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

Object Design Document

ReStart

Riferimento	2023_C08_ODD_V.1.0
Versione	1.0
Data	12/12/2023
Destinatario	Prof.ssa Filomena Ferrucci, Prof.re Fabio Palomba
Presentato da	C08 Gianfranco Barba, Francesco Corcione, Giuseppe Di Palma, Luigi Guida, Tullio Mansi, Matteo Panza, Serena Passiflora
Approvato da	/

Sommario

Revision History

Data	Versione	Descrizione	Autori
13/12/2023	0.1	Stesura Object Design Goals	Gianfranco Barba Tullio Mansi
13/12/2023	0.2	Stesura Object Design Trade-Off	Matteo Panza Giuseppe Di Palma
13/12/2023	0.3	Stesura Off-The-Shelf	Serena Passiflora Luigi Guida
13/12/2023	0.4	Stesura Linee Guida	Francesco Corcione
15/12/2023	0.5	Stesura Desgin Patterns	Tutti i TM
15/12/2023	0.6	Stesura packages	Tutti i TM
19/12/2023	0.7	Stesura class interfaces	Gianfranco Barba Tullio Masi Serena Passiflora
20/12/2023	0.8	Stesura class diagram	Giuseppe Di Palma Luigi Guida Matteo Panza
21/12/2023	0.9	Revisione stile	Gianfranco Barba Francesco Corcione Serena Passiflora
21/12/2023	0.10	Grammar Revision	Luigi Guida Tullio Mansi

09/01/2024	0.11	Modifica Class Interfaces e Class Diagram	Tutti i TM
20/01/2024	1.0	Revisione e Consegna	Tutti i TM

Team Members

Ruolo	Nome e Cognome	Acronimo	Email
PM	Rebecca Di Matteo	RDM	r.dimatteo10@studenti.unisa.it
PM	Leonardo Monaco	LM	l.monaco11@studenti.unisa.it
TM	Gianfranco Barba	GB	g.barba14@studenti.unisa.it
TM	Francesco Corcione	FC	f.corcione5@studenti.unisa.it
TM	Giuseppe Di Palma	GDP	g.dipalma23@studenti.unisa.it
TM	Luigi Guida	LG	l.guida15@studenti.unisa.it
TM	Tullio Mansi	TM	t.mansi@studenti.unisa.it
TM	Matteo Panza	MP	m.panza13@studenti.unisa.it
TM	Serena Passiflora	SP	s.passiflora@studenti.unisa.it

1. Introduzione

1.1 Object Design Goals

Rank	ID Design Goal	Descrizione	Origine
1	ODG_1 Robustezza	Il sistema deve dimostrarsi robusto e, in situazioni di emergenza, è fondamentale che sia in grado di rispondere efficacemente alle criticità. Tale affidabilità sarà assicurata attraverso la gestione del 90% degli errori mediante l'implementazione di opportune eccezioni.	DG_6
2	ODG_2 Sicurezza	Il sistema dovrà garantire un elevato livello di sicurezza attraverso la comunicazione su protocollo HTTPS, la corretta gestione dell'autenticazione e delle autorizzazioni.	DG_3
3	ODG_3 Manutenibilità	Il sistema deve presentare un alto livello di chiarezza e comprensibilità, al fine di agevolare interventi di manutenzione semplificati, in conformità con le direttive stabilite nello standard Sun.	DG_8, DG_13, DG_14
4	ODG_4 Riusabilità	Il sistema sarà riusabile per il 90% delle componenti di business logic tramite l'utilizzo di design pattern.	DG_13
5	ODG_5 Tempo di risposta	Il sistema dovrebbe, fornire una risposta al front-end ad una richiesta https, in meno di un secondo nel 95% dei casi.	DG_12

1.2 Object Design Trade-Off

Trade-off	Descrizione
Tempi di risposta vs Sicurezza	Il sistema, per poter garantire una maggiore sicurezza, potrebbe in alcuni casi avere dei tempi di risposta maggiori, i quali saranno comunque meno di un secondo nell'80% dei casi tale sicurezza sarà garantita dal pacchetto "Secure Storage" fornito da Flutter, per i dati sensibili.
Tempi di risposta vs Robustezza	Il sistema, per poter garantire una maggior robustezza, potrebbe in alcuni casi avere dei tempi di risposta maggiori per poter effettuare dei controlli sugli input inseriti dall'utente. Nonostante ciò, i tempi di risposta saranno comunque al di sotto dei 2 secondi nell'85% dei casi.
Costi di sviluppo vs Utilizzo COTS	Il sistema, al fine di rientrare nel budget prestabilito, farà utilizzo di COTS.

1.3 Linee guida per la documentazione dell'interfaccia

Le linee guida per la documentazione delle interfacce forniscono le convenzioni che gli sviluppatori saranno tenuti a seguire durante la progettazione e lo sviluppo delle interfacce del sistema.

Nello specifico, per le seguenti tecnologie saranno seguite le indicazioni sul coding style e sulle best practices nei link sottostanti:

- Dart: https://dart.dev/effective-dart/style
- Flutter: https://docs.flutter.dev/perf/best-practices

1.4 Definizioni, acronimi ed abbreviazioni

- **Package:** raggruppamento di classi, interfacce o file correlati;
- **Design pattern:** template di soluzioni a problemi ricorrenti impiegati per ottenere riuso e flessibilità;
- **Interfaccia:** insieme di signature delle operazioni offerte dalla classe;

1.5 Riferimenti

Libro: -- Object-Oriented Software Engineering (Using UML, Patterns, and Java) Third Edition Autori: -- Bernd Bruegge & Allen H. Dutoit

RAD -- 2023_RAD_C08 _V.1.0

SDD – 2023_SDD_C08 _V.1.0

1.6 Component Off-the-Shelf

Nella realizzazione della nostra piattaforma andremo ad utilizzare componenti Off-the-Shelf già disponibili per facilitare lo sviluppo del progetto.

- Per la progettazione del lato Backend verrà utilizzato il framework Flutter e l'interazione con esso avverrà tramite linguaggio di programmazione Dart che fornisce librerie standard di supporto necessarie per lo sviluppo.
- Per la progettazione del lato Frontend verrà utilizzato Flutter Engine che si occupa della composizione grafica dell'interfaccia utente, utilizza una grafica accelerata per garantire prestazioni elevate ed offre un'API per interagire con il framework Flutter.
- I dati verranno memorizzati all'interno di un DBMS PostGreSQL e le interazioni avverranno tramite le Rest API. La connessione è garantita tramite il pacchetto 'postgres' specifico per Dart/Flutter.

Tutte le tecnologie evidenziate possono essere utilizzate gratuitamente, in tal modo si riduce la quantità di lavoro necessario mantenendo inalterati i costi.

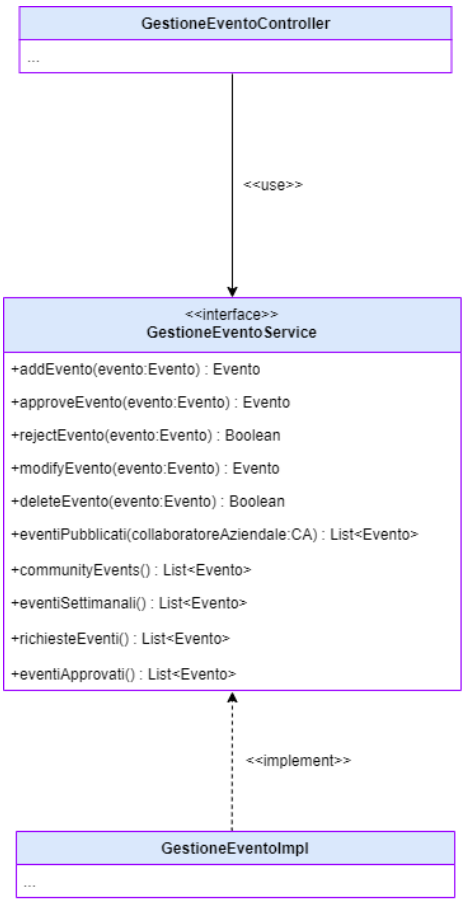
1.7 Design pattern

1.7.1 Facade

Il design pattern "Facade" che stiamo implementando nel nostro sistema è progettato per semplificare l'accesso a un sottosistema complesso. In questa implementazione, ogni sottosistema è dotato di un'interfaccia chiamata "Service", e agisce come una "facciata" che fornisce un punto di accesso unificato per i metodi necessari.

L'utilizzo di queste interfacce offre diversi vantaggi in termini di manutenibilità del codice. La riduzione dell'accoppiamento tra i vari componenti del sistema consente di apportare modifiche a un particolare layer senza impattare sull'intero sistema. Ciascuna interfaccia "Service" funge da strato di astrazione, e semplifica l'utilizzo dei servizi forniti dal sottosistema.

Di seguito un esempio di Facade nel sistema ReStart.

