ODD

Object Design Document

Partecipanti

Cognome	Nome	Matricola
Campofreda	Alessio	05121 05492
Caccia	Raffaele	05121 05514
Pincivalle	Rolando Antonio	05121 05192
Spera	Francesco	05121 05408

Indice

1. INTRODUZIONE

1.1 OBJECT DESIGN TRADE-OFFS

- *Comprensibilità vs Tempo:*

Il codice del sistema deve essere comprensibile, in modo da facilitare la fase di testing ed eventuali future modifiche da apportare. Al fine di rispettare queste linee guida il codice sarà integrato da commenti volti a migliorarne la leggibilità; tuttavia questo richiederà una maggiore quantità di tempo necessario per lo sviluppo del nostro progetto.

- *Prestazioni vs Costi:*

Dal momento che il budget allocato è spendibile principalmente in risorse umane e non consente l'acquisto di tecnologie proprietarie specifiche verranno utilizzati template open source e componenti hardware di nostra proprietà.

- *Interfaccia vs Usabilità:*

Verrà realizzata un'interfaccia grafica chiara e concisa, usando form e pulsanti predefiniti che hanno lo scopo di rendere semplice l'utilizzo del sistema da parte dell'utente finale.

- *Sicurezza vs Efficienza:*

La sicurezza rappresenta uno degli aspetti principali del sistema. Tuttavia, a causa di tempi di sviluppo molto limitati, ci limiteremo ad implementare sistemi di sicurezza basati su email e password.

1.2 LINEE GUIDA PER LA DOCUMENTAZIONE DELLE INTERFACCE

Naming convention:
Per la documentazione delle interfacce bisognerà utilizzare nomi:

- Descrittivi
- Pronunciabili
- Utilizzando solo caratteri consentiti (a-z, A-Z, 0-9)

Variabili:

- I nomi delle variabili dovranno iniziare con la lettera minuscola, le parole successive con la lettera maiuscola.

Metodi:

- I nomi dei metodi dovranno iniziare con la lettera minuscola, le parole successive con la lettera maiuscola, secondo la “Camel Notation”.
- Il nome del metodo sarà costituito da un verbo che ne identifica l’azione seguito da un sostantivo, eventualmente aggettivato.
- Il nome dei metodi accessori e modificatori seguirà, rispettivamente, i pattern getNomeVariabile e setNomeVariabile.

Classi Java e pagine JSP:

- I nomi delle classi dovranno iniziare con la lettera maiuscola cosi come le parole successive all’interno del nome.
- I nomi delle jsp dovranno iniziare con la lettera minuscola, le parole successive all’interno del nome con la lettera maiuscola.
- I nomi delle classi e delle jsp corrisponderanno alle informazioni e alle funzioni da loro fornite
- Le classi saranno strutturate prevedendo:
    - Dichiarazione della classe pubblica
    - Dichiarazione di variabili di classe
    - Costruttore
    - Metodi

Packages:

I nomi dei packages dovranno essere scritti in minuscolo concatenando insieme diversi sostantivi o sigle, separate dal carattere punto. Non sono ammessi caratteri speciali.

1.3 DEFINIZIONI, ACRONIMI E ABBREVIAZIONI

RAD: Requirements Analysis Document
SDD: System Design Document
ODD: Object Design Document
JSP: Java Server Page

1.4 RIFERIMENTI

Si rimanda ai documenti "RAD_MEDASSISTANT.pdf" e " SDD_MEDASSISTANT.pdf".

2. PACKAGES

Il sistema è diviso in package come segue:

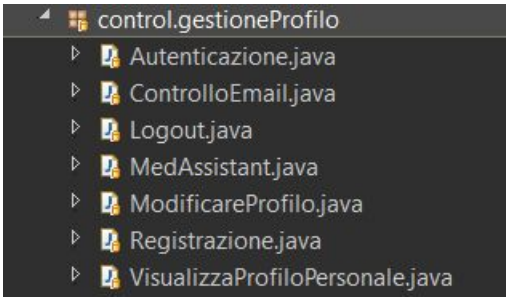
- controller
- model
- bean
- connectionPool
- presentation
- core
- test

2.1 CONTROLLER

Il package controller è suddiviso verticalmente in package come segue:

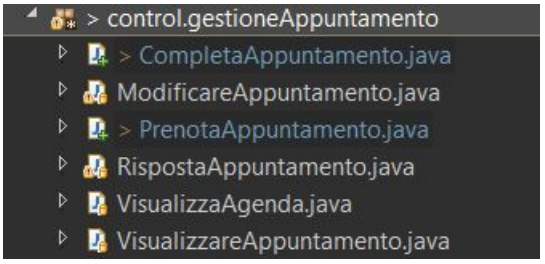
- gestioneProfilo
- gestioneAppuntamanento
- gestioneRicetta
- gestioneRicerca
- gestioneRecensione
- gestioneReferto
- gestioneCollegamento
- gestioneEmail

2.1.1 gestioneProfilo



Classe	Descrizione
Autenticazione	Servlet che permette di effettuare il login
ControlloEmail	Servlet che permette di verificare se un email è già presente nel database
Logout	Servlet che permette di effettuare il logout
MedAssistant	Servlet che indirizza alla home page
ModificareProfilo	Servlet che permette di modificare il profilo dei due tipi di account presenti
Registrazione	Servlet che permette di registrarsi
VisualizzaProfiloPersonale	Servlet che permette di visualizzare il profilo personale

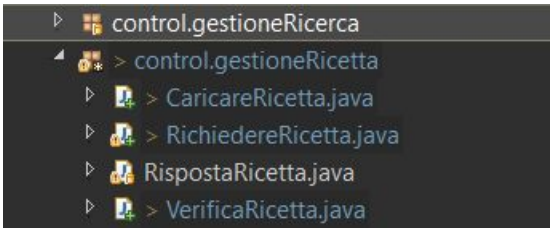
2.1.2 gestioneProfilo



Classe	Descrizione
CompletaAppuntamento	Servlet che permette di completare l'appuntamento
ModificareAppuntamento	Servlet che permette di modificare l'appuntamento
PrenotaAppuntamento	Servlet che permette di prenotare un appuntamento
RispostaAppuntamento	Servlet che permette di rispondere ad una richiesta di appuntamento

