UNIVERSITÀ DEGLI STUDI DI SALERNO

ANNO ACCADEMICO 2019/2020

OBJECT DESIGN DOCUMENT

Version 2.0

TOP MANAGER
Prof. Andrea De Lucia

PROJECT MANAGER
Giuseppe De Michele

Partecipanti

Nome	Matricola
Mario Balbi	0512102944
Giuseppe De Michele	0512102642
Singh Karanbir	0512104924

Revision History

Data	Versione	Descrizione	Autore
02/01/2020	1.0	Introduzione; Object design tade-off;	Singh Karanbir
05/01/2020	1.1	Package; Interfacce delle classi;	De Michele Giuseppe
07/01/2020	1.2	Design pattern Glossario	De Michele Giuseppe
17/02/2020	1.3	Revisione generale	Singh Karanbir De Michele Giuseppe Balbi Mario

Indice

1. Introduzione

1.1 Object Design Trade-off

Dopo aver stilato il documento di Requirements Analysis e il documento di System Design in cui vi è una descrizione sommaria di ciò che sarà il nostro sistema, definendo i nostri obiettivi ma tralasciando gli aspetti implementativi, andiamo ora a stilare il documento di Object Design che ha come obiettivo quello di produrre un modello che sia in grado di integrare in modo coerente e preciso tutte le funzionalità individuate nelle fasi precedenti. In particolar modo, in tale documento si definiscono le interfacce delle classi, le operazioni, i tipi, gli argomenti e il signature dei sottosistemi definiti nel System Design. Inoltre, sono specificati i trade-off e le linee guida.

Tempo di rilascio VS Manutenibilità:

Il codice del sistema deve essere comprensibile il più possibile, in modo da facilitare la fase di testing ed eventuali future modifiche da apportare. Per rispettare queste linee guida il codice sarà accompagnato da commenti volti a semplificarne la comprensione. Ovviamente questo comporterà un aumento del tempo di sviluppo del nostro progetto

Costi (ore uomo-lavoro) VS Robustezza:

Non è stata garantita la robustezza del sistema per ridurre le ore uomo-lavoro. Non sono stati effettuati test dei guasti e non è stata verificata la predisposizione per andare su cloud. Ci siamo limitati ad effettuare test di carico con 500 utenti.

Portabilità VS Efficienza:

Abbiamo garantito la portabilità, in quanto richiesto dal cliente, utilizzando un linguaggio non nativo.

Java è indipendente dalla piattaforma, purtroppo questa caratteristica comporta prestazioni inferiori in quanto non viene compilato in linguaggio macchina.

Throughput VS Sicurezza:

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

1.2 Linee guida per la documentazione dell'interfaccia

· Per il linguaggio Java usiamo la convenzione di Oracle: ·
https://www.oracle.com/technetwork/java/javase/overview/codeconvtoc-136057.html

· Per le Servlet/JSP usiamo la convenzione di Oracle: ·
https://www.oracle.com/technetwork/articles/java/servlets-jsp-140445.html

· Per il linguaggio SQL usiamo la seguente convenzione: ·
https://www.xaprb.com/blog/2008/10/26/the-power-of-a-good-sql-naming-convention/

