ODD - Object Design Document

Riferimento	*2024_C10_ODD_beehAlve_V1.0* *2024_C10_RAD_beehAlve_V2.0* *2024_C10_SDD_beehAlve_V2.0* *2024_C10_TCS_beehAlve_V2.0*
Versione	1.0
Data	21/01/2024
Destinatario	Prof.ssa Filomena Ferrucci
Presentato da	N. Gallotta, F. Festa, S. Valente, A. De Pasquale, L. Milione, C. Boninfante

Approvato da	Nicolò Delogu, Dario Mazza

Composizione Gruppo	
Francesco Festa	05121-13547
Nicolò Gallotta	05121-14639
Andrea De Pasquale	05121-14909
Sara Valente	05121-14627
Lorenzo Milione	05121-14107
Carmine Boninfante	05121-13309

Cronologia Revisioni

Data	Versione	Descrizione	Autori
22/12/2023	0.1	Creazione Documento	Tutto il Team
23/12/2023	0.2	Stesura Introduzione e Glossario	Lorenzo Milione
23/12/2023	0.3	Stesura Design Patterns	Francesco Festa Andrea De Pasquale Carmine Boninfante Sara Valente Nicolò Gallotta
26/12/2023	0.4	Aggiunta Packages	Francesco Festa Carmine Boninfante
26/12/2023	0.5	Aggiunta Class Diagram	Francesco Festa Sara Valente Lorenzo Milione
27/12/2023	0.6	Stesura Interfacce di classe	Francesco Festa Andrea De Pasquale Sara Valente Nicolò Gallotta
21/01/2024	1.0	Revisione Finale	Nicolò Delogu Dario Mazza

Sommario

1. Introduzione

1.1 Object Design Goals

Rank	ID Design Goal	Descrizione	Origine
1	ODG_1 Dependability	Il sistema deve garantire un alto livello di affidabilità, allo scopo di evitare il più possibile errori nella raccolta dei dati e nella previsione delle anomalie. Ciò viene garantito da una minuziosa fase di valutazione e aggiornamento del modello e dalla scelta di sensori top di gamma nel mercato.	DG_6, DG_7
2	ODG_2 Security	Il sistema deve assicurare la sicurezza dei dati e la protezione da attacchi informatici attraverso tecniche di crittografia e il controllo degli accessi alle pagine. Ciò viene garantito dall'utilizzo del modulo Security di Spring e dall'utilizzo di SHA-256.	Design Trade-off
3	ODG_3 Cost	Il sistema, mantenendo un alto grado di leggibilità e comprensibilità, agevolerà gli interventi di manutenzione, garantendo una gestione efficiente dei costi di sviluppo.	DG_11 DG_12
4	ODG_4 Maintenance	Il sistema deve presentare un alto livello di chiarezza e comprensibilità, al fine di assicurare interventi di manutenzione veloci e semplici. Ciò viene garantito dalla corretta documentazione del codice.	DG_8, DG_9, DG_10
5	ODG_5 Accessibility	Il sistema deve presentare un facile livello di comprensione, al fine di garantire all'utente una comoda navigazione del sistema senza l'utilizzo di manuali.	DG_4, DG_5

1.2 Object Design Trade-Off

Trade-Off	Descrizione
Dependability vs Accessibility	Il sistema deve prioritariamente favorire l'affidabilità rispetto allo sviluppo di una piattaforma intuitiva e facile da utilizzare, poiché è cruciale assicurare il corretto funzionamento degli algoritmi utilizzati. Inoltre, nonostante la piattaforma rappresenti un'esperienza innovativa per l'apicoltore, quest'ultimo conosce esattamente ciò di cui ha bisogno, data la sua esperienza nel settore.
Security vs End-User	Il sistema deve privilegiare la sicurezza rispetto all'usabilità, implementando, ad esempio, meccanismi come l'autenticazione forte e la crittografia, anziché adottare procedure più semplici e tempi di accesso più veloci.
Utilizzo COTS vs Costi di Sviluppo	Il sistema farà ampio utilizzo di COTS disponibili sul mercato al fine di rientrare nel budget orario e di gestire funzionalità molto complesse, come la sicurezza e i pagamenti.

1.3 Definizioni, Acronimi e Abbreviazioni

- **ODD**
  - Object Design Document
- **Design Pattern**
  - Template di soluzioni a problemi ricorrenti impiegati per ottenere riuso e flessibilità.
- **Javadoc**
  - Sistema di documentazione offerto da Java, che viene generato sotto forma di interfaccia in modo da rendere la documentazione accessibile e facilmente leggibile.

1.4 Riferimenti

- 2023_C10_RAD
- 2023_C10_SDD
- 2023_C10_TCS

1.5 Component Off-the-Shelf

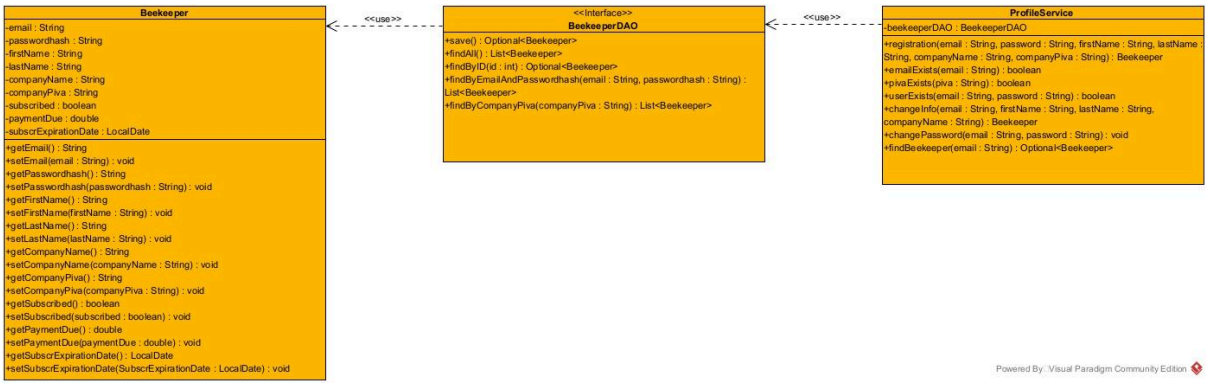
Per realizzare la nostra piattaforma siamo ricorsi all'utilizzo di componenti Off-the-Shelf già disponibili per semplificare lo sviluppo.

