Object Design Document

Green Leaf

Riferimento	
Versione	2.0
Data	11/02/2023
Destinatario	Prof.ssa F. Ferrucci, Prof. F. Palomba
Presentato da	Alessandro Borrelli, Vincenzo Cerciello, Michela Faella, Gerardo Napolitano, Mirko Vitale
Approvato da	

Revision History

Data	Versione	Descrizione	Autori
23/12/2022	1.0	Stesura del punto 1 e 2	Team
29/12/2022	2.0	Stesura dei rimanenti capitoli	Team
11/02/2023	2.0	Revisione pre consegna	Team

Sommario

1 Introduzione

1.1. Object design trade-off

Durante la fase di Object Design sono sorti diversi compromessi di progettazione, di seguito sono riportati i trade-off:

1.1.1. Prestazioni vs Costi

Tenuto conto del budget stanziato, e dalla necessità di avere funzioni chiave perfettamente operative nei tempi prestabiliti, si preferisce dedicare il monte ore a disposizione all'implementazione e revisione di quest'ultime.

1.1.2. Prestazioni vs Affidabilità

Dovendo il sistema gestire dati sensibili, si preferisce garantire un maggior controllo di input e consistenza a scapito della latenza.

1.2. Componenti off-the-shelf

Green Leaf farà affidamento su varie componenti off-the-shelf sia per il front-end che per il back-end. Per quanto riguarda il front-end verrà utilizzato HTML5, linguaggio di mark-up attualmente più diffuso, CSS3, per definire lo stile delle pagine Web, e Bootstrap, raccolta di strumenti per la creazione di applicazioni Web.
Lato back-end saranno utilizzate le seguenti componenti off-the-shelf: MySQL, Tomcat, Maven. MySQL è un celebre RDBMS di Oracle. Il suo utilizzo sarà fondamentale per la realizzazione, la gestione e l'implementazione della base di dati di GreenLeaf. Tutti i dati che per necessità di business devono essere salvati persistentemente saranno affidati ad un database basato su MySQL. Sarà inoltre utilizzata la componente MySQL Connector/J, il driver JDBC ufficiale per MySQL necessario per la comunicazione con la base di dati attraverso Java.
Tomcat è un web server open-source sviluppato dalla Apache Software Foundation. Il suo impiego è necessario, poiché tutto il back-end di GreenLeaf (nucleo di elaborazione principale, comunicazione con la base di dati) sarà eseguito in ambiente Tomcat. Esso permetterà di utilizzare le Java Servlet per l'implementazione della logica di business e le Java Servlet Pages per la generazione dinamica delle pagine web utilizzate dal client. Maven è uno strumento di gestione dei progetti software basati su Java, anch'esso sviluppato dalla Apache Software Foundation. Agevolerà la gestione delle varie librerie utilizzate, delle loro versioni e delle dipendenze tra esse, permettendo agli sviluppatori di concentrarsi esclusivamente sulla scrittura del codice.

1.3 Linee guida per la documentazione dell'interfaccia

È richiesto agli sviluppatori di seguire le seguenti linee guida al fine di essere consistenti nell'intero progetto e facilitare la comprensione delle funzionalità di ogni componente**.**

Tipi	Regole per la denominazione	Esempi
Package	Il prefisso di un nome di pacchetto univoco sempre scritto in minuscolo e tutte le lettere ASCII minuscole tranne le abbreviazioni come UI, in modo da non rendere il nome del pacchetto lungo	package greenleaf; package src;
Classi	I nomi delle classi dovrebbero essere sostantivi, scritti in CamelCase. I nomi delle classi semplici e descrittivi, utilizzando parole intere ed evitato per quanto possibile acronimi e abbreviazioni (a meno che l'abbreviazione non sia molto più utilizzata rispetto alla forma lunga, come DB per database e UI per interfaccia utente)	class UtenteDB; class Tracciamento;
Interfacce	I nomi delle interfacce devono essere scritti in CamelCase come i nomi delle classi.	interface Identificativo;
Metodi	I nostri metodi sono verbi, scritti nella forma camelCase (da notare la prima lettera minuscola).	insert(); createReport();
Variabili	I nomi delle variabili sono brevi ma significativi. La scelta del nome di una variabile è mnemonica, cioè finalizzata a indicare all'osservatore casuale l'intento del suo utilizzo. I nomi delle variabili composte da una sola lettera vengono evitati tranne che per le variabili "usa e getta" temporanee. Se i nomi delle varabili sono composti da due nomi allora verranno scritti nella forma camelCase (da notare la prima lettera minuscola).	int i; string nomeRegione; float costo;
Costanti	I nomi delle variabili dichiarate costanti di classe e delle costanti ANSI devono essere tutti maiuscoli con parole separate da trattini bassi ("_"). (Le costanti ANSI dovrebbero essere evitate, per facilitare il debug.)	static finali int MAX = 5;

