Ingegneria del Software

Dipartimento di Ingegneria e Scienza dell'Informazione

Gruppo G-51: Giuliano Campagnolo, Elisa Beltrame, Mattia Marini

Mergify - merge your events

Deliverable 3

Diagramma delle classi

Anno accademico 2022/2023

Università di Trento

Indice

Indice

1 Scopo del Documento

Questo documento presenta la struttura del progetto Mergify, utilizzando diagrammi delle classi in Unified Modeling Language (UML) e codice in Object Constraint Language (OCL). Alla luce del documento precedente, che comprende l'use case diagram, il context diagram e il component diagram, viene qui delineata l'architettura del sistema. Questo documento dettaglia le classi che saranno implementate nel codice e descrive la logica che governa il comportamento del software. Le classi sono rappresentate attraverso un class diagram in UML, mentre la logica del sistema viene espressa tramite OCL. Questo approccio viene adottato in quanto alcuni concetti non possono essere adeguatamente formalizzati altrimenti nel contesto di UML.

2 Diagramma delle classi

Ogni elemento identificato nel diagramma dei componenti viene trasformato in una o più classi. Ciascuna classe è caratterizzata da un nome, una serie di attributi che descrivono i dati gestiti dalla classe e una lista di metodi che definiscono le operazioni disponibili. Le classi possono anche essere associate tra loro, fornendo così informazioni sulle relazioni esistenti tra di loro. Durante questo processo, si è prestata particolare attenzione a massimizzare la coesione e a minimizzare l'accoppiamento tra le classi.

2.1 Utente, gruppo e chat

All'interno dell'applicazione vi è definito un solo tipo di utente. I metodi della classe `Utente` permettono di accedere e modificare i suoi attributi. La classe `Utente` rappresenta l'attore che contiene le informazioni riguardo l'utilizzatore. Il metodo `SincronizzaDaGruppo` si occupa di caricare gli eventi proposti nel gruppo nel calendario dello stesso utente, in modo da mantenerlo sincronizzato.
Gli utenti interagiscono tra di loro all'interno dei gruppi. La classe `Gruppo` rappresenta un insieme di utenti che possono interagire tra loro attraverso una chat e condividere eventi. Gli utenti possono essere aggiunti o rimossi dal gruppo, e gli eventi possono essere creati e gestiti all'interno del gruppo. La classe `Gruppo` si occupa di tenere aggiornate le informazioni per i partecipanti attraverso l'utilizzo della chat di gruppo.
La classe `Chat` gestisce la comunicazione tra i membri di un gruppo. La chat contiene una lista di messaggi (`messaggi`). Gli utenti possono inviare nuovi messaggi tramite il metodo `inviaMessaggio()`, e la lista di messaggi può essere ottenuta attraverso il metodo `ottieniMessaggi()`.

Sia la classe `Gruppo` che la classe `Utente` interagiscono con la classe `Calendario`, che si occupa di gestire gli eventi creati. L'ID del calendario si riferisce all'ID dell'utente o del gruppo a cui appartiene; se è null, non appartiene a nessuno.

