****

**ODD**

**Object Design Document**

**HELPSELLER**

Revision History

| **DATA** | **VERSIONE** | **DESCRIZIONE** | **AUTORI** |
| --- | --- | --- | --- |
| 06/12/2021 | 0.1 | Prima Stesura,  Aggiunta definizioni,acronimi e abbreviazioni, analisi dei trade-off,componenti off-the-shelf,linee guida per la documentazione,design patterns | Olexiy Lysytsya |
| 10/12/2021 | 0.2 | Aggiunta packages, interfacce delle classi | Marco Esposito, Giovanni Acquaro |
| 15/01/2022 | 0.3 | Revisione | Marco Esposito, Giovanni Aquaro |

Coordinatori del Progetto

| **Top Manager** | **Prof.ssa F. Ferrucci** |
| --- | --- |
| **Project Managers** | **Caccioppoli Mario, De Matteo Antonio** |

Partecipanti

| **Partecipante** | **Matricola** |
| --- | --- |
| Mario Caccioppoli | 0522500767 |
| Antonio De Matteo | 0522501035 |
| Daniele Iervolino | 0512104975 |
| Francesco Ciampa | 0512103910 |
| Antonio Abate Chechile | 0512103670 |
| Marco Esposito | 0512106946 |
| Giovanni Aquaro | 0512105317 |
| Olexiy Lysytsya | 0512108083 |

# 

# Sommario

[**Sommario**](#_1toesrce2frz) **3**

[**1 Introduzione**](#_fzb4rxh2zysw) **4**

[**1.1 Definizioni, acronimi e abbreviazioni**](#_tm4co6g91h54) **5**

[**1.2 Analisi dei trade-off**](#_g16ttpmz0q56) **6**

[**1.3 Componenti off-the-shelf**](#_j3n00hb33lpy) **7**

[**1.4 Linee guida per la documentazione dell’interfaccia**](#_p43jhb5ft0mi) **8**

[**1.4.1 Nome dei File**](#_68tz58fwbqaa) **8**

[**1.4.2 Struttura dei File Sorgente**](#_o90ptzjonaf9) **8**

[**1.4.3 Formattazione**](#_m191o97t99c) **9**

[**1.4.4 Organizzazione dei file**](#_xzchiesfepxi) **9**

[**2 Packages**](#_o4i19los88xt) **11**

[**2.1 HelpSellerAPI**](#_8uarqplj5fcn) **11**

[**2.2 HelpSellerInterface**](#_4gmyyaueorf8) **13**

[**3 Interfacce delle classi**](#_tgabijyj634l) **16**

[**3.1 AmministratoreController**](#_uay3tjvbh5o4) **16**

[**3.2 AziendaController**](#_q62zxtux5n3g) **17**

[**3.3 DistributoreController**](#_fr2db98ro5qc) **20**

[**3.4 DocumentoController**](#_bhm0blhyjnzo) **22**

[**3.5 FileController**](#_v3xlfpk5t9m7) **24**

[**3.6 OrdineController**](#_7gfx6awjdnlr) **25**

[**3.7 OrdineProdottoController**](#_vnk6h9icysln) **27**

[**3.8 ProdottoController**](#_aajx0c56a5v3) **31**

[**3.9 RecensioneController**](#_uo0g00t0aggq) **35**

[**3.10 ScontoController**](#_rc3q677mxkos) **38**

[**3.11 ScontoProdottoController**](#_61j19r9aq845) **41**

[**3.12 TrasportoController**](#_40zqg342qp0i) **44**

[**3.13 Utente Controller**](#_z94360v946cj) **46**

[3.14 Raccomandazione Controller](#_27bsri4xkjge) 47

[**4 Design patterns**](#_dvwlvtf6bm90) **48**

[**4.1 Bridge Design Pattern:**](#_la1x0w130s6i) **49**

[**4.2 Observer Design Pattern:**](#_vv40tlcpfib0) **49**

[**4.3 Facade Pattern:**](#_ranwd7crlnlb) **50**

[**4.4 Service Layer Design Pattern:**](#_dppriqtyfkyw) **50**

[**4.5 Repository Design Pattern:**](#_e3r19ilgln1p) **51**

[**4.6 Singleton Design Pattern:**](#_t37bpq5pkaj4) **52**

[**5 Glossario**](#_gbhxcmeebyg5) **53**

# 1 Introduzione

Help Seller è una piattaforma che si propone di semplificare le interazioni tra le Aziende ed i distributori, semplificando la distribuzione di prodotti di beni di consumo di tipo alimentare e non, nel territorio.

Questo documento, usato come supporto dell’implementazione, ha lo scopo di produrre un modello capace di integrare in modo coerente e preciso tutti i servizi individuati e descritti nelle fasi precedenti. In particolare definisce le interfacce delle classi, le operazioni, i tipi, gli argomenti e la signature dei sottosistemi definiti nel System Design.

In questa prima sezione del documento, verranno descritti i trade-offs e le linee guida per la fase di implementazione, riguardanti la nomenclatura, la documentazione e le convenzioni sui formati.

## 1.1 Definizioni, acronimi e abbreviazioni

| **Termine** | **Definizione** |
| --- | --- |
| **Front-end** | la parte visibile all'utente di un [programma](https://it.wikipedia.org/wiki/Programma_(informatica)) e con cui egli può interagire — tipicamente un'[interfaccia utente](https://it.wikipedia.org/wiki/Interfaccia_utente) |
| **Back-end** | La parte che permette l'effettivo funzionamento di queste interazioni. |
| **Framework** | Un'architettura logica di supporto (spesso un'implementazione logica di un particolare design pattern) sulla quale un software può essere progettato e realizzato, spesso facilitandone lo sviluppo |
| **DB** | Abbreviazione di database, entità facente parte della struttura complessiva del progetto che consente la memorizzazione di informazioni |
| **Enterprise Java Bean** | componenti software che implementano, lato server, la logica di business di un’applicazione web all'interno dell'architettura Java EE espletando servizi a favore della parte di front-end |
| **Java EE** | Java EE è l’acronimo di Java Enterprise Edition, oggi sostituito da Jakarta Enterprise Edition, è un insieme di specifiche le cui implementazioni vengono principalmente sviluppate in [linguaggio di programmazione](https://it.wikipedia.org/wiki/Linguaggio_di_programmazione) [Java](https://it.wikipedia.org/wiki/Linguaggio_Java) e ampiamente utilizzata nella [programmazione Web](https://it.wikipedia.org/wiki/Programmazione_Web). |
| **JPA** | Le ‘Java Persistence API’ sono un framework per il linguaggio di programmazione Java che si occupa della gestione della persistenza dei dati di un DBMS relazionale nelle applicazioni che usano le piattaforme Java Platform, Standard Edition e **Java Enterprise Edition**. |
| **Spring** | In informatica Spring è un framework open source per lo sviluppo di applicazioni su piattaforma Java. Con Spring Boot, è possibile creare una applicazione avente un metodo main che lancia l'intera applicazione web, compreso il web server integrato. |
| **Maven** | In informatica Apache Maven è uno strumento di gestione di progetti software basati su Java e build automation. Maven usa un costrutto conosciuto come Project Object Model (POM); un file XML che descrive le dipendenze fra il progetto e le varie versioni di librerie necessarie nonché le dipendenze fra di esse. In questo modo si separano le librerie dalla directory di progetto utilizzando questo file descrittivo per definirne le relazioni. |

## 

## 1.2 Analisi dei trade-off

**Prestazioni VS Costi**

Considerando il sistema che stiamo realizzando, possiamo dire che il non eccessivo budget a nostra disposizione ci ha spinto a realizzare la piattaforma utilizzando software open source minimizzando così i costi ma al tempo stesso ottenendo una piattaforma stabile e funzionale.

**Comprensibilità VS Costi**

La comprensibilità del codice è un aspetto molto importante soprattutto per la fase di testing. Ogni classe e metodo deve essere facilmente interpretabile anche da chi non ha collaborato al progetto. Si preferisce quindi aggiungere costi per la documentazione al fine di rendere il codice comprensibile anche alle persone non coinvolte nel progetto o che sono subentrate nel team dopo l’implementazione della piattaforma.

**Tempi VS Costi**

Per l'implementazione del sistema si è scelto di utilizzare lo Spring Framework e Maven per il lato back-end, per il front-end invece si è ricorso ad Angular e Bootstrap. Nonostante la poca esperienza del team con queste tecnologie, l'utilizzo di tali specifiche ci permette di risparmiare moltissimo tempo sulla scrittura di codice che verrà generato automaticamente. Questo risparmio è vantaggioso rispetto al tempo necessario all'apprendimento.

**Scalabilità VS Prestazioni**

Data la sua funzione, la scalabilità costituisce uno degli obiettivi chiave durante l’implementazione della piattaforma. Il quantitativo di prodotti offerti dalle aziende può crescere nel tempo ma in particolare la mole di distributori che interagiscono col sistema può crescere rapidamente ed all’improvviso, pertanto è fondamentale che la piattaforma sia in grado di affrontare eventuali picchi di utenza.

**Tempo di risposta VS Spazio di memoria**

La scelta di utilizzare un DB relazionale è scaturita dai diversi vantaggi che se ne derivano: - Gestione consistente dei dati; - Tempo di risposta basso rispetto all’utilizzo del file system; - Accesso veloce e concorrente ai dati.

Lo svantaggio nell’utilizzo di un DB relazionale è che richiede il triplo dello spazio di memoria rispetto ad un file system e si trovano difficoltà nel momento in cui si vuole gestire enormi quantità di dati.

Per questo motivo durante la progettazione del database si è cercato di evitare di memorizzare informazioni ridondanti ma bensì limitarsi a quelle strettamente necessarie per il corretto funzionamento dei servizi offerti.

**Usabilità vs Costi**

L’esperienza dell’utente sulla piattaforma deve essere il più piacevole e meno frustrante possibile, si è deciso quindi di aggiungere costi per rendere il sistema quanto più usabile possibile. La navigazione sarà facile ed intuitiva e le operazioni saranno quanto più immediate possibili.

**Sicurezza vs Tempi**

Per garantire la sicurezza dei dati e per limitare atti di sabotaggio da parte di malintenzionati si è deciso di favorire la sicurezza inserendo form di autenticazione e metodi di verifica da parte degli amministratori del sistema. Questo rende più lenta l'esecuzione di alcune operazioni

.

