



ALMA MATER STUDIORUM  
UNIVERSITÀ DI BOLOGNA

DIPARTIMENTO DI INFORMATICA - SCIENZA e INGEGNERIA

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA  
INFORMATICA

Analisi, Progettazione e Distribuzione in  
Cloud di applicativo multipiattaforma  
per l'organizzazione di eventi condivisi e  
la condivisione multimediale automatica  
in tempo reale

Relatore:  
Chiar.mo Prof.  
Michele Colajanni

Presentata da:  
Giacomo Romanini

---

Sessione Marzo 2025

Anno Accademico 2024/2025

# Abstract

Lo sviluppo di un applicativo multipiattaforma diretto all’organizzazione di eventi condivisi, caratterizzato in particolare dalla condivisione multimediale in tempo reale, richiede opportune capacità di scalabilità, atte a garantire una risposta efficace anche con alti volumi di richieste, offrendo prestazioni ottimali. Le tecnologie cloud, con la loro disponibilità pressoché illimitata di risorse e la completa e continua garanzia di manutenzione, offrono l’architettura ideale per il supporto di simili progetti.

Tuttavia, l’integrazione tra la logica applicativa ed i molteplici servizi cloud, insieme alla gestione delle loro interazioni reciproche, comporta sfide specifiche, in particolare legate all’ottimizzazione di tutte le risorse. L’individuazione e la selezione delle soluzioni tecnologiche più adatte per ogni obiettivo, così come l’adozione delle migliori pratiche progettuali devono procedere parallelamente con lo sviluppo del codice, al fine di sfruttare efficacemente le potenzialità offerte.

In tale prospettiva, questa tesi illustra le scelte progettuali e implementative adottate nello sviluppo dell’applicativo in questione, evidenziando l’impatto dell’integrazione delle risorse cloud sul risultato finale.

# Indice

<b>Introduzione</b>	<b>1</b>
Organizzazione dei capitoli . . . . .	3
<b>1 L'Analisi delle funzionalità</b>	<b>4</b>
1.1 I requisiti e i casi d'uso . . . . .	4
1.1.1 I requisiti e il vocabolario . . . . .	4
1.1.2 I casi d'uso . . . . .	7
1.1.3 I requisiti di sicurezza . . . . .	13
1.2 L'analisi del problema . . . . .	18
1.2.1 Analisi delle funzionalità . . . . .	18
1.2.2 Architettura logica . . . . .	22
<b>2 Struttura centrale</b>	<b>25</b>
2.1 Lo sviluppo dell'applicazione utente . . . . .	28
2.1.1 Il framework di sviluppo . . . . .	29
2.1.2 Le interfacce grafiche . . . . .	30
2.1.3 La logica applicativa . . . . .	34
2.1.4 La distribuzione . . . . .	40
2.1.5 L'aggiornamento . . . . .	41
2.2 L'architettura del server principale . . . . .	42
2.2.1 La scelta del linguaggio . . . . .	43
2.2.2 La logica applicativa . . . . .	44
2.2.3 La distribuzione . . . . .	48
2.3 Autenticazione . . . . .	49
2.4 La sicurezza . . . . .	51
2.5 Monitoraggio . . . . .	52
<b>3 La persistenza</b>	<b>53</b>
3.1 Memoria principale . . . . .	55
3.1.1 I database e la scalabilità . . . . .	55

3.1.2	La gestione delle relazioni tra le entità . . . . .	57
3.1.3	Analisi del dominio . . . . .	58
3.1.4	La scelta del database . . . . .	61
3.1.5	Le limitazioni dei database relazionali . . . . .	64
3.1.6	L' integrazione con C# . . . . .	65
3.2	Cache Locale . . . . .	68
3.2.1	Creazione della memoria locale . . . . .	68
3.2.2	L'allineamento con la memoria centrale . . . . .	71
3.3	Aggiornamenti . . . . .	72
3.3.1	Scelta della tecnologia . . . . .	72
3.3.2	Integrazione . . . . .	74
<b>4</b>	<b>Gestione dei dati multimediali</b>	<b>77</b>
4.1	Recupero . . . . .	79
4.2	Salvataggio . . . . .	82
4.2.1	Persistenza . . . . .	83
4.2.2	Concorrenza . . . . .	86
<b>5</b>	<b>Risultati</b>	<b>88</b>
5.0.1	Velocità in lettura . . . . .	88
5.0.2	Caricamento di immagini concorrenti . . . . .	90
<b>6</b>	<b>Conclusione</b>	<b>91</b>
6.1	Sviluppi futuri . . . . .	92
<b>7</b>	<b>Fonti bibliografiche e sitografia</b>	<b>93</b>

# Introduzione

In un contesto sociale sempre più connesso, la crescente quantità di contatti, la rapidità delle comunicazioni e l'accesso universale alle informazioni rendono la ricerca, l'organizzazione e la partecipazione ad eventi estremamente facile, ma al contempo generano un ambiente frenetico e spesso dispersivo.

Risulta infatti difficile seguire tutte le opportunità a cui si potrebbe partecipare, considerando le numerose occasioni che si presentano quotidianamente. Basti pensare, ad esempio, alle riunioni di lavoro, alle serate con amici, agli appuntamenti informali per un caffè, ma anche a eventi più strutturati come fiere, convention aziendali, concerti, partite sportive o mostre di artisti che visitano occasionalmente la città.

Questi eventi possono sovrapporsi, causando dimenticanze o conflitti di pianificazione, con il rischio di delusione o frustrazione. Quando si è invitati a un evento, può capitare di essere già impegnati, o di trovarsi in attesa di una conferma da parte di altri contatti. In questi casi, la gestione degli impegni diventa complessa: spesso si conferma la partecipazione senza considerare possibili sovrapposizioni, o dimenticandosi, per poi dover scegliere e disdire all'ultimo momento.

D'altra parte, anche quando si desidera proporre un evento, la ricerca di un'attività interessante può diventare un compito arduo, con la necessità di consultare numerosi profili social di locali e attività, senza avere inoltre la certezza che gli altri siano disponibili. Tali problemi si acuiscono ulteriormente quando si tratta di organizzare eventi di gruppo, dove bisogna allineare gli impegni di più persone.

In questo contesto, emergono la necessità e l'opportunità di sviluppare uno strumento che semplifichi la proposta e la gestione degli eventi, separando il momento della proposta da quello della conferma di partecipazione. In tal modo, gli utenti possono valutare la disponibilità degli altri prima di impegnarsi definitivamente, facilitando in contemporanea sia l'invito sia la partecipazione.

In risposta a tali richieste è nata Wyd, un'applicazione che permette ai clienti di organizzare i propri impegni, siano essi confermati oppure proposti. Essa permette anche di rendere più intuitiva la ricerca di eventi attraverso la creazione di uno spazio virtuale centralizzato dove gli utenti possano pubblicare e consultare tutti gli eventi disponibili, diminuendo l'eventualità di perderne qualcuno. La funzionalità chiave di questo progetto si fonda sull'idea di affiancare alla tradizionale agenda degli impegni certi un calendario separato, che mostri tutti gli eventi a cui si potrebbe partecipare.

Una volta confermata la partecipazione a un evento, questo verrà spostato automaticamente nell'agenda personale dell'utente. Gli eventi creati potranno essere condivisi con persone o gruppi, permettendo di visualizzare le conferme di partecipazione. Inoltre, considerando l'importanza della condivisione di contenuti multimediali, questo progetto prevede la possibilità di condividere foto e video con tutti i partecipanti all'evento, attraverso la generazione di link per applicazioni esterne o grazie all'ausilio di gruppi di profili. Al termine dell'evento, l'applicazione carica automaticamente le foto scattate durante l'evento, per allegarle a seguito della conferma dell'utente.



Figura 1: Il logo di Wyd

La realizzazione di un progetto come Wyd implica la risoluzione e la gestione di diverse problematiche tecniche. In primo luogo, garantire la persistenza dell'agenda dell'utente, che deve essere aggiornata e mantenuta in modo affidabile e coerente, tale da permettere un uso distribuito del servizio. Inoltre, la funzionalità di condivisione degli eventi richiede l'aggiornamento in tempo reale di tutti gli utenti coinvolti, al fine di rimanere sempre aggiornati. Infine, il caricamento e il salvataggio delle foto introducono la necessità di gestire richieste di archiviazione di dimensioni significative.

## Organizzazione dei capitoli

Il seguente elaborato è suddiviso in cinque capitoli.

Nel primo capitolo si affronta la fase di analisi delle funzionalità, durante la quale, partendo dall'idea astratta iniziale, si definiscono i requisiti e le necessità del sistema, per poi creare la struttura generale ad alto livello dell'applicazione.

Nel secondo capitolo si affrontano le principali scelte architetturali e di sviluppo che hanno portato a definire la struttura centrale dell'applicazione.

Il terzo capitolo osserva lo studio effettuato per gestire la memoria, in quanto fattore che più incide sulle prestazioni. Particolare attenzione è stata dedicata, infatti, a determinare le tecnologie e i metodi che meglio corrispondono alle esigenze derivate dal salvataggio e dall'interazione logica degli elementi.

Il quarto capitolo si concentra sulle scelte implementative adottate per l'inserimento le funzionalità legate alla gestione delle immagini, che, oltre ad introdurre problematiche impattanti sia sulle dimensioni delle richieste sia sull'integrazione con la persistenza, richiedono l'automatizzazione del recupero delle immagini.

Infine, nel quinto capitolo, verranno analizzati e discussi i risultati ottenuti testando il sistema.

# 1 L'Analisi delle funzionalità

La realizzazione di qualunque prodotto software inizia da una fase in cui, partendo dall'abstract del progetto, si analizzano i requisiti e le funzionalità da realizzare. L'obiettivo è arrivare ad una definizione delle proprietà e del comportamento desiderato nell'applicazione che sia concisa e condivisa col cliente, senza però entrare nel merito delle scelte implementative.

Solo a quel punto si può procedere con la progettazione del programma vero e proprio.

## 1.1 I requisiti e i casi d'uso

Lo studio dell'abstract del progetto porta all'individuazione e alla descrizione delle sue caratteristiche essenziali. In particolare, si distinguono i requisiti ed i casi d'uso. I requisiti formalizzano le funzionalità che l'applicazione deve fornire, sintetizzando e schematizzando le parti che descrivono del prodotto. I casi d'uso descrivono invece le interazioni previste tra l'utente e il sistema, suddividendo le funzionalità in azioni elementari.

### 1.1.1 I requisiti e il vocabolario

I requisiti devono risultare chiari e precisi per permettere di procedere alle fasi successive in maniera corretta e trasparente. Ogni requisito deve essere breve e puntuale, limitato ad un solo particolare desiderata, focalizzando una necessità specifica.

Si suddividono in funzionali o non funzionali in base alle loro caratteristiche.

I requisiti funzionali descrivono le funzionalità che il sistema deve fornire, mentre i requisiti non funzionali illustrano le caratteristiche che il sistema deve soddisfare per essere considerato valido.

Si noti come le caratteristiche di velocità e scalabilità vengono individuate ed introdotte fin da subito.

<b>ID</b>	<b>Requisiti</b>	<b>Tipo</b>
R1F	Registrazione di un account tramite l'interfaccia web	Funzionale
R2F	Identificazione attraverso mail univoca e password di almeno 6 caratteri	Funzionale
R3F	Visualizzazione degli eventi confermati	Funzionale
R4F	Visualizzazione degli eventi proposti	Funzionale
R5F	Creazione di un evento impostando almeno la data di inizio e quella di fine	Funzionale
R6F	La data di fine deve essere successiva alla data di inizio	Funzionale
R7F	Modifica di un evento	Funzionale
R8F	La conferma di un evento lo sposta negli eventi confermati	Funzionale
R9F	La disdetta di un evento lo sposta negli eventi proposti	Funzionale
R10F	Caricamento delle foto di un evento	Funzionale
R11F	Condivisione tramite link	Funzionale
R12F	Condivisione tramite gruppo o ad altri profili	Funzionale
R13F	Ricerca automatica delle foto sul dispositivo mobile	Funzionale
R14F	Conferma delle foto	Funzionale
R15F	Ricerca di altri profili	Funzionale
R16F	Creazione di un gruppo da due o più profili	Funzionale
R17F	Visualizzazione dei profili collegati	Funzionale
R18F	Creazione di un nuovo profilo	Funzionale
R19F	Cambio del profilo attualmente in uso	Funzionale
R20F	Aggiornamento in tempo reale delle modifiche agli eventi	Funzionale
R1NF	Per interagire l'utente deve essere autenticato	Non Funzionale
R2NF	Velocità di richiesta iniziale dei dati	Non Funzionale
R3NF	Semplicità e fluidità dell'interfaccia grafica	Non Funzionale
R4NF	Velocità in lettura e scrittura dei dati	Non Funzionale
R5NF	Velocità nella ricerca dei profili	Non Funzionale
R6NF	Scalabilità delle richieste	Non Funzionale

Tabella 1: Tabella dei requisiti di Wyd

Alla tabella dei requisiti si affianca quella del vocabolario, definendo i termini utilizzati nel progetto per allinearli definitivamente alle volontà del cliente.

<b>Voce</b>	<b>Definizione</b>	<b>Sinonimi</b>
Account	combinazione di mail e password che identifica un utente	
Utente	Persona che utilizza l'applicazione	
Profilo	Entità logica che raggruppa eventi e interazioni	
Profili collegati	Profili a cui l'utente può avere accesso	
Gruppo	Insieme di profili	
Evento	Azione(o previsione di azione) con una durata nel tempo	
Data e ora evento	Indicazione temporale del momento in cui avverrà l'azione	
Evento confermato	Evento a cui il profilo ha dato conferma di partecipazione	
Evento proposto	Evento a cui il profilo non ha dato conferma di partecipazione	Evento disdetto, evento condiviso
Email	Indirizzo di posta elettronica del cliente utilizzata anche per l'autenticazione	
Password	Codice alfanumerico di almeno 8 caratteri	
Credenziali	Insieme composto da email e password necessari per accedere al sistema	

Tabella 2: Vocabolario di Wyd

### 1.1.2 I casi d'uso

I casi d'uso descrivono le interazioni tra gli attori ed il sistema, suddividendo le funzionalità in azioni elementari. Si definiscono attori tutti gli elementi che compiono una parte attiva nei confronti del programma. Ogni attore può interagire con uno o più casi d'uso, ed ogni caso d'uso può essere relazionizzato con altri, definendo la loro relazione.

I casi d'uso possono essere collegati tra loro tramite rapporti di inclusione o estensione. Si dice che un caso include un altro se contiene il suo comportamento. Si dice invece che un caso d'uso ne estende un altro se il suo comportamento può essere inserito all'interno del secondo.

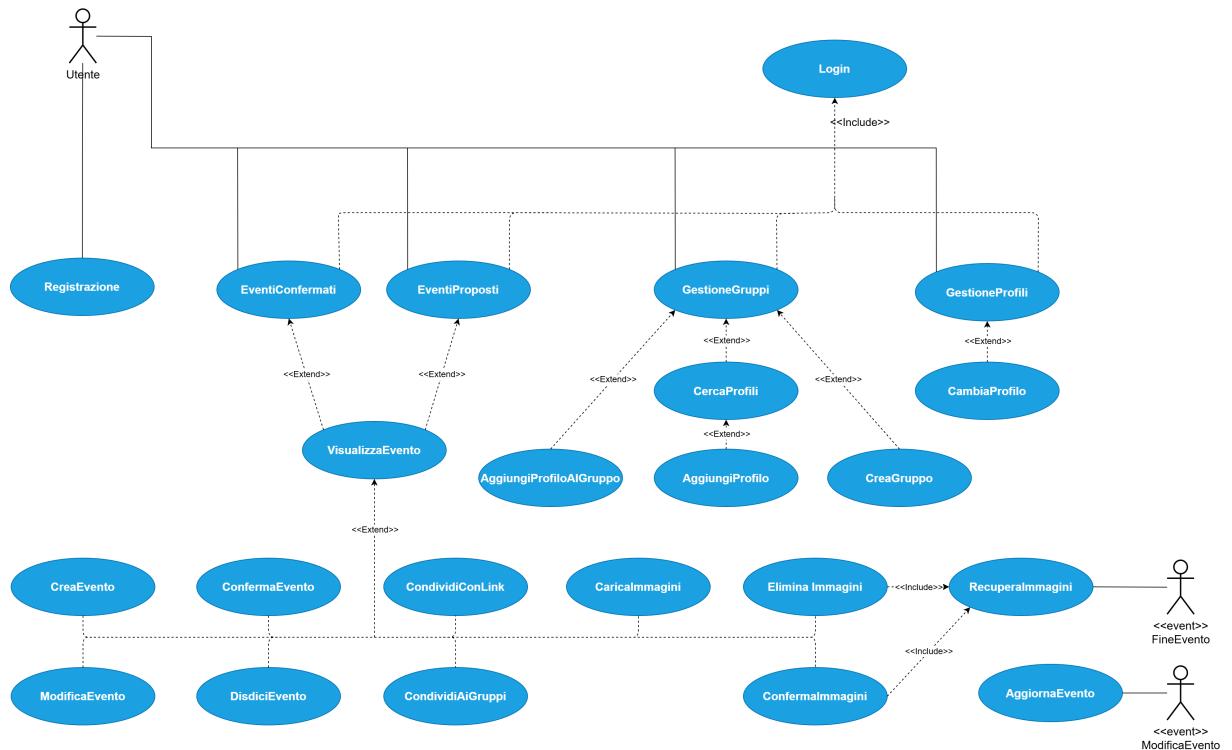


Figura 2: Diagramma dei casi d'uso

Per ogni caso d'uso viene identificato uno scenario di utilizzo, che chiarifica il contesto, il comportamento ed i punti critici dell'utilizzo. In particolare, si riportano gli scenari di utilizzo per i principali casi d'uso di Wyd, ovvero quelli che più andranno ad impattare sulla struttura e sulle esigenze del progetto.

Lo scenario di registrazione vede la responsabilità, oltre che di creare un account, di collegare un profilo all'utente. Questa separazione consente di avere una struttura gerarchica che permette di associare più profili ad un'unico utente, che può così in seguito crearne o unirne di nuovi.

<b>Titolo</b>	Registrazione
<b>Descrizione</b>	L'utente si registra al servizio
<b>Attori</b>	Utente
<b>Relazioni</b>	
<b>Precondizioni</b>	
<b>Postcondizioni</b>	L'utente è registrato nel sistema e può interagire con il resto dell'applicazione
<b>Scenario principale</b>	<ol style="list-style-type: none"> <li>1. L'utente accede alla schermata di registrazione</li> <li>2. L'utente inserisce email e password</li> <li>3. Il sistema crea un account con le credenziali inserite, associando un utente ed un primo profilo</li> <li>4. L'utente termina la registrazione, se avvenuta con successo viene reindirizzato alla pagina principale</li> </ol>
<b>Scenari Alternativi</b>	Il sistema verifica che è già presente un account con la mail inserita, quindi procede con la procedura di login normale.
<b>Requisiti non funzionali</b>	Per interagire l'utente deve essere autenticato Velocità in lettura e scrittura dei dati
<b>Punti aperti</b>	

Tabella 3: Scenario di registrazione

A seguito della modifica di un evento, che implica il salvataggio dei suoi nuovi dati, viene chiesto l'aggiornamento in tempo reale verso tutte le parti interessate. La modifica dei dati necessita inoltre un controllo sulle richieste che avvengono in contemporanea per evitare conflitti.