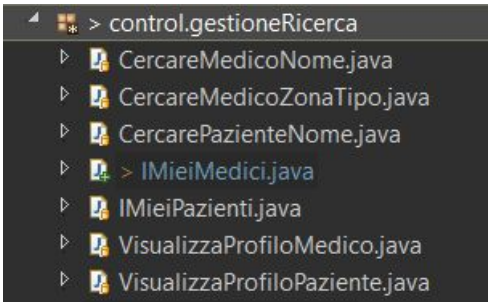
VisualizzaAgenda	Servlet che permette di visualizzare la propria agenda
VisualizzareAppuntamento	Servlet che permette di visualizzare le informazioni di un appuntamento

2.1.3 gestioneRicetta



Classe	Descrizione
CaricareRicetta	Servlet che permette di caricare una ricetta
RichiedereRicetta	Servlet che permette di richiedere una ricetta
RispostaRicetta	Servlet che permette di rispondere ad una richiesta di caricamento della ricetta
VerificaRicetta	Servlet che permette di cercare una ricetta

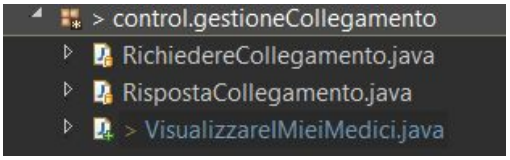
2.1.4 gestioneRicerca



Classe	Descrizione
CercareMedicoNome	Servlet che permette di cercare un medico per nome e/o cognome
CercareMedicoZonaTipo	Servlet che permette di cercare un medico in base alla zona e al tipo
CercarePazienteNome	Servlet che permette di cercare un paziente per nome e/o cognome
IMieiMedici	Servlet che permette di visualizzare i medici collegati e di cercarli
IMieiPazienti	Servlet che permette di visualizzare i pazienti collegati e di cercarli
VisualizzaProfiloMedico	Servlet che permette di visualizza il profilo di un medico dal punto di vista del paziente

VisualizzaProfiloPaziente	Servlet che permette di visualizza il profilo di un paziente dal punto di vista del medico

2.1.5 gestioneCollegamento



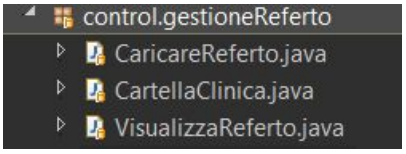
Classe	Descrizione
RichiedereCollegamento	Servlet che permette di richiedere un collegamento con un medico
RispostaCollegamento	Servlet che permette di rispondere ad una richiesta di collegamento
VisualizzareImieiMedici	Servlet che permette di visualizzare tutti i medici collegati ad un paziente

2.1.6 gestioneVotazione

Classe	Descrizione

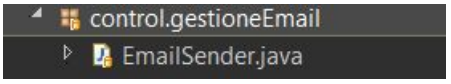
VotaMedico	Servlet che permette al paziente di votare un medico

2.1.7 gestioneReferto



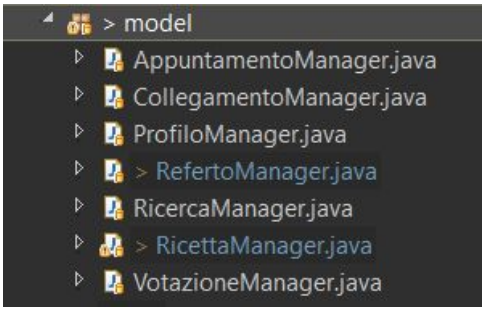
Classe	Descrizione
CaricaReferto	Servlet che permette di caricare un referto
CartellaClinica	Servlet che permette di visualizzare tutti i referti
VisualizzaReferto	Servlet che permette di visualizzare le informazioni di un referto

2.1.8 gestioneEmail



Classe	Descrizione
EmailSender	Servlet che permette di gestire l'invio delle mail

2.2 MODEL



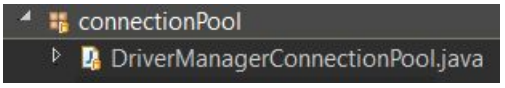
Classe	Descrizione
AppuntamentoManager	Classe che implementa le operazioni dell'appuntamento
CollegamentoManager	Classe che implementa le operazioni del collegamento
ProfiloManager	Classe che implementa le operazioni dell'utente
RefertoManager	Classe che implementa le operazioni del referto
RicercaManager	Classe che implementa le operazioni della ricerca
RicettaManager	Classe che implementa le operazioni della ricetta
VotazioneManager	Classe che implementa le operazioni della votazione

2.3 BEAN

Classe	Descrizione
Account	Classe che rappresenta le informazioni generali di un account

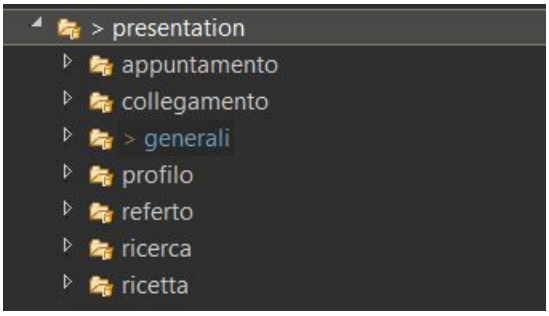
Appointment	Classe che rappresenta le informazioni di un appuntamento
Doctor	Classe che rappresenta le informazioni di un medico
Link	Classe che rappresenta le informazioni di un collegamento
MedicalReport	Classe che rappresenta le informazioni di un referto
Patient	Classe che rappresenta le informazioni di un paziente
Prescription	Classe che rappresenta le informazioni di una ricetta
Voting	Classe che rappresenta le informazioni di un voto

2.4 CONNECTIONPOOL

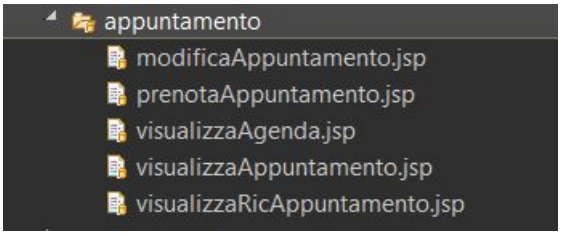


Classe	Descrizione
DriverManagerConnectionPool	Classe che implementa l'Object pool pattern, responsabile di fornire le connessioni al database

