ChemioPlan
GESTIONE INTEGRATA

ODD
OBJECT DESIGN
DOCUMENT

ChemioPlan

Riferimento	
Versione	1.0
Data	08/02/2023
Destinatario	Prof.ssa F. Ferrucci, Prof F.Palomba
Presentato da	Clericuzio Alessandro Contardo Vittorio di Pippa Francesco Pio Lo Conte Christian Matteis Francesco
Approvato da	

Revision History

Data	Versione	Cambiamenti	Autori
30/01/2023	1.0	Introduzione	Gruppo
04/02/2023	1.1	Packages, Design patterns	Gruppo
09/02/2023	1.1	Revisione Finale	Gruppo

Sommario

1. Introduzione

1.1 Object design trade-offs

Durante lo sviluppo dell'Object Design abbiamo riscontrato diversi compromessi di progettazione ed è stato necessario stabilire quali rispettare e quali rendere opzionali. Abbiamo quindi individuato i seguenti trade-off:

Buy vs Build: Abbiamo individuato diversi framework e librerie utili alla realizzazione del prodotto. Abbiamo quindi valutato e scelto di utilizzare alcuni componenti off-the-shelf dati i tempi stringenti per la consegna dell'applicazione, scegliendo con molta attenzione quali tra questi potessero fare al caso nostro. (Scegliamo il buy).

Tempo di risposta vs Safety: Abbiamo preferito dare priorità all'affidabilità rispetto ai tempi di risposta, garantendo un accurato controllo dei dati di input a discapito del tempo del tempo di risposta del sistema. (Scegliamo safety).

Costi vs Estensibilità: Il sistema dovrà permettere l'estensibilità così da dare la possibilità al cliente di richiedere lo sviluppo di nuove funzionalità, a discapito dei costi in termini di ore lavoro. Questa scelta è dettata dalla previsione che il prodotto software in oggetto avrà successive release. (Scegliamo l'estensibilità)

Costi vs Modificabilità: Il sistema dovrà permettere di poter modificare le proprie funzionalità in modo semplice, in modo da poter permettere agli sviluppatori di poter modificare e migliorare le funzionalità del sistema, a discapito dei costi in termini di ore di lavoro. (Scegliamo la Modificabilità)

Comprensibilità vs Tempo: Uno degli obiettivi dell'implementazione sarà quello di scrivere del codice che rispetti lo standard proposto da Google per la programmazione nel linguaggio Java, oltre all'uso di commenti sui metodi. Ciò favorisce anche la comprensibilità, agevolando il processo di mantenimento e di modifica del progetto anche per futuri sviluppatori che non hanno lavorato dall'inizio al progetto stesso. Questo vantaggio, tuttavia, comporta un incremento del tempo per lo sviluppo e la realizzazione dell'intero sistema, che però è ripagato da una maggiore manutenibilità e chiarezza dei contenuti implementativi. (Scegliamo Comprensibilità).

1.1.1 Componenti off-the-shelf

Abbiamo deciso di utilizzare diversi componenti già pronti per evitare di allungare inutilmente i tempi di sviluppo del software.

Per la gestione persistente dei dati abbiamo utilizzato MONGODB come DBMS non relazionale poiché rende l'integrazione dei dati più facile e veloce rispetto a MYSQL.

Per scambiare informazioni ed eseguire azioni tra il database e il software si utilizzerà MongoRepository che è un componente il cui utilizzo ed efficienza sul mercato è ampiamente documentato.

Tutti gli strumenti selezionati sono gratuiti ed open source, scelta che rispetta i requisiti di costo.

