

|  |
| --- |
| Object Design Document  Anno Accademico 2022/2023 |
|  |
| VeicHome   |  |  | | --- | --- | | Autore | Matricola | | Michele Del Mastro | 0512108937 | | Armando Imbimbo | 0512106867 | | Giuseppe Sabia | 0512106468 | |



Top Menager:

|  |
| --- |
| Professori |
| Prof. De Lucia Andrea |
| Prof. De Martino Vincenzo |

Partecipanti:

|  |  |
| --- | --- |
| Nome | Matricola |
| Imbimbo Armando | 0512108937 |
| Michele Del Mastro | 0512106867 |
| Giuseppe Sabia | 0512106468 |

Sommario

[1.Introduzione 4](#_Toc122336971)

[1.1 Object Design Trade-offs 4](#_Toc122336972)

[Comprensibilità vs Tempo: 4](#_Toc122336973)

[Affidabilità vs Performance: 4](#_Toc122336974)

[Interfaccia vs Usabilità: 4](#_Toc122336975)

[Supportabilità vs implementazione: 5](#_Toc122336976)

[1.2 Linee Guida per la Documentazione delle Interfacce 5](#_Toc122336977)

[Naming convention: 5](#_Toc122336978)

[Variabili: 5](#_Toc122336979)

[Metodi: 5](#_Toc122336980)

[Pacchetti: 6](#_Toc122336981)

[Classi: 6](#_Toc122336982)

[1.3 Definizioni, acronimi e abbreviazioni 6](#_Toc122336983)

[1.4 Riferimenti 6](#_Toc122336984)

[2. Packages 7](#_Toc122336985)

[2.1 Package core 8](#_Toc122336986)

[2.2 Package bean 9](#_Toc122336987)

[2.3 Package model 10](#_Toc122336988)

[2.4 Package control 11](#_Toc122336989)

[3.Class Interfaces 14](#_Toc122336990)

[3.1 QueryAcquisto 14](#_Toc122336991)

[3.2 QueryCarrello 14](#_Toc122336992)

[3.3 QueryCliente 15](#_Toc122336993)

[3.4 QueryVeicolo 17](#_Toc122336994)

[3.5 Login 18](#_Toc122336995)

# 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 obbiettivi, 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, le interfacce delle classi, le operazioni, i tipi, gli argomenti e le signature dei sottosistemi definiti nel System Design.

## **1.1 Object Design Trade-offs**

Comprensibilità vs Tempo:

Il codice per essere più comprensibile e per facilitarne il testing , abbiamo inserito le diverse classi in pacchetti i quali hanno dei nomi che fanno capire cosa faranno quelle classi.

Affidabilità vs Performance:

Viene garantito l’accesso con una coppia username e password. Il sistema deve generare dei messaggi di errore nel caso l’utente abbia inserito dei dati sbagliati o mancanti, questo implica un rallentamento della risposta da parte del sistema ciò varia anche in base alla connessione internet dell’utente.

### Interfaccia vs Usabilità:

Il sistema deve far si che l’utente riesca a utilizzare e visualizzare le funzionalità presenti, fornendo un menù contestuale con le relative funzionalità di login e logout posizionati in alto dello schermo. Nell’interfaccia saranno presenti dei pulsanti, delle immagini e form, tutto questo con l’aiuto di colori di background che permetteranno al sistema di far capire all’utente, nel caso stia compilando un form, ed ha inserito dei dati mancanti verrà mostrato un messaggio d’errore.

Supportabilità vs implementazione:

Utilizziamo il linguaggio java, con l’ausilio della piattaforma apache TomCat, come DBMS relazionale MySQL, HTML e CSS per la parte grafica, ed inoltre JQuery per la parte funzionale. Utilizziamo tali tecnologie perché essendo un applicazione web può essere utilizzata da qualsiasi dispositivo che abbia la possibilità di connettersi ad internet.

