Progetto Gestione Questionari:

> universita bicocca - ingegneria del software
> *mazzini andrea 885868*

Descrizione del progetto:

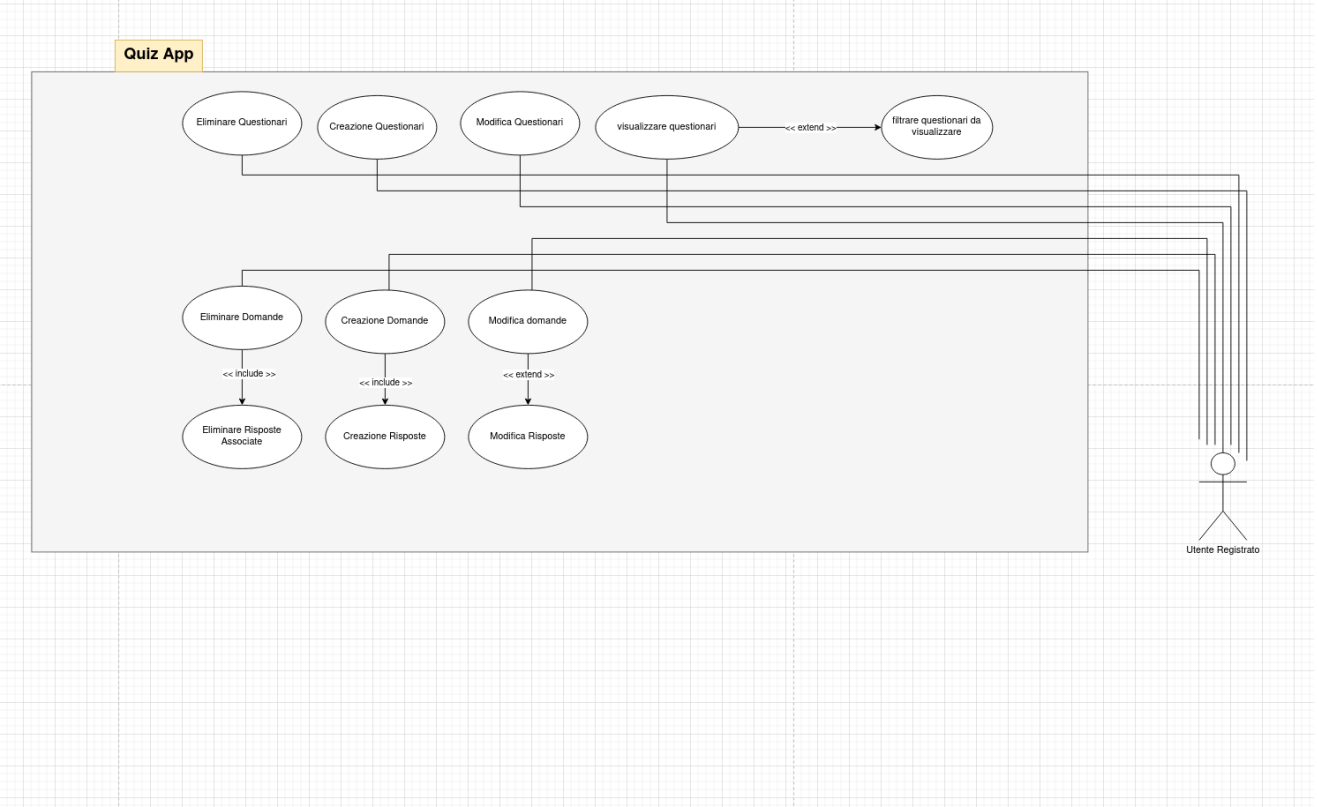
Il progetto consiste in una piattaforma per la creazione, gestione e compilazione di questionari su vari argomenti. Gli utenti registrati potranno creare domande testuali o con immagini, con risposte di tipo aperto (con limiti di caratteri) o chiuso (scelte multiple). Le domande saranno categorizzate e memorizzate in un database, con la possibilità di cercarle tramite una funzione di ricerca.