1.5 Riferimenti

Bernd Bruegge, Allen H. Dutoit - Object-Oriented Software Engineering

C09_RAD

C09_SDD

2 Packages

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

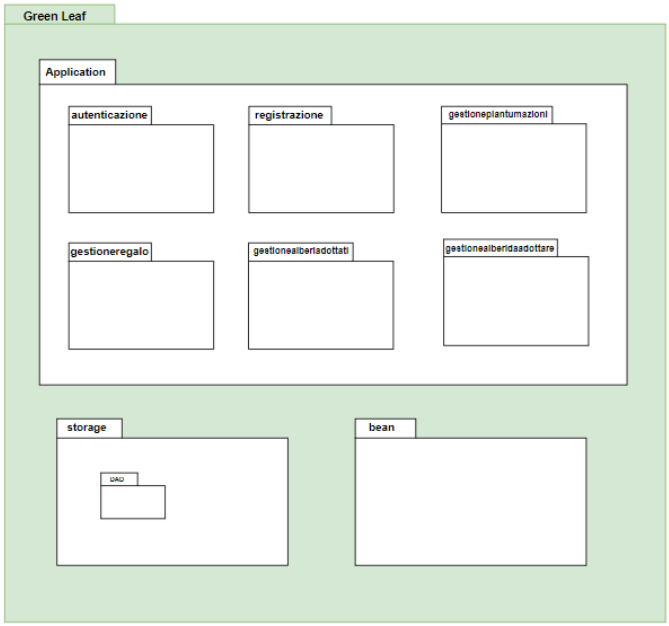
- **.idea**
- **.mvn**, contiene tutti i file di configurazione per Maven
- **librerieEsterne,** contiene il Connector/J
- **Presentation,** contiene i file di lavoro come: le pagine HTML CSS e JS, i png usati nel sistema
- **Script DB,** contiene lo script del Database SQL
- **src**, contiene tutti i file sorgente
  - main
    - **java**, contiene tutte le classi java relative alle componenti Application e Storage
      - **application,** contiene le Servlet utilizzate nel sistema
      - **bean,** contiene i bean per memorizzare le informazioni delle varie componenti nel sistema
      - **storage,** contiene le classi, metodi e query per modificare il Database
    - **webapp,** contiene tutte le componenti relative al Web
      - **risorse**, contiene tutti i file relativi alle componenti Presentation
        - **style**, contiene i fogli di stile CSS e le immagini essenziali ai fogli di stile
        - **js**, contiene gli script JS
        - **img**, contiene le immagini usate nelle pagine
  - **test**, contiene tutto il necessario per il testing
    - **storage**, contiene i test sulle componenti dello storage
    - **application,** contiene i test sulle componenti dell'application
- **target**, contiene tutti i file prodotti dal sistema di build di Maven

