Laurea Triennale in Informatica - Università di Salerno
Corso di *Ingegneria del Software* - Prof.ssa F. Ferrucci, Prof. F. Palomba

CHEMO SMART
SCHEDULER FOR CHEMOTERAPY

O.D.D.
Object Design Document
ChemoSmart

Versione	2.0
Data	17/02/2023
Destinatario	Prof.ssa F. Ferrucci, Prof. F. Palomba
Presentato da	C. Troiano, M. Purice, L. Miranda, A. Nappi, G. Basile, C. De Palma
Approvato da	A. Bergamo, F. P. Ianuzziello

Revision History

Data	Versione	Descrizione	Autori
13/12/2022	0.1	Prima stesura	LM, CT, AN, GB, MP, CDP
13/12/2022	0.2	Aggiunti Glossario, linee guida e riferimenti a documenti precedenti	LM, CT, AN, GB, MP, CDP
14/12/2022	0.3	Aggiunti nuovi metodi offerti dalle singole classi;	LM, CT, AN, GB, MP, CDP
16/12/2022	1.0	Revisione generale del documento	LM, CT, AN, GB, MP, CDP
07/01/2023	1.1	Revisione generale del documento e revisione della funzione modulo FIA e l'implementazione della classe adapter	LM, CT, AN, GB, MP, CDP
15/01/2023	1.2	Revisione generale del documento.	LM, CT, AN, GB, MP, CDP
17/02/2023	2.0	Revisione generale del documento e aggiunta nuove sezioni	LM, CT, AN, GB, MP, CDP

Team Members

Nome	Ruolo nel progetto	Acronimo	Informazioni di contatto
Alessandro Bergamo	Project Manager	AB	a.bergamo2@studenti.unisa.it
Francesco Pio Ianuzziello	Project Manager	FPI	f.ianuzziello1@studenti.unisa.it
Luigi Miranda	Team Member	LM	l.miranda11@studenti.unisa.it
Ciro Troiano	Team Member	CT	c.troiano17@studenti.unisa.it
Antonio Nappi	Team Member	AN	a.nappi47@studenti.unisa.it
Giuseppe Basile	Team Member	GB	g.basile36@studenti.unisa.it
Mihail Purice	Team Member	MP	m.purice@studenti.unisa.it
Claudio De Palma	Team Member	CDP	c.depalma5@studenti.unisa.it

Sommario

1 Introduzione

ChemoSmart si propone di semplificare la schedulazione delle terapie chemioterapiche, diminuendo lo spreco dei farmaci e ottimizzando il lavoro del personale del reparto di oncologia.
In questa prima sezione del documento verranno illustrati alcuni trade-off e le linee guida utilizzate nella fase di implementazione, riguardanti la documentazione e convenzioni tipiche sui formati e nomi.

1.1 Object design goals

Riusabilità:

Il sistema ChemoSmart deve basarsi sulla riusabilità, attraverso l'utilizzo di ereditarietà e design patterns.

Robustezza:

Il sistema deve risultare robusto, reagendo correttamente a situazioni impreviste attraverso il controllo degli errori e la gestione delle eccezioni.

Incapsulamento:

Il sistema garantisce la segretezza sui dettagli implementativi delle classi grazie all'utilizzo delle interfacce, rendendo possibile l'utilizzo di funzionalità offerte da diversi componenti o layer sottoforma di black-box.

1.2 Linee guida per la documentazione dell'interfaccia

Per la costruzione si è fatto riferimento alle convenzioni JavaScript fornite dal W3C note come **Java Script Coding Conventions.**

JavaScript Coding Conventions: https://www.w3schools.com/js/js_conventions.asp.

1.3 Definizioni, acronimi, e abbreviazioni

Di seguito un elenco di alcuni termini presenti nel documento:

- **Package:** insieme di classi, interfacce e file correlati;
- **Design Pattern:** soluzioni a problemi ricorrenti impiegati per ottenere riuso, flessibilità, manutenibilità;
- **Interfaccia:** insieme delle firme delle operazioni, o metodi, offerti dalla classe;
- **Service:** classi che implementano la logica di business e forniscono un'interfaccia per la comunicazione con il resto del sistema.

- **lowerCamelCase:** standard di scrittura ampiamente utilizzato, consiste nello scrivere frasi in modo tale che ogni parola nel mezzo della frase inizi con una lettera maiuscola, senza spazi o punteggiatura intermedia.
- **UpperCamelCase:** standard di scrittura ampiamente utilizzato, consiste nello scrivere frasi in modo tale che ogni parola della frase inizi con una lettera maiuscola, senza spazi o punteggiatura intermedia.

