RECIPE MANAGER

Object Design Document

Anno Accademico 2019/20

UNIVERSITA' DEGLI STUDI DI SALERNO – FACOLTA DI SCIENZE FF. MM. NN.

CORSO DI LAUREA TRIENNALE IN INFORMATICA – CORSO DI INGEGNERIA DEL SOFTWARE

SOMMARIO

Introduzione

Dopo la realizzazione dei documenti RAD e SDD abbiamo descritto, in linea di massima, quello che sarà il nostro sistema e quindi i nostri obiettivi, tralasciando gli aspetti dell'implementazione.

Il seguente documento ha lo scopo di produrre un modello capace di integrare in modo coerente e preciso tutte le diverse funzionalità individuate nelle fasi precedenti.

In particolare, 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 il signature dei sottosistemi definiti nel System Design.

Object Design Trade-offs

- Comprensibilità vs Tempo:
  Il codice deve essere al quanto più comprensibile per poter facilitare la fase di testing ed eventuali future modifiche del codice.
  A tale scopo, il codice sarà quindi accompagnato da commenti che ne semplifichino la comprensione. Questa caratteristica incrementerà il tempo di sviluppo, ma allo stesso tempo lo renderà più comprensibile.

  Interfaccia vs Usabilità:
  Il sistema verrà sviluppato con un'interfaccia grafica realizzata in modo da poter essere molto semplice, chiara ed intuitiva. Nell'interfaccia saranno presenti form, menu e pulsanti, disposti in maniera da rendere semplice l'utilizzo del sistema da parte dell'utente finale.

- Sicurezza vs Efficienza:
  La sicurezza, come descritto nei requisiti non funzionali, rappresenta uno degli aspetti importanti del sistema.
  A causa dei tempi di sviluppo molto limitati, ci limiteremo ad implementare un sistema di sicurezza basato sull'utilizzo di username e password degli utenti, quest'ultima con crittografia hash a singola via.

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
    6. Evitando la notazione ungherese
    7. Utilizzando solo caratteri consentiti (a-z, A-Z, 0-9)

Variabili:

- I nomi delle variabili devono cominciare con una lettera minuscola, e le parole seguenti con la lettera maiuscola.

  Quest'ultime devono essere dichiarate ad inizio blocco, solamente una per riga e devono essere tutte allineate e facilitarne la leggibilità. Esse possono essere annotate con dei commenti.
  Esempio: nomeRicetta

- E' inoltre possibile, in alcuni casi, utilizzare il carattere underscore ("_") per la definizione del nome.

Metodi:

- I nomi dei metodi devono cominciare con una lettera minuscola, e le parole seguenti con la lettera maiuscola. 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().

- I commenti dei metodi devono essere raggruppati in base alla loro funzionalità, la descrizione dei metodi deve apparire prima di ogni dichiarazione di metodo, e deve descriverne lo scopo. Deve includere anche informazioni sugli argomenti, sul valore di ritorno, e se applicabile, sulle eccezioni.

Classi e pagine:

- I nomi delle classi e delle pagine devono cominciare con una lettera maiuscola, e anche le parole seguenti all'interno del nome devono cominciare con una lettera maiuscola. I nomi di quest'ultime devono fornire informazioni sul loro scopo.

- La dichiarazione di classe deve essere caratterizzata da:

    1. Dichiarazione della classe pubblica
    2. Dichiarazioni di costanti
    3. Dichiarazioni di variabili di classe
    4. Dichiarazione di variabili d'istanza
    5. Costruttore
    6. Commento e dichiarazione dei metodi