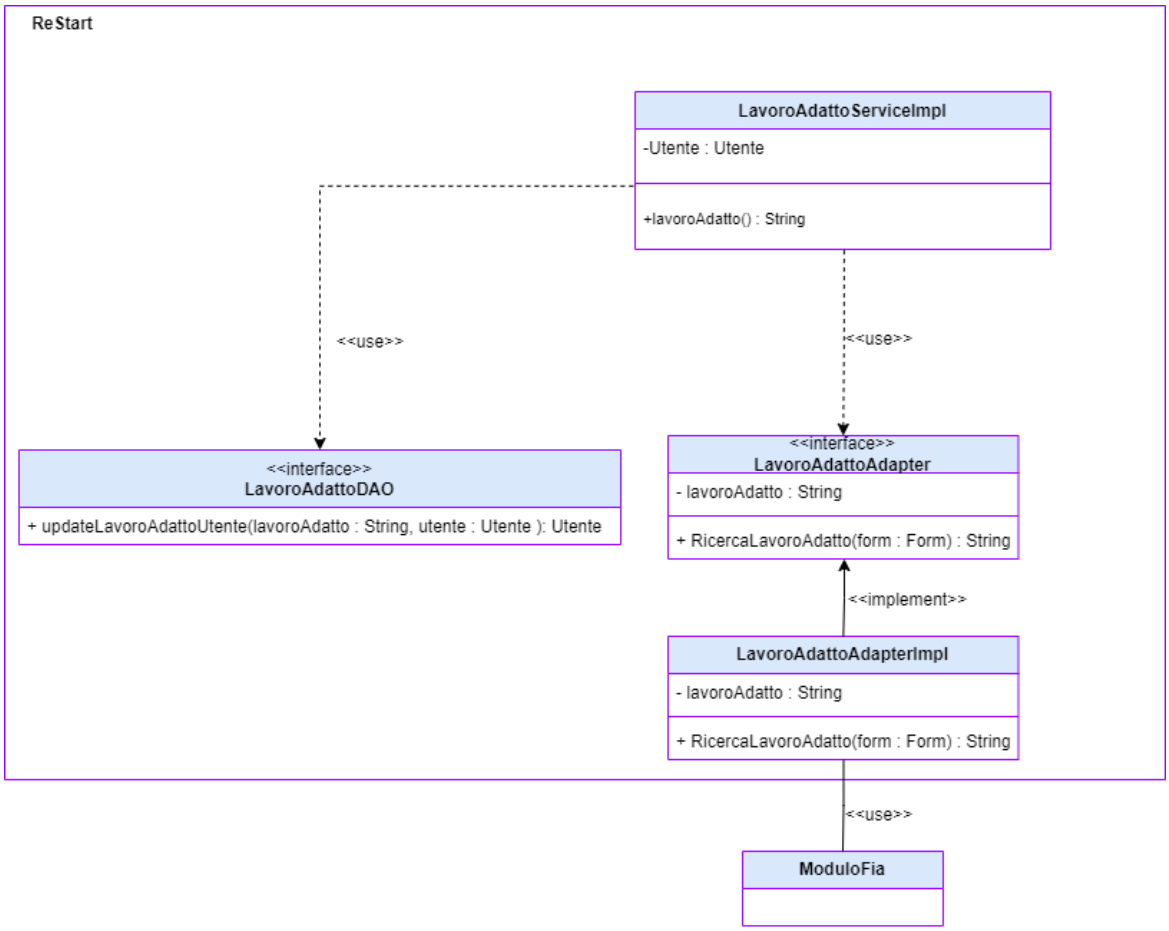


1.7.2 Adapter

L'Adapter rappresenta un design pattern strutturale, appartenete alla categoria di design pattern che agevolano la progettazione semplificando le relazioni tra entità. In particolare, l'Adapter consente a oggetti con interfacce diverse di collaborare efficacemente. Questo pattern si concretizza tramite una classe denominata "adapter", responsabile della conversione dei dati in formati comprensibili per il sistema.

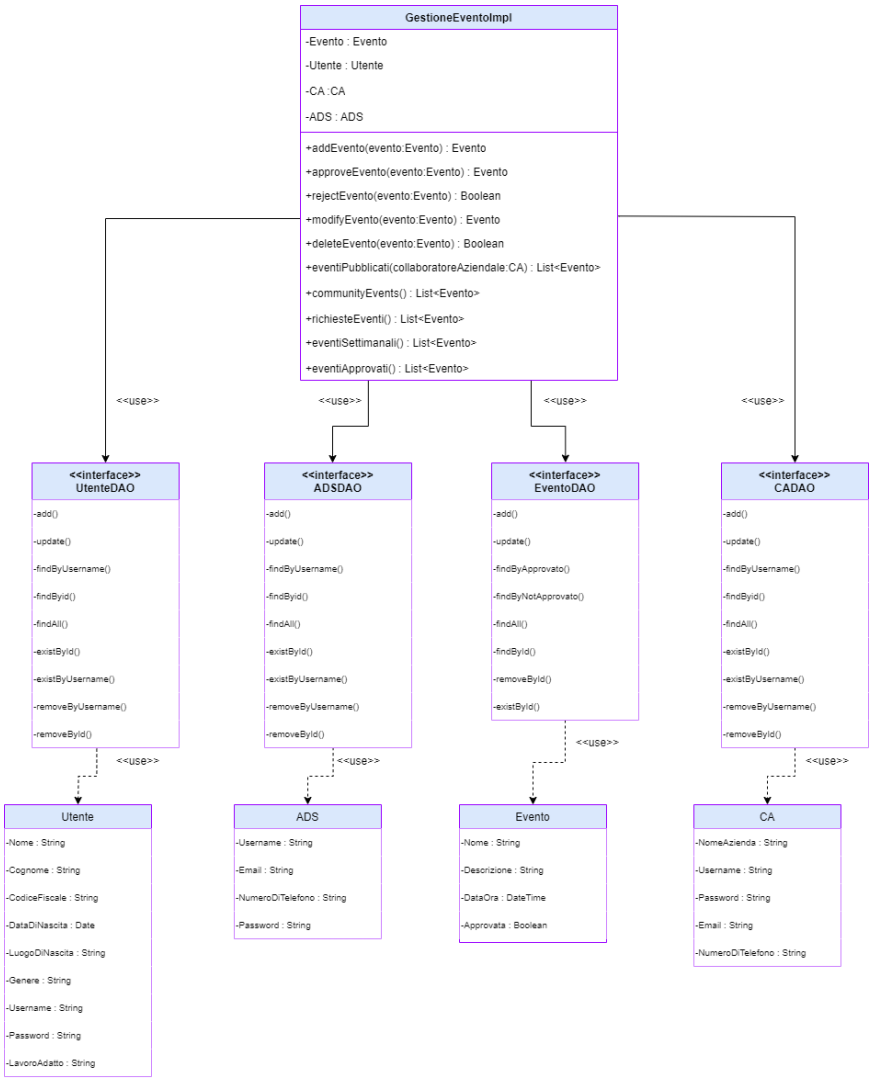
Nel contesto dell'integrazione del nostro modulo di Intelligenza Artificiale all'interno del sistema, l'utilizzo dell'Adapter si rivela fondamentale. La classe adapter sarà progettata per consentire lo scambio di informazioni tra il modulo di Intelligenza Artificiale e il sistema.

Di seguito un esempio di Adapter nel sistema ReStart.



1.7.3 DAO

Un DAO (Data Access Object) è un design pattern che fornisce un'interfaccia astratta per interagire con determinati tipi di database. Mediante la mappatura delle chiamate dell'applicazione allo stato persistente, il DAO consente l'esecuzione di operazioni specifiche sui dati senza rivelare i dettagli del database sottostante. ReStart, come mobile application dedicata alla gestione di numerosi servizi, si trova ad operare su un vasto database. La necessità di interagire con il database in modo rapido e sicuro, affrontando numerose query per gestire la moltitudine di dati, è cruciale per il corretto funzionamento dell'applicazione. Per soddisfare questa esigenza, abbiamo implementato diverse DAO nel nostro sistema. Di seguito è presentato un esempio di DAO utilizzato in ReStart.



2. Packages

In questa sezione viene mostrata la suddivisione del sistema in package, conforme a quanto definito nel documento di System Design. Tale suddivisione è motivata dalle scelte architetturali effettuate, con particolare attenzione alla rielaborazione della struttura Three Tier selezionata per la nostra applicazione.

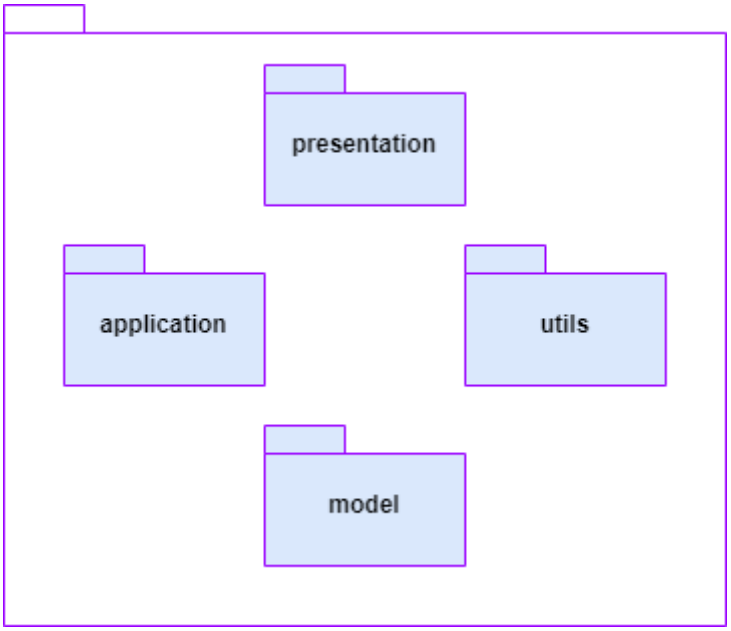
- **.idea**: contiene i file di configurazione specifici per l'IDE IntelliJ IDEA.
- **LavoroAdattoIA**: contiene i file riguardanti il modulo di IA
    - **Model**: contiene il modello di IA
    - **venv:** contiene tutte le dipendenze del modulo Py
- **ReStart**: contiene l'applicativo
    - **dart_tool**: contiene i file utilizzati dai vari tool di Dart.
    - **android**: contiene i file relativi all'applicazione lato Android.
    - **ios**: contiene i file relativi all'applicazione lato iOS.
    - **lib**: contiene tutti i file sorgente.
        - **presentation**: contiene tutti i file riguardanti le interfacce grafiche.
            - **components**: contiene tutte le componenti grafiche che vengono utilizzate in più screens.
            - **screens:** contiene una o più sottocartelle, ognuna delle quali corrisponde a una diversa schermata dell'applicazione.
        - **application**: contiene i vari packages relativi ad ogni sottosistema, con relative componenti Control e Service.
        - **model**: contiene tutti i file da utilizzare per l'accesso al database.
        - **utils**: contiene classi di utilità utilizzabili da più sottosistemi.
    - **test**: contiene tutti i file riguardanti il testing di unità.
    - **test_driver**: contiene i file riguardanti il testing di sistema.
    - **pubspec.yaml**: file che definisce le dipendenze del progetto, inclusi i pacchetti Dart e Flutter necessari. Inoltre, è anche possibile specificare la configurazione del progetto, come ad esempio nome, versione e altre informazioni di quest'ultimo.
    - **pubspec.lock**: contiene tutte le dipendenze effettive del nostro progetto, con informazioni riguardanti le loro versioni ed eventuali dipendenze transitive.

2.1 Package Restart

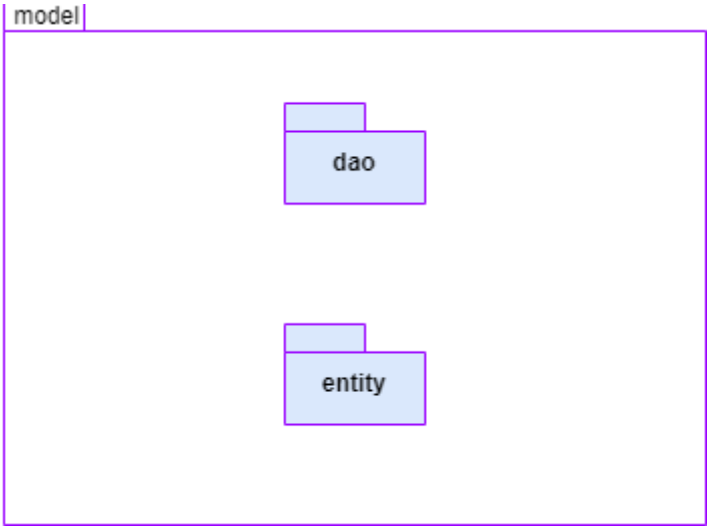
Nella presente sezione si mostra la struttura del package principale di ReStart.

La struttura generale che abbiamo adottato è stata di suddividere in quattro macro-package:

- **application:** al suo interno conterrà un package per ogni sottosistema;
- **presentation:** gestione dell'interfaccia utente;
- **model:** gestione delle classi entity e i DAO per l'accesso al DB;
- **utils:** in cui inserire eventuali classi di utilità per il sistema.