2.5 PRESENTATION



2.5.1 APPUNTAMENTO



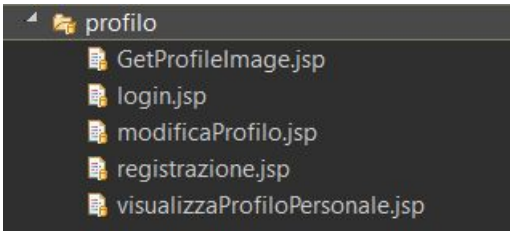
Pagina	Descrizione
modificaAppuntamento	Sezione in cui l'utente può modificare un appuntamento
prenotaAppuntamento	Sezione in cui un paziente può prenotare un appuntamento
visualizzaAgenda	Sezione in cui l'utente può visualizzare la sua personale agenda
visualizzaAppuntamento	Sezione in cui l'utente può visualizzare un appuntamento
visualizzaRicAppuntamento	Sezione in cui un medico può visualizzare la richiesta di appuntamento

2.5.2 COLLEGAMENTO

Pagina	Descrizione
visualizzaCollegamento	Sezione in cui il medico è possibile visualizzare la richiesta di collegamento

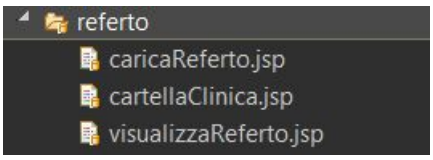
2.5.3 GENERALI

2.5.4 PROFILO



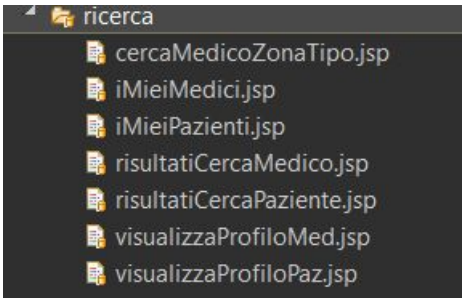
Pagina	Descrizione
login	Sezione in cui l'utente può accedere al sito
modificaProfilo	Sezione in cui l'utente può modificare il proprio profilo
registrazione	Sezione in cui l'utente può registrarsi al sito
visualizzaProfiloPersonale	Sezione in cui l'utente può visualizzare il suo profilo

2.5.5 REFERTO



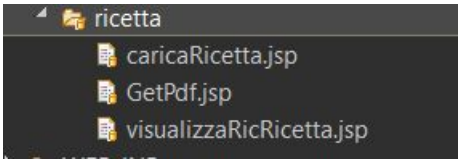
Pagina	Descrizione
caricaReferto	Sezione in cui è possibile caricare un referto
cartellaClinica	Sezione in cui è possibile visualizzare tutti i referti
visualizzaReferto	Sezione in cui è possibile visualizzare le informazioni di un referto

2.5.6 RICERCA



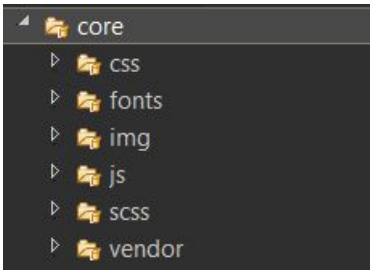
Pagina	Descrizione
cercaMedicoZonaTipo	Sezione in cui il paziente può cercare un medico in base alla zona e al tipo
iMieiMedici	Sezione in cui il paziente può visualizzare e cercare i medici con cui è collegato
iMieiPazienti	Sezione in cui il medico può visualizzare e cercare i pazienti con cui è collegato
risultatiCercaMedico	Sezione in cui il paziente visualizza i risultati della ricerca del medico
risultatiCercaPaziente	Sezione in cui il medico visualizza i risultati della ricerca del paziente
visualizzaProfiloMed	Sezione in cui il paziente visualizza il profilo del medico
visualizzaProfiloPaz	Sezione in cui il medico visualizza il profilo del paziente

2.5.7 RICETTA



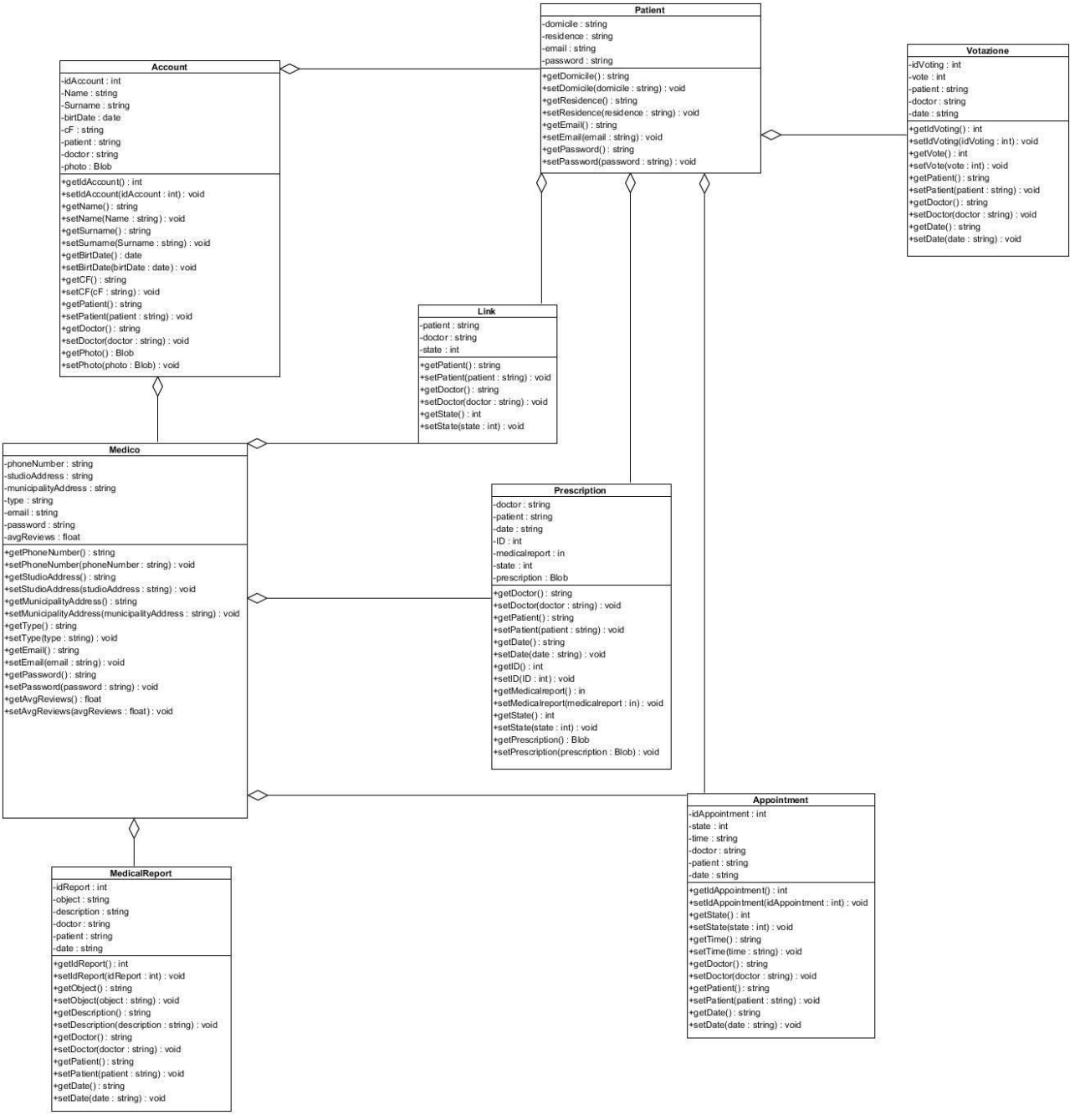
Pagina	Descrizione
caricaRicetta	Sezione in cui il medico può caricare la ricetta
visualizzaRicRicetta	Sezione in cui il medico visualizza la richiesta di caricamento di una ricetta

