RELAZIONE PROGETTO D'ESAME CORSO DI PROGRAMMAZIONE 3

Università degli studi di Napoli Parthenope, Dipartimento di scienze e Tecnologie

Luca Maiuri: MAT.0124001418

January 14, 2023

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUZIONE

Il progetto presentato vuole imitare il comportamento di una webApp che gestisce i progetti personali di un utente e il loro progresso, inoltre fornire informazioni sulle quantità da impiegare di volta in volta per raggiungere l'obbiettivo prefissato. Non avendo a disposizione uno spazio online per la gestione della webApp, si è optato nel riprodurre la maggior parte delle funzionalità della webApp in un ambiente locale, utilizzando un database locale per la memorizzazione dei dati e anche come supporto per la gestione degli utenti.

2 REQUISTI

Il progetto soddisfa i seguenti requisiti:

1. Login, logout e registrazione di un utente.
2. Modifica dei dati dell'utente, inclusa la cancellazione.
3. Visualizzazione dei progetti dell'utente, e delle informazioni relative a essi.
4. Aggiunta, eliminazione e modifica di un progetto e delle sue informazioni.

Il tutto è accompagnato da un GUI grafica minimale ma funzionale.

2.1 Implementazione dei requisiti

I requisiti sono stati implementati utilizzando il linguaggio di programmazione Java; la gui è stata implementata utilizzando JavaFX. La gestione dei dati è affidata a un database locale, in particolare PostgreSQL, che viene utilizzato tramite JDBC.

3 DIAGRAMMA DELLE CLASSI

Di seguito è riportato il diagramma delle classi del progetto, che mostra le relazioni tra le classi e le interfacce implementate. Tutti i diagrammi sono stati realizzati tramite il sistema integrato nell'IDE jetBrains Intellij IDEA. La struttura dei pacchetti del progetto a cui fanno riferimento le classi è riportata come segue:

- org.project
  - org.project.core
    - org.project.core.adapters
  - org.project.database
  - org.project.ORM
  - org.project.GUI

3.1 org.project.core

Il pacchetto org.project.core[1] contiene le classi che implementano la funzionalità del progetto che riguardano la gestione delle istanze sia degli utenti che della sessione con permessi privilegiati al database, inoltre contiene le classi che contengono la logica di creazione e modifica dei progetti.

