|  |
| --- |
| **Università degli Studi di Salerno  Corso di Ingegneria del Software** |

**NoteMarket  
Object Design Document  
Versione 1.5**

****

Data: 15/02/2021

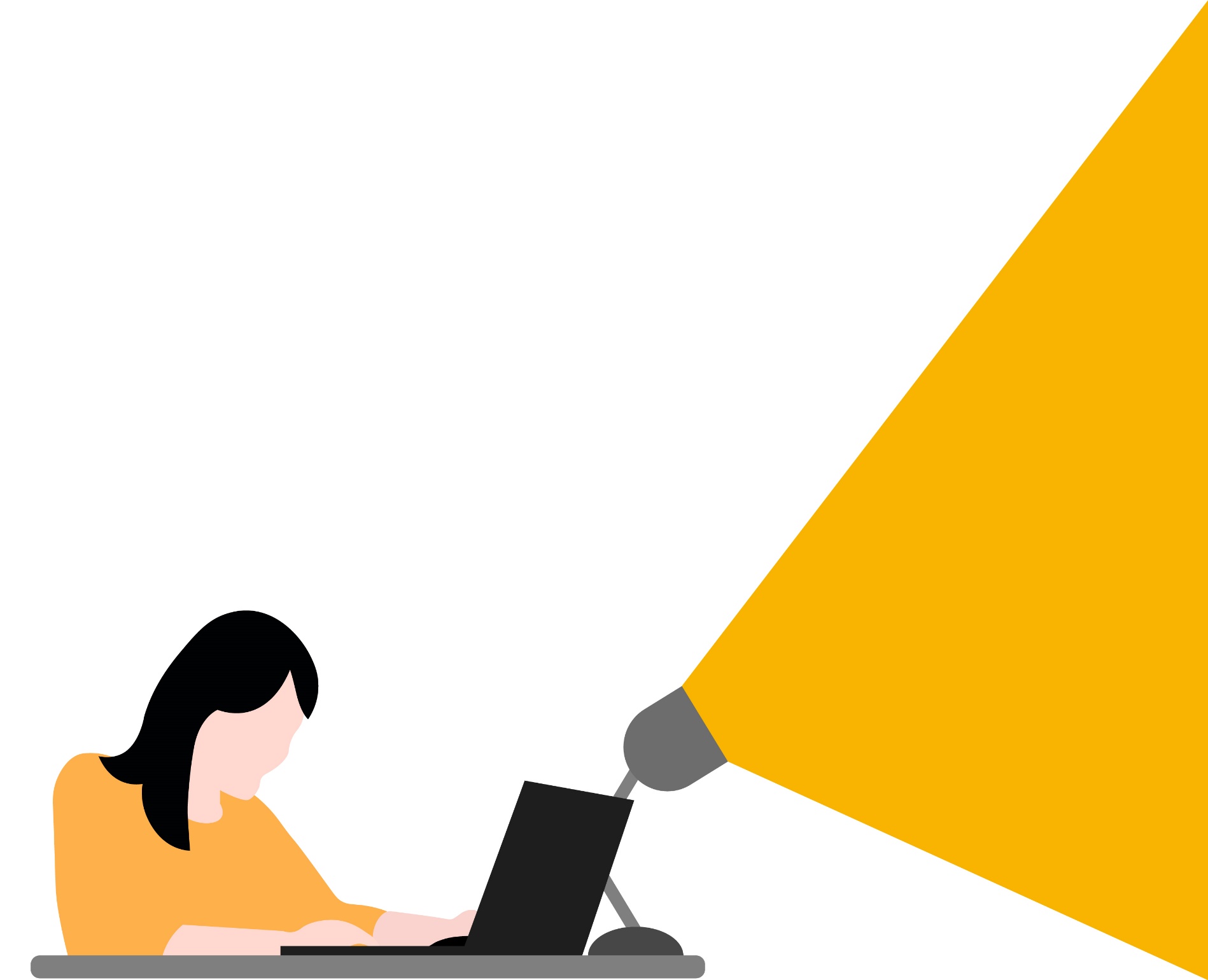
**Immagine che contiene disegnando

Descrizione generata automaticamente**

**Partecipanti:**

|  |  |
| --- | --- |
| Nome | Matricola |
| Mario De Riggi | 0512106024 |
| Antonio Cirillo | 0512106096 |
| Carmine Amendola | 0512106072 |

|  |  |
| --- | --- |
| **Scritto da:** | Mario De Riggi, Antonio Cirillo, Carmine Amendola |

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| Data | Versione | Descrizione | Autore |
| 22/01/2021 | 1.0 | Stesura del punto di Introduzione. | Carmine Amendola |
| 22/01/2021 | 1.0 | Design pattern: Singleton. | Mario De Riggi Antonio Cirillo |
| 23/01/2021 | 1.1 | Descrizione vista generale dei package. | Carmine Amendola |
| 23/01/2021 | 1.1 | Descrizione del package bean. | Antonio Cirillo |
| 06/02/2021 | 1.2 | Descrizione del package dao, manager e controller. | Antonio Cirillo |
| 11/02/2021 | 1.3 | Aggiunte interfacce classi: AccountManager, CartaDiCreditoManager, ImmagineProfiloManager, SaldoManager, AppuntoLibreriaManager, CarrelloManager. | Antonio Cirillo |
| 11/02/2021 | 1.4 | Aggiunte interfacce classi: ProdottoLibreriaManager, VideolezioneLibreriaManager, AppuntoManager, ModeratoreCatalogoManager, ModeratoreMasterManager, TutorManager, VideolezioneManager. | Antonio Cirillo |
| 15/02/2021 | 1.5 | Revisione completa del documento. | Antonio Cirillo Mario De Riggi Carmine Amendola |

Sommario

[**1.** **Introduction** 4](#_Toc64294892)

[**1.1.** **Object design trade-offs** 4](#_Toc64294893)

[**1.2.** **Interface documentation guidelines** 4](#_Toc64294894)

[**2.** **Design** **Pattern** 6](#_Toc64294895)

[**2.1.** **Singleton Pattern** 6](#_Toc64294896)

[**2.2.** **Facade Pattern** 6](#_Toc64294897)

[**3.** **Packages** 8](#_Toc64294898)

[**3.1.** **General vision about packages** 9](#_Toc64294899)

[**3.2.** **Package bean** 10](#_Toc64294900)

[**3.3.** **Package** **dao** 12](#_Toc64294901)

[**3.4.** **Package manager** 16](#_Toc64294902)

[**3.5.** **Package** **controller** 21](#_Toc64294903)

[**4.** **Class Interfaces** 25](#_Toc64294904)

[**4.1.** **AccountManager** 25](#_Toc64294905)

[**4.2.** **CartaDiCreditoManager** 26](#_Toc64294906)

[**4.3.** **ImmagineProfiloManager** 27](#_Toc64294907)

[**4.4.** **SaldoManager** 27](#_Toc64294908)

[**4.5.** **AppuntoLibreriaManager** 28](#_Toc64294909)

[**4.6.** **CarrelloManager** 28](#_Toc64294910)

[**4.7.** **ProdottoLibreriaManager** 29](#_Toc64294911)

[**4.8.** **VideolezioneLibreriaManager** 29](#_Toc64294912)

[**4.9.** **AppuntoManager** 30](#_Toc64294913)

[**4.10.** **ModeratoreCatalogoManager** 31](#_Toc64294914)

[**4.11.** **ModeratoreMasterManager** 31](#_Toc64294915)

[**4.12.** **TutorManager** 33](#_Toc64294916)

[**4.13.** **VideolezioneManager** 33](#_Toc64294917)

1. **Introduction**

Il seguente documento, redatto in seguito alla stesura di RAD ed SDD, ha lo scopo di produrre un modello capace di descrivere in modo coerente le diverse funzionalità individuate nelle fasi precedenti della progettazione. In particolare, nell’ ODD ci soffermeremo su:   
- trade-offs generali realizzati dagli sviluppatori;   
- linee guida relative alla documentazione delle interfacce e alle convenzioni di codifica;   
- informazioni relative alle interfacce delle classi, alle operazioni, ai tipi ed ai signatures dei sottosistemi definiti nel System Design.

