Progetto: "Telepass"

Simone Perrotta
Mat:0124002350

Flavio Ruggiero
Mat:0124002342

Anno accademico 2022-2023

Index

Progettazione
- Glossario, 3
- Analisi dei requisiti, 3
- Diagramma E/R / Relazionale, 4
- Diagramma dei casi d'uso, 5
- Diagramma delle attività, 6

Implementazione
- Diagramma delle classi, 10
- Pattern utilizzati, 10
- Codice Essenziale, 11

Glossario

Termine	Definizione	Sinonimi	Omonimi
Utenti	utenti presenti all'interno della tabella clienti	clienti	—

Analisi dei requisiti

La nostra web application ha il compito di simulare quello che è il funzionamento del telepass.
Telepass è un sistema di riscossione automatica del pedaggio autostradale. Un autoveicolo è identificato dalla targa, nome e cognome del proprietario, metodo di pagamento (e.g., carta di credito, bancomat) e possiede un dispositivo (transponder) identificato da un codice. Un autoveicolo viene riconosciuto all'entrata e all'uscita di un casello stradale e automaticamente viene addebitata la somma corrispondente al proprietario dell'autoveicolo. Il sistema deve prevedere l'accesso in modalità amministratore e in modalità utente (autoveicolo). Di seguito sono elencate le operazioni che deve poter fare l'admin:

- inserire un nuovo dispositivo Telepass;
- revocare un dispositivo Telepass;
- visualizzare periodicamente le statistiche di ingresso e di uscita dei singoli caselli;

Di seguito invece sono elencate le operazioni che devono poter fare gli utenti:

- entrare o uscire da un casello. Il sistema automaticamente calcola e gli addebita l'importo dopo un'entrata e un'uscita e lo visualizza sul transponder;
- richiedere l'associazione di una nuova targa al dispositivo Telepass;
- richiedere la conversione del suo contratto in Telepass+ (possibilità di assistenza in autostrada);

Dati questi requisiti abbiamo ritenuto opportuno sviluppare una web application anzichè un'interfaccia grafica, poichè quest'ultima applicata ad un contesto come il telepass non ci sembrava propriamente adatta.
Abbiamo inoltre aggiunto altri requisiti che il sistema dovrebbe avere, e saranno elencati di seguito:

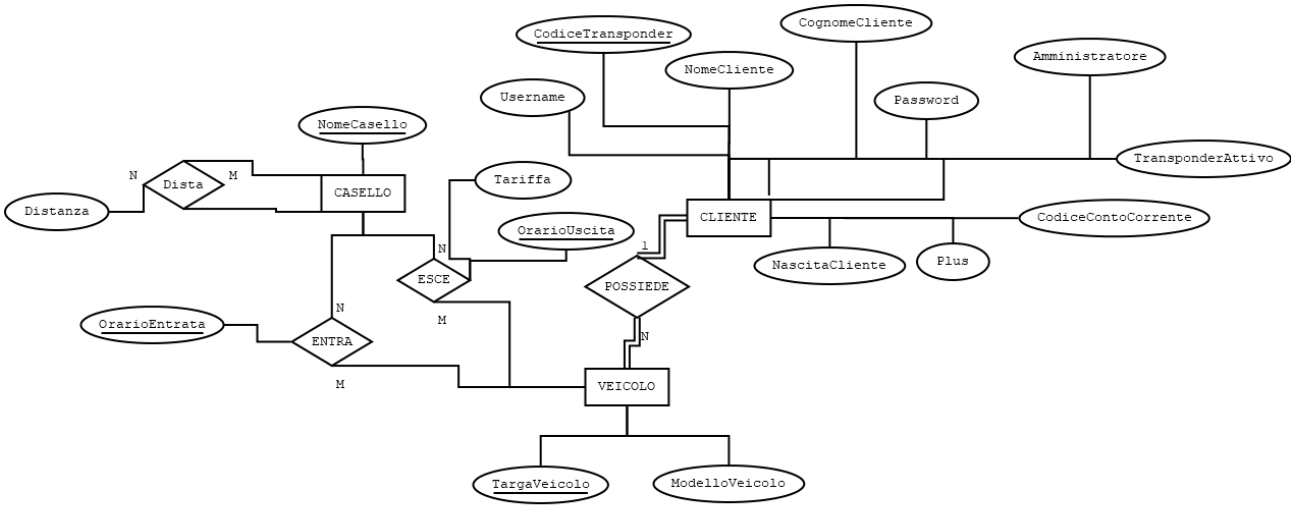
- I veicoli si differiscono per classe (A,B,3,4,5 di seguito saranno elencate le specifiche di queste 5 classi con le relative tariffe che applicheremo) affinchè, in base a questa, gli utenti paghino una tariffa diversa per ogni tratta;
- Le tariffe vanno arrotondate ai 10 centesimi più vicini (Es: 6.04 euro = 6 euro);
- Ogni utente non può avere associato più di due veicoli;
- L'utente può autonomamente gestire il suo abbonamento (compreso il Telepass+);
- L'utente deve avere almeno 18 anni per essere registrato;
- L'admin deve poter gestire non solo l'abbonamento ("normale") ma anche il Telepass+ dei vari utenti;

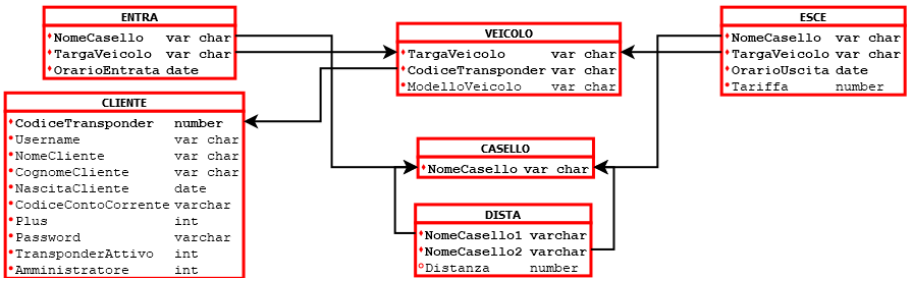
Le classi dei veicoli sono così definite: di seguito:

- classe A: auto e moto fino a 1.3 metri di altezza;
- classe B: più alti di 1.3 metri (Es: camper e camion piccoli);
- classe 3: 3 assi, quindi quattro ruote che formano due assi più uno (tipo carrello);
- classe 4: 4 assi camion con rimorchio non eccezionale);
- classe 5: 5 assi camion con rimorchio a 3 assi (quindi eccezionale);

Abbiamo ritenuto opportuno aggiungere gli ulteriori requisiti appena descritti per applicare il problema, datoci dalla traccia, quanto più vicino alla realtà. Per far sì che ci sia un'interfaccia admin e un'interfaccia utente abbiamo scelto di creare un unico admin (con credenziali semplici ai fini del progetto quali username: admin e password:admin) non replicabile (cioè non possono essere creati altri admin) e tanti utenti. All'inizio saranno quindi presenti come utenti: l'admin e un paio di clienti per poter controllare le interfacce (utente e admin) diverse che ci saranno. Riguardo la questione viaggi effettuati dagli utenti, questi saranno soltanto "simulati". Quindi dato un casello di entrata e uno di uscita si prende la distanza fra i due e si considera una velocità media di 100km/h, si calcola quindi quanto ci si dovrebbe mettere per percorrere la tratta, e l'orario risultante verrà inserito nel record in esce del viaggio in questione.

Diagramma E/R / Relazionale





Questo modello raffigura la nostra implementazione lato database. Ci sono alcune scelte progettuali di questo diagramma che vanno chiarite, come:

- Campo amministratore nella tabella cliente: abbiamo scelto di distinguere l'admin dagli utenti generali grazie a questo campo amministratore che si comporta come un booleano, quindi se settato ad 1 indicherà che quello non è un utente qualsiasi ma è proprio l'admin.

- Campo transponder attivo nella tabella cliente: poichè ci è sembrato opportuno che un utente possa essere registrato ma non avere l'abbonamento attivo, abbiamo creato questo campo. Così facendo se l'utente non ha più intenzione di utilizzare il sistema Telepass, può disdire l'abbonamento senza cancellare il proprio account, in maniera tale che se vorrà riattivare il suo abbonamento, potrà riloggarsi.