2.6 CORE

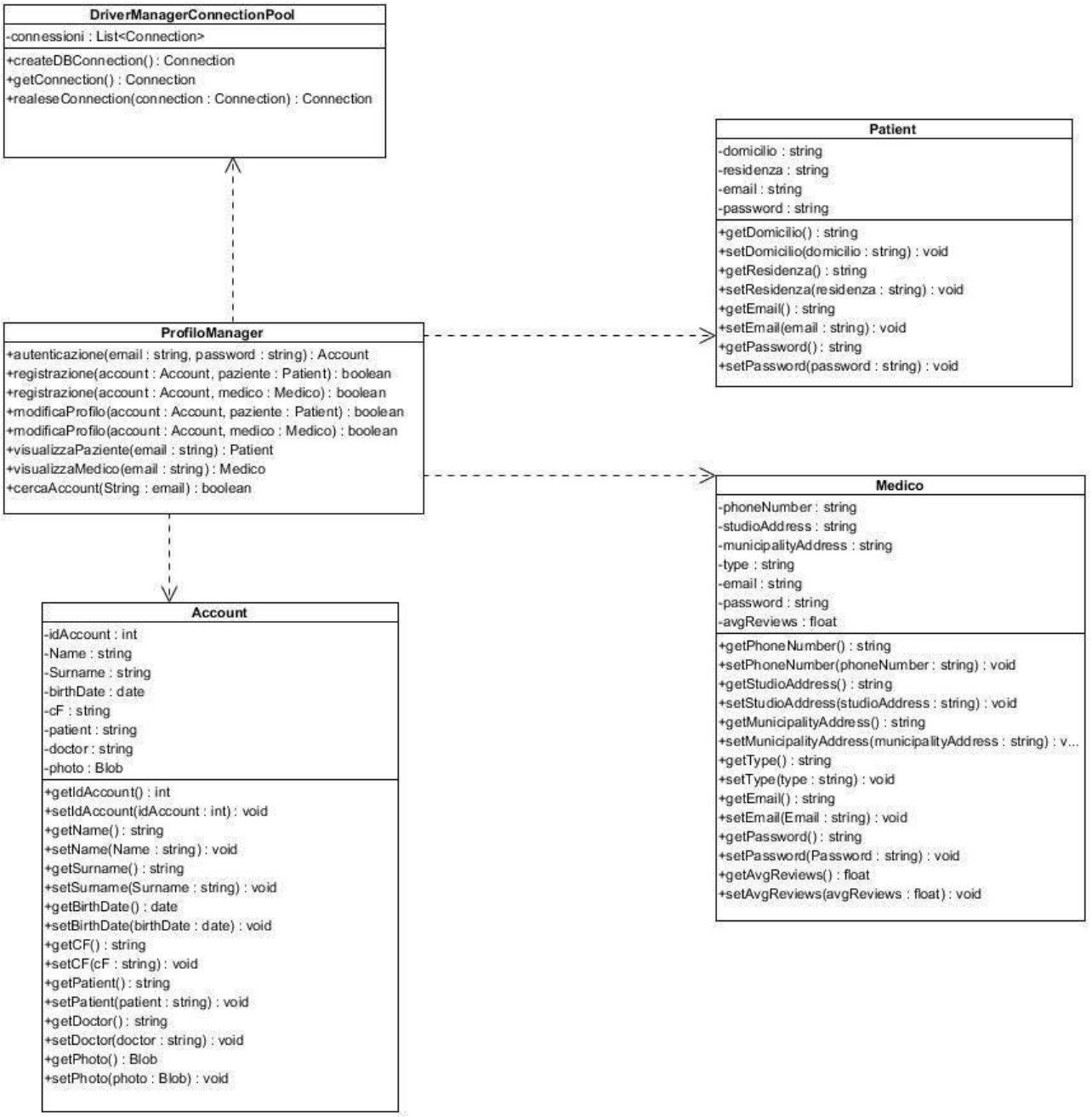


3. INTERFACCE DI CLASSE

3.1 INTERFACCE BEAN

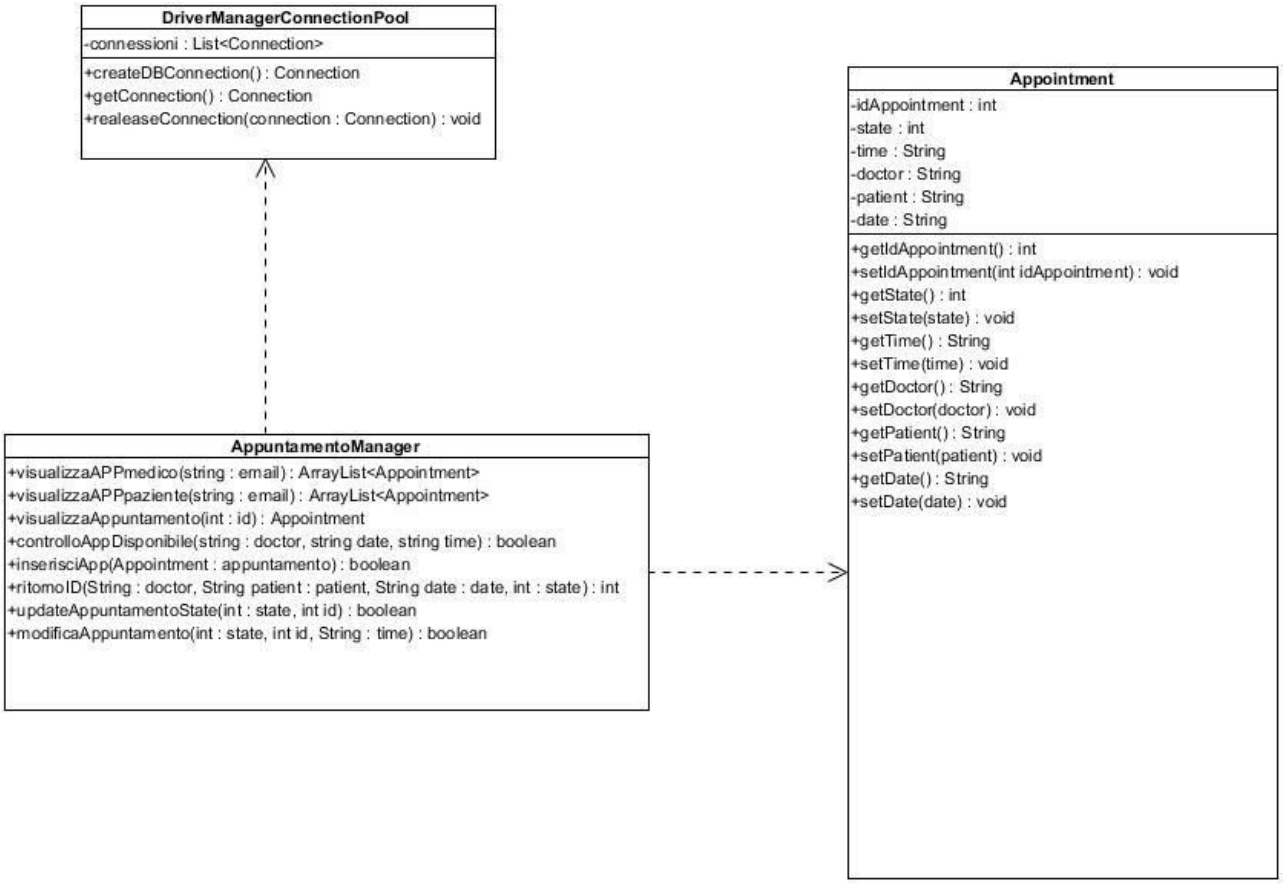


3.2 PROFILOMANAGER



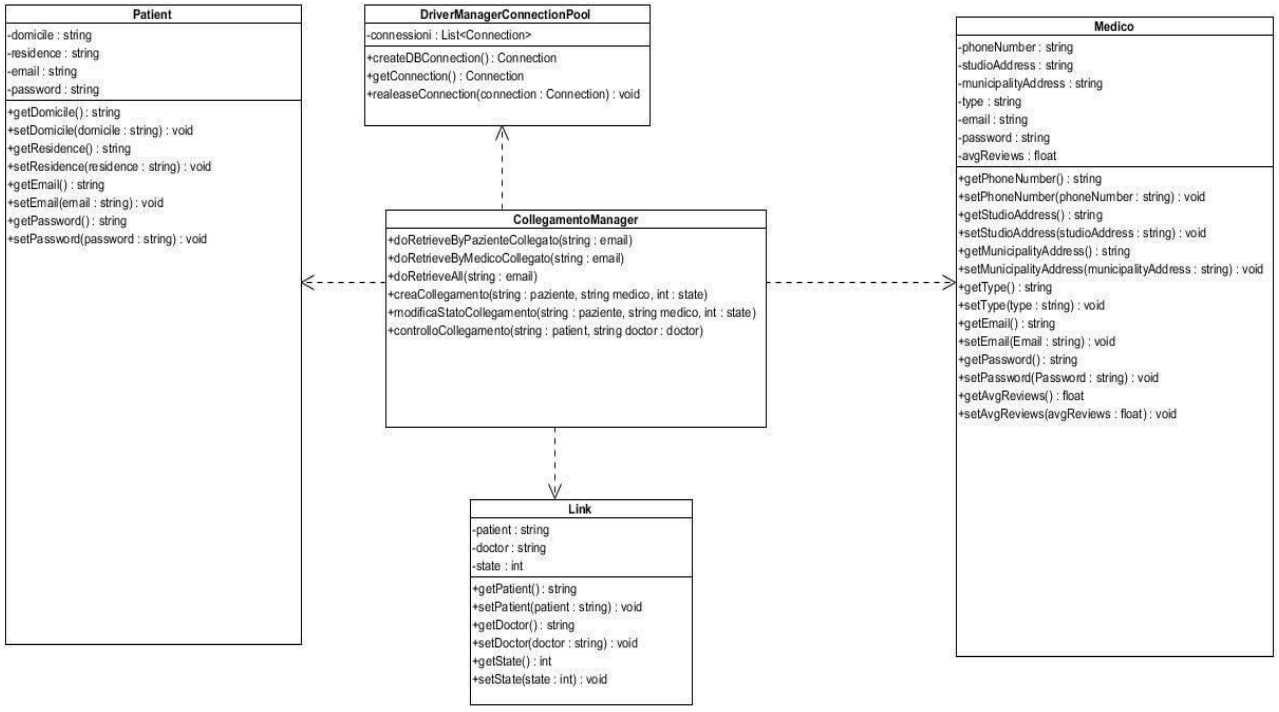
Nome Classe	*ProfiloManager*
Pre-condizione	**context** ProfiloManager::autenticazione (email: String, password: String) : Account **pre** email!=null && password != null; **context** ProfiloManager::registrazione(Account account, Paziente paziente) : boolean **pre** account!=null && paziente!=null; **context** ProfiloManager::registrazione(Account account, Medico medico) : boolean **pre** account!=null && medico!=null; **context** ProfiloManager:: modificaProfilo(Account account, Paziente paziente) : boolean **pre** account!= null && paziente!=null; **context** ProfiloManager:: modificaProfilo(Account account, Medico medico) : boolean **pre** account!= null && medico!=null; **context** AccountManager:: visualizzaPaziente(email: String) : Paziente **pre** email!=null; **context** AccountManager:: visualizzaMedico(email: String) : Medico **pre** email!=null; **context** AccountManager:: cercaAccount(email: String) : boolean **pre** email!=null;
Post-condizione	**context** ProfiloManager:: modificaProfilo(Account account, Paziente paziente) : boolean **post** account.name=newName && account.surname= newSurname && account.BirthDate=newBirthDate && account.CF= newCF && patient.password=newPassword && patient.Domicile= newDomicile && patient.Residence= newResidence **context** ProfiloManager:: modificaProfilo(Account account, Medico Medico) : boolean **post** account.name=newName && account.surname= newSurname && account.BirthDate=newBirthDate && account.CF= newCF && doctor.PhoneNumber=newPhoneNumber && doctor.StudioAddress=newStudioAddress && doctor.MunicipalityAddress=newMunicipalityAddress
Invarianti	NA

