ODD

Object Design Document

SearchQS

Riferimento	ODD_ver.1.0
Versione	1.0
Data	11/04/2024
Destinatario	Prof. Fabio Palomba
Presentato da	Gianluca Scisciolo
Approvato da	

Revision History

Data	Versione	Descrizione	Autori
05/12/2023	0.1	Prima stesura	GS
05/12/2023	0.2	Scrittura sezione 1	GS
06/12/2023	0.3	Scrittura sezione 2	GS
13/12/2023	0.4	Scrittura sezione 3	GS
14/12/2023	0.5	Scrittura sezione 4	GS
15/12/2023	0.6	Scrittura sezione 5	GS
15/12/2023	0.7	Revisione documento	GS
25/01/2024	0.8	Revisione documento	GS
06/03/2024	0.9	Revisione documento	GS
11/04/2024	1.0	Revisione finale documento	GS

Team members

Nome	Ruolo nel progetto	Acronimo	Informazioni di contatto
Gianluca Scisciolo	Software Engineer	GS	g.scisciolo@studenti.unisa.it

Sommario

1. Introduzione

Il sistema che si vuole realizzare ha come obiettivo l'analisi di un sistema quantistico per poter individuale i quantum code smells presenti, ovvero, i seguenti 8:

- Non-parameterized Circuit (NC).
- no-alignment between the Logical and Physical Qubits (LPQ).
- Idle Qubits (IdQ).
- Initialization of Qubits (IQ).
- Intermediate Measurement (IM).
- Long Circuit (LC).
- Repeated set of Operations on Circuit (ROC).
- use of Customized Gates (CG).

L'utente può eseguire delle analisi (ogni analisi è composta da analisi statica e analisi dinamica) sui vari sistemi quantistici. Una analisi coinvolge solo un tipo di transpilazione oppure nessuna transpilazione quindi, se per esempio sceglie 3 transpilazioni e la modalità senza transpilazione allora vengono eseguite in totale 4 analisi ognuna delle quali è costituita da analisi statica più analisi dinamica. Le transpilazioni offerte dal sistema sono le seguenti:

- original.
- ibm_perth.
- ibm_sherbrooke.
- rpcx.
- simple.

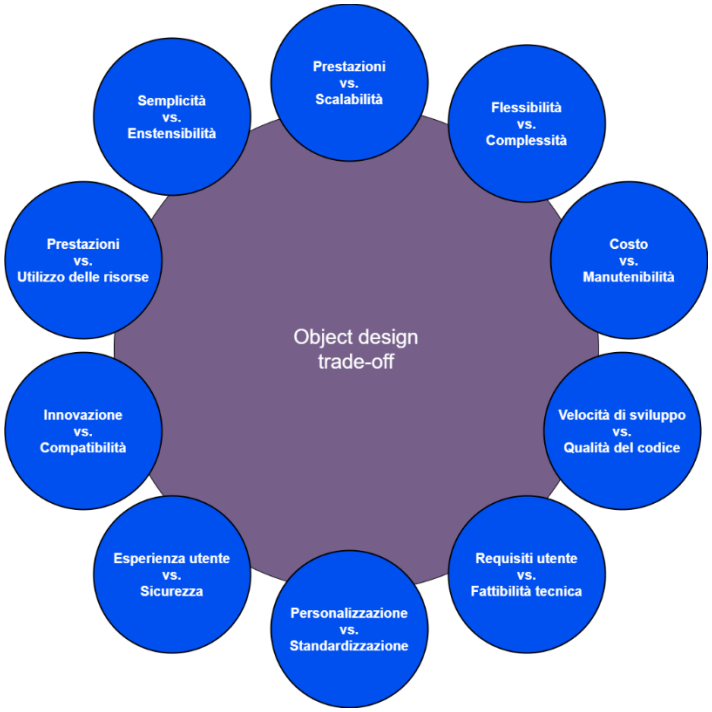
1.1. Object Design Goals

Gli Object Design Goals che si vogliono ottenere sono i seguenti:

- robustezza: si deve cercare di rendere il sistema robusto reagendo a situazioni non previste tramite il controllo degli errori e la gestione delle eccezioni.
- incapsulamento: si deve cercare di rendere alcuni dettagli implementativi sottoforma di black-box tramite l'utilizzo di interfacce.

1.2. Object Trade-Off:

Di seguito sono riportati gli Object Trade-off rappresentati graficamente dal seguente schema:



- **Prestazioni vs. Scalabilità:** si deve cercare di realizzare un sistema in grado di gestire una buona scalabilità ovvero, gestire dei carichi di lavoro con tanti utenti senza sacrificare di molto le prestazioni.
- **Flessibilità vs. Complessità:** si deve cercare di ottenere una buona flessibilità del software anche se l'esecuzione delle analisi di un sistema quantistico richiede l'utilizzo di molti metodi che rendono il codice più impegnativo da comprendere.
- **Costo vs. Manutenibilità:** l'unico costo individuato nel sistema riguarda il tempo di progettazione ed implementazione del sistema mentre, è molto importante cercare di ottenere un sistema abbastanza semplice da manutenere.
- **Velocità di sviluppo vs. Qualità del codice:** si deve cercare di ottenere una buona qualità del codice per ottenere un sistema abbastanza robusto e manutenibile.
- **Requisiti utente vs. Fattibilità tecnica:** si deve cercare di ottenere un sistema in cui sono implementati tutti i requisiti funzionali richiesti.
- **Personalizzazione vs. Standardizzazione:** si deve cercare di ottenere un buon compromesso tra personalizzazione e standardizzazione.
- **Esperienza utente vs. Sicurezza:** si deve cercare di rendere il sistema semplice da utilizzare e si deve anche cercare di ottenere un sistema sicuro tramite la cifratura delle password nel database.
- **Innovazione vs. Compatibilità:** il sistema da realizzare è abbastanza innovativo poiché la computazione quantistica è una nuova tecnologia in continua mutazione e in continuo miglioramento.
- **Prestazioni vs. Utilizzo delle risorse:** quando viene caricato un nuovo sistema quantistico bisogna caricarlo in una sottocartella isolata ottenendo un aumento delle risorse come la potenza di elaborazione.
- **Semplicità vs. Estensibilità:** si deve cercare di rendere il sistema semplice da estendere per dei futuri requisiti funzionali da progettare e implementare.

