Object Design Document

GreenBridge

Riferimento	
Versione	2.0
Data	03/02/2024
Destinatario	Prof. F. Ferrucci
Presentato da	Mauro Pasquariello Michele Martino Giuseppe Di Sarno Salvatore Mattiello Giovanni De Gregorio Davide Califano El Mehdi Zitouni
Approvato da	

Revision History

Data	Versione	Descrizione	Autori
22/12/2023	0.1	Prima stesura	Mauro Pasquariello
22/12/2023	0.2	Stesura Cap. 1	Michele Martino Giovanni De Gregorio Salvatore Mattiello
29/12/2023	0.3	Stesura Cap. 2-4-5	Michele Martino Giovanni De Gregorio
30/01/2024	1.0	Completamento Cap. 2-4 Stesura Cap. 3	Michele Martino Giovanni De Gregorio
02/02/2024	1.1	Revisione Dipendenze Packages	Michele Martino Giovanni De Gregorio
03/02/2024	2.0	Revisione Finale	Michele Martino Giovanni De Gregorio

Sommario

1. Introduzione

Greenbridge si pone come obiettivo quello di agevolare la compravendita di beni agroalimentari fra agricoltori e clienti. L'ulteriore obiettivo è quello di convincere sempre più agricoltori all'utilizzo di pratiche agricole ecosostenibili.

1.1 Object Design Trade-offs

- **Riusabilità**: La concezione del sistema GreenBridge deve centrarsi sulla sua capacità di essere utilizzato in modo flessibile, sfruttando concetti di ereditarietà e design patterns.
- **Robustezza**: Il sistema deve dimostrare solidità, affrontando adeguatamente scenari inaspettati mediante il monitoraggio degli errori e la gestione delle situazioni eccezionali.
- **Incapsulamento**: Attraverso l'adozione di interfacce, il sistema assicura la riservatezza riguardo ai dettagli implementativi delle classi. Ciò consente l'utilizzo di funzionalità fornite da varie componenti o livelli in modo trasparente, trattandole come scatole nere.

Trade-off	Descrizione
Metodo di pagamento COTS vs metodo di pagamento proprietario	Si è deciso di utilizzare Nexi come gestore dei metodi di pagamento invece di un gestore proprietario, in modo da velocizzare la fase di sviluppo e sollevarci dalla responsabilità del testing. Va da sé che la gestione dei pagamenti dipende dall'affidabilità di Nexi.
Spring Boot vs linguaggio nativo back-end	Spring Boot risulta facile da configurare e si presta molto bene all'architettura scelta per il nostro sistema. Si evince che l'utilizzo di un framework come Spring Boot ha un impatto negativo sui tempi di risposta di avvio e chiusura del sistema.
Thymeleaf vs framework front-end	Thymeleaf è un template engine ben configurato con Spring e risulta molto meno oneroso di un qualsiasi altro framework per il front-end.
Database relazionale vs altre tecnologie	Sarà utilizzato un database relazionale perché meglio si presta alle esigenze del nostro sistema.

1.2 Linee guida per la Documentazione delle Interfacce

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 **Oracle Java Code Conventions** [Oracle, 1997].

Riferimenti alla documentazione ufficiale

- **Java Oracle**: https://www.oracle.com/technetwork/java/codeconventions-150003.pdf
- **HTML5 & CSS3**: https://google.github.io/styleguide/htmlcssguide.html

1.3 Definizioni, acronimi e abbreviazioni

- **Package**: raggruppamento di classi, interfacce o file correlati;
- **Design pattern**: schema progettuale che fornisce un approccio standardizzato per risolvere determinati tipi di problemi durante la progettazione e l'implementazione del software;
- **Interfaccia**: insieme di signature delle operazioni offerte dalla classe;
- **lowerCamelCase**: è la pratica di scrivere frasi in modo tale che ogni parola o abbreviazione inizi con una lettera maiuscola, senza spazi o punteggiatura intermedi, fatta eccezione per la prima parola;
- **UpperCamelCase**: è la pratica di scrivere frasi in modo tale che ogni parola o abbreviazione inizi con una lettera maiuscola, senza spazi o punteggiatura intermedi;
- **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 tutti i riferimenti ad altre documentazioni utili:

- Statement Of Work;
- Business Case;
- Requirements Analysis Document;
- System Design Document;
- Test Plan;
- Matrice di tracciabilità.