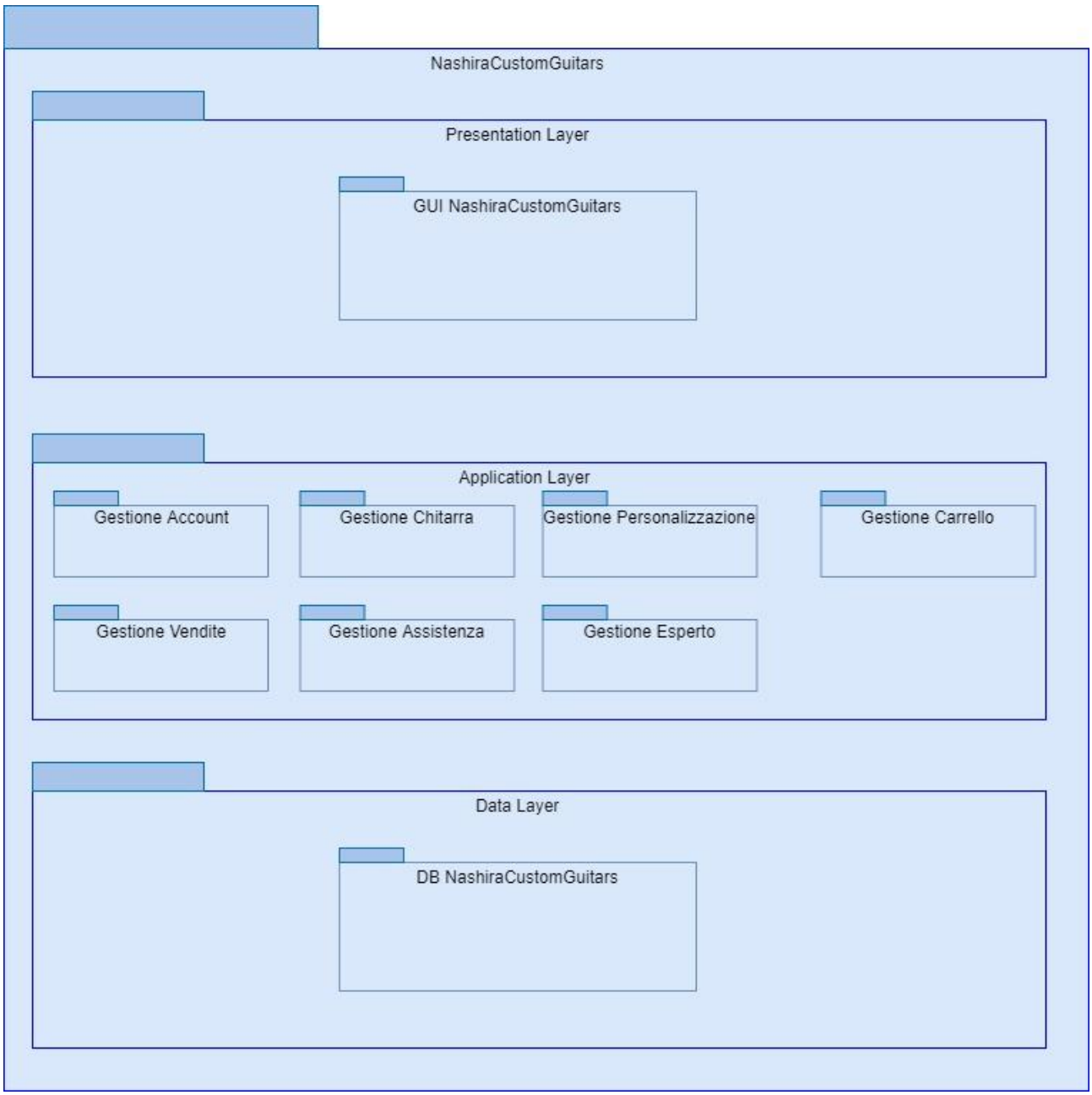
· Per i linguaggi HTML/CSS usiamo la convenzione di Google: ·
https://google.github.io/styleguide/htmlcssguide.html

·Per il linguaggio JavaScript usiamo la convenzione di Google: ·
https://google.github.io/styleguide/jsguide.html

2 Packages

La struttura del sistema NashiraCustomGuitars è strutturata secondo una divisione in package e sottopackage che raggruppano le classi che hanno il compito di gestirne la logica in base alle richieste dell'utente che ne fa uso.

