Laurea Magistrale in informatica-Università di Salerno
Corso di *Gestione dei Progetti Software*- Prof.ssa F.Ferrucci

TP

Test Plan

DryBlue

Riferimento	TP_C8
Versione	1.0
Data	20/12/2021
Destinatario	Prof.ssa F. Ferrucci
Presentato da	Piero Agosto, Michele Bisaccia, Sabrina Ceccarelli, Piero Dello Buono, Miriam Ferrara, Stefano Mungiello, Antonio Sellitto
Approvato da	Alessia Natale, Saverio De Stefano

Revision History

Data	Versione	Descrizione	Autori
19/12/2021	1.0	Introduzione	MF
19/12/2021	1.0	Relazione con altri documenti	MF
19/12/2021	1.0	Features da testare/da non testare	MB
19/12/2021	1.0	Testing di integrazione	MB
20/12/2021	1.0	Testing di sistema, di unità e altre informazioni	AN,SDS

Tabella dei contenuti

1. Introduzione

Il documento Test Plan ha l'obiettivo di descrivere ed analizzare le attività di Testing per il sistema DryBlue allo scopo di garantire al cliente che ogni aspetto funzioni in modo corretto.
All'interno del documento sono riportate le strategie di testing adottate, per valutare la conformità del sistema individuando il maggior numero di funzionalità e la frequenza di possibili errori che potrebbe causare un'inefficienza dell'utilizzo del sistema.
Inoltre, vedremo quali funzionalità saranno testate e con quali strumenti scelti saranno rilevati eventuali errori.

2. Relazione con altri documenti

Di seguito sono elencate le relazioni con altri documenti.

- RAD_DryBlue
- SDD_DryBlue
- ODD_DryBlue

Relazione con altri documenti di testing:

- TCD_DryBlue dove è possibile consultare le attività di testing specificate nei capitoli 3.Test Case Plan e 4.Test Case Specification

3. Panoramica del sistema

Il sistema da noi proposto è una web app che, allo scopo di garantire una buona coesione e un basso accoppiamento, si basa sul pattern architetturale Three Tier. Lo stile architetturale che utilizziamo divide i moduli che compongono i sottosistemi in 3 livelli separati:

- **Livello di presentazione:** è il livello più alto dell'applicazione e rappresenta l'interfaccia utente
- **Livello applicazione:** si occupa delle funzionalità dell'applicazione ed inoltre collega il livello dati al livello di presentazione scambiando i dati tra i due livelli;
- **Livello dati**: si occupa di memorizzare e recuperare dati dal database.

Le tecnologie che verranno utilizzati per sviluppare il sistema sono: **Java, HTML5, CSS3, JS, Bootstrap**. Il framework utilizzato per la logica del server è **Spring**, in particolare Spring Boot e Spring Web, mentre per il collegamento al database sarà utilizzato **Spring JPA.** Per il database in fase di produzione si è scelto di utilizzare H2.

4. Features da testare/da non testare

Di seguito la lista delle features di cui si effettuerà il testing per le varie gestioni:

- **Gestione Autenticazione**
  - Registrazione Cliente
  - Login
  - Logout
  - Reimposta Password
- **Gestione Ordini**
  - Creazione Ordine
  - Modifica Ordine Operatore
  - Proposta Modifica Ordine Cliente
  - Accettazione/rifiuto proposta modifica
  - Invio notifiche aggiornamenti
  - Stampa di un'etichetta
- **Gestione Servizi Lavanderia**
  - Aggiunta Nuovo servizio
  - Aggiunta Nuovo Macchinario
  - Aggiornamento Stato Macchinario

Le funzionalità di cui non si andrà ad effettuare le attività di testing riguardano requisiti funzionali di bassa o media priorità; sono inoltre escluse le funzionalità che non prevedono input manuale da parte dell'utente come, ad esempio attività riguardanti esclusivamente visualizzazioni di dati.

5. Pass/Fail criteria

Le attività di testing sono mirate ad identificare la presenza di errori (faults) all'interno del sistema, per effettuarne un successivo intervento di eliminazione.
L'esito di un test è valutato con il risultato atteso della sua esecuzione, basandosi sui requisiti. Un test ha successo (pass) se, dato un input al sistema, l'output ottenuto è diverso dall'output atteso dall'oracolo. Un test fallisce (fail) se, dato un input al sistema, l'output ottenuto è uguale all'output atteso dall'oracolo.
Tutto il testing sarà considerato valido se tutti i seguenti vincoli saranno rispettati:

- Testare tutti i requisiti funzionali ad alta priorità;
- Effettuare test di regressione ogni volta che si introducono nuove caratteristiche al sistema o vengono modificate quelle presenti;
- Raggiungere un branch coverage non inferiore al 75%.

