Università degli Studi di Napoli Parthenope

****

Progetto di Reti di Calcolatori

Docenti:

Aniello Castiglione

Alessio Ferone

Candidato:

Nicola Esposito - 0124002092

Anno accademico 2022/2023

Sommario

[1. Contesto progetto 2](#_Toc132972644)

[2. Informazioni client e server 4](#_Toc132972645)

[2.1. Condizioni richieste per il progetto 4](#_Toc132972646)

[**2.2.** Requisiti 4](#_Toc132972647)

[2.2.1. Requisitifunzionali 4](#_Toc132972648)

[2.2.2. Requisiti non funzionali 4](#_Toc132972649)

[3. Schermate 5](#_Toc132972650)

[3.1. Schermata di login 5](#_Toc132972651)

[3.2. Schermata di registrazione 6](#_Toc132972652)

[3.3. Schermata di home (cliente) 7](#_Toc132972653)

[3.4. Schermata di programmazione 8](#_Toc132972654)

[3.5. Schermata del film 9](#_Toc132972655)

[3.6. Schermata delle prenotazioni 10](#_Toc132972656)

[3.7. Schermata degli abbonamenti 11](#_Toc132972657)

[3.8. Schermata di pagamento 12](#_Toc132972658)

[3.9. Schermata di home (admin) 13](#_Toc132972659)

[3.10. Schermata di gestione dei film 14](#_Toc132972660)

[4. Database 15](#_Toc132972661)

[4.1. Sintesi dei requisiti 15](#_Toc132972662)

[4.2. Diagramma EE/R 16](#_Toc132972663)

[4.3. Diagramma relazionale 16](#_Toc132972664)

[4.4. Creazione tabelle 17](#_Toc132972665)

[4.5. Connessione al Database 19](#_Toc132972666)

[5. Design patterns 20](#_Toc132972667)

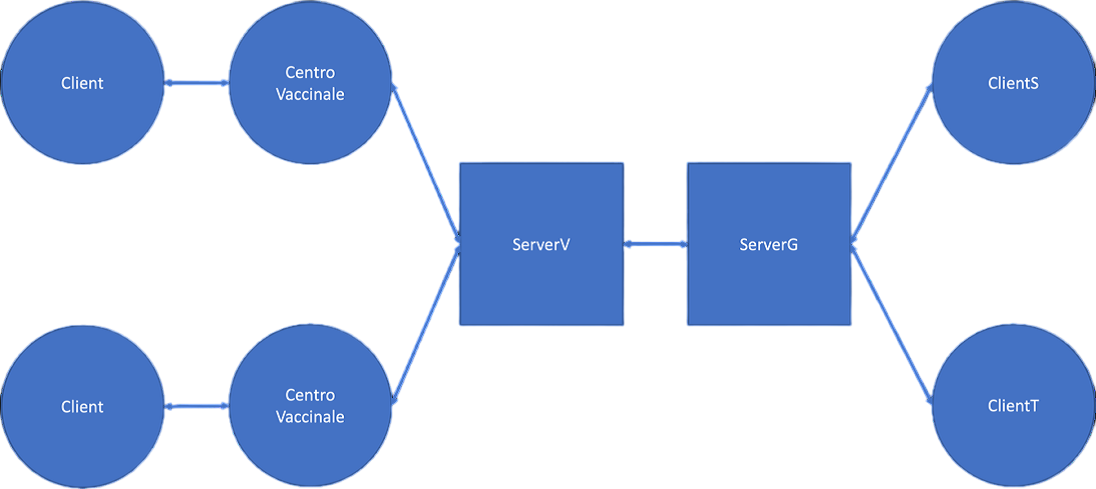
[5.1. Model-View-Controller (MVC) 21](#_Toc132972668)

[5.2. Facade 21](#_Toc132972669)

[5.3. Builder 23](#_Toc132972670)

# Contesto progetto

Si vuole progettare ed implementare un servizio di gestione dei green pass, secondo le seguenti specifiche. Un utente, una volta effettuata la vaccinazione, tramite un client si collega ad un centro vaccinale e comunica il codice della propria tessera sanitaria. Il centro vaccinale comunica al Server V il codice ricevuto dal client ed il periodo di validità del green pass. Un Client S, per verificare se un green pass è valido, invia il codice di una tessera sanitaria al Server G il quale richiede al Server V il controllo della validità. Un Client T, inoltre, può invalidare o ripristinare la validità di un green pass comunicando al Server G il contagio o la guarigione di una persona attraverso il codice della tessera sanitaria.



