

RAD Requirement Analysis Document

YouLearn Platform Project

UNIVERSITÁ DEGLI STUDI DI SALERNO

Ingegneria del Software

TEST PLAN



**ANNO ACCADEMICO 2018/2019**a

**TOP MANAGER:**

|  |
| --- |
| **Nome** |
| **Prof. Andrea De Lucia** |

**PARTECIPANTI:**

|  |  |
| --- | --- |
| **Nome** | **Matricola** |
| Mario Sessa | 0512104650 |
| Luigi Crisci | 0512104740 |
| Pasquale Ambrosio | 0512104704 |

**HISTORY:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Versione** | **Cambiamenti** | **Autori** |

1. Introduzione

Lo scopo del sistema è quello di pianificare l’attività di testing all’interno della piattaforma YouLearn con lo scopo di verificare se esistono differenze tra il comportamento atteso) con il comportamento osservato. In questa attività andremo ad individuare i possibili errori nel codice sorgente causanti incident all’interno del sistema prima che l’utente finale esegua delle interazioni con il sistema. Le attività di testing sono organizzate in modo da poter verificare il funzionamento dei seguenti sottosistemi:

1. Gestione utente
2. Gestione corso
3. Gestione lezioni

Si noti che non vi saranno attività di testing per i sottosistemi non descritti all’interno dell’Object design document e sviluppati in fase di implementazione. Quindi i sottosistemi seguenti saranno esclusi dalle verifiche delle funzionalità di sistema:

1. Gestione pagamenti
2. Gestione mail

I documenti precedenti al testing sono di grande importanza per la fase di testing, questo perché ogni documento, sviluppato secondo un differente livello di progettazione, saranno di grande importanza per la correttezza del testing.

2. Documenti correlati

La documentazione precedente è strettamente correlata con la pianificazione dei test, questo perché già nei documenti precedenti abbiamo definito come alcuni servizi devono funzionare. Questa caratteristica prevede di rilevare informazioni come i comportamenti attesi durante l’esecuzione di alcune funzionalità. Vediamo, quindi, che la fase di testing prevede di verificare se ci siano differenze tra il funzionamento del sistema che si è progettato e il reale funzionamento del sistema implementato.

* 1. **Relazione con il documento di analisi dei requisiti (RAD)**

La relazione tra la fase di testing e la fase di analisi dei requisiti si basano sui requisiti funzionali e non funzionali descritti nel RAD che devono essere rispettati dal sistema durante la fase di testing.

* 1. **Relazione con il System design document (SDD)**

La relazione tra la fase di testing e la fase di design del sistema si basa sulla suddivisione del sistema in sottosistemi e la divisione in strati (Presentation layer, Application layer, Storage layer). Il testing deve essere fedele alla suddivisione progettata in fase di design in modo tale da rimanere coerente con il sistema che si è progettati.

* 1. **Relazioni con l’Object design document (ODD)**

La relazione fra la fase di testing e la fase di object design si basa principalmente sul riferimento della verifica dei contratti e dei componenti raffinati all’interno di tale documento.

3 Panoramica del Sistema

La struttura definita dentro il System design document (SDD) prevede un architettura client-server three-trial basata sulla suddivisione in 3 strati:

* Presentation Layer – Strato che si occupa della presentazione dei risultati di una computazione agli utenti del sistema e di raccogliere gli input degli utenti
* Application layer – si occupa di fornire all’applicazione specifiche funzionalità
* Storage Layer – si occupa della gestione dei database di sistema

Il sistema, inoltre, è stato diviso in sottosistemi a secondo della gestione di una particolare categoria di servizi. La suddivisione è fatta nel seguente modo:

* Gestione utente
* Gestione corso
* Gestione lezioni
* Gestione pagamenti
* Gestione mail

Quasi tutte le gestioni prevedono operazioni di inserimento, modifica, cancellazione, visualizzazione e ricerca di dati. Saranno proprio tali funzionalità ad essere oggetto di testing del sistema. Ricordiamo, infine, che solamente i primi tre sottosistemi verranno testati prima del rilascio del progetto, qualsiasi servizio legato alla gestione dei pagamenti e di mail dipendenti dai sottosistemi da testare saranno sviluppati come stub o driver a secondo del tipo di dipendenza.

