Biblioteca

Team (Name)	WEEOPLÀ

Team Members		
Name & Surname	**Matriculation Number**	**E-mail address**
Alessandro De Amicis	*242388*	*Alessandro.deamicis@student.univaq.it*
Primiano Medugno	*231674*	*primiano.medugno@student.univaq.it*
Dario Nitti	*236948*	*dario.nitti@student.univaq.it*

Requirements Collection

Functional Requirements

Il sistema:

1. Presenta due tipi di utenza: **Passiva e Attiva**. A seconda del tipo di utenza variano le funzionalità del sistema.
2. Offre agli utenti una **Graphics User Iterface** dove poter visualizzare tutte le pubblicazioni. (Passivo e attivo)
3. **Inserire o modificare** una pubblicazione. (attivo)
4. Può **Filtrare** le pubblicazioni a seconda di: (passivo e attivo)
    - Nome (A-Z)
    - Ultime aggiunte(1-30gg)
    - Utenti con più pubblicazioni
    - Disponibili Download
5. Offre l'opportunità di **Recensire** una pubblicazione e di aggiungere un **"mi piace"** alla pubblicazione. (passivo e attivo)
6. Potrà visualizzare i **log** delle modifiche di ogni pubblicazione. (attivo)
7. **Ricerca** il catalogo delle pubblicazioni usando i loro dati: (passivo e attivo)
    - ISBN
    - Titolo
    - Autore
8. Permette la **Registrazione** dell'utente attraverso la mail, che verrà usata come username, e una password. (passivo e attivo)
9. Offre la visualizzazione di tutti i dati di una pubblicazione specificando il suo ID. (passivo e attivo)
10. **Tiene traccia** delle modifiche apportate ad ogni pubblicazione segnando **data e ora.** (attivo)
11. **Approvazione** delle **recensioni** di una pubblicazione. (attivo)
12. **Modifica** il livello di un utente. (attivo)

    La modifica del livello di un utente viene fatta se un utente attivo ha bisogno di un altro utente per gestire la piattaforma

Use Case

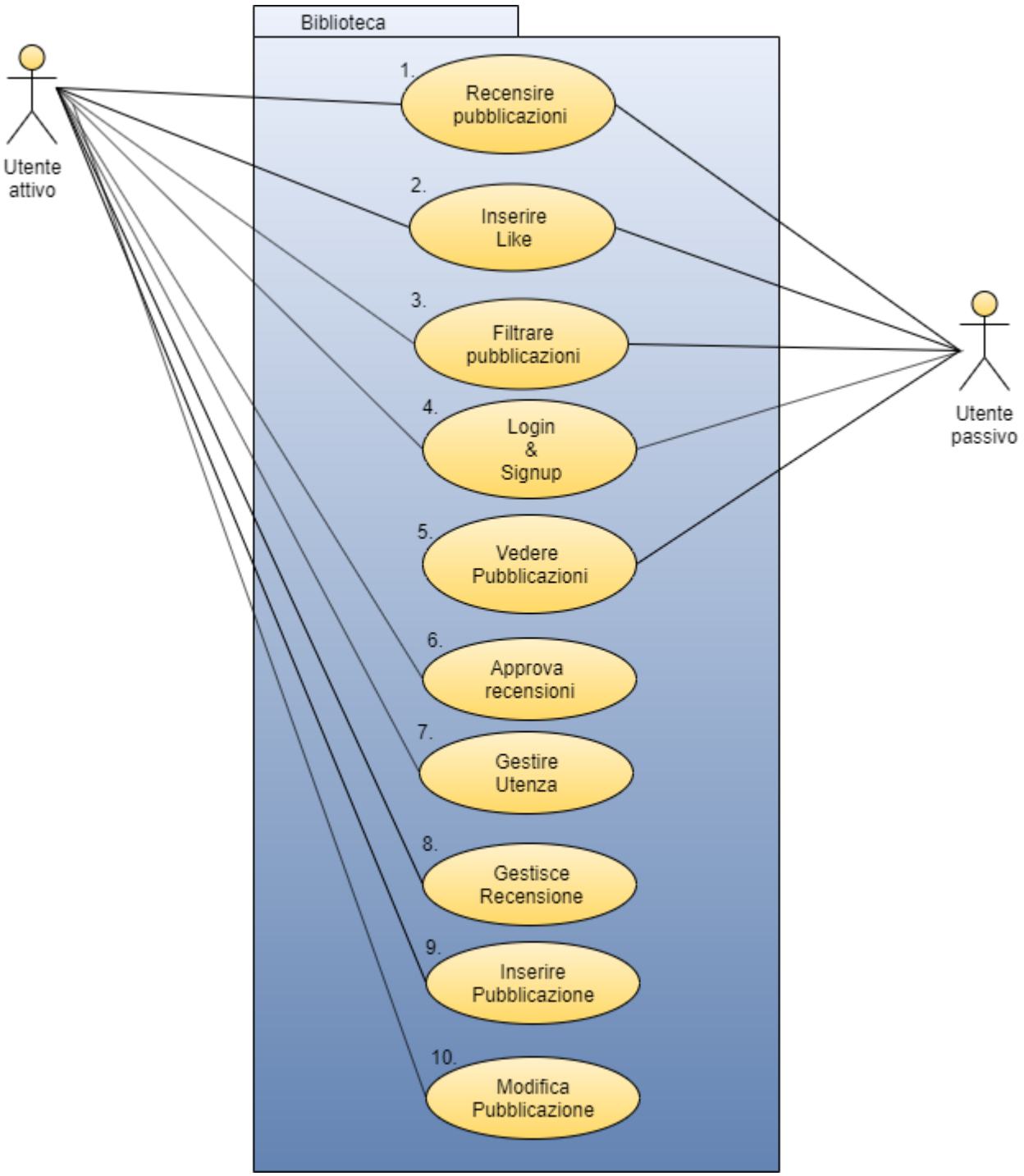


Figura 1 - Use Case Generale

Volendo mantenere un certo livello di leggibilità e sinteticità per dell'Use Case Diagram generale, il team ha deciso di prendere in considerazione due tipi di utenza.

- L'utente attivo (o moderatore) avrà il compito di approvare le recensioni fatte alle pubblicazioni, inserire una pubblicazione e modificare il livello di utenza degli utenti.

- Un utente Attivo può cambiare lo stato di un altro utente da passivo ad attivo, o viceversa.

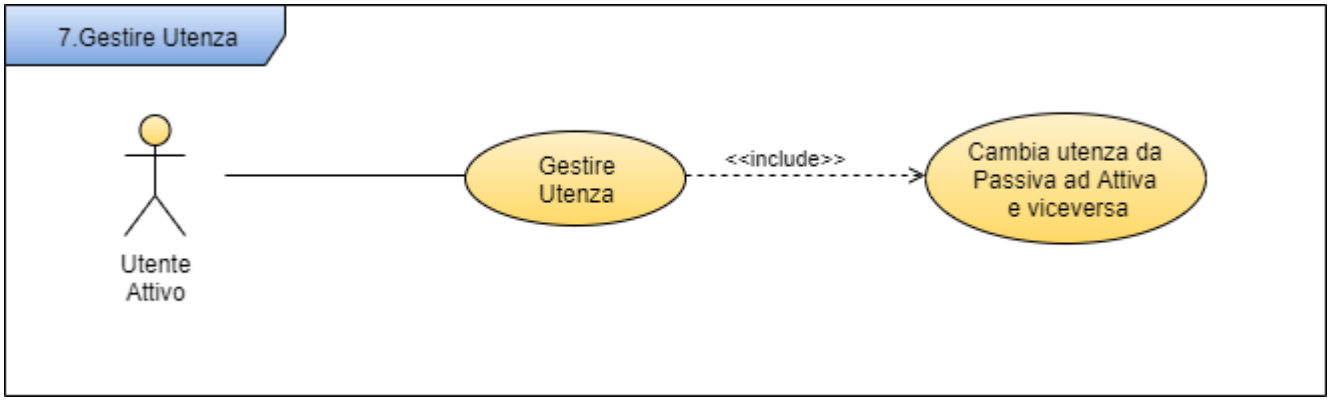


Figura 2 - Use Case Gestire Utenza

Il team ha anche deciso di rendere più leggibile lo Use Case Diagram raggruppando le opzioni di filtraggio delle pubblicazioni in un solo Use Case.