<b>Titolo</b>	ModificaEvento
<b>Descrizione</b>	Salva le modifiche ad un evento
<b>Attori</b>	Utente
<b>Relazioni</b>	VisualizzaEvento
<b>Precondizioni</b>	L'evento esiste e sono stati modificati dei dati
<b>Postcondizioni</b>	Le modifiche vengono salvate e propagate a tutti i profili collegati
<b>Scenario Principale</b>	<ol style="list-style-type: none"> <li>1. VisualizzaEvento</li> <li>2. Il sistema controlla che i dati modificati siano corretti</li> <li>3. I cambiamenti vengono salvati</li> <li>4. Tutti i dispositivi collegati ai profili collegati all'evento visualizzano le immagini</li> </ol>
<b>Scenari Alternativi</b>	<ol style="list-style-type: none"> <li>2. Se i dati risultano sbagliati, il sistema notifica l'utente originario indicando l'errore</li> </ol>
<b>Requisiti non funzionali</b>	Velocità in lettura e scrittura dei dati Scalabilità delle richieste
<b>Punti aperti</b>	Le modifiche all'evento devono essere consistenti, soprattutto in caso di richieste simultanee

Tabella 4: Scenario della modifica di un evento

Il salvataggio delle immagini è un'operazione di particolare importanza vista la sua rilevanza nel coinvolgimento degli utenti nell'utilizzo delle funzionalità centrali dell'applicazione, e quindi nel successo del progetto. Oltre a mostrare un'interfaccia intuitiva, il sistema deve essere in grado di gestire le richieste di caricamento, che generalmente richiedono più tempo e memoria. Prevedendo che la maggior parte di queste avvenga in seguito alla conclusione dell'evento, la probabilità che più richieste simultanee vertano sul-

lo stesso evento risulta elevata, creando la necessità di una gestione parallela di modifiche concorrenti.

<b>Titolo</b>	CaricaImmagini
<b>Descrizione</b>	Permette all'utente di selezionare immagini da collegare all'evento, salvandole
<b>Attori</b>	Utente
<b>Relazioni</b>	VisualizzaEvento
<b>Precondizioni</b>	L'evento esiste
<b>Postcondizioni</b>	Le immagini vengono salvate e propagate a tutti i profili collegati
<b>Scenario Principale</b>	<ol style="list-style-type: none"> <li>1. VisualizzaEvento</li> <li>2. L'utente seleziona le immagini che vuole caricare</li> <li>3. Le immagini vengono salvate</li> <li>4. Tutti i dispositivi relativi ai profili collegati all'evento visualizzano le immagini</li> </ol>
<b>Scenari Alternativi</b>	<p>Scenario alternativo A:</p> <ol style="list-style-type: none"> <li>3. Almeno una delle immagini crea problemi di lettura, l'utente viene notificato e può riprovare a caricare le immagini</li> </ol> <p>Scenario alternativo B:</p> <ol style="list-style-type: none"> <li>3. Solo una parte delle immagini vengono salvate, altre comportano errori</li> <li>4. l'utente viene notificato dell'errore e può riprovare a caricare le immagini</li> <li>5. Tutti i dispositivi relativi ai profili collegati all'evento visualizzano le immagini</li> </ol> <p>Scenario alternativo C:</p> <ol style="list-style-type: none"> <li>3. Nessuna immagine risulta salvata con successo</li> <li>4. l'utente viene notificato dell'errore e può riprovare</li> </ol>
<b>Requisiti non funzionali</b>	<p>Semplicità e fluidità dell'interfaccia grafica</p> <p>Velocità in lettura e scrittura dei dati</p> <p>Scalabilità delle richieste</p>
<b>Punti aperti</b>	

Tabella 5: Scenario del caricamento delle immagini

L’azione di recupero delle immagini facilita l’utilizzo dell’applicazione semplificando il procedimento di ricerca delle immagini, richiedendo all’utente solo la loro conferma. La sua corretta implementazione ne fa apprezzare l’utilità, con una significativa influenza sull’esperienza utente. Richiede però la pianificazione e l’automazione del processo di certifica di dati, con effetti sull’analisi tecnologica, sui processi in background e sulla gestione della memoria locale.

<b>Titolo</b>	RecuperaImmagini
<b>Descrizione</b>	L’applicazione controlla la galleria e salva in locale le foto scattate durante l’evento
<b>Attori</b>	FineEvento
<b>Relazioni</b>	EliminaImmagini, ConfermaImmagini
<b>Precondizioni</b>	L’evento esiste ed è concluso l’utente ha dato il permesso all’accesso alla galleria
<b>Postcondizioni</b>	Le immagini sono salvate in locale e l’utente viene notificato
<b>Scenario Principale</b>	<ol style="list-style-type: none"> <li>1. Il sistema attende la fine dell’evento</li> <li>2. Il sistema controlla la galleria per trovare le immagini scattate nell’arco temporale dell’evento</li> <li>3. Se ci sono immagini, vengono salvate in locale e l’utente viene notificato</li> </ol>
<b>Scenari Alternativi</b>	
<b>Requisiti non funzionali</b>	Velocità in lettura e scrittura dei dati
<b>Punti aperti</b>	L’implementazione dipende dal dispositivo su cui viene eseguita l’applicazione, alcuni dispositivi potrebbero non permetterne l’esecuzione

Tabella 6: Scenario di recupero delle immagini dal dispositivo dell’utente

### 1.1.3 I requisiti di sicurezza

Ogni sistema è esposto a vulnerabilità che impattano sul corretto funzionamento dell'applicazione e possono comportare disservizi in base alla loro rilevanza nel funzionamento del sistema. La definizione dei requisiti di sicurezza deriva dall'analisi del rischio. L'analisi del rischio individua i possibili vettori di attacco e serve ad orientare le risorse dove più necessario, tramite la valutazione dei beni, l'identificazione delle minacce e dei punti deboli noti nelle tecnologie.

La valutazione dei beni determina i componenti fondamentali da proteggere, risaltandone il valore e l'esposizione relativa. Questo permette di stabilire le priorità dei componenti sui cui concentrare le attenzioni.

Bene	Valore	Esposizione
Sistema Informativo	Alto. Fondamentale per il funzionamento del servizio	Alta. Perdita finanziaria e di immagine
Informazioni dei clienti	Alto. Informazioni personali	Alta. Perdita di immagine dovuta alla divulgazione di dati sensibili
Informazioni relativi agli eventi	Medio-alto, necessari per offrire il servizio e contenenti informazioni personali e potenzialmente riservate	Molto Alta. Perdita di immagine possibile con la divulgazione dei dati relativi ai clienti
Dati dei gruppi	Medio. Necessario per condividere gli eventi	Alta. Perdita di immagine

Tabella 7: Valutazione dei beni

La tabella delle minacce individua gli attacchi principali previsti che possono avvenire sul sistema. Esamina la loro probabilità, le azioni richieste per controllarli ed il costo di realizzazione delle contromisure necessarie. Fornisce quindi una prima analisi sulle necessità implementative.

<b>Minaccia</b>	<b>Probab.</b>	<b>Controllo</b>	<b>Fattibilità</b>
Furto credenziali utente	Alta	Controllo sulla sicurezza della password - Log delle operazioni, autenticazione a due fattori	Costo implementativo medio
Alterazione o intercettazione delle comunicazioni	Alta	Utilizzo di un canale sicuro - Log delle operazioni, autenticazione integrata nel messaggio	Basso costo di realizzazione con determinati protocolli
Accesso non autorizzato al database	Bassa	Accesso da macchine sicure - Log di tutte le operazioni	Basso costo di realizzazione, il server deve essere ben custodito
DoS	Bassa	Controllo e limitazione delle richieste	Media complessità di implementazione
Saturazione del database	Bassa	1. Limitazione delle richieste in un dato intervallo di tempo. 2. Limitazione della grandezza delle richieste singole 3. Limitazione della grandezza richiesta dallo stesso utente in un dato intervallo di tempo	Media complessità di implementazione

Tabella 8: Tabella delle minacce

L'analisi tecnologica della sicurezza entra nel merito delle tecnologie che si prevede necessarie. Per ognuna esamina i punti deboli e i limiti intrinseci, producendo un quadro delle particolarità su cui porre maggiore attenzione.

Tecnologia	Vulnerabilità
Autenticazione email/password	<ul style="list-style-type: none"> <li>• Utente rivela volontariamente la password</li> <li>• Utente rivela la password con un attacco di ingegneria sociale</li> <li>• Password banali</li> </ul>
Cifratura comunicazioni	<ul style="list-style-type: none"> <li>• In caso di cifratura simmetrica particolare attenzione va alla lunghezza delle chiavi ed alla loro memorizzazione</li> </ul>
Architettura Client/-Server	<ul style="list-style-type: none"> <li>• DoS</li> <li>• Man in the Middle</li> <li>• Sniffing delle comunicazioni</li> </ul>
Connessione Server/-Persistenza	<ul style="list-style-type: none"> <li>• Limite massimo di connessioni contemporanee</li> <li>• Saturazione del Database</li> </ul>

Tabella 9: Analisi tecnologica della sicurezza

Si prevedono quindi i principali attori malevoli e i relativi casi d'uso, per poi definire i requisiti su cui si baseranno le contromisure necessarie. I casi d'uso identificano le principali modalità di attacco, e ad ognuno ne viene corrisposto un altro che ne comporta la mitigazione. Vengono quindi integrati con i casi d'uso individuati in precedenza, evidenziando le necessità e la loro applicazione.

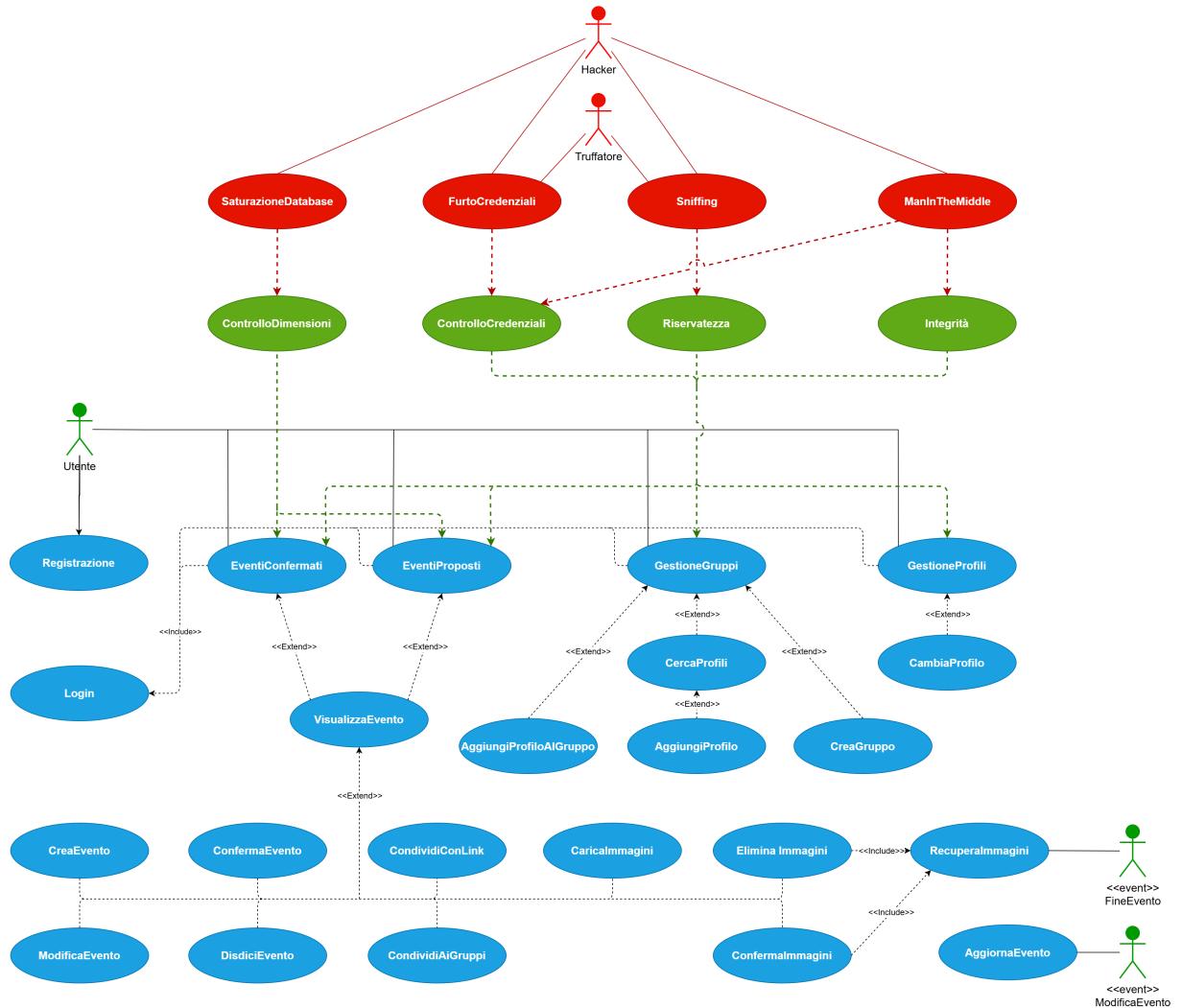


Figura 3: Casi d'uso relativi alla sicurezza

Visti i costi ed appurate le risorse a disposizione sussistono quindi i seguenti requisiti inerenti alla protezione dei dati e delle funzionalità di Wyd:

1. Implementare un sistema di log per tracciare tutti i messaggi tra i client e i server, inclusi gli accessi, le richieste di prenotazione, di conferma, di sospensione e di invio e ricezione di dati
2. I dati salvati devono essere protetti da un attaccante che abbia accesso al sistema, prendendo misure di sicurezza fisica, eventualmente cifrando i dati
3. I dati inviati tra le parti remote devono essere protetti, utilizzando la cifratura dei dati
4. Tutte le azioni avvenute sul sistema devono essere tracciate tramite un sistema di log.
5. Il sistema deve essere resistente ad un alto numero di richieste contemporanee
6. La dimensione delle richieste non deve superare una determinata soglia

La visione e l'analisi dei log verrà gestita con uno strumento esterno, accessibile solo al personale autorizzato.

ID	Requisiti	Tipo
R21F	Implementazione di un sistema di log per tracciare tutti i messaggi tra i client e i server	Funzionale
R22F	Le richieste non devono superare una certa dimensione	Funzionale
R7NF	I dati salvati devono essere protetti da un attaccante che abbia accesso al sistema, prendendo misure di sicurezza fisica, eventualmente cifrando i dati	Non Funzionale
R8NF	I dati inviati tra le parti remote devono essere protetti, utilizzando la cifratura dei dati	Non Funzionale
R9NF	Il sistema deve essere resistente ad un alto numero di richieste contemporanee	Non funzionale

Tabella 10: Requisiti di sicurezza

## 1.2 L'analisi del problema

A seguito dell'identificazione dei requisiti e dei casi d'uso, l'analisi del problema entra nel merito del comportamento dell'applicazione, evidenziandone il rapporto con le funzionalità. Determina quindi l'architettura logica, che delinea le relazioni fondamentali del sistema, individuando i componenti logici principali e le loro responsabilità.

### 1.2.1 Analisi delle funzionalità

Le funzionalità vengono dedotte dai casi d'uso, sintetizzando i servizi principali dell'applicazione. In particolare, le funzionalità vengono rilevate in base alla loro relazione con i casi d'uso, alla specificità del compito che assolvono e alla pertinenza reciproca.

Si riportano in tabella le funzionalità che racchiudono altri casi d'uso.

Funzionalità	Scomposizione
EventiConfermati	VisualizzaEvento
EventiProposti	VisualizzaEvento
VisualizzaEvento	CreaEvento, ModificaEvento, ConfermaEvento, DisdiciEvento, CondividiConLink, CondividiAiGruppi, CaricaImmagini, EliminaImmagini, ConfermaImmagini
GestioneGruppi	CercaProfili, AggiungiProfiloAlGruppo, CreaGruppo
CercaProfili	AggiungiProfilo
GestioneProfili	CambiaProfilo

Tabella 11: Scomposizione delle funzionalità

Di ogni funzionalità vengono evidenziati il grado di complessità, la tipologia di azione che svolgono ed i requisiti collegati. Il grado di complessità riassume la quantità e la difficoltà implementativa delle azioni che una funzionalità ricopre. La tipologia riporta in maniera generale la qualità dei servizi offerti. Infine si riportano gli identificatori dei requisiti funzionali che ogni funzionalità soddisfa.

<b>Funzionalità</b>	<b>Tipo</b>	<b>Grado di complessità</b>	<b>Requisiti Collegati</b>
Login	Interazione esterno e lettura dati	semplice	R2F
Registrazione	Interazione esterno e memorizzazione dati	semplice	R1F
EventiConfermati	Interazione esterno e gestione dati	complessa	R3F, R8F
EventiProposti	Interazione esterno e gestione dati	complessa	R4F, R9F
GestioneGruppi	Interazione esterno e gestione dati	complessa	R15F, R16F
GestioneProfili	Interazione esterno e gestione dati	complessa	R17F, R18F, R19F
VisualizzaEvento	Interazione esterno e gestione, lettura e memorizzazione dati	complessa	R5F, R6F, R7F, R8F, R9F, R10F, R11F, R12F, R14F
AggiornaEvento	Gestione dati	complessa	R20F
RecuperaImmagini	Lettura dati	complessa	R13F
ScritturaLog	Memorizzazione dati	semplice	R21F

Tabella 12: Funzionalità

Si procede analizzando i dati che ogni funzionalità gestisce, indicandone la tipologia, la protezione richiesta ed i vincoli correlati, per conoscere in maniera definitiva tutte le caratteristiche delle informazioni scambiate. A seguito dell’analisi delle informazioni, si procede con l’analisi dei vincoli, in cui si chiarificano i requisiti non funzionali, eviden-

ziandone le criticità e quali componenti ne vengono coinvolti.

<b>Requisito</b>	<b>Categorie</b>	<b>Impatto</b>	<b>Funzionalità</b>
Semplicità dell'interfaccia	Usabilità	Intuitività di utilizzo	Login, Registrazione, EventiConfermati, EventiProposti, GestioneGruppi, GestioneProfili, VisualizzaEvento, RecuperaImmagini
Velocità della ricerca dei dati	Tempo di Risposta	Maggiore reattività	EventiConfermati, EventiProposti, GestioneGruppi, GestioneProfili, RecuperaImmagini
Velocità di memorizzazione dei dati	Tempo di Risposta	Maggiore reattività	Registrazione, AggiornaEvento, RecuperaImmagini
Controllo Accessi	Sicurezza	Peggiorano tempo di risposta e usabilità, migliorano la privacy dei dati	EventiConfermati, EventiProposti, GestioneGruppi, GestioneProfili, VisualizzaEvento
Protezione dei Dati	Sicurezza	Peggiorano tempo di risposta, migliorano la privacy dei dati	Login, Registrazione, EventiConfermati, EventiProposti, GestioneGruppi, GestioneProfili, VisualizzaEvento, AggiornaEvento, RecuperaImmagini
Scalabilità delle richieste	Tempo di Risposta	Minor degrado-mento delle prestazioni	EventiConfermati, EventiProposti, AggiornaEvento, RecuperaImmagini

Tabella 13: Vincoli

Infine si definiscono logicamente le maschere, ovvero i componenti visuali essenziali del programma. Ad ogni maschera corrisponderà un’interfaccia grafica attraverso la quale l’utente potrà accedere alle funzionalità. Vengono quindi associate le maschere alle funzionalità di cui permettono l’esecuzione, indicando le informazioni relative.

<b>Maschera</b>	<b>Informazioni</b>	<b>Funzionalità</b>
View Login	email, password	Login
View Registrazione	email, password	Registrazione
View EventiConfermati	lista eventi confermati	EventiConfermati, AggiornaEvento
View EventiProposti	lista eventi proposti	EventiProposti, AggiornaEvento
View VisualizzaEvento	Identificativo utente, titolo, descrizione, data e orario di inizio, data e orario di fine, confermato, immagini, profili associati	VisualizzaEvento, RecuperaImmagini
View GestioneGruppi	lista gruppi	GestioneGruppi
View CercaProfili	tag di ricerca, lista profili	CercaProfili
View GestioneProfili	Lista profili, Identificativo utente, Identificativo profilo corrente	GestioneProfili

Tabella 14: Maschere

### 1.2.2 Architettura logica

Definite le relazioni e le informazioni relative alle funzionalità, si esprimono logicamente i componenti principali del sistema e le loro relazioni. Le funzionalità vengono espresse a livello logico tramite package e diagrammi delle classi, mentre i dati vengono descritti all'interno del dominio.