* 1. **Object design trade-offs**
* **Interfaccia VS Usabilità**: il sistema sarà sviluppato con un’ interfaccia user friendly, ossia più chiara ed intuitiva possibile. L’interfaccia presenterà vari form, menu, pulsanti, sistemi di notifiche che nel loro insieme garantiranno la semplicità di utilizzo da parte dell’utente e la semplicità di comunicazione con l’utente stesso;
* **Sicurezza VS Efficienza**: come descritto nei Requisiti non Funzionali, si dovrà garantire un livello di sicurezza adeguato, negando l’accesso agli utenti non autorizzati, ai dati degli utenti ed ai documenti presenti nel sistema. Le password inserite dagli utenti saranno crittografate tramite l’utilizzo della crittografia MD5, in quanto permette solo l’encrypt ma non il decrypt dei dati. Tutto ciò non dovrà in alcun modo andare ad inficiare sull’esperienza degli utenti**.**
* **Comprensibilità VS Tempo**: il codice, sviluppato usando il linguaggio Java, dovrà essere più comprensibile possibile in modo da facilitare future modifiche e cambiamenti ed in modo da facilitare il lavoro in fase di testing. La volontà di rendere il codice quanto più chiaro e leggibile possibile incrementerà però il tempo di sviluppo.
  1. **Interface documentation guidelines**

Gli sviluppatori seguiranno alcune linee guida per la scrittura del codice:

* **Naming convention**: è buona norma usare nomi descrittivi, facilmente assimilabili alla loro funzione, di lunghezza medio-corta, utilizzando solo caratteri consentiti;
* **Variabili**: i nomi delle variabili devono cominciare con lettera minuscola ed eventuali parole seguenti con lettera maiuscola. Le variabili saranno dichiarate a fine blocco nel modo più leggibile possibile; in alcuni casi si utilizzerà il carattere (“\_”) per il naming;
* **Metodi**: i nomi dei metodi devono cominciare con lettera minuscola ed eventuali parole seguenti con lettera maiuscola. Tipicamente i nomi dei metodi identificheranno un’azione, seguito dal nome di un oggetto. I nomi dei metodi per accesso e modifica delle variabili dovranno essere del tipo getNomeVariabile() e setNomeVariabile();
* **Classi e pagine**: i nomi delle classi e delle pagine devono cominciare con lettera maiuscola ed anche eventuali parole seguenti. I nomi devono fornire informazioni sullo scopo della classe. La dichiarazione di classe è caratterizzata da:
* dichiarazione della classe pubblica;
* dichiarazioni di costanti;
* dichiarazioni di variabili di classe;
* dichiarazioni di variabili d’istanza;
* costruttore.
  1. **Definitions, acronyms, and abbreviations**

**RAD:** Requirements Analysis Document;

**SDD:** System Design Document;

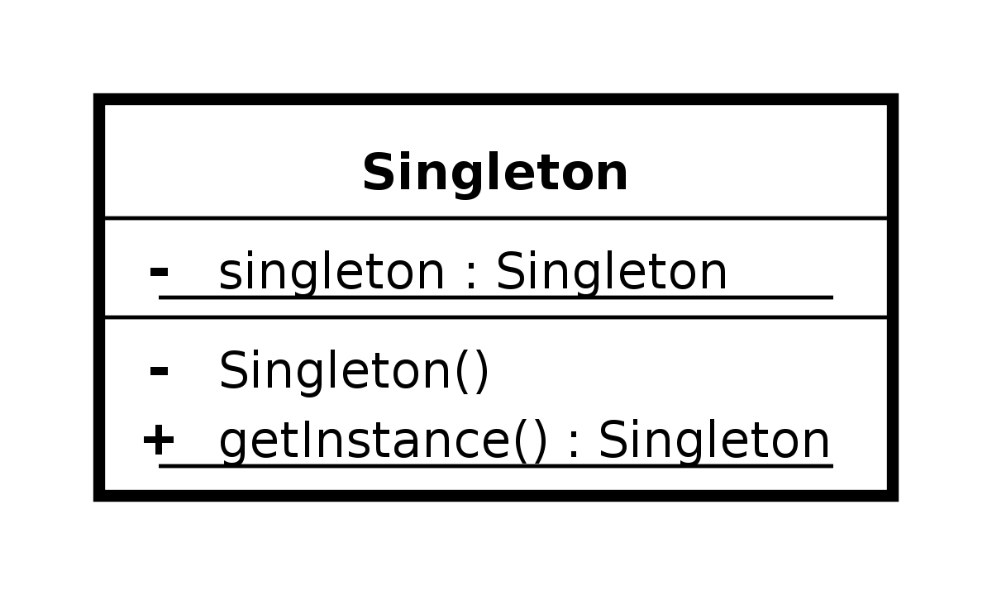
**ODD:** Object Design Document.

**DPD:** Documento Persistenza dei Dati.

* 1. **References**

RAD, SDD, DPD.

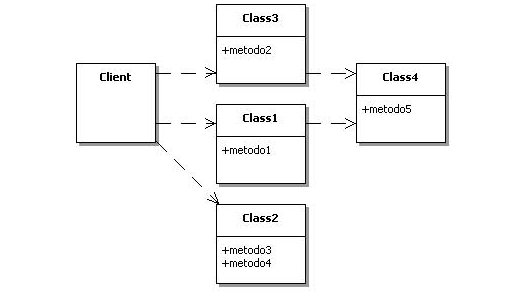
1. **Design** **Pattern**
   1. **Singleton Pattern**

****

Il singleton è un design pattern creazionale che ha lo scopo di garantire la creazione di una ed una sola istanza per una determinata classe e di fornire un punto di accesso globale a tale istanza. L'implementazione più semplice di questo pattern prevede che la classe singleton abbia un unico costruttore privato, in modo da impedire l'istanziazione diretta della classe. La classe fornisce inoltre un metodo "getter" statico che restituisce l'istanza della classe (sempre la stessa), creandola preventivamente o alla prima chiamata del metodo, e memorizzandone il riferimento in un attributo privato anch'esso statico. Il secondo approccio si può classificare come basato sul principio della lazy initialization (letteralmente "inizializzazione pigra") in quanto la creazione dell'istanza della classe viene rimandata nel tempo e messa in atto solo quando ciò diventa strettamente necessario (al primo tentativo di uso). Utilizziamo il singleton pattern per collegare le classi al nostro DB tramite la classe: DriverManagerConnectionPool.java .

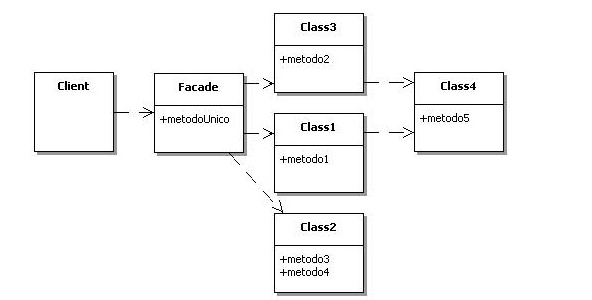
* 1. **Facade Pattern**

Il facade pattern, dall’inglese facade “facciata”, indica 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.



In questo diagramma, per esempio, viene mostrata la situazione in cui una classe Client per realizzare una singola operazione deve accedere ad alcune classi molto differenti tra loro.

Nel diagramma successivo il facade pattern è realizzato attraverso la classe Facade, la quale permette di nascondere la complessità dell’operazione della classe Client, la quale chiamerà soltanto il metodo metodoUnico per realizzare la stessa operazione.



1. **Packages**

L’architettura del nostro sistema si basa su tre livelli (three-tier):

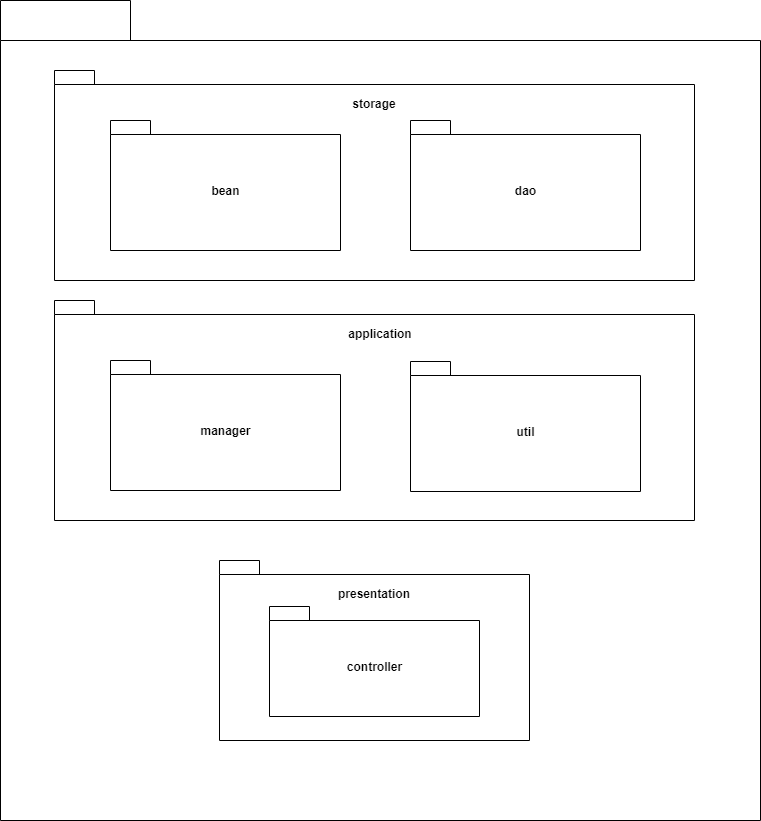
* + Presentation layer.
  + Application layer.
  + Storage layer.

