Università degli Studi di Salerno
Corso di Ingegneria del Software

EduCat
Test Plan Document
Versione 1.0

Data: 21/12/2025

Coordinatore del progetto:

Nome	Matricola
Isabella Dima	0512119719

Partecipanti:

Nome	Matricola
Isabella Dima	0512119719
Anna Chiara Esposito	0512119143

Scritto da:	Anna Chiara Esposito

Revision History

Data	Versione	Descrizione	Autore
21/12/2025	1.0	Test Plan Document	Tutto il team

Indice

1. Introduction 4
2. Relationship to other documents 4
3. System overview 4
4. Features to be tested/not to be tested 4
5. Pass/Fail criteria 5
6. Approach 5
6.1 Testing di Unità (White-Box) 5
6.2 Testing di Integrazione 5
6.3 Testing di Sistema (Black-Box) 6
7. Suspension and resumption 6
8. Testing materials (hardware/software requirements) 6
9. Test cases 6
9.1 Testing di Sistema (Black-Box) 6
9.2 Testing di Sistema (Black-Box) 7
10. Testing schedule 8

1. Introduction

Il presente documento descrive la strategia di testing pianificata per la piattaforma EduCat. L'obiettivo è definire un quadro di riferimento per la verifica e la validazione del software, assicurando che soddisfi i requisiti funzionali e non funzionali specificati.
Il testing mira a rilevare difetti nel codice e nella logica di business, garantendo che le funzionalità critiche come la prenotazione delle lezioni e la gestione dei pagamenti operino correttamente prima del rilascio finale previsto per gennaio 2026.

2. Relationship to other documents

Questo piano di test è strettamente correlato alla documentazione di progetto prodotta nelle fasi precedenti.

- **Requirements Analysis Document (RAD):** I test funzionali (System Testing) sono progettati per verificare gli scenari e i requisiti funzionali descritti nel RAD. Ogni *Test Case* funzionale è tracciabile verso uno specifico requisito (es. Acquisto Lezione).
- **System Design Document (SDD):** L'architettura del sistema (MVC, Java, MySQL) definita nel SDD guida la strategia di *Integration Testing* e la configurazione dell'ambiente di test.
- **Object Design Document (ODD):** La struttura delle classi e dei pacchetti definita nell'ODD costituisce la base per il *Unit Testing*.

3. System overview

Il sistema EduCat è basato su un'architettura Three-Tier (a tre livelli) che implementa il design pattern MVC (Model-View-Controller) su infrastruttura Client/Server. I componenti principali oggetto di test sono:

1. **Gestione Utenza:** Moduli per registrazione, login e gestione profili (Studenti, Genitori, Tutor).
2. **Gestione Lezione:** La parte centrale del sistema, che include ricerca, prenotazione, pagamento e gestione disponibilità.
3. **Recensione:** Sistema di feedback post-lezione.
4. **Segnalazione:** Modulo di moderazione per utenti con comportamenti scorretti.
5. **Persistenza:** Interazione con il database MySQL tramite pattern DAO.

4. Features to be tested/not to be tested

Le seguenti funzionalità, derivate dalla scomposizione dei sottosistemi, saranno oggetto di test:

Sottosistema	Funzionalità	Priorità
Gestione Utenza	Login, Registrazione (con distinzione ruoli), Modifica Profilo.	Alta

Gestione Lezione	Ricerca Lezioni, Prenotazione (check disponibilità), Acquisto, Annullamento.	Alta
Recensioni	Inserimento Recensione, Visualizzazione media voti.	Media
Segnalazioni	Invio segnalazione, Moderazione (Ban utente).	Bassa

Tutte le funzionalità con priorità medio/bassa non verranno testate, nello specifico anche:

- Gateway di Pagamento Reale: L'interazione con i circuiti bancari reali non sarà testata; verrà utilizzato un mock per simulare l'esito del pagamento.
- Servizio Email esterno: L'invio effettivo delle email sarà simulato tramite log di sistema o mock object per evitare spam durante il testing.

5. Pass/Fail criteria

I criteri per determinare il successo o il fallimento di un test sono i seguenti:

- **Pass** (Successo): Il risultato ottenuto (Output reale) coincide con il risultato atteso (Oracolo). Inoltre, non vengono sollevate eccezioni non gestite e lo stato del database riflette le modifiche previste.
- **Fail** (Fallimento): Il risultato ottenuto differisce da quello atteso, il sistema va in crash, o viene mostrata una pagina di errore generica (HTTP 500) invece di un messaggio di errore controllato.

6. Approach

La strategia di testing segue un approccio incrementale suddiviso in tre livelli:

- Testing di Unità
- Testing di Integrazione
- Testing di Sistema

6.1 Testing di Unità (White-Box)

Questa fase si concentra sulla verifica delle singole componenti software (classi), isolandole dalle loro dipendenze.

- Oggetto: Classi del livello *Application* (Manager) e *Storage* (DAO).
- Strategia: White-box testing mirato alla copertura dei rami condizionali (Branch Coverage).
- Tool: Framework JUnit 5.

6.2 Testing di Integrazione

Verifica la corretta interazione tra i sottosistemi e i livelli architetturali (Controller ←→ Manager←→ DAO).

