Object Design Document

EasyExpo

Riferimento	
Versione	1.0
Data	08/12/2020
Destinatario	Prof.ssa Ferrucci Filomena
Presentato da	Sabatino Strumolo, Davide Pappalardo, Katia Monaco De Simone, Lucrezia Robustelli, Gaetano Iuliano, Giuseppe Avino
Approvato da	Gilberto Recupito

Revision History

Data	Versione	Cambiamenti	Autori
10/12/2020	0.1	Scheletro Documento e Capitolo 1	Gaetano Iuliano, Lucrezia Robustelli
16/12/2020	0.2	Capitolo 2	Giuseppe Avino, Davide Pappalardo
05/01/2021	0.3	Interfaccia delle classi	Davide Pappalardo , Sabatino Strumolo
26/01/2021	0.4	Aggiunta Proxy , Class Diagram	Davide Pappalardo , Sabatino Strumolo
27/01/2021	1.0	Revisione	Davide Pappalardo , Sabatino Strumolo

Sommario

1. Introduzione..... 4

1.1 Object Design Trade-Off 4
1.2 Componenti Off-the-Shelf 4
1.3 Design Pattern 5
1.3.1 Singleton Pattern 5
1.3.2 Proxy Pattern 5
1.3.3 Data-Access-Object 6
1.4 Linee guida per la documentazione delle interfacce 6
1.4.1 *Classi e Interfacce Java* 6
1.4.2 Java Servlet Pages (JSP) 7
1.4.3 *Pagine HTML* 7
1.4.4 *File JavaScript* 7
1.4.5 *Fogli di stile CSS* 7
1.4.6 *Script SQL* 7
1.5 Definizioni, acronimi, abbreviazioni 8
1.6 Riferimenti 8
2. Packages 8
2.1 Model 8
2.2 Control 9
2.3 View 12
3. Interfacce delle Classi 12
4. Class Diagram 18

1. Introduzione

1.1 Object Design Trade-Off

Durante la fase di analisi e di progettazione del sistema abbiamo individuato diversi compromessi per lo sviluppo del sistema. Anche durante la fase di Object Design sorgono diversi compromessi che andremo ad analizzare in questo paragrafo:

Performance vs Memoria

Il sistema deve garantire risposte rapide a discapito della memoria utilizzata. Ciò significa che verranno introdotte delle ridondanze per evitare interrogazioni costose in termini di performance.

Tempo di risposta vs Affidabilità

Il sistema sarà implementato in modo tale da preferire l'affidabilità al tempo di risposta, in modo tale da garantire una risposta del sistema consistente a discapito del tempo impiegato per produrla.

Disponibilità vs Tolleranza ai guasti

Il sistema deve essere sempre disponibile al fruitore in caso di errore di una funzionalità a media o bassa priorità, anche a costo di rendere non disponibile quest'ultima per un lasso di tempo. Ovviamente se questa è una funzionalità core, il sistema verrà messo in manutenzione fin quando il guasto non verrà risolto.

Manutenibilità vs Performance

Il sistema sarà implementato preferendo la manutenibilità alla performance in modo da facilitare gli sviluppatori nel processo di aggiornamento del software a discapito delle performance del sistema.

Di seguito è riportata una tabella che mostra i design goal preferiti nei Trade Off. Il **grassetto** indica la preferenza.

Trade-Off	
Performance	Memoria
Affidabilità	Tempo di risposta
Disponibilità	Tolleranza ai guasti
Manutenibilità	Performance

1.2 Componenti Off-the-Shelf

Utilizzeremo diverse componenti Off-the-Shelf nell'implementazione del nostro sistema. Per quanto riguarda il Front End, utilizzeremo il framework Bootstrap. (https://getbootstrap.com/docs/4.4/getting-started/introduction/) per una gestione dello stile più

semplice. Questo framework è open source e integra all'interno di esso sistemi per gestire in maniera semplificata il codice HTML e CSS. Questo framework inoltre include alcune funzioni definite in JavaScript che dovranno essere integrate.

Dal punto di vista funzionale invece, utilizzeremo JQuery, un framework di JavaScript che permette di agire sul DOM del documento HTML, di effettuare chiamate AJAX e di gestire le animazioni della pagina web in maniera molto più semplice.

Per quanto riguarda il lato Back End, utilizzeremo il Web Container di JavaEE ed il WebServer Tomcat per gestire le richieste al nostro sistema.

1.3 Design Pattern

1.3.1 Singleton Pattern

Il singleton è un design pattern creazionale che ha lo scopo di garantire che di una determinata classe venga creata una e una sola istanza, e di fornire un punto di accesso globale a tale istanza. L'implementazione più semplice di questo pattern prevede che la classe singleton abbia un unico costruttore privato, in modo da impedire l'istanziazione diretta della classe. Dato che abbiamo bisogno di una sola istanza condivisa da più oggetti, attraverso questo pattern, è stata implementata la classe DBConnection (Connessione al DB).