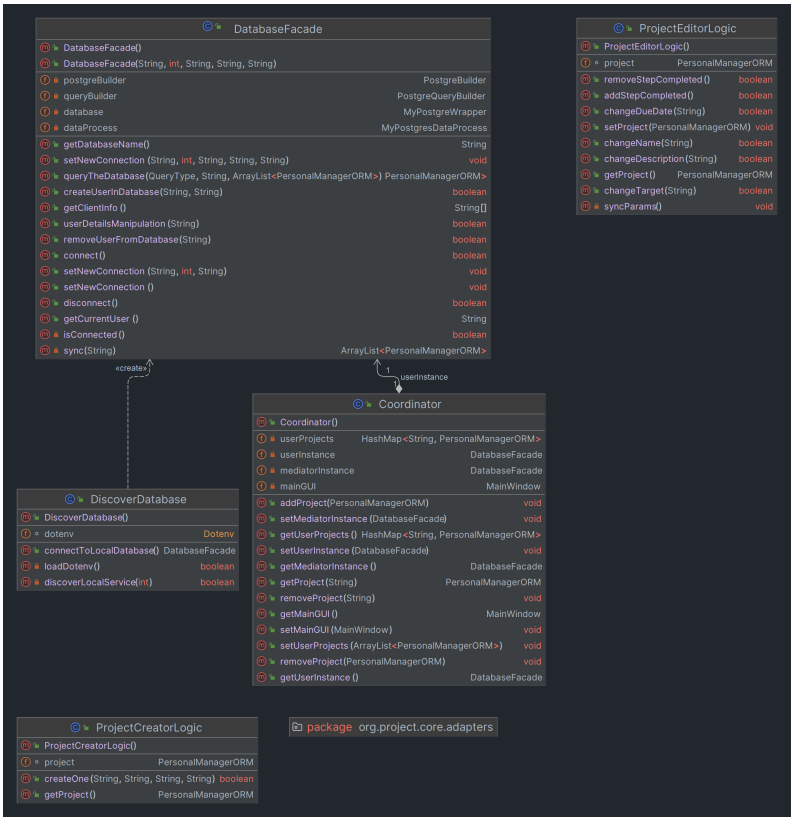

Figure 1: Diagramma delle classi del pacchetto org.project.core

3.2 org.project.database

Il pacchetto org.project.database[2] contiene le classi che implementano l'interazione con il database: connessione, creazione e modifica delle tabelle, inserimento, modifica e cancellazione dei dati tramite operazioni SQL/CRUD. Si tratta principalmente di classi wrapper per la libreria JDBC.

3.3 org.project.ORM

Il pacchetto org.project.ORM[3] contiene le classi che implementano l'object relational mapping, ovvero la mappatura tra le classi Java e le tabelle del database, ed i metodi per la gestione dei dati da essi contenuti.

3.4 org.project.GUI

Il pacchetto org.project.GUI[4] contiene le classi che implementano la GUI del progetto.

3.5 org.project.adapters

Il pacchetto org.project.core.adapters[5] contiene le classi che implementano l'adattamento dei dati tra le classi Java e il database, in particolare la conversione dei dati da e verso le classi Java e le tabelle del database.

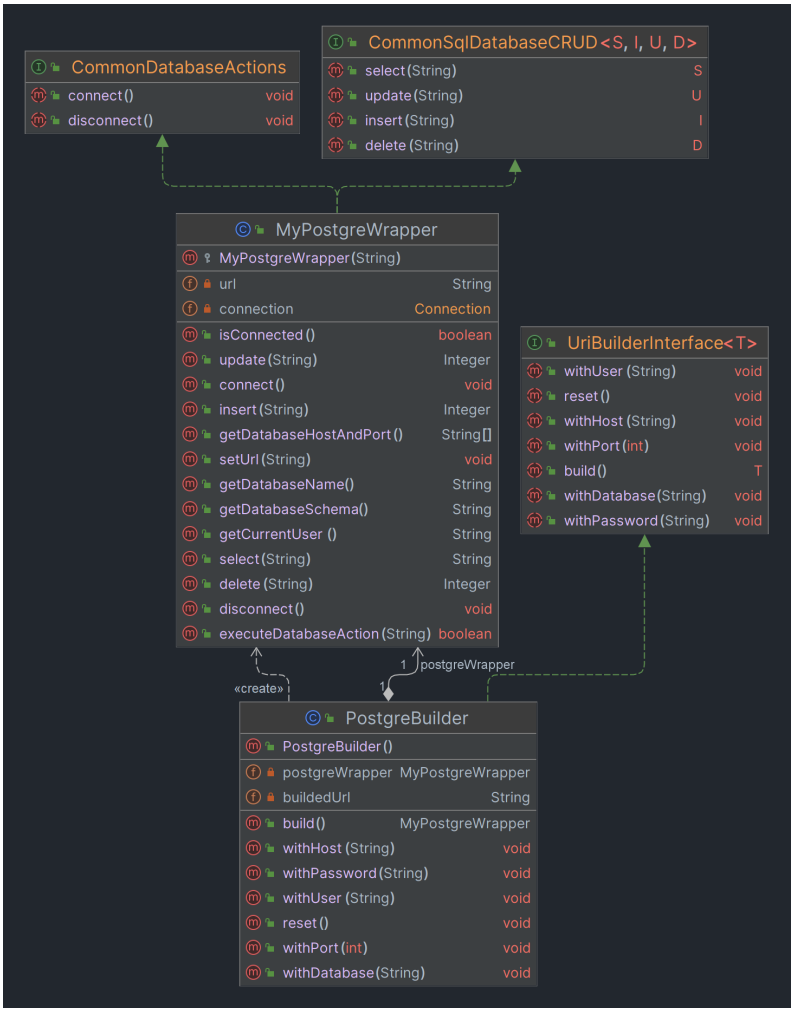

Figure 2: Diagramma delle classi del pacchetto org.project.database

4 DETTAGLI IMPLEMENTAZIONE

Questa sezione descrive i dettagli implementativi del progetto a livello di codice, e le scelte effettuate durante lo sviluppo a livello di architettura.

