TP:

Test Plan

Riferimento	
Versione	1.1
Data	07/01/2022
Destinatario	Prof. Carmine Gravino
Presentato da	Danilo Aliberti
Approvato da	

Data	Versione	Descrizione	Autore
29/12/2021	0.1	Stesura iniziale	Danilo Aliberti
02/01/2022	1.0	Aggiornamento	Danilo Aliberti
07/01/2022	1.1	Revisione e aggiornamento	Danilo Aliberti

Sommario

1. Introduzione

La fase di testing rappresenta la fase finale nella creazione di un sistema. Arrivati a questo punto, il focus è quello di provare se tutto funzioni correttamente, in modo da offrire ai destinatari del prodotto uno strumento che non presenti errori e agisca in modo conforme a quanto pianificato. Testare il sistema ci permette, quindi, di assicurare ai fruitori di questo un'esperienza di utilizzo che non riscontri problematiche e permetta agli utenti che ne fanno uso di migliorare la qualità dei propri servizi e gestire al meglio un campionato fantacalcistico.
A tal proposito, in questo documento è definito il piano di test utilizzato, il cui obiettivo principale è quello di analizzare e gestire lo sviluppo delle attività di testing relative al sistema proposto. Per testare il sistema è necessario che questo sia messo alla prova, sotto determinate condizioni. A questo fine sono stati ideati vari casi di possibili dati di input in grado di mettere alla prova le funzionalità presenti sulla piattaforma.
I risultati dei test che verranno eseguiti saranno il punto cruciale nell'analisi delle failure e delle loro cause (fault), per individuare dove bisognerà intervenire per correggere eventuali errori o apportare modifiche, per il miglioramento dei vari sottosistemi. Lo scopo è quindi verificare se esistono incongruenze tra il comportamento atteso e il comportamento osservato.

2. Documenti Correlati

Il test plan si trova in stretta relazione con il resto della documentazione prodotta finora, di seguito saranno descritte le diverse correlazioni con i vari documenti.

2.1 Relazione con il Requirement Analysis Document

I test eseguiti sulle varie funzionalità del sistema terranno conto delle specifiche contenute nel suddetto documento. In particolare, si fa riferimento agli scenari, use case ma soprattutto ai requisiti funzionali e non funzionali del sistema poiché i test che verranno eseguiti su quelle funzionalità terranno conto delle specifiche espresse nel documento.

2.2 Relazione con il System Design Document

Nel System Design Document è stata definita la suddivisione in sottosistemi del sistema. Il sistema è suddiviso in: Model, View e Template. La specifica di ognuno di essi è importante per la fase di testing; infatti, il test deve tenere conto di queste suddivisioni. In particolare, l'SDD contiene

informazioni rilevanti al test quali l'architettura del software corrente e proposto e i servizi dei sottosistemi.

2.2 Relazione con l'Object Design Document

Nel documento di Object Design sono state definite le classi che compongono il sistema e le loro mansioni, oltre ai packages e le class interfaces. Queste informazioni saranno quelle su cui ci si baserà per il test.

3. Panoramica del sistema

Biaset è una piattaforma web di gestione dei campionati fantacalcistici che mira all'ottimizzazione del management delle leghe di Fantacalcio. Il sistema nasce per svecchiare l'attuale piattaforma, ormai obsoleta.
Il sistema proposto prevede tre attori differenti:

- **League Admin (LA):** ovvero l'amministratore generale della Lega. Colui che gestisce e coordina tutti i campionati e gli amministratori di campionato;
- **Championship Admin (CA):** il sopracitato amministratore di campionato ovvero la persona preposta alla gestione di un singolo campionato;
- **Allenatore:** l'utente che partecipa ad un campionato. Sarà il proprietario di una squadra all'interno di un singolo campionato.

Nel System Design Document è stata definita l'architettura della piattaforma divisa in layer, seguendo il modello Model View Template (MVT). I sottosistemi rilevati, che vanno a comporre i vari livelli logici, collaborano tra loro, cercando di garantire il più possibile basso accoppiamento ed un'alta coesione.
Al fine di avere una struttura più definita, il sistema è stato suddiviso in più sottosistemi più piccoli qui sottoelencati:
Gestione utenza: è la parte che si occupa della gestione degli utenti registrati alla piattaforma e offre tutti i servizi relativi alla gestione dei dati personali. Gestisce l'autenticazione alla piattaforma da parte degli attori del sistema.
Gestione campionato: definisce ed offre tutti i servizi relativi alla gestione di un campionato fantacalcistico, dalla creazione del calendario degli scontri al calcolo della giornata fantacalcistica.
Gestione squadra: offre servizi relativi alla gestione della squadra di un Allenatore, come, ad esempio, il licenziamento di uno o più giocatori della squadra.