4 Funzionalità da testare e non testare

Di seguito saranno elencate tutte le funzionalità da testare in relazione al sottosistema a cui appartengono.

1. Gestione utente

* Login
* Logout
* Registrazione

1. Gestione corsi

* Crea corso
* Modifica corso
* Verifica corso
* Visualizzazione corso
* Iscrizione corso

1. Gestione lezioni

* Elimina lezione
* Inserimento lezione
* Visualizza lezione
* Inserimento commento

5 Criteri Pass/Failed

I dati di test verranno raggrupati in classi di equivalenza in grado di raggruppare elementi con le stesse caratteristiche, per tali classi sarà sufficiente testare le funzionalità per un singolo elemento per classe. Un input supererà il test se il comportamento atteso e quello risultate combaceranno, ossia se l’output desiderato è uguale all’output ottenuto. Il responsabile del testing saprà a priori quale sia l’output desiderato.

6 Approccio

Verrà applicata una strategia di testing bottom-up. Si inzierà con il testing di unità dei singoli componenti, in modo da testare nello specifico le unità atomiche nella loro correttezza. Seguirà il testing di integrazione che focalizzerà l’attenzione sulle interfacce dell’unità. Infine, verrà eseguito il testing di sistema che vedrà la verifica del comportamento dell’interno sistema assemblato partendo dalle sue componenti principali. Il testing di sistema è importante per verificare se le caratteristiche richieste dal committente vengono rispettate o meno.

**6.1 Testing di unità**

Durante questa fase, verranno ricercate le condizioni di fallimento all’interno delle singole componenti e, usando test driver e stub raffiguranti implementazioni parziali di componenti che dipendono o da cui dipendono le componenti testate, si verificano i comportamenti della singola unità. La strategia utilizzata per il testing si baserà esclusivamente sulla tecnica Black-Box. Questa scelta strutturerà il testing unitario in un’analisi Input/Ouput delle singole componenti andando ad astrarre la verifica della struttura interna. Per minimizzare i casi di test, gli input verranno divisi in classi di equivalenza e ogni componente avrà un singolo caso di test per ogni classe di equivalenza strutturata. In questa fase, quindi, si avrà particolare attenzione sulla suddivisione delle classi degli input cosi da poter verificare ogni componente su ogni possibile tipo di input del dominio. La gestione del caso di errore, ossia lo stato in cui il comportamento atteso non è equivalente al comportamento ottenuto, comporterà un aggiornamento del documento di Incident Report; Tale documento verrà utilizzato per informare gli sviluppatori della presenza di un fault in modo tale da poterlo correggerlo tempestivamente per poi ripassare ad una verifica della correzione. La strategia del report di fault verrà estesa anche per le altre fasi di testing.

**6.2 Testing di integrazione**

Questa fase di test prevede l’aggregazione delle singole componenti e il loro testing adottando una strategia bottom-up. Si passerà poi alla fase di composizione successiva finchè non si avrà un’integrazione che esaurirà le funzionalità del sistema implementate. Questa strategia è utilizzata per diminuire la dipendenza tra le differenti funzionalità in modo da facilitare la ricerca degli errori nelle interfacce di comunicazioni tra i sottosistemi.

**6.3 Testing di integrazione**

In questa fase verrà utilizzata nuovamente la strategia black-box per il testing delle funzionalità dell’intero sistema e verificare se i requisiti e i vincoli di progettazione sono stati rispettati. In seguito a tale test, se superato in maniera corretta, si avrà un sistema pronto all’uso per l’utente finale. Ci si concentrerà principalmente sul testing delle funzionalità principali basate sulle priorità dei requisiti specificati durante la fase di analisi.