1.3.2 Proxy Pattern

Il pattern Proxy fornisce una interfaccia per oggetti laboriosi, che richiedono risorse e tempo per essere creati.

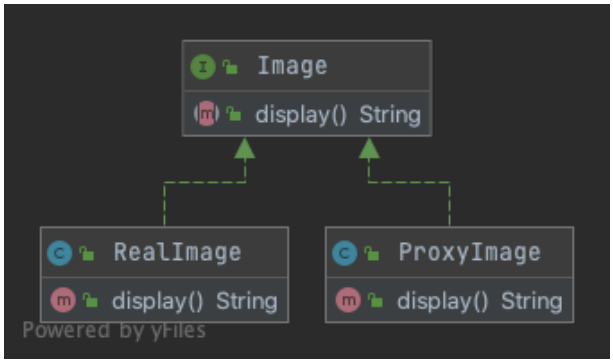
Il pattern proxy consente di posticipare l'effettivo accesso alle risorse dell'oggetto quando è davvero necessario, consentendo maggiore fluidità all'elaborazione.

Il pattern Proxy è utile anche tutte le volte che bisogna mediare l'accesso diretto ad un oggetto.

Il Proxy ha la stessa interfaccia dell'oggetto cui si vuole accedere, ed è un pattern strutturale.

Consideriamo quindi per il nostro progetto un oggetto che rappresenta un immagine memorizzata come file. Caricare l'immagine a massima risoluzione è costoso. Quindi possiamo realizzare un Proxy Pattern per risolvere il problema.

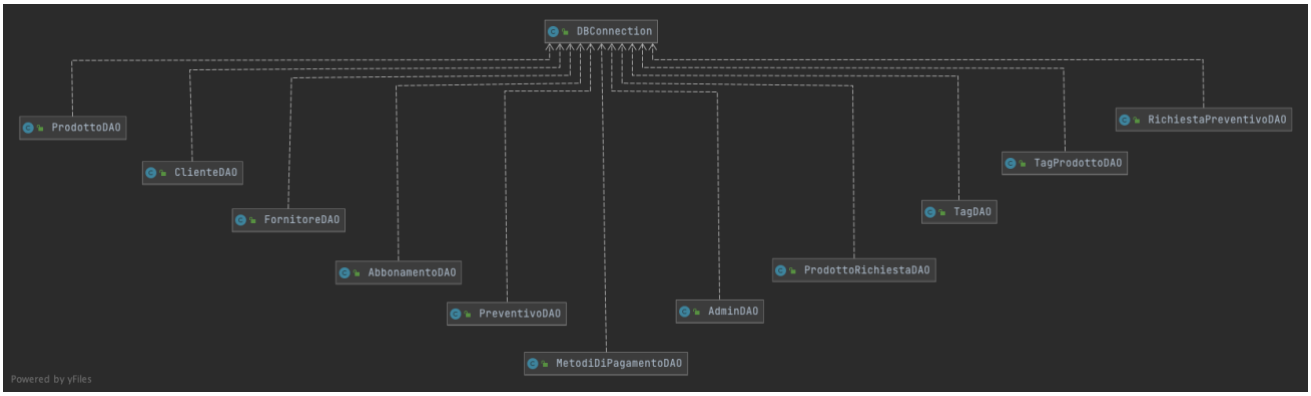
L'oggetto ImageProxy prende il posto dell' oggetto Image e utilizza la stessa interfaccia Image. Quando l'Image deve essere scaricata, ImageProxy scarica i dati dal server a crea un ReaImage Object. Se invece l'utente non ha bisogno di quell'immagine , e quindi non invoca il metodo paint(), l'oggetto Real Image non viene creato.



1.3.3 Data-Access-Object

Il modello DAO viene utilizzato per separare l'API o le operazioni di accesso ai dati di basso livello dai servizi aziendali di alto livello. Le componenti del Data Access Object Pattern sono:

- **Data Access Object Interface:** questa interfaccia definisce le operazioni standard da eseguire su uno o più oggetti del modello.
- **Data Access Object concrete class**: questa classe implementa l'interfaccia precedente. Inoltre ottiene i dati da un database o qualsiasi altro meccanismo di archiviazione.
- **Model Object or Value Object:** Questo oggetto è un semplice POJO contenente metodi get / set per memorizzare i dati recuperati utilizzando la classe DAO.



1.4 Linee guida per la documentazione delle interfacce

In questo paragrafo verranno definite le linee guida a cui un programmatore deve attenersi nell'implementazione del sistema.