1.3. Linee guida per la scrittura dei file

In questa sezione sono riportate le linee guida che saranno utilizzate nella scrittura dei vari file scritti in Python, HTML e JavaScript.

- Python: https://peps.python.org/pep-0008/#class-names
- HTML: https://www.w3schools.com/html/html5_syntax.asp
- JavaScript: https://www.w3schools.com/js/js_conventions.asp

Si è deciso di adottare le convenzioni per ogni linguaggio in maniera tale da rendere il codice più leggibile e facile da gestire.

1.4. Definizioni, acronimi e abbreviazioni

In questa sezione sono riportati alcune definizioni presenti nel documento:

- package: raggruppamento di moduli Python e/o file.
- modulo Python: file Python che contiene metodi e/o classi e/o variabili e/o interfacce.
- Interfaccia: insieme di signature delle operazioni offerte dalla classe.
- camelCase: scrittura di frasi tali che la prima parola inizia con la lettera minuscola e le successive parole con la lettera maiuscola.
- PascalCase: scrittura di frasi tali che tutte le parole iniziano con la lettera maiuscola.
- snake_case: scrittura di frasi tali che tutte le parole iniziano con la lettera minuscola e sono separate da un trattino basso (_).
- kebab-case: scrittura di frasi tali che tutte le parole iniziano con la lettera minuscola e sono separate da un trattino (-).

1.5. Riferimenti

di seguito sono riportati altri documenti utili durante la lettura.

- Statement Of Work (SOW).
- Business Case (BC).
- Requirements Analysis Document (RAD).
- System Design Document (SDD).
- DataBase Design Document (DBDD).
- Test Plan (TP).
- Test Case Specification (TCS).
- Test Incident Report (TIR).
- Test Incident Report Table (TIRT).
- Test Summary Report (TSR).
- Manuale Di Installazione (MDI).
- Manuale Utente (MU).

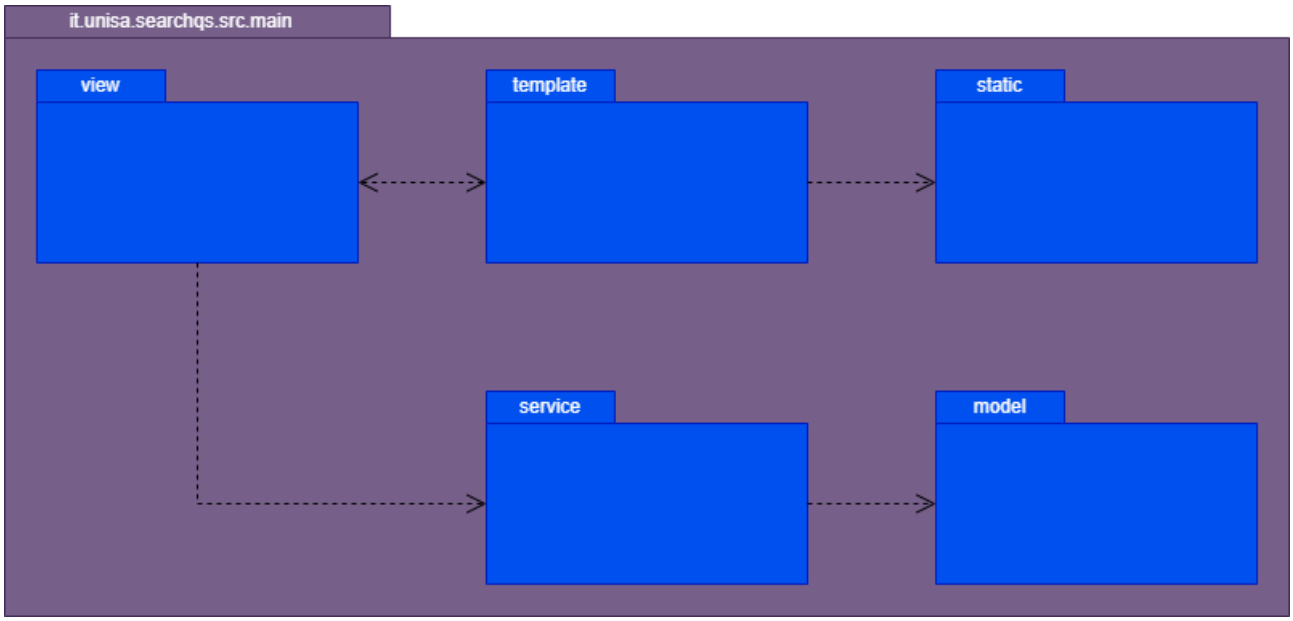
2. Packages

In questa sezione viene riportato la suddivisione del sistema SearchQS in packages in base a quanto definito nel documento SDD. Questa suddivisione si basa sulle scelte architetturali adottate:

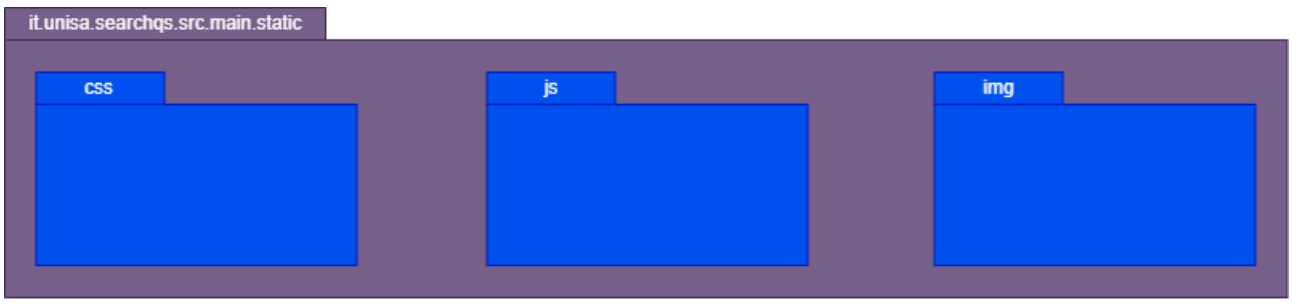
- it
    - unisa
        - searchqs
            - src
                - main
                    - model
                    - service
                    - static
                    - template
                    - view
                - test
            - projectdocs
            - tutorial

