UNIVERSITÀ DEGLI STUDI DI TRENTO

CivicTrento

# Progetto Ingegneria Del Software

Anno Accademico 2024/2025

## Gruppo ID7


Cecilia Cavosi
cecilia.cavosi@studenti.unitn.it
242644

Matilde Prati
matilde.prati@studenti.unitn.it
236042

Elena Rubbo
elena.rubbo@studenti.unitn.it
235858

Università di Trento
Dipartimento di Ingegneria e Scienza dell'Informazione
Via Sommarive 9, 38123
Povo (TN), Italia
E-mail: disi@disi.unitn.it

# Indice

# 1 Diagramma componenti

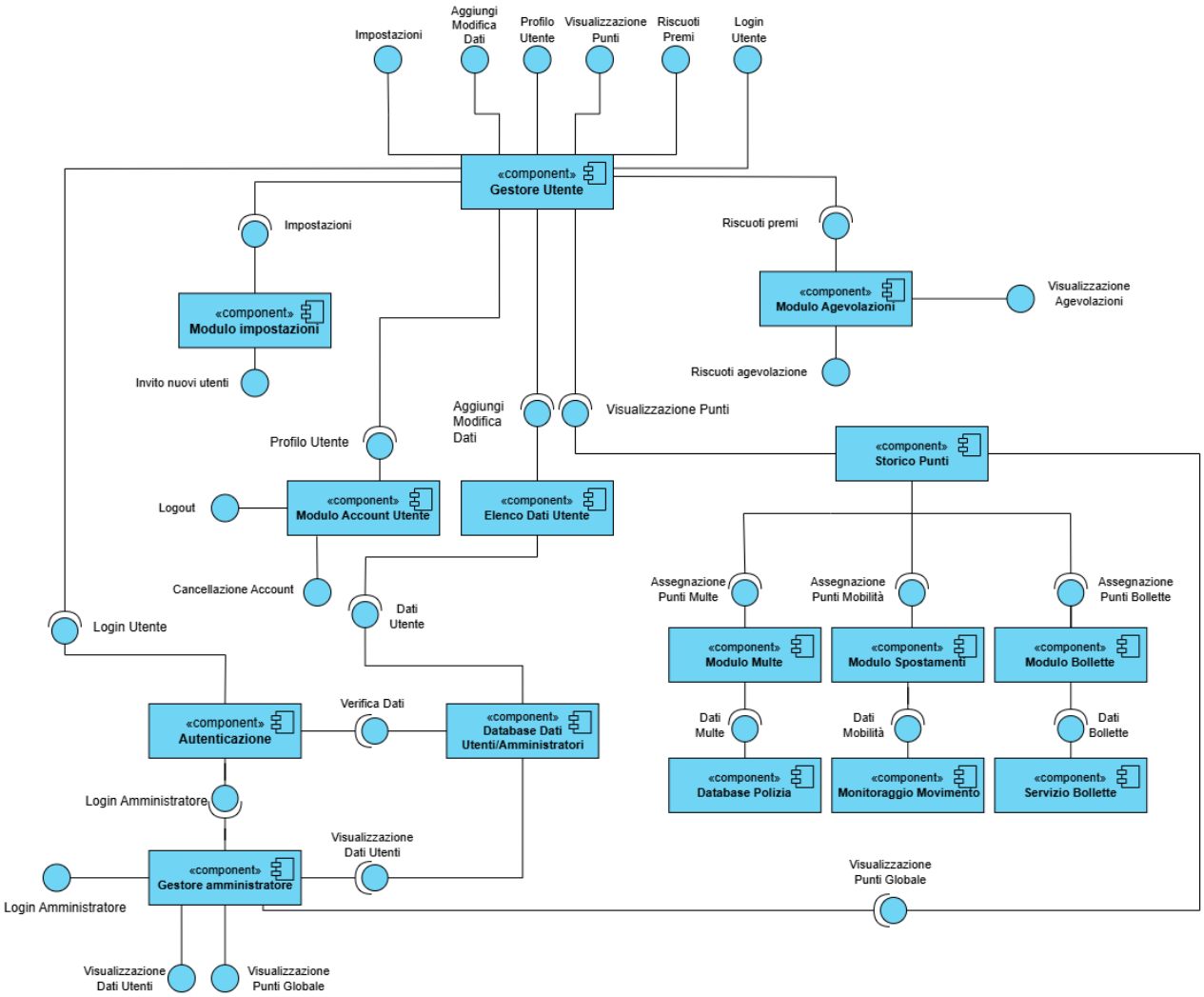
Nel diagramma dei componenti viene rappresentata l'architettura logica del sistema: al centro del sistema si trova il **Gestore Utente**, che costituisce il punto di accesso alle operazioni principali per l'utente; da questo poi si diramano diverse funzionalità come la gestione del profilo, la modifica dei dati personali, la visualizzazione dei punti accumulati e il riscatto di premi e agevolazioni.

È presente inoltre una sezione dedicata agli **Amministratori**, che si lega al proprio gestore e ai meccanismi di autenticazione. Questa parte consente l'accesso a funzionalità di controllo, gestione degli utenti e visualizzazione globale dei punti, mantenendo una chiara distinzione delle attività da svolgere degli amministratori e degli utenti.

Un **modulo di Autenticazione** gestisce l'accesso iniziale degli utenti e degli amministratori, verificando le credenziali e indirizzando l'utente all'interfaccia e le azioni previste per il proprio ruolo, mentre per l'amministratore ci sarà un secondo accesso che lo porterà alla pagina dedicata.

Un altro componente importante è lo **Storico Punti**, che raccoglie e organizza i dati relativi alle azioni compiute dagli utenti e che hanno un impatto sul punteggio accumulato. I componenti da cui provengono queste informazioni sono tre moduli dedicati: **Multe**, **Spostamenti** e **Bollette**. Ognuno di questi si interfaccia a sua volta con basi dati o servizi esterni, assicurando così l'integrazione con fonti affidabili e aggiornate.

