Object Design Document – SDD

Cognome | Nome	Matricola
Varone Giulio	0512104144
Romano Antonio	0512104768

Data	Versione	Descrizione	Autore
28/01/2019	0.1	Inizio documento	Giulio Varone
08/02/2019	0.2	Revisione documento	Antonio Romano
18/02/2019	0.3	Documento finale	Entrambi

Indice

1. Introduction

1.1 Object Design Trade-offs

Comprensibilita' vs Tempo

Il codice deve essere quanto più comprensibile possibile per facilitare future modifiche. Il codice deve essere quindi commentato opportunatamente, ma ciò richiederà più tempo per lo sviluppo.

Sicurezza vs Efficienza

La sicurezza rappresenta un elemento importante nel sistema, come indicato anche nei requisiti non funzionali. Pero', dati i tempi di sviluppo limitati, ci limiteremo ad implementare funzioni di sicurezza basati su username e password

Response Time vs Hardware

Il tempo di risposta rappresenta un fattore importante nel sistema, in particolare su determinate funzionalità. Tutto ciò dipenderà però anche dal tipo di hardware su cui verrà fatto eseguire il sistema

1.2 Interface documentation guidelines

Naming conventions

Gli sviluppatori seguiranno le seguenti linee guida per la definizione delle interfacce :

- Classi Java e Servlet :
    - I nomi dovranno iniziare con la lettera maiuscola

    - Se il nome contiene più parole, ognuna di esse dovrà iniziare con una lettera maiuscola
    - I nomi dovranno corrispondere alle informazioni e/o funzionalità che offre quella classe o Servlet
- Metodi :
    - I nomi dovranno iniziare con la lettera minuscola
    - Se il nome contiene più parole, ognuna di esse dovrà iniziare con la lettera maiuscola
    - I nomi dovranno corrispondere alle informazioni e/o funzionalità che offre quel metodo. Si utilizzerà un verbo più eventualmente aggettivi
    - I nomi dei metodi per ottenere e settare attributi seguiranno la regola di nominazione 'get[nomeattributo]' e 'set[nomeattributo]'
- Variabili :
    - I nomi delle variabili dovranno iniziare con la lettera minuscola
    - Se il nome contiene più parole, ognuna di esse dovrà iniziare con la lettera maiuscola

1.3 References

RAD : Requirement Analysis Document

SDD : System Design Document

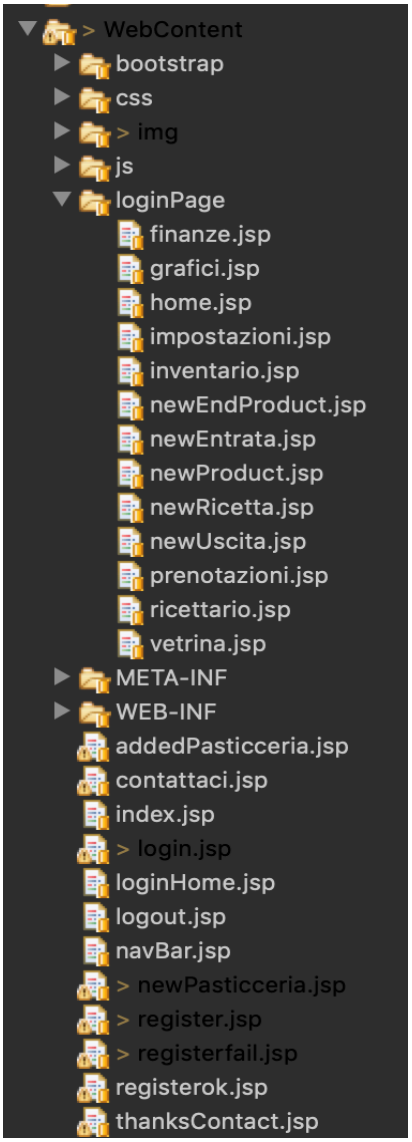
ODD : Object Design Document

2. Packages

Il sistema è diviso in packages nel modo seguente :

- webcontent
- bean
- control
- model
- test

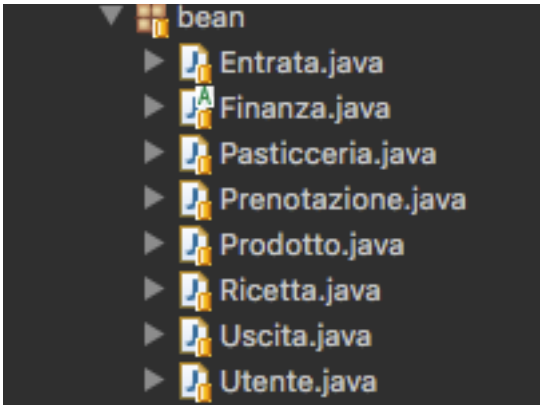
2.1 webcontent



Classe	Descrizione
finanze	Pagina che mostra all'utente le economie
grafici	Pagina che mostra all'utente i grafici
home	Pagina che mostra all'utente la home personale
impostazioni	Pagina che mostra all'utente le impostazioni dell'utente
inventario	Pagina che mostra all'utente l'inventario della pasticceria
newEndProduct	Pagina che mostra all'utente l'aggiunta di un nuovo prodotto finito
newEntrata	Pagina che mostra all'utente l'aggiunta di una nuova entrata
newProduct	Pagina che mostra all'utente l'aggiunta di un nuovo prodotto di inventario
newRicetta	Pagina che mostra all'utente l'aggiunta di una nuova ricetta
newUscita	Pagina che mostra all'utente l'aggiunta di una nuova uscita
prenotazioni	Pagina che mostra all'utente le prenotazioni di una pasticceria
ricettario	Pagina che mostra all'utente il ricettario della pasticceria
vetrina	Pagina che mostra all'utente i prodotti in vendita dalla pasticceria
login	Pagina che mostra all'utente il modulo di accesso al sistema