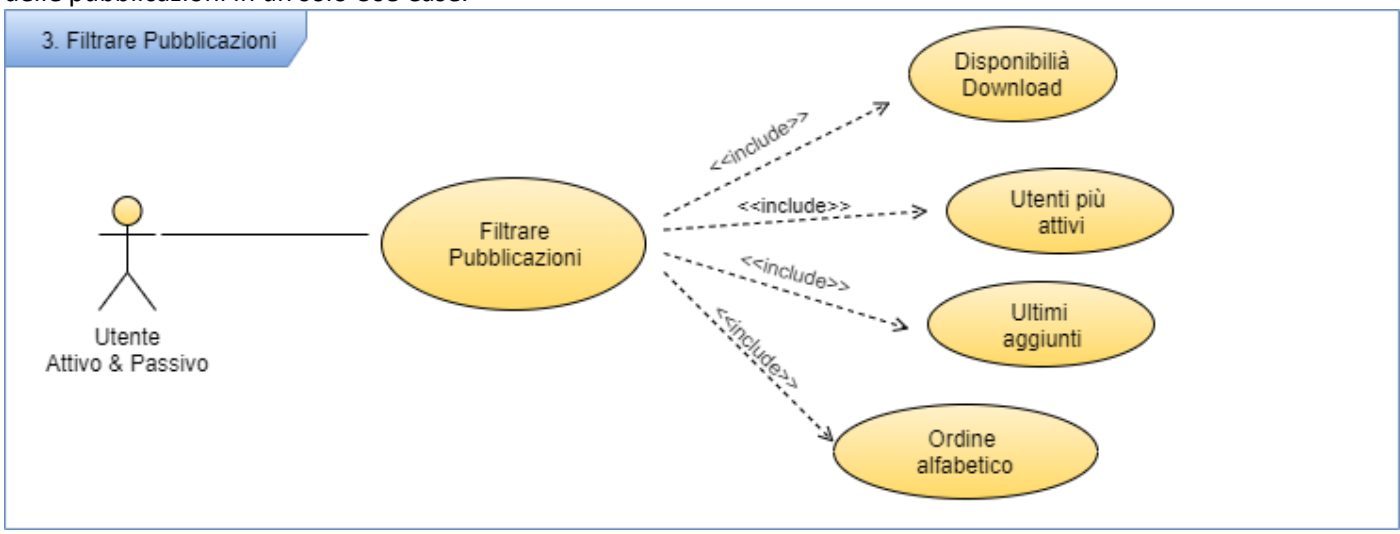


Figura 3 - Use Case Filtrare Pubblicazioni

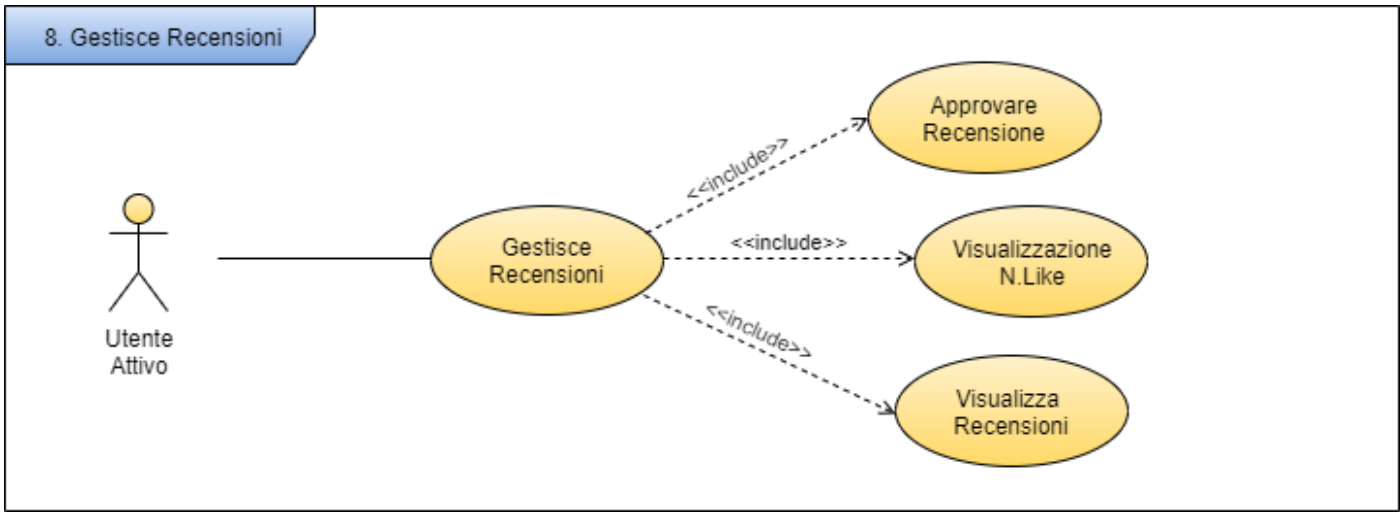


Figura 4 - Use Case Gestisce Recensioni

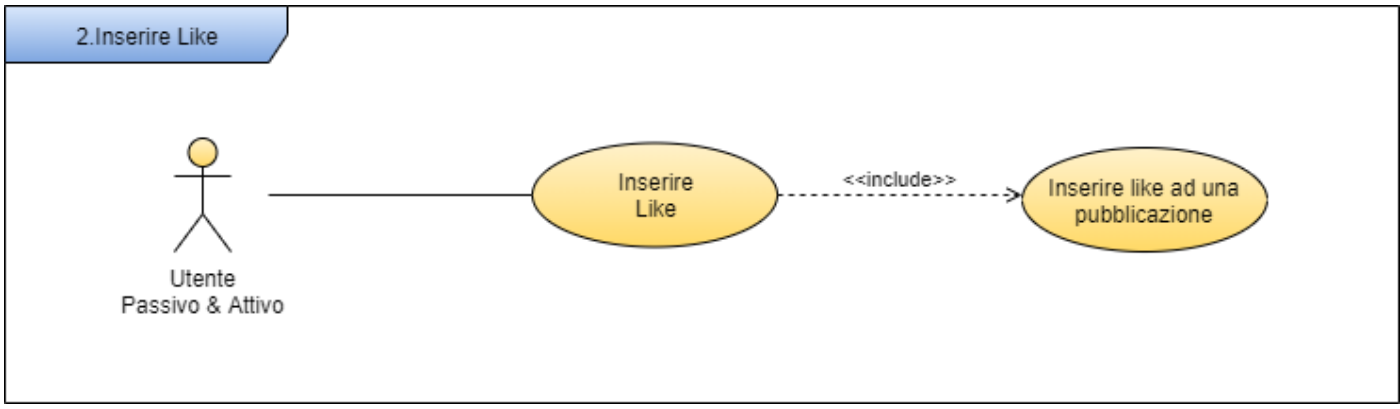


Figura 5 - Use Case Inserire Like

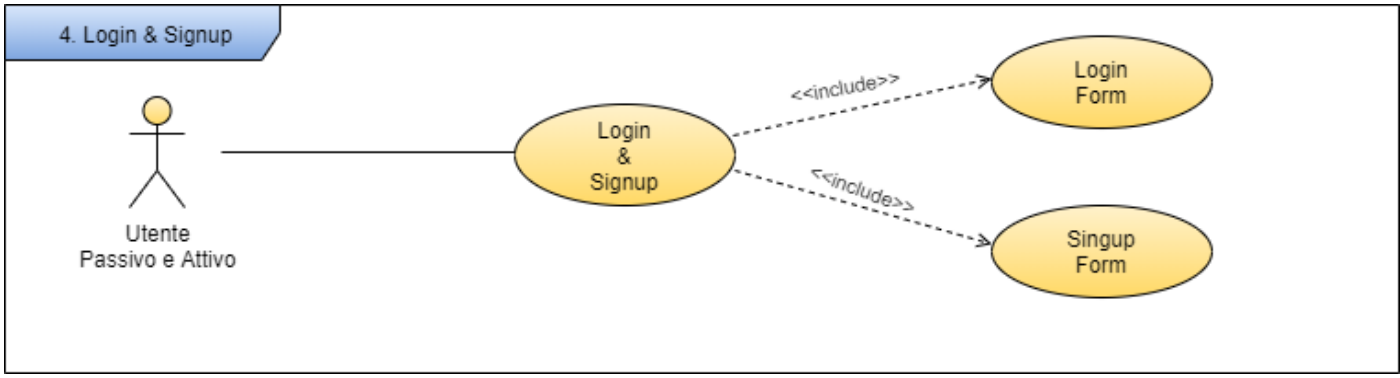


Figura 6 - Use Case Login & Signup

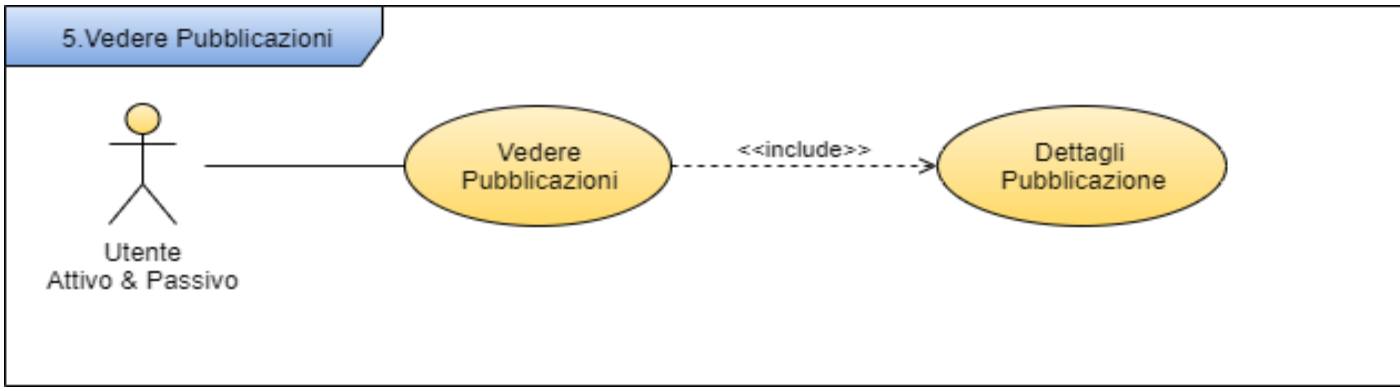


Figura 7 - Use Case Vedere Pubblicazioni

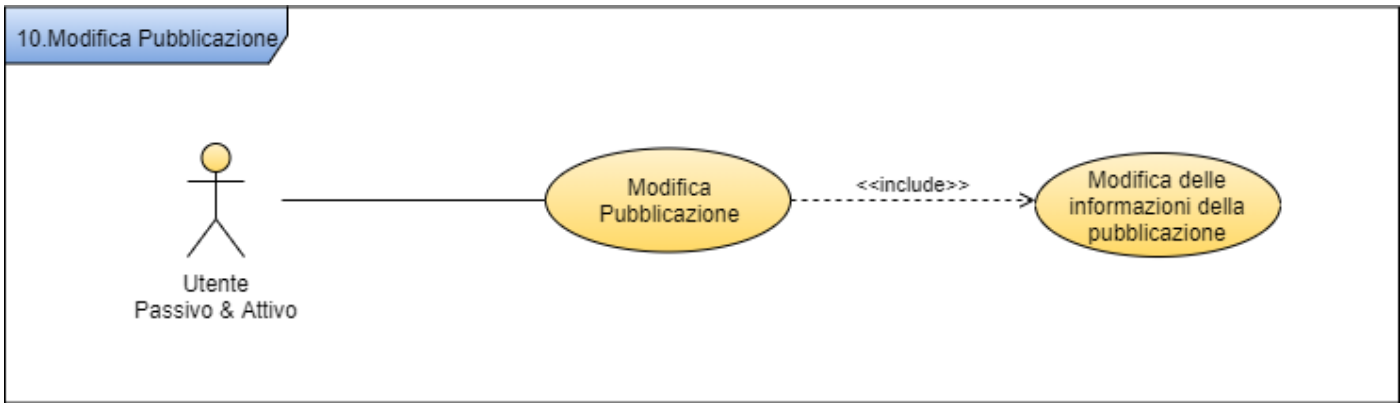


Figura 8 - Use Case Modifica Pubblicazioni

A seguire una descrizione tabellare (Cockburn) degli Use Case più significativi individuati dal team.

<table>
<tr><td>USE CASE 5</td><td>Vedere Pubblicazioni</td></tr>
<tr><td>GOAL IN CONTEXT</td><td>Si intente mostrare tutte le informazioni della pubblicazione.
Le informazioni saranno:
Isbn
Titolo
Editore
Data
Lingua
Pagine
Descrizione
Indice
Autore</td></tr>
<tr><td>SCOPE & LEVEL</td><td>Utente Attivo e Passivo, PRIMARY</td></tr>
<tr><td>PRECONDITION</td><td>L’utente le visualizza sotto forma di lista, una volta scelta la pubblicazione potrà vedere tutte le sue informazioni.</td></tr>
<tr><td>SUCCESS END CONDITION</td><td>Le informazioni vengono prese direttamente dal database.</td></tr>
<tr><td>FAILED END CONDITION</td><td>Nessuna pubblicazione</td></tr>
<tr><td>PRIMARY ACTOR/ACTORS</td><td>Utente Attivo e Passivo</td></tr>
