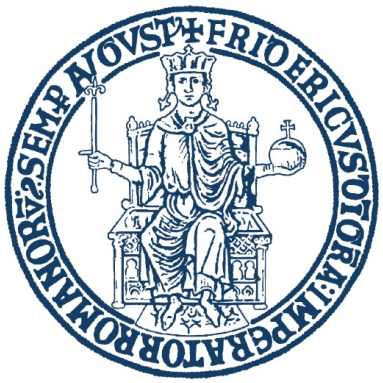


UNIVERSITÀ DEGLI STUDI
DI NAPOLI FEDERICO II

Progetto ToDo

Emanuele Gerardo Palmieri (N86005064), Marco Tessitore (N86005068)

April 2025

Indice

1 Traccia del progetto **3**

2 Introduzione **3**
2.1 Obiettivo del progetto 3
2.2 Contesto e scopo dell'applicazione 4

3 Analisi dei requisiti **4**
3.1 Requisiti fondamentali 4
3.2 Requisiti Non Funzionali 5
3.3 Attori del sistema 5

4 Progettazione e Architettura **5**
4.1 Diagramma delle Classi del Model 5
4.2 Diagramma delle Classi completo 6
4.3 Diagrammi di Sequenza 7
4.3.1 Sequenza: Autenticazione Utente 7
4.3.2 Sequenza: Aggiunta di un Nuovo ToDo 7

5 Dettagli implementativi **8**
5.1 Package gui - Interfaccia Utente (View) 9
5.1.1 Classe Main.java 9
5.1.2 Classe View.java 9
5.2 Package controller - Logica Applicativa 12
5.2.1 Classe Controller.java 12
5.3 Package org.ToDo - Classi di Dominio (Model) 14
5.3.1 Classe Utente.java 14
5.3.2 Classe Bacheca.java 14
5.3.3 Classe ListaUtenti.java 14
5.3.4 Classe Titolo.java 15
5.3.5 Classe ToDo.java 15
5.4 Package dao e implementazioniPostgresDAO - Accesso ai Dati (DAO) 15
5.4.1 Classe ToDoDAOImpl.java 15
5.4.2 Classe UtenteDAOImpl.java 16
5.4.3 Classe BachecaDAOImpl.java 16
5.4.4 Classe ListaUtentiDAOImpl.java 17
5.5 Package util - Classi di Utilità 18
5.5.1 Classe ConnectionFactory.java 18

1 Traccia del progetto

Si sviluppi un sistema informativo, composto da una base di dati relazionale e da un applicativo Java dotato di GUI (Swing) che consente di tenere traccia delle attività personali da svolgere (nel seguito chiamate semplicemente "ToDo"), ispirato al software Trello. Il software deve consentire all'utente di organizzare e gestire le proprie attività personali in modo efficiente, utilizzando un'interfaccia intuitiva e flessibile simile a quella di Trello (https://trello.com/). L'utente entra nel sistema specificando la propria login e password, che devono essere univoche. I ToDo sono organizzati in tre bacheche (ogni bacheca ha un titolo e una descrizione), che possono essere create, modificate ed eliminate. I titoli delle tre bacheche sono Università, Lavoro e Tempo Libero. I ToDo all'interno di una bacheca sono ordinati, secondo un ordine modificabile dall'utente. L'utente può creare, modificare ed eliminare ToDo, così come può spostare un ToDo da una bacheca all'altra oppure cambiarne la posizione all'interno della bacheca. Ogni ToDo ha un titolo e una data di scadenza, un link ad una URL correlata all'attività, una descrizione dettagliata e un'immagine. Tutti questi elementi sono opzionali e possono essere modificati in qualsiasi momento. Ogni ToDo può contenere inoltre una lista di altri utenti che condividono quel ToDo. In pratica quel ToDo comparirà nella bacheca corrispondente di ognuno di tali utenti . Ad esempio, se Pippo, Pluto e Paperino condividono un ToDo, ed esso si trova nella bacheca Tempo Libero di Pippo, allora esso comparirà anche nelle bacheche Tempo Libero di Pluto e Paperino. Ogni utente può leggere chi sono gli altri utenti con i quali il ToDo è condiviso. L'autore del ToDo può aggiungere o eliminare condivisioni. Infine il ToDo ha un colore di sfondo che viene mostrato nell'interfaccia grafica. L'utente deve poter scrivere e modificare ognuna di tali informazioni. Un ToDo può essere settato come completato oppure come non completato (di default è non completato). Il sistema deve poter fornire su richiesta dell'utente, l'elenco di ToDo in scadenza nella giornata odierna, oppure quelli in scadenza entro un certo giorno specificato dall'utente. Il sistema deve consentire anche la ricerca per nome o per titolo dei ToDo. Per i ToDo scaduti, il sistema mostra il nome del ToDo in rosso, per evidenziare il superamento della scadenza prevista.

2 Introduzione

2.1 Obiettivo del progetto

L'obiettivo primario di questo progetto è lo sviluppo di un sistema informativo completo per la gestione di attività personali. Il sistema si compone di un'applicazione desktop con interfaccia grafica (GUI) e di una base di dati relazionale per la persistenza delle informazioni. Il progetto mira a fornire uno strumento efficiente e intuitivo, per aiutare gli utenti a organizzare, monitorare e gestire i propri impegni quotidiani.

