MedQueue

ODD
Object Design Document

MedQueue

Afeltra Angelo
Amato Adriano
Fucile Andrea
Rapa Giovanni

Sommario

1. **Introduzione**
   - 1.1 Object design trade-offs
     - 1.1.1 Componenti off-the-shelf
     - 1.1.2 Design pattern
   - 1.2 Linee guida per la documentazione dell'interfaccia
     - 1.2.1 Classi e Interfacce Java
     - 1.2.2 Oggetti Vue
     - 1.2.3 Database SQL
   - 1.3 Definizioni, acronimi e abbreviazioni
   - 1.4 Riferimenti
2. **Packages**
   - 2.1 Divisione in pacchetti
   - **2.2 MedQueue**
     - 2.2.1 Package Bean
     - 2.2.2 Package Business
     - 2.2.3 Package Persistence
     - 2.2.4 Package Presentazione
   - **2.3 MedQueueAppDesktop**
     - 2.3.1 Package Persistence
     - 2.3.2 Package Business
     - 2.3.3 Package Presentazione
3. **Interfacce delle classi**
   - 3.1 PresentazoneDaoInterface
   - 3.2 StrutturaDaoInterface
   - 3.3 OperazioneDaoInterface
   - 3.4 AmbulatorioDaoInterface
   - 3.5 UtenteDaoInterface
   - 3.6 DaoInterface
   - 3.7 AccessoInterface
   - 3.8 GestioneInterface
4. **Class diagram**
   - 4.1 MedQueueAppDesktop
   - 4.2 MedQueue
5. **Glossario**

1. INTRODUZIONE

Dopo la realizzazione dei documenti RAD e SDD abbiamo descritto in linea massima quello che sarà il nostro sistema e quindi i nostri obiettivi, tralasciando gli aspetti d'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, si vanno a descrivere i trade-offs che dovranno essere rispettati dagli sviluppatori, i design pattern stabiliti, le linee guida sulla documentazione delle interfacce, la divisione in pacchetti e le interfacce delle classi da sviluppare.

1.1 Object design trade-offs

Leggibilità vs Tempo
Il codice deve essere il più comprensibile possibile per poter facilitare la fase di testing ed eventuali future modifiche.
A tale scopo andranno inseriti commenti nel codice che ne specificano la comprensione.
Questa caratteristica aumenta il tempo di sviluppo ma allo stesso tempo lo rende più comprensibile
Modificabilità vs Costi
La necessità di sviluppare un'applicazione web apre alla possibilità di utilizzare una vastissima collezione di framework e librerie utili per la realizzazione del prodotto.
Data la facilita di adattamento del team di sviluppo all'utilizzo di nuove componenti abbiamo adottato l'utilizzo di componenti off-the-shelf per il core dell'applicazione.
Questa caratteristica aumenta la modificabilità del prodotto e allo stesso tempo utilizzando framework free non inciderà sui cosi.
Memoria vs Estensibilità
Il sistema deve permettere l'estensibilità a discapito della memoria, in modo tale da dare la possibilità al cliente di richiedere funzionalità aggiuntive.

1.1.1 Componenti off-the-shelf

Per motivazioni legate al budget ridotto, si ricorrerà all'utilizzo di un framework per la realizzazione dell'interfaccia utente: Ionic.

Ionic Framework si pone come supporto per la creazione d'interfacce grafiche per applicazioni ibride, i loro autori hanno raccolto le migliori "best practice" per lo sviluppo di interfacce mobile con tecnologie Web e le hanno codificate in questo framework, evitando quindi che ogni sviluppatore riparta da zero nello sviluppo dell'interfaccia di una nuova applicazione.
Questa soluzione si presta alla realizzazione di un'interfaccia completa e minimale, ideale per il servizio che forniremo, offrendo allo stesso tempo la possibilità, qualora si voglia, di convertire la web app in un app Android o iOS senza dover modificare il codice di quest'ultima.
Il back-end farà invece forte affidamento sul framework Spring Boot, nota soluzione nell'ambito delle applicazioni distribuite Java. Composto da un core ben ottimizzato e prodotto con l'obiettivo di ridurre quello che il codice "boilerplate", consente agli sviluppatori di concentrarsi maggiormente sulla logica di business dell'applicazione, piuttosto che sulla comunicazione tra le varie componenti.
Per quanto riguarda l'applicazione desktop che sarà consegnata alla struttura visto il facile sviluppo non si userà nessuna componente off-the-shelf.

1.1.2 Design patterns

Per velocizzare lo sviluppo della web app, e anticipare i possibili cambiamenti del sistema si utilizza:

1.1.2.1 Design Pattern Facade

Il Design Pattern Facade fornisce un'interfaccia per accedere ad un'insieme di oggetti che compongono un sottosistema.