## 1.3 Componenti off-the-shelf

Come già detto nel paragrafo relativo ai trade offs, durante lo sviluppo dell’applicazione si è fatto uso di componenti off-the-shelf: componenti software disponibili sul mercato per facilitare la creazione del progetto. Per il sistema che si vuole realizzare ci interessa un framework per applicazioni web, per il front-end oltre che diverse librerie. Il framework che andremo ad utilizzare lato front-end è Angular, che è un framework open source che contiene una raccolta di strumenti liberi per la creazione di siti e applicazioni per il Web. Angular offre la possibilità di costruire applicazioni garantendo il riuso del codice oltre che a offrire la possibilità di rendere il sistema applicativo multipiattaforma.

Lato back-end invece si è fatto ricorso al framework “Springboot” che agevola lo sviluppo di applicazione stand-alone production-grade e Spring based. Tra le funzionalità principali offerte da Springboot troviamo:

* integrazione di Tomcat, il programmatore non deve preoccuparsi della gestione del container su cui girerà l’applicazione
* creazione di applicazioni stand-alone che quindi non richiedono software di terzi parti per il funzionamento
* offerta di dependencies Maven predefinite semplificando la build configuration
* configurazione automatica di Spring e librerie di terze parti

Spring non è altro che un ulteriore framework open source per lo sviluppo di applicazioni java. È stato largamente riconosciuto all'interno della comunità Java quale valida alternativa al modello basato su Enterprise Java Bean (EJB). Rispetto a quest'ultimo, il framework Spring lascia una maggiore libertà al programmatore fornendo allo stesso tempo un'ampia e ben documentata gamma di soluzioni semplici adatte alle problematiche più comuni.

Anche nello sviluppo della ‘componente intelligente’ sono state utilizzate diverse librerie che hanno fornito validi strumenti per la realizzazione dell’agente, applicando la filosofia del “non reinventare la ruota”. Nello specifico si è fatto ricorso alle seguenti librerie Python:

* Pandas il quale offre diversi metodi volti all’esecuzione di metodi matematici quali ad esempio la decomposizione ai valori singolari
* Numpy per la gestione di strutture quali matrici
* Matplotlib per la generazione di grafici volti all’analisi dei dati trattati

## 

## 1.4 Linee guida per la documentazione dell’interfaccia

* Prima dell’implementazione della logica di un qualsiasi indicatore è opportuno sottomettere ai Project Manager, l’algoritmo che si intende seguire per l’implementazione di questi, in modo che eventuali correzioni della logica da seguire possano essere apportate prima di imbattersi nella sintassi degli strumenti scelti.
* Ogni metodo e ogni file devono essere preceduti da un commento, o più precisamente da una documentazione che riporti l’obiettivo che si vuole e deve raggiungere con il nome/i dell’autore/i. Inoltre bisogna commentare, giustificare delle decisioni particolari o dei calcoli.
* La convenzione che deve essere adottata da tutti i team member per quanto riguarda i nomi delle variabili, è la nota **Camel Notation**, eccezion fatta per i nomi delle table e degli attributi presenti nel DB. Ciò è dovuto dal fatto che durante lo sviluppo del backend la connessione al server usufruisce di JPA, il quale necessita di una sua sintassi specifica, ad esempio anziché avere ordineUtente avremo ordine\_utente.

### 1.4.1 Nome dei File

La convenzione per quanto riguarda i nomi dei file (ma anche delle operazioni e delle variabili) è quella di avere nomi evocativi, ma soprattutto in lingua italiana. Eccezione fatta per tutti quei termini tipici dello sviluppo nei linguaggi utilizzati e per quei termini ove anche la migliore traduzione in italiano non regge il confronto.

Il nome del file sorgente corrisponde al nome della classe contenuta al suo interno con distinzione tra maiuscole e minuscole. Al nome della classe possono essere aggiunti tre suffissi:

* .java per il file sorgente;
* .class per il Bytecode del file.
* .py per gli script in Python del modulo FIA

### 1.4.2 Struttura dei File Sorgente

Ogni file sorgente contiene esattamente una sola classe pubblica o interfaccia. Ogni sezione del File deve essere separata con una linea vuota. La struttura interna del File è la seguente:

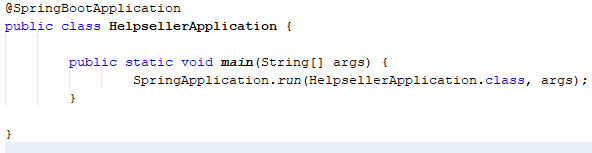
* **Descrizione Classe**: Tutti i file devono iniziare con un commento nel quale è esplicitato il nome della classe, una breve descrizione, l’autore, la versione e la data di creazione e infine le informazioni di Copyright.
* **Dichiarazione del Package e degli Import**: La prima istruzione dopo la descrizione della classe deve essere la dichiarazione del Package, successivamente è possibile dichiarare gli import necessari. Le dichiarazioni del Package e degli Import sono separati da una linea vuota. Infine gli Import sono ordinati come segue:
* ○ Tutti gli Import Statici in un singolo blocco;
* ○ Tutti gli Import Non Statici in un singolo blocco.

Ogni blocco è separato da una linea vuota

* **Dichiarazione della Classe o dell’Interfaccia**: Il file sorgente deve contenere una sola Classe o una sola Interfaccia. Gli elementi contenuti al loro interno devono essere ordinati nel seguente modo:
  + Dichiarazione Variabili Statiche della Classe: Prima le variabili di classe public, poi le variabili protected e infine quelle private;
  + Dichiarazioni Variabili di istanza: Prima le variabili di classe public, poi le variabili protected e infine quelle private;
  + Costruttori;
  + Metodi: Questi metodi devono essere raggruppati in base alla loro funzionalità piuttosto che in base a regole di visibilità o accessibilità. I metodi con nomi simili devono essere vicini tra loro.

### 1.4.3 Formattazione

Non è specificato se l’indentazione del codice deve avvenire tramite l’utilizzo degli spazi o tramite l’utilizzo delle tabulazioni. E’ però importante per la leggibilità del codice, seguire un’opportuna indentazione guidata dal livello di profondità dell’istruzione

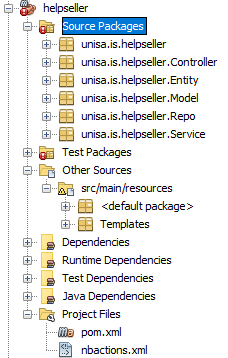


Nell’esempio troviamo al primo livello la definizione della classe, subito dopo la definizione del metodo e successivamente il contenuto di tale metodo.

I tre livelli hanno una profondità di indentazione diversa. Questa pratica deve essere usata soprattutto per le istruzioni FOR, IF dove è buona pratica scendere di livello.

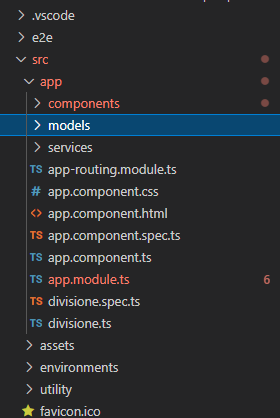
### 1.4.4 Organizzazione dei file

L’organizzazione generale delle directory dell’applicazione è già fortemente strutturata dai framework utilizzati, ad esempio per quanto riguarda il back-end le varie classi java vengono suddivise secondo la seguente struttura:



In **Source Packages** troviamo sei sotto-directory ognuna popolata da componenti dalla funzionalità ben definita, ad esempio in Entity troveremo la definizione degli oggetti che rappresentano le entità contenute nel DB, in Service troviamo i metodi forniti dalle entità.

Nella directory **Test Packages** troviamo quelli che sono i test eseguiti per assicurare il corretto funzionamento dell’applicazione In **Other Sources** possiamo trovare il file di configurazione “application.properties” fondamentale alla configurazione della connessione al DB. **Dependencies** conterrà tutte le dipendenze gestite automaticamente da Maven ed in **Project Files** possiamo trovare ulteriori file di configurazione.



Per quanto riguarda il front-end in **components** troviamo tutti i componenti che andranno a realizzare le funzionalità definite dai requisiti funzionali. I componenti sono un incapsulamento di stile, logica e visualizzazione, rispettivamente css, typescript e html. Nella cartella **models** troviamo tutte le interfacce per incapsulare in oggetti i dati recuperati dal back-end. Nella cartella **services** invece troviamo tutte le classi service che permettono le chiamate al back-end

# **2 Packages**

In questa sezione viene rappresentata la suddivisione in package del sistema in base all’architettura scelta.

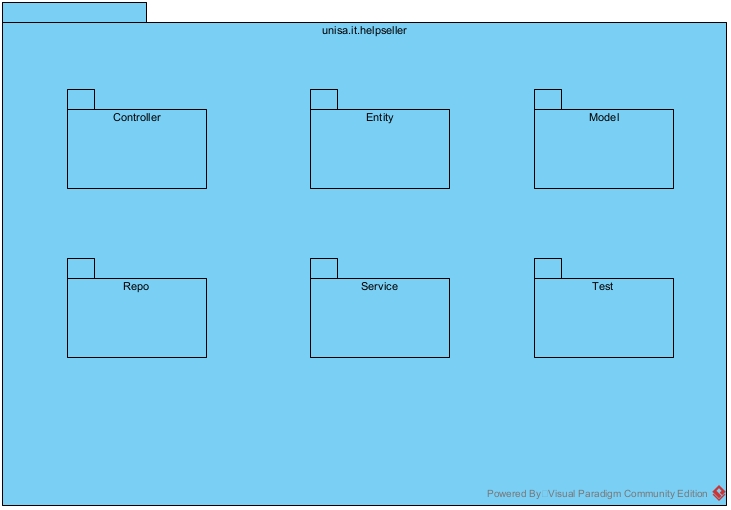
Si è scelto di dividere il sistema in due parti:

● HelpsellerAPI, nel quale verranno inserite le classi relative al Model e al Control;

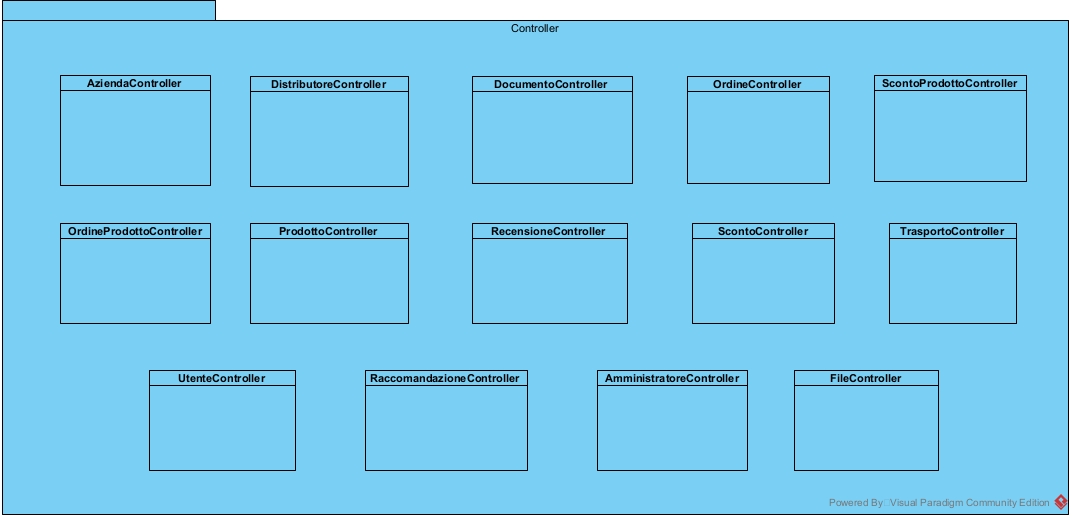
● HelpsellerInterface, nel quale verranno inserite le classi e i file relativi all’interfaccia grafica View.