Package it.unisa.searchqs.src.main

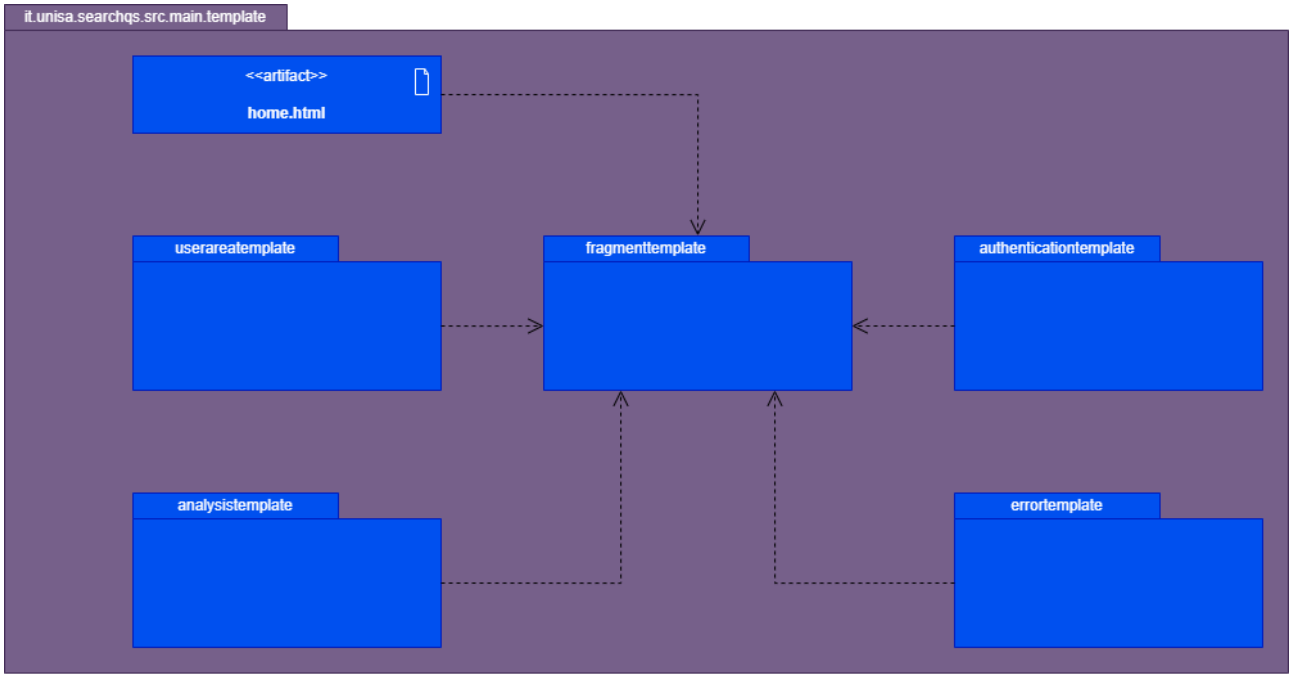
Di seguito è riportata la struttura del package it.unisa.searchqs.src.main definito precedentemente compreso le loro dipendenze:



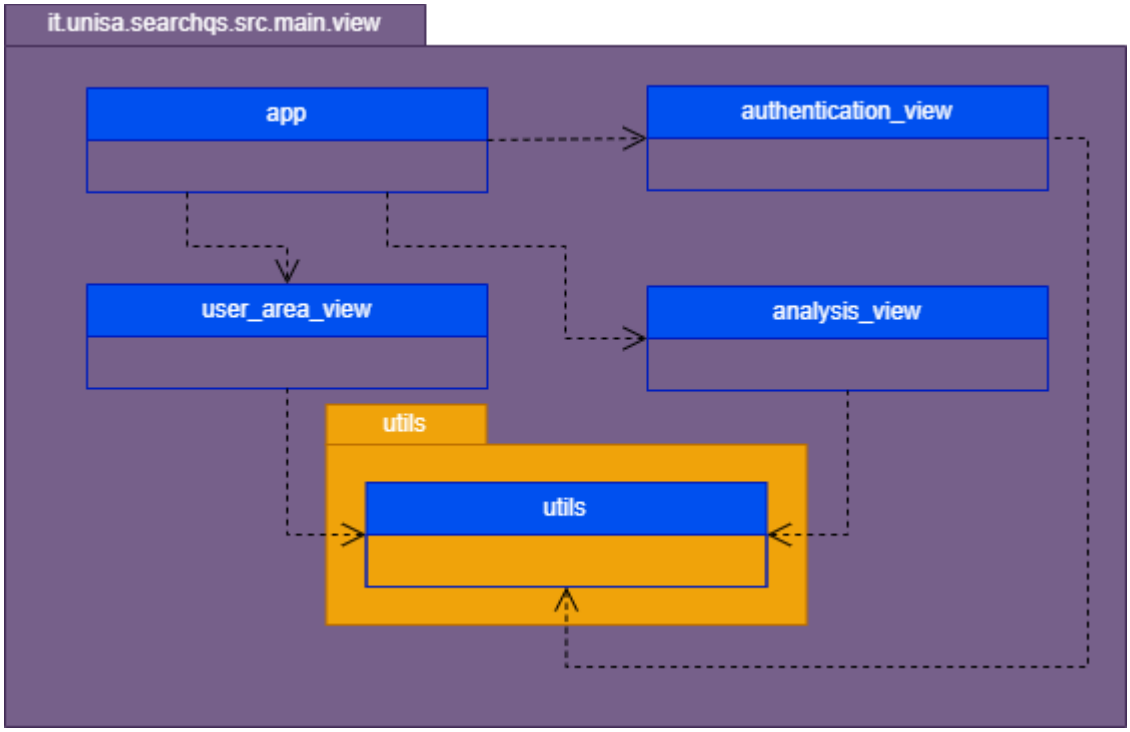
Package it.unisa.searchqs.src.main.static