1.4 Riferimenti

Di seguito una lista dei documenti progettuali:

- Requirements Analysis Document;
- System Design Document;
- Test Plan Document;
- Test Case Specification Document;

2 Packages

In questa sezione viene mostrata la suddivisione del sistema in package, sulla base del System Design. Bisogna notare che per via dell'architettura scelta ogni package rappresenta un microservizio indipendente. I microservizi individuati verranno divisi in altrettanti progetti Node.js. Ogni microservizio rispetterà la struttura qui illustrata:

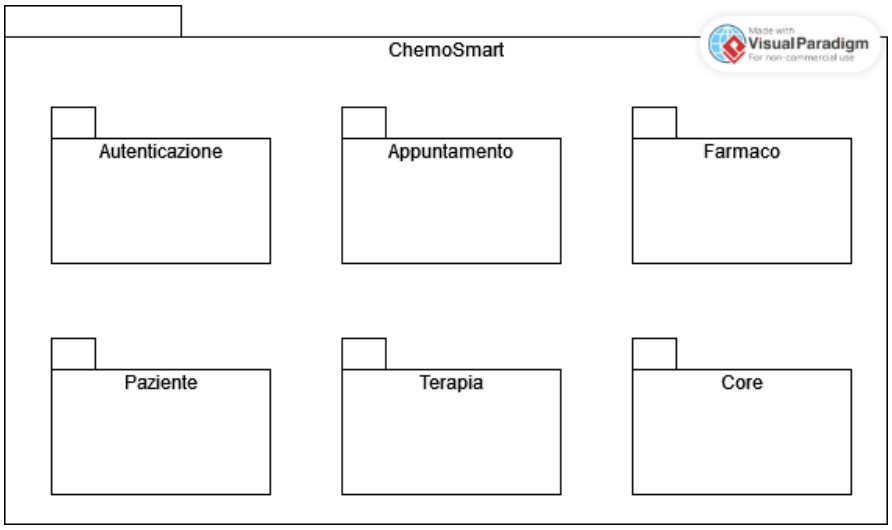
- **controllers,** contiene tutti i file relativi all'implementazione della logica di business, ovvero, tutti i servizi offerti dal microservizio e le interfacce utili per la comunicazione con il resto del sistema;
- **services,** contiene tutti i file relativi all'implementazione delle richieste verso altri microservizi;
- **jsdoc,** contiene tutti i file relativi alla documentazione delle funzioni del controller;
- **models,** contiene il file relativo all'implementazione dello schema mongoose relativo alla collection del database MongoDB;
- **node_modules,** contiene tutti i file prodotti da Node.js
- **routes,** contiene tutti i file che gestiscono il routing di sistema;
- **test,** contiene tutti i file relativi al testing;
- **index.js,** file relativo all'implementazione del server node su cui gira il microservizio;
- **package.json,** file relativo alle dipendenze necessarie al microservizio.

Allo stesso modo il microservizio del package "Core" verrà realizzato come microservizio in grado di unire tutti i servizi del sistema offerti dai microservizi e comunicare con la componente view del sistema. Per tanto la struttura di tale package sarà leggermente diversa:

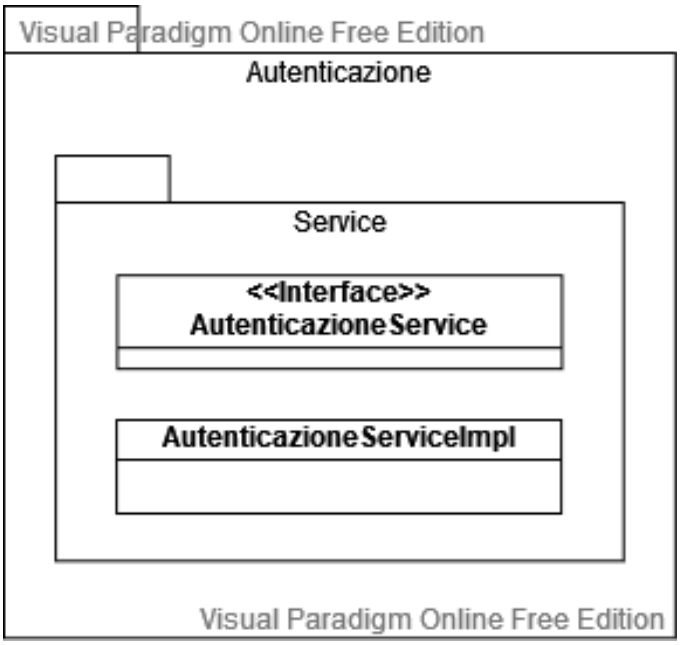
- **assets,** directory per gestire i file statici, quali immagini, file CSS e file JavaScript;
- **node_modules,** contiene tutti i file prodotti da Node.js;
- **services,** contiene tutti i file relativi all'implementazione delle richieste verso altri microservizi;
- **views,** contiene tutti i file relativi alla logica view del sistema, ovvero, il front-end del sistema;
  - **partials,** contiene tutti i componenti parziali che insieme formano una o più viste complete;
- **package.json,** file relativo alle dipendenze necessarie al microservizio.
- **server.js,** file relativo all'implementazione del server node su cui gira il microservizio e utilizza i services per richiedere risorse agli altri microservizi;

Package ChemoSmart

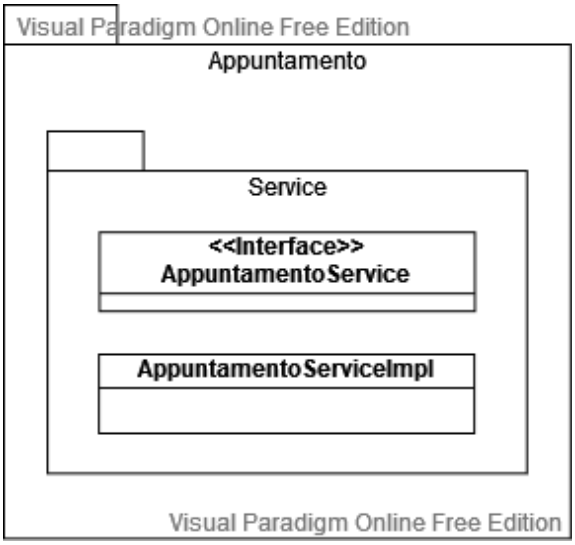
Di seguito è riportata la struttura del package principale di ChemoSmart. Per la suddivisione è stato usato come criterio la suddivisione in package per ogni sottosistema, contenente le classi service ed eventuali classi utili al contesto del sottosistema.