Il modello del dominio individua le entità che rappresentano logicamente le dipendenze tra i dati. Ogni entità presenta i suoi dati tramite proprietà, identificate da un nome e dalla tipologia del dato. Inoltre, all'interno del modello vengono indicati i rapporti tra le entità specificando le cardinalità reciproche.

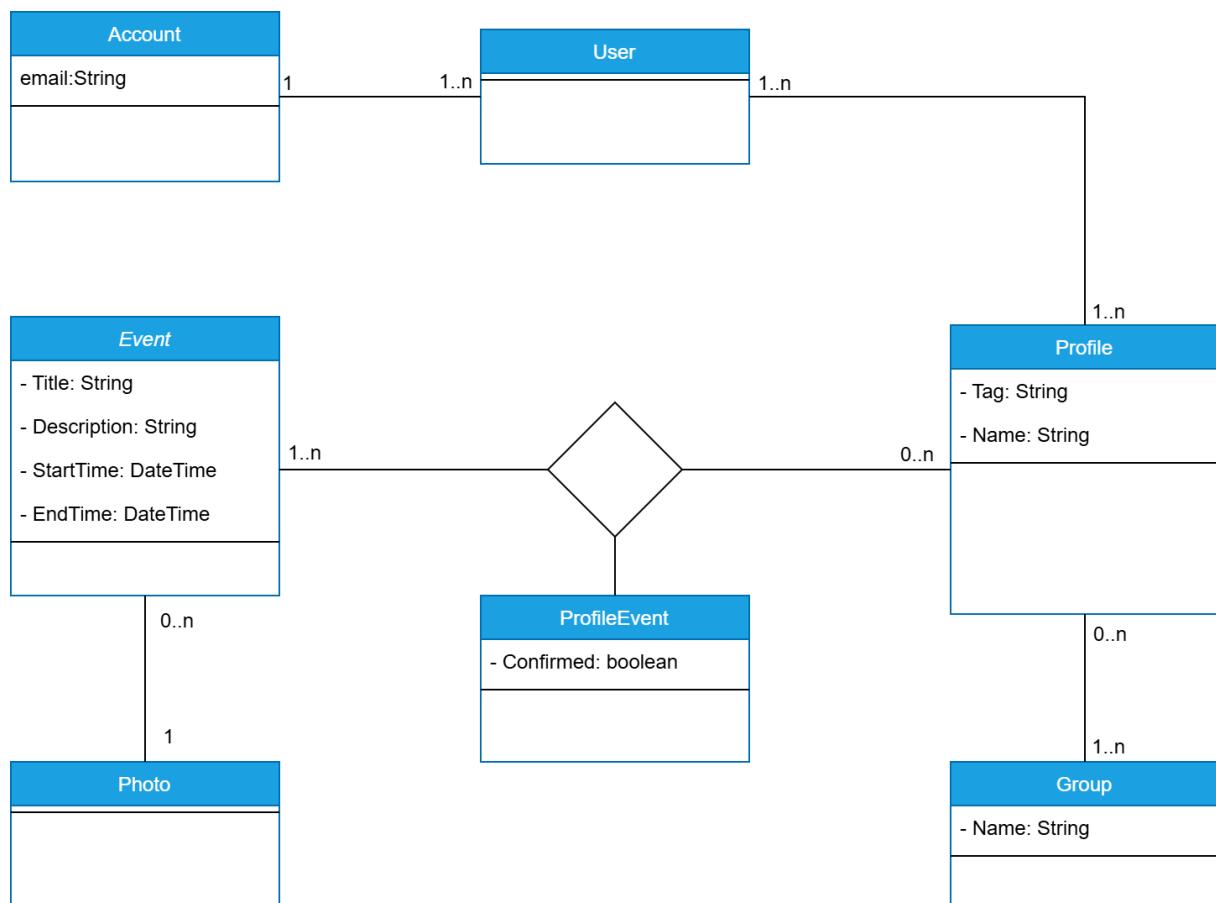


Figura 4: Modello del dominio

Il diagramma dei package descrive la divisione delle responsabilità logiche.

Ogni package rappresenta una parte di prodotto che soddisfa una determinata responsabilità. La responsabilità viene individuata in base alla peculiarità ed alle dipendenze delle funzionalità che ricopre. Si possono così distinguere, ad esempio, package relativi ad interfacce grafiche, logiche applicative o gestione della persistenza, in base alle caratteristiche specifiche del prodotto. Il diagramma dei package offre una prima struttura delle parti del progetto e del loro rapporto.

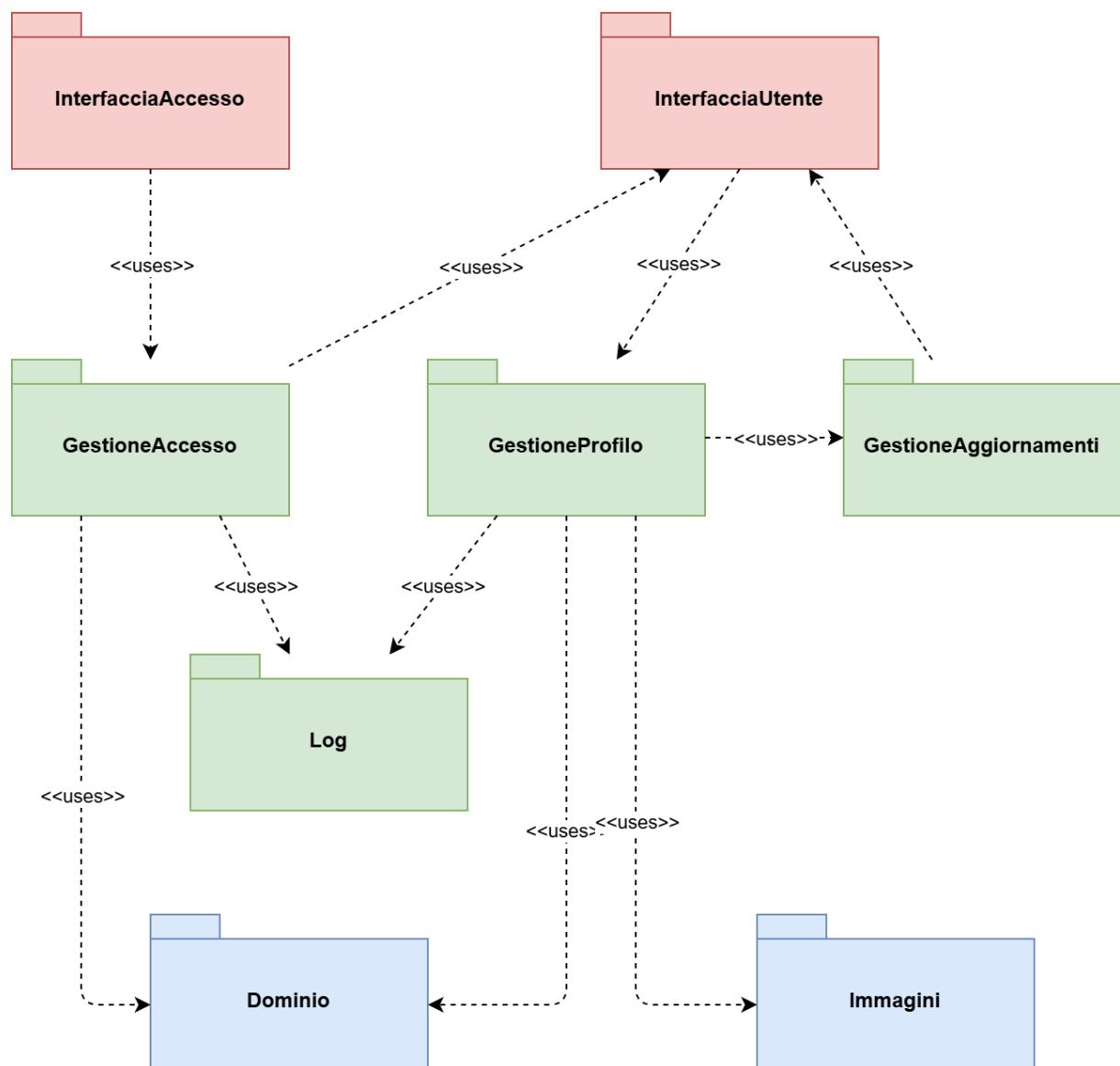


Figura 5: Diagramma dei Package

Ogni package contiene una o più classi che lo implementano. Ogni classe rappresenta un componente logico che assume uno specifico scopo. Le classi possono presentare dei metodi, ovvero delle istanze che descrivono le funzionalità fornite, alle quali altri componenti possono fare richiesta di esecuzione. La definizione delle classi permette di creare una struttura iniziale presentando le funzionalità minime e le dipendenze tra le parti.

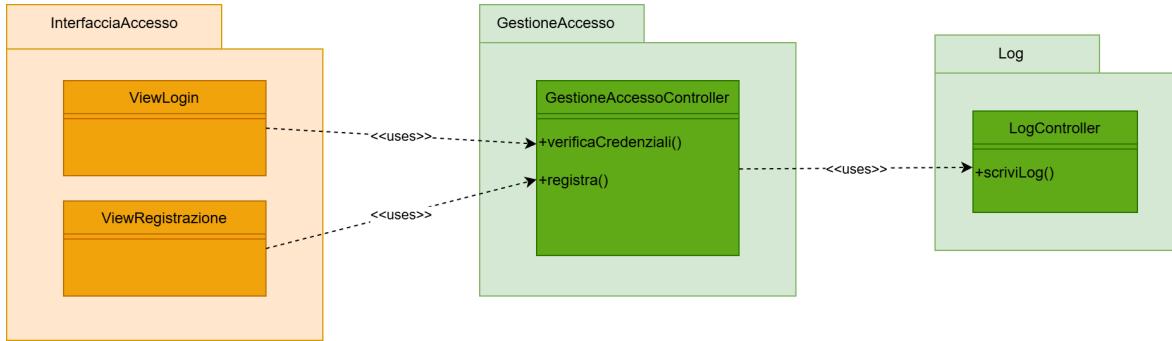


Figura 6: Diagramma delle classi: interfacce e gestione accesso

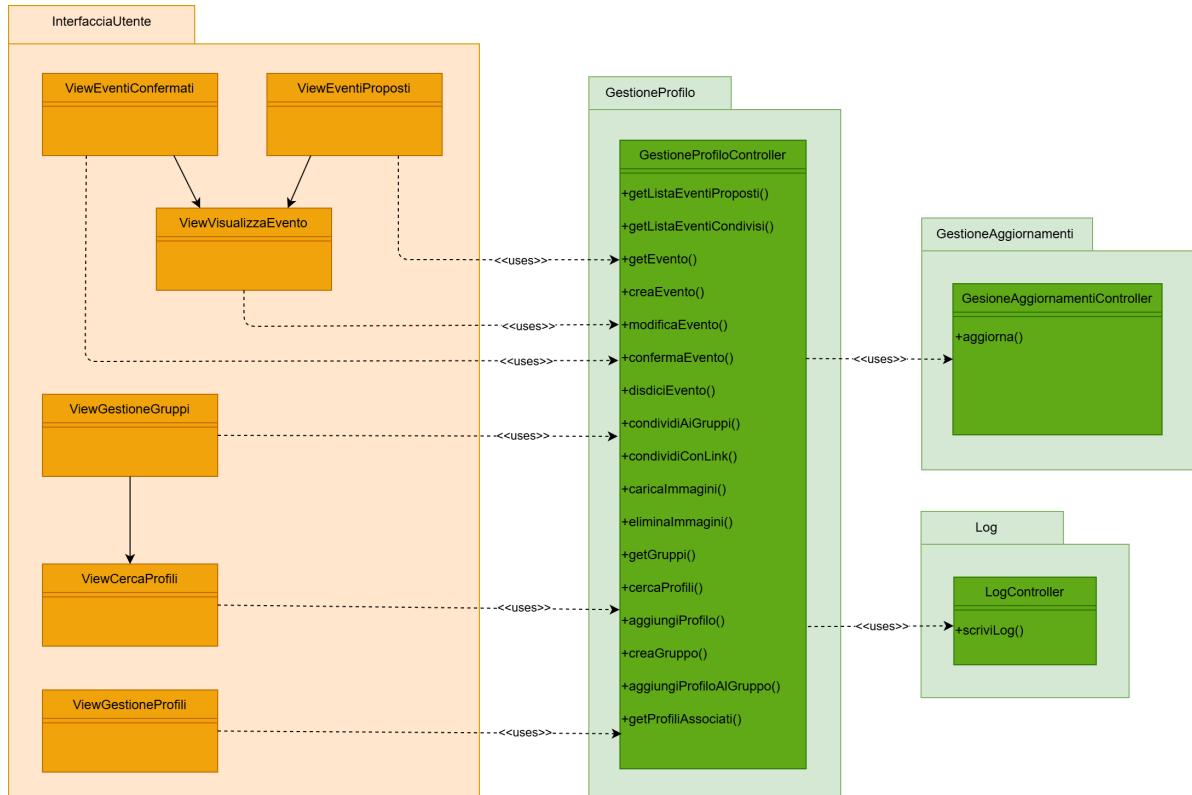


Figura 7: Diagramma delle classi: interfacce utente, gestione profilo ed aggiornamenti

## 2 Struttura centrale

Terminata l'analisi del problema, che ne ha stabilito i requisiti, le funzionalità e la struttura generale, si passa alla fase di progettazione. Durante questa fase l'obiettivo principale è identificare le caratteristiche funzionali e comportamentali del sistema, delineando le componenti principali e le rispettive responsabilità. Questo permette di definire un'architettura coerente, facilitando le successive scelte tecnologiche e implementative.

Nella fase successiva, di implementazione, si procede con il passaggio dall'analisi teorica alla realizzazione concreta, dove la selezione dell'architettura e delle tecnologie di riferimento assumono un ruolo centrale. Le decisioni prese in questa fase determinano il comportamento dei diversi componenti e le modalità con cui essi interagiscono tra loro. Un'attenta selezione delle soluzioni ottimali, più adatte ai requisiti definiti in fase di progettazione, permette di impostare fin dalle prime iterazioni uno sviluppo efficiente e strutturato, minimizzando la necessità di revisioni successive.

Nonostante alcune decisioni risultino immediate o intercambiabili, altre richiedono analisi approfondite per individuare la soluzione più adatta. Un approccio efficace consiste nello sviluppare inizialmente i componenti con requisiti ben definiti, per poi affinare progressivamente l'integrazione e la configurazione con gli altri elementi del sistema. L'identificazione, anche parziale, di una struttura iniziale consente di delineare i vincoli di integrazione e di semplificare la definizione delle soluzioni residue.

L'architettura dell'applicativo si basa su una chiara suddivisione in componenti, ciascuno con un ruolo specifico all'interno del sistema.

Tale organizzazione modulare consente di ottimizzare la scalabilità e la manutenibilità dell'applicativo, facilitando eventuali evoluzioni future. La suddivisione chiara delle responsabilità, unita a un'architettura flessibile e sicura, rappresenta quindi un elemento chiave per garantire la stabilità e l'efficienza del sistema nel lungo periodo.

L'interfaccia grafica è responsabile della presentazione e dell'interazione con l'utente, ponendo particolare attenzione alla coerenza visiva e alla fluidità dell'esperienza. La logica applicativa sarà gestita da un server dedicato, il quale si occupa di coordinare le comunicazioni tra i diversi servizi e di garantire il corretto flusso delle operazioni. La gestione dell'autenticazione degli utenti verrà separata dal resto del sistema, delegando questa responsabilità ad un servizio apposito, per migliorare sia le prestazioni che la sicurezza.

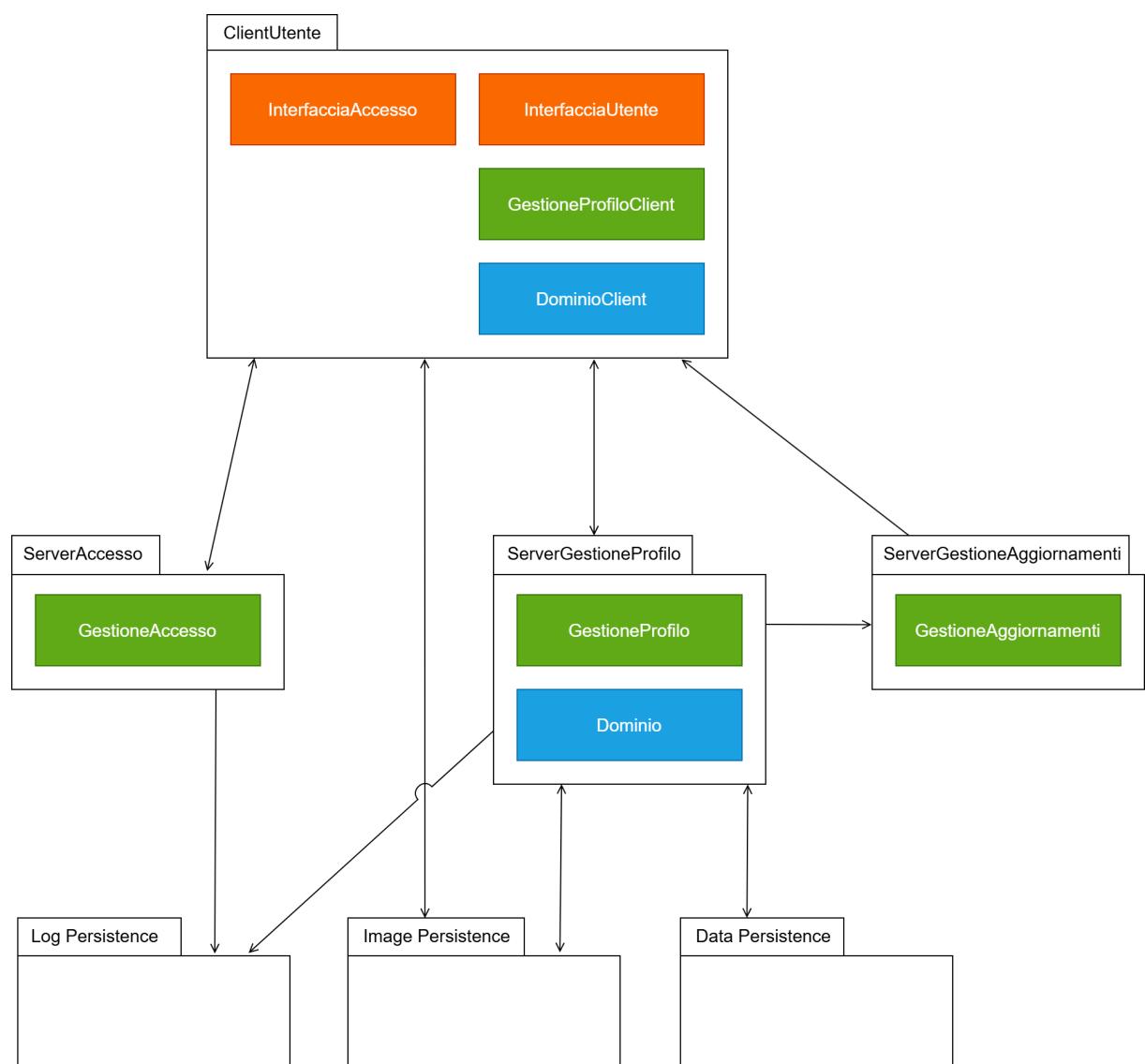


Figura 8: Struttura e responsabilità delle parti del progetto

Un ulteriore aspetto fondamentale nella progettazione del sistema riguarda la protezione delle comunicazioni e dei dati sensibili. L'adozione di misure di sicurezza adeguate è essenziale per garantire la protezione delle informazioni scambiate tra i vari componenti e per ridurre i rischi derivanti da eventuali attacchi esterni. Inoltre, per monitorare il corretto funzionamento dell'applicazione e identificare tempestivamente eventuali anomalie, il sistema è integrato con strumenti di logging e analisi delle prestazioni.

## 2.1 Lo sviluppo dell'applicazione utente

L'utilizzo delle applicazioni per la gestione degli eventi può essere suddiviso in due fasi distinte, ciascuna con specifiche esigenze funzionali.

