Università degli Studi di Salerno

Corso di Ingegneria del Software

AirDreams
ODD

Partecipanti

Nome	**Matricola**
Noemi Cipriano	0512105472
Teresa Elia	0512105106
Rosaria Rossi	0512105658
Grazia Varone	0512105490

Indice

1. Introduzione……4
1.1 Object design trade-offs……4
1.2 Linee Guida per la Documentazione delle Interfacce……5
1.3 Definizioni, acronimi e abbreviazioni……5
2. Packages……6
2.1 Packages AirDreams……6
2.1.1 Package PresentationUtente……6
2.1.2 Package PresentationVolo……9
2.1.3 Package PresentationCarrello……10
2.1.4 Package PresentationOrdine……11
2.1.5 Package PresentationCompagniaAerea……12
2.2 Package Bean……13
2.3 Package Model……14
2.4 Class Interfaces……16
2.4.1 UtenteManager……16
2.4.2 BigliettoManager……17
2.4.3 VoloManager……18
2.4.4 PoliticaBagagliManager……20
2.4.5 OrdineManager……21
2.4.6 BagaglioManager……22
2.4.7 CarrelloManager……23
2.4.8 CartaDiCreditoManager……24
2.4.9 CompagniaAereaManager……25
2.4.10 AeroportoManager……26

1. Introduzione

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

1.1 Object design trade-offs

Portata vs. Latenza :
Il sistema utilizza un flusso guidato dagli eventi. Il Web server assegna un nuovo thread per ogni richiesta, consentendo così la gestione parallela delle richieste. Con un numero elevato di richieste, quindi, questo aumenta la produttività ma anche la latenza, rallentando complessivamente il sistema.

Spazio di memoria vs. Tempo di risposta :
Il sistema si propone di garantire risposte alle richieste degli utenti nel minor tempo possibile in relazione al carico del sistema; ciò comporta l'inevitabile incremento di spazio di memoria necessario, dovuto al bisogno di tener traccia di una mole di dati maggiore, accessibile il più velocemente possibile.

Funzionalità vs. Usabilità :
Il software incorpora funzionalità che soddisfano le specifiche dei requisiti. Con MySQL per gestire i dati persistenti, Java come linguaggio di programmazione, e JSP per strutturare le pagine web; quindi l'usabilità del sistema è pienamente raggiunta.

Rapido sviluppo vs. Funzionalità
Seguendo un modello per lo sviluppo di sistemi software, il processo ha incluso oltre la necessaria fase di implementazione, anche quella di analisi dei requisiti, progettazione e testing. Pertanto il sistema risultante è capace di soddisfare le specifiche funzionali promesse con un preciso approccio di sviluppo delle applicazioni.

Costo vs. riutilizzabilità
Le interfacce software e le componenti possono essere riutilizzate all'interno del sistema per implementare funzionalità per diversi moduli; tuttavia il software è stato progettato utilizzando Java per la logica e MySQL per l'accesso ai dati e pertanto l'uso di quest'ultimo potrebbe risultare costoso.

1.2 Linee Guida per la Documentazione delle Interfacce

Gli sviluppatori dovranno seguire le seguenti convenzioni per la scrittura del codice:

Naming Convention

- È buona norma utilizzare nomi:
  - Descrittivi
  - Pronunciabili
  - Di uso comune
  - Lunghezza medio-corta

Variabili

- I nomi delle variabili devono cominciare con una lettera minuscola. Se il nome della variabile è costituito da più parole, solo l'iniziale delle altre parole sarà maiuscola (notazione CamelCase), inoltre è possibile abbreviare il nome della variabile solo se non peggiora la leggibilità e comprensibilità.
- Le costanti dovranno essere scritte interamente in maiuscolo e se il nome è costituito da più parole vengono separate con l'underscore.

Metodi

- I nomi dei metodi devono cominciare con una lettera minuscola, e le parole seguenti con la lettera maiuscola (notazione CamelCase). 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().
- 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

I nomi delle classi devono iniziare con una lettera maiuscola.
Il nome deve fornire informazioni utili relative al loro scopo.
Ogni file sorgente contiene una singola classe e deve essere strutturato in un determinato modo:

- L'istruzione package che permette di inserire la classe in un determinato package
- L'istruzione import che importa le librerie necessarie alla class
- Una piccola descrizione della classe

1.3 Definizioni, acronimi e abbreviazioni

Il documento e il sistema riporteranno i seguenti acronimi per l'identificazione di strumenti e/o elementi usati durante lo sviluppo.

- RAD: Requirements Analysis Document
- SDD: System Design Document
- ODD: Object Design Document
- DBMS: Database Management System
- API: Application Programming Interface
- JDBC: Java Database Connectivity

- CamelCase: pratica che consiste nello scrivere parole composte unendo tutte le parole tra loro, ma lasciando le loro iniziali maiuscole.

2. Packages

Il sistema è diviso in 7 packages:

- Interfaccia utente
- Gestione utente
- Gestione volo
- Gestione carrello
- Gestione ordine
- Gestione compagnia aerea
- Data access

I package del sistema contengono sia classi atte ad implementare le funzionalità del sistema offerte all'utente, sia atte a gestire la logica, sia atte ad offrire risposte sotto forma di interfaccia grafica.