## 1.1 Diagramma dei componenti




## 1.2 Descrizione dei componenti

### 1.2.1 Modulo Bollette

Questo componente permette di gestire i dati delle bollette e assegna i Civic Coins all'utente.

| Tipologia | Nome | Descrizione |
|---|---|---|
| Fornita | Assegnazione punti bollette | Il componente fornisce un'interfaccia per assegnare i punti in base ai consumi sulla bolletta |
| Richiesta | Dati bollette | Il componente utilizza un'interfaccia per accedere ai dati delle bollette |

### 1.2.2 Modulo Agevolazioni

Questo componente gestisce la visualizzazione e la riscossione di agevolazioni e premi legati ai Civic Coins.

| Tipologia | Nome | Descrizione |
|---|---|---|
| Fornita | Visualizza agevolazioni | Il componente fornisce l'interfaccia per visualizzare le agevolazioni disponibili |
| Fornita | Riscuoti agevolazione | Il componente fornisce l'interfaccia per riscuotere l'agevolazione scelta |
| Fornita | Riscuoti premi | Il componente fornisce l'interfaccia dove poter riscuotere il premio |

### 1.2.3 Database Dati Utenti/Amministratori

Questo componente rappresenta il database che contiene tutte le informazioni degli utenti e degli amministratori.

| Tipologia | Nome | Descrizione |
|---|---|---|
| Fornita | Dati utente | Il componente fornisce un'interfaccia per permettere all'elenco dati utente di accedere ai dati del cittadino |
| Fornita | Verifica dati | Il componente fornisce un'interfaccia per restituire i dati utente necessari alla verifica delle credenziali durante l'autenticazione |
| Fornita | Visualizza dati utente | Il componente fornisce un'interfaccia per permettere all'amministratore di visualizzare i dati dell'utente |

### 1.2.4 Elenco Dati Utente

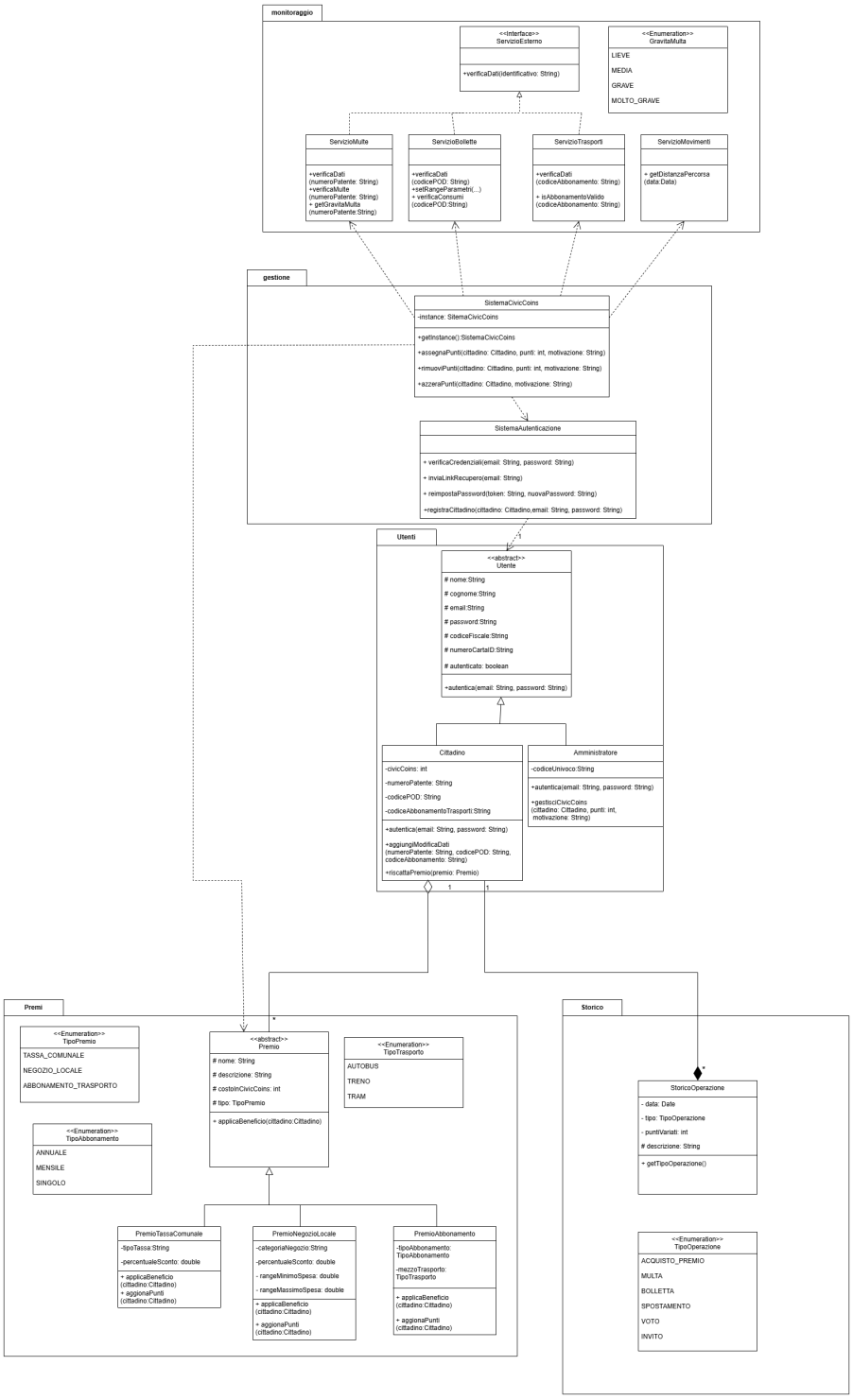
Questo componente gestisce la visualizzazione, l'aggiunta e la modifica dei dati dell'utente registrato all'interno del sistema Civic Coins.

| Tipologia | Nome | Descrizione |
|---|---|---|
| Fornita | Aggiungi/Modifica dati | Il componente fornisce un'interfaccia per poter permettere all'utente di aggiungere o modificare dati |
| Richiesta | Dati Utente | Il componente utilizza un'interfaccia per accedere ai dati utente tramite il database |

# 2 Diagramma Classi

In questa sezione viene presentata un'analisi delle classi necessarie per l'implementazione del sistema, supportata dall'uso del diagramma delle classi.