- Per la progettazione del lato back-end, abbiamo scelto il framework **Spring**, implementato in Java. In particolare, abbiamo usato **Spring Web MVC** (per separare le responsabilità tra la logica di business, la presentazione e la gestione delle richieste utente), **Spring Security** (per garantire la sicurezza in termini di autenticazione, autorizzazione e protezione delle risorse), **Spring Data JPA** (per semplificare l'accesso ai dati) e **Thymeleaf** (per agevolare l'invio di dati da Java alle pagine HTML).
- Per l'integrazione dei servizi di pagamento, abbiamo adottato l'**API** di **PayPal**, che fornisce un modo affidabile per effettuare pagamenti online.
- Per la comunicazione con il DBMS MySQL, abbiamo utilizzato la comunicazione **JDBC** per gestire e manipolare i dati conservati nel database.
- Per standardizzare lo sviluppo Java e semplificare la gestione del progetto, ci siamo avvalsi di **Maven**, uno strumento che mira a semplificare la compilazione, la distribuzione e la gestione delle dipendenze.
- Per l'implementazione del modulo di Intelligenza Artificiale, abbiamo utilizzato le librerie **SciKit Learn** e **Keras** per operazioni come la creazione e configurazione della rete neurale, l'addestramento e la valutazione.
- Per il caricamento, preparazione e manipolazione dei dati sfruttati dal modulo di Intelligenza Artificiale, abbiamo scelto le librerie **Pandas** e **NumPy**.
- Per l'integrazione del modulo di Intelligenza Artificiale nel sistema, abbiamo scelto il framework **Flask**.
- Per la progettazione del lato front-end della piattaforma, abbiamo sfruttato **Bootstrap**, una libreria dedicata allo sviluppo Web dotata di script basati principalmente su HTML, CSS e JavaScript, rivolti alla realizzazione di diversi tipi di componenti che caratterizzano l'interfaccia grafica quali pulsanti, finestre, moduli e altri.

1.6 Design Patterns

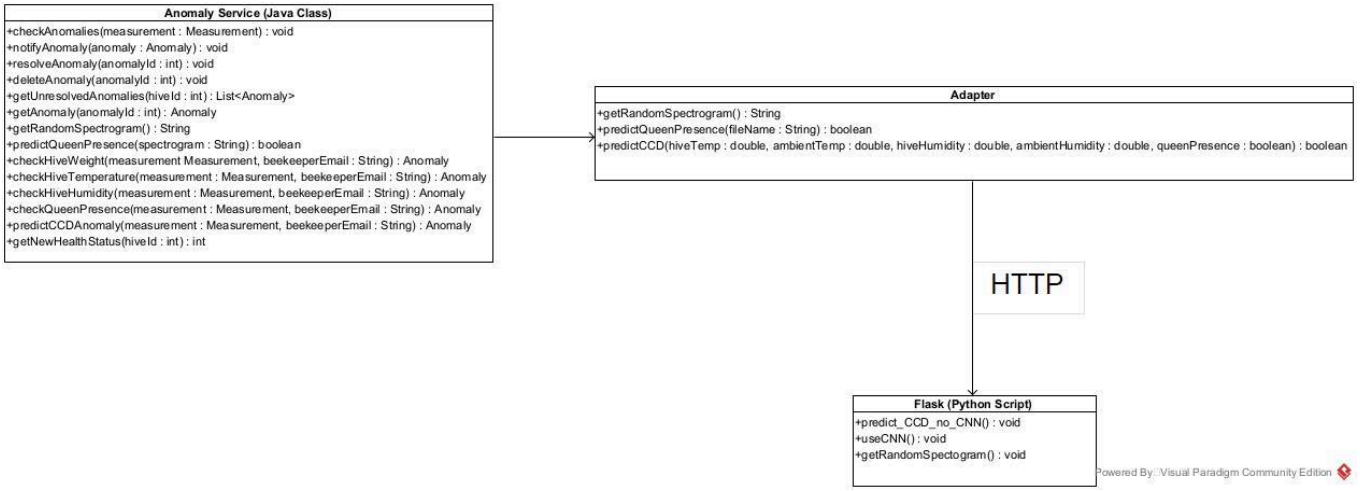
1.6.1 DAO

Il **DAO** (Data Access Object) è un design pattern strutturale che consente di dividere il layer di persistenza dal layer di logica di business attraverso l'utilizzo di una API astratta e delegando a degli oggetti specifici l'onere delle operazioni CRUD all'interno del Database, senza però esporre i dettagli di esso.
Essendo il nostro sistema una web application che sfrutta un'architettura Three-Tier, il DAO è ideale per separare nettamente il secondo e il terzo tier, senza mischiare le query al database con la logica di back-end e velocizzando di conseguenza lo sviluppo. Inoltre, un ulteriore vantaggio che si può ottenere è quello di poter sviluppare e testare gli ultimi due tier in maniera indipendente.
Riportiamo di seguito un esempio.