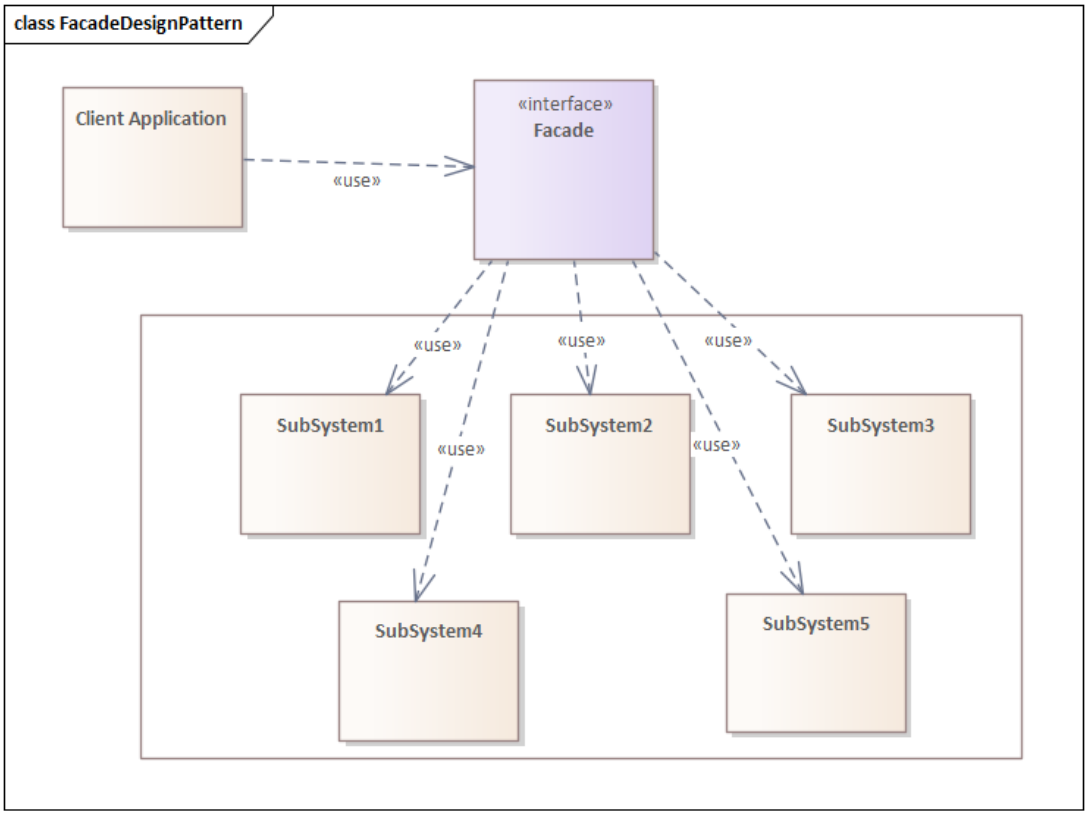
1.1.2 Design Patterns

Abbiamo utilizzati alcuni design patterns per velocizzare lo sviluppo del sistema.

La scelta è ricaduta su un Facade Patterns che permette l'accesso ai sottosistemi che espongono interfacce complesse e molto diverse tra loro attraverso un'unica interfaccia più semplice.

Nel nostro sistema è compito dell'API Gateway fornire un'interfaccia Facade tra client e sottosistemi.

Questo design patterns permette di effettuare le chiamate ai vari microservizi in modo più semplice e intuitivo.



1.2 Linea guida per la documentazione dell'interfaccia

Per facilitare la comprensione delle funzionalità dei componenti, abbiamo deciso di seguire le seguenti linee guida che fanno riferimento allo standard Google Java e abbiamo deciso di utilizzare il tool CheckStyle con configurazione settata su GoogleCheck per avere una comprensibilità maggiore del codice .

Nomenclatura delle componenti:

- Nomi delle classi
  - Ogni classe deve avere nome in CamelCase;
  - Ogni classe deve avere nome singolare;
  - Ogni classe che esegue la logica applicativa deve avere nome composto dal sostantivo che descrive l'azione seguito dal suffisso "Controller".
- Nomi dei metodi
  - Ogni metodo deve avere nome in camelCase(lower);
- Nomi delle variabili

  - Ogni variabile deve avere nome in camelCase(lower);
- Nomi delle eccezioni
  - Ogni eccezione deve avere nome esplicativo del problema segnalato.

Quando si codificano classi e interfacce Java, si devono usare le seguenti regole di formattazione

- Non inserire spazi tra il nome del metodo e la parentesi tonda "(“che apre la lista dei parametri
- La parentesi graffa aperta "{“si trova alla fine della stessa linea dell'istruzione di dichiarazione staccata di uno spazio
- 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 singole ogni linea dovrà contenere un'istruzione. Nel caso in cui un'istruzione è più lunga di 100 caratteri verrà separata andando a capo con uno spazio di indentazione rispetto alla sua riga.

Nel caso di istruzioni più complesse e articolate dovranno essere seguite le seguenti indicazioni

- Le istruzioni racchiuse all'interno di un blocco (esempio: for), devono essere indentate di un'unità all'interno dell'istruzione composta
- La parentesi di apertura del blocco deve trovarsi alla fine della riga dell'istruzione composta staccata di uno spazio
- La parentesi di chiusura del blocco deve trovarsi allo stesso livello di indentazione dell'istruzione composta
- Le istruzioni composte formate da un'unica istruzione devono essere racchiuse da parentesi

2. Packages

In questa sezione verrà presentata quella che è la divisione in sottosistemi e l'organizzazione del codice in files.

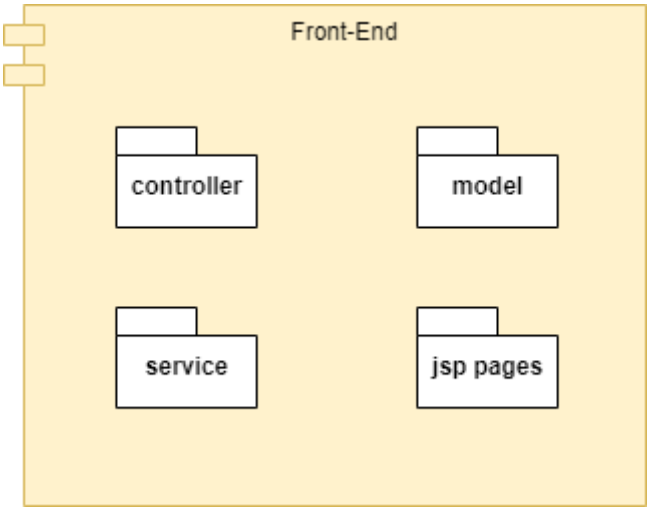
2.1 Divisione in pacchetti

ChemioPlan si basa su un'architettura a microservizi come specificato nel System Design Document.
Il sistema è composto da :