Package Green Leaf

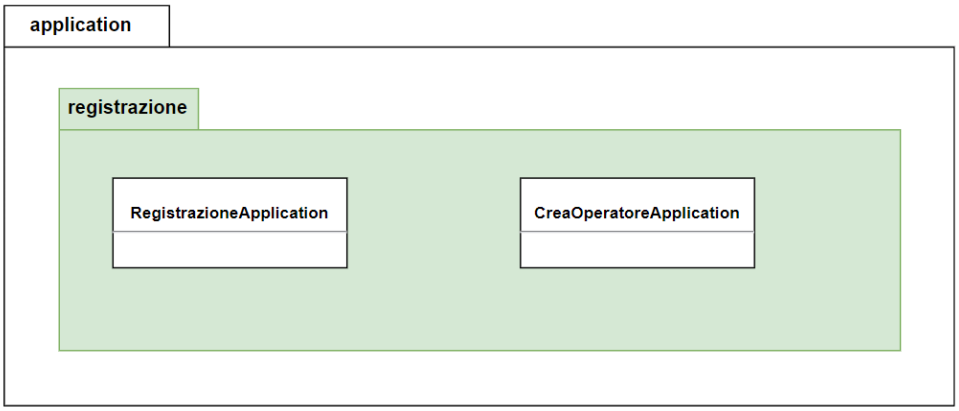
Nella presente sezione si mostra la struttura del package principale di Green Leaf. Questa struttura è stata ottenuta a partire da tre principali scelte:

1. Creare un package separato per ogni sottosistema, contenente le classi Application del suddetto, ed eventuali classi di utilità usate unicamente da esso.
2. Creare un package separato per le classi dello *storage*, contenente i **DAO** per l'accesso al DB. Tale scelta è stata presa vista l'elevata complessità del database di Green Leaf che prevede numerose relazioni tra le entità. Si è quindi preferito tenere tutto in un package separato e collegato a tutti gli altri package dei sottosistemi.
3. Creare un package chiamato *bean* in cui inserire le classi bean usabili da più sottosistemi.

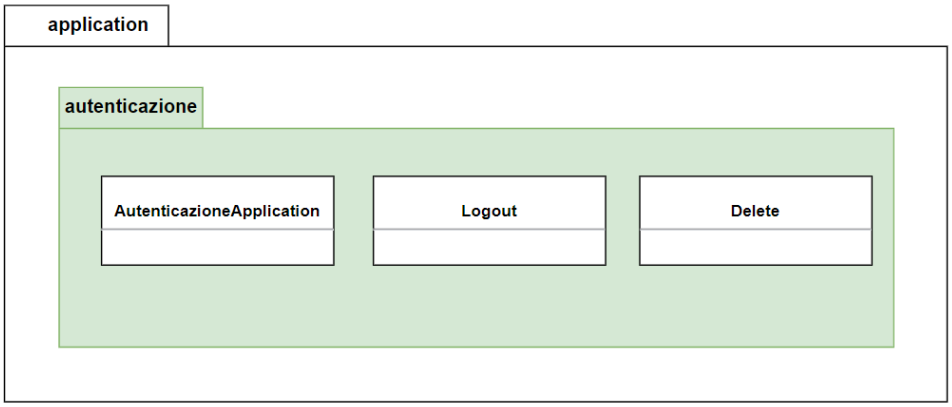
Per ciò che concerne la dipendenza tra i packages, la suddivisione precedentemente illustrata ha portato alla creazione di una relazione tra il package storage e tutti gli altri package del sistema. Di seguito sono illustrati solo i package che non sono puramente statici (ovvero che abbiano una Servelt).