1.6.2 Adapter

L'**Adapter** è un design pattern strutturale che permette ad oggetti o sottosistemi con interfacce diverse di comunicare tra loro, convertendo i dati di una nel formato leggibile all'altra.
All'interno del nostro sistema è centrale la possibilità di diagnosticare anomalie, come il CCD, attraverso il Machine Learning, ma questa funzionalità verrà implementata in Python, il quale non ha un'interfaccia compatibile col framework Java Spring. Per risolvere questo problema utilizzeremo l'Adapter così da convertire le misurazioni, effettuate dai sensori IoT, da oggetto Java a un formato leggibile dal modello, per poi convertire la previsione di quest'ultimo di nuovo in un formato leggibile da Java.
Riportiamo di seguito l'implementazione di tale meccanismo.



1.7 Linee guida per la documentazione delle interfacce

Si è deciso di adottare le convenzioni standardizzate di Google:

- Java: https://google.github.io/styleguide/javaguide.html
- HTML/CSS: https://google.github.io/styleguide/htmlcssguide.html
- Javascript: https://google.github.io/styleguide/jsguide.html

2. Packages

In questa sezione è presentata la struttura dei packages del progetto, determinata da alcune scelte:

- Creare i package **Controller** e **Service** che contengono a loro volta i package dedicati ai diversi sottosistemi;
- Creare un package **Persistence** che contiene il package che contiene tutti i DAO e il package che contiene tutte le Entities;
- Creare un package **Misc** che contiene eventuali classi di utilità per il sistema, utilizzabili da diversi sottosistemi.

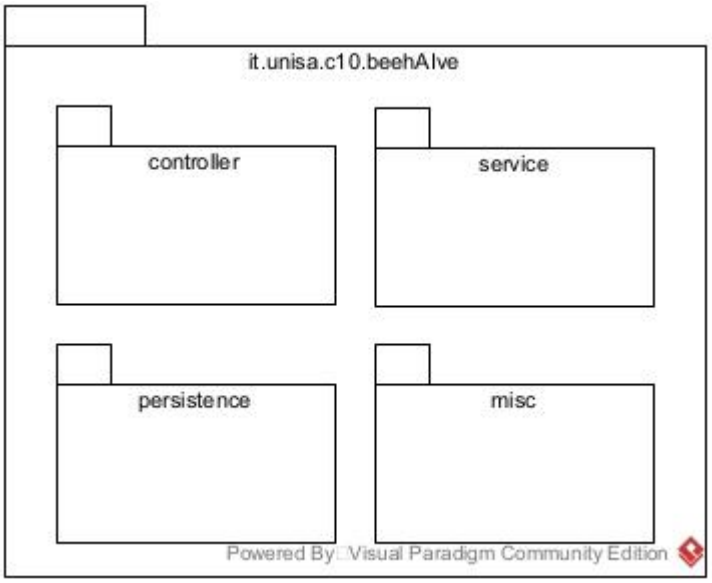
2.1 Struttura Directory

- **.idea**
- **src**: contiene i file sorgente
    - **ai**
        - **dataset**: contiene i dataset utilizzati per il modello di Machine Learning e la CNN
        - **development**
            - **CNN**: contiene i moduli per la realizzazione della CNN
            - **ML**: contiene i moduli per la realizzazione del modelli di Machine Learning
        - **logs**: contiene il log prodotto dalla validazione della CNN
        - **resources**: contiene gli spettrogrammi forniti alla CNN, derivati dai file audio del dataset
    - **db**: contiene il database MySQL
    - **main**
        - **java**: contiene le classi Java relative alla logica di business, alla logica di gestione degli eventi e alla persistenza
            - **it.unisa.c10.beehAIve**
                - **controller**: contiene le classi controller per ogni sottosistema
                - **gestioneAnomalie**: contiene il controller che gestisce la comunicazione con l'adapter e i parametri della misurazione
                - **gestioneArnia**
                - **gestioneDashboard**: contiene il controller per gestire la dashboard
                - **gestioneInterventi**: contiene i controller per gestire gli interventi
                - **gestioneStato**: contiene il controller per gestire lo stato di salute dell'arnia

