UNIVERSITÀ DEGLI STUDI DI SALERNO

Corso di Laurea in Informatica, prof. A. De Lucia,
a.a 2021/2022
Progetto di Ingegneria del Software

Object Design Document

Partecipanti	**Matricola**
Marta Coiro	0512108154
Katia Buonocore	0512106528
Rita Cuccaro	0512109495

Revision History

Data	Versione	Descrizione	Autore
16/11/2021	1.0	Prima stesura del documento (Problem Statement)	Membri del team
30/11/2021	1.0	Requirement Analysis Document	Membri del team
06/12/2021	1.0	System Design Document	Membri del team
20/12/2021	1.0	Gestione Dati Persistenti_MusicConsole	Membri del team
27/12/2021	1.0	Object Design Document	Membri del team
15/01/2022	1.0	Test Plan	Membri del team
28/01/2022	1.0	Test Execution Report	Membri del team
18/02/2022	1.0	Test Summary Report	Membri del team

Indice

1. INTRODUZIONE

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 dell'implementazione. Il seguente documento ha lo scopo di produrre un modello capace di integrare in modo coerente e preciso tutte le diverse funzionalità individuate nelle fasi precedenti. In particolare, in questo documento si vanno a descrivere i trade-offs generali realizzati dagli sviluppatori, le linee guida sulla documentazione delle interfacce e le convenzioni di codifica, una panoramica dei design pattern utilizzati dalle classi che compongono il sistema e dei packages che le contengono e una descrizione delle interfacce dei sottosistemi individuati.

1.1 Object Design Trade-offs

Seguono i trade-offs trovati nello sviluppo del sistema.

COMPRENSIBILITA' 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 USABILITA':

- 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 a quanti più utenti possibili.

SICUREZZA vs EFFICIENZA:

- La sicurezza, come descritto nei requisiti non funzionali del Requirement Analysis Document, rappresenta uno degli aspetti chiave della piattaforma. Tuttavia, dati i tempi di sviluppo molto limitati, ci limiteremo ad implementare sistemi di sicurezza basati sulla crittografia della password degli utenti.

RESPONSE TIME vs HARDWARE:

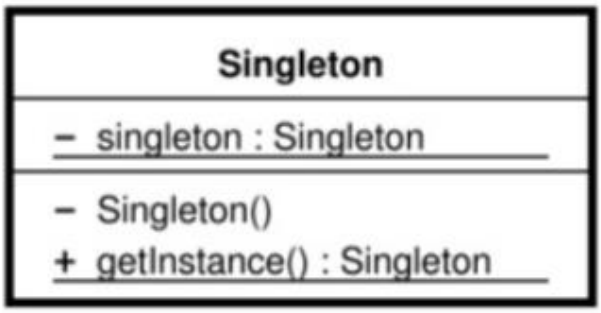
- Il sistema garantisce una certa reattività alle richieste, e quindi è in grado di poter comunque offrire una contemporaneità di servizi agli utenti. Ovviamente questa caratteristica sarà limitata dall'hardware del sistema.

PRESTAZIONI vs COSTI:

- Il sistema prevede l'utilizzo di fogli di stile semplici e open per mantenere prestazioni elevate, e cioè non s'intende utilizzare librerie grafiche esterne a pagamento, essendo il progetto sprovvisto di budget.

1.1.2 DESIGN PATTERN

1.1.2.1 Singleton



Il design pattern del singleton permette di restringere il numero di istanze di una classe ad una singola istanza.

Funzionalità

Il pattern del singleton permette la riduzione della memoria utilizzata per le classi che non hanno bisogno di mantenere uno stato o che possono usufruire di uno stato condiviso tra i suoi utilizzatori, in quanto esiste una singola istanza per ogni classe che adotta l'utilizzo del pattern. Il design pattern del singleton viene ampiamente utilizzato per l'accesso centralizzato a funzionalità e componenti in un'unica entità condivisa dai suoi utilizzatori.

Utilizzo

Il sistema utilizza il design pattern del singleton per la maggior parte delle classi che non mantengono uno stato o con stato condiviso, che quindi possono essere accedute concorrentemente, come classi d'utilità, classi realizzanti i servizi che implementano le diverse funzionalità individuate (funzionalità riguardanti gli account utenti).

Nel nostro caso utilizziamo il Singleton Pattern per collegare le classi al nostro DB tramite la classe:
DriverManagerConnectionPool.java.

1.1.2.2 Observer Design Pattern(DAO)

Il design pattern dell'observer è un pattern in cui un oggetto detto subject notifica un insieme di osservatori riguardo i suoi cambi di stato o esecuzioni di azioni su questo.

Funzionalità

Il pattern observer viene generalmente usato in sistemi di gestione di eventi in cui il subject viene detto "sorgente del flusso di eventi", mentre gli observer vengono denominati "pozzi di eventi". Questo pattern risulta particolarmente utile per soddisfare dipendenze tramite accoppiamento debole.