La prima fase riguarda la pianificazione a lungo termine e l'organizzazione degli impegni. In questa circostanza l'utente decide come distribuire il proprio tempo, pianificando attività e appuntamenti, e strutturando il proprio calendario nel modo più efficiente per le proprie necessità. La seconda fase riguarda invece la gestione degli eventi non ancora certi e definiti; ciò include l'invito a un evento, l'eventuale conferma da parte dell'utente, l'identificazione degli impegni a breve termine e l'aggiornamento del loro stato (ad esempio, se l'evento sia ancora confermato, quante persone vi partecipano, se qualcuno ha annullato o se l'evento è già concluso) con la gestione degli eventuali contenuti multimediali successivi all'evento. Queste due fasi implicano un approccio diverso da parte dell'utente, comportando di conseguenza esigenze differenti a cui l'applicazione deve rispondere adeguatamente.

Per rispondere a tali necessità, è fondamentale che l'applicazione offra un'interfaccia utente versatile, fruibile sia da desktop che da dispositivi mobili. La versione desktop consente una pianificazione a lungo termine, offrendo una visione d'insieme chiara e completa di tutti gli impegni, tale da facilitare la gestione del tempo. D'altra parte, la versione mobile deve permettere una gestione rapida e dinamica degli eventi quotidiani, garantendo che l'utente possa rimanere sempre connesso e aggiornato sugli sviluppi in tempo reale.

Inoltre, considerando che l'applicazione è destinata a un utilizzo diffuso e a un'utenza potenzialmente elevata, è necessario garantire tempi di risposta ridotti e una gestione efficiente delle richieste concorrenti. Ciò implica la progettazione di un sistema in grado di scalare facilmente, per supportare un ampio numero di utenti simultanei senza compromettere le prestazioni.

Già consolidata e affidabile, sin dalle prime fasi di sviluppo del progetto è stata adottata la piattaforma cloud Azure, per garantire un'infrastruttura solida e scalabile.

### **2.1.1 Il framework di sviluppo**

Al fine di ottenere tutte le prestazioni precedentemente elencate, la scelta è ricaduta sull'adozione del framework di sviluppo Flutter. Diversi fattori motivano tale decisione.

In primo luogo, l'architettura di Flutter si basa su un motore grafico indipendente dalla piattaforma di esecuzione, il che consente di ottenere elevate prestazioni e garantire un'esperienza utente uniforme su dispositivi diversi. In secondo luogo, Flutter adotta un approccio dichiarativo nella progettazione dell'interfaccia grafica, che facilita lo sviluppo di componenti reattivi attraverso un codice conciso, facilmente manutenibile.

Un ulteriore vantaggio di Flutter è rappresentato dalla crescente adozione nel settore, dalla solidità della community di sviluppo e dal supporto offerto da Google, che ne assicurano la stabilità, l'efficienza, la sicurezza e la disponibilità di componenti personalizzabili per l'intero ciclo di vita del prodotto. Infine, Flutter consente uno sviluppo rapido e interattivo grazie alla sua sintassi intuitiva e al meccanismo di hot reload, che riduce significativamente i tempi di compilazione e facilita il testing in tempo reale.

Tra le altre tecnologie valutate per lo sviluppo dell'interfaccia grafica, vi erano React Native e Xamarin. Tuttavia, entrambe presentano alcune limitazioni: le applicazioni finali sviluppate con React Native tendono ad avere dimensioni più elevate e le prestazioni risultano inferiori, in particolare nella gestione della memoria. Xamarin, pur essendo una valida opzione, presenta una curva di apprendimento più ripida e una comunità di sviluppatori ridotta rispetto a Flutter, con una conseguente minore disponibilità di componenti e librerie.

L'applicazione utente ha come obiettivo la soddisfazione di due compiti principali: interagire con l'utente e comunicare con il server, per recuperare i dati e salvare le modifiche apportate.

### 2.1.2 Le interfacce grafiche

L'interazione utente avviene tramite interfacce grafiche che permettono di visualizzare i dati e le funzionalità a disposizione. Per rispettare il requisito di semplicità e fluidità dell'esperienza è essenziale che ogni interfaccia sia il più intuitiva possibile, tramite una limitata varietà di azioni nella stessa pagina, ognuna delle quali facilmente accessibile, ma anche riconoscibile in base alla sua importanza e funzionalità.

Per ogni maschera individuata in fase di analisi corrisponde almeno un'interfaccia grafica che, oltre a gestire la navigazione con le altre interfacce, permette all'utente di eseguire le proprie funzionalità, esponendo chiaramente le informazioni, concentrando l'attenzione sui dati eventualmente richiesti e segnalando le azioni eseguibili.

Nei diagrammi di dettaglio le interfacce vengono presentate elencando le funzionalità di cui dispongono, assieme alle loro relazioni di dipendenza. Si riportano le interfacce di gestione dei gruppi e di visualizzazione degli eventi, in quanto funzionalità centrali, il cui stile grafico è stato rispettato nella creazione del resto dell'applicazione.

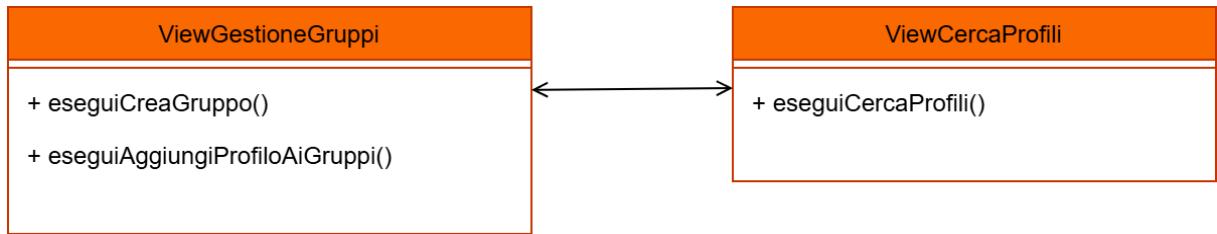
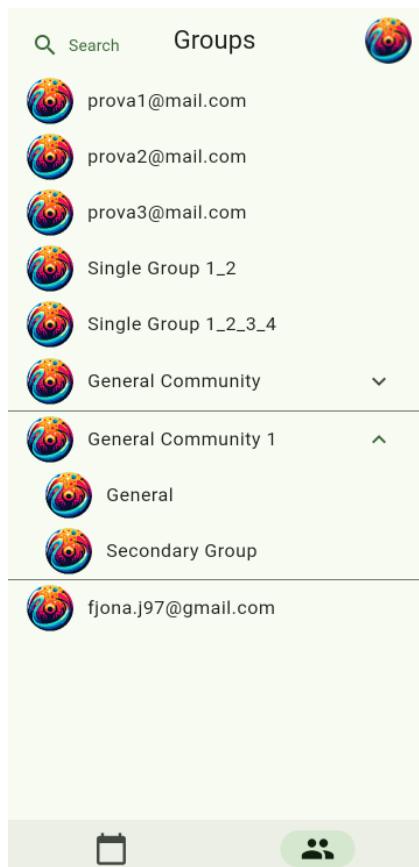


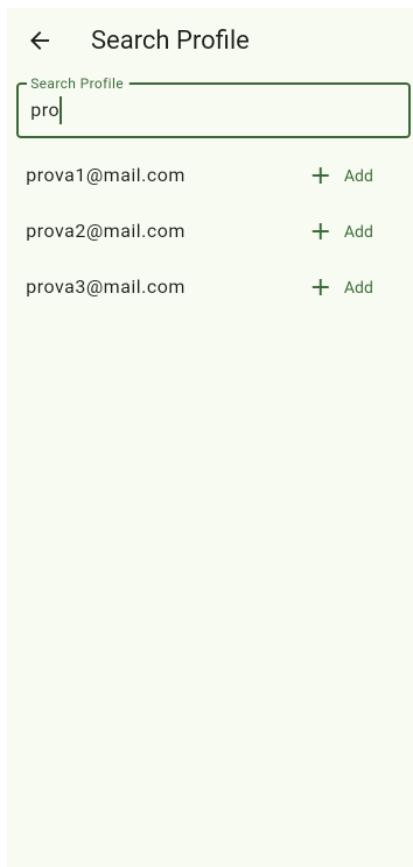
Figura 9: Diagramma di dettaglio delle interfacce di gestione dei gruppi

L'interfaccia della gestione dei gruppi ha il compito di presentare tutti i gruppi associati al profilo attualmente in uso, fornendo l'accesso alle azioni relative. Li elenca quindi in maniera chiara, definendo la differenza tra gruppi di due o più persone. Per ognuno mostra un bottone dal quale, se selezionato, compariranno le azioni attuabili sul gruppo (ad esempio, di aggiungere un profilo).

Correlata alla gestione dei profili c'è la loro ricerca, che consente il ritrovamento e la successiva aggiunta dei profili tra i propri gruppi. La schermata risulta minimale, consentendo all'utente di concentrarsi sulle sole informazioni e funzionalità essenziali.



(a) Elenco dei gruppi



(b) Ricerca dei profili

Figura 10: Schermate dei gruppi

La visualizzazione degli eventi prevede due componenti principali.

Il primo consiste in una panoramica generale, affiancando gli eventi tra loro a livello settimanale, per fornire all'utente un quadro complessivo degli impegni. Tale vista è ripetuta sia per gli eventi proposti che per quelli confermati, con la possibilità di navigare tra le due schermate.

Il secondo entra nel particolare dell'evento, mostrando i dettagli relativi e fornendo la possibilità di modificarli. Concentra inoltre le principali funzionalità dell'applicazione, quali la conferma della partecipazione all'evento, la condivisione con i gruppi e il caricamento delle immagini.

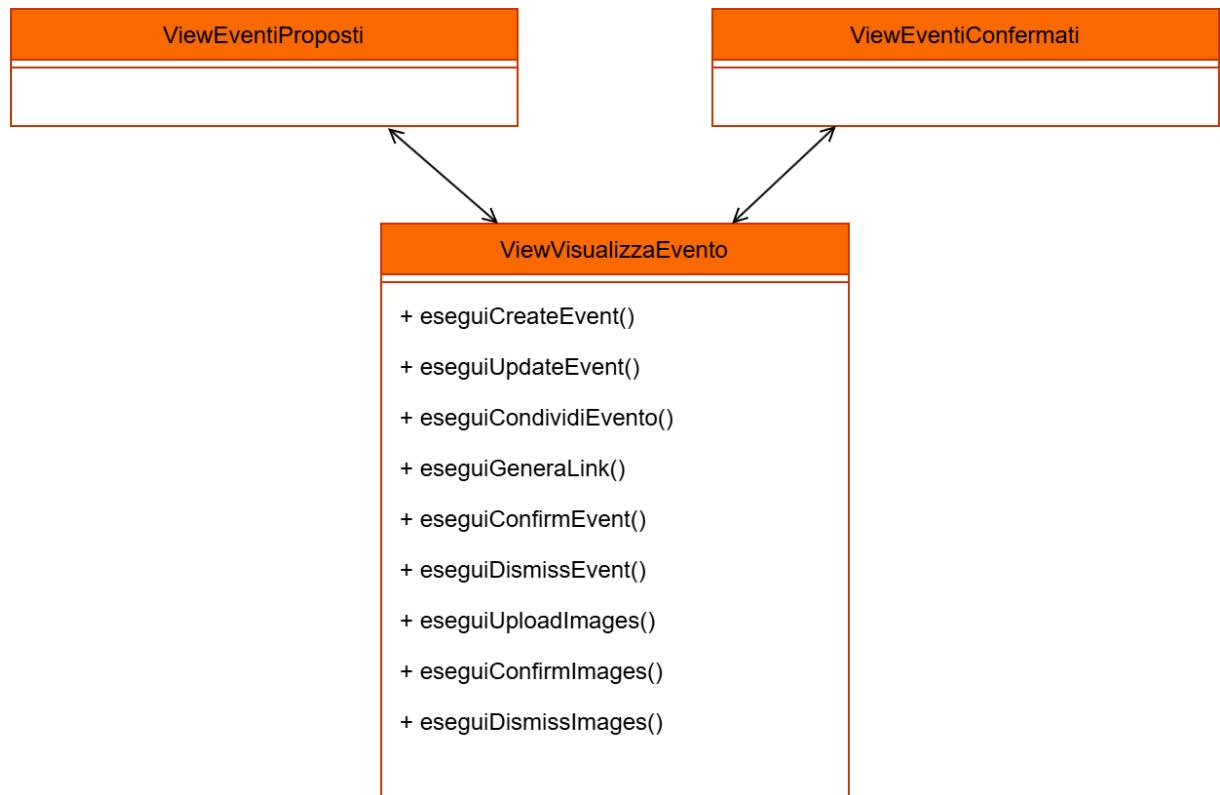
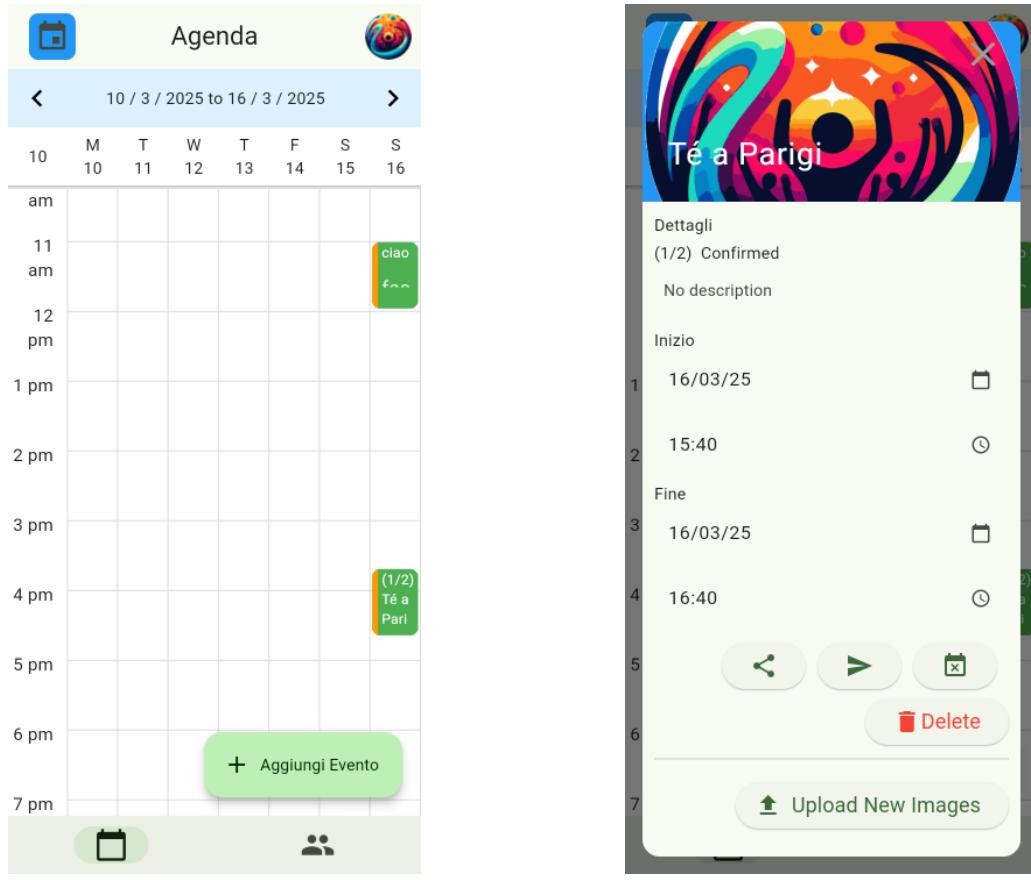


Figura 11: Diagramma di dettaglio delle interfacce di visualizzazione eventi

Queste due funzionalità sono al centro del servizio del sistema, ed è quindi essenziale che l'interfaccia proposta sia veloce ma soprattutto intuitiva. Fondamentale in questo riguardo è l'importanza data dai colori, attraverso i quali ogni elemento risalta in base alla sua importanza, e fornisce il suo contesto e le sue proprietà grazie al puro impatto visivo.



(a) Visualizzazione generale degli eventi

(b) Dettaglio di un evento

Figura 12: Schermate degli eventi

### **2.1.3 La logica applicativa**

Ogni interazione con l’utente scatena una qualche forma di elaborazione di dati. La visualizzazione di qualunque componente comporta il ritrovamento delle infomazioni, la loro modifica necessita di essere salvata e la loro condivisione esige la propagazione degli aggiornamenti. Inoltre, alcuni casi d’uso richiedono azioni da svolgere in autonomia. L’implementazione della logica necessaria, per rispondere efficacemente ai requisiti di velocità ed efficienza, avviene tramite la creazione di diversi componenti.

La suddivisione del programma individua e raggruppa le funzionalità in base al loro contesto, affidando ad ogni componente meno responsabilità possibili. Questo permette di concentrare le logiche condivise, evitando duplicazioni e definendo chiaramente il ruolo di ogni metodo. La semplicità del codice così raggiunta semplifica il futuro sviluppo e la sua manutenzione.

La principale suddivisione dei componenti avviene in base agli elementi del dominio. Per ogni principale entità, infatti, viene creato un servizio che ne racchiude le richieste di ritrovamento, modifica e salvataggio correlate. Collegando i servizi al dominio si concentrano anche le eventuali dipendenze da altri servizi, riducendole alle sole inerenti all’elemento specifico, mantenendo un parallelismo logico anche a livello di relazione.

La maggior parte delle richieste che riguardano gli elementi del dominio prevede la comunicazione con il server esterno. La ricezione e l’aggiornamento dei dati, così come la permanenza delle modifiche, avviene infatti attraverso l’interazione con la persistenza principale, a cui si può accedere tramite il server. Vista la complessità specifica nella creazione delle trasmissioni e la loro secondaria importanza a livello logico, vengono realizzati dei componenti dedicati, chiamati API. I componenti API permettono quindi l’astrazione delle trasmissioni con il server, semplificando il codice e separando la logica applicativa dalle complessità richieste dalla tecnologia dei protocolli usata.

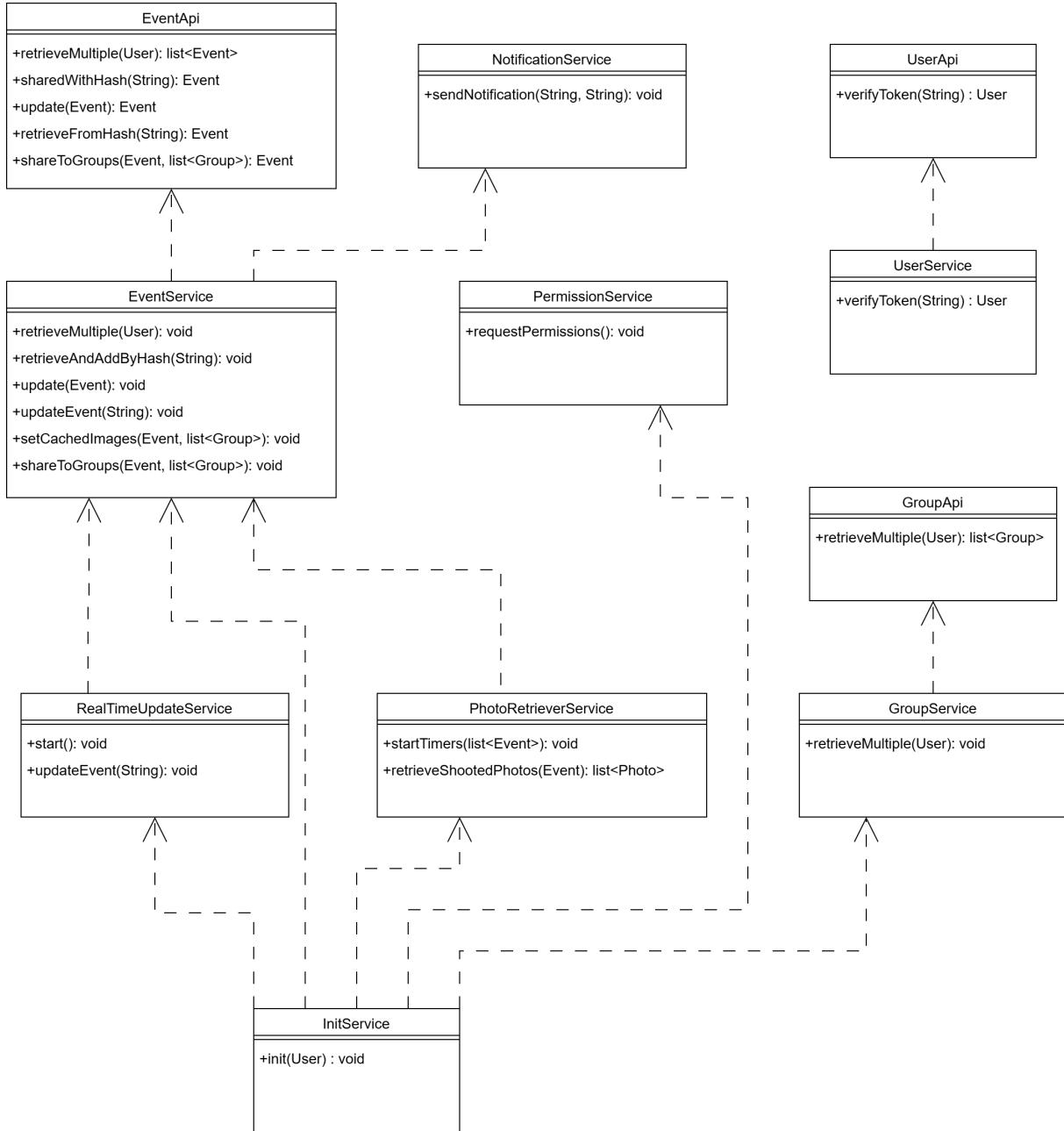


Figura 13: Modello delle classi del client

Non tutte le funzionalità sono però correlate direttamente al dominio. Per questo motivo si creano servizi ausiliari dedicati, anch'essi separati in base al ruolo che ricoprono.

