WYD

Giacomo Romanini 0001093086

Marzo 2025

Indice

1 Abstract

Wyd è un'applicazione che permette ai clienti di organizzare i propri impegni, siano essi confermati oppure proposti.
Mette a disposizione due calendari, il primo con gli eventi in cui l'utente è convinto di partecipare, il secondo in cui vengono riuniti gli eventi a cui l'utente è stato invitato ma senza aver ancora dato disponibilità.
L'utente ha la possibilità di creare, modificare, confermare o disdire un evento, ma anche condividerlo con altri o allegarci foto. La condivisione di un evento può avvenire con applicazioni esterne tramite la generazione di un link o grazie all'ausilio di gruppi di profili. Inoltre, al termine di un evento, l'applicazione carica automaticamente le foto scattate durante l'evento, per allegarle a seguito della conferma dell'utente.
L'utente può infatti cercare altri profili e creare gruppi con i profili trovati.
Tutta l'interazione avviene tramite l'utilizzo di profili, che permettono di suddividere semanticamente gli eventi e le relazioni.

2 Progettazione

2.1 Progettazione Architetturale

2.1.1 Requisiti non funzionali

Dall'analisi dei requisiti sono emersi i seguenti requisiti non funzionali:

- Tempo di risposta
- Usabilità
- Affidabilità
- Scalabilità
- Integrità dei dati
- Protezione dei dati
- Sicurezza delle comunicazioni

L'affidabilità e la scalabilità assumono fondamentale importanza vista la natura del software, che deve permettere agli utenti di poter organizzare, coordinare e condividere eventi. La compromissione di questi risulterebbe in un peggioramento dell'esperienza utente, da cui consegue una perdita di reputazione o di fedeltà del cliente. Sarà necessario assicurare la sicurezza fisica dei dati immagazzinati nel sistema, così come la sicurezza software dei dati e delle comunicazioni. In caso di compromissione la perdita d'immagine e i risvolti legali sarebbero significativi. L'utilizzo di protocolli e tecnologie standard del settore dovrebbero garantire la sicurezza del sitema senza peggiorarne significativamente l'usabilità o le prestazioni. Nonostante il sistema non presenti vincoli di tempo stringenti, una caratteristica essenziale dell'applicazione sarà la velocità di distribuire gli aggiornamenti ai vari utenti. Inoltre, l'intuitività dell'interfaccia è fondamentale per l'usabilità e la diffusione del prodotto.

2.1.2 Scelte tecnologiche

La scelta tecnologica principale ricade sul tipo di applicazione che si andrà a sviluppare. In questo caso la scelta è quella di sviluppare sia un'applicazione sia da browser web, che tramite dispositivo mobile, per i seguenti motivi:

1. un'interfaccia web consente di avere una piattaforma standard accessibile da quasi tutti i dispositivi, con il solo requisito di un browser, potenzialmente permettendo(in base alla dimensione dello schermo) una maggior facilità di utilizzo, per rispondere all'esigenza organizzativa di medio o lungo termine. In questo modo si evita di restringere le possibilità di accesso al servizio.

2. l'applicazione mobile permette una gestione a breve termine e un aggiornamento costante, ed è fondamentale per recuperare le immagini dell'utente.

2.1.3 Scelta dell'architettura

Dopo una rapida analisi, si è constatato che l'architettura più adeguata per il sistema è l'**architettura client-server a 3 livelli**.

L1 – Client
La componente lato Client implementerà l'interfaccia utente, gestendo le interazioni del cliente e richiedendo i dati al server, salvandoli in una cache locale laddove la tecnologia lo renda possibile. Inoltre avrà la responsabilità di recuperare le immagini scattate durante l'evento.

L2 – Server
Per distribuire meglio il carico, si è deciso di scomporre i server in base alle funzionalità offerte. Si hanno quindi tre server:

- Un server per le funzionalità di autenticazione
- Un server principale che fornisce i servizi agli utenti
- Un server che gestisce la propagazione degli aggiornamenti agli utenti

L3 – Persistenza
La gestione della persistenza verrà implementata in due server con responsabilità differenti:

- Un server sul quale sarà installato un DBMS relazionale che gestisca i dati e le relazioni dei componenti
- Un server per il salvataggio delle immagini e file

Il database relazionale sarà accessibile solo dal server che fornisce i servizi, per garantire il controllo dei ruoli e dei permessi. Le immagini saranno invece accessibili solo in lettura direttamente dai client, che dovranno però essere a conoscenza dei codici che le identificano. Questa conoscenza è considerata sufficente per garantire la confidenza, ammesso che l'identificativo sia lungo abbastanza, ma anche un rischio accettabile, in quanto ridurrà di molto il carico del server principale.

2.1.4 Pattern architetturali e di design

Il pattern Client-Server è stato scelto come pattern architetturale per gestire l'accesso, le immagini e i servizi principali.
Il pattern Event Driven sarà invece usato per notificare i Client degli aggiornamenti. A livello di design distinguiamo tre componenti principali: Model, View e Controller. Mentre la View sarà delegata completamente ai Client, Model e Controller saranno invece suddivisi tra il Client e il Server, in base alle necessità di caching dei dati e della località delle operazioni di business. Si riportano di seguito i diagrammi di package e componenti che descrivono l'architettura del sistema.

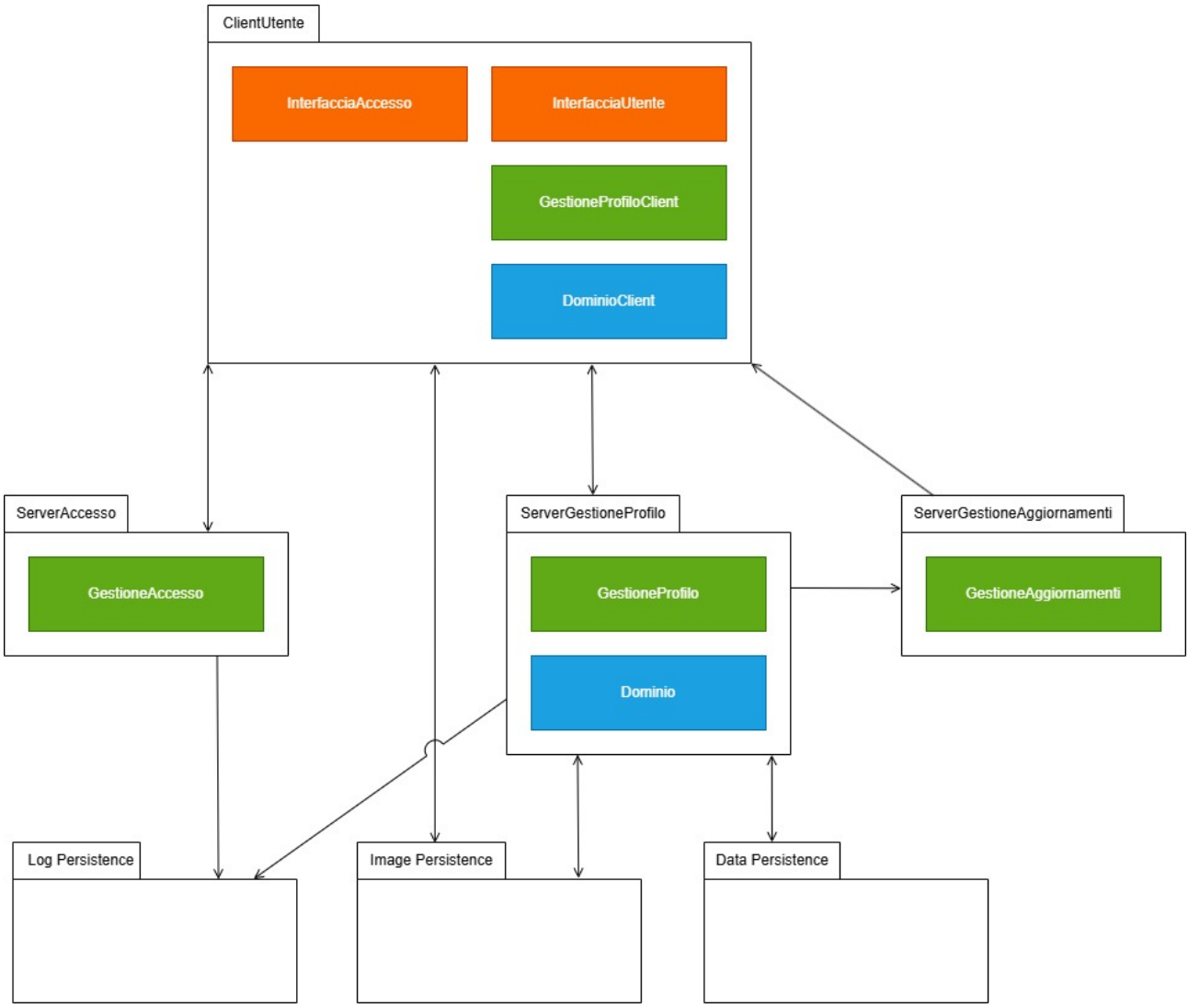


Figura 1: Diagramma dei package

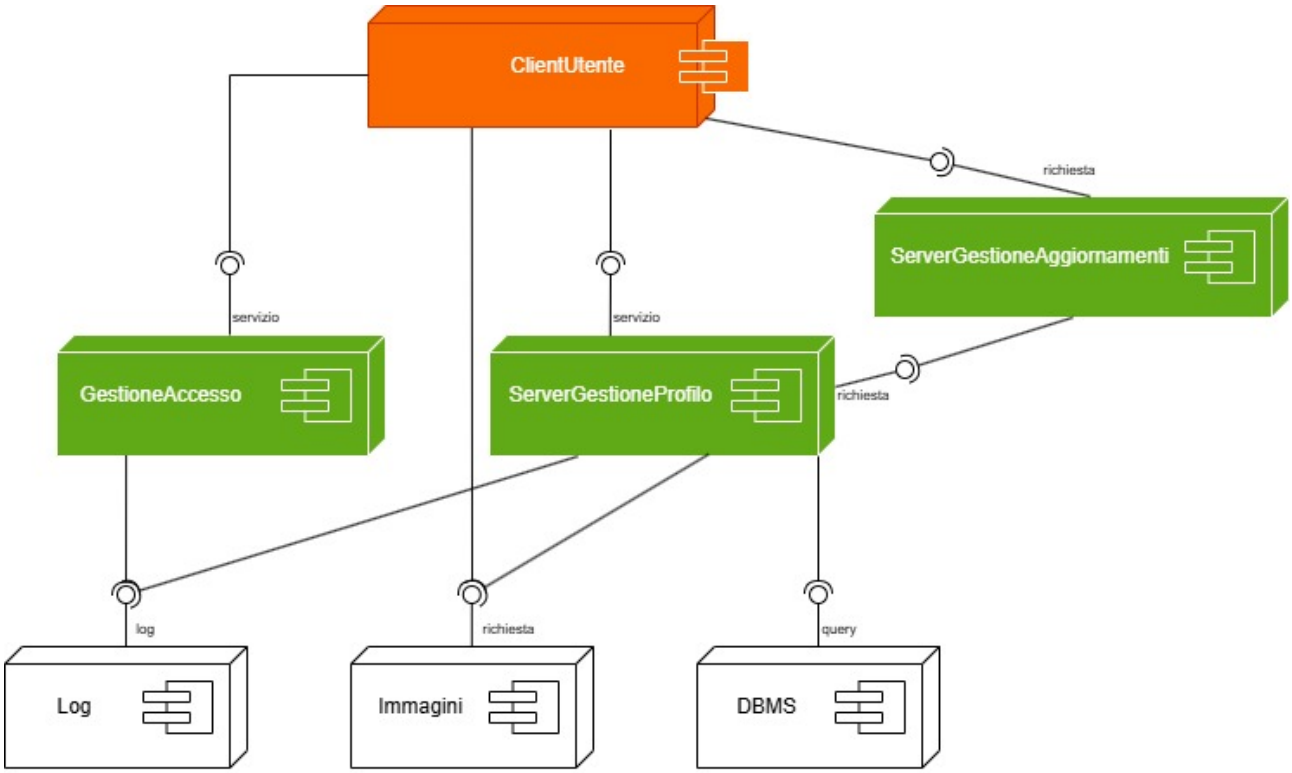


Figura 2: Diagramma dei componenti

2.2 Progettazione di dettaglio

2.2.1 Struttura

Struttura: Dominio

Per quanto riguarda il dominio, distinguiamo il dominio del server dal dominio del client.

Diagramma di dettaglio: Dominio Server

Aggiungiamo a tutti i componenti principali un identificativo anfanumerico chiamato hash, e le date di creazione e di ultima modifica per gestire meglio i log e gli aggiornamenti. Inoltre, aggiungiamo un componente per definire il ruolo dell'utente sul profilo.

