Laurea Magistrale in informatica - Università di Salerno
Corso di *Gestione dei Progetti Software*- Prof.ssa F.Ferrucci, Prof. F. Palomba

ODD – Object Design Document Progetto TalkAID

Riferimento	C16_ODD_ver.1.7
Versione	1.7
Data	***19/01/2024***
Destinatario	Dipartimento di Informatica dell'Università degli studi di Salerno
Presentato da	C16 Team Member
Approvato da	Carmine Pastore e Nicola Laurino

Revision History

Data	Versione	Descrizione	Autori
20/10/2023	1.0	Prima stesura	Michele D'Arienzo
07/12/2023	1.1	Aggiunte sezioni da 1.1 a 1.4	Cristian Porzio
11/12/2023	1.2	Aggiunti chiarimenti ai Packages	Cristian Porzio
13/12/2023	1.3	Aggiunto sezione 4 e 5	Raffaele Monti
30/12/2023	1.4	Aggiunte Class Interfaces Implementate	Cristian Porzio
09/01/2024	1.5	Revisione pre-consegna	Cristian Porzio
17/01/2024	1.6	Ultima revisione	Anna Benedetta Salerno
18/01/2024	1.7	Revisione completa	Tutti i Membri

Team Composition

Ruolo	Nome	Posizione	Contatti
Top Manager	Filomena Ferrucci	Rappresentante del cliente	fferrucci@unisa.it
Top Manager	Fabio Palomba	Rappresentante del cliente	fpalomba+is@unisa.it
Project Manager	Nicola Laurino	Project Manager	n.laurino1@studenti.unisa.it
Project Manager	Carmine Pastore	Project Manager	c.pastore35@studenti.unisa.it
Team Member	Michele D'Arienzo	Team Member	m.darienzo20@studenti.unisa.it
Team Member	Raffaele Monti	Team Member	r.monti2@studenti.unisa.it
Team Member	Cristian Porzio	Team Member	c.porzio3@studenti.unisa.it
Team Member	Luigi Salvatore Pio Petrillo	Team Member	l.petrillo6@studenti.unisa.it
Team Member	Anna Benedetta Salerno	Team Member	a.salerno45@studenti.unisa.it
Team Member	Samuele Sparno	Team Member	s.sparno@studenti.unisa.it

Sommario

Object Design Document (ODD)
del Progetto TalkAID

1. Introduzione

Il progetto TalkAID rappresenta un passo avanti significativo nel campo della riabilitazione e del supporto alle persone con disturbi del linguaggio. La possibilità di offrire trattamenti completamente a distanza e in maniera asincrona è un'innovazione che potrebbe aprire nuove opportunità per un numero ancora maggiore di individui.

La componente di Intelligenza Artificiale è ritenuta fondamentale ai nostri obiettivi siccome aggiunge un livello di personalizzazione e adattabilità, permettendo ai pazienti di ricevere esercizi mirati in base ai loro errori e difficoltà. Questo non solo rende il trattamento più efficace, ma anche più coinvolgente per i pazienti, incoraggiandoli a perseguire con impegno il percorso di miglioramento.

1.1 Object Design Goals

Trade-Off	Descrizione
Affidabilità vs Prestazioni	Dovendo il sistema gestire dati sensibili, essere stabile ed affidabile, si preferisce eccedere nell'utilizzo di risorse ed avere un minore controllo dell'efficienza delle operazioni invece di compromettere l'affidabilità.
Sicurezza vs Prestazioni	Il sistema deve garantire il massimo livello di sicurezza e privacy per i dati personali e terapeutici. Tuttavia, questo può talvolta rendere il sistema meno prestante, in quanto può richiedere risorse e misure aggiuntive per garantire la sicurezza.
Manutenibilità vs Alta Ottimizzazione	Manutenere un sistema altamente ottimizzato è cruciale per le performance, può scontrarsi con l'obiettivo di una semplice manutenibilità. Del codice eccessivamente ottimizzato può a volte essere più difficile da capire e modificare, portando a potenziali sfide in aggiornamenti e modifiche future.

1.2 Componenti Off-the-Shelf

TalkAID utilizzerà alcuni software off-the-shelf sia per il front-end che per il back-end quali: HTML5, CSS3, JavaScript, jQuery, Bootstrap, Python3, MySQL, Tomcat, Maven, FFmpeg e Azure AI. La connessione al database avverrà grazie alla componente MySQL Connector/J, conosciuto come Driver JDBC ufficiale per MySQL per permettere la comunicazione tra il codice sorgente Java e la base di dati.

- **Tomcat**: web server open-source sviluppato da Apache Software Foundation. Fondamentale per l'intera esecuzione della business implementation del sistema.
- **Maven**: è uno strumento di gestione di progetti software basati su Java. Agevola la gestione delle librerie utilizzate, permettendo agli sviluppatori di concentrarsi esclusivamente sulla scrittura del codice.
- **MySQL**: famoso RDBMS di Oracle. Tutti i dati persistenti risiederanno al suo interno.
- **FFmpeg**: suite di librerie e programmi per gestire file e flussi multimediali come video, audio e altri formati.
- **Azure AI**: fornisce lo Speech to Text, un servizio utilizzato per la trascrizione in tempo reale o in batch di flussi audio in testo. Inoltre, fornisce l'Accuracy Rate del parlante e numerosi parametri per calcolarla.

1.3 Linee guida per la documentazione dell'Interfaccia

Facendo riferimento alla convenzione Java nota come **Sun Java Coding Conventions** ed alle numerose guide accessibili online per lo sviluppo web, è richiesto agli sviluppatori di rispettare le seguenti linee guida al fine di sviluppare del codice che risulti consistente e di lettura non mutevole:

Lista di Link alle convenzioni per definire le linee guida:

- Java Sun: https://checkstyle.sourceforge.io/sun_style.html

Tipi	Regole per la Denominazione	Esempi
Package	È un raggruppamento di classi, interfacce o file correlati. Il prefisso di un nome di pacchetto univoco è scritto in lower camelCase se non per acronimi come UI	package controller; package src;
Design Pattern	Template di soluzioni a problemi ricorrenti impiegati per ottenere riuso e flessibilità.	More Info
Classe	È la definizione di un tipo di oggetto. I nomi delle classi sono sostantivi scritti in Upper CamelCase. I nomi delle classi sono semplici e descrittivi ed utilizzano parole intere od abbreviazioni comuni.	class DAOUser; class Exercise
Interfaccia	Insieme di signature elargite dalla classe. I nomi delle interfacce devono essere scritti in Upper CamelCase come i nomi delle classi	Interface Identifier;
Metodo	Raggruppamento di un'operazione che manipola dati od oggetti. I metodi sono verbi scritti in lower camelCase ed esprimono brevemente l'azione.	insert(); createUser();
Variabile	'Contenitori' in cui è possibile memorizzare valori che serviranno in seguito. I nomi delle variabili sono scritti in lower camelCase. Se non per utilizzi come indici, il loro nome fa sempre riferimento a ciò che contiene. Possono contenere abbreviazioni per migliorarne la leggibilità.	Int i; String name; String userID
Costante	'Contenitori' il cui valore memorizzato non può cambiare. I nomi delle costanti devono essere completamente in maiuscolo.	static final PREFISSO = 39
Javadoc	Commenti del codice esportabili automaticamente tramite JDK. Il codice deve essere commentato in italiano diretto utilizzando terminologie dell'ambito informatico.	**/**** * *spiegazione del metodo* * **@param** *qualcosa contiene...* * **@return** questo oggetto ***/**
View	Rappresenta ciò che è visualizzato a schermo da un utente, offrendo alcune funzionalità. Impiegato nel pattern MVC	More Info

Controller	Rappresenta il lato business che si interpone tra una View e, se necessario tra uno o più Model. Gestisce le operazioni e controlli da effettuare ad ogni interazione.	More Info
Model	Rappresenta la struttura dei dati e le relative operazioni. Fornisce metodi per accedere ai dati utili al sistema.	More Info
Lower CamelCase	Pratica di scrivere la prima parola interamente in minuscolo, mentre la prima lettera di ogni parola successiva è maiuscola.	numberOfDonuts favePhrase userInput
Upper CamelCase	Pratica di scrivere la prima lettera di ogni parola in maiuscolo.	NumberOfDonuts FavePhrase
Naming Convention	Convenzione per nominare gli oggetti. Per una questione di leggibilità è **obbligatorio** utilizzare la lingua inglese. I nomi delle classi possono invece essere espressi in italiano.	