Utilizzo

Il pattern observer viene utilizzato dal sistema per aggiornare le sessioni degli utenti autenticati alla piattaforma se lo stato del loro account utente viene aggiornato, per mantenere le sessioni consistenti con lo stato dell'account utente. Viene inoltre utilizzato per aggiornare il contenuto delle cache implementate dall'applicazione in maniera trasparente, senza richiedere alcuna azione esplicita dall'utilizzatore.

1.2.2.3 Data Access Object

Il design pattern del data access object (DAO) è un pattern utilizzato per fornire un'interfaccia ad una o più basi di dati o altri tipi di meccanismi di persistenza.

Funzionalità

Il pattern del DAO fornisce l'accesso a delle operazioni specifiche sui dati persistenti senza esporre dettagli sulla base di dati sottostante attraverso la mappatura di chiamate interne all'applicazione al layer di persistenza. Un data access object permette di separare l'insieme di dati di cui un'applicazione richiede l'utilizzo da come gli stessi vengono trattati ad un livello più basso, permettendo all'utilizzatore di ignorare la tipologia di base di dati sottostante e le strutture dati che utilizzate.

Utilizzo

Il sistema adotta il pattern DAO per le classi che lavorano con dati persistenti.

1.2 Linee Guida per la Documentazione delle Interfacce

Per assicurare un'alta leggibilità e di conseguenza una buona manutenibilità del codice, gli sviluppatori devono sottostare a delle linee guida per la stesura del codice.

1.2.1 Code Style

Gli sviluppatori devono attenersi quanto più possibile a delle linee guida ben precise dettate dalle convenzioni di stile del codice Google Java. In particolare, il codice deve essere indentato utilizzando tab e non spazi, ed essere formattato seguendo uno stile conciso e comprensibile.

1.2.2 Naming convention

E' buona pratica che i nomi da utilizzare siano:

- Descrittivi.
- Pronunciabili.
- Di uso comune.
- Non abbreviati(se non per variabili temporanee).
- Contenenti esclusivamente caratteri consentiti(A-Z,a-z,0-9).

1.2.2.1 Variabili

Anche i nomi delle variabili locali, istanze di variabili, e variabili di classe devono rispettare lo stile lower camel case. I nomi delle variabili non dovrebbero iniziare per underscore (_) o per il simbolo del dollaro ($), anche se entrambi sono permessi. È importante far sì quanto più possibile che i nomi delle variabili siano corti ma significativi; la scelta del nome di una variabile dovrebbe essere mnemonica. Nomi di variabili composte da un singolo carattere andrebbero evitati se non per variabili temporanee come contatori o variabili di buffer.

Esempi:

- int i;
- char c;
- float numeroAscoltatori;

1.2.2.2 Metodi e funzioni

I nomi dei metodi e delle funzioni devono rispettare lo stile lower camel case (o Dromedary case), dove la prima lettera della prima parola è in minuscolo e la prima lettera di ogni parola successiva è in maiuscolo. I nomi di metodi e funzioni devono essere composti da verbi o da più parole che cominciano per un verbo.

Esempi:

- run();
- getName();

1.2.2.3 Classi

I nomi delle classi devono rispettare lo stile upper camel case (o Pascal case), dove la prima lettera di ogni parola è in maiuscolo. I nomi delle classi non dovrebbero contenere acronimi o abbreviazioni, a meno che l'abbreviazione non sia molto più diffusa della forma normale, come ad esempio URL o HTML.

Esempi:

- class User {}

1.2.2.4 Costanti

I nomi delle costanti dovrebbero essere composte da parole in maiuscolo separate da underscore. I nomi delle costanti possono anche contenere cifre se appropriato, ma non come primo carattere.

Esempi:

- static final int MAX_LENGTH = 10;
- const ERROR_401 = "Unauthorized";

1.3 Definizioni, acronimi e abbreviazioni

- RAD : Requirements Analysis Document
- SDD : System Design Document
- ODD : Object Design Document

1.4 Riferimenti

- Documento RAD del progetto MusicConsole.
- Documento Dati Persistenti del progetto MusicConsole.

2. PACKAGES

Il nostro sistema presenta una suddivisione basata su tre livelli(three-tier):

- Presentation Layer.
- Application Logic Layer.
- Storage Layer.