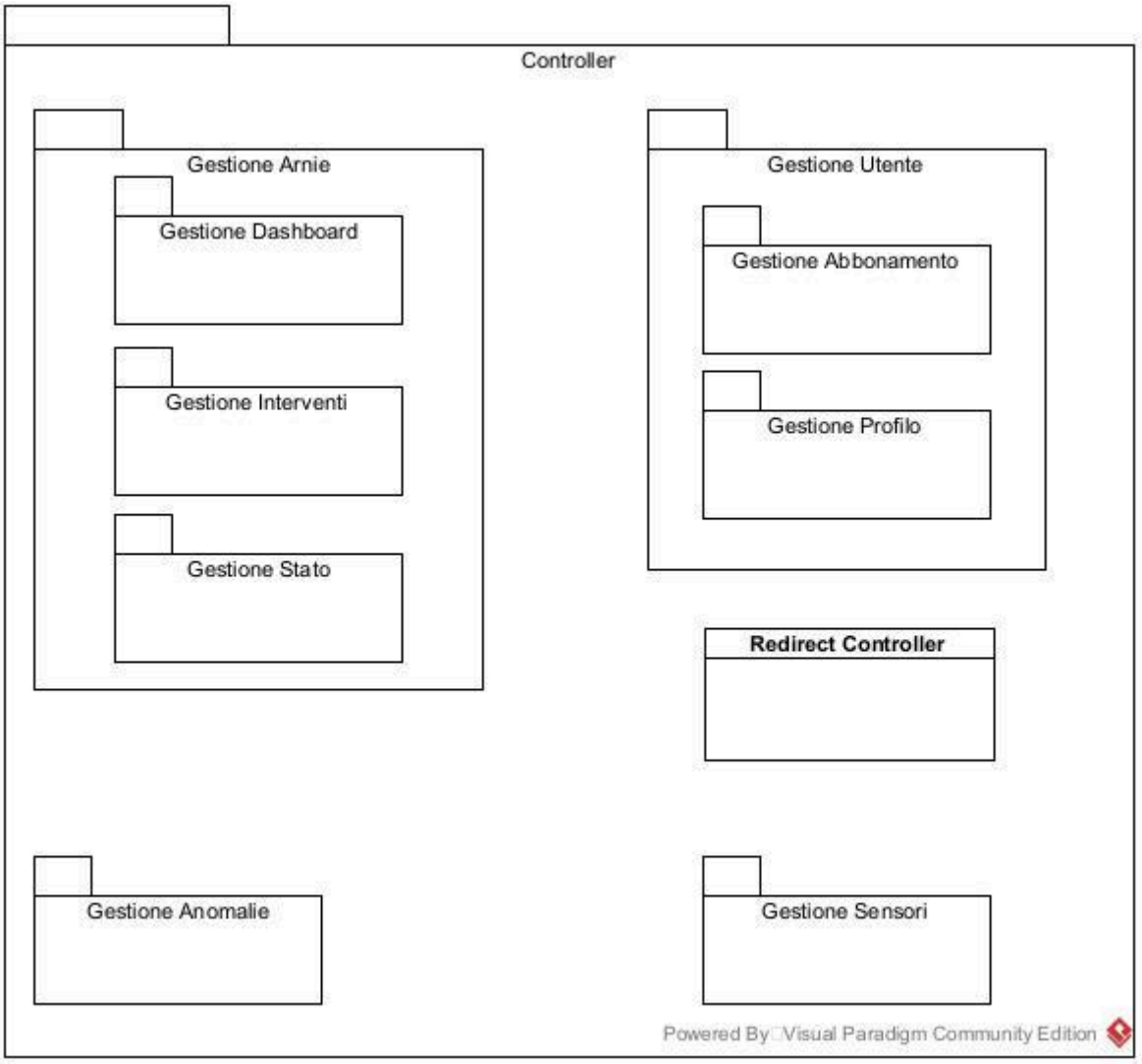
        - **gestioneSensori**: contiene il controller dedicato alla simulazione dei sensori IoT
        - **gestioneUtente**
            - **gestioneAbbonamento**: contiene il controller per la gestione dell'abbonamento con PayPal
            - **gestioneProfilo**: contiene i controller per il login, la registrazione e la modifica delle informazioni personali
    - **misc**: contiene l'adapter, la configurazione di PayPal e della sicurezza del sistema
    - **persistence**: contiene i DAO e le Entities per comunicare con il database
        - **dao**
        - **entities**
    - **service**: contiene le classi Service per ogni sottosistema
        - **gestioneAnomalie**: contiene il service per il controllo dei parametri misurati sull'arnia
        - **gestioneArnie**: contiene i service per la dashboard, per la gestione degli Interventi, per la generazione dei grafici e del report
        - **gestioneSensori**: contiene il service per la simulazione delle misurazioni dei sensori
        - **gestioneUtente**: contiene i service per la gestione del profilo (registrazione, login, modifica informazioni personali) e dell'abbonamento
- **resources**: contiene le pagine e gli script per il frontend
    - **static**
        - **assets**: contiene le immagini presenti nel sistema
        - **Bootstrap**
            - **css**: contiene gli script CSS della libreria
            - **js**: contiene gli script JS della libreria
        - **css**: contiene tutti gli script CSS utilizzati dal sistema
            - **error**: contiene tutti gli style delle pagine d'errore
            - **fragments**: contiene tutti gli style dei frammenti
            - **hive**: contiene tutti gli style delle pagine relative alle arnie

        - **js**: contiene gli script JavaScript utilizzati dal sistema e la libreria jQuery
    - **templates**: contiene tutte le pagine HTML che costituiscono il sistema
        - **error**: contiene tutte le pagine d'errore del sistema
        - **fragments**: contiene tutti i componenti aggiuntivi delle pagine, come header o footer
        - **hive**: contiene tutte le pagine relative alle arnie
        - **payments**: contiene le pagine relative al pagamento
- **test**
    - **it.unisa.c10.beehAlve**: contiene le classi Java dedicate ai test di unità
- **target**: contiene i file prodotti da Maven (sistema di build)