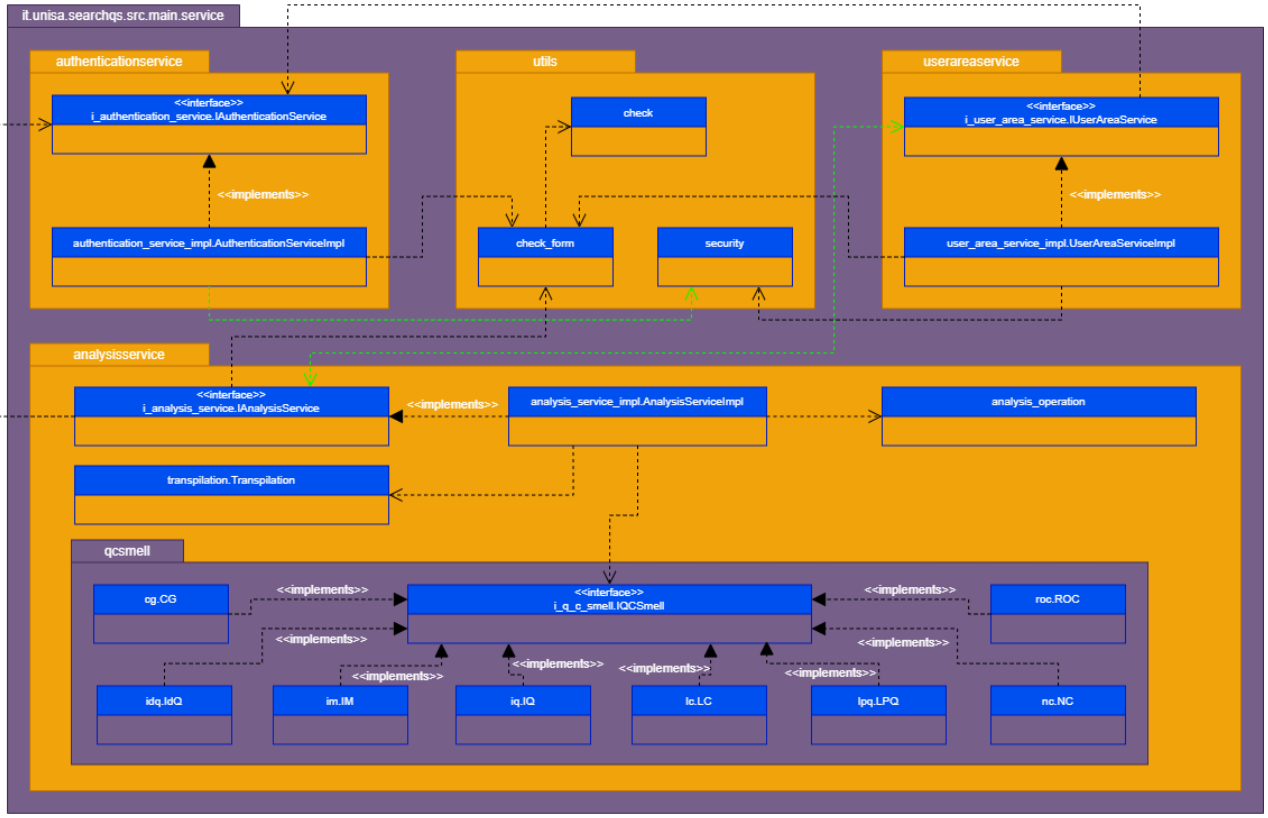
Package it.unisa.searchqs.src.main.template