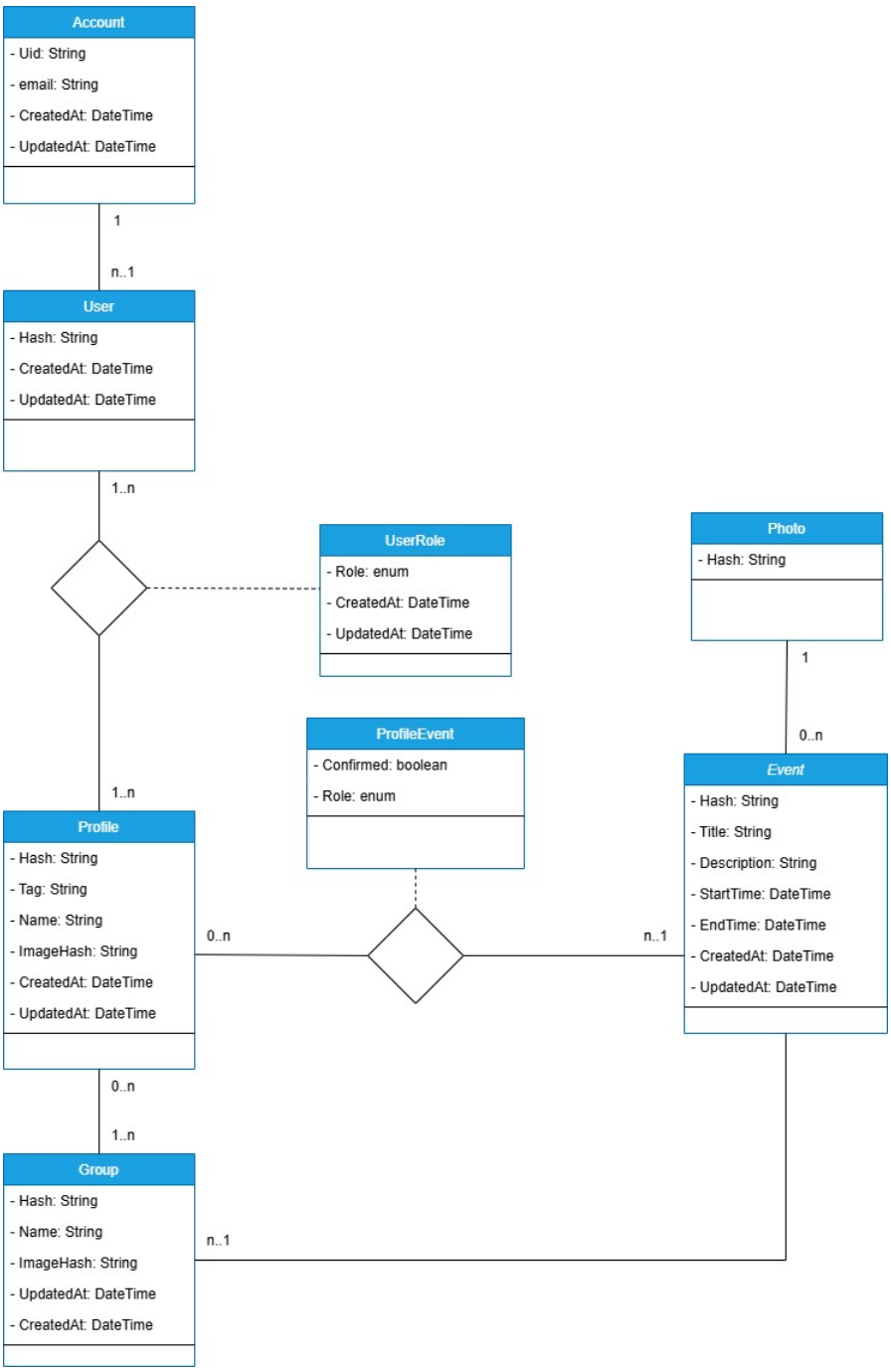
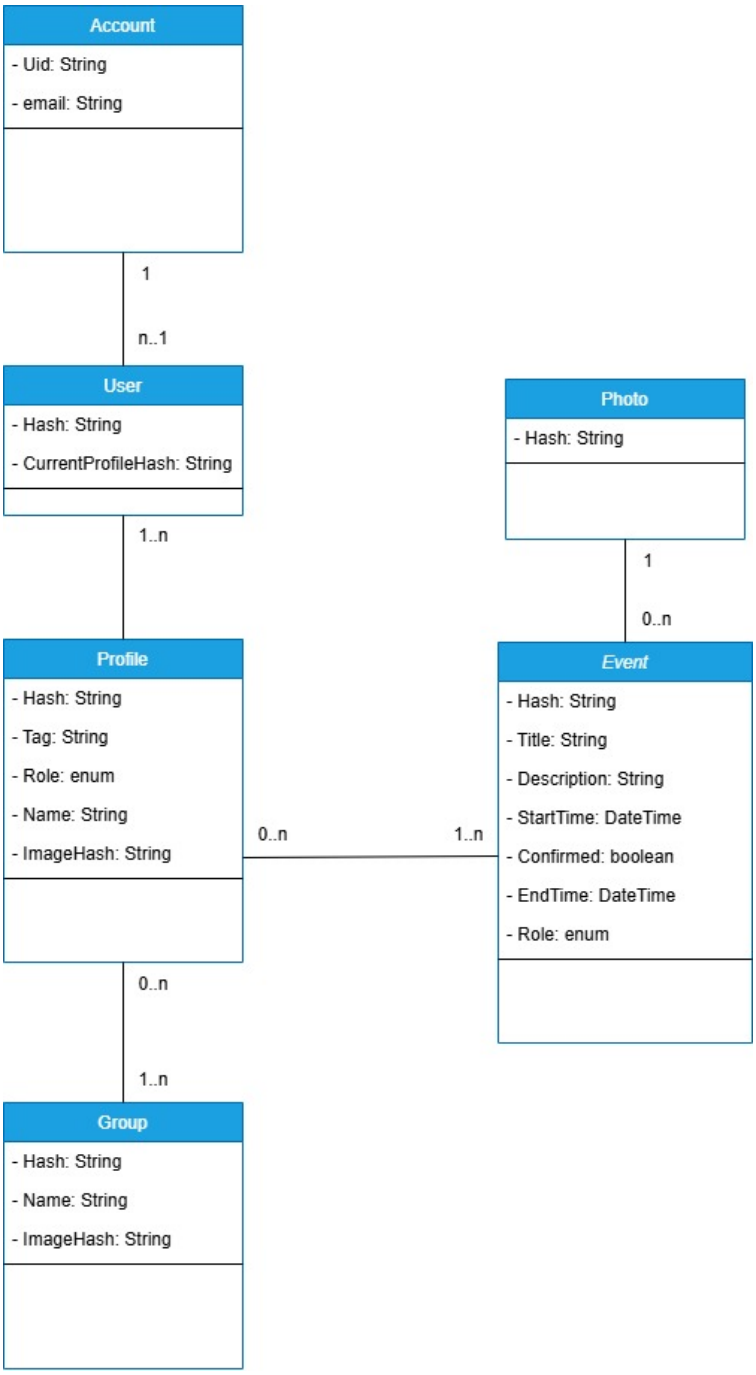


Diagramma di dettaglio: Dominio Client

Dal punto di vista del client alcune relazioni non sono più rilevanti o vengono semplificate integrando alcuni valori all'interno di altri elementi del dominio.



Struttura: Interfacce
Diagramma di dettaglio: Interfacce Gestione Accesso e Gestione Aggiornamenti

Diagramma di dettaglio: Interfacce Client

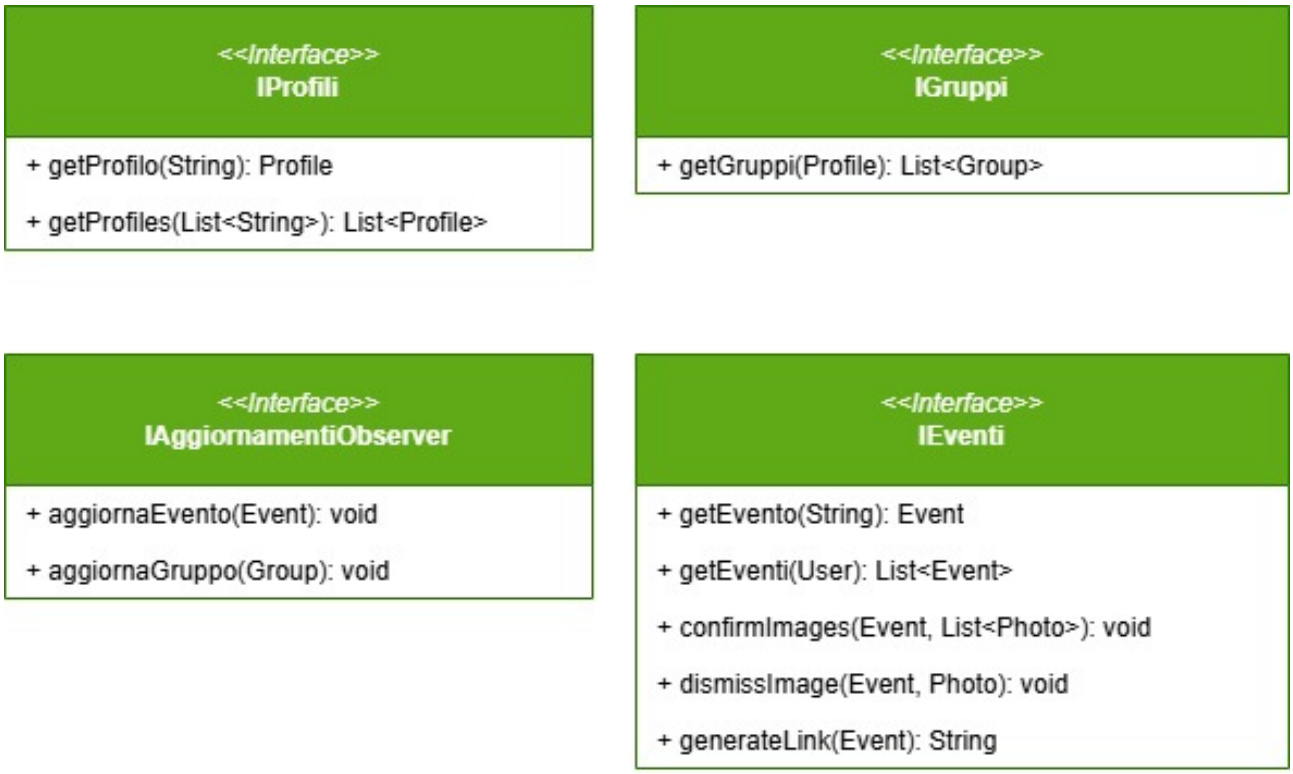
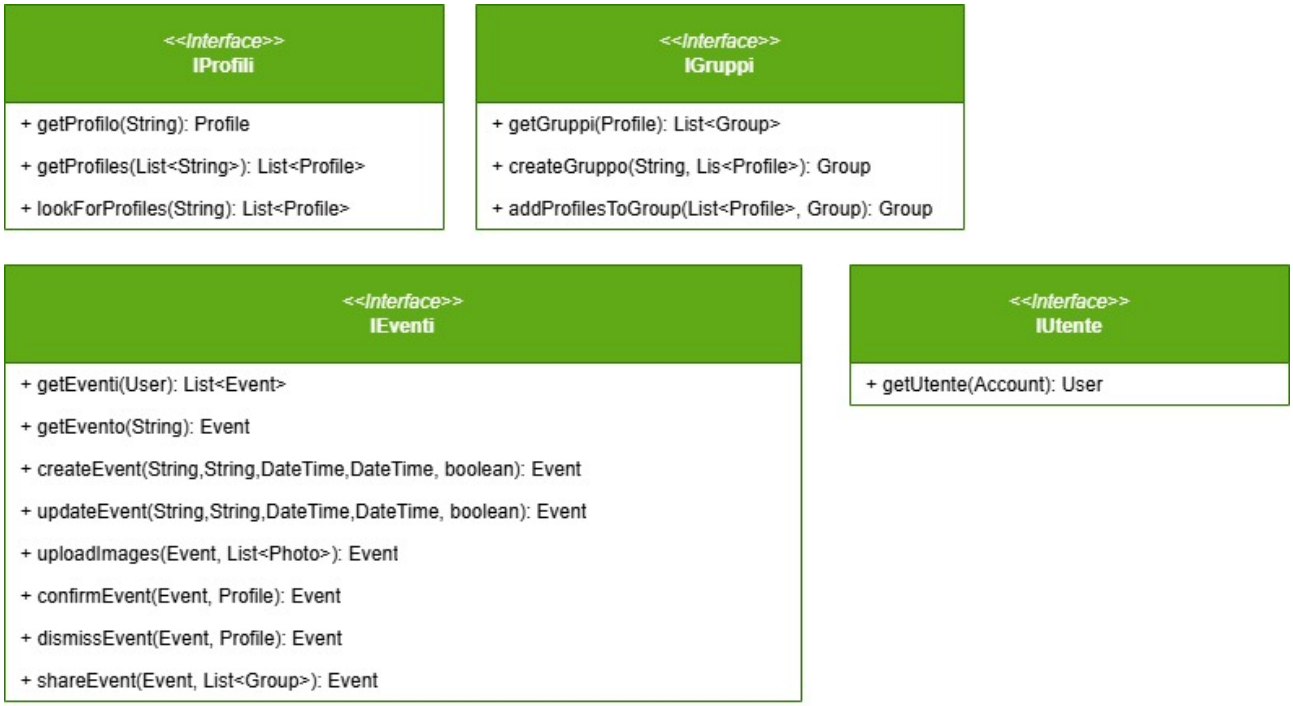


Diagramma di dettaglio: Interfacce Server



Le interfacce client e server risultano molto simili in quanto il client, prima di fare una richiesta dati al server, controllerà la cache locale per cercare i dati. L'aggiunta di tali interfacce consente di applicare il *Dependency Inversion Principle* in modo da disaccoppiare gli utilizzatori dalle implementazioni, che potrebbero cambiare.