2.2 Contesto e scopo dell'applicazione

Nell'attuale contesto dinamico, la capacità di gestire efficacemente il proprio tempo e le proprie attività è diventata una necessità fondamentale sia in ambito lavorativo che personale. L'applicazione "ToDo" nasce per rispondere a questa esigenza, offrendo una piattaforma centralizzata dove l'utente può non solo elencare i propri compiti, ma anche categorizzarli, stabilire priorità tramite scadenze e collaborare con altri utenti. Lo scopo è quello di superare i limiti di una semplice lista di attività, fornendo funzionalità avanzate come l'organizzazione in bacheche tematiche (Bacheca), la possibilità di allegare dettagli come URL e immagini, e la condivisione collaborativa di specifici ToDo, rendendo il software uno strumento flessibile e adatto a molteplici scenari d'uso.

3 Analisi dei requisiti

Questa sezione delinea i requisiti che il sistema deve soddisfare, suddivisi in funzionali, non funzionali, e identifica gli attori che interagiranno con esso. Le principali entità logiche individuate per soddisfare tali requisiti sono: **Utente, Bacheca, ToDo, TitoloB** (come enumerazione per i titoli delle bacheche) e **ListaUtenti** (per la gestione delle condivisioni).

3.1 Requisiti fondamentali

I requisiti funzionali descrivono le operazioni specifiche che il sistema deve essere in grado di eseguire.

- **RF1: Autenticazione Utente**
  - Il sistema deve permettere a un utente di accedere tramite credenziali univoche (login e password).
- **RF2: Gestione delle Bacheche**
  - I ToDo devono essere organizzati in tre bacheche fisse: **Università, Lavoro e Tempo Libero**.
  - Ogni bacheca deve avere un titolo e una descrizione modificabile.
- **RF3: Gestione dei ToDo**
  - L'utente deve poter creare, modificare ed eliminare i ToDo.
  - Ogni ToDo deve poter contenere: un titolo, una data di scadenza, un link URL, una descrizione dettagliata, un'immagine e un colore di sfondo per l'interfaccia.
  - Un ToDo può essere impostato come "completato" o "non completato" (stato di default).
- **RF4: Ordinamento e Spostamento**

  - L'utente deve poter modificare l'ordine dei ToDo all'interno di una bacheca.
  - L'utente deve poter spostare un ToDo da una bacheca all'altra.

- **RF5: Condivisione dei ToDo**
  - L'autore di un ToDo può condividerlo con una lista di altri utenti registrati.
  - Un ToDo condiviso deve apparire nella bacheca corrispondente di tutti gli utenti con cui è condiviso.
  - Ogni utente può visualizzare la lista degli utenti con cui un ToDo è condiviso.
  - Solo l'autore può aggiungere o rimuovere utenti dalla lista di condivisione.

3.2 Requisiti Non Funzionali

I requisiti non funzionali definiscono le qualità e i vincoli del sistema.

- **RNF1: Interfaccia Utente:** La GUI deve essere intuitiva.
- **RNF2: Persistenza dei Dati:** I dati dell'applicazione devono essere memorizzati in modo persistente su una **base di dati relazionale.**
- **RNF3: Usabilità:** L'applicazione deve fornire feedback visivi chiari all'utente (es. colore per ToDo scaduti) per migliorare l'esperienza d'uso.

3.3 Attori del sistema

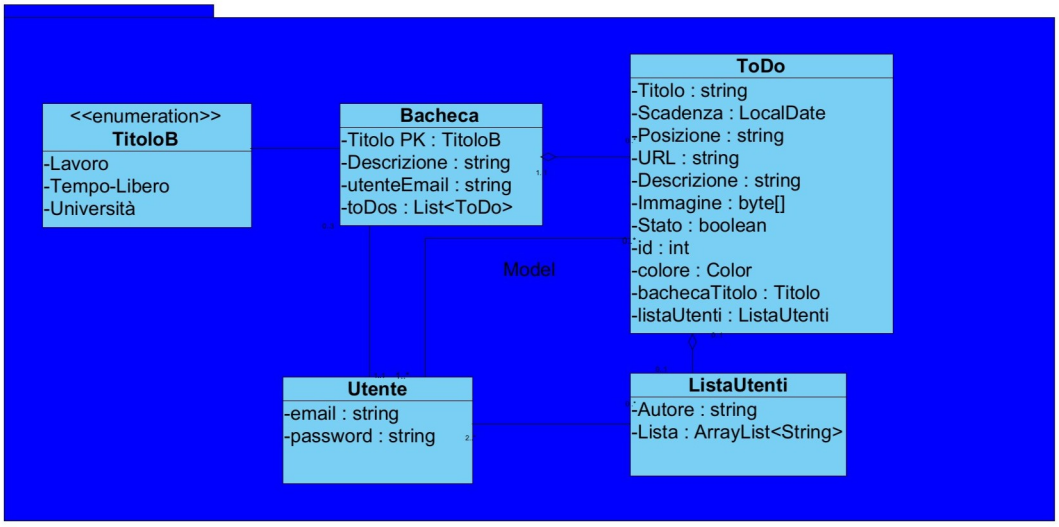
Dall'analisi della traccia emerge un unico attore principale:

- **Utente:** Una persona registrata al sistema che, dopo essersi autenticata, può eseguire tutte le funzionalità descritte nei requisiti, come la gestione delle proprie bacheche, la creazione e condivisione di ToDo.