## **1.2 Linee Guida per la Documentazione delle Interfacce**

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

### Naming convention:

E’ buona norma utilizzare nomi:

1. Descrittivi
2. Pronunciabili
3. Di uso comune
4. Di lunghezza medio-corta
5. Non abbreviati

### Variabili:

I nomi delle variabili devono cominciare con una lettera minuscola. Se la variabile è composta da più parole utilizziamo l’underscore ( \_ ). Inoltre utilizziamo all’interno del codice delle variabili di supporto.

Esempio: codice\_telaio

### Metodi:

I nomi dei metodi devono cominciare con una lettera minuscola. Il nome del metodo tipicamente consiste in un verbo che identifica una 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 maggior parte dei metodi hanno un modificatore di visibilità a public.

### Pacchetti:

I nomi dei pacchetti devono cominciare con una lettera minuscola, le parole seguenti sono separate dal punto e iniziano con lettera minuscola. La funzionalità del punto viene usata per descrivere l’operazione delle classi che effettueranno all’interno.

Per esempio:

it.unisa.query

### Classi:

I nomi delle classi devono cominciare con una lettera maiuscola, e anche le parole seguenti all’interno del nome devono cominciare con una lettera maiuscola. I nomi delle classi forniscono informazioni del loro lavoro. La dichiarazione delle classi sono prevalentemente con modificatori di visibilità public.

## **1.3 Definizioni, acronimi e abbreviazioni**

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

## **1.4 Riferimenti**

* B.Bruegge, A. H. Dutoit, Object Oriented Software Engineering - Using UML, Pattern and Java, Prentice Hail, 3rd edition, 2009.
* Documento RAD del progetto.
* Documento Dati Persistenti del progetto.

# 2. Packages

L’architettura del nostro sistema è basata su tre livelli.

Architettura three-tier:

* Interface Layer
* Application Logic Layer
* Storage Layer

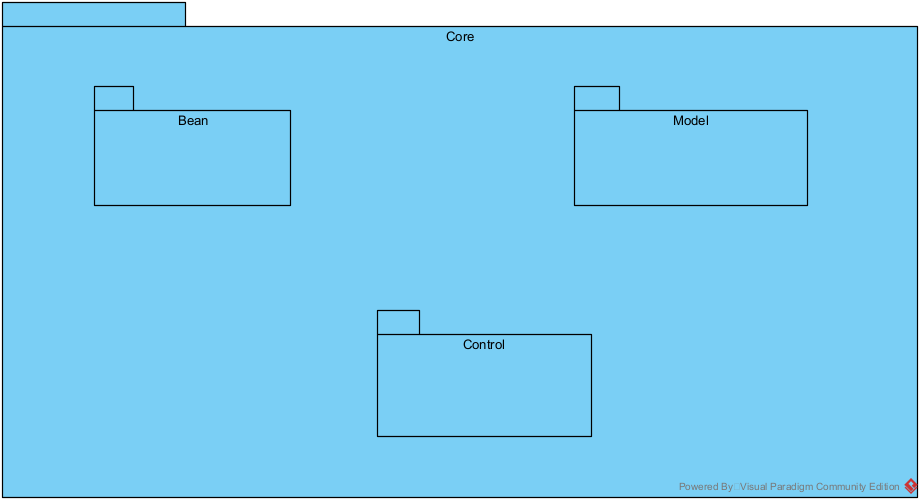
I tre pacchetti sono suddivisi:

|  |  |
| --- | --- |
| Interface Layer | E’ un livello che permette di rappresentare l’interfaccia, in cui l’utente può interagire con essa. Viene data la possibilità di visualizzare, inviare dati in input. |