<tr><td>TRIGGER</td><td>Disponibilità pubblicazioni</td></tr>
<tr><td>DESCRIPTION</td><td><table><tr><th>STEP</th><th>ACTION</th></tr><tr><td>1</td><td>Riceviamo la lista delle pubblicazioni</td></tr><tr><td>2</td><td>La mostriamo sulla bacheca principale</td></tr><tr><td>3</td><td>Una volta cliccata la pubblicazione l’utente può vedere le informazioni delle pubblicazioni.</td></tr></table></td></tr>
<tr><td>PRIORITY</td><td>Alta</td></tr>
<tr><td>FREQUENCY</td><td>Ogni volta che viene aggiornata la lista delle pubblicazioni</td></tr>
</table>

<table>
<tr><td>USE CASE 9</td><td>Inserire Pubblicazione</td></tr>
<tr><td>GOAL IN CONTEXT</td><td>Inserimento della pubblicazione viene fatto solo dall’utente attivo.
Ogni volta che viene inserita una pubblicazione vengono inserite anche tutte le informazioni:
Isbn
Titolo
Editore
Data
Lingua
Pagine
Descrizione
Indice
Autore</td></tr>
<tr><td>SCOPE & LEVEL</td><td>Utente Attivo, PRIMARY</td></tr>
</table>

PRECONDITION	L'utente nella bacheca principale può selezionare il bottone di inserimento che rimanderà al form di inserimento della pubblicazione dove riempirà tutti i campi delle informazioni.
SUCCESS END CONDITION	Le informazioni vengono inserite direttamente dal database.
FAILED END CONDITION	Nessuna informazione da inserire
PRIMARY ACTOR/ACTORS	**Utente Attivo**
TRIGGER	Disponibilità delle informazioni
DESCRIPTION	