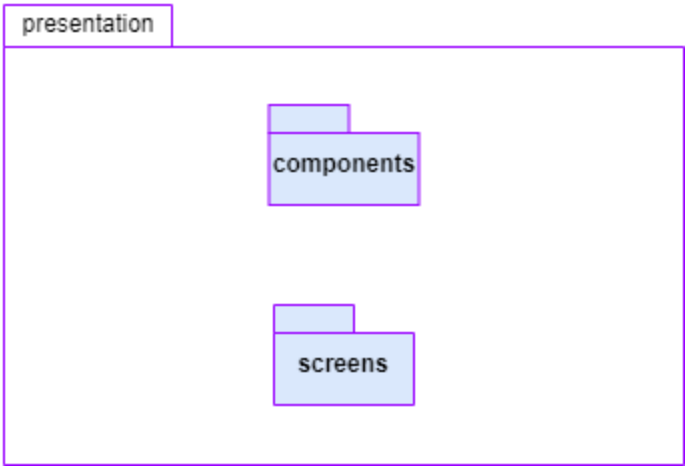
2.2 Package model



Questo package è suddiviso in due ulteriori packages.

- **connection**: contiene la connessione al database;
- **dao:** contiene tutti i DAO;
- **entity:** contiene tutte le entità del sistema.

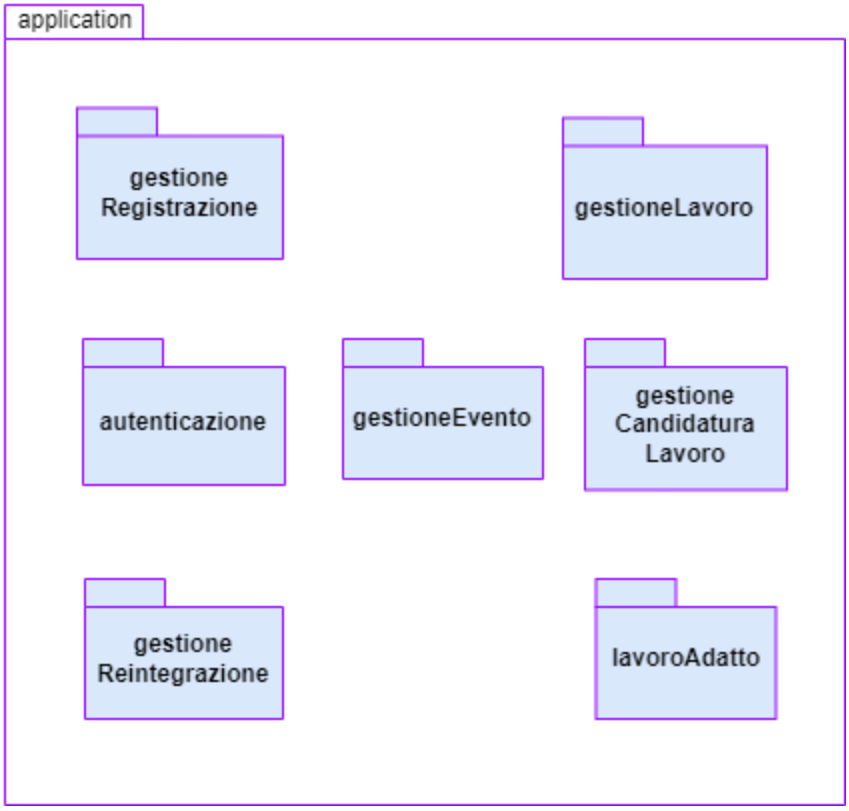
2.3 Package Presentation



Questo package è suddiviso in due ulteriori packages.

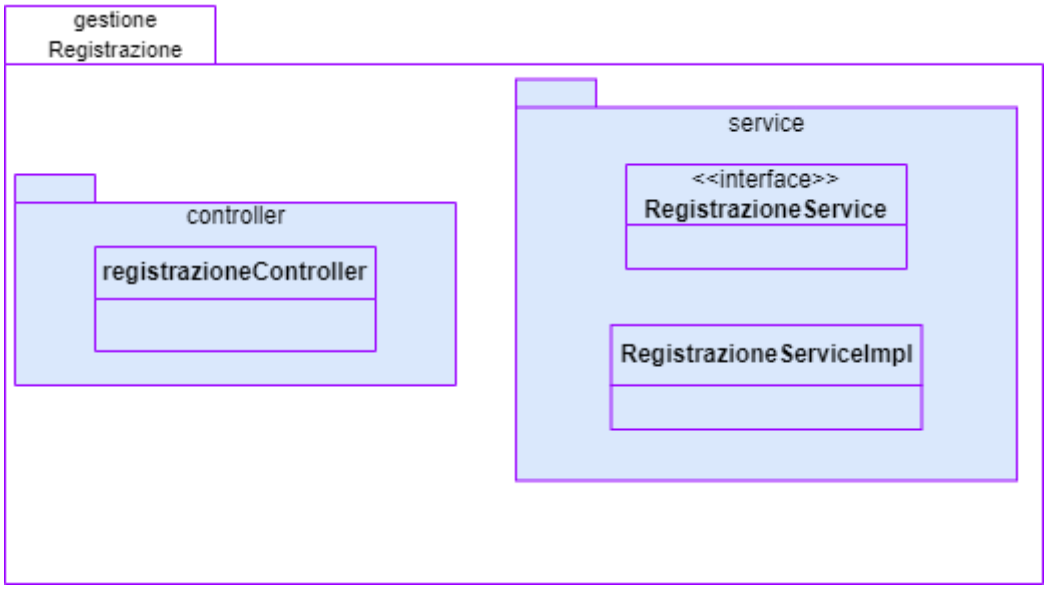
- **components**: comprende tutte le componenti di interfaccia grafica utilizzate da più screens.
- **screens:** contiene i vari screen, che sarebbero le varie pagine con relativi component, widget ecc.

2.4 Package application



Questo package è suddiviso in sette ulteriori packages, uno per ogni sottosistema della piattaforma.

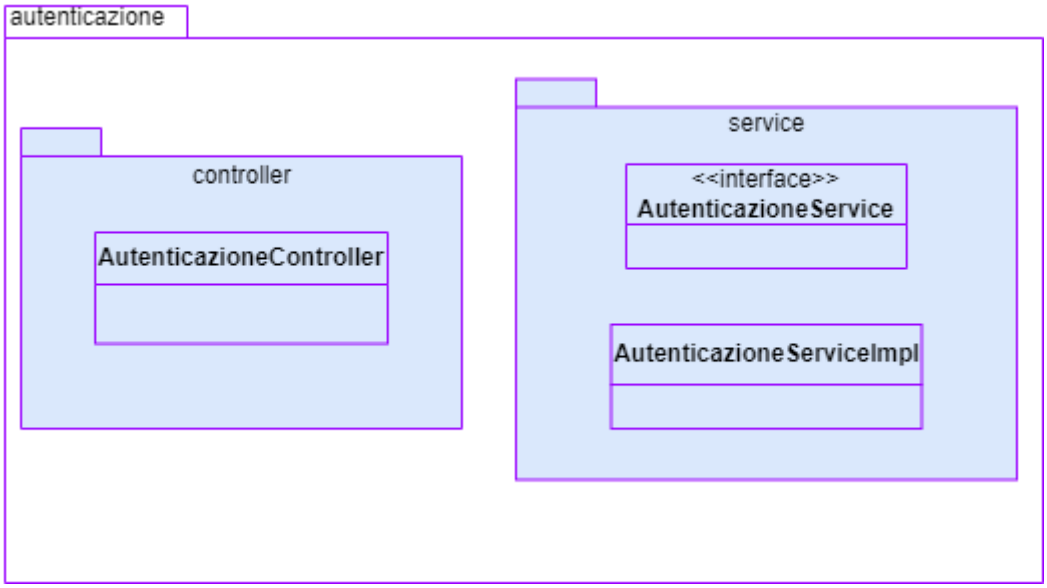
2.5 Package gestioneRegistrazione



Questo package è suddiviso in due ulteriori packages:

- **controller:** si occupa della comunicazione fra il service e il client;
- **service:** permette la comunicazione tra il service e i DAO.

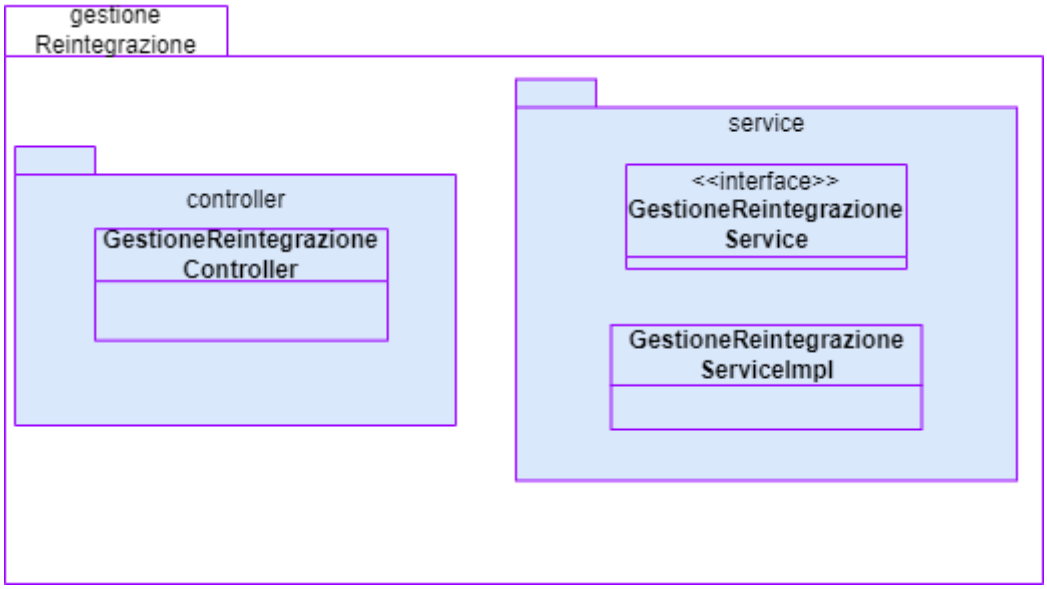
2.6 Package autenticazione



Questo package è suddiviso in due ulteriori packages:

- **controller:** si occupa della comunicazione fra il service e il client;
- **service:** permette la comunicazione tra il service e i DAO.

