Object Design Document

FindYourPlace

Riferimento	
Versione	1.1
Data	08/02/2024
Presentato da:	NC28: Pietro Esposito, Alessandro Nacchia, Lorenzo Castellano
Approvato da:	

Revision History

Data	Versione	Descrizione	Autori
04/02/2024	0.1	Prima stesura.	Alessandro Nacchia
04/02/2024	0.2	Stesura Capitolo 1-2	Lorenzo Castellano
05/02/2024	0.9	Stesura Capitolo 3-4-5-6	Alessandro Nacchia
08/02/2024	1.0	Fix Struttura Directory, aggiunti Design Patterns e Revisione	Pietro Esposito
08/02/2024	1.1	Corretti alcuni errori	Alessandro Nacchia

Sommario

1. Introduzione

FindYourPlace è un sistema che propone la semplificazione del processo di ricerca del luogo adatto al proprio stile di vita comprimendo tutti i vari passaggi in un unico sito semplice e intuitivo.

In questa prima sezione del documento, verranno descritti i trade-offs e le linee guida per la fase di implementazione, riguardanti la nomenclatura, la documentazione e le convenzioni sui formati.

1.1 Object Design Goals & Trade-offs

- Prestazioni: Il sistema deve risultare prestante, dimostrando velocità e adattabilità;
- Affidabilità: Il sistema deve risultare Affidabile, garantendo disponibilità, robustezza e sicurezza all'utente e ai suoi dati, reagendo correttamente a situazioni impreviste;
- Sostenibilità: Il sistema prevede e permette la correzione di bug o errori e favorisce l'implementazione di eventuali futuri miglioramenti;
- End User: Il sistema si basa sul facilitare l'esperienza dell'utente, preoccupandosi di essere il più possibile intuitivo e facile da usare.

Trade-off	Descrizione
Tempi di risposta vs Sicurezza	Il sistema, col fine di garantire una maggiore sicurezza, potrebbe richiedere un tempo di risposta maggiore. Saranno implementati controlli e crittografia sui dati più sensibili.
Costi vs Prestazioni	Considerando il budget ridotto a disposizione, si preferisce rientrare nei costi previsti dedicando meno tempo all'ottimizzazione delle prestazioni del sistema, possibilmente aumentando i tempi di caricamento.
Prestazioni vs Affidabilità	Il sistema gestisce dati piuttosto sensibili, per cui si preferisce che vi siano più controlli di input e consistenza, rallentando possibilmente i tempi di risposta.

1.2 Linee guida per la documentazione dell'interfaccia

Le linee guida includono una lista di regole che gli sviluppatori dovrebbero rispettare durante la progettazione delle interfacce. Per la loro costruzione si è fatto riferimento alla convenzione java nota come Sun Java Coding Conventions [Sun, 2009].

1.3 Definizioni, acronimi, e abbreviazioni

Vengono riportati di seguito alcune definizioni presenti nel documento:

- **Package**: raggruppamento di classi, interfacce o file correlati;
- **Design pattern**: template di soluzioni a problemi ricorrenti impiegati per ottenere riuso e flessibilità;
- **Javadoc**: sistema di documentazione offerto da Java, che viene generato sottoforma di interfaccia in modo da rendere la documentazione accessibile e facilmente leggibile.

1.4 Riferimenti

Di seguito un elenco di altre documentazioni utili che sono state alla base dello sviluppo del progetto e quindi di questo documento:

- **Statement Of Work;**
- **Business Case;**
- **Requirements Analysis Document;**
- **System Design Document;**
- **Test Plan.**

2. Packages

In questa sezione viene mostrata la suddivisione del sistema in Packages. Tale suddivisione è motivata dalle scelte architetturali prese e ricalca la struttura di directory standard definita da Maven, Springboot e Thymeleaf.

2.1 Struttura Directory

- **.idea**: cartella di base di un progetto creato con Intellij Idea
- **.mvn**: contiene tutti i file di configurazione per Maven
- **src**: contiene tutti i file sorgente
    - **ai**: contiene i file python necessari al collegamento e uso del modulo di IA
    - **db**: contiene i file .sql usati per inizializzare il Database MySQL
    - **main**:
        - **java**: contiene le classi Java relative alla logica di business, gestione eventi e persistenza
            - **com.is.findyourplace**
                - **configuration**: contiene le configurazioni degli accessi alle varie risorse del sito Web
                - **controller**: contiene le classi Controller per ogni sottosistema
                    - **gestioneAmministratori**: contiene i Controller che gestiscono le azioni effettuate dagli amministratori
                    - **gestioneRicerca**: contiene i Controller che gestiscono le azioni collegate ad una ricerca del luogo migliore
                    - **gestioneUtenza**: contiene i Controller che gestiscono le azioni effettuate dagli utenti collegate al proprio account
                - **persistence**: contiene tutte le classi utili al collegamento con il database, alla rappresentazione dei dati e al passaggio dei dati tra Client e Server
                    - **dto**: classi usate per passare dati tra Client e Server, qualora le entità non risultassero perfettamente adeguate
                    - **entity**: rappresentano ogni Entità / Tabella all'interno del Database. Ogni entity è gestita dal sistema JPA
                        - **CompositeKeys**: contiene le chiavi composte usate nelle entity che le richiedono
                - **service**: contiene le classi Service per ogni sottosistema
                    - **gestioneAmministratori**: contiene i Service che gestiscono le azioni effettuate dagli amministratori

        - **gestioneRicerca**: contiene i Service che gestiscono le azioni collegate ad una ricerca del luogo migliore
        - **gestioneUtenza**: contiene i Service che gestiscono le azioni effettuate dagli utenti collegate al proprio account
    - **resources**: contiene i file relativi al frontend
        - **static**
            - **css**: contiene tutti gli Style usati dalle pagine
            - **images**: contiene le immagini usate sul sito
            - **js**: contiene tutti gli Script JS usati nelle pagine
        - **templates**: contiene i file HTML renderizzati da Thymeleaf
            - **account**: file HTML relative all' account di un utente
            - **admin**: file HTML usati nelle operazioni da amministratore
            - **ricerca**: file HTML relativi alle ricerche
            - **fragments**: contiene i componenti riusati in un sito, come header e footer
- **test**
    - **java**: contiene le classi Java per l'implementazione del testing
        - **com.is.findyourplace**
            - **System**: contiene I test creati facendo uso di Selenium IDE
            - **Unit**: contiene i test fatti imitando il funzionamento dei vari collegamenti al Database, facendo uso di Mocking, sui Controller e Service
    - **resources**: file di proprietà usato nei test
- **target**: contiene tutti i file prodotti dal sistema di build di Maven