Gestione messaggistica: definisce tutti i servizi relativi alla messaggistica tra gli utenti della piattaforma, in modo da permettergli una comunicazione interna.

4. Funzionalità da testare

Il testing riguarderà nel dettaglio le funzionalità elencate di seguito, suddivise in base al sottosistema che le contiene:

- **Gestione Utenza**
  - Registrazione alla piattaforma tramite invito;
- **Gestione Campionato**
  - Inserimento campionato;
  - Inserimento formazione (titolari e riserve);
  - Modifica campionato;
  - Generazione calendario;
- **Gestione Squadra**
  - Aggiunta giocatore a squadra;
  - Rimozione giocatore a squadra;
  - Licenziamento giocatore;

Non saranno testate, invece, le parti riguardanti la sicurezza del sistema e le componenti off the shelf messe a disposizione dal Framework Django, in quanto già opportunamente testate.

5. Criteri Pass/Fail

Per poter stabilire se un test ha avuto successo, bisogna che questo riporti dei fault di sistema, nel caso in cui il comportamento osservato sia differente da quello specificato nei requisiti funzionali presenti nel RAD. Una volta riscontrato un fault, saranno analizzati i sottosistemi coinvolti nell'errore e sarà iterata la fase di testing per verificare che le modifiche apportate agli stessi non abbiano avuto impatti negativi su altre componenti del sistema. Per poter effettuare tutte queste operazioni e quindi aumentare le possibilità di riscontare fault, è necessario determinare ed utilizzare un insieme variegato di input che ci permetta di scovare quanti più errori nel sistema.

6. Approccio

Per testare diverse parti del sistema, sono spesso necessarie diverse strategie. Questa sezione descrive quelle individuate per effettuare i test di unità, integrazione e sistema.

6.1 Testing di unità

Il testing di unità si focalizza sul comportamento di una componente e sui building block del sistema (oggetti e sottosistemi), permettendo di eseguire testing in modalità black-box o white- box. Inoltre, permette il refactoring, quindi facilita la modifica del codice del modulo in momenti successivi, con la sicurezza che questo continuerà a funzionare correttamente. Il procedimento consiste nello scrivere dei test case per tutte le funzioni e i metodi, in modo che, se una modifica produce un fallimento del test, questa si possa facilmente individuare. Per realizzare il testing di ogni singola componente, il team ha alternato White-box testing, andando ad analizzare la struttura interna dei metodi e Black-box testing, andando ad esaminare le funzionalità dell'applicazione ed il comportamento input/output delle singole componenti senza tener conto della loro struttura interna. Per quanto riguarda il Black-box testing, il modulo supererà il test solamente se, per ogni input dato, l'output corrisponde a quello predetto; l'input sarà rappresentato sottoforma di classi d'equivalenza, scegliendo per ognuna un test case in modo da ridurre la ridondanza e rendere il test più efficiente. I risultati del testing verranno analizzati e usati per correggere gli errori che causano il fallimento del sistema.

6.2 Testing di integrazione

Dopo aver testato singolarmente le componenti del sistema attraverso il testing di unità, aver corretto gli eventuali errori da esso scaturiti ed averle integrate in sottosistemi più grandi, è possibile

procedere a testarne le integrazioni. Per effettuare l'integration testing la strategia utilizzata è la bottom-up: questa scelta permette di garantire la presenza di fondamenta solide alla base del sistema ma richiede la messa in campo di test driver per simulare le componenti dei layer più in alto che non sono stati ancora integrati.

6.3 Testing di sistema

Per il testing di sistema, che avviene testando i possibili input degli utenti, è stata adottata la soluzione del category partition tramite il riutilizzo dei test case utilizzati nella fase di testing di unità; per questa tipologia di test si è ricorso oltretutto all'utilizzo di Selenium IDE, al fine di osservare il comportamento del sistema in presenza di combinazioni di input utente non ammesse. Il testing di sistema concluderà la fase di test del prodotto. Tale testing sarà eseguito su un browser Google Chrome in locale, il quale riprodurrà i Test Cases registrati con Selenium IDE.

7. Sospensione e terminazione dei test

In questa sezione sono descritti i criteri stabiliti secondo i quali le attività dei test saranno sospese o terminate.