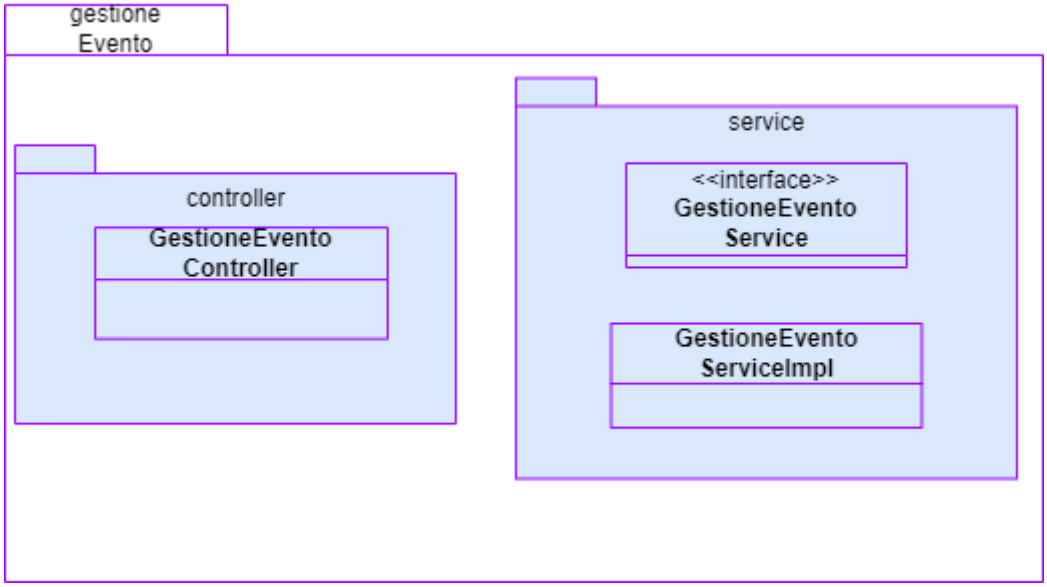
2.7 Package gestioneReintegrazione



Questo package è suddiviso in due ulteriori packages:

- **controller:** si occupa della comunicazione fra il service e il client;
- **service:** permette la comunicazione tra il service e i DAO.

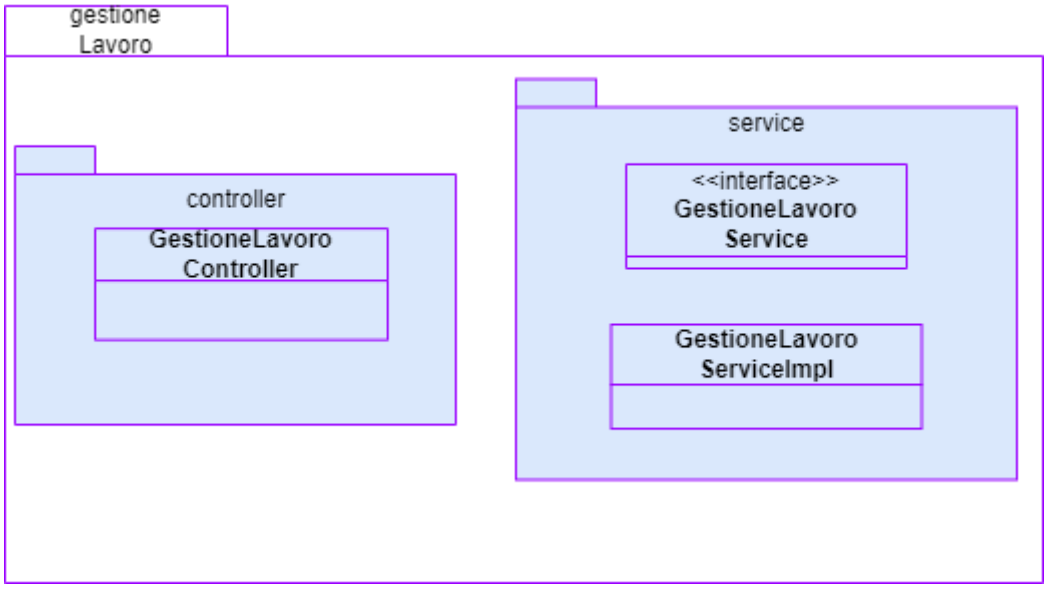
2.8 Package gestioneEvento



Questo package è suddiviso in due ulteriori packages:

- **controller:** si occupa della comunicazione fra il service e il client;
- **service:** permette la comunicazione tra il service e i DAO.

2.9 Package gestioneLavoro

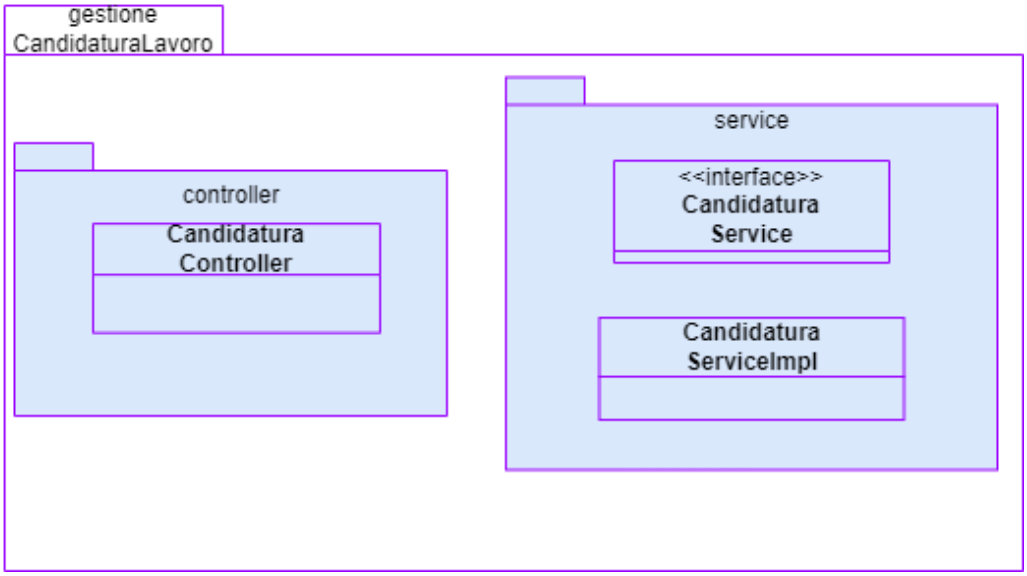


Questo package è suddiviso in due ulteriori packages:

- **controller:** si occupa della comunicazione fra il service e il client;
- **service:** permette la comunicazione tra il service e i DAO.

.

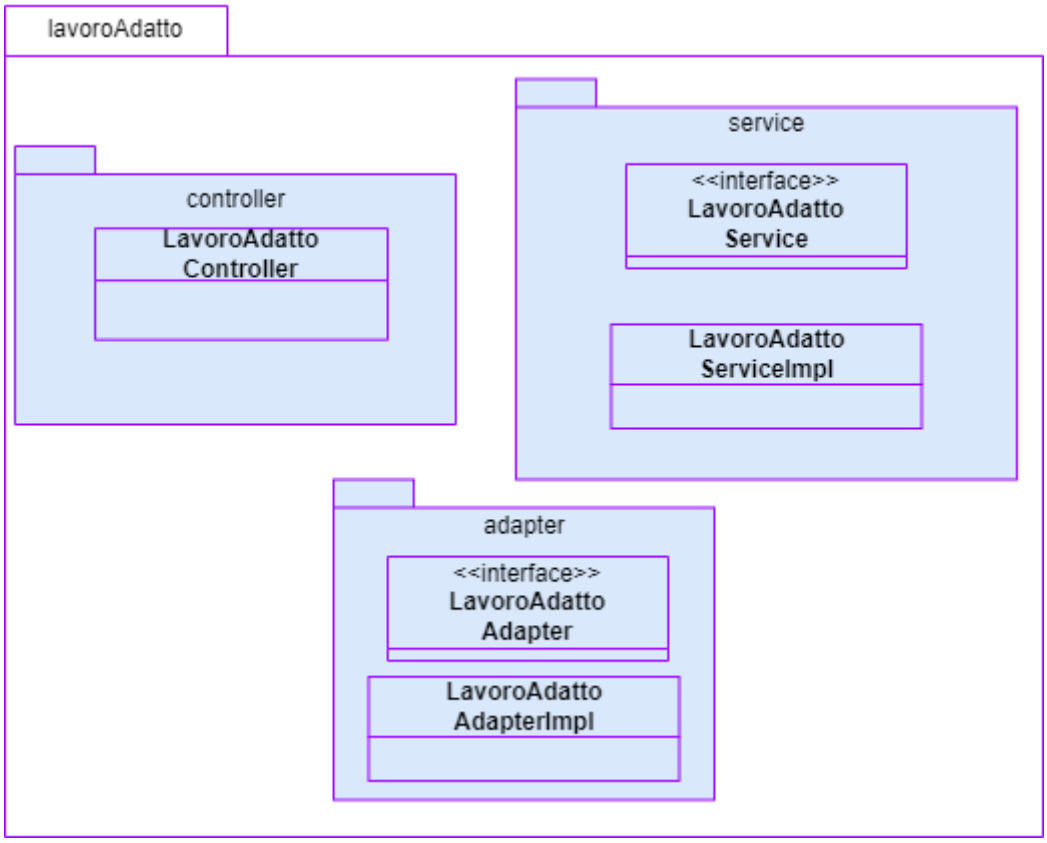
2.10 Package gestioneCandidaturaLavoro



Questo package è suddiviso in due ulteriori packages:

- **controller:** si occupa della comunicazione fra il service e il client;
- **service:** permette la comunicazione tra il service e i DAO.

2.11 Package lavoroAdatto



Questo package è suddiviso in due ulteriori packages:

- **controller:** si occupa della comunicazione fra il service e il client;
- **service:** permette la comunicazione tra il service e i DAO;
- **adapter:** si occupa di adattare i dati estrapolati dalla sezione LavoroAdatto trasformandoli in collezioni di oggetti manipolabili dal sistema.

