UNIVERSITÀ DEGLI STUDI
DI SALERNO

INGEGNERIA DEL SOFTWARE

A.A 2019/2020

Professore:

Andrea De Lucia

Alunni:

Antonio De Matteo

Girolamo Giordano

1.Introduzione

1.1 Object Design Trade-offs

Dopo la realizzazione dei documenti RAD e SDD abbiamo descritto in linea di massima quello che sarà il nostro sistema e quindi i nostri obiettivi, tralasciando gli aspetti implementativi. Il seguente documento ha lo scopo di produrre un modello capace di integrare in modo coerente e preciso tutte le funzionalità individuate nelle fasi precedenti. In particolare definisce le interfacce delle classi, le operazioni, i tipi, gli argomenti e la signature dei sottosistemi definiti nel System Design. Inoltre sono specificati i trade-off e le linee guida.

Comprensibilità vs Tempo:

Il codice deve essere quanto più comprensibile possibile per facilitare la fase di testing ed eventuali future modifiche. Il codice sarà quindi accompagnato da commenti che ne semplifichino la comprensione. Ovviamente questa caratteristica aggiungerà un incremento di tempo allo sviluppo del nostro progetto.

Interfaccia vs Usabilità:

L'interfaccia grafica è stata realizzata in modo da essere molto semplice, chiara e concisa. Fa uso di form e pulsanti disposti in maniera da rendere semplice l'utilizzo del sistema da parte dell'utente finale.

Sicurezza vs Efficienza:

La sicurezza, come descritto nei requisiti non funzionali del RAD, rappresenta uno degli aspetti importanti del sistema. Tuttavia, dati i tempi di sviluppo molto limitati, ci limiteremo ad implementare sistemi di sicurezza basati su username e password degli utenti, consultando il DBMS usato

1.2 Linee Guida per la Documentazione delle Interfacce

Gli sviluppatori dovranno seguire alcune linee guida per la scrittura del codice:

Naming Convention

- È buona norma utilizzare nomi:

1. Descrittivi
2. Pronunciabili
3. Di uso comune

4. Lunghezza medio-corta

5. Non abbreviati

6. Evitando la notazione ungherese

7. Utilizzando solo caratteri consentiti (a-z, A-Z, 0-9)

Variabili

● I nomi delle variabili devono cominciare con una lettera minuscola, e le parole seguenti con la maiuscola. Quest'ultime devono essere dichiarate ad inizio blocco, solamente una per riga e devono essere tutte allineate per facilitare la leggibilità.

Esempio: elaboratoSessioneId

● E' inoltre possibile, in alcuni casi, utilizzare il carattere underscore "_", ad esempio quando utilizziamo delle variabili costanti oppure quando vengono utilizzate delle proprietà statiche.

Esempio: CREATE_ARGOMENTO

Metodi:

● I nomi dei metodi devono cominciare con una lettera minuscola, e le parole seguenti con la lettera maiuscola. Il nome del metodo tipicamente consiste di un verbo che identifica un'azione, seguito dal nome di un oggetto. I nomi dei metodi per l'accesso e la modifica delle variabili dovranno essere del tipo getNomeVariabile() e setNomeVariabile(). Le variabili dei metodi devono essere dichiarate appena prima del loro utilizzo e devono servire per un solo scopo, per facilitarne la leggibilità. Esistono però casi particolari come ad esempio nell'implementazione dei model, dove viene utilizzata l'interfaccia CRUD.

Esempio: getId(), setId()

● I commenti dei metodi devono essere raggruppati in base alla loro funzionalità, la descrizione dei metodi deve apparire prima di ogni dichiarazione di metodo, e deve descriverne lo scopo. Deve includere anche informazioni sugli argomenti, sul valore di ritorno e, se applicabile, sulle eccezioni.

Classi e pagine:

● I nomi delle classi devono cominciare con una lettera maiuscola, e anche le parole seguenti all'interno del nome devono cominciare con una lettera maiuscola. I nomi di queste ultime devono fornire informazioni sul loro scopo.

● I nomi delle pagine devono cominciare con una lettera minuscola. Tutte le parole al loro interno sono scritte in minuscolo e separate dal carattere '-'.

● I nomi delle servlet sono analoghi a quelli delle classi, con l'aggiunta della parola "Servlet" come ultima parola.

Esempio: ManagerAutenticazione.java, registra-biblioteca.jsp, LoginServlet.java

Ogni file sorgente java contiene una singola classe e dev'essere strutturato in un determinato modo:

● Una breve introduzione alla classe

L'introduzione indica: l'autore, la versione e la data.

```
/** * * @author nome dell'autore *
        @version numero di versione della classe *
        @since data dell'implementazione */
```

● L'istruzione include che permette di importare all'interno della classe gli altri oggetti che la classe utilizza.

● La dichiarazione di classe caratterizzata da:

1. Dichiarazione della classe pubblica
2. Dichiarazioni di costanti
3. Dichiarazioni di variabili di classe
4. Dichiarazioni di variabili d'istanza
5. Costruttore
6. Commento e dichiarazione metodi.

```
/**
 *
 */
package it.etransfer.entities;

/**

 *
 */
```

```
public class Autobus {

	private String modello;
	private int numeroposti;
	private int annoimm;
	private int chilometri;

	/**
	 * Questo è il costruttore vuoto dell'Autobus
	 */
	public Autobus() {

	}

	/**
	 * Questo è il costruttore dell'autobus dove vengono passati come parametri modello, numero dei posti,
	 * anno dell'immatricolazione e i chilometri
	 *
	 * @param modello è il modello
	 * @param numeroposti è il numero dei posti
	 * @param annoimm è l'anno di immatricolazione
	 * @param chilometri sono i chilometri
	 */
	public Autobus(String modello,int numeroposti,int annoimm,int chilometri)
	{
		this.modello=modello;
		this.numeroposti=numeroposti;
		this.annoimm=annoimm;
		this.chilometri=chilometri;
```

```
}

/**
 * @return modello restituisce un modello
 */
public String getModello() {
	return modello;
}
/**
 * @param modello è il modello
 */
public void setModello(String modello) {
	this.modello = modello;
}
/**
 * @return numeroposti restituisce il numero dei posti
 */
public int getNumeroposti() {
	return numeroposti;
}
/**
 * @param numeroposti è il numero di posti
 */
public void setNumeroposti(int numeroposti) {
	this.numeroposti = numeroposti;
}
/**
 * @return annoimm restituisce l'anno di immatricolazione
 */
public int getAnnoimm() {
```

```
		return annoimm;
	}
	/**
	 * @param annoimm è l'anno di immatricolazione
	 */
	public void setAnnoimm(int annoimm) {
		this.annoimm = annoimm;
	}
	/**
	 * @return chilometri restituisce il numero di chilometri
	 */
	public int getChilometri() {
		return chilometri;
	}
	/**
	 * @param chilometri sono i chilometri
	 */
	public void setChilometri(int chilometri) {
		this.chilometri = chilometri;
	}



	@Override
	public String toString() {
		return "Autobus [modello=" + modello + ", numeroposti=" + numeroposti + ",
annoimm=" + annoimm + ", chilometri="
				+ chilometri + "]";
	}

}
```

1.3 Definizioni, acronimi e abbreviazioni

Acronimi:

- RAD: Requirements Analysis Document
- SDD: System Design Document
- ODD: Object Design Document
- CRUD: Create Read Update Delete

Abbreviazioni:

- DB: Database

1.4 Riferimenti:

- B. Bruegge, A. H. Dutoit, Object Oriented Software Engineering - Using UML, Pattern and Java,Prentice Hall, 3rd edition, 2009
- Documento SDD del progetto ETransfer
- Documento RAD del progetto ETransfer