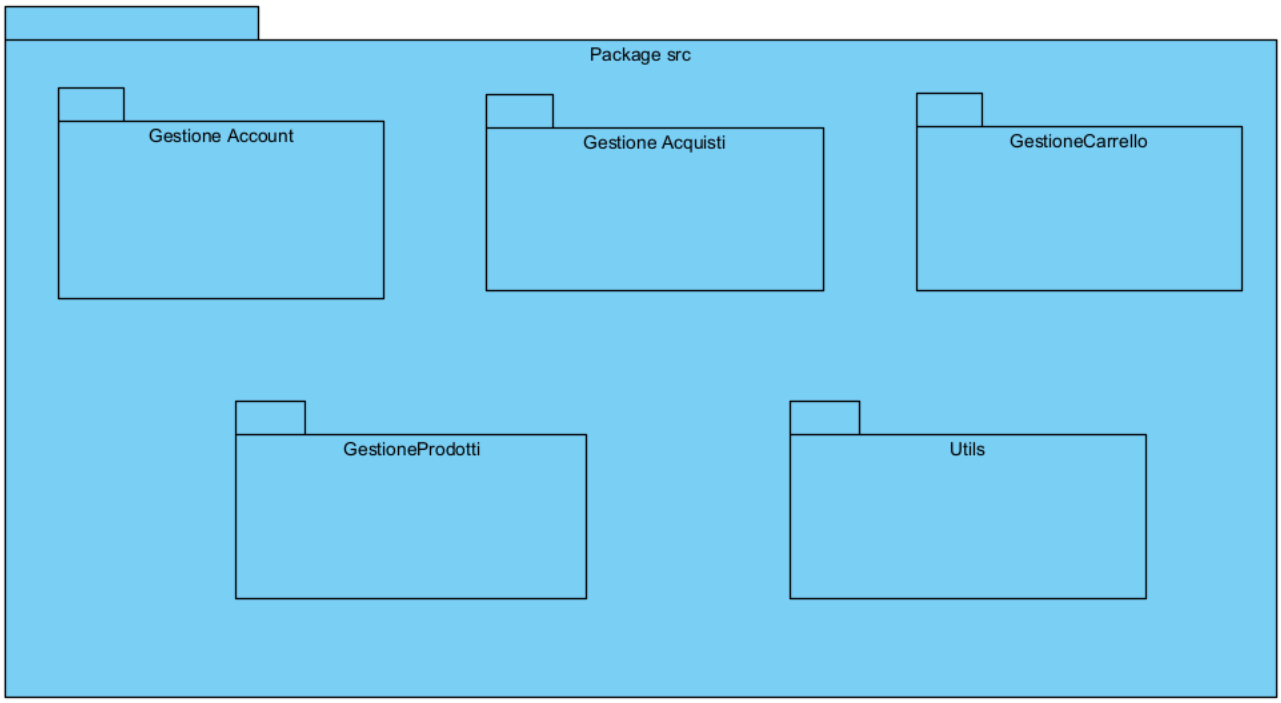
Il package MusicConsole contiene sottopackage che a loro volta inglobano classi atte alla gestione delle richieste utente. Le classi contenute del package svolgono il ruolo di gestore logico del sistema.

Presentation Layer	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, i dati.

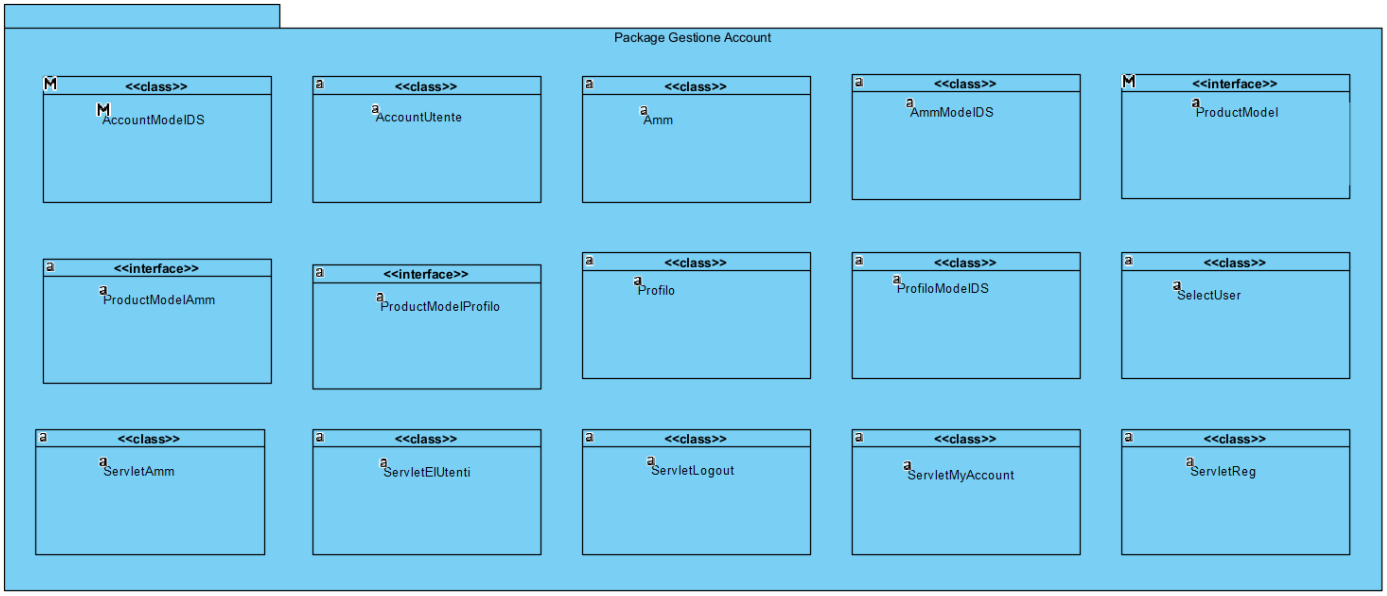
Application Logic Layer	Ha il compito di elaborare i dati da inviare al client, e spesso grazie a delle richieste fatte al database, tramite lo Storage Layer, accede ai dati persistenti. Si occupa di varie gestioni quali: • Gestione Account • Gestione Acquisti • Gestione Carrello • Gestione Prodotti • Utils

Storage Layer	Ha il compito di memorizzare i dati sensibili del sistema, utilizzando un DBMS, inoltre riceve le varie richieste dell'Application Logic Layer inoltrandole al DBMS e restituendo i dati richiesti.