4.1 Architettura del progetto

Il progetto è stato sviluppato utilizzando il paradigma di programmazione orientata agli oggetti, è stato fatto utilizzo di alcuni design pattern, ove è stato possibile si è cercato di aderire ai principi SOLID. L'intero progetto verte sull'esistenza di un database raggiungibile tramite composizione di un url a partire da dati pre-esistenti, nel nostro caso un database PostgreSQL in localhost, che viene utilizzato come database di riferimento per la gestione dei dati e anche come autenticazione per l'accesso al programma. Il programma in se e per se può essere ridotto a un semplice client che si interfaccia con il database, ne manipola i dati e li visualizza in una GUI, questo è dovuto all'idea di partenza di "imitare" un webClient il cui scopo era interfacciarsi con un webApp nella rete. In alcune sue parti il programma è stato sviluppato in modo da essere facilmente estendibile, in particolare la parte relativa ai wrapper del database, altre parti come la GUI risultano più difficili(se non impossibili) da estendere o modificare.

4.2 Design pattern

Si è fatto uso di alcuni design pattern, nello specifico:

- **Builder**: implementato per la creazione dell'url del database e delle stringhe di query, con metodi in cascata, uno snippet del codice per la costruzione di query SQL è riportato in nella Lst.[1].
- **Factory**: implementato per la creazione di oggetti di tipo ORM specifici per un dato tipo di tabella e database [2]
- **Adapter**: implementato per la conversione dei dati tra le classi Java e le tabelle del database tramite oggetti creati tramite l'ORM factory, tramite l'interfaccia *AdapterDataDB.java* vengono implementati dei metodi per la conversione dei dati nella classe concreta *MyPostgresDataProcess.java* frammenti di codice delle due classi sono riportati nella Lst.[3]

Listing 1: Frammento di codice da PostgreQueryBuilder.java

```
        ...

        /**
        * Builds the query depending on the query type
        *
        * @return the object with a built query set
        */
        public PostgreQueryBuilder buildQuery() {
            this.query = switch (this.type) {
            case INSERT -> addColumnsAndValues(insert);
            case UPDATE -> addColumnsAndValues(update);
            case DELETE -> addColumnsAndValues(delete);
            case SELECT -> select.replace("tablename", this.table);
            };
            return this;
        }

        public String getQuery() {
            return query;
        }
        ...

```

Listing 2: FactoryORM.java

```java
package org.project.ORM;

/**
 * Factory Class for ORM classes.
 *
 * @author Luca Maiuri
 */
@SuppressWarnings("unused")
public class FactoryORM {
    private CommonORM orm;

    public FactoryORM() {
        this.orm = null;
    }

    public CommonORM getORM() {
        return this.orm;
    }

    /**
     * Build a specific ORM class.
     * <p></p>
     * The ORM class is built according to the type passed as parameter.
     *
     * @param ormName The name of the ORM class to build.
     * @see ORMtypes
     */
    public void buildORM(ORMtypes ormName) {
        if (ormName == ORMtypes.PERSONALMANAGERORM) {
            this.orm = new PersonalManagerORM();
        } else {
            this.orm = null;
        }
    }
}
```

Listing 3: Interfaccia AdapterDataDB

```java
package org.project.core.adapters;

/**
 * DB data adapter.
 *
 * @param <T> The type of the data to adapt.
 */
public interface AdapterDataDB<T> {
    T packData();

    String unpackData(T data);
}
```