2. Packages

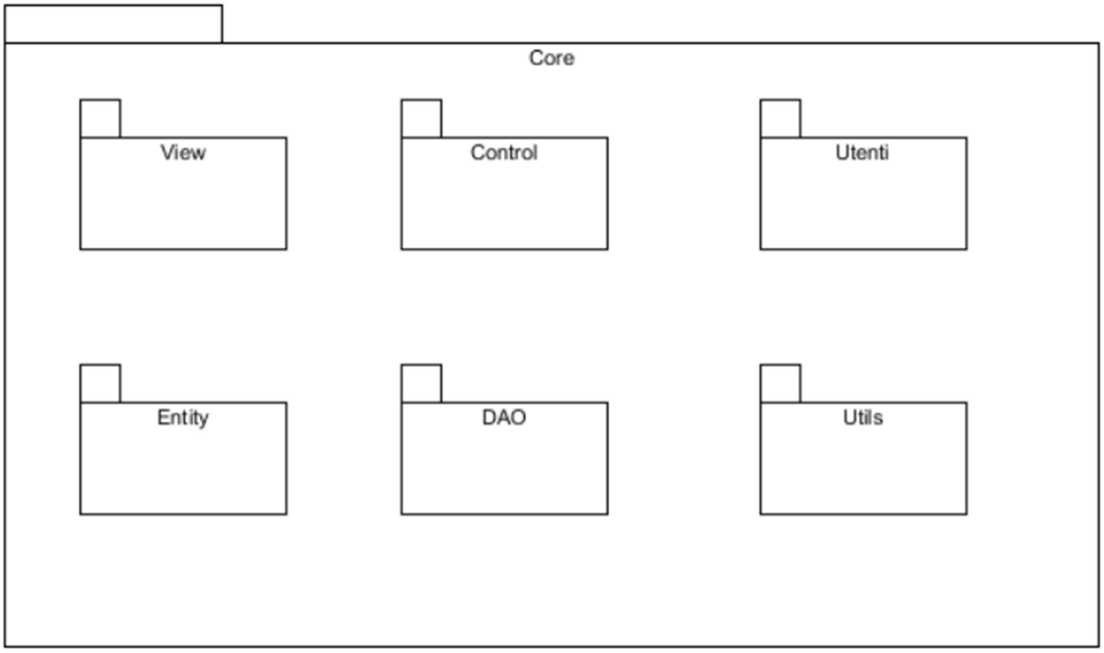
La gestione del nostro sistema è suddivisa in tre livelli (three-tier):

- Presentation layer
- Application layer
- Storage layer

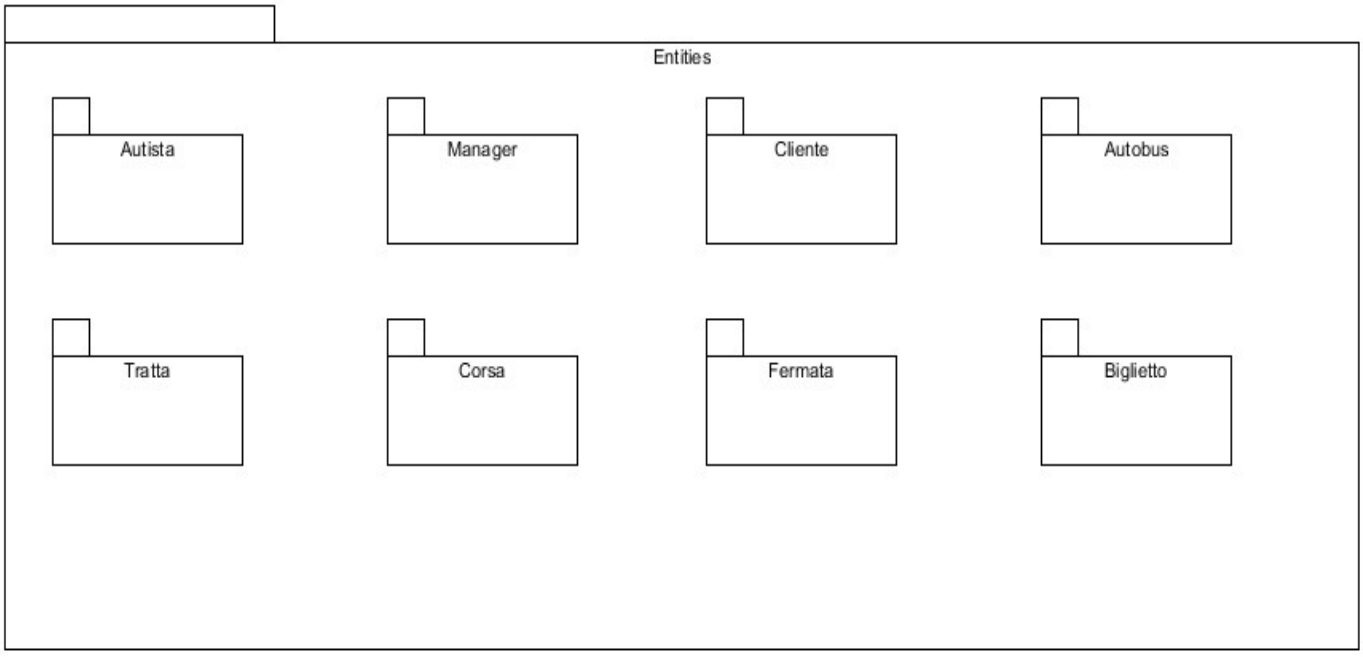
Il package ETransfer contiene sottopackage che a loro volta inglobano classi atte alla gestione delle richieste utente. Le classi contenute nel package svolgono il ruolo di gestore logico del sistema.

Presentation Layer	Include tutte le interfacce grafiche e in generale i boundary objects, come le form con cui interagisce l'utente. L'interfaccia verso l'utente è rappresentata da un Web server tramite pagine statiche come pagine HTML e varie jsp
Application Layer	Include tutti gli oggetti relativi al controllo e all'elaborazione dei dati. Questo avviene interrogando il database tramite lo storage layer per generare contenuti dinamici e accedere a dati persistenti. Si occupa di varie gestioni quali: ● Gestione Autenticazione ● Gestione Account ● Gestione Autobus ● Gestione Biglietto ● Gestione Corsa ● Gestione Tratta ● Gestione Fermata
Storage Layer	Ha il compito di effettuare memorizzazione, il recupero e l'interrogazione degli oggetti persistenti. I dati, i quali possono essere acceduti dall'application layer, sono depositati in maniera persistente su un database tramite DBMS.