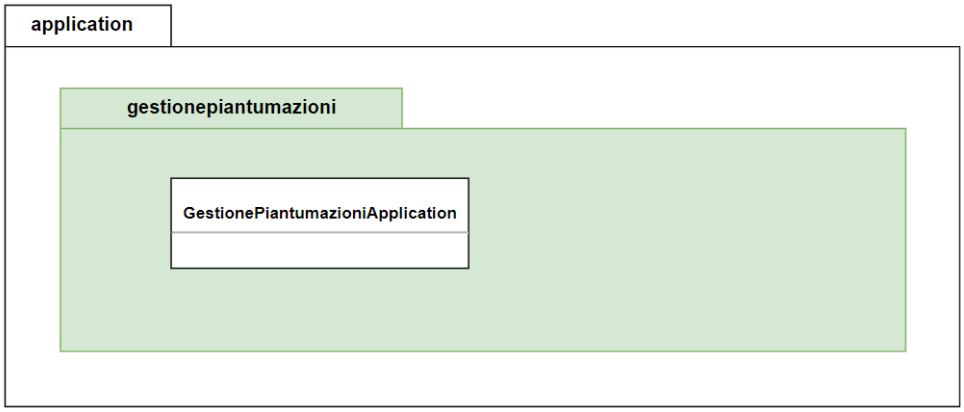
Package Registrazione



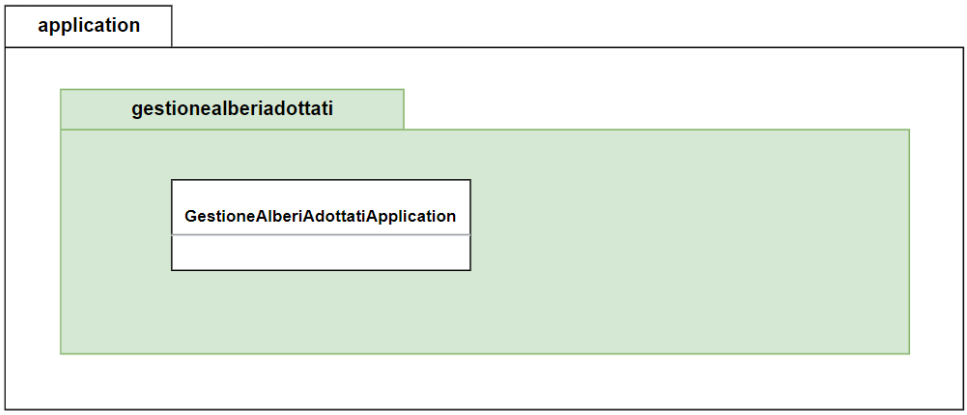
Package Autenticazione



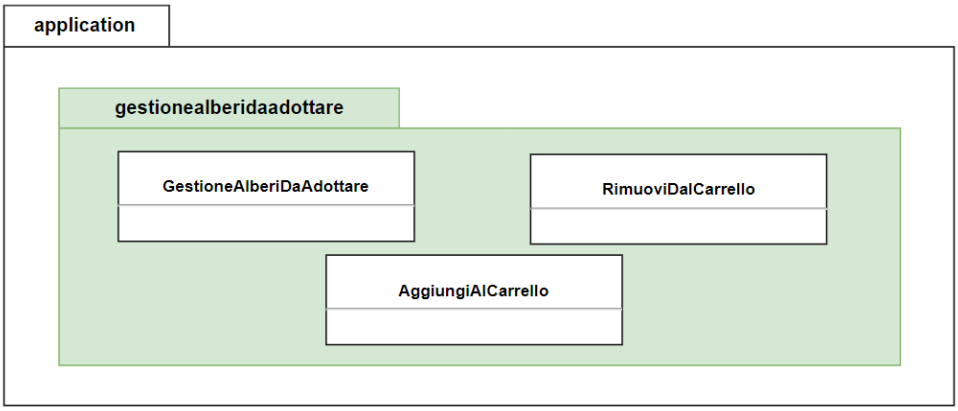
Package Gestione piantumazioni



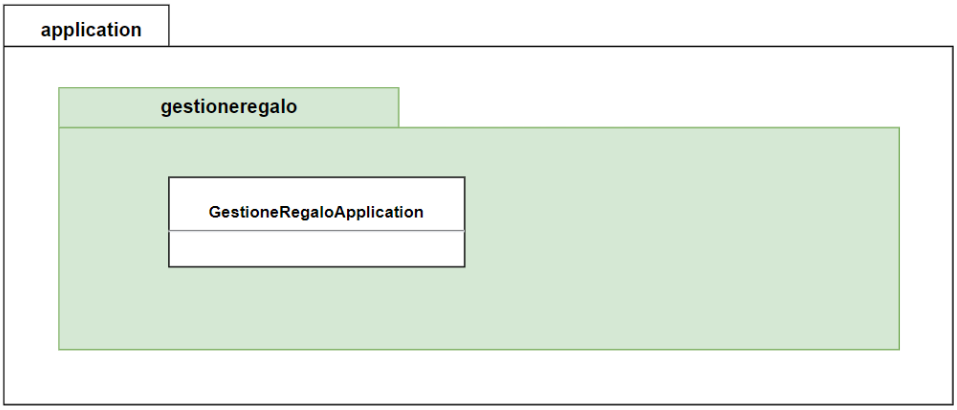
Package Alberi adottati



Package Gestione alberi da adottare



Package Gestione regalo



3 Class Interfaces

Package Application

1 Package Registrazione

Nome classe	RegistrazioneApplication
Descrizione	Questa classe permette di gestire le operazioni relative alla registrazione.
Metodi	+doGet(HttpServletRequest, HttpServletResponse): void +doPost(HttpServletRequest, HttpServletResponse): void
Invariante di classe	/

Nome metodo	+doGet(HttpServletRequest, HttpServletResponse): void
Descrizione	Questo metodo richiama la doPost.
Pre-condizione	L'utente non è registrato
Post-condizione	**post**: RegistrazioneApplication.doPost(request,response)

Nome metodo	+doPost(HttpServletRequest, HttpServletResponse): void
Descrizione	Questo metodo consente di registrare un nuovo utente.
Pre-condizione	L'utente non è registrato
Post-condizione	**context**: RegistrazioneApplication:: doPost (request,response) **post**: UtenteDAO.registrazione(utente) == true

Nome classe	CreaOperatoreApplication
Descrizione	Questa classe permette di gestire le operazioni relative alla registrazione di un nuovo operatore.
Metodi	+doGet(HttpServletRequest, HttpServletResponse): void +doPost(HttpServletRequest, HttpServletResponse): void
Invariante di classe	/

Nome metodo	+doGet(HttpServletRequest, HttpServletResponse): void
Descrizione	Questo metodo richiama la doPost.
Pre-condizione	• Colui che effettua la registrazione è un admin • L'operatore non è registrato
