GreenTrails

Progetto GreenTrails - Azienda GreenSpireAI
Dipartimento di Informatica - Università degli Studi di Salerno
Progetto GPS/IS A.A. 2023-2024

Object Design Document

GreenTrails

Riferimento	2023_C03_ODD
Versione	1.0
Data	07/02/2024
Destinatario	Prof.ssa Filomena Ferrucci, Prof. Fabio Palomba
Presentato da	C03 GreenTrails TM: Diana Lavinia Cojoc, Ernesto De Iesu, Gabriele Di Stefano, Roberta Galluzzo, Michela Percaccio, Emanuele Setaro
Approvato da	

Revision History

Data	Versione	Descrizione	Autori
17/12/2023	0.1.0	Stesura Design Goals e Trade-off	Roberta Galluzzo, Emanuele Setaro, Michela Percaccio, Diana Lavinia Cojoc
17/12/2023	0.2.0	Stesura COTS, Linee Guida, Riferimenti e Glossario	Gabriele Di Stefano, Ernesto De Iesu
20/12/2023	0.3.0	Stesura Design Patterns	Tutto il Team
23/12/2023	0.4.0	Stesura Packages	Tutto il Team
27/12/2023	0.5.0	Stesura Class Interfaces	Tutto il Team
03/01/2024	0.6.0	Class Diagram	Tutto il Team
05/01/2024	0.7.0	Completamento	Diana Lavinia Cojoc, Michela Percaccio
07/02/2024	1.0	Revisione finale	Tutto il Team

GreenTrails

Progetto GreenTrails - Azienda GreenSpireAI
Dipartimento di Informatica - Università degli Studi di Salerno
Progetto GPS/IS A.A. 2023-2024

Sommario

1. Introduzione

1.1 Object Design Goals

Rank	ID Design Goal	Descrizione design goal	Origine	Autore
2	ODG_1 Tempo di risposta	Il sistema deve garantire un tempo di risposta inferiore a 2 secondi.	DG_12	Gabriele Di Stefano
1	ODG_2 Sicurezza	Il sistema dovrebbe garantire la sicurezza degli utenti assicurandosi la corretta gestione delle autorizzazioni tramite la componente Security del framework Spring.	DG_11	Roberta Galluzzo
3	ODG_3 Spazio di memoria	Il sistema dovrebbe essere in grado, in caso di necessità, di allocare dinamicamente maggiore memoria.	DG_2	Gabriele Di Stefano
4	ODG_4 Estendibilità	Il sistema dovrebbe garantire facilmente l'aggiunta di nuove funzionalità attraverso lo sviluppo di un'architettura modulare.	DG_4	Roberta Galluzzo

1.2 Object Design Trade-off

In questo paragrafo si descriveranno le scelte e i compromessi progettuali maturati dal team. Di seguito, si presenteranno i trade-off individuati.

Buy vs Build: è possibile scegliere di adottare una politica DIY, realizzando per intero tutte le componenti del sistema, al fine di contenere le spese di budget; tuttavia, a causa della ridotta quantità di tempo, è preferibile utilizzare diverse componenti Off-The-Shelf.

Spazio di memoria vs Tempo di risposta: il sistema deve garantire che i tempi di risposta non siano superiori a 2 secondi, ragion per cui potrebbe utilizzare uno spazio di memoria maggiore in caso di necessità.

1.2.1 Componenti Off-The-Shelf

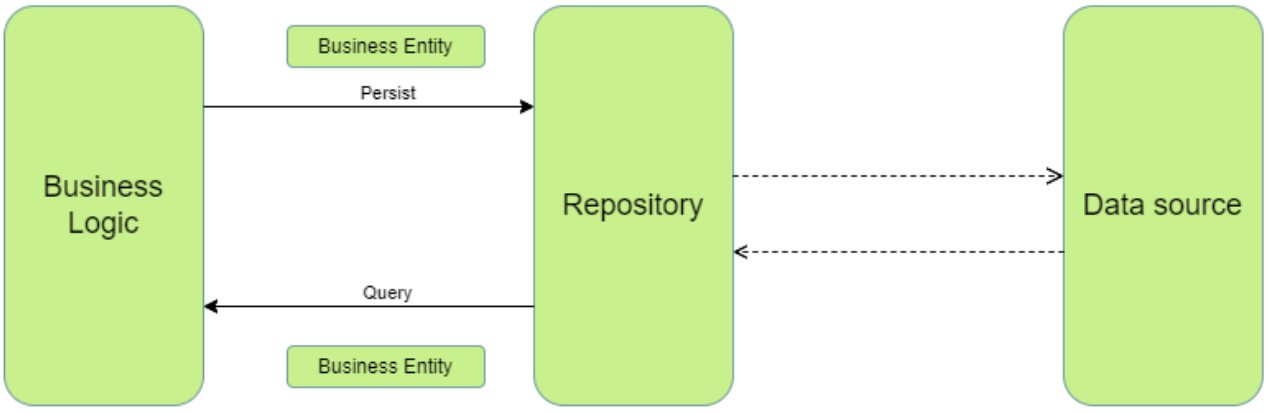
Il sistema, al fine di semplificare il lavoro di implementazione e di garantire il corretto funzionamento di tutti i sottosistemi, utilizzerà le seguenti componenti Off-The-Shelf:

- **Spring**: framework open source per lo sviluppo di applicazioni Java, ne verranno utilizzati i vari moduli (Security, JPA, ecc.) per poter gestire la logica di business;
- **MySQL**: DBMS relazionale, utilizzato per poter gestire i dati persistenti;
- **Stripe**: componente tecnica dei servizi di pagamento, utilizzata per poter gestire con facilità i pagamenti;
- **Angular**: framework open source per lo sviluppo di applicazioni web, utilizzato per la realizzazione dell'interfaccia utente.

1.2.2 Design Pattern

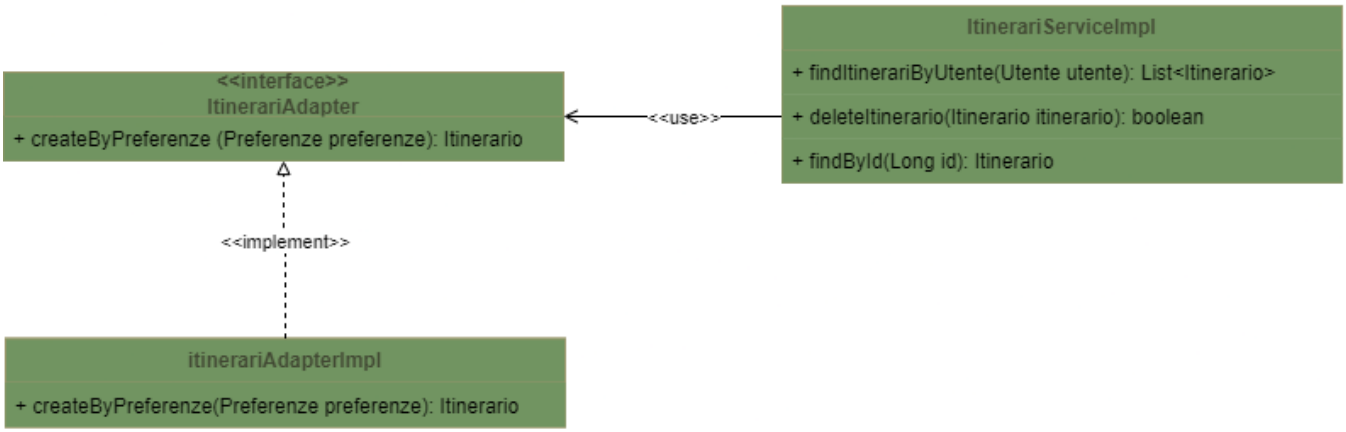
1.2.2.1 Repository

Dopo aver esaminato attentamente la documentazione di Spring e la struttura del nostro sistema, è stata decisa l'implementazione del design pattern "Repository". Questo pattern si concentra sulla separazione della logica di accesso ai dati, associandola alle entità di business nel livello delle Entity. La connessione tra la logica di accesso ai dati e la logica di business avviene attraverso l'uso di interfacce.