1.4 Riferimenti

Di seguito una lista di riferimenti ad altri documenti utili durante la lettura:

- Statement of Work
- Business Case
- Requirements Analysis Document
- System Design Document
- TestPlan
- Matrice di Tracciabilità
- Manuale di Installazione
- Manuale Utente

Di seguito il link al sito contenente il javadoc di TalkAID

- Javadoc

2. Packages

In questa sezione viene mostrata la suddivisione del sistema in package, in base a quanto definito nel documento di System Design. Tale suddivisione è motivata dalle scelte architetturali prese e ricalca la struttura di directory standard definita da Maven.

- **.idea**
- **.mvn**, contiene tutti i file di configurazione per Maven
- **src**, contiene tutti i file sorgente
    - **main**
        - **java,** contiene tutte le classi relative alle componenti di sistema
            - **controller**, contiene le servlet del sistema.
            - **model**, contiene le classi e bean per memorizzare le informazioni delle varie componenti nel sistema.
                - **DAO**, contiene tutte le classi che si interfacciano col database fornendo metodi e query per esso.
                - **entity**, contiene le classi che permettono di mappare i dati dal database ad oggetti Java che verranno successivamente manipolati dalla logica di business.
                - **service**, contiene le classi relative ai singoli sottosistemi individuati per la logica di business.

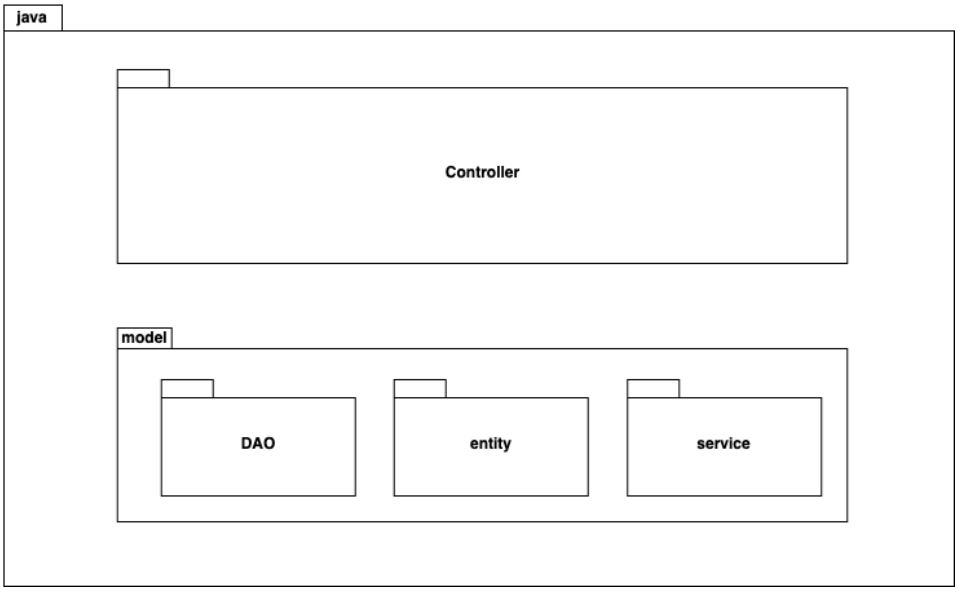
        - **webapp,** contiene tutte le componenti relative al Web
            - **JSP**, contiene tutte le pagine del sistema visitabili.
            - **JS**, contiene tutti gli script JavaScript.
            - **CSS**, contiene i fogli di stile.
            - **img**, contiene le immagini usate nelle pagine.
    - **test**, contiene tutto il necessario per il testing di unità ed integrazione.
        - **java**, contiene le classi Java per l'implementazione del testing
- **target**, contiene tutti i file prodotti dal sistema di build di Maven

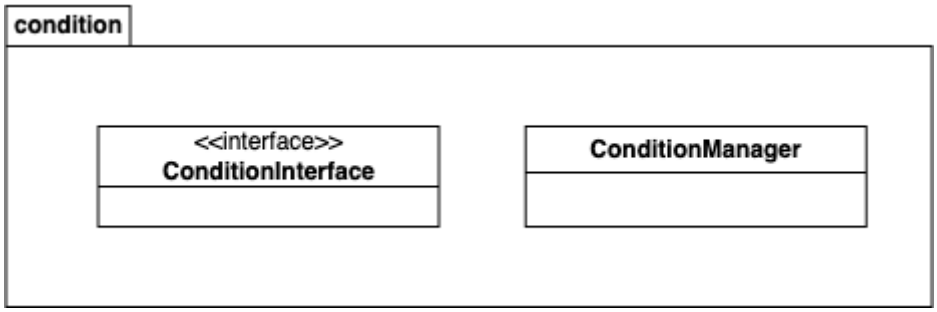
Package TalkAID

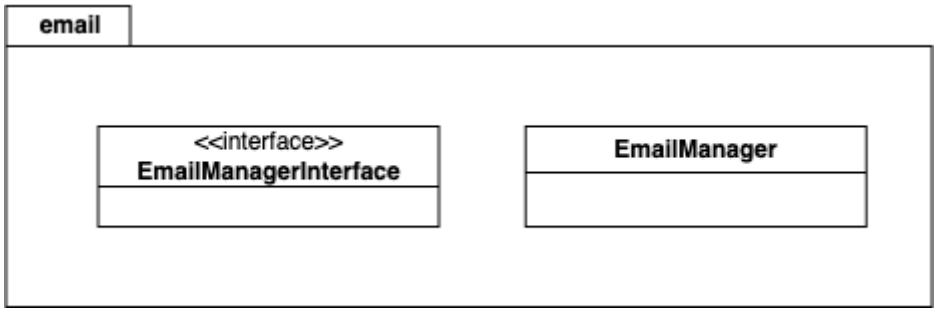
Nella sezione corrente, vengono illustrati i dettagli della struttura del package principale di TalkAID. Questa struttura è stata formulata attraverso quattro decisioni chiave:

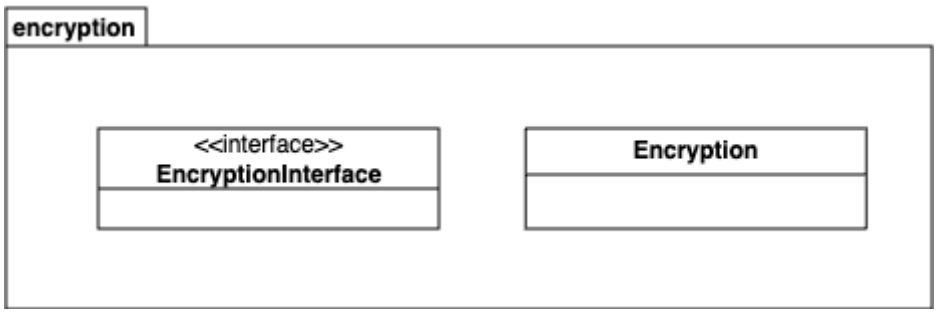
1. È stato deciso di creare nel package *src/main/java/**model**/* 3 ulteriori suddivisioni:
    a. */entity/* ovvero gli oggetti che conterranno i dati estratti dalle query con il database.
    b. */DAO/* per l'interazione con il database e la manipolazione di oggetti.
    c. */service/* contenente i servizi elargiti dal sistema.
2. È stato scelto di creare un package distinto per ogni sottosistema. Questi risiedono all'interno di */model/**service***. Tali "Servizi" forniscono delle classi con metodi intuitivi, modulari e leggibili predisposti per l'utilizzo all'interno delle servlet.
3. Le servlet risiedono in *src/main/java/**controller**/*. Non implementeranno logiche di business particolarmente complesse ma utilizzeranno ciò che è stato codificato negli strati inferiori, in particolare da *service*.
4. Alla luce di tali caratteristiche, all'interno della cartella ***service*** sono presenti i nomi dei sottosistemi identificati attraverso un package avente come sostantivo il servizio che forniscono. Esempio: *model/service/user* possiede dei servizi relativi all'utenza generici come la modifica dei dati personali.

