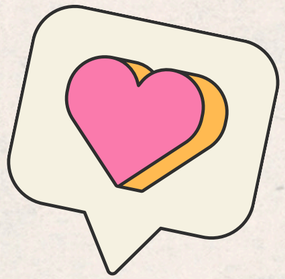
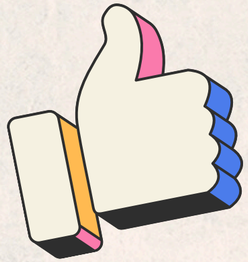
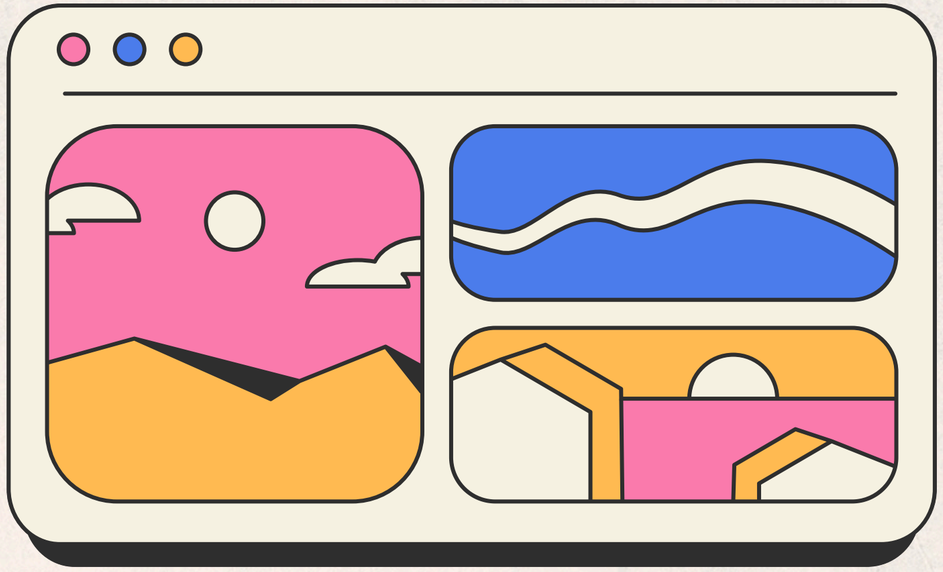