2. Packages

In questa sezione mostriamo la suddivisione del sistema in packages. Tale scelta è dettata dalle scelte architetturali e dalla struttura in directory fornita da Maven.

- **.idea**
- **.mvn**, contiene tutti i file di configurazione per Maven
- **chatbot**, contiene il server Flask
- **src**, contiene tutti i file sorgente
    - **main**
        - **java,** contiene le classi Java relative alle componenti Control e Model
        - **resources**, contiene i file relativi alle componenti View
            - **static**, contiene i fogli di stile CSS e gli script JS
            - **templates**, contiene i file HTML renderizzati da Thymeleaf
    - **test**
        - **java**, contiene le classi Java per l'implementazione del testing
- **target**, contiene tutti i file prodotti dal sistema di build di Maven

Vengono utilizzate le seguenti COTS:

- **XPay**, API messa a disposizione da NEXI
- **JPA**, Java Persistance API integrata in Spring che semplifica l'interazione tra il sistema e il database

Package Greenbridge

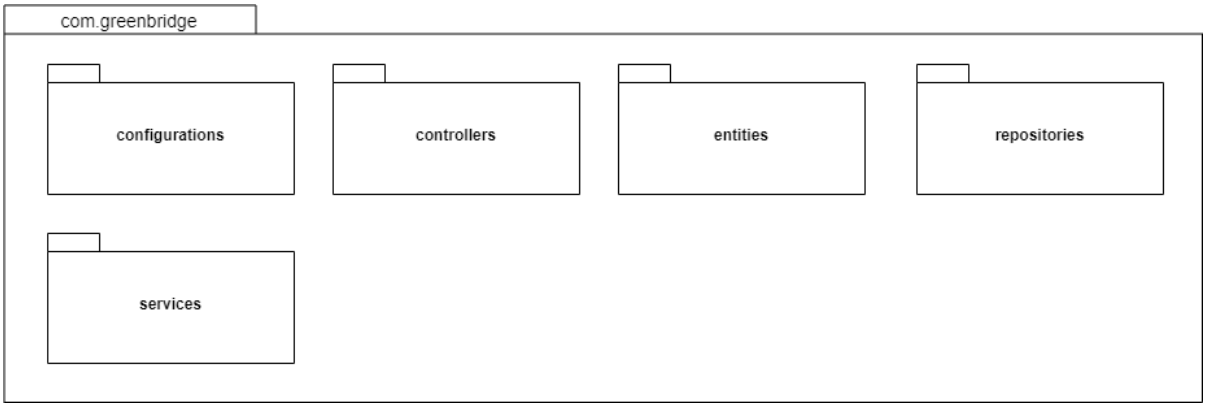
L'organizzazione è strettamente collegata all'aspetto prettamente implementativo. Di conseguenza, si è deciso di suddividere il package principale in:

- Controllers: dov'è presente la gestione di tutti i casi d'uso.
- Entities: dove sono presenti le entità che caratterizzano la base di dati della piattaforma.
- Repositories: dov'è gestita l'interazione tra il sistema e la base di dati.
- Services: dov'è gestita l'erogazione dei servizi per ogni entità.

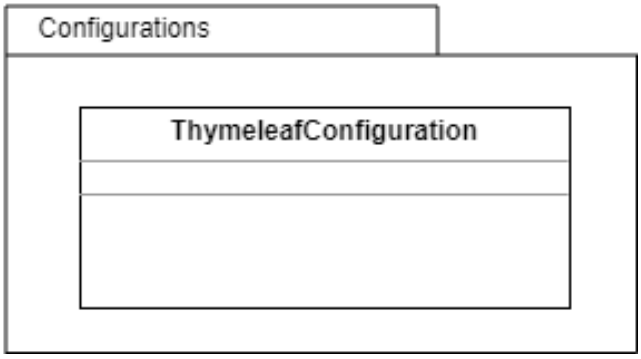
In base all'organizzazione dei package, evidenziamo le seguenti **dipendenze**:

- I Controller hanno una dipendenza con i Services.
- I Services hanno una dipendenza nei confronti dei Repositories.
- Tutti i package sono, infine, dipendenti dal package Entities.

