Laurea Triennale in informatica - Università di Salerno
Corso di Ingegneria del Software - Prof.ssa F.Ferrucci, Prof F.Palomba

# ODD

# Object Design Document

# Environmental Intelligence For Agricolture

| | |
|---|---|
| Riferimento | C04_ODD_v1.0 |
| Versione | 1.0 |
| Data | 28/12/2021 |
| Destinatario | Prof.ssa F. Ferrucci, Prof. F. Palomba |
| Presentato da | C04 IS_Panda |

| | |
|---|---|
| Approvato da | Carmine Laudato, Pierluigi Lambiase |

## Revision History

| Data | Versione | Descrizione | Autori |
|---|---|---|---|
| 23/12/2022 | 0.1 | Introduzione ODD | Maria Lombardi<br>Gerardo Frino |
| 23/12/2022 | 0.2 | Stesura Package | Francesco Maria Puca<br>Francesco Fattorusso<br>Maria Lombardi<br>Gerardo Frino<br>Benedetto Scala |
| 26/12/2022 | 0.3 | Stesura Class Interface | Benedetto Scala |
| 27/12/2022 | 0.4 | Stesura Class Interface | Maria Lombardi<br>Francesco Maria Puca<br>Gerardo Frino |
| 28/12/2022 | 0.5 | Stesura Design Pattern | Benedetto Scala<br>Francesco Maria Puca |
| 28/12/2022 | 0.6 | Stesura Glossario | Maria Lombardi |
| 07/02/2022 | 1.0 | Completamento documento | Francesco Fattorusso |

Sommario

# 1. Introduzione

EnIA nasce con il principale scopo di offrire e semplificare l'amministrazione delle colture attraverso la gestione dei terreni, al tracciamento del livello di inquinamento, all' esposizione ambientale e gli agenti atmosferici.

In questa sezione del presente documento verrà illustrato l'Object Design, verranno descritti i design goal, le linee guida per la fase di implementazione, nonché una vista dei package e classi del sistema.

## 1.1 Object design goals

### Grafica responsive

Il sistema EnIa avrà responsive in grado di adattarsi a sistema in uso, in tale modo non ci sono vincoli su dove usare e quando usare la piattaforma

### Facilita d'uso

Il sistema deve risultare "facile" ed intuitivo, poiché tale piattaforma si rivolge ad una platea di utenti che va dalla persona con abilita informante fino a persone con scarse abilita informatiche

### Gestione permessi

Il sistema permette di separare gli utenti in diversi ruoli, al fine di consentire ad i singoli utenti di visionare solo le funzionalità concesse per quello specifico utente

## 1.2 Linee guida per la documentazione dell'interfaccia

Le linee guida, di seguito riportate, includono un insieme di regole e convenzioni alle quali gli sviluppatori devono attenersi in fase di implementazione. Laddove ciò non sia possibile, spetta al programmatore riportare e motivare le nuove scelte.

### 1.2.1. Python

La qualità del codice farà riferimento alle linee guida ufficiali presenti al seguente link: https://peps.python.org/pep-0008/ .

### 1.2.2. HTML

Le varie pagine HTML seguiranno gli standard specificati al seguente link: https://www.w3.org/TR/2011/WD-html5-20110405/syntax.html .

#### 1.2.3. JavaScript

La parti di codice JavaScript dovranno seguire le seguenti linee guida:

https://www.w3.org/wiki/JavaScript_best_practices .

#### 1.2.4. CSS

Per la modellazione CSS verranno seguiti i seguenti standard: https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/Organizing .

Per maggiori informazioni sugli standard implementativi consultare il documento di Object Design.

### 1.3 Riferimenti

Di seguito una lista di riferimenti ad altri documenti utili durante la lettura:

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

**Commentato [BS1]:** Aggiungere Link

## 2. Packages

In questa sezione viene mostrata la suddivisione del sistema in package, secondo quanto specificato nel documento di System Design.
È stata effettuata tenendo conto delle scelte architetturali prese e si basa sulla struttura di directory standard per un progetto Flask.

- **src –** Contiene tutti i file sorgente del progetto
  - **logic –** Contiene file dei layer Application e Storage
  - **static –** Contiene gli script di inserimento ed estrazione dati per DB e Decision Intelligence da varie fonti, tra cui le API
  - **templates –** Contiene i file che compongono il layer Presentation
    - **fonts**
    - **images**
    - **js**
    - **css**
    - **pages**
    - **partials**
    - **scss**
    - **vendors**

**Commentato [BS2]:** qui si mette la parte di estrazione dei dati da diverse sorgenti

- **unittest –** Contiene tutti i file relativi alla fase di testing
- **target –** Contiene i report di build
- **documentation –** contiene la documentazione generata automaticamente da Sphinx

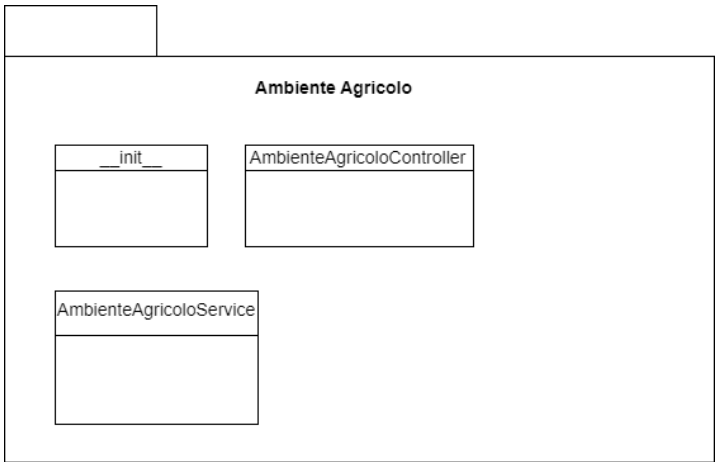
Sono state prese le seguenti scelte di design per la strutturazione dei package del sistema:

- Creazione di un package per ogni sottosistema, composto dai file che ne contengono la logica per il layer Application e Presentation.
- Creazione di un package Storage, contenente i file del layer Storage, in modo da separarlo logicamente e poterlo collegare agli altri package che fanno uso delle sue funzioni.
- Creazione di un package separato contenente tutte le classi usate per il Testing, per tutti i tool usati, i relativi oracoli ed altri file necessari, quali driver e stub.