Hub Social

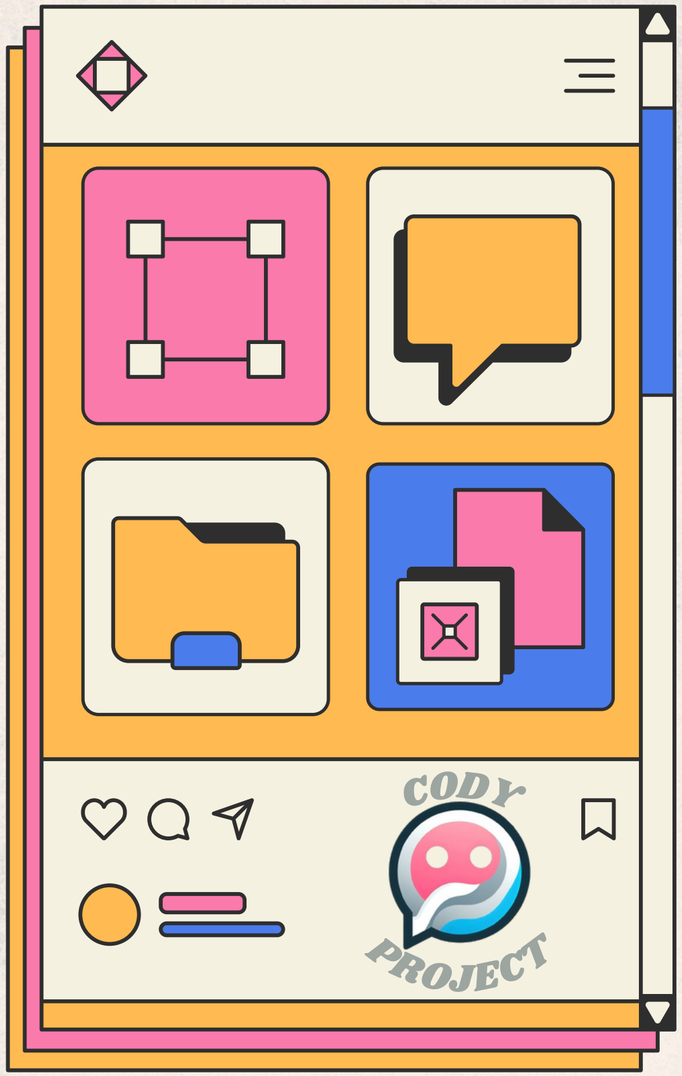
CoDy Project





CODY PROJECT

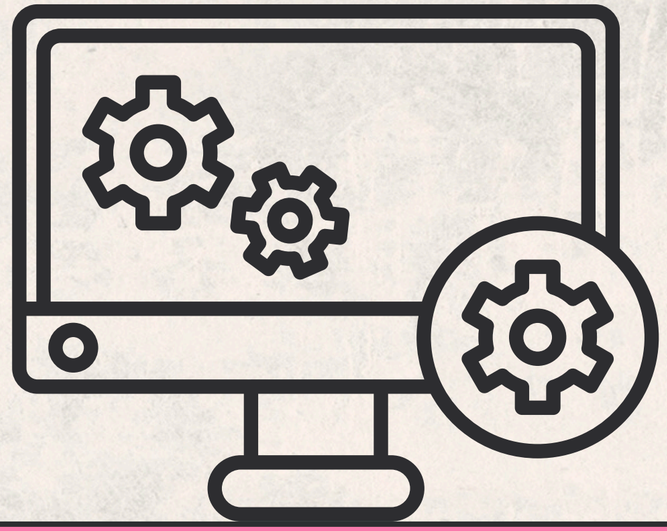
Gruppo B23:
Bruschi Corrado
Palomba Dylan

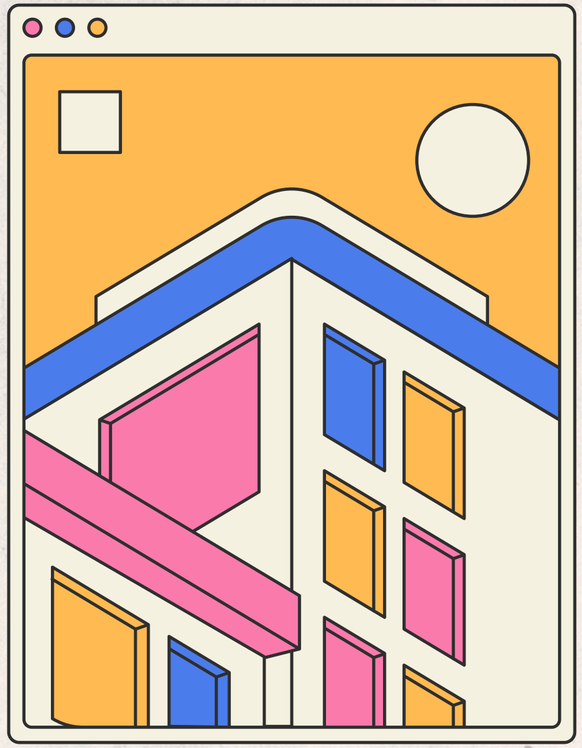


Introduzione

L’idea del progetto è quella di offrire un social network con le relative funzionalità base tramite una semplice interfaccia grafica:

- **Registrazione e login con i propri dati**
- **Scrivere post e commenti**
- **Interagire con gli altri utenti**
- **Visualizzare gli utenti e scegliere chi seguire**