4 Progettazione e Architettura

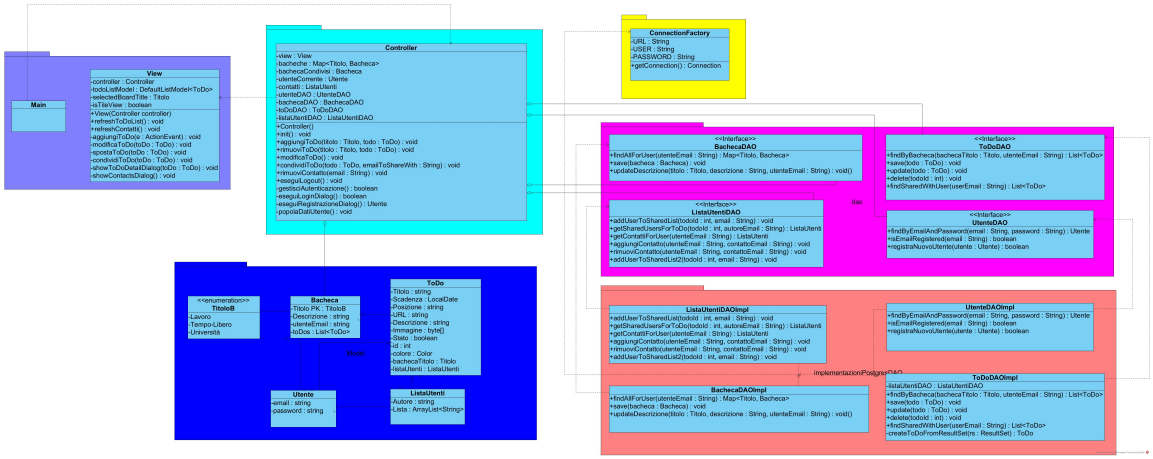
4.1 Diagramma delle Classi del Model

Il seguente diagramma delle classi illustra la struttura statica del package Model *(org.ToDo)*. Questo diagramma si concentra esclusivamente sulle entità di dominio che rappresentano il nucleo dei dati dell'applicazione, come *Utente*, *Bacheca* e *ToDo*. Vengono mostrati gli attributi principali di ogni classe e le relazioni strutturali (come l'aggregazione) che le legano, offrendo una visione chiara di come i dati sono organizzati.



4.2 Diagramma delle Classi completo

Questo diagramma delle classi offre una visione d'insieme dell'intera architettura del sistema. Vengono rappresentati tutti i package *(model, gui, controller, dao, implementazioniPostgresDAO, util)* e le relative classi. L'obiettivo è mostrare non solo le entità dati, ma anche come i diversi layer dell'applicazione interagiscono tra loro secondo il pattern **Model-View-Controller (MVC)** e **Data Access Object (DAO)**, evidenziando le dipendenze e le relazioni che governano la logica del software.

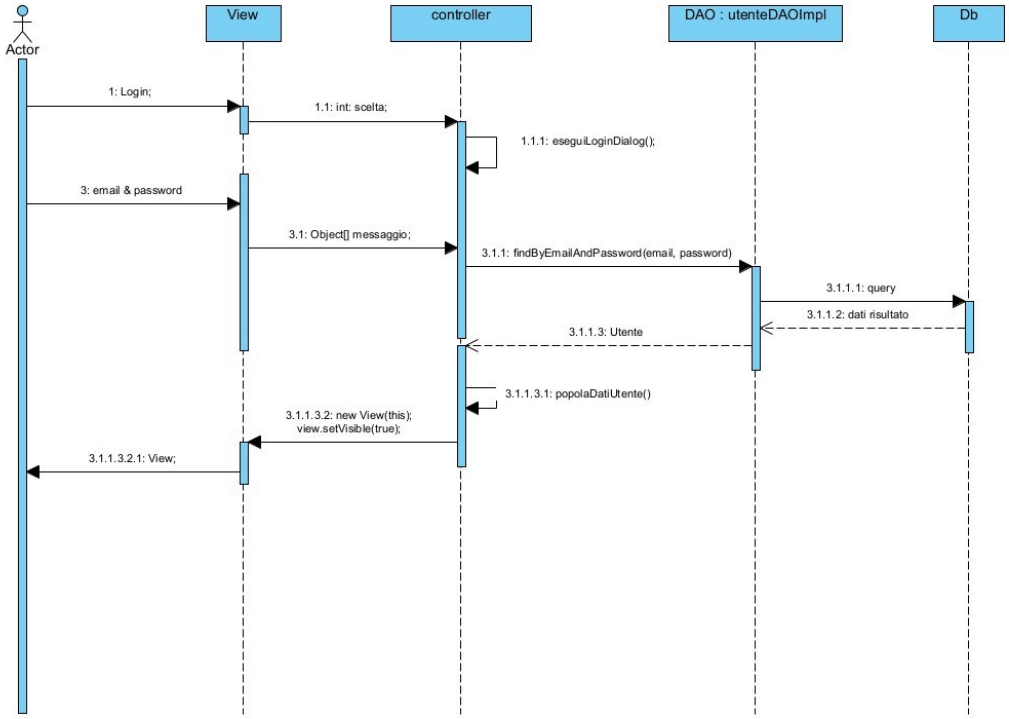


4.3 Diagrammi di Sequenza

I diagrammi di sequenza descrivono il comportamento dinamico del sistema, illustrando le interazioni tra gli oggetti nel tempo.