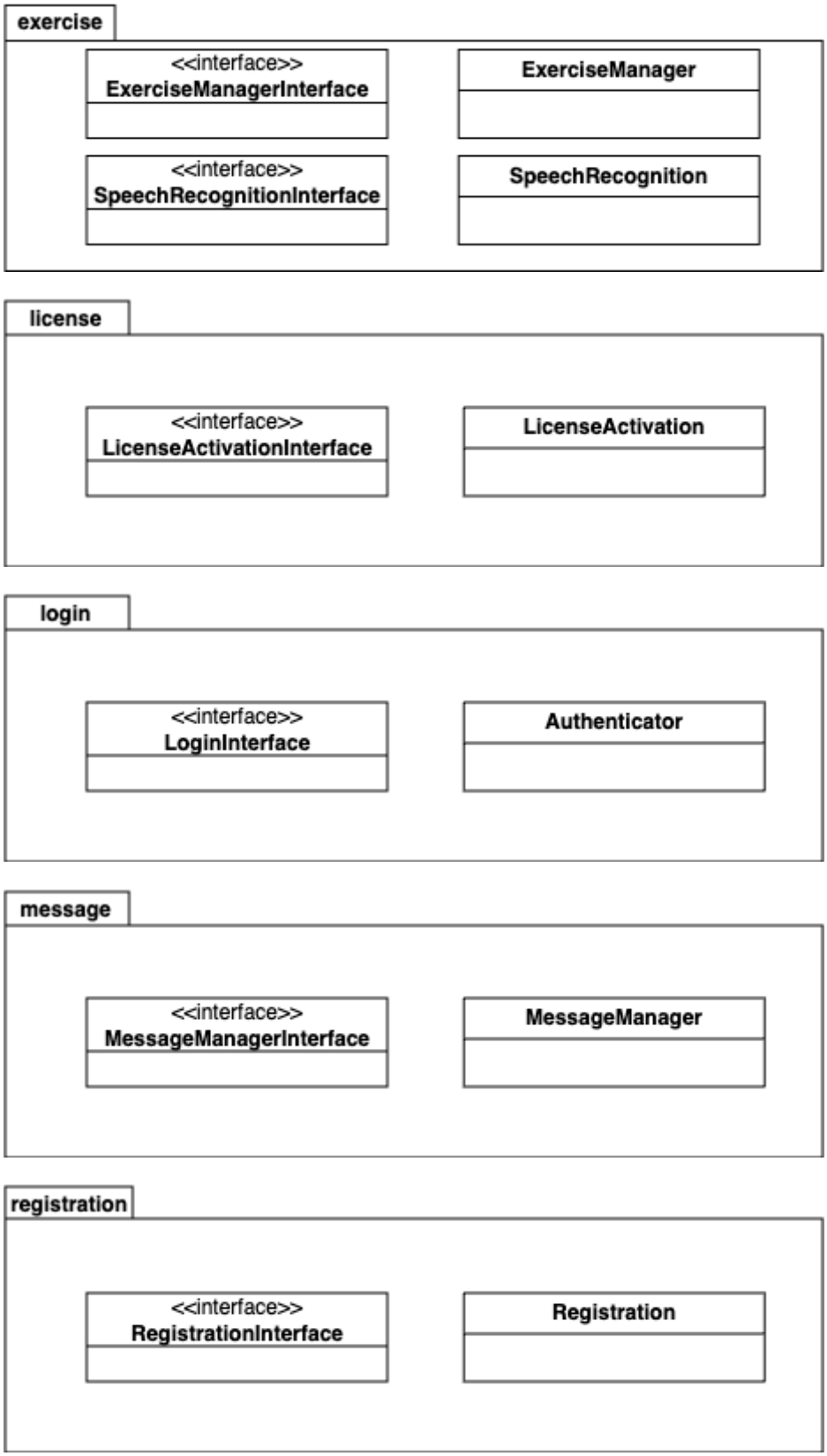
Per quanto riguarda le dipendenze tra i packages, attraverso la suddivisione precedentemente descritta, risulta più immediato comprendere quali siano le servlet od i DAO utilizzati od inerenti a degli specifici servizi.

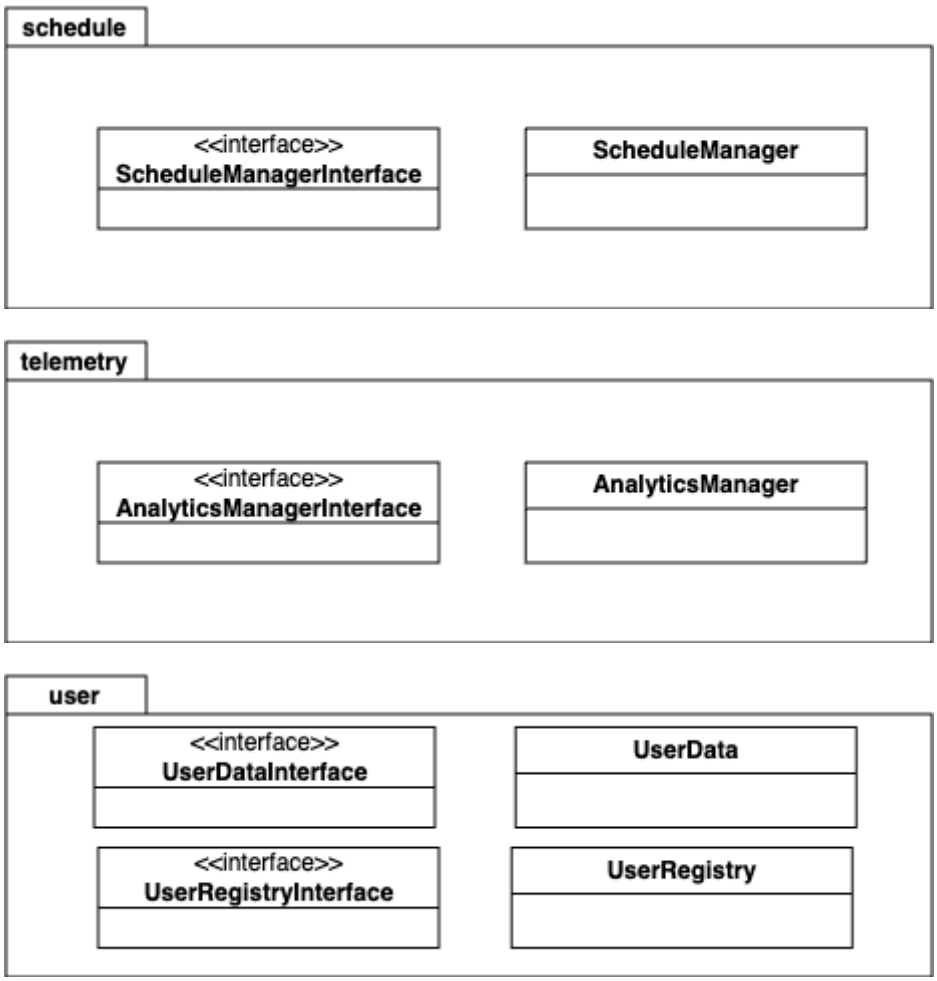

java
Controller
model
DAO
entity
service


condition
<<interface>>
ConditionInterface
ConditionManager


email
<<interface>>
EmailManagerInterface
EmailManager


encryption
<<interface>>
EncryptionInterface
Encryption


exercise
<<interface>>
ExerciseManagerInterface
ExerciseManager
<<interface>>
SpeechRecognitionInterface
SpeechRecognition
license
<<interface>>
LicenseActivationInterface
LicenseActivation
login
<<interface>>
LoginInterface
Authenticator
message
<<interface>>
MessageManagerInterface
MessageManager
registration
<<interface>>
RegistrationInterface
Registration



3. Class Interfaces

Di seguito sono riportate le classi implementate presenti nei sub-package di Service. Ciascuna classe è l'implementazione di un'Interfaccia. È stato scelto di riportare anche i rispettivi metodi privati, annotati con (-) prodotti durante la codifica per suddividere al meglio metodi che sarebbero risultati troppo grandi. I nomi dei package sono stati solo in questo documento scritti in CamelCase per una miglior visibilità