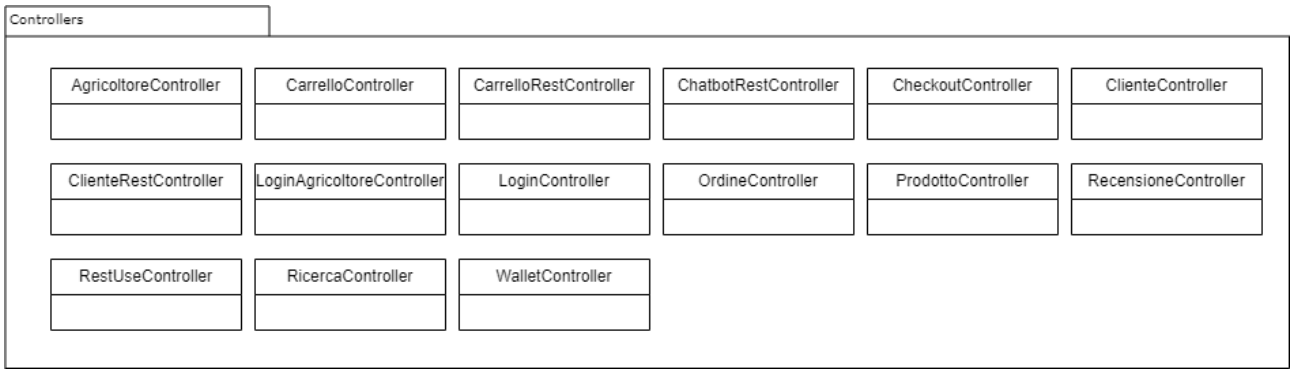
Si illustra la struttura del package generale di progetto con il seguente diagramma.