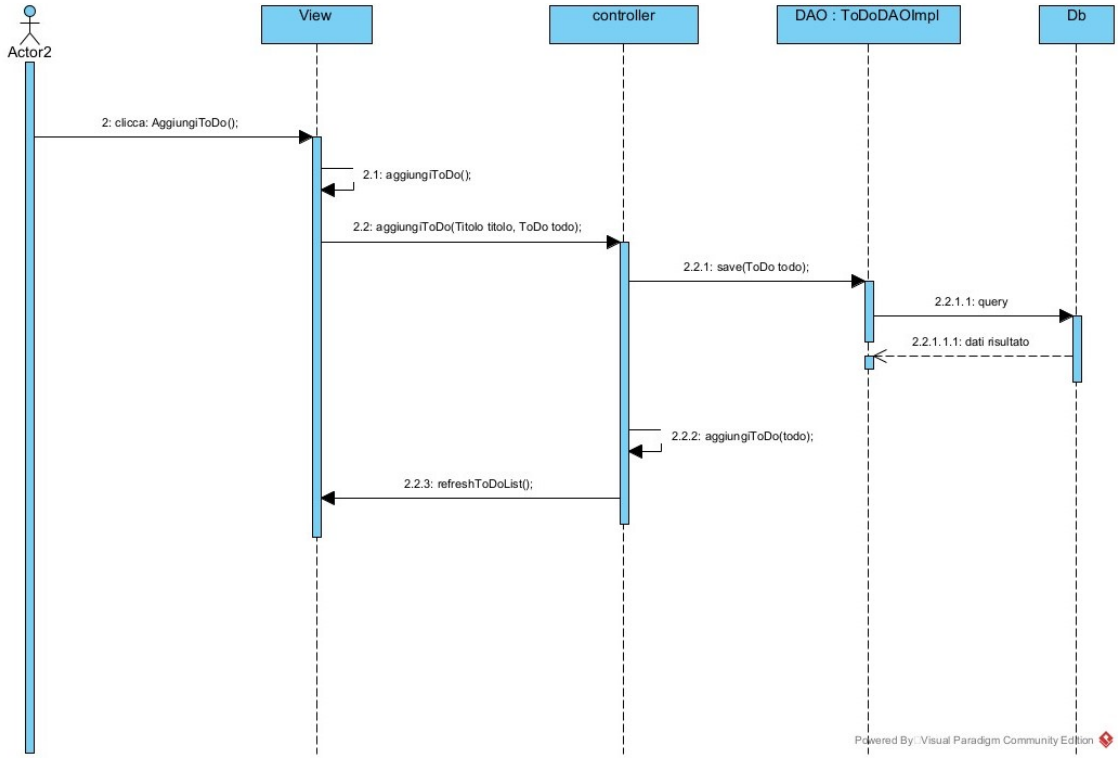
4.3.1 Sequenza: Autenticazione Utente

Questo diagramma mostra la sequenza di messaggi scambiati tra l'attore *Utente*, l'interfaccia *(View)*, il *Controller* e il *DAO* durante il processo di login. Vengono dettagliate le chiamate ai metodi necessarie per verificare le credenziali dell'utente sul database e per avviare la sessione di lavoro.



4.3.2 Sequenza: Aggiunta di un Nuovo ToDo

Questo diagramma illustra il flusso di interazioni che si verificano quando un utente crea una nuova attività. La sequenza parte dall'input dell'utente sulla *View*, passa per il *Controller* che gestisce la logica di business e arriva fino al *DAO*, che si occupa del salvataggio del nuovo *ToDo* nel database e del successivo aggiornamento dell'interfaccia.



5 Dettagli implementativi

L'applicazione segue il pattern architetturale Model-View-Controller (MVC), una scelta progettuale che separa nettamente le responsabilità del software in tre componenti principali, rendendo il codice più organizzato, manutenibile e scalabile:

1. **Model:** È il cuore dei dati e della logica di business dell'applicazione. È a sua volta suddiviso in:

   - **Classi di Dominio:** Semplici oggetti Java (POJO) come Utente, Bacheca e ToDo, che rappresentano le entità fondamentali gestite dal programma.
   - **Data Access Objects (DAO):** Un insieme di interfacce (ToDoDAO, UtenteDAO) e delle loro implementazioni concrete (ToDoDAOImpl, UtenteDAOImpl) che incapsulano tutta la logica di accesso al database PostgreSQL, isolando il resto dell'applicazione dai dettagli specifici del linguaggio SQL.