## 2.1 Diagramma delle classi



Il diagramma delle classi fornisce una rappresentazione statica della struttura del sistema, suddividendo il sistema in package logici, ciascuno con responsabilità ben definite. I package principali rappresentati sono:

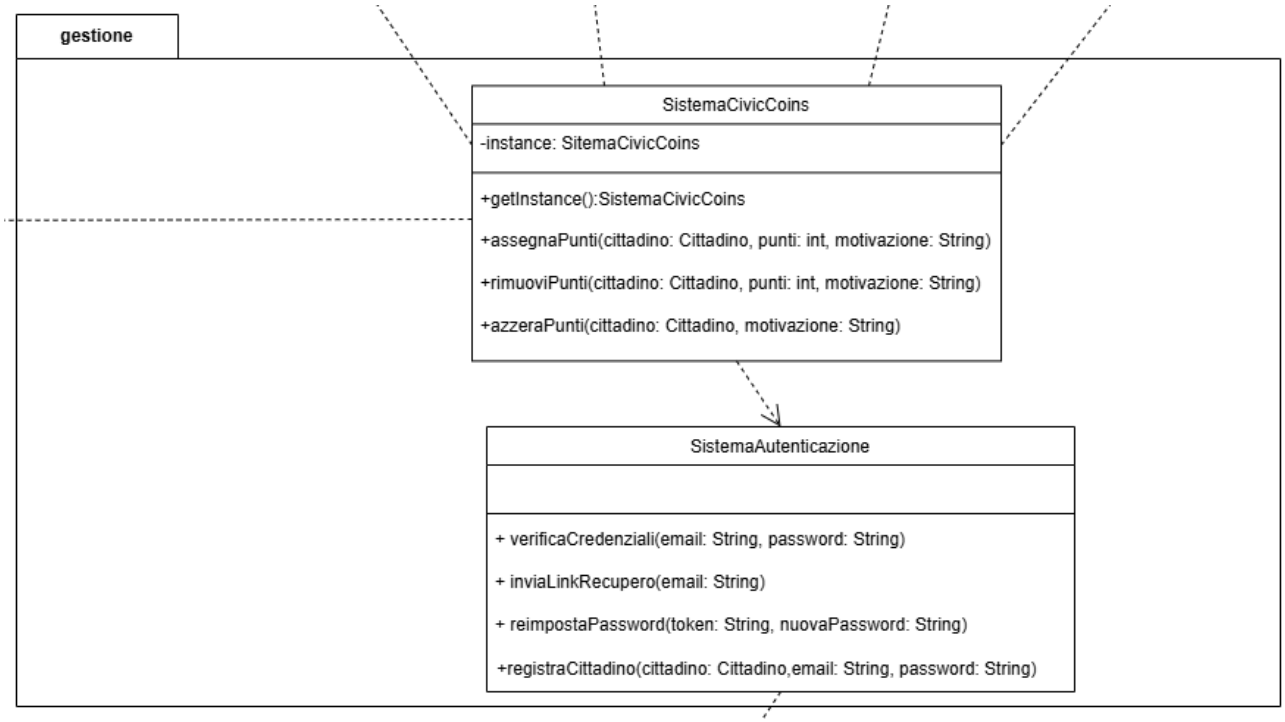
- **Gestione**, che si occupa delle funzionalità centrali come l'assegnazione dei punti e l'autenticazione;
- **Utenti**, dedicato alla gestione delle diverse tipologie di utenti;
- **Premi**, per la rappresentazione dei premi disponibili;
- **Monitoraggio**, che definisce l'interfaccia con i servizi esterni da cui vengono raccolti i dati;
- **Storico**, responsabile del tracciamento delle operazioni effettuate nel sistema;

Le relazioni tra i package evidenziano una struttura coordinata: il package **Gestione** svolge il ruolo di nucleo centrale, interagendo con tutti gli altri pacchetti. Il package **Utenti** rappresenta invece il punto terminale delle interazioni, poiché riceve dati da **Monitoraggio**, interagisce con i **Premi**, e genera operazioni che vengono registrate nello **Storico**.

Tra i principali vantaggi offerti da questo diagramma si evidenziano:

- **Flessibilità**: l'architettura è progettata per facilitare l'estendibilità del sistema, permettendo l'aggiunta di nuovi premi o servizi esterni senza richiedere modifiche strutturali al modello esistente.
- **Sicurezza**: i dati sensibili degli utenti sono adeguatamente protetti grazie all'impiego di meccanismi di incapsulamento e, laddove previsto, tecniche di crittografia.
- **Usabilità e manutenibilità**: la chiara separazione tra la logica di business (incapsulata nel componente *SistemaCivicCoins*) e le entità che rappresentano l'interfaccia con l'utente (come *Cittadino* e *Amministratore*) favorisce una gestione modulare e comprensibile del sistema.

## 2.2 Package Gestione




Il package **gestione** è stato introdotto all'interno del sistema per raggruppare tutti quei servizi centrali che costituiscono il cuore funzionale dell'applicazione, ma che non dipendono direttamente dalle entità. La sua esistenza risponde all'esigenza di separare la logica di orchestrazione dei processi dalla rappresentazione dei dati, migliorando così la modularità e la manutenibilità del sistema.

Le classi che fanno parte del package sono: SistemaCivicCoins e SistemaAutenticazione. La relazione che c'è tra queste è di dipendenza in quanto SistemaAutenticazione dipende da Utente per verificare le credenziali.

### 2.2.1 Classe SistemaCivicCoins

Questa classe serve per la gestione dei punti CivicCoins assegnati o decurtati all'utente, secondo comportamenti virtuosi o penalizzazioni.