Post-condizione	**post**: CreaOperatoreApplication.doPost(request,response)

Nome metodo	+doPost(HttpServletRequest, HttpServletResponse): void
Descrizione	Questo metodo consente di registrare un nuovo operatore.
Pre-condizione	• Colui che effettua la registrazione è un admin • L'operatore non è registrato
Post-condizione	**context**: CreaOperatoreApplication:: doPost(request,response) **post**: OperatoreDAO.registrazione(Operatore) == true

2 Package Autenticazione

Nome classe	AutenticazioneApplication
Descrizione	Questa classe permette di gestire le operazioni relative all'autenticazione.
Metodi	+doGet(HttpServletRequest, HttpServletResponse): void +doPost(HttpServletRequest, HttpServletResponse): void
Invariante di classe	/

Nome metodo	+doGet(HttpServletRequest, HttpServletResponse): void
Descrizione	Questo metodo richiama la doPost.
Pre-condizione	L'utente non è loggato
Post-condizione	**post**: AutenticazioneApplication.doPost(request,response)

Nome metodo	+doPost(HttpServletRequest, HttpServletResponse): void
Descrizione	Questa classe permette di effettuare il login nel sistema.
Pre-condizione	L'utente non è loggato
Post-condizione	**context**: Autenticazione:: doPost (request,response) **post**: UtenteDAO.login(string,string) == true

Nome classe	Logout
Descrizione	Questa classe permette di sloggarsi dal sistema Green Leaf.
Metodi	+doGet(HttpServletRequest, HttpServletResponse): void +doPost(HttpServletRequest, HttpServletResponse): void
Invariante di classe	/

Nome metodo	+doPost(HttpServletRequest, HttpServletResponse): void
Descrizione	Questo metodo richiama la doGet.
Pre-condizione	A effettuare questa operazione è un attore del sistema loggato
Post-condizione	**post**: Logout.doGet(request,response)

