Università degli Studi di Salerno
Corso di Ingegneria del Software

ResolveIT
Test Plan
Versione 1.2

Data: 18/01/2026

Partecipanti:

Nome	Matricola
Samuele Nacchia	0512119128
Andrea Generale	0512119134
Mario Di Feo	0512119320

Scritto da:	MDF, AG, SN

Revision History

Data	Versione	Descrizione	Autore
18/12/2025	1	Prima stesura	SN, AG, MDF
20/12/2025	1.1	Completamento globale, revisione e testing schedule	SN, AG
18/01/2026	1.2	Revisione finale	SN, AG, MDF

Indice

1. Introduction

ResolveIT si propone di fornire un sistema di assistenza clienti personalizzabile per più piattaforme. Questo documento ha l'obiettivo di descrivere ed analizzare le attività di Testing. Il fine è quello di garantire che ogni aspetto funzioni in modo corretto. All'interno del documento sono riportate le strategie di testing adottate, quali funzionalità saranno testate e gli strumenti scelti per la rilevazione degli errori, con lo scopo di presentare al cliente finale una piattaforma priva di malfunzionamenti.

Sono state pianificate attività di testing per le seguenti gestioni:

- Gestione Utenti
- Gestione Ticket
- Gestione Categorie
- Gestione Account

2. Relationship to other documents

Per la corretta individuazione dei test case, si fa riferimento ad altri documenti prodotti. I test case pianificati nel Test Plan sono elaborati in relazione ai requisiti funzionali e non funzionali presentati nel Requirements Analysis Document (RAD). I test case pianificati nel Test Plan devono rispettare la suddivisione in sottosistemi presentata nel System Design Document (SDD). Per ciò che concerne il test di unità e di integrazione, maggiormente legati all'Object Design Document (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. System overview

Il sistema proposto basa la sua architettura sul sistema three-tier, in particolare usando Spring MVC. Verranno usati HTML5, CSS3 e Thymeleaf per la parte di front-end e la generazione delle view. Per la logica applicativa e quindi il back-end sarà utilizzato Java SPRING.
Per la gestione del database saranno usati:

- Spring JPA per il collegamento al database.
- MySQL come database in fase di produzione e deployment.

4. Features to be tested/not to be tested

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

- Gestione Utenti
    - Login
    - Registrazione Cliente
- Gestione Ticket
    - Invio Ticket
    - Assegnazione Ticket
    - Annullazione Ticket
    - Risoluzione Ticket
    - Rilascio Ticket
- Gestione Categorie
    - Creazione Categoria
    - Modifica Categoria
- Gestione Account
    - Creazione Account Operatore
    - Modifica Dati Account

Le funzionalità di cui non si andrà ad effettuare le attività di testing riguardano requisiti funzionali non essenziali al corretto funzionamento del sistema.
Sono esclusi quindi servizi come la verifica della registrazione, il cambio password ed il servizio di messaggistica.

5. Pass/Fail criteria

Le attività di testing sono mirate ad identificare la presenza di faults all'interno del sistema, per effettuare un successivo intervento di eliminazione.
L'esito di un test case è valutato mediante un oracolo, inteso come il risultato atteso della sua esecuzione, basandosi sui requisiti.
Un test ha successo (pass) se, dato un input al sistema, l'output ottenuto è uguale all'output atteso dall'oracolo.
Un test fallisce (fail) se, dato un input al sistema, l'output ottenuto è diverso dall'output atteso dall'oracolo.

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

- Testare con successo 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/risolti faults trovati durante le attività di testing;
- Raggiungere un branch coverage non inferiore al 80%

6. Approach

Il testing dell'intero sistema si compone di tre fasi: testing di sistema, testing di integrazione e testing di unità. Verranno progettati nell'ordine appena definito, ma verranno eseguiti in ordine inverso. Prima della fase di implementazione del sistema, avverrà la progettazione dei casi di test di sistema, perfezionati in seguito nella loro fase di esecuzione; durante la fase implementativa avverrà la progettazione dei casi di test di unità.

Testing di Sistema

Per il testing di sistema sarà utilizzato il tool Selenium IDE, che permette di registrare le azioni che un l'utente può intraprendere sul browser, in modo da poter implementare ed eseguire i test case di sistema. Il server, per la fase di testing, verrà eseguito in locale.