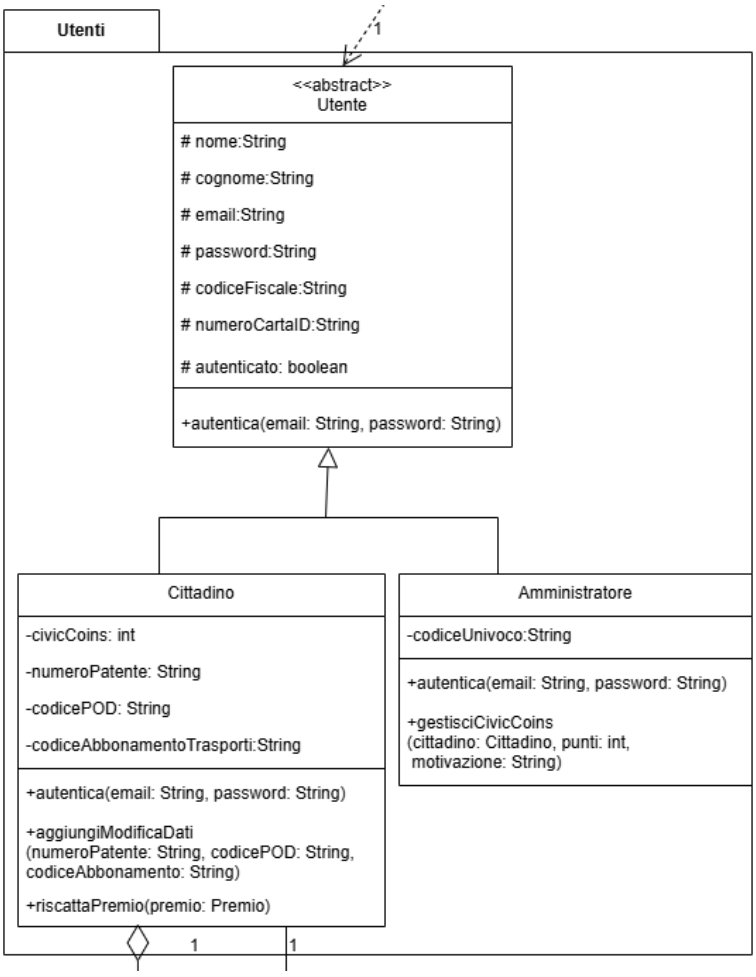
| **Operazione** | **Descrizione** |
|---|---|
| assegnaPunti(Cittadino cittadino, int punti, String motivazione) | Assegna una certa quantità di punti a un cittadino, scrivendo una breve spiegazione del motivo di tale operazione. |
| rimuoviPunti(Cittadino cittadino, int punti, String motivazione) | Rimuove punti al cittadino, scrivendo una breve spiegazione del motivo di tale motivazione. |
| azzeraPunti(Cittadino cittadino, String motivazione) | Imposta a zero il montante punti del cittadino, scrivendo una breve spiegazione del motivo di tale motivazione. |

### 2.2.2 Classe SistemaAutenticazione

Questa classe gestisce il login, il recupero della password e la sicurezza del sistema.

| **Operazione** | **Descrizione** |
|---|---|
| verificaCredenziali(String email, String password) | Legge email e password inserite dall'utente e verifica se corrispondono a un account registrato. |
| inviaLinkRecupero(String email) | Invia una mail all'utente contenente un link per il recupero della password. |
| reimpostaPassword(String token, String nuovaPassword) | Consente all'utente di impostare una nuova password, a seguito di una procedura di recupero tramite email. Utilizza un token temporaneo precedentemente inviato all'utente per garantire la sicurezza della richiesta. |
| registraCittadino(cittadino: Cittadino, email: String, password: String) | Consente al sistema di registrare l'account dell'utente nel database |

## 2.3 Package Utenti



Il package utenti definisce la gerarchia delle classi relative ai soggetti che interagiscono con il sistema. Esso comprende la classe astratta Utente, da cui derivano le classi concrete Cittadino e Amministratore. Tali classi rappresentano le uniche tipologie di utenti previste dal sistema e sono gli utilizzatori diretti dei servizi forniti dagli altri package. La scelta di rendere Utente una classe astratta è motivata dal fatto che un'istanza generica di utente non ha significato applicativo: ogni utente deve necessariamente assumere il ruolo specifico di cittadino o di amministratore.

Le relazioni tra le classi all'interno del package utenti si articolano secondo diverse tipologie. In primo luogo, è presente una relazione di generalizzazione tra la classe astratta Utente e le sue specializzazioni Cittadino e Amministratore, le quali ereditano attributi e comportamenti comuni dalla superclasse. Inoltre, sussiste una relazione di composizione tra Cittadino e StoricoOperazione: ogni cittadino possiede una lista di oggetti StoricoOperazione, la cui esistenza è strettamente dipendente da quella del cittadino stesso; qualora un'istanza di Cittadino venga rimossa, anche le relative istanze di StoricoOperazione cessano di esistere. Infine, vi è una relazione di aggregazione tra Cittadino e Premio: anche in questo caso il cittadino mantiene una collezione di premi, ma a differenza della composizione, gli oggetti Premio possono esistere indipendentemente dal cittadino che li ha ricevuti.

### 2.3.1 Classe Utente

Questa classe astratta ha il compito principale di gestire il processo di autenticazione dell'utente, sia esso un cittadino o un amministratore. I suoi attributi comprendono le informazioni necessarie all'identificazione e alla gestione dell'account associato al profilo utente cittadino.

| **Operazione** | **Descrizione** |
|---|---|
| autentica(String email, String password) | Verifica che le credenziali fornite corrispondano a quelle dell'utente e, in caso positivo, lo contrassegna come autenticato. Riguarda qualsiasi tipo di utente, sia cittadino che amministratore. |

### 2.3.2 Classe Cittadino

Questa classe estende la classe astratta Utente, specializzandosi con funzionalità specifiche legate alla gestione del profilo personale. In particolare, consente l'inserimento e la modifica dei dati relativi al proprio account, nonché il riscatto dei premi disponibili. Gli attributi associati alla classe rappresentano pertanto le informazioni identificative e gestionali del profilo utente.