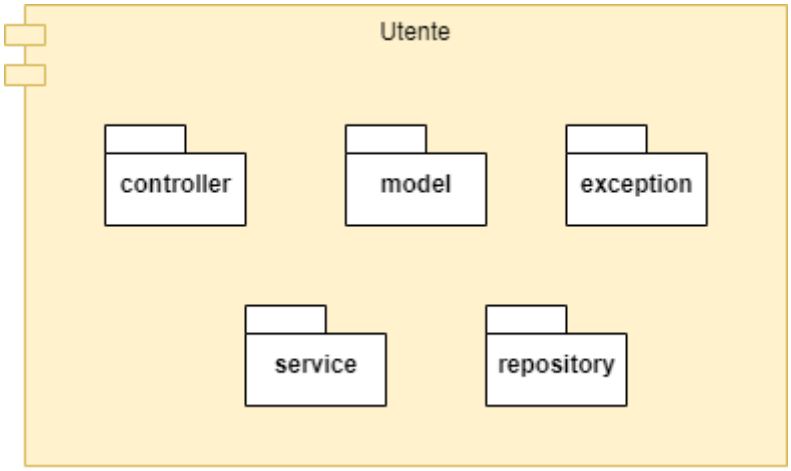
- 6 microservizi (utente, paziente, prenotazione, farmacia, ottimizzazione acquisiti e ottimizzazione scheduling) ognuno di questi avrà un database ad esso collegato.
- Un API Gateway che farà da intermediario tra i microservizi e il layer di presentazione.
- Un layer di presentazione.

Il layer di presentazione è contenuto in un progetto spring denominato come Front-End che ha il compito di gestire la parte grafica del Software.

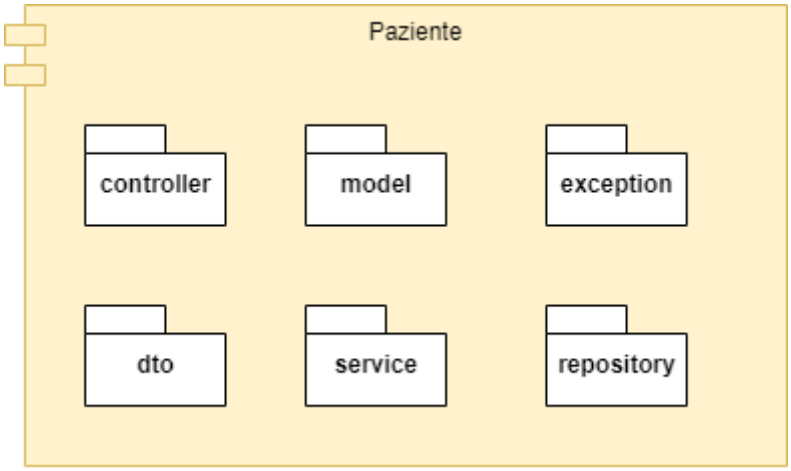
Nome pacchetto	Front-End
Descrizione	Contiene le classi necessarie alla creazione delle interfacce
Dipendenze	N\A



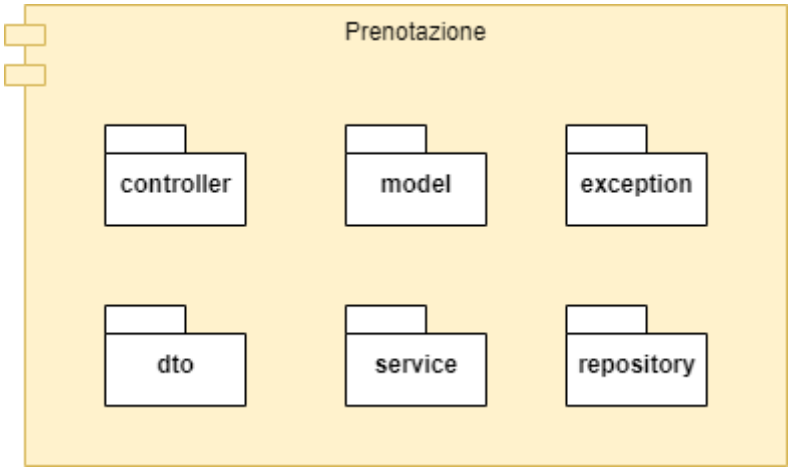
Nome pacchetto	Utente
Descrizione	Contiene le classi necessarie alla gestione dell' utente e gli eventi ad esso collegati
Dipendenze	N\A



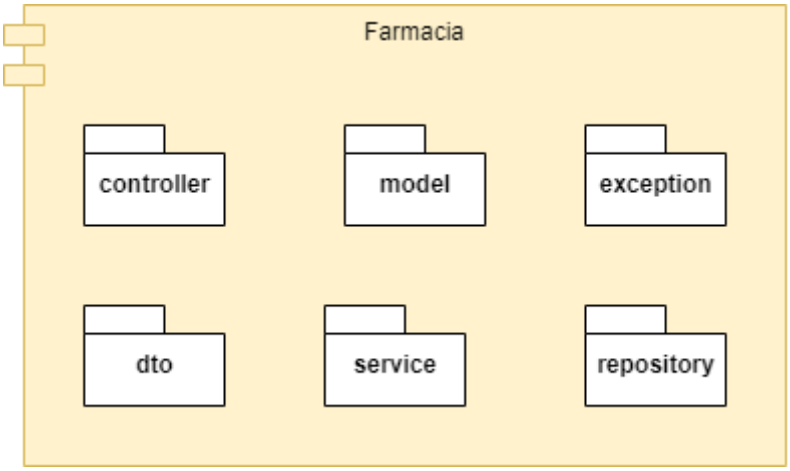
Nome pacchetto	Paziente
Descrizione	Contiene le classi necessarie alla gestione del paziente e gli eventi ad esso collegati
Dipendenze	N\A



