Progetto di Ingegneria del Software

UNIVERSITÀ DEGLI STUDI DI TRENTO

Dipartimento di Ingegneria e Scienza dell'Informazione

Corso di Laurea:
Ingegneria Informatica, delle Comunicazioni ed Elettronica

EcoTrack

Gruppo 1

Giulio Gualtiero	Jago Revrenna	Tommaso Onori
234656	235081	234893

Anno Accademico 2024/2025

Indice

1 Diagramma Componenti

1.1 Diagramma dei Componenti

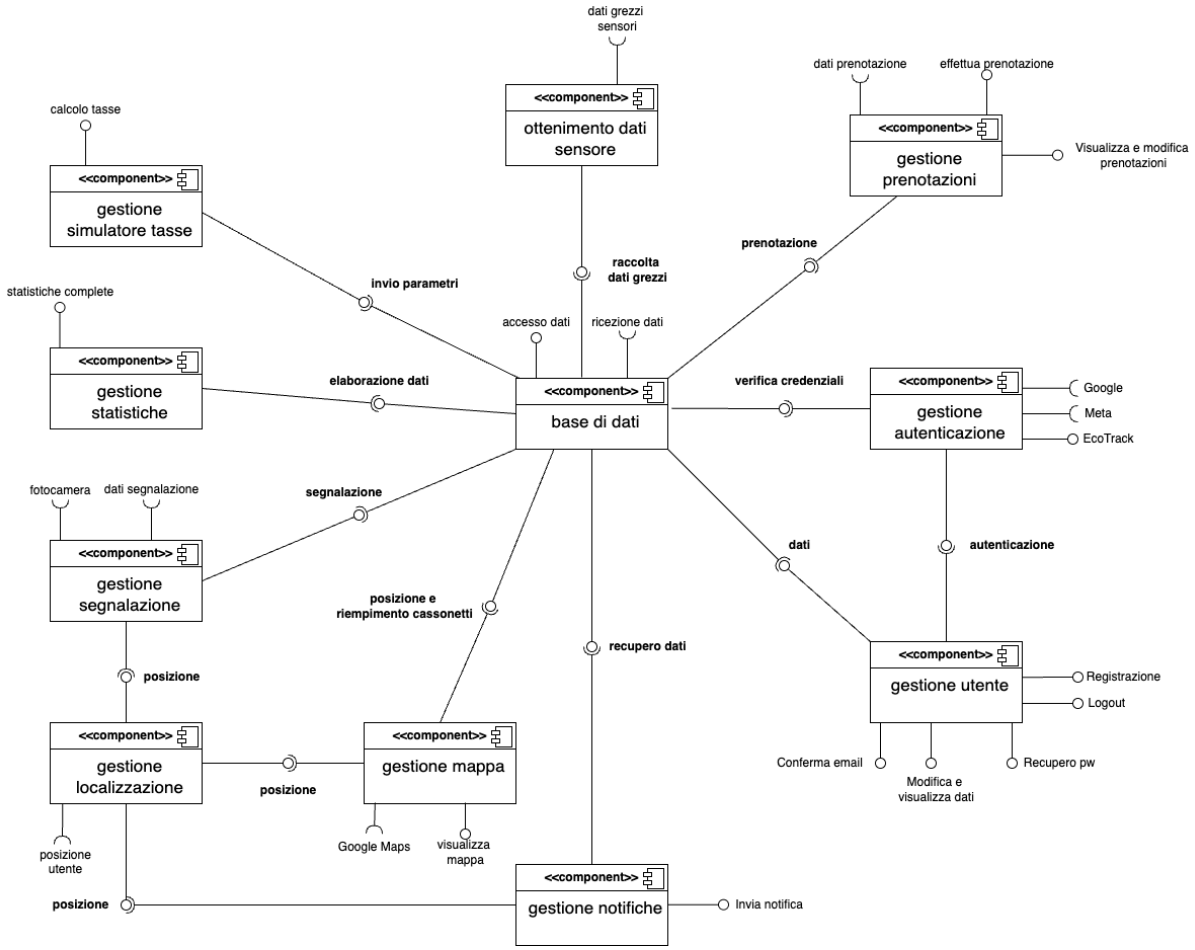

Figura 1: Diagramma dei componenti

1.2 Descrizione dei Componenti

Gestione Autenticazione Il componente gestisce l'autenticazione dell'utente, la quale può essere effettuata in diversi modi.