Struttura: Controller

Ogni interfaccia verrà implementata da un relativo controller.

Diagramma di dettaglio: Gestione Accesso e Gestione Aggiornamenti

Introduciamo un Controller per la connessione con la persistenza del log

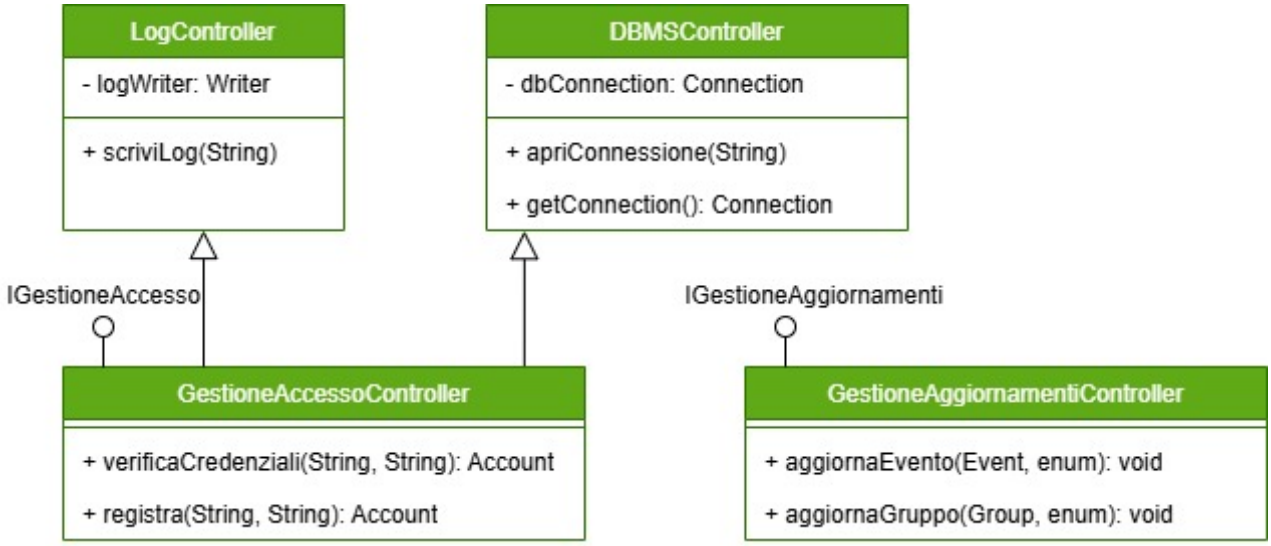


Diagramma di dettaglio: Client

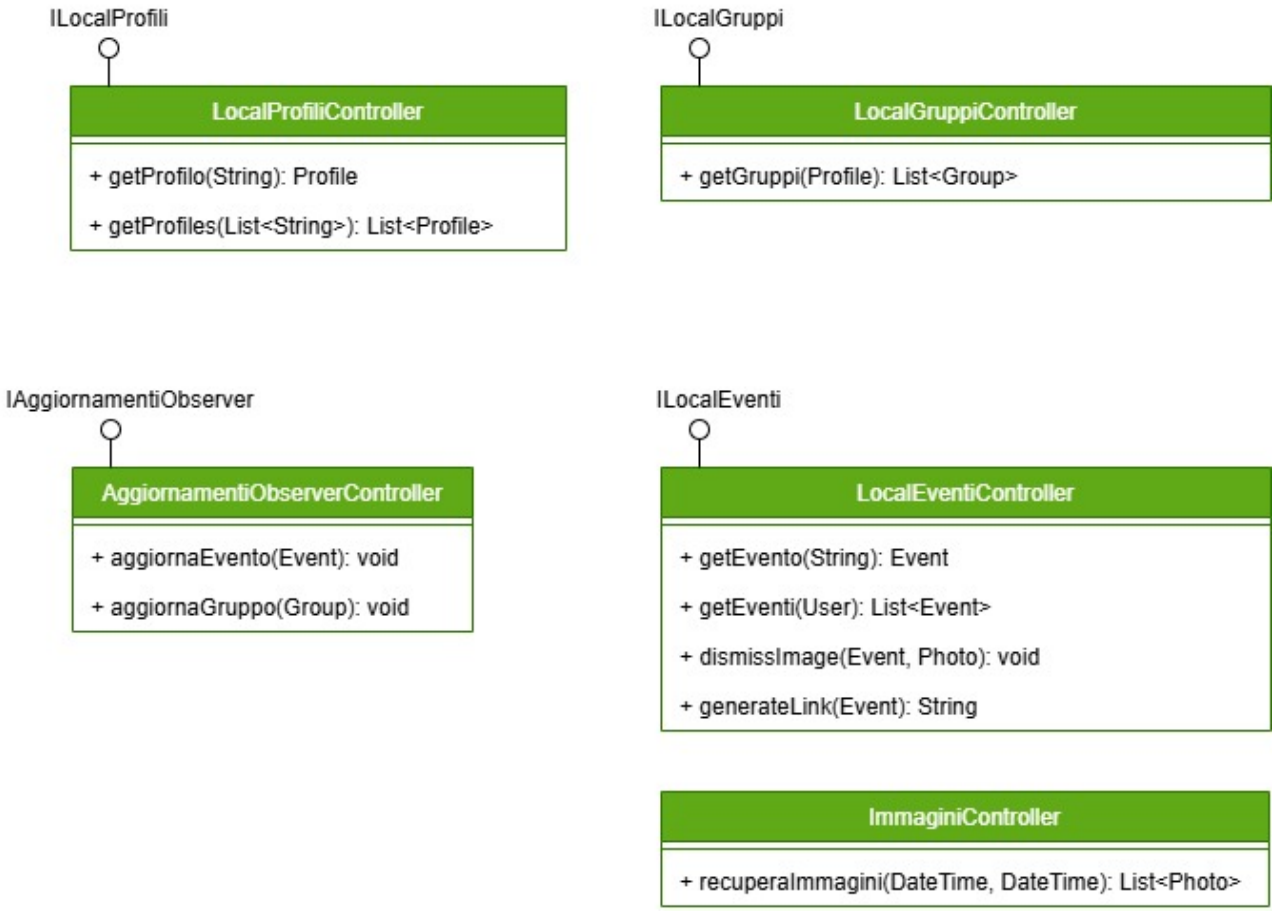


Diagramma di dettaglio: Server

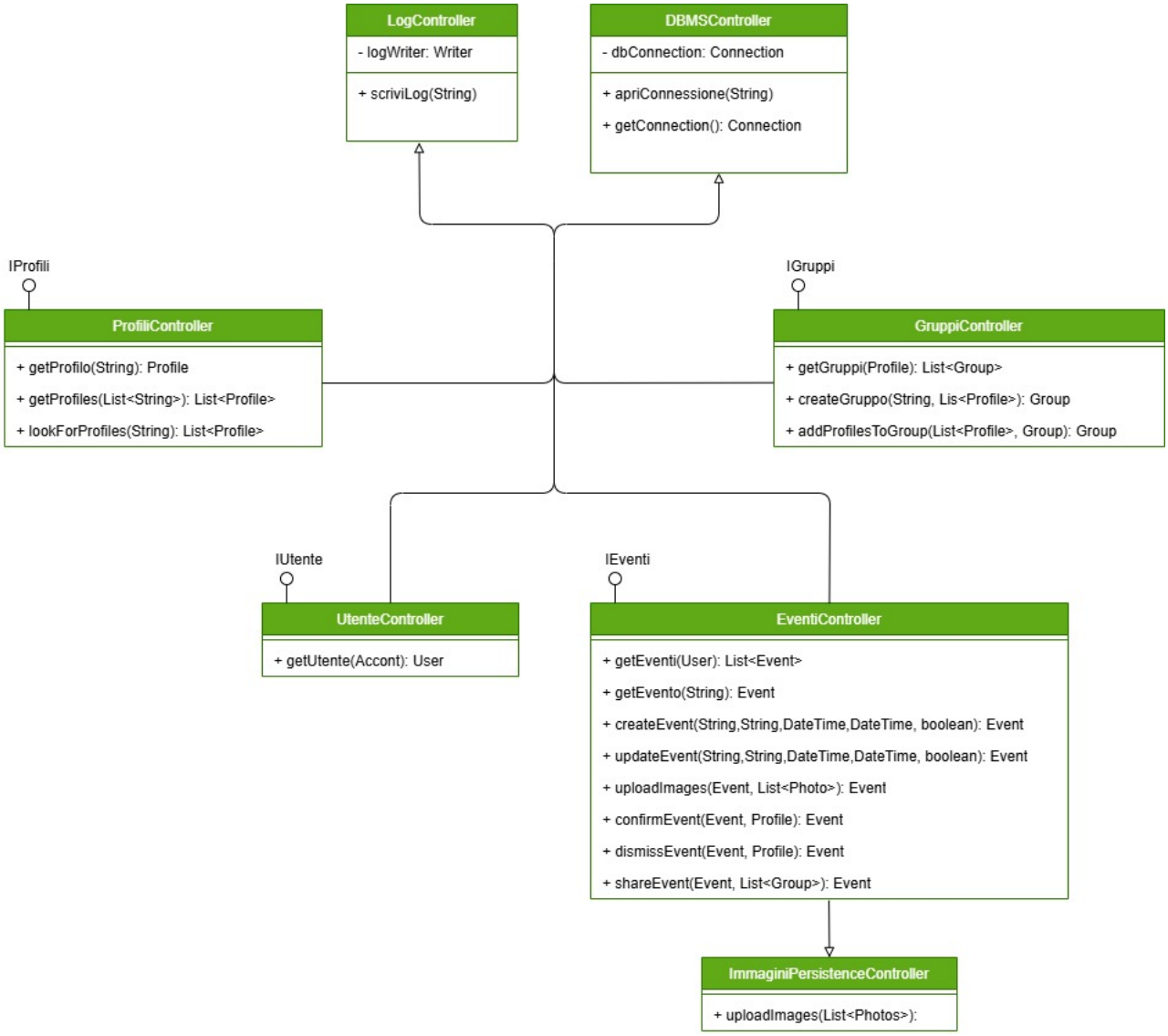
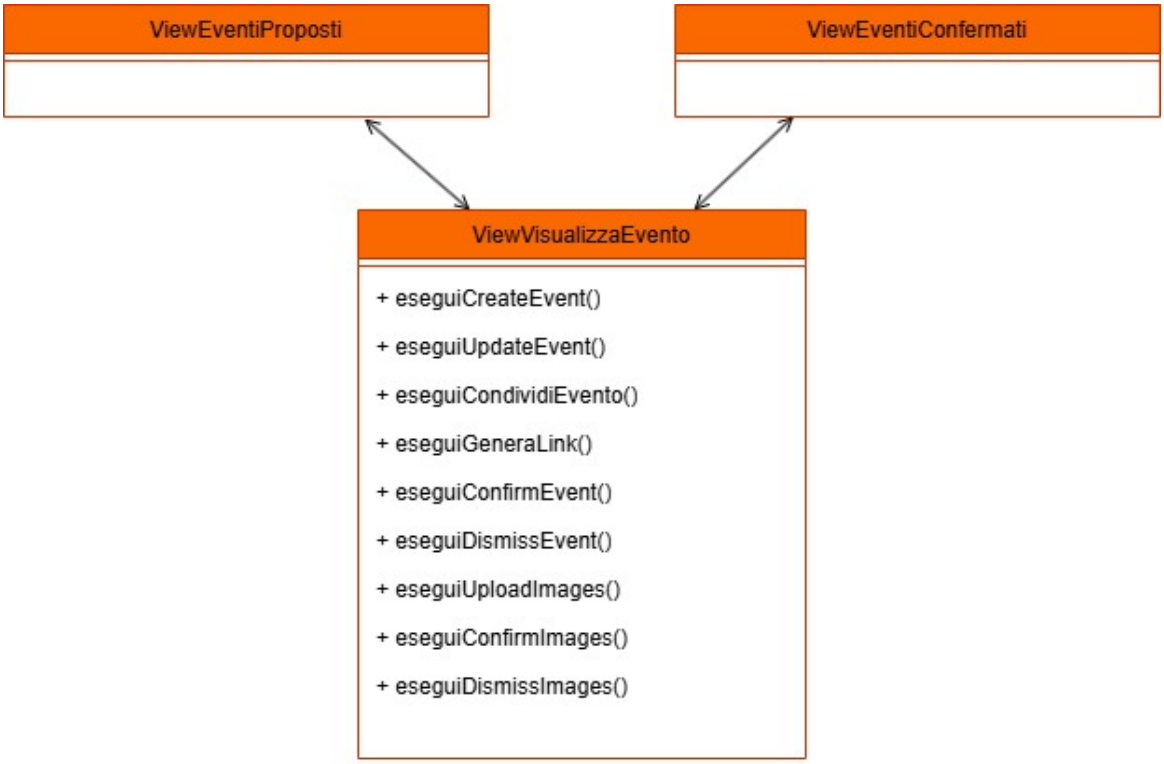


