Università degli Studi di Salerno
Corso di Ingegneria del Software

SwaGGed
Test Plan Document
Versione 2.0

Data: 14/01/2025

Coordinatore del progetto

Nome	Matricola

Partecipanti:

Nome	Matricola
Choaib Goumri	0512118390
Mattia Gallucci	0512116893

Scritto da:	Choaib Goumri
	Mattia Gallucci

Revision History

Data	Versione	Descrizione	Autore
13/12/2024	1.0	Definizione del test plan	Choaib Goumri, Gallucci Mattia
14/01/2025	2.0	Finalizzazione del documento	Choaib Goumri, Gallucci Mattia

Sommario

1. INTRODUZIONE 4
2. RELAZIONE CON ALTRI DOCUMENTI 4
3. PANORAMICA DEL SISTEMA 5
4. FUNZIONALITÀ DA TESTARE 6
4.1 Funzionalità da testare: 6
5 PASS/FAIL CRITERIA 7
6 APPROCCIO 8
7 SOSPENSIONE E RIPRESA 10
8 TESTING MATERIALS 11
9 TEST CASES 12
10 TESTING SCHEDULE 13

1. INTRODUZIONE

Il presente documento di Piano di Test descrive le strategie, le metodologie e le risorse necessarie per condurre il processo di verifica del sistema SwaGGed. L'obiettivo principale è assicurare che il software soddisfi i requisiti delineati nel documento

- Requirements_Analysis_Document_SwaGGed.

Questo piano ha lo scopo di individuare eventuali anomalie, malfunzionamenti o difetti all'interno del sistema attraverso l'esecuzione di test unitari, di integrazione e complessivi, garantendo così un prodotto finale stabile e conforme alle aspettative del cliente.

Il sistema SwaGGed, sviluppato per gestire contenuti, commenti, community e profili utente, è organizzato in sottosistemi distinti che richiedono attività di test per verificare la funzionalità e l'efficienza complessiva. I test saranno condotti adottando un approccio bottom-up e utilizzando tecniche di black-box, al fine di verificare sia il funzionamento dei singoli componenti sia le loro interazioni.

2. RELAZIONE CON ALTRI DOCUMENTI

- **Relazioni con il Requirements Analysis Document (RAD)**
  I test case definiti nel Test Plan sono sviluppati in base ai requisiti funzionali e non funzionali descritti nel RAD.

- **Relazioni con il System Design Document (SDD)**
  I test case previsti nel Test Plan devono rispettare la suddivisione del sistema in sottosistemi così come illustrato nell'SDD.

- **Relazioni con l'Object Design Document (ODD)**
  Per quanto riguarda il test di unità e di integrazione, strettamente connessi all'ODD e alla suddivisione in package del sistema, tali test saranno documentati esclusivamente all'interno del codice dell'applicativo. Di conseguenza, nel presente documento non saranno inclusi riferimenti al loro design

3. PANORAMICA DEL SISTEMA

L'architettura prevista, come descritta nel documento System Design Document (SDD), adotta un modello a tre livelli (three-tier), separando chiaramente la presentazione, la logica applicativa e la gestione dei dati. Questa organizzazione migliora la modularità e la facilità di manutenzione del sistema, semplificando il processo di testing e aggiornamento.

Il livello di Presentazione del sistema SwaGGed è destinato all'interazione con l'utente finale. Grazie a un'interfaccia grafica responsive sviluppata con tecnologie come JSP, HTML5, CSS e JavaScript, le pagine JSP, denominate secondo la convenzione camelCase, sono progettate per interagire con il livello applicativo e gestire in modo dinamico i dati visualizzati.

Il livello applicativo (Application Layer) costituisce il nucleo della logica di business del sistema. Questo strato funge da intermediario tra il livello di presentazione (Presentation Layer) e quello di gestione dei dati (Data Layer), processando le richieste degli utenti e orchestrando le operazioni necessarie.

Il livello di gestione dei dati (Data Layer), invece, è responsabile della persistenza delle informazioni. Questo strato utilizza un database relazionale MySQL per memorizzare in modo organizzato i dati relativi alle diverse entità definite nel sistema.

Le operazioni di lettura e scrittura sui dati sono gestite tramite classi DAO (Data Access Object), progettate per interfacciarsi direttamente con il database, assicurando un corretto salvataggio e recupero delle informazioni. Il livello di presentazione (Presentation Layer) inoltra le richieste al livello applicativo (Application Layer), il quale le elabora e comunica con il livello dati (Data Layer) per effettuare le operazioni necessarie di accesso o memorizzazione.

