Social Media Platform group 32 - JPA Assignment

Informazioni Studenti

Nome: Francesco Saponara

Matricola: 886465

Nome: Mattia Chittoni

Matricola: 886462

Corso: Software Development Processes

Anno Accademico: 2025-2026

Link Repository GitLab

Repository: https://gitlab.com/f.saponara/2025_assignment3_social_media_group32

Panoramica del Progetto

Descrizione dell'Applicazione

Questa applicazione Java implementa un sistema di social media utilizzando JPA (Java Persistence API) con Hibernate come provider ORM. Il sistema gestisce utenti, post, commenti e hashtag con relazioni complesse tra le entità.

Architettura del Sistema

Diagramma delle Classi

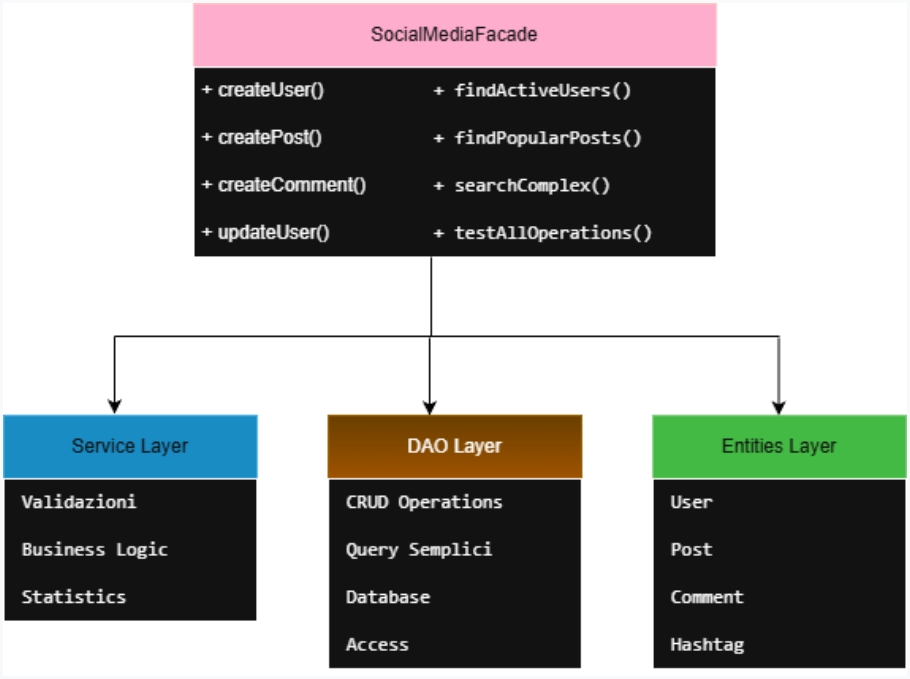
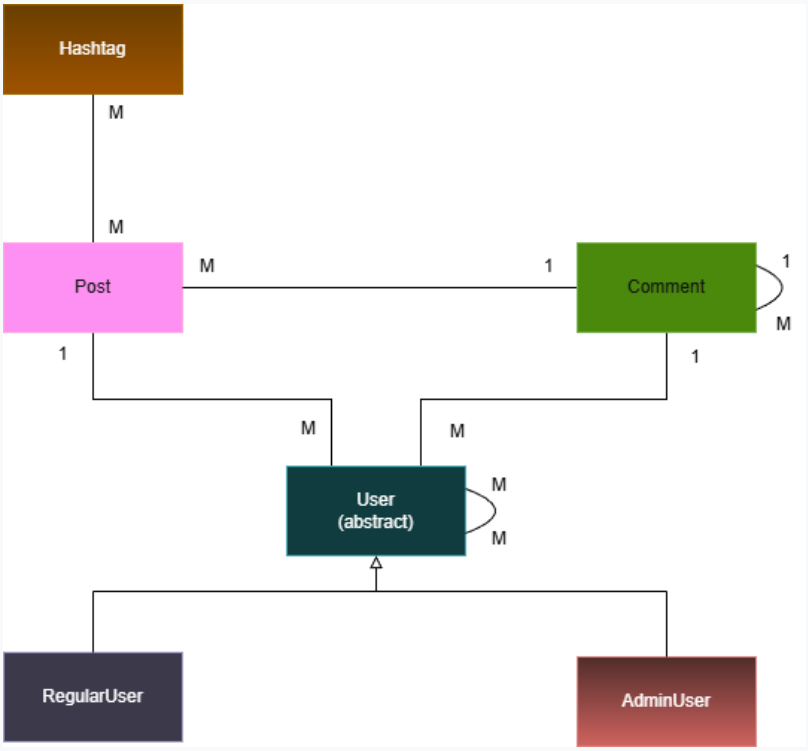


Diagramma ER (Entity-Relationship)



Entità e Relazioni

1. User (Entità Base Astratta)

```
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
public abstract class User {
    private Long id;
    private String username;
    private String email;
    private LocalDate joinDate;
}
```

2. RegularUser & AdminUser (Ereditarietà)

```
@Entity
public class RegularUser extends User {
    private String bio;
    private LocalDate birthDate;
    private String location;
}

@Entity
public class AdminUser extends User {
    private String adminLevel;
    private String department;
    private LocalDate promotionDate;
}
```

3. Post (Contenuti)

```
@Entity
public class Post {
    private Long id;
    private String content;
    private String postType;
    private LocalDateTime timestamp;
    private User author;  // Many-to-One
    private Set<Hashtag> hashtags;  // Many-to-Many
    private Set<User> likedBy;  // Many-to-Many
}
```