I questionari saranno composti da domande già esistenti, con funzionalità di salvataggio, modifica e cancellazione. Gli utenti potranno compilare i questionari con la possibilità di salvare temporaneamente le risposte o finalizzare la compilazione. Una volta completato il questionario, verrà generato un PDF e inviata una notifica via email.

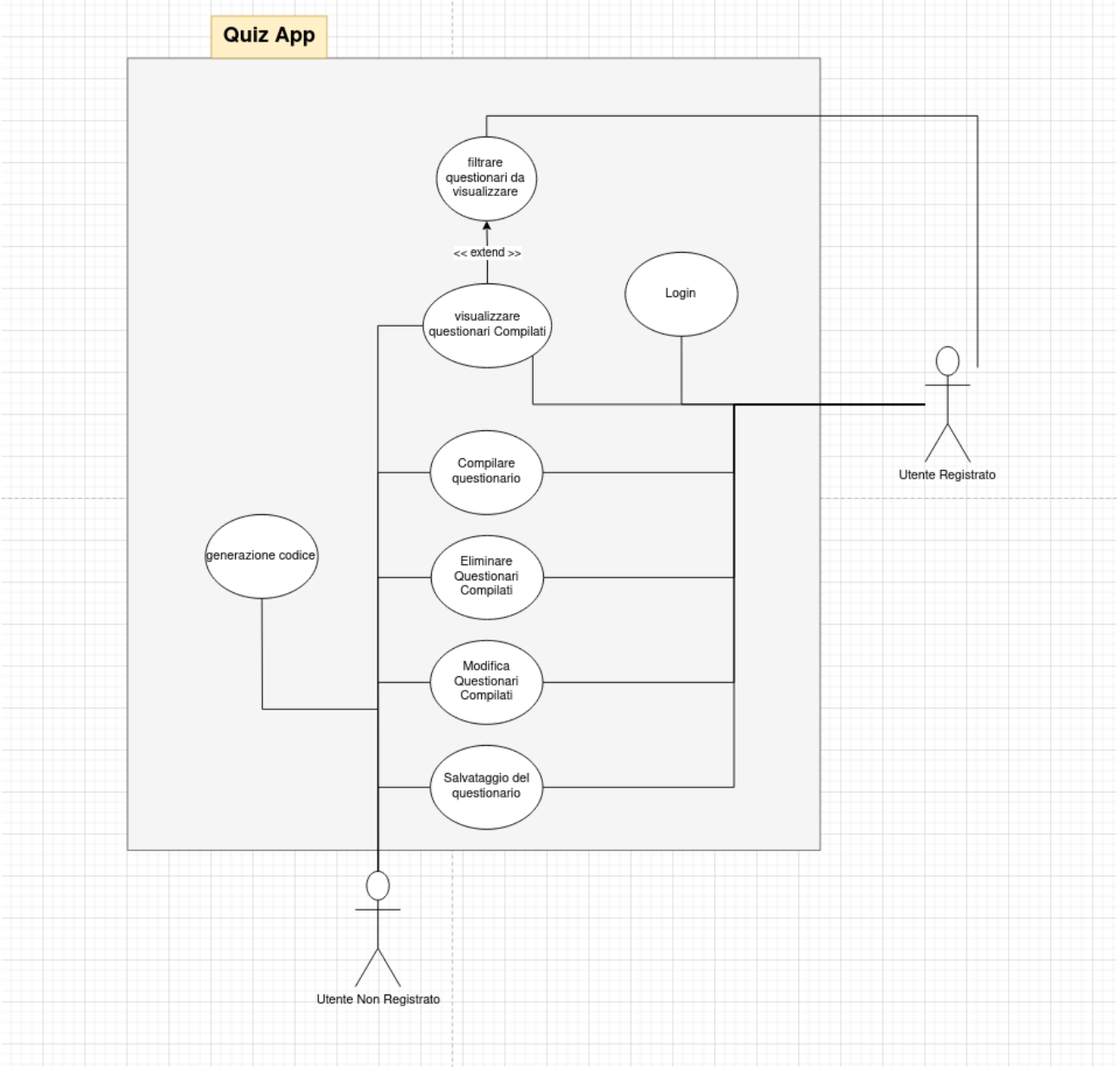
Gli utenti registrati potranno consultare i questionari già compilati, mentre agli utenti non registrati verrà fornito un codice univoco per visualizzare, modificare o cancellare i questionari compilati.