*Fig. 1: Architettura progetto*

Il progetto è realizzato in linguaggio C utilizzando i socket per la comunicazione tra processi, ed è pronto ad essere eseguito su qualsiasi piattaforma UNIX.

# Informazioni client e server

Di seguito verranno riportate le informazioni specifiche sui client e sui server del progetto, cercando di evidenziare scopi e collegamenti tra quest’ultimi.

## Condizioni richieste per il progetto

Il progetto è stato sviluppato tenendo conto delle seguenti linee guida:

* Rispetto del requisito relativo al numero dei design pattern (GoF) da implementare (in questo caso due design pattern);
* Rispetto di alcuni dei principi SOLID: SRP, OCP, LSP, ISP, DIP;
* Utilizzo del linguaggio Java;
* Utilizzo di interfacce grafiche per una più chiara e coincisa visualizzazione.

## Requisiti

### Requisitifunzionali

Di seguito sono riportati i requisiti funzionali, suddivisi in base alla tipologia di utente loggato all’applicazione:

* Amministratore:
* Gestione dell’elenco dei film in programmazione: inserimento, modifica e rimozione a livello di database, dei film da proiettare.
* Gestione della programmazione: inserimento e modifica delle informazioni relative ai film presenti.
* Cliente:
* Visualizzazione dell’elenco dei film presenti nell’applicazione con eventuale possibilità di acquisto. In tale elenco sono presenti le seguenti informazioni: titolo, durata, età minima, costo, informazioni varie.
* Visualizzazione della scheda relativa alle prenotazioni dei film. In tale scheda sono presenti le seguenti informazioni: titolo, data di proiezione con il corrispondente orario, sala e la validità della prenotazione.
* Visualizzazione della scheda relativa all’abbonamento. In tale scheda l’utente può sottoscrivere un abbonamento mensile o annuale. Inoltre verrà mostrata in alto la data di scadenza dell’attuale abbonamento.

### Requisiti non funzionali

Di seguito sono riportati i requisiti non funzionali:

* Usabilità: l’applicazione deve essere intuitiva e le interfacce utente devono essere di facile comprensione per tutte le categorie di utente;
* Privacy e sicurezza: il sistema deve garantire che i dati personali degli utenti, che utilizzano l’applicazione, non siano resi pubblici;
* Conformità: l’applicazione deve essere conforme al sistema operativo;
* Portabilità: l’applicazione deve potersi adattare a più piattaforme;
* Database: l’applicazione deve essere basata sull’utilizzo del database.

# Schermate

La grafica del software è realizzata mediante l’utilizzo del package grafico Java Swing. Di seguito sono riportate le varie schermate con la relativa spiegazione.

## Schermata di login

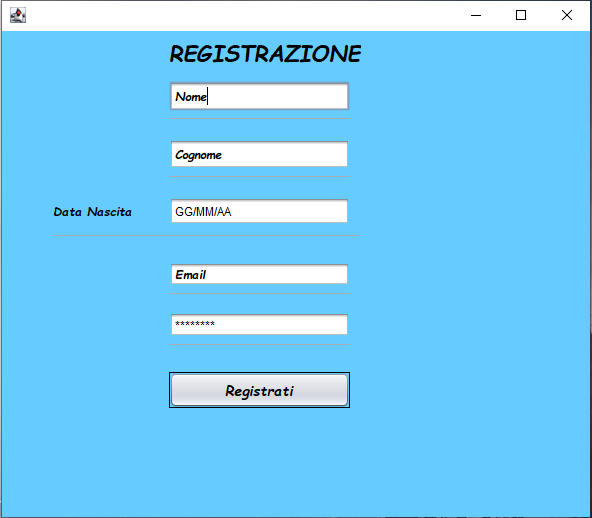
In tale schermata, è possibile effettuare l’accesso all’applicazione usando l’username e la password. La scelta del tipo di utente è effettuata attraverso la ComboBox (cliente, amministratore). Se un utente non ha un account registrato, può cliccare sul bottone “Registrati” e verrà reindirizzato alla pagina della registrazione. Invece, nel caso in cui un utente abbia inserito delle credenziali non corrette, verrà visualizzato un messaggio di errore. Analogamente se il database non è connesso all’applicazione.