Il sistema ingloba un insieme di package (storage, application, presentation) che contengono le classi Java adatte:

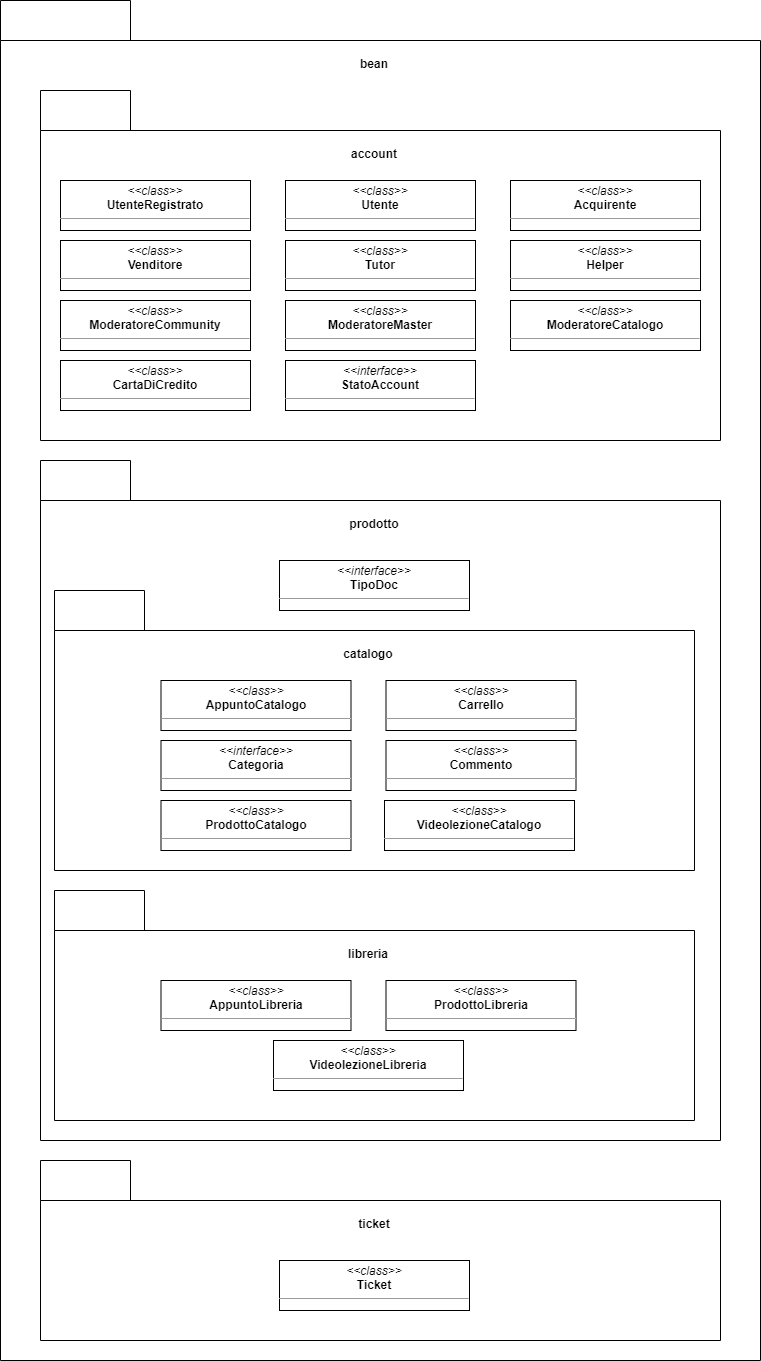
* + Nel package storage ci saranno i package bean e dao. Nel package bean ci saranno tutte le classi Java che rappresentano gli oggetti del sistema, mentre nel package dao, ci saranno le classi Java che permetteranno di effettuare le operazioni CRUD sul database.
  + Nel package presentation ci sarà a sua volta il package controller, dove ci saranno le classi JavaServlet adatte a gestire le richieste provenienti dai client.
  + Nel package application ci saranno i package manager e util. Nel package manager ci saranno le classi adatte che conterranno la vera e propria logica di business del sistema. Nel package util ci saranno le classi di utilità del sistema, come la classe per mandare l’e-mail, e le classi per poter effettuare controlli sui formati dei dati.

|  |  |
| --- | --- |
| Presentation layer | Rappresenta l’interfaccia del sistema con la quale l’utente può interagire inviando o visualizzando dati in input e che visualizzare e inviare dati in output. |
| Application layer | Si occupa di varie gestioni quali: - Gestione Account. - Gestione Appunto. - Gestione Videolezione. - Gestione Tutor. - Gestione Acquisto. - Gestione Ticket. - Gestione Commento. - Gestione Staff. |
| Storage layer | Ha il compito di memorizzare i dati sensibili del sistema, utilizzando un DBMS. Comprende i bean che rappresentano i vari oggetti presenti nel database, e i dao, ovvero oggetti con il quale effettuare operazioni CRUD sul database. |