Definizioni, acronimi e abbreviazioni

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

Riferimenti

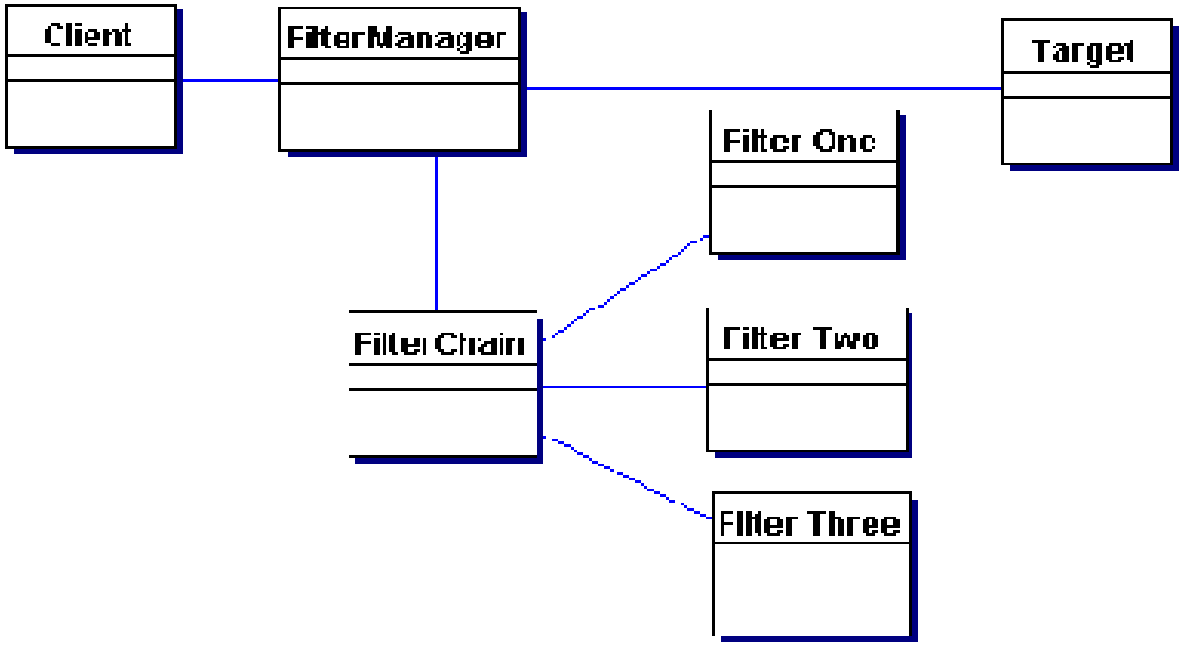
- B.Bruegge, A. H. Dutoit, Object Oriented Software Engineering - Using UML, Pattern and Java, Prentice Hail, 3rd edition, 2009.

- Documento RAD del progetto RecipeManager.

- Docuemento SDD del progetto RecipeManager.

Design pattern

Protection Proxies



RecipeManager fa uso del Protection Proxies Pattern, quest'ultimo deriva dal Proxy Pattern. Nella sua forma più generale, un proxy è una classe che funziona come interfaccia per qualcos'altro.

L'altro potrebbe essere qualunque cosa: una connessione di rete, un grosso oggetto in memoria, un file e altre risorse che sono costose o impossibili da duplicare. Nelle situazioni in cui molte copie di un oggetto complesso devono esistere, il proxy pattern può essere adottato per incorporare il Flyweight pattern per ridurre l'occupazione di memoria dell'oggetto.

Tipicamente viene creata un'istanza di oggetto complesso, e molteplici oggetti proxy, ognuno dei quali contiene un riferimento al singolo oggetto complesso. Ogni operazione svolta sui proxy viene trasmessa all'oggetto originale. Una volta che tutte le istanze del proxy sono distrutte, l'oggetto in memoria può essere deallocato.
Le classi partecipanti nel proxy pattern sono:

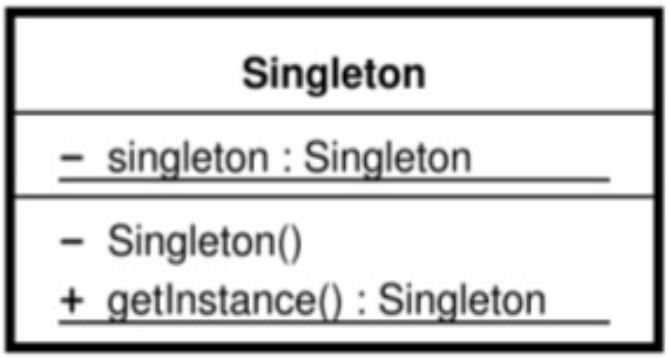
Oggetto: interfaccia implementata da RealSubject e che rappresenta i suoi servizi. L'interfaccia deve essere implementata dal proxy, in modo che il proxy possa essere utilizzato in qualsiasi posizione in cui è possibile utilizzare RealSubject.

Proxy: Mantiene un riferimento che consente al proxy di accedere a RealSubject. Implementa la stessa interfaccia implementata da RealSubject in modo che il Proxy possa essere sostituito da RealSubject. Controlla l'accesso a RealSubject e potrebbe essere responsabile della sua creazione e cancellazione.

