

Documentazione Progetto Object Orientation



UNIVERSITÀ DEGLI STUDI
DI NAPOLI FEDERICO II

Cristian Proroga [N86005629]

Maria Grazia Toppi [N86005183]

Pasquale Junior Montò [N86005762]

Anno accademico 2024/2025

Link repository: https://github.com/pakyjr/ToDo_Maven

Indice

Capitolo 1 – Introduzione

1.1 Traccia del Progetto

1.2 Analisi dei requisiti

1.2.1 Utente

1.2.2 Board

1.2.3 ToDo

1.2.4 Relazioni

Capitolo 2 – Diagramma UML delle classi del dominio del progetto

Capitolo 3 – Diagramma UML di dettaglio delle classi nell’implementazione

Capitolo 4 – Sequence Diagram di funzionalità a scelta

4.1 login(username, plainPassword)

Capitolo 5 – Manuale della GUI

5.1 Schermate introduttive

5.1.1 Interfaccia di login/registrazione

5.1.2 Interfaccia di registrazione

5.2 Schermate Personal Area

5.3 Schermate dei ToDo

Capitolo 6 - Gestione degli errori

Capitolo 1

Introduzione

1.1 Traccia del Progetto

1. Obiettivo del Sistema

Il sistema sviluppato per la gestione delle attività personali (ToDo) si ispira al modello di

interazione offerto da **Trello**, ponendosi l'obiettivo di offrire un'interfaccia semplice, intuitiva e modulare per l'organizzazione dei compiti. Il progetto è stato realizzato seguendo i principi dell'Object Orientation e suddiviso logicamente in tre livelli principali: interfaccia utente (GUI), logica applicativa (Controller) e accesso ai dati (DAO).

2. Accesso

L'accesso all'applicazione avviene tramite autenticazione con credenziali personali (login e password).

3. Gestione delle Board

La classe Board contiene gli attributi *boardName*, *description*, *color*, *owner* e *todoList*. L'attributo *owner* consente di distinguere tra chi ha creato una board e chi la visualizza in seguito a una condivisione. La gestione dei ToDo all'interno della board è strutturata in modo simile alla gestione delle board da parte dell'utente, con metodi dedicati all'ordinamento (*sortDueDate*) e alla ricerca (*searchTitle*) dei ToDo.

4. Gestione dei To Do

Ogni oggetto ToDo è caratterizzato da attributi obbligatori (come *title*, *status*, *owner*) e facoltativi (tra cui *dueDate*, *url*, *color*, *image*). Il campo *done* è rappresentato da un valore booleano, inizialmente impostato a false, e può essere modificato dall'utente tramite il metodo *toggle*. L'attributo *activityList* contiene un insieme di attività da completare affinché il ToDo risulti terminato; il metodo *checkValidActivity* consente di verificarne lo stato. Inoltre, ogni ToDo è identificato in modo univoco tramite un *UUID*, utile per la gestione della condivisione e cancellazione.

1.2 Analisi dei requisiti

Per la realizzazione di un applicativo Java utile alla gestione dei To Do, in accordo ai requirements sono state individuate le seguenti classi:

1.2.1 Utente

- **Attributi:**
 - *UUID id*: identificativo univoco dell'utente

- String username: nome utente
- String hashedPassword: password cifrata
- ArrayList<Board> boardList: lista delle bacheche associate all'utente
- **Metodi:**
 - checkPassword(String): verifica la correttezza della password
 - addBoard(BoardName, String): crea e aggiunge una nuova bacheca
 - addBoard(Board): aggiunge una bacheca esistente
 - clearBoards(): rimuove tutte le bacheche
 - fillBoard(String): aggiunge bacheche predefinite se mancanti
 - deleteBoard(BoardName): elimina una bacheca
 - getId(), getUsername(), getHashedPassword(), getBoardList(): metodi getter
 - getBoard(BoardName): restituisce una bacheca specifica
 - moveToDoToAnotherBoard(BoardName, BoardName, int): sposta un ToDo tra bacheche
- **Descrizione:**

Rappresenta un utente registrato nel sistema, con la possibilità di gestire bacheche personali contenenti attività (ToDo). Gestisce l'autenticazione tramite password cifrata e consente operazioni CRUD sulle bacheche.