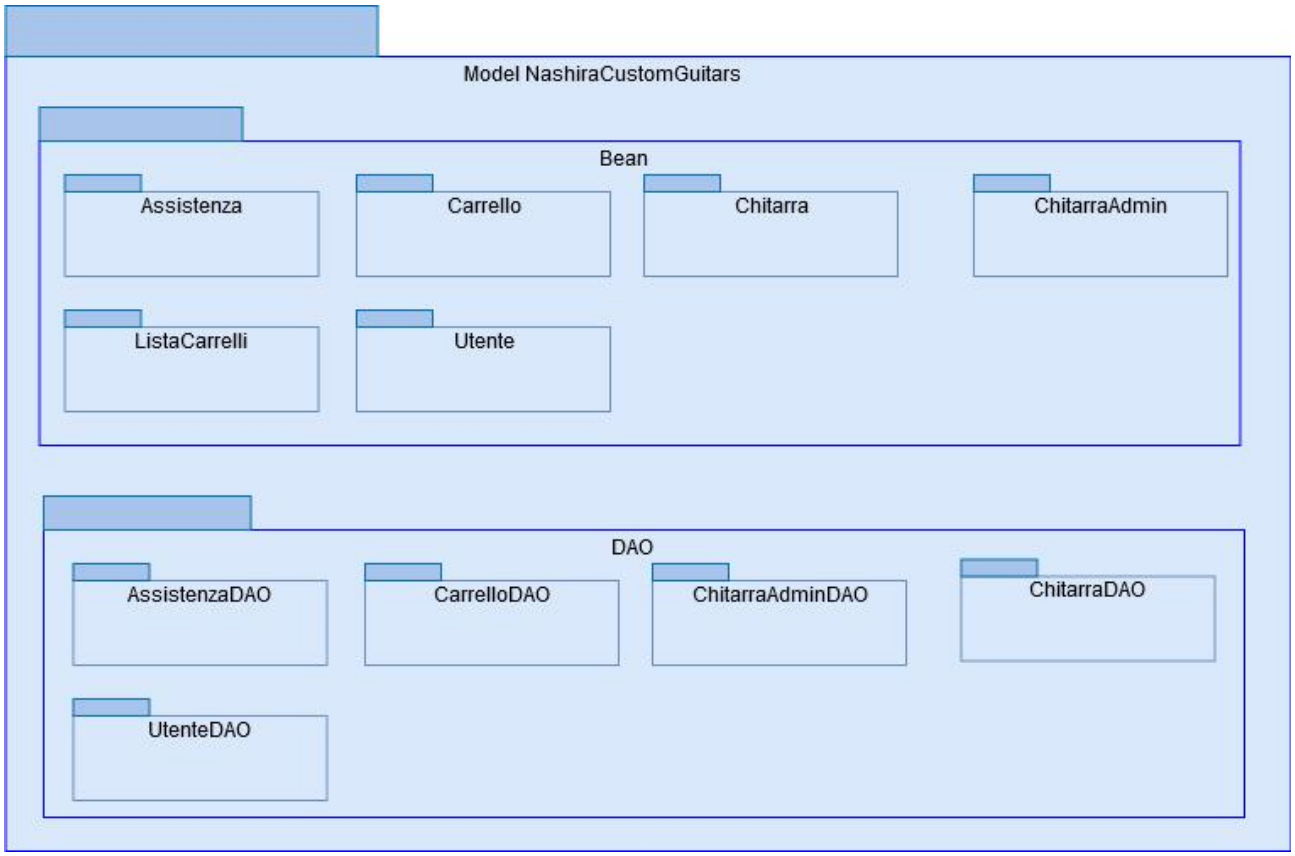
2.1 NashiraCustomGuitars



2.2 Descrizione delle classi

2.2.1 Model

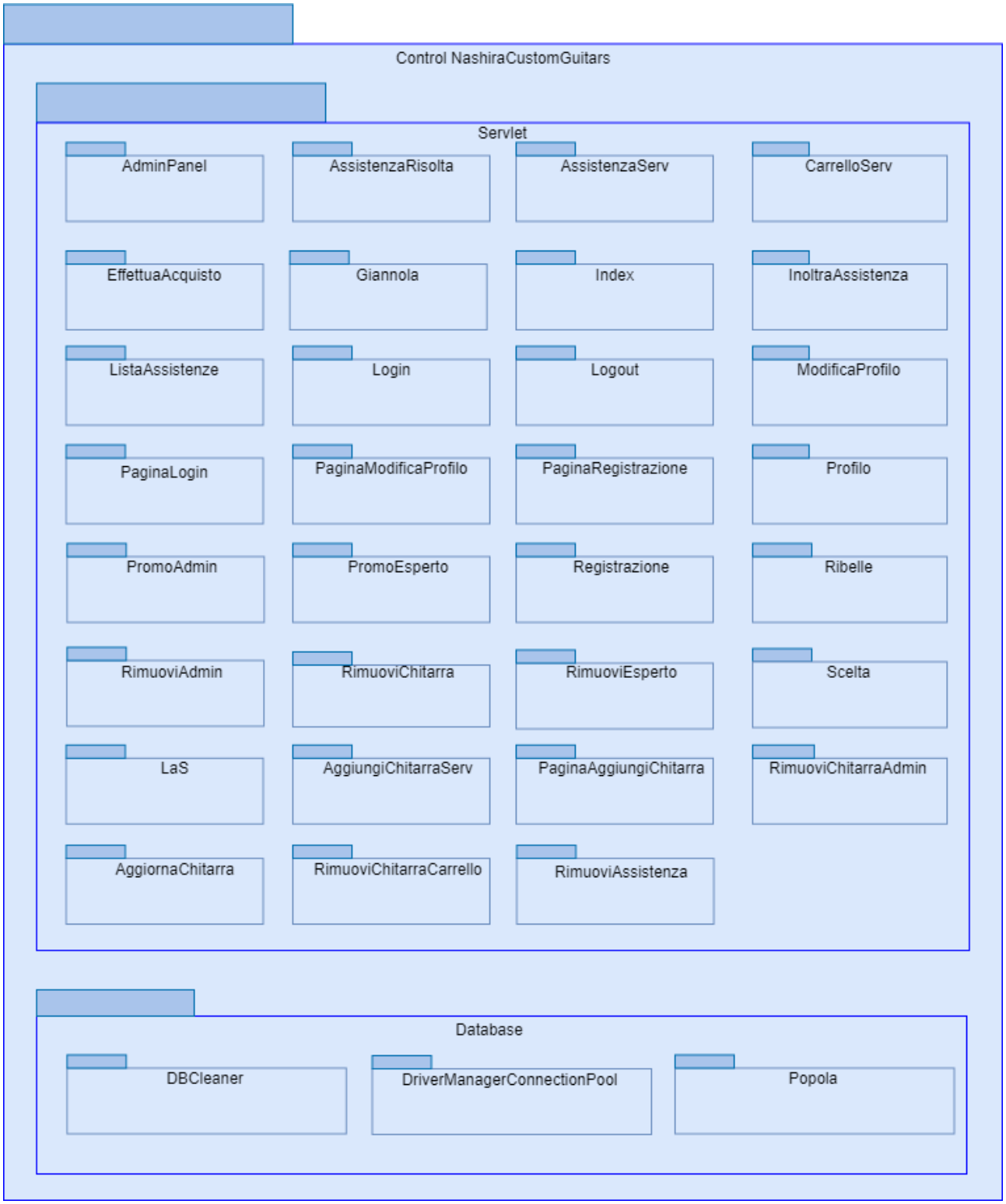
Il sottopackage “Model” è suddiviso nei package “Bean” e “DAO” presentati nei seguenti schemi e contengono rispettivamente le classi java rappresentanti le entità e le tabelle del database presenti all’interno del sistema.



Classe:	Descrizione:
UtenteBean	Descrive un utente registrato al sistema.
CarrelloBean	Descrive un carrello.
ChitarraBean	Descrive una chitarra.
AssistenzaBean	Descrive una richiesta di assistenza.
ListaCarrelloBean	Descrive una lista di carrelli.
ChitarraAdminBean	Descrive una base di una chitarra.
UtenteDAO	Descrive un utente nel database.
CarrelloDAO	Descrive un carrello nel database.
ChitarraDAO	Descrive una chitarra nel database.
AssistenzaDAO	Descrive una assistenza nel database.
ChitarraAdminDAO	Descrive una base di una chitarra nel database.