## 2.1 HelpSellerAPI

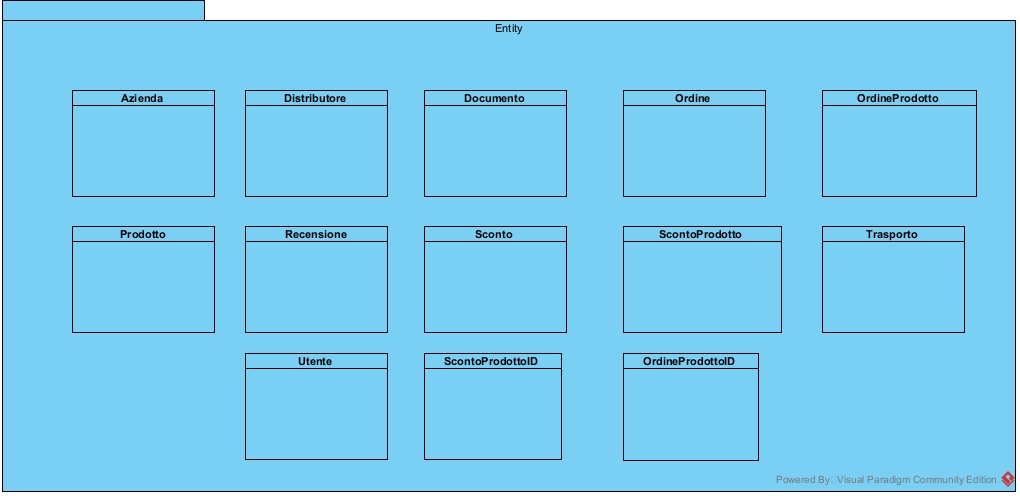
Nella presente sezione si mostra la struttura del package principale di HelpSellerAPI. La struttura generale è stata ottenuta a partire dalla scelta di creare un package separato per ogni tipo di oggetto (Control, Model, Service ,Repo, Entity) contenente tutte le classi che riguardano quel tipo di oggetto.