register	Pagina che mostra all'utente il modulo di registrazione al sistema

2.2 bean

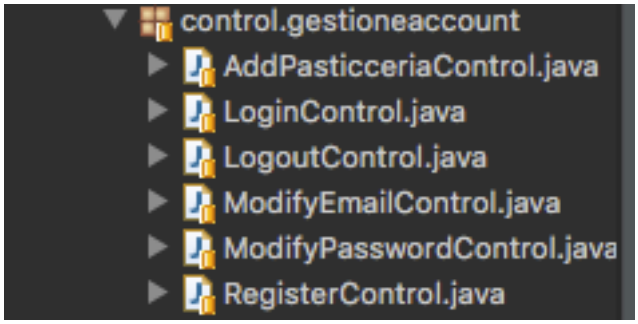


Classe	Descrizione
Entrata	Classe che rappresenta le informazioni di un'entrata economica
Finanza	Classe che rappresenta le informazioni di una finanza economica
Pasticceria	Classe che rappresenta le informazioni di una pasticceria
Prenotazione	Classe che rappresenta le informazioni di una prenotazione di un cliente
Prodotto	Classe che rappresenta le informazioni di un prodotto di inventario
Ricetta	Classe che rappresenta le informazioni di una ricetta
Uscita	Classe che rappresenta le informazioni di un'uscita economica

Utente	Classe che rappresenta le informazioni di un utente

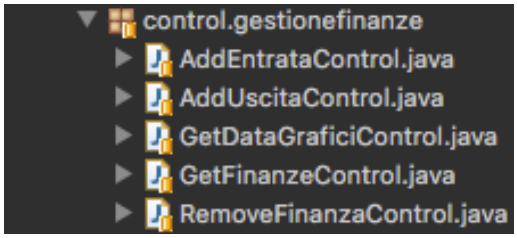
2.3 control

2.3.1 control.gestioneaccount



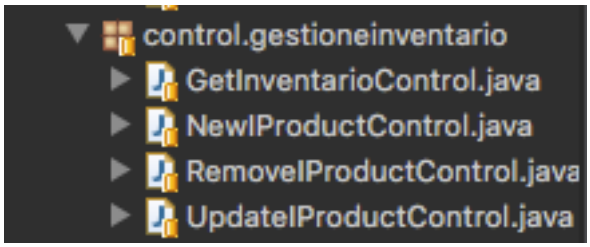
Classe	Descrizione
AddPasticceriaControl	Servlet che gestisce l'aggiunta di una pasticceria ad un utente
LoginControl	Servlet che gestisce l'accesso ad un utente
LogoutControl	Servlet che gestisce l'uscita dal sistema di un utente
ModifyEmailControl	Servlet che gestisce la modifica dell' email di un utente
ModifyPasswordControl	Servlet che gestisce la modifica della password di un utente
RegisterControl	Servlet che gestisce la registrazione di un utente

2.3.2 control.gestionefinanze



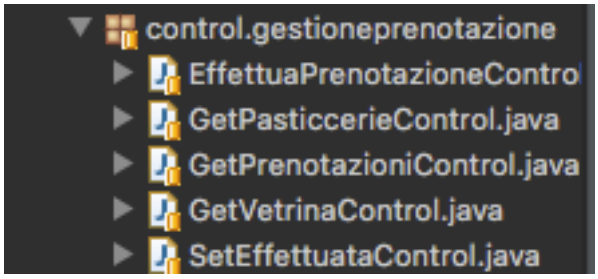
Classe	Descrizione
AddEntrataControl	Servlet che gestisce l'aggiunta di una entrata economica di una pasticceria
AddUscitaControl	Servlet che gestisce l'aggiunta di una uscita economica di una pasticceria
GetDataGraficiControl	Servlet che gestisce la visualizzazione dei grafici delle finanze di una pasticceria
GetFinanzeControl	Servlet che gestisce la visualizzazione delle finanze di una pasticceria
RemoveFinanzaControl	Servlet che gestisce la rimozione di una finanza di una pasticceria

2.3.3 control.gestioneinventario



Classe	Descrizione
GetInventarioControl	Servlet che gestisce la visualizzazione dei prodotti di inventario
NewIProductControl	Servlet che gestisce l'aggiunta di un nuovo prodotto di inventario
RemoveIProductControl	Servlet che gestisce la rimozione di un prodotto dall'inventario
UpdateIProductControl	Servlet che gestisce la modifica di un prodotto di inventario

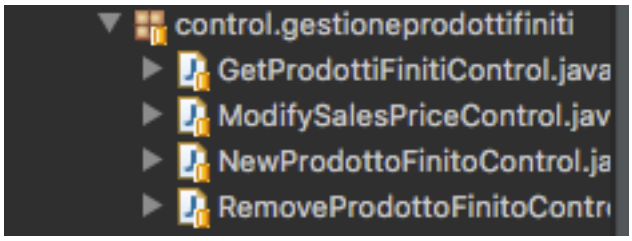
2.3.4 control.gestioneprenotazione



Classe	Descrizione
EffettuaPrenotazioneControl	Servlet che gestisce la prenotazione di prodotti dalla pasticceria
GetPasticcerieControl	Servlet che gestisce la visualizzazione delle pasticcerie
GetPrenotazioniControl	Servlet che gestisce la visualizzazione delle prenotazioni ricevute
GetVetrinaControl	Servlet che gestisce la visualizzazione dei prodotti in vendita da una pasticceria