Altre responsabilità dipendono dal tipo di proxy. RealSubject: l'oggetto reale rappresentato dal proxy.

Ci sono varie situazioni in cui è applicabile il proxy pattern nel nostro caso come detto all'inizio utilizziamo il Protection Proxies dove un proxy controlla l'accesso alle risorse, dandone l'accesso ad alcuni utenti negando l'accesso ad altri. L'accesso alle risorse viene effettuato in particolare sul tipo di utenti, distinguendo l'utente non registrato dall'utente registrato e dall'utente amministratore.

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.

La classe fornisce inoltre un metodo "getter" statico che restituisce l'istanza della classe (sempre la stessa), creandola preventivamente o alla prima chiamata del metodo, e memorizzandone il riferimento in un attributo privato anch'esso statico.

Il secondo approccio si può classificare come basato sul principio della lazy initialization (letteralmente "inizializzazione pigra") in quanto la creazione dell'istanza della classe viene rimandata nel tempo e messa in atto solo quando ciò diventa strettamente necessario (al primo tentativo di uso).

Utilizziamo il singleton pattern per collegare le classi al nostro DB tramite la classe: Database.java.

Packages

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

- Interface layer
- Application Logic layer
- Storage layer.

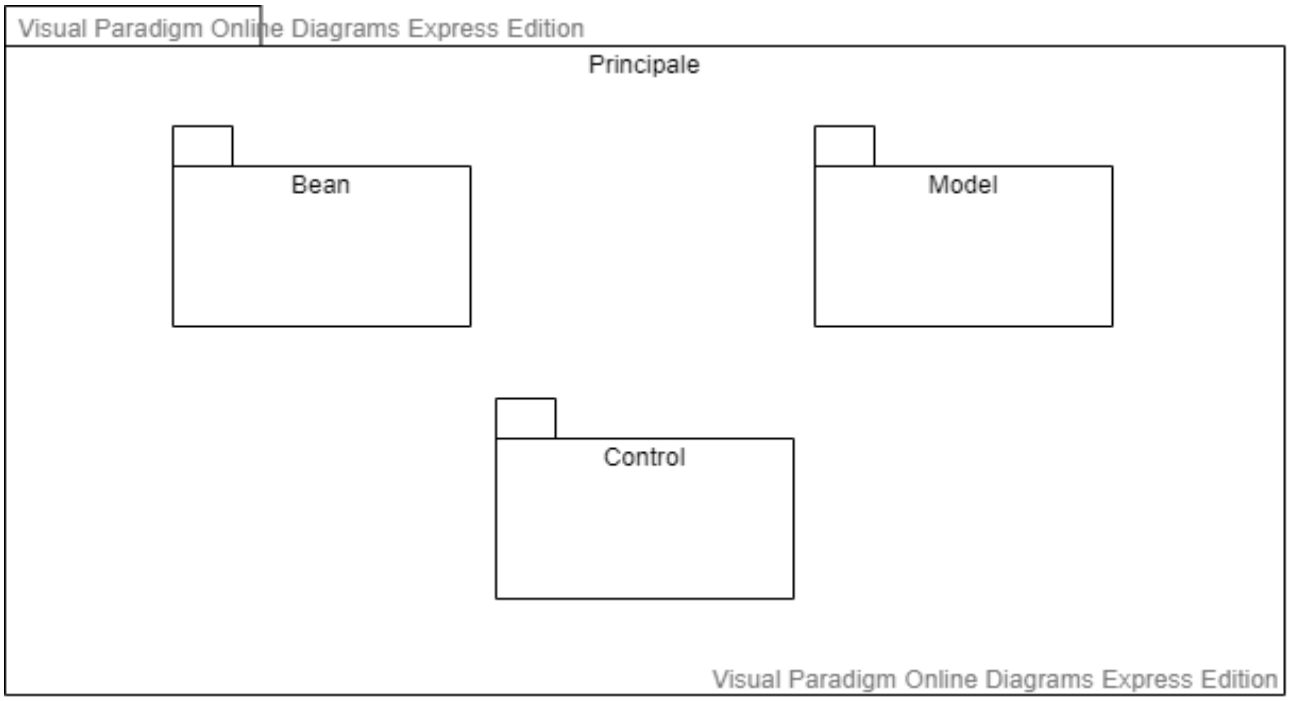
Il package RecipeManager contiene sottopackage che a loro volta inglobano classi atte alla gestione delle richieste utente. Le classi contenute nel package svolgono il ruolo di gestore logico del sistema.