*Fig. 1: Mock-up login utente*

## Schermata di registrazione

In tale schermata, l’utente può effettuare la propria registrazione all’applicazione inserendo i propri dati anagrafici (nome, cognome, data di nascita), la propria email e creando una password.

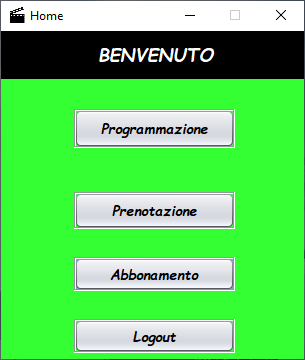


*Fig.2: Mock-up registrazione utente*

## Schermata di home (cliente)

Una volta effettuato il login come cliente, l’utente accede alla schermata di benvenuto. All’utente sono presentate quattro scelte:

* Visualizzare l’elenco completo dei film in programmazione con le relative informazioni;
* Visualizzare l’elenco completo delle prenotazioni effettuate, potendo inoltre distinguere le prenotazioni ancora valide da quelle invece già scadute;
* Visualizzare, se lo si possiede, le informazioni relative al proprio abbonamento (scadenza, tipologia e costo) o acquistare un nuovo abbonamento;
* Effettuare il logout per tornare alla schermata di login.



*Fig.3: Mock-up home cliente*

## Schermata di programmazione

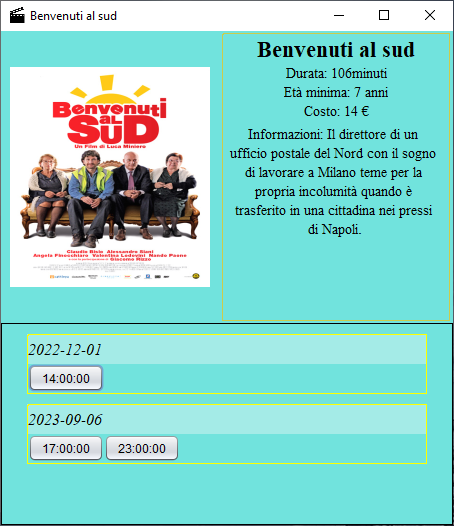
In tale schermata, l’utente può visualizzare le informazioni relative ad ogni singolo film, come la locandina, il titolo, la durata, l’età minima, il costo e una breve descrizione.



*Fig.4: Mock-up programmazioni*

## Schermata del film

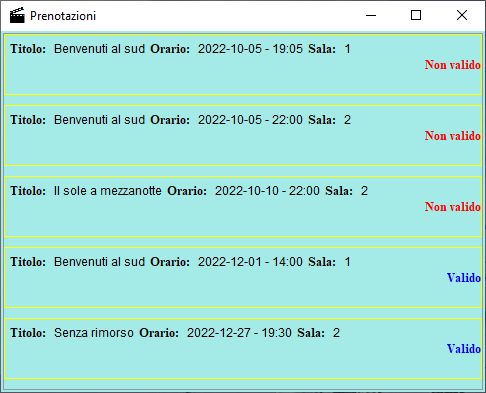
In tale schermata, l’utente può inoltre visualizzare le date e gli orari disponibili per il film selezionato e decidere di acquistare un biglietto.



*Fig.5: Mock-up film*

## Schermata delle prenotazioni

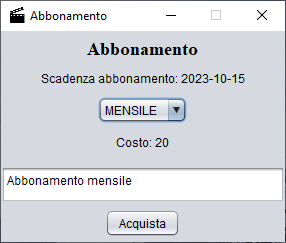
In tale schermata, l’utente può visualizzare l’elenco completo delle prenotazioni dei film effettuate, distinguendo le prenotazioni ancora valide da quelle invece già scadute.