STEP	ACTION
1	Premere il bottone di inserimento presente nella bacheca principale
2	Riempire i campi delle informazioni
3	Premere il bottone che inserirà tutte le informazioni e la pubblicazione nel database

PRIORITY	Alta
FREQUENCY	Ogni volta che si vuole aggiungere una pubblicazione

USE CASE 1	Recensire Pubblicazioni
GOAL IN CONTEXT	Ogni utente potrà recensire una pubblicazione con un piccolo testo.
SCOPE & LEVEL	**Utente Attivo & Passivo, PRIMARY**
PRECONDITION	Una volta visualizzata la pubblicazione (dalla lista) è possibile recensirla attraverso un tasto presente nei dettagli di ogni pubblicazione.
SUCCESS END CONDITION	Le informazioni vengono inserite direttamente dal database.
FAILED END CONDITION	Nessuna recensione da inserire / Nessuna pubblicazione presente nella lista
PRIMARY ACTOR/ACTORS	**Utente Attivo & Passivo**
TRIGGER	
DESCRIPTION	

STEP	ACTION
1	Scegliere la pubblicazione nella lista della bacheca principale
2	Una volta aperta la finestra delle informazioni della pubblicazione è possibile recensirla premendo il bottone "Inserisci Recensioni"
3	Riempire i campi del nome e cognome (dell'utente) e l'area di testo presente all'interno del form
4	Premere il bottone "Inserisci" per inserire la recensione nel database e questa verrà messa inizialmente in stato di attesa. In seguito, verrà approvata dall'utente attivo

PRIORITY	Alta
FREQUENCY	Ogni volta che si vuole aggiungere una recensione ad una pubblicazione

USE CASE 2	Inserire Like.
GOAL IN CONTEXT	Ogni utente potrà inserire un "like" ad una pubblicazione particolarmente piaciuta.
SCOPE & LEVEL	**Utente Attivo & Passivo, PRIMARY**

<table>
<tr><td>PRECONDITION</td><td>Una volta visualizzata la pubblicazione (dalla lista) è possibile inserire il “like” attraverso un tasto a forma di cuore presente nei dettagli di ogni pubblicazione.</td></tr>
<tr><td>SUCCESS END CONDITION</td><td>Le informazioni vengono inserite in una lista “like” presente nel database e poi recuperate e calcolate in seguito.</td></tr>
<tr><td>FAILED END CONDITION</td><td>Nessun “like” da inserire / Nessuna pubblicazione presente nella lista</td></tr>
<tr><td>PRIMARY ACTOR/ACTORS</td><td>Utente Attivo & Passivo</td></tr>
<tr><td>TRIGGER</td><td>Disponibilità della pubblicazione</td></tr>
<tr><td>DESCRIPTION</td><td>
<table>
<tr><th>STEP</th><th>ACTION</th></tr>
<tr><td>1</td><td>Scegliere la pubblicazione nella lista della bacheca principale</td></tr>
<tr><td>2</td><td>Una volta aperta la finestra delle informazioni della pubblicazione è possibile inserire il like premendo il bottone a forma di cuore</td></tr>
</table>
</td></tr>
<tr><td>PRIORITY</td><td>Alta</td></tr>
<tr><td>FREQUENCY</td><td>Ogni volta che si vuole aggiungere un “like” ad una pubblicazione</td></tr>
</table>

Non Functional Requirements

Il sistema dovrà essere:

1. Manutenibilità

   Perseguita in primis applicando la separation of concerns per produrre un sistema modulare e favorirne futuri aggiornamenti. Ogni parte del sistema è incapsulato in routines facilmente modificabili o totalmente rimpiazzabili senza necessità di ulteriori modifiche, nel caso in cui cambi il ciclo lavorativo del sistema.
   Per far sì che questo accada ci sarà una leggibilità e chiarezza elevata del codice.

2. Usabilità

   Il sistema sarà facilmente utilizzabile per gli utenti esperti e non attraverso un’interfaccia User-Friendly. Con forme minimali e non complesse l’utente potrà svolgere tutte le funzioni del sistema. Ogni azione è seguita da un messaggio che comunica all’utente se l’operazione fatta è andata a buon fine.

   Ex: Ogni volta che un utente proverà ad autenticarsi con e-

   +mail o password errati apparirà un message dialog: “Password e Email errati!”

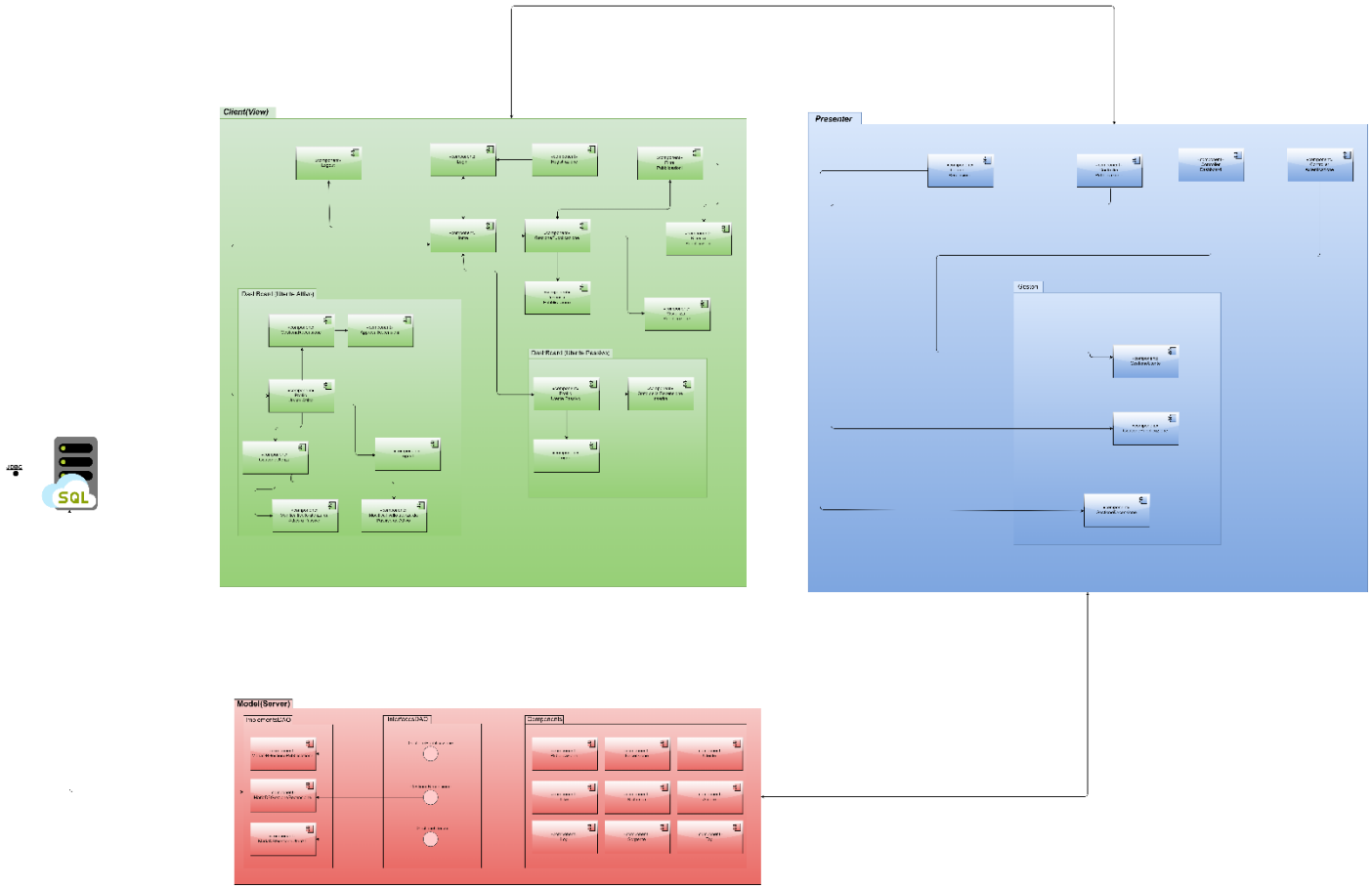
3. Security

   Il sistema applicherà una crittografia SHA256 su ogni password dell’utente. Questo permetterà di contrastare attacchi brute force esterni creando un hash per ogni password.
   Ogni volta che l’utente farà il login sulla piattaforma verranno confrontati gli hash della password immessa nel campo di login e quella presente nel database, e se ci sarà un matching tra le due l’utente potrà accedere alla piattaforma.