2.1 Packages Core

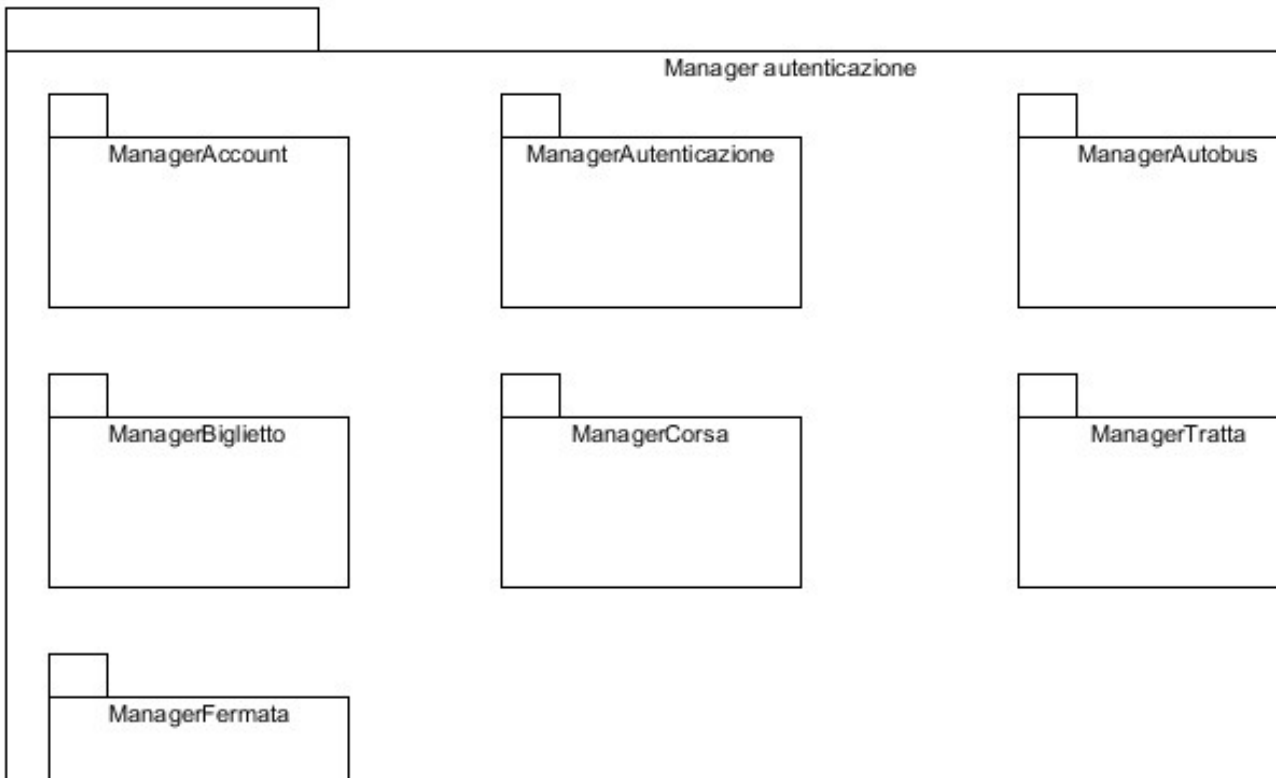


2.1.1 Packages Entities



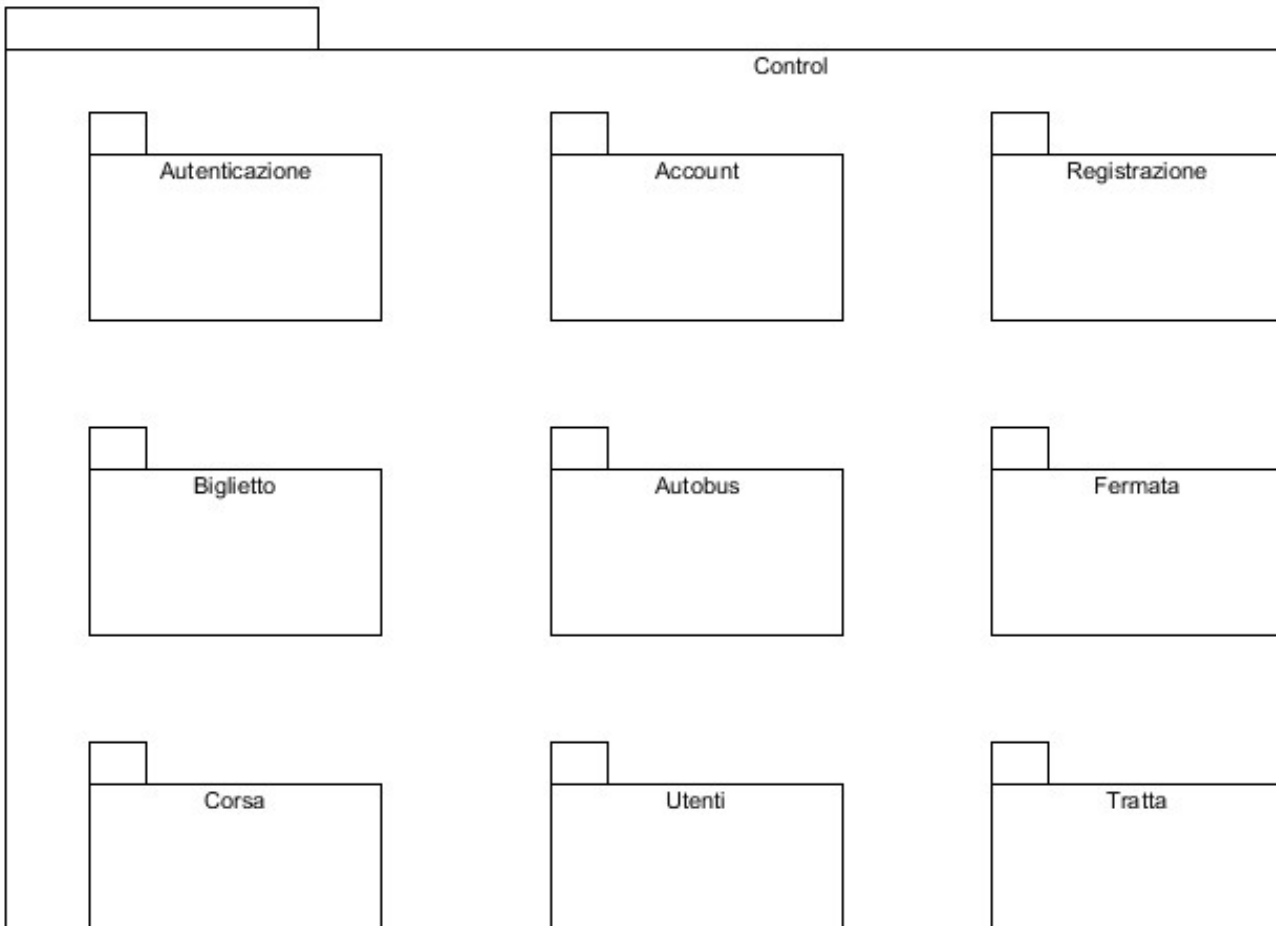
CLASSE	DESCRIZIONE
Autista	Descrive un’autista di un autobus
Manager	Descrive un manager all’interno del sistema
Cliente	Descrive un cliente all’interno del sistema
Autobus	Descrive un autobus all’interno del sistema
Tratta	Descrive punto di partenza e arrivo compiuto da un’autista
Corsa	Descrive una tratta effettuata da un’autista
Fermata	Descrive un punto di fermata effettuata all’interno della corsa
Biglietto	Descrive un biglietto acquistato da un cliente

2.1.2 Packages Manager

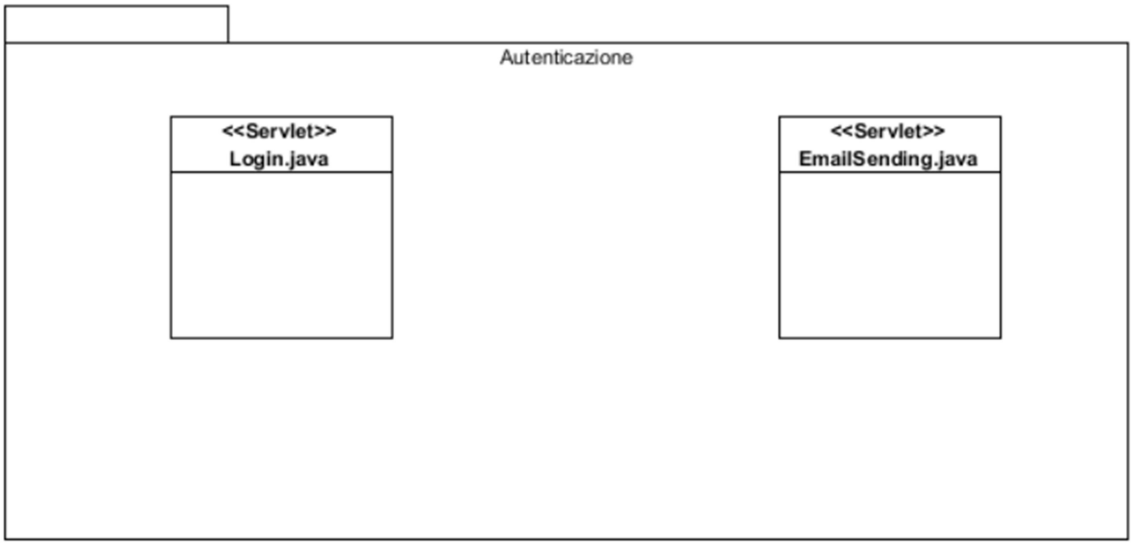


CLASSE	DESCRIZIONE
ManagerAccount	Si tratta di una classe che funge da interfaccia del sottosistema che gestisce l'account
ManagerAutenticazione	Si tratta di una classe che funge da interfaccia del sottosistema che gestisce l'autenticazione
ManagerAutobus	Si tratta di una classe che funge da interfaccia del sottosistema che gestisce l'autobus
ManagerBiglietto	Si tratta di una classe che funge da interfaccia del sottosistema che gestisce il biglietto
ManagerCorsa	Si tratta di una classe che funge da interfaccia del sottosistema che gestisce la corsa
ManagerTratta	Si tratta di una classe che funge da interfaccia del sottosistema che gestisce la tratta
ManagerFermata	Si tratta di una classe che funge da interfaccia del sottosistema che gestisce la fermata