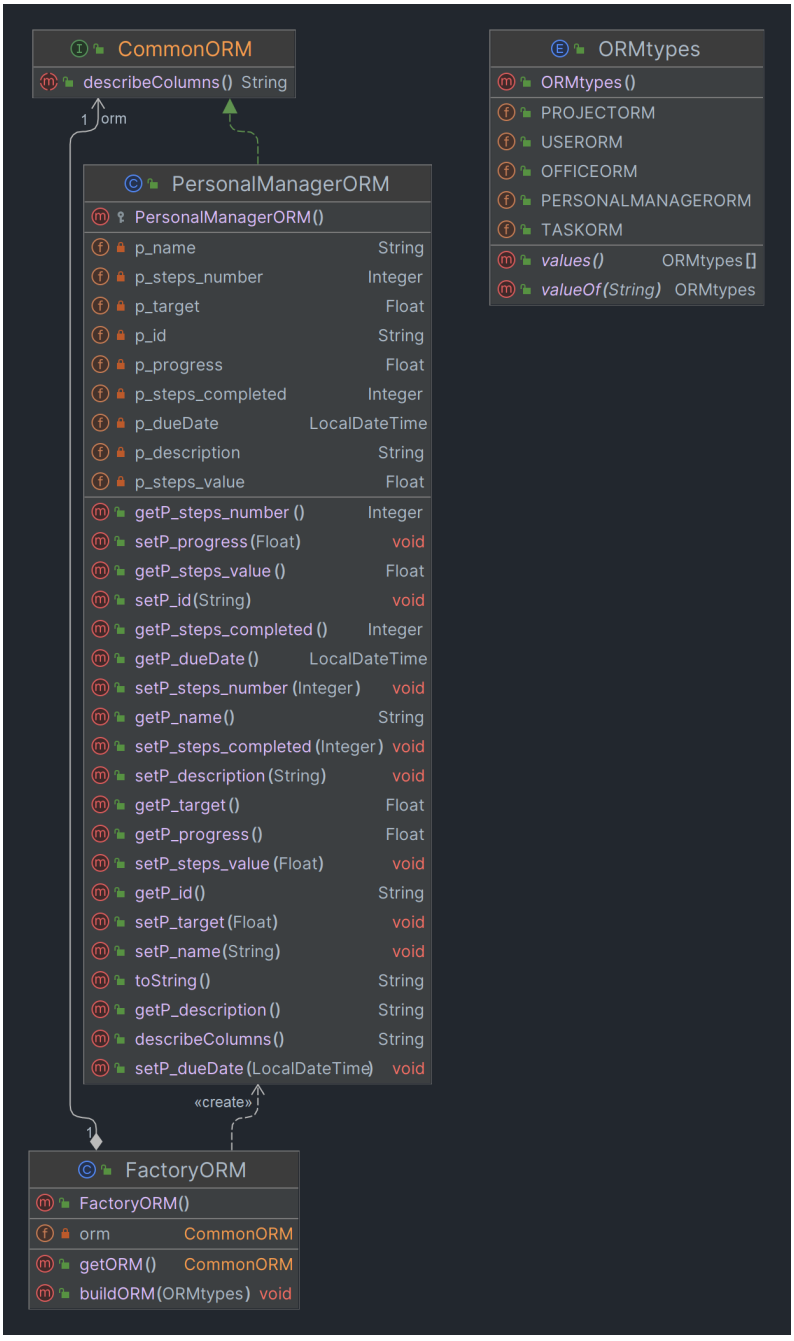

Figure 3: Diagramma delle classi del pacchetto org.project.ORM

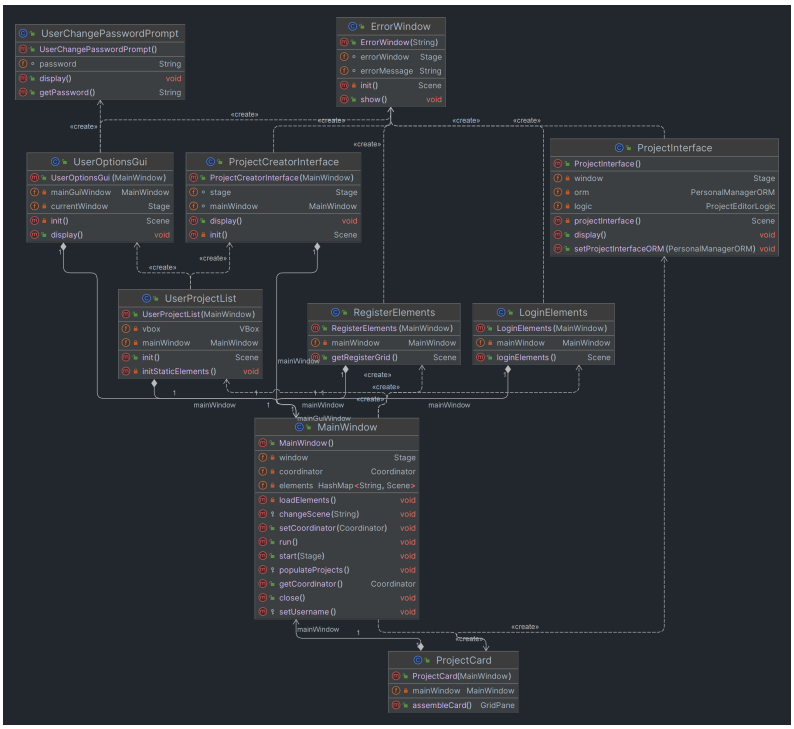