6. Approccio

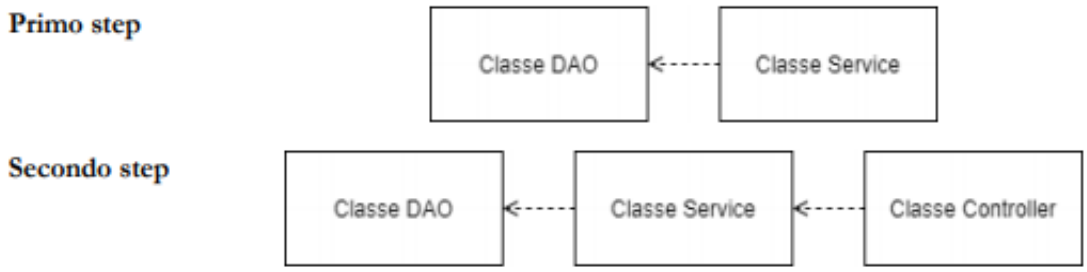
La fase di testing si compone di tre attività: una prima fase si occuperà di trovare errori in una singola componente; la seconda fase, invece, avrà come compito quello di testare le funzionalità nate dall'integrazione dei vari sottosistemi e per ultimo andremo a testare l'intero sistema assemblato al fine di verificare soprattutto che esso soddisfi i desideri del cliente. Di seguito verranno descritte brevemente le strategie individuate per effettuare il test di unità, d'integrazione e di sistema.

Testing di sistema

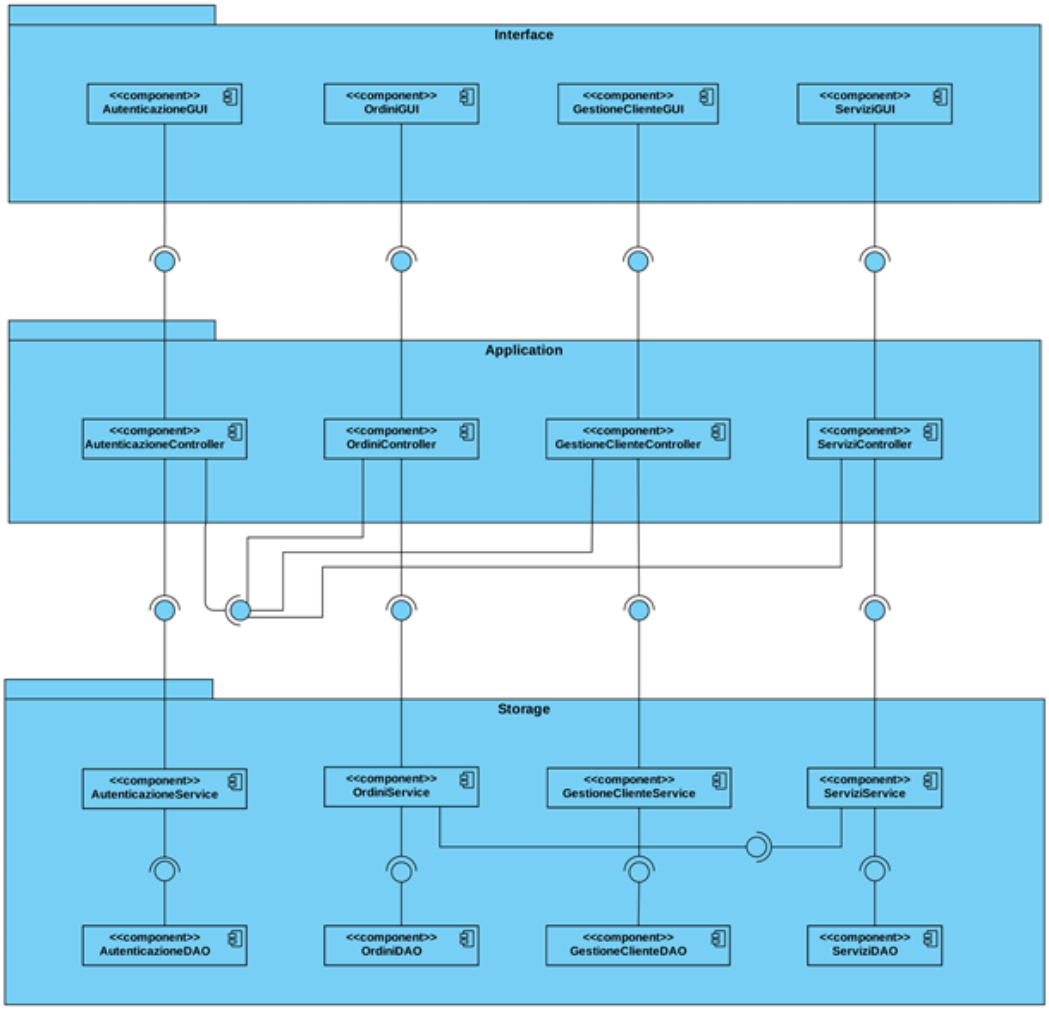
La verifica sulle funzionalità del sistema avviene testando i possibili input degli utenti. Si effettuerà quindi, un testing funzionale che riguarderà l'insieme delle funzionalità del software implementato e verranno utilizzati i test case individuati con il category partition.
In particolare, il testing 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 Selenium al fine di osservare il comportamento del sistema in presenza di combinazioni di input utente non ammesse.