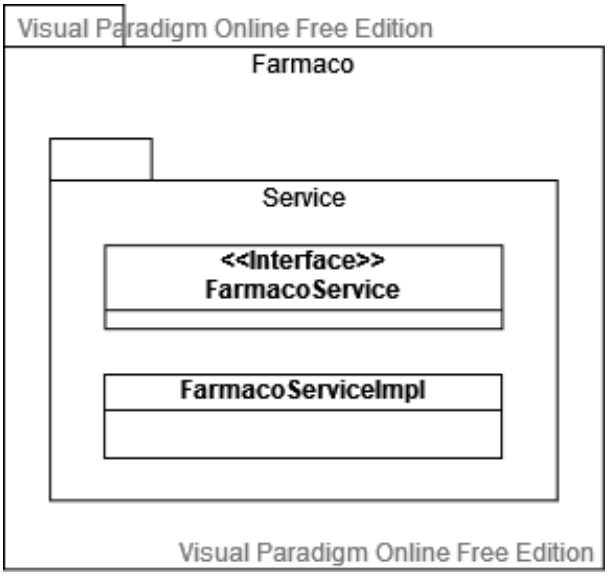
Package Autenticazione



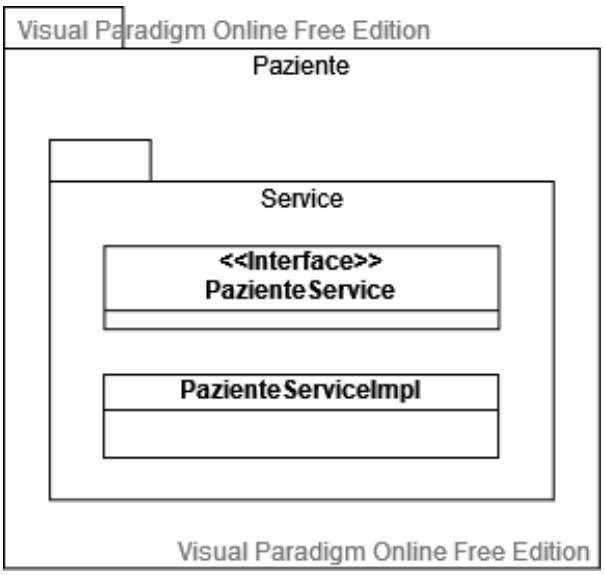
Package Appuntamento



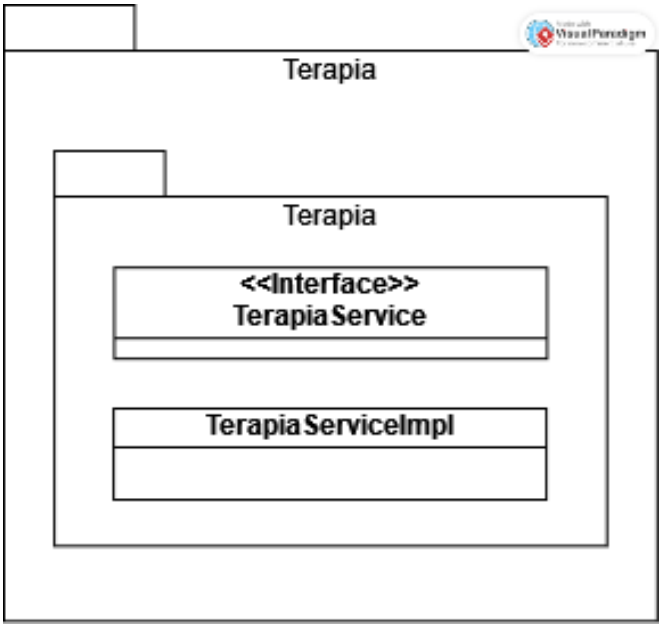
Package Farmaco



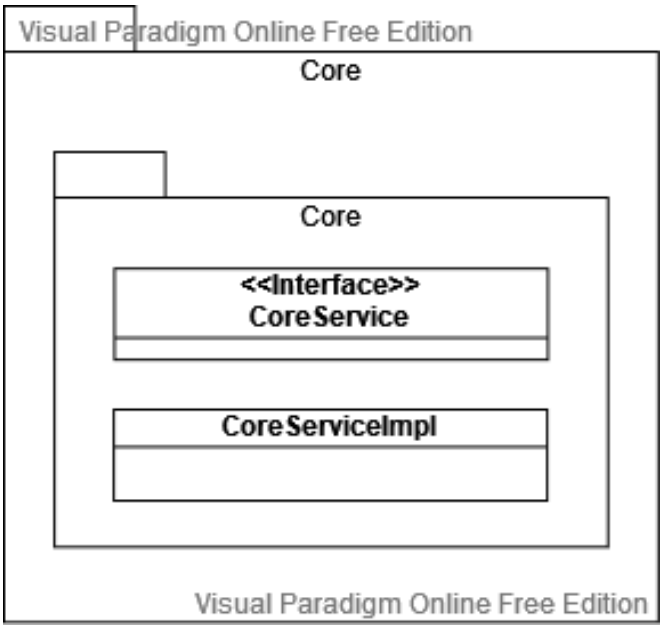
Package Paziente



Package Terapia



Package Core



3 Class Interfaces

3.1 Microservizio Core

Nome Metodo	+addAppuntamento(Appuntamento appuntamento)
Descrizione	Questo metodo consente di aggiungere manualmente un nuovo appuntamento nel calendario.
Pre-condizione	**Context**: GestioneAppuntamentiService::addAppuntamento(appuntamento) **Pre:** not existsAppuntamento(appuntamento)
Post-condizione	**Context:** GestioneAppuntamentiService::addAppuntamento(appuntamento) **Post:** getAppuntamentiOdierni() = @pre.getAppuntamentiOdierni () + 1 **Post:** existsAppuntamento (appuntamento)

Nome Metodo	+createAppuntamento(String cf, String nome, String cognome, String farmaco, Date dataInizio, Date dataFine)
Descrizione	Questo metodo consente di creare un oggetto appuntamento da inserire in una lista di appuntamenti di una terapia.
Pre-condizione	**Context**: GestioneAppuntamentiService::createAppuntamento(String cf, String nome, String cognome, String farmaco, Date dataInizio, Date dataFine) **Pre:** cf != null AND nome!= null AND cognome != null AND farmaco != null AND dataInizio != null AND dataFine != null
Post-condizione	**Context:** GestioneAppuntamentiService:: createAppuntamento(String cf, String nome, String cognome, String farmaco, Date dataInizio, Date dataFine) **Post:** existsAppuntamento (appuntamento)

Nome Metodo	+createAppuntamentiTerapia(String cf, String nome, String cognome, String farmaco, Date dataInizio, Integer durata, Integer numAppuntamenti, Integer frequenza, String priorita):: List<Appuntamento> listaAppuntamenti
Descrizione	Questo metodo consente di creare una lista di appuntamenti relativi ad una terapia.

Pre-condizione	**Context**: GestioneAppuntamentiService::createAppuntamento(String cf, String nome, String cognome, String farmaco, Date dataInizio, Integer durata, Integer numAppuntamenti, Integer frequenza, String priorita) **Pre**: not existsList<Appuntamento> (listaAppuntamenti) **Pre:** cf != null AND nome!= null AND cognome != null AND farmaco != null AND dataInizio != null AND durata!= null AND numAppuntamenti != null AND frequenza != null AND priorita != null