Un servizio è stato dedicato all'acquisizione e al salvataggio dei permessi necessari per operare, quali l'invio delle notifiche e l'accesso alla galleria. Mette a disposizione degli altri processi, quindi, la conferma dell'accesso ai permessi richiesti o, in caso non lo si possedga, gestirà il suo ottenimento.

L'invio delle notifiche e la ricezione degli aggiornamenti, per quanto concettualmente simili e strettamente correlati, sono stati implementati in due componenti differenti. Le notifiche possono essere infatti richieste anche da altri metodi, e ad ogni aggiornamento potrebbe non corrispondere una notifica. La ricezione delle modifiche in tempo reale avviene usando il pattern observer, nel quale il servizio si connette ad un canale e rimane in attesa di eventuali messaggi.

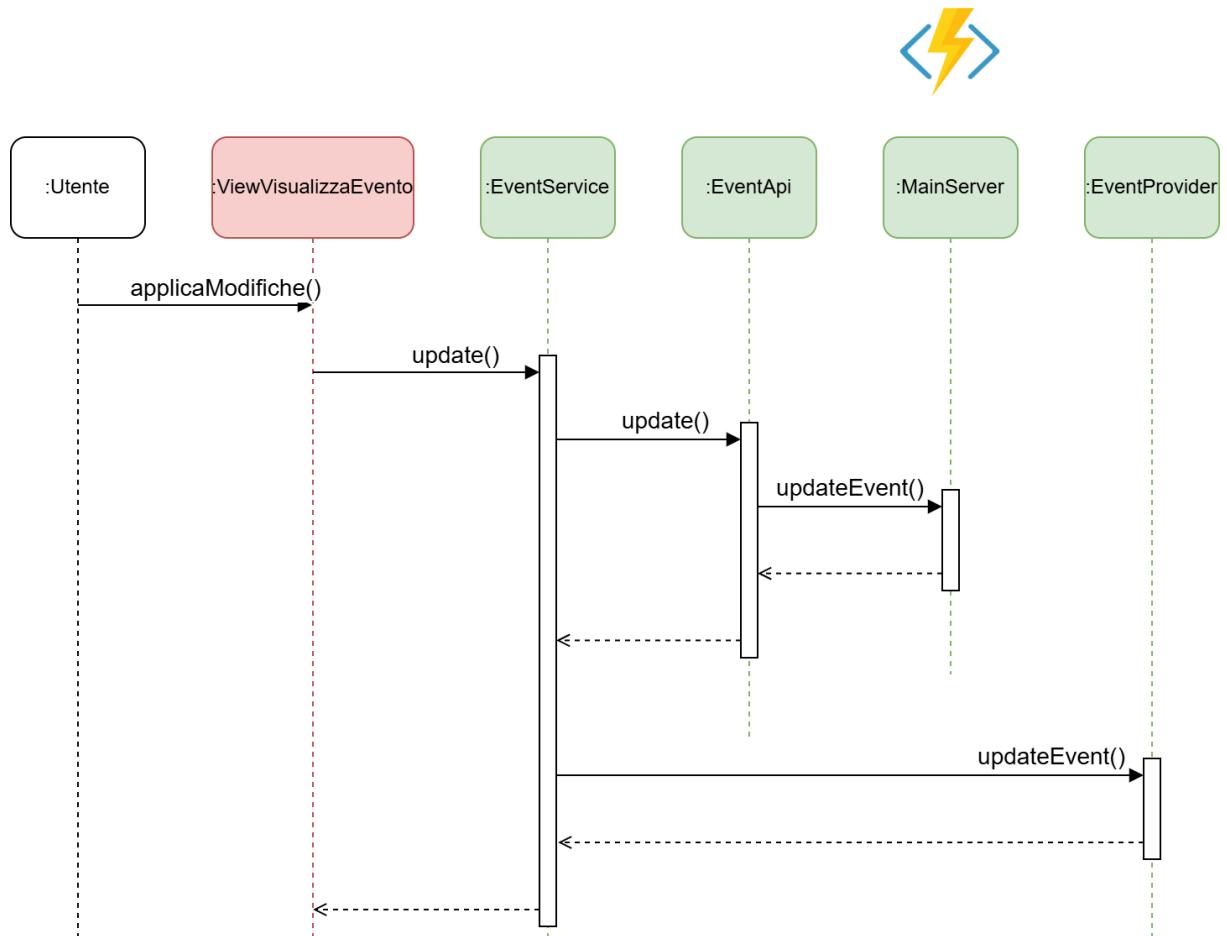


Figura 14: Diagramma di sequenza della modifica di un evento

Il salvataggio in memoria locale dei dati risulta fondamentale per la reattività dell'applicazione, in quanto permette di ridurre le richieste di dati e velocizza il loro recupero. La memoria locale viene implementata grazie a classi Provider, create in relazione agli elementi del dominio. Un'altra funzionalità centrale dell'applicazione è il recupero automatico delle foto scattate durante l'evento. Questo richiede la pianificazione di azioni automatiche nel tempo, così come la scansione della galleria per trovare le immagini interessate. Sia la gestione locale della memoria che il recupero delle immagini vedono uno o più componenti dedicati. La loro realizzazione viene trattata nei capitoli seguenti.

Durante l'implementazione della logica applicativa si sono dovuti affrontare altri problemi quali, degni di nota, la gestione della condivisione di un evento tramite link e l'inizializzazione dell'applicazione.

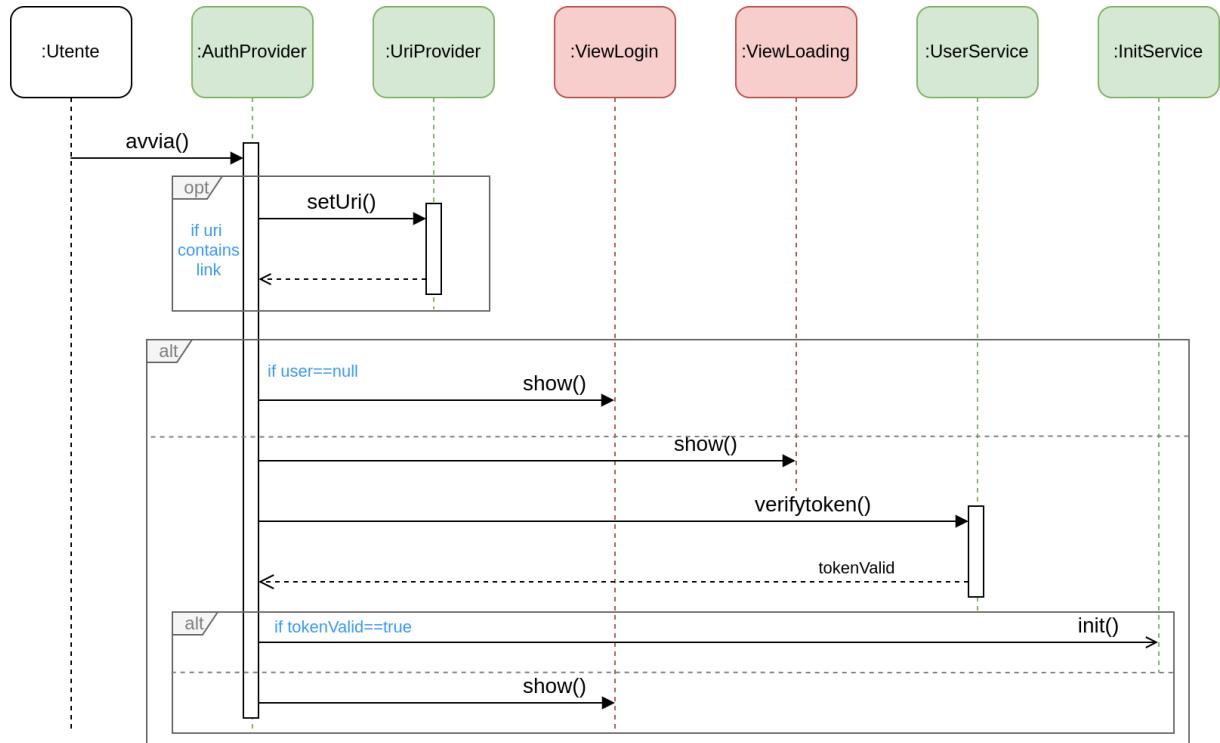


Figura 15: Diagramma di sequenza dell'avvio dell'applicazione

La condivisione di un evento tramite link consiste in due passaggi principali: la creazione del link stesso e il successivo ritrovamento dell'evento associato. La generazione del link avviene tramite l'unione del dominio del server con il codice identificativo dell'evento.

All'apertura dell'applicazione tramite link viene estratto il codice identificativo dell'evento per la successiva richiesta dei dati al server. Se l'utente non si è ancora autenticato, però, il router dell'applicazione lo reindirizza alla schermata di login, cambiando il link e perdendo l'informazione allegata. Per evitare questo problema, nel momento in cui l'utente accede all'applicazione tramite un link, le sue informazioni vengono salvate in memoria locale. Al termine del login, se sono presenti dati salvati, l'utente verrà indirizzato alla schermata degli eventi proposti, che recupererà i dati relativi all'evento, per poi mostrarli a video.

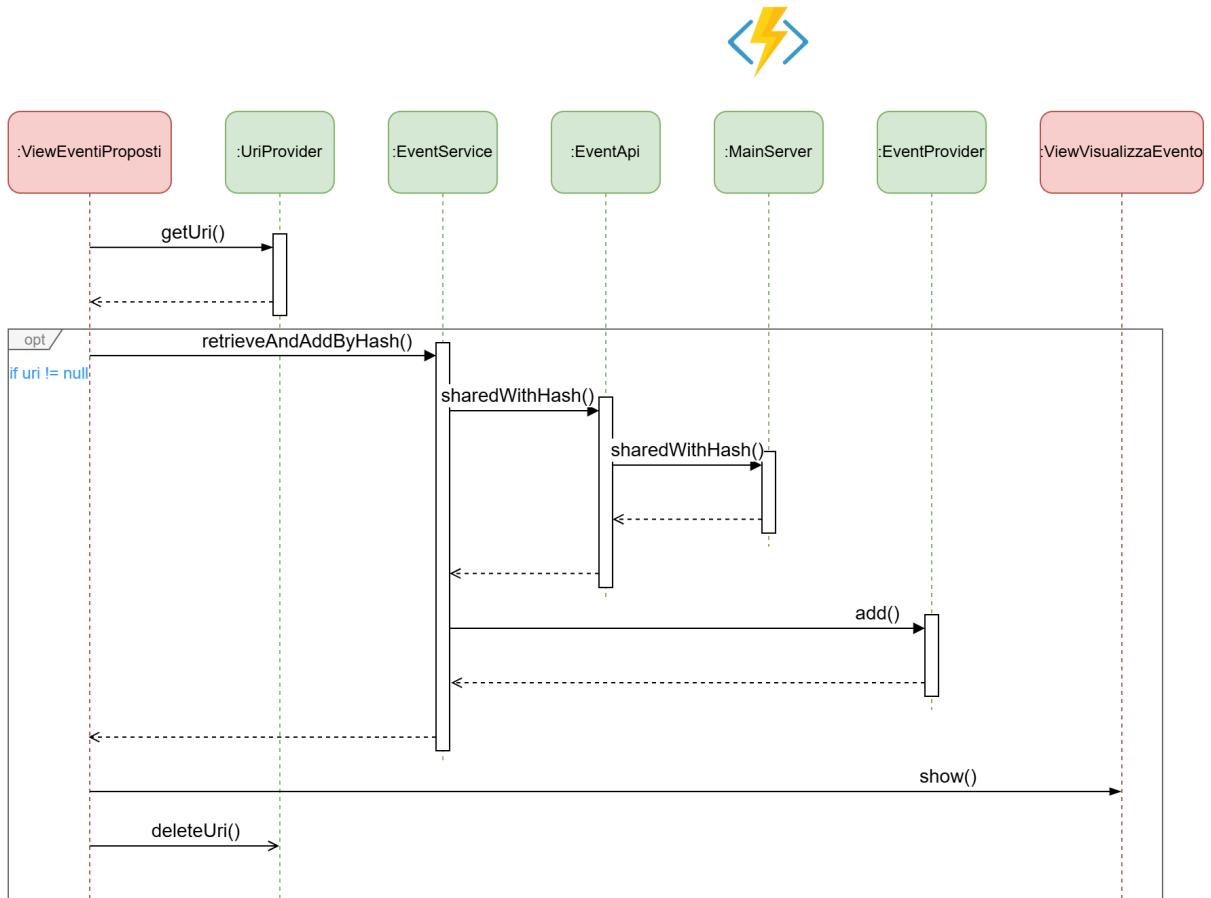


Figura 16: Diagramma di sequenza della visualizzazione degli eventi proposti

La fase di inizializzazione avviene a seguito di un login andato a buon fine. Parallelamente alla visualizzazione della schermata iniziale si recuperano i dati relativi ai profili associati all'utente e vengono fatti partire i servizi autonomi. In particolare, vengono controllati i permessi necessari per i quali, se non ancora concessi, verrà richiesto l'ottenimento. Viene inoltre avviato il servizio di ricezione degli aggiornamenti, che si connette al canale relativo all'utente. Per ogni profilo vengono, sempre in maniera parallela, recuperati i gruppi e gli eventi associati. Al termine della ricezione degli eventi, indipendentemente dalle altre richieste, per ogni evento successivo al momento attuale viene avviato un timer, che scatena, al momento giusto, il recupero delle immagini. Se l'applicazione è stata aperta tramite link di condivisione, la schermata a cui si verrà reindirizzati sarà quella degli eventi proposti, altrimenti quella degli eventi confermati.

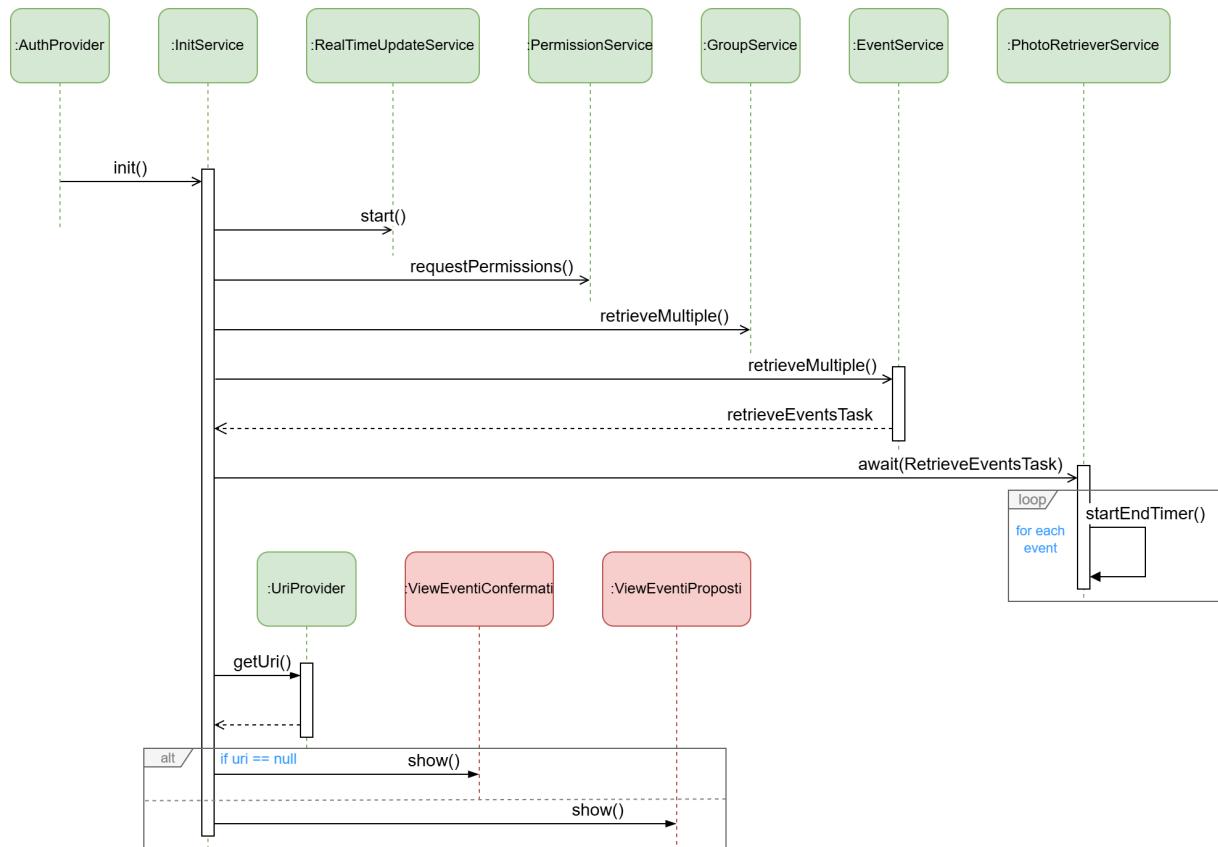


Figura 17: Diagramma di sequenza della fase di inizializzazione

#### **2.1.4 La distribuzione**

Per quanto Flutter consenta di uniformare l’esperienza utente su dispositivi diversi e semplifichi la compilazione per le varie piattaforme, alcune configurazioni rimangono comunque dipendenti dalla tecnologia su cui l’applicazione viene eseguita. Di conseguenza, ciascun eseguibile richiede una manutenzione aggiuntiva, inclusi gli aggiornamenti delle dipendenze specifiche, sia a livello di deployment che di gestione delle versioni.

In una fase iniziale dello sviluppo, nell’ottica di coprire il più ampio mercato possibile con il minor numero di piattaforme, si è deciso di sviluppare una versione fruibile via web e una per dispositivi Android.

Nell’ambito delle tecnologie offerte da Azure per la distribuzione del codice web, si è scelto Azure Static Web App, un servizio di hosting progettato per applicazioni web statiche.

La preferenza per questa soluzione è derivata dalla affidabilità dell’infrastruttura di Azure e dai bassi costi operativi per un utilizzo limitato, fattori che la rendono particolarmente vantaggiosa.