4. FUNZIONALITÀ DA TESTARE

Per ragioni di costi e tempi, il comportamento del sistema è stato testato esclusivamente in base ai casi di test definiti per le seguenti funzionalità:

4.1 Funzionalità da testare:

- Funzionalità Utente
  - Autenticazione Utente
  - Registrazione Utente
  - Ricerca Utente
- Funzionalità Gestore Post
  - Aggiunta Post
  - Eliminazione Post
  - Apprezzamento Post
  - Ricerca Post
- Funzionalità Gestore Commenti
  - Aggiunta Commento
  - Eliminazione Commento
- Funzionalità Gestore Community
  - Aggiunta Community
  - Eliminazione Community
  - Iscrizione alla Community
  - Ricerca Community

5 PASS/FAIL CRITERIA

Il processo di testing del sistema SwaGGed ha l'obiettivo principale di verificare la piena conformità del sistema ai requisiti funzionali e non funzionali definiti. L'attività di test si concentra sull'identificare eventuali anomalie o difformità, assicurando che ogni funzionalità implementata operi correttamente e generi risultati coerenti con i dati di input forniti.

L'approccio adottato prevede un confronto sistematico tra i risultati effettivi e quelli attesi, con i seguenti possibili esiti:

- **Esito positivo (PASS):** il test è considerato superato se vengono individuati comportamenti non conformi rispetto alle specifiche, poiché ciò indica che il processo di verifica è stato efficace nell'identificazione delle problematiche.

- **Esito negativo (FAIL):** il test fallisce qualora il comportamento del sistema corrisponda completamente a quello previsto, segnalando l'assenza di errori rilevati nel contesto analizzato.

Gli errori vengono identificati tramite un "oracolo", ossia un riferimento che definisce il comportamento ideale del sistema. In caso di rilevamento di anomalie, il processo prevede la correzione del codice sorgente, la ripetizione dei test e una verifica approfondita per garantire che tutte le discrepanze siano risolte e che il sistema soddisfi i requisiti.

6 APPROCCIO

L'approccio al testing di SwaGGed è strutturato in tre fasi principali, ognuna pensata per garantire che il sistema rispetti i requisiti previsti e funzioni correttamente in tutti gli aspetti.

1. **Test a livello di singole unità**
   La prima fase è dedicata al controllo delle singole classi e dei metodi del sistema. Utilizzando una tecnica di testing black-box, ci si concentra esclusivamente sugli input e sugli output delle componenti, senza esaminare la loro struttura interna. Per affrontare la complessità degli input possibili, si applica una metodologia che suddivide i dati in categorie, trattando ogni categoria come un caso di test distinto. Questo approccio permette di testare ogni classe in modo completo ed efficiente, riducendo il rischio di test ridondanti. Eventuali anomalie individuate durante questa fase vengono corrette e ritestate, per garantire che ogni componente rispetti le specifiche richieste.

2. **Test di integrazione**
   La seconda fase si concentra sul funzionamento del sistema nel suo complesso, verificando come le diverse componenti interagiscono tra loro. L'obiettivo è assicurarsi che tutti i moduli del sistema, funzionino correttamente quando operano insieme. Anche qui, vengono utilizzati test basati su combinazioni significative di input, ottimizzando così il numero di casi di test da eseguire. In questa fase vengono verificati i flussi di dati tra i moduli e il loro corretto interscambio, al fine di evitare errori nelle comunicazioni interne del sistema.

3. **Test complessivo del sistema**
   L'ultima fase prevede la valutazione dell'intero sistema, simulando l'uso reale da parte degli utenti. Oltre a testare le funzionalità principali, vengono esaminati anche gli aspetti non funzionali, come la performance e la sicurezza. Questo tipo di test ha lo scopo di verificare che il sistema rispetti pienamente tutti i requisiti definiti nei documenti di progettazione (RAD, SDD, ODD). Il sistema viene testato in un ambiente che replica quello di produzione, per identificare e risolvere eventuali problemi prima del rilascio finale.

7 SOSPENSIONE E RIPRESA

Il testing del sistema SwaGGed verrà interrotto nel caso si verifichino situazioni che impediscano di proseguire con le attività. Le principali cause di sospensione includono il rilevamento di errori critici o bloccanti, come malfunzionamenti che compromettono le funzionalità essenziali, ad esempio guasti del sistema o anomalie nel database che impediscono l'esecuzione dei test pianificati. Inoltre, la sospensione avverrà se ci sono modifiche al sistema che non sono state integrate correttamente, come aggiornamenti o modifiche al codice che non sono stati adeguatamente testati o documentati.