| **Operazione** | **Descrizione** |
|---|---|
| aggiungiModificaDati (String numeroPatente, String codicePOD, String codiceAbbonamento) | Permette al cittadino di inserire o aggiornare i dati relativi a: numero patente, codice POD e codice abbonamento |
| riscattaPremio(Premio premio) | Consente al cittadino di riscuotere il premio specificato |

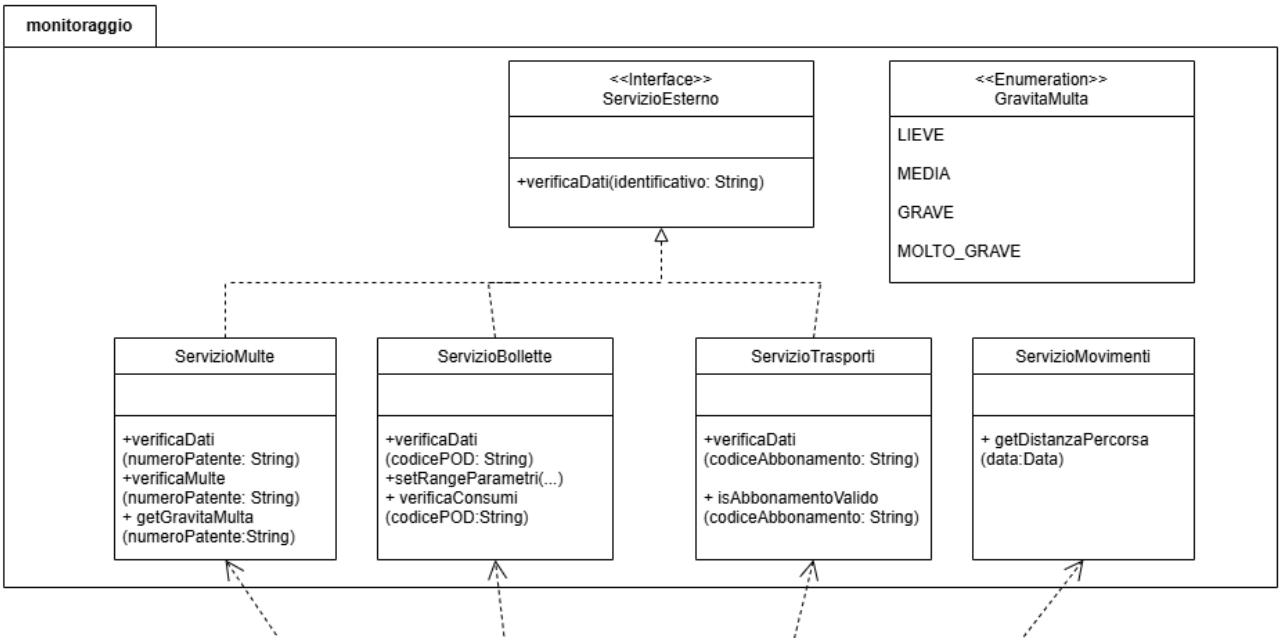
### 2.3.3 Classe Amministratore

Questa classe estende la classe astratta Utente, specializzandosi con funzionalità dedicate alla gestione del profilo amministratore. In particolare, essa consente la supervisione e l'amministrazione dei CivicCoins, grazie alla possibilità di accedere in modo centralizzato a tutti i profili presenti

nell'applicazione. Gli attributi associati alla classe includono un codice univoco che identifica e consente l'accesso al profilo amministratore.

| **Operazione** | **Descrizione** |
|---|---|
| gestisciCivicCoins(Cittadino cittadino, int punti, String motivazione) | Permette all'amministratore di gestire il saldo Civic Coins del cittadino specificato con azioni come assegnazione, rimozione e azzeramento Civic Coins, solo qualora ci fossero degli errori di sistema. |

## 2.4 Package Monitoraggio




Il package monitoraggio definisce l'interfaccia ServizioEsterno, implementata da diverse classi che rappresentano fonti di dati esterne incaricate di verificare l'effettiva esecuzione di comportamenti virtuosi da parte degli utenti. Tale package è progettato per essere integrato con la classe SistemaCivicCoins (definita nel package gestione), alla quale fornisce funzionalità di aggiornamento e verifica dei dati comportamentali. Ne deriva una relazione funzionale diretta tra il modulo di monitoraggio e gli utenti del sistema.

Le relazioni tra le classi considerate sono principalmente di tipo implementativo, in quanto esse forniscono un'implementazione concreta di servizi esterni definiti attraverso interfacce. È inoltre presente una relazione di dipendenza: il sistema SistemaCivicCoins si avvale di tali servizi per effettuare operazioni di assegnazione o rimozione dei punti, basando il proprio funzionamento sulle funzionalità messe a disposizione dalle classi implementative.

### 2.4.1 Classe ServizioEsterno

Questa interfaccia definisce un metodo implementato in maniera specifica da ogni classe che la realizza, adattando il comportamento alle esigenze del proprio contesto. Il metodo ha la funzione di verificare che i dati ricevuti rispettino determinati parametri prestabiliti, conformemente alla logica del contesto applicativo in cui viene utilizzato.

| **Operazione** | **Descrizione** |
|---|---|
| verificaDati(String identificativo) | Verifica l'esistenza di un identificativo in base alla classe Servizio che lo implementa |

### 2.4.2 Classe ServizioMulte

Questa classe ha lo scopo di fornire il controllo delle infrazioni. In particolare fornisce l'implementazione del metodo verificaDati, il quale ha lo scopo di accertare l'esistenza del numero di patente fornito. Inoltre, il metodo verificaMulte esegue una verifica periodica per accertare se il cittadino abbia ricevuto delle multe; in caso affermativo, viene determinata la gravità della multa mediante l'utilizzo del metodo getGravitaMulta.

| **Operazione** | **Descrizione** |
|---|---|
| verificaDati(String numeroPatente) | Verifica l'esistenza del numero di patente |
| getGravitaMulta(String numeroPatente) | Restituisce la gravità della multa associata al numero di patente usando la classe enumeratore GravitaMulta |
| verificaMulte(String numeroPatente) | Verifica se ci sono multe associate al numero di patente dato |