2.1.3 Packages Control

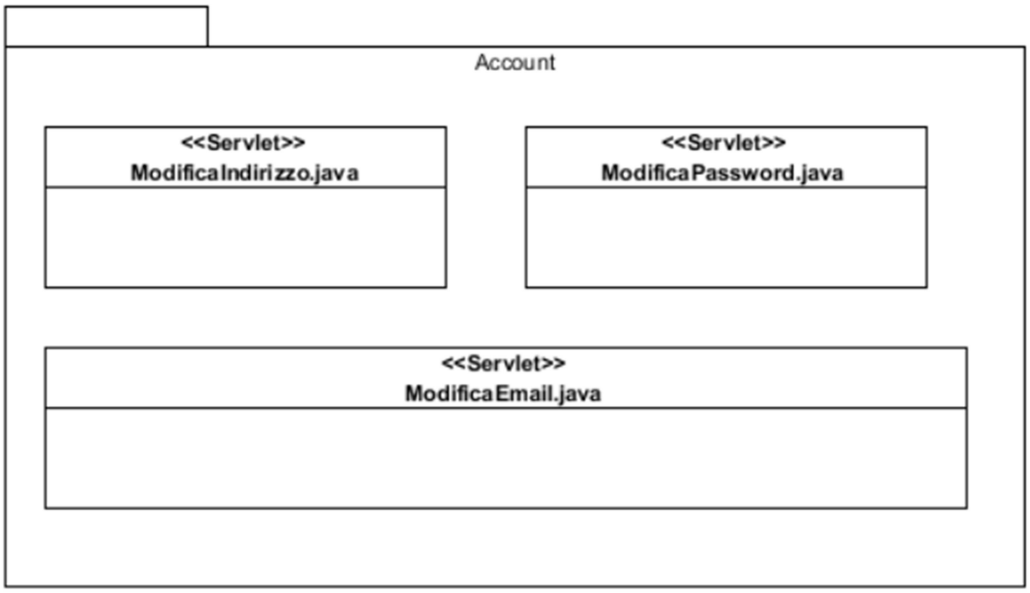


2.1.3.0 Autenticazione



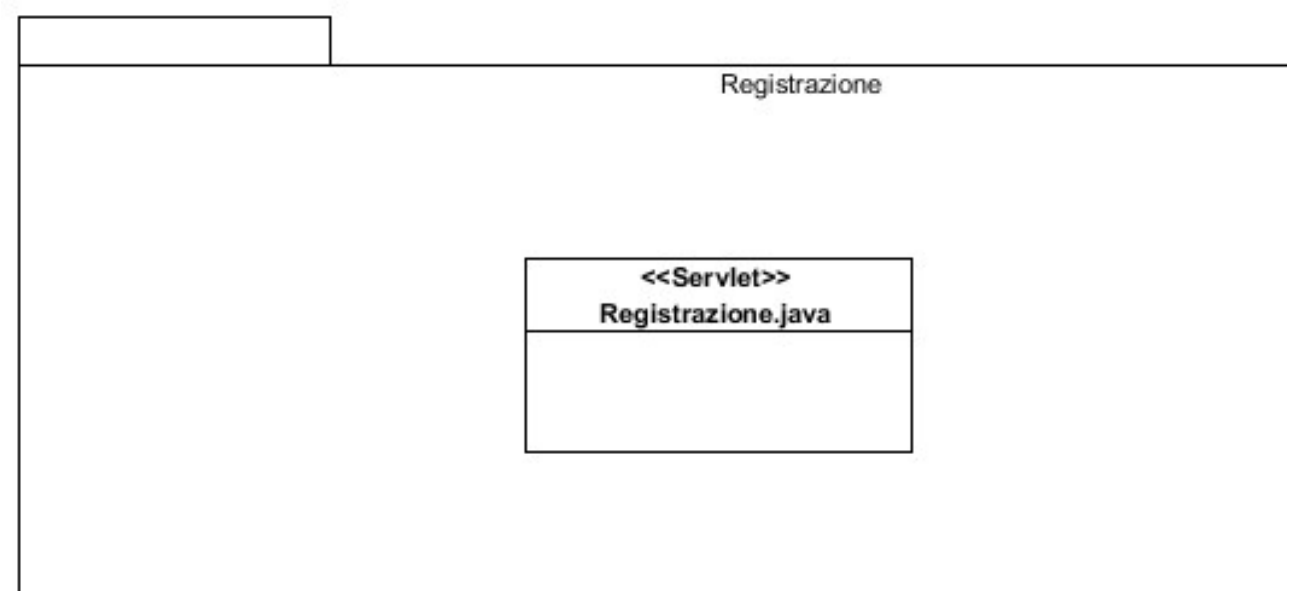
CLASSE	DESCRIZIONE
LoginServlet.java	Gestisce il login tra gli utenti
EmailSending.java	Gestisce il ricorda password

2.1.3.1 Account



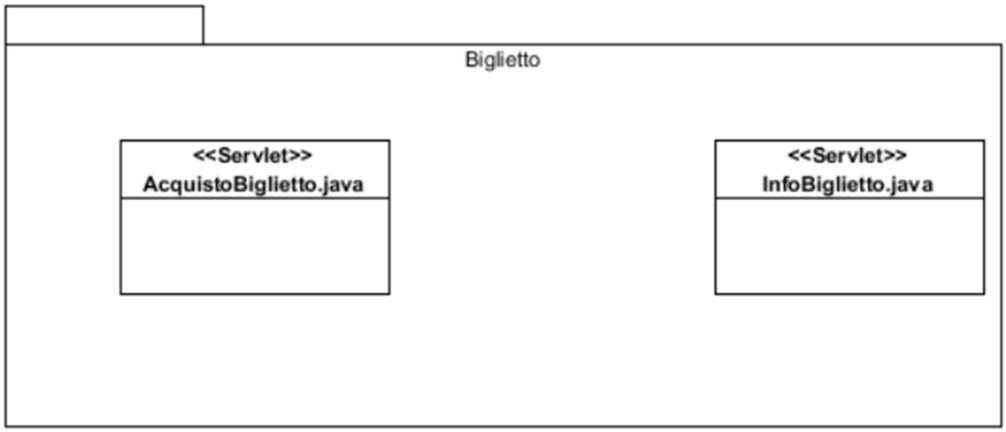
CLASSE	DESCRIZIONE
ModificaIndirizzoServlet.java	Permette di modificare l'indirizzo
ModificaPasswordServlet.java	Permette di modificare la password
ModificaEmailServlet.java	Permette di modificare l'email