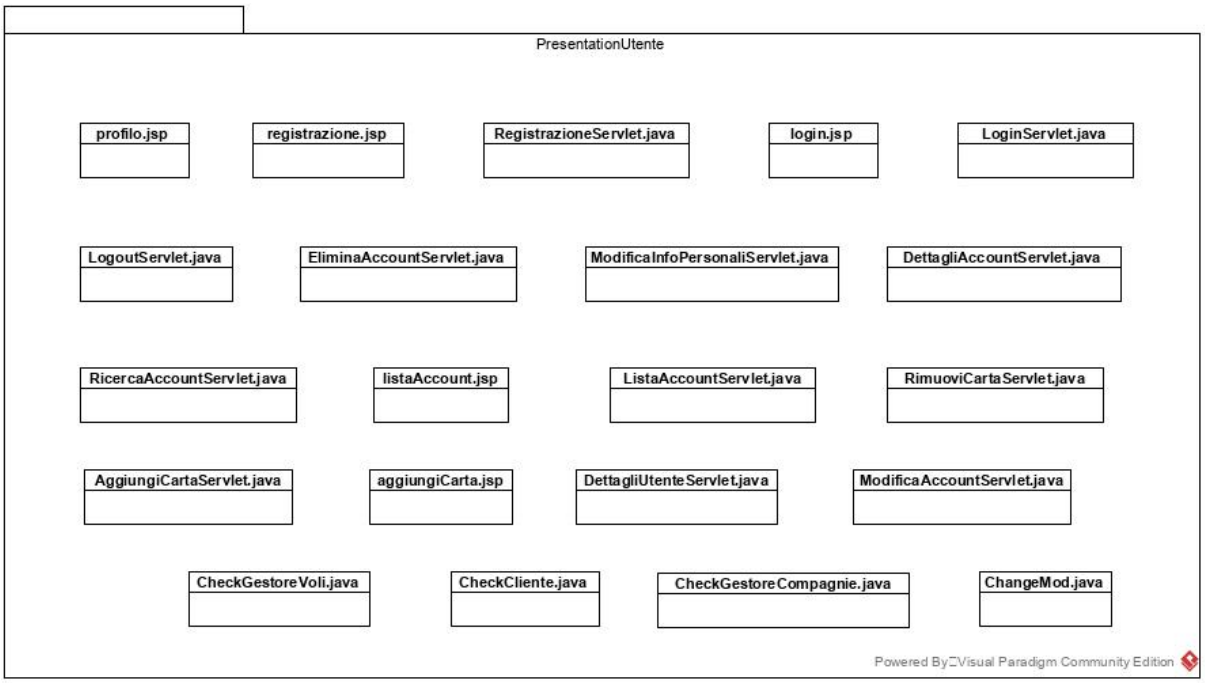
Interfaccia utente: rappresenta l'interfaccia del sistema, ed offre la possibilità all'utente di interagire con quest'ultimo, offrendo sia la possibilità di inviare, in input, che di visualizzare, in output, dati.

Gestione utente, gestione volo, gestione carrello, gestione ordine, gestione compagnia aerea: si occupano dell'elaborazione dei dati da inviare al client.

Data access: ha il compito di memorizzare i dati sensibili del sistema, utilizzando un DBMS che gestisce i dati. La comunicazione verso il DBMS è realizzata tramite le API JDBC.

2.1 Packages AirDreams

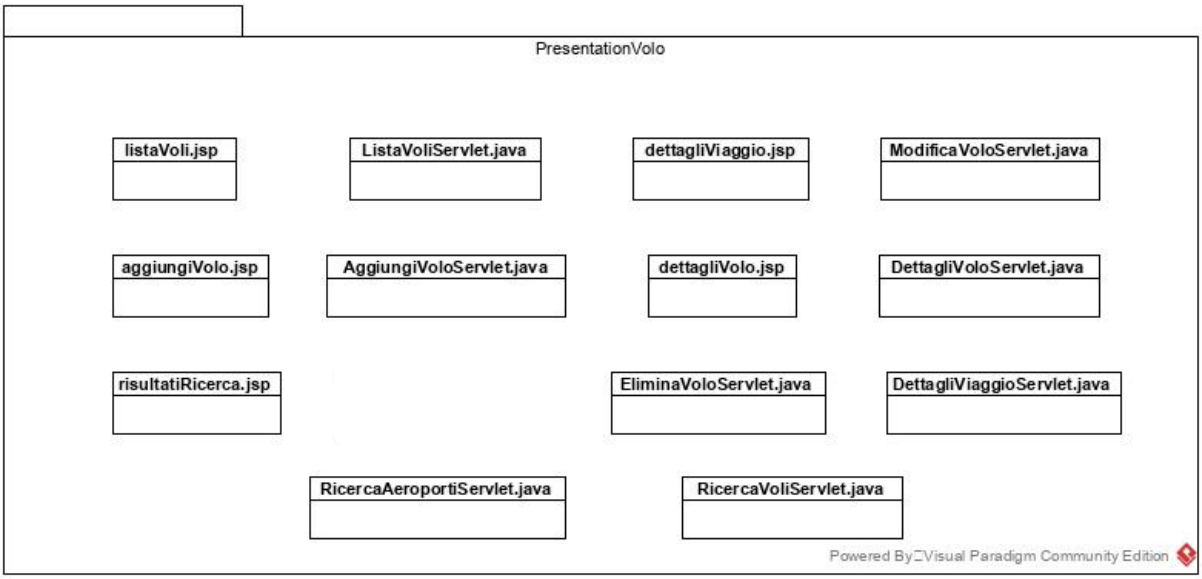
2.1.1 Package PresentationUtente



Classe	Descrizione
profilo.jsp	Pagina che rappresenta l’area personale dell’utente.
aggiungiCarta.jsp	Pagina che mostra un form per aggiungere una carta di credito.
login.jsp	Pagina che mostra un form per effettuare l’accesso al sistema.
registrazione.jsp	Pagina che mostra un form per registrarsi al sistema, creando un account che fornisce l’accesso al sistema.
listaAccount.jsp	Pagina che mostra la lista di tutti gli account registrati al sistema.
AggiungiCartaServlet	Questa servlet gestisce le operazioni di inserimento di una carta di credito all'account dell'utente correntemente loggato, che potrà utilizzarla per l'acquisto di biglietti.
ModificaInfoPersonaliServlet	Questa servlet permette di modificare le informazioni personali di un determinato account.
ModificaAccountServlet	Questa servlet gestisce le operazioni per la modifica delle informazioni personali di un utente, a cura di un gestore compagnie.
EliminaAccountServlet	Questa servlet permette di eliminare l’account di un utente.
RimuoviCartaServlet	Questa servlet permette di rimuovere la carta di credito di un utente.
LoginServlet	Questa servlet permette di far effettuare l’accesso al sistema ad un determinato utente.
RegistrazioneServlet	Questa servlet permette di effettuare la registrazione al sistema da parte di un utente.
LogoutServlet	Questa servlet permette di far disconnettere un utente dal proprio account.

