
# ODD
# Object Design Document
# SmartCargo

| | |
|---|---|
| **Riferimento** | |
| **Versione** | 2.0 |
| **Data** | 12/12/2023 |
| **Destinatario** | Prof.ssa Filomena Ferrucci (FF) |
| **Presentato da** | Mariapia Sorrentino (MS) - 0512113750<br>Roksana Duda (RD) – 0512114326<br>Francesco F. Ambrosio (FA) - 0512114152<br>Paolo Murino (PM) - 0512116057<br>Amedeo Napolitano (AN) - 0512111956<br>Andreea C. C. Oprisescu (AO) - 0512114104 |
| **Approvato da** | Vincenzo Esposito (VE)<br>Nicola Tortora (NT) |

**REVISION HISTORY** ..... **3**

**1. INTRODUZIONE** ..... **4**

1.1. OBJECT DESIGN TRADE OFF ..... 4
1.2. INTERFACE DOCUMENTATION GUIDELINES ..... 4
1.3. DEFINIZIONE, ACRONIMI E ABBREVIAZIONI ..... 5
1.4. RIFERIMENTI ..... 5

**2. DESIGN PATTERNS** ..... **6**

DAO ..... 6
FACADE ..... 6

**3. PACKAGES** ..... **8**

3.1. PACKAGES FRONT-END ..... 8
3.1.1. STRUTTURA FRONTEND ..... 8
PACKAGE AUTENTICAZIONE ..... 10
3.2. PACKAGES BACK-END ..... 21
3.2.1 STRUTTURA BACKEND ..... 23

**4. CLASS INTERFACE** ..... **29**

**5. GLOSSARIO** ..... **37**

# Revision History

| Data | Versione | Descrizione | Autori |
|---|---|---|---|
| 30/10/2023 | 1.0 | Definizione del Template | V.E. |
| 17/12/2023 | 1.1 | Aggiunta Sezione 1.1 Introduzione - Object Design Trade off | M.S |
| 17/12/2023 | 1.2 | Aggiunta Sezione 1.2 Introduzione – Interface documentation guidelines | R.D |
| 17/12/2023 | 1.3 | Aggiunta della sezione 1.3 | A.O |
| 17/12/2023 | 1.4 | Aggiunta della sezione 1.4 | P.M |
| 27/12/2023 | 1.5 | Aggiunta della sezione 2 | M.S, R.D, P.M, A.O, F.A, A.N |
| 27/12/2023 | 1.6 | Aggiunta della sezione 3 | M.S, R.D, P.M, A.O, F.A, A.N |
| 27/12/2023 | 1.7 | Aggiunta della sezione 4 | M.S, R.D, P.M, A.O, F.A, A.N |
| 27/12/2023 | 1.8 | Aggiunta del glossario | M.S |
| 22/01/2024 | 1.9 | Revisione per consegna finale | A.N., F.A. |
| 22/01/2024 | 2.0 | Revisione per consegna finale | P.M, M.S., A.O. |

# 1. Introduzione

SmartCargo mira a ottimizzare e semplificare le operazioni di gestione dei trasporti all'interno del porto. L'obiettivo principale è consentire un controllo efficiente e completo attraverso strumenti di identificazione, monitoraggio, registrazione e gestione delle attività.
Saranno inclusi strumenti di comunicazione per gestire le interazioni tra operatori e autotrasportatori, facilitando la segnalazione e consentendo anche di gestire in modo efficiente le problematiche rilevate.
Uno degli elementi distintivi di SmartCargo è l'impiego di tecnologie avanzate di tracciamento per definire e confrontare costantemente la posizione reale dell'autotrasportatore con un percorso ottimale. In caso di deviazioni significative, il sistema genererà immediatamente un avviso.
Questo approccio non solo garantisce il monitoraggio sicuro del trasporto, ma consente anche interventi rapidi in situazioni critiche.

## 1.1. Object Design Trade off

Sono stati individuati i seguenti trade-off:

- **Massima compatibilità vs nuove funzionalità:** Si privilegerà l'identificazione e l'implementazione solo delle funzionalità essenziali che migliorano l'efficienza operativa, evitando l'introduzione di nuove funzionalità che potrebbero comportare problemi di compatibilità.
- **Memorie efficienti vs costi:** Saranno utilizzate memorie di ultima generazione per garantire prestazioni ottimali, anche a discapito della minimizzazione dei costi. L'obiettivo è massimizzare l'efficienza delle operazioni di gestione dei dati.
- **Tempo di rilascio vs funzionalità:** Nel caso di scadenze stringenti, si potrebbe optare per il rilascio di una versione con meno funzionalità rispetto a quelle richieste, garantendo però il rispetto dei tempi di consegna. L'obiettivo è soddisfare le esigenze temporali senza compromettere la qualità del prodotto.

## 1.2. Interface Documentation GuideLines

Le linee guida includono un insieme di regole che gli sviluppatori dovrebbero seguire durante la progettazione delle interfacce.

Ecco una lista di link alla documentazione ufficiale sulle convenzioni utilizzate per definire le linee guida:

- Python: https://www.python.org/dev/peps/pep-0008/
- Angular Style Guide: https://angular.io/guide/styleguide
- HTML: https://www.w3schools.com/html/html5_syntax.asp

### 1.3. Definizione, Acronimi e Abbreviazioni

• HTML: HyperText Markup Language

• SCSS: Sassy Cascade Style Sheet

• .TS: file con estensione Type Script

• BE: Back-End

• FE: Front-End

• DAO (Data Access Object): È un design pattern utilizzato nella programmazione software per separare la logica di accesso ai dati dal resto della logica di business.

• Package: sono delle strutture organizzative che consentono di aggregare e categorizzare logicamente le componenti del codice, come classi, interfacce o altre risorse correlate.

### 1.4. Riferimenti

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

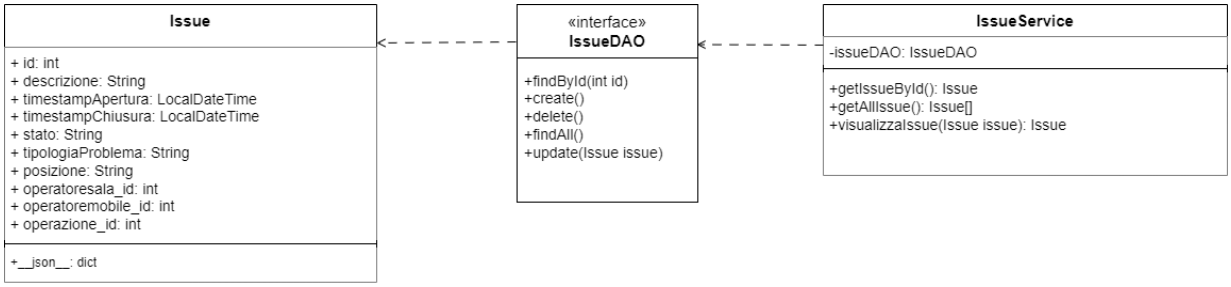
- Statement of Work: C07_SOW_Educational_SmartCargo_V2.0
- Business Case: C07_BC_SmartCargo_V2.0.pdf
- Requirements Analysis Document: C07_RAD_SmartCargo_V2.0
- System Design Document: C07_SDD_SmartCargo_V2.0.pdf
- Object Design Document: C07_ODD_SmartCargo_V1.8
- Test Plan: C07_TP_SmartCargo_V2.0.pdf
- Matrice di tracciabilità: Matrice di tracciabilità
- Manuale Utente: C07_MU_SmartCargo_V2.0
- Manuale Di Installazione: C07_MI_SmartCargo_V2.0

# 2. Design patterns

## DAO

Un DAO (Data Access Object) è un design pattern architetturale che fornisce un'interfaccia unificata per l'accesso ai dati. Questo significa che l'applicazione non deve preoccuparsi di come i dati sono memorizzati o recuperati: il DAO gestisce queste operazioni per conto dell'applicazione. In pratica, il DAO traduce le richieste dell'applicazione in operazioni specifiche sui dati, senza rivelare i dettagli di implementazione del database sottostante. Questo modello è utile in vari linguaggi di programmazione e software che richiedono la persistenza dei dati. Spesso è associato alle applicazioni JavaEE che utilizzano database relazionali, consentendo una chiara separazione tra la logica dell'applicazione e la gestione dei dati.

