Documento di Design dell’Architettura Software

1. Introduzione

1.1 Scopo del documento

Il presente documento descrive l'architettura software del progetto **Web App per la gestione di gruppi di studio**, con particolare attenzione alle componenti principali, alle interazioni tra i vari livelli e alle scelte tecnologiche adottate. Inoltre, verrà presentato lo pseudocodice degli algoritmi fondamentali utilizzati all'interno del sistema.

1.2 Contesto del sistema

L'architettura del sistema segue il pattern **Model-View-Controller (MVC)**, che consente di separare la logica di business, la presentazione e il controllo del flusso dell'applicazione. La web app, sviluppata utilizzando **Spring Framework** e **Spring Security**, interagisce con un database embedded **H2** per la gestione dei dati relativi agli utenti, ai gruppi di studio e alle notifiche.

2. Descrizione dell’Architettura

2.1 Architettura a tre livelli (MVC)

L'applicazione è progettata utilizzando un'architettura a tre livelli composta da:

- **Presentation Layer (View)**: Gestisce l'interfaccia utente e la visualizzazione delle informazioni. In questo livello, viene utilizzato **Thymeleaf** come motore di template per generare dinamicamente le pagine HTML.
- **Business Logic Layer (Controller)**: Gestisce la logica di business e l'elaborazione delle richieste provenienti dalla vista. Utilizza i **Controller di Spring** per coordinare le operazioni tra la vista e il modello.
- **Data Access Layer (Model)**: Gestisce l'accesso ai dati e la persistenza. Questa parte dell'architettura utilizza **JPA (Java Persistence API)** e **H2 Database** per l'interazione con il database.

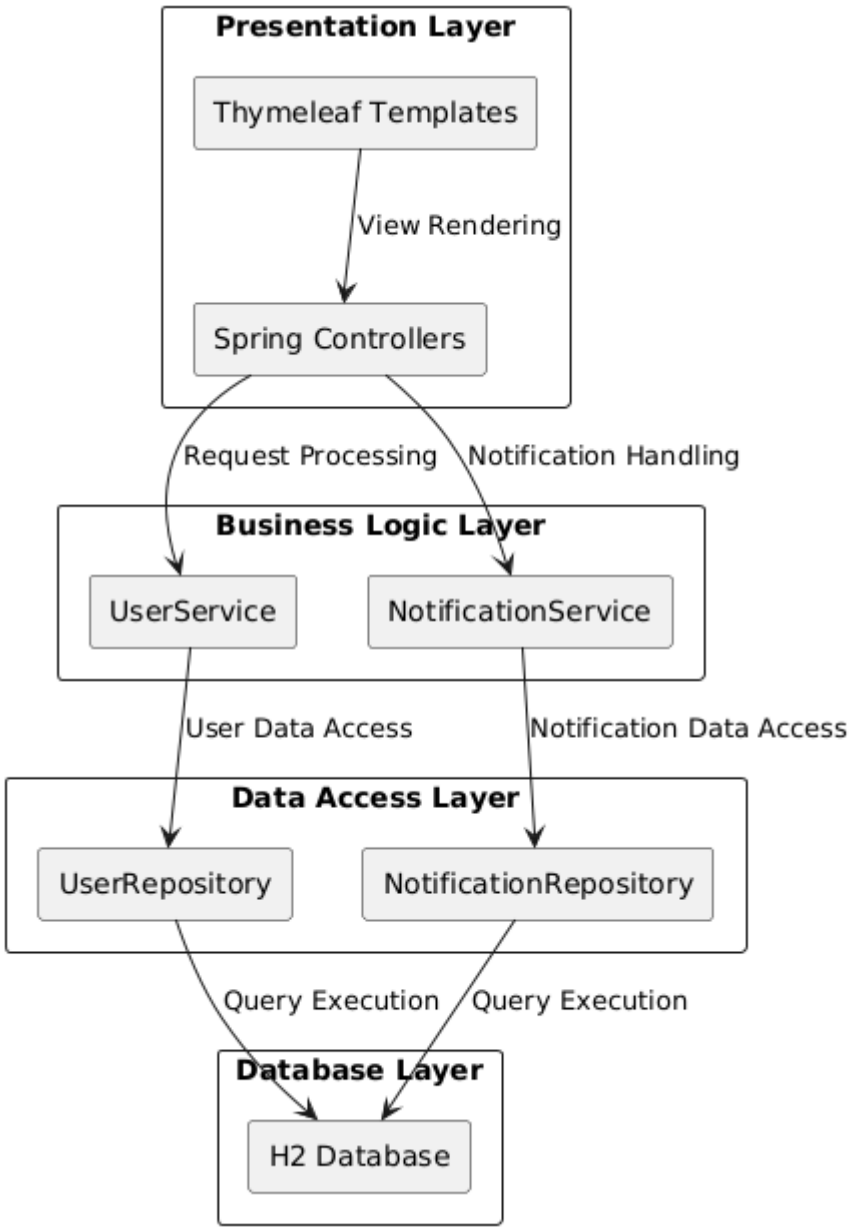
2.2 Componenti principali dell’architettura

1. **Controller**: I controller di Spring elaborano le richieste HTTP e orchestrano le interazioni tra la vista e il modello. Alcuni esempi di controller sono:
    - **LoginController**: Gestisce le richieste di login.
    - **RegistrationController**: Gestisce le richieste di registrazione e validazione delle informazioni.
    - **ProfileController**: Permette agli utenti di visualizzare e aggiornare i propri profili.
    - **NotificationController**: Gestisce l'invio e la ricezione di notifiche per formare gruppi di studio.
2. **Service Layer**: Contiene la logica di business dell'applicazione. Gli oggetti service si occupano di eseguire operazioni complesse, come la gestione delle password e delle notifiche. Alcuni servizi includono:
    - **UserService**: Responsabile della gestione degli utenti (registrazione, aggiornamento profilo, ecc.).
    - **NotificationService**: Gestisce l'invio di notifiche agli utenti.
3. **Repository Layer**: Utilizza **JPA** per l'interazione con il database. Le classi repository forniscono metodi per eseguire query e operazioni CRUD (Create, Read, Update, Delete). Alcuni repository includono:

- **UserRepository**: Permette la gestione dei dati relativi agli utenti.
- **NotificationRepository**: Permette la gestione delle notifiche inviate e ricevute dagli utenti.