* 1. **General vision about packages**

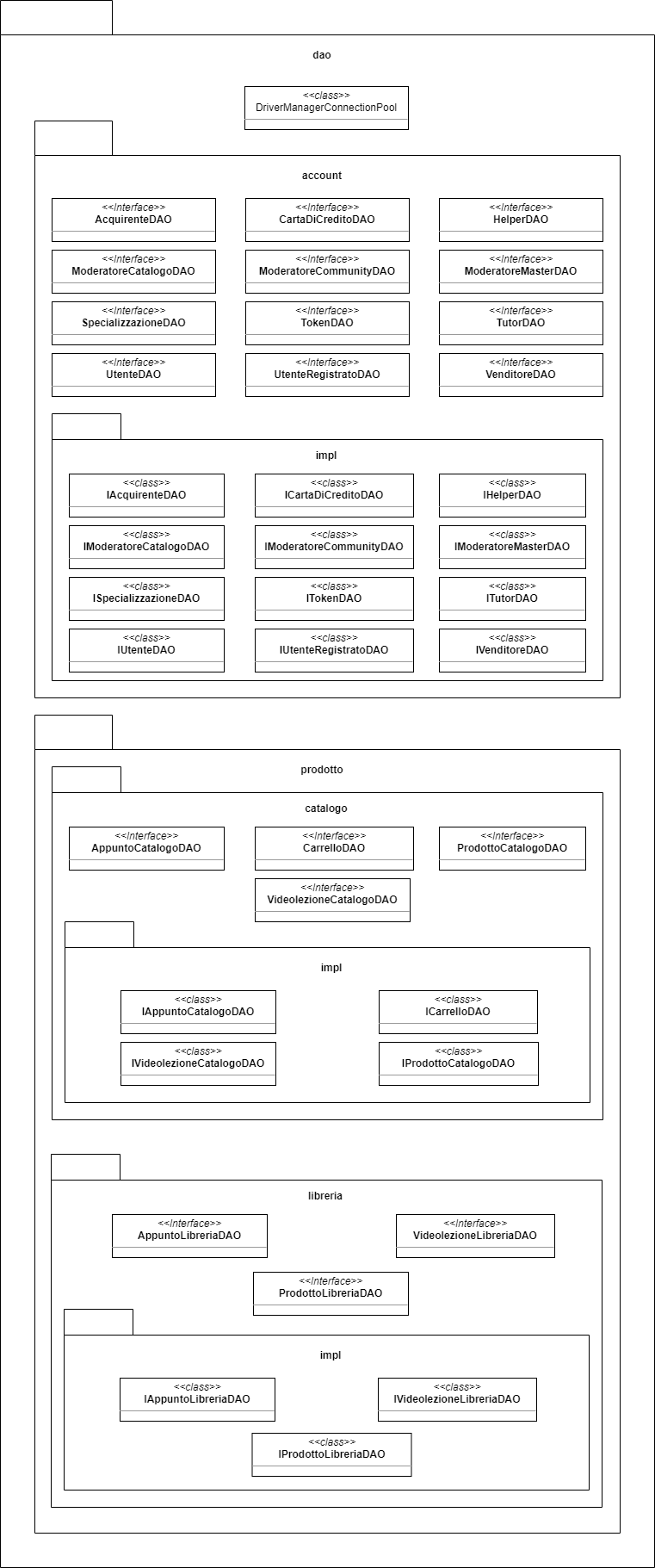
****

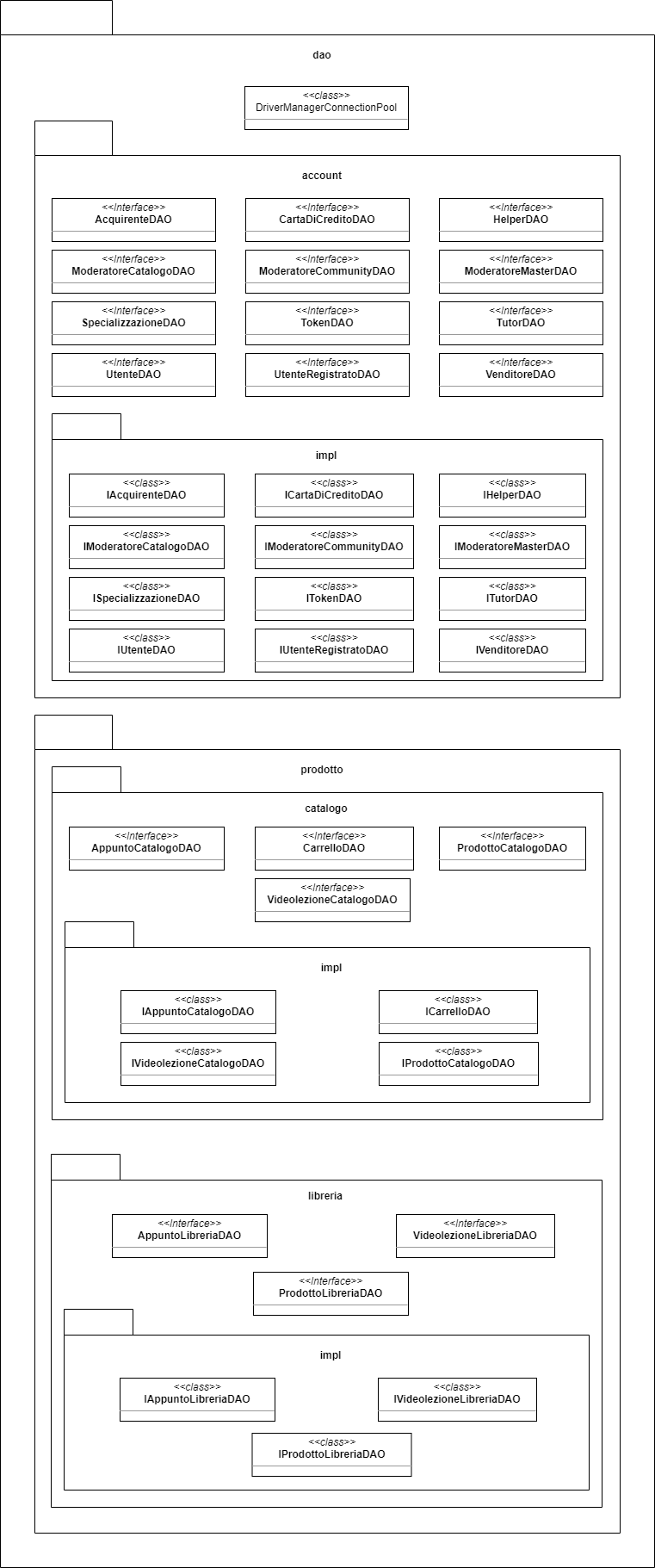
* 1. **Package bean**



|  |  |
| --- | --- |
| Classe | Descrizione |
| bean.account.Acquirente | Questa classe rappresenta l’acquirente. |
| bean.account.CartaDiCredito | Questa classe rappresenta la carta di credito. |
| bean.account.Helper | Questa classe rappresenta l’helper. |
| bean.account.ModeratoreCatalogo | Questa classe rappresenta il moderatore catalogo. |
| bean.account.ModeratoreCommunity | Questa classe rappresenta il moderatore community. |
| bean.account.ModeratoreMaster | Questa classe rappresenta il moderatore master. |
| bean.account.StatoAccount | Questa classe rappresenta l’enum utilizzato per identificare lo stato corrente degli account acquirente e venditore. |
| bean.account.Tutor | Questa classe rappresenta il tutor. |
| bean.account.Utente | Questa classe rappresenta l’utente. |
| bean.account.UtenteRegistrato | Questa classe rappresenta l’utente registrato. |
| bean.account.Venditore | Questa classe rappresenta il venditore. |
| bean.prodotto.TipoDoc | Questa classe rappresenta l’enum utilizzato per identificare il tipo di appunto. |
| bean.prodotto.catalogo.AppuntoCatalogo | Questa classe rappresenta l’appunto catalogo. |
| bean.prodotto.catalogo.Carrello | Questa classe rappresenta il carrello. |
| bean.prodotto.catalogo.Categoria | Questa classe rappresenta l’enum utilizzato per identificare la categoria del prodotto venduto. |
| bean.prodotto.catalogo.Commento | Questa classe rappresenta il commento. |
| bean.prodotto.catalogo.ProdottoCatalogo | Questa classe rappresenta il prodotto catalogo. |
| bean.prodotto.catalogo.VideolezioneCatalogo | Questa classe rappresenta la videolezione catalogo. |
| bean.prodotto.libreria.AppuntoLiberia | Questa classe rappresenta l’appunto libreria. |
| bean.prodotto.libreria.ProdottoLibreria | Questa classe rappresenta il prodotto libreria. |
| bean.prodotto.libreria.VideolezioneLibreria | Questa classe rappresenta la videolezione libreria. |
| bean.ticket.Ticket | Questa classe rappresenta il ticket |