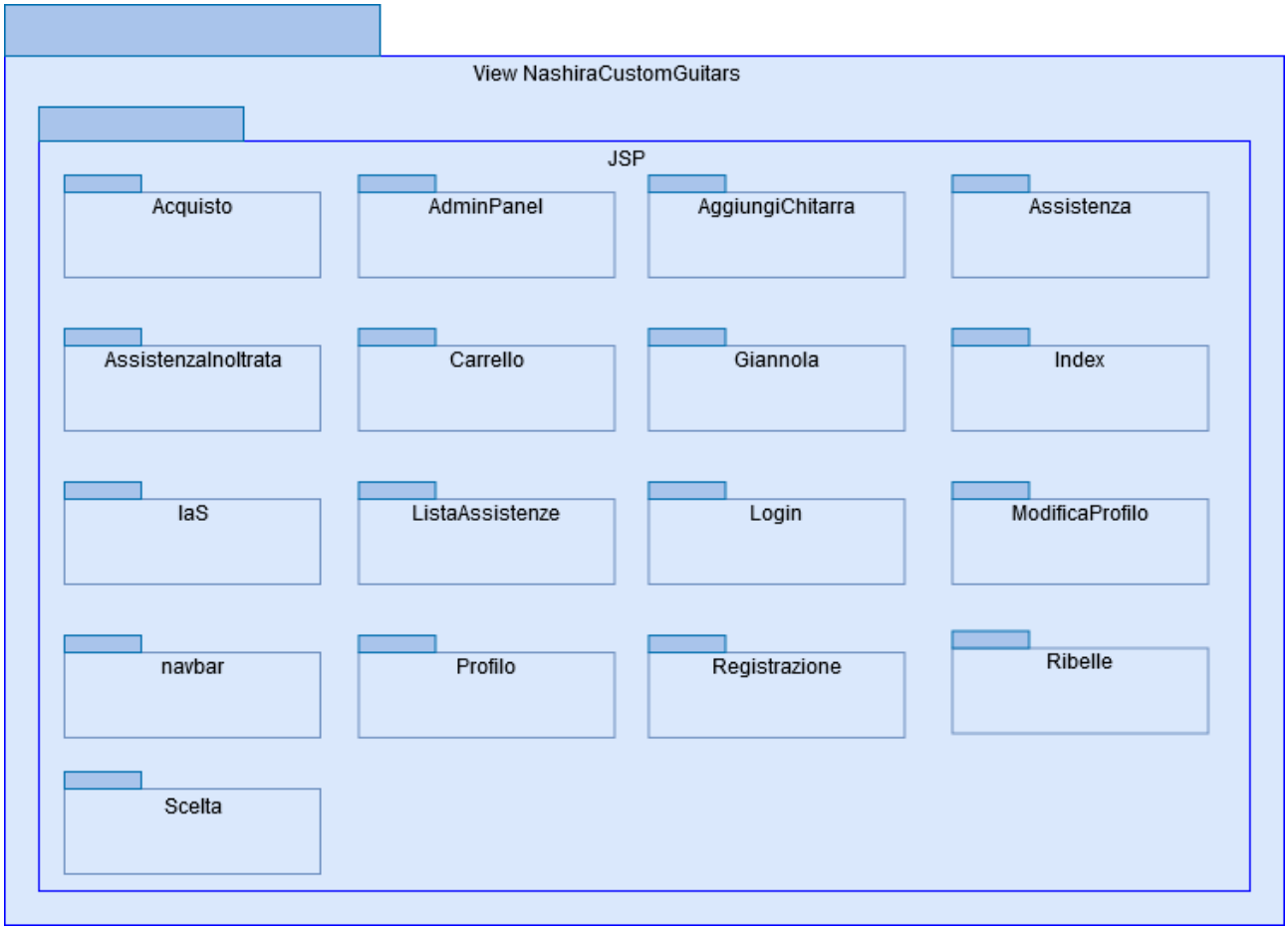
2.2.2 Control

Il “Control” è diviso nei due sottopackage “Servlet” e “database” presentati nel seguente schema e contiene le classi Java che si occupano rispettivamente della logica di controllo del sistema e della connessione con il database.



2.2.3 View

Il sottopackage “View” è presentato nel seguente schema e contiene le classi Java che si occupano della logica di presentazione del sistema.