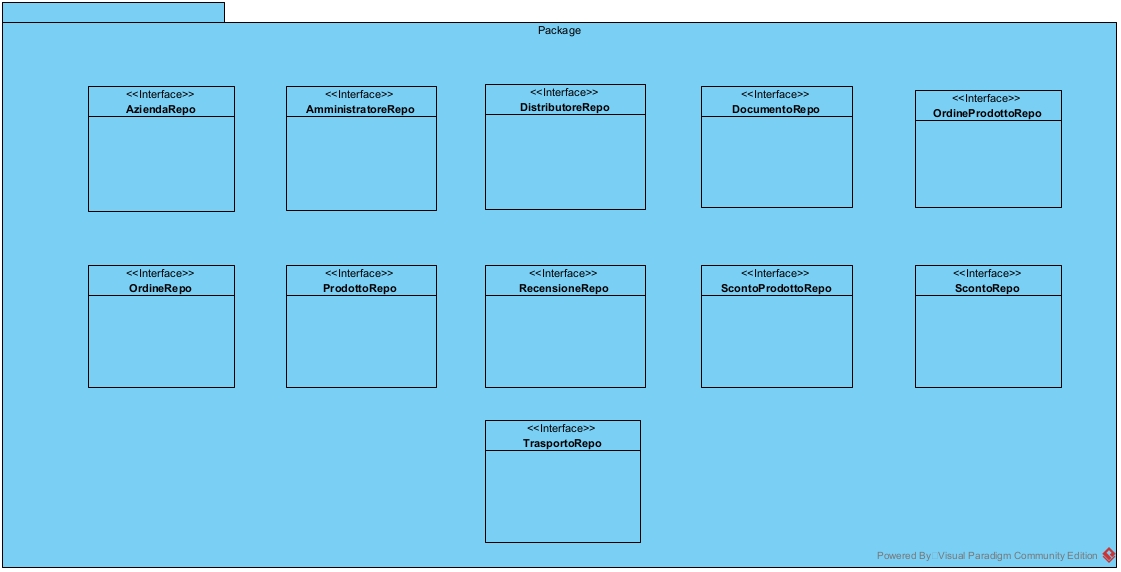
**2.1.1 Package Controller**



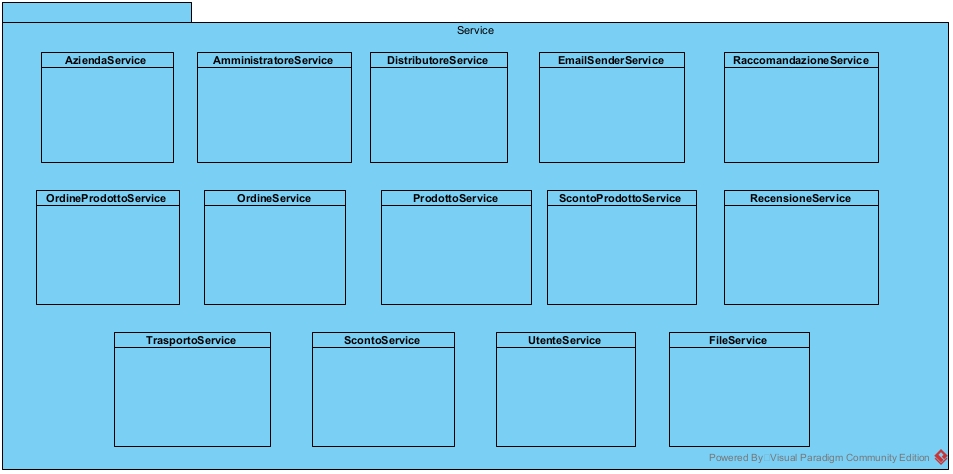
**2.1.2 Package Entity**



**2.1.3 Package Repo**

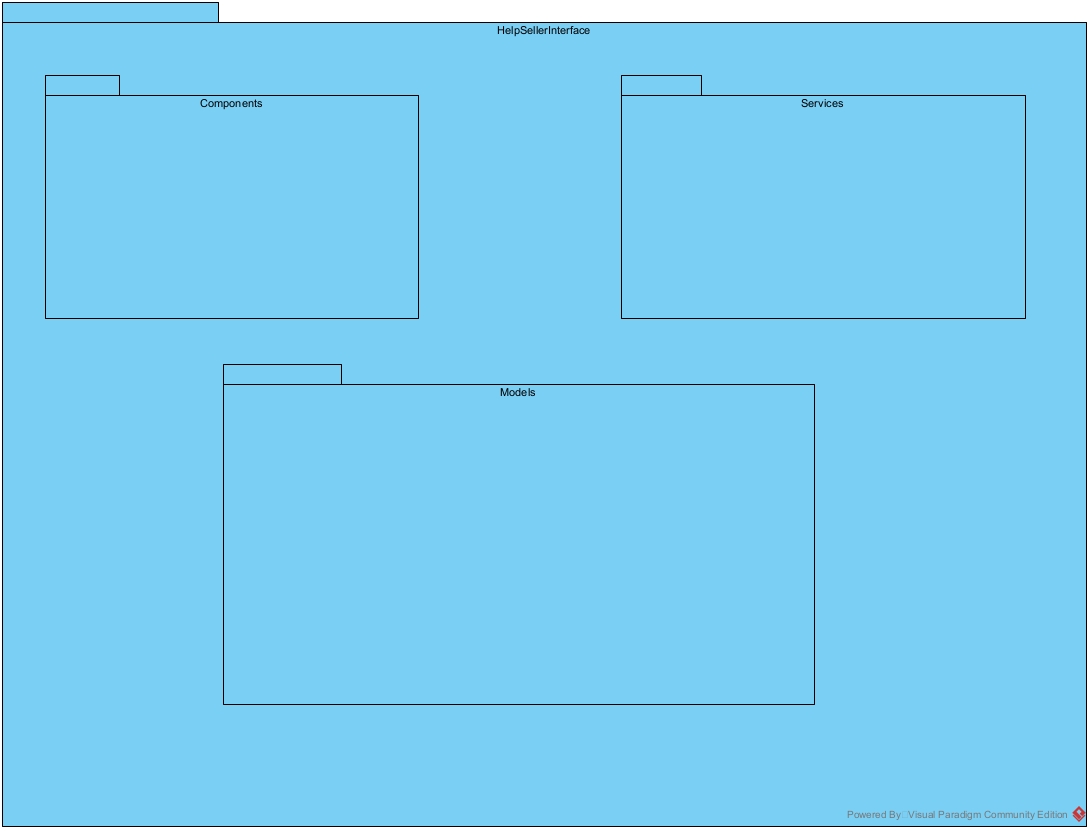


**2.1.4 Package Service**



## 2.2 HelpSellerInterface

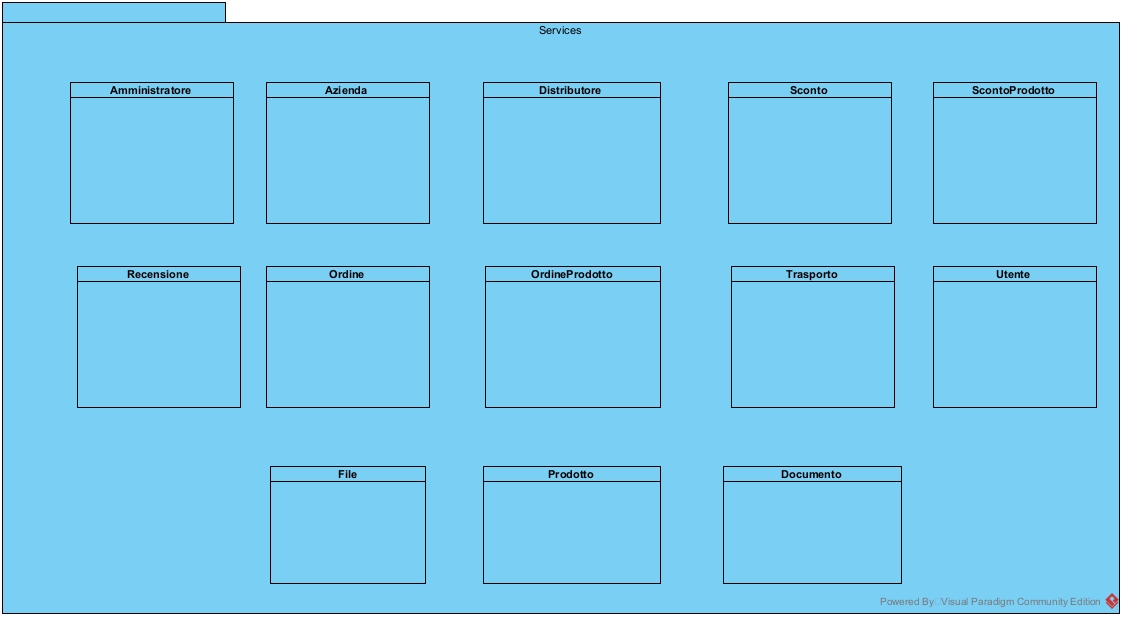
Nella presente sezione si mostra la struttura del package principale di HelpSellerInterface. La struttura generale è stata ottenuta suddividendo i components dai servizi e dai model

****

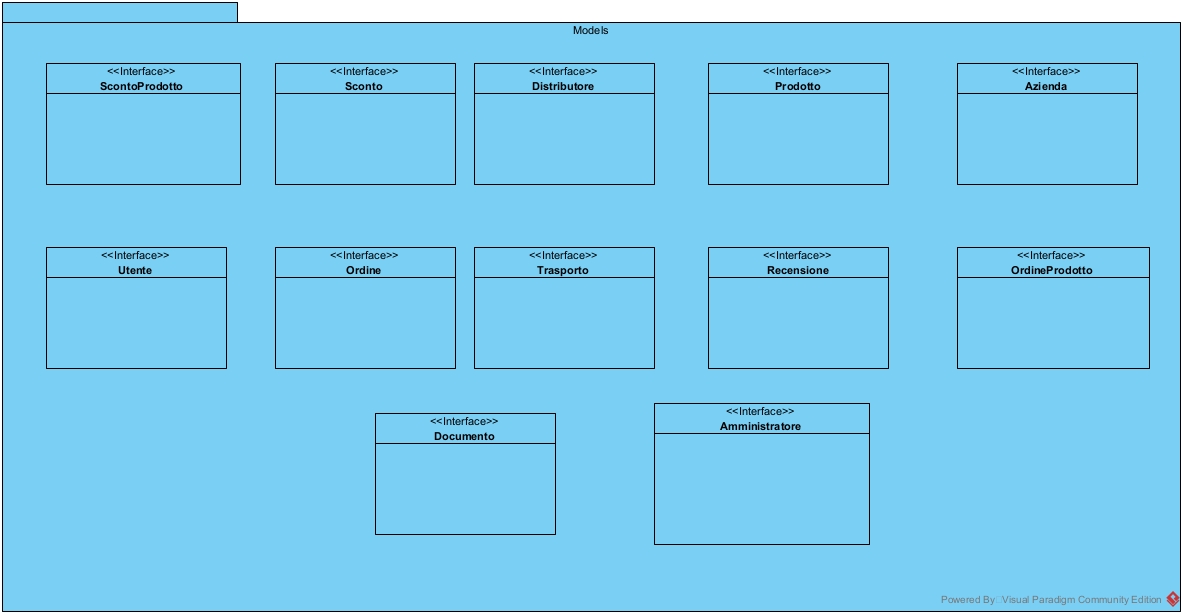
**2.2.1 Components**

****

**2.2.2 Services**

****

**2.2.2 Models**

****

# 3 Interfacce delle classi

## 3.1 AmministratoreController

| **NOME CLASSE** | AmministratoreController |
| --- | --- |
| **DESCRIZIONE** | Questa classe fornisce i servizi per l’Amministratore |
| **METODI** | +findAll(): List<Amministratore>  +findId(int i): Amministratore  +update(AmministratoreModel amministratore) : AmministratoreModel |

| **NOME CLASSE** | AmministratoreController |
| --- | --- |
| **NOME METODO** | +findAll(): List<Amministratore> |
| **DESCRIZIONE** | Questo metodo restituisce la lista di amministratori |
| **PRE-CONDIZIONI** | context: Amministratore :: findAll()  pre:/ |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | AmministratoreController |
| --- | --- |
| **NOME METODO** | +findId(int i): Amministratore |
| **DESCRIZIONE** | Questo metodo restituisce l’amministratore con id=i |
| **PRE-CONDIZIONI** | context: Amministratore :: findId(int i)  pre: i >0 |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | AmministratoreController |
| --- | --- |
| **NOME METODO** | +update(AmministratoreModel amministratore) : AmministratoreModel |
| **DESCRIZIONE** | Questo metodo modifica il profilo dell’amministratore |
| **PRE-CONDIZIONI** | context: Amministratore :: update(AmministratoreModel amministratore)  pre: Amministratore != null |
| **POST-CONDIZIONI** | / |

## 3.2 AziendaController

| **NOME CLASSE** | AziendaController |
| --- | --- |
| **DESCRIZIONE** | Questa classe fornisce i servizi per l’azienda |
| **METODI** | +findAll(): List<Azienda>  +findId(int i): Azienda  +deleteId(int i):boolean  +insert(AziendaModel a) : integer  +update(AziendaModel a) : integer  +findAziendeByNome(String name) : List<AziendaModel>  +findAziendaByProdotto(int id) : ResponseEntity<AziendaModel> |

| **NOME CLASSE** | AziendaController |
| --- | --- |
| **NOME METODO** | +findAll(): List<Azienda> |
| **DESCRIZIONE** | Questo metodo trova tutte le aziende |
| **PRE-CONDIZIONI** | context: Azienda :: findAll() |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | AziendaController |
| --- | --- |
| **NOME METODO** | +findId(int i): Azienda |
| **DESCRIZIONE** | Questo metodo trova l’azienda con l’id=i |
| **PRE-CONDIZIONI** | context: Azienda :: findId(int i) pre: i>0 |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | AziendaController |
| --- | --- |
| **NOME METODO** | +deleteId(int i): boolean |
| **DESCRIZIONE** | Questo metodo elimina l’azienda con l’id=i |
| **PRE-CONDIZIONI** | context: Azienda :: deleteId(int i) pre: i > 0 |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | AziendaController |
| --- | --- |
| **NOME METODO** | +insert(AziendaModel a) : integer |
| **DESCRIZIONE** | Questo metodo aggiunge l’azienda a nel database |
| **PRE-CONDIZIONI** | context: Azienda :: insert(AziendaModel a) pre: a != null |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | AziendaController |
| --- | --- |
| **NOME METODO** | +update(AziendaModel a) : integer |
| **DESCRIZIONE** | Questo metodo modifica l’azienda a |
| **PRE-CONDIZIONI** | context: Azienda :: update(AziendaModel a)  pre: a!= null |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | AziendaController |
| --- | --- |
| **NOME METODO** | +findAziendeByNome(String nome) : integer |
| **DESCRIZIONE** | Questo metodo restituisce tutte le aziende che hanno un nome pari al nome passato come parametro |
| **PRE-CONDIZIONI** | context: Azienda :: findAziendeByNome(String nome) pre: nome != null |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | AziendaController |
| --- | --- |
| **NOME METODO** | +findAziendeByProdotto(int id) : integer |
| **DESCRIZIONE** | Questo metodo restituisce l’azienda che vende un prodotto con id pari all’id passato come parametro |
| **PRE-CONDIZIONI** | context: Azienda :: findAziendeByProdotto(int id)  pre: id > 0 |
| **POST-CONDIZIONI** | / |

## 3.3 DistributoreController

| **NOME CLASSE** | DistributoreController |
| --- | --- |
| **DESCRIZIONE** | Questa classe fornisce i servizi per il Distributore |
| **METODI** | +findAll(): List<Distributore>  +findId(int i): Distributore  +deleteId(int i):boolean  +insert(DistributoreModel a) : integer  +update(DistributoreModel a) : integer |

| **NOME CLASSE** | DistributoreController |
| --- | --- |
| **NOME METODO** | +findAll(): List<Distributore> |
| **DESCRIZIONE** | Questo metodo trova tutti i distributori |
| **PRE-CONDIZIONI** | context: Distributore :: findAll() |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | DistributoreController |
| --- | --- |
| **NOME METODO** | +findId(int i): Distributore |
| **DESCRIZIONE** | Questo metodo trova il Distributore con id=i |
| **PRE-CONDIZIONI** | context: Distributore :: findId(int i)  pre: i > 0 |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | DistributoreController |
| --- | --- |
| **NOME METODO** | +deleteId(int i): boolean |
| **DESCRIZIONE** | Questo metodo elimina il Distributore con id=i |
| **PRE-CONDIZIONI** | context: Distributore :: deleteId(int i)  pre: i > 0 |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | DistributoreController |
| --- | --- |
| **NOME METODO** | +insert(DistributoreModel d) : integer |
| **DESCRIZIONE** | Questo metodo aggiunge il Distributore d nel database |
| **PRE-CONDIZIONI** | context: Distributore :: insert(DistributoreModel a)  pre: a!=null |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | DistributoreController |
| --- | --- |
| **NOME METODO** | +update(DistributoreModel d) : integer |
| **DESCRIZIONE** | Questo metodo modifica il distributore d |
| **PRE-CONDIZIONI** | context: Distributore :: update(DistributoreModel a) pre: a!=null |
| **POST-CONDIZIONI** | / |