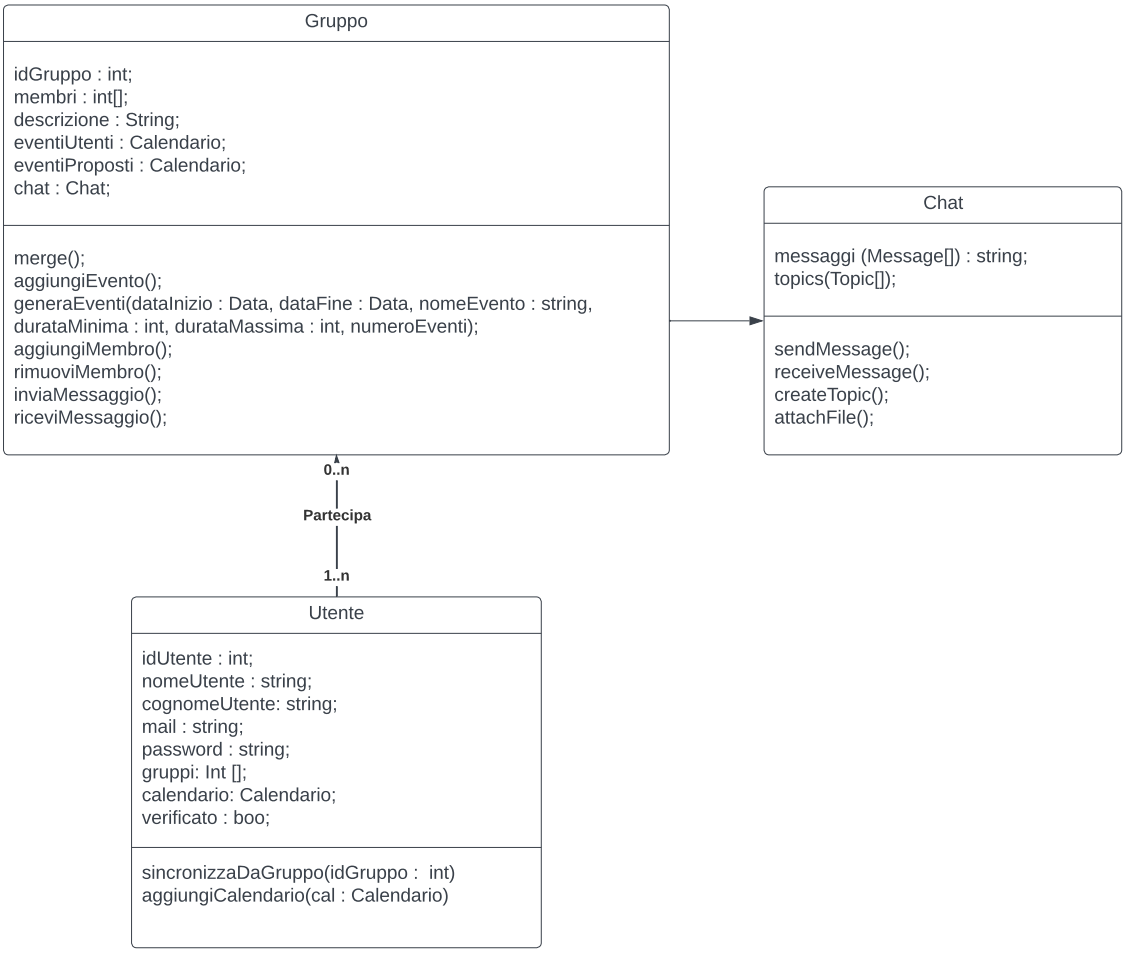

Figura 1: Utente, gruppo e chat

2.2 Gestione Calendario

La funzione calendario di gruppo è una delle classi più importanti dell'applicazione. Essa gestisce gli eventi creati dagli utenti e dai gruppi, permettendo una gestione centralizzata e sincronizzata degli eventi. Ogni calendario è identificato da un identificatore unico (`idCalendario`) e contiene una lista di eventi (`eventi`). Gli utenti e i gruppi possono aggiungere eventi al calendario utilizzando i metodi `aggiungiEvento()`, `rimuoviEvento()`, e `ottieniEventi()`. L'ID del calendario (`idCalendario`) si riferisce all'ID dell'utente o del gruppo a cui appartiene; se è null, il calendario non appartiene a nessuno.

La classe `Evento` è strettamente associata alla classe `Calendario` e definisce gli eventi che possono essere creati all'interno di un gruppo. Ogni evento ha un identificatore unico (`idEvento`), un nome (`nomeEvento`), una descrizione (`descrizione`), e un organizzatore (`organizzatore`). I metodi della classe `Evento`, come `getNomeEvento()`, `setNomeEvento()`, `getDescrizione()`, `setDescrizione()`, `getOrganizzatore()`, e `setOrganizzatore()`, permettono di gestire e modificare i dettagli dell'evento.
La classe `Parser` è una funzionalità aggiuntiva che si occupa di convertire i dati degli eventi tra diversi formati, facilitando l'importazione e l'esportazione delle informazioni. I metodi della classe `Parser`, come `parseEventoDaJson()`, `parseEventoDaXml()`, e `convertiEventoInJson()`, permettono di trasformare i dati degli eventi da e verso formati JSON e XML, garantendo così una maggiore interoperabilità del sistema.

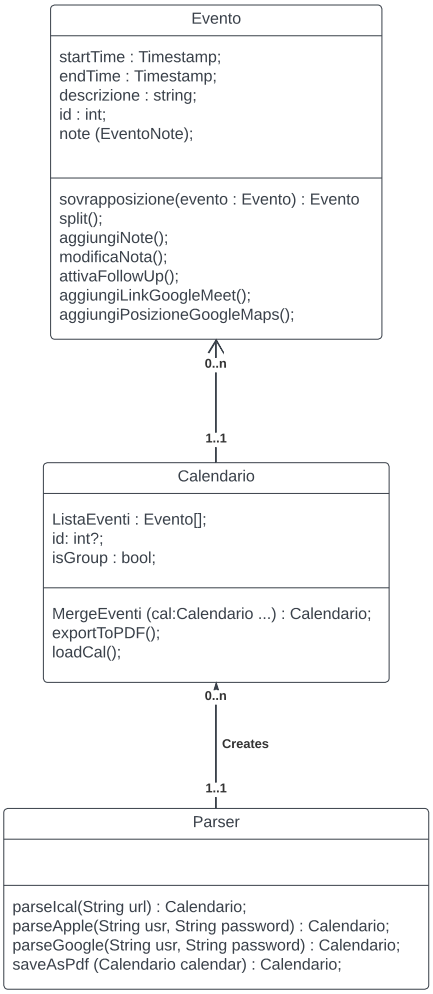

Figura 2: Gestione Calendario

2.3 Gestione Evento

Gli eventi sono ulteriormente dettagliati attraverso l'uso delle classi `Timestamp` e `Data`. La classe `Timestamp` gestisce le informazioni temporali relative a un evento, assicurando