3 Interfacce delle Classi

UtenteDAO	
Servizio	**Descrizione**
public void doSave(UtenteBean utente)	Il sottosistema permette di registrare un utente nel sistema attraverso la compilazione di un apposito form. L’oggetto utente passato come parametro verrà salvato nel database

public void doDelete(UtenteBean utente)	Il sottosistema permette di cancellare un account utente. L' oggetto utente passato come parametro verrà eliminato dal database.
public List<UtenteBean> doRetrieveAll()	Il sottosistema permette di visualizzare i dati degli utenti del sito. La lista di utenti passati come parametro verrà prelevato dal database
public Utente doRetrieveByKey(String email)	Il sottosistema permette di recuperare i dati relativi a un utente registrato nel sito. L'idUtente è il codice univoco passato come parametro per prelevare i dati dell' utente dal database.
Public void doSave (Utente utente)	Il sottosistema permette di registrare i dati di un cliente registrato nel sistema.
Public void doUpdate (Utente utente)	Il sottosistema permette di modificare i dati di un cliente registrato nel sistema.

ChitarraDAO	
Servizio	**Descrizione**
public void doSave(ChitarraBean chitarra)	Il sottosistema permette di registrare una chitarra nel sistema attraverso la compilazione di un apposito form. L'oggetto utente passato come parametro verrà salvato nel database
public void doDelete(ChitarraBean chitarra)	Il sottosistema permette di cancellare una chitarra. L' oggetto chitarra passato come parametro verrà eliminato dal database.
public List<ChitarraBean> doRetrieveAll()	Il sottosistema permette di visualizzare i dati delle chitarre del sito. La lista di chitarre passate come parametro verrà prelevato dal database
public Chitarra doRetrieveByKey(String email)	Il sottosistema permette di recuperare i dati relativi ad una chitarra salvata

	nel sito. L'idChitarra è il codice univoco passato come parametro per prelevare i dati della chitarra dal database.
Public void doSave (Chitarra chitarra)	Il sottosistema permette di registrare i dati di una chitarra nel sistema.
Public void doUpdate (Chitarra chitarra)	Il sottosistema permette di modificare i dati di una chitarra presente nel sistema.

CarrelloDAO	
Servizio	**Descrizione**
public void doSave(CarrelloBean carrello)	Il sottosistema permette di registrare un carrello nel sistema. L'oggetto carrello passato come parametro verrà salvato nel database
public void doDelete(CarrelloBean carrello)	Il sottosistema permette di cancellare un carrello. L' oggetto carrello passato come parametro verrà eliminato dal database.
public List<CarrelloBean> doRetrieveAll()	Il sottosistema permette di visualizzare i dati dei carrelli del sito. La lista di carrelli passati come parametro verrà prelevato dal database
public Carrello doRetrieveByKey(String email)	Il sottosistema permette di recuperare i dati relativi a un carrello di un utente registrato nel sito. L'email è il codice univoco passato come parametro per prelevare i dati del carrello dal database.
Public void doSave (Carrello carrello)	Il sottosistema permette di registrare i dati di un carrello di un utente registrato nel sistema.
Public void doUpdate (Crrello carrello)	Il sottosistema permette di modificare i dati di un carrello di un cliente registrato nel sistema.

AssistenzaDAO	
Servizio	**Descrizione**
public void doSave(AssistenzaBean assistenza)	Il sottosistema permette di inviare un messaggio di assistenza nel sistema attraverso la compilazione di un apposito form. L'oggetto assistenza passato come parametro verrà salvato nel database
public void doDelete(AssistenzaBean assistenza)	Il sottosistema permette di cancellare un messaggio di assistenza. L' oggetto assistenza passato come parametro verrà eliminato dal database.
public List<AssistenzaBean> doRetrieveAll()	Il sottosistema permette di visualizzare i messaggi di assistenza degli utenti del sito. La lista di messaggi di assistenza passati come parametro verrà prelevato dal database
public Assistenza doRetrieveByKey(String email)	Il sottosistema permette di recuperare i dati relativi ai messaggi di assistenza di un utente registrato nel sistema. L'email è il codice univoco passato come parametro per prelevare i dati dei messaggi di assistenza dal database.
public Assistenza doRetrieveByKey(Boolean risolto)	Il sottosistema permette di recuperare i dati relativi ai messaggi di assistenza di un utente registrato nel sistema. Risolto è il codice univoco passato come parametro per prelevare i dati dei messaggi di assistenza dal database.
Public void doUpdate (Assistenza assistenza)	Il sottosistema permette di modificare i dati di un messaggio di assistenza di un cliente registrato nel sistema.

