Test Plan

Riferimento	TP
Versione	2.0.0
Data	12/12/2024
Destinatario	TMs
Presentato da	Raffaella Spagnuolo, Alessia Ture
Approvato da	Raffaella Spagnuolo, Alessia Ture

Revision History

Data	Versione	Descrizione	Autori
25/11/24	1.0.0	Prima Stesura	AT
12/12/24	2.0.0	Revisione	RS

Project Managers

Nome	Acronimo	Contatto
Raffaella Spagnuolo	RS	r.spagnuolo6@studenti.unisa.it
Alessia Ture	ATu	a.ture@studenti.unisa.it

Sommario

1 Introduzione

1.1 Obiettivo del documento

Il presente **Test Plan** definisce l'approccio, le strategie e le metodologie utilizzate per il testing del progetto *ZeroWaste Home*. Questo documento guida l'esecuzione del processo di testing al fine di garantire che il sistema soddisfi i requisiti funzionali e non funzionali specificati.

Il Test Plan include una descrizione delle funzionalità da testare, i criteri di accettazione per il successo dei test e il dettaglio delle attività di verifica. Inoltre, identifica gli strumenti e le risorse necessarie per condurre i test in modo efficace, minimizzando i rischi di rilascio di un prodotto non conforme.

2 Relazione con altri documenti

2.1 Documenti collegati

Questo Test Plan si basa sui documenti prodotti durante le fasi di analisi e progettazione del sistema *ZeroWaste Home*. I test case, i criteri di accettazione e le strategie di testing derivano dai dettagli tecnici e funzionali descritti in tali documenti. Poiché lo sviluppo del sistema è in corso, questo documento sarà aggiornato in seguito al rilascio di altri deliverable.

2.1.1 Relazioni con il Requirements Analysis Document (RAD)

I test case pianificati sono strettamente correlati ai requisiti funzionali e non funzionali definiti nel RAD. Ogni requisito identificato nel RAD è stato analizzato per creare casi di test che ne verificano la corretta implementazione e conformità.

2.1.2 Relazioni con il System Design Document (SDD)

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

3 Panoramica del sistema

3.1 Descrizione del sistema

Come analizzato e deciso nel System Design Document, il sistema *ZeroWaste Home* adotta il pattern integrato in un'architettura Three-Tier per garantire modularità, scalabilità e separazione delle responsabilità.

- **Presentation Layer (Livello di Presentazione)**: Gestisce l'interazione con l'utente tramite un'interfaccia user-friendly sviluppata in **Angular**
- **Logic Layer (Control):** Implementa la logica di business tramite il framework Spring Boot.
- **Data Layer:** Gestisce i dati persistenti tramite **MySQL** per l'archiviazione e il recupero sicuro.

4 Features da testare/da non testare

4.1 Funzionalità incluse ed escluse

Le seguenti funzionalità sono state selezionate come prioritarie per i test, poiché rivestono un ruolo cruciale per il funzionamento e il rilascio iniziale del sistema:

- **Autenticazione:** Autenticazione;
- **Gestione Prodotti:** Visualizza Prodotti Dispenza, Aggiungi Prodotti Frigo, Ricerca Prodotti per Nome;
- **Gestione della Community:** Condivisione Ricette;
- **Gestione Lista della Spesa:** Generazione lista della spesa;
- **Gestione Amministrativa:** Risoluzione dei Problemi di Pagamento, Risoluzione delle Segnalazioni Community.

Le funzionalità che non verranno testate sono le seguenti:

- **Registrazione:** Esclusa per **bassa priorità**: funzionalità già consolidata e verificata in altri contesti;
- **Gestione Profilo:** Include il recupero password e la modifica dei dati personali. Esclusa per essere testata in una **fase successiva** del progetto;
- **Gestione Prodotti:** Funzionalità come la modifica ed eliminazione dei prodotti sono escluse per **limitazioni di tempo** e saranno testate in futuro;
- **Gestione della Community:** La modifica delle ricette è stata esclusa in quanto non critica per il rilascio iniziale del sistema;

- **Gestione Lista della Spesa:** La modifica della lista della spesa non verrà testata, essendo considerata una funzionalità di **priorità bassa**;
- **Gestione Abbonamenti e Pagamenti:** Gestione del piano abbonamenti esclusa per essere testata durante **iterazioni future**;
- **Gestione Impatto Ambientale:** Funzionalità come il calcolo e la visualizzazione dell'impatto ambientale non saranno testate, poiché rientrano in una fase avanzata del progetto.