4. Comment (Commenti con Self-Loop)

```
@Entity
public class Comment {
    private Long id;
    private String content;
    private LocalDateTime timestamp;
    private Post post;  // Many-to-One
    private User author;  // Many-to-One
    private Comment parentComment;  // Self-loop (replies)
    private List<Comment> replies;  // Self-loop
}
```

5. Hashtag (Categorie)

```
@Entity
public class Hashtag {
    private String tag;  // Primary Key
    private int usageCount;
    private Set<Post> posts;  // Many-to-Many
}
```

Relazioni Implementate

1. Many-to-Many: User ↔ User (Following/Followers)

```
// In User entity:
@ManyToMany
@JoinTable(name = "user_following")
private Set<User> following;

@ManyToMany(mappedBy = "following")
private Set<User> followers;
```

2. Many-to-Many: Post ↔ Hashtag

```
// In Post entity:
@ManyToMany
@JoinTable(name = "post_hashtags")
private Set<Hashtag> hashtags;

// In Hashtag entity:
@ManyToMany(mappedBy = "hashtags")
private Set<Post> posts;
```

3. Self-Loop: Comment ↔ Comment (Replies)

```
// In Comment entity:
@ManyToOne
private Comment parentComment;

@OneToMany(mappedBy = "parentComment")
private List<Comment> replies;
```

4. One-to-Many: User → Post (Author)

```
// In User entity:
@OneToMany(mappedBy = "author")
private Set<Post> posts;

// In Post entity:
@ManyToOne
private User author;
```

Operazioni Implementate

CRUD Operations (per tutte le entità)

- **Create:** createUser(), createPost(), createComment()
- **Read:** findById(), findAll(), findByUsername()
- **Update:** updateUser(), updatePost()
- **Delete:** deleteUser(), deletePost()

Complex Search Operations

```
// 1. Utenti attivi (post recenti con followers)
List<User> findActiveUsers(LocalDateTime since);

// 2. Post popolari per hashtag e like minimi
List<Post> findPopularPostsByHashtag(String hashtag, int minLikes);

// 3. Utenti che interagiscono con hashtag specifici
List<User> findUsersEngagingWithHashtags(List<String> hashtags);

// 4. Discussioni attive (post con molti commenti e risposte)
List<Post> findActiveDiscussions(int minComments, int minReplies);
```

Struttura del Progetto

```
assignment3/
├── src/main/java/com/socialmedia/
│   ├── entities/            # JPA Entities (6 classi)
│   │   ├── User.java (abstract)
│   │   ├── RegularUser.java
│   │   ├── AdminUser.java
│   │   ├── Post.java
│   │   ├── Comment.java
│   │   └── Hashtag.java
│   ├── dao/                 # Data Access Objects
│   │   ├── GenericDAO.java (base)
│   │   ├── UserDAO.java
│   │   ├── PostDAO.java
│   │   ├── CommentDAO.java
│   │   └── HashtagDAO.java
│   ├── service/             # Business Logic
│   │   └── SocialMediaService.java
│   ├── facade/              # Facade Pattern
│   │   └── SocialMediaFacade.java
│   └── main/                # Application Entry
│       └── MainApp.java
├── src/test/java/com/socialmedia/
│   └── SocialMediaTest.java (15+ test cases)
├── src/main/resources/
│   └── META-INF/
│       └── persistence.xml (JPA Configuration)
└── pom.xml (Maven Configuration)
```

Testing

Test Suite Completa

- 14 test cases organizzati in ordine logico
- Test di tutte le operazioni CRUD
- Test delle relazioni (following, likes, comments)
- Test delle ricerche complesse
- Test di integrazione con database H2 in-memory

Coverage dei Test

- Test 1-3: Creazione e ricerca utenti
- Test 4-6: Creazione post, commenti e risposte
- Test 7-8: Relazioni (following e likes)
- Test 9-12: Ricerche complesse
- Test 13: Update e Delete
- Test 14: Test completo di tutte le operazioni

Tecnologie Utilizzate

Java 11 JPA 2.2 Hibernate 5.6 H2 Database JUnit 5 Maven Git

Istruzioni per Eseguire

Prerequisiti

- Java JDK 11 o superiore
- Maven 3.6+
- Git

Passaggi

```
# 1. Clonare il repository
git clone
https://gitlab.com/f.saponara/2025_assignment3_social_media_group32

# 2. Navigare nella cartella
cd 2025_assignment3_social_media_group32

# 3. Compilare il progetto
mvn clean compile

# 4. Eseguire i test
mvn test

# 5. Eseguire l'applicazione
mvn exec:java -Dexec.mainClass="com.socialmedia.main.MainApp"
```

Note Tecniche

Configurazione JPA

```
<!-- persistence.xml -->
<persistence-unit name="socialmedia-pu">
    <properties>
        <property name="javax.persistence.jdbc.url"
                  value="jdbc:h2:mem:socialmedia"/>
        <property name="hibernate.hbm2ddl.auto" value="create-drop"/>
        <property name="hibernate.show_sql" value="true"/>
    </properties>
</persistence-unit>
```

Pattern Utilizzati

- **DAO Pattern** - Astrazione dell'accesso ai dati
- **Service Layer** - Separazione della logica di business
- **Facade Pattern** - Punto di ingresso unico semplificato
- **Inheritance Strategy** - JOINED per ottimizzazione database