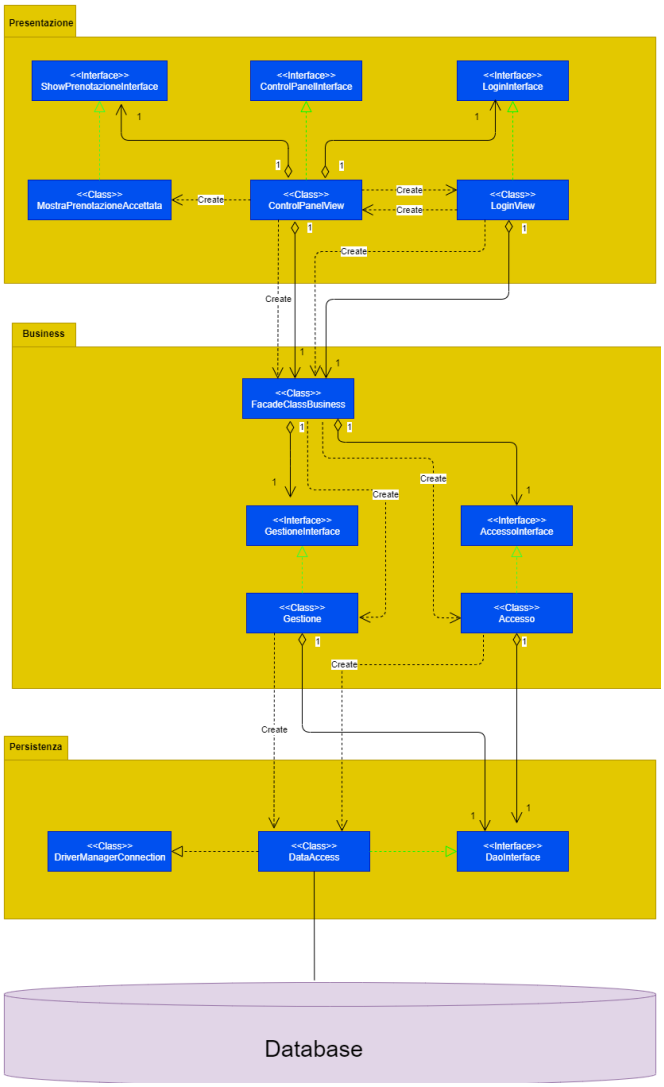
Dovrebbe essere utilizzato da tutti i sottosistemi in sistema software, definisce tutti i servizi del sottosistema e fornisce un'architettura chiusa.

Nel nostro sistema il pattern facade viene utilizzato per accedere ai sottosistemi che compongono il layer di business, in modo tale da avere un architettura chiusa.

1.1.2.2 Design Pattern DAO

Il DAO (Data Access Object) è un pattern architetturale per la gestione della persistenza: si tratta fondamentalmente di una classe con relativi metodi che rappresenta un'entità tabellare di un RDBMS, usata principalmente in applicazioni web per stratificare e isolare l'accesso ad una tabella tramite query.

I metodi del DAO contenenti le rispettive query verranno così richiamati dalle classi della business logic.



1.2 Linee guida per la documentazione dell'interfaccia

E richiesto agli sviluppatori di seguire le seguenti linee guida al fine di essere consistenti nell'intero progetto e facilitare la comprensione delle funzionalità di ogni componente.

1.2.1 Classi e Interfacce Java

Nella scrittura di codice per le classi Java ci si atterra allo standard Google Java.
Tale standard prevede delle regole da rispettar.
Quando si codificano classi e interfacce Java, si dovrebbero rispettare le seguenti regole di formattazione:

1) Non inserire spazi tra il nome del metodo e la parentesi tonda “(” che apre la lista dei parametri.
2) La parentesi graffa aperta “{“si trova alla fine della stessa linea dell’istruzione di dichiarazione.
3) La parentesi graffa chiusa “}” inizia su una nuova riga vuota allo stesso livello di indentazione del nome della classe o dell’interfaccia.

Nel caso di istruzioni semplici, ogni linea deve contenere al massimo una sola istruzione. Mentre nel caso di istruzioni composte vanno rispettate le seguenti regole:

1) Le istruzioni racchiuse all’interno di un blocco (esempio: for), devono essere indentate di un’unità all’interno dell’istruzione composta.
2) La parentesi di apertura del blocco deve trovarsi alla fine della riga dell’istruzione composta.
3) La parentesi di chiusura del blocco deve trovarsi allo stesso livello di indentazione dell’istruzione composta
4) Le istruzioni composte formate da un’unica istruzione devono essere racchiuse da parentesi.

I nomi di classe devono essere sostantivi, con lettere minuscole e, sia la prima lettera del nome della classe sia la prima lettera di ogni parola interna, deve essere maiuscola. I nomi delle classi dovrebbero essere semplici, descrittivi e che rispettino il dominio applicativo. I nomi dei metodi iniziano con una lettera minuscola (non sono consentiti caratteri speciali) e seguono la notazione a cammello. Dovranno essere semplici, descrittivi e che rispettino il dominio applicativo.

1.2.2 Oggetti Vue

Per gli oggetti Vue ci atteremo al checkstyle predefinito di Vue.js reperibile: CheckStyleVue

1.2.3 Database SQL

I nomi delle tabelle devono seguire le seguenti regole:

1) Devono essere costituiti da sole lettere maiuscole;
2) Il nome deve essere un sostantivo singolare tratto dal dominio del problema ed esplicativo del contenuto

I nomi dei campi devono seguire le seguenti regole:

1) Devono essere costituiti da sole lettere maiuscole;
2) Se il nome è costituito da più parole, non è previsto l’usi di underscore;

1.3 Definizione, acronimi e abbreviazioni

CheckStyle: strumento di analisi statica del codice utilizzato nello sviluppo software per verificare se il codice sorgente Java è conforme alle regole di codifica specificate.