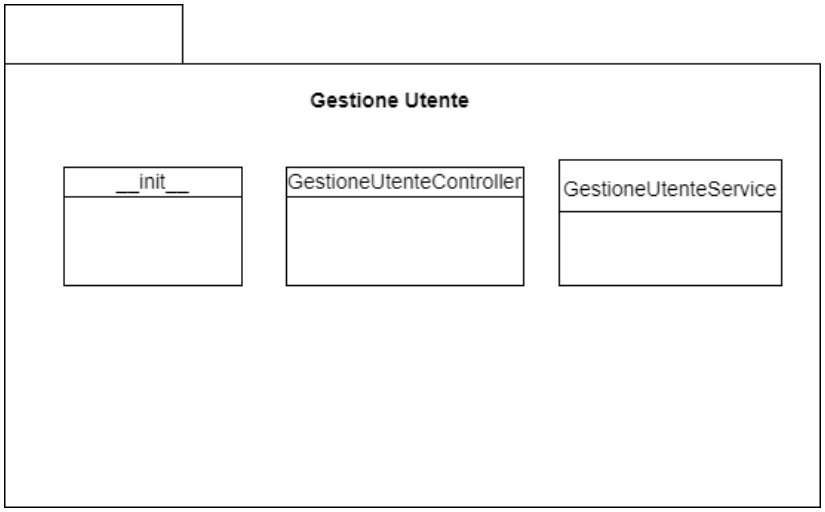
Riguardo le dipendenze tra i package, la struttura così descritta presenta un'unica dipendenza, quella tra il package Storage e tutti gli altri package.

### Package Registrazione

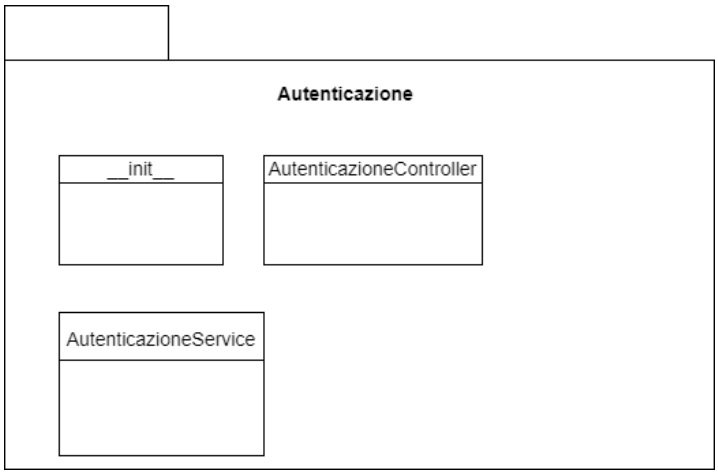
## Package Ambiente Agricolo



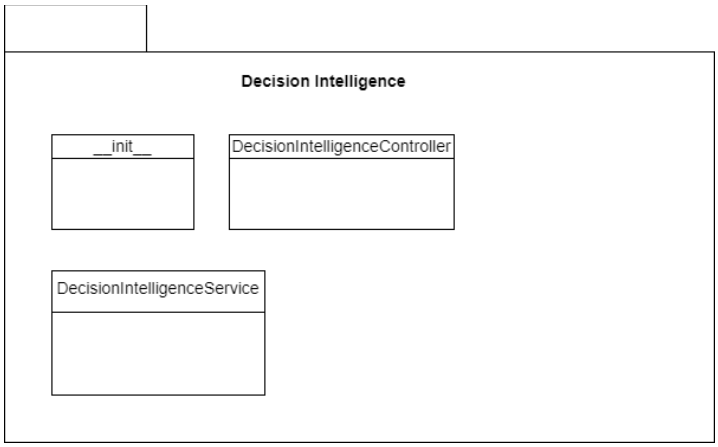

## Package Gestione Utente



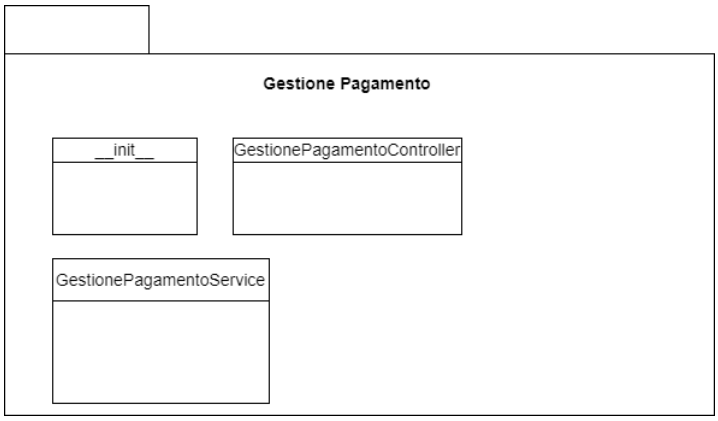
## Package Autenticazione



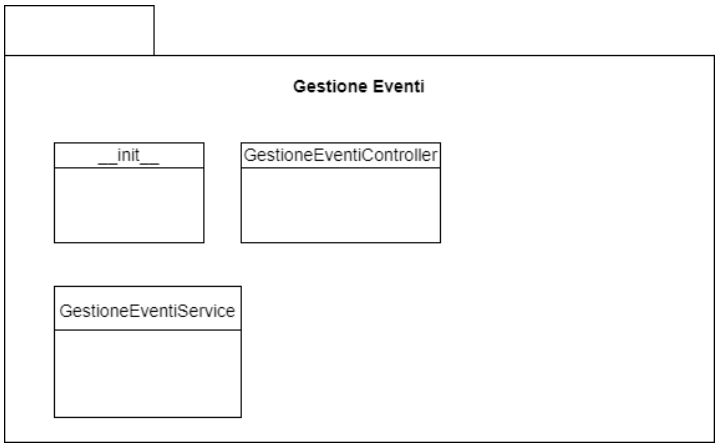

## Package Decision Intelligence



### Package Gestione Pagamento




### Package Gestione Eventi




## 2. Class Interfaces

Di seguito saranno presentate le interfacce di ciascun package.