Post-condizione	**Context**: GestioneAppuntamentiService::createAppuntamento(String cf, String nome, String cognome, String farmaco, Date dataInizio, Integer durata, Integer numAppuntamenti, Integer frequenza, String priorita) **Post:** existsList<Appuntamento> (listaAppuntamenti)

Nome Metodo	+updateAppuntamento(String id, Appuntamento appuntamento)
Descrizione	Questo metodo consente di modificare un appuntamento già esistente nel calendario tramite il suo ID.
Pre-condizione	**Context**: GestioneAppuntamentiService::updateAppuntamento(id, appuntamento) **Pre:** existsAppuntamento(appuntamento)
Post-condizione	**Context:** GestioneAppuntamentiService::updateAppuntamento(id, appuntamento) **Post:** appuntamento.getDate() != @pre.appuntamento.getDate() **Post:** getAppuntamentiOdierni () = @pre.getAppuntamentiOdierni () + 1 OR @pre.getAppuntamentiOdierni

Nome Metodo	+getAppuntamenti(): List<Appuntamento>
Descrizione	Questo metodo consente visualizzare il calendario con tutti gli appuntamenti di uno specifico mese.
Pre-condizione	**Context**: GestioneAppuntamentiService::getAppuntamenti() **Pre:** /
Post-condizione	**Context**: GestioneAppuntamentiService::getAppuntamenti() **Post:** /

Nome Metodo	+getAppuntamentoById(String id): Appuntamento
Descrizione	Questa funzione consente di restituire le informazioni di un singolo appuntamento in base al suo ID.

Pre-condizione	**Context**: GestioneAppuntamentiService::getAppuntamentoById(id) **Pre: /**
Post-condizione	**Context**: GestioneAppuntamentiService::getAppuntamentoById(id) **Post: /**

Nome Metodo	+ getPriorita(Paziente paziente)
Descrizione	Questo metodo consente di inviare un paziente al modulo IA per ottenere una priorità.
Pre-condizione	**Context:** GestioneAppuntamentiService::getPriorita(paziente) **Pre:** /
Post-condizione	**Context:** GestioneAppuntamentiService::getPriorita(paziente) **Post:** p.priorita != "None"

Nome Metodo	+getFarmaci(): List<Farmaco> listaFarmaci
Descrizione	Questo metodo consente di visualizzare tutti i farmaci della piattaforma.
Pre-condizione	**Context:** GestioneFarmaciService::getFarmaci() **Pre:** /
Post-condizione	**Context:** GestioneFarmaciService::getFarmaci() **Post:** /
Nome Metodo	+getPazienti(): List<Paziente> listaPazienti
Descrizione	Questo metodo consente di visualizzare tutti i pazienti della piattaforma.
Pre-condizione	**Context:** GestionePazientiService::getPazienti() **Pre:** /
Post-condizione	**Context:** GestionePazientiService:: getPazienti () **Post:** /

