**Test Plan**

1. **Introduzione**

Il documento è realizzato al fine di pianificare la fase di testing, ha come obbiettivo di verificare che non ci siano incongruenze tra comportamento atteso ed effettivo. Il piano si baserà sull’ottenere quanti più errori possibili al fine di risolverli, diminuendo la probabilità che si presentino durante l’effettivo utilizzo del sistema.

Verranno testate le funzionalità comprese nei sottosistemi che abbiamo definito in fase di System

Design, quindi: gestione utenti, gestione bacheca e gestione notifiche ed inviti.

1. **Documenti Correlati**
   1. **Correlazione con il RAD**

In particolare, si riprenderanno i requisiti funzionali e non funzionali, per i funzionali andremo a testare il loro comportamento.

* 1. **Correlazione con SDD**

Nel documento di system design si è deciso di suddividere l’architettura in 3 livelli: presentation, application e storage, di conseguenza i test prenderanno in considerazione la suddetta divisione.

1. **Funzionalità da testare**

Data la suddivisione presente nel system design, di seguito verranno elencate le funzionalità che verranno testate in base al sottosistema di riferimento.

*Gestione utenti:*

* RF4: Registrazione inoccupato o azienda
* RF6: Login
* RF10: Modifica curriculum
* RF19: Rimozione di un utente

Non verranno testati i seguenti requisiti: RF3, RF5.

*Gestione Bacheca:*

* RF2: Ricerca degli annunci
* RF7: Candidatura ad un annuncio
* RF8: Visualizzazione delle candidature effettuate
* RF14: Visualizzazione del curriculum di un candidato
* RF15: Pubblicazione di un annuncio
* RF17: Rimozione di un annuncio

Non verranno testati i seguenti requisiti: RF1, RF11, RF12, RF16.

*Gestione Notifiche ed Inviti:*

* RF13: Contattare il candidato
* RF18: Segnalazione di un utente

Non verranno testati i seguenti requisiti: RF9, RF20.

1. **Criteri Pass/Failed**

Gli input verranno suddivisi in classi di equivalenza, cioè si raggrupperanno per caratteristiche comuni, per i quali sarà sufficiente testare un solo elemento rappresentativo. Se l’elaborazione dell’input genererà il risultato atteso (specificato nel test case), allora la classe di equivalenza presa in esame avrà superato il test.

1. **Approccio**

Il testing partirà dal test delle singole unità, in modo tale da verificare la loro correttezza prima di essere integrate nel sistema. Una volta superato il test di unità, seguirà il test di integrazione, in cui la componente appena testata viene aggiunta alle altre componenti che hanno già superato il test di unità, e se ne analizza il funzionamento. Dopo che tutte le componenti hanno superato test di unità e test di integrazione, si passerà al test di sistema, in cui si verificherà il corretto funzionamento dell’intero sistema.

* 1. **Test di unità**

Il test di unità considera una componente isolata che verrà testata attraverso un’implementazione parziale sfruttando test driver e test stub, con l’ausilio di tecniche black-box. Il test driver ha l’obbiettivo di inviare input alla componente e di ricevere l’output risultante, così da poterlo confrontare con un oracolo. Se la componente da testare ha delle dipendenze con altre componenti, allora verrà realizzato un test stub, ovvero un’implementazione fittizia di tale componente.

* 1. **Test di integrazione**Una volta che una componente ha superato il test di unità, può essere integrata con le altre componenti già testate. La strategia di integrazione sarà di tipo bottom-up ed il procedimento verrà iterato per tutte le funzionalità implementate.
  2. **Test di sistema**

Una volta che tutte le componenti hanno superato il test d’integrazione, si può passare al testing dell’intero sistema, dove verrà dimostrato che il sistema risponde a tutti i requisiti definiti nel RAD.

Anche in questo caso utilizzeremo un approccio black-box per testare globalmente le funzionalità.

1. **Sospensione e ripresa**
   1. **Criteri di sospensione**

Il testing di un sottosistema dovrà essere interrotto nel caso in cui sottosistemi dipendenti non sono ancora disponibili o presentano degli errori; nel caso in cui invece non si hanno dipendenze, il testing può procedere senza sospensioni.

* 1. **Criteri di ripresa**

Il testing di un sottosistema sarà ripreso quando sarà stato comunicato dagli sviluppatori che gli errori riscontrati sono stati risolti.

In caso di re-testing di un sottosistema in cui sono stati riscontati errori, non è possibile riprendere il test da dove era stato interrotto, ma dovrà essere rieseguito completamente per verificare che non sono stati introdotti nuovi errori.

1. **Materiale per il testing**

Si necessita di un computer che supporti Javascript ed abbia una connessione a internet.

Utilizzeremo inoltre strumenti di supporto software per l’automazione dell’esecuzione del testing, in particolare JUnit e Selenium.

1. **Test Cases**

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

1. **Pianificazione del testing**

Ogni membro del team si occuperà di sviluppare una o più componenti, ma la fase di testing sarà organizzata in modo tale che ogni sviluppatore sarà il tester di una componente da lui non implementata. In caso vengano riscontrati degli errori, il tester avrà il compito di compilare un Test Incident Report, al fine di comunicare allo sviluppatore che si necessita di una o più correzioni.