1. Package Condition

Nome Classe	Condition
Descrizione	Questa classe si occupa della gestione delle patologie dei pazienti.
Metodi	+ getConditionsOfPatient(int id_patient): ArrayList<Condition> + getConditionsNOTOfPatient(int id_patient): ArrayList<Condition> + addConditionPatient(int IdCondition, int IdPatient, int severity): boolean + removeConditionPatient(int IdCondition, int IdPatient): boolean
Invariante di Classe	/

Nome Metodo	**+ getConditionsOfPatient(int id_patient): ArrayList<Condition>**
Descrizione	Questo metodo restituisce tutte le patologie del paziente.
Pre-Condizione	**context** ConditionInterface::getConditionsOfPatient(id_patient : Integer) : ArrayList<Condition> **pre**: id_patient >= 0
Post-Condizione	post: result != null
Nome Metodo	**+ getConditionsNOTOfPatient(int id_patient): ArrayList<Condition>**
Descrizione	Questo metodo restituisce tutte le patologie non assegnate al paziente.
Pre-Condizione	**context** ConditionInterface::getConditionsNOTOfPatient(id_patient : Integer) : ArrayList<model.entity.Condition> pre: id_patient >= 0
Post-Condizione	**post**: result != null
Nome Metodo	**+ addConditionPatient(int IdCondition, int IdPatient, int severity): boolean**
	Questo metodo aggiunge una patologia ad un paziente.
Pre-Condizione	**context** ConditionInterface::AddConditionPatient(ID_condition : Integer, ID_patient : Integer, Severity : Integer) : Boolean **pre**: ID_condition >= 0 and ID_patient >= 0 and Severity >= 0
Post-Condizione	**post**: result = true or result = false
Nome Metodo	**+ removeConditionPatient(int IdCondition, int IdPatient): boolean**

Descrizione	Questo metodo rimuove una patologia ad un paziente.
Pre-Condizione	**context** ConditionInterface::RemoveConditionPatient(ID_condition : Integer, ID_patient : Integer) : Boolean **pre**: ID_condition >= 0 and ID_patient >= 0
Post-Condizione	**post**: result = true or result = false

2. Package Exercise

Nome Classe	**ExerciseManager**
Descrizione	Questa classe si occupa della gestione degli esercizi dei pazienti
Metodi	+ getExercise(int exerciseId): ExerciseGlossary + getExecution(int exerciseId, int userId, Date insertionDate): Blob + saveExecution(int userId, int exerciseId, Date insertDate, Blob execution): boolean + saveEvaluation(int userId, int exerciseId, Date insertDate, int evaluation): boolean + retrieveAllPatientExerciseDone(int userId): List<Exercise> + retrieveDoneExercises(int patientId): List<SlimmerExercise> + retrieveNotDoneExercises(int patientId): List<SlimmerExercise>
Invariante di Classe	/

Nome Metodo	**+ getExercise(int exerciseId): ExerciseGlossary**
Descrizione	Questo metodo restituisce la classe Esercizio dato l'ID.
Pre-Condizione	**context** ExerciseManager::getExercise(exerciseID : Integer) : ExerciseGlossary **pre**: exerciseID >= 0
Post-Condizione	**post**: result != null
Nome Metodo	**+ getExecution(int exerciseId, int userId, Date insertionDate): Blob**
Descrizione	Questo metodo restituisce l'oggetto Blob dell'esecuzione di uno specifico esercizio.
Pre-Condizione	**context** ExerciseManager::getExecution(exerciseID : Integer, userID : Integer, insertionDate : Date) : Blob **pre**: exerciseID >= 0 and userID >= 0 and insertionDate != null
Post-Condizione	**post**: result != null
Nome Metodo	**+ saveExecution(int userId, int exerciseId, Date insertDate, Blob execution): boolean**

	Questo metodo salva l'esecuzione dell'esercizio.
Pre-Condizione	**context** ExerciseManager::saveExecution(userID : Integer, exerciseId : Integer, insertDate : Date, execution : Blob) : Boolean **pre**: userID >= 0 and exerciseId >= 0 and insertDate != null and execution != null
Post-Condizione	**post**: result = true or result = false
Nome Metodo	**+ saveEvaluation(int userId, int exerciseId, Date insertDate, int evaluation): boolean**
Descrizione	Questo metodo salva la valutazione associata all'esercizio.
Pre-Condizione	**context** ExerciseManager::saveEvaluation(userID : Integer, exerciseId : Integer, insertDate : Date, evaluation : Integer) : Boolean **pre**: userID >= 0 and exerciseId >= 0 and insertDate != null and evaluation >= 0
Post-Condizione	**post**: result = true or result = false
Nome Metodo	**+ retrieveAllPatientExerciseDone(int userId): List<Exercise>**
Descrizione	Questo metodo restituisce tutti gli esercizi svolti dal paziente.
Pre-Condizione	**context** ExerciseManager::retrieveAllPatientExerciseDone(userID : Integer) : List<Exercise> **pre**: userID >= 0
Post-Condizione	**post**: result != null
Nome Metodo	**+ retrieveDoneExercises(int patientId): List<SlimmerExercise>**
Descrizione	Questo metodo restituisce tutti gli esercizi svolti dal paziente in un formato più leggero.
Pre-Condizione	**context** ExerciseManager::retrieveDoneExercises(patientId : Integer) : List<SlimmerExercise> **pre**: patientId >= 0
Post-Condizione	**post**: result != null
Nome Metodo	**+ retrieveNotDoneExercises(int patientId): List<SlimmerExercise>**
Descrizione	Questo metodo restituisce tutti gli esercizi non svolti dal paziente in un formato più leggero
Pre-Condizione	context ExerciseManager::retrieveNotDoneExercises(patientId : Integer) : List<SlimmerExercise> pre: patientId >= 0
Post-Condizione	post: result != null

Nome Classe	SpeechRecognition
Descrizione	Questa classe si occupa della gestione degli esercizi dei pazienti
Metodi	+ azureSTT(InputStream audio): String + generateFile(InputStream inputAudio): String
Invariante di Classe	/

Nome Metodo	+ azureSTT(InputStream audio): String
Descrizione	Questo metodo restituisce il testo dalla fonte audio.
Pre-Condizione	**context** SpeechRecognition::azureSTT(audio : InputStream) : String **pre**: audio != null
Post-Condizione	**post**: result != null or result = null
Nome Metodo	+ generateFile(InputStream inputAudio): String
Descrizione	Questo metodo restituisce il path dove è presente il file audio convertito e generato.
Pre-Condizione	**context** SpeechRecognition::generateFile(inputAudio : InputStream) : String **pre**: inputAudio != null
Post-Condizione	**post**: result != null and result != ""

3. Package Telemetry

Nome Classe	TelemetryManager
Descrizione	Questa classe si occupa di fornire la funzionalità della telemetria.
Metodi	+ storeAnalytics(int userId, String type, String Description)
Invariante di Classe	/

Nome Metodo	+ storeAnalytics(int userId, String type, String Description)
Descrizione	Questo metodo, salva, se possibile e se l'utente lo ha concesso, i dati di telemetria.
Pre-Condizione	**context** storeAnalytics(userId : Integer, type : String, description : String) **pre**: userId >= 0 and type != null and type != "" and description != null and description != ""
Post-Condizione	/

4. Package Schedule

Nome Classe	ScheduleManager
Descrizione	Questa classe si occupa della gestione delle prenotazioni e dell'agenda per le visite mediche.
Metodi	+ createNewSchedule(int idTherapist, String date, String timeSlot) + modifySchedule(int idTherapist, String date, String timeSlot, String nDate, string ntimeSlot, int reserved) + deleteSchedule(int idTherapist, String date, String timeSlot) + retrieveAllPatientSchedules(int reserved): List<Schedule> + retrieveAllTherapistSchedules(int idTherapist): List<Schedule> + retrieveAllPrenotedSchedules(int idTherapist): List<Schedule> + retrieveAllNotPrenotedSchedules(Int idTherapist): List<Schedule> + checkDate(int idTherapist, String date, String timeSlot): boolean
Invariante di Classe	/