Tipologia	**Nome**	**Descrizione**
Richiesta	Google	Il componente utilizza un'interfaccia fornita da Google per autenticarsi tramite un token (SSO)
Richiesta	Meta	Il componente utilizza un'interfaccia fornita da Meta per autenticarsi tramite un token (SSO)
Fornita	EcoTrack	Il componente fornisce un'interfaccia per autenticarsi mediante gli account locali
Fornita	Autenticazione	Il componente fornisce un'interfaccia per autenticarsi
Richiesta	Verifica credenziali	Il componente richiede che le credenziali vengano verificate

Tabella 1: Componenti coinvolti nella gestione dell'autenticazione

Gestione localizzazione Questo componente gestisce le interazioni con il sistema di localizzazione (GPS).

Tipologia	Nome	Descrizione
Fornita	Posizione	Il componente fornisce (in coordinate) la posizione attuale dell'utente
Richiesta	Posizione utente	Il componente richiede la posizione attuale dell'utente

Tabella 2: Componenti coinvolti nella gestione della localizzazione

Base di dati Questo componente gestisce tutti i dati coinvolti nel funzionamento dell'applicazione e permette agli altri componenti di relazionarsi.

Tipologia	Nome	Descrizione
Fornito	Verifica credenziali	Il componente effettua la verifica delle credenziali utilizzate
Fornito	Prenotazione	Il componente verifica gli orari disponibili e salva una prenotazione
Richiesto	Raccolta dati grezzi	Il componente raccoglie i dati grezzi direttamente dai sensori
Richiesto	Invio parametri	Il componente salva i parametri relativi al simulatore tasse
Fornito	Elaborazione dati	Il componente fornisce i dati grezzi, che devono essere elaborati. In seguito, i dati elaborati verranno inviati al componente in questione.
Richiesto	Segnalazione	Il componente salva un'eventuale segnalazione e mostra le segnalazioni già salvate
Fornito	Posizione e riempimento cassonetti	Il componente fornisce la posizione (coordinate) e il livello di riempimento dei cassonetti
Fornito	Recupero dati	Il componente fornisce i dati che potrebbero far scaturire una notifica
Richiesto	Dati	Il componente richiede e mostra i dati relativi ai vari utenti