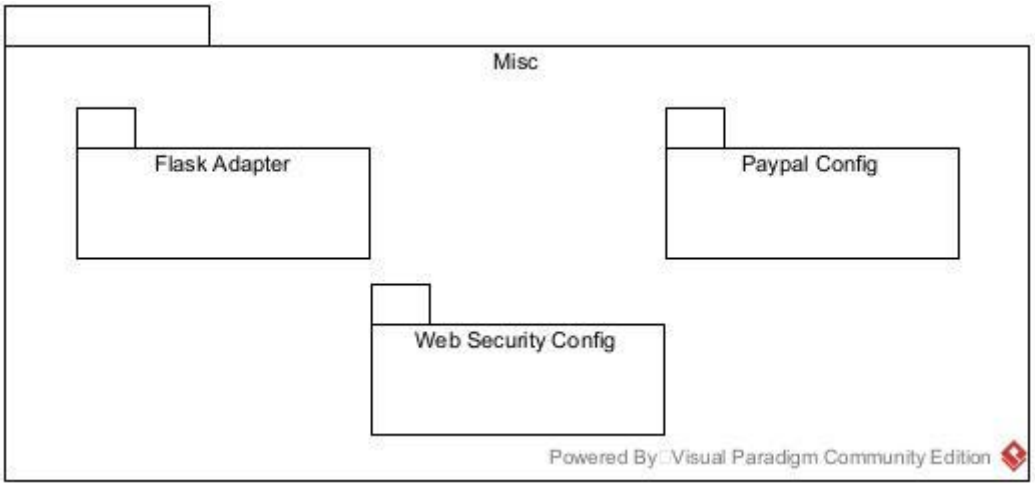
2.1 Package BeehAlve



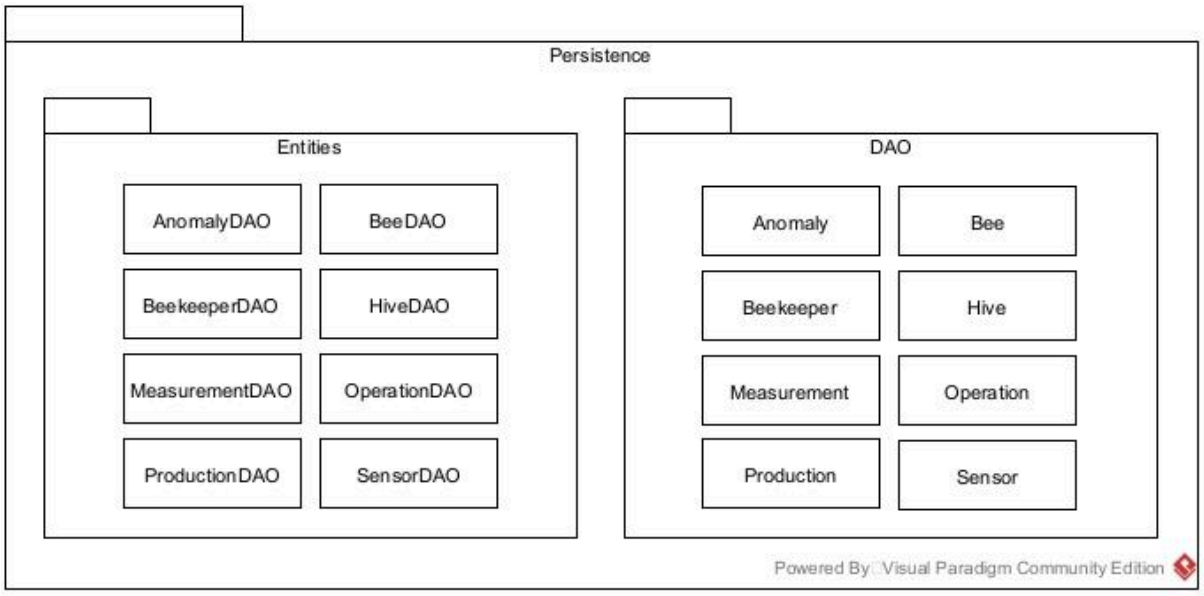
2.2 Package Controller



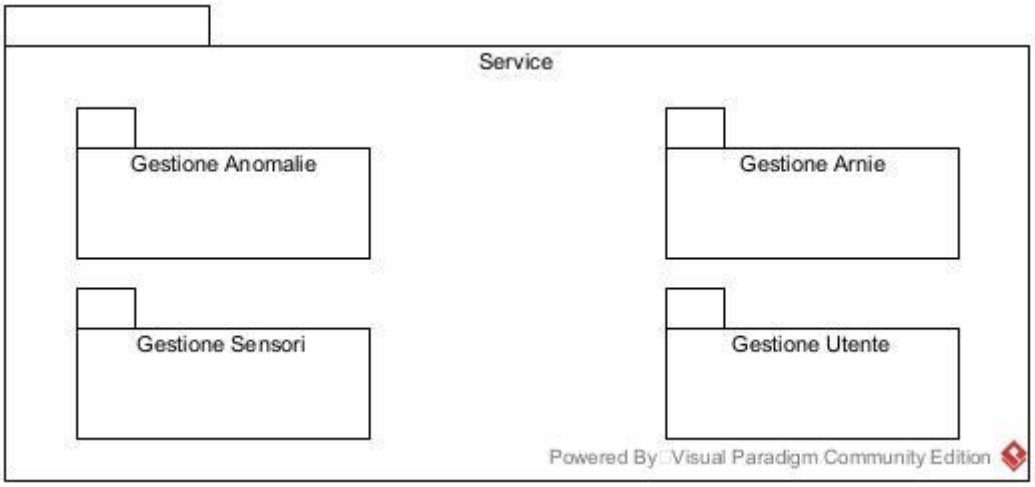
2.3 Package Misc



2.4 Package Persistence



2.5 Package Service



3. Class Interfaces

Riportiamo di seguito i sottosistemi che costituiscono il sistema con i metodi più caratterizzanti.

Per consultare la documentazione JavaDoc: consultare questo link.

3.1 Package Gestione Anomalie

Nome Classe	**AnomalyService**
Descrizione	Questa classe permette di interfacciarsi con il sottosistema per la gestione delle anomalie.
Metodi	+checkAnomalies(Measurement measurement): void +notifyAnomaly(Anomaly anomaly): void +resolveAnomaly(int anomalyId): void +deleteAnomaly(id: int): void +predictQueenPresence(spectrogram: String): boolean
Invarianti di classe	N/A

Nome Metodo	**+checkAnomalies(Measurement measurement): void**
Descrizione	Analizza la misurazione per rilevare eventuali anomalie.
Pre-condizione	**Context:** AnomalyService::checkAnomalies(Measurement measurement): void **Pre:** measurement <> null
Post-condizione	/

Nome Metodo	**+notifyAnomaly(Anomaly anomaly): void**
Descrizione	Modifica lo stato dell'arnia in base all'anomalia rilevata.
Pre-condizione	**Context:** AnomalyService::notifyAnomaly(Anomaly anomaly): void **Pre:** anomaly <> null
Post-condizione	**Context:** AnomalyService::notifyAnomaly (Anomaly anomaly): void **Post**: anomaly.getHiveId().getHive().getHiveHealth() = 2 or anomaly.getHiveId().getHive().getHiveHealth() = 3