- Tabella entra ed esce: queste due tabelle sarebbero potute essere anche soltanto una. Abbiamo però deciso di dividerle per un motivo, ovvero: considerando che l'inserimento del veicolo che esce da un casello venga fatto dopo minuti (se non ore) dall'inserimento del veicolo in entra, questo avrebbe portato a valori null per ore nell'unica tabella che avremmo fatto.

Diagramma dei casi d'uso

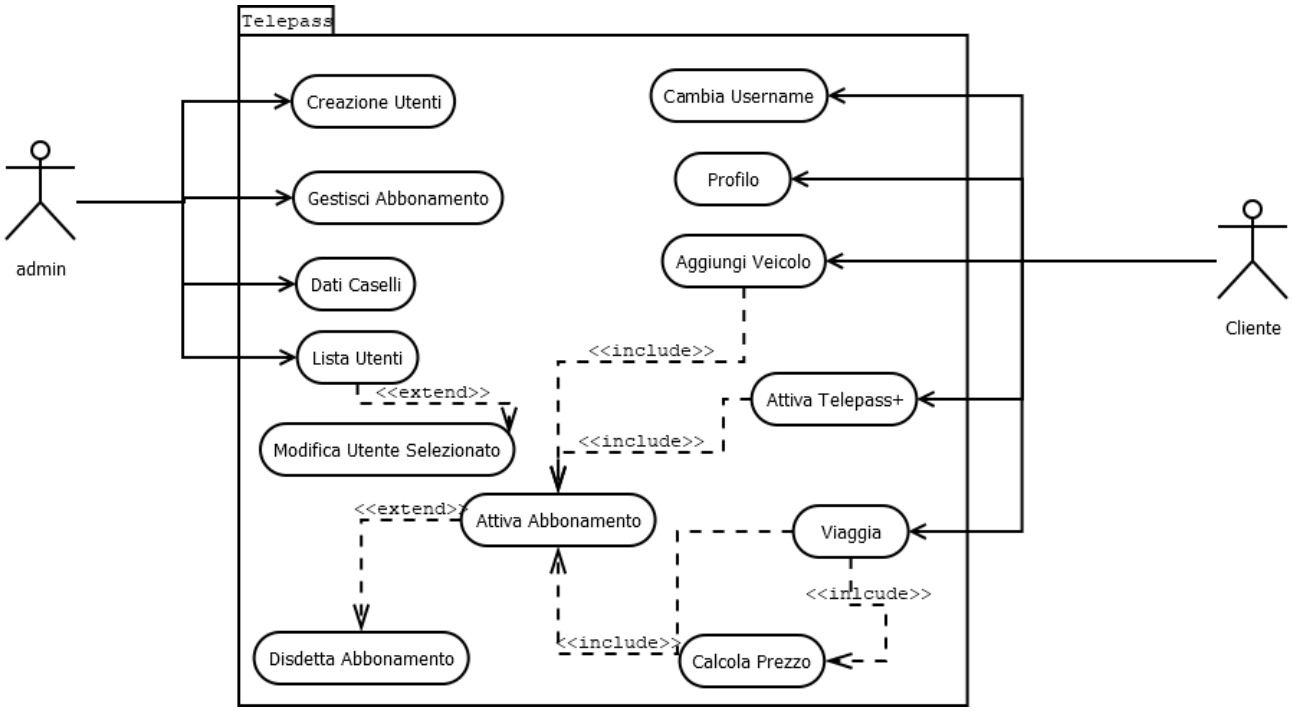
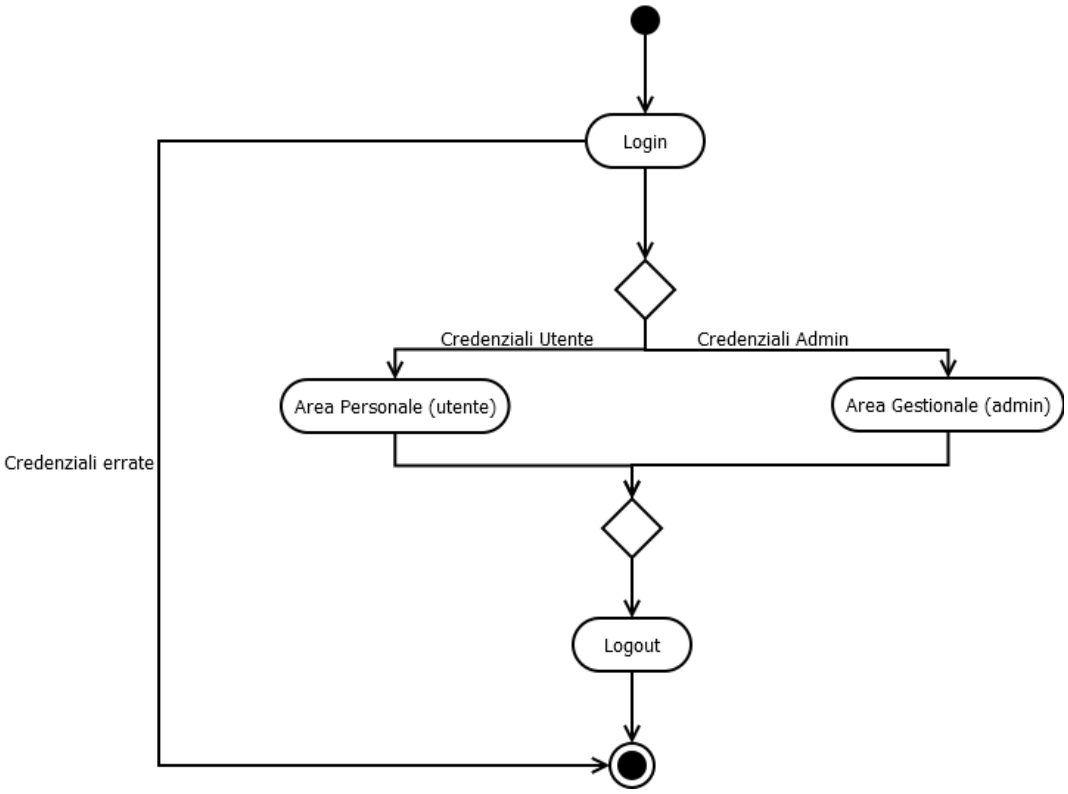


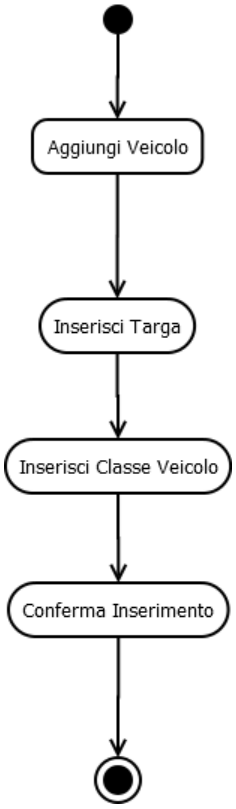
Diagramma delle attività

Attività Login



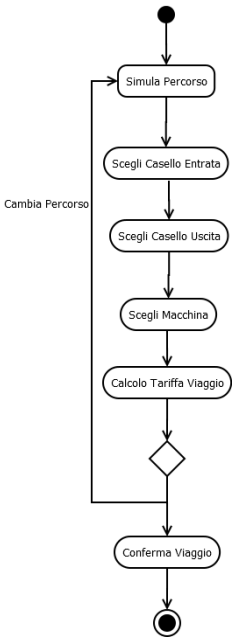
Questo diagramma delle attività mostra come viene effettuato il login.