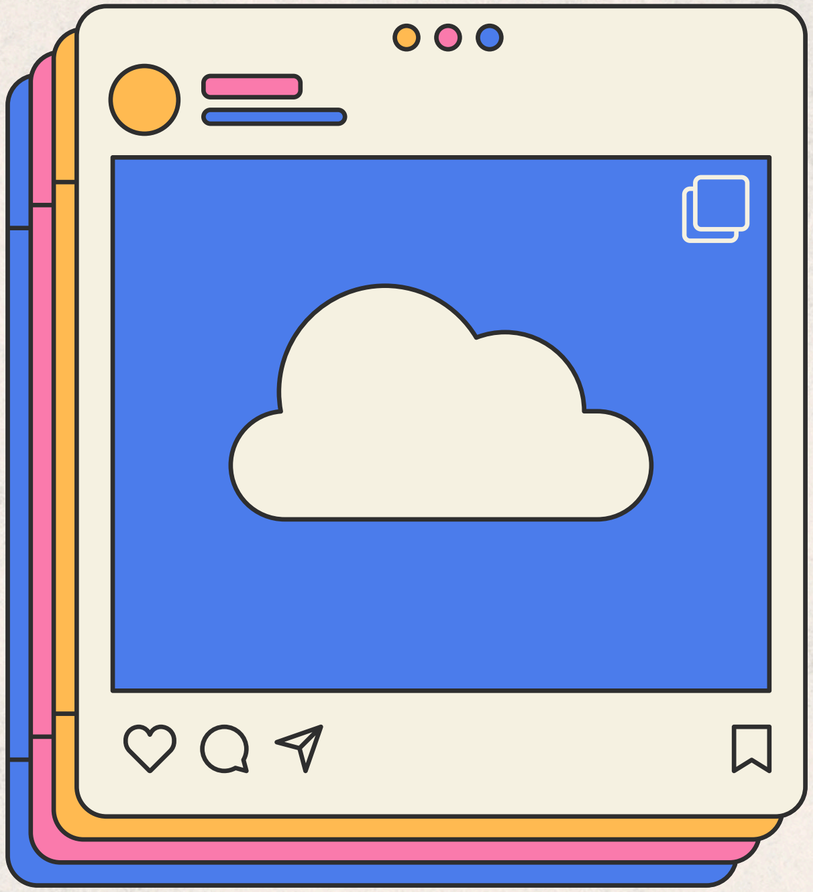




Dettagli implementativi

- **Linguaggi utilizzati: Java 23, SQL**
- **Database e server: MySQL**
- **Interfaccia grafica: Java Swing**
- **Librerie aggiuntive: JUnit e MySQL-connector**

CODY PROJECT



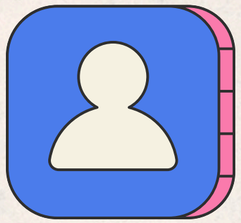
Unified Process (UP)

Metodo di sviluppo software:

- Iterativo, incrementale e adattabile
- Tiene conto dei rischi critici nelle fasi inziali
- Basato su UML
- Complesso per piccoli progetti
- Rischio di over-engineering
- Costi iniziali elevati prima di un prototipo pronto

CODY PROJECT

Fasi del progetto



Ideazione

- Definizione della visione del progetto e degli obiettivi principali.
- Raccolta e analisi iniziale dei requisiti.
- Valutazione della fattibilità e dei rischi.
- Output: documento di visione e casi d'uso principali

Elaborazione

- Progettazione dell'architettura base del sistema.
- Dettaglio dei requisiti principali.
- Riduzione dei rischi critici.
- Output: diagrammi UML e un modello architetturale.

Costruzione

- Sviluppo iterativo delle funzionalità.
- Integrazione e test del sistema.
- Output: versione del sistema parzialmente o completamente funzionante.

Transizione

- Consegna e distribuzione del prodotto agli utenti finali.
- Betatesting
- Formazione, supporto e gestione del feedback.
- Output: prodotto finale, manuali e report di valutazione.