Questa selezione è stata resa possibile in quanto l’interfaccia sviluppata non prevede la creazione dinamica di contenuti, ovvero la pagina che viene restituita rimane invariata indipendentemente dall’utente che effettua la richiesta. I dati specifici dell’utente verranno infatti richiesti ad un server terzo (qualora non siano già presenti in una cache locale). Questa scelta consente di rendere l’interfaccia grafica completamente indipendente dall’identità dell’utente, migliorando così le prestazioni e riducendo il carico computazionale delegato a Azure Static Web App.

Inoltre, in attesa della pubblicazione dell’applicazione sull’App Store di Android, l’eseguibile è stato temporaneamente reso disponibile tramite Azure Storage Container. Questo servizio consente l’archiviazione e la distribuzione di file di varie tipologie, fornendo un link diretto per il recupero.

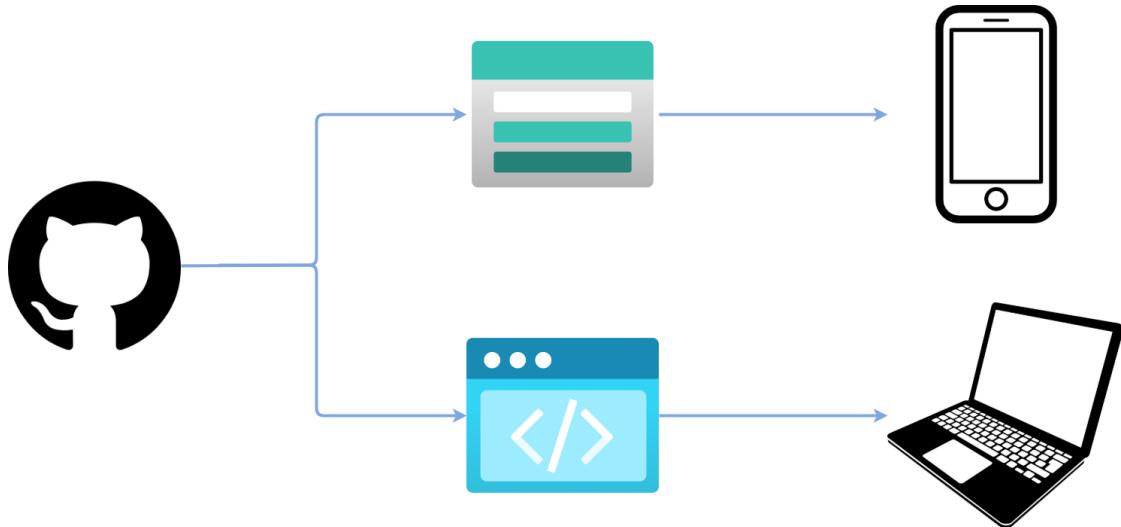


Figura 18: Diagramma di aggiornamento e distribuzione del client

### 2.1.5 L'aggiornamento

Per garantire un processo di aggiornamento efficiente e automatizzato, sia il codice distribuito sulla web app che l'applicativo ospitato nel container vengono gestiti tramite GitHub Actions, che ne assicura l'aggiornamento automatico a ogni nuova versione.

Nella fruizione tramite browser, il codice si aggiorna automaticamente a ogni accesso, mentre per l'applicativo su dispositivo mobile è necessaria una reinstallazione manuale. Per questa ragione, gli utenti dell'applicazione verranno notificati tempestivamente ogni volta che sarà disponibile una nuova versione.

## 2.2 L'architettura del server principale

Per poter essere in grado di gestire un numero elevato di richieste in tempi ridotti, l'applicazione deve garantire un'elevata scalabilità.

Capace di soddisfare al meglio questa esigenza è Azure Functions, un servizio serverless che consente di suddividere il codice in unità indipendenti, ciascuna eseguita in un ambiente di esecuzione unico per ogni richiesta. Questa caratteristica, combinata con la virtualizzazione dell'ambiente di esecuzione offerta dal cloud, consente una scalabilità potenzialmente illimitata.

L'indipendenza e la natura stateless di ogni funzione, ovvero la sua esecuzione svincolata dalle informazioni sullo stato o sulla sessione, sono responsabilità dello sviluppatore. Ogni funzione segue il principio di singola responsabilità, eseguendo un unico compito specifico. Tuttavia, nel caso di richieste che necessitino l'esecuzione coordinata di più funzioni, Azure Durable Functions fornisce una soluzione efficace.

Integrata all'interno delle Azure Functions, Azure Durable Function consente la creazione di una funzione orchestrator, incaricata di gestire l'ordine, lo stato, il ciclo di vita e le risposte delle varie funzioni coinvolte nell'elaborazione della richiesta.

Mantenendo un'architettura indipendente e scalabile, questa soluzione consente di gestire efficacemente scenari in cui un'esecuzione sequenziale delle operazioni è cruciale, dove è necessario effettuare tentativi aggiuntivi in caso di errore o fallimento o si richiede l'attesa del completamento di operazioni con un tempo di esecuzione prolungato.

Tuttavia, l'architettura stateless e l'accoppiamento debole tra orchestrator e funzioni in esecuzione causano un tempo di risposta delle Azure Durable Functions più elevato. Per ottimizzare l'allocazione delle risorse, il sistema avvia perciò solo le funzioni strettamente necessarie all'esecuzione del compito determinato, stanziando al minimo il consumo computazionale.

### **2.2.1 La scelta del linguaggio**

Come linguaggio di programmazione per lo sviluppo delle funzioni è stato utilizzato C#. La consapevolezza che sia l'ambiente di sviluppo di C#, ovvero il framework .Net, sia la piattaforma Azure siano entrambi sviluppati e mantenuti dalla stessa azienda, Microsoft, garantisce elevati livelli di stabilità, supporto e coordinamento delle tecnologie adottate.

Inoltre, l'integrazione con Entity Framework Core permette la mappatura direttamente in oggetti dei componenti del dominio, semplificando così la logica delle relazioni ed astraiendo le comunicazioni con il database. Grazie all'utilizzo delle proprietà virtuali degli oggetti si può applicare il lazy loading, riducendo il numero di richieste al database solo a quando esse sono strettamente necessarie, mantenendo in codice a livello logico ed ottimizzandone le prestazioni.

Azure Functions in ambiente .Net supporta due modelli di esecuzione e sviluppo: in-process worker o isolated worker. Il worker è il processo all'interno dell'applicativo che gestisce la creazione delle risorse e l'esecuzione delle funzioni in risposta alle richieste. Nella modalità in-process, la funzione viene eseguita all'interno dello stesso processo del worker che l'ha generata, riducendo la quantità di allocazione delle risorse necessarie ma condividendo l'ambiente di esecuzione. Nel modello isolated, invece, ogni funzione viene eseguita attraverso un processo indipendente dedicato, garantendo maggiore isolamento e quindi riducendo le possibili dipendenze tra le funzioni.

Inoltre, il modello isolated worker offre ulteriori vantaggi grazie al maggiore supporto fornito: innanzitutto esso prevede una maggiore compatibilità, grazie al più ampio numero di versioni del framework .Net a disposizione, a differenza del modello in-process, limitato alle sole versioni con supporto a lungo termine. In secondo luogo, il supporto per la creazione di middleware personalizzati permette l'elaborazione di un codice intermedio tra la chiamata e l'esecuzione della funzione, funzionalità invece non disponibile nel modello in process. Considerati questi vantaggi, le funzioni sono state sviluppate utilizzando il modello isolated worker per garantire maggiore flessibilità, compatibilità e modularità dell'architettura.

### 2.2.2 La logica applicativa

Per quanto ogni funzione ricopra un unico compito, alcuni parti possono dover essere condivise. Per questo motivo, la logica applicativa è stata suddivisa creando un metono specifico per ogni singola esigenza, che le funzioni chiameranno ogni volta che ne hanno bisogno. I metodi vengono quindi raggruppati in classi in base all'inerenza dei loro scopi, concentrando il codice che condivide le stesse necessità e uniformando il suo stile. Le dipendenze vengono quindi inizializzate un'unica volta a livello di classe, creando un software più ordinato.

Nella stessa modalità di suddivisione delle responsabilità del client, le classi sono state sviluppate tenendo conto delle divisioni del dominio e di ulteriori responsabilità specifiche. Per ogni elemento principale del dominio è stata sviluppata una classe service che implementa le operazioni relative, mentre, per compiti che richiedono particolare attenzione o che astraggono l'interazione con una particolare risorsa, vengono implementate classi apposite.

Ogni servizio relativo agli elementi strutturali ha bisogno di una connessione con il database per poter applicare le modifiche eseguite. Questa viene implementata da una classe chiamata WydDbContext che racchiude la logica e le impostazioni legate alla persistenza principale. Si concentrano così in un'unico luogo tutte le necessità e le configurazioni di basso livello relative alla sua interazione, quali la definizione del dominio e delle sue relazioni ed eventuali operazioni da applicare in automatico qualora la natura dell'oggetto lo richieda. Ad esempio, alcune classi richiedono l'aggiornamento automatico della data di ultima modifica. L'implementazioni delle classi del dominio viene trattata nei capitoli seguenti.

Per alcuni compiti specifici sono state implementate classi apposite. In particolare, AuthorizationService si occupa dell'autenticazione e dell'autorizzazione della richiesta, mentre NotificationService astrae la relazione con il servizio di aggiornamento in tempo reale.

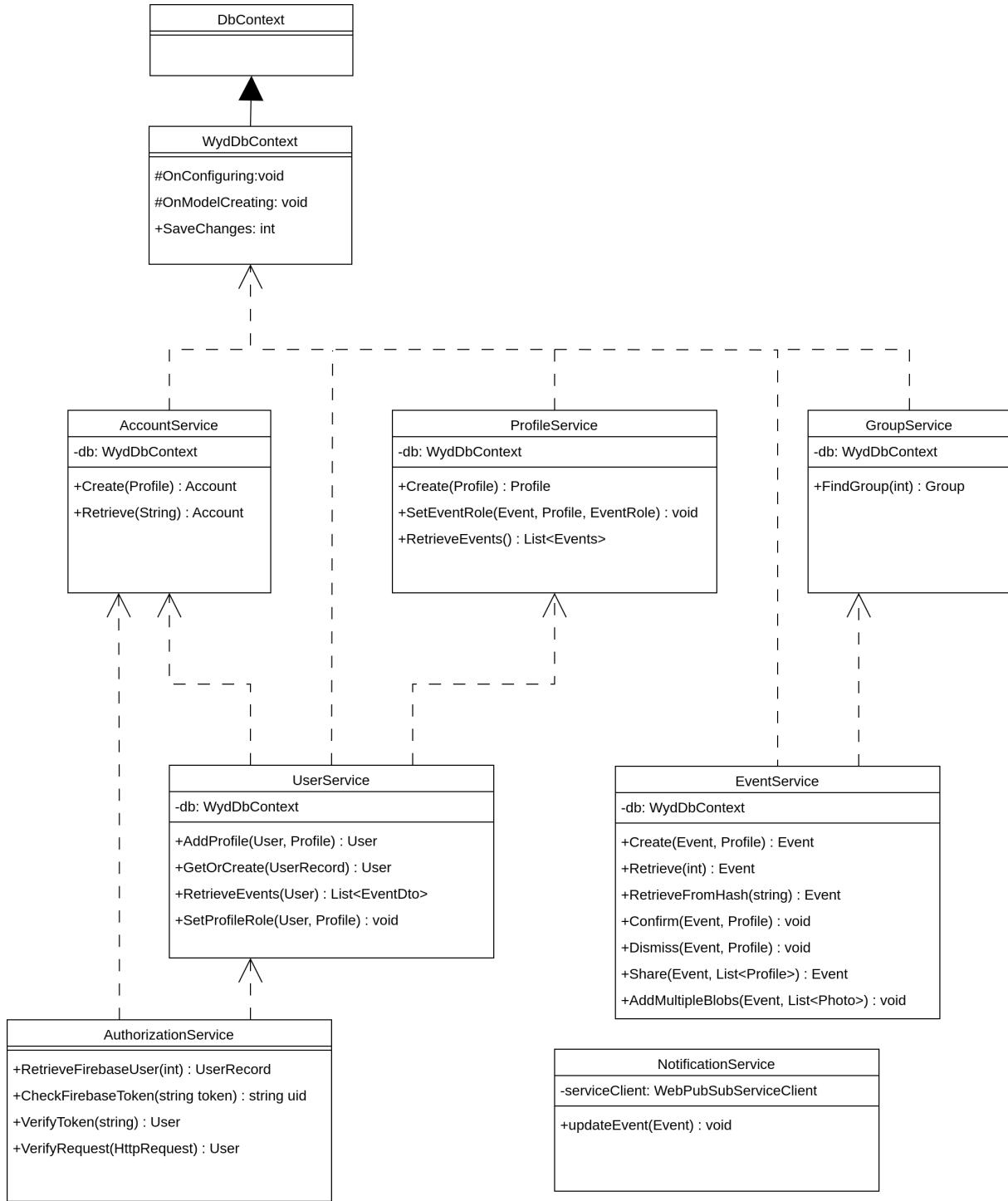


Figura 19: Modello delle classi del server

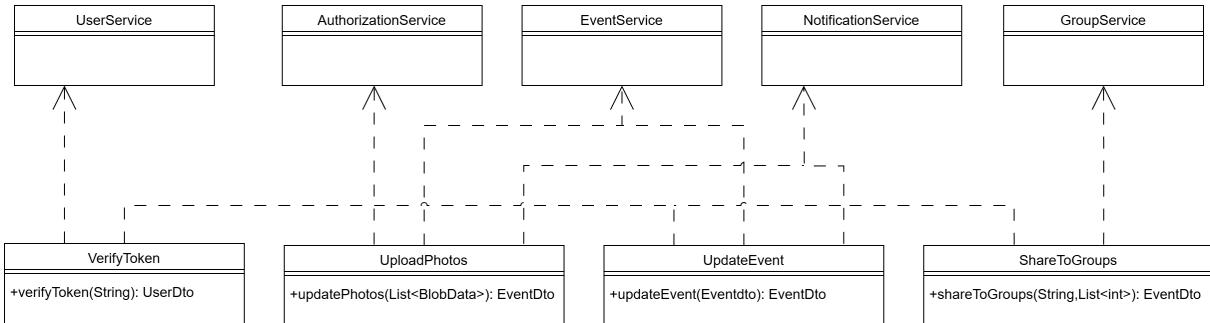


Figura 20: Modello delle relazioni tra Functions e i servizi usati

Per allineare i dati a disposizione del server con il dominio del client e per ridurre l'invio delle informazioni non necessarie, sono stati creati dei Data Transfer Object(DTO). I DTO sono classi logiche che prevedono almeno un costruttore che, dato l'elemento del dominio, ne copia solo le informazioni necessarie. Questo permette di creare rappresentazioni dei dati come necessarie al client, mascherando le logiche applicative e di fatto separando le dipendenze del dominio dai requisiti di comunicazione.

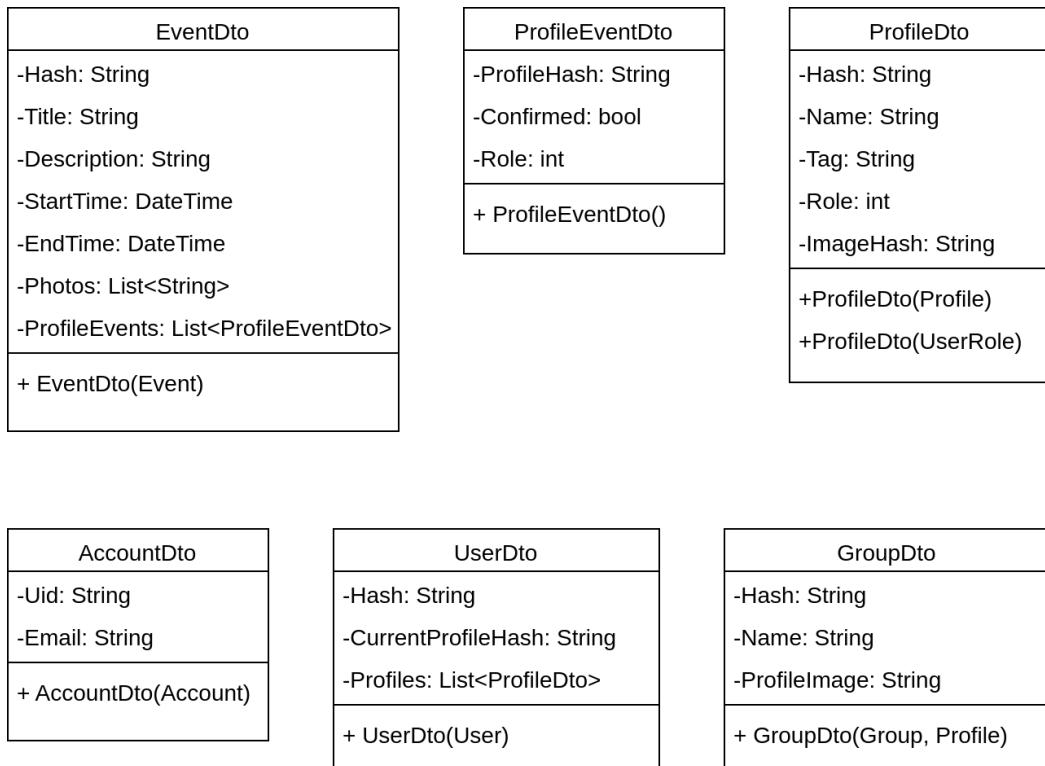


Figura 21: Modello delle classi dei data transfer object

Per ogni metodo pubblico delle classi service sono stati implementati dei test. I test permettono la simulazione di differenti situazioni per controllare che il codice segua il comportamento desiderato. La loro implementazione è quindi precedente allo sviluppo stesso delle classi, in quanto le aspettative sono già note, e il superamento dei test determina la correttezza del metodo. Inoltre, in caso di necessità particolari che escono dalle normali aspettative della funzione, quali, ad esempio, il controllo di un valore particolare o l'implementazione di un vincolo specifico, i test assicurano la loro futura presa in carico anche in caso di modifica totale del codice.

```
5 references
private static WydDbContext GetInMemoryDbContext()
{
    var options = new DbContextOptionsBuilder<WydDbContext>()
        .UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())
        .Options;

    return new WydDbContext(options);
}
```

Figura 22: Simulazione del database

```
[Fact]
0 references
public void Create_ValidAccount_ReturnsAccount()
{
    var dbContext = GetInMemoryDbContext();
    var service = new AccountService(dbContext);

    var newUser = new User { };
    dbContext.Users.Add(newUser);
    dbContext.SaveChanges();

    var account = new Account
    {
        Mail = "test@example.com",
        Uid = "uid123",
        User = newUser
    };

    var result = service.Create(account);

    Assert.NotNull(result);
    Assert.Equal("test@example.com", result.Mail);
    Assert.Equal("uid123", result.Uid);
}
```

Figura 23: Test di creazione di un account

### 2.2.3 La distribuzione

Lo sviluppo è stato condotto utilizzando Visual Studio Code, programma sviluppato dalla stessa Microsoft per la creazione di codice. Visual Studio Code permette l'integrazione con molteplici estensioni fornendo il supporto per la maggior parte delle tecnologie.

In particolare, grazie alle estensioni dedicate al provider Azure, è possibile collegare il proprio ambiente di lavoro con i servizi in cloud. Il legame così creato consente un aggiornamento immediato ed intuitivo del codice, gestito interamente dal programma.