3.1 Object Constraint Language

Nome classe	UtenteDAO
Pre-condizione	**context** UtenteDAO::public doSave(UtenteBean utente); **pre** String utente.email!= null && String utente.password!=null && String utente.citta!=null && String utente.nome!=null && String utente.cognome!=null && String utente.zip!=null && Boolean utente.esperto!=null && Boolean utente.admin!=null **context** UtenteDAO::public doUpdate(UtenteBean utente); **pre** String utente.email!= null && String utente.password!=null && String utente.citta!=null && String utente.nome!=null && String utente.cognome!=null && String utente.zip!=null && Boolean utente.esperto!=null && Boolean utente.admin!=null **context** UtenteDAO::public doDelete(String username); **pre** String utente.email!=null **context** UtenteDAO::public doRetrieveByKey(String email); **pre** String email!=null
Post-condizione	**context** UtenteDAO::public doSave(UtenteBean utente); **post** doSave =true **context** UtenteDAO::public doUpdate(UtenteBean utente); **post** doUpdate = true **context** UtenteDAO::public doDelete(String email); **post** doDelete = true **context** UtenteDAO::public doRetrieveByKey(String email); **post** doRetrieveByKey = true
Invarianti	

Nome classe	ChitarraDAO

Pre-condizione	**context** ChitarraDAO::public doSave(ChitarraBean chitarra); **pre** int chitarra.idchitarra!= null && String chitarra.pemail!=null && String chitarra.nome!=null && String chitarra.corpo!=null && String chitarra.tastiera!=null && String chitarra.top!=null && String chitarra.pickup!=null && Float chitarra.prezzo!=null && int chitarra.qnt!=null **context** UtenteDAO::public doUpdate(UtenteBean utente); **pre** int chitarra.idchitarra!= null && String chitarra.pemail!=null && String chitarra.nome!=null && String chitarra.corpo!=null && String chitarra.tastiera!=null && String chitarra.top!=null && String chitarra.pickup!=null && Float chitarra.prezzo!=null && int chitarra.qnt!=null **context** UtenteDAO::public doDelete(String email); **pre** String chitarra.email!=null **context** UtenteDAO::public doRetrieveByKey(String email); **pre** String email!=null
Post-condizione	**context** UtenteDAO::public doSave(ChitarraBean chitarra); **post** doSave =true **context** ChitarraDAO::public doUpdate(ChitarraBean chitarra); **post** doUpdate = true **context** ChitarraDAO::public doDelete(String email); **post** doDelete = true **context** ChitarraDAO::public doRetrieveByKey(String email); **post** doRetrieveByKey = true
Invarianti	

Nome classe	**AssistenzaDAO**
Pre-condizione	**context** AssistenzaDAO::public doSave(AssistenzaBean assistenza); **pre** Int assistenza.IdAssistenza!= null && String assistenza.email!=null && String assistenza.Problema!=null && Boolean assistenza.risolto!=null

	context AssistenzaDAO::public doUpdate(AssistenzaBean assistenza); **pre** Int assistenza.IdAssistenza!= null && String assistenza.email!=null && String assistenza.Problema!=null && Boolean assistenza.risolto!=null **context** AssistenzaDAO::public doDelete(Int IdAssistenza); **pre** Int assistenza.idAssistenza!=null **context** AssistenzaDAO::public doRetrieveByKey(String email); **pre** String email!=null **context** AssistenzaDAO::public doRetrieveByKey(Boolean risolto); **pre** Boolean risolto!=null
Post-condizione	**context** AssistenzaDAO::public doSave(AssistenzaBean assistenza); **post** doSave =true **context** AssistenzaDAO::public doUpdate(AssistenzaBean assistenza); **post** doUpdate = true **context** AssistenzaDAO::public doDelete(Int idAssistenza); **post** doDelete = true **context** AssistenzaDAO::public doRetrieveByKey(String email); **post** doRetrieveByKey = true **context** AssistenzaDAO::public doRetrieveByKey(Boolean risolto); **post** doRetrieveByKey = true
Invarianti	