3. Class Interfaces

3.1 Package gestioneRegistrazione

Nome Classe	RegistrazioneService
Descrizione	**Questa classe permette la gestione delle operazioni relative alla Registrazione.**
Metodi	+ signUp(utente:Utente):Utente
Invariante di classe	n/a

Nome Metodo	+signUp(utente:Utente): Utente
Descrizione	**Questo metodo permette di creare un Utente.**
Pre-condizione	**context**: RegistrazioneService::signUp(utente:Utente) **pre**: utente <> null
Post-condizione	**context**: RegistrazioneService::signUp(utente:Utente) **post**: utente <> null

3.2 Package autenticazione

Nome Classe	AutenticazioneService
Descrizione	**Questa classe permette la gestione delle operazioni relative all'Autenticazione.**
Metodi	+login (username:String, psw:String) : UtenteGenerico +modifyUtente(utente:Utente) : Utente +listaUtenti() : List<Utente> +deleteUtente(username:String):Boolean
Invariante di classe	n/a

Nome Metodo	+login (username:String, psw: String) : UtenteGenerico
Descrizione	**Questo metodo permette ad un utente di autenticarsi alla piattaforma.**
Pre-Condizione	**context**: AutenticazioneService:: login (username:String, psw:String) **pre**: username<> null AND psw<>null
Post-Condizione	**contex**: AutenticazioneService:: login (username:String, psw:String) **post:** isAutenticato(utente:Utente) <> false

Nome Metodo	+modifyUtente(utente:Utente) : Utente
Descrizione	**Questo metodo permette di modificare i dati di un utente.**
Pre-Condizione	**context**: AutenticazioneService::modifyUtente(utente:Utente) **pre**: utente <> null
Post-Condizione	**contex**: AutenticazioneService::modifyUtente(utente:Utente) : Utente **post:** utente <> null

Nome Metodo	+listaUtenti():List<Utente>
Descrizione	**Questo metodo permette di visualizzare tutti gli utenti presenti sulla piattaforma.**
Pre-Condizione	**context**: AutenticazioneService::listaUtenti() **pre**: n/a
Post-Condizione	**contex**: AutenticazioneService::listaUtenti() **post:** listaUtenti <> null

Nome Metodo	+deleteUtente(usarname:String) : Boolean
Descrizione	**Questo metodo permette di eliminare un utente.**
Pre-Condizione	**context**: AutenticazioneService::deleteUtente(username:String) **pre**: usarneme <> null
Post-Condizione	**contex**: AutenticazioneService::deleteUtente(username:String) **post:** flag <> false

3.3 Package gestioneReintegrazione

Nome Classe	GestioneReintegrazioneService
Descrizione	**Questa classe permette la gestione delle operazioni relative alla Gestione della reintegrazione.**
Metodi	+supportiMedici():List<SupportoMedico> +alloggiTemporanei():List<AlloggioTemporaneo> +corsiDiFormazione():List<CorsoDiFormazione> +addSupportoMedico(sm:SupportoMedico):SupportoMedico +addAlloggio(alloggio:AlloggioTemporaneo): AlloggioTemporaneo +addCorso(corso:CorsoDiFormazione): CorsoDiFormazione
Invariante di classe	n/a

Nome Metodo	+supportiMedici(): List<SupportoMedico>
Descrizione	**Questo metodo permette la visualizzazione della lista dei supporti medici.**
Pre-condizione	**context:** GestioneReintegrazioneService::supportiMedici() **pre:** n/a
Post-condizione	**context:** GestioneReintegrazioneService::supportiMedici() **post**: listaSupportoMedico <> null

Nome Metodo	+alloggiTemporanei(): List<AlloggioTemporaneo>
Descrizione	**Questo metodo permette la visualizzazione della lista degli alloggi temporanei.**
Pre-condizione	**context:** GestioneReintegrazioneService::alloggiTemporanei() **pre:** n/a
Post-condizione	**context:** GestioneReintegrazioneService::alloggiTemporanei() **post**: listaAlloggiTemporanei <> null

Nome Metodo	+corsiDiFormazione(): List<CorsoDiFormazione>
Descrizione	**Questo metodo permette la visualizzazione della lista dei corsi di formazione.**
Pre-condizione	**context:** GestioneReintegrazioneService::corsiDiFormazione() **pre:** n/a
Post-condizione	**context:** GestioneReintegrazioneService::corsiDiFormazione() **post**: listaCorsiDiFormazione <> null

Nome Metodo	+addSupportoMedico(sm:SupportoMedico): SupportoMedico
Descrizione	**Questo metodo permette l'aggiunta di un supporto medico.**
Pre-condizione	**context:** GestioneReintegrazioneService::addSupportoMedico(sm:SupportoMedico) **pre:** sm <> null
Post-condizione	**context:** GestioneReintegrazioneService::addSupportoMedico(sm:SupportoMedico) **post:** sm <> null

Nome Metodo	+addAlloggio(alloggio:AlloggioTemporaneo): AlloggioTemporaneo
Descrizione	**Questo metodo permette l'aggiunta di un alloggio temporaneo.**
Pre-condizione	**context:** GestioneReintegrazioneService::addAlloggio(AlloggioTemporaneo:alloggio) **pre:** alloggio <> null
Post-condizione	**context:**GestioneReintegrazioneService::addAlloggio(alloggio:AlloggioTemporaneo) **post:** alloggio <> null

Nome Metodo	+addCorso(corso:CorsoDiFormazione): CorsoDiFormazione
Descrizione	**Questo metodo permette l'aggiunta di un corso di formazione.**
Pre-condizione	**context:**GestioneReintegrazioneService::addCorso(corso:CorsoDiFormazione) **pre**: corso <> null
Post-condizione	**context:**GestioneReintegrazioneService::addCorso(corso:CorsoDiFormazione): **post:** corso <> null

3.4 Package gestioneEvento

Nome Classe	GestioneEventoService
Descrizione	**Questa classe permette la gestione delle operazioni relative alla Gestione Evento.**
Metodi	+CommunityEvents() : List<Evento> +eventiPubblicati(collaboratoreAziendale:CA): List<Evento> +addEvento(evento:Evento) : Evento +modifyEvento(evento:Evento) : Evento +deleteEvento(evento:Evento) : Boolean +approveEvento(evento:Evento) : Evento +rejectEvento(evento:Evento) : Boolean +richiesteEventi() : List<Evento> +eventiSettimanali() : List<Evento> +eventiApprovati() : List<Evento>
Invariante di classe	n/a

Nome Metodo	+CommunityEvents():List<Evento>
Descrizione	**Questo metodo permette di visualizzare la lista di tutti gli eventi presenti sulla piattaforma.**
Pre-condizione	**context**: GestioneEventoService::CommunityEvents():List<Evento> **pre**: n/a
Post-condizione	**context**: GestioneEventoService::CommunityEvents():List<Evento> **post**: listaEventi <> null

Nome Metodo	+eventiPubblicati(collaboratoreAziendale:CA):List<Evento>
Descrizione	**Questo metodo permette di visualizzare la lista di tutti gli eventi pubblicati da uno specifico Collaboratore Aziendale.**
Pre-condizione	**context**: GestioneEventoService::eventiPubblicati(collaboratoreAziendale:CA) **pre**: collaboratoreAziendale <> null
Post-condizione	**context:** GestioneEventoService::eventiPubblicati(collaboratoreAziendale:CA) **post:** listaEventiPubblicati <> null

Nome Metodo	+ addEvento(evento:Evento) : Evento
Descrizione	**Questo metodo permette di inserire un evento sulla piattaforma.**
Pre-condizione	**context**: GestioneEventoService::addEvento(evento:Evento) **pre**: evento <> null
Post-condizione	**context**: GestioneEventoService::addEvento(evento:Evento) **post**: evento <> null

Nome Metodo	+ modifyEvento(evento:Evento):Evento
Descrizione	**Questo metodo permette di modificare un evento.**
Pre-condizione	**context**:GestioneEventoService::modifyEvento(evento:Evento) **pre**: evento <> null
Post-condizione	**context**: GestioneEventoService::modifyEvento(evento:Evento) **post**: evento <> null

Nome Metodo	+deleteEvento(evento:Evento):Boolean
Descrizione	**Questo metodo permette di eliminare un evento.**
Pre-condizione	**context**:GestioneEventoService::deleteEvento(evento:Evento) **pre**: evento <> null
Post-condizione	**context**: GestioneEventoService::deleteEvento(evento:Evento) **post**: flag <> false

Nome Metodo	+approveEvento(Evento evento):Evento
Descrizione	**Questo metodo permette di approvare un evento.**
Pre-condizione	**context**: GestioneEventoService:: approveEvento (evento:Evento) **pre**: evento <> null
Post-condizione	**context**: GestioneEventoService::approveEvento (evento:Evento) **post**: evento <> null

Nome Metodo	+rejectEvento(evento:Evento):Boolean
Descrizione	**Questo metodo permette di rifiutare un evento.**
Pre-condizione	**context**: GestioneEventoService:: rejectEvento (evento:Evento) **pre**: evento <> null
Post-condizione	**context**: GestioneEventoService:: rejectEvento (evento:Evento) **post**: flag <> false

Nome Metodo	+richiesteEventi() : List<Evento>
Descrizione	**Questo metodo permette di visualizzare le richieste per l'aggiunta di un evento.**
Pre-condizione	**context**: GestioneEventoService:: richiesteEventi(): List<Evento> **pre**: n/a
Post-condizione	**context**: GestioneEventoService:: richiesteEventi(): List<Evento> **post**: listaEventi <> null

Nome Metodo	+eventiSettimanali(): List<Evento>
Descrizione	**Questo metodo permette di visualizzare gli eventi settimanali.**
Pre-condizione	**context**: GestioneEventoService:: eventiSettimanali(): List<Evento> **pre**: evento <> null
Post-condizione	**context**: GestioneEventoService:: eventiSettimanali(): List<Evento> **post**: listaEventiSettimanali <> null

Nome Metodo	+eventiApprovati() : List<Evento>
Descrizione	**Questo metodo permette di visualizzare gli eventi approvati.**
Pre-condizione	**context**: GestioneEventoService:: eventiApprovati(): List<Evento> **pre**: evento <> null
Post-condizione	**context**: GestioneEventoService:: eventiApprovati(): List<Evento> **post**: listaEventiApprovati <> null

3.5 Package gestioneLavoro

Nome Classe	GestioneLavoroService
Descrizione	**Questa classe permette la gestione delle operazioni relative alla Gestione Lavoro.**
Metodi	+offerteLavoro() : List< AnnuncioDiLavoro > +offertePubblicate(collaboratoreAziendale:CA) : List< AnnuncioDiLavoro > +addLavoro(annuncioDiLavoro : AnnuncioDiLavoro): AnnuncioDiLavoro +modifyLavoro(annuncioDiLavoro : AnnuncioDiLavoro): AnnuncioDiLavoro +deleteLavoro(annuncioDiLavoro : AnnuncioDiLavoro):Boolean +UtentiCandidati(annuncioDiLavoro : AnnuncioDiLavoro): List<Utente> +profiloUtenteCandidato(utente:Utente) : Utente +approveLavoro(annuncioDiLavoro : AnnuncioDiLavoro): AnnuncioDiLavoro +rejectLavoro(annuncioDiLavoro : AnnuncioDiLavoro): Boolean
Invariante di Classe	n/a

Nome Metodo	+offerteLavoro() : List<AnnuncioDiLavoro>
Descrizione	**Questo metodo permette di visualizzare la lista di tutte le offerte di lavoro presenti sulla piattaforma.**
Pre-condizione	**context**: GestioneLavoroService::offerteLavoro():List<AnnuncioDiLavoro > **pre**: n/a
Post-condizione	**context:** GestioneLavoroService::offerteLavoro():List<AnnuncioDiLavoro > **post:** listaLavoro <> null

Nome Metodo	+offertePubblicate(collaboratoreAziendale:CA) : List<AnnuncioDiLavoro>
Descrizione	**Questo metodo permette di visualizzare la lista di tutte le offerte di lavoro pubblicate da uno specifico Collaboratore Aziendale.**
Pre-condizione	**context**: GestioneLavoroService::offertePubblicate(collaboratoreAziendale:CA) **pre**: collaboratoreAziendale <> null
Post-condizione	**context:** GestioneLavoroService:: offertePubblicate(collaboratoreAziendale:CA) **post:** listaLavoro<> null