Diagramma di dettaglio: InterfacciaUtente - Eventi



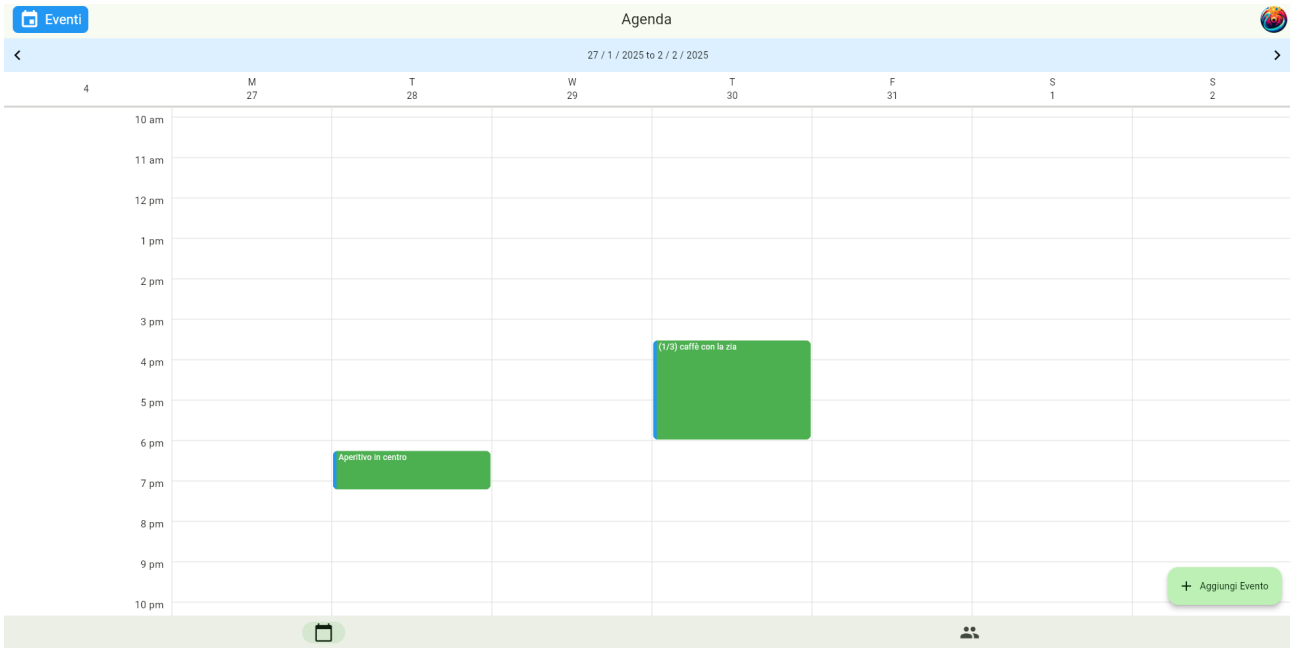


Figura 3: Eventi confermati

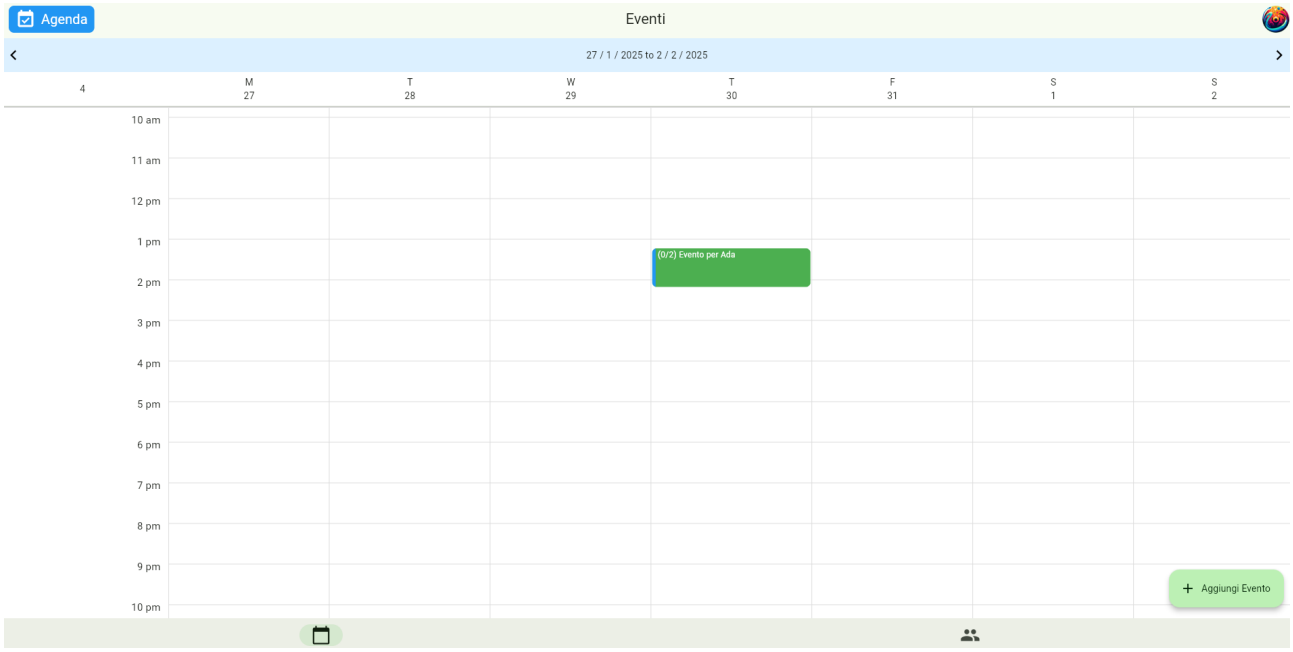


Figura 4: Eventi proposti

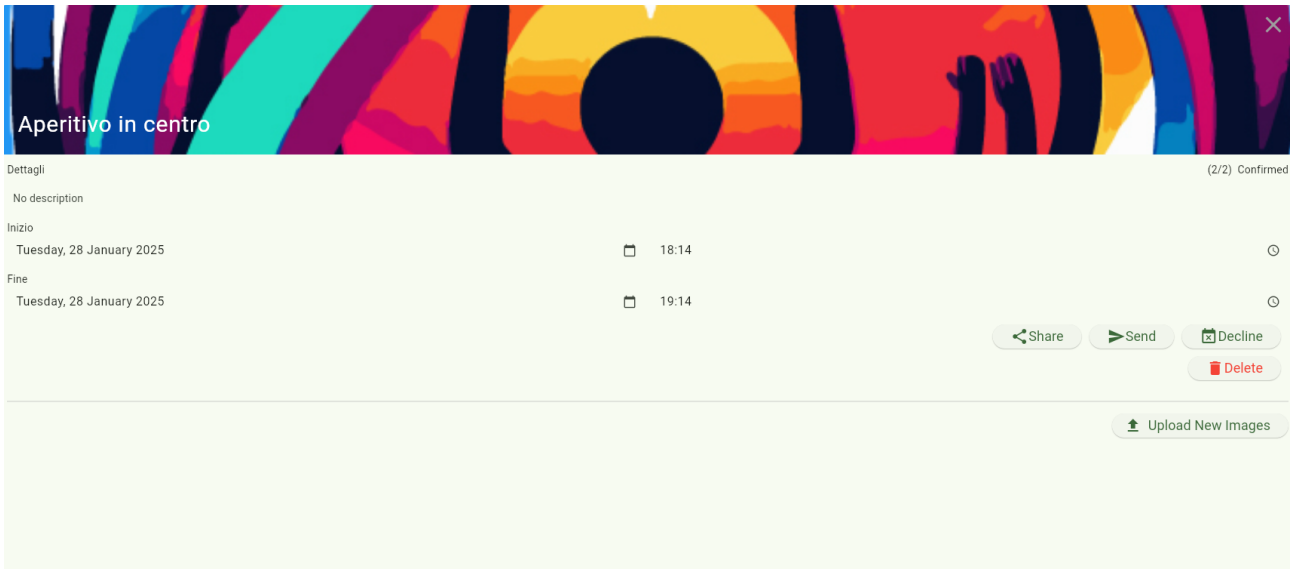
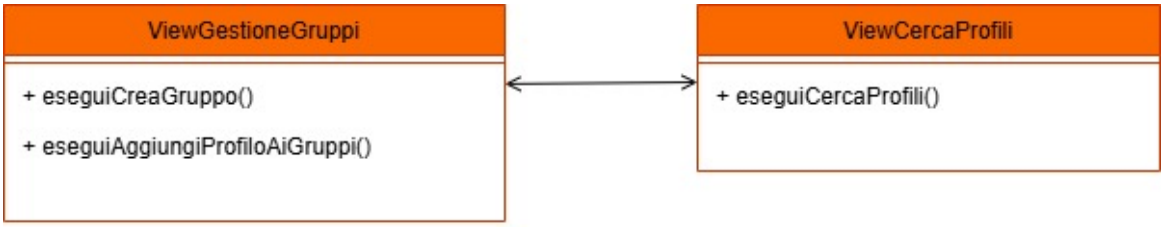