2.1 Package src



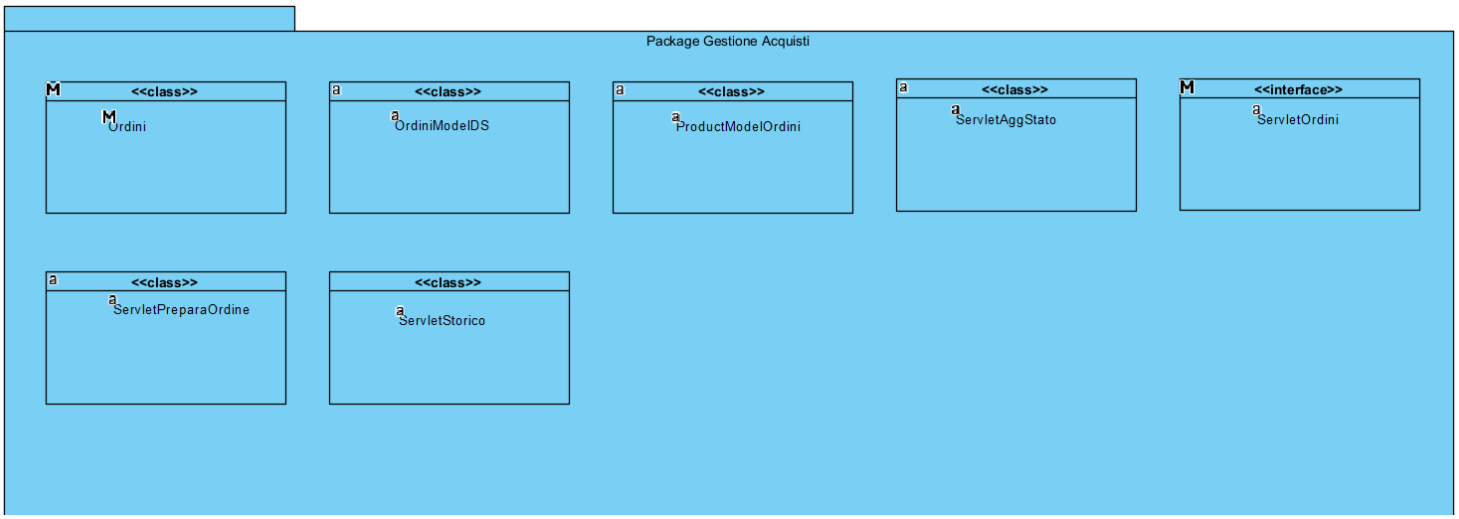
2.1.1 Package Gestione Account



Package Gestione Account	Descrizione
AccountModelDS	Questa classe contiene i metodi che permettono di effettuare l'inserimento, aggiornamento e cancellazione di un AccountUtente.
AccountUtente	Questa classe rappresenta l'AccountUtente.
Amm	Questa classe rappresenta l'Amministratore.
AmmModelDS	Questa classe contiene i metodi che permettono di effettuare l'inserimento, e cancellazione di un Amministratore.
Profilo	Questa classe rappresenta il Profilo dell'utente.
ProfiloModelDS	Questa classe contiene i metodi che permettono di effettuare l'inserimento, aggiornamento e cancellazione del Profilo di un utente.
SelectUser	Questa servlet permette di effettuare l'autenticazione di un utente.