Nome Metodo	+addLavoro(annuncioDiLavoro:AnnuncioDiLavoro) : AnnuncioDiLavoro
Descrizione	Questo metodo permette di inserire un annuncio di lavoro.
Pre-condizione	**context:** GestioneLavoroService::addLavoro(annuncioDiLavoro : AnnuncioDiLavoro) **pre:** annuncioDiLavoro <> null
Post-condizione	**context:** GestioneLavoroService:: addLavoro(annuncioDiLavoro : AnnuncioDiLavoro) **post:** annuncioDiLavoro <> null

Nome Metodo	+modifyLavoro(annuncioDiLavoro:AnnuncioDiLavoro) :AnnuncioDiLavoro
Descrizione	**Questo metodo permette di modificare un annuncio di lavoro.**
Pre-condizione	**context:** GestioneLavoroService::modifyLavoro(annuncioDiLavoro : AnnuncioDiLavoro) **pre:** annuncioDiLavoro <> null
Post-condizione	**context:** GestioneLavoroService::modifyLavoro(annuncioDiLavoro : AnnuncioDiLavoro) **post:** annuncioDiLavoro <> null

Nome Metodo	+deleteLavoro(annuncioDiLavoro:AnnuncioDiLavoro) :Boolean
Descrizione	**Questo metodo permette di eliminare un annuncio di lavoro.**
Pre-condizione	**context:** GestioneLavoroService:: deleteLavoro(annuncioDiLavoro : AnnuncioDiLavoro) **pre:** annuncioDiLavoro <> null
Post-condizione	**context:** GestioneLavoroService::deleteLavoro(annuncioDiLavoro : AnnuncioDiLavoro) **post:** flag <> false

Nome Metodo	+utentiCandidati(annuncioDiLavoro:AnnuncioDiLavoro) : List<Utente>
Descrizione	**Questo metodo permette di visualizzare gli utenti candidati ad un annuncio di lavoro.**
Pre-condizione	**context:** GestioneLavoroService:: UtentiCandidati(annuncioDiLavoro : AnnuncioDiLavoro) **pre:** annuncioDiLavoro <> null
Post-condizione	**context:** GestioneLavoroService:: utentiCandidati(annuncioDiLavoro : AnnuncioDiLavoro) **post:** listaUtentiCandidati <> null

Nome Metodo	+profiloUtenteCandidato(annuncioDiLavoro : AnnuncioDiLavoro) : Utente
Descrizione	**Questo metodo permette di visualizzare il profilo di uno specifico utente candidato ad un annuncio di lavoro.**
Pre-condizione	**context:** GestioneLavoroService::profiloUtenteCandidati(annuncioDiLavoro : AnnuncioDiLavoro) **pre:** n/a
Post-condizione	**context:** GestioneLavoroService::profiloUtenteCandidati(annuncioDiLavoro : AnnuncioDiLavoro) **pre:** Utente <> null

Nome Metodo	+approveLavoro(annuncioDiLavoro:AnnuncioDiLavoro) : Lavoro
Descrizione	**Questo metodo permette di approvare un annuncio di lavoro.**
Pre-condizione	**context**: GestioneLavoroService:: approveLavoro (annuncioDiLavoro : AnnuncioDiLavoro) **pre**: annuncioDiLavoro <> null
Post-condizione	**context:** GestioneLavoroService:: approveLavoro(annuncioDiLavoro : AnnuncioDiLavoro) **post:** annuncioDiLavoro <> null

Nome Metodo	+rejectLavoro(annuncioDiLavoro:AnnuncioDiLavoro) : Boolean
Descrizione	**Questo metodo permette di rifiutare una richiesta di un annuncio di lavoro.**
Pre-condizione	**context:** GestioneLavoroService:: rejectLavoro(annuncioDiLavoro : AnnuncioDiLavoro) **pre:** annuncioDiLavoro <> null
Post-condizione	**context**: GestioneLavoroService::rejectLavoro (annuncioDiLavoro : AnnuncioDiLavoro) **post**: flag <> false

3.6 Package gestioneCandidaturaLavoro

Nome Classe	CandidaturaLavoroService
Descrizione	**Questa classe permette la gestione delle operazioni relative alla Candidatura Lavoro.**
Metodi	+candidatura (annuncioDilavoro:AnnuncioDiLavoro, utente:Utente):Boolean
Invariante di classe	n/a

Nome Metodo	+candidatura(annuncioDiLavoro:AnnuncioDiLavoro, utente:Utente):Boolean
Descrizione	**Questo metodo permette ad un utente di candidarsi ad un lavoro.**
Pre-condizione	**context**: CandidaturaLavoroService::candidatura(annuncioDiLavoro:AnnuncioDiLavoro, utente:Utente) **pre**: annuncioDiLavoro <> null AND utente <> null
Post-condizione	**context**: CandidaturaLavoroService::candidatura(annuncioDiLavoro:AnnuncioDiLavoro, utente:Utente) **post**: flag <> false

3.7 Package lavoroAdatto

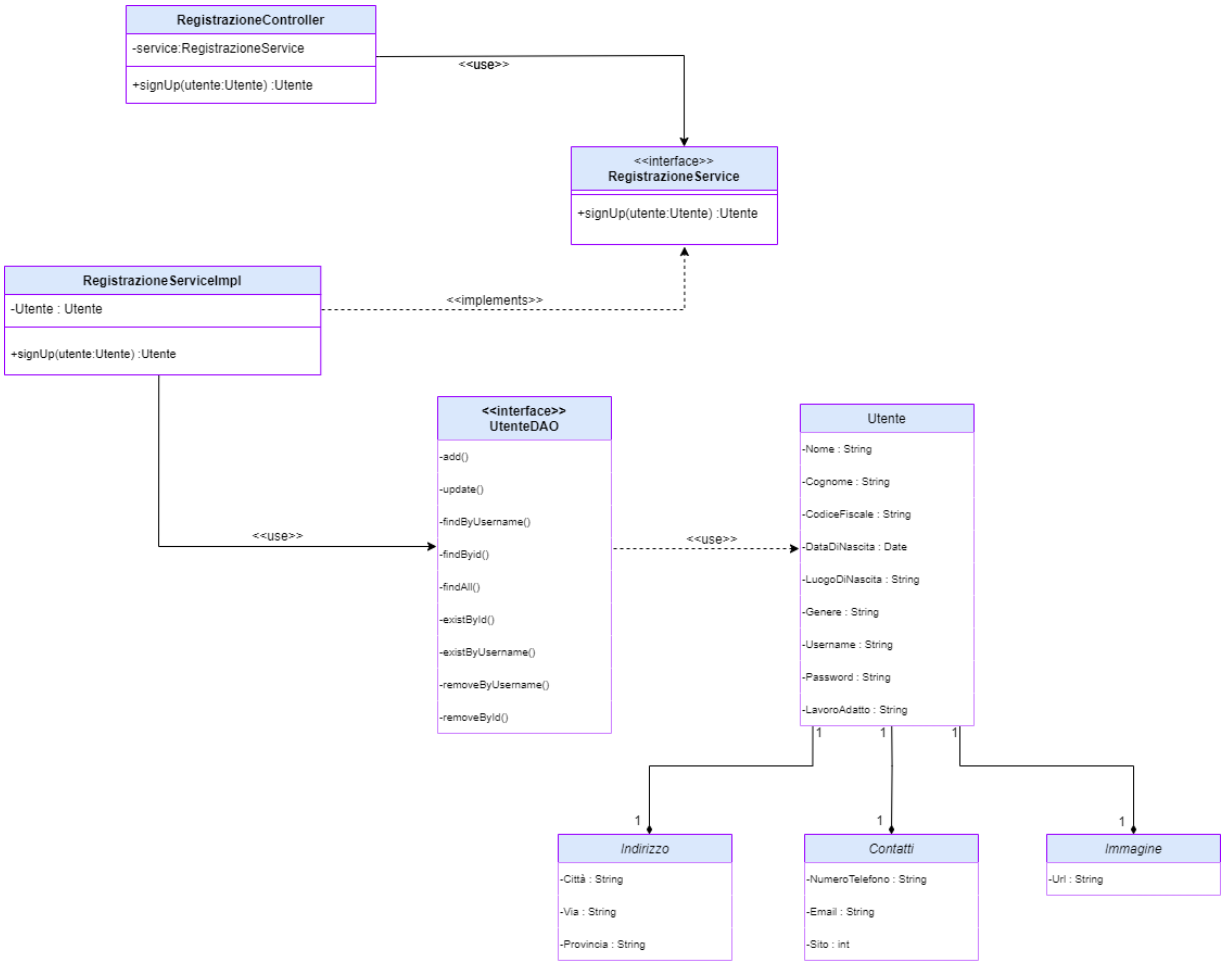
Nome Classe	LavoroAdattoService
Descrizione	**Questa classe permette la gestione delle operazioni relative al Lavoro Adatto.**
Metodi	+lavoroAdatto() : String
Invariante di classe	n/a

Nome Metodo	+lavoroAdatto(): String
Descrizione	**Questo metodo permette all'utente di trovare il lavoro adatto a lui.**
Pre-condizione	**context:**LavoroAdattoService::lavoroAdatto() **pre:** n/a
Post-condizione	**context:**LavoroAdattoService::lavoroAdatto() **post:** lavoroAdatto <> null