Tabella 3: Componenti coinvolti nella gestione della base di dati

2 Diagramma e Descrizione delle Classi

2.1 Diagramma delle Classi

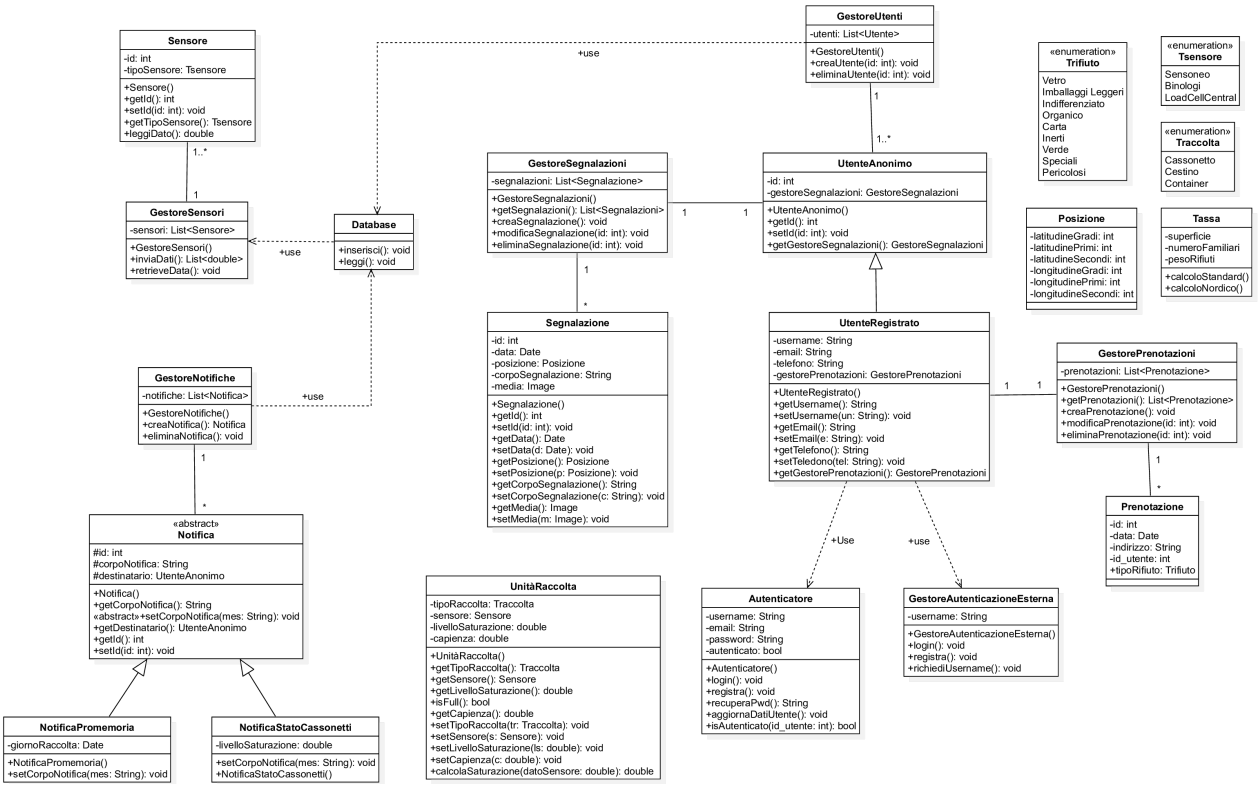

Figura 2: Diagramma delle Classi

2.2 Descrizione delle Classi

Classi del modulo utente e autenticazione

UtenteAnonimo

La classe **UtenteAnonimo** rappresenta un utente non autenticato, che può comunque interagire con il sistema effettuando segnalazioni riguardanti problematiche o situazioni relative alla gestione dei rifiuti. Questa classe è pensata per garantire accesso a funzionalità base anche senza richiedere dati personali, mantenendo però una certa tracciabilità tramite un identificativo numerico.

Metodo	Descrizione
`UtenteAnonimo()`	Costruttore della classe, inizializza un nuovo utente anonimo con un gestore dedicato alle segnalazioni.
`getId()`	Restituisce l'identificativo numerico associato all'utente anonimo.
`setId(int id)`	Permette di impostare o aggiornare l'identificativo dell'utente.
`getGestoreSegnalazioni()`	Restituisce il riferimento all'oggetto incaricato di gestire le segnalazioni dell'utente.

Tabella 4: Metodi della classe `UtenteAnonimo`

UtenteRegistrato

La classe **UtenteRegistrato** estende le funzionalità dell'utente anonimo offrendo servizi avanzati, tra cui la possibilità di gestire le prenotazioni per il ritiro dei rifiuti. L'utente fornisce informazioni personali al momento della registrazione e ha accesso a funzionalità aggiuntive.

Metodo	Descrizione
`UtenteRegistrato()`	Costruttore della classe. Inizializza un utente registrato e associa un gestore delle prenotazioni.
`getUsername()`	Restituisce il nome utente con cui l'utente è registrato nel sistema.
`setUsername(String un)`	Aggiorna lo username dell'utente.
`getEmail()`	Restituisce l'email associata all'account dell'utente.
`setEmail(String e)`	Permette di modificare l'indirizzo email.
`getTelefono()`	Restituisce il numero di telefono dell'utente.
`setTelefono(String tel)`	Aggiorna il numero di telefono.
`getGestorePrenotazioni()`	Restituisce l'oggetto incaricato della gestione delle prenotazioni per il ritiro dei rifiuti.

Tabella 5: Metodi della classe `UtenteRegistrato`

Autenticatore

La classe **Autenticatore** si occupa della gestione dell'accesso al sistema da parte degli utenti. Fornisce metodi per l'autenticazione tramite credenziali, per la registrazione di nuovi account, per il recupero della password e per la verifica dello stato di autenticazione. Gestisce inoltre l'aggiornamento delle informazioni utente.