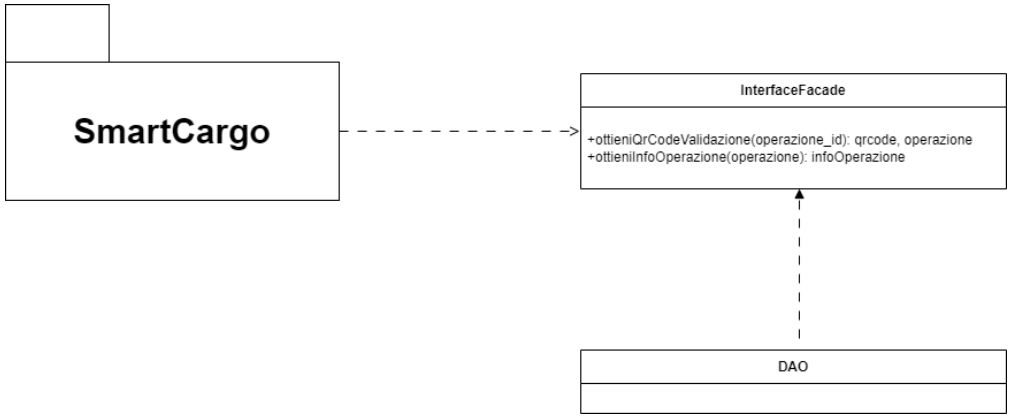
SmartCargo, essendo una web application che si pone di andare a migliorare la sicurezza e l'efficienza delle operazioni che avvengono all'interno del porto, presenta un database molto vasto e quindi necessita di poter interagire con esso in modo rapido e sicuro attraverso numerose query per quella che è la grande mole di dati da gestire.




## Facade

Il Facade è un design pattern strutturale che consente di semplificare l'accesso a sottosistemi più complessi mediante l'implementazione di un'interfaccia unificata. Questo approccio permette di nascondere al sistema la complessità sottostante di librerie, framework o set di classi in uso. Il risultato è un elevato livello di disaccoppiamento, il che rende la piattaforma più manutenibile e facilmente aggiornabile. In pratica, è sufficiente modificare l'implementazione dei metodi nell'interfaccia per apportare cambiamenti, consentendo una gestione più agevole delle modifiche nel sistema senza influenzare il resto dell'applicazione.

In SmartCargo, il design pattern Facade è sfruttato per semplificare l'accesso e l'utilizzo dei models. In particolare, il sistema adotta il Facade per ciascuno dei DAO. Questa implementazione avviene tramite l'uso di un'interfaccia che funge da punto di accesso per accedere ai metodi interni dei DAO.


SmartCargo
InterfaceFacade
+ottieniQrCodeValidazione(operazione_id): qrcode, operazione
+ottieniInfoOperazione(operazione): infoOperazione
DAO

# 3. Packages

## 3.1. Packages Front-end

In questa sezione viene mostrata la suddivisione del sistema in package per quanto riguarda il FrontEnd.

- **FrontEnd**
  - angular
  - .idea
  - .vscode
  - node_modules
  - frontend
    - src
      - app
        - autenticazione
          - login
          - signUp
        - issue
          - gestisciIssue
        - ingresso
          - registrazioneIngresso
        - monitoraggio
          - monitoraggioOperazioniAttive
          - monitoraggioOperazioniCarScar
          - dettaglioOperazioni
          - storicoOperazioni
        - utente
          - EditProfilo
          - homePage
        - admin
          - modificaAccount
          - configuraAccount

### 3.1.1. Struttura Frontend

In questa sezione, viene esposta la struttura del package frontend di SmartCargo, fornendo una descrizione delle funzionalità specifiche svolte da ciascun package.

- **dataManagement:** Contiene sottocartelle per le diverse aree funzionali del sistema, ognuna con le rispettive interfacce.

- **autenticazione:** Fornisce interfacce user-friendly per garantire un accesso controllato al sistema e facilitare la gestione delle credenziali utente.

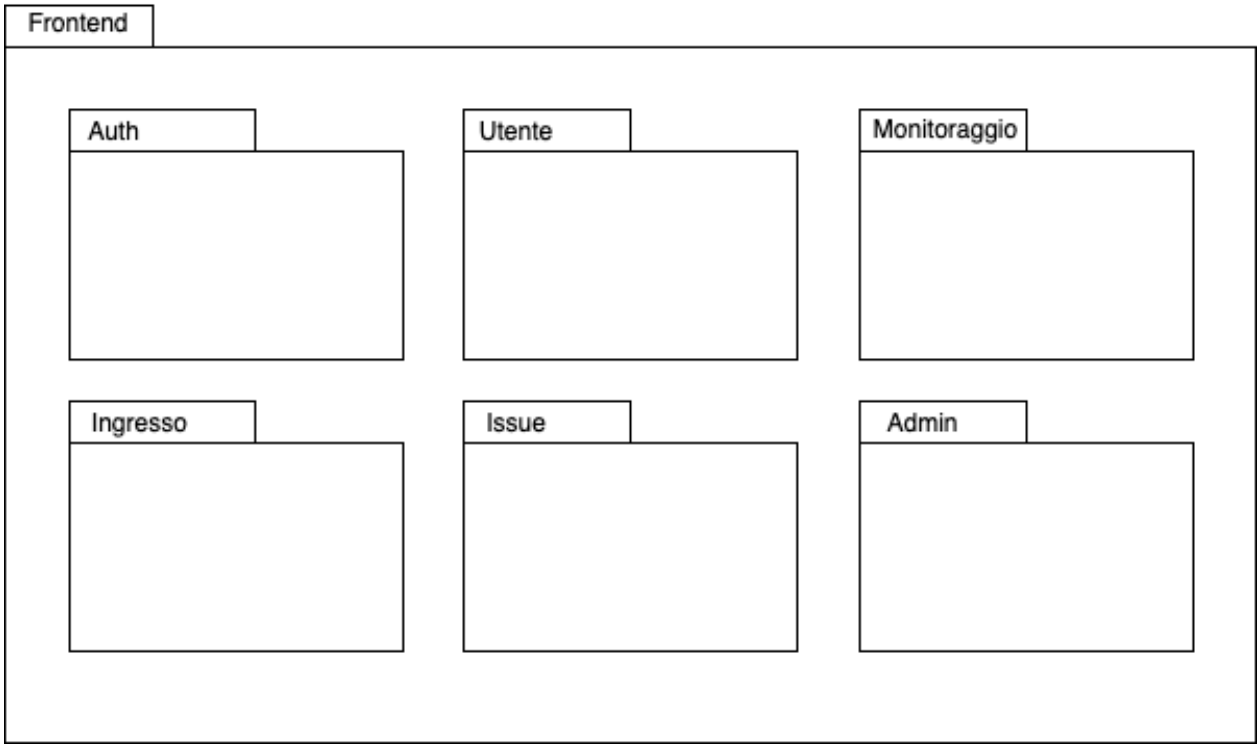
- **utente:** Raccoglie interfacce, modelli e servizi frontend per la gestione degli utenti nell'applicazione SmartCargo. Include funzionalità come la modifica del profilo, consentendo una gestione efficiente delle informazioni utente attraverso interfacce intuitive.

- **monitoraggio:** Integra interfacce frontend per il monitoraggio delle operazioni attive e delle operazioni di carico/scarico all'interno del porto in SmartCargo. Fornisce una visione chiara delle attività chiave attraverso interfacce appositamente progettate per garantire un monitoraggio efficace.

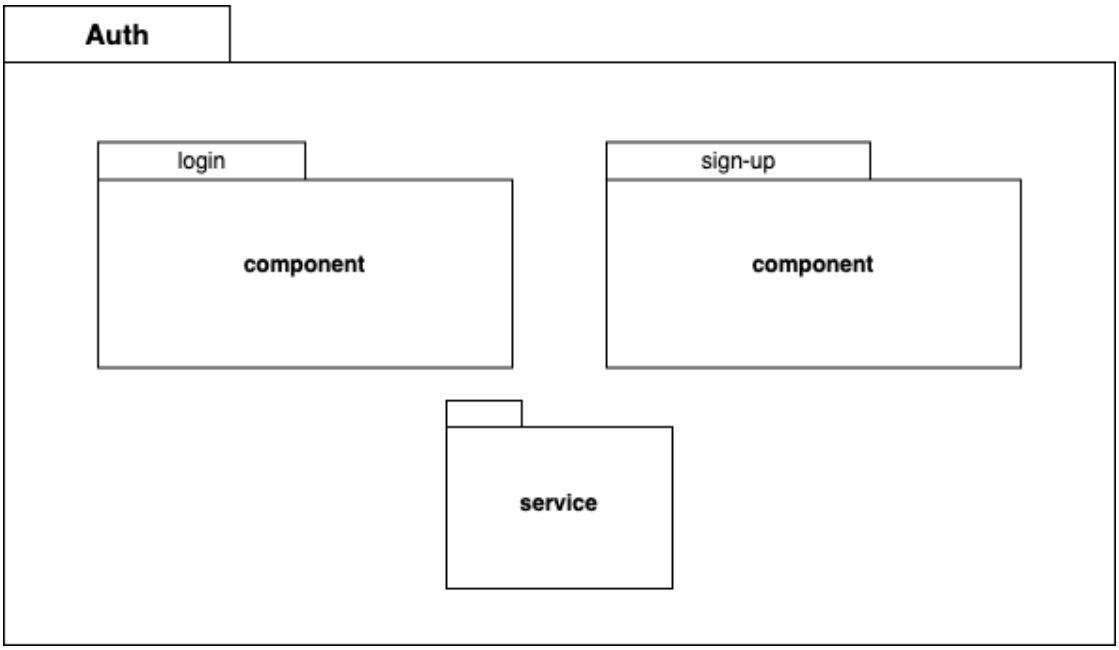
- **ingresso:** Gestisce le operazioni di registrazione e convalida dell'ingresso degli autotrasportatori attraverso interfacce frontend. Semplifica la gestione delle fasi di registrazione e convalida prenotazioni, offrendo un'interfaccia chiara e intuitiva.