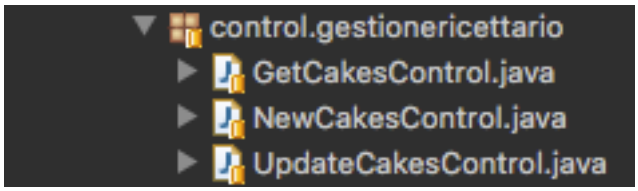
SetEffettuataControl	Servlet che gestisce l'effettuazione di una ricetta

2.3.5 control.gestioneprodottifiniti



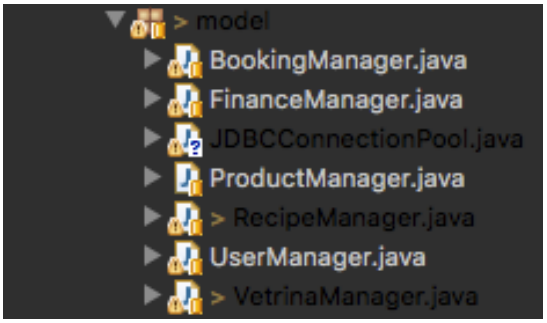
Classe	Descrizione
GetProdottiFinitiControl	Servlet che gestisce la visualizzazione dei prodotti finiti di una pasticceria
ModifySalesPriceControl	Servlet che gestisce la modifica del prezzo di vendita di un prodotto finito
NewProdottoFinitoControl	Servlet che gestisce l'aggiunta di un nuovo prodotto finito
RemoveProdottoFinitoControl	Servlet che gestisce la rimozione di un prodotto finito

2.3.6 control.gestionericettario



Classe	Descrizione
GetCakesControl	Servlet che gestisce la visualizzazione delle ricette di una pasticceria
NewCakesControl	Servlet che gestisce l'aggiunta di una nuova ricetta
UpdateCakesControl	Servlet che gestisce la modifica di una ricetta

2.4 model

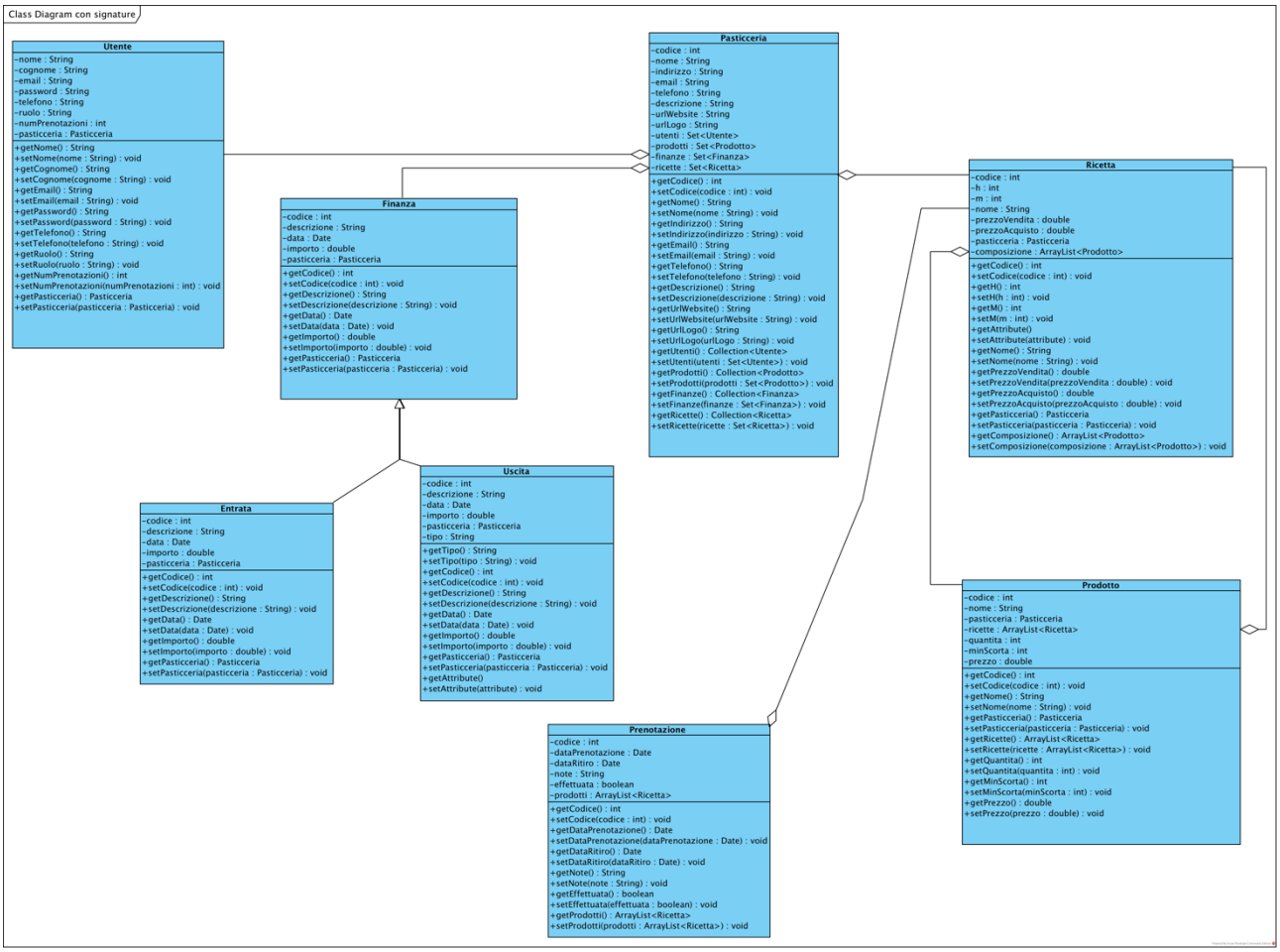


Classe	Descrizione
BookingManager	Classe che implementa le operazioni riguardanti la prenotazione
FinanceManager	Classe che implementa le operazioni riguardanti le economie
ProductManager	Classe che implementa le operazioni riguardanti i prodotti di inventario