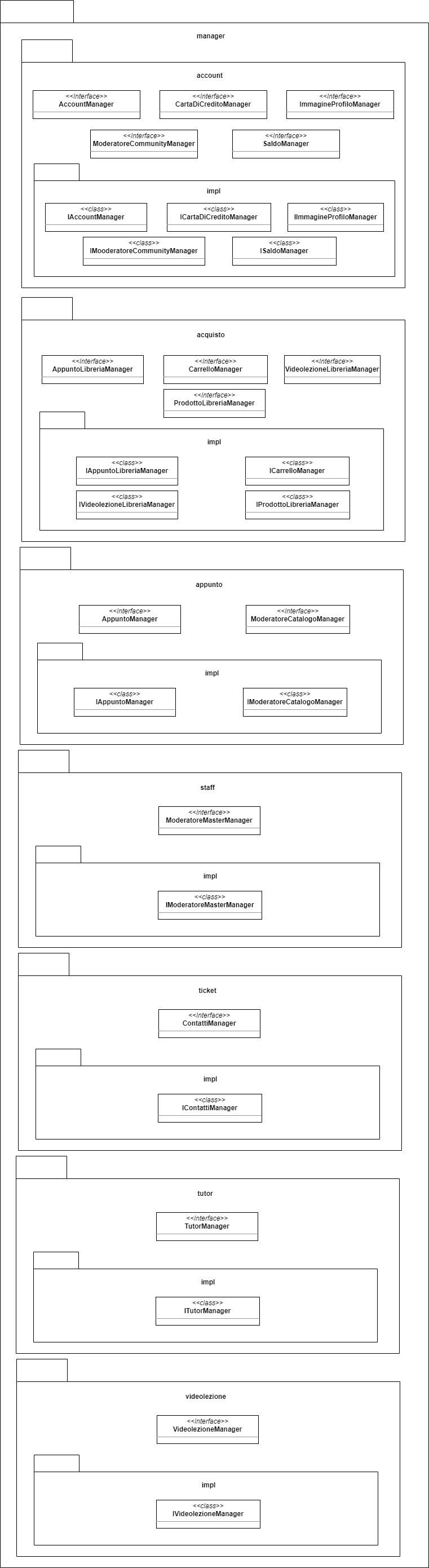
* 1. **Package** **dao**

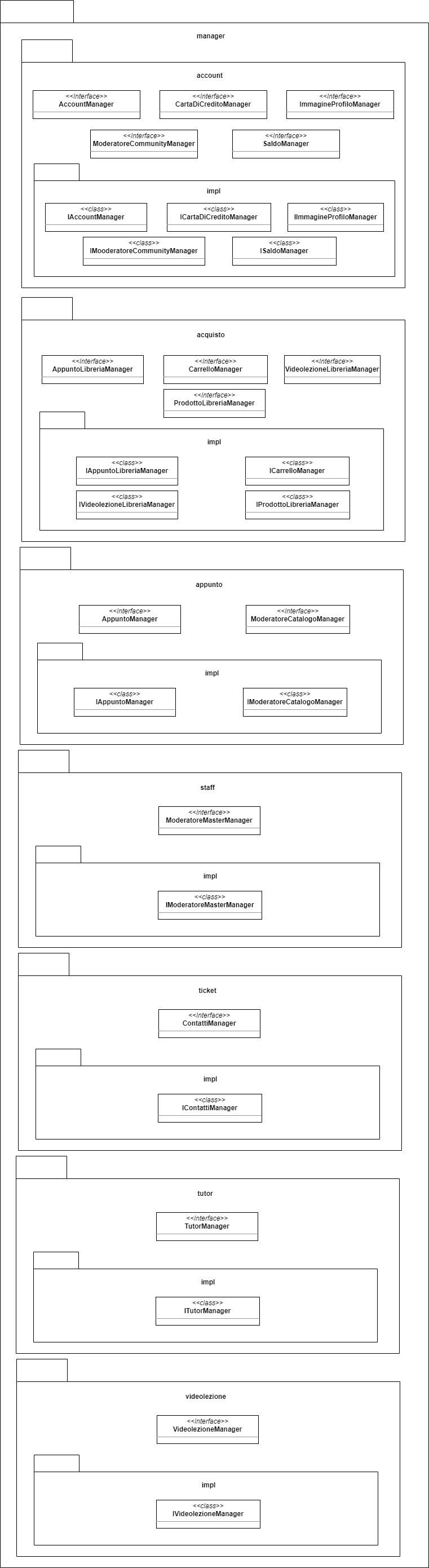


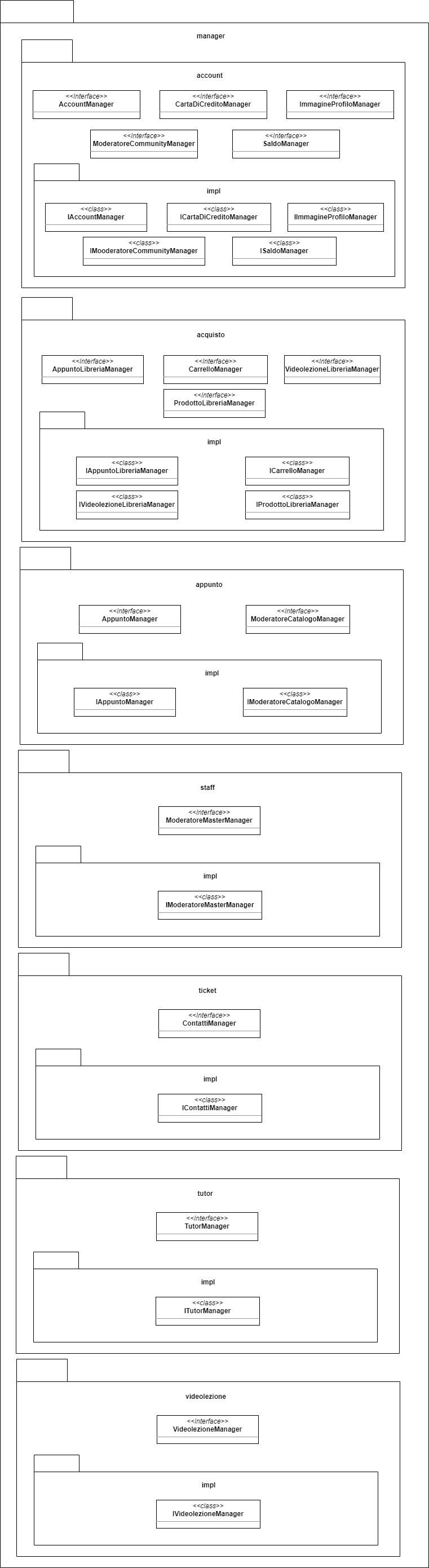


|  |  |
| --- | --- |
| Classe | Descrizione |
| dao.DriverManagerConnectionPool | Singleton che gestisce la pool di connessioni. |
| dao.account.AcquirenteDAO | Interfaccia che rappresenta i metodi CRUD sulla tabella Acquirente. |
| dao.account.CartaDiCreditoDAO | Interfaccia che rappresenta i metodi CRUD sulla tabella CartaDiCredito. |
| dao.account.HelperDAO | Interfaccia che rappresenta i metodi CRUD sulla tabella Helper. |
| dao.account.ModeratoreCatalogoDAO | Interfaccia che rappresenta i metodi CRUD sulla tabella ModeratoreCatalogo. |
| dao.account.ModeratoreCommunityDAO | Interfaccia che rappresenta i metodi CRUD sulla tabella ModeratoreCommunity. |
| dao.account.ModeratoreMasterDAO | Interfaccia che rappresenta i metodi CRUD sulla tabella ModeratoreMaster. |
| dao.account.SpecializzazioneDAO | Interfaccia che rappresenta i metodi CRUD sulla tabella Specializzazione e Possiede. |
| dao.account.TokenDAO | Interfaccia che rappresenta i metodi CRUD sulla tabella Token. |
| dao.account.TutorDAO | Interfaccia che rappresenta i metodi CRUD sulla tabella Tutor. |
| dao.account.UtenteDAO | Interfaccia che rappresenta i metodi CRUD sulla tabella Utente. |
| dao.account.UtenteRegistratoDAO | Interfaccia che rappresenta i metodi CRUD sulla tabella UtenteRegistrato. |
| dao.account.VenditoreDAO | Interfaccia che rappresenta i metodi CRUD sulla tabella Venditore. |
| dao.account.impl.IAcquirenteDAO | Implementazione dell’interfaccia AcquirenteDAO. |
| dao.account.impl.ICartaDiCreditoDAO | Implementazione dell’interfaccia CartaDiCreditoDAO. |
| dao.account.impl.IHelperDAO | Implementazione dell’interfaccia HelperDAO. |
| dao.account.impl.IModeratoreCatalogoDAO | Implementazione dell’interfaccia ModeratoreCatalogoDAO. |
| dao.account.impl.IModeratoreCommunityDAO | Implementazione dell’interfaccia ModeratoreCommunityDAO. |
| dao.account.impl.IModeratoreMasterDAO | Implementazione dell’interfaccia ModeratoreMasterDAO. |
| dao.account.impl.ISpecializzazioneDAO | Implementazione dell’interfaccia SpecializzazioneDAO. |
| dao.account.impl.ITokenDAO | Implementazione dell’interfaccia TokenDAO. |
| dao.account.impl.ITutorDAO | Implementazione dell’interfaccia TutorDAO. |
| dao.account.impl.IUtenteDAO | Implementazione dell’interfaccia UtenteDAO. |
| dao.account.impl.IUtenteRegistratoDAO | Implementazione dell’interfaccia UtenteRegistratoDAO. |
| dao.account.impl.IVenditoreDAO | Implementazione dell’interfaccia IVenditoreDAO. |
| dao.prodotto.catalogo.AppuntoCatalogoDAO | Interfaccia che rappresenta i metodi CRUD sulla tabella AppuntoCatalogo. |
| dao.prodotto.catalogo.CarrelloDAO | Interfaccia che rappresenta i metodi CRUD sulla tabella Carrello. |
| dao.prodotto.catalogo.ProdottoCatalogoDAO | Interfaccia che rappresenta i metodi CRUD sulla tabella ProdottoCatalogo. |
| dao.prodotto.catalogo.VideolezioneCatalogoDAO | Interfaccia che rappresenta i metodi CRUD sulla tabella VideolezioneCatalogo. |
| dao.prodotto.catalogo.impl IAppuntoCatalogoDAO | Implementazione dell’interfaccia AppuntoCatalogoDAO. |
| dao.prodotto.catalogo.impl ICarrelloDAO | Implementazione dell’interfaccia CarrelloDAO. |
| dao.prodotto.catalogo.impl IProdottoCatalogoDAO | Implementazione dell’interfaccia ProdottoCatalogoDAO. |
| dao.prodotto.catalogo.impl IVideolezioneCatalogoDAO | Implementazione dell’interfaccia VideolezioneCatalogoDAO. |
| dao.prodotto.libreria.AppuntoLibreriaDAO | Interfaccia che rappresenta i metodi CRUD sulla tabella AppuntoLibreria. |
| dao.prodotto.libreria.VideolezioneLibreriaDAO | Interfaccia che rappresenta i metodi CRUD sulla tabella VideolezioneLibreria. |
| dao.prodotto.libreria.ProdottoLibreriaDAO | Interfaccia che rappresenta i metodi CRUD sulla tabella ProdottoLibreria. |
| dao.prodotto.libreria.impl IAppuntoLibreriaDAO | Implementazione dell’interfaccia AppuntoLibreriaDAO. |
| dao.prodotto.libreria.impl IVideolezioneLibreriaDAO | Implementazione dell’interfaccia VideolezioneLibreriaDAO. |
| dao.prodotto.libreria.impl IProdottoLibreriaDAO | Implementazione dell’interfaccia ProdottoLibreriaDAO. |

