Test Plan Document

GreenLeaf

Riferimento	
Versione	1.0
Data	02/12/2022
Destinatario	Prof.ssa F. Ferrucci Prof F. Palomba
Presentato da	Angelo Afeltra, Antonio Giametta, Raffaele Squillante, Alessandro Borrelli, Vincenzo Cerciello, Michela Faella, Gerardo Napolitano, Mirko Vitale
Approvato da	

Revision History

Data	Versione	Descrizione	Autori
08/12/2022	1.0	Stesura completa fino al capitolo 9	A. Afeltra, R. Squillante, A. Giametta
08/12/2022	2.0	Stesura dei rimanenti capitoli	Team

Sommario

Revision History 2
1. Introduzione 4
2. Documenti correlati 4
   2.1. Relazione con il documento di analisi 4
   2.2. Relazione con il System Design Document 4
   2.3. Relazione con l'Object Design Document 5
3. Panoramica del sistema 5
4. Funzionalità da testare 5
5. Criteri di Pass/Failed 6
6. Approccio 6
   6.1. Testing di unità 7
   6.2. Testing di integrazione 7
   6.3. Testing di sistema 7
7. Sospensione e ripresa 8
   7.1. Criteri di sospensione 8
   7.2. Criteri di ripresa 8
   7.3. Criteri di terminazione 8
8. Materiale per il testing 8
9. Test cases 9

1. Introduzione

Durante la realizzazione di un software, ci si pone sempre l'obiettivo di ottenere un buon prodotto. Al fine di garantire che ciò avvenga, è necessario realizzare un sistema che possa essere considerato affidabile. Nasce così la necessità di creare un prodotto che sia privo (o quasi) di errori prodotti durante la fase di implementazione per far si che il prodotto diventi uno strumento di cui l'utente finale si possa fidare.

A tal proposito è stato definito il seguente piano di test, il cui obiettivo è quello di analizzare e pianificare le attività di testing relative al sistema proposto. Visto che è necessario garantire il corretto funzionamento del sistema, sono stati pensati input e casi ditest specifici in modo da mettere alla prova le funzionalità offerte dal sistema stesso. I risultati dei test, che verranno eseguiti successivamente, saranno fondamentali al fine di individuare le aree su cui bisogna intervenire per rimuovere i fault presenti all'interno del sistema.

2. Documenti correlati

2.1. Relazione con il documento di analisi

La progettazione dei casi di test avviene ignorando la struttura interna del sistema e operando solo sulle specifiche. Grazie agli scenari e agli use case prodotti nel documento di analisi otteniamo in manieradettagliata le specifiche del sistema e per questo motivo è necessario far riferimento a questo documento.

2.2. Relazione con il System Design Document

I test case pianificati nel Test Plan devono rispettare la suddivisione in sottosistemi presentata nell'SDD.

2.3. Relazione con l'Object Design Document

Per ciò che concerne il test di unità e di integrazione, maggiormente legati all'ODD e alla divisione in package del sistema, essi saranno scritti e documentati unicamente all'interno del codice dell'applicativo. Per tale motivo, nel presente documento, non vi saranno riferimenti al loro design.

3. Panoramica del sistema

Green Leaf è una piattaforma web che ha lo scopo di diminuire l'inquinamento attuale i cui livelli, al giorno d'oggi, sono molto elevati. L'obbiettivo principale è quello di sensibilizzare andando ad evidenziare come potrebbe cambiare la situazione attuale attraverso un piccolo contributo, ovvero adottando un albero.
Tenendo conto degli obiettivi sopra elencati, abbiamo individuato i seguenti sottosistemi:

- Gestione Utente
- Sensibilizzazione
- Adozione Albero
- Informazioni Albero
- Gestione Piantumazioni

Il sistema proposto basa la sua architettura sul sistema three-tier, in particolare usando un sistema MVC (Model-View-Control).

Verranno usati HTML5, CSS3 e BootStrap per la parte di front-end e la generazione delle view.

Per la logica applicativa e quindi il back-end sarà utilizzato Java.

Per la gestione del database saranno usati:

- Java per il collegamento al database.
- MySQL come database.

4. Funzionalità da testare

Per quanto l'ideale sia testare ogni singolo componente del sistema, a causa del budget ridotto non è possibile pensare ad uno scenario del genere. Il team si occuperà pertanto di testare esclusivamente il livello di application logic a discapito delle interfacce utente. Questa scelta è stata presa in quanto errori sulla GUI non intaccherebbero negativamente sulla robustezza del sistema che invece viene garantita dal livello sottostante.