Figura 5: Dettaglio evento

Diagramma di dettaglio: InterfacciaUtente - Gruppi



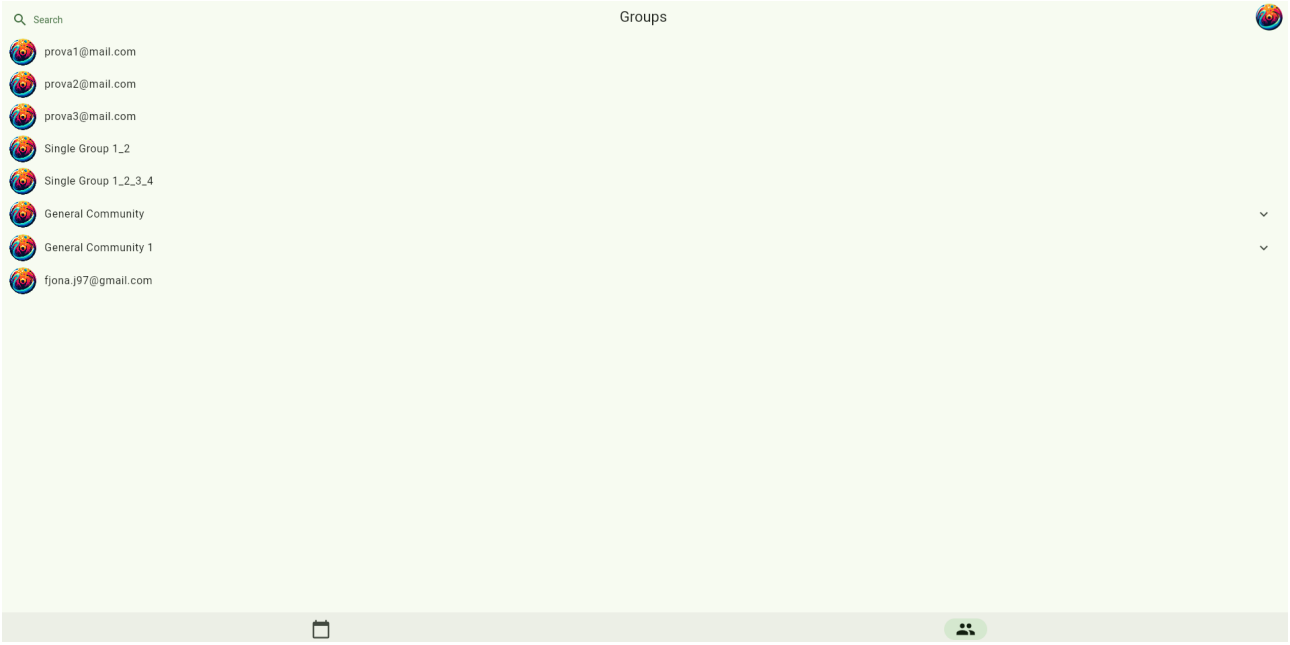


Figura 6: Gruppi

Figura 7: Cerca profili

Diagramma di dettaglio: InterfacciaUtente - Profili

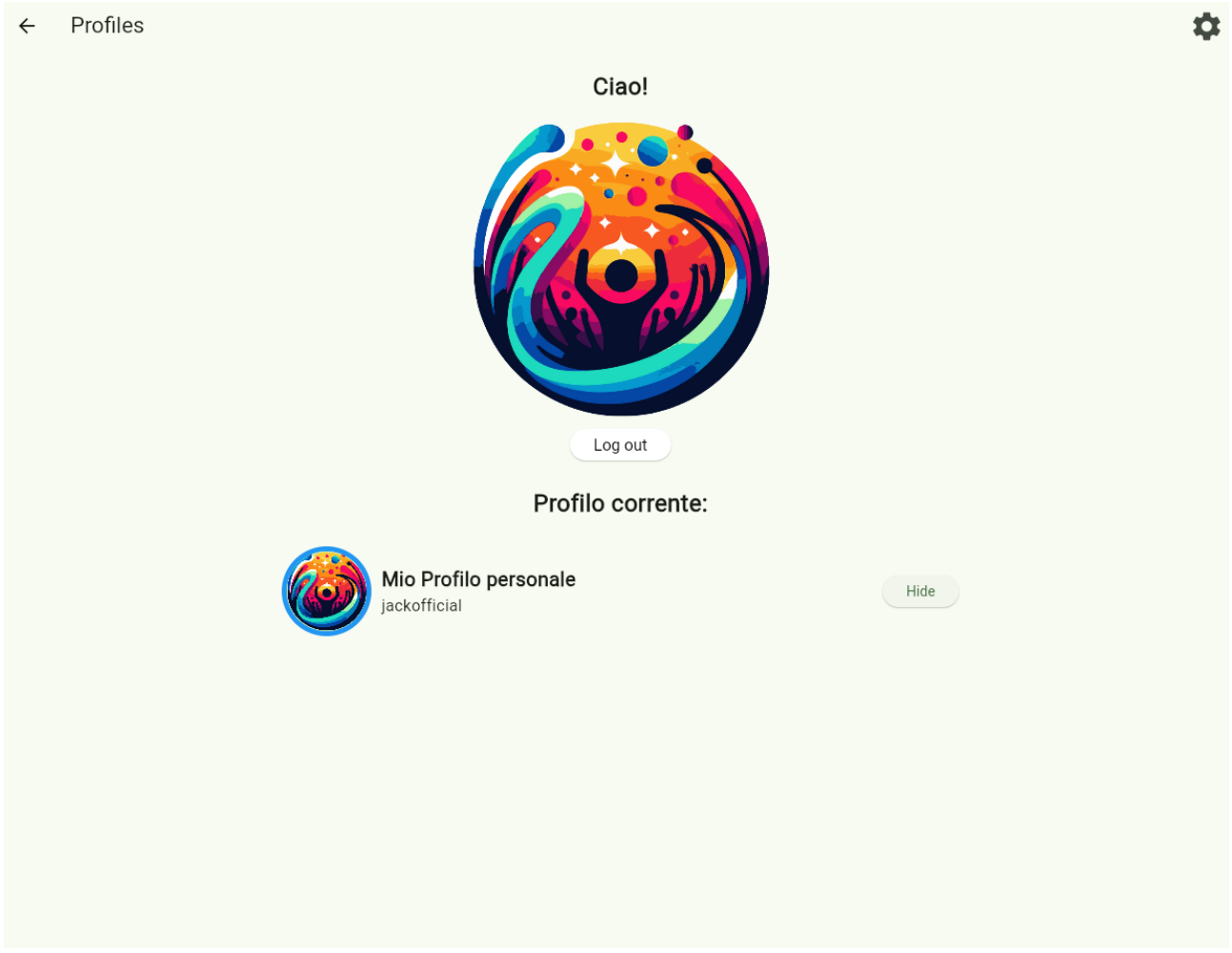


Figura 8: Profili collegati

Diagramma di dettaglio: InterfacciaGestioneAccesso

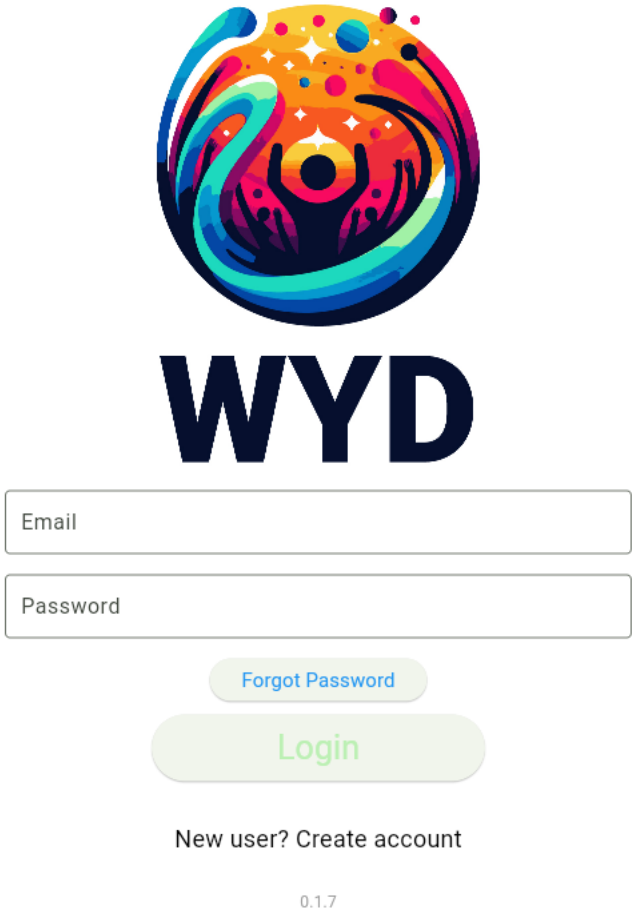


Figura 9: Login

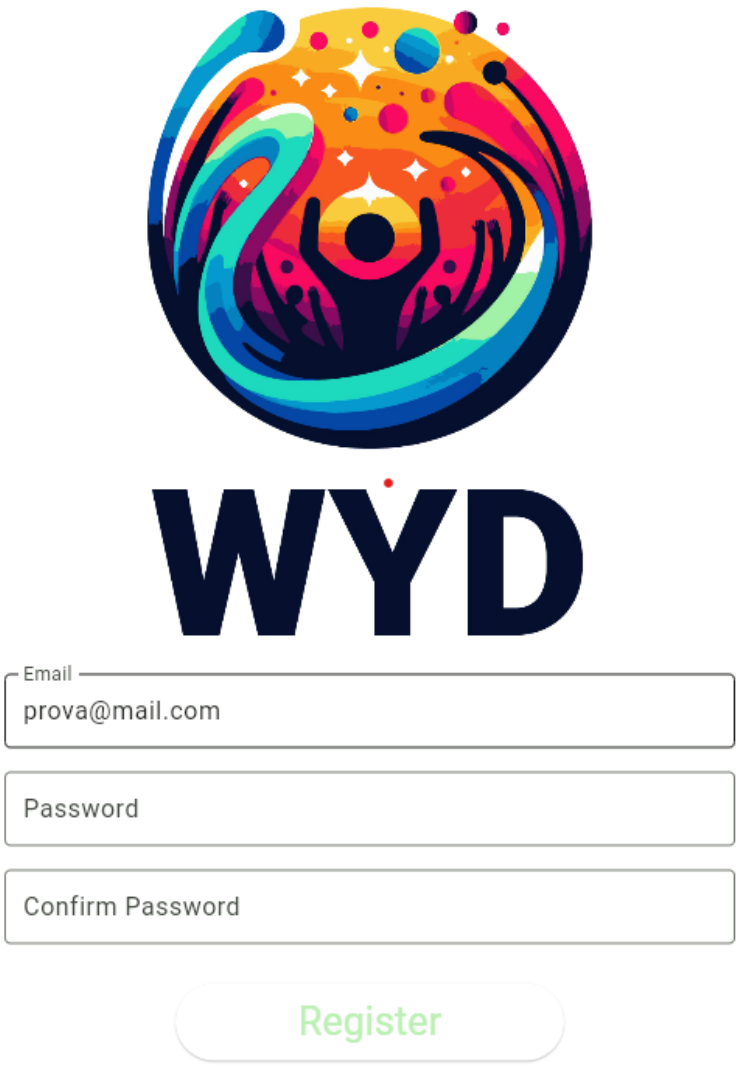


Figura 10: Registrazione

2.2.2 Interazione

Si riportano di seguito i vari diagrammi di sequenza, aggiornati rispetto a quelli visti in fase di analisi.
TODO aggiungere GestioneAggiornamentiController