1.4 Riferimenti

- Design goals: sezione 1.2 dell'SDD
- Off-The-Shelf: servizi esterni al sistema di cui viene fatto l'utilizzo
- Spring: Spring
- Ionic: Ionic

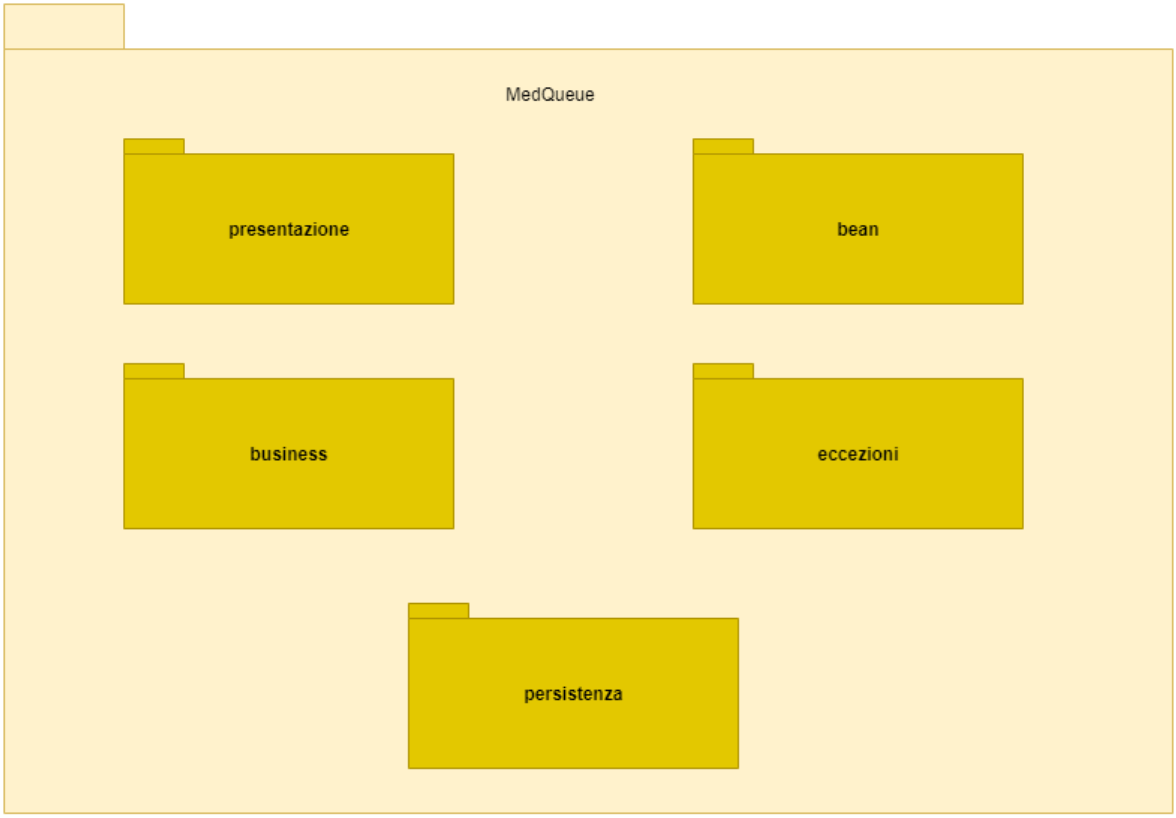
2. Packages

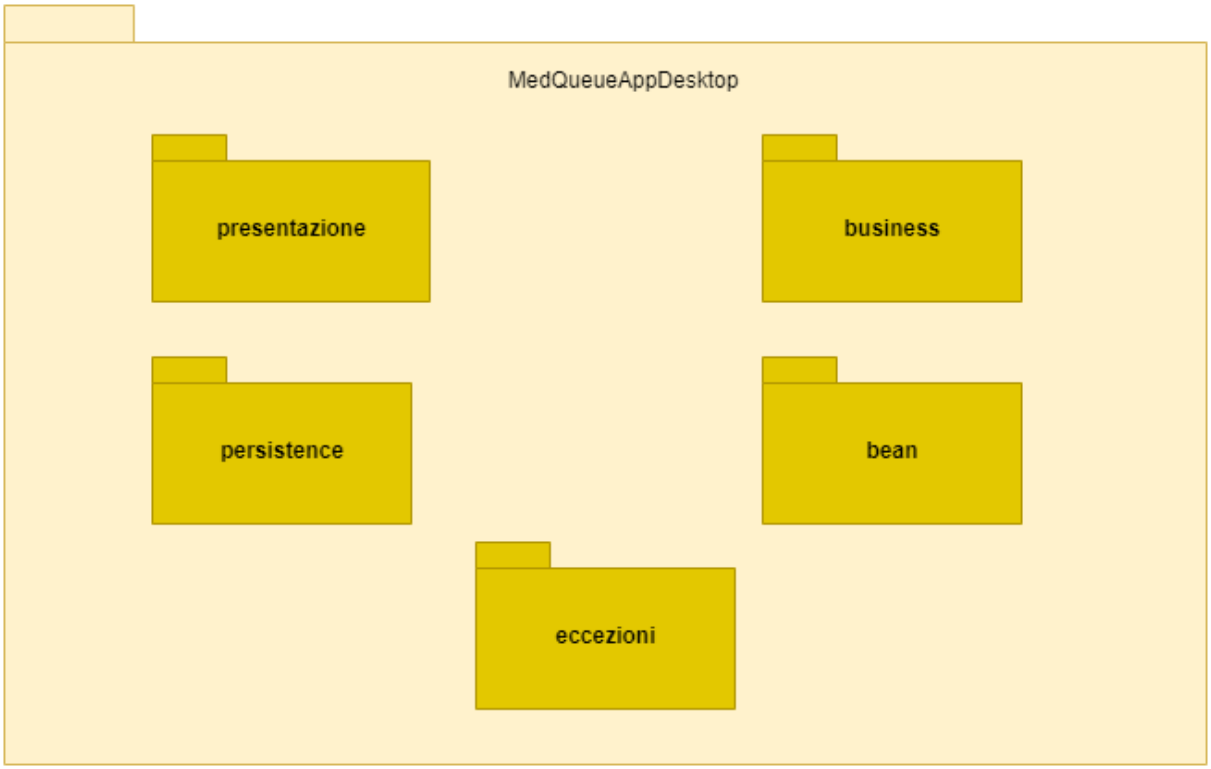
In questa sezione presentiamo in modo più approfondito quella che è la divisione in sottosistemi e l'organizzazione del codice in file