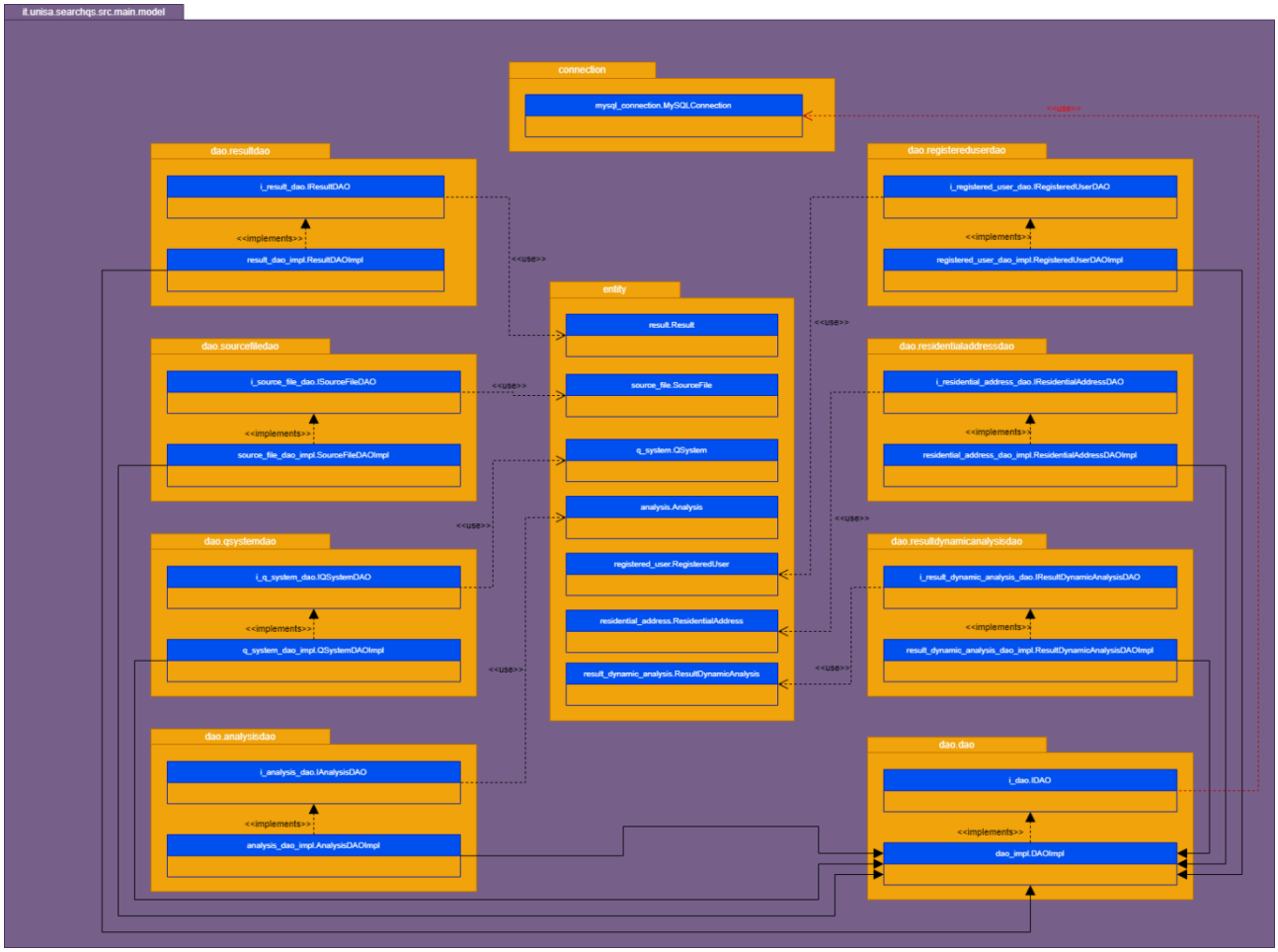
Package it.unisa.searchqs.src.main.view



Package it.unisa.searchqs.src.main.service



Package it.unisa.searchqs.src.main.model



3 Class Interfaces

Di seguito sono riportate le interfacce di ciascun package del modulo it.unisa.searchqs.src.main.service.

3.1 Class Interfaces package service:

3.1.1 Class Interfaces package authenticationservice

it.unisa.searchqs.src.main.service.i_authentication_service.IAuthenticationService

Classe	IAuthenticationService
Descrizione	Questa classe permette di gestire le operazioni relative all'autenticazione.
Metodi	+ display_form_registration(self): Dictionary + registration(self, attributes: Tuple): Dictionary + display_form_login(self): Dictionary + login(self, attributes: Tuple): Dictionary + logout(self): void
Invariante di classe	/

Metodo	+ display_form_registration(self): Dictionary
Descrizione	Questo metodo permette la visualizzazione del form di registrazione.
Pre-condizione	/
Post-condizione	/

Metodo	+ registration(self, attributes: Tuple): Dictionary
Descrizione	Questo metodo permette la registrazione alla web application SearchQS.
Pre-condizione	Context: IAuthenticationService:: registration(self, attributes: Tuple) Pre: attributes deve contenere i seguenti dati: "name_ru": String, "surname_ru": String, "gender_ru": char, "birthdate_ru": String, "city_birthplace_ru": String, "nation_birthplace_ru": String, "nationality_ru": String, "profession_ru": String, "num_cellphone_ru": String, "email_ru": String, "password_ru": String, "name_ra": String, "number_ra": int, "city_ra": String, "province_ra": String, "cap_ra": String, "confirm_password_ru": String AND session == {}
Post-condizione	/

Metodo	+ display_form_login(self): Dictionary
Descrizione	Questo metodo permette la visualizzazione del form di login.
Pre-condizione	/
Post-condizione	/

Metodo	+ login(self, attributes: Tuple): Dictionary
Descrizione	Questo metodo permette il login alla web application SearchQS.
Pre-condizione	Context: IAuthenticationService:: login(self, attributes: Tuple) Pre: attributes deve contenere i seguenti dati: "email_ru": String, "password_ru": String AND session == {}
Post-condizione	Context: IAuthenticationService:: login(self, attributes: Tuple) Post: Se il login va a buon fine allora session deve contenere le seguenti chiavi: 'is_logger', 'email' e 'actor', dove: 'is_logger' = True e 'actor' = 'registered_user'. altrimenti session = {}

Metodo	+ logout(self): void
Descrizione	Questo metodo permette il logout dalla web application SearchQS.
Pre-condizione	/
Post-condizione	Context: IAuthenticationService:: logout(self) Post: session = {}

3.1.2 Class Interfaces package user area service

it.unisa.searchqs.src.main.service.i_user_area_service.IUserAreaService

Classe	IUserAreaService
Descrizione	Questa classe permette di gestire le operazioni relative all'area utente.
Metodi	+ display_user_area(self): Dictionary + display_form_deletion_account(self): Dictionary + deletion_account(self): Dictionary + display_personal_data(self): Dictionary + modification_personal_data(self, attributes: Tuple): Dictionary
Invariante di classe	/

Metodo	+ display_user_area(self): Dictionary
Descrizione	Questo metodo permette la visualizzazione dell'area utente.
Pre-condizione	/
Post-condizione	/

Metodo	+ display_form_deletion_account(self): Dictionary
Descrizione	Questo metodo permette la visualizzazione del form per l'eliminazione dell'account.
Pre-condizione	/
Post-condizione	/

Metodo	+ deletion_account(self): Dictionary
Descrizione	Questo metodo permette l'eliminazione dell'account.
Pre-condizione	Context: IUserAreaService:: deletion_account(self) Pre: session deve contenere le seguenti chiavi: 'email' e 'actor' dove 'actor' = 'registered_user'.
Post-condizione	Context: IUserAreaService:: deletion_account(self) Post: Se l'eliminazione dell'account va a buon fine allora session = {}