Interface layer	Rappresenta l'interfaccia del sistema, ed offre la possibilità all'utente di interagire con quest'ultimo, offrendo sia la possibilità di inviare, in input, che di visualizzare, in output, dati.

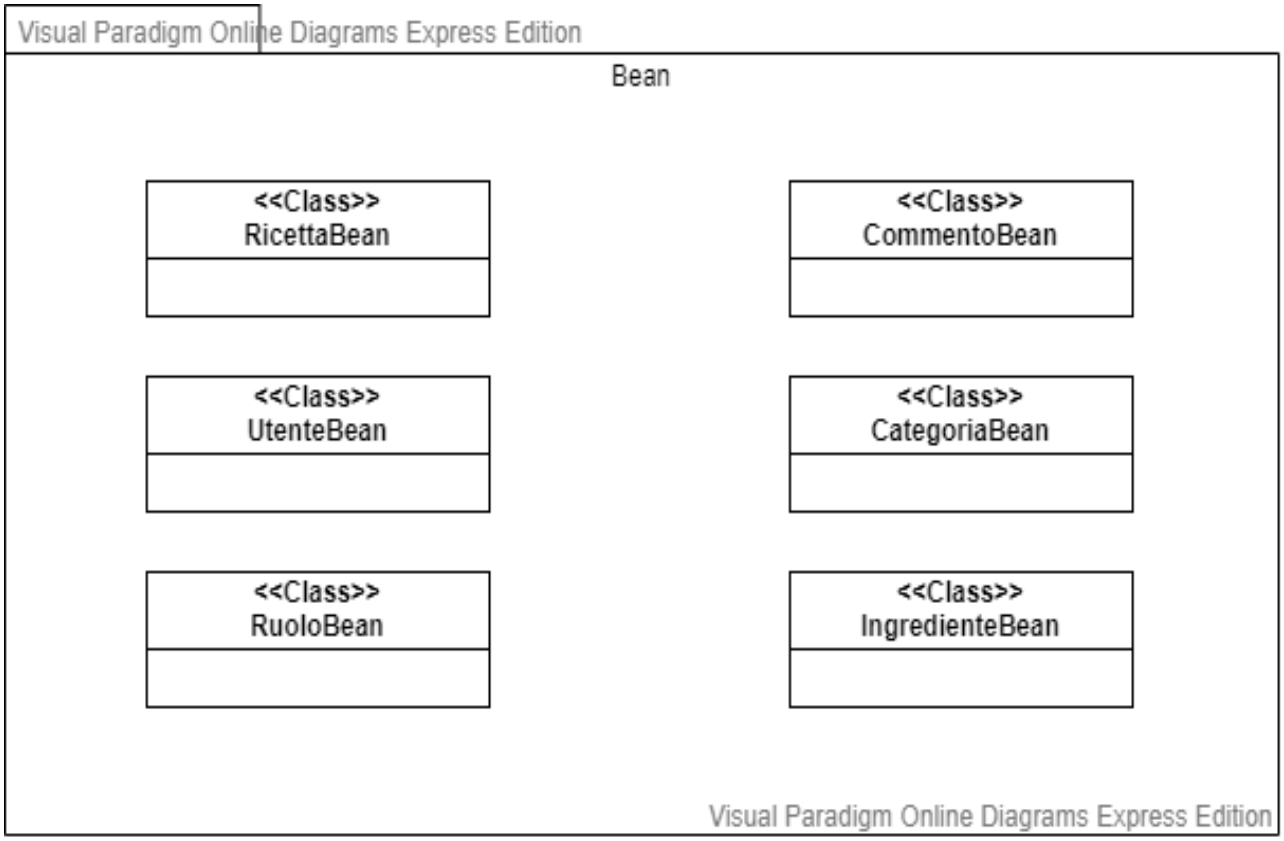
Application Logic layer	Ha il compito di elaborare i dati da inviare al client, e spesso grazie a delle richieste fatte al database, tramite lo Storage Layer, accede ai dati persistenti. Si occupa di varie gestioni quali: ● Gestione Utenti ● Gestione Ricetta ● Gestione Amministrazione