- **issue:** Fornisce interfacce frontend dedicate per la creazione e la modifica di issue associate alle operazioni attive nel porto nell'applicazione SmartCargo. Questo package facilita la gestione delle segnalazioni di problemi, offrendo agli operatori un'interfaccia intuitiva per registrare e aggiornare le problematiche riscontrate durante l'esecuzione delle operazioni portuali.

- **admin:** Raccoglie interfacce, modelli e servizi frontend per la gestione amministrativa degli account utenti in SmartCargo. Concentrandosi su modifiche, impostazioni dei privilegi e creazione di account, questo package offre un'interfaccia centralizzata per la gestione degli account con privilegi amministrativi




## Package Autenticazione




| Package | Descrizione |
|---|---|

| | |
|---|---|
| login | Package che si occupa dell'interattività, della formattazione e dello stile degli elementi grafici per quanto riguarda l'autenticazione nel sistema |
| signup | Package che si occupa dell'interattività, della formattazione e dello stile degli elementi grafici per quanto riguarda la registrazione dell'account all'interno del sistema |

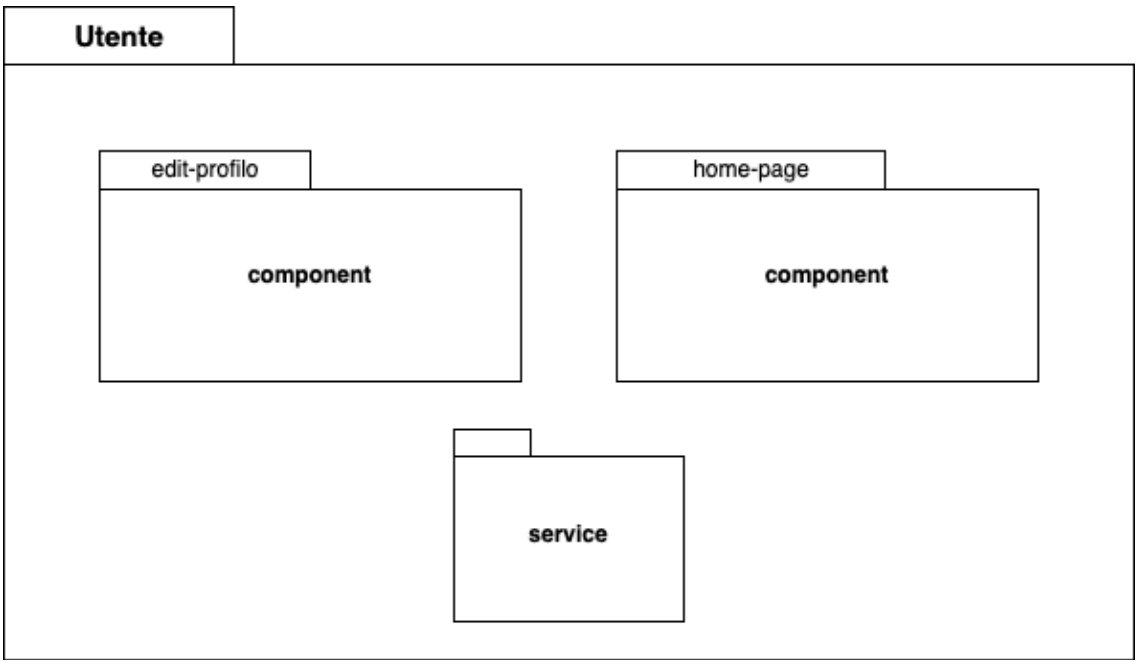
1. **Package Login**

| Classe | Descrizione |
|---|---|
| login.component.html | Classe che organizza la formattazione degli elementi html della pagina. |
| login.component.scss | Classe che organizza lo stile degli elementi html della pagina. |
| login.component.spec.ts | Classe che si occupa della logica dei test. |
| login.component.ts | Classe che si occupa della logica degli elementi grafici della pagina. |

2. **Package Signup**

| Classe | Descrizione |
|---|---|
| signup.component.html | Classe che organizza la formattazione degli elementi html della pagina |
| signup.component.scss | Classe che organizza lo stile degli elementi html della pagina |
| signup.component.spec.ts | Classe che si occupa della logica dei test. |
| signup.component.ts | Classe che si occupa della logica degli elementi grafici della pagina |

### 3.2.2 Package Utente




| Package | Descrizione |
|---|---|
| edit-profilo | Package che si occupa dell'interattività, della formattazione e dello stile degli elementi grafici per quanto riguarda la modifica del profilo utente nel sistema. |
| home-page | Package che si occupa dell'interattività, della formattazione e dello stile degli elementi grafici per quanto riguarda la pagina home |

### 1. Package Edit Profilo

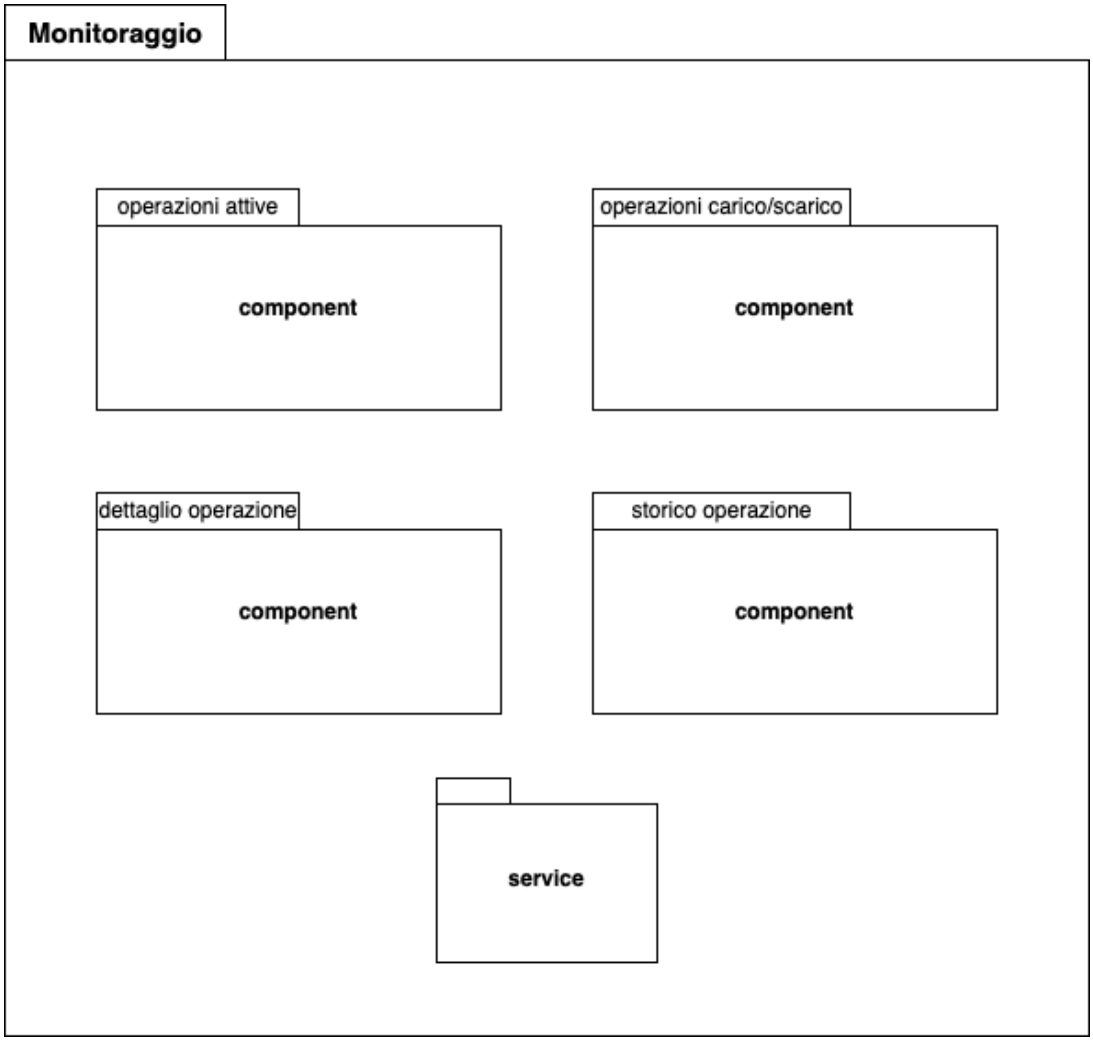
| Classe | Descrizione |
|---|---|
| edit-profilo.component.html | Classe che organizza la formattazione degli elementi html della pagina |
| edit-profilo.component.scss | Classe che organizza lo stile degli elementi html della pagina |
| edit-profilo.component.spec.ts | Classe che si occupa della logica dei test. |

| | |
|---|---|
| edit-profilo.component.ts | Classe che si occupa della logica degli elementi grafici della pagina |