**Documentazione Sphinx di ENIA**

Per motivi di leggibilità si è scelto di creare un sito, hostato tramite GitHub pages, contenente la documentazione di ENIA generata con Sphinx. In tale maniera, chiunque può consultare la documentazione aggiornata dell'intero sistema.
Di seguito, il link al sito in questione: [DA CREARE LINK]

## Package Registrazione

| **Nome classe** | **RegistrazioneService** |
|---|---|
| **Descrizione** | Questa classe si occupa della registrazione di tutti gli attori sul sistema, quindi farmer, Air pollution analyst e irrigation manager |
| **Metodi** | + trovaUtenteByCodiceDiAccesso(codice:str) : Utente<br>+ trovaUtenteByEmail(email:str) : Utente<br>+ modificaUtente(nome:str, cognome:str, email:str, password:str, dataDiNascita:str, codice:str, indirizzo:str) : Dict<br>+ creaFarmer(nome:str, cognome:str, email:str, password:str, dataDiNascita:str, partitaiva:str, indirizzo:str): String<br>+ creaLicenza(id:str, tipo:str): Licenza<br>+ creaMetodoDiPagamento(numerocarta, titolare, scadenza, cvv, id) : MetodoDiPagamento |
| **Invariante di classe** | / |

| **Nome metodo** | + trovaUtenteByCodiceDiAccesso(codice:str) : Utente |
|---|---|
| **Descrizione** | Cerca un utente attraverso il suo codice di accesso |
| **Pre-condizione** | Context:RegistrazioneService::trovaUtenteByCodiceDiAccesso(codice:str)<br>Pre: |
| **Post-condizione** | Context:RegistrazioneService::trovaUtenteByCodiceDiAccesso(codice:str)<br>Post: AutenticazioneDAO.trovaUtenteByCodiceDiAccesso(codice) |
| **Nome metodo** | + trovaUtenteByEmail(email:str) : Utente |
| **Descrizione** | Questo metodo trova un utente nel database, utilizzando la sua email |
| **Pre-condizione** | Context: RegistrazioneService::trovaUtenteByEmail(email:str)<br><br>Pre: |
| **Post-condizione** | Context: RegistrazioneService::trovaUtenteByEmail(email:str)<br>(Utente utente)<br>Post:<br>AutenticazioneDAO.trovaUtenteByEmail(email) |
| **Nome metodo** | + modificaUtente(nome:str, cognome:str, email:str, password:str, dataDiNascita:str, codice:str, indirizzo:str) : Dict |
| **Descrizione** | Modifica i dati relativi ad un utente |

| | |
|---|---|
| **Pre-condizione** | Context:RegistrazioneService::modificaUtente(nome:str,cognome:str, email:str, password:str, dataDiNascita:str, codice:str, indirizzo:str)<br><br>Pre:<br>AutenticazioneDAO.trovaUtenteByCodiceDiAccesso(codice) != null |
| **Post-condizione** | Context:RegistrazioneService::modificaUtente(nome:str,cognome:str,email:str, password:str,dataDiNascita:str,codice:str,indirizzo:str)<br>Post:<br>AutenticazioneDAO.modificaUtente(utentetrovato)<br>Risposta[“utenteregistrato”] = true |
| **Nome metodo** | + creaLicenza(id:str, tipo:str): Licenza |
| **Descrizione** | Crea una nuova licenza |
| **Pre-condizione** | Context:RegistrazioneService::creaLicenza(id:str, tipo:str)<br>Pre:<br><br>RegistrazioneService.trovaUtenteByEmail(email) is None |
| **Post-condizione** | Context:RegistrazioneService::creaLicenza(id:str, tipo:str)<br><br>Post:<br>LicenzaDAO.creaLicenza(licenza) |
| **Nome metodo** | + creaFarmer(nome:str, cognome:str, email:str, password:str, dataDiNascita:str, partitaiva:str, indirizzo:str): String |
| **Descrizione** | Crea un nuovo utente farmer |
| **Pre-condizione** | Context:RegistrazioneService::creaFarmer(nome:str,cognome:str,email:str, password:str, dataDiNascita:str, partitaiva:str, indirizzo:str)<br><br>Pre:<br>RegistrazioneService.trovaUtenteByEmail(email) is None |
| **Post-condizione** | Context:RegistrazioneService::creaFarmer(nome:str, cognome:str, email:str, password:str, dataDiNascita:str, partitaiva:str, indirizzo:str)<br><br>Post:<br>AutenticazioneDAO.creaUtente(utente) |
| **Nome metodo** | + creaMetodoDiPagamento(numerocarta, titolare, scadenza, cvv, id) : MetodoDiPagamento |
| **Descrizione** | Crea un nuovo metodo di pagamento |
| **Pre-condizione** | Context:RegistrazioneService::creaMetodoDiPagamento(numerocarta,titolare, scadenza, cvv, id)<br>Pre:<br>RegistrazioneService.trovaUtenteByEmail(email) is None |

| | |
|---|---|
| **Post-condizione** | Context:RegistrazioneService::creaMetodoDiPagamento(numerocarta,titolare, scadenza, cvv, id)<br>Post:<br>MetodoDiPagamentoDAO.creaMetodo(metodo) |