Casi d’uso

Registrazione utente

Autenticazione

Creazione di un Post

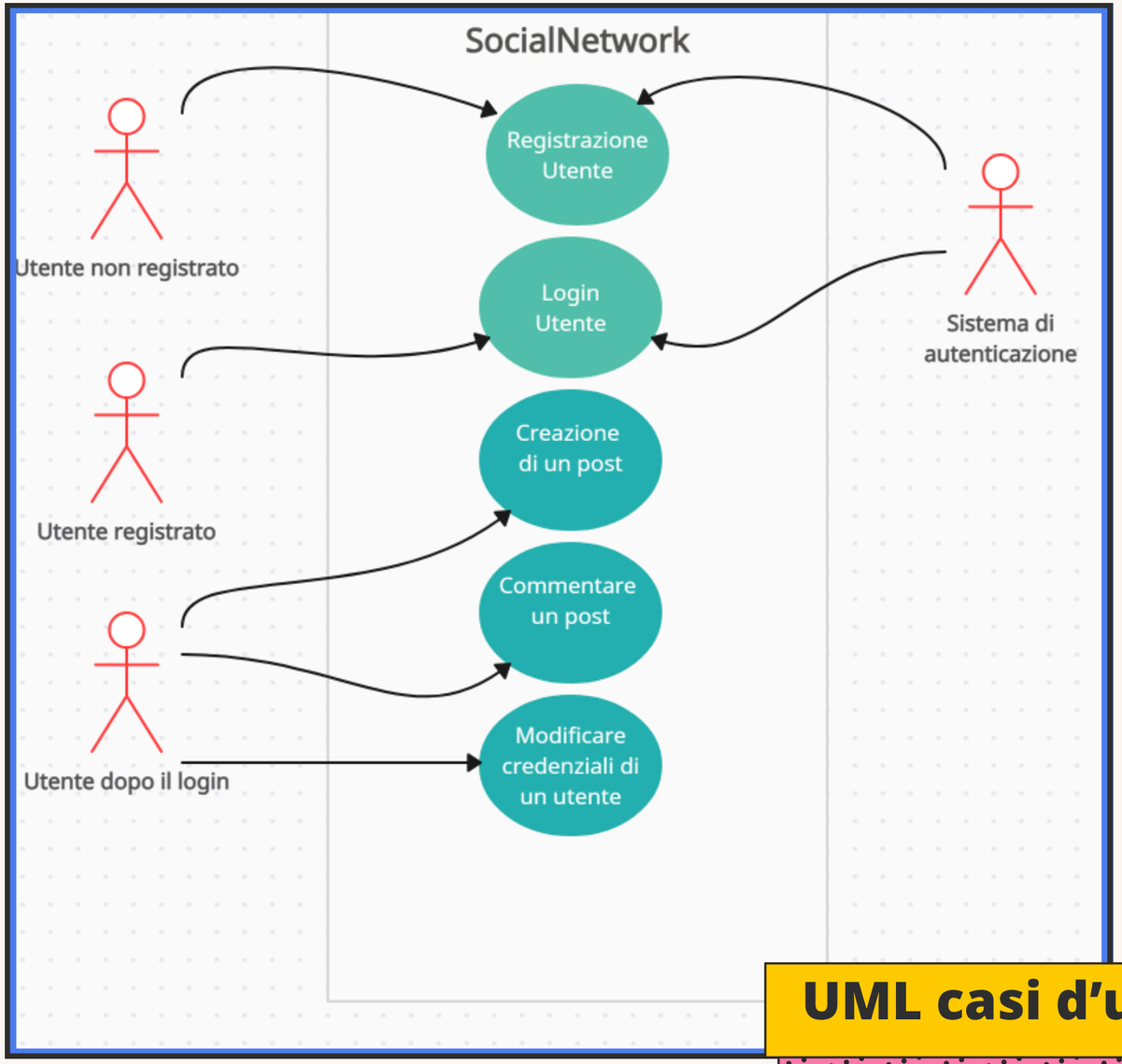
Commento di un post

Modifica credenziali

Requisiti funzionali

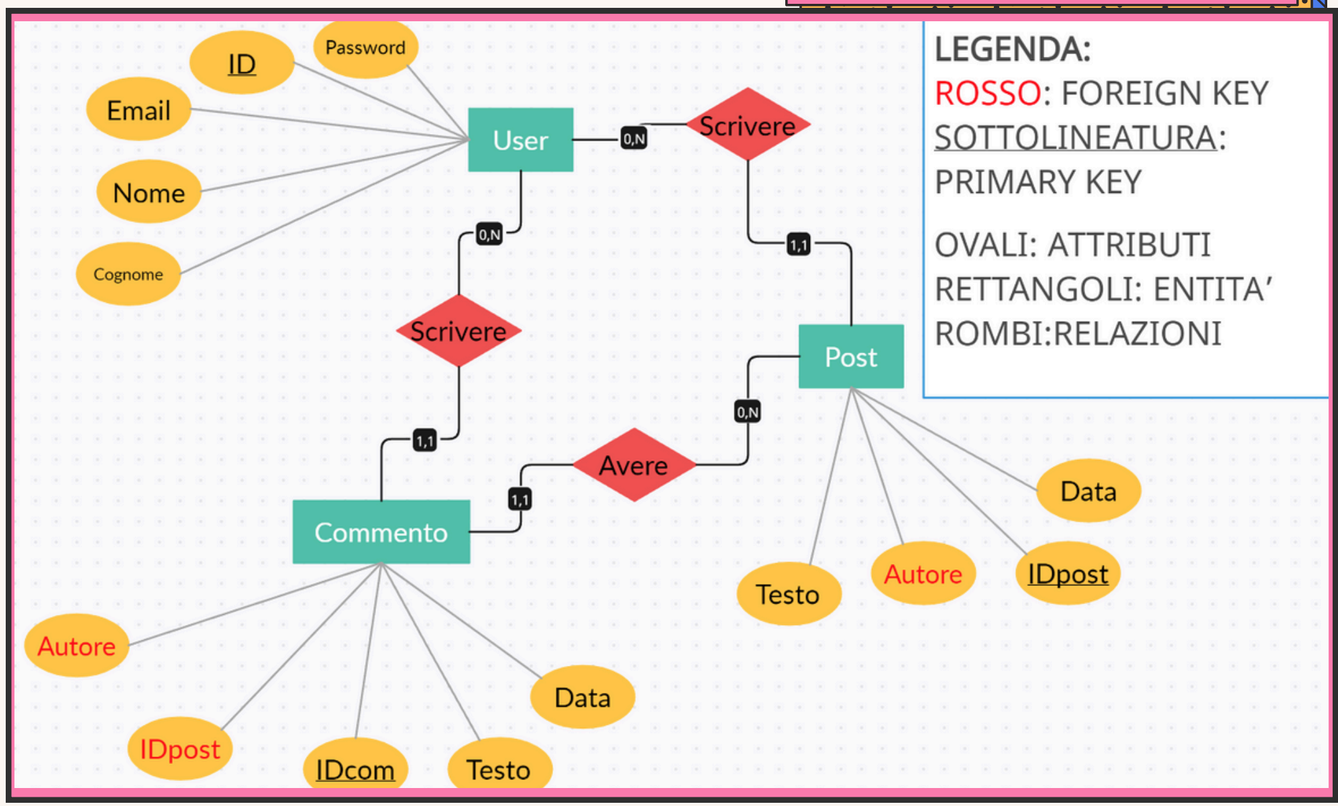