Nome Metodo	+createNewSchedule(idTherapist : Integer, date : String, timeSlot : String)
Descrizione	Questo metodo crea un nuovo schedule per un terapista. La precondizione garantisce che l'ID del terapista, la data e l'intervallo di tempo immessi siano validi. La post-condizione indica che lo stato del sistema non cambia come risultato di questa operazione.
Pre-Condizione	**context** ScheduleManager::createNewSchedule(idTherapist : Integer, date : String, timeSlot : String) **pre**: idTherapist >= 0 and date != null and date != "" and timeSlot != null and timeSlot != ""
Post-Condizione	**post**: no change
Nome Metodo	**+modifySchedule(idTherapist : Integer, date : String, timeSlot : String, ndate : String, ntimeSlot : String, reserved : Integer)**
Descrizione	Questo metodo modifica uno schedule per un terapista. La precondizione garantisce che l'ID del terapista immesso, la data, la fascia oraria, la nuova

	data, la nuova fascia oraria e lo stato riservato siano validi. La post-condizione indica che lo stato del sistema non cambia come risultato di questa operazione.
Pre-Condizione	**context** ScheduleManager::modifySchedule(idTherapist : Integer, date : String, timeSlot : String, ndate : String, ntimeSlot : String, reserved : Integer) **pre**: idTherapist >= 0 and date != null and date != '''' and timeSlot != null and timeSlot != '''' and ndate != null and ndate != '''' and ntimeSlot != null and ntimeSlot != '''' and reserved >= 0
Post-Condizione	**post**: no change

Nome Metodo	**+ScheduleManager::deleteSchedule(idTherapist : Integer, date : String, timeSlot : String)**
Descrizione	Questo metodo elimina uno schedule per un terapista. La precondizione garantisce che l'ID del terapista, la data e l'intervallo di tempo immessi siano validi. La post-condizione indica che lo stato del sistema non cambia come risultato di questa operazione.
Pre-Condizione	**context** ScheduleManager::deleteSchedule(idTherapist : Integer, date : String, timeSlot : String) **pre:** idTherapist >= 0 and date != null and date != '''' and timeSlot != null and timeSlot != ''''
Post-Condizione	**post**: no change
Nome Metodo	**+ScheduleManager::retrieveAllPatientSchedules(reserved : Integer) : List<Schedule>**
Descrizione	Questo metodo recupera tutte le pianificazioni per un paziente. La precondizione garantisce che lo stato riservato dell'input sia valido e la postcondizione garantisce che il metodo restituisca un elenco di pianificazioni.
Pre-Condizione	**context** ScheduleManager::retrieveAllPatientSchedules(reserved : Integer) : List<Schedule> **pre**: reserved >= 0
Post-Condizione	`post: result != null`
Nome Metodo	**+ScheduleManager::retrieveAllTherapistSchedules(idTherapist : Integer) : List<Schedule>**
Descrizione	Questo metodo recupera tutti gli orari di un terapista. La precondizione garantisce che l'ID del terapista immesso sia

	valido e la postcondizione garantisce che il metodo restituisca un elenco di pianificazioni.
Pre-Condizione	**context** ScheduleManager::retrieveAllTherapistSchedules(idTherapist : Integer) : List<Schedule> **pre**: idTherapist >= 0
Post-Condizione	**post**: no change
Nome Metodo	**+ScheduleManager::retrieveAllPrenotedSchedules(idTherapist : Integer) : List<Schedule>**
Descrizione	Questo metodo recupera tutti gli orari preannotati per un terapista. La precondizione garantisce che l'ID del terapista immesso sia valido e la postcondizione garantisce che il metodo restituisca un elenco di pianificazioni.
Pre-Condizione	**context** ScheduleManager::retrieveAllPrenotedSchedules(idTherapist : Integer) : List<Schedule> **pre**: idTherapist >= 0
Post-Condizione	`post: result != null`
Nome Metodo	**+ScheduleManager::retrieveAllNotPrenotedSchedules(idTherapist : Integer) : List<Schedule>**
Descrizione	Questo metodo recupera tutti gli orari non preannotati per un terapista. La precondizione garantisce che l'ID del terapista immesso sia valido e la postcondizione garantisce che il metodo restituisca un elenco di pianificazioni.
Pre-Condizione	**context** ScheduleManager::retrieveAllNotPrenotedSchedules(idTherapist : Integer) : List<Schedule> **pre**: idTherapist >= 0
Post-Condizione	**post**: result != null
Nome Metodo	**+ScheduleManager::checkData(idTherapist : Integer, date : String, timeSlot : String) : Boolean**
Descrizione	Questo metodo recupera il conteggio di tutti gli orari preannotati per un terapista. La precondizione garantisce che l'ID del terapista immesso sia valido e la postcondizione garantisce che il metodo restituisca un numero intero che rappresenta il conteggio delle pianificazioni preannotate.
Pre-Condizione	**context** ScheduleManager::checkData(idTherapist : Integer, date : String, timeSlot : String) : Boolean **pre**: idTherapist >= 0 and date != null and date != "" and timeSlot != null and timeSlot != ""
Post-Condizione	`post: result = true or result = false`

5. Package Encryption

Nome Classe	Encryption
Descrizione	Questa classe si occupa dell'hashing e confronto delle password.
Metodi	+ encryptPassword(String plainTextPassword): String + verifyPassword(String plainTextPassword, String hashedPassword): Boolean
Invariante di Classe	/

Nome Metodo	**+ encryptPassword(String plainTextPassword): String**
Descrizione	Questo metodo, data una password in chiaro restituisce la password criptata con BCrypt.
Pre-Condizione	**context** Encryption::encryptPassword(plainTextPassword : String) : String **pre**: plainTextPassword != null and plainTextPassword != ""
Post-Condizione	**post**: result != null and result != ""
Nome Metodo	**+ verifyPassword(String plainTextPassword, String hashedPassword): Boolean**
Descrizione	Questo metodo restituisce se la password corrisponde o meno a quella criptata con BCrypt.
Pre-Condizione	**context** Encryption::verifyPassword(plainTextPassword : String, hashedPassword : String) : Boolean **pre**: plainTextPassword != null and plainTextPassword != "" and hashedPassword != null and hashedPassword != ""
Post-Condizione	**post**: result = true or result = false

6. Package License

Nome Classe	LicenseActivation
Descrizione	Questa classe si occupa dell'attivazione e generazione delle licenze
Metodi	+ getLicense(String code): License + isActivable(License license): Boolean + isForTherapist(License license): int + activate(License license, int userId) + generatePin(int therapistId) + generateLicense()
Invariante di Classe	/

Nome Metodo	+ getLicense(String code): License
Descrizione	Questo metodo restituisce la licenza, se presente, nel database
Pre-Condizione	**context** LicenseActivation::getLicense(code : String) : License **pre**: code != null and code != ""
Post-Condizione	**post**: result != null
Nome Metodo	+ isActivable(License license): Boolean
Descrizione	Questo metodo restituisce *true* se la licenza è attivabile, *false* altrimenti.
Pre-Condizione	**context** LicenseActivation::isActivable(license : License) : Boolean **pre**: license != null
Post-Condizione	**post**: result = true or result = false
Nome Metodo	+ isForTherapist(License license): int
Descrizione	Questo metodo restituisce 0 se è una licenza per il terapeuta, altrimenti restituisce l'ID del terapeuta che l'ha generata
Pre-Condizione	**context** LicenseActivation::isForTherapist(license : License) : Integer **pre**: license != null
Post-Condizione	**post**: result >= 0
Nome Metodo	+ activate(License license, int userId)
Descrizione	Questo metodo attiva la licenza con l'ID dell'utente inserito
Pre-Condizione	**context** LicenseActivation::activate(license : License, userId : Integer) **pre**: license != null and userId >= 0
Post-Condizione	/
Nome Metodo	+ generatePin(int therapistId)