### 2. Package Home Page

| Classe | Descrizione |
|---|---|
| home-page.component.html | Classe che organizza la formattazione degli elementi html della pagina |
| home-page.component.scss | Classe che organizza lo stile degli elementi html della pagina |
| home-page.component.spec.ts | Classe che si occupa della logica dei test. |
| home-page.component.ts | Classe che si occupa della logica degli elementi grafici della pagina |

### 3.2.3 Package Monitoraggio




| Package | Descrizione |
|---|---|
| operazioni-attive | Package che si occupa dell'interattività, della formattazione e dello stile degli elementi grafici per quanto riguarda il monitoraggio delle operazioni attive nel sistema |

| | |
|---|---|
| dettaglio-operazione | Package che si occupa dell'interattività, della formattazione e dello stile degli elementi grafici per quanto riguarda il dettaglio delle operazioni attive nel sistema |

| | |
|---|---|
| operazioni-car-scar | Package che si occupa dell'interattività, della formattazione e dello stile degli elementi grafici per quanto riguarda il monitoraggio delle operazioni di carico/scarico nel sistema |
| storico-operazioni | Package che si occupa dell'interattività, della formattazione e dello stile degli elementi grafici per quanto riguarda lo storico delle attività presenti nel sistema |

### 1. Package Operazioni Attive

| Classe | Descrizione |
|---|---|
| operazioni-attive.component.html | Classe che organizza la formattazione degli elementi html della pagina |
| operazioni-attive.component.scss | Classe che organizza lo stile degli elementi html della pagina |
| operazioni-attive.component.spec.ts | Classe che si occupa della logica dei test. |
| operazioni-attive.component.ts | Classe che si occupa della logica degli elementi grafici della pagina |

### 2. Package Dettaglio Operazione

| Classe | Descrizione |
|---|---|
| dettaglio-operazione.component.html | Classe che organizza la formattazione degli elementi html della pagina |
| dettaglio-operazione.component.scss | Classe che organizza lo stile degli elementi html della pagina |
| dettaglio-operazione.component.spec.ts | Classe che si occupa della logica dei test . |
| dettaglio-operazione.component.ts | Classe che si occupa della logica degli elementi grafici della pagina |

### 1. Package Operazioni Car Scar

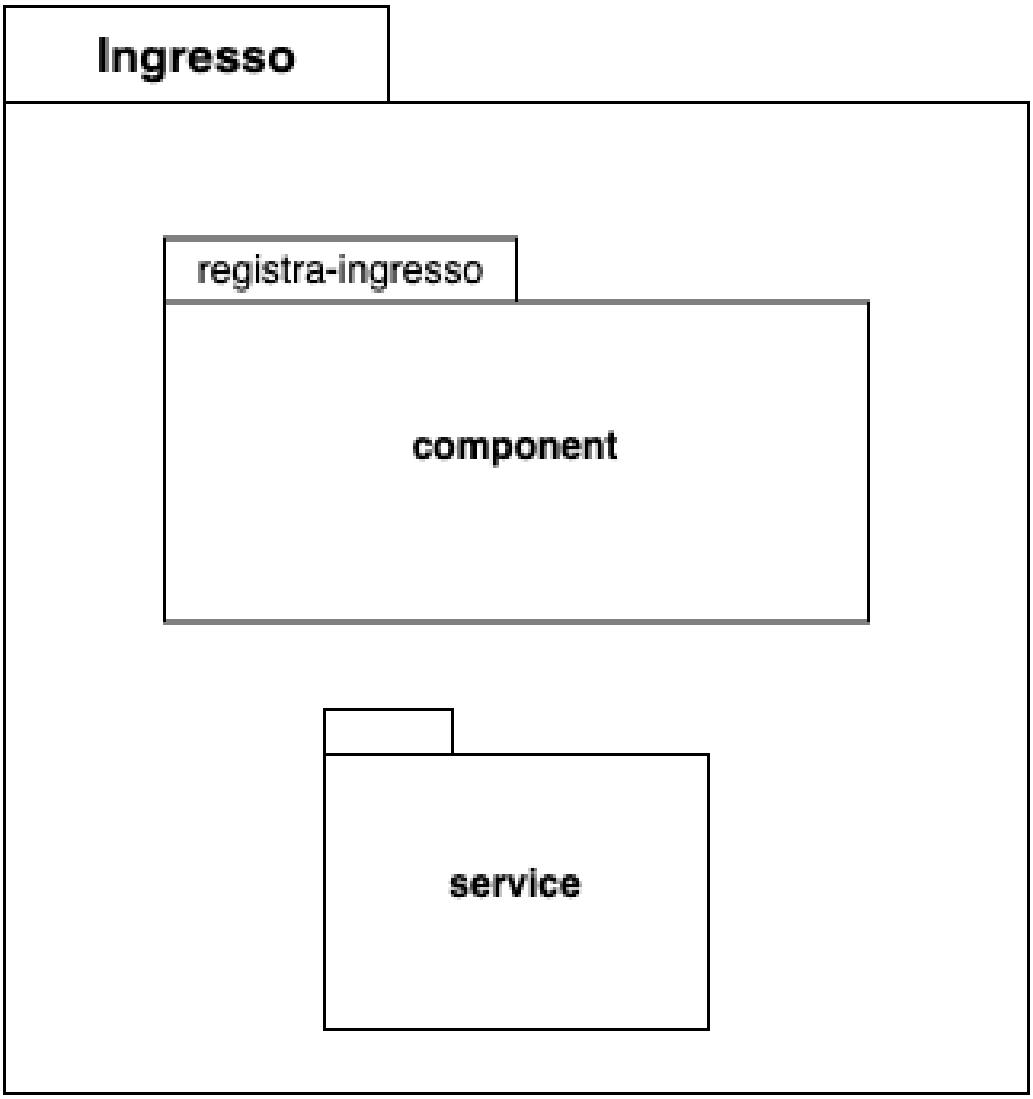
| Classe | Descrizione |
|---|---|
| operazioni-car-scar.component.html | Classe che organizza la formattazione degli elementi html della pagina |
| operazioni-car-scar. component.scss | Classe che organizza lo stile degli elementi html della pagina |
| operazioni-car-scar.component.spec.ts | Classe che si occupa della logica dei test . |
| operazioni-car-scar.component.ts | Classe che si occupa della logica degli elementi grafici della pagina |

### 2. Package Storico Operazioni

| Classe | Descrizione |
|---|---|
| storico-operazioni.component.html | Classe che organizza la formattazione degli elementi html della pagina |
| storico-operazioni.component.scss | Classe che organizza lo stile degli elementi html della pagina |

| | |
|---|---|
| storico-operazioni.component.spec.ts | Classe che si occupa della logica dei test . |
| storico-operazioni.component.ts | Classe che si occupa della logica degli elementi grafici della pagina |

### 3.2.4 Package Ingresso



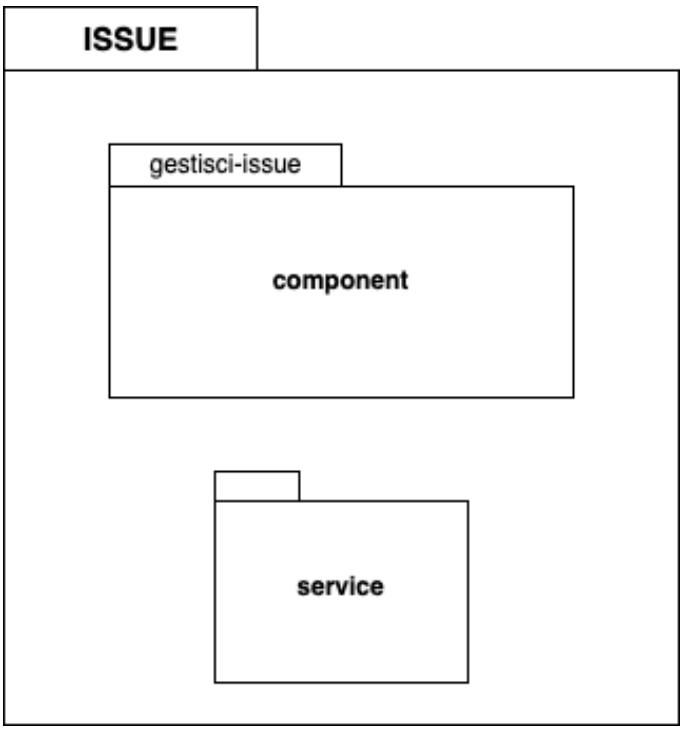

| Package | Descrizione |
|---|---|

| | |
|---|---|
| registra-ingresso | Package che si occupa dell'interattività, della formattazione e dello stile degli elementi grafici per quanto riguarda la gestione dell'ingresso nel porto. |