In ciascun sotto paragrafo, quando si parla di indentazione ci si riferisce ad una tabulazione.

1.4.1 Classi e Interfacce Java

Lo standard nella definizione delle classi e delle interfacce Java è quello definito da Google (http://google.github.io/styleguide/javaguide.html). Ciascuna classe e ciascun metodo deve essere documentato seguendo lo stile di documentazione JavaDoc.

1.4.2 Java Servlet Pages (JSP)

Le JSP costruiscono pagine HTML in maniera dinamica che devono rispettare il formato definito nel sotto paragrafo sottostante. Devono anche attenersi alle linee guida per le classi Java definito nel sotto paragrafo soprastante. Inoltre, le JSP devono attenersi alle seguenti puntualizzazioni:

1. Il tag di apertura (<%) è seguito immediatamente dalla fine della riga;
2. Il tag di chiusura (%>) si trova all'inizio di una riga che non contiene nient'altro;
3. Istruzioni Java che consistono in una sola riga possono contenere il tag di apertura e chiusura sulla stessa riga;

1.4.3 Pagine HTML

Le pagine HTML devono essere conformi allo standard HTML5. Inoltre, il codice HTML deve utilizzare l'indentazione per facilitare la lettura e di conseguenza la manutenzione. Le regole di indentazione sono le seguenti:

1. Ogni tag deve avere un'indentazione maggiore del tag che lo contiene;
2. Ogni tag di chiusura deve avere lo stesso livello di indentazione di quello di apertura;

1.4.4 File JavaScript

Gli script JavaScript seguono anche essi le linee guida definite dallo standard Google (https://google.github.io/styleguide/jsguide.html). Sia i file che i metodi JavaScript devono essere documentati seguendo lo stile di JavaDoc.

1.4.5 Fogli di stile CSS

Ogni foglio di stile deve essere inizializzato da un commento analogo a quello presente nei file Java.

Ogni regola CSS deve essere formattata nel seguente modo:

1. I selettori della regola si trovano a livello 0 di indentazione, uno per riga;
2. L'ultimo selettore della regola è seguito da parentesi graffa aperta ({);
3. Le proprietà che costituiscono la regola sono listate una per riga e sono indentate rispetto ai selettori;
4. La regola è terminata da una parentesi graffa chiusa (}), collocata da sola su una riga;

1.4.6 Script SQL

Le istruzioni e le clausole SQL devono essere costituite da sole lettere maiuscole.

I nomi delle tabelle hanno la prima lettera maiuscola e le successive minuscole e il loro nome deve essere un sostantivo singolare.

I nomi degli attributi devono essere costituiti da sole lettere minuscole e possono contenere più parole separate da un underscore (_).

1.5 Definizioni, acronimi, abbreviazioni

Definizioni

DOM = Modella la struttura di pagine web.

Acronimi

ODD = Object Design Document

GUI = Graphical User Interface (Interfaccia grafica utente)

HTML = HyperText Markup Language

CSS = Cascading Style Sheet

DOM = Document Object Model

AJAX = Asynchronous JavaScript And XML

JavaEE = Java Enterprise Edition

SQL = Structured Query Language

JSP = Java Servlet Pages

1.6 Riferimenti

- Libro:

  -- Object-Oriented Software Engineering (Using UML, Patterns, and Java) Third Edition

  Autori:

  -- Bernd Bruegge

  -- Allen H. Dutoit

- Wikipedia.org

2. Packages

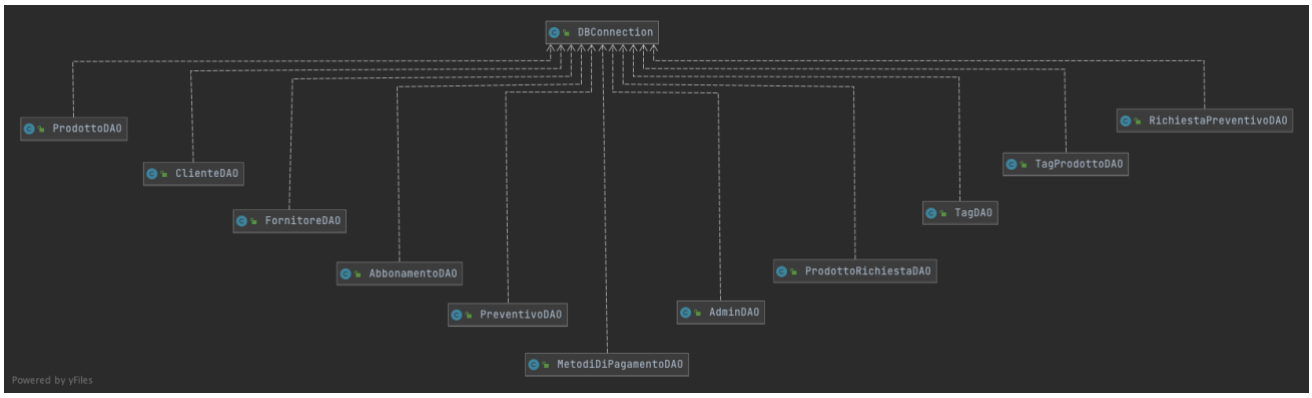
In questo paragrafo è riportata la decomposizione dei sottosistemi nei vari packages delle classi Java.