2.1.3.2 Registrazione



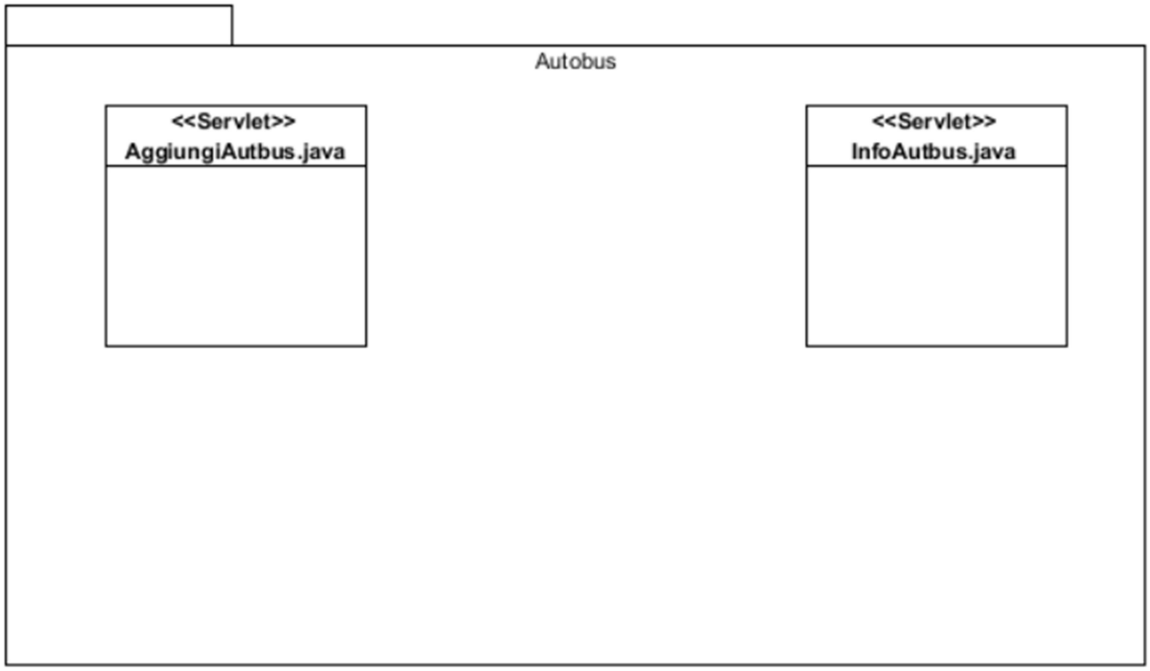
CLASSE	DESCRIZIONE
RegistrazioneServlet.java	Permette di registrarsi al sistema

2.1.3.3 Biglietto



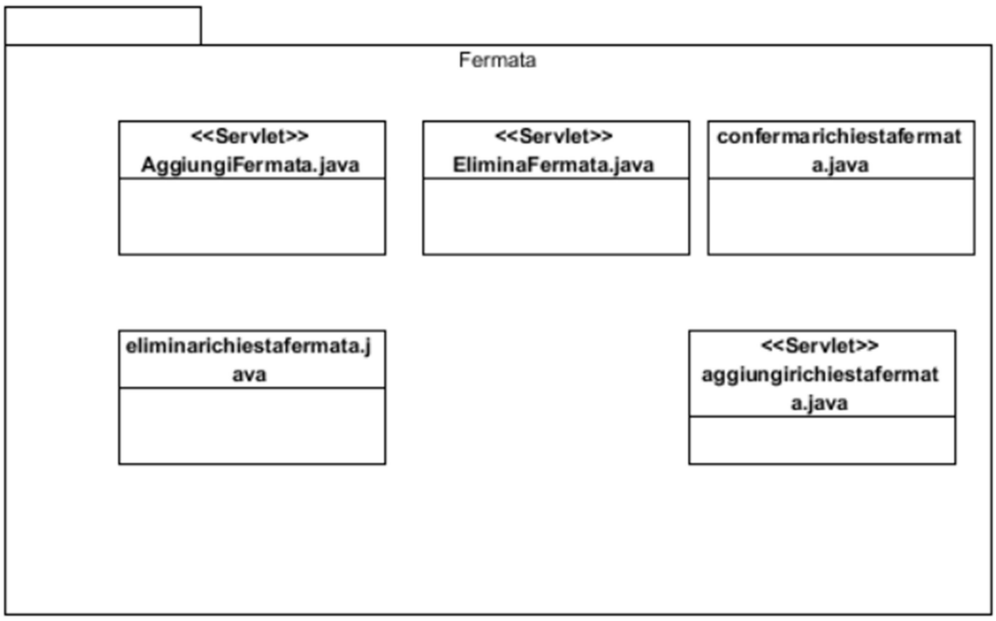
CLASSE	DESCRIZIONE
AcquistoBigliettoServlet.java	Permette di acquistare un biglietto per una corsa
InfoBigliettoServlet.java	Permette di ricevere informazioni su un determinato biglietto

2.1.3.4 Autobus



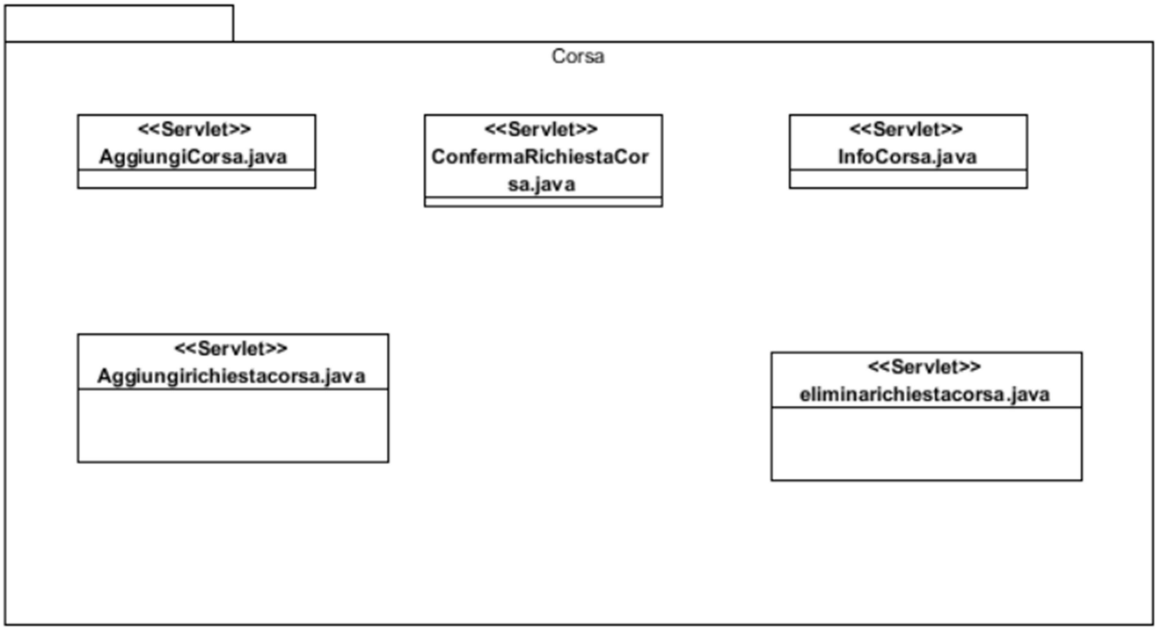
CLASSE	DESCRIZIONE
AggiungiAutobusServlet.java	Permette al manager di aggiungere un autobus
InfoAutobusServlet.java	Permette di ricevere informazioni su un determinato autobus