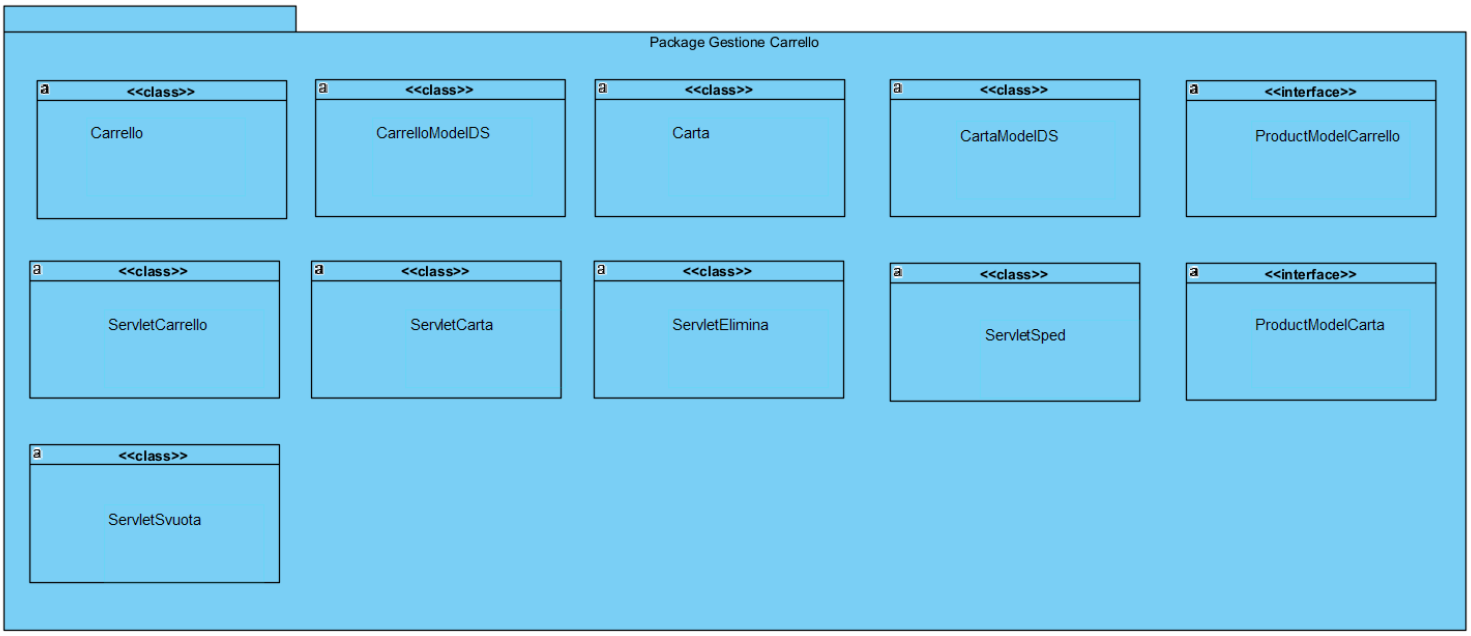
ServletAmm	Questa servlet permette l'autenticazione di un amministratore.
ServletElUtenti	Questa servlet permette di eliminare un account associato ad uno specifico utente.
ServletLogout	Questa servlet permette all'utente/amministratore di effettuare il Log-out.
ServletMyAccount	Questa servlet permette di aggiornare i dati del profilo utente, quando questi vengono modificati.
ServletReg	Questa servlet permette di recuperare i dati di registrazione di uno specifico utente, creando il proprio Account.

2.1.2 Package Gestione Acquisti



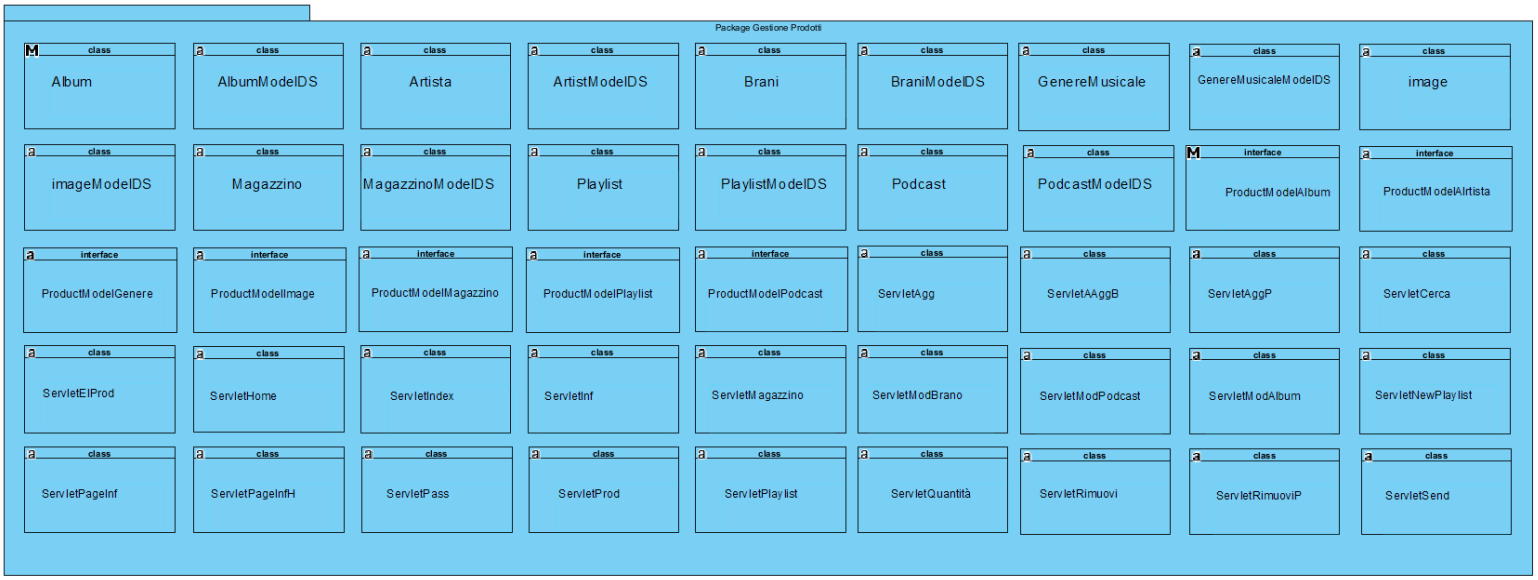
Package Gestione Acquisti	Descrizione
Ordini	Questa classe rappresenta gli Ordini.
OrdiniModelDS	Questa classe contiene i metodi che permettono al gestore degli ordini di visualizzare tutti gli ordini del sito ordinati in ordine decrescente, di aggiungere e convalidare gli ordini e restituisce gli ordini di uno specifico utente.
ServletAggStato	Questa servlet permette di settare e convalidare lo stato di un ordine.
ServletOrdini	Questa servlet restituisce tutti gli ordini del sito in ordine decrescente.
ServletPreparaOrdine	Questa servlet permette di emettere un ordine con la relativa data e fornitore.
ServletStorico	Questa servlet restituisce gli ordini che sono stati emessi di uno specifico utente.