7.1 Criteri di sospensione

Il testing è sospeso se almeno il 10% dei casi di test riportano errori: in queste condizioni, il team deve provvedere a correggere i fault prima di procedere con l'implementazione o il testing di nuove funzionalità.

7.2 Criteri di terminazione

Il testing si considera terminato secondo un criterio di copertura, ovvero una percentuale predefinita degli elementi di un modello del software da testare. In particolare, il testing può essere interrotto al raggiungimento del 75% di branch coverage.

8. Materiale per il testing

L'esecuzione dei test necessita di un server correttamente configurato, sul quale sia installato **Docker.** La configurazione dovrà essere eseguita seguendo il manuale d'installazione.
Il testing è condotto utilizzando una libreria Python integrata nel framework Django: **unittest.** Come indicato nel punto precedente, i test sono eseguiti ad ogni modifica apportata al sistema. Per quanto riguarda le informazioni sulla coverage, il team si è affidato al tool **Coverage.py.**
Per quanto riguarda il testing di sistema, sarà necessario avere un browser Google Chrome installato in locale, con il plug-in Selenium IDE attivo.

9. Test Cases

Per sviluppare i test cases il metodo utilizzato è stato quello del category partition. Il category partition è un metodo per generare test funzionali da specifiche informali.
Si compone di tre passi:

1. *Decomporre le specifiche in feauture testabili indipendentemente:*
   In questo passo, il test designer deve identificare le feautures che devono essere testate in modo separato, identificando i parametri e qualunque altro elemento dell'ambiente di esecuzione da cui dipende.
2. *Identificare Valori Rappresentativi:*
   Questo passo, prevede che il test designer identifichi un insieme di valori rappresentativi per ciascuna caratteristica di ogni parametro definito nella fase precedente. I valori dovrebbero essere identificati per ciascuna categoria indipendentemente dai valori delle altre categorie.
3. *Generare Specifiche di Casi di Test:*
   In questa fase si stabiliscono dei vincoli semantici sui valori per indicare le combinazioni non valide e ridurre il numero di quelle valide. Una specifica di caso di test per una feauture è data da una combinazione di classi di valori, una per ciascuna caratteristica dei parametri. Il metodo del category partition permette di eliminare alcune combinazioni indicando classi di valori che non hanno bisogno di esser combinate con tutti gli altri valori.

9.1 CreazioneInvitoForm

Parametro: Destinatario **Formato ^[A-z0-9._%+-]+@[A-z0-9.-]+\.[A-z]{2,10}$**	
Lunghezza [LD]	1. Lunghezza >50 [error] 2. Lunghezza <=50 [PROPERTY **LD_OK**]

Formato [FD]	1. Non rispetta il formato [error] 2. Rispetta il formato [PROPERTY **FD_OK**]

Parametro: Data **Formato mm-dd-yyyy**	
Correttezza [CD]	1. Data <= DataCorrente [error] 2. Data > DataCorrente [PROPERTY **CD_OK**]
Formato [FDa]	1. Non rispetta il formato [error] 2. Rispetta il formato [PROPERTY **FDa_OK**]

ID	Test frame	Esito
TC_1.1_1	LD1	Errore: il destinatario è troppo lungo
TC_1.1_2	LD2, FD1	Errore: l'email non è corretta
TC_1.1_3	LD2, FD2, CD1	Errore: la data è precedente a quella odierna
TC_1.1_4	LD2, FD2, CD2, FDa1	Errore: la data non rispetta il formato
TC_1.1_5	LD2, FD2, CD2, FDa2	Corretto

9.2 CreaCampionatoForm

Parametro: NomeCampionato **Formato [A-Za-z0-9]**	
Lunghezza [LNc]	1. Lunghezza >50 [error] 2. Lunghezza <=50 [PROPERTY **LNc_OK**]
Formato [FNc]	1. Non rispetta il formato [error] 2. Rispetta il formato [PROPERTY **FNc_OK**]

Parametro: NumeroPartecipanti **Formato intero**	
Correttezza [CNp]	1. Intero < 6 [error] 2. Intero >10 [error] 3. Intero >=6 AND Intero <=10 AND dispari [error] 4. Intero >=6 AND Intero<=10 AND pari [PROPERTY **CNp_OK**]