## Package Ambiente Agricolo

| **Nome classe** | **AmbienteAgricoloService** |
|---|---|
| **Descrizione** | Questa classe si occupa della gestione e della visualizzazione di tutti i dettagli inerenti ad uno o più ambienti agricoli, tra cui anche la visualizzazione meteo, agenti inquinanti, e infine si occupa anche della gestione degli irrigatori. |
| **Metodi** | + isValidTerreno(terreno: Terreno): bool<br>+ validateTerreno(terreno: Terreno): dict<br>+ visualizzaTerreni(farmer:str): List<Terreno><br>+aggiungiTerreno(nome: str, coltura:str, stadio_crescita: str, posizione, preferito:bool, priorita:int, proprietario: str): bool<br>+ trovaTerreno(id: str): Terreno<br>+ modificaTerreno(id:str, nome: str, coltura:str, stadio_crescita: str, posizione, preferito:bool, priorita:int, proprietario: str): bool<br>+ eliminaTerreno(id:str): bool<br>+ cercaPosizione(id:str): dict<br>+ cercaInquinamento(provincia:str, regione:str, nazione:str, comune:str): str<br>+ cercaStoricoInquinamento(dataInizio:str, dataFine:str, comune:str, regione:str, nazione:str, provincia:str, formato:str): str<br>+ aggiungiIrrigatore(id_terreno: str, nome_irrigatore: str, posizione_irrigatore: str) : str<br>+ modificaIrrigatore(idIrrigatore: str, nomeIrrigatore: str, posizioneIrrigatore: str): bool<br>+ getIrrigatore(idIrrigatore: str): Irrigatore<br>+ visualizzaListaIrrigatori(idTerreno: str): List<Irrigatore><br>+ eliminaIrrigatore(idIrrigatore: str): bool<br>+ attivaDisattivaIrrigatore(idIrrigatore:str): bool<br>+ visualizzaListaEventi(idTerreno: str): List<Evento><br>+ cercalat(id:str): float<br>+ cercalon(id:str): float<br>+ restituisciPredizioneLivelliIrrigazione(lon:float, lat:float, crop:str, stage:str): void |

| Invariante di classe | / |
|---|---|

| Nome metodo | + isValidTerreno(terreno: Terreno): bool |
|---|---|
| Descrizione | Controlla se un terreno è valido |
| Pre-condizione | Context: AmbienteAgricoloService::isValidTerreno(terreno: Terreno)<br>Pre: |
| Post-condizione | Context: AmbienteAgricoloService::isValidTerreno(terreno: Terreno)<br>Post:<br>AmbienteAgricoloService.validateTerreno(terreno) |
| Nome metodo | +validateTerreno(terreno: Terreno): dict |
| Descrizione | Controlla se tutti gli attributi di terreno sono validi |
| Pre-condizione | Context: AmbienteAgricoloService::validateTerreno(terreno: Terreno)<br>Pre:<br>AmbienteAgricoloService.isValidTerreno(terreno) |
| Post-condizione | Context: AmbienteAgricoloService::validateTerreno(terreno: Terreno)<br>Post:<br>Risultato[“esitoControllo”] = bool |
| Nome metodo | + visualizzaTerreni(farmer:str): List<Terreno> |
| Descrizione | Restituisce una lista dei terreni di cui un utente farmer è proprietario |
| Pre-condizione | Context: AmbienteAgricoloService::visualizzaTerreni(farmer:str)<br>Pre:<br>LoginManager.login_required && current_user.ruolo == “farmer” |
| Post-condizione | Context: AmbienteAgricoloService::visualizzaTerreni(farmer:str)<br>Post:<br>List<Terreno> terreni = TerrenoDAO.restituisciTerreniByFarmer(farmer) |
| Nome metodo | +aggiungiTerreno(nome: str, coltura:str, stadio_crescita: str, posizione, preferito:bool, priorita:int, proprietario: str): bool |
| Descrizione | Questo metodo aggiunge un terreno al database |
| Pre-condizione | Context:<br>AmbienteAgricoloService::aggiungiTerreno(nome: str, coltura:str, stadio_crescita: str, posizione, preferito:bool, priorita:int, proprietario: str)<br>Pre:<br>LoginManager.login_required && current_user.ruolo == “farmer” |
| Post-condizione | Context:<br>AmbienteAgricoloService::aggiungiTerreno(nome: str, coltura:str, stadio_crescita: str, posizione, preferito:bool, priorita:int, proprietario: str)<br>Post:<br>risultato[“esitoOperazione”] = bool<br>risultato[“restituito”] = Terreno \| \| None |
| Nome metodo | + trovaTerreno(id: str): Terreno |
| Descrizione | Questo metodo restituisce un terreno in base all'id |
| Pre-condizione | Context: AmbienteAgricoloService::trovaTerreno(id: str) |