### 2.4.3 Classe ServizioBollette

Questa classe si occupa dell'analisi dei consumi domestici. In particolare, fornisce l'implementazione del metodo verificaDati, il quale ha lo scopo di verificare l'esistenza del codice POD fornito. Il metodo setRangeParametri consente di definire i parametri di riferimento per stabilire il range di consumi "green", entro il quale il cittadino deve rimanere per ottenere punti. Infine, il metodo verificaConsumi esegue una verifica concreta per assicurarsi che i consumi rientrino nei range prestabiliti.

| **Operazione** | **Descrizione** |
|---|---|
| verificaDati(String codicePOD) | Verifica l'esistenza del codice POD. |
| verificaConsumi(String codicePOD) | Controlla se i consumi associati al codice POD rispettano i criteri previsti |
| setRangeParametri(...) | Imposta l'intervallo di parametri dei consumi per cui possono essere assegnati Civic Coins |

### 2.4.4 Classe ServizioTrasporti

Questa classe ha il compito di verificare la validità di un abbonamento. Per tale scopo, utilizza il metodo verificaDati, che si occupa di accertare l'esistenza di tale abbonamento, e successivamente impiega il metodo isAbbonamentoValido per determinare se l'abbonamento sia effettivamente valido.

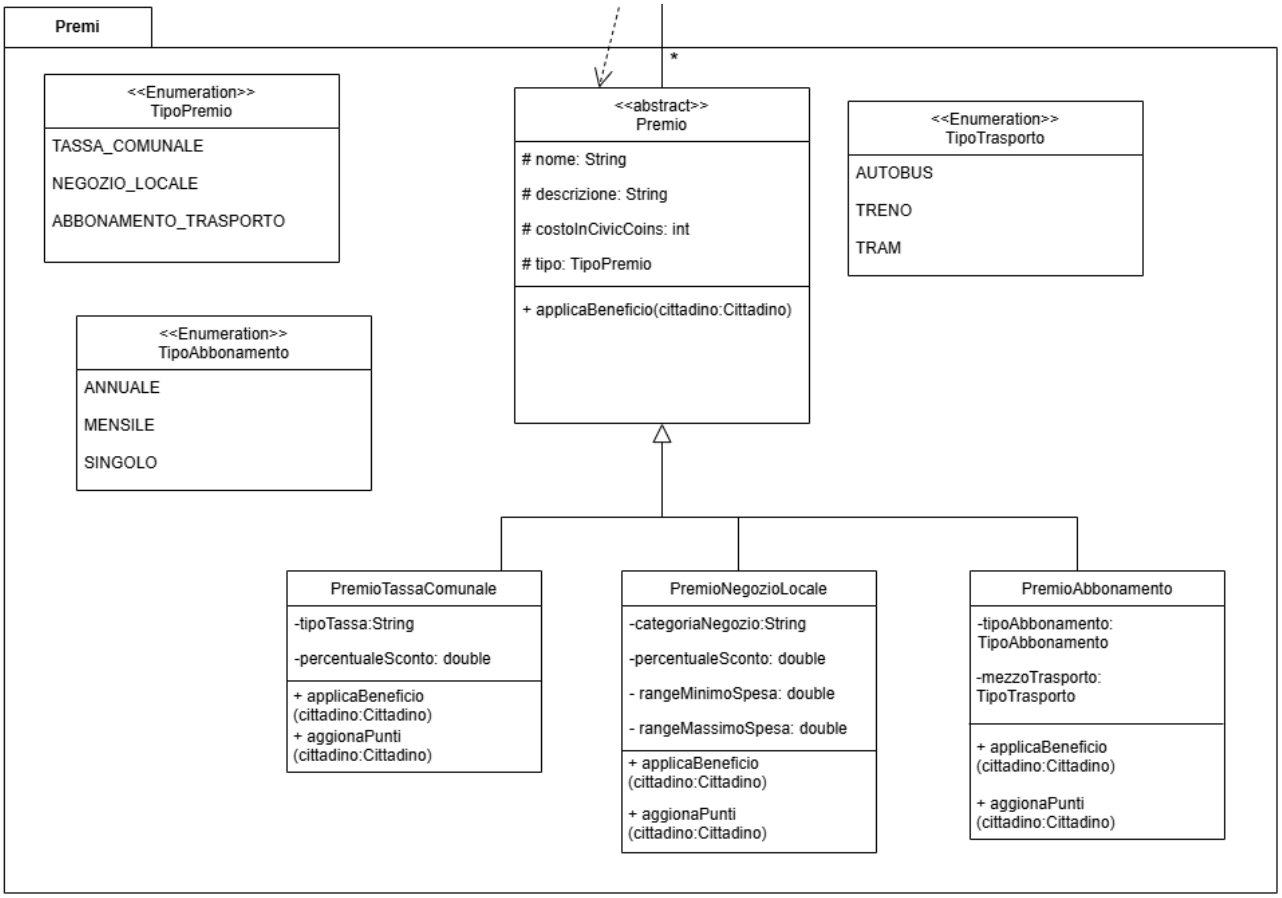
| **Operazione** | **Descrizione** |
|---|---|
| verificaDati(String codiceAbbonamento) | Verifica l'esistenza del codice abbonamento. |
| isAbbonamentoValido(String codiceAbbonamento) | Restituisce *true* se l'abbonamento è ancora valido, *false* altrimenti. |

### 2.4.5 Classe ServizioMovimenti

Questa classe è responsabile del tracciamento degli spostamenti del cittadino. In particolare, tramite l'uso del metodo getDistanzaPercorsa, è in grado di determinare la distanza percorsa a piedi o in bicicletta.

| **Operazione** | **Descrizione** |
|---|---|
| getDistanzaPercorsa(Data data) | Restituisce la distanza totale (in bici o a piedi) percorsa in un certo giorno. |

## 2.5 Package Premi



Il package premi definisce la gerarchia delle classi relative alle agevolazioni che possono essere riscosse dall'utente e con le quali interagiscono. Esso comprende la classe astratta Premio da cui derivano le classi concrete che descrivono le diverse categorie di possibili premi riscuotibili: PremioTassaComunale, PremioNegozioLocale e PremioAbbonamento, tali classi rappresentano le uniche tipologie di premi previste dal sistema. La scelta di rendere Premio una classe astratta è motivata dal fatto che un'istanza generica di premio non ha significato applicativo: ogni premio deve necessariamente assumere il ruolo specifico di PremioTassaComunale o PremioNegozioLocale o PremioAbbonamento.