1.2.2 Board

- **Attributi:**
 - int id: identificativo della bacheca
 - BoardName name: nome della bacheca
 - String owner: creatore della bacheca
 - String color: colore associato
 - List<ToDo> todoList: lista di attività (ToDo)
- **Metodi:**

- `addTodo(String)`, `addTodo(String, String)`: aggiunge un nuovo `ToDo`
- `addExistingTodo(ToDo)`: aggiunge un `ToDo` esistente
- `removeTodo(ToDo)`: rimuove un `ToDo` e aggiorna le posizioni
- `getTodoList()`, `getName()`, `getOwner()`, `getColor()`, `getId()`: metodi getter
- `setColor(String)`, `setId(int)`: metodi setter
- `equals(Object)`, `hashCode()`: confronta bacheche per uguaglianza
- **Descrizione:**

Rappresenta una bacheca di attività associata a un utente. Gestisce l'aggiunta, rimozione e visualizzazione dei `ToDo`, assicurando unicità e mantenendo l'ordine.

1.2.3 ToDo

- **Attributi:**
 - `UUID id`: identificativo univoco
 - `String title`: titolo del `ToDo`
 - `String description`: descrizione
 - `String status`: stato dell'attività
 - `LocalDate dueDate`, `createdDate`: data di scadenza e di creazione
 - `int position`: posizione nella lista
 - `String owner`: creatore del `ToDo`
 - `String url`, `color`, `image`: metadati grafici e link utili
 - `Map<String, Boolean> activityList`: sotto-attività
 - `Set<User> sharedUsers`: utenti con cui è condiviso
- **Metodi:**
 - Getter e setter per tutti gli attributi
 - `addActivity(String)`, `deleteActivity(String)`: gestisce sotto-attività
 - `addSharedUser(User)`, `removeSharedUser(String)`, `clearUsers()`: gestisce utenti condivisi

- equals(Object), hashCode(): confronta ToDo per ID
- **Descrizione:** Rappresenta una singola attività nella bacheca, con dettagli personalizzabili, sotto-attività, e possibilità di condivisione con altri utenti.

1.2.4 Relazioni

- Ha (User → Board)
"Ogni utente può avere 0 o più bacheche (boards)"
- Implementata nei metodi:

```
saveBoard(Board board, UUID userId)
```

```
loadUserBoardsAndTodos(User user)
```

```
getBoardId(BoardName boardName, String username)
```

La relazione è realizzata tramite la colonna user_id nella tabella boards.

Un utente può possedere più board, ma ogni board è associata a un solo utente.

- Ha (Board → ToDo)
"Ogni board può avere 0 o più ToDo associati"
- Gestita nei metodi:

```
saveToDo(ToDo toDo, int boardId)
```

```
loadUserBoardsAndTodos(User user)
```

```
updateToDo(ToDo toDo, int boardId)
```

```
updateToDoBoardId(String toDoId, int newBoardId)
```

Ogni ToDo è collegato a una board tramite board_id.

Le board possono contenere più ToDo, ma ogni ToDo appartiene a una sola board.

- Ha (ToDo → Activity)
"Ogni ToDo può avere 0 o più attività associate (checklist)"
Relazione realizzata nella tabella activities, dove todo_id è la chiave esterna.

- Metodi coinvolti:

```
saveActivities(String toDold, Map<String, Boolean> activities)
clearActivities(String toDold)
```

Parte di

```
loadUserBoardsAndTodos()
```

 per il caricamento delle attività.

- Condivide (ToDo → SharedToDo)
"Un ToDo può essere condiviso con 0 o più utenti"
Rappresentata dalla tabella shared_todos.

