Testing Document UniMeet

Versione 1.0.0 del 04/02/2026

Versione	Modifiche
1.0.0	Compilazione

1. Test Plan

Si seguito verrà spiegato l'approccio al testing adottato dal team.
Per lo sviluppo del sistema è stata presa la decisione da parte del team di mettersi alla prova ed utilizzare tecnologie completamente nuove, sia per nome che per natura; quali il Framework Spring Boot accompagnato da JPA Hibernate e Vaadin.

Il team ha deciso quindi di adottare una strategia di **Just In Time Learning**, acquisendo quindi competenza parallelamente allo sviluppo effettivo del sistema.

Di conseguenza il testing delle funzionalità, qualità, ed integrità dei componenti dell'applicazione è stato pianificato e successivamente svolto manualmente, in modo tale da non appesantire ulteriormente il processo di apprendimento, creando quindi test automatizzati ancor prima di sapere cosa fare ma soprattutto come, e di conseguenza non essere preparati sul come testare.

Una volta finito lo sviluppo sono stati implementati dei test automatizzati:

- **Obiettivo**: Test automatizzato di supporto in previsione di manutenzioni e future implementazioni.
  I test automatizzati sono stati pensati come meccanismo rapido di controllo finalizzato a futuro refactoring, manutenzione e aggiunta di funzionalità, in modo tale da rilevare immediatamente errori e fallimenti qualora vengano compromessi servizi fondamentali core dell’applicazione, dando l’opportunità al manutentore di correggere subito il nuovo errore introdotto.

- **Ambiente di Test**: Java 17 (24.4.4), Junit 5 (5.10.5 ereditata da spring-boot-starter-test), Spring Boot 3.3.6, H2 Database

- **Strategia**: Dato che è stato seguito il Pattern entity - repository - service - UI dettato dall’utilizzo di Spring Boot + Vaadin, il testing automatizzato parte appunto dalle entità, vengono poi testate le repository e infine in modo più approfondito in quanto elementi logici più importanti i servizi.
  Il testing per l’interfaccia UI è stato svolto strettamente manualmente in quanto facilmente testabile in tal modo ed estremamente difficile se automatizzato, soprattutto in quanto l’UI è continuamente soggetta a variazione, a differenza dell’output atteso dalle funzioni core del sistema.

2. Casi di Test

Entità

StudySessionTest

- **testGetStatus_Upcoming**: Verifica che una studySession fissata nel futuro ritorni lo stato upcoming
- **testGetStatus_Ended**: Verifica che una studySession fissata nel passato ritorni lo stato di ended
- **testGetParticipantsAndOwner**: Controlla che in una sessione risulti che un partecipante e l'owner ne facciano parte

UniversityTest

- **testAddBuilding**: Verifica che l'aggiunta di un edificio crei un collegamento bidirezionale, assicurando che l'università contenga l'edificio e l'edificio riconosca l'università.
- **testRemoveBuilding**: Controlla che la rimozione di un edificio sciolga correttamente il legame tra le due entità, azzerando i riferimenti reciproci.

UserProfileTest

- **testAddReviewRating_FirstVote**: Verifica che, all'inserimento del primo voto, la reputazione del profilo corrisponda esattamente al punteggio assegnato e che il contatore dei votanti salga a uno.
- **testAddReviewRating_MultipleVotes**: Controlla il calcolo della media aritmetica della reputazione in presenza di più voti, assicurandosi che il totale dei votanti sia aggiornato correttamente.
- **testAddReviewRating_InvalidScore**: Accerta che il sistema impedisca l'inserimento di voti non validi (fuori dal range 1-5) lanciando l'apposita eccezione con il relativo messaggio di errore.

UserTest

- **testAddColleague**: Verifica che l'aggiunta di un collega crei un legame reciproco, assicurando che entrambi gli utenti compaiano reciprocamente nelle rispettive liste dei colleghi.
- **testRemoveColleague**: Controlla che la rimozione di un collega interrompa la relazione in modo bidirezionale, eliminando ogni riferimento tra i due utenti.
- **testEqualsAndHashCode**: Valida la logica di identità della classe, confermando che due utenti siano considerati uguali se e solo se possiedono lo stesso username.