Requisiti non funzionali

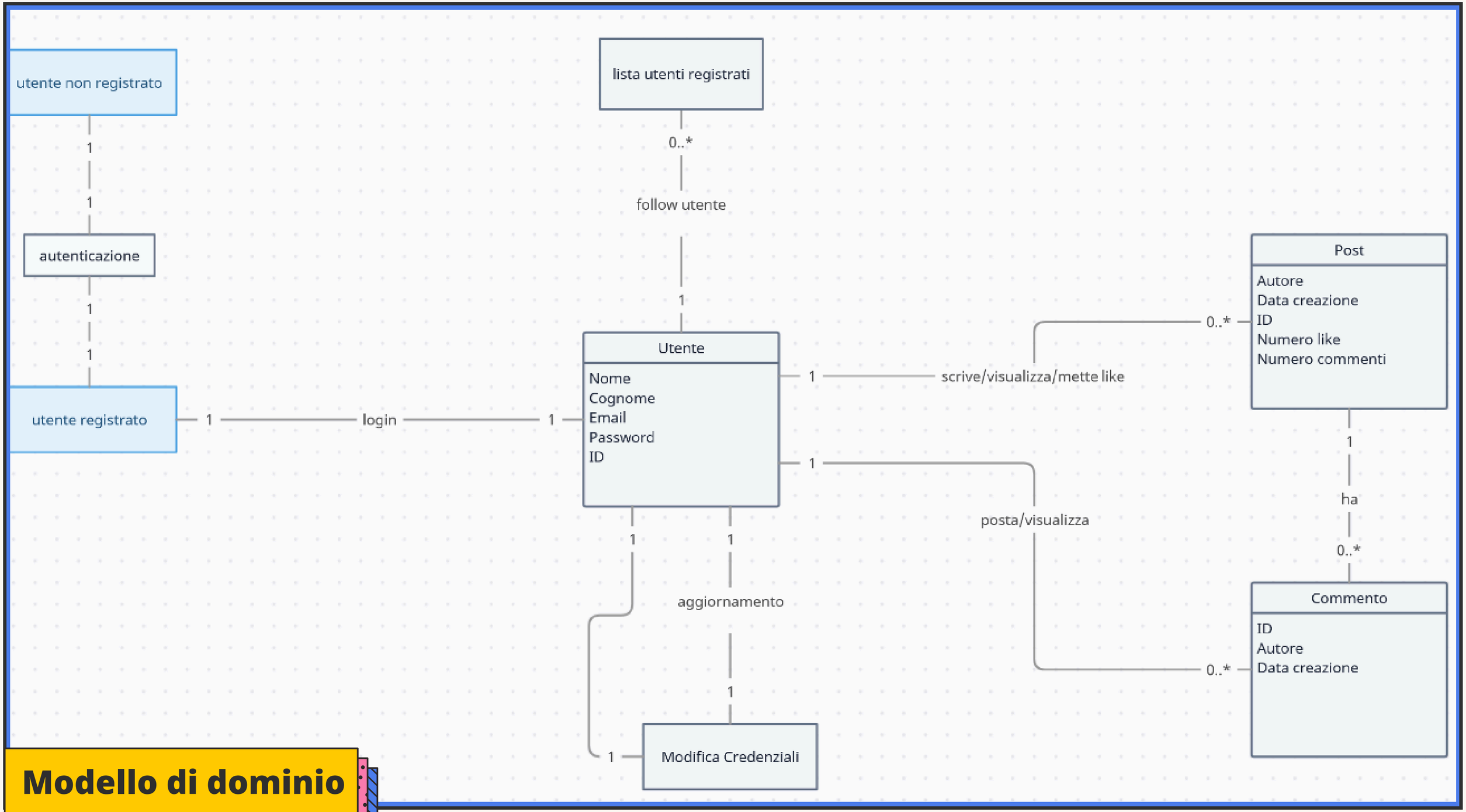
Elaborazione



UML casi d'uso

Diagramma ER





Modello di dominio

Pattern utilizzati

GoF

Observer

Abstract factory