Diagramma di Sequenza: Registrazione

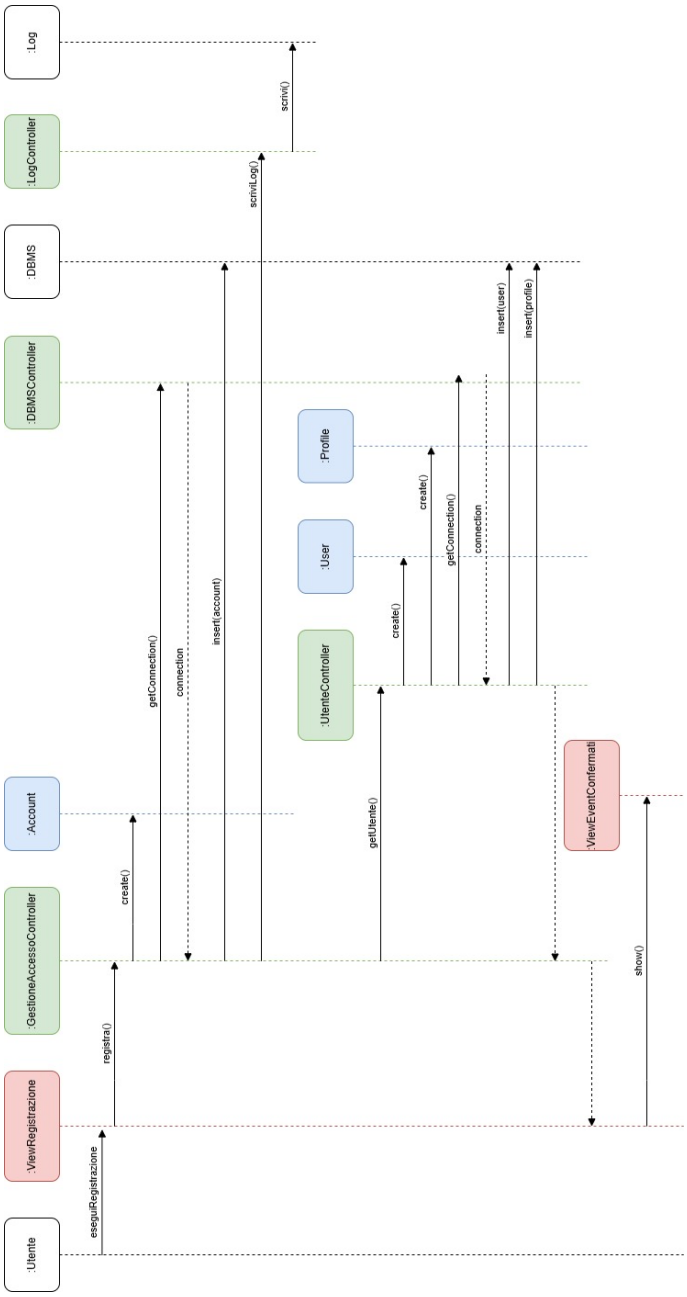


Diagramma di Sequenza: Visualizza Eventi Confermati

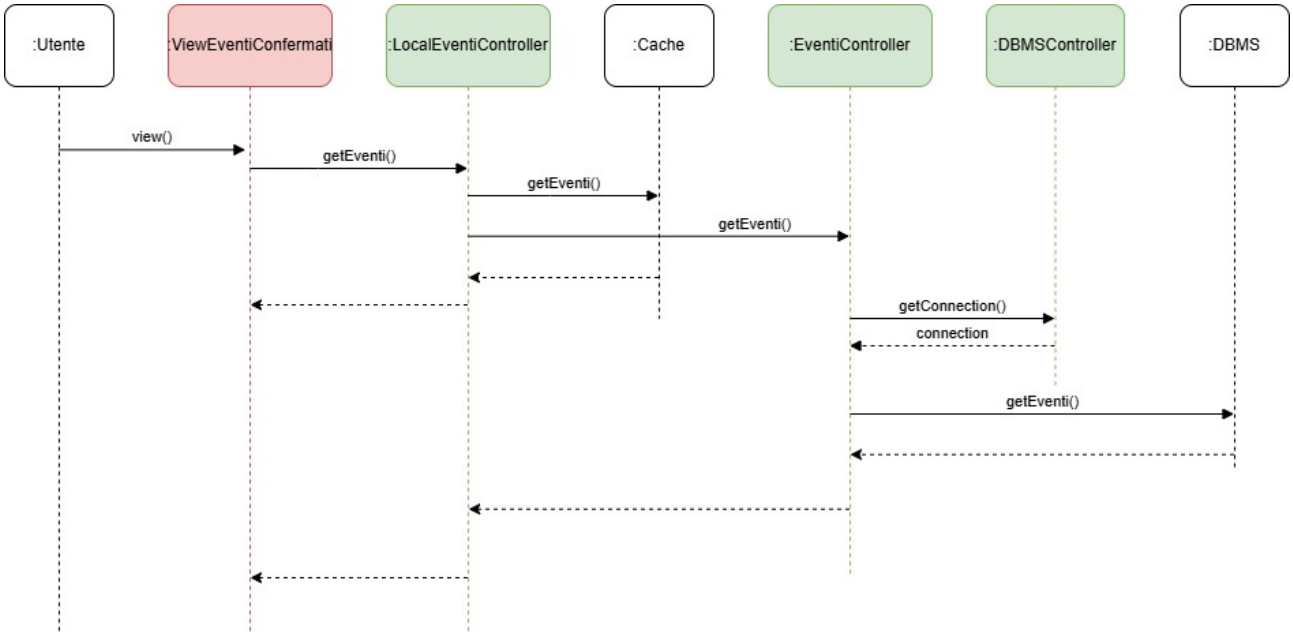


Diagramma di Sequenza: Visualizza Evento

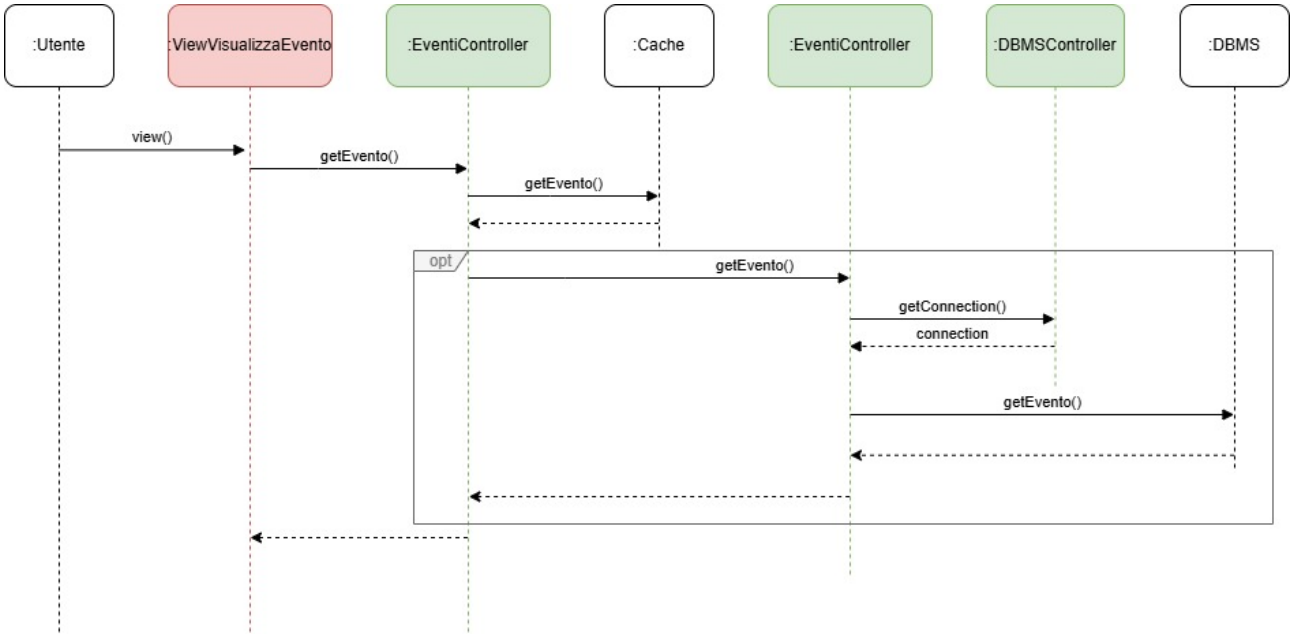


Diagramma di Sequenza: Crea Evento

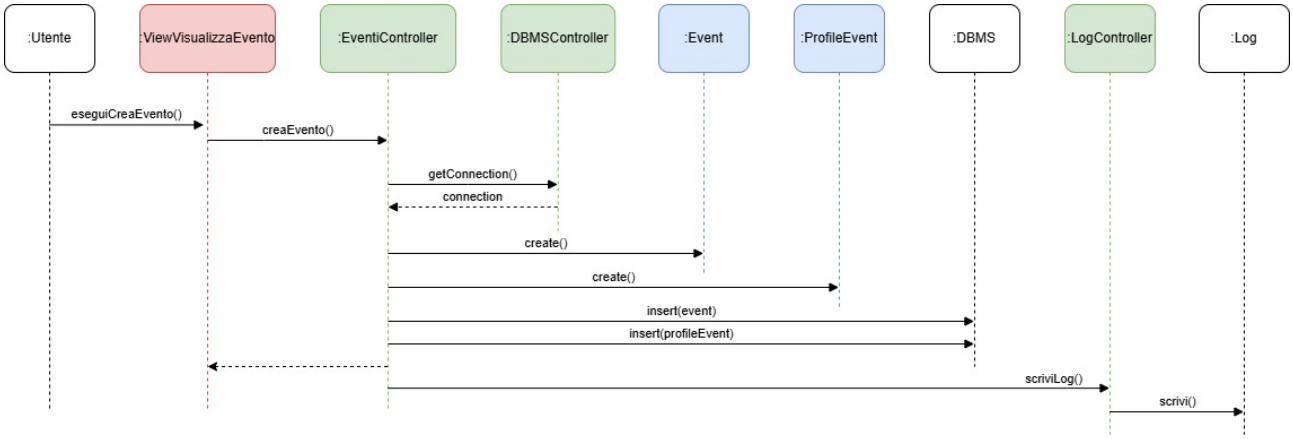


Diagramma di Sequenza: Modifica Evento

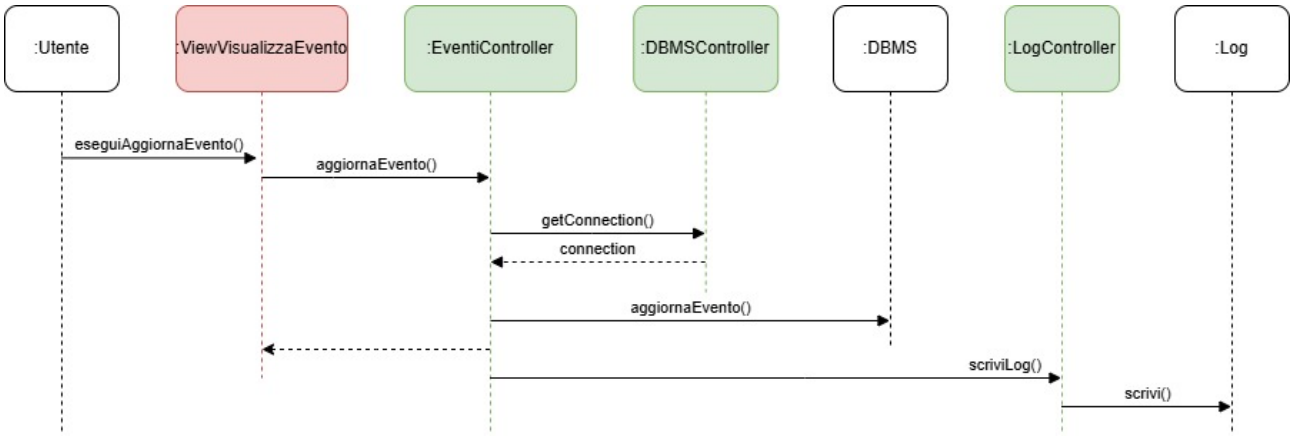


Diagramma di Sequenza: Conferma Evento

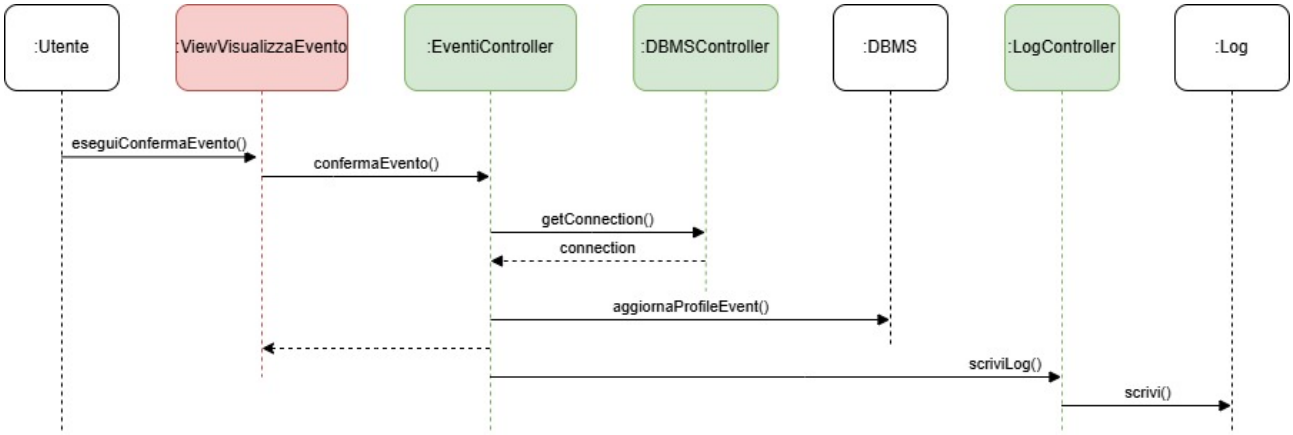


Diagramma di Sequenza: Conferma Immagini

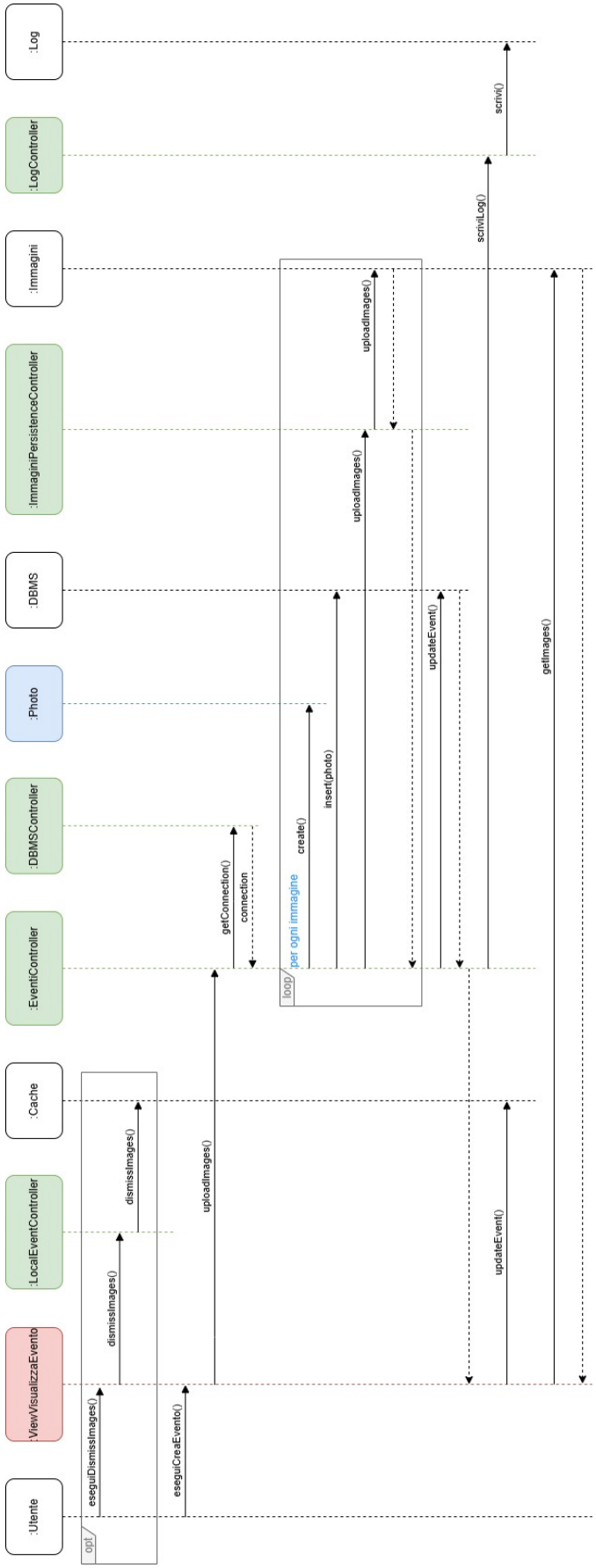
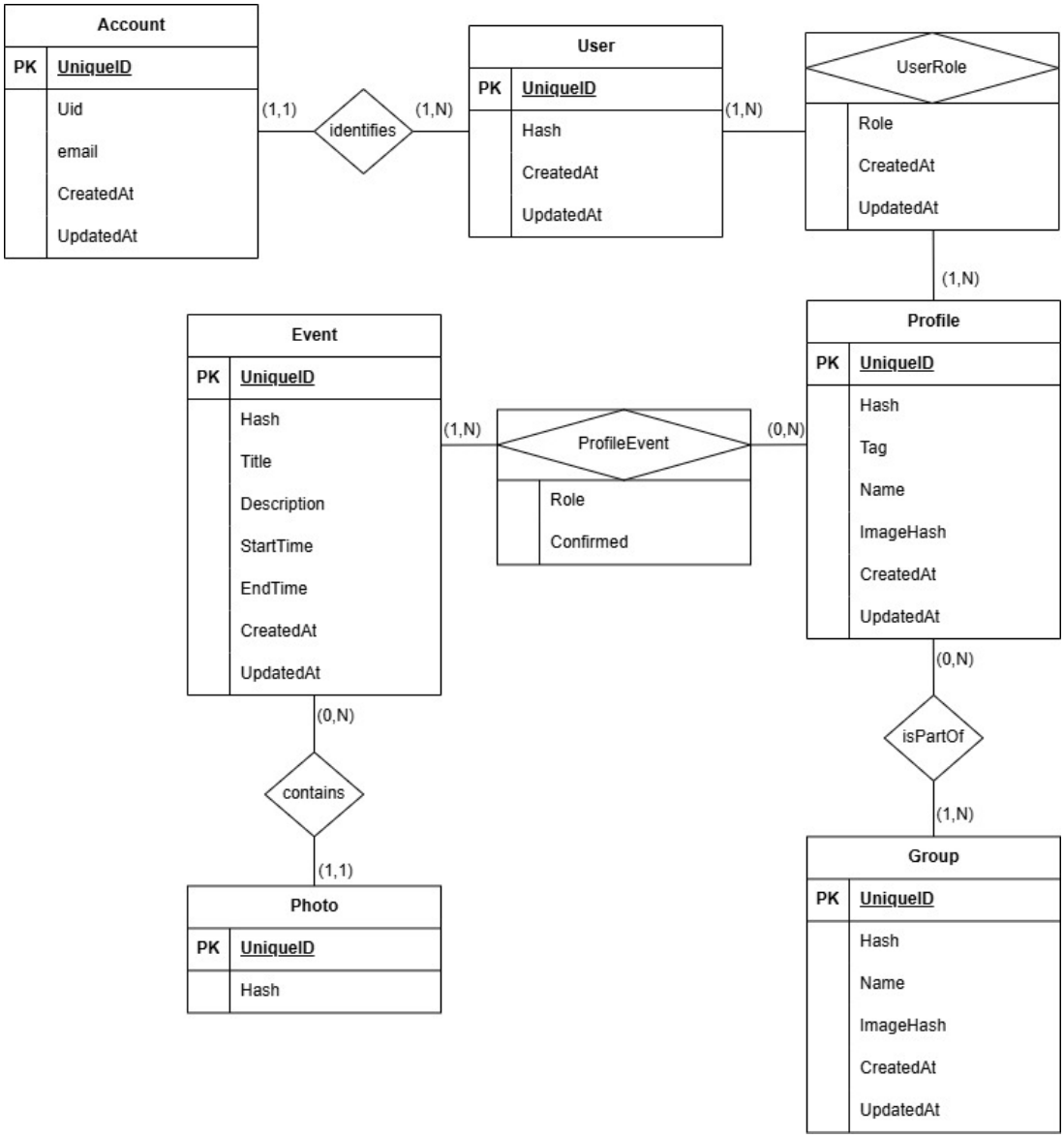


Diagramma di Sequenza: condividi Evento ai Gruppi TODO
Diagramma di Sequenza: Cerca profili TODO
Diagramma di Sequenza: aggiorna TODO

2.3 Progettazione della persistenza