Use Case:

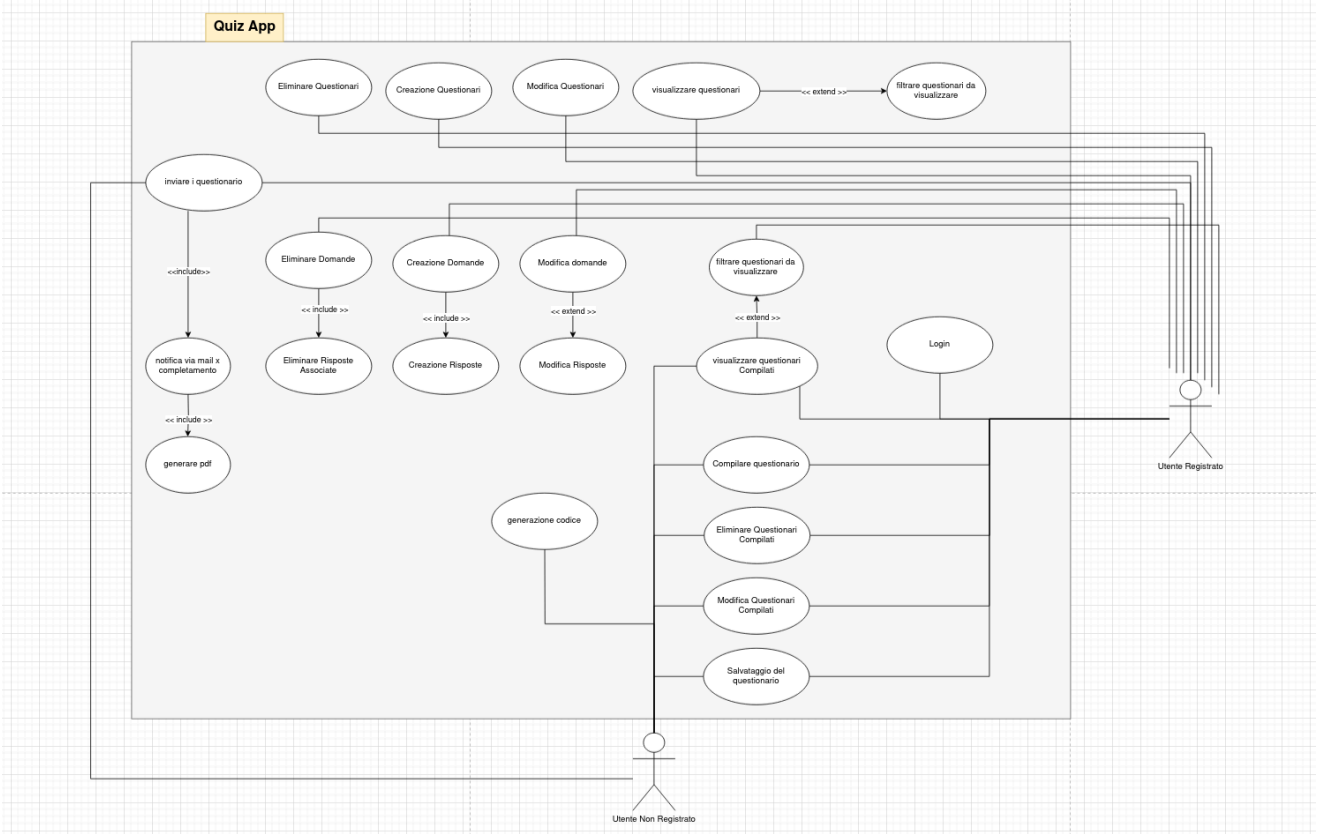
Gli use case descrivono le funzionalità principali dell'applicazione di gestione dei questionari. Ogni caso d'uso identifica le interazioni tra gli utenti (registrati e non) e il sistema, coprendo l'intero ciclo di vita dei questionari, dalla creazione e gestione delle domande fino alla compilazione e revisione. I seguenti casi d'uso evidenziano i tre principali domini operativi dell'app: gestione dei quiz e delle domande, gestione degli utenti, e compilazione dei questionari, descrivendo in dettaglio come l'app supporti queste attività chiave.