Attività Aggiungi Veicolo



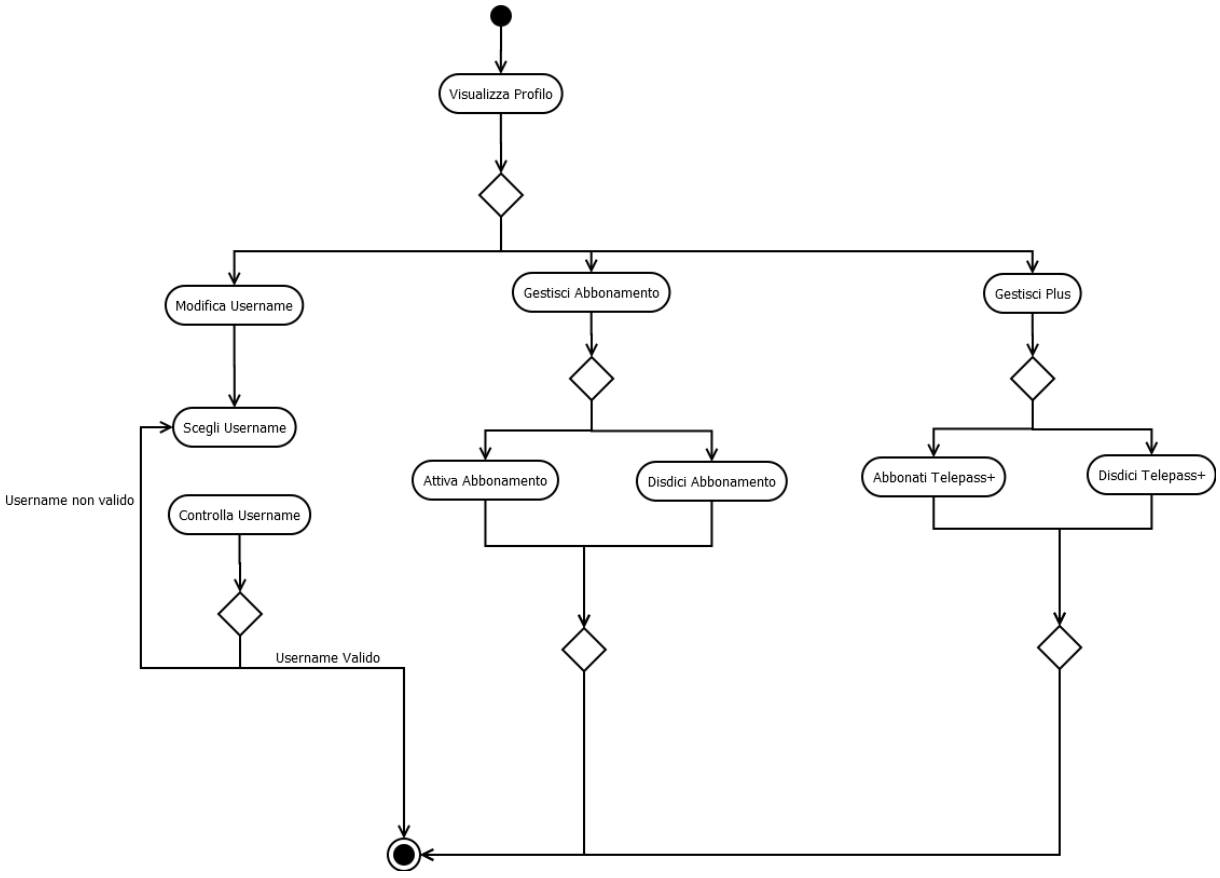
Questo diagramma delle attività mostra come viene aggiunto un veicolo dagli utenti che hanno meno di 2 veicoli associati.

Attività Simula Percorso



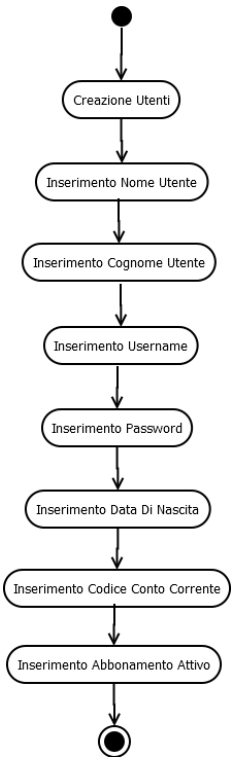
Questo diagramma delle attività mostra come avviene la simulazione di un viaggio da parte dell'utente.

Attività Visita Profilo



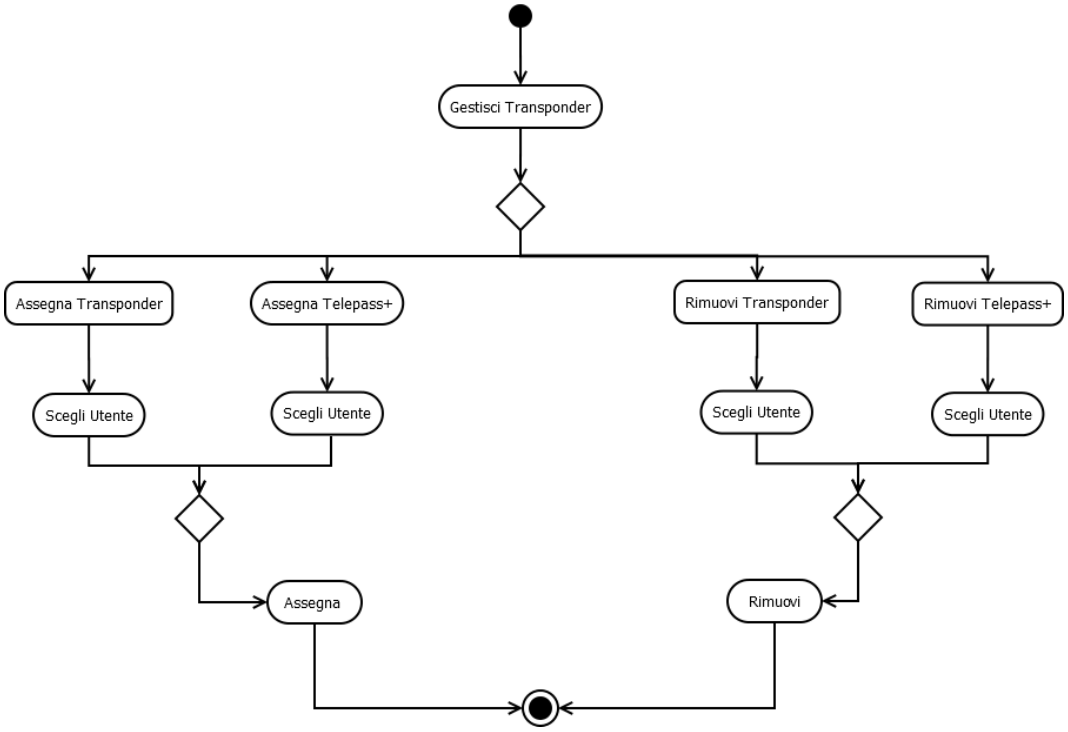
Questo diagramma delle attività mostra quello che può fare un utente dal suo profilo (e soprattutto come verrà fatto).

Attività Creazione Utenti



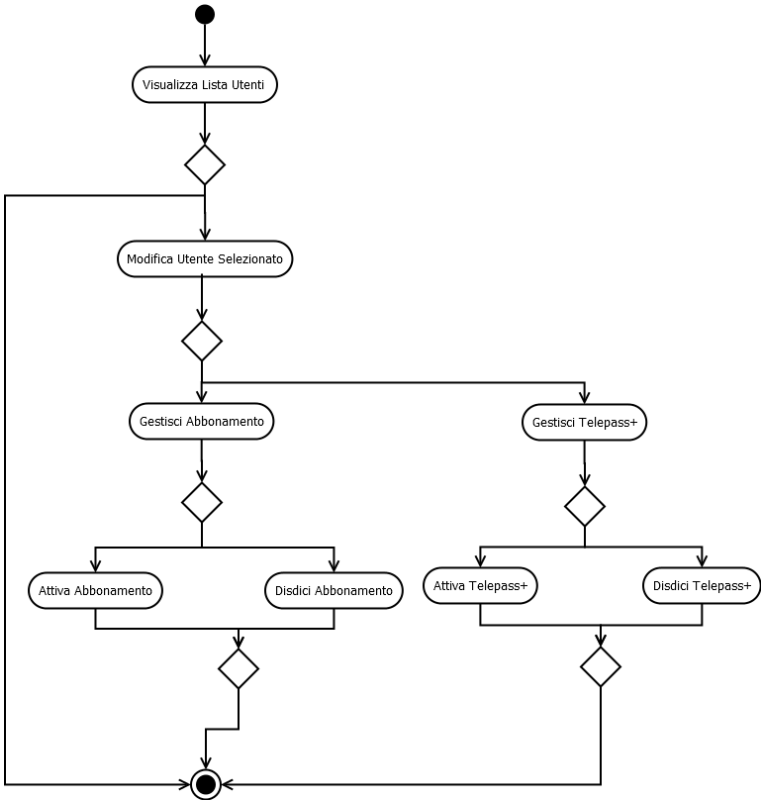
Questo diagramma delle attività mostra come l'admin può effettuare la creazione degli utenti.