### 1. Package Registra Ingresso

| Classe | Descrizione |
|---|---|
| registra-ingresso.component.html | Classe che organizza la formattazione degli elementi html della pagina |
| registra-ingresso.component.scss | Classe che organizza lo stile degli elementi html della pagina |
| registra-ingresso.component.spec.ts | Classe che si occupa della logica dei test . |
| registra-ingresso component.ts | Classe che si occupa della logica degli elementi grafici della pagina |

### 3.2.5 Package Issue



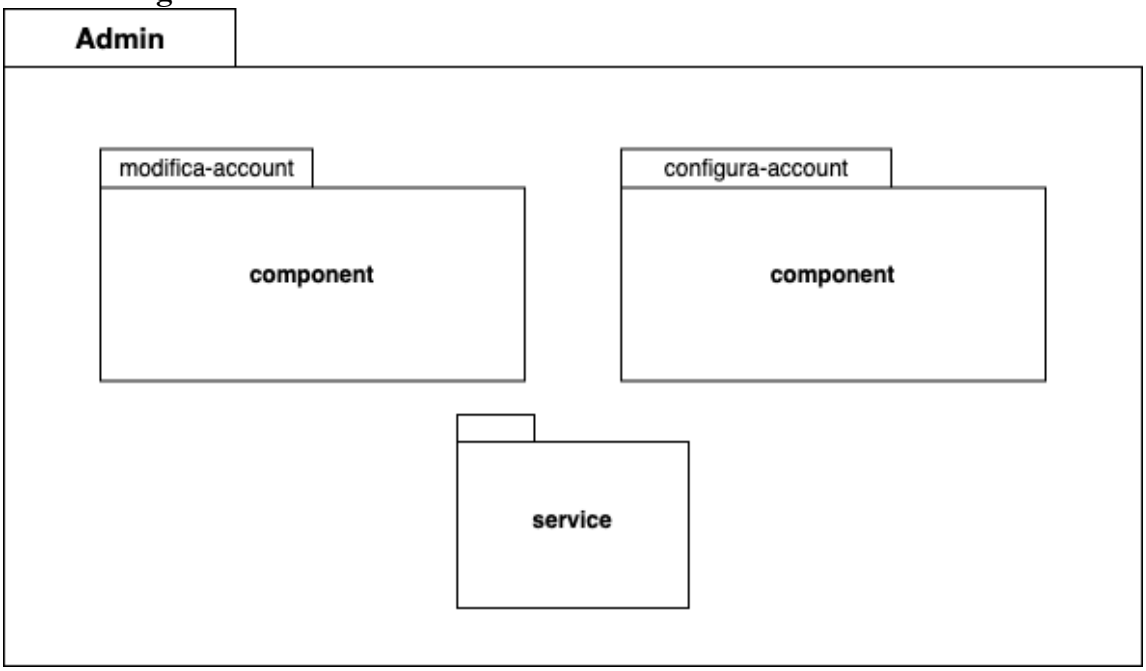

| Package | Descrizione |
|---|---|
| gestisci-issue | Package che si occupa dell'interattività, della formattazione e dello stile degli elementi grafici per quanto riguarda la gestione delle issue. |

## 1. Package Gestisci Issue

| Classe | Descrizione |
|---|---|
| gestisci-issue.component.html | Classe che organizza la formattazione degli elementi html della pagina |
| gestisci-issue.component.scss | Classe che organizza lo stile degli elementi html della pagina |
| gestisci-issue.component.spec.ts | Classe che si occupa della logica dei test . |

| | |
|---|---|
| gestisci-issue.component.ts | Classe che si occupa della logica degli elementi grafici della pagina |

### 3.2.6 Package Admin




| Package | Descrizione |
|---|---|
| modifica-account | Package che si occupa dell'interattività, della formattazione e dello stile degli elementi grafici per quanto riguarda la modifica dell'account all'interno del sistema |
| configura-account | Package che si occupa dell'interattività, della formattazione e dello stile degli elementi grafici per quanto riguarda la modifica dei permessi dell'account all'interno del sistema |

1. **Package Modifica Account**

| Classe | Descrizione |
|---|---|
| modifica-account.component.html | Classe che organizza la formattazione degli elementi html della pagina |
| modifica-account.component.scss | Classe che organizza lo stile degli elementi html della pagina |
| modifica-account.component.spec.ts | Classe che si occupa della logica dei test. |
| modifica-account.component.ts | Classe che si occupa della logica degli elementi grafici della pagina |

2. **Package Configura Account**

| Classe | Descrizione |
|---|---|
| configura-account.component.html | Classe che organizza la formattazione degli elementi html della pagina |
| configura-account.component.scss | Classe che organizza lo stile degli elementi html della pagina |
| configura-account.component.spec.ts | Classe che si occupa della logica dei test. |
| configura-account.component.ts | Classe che si occupa della logica degli elementi grafici della pagina |

## 3.2. Packages Back-end

In questa sezione viene mostrata la suddivisione del sistema in package per quanto riguarda il Backend.

- **Backend**
  - **Src**
    - dataManagement
      - autenticazione
        - LoginController.py

- ingresso
  - IngressoController.py
- monitoraggio
  - MonitoraggioController.py
  - detection
- operazioni
  - OperazioniController.py
- issue
  - IssueController.py
- account
  - AccountController.py
  - AccountAutotrasportatoreController.py
- Services
  - AutotrasportatoreService.py
  - VeicoloService.py
  - QrcodeService.py
  - OperatoreIngressoService.py
  - OperatoreMagazzinoService.py
  - OperatoreMobileService.py
  - OperatoreSalaService.py
  - IssueService.py
  - OperazioneService.py
  - IncludeService.py
  - MerceService.py
  - PercorsoService.py

- models
  - Autotrasportatore.py
  - Veicolo.py
  - Qrcode.py
  - UtenteRegistrato.py
  - Issue.py
  - Operazione.py
  - Include.py
  - Merce.py
  - Percorso.py
  - AutotrasportatoreDAO.py
  - VeicoloDAO.py
  - QrcodeDAO.py
  - OperatoreIngressoDAO.py
  - OperatoreMagazzinoDAO.py
  - OperatoreMobileDAO.py

- OperatoreSalaDAO.py
- IssueDAO.py
- OperazioneDAO.py
- IncludeDAO.py
- MerceDAO.py
- PercorsoDAO.py

- config
  - database.py
- app.py
- migrations
- tests
- Requirements.txt

### 3.2.1 Struttura Backend

In questa sezione, viene esposta la struttura del package Back end di SmartCargo, fornendo una descrizione delle funzionalità specifiche svolte da ciascun package.

- **dataManagement:** Contiene sottocartelle per le diverse aree funzionali del sistema, ognuna con i rispettivi controller e inoltre i vari services utili alle funzionalità dei controller.

- **autenticazione**: Gestisce l'autenticazione degli utenti.
  - **LoginController.py:** Contiene i metodi per gestire le richieste relative all'autenticazione.

- **ingresso**: Gestisce le operazioni di ingresso.
  - **IngressoController.py:** Contiene i metodi per gestire le richieste relative alle operazioni di ingresso.

- **monitoraggio:** Gestisce le operazioni di monitoraggio.
  - **MonitoraggioController.py:** Contiene i metodi per gestire le richieste relative alle operazioni di monitoraggio.