Descrizione	Questo metodo genera un PIN di invito per un paziente che se utilizzato per la registrazione, lo associerà come paziente del logopedista che lo ha generato
Pre-Condizione	**context** LicenseActivation::generatePin(therapistId : Integer) : String **pre**: therapistId >= 0
Post-Condizione	**post**: result != null and result != ""
Nome Metodo	**+ generateLicense()**
Descrizione	Questo metodo genera una Licenza per un nuovo logopedista
Pre-Condizione	**context** LicenseActivation::generateLicense() : String
Post-Condizione	**post**: result != null and result != ""

7. Package Login

Nome Classe	**Authenticator**
Descrizione	Questa classe provvede ad autenticare gli utenti del sistema
Metodi	+ authenticate(String email, String password): int + resetPassword(String email, String plainTextPassword): Boolean + sendPin(String email): String - generatePin(): String
Invariante di Classe	/

Nome Metodo	**+ authenticate (String email, String password): int**
Descrizione	Questo metodo restituisce l'ID utente se le credenziali sono corrette, altrimenti fornisce un codice di errore
Pre-Condizione	**context** Authenticator::authenticate(email : String, password : String) : Integer **pre**: email != null and email != "" and password != null and password != ""
Post-Condizione	**post**: result >= -1
Nome Metodo	**+ resetPassword(String email, String plainTextPassword): boolean**
Descrizione	Questo metodo effettua il cambio della password con la nuova sottomessa.
Pre-Condizione	**context** Authenticator::resetPassword(email : String, plainTextPassword : String) : Boolean **pre**: email != null and email != "" and plainTextPassword != null and plainTextPassword != ""
Post-Condizione	**post**: result = true or result = false
Nome Metodo	**+ sendPin(String email): String**

	Questo metodo invia il pin generato alla casella email dell'utente per procedere al cambio della password.
Pre-Condizione	**context** Authenticator::sendPin(email : String) : String **pre**: email != null and email != ""
Post-Condizione	**post**: result != null and result != ""
Nome Metodo	**- generatePin(): String**
Descrizione	Questo metodo genera un pin casuale da 8 cifre
Pre-Condizione	**context** Authenticator::generatePin() : String
Post-Condizione	**post**: result != null and result != ""

8. Package Message

Nome Classe	**MessageManager**
Descrizione	Questa classe si occupa della messaggistica del sistema
Metodi	+ markMessageAsRead(int sender, int recipientId) + countReceivedMessages(int recipientId): int + sendMessage(int sender, int recipientId, String text) + retrieveMessage(int userId, int contact): List<Message> + getUnreadMessagesForConversation(int userId, int contact): int + retrieveAllTheContacts(int userId): List<Integer>
Invariante di Classe	/

Nome Metodo	**+ markMessageAsRead(int senderId, int recipientId)**
Descrizione	Questo metodo segna come letto i messaggi ricevuti in una conversazione.
Pre-Condizione	**context** MessageManager::markMessagesAsRead(senderId : Integer, recipientId : Integer) **pre**: senderId >= 0 and recipientId >= 0
Post-Condizione	/
Nome Metodo	**+ countReceivedMessages(int recipientId): int**
Descrizione	Questo metodo conta il numero totale di messaggi ricevuti.
Pre-Condizione	**context** MessageManager::countReceivedMessages(recipientId : Integer) : Integer **pre**: recipientId >= 0
Post-Condizione	**post**: result >= 0
Nome Metodo	**+ sendMessage(int sender, int recipientId, String text)**
	Questo metodo invia un messaggio conoscendone le informazioni essenziali.

Pre-Condizione	**context** MessageManager::sendMessage(sender : Integer, recipientId : Integer, text : String) **pre**: sender >= 0 and recipientId >= 0 and text != null and text != ""
Post-Condizione	/
Nome Metodo	**+ retrieveMessages(int userId, int contact): List<Message>**
Descrizione	Questo metodo restituisce una lista in ordine cronologico dei messaggi scambiati tra 2 contatti
Pre-Condizione	**context** MessageManager::retrieveMessages(userId : Integer, contact : Integer) : List<Message> **pre**: userId >= 0 and contact >= 0
Post-Condizione	**post:** result != null
Nome Metodo	**+ getUnreadMessagesForConversation(int userId, int contact): int**
Descrizione	Questo metodo restituisce il numero di messaggi non letti da parte dell'userId su un'unica conversazione.
Pre-Condizione	**context** MessageManager::getUnreadMessagesForConversation(userId : Integer, contact : Integer) : Integer **pre**: userId >= 0 and contact >= 0
Post-Condizione	**post:** result >= 0
Nome Metodo	**+ retrieveAllTheContacts(int userId): List<Integer>**
Descrizione	Questo metodo restituisce una lista di tutti i contatti con cui un utente può comunicare.
Pre-Condizione	**context** MessageManager::retrieveAllTheContacts(userId : Integer) : List<Integer> **pre**: userId >= 0
Post-Condizione	**post:** result != null

9. Package Registration

Nome Classe	Registration
Descrizione	Questa classe, utilizzando metodi degli altri sottosistemi si occupa di verificare i dati per il primo accesso degli utenti al sistema.
Metodi	+ registerNewUser(String licenseCode, String email, String password, String name, String surname): int - validateLicense(String licenseCode): License - isEmailExists(String email): boolean - encryptPassword(String password): String - createNewUser(String email, String hashed, License license): int

	- createUserPersonalInformation(int theNewId, String name, String surname): boolean
Invariante di Classe	/

Nome Metodo	**+ registerNewUser(String licenseCode, String email, String password, String name, String surname): int**
Descrizione	Questo metodo registra un nuovo utente al sistema. Restituisce un codice di errore, 0 se non ce ne sono.
Pre-Condizione	**context** Registration::registerNewUser(licenseCode : String, email : String, password : String, name : String, surname : String) : Integer **pre**: licenseCode != null and licenseCode != "" and email != null and email != "" and password != null and password != "" and name != null and name != "" and surname != null and surname != ""
Post-Condizione	**post**: result >= 0
Nome Metodo	**- validateLicense(String licenseCode): License**
Descrizione	Restituisce la licenza, se questa è attivabile
Pre-Condizione	**context** Registration::validateLicense(licenseCode : String) : License **pre**: licenseCode != null and licenseCode != ""
Post-Condizione	**post**: result != null or result = null
Nome Metodo	**- isEmailExists(String email): boolean**
Descrizione	Verifica che questa email sia utilizzata nel sistema.
Pre-Condizione	**context** Registration::isEmailExists(email : String) : Boolean **pre**: email != null and email != ""
Post-Condizione	**post**: result = true or result = false
Nome Metodo	**- encryptPassword(String password): String**
Descrizione	Questo metodo restituisce la password criptata
Pre-Condizione	**context** Registration::encryptPassword(password : String) : String **pre**: password != null and password != ""
Post-Condizione	**post:** result != null and result != ""
Nome Metodo	**- createNewUser(String email, String hashed, License license): int**
Descrizione	Questo metodo restituisce l'ID del nuovo utente generato con le informazioni sottomesse.
Pre-Condizione	**context** Registration::createNewUser(email : String, hashed : String, license : License) : Integer **pre**: email != null and email != "" and hashed != null and hashed != "" and license != null
Post-Condizione	**post:** result >= 0

Nome Metodo	**- createUserPersonalInformation(int theNewId, String name, String surname): boolean**
Descrizione	Questo metodo restituisce True se la scrittura è terminata con successo nel database.
Pre-Condizione	**context** Registration::createUserPersonalInformation(theNewId : Integer, name : String, surname : String) : Boolean **pre**: theNewId >= 0 and name != null and name != "" and surname != null and surname != ""
Post-Condizione	**post:** result = true or result = false

10. Package User

Nome Classe	UserData
Descrizione	Questa classe si occupa di gestire le informazioni presenti nella tabella Utente presente nel database
Metodi	+ checkIfEmailExists(String email): boolean + createUser(String email, String password, int therapistId): int + getUser(Object idOrEmail): User + isTherapist(User user): boolean + updateAnalyticsPreference(String id, Boolean value): boolean + updateEmailTime(String id, String value): boolean
Invariante di Classe	/

Nome Metodo	**+ checkIfEmailExists(String email): boolean**
Descrizione	Controlla se l'email è in uso o meno
Pre-Condizione	**context** UserData::checkIfEmailExists(email : String) : Boolean **pre**: email != null and email != ""
Post-Condizione	**post**: result = true or result = false
Nome Metodo	**+ createUser(String email, String password, int therapistId): int**
Descrizione	Genera l'utente con le informazioni essenziali
Pre-Condizione	**context** UserData::createUser(email : String, password : String, therapistId : Integer) : Integer **pre**: email != null and email != "" and password != null and password != "" and therapistId >= 0
Post-Condizione	**post**: result >= 0
Nome Metodo	**+ getUser(Object idOrEmail): User**
Descrizione	Restituisce l'utente, se presente, dal database che possiede o l'email o l'ID richiesto.