2. **View:** Costituisce l'interfaccia utente grafica (GUI) con cui l'utente interagisce. Realizzata con la libreria Java Swing, la sua unica responsabilità è quella di visualizzare i dati forniti dal Controller e di catturare gli input dell'utente (come click sui pulsanti o testo inserito nei campi).

3. **Controller:** Agisce da intermediario tra il Model e la View. Riceve le richieste dell'utente dalla View, le elabora invocando la logica di business e di accesso ai dati presente nel Model, e infine aggiorna la View per mostrare i risultati all'utente.

   La seguente analisi dettagliata esamina ogni classe e metodo all'interno di questi tre livelli, fornendo una spiegazione approfondita del loro scopo specifico e del loro funzionamento nel contesto generale dell'applicazione.

5.1 Package gui - Interfaccia Utente (View)

5.1.1 Classe Main.java

Questa classe è il **punto di ingresso** (entry point) dell'intera applicazione.

- **public static void main(String[] args)**
- **Scopo:** È il primo metodo che viene eseguito all'avvio del programma.
- **Funzionamento:**

  1. Tenta di impostare il "Look and Feel" dell'interfaccia grafica su "Nimbus" per un aspetto più moderno.
  2. Utilizza **SwingUtilities.invokeLater** per garantire che l'interfaccia grafica venga creata e gestita sul thread corretto di Swing (l'Event Dispatch Thread - EDT). Questo è fondamentale per evitare problemi di concorrenza.
  3. Crea una nuova istanza del **Controller** e chiama il suo metodo init(), che dà il via a tutta la logica dell'applicazione.

5.1.2 Classe View.java

Rappresenta la finestra principale **(JFrame)** dell'interfaccia grafica (GUI) e orchestra tutti i componenti visivi dell'applicazione. La sua responsabilità è visualizzare i dati (bacheche, ToDo) e catturare le interazioni dell'utente, delegando la logica di business al **Controller.** È stata potenziata per supportare molteplici modalità di visualizzazione e una navigazione più ricca.

Struttura e Componenti Principali

- **JSplitPane:** La finestra è dominata da un **JSplitPane** che separa l'area di navigazione a sinistra (dove vengono mostrate le bacheche) dall'area di contenuto principale a destra (dove vengono visualizzati i ToDo).

- **Barra di Navigazione Superiore:** Un pannello **(JPanel)** in alto contiene il nome dell'applicazione e una serie di pulsanti per le azioni globali:
  - **Home:** Riporta alla schermata iniziale di selezione delle bacheche.
  - **Profilo:** Mostra le informazioni base dell'utente.
  - **Condivisi:** Visualizza la bacheca speciale con i ToDo condivisi da altri utenti.
  - **ToDo:** Mostra una vista aggregata di tutti i ToDo non completati.
  - **Contatti:** Apre un dialogo per la gestione della lista contatti.
  - **Logout:** Permette all'utente di disconnettersi.
- **JToggleButton per la Vista:** Un pulsante speciale **(JToggleButton)** permette di alternare dinamicamente la visualizzazione dei ToDo tra una vista a lista tradizionale e una vista a riquadri (o "card").

Metodi e Funzionalità Chiave

- **Costruttore public View(Controller controller)** La finestra è dominata da un JSplitPane che separa l'area di navigazione a sinistra (dove vengono mostrate le bacheche) dall'area di contenuto principale a destra (dove vengono visualizzati i ToDo).
  - **Scopo:** Inizializza la finestra principale e i suoi componenti.
  - **Funzionamento:** Imposta il titolo, le dimensioni e la chiusura della finestra. Crea i componenti principali come la barra di navigazione (topNavBarPanel), il contenitore per la selezione delle bacheche (boardSelectionContainer) e il pannello per la visualizzazione dei ToDo (todoListPanel). Inizializza anche il JSplitPane che gestisce il layout principale.
- **private void showToDoListForBoard()**
  - **Scopo:** Mostra la lista dei ToDo per una bacheca specifica dopo la selezione dell'utente.
  - **Funzionamento:** Configura e crea una JList per i ToDo. Aggiunge un MouseListener che distingue un click sull'area del checkbox (a sinistra) per cambiare lo stato, da un click sul resto dell'elemento, che apre i dettagli del ToDo. Infine, invoca refreshToDoList() per popolare la vista.
- **public void refreshToDoList()**
  - **Scopo:** Aggiorna e ricarica la visualizzazione dei ToDo. Viene chiamato dal **Controller** ogni volta che i dati cambiano.

  - **Funzionamento:** È un metodo centrale che ripulisce completamente il **todoListPanel**. Successivamente, ricostruisce l'interfaccia chiamando **populateListView()** o **populateTileView()** in base allo stato del pulsante **viewToggleButton**, garantendo che l'utente veda sempre la vista corretta e aggiornata.

- **private void aggiungiToDo(...) / private void modificaToDo(...)**
  - **Scopo:** Gestiscono l'apertura delle finestre di dialogo **(JOptionPane)** per aggiungere un nuovo ToDo o modificare uno esistente.
  - **Funzionamento:** Utilizzano la classe interna **ToDoFormElements** per creare un pannello con tutti i campi necessari (titolo, descrizione, data, colore, etc.). Alla conferma dell'utente, raccolgono i dati dai campi del form e invocano il metodo appropriato nel **Controller** per salvare o aggiornare il ToDo.