3.3 APPUNTAMENTOMANAGER



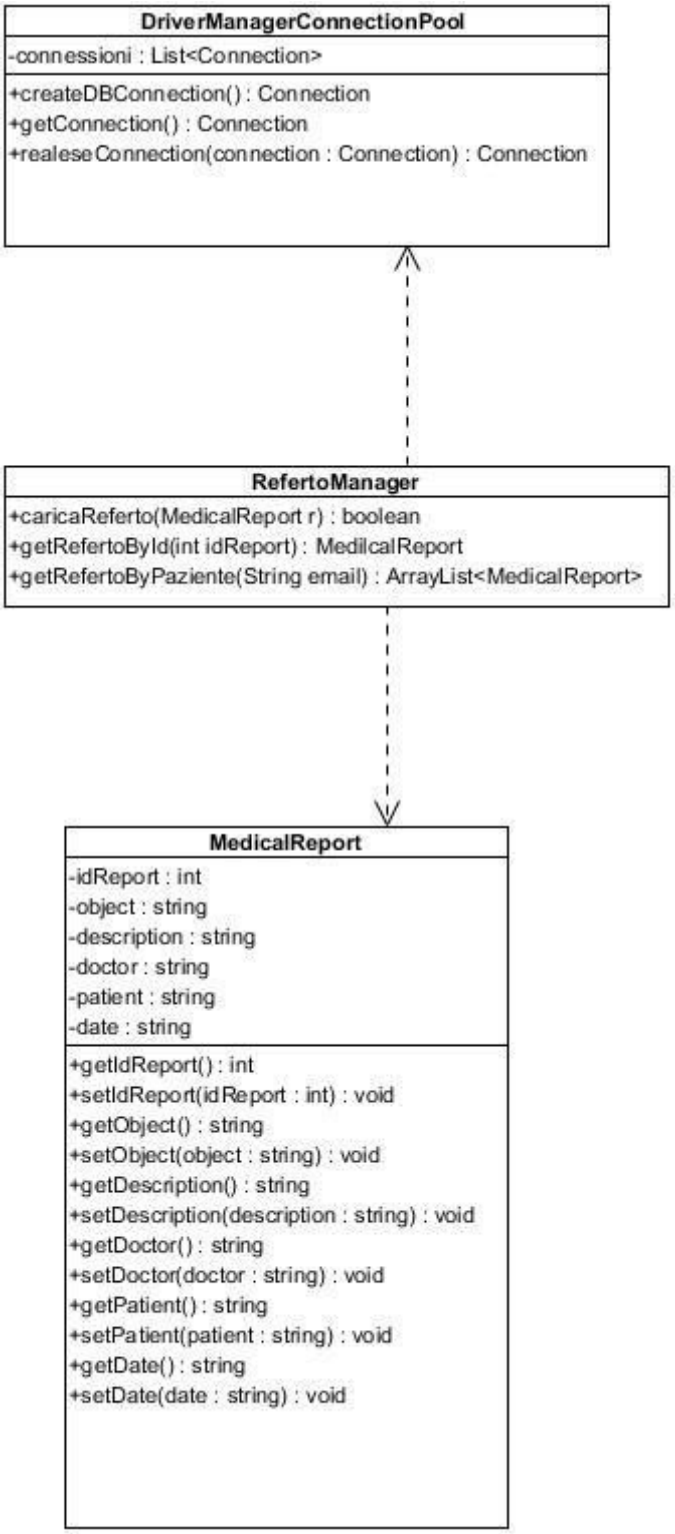
Nome Classe	*AppuntamentoManager*
Pre-condizione	**context** AppuntamentoManager::visualizzaAppmedico (email: String) : Collection<Appuntamento> **pre** email!=null **context** AppuntamentoManager::visualizzaApppaziente (email: String) : Collection<Appuntamento> **pre** email!=null **context** AppuntamentoManager::visualizzaAppuntamento (id: int) : Appuntamento **pre** id!=null **context** AppuntamentoManager::controlloAppDisponibile (email: String, date: String, time: String) : boolean **pre** email!=null && date!=null && time!=null **context** AppuntamentoManager::inserisciApp (Appuntamento appuntamento) : boolean **pre** appuntamento!=null **context** AppuntamentoManager::ritornoID (emailmed: String, emailpaz: String, date: String, state:int) : int **pre** emailmed!=null && emailpaz!=null && date!=null && state!=null **context** AppuntamentoManager::updateAppuntamentoState (state: int, id: int) : boolean **pre** state!=null && id!=null **context** AppuntamentoManager::modificaAppuntamento (state: int, id: int; date: String, time:String) : boolean **pre** state!=null && id!=null && date!=null && time!=null
Post-condizione	**context** AppuntamentoManager::updateAppuntamentoState (newState: int, newId: int) : boolean **post** appointment.state=newState **context** AppuntamentoManager::modificaAppuntamento (newState: int, newId: int; date: String, newTime:String) : boolean **post** appointment.state=newState && appointment.id=newId && appointment.date=newDate && appointment.time=newTime
Invarianti	NA

3.4 COLLEGAMENTOMANAGER



Nome Classe	*CollegamentoManager*
Pre-condizione	**context** CollegamentoManager::doRetrieveAll (email:String) : Collection<Medico> **pre** email!=null **context** CollegamentoManager::creacollegamento (emailpaz: String, emailmed: String, state: int) : boolean **pre** emailpaz!=null && emailmed!=null && state!=null **context** CollegamentoManager::modificaStatoCollegamento (emailpaz: String, emailmed: String, state: int) : boolean **pre** emailpaz!=null && emailmed!=null && state!=null **context** CollegamentoManager::controlloCollegamento (emailpaz: String, emailmed: String) : boolean **pre** emailpaz!=null && emailmed!=null
Post-condizione	**context** CollegamentoManager::modificaStatoCollegamento (emailpaz: String, emailmed: String, nuovoState: int) : boolean **post**: link.state= nuovoState
Invarianti	NA

