approccio, richiede la costruzione di driver per simulare l'ambiente chiamante.

In generale però, può indurre il rischio che i moduli possano essere codificati senza avere una chiara idea di come dovranno essere connessi ad altre parti del sistema.

La riusabilità del codice è uno dei principali benefici dell'approccio bottom-up. Nonostante questa strategia di testing di integrazione abbia alcune limitazioni, risulta essere la più semplice e naturale forma con cui eseguire questo tipo di testing.

7.3 Testing di sistema

Prima della messa in esercizio del sistema verrà effettuata una fase di testing di sistema per dimostrare che i requisiti commissionati dal cliente siano soddisfatti. In questo tipo di testing andremo a testare le funzionalità usate più frequentemente dal cliente e quelle che presentano maggiore possibilità di fallimento. Trattandosi di una applicazione web verrà utilizzato il tool Selenium, che si occuperà di simulare l'interazione con il sistema dal punto di vista dell'utente.

8. Criteri di Pass/Fail

Una volta individuati i vari dati di input del test, questi verranno raggruppati in base a caratteristiche comuni in insiemi. In questo modo ci sarà possibile diminuire ed ottimizzare il numero di test.

La fase di test avrà successo se individuerà una failure, cioè se l'output osservato sarà diverso da quello atteso. Ogni qual volta verrà individuata una failure, questa verrà analizzata e, se legata ad un fault, si procederà alla sua correzione. Una volta completata la correzione, si procederà, in modo iterativo, ad una nuova fase di test per verificare che la modifica non ha impattato su altri componenti del sistema.

Al contrario, il testing fallirà se l'output osservato sarà uguale all'oracolo.

9. Criteri di Sospensione e Ripresa

9.1 Criteri di sospensione

La fase di testing verrà interrotta quando verranno raggiunti i risultati attesi in accordo con in tempi e i costi allocati alle attività di testing.

9.2 Criteri di ripresa

Le attività di testing riprenderanno in seguito a delle modifiche che potrebbero introdurre nuovi errori all'interno del sistema.

10. Test Deliverables

I documenti rilasciati durante e al termine della fase di test sono:

- Test Plan;
- System Test Case;
- Category Partition;
- Test Case Specification;
- Unit Test Report;
- Test Execution Report;
- Test Incident Report;
- Test Summary Report;

11. Materiale per il Testing

Per svolgere le attività di testing è necessario un computer e una copia del DB in locale, in modo tale da non appesantire il DB remoto con dati di prova.

12. Necessità di Training per lo Staff

Il team non ha mai effettuato una attività di test quindi non conoscono il meccanismo delle espressioni regolari e tantomeno i tool da utilizzare per le varie fasi di test. Per far fronte a questa esigenza i PM svolgeranno delle attività tutorie mirate a colmare la mancanza di skills e

successivamente forniranno esempi utili per rendere più chiari i concetti spiegati durante questi tutorati.

13. Resposabilità

Ogni team member sarà resposabilie del testing della gestione alla quale è stato assegnato. Il team non verrà diviso in "tester" e "developer" per evitare overhead di comunicazione.

14. Glossario

- **Chat**: 'conversazione' fra più interlocutori costituita da uno scambio di messaggi scritti che appaiono in tempo reale sul monitor di ciascun partecipante;
- **Coordinatori**: docente o rappresentante di una azienda che si occupa di coordinare uno studente Erasmus;
- **Developer:** un individuo che costruisce e crea software e applicazioni. Lei o lui scrive, esegue il debug ed esegue il codice sorgente di un'applicazione software;
- **ECTS:** sistema di credito progettato per facilitare lo spostamento degli studenti tra diversi paesi;
- **Errore**: una misura della differenza stimata tra il valore osservato o calcolato di una quantità e il suo valore reale;
- **Forum**: servizio che permette di inviare e leggere messaggi su un argomento specifico, che restano a disposizione per i commenti altrui;
- **Post:** messaggio inviato all'interno del forum**;**
- **Profilo Utente**: pagina che mostra tutte le informazioni relative ad un utente;
- **Rating:** valutazione che può essere associata ad un utente;
- **Rischio:** risultato potenziale, imprevedibile e incontrollabile di un'azione (o inazione);
- **Tester:** una persona, una macchina o dispositivo utilizzato per verificare se un sistema o componente funziona correttamente;
- **Testing**: processo o metodo per trovare gli errori in un'applicazione o un programma software in modo che l'applicazione funzioni in base ai requisiti dell'utente finale;
- **Timeline**: Rappresentazione grafica della sequenza cronologica degli eventi più significativi (che include anche i documenti più importanti) che si verificano prima di partire, durante e

dopo l'Erasmus**;**

- **Tool:** Strumento software utilizzato per ottenere un dato risultato.