2.1 Model

Il package Model contiene le classi che si occupano di modellare la logica di business del sistema. Tale package è a sua volta suddiviso in due diversi sottopackage:

- **DAO**: Contiene le interfacce dei Data Access Object (DAO), ovvero oggetti che si relazionano con il gestore della persistenza per effettuare operazioni di tipo Create, Read, Update e Delete. Inoltre contiene una classe DBConnection attraverso la quale avviene la connessione al database.
- **POJO**: contiene tutte le classi che servono per creare gli oggetti utilizzati per l'implementazione. Queste classi definiscono dei metodi get e set. Contiene delle classi Enum utili al mantenimento di un'alta estensibilità del codice.

Model DAO:



Model POJO:

2.2 Control

Il package Control contiene tutte le Servlet usate per gestire le richieste del client.

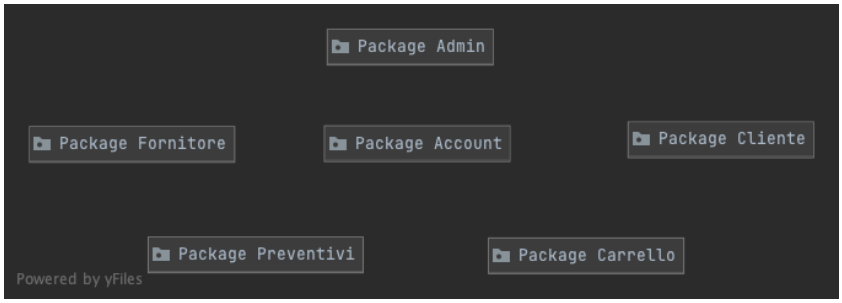
Tale package è a sua volta suddiviso in diversi sottopackage:

- Account: Contiene tutte le Servlet per la gestione delle operazioni di registrazione, login e di logout;
- Cliente: Contiene tutte le Servlet per la gestione delle richieste da parte del Cliente;
- Fornitore: Contiene tutte le Servlet per la gestione delle richieste da parte del Fornitore;
- Admin: Contiene tutte le Servlet per la gestione delle richieste da parte dell'Admin;

- Carrello: Contiene le Servlet per la gestione delle richieste da parte del Cliente per le operazioni di Carrello e di RichiestaPreventivo;
- Preventivo: Contiene le Servlet per la gestione dei preventivi da parte del Cliente e del Fornitore;

Preventivo: Contiene le Servlet per la gestione delle richieste da parte del Cliente per l'operazione di VisualizzaPreventiviRicevuti e le richieste da parte del Fornitore per le operazioni di VisualizzaRichiestaPreventivo e CompilaPreventivo

Controller:

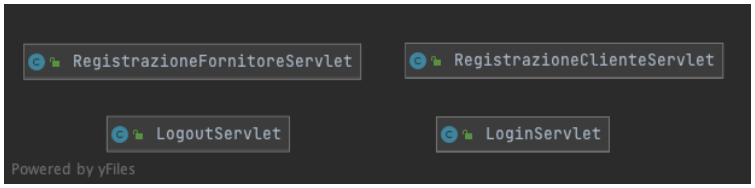


Controller.Fornitore:

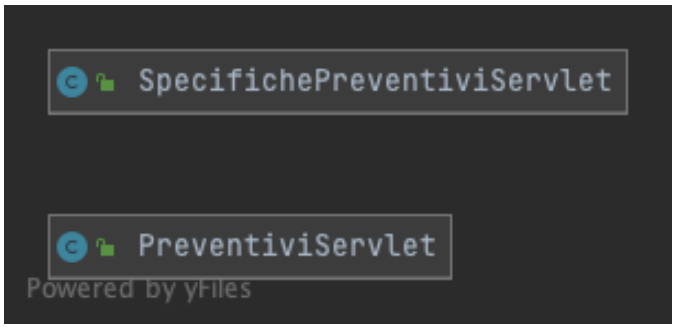
Controller.Cliente:

Controller.Carrello:

Controller.Account:



Controller.Preventivo:

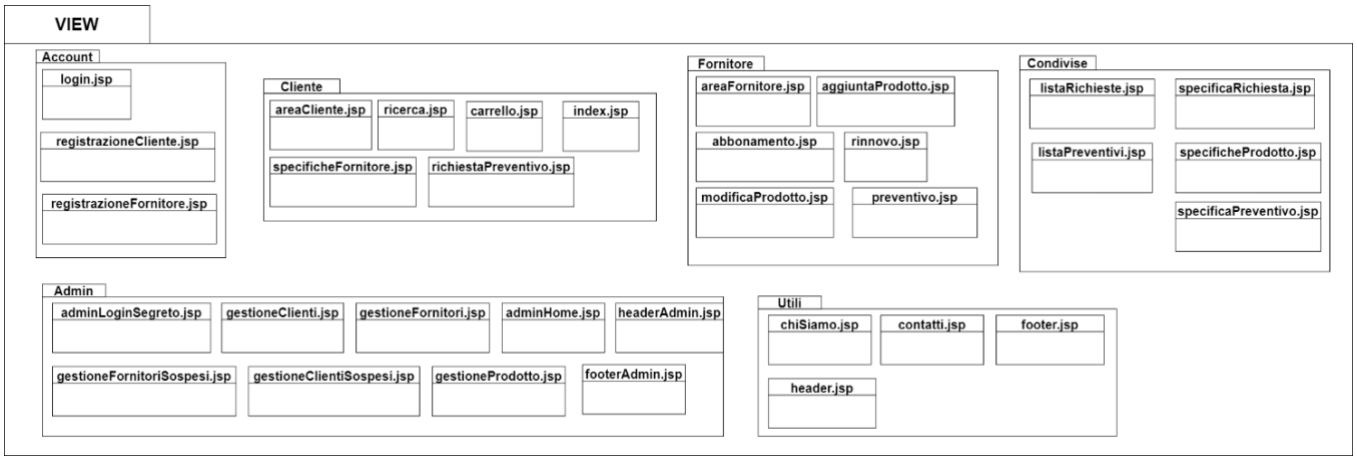


Controller.Admin:

2.3 View

Il package View contiene tutte le pagine JSP create dinamicamente tramite le quali c'è l'interazione con il sistema.
Tale package è a sua volta suddiviso in diversi sottopackage:

- Account: Contiene le pagine JSP riguardanti ai servizi di login, registrazione e logout da parte del Cliente e del Fornitore;
- Cliente: Contiene le pagine JSP attraverso le quali il Cliente comunica con il sistema;
- Fornitore: Contiene le pagine JSP attraverso le quali il Fornitore comunica con il sistema;
- Admin: Contiene le pagine JSP attraverso le quali l'Admin comunica con il sistema.
- Preventivi: Contiene le pagine JSP attraverso le quali il Cliente e il Fornitore possono comunicare con il sistema.
- Utili: Contiene le pagine JSP attraverso le quali il Cliente e il Fornitore possono comunicare con il sistema.



3. Interfacce delle Classi

Nome classe	AbbonamentoDAO
Descrizione	Classe che definisce i metodi dell'entità Abbonamento
Attributi e metodi	+doRetrieveByIdAbbonamento(int) +createAbbonamento(Abbonamento)
Precondizione	**Context** AbbonamentoDAO::doRetrieveByIdAbbonamento(idAbbonamento:int): Abbonamento **Pre** idAbbonamento != null **Context** AbbonamentoDAO::createAbbonamento(abbonamento:Abbonamento): void **Pre** abbonamento != null
Postcondizione	

Invariante	

Nome classe	AdminDAO
Descrizione	Classe che definisce i metodi dell'entità Admin
Attributi e metodi	+doRetrieveByEmail(String)
Precondizione	**Context** AdminDAO::doRetrieveByEmail(email:String): Admin **Pre** email != null
Postcondizione	
Invariante	

Nome classe	ClienteDAO
Descrizione	Classe che definisce i metodi dell'entità Cliente
Attributi e metodi	+doRetriveByCF(String) +doRetrieveByEmailandPassword(String,String) +createCliente(Cliente) +deleteCliente(String) +doRetrievebyStato(int) +updateStato(int,String)
Precondizione	**Context** ClienteDAO::doRetrieveByCF(codiceFiscale:String): Cliente **Pre** codiceFiscale != null **Context** ClienteDAO::doRetrieveByEmailandPassword(email:String , password:String): Cliente **Pre** email != null && password!=null **Context** ClienteDAO::createCliente(cliente:Cliente): void **Pre** cliente!= null **Context** ClienteDAO::deleteCliente(codiceFiscale:String): void **Pre** codiceFiscale != null **Context** ClienteDAO:: doRetrieveByStato(val:int): List<Cliente> **Pre** val != null **Context** ClienteDAO:: doRetrieveByStato(val:int): List<Cliente> **Pre** val != null **Context** ClienteDAO:: updateStato(val:int,codiceFiscale:String): void **Pre** val != null && codiceFiscale != null
Postcondizione	
Invariante	