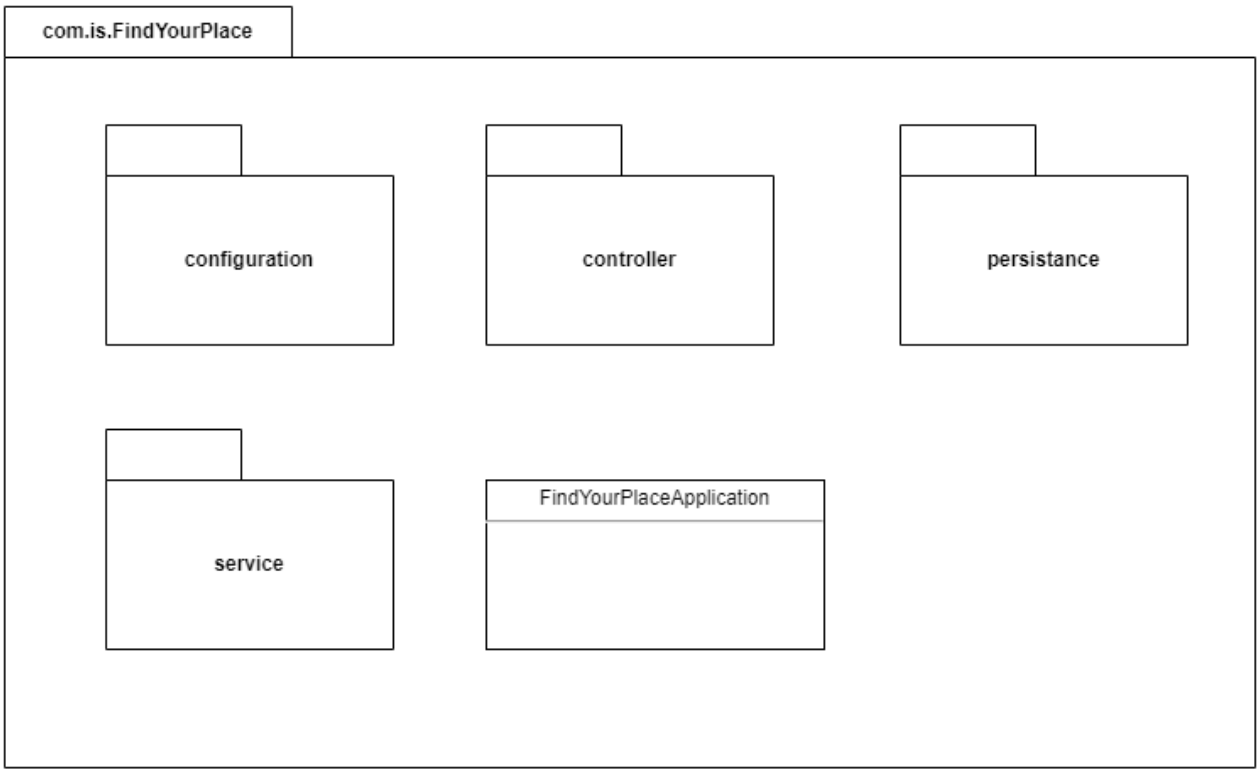
2.2 Struttura Packages

Package FindYourPlace

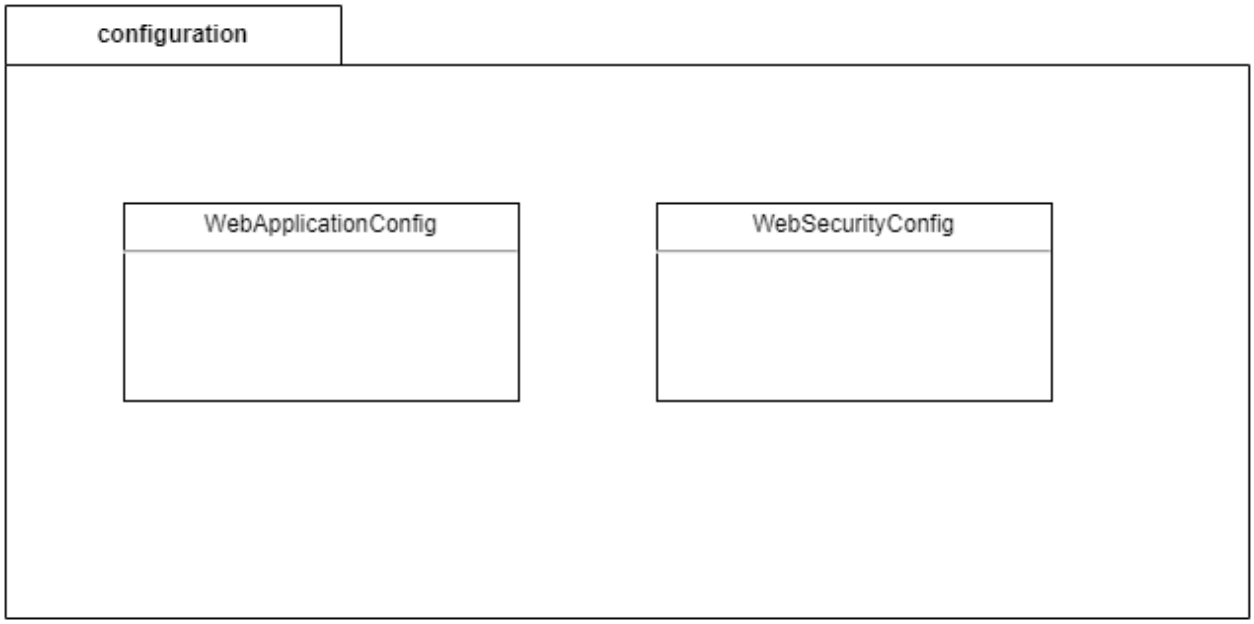
Nella presente sezione si mostra la struttura del package principale di FindYourPlace.
La struttura generale è stata ottenuta a partire dalle seguenti scelte principali:

1. Creare un Package separato per ogni sottosistema, contenente le classi Service e Controller del sottosistema, ed eventuali classi di utilità usate unicamente da esso;
2. Creare un Package separato per le classi del Persistence, contenente le classi Entity, Repository e Dto.

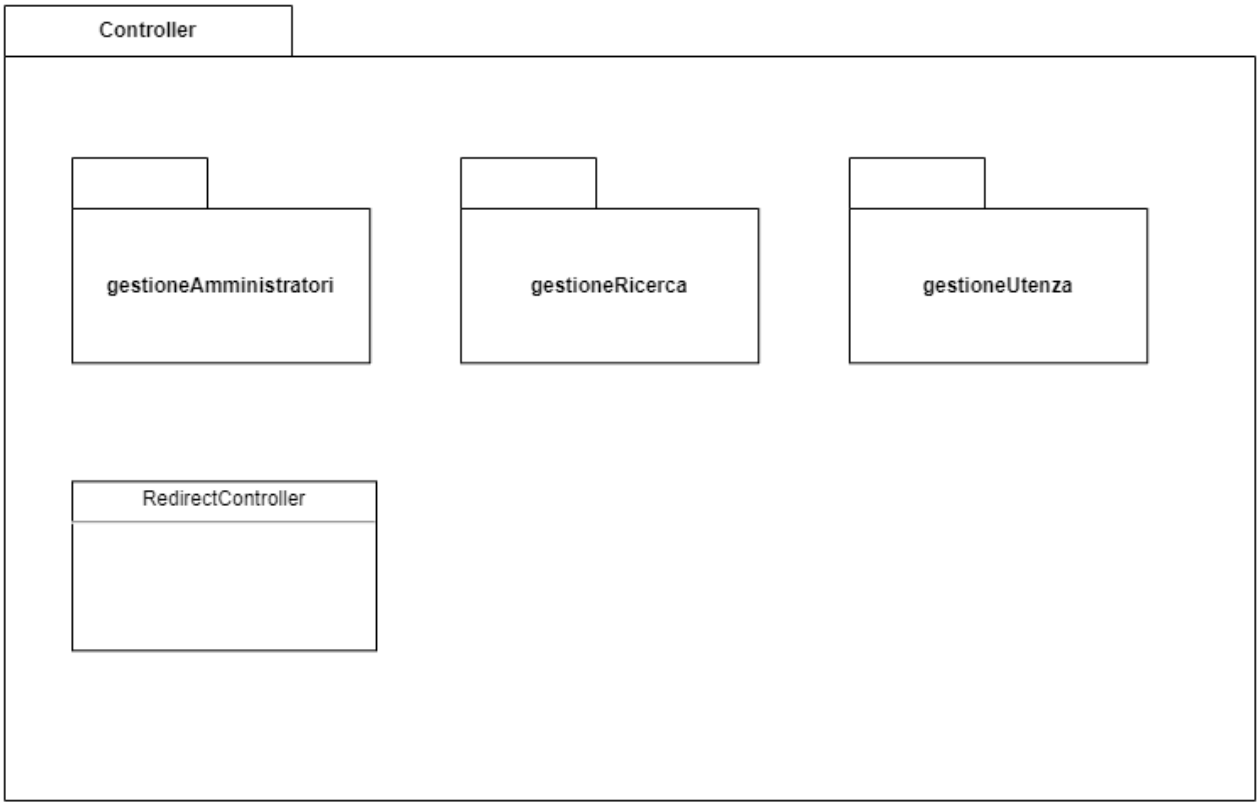
Si illustra la struttura del package generale di progetto con il seguente diagramma:



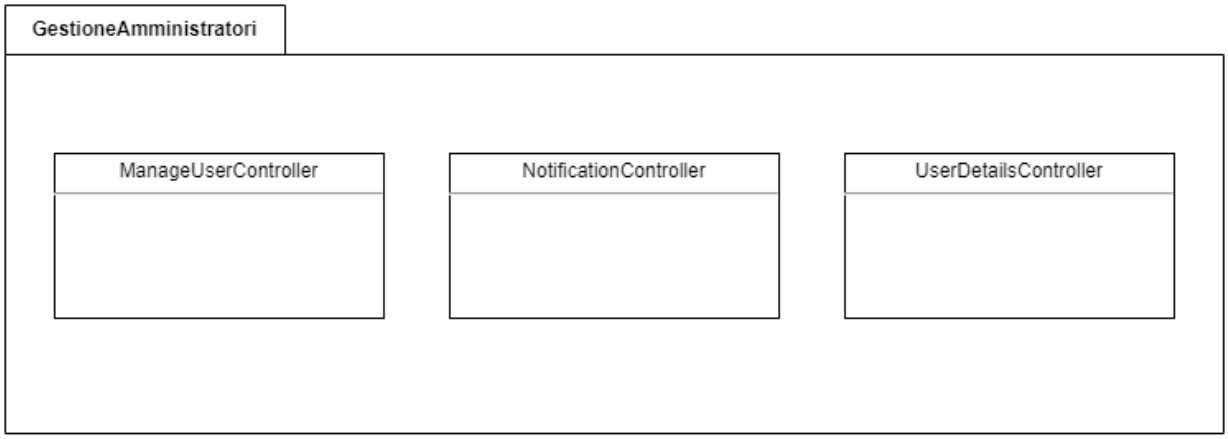
Package Configuration



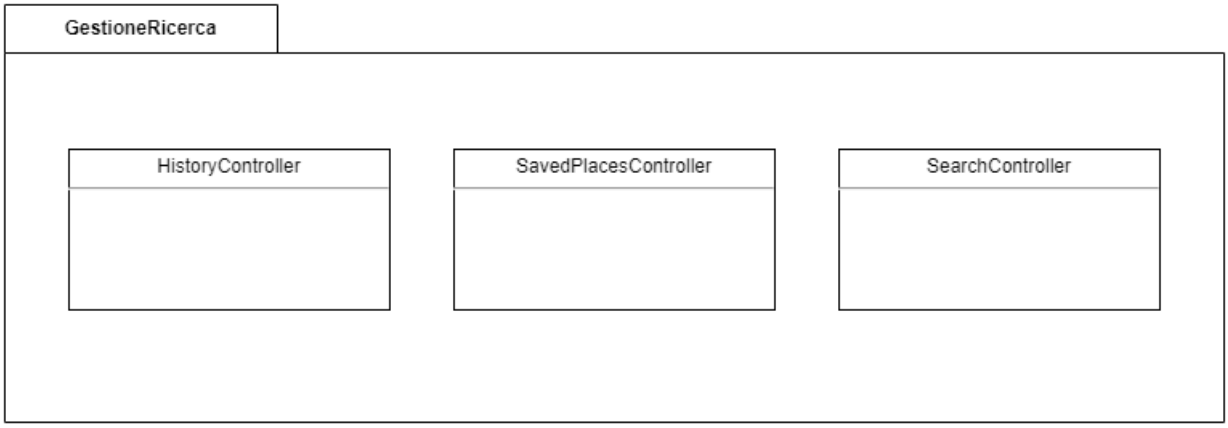
Package Controller



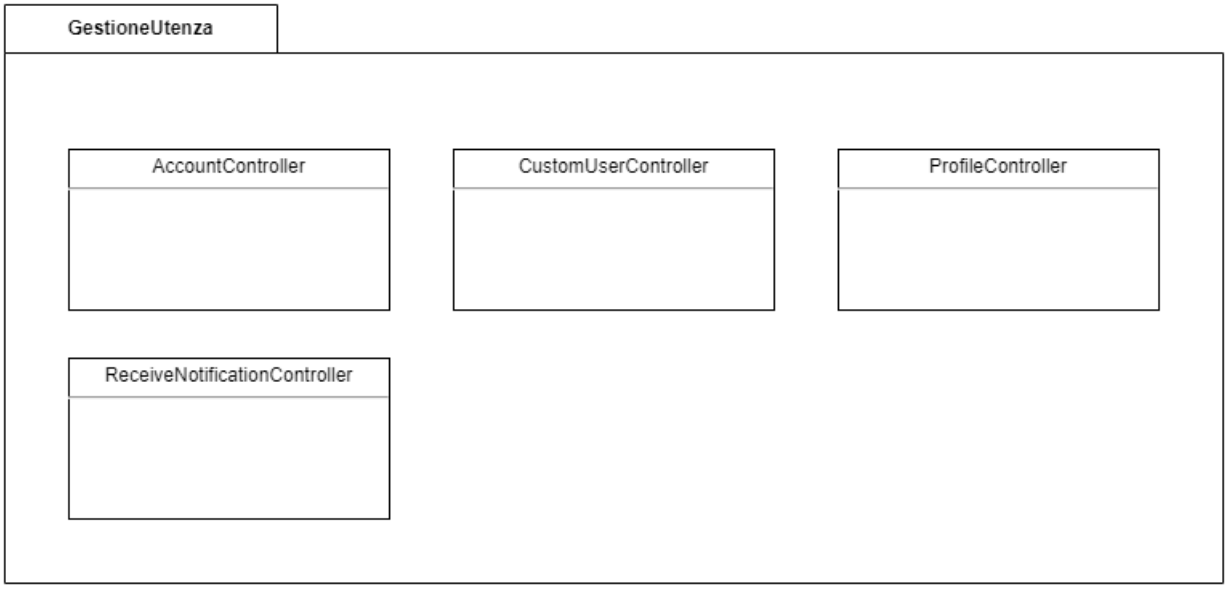
Package GestioneAmministratori (Controller)



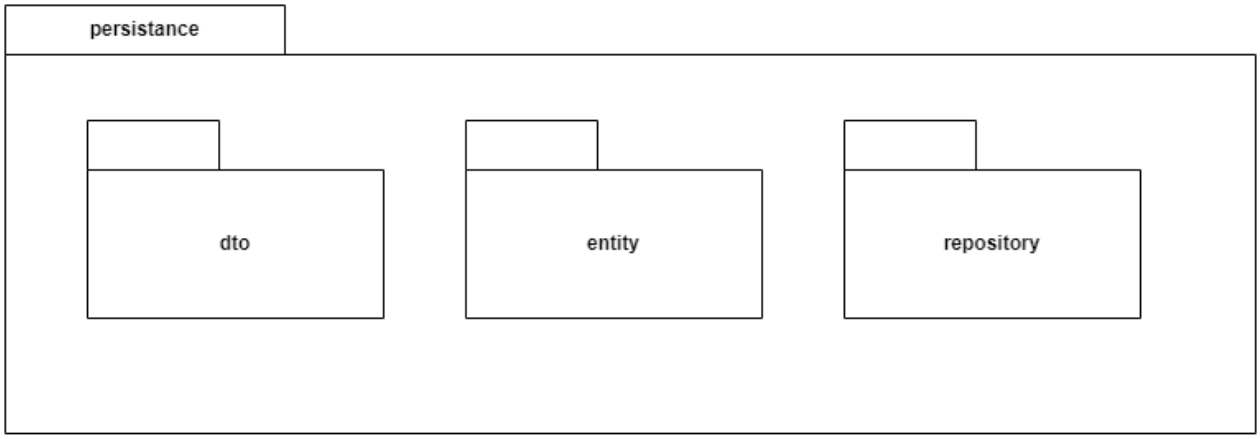
Package GestioneRicerca (Controller)



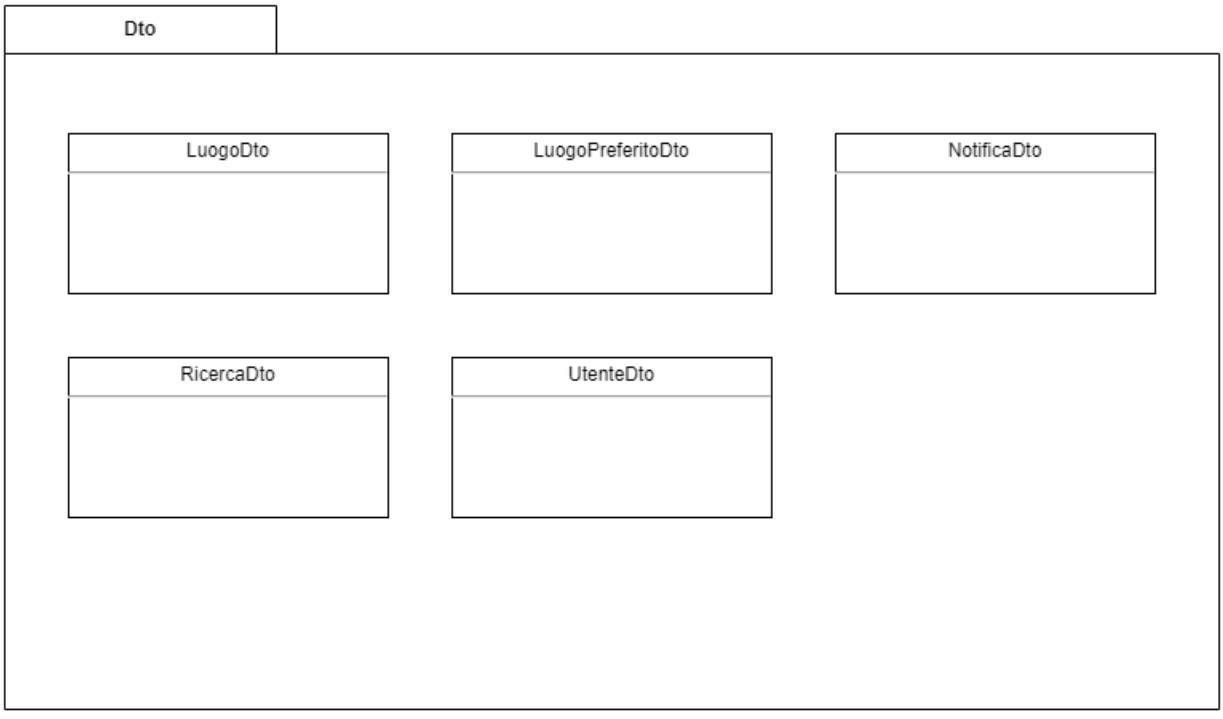
Package GestioneUtenza (Controller)