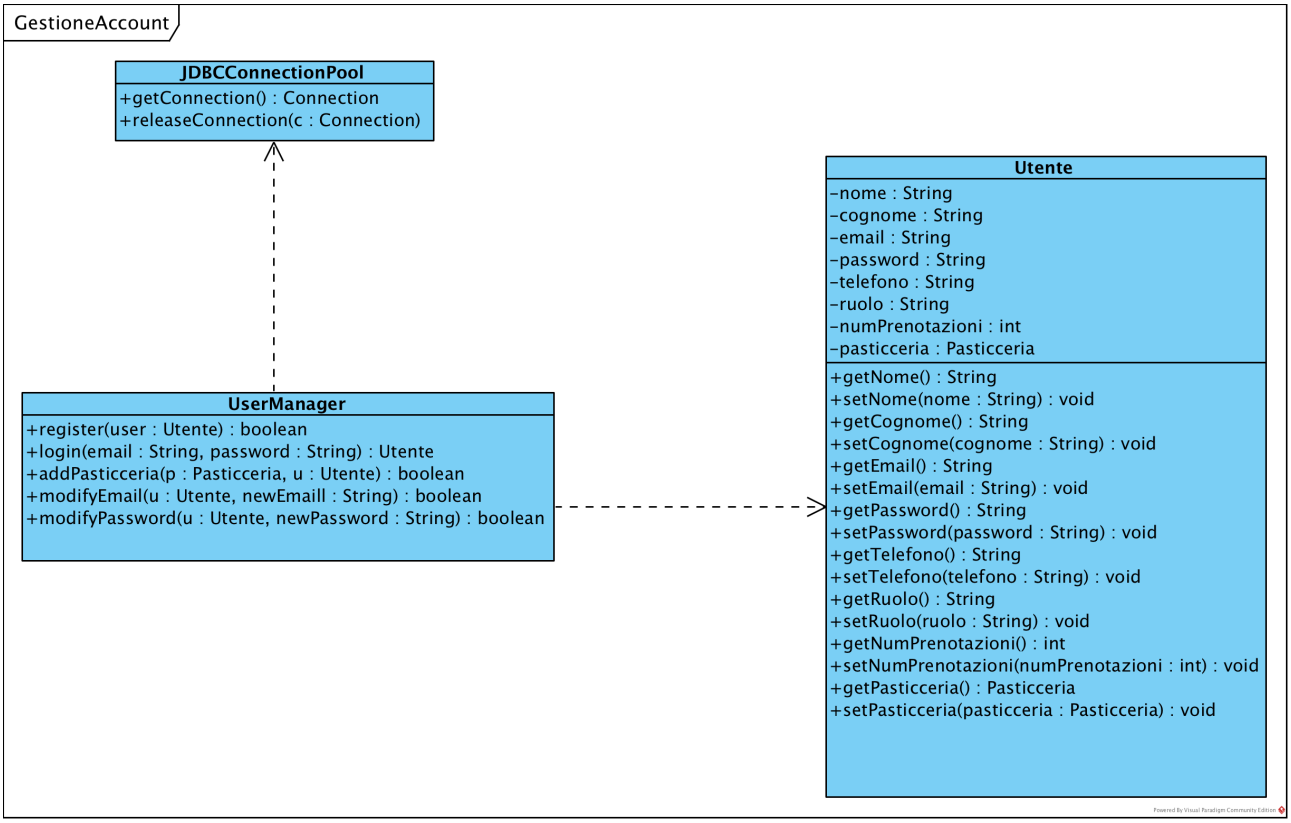
RecipeManager	Classe che implementa le operazioni riguardanti le ricette di una pasticceria
UserManager	Classe che implementa le operazioni riguardanti gli account
VetrinaManager	Classe che implementa le operazioni riguardanti i prodotti finiti di una pasticceria
JDBCConnectionPool	Classe che implementa il pattern Object pool per fornire le connessioni al database