1.2.2.2 Adapter

Il design pattern "Adapter" permette di facilitare la collaborazione tra oggetti che avranno differenti interfacce, tramite la creazione di classi "Adapter", utili alla manipolazione dei dati. Inoltre, questo design pattern sarà utilizzato per trasformare i risultati prodotti dal modulo di intelligenza artificiale in dati processabili dal sistema grazie all'incapsulamento. Nel nostro caso il modulo di intelligenza artificiale è atto alla pianificazione di itinerari automatica in base alle preferenze che il visitatore specifica nel questionario presente nella sua area riservata. Quindi con il comando apposito nell'interfaccia del sistema il visitatore ha la possibilità di accedere a questa funzionalità permessa dall'adapter che lavora con l'interfaccia del sistema



1.3 Linee guida per la documentazione dell'interfaccia

Di seguito, è possibile trovare una lista delle regole da seguire durante la scrittura del codice, suddivise in sezioni, ognuna dedicata ai linguaggi che verranno utilizzati.

1.3.1 Java

Il codice Java aderirà alle linee guida fornite dalla convenzione "Google Java Style Guide".

1.3.2 Python

Il codice Python aderirà alle linee guida fornite dalla convenzione "PEP 8".

1.3.3 SQL

Gli script SQL aderiranno alle linee guida fornite dalla convenzione "SQL Style Guide".

1.3.4 Angular (HTML, JavaScript, CSS)

Il codice HTML, JavaScript e CSS - parti del framework Angular - aderirà alle linee guida fornite dalla convenzione "Angular Coding Style Guide".

1.4 Definizioni, acronimi e abbreviazioni

Definizioni: Glossario

Acronimi:

GA = Gestore Attività

UG = Utente Generico

AM = Amministratore

VI = Visitatore

RF = Requisito funzionale

RNF = Requisito Non Funzionale

UC = Use Case

UCD = Use Case Diagram

SC = Scenario

MU = Mock-Up

NA = Nessuna

UI = Mock-up

NP = Navigational Path

CD = Class Diagram

OD = Object Diagram

SD = Sequence Diagram

SCD = Statechart Diagram

FURPS+ = Rappresenta:

- Funzionalità;
- Usabilità;
- Affidabilità;
- Prestazioni;
- Sopportabilità.

Il "+" sta per pseudo-requisiti o vincoli del sistema:

- Implementazione
- Interfaccia
- Operazioni
- Packaging
- Legali

DIY = Do It Yourself
COTS = Componenti Off-The-Shelf

1.5 Riferimenti

- Libro di testo “Object-Oriented Software Engineering (Using UML, Patterns, and Java) terza Edizione di Bernd Bruegge & Allen H. Dutoit”
- Statement Of Work
- Business Case
- Project Charter
- Matrice Di Tracciabilità
- Requirements Analysis Document
- System Design Document

2. Packages

Per facilità di implementazione è stato scelto di descrivere in dettaglio solo i packages del lato backend in modo da avere in principio una linea guida ferrea per la stesura del codice, essendo il lato backend quello più corposo e meno propenso a cambiamenti.

Cartelle lato Backend:

- **.idea** contiene tutti i file di configurazione per l'IDE IntelliJ
- **src** contiene tutti i file del codice sorgente
    - **main**
        - **java** contiene tutte le classi Java relative alle varie componenti del sistema
        - **resources**
    - **test**
        - **java** contiene tutte le classi Java relative al testing
- **target** contiene tutti i file prodotti da Maven durante il processo di build

Cartelle lato Frontend:

- **.idea** contiene tutti i file di configurazione per l'IDE IntelliJ
- **node_modules** contiene tutti i moduli utilizzati dall'applicazione
- **src** contiene tutti i file del codice sorgente

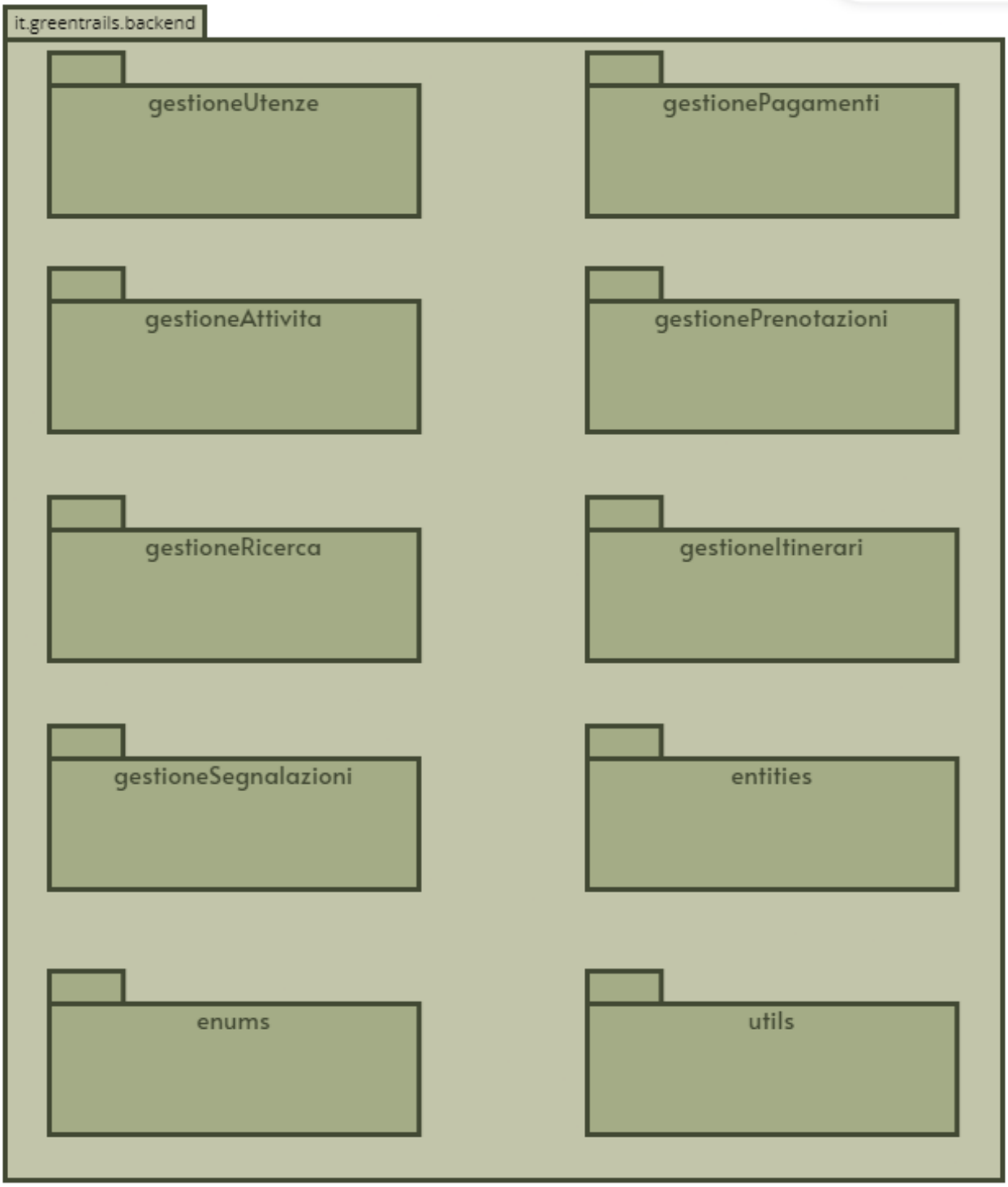
2.1 Package GreenTrails

In questo paragrafo è illustrata la struttura del package principale “GreenTrails”.

Si noti che la suddivisione in packages è stata eseguita creando un package per ogni sottosistema, all'interno dei quali saranno presenti i sotto-packages “Repository”, “Service”, “Controller”, con le relative classi Repository, Service e Controller. Il sotto-package “Controller” gestirà la comunicazione tra “Service” e client, mentre il sotto-package “Repository” sarà responsabile della comunicazione tra il sotto-package “Service” e il database.