Nome classe	FornitoreDAO
Descrizione	Classe che definisce i metodi dell'entità Fornitore

Attributi e metodi	+doRetrieveByPIVA(String) +createFornitore(Fornitore) +doRetrieveByStato(String) +doRetrieveByEmailandPassword(String,String) +deleteFornitore(String) +doRetrieveByNomeECognome(String) +updateBooleanFornitore(Fornitore) +updateStato(int,String)
Precondizione	**Context** FornitoreDAO::doRetrieveByPIVA(partitaIva:String): Fornitore **Pre** partitaIva != null **Context** FornitoreDAO::createFornitore(fornitore:Fornitore): void **Pre** fornitore != null **Context** FornitoreDAO::doRetrieveByStato(val:int): List<Fornitore> **Pre** val != null **Context** FornitoreDAO::doRetrieveByEmailandPassword(email:String, password:String): Fornitore **Pre** email != null && password != null **Context** FornitoreDAO::deleteFornitore(partitaIva:String): void **Pre** partitaIva != null **Context** FornitoreDAO::doRetrieveByNomeECognome(ricercato:String): List<Fornitore> **Pre** ricercato != null **Context** FornitoreDAO::updateBooleanFornitore(fornitore:Fornitore): void **Pre** fornitore != null **Context** FornitoreDAO::updateStato(val:int,codiceFiscale:String): void **Pre** val != null && codiceFiscale != null
Postcondizione	
Invariante	

Nome classe	MetodoDiPagamentoDAO
Descrizione	Classe che definisce i metodi dell'entità MetodoPagamento
Attributi e metodi	+doRetrieveAllByPartitaIva(String) +createMetodoPagamento(MetodoPagamento) +doRetrieveByNumCarta(String)
Precondizione	**Context** MetodoDiPagamentoDAO::doRetrieveAllByPartitaIva(partitaIva:String): List<MetodoPagamento> **Pre** partitaIva != null **Context** MetodoDiPagamentoDAO::createMetodoPagamento(metodoPagamento:MetodoPagamento): void **Pre** metodoPagamento != null **Context** MetodoDiPagamentoDAO::doRetrieveByNumCarta(numeroCarta:String): MetodoPagamento

	Pre numeroCarta != null
Postcondizione	
Invariante	

Nome classe	PreventivoDAO
Descrizione	Classe che definisce i metodi dell'entità Abbonamento
Attributi e metodi	+doRetrieveByIdPreventivo(int) +doRetrieveByPartitaIva(String) +createPreventivo(Preventivo) +deletePreventivo(int) +doRetrieveByCodiceFiscale(String)
Precondizione	**Context** PreventivoDAO::doRetrieveByIdPreventivo(idPreventivo:int): Preventivo **Pre** idPreventivo != null **Context** PreventivoDAO::doRetrieveByPartitaIva(partitaIva:String): List<Preventivo> **Pre** partitaIva != null **Context** PreventivoDAO::createPreventivo(preventivo:Preventivo): void **Pre** preventivo != null **Context** PreventivoDAO::deletePreventivo(idPreventivo:int): void **Pre** idPreventivo != null **Context** PreventivoDAO::doRetrieveByCodiceFiscale(codiceFiscale:String): List<Preventivo> **Pre** codiceFiscale != null
Postcondizione	
Invariante	

Nome classe	ProdottoDAO
Descrizione	Classe che definisce i metodi dell'entità Prodotto
Attributi e metodi	+doRetrieveByIdProdottoEPartitaIva(int,String) +doRetrieveByPartitaIva(String) +doRetrieveAll() +createProdotto(Prodotto) +deleteProdotto(int) +updateProdotto(Prodotto) +doRetrieveRandom(int,int) +doRetrieveByTitolo(String)
Precondizione	**Context** ProdottoDAO::doRetrieveByIdProdottoEPartitaIva(idProdotto:int,partitaIva:String): Prodotto **Pre** idProdotto != null && partitaIva !=null

	Context ProdottoDao:: doRetrieveByPartitaIva(partitaIva: String): List<Prodotto> **Pre** partitaIva != null **Context** ProdottoDao::doRetrieveAll(offset: int , limit: int): List<Prodotto> **Pre** - **Context** ProdottoDao::createProdotto(prodotto: Prodotto):void **Pre** prodotto != null **Context** ProdottoDao::deleteProdotto(idProdotto: int): void **Pre** idProdotto != null **Context** ProdottoDao::updateProdotto(prodotto: Prodotto): void **Pre** prodotto != null **Context** ProdottoDao::doRetrieveRandom(offset: int,limit: int): List<Prodotto> **Pre** offset != null && limit != null **Context** ProdottoDao:: doRetrieveByTitolo(titolo: String): List<Prodotto> **Pre** titolo != null
Postcondizione	
Invariante	