3. Class interfaces

1. INTERFACCE BEAN

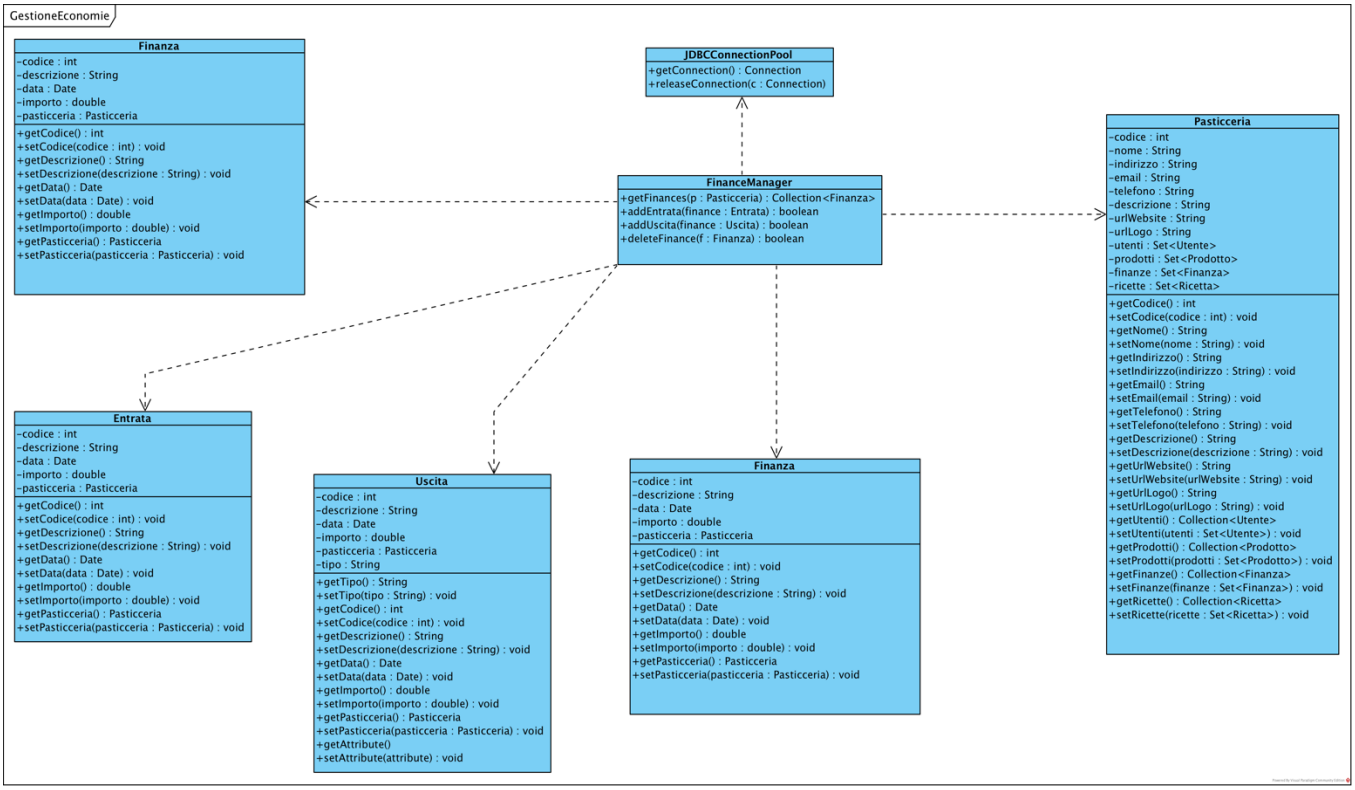


2. UserManager



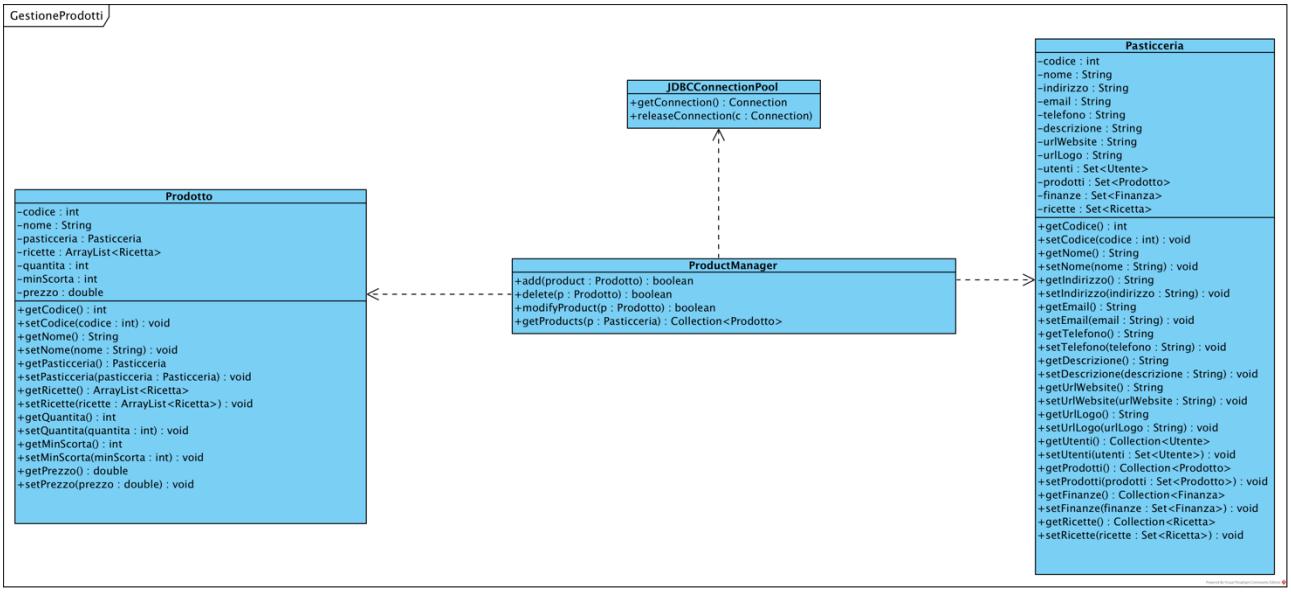
Nome :	UserManager
Pre-condizioni	**context** UserManager::register(user: Utente) : boolean **pre:** user!=null **context** UserManager::login(email:String, password:String) : Utente **pre:** email != null && password != null **context** UserManager::addPasticceria(p: Pasticceria, u: Utente) : boolean **pre:** p!=null && u!=null **context** UserManager::modifyEmail(u: Utente, newEmail:String) : boolean **pre:** u!=null && newEmail != null && ! (u.getEmail().equals(newEmail)) **context** UserManager::modifyPassword(u: Utente, newPassword:String) : boolean **pre:** u!=null && newPassword != null
Post-condizioni	**context** UserManager::addPasticceria(p: Pasticceria) : boolean **post:** user.getPasticceria() = p **context** UserManager::modifyEmail(u: Utente, newEmail: String) : boolean **post:** u.getEmail() = newEmail **context** UserManager::modifyPassword(u: Utente, newPassword:String) : boolean **post:** u.getPassword() = newPassword