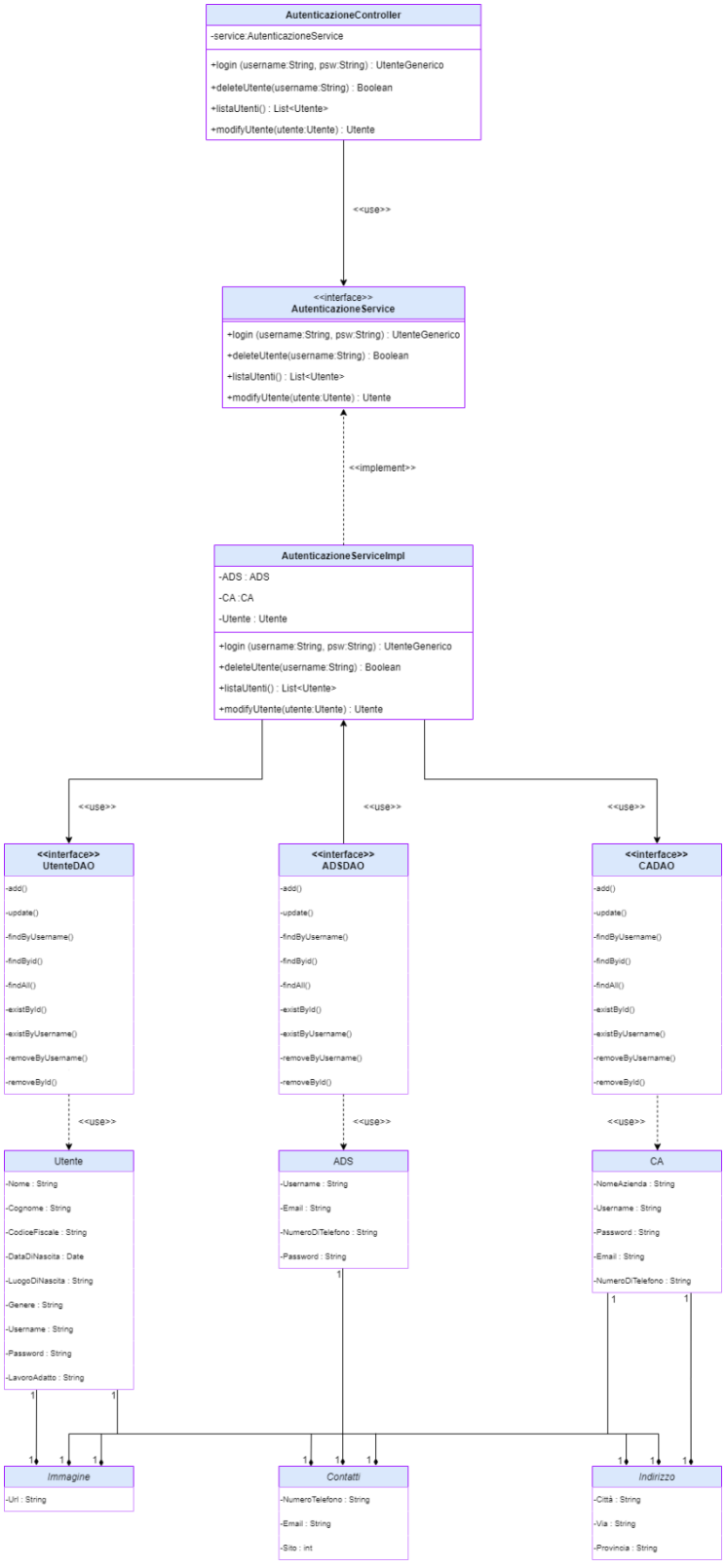
4. Class Diagram

Per una questione di ordine e leggibilità del class diagram lo abbiamo diviso per package.