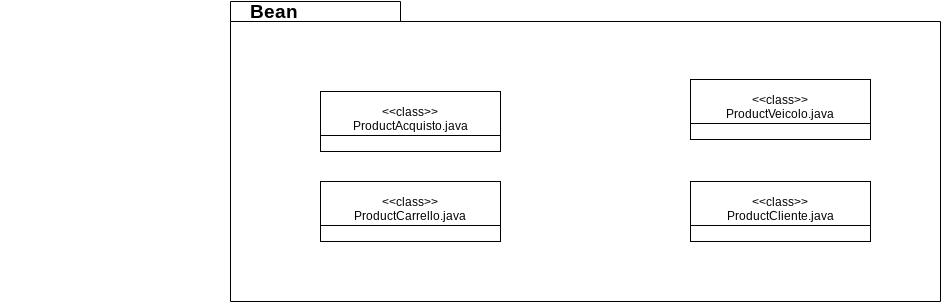
|  |  |
| --- | --- |
| Application Logic Layer | E’ un livello che ci permette di accedere ai dati persistenti che sono all’interno del nostro Storage Layer. Esso permette di elaborarli in modo da gestire le richieste che l’utente effettuerà. L’entità con le quali lavora sono:  Gestione veicolo  Gestione carrello  Gestione account  Gestione acquisto  Gestione ordini |

|  |  |
| --- | --- |
| Storage Layer | E’ un livello che ci permette di memorizzare i dati, utilizzando il DBMS. Utilizziamo MySQL con l’aiuto della sua semplice interfaccia, ci faciliterà la scrittura delle operazioni CRUD per restituirci i dati richiesti. |