3. FinanceManager



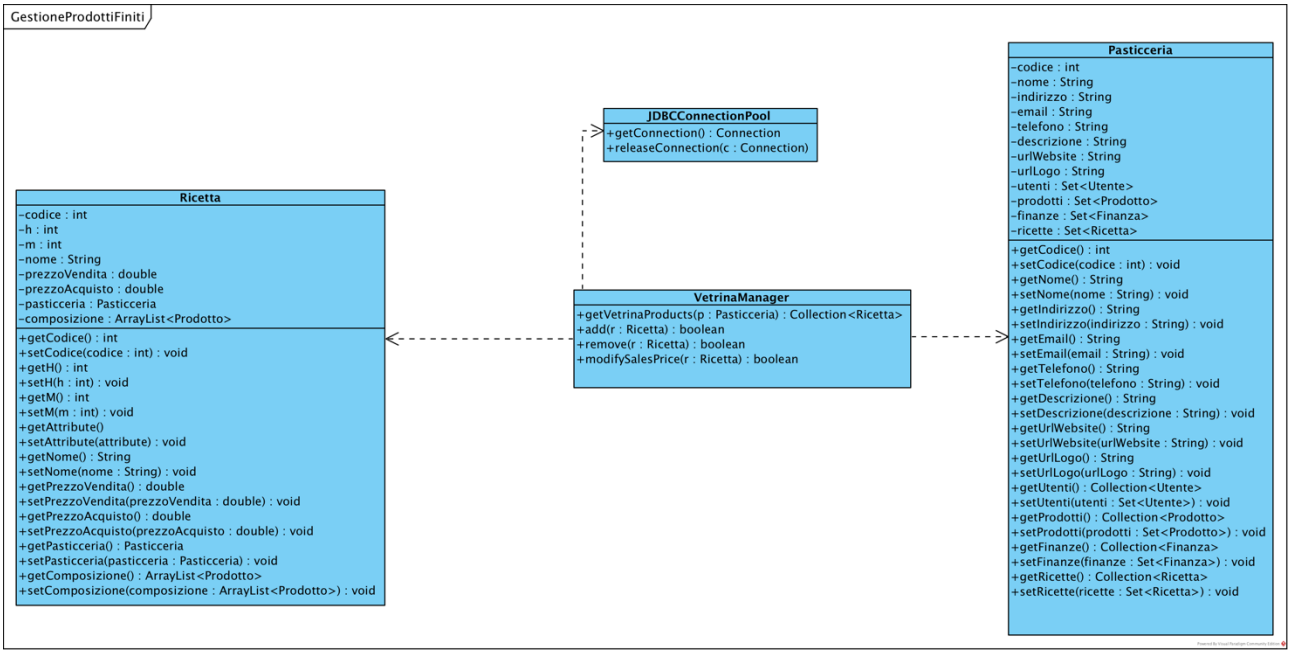
Nome :	FinanceManager
Pre–condizioni	**context** FinanceManager::getFinances(p: Pasticceria) : Collection<Finanza> **pre:** p!=null **context** FinanceManager::addEntrata(finance: Entrata) : boolean **pre:** finance!=null **context** FinanceManager::addUscita(finance: Uscita) : boolean **pre:** finance!=null **context** FinanceManager::deleteFinance(f: Finanza) : boolean **pre:** f!=null
Post–condizioni	**context** FinanceManager::getFinances(p: Pasticceria) : Collection<Finanza> **post:** p!=null **context** FinanceManager::addEntrata(finance: Entrata) : boolean **post:** finance.getPasticceria().getFinanze().size = @pre.finance.getPasticceria().getFinanze().size() + 1 **context** FinanceManager::addUscita(finance: Uscita) : boolean **post:** finance.getPasticceria().getFinanze().size = @pre.finance.getPasticceria().getFinanze().size() + 1 **context** FinanceManager::deleteFinance(f: Finanza) : boolean **post:** finance.getPasticceria().getFinanze().size = @pre.finance.getPasticceria().getFinanze().size() – 1