Testing di integrazione

Nel testing di integrazione sarà utilizzato un approccio bottom-up che è un metodo ritenuto adatto per un software orientato agli oggetti e quindi basato sul paradigma Object Oriented. La definizione dei test case sarà effettuata con il framework JUnit, mentre verrà usato Mockito per il mocking. L'automatizzazione del run dei test sarà gestita da Maven, ed infine come tool di misurazione e report coverage sarà utilizzato Jacoco. Il test di integrazione sarà lo stesso per tutte le componenti da testare. Nello specifico, si procederà prima con il test delle classi Service, e successivamente con il test delle classi Controller. Durante questa seconda esecuzione, la chiamata al controller sarà effettuata usando Mockito. Di seguito viene rappresentato un esempio grafico di test di integrazione diviso nei due fasi.



Per ciò che concerne le dipendenze tra i sottosistemi, si riporta di seguito il diagramma architetturale.



Testing di unità

Durante la fase di testing è necessario testare singole componenti al fine di evidenziarne gli errori. Il testing di unità, infatti, si focalizza sul comportamento di una componente permettendo di eseguire testing in modalità black-box o white-box. Saranno escluse le interfacce e le classi entity in quanto quest'ultime presentano esclusivamente metodi getter e setter. Le nostre componenti saranno testate secondo il metodo white-box. Infatti, durante questa fase l'attenzione sarà focalizzata sulla struttura del codice che realizza le funzionalità fornite dalla componente al fine di individuare errori sia di logica che di implementazione.
Le classi di test verranno sviluppate parallelamente alle classi del sistema, e avranno il nome: classediproduzioneTest.
Per questa fase verranno utilizzati i tool Junit, mockito e maven.

7. Sospensione e ripristino

Tenuto conto delle risorse necessarie impiegate durante la fase di testing, abbiamo stabilito dei criteri in base ai quali le attività di test saranno sospese o riprese.

7.1. Criteri di sospensione

Il test 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 all'implementazione o al testing di nuove funzionalità.

7.2. Criteri di ripristino

Il testing riprenderà quando i fault che hanno causato le failure saranno risolti.

7.3. Criteri di terminazione

Il test si considera terminato quando la totalità dei casi di test somministrati al sistema riporta esito negativo. Come da indicazione del top management, la suddetta condizione sussiste solo se il 75% dei branch sviluppati viene ricoperto in questa fase.

8. Materiale di testing

Il materiale necessario ai fini del testing è in primis tutti gli artefatti precedentemente prodotti durante la fase di progettazione (RAD, SDD, ODD) e tutti i vari deliverables di testing in cui sono specificati i casi di test che il team dovrà eseguire.

9. Test cases

Per una visualizzazione più fluida i test cases sono presenti all'interno del seguente documento: TCD_DryBlue.

10. Testing schedule

Le attività di testing sono schedulate nel seguente modo:

- Pianificazione: dopo la fase di design;
- Scrittura casi di test: durante l'implementazione;
- Esecuzione dei test: durante e dopo l'implementazione.