3.5 REFERTOMANAGER

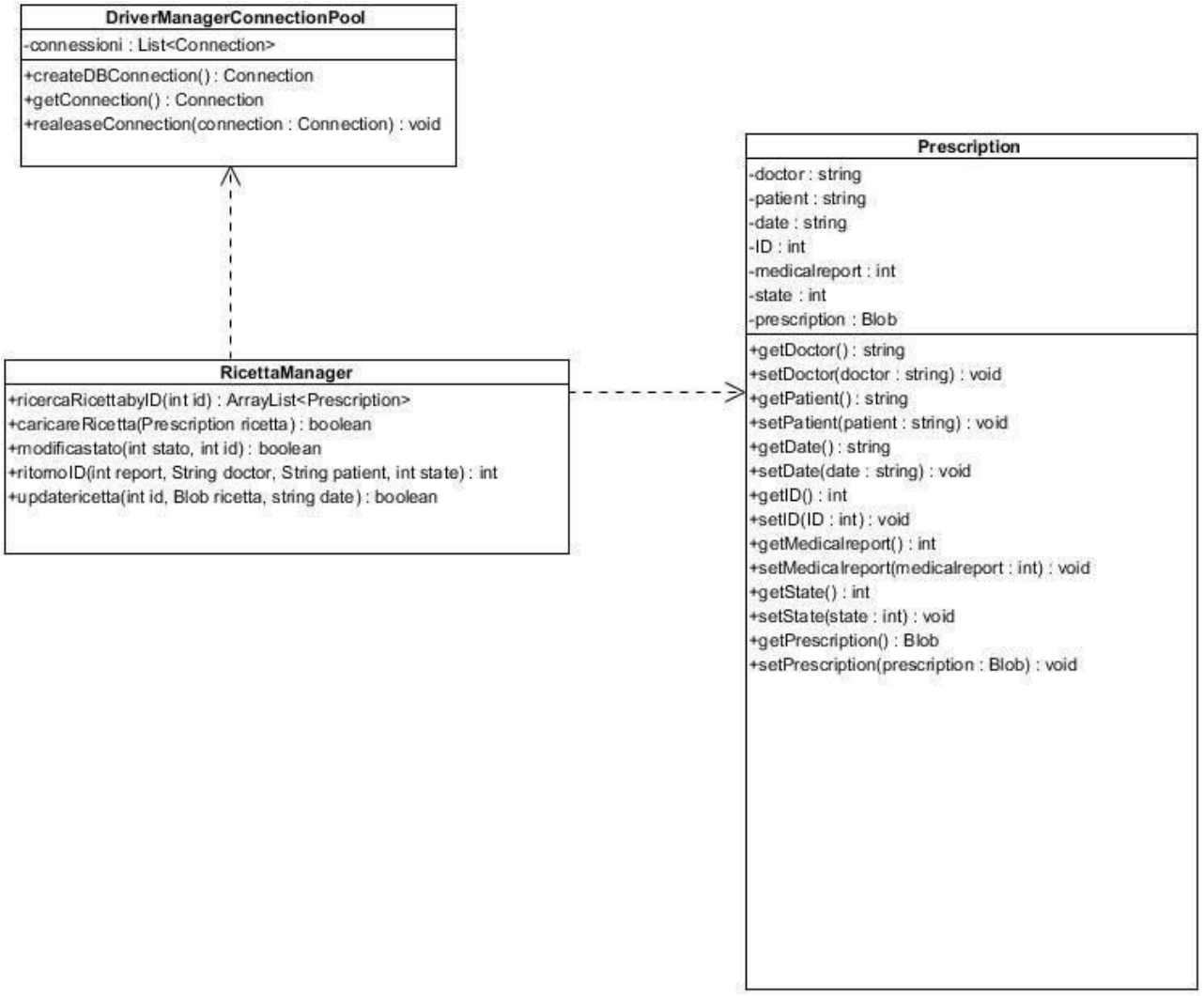


Nome Classe	*RefertoManager*
Pre-condizione	**context** RefertoManager::caricaReferto(referto: Referto) : boolean **pre** referto!=null; **context** RefertoManager::getRefertoById(idReferto: int) : Referto **pre** idReferto!=null; **context** RefertoManager::getRefertoByPaziente(email: String) : Collection<referto> **pre** email!=null;
Post-condizione	NA
Invarianti	NA

3.6 RICERCAMANAGER

Nome Classe	*RicercaManager*
Pre-condizione	**context** RicercaManager::cercaMedicoNome (nome: String, cognome: String, email:String) : Collection<Medico> **pre** nome!=null && cognome!=null && email!=null; **context** RicercaManager::cercaPazienteNome (nome: String, cognome: String, email:String) : Collection<Paziente> **pre** nome!=null && cognome!=null && email!=null; **context** RicercaManager:getMedici (email:String) : Collection<Medico> **pre** email!=null **context** RicercaManager:getPazienti (email:String) : Collection<Paziente> **pre** email!=null **context** RicercaManager:cercaAccountMedico (email:String) : Account **pre** email!=null **context** RicercaManager:cercaAccountPaziente (email:String) : Account **pre** email!=null **context** RicercaManager:cercaMedico (email:String) : Medico **pre** email!=null **context** RicercaManager:cercaPaziente (email:String) : Paziente **pre** email!=null
Post-condizione	NA
Invarianti	NA