## 3.4 DocumentoController

| **NOME CLASSE** | DocumentoController |
| --- | --- |
| **DESCRIZIONE** | Questa classe fornisce i servizi per Documento |
| **METODI** | +findAll(): List<DocumentoModel>  +findId(int i): DocumentoModel  +deleteId(int i):boolean  +insert(DocumentoModel d) : integer  +update(DocumentoModel d) : integer  +findDocumentiByOrdine(int id) : List<DocumentoModel> |

| **NOME CLASSE** | DocumentoController |
| --- | --- |
| **NOME METODO** | +findAll(): List<DocumentoModel> |
| **DESCRIZIONE** | Questo metodo trova tutti i documenti |
| **PRE-CONDIZIONI** | context: Documento :: findAll() |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | DocumentoController |
| --- | --- |
| **NOME METODO** | +findId(int i): DocumentoModel |
| **DESCRIZIONE** | Questo metodo trova il documento con id=i |
| **PRE-CONDIZIONI** | context: Documento :: findId(int i)  pre: i > 0 |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | DocumentoController |
| --- | --- |
| **NOME METODO** | +deleteId(int i): boolean |
| **DESCRIZIONE** | Questo metodo elimina il documento con id=i |
| **PRE-CONDIZIONI** | context: Documento :: deleteId(int i)  pre: i >0 |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | DocumentoController |
| --- | --- |
| **NOME METODO** | +insert(DocumentoModel d) : integer |
| **DESCRIZIONE** | Questo metodo aggiunge il Documento d nel database |
| **PRE-CONDIZIONI** | context: Documento :: insert(DocumentoModel d) pre: d != null |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | DocumentoController |
| --- | --- |
| **NOME METODO** | +update(DocumentoModel d) : integer |
| **DESCRIZIONE** | Questo metodo modifica il Documento d |
| **PRE-CONDIZIONI** | context: Documento :: update(DocumentoModel d) pre: d != null |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | DocumentoController |
| --- | --- |
| **NOME METODO** | +findDocumentiByOrdine(int id) : List<DocumentoModel> |
| **DESCRIZIONE** | Questo metodo restituisce i documenti di un ordine avente id pari all’id passato come parametro |
| **PRE-CONDIZIONI** | context: Documento :: findDocumentiByOrdine(int id) pre: id > 0 |
| **POST-CONDIZIONI** | / |

## 3.5 FileController

| **NOME CLASSE** | FileController |
| --- | --- |
| **DESCRIZIONE** | Questa classe fornisce i servizi per i file |
| **METODI** | +uploadFiles (List<MultipartFile> multipartFiles) : List<String>  +downloadFiles (String filename) : Resource  +getImageWithMediaType (String fileName) : Byte[] |

| **NOME CLASSE** | FileController |
| --- | --- |
| **NOME METODO** | +uploadFiles (List<MultipartFile> multipartFiles) : List<String> |
| **DESCRIZIONE** | Questo metodo permette di caricare file |
| **PRE-CONDIZIONI** | context: File :: uploadFiles (List<MultipartFile> multipartFiles)  pre: multipartFiles != null |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | FileController |
| --- | --- |
| **NOME METODO** | +downloadFiles (List<MultipartFile> multipartFiles) : List<String> |
| **DESCRIZIONE** | Questo metodo permette di scaricare file |
| **PRE-CONDIZIONI** | context: File :: downloadFiles(List<MultipartFile> multipartFiles)  pre: multipartFiles != null |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | FileController |
| --- | --- |
| **NOME METODO** | +getImageWithMediaType (String fileName) : Byte[] |
| **DESCRIZIONE** | Questo metodo permette di ottenere un’immagine dal nome passato come parametro |
| **PRE-CONDIZIONI** | context: File :: fgetImageWithMediaType (String fileName)  pre: fileName != null |
| **POST-CONDIZIONI** | / |

## 3.6 OrdineController

| **NOME CLASSE** | OrdineController |
| --- | --- |
| **DESCRIZIONE** | Questa classe fornisce i servizi per Ordine |
| **METODI** | +findAll(): List<Ordine>  +findId(int i): Ordine  +insert(OrdineModel ord) : integer  +findOrdiniByDistributore(int id) : List<OrdineModel>  +findOrdiniByAzienda(int id) : List<OrdineModel> |

| **NOME CLASSE** | OrdineController |
| --- | --- |
| **NOME METODO** | +findAll(): List<Ordine> |
| **DESCRIZIONE** | Questo metodo trova tutti gli Ordini |
| **PRE-CONDIZIONI** | context: Ordine :: findAll() |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | OrdineController |
| --- | --- |
| **NOME METODO** | +findId(int i): Ordine |
| **DESCRIZIONE** | Questo metodo trova l’Ordine con id=i |
| **PRE-CONDIZIONI** | context: Ordine :: findId(int i) pre: i >0 |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | OrdineController |
| --- | --- |
| **NOME METODO** | +insert(OrdineModel ord) : integer |
| **DESCRIZIONE** | Questo metodo aggiunge l’Ordine ord nel database |
| **PRE-CONDIZIONI** | context: Ordine :: insert(OrdineModel ord) pre:  ord != null |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | OrdineController |
| --- | --- |
| **NOME METODO** | +findOrdiniByDistributore(int id) : List<OrdiniModel> |
| **DESCRIZIONE** | Questo metodo restituisce gli ordini effettuati da un distributore avente id pari all’id passato come parametro |
| **PRE-CONDIZIONI** | context: Ordine :: findOrdiniByDistributore(int id) pre:  id > 0 |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | OrdineController |
| --- | --- |
| **NOME METODO** | +findOrdiniByAzienda(int id) : List<OrdiniModel> |
| **DESCRIZIONE** | Questo metodo restituisce gli ordini effettuati da un’azienda avente id pari all’id passato come parametro |
| **PRE-CONDIZIONI** | context: Ordine :: findOrdiniByAzienda(int id) pre:  id > 0 |
| **POST-CONDIZIONI** | / |

## 3.7 OrdineProdottoController

| **NOME CLASSE** | OrdineProdottoController |
| --- | --- |
| **DESCRIZIONE** | Questa classe fornisce i servizi per OrdineProdotto |
| **METODI** | +findAll(): List<OrdineProdotto>  +findId(int i): OrdineProdotto  +deleteId(int i):boolean  +insert(OrdineProdottoModel ord) : integer  +update(OrdineProdottoModel ord) : integer  +findDettagliOrdine(int id) : List<OrdineProdottoModel>  +findReportAnnuale(int anno) : integer  +findReportAnnualeAzienda(int anno, int id): integer  +findReportMensileGruppo (Integer anno) : Integer[]  +findReportMensile (Integer anno, Integer id) : Integer[] |

| **NOME CLASSE** | OrdineProdottoController |
| --- | --- |
| **NOME METODO** | +findAll(): List<OrdineProdotto> |
| **DESCRIZIONE** | Questo metodo trova tutti gli “OrdineProdotto” |
| **PRE-CONDIZIONI** | context: OrdineProdotto :: findAll() |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | OrdineProdottoController |
| --- | --- |
| **NOME METODO** | +findId(int i): OrdineProdotto |
| **DESCRIZIONE** | Questo metodo trova l’OrdineProdotto con id=i |
| **PRE-CONDIZIONI** | context: OrdineProdotto :: findId() pre: i >0 |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | OrdineProdottoController |
| --- | --- |
| **NOME METODO** | +deleteId(int i): boolean |
| **DESCRIZIONE** | Questo metodo elimina l’OrdineProdotto con id=i |
| **PRE-CONDIZIONI** | context: OrdineProdotto :: findId() pre: i > 0 |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | OrdineProdottoController |
| --- | --- |
| **NOME METODO** | +insert(OrdineProdotto ordprod) : integer |
| **DESCRIZIONE** | Questo metodo associa il prodotto selezionato all’ ordine nel database insieme alla quantità e al prezzo unitario |
| **PRE-CONDIZIONI** | context: Ordine :: insert(OrdineProdotto ord) pre:  ord != null |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | OrdineProdottoController |
| --- | --- |
| **NOME METODO** | +update(OrdineProdotto ord) : integer |
| **DESCRIZIONE** | Questo metodo modifica i prodotti associati all’ordine insieme alla quantità e al prezzo unitario |
| **PRE-CONDIZIONI** | context: Ordine :: update(OrdineProdotto ord) pre:  ord != null |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | OrdineProdottoController |
| --- | --- |
| **NOME METODO** | +findDettagliOrdine(int id) : List<OrdineProdottoModel> |
| **DESCRIZIONE** | Questo metodo restituisce i dettagli di un ordine con id pari all’id passato come parametro |
| **PRE-CONDIZIONI** | context: OrdineProdotto :: findDettagliOrdine(int id) pre:  id > 0 |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | OrdineProdottoController |
| --- | --- |
| **NOME METODO** | +findReportAnnuale(int anno) : integer |
| **DESCRIZIONE** | Questo metodo restituisce il report annuale di un azienda dell’anno pari all’anno passato come parametro |
| **PRE-CONDIZIONI** | context: OrdineProdotto :: findReportAnnuale(int anno)  pre: anno > 0 |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | OrdineProdottoController |
| --- | --- |
| **NOME METODO** | +findReportAnnualeAzienda(int id): integer |
| **DESCRIZIONE** | Questo metodo restituisce il report annuale di un’azienda = id |
| **PRE-CONDIZIONI** | context: OrdineProdotto :: findReportAnnuale(int anno, int id) pre:  id >0 && anno >0 |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | OrdineProdottoController |
| --- | --- |
| **NOME METODO** | +findReportMensileGruppo (Integer anno) : Integer[] |
| **DESCRIZIONE** | Questo metodo restituisce il report mensile del gruppo Unilever riferito all’anno passato come parametro |
| **PRE-CONDIZIONI** | context: OrdineProdotto :: findReportMensileGruppo(int anno)  pre: anno >0 |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | OrdineProdottoController |
| --- | --- |
| **NOME METODO** | +findReportMensile (Integer anno, Integer id) : Integer[] |
| **DESCRIZIONE** | Questo metodo restituisce il report mensile di un'azienda con id pari all’id passato come parametro e con anno pari all’anno passato come parametro |
| **PRE-CONDIZIONI** | context: OrdineProdotto :: findReportMensile(int anno, int id) pre:  id >0 && anno >0 |
| **POST-CONDIZIONI** | / |