che la data e l'ora siano precise e ben definite. Gli attributi principali includono la data (`data`) e l'ora (`ora`) dell'evento. I metodi `getData()`, `setData()`, `getOra()`, e `setOra()` permettono di gestire queste informazioni in modo accurato. La classe `Data` rappresenta le date associate agli eventi e lavora in stretta collaborazione con la classe `Timestamp`. Gli attributi della classe `Data` includono l'anno (`anno`), il mese (`mese`), e il giorno (`giorno`). I metodi `getAnno()`, `setAnno()`, `getMese()`, `setMese()`, `getGiorno()`, e `setGiorno()` permettono di accedere e modificare i dettagli della data, garantendo che le informazioni temporali siano complete e corrette.

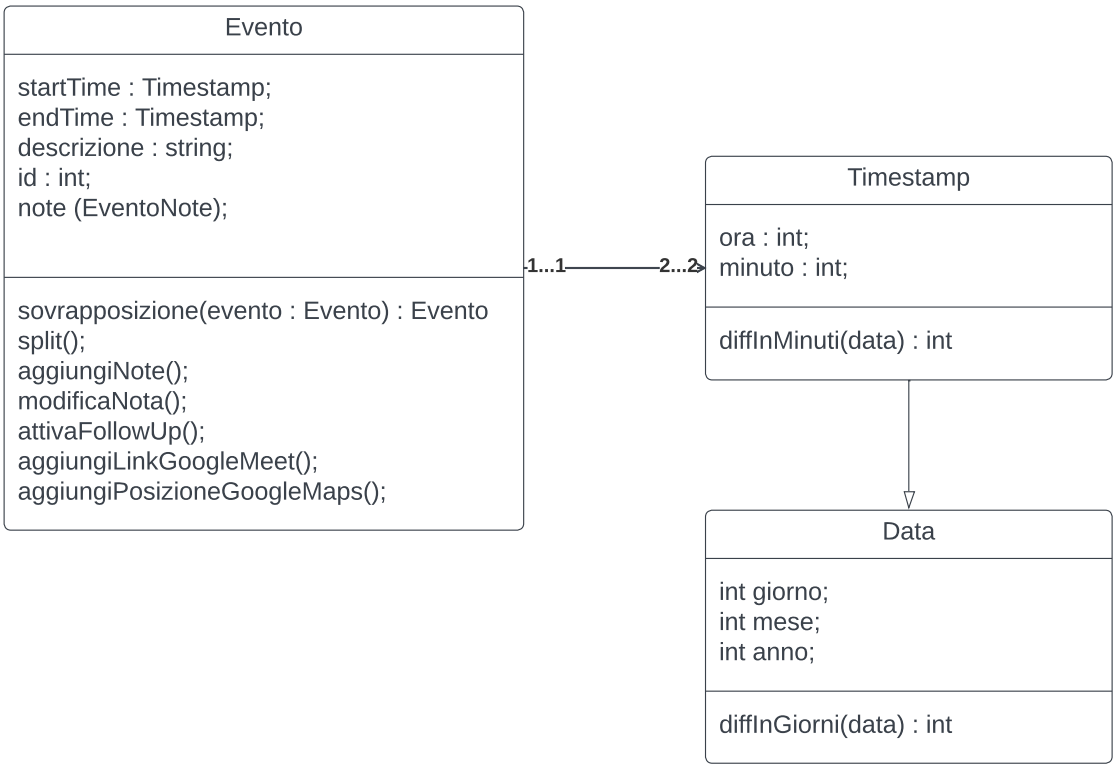

Figura 3: Gestione Evento

2.4 Visualizzatore Calendario

Dall'analisi del diagramma dei componenti emerge chiaramente la necessità di implementare funzioni di visualizzazione per il calendario, come evidenziato nella pagina "Mio Calendario".

Per soddisfare questa esigenza, è stata progettata la classe `VisualizzatoreCalendario`, che gioca un ruolo centrale nella gestione e presentazione del calendario.

La classe `VisualizzatoreCalendario` è responsabile della visualizzazione delle componenti del calendario. Essa mostra in modo chiaro e organizzato gli eventi programmati