| | |
|---|---|
| | Pre:<br>LoginManager.login_required |
| **Post-condizione** | Context: AmbienteAgricoloService::trovaTerreno(id: str)<br>Post:<br>terreno = TerrenoDAO.trovaTerreno(id) |
| **Nome metodo** | +modificaTerreno(id:str, nome: str, coltura:str, stadio_crescita: str, posizione, preferito:bool, priorita:int, proprietario: str): bool |
| **Descrizione** | Questo metodo modifica i dati di un terreno |
| **Pre-condizione** | Context: AmbienteAgricoloService::modificaTerreno(id:str, nome: str, coltura:str, stadio_crescita: str, posizione, preferito:bool, priorita:int, proprietario: str)<br>Pre:<br><br>LoginManager.login_required && current_user.ruolo == “farmer” |
| **Post-condizione** | // |
| **Nome metodo** | +eliminaTerreno(id:str): bool |
| **Descrizione** | Questo metodo permette di eliminare un terreno |
| **Pre-condizione** | Context: AmbienteAgricoloService::eliminaTerreno(id:str)<br>Pre:<br>LoginManager.login_required && current_user.ruolo == “farmer” && AmbienteAgricoloService.trovaTerreno(id) != None |
| **Post-condizione** | Context: AmbienteAgricoloService:: eliminaTerreno(id:str)<br>Post:<br>AmbienteAgricoloService.trovaTerreno(id) == None |
| **Nome metodo** | +cercaPosizione(id:str): dict |
| **Descrizione** | Questo metodo restituisce la posizione di un terreno come oggetto GeoJSON. |
| **Pre-condizione** | Context: AmbienteAgricoloService::cercaPosizione(id:str):<br>Pre:<br>LoginManager.login_required && AmbienteAgricoloService.trovaTerreno(id) != None |
| **Post-condizione** | // |
| **Nome metodo** | +cercaInquinamento(provincia:str, regione:str, nazione:str, comune:str): str |
| **Descrizione** | Questo metodo cerca i dati relativi all'inquinamento di una specifica zona |
| **Pre-condizione** | // |
| **Post-condizione** | Context: cercaInquinamento(provincia:str, regione:str, nazione:str, comune:str)<br>Post: |

| | |
|---|---|
| | Datiapi = SenseSquareAdapter(nazione, regione, provincia, comune).get_data_for_today() |
| **Nome metodo** | +cercaStoricoInquinamento(dataInizio:str, dataFine:str, comune:str, regione:str, nazione:str, provincia:str, formato:str): str |
| **Descrizione** | Questo metodo cerca i dati relativi all'inquinamento di una specifica zona in uno specifico range temporale |
| **Pre-condizione** | // |
| **Post-condizione** | Context: cercaInquinamento(provincia:str, regione:str, nazione:str, comune:str)<br>Post:<br><br>Datiapi = SenseSquareAdapter(nazione, regione, provincia, comune, start_date=dataInizio, end_date=dataFine, formato=formato) ).get_data_time_interval() |
| **Nome metodo** | +aggiungiIrrigatore(id_terreno: str, nome_irrigatore: str, posizione_irrigatore: str) : str |
| **Descrizione** | Questo metodo permette di associare un irrigatore ad un ambiente agricolo |
| **Pre-condizione** | Context:aggiungiIrrigatore(id_terreno: str, nome_irrigatore: str, posizione_irrigatore: str)<br>Pre:<br><br>LoginManager.login_required |
| **Post-condizione** | Context:aggiungiIrrigatore(id_terreno: str, nome_irrigatore: str, posizione_irrigatore: str)<br>Post:<br>Id = ImpiantoDiIrrigazioneDAO.creaImpianto(impianto, id_terreno)<br>Evento = GestioneEventiService.creaEvento(evento) |
| **Nome metodo** | +modificaIrrigatore(idIrrigatore: str, nomeIrrigatore: str, posizioneIrrigatore: str): bool |
| **Descrizione** | Questo metodo permette di modificare un irrigatore |
| **Pre-condizione** | Context:<br>AmbienteAgricoloService::modificaIrrigatore(idIrrigatore: str, nomeIrrigatore: str, posizioneIrrigatore: str)<br>Pre:<br><br>AmbienteAgricoloService.getIrrigatore(idIrrigatore != null) &&<br>LoginManager.login_required |
| **Post-condizione** | // |

| | |
|---|---|
| **Nome metodo** | +getIrrigatore(idIrrigatore: str): Irrigatore |
| **Descrizione** | Questo metodo restituisce un irrigatore in base all'id |
| **Pre-condizione** | Context:<br>AmbienteAgricoloService::getIrrigatore(idIrrigatore: str)<br>Pre:<br><br>LoginManager.login_required |
| **Post-condizione** | Context:<br>AmbienteAgricoloService::getIrrigatore(idIrrigatore: str)<br>Post:<br><br>Impianto = ImpiantoDiIrrigazioneDAO.findImpiantoById(idIrrigatore) |
| **Nome metodo** | +visualizzaListaIrrigatori(idTerreno: str): List<Irrigatore> |
| **Descrizione** | Questo metodo restituisce una lista di tutti gli impianti di irrigazione collegati ad un ambiente agricolo |
| **Pre-condizione** | Context:<br>AmbienteAgricoloService::visualizzaListaIrrigatori(idTerreno: str)<br>Pre:<br><br>LoginManager.login_required |
| **Post-condizione** | Context:<br>AmbienteAgricoloService::visualizzaListaIrrigatori(idTerreno: str)<br>Post:<br><br>List<Terreno> lista =<br>ImpiantoDiIrrigazioneDAO.findImpiantoById(idIrrigatore) |
| **Nome Metodo** | +eliminaIrrigatore(idIrrigatore: str): bool |
| **Descrizione** | Questo metodo permette di eliminare un irrigatore dal sistema |
| **Pre-condizione** | Context:<br>AmbienteAgricoloService::eliminaIrrigatore(idIrrigatore: str)<br>Pre:<br><br>LoginManager.login_required |
| **Post-condizione** | // |
| **Nome Metodo** | + attivaDisattivaIrrigatore(idIrrigatore:str): bool |
| **Descrizione** | Attiva o disattiva un irrigatore |
| **Pre-condizione** | Context: AmbienteAgricoloService::attivaDisattivaIrrigatore(idIrrigatore:str)<br>Pre: |