5 Pass/ Fail criteria

5.1 Criteri di accettazione e rifiuto

Il test si considera **superato** se l'output ottenuto corrisponde all'output atteso, cioè l'oracolo.

Il test si considera **non superato** se l'output ottenuto non corrisponde all'oracolo.

La fase di testing avrà successo se individuerà una failure. In tal caso questa verrà analizzata e, se legata ad un fault, si procederà alla sua correzione. Sarà infine iterata la fase di testing per verificare che la modifica non abbia impattato su altri componenti del sistema. La failure quindi è uno stato di condizione nel quale non si trova l'output desiderato.

6 Approccio

6.1 Testing d'Unità

Il **testing d'unità** rappresenta la fase iniziale del processo di testing, con l'obiettivo di verificare che le singole unità di codice, come moduli, metodi o classi, funzionino correttamente in isolamento. Ogni unità viene testata separatamente per garantire che risponda correttamente agli input definiti e produca gli output attesi, in conformità ai requisiti specificati.

In questa fase, verrà utilizzato l'approccio **Category Partition** per creare casi di test strutturati e coprire una vasta gamma di scenari significativi. Questo metodo aiuta a suddividere gli input di un'unità in **categorie significative** e a identificare combinazioni di test rilevanti, riducendo il numero di test ridondanti e aumentando l'efficacia del processo.

L'approccio Category Partition verrà applicato come segue:

1. **Suddivisione degli Input in Categorie:**
    - Gli input delle unità saranno analizzati e suddivisi in gruppi basati sulle loro caratteristiche (ad esempio, valori validi, edge cases e valori non validi),
    - Ogni categoria rappresenterà un insieme di condizioni da testare.
2. **Definizione di Valori Rappresentativi:**

- Per ciascuna categoria, saranno selezionati valori specifici per rappresentare scenari chiave,
- Saranno considerati valori standard, casi limite e valori non validi.

3. **Generazione di Combinazioni di Test:**
   - Verranno creati test case combinando i valori rappresentativi delle diverse categorie, concentrandosi sulle combinazioni più significativi.

6.2 Testing d'integrazione

Il **testing d'integrazione** si concentra sulla verifica delle interazioni tra le componenti del sistema, assicurandosi che lavorino insieme come previsto. Sebbene non applicabile in questa fase iniziale, sarà affrontato nelle fasi successive dello sviluppo, quando saranno disponibili componenti sufficientemente stabili per essere integrate.

Gli obiettivi principali del testing d'integrazione includono:

- Verificare che le componenti comunichino correttamente tra loro attraverso le API e gli altri punti di integrazione,
- Garantire la coerenza dei dati tra i livelli di sistema (ad esempio, tra il Logic Layer e il Data Layer),
- Identificare e correggere eventuali problemi derivanti da dipendenze tra moduli.

Saranno previsti casi di test specifici per ogni punto di integrazione critico, come:

- Interazioni tra l'interfaccia utente sviluppata in **Angular** e i servizi backend forniti da **Spring Boot**,
- Trasferimento e validazione dei dati tra il **Logic Layer** e il **Data Layer**, assicurandosi che i dati vengano salvati e recuperati correttamente tramite **MySQL**.

6.3 Testing di Sistema

Il testing di sistema mira a verificare il comportamento del sistema nel suo insieme, valutandone la conformità ai requisiti funzionali e non funzionali. Tuttavia, in questa fase iniziale, il focus rimane sul testing delle singole unità.

Quando sarà eseguito, il testing di sistema includerà:

- **Test funzionali:** Per confermare che il sistema soddisfi i requisiti descritti nel RAD,
- **Test non funzionali:** Per valutare aspetti come scalabilità, usabilità e sicurezza.