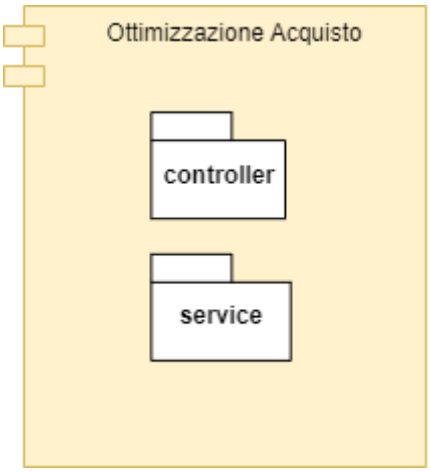
Nome pacchetto	Prenotazione
Descrizione	Contiene le classi necessarie alla gestione delle prenotazioni e gli eventi ad esse collegati
Dipendenze	N\A



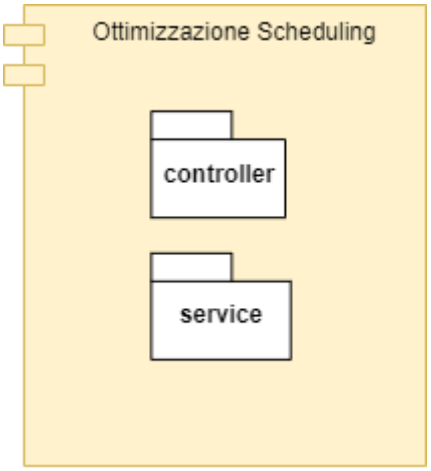
Nome pacchetto	Farmacia
Descrizione	Contiene le classi necessarie alla gestione della farmacia e gli eventi ad essa collegati
Dipendenze	N\A



Nome pacchetto	Ottimizzazione Acquisto
Descrizione	Contiene le classi necessarie alla gestione del modulo di intelligenza artificiale per l'ottimizzazione degli acquisti e gli eventi ad esso collegati
Dipendenze	N\A



Nome pacchetto	Ottimizzazione Scheduling
Descrizione	Contiene le classi necessarie alla gestione del modulo di intelligenza artificiale per l'ottimizzazione dello scheduling e gli eventi ad esso collegati
Dipendenze	N\A



2.2 Organizzazione del codice in file

Avremo otto diversi progetti di seguito elencati:

- Front-End
- API Gateway
- Farmacia
- Paziente
- Utente
- Prenotazione
- Ottimizzazione Acquisto
- Ottimizzazione Scheduling

Ogni classe del sistema sarà collocata nel relativo file. Ognuno di essi sarà quindi collocato nella cartella dedicata.

I nomi dei pacchetti saranno mappati nel rispettivo percorso src/main/java/com/example/… ad eccezione del pacchetto contente i file jsp che saranno collocati nella directory src/main/webapp/WEB-INF/jsp