ListaAccountServlet	Questa servlet permette di visualizzare tutti gli account registrati al sistema.
RicercaAccountServlet	Questa servlet permette al gestore delle compagnie di ricercare determinati account.
DettagliAccountServlet	Questa servlet permette di visualizzare i dettagli di un determinato account.
DettagliUtenteServlet	Questa servlet gestisce tutte le operazioni per la visualizzazione dei dettagli di un dato utente da parte del gestore compagnie.
ChangeMod	Questa servlet gestisce tutte le operazioni per permettere il cambio di ruolo da parte di un utente che generalizza le figure di cliente, gestore voli e gestore compagnie.
CheckGestoreVoli	Questa servlet filter permette di controllare il ruolo dell'utente correntemente loggato per permettere o negare l'accesso alle pagine riservate al gestore voli.
CheckGestoreCompagnie	Questa servlet filter permette di controllare il ruolo dell'utente correntemente loggato per permettere o negare l'accesso alle pagine riservate al gestore compagnie.
CheckCliente	Questa servlet filter permette di controllare il ruolo dell'utente correntemente loggato per permettere o negare l'accesso alle pagine riservate al cliente.

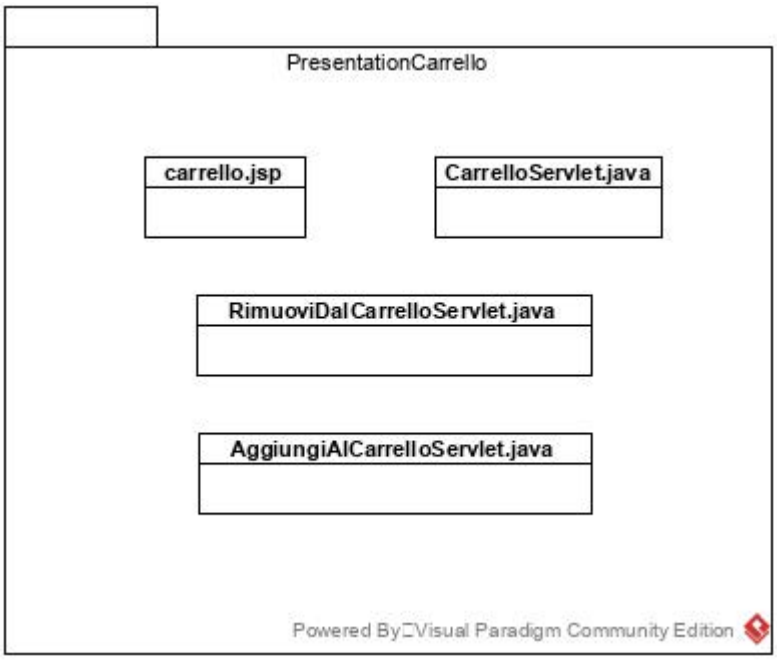
2.1.2 Package PresentationVolo



Classe	**Descrizione**
listaVoli.jsp	Pagina che mostra una lista di diversi voli con le informazioni principali.
dettagliVolo.jsp	Pagina che mostra i precisi dettagli di un singolo volo.
aggiungiVolo.jsp	Pagina che mostra un form che permette di inserire informazioni riguardo un volo da aggiungere al catalogo del sistema.
dettagliViaggio.jsp	Pagina che mostra
risultatiRicerca.jsp	Pagina che mostra i risultati di una ricerca di voli richiesta da un utente.
ListaVoliServlet	Questa servlet permette di visualizzare una certa lista di voli.
AggiungiVoloServlet	Questa servlet permette di aggiungere un volo al catalogo del sistema.
EliminaVoloServlet	Questa servlet permette di rimuovere un volo dal catalogo del sistema.
ModificaVoloServlet	Questa servlet permette di aggiornare un volo presente nel catalogo del sistema.
DettagliVoloServlet	Quest servlet permette di visualizzare i

	dettagli di un volo presente nel catalogo del sistema.
DettagliViaggioServlet	Questa servlet gestisce le operazioni per la visualizzazione di un dato volo recuperato dalla lista dei risultati di una ricerca.
RicercaVoliServlet	Questa servlet gestisce le operazioni per la ricerca di voli secondo determinati criteri.
RicercaAeroportiServlet	Questa servlet gestisce tutte le operazioni per la ricerca AJAX di aeroporti nel sistema.