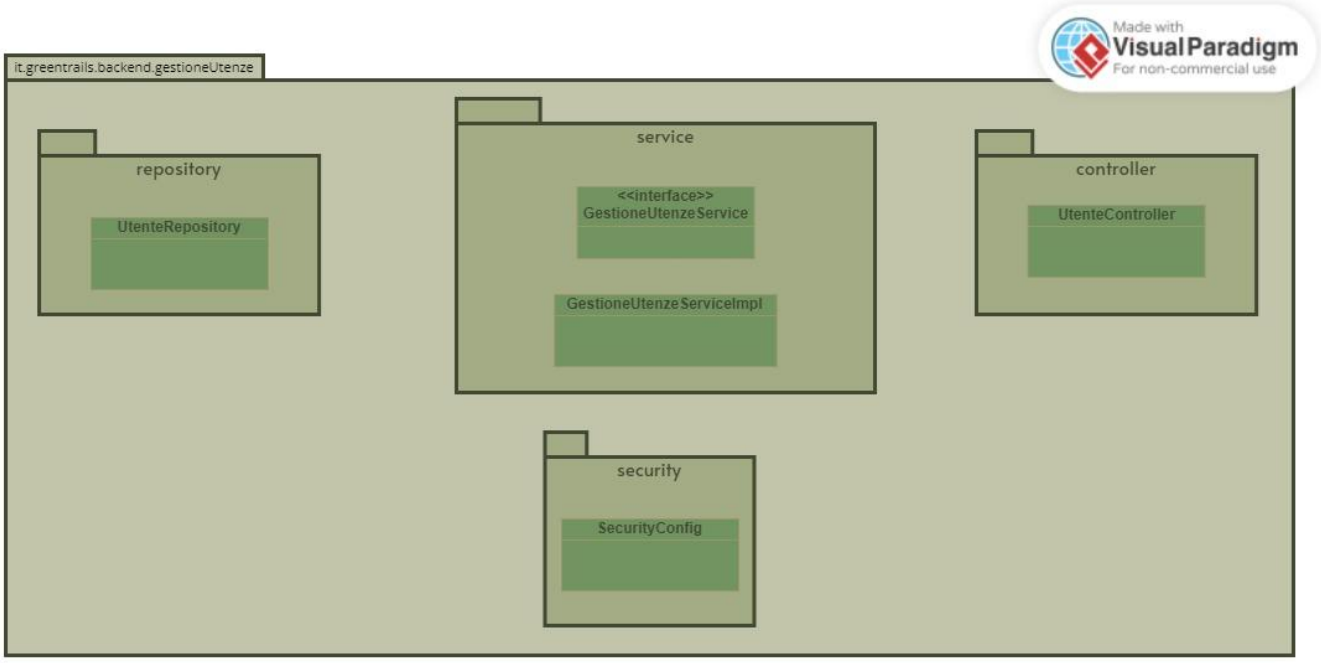
Infine, sono stati aggiunti i seguenti packages:

- “entities”: conterrà tutte le entità del sistema;
- “enums": conterrà tutte le enumerations che verranno assegnate alle entità;
- “utils”: conterrà tutte le classi che forniranno le utility comuni ad ogni sottosistema.



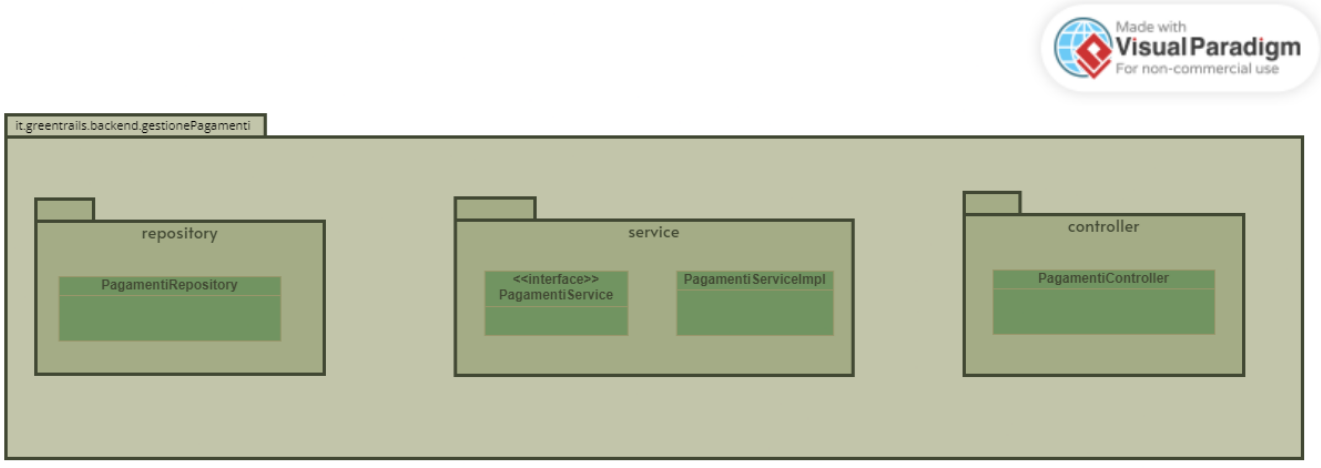
2.2 Package GestioneUtenze

In questo package saranno collocate tutte le classi che formano il sottosistema della gestione delle utenze. Inoltre, qui verranno gestiti i permessi e l'autenticazione.



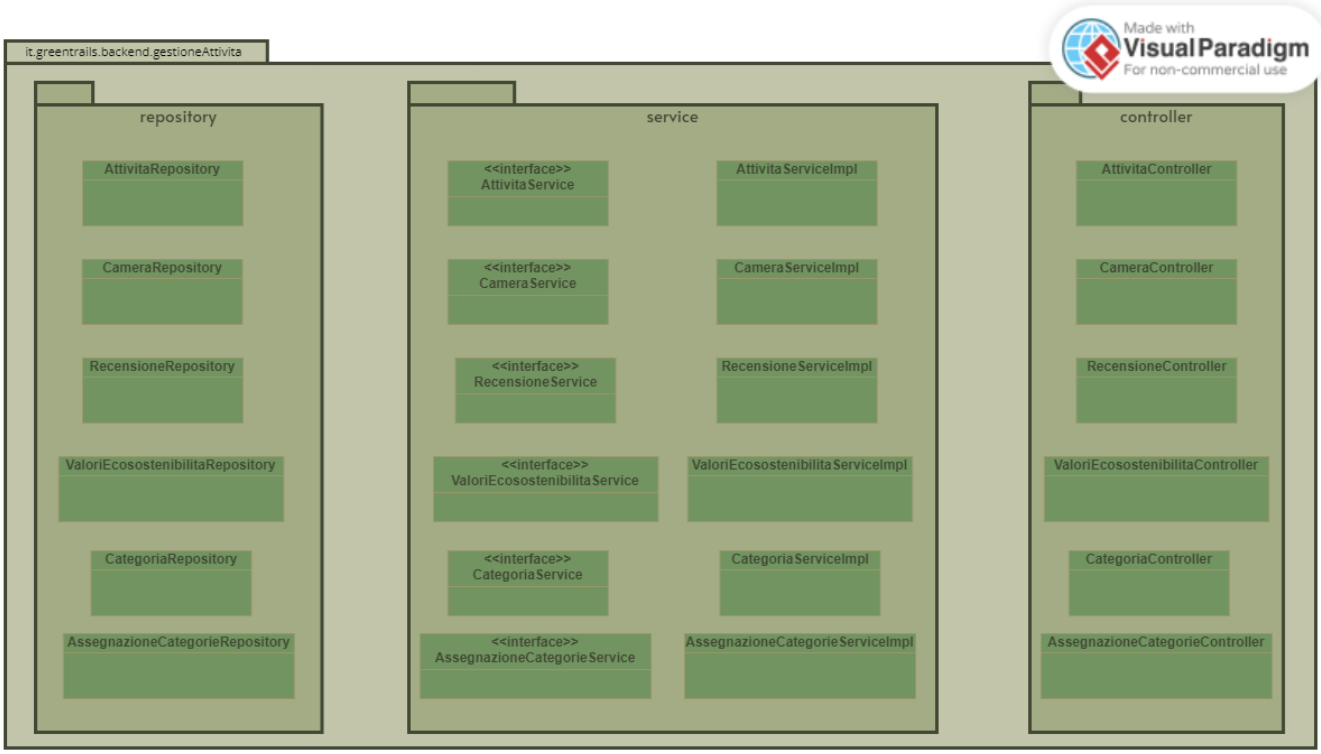
2.3 Package GestionePagamenti

In questo package si troveranno tutte le classi del sottosistema per la gestione dei pagamenti.



