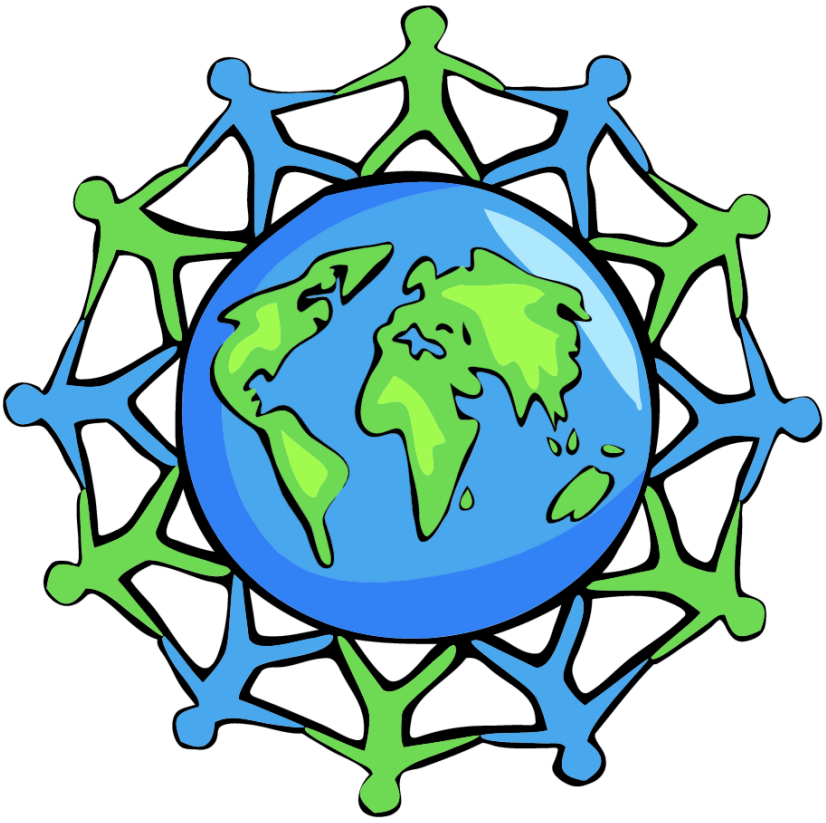
Laurea Magistrale in informatica - Università di Salerno

Corso di *Ingegneria, Gestione ed Evoluzione del Software*

Prof. De Lucia Andrea

Test Plan



Riferimento	
Versione	1.3
Data	3/06/2021
Destinatario	Prof.re De Lucia
Presentato da	Alice Vidoni e Salvatore Amideo

Data	Versione	Descrizione	Autore
3/06/2021	1.0	Creazione del documento TestPlan con intro, funzionalità da testare -Approccio	Salvatore Amideo Alice Vidoni

Indice

1. Introduzione

Nel documento in oggetto verranno definiti gli approcci e le attività di testing in merito alle nuove funzionalità aggiunte alla piattaforma web EasyAgreement. Verranno identificate le funzionalità della piattaforma da testare o meno, approcci e strumenti da utilizzare per l'attività di testing. L'obiettivo principale del testing è quello di identificare e prevedere quanti più difetti possibili di un'applicazione tramite i suoi malfunzionamenti per evitare che essi possano verificarsi durante il normale funzionamento. Per cui, diremo che le attività di testing avranno avuto successo se riusciremo ad identificare quanti più possibili fault e failure presenti nel sistema prima che il nuovo software venga messo in esercizio. Considereremo solo le gestioni aventi requisiti con priorità alta o media, e quindi:

- Gestione Commissari
- Gestione Colloqui
- Gestione Graduatoria

2. Documenti correlati

Il test plan è strettamente collegato ai documenti prodotti fino ad ora, poiché prima di passare alla fase di testing abbiamo bisogno di avere una gran parte delle funzioni previste già implementate, queste sono state definite nei precedenti documenti. Di seguito verranno descritte le relazioni che ci sono fra il test plan e gli altri documenti

2.1 Master Document (MD)

La relazione tra test plan e MD riguarda in particolare i requisiti funzionali e non funzionali del sistema poiché i test che saranno eseguiti su ogni funzionalità terranno conto delle specifiche espresse nel MD.

2.2 Impact Analysis (IA)

Nell' impact analysis sono state individuate tutte le componenti del sistema che verranno sottoposte a modifiche. Queste componenti dovranno essere testate utilizzando attività di testing funzionale.

3. Funzionalità da testare

Avendo già la documentazione relativa al sistema prima delle modifiche, riportiamo solo le nuove funzionalità da testare e quelle che hanno subito modifiche.

Di seguito vengono elencate le singole funzionalità da testare per ogni sottosistema:

- Gestione Commissari
  - Login
  - Modifica Pawword
  - Inserimento nuovo commissario
  - Rimozione di un commissario
- Gestione Colloqui
  - Creazione nuovo colloquio
- Gestione Graduatorie
  - Assegnazione punteggio

Ogni qualvolta sarà testata una di queste nuove funzionalità elencate sopra, verrà svolto test di regressione per verificare se sono state aggiunte delle failure all'interno delle funzionalità del vecchio sistema.

4. Criteri Pass/Field

Una volta individuati i vari dati di input del test, questi verranno raggruppati in base a caratteristiche comuni in insiemi. In questo modo 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. Di contro, il testing fallirà se l'output osservato sarà uguale all'oracolo.

5. Approccio

La fase di testing sarà suddivisa in tre fasi:

1. **Testing di unità:**
   verrà testata nello specifico il funzionamento di ogni singola unità del sistema;

2. **Testing di integrazione:**
   dove verranno testate le interfacce delle suddette unità;
3. **Testing di sistema:**
   dove verrà testato l'intero sistema assemblato.

5.1 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.

5.2 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 portare alla problematica 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.

5.3 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 sono 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

6. Materiale per il testing

L'hardware necessario alla attività di testing è un PC e si necessita una copia locale del DB.