Repository

ColleagueRequestRepositoryTest

- **testExistsRequestBetween**: Verifica che il sistema riconosca l'esistenza di una richiesta tra due utenti indipendentemente dall'ordine (chi ha inviato e chi ha ricevuto), garantendo l'assenza di falsi positivi con utenti non coinvolti.
- **testAreTheyColleagues**: Valida il funzionamento di una query nativa per accertare se due utenti sono già colleghi, testando la simmetria della relazione nel database e la corretta gestione dei casi di mancata associazione.

DepartmentRepositoryTest

- **testFindByUniversity**: Verifica che la ricerca per università restituisca correttamente l'elenco completo dei dipartimenti appartenenti a uno specifico ateneo, ignorando quelli associati ad altre università.
- **testFindByNameAndUniversity**: Controlla la capacità di individuare un singolo dipartimento tramite il match combinato di nome e università, validando che la ricerca fallisca correttamente se l'accoppiata non è coerente.

StudySessionRepositoryTest

- **testFindUpcomingOwnedSessions**: Verifica che vengano estratte solo le sessioni create dall'utente con data futura, escludendo correttamente quelle passate o appartenenti ad altri utenti.
- **testFindAllMyEndedHistory**: Accerta che il sistema recuperi correttamente lo storico delle sessioni terminate a cui l'utente ha partecipato, filtrando in base alla data antecedente a quella odierna.
- **testFindSessionWithDetailsById**: Valida l'efficacia del caricamento "Eager" (JOIN fetch), assicurandosi che l'oggetto caricato contenga l'intera gerarchia di dati (aula, edificio, università) senza incorrere in eccezioni di tipo LazyInitialization.
- **testFindActiveSessionsForUser**: Controlla che vengano individuate le sessioni attualmente in corso, ovvero quelle in cui l'orario attuale è compreso tra l'inizio e la fine della sessione nella giornata odierna.

StudyTableRepositoryTest

- **testFindByNumberAndRoom**: Verifica la capacità di trovare uno specifico tavolo tramite il suo numero identificativo all'interno di una determinata aula, validando anche il caso di ricerca per un numero inesistente.
- **testFindAllDetailsByRoomId**: Conferma che il recupero dei tavoli di una stanza avvenga con tutti i dettagli strutturali popolati, garantendo l'integrità dei riferimenti fino al livello dell'ateneo.
- **testFindByRoom_Building_University**: Valida la ricerca gerarchica, assicurando che sia possibile filtrare i tavoli appartenenti a un'intera università e che la lista risulti vuota per atenei senza infrastrutture registrate.

Service

BuildingServiceTest

- **testCreateBuilding**: Verifica la corretta creazione di un edificio e la sua associazione all'università specificata, accertando che i dati vengano persistiti correttamente.
- **testCreateDuplicate**: Assicura che il sistema impedisca la creazione di edifici duplicati (stesso nome) all'interno dello stesso ateneo, lanciando un'eccezione mirata.
- **testGetBuildingsForUser_UniAdmin**: Controlla che un amministratore universitario possa visualizzare esclusivamente gli edifici appartenenti al proprio ateneo, garantendo il corretto isolamento dei dati.
- **testGetBuildingsForUser_Admin**: Valida i permessi del Super Admin, verificando che abbia la visibilità completa su tutti gli edifici censiti nel sistema, indipendentemente dall'università.

ColleagueRequestServiceTest

- **testSendRequest**: Verifica l'invio di una richiesta di amicizia/collega tra due utenti, accertando che la richiesta risulti pendente per il destinatario.
- **testAcceptRequest**: Controlla che l'accettazione di una richiesta stabilisca una relazione bidirezionale tra gli utenti e che la richiesta stessa venga contestualmente rimossa dal database.
- **testSendRequest_AlreadyColleagues**: Assicura che non sia possibile inviare una richiesta a un utente che è già registrato come collega, lanciando un'eccezione di stato illegale.
- **testRemoveColleague**: Valida la logica di interruzione del rapporto, verificando che la rimozione di un collega sciolga correttamente il legame tra entrambi i profili.