- AbstractButton

Singleton

- DbConnectionSingleton

GRASP

Polymorphism

Low coupling & high cohesion

Controller

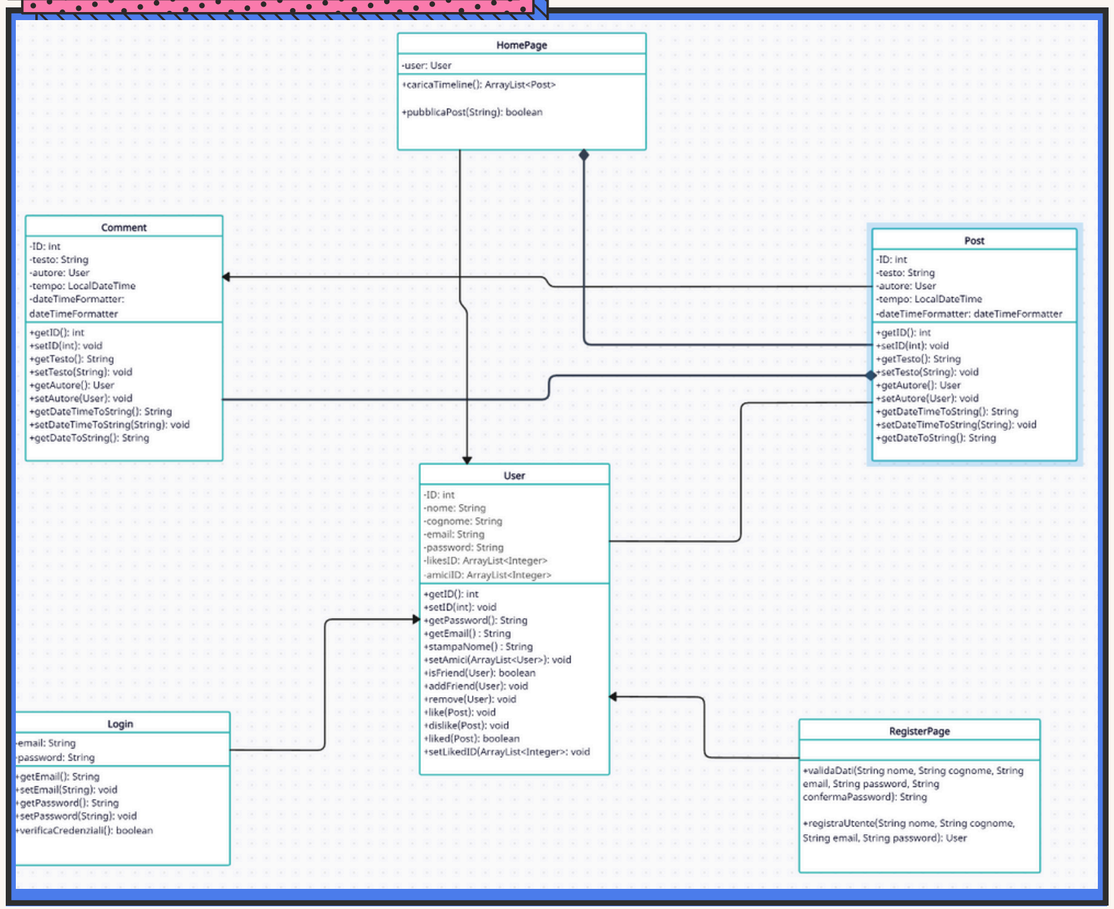
- ChangePasswordController
- CommentsTabController