2.1 Divisione in pacchetti

Il sistema sia (web app che applicazione desktop) rispecchia l'architettura definita nel documento SDD, quindi avremo un sistema suddiviso in tre livelli (three-tier):

Presentazione 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
Business Layer	Ha il compito di elaborare i dati da inviare al client, e spesso grazie a delle richieste fatte al database, tramite il Persistence Layer, accede ai dati persistenti. Si occupa di varie gestioni quali: • Accesso • Prenotazione • Visualizzazione Coda • Gestione
Persistence Layer	Ha il compito di memorizzare i dati sensibili del sistema utilizzando un DBMS, inoltre riceve le varie richieste dal Business Layer inoltrandole al DBMS e restituendo i dati richiesti





ODD-Object Design Document

2.2 MedQueue

2.2.1 Package Bean

Una versione a dimensione reale e disponibile : “NC12\Documenti\img\Package WebApp\Beans”

Classe:	Descrizione
AmbulatorioBean	Questa classe rappresenta un ambulatorio della struttura ospedaliera
PrenotazioneBean	Questa classe rappresenta la prenotazione dell’utente
StrutturaBean	Questa classe rappresenta la struttura ospedaliera
OperazioneBean	Questa classe rappresenta le operazioni per cui si puo prendere una prenotazione
UtenteBean	Questa classe rappresenta l’utente registrato

2.2.2 Package Business

Una versione a dimensione reale e disponibile : “NC12\Documenti\img\Package WebApp”

Classe:	Descrizione
PrenotazioneController	Questa classe implementa la logica di controllo per tutte le operazioni che riguardano le prenotazioni
AmbulatorioController	Questa classe implementa la logica di controllo per tutte le operazioni che riguardano gli ambulatori
OperazioneController	Questa classe implementa la logica di controllo per tutte le operazioni per cui si può prendere una prenotazione
StrutturaController	Questa classe implementa la logica di controllo per tutte le operazioni che riguardano una struttura
UtenteController	Questa classe implementa la logica di controllo per tutte le operazioni che riguardano un’utente
ViewCoda	Questa classe implementa la logica di controllo per tutte le operazioni che riguardano la visualizzazione delle code
LogInController	Questa classe implementa la logica di controllo per tutte le operazioni d’accesso

2.2.3 Package Persistence

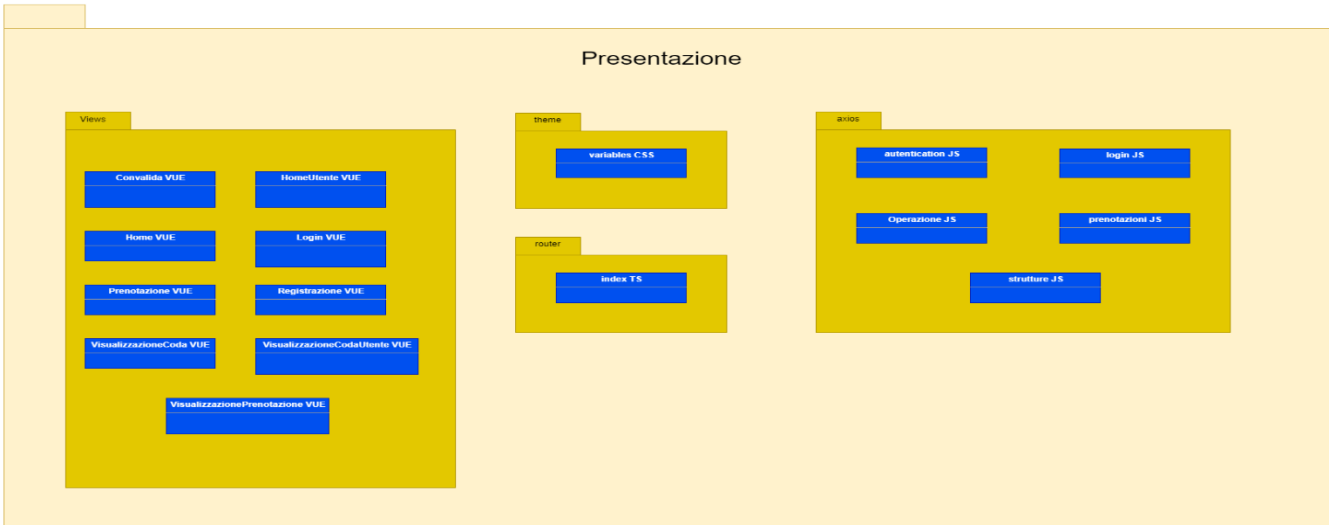
Una versione a dimensione reale e disponibile : “NC12\Documenti\img\Package WebApp\Persistence”