- Metodi:

```
shareToDo(String toDold, String sharedWithUsername)
```

```
removeToDoSharing(String toDold, String sharedWithUsername)
```

```
removeAllToDoSharing(String toDold)
```

```
getSharedUsernamesForToDo(String toDold)
```

La relazione è multi-a-molti (ToDo ↔ Utente) mediata dalla tabella shared_todos.

- Ha (ToDo → Proprietario)
"Ogni ToDo ha un proprietario (username)"
Il campo owner_username in todos identifica l'utente creatore del ToDo.
Usato per distinguere tra ToDo propri e quelli condivisi.
- Accede (User → Shared ToDo)
"Ogni utente può accedere a 0 o più ToDo condivisi con lui"
Implementato nella parte finale di

```
loadUserBoardsAndTodos(User user):
```


Il DAO carica i ToDo condivisi per un determinato utente e li assegna alla board originale, se presente.

Capitolo 2

Diagramma UML delle classi del dominio del problema

Di seguito forniamo un diagramma, realizzato secondo il formalismo UML, utilizzato per modellare il dominio del problema. Il diagramma delle classi rappresenta i principali concetti coinvolti nel sistema informativo per la gestione dei ToDo, evidenziando le entità, gli attributi rilevanti e le relazioni tra gli oggetti che compongono l'architettura logica dell'applicazione.

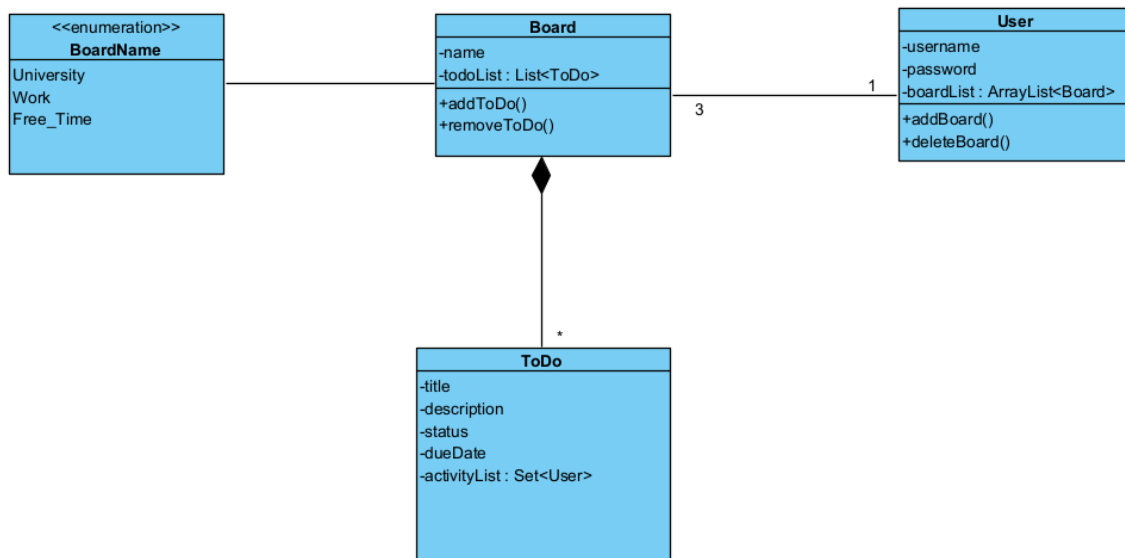


Figura 2.1

Capitolo 3

Diagramma UML di dettaglio delle classi nel dominio della soluzione

Il seguente diagramma, costruito secondo il formalismo UML, rappresenta il dominio della soluzione e descrive l'architettura logica del sistema informativo così come implementato. In particolare, il diagramma evidenzia le principali classi software dell'applicazione, con i relativi attributi e metodi, nonché le relazioni statiche tra di esse (come associazioni, aggregazioni e

dipendenze). Vengono inoltre riportati i legami con la struttura dei dati persistenti, offrendo una visione chiara della corrispondenza tra il modello concettuale e la sua realizzazione concreta nel codice. Questo modello è stato progettato seguendo i principi di separazione delle responsabilità e modularità, facilitando così la manutenibilità, l'estensibilità e il riutilizzo del software. In particolare, si è adottata un'architettura a livelli, che distingue logicamente tra interfaccia utente, logica di funzionamento e accesso ai dati.