Nome classe	ProdottoRichiestaDAO
Descrizione	Classe che definisce i metodi dell'entità ProdottoRichiesta
Attributi e metodi	+doRetriveById(int) +doRetriveByIdRichiesta(int) +createProdottoRichiesta(ProdottoRichiesta) +doRetriveByIdProdottoePartitaIva(int,String)
Precondizione	**Context** ProdottoRichiestaDAO::doRetriveById(id:int): ProdottoRichiesta **Pre** id != null **Context** ProdottoRichiestaDAO::doRetriveByIdRichiesta(idRichiesta:int): List<ProdottoRichiesta> **Pre** idRichiesta != null **Context** ProdottoRichiestaDAO::createProdottoRichiesta(prodottoRichiesta:ProdottoRichiesta): void **Pre** prodottoRichiesta != null **Context** ProdottoRichiestaDAO:: doRetriveByIdProdottoePartitaIva (idProdotto:int, partitaIva:String): List<ProdottoRichiesta> **Pre** idProdotto != null && partitaIva != null
Postcondizione	
Invariante	

Nome classe	RichiestaPreventivoDAO
Descrizione	Classe che definisce i metodi dell'entità RichiestaPreventivo

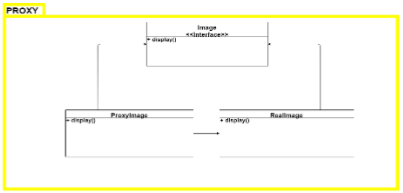
Attributi e metodi	+doRetrieveByIdRichiesta(int) +doRetrieveByPartitaIva(String) +doRetrieveByCodiceFiscale(String) +createRichiestaPreventivo(RichiestaPreventivo) +deleteRichiestaPreventivo(int) +updateRichiestaPreventivo(RichiestaPreventivo)
Precondizione	**Context** RichiestaPreventivoDAO::doRetrieveByIdRichiesta(idRichiesta:int): RichiestaPreventivo **Pre** idRichiesta != null **Context** RichiestaPreventivoDAO:: doRetrieveByPartitaIva (partitaIva:String): List<RichiestaPreventivo> **Pre** partitaIva != null **Context** RichiestaPreventivoDAO:: doRetrieveByCodiceFiscale(codiceFiscale:String): List<RichiestaPreventivo> **Pre** codiceFiscale != null **Context** RichiestaPreventivoDAO::createRichiestaPreventivo(richiestaPreventivo:RichiestaPreventivo): void **Pre** richiestaPreventivo != null **Context** RichiestaPreventivoDAO::deleteRichiestaPreventivo(idRichiesta:int): void **Pre** idRichiesta != null **Context** RichiestaPreventivoDAO::updateRichiestaPreventivo(richiestaPreventivo:RichiestaPreventivo): void **Pre** richiestaPreventivo != null
Postcondizione	
Invariante	