Classe:	Descrizione
PrenotazioneDaoInterface	Questa classe contiene i metodi che permettono di prelevare dal database una prenotazione in base all’id, prelevare le prenotazioni dal database in base ad un ordine, aggiungere, aggiornare una prenotazione, ottenere la prenotazione di un utente e ottenere la coda di prenotazioni di una struttura
StrutturaDaoInterface	Questa classe contiene i metodi che permettono di prelevare dal database una struttura in base all’id, ottenere tutte le strutture in un determinato ordine, aggiungere, aggiornare e cancellare una struttura
OperazioneDaoInterface	Questa classe contiene i metodi che permettono di ottenere una prenotazione in base all’id, ottenere tutte le operazioni in un determinato ordine, aggiungere, aggiornare e cancellare un operazione
AmbulatoriDaoInterface	Questa classe contiene i metodi che permettono di ottenere un ambulatorio in base all’id, ottenere tutti gli ambulatori in un determinato ordine, aggiungere, aggiornare e cancellare un impiegato

UtenteDaoInterface	Questa classe contiene i metodi che permettono di ottenere un utente in base al codice fiscale, ottenere tutti gli utenti in un determinato ordine, aggiungere, aggiornare e cancellare un utente

2.2.4 Package Presentazione

Una versione a dimensione reale e disponibile : “NC12\Documenti\img\Package WebApp\Presentazione”



Il package Presentazione è formato a sua volta da quattro packages: Views, Theme, Router e Axios.
Il package Views implementa interfacce grafiche dell’ applicazione web.
Il package theme contiene il css nativo di ionic.
Il package router contiene tutte le rotte per accedere alle pagine.
Il package axios contiene le chiamate alle funzioni del back end.

2.3 MedQueueAppDesktop

2.3.1 Package Persistence

Una versione a dimensione reale e disponibile : “NC12\Documenti\img\Package AppDesktop\Persistence”

Classe:	Descrizione
DataAccessInterface	Questa classe contiene i metodi che permettono di prelevare dal database prenotazione, struttura, operazione, impiegato in base alla loro chiave, eliminare una prenotazione, ottenere la coda di prenotazioni e servire una prenotazione
DriverManagerConnectionPool	Questa classe permette di gestire i driver per la connessione al Database

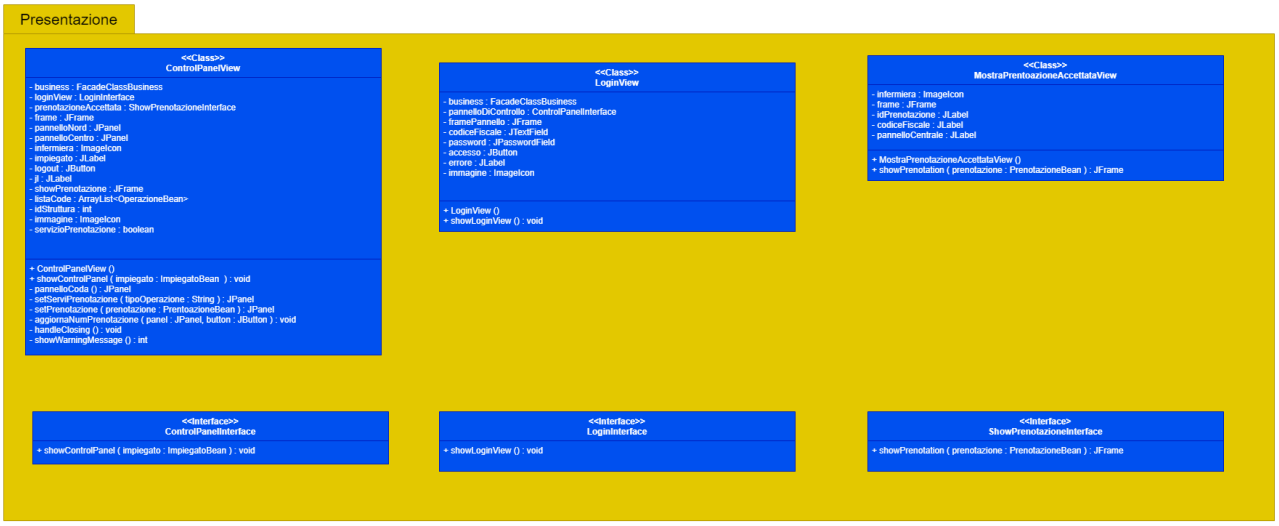
2.3.2 Package Business

Una versione a dimensione reale e disponibile : “NC12\Documenti\img\Package AppDesktop\Business”

Classe:	Descrizione
GestioneInterface	Questa classe contiene i metodi che permettono di accettare una prenotazione, ottenere le code gestibili, ottenere una determianta coda gestibile e ottenere il numero di prenotazioni da accettare
AccessoInterface	Questa classe contiene i metodi che permettono di verificare le credenziali dell’utente
FacadeClassBusiness	Questa classe contiene tutte le funzionalità di business