## **2.1 Package core**

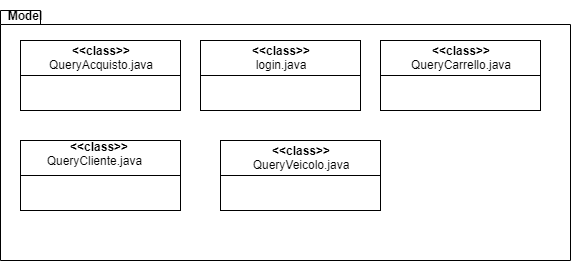


## **2.2 Package bean**



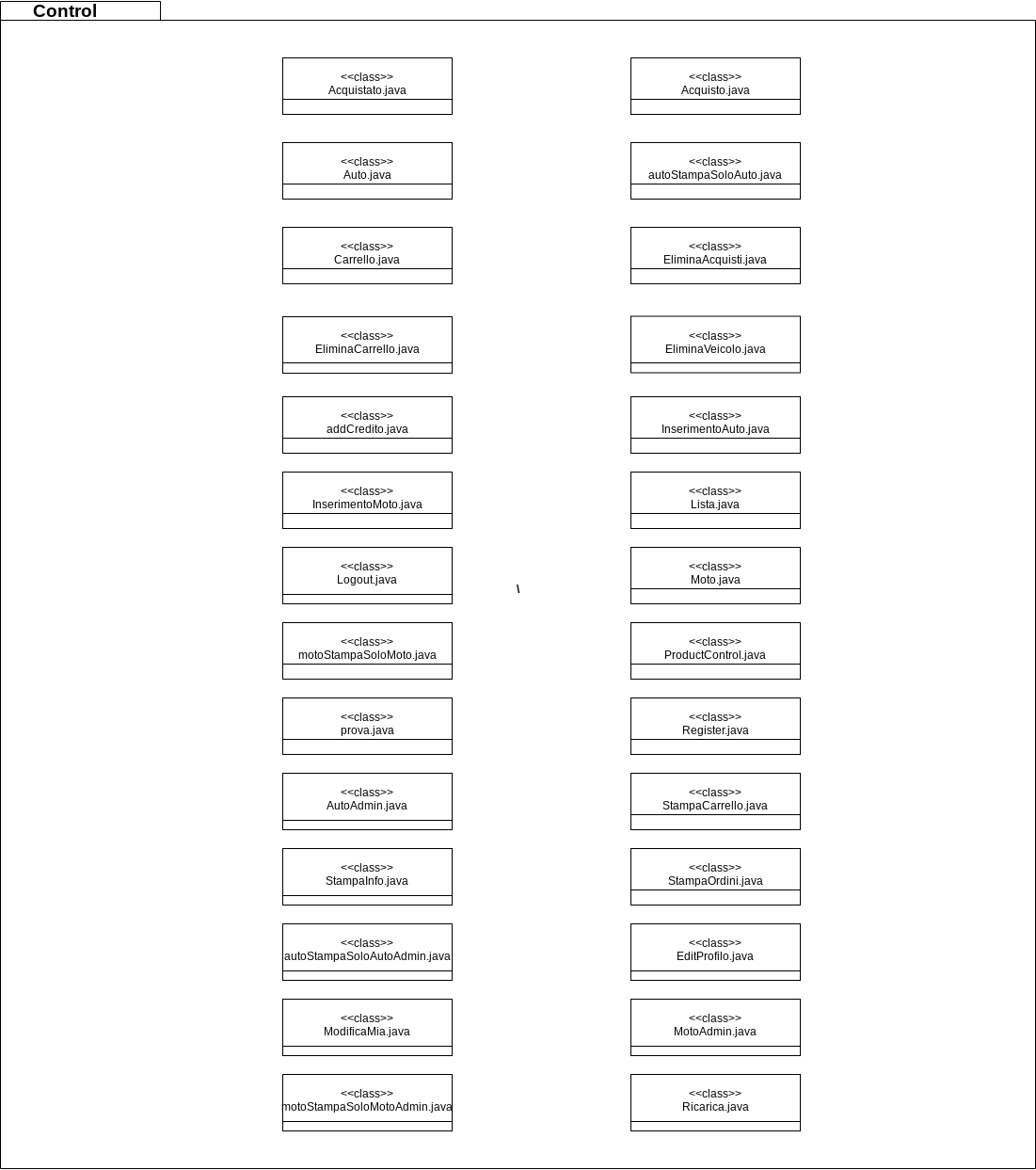
|  |  |
| --- | --- |
| Classe | Descrizione |
| ProductAcquisto.java | Questa classe descrive l’acquisto di un veicolo associato all’utente |
| ProductCarrello.java | Questa classe descrive gli elementi che si trovano all’interno del carrello |
| ProductCliente.java | Questa classe descrive il cliente registrato alla piattaforma |
| ProductVeicolo.java | Questa classe descrive i veicoli compresi auto e moto |

## **2.3 Package model**



|  |  |
| --- | --- |
| Classe | Descrizione |
| QueryAcquisto.java | Questa classe permette di effettuare un insert nella tabella acquisto, una select per la ricerca, e delete per eliminare gli acquisti nel DB. |
| QueryCarrello.java | Questa classe permette di effettuare un insert nella tabella carrello, una select per la ricerca, e delete per eliminare gli elementi nel carrello. |
| QueryCliente.java | Questa classe permette di effettuare un insert nella tabella cliente,una update per aggiornare i dati dell’utente ed una select per la ricerca. |
| QueryVeicolo.java | Questa classe permette di effettuare un insert nella tabella veicolo, una select per la ricerca, e delete per eliminare i veicoli nel DB. |
| Login.java | Questa classe permette di effettuare una select sul cliente per verificare se è un cliente o un admin |