DepartmentServiceTest

- **testCreateDepartment_Success**: Valida la creazione di un nuovo dipartimento, assicurandosi che venga assegnato correttamente all'università di riferimento e persistito nel sistema.
- **testCreateDepartment_Duplicate**: Verifica l'integrità dei dati impedendo la creazione di dipartimenti con nomi duplicati all'interno della stessa università tramite il lancio di un'eccezione.
- **testGetDepartmentsByUniversity**: Controlla che il servizio restituisca l'elenco completo dei dipartimenti associati a uno specifico ateneo, garantendo la correttezza dei filtri di ricerca.

ReviewServiceTest

- **testSubmitReview_Success**: Verifica che l'invio di una recensione aggiorni correttamente la reputazione media e il numero totale di votanti del profilo dell'utente recensito.
- **testReputationMath_MultipleReviews**: Testa la precisione del calcolo della media ponderata e degli arrotondamenti quando un utente riceve più recensioni da diversi partecipanti alla stessa sessione.

- **testSubmitReview_NotParticipant**: Accerta che il sistema blocchi i tentativi di recensione da parte di utenti che non hanno partecipato alla sessione di studio, garantendo la veridicità dei feedback.
- **testSubmitReview_DoubleVoting**: Assicura che un partecipante non possa inviare più di una recensione allo stesso utente per la medesima sessione, prevenendo manipolazioni del punteggio.

SessionInvitationServiceTest

- **testSendInvite_Success**: Verifica che l'invio di un invito per una sessione di studio crei correttamente un record di invito pendente associato al destinatario.
- **testAcceptInvite**: Controlla che l'accettazione di un invito aggiunga effettivamente l'utente alla lista dei partecipanti della sessione e rimuova contestualmente l'invito pendente dal sistema.
- **testSendInvite_UserBusy**: Valida la logica di prevenzione dei conflitti, assicurandosi che il sistema impedisca l'invito di un utente che è già impegnato in un'altra sessione (come proprietario o partecipante) nello stesso intervallo orario.

StudyTableServiceTest

- **testCreateStudyTable**: Accerta la corretta creazione di un tavolo di studio all'interno di un'aula specifica, verificando che i dati identificativi siano persistiti correttamente.
- **testCreateDuplicate**: Garantisce l'integrità del censimento delle aule impedendo la creazione di tavoli con lo stesso numero identificativo all'interno della medesima stanza.
- **testGetTablesForUser_UniAdmin**: Verifica che un amministratore universitario visualizzi esclusivamente i tavoli appartenenti alle strutture del proprio ateneo, confermando l'isolamento dei permessi.
- **testGetStudyTableByRoom**: Controlla il recupero della lista dei tavoli per una specifica aula, assicurandosi che il servizio carichi correttamente anche i dettagli gerarchici dell'infrastruttura.

StudySessionServiceTest

Servizio core dell’applicazione, di conseguenza testata in maniera più completa possibile

Test di Validazione Salvataggio

- **testSaveValidSession**: Verifica il corretto salvataggio di una sessione futura valida, accertando che le venga assegnato lo stato `UPCOMING`.
- **testInvalidTimes**: Controlla che il sistema impedisca la creazione di sessioni con orario di fine precedente a quello di inizio.
- **testPastDate**: Assicura che non sia possibile programmare sessioni in date già trascorse.

Test Sovrapposizioni e Conflitti

- **testTableOccupied**: Valida i tre casi di conflitto del tavolo (orario identico, sovrapposizione finale o iniziale), garantendo che il tavolo non sia prenotabile se già occupato.
- **testAdjacentSessions**: Verifica che sia possibile prenotare una sessione che inizia esattamente nello stesso istante in cui ne finisce un'altra allo stesso tavolo.
- **testOwnerBusyElsewhere**: Accerta che un utente non possa creare una sessione se è già impegnato come proprietario in un'altra sessione contemporanea su un tavolo diverso.