Nome Metodo	+updatePaziente(String id, Paziente paziente)
Descrizione	Questo metodo consente di modificare la priorità di un paziente già esistente tramite il suo ID.
Pre-condizione	**Context**: GestionePazienteService::updatePaziente(id, paziente) **Pre:** existsPaziente (paziente)
Post-condizione	**Context:** GestionePazienteService::updatePaziente(id, paziente) **Post:** getPaziente (paziente) != @pre.getPaziente (paziente)

Nome Metodo	+ startTerapia(String id, Date dataInizio, String stato):
Descrizione	Questo metodo consente di inserire una nuova terapia partendo da un determinato giorno e modificare lo stato da "non schedulata" a "in corso".
Pre-condizione	**Context:** GestioneTerapiaService:: startTerapia(String id, Date dataInizio, String stato) **Pre:** dataInizio == null **Pre:** stato== "non schedulata"
Post-condizione	**Context:** GestioneTerapiaService:: startTerapia(String id, Date dataInizio, String stato) **Post:** dataInizio != null **Post:** stato != "non schedulata"

Nome Metodo	+aggiungiTerapia (Terapia terapia)
Descrizione	Questo metodo consente di aggiungere una nuova terapia.
Pre-condizione	**Context**: GestioneTerapiaService::aggiungiterapia(terapia)

	Pre: not existsTerapia(terapia)
Post-condizione	**Context:** GestioneTerapiaService::aggiungiTerapia(terapia) **Post:** existsTerapia(terapia)

Nome Metodo	+getTerapie(): List<Terapia> listaTerapie
Descrizione	Questo metodo consente visualizzare tutte le terapie della piattaforma.
Pre-condizione	**Context**: GestioneTerapiaService::getTerapie() **Pre:** /
Post-condizione	**Context**: GestioneTerapiaService::getTerapia() **Post:** /

Nome Metodo	+getTerapiaById(String id): Terapia terapia
Descrizione	Questa funzione consente di restituire le informazioni di una singola terapia in base al suo ID.
Pre-condizione	**Context**: GestioneTearpiaService::getTerapiaById(id) **Pre: /**
Post-condizione	**Context**: GestioneTerapiaService::getTerapiaById(id) **Post: /**

Nome Metodo	+updateTerapia(String id, Terapia terapia)
Descrizione	Questo metodo consente di modificare una terapia tramite il suo ID.
Pre-condizione	**Context**: GestioneTerapiaService::updateTerapia(id, terapia) **Pre:** existsTerapia(terapia)
Post-condizione	**Context**: GestioneTerapiaService::updateTerapia(id, terapia) **Post:** getStato(terapia) != @pre.getStato(terapia)

3.2 Microservizio Gestione Appuntamenti

Nome Metodo	+insertAppuntamento(Request req, Response res)
Descrizione	Questo metodo consente di aggiungere manualmente un nuovo appuntamento nel database.

Pre-condizione	**Context**: GestioneAppuntamentiService::insertAppuntamento Request req, Response res) **Pre:** not existsAppuntamento(req)
Post-condizione	**Context:** GestioneAppuntamentiService::insertAppuntamento Request req, Response res) **Post:** existsAppuntamento (req)

Nome Metodo	+getAllAppuntamenti(Request req, Response res): List<Appuntamento> listaAppuntamenti
Descrizione	Questo metodo restituisce una lista di tutti gli appuntamenti del database.
Pre-condizione	**Context**: GestioneAppuntamentiService::getAllAppuntamenti(Request req, Response res) **Pre:** /
Post-condizione	**Context**: GestioneAppuntamentiService:: getAllAppuntamenti (Request req, Response res) **Post: /**

Nome Metodo	+getAppuntamentoById(Request req, Response res): Appuntamento appuntamento
Descrizione	Questo metodo restituisce un appuntamento dal database in base al suo ID.
Pre-condizione	**Context**:GestioneAppuntamentiService::getAppuntamentoById(Request req, Response res) **Pre**: /
Post-condizione	**Context**:GestioneAppuntamentiService::getAppuntamentoById(Request req, Response res) **Post:** /

Nome Metodo	+deleteAppuntamento(Request req, Response res)
Descrizione	Questo metodo consente di eliminare un appuntamento dal database.