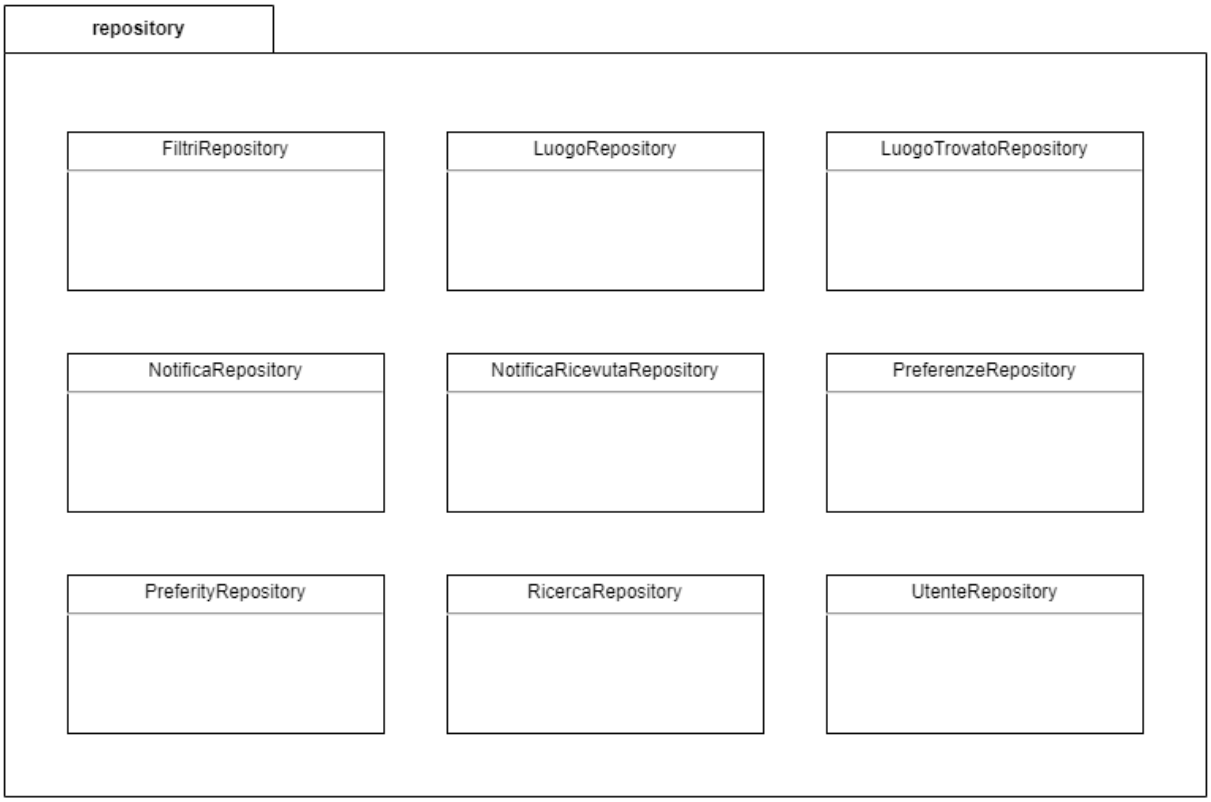
Package Persistance



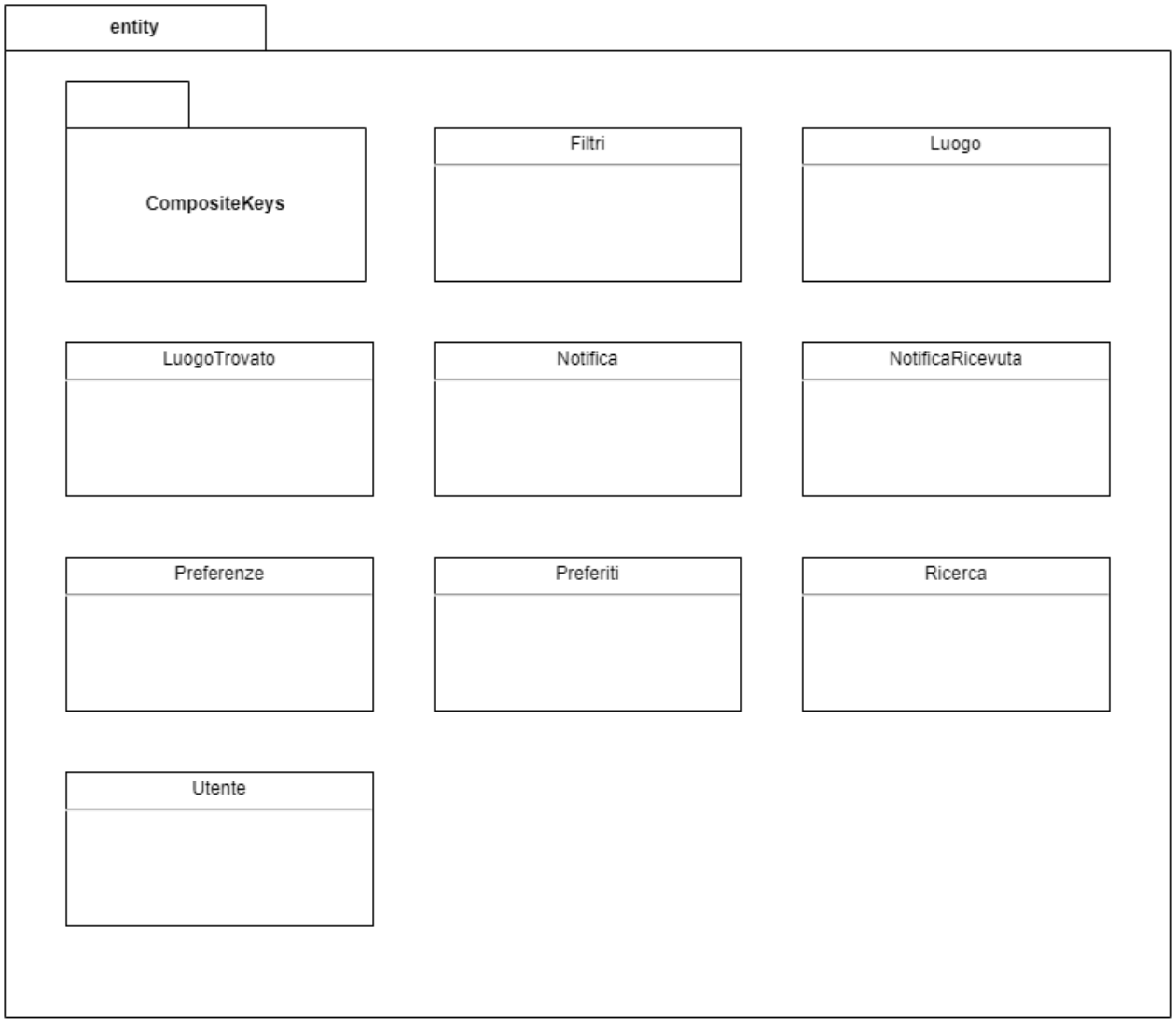
Package Dto



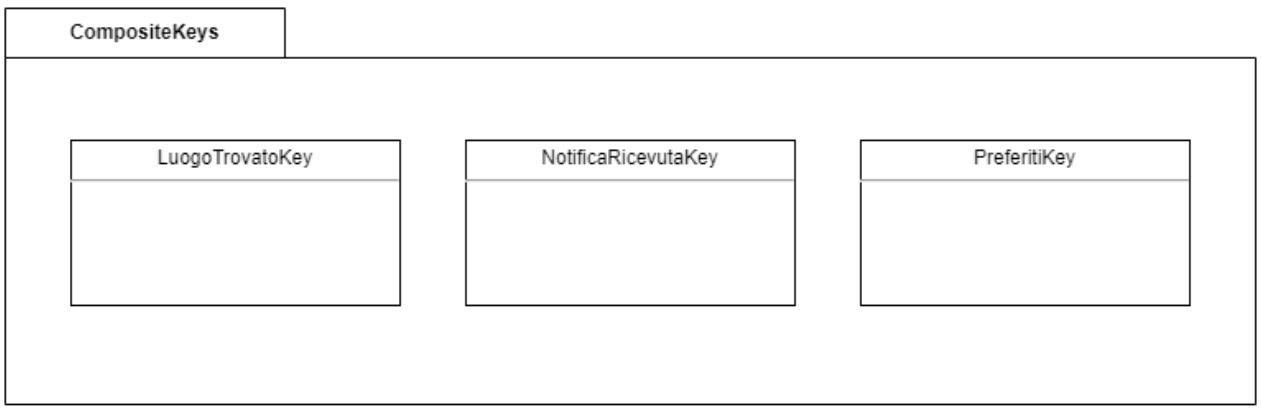
Package repository



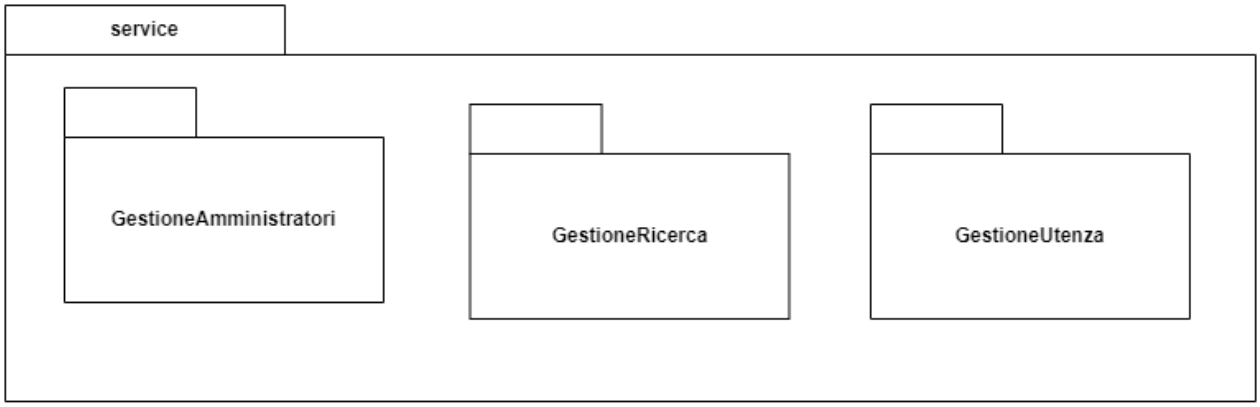
Package Entity



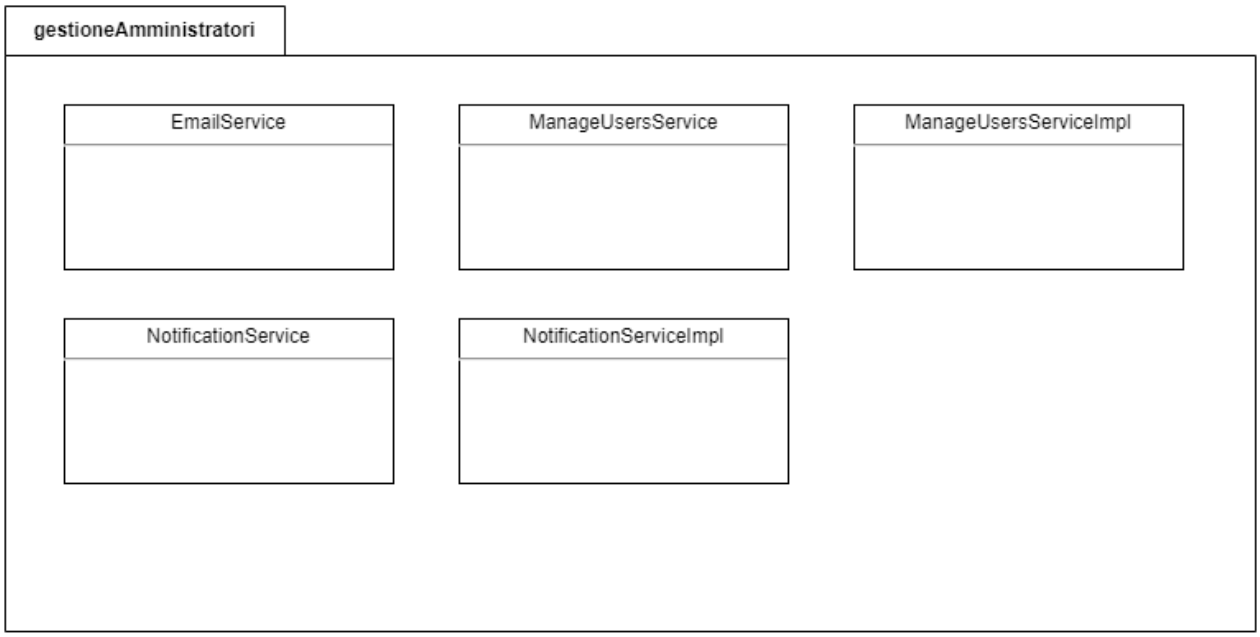
Package CompositeKeys



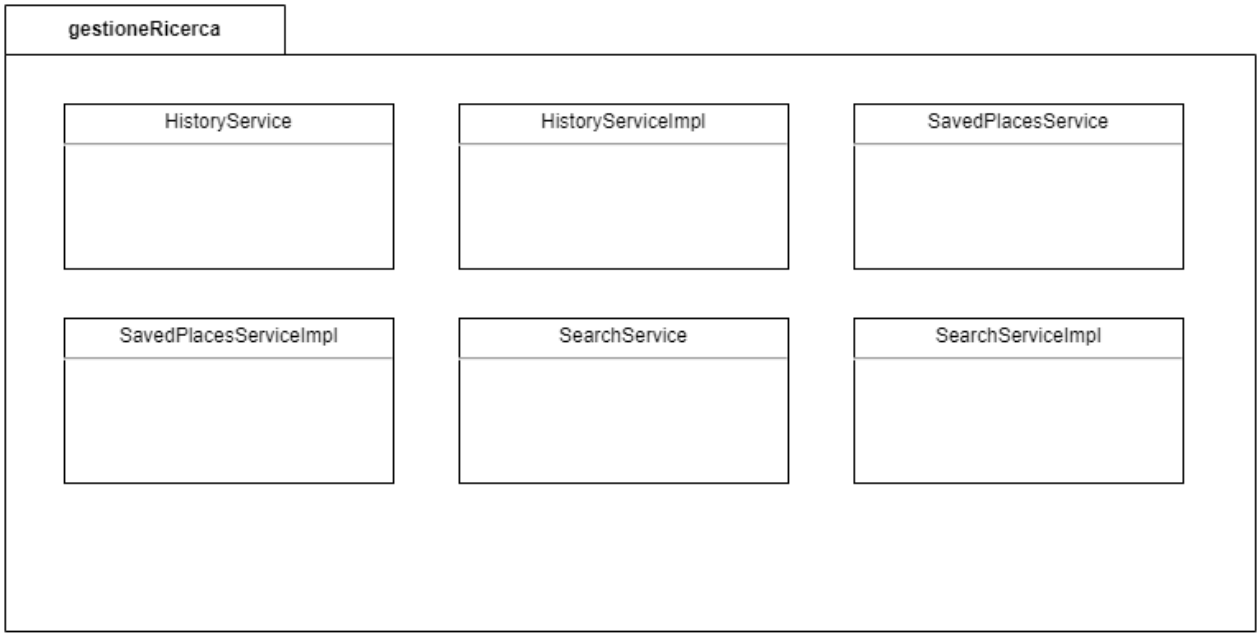
Package Service



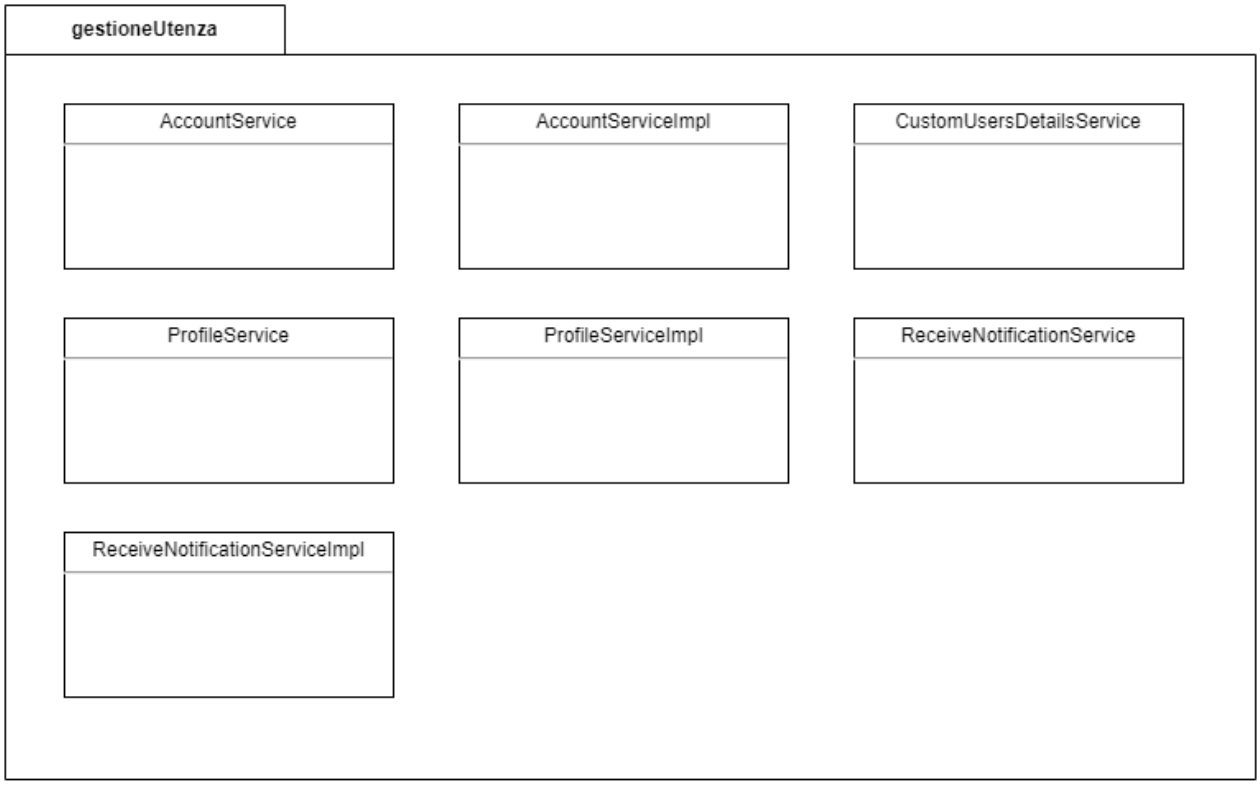
Package GestioneAmministratori (Service)



Package GestioneRicerca (Service)



Package GestioneUtenza (Service)



3. Class Interfaces

Per una più completa ed intuitiva visualizzazione delle classi del sistema FindYourPlace, si consiglia di consultare la documentazione **Javadoc** presente a questo link: FindYourPlace Javadoc

Package GestioneAmministratori (Service)

Nome Classe	EmailService
Descrizione	Permette di gestire le operazioni relative all'invio automatico delle email all'azione di reset username o password di un utente da parte di un amministratore.
Metodi	+ sendEmail(final String toEmail, final String subject, final String message): void
Invariante d classe	n/a

Nome Metodo	+ sendEmail(final String toEmail, final String subject, final String message): void
Descrizione	Permette di inviare una email.
Pre-condizione	Context: ManageUsersController:: sendEmail(Utente.email, subject, message) Pre: Utente.email, subject, message <> null
Post-condizione	n/a

Nome Classe	ManageUsersService
Descrizione	Permette di gestire le operazioni relative alla gestione degli utenti.
Metodi	+ findByUsernameOrEmail(String username): UtenteDto + findAllUtenti(): List<UtenteDto> + updateUtenteUsername(Utente utente): void + updareUtentePassword(Utente utente, String password): void
Invariante d classe	n/a

Nome Metodo	+ findByUsernameOrEmail(String username): UtenteDto