System Architecture

<Report here both the static and the dynamic view of your system design, in terms of a Component Diagram, Class Diagrams>

The static view of the system: Component Diagram



Le macro-componenti che suddividono la nostra architettura, sono le seguenti:

- **SQL (Data Storage):** il sistema dovrà includere “database”, ovvero una component che ha il compito di **immagazzinare i dati**;

- **Client (View):** conterrà **due packages** chiamati rispettivamente “Dashboard (Utente Attivo)” e “Dashboard (Utente Passivo)”, che al loro interno avranno come componenti:
  “Profilo Utente Attivo”, “GestioneRecensione”, “ApprovaRecensione”,”Logout”,”GestioneUtenza”, ”Modifica livello di utenza da Attivo a Passivo”, “Modifica livello di utenza da Passivo ad Attivo”, “Profilo Utente Passivo”, “Stato Recensione Inserita”. Questo macro-blocco ha il compito di **interfacciare** l’utente con il sistema (View). **“Dashboard (Utente Attivo)”** rappresenta l’interfaccia grafica tra l’Utente Attivo del sistema ed il sistema stesso, mentre la **“Dashboard (Utente Passivo)”** rappresenta l’interfaccia grafica tra l’Utente Passivo ed il sistema Biblioteca.
  Il component “Login” svolge un ruolo importante, ovvero ha il compito di identificare un determinato utilizzatore (attivo o passivo) del sistema che effettua l’accesso. Il component “Registrazione” ha il compito di registrare un nuovo utente (passivo) all’interno del sistema. Il team ha deciso di strutturare il package “View” in due subpackages chiamati rispettivamente “DashboardUtenteAttivo” e “DashboardUtentePassivo” in modo tale da distinguere tutte le componenti che riguardano la figura Utente Attivo dalla figura Utente Passivo. Le componenti “Registrazione”, “Login” non sono situate all’interno dei due subpackages in quanto sono componenti che offrono servizi ad entrambe le figure di utenza che usufruiranno del sistema.

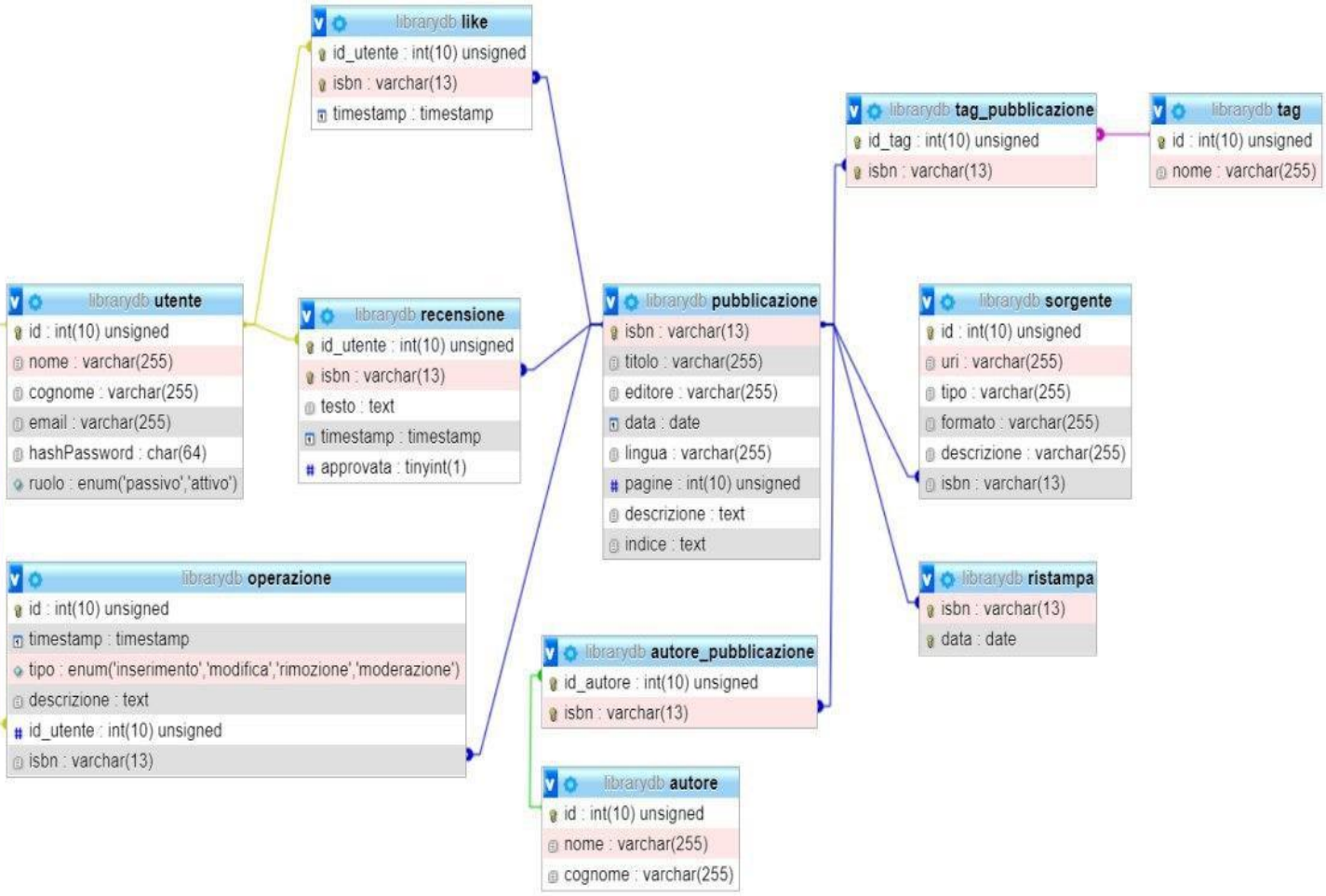
- **Presenter:** è una macro-componente che ha il compito di elaborare le richieste in ingresso, gestire gli input e le interazioni del singolo utente passivo o utente attivo, in base ai requisiti funzionali, per eseguire la logica del sistema appropriato. Al suo interno troveremo quattro componenti denominate rispettivamente: “PresenterRecensione”, “PresenterPubblicazione”, “PresenterDashboard”, “PresenterAutenticazione”. Che richiamano rispettivamente le classi definite nel subpackage Controller **Gestori** denominati rispettivamente: “GestioneRecensione”, “GestionePubblicaizone”, “GestioneUtente”.