- **private void showToDoDetailDialog(ToDo todo)**
  - **Scopo:** Mostra una finestra di dialogo **(JDialog)** con tutti i dettagli di un ToDo.
  - **Funzionamento:** Costruisce un pannello con etichette **(JLabel)** che mostrano tutte le informazioni del ToDo, inclusi attributi opzionali come URL (reso cliccabile) e un'anteprima dell'immagine. Aggiunge un pannello con i pulsanti per le azioni possibili (Modifica, Elimina, Sposta, Condividi).

- **private void condividiToDo(ToDo todo)**
  - **Scopo:** Apre un dialogo per permettere all'utente di condividere un ToDo.
  - **Funzionamento:** Mostra una lista di checkbox con i contatti esistenti dell'utente e un campo di testo per inserire una nuova email. Alla conferma, chiama il metodo **controller.condividiToDo** per ogni utente selezionato o inserito.

- **private void performLogout(...)**
  - **Scopo:** Gestisce il logout dell'utente.
  - **Funzionamento:** Chiede conferma tramite un JOptionPane e, se l'utente acconsente, invoca controller.eseguiLogout() per terminare la sessione corrente.

Classi, Record e Metodi Interni Rilevanti

- **private static class ToDoFormElements**

- **Scopo:** È una classe interna **static** che agisce da contenitore per tutti i componenti Swing di un form (es. **JTextField, JComboBox**).
- **Funzionamento:** Incapsula gli elementi di un form di aggiunta/modifica ToDo in un unico oggetto. Questo approccio pulisce il codice e rende più semplice e leggibile la creazione e la gestione dei form, evitando di passare numerosi parametri ai metodi.

- **public static class ToDoCellRenderer**
  - **Scopo:** Classe interna che definisce come un singolo oggetto ToDo viene disegnato all'interno di una JList (nella vista a lista).
  - **Funzionamento:** Estende JCheckBox per mostrare lo stato di completamento. Imposta il testo con titolo e scadenza. Gestisce dinamicamente il colore di sfondo e del testo:
    * Usa il colore personalizzato del ToDo.
    * Evidenzia la riga di un rosso chiaro se il ToDo è scaduto e non completato.
    * Sceglie il colore del testo (bianco o nero) per garantire la massima leggibilità in base alla luminanza del colore di sfondo, grazie al metodo isColorDark.

- **record NamedColor(String name, Color color)**
  - **Scopo:** Un **record** Java che associa un nome leggibile (es. "Rosso Chiaro") a un oggetto Color.
  - **Funzionamento:** Viene usato per popolare la **JComboBox** per la selezione del colore. L'override del metodo toString() è fondamentale, perché permette alla **JComboBox** di mostrare il nome del colore invece di un riferimento all'oggetto, migliorando notevolmente l'esperienza utente.

- **private JPanel createToDoCard(ToDo todo)**
  - **Scopo:** Crea un pannello **(JPanel)** che rappresenta visivamente un singolo ToDo come una "card" nella vista a riquadri.
  - **Funzionamento:** Costruisce un **JPanel** con titolo e descrizione. Come il **ToDoCellRenderer**, imposta il colore di sfondo in base alle impostazioni del ToDo e lo evidenzia in rosso chiaro se è scaduto. Aggiunge un **MouseListener** per rendere la card cliccabile e aprire i dettagli del ToDo.

5.2 Package controller - Logica Applicativa

5.2.1 Classe Controller.java

Funziona da intermediario tra l'interfaccia utente **(View)** e i dati **(Model e DAO)**.

- **public void init()**
  - **Scopo:** Avvia l'applicazione, gestendo il flusso di autenticazione.
  - **Funzionamento:** Entra in un ciclo while che mostra all'utente le opzioni di Login o Registrazione. Se l'autenticazione ha successo, esce dal ciclo, popola i dati dell'utente e avvia la **View.** Altrimenti, chiede all'utente se vuole riprovare o uscire.
- **private void popolaDatiUtente()**
  - **Scopo:** Carica tutti i dati dell'utente dal database dopo un login riuscito.
  - **Funzionamento:** Utilizza le classi DAO (**bachecaDAO, toDoDAO, listaUtentiDAO**) per recuperare tutte le bacheche, i ToDo associati a ciascuna bacheca, i ToDo condivisi e la lista dei contatti dell'utente corrente.
- **public void aggiungiToDo(Titolo titolo, ToDo todo)**
  - **Scopo:** Gestisce la logica per aggiungere un nuovo ToDo.
  - **Funzionamento:** Riceve il ToDo dalla **View**, lo salva nel database tramite **toDoDAO.save()**, lo aggiunge all'oggetto Bacheca in memoria e infine chiama **view.refreshToDoList()** per aggiornare l'interfaccia.
- **public void modificaToDo(...)**
  - **Scopo:** Gestisce la logica per modificare un ToDo esistente.
  - **Funzionamento:** Aggiorna gli attributi dell'oggetto **ToDo** con i nuovi dati ricevuti dalla **View**, lo aggiorna nel database tramite **toDoDAO.update()** e rinfresca la **View.**
- **public void rimuoviToDo(Titolo titolo, ToDo todo)**
  - **Scopo:** Gestisce la logica per eliminare un ToDo.
  - **Funzionamento:** Elimina il ToDo dal database tramite **toDoDAO.delete()**, lo rimuove dalla lista in memoria e aggiorna la **View.**
- **public void spostaToDoGUI(...)**
  - **Scopo:** Sposta un ToDo da una bacheca all'altra.
  - **Funzionamento:** Aggiorna il **bachecaTitolo** del ToDo, lo salva nel database, lo rimuove dalla lista della bacheca di origine e lo aggiunge a quella di destinazione, per poi aggiornare la **View.**
- **public void condividiToDo(ToDo todo, String emailToShareWith)**
  - **Scopo:** Condivide un ToDo con un altro utente.