Descrizione	Trova utente, mappandolo in UtenteDto, tramite username o email.
Pre-condizione	Context: ManageUsersController:: findByUsernameOrEmail(username) Pre: username <> null
Post-condizione	Context: ManageUsersController:: findByUsernameOrEmail(username) Post: UtenteDto <> null

Nome Metodo	+ findAllUtenti(): List<UtenteDto>
Descrizione	Restituisce la lista di tutti gli utenti, mappati in UtenteDto.
Pre-condizione	n/a
Post-condizione	Context: ManageUsersController:: findAllUtenti () Post: List<UtenteDto> <> null

Nome Metodo	+ updateUtenteUsername(Utente utente): void
Descrizione	Aggiorna l'username di un utente.
Pre-condizione	Context: ManageUsersController:: updateUtenteUsername (username) Pre: utente <> null
Post-condizione	n/a

Nome Metodo	+ updareUtentePassword(Utente utente, String password): void
Descrizione	Aggiorna la password di un utente.
Pre-condizione	Context: ManageUsersController:: updareUtentePassword (utente, password) Pre: utente, password <> null
Post-condizione	n/a

Nome Classe	NotificationService
Descrizione	Permette di gestire le operazioni relative all'invio notifiche.
Metodi	+ saveNotifica(NotificaDto notificaDto): void + saveNotificaBroadcast(NotificaDto notificaDto): void
Invariante d classe	n/a

Nome Metodo	+ saveNotifica(NotificaDto notificaDto): void
Descrizione	Salva notifica verso un singolo Utente.
Pre-condizione	Context: NotificationController:: saveNotifica (notificaDto) Pre: notificaDto <> null
Post-condizione	n/a

Nome Metodo	+ saveNotificaBroadcast (NotificaDto notificaDto): void
Descrizione	Salva notifica verso tutti gli utenti.
Pre-condizione	Context: NotificationController:: saveNotificaBroadcast (notificaDto) Pre: notificaDto <> null
Post-condizione	n/a