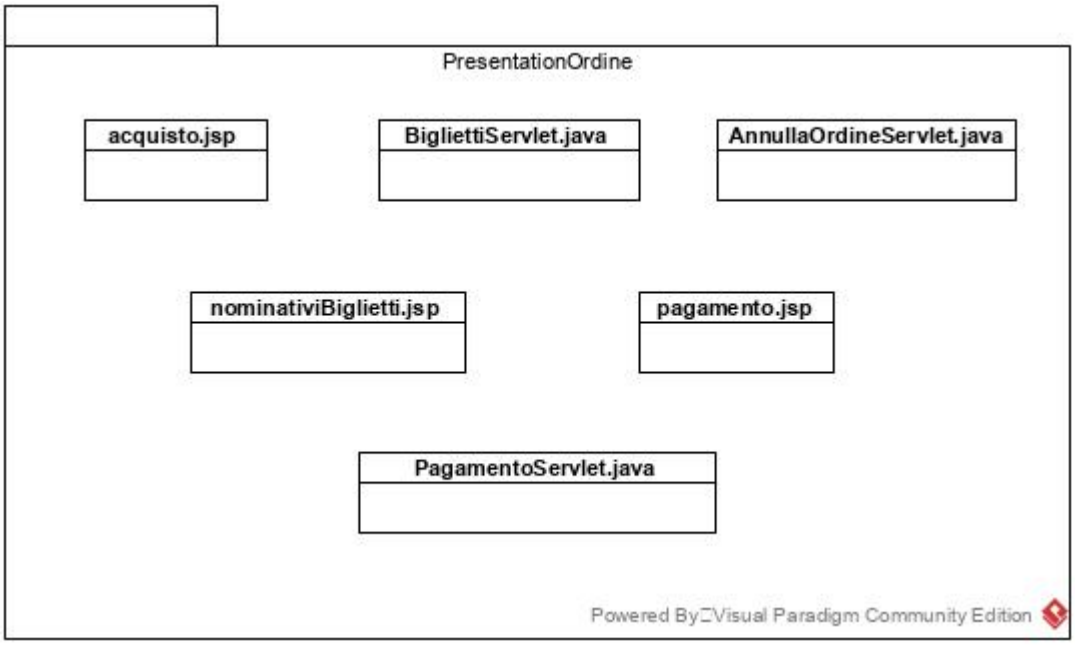
2.1.3 Package PresentationCarrello



Classe	**Descrizione**
carrello.jsp	Pagina che mostra il carrello di un utente con gli eventuali voli inseriti all'interno del carrello.
CarrelloServlet	Questa servlet permette di visualizzare il carrello relativo ad un utente.
AggiungiAlCarrelloServlet	Questa servlet permette di aggiungere un volo al carrello relativo ad un utente.
RimuoviDalCarrelloServlet	Questa servlet permette di rimuovere un

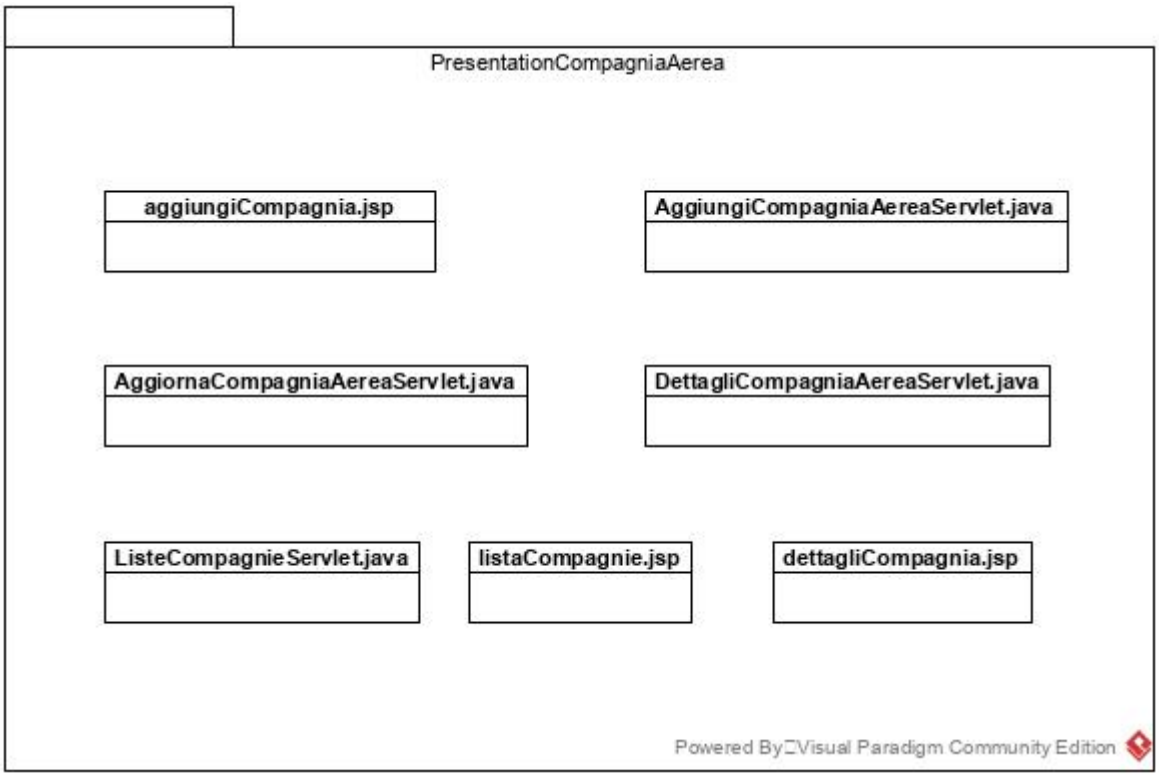
	volo dal carrello relativo ad un utente.

2.1.4 Package PresentationOrdine



Classe	Descrizione
pagamento.jsp	Pagina che permette di selezionare una carta di credito già registrata, o di inserirne una al momento per concludere un ordine.
acquisto.jsp	Pagina che mostra una notifica di successo per l'avvenuto acquisto di biglietti da parte dell'utente sul sistema.
nominativiBiglietti.jsp	Pagina che mostra, durante la procedura di un ordine, il form per inserire i dati personali dei passeggeri ed aggiungere eventuali bagagli scelti.
PagamentoServlet	Questa servlet gestisce tutte le operazioni per il pagamento di biglietti da parte di un utente correntemente loggato al sistema.
AnnullaOrdineServlet	Questa servlet permette di annullare un ordine effettuato da un utente.
BigliettiServlet	Questa servlet permette di creare biglietto/i durante la procedura di ordine.

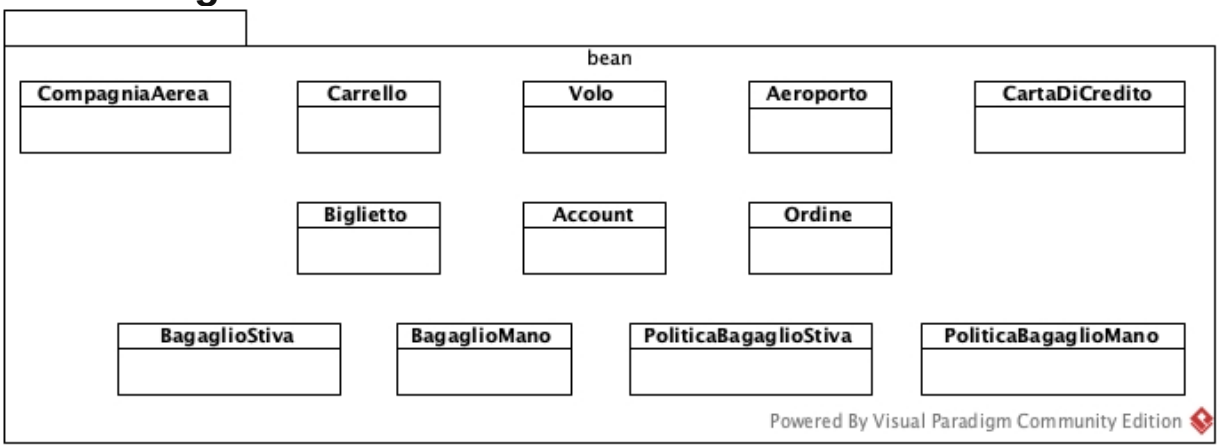
2.1.5 Package PresentationCompagniaAerea