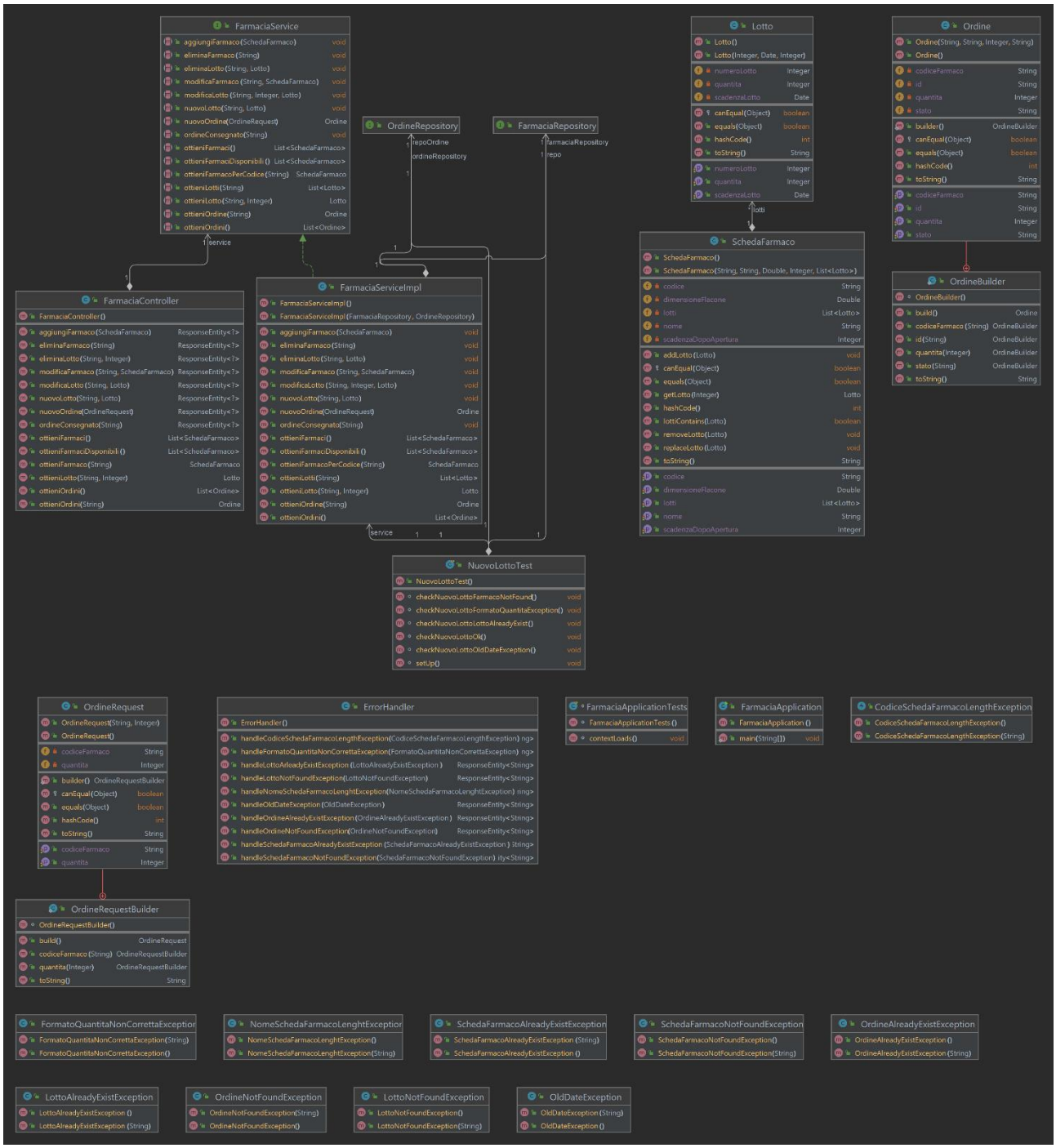
3. Interfacce delle classi

È possibile reperire la documentazione relativa all'interfaccia pubblica delle varie classi nei file Javadoc allegati presenti in /NomeProgetto/javadoc/.