2.3.3 Package Presenzatione

Una versione a dimensione reale e disponibile : “NC12\Documenti\img\Package AppDesktop\Presentazione”



ODD-Object Design Document

Classe:	Descrizione
ControlPanelInterface	Questa classe genera un frame dove sarà possibile gestire le operazioni e accettare la prenotazione di una determinata coda
LoginInterface	Questa classe genera un frame che permette all'utente di inserire le proprie credenziali per accedere al pannello di controllo
ShowPrenotazioneInterface	Questa classe genera un frame che mostra la prenotazione accettate dall'impiegato

3 Class Interfaces

È possibile reperire la documentazione relativa all'interfaccia pubblica delle varie classi nei file Javadoc :

- AppDesktop: "NC12\JavaDoc\JavaDocAppDesktop"
- WebApp: "NC12\JavaDoc\JavaDocWebApp"

Tali documenti includono la definizione di precondizioni, postcondizioni e inviarianti per i metodi delle classi coinvolte.

Per una migliore navigazione si consiglia di accedere alla documentazione tramite il file index.html.

3.1 PrenotazioneDaoInterface

PrenotazioneDaoInterface	
doRetrieveByKey	**Context:** PrentoazioneDaoIInterface : doRetrieveByKey (id) **Pre:** (id>0); **Post:** Prenotazione->select(p|p.id=id);
doRetrieveAll	**Context:** PrenotazioneDaoInterface : doRetrieveByAll (order) **Pre:** (order==”id” || order==”data” || order==”ora” || order==”convalida” || order==”codiceFiscale” || order==”idOperazione” || order==”idStruttura”); **Post:** Prenotazione->asSet(Prenotazione)
doSave	**Context:** PrenotazioneDaoInterface : doSave (prenotazione) **Pre:** (prenotazione!=null && prenotazione.id>0 && prenotazione.data!=null && prenotazione.ora!= null && prenotazione.convalida!= null && prenotazione.codiceFiscale!=null &&

	prenotazione.idOperazione>0 && prenotazione.idStruttura>0); **Post:** Prenotazione->include(prenotazione);
doDelete	**Context:** PrenotazioneDaoInterface : doDelete (prenotazione) **Pre:** (prenotazione!=null); **Post:** (!Prenotazione->exists(p|p.id==prenotazione.id);
doUpdate	**Context:** PrenotazoneDaoInterface : doUpdate (prenotazione) **Pre:** (prenotazione!=null && prenotazione.id>0 && prenotazione.data!=null && prenotazione.ora!= null && prenotazione.convalida!= null && prenotazione.codiceFiscale!=null && prenotazione.idOperazione>0 && prenotazione.idStruttura>0); **Post:** Prenotazione->include (prenotazione);
getUtentePrenotazioni	**Context:** PrenotazioneDaoInterface : getUtentePrenotazioni (codicefiscale) **Pre:** (codicefiscale!=null && codicefiscle.lenght==16); **Post:** Prenotazioni->select(p|p.codiceFiscale==codicefiscale);
getCodaStruttura	**Context:** PrenotazioneDaoInterface : getCodaStruttura (idStruttura) **Pre:** (idStruttura>0) **Post:** Prenotazione->select(p|p.idStruttura==idStruttura);
getOrarioPrenotazione	**Context:** PrenotazioneDaoInterface : getOrarioPrenotazione (idStruttura, idOperazione, dataPrenotazione) **Pre:** (idStruttura>0 && idOperazione>0 && dataPrenotazione!=null)

3.2 StrutturaDaoInterface

StrutturaDaoInterface	
doRetrieveByKey	**Context:** StrutturaDaoInterface : doRetrieveByKey (id) **Pre:** (id>0); **Post:** Struttura->select(s|s.id=id);
doRetrieveByName	**Context:** StrutturaDaoInterface : doRetrieveByName (nome)

	Pre: nome!=null **Post:** Struttura->Select(s|s.nome==nome)
doRetrieveAll	**Context:** StrutturaDaoInterface : doRetrieveAll () **Post:** Struttura->asSet(Struttura)
doSave	**Context:** StrutturaDaoInterface : doSave (struttura) **Pre:** (struttura!=null && struttura.id>0 && struttura.nome!=null && struttura.indirizzo!= null && struttura.numeroDiTelefono!=null); **Post:** Struttura->include(struttura);
doDelete	**Context:** StrutturaDaoInterface : doDelete (struttura) **Pre:** (struttura!=null); **Post:** (!Struttura->exists(s|s.id==struttura.id);
doUpdate	**Context:** StrutturaDaoInterface : doUpdate (struttura) **Pre:** (struttura!=null && struttura.id>0 && struttura.nome!=null && struttura.indirizzo!= null && struttura.numeroDiTelefono!=null); **Post:** Struttura->include (struttura);