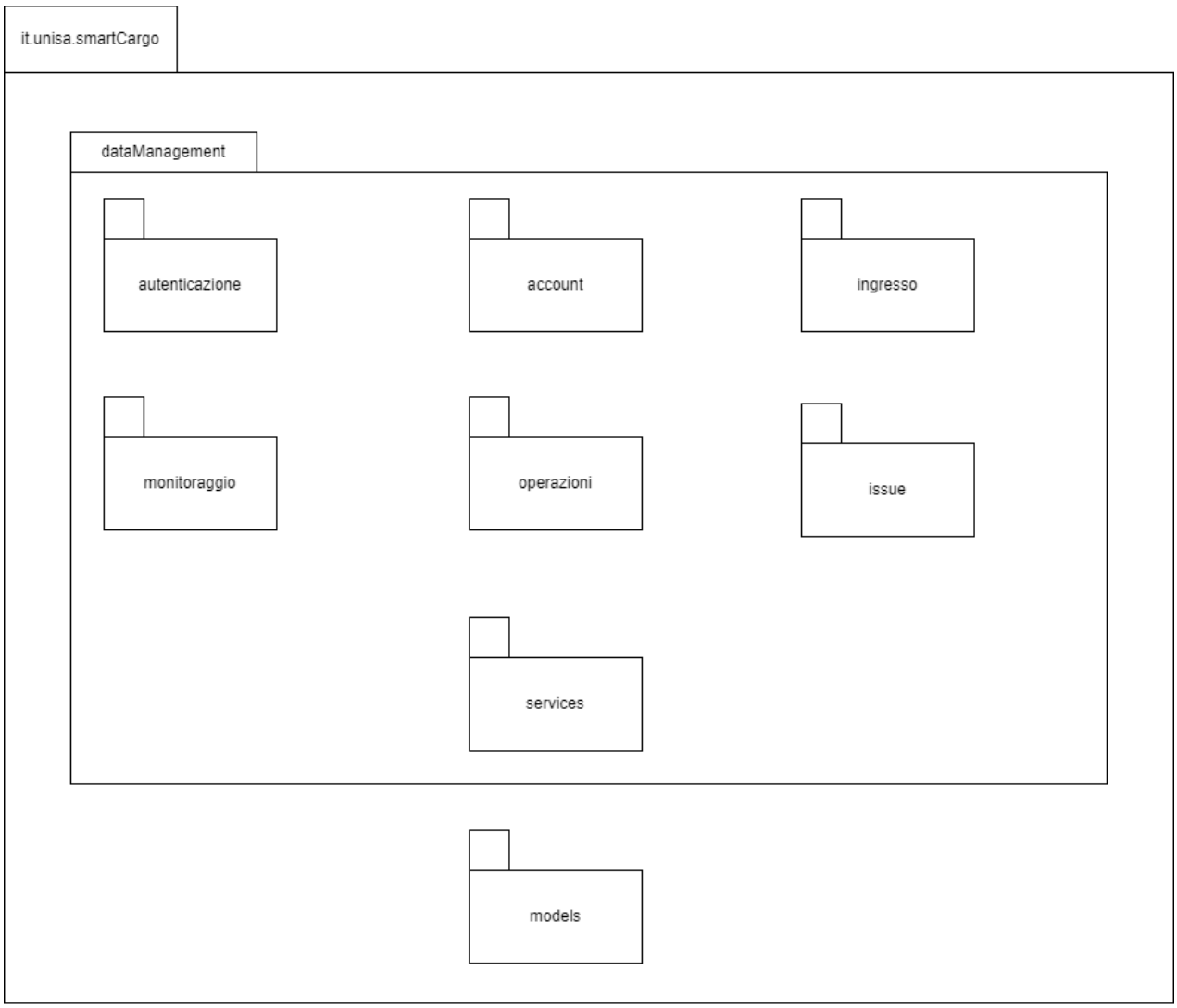
- **operazioni:** Gestisce operazioni generiche.
  - **OperazioniController.py:** Contiene i metodi per gestire le richieste relative alle operazioni generiche.

- **issue**: Gestisce le issue.
  - **IssueController.py:** Contiene i metodi per gestire le richieste relative alle issue.

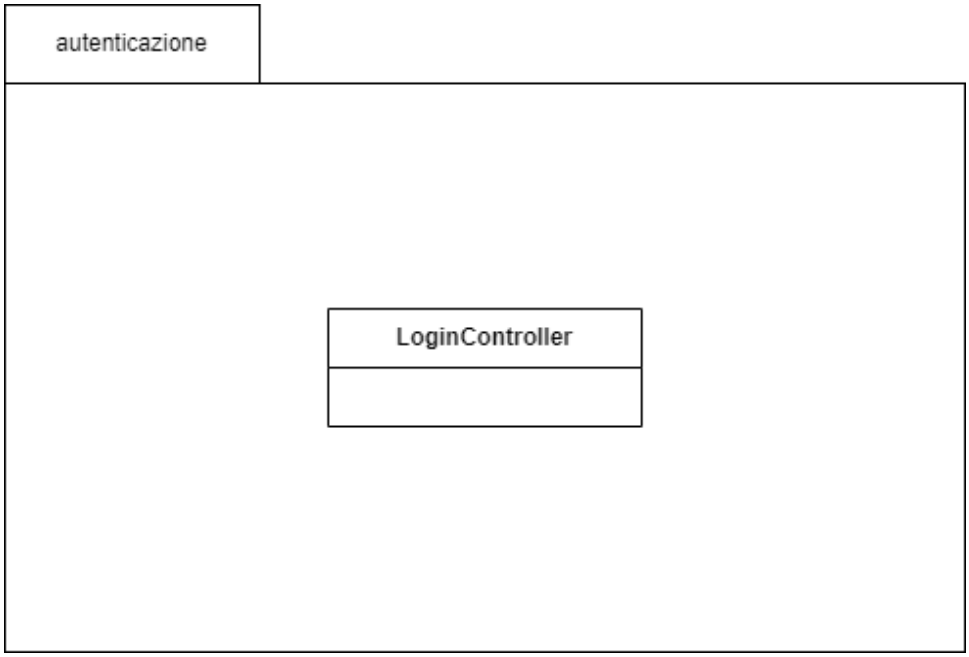
- **account**: Gestisce le operazioni legate agli account utente.

  - **AccountController.py:** Contiene i metodi per gestire le richieste relative alle operazioni sugli account.
  - **AccountAutotrasportatoreController.py:** Contiene i metodi per gestire le richieste relative alle operazioni sugli account degli autotrasportatori.

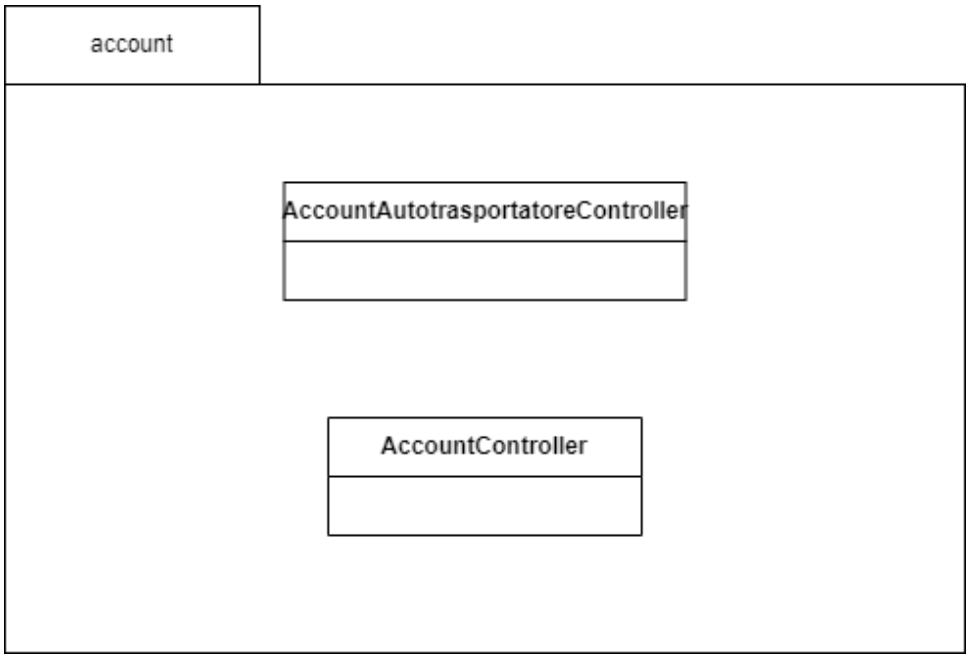
- **models:** Contiene le classi che rappresentano le entità del sistema.
- **config:** Contiene configurazioni dell'applicazione, come configurazioni di database e altre impostazioni.
- **app.py:** Il file principale che configura l'applicazione.
- **tests:** Contiene test per verificare il corretto funzionamento del backend.
- **requirements.txt:** Elenca le dipendenze del progetto Python.


it.unisa.smartCargo
dataManagement
autenticazione
account
ingresso
monitoraggio
operazioni
issue
services
models