Attività Gestisci Transponder



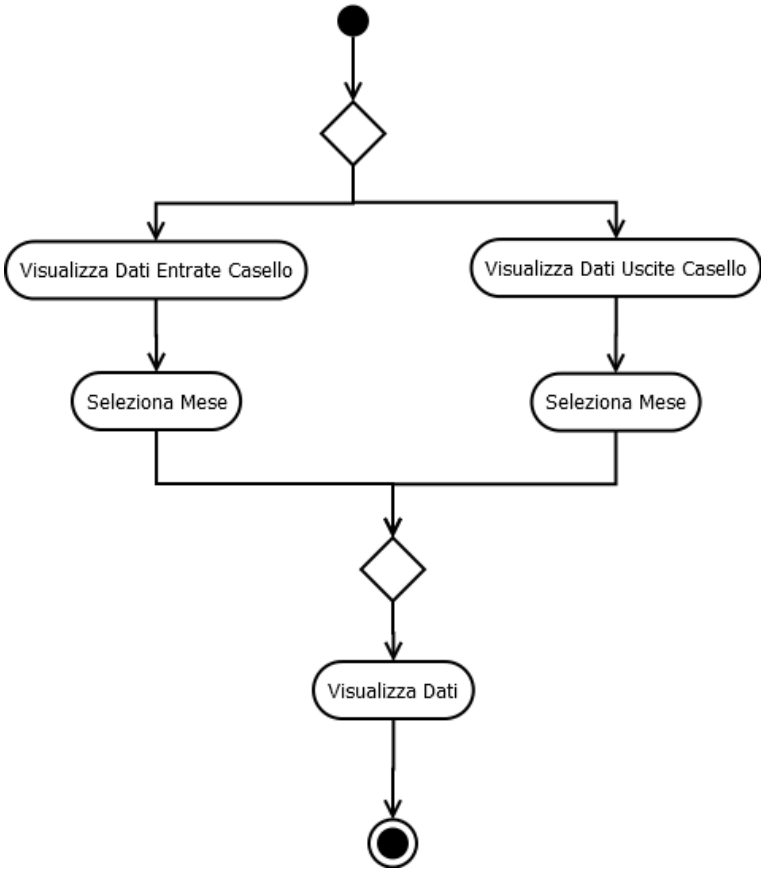
Questo diagramma delle attività mostra come l'admin può gestire (assegnare e rimuovere) gli abbonamenti e i Telepass+ degli utenti.

Attività Lista Utenti



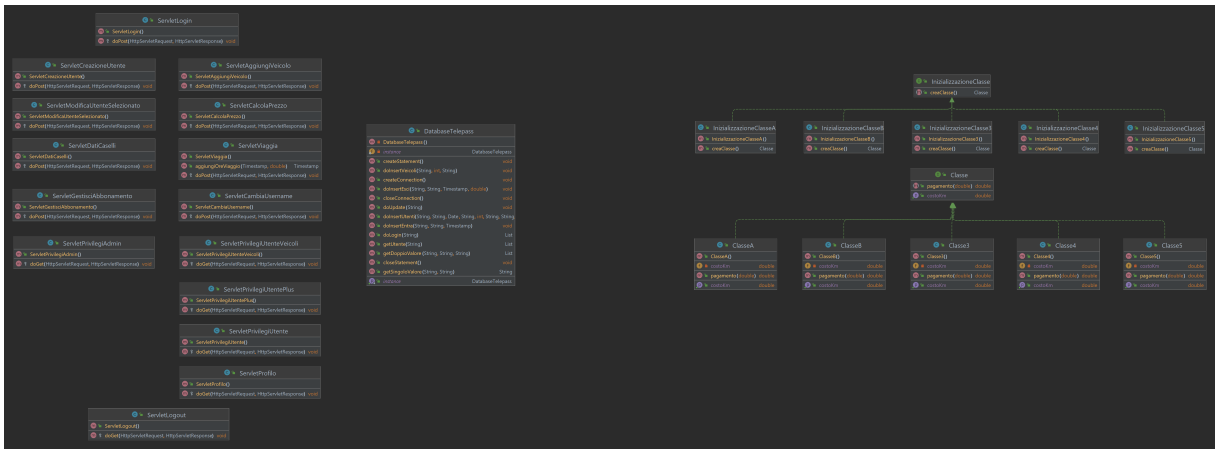
Questo diagramma delle attività mostra quello che l'admin può fare accedendo alla lista utenti e come lo farà.

Attività Dati Caselli



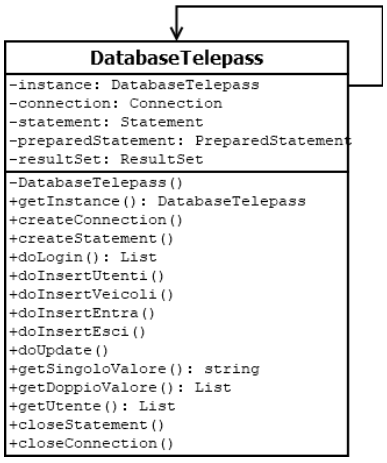
Questo diagramma delle attività mostra come l'admin può visionare le statistiche di entrata e uscita dai caselli, ovvero quanti veicoli sono entrati o usciti quel mese dai caselli.

Diagramma delle classi

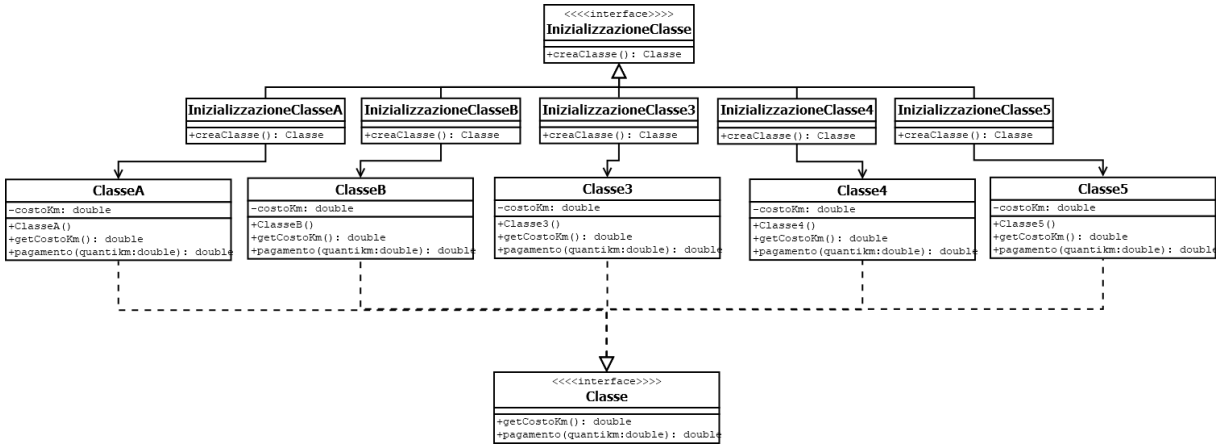


Questo diagramma rappresenta il diagramma delle classi del nostro progetto. Come possiamo vedere sono presenti le servlet con i rispettivi metodi. Poi vediamo le varie classi che implementano l'interfaccia classe e le varie iniziliazzazioni classi che implementano l'interfaccia inizializzazione classe. Queste rappresentano l'implementazione del pattern factory all'interno del nostro progetto. Ovvero viene istanziato un oggetto di tipo InizializzazioneClasse (tipo interfaccia) a cui verrà assegnato un oggetto, per esempio, di tipo InizializzazioneClasseA che si occuperà di creare e ritornare una nuova ClasseA. Poi possiamo vedere la classe DatabaseTelepass che rappresenta il Model della nostra web application, si occupa dunque di interfacciarsi con il db facendo: inserimenti, update, query, ecc (di seguito verrà mostrato il codice di questa classe). Di seguito troviamo le parti fondamentali di codice sviluppato (compreso di commenti):