2.1.3.5 Fermata



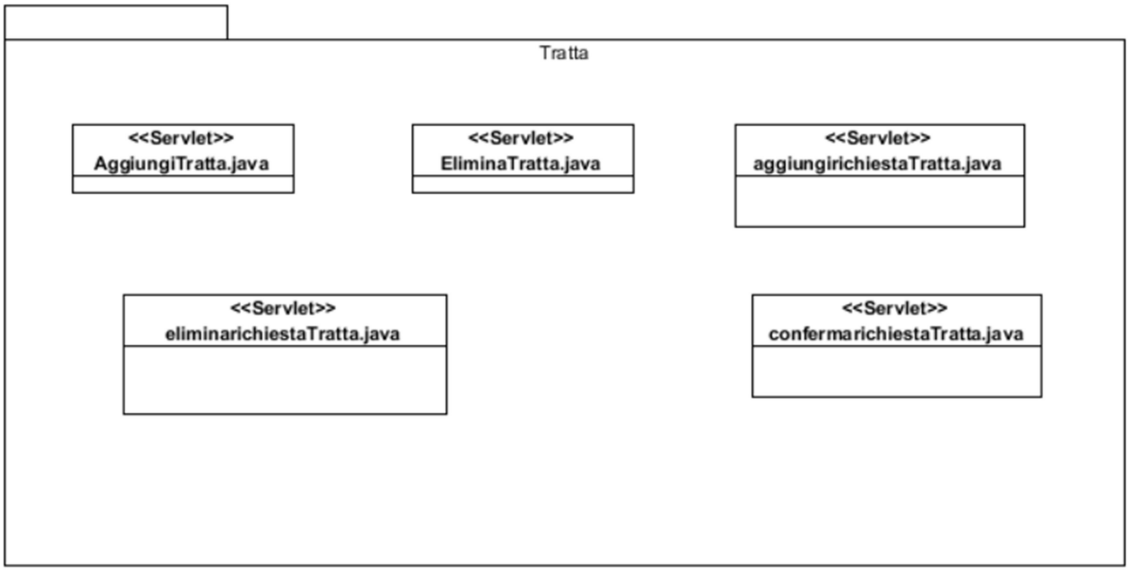
CLASSE	DESCRIZIONE
AggiungiFermataServlet.java	Permette di aggiungere una fermata
EliminaFermataServlet.java	Permette di eliminare una fermata
Confermarichiestafermata.java	Permette di confermare una richiesta fermata
eliminarichiestaFermataServlet.java	Permette di eliminare una richiestafermata
Aggiungirichiestafermata.java	Permette di aggiungere una richiesta di fermata

2.1.3.6 Corsa



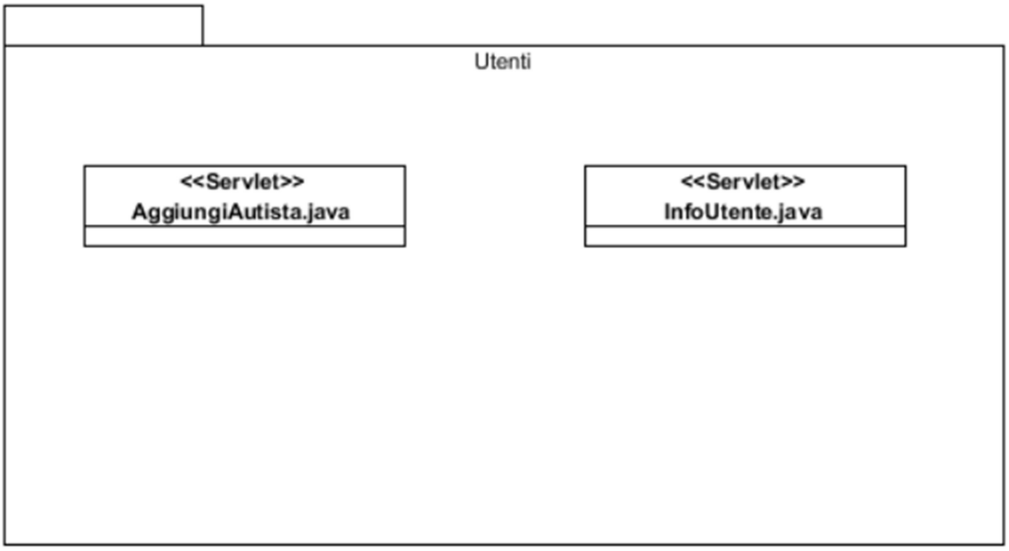
CLASSE	DESCRIZIONE
AggiungiCorsaServlet.java	Permette al manager di aggiungere una corsa
EliminaRichiestaCorsaServlet.java	Permette al manager di eliminare una richiesta corsa
InfoCorsaServlet.java	Permette di ricevere informazioni su una determinata corsa
Aggiungirichiestacorsa.java	Permette al manager di aggiungere una richiesta corsa
Confermarichiestacorsa.java	Aggiunge una corsa nell'elenco

2.1.3.7 Tratta



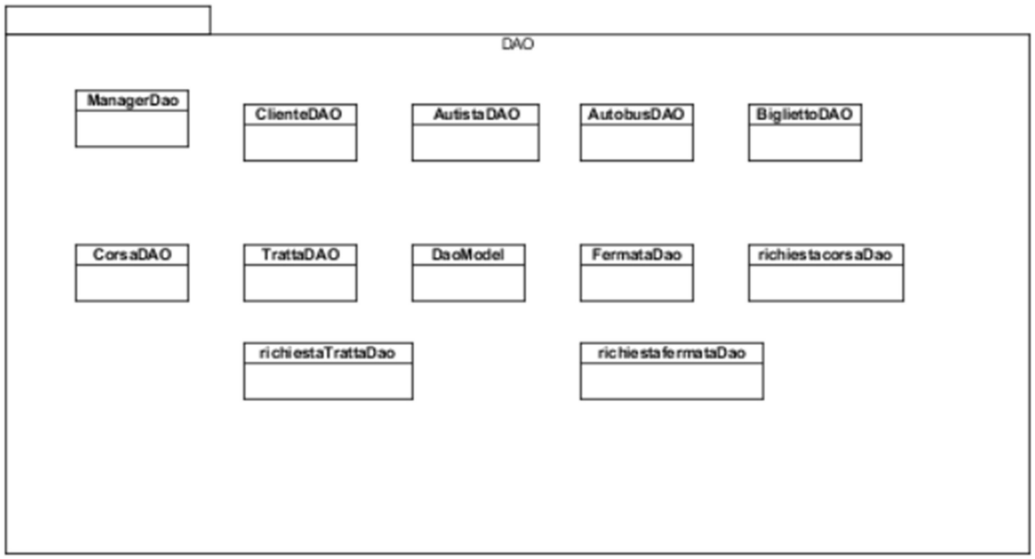
CLASSE	DESCRIZIONE
AggiungiTrattaServlet.java	Permette al manager di aggiungere una tratta
EliminaTrattaServlet.java	Permette al manager di eliminare una tratta
aggiungirichiestaTratta.java	Permette di ricevere informazioni su una determinata tratta
EliminaRichiestatìTratta.java	Permette al manager di modificare una tratta
Confermarichiestatratta.java	Permette di inserire la tratta nell'elenco delle tratte

2.1.3.8 Utenti



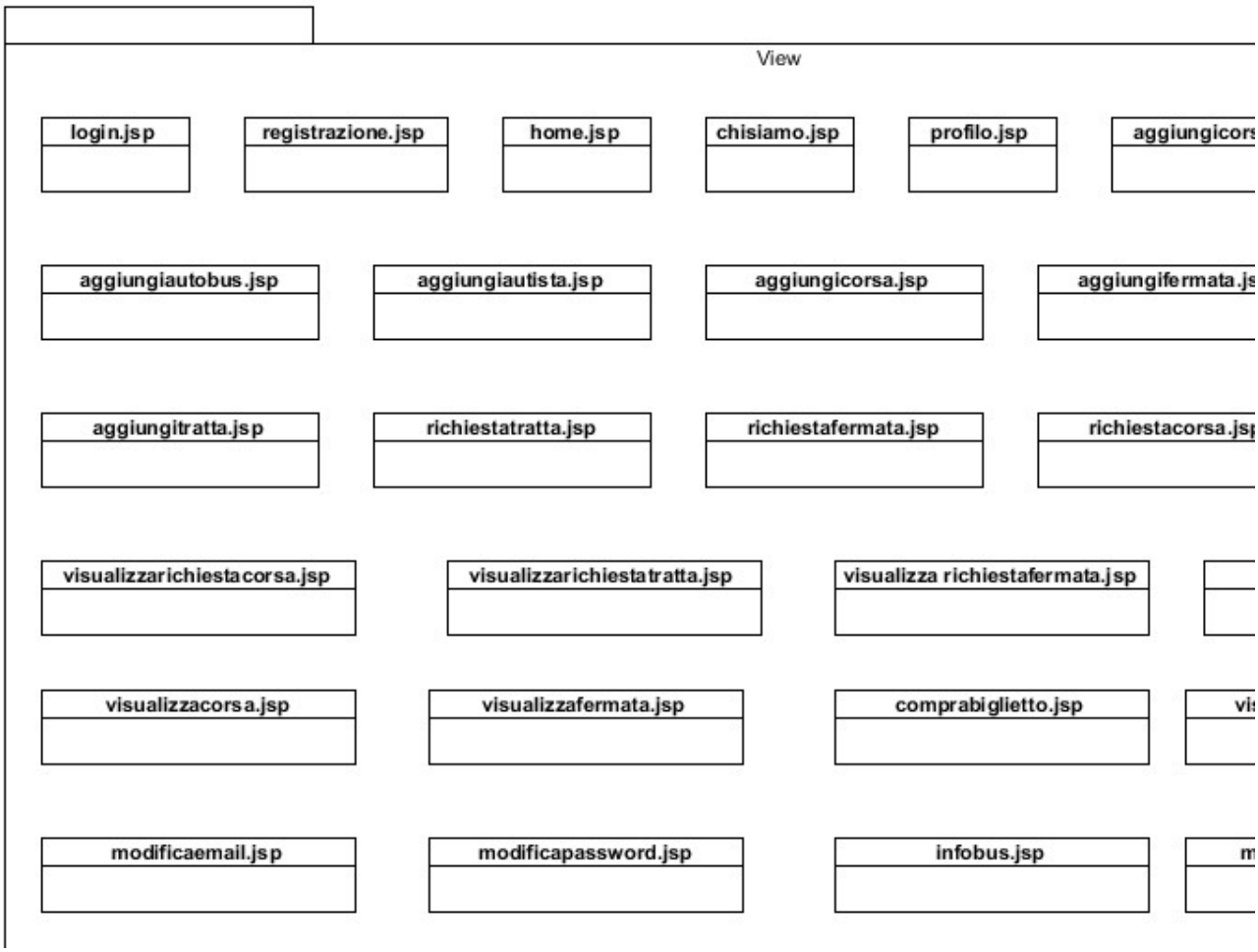
CLASSE	DESCRIZIONE
AggiungiAutistaServlet.java	Permette al manager di aggiungere un'autista
InfoUtente.java	Permette di ricevere informazioni su un determinato utente