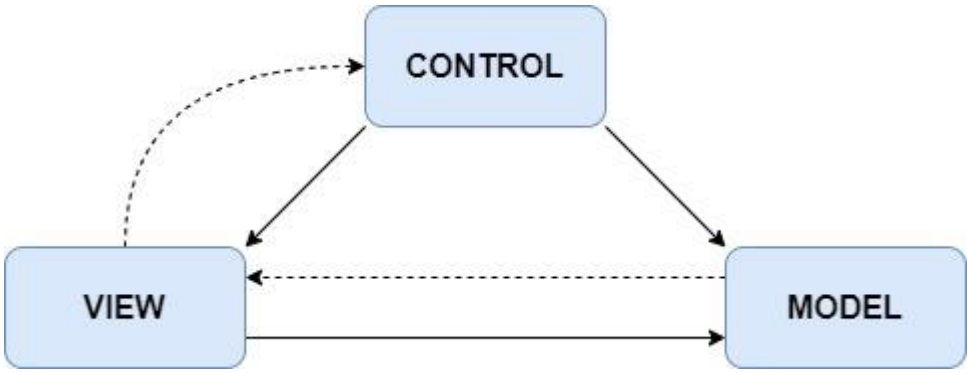
Nome classe	**CarrelloDAO**
Pre-condizione	**context** CarrelloDAO::public doSave(CarrelloBean carrello); **pre** Int carrello.IdCarrello!= null && String carrello.email!=null && Float carrello.prezzo!=null &&

	context CarrelloDAO::public doUpdate(CarrelloBean carrello); **pre** Int carrello.IdCarrello!= null && String carrello.email!=null && Float carrello.prezzo!=null && **context** CarrelloDAO::public doDelete(Int IdCarrello); **pre** Int carrello.idCarrello!=null **context** CarrelloDAO::public doRetrieveByKey(String email); **pre** String email!=null
Post-condizione	**context** CarrelloDAO::public doSave(CarrelloBean carrello); **post** doSave =true **context** CarrelloDAO::public doUpdate(CarrelloBean carrello); **post** doUpdate = true **context** CarrelloDAO::public doDelete(Int idCarrello); **post** doDelete = true **context** CarrelloDAO::public doRetrieveByKey(String email); **post** doRetrieveByKey = true
Invarianti	

4.Design Pattern

Model-View-Controller

Il pattern architetturale Model-View-Controller(MVC) è molto diffuso nello sviluppo di un sistema software, e permette la separazione della logica di business dalla logica di presentazione dei dati.



- Il model fornisce i metodi per accedere ai dati utili all'applicazione;
- Il view visualizza i dati contenuti nel model e si occupa dell'interazione degli utenti con il sistema.
- Il controller riceve i comandi dell'utente (in genere attraverso il view) e li attua modificando lo stato degli altri due componenti.

Data-Access-Object

Il *DAO* (*Data Access Object*) è un pattern architetturale per la gestione della persistenza. Si tratta di una classe usata principalmente in applicazioni web , per stratificare e isolare l'accesso ad una tabella tramite query (poste all'interno dei metodi della classe) ovvero al *data layer* da parte della *business logic* creando un maggiore livello di astrazione ed una più facile manutenibilità. I metodi del DAO con le rispettive query dentro verranno così richiamati dalle classi della business logic.
Il vantaggio relativo all'uso del DAO è dunque il mantenimento di una rigida separazione tra le componenti di un'applicazione, le quali potrebbero essere il "Modello" e il "Controllo" in un'applicazione basata sul paradigma MVC.

5. Glossario

Amministratore	Il termine indica una particolare tipologia di utente che si occupa dell'amministrazione del sistema e degli altri utenti.
Assistenza	Il termine identifica una richiesta di assistenza inviata nel sistema.
Chitarra	Il termine indica una chitarra pubblicata nel sistema.
Esperto	Il termine indica una particolare tipologia di utente che si occupa della gestione dell'assistenza.
Nashira Custom Guitars	Nome del sistema che verrà sviluppato

Personalizzazione	Il termine indica una personalizzazione allegata ad una delle chitarre
Utente	Il termine indica l'utente generico che usufruisce del sistema