2.1.3 Package Gestione Carrello



Package Gestione Carrello	Descrizione
Carrello	Questa classe rappresenta il carrello.
CarrelloModelDS	Questa classe contiene i metodi che restituiscono il carrello di uno specifico utente, ci permette poi di aggiungere, rimuovere e aggiornare la quantità dei prodotti nel carrello e ci restituisce anche la somma totale del carrello.
Carta	Questa classe rappresenta le Carte di Credito degli utenti.
CartaModelDS	Questa classe contiene i metodi che ci permettono di salvare, aggiornare e restituire la carta di uno specifico utente.
ServletCarrello	Questa servlet ci restituisce gli elementi del carrello di uno specifico utente e controlla la relativa disponibilità.
ServletCarta	Questa servlet ci permette di memorizzare la carta di uno specifico utente.
ServletElimina	Questa servlet ci permette di eliminare un prodotto dal carrello e controlla la relativa disponibilità.
ServletSped	Questa servlet ci permette di emettere un ordine e svuotare il carrello.
ServletSvuota	Questa servlet ci permette di svuotare direttamente il carrello.

2.1.4 Package Gestione Prodotti

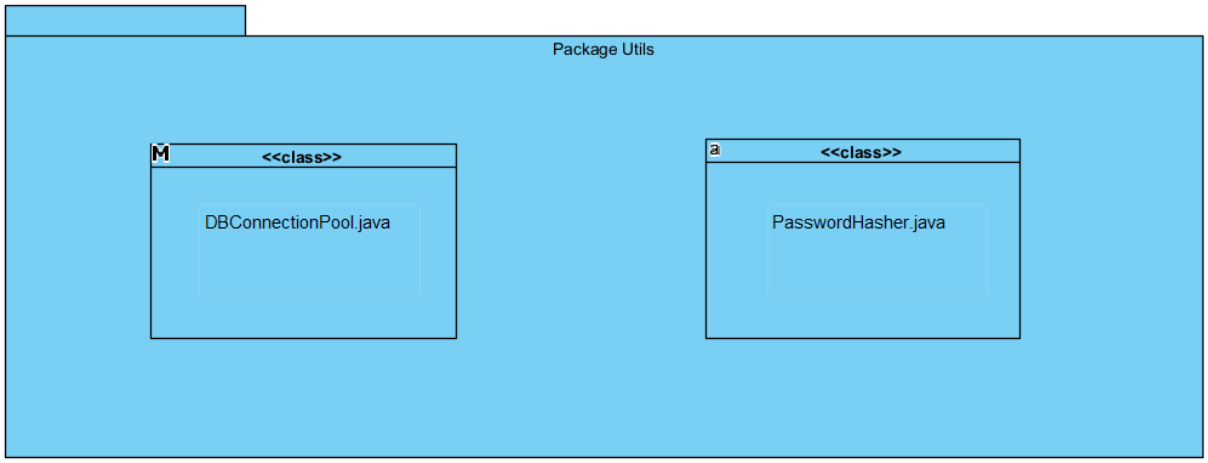


Package Gestione Prodotti	Descrizione
Album	Questa classe rappresenta il prodotto Album.
AlbumModelDS	Questa classe restituisce tutti gli Album presenti nel catalogo, ci permette anche di aggiungere, modificare e cancellare un Album.
Brani	Questa classe rappresenta il prodotto Brano.
BraniModelDS	Questa classe restituisce tutti i Brani presenti nel catalogo, ci permette anche di aggiungere, modificare e cancellare un Brano.
Magazzino	Questa classe rappresenta il Magazzino dei prodotti.
MagazzinoModelDS	Questa classe restituisce tutti i prodotti presenti in magazzino, ci permette di aggiungere, modificare la quantità ed eliminare un prodotto dal magazzino
Playlist	Questa classe rappresenta le Playlist(promemoria).