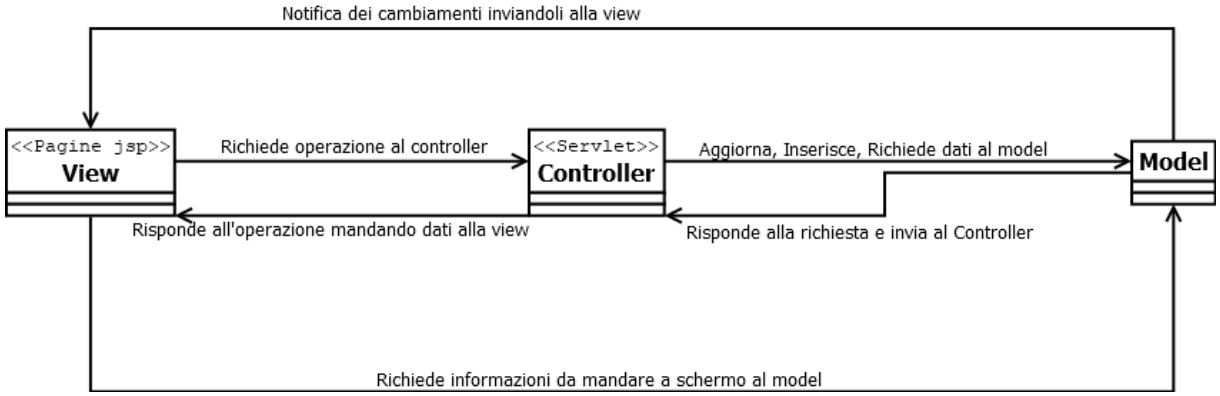
Pattern utilizzati



Il pattern Singleton lo utilizziamo per far sì che l'istanza del databaseTelepass sia unica in tutto il nostro codice.



Il pattern Factory lo utilizziamo per allocare a run time classi di veicoli in base alla scelta dell'utente. In questo caso abbiamo che il Factory è l'inizializzazioneClasse (interfaccia poichè implementa un solo metodo che andrà ridefinito nei vari concreteFactory), il concreteFactory sono le varie inizializzazioneClasseA, B, ecc., mentre i prodotti concreti sono le effettive ClasseA, B, ecc., e il prodotto è la Classe (interfaccia).



Il pattern MVC lo utilizziamo per suddividere in macro aree quelle che sono le varie sezioni della web applicaion. Troviamo nella view tutte le pagine jsp (costituite da html, css, javascript) che rappresentano appunto tutto ciò che l'utente finale vede. Come controller abbiamo tutte le servlet; queste implementano tutte le funzionalità dinamiche che ci sono all'interno dell'applicazione web (dalla creazione degli utenti alla gestione degli abbonamenti, dalla simulazione del percorso alla visualizzazione del proprio profilo). Il controller comunica a stretto contatto con il model (ovvero il db) DatabaseTelepass che fa da "interfaccia" al nostro vero model che è il db. La classe databaseTelepass integra tutti i metodi che agiscono direttamente sul db. Quindi il controller chiede informazioni, chiede di attuare aggiornamenti o inserimenti ed è il model poi a interfacciarsi con la base di dati utilizzata, e a fornire una risposta al controller di turno (in base alla servlet utilizzata in quel momento).