Database

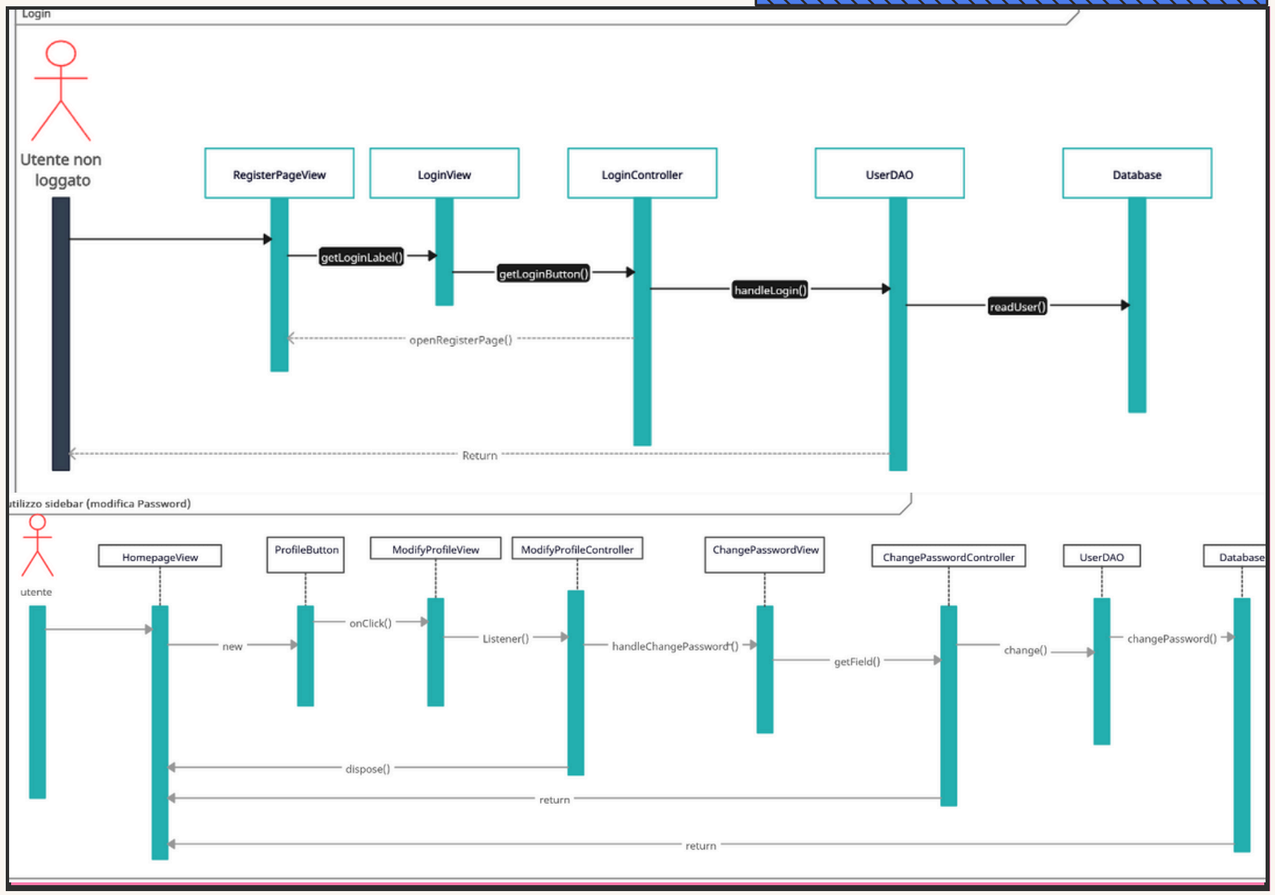
Pattern DAO

- FriendDAO
- LikeDAO
- PostDAO
- UserDAO

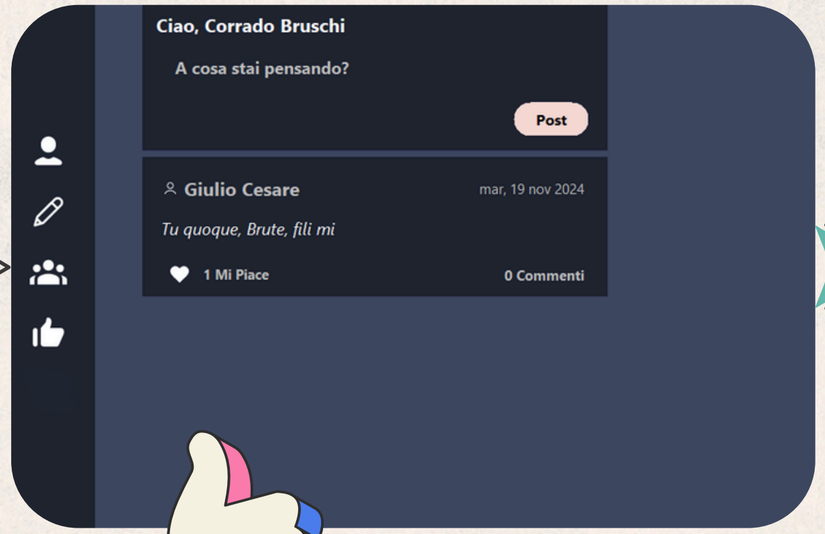
UML classi



sequenza



Alcune iterazioni



Testing

```java
class ReadUserTest {
    private Connection connection;  4 usages
    @BeforeEach
    public void setUp() throws SQLException {
        connection = DbConnectionSingleton.getInstance().getConnection();


        String createUser = "INSERT INTO users (ID, Nome, Cognome, Email, Password) VALUES (?, ?, ?, ?, ?)";
        try (PreparedStatement ps = connection.prepareStatement(createUser)) {
            ps.setInt( parameterIndex: 1,  x: 11111);
            ps.setString( parameterIndex: 2,  x: "Nome");
            ps.setString( parameterIndex: 3,  x: "Cognome");
            ps.setString( parameterIndex: 4,  x: "test@email.com");
            ps.setString( parameterIndex: 5,  x: "password123");
            ps.executeUpdate();
        }
        DbConnectionSingleton.getInstance().closeConnection();
    }

    @AfterEach
    public void tearDown() throws SQLException {
        connection = DbConnectionSingleton.getInstance().getConnection();
        String deleteUser = "DELETE FROM users WHERE Email = ?";
        try (PreparedStatement ps = connection.prepareStatement(deleteUser)) {
            ps.setString( parameterIndex: 1,  x: "test@email.com");
            ps.executeUpdate();
        }
        DbConnectionSingleton.getInstance().closeConnection();
    }

    @Test
    public void testLoginSuccess() {
        UserDAO readUser = new UserDAO();