Pre-condizione	**Context**: GestioneAppuntamentiService:: deleteAppuntamento(Request req, Response res) **Pre:** /
Post-condizione	**Context**: GestioneAppuntamentiService:: deleteAppuntamento(Request req, Response res) **Post:** /

Nome Metodo	+updateAppuntamento(Request req, Response res):Appuntamento appuntamento
Descrizione	Questo metodo consente di modificare un appuntamento del database.
Pre-condizione	**Context**:GestioneAppuntamentiService::updateAppuntamento(Request req, Response res) **Pre:** /
Post-condizione	**Context**:GestioneAppuntamentiService::updateAppuntamento(Request req, Response res) **Post:** /

3.3 Microservizio Gestione Terapia

Nome Metodo	+insertTerapia (Request req, Response res)
Descrizione	Questo metodo consente di aggiungere manualmente una nuova terapia nel database.
Pre-condizione	**Context**: GestioneTerapiaService::insertTerapia (Request req, Response res) **Pre:** not existsTerapia(req)
Post-condizione	**Context:** GestioneTerapiaService::insertTerapia(Request req, Response res) **Post:** existsTerapia(req)

Nome Metodo	+getAllTerapie(Request req, Response res): List<Terapia> listaTerapie

Descrizione	Questo metodo restituisce una lista di tutti gli appuntamenti del database.
Pre-condizione	**Context**: GestioneTerapiaService::getAllTerapie(Request req, Response res) **Pre:** /
Post-condizione	**Context**: GestioneTerapiaService::getAllTerapie(Request req, Response res) **Post:** /

Nome Metodo	+getTerapiaById(Request req, Response res):Terapia terapia
Descrizione	Questo metodo restituisce una terapia dal database in base al suo ID.
Pre-condizione	**Context**:GestioneTerapiaService::getTerapiaById(Request req, Response res) **Pre**: /
Post-condizione	**Context**:GestioneTerapiaService::getTerapiaById(Request req, Response res) **Post:** /

Nome Metodo	+deleteTerapia(Request req, Response res)
Descrizione	Questo metodo consente di eliminare una terapia dal database.
Pre-condizione	**Context**: GestioneTerapieService:: deleteTerapia(Request req, Response res) **Pre:** /
Post-condizione	**Context**:GestioneTerapiaService::getTerapiaById(Request req, Response res) **Post:** /

Nome Metodo	+updateTerapia(Request req, Response res):Terapia terapia
Descrizione	Questo metodo consente di modificare una terapia del database.

Pre-condizione	**Context**:GestioneTerapiaService::updateTerapia(Request req, Response res) **Pre:** /
Post-condizione	**Context**:GestioneTerapiaService::updateTerapia(Request req, Response res) **Post:** /

3.4 Microservizio Gestione Paziente

Nome Metodo	+insertPaziente(Request req, Response res)
Descrizione	Questo metodo consente di aggiungere manualmente un nuovo paziente nel database.
Pre-condizione	**Context**: GestionePazienteService::insertPaziente(Request req, Response res) **Pre:** not existsPaziente(req)
Post-condizione	**Context**: GestionePazienteService::insertPaziente(Request req, Response res) **Post:** existsPaziente(req)

Nome Metodo	+getAllPazienti(Request req, Response res): List<Paziente> listaPazienti
Descrizione	Questo metodo restituisce una lista di tutti i pazienti del database.
Pre-condizione	**Context**: GestionePazienteService::getAllPazienti(Request req, Response res) **Pre:** /
Post-condizione	**Context**: GestionePazienteService::getAllPazienti(Request req, Response res) **Post:** /

Nome Metodo	+getPazienteById(Request req, Response res):Paziente paziente
Descrizione	Questo metodo restituisce un paziente dal database in base al suo ID.

Pre-condizione	**Context**:GestionePazienteService::getPazienteById(Request req, Response res) **Pre**: /
Post-condizione	**Context**:GestionePazienteService::getPazienteById(Request req, Response res) **Post:** /

Nome Metodo	+deletePazienteRequest req, Response res)
Descrizione	Questo metodo consente di eliminare un paziente dal database.
Pre-condizione	**Context**: GestionePazienteService:: deletePaziente(Request req, Response res) **Pre:** /
Post-condizione	**Context**: GestionePazienteService:: deletePaziente(Request req, Response res) **Post:** /

Nome Metodo	+updatePaziente(Request req, Response res):Paziente paziente
Descrizione	Questo metodo consente di modificare un paziente del database.
Pre-condizione	**Context**:GestionePazienteService::updatePaziente(Request req, Response res) **Pre:** /
Post-condizione	**Context**:GestionePazienteService::updatePaziente(Request req, Response res) **Post:** /

3.5 Microservisio Gestione Farmaci

Nome Metodo	+insertFarmaco(Request req, Response res)
Descrizione	Questo metodo consente di aggiungere un nuovo farmaco nel database.

Pre-condizione	**Context**: GestioneFarmacoService::insertFarmaco(Request req, Response res) **Pre:** not existsFarmaco(req)
Post-condizione	**Context**: GestioneFarmacoService::insertFarmaco(Request req, Response res) **Post:** existsFarmaco(req)

Nome Metodo	+getAllFarmaci(Request req, Response res): List<Farmaco> listaFarmaci
Descrizione	Questo metodo restituisce una lista di tutti i farmaci del database.
Pre-condizione	**Context**: GestioneFarmacoService::getAllFarmaci(Request req, Response res) **Pre:** /
Post-condizione	**Context**: GestioneFarmacoService::getAllFarmaci(Request req, Response res) **Post:** /

Nome Metodo	+getFarmacoyId(Request req, Response res):Farmaco farmaco
Descrizione	Questo metodo restituisce un farmaco dal database in base al suo ID.
Pre-condizione	**Context**: GestioneFarmacoService::getFarmacoByIDRequest(Request req, Response res) **Pre**: /
Post-condizione	**Context**: GestioneFarmacoService::getFarmacoByIDRequest(Request req, Response res) **Post:** /

Nome Metodo	+deleteFarmaco(Request req, Response res)
Descrizione	Questo metodo consente di eliminare un farmaco dal database.