2.1.4 Packages DAO



CLASSE	DESCRIZIONE
ManagerDAO.java	Permette di interagire con la tabella Admin all'interno del database
ClienteDAO.java	Permette di interagire con la tabella Cliente all'interno del database
AutistaDAO.java	Permette di interagire con la tabella Autista all'interno del database
AutobusDAO.java	Permette di interagire con la tabella Autobus all'interno del database
BiglettoDAO.java	Permette di interagire con la tabella Biglietto all'interno del database
CorsaDAO.java	Permette di interagire con la tabella Corsa all'interno del database
TrattaDAO.java	Permette di interagire con la tabella Tratta all'interno del database
FermataDAO.java	Permette di interagire con la tabella Fermata all'interno del database
richiestaTrattaDao.java	Permette di interagire con la tabella richiestaTratta all'interno del database
richiestaFermataDao.java	Permette di interagire con la tabella richiestaFermata all'interno del database
richiestaCorsaDao.java	Permette di interagire con la tabella richiestaCorsa all'interno del database

2.1.5 Packages View



CLASSE	DESCRIZIONE
login.jsp	Visualizza la pagina che permette di effettuare il login
Registrazione.jsp	Visualizza la pagina che permette di effettuare la registrazione
Home.jsp	Visualizza la pagina iniziale del sistema
Chisiamo.jsp	Visualizza la pagina che permette di vedere le informazioni riguardanti la piattaforma
Profilo.jsp	Visualizza la pagina che visualizza le informazioni dell'utente
Aggiungicorsa.jsp	Visualizza la pagina che permette di aggiungere una corsa nel sistema
Aggiungiautobus.jsp	Visualizza la pagina che permette di aggiungere un autobus nel sistema
Aggiungiautista.jsp	Visualizza la pagina che permette di aggiungere un'autista nel sistema
Aggiungicorsa.jsp	Visualizza la pagina che permette di aggiungere una corsa nel sistema

Aggiungifermata.jsp	Visualizza la pagina che permette di aggiungere una fermata nel sistema
Aggiungitratta.jsp	Visualizza la pagina che permette di aggiungere una tratta nel sistema
Richiestatratta.jsp	Visualizza la pagina che permette di inserire una richiesta di una tratta
Richiestafermata.jsp	Visualizza la pagina che permette di inserire una richiesta di una fermata
Richiestacorsa.jsp	Visualizza la pagina che permette di inserire una richiesta di una corsa
Visualizzarichiestacorsa.jsp	Visualizza la pagina che permette di visualizzare le richieste per una corsa
Visualizzarichiestatratta.jsp	Visualizza la pagina che permette di visualizzare le richieste per una tratta
Visualizzarichiestafermata.jsp	Visualizza la pagina che permette di visualizzare le richieste per una fermata
Visualizzatratta.jsp	Visualizza la pagina che permette di visualizzare la tratta
Visualizzacorsa.jsp	Visualizza la pagina che permette di visualizzare la corsa
Visualizzafermata.jsp	Visualizza la pagina che permette di visualizzare la fermata
Comprabiglietto.jsp	Visualizza la pagina che permette di comprare un biglietto per una corsa
Visualizzabiglietto.jsp	Visualizza la pagina che permette di visualizzare le informazioni di un determinato biglietto
Modificaemail.jsp	Visualizza la pagina che permette di modificare l'email dell'utente
Modificapassword.jsp	Visualizza la pagina che permette di modificare la password dell'utente
Infobus.jsp	Visualizza la pagina che permette di visualizzare le informazioni dell'autobus
Modificaindirizzo.jsp	Visualizza la pagina che permette di modificare l'indirizzo dell'utente
Eliminaaccount.jsp	Visualizza la pagina che permette di eliminare un account
Visualizzabigliettivenduti.jsp	Visualizza la pagina che permette di visualizzare una lista di biglietti venduti
Eliminaautista.jsp	Visualizza la pagina che permette di eliminare un'autista
Eliminacorsa.jsp	Visualizza la pagina che permette di eliminare una corsa
Eliminatratta.jsp	Visualizza la pagina che permette di eliminare una tratta
Eliminafermata.jsp	Visualizza la pagina che permette di eliminare una fermata
Infoautista.jsp	Visualizza la pagina che permette di visualizzare le informazioni di un'autista
Infocliente.jsp	Visualizza la pagina che permette di visualizzare le informazioni di un cliente

3.Interfaccia delle classi

3.1 Package Manager

3.1.0 Manager Autenticazione

Si tratta di una classe che funge da interfaccia del sottosistema che gestisce l'autenticazione.

METODO	DESCRIZIONE
public Utente login(String email, String password,Tipo utente)	Questo metodo permette di effettuare l'accesso
public boolean logout(HttpSession session)	Questo metodo permette di effettuare il logout

3.1.1 Manager Account

Si tratta di una classe che funge da interfaccia del sottosistema che gestisce l'account.

METODO	DESCRIZIONE
public boolean modificaPassword(String newPsw, String conferma password) context Account inv: self.isLogged() context Account::modificaPassword(String newPsw, String vecchiaPsw) pre: newPsw==vecchiaPsw context Account::modificaPassword(String newPsw, String vecchiaPsw) post: self.getPassword()==newPsw	Questo metodo permette di modificare la password corrente dell'utente e ritorna un booleano.
Public boolean modificaIndirizzo(String indirizzo) Context Account inv: self.isLogged() context Account::modificaIndirizzo(String indirizzo)pre: indirizzo. lenght()>3 && indirizzo. lenght()<25 context Account::modificaIndirizzo(String indirizzo)post: self.getIndirizzo()==indirizzo	Questo metodo permette di modificare un indirizzo.
Public boolean modificaEmail(String email) Context Account inv:	Questo metodo permette di modificare l'email

self.isLogged() context Account::modificaEmail(String email)pre: email. lenght()>3 && email. lenght()<25 && self.getEmail()!=email context Account::modificaEmail(String email)post: self.getEmail()==email	

3.1.2 Manager Autobus

Si tratta di una classe che funge da interfaccia del sottosistema che gestisce l'autobus.