Le funzionalità che verranno testate sono riassunte nella seguente tabella:

Sottosistema	Funzionalità
Sensibilizzazione	Calcolo CO2 causata
Adozione Albero	Selezione albero
Sensibilizzazione	Monitoraggio inquinamento
Adozione Albero	Generazione regalo
Sensibilizzazione	Previsione inquinamento

Non verranno invece testate:

- Sicurezza
- Performance

5. Criteri di Pass/Failed

L'approccio utilizzato per testare il sistema sarà del tipo test to fail. L'obiettivo che ci poniamo è quello di individuare quanti più fault possibili durante le fasi di sviluppo in modo che, una volta rilasciato il software, esso contenga quanti meno fault possibili. L'approccio test to fail ci aiuta in questo senso in quanto spinge la nostra soluzione ai suoi limiti, evitando così che la conoscenza di ciò che abbiamo realizzato inganni la nostra fase di testing. Piuttosto invece incita a provare a riprovare una funzionalità fino a quando un errore non è stato individuato.
Pertanto, il test viene marcato come PASS se il comportamento osservato è diverso da quello atteso. In questo caso analizzeremo la causa dell'errore e verrà risolto. Il test viene marcato come FAILED nel caso in cui non vengano scovati errori nelle componenti.

Tutto il testing sarà considerato valido se tutti i seguenti vincoli saranno rispettati:

- Testare una funzionalita per ogni membro del team
- Ogni membro del team dovrà testare un metodo di una classe sviluppata

6. Approccio

Le attività da svolgere per realizzare il testing sono tre. Nella fase iniziale ci occuperemo di individuare gli errori su una singola componente; nella fase successiva invece, testeremo le funzionalità nate

dall'integrazione dei vari sottosistemi; infine, andremo a testare l'intero sistema per verificare che le caratteristiche richieste dal nostro committente siano rispettate o meno.

6.1. Testing di unità

Per il testing di unità la strategia prevista consiste nel testare ogni metodo delle classi del sistema. Da esse,sono escluse le interfacce e le classi entity, poiché quest'ultime presentano solo metodi getters e setters. I casi di test saranno definiti attraverso un approccio black-box e saranno documentati direttamente nelcodice, attraverso l'uso del framework per il testing di classi Java *JUnit*.

Per ogni Production Class sarà definita una Test Class che rispetterà il formato *NomeProductionClassTest*.Tali classi saranno scritte in parallelo alle Production class, per garantire una più facile copertura del codice. Le stesse classi saranno poi revisionate e modificate da sviluppatori differenti.

Altre tecnologie usate in tale fase saranno:

- *Mockito*: per la costruzione degli stub e l'isolamento della componente testata.
- *Maven*: per la build e l'esecuzione automatica dei tests

6.2. Testing di integrazione

Verrà utilizzato un approccio bottom-up, metodo ritenuto più adatto per un software basato sul paradigma Object Oriented. La definizione dei test case avverrà tramite il framework *JUnit*, mentre verràusato *Mockito* per il mocking. Verrà valutato l'utilizzo Travis CI per realizzare la Continuous Integration. L'automatizzazione del run dei test sarà gestita da *Maven*.

6.3. Testing di sistema

Il testing di sistema concluderà la fase di test del prodotto ed il primo ciclo di sviluppo. Per questa tipologia di test, ci affidiamo all'utilizzo di un software ausiliario come Katalon al fine di osservare ilcomportamento del sistema.

7. Sospensione e ripresa

7.1. Criteri di sospensione

La fase di testing verrà sospesa nel momento in cui 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'implementazioneo il testing di nuove funzionalità.

7.2. Criteri di ripresa

Visto che l'approccio per lo sviluppo del software è di tipo incrementale, la fase di testing potrebbe riprendere nel momento in cui verranno introdotti cambiamenti e/o nuove componenti. In questo caso, andranno testati i nuovi elementi introdotti e, tramite regression testing, anche quelli già precedentementetestati. Pertanto, faremo affidamento su un servizio che ci permetta di lavorare in un ambiente di Continous Integration.

7.3. Criteri di terminazione

Il test si considera concluso nel momento in cui verrà testa una funzionalità per ogni membro del team.

8. Materiale per il testing

L'esecuzione dei test necessita di un ambiente in cui siano installati Java SE 14 o successivi e MySQL. Il testing viene effettuato utilizzando i framework JUnit e Mockito, molto affermati in ambiente Java e il software Katalon.

Mentre JUnit viene utilizzato per il testing d'unità, Mockito viene utilizzato per mascherare le dipendenze tra le componenti. Katalon viene utilizzato per realizzare i test di sistema. Come evidenziato prima, si lavorerà in un ambiente di continous integration: ciò è possibile grazie all'utilizzo di Travis CI.