Package GestioneRicerca (Service)

Nome Classe	HistoryService
Descrizione	Permette di gestire le operazioni relative alla cronologia.
Metodi	+ findRicercheDtoByIdUtente(Long idUtente): List<RicercaDto> + findRicerca(Long idRicerca, Long idUtente): Ricerca + removeIdUtente(Ricerca ricerca): void
Invariante d classe	n/a

Nome Metodo	+ findRicercheDtoByIdUtente(Long idUtente): List<RicercaDto>
Descrizione	Query per recuperare la cronologia delle ricerche di un determinato utente tramite il suo id.
Pre-condizione	Context: HistoryController:: findRicercheDtoByIdUtente (idUtente) Pre: idUtente <> null
Post-condizione	Context: HistoryController:: findRicercheDtoByIdUtente (idUtente) Post: List<RicercaDto> <> null

Nome Metodo	+ findRicerca(Long idRicerca, Long idUtente): Ricerca
Descrizione	Query per recuperare una ricerca tramite l'id dell'utente e l'id della ricerca.
Pre-condizione	Context: HistoryController:: findRicerca (idRicerca, idUtente) Pre: idRicerca, idUtente <> null
Post-condizione	Context: HistoryController:: findRicerca (idRicerca, idUtente) Post: Ricerca <> null

Nome Metodo	+ removeIdUtente(Ricerca ricerca): void
Descrizione	Query per rimuovere l'id di un utente da una ricerca.
Pre-condizione	Context: HistoryController:: removeIdUtente (ricerca) Pre: ricerca <> null
Post-condizione	n/a

Nome Classe	SavedPlacesService
Descrizione	Permette di gestire le operazioni relative ai preferiti.
Metodi	+ savePreferito(Utente utente, Luogo luogo): void + findLuoghiPreferitiDtoByIdUtente(Long idUtente): List<LuogoPreferitoDto> + findPreferito(Long idUtente, Long idLuogo): Preferiti + deletePreferito(Preferiti luogoSalvato): void + updateNotPreferito(Preferiti preferito, boolean notifiche): void + findLuogoById(Long idLuogo): Luogo
Invariante d classe	n/a