## 3.8 ProdottoController

| **NOME CLASSE** | ProdottoController |
| --- | --- |
| **DESCRIZIONE** | Questa classe fornisce i servizi per Prodotto |
| **METODI** | +findAll(): List<Prodotto>  +findId(int i): Prodotto  +deleteId(int i):boolean  +insert(ProdottoModel prod) : Integer  +update(ProdottoModel prod) : Integer  +findProdottiByAzienda(int id) : List<ProdottoModel>  +findProdottiByNome(String name) :List<ProdottoModel>  +findProdottiByNomeInAzienda(String name, int id) :List<ProdottoModel>  +findProdottiInOrdine(int id) : List<ProdottoModel>  +findProdottiInSconto( int id) : List<ProdottoModel>  +findProdottiByIdInAzienda( int id\_prodotto, int id\_azienda)  : List<ProdottoModel> |

| **NOME CLASSE** | ProdottoController |
| --- | --- |
| **NOME METODO** | +findAll(): List<Prodotto> |
| **DESCRIZIONE** | Questo metodo trova tutti i Prodotti |
| **PRE-CONDIZIONI** | context: Prodotto :: findAll() |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | ProdottoController |
| --- | --- |
| **NOME METODO** | +findId(int i): Prodotto |
| **DESCRIZIONE** | Questo metodo trova il Prodotto con id=i |
| **PRE-CONDIZIONI** | context: Prodotto :: findId(int i) pre: i>0 |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | ProdottoController |
| --- | --- |
| **NOME METODO** | +deleteId(int i): boolean |
| **DESCRIZIONE** | Questo metodo elimina il Prodotto con id=i |
| **PRE-CONDIZIONI** | context: Prodotto :: deleteId(int i) pre: i>0 |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | ProdottoController |
| --- | --- |
| **NOME METODO** | +insert(ProdottoModel prod) : Integer |
| **DESCRIZIONE** | Questo metodo inserisce il Prodotto prod nel database |
| **PRE-CONDIZIONI** | context: Prodotto :: insert(ProdottoModel prod) pre:  prod != null |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | ProdottoController |
| --- | --- |
| **NOME METODO** | +update(ProdottoModel prod) : Integer |
| **DESCRIZIONE** | Questo metodo modifica il Prodotto prod nel database |
| **PRE-CONDIZIONI** | context: Prodotto :: update(ProdottoModel prod) pre:  prod != null |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | ProdottoController |
| --- | --- |
| **NOME METODO** | +findProdottiByAzienda(int id) : List<ProdottoModel> |
| **DESCRIZIONE** | Questo metodo restituisce tutti i prodotti di un’azienda con id pari all’id passato come parametro |
| **PRE-CONDIZIONI** | context: Prodotto :: findProdottiByAzienda(int id) pre:  id > 0 |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | ProdottoController |
| --- | --- |
| **NOME METODO** | +findProdottiByNome(String name) :List<ProdottoModel> |
| **DESCRIZIONE** | Questo metodo restituisce tutti i prodotti con nome pari all’nome passato come parametro |
| **PRE-CONDIZIONI** | context: Prodotto :: findProdottiByNome(String name) pre:  name!= null |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | ProdottoController |
| --- | --- |
| **NOME METODO** | +findProdottiByNomeInAzienda(int id\_prodotto, int id\_azienda) :List<ProdottoModel> |
| **DESCRIZIONE** | Questo metodo restituisce tutti i prodotti di un’azienda= id\_azienda che hanno codice prodotto compreso nel valore id\_prodotto passato come parametro |
| **PRE-CONDIZIONI** | context: Prodotto :: findProdottiByNomeInAzienda(int id\_prodotto, int id\_azienda)  pre: id\_prodotto > 0 && id\_azienda >0 |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | ProdottoController |
| --- | --- |
| **NOME METODO** | +findProdottiInOrdine(int id) :List<ProdottoModel> |
| **DESCRIZIONE** | Questo metodo restituisce tutti i prodotti di un ordine = id |
| **PRE-CONDIZIONI** | context: Prodotto findProdottInOrdine(int id) pre:  id>0 |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | ProdottoController |
| --- | --- |
| **NOME METODO** | +findProdottiInSconto(int id) :List<ProdottoModel> |
| **DESCRIZIONE** | Questo metodo restituisce tutti i prodotti con uno sconto = id |
| **PRE-CONDIZIONI** | context: Prodotto :: findProdottInSconto(int id) pre:  id>0 |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | ProdottoController |
| --- | --- |
| **NOME METODO** | +findProdottiInSconto(int id) :List<ProdottoModel> |
| **DESCRIZIONE** | Questo metodo restituisce tutti i prodotti con uno sconto = id |
| **PRE-CONDIZIONI** | context: Prodotto :: findProdottInSconto(int id) pre:  id>0 |
| **POST-CONDIZIONI** | / |

## 

## 3.9 RecensioneController

| **NOME CLASSE** | RecensioneController |
| --- | --- |
| **DESCRIZIONE** | Questa classe fornisce i metodi per le Recensioni |
| **METODI** | +findAll(): List<Recensione>  +findId(int i) : Recensione  +deleteId(int i) : boolean  +insert(RecensioneModel rec) : Integer  +update(RecensioneModel rec) : Integer  +findRecensioniByProdotto(int id) : List<RecensioneModel> |

| **NOME CLASSE** | Recensione |
| --- | --- |
| **NOME METODO** | +findAll(): List<Recensione> |
| **DESCRIZIONE** | Questo metodo trova tutte le Recensioni |
| **PRE-CONDIZIONI** | context: Recensione :: findAll() |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | Recensione |
| --- | --- |
| **NOME METODO** | +findId(int i): Recensione |
| **DESCRIZIONE** | Questo metodo trova la Recensione con id=i |
| **PRE-CONDIZIONI** | context: Recensione :: findId(int i)  pre: i>0 |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | Recensione |
| --- | --- |
| **NOME METODO** | +deleteId(int i): boolean |
| **DESCRIZIONE** | Questo metodo elimina la Recensione con id=i |
| **PRE-CONDIZIONI** | context: Recensione :: deleteId(int i) pre: i > 0 |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | RecensioneController |
| --- | --- |
| **NOME METODO** | +insert(RecensioneModel rec) : Integer |
| **DESCRIZIONE** | Questo metodo inserisce la recensione rec nel database |
| **PRE-CONDIZIONI** | context: Recensione :: insert(RecensioneModel rec) pre:  rec != null |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | RecensioneController |
| --- | --- |
| **NOME METODO** | +update(RecensioneModel rec) : Integer |
| **DESCRIZIONE** | Questo metodo modifica la recensione rec nel database |
| **PRE-CONDIZIONI** | context: Recensione :: update(RecensioneModel rec) pre:  rec != null |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | RecensioneController |
| --- | --- |
| **NOME METODO** | +findRecensioniByProdotto(int id): List<RecensioneModel> |
| **DESCRIZIONE** | Questo metodo restituisce tutte le recensioni di un prodotto= id |
| **PRE-CONDIZIONI** | context: Prodotto :: findRecensioniByProdotto(int id): pre:  id > 0 |
| **POST-CONDIZIONI** | / |

## 3.10 ScontoController

| **NOME CLASSE** | ScontoController |
| --- | --- |
| **DESCRIZIONE** | Questa classe fornisce i servizi per Sconto |
| **METODI** | +findAll(): List<Sconto>  +findId(int i): Sconto  +deleteId(int i):boolean  +insert(ScontoModel sc) : Integer  +update(ScontoModel rec) : Integer  +findScontiByAzienda(int id) : List<ScontoModel>  +findScontiByTipo(String tipo) : List<ScontoModel>  +findScontiAziendaByTipo(String tipo, int id) : List<ScontoModel>  +findScontiByNomeInAzienda(String nome\_sconto, int id) : List<ScontoModel> |

| **NOME CLASSE** | ScontoController |
| --- | --- |
| **NOME METODO** | +findAll(): List<Sconto> |
| **DESCRIZIONE** | Questo metodo trova tutti gli Sconti |
| **PRE-CONDIZIONI** | context: Sconto :: findAll() |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | ScontoController |
| --- | --- |
| **NOME METODO** | +findId(int i): Sconto |
| **DESCRIZIONE** | Questo metodo trova lo Sconto con id=i |
| **PRE-CONDIZIONI** | context: Sconto :: findId(int i) pre: i >0 |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | ScontoController |
| --- | --- |
| **NOME METODO** | +deleteId(int i): boolean |
| **DESCRIZIONE** | Questo metodo elimina lo Sconto con id=i |
| **PRE-CONDIZIONI** | context: Sconto :: deleteId(int i) pre: i >0 |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | ScontoController |
| --- | --- |
| **NOME METODO** | +insert(ScontoModel sc) : Integer |
| **DESCRIZIONE** | Questo metodo inserisce lo sconto sc nel database |
| **PRE-CONDIZIONI** | context: Prodotto :: insert(ScontoModel sc) pre:  sc != null |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | ScontoController |
| --- | --- |
| **NOME METODO** | +update(ScontoModel sc) : Integer |
| **DESCRIZIONE** | Questo metodo modifica lo sconto sc nel database |
| **PRE-CONDIZIONI** | context: Sconto :: update(ScontoModel sc) pre:  sc != null |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | ScontoController |
| --- | --- |
| **NOME METODO** | +findScontiByAzienda(int id) : List<ScontoModel> |
| **DESCRIZIONE** | Questo metodo restituisce una lista di sconti di un’azienda = id |
| **PRE-CONDIZIONI** | context: Sconto :: findScontiByAzienda(int id) pre:  id >0 |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | ScontoController |
| --- | --- |
| **NOME METODO** | +findScontiByTipo(String tipo) : List<ScontoModel> |
| **DESCRIZIONE** | Questo metodo restituisce una lista di sconti di un tipo pari al tipo passato come parametro |
| **PRE-CONDIZIONI** | context: Sconto :: findScontiByTipo(String tipo) pre:  tipo!= null |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | ScontoController |
| --- | --- |
| **NOME METODO** | +findScontiAziendaByTipo(String tipo, int id) : List<ScontoModel> |
| **DESCRIZIONE** | Questo metodo restituisce una lista di sconti di un tipo pari al tipo passato come parametro di un’azienda = id |
| **PRE-CONDIZIONI** | context: Sconto :: findScontiAziendaByTipo(String tipo, int id) pre:  id >0 && tipo!=null |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | ScontoController |
| --- | --- |
| **NOME METODO** | +findScontiByNomeInAzienda(String nome\_sconto, int id) : List<ScontoModel> |
| **DESCRIZIONE** | Questo metodo restituisce una lista di sconti di un nome= nome\_sconto e di un’azienda = id |
| **PRE-CONDIZIONI** | context: Sconto :: findScontiByNomeInAzienda(String nome\_sconto, int id)  pre: nome\_sconto!= null && id >0 |
| **POST-CONDIZIONI** | / |