| | |
|---|---|
| | LoginManager.login_required &&<br>ImpiantoDiIrrigazioneDAO.findImpiantoById(idIrrigatore) != None |
| **Post-condizione** | // |
| **Nome metodo** | + visualizzaListaEventi(idTerreno: str): List<Evento> |
| **Descrizione** | Restituisce la lista degli eventi su un terreno |
| **Pre-condizione** | Context: AmbienteAgricoloService::visualizzaListaEventi(idTerreno: str)<br>Pre:<br>LoginManager.login_required &&<br>TerrenoDAO.TrovaTerreno(idTerreno) != None |
| **Post-condizione** | // |
| **Nome metodo** | + cercalat(id:str): float |
| **Descrizione** | Recupera la latitudine di un terreno |
| **Pre-condizione** | Context: AmbienteAgricoloService::cercalat(id:str)<br>Pre:<br>TerrenoDAO.TrovaTerreno(idTerreno) != None &&<br>isValidTerreno(terreno) |
| **Post-condizione** | // |
| **Nome metodo** | + cercalon(id:str): float |
| **Descrizione** | Recupera la longitudine di un terreno |
| **Pre-condizione** | Context: AmbienteAgricoloService::cercalon(id:str)<br>Pre:<br>TerrenoDAO.TrovaTerreno(idTerreno) != None &&<br>isValidTerreno(terreno) |
| **Post-condizione** | // |
| **Nome metodo** | + restituisciPredizioneLivelliIrrigazione(lon:float, lat:float, crop:str, stage:str): void |
| **Descrizione** | Recupera la predizione dei livelli di irrigazione forniti dal modulo AI |
| **Pre-condizione** | // |
| **Post-condizione** | // |

### Package Gestione Pagamento

| **Nome classe** | **GestionePagamentoService** |
|---|---|
| **Descrizione** | Si occupa delle funzionalità di inserimento, modifica e rimozione di un metodo di pagamento di un farmer |
| **Metodi** | +creaMetodoDiPagamento(numerocarta, titolare, scadenza, cvv, id): MetodoDiPagamento |

| | |
|---|---|
| | +modificaMetodo(mp: MetodoDiPagamento, num_carta:str, titolare:str, scadenza:str, cvv:str): void<br>+findMetodoByProprietario(id:str): MetodoDiPagamento |
| **Invariante di classe** | / |

| | |
|---|---|
| **Nome metodo** | +creaMetodoDiPagamento(numerocarta, titolare, scadenza, cvv, id): MetodoDiPagamento: |
| **Descrizione** | Questa funzionalità permette la creazione di un metodo di pagamento |
| **Pre-condizione** | // |
| **Post-condizione** | // |
| **Nome metodo** | +modificaMetodo(mp: MetodoDiPagamento, num_carta:str, titolare:str, scadenza:str, cvv:str): void |
| **Descrizione** | Questa funzionalità permette di modificare un metodo di pagamento |
| **Pre-condizione** | // |
| **Post-condizione** | Context: GestionePagamentoService::modificaMetodo(mp: MetodoDiPagamento, num_carta:str, titolare:str, scadenza:str, cvv:str)<br>Post:<br><br>Mp == creaMetodoDiPagamento(num_carta, titolare, scadenza, cvv, mp.id) |
| **Nome metodo** | +findMetodoByProprietario(id:str): MetodoDiPagamento |
| **Descrizione** | Questa funzionalità permette di recuperare il metodo  di un utenete |
| **Pre-condizione** | Context: GestionePagamentoService::findMetodoByProprietario(id:str)<br>Pre:<br><br>AutenticazioneDAO.trovaUtente(id) != None |
| **Post-condizione** | // |

### Package Gestione Eventi

| | |
|---|---|
| **Nome classe** | **GestioneEventiService** |
| **Descrizione** | Si occupa della gestione degli alert e delle notifiche da inviare agli utenti, oltre che alla visualizzazione dello storico degli eventi estremi o non passati. |
| **Metodi** | +creaEvento(evento : Evento): void |

| | |
|---|---|
| | +visualizzaEventiByTerreno(idTerreno:str): List<Evento><br>+cancellaTuttiEventiByTerreno(idTerreno:str): void<br>+cancellaEvento(idEvento:str): void |
| **Invariante di classe** | / |

| **Nome metodo** | **+ creaEvento(evento : Evento): void** |
|---|---|
| **Descrizione** | Permette la creazione di un evento. |
| **Pre-condizione** | // |
| **Post-condizione** | // |
| **Nome metodo** | + visualizzaEventiByTerreno(idTerreno:str): List<Evento> |
| **Descrizione** | Questa funzionalità permette di visualizzare tutti gli eventi di un ambiente agricolo |
| **Pre-condizione** | Context:GestioneEventiService::visualizzaEventiByTerreno(idTerreno:str)Pre:<br>TerrenoDAO.trovaTerreno(idTerreno) != None |
| **Post-condizione** | // |
| **Nome metodo** | + cancellaTuttiEventiByTerreno(idTerreno:str): void |
| **Descrizione** | Questa funzionalità permette di cancellare tutti gli eventi di un ambiente agricolo |
| **Pre-condizione** | Context:GestioneEventiService::cancellaTuttiEventiByTerreno(idTerreno:str)<br>Pre:<br>TerrenoDAO.trovaTerreno(idTerreno) != None |
| **Post-condizione** | Context:GestioneEventiService::cancellaTuttiEventiByTerreno(idTerreno:str)<br>Post:<br>Len(VisualizzaEventiByTerreno(idTerreno)) == 0 |
| **Nome metodo** | + cancellaEvento(idEvento:str): void |
| **Descrizione** | Questa funzionalità permette di cancellare un evento da un ambiente agricolo |
| **Pre-condizione** | // |
| **Post-condizione** | // |