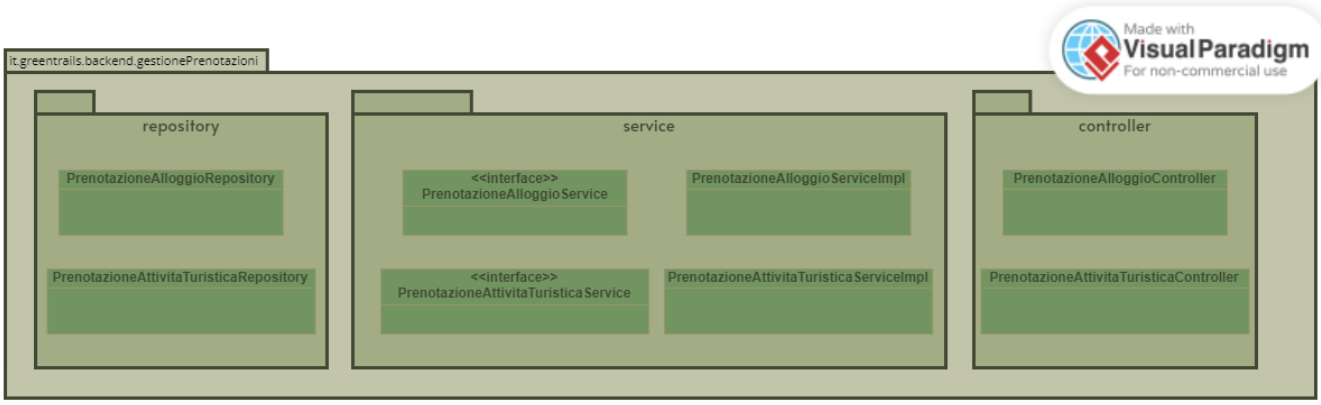
2.4 Package GestioneAttivita

In questo package si troveranno tutte le classi del sottosistema per la gestione delle attività.



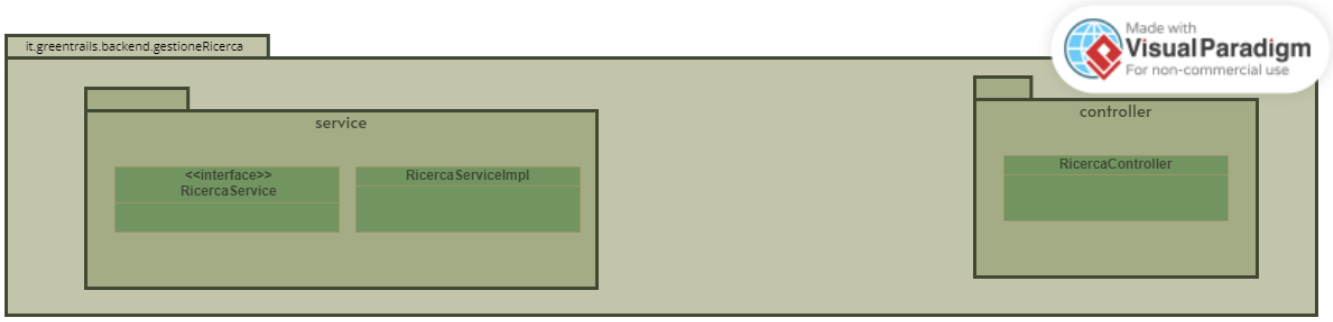
2.5 Package GestionePrenotazioni

In questo package, invece, saranno collocate le classi riguardanti il sottosistema della gestione delle prenotazioni.



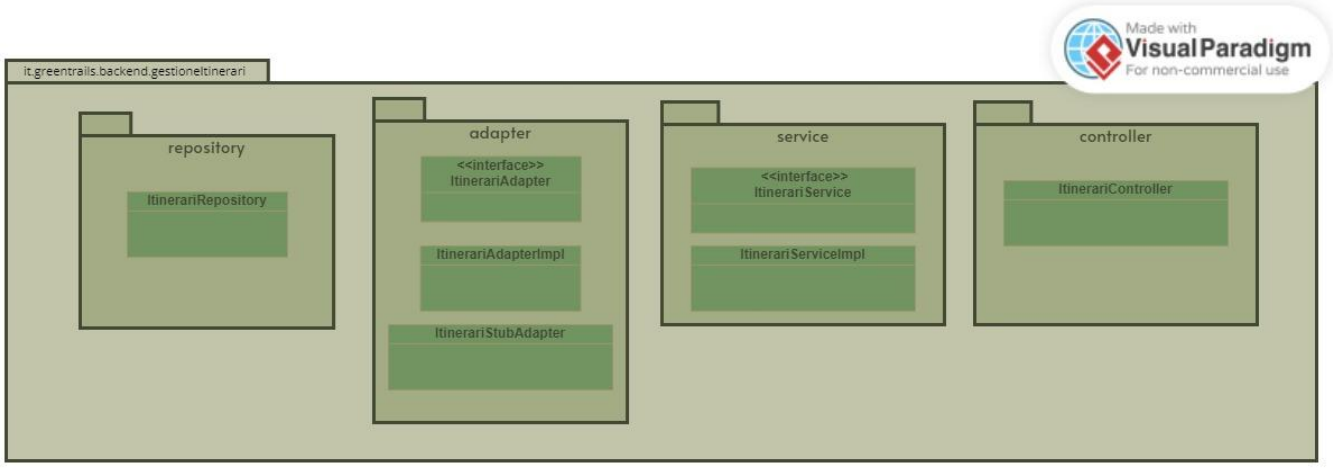
2.6 Package GestioneRicerca

In questo package verranno mostrate le classi facenti parte del sottosistema della gestione della ricerca.



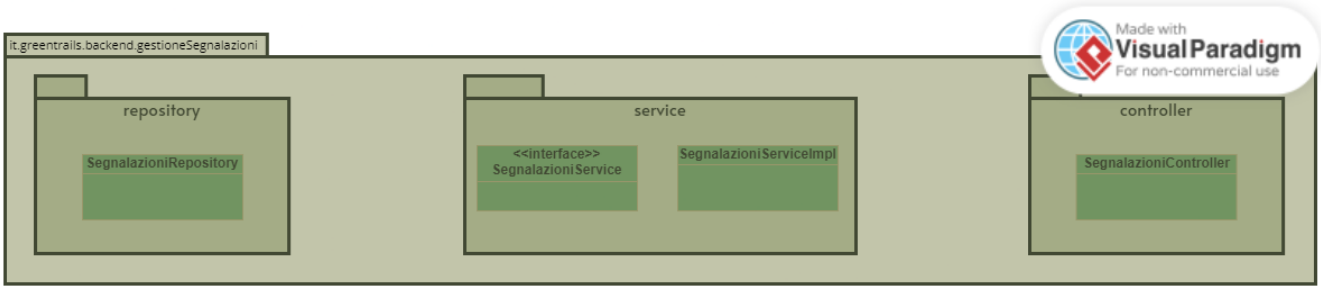
2.7 Package GestioneItinerari

In questo package verranno presentate le classi facenti parte del sottosistema della gestione degli itinerari. Inoltre, qui verrà collocata l'integrazione con il modulo di intelligenza artificiale, garantita attraverso un Adapter.