Nome metodo	+doGet(HttpServletRequest, HttpServletResponse): void
Descrizione	Questo metodo permette di sloggarsi dal sistema Green Leaf.
Pre-condizione	A effettuare questa operazione è un attore del sistema loggato
Post-condizione	**context**: Logout:: doGet(request,response)

Nome classe	Delete
Descrizione	Questa classe permette a un attore del sistema di cancellare il suo account.
Metodi	+doGet(HttpServletRequest, HttpServletResponse): void +doPost(HttpServletRequest, HttpServletResponse): void
Invariante di classe	/

Nome metodo	+doPost(HttpServletRequest, HttpServletResponse): void
Descrizione	Questo metodo richiama la doGet.
Pre-condizione	Pre-condizione per eliminare un operatore: • L'admin è loggato • L'operatore da eliminare è di sua competenza Pre-condizione per eliminare un utente: • L'utente è loggato
Post-condizione	**post**: Delete.doGet(request,response)

Nome metodo	+doGet(HttpServletRequest, HttpServletResponse): void
Descrizione	Questo metodo permette all'admin di eliminare un operatore e all'utente di eliminare il proprio account.
Pre-condizione	Pre-condizione per eliminare un operatore: • L'admin è loggato • L'operatore da eliminare è di sua competenza Pre-condizione per eliminare un utente: • L'utente è loggato
Post-condizione	**context**: Delete:: doGet(request,response) **post**: OperatoreDAO.eliminaAccount(Operatore) **context**: Delete:: doGet(request,response) **post**: UtenteDAO.eliminaAccount(Utente)

3 Package Gestione piantumazioni

Nome classe	GestionePiantumazioniApplication
Descrizione	Questa classe permette di gestire le operazioni relative alla piantumazione.
Metodi	+doGet(HttpServletRequest, HttpServletResponse): void +doPost(HttpServletRequest, HttpServletResponse): void
Invariante di classe	/

Nome metodo	+doPost(HttpServletRequest, HttpServletResponse): void
Descrizione	Questo metodo richiama la doGet.
Pre-condizione	A effettuare questa operazione è un operatore
Post-condizione	**post**: GestionePiantumazioniApplication.doGet(request,response)

Nome metodo	+doGet(HttpServletRequest, HttpServletResponse): void
Descrizione	Questo metodo permette all'operatore modificare lo stato di un albero.
Pre-condizione	A effettuare questa operazione è un operatore
Post-condizione	**context**: GestionePiantumazioniApplication:: doGet(request,response) **post**: AlberoDAO.inserisciPiantumazioni(id)

4 Package Gestione alberi adottati

Nome classe	GestioneAlberiAdottatiApplication
Descrizione	Questa classe permette di gestire le operazioni relative agli alberi già adottati.
Metodi	+doGet(HttpServletRequest, HttpServletResponse): void +doPost(HttpServletRequest, HttpServletResponse): void
Invariante di classe	/

Nome metodo	+doPost(HttpServletRequest, HttpServletResponse): void
Descrizione	Questo metodo richiama la doGet.
Pre-condizione	A effettuare questa operazione è un utente
Post-condizione	**post**: GestioneAlberiAdottatiApplication.doGet(request,response)

Nome metodo	+doGet(HttpServletRequest, HttpServletResponse): void
Descrizione	Questo metodo permette all'utente di visualizzare tutti gli alberi da lui adottati.
Pre-condizione	A effettuare questa operazione è un utente
Post-condizione	**context**: GestioneAlberiAdottati:: visualizzaAlberiAdottati ()

5 Package Gestione alberi da adottare

Nome classe	GestioneAlberiDaAdottareApplication
Descrizione	Questa classe permette di gestire le operazioni relative agli alberi da adottare.
Metodi	+doGet(HttpServletRequest, HttpServletResponse): void +doPost(HttpServletRequest, HttpServletResponse): void
Invariante di classe	/

Nome metodo	+doGet(HttpServletRequest, HttpServletResponse): void
Descrizione	Questo metodo richiama la doPost.
Pre-condizione	A effettuare questa operazione è un utente
Post-condizione	**post**: GestioneAlberiDaAdottareApplication.doPost(request,response)