## Package Gestione Utente

| **Nome classe** | GestioneUtenteService |
|---|---|
| **Descrizione** | Questa classe si occupa di tutte le funzionalità inerenti all'utente, come: visualizzazione, modifica, eliminazione dell'account e assegnazione, modifica e revoca dei ruoli (Air Pollution Analyst e Irrigation Manager) da parte del Farmer. |
| **Metodi** | + modificaUtente(current_user): void<br>+ findLicenzaByProprietario(id: str): Licenza<br>+ getUtenti(id: str): List<Utente><br>+ findMetodoByProprietario(id: str): MetodoDiPagamento |

| | |
|---|---|
| | + removeUtenteFromAzienda(id: str): bool<br>+ GenerateCode(ruolo: str, datore: str): str |
| **Invariante di classe** | // |

| | |
|---|---|
| **Nome metodo** | + modificaUtente(current_user): void |
| **Descrizione** | Permette la modifica dei dati di un utente. |
| **Pre-condizione** | Context: GestioneUtenteService::modificaUtente(current_user)<br>Pre:<br><br>LoginManager.Login_required &&<br>AutenticazioneDAO.trovaUtente(str(current_user.id)) != None |
| **Post-condizione** | // |
| **Nome metodo** | **+ findLicenzaByProprietario(id: str): Licenza** |
| **Descrizione** | Cerca la licenza di un utente farmer |
| **Pre-condizione** | Context: GestioneUtenteService::findLicenzaByProprietario(id: str)<br>Pre:<br><br>AutenticazioneDAO.trovaUtente(id) != None && utente.ruolo == “farmer” |
| **Post-condizione** | // |
| **Nome metodo** | **+ getUtenti(id: str): List<Utente>** |
| **Descrizione** | Permette di recuperare la lista dei dipendenti di un utente farmer |
| **Pre-condizione** | Context: GestioneUtenteService::getUtenti(id: str)<br>Pre:<br><br>AutenticazioneDAO.trovaUtente(id) != None && utente.ruolo == “farmer” |
| **Post-condizione** | |
| **Nome metodo** | **+ findMetodoByProprietario(id: str): MetodoDiPagamento** |
| **Descrizione** | Cerca il metodo di un utente farmer |
| **Pre-condizione** | Context: GestioneUtenteService::findMetodoByProprietario(id: str)<br>Pre:<br><br>AutenticazioneDAO.trovaUtente(id) != None && utente.ruolo == “farmer” |
| **Post-condizione** | // |
| **Nome metodo** | **+ removeUtenteFromAzienda(id: str): bool** |
| **Descrizione** | Rimuove un dipendente |
| **Pre-condizione** | Context: GestioneUtenteService:removeUtenteFromAzienda(id: str)<br>Pre:<br><br>LoginManager.login_required && current_user.ruolo == “farmer” &&<br>getUtenti(current_user.id) > 0 |

| | |
|---|---|
| **Post-condizione** | Context: GestioneUtenteService::removeUtenteFromAzienda(id: str)<br>Post:<br><br>getUtenti(current_user.id) = getUtenti(current_user.id) - 1 |
| **Nome metodo** | **+ GenerateCode(ruolo: str, datore: str): str** |
| **Descrizione** | Genera un codice di accesso per un dipendente |
| **Pre-condizione** | Context: GestioneUtenteService::GenerateCode(ruolo:str, datore:str)<br>Pre:<br><br>LoginManager.login_required && current_user.ruolo == "farmer" |
| **Post-condizione** | Context: GestioneUtenteService::GenerateCode(ruolo:str, datore:str)<br>Post:<br><br>AutenticazioneDAO.trovaUtenteByCodiceDiAccesso(codice) != None |

### Package Autenticazione

| | |
|---|---|
| **Nome classe** | AutenticazioneService |
| **Descrizione** | Questa classe si occupa delle funzionalità di login e logout. |
| **Metodi** | + login(email: str, password: str): bool<br>+ trovaUtenteById(id: str): Utente<br>+ logout(): void<br>+ getDatoreFromDipendente(idDipendente: str): Utente |
| **Invariante di classe** | / |

| | |
|---|---|
| **Nome metodo** | + login(email: str, password: str): bool |
| **Descrizione** | Controlla se le credenziali di accesso sono presenti sul database |
| **Pre-condizione** | Context: AutenticazioneService::login(email: str, password: str)<br>Pre:<br><br>AutenticazioneDAO.trovaUtenteByEmail(email) != None |
| **Post-condizione** | Context: AutenticazioneService::login(email: str, password: str)<br>Post:<br><br>LoginManager.login_user = True |
| **Nome metodo** | **+ trovaUtenteById(id: str): Utente** |
| **Descrizione** | Cerca un utente sul DataBase attraverso il suo id |

| | |
|---|---|
| **Pre-condizione** | // |
| **Post-condizione** | // |
| **Nome metodo** | **+ logout(): void** |
| **Descrizione** | Effettua il logout |
| **Pre-condizione** | Context: AutenticazioneService::logout()<br>Pre:<br><br>LoginManager.login_user = True |
| **Post-condizione** | Context: AutenticazioneService::logout()<br>Pre:<br><br>LoginManager.login_user = False |
| **Nome metodo** | **+ getDatoreFromDipendente(idDipendente: str): Utente** |
| **Descrizione** | Recupera il datore del dipendente |
| **Pre-condizione** | Context:<br>AutenticazioneService::getDatoreFromDipendente(idDipendente: str)<br>Pre:<br><br>AutenticazioneDAO.trovaUtente(idDipendente) != None |
| **Post-condizione** | // |