Pre-condizione	**Context**: GestioneFarmacoService:: deleteFarmaco(Request req, Response res) **Pre:** /
Post-condizione	**Context**: GestioneFarmacoService:: deleteFarmaco(Request req, Response res) **Post:** /

Nome Metodo	+updateFarmaco(Request req, Response res):Farmaco farmaco
Descrizione	Questo metodo consente di modificare un farmaco del database.
Pre-condizione	**Context**:GestioneFarmacoService::updateFarmaco(Request req, Response res) **Pre:** /
Post-condizione	**Context**:GestioneFarmacoService::updateFarmaco(Request req, Response res) **Post:** /

3.6 Autenticazione

Nome Metodo	+Login(String email, String password)
Descrizione	Questo metodo consente di loggare un utente registrato.
Pre-condizione	**Context**: AutenticazioneService::Login(email,password) **Pre:** /
Post-condizione	**Context**: AutenticazioneService::Login(email,password) **Post**: isMedico(loggedUser) | |isSegretario(loggeUser() | |isInfermiere()==true

Nome Metodo	+Logout(String email,String password)
Descrizione	Questo metodo consente di scollegare un utente registrato
Pre-condizione	**Context**: AutenticazioneAService::Logout(email,password) **Pre:** /
Post-condizione	**Context**: AutenticazioneAService::Logout(email,password) **Post**: isMedico(loggedUser) | |isSegretario(loggeUser() | |isInfermiere()==true

4 Design Patterns

In questa sezione verranno descritti nel dettaglio i design patterns utilizzati nello sviluppo del sistema ChemoSmart. Per ogni pattern verrà fornito:

- Una introduzione teorica.
- Il problema che risolve nel sistema ChemoSmart.
- Una spiegazione della risoluzione del problema.
- Un grafico della struttura delle classi che implementano il pattern.

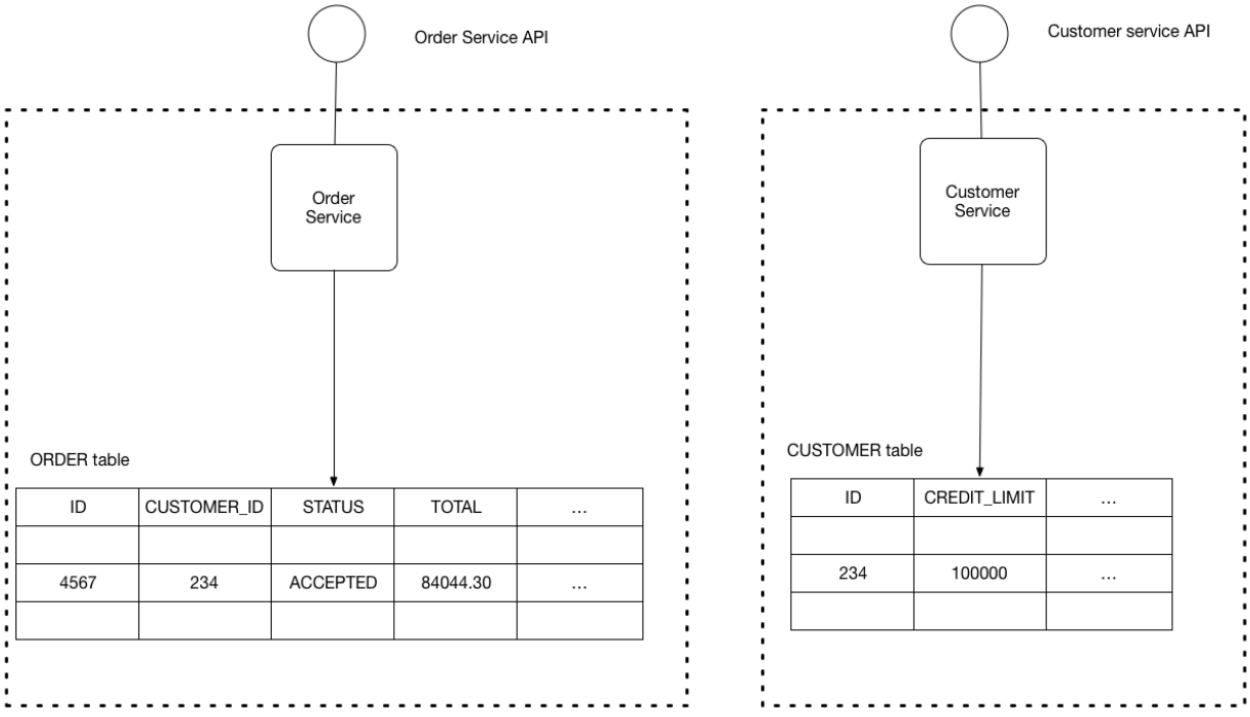
Database-per-Service Pattern

Il Database-per-Service pattern viene utilizzato spesso, ma non solo, nell'ambito di applicativi realizzati tramite il Design Pattern Architetturale dei **Microservizi.** I microservizi, per natura, sono realizzati cercando di massimizzare l'autonomia e avvicinandosi quanto piu possibile al concetto di **loose coupling.** Il Database-per-Service pattern è un pattern utilizzato per realizzare tale indipendenza fra microservizi.