        assertNotNull(readUser.readUser( email: "test@email.com",  password: "password123"),  message: "Il Login dovrebbe dare esito positivo");

        User user = new UserDAO().readUser( email: "test@email.com",  password: "password123");

        assertNotNull(user,  message: "l'utente non dovrebbe essere nullo dopo il login");
        assertEquals( expected: "Nome", user.getNome(),  message: "i nomi dovrebbero essere uguali");
        assertEquals( expected: "Cognome", user.getCognome(),  message: "i cognomi dovrebbero essere uguali");
        assertEquals( expected: "test@email.com", user.getEmail(),  message: "le mail dovrebbero combaciare");
    }

    @Test
    public void testLoginFailure() {
        UserDAO readUser = new UserDAO();
        assertNull(readUser.readUser( email: "test@emailsbagliata.com",  password: "password123"),  message: "Il login dovrebbe fallire per mail sbagliata");
        UserDAO readUserP = new UserDAO();
        assertNull(readUserP.readUser( email: "test@email.com",  password: "password321"),  message: "Il login dovrebbe fallire per password sbagliata");
    }

    @Test
    public void testCaseInsensitive() {
        UserDAO readUser = new UserDAO();
        assertNull(readUser.readUser( email: "TEST@EMAIL.COM",  password: "password123"),  message: "il Login dovrebbe dare esito negativo perchè email case sensitive");

        UserDAO readUserP = new UserDAO();
        assertNull(readUserP.readUser( email: "test@email.com",  password: "PASSWORD123"),  message: "il Login dovrebbe dare esito negativo perchè password case sensitive");
    }
```

Grazie per l’attenzione!