Codice Essenziale

Codice Classe DatabaseTelepass

```
N.B ESSENDO IL DATABASE SVILUPPATO IN MYSQL PER FAR SI CHE IL PROGETTO FUNZIONI BISOGNA CHE VENGA CREATO PRIMA IL DB COMPRESO IL RIEMPIMENTO SPECIFICATO NEL FILE TELEPASS.SQL
E ANCHE CHE VENGA CREATO ALL'INTERNO DEL PANNELLO PHPMYADMIN UN UTENTE ROOT CON PASSWORD ROOT CHE ABBIA TUTTI I PRIVILEGI.

/* Questa classe fa parte della sezione Model del pattern MVC. Difatti questa ci permette di comunicare con la base di dati
esistente e di inserire, modificare, leggere e eliminare valori. Questa classe è implementata tramite il pattern Singleton
così dà avere un'unica istanza del db nel nostro programma backend senza dichiararne di più.
Ogni metodo di questa classe prevedrà la gestione delle eccezzioni, come eccezzioni nell'apertura delle connessioni, degli
statement. Tutte queste eccezzioni saranno gestite con dei catch, che stamperanno qualcosa in output per notificare
allo sviluppatore quali e dove sono gli errori qualora ci dovessero essere*/
public class DatabaseTelepass {
    private static DatabaseTelepass instance;//unica istanza del db
    public static Connection connection =null;//connessione da allocare ogni volta con il db
    private static Statement statement;//statement da allocare ogni volta per eseguire operazioni sul db
    private static PreparedStatement preparedStatement;//prepared statement da allocare ogni volta che serve per eseguire operazioni sul db
    private ResultSet resultSet;//conterrà i risultati delle query fatte dai vari metodi
```

```
//come da definizione delle classi Singleton il costruttore risulta essere privato
private DatabaseTelepass(){}

/*punto di accesso globale all'istanza allocata per poter fare qualsiasi operazione sul db
Se l'istanza non è stata creata, la crea, altrimenti ritorna semplicemente l'istanza*/
public static DatabaseTelepass getInstance(){
    if(instance == null)
        instance = new DatabaseTelepass();

    return instance;
}
/*Questo metodo si occupa di creare la connessione con il db mysql Telepass con l'utilizzo dell'utente ROOT.
Ricordiamo che quest'utente va creato in separata sede dal resto della creazione del db*/
public void createConnection(){
    //proviamo ad aprire la connessione
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/telepass", "ROOT","ROOT");
    }//nel caso in cui non si sia aperta lanciamo l'eccezzione con un messaggio in output
    catch (Exception e){
        System.out.println("errore nell'apertura della connessione");
    }
}
//Questo metodo si occupa della creazione dello statement
public void createStatement(){
    //proviamo ad aprire lo statement
    try {
        statement = connection.createStatement();
    }//nel caso in cui non si sia aperto lanciamo l'eccezzione con un messaggio in output
    catch (Exception e){
        System.out.println("errore nell'apertura dello statement");
    }
}
/*Questo metodo si occupa di fare il login (unico sia per admin che per utenti) grazie alla query passata come
parametro stringa.
Restiutuisce una lista di parametri che verranno usati come parametri di sessione per l'utente loggato*/
public List doLogin(String sql){
    List parametri = new ArrayList();//lista che conterrà i parametri da restituire
    createConnection();//crea la connessione
    createStatement();//crea lo statement
    try{
        resultSet = statement.executeQuery(sql);//esegue la query passata come parametro
        //se ci sono risultati dall'esecuzione della query
        if(resultSet.next()) {
            //aggiunge tutto alla lista che verrà data in output
            parametri.add(resultSet.getInt("Amministratore"));//setta il ruolo come int
            parametri.add(resultSet.getInt("CodiceTransponder"));//setta il codice transponder
            parametri.add(resultSet.getInt("TransponderAttivo"));//setta se il transponder è attivo o meno con un int
            parametri.add(resultSet.getInt("Plus"));//setta se il plus è attivo o meno con un int
        }
        //se non ci dovessero essere utenti corrispondenti (dati inseriti nel form errati)
        else{
            //inseriamo come primo valore nella lista null così da inviare un "errore" alla servlet che chiamerà questo metodo
            parametri.add(null);
            System.out.println("quest'utente non esiste");
        }
    }//nel caso ci fosse stato un errore nell'esecuzione della query
    catch (Exception e){
        System.out.println("errore nell'esecuzione della query");
    }
    //a prescindere se ci sono stati errori o meno
    finally {
        try{
            //se ci sono stati risultati all'interno del resultSet, la chiudiamo
            if(resultSet!=null)
                resultSet.close();
        }//se ci sono errori nella chiusura del resultSet
        catch (Exception e){
            //diamo un messaggio in output di errore
            System.out.println("errore nella chiusura del risultato della query");
        }
        closeStatement();//chiusura dello statement
        closeConnection();//chiusura della connessione

        return parametri;
    }
}
/*Questo metodo si occupa dell'inserimento degli utenti nel db.
Vengono passati come parametri tutti i campi da inserire nella tabella del db "clienti" (ovvero gli utenti)*/
public void doInsertUtenti(String nomeCliente, String cognomeCliente, Date nascitaCliente, String codiceContoCorrente, int plus, String username, String password, int transponderattivo)
throws SQLException {
    createConnection();//crea la connessione
    try{
        //prepara il preparedStatement per l'inserimento di tutti i campi passati come parametri
        preparedStatement = connection.prepareStatement("INSERT INTO Cliente(NomeCliente, CognomeCliente, NascitaCliente, CodiceContoCorrente, Plus, Password, Username, TransponderAttivo,
        Amministratore) VALUES (?, ?, ?, ?, ?, ?, ?, ?, 0)");
        preparedStatement.setString(1,nomeCliente);//setta il nomeCliente
        preparedStatement.setString(2,cognomeCliente);//setta il cognomeCliente
        preparedStatement.setDate(3, nascitaCliente);//setta la nascitaCliente
        preparedStatement.setString(4,codiceContoCorrente);//setta il codiceContoCorrente
        preparedStatement.setInt(5, plus);//setta il plus
        preparedStatement.setString(6, password);//setta la password
        preparedStatement.setString(7, username);//setta lo username
        preparedStatement.setInt(8, transponderattivo);//setta se il transponder è attivo o meno
        preparedStatement.execute();//esegue il preparedstatement per l'insert
    }
    //se c'è stato un errore nella creazione del preparedStatement
    catch (Exception e){
        //lanciamo un messaggio di errore in output
        System.out.println("errore nell'inserimento dell' utenti");
    }
    //a prescindere se ci sono stati errori o meno
    finally {
        preparedStatement.close();//chiudiamo il preparedStatement
        closeConnection();//chiusura della connessione
    }
}
/*Questo metodo si occupa di inserire i veicoli all'interno del db
Vengono passati come parametri tutti i campi da inserire nella tabella del db "veicoli"*/
public void doInsertVeicoli(String targa, int cod, String classe) throws SQLException {
    createConnection();//crea la connessione
    try{
        //prepara il preparedStatement per l'inserimento di tutti i campi passati come parametri
        preparedStatement = connection.prepareStatement("INSERT INTO Veicolo() VALUES (?, ?, ?)");
        preparedStatement.setString(1,targa);//setta la targa
        preparedStatement.setInt(2,cod);//setta codice trasponder
        preparedStatement.setString(3,classe);//setta la classe del veicolo

        preparedStatement.execute();//esegue il preparedstatement per l'insert
    }
    //se c'è stato un errore nella creazione del preparedStatement
    catch (Exception e){
```

```
        //lanciamo un messaggio di errore in output
        System.out.println("errore nell'inserimento del veicolo");
    }
    //a prescindere se ci sono stati errori o meno
    finally {
        preparedStatement.close();//chiudiamo il preparedStatement
        closeConnection();//chiusura della connessione
    }
}
/*Questo metodo si occupa di inserire le entrate dei veicoli ai caselli.
Vengono passati come parametri tutti i campi da inserire nella tabella del db "entra"*/
public void doInsertEntra(String targa, String casello1, Timestamp oggi) throws SQLException {
    createConnection();//crea la connessione
    try{
        //prepara il preparedStatement per l'inserimento di tutti i campi passati come parametri
        preparedStatement = connection.prepareStatement("INSERT INTO entra() VALUES (?, ?, ?)");
        preparedStatement.setString(1,targa);//setta la targa
        preparedStatement.setString(2,casello1);//setta il casello di entrata
        preparedStatement.setTimestamp(3,oggi);//setta la data corrente fittizia come data di entrata

        preparedStatement.execute();//esegue il preparedstatement per l'insert
    }
    //se c'è stato un errore nella creazione del preparedStatement
    catch (Exception e){
        //lanciamo un messaggio di errore in output
        System.out.println("errore nell'entrata del veicolo nel casello");
    }
    //a prescindere se ci sono stati errori o meno
    finally {
        preparedStatement.close();//chiudiamo il preparedStatement
        closeConnection();//chiusura della connessione
    }
}
/*Questo metodo si occupa di inserire le uscite dei veicoli dai caselli.
Vengono passati come parametri tutti i campi da inserire nella tabella del db "esci"*/
public void doInsertEsci(String targa, String casello2, Timestamp oggi, double tariffa) throws SQLException {
    createConnection();//crea la connessione
    try{
        //prepara il preparedStatement per l'inserimento di tutti i campi passati come parametri
        preparedStatement = connection.prepareStatement("INSERT INTO esce() VALUES (?, ?, ?, ?)");
        preparedStatement.setString(1,targa);//setta la targa
        preparedStatement.setString(2,casello2);//setta il casello di uscita
        preparedStatement.setTimestamp(3,oggi);//setta la data di uscita calcolata nella servlet
        preparedStatement.setDouble(4,tariffa);//setta la tariffa

        preparedStatement.execute();//esegue il preparedstatement per l'insert
    }
    //se c'è stato un errore nella creazione del preparedStatement
    catch (Exception e){
        //lanciamo un messaggio di errore in output
        System.out.println("errore nell'entrata del veicolo nel casello");
    }
    //a prescindere se ci sono stati errori o meno
    finally {
        preparedStatement.close();//chiudiamo il preparedStatement
        closeConnection();//chiusura della connessione
    }
}
/*Questo metodo si occupa di eseguire degli update sul db*/
public void doUpdate(String sql){
    createConnection();//crea la connessione
    createStatement();//crea lo statement
    try{
        //prova ad eseguire la query passata come parametro
        statement.executeUpdate(sql);
    }
    //se c'è stato un errore durante l'update
    catch (Exception e){
        //lanciamo un messaggio di errore in output
        System.out.println("errore nell'esecuzione della query");
    }
    //a prescindere se l'update viene fatto oppure no
    finally {
        closeStatement();//chiusura dello statement
        closeConnection();//chiusura della connessione
    }
}
/*Questo metodo si occupa di eseguire una query che debba ritornare valori da un solo campo.
Es: SELECT NomeCliente FROM Cliente. Questo metodo però prenderà solo il primo valore che restituirà la query,
restituendolo sottoforma di stringa; perchè questo metodo viene utilizzato solo per controllare se ci
siano valori già esistenti nel db (es: username già presente, codice conto corrente già presente, ecc).
Quindi a noi non interessa quanti utenti risulteranno dalla query, ci interessa sapere se c'è n'è anche solo uno.*/
public String getSingoloValore(String sql, String campo){
    String risultato = null;//prepariamo la lista che conterrà i risultati della query
    createConnection();//crea la connessione
    createStatement();//crea lo statement
    try{
        //prova ad eseguire la query
        resultSet = statement.executeQuery(sql);
        //se ci sono risultati
        if(resultSet.next()) {
            //se ci sono risultati dalla query, lo prende
            risultato =  resultSet.getString(campo);
        }
        //se non ci sono risultati dalla query
        else{
            //mette a null risultato
            risultato = null;
            System.out.println("non ci sono risultati");
        }
    }
    //se c'è stato un errore durante la query
    catch (Exception e){
        //lanciamo un messaggio di errore in output
        System.out.println("errore nell'esecuzione della query");
    }
    //a prescindere se l'update viene fatto oppure no
    finally {
        try{
            //se ci sono stati risultati
            if(resultSet!=null)
                resultSet.close();//chiude resultSet
        }
        //se ci sono stati errori nel chiudere resultSet
        catch (Exception e){
            //lanciamo un messaggio di errore in output
            System.out.println("errore nella chiusura del risultato della query");
        }
        closeStatement();//chiusura dello statement
        closeConnection();//chiusura della connessione

        return risultato;
    }
```

```
    }
    /*Questo metodo si occupa di eseguire una query che debba ritornare valori da due soli campi.
    Es: SELECT NomeCliente, NascitaCliente FROM Cliente. Questo metodo però prenderà solo la prima tupla che restituirà la query,
    restituendolo sottoforma di lista.
    Quindi a noi non interessa quanti utenti risulteranno dalla query, ci interessa sapere se c'è n'è anche solo uno.
    Ritorna quindi una lista contenente 2 valori risultanti dai 2 campi specificati.*/
    public List getDoppioValore(String sql, String campo1, String campo2) {
        List risultati = new ArrayList();//crea la lista che conterrà i risultati
        createConnection();//crea la connessione
        createStatement();//crea lo statement
        try{
            //prova ad eseguire la query
            resultSet = statement.executeQuery(sql);
            //se ci sono risultati
            if(resultSet.next()) {
                risultati.add(resultSet.getString(campo1));//aggiunge il valore contenuto nel primo campo a risultati
                risultati.add(resultSet.getString(campo2));//aggiunge il valore contenuto nel secondo campo a risultati
            }
            //se non ci sono risultati
            else{
                risultati.add(null);//aggiunge null a risultati
                System.out.println("non ci sono risultati");
            }
        }
        //se c'è stato un errore durante la query
        catch (Exception e){
            //lanciamo un messaggio di errore in output
            System.out.println("errore nell'esecuzione della query");
        }
        //a prescindere se l'update viene fatto oppure no
        finally {
            try{
                //se ci sono stati risultati
                if(resultSet!=null)
                    resultSet.close();//chiude resultSet
            }
            //se ci sono stati errori nel chiudere resultSet
            catch (Exception e){
                //lanciamo un messaggio di errore in output
                System.out.println("errore nella chiusura del risultato della query");
            }
            closeStatement();//chiusura dello statement
            closeConnection();//chiusura della connessione

            return risultati;
        }
    }
    /*Questo metodo si occupa di eseguire una query che ritorni tutti i parametri di un utente.
    Difatti questo metodo ritorna una Lista (risultati) che conterrà tutti i parametri di un utente passato all'interno
    della query stessa*/
    public List getUtente(String sql) {
        List risultati = new ArrayList();//crea la lista che conterrà i risultati
        createConnection();//crea la connessione
        createStatement();//crea lo statement
        try{
            //prova ad eseguire la query
            resultSet = statement.executeQuery(sql);
            //se ci sono risultati
            if(resultSet.next()) {
                //aggiunge tutto alla lista che verrà data in output
                risultati.add(resultSet.getString("NomeCliente"));//setta il NomeCliente
                risultati.add(resultSet.getString("CognomeCliente"));//setta il CognomeCliente
                risultati.add(resultSet.getInt("TransponderAttivo"));//setta se il trasponder è attivo oppure no
                risultati.add(resultSet.getInt("Plus"));//setta se il plus è attivo oppure no
                risultati.add(resultSet.getInt("CodiceTransponder"));//setta il codice Transponder
            }
            //se non ci sono risultati
            else{
                risultati.add(null);//aggiunge null a risultati
                System.out.println("non ci sono risultati");
            }
        }
        //se c'è stato un errore durante la query
        catch (Exception e){
            //lanciamo un messaggio di errore in output
            System.out.println("errore nell'esecuzione della query");
        }
        //a prescindere se l'update viene fatto oppure no
        finally {
            try{
                //se ci sono stati risultati
                if(resultSet!=null)
                    resultSet.close();//chiude resultSet
            }
            //se ci sono stati errori nel chiudere resultSet
            catch (Exception e){
                //lanciamo un messaggio di errore in output
                System.out.println("errore nella chiusura del risultato della query");
            }
            closeStatement();//chiusura dello statement
            closeConnection();//chiusura della connessione

            return risultati;
        }
    }
    //Questo metodo si occupa della chiusura dello statement nel caso in cui fosse aperto
    public void closeStatement(){
        try{
            //se lo statement è aperto
            if(statement!=null)
                statement.close();//lo chiude
        }//se ci fosse un errore nella chiusura dello statement
        catch (Exception e){
            //diamo in output un messaggio di errore
            System.out.println("errore nella chiusura dello statement");
        }
    }
    //Questo metodo si occupa della chiusura della connessione nel caso in cui fosse aperta
    public void closeConnection(){
        try{
            //se la connessione è aperta
            if(connection!=null)
                connection.close();//la chiude
        }//se ci fosse un errore nella chiusura della connessione
        catch(Exception e){
            //diamo in output un messaggio di errore
            System.out.println("errore nella chiusura della connessione");
        }
    }
}
```