Metodo	Descrizione
`Autenticatore()`	Costruttore della classe. Inizializza un nuovo oggetto autenticatore.
`login()`	Verifica le credenziali fornite e autentica l'utente se valide.
`registra()`	Consente la registrazione di un nuovo utente nel sistema.
`recuperaPwd()`	Avvia la procedura di recupero password restituendo un link o codice temporaneo.
`aggiornaDatiUtente()`	Permette di aggiornare le informazioni personali dell'utente registrato.
`isAutenticato(int id_utente)`	Verifica se l'utente con l'ID specificato ha completato l'autenticazione.

Tabella 6: Metodi della classe `Autenticatore`

GestoreAutenticazioneEsterna

La classe **GestoreAutenticazioneEsterna** gestisce l'autenticazione attraverso provider esterni, come ad esempio Google o Facebook. Permette all'utente di accedere al sistema senza dover creare un account locale, semplificando il processo di login e registrazione.

Metodo	Descrizione
`GestoreAutenticazioneEsterna()`	Costruttore della classe. Inizializza il gestore per l'interazione con provider esterni.
`login()`	legge le credenziali immesse dall'utente e chiama l'API del provider esterno. In caso di successo aggiorna i valori degli attributi della classe Utente (username e ID)
`registra()`	legge le credenziali immesse dall'utente e chiama l'API del provider esterno. In caso di successo salva del Database i dati relativi all'utente
`richiediUsername()`	richiede l'immissione di uno username in fase di registrazione tramite provider esterno e lo salva nell'attributo username

Tabella 7: Metodi della classe `GestoreAutenticazioneEsterna`

Classi per la gestione delle segnalazioni e degli utenti

GestoreUtenti

La classe **GestoreUtenti** rappresenta il componente incaricato della gestione centralizzata di tutti gli utenti della piattaforma, siano essi anonimi o registrati. Il gestore mantiene una collezione interna di utenti e fornisce operazioni per la loro creazione e rimozione. Questa classe garantisce coerenza e tracciabilità degli utenti attivi nel sistema, relazionandosi con il database così da poterla mantenere aggiornata.

Metodo	Descrizione
`GestoreUtenti()`	Costruttore della classe. Inizializza la struttura dati per contenere l'insieme degli utenti gestiti.
`creaUtente(int id)`	Aggiunge un nuovo utente al sistema, utilizzando l'identificativo fornito.
`eliminaUtente(int id)`	Rimuove l'utente corrispondente all'identificativo specificato, liberando eventuali risorse.

Tabella 8: Metodi della classe `GestoreUtenti`

GestoreSegnalazioni

La classe **GestoreSegnalazioni** è responsabile della gestione delle segnalazioni inviate dagli utenti. Ogni istanza di questa classe gestisce un insieme di segnalazioni, offrendo funzionalità di visualizzazione, creazione, modifica e cancellazione (CRUD). È pensata per essere utilizzata da un ciascun utente, in modo da mantenere separate le segnalazioni.

Metodo	Descrizione
`GestoreSegnalazioni()`	Costruttore della classe. Inizializza la lista delle segnalazioni da gestire.
`getSegnalazioni()`	Restituisce l'elenco completo delle segnalazioni presenti per l'utente corrente.
`creaSegnalazione()`	Permette la creazione di una nuova segnalazione, che viene aggiunta alla lista.
`modificaSegnalazione(int id)`	Modifica una segnalazione esistente individuata dal suo identificativo, aggiornando le sue informazioni.
`eliminaSegnalazione(int id)`	Elimina la segnalazione identificata dall'id specificato dalla collezione interna.

Tabella 9: Metodi della classe `GestoreSegnalazioni`

Segnalazione

La classe **Segnalazione** rappresenta un'istanza concreta di segnalazione effettuata da un utente. Ogni segnalazione contiene una serie di informazioni quali data, posizione geografica, testo descrittivo e, opzionalmente, un elemento multimediale (foto o video). Questa classe incapsula i dettagli dell'evento segnalato.