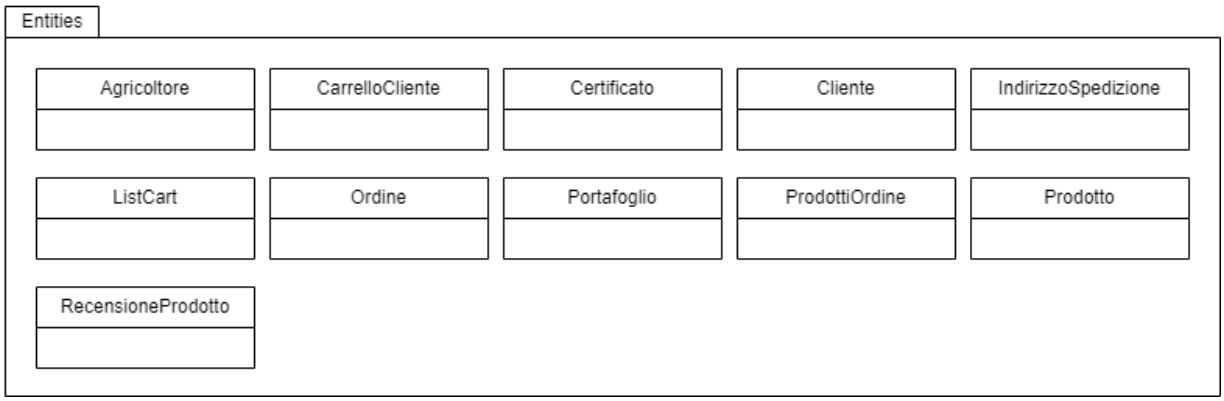
Package Configurations



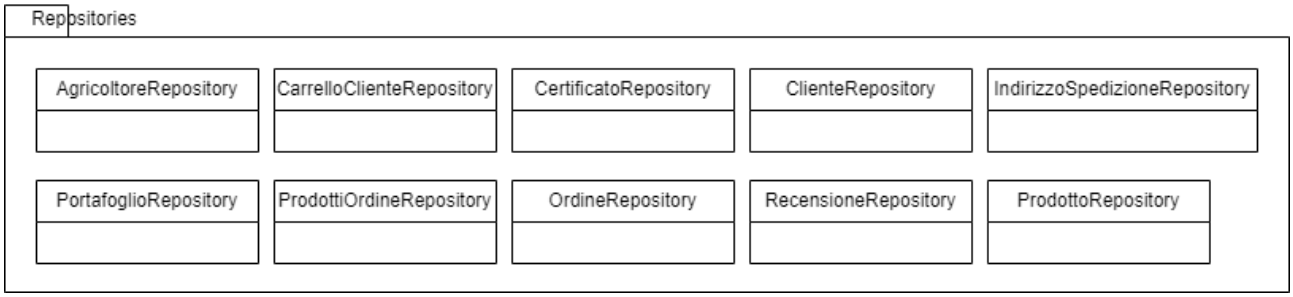
Package Controllers



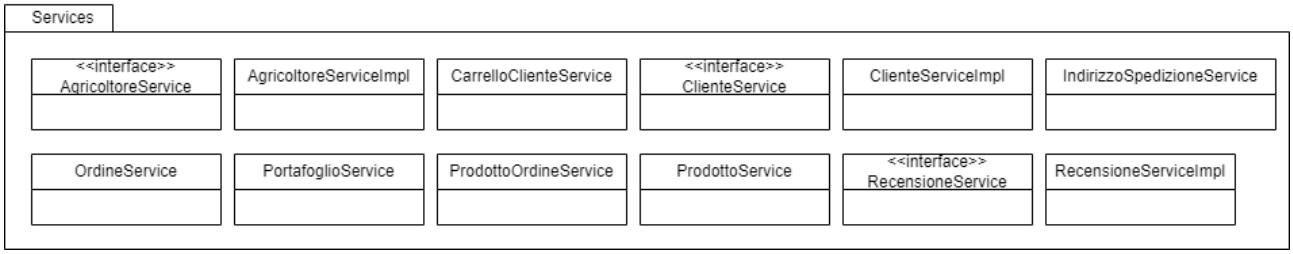
Package Entities



Package Repositories



Package Services



3. Class interfaces

A questo link è possibile consultare la JavaDoc di Greenbridge: Greenbridge JavaDoc.

4. Design Patterns

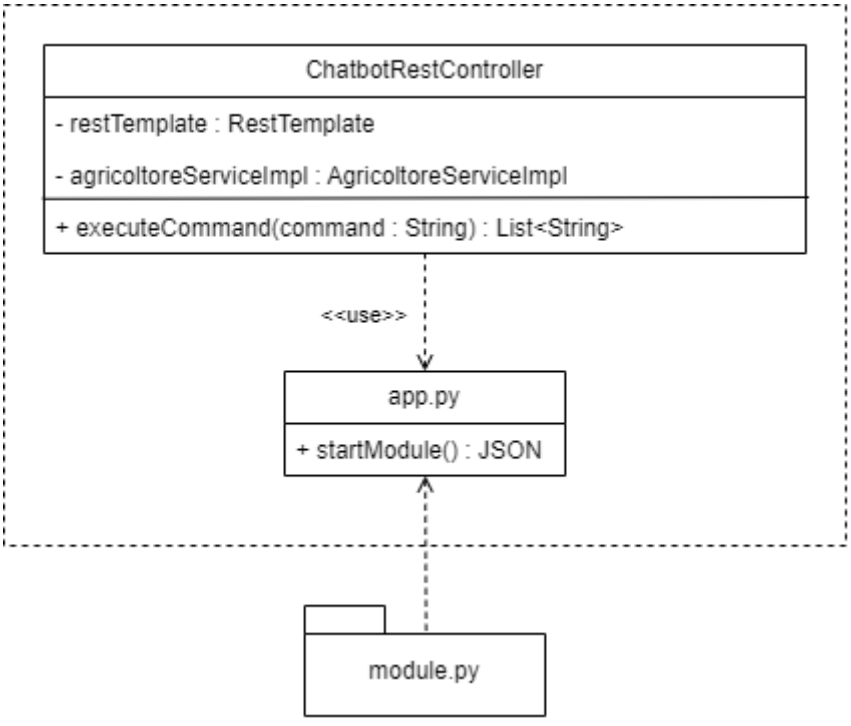
In questa sezione illustriamo i Design Pattern scelti per il sistema Greenbridge.