Questo codice rappresenta il Model della nostra web application, ovvero la classe (denominata DatabaseTelepass)

che si occupa di interfacciarsi con il database effettuando: inserimenti, update, query, ecc.

Codice Servlet Login

```
/*Questa servlet viene invocata quando un utente vuole loggarsi.
Dopo aver controllato i parametri necessari affinchè l'utente possa loggarsi,
settiamo tutti i parametri di sessione e il login avviene con successo.*/
@WebServlet("/login")
public class ServletLogin extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response) {
        try{
            //proviamo a fare il login cercando lo username e la password nel db
            List parametri = DatabaseTelepass.getInstance().doLogin("SELECT Amministratore, CodiceTransponder, TransponderAttivo, Plus FROM CLIENTE WHERE
            Username='"+request.getParameter("username")+"' AND Password='"+request.getParameter("password")+"'");
            //se c'è un risultato quindi lo username esiste e la password è corretta
            if(parametri.get(0)!=null)
            {
                //creiamo la sessione per l'utente loggato
                HttpSession session = request.getSession(true);
                //settiamo tutti i parametri di sessione che ci serviranno di seguito
                int ruolo = (int) parametri.get(0);//prendiamo il ruolo
                session.setAttribute("username", request.getParameter("username"));//settiamo il suo username dalla request
                session.setAttribute("ruolo", parametri.get(0));//settiamo il suo rulo
                session.setAttribute("codice", parametri.get(1));//settiamo il codice transponder
                session.setAttribute("attivo", parametri.get(2));//settiamo se l'abbonamento è attivo o meno
                session.setAttribute("plus", parametri.get(3));//settiamo se il plus è attivo o meno
                //se è l'admin ad essersi loggato restituiamo un messaggio e lo indirizziamo all'area admin protetta
                if(ruolo==1){
                    request.setAttribute("successMessage", "Credenziali corrette");
                    request.getRequestDispatcher("protected_area_admin.jsp").forward(request, response);
                }
                //se è l'utente ad essersi loggato restituiamo un altro messaggio e lo indirizziamo all'area utente protetta
                else{
                    request.setAttribute("successMessage", "Credenziali corrette");
                    request.getRequestDispatcher("protected_area_utente.jsp").forward(request, response);
                }
            }
            //se non si è trovato l'utente nel db restituiamo un messaggio di utente non esistente e rimandiamo alla pagina di accesso
            else{
                request.setAttribute("messageUtenteNonEsistente", "L'utente inserito non esiste");
                request.getRequestDispatcher("Accedi.jsp").forward(request, response);
            }
        }
        //se vi sono stati errori nell'esecuzione della query
        catch (Exception e) {
            //mandiamo in output un messaggio di errore
            System.out.println("errore nella connessione");
        }
    }
}
```

Questa servlet è importante poichè riguarda l'operazione di login e smista, in base a chi si è, (se admin o utente) verso la propria area protetta o di nuovo alla pagina di accesso (se le credenziali sono errate). Setta inoltre, se il login è avvenuto con successo, tutte le variabili di sessione in modo tale da recuperarli nelle varie operazioni (servlet) che l'utente farà.