1. Dominio dei Quiz e delle Domande:
    - Eliminazione, Creazione e Modifica dei Questionari: L'app consente di gestire i questionari con operazioni di eliminazione, creazione e modifica. Le domande possono essere create, modificate o eliminate e associate ai questionari.
    - Creazione, Modifica ed Eliminazione delle Domande: Il sistema supporta l'inclusione di domande in base a tipi di risposte, immagini o testo, con la possibilità di eliminare o modificare le risposte associate.
    - Visualizzazione e Filtraggio dei Questionari: Gli utenti possono visualizzare e filtrare i questionari esistenti in base a criteri definiti.

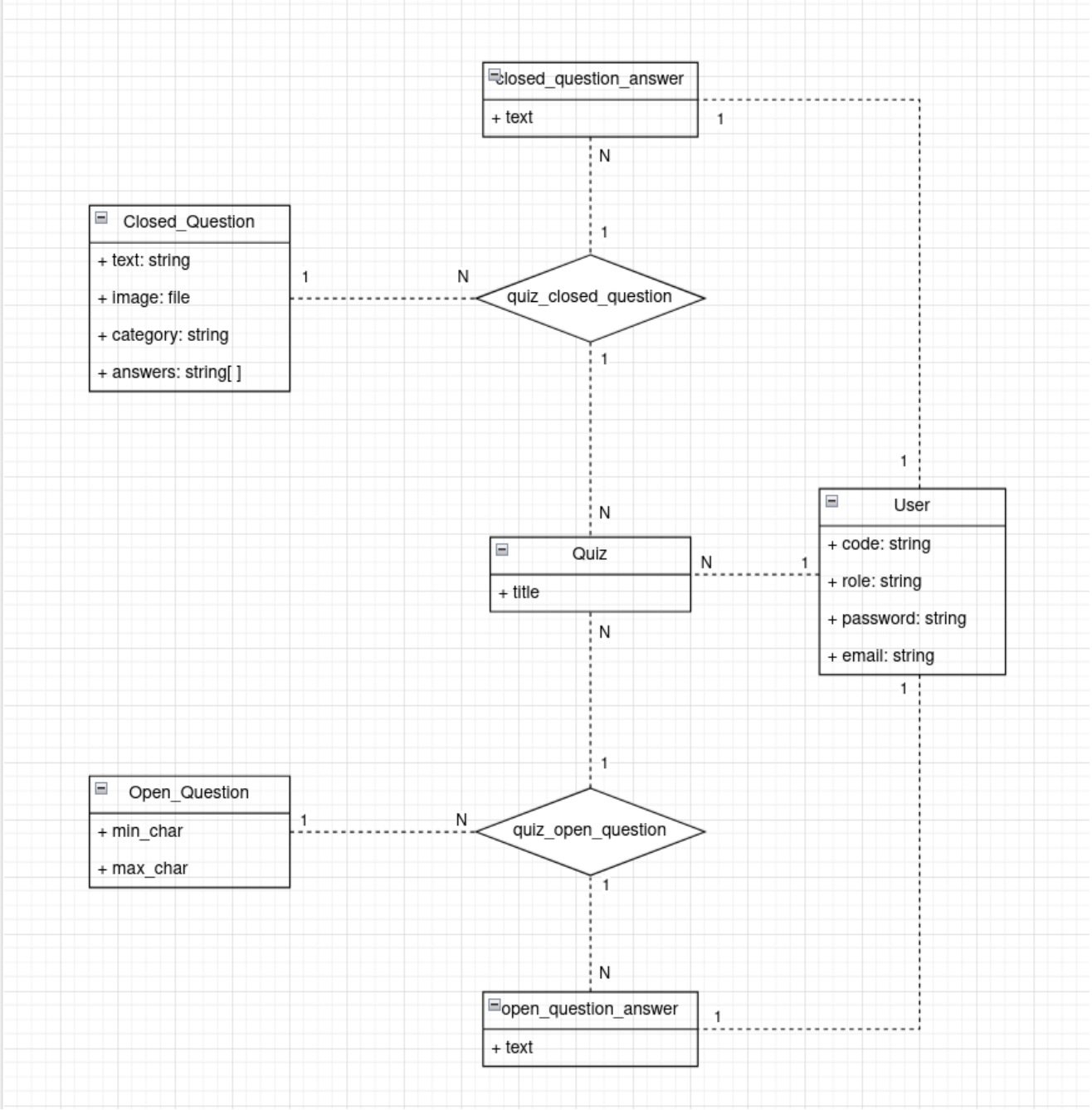


2. Dominio della Gestione degli Utenti:
    - Utenti Registrati: Gli utenti registrati possono accedere all'app attraverso il login e gestire i questionari completati, inclusa la modifica o l'eliminazione dei questionari compilati.
    - Utenti Non Registrati: Viene fornito un codice univoco per permettere agli utenti non registrati di visualizzare, modificare o eliminare i loro questionari completati.
    - Generazione di Codice: Per gli utenti non registrati, il sistema genera un codice associato a ciascun questionario compilato.
    - Compilazione del Questionario: Gli utenti, sia registrati che non, possono compilare i questionari. Il salvataggio intermedio e finale delle risposte è consentito.



3. Dominio della Compilazione dei Questionari:
    - Gestione dei Questionari Compilati: Una volta completato, il questionario può essere visualizzato, modificato o eliminato.
    - Generazione PDF e Notifica: Al completamento della compilazione, viene generato un PDF e viene inviata una notifica via email.

Modello di Dominio:



SSD: create open question

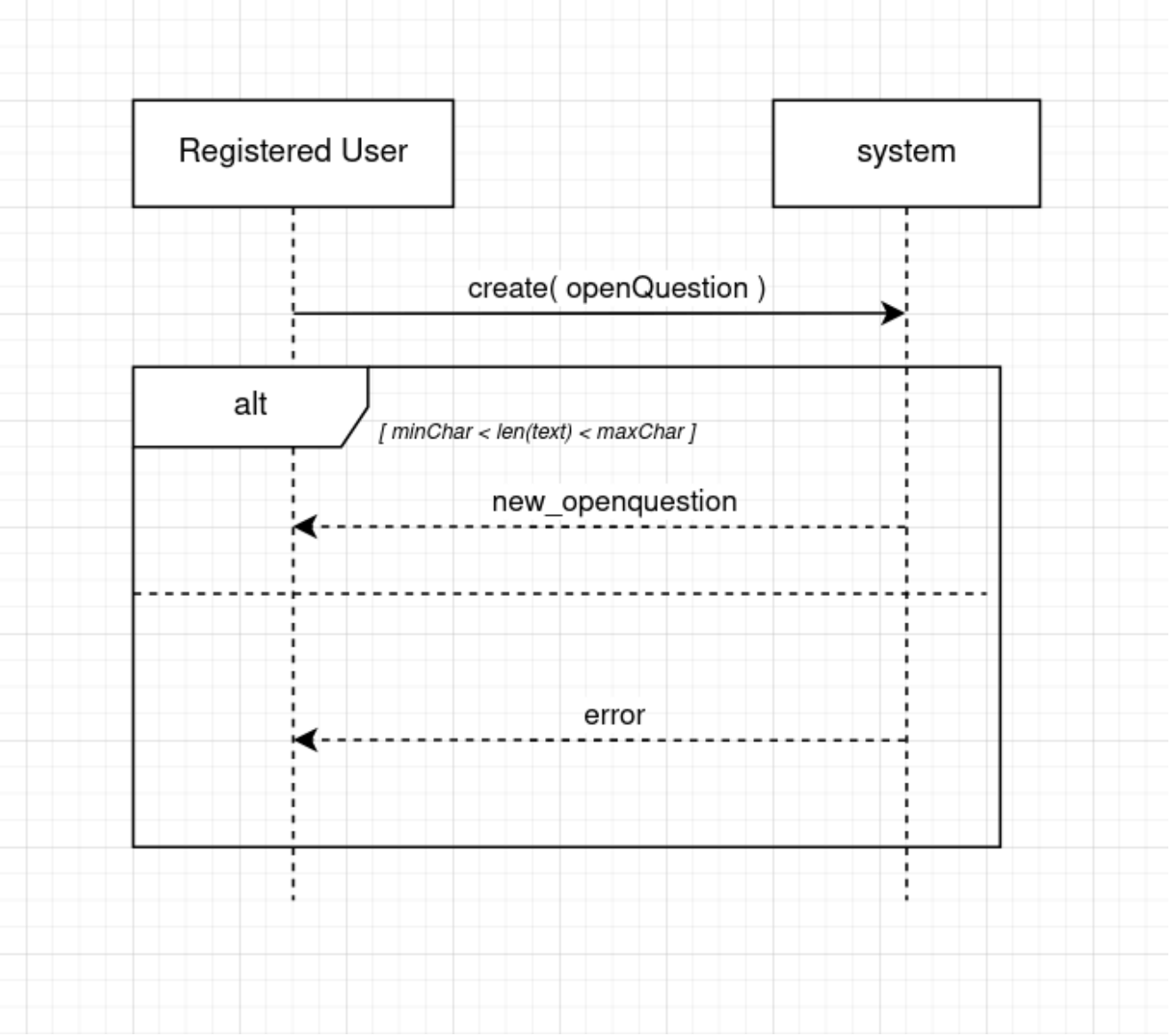