Le relazioni tra le classi all'interno del package premi si articolano secondo diverse tipologie. La prima tipologia è la generalizzazione, in quanto le classi figlie PremioTassaComunale, PremioNegozioLocale e Premio Abbonamento ereditano dalla classe padre Premio. L'altra relazione è una aggregazione, poiché la classe Cittadino può avere uno o più Premi. Infine è inoltre presente una relazione di dipendenza: il sistema SistemaCivicCoins si avvale del servizio Premio per far riscattare all'utente il beneficio selezionato per poi decurtare i punti di questa operazione dal montante di CivicCoins.

### 2.5.1 Classe Premio

Questa classe astratta ha il compito principale di gestire l'applicazione dei benefici riscossi dall'utente, che possono consistere in premi per ottenere sconti su tasse comunali, buoni per negozi locali o abbonamenti per i mezzi pubblici. Gli attributi della classe comprendono le informazioni necessarie per l'identificazione e la gestione di ciascun premio.

| **Operazione** | **Descrizione** |
|---|---|
| applicaBeneficio(Cittadino cittadino) | Applica un beneficio al cittadino in base alla classe Premio che lo implementa |

### 2.5.2 Classe PremioTassaComunale

Questa classe estende la classe astratta Premio, specializzandosi con metodi specifici legati alla gestione del beneficio selezionato per le tasse comunali e per l'aggiornamento dei punti dell'utente qualora avesse riscosso un premio. Gli attributi associati alla classe rappresentano pertanto le informazioni identificative dell'agevolazione sulle tasse comunali ovvero il tipo di tassa e la percentuale di sconto.

| **Operazione** | **Descrizione** |
|---|---|
| applicaBeneficio(Cittadino cittadino) | Applica uno sconto in percentuale sulla tassa comunale. |

### 2.5.3 Classe PremioNegozioLocale

Questa classe estende la classe astratta Premio specializzandosi con metodi specifici legati alla gestione del beneficio selezionato per i negozi locali e per l'aggiornamento dei punti dell'utente qualora avesse riscosso un premio. Gli attributi associati alla classe rappresentano pertanto le informazioni identificative dei buoni sconto per i negozi locali ovvero la categoria del negozio, la percentuale di sconto, il range minimo della spesa e il range massimo della spesa.

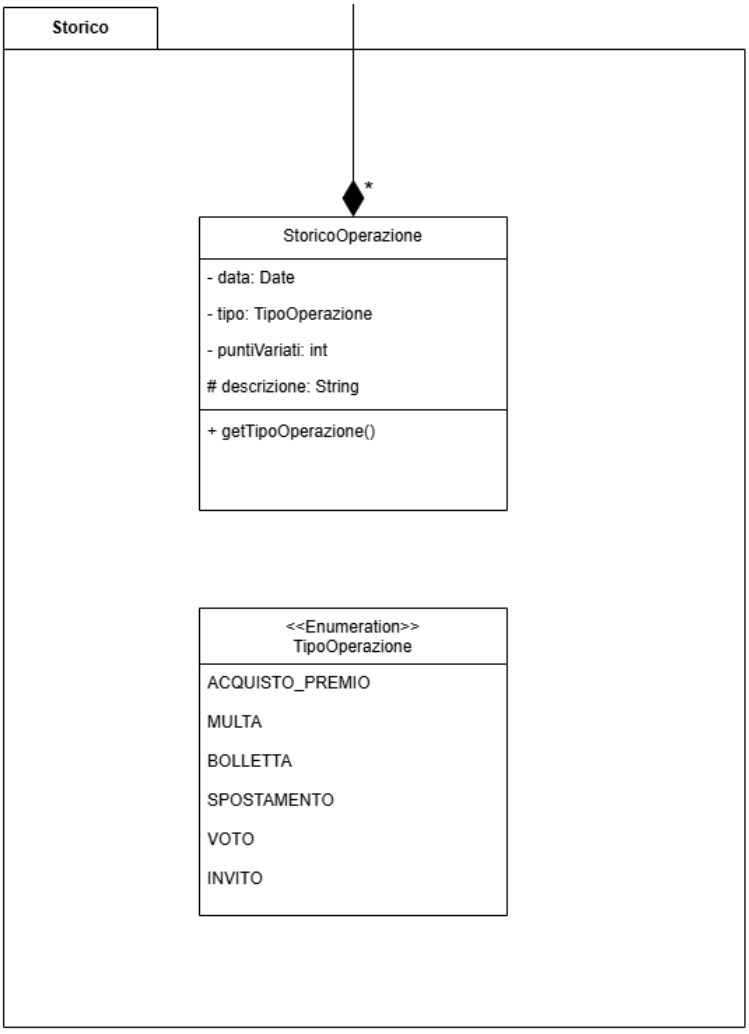
| **Operazione** | **Descrizione** |
|---|---|
| applicaBeneficio(Cittadino cittadino) | Applica uno sconto in percentuale sugli acquisti in un negozio locale di una certa categoria. |

### 2.5.4 Classe PremioAbbonamento

Questa classe estende la classe astratta Premio specializzandosi con metodi specifici legati alla gestione del beneficio selezionato per riscattare un abbonamento per i mezzi pubblici e per l'aggiornamento dei punti dell'utente qualora avesse riscosso un premio. Gli attributi associati alla classe rappresentano pertanto le informazioni identificative all'abbonamento per i mezzi di trasporto.

| **Operazione** | **Descrizione** |
|---|---|
| applicaBeneficio(Cittadino cittadino) | Concede un abbonamento agevolato per il trasporto pubblico locale. |

## 2.6 Package Storico




Il package storico registra tutte le operazioni effettuate, ovvero tutti quei movimenti che comportano un incremento o una decurtazione del saldo di Civic Coins.