Nome Metodo	+ savePreferito(Utente utente, Luogo luogo): void
Descrizione	Salva un luogo tra i preferiti fi un utente.
Pre-condizione	Context: SavedPlacesController:: savePreferito (utente, luogo) Pre: utente, luogo <> null
Post-condizione	n/a

Nome Metodo	+ findLuoghiPreferitiDtoByIdUtente(Long idUtente): List<LuogoPreferitoDto>
Descrizione	Query per recuperare la lista dei luoghi preferiti di un utente.
Pre-condizione	Context: SavedPlacesController:: findLuoghiPreferitiDtoByIdUtente (idUtente) Pre: idUtente <> null
Post-condizione	Context: SavedPlacesController:: findLuoghiPreferitiDtoByIdUtente (idUtente) Post: List<LuogoPreferitoDto> <> null

Nome Metodo	+ findPreferito(Long idUtente, Long idLuogo): Preferiti
Descrizione	Query per recuperare il luogo preferito tramite il suo id e quello dell'utente.
Pre-condizione	Context: SavedPlacesController:: findPreferito (idUtente, idLuogo) Pre: idUtente, idLuogo <> null
Post-condizione	Context: SavedPlacesController:: findPreferito (idUtente, idLuogo) Post: Preferiti <> null

Nome Metodo	+ deletePreferito(Preferiti luogoSalvato): void
Descrizione	Query per rimuovere un luogo dai preferiti.
Pre-condizione	Context: SavedPlacesController:: deletePreferito (luogoSalvato) Pre: luogoSalvato <> null
Post-condizione	n/a

Nome Metodo	+ updateNotPreferito(Preferiti preferito, boolean notifiche): void
Descrizione	Query per la gestione delle notifiche di un luogo preferito.
Pre-condizione	Context: SavedPlacesController:: updateNotPreferito (preferito, notifiche) Pre: preferito, notifiche <> null
Post-condizione	n/a

Nome Metodo	+ findLuogoById(Long idLuogo): Luogo
Descrizione	Query per recuperare il luogo tramite il suo id.
Pre-condizione	Context: SavedPlacesController:: findLuogoById (idLuogo) Pre: idLuogo <> null
Post-condizione	Context: SavedPlacesController:: findLuogoById (idLuogo) Post: Luogo <> null

Nome Classe	SearchService
Descrizione	Permette di gestire le operazioni relative ai preferiti.
Metodi	+ saveRicerca(RicercaDto ricercaDto): Long + saveLuogoDto(LuogoDto luogoDto): void + findLuoghiByIdRicerca(Long idRicerca): List<LuogoDto> + findFiltriByIdRicerca(Long idRicerca): Filtri
Invariante d classe	n/a

Nome Metodo	+ saveRicerca(RicercaDto ricercaDto): Long
Descrizione	Salva una Ricerca.
Pre-condizione	Context: SearchController:: saveRicerca (ricercaDto) Pre: ricercaDto <> null
Post-condizione	Context: SearchController:: saveRicerca (ricercaDto) (idLuogo) Post: Long <> null

Nome Metodo	+ saveLuogoDto(LuogoDto luogoDto): void
Descrizione	Salva un Luogo.
Pre-condizione	Context: SearchController:: saveLuogoDto (luogoDto) Pre: luogoDto <> null
Post-condizione	n/a

Nome Metodo	+ findLuoghiByIdRicerca(Long idRicerca): List<LuogoDto>
Descrizione	Trova lista di luoghi Dto tramite idRicerca.
Pre-condizione	Context: SearchController:: findLuoghiByIdRicerca (idRicerca) Pre: idRicerca <> null
Post-condizione	Context: SearchController:: findLuoghiByIdRicerca (idRicerca) Post: List<LuogoDto> <> null

Nome Metodo	+ findFiltriByIdRicerca(Long idRicerca): Filtri
Descrizione	Trova i Filtri di una Ricerca.
Pre-condizione	Context: SearchController:: findFiltriByIdRicerca (idRicerca) Pre: idRicerca <> null
Post-condizione	Context: SearchController:: findFiltriByIdRicerca (idRicerca) Post: Filtri <> null

Package GestioneUtenza (Service)

Nome Classe	AccountService
Descrizione	Permette di gestire le operazioni relative ai preferiti.
Metodi	+ saveUtente(UtenteDto utenteDto): void + findByUsernameOrEmail(String username): Utente + findByUsername(String username): UtenteDto + existsByUsername(String username): Boolean + existsByEmail(String email): boolean
Invariante d classe	n/a

Nome Metodo	+ saveUtente(UtenteDto utenteDto): void
Descrizione	Crea un nuovo Utente.
Pre-condizione	Context: AccountController:: saveUtente (utenteDto) Pre: utenteDto <> null
Post-condizione	n/a

Nome Metodo	+ findByUsernameOrEmail(String username): Utente
Descrizione	Trova utente tramite username o email.
Pre-condizione	Context: AccountController:: findByUsernameOrEmail (username) Pre: username <> null
Post-condizione	Context: AccountController:: findByUsernameOrEmail (username) Post: Utente <> null

Nome Metodo	+ findByUsername(String username): UtenteDto
Descrizione	Trova utente, mappandolo in UtenteDto, tramite username.
Pre-condizione	Context: AccountController:: findByUsername (username) Pre: username <> null
Post-condizione	Context: AccountController:: findByUsername (username) Post: UtenteDto <> null

Nome Metodo	+ existsByUsername(String username): Boolean
Descrizione	Controlla se esiste un Utente tramite l'username.
Pre-condizione	Context: AccountController:: existsByUsername (username) Pre: username <> null
Post-condizione	Context: AccountController:: findByUsername (username) Post: Boolean = null

Nome Metodo	+ existsByEmail(String email): boolean
Descrizione	Controlla se esiste un Utente tramite l'email.
Pre-condizione	Context: AccountController:: existsByEmail (email) Pre: email <> null
Post-condizione	Context: AccountController:: existsByEmail (email) Post: Boolean = null

Nome Classe	CustomUserDetailsService
Descrizione	Permette di gestire le operazioni relative al caricamento di un utente dal database per l'autenticazione.
Metodi	+ loadUserByUsername(final String username): UserDetails
Invariante d classe	n/a

Nome Metodo	+ loadUserByUsername(final String username): UserDetails
Descrizione	Carica un utente dal database e lo adatta al metodo di autenticazione utilizzato (Spring Boot Security).
Pre-condizione	Context: CustomUserDetails:: loadUserByUsername (username) Pre: username <> null
Post-condizione	n/a

Nome Classe	ProfileService
Descrizione	Permette di gestire le operazioni relative alla modifica del proprio profilo utente.
Metodi	+ updateUtente(UtenteDto utenteDto): void + findPrefByUtente(Utente utente): Preferenze + createPreferenze(Utente utente): Preferenze + updatePreferenze(Preferenze preferenze): void
Invariante d classe	n/a

Nome Metodo	+ updateUtente(UtenteDto utenteDto): void
Descrizione	Aggiorna dati Utente.
Pre-condizione	Context: ProfileController:: updateUtente (utenteDto) Pre: utenteDto <> null
Post-condizione	n/a

Nome Metodo	+ findPrefByUtente(Utente utente): Preferenze
Descrizione	Trova Preferenze di un Utente.
Pre-condizione	Context: ProfileController:: findPrefByUtente (utente) Pre: utente <> null
Post-condizione	Context: ProfileController:: findPrefByUtente (utente) Post: Preferenze <> null

Nome Metodo	+ createPreferenze(Utente utente): Preferenze
Descrizione	Crea Preferenze se un Utente non le possiede.
Pre-condizione	Context: ProfileController:: createPreferenze (utente) Pre: utente <> null
Post-condizione	Context: ProfileController:: createPreferenze (utente) Post: Preferenze <> null

Nome Metodo	+ updatePreferenze(Preferenze preferenze): void
Descrizione	Aggiorna le preferenze di un Utente
Pre-condizione	Context: ProfileController:: updatePreferenze (preferenze) Pre: preferenze <> null
Post-condizione	n/a

Nome Classe	ReceiveNotificationService
Descrizione	Permette di gestire le operazioni relative alla ricezione di notifiche.
Metodi	+ findAllNotificheRicevuteByIdUtente(Long id): List<NotificaRicevuta> + findByIdNotifica(long id): NotificaDto + findByIdUtenteAndIdNotifica(long idUtente, long idNotifica): NotificaRicevuta + setRead(NotificaRicevuta notificaRicevuta, boolean isRead): void
Invariante d classe	n/a