Classe	Descrizione
aggiungiCompagnia.jsp	Pagina che mostra un form per aggiungere una nuova compagnia aerea al sistema.
listaCompagnie.jsp	Pagina che mostra una lista di tutte le compagnie aeree i cui voli sono offerti dalla piattaforma.
dettagliCompagnia.jsp	Pagina che mostra i precisi dettagli di una compagnia aerea.
AggiungiCompagniaAereaServlet	Questa servlet permette di aggiungere una nuova compagnia aerea all'interno del sistema.
AggiornaCompagniaAereaServlet	Questa servlet permette di aggiornare i dettagli di una compagnia aerea esistente all'interno del sistema.
DettagliCompagniaAereaServlet	Questa servlet permette di visualizzare i dettagli di una compagnia aerea esistente all'interno del sistema.
ListaCompagnieServlet	Questa servlet gestisce tutte le operazioni per la visualizzazione di tutta

	la lista delle compagnie aeree i cui voli sono offerti dal sistema e singolarmente gestiti da un determinato gestore voli.

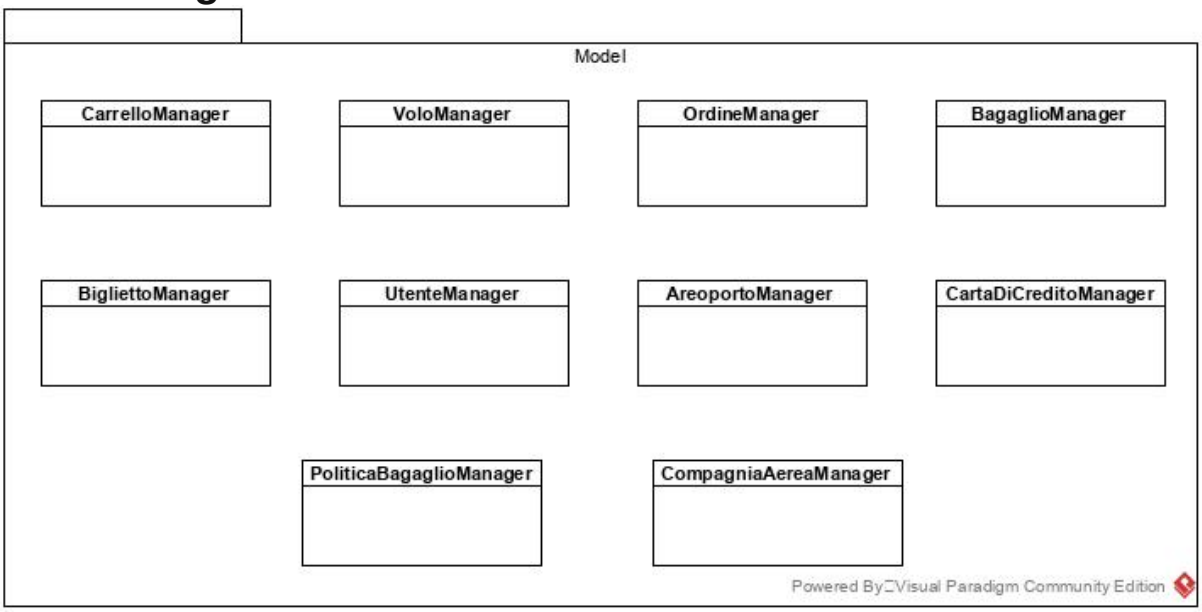
2.2 Package Bean



Classe	Descrizione
Ordine	Questa classe rappresenta l'ordine concluso da un cliente.
Carrello	Questa classe rappresenta il carrello assegnato ad ogni cliente.
Volo	Questa classe contiene tutte le informazioni di un possibile volo offerto dal sistema.
Aeroporto	Questa classe rappresenta l'aeroporto, ovvero un luogo dove sono installati tutti gli impianti necessari al decollo, atterraggio e allo stazionamento degli aerei.
CartaDiCredito	Questa classe rappresenta una carta di credito memorizzata per un cliente e che può essere utilizzata per l'acquisto di biglietti aerei.
Biglietto	Questa classe rappresenta il biglietto per un determinato volo con le sue informazioni.
CompagniaAerea	Questa classe tiene traccia della compagnia aerea con le relative informazioni.
BagaglioMano	Questa classe rappresenta il bagaglio a mano incluso nel biglietto di un singolo passeggero con le relative dimensioni.

BagaglioStiva	Questa classe rappresenta il bagaglio a stiva incluso nel biglietto di un singolo passeggero con le relative dimensioni e prezzo.
PoliticaBagaglioMano	Questa classe definisce la politica sul bagaglio a mano di una singola compagnia aerea.
PoliticaBagaglioStiva	Questa classe definisce la politica sul bagaglio a stiva di una singola compagnia aerea.
Account	Questa classe rappresenta l'utente registrato, ovvero un qualsiasi utente che si è registrato al sistema e può effettuare il login in qualunque momento, accedendo alla propria area personale.

2.3 Package Model



CarrelloManager	Questa classe si occupa di implementare i metodi per effettuare l'inserimento, la cancellazione e la ricerca di un carrello relativo ad un particolare utente, memorizzato all'interno del DB.
VoloManager	Questa classe si occupa di implementare i metodi per effettuare l'inserimento, la cancellazione e la ricerca di un volo presente nel catalogo di sistema, memorizzato all'interno del DB.