Pre-Condizione	**context** UserData::getUser(idOrEmail : Object) : User **pre**: idOrEmail != null
Post-Condizione	**post**: result != null or result == null
Nome Metodo	**+ isTherapist(User user): boolean**
Descrizione	Determina o meno se un utente è un terapista.
Pre-Condizione	**context** UserData::isTherapist(user : User) : Boolean **pre**: user != null
Post-Condizione	**post**: result = true or result = false
Nome Metodo	**+ updateAnalyticsPreference(String id, Boolean value): boolean**
Descrizione	Modifica il campo col nuovo valore.
Pre-Condizione	**context** UserData::updateAnalyticsPreference(id : String, value : Boolean) : Boolean **pre**: id != null and id != "" and value != null
Post-Condizione	**post**: result = true or result = false
Nome Metodo	**+ updateEmailTime(String id, String value): boolean**
Descrizione	Modifica il campo col nuovo valore.
Pre-Condizione	**context** UserData::updateEmailTime(id : String, value : String) : Boolean **pre**: id != null and id != "" and value != null and value != ""
Post-Condizione	**post**: result = true or result = false

Nome Classe	**UserRegistry**
Descrizione	Questa classe si occupa di gestire le informazioni presenti nella tabella PersonalInfo presente nel database
Metodi	+ firstAccess(int id, String name, String surname): boolean + getPersonalInfo(int id) : PersonalInfo + updatePersonalInfo(int id, String firstName, String lastName, String phone): boolean
Invariante di Classe	/

Nome Metodo	**+ firstAccess(int id, String name, String surname):**
Descrizione	Inserisce i dati essenziali dell'utente nel database
Pre-Condizione	**context** UserRegistry::firstAccess(id : Integer, name : String, surname : String) : Boolean **pre**: id >= 0 and name != null and name != "" and surname != null and surname != ""
Post-Condizione	**post**: result = true or result = false
Nome Metodo	**+ getPersonalInfo(int id) : PersonalInfo**
Descrizione	Restituisce, se presenti, i dati personali dell'utente

Pre-Condizione	**context** UserRegistry::getPersonalInfo(id : Integer) : PersonalInfo **pre**: id >= 0
Post-Condizione	**post**: result != null
Nome Metodo	**+ updatePersonalInfo(int id, String firstName, String lastName, String phone): boolean**
Descrizione	Aggiorna i dati personali dell'utente
Pre-Condizione	**context** UserRegistry::updatePersonalInfo(id : Integer, FirstName : String, LastName : String, Phone : String) : Boolean **pre**: id >= 0 and FirstName != null and FirstName != "" and LastName != null and LastName != "" and Phone != null and Phone != ""
Post-Condizione	**post**: result = true or result = false

11. Package Email

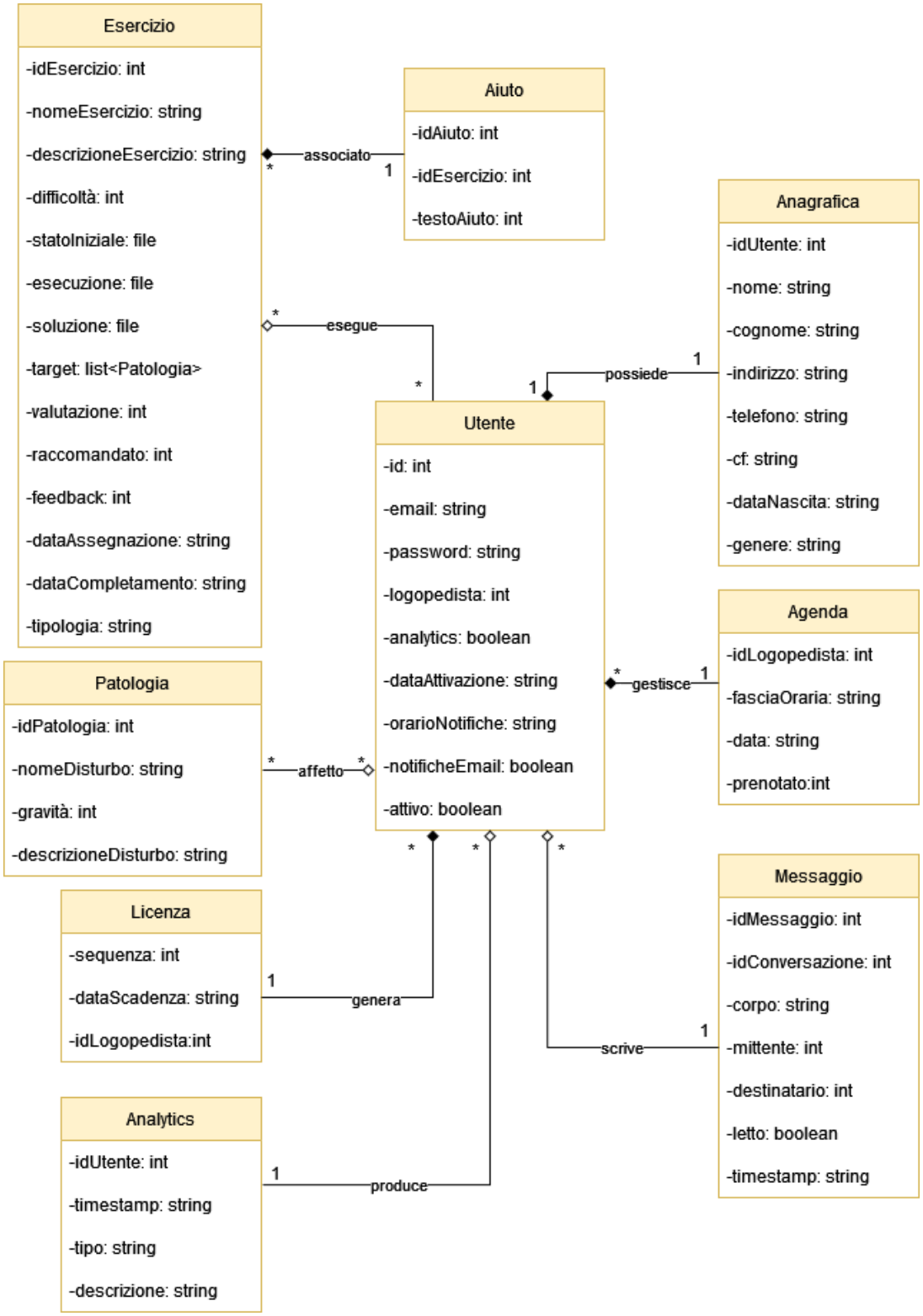
Nome Classe	**EmailManager**
Descrizione	Questa classe permette l'invio delle email.
Metodi	+ sendEmail(String toAddress, String subject, String body) - getSession(): Session - sendEmail(Session session, MimeMessage message) - generateMessage(Mimemessage message, String toAddress, String subject, String body) - loadEmailProperties()
Invariante di Classe	/

Nome Metodo	**+ sendEmail(String toAddress, String subject, String body)**
Descrizione	Invia l'email
Pre-Condizione	**context** EmailManager::sendEmail(toAddress : String, subject : String, body : String) **pre**: toAddress != null and toAddress != "" and subject != null and subject != "" and body != null and body != ""
Post-Condizione	/
Nome Metodo	**- getSession(): Session**
Descrizione	Genera la sessione, successiva all'autenticazione per l'invio dell'email.
Pre-Condizione	**context** EmailManager::getSession() : Session
Post-Condizione	**post**: result != null
Nome Metodo	**- sendEmail(Session session, MimeMessage message)**