## Package DecisionIntelligence

| | |
|---|---|
| **Nome classe** | DecisionIntelligenceService |
| **Descrizione** | Questa classe permette l'inserimento, modifica, rimozione e visualizzazione suggerimenti dello scheduling di irrigazione. |
| **Metodi** | +getPredizioneLivelliIrrigazione(lon : float, lat : float, crop : str, stage : str): dict<br>+cercaMeteo(lat:float, lon:float): dict |
| **Invariante di classe** | // |

| | |
|---|---|
| **Nome metodo** | + getPredizioneLivelliIrrigazione(lon : float, lat : float, crop : str, stage : str): dict |

| | |
|---|---|
| **Descrizione** | Questo metodo permette ad un utente autorizzato di ottenere uno schedule per l'irrigazione di un ambiente agricolo, a seguito di un'analisi dei dati meteo rilevati |
| **Pre-condizione** | Context: DecisionIntelligenceService::getPredizioneLivelliIrrigazione(lon: float, lat: float, crop: str, stage: str)<br>Pre:<br><br>Crop in ["Orzo", "Fagiolo", "Cavolo", "Carota", "Cotone","Cetriolo", "Melanzana", "Grano", "Lenticchia", "Lattuga"]<br>stage in ["Iniziale", "Sviluppo", "Metà Stagione", "Fine Stagione"] |
| **Post-condizione** | // |
| **Nome metodo** | **+cercaMeteo(lat:float, lon:float): dict** |
| **Descrizione** | Questo metodo permette ad un utente autorizzato di effettuare un'analisi dei dati meteo relativi ad un ambiente agricolo |
| **Pre-condizione** | Context: DecisionIntelligenceService:cercaMeteo(lat:float, lon:float)<br>Pre:<br><br>LoginManager.login_required |
| **Post-condizione** | // |

# 3. Design Patterns

I design pattern a cui si è fatto affidamento sono i seguenti:

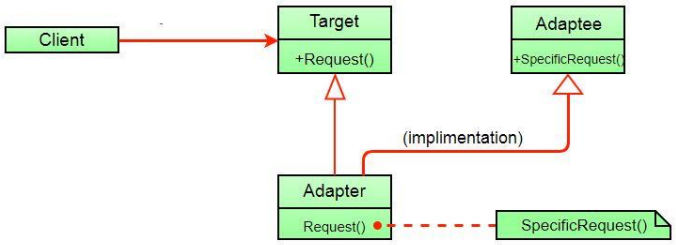
**Adapter**

Il design pattern Adapter è uno dei design pattern di creazione che viene utilizzato per rendere compatibili due interfacce che altrimenti non lo sarebbero. Permette di utilizzare una classe con un'interfaccia specifica in un sistema che richiede un'interfaccia differente.

In altre parole, l'adapter consente di "adattare" una classe esistente a un'interfaccia desiderata, senza modificare la classe esistente. Ciò è utile quando si vuole utilizzare una classe esistente in un sistema che richiede un'interfaccia diversa, o quando si vuole estendere le funzionalità di una classe senza modificare il suo codice sorgente.

Il pattern Adapter viene implementato creando una nuova classe che implementa l'interfaccia desiderata e che contiene una istanza della classe esistente. La nuova classe "adapter" delega a questa istanza le chiamate ai metodi dell'interfaccia, adattando eventualmente i parametri e i valori di ritorno.

**Commentato [BS3]:** Ho aggiunto l'adapter nel caso, non vadano bene gli altri design pattern

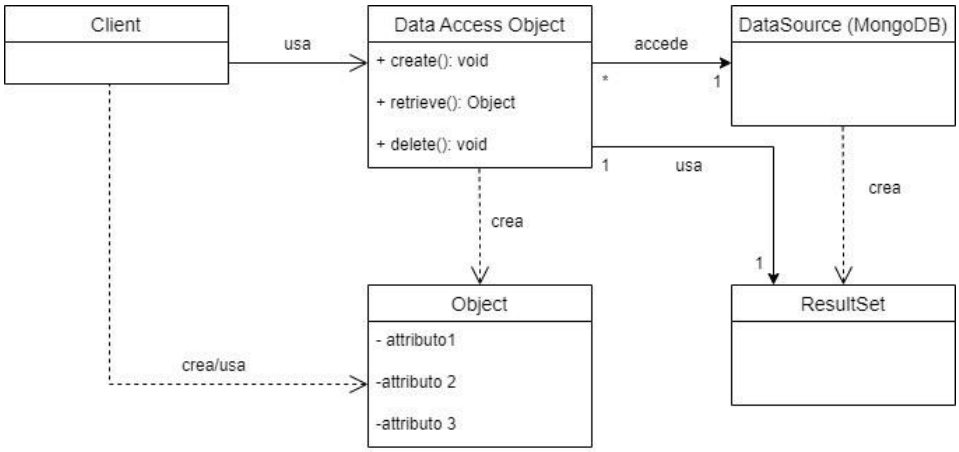



**DAO**

Il design pattern Data Access Object (DAO) è un modello di progettazione utilizzato per isolare l'accesso ai dati da un'applicazione. Si tratta di un livello di astrazione che consente di separare la logica di accesso ai dati dalla logica di business dell'applicazione.

Il pattern DAO prevede la creazione di una classe dedicata all'accesso ai dati, che si occupa di effettuare le operazioni di lettura, scrittura, aggiornamento e cancellazione dei dati. Questa classe, nota come oggetto DAO, esegue le operazioni di accesso ai dati utilizzando una connessione ad un database o a un'altra fonte di dati, nel nostro caso MongoDB.

Il vantaggio di utilizzare il pattern DAO è che permette di modificare le fonti di dati senza dover modificare la logica di business dell'applicazione. Inoltre, il pattern DAO consente di isolare la logica di accesso ai dati, rendendo più facile il test e la manutenzione del codice.




# 4. Glossario

| Sigla/Termine | Definizione |
|---|---|

| | |
|---|---|
| **Package** | Raggruppamento di classi ed interfacce. |
| **DAO** | Data Access Object, implementazione dell'omonimo pattern architetturale che si occupa di fornire un accesso in modo astratto ai dati persistenti. |
| **Design pattern** | Template di soluzioni a problemi ricorrenti impiegati per ottenere riuso e flessibilità; |
| **MongoDB** | Applicativo software progettato per la creazione, la manipolazione e l'interrogazione di un database |
| **Schedule** | Calendario che stabilisce, dato un determinato Ambiente Agricolo, gli orari in cui sarà accesa o meno l'irrigazione dello stesso. Generata automaticamente dal modulo di Decision Intelligence del Sistema |