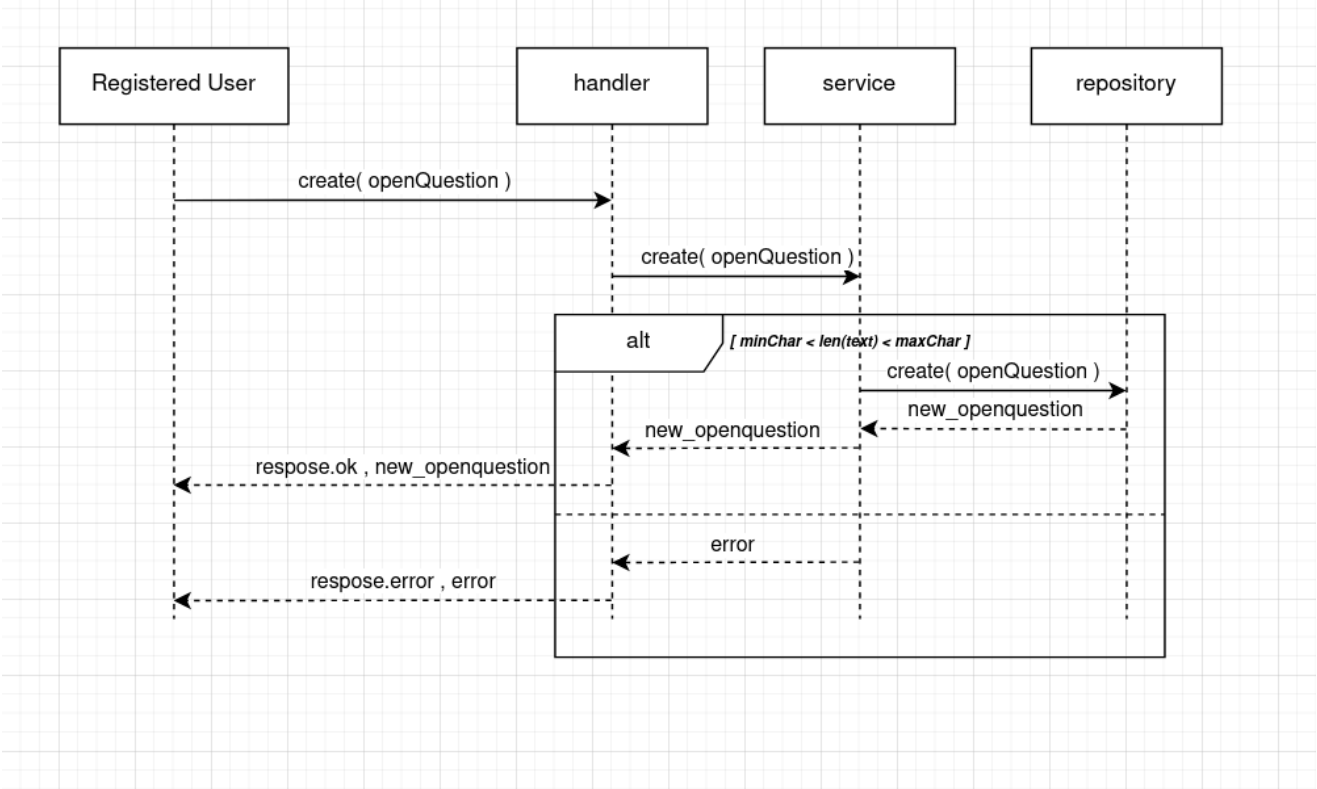