Nome metodo	+doPost(HttpServletRequest, HttpServletResponse): void
Descrizione	Questo metodo consente di comprare un albero o un buono regalo.
Pre-condizione	A effettuare questa operazione è un utente
Post-condizione	**context**: GestioneAlberiDaAdottareApplication:: doPost(request,response)

Nome metodo	+GeneraBuono(): String
Descrizione	Questo metodo consente di generare un buono regalo.
Pre-condizione	A effettuare questa operazione è una servlet
Post-condizione	/

Nome classe	AggiungiAlCarrello
Descrizione	Questa classe permette di gestire l'operazione di aggiunta al carrello.
Metodi	+doGet(HttpServletRequest, HttpServletResponse): void +doPost(HttpServletRequest, HttpServletResponse): void
Invariante di classe	/

Nome metodo	+doGet(HttpServletRequest, HttpServletResponse): void
Descrizione	Questo metodo richiama la doPost.
Pre-condizione	A effettuare questa operazione è un utente
Post-condizione	**post**: AggiungiAlCarrello.doPost(request,response)

Nome metodo	+doPost(HttpServletRequest, HttpServletResponse): void
Descrizione	Questo metodo consente di aggiungere un articolo al carrello.
Pre-condizione	A effettuare questa operazione è un utente
Post-condizione	**context**: AggiungiAlCarrello:: doPost(request,response)

Nome classe	RimuoviDalCarrello
Descrizione	Questa classe permette di gestire l'operazione di rimozione dal carrello.
Metodi	+doGet(HttpServletRequest, HttpServletResponse): void +doPost(HttpServletRequest, HttpServletResponse): void
Invariante di classe	/

Nome metodo	+doGet(HttpServletRequest, HttpServletResponse): void
Descrizione	Questo metodo richiama la doPost.
Pre-condizione	A effettuare questa operazione è un utente
Post-condizione	**post**: RimuoviDalCarrello.doPost(request,response)

Nome metodo	+doPost(HttpServletRequest, HttpServletResponse): void
Descrizione	Questo metodo consente di togliere un articolo dal carrello.
Pre-condizione	A effettuare questa operazione è un utente
Post-condizione	**context**: RimuoviDalCarrello:: doPost(request,response)

6 Package Gestioni regalo

Nome classe	GestioneRegaloApplication
Descrizione	Questa classe permette di gestire le operazioni relative ai buoni regalo.
Metodi	+doGet(HttpServletRequest, HttpServletResponse): void +doPost(HttpServletRequest, HttpServletResponse): void
Invariante di classe	/

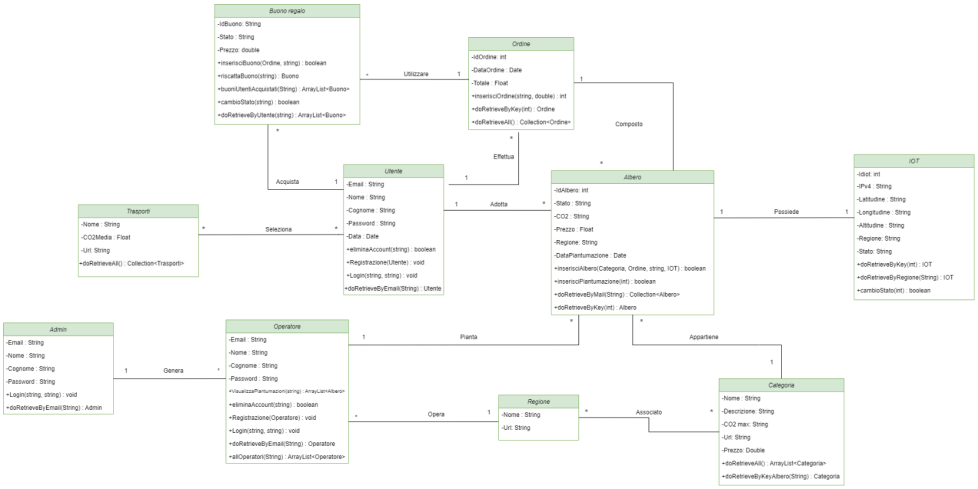
Nome metodo	+doGet(HttpServletRequest, HttpServletResponse): void
Descrizione	Questo metodo richiama la doPost.
Pre-condizione	A effettuare questa operazione è un utente
Post-condizione	**post**: GestioneRegaloApplication.doPost(request,response)