- **Model (Server):** è una macro-componente che fornisce i metodi per accedere ai dati utili all’applicazione presenti all’interno del Database. Il model non si occupa soltanto dell’accesso fisico ai dati, ma anche di **creare il necessario livello di astrazione** tra il formato in cui i dati sono memorizzati ed il formato in cui i livelli di Controller e View si aspettano di riceverli. Oltretutto fornisce un’interpretazione intermedia dei dati arricchendo il database con nuove informazioni. Cosa importante, il Model non contiene direttamente i dati del Database, ma ha solo il compito di **fornire i metodi** e di restituire i dati al Controller. La macro-componente Server conterrà tre packages chiamati rispettivamente **“InterfacesDAO”, “ImplementsDAO”** e **“Components**”. Le Components presenti all’interno del package **“Components”** sono:
  “Pubblicazione” una classe che permette di istanziare oggetti di tipo Pubblicazione con i relativi metodi di get e set;
  “Like” una classe dov’è presente un contatore dei like di ogni pubblicazione;
  “Log” una classe che permette di istanziare oggetti di tipo Log con i relativi metodi di get e set, questo oggetto consente di visualizzare la data e la descrizione della modifica apportata ad una pubblicazione;
  “Recensione” una classe che permette di istanziare oggetti di tipo Recensione con i relativi metodi di get e set;
  “Ristampa” una classe che fa parte della descrizione dell’oggetto Pubblicazione;
  “Sorgente” una classe che è collegata con la Pubblicazione dove sono presenti gli Uri;
  “Utente” una classe che permette di istanziare oggetti di tipo Utente con i relativi metodi di get e set, serve per la parte di Login e Registrazione;
  “Autore” una classe che permette di istanziare oggetti di tipo Autore con i relativi metodi di get e set, questa classe è collegata alla descrizione della pubblicazione;
  “Tag” una classe che permette di istanziare oggetti di tipo Parola Chiave con i relativi metodi di get e set, questa classe ha al suo interno una parola chiave che contraddistingue ogni pubblicazione.
  Nel subpackages “**ImplementsDAO**” sono presenti: “MariaDBGestionePubblicazione” che implementa l’interfaccia “GestionePubblicazione”, “MariaDBGestioneRecensione” che implementa l’interfaccia “GestioneRecensione”, “MariaDBGestioneUtenza” che implementa l’interfaccia “GestioneUtenza”. Le **interfacce** sono presenti all’interno di un subpackage denominato “**InterfacesDAO**”. Le

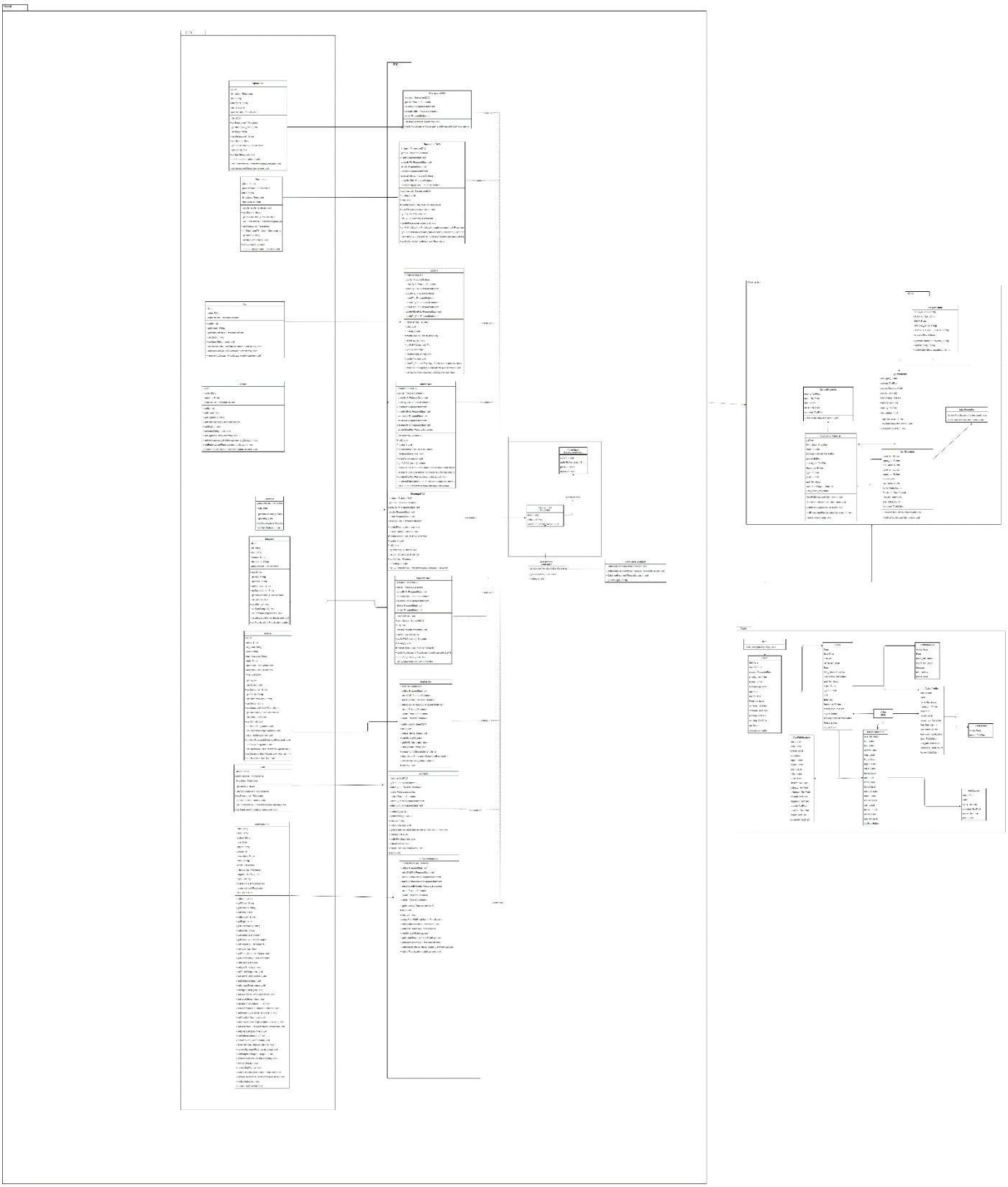
classi presenti nel subpackage “**ImplementsDAO**” hanno il compito di accedere, modificare, inserire e cancellare dati presenti nel Database.

ER Design

<Report here the Entity Relationship Diagram of the system DB>



Class Diagram of the implemented System



Descrizione del Class Diagram:

Il class diagram presenta 3 macro-package, che rispecchiano l'architettura presente nel sistema (MVP)

Nella **view** sono presenti i file dell'interfaccia grafica, attraverso la quale l'utente potrà interfacciarsi con il sistema. Il package è composto da file fxml, questi sono le singole finestre dell'interfaccia grafica.
Login: dove l'utente potrà accedere o registrarsi al sistema.
Home: dove l'utente potrà visualizzare le pubblicazioni e filtrarle. Inoltre, nella finestra Home saranno presenti dei bottoni: Logout che permetterà all'utente di uscire dal sistema, e il bottone del profilo attraverso il quale l'utente potrà accedere al suo profilo personale.
ActiveProfile: profilo dell'utente attivo, al cui interno sono presenti le recensioni degli utenti passivi che devono essere approvate e una lista con gli utenti passivi, con all'interno l'opzione di cambiare il tipo di utenza.
ProfilePassive: profilo dell'utente passivo, al cui interno sono presenti le recensioni fatte da quest'ultimo.
AddPubblication: dove l'utente(attivo) potrà inserire una nuova pubblicazione attraverso dei campi.
DetailsPublication: dove l'utente potrà vedere tutte le informazioni della pubblicazione.
AddReview: dove l'utente potrà aggiungere una recensione alla pubblicazione.
ViewReview: questa finestra permette all'utente attivo di controllare l'intera recensione degli utenti.