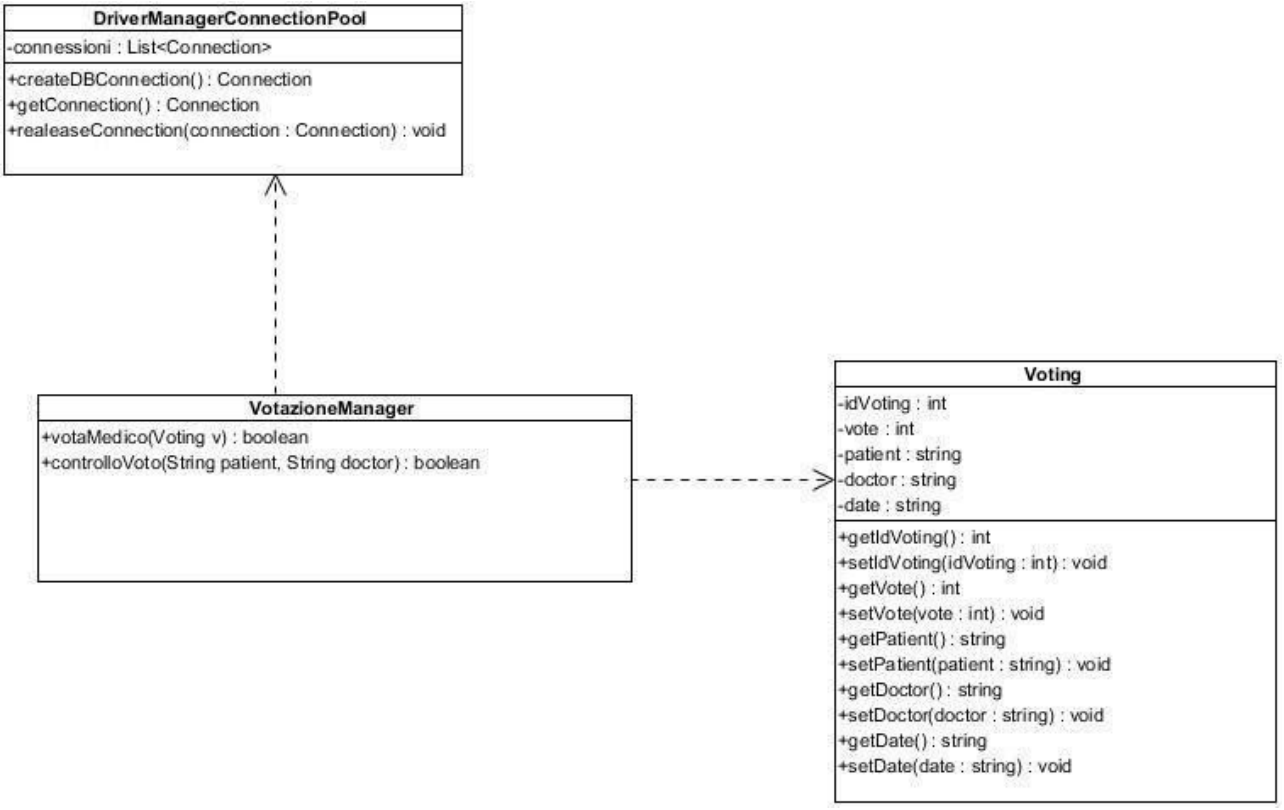
3.7 RICETTAMANAGER



Nome Classe	*GestioneRicetta*
Pre-condizione	**context** GestioneRicetta::richiedereRicetta (Prescription ricetta) : boolean **pre ricetta!=null** **context** GestioneRicetta::scaricareRicetta (id: int) : Ricetta **pre** id!=null **context** GestioneRicetta::caricaRicetta (Prescription ricetta) : boolean **pre** ricetta!=null **context** GestioneRicetta::ritornoID (report: int, doctor: String, patient: String, state: int) : int **pre** report!=null && doctor!=null && patient!=null and state!=null

	context GestioneRicetta::modificastato(stato: int, id: int) boolean **pre**: stato!=null && id!=null **context** GestioneRicetta::updatericetta(stato: int, ricetta: Blob, date: String) boolean **pre**: stato!=null && ricetta!=null && date!=null
Post-condizione	**context** GestioneRicetta::modificastato(stato: int, id: int) boolean **post**: ricetta.state=stato **context** GestioneRicetta::updatericetta(stato: int, ricetta: Blob, newDate: String) boolean **post**: ricetta.prescription=ricetta && ricetta.date= newDate
Invarianti	NA

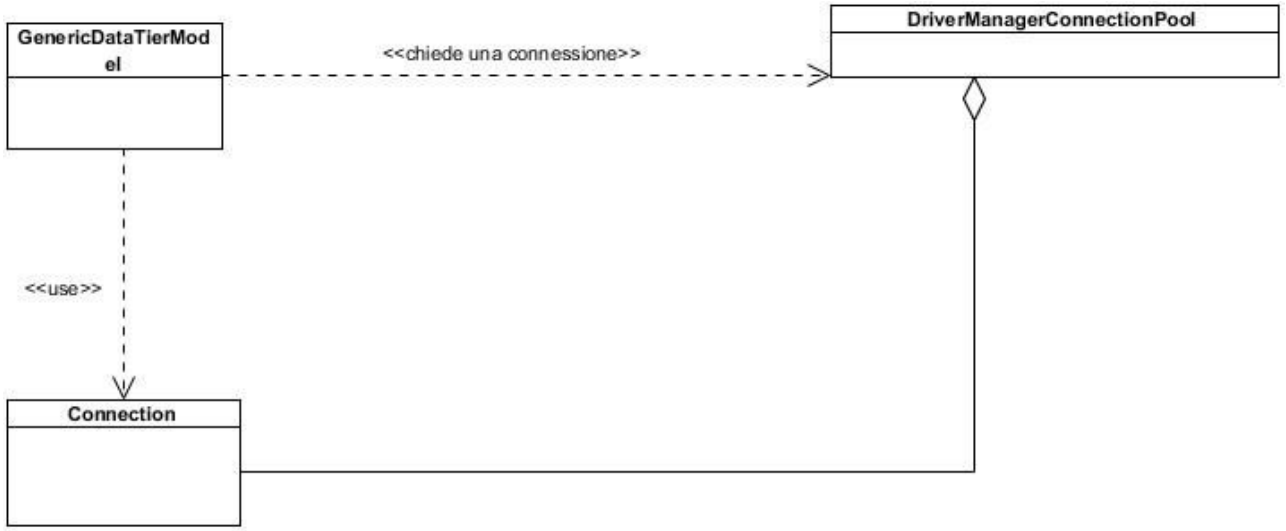
3.8 VOTAZIONEMANAGER



Nome Classe	*VotazioneManager*
Pre-condizione	**context** VotazioneManager::votaMedico (v: Voting) : boolean **pre** v!=null **context** VotazioneManager::controlloVoto (patient: String, doctor: String) : boolean **pre** patient!=null && doctor!=null
Post-condizione	NA
Invarianti	NA

4. 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.
Object pool pattern



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.