Verranno simulate condizioni operative realistiche per testare scenari end-to-end, come:

- Registrazione di un utente, aggiunta di prodotti al frigo e suggerimenti di ricette basate sugli ingredienti disponibili,
- Generazione di notifiche per prodotti in scadenza e la loro gestione tramite il backend.

6.4 Performance Testing

Il performance testing valuta come il sistema si comporta sotto carico, misurando aspetti come velocità, reattività e stabilità. Sebbene non applicabile in questa fase, sarà eseguito in fasi avanzate per garantire che il sistema soddisfi i requisiti di performance.

Durante questa fase, saranno eseguiti:

- **Stress Test**: Per determinare il punto di rottura del sistema,
- **Load Test:** Per misurare il comportamento del sistema sotto carichi progressivi di utenti simultanei,
- **Test di capacità:** Per verificare il numero massimo di richieste gestibili dal backend.

Saranno utilizzati strumenti come Apache JMeter o Gatling per simulare utenti concorrenti e misurare metriche chiave come il tempo di risposta e l'uso delle risorse.

6.5 Pilot Testing

Il pilot testing prevede il coinvolgimento di un gruppo selezionato di utenti finali per testare il sistema in un ambiente reale, al fine di raccogliere feedback e identificare potenziali problemi prima del rilascio definitivo.

In questa fase iniziale, il pilot testing non sarà applicato, ma nelle fasi successive includerà:

- **Esecuzione di scenari reali**: Gli utenti testeranno funzionalità principali come la gestione della dispensa, le notifiche e la generazione della lista della spesa,
- **Raccolta di feedback qualitativo:** Per valutare l'usabilità dell'interfaccia e l'efficacia delle funzionalità implementate.

I risultati saranno utilizzati per apportare miglioramenti sia tecnici che di design.

6.6 Acceptance Testing

Il testing di accettazione simula il comportamento del cliente per verificare che il sistema soddisfi tutti i requisiti concordati e sia pronto per il rilascio.

Le attività pianificate per il testing di accettazione includono:

- **Simulazioni guidate:** Gli stakeholder simuleranno scenari operativi reali per confermare che il sistema soddisfi i requisiti funzionali;

- **Validazione dei requisiti critici:** Ogni requisito identificato come critico nel RAD sarà verificato attraverso casi di test specifici;
- **Test su casi limite:** Saranno inclusi test per situazioni insolite o estreme, come input non validi o carichi elevati, per garantire che il sistema gestisca adeguatamente queste condizioni.

Il successo del testing di accettazione sarà determinato dal superamento di tutti i casi di test pianificati.

6.7 Installation Testing

Il **testing dell'installazione** verifica che il sistema possa essere installato correttamente negli ambienti target, garantendo una configurazione semplice e funzionale.

Le attività di questa fase includeranno:

1. **Verifica della configurazione server:** Assicurarsi che **Spring Boot** e **MySQL** siano installati e configurati correttamente;
2. **Installazione client-side:** Controllare che l'applicazione **Angular** sia facilmente accessibile tramite browser e sia compatibile con i sistemi operativi supportati;
3. **Testing delle dipendenze:** Verificare che tutte le librerie richieste siano disponibili e funzionino come previsto;
4. **Rollback e recovery:** Garantire che in caso di errore durante l'installazione, sia possibile ripristinare lo stato precedente senza perdita di dati o funzionalità.

Il testing sarà condotto utilizzando una checklist dettagliata per assicurare che tutte le configurazioni siano state implementate correttamente.

7 Sospensione e ripristino

7.1 Criteri di sospensione

Il testing verrà **sospeso** se almeno il 10% dei test cases risulta essere testato con successo: se è stato rilevata la presenza di almeno un fault. In caso di sospensione, il team dovrà correggere tutti i fault rilevati prima di procedere con l'implementazione di nuove funzionalità.

7.2 Criteri di ripristino

Il testing verrà **ripreso** una volta effettuati tutti i cambiamenti necessari per la migrazione dei fault individuati e dopo aver ricontrollato che non ce ne siano altri. Bisogna assicurarsi anche che i cambiamenti apportati non abbiano loro stessi introdotto nuovi fault, i quali avranno impatto sulle componenti già esistenti: in questi casi si testeranno anche quest'ultime componenti tramite testing di regressione.