METODO	DESCRIZIONE
public boolean aggiungiAutobus(String modello, int posti, int anno, int immatricolazione, int chilometri) Context Account inv: self.isLogged() context Autobus:: aggiungiAutobus(String modello, int posti, int immatricolazione, int chilometri) pre: modello. lenght()>3 && modello. lenght()<25 && posti >0 && posti <999 context Autobus:: aggiungiAutobus(String modello, int posti, int immatricolazione, int chilometri) post: getNumAutobus()= @pre.getNumAutobus()-1	Questo metodo permette di aggiungere un autobus e restituisce un booleano
Public String infoAutobus(Autobus autobus)	Questo metodo permette di ricevere le informazioni riguardanti l'autobus

3.1.3 Manager Biglietto

Si tratta di una classe che funge da interfaccia del sottosistema che gestisce il biglietto.

METODO	DESCRIZIONE
public boolean acquistaBiglietto(Corsa corsa, int numeri) Context Account inv: self.isLogged() context Biglietto::acquistaBiglietto(Corsa corsa, int numeri) pre: corsa.getPostiDisponibili()>= numeri	Questo metodo permette di acquistare un biglietto e ritorna un booleano.

context Biglietto::acquistaBiglietto(Corsa corsa, int numeri) post: corsa.getPostiDisponibili()=@pre.corsa.getPostiDisponibili()-numeri	
public String infoBiglietto(Biglietto biglietto)	Questo metodo permette di ricevere informazioni su un biglietto

3.1.4 Manager Corsa

Si tratta di una classe che funge da interfaccia del sottosistema che gestisce la corsa.

METODO	DESCRIZIONE
public boolean aggiungiCorsa(Tratta tratta, String orario, int durata, List<Fermata> fermate) Context Account inv: self.isLogged() context Corsa:: aggiungiCorsa(Tratta tratta, String orario, int durata, String data,List<Fermata> fermate) pre: orario.lenght()==5 && data.lenght()==10 && durata >0 context Corsa:: aggiungiCorsa(Tratta tratta, String orario, int durata, String data,List<Fermata> fermate) post: getNumCorse()=@pre.getNumCorse()-1	Questo metodo permette di aggiungere una corsa e ritorna un booleano.
Public String infoCorsa(Corsa corsa)	Questo metodo permette di ricevere informazioni su una corsa.

3.1.5 Manager Tratta

Si tratta di una classe che funge da interfaccia del sottosistema che gestisce la tratta.

METODO	DESCRIZIONE
public boolean AggiungiTratta(String cittàpartenza, String cittàvo) Context Account inv: self.isLogged()	Questo metodo permette di aggiungere una tratta e ritorna un booleano.

context Tratta:: AggiungiTratta(String cittàpartenza, String cittàvo) pre: cittàpartenza.lenght()>3 && cittàpartenza.lenght()<25 && cittàvo.lenght()>3 && cittàvo.lenght()<25 context Tratta:: AggiungiTratta(String cittàpartenza, String cittàvo) post: getNumTratta()=@pre.getNumTratta()-1	
Public String infoTratta(Tratta tratta)	Questo metodo permette di ricevere informazioni su una tratta.

3.1.6 Manager Fermata

Si tratta di una classe che funge da interfaccia del sottosistema che gestisce la fermata.

METODO	DESCRIZIONE
public boolean aggiungiFermata(String località) Context Account inv: self.isLogged() context Fermata::aggiungiFermata(String località) pre: località.lenght()>3 && località.lenght()<25 context Fermata:: aggiungiFermata(String località) post: getNumFermata()=@pre.getNumFermata()-1	Questo metodo permette di creare una fermata e ritorna un booleano.
Public String infoFermata(Fermata fermata)	Questo metodo permette di modificare un indirizzo.

4.Design Pattern

4.1 - Dao Pattern

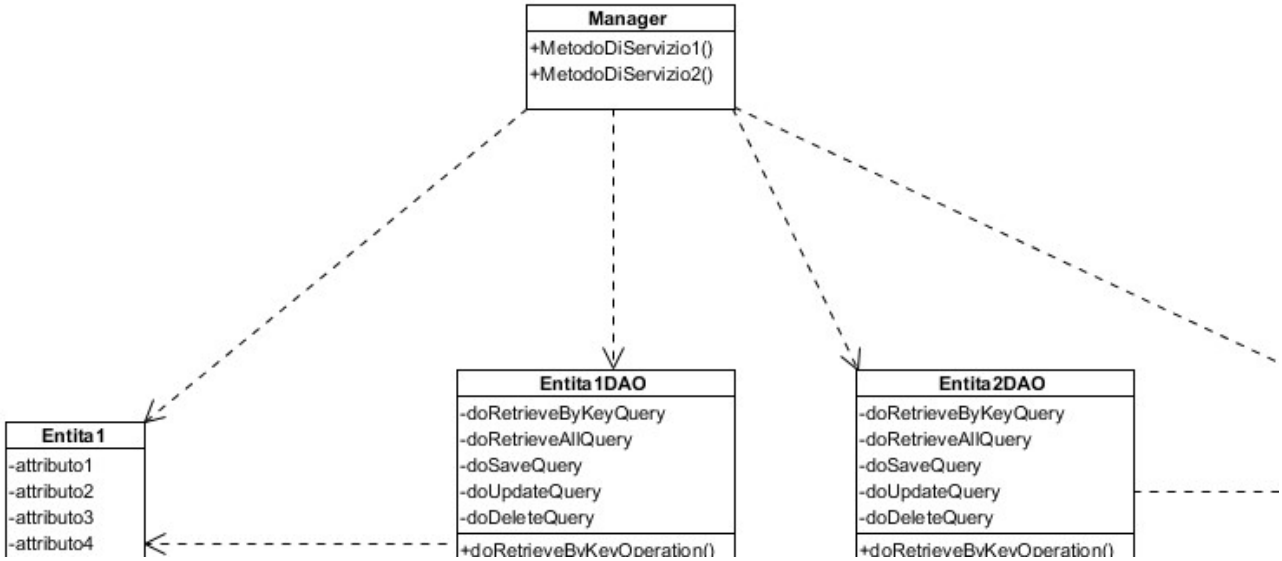
Per l'accesso al database è stato impiegato il pattern DAO (Data Access Object). Il pattern è usato per separare le API, di basso livello, di accesso ai dati, dalla logica di business di alto livello. Per ogni tabella presente nel DB esisterà una sua corrispondente classe DAO che possiederà metodi per effettuare le

operazioni CRUD. Ogni classe DAO utilizzerà esclusivamente la classe Entity corrispondente alla tabella su cui effettua le operazioni. Di seguito è presentato un modello di utilizzo del pattern:

4.2 - Façade Pattern

Per la realizzazione dei servizi dei sottosistemi è stato usato il pattern Façade. Il pattern si basa sull'utilizzo di un oggetto che permette, attraverso un'interfaccia più semplice, l'accesso a sottosistemi che espongono interfacce complesse e molto diverse tra loro, nonché a blocchi di codice complessi.

Nel nostro caso, ogni sottosistema, identificato nell'SDD, possiede una classe chiamata manager. Questa implementa dei metodi che corrispondono ai servizi offerti dal sottosistema. tali metodi, nella loro implementazione, utilizzeranno le classi entity e le classi DAO per eseguire il servizio da essi offerto. In output presenteranno un oggetto che servirà alla servlet per la presentazione del risultato dell'operazione. In questo modo si svincola completamente la logica di business, implementata dai manager, e la logica di controllo, implementata dalla servlet. Viene qui presentato un modello di utilizzo del pattern:


Manager
+MetodoDiServizio1()
+MetodoDiServizio2()
Entita1
-attributo1
-attributo2
-attributo3
-attributo4
Entita1DAO
-doRetrieveByKeyQuery
-doRetrieveAllQuery
-doSaveQuery
-doUpdateQuery
-doDeleteQuery
Entita2DAO
-doRetrieveByKeyQuery
-doRetrieveAllQuery
-doSaveQuery
-doUpdateQuery
-doDeleteQuery