Diagramma E-R



Come si può notare, il diagramma E-R della persistenza segue precisamente la struttura del modello del dominio mostrato precedentemente. La differenza sta nelle associazioni, che presumibilmente in fase di progettazione logica ed implementazione del database verranno concretizzate in classi di associazione, si avranno quindi due tabelle ulteriori (`ProfileEvent` e `UserRole`) per modellare le associazioni.

2.3.1 Formato dei file di log

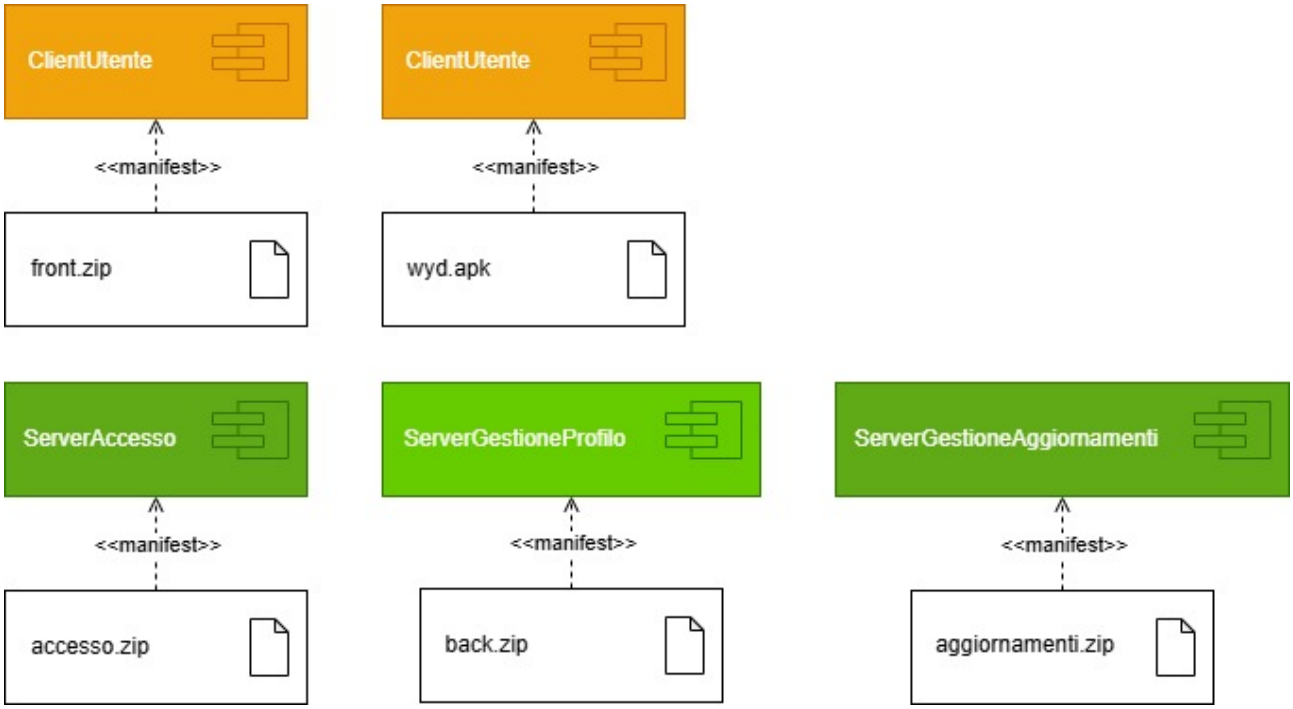
Il formato del file di log su cui il sistema terrà traccia delle operazioni sarà il seguente:

Esempio: File `/var/log/wyd.log`

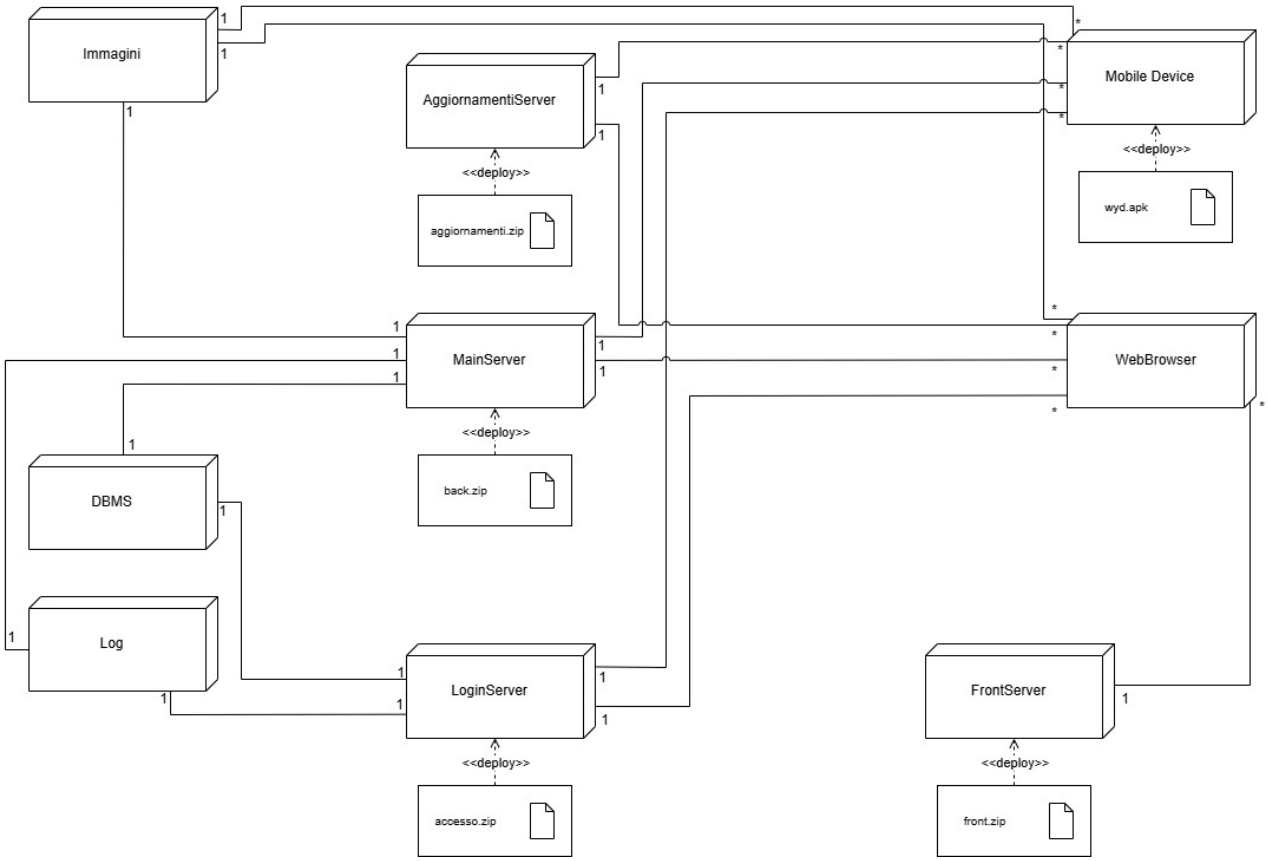
```
$ Data - Ora - Operazione - Descrizione - ID utente
```
Nota: l'ID utente è l'identificativo dell'esecutore di tale operazione.