Nome Metodo	+ findAllNotificheRicevuteByIdUtente(Long id): List<NotificaRicevuta>
Descrizione	Trova Lista di Notifiche Ricevute di un utente.
Pre-condizione	Context: ReceiveNotificationController:: findAllNotificheRicevuteByIdUtente (id) Pre: id <> null
Post-condizione	Context: ReceiveNotificationController:: findAllNotificheRicevuteByIdUtente (id) Post: List<NotificaRicevuta> <> null

Nome Metodo	+ findByIdNotifica(long id): NotificaDto
Descrizione	Trova NotificaDto tramite il suo id.
Pre-condizione	Context: ReceiveNotificationController:: findByIdNotifica (id) Pre: id <> null
Post-condizione	Context: ReceiveNotificationController:: findByIdNotifica (id) Post: NotificaDto <> null

Nome Metodo	+ findByIdUtenteAndIdNotifica(long idUtente, long idNotifica): NotificaRicevuta
Descrizione	Trova NotificaRicevuta tramite l'Id di un utente e l'Id della notifica.
Pre-condizione	Context: ReceiveNotificationController:: findByIdUtenteAndIdNotifica (idUtente, idNotifica) Pre: idUtente, idNotifica <> null
Post-condizione	Context: ReceiveNotificationController:: findByIdUtenteAndIdNotifica (idUtente, idNotifica) Post: NotificaRicevuta <> null

Nome Metodo	+ setRead(NotificaRicevuta notificaRicevuta, boolean isRead): void
Descrizione	Cambia il valore che indica se una notifica è stata letta.
Pre-condizione	Context: ReceiveNotificationController:: setRead (notificaRicevuta, isRead) Pre: notificaRicevuta, isRead <> null
Post-condizione	Context: ReceiveNotificationController:: setRead (notificaRicevuta, isRead) Post: isRead = null

4. Design Patterns

Il Progetto FindYourPlace fa uso di alcuni Design Patterns per facilitare la creazione ed utilizzo del sistema, in particolare, oltre alle convenzioni già utilizzate per la disposizione dei packages e directory, si fa uso di **Adapter** e **Facade**.

Adapter

L'Adapter è un design pattern strutturale che consente la comunicazione tra oggetti e classi con interfacce differenti, in modo che possano operare insieme nonostante tali interfacce siano incompatibili tra loro.

FindYourPlace fornisce un modulo di Intelligenza Artificiale usato per calcolare l'Indice di Qualità di un luogo ricercato. Questo è implementato con un linguaggio di programmazione diverso da quello utilizzato principalmente nel nostro progetto.

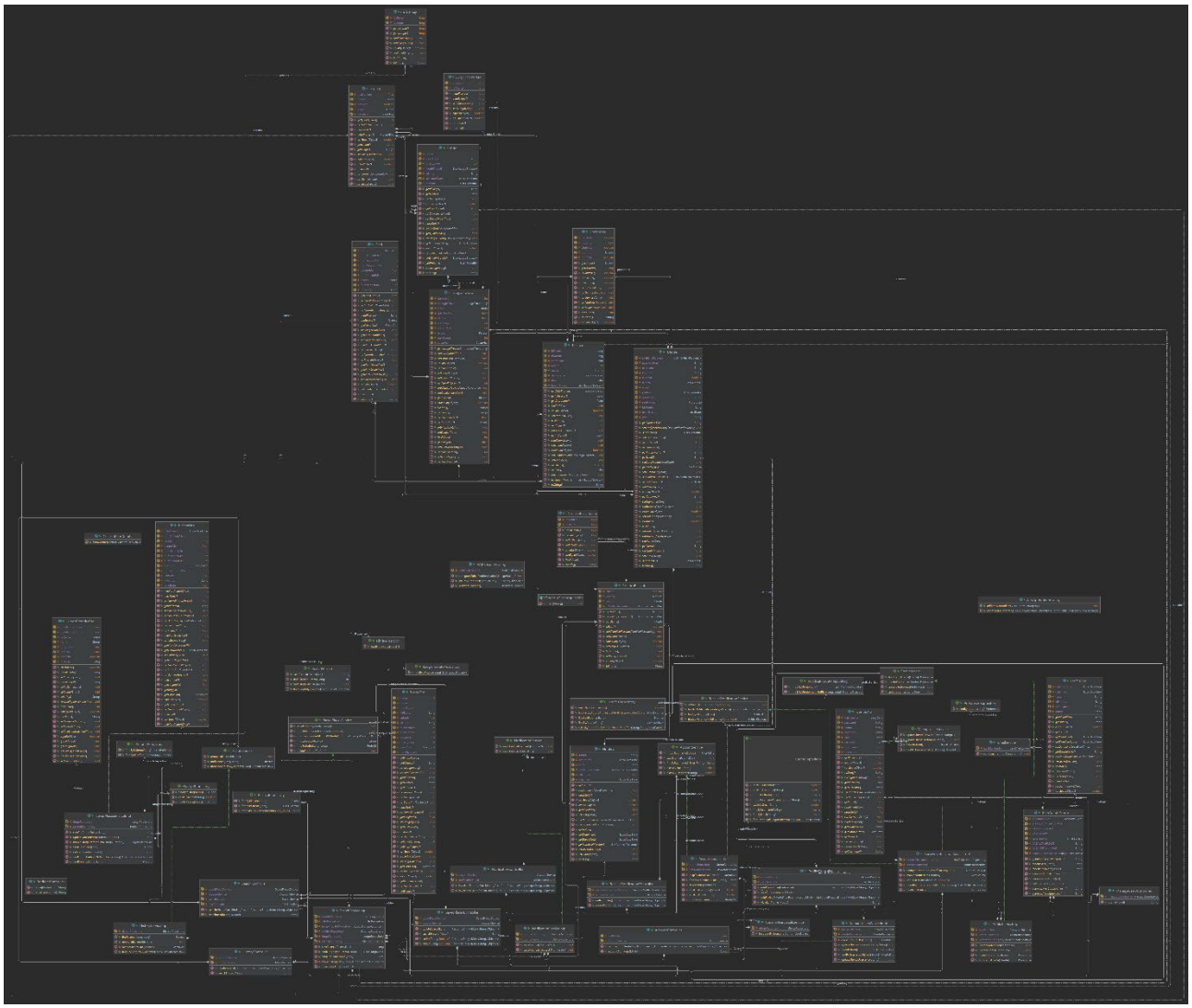
Essendo quindi un modulo esterno, viene utilizzato un server **Flask** come tramite tra il modulo e il sistema FYP. Il modulo si preoccuperà di restituire una lista di luoghi corrispondente ai dati inseriti dall'utente in fase di ricerca, utilizzando un formato standard adatto allo scambio di dati, come il **JSON**.

Facade

Il Facade è un design pattern strutturale che, tramite un'interfaccia semplificata, permette l'accesso a sottosistemi più complessi, così da nascondere al sistema la complessità dell'insieme di oggetti sottostanti, favorendo inoltre la futura manutenibilità dei metodi sviluppati.

In particolare, il sistema FindYourPlace fa uso di Facade nel caso dei Service, in cui ogni classe possiede dapprima un'interfaccia per l'interazione con altri sottosistemi e poi ad ognuna è associata la relativa implementazione, rendendo l'utilizzo più semplice e favorendo una migliore esperienza di manutenzione futura.

5. Class Diagram



6. Glossario

- **Utente**: Persona che interagisce col sistema, la quale può essere sia registrata che non registrata;
- **Amministratore**: Utente con maggiori privilegi, in grado di gestire l'accesso degli utenti generici e inviare loro delle notifiche;
- **Preferenze**: Preferenze aggiuntive rispetto ai dati utili al profilo di un utente, usate per ricerche future.
- **Ricerca**: Form usato per la ricerca di un luogo in base ai parametri inseriti;
- **Luogo**: Luogo trovato da una ricerca, con nome più associabile a questo e indice di qualità calcolato dalla IA;
- **Cronologia Ricerche**: Lista di ricerche effettuate da un utente;
- **Luoghi Preferiti**: Lista di luoghi trovati da ricerche salvati da un utente per ricevere eventuali notifiche su probabili cambiamenti;
- **Notifica**: Notifica ricevuta da un utente, che essa sia inviata dal sistema o da un amministratore;
- **Entity**: Classe di oggetti che rappresenta un concetto del dominio del problema;
- **Control**: Classe di oggetti che gestisce il comportamento del sistema dopo un input, ponendosi come collegamento tra i vari oggetti Boundary usati da una Entity;
- **Package**: Raggruppamento di classi ed interfacce.