Metodo	**Descrizione**
`Segnalazione()`	Costruttore della classe. Inizializza una nuova segnalazione vuota o con valori di default.
`getId()`	Restituisce l'identificativo univoco della segnalazione.
`setId(int id)`	Imposta o aggiorna l'identificativo della segnalazione.
`getData()`	Restituisce la data in cui la segnalazione è stata effettuata.
`setData(Data d)`	Imposta o aggiorna la data della segnalazione.
`getPosizione()`	Restituisce la posizione geografica associata alla segnalazione.
`setPosizione(Posizione p)`	Imposta o aggiorna la posizione della segnalazione.
`getCorpoSegnalazione()`	Restituisce il contenuto testuale descrittivo della segnalazione.
`setCorpoSegnalazione(String c)`	Imposta o aggiorna il testo descrittivo della segnalazione.
`getMedia()`	Restituisce l'eventuale contenuto multimediale associato alla segnalazione.
`setMedia(Media m)`	Permette di allegare o aggiornare il supporto multimediale.

Tabella 10: Metodi della classe `Segnalazione`

GestorePrenotazioni

La classe **GestorePrenotazioni** rappresenta il componente incaricato della gestione delle prenotazioni per il conferimento dei rifiuti presso un ecocentro. Ogni utente registrato dispone di un proprio gestore delle prenotazioni, che permette di accedere, creare, modificare e cancellare le prenotazioni.

Metodo	**Descrizione**
`GestorePrenotazioni()`	Costruttore della classe. Inizializza la lista interna delle prenotazioni associate all'utente.
`getPrenotazioni()`	Restituisce la lista completa delle prenotazioni effettuate dall'utente.
`creaPrenotazione()`	Permette la creazione di una nuova prenotazione, che viene salvata nella lista e associata all'utente registrato.
`modificaPrenotazione(int id)`	Consente di modificare una prenotazione esistente identificata tramite il suo ID, aggiornandone i dettagli (data, indirizzo, tipo di rifiuto).
`eliminaPrenotazione(int id)`	Rimuove una prenotazione dalla lista interna, in base all'ID specificato.

Tabella 11: Metodi della classe `GestorePrenotazioni`

Prenotazione

La classe **Prenotazione** rappresenta una richiesta di conferimento rifiuti, effettuata da un utente registrato. La classe incapsula tutte le informazioni necessarie alla gestione operativa del servizio.

Metodo	Descrizione
`Prenotazione()`	Costruttore della classe. Inizializza una nuova istanza di prenotazione con valori predefiniti o nulli.
`getId()`	Restituisce l'identificativo univoco della prenotazione.
`setId(int id)`	Imposta o aggiorna l'identificativo della prenotazione.
`getData()`	Restituisce la data prevista per il ritiro dei rifiuti.
`setData(Data d)`	Imposta o aggiorna la data della prenotazione.
`getIndirizzo()`	Restituisce l'indirizzo dell'ecocentro presso il quale conferire i rifiuti
`setIndirizzo(String i)`	Imposta o aggiorna l'indirizzo dell'ecocentro.
`getIdUtente()`	Restituisce l'identificativo dell'utente che ha richiesto la prenotazione.
`setIdUtente(int id)`	Imposta o aggiorna l'ID dell'utente associato alla prenotazione.
`getTipoRifiuto()`	Restituisce il tipo di rifiuto selezionato per il ritiro.
`setTipoRifiuto(Trifiuto t)`	Imposta o aggiorna il tipo di rifiuto oggetto della prenotazione.

Tabella 12: Metodi della classe `Prenotazione`

UnitàRaccolta

La classe **UnitàRaccolta** rappresenta una singola unità fisica di raccolta rifiuti. Ogni unità è caratterizzata da una tipologia di raccolta, da un sensore che rileva il livello di riempimento e da proprietà come la capienza massima e il livello attuale di saturazione.