4. ProductManager



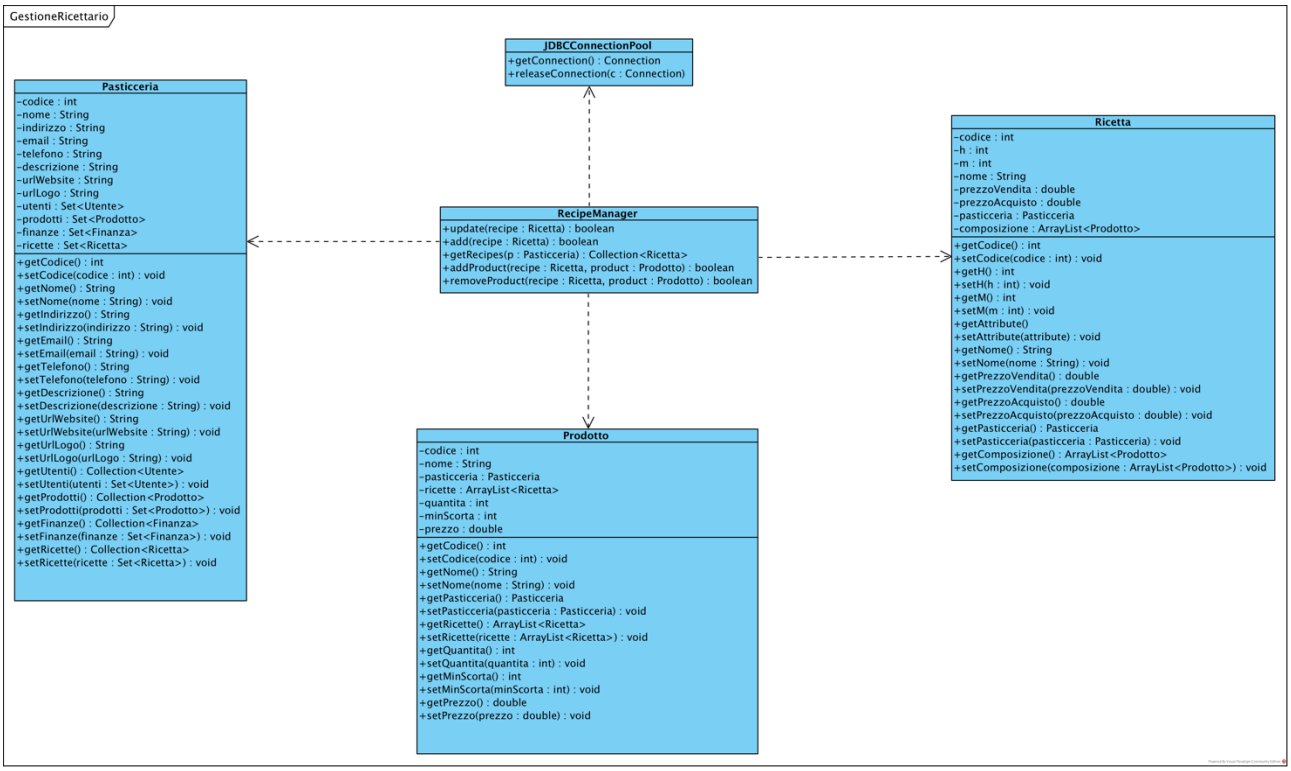
Nome :	ProductManager
Pre–condizioni	**context** ProductManager::add(product: Prodotto) : boolean **pre:** product!=null **context** ProductManager::delete(p: Prodotto) : boolean **pre:** p!=null **context** ProductManager::modifyProduct(p: Prodotto) : boolean **pre:** p!=null **context** ProductManager::getProducts(p: Pasticceria) : boolean **pre:** p!=null
Post–condizioni	**context** ProductManager::add(product: Prodotto) : boolean **post:** product.getPasticceria().getProdotti().size() = @pre. product.getPasticceria().getProdotti().size() + 1 **context** ProductManager::delete(p: Prodotto) : boolean **post:** product.getPasticceria().getProdotti().size() = @pre. product.getPasticceria().getProdotti().size() – 1

5. VetrinaManager



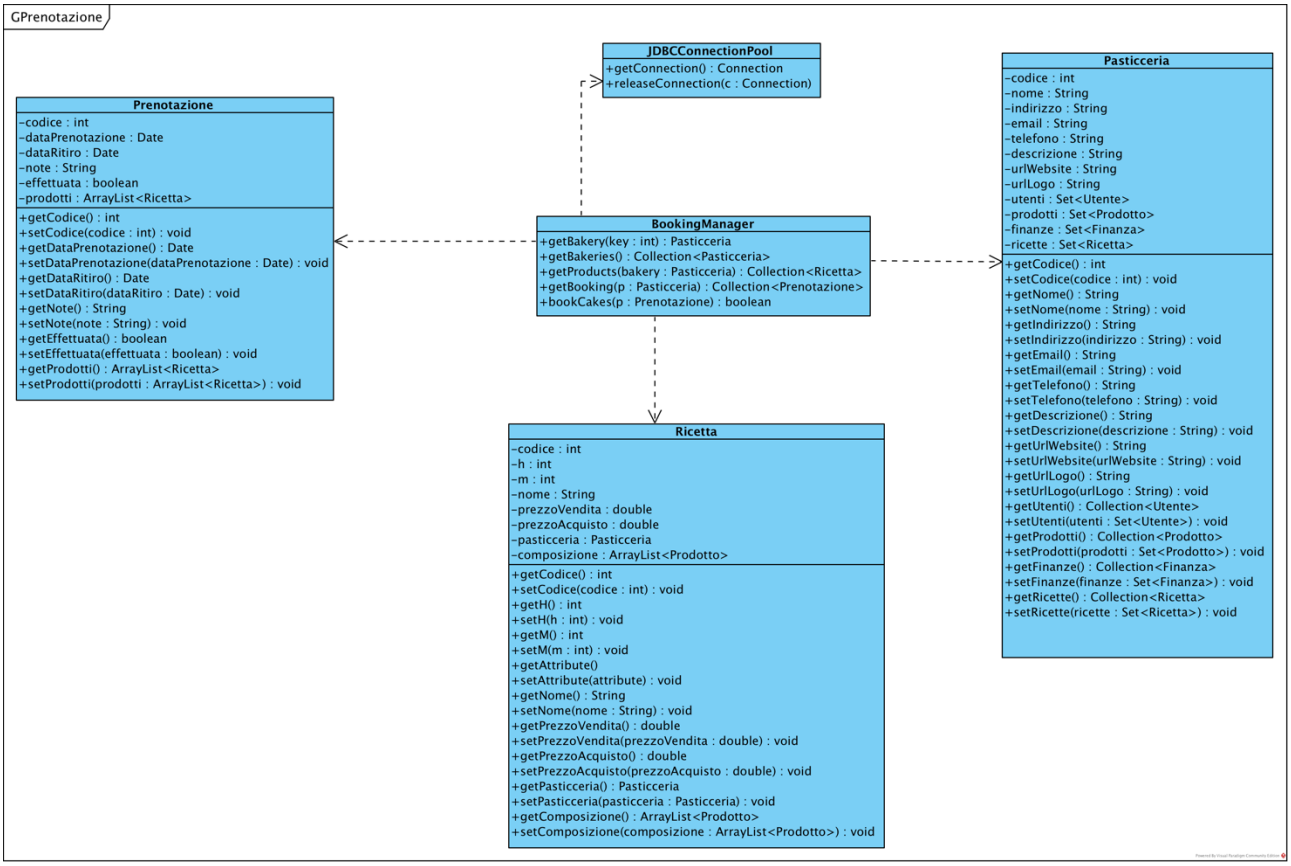
Nome :	VetrinaManager
Pre–condizioni	**context** VetrinaManager::getVetrinaProducts(p: Pasticceria) : Collection<Ricetta> **pre:** p!=null **context** VetrinaManager::add(r: Ricetta) : boolean **pre:** r!=null **context** VetrinaManager::remove(r: Ricetta) : boolean **pre:** r!=null **context** VetrinaManager::modifySalesPrice(r: Ricetta) : boolean **pre:** r!=null
Post–condizioni	**context** VetrinaManager::add(r: Ricetta) : boolean **post:** r.getPasticceria().getFinanze().size() = @pre.r.getPasticceria().getFinanze().size() + 1 **context** VetrinaManager::remove(r: Ricetta) : boolean **post:** r.getPasticceria().getFinanze().size() = @pre.r.getPasticceria().getFinanze().size() – 1