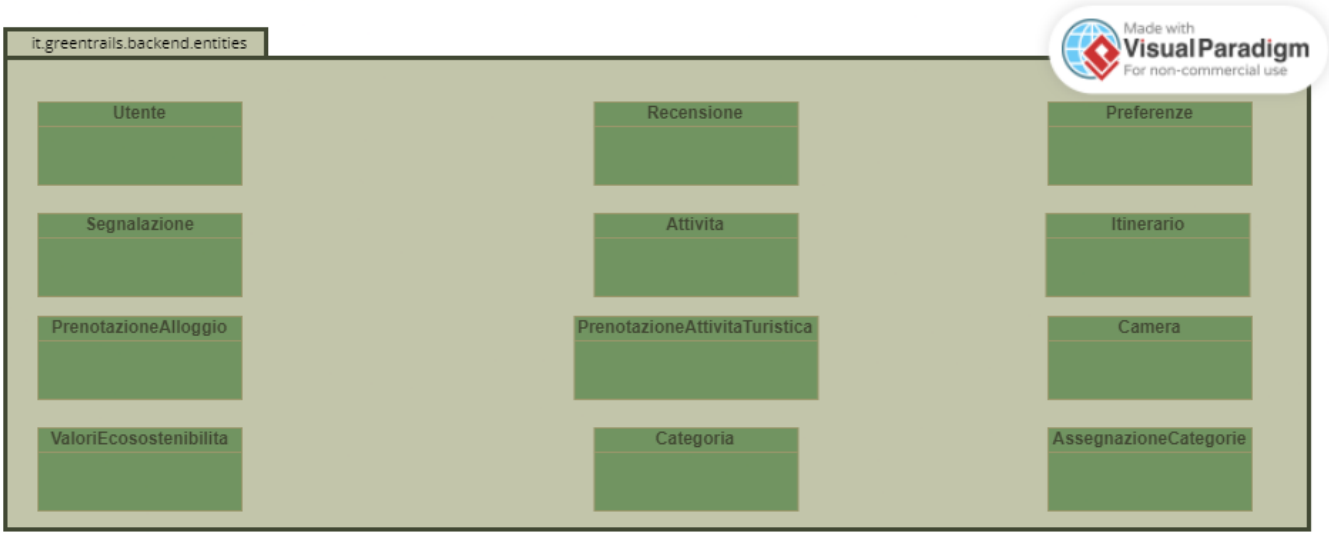
2.8 Package GestioneSegnalazioni

In questo package saranno collocate le classi facenti parte del sottosistema della gestione delle segnalazioni.



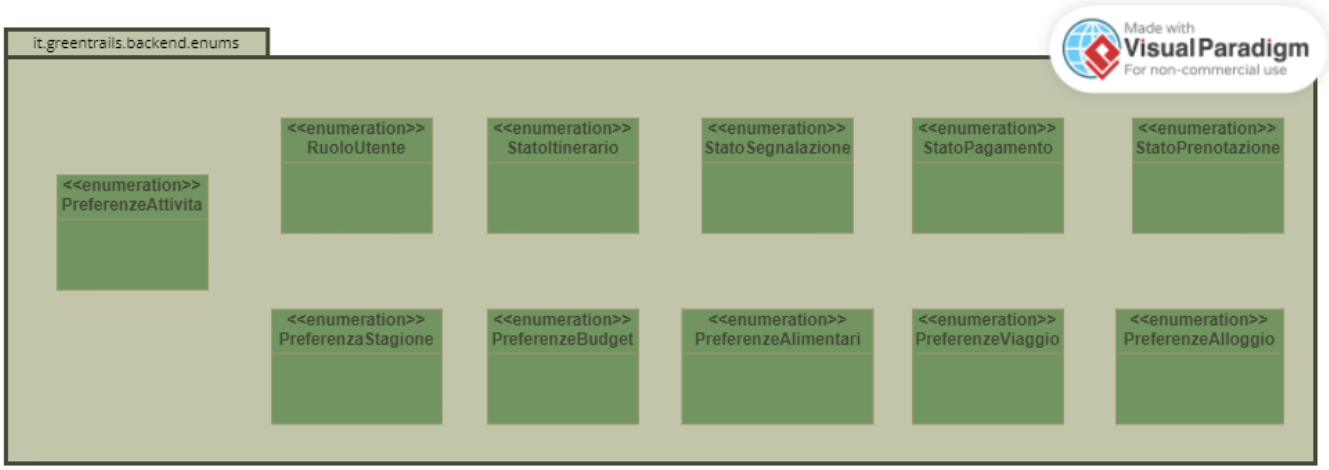
2.9 Package Entities

Questo package presenta le entità presenti nel sistema proposto.



2.10 Package Enums

In questo package ci sono tutti gli enums presenti nel sistema proposto.



3. Interfacce delle classi

3.1 Package GestioneUtenze

Nome Classe	*GestioneUtenzeService*
Descrizione	Permette di gestire le operazioni relative agli utenti.
Metodi	+ findById(Long id): Utente + saveUtente(Utente utente): Utente + findByEmail(String email): Optional<Utente> + deleteUtente(Utente utente): boolean
Invariante di classe	**NA**

Nome Metodo	*+ findById(Long id): Utente*
Descrizione	Permette di recuperare un utente in base al suo id.
Pre-condizione	**context:** GestioneUtenze::findById(Long id) **post:** id <> null && id >= 0
Post-condizione	**context:** GestioneUtenze::findById(Long id) **post:** Utente <> null

Nome Metodo	*+ saveUtente(Utente utente): Utente*
Descrizione	Permette la creazione e la modifica dell'utente.
Pre-condizione	**context:** GestioneUtenze::saveUtente(Utente utente) **pre:** utente <> null
Post-condizione	**context:** GestioneUtenze::saveUtente(Utente utente) **post:** Utente <> null

Nome Metodo	*+ findByEmail(String email): Optional<Utente>*
Descrizione	Permette di recuperare un utente in base al suo indirizzo e-mail.
Pre-condizione	**NA**
Post-condizione	**NA**

Nome Metodo	*+ deleteUtente(Utente utente): Utente*
Descrizione	Permette di eliminare un utente dalla piattaforma.
Pre-condizione	**context:** GestioneUtenze::deleteUtente(Utente utente) **pre:** utente<>null
Post-condizione	**NA**

3.2 Package GestionePagamenti

Nome Classe	*PagamentiService*
Descrizione	Permette di gestire le operazioni relative ai pagamenti.
Metodi	+ modificaFee(Double fee): void + aggiornaStatoPagamentoAlloggio(PrenotazioneAlloggio prenotazioneAlloggio, StatoPagamento stato): PrenotazioneAlloggio + aggiornaStatoPagamentoAttivitaTuristica(PrenotazioneAttivitaTuristica prenotazioneAttivitaTuristica, StatoPagamento stato): PrenotazioneAttivitaTuristica
Invariante di classe	**NA**

Nome Metodo	*+ modificaFee(Double fee): void*
Descrizione	Permette di modificare la percentuale sulla fee aggiuntiva sui pagamenti.
Pre-condizione	**context:** PagamentiService::modificaFee(Double fee) **pre:** fee <> null && fee >= 0 && fee <= 100
Post-condizione	**NA**

Nome Metodo	*+ aggiornaStatoPagamentoAlloggio(PrenotazioneAlloggio prenotazioneAlloggio, StatoPagamento stato): PrenotazioneAlloggio*
Descrizione	Permette di recuperare l'aggiornamento delle informazioni sullo stato del pagamento di un alloggio.
Pre-condizione	**context:**PagamentiService::aggiornaStatoPagamentoAlloggio(PrenotazioneAlloggio prenotazioneAlloggio, StatoPagamento stato) **pre:** prenotazioneAlloggio <> null && stato <> null
Post-condizione	**context:** PagamentiService:: aggiornaStatoPagamentoAlloggio(PrenotazioneAlloggio prenotazioneAlloggio, StatoPagamento stato) **post:** PrenotazioneAlloggio <> null

Nome Metodo	*+ aggiornaStatoPagamentoAttivitaTuristica(PrenotazioneAttivitaTuristica prenotazioneAttivitaTuristica, StatoPagamento stato): PrenotazioneAttivitaTuristica*
Descrizione	Permette di recuperare l'aggiornamento delle informazioni sullo stato del pagamento di un alloggio.
Pre-condizione	**context:**PagamentiService::aggiornaStatoPagamentoAttivitaTuristica(PrenotazioneAttivitaTuristica prenotazioneAttivitaTuristica, StatoPagamento stato) **pre:** prenotazioneAttivitaTuristica<> null && stato <> null
Post-condizione	**context:** PagamentiService::aggiornaStatoPagamentoAttivitaTuristica(PrenotazioneAttivitaTuristica prenotazioneAttivitaTuristica, StatoPagamento stato) **post:** PrenotazioneAttivitaTuristica<> null