Descrizione	Invia l'email formattata
Pre-Condizione	**context** EmailManager::sendEmail(session : Session, message : MimeMessage) **pre**: session != null and message != null
Post-Condizione	/
Nome Metodo	**- generateMessage(Mimemessage message, String toAddress, String subject, String body)**
Descrizione	Genera il messaggio da inviare in maniera formattata per gli altri metodi.
Pre-Condizione	**context** EmailManager::generateMessage(message : MimeMessage, toAddress : String, subject : String, body : String) **pre**: message != null and toAddress != null and toAddress != "" and subject != null and subject != "" and body != null and body != ""
Post-Condizione	/
Nome Metodo	**- loadEmailProperties()**
Descrizione	Carica le informazioni per l'autenticazione al servizio di email.
Pre-Condizione	**context** EmailManager::loadEmailProperties() : Properties
Post-Condizione	/

4. Class Diagram e Design Patterns

Vieni riportato di seguito il Class Diagram ristrutturato:



Design Patterns

In questa sezione sono presentati i design patterns applicati nello sviluppo del sistema TalkAID. Per ogni pattern è presente:

- Introduzione al design pattern;
- Il problema che ha causato la scelta di adottare tale design pattern;
- Spiegazione di com'è stato possibile risolverlo grazie al design pattern;
- Struttura delle classi che implementano il design pattern.

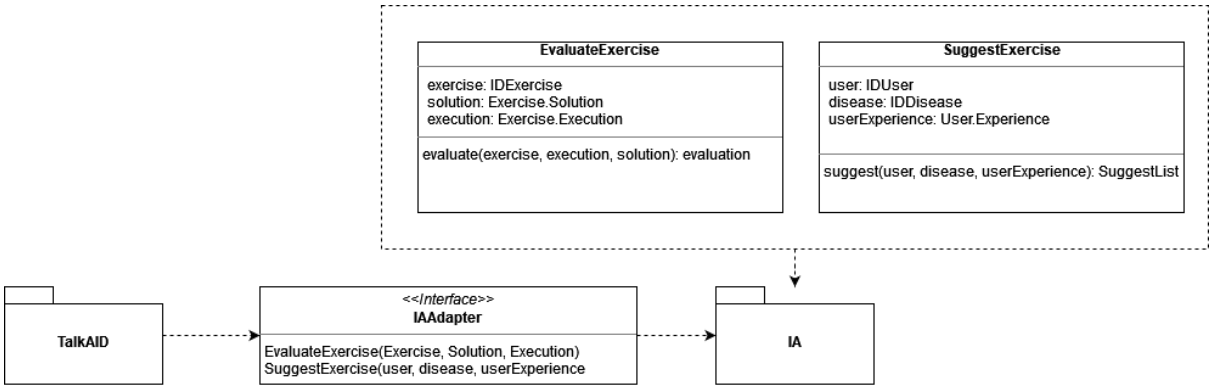
Adapter

Un Adapter è un pattern Strutturale che permette di convertire l'interfaccia di una classe in una interfaccia diversa, in maniera tale che classi diverse possano operare insieme nonostante abbiano interfacce incompatibili. In altre parole, l'Adapter crea un livello che permette la comunicazione tra due interfacce differenti.

Uno dei punti di forza del sistema TalkAID è la componente IA, capace di raccomandare e valutare gli esercizi dei nostri utenti. Poiché il sistema è pensato come webapp ed è implementata principalmente con Java, HTML e CSS, contrariamente, la componente IA prevede un'implementazione in Python3. Ovviamente la comunicazione tra linguaggi di programmazione diversi non è possibile di base.

Per risolvere dunque tale problema, si è optato di utilizzare questo design pattern affinché faccia da tramite tra il sistema di IA ed il resto del sistema di TalkAID, permettendo quindi la comunicazione tra questi due sistemi indipendenti e normalmente incompatibili.

Di seguito è rappresentata la struttura della soluzione:



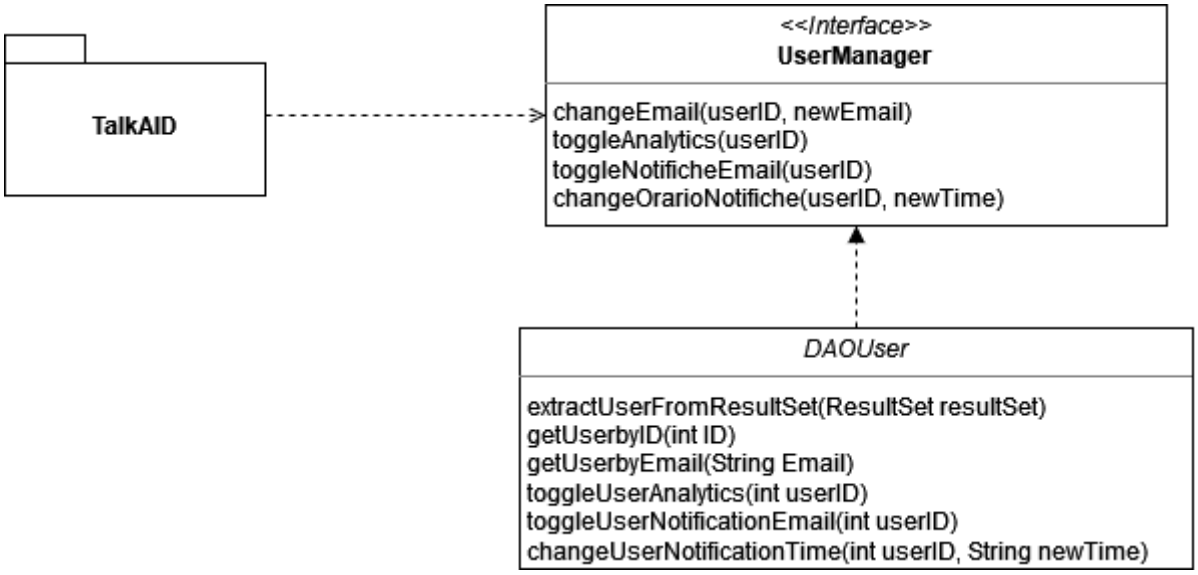
Façade

Un Façade è un pattern Strutturale che fornisce un'unica interfaccia semplificata per accedere ad un insieme di oggetti che compongono un sottosistema. In altre parole, si sovrappone ad un'interfaccia complessa e la rimpiazza con una semplice ed intuitiva.

Il sistema TalkAID si propone di avere un codice estremamente leggibile ed intuitivo, ma non sempre questo è possibile. Difatti, molte delle operazioni che incorrono alla necessità di dover manipolare dati persistenti, sono spesso composte da funzioni lunghe e complesse, che richiedono vari parametri e l'utilizzo di più funzioni.

Per rendere il codice meno verboso si è optato di utilizzare questo pattern, per creare delle interfacce semplici ed intuitive, con poche linee di codice autoesplicative, che rendono quindi il codice più manutenibile e chiaro.

Di seguito è rappresentato un esempio di utilizzo:



5. Glossario

Termine	Significato
Off-the-Shelf	Hardware o Software preconfezionati e pronti all'uso, disponibili sul mercato senza la necessità di personalizzazioni specifiche.
Package	Un raggruppamento di classi, interfacce o file correlati.
Design Pattern	Template di soluzioni a problemi ricorrenti.
Javadoc	Applicativo usato per la generazione automatica della documentazione del codice sorgente.
JDK	Java Development Kit, insieme di strumenti per sviluppare programmi Java.
Servlet	È un componente del lato server in Java che estende le funzionalità di un server web per generare dinamicamente i contenuti.
DAO	Data Access Object, pattern che si occupa di fornire un accesso in modo astratto ai dati persistenti presenti su un database.
Façade	Pattern che si occupa di fornire un'interfaccia semplice a sottosistemi complessi.
Adapter	Pattern che si occupa di permettere la comunicazione tra due interfacce non compatibili.
MVC	Model-View-Controller, pattern che permette la separazione logica tra presentazione e logica di business.
CamelCase	Pratica usata nella scrittura di parole composte lasciando intenzionalmente le iniziali delle parole in maiuscolo per migliorare la leggibilità.