e le date. La visualizzazione del calendario è una funzione cruciale, poiché consente agli utenti di avere una panoramica completa dei loro appuntamenti e delle attività future.
Oltre alla visualizzazione del calendario, `VisualizzatoreCalendario` gestisce anche la `BarraStrumenti`, una componente implementata nella classe omonima. La `BarraStrumenti` offre agli utenti diverse opzioni per interagire con il calendario, come aggiungere nuovi eventi, modificare quelli esistenti e accedere ad altre funzioni utili. Questo strumento è essenziale per garantire un'interazione fluida e intuitiva con il calendario.
Un'altra funzione importante di `VisualizzatoreCalendario` è la gestione della `BarraNavigazione`, implementata nella classe `BarraNavigazione`. La `BarraNavigazione` consente agli utenti di spostarsi agevolmente tra i diversi mesi e di selezionare i giorni specifici. Questo aspetto della navigazione è fondamentale per una gestione efficace del calendario, permettendo agli utenti di trovare rapidamente le date e le informazioni desiderate.
In sintesi, la classe `VisualizzatoreCalendario` integra diverse funzionalità per fornire una soluzione completa per la visualizzazione e la gestione del calendario. Essa combina la visualizzazione degli eventi con gli strumenti di navigazione e gestione, offrendo così un'esperienza utente coerente e ben strutturata.

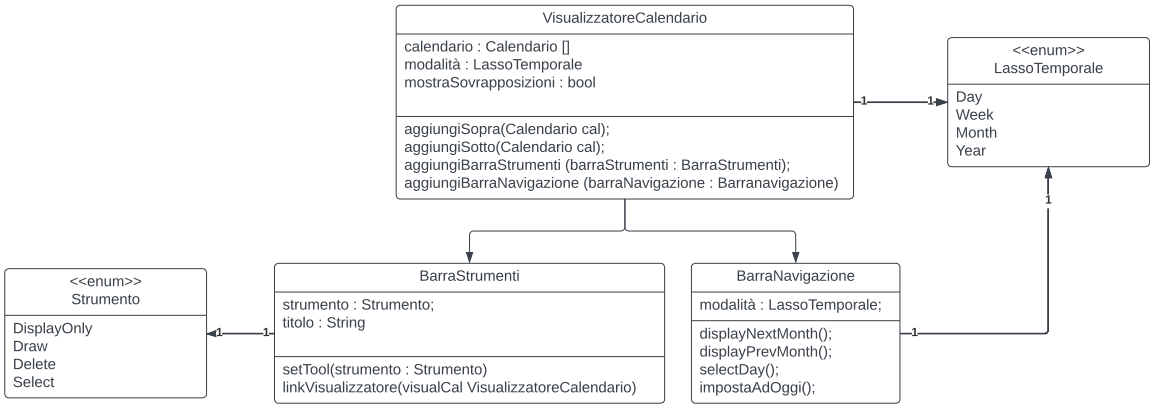

Figura 4: Visualizzatore Calendario

2.5 Gestore Autenticazione

La gestione della creazione dell'account e dell'autenticazione degli utenti è centralizzata nella classe `GestoreAccessi`. Questa classe si occupa di tutte le operazioni relative agli accessi, garantendo che le operazioni di registrazione e login siano gestite in modo sicuro e organizzato.
La classe `GestoreAccessi` si avvale di `LettoreDB` e `ScrittoreDB` per gestire le operazioni di lettura e scrittura sul database. `LettoreDB` è responsabile della lettura dei dati, come la verifica delle credenziali durante il login, mentre `ScrittoreDB` gestisce la scrittura

dei dati, come la registrazione di nuovi utenti e la memorizzazione delle informazioni di autenticazione.
In aggiunta, la `GestioneMail` è gestita dalla classe `GestoreMail`, che è dedicata esclusivamente alle operazioni legate all'uso delle email. Questa classe si occupa di inviare e ricevere messaggi, gestire notifiche e altre comunicazioni via email necessarie per il funzionamento del sistema.
In sintesi, la classe `GestoreAccessi` centralizza la gestione dell'autenticazione e della creazione degli account, supportata dalle operazioni di database tramite `LettoreDB` e `ScrittoreDB`, mentre `GestoreMail` si occupa delle comunicazioni via email.