Metodo	Descrizione
`UnitàRaccolta()`	Costruttore della classe. Inizializza una nuova unità di raccolta con valori predefiniti o nulli.
`getTipoRaccolta()`	Restituisce il tipo di raccolta associato all'unità (es. vetro, carta, imballaggi leggeri).
`setTipoRaccolta(Traccolta tr)`	Imposta o aggiorna il tipo di raccolta dell'unità.
`getSensore()`	Restituisce il sensore attualmente collegato all'unità.
`setSensore(Sensore s)`	Imposta o aggiorna il sensore utilizzato per monitorare la saturazione.
`getLivelloSaturazione()`	Restituisce il livello attuale di riempimento dell'unità, espresso come valore percentuale o frazionario.
`setLivelloSaturazione(double ls)`	Imposta o aggiorna il valore corrente del livello di riempimento.
`getCapienza()`	Restituisce la capienza massima dell'unità.
`setCapienza(double c)`	Imposta o aggiorna il valore della capienza massima dell'unità.
`isFull()`	Restituisce un valore booleano che indica se l'unità è completamente piena o ha superato una soglia di saturazione critica.
`calcolaSaturazione(double datoSensore)`	Calcola il nuovo livello di saturazione a partire da una lettura fornita dal sensore, aggiornando lo stato interno.

Tabella 13: Metodi della classe `UnitàRaccolta`

3 Vincoli Object Constraint Language

In questa sezione vengono specificati i vincoli espressi in linguaggio OCL (Object Constraint Language). Si includono invarianti, precondizioni e postcondizioni relative alle operazioni principali delle classi. I vincoli permettono di garantire la consistenza logica del sistema.

3.1 Classe Autenticatore

Precondizione per `login()`: L'username e la password non devono essere nulli o vuoti.

```
context Autenticatore::login(username: String, password: String): Boolean
pre: not username.oclIsUndefined() and username <> '' and
     not password.oclIsUndefined() and password <> ''
```

Postcondizione per `login()`: Se le credenziali sono corrette, l'utente risulta autenticato.

```
context Autenticatore::login(username: String, password: String): Boolean
post: result = true implies self.isAutenticato(self.idUtente) = true
```

Invariante: Un utente autenticato deve avere un ID valido (positivo).

```
context Autenticatore
inv: self.isAutenticato(self.idUtente) implies self.idUtente > 0
```

3.2 Classe GestoreAutenticazioneEsterna

Postcondizione per `login()`: L'utente viene autenticato solo se il provider conferma le credenziali.

```
context GestoreAutenticazioneEsterna::login(): Boolean
post: result = true implies self.isAutenticato(self.idUtente) = true
```

3.3 Classe UtenteRegistrato

Precondizione per `setTelefono()`: Il numero di telefono deve essere lungo almeno 9 cifre.

```
context UtenteRegistrato::setTelefono(tel: String): void
pre: tel.size() >= 9
```

Postcondizione per `setUsername()`: L'attributo username viene aggiornato.

```
context UtenteRegistrato::setUsername(un: String): void
post: self.getUsername() = un
```

3.4 Classe GestoreSegnalazioni

Invariante: Nessuna segnalazione può avere una data futura.

```
context GestoreSegnalazioni
inv: self.getSegnalazioni()->forAll(s | s.getData() <= Date::now())
```

3.5 Classe Prenotazione

Invariante: La data della prenotazione deve essere futura.

```
context Prenotazione
inv: self.getData() > Date::now()
```

3.6 Classe UnitaRaccolta

Invariante: Il livello di saturazione non può superare 100%.

```
context UnitaRaccolta
inv: self.getLivelloSaturazione() <= 1.0
```

Postcondizione per `calcolaSaturazione()`: Il nuovo livello corrisponde al dato sensore/capienza.

```
context UnitaRaccolta::calcolaSaturazione(datoSensore: Real): void
post: self.getLivelloSaturazione() = datoSensore / self.getCapienza()
```

Invariante: Una unità è critica se supera l'85% di saturazione.

```
context UnitaRaccolta
inv: self.isFull() = (self.getLivelloSaturazione() >= 0.85)
```

3.7 Classe GestoreUtenti

Precondizione per `creaUtente()`: L'ID non deve essere già utilizzato.

```
context GestoreUtenti::creaUtente(id: Integer): void
pre: self.utenti->forAll(u | u.getId() <> id)
```