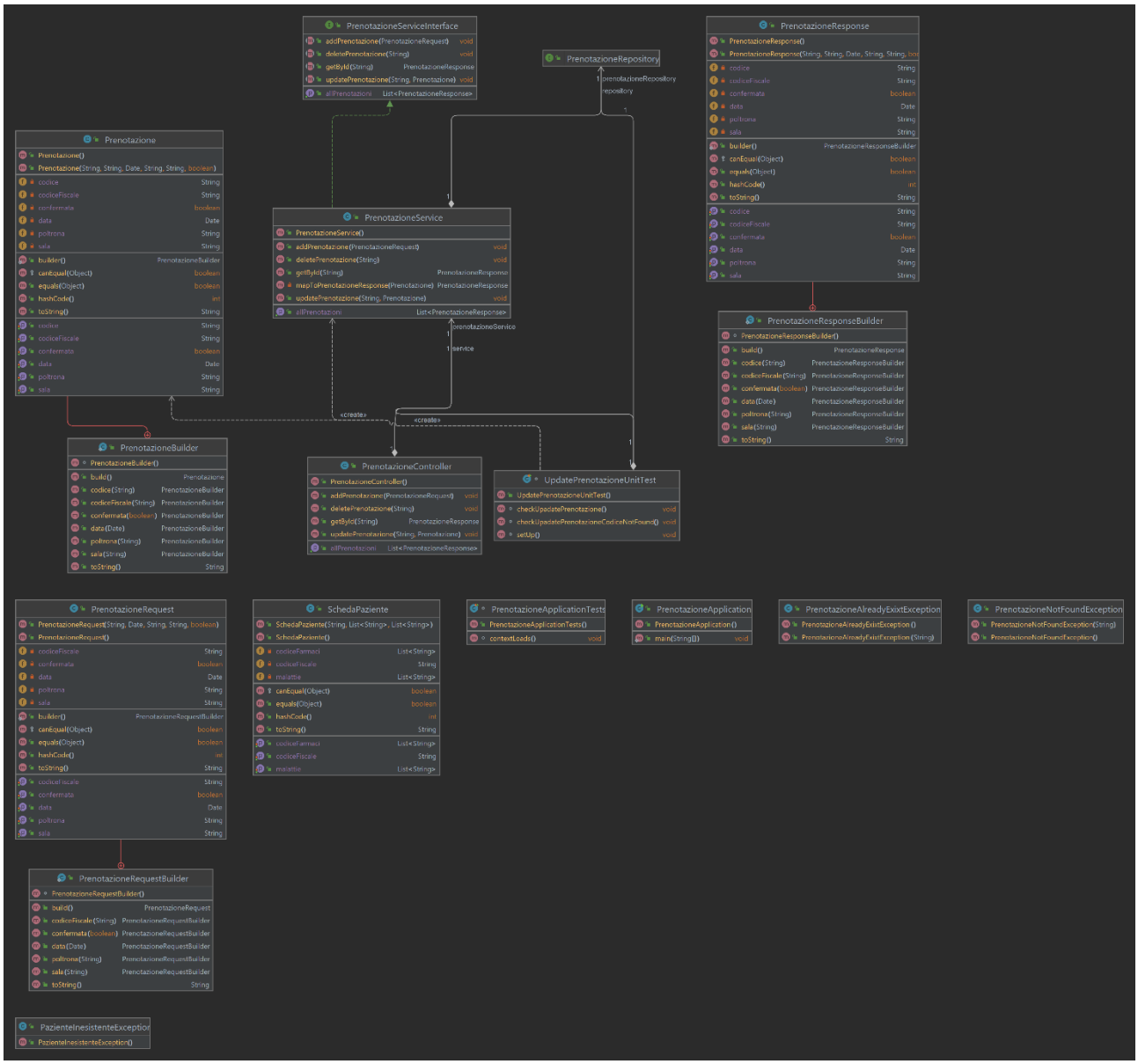
4. Class Diagram

4.1 API Gateway

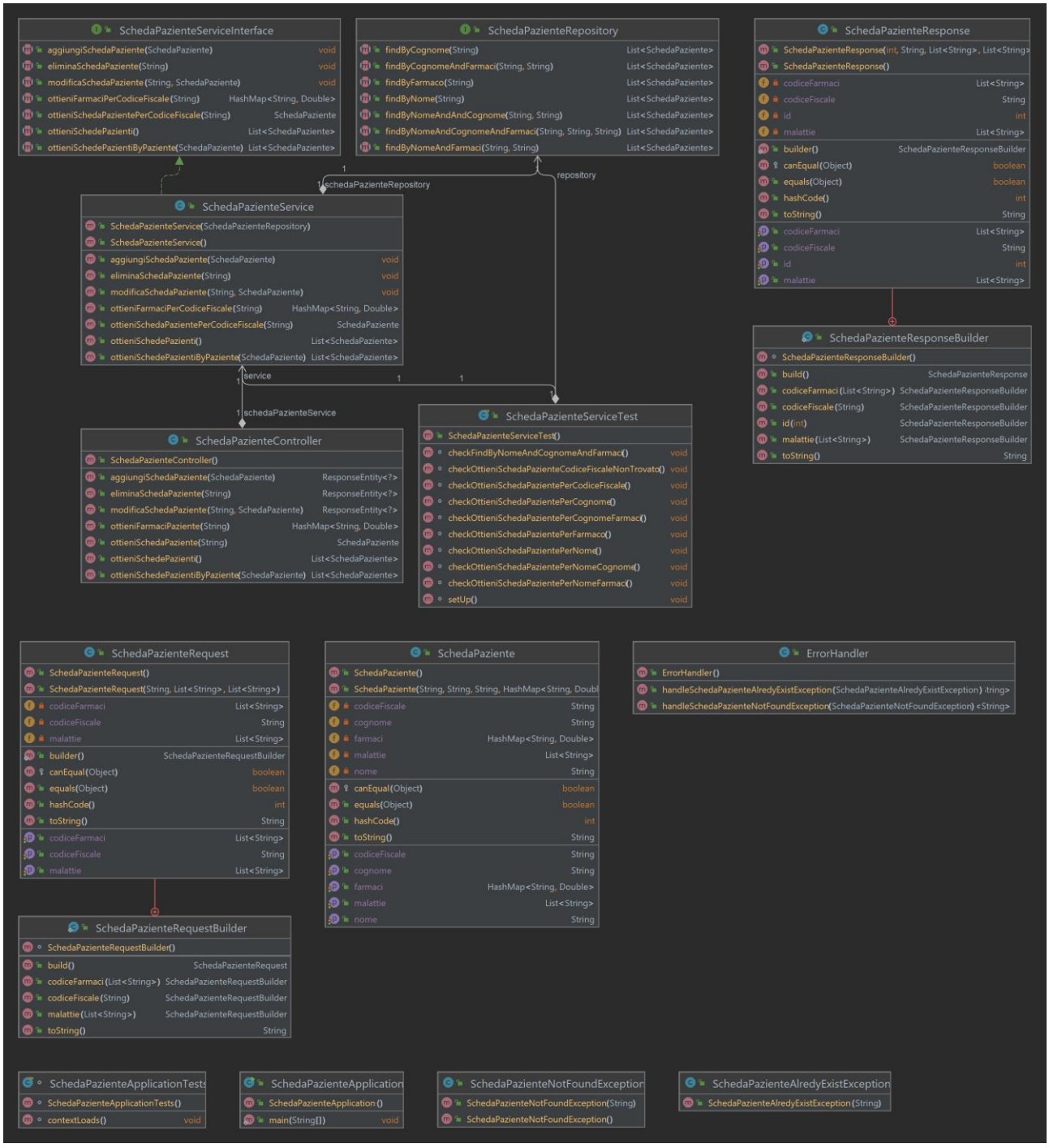
4.2 Farmacia


FarmaciaService
OrdineRepository
FarmaciaRepository
Lotto
Ordine
SchedaFarmaco
OrdineBuilder
FarmaciaController
FarmaciaServiceImpl
NuovoLottoTest
OrdineRequest
ErrorHandler
FarmaciaApplicationTests
FarmaciaApplication
CodiceSchedaFarmacoLengthException
OrdineRequestBuilder
FormatoQuantitaNonCorrettaException
NomeSchedaFarmacoLenghtException
SchedaFarmacoAlreadyExistException
SchedaFarmacoNotFoundException
OrdineAlreadyExistException
LottoAlreadyExistException
OrdineNotFoundException
LottoNotFoundException
OldDateException