## 3.11 ScontoProdottoController

| **NOME CLASSE** | ScontoProdottoController |
| --- | --- |
| **DESCRIZIONE** | Questa classe fornisce i servizi per ScontoProdotto |
| **METODI** | +findAllScontoProdotto(): List<ScontoProdotto>  +findId(int i) : ScontoProdotto  +deleteId(int i) : boolean  +insert() : Integer  +update() : Integer  +findBySconto(int id) : List<ScontoProdottoModel>  +findByProdotto(int id) : <List<ScontoProdottoModel> |

| **NOME CLASSE** | ScontoProdottoController |
| --- | --- |
| **NOME METODO** | +findAllScontoProdotto() : List<ScontoProdotto> |
| **DESCRIZIONE** | Questo metodo trova tutti gli ScontiProdotto |
| **PRE-CONDIZIONI** | context: ScontoProdotto :: findAll() |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | ScontoProdottoController |
| --- | --- |
| **NOME METODO** | +findId(int i) : ScontoProdotto |
| **DESCRIZIONE** | Questo metodo trova lo ScontoProdotto con id=i |
| **PRE-CONDIZIONI** | context: ScontoProdotto :: findId(int i) pre: i >0 |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | ScontoProdottoController |
| --- | --- |
| **NOME METODO** | +deleteId(int i) : ScontoProdotto |
| **DESCRIZIONE** | Questo metodo elimina un’associazione Sconto-Prodotto con id=i |
| **PRE-CONDIZIONI** | context: ScontoProdotto :: deleteId(int i) pre: i >0 |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | ScontoProdottoController |
| --- | --- |
| **NOME METODO** | +insert(ScontoProdottoModel sc) : Integer |
| **DESCRIZIONE** | Questo metodo associa un prodotto ad uno sconto sc o viceversa nel database |
| **PRE-CONDIZIONI** | context: Prodotto :: insert(ScontoProdottoModel sc) pre:  sc != null |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | ScontoProdottoController |
| --- | --- |
| **NOME METODO** | +update(ScontoProdottoModel sc) : Integer |
| **DESCRIZIONE** | Questo metodo modifica l’associazione prodotto-sconto sc nel database |
| **PRE-CONDIZIONI** | context: Sconto :: update(ScontoProdottoModel sc) pre:  sc != null |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | ScontoProdottoController |
| --- | --- |
| **NOME METODO** | +findBySconto(int id) : List<ScontoProdottoModel> |
| **DESCRIZIONE** | Questo metodo restituisce i prodotti con uno sconto=id |
| **PRE-CONDIZIONI** | context: Sconto :: findBySconto(int id) pre:  id >0 |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | ScontoProdottoController |
| --- | --- |
| **NOME METODO** | +findByProdotto(int id) : <List<ScontoProdottoModel> |
| **DESCRIZIONE** | Questo metodo restituisce gli sconti associati ad un prodotto=id |
| **PRE-CONDIZIONI** | context: ScontoProdotto :: findByProdotto(int id) pre:  id >0 |
| **POST-CONDIZIONI** | / |

## 3.12 TrasportoController

| **NOME CLASSE** | TrasportoController |
| --- | --- |
| **DESCRIZIONE** | Questa classe fornisce i servizi per Trasporto |
| **METODI** | +findAll(): List<Trasporto>  +findId(int i): Trasporto  +deleteId(int i):boolean  +insert() : Integer  +update() : Integer |

| **NOME CLASSE** | TrasportoController |
| --- | --- |
| **NOME METODO** | +findAll(): List<Trasporto> |
| **DESCRIZIONE** | Questo metodo trova tutti i Trasporti |
| **PRE-CONDIZIONI** | context: Trasporto :: findAll() |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | TrasportoController |
| --- | --- |
| **NOME METODO** | +findId(int i): Trasporto |
| **DESCRIZIONE** | Questo metodo trova il Trasporto con id=i |
| **PRE-CONDIZIONI** | context: Trasporto :: findId(int i) pre: i >0 |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | TrasportoController |
| --- | --- |
| **NOME METODO** | +deleteId(int i): Trasporto |
| **DESCRIZIONE** | Questo metodo elimina il Trasporto con id=i |
| **PRE-CONDIZIONI** | context: Trasporto :: deleteId(int i) pre: i >0 |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | TrasportoController |
| --- | --- |
| **NOME METODO** | +insert(TrasportoModel tr) : Integer |
| **DESCRIZIONE** | Questo metodo aggiunge una gestione pratica trasporto nel database |
| **PRE-CONDIZIONI** | context: Prodotto :: insert(TrasportoModel tr) pre:  tr!= null |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | TrasportoController |
| --- | --- |
| **NOME METODO** | +update(TrasportoModel tr) : Integer |
| **DESCRIZIONE** | Questo metodo modifica una gestione pratica trasporto nel database |
| **PRE-CONDIZIONI** | context: Sconto :: update(TrasportoModel tr) pre:  tr != null |
| **POST-CONDIZIONI** | / |

## 3.13 Utente Controller

| **NOME CLASSE** | UtenteController |
| --- | --- |
| **DESCRIZIONE** | Questa classe fornisce i servizi per l’autenticazione e la registrazione degli users |
| **METODI** | +auth(String email, String password, String tipo) : UtenteModel  +recuperoPassword( String email) : Integer |

| **NOME CLASSE** | Utente Controller |
| --- | --- |
| **NOME METODO** | auth(String email, String password, String tipo): UtenteModel |
| **DESCRIZIONE** | Questo metodo controlla che i dati inseriti lato frontend per il login siano validi o meno |
| **PRE-CONDIZIONI** | context: Utente :: auth(email, password, tipo) pre: email != null && password != null && tipo == ‘Amministratore’ || ‘Distributore’ || ‘Azienda’ |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | Utente Controller |
| --- | --- |
| **NOME METODO** | recuperoPassword(String email) : integer |
| **DESCRIZIONE** | Questo metodo si occupa inviare email per il recupero password |
| **PRE-CONDIZIONI** | context: Utente :: recuperoPassword(email, password, tipo) pre: email != null && password!=null && tipo != null |
| **POST-CONDIZIONI** | / |

## 3.14 Raccomandazione Controller

| **NOME CLASSE** | RaccomandazioneController |
| --- | --- |
| **DESCRIZIONE** | classe di mappatura dei servizi relativa al Raccomandation Engine affinché siano accessibili dal frontend |
| **METODI** | +creaCsv() : void  +firstLayer() : ProdottoModel[]  +secondLayer(int id) : ProdottoModel[] |

| **NOME CLASSE** | RaccomandazioneController |
| --- | --- |
| **NOME METODO** | creaCsv():void |
| **DESCRIZIONE** | Metodo per la generazione di un file .csv contenente le recensioni presenti nel db |
| **PRE-CONDIZIONI** | context: Recensione :: creaCsv() |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | RaccomandazioneController |
| --- | --- |
| **NOME METODO** | firstLayer() : ProdottoModel[] |
| **DESCRIZIONE** | metodo per il recupero dei prodotti più popolari sulla piattaforma |
| **PRE-CONDIZIONI** | context: Prodotto :: firstLayer() |
| **POST-CONDIZIONI** | / |

| **NOME CLASSE** | RaccomandazioneController |
| --- | --- |
| **NOME METODO** | secondLayer(int id) : ProdottoModel[] |
| **DESCRIZIONE** | metodo per il recupero dei prodotti da suggerire all'utente |
| **PRE-CONDIZIONI** | context: Prodotto :: secondLayer(int id)  pre: id>0 |
| **POST-CONDIZIONI** | / |

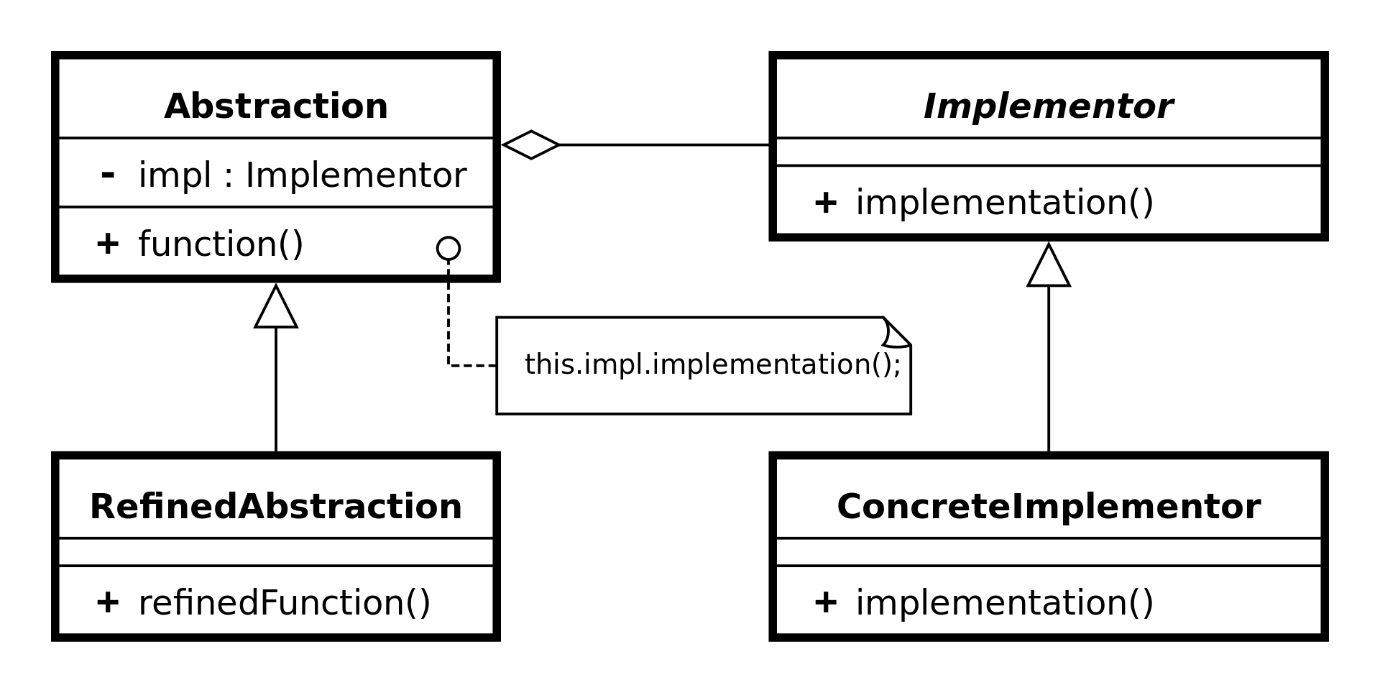
# 4 Design patterns

“Ogni pattern descrive un problema che ricorre più volte nel nostro ambiente, descrive poi il nucleo della soluzione del problema in modo da poter utilizzare tale soluzione un milione di volte senza mai farlo allo stesso modo”

Nell’object oriented, i design pattern sono template di soluzioni utilizzabili per risolvere un insieme di problemi ricorrenti. I design pattern vengono classificati in tre famiglie:

* I PATTERN STRUTTURALI consentono di riutilizzare degli oggetti esistenti fornendo agli utilizzatori un'interfaccia più adatta alle loro esigenze.
* I PATTERN COMPORTAMENTALI forniscono soluzione alle più comuni tipologie di interazione tra gli oggetti.
* I PATTERN CREAZIONALI nascondono i costruttori delle classi e mettono dei metodi al loro posto creando un'interfaccia. In questo modo si possono utilizzare oggetti senza sapere come sono implementati.