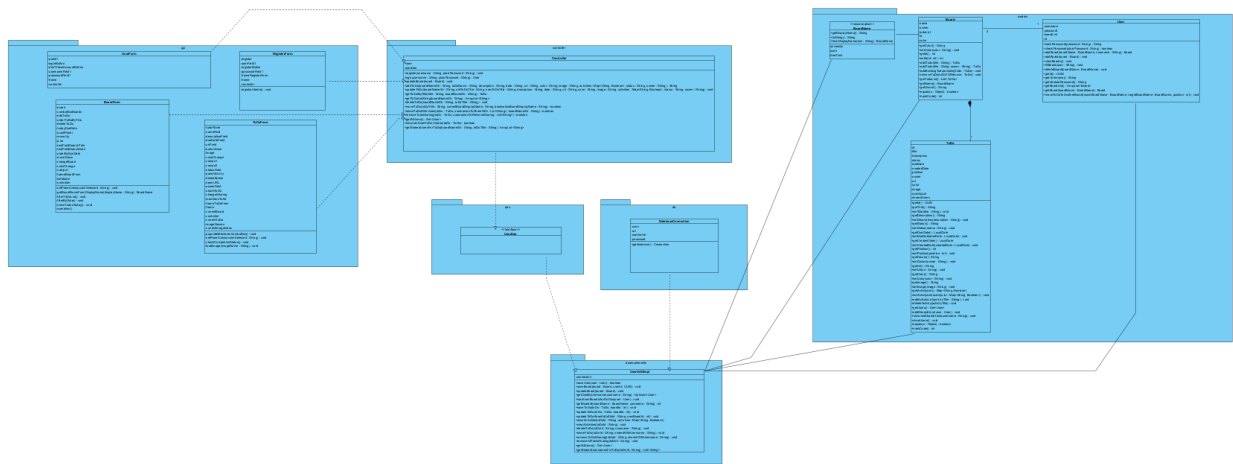


Figura 3.1

Capitolo 4

Sequence diagram di funzionalità a scelta

Di seguito viene presentato un diagramma di sequenza UML, selezionato per illustrare in modo puntuale una delle principali funzionalità offerte dall'applicativo. Tali diagrammi descrivono l'interazione dinamica tra gli oggetti del sistema durante l'esecuzione di specifici casi d'uso, evidenziando l'ordine temporale dei messaggi scambiati e la responsabilità dei diversi componenti. L'obiettivo è fornire una rappresentazione chiara del comportamento del sistema in

risposta a determinati eventi, mettendo in luce il flusso delle informazioni tra interfaccia utente, logica applicativa e livello di persistenza.

4.1 login(username, plainPassword)

Il seguente diagramma di sequenza descrive l'interazione tra i componenti del sistema durante l'esecuzione del metodo `login(username, plainPassword)`.