Nel **presenter** sono presenti le classi che controllano sia l'interfaccia grafica che il model.
Ogni presenter si riferisce al dato da gestire.
LoginPresenter: si riferisce alla gestione dell'accesso e della registrazione dell'utente.
LikePresenter:si riferisce alla gestione dei like delle pubblicazioni.
PublicationPresenter: si riferisce alla gestione della pubblicazione e tutto cioò che la riguarda, tipo il suo filtraggio.
ReviewPresenter: si riferisce alla gestione delle recensioni scritte dagli utenti.
UserPresenter: si riferisce alla gestione dell'utente e in particolare alla gestione del suo profilo, sia attivo che passivo.
Nel sotto-pacchetto utils è presente la classe **PasswordUtility** , questa si riferisce alla crittografia in SHA256 della password dell'utente

Nel **Model** è presente la rappresentazione dei dati.
Il Datalayer è una classe al cui interno è presente la connessione al database e la sua istanza.
Composto da tre sotto-pacchetti:
dao: dove sono presenti BaseDAO e DataAccessObject
impl: dove sono presenti le classi che effettuano le operazioni al database(query)

- AutoreDAO
- LikeDAO
- OperazioneDAO
- PubblicazioneDAO
- RecensioneDAO
- RistampaDAO
- SorgenteDAO

- TagDAO
- UtenteDAO

dto: dove sono presenti le classi di ogni singolo elemento del database, ovvero gli oggetti.

- Autore
- Like
- Operazione
- Pubblicazione
- Recensione
- Ristampa
- Sorgente
- Tag
- Utente

Design Decisions

Descrizione delle scelte e strategie adottate compresi Pattern Architetturali/Design

---

Linguaggio di programmazione: Java

Scelta dovuta per la sua portabilità ed in particolare, si è deciso di usare Java SE e il framework di widget JavaFx per quel che riguarda la GUI.
Il team ha scelto di usare JavaFx poiché è una tecnologia recente che si adatta facilmente ai pattern architetturali più conosciuti e ha delle componenti ricche di funzionalità rispetto ad altre tecnologie. La scelta di questo framework comporta anche le proprietà di Binding. È stato preso in considerazione un High-Level Binding nella parte di registrazione dell'utenza, quest'ultimo sarà unidirezionale poiché verranno controllati i campi ca compilare per la registrazione, e se il campo e-mail e il campo password avrà uguale valore (stesse parole) ci sarà un messaggio di errore.

Database: MariaDB

Tra i più popolari open-source relational database management system (RDBMS), basato quindi sul Modello Relazionale che offre disponibilità di documentazione, scalabilità e portabilità. Il team ha preferito usare questo tipo di database poiché avendo usato il software XAMPP era presente al suo interno e a differenza di altri database utilizza un linguaggio di query standard e popolare

Pattern Architetturale: MVP (Model-View-Presenter)

È stato scelto questo pattern architetturale poiché permette di separare la vista dalla logica dei dati, in modo che tutto ciò che riguarda l'interfaccia sia separato da come deve essere rappresentato sullo schermo.
MVP è simile al MVC, la principale differenza è che il Presenter gestisce la logica tra la View e il Model. MVP e MVC condividono la separazione tra la UI, i controller e la base date (i Model), ma l'implementazione del Presenter permette di gestire la UI e soprattutto il collegamento tra UI e base dati in maniera più efficiente.
Separare la UI dalla logica non è sempre immediato ma il pattern Model-View-Presenter rende questo compito più facile evitando di creare classi che si occupano sia dei dati che delle interfacce, classi che spesso superano il migliaio di righe di codice.
Specificando ogni parte avremo:

Model

Fornisce i metodi di accesso ai dati definendo le regole di business per l'accesso agli stessi esponendo lo stato dell'applicazione alla componente View e Presenter.
Il team ha deciso di implementare i design patterns **Singleton, Data Access Object (DAO), Data Transfer Object (DTO)** e la realizzazione di un Data Access Layer per la gestione della persistenza, la stratificazione e l'isolazione dell'accesso ai dati.
È stato preso in considerazione il pattern **Singleton** in versione standard unito al pattern Data Access Object. Questo pattern è risultato idoneo al sistema poiché era necessario controllare l'accesso simultaneo a una risorsa condivisa e poiché, a livello concettuale, è stato appurato che serviva un'unica istanza di quelle classi ed è stato deciso di renderla unica e accessibile.

```
// Singleton
  private UtenteDAO () {...} // impedisce l'instanziazione dall'esterno
  private static UtenteDAO instance = new UtenteDAO();
  public static UtenteDAO getInstance() {
    return instance;
  }
```

Gli altri pattern sono costituiti da:

-Datalayer: Classe dove risiede la connessione al database.

```java
public abstract class DataLayer {
  private static final MariaDbPoolDataSource dataSource = new MariaDbPoolDataSource("jdbc:mariadb://localhost:3306/librarydb?user=root&maxPoolSize=10");

  /**
   * Recupero una connessione dal DataSource tramite Connection Pool
   * @return Una connessione al DB
   * @throws SQLException in caso di errori
   */
  public static Connection getConnection () throws SQLException {
    return dataSource.getConnection();
  }

  // Chiudo il DataSource
  public static void destroy() {
    dataSource.close();
  }
}
```

- DAO Interface: Definisce le operazioni standard da eseguire su un Data Transfer Object.

```java
public interface UtenteDAO {

  Utente getByPK (int id);
  List<Utente> getAll ();
  boolean execInsert (Utente u);
  boolean execUpdate (Utente u);
  boolean execDelete (int id);
}
```

- DAO Implementation: Classe che implementa la precedente interfaccia ed è responsabile del recupero di informazioni da una "data source" come può esserlo un database.

```java
  /**
   * INSERT che aggiunge un Utente nel DB
   *
   * @param utente Utente da aggiungere
   * @return true in caso di successo, false altrimenti
   */
  @Override
  public boolean execInsert (Utente utente) {
    try (Connection connection = DataLayerMariaDB.getInstance().getConnection();
         PreparedStatement statement = connection.prepareStatement("INSERT INTO utente (nome, cognome, email, hashPassword) VALUES (?, ?, ?, ?)")) {

      statement.setString(1, utente.getNome());
      statement.setString(2, utente.getCognome());
      statement.setString(3, utente.getEmail());
      statement.setString(4, utente.getHashPassword());

      if (statement.executeUpdate() == 1) return true;
    } catch (SQLException e) {
      e.printStackTrace(); //TODO eccezione
    }
    return false;
  }
```

- Data Transfer Object: Oggetto che offre i metodi get/set per manipolare le informazioni ottenute tramite le classi DAO.

```java
public class Utente {
  private int id;
  private String nome;
  private String cognome;
  private String email;
  private String hashPassword;
  private String ruolo;

  public Utente () {}
  public Utente (int id, String nome, String cognome, String email, String hashPassword, String ruolo) {
    this.id = id;
    this.nome = nome;
    this.cognome = cognome;
    this.email = email;
    this.hashPassword = hashPassword;
    this.ruolo = ruolo;
  }

  public int getId () {
    return id;
  }

  public void setId (int id) {
    this.id = id;
  }

  public String getNome () {
    return nome;
  }

  public void setNome (String nome) {
    this.nome = nome;
  }

  public String getCognome () {
    return cognome;
  }

  public void setCognome (String cognome) {
    this.cognome = cognome;
  }

  public String getEmail () {
    return email;
  }

  public void setEmail (String email) {
    this.email = email;
  }

  public String getHashPassword () {
    return hashPassword;
  }

  public void setHashPassword (String hashPassword) {
    this.hashPassword = hashPassword;
  }

  public String getRuolo () {
    return ruolo;
  }

  public void setRuolo (String ruolo) {
    this.ruolo = ruolo;
  }
}
```