*Fig.6: Mock-up prenotazioni*

## Schermata degli abbonamenti

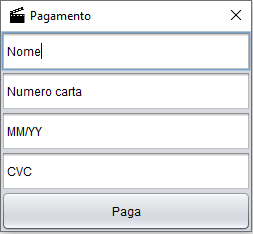
In tale schermata, l’utente può visualizzare, se lo si possiede, la scadenza, la tipologia e il costo relativi al proprio abbonamento. Inoltre può decidere di acquistare un nuovo abbonamento.



*Fig.7: Mock-up abbonamenti*

## Schermata di pagamento

In tale schermata, l’utente può procedere al pagamento del nuovo abbonamento scelto in precedenza. Le informazioni da inserire sono il nome, il numero della carta di credito, mese e anno della sua scadenza e il suo CVC.



*Fig.8: Mock-up pagamento*

## Schermata di home (admin)

Una volta effettuato il login come admin, l’utente accede alla schermata di benvenuto. All’utente sono presentate due scelte:

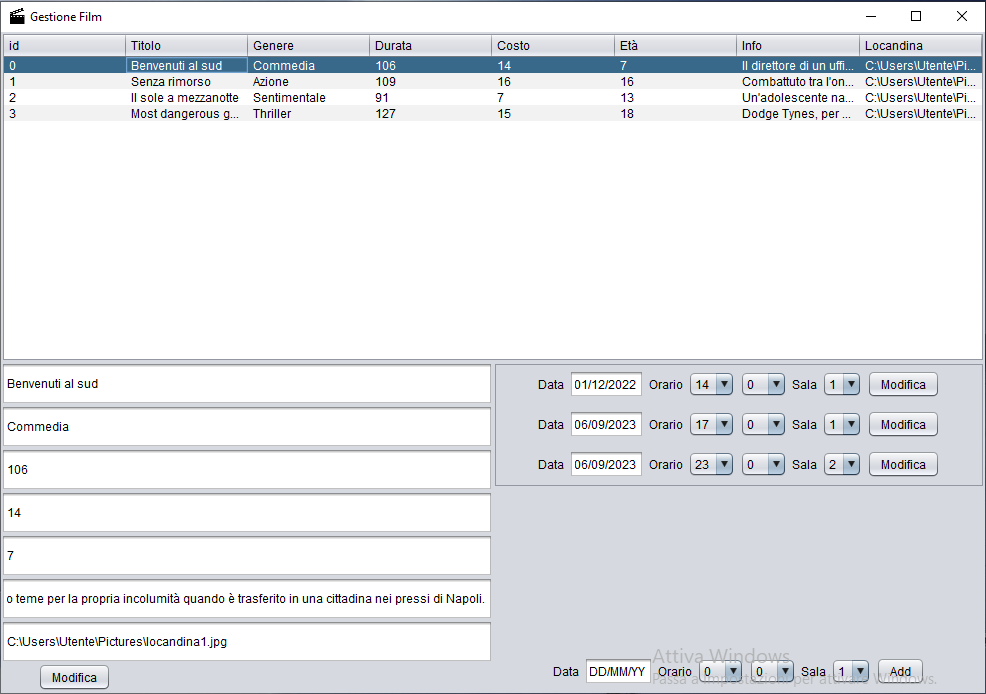
* Visualizzare la scheda relativa alla gestione dei film in programmazione con le relative informazioni;
* Effettuare il logout per tornare alla schermata di login.



*Fig.9: Mock-up home admin*

## Schermata di gestione dei film

In tale schermata, l’admin può apportare modifiche a tutte le informazioni relative ad un film, così come aggiungere delle programmazioni.