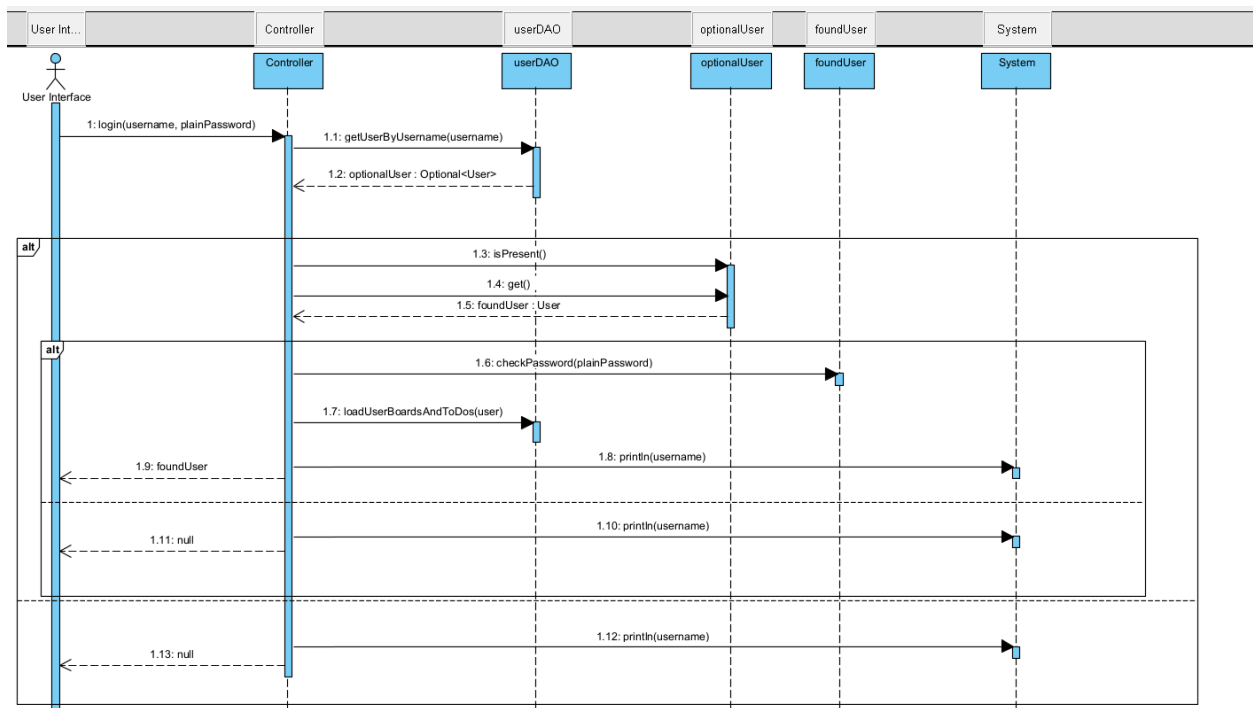


Figura 4.1

Capitolo 5

Manuale della GUI

5.1 Schermate introduttive

5.1.1 Interfaccia di login/registrazione

All'avvio dell'applicazione viene presentata una schermata di benvenuto, dalla quale l'utente può scegliere se procedere con il login o con la registrazione di un nuovo account. Questa interfaccia funge da punto di ingresso al sistema e garantisce una separazione chiara tra utenti già registrati e nuovi utenti.



Figura 5.1

5.1.2 Interfaccia di registrazione

L'interfaccia di registrazione consente la creazione di un nuovo account utente mediante l'inserimento obbligatorio delle seguenti informazioni: username, password.

È previsto un pulsante per tornare alla schermata di login in caso di errore o ripensamento. Il sistema verifica che tutti i campi siano compilati correttamente prima di consentire la registrazione, garantendo la consistenza dei dati utente.