Figure 4: Diagramma delle classi del pacchetto org.project.GUI

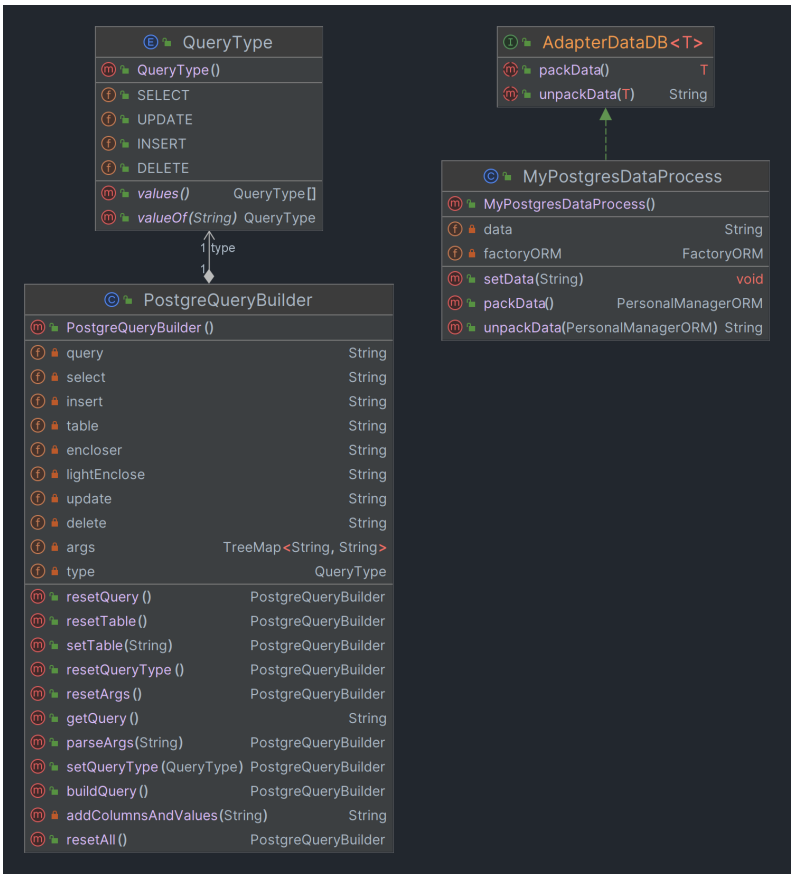

Figure 5: Diagramma delle classi del pacchetto org.project.core.adapters