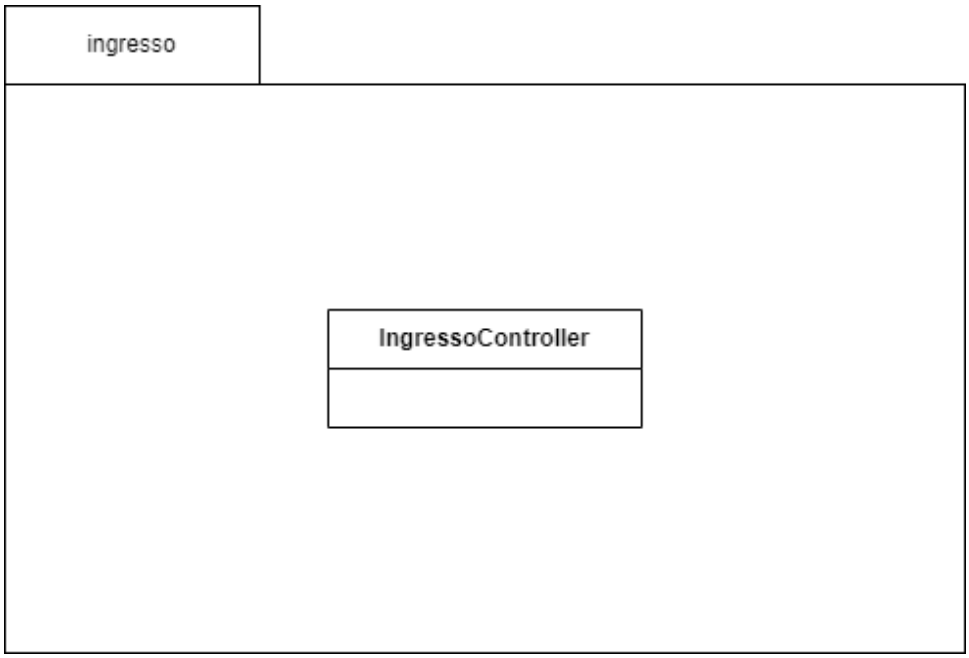
**Package Autenticazione**



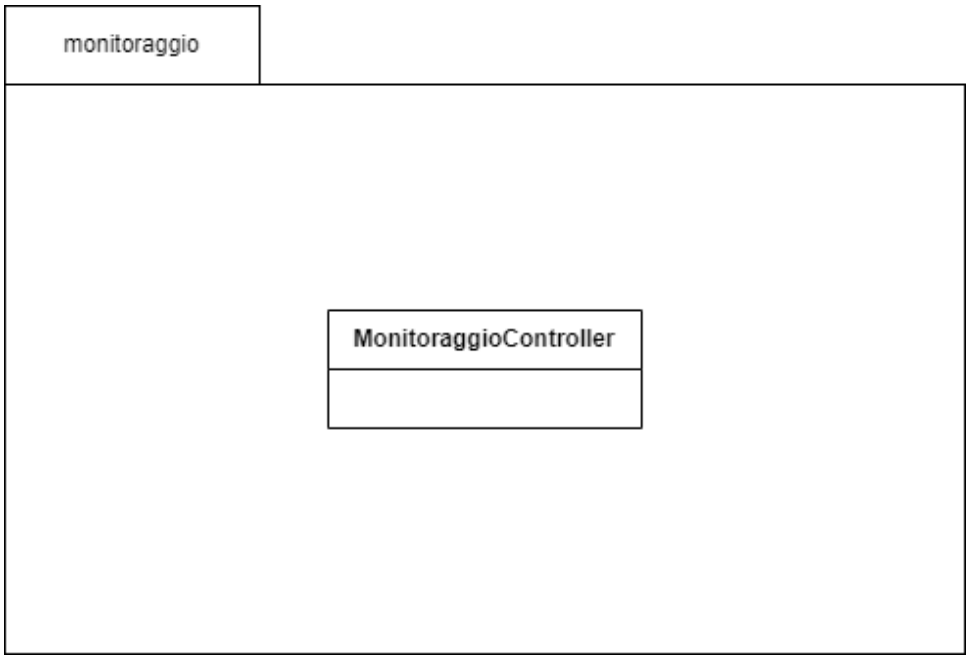

**Package Account**



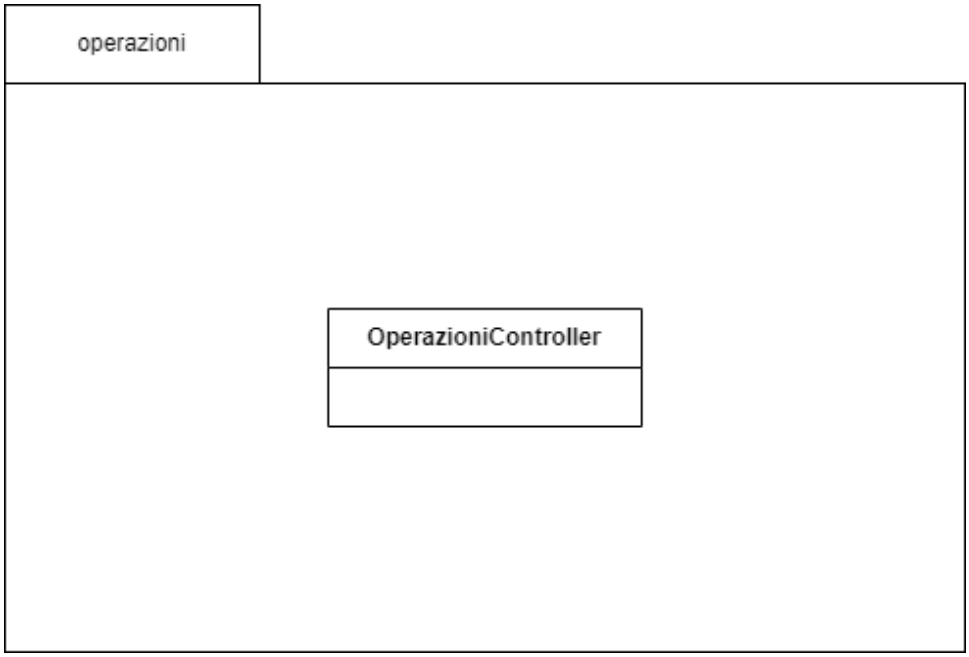
**Package Ingresso**



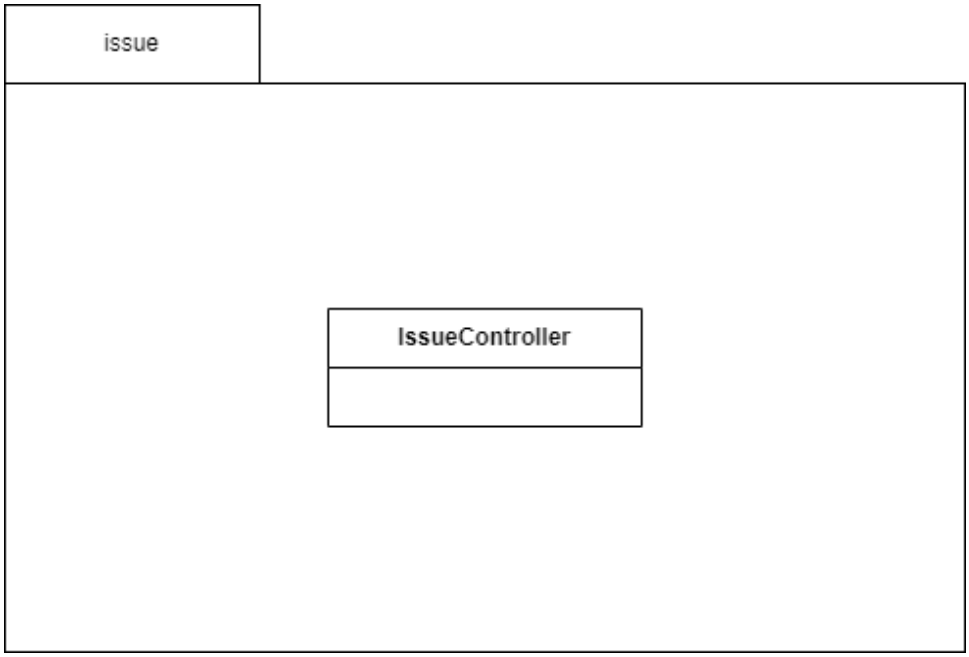

**Package Monitoraggio**



**Package Operazioni**




**Package Issue**



# 4. Class Interface

**Package autenticazione**

| Nome classe | **autenticazioneController** |
|---|---|
| **Descrizione** | Questa classe permette di gestire la logica di business dell'autenticazione. |
| **Metodi** | **+login(String email, String password): Utente**<br>**+logout()** |
| **Invariante di classe** | / |

| Nome Metodo | **+login(String email, String password)** |
|---|---|
| **Descrizione** | Questo metodo permette l'accesso ad un utente registrato. |
| **Pre-condizione** | / |
| **Post-condizione** | **context:** autenticazioneController::login(email,password)<br>**post:** isAutotrasportatore(loggedUser) \|\| isOpSala(loggedUser) \|\| isOpIngresso(loggedUser) \|\| isMagazzino(loggedUser) \|\| isMobile(loggedUser) \|\| isAmministratore(loggedUser) == true |

| **Nome Metodo** | **+logout()** |
|---|---|
| **Descrizione** | Questo metodo permette di disconnettere un utente autenticato dal sistema. |
| **Pre-condizione** | / |
| **Post-condizione** | / |