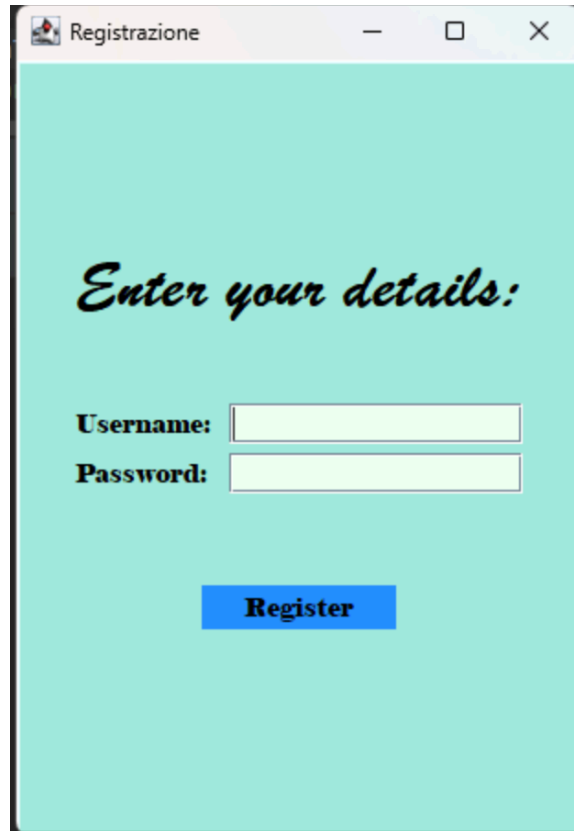


Figura 5.2

5.2 Schermate Personal Area

Dopo l'autenticazione l'utente viene reindirizzato alla schermata Personal Area. Qui è possibile gestire le proprie board, e i ToDo all'interno di esse, è possibile tramite i pulsanti nella sidebar di creare, modificare e cancellare i propri ToDo. L'utente inoltre è in grado di personalizzare la schermata di ogni board, di filtrare i ToDo (per esempio visualizzando esclusivamente quelli che scadranno nella giornata odierna) e condividere il proprio ToDo con altri utenti.

Sulla parte superiore della schermata è possibile eseguire operazioni di ricerca, tramite il titolo del ToDo, ed eventualmente con la sua data di scadenza.

Da notare che il secondo elemento della lista è rappresentato da un font di colore rosso. Il font rappresenta che il ToDo non è stato completato prima della scadenza imposta dall'utente.



Figura 5.3

5.3 Schermate dei ToDo

La schermata “Edit ToDo” rappresenta l’interfaccia grafica dedicata alla modifica dettagliata di un’attività (ToDo) all’interno dell’applicazione. La finestra si presenta con un layout a due colonne: in alto sinistra è presente un riquadro che raccoglie tutti i principali campi editabili dell’attività, mentre sulla destra sono disponibili i controlli relativi alle funzionalità aggiuntive, accompagnati da un’immagine decorativa, è possibile cliccare sull’immagine e navigare sui diversi tipi di immagine che l’utente può scegliere in base a cosa esso pensi che rappresenti più accuratamente il proprio ToDo.

I campi compilabili includono il nome del ToDo, la descrizione, la data di scadenza (formattata in stile europeo), un URL, lo stato corrente e il nome del proprietario.

Al centro della finestra si trova un’area bianca destinata alla visualizzazione e gestione delle sotto-attività (activity list). Sulla destra, una serie di pulsanti consente all’utente di interagire con il ToDo: è possibile aprire il link associato nel browser di sistema, eliminare una sotto-attività, avviare o modificare la condivisione del ToDo con altri utenti e selezionare un colore per identificare visivamente l’attività. Un menu a tendina consente inoltre di visualizzare l’elenco degli utenti con cui l’attività è condivisa, mostrando il proprietario come predefinito.

In fondo alla colonna destra è presente il pulsante “Save”, che consente di salvare tutte le modifiche apportate.

Edit ToDo

Name: mary

Description: prova

Due Date: 19/07/2025


URL: https://it.pinterest.com/

Status: Complete

Owner: mary evans

☒ primo

☒ secondo



Delete Activity

Open Url

Share ToDo

Change Sharing

mary evans (Owner) ▼

Orange ▼

Save

Figura 5.4

Menu che mostra in una lista gli utenti disponibili per la condivisione del ToDo.