OrdineManager	Questa classe si occupa di implementare i metodi per effettuare l'inserimento, la cancellazione e la ricerca di un ordine relativo ad un dato utente che lo ha effettutato, memorizzato all'interno del DB.
BagaglioManager	Questa classe si occupa di implementare i metodi per effettuare l'inserimento, la cancellazione e la ricerca di un bagaglio relativo ad un dato biglietto presente in un ordine di un particolare cliente, memorizzato all'interno del DB.
BigliettoManager	Questa classe si occupa di implementare i metodi per effettuare l'inserimento, la cancellazione e la ricerca di un biglietto relativo ad un dato ordine di un cliente, memorizzato all'interno del DB.
UtenteManager	Questa classe si occupa di implementare i metodi per effettuare l'inserimento, la cancellazione, la modifica e la ricerca di un utente registrato al sistema, memorizzato all'interno del DB.
AeroportoManager	Questa classe si occupa di implementare i metodi per effettuare l'inserimento, la cancellazione e la ricerca di un aeroporto possibilmente utilizzato in un volo come luogo di partenza o destinazione, memorizzato all'interno del DB.
CompagniaAereaManager	Questa classe si occupa di implementare i metodi per effettuare l'inserimento, la cancellazione e la ricerca di una compagnia aerea di cui il sistema offre i voli, memorizzata all'interno del DB.
PoliticaBagaglioManager	Questa classe si occupa di implementare i metodi per effettuare l'inserimento, la cancellazione e la ricerca di una politica bagaglio relativa ad una data compagnia aerea, memorizzata all'interno del DB.
CartaDiCreditoManager	Questa classe si occupa di implementare i metodi per effettuare l'inserimento, la cancellazione e la ricerca di una carta di credito associata ad un dato utente, memorizzata all'interno del DB.

2.4 Class interfaces

2.4.1 UtenteManager

UtenteManager	
signUp	**Context:** UtenteManager.signUp(utente) **Pre:** utente != null && utente.email != null && utente.nome != null && utente.cognome != null && utente.password != null && getListaUtenti() -> forAll(u|u.email != utente.email); **Post:** getListaUtenti() -> include(utente)
signIn	**Context:** UtenteManager.signIn(utente) **Pre:** utente != null && utente.email != null && utente.password != null
eliminaAccount	**Context:** UtenteManager.eliminaAccount(email) **Pre:** email != null && getListaUtenti() -> exists(u|u.email==email) **Post**: !getListaUtenti() -> exists(u|u.email==email)
findAccountByLetter	**Context:** UtenteManager.findAccountByLetter(nome, cognome) **Pre:** nome != null && cognome != null **Post:** getListaUtenti() -> Select((u|u.nome.chartAt(0)==nome) || (u|u.cognome.charAt(0)==cognome))
aggiornaProfilo	**Context:** UtenteManager.aggiornaProfilo(utenteVecchio, utenteNuovo) **Pre:** utenteNuovo != null && utenteNuovo.email != null && utenteNuovo.nome != null && utenteNuovo.cognome != null && utenteNuovo.password != null && utenteVecchio,email != null && getListaUtenti().reject(u|u.email == utenteVecchio.email) -> forAll(u|u.email!=utenteNuovo.email) && getListaUtenti() ->

	exists(u|u.email==utenteVecchio.email) **Post:** getListaUtenti() ->\ include(u|u.email==utenteNuovo.email)
findAccountByEmail	**Context:** UtenteManager.findAccountByEmail(email) **Pre:** getListaUtenti() ->\ exists(u|u.email==email) && email != null **Post:** getListaUtenti() ->\ select(u|u.email==email)
getAllUsers	**Context:** UtenteManager.getAllUsers() **Post:** getListaUtenti()

2.4.2 BigliettoManager

BigliettoManager	
aggiungiBiglietto	**Context:** BigliettoManager.aggiungiBiglietto(biglietto) **Pre:** biglietto.nome != null && biglietto.cognome != null && biglietto.sesso != null && biglietto.prezzoBiglietto != null && biglietto.ordine != null && biglietto.volo != null **Post:** getListaBiglietti() -> include(biglietto)
trovaBigliettiOrdine	**Context:** BigliettoManager.trovaBigliettiOrdine(codOrdine) **Pre:** biglietto.codOrdine != 0 && getListaBiglietti() ->\ exists(b|b.ordine==codOrdine) **Post:** getListaBiglietti() ->\ select(b|b.ordine==codOrdine)

2.4.3 VoloManager

VoloManager	
aggiungiVolo	**Context:** VoloManager.aggiungiVolo(volo)

	Pre: volo.dataPart != null && volo.prezzo != null && volo.postiDisponibili != null && volo.durata != null && volo.orarioPart != null && aeroportoPart != null && aeroportoArr != null && compagniaAerea != null && volo.bagaglioStivaCompreso != null **Post:** getListaVoli() -> include(volo)
eliminaVolo	**Context:** VoloManager.rimuoviVolo(idVolo) **Pre:** getListaVoli() -> exists(v|v.idVolo==idVolo) && idVolo != 0 **Post:** !getListaVoli()->exists(v|v.idVolo==idVolo)
modificaVolo	**Context:** VoloManager.modificaVolo(volo) **Pre:** volo.dataPart != null && volo.prezzo != null && volo.postiDisponibili != null && volo.durata != null && volo.orarioPart != null && aeroportoPart != null && aeroportoArr != null && compagniaAerea != null && volo.bagaglioStivaCompreso != null && getListaVoli() -> exists(v|v.idVolo==volo.idVolo) **Post:** getListaVoli() -> include(volo)
findById	**Context:** VoloManager.findById(idVolo) **Pre:** idVolo != null && getListaVoli() ->\nexists(v|v.idVolo==idVolo) **Post:** getListaVoli() -> select(v|v.idVolo==idVolo)
cercaVoli	**Context:** VoloManager.cercaVoli(ricerca,\ncompagnia) **Pre:** ricerca != null && compagnia != null && ((ricerca.aeroportoPart != null) || (ricerca.aeroportoArr != null) || (dataPart != null)) **Post:** getListaVoli() ->\nselect(a|a.aeroportoPart == ricerca.aeroportoPart) || getListaVoli() ->\nselect (a|a.aeroportoArr == ricerca.aeroportoArr) || getListaVoli() ->\nselect (a|a.dataPart == dataPart)
cercaVoli	**Context:** VoloManager.cercaVoli(compagnia)