Metodo	+ display_personal_data(self): Dictionary
Descrizione	Questo metodo permette la visualizzazione dei dati personali.
Pre-condizione	Context: IUserAreaService:: display_personal_data(self) Pre: session deve contenere le seguenti chiavi: 'email' e 'actor' dove 'actor' = 'registered_user'.
Post-condizione	/

Metodo	+ modification_personal_data(self, attributes: Tuple): Dictionary
Descrizione	Questo metodo permette la modifica dei dati personali.
Pre-condizione	Context: IUserAreaService:: modification_personal_data(self, attributes: Tuple) Pre: session deve contenere le seguenti chiavi: 'email' e 'actor' dove 'actor' = 'registered_user'.
Post-condizione	/

3.1.3 Class Interfaces package analysis service

it.unisa.searchqs.src.main.service.i_analysis_service.IAnalysisService

Classe	IAnalysisService
Descrizione	Questa classe permette di gestire le operazioni relative alle analisi.
Metodi	+ display_analysis_area(self): Dictionary + display_names_transpilation(self): Dictionary + display_analyses_transpilation_selected(self, attributes: Tuple): Dictionary + display_form_loading_q_system(self, attributes: Tuple): Dictionary + loading_q_system(self, attributes: Tuple): Dictionary + execution_analyses(self, attributes: Tuple): Dictionary + display_analysis(self, attributes: Tuple): Dictionary + display_form_deletion_analysis(self, attributes: Tuple): Dictionary + deletion_analysis(self, attributes: Tuple): Dictionary
Invariante di classe	/

Metodo	+ display_analysis_area(self): Dictionary
Descrizione	Questo metodo permette la visualizzazione dell'area analisi.
Pre-condizione	/
Post-condizione	/

Metodo	+ display_names_transpilation(self): Dictionary
Descrizione	Questo metodo permette la visualizzazione del form contenente i nomi delle transpilazioni offerte dal sistema.
Pre-condizione	/
Post-condizione	/

Metodo	+ display_analyses_transpilation_selected(self, attributes: Tuple): Dictionary
Descrizione	Questo metodo permette la visualizzazione di una tabella contenente le analisi eseguite per una transpilazione selezionata oppure per nessuna transpilazione.
Pre-condizione	Context: IAnalysisService:: display_analyses_transpilation_selected(self, attributes: Tuple) Pre: attributes deve contenere i seguenti dati: 'name_transpilation': String AND session deve contenere le seguenti chiavi: 'email' e 'actor' dove 'actor' = 'registered_user'.
Post-condizione	/

Metodo	+ display_form_loading_q_system(self, attributes: Tuple): Dictionary
Descrizione	Questo metodo permette la visualizzazione del form per caricare un sistema quantistico.
Pre-condizione	Context: IAnalysisService:: display_form_loading_q_system(self, attributes: Tuple) Pre: attributes deve contenere i seguenti dati: 'name_transpilation': String
Post-condizione	/

Metodo	+ loading_q_system(self, attributes: Tuple): Dictionary
Descrizione	Questo metodo permette il caricamento di un sistema quantistico.
Pre-condizione	Context: IAnalysisService:: loading_q_system(self, attributes: Tuple) Pre: attributes deve contenere i seguenti dati:

	'file': una cartella compressa .zip, 'filename': il nome della cartella compressa .zip AND session deve contenere le seguenti chiavi: 'email' e 'actor' dove 'actor' = 'registered_user'.
Post-condizione	/
Metodo	+ execution_analyses(self, attributes: Tuple): Dictionary
Descrizione	Questo metodo permette l'esecuzione di una o più analisi per il sistema quantistico appena caricato.
Pre-condizione	Context: IAnalysisService:: execution_analyses(self, attributes: Tuple) Pre: attributes deve contenere i seguenti dati: 'id_qs': Integer, 'name_qs': String, 'optimization': Integer, 'files_json': JSON, 'transpilations_json': JSON, 'files_selected': List, 'transpilations_selected': List AND session deve contenere le seguenti chiavi: 'email' e 'actor' dove 'actor' = 'registered_user'.
Post-condizione	/
Metodo	+ display_analysis(self, attributes: Tuple): Dictionary
Descrizione	Questo metodo permette la visualizzazione completa di una analisi eseguita su un sistema quantistico.
Pre-condizione	Context: IAnalysisService:: display_analysis(self, attributes: Tuple) Pre: attributes deve contenere i seguenti dati: 'id_qs': Integer, 'name_qs': String, 'save_date': String, 'id_analysis': Integer, 'name_transpilation': String, 'optimization': Integer AND session deve contenere le seguenti chiavi: 'email' e 'actor' dove 'actor' = 'registered_user'.
Post-condizione	/
Metodo	+ display_form_deletion_analysis(self, attributes: Tuple): Dictionary
Descrizione	Questo metodo permette la visualizzazione del form per eliminare una analisi selezionata.
Pre-condizione	Context: IAnalysisService:: display_form_deletion_analysis(self, attributes: Tuple) Pre: Attributes deve contenere i seguenti dati: 'id_analysis': Integer, 'id_qs': Integer, 'name_transpilation': String, 'route': String, 'save_date_qs': String, 'name_qs': String.
Post-condizione	/
Metodo	+ deletion_analysis(self, attributes: Tuple): Dictionary
Descrizione	Questo metodo permette l'eliminazione di una analisi selezionata.
Pre-condizione	Context: IAnalysisService:: deletion_analysis(self, attributes: Tuple) Pre: attributes deve contenere i seguenti dati: 'id_analysis': Integer, 'name_transpilation': String, 'id_qs': Integer AND session deve contenere le seguenti chiavi: 'email' e 'actor' dove 'actor' = 'registered_user'.
Post-condizione	/

4 Design Patterns

In questa sezione verranno riportati i design pattern che verranno utilizzati nella web application che si sta realizzando.

Per ogni design pattern verranno riportati i seguenti punti:

- una breve introduzione teorica.
- il problema da risolvere all'interno del sistema SearchQS.
- una breve descrizione di come è stato risolto il problema nel sistema SearchQS.
- un grafico della struttura delle classi che implementano il design pattern.