Nome Metodo	**+resolveAnomaly(int anomalyId): void**
Descrizione	Contrassegna l'anomalia come risolta.
Pre-condizione	**Context:** AnomalyService::resolveAnomaly(int anomalyId): void **Pre:** anomalyId > 0
Post-condizione	**Context:** AnomalyService::resolveAnomaly (int anomalyId): void **Post**: getAnomaly(int anomalyId).resolved = true and anomaly.getHiveId().getHive().getUnresolvedAnomalies() = @pre anomaly.getHiveId().getHive().getUnresolvedAnomalies() - 1

Nome Metodo	**+deleteAnomaly(id: int): void**
Descrizione	Cancella l'anomalia e aggiorna lo stato di salute dell'arnia.
Pre-condizione	**Context:** AnomalyService::deleteAnomaly(int id): void **Pre:** id > 0 and Anomaly.getUnresolvedAnomalies(Anomaly.getHiveId()) -> exists (a | a.getId() = id)
Post-condizione	**Context:** AnomalyService::deleteAnomaly(int id): void **Post:** getAnomaly(id) = null

Nome Metodo	**+predictQueenPresence(spectrogram: String): boolean**
Descrizione	Fa una previsione della presenza dell'ape regine in base allo spettrogramma.
Pre-condizione	**Context:** AnomalyService::predictQueenPresence(String spectrogram): boolean **Pre:** spectrogram <> null
Post-condizione	/

3.2 Package Gestione Arnie

Nome Classe	DashboardService
Descrizione	Questa classe permette di interfacciarsi con il sottosistema per la gestione delle arnie, in particolare riguardo alla dashboard.
Metodi	+createHive(String beekeeperEmail, String nickname, String hiveType, String beeSpecies): void +modifyHive(int hiveId, String nickname, String hiveType, String beeSpecies): void +deleteHive(id: int): void +getHive(int hiveId): Hive +getBeekeeperHives(String beekeeperEmail): List<Hive> +deleteHive(int id): void
Invarianti di classe	N/A

Nome Metodo	+createHive(String beekeeperEmail, String nickname, String hiveType, String beeSpecies): void
Descrizione	Permette di creare una nuova arnia.
Pre-condizione	**Context:** DashboardService::createHive(String beekeeperEmail, String nickname, String hiveType, String beeSpecies): void **Pre**: beekeeperEmail <> null and nickname <> null and hiveType <> null and beeSpecies <> null and ProfileService::emailExists(beekeeperEmail) = true
Post-condizione	**Context:** DashboardService::createHive(String beekeeperEmail, String nickname, String hiveType, String beeSpecies): void **Post**: getBeekeeperHives(beekeeperEmail) = @pre getBeekeeperHives(beekeeperEmail) + 1

Nome Metodo	**+modifyHive(int hiveId, String nickname, String hiveType, String beeSpecies): void**
Descrizione	Permette di modificare un'arnia.
Pre-condizione	**Context:** DashboardService::modifyHive(int hiveId, String nickname, String hiveType, String beeSpecies): void **Pre**: beekeeperEmail <> null and nickname <> null and hiveType <> null and beeSpecies <> null and ProfileService::emailExists(beekeeperEmail) = true
Post-condizione	/

Nome Metodo	**+deleteHive(int id): void**
Descrizione	Permette di modificare un'arnia.
Pre-condizione	**Context:** DashboardService::deleteHive(int id): void **Pre**: id <> null and HiveDAO::findAll() -> exists(h | h.getId() = id)
Post-condizione	**Context:** DashboardService::+deleteHive(int id): void **Post:** getHive(id) = null

Nome Metodo	**+getHive(int hiveId): Hive**
Descrizione	Permette di ottenere l'arnia con tutte le sue informazioni.
Pre-condizione	**Context:** DashboardService::getHive(int hiveId): Hive **Pre**: hiveId <> null and HiveDAO::findAll() -> exists(h | h.getId() = id)
Post-condizione	**Context:** DashboardService::getHive(int hiveId): Hive **Post**: Hive <> null

Nome Metodo	**+getBeekeeperHives(String beekeeperEmail): List<Hive>**
Descrizione	Permette di ottenere tutte le arnie possedute da un Beekeeper.
Pre-condizione	**Context:** DashboardService::getBeekeeperHives(String beekeeperEmail): List<Hive> **Pre:** beekeeperEmail <> null and ProfileService::emailExists(beekeeperEmail) = true
Post-condizione	/

Nome Classe	**OperationService**
Descrizione	Questa classe permette di interfacciarsi con il sottosistema per la gestione delle arnie, in particolare riguardo agli interventi.
Metodi	+planningOperation(String operationName, String operationType, String operationStatus, LocalDateTime operationDate, String notes, int hiveId, String beekeeperEmail): void +modifyScheduledOperation(int id, String operationName, String operationType, String operationStatus, LocalDateTime operationDate, String notes, int hiveId, String beekeeperEmail): void +cancelScheduledOperation(int id): void +viewHiveOperations(int hiveId): List<Operation> +viewAllBeekeperOperations(String beekeeperEmail): List<Operation>
Invarianti di classe	N/A

Nome Metodo	**+planningOperation(String operationName, String operationType, String operationStatus, LocalDateTime operationDate, String notes, int hiveId, String beekeeperEmail): void**
Descrizione	Permette di pianificare l'intervento sull'arnia.
Pre-condizione	**Context:** OperationService::planningOperation(String operationName, String operationType, String operationStatus, LocalDateTime operationDate, String notes, int hiveId, String beekeeperEmail): void **Pre:** operationName <> null and operationType <> null and operationStatus <> null and operationDate <> null and hiveId <> null and beekeeperEmail <> null and hiveId -> exists(hive.getId = hiveId) and ProfileService::emailExists(beekeeperEmail) = true
Post-condizione	**Context:** OperationService::planningOperation(String operationName, String operationType, String operationStatus, LocalDateTime operationDate, String notes, int hiveId, String beekeeperEmail): void **Post**: getHive(hiveId).uncompletedOperations = true and getHiveUncompletedOperations(hiveId) = @pre getHiveUncompletedOperations(hiveId) + 1