La relazione tra le classi esterne al package storico e la classe Cittadino è di tipo **composizione**, in quanto il Cittadino possiede uno storico; qualora il Cittadino venga eliminato, anche lo storico ad esso associato verrà rimosso.

### 2.6.1 Classe StoricoOperazione

Questa classe tiene traccia di azioni come multe, bollette, spostamenti, acquisti premi con il metodo getTipoOperazione. Gli attributi associati alla classe rappresentano pertanto le informazioni identificative alla operazione ovvero data, tipo, punti variati e descrizione.

| **Operazione** | **Descrizione** |
|---|---|
| getTipoOperazione() | Restituisce il tipo di operazione eseguita rappresentato da un valore dell'enumerazione TipoOperazione. Serve per consultare lo storico e capire che operazione ha comportato una modifica sul montante di Civic Coins. |

# 3 Constraint OCL

## 3.1 Classe Utente

| **Descrizione:** Dopo aver eseguito il login, l'attributo autenticato viene settato a *true* e gli attributi email e password popolati |
|---|
| context Utente::autentica(email, password)<br>post: self.autenticato = true AND self.email = email AND self.password = password |

| **Descrizione:** Dopo aver eseguito il logout l'attributo autenticato viene settato a *false* |
|---|
| context Utente::autentica(email,password)<br>post: self.autenticato = *false* |

| **Descrizione:** La password di un utente deve contenere almeno 8 caratteri, di cui almeno una lettera maiuscola, una lettera minuscola, un numero, e un carattere speciale |
|---|
| context Utente::autentica(email,password)<br>pre: password.size >= 8 AND<br>password.substring(1, password.size())->exists(c \| c >= 'A' and c <= 'Z') AND<br>password.substring(1, password.size())->exists(c \| c >= 'a' and c <= 'z') AND<br>password.substring(1, password.size())->exists(c \| c >= '0' and c <= '9') AND<br>password.substring(1, password.size())->exists(c \| c = '@' or c = '#' or c = '$' or c = '%') AND<br>NOT password.substring(1, password.size())->exists(c \| c = ' ') |

| **Descrizione:** Un utente può avere un solo ruolo alla volta (Cittadino o Amministratore) |
|---|
| context Utente<br>inv: self.oclIsTypeOf(Cittadino) XOR self.oclIsTypeOf(Amministratore) |

| **Descrizione:** Un utente deve avere nome, cognome, email, password e codice fiscale non nulli |
|---|
| context Utente<br>inv: NOT nome.oclIsUndefined() AND NOT cognome.oclIsUndefined() AND<br>NOT email.oclIsUndefined() AND NOT password.oclIsUndefined() AND<br>NOT codiceFiscale.oclIsUndefined() |

## 3.2 Classe SistemaAutenticazione

| **Descrizione:** L'email e la password devono essere non nulli quando vengono verificate le credenziali, e la password deve contenere almeno 8 caratteri, di cui almeno una lettera maiuscola, una lettera minuscola, un numero, e un carattere speciale |
|---|
| context SistemaAutenticazione::verificaCredenziali(email, password)<br>pre: NOT email.isEmpty() AND NOT password.isEmpty()<br>password.size >= 8 AND<br>password.substring(1, password.size())->exists(c \| c >= 'A' and c <= 'Z') AND<br>password.substring(1, password.size())->exists(c \| c >= 'a' and c <= 'z') AND<br>password.substring(1, password.size())->exists(c \| c >= '0' and c <= '9') AND<br>password.substring(1, password.size())->exists(c \| c = '@' or c = '#' or c = '$' or c = '%') AND<br>NOT password.substring(1, password.size())->exists(c \| c = ' ') |

| **Descrizione:** Quando la password viene reimpostata deve avere almeno 8 caratteri, di cui almeno una lettera maiuscola, una lettera minuscola, un numero, e un carattere speciale |
|---|
| context SistemaAutenticazione::reimpostaPassword(token, nuovaPassword)<br>password.size >= 8 AND<br>password.substring(1, password.size())->exists(c \| c >= 'A' and c <= 'Z') AND<br>password.substring(1, password.size())->exists(c \| c >= 'a' and c <= 'z') AND<br>password.substring(1, password.size())->exists(c \| c >= '0' and c <= '9') AND<br>password.substring(1, password.size())->exists(c \| c = '@' or c = '#' or c = '$' or c = '%') AND<br>NOT password.substring(1, password.size())->exists(c \| c = ' ') |

| **Descrizione:** Dopo la registrazione, il cittadino ottiene 800 Civic Coins iniziali |
|---|
| context SistemaAutenticazione::registraCittadino(cittadino, email, password)<br>post: result.civicCoins = 800 |

## 3.3 Classe StoricoOperazione

| **Descrizione:** Se il tipo di operazione è un acquisto di un premio, i punti variati dovranno avere un valore non nullo |
|---|
| context StoricoOperazione<br>inv: tipo = TipoOperazione::ACQUISTO-PREMIO implies puntiVariati > 0 |

| **Descrizione:** La data dell'operazione non può essere futura |
|---|
| context StoricoOperazione<br>inv: date <= Date::now() |

## 3.4 Classe SistemaCivicCoins

| **Descrizione:** I punti assegnati devono essere maggiori di zero |
| --- |
| context SistemaCivicCoins::assegnaPunti(cittadino,punti, motivazione)<br>pre: punti $>$ 0 |

| **Descrizione:** I punti rimossi devono essere maggiori di zero e minori dei Civic Coins totali del cittadino |
| --- |
| context SistemaCivicCoins::rimuoviPunti(cittadino,punti, motivazione)<br>pre: punti $>$ 0 AND punti $<=$ cittadino.civicCoins |