* 1. **Package manager**





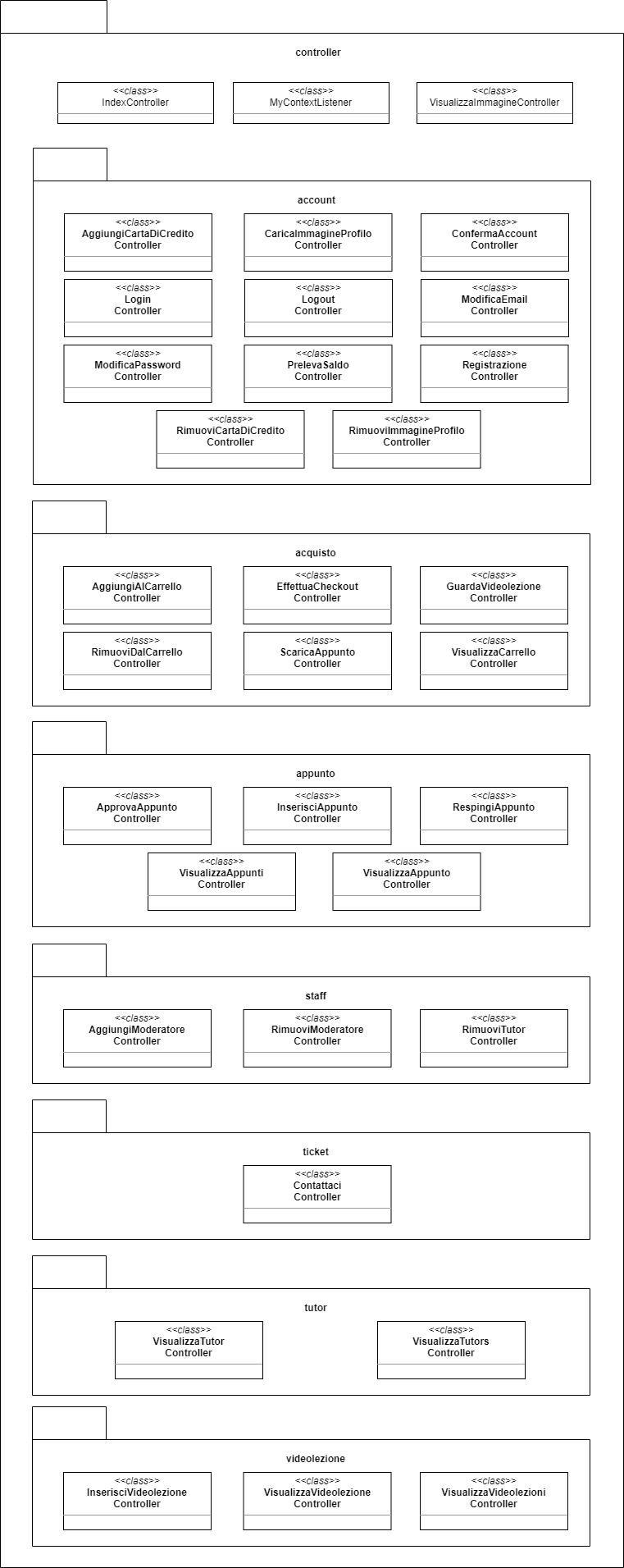


|  |  |
| --- | --- |
| Classe | Descrizione |
| manager.account.AccountManager | Manager che gestisce le operazioni di check e-mail, registrazione, conferma account, login, modifica e-mail/password e l’eliminazione |
| manager.account.CartaDiCreditoManager | Manager che gestisce le operazioni di aggiunta/rimozione e restituzione di carta di credito. |
| manager.account.ImmagineProfiloManager | Manager che gestisce le operazioni di upload e rimozione immagine profilo. |
| manager.account.ModeratoreCommunityManager | Manager che gestisce le operazioni ban degli utenti (acquirente, venditore). |
| manager.account.SaldoManager | Manager che gestisce la visualizzazione, aggiornamento e prelievo del saldo del venditore. |
| manager.account.impl IAccountManager | Implementazione dell’interfaccia AccountManager. |
| manager.account.impl ICartaDiCreditoManager | Implementazione dell’interfaccia CartaDiCreditoManager. |
| manager.account.impl IImmagineProfiloManager | Implementazione dell’interfaccia ImmagineProfiloManager. |
| manager.account.impl IModeratoreCommunityManager | Implementazione dell’interfaccia ModeratoreCommunityManager. |
| manager.account.impl ISaldoManager | Implementazione dell’interfaccia SaldoManager. |
| manager.acquisto.AppuntoLibreriaManager | Manager che gestisce la restituzione degli appunti acquistati. |
| manager.acquisto.CarrelloManager | Manager che gestisce la restituzione del carrello, l’aggiunta/rimozione di un prodotto da esso, e il checkout. |
| manager.acquisto.VideolezioneLibreriaManager | Manager che gestisce la restituzione delle videolezioni acquistate. |
| manager.acquisto.ProdottoLibreriaManager | Manager che gestisce la restituzione dei prodotti acquistati. |
| manager.acquisto.impl IAppuntoLibreriaManager | Implementazione dell’interfaccia AppuntoLibreriaManager. |
| manager.acquisto.impl ICarrelloManager | Implementazione dell’interfaccia CarrelloManager. |
| manager.acquisto.impl. IVideolezioneLibreriaManager | Implementazione dell’interfaccia VideolezioneLibreriaManager. |
| manager.acquisto.IProdottoLibreriaManager | Implementazione dell’interfaccia ProdottoLibreriaManager. |
| manager.appunto.AppuntoManager | Manager che gestisce la restituzione e assegnazione per l’approvazione di appunti. |
| manager.appunto.ModeratoreCatalogoMaster | Manager che gestisce approvazione/respinta di appunti. |
| manager.appunto.impl. IAppuntoManager | Implementazione dell’interfaccia AppuntoManager. |
| manager.appunto.impl. IModeratoreCatalogoMaster | Implementazione dell’interfaccia ModeratoreCatalogoMaster. |
| manager.staff.ModeratoreMasterManager | Manager che gestisce la registrazione/rimozione di tutor e moderatori. |
| manager.staff.impl. IModeratoreMasterManager | Implementazione dell’interfaccia ModeratoreMasterManager. |
| manager.ticket.ContattiManager | Manager che si occupa di inviare sia il riepilogo e la notifica di contatto da parte di un utente. |
| manager.ticket.impl.IContattiManager | Implementazione dell’interfaccia ContattiManager |
| manager.tutor.TutorManager | Manager che si occupa di restituzione di Tutor. |
| manager.tutor.impl.ITutorManager | Implementazione dell’interfaccia TutorManager |
| manager.videolezione.VideolezioneManager | Manager che si occupa di restituzione e salvataggio di VideolezioneCatalogo. |
| manager.videolezione.impl.IVideolezioneManager | Implementazione dell’interfaccia VideolezioneManager. |

* 1. **Package** **controller**

Immagine che contiene tavolo

Descrizione generata automaticamente



|  |  |
| --- | --- |
| Classe | Descrizione |
| controller.IndexController | Controller che gestisce il redirect alla pagina personale appropriata per l’utente loggato al sistema. |
| controller.MyContextListener | Controller che gestisce la connessione al database e la rimozione degli account non verificati con frequenza giornaliera. |
| controller.VisualizzaImmagineController | Controller che gestisce la visualizzazione delle immagini presenti nel database. |
| controller.account.AggiungiCartaDiCredito Controller | Controller che gestisce l’aggiunta di una carta di credito. |
| controller.account.CaricaImmagineProfilo Controller | Controller che gestisce il caricamento dell’immagine profilo. |
| controller.account.ConfermaAccountController | Controller che gestisce la conferma dell’account tramite token. |
| controller.account.LoginController | Controller che gestisce il login sul sistema. |
| controller.account.LogoutController | Controller che gestisce il logout dal sistema. |
| controller.account.ModificaEmailController | Controller che gestisce la modifica dell’e-mail del proprio account. |
| controller.account.ModificaPasswordController | Controller che gestisce la modifica della password del proprio account. |
| controller.account.PrelevaSaldoController | Controller che gestisce le richieste di prelievo del saldo del proprio account. |
| controller.account.RegistrazioneController | Controller che gestisce la registrazione sul sistema. |
| controller.account.RimuoviCartaDiCredito Controller | Controller che gestisce la rimozione della carta di credito. |
| controller.account.RimuoviImmagineProfilo Controller | Controller che gestisce la rimozione dell’immagine profilo. |
| controller.acquisto.AggiungiAlCarrelloController | Controller che gestisce l’aggiunta di un nuovo prodotto al carrello. |
| controller.acquisto.EffettuaCheckoutController | Controller che gestisce il checkout del carrello. |
| controller.acquisto.GuardaVideolezioneController | Controller che gestisce il redirect alla pagina protetta delle videolezioni acquistate. |
| controller.acquisto.RimuoviDalCarrelloController | Controller che gestisce la rimozione di un prodotto dal carrello. |
| controller.acquisto.ScaricaAppuntoController | Controller che gestisce il download degli appunti acquistati. |
| controller.acquisto.VisualizzaCarrelloController | Controller che gestisce la visualizzazione del riepilogo carrello. |
| controller.appunto.ApprovaAppuntoController | Controller che gestisce l’approvazione degli appunti. |
| controller.appunto.InserisciAppuntoController | Controller che gestisce l’inserimento di un appunto sul catalogo. |
| controller.appunto.RespingiAppuntoController | Controller che gestisce la respinta degli appunti. |
| controller.appunto.VisualizzaAppuntiController | Controller che gestisce la visualizzazione degli appunti filtrati sul catalogo. |
| controller.appunto.VisualizzaAppuntoController | Controller che gestisce la visualizzazione del singolo appunto sul catalogo. |
| controller.staff.AggiungiModeratoreController | Controller che gestisce l’aggiunta di un moderatore sul sistema. |
| controller.staff.RimuoviModeratoreController | Controller che gestisce la rimozione di un moderatore dal sistema. |
| controller.staff.RimuoviTutorController | Controller che gestisce la rimozione di un tutor dal sistema. |
| controller.ticket.ContattaciController | Controller che gestisce le richieste di contatto. |
| controll.tutor.VisualizzaTutorController | Controller che gestisce la visualizzazione dei tutor filtrati sulla rubrica. |
| controll.tutor.VisualizzaTutorsController | Controller che gestisce la visualizzazione del singolo tutor sulla rubrica. |
| controller.videolezione.InserisciVideolezione Controller | Controller che gestisce l’inserimento di una videolezione sul catalogo. |
| controller.videolezione.VisualizzaVideolezione Controller | Controller che gestisce la visualizzazione delle videolezioni filtrate sul catalogo. |
| controller.videolezione.VisualizzaVideolezioni Controller | Controller che gestisce la visualizzazione della singola videolezione sul catalogo. |