3.3 OperazioneDaoInterface

OperazioneDaoInterface	
doRetrieveByKey	**Context:** OperazioneDaoInterface : doRetrieveByKey (id) **Pre:** (id>0 &&); **Post:** Operazione->select(o|o.id=id);
doRetrieveAll	**Context:** OperazioneDaoInterface : doRetrieveAll (order) **Pre:** (order==”id” || order==”tipoOperazione” || order==”descrizione”); **Post:** Operazione->asSet(Operazione)
doRetrieveByTipo	**Context:** OperazioneDaoInterface : doRetrieveByTipo (tipo) **Pre:** tipo!=null **Post:** Operazione->Select(o|o.tipoOperazione==tipo)
doSave	**Context:** OperazioneDaoInterface : doSave (operazione) **Pre:** (operazione!=null && operazione.id>0 && operazione.tipoOperazione!=null && operazione.descrizione!= null); **Post:** Operazione->include(operazione);

doDelete	**Context:** OperazioneDaoInterface : doDelete (operazione) **Pre:** (operazione!=null); **Post:** (!Operazione->exists(o|o.id==operazione.id);
doUpdate	**Context:** OperazioneDaoInterface : doUpdate (operazione) **Pre:** (operazione!=null && operazione.id>0 && operazine.tipoOperazione!=null && operazione.descrizione!= null); **Post:** Operazione->include (operazione);

3.4 AmbulatoriDaoInterface

AmbulatoriDaoInterface	
doRetrieveByKey	**Context:** AmbulatoriDaoInterface : doRetrieveByKey (id) **Pre:** (id>0 && Ambulatorio->exists(a|a.id==id)); **Post:** Ambulatorio->select(a|a.id=id);
doRetrieveAll	**Context:** AmbulatoriDaoInterface : doRetrieveAll (order) **Pre:** (order==”id” || order==”nome” || order==”idStruttura”); **Post:** Ambulatorio->asSet(Ambulatorio)
doSave	**Context:** AmbulatoriDaoInterface : doSave (ambulatorio) **Pre:** (ambulatorio!=null && ambulatorio.id>0 && ambulatorio.nome!=null && ambulatorio.idStruttura>0); **Post:** Ambulatorio->include(ambulatorio);
doDelete	**Context:** AmbulatoriDaoInterface : doDelete (ambulatorio) **Pre:** (ambulatorio!=null); **Post:** (!Ambulatorio->exists(a|a.id==ambulatorio.id);
doUpdate	**Context:** AmbulatoriDaoInterface : doUpdate (ambulatorio) **Pre:** (ambulatorio!=null && ambulatorio.id>0 && ambulatorio.nome!=null && ambulatorio.idStruttura>0); **Post:** Ambulatorio->include (ambulatorio);

3.5 UtenteDaoInterface

UtenteDaoInterface	
doRetrieveByKey	**Context:** UtenteDaoInterface : doRetrieveByKey (codicefiscale) **Pre:** (codicefiscale!=null && codicefiscale.lenght==16); **Post:** Utente->select(u|u.codicefiscale==codicefiscale);
doRetrieveAll	**Context:** UtenteDaoInterface : doRetrieveAll (order) **Pre:** (order==”codicefiscale” || order==”password” || order==”nome”) || order==”cognome” || order==”dataDiNascita” || order==”indirizzoEmail” || order==”numeroDiTelefono”); **Post:** Utente->asSet(Utente)
doSave	**Context:** UtenteDaoInterface : doSave (utente) **Pre:** utente!=null && utente.codiceFiscale!=null && utente.password!=null && utente.nome!=null && utente.cognome!=null && utente.dataDiNascita!=null && utente.indirizzoEmail!=null && utente.numeroDiTelefono!=null); **Post:** Utente->include(utente);
doDelete	**Context:** UtenteDaoInterface : doDelete (utente) **Pre:** (utente!=null); **Post:** (!Utente->exists(u|u.codiceFiscale==utente.codicefiscale);
doUpdate	**Context:** UtenteDaoInterface : doUpdate (utente) **Pre: (**utente!=null && utente.codiceFiscale!=null && utente.password!=null && utente.nome!=null && utente.cognome!=null && utente.dataDiNascita!=null && utente.indirizzoEmail!=null && utente.numeroDiTelefono!=null); **Post:** Utente->include (utente);

3.6 DaoInterface

DaoInterface	
getPrenotazione	**Context:** DaoInterface : getPrenotazione (id) **Pre:** (id>0); **Post:** Prenotazione->select(p|p.id==id);
getImpiegato	**Context:**DaoInterface : getImpiegato (codicefiscale) **Pre:** (codicefiscale!=null && codicefiscale.lenght==16); **Post:** Impiegato-> select(i|i.codicefiscale==codicefiscale);
getStruttura	**Context:** DaoInterface : getStruttura (id) **Pre:** (id>0); **Post:** Struttura->select(s|s.id=id);
getOperazione	**Context:** DaoInterface : getOperazione (id) **Pre:** (id>0); **Post:** Operazione->select(o|o.id==id);
getOperazioni	**Context:** DaoInterface : getOperazioni () **Post:** Operazioni->asSet(Operazioni);
deletePrenotazione	**Context:** DaoInterface : deletePrenotazioni (id) **Pre:** (id>0); **Post:** (!Prenotazione->exists(p|p.id==id);
numPrenotazione	**Context:** DaoInterface : numPernotazione (idStruttura, idOperazione) **Pre:** (idStruttura>0 && idOperazione>0); **Post:** (Prenotazione-> exists (p|p.idStruttura==idStruttua && p.idOperazione==idOperazione && p.convalida==true).size());
serviPrenotazione	**Context:** DaoInterface : serviPrenotazione (idStruttura, idOperazione) **Pre:** (idStruttura>0 && idOperazione>0); **Post:** (Prenotazione-> select(p|p.idStruttura==idStruttua && p.idOperazione==idOperazione && p.convalida==true));