**Package Ingresso**

| **Nome classe** | **ingressoService** |
|---|---|
| **Descrizione** | Contiene i servizi per la logica di business dell'ingresso. |
| **Metodi** | **+validaAccesso(int idAutotrasportatore, Operazione operazione): Boolean valido**<br>**+registrazioneIngresso(Autotrasportatore autotrasportatore, Operazione operazione)** |
| **Invariante di classe** | / |

| **Nome Metodo** | **+validaAccesso(int idAutotrasportatore, Operazione operazione): Boolean valido** |
|---|---|
| **Descrizione** | Questo metodo permette di validare l'accesso all'autotrasportatore. |
| **Pre-condizione** | **context**:ingressoController::validaAccesso(int idAutotrasportatore, Operazione operazione)<br>**pre**: idAutotrasportatore != null<br>and<br>operazione! = null<br>and<br>visualizzaListaUtenti(idAutotrasportatore).contains(account) |
| **Post-condizione** | / |

| **Nome Metodo** | **+registrazioneIngresso(dict infoOperazione)** |
|---|---|
| **Descrizione** | Questo metodo permette di registrare i dati inerenti all'autotrasportatore e alle operazioni che svolgerà. |
| **Pre-condizione** | **context**:ingressoController::registrazioneIngresso(Dict infoOperazione)<br>**pre**: idAutotrasportatore != null |
| **Post-condizione** | / |

**Package Issue**

| Nome classe | **IssueController** |
|---|---|
| **Descrizione** | Questa classe permette di gestire la logica di business della Issue. |
| **Metodi** | **+nuovaIssue(Issue issue): Issue**<br>**+modificaIssue(Issue issue): Issue** |
| **Invariante di classe** | / |

| Nome Metodo | **+nuovaIssue(dict infoIssue): Issue** |
|---|---|
| **Descrizione** | Questo metodo permette la creazione di una nuova Issue. |
| **Pre-condizione** | **context:** issueController:: nuovaIssue(dict infoIssue)<br>**pre:** !IssueDAO.doRetrieve().contains(issue) |
| **Post-condizione** | **context:** issueController:: nuovaIssue(dict infoIssue)<br>**post**:IssueDAO.save(issue)==true |

| Nome Metodo | **+modificaIssue(dict infoIssue): Issue** |
|---|---|
| **Descrizione** | Questo metodo permette la modifica di una Issue. |
| **Pre-condizione** | **context:** issueController:: modificaIssue(dict infoIssue)<br>**pre:** IssueDAO.doRetrieve().contains(issue) |
| **Post-condizione** | / |

**Package Operazione**

| Nome classe | **operazioneController** |
|---|---|
| **Descrizione** | Questa classe permette di gestire la logica di business per una corretta esecuzione delle operazioni. |
| **Metodi** | **+segnalazioneEsitoOperazione(dict esitoOp): operazione** |
| **Invariante di classe** | / |

| Nome Metodo | **+segnalazioneEsitoOperazione(dict esitoOp): operazione** |
|---|---|
| **Descrizione** | Questo metodo permette all'Operatore di Magazzino di poter segnalare l'esito di un'operazione. |
| **Pre-condizione** | **context:** operazioneController::segnalazioneEsitoOperazione(**dict esitoOp**)<br>**pre:** idOperazione != null and idOperazione > 0 and operazioneDAO.doRetrieveById(int idOperazione) != null |
| **Post-condizione** | **context:** operazioneController::segnalazioneEsitoOperazione(**dict esitoOp**)<br>**post: operazione.getStato() == closed or operazione.getStato() == issue** |

**Package Monitoraggio**

| Nome classe | **monitoraggioController** |
|---|---|
| **Descrizione** | Questa classe permette di gestire la logica di business per una corretta esecuzione del monitoraggio. |
| **Metodi** | **+detection(Percorso, int soglia): Percorso**<br>**+visualizzaStorico(dict filtri): operazioni[]** |
| **Invariante di classe** | / |

| Nome Metodo | **+detection(Percorso, int soglia): Percorso** |
|---|---|
| **Descrizione** | Questo metodo permette all'Operatore di Sala di poter visualizzare i percorsi effettuati dall'autotrasportatore. |
| **Pre-condizione** | **context:** monitoraggioController::detection(Percorso, int soglia)<br>**Pre:** Percorso != null and soglia > 0 |
| **Post-condizione** | / |

| Nome Metodo | **+visualizzaStorico(dict filtri): Operazioni[]** |
|---|---|
| **Descrizione** | Questo metodo permette all'operatore di sala di visualizzare uno storico di tutte le operazioni concluse all'interno del porto. |
| **Pre-condizione** | **Context**: monitoraggioController:: visualizzaStorico(dict filtri) : Operazioni[]<br>**pre**: numero_operazioni > 0 |
| **Post-condizione** | / |

**Package Account**

| Nome classe | **accountController** |
|---|---|
| **Descrizione** | Questa classe permette di gestire la logica di business della gestione degli account. |
| **Metodi** | **+CreaAccount(Utente utente): Utente**<br>**+ModificaAccount(Utente utente): Utente** |
| **Invariante di classe** | / |

| Nome Metodo | **+CreaAccount(Utente utente): Utente** |
|---|---|
| **Descrizione** | Questo metodo permette ad un utente di creare un account |
| **Pre-condizione** | **context**: accountController::CreaAccount(Utente utente)<br>**pre**: not visualizzaListaUtenti(account.utente).includes(account) |
| **Post-condizione** | **context**: accountController::CreaAccount(Utente utente)<br>**post**: visualizzaListaUtenti(account.utente).includes(account)<br>and<br>visualizzaListaUtenti(account.utente).size == @pre.<br>visualizzaListaUtenti(account.utente).size+1 |

| Nome Metodo | **+ModificaAccount(Utente utente): Utente** |
|---|---|
| **Descrizione** | Questo metodo permette ad un utente di modificare un account |
| **Pre-condizione** | **context**:<br>accountController::ModificaAccount(Utente utente) |

| | |
|---|---|
| | **pre**: visualizzaListaUtenti(account.utente).includes(account) |
| **Post-condizione** | **context**: accountController::ModificaAccount(Utente utente)<br>**post:** visualizzaListaUtenti(account.utente).size == @pre.visualizzaListaUtenti(account.utente.size |

| **Nome classe** | **accountAutotrasportatoreController** |
|---|---|
| **Descrizione** | Questa classe permette di gestire la logica di business della gestione degli account degli autotrasportatori. |
| **Metodi** | **+registrazioneAutotrasportatore(Utente utente): Utente**<br>**+ModificaAutotrasportatore(Utente utente): Utente** |
| **Invariante di classe** | / |

| **Nome Metodo** | **+ registrazioneAutotrasportatore (Utente utente): Utente** |
|---|---|
| **Descrizione** | Questo metodo permette ad un autotrasportatore di creare un account |
| **Pre-condizione** | **context**: accountController::<br>**registrazioneAutotrasportatore**(Utente utente)<br>**pre**: not visualizzaListaUtenti(account.utente).includes(account) |
| **Post-condizione** | **context**: accountAutotrasportatoreController::<br>registrazioneAutotrasportatore(Utente utente)<br>**post**:<br>visualizzaListaUtenti(account.utente).includes(account) and<br>visualizzaListaUtenti(account.utente).size == @pre. visualizzaListaUtenti(account.utente).size+1 |

| Nome Metodo | **+modificaAutotrasportatore(Utente utente): Utente** |
|---|---|
| **Descrizione** | Questo metodo permette ad un autotrasportatore di modificare un account |
| **Pre-condizione** | **context**: accountController::modificaAutotrasportatore(Utente utente)<br>**pre**: visualizzaListaUtenti(account.utente).includes(account) |
| **Post-condizione** | **context**: accountController::ModificaAccount(Utente utente)<br>**post:** visualizzaListaUtenti(account.utente).size == @pre.visualizzaListaUtenti(account.utente.size |

# 5. Glossario

| Termine | Definizione |
|---|---|
| **Package** | Raccolta di classi, interfacce e sottopackage raggruppati insieme in una struttura gerarchica. |
| **DAO** | Il termine "DAO" sta per "Data Access Object" (Oggetto di Accesso ai Dati). Un DAO è un design pattern che fornisce un'interfaccia astratta per la comunicazione con una fonte di dati, come un database, senza esporre i dettagli specifici della persistenza al resto dell'applicazione. |
| **Facade** | Design pattern strutturale che consente di semplificare l'accesso a sottosistemi più complessi mediante l'implementazione di un'interfaccia unificata. |
| **SCSS** | SCSS (Sassy CSS) è un'estensione del linguaggio CSS che aggiunge alcune funzionalità utili e avanzate per semplificare e migliorare la gestione del codice CSS. |