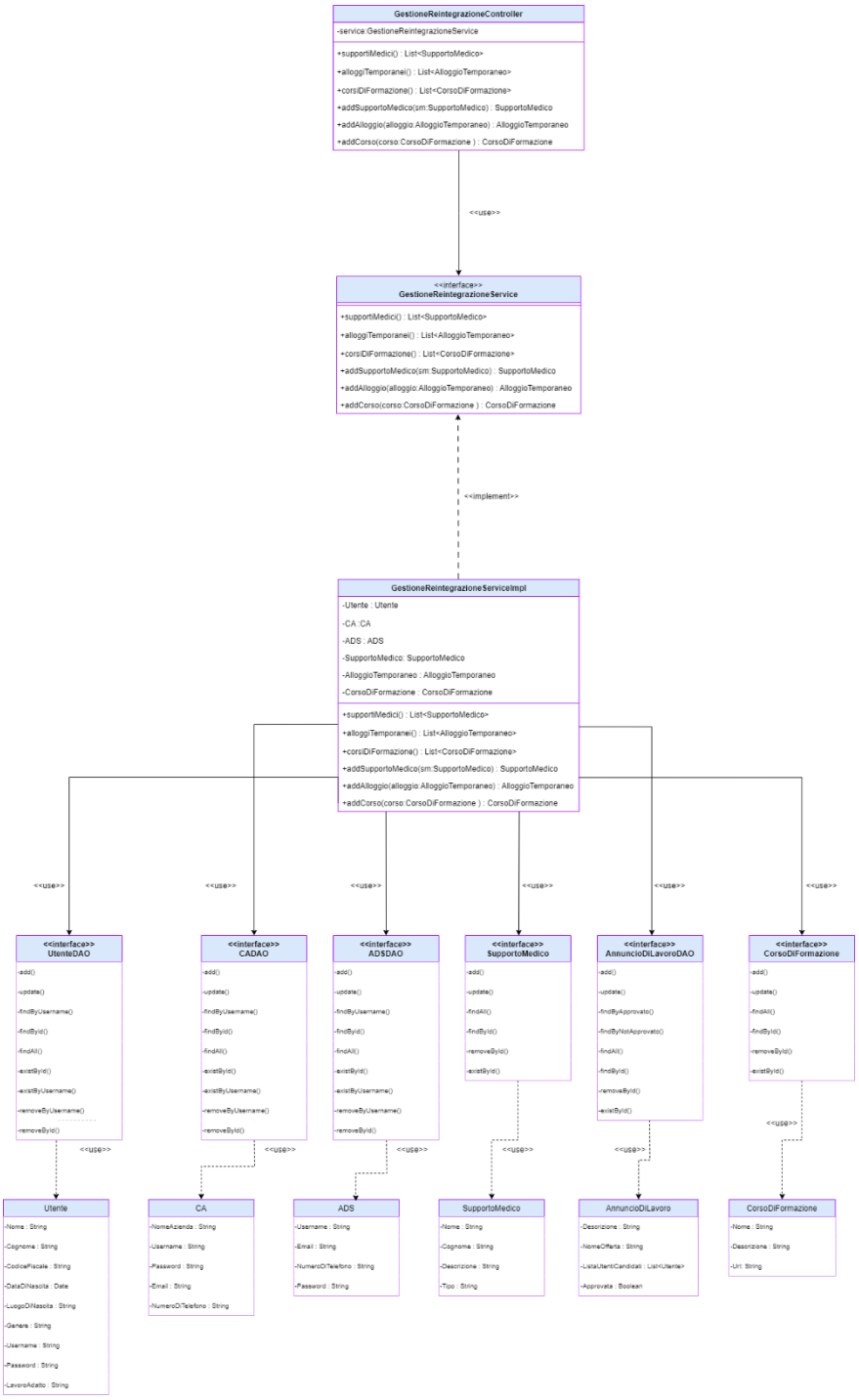
4.1 Package gestioneRegistrazione



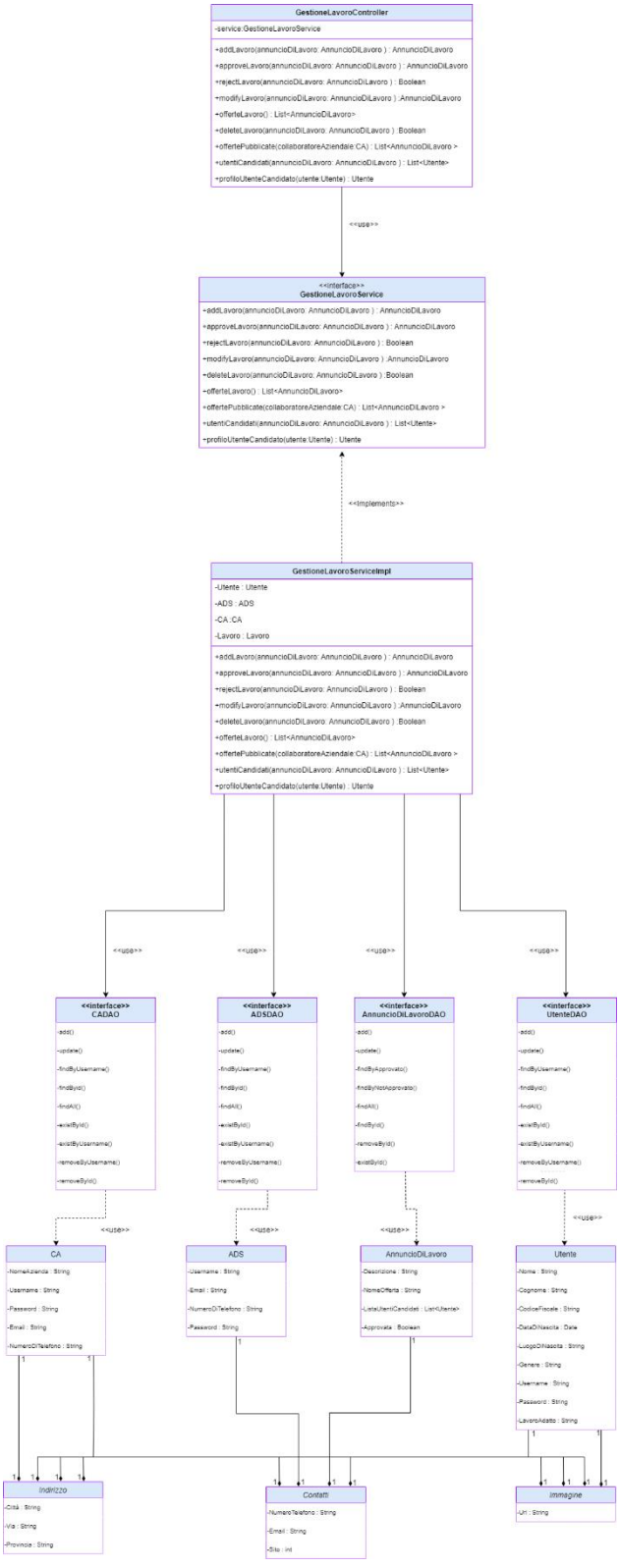
4.2 Package autenticazione



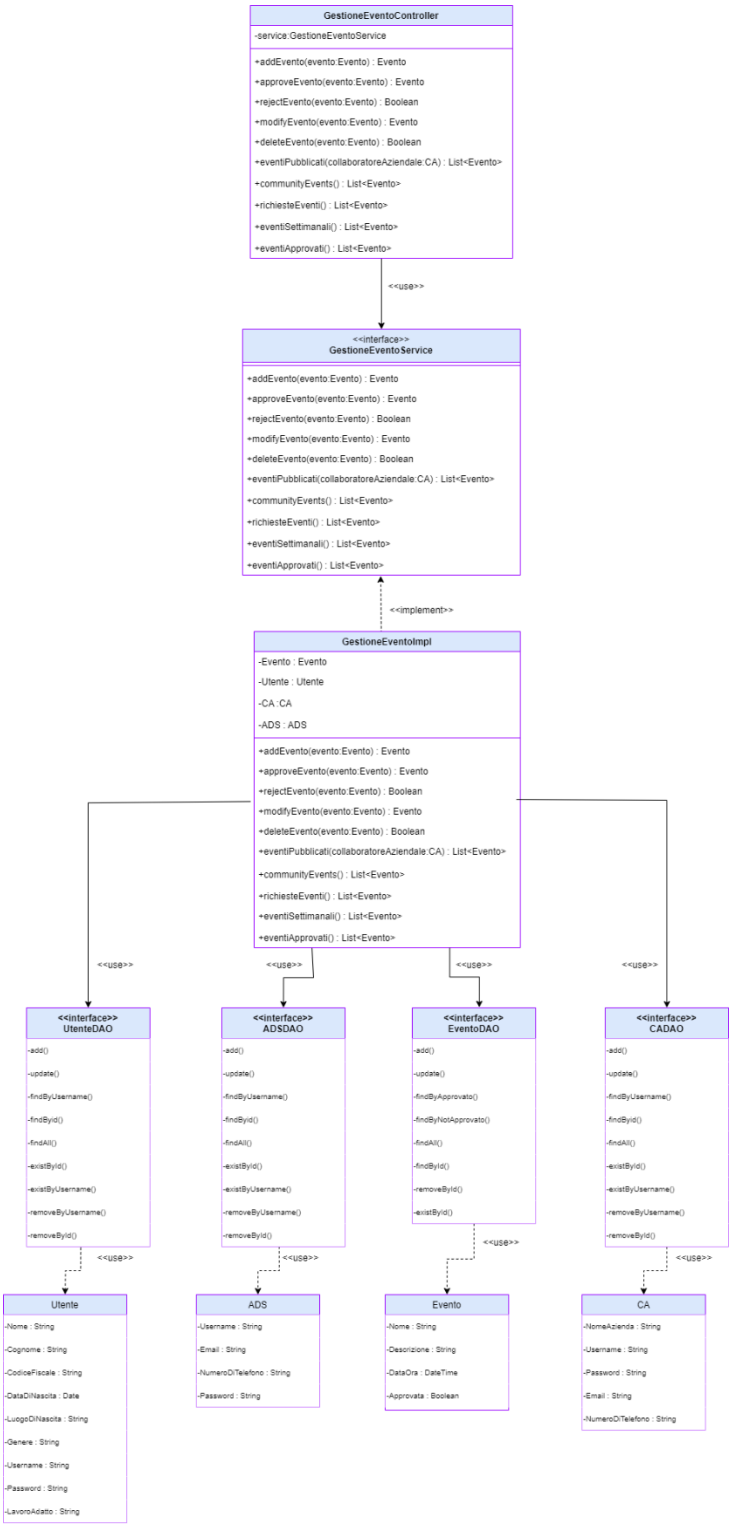
4.3 Package gestioneReintegrazione



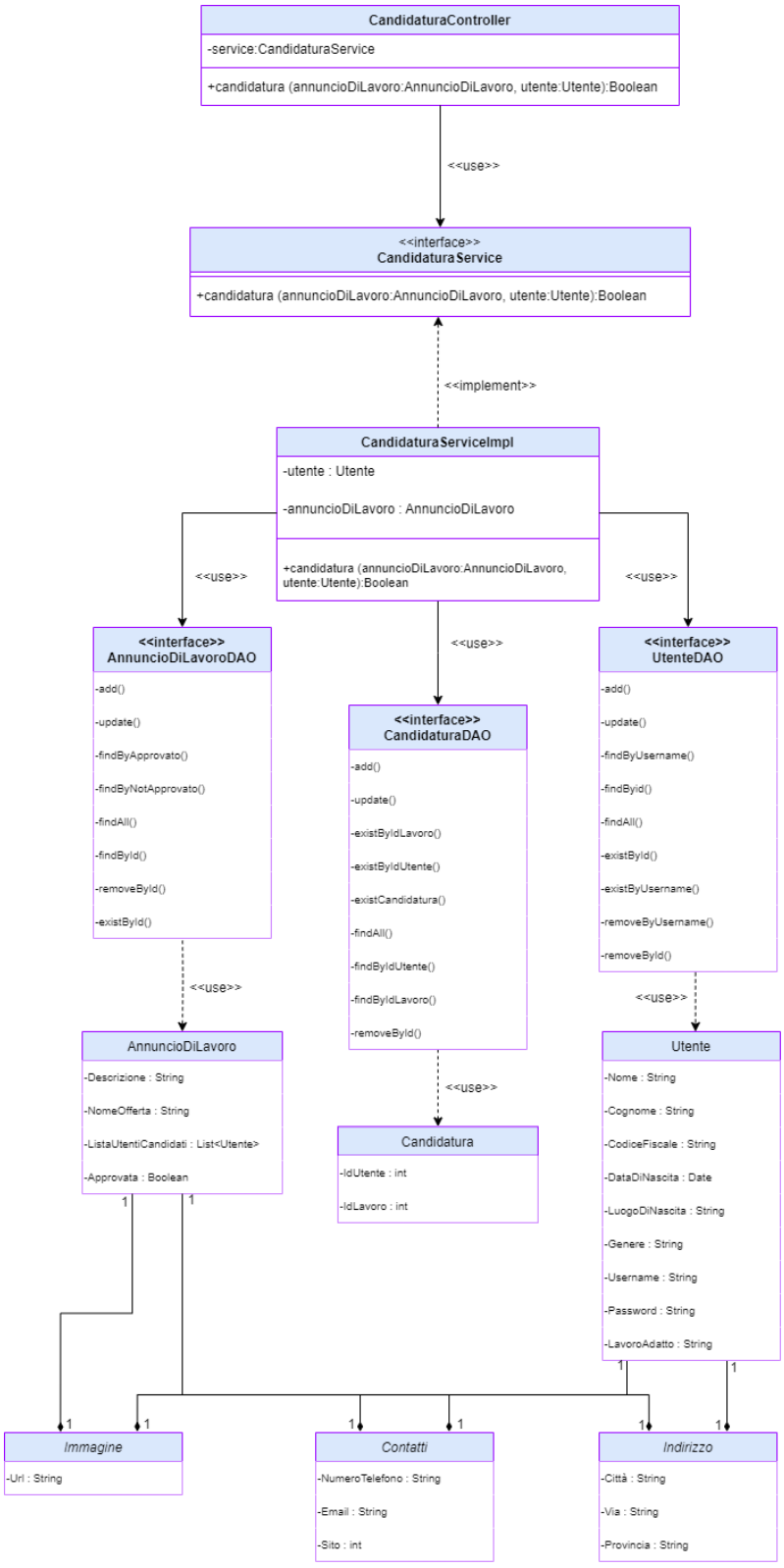
4.4 Package gestioneLavoro



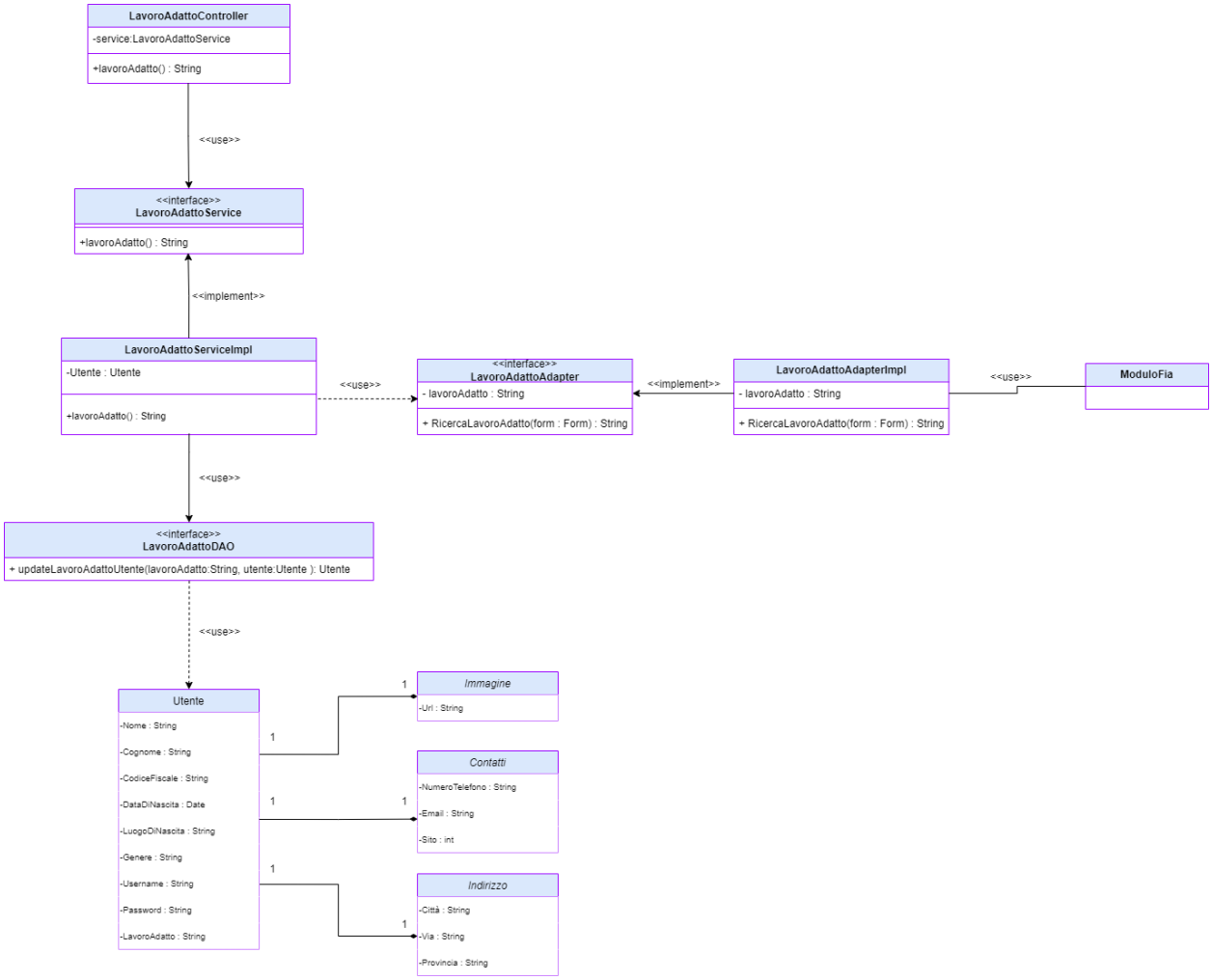
4.5 Package gestioneEvento



4.6 Package gestioneCandidaturaLavoro



4.7 Package lavoroAdatto



5. Glossario

Sigla/Termine	Definizione
Componenti Off-the-Shelf	Componenti hardware e software disponibili sul mercato per l'acquisto da parte di aziende di sviluppo interessate a utilizzarli nei loro progetti.
Backend	Con il termine backend si indica l'interfaccia con la quale il gestore di un'applicazione ne gestisce i contenuti e le funzionalità.
Frontend	Con il termine frontend si indica l'insieme delle applicazioni e dei programmi informatici con cui l'utente interagisce direttamente (contrapposto a backend).
PostgreSQL	PostgreSQL è un completo DBMS ad oggetti rilasciato con licenza libera.
DBMS	Sistema software progettato per consentire la creazione, la manipolazione e l'interrogazione efficiente di database, ospitato su architettura hardware dedicata oppure su semplice computer.
Package	Raggruppamento di classi ed interfacce.
DAO	Data Access Object, implementazione dell'omonimo pattern architetturale che si occupa di fornire un accesso in modo astratto ai dati persistenti.
Controller	Classe che si occupa di gestire le richieste effettuate dal client.
Service	Classe che implementa la logica di business viene utilizzata dal controller o da un altro sottosistema.
Facade	Un oggetto che permette, attraverso un'interfaccia più semplice, l'accesso a sottosistemi che espongono interfacce complesse e molto diverse tra loro.
Adapter	È un pattern strutturale che può essere basato sia su classi che su oggetti il cui fine è fornire una soluzione astratta al problema dell'interoperabilità tra interfacce differenti.