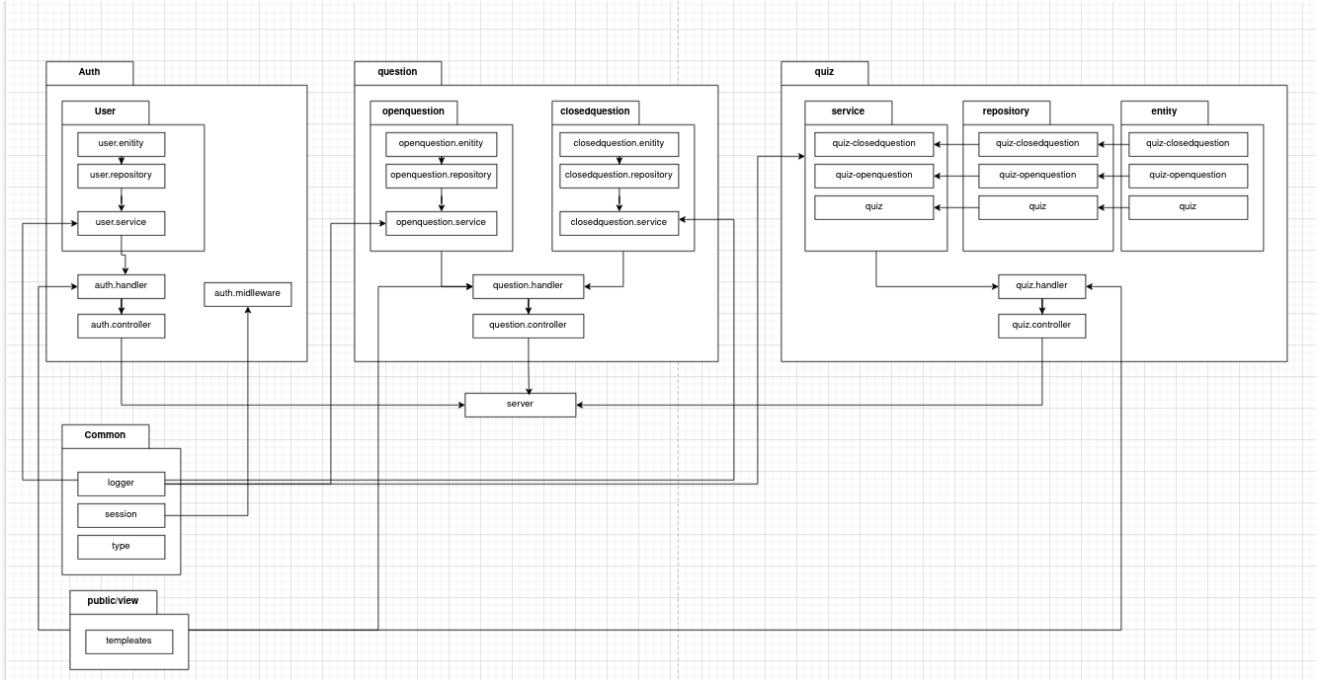