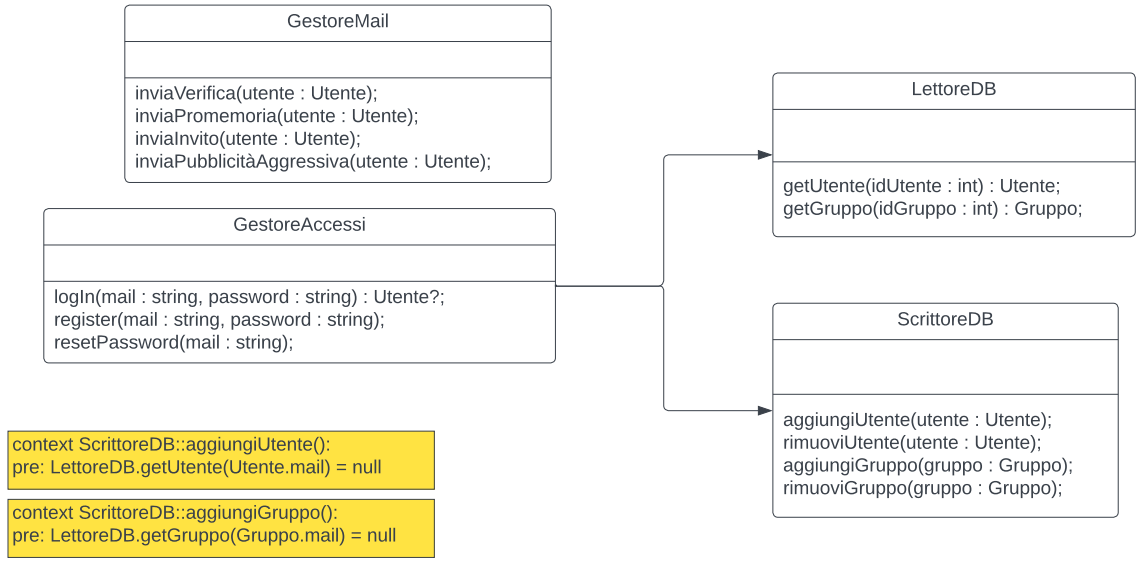

Figura 5: Gestore Autenticazione

2.6 Diagramma delle classi Complessivo

Segue il diagramma delle classi, che include tutte le classi del sistema. Alcuni commenti sono presenti nel diagramma per chiarire le funzionalità che potrebbero non essere immediatamente evidenti.

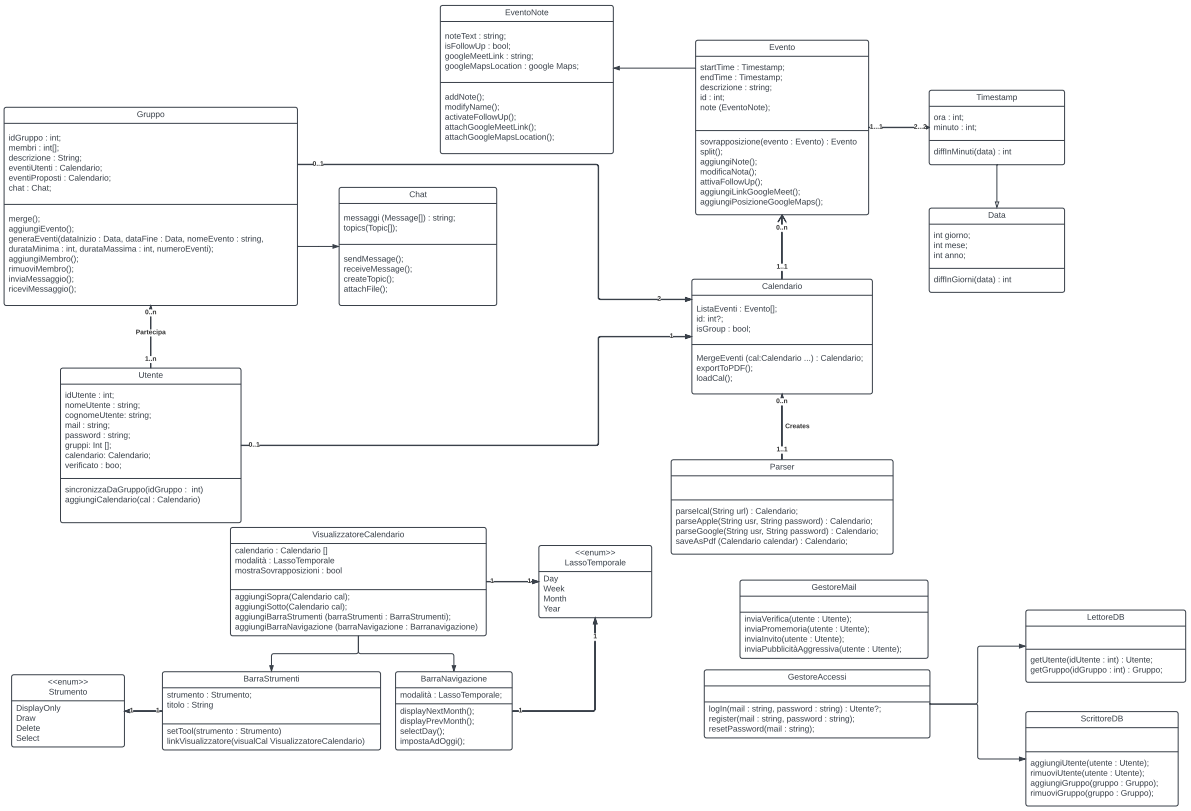

Figura 6: Diagramma delle classi Complessivo

3 Codice In Object Constraint Language

In questo capitolo è descritta in modo formale la logica prevista nell'ambito di alcune operazioni di alcune classi. Tale logica viene descritta in Object Constraint Language (OCL) perché tali concetti non sono esprimibili in nessun altro modo formale nel contesto di UML.