3.3 Package GestioneAttività

Nome Classe	*AttivitaService*
Descrizione	Permette di gestire le operazioni relative alle attività.
Metodi	+ saveAttivita(Attivita attivita): Attivita + findById(Long id): Attivita + findAllAttivitaByGestore(Utente utente): List<Attivita> + findGestoreByAttivita(Attivita attivita): Utente + deleteAttivita(Attivita attivita): boolean
Invariante di classe	**NA**

Nome Metodo	*+ saveAttivita(Attivita attivita): Attivita*
Descrizione	Permette di creare o modificare un'attività.
Pre-condizione	**context:** AttivitaService::saveAttivita(Attivita attivita) **pre:** attivita <> null
Post-condizione	**context:** AttivitaService::saveAttivita(Attivita attivita) **pre:** Attivita <> null

Nome Metodo	*+ findById(Long id): Attivita*
Descrizione	Permette di recuperare un'attività tramite il proprio id.
Pre-condizione	**context:** AttivitaService::findById(Long id) **pre:** id <> null && id >= 0
Post-condizione	**context:** AttivitaService::findById(Long id) **post:** Attivita <> null

Nome Metodo	*+ findAllAttivitaByGestore(Long id): List<Attivita>*
Descrizione	Permette di trovare la lista di attività che un gestore possiede.
Pre-condizione	**context:** AttivitaService::findAttivitaByGestore(Long id) **pre:** id <> null && id >= 0
Post-condizione	**NA**

Nome Metodo	*+ findGestoreByAttivita(Attivita attivita): Utente*
Descrizione	Permette di recuperare il gestore di una specifica attività.
Pre-condizione	**context:** GestioneUtenze::findByAttivita(Attivita attivita) **pre:** attivita <> null
Post-condizione	**context:** GestioneUtenze::findByAttivita(Attivita attivita) **pre:** Utente <> null

Nome Metodo	*+ deleteAttivita(Attivita attivita): boolean*
Descrizione	Permette la rimozione di un'attività da parte di un gestore.
Pre-condizione	**context:** AttivitaService::deleteAttivita(Attivita attivita) **pre:** attivita <> null
Post-condizione	**NA**

Nome Classe	*CameraService*
Descrizione	Permette di gestire le operazioni relative alle camere di un alloggio.
Metodi	+ saveCamera(Camera camera): Camera + findById(Long id): Camera + deleteCamera(Camera camera): boolean
Invariante di classe	**NA**

Nome Metodo	*+ saveCamera(Camera camera): Camera*
Descrizione	Permette la creazione e la modifica di una camera di un alloggio.
Pre-condizione	context: CameraService::getCamereByAlloggio(Attivita alloggio) pre: alloggio <> null && isAlloggio == true
Post-condizione	**NA**

Nome Metodo	*+ getCamereByAlloggio (Attivita alloggio): List<Camera>*
Descrizione	Permette di visualizzare tutte le camere di un alloggio
Pre-condizione	**context:** CameraService::getCamereByAlloggio(Attivita alloggio) **pre:** alloggio <> null && isAlloggio == true
Post-condizione	**NA**

Nome Metodo	*+ findById(Long id): Camera*
Descrizione	Permette di recuperare una camera tramite il proprio id.
Pre-condizione	**context:** CameraService::findById(Long id) **pre:** id <> null && id >= 0
Post-condizione	**context:** CameraService::findById(Long id) **post:** Camera <> null

Nome Metodo	*+ deleteCamera(Camera camera): boolean*
Descrizione	Permette l'eliminazione di una camera di un alloggio.
Pre-condizione	**context:** CameraService::deleteCamera(Camera camera)

	pre: camera <> null
Post-condizione	**NA**

Nome Classe	*RecensioneService*
Descrizione	Permette di gestire le operazioni relative alle recensioni.
Metodi	+ saveRecensione(Recensione recensione): Recensione + findById(Long id): Recensione + deleteRecensione(Recensione recensione): boolean + getRecensioniByAttivita(Attivita attivita): List<Recensione> + getAllRecensioniByVisitatore(Utente utente): List<Recensione>
Invariante di classe	**NA**

Nome Metodo	*+ saveRecensione(Recensione recensione): Recensione*
Descrizione	Permette l'inserimento e la modifica di una recensione da parte di un utente ad un'attività.
Pre-condizione	**context:** RecensioneService::saveRecensione(Recensione recensione) **pre:** recensione <> null
Post-condizione	**NA**

Nome Metodo	*+ findById(Long id): Recensione*
Descrizione	Permette di recuperare una recensione tramite il proprio id.
Pre-condizione	**context:** RecensioneService::findById(Long id) **pre:** id <> null && id >= 0
Post-condizione	**context:** RecensioneService::findById(Long id) **post:** Recensione <> null

Nome Metodo	*+ deleteRecensione (Recensione recensione): boolean*
Descrizione	Permette l'eliminazione di una recensione di un'attività.
Pre-condizione	**context:** RecensioneService::deleteRecensione(Recensione recensione) **pre:** recensione <> null
Post-condizione	**NA**

Nome Metodo	*+ getRecensioniByAttivita (Attivita attivita): List <Recensione>*
Descrizione	Visualizza tutte le recensioni di un'attività.
Pre-condizione	**context:** RecensioneService::*getRecensioniByAttivita (Attivita attivita)* **pre:** attivita <> null
Post-condizione	**NA**

Nome Metodo	*+ getAllRecensioniByVisitatore(Utente utente): List<Recensione>*
Descrizione	Permette di recuperare tutte le recensioni scritte da un utente.
Pre-condizione	**context:** RecensioneService::getAllRecensioniByVisitatore(Utente utente) **pre:** utente <> null
Post-condizione	**NA**

Nome Classe	*ValoriEcosostenibilitaService*
Descrizione	Permette di gestire le operazioni relative ai valori di ecosostenibilità di un'attività.
Metodi	+ saveValori(ValoriEcosostenibilita valori): ValoriEcosostenibilita + deleteValori(ValoriEcosostenibilita valori): boolean + findById(Long id): ValoriEcosostenibilita
Invariante di classe	**NA**

Nome Metodo	*+ saveValori(ValoriEcosostenibilita valori): ValoriEcosostenibilita*
Descrizione	Permette di creare e modificare i valori di ecosostenibilità di un'attività.
Pre-condizione	**context:** ValoriEcosostenibilitaService::*saveValori (ValoriEcosostenibilita valori)* **pre:** valori<> null
Post-condizione	**NA**

Nome Metodo	*+ deleteValori(ValoriEcosostenibilita valori): boolean*
Descrizione	Permette di eliminare i valori di ecosostenibilità di un'attività.
Pre-condizione	**context:** ValoriEcosostenibilitaService::delete*Valori (ValoriEcosostenibilita valori)* **pre:** valori<> null
Post-condizione	**NA**

Nome Metodo	*+ findById(Long id): ValoriEcosostenibilita*
Descrizione	Permette di recuperare i valori di ecosostenibilità di un'attività tramite il proprio id.
Pre-condizione	**context:** ValoriEcosostenibilitaService::findById(Long id) **pre:** id <> null && id >= 0
Post-condizione	**context:** ValoriEcosostenibilitaService::findById(Long id) **post:** ValoriEcosostenibilita <> null

Nome Classe	*CategoriaService*
Descrizione	Permette di gestire le operazioni relative alle categorie.
Metodi	+ saveCategoria(Categoria categoria): Categoria + deleteCategoria(Categoria categoria): boolean + findById(Long id): Categoria
Invariante di classe	**NA**

Nome Metodo	*+ saveCategoria(Categoria categoria): Categoria*
Descrizione	Permette l'inserimento e la modifica delle categorie delle attività.
Pre-condizione	**context:** CategoriaService::saveCategoria (Categoria categoria) **pre:** categoria <> null
Post-condizione	**NA**