Design pattern utilizzati:

- design pattern architetturali
    - MVTS (Model View Template Service).
    - DAO (Data Access Object)

MVTS (Model View Template Service):

Introduzione teorica:

MVTS è un design pattern architetturale molto utilizzato quando si ha a che fare con le web applications e in generale quando si utilizza il micro framework Flask.

MVTS è un estensione del MVT (Model View Template), infatti, viene aggiunto il package Service. Inoltre, viene aggiunto anche un altro package che non viene citato nella sigla del design pattern, ovvero, il package Static.

- Static: package che contiene i file statici, nel nostro caso, tutti i file CSS, JavaScript e tutte le immagini.
- Template: package che contiene tutti i file HTML e quindi i file che si occupano di interfacciare il sistema con l'utente. Il Template interagisce con il package Static per ottenere i suoi file e invia delle richieste HTTP al package View.
- View: package che si occupa di gestire le richieste di tipo HTTP che provengono dal livello View e le reindirizza al package service. Quindi, il package View interagisce con i package Template e Service.
- Service: package che si occupa della logica di business e, se necessario, interagisce con il package model. Quindi, il package service interagisce solo con il package Model.
- Model: package che gestire i dati persistenti di SearchQS presenti nel database. Quindi, il package Model interagisce solamente con il database.
- Inoltre, interagiamo con il DataBase tramite un RDBMS.

L'MVTS, nel nostro caso, implementa lo stile architetturale Three Tier formato dai livelli:

- Interface (Template, Static).
- Application (View).
- Storage (Service, Model, RDBMS, DataBase).

Problema da risolvere:

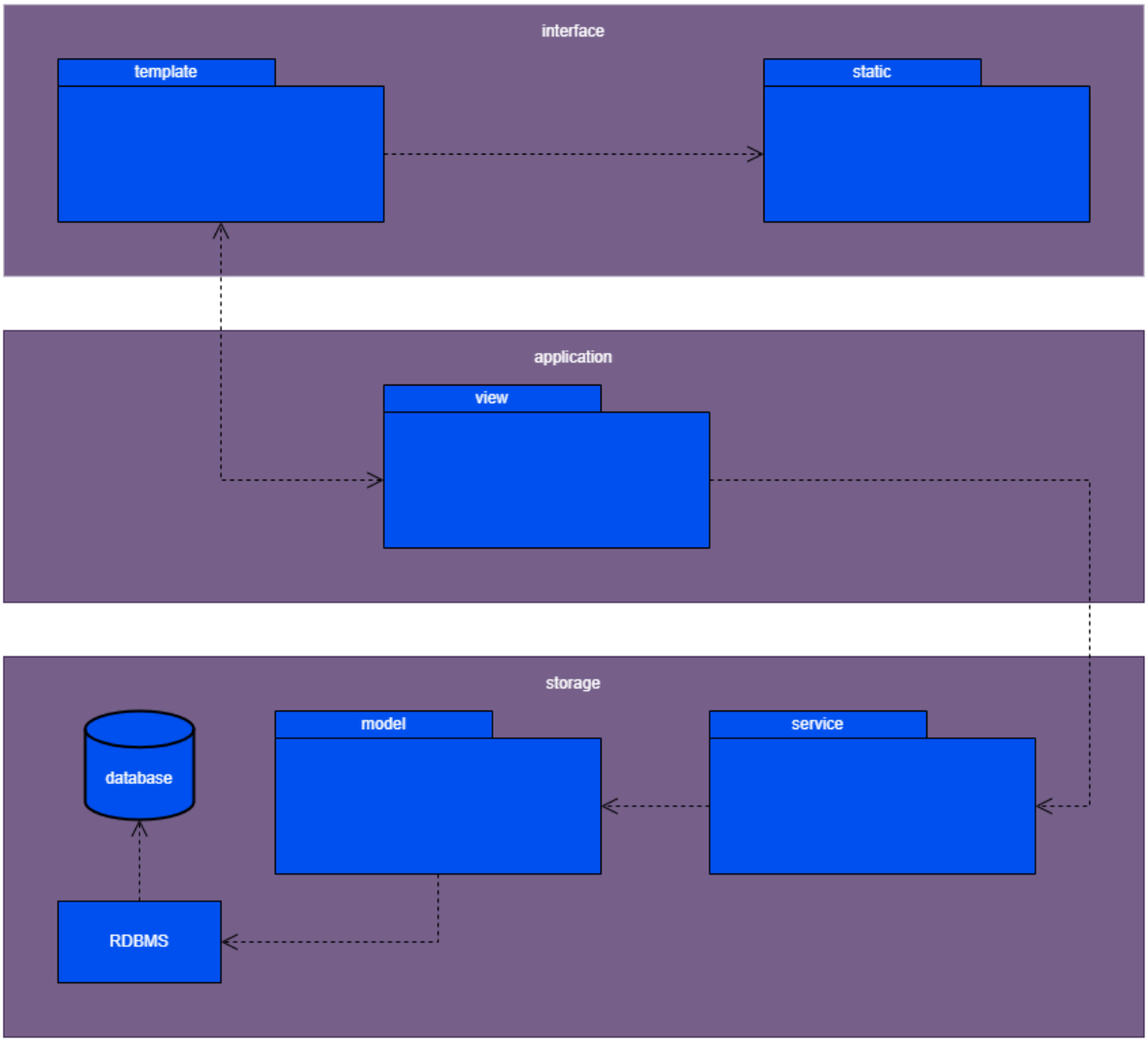
Nello sviluppo di una web application è molto importante cercare di separare la logica di presentazione dalla logica di elaborazione poiché questo ci permette di sapere dove toccare il codice se volessimo eseguire delle modifiche su di essa.

Come è stato risolto il problema:

È stato utilizzato il design pattern MVTS insieme allo stile architetturale Three Tier poiché aiutano a separare la logica di presentazione dalla logica di elaborazione migliorando anche alcune qualità, tra cui:

- La leggibilità del codice.
- La manutenzione del codice.
- Il riuso del codice.