2.4 Deployment

2.4.1 Artefatti



2.4.2 Deployment Type-Level



3 Implementazione

3.1 Progettazione Architetturale

3.1.1 Requisiti non funzionali

Per garantire la scalabilità e l'affidabilità del progetto useremo tecnologie in cloud, che permettono prestazioni altrimenti irraggiungibili, soprattutto nelle fasi iniziali dell'applicazione.
Questo comporta un sostanziale riduzione del carico di lavoro, in quanto al gestore cloud verranno delegati i requisiti di affidabilità, scalabilità, protezione da attacchi Ddos e la velocità delle comunicazioni

3.1.2 Scelte tecnologiche

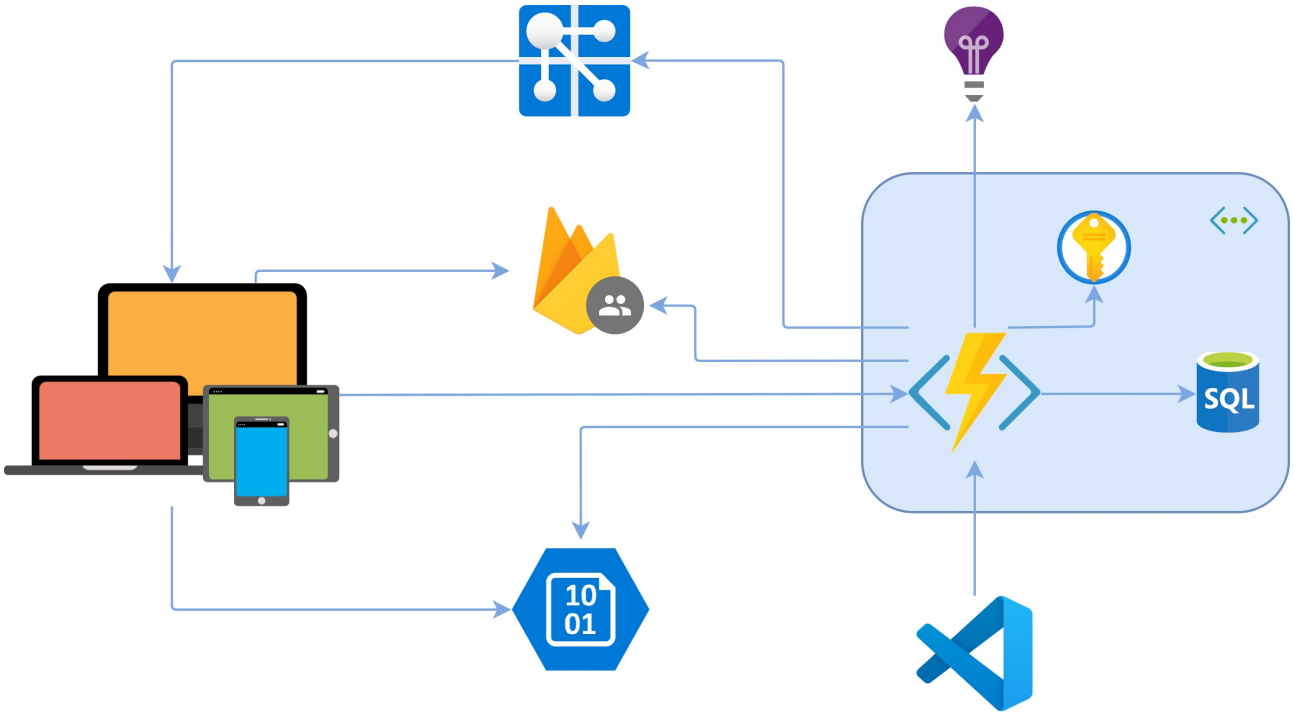
Per familiarità con il sistema, disponibilità dei fondi e servizi offerti, la scelta del gestore cloud ricade su:

- Google Firebase, per quanto la gestione degli accessi
- Microsoft Azure, per la logica applicativa e la persistenza.

In particolare, Google Firebase Authentication fornisce un sistema di autenticazione personalizzabile, oltre a dare la possibilità di gestire più identity providers.
La logica di business dell'applicazione verrà sviluppata su Azure Functions, un servizio Faas che scala automaticamente le richieste al server. La persistenza logica sarà affidata a AzureSQL, un database relazionale gestito dalla piattaforma, mentre le foto e i file multimediali verranno salvati su Azure Storage Container. Per aggiornare i devices in tempo reale utilizzeremo Azure PubSub, che permette di creare un canale con ogni dispositivo.
Infine, per la fruizione del servizio tramite browser useremo Azure Static Web App, e, in attesa del rilascio ufficiale sul Play Store dell'applicazione, l'applicazione mobile sarà resa disponibile tramite Azure Storage Container.
Per quanto riguarda lo sviluppo, i principali linguaggi di programmazione saranno:

- Flutter per l'interfaccia grafica, che permette di racchiudere in un'unica implementazione applicazioni per dispositivi di tecnologia differente
- C# per la logica di business, nativamente supportato dalle Functions, permette un collegamento diretto tra oggetti logici e database, semplificando la gestione della persistenza.

3.1.3 Scelta dell'architettura



Nel grafico possiamo notare come al centro dell'architettura ci siano le Azure Functions che, all'interno della stessa rete, comunicano con il database relazionale e il Key Vault. In base alla richiesta, che partirà sempre dal dispositivo dell'utente, le Functions potranno poi interagire con il servizio Pub-Sub per gli aggiornamenti in tempo reale, Google Firebase per l'autenticazione o Azure Storage Container per le immagini. I devices degli utenti avranno quindi la possibilità di contattare autonomamente, oltre alle Functions, sia Google Firebase che il server per la persistenza delle immagini.

3.2 Monitoraggio

Il monitoraggio del sistema avverrà in due modi: tramite salvataggio dei log e controllo delle prestazioni del sistema.
Per quanto riguarda Firebase Authentication vengono forniti in con il servizio sia le interfacce per il controllo delle prestazioni che la gestione dei log.
Per monitorare le Azure Functions sarà necessario collegare Azure Application Insights che provvede a controllare il funzionamento e la risposta del servizio. La creazione dei log è invece affidata al programmatore, che verranno poi salvarti tramite function su un file sistem dedicato su un Azure Container.

3.3 Sicurezza

Per garantire la sicurezza nelle comunicazioni ogni interazione avverrà utilizzando un canale sicuro cifrato grazie al protocollo HTTPS.
Tutte le credenziali e variabili di ambiente usate dalle Azure Functions per collegarsi ai database e ai vari servizi saranno conservate e ottenute tramite Azure Key Vault, per garantire la confidenzialità delle credenziali.

Il server di accesso rilascerà un token, unico per ogni utente, che verrà allegato ad ogni richiesta, per essere usato dal server principale per identificare l'utente ed eventualmente controllare i permessi relativi. Il token deve permettere di identificare ed autenticare univocamente l'utente.
La generazione degli hash di identificazione dei componenti deve garantire una diffusione tale da rendere altamente improbabile una collisione.
Le richieste di salvataggio dati non possono superare i 100 Kb di dimensione, e ogni utente non può fare in un arco orario richieste in scrittura che vadano complessivamente oltre i 20 MB.
Come evidenziato precedentemente, la difesa da attacchi di tipo denial of service verranno affidati al cloud provider
Inoltre, sarà necessario che tutto il codice, in fase di sviluppo, venga controllato da esperti di sicurezza, per ridurre il più possibile l'integrazione di falle nel sicurezza.

3.4 Progettazione di dettaglio

Di seguito verranno riportate solo le modifiche principali rispetto al piano di progettazione

3.4.1 Struttura

Struttura: Dominio

Sia per il client che per il server, ogni entità descritta nel dominio verrà mappata con una classe corrispondente.

Struttura: Controller - Client
Ogni richiesta di dati verso l'esterno verrà implementata grazie a api services, che nasconderanno i dettagli necessari per la comunicazione.
Inoltre, funzionalità che vengono ripetute in parti diverse del programma verranno delegate a classi chiamati servizi, che raggrupperanno le richieste per affinità logica.
A titolo esemplificativo, si riportano alcune classi:
Flutter ha una gestione dello stato che si basa sulla vita dei componenti a cui è associato. La persistenza a livello locale ha quindi durata limitata e ricade nella normale programmazione del framework.
La persistenza a livello applicazione verrà suddivisa in classi, logicamente in base al dominio. Le classi saranno singleton e verranno create all'avvio dell'applicazione.

Struttura: Controller - Server
Per sfruttare le caratteristiche di scalabilità delle Functions, il codice dovà essere suddiviso in nuclei di codice indipendenti tra loro.

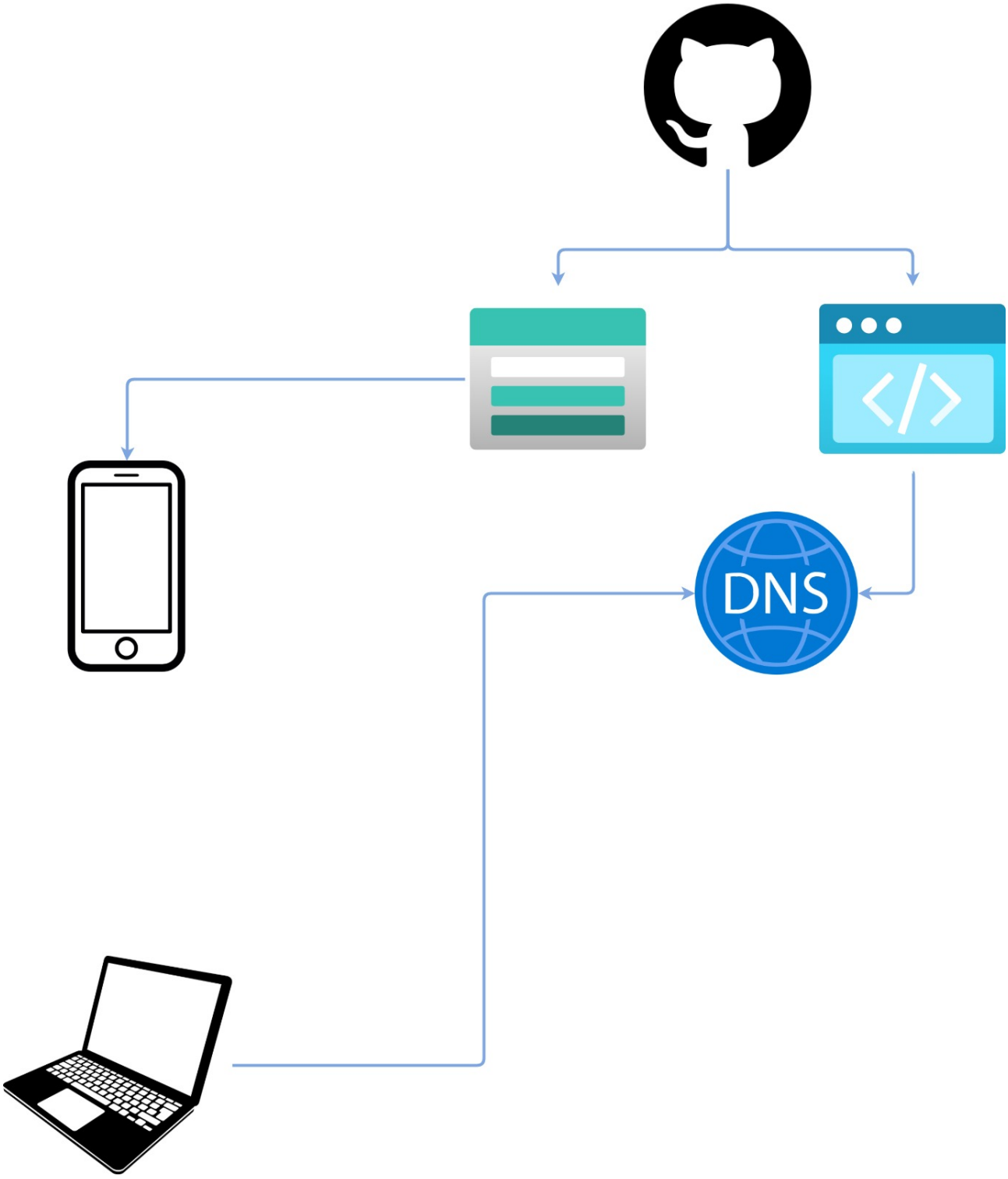
divisione responsabilità tra componenti: controller, service e API, DTO

3.4.2 Interazione

TODO controlla condivisione con gruppi e salvataggio immagini
condivisione tramite link

3.5 Deployment e Aggiornamento del Codice

3.5.1 Deployment Type-Level



Aggiornamento lato web gestito automaticamente, lato applicazione tramite notifica che chiederà all'utente di aggiornare la sua applicazione.

3.6 Testing e Performances