PlaylistModelDS	Questa classe restituisce tutte le playlist di uno specifico utente coi corrispettivi brani, ci permette di aggiungere una playlist, di aggiungere i brani all'interno di una specifica playlist, di eliminare una playlist ma anche di eliminare i relativi brani all'interno.
Podcast	Questa classe rappresenta il prodotto Podcast.
PodcatsModelDS	Questa classe restituisce tutti i Podcast presenti nel catalogo, ci permette anche di aggiungere, modificare e cancellare un Podcast.
ServletAgg	Questa servlet ci permette di aggiungere un Album al catalogo e nel magazzino.
ServletAggB	Questa servlet ci permette di aggiungere un Brano al catalogo.
ServletAggP	Questa servlet ci permette di aggiungere un Podcast al catalogo.
ServletCerca	Questa servlet ci permette di ricercare un prodotto secondo il suo nome, artista e genere.
ServletElProd	Questa servlet ci permette di eliminare un prodotto dal catalogo.
ServletHome	Questa servlet restituisce tutti i prodotti presenti nel catalogo per l'utente registrato.
ServletIndex	Questa servlet restituisce tutti i prodotti presenti nel catalogo per l'utente non registrato.
ServletInf	Questa servlet ci permette di aggiungere un prodotto al carrello.
ServletMagazzino	Questa servlet restituisce tutti i prodotti del catalogo di tipo Album con la relativa quantità.
ServletModBrano	Questa servlet mi permette di modificare le informazioni dei Brani.
ServletModPodcast	Questa servlet mi permette di modificare le informazioni dei Podcast.

ServletModAlbum	Questa servlet mi permette di modificare le informazioni degli Album.
ServletNewPlaylist	Questa servlet mi permette di creare una nuova playlist.
ServletPageInf	Questa servlet mi permette di recuperare le informazioni dei prodotti per un utente registrato.
ServletPageInfH	Questa servlet mi permette di recuperare le informazioni dei prodotti Per un utente non registrato.
ServletPass	Questa servlet mi permette di recuperare le informazioni da modificare dei prodotti.
ServletPlaylist	Questa servlet mi permette di aggiungere un brano in una specifica playlist di un utente.
ServletProd	Questa servlet mi restituisce tutti i prodotti presenti nel catalogo.
ServletQuantità	Questa servlet mi permette di modificare la quantità di uno specifico prodotto.
ServletRimuovi	Questa servlet mi permette di rimuovere un brano da una specifica playlist di un utente.
ServletRimuoviP	Questa servlet mi permette di rimuovere una playlist di uno specifico utente.
ServletSend	Questa servlet mi permette di recuperare i brani di una specifica playlist di un utente.

2.1.5 Package Utils



Classe Utils	Descrizione
DBConnectionPool.java	Questa classe permette a Tomcat di accedere al database.
PasswordHasher.java	Questa classe permette di crittografare le password

3. CLASS INTERFACES

3.1 ProductModel

Nome Interfaccia	ProductModel
Descrizione	-Interfaccia utilizzata nell'autenticazione di un utente per controllare il corretto inserimento delle credenziali. -Interfaccia utilizzata per l'eliminazione di un account utente. -Interfaccia utilizzata per effettuare la modifica delle password. -Interfaccia utilizzata per l'aggiunta di un nuovo account utente.
Signature dei metodi	+ doRetriveByKey(String utente , String password) : AccountUtente + doRetriveAll() : Collection<AccountUtente> + doSave (item) : boolean + doUpdate(String p1, String p2, String p3) : void + doDelete(String utente) : boolean
Pre-condizione	
Post-condizione	
Invariante	

3.2 ProductModelAlbum

Nome Interfaccia	ProductModelAlbum
Descrizione	-Interfaccia utilizzata per l'aggiunta, modifica ed eliminazione di un Album. Restituisce tutti gli album presenti nel catalogo.
Signature dei metodi	+ doRetriveByKey(String nome, String artista) : Album + doRetriveAll() : Collection<Album> + doSave (item) : void + doUpdate(String p1, String p2, String p3) : void + doUpdatePrezzo(String p1, Float p2, Integer p3) : void + doDelete(Int code) : boolean
Pre-condizione	
Post-condizione	
Invariante	

3.3 ProductModelAmm

Nome Interfaccia	ProductModelAmm
Descrizione	-Interfaccia utilizzata per gestire l'autenticazione degli amministratori.
Signature dei metodi	+ doRetriveByKey(String utente, String password) : Amministratore + doRetriveAll() : Collection<Amministratore> + doSave (item) : void + doUpdate(item): void + doDelete(Int code) : boolean
Pre-condizione	
Post-condizione	
Invariante	

3.4 ProductModelBrani

Nome Interfaccia	ProductModelBrani
Descrizione	-Interfaccia utilizzata per l'aggiunta, modifica ed eliminazione di un Brano. Restituisce tutti i brani presenti nel catalogo.
Signature dei metodi	+ doRetriveByKey(String nome, String artista) : Brano + doRetriveAll() : Collection<Brano> + doSave (item) : void + doUpdate(String p1, String p2, String p3) : void + doUpdatePrezzo(String p1, Float p2. Integer p3) : void + doDelete(Int code) : boolean
Pre-condizione	
Post-condizione	
Invariante	