Grafico della struttura delle classi:



DAO (Data Access Object):

Introduzione teorica:

Il DAO è un design pattern architetturale implementato nel package model e si occupa di interagire con il database. Verrà creato un DAO per ogni tabella del nostro database, questo perché, ogni DAO dovrebbe gestire solamente una singola tabella creando i cosiddetti metodi CRUD (Create Read Update Delete) per interagire su di essa.

Problema da risolvere:

SearchQS è una web application che deve gestire varie operazioni sul database in modo rapido e sicuro.

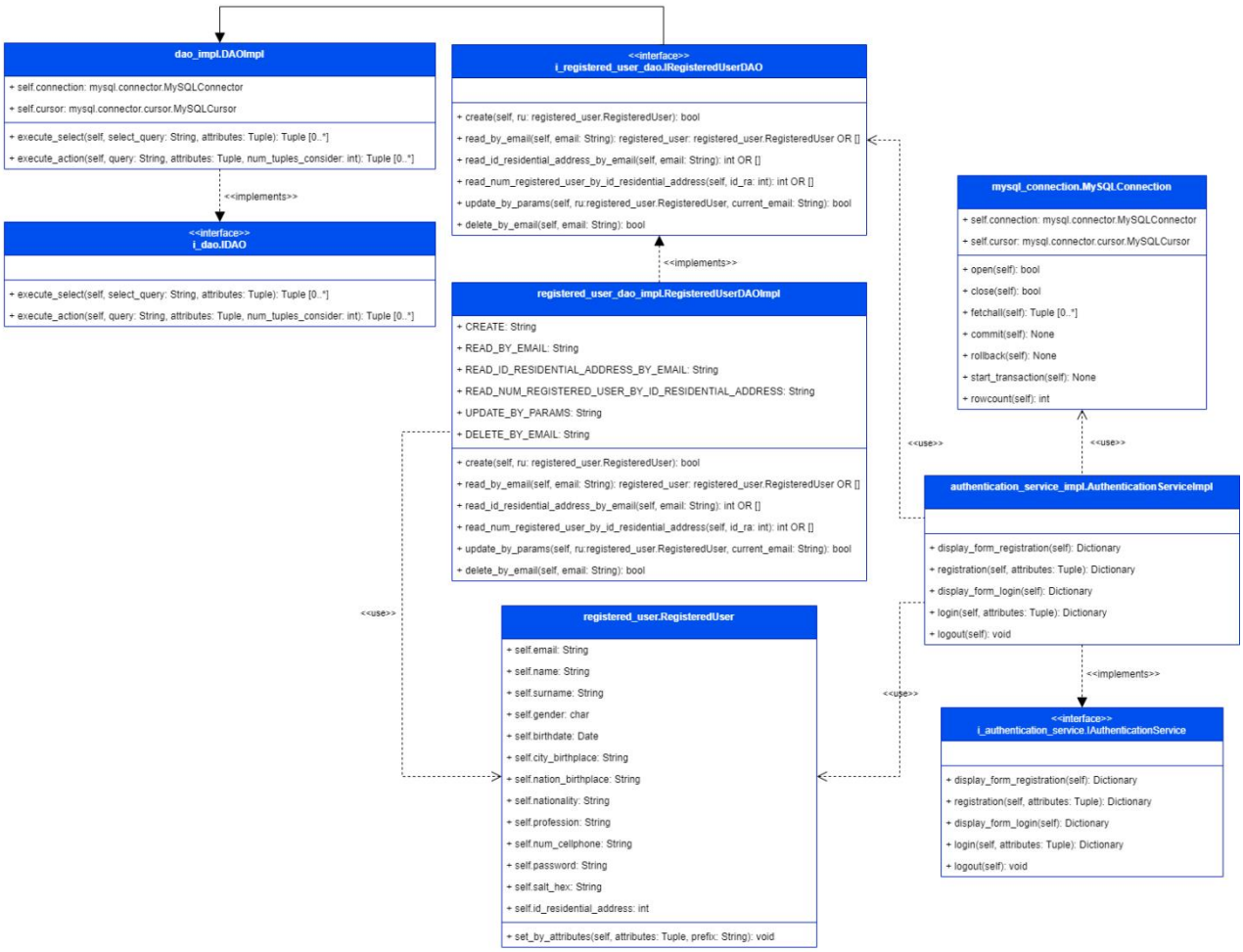
Come è stato risolto il problema:

Per risolvere il problema è stato utilizzato il design pattern architetturale DAO all'interno del package Model del design pattern architetturale MVTS. Inoltre, la connessione al database nei vari DAO verrà gestita tramite una classe chiamata MySQLConnection presente nel modulo mysql_connection, questo perché, siccome alcune azioni nel database prendono in considerazione più tabelle sotto

un’unica transazione MySQL, l’uso della classe MySQLConnection ci permette di mantenere le operazioni sulle tabelle separate.

Grafico della struttura delle classi

Di seguito è riportato un esempio di utilizzo di un DAO nella web application SearchQS e delle relazioni che ha con altre classi dell’applicazione



5 Glossario

Nella presente sezione sono raccolti le sigle o i termini del documento che necessitano di una definizione.

Sigla / Termine	Definizione
Package	Raccolta di moduli Python e / o di file HTML e / o di file JavaScript e / o di file CSS e / o di immagini e / o di altri tipi di file.
DAO	Data Access Object, implementazione dell'omonimo pattern architetturale che si occupa di fornire un accesso ai dati persistenti.
Service	Classe Python che implementa la logica di business. Viene utilizzata dal modulo view o da un altro sottosistema.
View	Modulo Python che si occupa di gestire le richieste effettuate dal client.
Model	Parte del design architetturale MVTS che fornisce al sistema i metodi per accedere ai dati utili al sistema.
MVTS	Model View Template Service. design architetturale che permette di separare la logica di presentazione dalla logica di business alla base del sistema.