Nome Metodo	*+ deleteCategoria(Recensione recensione): boolean*
Descrizione	Permette l'eliminazione di una categoria di attività.
Pre-condizione	**context:** CategoriaService::deleteCategoria(Categoria categoria) **pre:** categoria <> null
Post-condizione	**NA**

Nome Metodo	*+ findById(Long id): Categoria*
Descrizione	Permette di recuperare una categoria tramite il proprio id.
Pre-condizione	**context:** CategoriaService::findById(Long id) **pre:** id <> null && id >= 0
Post-condizione	**context:** CategoriaService::findById(Long id) **post:** Categoria <> null

Nome Classe	*AssegnazioneCategorieService*
Descrizione	Permette di gestire le operazioni relative all'assegnazione delle categorie ad un'attività.
Metodi	+ assegnazioneCategoria(Attivita attivita, Categoria categoria): Attivita + deleteAssegnazione(Attivita attivita, Categoria categoria): boolean
Invariante di classe	**NA**

Nome Metodo	*+ assegnazioneCategoria(Attivita attivita, Categoria categoria): Attivita*
Descrizione	Permette di assegnare categorie alle attività.
Pre-condizione	**context:** CategoriaService::assegnaCategoria (Attivita attivita, Categoria categoria) **pre:** attivita <> null && categoria <> null
Post-condizione	**NA**

Nome Metodo	*+ deleteAssegnazione(Attivita attivita, Categoria categoria): boolean*
Descrizione	Permette di rimuovere una categoria da un'attività.
Pre-condizione	**context:** CategoriaService:: deleteAssegnazione (Attivita attivita, Categoria categoria) **pre:** attivita <> null && categoria <> null
Post-condizione	**NA**

3.4 Package GestionePrenotazioni

Nome Classe	*PrenotazioneAlloggioService*
Descrizione	Permette la gestione delle operazioni relative alle prenotazioni di un alloggio.
Metodi	+ savePrenotazioneAlloggio(Attivita attivita, PrenotazioneAlloggio prenotazioneAlloggio): PrenotazioneAlloggio + deletePrenotazioneAlloggio(PrenotazioneAlloggio prenotazioneAlloggio): boolean + getAllPrenotazioniAlloggio(): List<PrenotazioneAlloggio> + getPrenotazioniAlloggioByStato(StatoPrenotazione stato): List<PrenotazioneAlloggio> + findById(Long id): PrenotazioneAlloggio + getPrenotazioniByAlloggio(Attivita attivita): List<PrenotazioneAlloggio>
Invariante di classe	**NA**

Nome Metodo	*+ savePrenotazioneAlloggio(Attivita attivita, PrenotazioneAlloggio prenotazioneAlloggio): PrenotazioneAlloggio*
Descrizione	Permette la creazione e la modifica di una prenotazione di un alloggio.
Pre-condizione	**context:** PrenotazioneAlloggioService::savePrenotazioneAlloggio(Attivita attivita, PrenotazioneAlloggio prenotazioneAlloggio) **pre:** prenotazioneAlloggio <> null
Post-condizione	**NA**

Nome Metodo	*+ deletePrenotazioneAlloggio(Prenotazione prenotazioneAlloggio): boolean*
Descrizione	Permette la cancellazione di una prenotazione di un alloggio.
Pre-condizione	**context:** PrenotazioniAlloggioService::deletePrenotazioneAlloggio(Prenotazione prenotazioneAlloggio) **pre:** prenotazioneAlloggio <> null
Post-condizione	**NA**

Nome Metodo	*+ getAllPrenotazioniAlloggio(): List<PrenotazioneAlloggio>*
Descrizione	Permette di visualizzare la lista di tutte le prenotazioni di un alloggio.
Pre-condizione	**NA**
Post-condizione	**NA**

Nome Metodo	*+ getPrenotazioniAlloggioByStato(StatoPrenotazione stato): List<PrenotazioneAlloggio>*
Descrizione	Permette la visualizzazione delle prenotazioni di un alloggio, tramite una lista, in base al loro stato.
Pre-condizione	**context:**PrenotazioneAlloggioService::getPrenotazioniAlloggioByStato(StatoPrenotazione stato) **pre:** stato <> null
Post-condizione	**NA**

Nome Metodo	*+ findById(Long id): PrenotazioneAlloggio*
Descrizione	Permette di recuperare una prenotazione di un alloggio tramite il proprio id.
Pre-condizione	**context:** PrenotazioneAlloggioService::findById(Long id) **pre:** id <> null && id >= 0
Post-condizione	**context:** PrenotazioneAlloggioService::findById(Long id) **post:** PrenotazioneAlloggioService<> null

Nome Metodo	*+ getPrenotazioniByAlloggio(Attivita attivita): List<PrenotazioneAlloggio>*
Descrizione	Permette la visualizzazione delle prenotazioni di un alloggio, tramite una lista, in base all'attività.
Pre-condizione	**context:** GestionePrenotazioniService::getPrenotazioniByAlloggio(Attivita attivita) **pre:** attivita <> null && attivita instanceof Alloggio
Post-condizione	**NA**

Nome Classe	*PrenotazioneAttivitaTuristicaService*
Descrizione	Permette la gestione delle operazioni relative alle prenotazioni di un'attività turistica.
Metodi	+ savePrenotazioneAttivitaTuristica(Attivita attivita, PrenotazioneAttivitaTuristica prenotazioneAttivitaTuristica): PrenotazioneAttivitaTuristica + deletePrenotazioneAttivitaTuristica(PrenotazioneAttivitaTuristica prenotazioneAttivitaTuristica): boolean + getAllPrenotazioniAttivitaTuristica(): List<PrenotazioneAttivitaTuristica> + getPrenotazioniAttivitaTuristicaByStato(StatoPrenotazione stato): List<PrenotazioneAttivitaTuristica> + findById(Long id): PrenotazioneAttivitaTuristica + getPrenotazioniByAttivitaTuristica(Attivita attivita): List<PrenotazioneAttivitaTuristica>
Invariante di classe	**NA**

Nome Metodo	*+ savePrenotazioneAttivitaTuristica(Attivita attivita, PrenotazioneAttivitaTuristica prenotazioneAttivitaTuristica): PrenotazioneAttivitaTuristica*
Descrizione	Permette la creazione e la modifica di una prenotazione di un'attività turistica.
Pre-condizione	**context:**PrenotazioneAttivitaTuristicaService::savePrenotazioneAttivitaTuristica(Attivita attivita, PrenotazioneAttivitaTuristica prenotazioneAttivitaTuristica) **pre:** prenotazioneAttivitaTuristica <> null
Post-condizione	**NA**

Nome Metodo	*+ deletePrenotazioneAttivitaTuristica(Prenotazione prenotazioneAttivitaTuristica): boolean*
Descrizione	Permette la cancellazione di una prenotazione di un'attività turistica.
Pre-condizione	**context:**PrenotazioneAttivitaTuristicaService::deletePrenotazioneAttivitaTuristica(Prenotazione prenotazioneAttivitaTuristica) **pre:** prenotazioneAttivitaTuristica <> null
Post-condizione	**NA**

Nome Metodo	*+ getAllPrenotazioniAttivitaTuristica(): List<PrenotazioneAttivitaTuristica>*
Descrizione	Permette di visualizzare la lista di tutte le prenotazioni di un'attività turistica.