Nome Metodo	**+modifyScheduledOperation(int id, String operationName, String operationType, String operationStatus, LocalDateTime operationDate, String notes, int hiveId, String beekeeperEmail): void**
Descrizione	Permette di modificare l'intervento sull'arnia.
Pre-condizione	**Context:** OperationService::modifyScheduledOperation(int id, String operationName, String operationType, String operationStatus, LocalDateTime operationDate, String notes, int hiveId, String beekeeperEmail): void **Pre**: operationName <> null and operationType <> null and operationStatus <> null and operationDate <> null and hiveId <> null and beekeeperEmail <> null and hiveId -> exists(hive.getId = hiveId) and ProfileService::emailExists(beekeeperEmail) = true
Post-condizione	/

Nome Metodo	**+cancelScheduledOperation(int id): void**
Descrizione	Permette di modificare l'intervento sull'arnia.
Pre-condizione	**Context:** OperationService::cancelScheduledOperation(int id) **Pre**: id <> null and retrieveOperationFromDB(id) <> null
Post-condizione	**Context:** OperationService::cancelScheduledOperation(int id) **Post**: retrieveOperationFromDB(id) = null

Nome Metodo	**+viewHiveOperations(int hiveId): List<Operation>**
Descrizione	Permette di modificare l'intervento sull'arnia.
Pre-condizione	**Context:** OperationService::viewHiveOperations(int hiveId): List<Operation> **Pre**: hiveId <> null and HiveDAO::findAll() -> exists(h | h.getId() = hiveId)
Post-condizione	/

Nome Metodo	+viewAllBeekeperOperations(String beekeeperEmail): List<Operation>
Descrizione	Permette di modificare l'intervento sull'arnia.
Pre-condizione	**Context:** OperationService::viewAllBeekeperOperations(String beekeeperEmail): List<Operation> **Pre**: beekeeperEmail <> null and ProfileService::emailExists(beekeeperEmail) = true
Post-condizione	/

3.3 Package Gestione Sensori

Nome Classe	StatusService
Descrizione	Questa classe permette di interfacciarsi con il sottosistema per la gestione delle arnie, in particolare riguardo alla salute dell'arnia.
Metodi	+getGraphData(int hiveId): List<ArrayList<Object>> +generateReport(int hiveId, HttpServletResponse response): void
Invarianti di classe	N/A

Nome Metodo	+getGraphData (int hiveId): List<ArrayList<Object>>
Descrizione	Permette di generare una stringa JSON sui parametri dell'arnia.
Pre-condizione	**Context:** StatusService::getGraphData(int hiveId): List<ArrayList<Object>> **Pre:** hiveId <> null and HiveDAO::findAll() -> exists(h | h.getId() = hiveId)
Post-condizione	/

Nome Metodo	**+generateReport (int hiveId, HttpServletResponse response): void**
Descrizione	Permette di generare un report sullo stato di salute dell'arnia.
Pre-condizione	**Context:** StatusService::checkAnomalies(Measurement measurement): void **Pre:** hiveId <> null and HiveDAO::findAll() -> exists(h | h.getId() = hiveId)
Post-condizione	/

Nome Classe	**SimulatedSensorService**
Descrizione	Questa classe permette di interfacciarsi con il sottosistema per la gestione dei sensori.
Metodi	+simulateMeasurements(): void
Invarianti di classe	N/A

Nome Metodo	**+simulateMeasurements(): void**
Descrizione	Simula le misurazioni effettuate dai sensori.
Pre-condizione	/
Post-condizione	/

3.4 Package Gestione Utente

Nome Classe	ProfileService
Descrizione	Questa classe permette di interfacciarsi con il sottosistema per la gestione dell'utente.
Metodi	+registration(String email, String password, String firstName, String lastName, String companyName, String companyPiva): Beekeeper +changeInfo(String email, String firstName, String lastName, String companyName): Beekeeper +changePassword(String email, String password): void
Invarianti di classe	N/A

Nome Metodo	+registration(String email, String password, String firstName, String lastName, String companyName, String companyPiva): Beekeeper
Descrizione	Permette la registrazione di un utente sulla piattaforma.
Pre-condizione	**Context:** ProfileService::registration(String email, String password, String firstName, String lastName, String companyName, String companyPiva): Beekeeper **Pre**: email <> null and password <> null and firstName <> null and lastName <> null and companyName <> null and companyPiva <> null and not ProfileService::emailExists(email) = true and not ProfileService::pivaExists(companyPiva) = true
Post-condizione	/

Nome Metodo	+changeInfo(String email, String firstName, String lastName, String companyName): Beekeeper
Descrizione	Permette la modifica delle informazioni personali dell'utente.
Pre-condizione	**Context:** ProfileService::changeInfo(String email, String firstName, String lastName, String companyName): Beekeeper **Pre**: email <> null and firstName <> null and lastName <> null and companyName <> null and emailExists(email) = true
Post-condizione	/

Nome Metodo	**+changePassword(String email, String password): void**
Descrizione	Permette la modifica della password dell'utente.
Pre-condizione	**Context:** ProfileService::changePassword(String email, String password): void **Pre**: email <> null and emailExists(email) = true and password <> null
Post-condizione	/