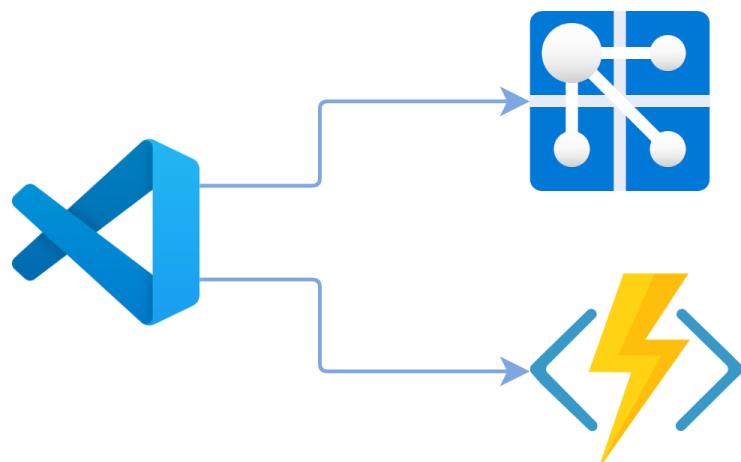


Figura 24: Diagramma di aggiornamento e distribuzione del server

## 2.3 Autenticazione

Poiché la modalità di autenticazione rappresenta un elemento cruciale per l’esperienza dell’utente, in quanto deve assicurare un accesso sicuro all’applicazione mantenendone la semplicità, la facilità del processo autenticazione deve essere garantita. Offrire agli utenti la possibilità di autenticarsi tramite il proprio authentication provider di fiducia migliora sicuramente l’usabilità e l’apprezzamento degli utenti, ma è altrettanto essenziale consentire la possibilità di creare account dedicati esclusivamente all’applicazione. Di conseguenza, il sistema di gestione degli accessi deve supportare sia la registrazione e la gestione autonoma degli account specifici per il servizio, sia fornire l’integrazione con provider di autenticazione esterni.

Per lo scopo, Azure fornisce Microsoft Entra ID, parte della suite di servizi di autenticazione e autorizzazione Microsoft Entra. Sebbene teoricamente in grado di soddisfare i requisiti sopra indicati, la complessità della documentazione e le difficoltà riscontrate nell’integrazione con il servizio dell’applicativo hanno portato a valutare soluzioni alternative negli ambienti cloud.

La scelta è ricaduta su Firebase Authentication, che garantisce sia la possibilità di creare account dedicati che di collegarsi attraverso altri authentication providers. Inoltre, la piattaforma presenta un’interfaccia chiara e offre servizi di integrazione di facile utilizzo sia tramite Flutter che tramite C#. Dal punto di vista economico, il servizio risulta vantaggioso, essendo gratuito fino ai cinquantamila utenti mensili attivi.

Uno dei requisiti fondamentali del progetto prevede che ogni account sia associato in modo univoco a un singolo utente. Durante la fase di creazione del profilo, tuttavia, l’account viene inizialmente registrato nel database gestito da Firebase. Pertanto, al primo accesso, il server, dopo aver verificato l’autenticità della richiesta, provvede a creare una copia dell’account, generando poi il relativo nuovo oggetto utente e il primo profilo associato.

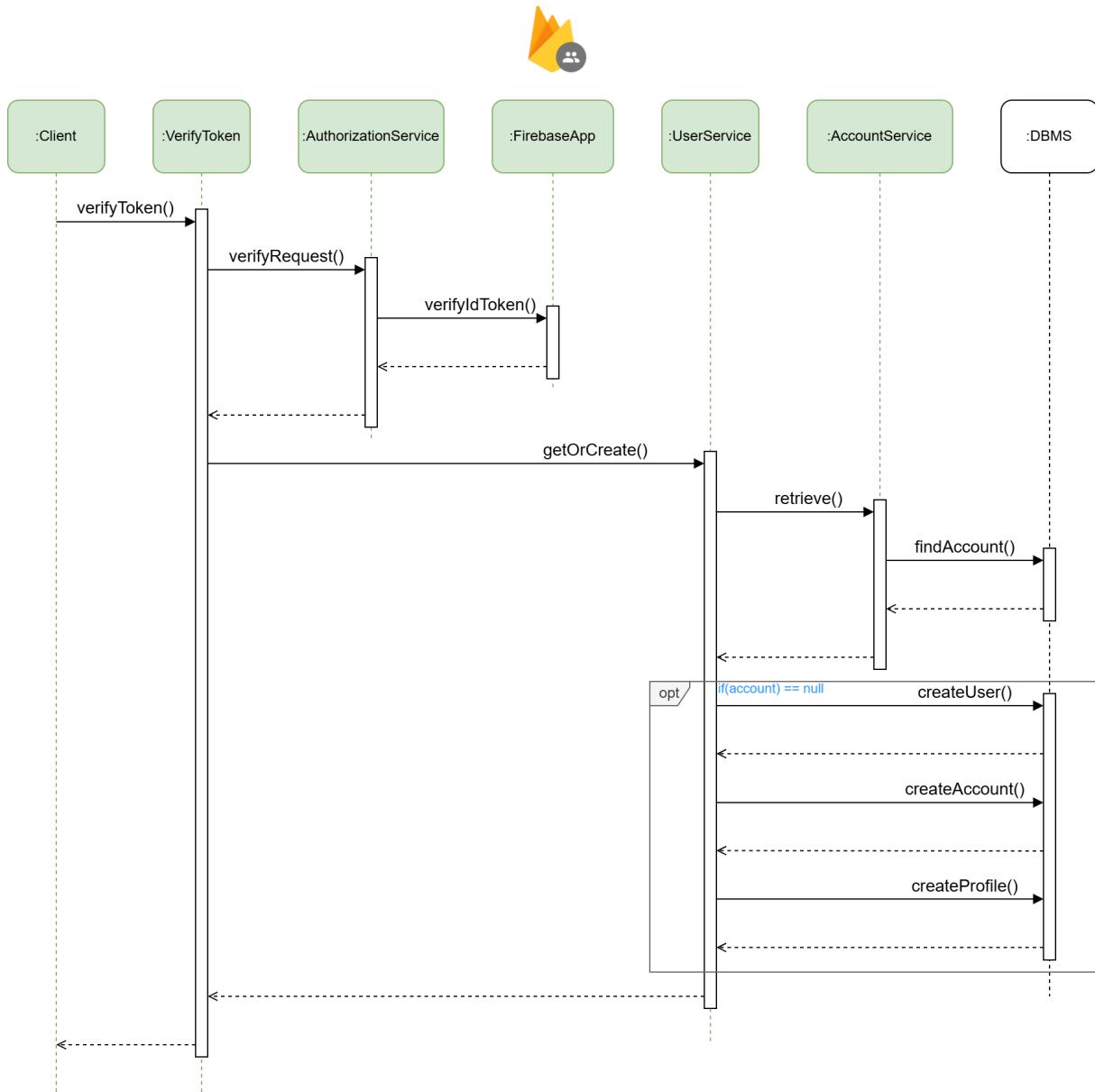


Figura 25: Diagramma di sequenza per la creazione di un account

Per garantire un processo di autenticazione sicuro ed efficiente, a ogni richiesta che richiede identificazione, il dispositivo utente aggiunge anche il token di autenticazione, salvato durante il primo accesso e la verifica iniziale. Il server verifica quindi la validità del token prima di procedere con l'esecuzione della richiesta.

## 2.4 La sicurezza

Il collegamento tra i vari componenti all'interno dell'ambiente Azure richiede l'utilizzo di chiavi e stringhe di connessione. Il salvataggio di tutte le chiavi sensibili è stato affidato al servizio Azure Key Vault, un server che permette la centralizzazione dei dati, cifrando il contenuto e garantendo un controllo maggiore sul loro utilizzo. Quando necessario i servizi, in particolare le Azure Functions, contatteranno il Key Vault per l'ottenimento delle chiavi necessarie, riducendo il rischio di un'intercettazione data magari da un errore durante lo sviluppo.

Le comunicazioni tra i vari componenti devono avvenire in sicurezza, garantendo autenticità e confidenzialità. Per questo motivo tutte le comunicazioni tra dispositivi client e i vari servizi utilizzano la tecnologia TLS, che permette di cifrare i messaggi grazie ad uno standard collaudato. In particolare, le comunicazioni tra i client e Azure Functions, così come con Firebase Authentication e il server per la persistenza delle immagini, avvengono tramite protocollo HTTPS, mentre le comunicazioni con il server per gli aggiornamenti in tempo reale usano il protocollo WSS.

Il rischio di saturazione delle risorse viene mitigato aggiungendo un duplice controllo sulle dimensioni delle richieste. In primo luogo si limita la dimensione massima della singola richiesta, facendo particolare attenzione alle richieste che contengono immagini, controllandola sia nel momento dell'invio che nel momento della ricezione. Inoltre, alla fine di ogni richiesta più grande di una determinata soglia, la dimensione viene sommata alle precedenti nell'ultimo periodo e, se la somma risulta troppo elevata, viene limitato l'utilizzo per quell'utente.

Per evitare un numero eccessivo di richieste totali, che possono provocare anch'esse una riduzione del servizio, è possibile integrare nel sistema risorse create appositamente da Azure, quali Azure DDOS Protection.

L'accesso al database è ristretto alle sole risorse Azure, garantendo l'isolamento dall'esterno, che comprometterebbe altrimenti l'affidabilità dei dati.

Infine, l'identificativo di ogni elemento del dominio è nascosto all'utente tramite la creazione di codici hash univoci che permettono comunque l'identificazione dell'oggetto senza rivelare ulteriori informazioni. In particolare, il caricamento delle immagini avviene grazie ad un link univoco dato dalla combinazione degli identificativi dell'evento e dell'immagine. Utilizzando i codici di hash diventa molto complicato il ritrovamento delle immagini senza essere a conoscenza dei codici, che non avendo natura incrementale ma distribuita rende indovinare un link valido.

## 2.5 Monitoraggio

Il monitoraggio del sistema è attuato in due modalità: tramite salvataggio dei log e controllo delle prestazioni del sistema.

Relativamente a Firebase Authentication vengono forniti con il servizio sia le interfacce per il controllo delle prestazioni che la gestione dei log. Non è quindi richiesta alcuna ulteriore azione.

Per monitorare le Azure Functions sarà invece necessario collegare Azure Application Insights, servizio che provvede a controllare il funzionamento e la risposta del servizio. Una volta unito il servizio, infatti, Azure Application Insight permette la presentazione e l'analisi di numerose metriche, quali il tempo di risposta e il consumo di risorse. Consente inoltre di testare la risposta dell'applicativo simulando diversi scenari e riassumendo il loro comportamento.

La creazione dei log è invece delegata al programmatore, in quanto è necessario integrarli nel codice. Nel momento della creazione, ogni funzione riceve, tramite dependency injection, un servizio Logger che permette la creazione e il salvataggio dei log. Tali log saranno poi consultabili e analizzabili tramite l'interfaccia fornita da Azure Application Insight.

### 3 La persistenza

Nel contesto di un sistema che prevede la gestione di eventi e impegni, è essenziale garantire che l'utente possa accedere in qualsiasi momento alla propria agenda, visualizzando gli appuntamenti più urgenti e pianificando efficacemente il proprio tempo. Deve perciò essere possibile trovare e mostrare nel minor tempo possibile i dati relativi all'agenda. Per soddisfare questo requisito, il recupero e la visualizzazione dei dati devono avvenire con la massima rapidità possibile, riducendo i tempi di latenza e ottimizzando il flusso di interazione con l'interfaccia utente.

L'adozione di un meccanismo di salvataggio locale sui dispositivi offre il vantaggio di migliorare le prestazioni, consentendo un accesso immediato alle informazioni senza dover effettuare continue richieste al server remoto. Tuttavia, questa soluzione presenta dei limiti, in quanto non garantisce una persistenza a lungo termine dei dati né assicura la loro disponibilità su più dispositivi. Per superare tale criticità, è necessario definire una strategia di gestione della memoria che preveda una fonte di dati centrale e autorevole, alla quale tutti i dispositivi possano fare riferimento per recuperare e aggiornare le informazioni in modo coerente e affidabile.

Un sistema di persistenza efficace deve quindi prevedere un meccanismo di sincronizzazione tra i dati salvati localmente e la loro controparte ufficiale memorizzata nel database centrale. Questo processo deve essere progettato in modo da garantire integrità, coerenza e scalabilità nelle interazioni, per essere resistente anche in presenza di un volume significativo di richieste concorrenti. La struttura del sistema di memorizzazione deve inoltre essere progettata tenendo conto del dominio applicativo e delle esigenze specifiche di utilizzo, al fine di assicurare un bilanciamento ottimale tra efficienza e robustezza operativa.

Oltre alla gestione della persistenza dei dati per il singolo utente, è necessario affrontare il problema della condivisione e della modifica collaborativa delle informazioni. Poiché l'applicazione consente a più utenti di interagire sugli stessi eventi, le modifiche effettuate da un partecipante devono essere propagate in tempo reale agli altri dispositivi coinvolti.

Questo introduce la necessità di implementare un sistema di aggiornamento distribuito, in grado di mantenere sincronizzati non solo i dispositivi di un singolo utente, ma anche quelli di tutti gli utenti interessati dalle modifiche.

La gestione dell'accesso ai dati richiede quindi l'implementazione di un'architettura che preveda un punto di riferimento centrale chiaro e affidabile, capace di fungere da fonte primaria delle informazioni. Al tempo stesso, l'utilizzo di copie locali dei dati sui dispositivi client consente di ridurre l'impatto delle latenze di rete, migliorando la reattività dell'interfaccia e offrendo un'esperienza utente più fluida.

L'adozione di una memoria centrale unica garantisce l'integrità e l'affidabilità dei dati ufficiali, fornendo un punto di accesso sicuro per la consultazione e la modifica delle informazioni. Tuttavia, questa scelta introduce la necessità di gestire due livelli distinti di responsabilità: da un lato, il client deve occuparsi di mantenere aggiornati i dati memorizzati localmente, mentre il server deve garantire la corretta distribuzione delle modifiche agli altri dispositivi interessati. La sincronizzazione e la gestione delle versioni dei dati diventano quindi elementi chiave per assicurare la coerenza del sistema e prevenire eventuali conflitti tra modifiche concorrenti.

### **3.1 Memoria principale**

Nell’implementazione di applicazioni scalabili, la gestione del salvataggio dei dati può essere strutturata secondo un modello centralizzato o distribuito, a seconda delle esigenze di affidabilità, scalabilità e prestazioni del sistema. L’adozione di un’architettura di memoria distribuita offre molteplici vantaggi, tra cui una maggiore resilienza ai guasti di una singola fonte, la riduzione del carico di memoria su un’unica risorsa di archiviazione e una migliore scalabilità complessiva del sistema. Tuttavia, questa soluzione introduce una maggiore complessità infrastrutturale, poiché richiede meccanismi avanzati per garantire la persistenza, l’affidabilità e la consistenza delle informazioni.

Salvo specifici requisiti che rendano indispensabile la distribuzione totale o parziale della memoria, una strategia basata su un database centralizzato può risultare più efficiente dal punto di vista prestazionale e semplificare la gestione complessiva del sistema. L’adozione di un’architettura centralizzata consente infatti di ottimizzare i tempi di accesso ai dati e ridurre la latenza delle operazioni, a fronte di una minore complessità nella sincronizzazione e nel mantenimento della consistenza delle informazioni.

#### **3.1.1 I database e la scalabilità**

I database si suddividono in due macro categorie principali: relazionali e non relazionali. I database relazionali si caratterizzano per strutture dati rigide e schematizzate, che consentono di stabilire connessioni tra le diverse risorse in tempi estremamente rapidi. Viceversa, i database non relazionali offrono una maggiore flessibilità strutturale, permettendo l’archiviazione di dati eterogenei con un accoppiamento più debole tra le entità. Questa caratteristica li rende particolarmente adatti a scenari in cui è richiesta un’elevata scalabilità orizzontale e in cui il volume di richieste è significativo.

La scelta della tipologia di database più appropriata dipende strettamente dalle esigenze specifiche del progetto. I database relazionali sono gli unici in grado di garantire le proprietà di Atomicità, Consistenza, Isolamento e Durabilità (ACID), grazie all'implementazione di un rigoroso sistema di transazioni. Questo meccanismo, se da un lato assicura un'elevata affidabilità dei dati, dall'altro introduce un temporaneo blocco delle risorse durante le operazioni di scrittura, potenzialmente influenzando le prestazioni complessive del sistema.

Dal punto di vista della scalabilità, invece, i principali fattori da considerare includono il rapporto tra operazioni di lettura e scrittura, la natura dei dati gestiti e la relazione tra le entità. Le operazioni di lettura possono essere ottimizzate tramite la creazione di copie dei dati e la parallelizzazione dei processi, permettendo di soddisfare un elevato numero di richieste simultanee.

Al contrario, le operazioni di scrittura possono introdurre ritardi significativi, in quanto implicano la temporanea indisponibilità della risorsa fino al completamento dell'aggiornamento. In alcuni casi, in base alla tipologia del database, ciò può invalidare le copie presenti nel sistema fino alla propagazione dell'aggiornamento, compromettendo temporaneamente la consistenza delle informazioni. Grazie alla loro struttura flessibile e a requisiti meno stringenti in termini di garanzia ACID, i database non relazionali risultano generalmente più efficienti in contesti caratterizzati da operazioni di scrittura frequenti.

Quando il sistema richiede una gestione dinamica delle informazioni, con dati opzionali o variabili, i database non relazionali offrono un livello di adattabilità superiore. Il salvataggio delle informazioni in forma di documenti consente infatti di modificare la struttura dei dati senza la necessità di riconfigurare lo schema del database, garantendo una maggiore elasticità. Tuttavia, questa necessità si manifesta prevalentemente in casi specifici, che rappresentano solo una parte del dominio applicativo complessivo.

### 3.1.2 La gestione delle relazioni tra le entità

Un aspetto critico nella scelta del database riguarda la gestione delle relazioni tra le entità. È fondamentale analizzare la distribuzione delle richieste per ciascun elemento e il carico computazionale associato a ogni operazione. Tra le operazioni più costose in termini di prestazioni, che più ostacola e rallenta il recupero dei dati, vi è l'operazione di unione necessaria. Durante le operazioni di unione vengono incrociati i dati di vari elementi per restituire un oggetto coerente che presenta tutte le proprietà necessarie, originariamente distribuite in molteplici tabelle. L'esecuzione di unione su database relazionali, pur essendo ottimizzata, introduce azioni computazionali di grande entità che possono impattare significativamente sui tempi di risposta delle richieste. Per questo motivo si cerca di ridurre il più possibile le richieste che comportano l'incrocio di dati da tabelle diverse.

Le relazioni tra elementi possono essere classificate in tre categorie principali: di tipo uno a uno, uno a molti, e molti a molti.

Nelle relazioni uno a uno il recupero dei dati è diretto e richiede uno sforzo computazionale limitato. Nei casi uno a molti e molti a molti il reperimento delle informazioni richiede un'attenta valutazione delle modalità di accesso per ottimizzare le prestazioni: in questo caso, una delle strategie possibili per migliorare l'efficienza delle richieste offerte dai database non relazionali è il salvataggio per copia del dato all'interno dell'oggetto. Questa operazione risulta efficiente se le richieste sono sproporzionate verso una delle due parti.

Ad esempio, in una relazione uno a molti, nel caso in cui la richiesta di quell'elemento non sia frequente ma sia importante ottenere gli elementi collegati; conviene copiare gli oggetti relativi all'interno dell'elemento singolo. L'impostazione inversa, in cui si copia il singolo all'interno dei molti elementi, comporterebbe l'ispezione di tutti i componenti esistenti alla ricerca di quelli che contengono l'elemento dato. Se però, viceversa, sono frequenti le richieste relative agli elementi multipli, e la loro relazione è importante, conviene copiare il singolo all'interno di detti elementi, per evitarne il recupero ogni volta.

I database relazionali non permettono la copia di elementi all'interno di altri, ma so-

no ottimizzati per unire tra loro tabelle, richiedendo comunque tempistiche e capacità elaborative non indifferenti.

Tuttavia, non vi è alcun vincolo che impedisca l'affiancamento di database di tipologia diversa per rispondere a esigenze specifiche e sfruttare i punti di forza di entrambe le tecnologie.

### 3.1.3 Analisi del dominio

La scelta del database deriva da un'attenta analisi delle principali interazioni tra gli elementi del dominio. Il dominio descrive i componenti dell'applicazione e le loro relazioni. In particolare, vengono espresse le dipendenze, i rapporti reciproci e le cardinalità delle relazioni. A titolo esemplificativo, dal diagramma del dominio si deduce che ad un oggetto Event possono corrispondere più oggetti Photo, ma soprattutto che l'oggetto Photo è strettamente legato all'oggetto Event.