1. **Class Interfaces**
   1. **AccountManager**

|  |  |
| --- | --- |
| AccountManager | |
| checkEmail | **Context:** AccountManager::checkEmail(e-mail: String) : boolean  **Pre:** email != null  **Post:** result = UtenteRegistrato -> exist(u | u.email == email) |
| registra | **Context:** AccountManager::registra(acquirente: Acquirente) : String  **Pre:** acquirente != null AND acquirente.email != null AND acquirente.password != null AND acquirente.nome != null AND acquirente.cognome != null acquirente.dataNascita != null AND acquirenti -> forAll(a | a.email != acquirente.email)  **Post:** Acquirente -> include(acquirente) AND acquirente.stato == StatoAccount.DAVERIFICARE AND result = Token -> select(t.token | t.email == acquirente.email) |
| registra | **Context:** AccountManager::registra(venditore: Venditore) : String  **Pre:** venditore != null AND venditore.email != null AND venditore.password != null AND venditore.nome != null AND venditore.cognome != null venditore.dataNascita != null AND Venditore -> forAll(v | v.email != venditore.email)  **Post:** Venditore -> include(venditore) AND venditore.stato == StatoAccount.DAVERIFICARE AND result = Token -> select(t.token | t.email == venditore.email) |
| login | **Context:** AccountManager::login(e-mail: String, password: String)  **Pre:** email != null AND password != null  **Post:** result = UtenteRegistrato -> select (u | u.email == email AND u.password == password) |
| modificaEmail | **Context:** AccountManager::modificaEmail(utenteRegistrato : UtenteRegistrato, email : String) : boolean  **Pre:** utenteRegistrato!= null AND utenteRegistrato.email != null AND utenteRegistrato.password != null AND utenteRegistrato.nome != null AND utenteRegistrato.cognome != null AND email != null  **Post:** result = UtenteRegistrato -> exist(u | u.email == email AND u.password == @pre.utenteRegistrato.password AND u.nome == @pre.utenteRegistarto.nome AND u.cognome == @pre.utenteRegistrato.cognome) |
| modificaPassword | **Context:** AccountManager::modificaPassword(utenteRegistrato : UtenteRegistrato, email : String) : boolean  **Pre:** utenteRegistrato!= null AND utenteRegistrato.email != null AND utenteRegistrato.password != null AND utenteRegistrato.nome != null AND utenteRegistrato.cognome != null AND password != null  **Post:** result = UtenteRegistrato -> exist(u | u.email == @pre.utenteRegistrato.email AND u.password == password AND u.nome == @pre.utenteRegistarto.nome AND u.cognome == @pre.utenteRegistrato.cognome) |
| eliminaAccount | **Context:** AccountManager::eliminaAccount(e-mail: String) : void  **Pre:** email != null  **Post:** !(UtenteRegistrato -> exist(u | u.email == email)) |
| eliminaAccountNonVerificati | **Context:** AccountManager::eliminaAccountNonVerificati() : void **Post**: !(Acquirente -> exist(a | a.stato == StatoAccount.DAVERIFICARE) AND !Venditore -> exist(v | v.stato == StatoAccount.DAVERIFICARE)) |

* 1. **CartaDiCreditoManager**

|  |  |
| --- | --- |
| CartaDiCreditoManager | |
| checkNumeroCartaDiCredito | **Context:** CartaDiCreditoManager:: checkNumeroCartaDiCredito(numeroCarta: String) : boolean  **Post:** result = CartaDiCredito -> exist(c | c.numeroCarta == numeroCarta) |
| aggiungiCartaDiCredito | **Context:** CartaDiCreditoManager:: aggiungiCartaDiCredito(cartaDiCredito: CartaDiCredito, email: String)  **Pre:** cartaDiCredito != null AND cartaDiCredito.nomeIntestatario != null AND cartaDiCredito.numeroCarta != null AND cartaDiCredito.scadenza != null AND CartaDiCredito -> forAll(c | c.numeroCarta != cartaDiCredito.numeroCarta AND Acquirente -> exist(a | a.email == email))  **Post:** CartaDiCredito -> include(cartaDiCredito) |
| rimuoviCartaDiCredito | **Context:** CartaDiCreditoManager:: rimuoviCartaDiCredito(numeroCarta: String, email: String)  **Pre:** numeroCarta != null AND email != null AND CartaDiCredito -> exist(c | c.numeroCarta == numeroCarta AND c.email == email)  **Post:** !CartaDiCredito -> exist(c | c.numeroCarta == numeroCarta) |
| checkCartaDiCredito | **Context:** CartaDiCreditoManager: checkCartaDiCredito(numeroCarta: String, email: String)  **Pre:** email != null AND numeroCarta != null **Post:** result = CartaDiCredito(c | c.numeroCarta == numeroCarta && c.email == email) |

* 1. **ImmagineProfiloManager**

|  |  |
| --- | --- |
| ImmagineProfiloManager | |
| caricaImmagineProfilo | **Context:** ImmagineProfiloManager::caricaImmagineProfilo(email: String , imgProfilo: InputStream)  **Pre:** imgProfilo != null AND Utente -> exist(u | u.email == email)  **Post:** utente.imgProfilo == imgProfilo |
| rimuoviImmagineProfilo | **Context:** ImmagineProfiloManager:: rimuoviImmagineProfilo(email: String)  **Pre:** Utente -> exist(u | u.email == email)  **Post:** utente.imgProfilo == null |
| ottieniImmagineProfilo | **Context:** ImmagineProfiloManager:: ottieniImmagineProfilo(email: String): byte[] **Pre**: Utente -> exist(u | u.email == email) **Post**: result = utente.imgProfilo |

* 1. **SaldoManager**

|  |  |
| --- | --- |
| SaldoManager | |
| visualizzaSaldo | **Context:** SaldoManager::visualizzaSaldo(email: String): double  **Pre:** Venditore -> exist(v | v.email == email) **Post:** result = v.saldo |
| modificaSaldo | **Context:** SaldoManager::modificaSaldo(email: String, saldo: double)  **Pre:** Venditore -> exist(v | v.email == email)  **Post:** result: v.saldo = @pre.v.saldo + saldo |

* 1. **AppuntoLibreriaManager**

|  |  |
| --- | --- |
| AppuntoLibreriaManager | |
| ottieniProdottoDi | **Context:** AppuntoLibreriaManager::ottieniProdottoDi(id: int, email: String): AppuntoLibreria  **Pre:** Acquirente -> exist(a | a.email == email) AND id != null  **Post:** result = AppuntoLibreria -> select(a | a.id == id AND a.email == email) |

* 1. **CarrelloManager**

|  |  |
| --- | --- |
| CarrelloManager | |
| ottieniCarrelloDi | **Context:** CarrelloManager::ottieniCarrelloDi(emailAcquirente: Sring): Carrello  **Pre:** Acquirente -> exist(a | a.email == emailAcquirente)  **Post:** Carrello -> select(c | c.email == emailAcquirente) |
| checkProdotto | **Context:** CarrelloManager::checkProdotto(idProdotto: int, emailAcquirente: String): boolean  **Pre:** Acquirente -> exist(a | a.email == emailAcquirente)  **Post:** Contine -> exist(c | c.idProdotto = idProdotto AND c.emailAcquirente == (Carrello -> select(car.id | car.email == emailAcquirente)) |
| aggiungiProdottoAlCarrello | **Context:** CarrelloManager:: aggiungiProdottoAlCarrello(idProdotto: int , emailAcquirente: String)  **Pre:** Acquirente -> exist(a | a.email == emailAcquirente) AND ProdottoCatalogo -> exist(p | p.id == idProdotto)  **Post:** Contiene -> include(c | c.idPrdotto == idProdotto AND c.idCarrello == (Carrello -> select(car.id | car.email == emailAcquirente)) |
| rimuoviProdottoDalCarrello | **Context:** CarrelloManager::rimuoviDalCarrello(idProdotto, emailAcquirente)  **Pre:** Acquirente -> exist(a | a.email == emailAcquirente) AND Contiene -> include(c | c.idProdotto == idProdotto AND c.email == (Carrello -> select(c.id | c.email == emailAcquirente))  **Post:** !(Contiene -> include(c | c.idProdotto == idProdotto AND c.email == (Carrello -> select(car.id | car.email == emailAcquirente))) |
| effettuaCheckoutCarrelloDi | **Context:** CarrelloManager:: effettuaCheckoutCarrelloDi(emailAcquirente: String) **Pre:** Acquirente -> exist(a | a.email == emailAcquirente) AND Contiene -> exist(c | c.idCarrello == select(c.id | c.email == emailAcquirente))  **Post:** @Pre.Contiene -> forAll(c | c.idCarrello == (Carrello -> select(car.id | car.email == emailAcquirente)) ProdottoLibreria -> exist(p | p.id = c.id AND p.emailAcquirente = emailAcquirente)) AND (!Contiene -> exist(c | c.idCarrello == (Carrello -> select(car.id | car.email == emailAcquirente))) |
| isVuoto | **Context:** CarrelloManager::isVuoto(emailAcquirente: String): boolean  **Pre:** Carrello -> exist(c | c.emailAcquirente == emailAcquirente)  **Post:** result = Contiene -> exist(con | con.idCarrello = (Carrello -> select (c | c.emailAcquirente == emailAcquirente)) |

* 1. **ProdottoLibreriaManager**

|  |  |
| --- | --- |
| ProdottoLibreriaManager | |
| isAcquistato | **Context:** ProdottoLibreriaManager::isAcquistato(id: int, email: String): boolean  **Post:** result = ProdottoLibreria -> exist(p | p.id == id AND p.email == email) |
| ottieniProdottiAcquistatiDi | **Context:** ProdottoLibreriaManager::ottieniProdottiAcquistatiDi( email: String): List <ProdottoLibreria>  **Pre:** Acquirente -> exist(a | a.email == email)  **Post:** result = ProdottoLibreria -> select(p | p.emailAcquirente == emailAcquirente) |