Diagramma di sequenza di progettazione:



Architettura software:



Design principle e Design Pattern:

Nel progetto si segue il design pinciple SOLID che stabilisce le seguenti linee guida:

1. S - Single Responsibility Principle (SRP)

- **Principio di Responsabilità Unica**: Ogni classe o modulo dovrebbe avere una sola ragione per cambiare, cioè dovrebbe avere una singola responsabilità.
- Esempio: nel progetto ogni modulo (`auth`, `quiz` e `question`) ha una responsabilità ben definta. Ad esempio, il modulo `auth` si occupa dell'autenticazione degli utenti, mentre i moduli `openquestion` e `closedquestion` gestiscono i diversi tipi di domande. Questa separazione di responsabilità segue il SRP.

2. **O - Open/Closed Principle (OCP)**
   - **Principio Aperto/Chiuso**: Le classi devono essere aperte per estensioni ma chiuse per modifiche.
   - Esempio: non c'è bisogno di modificare i tipi esistenti per gestire future modifiche, quindi il sistema è aperto per estensioni ma chiuso per modifiche
3. **L - Liskov Substitution Principle (LSP)**
   - **Principio di Sostituzione di Liskov**: Gli oggetti di una sottoclasse devono poter essere sostituiti con oggetti della loro superclasse senza alterare il comportamento corretto del programma.
4. **I - Interface Segregation Principle (ISP)**
   - **Principio di Segregazione delle Interfacce**: Le classi non dovrebbero essere forzate a implementare interfacce che non usano. È meglio avere più interfacce piccole e specifiche per i clienti piuttosto che una grande interfaccia generica.
   - Esempio: la suddivisione nei vari handler e nei services permette di importare avere funzioni ben specializzate.
5. **D - Dependency Inversion Principle (DIP)**
   - **Principio di Inversione delle Dipendenze**: I moduli di alto livello non dovrebbero dipendere da quelli di basso livello. Entrambi dovrebbero dipendere da astrazioni. Le astrazioni non dovrebbero dipendere dai dettagli, ma i dettagli devono dipendere dalle astrazioni.
   - Esempio: Il modulo `auth.handler`, `question.handler`, e `quiz.handler` gestirscono la logica principale senza dipendere direttamente da implementazioni specifiche, ma dai services che vegono iniettati direttamente dove serve.

Design Pattern Applicati:

- **Repository Pattern**:
  il Repository Pattern, viene usato con le entità `user.repository`,

`openquestion.repository`, `closedquestion.repository`, e `quiz.repository`. Questo pattern è utilizzato per separare la logica di accesso ai dati dalla logica di business, fornendo un'interfaccia per interagire con il database in modo astratto.

- **Service Layer Pattern:**
  I moduli `user.service`, `openquestion.service`, e `closedquestion.service` applicano il **Service Layer Pattern**, che separa la logica di business dalle operazioni più basilari del database o dell'infrastruttura. Questo facilita il riuso e la manutenibilità del codice.ù
- **Dependency Injection (DI)**
  la dependency injection è un pattern di progettazione che segue il **principio dell'inversione delle dip endenze (DIP)** e viene utilizzato per rendere il codice più modulare, flessibile e testabile. Nel progetto viene usato sopratutto con i vari `handler` a cui viene iniettato tutti i `services` di cui ha bisogno.