Maintenance Document UniMeet

Versione 1.0.0 del 25/01/26

Versione	Modifiche
1.0.0	Compilazione

1. Introduzione e Finalità

Il presente documento descrive l'attenzione che è stata data e verrà data ad UniMeet.

Vogliamo garantire la corretta e consistente attività del sistema, servizio sia sempre disponibile e integrità dei dati relativi a sessioni di studio, inviti, richieste e informazioni personali degli studenti

Il sistema si appoggia sul framework di sviluppo Spring Boot, i microservizi implementati atti all'inserimento e alla manutenzione dei dati devono garantire continuità nel rispetto dell'integrità dei dati, anche del caso di aggiunta di nuovi elementi funzionali.

2. Manutenzione software:

Durante lo sviluppo del software è stato necessario svolgere un continuo perfezionamento e ripensamento delle logiche sia strutturali che locali di gestione.

2.1 Manutenzione Correttiva

Essendo l'applicazione stata sviluppata con una strategia Just In Time Learning (come specificato in documenti precedenti), è stato spesso necessario rivisitare il codice e le funzionalità già scritte.

- **Accorgimenti, redesigning UI**: Durante lo sviluppo a fronte dell'implementazione di nuovi requisiti è stato spesso necessario rivisitare le vecchie interfacce sviluppate, oltre che a spesso ribaltare completamente certe disposizioni.

- **Correzione falle logiche in servizi:** Spesso è stato necessario ripensare la logica con la quale un certo servizio gestiva entità del sistema, durante il testing manuale si rilevavano spesso come certi requisiti di integrità dei dati non venissero rispettati e fosse quindi necessario ripensare l'intera logica di integrità del servizio, ogni tanto riscendendo anche per la struttura a layer dell'applicazione, passando da @Service a @Repository a @Entity, facendo quindi del vero e proprio **Reverse Engineering**, ridefinendo le logiche di funzionamento a partire dal basso, spesso cambiano la struttura di una classe:

Esempio: Inizialmente gli inviti ad una sessione erano stati pensati come permanentemente memorizzati all'interno dell'applicazione e classificati e gestiti tramite un relativo stato: PENDING / ACCEPTED / REJECTED.
Questa logica seppur completa si è rilevata inutilmente complicata per un'applicazione di questo livello e a seguito di un processo di reengineering lo stato dell'invito è stato rimosso, ora un invito è pending solo se è presente nel database, una volta che viene accettato o rifiutato viene istantaneamente eliminato.

- **Refactoring: togliere Repositories dall'UI:** Spesso per implementare velocemente aspetti grafici del sistema è successo di importare Repository all'interno dell'UI al posto di servizi.
  Per convenzione e correttezza logica in un'applicazione spring boot vaadin, nelle classi di view, di interfaccia vanno importati solo i servizi, i quali detengono la logico core del sistema.
  Un esempio di refactoring eseguito è stato quindi sostituire Repository con i relativi servizi dove necessario.

2.2 Manutenzione Adattiva

Scenari nel caso sarebbe necessario ricorrere a manutenzione adattiva

- **Aggiornamento del Framework:** Adeguamento alle nuove release di Spring Boot per garantire la sicurezza del sistema, questa sarà probabilmente necessaria in futuro in quanto a inizio progetto a scopo esemplificativo sono stati adottati dei metodi vaadin deprecati (quindi a EOL).
- **Aggiornamento policy Universitarie:** Nel caso certe università muovano richieste in termini di privacy nei confronti di UniMeat probabilmente non si avrà alternativa se non essere accomodanti.

2.3 Manutenzione Perfettiva

- **Refactoring: Ottimizzazione Query:** Soprattutto a inizio progetto è stato ricorrente sviluppare velocemente le Repository e le entity, mettendo il fetch type = EAGER per evitare errori di LAZY fetch con Hibernate, ossia forzare il fetch ricorsivo e completo degli elementi impegnati in relazioni con altri, in modo tale da non trovarsi in situazioni dove certi oggetti restituissero relazioni nulle provocando il crash del sistema.
  Seppur funzionante questo approccio è tutt'altro che efficiente, è stata quindi svolta un attività di ottimizzazione, scrivendo in certe repository query personalizzate per il retrieval completo di certi dati a seguito di join senza dover forzare un tipo di fetch troppo pesante ed aggressivo.
- **Miglioramento Interfaccia:** Ottimizzazione dei componenti di navigazione Vaadin affinchè l'interfaccia web dell'utente risulti più intuitiva e scorrevole possibile.

2.4 Manutenzione Preventiva:

Non è stata eseguita particolare attività di manutenzione preventiva a causa si assenza di tempo, per lo più purtroppo la javadoc dell'applicazione risulta essere scarsa.

3. Modello di Governance

Qui definiamo in quale modo si deve eseguire la manutenzione di UniMeet in un ipotetico futuro in cui una community utilizza l'applicazione e viene aperto un canale di segnalazione bug, consigli implementativi ufficiale.

- **Segnalazione:** Registrazione dell'anomalia o della richiesta di modifica nel sistema
- **Valutazione:** Analisi delle dipendenze tra i componenti UI e i servizi di backend .
- **Ambiente di Staging:** Sviluppo e test delle soluzioni in un ambiente isolato.
- **Validazione:** Verifica e validazione della soluzione trovata.
- **Rilascio:** rilascio della nuova versione aggiornata.

4. Manutenzione Prevista

VI sono ancora dei ritocchi da fare già individuati che non saranno possibile essere fatti entro la consegna del progetto, quali:

- Eliminare definitivamente import di Repository dall'UN: seppur in modo limitato è ancora possibile vedere come alcune view importino repositories
- Eliminazione Servizio riassuntivo di supporto Dataservice: dataservice è servito come supporto rapido per sviluppare l'interfaccia relativa al profilo utente, purtroppo non vi è stato il tempo di rimuoverla e rendere appunto l'interfaccia profilo utente indipendente da essa.
- Metodi duplicati / Refactoring generale: non sono stati fatti controlli in profondità riguardo metodi simili se non uguali ripetuti in varie classi, sovracomplesssità o sovraffollamneto del codice in singole classi, o comunque rispetto di standard di qualità alti di programmazione.
- Aumento Consistente Javadoc: purtroppo non c'è stato il tempo di produrre una javadoc esaustiva, pochissimi metodi sono disposti di javadoc e i pochi che lo sono hanno una descrizione molto sintetica.

5. Gestione del Rischio e Continuità Operativa

Ovviamente un aspetto fondamentale è la certezza di fornire continuità nel servizio, nel caso l'applicazione abbia un riscontro pubblico verranno adottate le seguenti strategie:.

Miglioramento Database: Si passerà dal debole e temporaneo database embedded H2 a un vero e proprio database in ridondanza, in modo tale da essere fail-proof.

Piano di Rollback: Ogni aggiornamento software verrà congiunto ad una procedura di ripristino immediato. Qualora il nuovo rilascio presenti instabilità, il sistema verrà portato alla versione precedente entro un tempo massimo di 30 minuti.