Nome metodo	+doPost(HttpServletRequest, HttpServletResponse): void
Descrizione	Questo metodo permette all'utente di riscattare un buono regalo.
Pre-condizione	A effettuare questa operazione è un utente
Post-condizione	**context**: GestioneRegaloApplication:: doPost(request,response) **post**: BuonoRegaloDAO.RiscattaBuono (string)

4 Class Diagram e Design Patterns

Di seguito viene riportato il Class Diagram ristrutturato:

Commentato [MF1]: Ci manca il CD



Design Patterns

Nella presente sezione si andranno a descrivere e dettagliare i design patterns utilizzati nello sviluppo dell'applicativo Green Leaf. Per ogni pattern si darà:

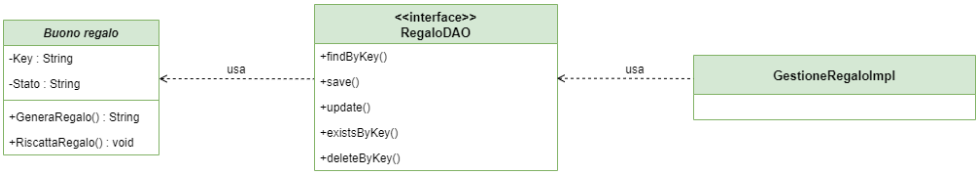
- Una brevissima introduzione teorica.
- Il problema che da risolvere all'interno del sistema.
- Una brevissima spiegazione di come si è risolto il problema.
- Un grafico della struttura delle classi che implementano il pattern.

DAO

Un DAO (Data Access Object) è un pattern che offre un'interfaccia astratta per alcuni tipi di database. Mappando le chiamate dell'applicazione allo stato persistente, il DAO fornisce alcune operazioni specifiche sui dati senza esporre i dettagli del database. I DAO sono utilizzabili nella maggior parte dei linguaggi e la maggior parte dei software con bisogni di persistenza, principalmente viene associato con applicazioni JavaEE che utilizzano database relazionali.

Essendo Green Leaf una Web Application che punta di gestire sia il monitoraggio dell'inquinamento che numerose tipologie di alberi, presenta un database molto vasto. Ha, quindi, bisogno di poter interagire con il database in modo rapido e sicuro utilizzando molte query. Per questo motivo si è scelto di utilizzare varie interfacce DAO all'interno del nostro sistema.

Qui è riportato un esempio di DAO utilizzato in Green Leaf e delle relazioni che ha con altre classi dell'applicazione.



Singleton

Singleton è un design pattern creazionale, ossia un design pattern che si occupa dell'istanziazione degli oggetti, che ha lo scopo di garantire che di una determinata classe venga strutturata una sola istanza e di fornire un punto di accesso globale a tale istanza.

5 Glossario

Termine	Definizione
Admin	Amministratore del sistema Green Leaf.
Operatore	Personale registrato a Green Leaf che effettua l'operazione di Piantumazione e può visionare informazioni formative.
Area personale	Un'area riservata a un qualsiasi utente che ha effettuato l'autenticazione, da cui può accedere a diverse funzionalità.
Piantumazione	Operazione che permette ad un qualsiasi albero adottato di essere piantato da un operatore.
Calcolo CO2 emessa	Operazione che permette ad un qualsiasi tipo di utente di calcolare la CO2 emessa durante l'arco della giornata.
Monitoraggio	Operazione che permette di mostrare la percentuale di inquinamento, odierno o futuro, di una determinata regione di Italia o dell'intera Nazione.
RDBMS	Un sistema di gestione di database relazionali (RDBMS) è un programma utilizzato per creare, aggiornare e gestire i database relazionali.