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.

Greenbridge fornisce una chatbot per la raccomandazione di agricoltori ai clienti, implementata con una tecnologia diversa da quella utilizza per Greenbridge (Flask). Le raccomandazioni vengono gestite attraverso un modulo esterno, contattato dal server Flask in risposta alle richieste da parte del sistema Greenbridge. Successivamente, sarà il server Flask a preoccuparsi di applicare l'Adapter per convertire i dati in un formato standard ampiamente utilizzato, come il JSON.

L'Adapter, di conseguenza, si preoccupa di convertire le risposte da parte del server Flask in oggetti JSON da inviare al server Spring Boot, in modo che siano processate.



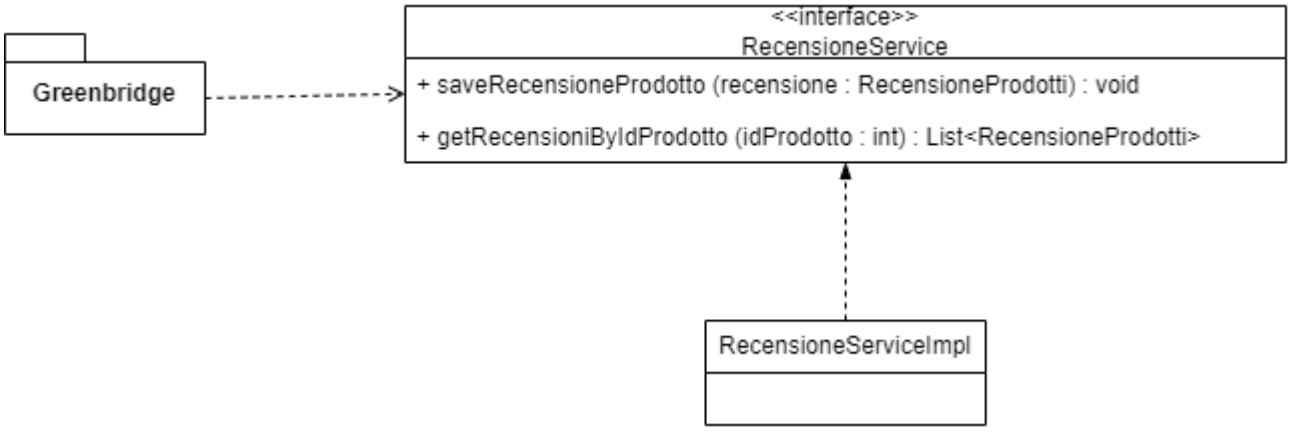
Facade

Il Facade è un design pattern strutturale che permette, tramite un'interfaccia semplificata, l'accesso a sottosistemi più complessi, così da nascondere al sistema la complessità dell'insieme di oggetti sottostanti. Questa cosa permette una gestione maggiormente semplificata di accesso alle

funzionalità del sistema, a favore del disaccoppiamento e, dunque, di una futura manutenzione dei metodi.

In Greenbridge, data la complessità del sistema, viene sfruttato il design pattern Facade per implementare la logica di business. In particolare, in Greenbridge i service possiedono delle interfacce per l'interazione con altri sottosistemi e ad esse sono associate le relative implementazioni.

Di conseguenza, con l'applicazione di questo design pattern, si punta ad astrarre i dettagli dei sottosistemi, in modo da renderli più facili da utilizzare. Di seguito, viene fornito un esempio di applicazione del Facade in Greenbridge.



5. Glossario

Nella seguente sezione saranno presentati i termini del documento che necessitano di una definizione.

Sigla/Termine	Definizione
Package	Raggruppamento di classi ed interfacce.
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.
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.
Facade	Un oggetto che permette, attraverso un'interfaccia più semplice, l'accesso a sottosistemi che espongono interfacce complesse e molto diverse tra loro.
JPA	Java Persistance API integrata in Spring che semplifica l'interazione tra il sistema e il database
XPay	API messa a disposizione da NEXI per effettuare i pagamenti online.
Flask	Micro-framework Web scritto in Python utilizzato per la chatbot.
JSON	Formato per lo scambio dati basato sul linguaggio di programmazione JavaScript.