4. **Entity Layer**: Contiene le classi che mappano le tabelle del database. Gli oggetti in questo livello rappresentano le entità del sistema, come:
   - **User**: Rappresenta gli utenti della piattaforma.
   - **Notification**: Rappresenta le notifiche inviate tra utenti.
   - **Role**: Gestisce i ruoli degli utenti (es. utente standard, amministratore).

2.3 Diagramma dell'architettura



3. Design Dettagliato delle Componenti

3.1 LoginController

Il **LoginController** gestisce il processo di autenticazione. Utilizza **Spring Security** per la validazione delle credenziali e il controllo degli accessi.

Pseudocodice per il login:

```
// Metodo per gestire la richiesta di login
funzione gestisciLogin(username, password):
    utente = UserService.trovaUtentePerUsername(username)
    se utente == null:
        mostraErrore("Utente non trovato")
    se !PasswordValidator.valida(password, utente.password):
        mostraErrore("Password errata")
    altrimenti:
        autenticaUtente(utente)
        reindirizzaVersoHome()
```

3.2 RegistrationController

Il **RegistrationController** gestisce la registrazione di nuovi utenti, con validazione delle informazioni inserite e salvataggio nel database.

Pseudocodice per la registrazione:

```
// Metodo per gestire la richiesta di registrazione
funzione gestisciRegistrazione(formRegistrazione):
    se !validaEmail(formRegistrazione.email):
        mostraErrore("Email non valida")
    se UserService.trovaUtentePerEmail(formRegistrazione.email) != null:
        mostraErrore("Email già in uso")
    se !validaPassword(formRegistrazione.password):
        mostraErrore("Password non valida")
    altrimenti:
        utente = creaUtente(formRegistrazione)
        UserService.salva(utente)
        mostraMessaggio("Registrazione completata con successo")
```

3.3 NotificationService

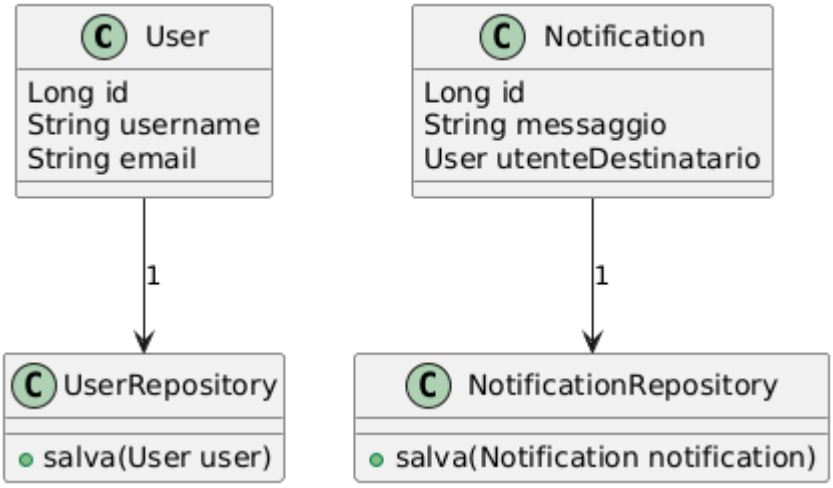
Il **NotificationService** gestisce l’invio e la ricezione di notifiche tra utenti, coordinando il salvataggio delle notifiche nel database.

Pseudocodice per l'invio di notifiche:

```
// Metodo per inviare una notifica a un utente
funzione inviaNotifica(utenteDestinatario, messaggio):
    notifica = creaNotifica(utenteDestinatario, messaggio)
    NotificationRepository.salva(notifica)
    inviaNotificaReale(utenteDestinatario.email, messaggio)
    mostraMessaggio("Notifica inviata con successo")
```

4. Diagrammi UML delle Classi Principali

4.1 Diagramma delle Classi (Esempio semplificato)



5. Scelte Progettuali

5.1 Pattern Architetturali

- **MVC (Model-View-Controller)**: Questo pattern separa chiaramente la logica di presentazione dalla logica di business e dai dati, facilitando la manutenibilità e la scalabilità del sistema.
- **DTO (Data Transfer Object)**: I DTO vengono utilizzati per trasferire dati tra le varie componenti del sistema, riducendo il legame tra il modello di dominio e la vista.

5.2 Sicurezza

L'autenticazione e l'autorizzazione degli utenti sono gestite tramite **Spring Security**, che fornisce una gestione sicura delle sessioni utente e delle credenziali. Le password vengono memorizzate in formato criptato utilizzando algoritmi sicuri (ad esempio **bcrypt**).

6. Conclusioni

Il design dell'architettura presentato in questo documento fornisce una soluzione modulare, scalabile e sicura per la gestione di una piattaforma che permette la formazione di gruppi di studio online. L’utilizzo di pattern consolidati come MVC e l'integrazione di strumenti come Spring Security e JPA garantiscono la robustezza e l'efficienza del sistema.