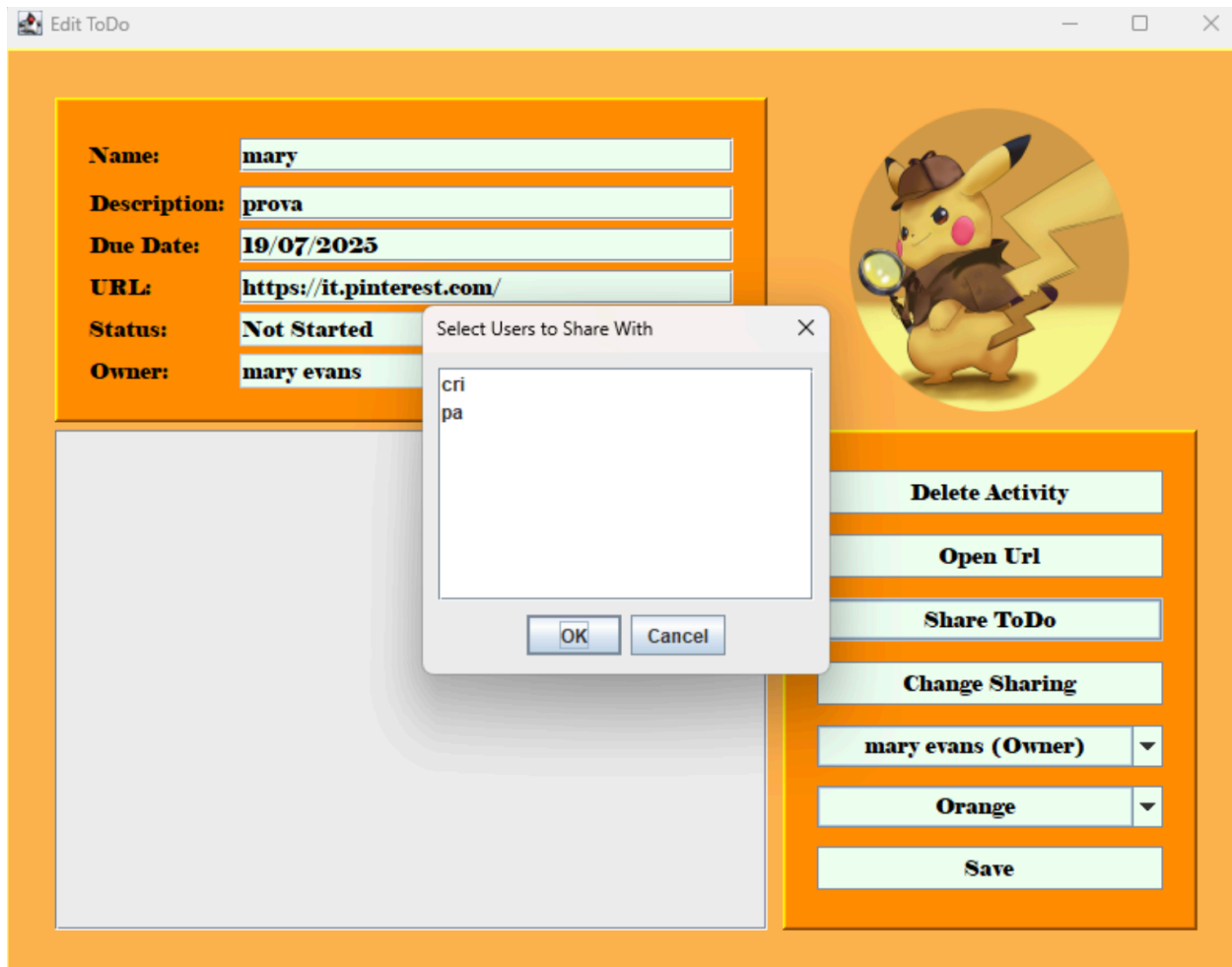


Figura 5.5

Se un utente che non risulta essere il *owner* del ToDo, esso avrà una visualizzazione leggermente diversa della schermata. Da notare che alcuni campi risultano essere grigi, perché sono modificabili soltanto dall'*owner*.