  - **Funzionamento:** Verifica che l'utente con cui si condivide esista. Aggiunge l'utente alla lista di condivisione del ToDo nel database tramite **listaUtentiDAO.addUserToSharedList().**

- **public void eseguiLogout()**
  - **Scopo:** Esegue il logout.
  - **Funzionamento:** Imposta l'utente corrente a **null**, chiude la **View (dispose())** e crea una nuova istanza del **Controller** per ricominciare il ciclo di autenticazione.

5.3 Package org.ToDo - Classi di Dominio (Model)

5.3.1 Classe Utente.java

Un semplice POJO (Plain Old Java Object) che modella un utente.

- **Attributi: email** (String) e **password** (String).
- **Metodi:** Contiene solo il costruttore e i metodi **get()** per accedere ai suoi attributi privati.

5.3.2 Classe Bacheca.java

Rappresenta una bacheca tematica.

- **Attributi: titolo** (enum Titolo), **descrizione** (String), **utenteEmail** (String) e una **List<ToDo>.**
- **Metodi:** Oltre ai metodi **get() e set()**, ha metodi per manipolare la lista di ToDo, come **aggiungiToDo(ToDo todo)** e **rimuoviToDo(int indice).**

5.3.3 Classe ListaUtenti.java

Rappresenta un elenco di utenti associati a un contesto specifico, come la condivisione di un ToDo o una lista di contatti.

- **Attributi: Autore** (String), ovvero l'email del proprietario della lista, e **Lista** (ArrayList)
- **Metodi:** Oltre al costruttore e ai metodi **get()** per accedere ai suoi attributi, dispone di metodi per gestire la lista degli utenti, come **aggiungiUtente(String email)** per aggiungere un nuovo utente (evitando duplicati) e **rimuovi(String utente)** per eliminarne uno.

5.3.4 Classe Titolo.java

È un'enumerazione **(enum)** che definisce i titoli predefiniti e immutabili per le bacheche tematiche, garantendo coerenza e sicurezza dei tipi in tutta l'applicazione.

- **Attributi:** Definisce un insieme fisso di costanti che rappresentano i titoli delle bacheche: **LAVORO, TEMPO LIBERO e UNIVERSITA.**
- **Metodi:** Essendo un'enumerazione, non ha metodi personalizzati definiti nel file, ma fornisce implicitamente metodi per accedere alle costanti e ai loro nomi.

5.3.5 Classe ToDo.java

Modella una singola attività.

- **Attributi:** Contiene tutti i dettagli di un'attività: **id, titolo, descrizione, scadenza, stato, url, posizione, colore, immagine, bachecaTitolo**, e un oggetto **ListaUtenti** per la condivisione.
- **Metodi:** Principalmente metodi **get()** e **set()** per accedere e modificare i suoi attributi.

5.4 Package dao e implementazioniPostgresDAO - Accesso ai Dati (DAO)

5.4.1 Classe ToDoDAOImpl.java

Implementazione concreta dell'interfaccia ToDoDAO per il database PostgreSQL.

- **private ToDo createToDoFromResultSet(ResultSet rs)**
  - **Scopo:** Metodo di utilità per creare un oggetto **ToDo** a partire da una riga di un **ResultSet** del database. Evita la duplicazione del codice.
  - **Funzionamento:** Legge ogni colonna dal **ResultSet (rs.getString("titolo"), rs.getDate("scadenza")**, ecc.) e li usa per costruire e restituire un nuovo oggetto **ToDo.**
- **public List<ToDo> findByBacheca(...)**
  - **Scopo:** Trova tutti i ToDo di una specifica bacheca.
  - **Funzionamento:** Esegue una query **SELECT * FROM todo WHERE bacheca titolo = ? AND autore email = ?**, cicla sui risultati e usa **createToDoFromResultSet** per creare la lista.
- **public void save(ToDo todo)**

- **Scopo:** Salva un nuovo ToDo nel database.
- **Funzionamento:** Esegue una query **INSERT INTO todo (...) VALUES (?, ?, ...)** usando i dati dell'oggetto **ToDo.** Recupera l'ID generato automaticamente dal database per impostarlo nell'oggetto **ToDo**.