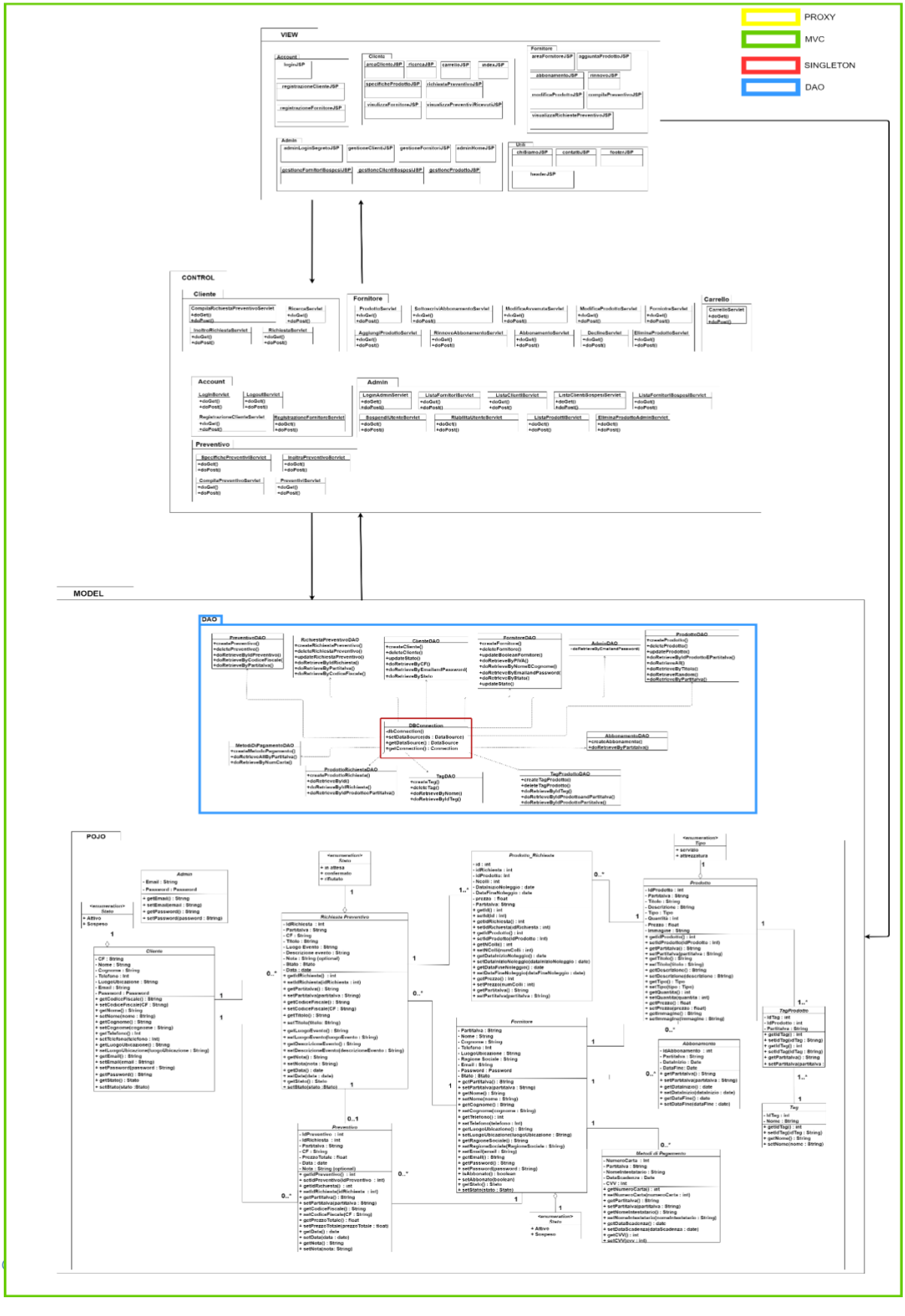
Nome classe	TagDAO
Descrizione	Classe che definisce i metodi dell'entità Tag
Attributi e metodi	+doRetrieveByIdTag(int) +doRetrieveByNome(int) +createTag(Tago) +deleteTag(int)
Precondizione	**Context** TagDAO::doRetrieveByIdTag(idTag:int): Tag **Pre** idTag != null **Context** TagDAO::doRetrieveByNome(nome:String): List<Tag> **Pre** nome != null **Context** TagDAO::createTag(tag:Tag): void **Pre** tag != null **Context** TagDAO::deleteTag(idTag:int): void **Pre** idTag != null

Postcondizione	
Invariante	

Nome classe	TagProdottoDAO
Descrizione	Classe che definisce i metodi dell'entità TagProdotto
Attributi e metodi	+doRetrieveByIdTag(int) +createTagProdotto(TagProdotto) +deleteTagProdotto(int,int,String) +doRetrieveByIdProdottoandPartitaIva(int,String) + doRetrieveByIdProdottoPartitaIva(int,String)
Precondizione	**Context** TagProdottoDAO:doRetrieveByIdTag(idTag:int): TagProdotto **Pre** idTag != null **Context** TagProdottoDAO::createTagProdotto(tagProdotto:TagProdotto): void **Pre** tagProdotto != null **Context** TagProdottoDAO::deleteTagProdotto(idTag:int,idProdotto:int,partitaIva:String): void **Pre** idTag != null && idProdotto != null && partitaIva != null **Context** TagProdottoDAO:doRetrieveByIdProdottoandPartitaIva(idProdotto:int, partitaIva:String): TagProdotto **Pre** idProdotto != null && partitaIva !=null **Context** TagProdottoDAO:doRetrieveByIdProdottoPartitaIva(idProdotto:int, partitaIva:String): TagProdotto **Pre** idProdotto != null && partitaIva !=null
Postcondizione	
Invariante	

4. Class Diagram


PROXY
Image
«interface»
+ display()
ProxyImage
+ display()
RealImage
+ display()


PROXY
MVC
SINGLETON
DAO
VIEW
CONTROL
Cliente
Fornitore
Carrello
Account
Admin
Preventivo
MODEL
DAO
DBConnection
POJO