La ripresa avverrà soltanto quando saranno risolti i problemi che ne hanno determinato la sospensione e riprenderà dal test case che ne ha causato la sospensione.

L'attività di testing risulta essere **terminata** quando la totalità dei test cases risulta avere esito negativo, sarà necessario raggiungere un **branch coverage** pari almeno al 60%.

8 Test Deliverables

8.1 Risultati e Report del Test

Il processo di testing produrrà una serie di deliverable documentati per garantire una visione chiara ed esaustiva delle attività svolte, dei risultati ottenuti e delle raccomandazioni finali.

La **Test Case Specification** includerà un elenco dettagliato dei test case, ognuno dei quali sarà descritto con:

- ID univoco per il tracciamento,
- Obiettivo chiaro del test,
- Condizioni iniziali necessarie per l'esecuzione,
- Input attesi e output previsti, basati sui requisiti,
- Risultato effettivo osservato durante il test,
- Stato finale (Pass/Fail) per indicare se il test è stato superato o meno.

Saranno inclusi test funzionali per le funzionalità principali, edge cases per situazioni limite, e test per verificare la robustezza del sistema in presenza di errori. La priorità di ogni test case sarà specificata per garantire che le funzionalità critiche siano affrontate per prime.

Il **Test Execution Report** fornirà una panoramica completa dei risultati ottenuti durante il testing. Questo includerà la percentuale di test superati rispetto al totale, il numero di bug identificati e risolti, e la copertura del codice misurata attraverso metriche come Line Coverage, Branch Coverage e Function Coverage. Sarà indicato il tempo totale di esecuzione dei test, insieme a un elenco dei test non superati, con una descrizione delle potenziali cause di failure e delle azioni correttive proposte. Inoltre, saranno incluse raccomandazioni finali per migliorare il sistema e prepararlo al rilascio.

La documentazione sarà completata da grafici e tabelle che sintetizzano i dati chiave, rendendo il report facilmente interpretabile da tutti gli stakeholder. Questi deliverable saranno essenziali per monitorare l'efficacia del processo di testing e per garantire che il sistema soddisfi i requisiti di qualità stabiliti.

9 Strumenti per il testing

9.1 Mojito - Testing di Unità

Mojito è uno strumento dedicato al testing di unità, utilizzato per verificare il corretto funzionamento delle singole componenti del sistema in isolamento. Le principali caratteristiche di Mojito includono:

- **Facilità di utilizzo:** Consente di scrivere casi di test in modo intuitivo, grazie a un'API chiara e ben documentata;
- **Supporto per mocking:** Permette di simulare comportamenti di dipendenze esterne, riducendo la complessità dei test;
- **Integrazione con framework Java:** Progettato per funzionare senza problemi con Spring Boot, garantendo un testing fluido della logica di business;
- **Gestione automatizzata:** Genera report dettagliati sullo stato dei test, evidenziando i moduli che richiedono ulteriori correzioni.

Utilizzando Mojito, il team può individuare bug nelle fasi iniziali dello sviluppo, riducendo i costi e i rischi associati al rilascio di codice non conforme.

9.2 JaCoCo - Code Coverage

JaCoCo è uno strumento avanzato per il calcolo della copertura del codice, utilizzato per valutare l'efficacia dei test scritti. JaCoCo analizza il codice eseguito durante i test e produce metriche dettagliate, tra cui:

- **Line Coverage:** Percentuale di righe di codice eseguite,
- **Branch Coverage:** Percentuale di ramificazioni condizionali esplorate,
- **Complexity Analysis:** Valutazione della complessità ciclomatica per identificare punti critici del codice.

Principali vantaggi:

- **Report dettagliati:** Genera report visivi e intuitivi in formati standard, come HTML o XML, per facilitare l'interpretazione dei dati;
- **Integrazione con build tools:** Compatibile con **Maven** e **Gradle**, permettendo una facile integrazione nei processi di build automatizzati;
- **Monitoraggio continuo:** Consente di verificare la copertura del codice in tempo reale durante le pipeline CI/CD.