ID	Test frame	Esito
TC_1.2_1	LNc1	Errore: il nome è troppo lungo
TC_1.2_2	LNc2, FNc1	Errore: il nome non rispetta il formato
TC_1.2_3	LNc2, FNc2, CNp1	Errore: il numero partecipanti è troppo piccolo
TC_1.2_4	LNc2, FNc2, CNp2	Errore: il numero partecipanti è troppo grande
TC_1.2_5	LNc2, FNc2, CNp3	Errore: il numero partecipanti è dispari
TC_1.2_6	LNc2, FNc2, CNp4	Corretto

9.3 ModificaNomeSquadraView [Allenatore]

Parametro: Nome **Formato [A-Za-z0-9]**	
Lunghezza [LN]	1. Lunghezza >50 [error] 2. Lunghezza <=50 [PROPERTY **LN_OK**]
Formato [FN]	1. Non rispetta il formato [error] 2. Rispetta il formato [PROPERTY **FN_OK**]

ID	Test frame	Esito
TC_1.3_1	LN1	Errore: il nome è troppo lungo
TC_1.3_2	LN2, FN1	Errore: il nome non rispetta il formato
TC_1.3_3	LN2, FN2	Corretto

9.4 ModificaCampionatoForm

Per questo form è possibile fare riferimento al test case specificato al punto precedente (9.3).

9.5 ModificaSquadraView [League/Championship Admin]

Parametro: Nome **Formato [A-Za-z0-9]**	
Lunghezza [LN]	1. Lunghezza >50 [error] 2. Lunghezza <=50 [PROPERTY **LN_OK**]
Formato [FN]	1. Non rispetta il formato [error] 2. Rispetta il formato [PROPERTY **FN_OK**]

Parametro: Allenatore

Formato [0-9]{1,9}	
Correttezza [CA]	1. Allenatore possiede già una squadra [error] 2. Allenatore non possiede una squadra [PROPERTY **CA_OK**]

ID	Test frame	Esito
TC_1.4_1	LN1	Errore: il nome è troppo lungo
TC_1.4_2	LN2, FN1	Errore: il nome non rispetta il formato
TC_1.4_3	LN2, FN2, CA1	Errore: l'allenatore già è associato ad una squadra
TC_1.4_4	LN2, FN2, CA2	Corretto

9.6 InserisciSquadraView [League/Championship Admin]

Parametro: Nome **Formato [A-Za-z0-9]**	
Lunghezza [LN]	1. Lunghezza >50 [error] 2. Lunghezza <=50 [PROPERTY **LN_OK**]
Formato [FN]	1. Non rispetta il formato [error] 2. Rispetta il formato [PROPERTY **FN_OK**]

Parametro: Campionato **Formato [0-9]{1,9}**	
Formato [FC]	1. Non rispetta il formato [error] 2. Rispetta il formato [PROPERTY **FC_OK**]

Parametro: Allenatore **Formato [0-9]{1,9}**	
Correttezza [CA]	1. Allenatore possiede già una squadra [error] 2. Allenatore non possiede una squadra [PROPERTY **CA_OK**]

ID	Test frame	Esito
TC_1.5_1	LN1	Errore: il nome è troppo lungo

TC_1.5_2	LN2, FN1	Errore: il nome non rispetta il formato
TC_1.5_3	LN2, FN2, FC1	Errore: il campionato non rispetta il formato
TC_1.5_4	LN2, FN2, FC2, CA1	Errore: l'allenatore possiede già una squadra
TC_1.5_5	LN2, FN2, FC2, CA2	Corretto

10. Glossario

Test Plan: Test che si focalizza sugli aspetti manageriali.

Test Case Document: documento di testing che contiene una rappresentazione di un insieme di condizioni o variabili sotto le quali un tester determina se una applicazione o sistema software risponde correttamente o meno.

Test Incident Report: Il Test Incident Report documenta tutte le problematiche trovate durante questa fase di test. Questo documento specifica i risultati previsti dal test, quando un test fallisce e qualsiasi indicazione del perché un test fallisce.

Test Summary Report: documento che contiene un riepilogo di tutte le attività di test e i risultati finali dei test di un progetto di test.

Pass/Fail Criteria: criteri per determinare il successo o il fallimento di un test case.

Testing di unità: Trova fault isolando una componente individuale usando test stub e test driver e esercitando la componente tramite un test case.

Testing di integrazione: Trova fault integrando differenti componenti insieme.

Testing di sistema: Si focalizza sul sistema completo, i suoi requisiti funzionali e non funzionali, e il suo ambiente.

Black Box Testing: Si focalizza sul comportamento I/O. Non si preoccupa della struttura interna della componente.

Classi d'equivalenza: Si focalizza sul comportamento I/O. Non si preoccupa della struttura interna della componente.