- **public void update(ToDo todo)**
  - **Scopo:** Aggiorna un ToDo esistente.
  - **Funzionamento:** Esegue una query **UPDATE todo SET titolo = ?, ... WHERE id = ?.**

- **public void delete(int todoId)**
  - **Scopo:** Elimina un ToDo.
  - **Funzionamento:** Esegue una query con una **JOIN** tra la tabella **todo** e la tabella di associazione **todo utenti condivisi** per trovare le corrispondenze.

- **public List<ToDo> findSharedWithUser(String userEmail)**
  - **Scopo:** Trova i ToDo condivisi con un utente.
  - **Funzionamento:** Esegue una query con una **JOIN** tra la tabella **todo** e la tabella di associazione **todo utenti condivisi** per trovare le corrispondenze.

5.4.2 Classe UtenteDAOImpl.java

Implementazione di **UtenteDAO.**

- **public Utente findByEmailAndPassword(...):** Esegue una **SELECT** per verificare se esiste un utente con l'email e la password fornite.
- **public boolean isEmailRegistered(String email):** Esegue una **SELECT 1 FROM utente WHERE email = ?** per verificare rapidamente se un'email è già in uso.
- **public boolean registraNuovoUtente(Utente utente):** Esegue una **INSERT** per creare un nuovo record nella tabella **utente.**

5.4.3 Classe BachecaDAOImpl.java

Implementa l'interfaccia BachecaDAO, fornendo la logica concreta per interagire con la tabella bacheca in un database PostgreSQL.

- **public Map<Titolo, Bacheca> findAllForUser(String utenteEmail)**

- **Scopo:** Recupera dal database tutte le bacheche appartenenti a un utente specifico.
- **Funzionamento:** Esegue una query SQL **SELECT titolo, descrizione FROM bacheca WHERE utente email = ?.** Per ogni riga restituita, crea un oggetto **Bacheca** e lo inserisce in una mappa che viene poi restituita al chiamante (il **Controller**).

- **public void save(Bacheca bacheca)**
  - **Scopo:** Salva una nuova bacheca nel database.
  - **Funzionamento:** Esegue una query SQL **INSERT INTO bacheca (titolo, descrizione, utente email) VALUES (?, ?, ?)** utilizzando i dati forniti dall'oggetto Bacheca passato come parametro.

- **public void updateDescrizione(Titolo titolo, String descrizione, String utenteEmail)**
  - **Scopo:** Aggiorna la descrizione di una bacheca esistente.
  - **Funzionamento:** Esegue una query SQL **UPDATE bacheca SET descrizione = ? WHERE titolo = ? AND utente email = ?** per modificare la descrizione della bacheca specificata per un dato utente.

5.4.4 Classe ListaUtentiDAOImpl.java

Implementa l'interfaccia **ListaUtentiDAO**, gestendo la persistenza delle liste di contatti e delle condivisioni dei ToDo. Interagisce con le tabelle **contatto** e **todo utenti condivisi.**

- **public void addUserToSharedList(int todoId, String email)**
  - **Scopo:** Associa un utente a un ToDo per la condivisione.
  - **Funzionamento:** Esegue una query **INSERT INTO todo utenti condivisi (todo id, utente email) VALUES (?, ?).** Utilizza la clausola **ON CONFLICT DO NOTHING** per evitare errori nel caso in cui la condivisione esista già.

- **public ListaUtenti getSharedUsersForToDo(int todoId, String autoreEmail)**
  - **Scopo:** Recupera la lista degli utenti con cui un ToDo è condiviso.
  - **Funzionamento:** Esegue una **SELECT utente email FROM todo utenti condivisi WHERE todo id = ?** per ottenere tutte le email associate a un **todoId** e le restituisce in un oggetto **ListaUtenti.**

- **public ListaUtenti getContattiForUser(String utenteEmail)**

  - **Scopo:** Recupera la lista dei contatti di un utente.
  - **Funzionamento:** Esegue una **SELECT contatto email FROM contatto WHERE utente email = ?** per ottenere la lista di tutti i contatti salvati da un utente.

- **public void aggiungiContatto(String utenteEmail, String contattoEmail)**
  - **Scopo:** Aggiunge un nuovo contatto alla lista di un utente.
  - **Funzionamento:** Esegue una **INSERT INTO contatto (utente email, contatto email) VALUES (?, ?)**, utilizzando anche in questo caso **ON CONFLICT DO NOTHING** per prevenire duplicati.

- **public void rimuoviContatto(String utenteEmail, String contattoEmail)**
  - **Scopo:** Rimuove un contatto dalla lista di un utente.
  - **Funzionamento:** Esegue una **DELETE FROM contatto WHERE utente email = ? AND contatto email = ?** per eliminare la riga che rappresenta il legame tra l'utente e il suo contatto.

5.5 Package util - Classi di Utilità

5.5.1 Classe ConnectionFactory.java

Classe di utilità per la gestione della connessione al database.

- **Attributi:** Contiene le costanti **private static final** con l'URL del database, il nome utente e la password, centralizzando le credenziali in un unico posto.

- **public static Connection getConnection()**
  - **Scopo:** Fornire una connessione al database.
  - **Funzionamento:** Carica il driver JDBC di PostgreSQL e usa **DriverManager.getConnection()** con le credenziali definite nella classe per stabilire e restituire una connessione. Qualsiasi classe DAO che necessita di parlare con il database chiama questo metodo.