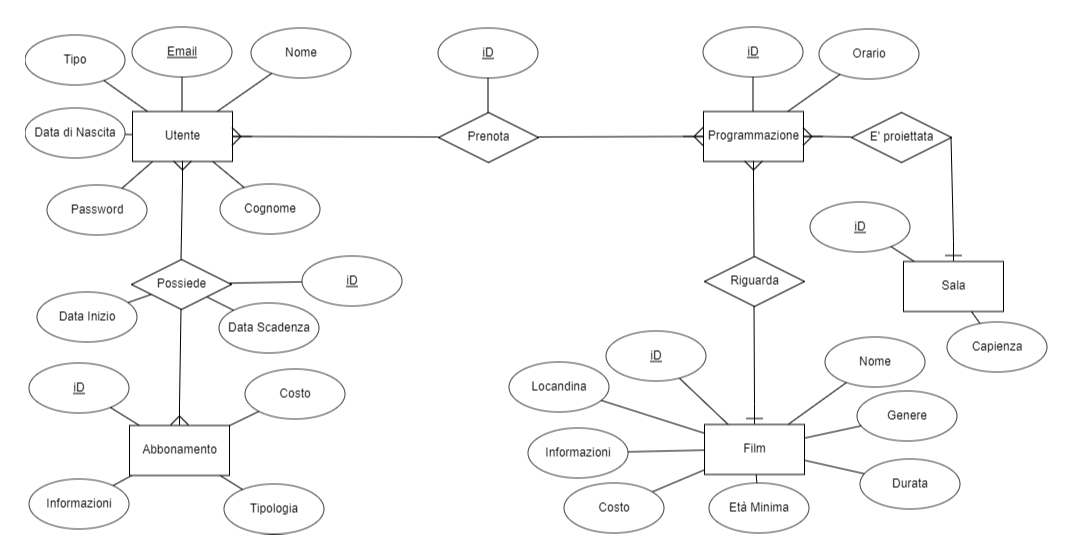
*Fig.10: Mock-up gestione film*

# Database

## Sintesi dei requisiti

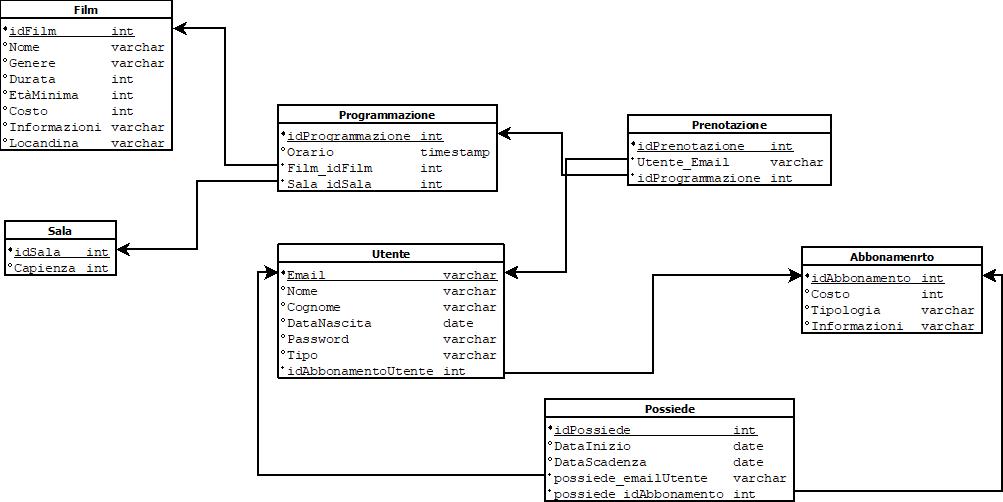
Si vuole realizzare un database per la gestione di un cinema. La gestione del cinema è affidata a un utente, identificato da una email, il tipo (in questo caso un amministratore), il nome, il cognome, la password e la data di nascita. Un utente, in questo caso un cliente, può sottoscrivere un abbonamento, identificato da un ID, il costo, la tipologia (mensile o annuale) e le informazioni. Attraverso l’abbonamento, un utente può prenotare una programmazione, identificata da un ID e dall’orario, che riguarda i film. Questi ultimi sono identificati da un ID, il nome, il genere, la durata, l’età minima, il costo, la locandina e le informazioni. Inoltre le programmazioni sono proiettate nelle sale, che sono identificate da un ID e dalla capienza.