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

Package Principale

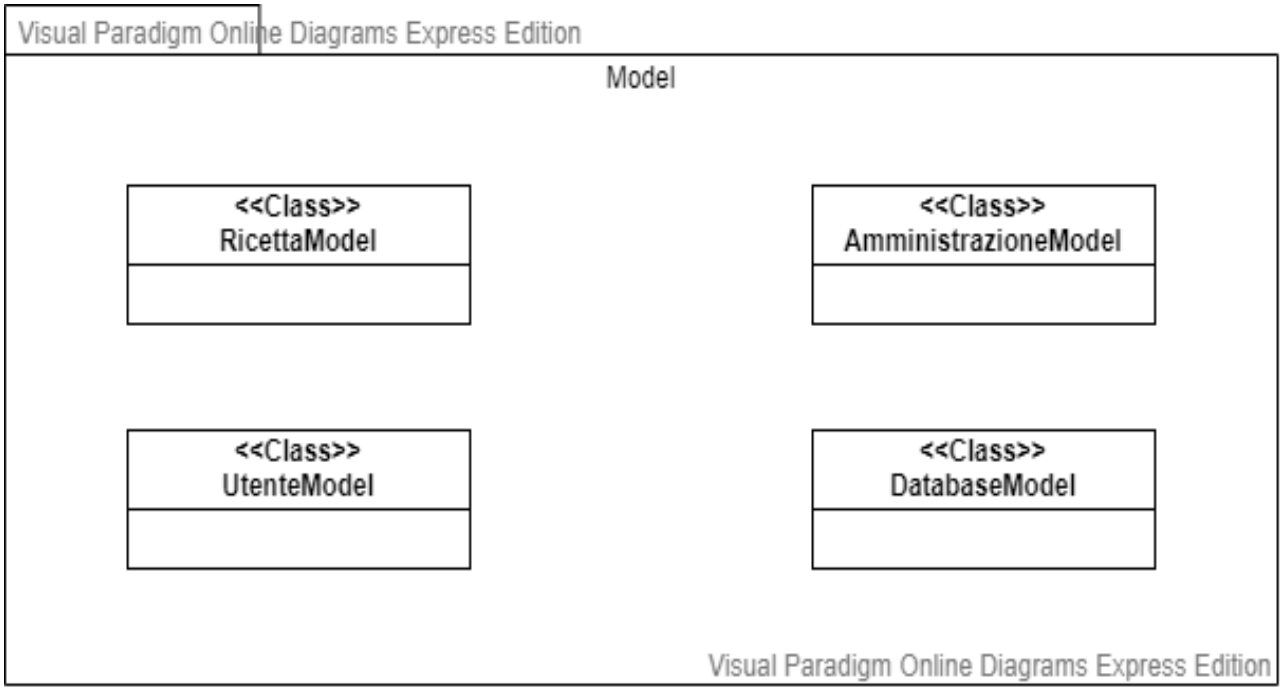


3.2 Package bean



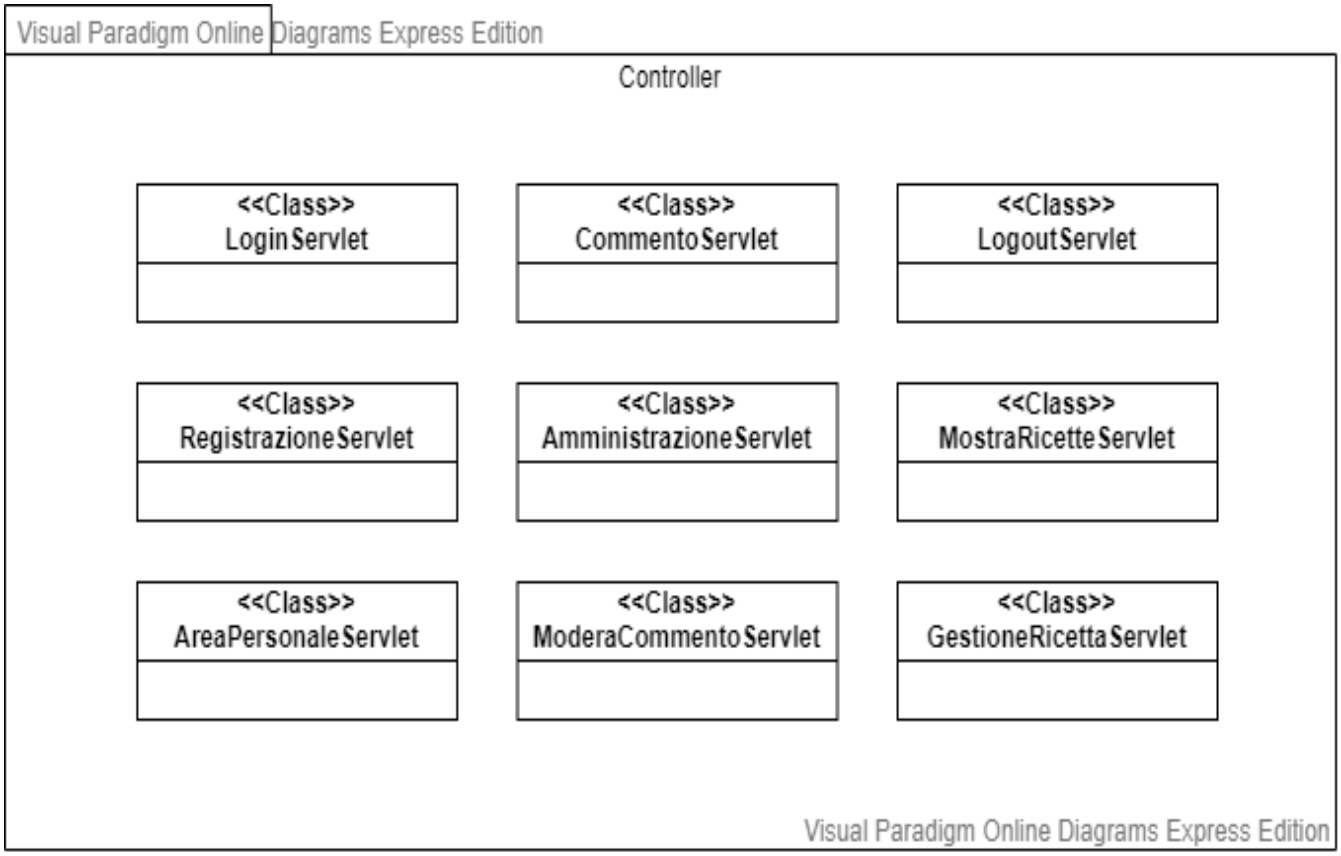
Classe:	Descrizione:
RicettaBean.java	Questa classe rappresenta la ricetta.
UtenteBean.java	Questa classe rappresenta l'utente.
RuoloBean.java	Questa classe rappresenta il ruolo degli utenti
CommentoBean.java	Questa classe rappresenta il commento di una ricetta.
CategoriaBean.java	Questa classe rappresenta la categoria di ricetta.
IngredienteBean.java	Questa classe rappresenta l'ingrediente di una ricetta.

Package model



Classe:	Descrizione:
RicettaModel.java	Questa classe contiene i metodi che permettono di effettuare inserimento, ricerca e cancellazione di una ricetta.
UtenteModel.java	Questa classe contiene i metodi che permettono di registrare, cercare o far accedere un Utente
AmministrazioneModel.java	Questa classe contiene i metodi che permette di creare nuove categorie o ruoli, eliminare o modifcare un commento
DatabaseModel.java	Questa classe contiene i metodi che permettono di effettuare operazioni di inserimento, ricerca o cancellazione sul Database

Package control



Classe:	Descrizione:
AreaPersonale.java	Questa servlet si occupa di ricevere l'operazione che si vuole effettuare e comunicare con il rispettivo model dell'operazione
LoginServlet.java	Questa servlet si occupa di ricevere i dati di login, elaborarli e decidere se consentire o meno l'accesso all'area personale.
CommentoServlet.java	Questa servlet si occupa di ricevere il commento scritto dall'utente ed inserirlo nel database attraverso il model di quest'ultimo

AmministrazioneServlet.java	Questa servlet si occupa di aggiungere un nuovo ruolo o una nuova categoria nel database in base ai dati inseriti dall'amministratore
ModeraCommentoServlet.java	Questa servlet si occupa di far modificare o eliminare un commento scritto da un utente
LogoutServlet.java	Questa servlet si occupa di invalidare la sessione per consentire il logout.
MostraRicetteServlet.java	Questa servlet si occupa di ottenere una lista di ricette da visualizzare in base ai filtri impostati da un utente
GestioneRicettaServlet.java	Questa servlet si occupa di modificare o eliminare una ricetta
RegistrazioneServlet.java	Questa servlet si occupa di ricevere i dati relativi alla registrazione e di elaborarli. Se i dati sono corretti usa i servizi di UtenteModel per completare la registrazione.