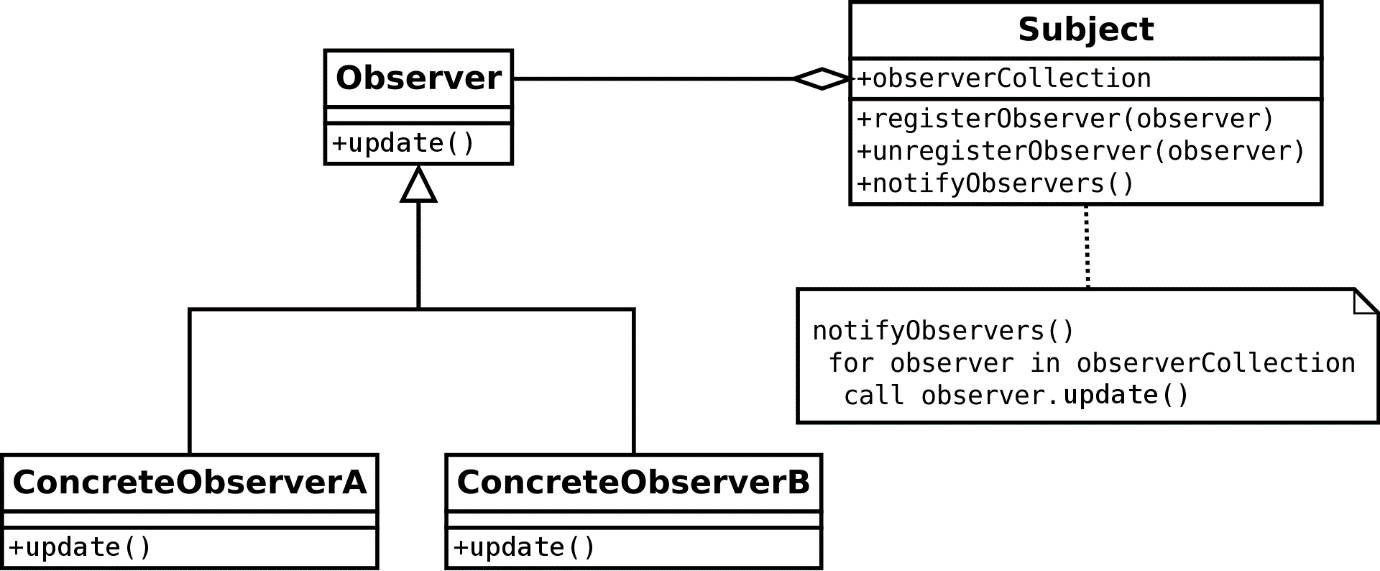
## 4.1 **Bridge Design Pattern**:



Tale pattern permette di separare l'interfaccia di una classe (che cosa si può fare con la classe) dalla sua implementazione (come lo fa). In tal modo si può usare l'ereditarietà per fare evolvere l'interfaccia o l'implementazione in modo separato.

Help Seller fa uso del Bridge Pattern per dare l’opportunità di sviluppare diverse implementazioni dello Storage Layer. Attualmente come sistema di storage è stato scelto di utilizzare un DBMS attraverso la tecnologia JPA. Grazie al Bridge Pattern in futuro sarà possibile utilizzare tecnologie come Hibernate.

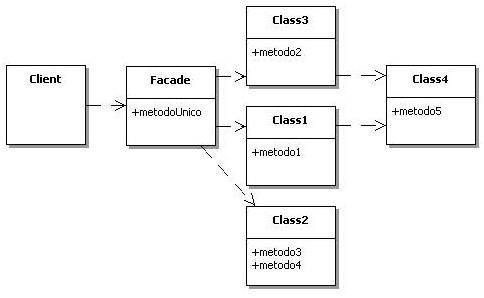
## 4.2 Observer Design Pattern:



il pattern si basa su uno o più oggetti, chiamati osservatori o observer, che vengono registrati per gestire un evento che potrebbe essere generato dall'oggetto "osservato", che può essere chiamato soggetto. Uno degli aspetti fondamentali è che tutto il funzionamento dell'observer si basa su meccanismi di [callback](https://it.wikipedia.org/wiki/Callback): in genere, una funzione, o un "blocco di codice" che viene passata come parametro ad un'altra funzione.

La piattaforma utilizza l’Observer Pattern nel front-end per effettuare chiamate a servizi back-end mandando e restituendo dati attraverso un oggetto Observable in modo asincrono.

## 4.3 Facade Pattern:



Letteralmente façade significa "facciata", ed infatti nella programmazione ad oggetti 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. Nell’esempio dell’immagine l’interfaccia ‘facade’ consente al client di utilizzare le classi descritte dalle varie classi attraverso una singola implementazione, nascondendo la complessità dell’operazione.

In Help Seller il design pattern del façade è protagonista nella parte beck-end dell’applicazione.

## 

## 4.4 Service Layer Design Pattern:

Il service layer è un pattern strutturale dell’architettura applicato nel lato backend dell’applicazione. Consiste nell’organizzazione dei servizi aventi funzionalità simili, nel nostro caso i servizi riguardanti le medesime entità.

Raggruppare i servizi in functional layer (livelli funzionali) riduce l’impatto di eventuali modifiche. La maggior parte dei cambiamenti effettuati in un layer, incidono solo sul layer in questione ed in minima parte sugli altri. Ciò semplifica notevolmente la manutenibilità del sistema.

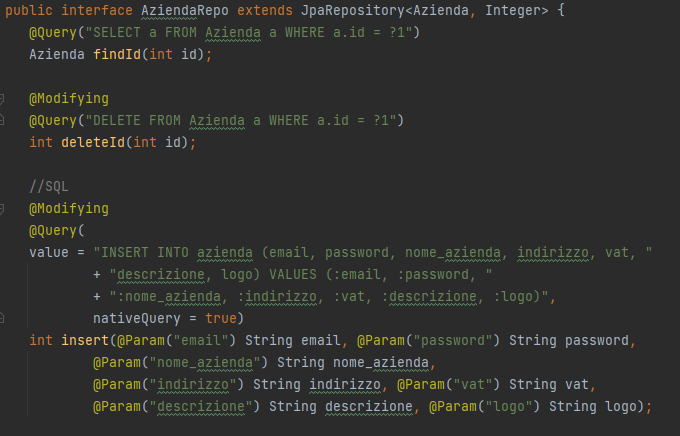
## 4.5 Repository Design Pattern:

Il repository è un modello consolidato nelle applicazioni Enterprise.

Una classe Repository contiene tutto il codice relativo alla persistenza ma nessuna logica di business. Permette l’astrazione del livello dati e si rivela un buon modo per centralizzare la gestione degli oggetti di dominio.

Aiuta la nostra applicazione a rimanere coesa e con un basso accoppiamento tra business logic e il livello di persistenza dei dati, semplificando l'implementazione per le classi entity.

Fornisce metodi per inserire, aggiornare e rimuovere un’entità e metodi che istanziano ed eseguono query specifiche.

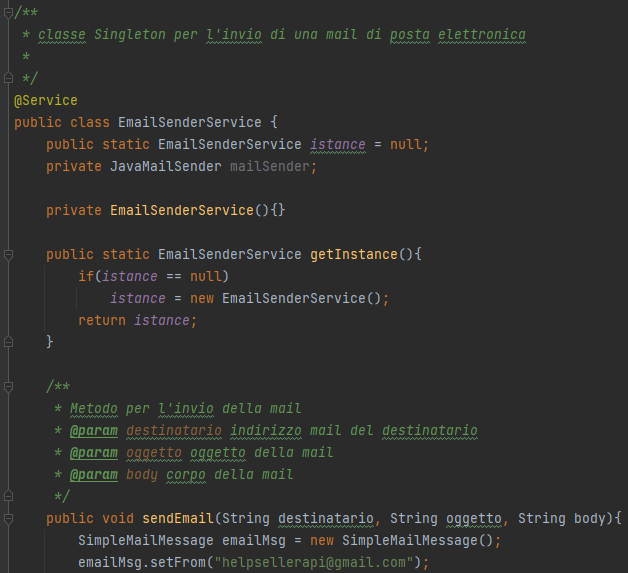


Snippet di codice che illustra una delle repository presente nel lato backend dell’applicazione Help Seller

## 4.6 Singleton Design Pattern:

Il Singleton è un design pattern creazionale che la lo scopo di garantire che di una determinata classe venga creata una e una sola istanza, e di fornire un punto di accesso globale ad essa.

Nel codice sorgente dell’applicazione il singleton design pattern è stato applicato nel servizio di posta elettronica SMTP, invocato nell’inserimento di un nuovo ordine

****

# 5 Glossario

Nella seguente sezione sono presenti le/i sigle/termini del documento che necessitano di una definizione.

| **Sigla/termine** | **Definizione** |
| --- | --- |
| **Package** | Raggruppamento di classi e interfacce |
| **Controller** | Classe che si occupa di gestire le richieste effettuate dal client |
| **Service** | Classe che implementa la logica di business, viene utilizzata dai controller o da un altro sottosistema |
| **Model** | Parte del sistema architetturale MVC che fornisce al sistema i metodi per accedere ai dati utili del sistema |
| **Camel notation** | La pratica di costruire parole composte o frasi unendo tutte le parole tra loro ma lasciando le loro iniziali maiuscole |
| **Facade** | Un oggetto che permette, attraverso un’interfaccia più semplice, l’accesso a sottosistemi che espongono interfacce complesse e molto diverse tra loro. |
| **Observer Design Pattern** | Design pattern usato come base per la gestione di eventi. |
| **Hibernate** | Piattaforma open source per la gestione della persistenza dei dati sul database. |