	Pre: compagnia != null && getListaVoli() ->> exists(a|a.compagniaAerea == compagnia) **Post:** getListaVoli() ->> select(a|a.compagniaAerea == compagnia)
cercaDueScali	**Context:** VoloManager.cercaDueScali(aeroportoP, aeroportoA, datatDepartureLd, passeggeri, durata, prezzo) **Pre:** aeroportoP != null && aeroportoA != null && datatDepartureLd != null && passeggeri != null && durata != null && prezzo != null **Post:** getListaVoli() ->> select (a|a.aeroportoPart == aeroportoP && a.aeroportoArr == aeroportoA && a.dataPart == datatDepartureLd)
cercaUnoScalo	**Context:** VoloManager.cercaUnoScalo(aeroportoP, aeroportoA, datatDepartureLd, passeggeri, durata, prezzo) **Pre:** aeroportoP != null && aeroportoA != null && datatDepartureLd != null && passeggeri != null && durata != null && prezzo != null **Post:** getListaVoli() ->> select (a|a.aeroportoPart == aeroportoP && a.aeroportoArr == aeroportoA && a.dataPart == datatDepartureLd)
cercaDiretti	**Context:** VoloManager.cercaDiretti(aeroportoP, aeroportoA, datatDepartureLd, passeggeri, durata, prezzo) **Pre:** aeroportoP != null && aeroportoA != null && datatDepartureLd != null && passeggeri != null && durata != null && prezzo != null **Post:** getListaVoli() ->> select (a|a.aeroportoPart == aeroportoP && a.aeroportoArr == aeroportoA && a.dataPart == datatDepartureLd)

2.4.4 PoliticaBagaglioManager

PoliticaBagagliManager	
aggiungiPoliticaBagaglioStiva	**Context:** PoliticaBagaglioManager.aggiungiPoliticaBagaglioStiva(politicaBagaglio) **Pre:** politicabagaglio.peso != null && politicabagaglio.dimensioni != null && politicabagaglio.compagniaAerea != null && politicabagaglio.prezzo != null **Post:** getPolitiche Stiva() -> include(politicaBagaglio)
aggiungiPoliticaBagaglioMano	**Context:** PoliticaBagaglioManager.aggiungiPoliticaBagaglioMano(politicaBagaglio) **Pre:** politicabagaglio.peso!=null && politicabagaglio.dimensioni!=null && politicabagaglio.compagniaAerea!=null **Post:** getPoliticheMano() -> include(politicaBagaglio)
aggiornaPoliticaBagaglioMano	**Context:** PoliticaBagaglioManager.aggiornaPoliticaBagaglioMano(politicaBagaglio) **Pre:** getPoliticheMano() -> exists(p|p.compagniaAerea == politicaBagaglio.compagniaAerea) && politicaBagaglio != null **Post:** getPoliticheMano() -> include(p|p.compagniaAerea == politicaBagaglio.compagniaAerea)
aggiornaPoliticaBagaglioStiva	**Context:** PoliticaBagaglioManager.aggiornaPoliticaBagaglioStiva(politicaBagaglio) **Pre:** getPoliticheStiva() -> exists(p|p.compagniaAerea == politicaBagaglio.compagniaAerea) && politicaBagaglio != null **Post:** getPoliticheStiva() -> include(p|p.compagniaAerea == politicaBagaglio.compagniaAerea)
trovaPoliticaCompagniaStiva	**Context:**

	PoliticaBagaglioManager.trovaPoliticaCompagniaStiva(nome) **Pre:** getPoliticheStiva() -> exists(p|p.compagniaAerea == nome) **Post:** getPoliticheStiva() -> select(p|p.compagniaAerea == nome)
trovaPoliticaCompagniaMano	**Context:** PoliticaBagaglioManager.trovaPoliticaCompagniaMano(nome) **Pre:** getPoliticheMano() -> exists(p|p.compagniaAerea == nome) **Post:** getPoliticheMano() -> select(p|p.compagniaAerea == nome)

2.4.5 OrdineManager

OrdineManager	
aggiungiOrdine	**Context:** ordineManager.aggiungiOrdine(ordine) **Pre:** ordine!=null && ordine.dataAcquisto !=null && ordine.ncarta!=null && ordine.email!=null **Post:** getOrdini()->include(ordine) && ordine.biglietti -> forAll(b|b.volo.getPostiDisponibili()== b.volo.@pre.getPostiDisponibili()-1)
cercaOrdiniUtente	**Context:** ordineManager.cercaOrdiniUtente(email) **Pre**: email != null && getListaOrdini() -> exists(o|o.email == email) **Post**: getListaOrdini() -> select(o|o.email == email)
annullaOrdine	**Context:** ordineManager.annullaOrdine(codice) **Pre**: codice != null && getListaOrdini() -> exists(o|o.codOrdine == codice) **Post**: !getListaOrdini() -> select(o|o.codOrdine == codice) && ordine.biglietti -> forAll(b|b.volo.getPostiDisponibili()== b.volo.@pre.getPostiDisponibili()+1)

