Test Plan

1. Introduzione

La realizzazione di un buon sistema software richiede che al centro dei processi di sviluppo, manutenzione ed evoluzione del software stesso vi sia il concetto di qualità.

Il Testing rappresenta una delle più importanti tecniche per verificare la qualità del software, in quanto consente di analizzare, valutare e promuovere il miglioramento della correttezza dell'implementazione con riferimento alle caratteristiche definite in particolare dal modello dei requisiti software. Lo scopo di tale attività è quello di provare il sistema e rilevare problemi.

Tale attività va in contrasto con quelle svolte in precedenza: analisi, design, implementazione sono attività costruttive mentre il testing tenta di "rompere il sistema".

2. Panoramica del Sistema

UFL è un sistema software che implementa il gioco del FANTACALCIO, caratterizzato da un'architettura di tipo Model-View-Controller, in quanto prevede un controller che si occupa della logica di sistema, il model che si occupa di accedere ai dati persistenti mentre la vista si occupa di offrire un'interfaccia capace di comunicare con il controller e fruire i dati persistenti a front-end.

Tra i suoi sottosistemi, dunque, è presente quello volto ad ospitare le funzionalità relative alla gestione del database MySql.

Il controller offre delle API, espandibili e riutilizzabili, che permettono di soddisfare i servizi richiesti dal client per la visualizzazione dei dati all'interno di un Browser interpellando il model.

La Vista comunica con il controller con oggetti JSON, in modo da minimizzare il flusso di dati dal server al client. Ogni richiesta risponde con l'esito della stessa, permettendo in ogni evenienza di gestire al meglio gli errori aumentando così la distinzione tra i layer.

Le attività di testing saranno rivolte principalmente a testare le API di back-end e verificare l'integrazione con l'interfaccia grafica.

Avendo adottato una strategia implementativa con tendenze alle metodologie AGILE, ogni parte viene testata a front-end e back-end cosi da rilasciare porzioni di prodotto funzionanti e testare man mano le integrazioni.

Le caratteristiche da testare per il controllo del corretto funzionamento di ciascuna funzionalità saranno:

- robustezza: capacità del sistema di reagire fornendo input non valido per il dominio applicativo;
- usabilità: verrà testata l'interfaccia grafica i requisiti espressi dai mockups scenario;
- sicurezza: verrà testato che i permessi siano rispettati e che non siano accessibili i dati persistenti tranne se loggati nel sistema;
- correttezza: verrà testato che le operazioni vengano eseguite correttamente così come dalla specifica dei requisiti sono stati designati.

3. Relazione con gli Altri Documenti

Le relazioni tre il Test Plan sono necessarie al fine di produrre una fase di testing coerente con tutte le specifiche dedotte durante l'analisi e sia le decisioni che gli obiettivi che caratterizzano il System Design e la fase di Object Design. Inoltre, sono importanti anche per il raffinamento dei requisiti e delle funzionalità che il sistema dovrà fornire.

3.1. Relazioni con il documento di Analisi dei Requisiti

I test dovranno tenere conto delle specifiche espresse nel RAD. Naturalmente, si darà una maggiore rilevanza ai "casi limite", cioè tutti quei casi che sono al limite del nostro dominio di dati d'ingresso. Si darà rilevanza anche ai requisiti non funzionali ed ai vari modelli prodotti in fase di analisi dei requisiti.

3.2. Relazioni con il documento di System Design

Il testing dovrà garantire la coerenza tra il software e gli obiettivi di design definiti in fase di System Design, specificati nel SDD. Si analizzeranno, quindi, i conflitti e le inconsistenze presenti tra le componenti del sistema testate e tali obiettivi. Si analizzerà, inoltre, la struttura del sistema al fine di scoprire le differenze presenti tra essa e quella prevista durante la fase di System Design.

3.3. Relazioni con il documento di Object Design

La fase di testing dovrà considerare il contenuto del documento di Object Design, in quanto quest'ultimo rappresenta la base per la realizzazione dell'implementazione, fondamentale per il testing. Pertanto, sarà necessario effettuare il testing delle unità per individuare le differenze tra ciò che è stato stabilito in fase di Object Design ed il sistema effettivo.

4. Funzionalità da Testare e da non Testare

Si è deciso di testare soltanto alcune delle funzionalità che il sistema UFL offre e che sono state implementate. Tali funzionalità sono quelle definite nelle fasi di Raccolta e di Analisi dei Requisiti e le funzionalità che garantiscono un controllo su queste ultime. Si è stabilito, dunque, di evitare il testing di gran parte delle funzionalità non richieste dal cliente o che servono come semplice supporto a quelle testate. Ad esempio, sarà evitato il testing di componenti prettamente se non esclusivamente grafiche e componenti molto elementari.

4.1. Funzionalità da Testare

Le funzionalità da testare sono divise nelle seguenti categorie:

- Login e gestione dei permessi
- Logout
- Registrazione
- Api di back-end

4.2. Funzionalità da non Testare

Le funzionalità che non saranno testate sono divise in più categorie:

- funzionalità appartenenti a componenti prettamente grafiche che non offrono funzionalità di spicco per il corretto funzionamento del sistema
- funzionalità appartenenti a componenti molto semplici, il cui testing è implicito nella loro stessa struttura

Alla prima categoria appartengono tutte le API che possono essere testate semplicemente osservando ciò che è visualizzato dal software a livello grafico.
Alla seconda categoria appartengono metodi molto semplici come quelli che permettono esclusivamente la modifica o la restituzione di un campo di una determinata classe. Il testing di tali metodi è stato svolto implicitamente dagli implementatori del software, già a tempo di implementazione.