Codice Lista Utenti (Pagina jsp)

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri = "http://java.sun.com/jsp/jstl/sql" prefix = "sql"%>
<html>
<head>
  <!-- CSS only -->
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-rbsA2VBKQhggwzxH7pPCaAqO46MgnOM80zW1RWuH61DGLwZJEdK2Kadq2F9CUG65"
  crossorigin="anonymous">
  <!-- JavaScript Bundle with Popper -->
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-kenU1KFdBIe4zVFOs0G1M5b4hcpxyD9F7jL+jjXkk+Q2h455rYXK/7HAuoJl+0I4"
  crossorigin="anonymous"></script>
  <link rel="stylesheet" href="css/mystyle.css">
  <title>Telepass</title>
  <link rel="shortcut icon" href="https://logo.clearbit.com/telepass.com">
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.9.1/font/bootstrap-icons.css">

</head>
<body>
<!-- chiamata per controllare i privilegi di accesso dei vari utenti -->
<jsp:include page="PrivilegiAdmin"></jsp:include>

<nav class="navbar navbar-expand-lg bg-light"><a class="navbar-brand" href="protected_area_admin.jsp"><img src="images/Logo_Telepass_2021.png" style="height:30px; margin-left: 5px;"></a></nav>
<center>
  <div>
    <div class="col-md-8">
      <h2 class="h2div">Lista Utenti</h2>
      <table class="table" style="border-width: 3px; border-color:#0d6efd; border-style: solid ">
        <thead style="background-color: #0d6efd; color:white">
        <tr>
          <th scope="col" style="text-align: center;">Nome Utente</th>
          <th scope="col" style="text-align: center;">Cognome Utente</th>
          <th scope="col" style="text-align: center;">Nascita Cliente</th>
          <th scope="col" style="text-align: center;">Codice Transponder</th>
          <th scope="col" style="text-align: center;">Username</th>
          <th scope="col" style="text-align: center;">Codice Conto Corrente</th>
          <th scope="col" style="text-align: center;">Abbonamento</th>
          <th scope="col" style="text-align: center;">Telepass+</th>
```

```
          <th scope="col" style="text-align: center;">Modifica</th>
        </tr>
        </thead>
        <tbody>

        <sql:setDataSource var="snapshot" driver="com.mysql.cj.jdbc.Driver"
                           url="jdbc:mysql://localhost:3306/telepass"
                           user="ROOT"  password="ROOT"/>

        <sql:query dataSource="${snapshot}" var="result">
          SELECT * FROM CLIENTE WHERE Amministratore=0;
        </sql:query>
        <c:forEach var="row" items="${result.rows}">
          <c:choose>
            <c:when test="${row.TransponderAttivo==1  && row.Plus==1}">
                <tr class="table-success">
            </c:when>

            <c:when test="${row.TransponderAttivo==1  && row.Plus==0}">
                <tr class="table-primary">
            </c:when>
            <c:otherwise>
              <tr class="table-danger">
            </c:otherwise>
          </c:choose>

          <td style="text-align: center;"><c:out value="${row.NomeCliente}"/></td>
          <td style="text-align: center;"><c:out value="${row.CognomeCliente}"/></td>
          <td style="text-align: center;"><c:out value="${row.NascitaCliente}"/></td>
          <td style="text-align: center;"><c:out value="${row.CodiceTransponder}"/></td>
          <td style="text-align: center;"><c:out value="${row.Username}"/></td>
          <td style="text-align: center;"><c:out value="${row.CodiceContoCorrente}"/></td>
          <c:choose>
            <c:when test="${row.TransponderAttivo==1}">
              <td style="text-align: center;"><span class="badge rounded-pill text-bg-success">Attivo</span></td>
            </c:when>
            <c:otherwise>
              <td style="text-align: center;"><span class="badge rounded-pill text-bg-danger">Disattivo</span></td>
            </c:otherwise>
          </c:choose>

          <c:choose>
            <c:when test="${row.Plus==1}">
              <td style="text-align: center;"><span class="badge rounded-pill text-bg-success">Attivo</span></td>
            </c:when>

            <c:otherwise>
              <td style="text-align: center;"><span class="badge rounded-pill text-bg-danger">Disattivo</span></td>
            </c:otherwise>
          </c:choose>
          <form action="modificaUtenteSelezionato" method="POST">
            <td style="text-align: center;"><a href="#"><button class="btn btn-sm btn-primary" name="username" value=${row.Username}><i class="bi bi-pencil-square"></i></button></a></td>
          </form>

          </tr>
        </c:forEach>
        </tbody>
      </table>
    </div>
  </div>
</center>

<section class="pageform">
  <!-- Footer -->
  <footer class="text-center text-white" style="background-color: #002752;" id="staticfooter">
    <!-- Grid container -->
    <div class="container p-4 pb-0">
      <!-- Section: CTA -->
      <section class="">
        <img src="images/Logo_Telepass_2021.png" style="height:50px;">
      </section>
      <!-- Section: CTA -->
    </div>
    <!-- Grid container -->

    <!-- Copyright -->
    <div class="text-center p-3" style="background-color: #002752;">
      © 2022 Copyright:
      <a class="text-white" href="#">Telepass.com</a>
    </div>
    <!-- Copyright -->
  </footer>
  <!-- Footer -->
</section>
</body>
</html>
```

Questo appena visto è un esempio di pagina jsp che contiene: html, css (bootstrap) jstl, ovvero tag che permettono di comunicare ad esempio con il db, facendo quindi query e ottenendo risultati da poter mandare a schermo (anche applicando dei controlli). Questa specifica jsp mostra ad esempio tutti gli utenti con le varie informazioni, e ha inoltre la possibilità di modificare l'abbonamento di ogni utente (ricordiamo che questa pagina è accessibile solo dall'admin).

Codice Servlet Modifica Username

```
/*Questa servlet viene invocata al fine di cambiare username a un utente.
Ricordiamo che gli utenti possono cambiare username solo per il loro account (non può l'admin cambiare username).
Questa servlet si occupa di controllare se il cambio username è possibile, e se lo è, di inviare un
messaggio di successo alla jsp chiamante per l'avvenuta modifica allo username.*/
@WebServlet("/cambiausername")
public class ServletCambiaUsername extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response){
        String nuovo=request.getParameter("username");//ci prendiamo il nuovo username scelto dall'utente dalla request
        HttpSession session = request.getSession(false);//non creiamo una nuova sessione ma utilizziamo quella già esistente

        try{
```

```
            //controlliamo se lo username nuovo che l'utente ha inserito esiste già nel db
            String Username = DatabaseTelepass.getInstance().getSingoloValore("SELECT Username FROM CLIENTE WHERE Username='"+nuovo+"'","Username");
            //se lo username risulta essere già presente
            if(Username != null){
                //mandiamo un messaggio di errore alla jsp chiamante
                request.setAttribute("messageUsernameUsato", "L'username scelto è gia utilizzato");
                //rimandiamo l'utente alla pagina cambia username per farlo re-inserire
                request.getRequestDispatcher("/cambiausername.jsp").forward(request, response);
            }
            //se non risulta essere utilizzato
            else{
                //eseguiamo l'update sul db modificando l'utente con il nuovo username
                DatabaseTelepass.getInstance().doUpdate("UPDATE CLIENTE SET Username='"+nuovo+"'WHERE Username='"+session.getAttribute("username")+"'");
                //settiamo come variabile di sessione il nuovo username
                session.setAttribute("username", nuovo);
                //mandiamo un messaggio di successo alla jsp chiamante
                request.setAttribute("messageUsername", "Il tuo username è stato modificato correttamente");
                //rimandiamo l'utente alla sua area protetta
                request.getRequestDispatcher("/protected_area_utente.jsp").forward(request, response);
            }
        }
        //se ci sono stati errori durante le query fatte al db
        catch (Exception e) {
            //mandiamo in output un messaggio di errore
            System.out.println("errore nella connessione");
        }
    }
}
```

Questa appena mostrata invece è un classico esempio di servlet che da la possibilità, in questo caso, agli utenti di poter modificare il proprio username. Come possiamo vedere le istruzioni vengono fatte all'interno di un try catch, quindi vengono gestite le eccezzioni con messaggi di output utili per il debugging. Quasi tutte le servlet comunicano con il file di model (DatabaseTelepass) effettuando operazioni di vario tipo. In questo caso viene chiamato il metodo getSingoloValore che restituisce un valore da una singola colonna di output (mentre getDoppioValore restituisce 2 valori da due colonne risultanti dalla query). Viene inoltre chiamato il metodo doUpdate, che si occupa di fare l'update del cliente settando il nuovo username scelto.