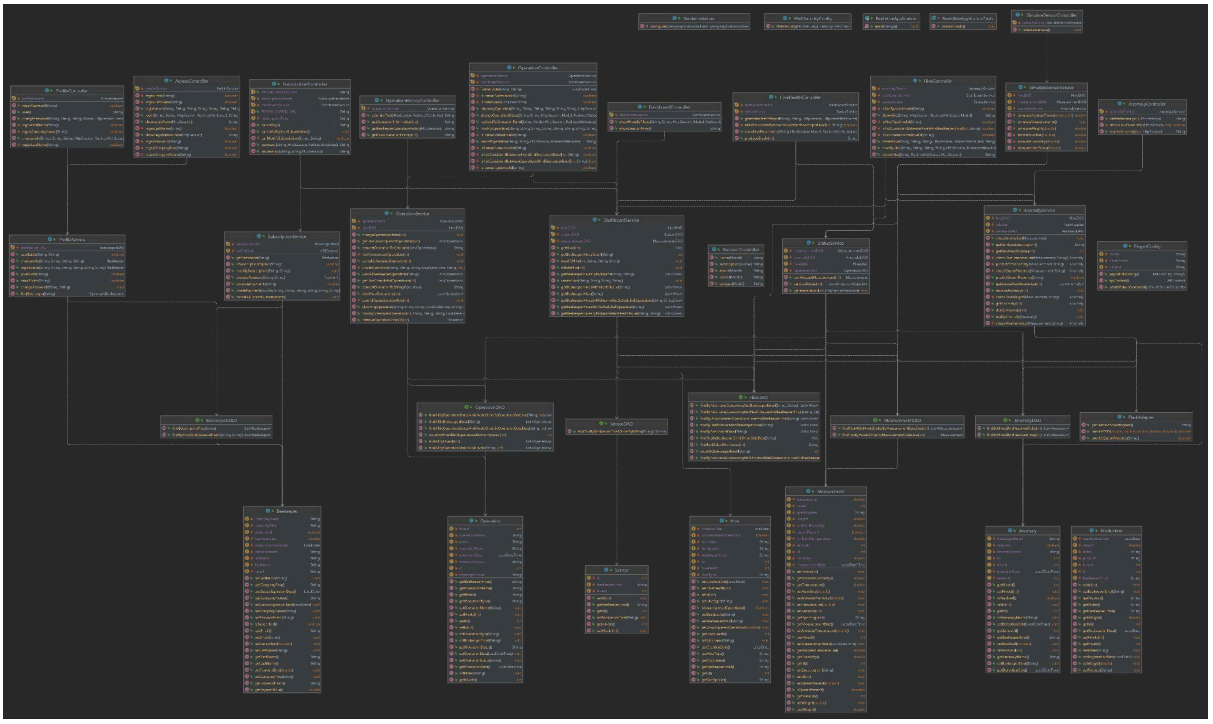
Nome Classe	**SubscriptionService**
Descrizione	Questa classe permette di interfacciarsi con il sottosistema per la gestione dell'utente, in particolare riguardo all'abbonamento.
Metodi	+modifySubscription(String beekeeperEmail, String subscriptionType): void +cancelSubscription(String beekeeperEmail): void +calculatePayment(String subscriptionType): double
Invarianti di classe	N/A

Nome Metodo	**+modifySubscription(String beekeeperEmail, String subscriptionType): void**
Descrizione	Permette la modifica dell'abbonamento.
Pre-condizione	**Context:** SubscriptionService::modifySubscription(String beekeeperEmail, String subscriptionType): void **Pre:** subscriptionType <> null and beekeeperEmail <> null and emailExists(beekeeperEmail) = true
Post-condizione	/

Nome Metodo	**+calculatePayment(String subscriptionType): double**
Descrizione	Calcola la quota da pagare per acquistare l'abbonamento scelto.
Pre-condizione	**Context:** SubscriptionService::calculatePayment(String subscriptionType): double **Pre**: subscriptionType <> null
Post-condizione	/

4. Class Diagram

Di seguito è riportato il class diagram che rispecchia il sistema. Per visualizzare meglio l'immagine del diagramma, consultare questo link.



5. Glossario

Termine	Descrizione
Componenti Off-the-Shelf	Componenti hardware e software attualmente reperibili in commercio, destinati all'acquisto da parte di aziende di sviluppo che desiderano integrare nei propri progetti.
Back-end	Il concetto di back-end nel campo del web publishing fa riferimento all'interfaccia utilizzata dal gestore di un sito web dinamico per amministrare i suoi contenuti e le sue funzionalità.
Front-end	In un servizio pubblico erogato attraverso una rete telematica o telefonica, la totalità delle applicazioni e dei programmi informatici con i quali l'utente interagisce direttamente è denominata frontend, in contrapposizione al backend.
Spring	Un framework open-source per lo sviluppo di applicazioni Java. Offre una vasta gamma di funzionalità, tra cui la gestione del ciclo di vita degli oggetti, l'iniezione di dipendenze, l'accesso ai dati e la sicurezza. Semplifica lo sviluppo di applicazioni Java fornendo un approccio modulare e favorisce la creazione di codice pulito e manutenibile.
Thymeleaf	Un motore di template per Java utilizzato principalmente nello sviluppo web, integrato con i framework Java come Spring. Consente di incorporare dinamicamente contenuti dinamici nelle pagine HTML.
JavaScript	JavaScript è un linguaggio di programmazione orientato agli oggetti e agli eventi, comunemente utilizzato nella programmazione Web lato client.

JDBC	Acronimo di Java Database Connectivity, è un connettore per database che facilita l'accesso e la gestione della persistenza dei dati nelle basi di dati da qualsiasi programma scritto con il linguaggio di programmazione Java. Questa facilità di accesso è indipendente dal tipo di DBMS utilizzato.
DBMS	Acronimo di Database Management System, è un sistema software progettato per agevolare la creazione, manipolazione e interrogazione efficiente di database, può essere ospitato su un'architettura hardware dedicata o su un comune computer.
CRUD	Acronimo di Create, Read, Update, Delete. Si riferisce alle quattro operazioni di base eseguite sui dati contenuti all'interno di un database.
Apicoltore	Utente che usufruisce del servizio.