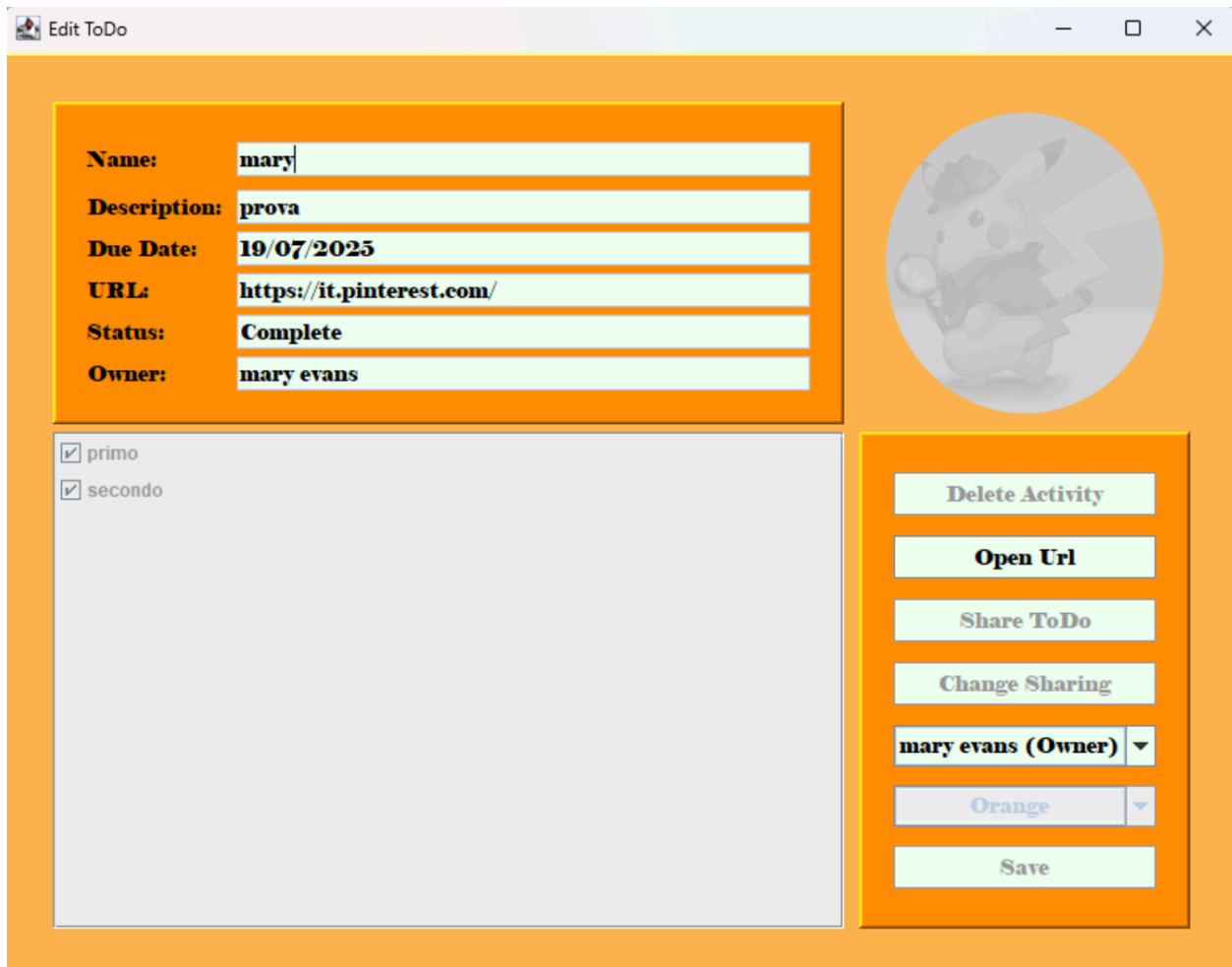


Figura 5.6

Capitolo 6

Gestioni degli errori

Al fine di garantire l'affidabilità e la coerenza dei dati inseriti, l'applicazione implementa un sistema di validazione preventiva sui campi di input presenti nelle schermate di login, registrazione e inserimento dati. In particolare, l'interfaccia fornisce messaggi di errore informativi e contestuali nel caso in cui l'utente inserisca valori non conformi ai vincoli richiesti. Questa logica di controllo consente di intercettare eventuali anomalie in fase di input, evitando comportamenti imprevisti o compromissioni del flusso applicativo. I messaggi sono mostrati in tempo reale o mediante finestre modali, contribuendo a migliorare l'usabilità e la robustezza del sistema.