Test Gestione Partecipanti

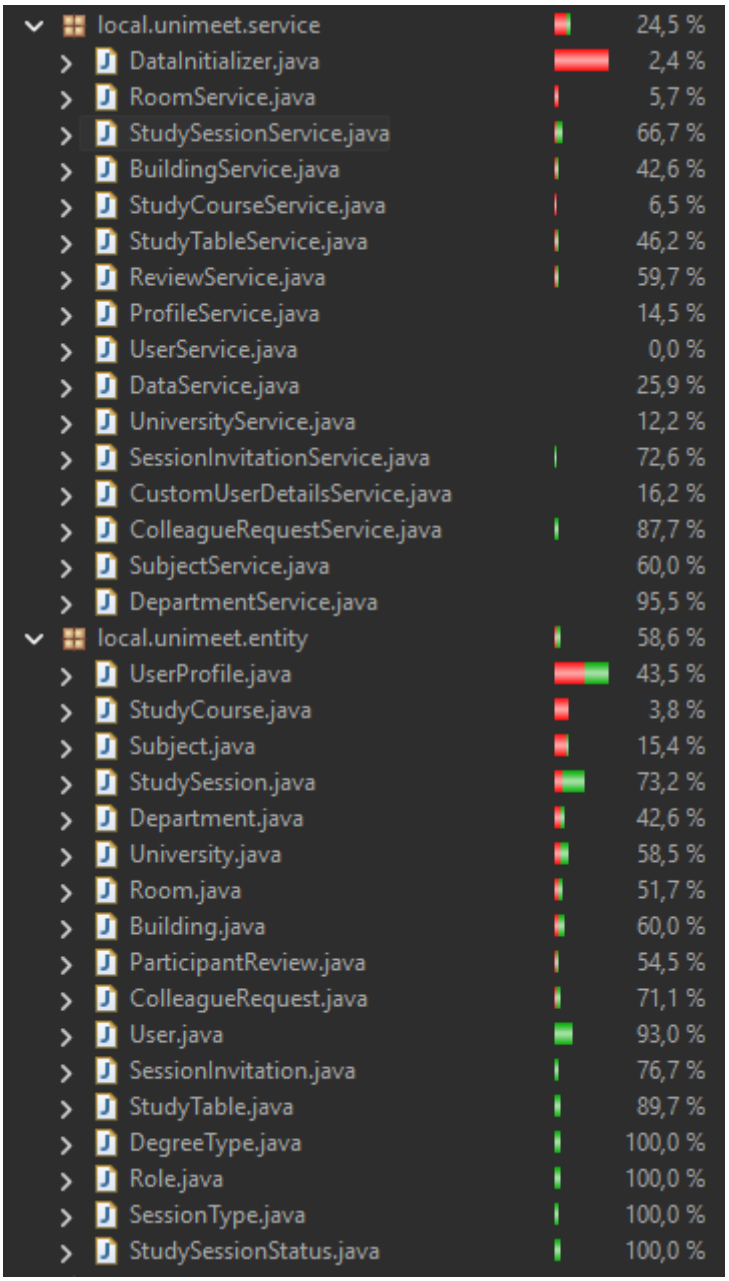
- **testAddParticipant_Success**: Conferma che i partecipanti vengano aggiunti correttamente alla sessione fino al raggiungimento della capacità massima.
- **testAddParticipant_CapacityExceeded**: Verifica che il sistema blocchi l'aggiunta di utenti oltre la capienza fisica del tavolo di studio.
- **testAddParticipant_UserBusy**: Assicura che un utente non possa essere aggiunto come partecipante se risulta già occupato in un'altra sessione nello stesso intervallo orario.
- **testAddOwnerAsParticipant**: Controlla che il sistema impedisca l'aggiunta ridondante del proprietario della sessione alla lista dei partecipanti.

Test Ciclo di Vita e Cancellazione

- **testDeleteUpcoming**: Valida la possibilità per il proprietario di eliminare correttamente una sessione programmata per il futuro.
- **testDeleteEnded**: Verifica il vincolo di integrità che impedisce l'eliminazione di sessioni già terminate.
- **testDeleteNonExistent**: Accerta che il tentativo di eliminare una sessione non censita sollevi l'eccezione corretta.

3. Test Coverage

Le classi testate dei Test case automatici Junit sono quelle principali del programma, soffermandosi maggiormente sui servizi core dell'applicazione, quali l'handling delle StudySession, la gestione di SessioneInvitation e ColleagueRequest e la creazione di strutture.



Si noti come i servizi core siano ampiamente coperti.