3.7 AccessoInterface

AccessoInterface	
verificaCredenziali	**Context:** AccessoInterface : verificaCredenziali (codiceficale, password) **Pre:** (codicefiscale!=null && codicefiscale.lenght==16 && password!=null); **Post:** Impiegato-> select(i|i.codiceficale==codicefiscale && i.password==password);

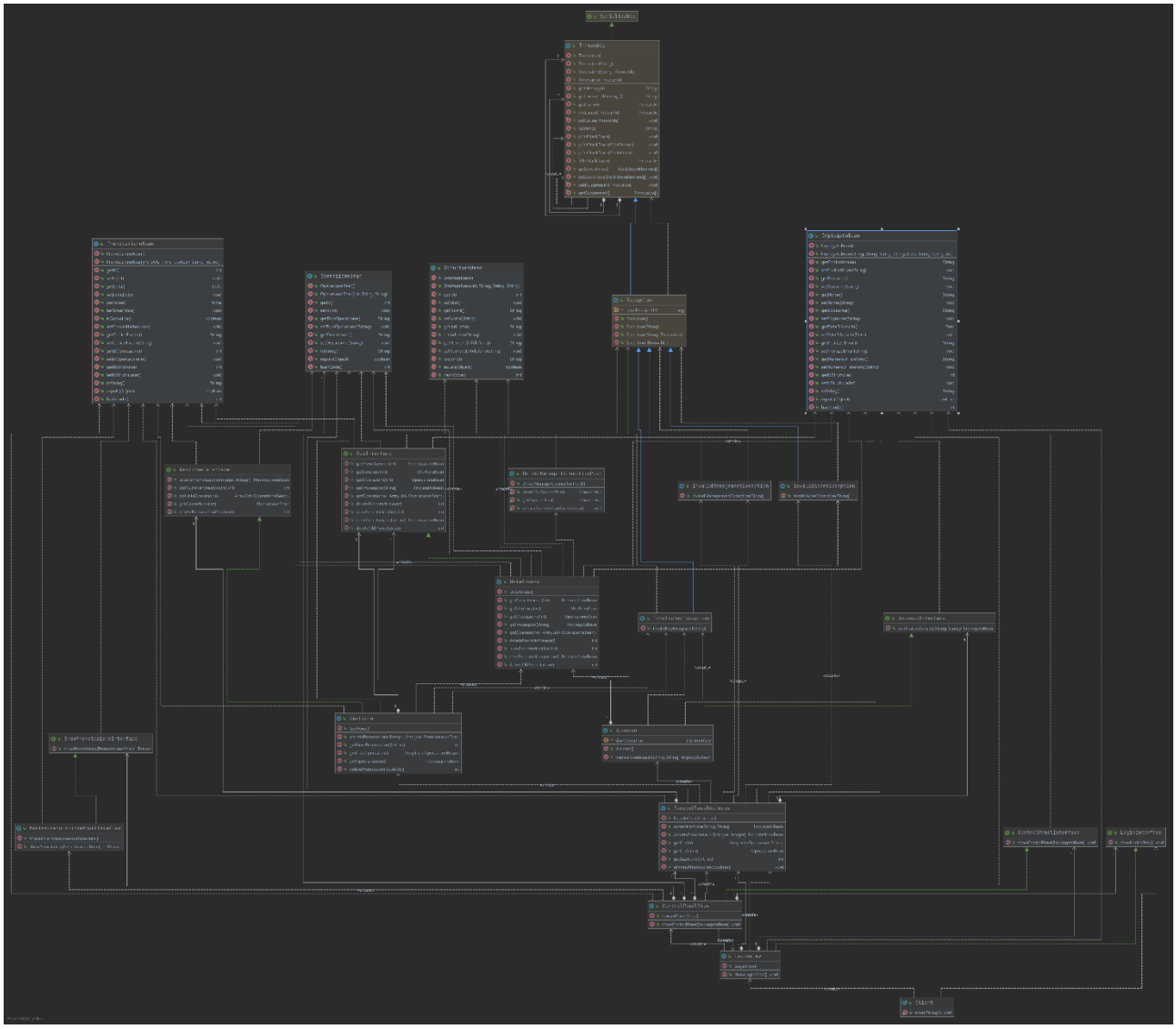
3.8 GestioneInterface

GestioneInterface	
accettaPrenotazione	**Context:** GestioneInterface : accettaPrenotazione (idStruttura, idOperazione) **Pre:** (idStruttura>0 && idOperazione>0); **Post:** (Prenotazione-> select(p|p.idStruttura==idStruttua && p.idOperazione==idOperazione && p.convalida==true));
getListaOperazioni	**Context:** GestioneInterface : getListaOperazioni () **Post:** Operazioni->asSet(Operazioni);
getNumPrenotazione	**Context:** GestioneInterface : getNumPernotazione (idStruttura, idOperazione) **Pre:** (idStruttura>0 && idOperazione>0); **Post:** (Prenotazione-> exists (p|p.idStruttura==idStruttua && p.idOperazione==idOperazione).size());
getOperazione	**Context:** GestioneInterface : getOperazione(id) **Pre:** (id>0) **Post:** Operazione->select(o|o.idOperazione==id)

4 Class Diagram

4.1 MedQueueAppDesktop

Una versione a dimensione reale e disponibile :
“NC12\Documenti\img\AppDesktopClassDiagram”



4.2 MedQueue

Una versione a dimensione reale e disponibile:
“NC12\Documenti\img\MedQueueClassDiagram”