## **2.4 Package control**



|  |  |
| --- | --- |
| Classe | Descrizione |
| Acquistato.java | Questa servlet permette di effettuare l’acquisto al cliente |
| Auto.java | Questa servlet ricerca il modello dell’auto e stampa le informazioni |
| autoStampaSoloAuto.java | Questa servlet ha il compito di ricercare all’interno del DB le auto presenti nella piattaforma e le stampa |
| Carrello.java | Questa servlet ha il compito di inserire veicoli nel carrello |
| EliminaAcquisti.java | Questa servlet ha il compito di eliminare acquisti effettuati all’interno del DB(da parte dell’admin) |
| EliminaCarrello.java | Questa servlet ha il compito di eliminare veicoli all’interno del carrello |
| EliminaVeicolo.java | Questa servlet ha il compito di eliminare un veicolo nel DB |
| InserimentoAuto.java | Questa servlet ha il compito di inserire auto sulla piattaforma |
| InserimentoMoto.java | Questa servlet ha il compito di inserire moto sulla piattaforma |
| Lista.java | Questa servlet ha il compito di ricerca e stampare la lista dei veicoli acquistati dal cliente nella sua area personale. |
| Logout.java | Questa servlet ha il compito di eliminare la sessione del cliente e uscire dal sistema |
| Moto.java | Questa servlet ricerca il modello della moto e stampa le informazioni |
| motoStampaSoloMoto.java | Questa servlet ha il compito di ricercare all’interno del DB le moto presenti nella piattaforma |
| ProductControl.java | Questa servlet ha il compito di dare un numero univoco di sessione al cliente quando accede alla piattaforma |
| Prova.java | Questa servlet permette di recuperare l’utente che ha effettuato il login e ritorna alla home del sito |
| Register.java | Questa servlet ha il compito di registrare un cliente alla piattaforma |
| StampaCarrello.java | Questa servlet ha il compito di stampare i veicoli presenti nel carrello |
| StampaInfo.java | Questa servlet ha il compito di ricercare e stampare informazioni del cliente |
| StampaOrdini.java | Questa servlet ha il compito di ricercare e stampare informazioni sugli ordini del cliente |
| addCredito.java | Questa servlet ha il compito di ridirezionare il cliente verso la pagina di ricarica del saldo |
| AutoAdmin.java | Questa servlet ricerca il modello dell’auto e stampa le informazioni |
| AutoStampaSoloAutoAdmin.java | Questa servlet ha il compito di ricercare all’interno del DB le auto presenti nella piattaforma e le stampa |
| EditProfilo.java | Questa servlet ha il compito di aggiornare i dati del cliente sulla piattaforma |
| ModificaMia.java | Questa servlet ha il compito di ricercare all’interno del DB le auto presenti nella piattaforma e le stampa |
| MotoAdmin.java | Questa servlet ricerca il modello della moto e stampa le informazioni |
| MotoStampaSoloMotoAdmin.java | Questa servlet ha il compito di ricercare all’interno del DB le moto presenti nella piattaforma e le stampa |
| Ricarica.java | Questa servlet ha il compito di ricaricare il saldo deciso dal cliente sulla piattaforma |
| Acquisto.java | Questa servlet ridireziona il cliente alla home dopo l’acquisto |

# 3.Class Interfaces

## **3.1 QueryAcquisto**

|  |  |
| --- | --- |
| Nome Classe | QueryAcquisto |
| Descrizione | Questa classe permette di accedere all’informazioni relativi agli acquisti presenti nel DB. |
| Metodi | * doSave (ProductAcquisto)   pre: !product.getTargaFK().equals("") && !product.getCodice\_fiscaleFK().equals("")&& !product.getData\_di\_acquisto().equals("")  post:inserisce acquisto   * doRetrieveByKey (codice\_fiscale: String)   pre: !codice\_fiscaleFK.equals("")  post:producacquisto bean   * doRetrieveAll (order: String)   pre:order != null && !order.equals("")  post:productacquisto bean   * doDelete () |