5. Approccio

Le tipologie di testing che si svolgeranno per testare il software UFL saranno tre:

- Unit Testing: un testing che coinvolge una singola unità del software e svolto indipendentemente dalle altre unità
- Integration Testing: un testing che coinvolge più unità che interagiscono tra loro e che sono integrate per essere testate insieme
- System Testing: un testing che coinvolge l'intero sistema e che quindi testa il software nella sua interezza, analizzando le interazioni tra tutte le unità, i gruppi di unità e le funzionalità

5.1. Unit Testing

Sarà effettuato un testing per ogni unità del software individuata. Per ogni componente verranno costruiti uno o più test case "Black Box".

I test case "Black Box" saranno svolti analizzando unicamente il comportamento di I/O delle unità in varie situazioni, senza esaminare il codice.

5.2. System Testing

Dopo aver effettuato il testing d'integrazione, si svolgerà il testing dell'intero sistema software UFL con l'uso di molteplici test case. Si analizzeranno, in particolare, le differenze tra il comportamento effettivo software e quello previsto, dettato dal cliente e descritto nel Documento di Analisi dei Requisiti. Sarà svolto, quindi, un testing funzionale, che riguarderà l'insieme delle funzionalità del software implementato. Anche in tal caso si costruiranno test case di tipologia "Black Box", che potranno far individuare errori e fallimenti riscontrabili utilizzando il software.

6. Materiale per il Testing

Le risorse software necessarie per effettuare le tre tipologie di testing sono le seguenti:

- Server e client
- Browser web con supporto Javascript(Google Chrome, Firefox)
- Microsoft Office Word 2003
- SublimeText (editore di testo
- mySqlWorkBench
- Sequel Pro
- MAMP
- Debugger: Firebug di Firefox

Le risorse hardware necessarie per effettuare il testing sono:

- Computer fisso o portatile che simuleranno le iterazioni tra client e server attraverso un'unica macchina oppure client con browser e server con la piattaforma installata.

7. Test Case

Le varie fasi di testing necessiteranno ognuna di test case utili ad individuare errori ed anomalie sia analizzando il codice che provando la sua esecuzione.
Si sono individuate varie classi di equivalenza per ogni input che possa essere immesso per l'utilizzo di una o più componenti. In tal modo, è possibile sviluppare test case con input delle tipologie identificate per testare una o più unità.
Di seguito, sono elencate le classi di equivalenza che saranno prese in considerazione durante Unit Testing, Integration Testing e System Testing per sviluppare i test case.

Utente

1.1. Registrazione

Input	**Classe**	**Valido**	**Classe**	**Non Valido**
NomeSquadra	CE01	squadra ≠ ‘ ’	CE02	NomeSquadra = ‘ ’
Matricola	CE03	Matricola ≠ ‘ ’	CE04	matricola= ‘ ’ & matricola <=12
Nome	CE05	Nome ≠ ‘ ’	CE06	Nome = ‘ ’
Cognome	CE07	Cognome ≠‘ ’	CE08	Cognome =‘ ’
Password	CE09	Password ≠‘ ’	CE10	Password = ‘ ’
Username	CE11	Username ≠‘ ’	CE12	Username = ‘ ’
Facolta	CE13	Facolta ≠‘ ’	CE14	Facolta ≠ ‘ ’

1.2. Login

Input	**Classe**	**Valido**	**Classe**	**Non Valido**
Matricola	CE01	Matricola ≠‘ ’	CE02	Matricola>11, Matricola =”
Password	CE01	Password ≠‘ ’	CE02	Password =”

1.3 Test API back-end

Input	**Classe**	**Valido**	**Classe**	**Non Valido**
API	CE01	Matricola ≠‘ ’	CE02	Return “status”:”false”

8. Schedule del Testing

8.1. Responsabilità

- Unit Testing → Giuseppe Paglialonga

Il documento di Test Plan è stato assegnato a: Giuseppe Paglialonga.

8.2. Rischi

Il testing è una fase caratterizzata da rischi che possono influenzare negativamente il suo svolgimento.
Alcuni di questi rischi sono stati previsti. Inoltre, per quelli previsti è stata ideato un rimedio.

I rischi previsti per la fase di testing sono i seguenti:

- **Difficoltà nel testare una componente o più componenti raggruppate**
  Soluzione: revisione del codice della componente o dell'insieme di Componenti, del suo scopo, della sua corrispondente funzionalità espressa nel RAD; ricostruzione di driver e stub con consigli degli implementatori delle componenti coinvolte
- **Difficoltà nel costruire driver o stub**
  Soluzione: osservazione di driver e stub costruiti per funzioni simili; consultazione degli implementatori delle componenti per cui è difficile costruire driver o stub
- **Numero stimato di driver e stub troppo elevato**
  Soluzione: cambio della strategia di testing; riuso e modifica di driver e stub per funzioni simili
- **Tempi di consegna del software non rispettati per via di un testing eccessivamente lungo in termini di tempo**
  Soluzione: aumento delle ore dedicate al testing all'interno della giornata lavorativa.

8.3. Training

Le conoscenze richieste per svolgere la fase di testing descritta in questo documento richiede una buona conoscenza del dominio del gioco del Fantacalcio.