3.1 Verifica dello stato di autenticazione

Il valore booleano verifica se l'utente è stato verificato attraverso l'autenticazione via mail. Ogni azione dell'utente è preclusa fino a quando non è verificato.
La condizione è sulla classe `Utente`:

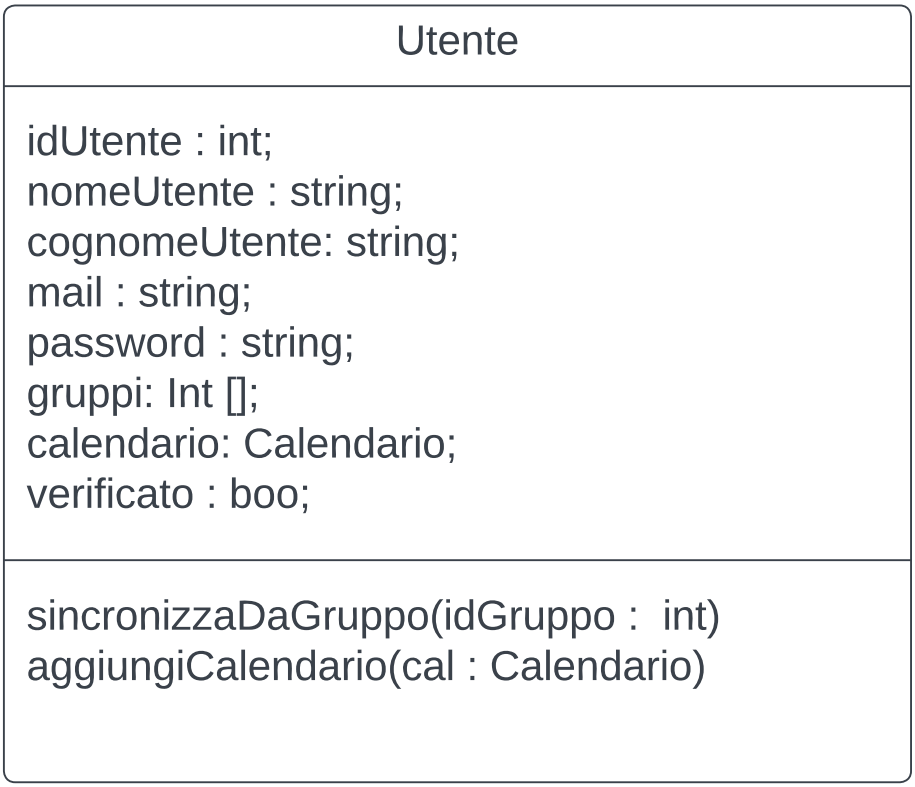

Figura 7: Classe Utente

è espressa in OCL attraverso una precondizione con questo codice:

```
context Utente::sincronizzaDaGruppo():
pre: verificato = true
```

```
context Utente::aggiungiCalendario():
pre: verificato = true
```

Figura 8: Codice OCL per verifica dello stato di autenticazione

3.2 Aggiunta nuovo utente e nuovo Gruppo

L'utente che aggiunge un account non può essere già registrato con un account con la stessa mail.
La condizione riguarda la classe `ScrittoreDB` e coinvolge:

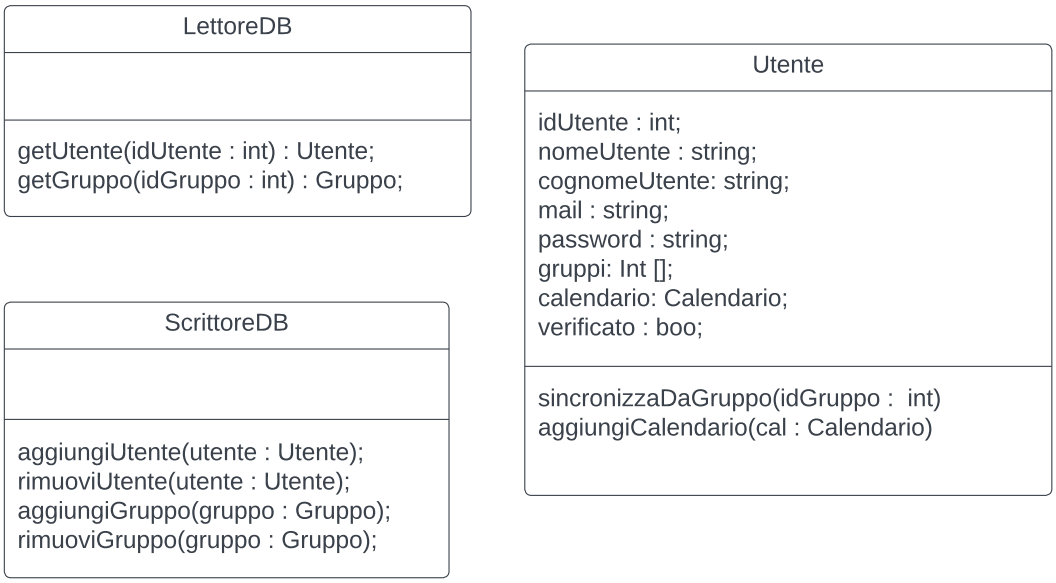

Figura 9: Aggiunta nuovo utente

è espressa in OCL attraverso una precondizione con questo codice:

```
context ScrittoreDB::aggiungiUtente():
pre: LettoreDB.getUtente(Utente.mail) = null
```