9. Test cases

In questa sezione, per ogni sottosistema verranno mostrate le funzionalità che verranno testate. Ogni funzionalità avrà una tabella per ognuno dei suoi parametri, che conterrà i vincoli affinché il valore dell'input sia valido. Infine, vi sarà una tabella che va ad indicare i test case individuati e i relativi esiti attesi dalle combinazioni dei vari vincoli: in caso di errore l'esito sarà uguale a ✗, altrimenti sarà uguale a ✓.

9.1 Calcolo CO2

Parametro: Trasporto	
Nome categoria	Scelta per la categoria
Presenza [PT]	1. Mezzo di trasporto non selezionato [errore] 2. Mezzo di trasporto inserito correttamente [PROPERTY_PT_OK]
Parametro: Kilometri	
Nome categoria	Scelta per la categoria
Correttezza [CK]	1. Kilometri <=0 [errore] 2. Kilometri >= 0 [PROPERTY_CK_OK]

Test Case ID	Test frame	Esito
TC_S_2.1_1	PT1	Errato: Selezione errata
TC_S_2.1_2	PT2, CK1	Errato: Kilometri non corretti
TC_S_2.1_3	PT2, CK2	Corretto: Calcolo CO2 effettuato

9.2 Selezione albero

Parametro: Albero	
Nome categoria	Scelta per la categoria
Selezione [SA]	1. Nessun albero selezionato [errore] 2. Albero selezionato correttamente [PROPERTY_SA_OK]
Parametro: Regione	
Nome categoria	Scelta per la categoria
Correttezza [CR]	1. Nessuna regione selezionata [errore] 2. Regione non compatibile[errore] 3. Dato selezionato regione compatibile [PROPERTY_CR_OK]

Test Case ID	Test frame	Esito
TC_AA_3.1_1	SA1	Errato: Nessun albero inserito
TC_AA_3.1_2	SA2, CR1	Errato: Nessuna regione selezionata
TC_AA_3.1_3	SA2, CR2	Errato: Regione non compatibile
TC_AA_3.1_4	SA2, CR3	Corretto: Albero selezionato correttamente

9.3 Monitoraggio inquinamento

Parametro: Scelta	
Nome categoria	Scelta per la categoria
Correttezza [CS]	1. Nessun dato selezionato [errore] 2. Dato selezionato regione o nazionalità [PROPERTY_CS_OK]

Test Case ID	Test frame	Esito
TC_S_2.3_1	CS1	Errato: Nessun dato inserito
TC_S_2.3_2	CS2	Corretto: Dato inserito correttamente

9.4 Generazione regalo

Parametro: E-mail	
Nome categoria	Scelta per la categoria
Correttezza [CE]	1. E-mail non presente nel Database [errore] 2. E-mail presente nel Database [PROPERTY_CE_OK]
Parametro: Password	
Nome categoria	Scelta per la categoria
Correttezza [CP]	1. Password errata [errore] 2. Password corretta [PROPERTY_CP_OK]
Parametro: Pagamento	
Nome categoria	Scelta per la categoria
Presenza [PP]	1. Metodo di pagamento assente [errore] 2. Metodo di pagamento presente [PROPERTY_PP_OK]

Test Case ID	Test frame	Esito
TC_AA_3.2_1	CE1, CP1	Errato: Accesso negato, credenziali errate
TC_AA_3.2_2	CE2, CP1	Errato: Password errata
TC_AA_3.2_3	CE1, CP2	Errato: E-mail errata
TC_AA_3.2_4	CE2, CP2, PP1	Errato: Mancato metodo di pagamento
TC_AA_3.2_5	CE2, CP2, PP2	Corretto: Generazione buono regalo

9.5 Previsione inquinamento

Parametro: Scelta	
Nome categoria	Scelta per la categoria
Correttezza [CS]	1. Nessun dato selezionato [errore] 2. Dato selezionato regione o nazionalità [PROPERTY_CS_OK]
Parametro: Data	
Nome categoria	Scelta per la categoria
Correttezza [CD]	1. Data<=DataCorrente [errore] 2. Data>=DataCorrente [PROPERTY_CD_OK]

Test Case ID	Test frame	Esito
TC_S_2.4_1	CS1, CD1	Errato: Nessun dato inserito e Data errata
TC_S_2.4_2	CS2, CD1	Errato: Dato inserito correttamente e Data errata
TC_S_2.4_3	CS1, CD2	Errato: Nessun dato inserito e Data corretta
TC_S_2.4_4	CS2, CD2	Corretto: Dato inserito correttamente e Data corretta