## **3.2 QueryCarrello**

|  |  |
| --- | --- |
| Nome Classe | QueryCarrello |
| Descrizione | Questa classe permette di accedere all’informazioni relativo al carrello dei clienti registrati o non alla piattaforma. |
| Metodi | * doSave (ProductCarrello)   pre:!product.getTarga().equals("") && !product.getModello().equals("") && !product.getPrezzo() == 0&& !product.getSessionid().equals("")  post:return true   * doRetrieveAll (order: String)   pre:order != null && !order.equals("")  post:productcarrello bean   * doDelete () * doRetrieveByKey (sessionId: String) * doDelete (product: String) * doRetrieveByKey (targa: String)   pre:!targa.equals("")  post:productcarrello bean   * doDelete (ProductCarrello)   pre:!product.getTarga().equals("")  post:carrello svuotato |

## **3.3 QueryCliente**

|  |  |
| --- | --- |
| Nome Classe | QueryCliente |
| Descrizione | Questa classe permette di accedere all’informazioni relative al cliente presenti nel DB |
| Metodi | * doRetrieveAll1 (order: String)   pre:order != null && !order.equals("")  post:productcliente bean   * doUpdate (ProductCliente)   pre:(cl.getUsername() != "" && cl.getIndirizzo() != "") && b1 != true &&!(cl.getNumero\_di\_carta() == "")  post:dati utente update   * doSave (ProductCliente)   pre:!product.getCodice\_fiscale().equals("") && !product.getNome().equals("")  && !product.getCognome().equals("") && !product.getSesso().equals("")  && !product.getIndirizzo().equals("") && !product.getData\_di\_nascita().equals("")  && !product.getPassword().equals("") && !product.getNumero\_di\_carta().equals("")  && !product.getData\_scadenza().equals("") && !product.getCvv().equals("")  && !product.getUsername().equals("") && !product.getComune\_di\_nascita().equals("")  post:utente inserito   * doRetrieveByKey (nome: String password: String)   pre:!nome.equals("") || !password.equals("")  post productCliente bean   * doUpdateSaldo (ProductCliente)   pre:!codice\_fiscale.equals(""))&&cl.getSaldo() >= 0)  post: return true (saldo aggiornato)   * controlloCarta(numero\_di\_carta:String) * controllo(String username) |

## **3.4 QueryVeicolo**

|  |  |
| --- | --- |
| Nome Classe | QueryVeicolo |
| Descrizione | Questa classe permette di accedere all’informazioni relative ai veicoli (auto/moto) presenti nel DB |
| Metodi | * doRetrieveAll5 (order: String)   pre:order != null && !order.equals("")  post:productveicolo bean   * doRetrieveByKey1 (targa: String codice\_telaio: String)   pre:!targa.equals("") && !codice\_telaio.equals("")  post:productveicolo bean   * doDelete (ProductVeicolo) * doInsertVeicoloAuto (ProductVeicolo)   pre:!product.getCodice\_telaio().equals("") && !product.getTarga().equals("") && !product.getColore().equals("")  && !product.getMarchio().equals("") && !product.getModello().equals("") && !product.getKw() == 0  && !product.getPrezzo() == 0 && !product.getPhoto().equals("") && !product.getNumero\_passegeri() == 0  && !product.getSconto().equals("")  post:il veicolo viene inserito   * doInsertVeicoloMoto (ProductVeicolo)   pre:!product.getCodice\_telaio().equals("") && !product.getTarga().equals("") && !product.getColore().equals("")  && !product.getMarchio().equals("") && !product.getModello().equals("") && !product.getKw() == 0  && !product.getPrezzo() == 0 && !product.getPhoto().equals("") && !product.getAccessori().equals("")  && !product.getCustom().equals("")  post:il veicolo viene inserito   * cercaAuto(order: String)   pre:order != null && !order.equals("")  post:productveicolo bean   * cercaMoto(order: String) * pre:order != null && !order.equals("") * post:productveicolo bean |

## **3.5 Login**

|  |  |
| --- | --- |
| Nome Classe | Login |
| Descrizione | Questa classe permette di controllare chi ha effettuato il login, cliente oppure amministratore |
| Metodi | * login(nome: String password: String)   pre:nome!=null&&password!=null  post:return cf |