Functional testing

Il functional testing ha il fine di validare i requisiti funzionali. Consiste nell'individuare i possibili faults generati dagli input degli utenti.

Acceptance Testing

L'acceptance testing verrà fatto simulando la figura del cliente e basandosi sui criteri di accettazione definiti nei requisiti funzionali, agendo come validazione finale.

Testing di integrazione

In linea con il paradigma a oggetti, si adotterà un approccio Bottom-up e l'esecuzione dei test procederà dai layer più interni a quelli più esterni:

- **Layer Model & Service:** Si procederà inizialmente con la validazione della business logic e della persistenza dei dati.
- **Layer Controller:** Una volta stabilizzati i servizi, verranno testate le Servlet (Spring Controllers) per verificare la corretta gestione delle richieste HTTP e il flusso di navigazione. Per isolare i test dei Controller dai layer inferiori qualora necessario, si valuterà l'uso di librerie di Mocking (Mockito) per simulare il comportamento dei Service.

La definizione dei test case avverrà tramite il tool JUnit.
L'automatizzazione del run dei test sarà gestita da Maven.
Il test di integrazione sarà il medesimo per tutte le componenti da testare.
In generale, il test di integrazione sarà compreso nella stessa classe di test di quelli unitari.

Testing di Unità

La strategia di testing di unità prevede la verifica puntuale di ogni metodo delle classi del sistema, ad eccezione delle interfacce e delle classi Entity, la cui logica è considerata banale.

La definizione dei casi di test seguirà un approccio ibrido: sebbene la progettazione iniziale sia guidata dai requisiti funzionali (**Black-box**), verrà adottata parallelamente una strategia **White-box** per analizzare la struttura interna del codice. Tale approccio complementare ha l'obiettivo di massimizzare la **Statement e Branch Coverage**, assicurando che tutti i percorsi logici e le clausole di gestione delle eccezioni siano opportunamente verificati.

I test saranno implementati utilizzando il framework Java **JUnit** e sviluppati parallelamente alle classi di produzione per facilitare la copertura. Ad ogni classe corrisponderà una Test Class secondo la convenzione di nomenclatura NomeClassTest. Un altra tecnologia usata in tale fase sarà JaCoCo per il calcolo di metriche tra le quali la Branch Coverage.

7. Suspension and resumption

In questa sezione verranno specificati i criteri di sospensione del test e le attività di test che dovranno essere ripetute quando si riprende il test.

Criteri di sospensione
Il testing verrà sospeso nei seguenti casi di blocco infrastrutturale:

- **Corruzione degli script di test:** se i file JUnit o Selenium presentano errori sintattici o logici che rendono i risultati non deterministici.
- **Indisponibilità dell'ambiente:** mancata raggiungibilità del database o del server Tomcat locale.
- **Failure Critica di Build:** se Maven fallisce la compilazione del progetto, rendendo impossibile l'esecuzione di qualsiasi test.

Criteri di ripristino
Al ripristino, verranno ripetuti i test falliti e quelli delle componenti correlate.

8. Testing materials

L'hardware necessario per l'attività di test è un computer, anche senza connessione ad internet.

9. Test Cases

L'approccio per la definizione dei test frame sarà il category partition.
Per definire l'output atteso si userà un'analisi manuale dei risultati a fronte dei requisiti, per via dell'assenza di specifiche formali/semi-formali.
I test case verranno specificati nella documentazione apposita.

10. Testing schedule

La pianificazione dei test segue il modello **incrementale** adottato per lo sviluppo della piattaforma. Le attività di testing inizieranno nella fase di analisi e si concluderanno con la validazione finale prima del rilascio. Indicativamente le fasi saranno:

- **Fase 1 :** Definizione della strategia e progettazione dei test case di Sistema. Identificazione delle categorie per i test critici.
- **Fase 2 :** Esecuzione dei test unitari e di integrazione sul layer Model e Service non appena i POJO e i Repository vengono implementati.
- **Fase 3 :** Test di integrazione dei Controller (Servlet) e verifica della logica di navigazione.
- **Fase 4 :** Esecuzione dei test di sistema automatizzati con Selenium IDE per validare i requisiti funzionali e l'interfaccia web.
- **Fase 5 :** Stesura del Test Summary Report.