Per riprendere i test, sarà necessario risolvere gli errori bloccanti identificati. Una volta corretti i problemi, si procederà con test di regressione mirati per garantire che le modifiche non abbiano introdotto nuovi malfunzionamenti.

8 TESTING MATERIALS

Per garantire che il sistema SwaGGed soddisfi tutti i requisiti funzionali e non funzionali, saranno utilizzati strumenti software specializzati che ottimizzano il processo di testing. Di seguito vengono descritti gli strumenti principali che verranno adottati:

Selenium

Selenium sarà utilizzato per l'automazione dei test sull'interfaccia utente (UI). Questo strumento consentirà di simulare le azioni dell'utente finale, come il login, la registrazione, aggiunta dui post. Attraverso l'automazione, si potrà verificare l'affidabilità e la correttezza delle interazioni tra l'utente e il sistema, permettendo di eseguire test ripetibili su vari browser e dispositivi.

Selenium è particolarmente utile per rilevare anomalie nell'interfaccia grafica, garantendo che l'esperienza utente rimanga coerente e priva di errori.

JUnit

JUnit sarà impiegato come framework principale per i test unitari e di integrazione. Con JUnit, verranno testate le singole classi, come i metodi delle classi DAO, i controller e la logica di business. I test unitari garantiranno che ciascun componente funzioni in modo corretto e isolato, mentre i test di integrazione verificheranno la corretta interazione tra le diverse parti del sistema.

Grazie a JUnit, sarà possibile generare report dettagliati che consentiranno di monitorare i risultati dei test e identificare tempestivamente discrepanze o errori, assicurando che tutte le funzionalità siano implementate

Mockito

Mockito sarà impiegato per eseguire i test unitari e di integrazione del sistema. Grazie a questo framework, sarà possibile verificare la correttezza delle interazioni tra le diverse componenti, come ad esempio quelle tra le classi DAO e il database o tra i controller e i servizi di business.

L'utilizzo di oggetti mock consentirà di simulare il comportamento delle dipendenze esterne, facilitando l'individuazione e l'analisi di eventuali errori nei moduli integrati, oltre a garantire un isolamento efficace durante le fasi di test.

9 TEST CASES

Questa sezione costituisce il cuore del piano di test, elencando tutti i casi di test impiegati nel processo di validazione. Ogni caso di test viene descritto in modo approfondito in un documento separato chiamato **Test Case Specification**. Inoltre, per ogni esecuzione dei test, verrà redatto un **Test Incident Report** che registrerà i risultati e segnalerà eventuali anomalie o problemi riscontrati. I dettagli relativi alla gestione di questi documenti saranno forniti in modo più esaustivo nelle sezioni successive.

10 TESTING SCHEDULE

Il piano di testing per il sistema SwaGGed è stato sviluppato per garantire un processo di verifica completo, ben strutturato e sincronizzato con il ciclo di sviluppo. Il testing si articolerà in tre fasi principali, ciascuna focalizzata su un aspetto specifico del sistema.

- **Test Unitari**: Durante questa fase, che durerà una settimana, verranno testate le singole classi e i metodi tramite JUnit e Mockito per verificarne il corretto funzionamento in conformità ai requisiti. I test saranno eseguiti parallelamente allo sviluppo del codice, in modo da ottimizzare i tempi e risolvere tempestivamente eventuali problemi a livello di singoli componenti.

- **Test di Integrazione**: Nella seconda settimana, sarà testato l'intero flusso di interazione tra i moduli del sistema, comprese le comunicazioni tra i DAO e il database tramite il DataSource, nonché le interazioni tra le classi di gestione e le JSP.

  L'obiettivo di questa fase è verificare che tutte le dipendenze tra i vari componenti siano correttamente gestite e che i dati vengano scambiati senza errori tra le diverse parti del sistema.

- **Test di Sistema**: Nell'ultima settimana, il sistema sarà testato nel suo complesso. Verranno eseguiti test sulle funzionalità generali e sulle performance del sistema.

  Utilizzando Selenium, saranno simulati scenari d'uso reali, come la registrazione degli utenti, la prenotazione dei posti e la gestione delle programmazioni, per garantire che tutte le funzionalità siano operative e performanti.

Alla fine dello sviluppo, tutti i test verranno rieseguiti per una verifica finale, assicurando che il sistema funzioni correttamente. Ogni fase di test sarà seguita da una revisione dettagliata dei risultati per correggere eventuali anomalie prima di passare alla fase successiva, in modo da garantire un rilascio privo di bug critici.