TP:
Test Plan

Transport Efficiency Manager

Riferimento	
Versione	0.3
Data	07/03/2021
Destinatario	Prof.ssa F. Ferrucci
Presentato da	Team NC08
Approvato da	

Revision History

Data	Versione	Descrizione	Autori
27/02/2021	0.1	Versione iniziale	Francesca Moschella, Federica Attianese, Federica Pica
06/03/2021	0.2	Aggiornamenti e modifiche	Francesca Moschella, Federica Attianese Federica Pica
07/03/2021	0.3	Revisione finale	Francesca Moschella, Federica Attianese Federica Pica

Indice dei contenuti

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 effettivamente tutto funzioni correttamente, in modo da offrire ai destinatari del prodotto, quindi le aziende di trasporto, 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 alle aziende che ne fanno uso di migliorare la qualità dei propri servizi e gestire al meglio le risorse a disposizione nel proprio asset.

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, quindi testato, 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, Controller e View. 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 package e i class interface. Queste informazioni saranno quelle su cui ci si baserà per il test.

3. Panoramica del sistema

Transport Efficiency Manager è una piattaforma web che mira all'ottimizzazione dei servizi fruibili dalle utenti dei mezzi di transporto pubblici e non offerti da aziende di transporto. Il sistema che proponiamo prevede due attori principali:

• **Admin**: ha la possibilità di consultare e gestire tutte le aziende registrate sulla piattaforma ed i dati e le informazioni che queste hanno inserito sulla piattaforma.

• **Utente** (Addetto dell'azienda): può usufruire delle funzionalità di generazione dei programmi di corsa, di inserimento e gestione delle risorse dell'asset aziendale e disporre quindi di tutte le principali funzionalità della piattaforma.

Nel System Design abbiamo definito l'architettura della piattaforma divisa in layer (seguendo il modello MVC). I sottosistemi rilevati, che compongono i vari livelli logici, collaborano tra loro cercando di garantire il più possibile basso accoppiamento ed alta coesione.

Al fine, però, di avere una più definita strutturazione della piattaforma, il sistema è stato suddiviso in sottosistemi più piccoli, in particolare è stato diviso per gestioni:

Gestione utente: definisce l'utente del sistema ed offre tutti i servizi relativi alla registrazione sulla piattaforma; per l'azienda attraverso un solito form, mentre per l'admin con credenziali già predisposte e preesistenti nel database.

Gestione autenticazione: definisce ed offre tutti i servizi relativi all'autenticazione degli attori del sistema.

Gestione risorse: modella tutte e tre le tipologie di risorse inseribili (mezzo, linea, conducente) e le operazioni ad esse relative.

Gestione dati corsa: modella i dati corsa collezionati dall'azienda e tutte le operazioni eseguibili su questi.

Gestione programma corse: modella le due diverse tipologie di programmi corsa generabili (automatico e manuale) e le operazioni che è possibile effettuare su di essi.

4. Funzionalità da testare

L'obiettivo di questa fase sarà quello di testare interamente la comunicazione tra webapp e database sottostante, così come l'implementazione della logica di business e dei servizi offerti dai sottosistemi del livello centrale. Il testing funzionale riguarderà nel dettaglio le seguenti funzionalità, elencate in base al sottosistema che le realizza:

- **Gestione utente:**
  - Registrazione alla piattaforma;
- **Gestione autenticazione:**
  - Accesso alla piattaforma;

- **Gestione risorse:**
  - o Inserimento mezzo;
  - o Modifica mezzo;
  - o Eliminazione mezzo;
  - o Inserimento linea;
  - o Modifica linea;
  - o Eliminazione linea;
  - o Inserimento conducente;
  - o Modifica conducente;
  - o Eliminazione conducente;
- **Gestione dati corsa:**
  - o Inserimento dati corsa;
  - o Modifica dati corsa;
  - o Eliminazione dati corsa;
- **Gestione programma corse:**
  - o Creazione programma manuale;
  - o Creazione programma automatico (modulo di intelligenza artificiale);
  - o Eliminazione programma;
  - o Modifica programma (che riguarda l'aggiunta, la modifica e la cancellazione delle corse associate);
- **Non saranno testate invece:**
  - o Sicurezza, in quanto il framework Spring Security è già opportunamente testato;
  - o Performance.

5. Criteri Pass/Fail

Per poter stabilire se il test ha avuto successo bisogna che questo riporti dei fault nel sistema, se quindi il comportamento osservato è diverso dal comportamento specificato nei requisiti funzionali presenti nel RAD. Una volta riscontrato un fault, saranno analizzati i sottoinsiemi coinvolti nell'errore, e sarà iterata la fase di testing per verificare che le modifiche apportate agli stessi non abbiamo 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 l'utilizzo dei test case utilizzati nella fase di testing di unità; per questa tipologia di test si è ricorso oltretutto all'utilizzo del software Katalon Studio, 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.

7. Sospensione e terminazione dei test

In questa sezione sono descritti i criteri stabiliti in base ai quali le attività di 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.

7.3 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 su cui siano installati Java e MySQL. La configurazione deve avvenire come da manuale d'installazione. Il testing è condotto utilizzando alcuni dei framework più famosi ed efficaci in ambienti Java: JUnit, Mockito e PowerMockito. Ad essi viene affiancata tutta la suite di test relativa a Spring Boot, framework adottato per la realizzazione della webapp. Come detto al punto precedente, i test sono eseguiti ad ogni modifica apportata al sistema.

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.

10. Riferimenti ad altri documenti di test

Per la fase di testing di unità, i test case e le specifiche di ognuno di essi sono riportati nel **Test Case Document**.

La fase di testing di integrazione è descritta nel **Test Integration Document**.

Eventuali errori rilevati verranno riportati nel **Test Incident Report**.

I risultati ottenuti nella totalità della fase di testing saranno consultabili nel **Test Summary Report**.

11. 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