6. RecipeManager



Nome :	RecipeManager
Pre–condizioni	**context** RecipeManager::update(recipe: Ricetta) : boolean **pre:** recipe!=null **context** RecipeManager::add(recipe: Ricetta) : boolean **pre:** recipe!=null **context** RecipeManager::getRecipes(p: Pasticceria) : Collection<Ricetta> **pre:** p!=null **context** RecipeManager::addProduct(recipe: Ricetta, product: Prodotto) : boolean **pre:** recipe!=null && product!=null **context** RecipeManager::removeProduct(recipe: Ricetta, product: Prodotto) : boolean **pre:** recipe!=null && product!=null
Post–condizioni	**context** RecipeManager::add(recipe: Ricetta) : boolean **post:** recipe.getPasticceria().getRicette().size() = @pre. recipe.getPasticceria().getRicette().size() +1 **context** RecipeManager::addProduct(recipe: Ricetta, product: Prodotto) : boolean **post:** recipe.getComposizione().size() = @pre.recipe.getComposizione().size() + 1 **context** RecipeManager::removeProduct(recipe: Ricetta, product: Prodotto) : boolean **post:** recipe.getComposizione().size() = @pre.recipe.getComposizione().size() – 1

7. BookingManager

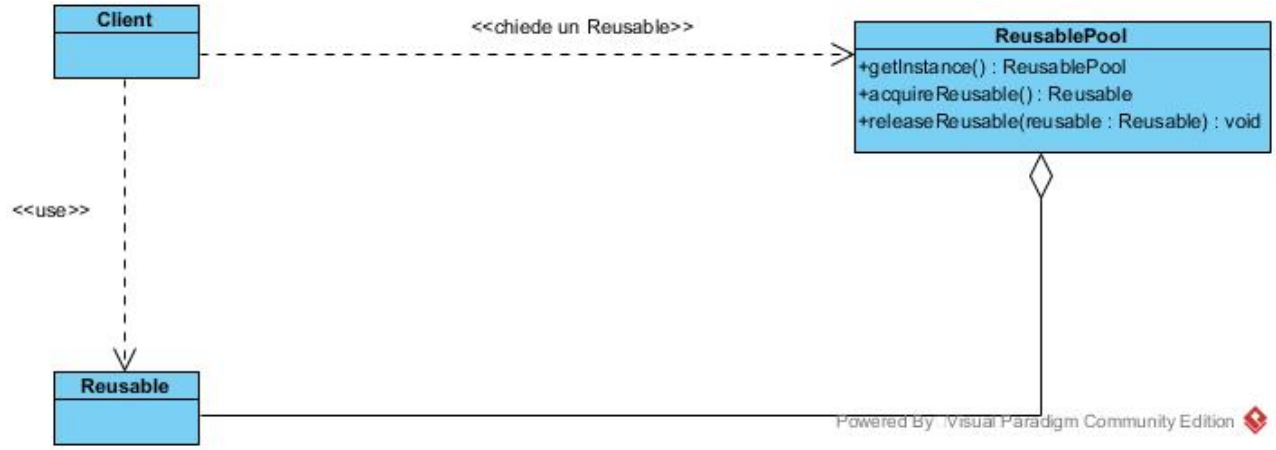


Nome :	BookingManager
Pre–condizioni	**context** BookingManager::getBakery(key: int) : Pasticceria **pre:** key > 0 **context** BookingManager::getProducts(bakery: Pasticceria) : Collection<Ricetta> **pre:** bakery!=null **context** BookingManager::getBooking(p: Pasticceria) : Collection<Prenotazione> **pre:** p!=null **context** BookingManager::bookCakes(p: Prenotazione) : boolean **pre:** p!=null
Post–condizioni	**context** BookingManager::getBakery(key: int) : Pasticceria **post :** pasticceria != null

4. Design pattern

Object pool pattern

L'Object pool pattern è un design pattern creazionale che usa un insieme di oggetti inizializzati pronti per l'uso mantenuti in una "pool", che si occupa di allocarli e de–allocherà su richiesta. Il client della pool invierà richiesta a un oggetto nella pool ed eseguirà operazioni sull'oggetto ritornato. Quando il client ha finito, ritorna l'oggetto alla pool che lo de–allocherà.



Utilizzo: L'Object Pool Pattern sarà utilizzato per gestire le connessioni con il database. Più precisamente, un oggetto Model richiederà connessioni al DriverManagerPool che ritornerà un oggetto Connection, l'oggetto Model effettuerà operazioni con l'oggetto connection e successivamente richiederà al DriverManagerPool la de–allocazione.