Pre-condizione	**NA**
Post-condizione	**NA**

Nome Metodo	***+ getPrenotazioniAttivitaTuristicaByStato(StatoPrenotazione stato): List<PrenotazioneAttivitaTuristica>***
Descrizione	Permette la visualizzazione delle prenotazioni di un alloggio, tramite una lista, in base al loro stato.
Pre-condizione	**context:**PrenotazioneAttivitaTuristicaService::getPrenotazioniAlttivitaTuristicaByStato (StatoPrenotazione stato) **pre:** stato <> null
Post-condizione	**NA**

Nome Metodo	***+ getPrenotazioniByVisitatore(Utente visitatore): List<PrenotazioneAttivitaTuristica>***
Descrizione	Permette la visualizzazione delle prenotazioni delle attività turistiche di un visitatore.
Pre-condizione	**context:**PrenotazioneAttivitaTuristicaService::getPrenotazioniByVisitatore(Utente visitatore) **pre:** visitatore <> null && ruolo == VISITATORE
Post-condizione	**NA**

Nome Metodo	*+ findById(Long id): PrenotazioneAttivitaTuristica*
Descrizione	Permette di recuperare una prenotazione di un'attività turistica tramite il proprio id.
Pre-condizione	**context:**PrenotazioneAttivitaTuristicaSevice::findById(Long id) **pre:** id <> null && id >= 0
Post-condizione	**context:** PrenotazioneAttivitaTuristicaSevice::findById(Long id) **post:** PrenotazioneAttivitaTuristica <> null

Nome Metodo	*+ getPrenotazioniByAttivitaTuristica(Attivita attivita): List<PrenotazioneAttivitaTuristica>*
Descrizione	Permette la visualizzazione delle prenotazioni di un'attività turistica, tramite una lista, in base all'attività.
Pre-condizione	**context:**PrenotazioneAttivitaTuristicaService::gePrenotazionitByAttivitaTuristica(Attiv ita attivita) **pre:** attivita <> null && attivita instanceof AttivitaTuristica
Post-condizione	**NA**

3.5 Package GestioneRicerca

Nome Classe	*RicercaService*
Descrizione	Permette di gestire le operazioni relative alla ricerca di attività.
Metodi	+ findAttivita(String query): List<Attivita> + findAttivitaByCategorie(List<Categoria> categorie): List<Attivita> + findAttivitaByPosizione(Point coordinate, Double raggio): List<Attivita>
Invariante di classe	**NA**

Nome Metodo	*+ findAttivita(String query): List<Attivita>*
Descrizione	Permette la ricerca di una lista di attività.
Pre-condizione	**context:** RicercaService::findAttivita(String query) **pre:** text <> null
Post-condizione	**NA**

Nome Metodo	*+ findAttivitaByCategorie(List<Categoria> categorie): List<Attivita>*
Descrizione	Permette la ricerca di una lista di attività in base alle categorie inserite.
Pre-condizione	**context:** RicercaService::findAttivitaByCategorie(List<Categoria> categorie) **pre:** categorie <> null
Post-condizione	**NA**

Nome Metodo	+ findAttivitaByPosizione(Point coordinate, Double raggio): List<Attivita>
Descrizione	Permette la ricerca di una lista di attività in base alla sua posizione.
Pre-condizione	**context:** RicercaService::findAttivitaByPosizione(Point coordinate, Double raggio) **pre:** coordinate <> null && raggio <> null && raggio >= 0
Post-condizione	**NA**

3.6 Package GestioneItinerari

Nome Classe	*ItinerariService*
Descrizione	Permette di gestire le operazioni relative agli itinerari.
Metodi	+ saveItinerario(Itinerario itinerario): Itinerario + createByPreferenze(Preferenze preferenze): Itinerario + findItinerariByUtente(Utente utente): List<Itinerario> + deleteItinerario(Itinerario itinerario): boolean + findById(Long id): Itinerario
Invariante di classe	**NA**

Nome Metodo	*+ saveItinerario(Itinerario itinerario): Itinerario*
Descrizione	Permette la creazione e la modifica di un itinerario.
Pre-condizione	**context:** ItinerariService::savetinerario (Itinerario itinerario) **pre**: itinerario <> null
Post-condizione	**NA**

Nome Metodo	*+ createByPreferenze(Preferenze preferenze): Itinerario*
Descrizione	Permette la creazione automatica di un nuovo itinerario in base alle preferenze dell'utente.
Pre-condizione	**context:** ItinerariService::createByPreferenze (Preferenze preferenze) **pre**: preferenze <> null
Post-condizione	**NA**

Nome Metodo	*+ findItinerariByUtente(Utente utente): List <Itinerario>*
Descrizione	Permette la visualizzazione degli itinerari di un utente
Pre-condizione	**context:** ItinerariService::findItinerarioByUtente(Utente utente) **pre**: utente <> null && ruolo == VISITATORE
Post-condizione	**NA**

Nome Metodo	*+deleteItinerario(Itinerario itinerario): boolean*
Descrizione	Permette la rimozione degli itinerari.
Pre-condizione	**context:** ItinerariService::deleteIneratio(Itinerario itinerario) **pre**: itinerario <> null
Post-condizione	**NA**

Nome Metodo	*+ findById(Long id): Itinerario*
Descrizione	Permette di recuperare un itinerario tramite il proprio id.
Pre-condizione	**context:** ItinerariService::findById(Long id) **pre:** id <> null && id >= 0
Post-condizione	**context:** ItinerariService::findById(Long id) **post:** Itinerario <> null

3.7 Package GestioneSegnalazioni

Nome Classe	*SegnalazioniService*
Descrizione	Permette di gestire le operazioni relative alle segnalazioni.
Metodi	+ saveSegnalazione(Segnalazione segnalazione): Segnalazione + getAllSegnalazioniByTipo(boolean isForRecensione): List<Segnalazione> + findById(Long id): Segnalazione + getSegnalazioniByStato(StatoSegnalazione stato): List<Segnalazione>
Invariante di classe	**NA**

Nome Metodo	*+ saveSegnalazione(Segnalazione segnalazione): Segnalazione*
Descrizione	Permette la creazione e la modifica di una segnalazione.
Pre-condizione	**context:** SegnalazioniService::saveSegnalazione(Segnalazione segnalazione) **pre:** segnalazione <> null
Post-condizione	**NA**

Nome Metodo	*+ getAllSegnalazioniByTipo(boolean isForRecensione): List<Segnalazione>*
Descrizione	Permette di visualizzare la lista di tutte le segnalazioni delle recensioni
Pre-condizione	**context:** SegnalazoniService::getAllSegnalazioniByTipo(boolean IsForRecensione) **pre**: IsForRecensione <> null
Post-condizione	**NA**

Nome Metodo	*+ findById(Long id): Segnalazione*
Descrizione	Permette di recuperare un itinerario tramite il proprio id.
Pre-condizione	**context:**SegnalazioniService::findById(Long id) **pre:** id <> null && id >= 0
Post-condizione	**context:** SegnalazioniService::findById(Long id) **post:** Segnalazione <> null

Nome Metodo	***+ getSegnalazioniByStato(StatoSegnalazione stato): List<Segnalazione>***
Descrizione	Permette la visualizzazione delle segnalazioni, tramite una lista, in base al loro stato.
Pre-condizione	**context:**SegnalazioneService:: getSegnalazioniByStato(StatoSegnalazione stato) **pre:** stato <> null
Post-condizione	**NA**

3.8 Package Utils

Nome Classe	*EmailService*
Descrizione	Permette di gestire le operazioni relative alle mail.
Metodi	+ sendEmail(String destinatario, String oggetto, String corpo): void
Invariante di classe	**NA**

Nome Metodo	*+ sendEmail(String msg): String*
Descrizione	Permette l'invio di una mail.
Pre-condizione	**context:** EmailService::sendEmail(String msg) **pre**: msg <> null
Post-condizione	**NA**

4. Class Diagram

Per una migliore visualizzazione del Class Diagram, cliccare QUI

5. Glossario

Sigla/Termine	Definizione
Java	Linguaggio di programmazione orientato agli oggetti general purpose e multipiattaforma, eseguito sulla Java Virtual Machine.
Python	Linguaggio di programmazione multipiattaforma dedicato principalmente alla computazione numerica.
SQL	Linguaggio di interrogazione per gestire dati in database relazionali.
HTML	Linguaggio di markup standard usato per i documenti web.
JavaScript	Linguaggio di programmazione multi-paradigma, utilizzato come standard per lo scripting nei web browser.
CSS	Linguaggio utilizzato per la formattazione e la stilizzazione di documenti HTML, XHTML e XML.
Package	Meccanismo per organizzare classi e interfacce Java in gruppi logici, allo scopo di definire namespace distinti in contesti differenti.
Namespace	Spazio dei nomi utilizzato allo scopo di evitare confusione nel caso di molte entità con nomi simili.
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.
Do It Yourself	Etica "Fai-da-te" che indica una qualsiasi attività eseguita per conto proprio e senza la partecipazione di terzi.