- Strategia: Approccio Bottom-up, integrando progressivamente i componenti dai livelli più bassi (Storage) a quelli più alti (Presentation).
- Tool: Mockito per simulare le dipendenze esterne durante i test.

6.3 Testing di Sistema (Black-Box)

Valida il sistema completo simulando il comportamento dell'utente finale per accertare il soddisfacimento dei requisiti funzionali e non funzionali.

- Oggetto: Interfaccia Web e flussi utente completi (es. Prenotazione Lezione).
- Strategia: Black-box testing basato sugli scenari d'uso del RAD.
- Tool: Selenium WebDriver per l'automazione dell'interazione browser.

7. Suspension and resumption

In questa sezione verranno illustrati 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 se viene individuato un "Blocking Bug" (es. impossibilità di effettuare il login o crash del database) che impedisce l'esecuzione dei test successivi.
- Criteri di Ripresa: Il testing verrà ripreso solo quando tutti i problemi relativi alla sospensione dello stesso sono stati risolti. L'attività di testing riprenderà dal test case che ha causato la sospensione.

8. Testing materials (hardware/software requirements)

- Hardware: Workstation di sviluppo (Windows/Mac) con almeno 8GB RAM.
- Ambiente Software: Java Development Kit (JDK) 23, Apache Tomcat 10.1, MySQL Server 8.0.
- Software Testing:
  - Eclipse IDE / IntelliJ IDEA.
  - JUnit 5 (Framework di test).
  - Mockito (Framework di mocking).
  - Selenium WebDriver (Browser automation).
- Dati di test: Un database popolato con dati fittizi (Studenti, Tutor, Lezioni pre-inserite).

9. Test cases

9.1 Testing di Sistema (Black-Box)

Obiettivo: Verificare che il sistema soddisfi i Requisiti Funzionali descritti nel RAD senza preoccuparsi del codice interno. Questo livello di test si concentra sugli input (dati inseriti dallo studente) e sugli output (successo prenotazione o messaggio di errore).

- Tecnica: Equivalence Partitioning (Partizione delle classi di equivalenza) e Boundary Value Analysis (Analisi dei valori limite).
- Componenti sotto test: Interfaccia Utente (GestioneLezioneGUI) verso il Controller (PrenotazioneServlet).
- Scenari di Test (Black Box):

Test Case	Riferimento RAD	Input	Risultato Atteso
TC_LEZ_01: Prenotazione con successo (Happy Path)	Scenario “Acquisto Lezione”	Utente “Studente” autenticato, ID Lezione valida, Slot orario libero, Dati carta di credito validi (formato corretto).	Il sistema restituisce la conferma, invia l'email e lo stato della prenotazione diventa “attiva”.
TC_LEZ_02: Tentativo di prenotazione su slot già occupato	UC 2.1 “Fascia oraria già occupata”	Utente seleziona uno slot orario per cui esiste già una prenotazione nel sistema (anche se non visibile graficamente in quel momento per lag di rete).	Messaggio di errore “Hai già prenotato una lezione per questa fascia oraria” o “Slot non più disponibile”.
TC_LEZ_03: Prenotazione con dati di pagamento errati	UC 2.2 “Dati di pagamento non corretti”	Numero carta < 16 cifre, oppure CVV non numerico, o data scadenza passata.	Il sistema impedisce il checkout e mostra “Dati di pagamento non corretti”.

9.2 Testing Strutturale (White-Box)

- Obiettivo: Analisi del codice interno, in particolare la logica di “GestioneLezioneManager” definita nell'ODD, per assicurarci che tutti i percorsi (branch coverage) e le gestioni delle eccezioni siano testati.
- Tecnica: Branch Coverage e Path Testing.
- Componente sotto test: “it.unisa.educat.business.GestioneLezioneManager” e le sue interazioni con il DAO.
- Scenari di Test (White Box):

Test Case	Logica da testare
TC_LEZ_04: Verifica eccezione “SlotOccupatoException”	Il metodo “prenotaLezione” del Manager deve sollevare l’eccezione “SlotOccupatoException” quando il controllo di disponibilità ha esito negativo.
TC_LEZ_05: Rollback della transazione Database in caso di fallimento pagamento	Blocco “try-catch” all'interno del Manager. Se il pagamento lancia eccezione, verificare che venga chiamato il metodo di rollback o delete della prenotazione temporanea.
TC_LEZ_06: Gestione Concorrenza (Race Condition)	Il meccanismo di sincronizzazione nel metodo “prenotaLezione” deve permettere una sola richiesta e sollevare eccezione sull’altra se due thread chiamano il metodo nello stesso millisecondo.
TC_LEZ_07: Copertura del path “Annullamento Lezione” con rimborsi	Il metodo “annullaLezione” contiene un controllo: “if (dataLezione - dataCorrente < 24 ore)”. Passare date esattamente al limite (es. 23 ore e 59 minuti vs 24 ore e 1 minuto) per verificare che il ramo else (nessun rimborso) e if (rimborso) siano eseguiti correttamente.

10. Testing schedule

La pianificazione temporale delle attività di test segue le deadline del progetto

Fase	Attività	Responsabile	Inizio	Fine
1	Unit Testing			
2	Integration Testing			
3	System Testing	Team		
4	Correzione Bug e Finalizzazione	Team		