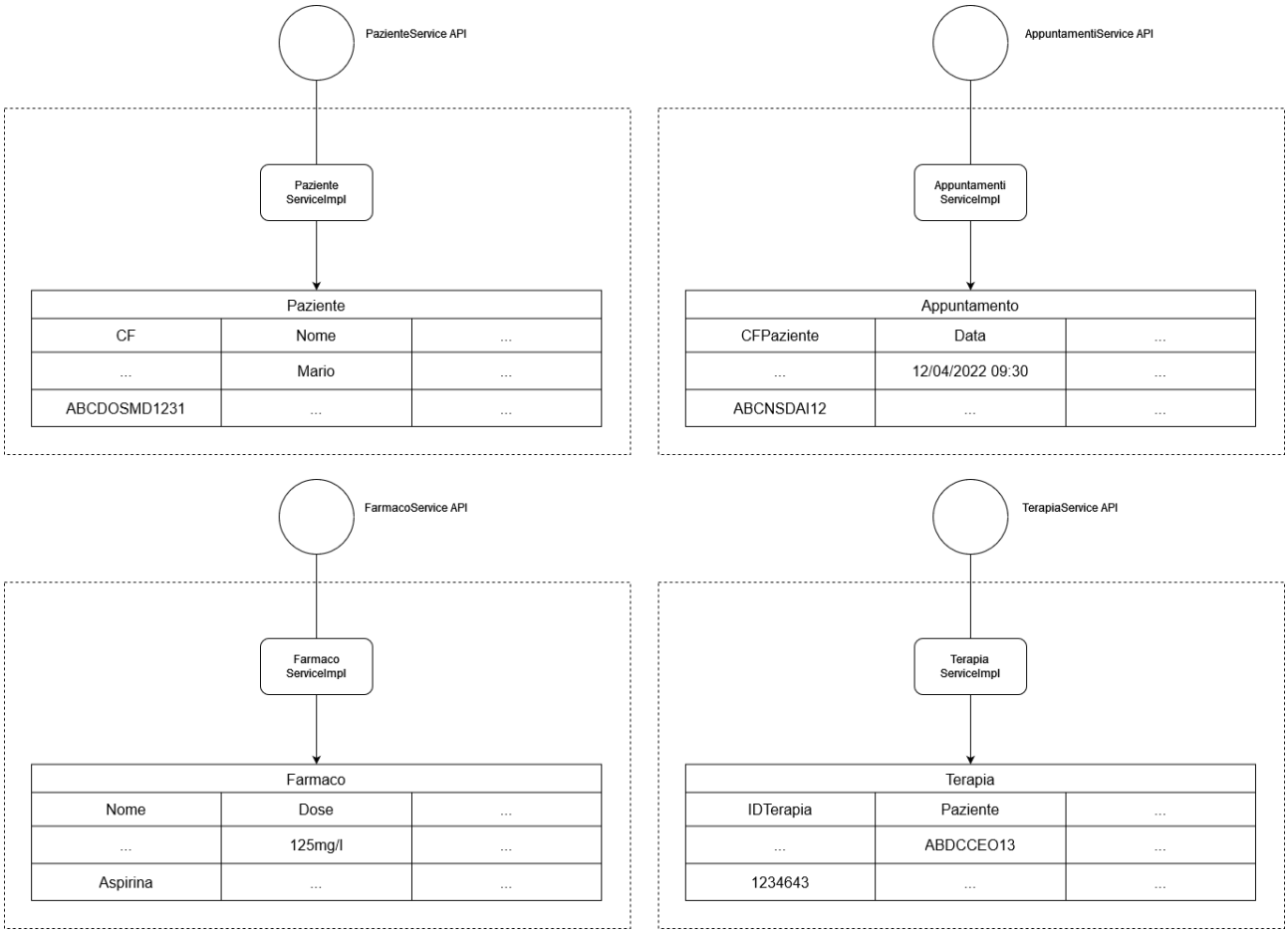
Questo design pattern ci permette di separare completamente anche la gestione dei dati persistenti ai soli microservizi che gestiscono quest'ultimi. In particolare, il database viene completamente "incorporato" nella logica di business del microservizio che sarà l'unico a poter operare direttamente su di esso e fornire al resto del sistema la possibilità di utilizzare tali dati persistenti solo tramite l'interfaccia pubblica implementata stesso dal microservizio.

In particolare nel nostro sistema questo design pattern è stato utilizzato sin dal principio in modo da realizzare dei microservizi in grado di gestire completamente i diversi tipi di dati persistenti che il sistema utilizza.

Ogni microservizio è realizzato seguendo l'esempio di implementazione del pattern del seguente grafico:



In particolare, applicando il grafico di esempio al nostro sistema, abbiamo quindi:



5 Glossario

Sigla/Termine	Definizione
Package	Raggruppamento di Classi ed Interfacce
Service	Classe che implementa la logica di business; viene utilizzata dal API Client.
Adapter	Un design pattern strutturale che permette, attraverso un'interfaccia, di fare collaborare oggetti con interfacce diverse.
Design Pattern	descrizione o modello logico da applicare per la risoluzione di un problema che può presentarsi in diverse situazioni durante le fasi di progettazione e sviluppo del software.
Design Pattern Strutturale	Design pattern che mirano a risolvere problematiche inerenti alla struttura delle classi e degli oggetti.