2.4.6 BagaglioManager

BagaglioManager	
aggiungiBagaglioMano	**Context:** bagaglioManager.aggiungiBagaglioMano(bagaglioMano) **Pre:** bagaglioMano != null && bagaglioMano.peso =! null && bagaglioMano.dimensioni != null && bagaglioMano.biglietto != null **Post:** getBagagliMano() -> include(bagaglioMano)
aggiungiBagaglioStiva	**Context:** bagaglioManager.aggiungiBagaglioStiva(bagaglioStiva) **Pre:** bagaglioStiva!=null && bagaglioStiva.peso =! null && bagaglioStiva.dimensioni !=null && bagaglioStiva.prezzo !=null && bagaglioStiva.quantity !=null && bagaglioStiva.biglietto != null **Post:** getBagagliStiva() -> include(bagaglioStiva)
cercaBagaglioManoBiglietto	**Context:** bagaglioManager.cercaBagaglioManoBiglietto(biglietto) **Pre**: biglietto != null && getBagagliMano() -> exists(b|b.biglietto.codBiglietto == biglietto.codBiglietto) **Post**: getBagagliMano() -> select(b|b.biglietto.codBiglietto == biglietto.codBiglietto)
cercaBagaglioStivaBiglietto	**Context:** bagaglioManager.cercaBagaglioStivaBiglietto(biglietto) **Pre**: biglietto != null && getBagagliStiva() -> exists(b|b.biglietto.codBiglietto == biglietto.codBiglietto) **Post**: getBagagliStiva() -> select(b|b.biglietto.codBiglietto == biglietto.codBiglietto)

2.4.7 CarrelloManager

CarrelloManager	
getCarrelloUtente	**Context**: CarrelloManager.getCarrelloUtente(email) **Pre**: email != null && getCarrelli() -> exists(c|c.email == email) **Post**: getCarrelli() -> select(c|c.email == email);
aggiungiVoloAlCarrello	**Context:** CarrelloManager.aggiungiVoloAlCarrello(email, id, quantity) **Pre**: email != null && id != null && quantity != null && getCarrelli() -> exists(c|c.email == email) **Post**: (getCarrelli()->select(c|c.email==email).getVoli().size==(getCarrelli()->select(c|c.email==email).getVoli().size+1)
rimuoviVoloDalCarrello	**Context**: CarrelloManager.rimuoviVoloDalCarrello(email, id) **Pre**: getCarrelli() -> exists(v|v.email == email) && id != null && email != null **Post:** (getCarrelli() -> select(c|c.email==email).getVoli().size==(getCarrelli() -> select(c|c.email==email).getVoli().size-1)
updateQuantity	**Context:** CarrelloManager.updateQuantity(email, id, quantity) **Pre:** email != null && id != null && quantity != null && getCarrelli() -> exists(c|c.email == email) **Post**: (getCarrelli() -> select(c|c.email==email).getVoli().size==(

	getCarrelli() -> select(c|c.email==email).getVoli().size+1
cercaVoloNelCarrello	**Context:** CarrelloManager.cercaVoloNelCarrello(email, id) **Pre**: id != null && email != null **Post:** getCarrelli() ->\ select(c|c.email==email).getVoli(). getIdVolo() == id
scuotaCarrello	**Context:** CarrelloManager.svuotaCarrello(email) **Pre:** getCarrelli() ->\ exists(c|c.email == email) **Post**: !getCarrelli() ->\ exists(c|c.email == email)

2.4.8 CartaDiCreditoManager

CartaDiCreditoManager	
creaCartaDiCredito	**Context:** CartaDiCreditoManager.creaCartaDiCredito(carta DiCredito) **Pre:** cartaDiCredito!=null && cartaDi cartaDiCredito.email =! null && cartaDiCredito.nCarta =! null && cartaDiCredito.titolare !=null && cartaDiCredito.dataScadenza!=null && cartaDiCredito.cvc!=null **Post:** getCarte() -> include(cartaDiCredito)
eliminaCarta	**Context:** CartaDiCreditoManager.eliminaCartaDiCredito(nc arta, email) **Pre:** getCarte() -> exists(c|c.nCarta == ncarta) && cartaDi cartaDiCredito.email =! null && cartaDiCredito.nCarta =! null **Post:** !getCarte() -> exists(c|c.nCarta == ncarta)
findAll	**Context:** CartaDiCreditoManager.findAll(email) **Pre:** getListCarte() ->\ exists(c|c.email == email) && email != null **Post:** getListCarte() ->

	select(c|c.email == email)
cercaCarta	**Context:** CartaDiCreditoManager.cercaCarta (nCarta, email) **Pre**: getCarte() -> exists(c|c.email == email) && nCarta != null && email != null **Post:** getCarte() -> select(c|c.email==email && c.nCarta == nCarta)

2.4.9 CompagniaAereaManager

CompagniaAereaManager	
aggiungiCompagnia	**Context:** CompagniaAreaManager.aggiungiCompagnia(compagniaArea) **Pre:** compagniaArea != null && compagniaAerea.nome =! null && compagniaAerea.sito != null **Post:** getCompagnieAeree() -> include(compagniaAerea)
visualizzaInfoCompagniaAerea	**Context:** compagniaAereaManager.visualizzaInfoCompagniaAerea(nome) **Pre:** getCompagnieAeree() -> exists(c|c.nome == nome) && nome != null **Post:** getCompagnieAeree() -> select(c|c.nome == nome)
aggiornaCompagnia	**Context:** compagniaAereaManager.aggiornaCompagnia(compagniaAerea) **Pre:** getCompagnieAeree() -> exists(c|c.compagniaAerea == compagniaAerea) && compagniaAerea.nome =! null && compagniaAerea.sito != null **Post:** getCompagnieAeree() -> include(c|c.compagniaAerea == compagniaAerea)
getAllCompanies	**Context:** compagniaAereaManager.getAllCompagnie()

	Post: getListCompagnieAeree()
eliminaCompagnia	**Context:** compagniaAereaManager.eliminaCompagnia(nome) **Pre:** nome != null && getListCompagnieAeree() -> exists(c|c.nome == nome) **Post:** !getListCompagnieAeree() -> exists(c|c.nome == nome)

2.4.10 AeroportoManager

AeroportoManager	
getAeroportiByCity	**Context:** AeroportoManager.getAeroportiByCity(cityAeroporto) **Pre:** cityAeroporto != null && getAeroporti() -> exists(a|a.city == cityAeroporto) **Post:** getAeroporti() -> select(a|a.city == cityAeroporto)
findAeroportoById	**Context:** AeroportoManager.findAeroportoById(idAeroporto) **Pre:** idAeroporto !=null && getAeroporti() -> exists(a|a.codice == idAeroporto) **Post:** getAeroporti() -> select(a|a.codice == idAeroporto)