## Diagramma EE/R



*Fig.11: Diagramma EE/R*

## Diagramma relazionale



*Fig.12: Diagramma relazionale*

## Creazione tabelle

create table Abbonamento (

idAbbonamento number (3) primary key,

Costo number (3),

Tipologia varchar (45),

Informazioni varchar (150));

create table Utente (

Email varchar (100) primary key,

Nome varchar (20) not null,

Cognome varchar (20) not null,

DataNascita date,

Password varchar (100),

Tipo varchar (20),

constraint fk\_Email\_Utente foreign key (idAbbonamentoUtente) references Abbonamento (idAbbonamento));

create table Possiede (

idPossiede number (3),

DataInizio date,

DataScadenza date,

possiede\_emailUtente varchar (256),

possiede\_idAbbonamento number (3),

constraint pk\_Pos primary key (idPossiede, possiede\_emailUtente, possiede\_idAbbonamento),

constraint fk\_Pos\_EmailUtente foreign key (possiede\_emailUtente) references Utente (Email)

constraint fk\_Pos\_idAbbonamento foreign key (possiede\_idAbbonamento) references Abbonamento (idAbbonamento));

create table Film (

idFilm number (3) primary key,

Nome varchar (30),

Genere varchar (20),

Durata number (3),

EtàMinima number (2),

Costo number (3),

Informazioni varchar (500),

Locandina varchar (256));

create table Sala (

idSala number (2) primary key,

Capienza number (3));

create table Programmazione (

idProgrammazione number (3),

Orario timestamp (5),

Film\_idFilm number (3),

Sala\_idSala number (3),

constraint pk\_Programmazione primary key (idProgrammazione, Film\_idFilm, Sala\_idSala),

constraint fk\_Prog\_idFilm foreign key (Film\_idFilm) references Film (idFilm),

constraint fk\_Prog\_idSala foreign key (Sala\_idSala) references Sala (idSala));

create table Prenotazione (

idPrenotazione number (3),

Utente\_Email varchar (100),

idProgrammazione number (3),

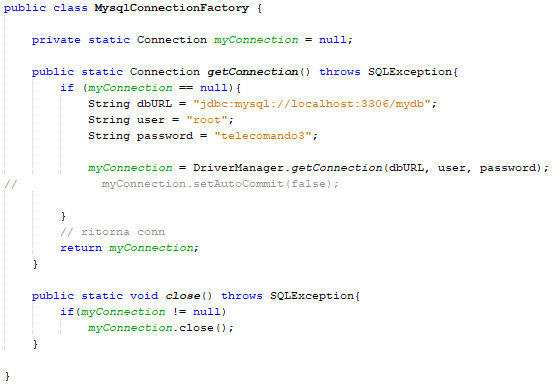
constraint pk\_Prenotazione primary key (idPrenotazione, Utente\_Email, idProgrammazione),

constraint fk\_Prenota\_Email foreign key (Email) references Utente (Email),

constraint fk\_Prenota\_idProgrammazione foreign key (idProgrammazione) references Programmazione (idProgrammazione));

## Connessione al Database

Di seguito è riportato il codice per la connessione al Database.



*Fig.13: Mock-up connessione al database*

# Design patterns

Di seguito sono riportati i design patterns utilizzati nel progetto:

* Model-View-Controller Pattern;
* Facade Pattern;
* Builder Pattern.

## Model-View-Controller (MVC)

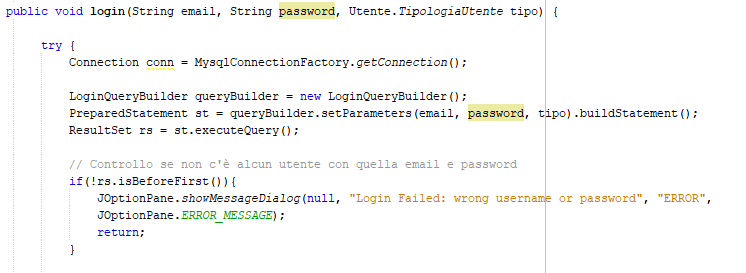
Il Model-View-Controller (MVC) è un modello di architettura del software. Prevede un’architettura composta da tre parti diverse:

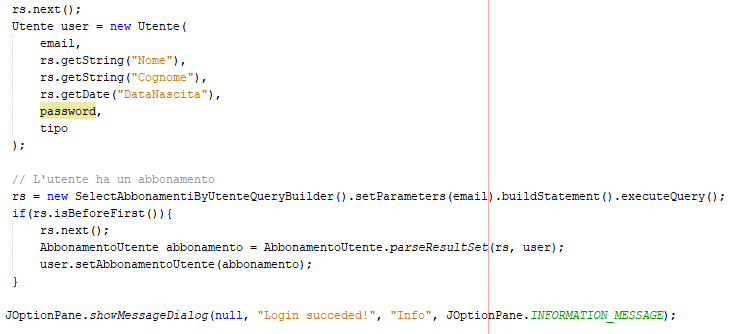
* Model: il back-end che contiene tutta la logica dei dati;
* View: il front-end o interfaccia utente grafica (GUI);
* Controller: il cervello dell’applicazione che controlla il modo in cui i dati vengono visualizzati.

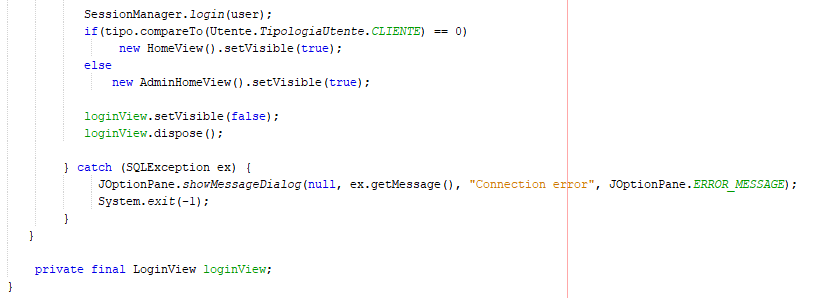
Questi tre componenti sono interconnessi: il Model viene mostrato tramite la View all’utente, il quale produce gli input con cui il Controller aggiorna il Model. Mantenerli logicamente separati ha grandi vantaggi nella gestione del codice. Infatti questo pattern favorisce lo sviluppo, il test e la manutenzione di ciascuna parte indipendentemente dall’altra.

## Facade

Il Facade Pattern è un pattern strutturale, ovvero consente di riutilizzare degli oggetti esistenti fornendo agli utilizzatori un’interfaccia più adatta alle loro esigenze. Nello specifico, tale pattern offre una soluzione per raggruppare con facilità le varie interfacce presenti all’interno dei sistemi più complessi, fornendo un’interfaccia unificata. Di seguito viene riportato il codice del Facade, utilizzato per il Login:



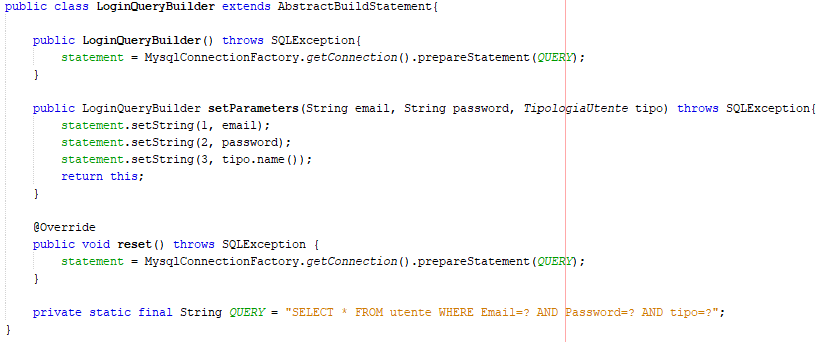




*Fig.14: Facade pattern*

## Builder

Il Builder Pattern è un pattern creazionale, separa la costruzione di un oggetto complesso dalla sua rappresentazione, cosicché il processo di costruzione stesso possa creare diverse rappresentazioni. In questo modo, l’algoritmo per la creazione di un oggetto complesso è indipendente dalle varie parti che costituiscono l’oggetto e da come vengono assemblate. Ciò ha l’effetto immediato di rendere più semplice la classe, permettendo a una classe builder separata di focalizzarsi sulla corretta costruzione di un’istanza e lasciando che la classe originale si concentri sul funzionamento degli oggetti. Migliora sia la sicurezza durante il processo di costruzione che la leggibilità del codice del programma. Nel nostro caso, ad esempio, viene utilizzato per creare oggetti di tipo LoginQuery. Di seguito viene riportato il codice del Builder:



*Fig.15: Builder pattern*