* 1. **VideolezioneLibreriaManager**

|  |  |
| --- | --- |
| VideolezioneLibreriaManager | |
| ottieniProdottoDi | **Context:** VideolezioneLibreriaManager::ottieniProdottoDi( id: int, email: String): VideolezioneLibreria  **Pre:** Acquirente -> exist(a | a.email == email)  **Post:** result = VideolezioneLibreria -> exist(v | v.id == id AND v.email == email) |

* 1. **AppuntoManager**

|  |  |
| --- | --- |
| AppuntoManager | |
| ottieniAppunto | **Context:** AppuntoManager::ottieniAppunto(id: int): AppuntoCatalogo  **Pre:** id != null  **Post:** result = AppuntoCatalogo -> select(a | a.id == id) |
| ottieniAppunti | **Context:** AppuntoManager: ottieniAppunti(): List <AppuntoCatalogo>  **Post:** result = AppuntoCatalogo -> asSet |
| ottieniAppuntiApprovati | **Context:** AppuntoManager::ottieniAppuntiApprovati(): List <AppuntoCatalogo>  **Post:** result = AppuntoCatalogo -> select(a | a.stato == StatoAppunto.APPROVATO) |
| ottieniAppuntiApprovatiFiltrati | **Context:** AppuntoManager::ottieniAppuntiApprovatiFiltrati( categorie: ArrayList <Categoria>, tipoDocs: ArrayList <TipoDoc>, titolo: String)  **Pre:** categoria != null && tipoDocs != null  **Post:** result = AppuntoCatalogo -> select(a | a.titolo LIKE %titolo% AND categoria -> include(a.categoria) AND tipoDocs -> include (a.tipoDoc) AND a.stato == StatoAppunto.APPROVATO) |
| ottieniAppuntiDi | **Context:** AppuntoManager::ottieniAppuntiDi(String email): List <AppuntoCatalogo>  **Pre:** Venditore -> exist(v | v.email == email)  **Post:** result = AppuntoCatalogo -> select(a | a.emailVenditore == email) |
| isAppunto | **Context:** AppuntoManager::isAppunto(id: int): boolean  **Pre:** id != null  **Post:** result = AppuntoCatalogo -> exist(a | a.id == id) |
| assegnaRevisione | **Context:** AppuntoManager::assegnaRevisione(appunto: AppuntoCatalogo)  **Pre:** appunto != null AND appunto.titolo != null AND appunto.descrizione != null AND appunto.prezzo != null AND appunto.categoria != null AND appunto.tipoDoc != null AND appunto.file != null AND Venditore -> exist(v | v.email == appunto.emailVenditore)  **Post:** Revisione -> exist(r | r.idAppunto == appunto.id AND r.emailmoderatore == (ModeratoreCatalogo -> select(m | m.revisioniAssengate.min((ModeratoreCatalogo -> asSet).revisioniAssegnate))) |

* 1. **ModeratoreCatalogoManager**

|  |  |
| --- | --- |
| ModeratoreCatalogoManager | |
| approvaAppunto | **Context:** ModeratoreCatalogoManager::approvaAppunto(id: int)  **Pre:** id != null  **Post:** AppuntoCatalogo -> exist(a | a.id == id AND a.stato == StatoAppunto.APPROVATO) |
| respingiAppunto | **Context:** ModeratoreCatalogoManager::RespingiAppunto(id: int)  **Pre:** id != null  **Post:** !(AppuntoCatalogo -> exist(a | a.id == id)) |
| ottieniAppuntiDaRevisionare | **Context:** ModeratoreCatalogoManager:: ottieniAppuntiDaRevisionare(email: String): List <AppuntoCatalogo>  **Pre:** ModeratoreCatalogo -> exist(m | m.email == email)  **Post:** result = AppuntoCatalogo -> select(a | (Revisione -> select(r.id | r.emailModeratore == email)) -> include(r | r.id == a.id)) |

* 1. **ModeratoreMasterManager**

|  |  |
| --- | --- |
| ModeratoreCatalogoManager | |
| registra | **Context:** ModeratoreMasterManager::registra(tutor: Tutor)  **Pre:** tutor != null AND tutor.nome != null AND tutor.cognome != null AND tutor.email != null AND tutor.cognome != null AND tutor.curriculum != null AND tuttor.specializzazioni.size() > 1 AND !(Tutor -> exist(t | t.email == email))  **Post:** Tutor -> exist(t | t.email == email) |
| registra | **Context:** ModeratoreMasterManager::registra(helper: Helper)  **Pre:** helper!= null AND helper.nome != null AND helper.cognome != null AND helper.email != null AND helper.cognome != null AND !(Helper -> exist(h | h.email == email)  **Post:** Helper -> exist(h | h.email == email) |
| registra | **Context:** ModeratoreMasterManager::registra(moderatore: ModeratoreCatalogo)  **Pre:** moderatore!= null AND moderatore.nome != null AND moderatore.cognome != null AND moderatore.email != null AND moderatore.cognome != null AND !(ModeratoreCatalogo -> exist(m | m.email == email)  **Post:** ModeratoreCatalogo -> exist(m | m.email == email) |
| registra | **Context:** ModeratoreMasterManager::registra(moderatore: ModeratoreCommunity)  **Pre:** moderatore!= null AND moderatore.nome != null AND moderatore.cognome != null AND moderatore.email != null AND moderatore.cognome != null AND !(ModeratoreCommunity -> exist(m | m.email == email)  **Post:** ModeratoreCommunity -> exist(m | m.email == email) |
| registra | **Context:** ModeratoreMasterManager::registra(moderatore: ModeratoreMaster)  **Pre:** moderatore!= null AND moderatore.nome != null AND moderatore.cognome != null AND moderatore.email != null AND moderatore.cognome != null AND !( ModeratoreMaster -> exist(m | m.email == email)  **Post:** ModeratoreMaster -> exist(m | m.email == email) |
| rimuoviModeratore | **Context:** ModeratoreMasterManager::rimuoviModeratore(email: String): boolean  **Pre: i**f <ModeratoreCatalogo -> exist(m | m.email == email)> ModeratoreCatalogo -> asSet.size() > 1 endif   **Post:** !(ModeratoreCatalogo -> exist(m | m.email == email)) AND !(ModeratoreCommunity -> exist(m | m.email == email)) AND !(ModeratoreMaster -> exist(m | m.email == email)) AND !(Helper -> exist(h | h.email == email)) |
| rimuoviTutor | **Context:** ModeratoreMasterManagaer::rimuoviTutor(email: String)  **Pre:** email != null  **Post:** !(Tutor -> exist(t | t.email == email) |
| ottieniModeratori | **Context:** ModeratoreMasterManager::ottieniModeratori(): List <UtenteRegistrato>  **Post:** result = (Helper -> asSet) -> union -> (ModeratoreCatalogo -> asSet) -> union -> (ModeratoreCommunity -> asSet) -> union -> (ModeratoreMaster -> asSet) |

* 1. **TutorManager**

|  |  |
| --- | --- |
| TutorManager | |
| ottieniTutor | **Context:** TutorManager::ottieniTutor(email: String): Tutor  **Pre:** email != null  **Post** result = Tutor -> select(t | t.email == email) |
| ottieniTutorsFiltrati | **Context** TutorManager::ottieniTutorsFiltrati(categorie: ArrayList <Categoria>, cognome: String): List <Tutor>  **Pre:** categoria != null AND cognome != null  **Post:** Tutor -> select(t | categoira.include(t.categoria) AND t.cognome LIKE %cognome%) |
| ottieniTutti | **Context:** TutorManager::ottieniTutti(): List <Tutor>  **Post:** Tutor -> asSet |

* 1. **VideolezioneManager**

|  |  |
| --- | --- |
| VideolezioneManager | |
| ottieniVideolezione | **Context:** VideolezioneManager::ottieniVideolezione(id: int): VideolezioneCatalogo  **Pre:** id != null  **Post:** result = VideolezioneCatalogo -> select(v | v.id == id) |
| ottieniVideolezioni | **Context:** VideolezioneManager::ottieniVideolezioni(): List <VideolezioneCatalogo>  **Post:** result = VideolezioneCatalogo -> asSet |
| ottieniVideolezioniFiltrate | **Context:** VideolezioneManagaer::ottieniVideolezioniFiltrate( categorie: ArrayList <Categoria>, durata: double, titolo: String): List <VideolezioneCatalogo>  **Pre:** categorie != null AND titolo != null AND durata != null  **Post:** VideolezioneCatalogo -> select(v | categorie.include(v.categoria) (if <durata > 0> AND v.durata > durata endif) AND v.titolo LIKE %titolo%) |
| ottieniVideolezioniDi | **Context:** VideolezioneManager::ottieniVideolezioniDi(email: String): List <VideolezioneCatalogo>  **Pre:** email != null  **Post:** VideolezioneCatalogo -> select(v | v.emailTutor == email) |
| salvaVideolezione | **Context:** VideolezioneManager::salvaVideolezione(videolezione: VideolezioneCatalogo)  **Pre:** videlezione != null AND videolezione.titolo != null AND videolezione.descrizione != null AND videolezione.emailTutor != null AND videolezione.prezzo != null AND videolezione.durata != null AND videolezione.link != null  **Post:** Videolezione -> include(videolezione) |