3.5 ProductModelCarrello

Nome Interfaccia	ProductModelCarrello
Descrizione	-Interfaccia che permette di restituire tutti i prodotti contenuti all'interno del carrello di un determinato utente. -Interfaccia che permette di aggiungere e rimuovere prodotti. -Interfaccia che permette di ottenere il prezzo totale dei prodotti presenti nel carrello. -Interfaccia che permette di aggiornare la quantità relativa ad un prodotto.
Signature dei metodi	+ doRetriveByKey(String parola) : Carrello + doRetriveAll() : Collection<Carrello> + doSave (item) : void + doUpdate(Int quantità, Float prezzo, Int cod) : void + doDelete(Integer code) : boolean + doSum(String utente) : float + RestXUtente(String utente) : Collection<Carrello>
Pre-condizione	
Post-condizione	
Invariante	

3.6 ProductModelCarta

Nome Interfaccia	ProductModelCarta
Descrizione	-Interfaccia che permette di restituire i dati associati alla carta di credito di un determinato utente. -Interfaccia che permette di salvare e modificare i dati associati alla carta di credito di un determinato utente.
Signature dei metodi	+ doRetriveByKey(Int cvv) : Carta + doRetriveAll() : Collection<Carta> + doSave (item) : void + doUpdate(String data, int cvv) : void + doDelete(Int cvv) : boolean
Pre-condizione	
Post-condizione	
Invariante	

3.7 ProductModelMagazzino

Nome Interfaccia	ProductModelMagazzino
Descrizione	-Interfaccia che permette di restituire tutti i prodotti presenti nel magazzino. -Interfaccia che permette di aggiungere, modificare(quantità) ed eliminare prodotti.
Signature dei metodi	+ doRetriveByKey(Int cod) : Magazzino + doRetriveAll() : Collection<Magazzino> + doSave (item) : void + doUpdate(Int quantità, Int cod) : void + doDelete(Int cod) : boolean
Pre-condizione	
Post-condizione	
Invariante	

3.8 ProducModelOrdini

Nome Interfaccia	ProductModelOrdini
Descrizione	-Interfaccia che permette di restituire tutti gli ordini presenti sul nostro sito, ordinandoli in maniera Decrescente. -Interfaccia che permette di restituire gli ordini associati ad uno specifico utente. -Interfaccia che permette di aggiungere un ordine e modificare il suo stato.
Signature dei metodi	+ doRetriveByKey(String nome) : Ordini + doRetriveAll() : Collection<Ordini> + doSave (item) : void + doUpdate(String val, Int ind) : void + doDelete(Int cod) : boolean + getIndici() : collection<Ordini > + doRetrieveAllOrdinato() : collection < Ordini >
Pre-condizione	
Post-condizione	
Invariante	

3.9 ProductModelPlaylist

Nome Interfaccia	ProductModelPlaylist
Descrizione	-Interfaccia che permette di restituire tutte le playlist create da un determinato utente. -Interfaccia che permette di restituire i brani associati ad una specifica playlist. -Interfaccia che permette di creare ed eliminare una determinata playlist. -Interfaccia che permette di eliminare un brano associato ad una specifica playlist.
Signature dei metodi	+ doRetriveByKey(String parola) : Playlist + doRetriveAll() : Collection<Playlist> + doSave (item) : void + doUpdate(item) : void + doDelete(String brano, String rtista, String nplaylist, String nutente) : boolean + doDeleteP(Int code) : boolean + doDeleteProd(String brano, String artista) : boolean
Pre-condizione	
Post-condizione	
Invariante	

3.10 ProductModelPodcast

Nome Interfaccia	ProductModelPodcast
Descrizione	-Interfaccia utilizzata per l'aggiunta, modifica ed eliminazione di un Podcast. Restituisce tutti i podcast presenti nel catalogo.
Signature dei metodi	+ doRetriveByKey(String parola) : Podcast + doRetriveAll() : Collection<Podcast> + doSave (item) : void + doUpdate(String p1, String p2, String p3) : void + doUpdatePrezzo(string1, string2, string3) : void + doDelete(String nomepod) : boolean + Restituisci(String nome) : Podcast
Pre-condizione	
Post-condizione	
Invariante	

3.11 ProductModelProfilo

Nome Interfaccia	ProductModelProfilo
Descrizione	-Interfaccia che permette di restituire il profilo associato ad un determinato utente. -Interfaccia che permette di aggiungere e rimuovere un profilo associato ad un determinato utente. -Interfaccia che permette di modificare il profilo associato ad un determinato utente.
Signature dei metodi	+ doRetriveByKey(String nome) : Profilo + doRetriveAll() : Collection<Profilo> + doSave (item) : void + doUpdate(String p1, String p2, String p3) : void + doDelete(Int cod) : boolean
Pre-condizione	
Post-condizione	
Invariante	

4. GLOSSARIO

- **Componenti off-the-shelf**: prodotti software sviluppati da terzi riutilizzabili.
- **CSS**: Linguaggio per la definizione degli stili delle pagine web.
- **Framework**: Software di supporto allo sviluppo.
- **HTML**: Linguaggio per la strutturazione delle pagine web.
- **JavaScrip**t: Linguaggio di scripting nato per dare dinamicità alle pagine HTML.