Figura 10: Codice OCL per aggiunta nuovo utente

Analogamente per i gruppi che coinvolge, oltre alle classi `LettoreDB` e `ScrittoreDB` sopra, anche la classe `Gruppo`:

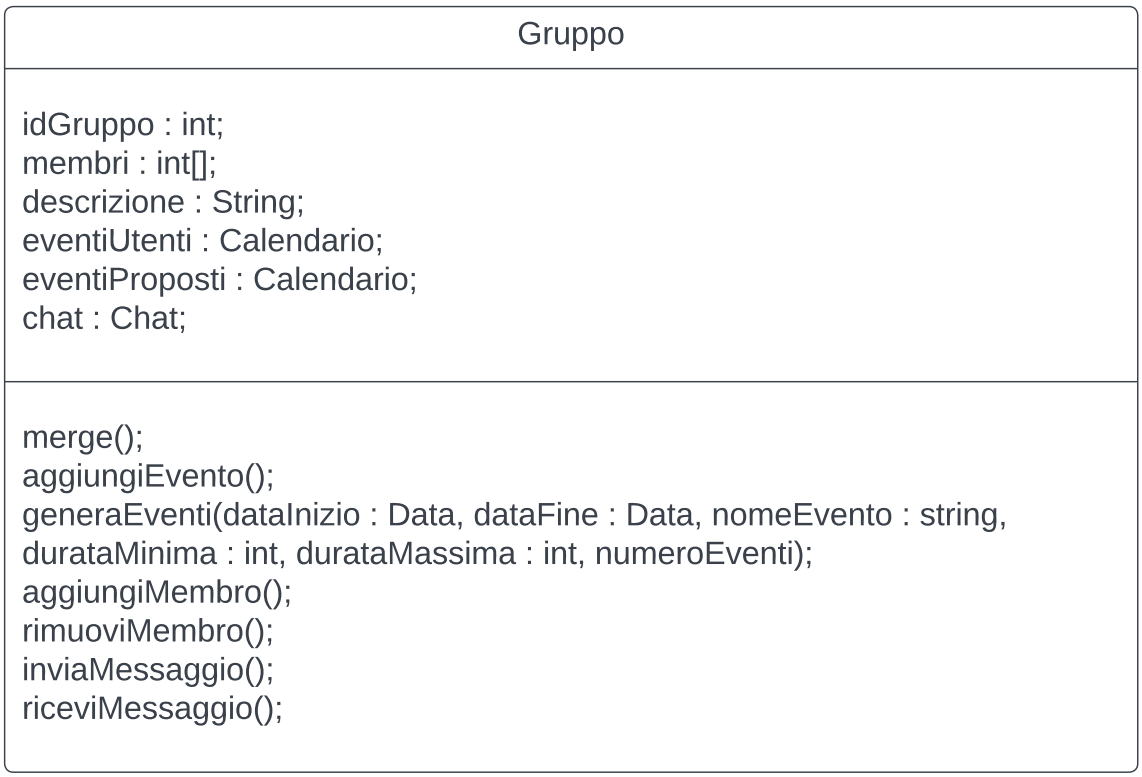

Figura 11: Classe Gruppo

è espressa quindi in OCL:

```
context ScrittoreDB::aggiungiGruppo():
pre: LettoreDB.getGruppo(Gruppo.mail) = null
```

Figura 12: Codice OCL per aggiunta nuovo gruppo

3.3 Id del Calendario

L'id del calendario si riferisce all'id dell'utente o del gruppo al quale appartiene.
La condizione è sulla classe `Calendario` e coinvolge anche le seguenti classi:

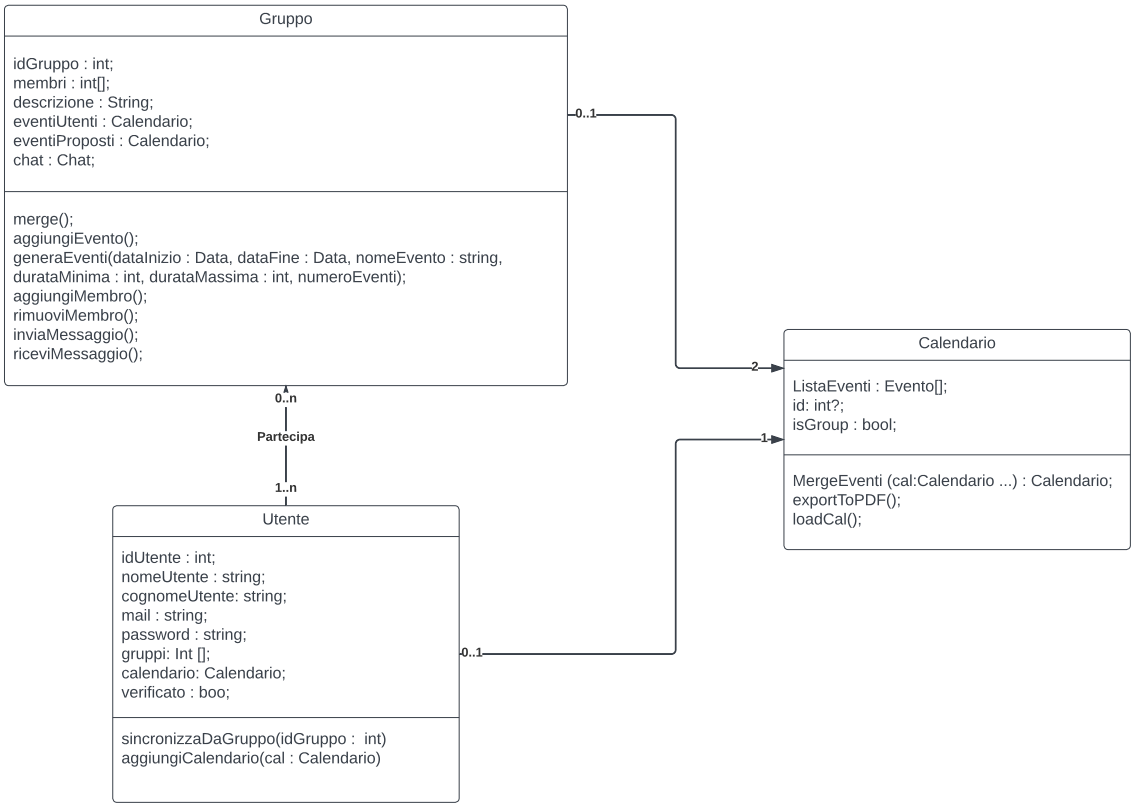

Figura 13: Id del Calendario

è espressa in OCL attraverso una invariante con questo codice:

```
context Calendario inv:
(self.id = Utente.idUtente) or (self.id = Gruppo.idGruppo)
```

Figura 14: Codice OCL per Id del Calendario

4 Diagramma delle classi con codice OCL

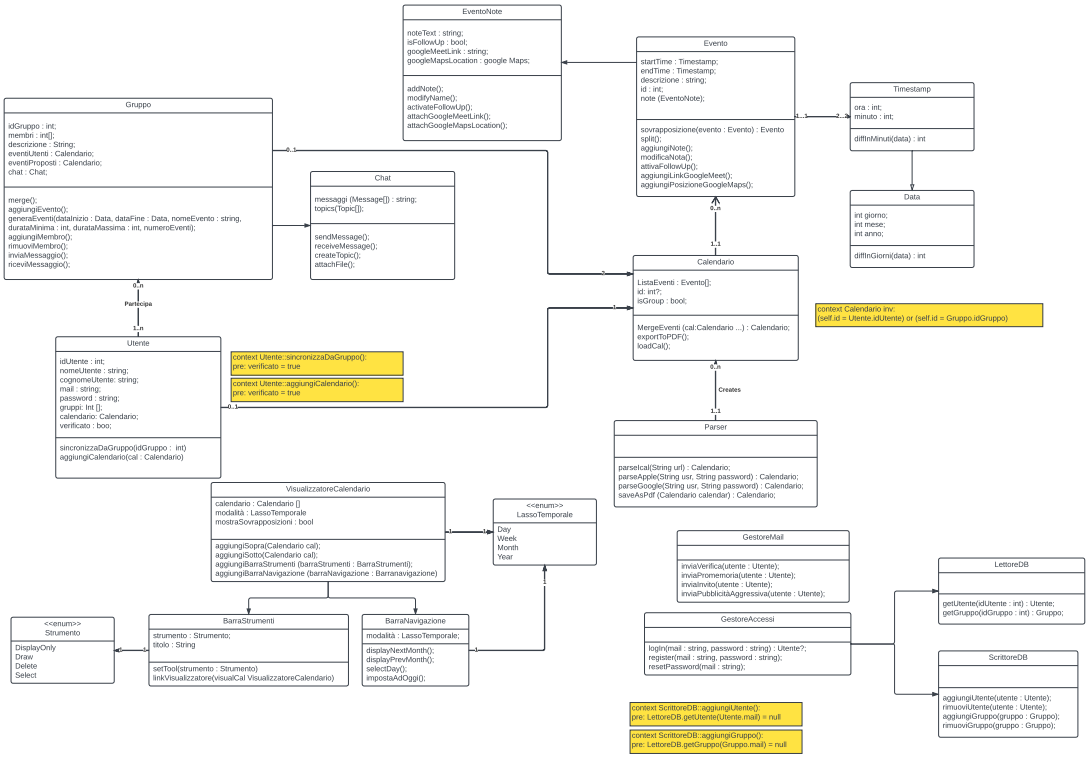

Figura 15: Diagramma delle classi con codice OCL