Class Interfaces

Utente Model

UtenteModel	
doSave	**Context:** UtenteModel: doSave(utente) **Pre:** utente!=null && utente.email !=null && utente.nome !=null && utente.cognome != null && utente.telefono !=null && utente.dataDiNascita !=null && utenti->forAll(u |u.email != utente.email); **Post:** utenti ->include(utente)
doDelete	**Context:** UtenteModel: doDelete(email) **Pre:** Utenti-> exists(u|u.email==email); **Post:** !Utenti-> exists(u|u.email==email);
doRetrieveByKey	**Context:** UtenteModel: doRetrieveByKey(email) **Pre:** Utenti-> exists(u|u.email==email); **Post:** Utenti->select (u|u.email == email);

Amministrazione Model

Amministrazione Model	
doAdd	**Context:** AmministrazioneModel: doAdd(ruolo||categoria) **Pre:** ruolo !=null || categoria !=null && && ruolo ->
forAll(r|r.nome !=ruolo.nome) || categoria ->
forAll(c|c.nome !=categoria.nome) **Post:** ruolo-> include(ruolo) || categoria->include(categoria)
doDelete	**Context:** AmministratoreModel: doDelete(commento) **Pre:** commento !=null && commento->
exists(c|c.id==id); **Post:** !Commento-> exists(c|c.id==id);
doModify	**Context:** AmministratoreModel: doModify(commento) **Pre:** commento !=null && Commento->
exists(c|c.id==id); **Post:** commento->merge(c.testo==commento.testo);

Ricetta Model

Ricetta Model	
doSave	Context: RicettaModel: doSave(Ricetta) Pre: Ricetta.codiceRicetta != null&& Ricetta.titolo != null && Ricetta.descrizione != null && !(Ricette->exist(r|r.codiceRicetta == Ricetta.codiceRicetta); Post: Ricette->exist(r|r.codiceRicetta== Ricetta.codiceRicetta);
doDelete	Context: RicettaModel: doDelete(codiceRicetta) Pre: Ricette->exist(r|r.codiceRicetta == Ricetta.codiceRicetta); Post: !Ricette->exist(r|r.codiceRicetta == Ricetta.codiceRicetta);
doRetrieveByKey	Context: RicettaModel: doRetrieveByKey(codiceRicetta) Pre: codiceRicetta !=null && Ricette-> exists(r|r.codiceRicetta == codiceRicetta); Post: Ricette->select(Ricette.codiceRicetta == codiceRicetta);

Servlet Ricetta

Servlet Ricetta	
aggiungiRicetta	**Pre:** ricetta.idRicetta != null && ricetta.titolo != null && ricetta.descrizione!= null && ricetta.autore != null && ricetta.categoria != null && ricetta.ingredienti != null && !(ricetta->exist(r| r.idRicetta == r.idRicetta); **Post:** Ricetta->exists(r| r.idRicetta== idRicetta);
ricercaRicetta	**Pre:** idRicetta !=null && Ricetta-> exists(r|r.idRicetta==idRicetta); **Post:** Ricetta->select(Ricetta.IdRicetta == idRicetta);
eliminaRicetta	**Pre:** idRicetta!=null && Ricetta-> exists(r|r.idRicetta==idRicetta); **Post:** !Ricetta-> exists(r| r.idRicetta==idRicetta);

Servlet Account

Servlet Account	
LoginUtente	**Pre:** UserName =! Null && Password != null; **Post:** utenti->exist(c | c.username == username && c.password == password)
RegistrazioneUtente	**Pre:** utente !=null && utente.email !=null && utente.nome !=null && utente.cognome != null && utente.telefono !=null && utente.dataDiNascita !=null && utenti->forAll(c |c.email != utente.email); **Post:** utenti ->include(utente)

5. Glossario

RAD: Documento di Analisi dei Requisiti.
DBMS: Sistema di gestione di basi di dati.
SDD: Documento di System Design.
ODD: Documento di Object Design.
Database: Insieme organizzato di dati persistenti.
Query: Termine utilizzato per indicare l'interrogazione da parte di un utente di un database
Utente Registrato: Il termine identifica l'utente che ha effettuato la registrazione sul sistema.