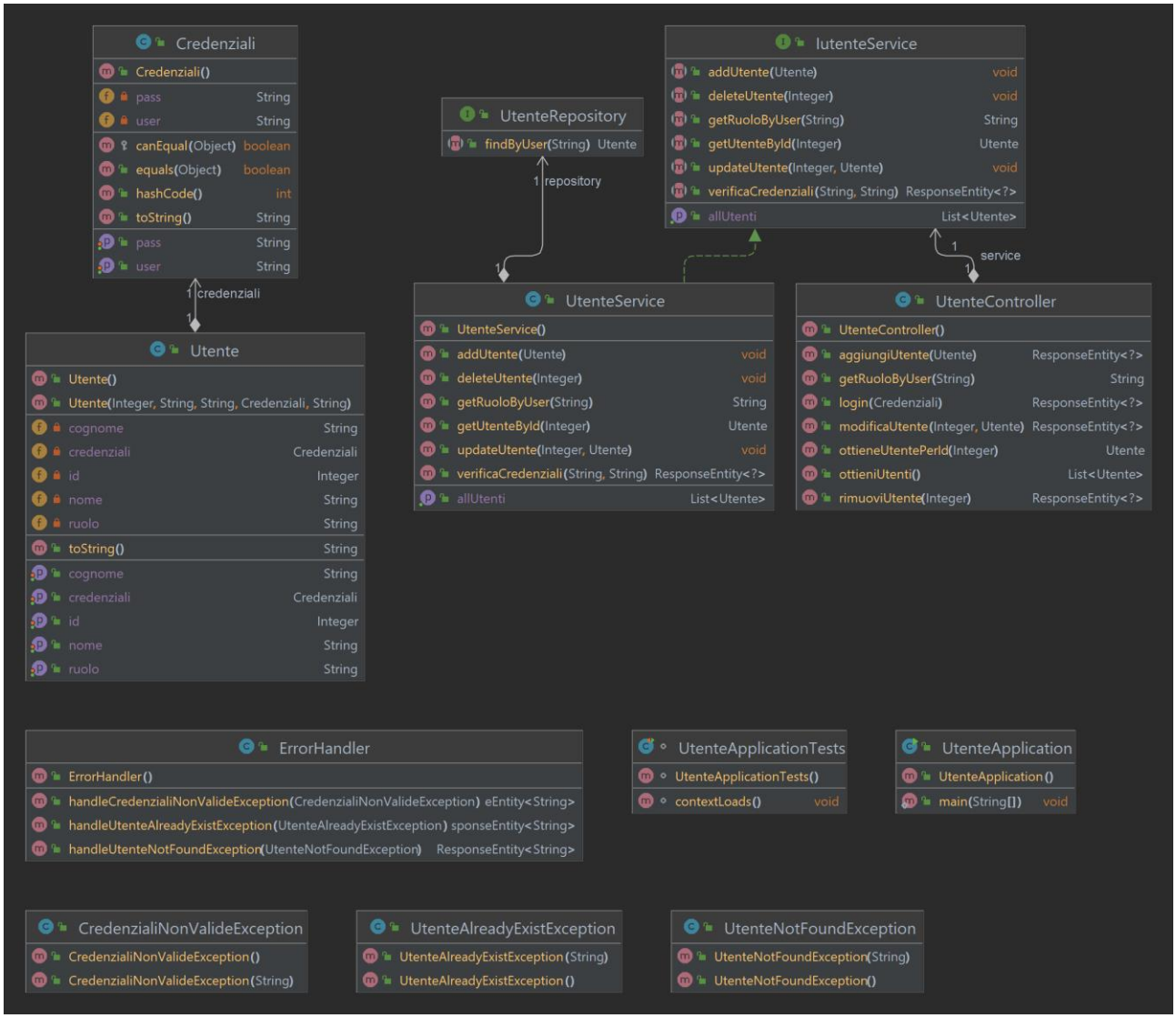
4.3 Prenotazione



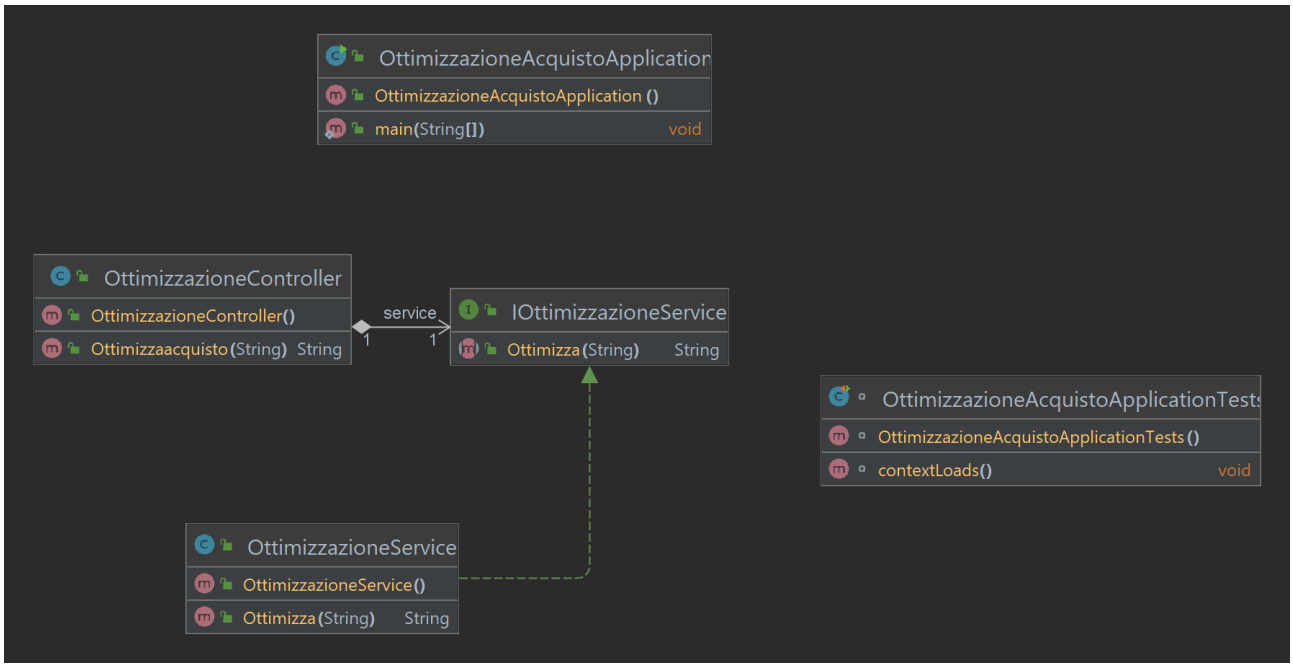
4.4 Scheda Paziente



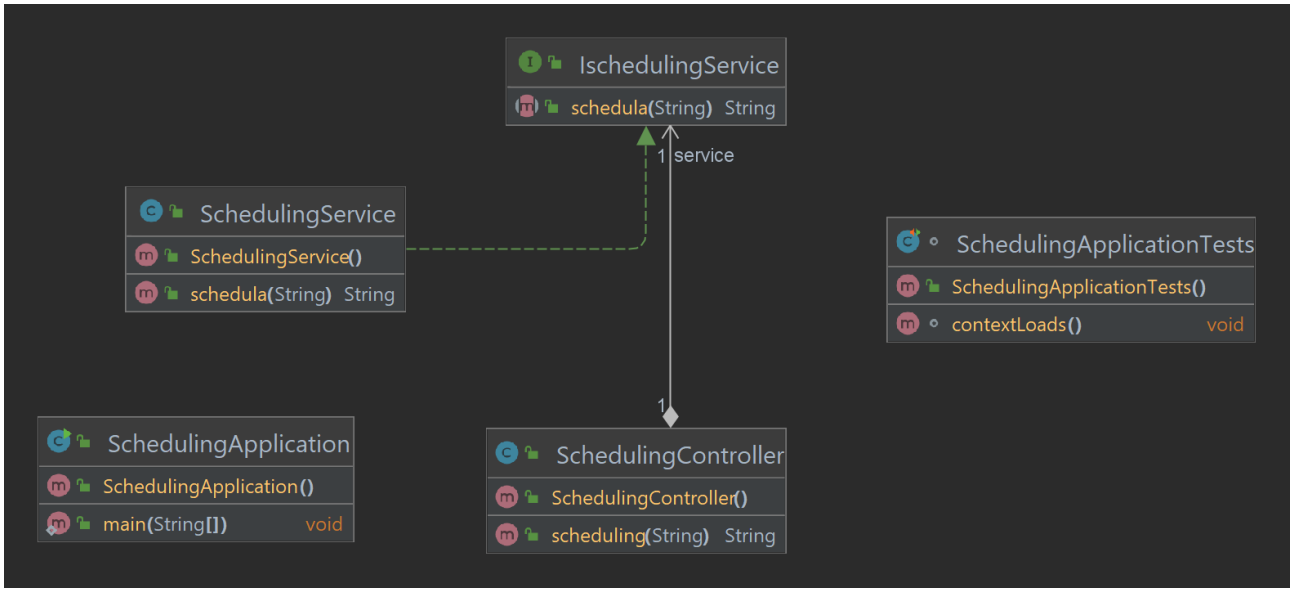
4.5 Utente


Credenziali
Credenziali()
pass String
user String
canEqual(Object) boolean
equals(Object) boolean
hashCode() int
toString() String
pass String
user String
1 credenziali
1
Utente
Utente()
Utente(Integer, String, String, Credenziali, String)
cognome String
credenziali Credenziali
id Integer
nome String
ruolo String
toString() String
cognome String
credenziali Credenziali
id Integer
nome String
ruolo String
UtenteRepository
findByUser(String) Utente
1 repository
1
IutenteService
addUtente(Utente) void
deleteUtente(Integer) void
getRuoloByUser(String) String
getUtenteById(Integer) Utente
updateUtente(Integer, Utente) void
verificaCredenziali(String, String) ResponseEntity<?>
allUtenti List<Utente>
1
service
1
UtenteService
UtenteService()
addUtente(Utente) void
deleteUtente(Integer) void
getRuoloByUser(String) String
getUtenteById(Integer) Utente
updateUtente(Integer, Utente) void
verificaCredenziali(String, String) ResponseEntity<?>
allUtenti List<Utente>
UtenteController
UtenteController()
aggiungiUtente(Utente) ResponseEntity<?>
getRuoloByUser(String) String
login(Credenziali) ResponseEntity<?>
modificaUtente(Integer, Utente) ResponseEntity<?>
ottieneUtentePerId(Integer) Utente
ottieniUtenti() List<Utente>
rimuoviUtente(Integer) ResponseEntity<?>
ErrorHandler
ErrorHandler()
handleCredenzialiNonValideException(CredenzialiNonValideException) eEntity<String>
handleUtenteAlreadyExistException(UtenteAlreadyExistException) sponseEntity<String>
handleUtenteNotFoundException(UtenteNotFoundException) ResponseEntity<String>
UtenteApplicationTests
UtenteApplicationTests()
contextLoads() void
UtenteApplication
UtenteApplication()
main(String[]) void
CredenzialiNonValideException
CredenzialiNonValideException()
CredenzialiNonValideException(String)
UtenteAlreadyExistException
UtenteAlreadyExistException(String)
UtenteAlreadyExistException()
UtenteNotFoundException
UtenteNotFoundException(String)
UtenteNotFoundException()

4.6 Ottimizzazione Acquisto



4.7 Ottimizzazione Scheduling



4.8 Front-End