View

Implementata con JavaFx conterrà un riferimento del Presenter in genere inviato attraverso un Dependency Injector oppure utilizzando altri metodi che permettono di creare un riferimento al Presenter. L' unica cosa che la View dovrà fare è richiamare un metodo del Presenter ogni volta che ci sarà un'interazione tra l'interfaccia (per esempio un clic su un Button) e l'accesso alla base dati.

```
<AnchorPane prefHeight="400.0" prefWidth="600.0" xmlns="http://javafx.com/javafx/8.0.112" xmlns:fx="http://javafx.com/fxml/1" fx:controller="controller.core.ControlAutenticazione">
    <children>
        <SplitPane dividerPositions="0.5117056856187291" layoutY="6.0" prefHeight="394.0" prefWidth="600.0">
            <items>
                <AnchorPane minHeight="0.0" minWidth="0.0" prefHeight="160.0" prefWidth="100.0">
                    <children>
                        <Label layoutX="105.0" layoutY="14.0" prefHeight="21.0" prefWidth="66.0" text="Accedi">
                            <font>
                                <Font name="Corbel Italic" size="17.0" />
                            </font>
                        </Label>
                        <PasswordField fx:id="passlog" layoutX="56.0" layoutY="156.0" promptText="Password" />
                        <TextField fx:id="emaillog" layoutX="56.0" layoutY="109.0" promptText="Email" />
                        <Button fx:id="accedi" layoutX="108.0" layoutY="236.0" mnemonicParsing="false" onAction="#Login" text="Accedi" />
                      <Label fx:id="messagelog1" layoutX="60.0" layoutY="301.0" prefHeight="17.0" prefWidth="162.0" />
                    </children>
                </AnchorPane>
                <AnchorPane minHeight="0.0" minWidth="0.0" prefHeight="160.0" prefWidth="100.0">
                    <children>
                        <Label layoutX="117.0" layoutY="14.0" text="Registrati">
                            <font>
                                <Font name="Corbel Italic" size="17.0" />
                            </font>
                        </Label>
                        <PasswordField fx:id="passreg" layoutX="70.0" layoutY="160.0" promptText="Password" />
                        <TextField fx:id="emailreg" layoutX="70.0" layoutY="112.0" promptText="Email" />
                        <Button fx:id="reg" layoutX="112.0" layoutY="238.0" mnemonicParsing="false" onAction="#Signup" text="Registrati" />
                        <Label fx:id="messagereg" layoutX="60.0" layoutY="303.0" prefHeight="17.0" prefWidth="162.0" />
                    </children>
                </AnchorPane>
            </items>
        </SplitPane>
    </children>
</AnchorPane>
```

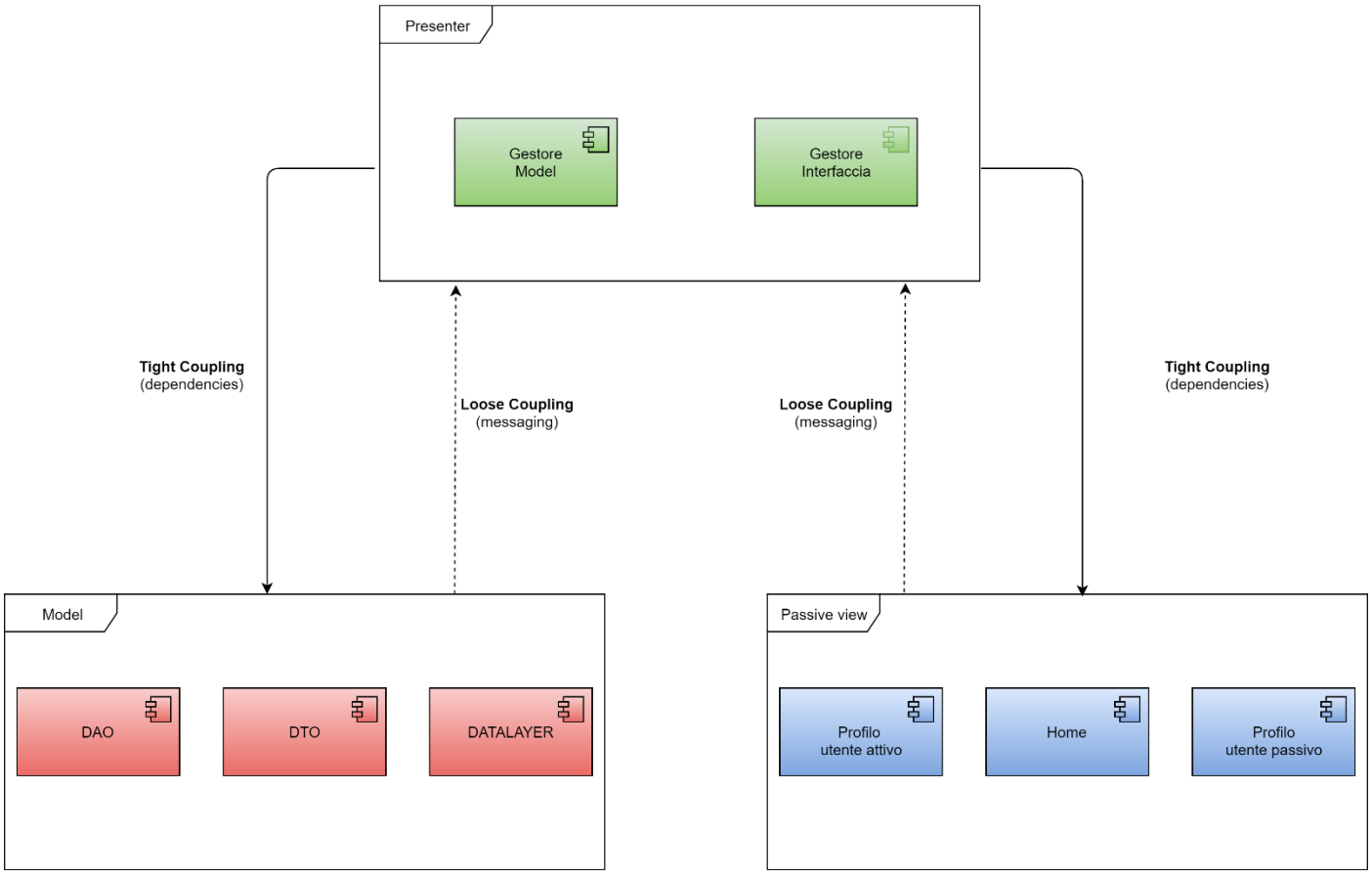
Presenter

È il mediatore tra la vista e il modello, recupera i dati, i Model e li invia alle Views ma a differenza di MVC, decide anche cosa succede quando si interagisce con la view.

```java
@FXML
private void Login(ActionEvent event) throws DataLayerException {
    try (Connection connection = DataLayer.getConnection()) {
    if (emaillog != null && emaillog != null) {
        Utente utente = new Utente();
        utente.setEmail(emaillog.toString());
        utente.setHashPassword(PasswordUtility.getSHA256(passlog.toString()));
            UtenteDAO dao = UtenteDAO.getInstance();
            int utenteID = dao.loginUtente(utente.getEmail(), utente.getHashPassword());
            if (utenteID > 0) {
                Utente utenteLog = dao.getByPK(utenteID);
                messagelog.setText("Login effettuato");
                Stage loginSuccessfulStage = new Stage();
                Parent root = FXMLLoader.load(getClass().getResource("/view/Home.fxml"));
                loginSuccessfulStage.setTitle("Biblioteca");
                loginSuccessfulStage.setScene(new Scene(root));
                loginSuccessfulStage.show();
                loginSuccessfulStage.setResizable(false);
            } else {
                messagelog.setText("Password e Email errati!");
            }
    } else {
        messagelog.setText("Inserisci i parametri");
    }
    } catch (SQLException | IOException ex) {
        throw new DataLayerException("Errore connessione al DB", ex);
    }
}
```

Il team ha deciso anche di spiegare attraverso un Component Diagram il pattern architetturale MVP applicato al sistema.

- Può ricevere messaggi dal model e dalla view
- Chiama(Backend) i metodi di servizio
- Chiama / Accede il/al Buisness Logic Layer
- Chiama i metodi della view e aggiorna l'interfaccia grafica



- Ha POJO come attibuti con i metodi di get e set oppure
- Ha POJO come attributi con i metodi delegati
- Può inviare messaggi al Presenter

- Implementa l'interfaccia grafica
- Crea i componenti User Interface
- Contiene i comportamenti dell'interfaccia utente
- Può inviare messaggi al Presenter