Le richieste riguardo alle informazioni tra Account, User e Profile, con i relativi UserRole, vengono eseguite all'avvio del programma, per poi essere mantenute in memoria locale. Le modifiche a questi elementi sono sporadiche. Allo stesso modo gli oggetti Group vengono recuperati solo all'avvio dell'applicazione, e, salvo rari aggiornamenti, non occupano ulteriormente lo spazio delle richieste.

La maggioranza delle richieste verterà sull'ottenimento dei dati relativi agli Event e ai Profile. Infatti ad ogni avvio dell'applicazione sarà richiesto di recuperare gli eventi di ogni profilo, mentre, ogni volta che si apre il dettaglio di un evento, sarà necessario recuperare i rispettivi dati, includendo i profili associati. Si prevede che la cardinalità dei profili associati ad un evento risieda nell'ordine delle decine, mentre agli eventi che si associano ai profili l'ordine di grandezza previsto risiede nelle migliaia.

La relazione che accomuna Event e Profile è di tipologia molti a molti, identificata con l'oggetto ProfileEvent. L'elemento ProfileEvent, oltre a descrivere la relazione, contiene la

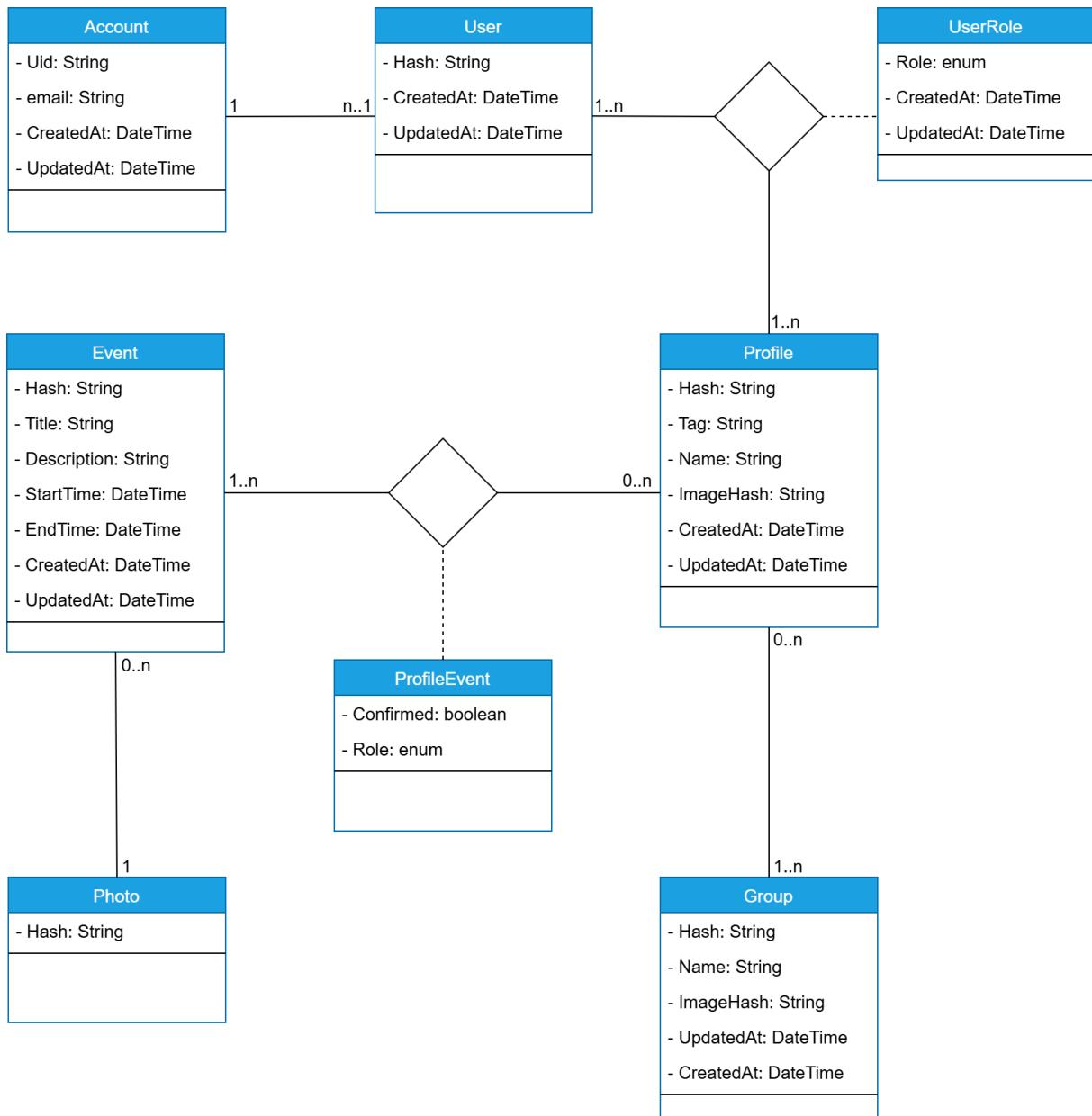


Figura 26: Diagramma del dominio

proprietà Confirmed, che indica la conferma di partecipazione di un profilo ad un evento. Vista la natura del progetto, la proprietà Confirmed, assieme ai dati degli Event, sarà tra i campi che subiranno modifiche con maggiore frequenza.

Riassumendo, le operazioni principali che vertono sulle prestazioni del database sono il recupero degli eventi relativi ai profili, il recupero dei profili collegati agli eventi, l'aggiornamento dei campi degli oggetti Event e la modifica del campo Confirmed relativa ai ProfileEvent.

Nonostante sia più probabile che venga richiesto il dettaglio di un evento, e quindi sia più frequente il dover recuperare i Profile relativi all'Event, la proporzione delle richieste prevista non giustifica lo sbilanciamento della relazione sugli Event. Infatti, se si salvassero tutti i ProfileEvent sull'oggetto Event, le richieste degli eventi appartenenti ai profili, sebbene meno frequenti, richiederebbero l'ispezione di tutti gli Event alla ricerca del Profile indicato. Infine, se si duplicatesse l'oggetto ProfileEvent, oltre che nella sua tabella originaria, anche sugli Event la sua creazione, la sua eliminazione e la modifica del campo Confirmed richiederebbero il doppio delle scritture.

Vista quindi la necessità di letture frequenti da entrambe le parti di una relazione molti a molti e la necessità di scrittura dell'oggetto che la identifica, si individua in un database relazionale la soluzione più efficace per il dominio del progetto. Infatti, i database relazionali sono ottimizzati sulle operazioni di unione tra tabelle, garantendo velocità di recupero in lettura da entrambe le parti. Permettendo la separazione delle tabelle e gestendo gli oggetti ProfileEvent come entità indipendenti, la modifica del campo Confirmed comporta il blocco del solo elemento ProfileEvent. Le caratteristiche ACID forniscono inoltre uno stato centrale per tutta l'applicazione, che, per quanto non strettamente necessario, garantisce l'uniformità delle informazioni per tutti gli utenti.

### 3.1.4 La scelta del database

Azure offre un'ampia scelta di database relazionali che possono essere integrati con il resto dell'ecosistema. Oltre alla tecnologia proposta, nella scelta del database più adatto bisogna considerare soprattutto l'integrazione con servizi accessori, le particolarità del server su cui viene eseguito, e i costi che si andrà ad affrontare.

Name	Publisher	Plan	Price starts at
Azure Database for PostgreSQL	Microsoft	Azure Database for PostgreSQL	
SQL Database	Microsoft	SQL Database	
Oracle Database@Azure	Oracle America, Inc.	Oracle Database@Azure Pay As You... ▾	Price varies
Database Performance Analyzer	SolarWinds Worldwide LLC	Private Offer	€0.936/month
SQL Elastic database pool	Microsoft	SQL Elastic database pool	
Azure Database for PostgreSQL Flexible Server	Microsoft	Azure Database for PostgreSQL Flexible Server	€0.234/3 years
Vector Database PostgreSQL with pvector extension	Info Inlet Inc.	vector database postgresql	
SAP HANA Database Deployment on Azure	Public Cloud Group GmbH	SAP HANA database	Price varies
Postgres Pro Standard Database 16 (VM)	Postgres Professional	Postgres Pro Standard Database 16... ▾	
HCLTech Database Migration Solution to Azure	HCL Technologies Limited.	Database Migration from HCL	Price varies
Couchbase Capella - Database-as-a-Service	Couchbase	Couchbase Capella Free Tier ▾	Price varies
Postgres Pro Standard Database 13 (VM)	Postgres Professional	Postgres Pro Standard Database 13... ▾	
Postgres Pro Standard Database 15 (VM)	Postgres Professional	Postgres Pro Standard Database 15... ▾	
FlashGrid for Oracle RAC and Database HA	FlashGrid Inc.	FlashGrid Cluster for Oracle RAC / S... ▾	€0.743/hour
Pre-configured DBeaver for Seamless Database Management	TechLatest	Pre-configured DBeaver for Seamless Database Management S...	€0.026/hour
Migrate and Manage Oracle Database (Exadata) at Azure	Tessell	Database Lifecycle Management for Oracle Exadata on Azure	Price varies
Kinecta, The Database for Time & Space (Bring Your Own License)	Kinecta	Bring Your Own License	Free
Exasol Analytics Database (EnterpriseSupport:PAYG)	EXASOL	Exasol 7.1.26 ▾	€0.52/hour
Oracle Database Client 19c on OEL 8	cloudimg	Oracle Database Client 19c on OEL 8	€0.021/3 years

Figura 27: Proposte di Azure per i database relazionali

A meno di necessità particolari che richiedono l'utilizzo di una tecnologia implementata da uno specifico database, la grande sovrapposizione di funzionalità delle diverse offerte di database relazionali presenti sul mercato garantisce il soddisfacimento delle necessità rilevate dall'analisi del dominio indipendentemente dalla tecnologia proposta dal servizio.

La grande differenza tra i vari servizi sta nelle proprietà del server incaricato di fornire il potere computazionale necessario per l'esecuzione. L'architettura del server e la sua integrazione con la tecnologia del database, infatti, determinano l'effettiva capacità di scalabilità del servizio.

Si intende scalabilità verticale la capacità di aumentare le risorse della stessa macchina in cui si esegue il codice. La scalabilità verticale viene definita nel momento di creazione del servizio, in cui si determinano le risorse da dedicare alla macchina che esegue il programma. Trattandosi di macchine virtualizzate, è sempre possibile in un secondo momento aumentare le prestazioni in caso di necessità.

Per scalabilità orizzontale si intende invece la capacità di delegare il carico di lavoro ad altre macchine, eventualmente coordinando le modifiche. Questo permette una risposta alle richieste più resistente, riducendo il rischio di colli di bottiglia che potrebbero venirsi a formare nell'utilizzo di un nodo singolo. La scalabilità orizzontale richiede però l'implementazione di tecnologie apposite integrate con il database che permettano l'esecuzione in nodi fisici differenti.

Una volta individuata la tecnologia adatta e il livello di scalabilità desiderati, è bene considerare le altre necessità o le opportunità aggiuntive generate dalla presenza di un database nel progetto.

L'alta disponibilità(HA) è la proprietà di garantire l'accesso al servizio nonostante i guasti. Ad esempio, si può mantenere una macchina identica al server principale in grado di replicare il servizio, spostando il carico in caso di guasto del server principale. Si misura in “numero di nove”, ovvero la quantità di nove presenti nella percentuale del tempo per il quale si garantisce la disponibilità del servizio. I servizi offrono diverse qualità di HA, in base alle funzionalità desiderate.

Alcuni servizi possono presentare offerte di backup per riportare il server nello stesso stato di qualche momento precedente. Questo permette il ripristino del sistema ad un punto precedente rispetto all'avvenimento di eventuali errori o guasti del sistema.

Inoltre, Azure mette a disposizione molteplici servizi accessori che possono essere uniti al servizio. Questo permette di estendere le potenzialità del database tramite l'analisi e il monitoraggio dei dati, generando prestazioni aggiuntive o integrando i dati per lo sviluppo di altre tecnologie.

Infine è necessario controllare i costi che le scelte progettuali e prestazionali hanno comportato: pur generalmente legati al consumo effettivo delle risorse, e quindi in grado di fornire solo una previsione del costo finale, ogni decisione presa conduce ad un possibile aumento di prezzo, ed è quindi bene controllare che le risorse selezionate siano effettivamente necessarie a soddisfare i requisiti del progetto.

Riducendo al minimo i costi, visto l'utilizzo iniziale dell'applicazione, e nessuna necessità tecnologica specifica, la scelta del database per la persistenza del progetto è ricaduta su Azure SQL Database. Presentando un database relazionale di tecnologia proprietaria di Microsoft, Azure SQL database esegue su un solo nodo, fornendo però la possibilità di scalare verticalmente tramite la possibilità di modificare le risorse assegnate in ogni momento, garantendo comunque prestazioni soddisfacenti per il servizio.

Al server principale è stata affiancata una replica che rimane costantemente aggiornata. Situata in una località differente dal server principale, garantisce alta disponibilità continuando a fornire i servizi anche in caso di malfunzionamenti al server principale.

### 3.1.5 Le limitazioni dei database relazionali

La scelta di un database relazionale può però comportare limitazioni a livello prestazionale. Il più grande tallone d'Achille dei database relazionali è il numero limitato di connessioni contemporanee permesso. Questo comporta un numero massimo di richieste in contemporanea che il database può gestire, minacciando la scalabilità.

Per ovviare ai problemi di scalabilità ci sono diverse soluzioni non esclusive che possono migliorare le prestazioni. Sicuramente si deve limitare al minimo il tempo in cui ogni richiesta mantiene la connessione, distinguendo nel codice momenti precisi e definiti in cui vengono richieste le modifiche al database.

Inoltre, si può interporre un livello di caching tra la logica applicativa e le richieste al database. Il livello di caching si occupa di gestire le richieste al database fornendo e duplicando le risposte che possiede già in memoria, eventualmente centralizzando le richieste in caso i dati siano invece da recuperare. Per i dati in scrittura, invece, salva temporaneamente le modifiche richieste, aggiornando subito la memoria locale, per poi apportare le modifiche al database in momenti di carico ridotto. Garantisce così un tempo di risposta e di propagazione degli aggiornamenti ridotto. Questo consente di alleviare le richieste al database, estendendo di molto le prestazioni fornite.

Nel caso in cui però fossero necessarie ulteriori prestazioni, se il dominio e i requisiti lo permettono, si può eventualmente delegare ad un database non relazionale le modifiche ai dati e alle relazioni che non necessitano delle qualità ACID ma richiedono un'alta frequenza di scrittura.

Infine, se fosse necessario mantenere un database relazionale, per aumentare le prestazioni conviene cambiare architettura e adottarne una che supporti la scalabilità orizzontale, per suddividere il carico su più nodi.

### 3.1.6 L'integrazione con C#

La scelta di un database relazionale per la persistenza ha comportato sviluppi progettuali precisi. In primis si rende necessario tradurre il dominio in componenti relazionali che possano essere espressi e salvati nelle tabelle del database.

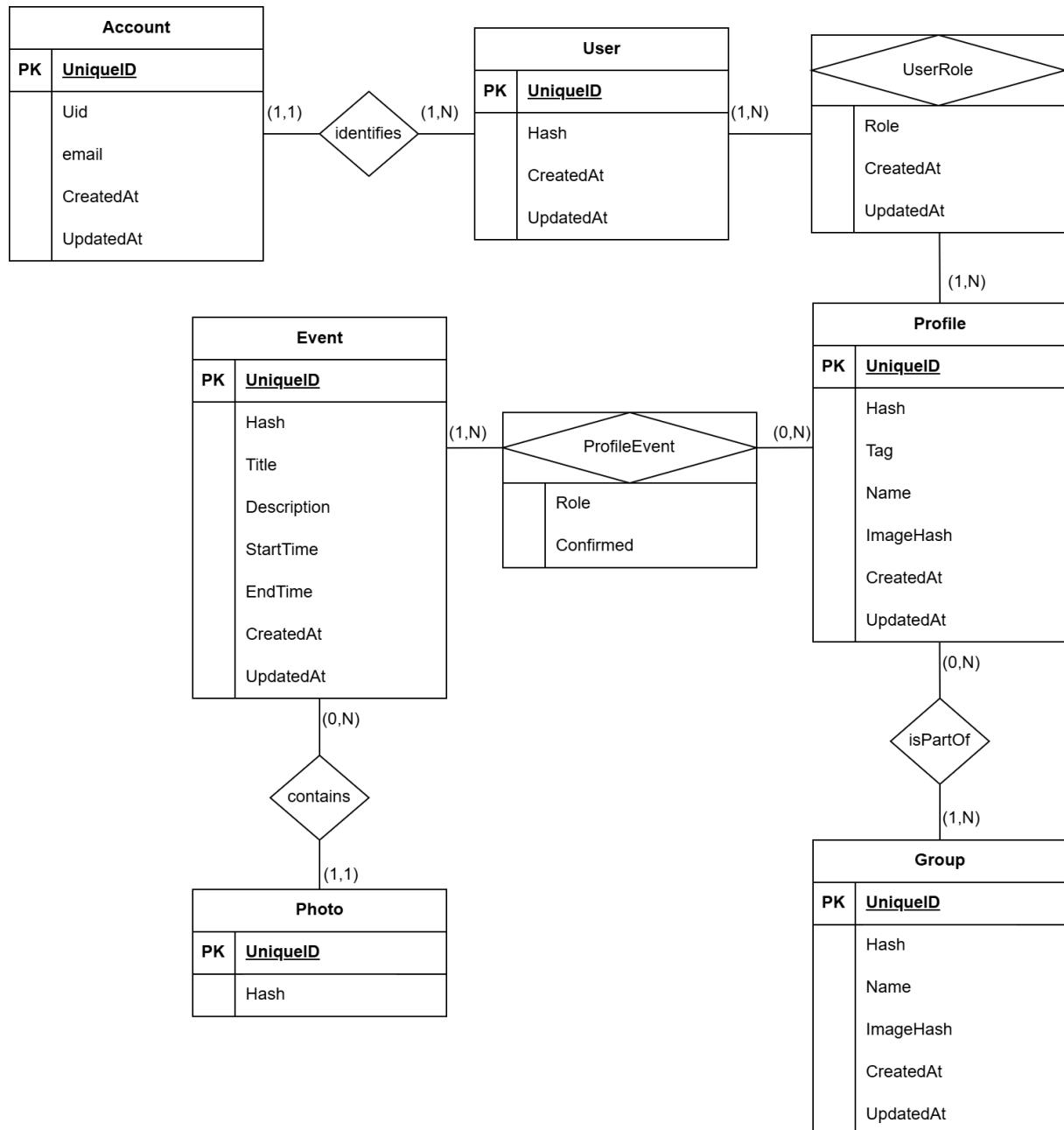


Figura 28: Diagramma Entità - Relazione del dominio

Si creano quindi sul server le classi logiche del programma, a partire dal dominio. Ogni classe corrisponde ad un oggetto del dominio, presentando i valori e le relazioni dei

componenti come attributi dell'oggetto.

Entity Framework Core di .Net(EFCore) è una libreria di C# che permette di unire le classi logiche del programma alle tabelle del database. Fornisce un'astrazione logica del collegamento con il database e le richieste relative, fornendo una rappresentazione di alto livello delle connessioni sottostanti.

Una volta collegato il server con il database tramite le stringhe di connessione salvate sull'Azure Key Vault, sono state definite le proprietà tra le varie entità, per poi inizializzare in automatico la struttura del database. Le modifiche alla struttura del database vengono infatti generate automaticamente da EFCore in seguito alla creazione o alla modifica degli attributi degli oggetti. Questo permette di star dietro agli aggiornamenti, generando e salvando le modifiche da applicare ad ogni modifica delle proprietà del dominio.

Per la riduzione del carico computazionale richiesto da elementi con tante relazioni si utilizza la tecnica del lazy loading. La tecnica del Lazy Loading consiste nel richiedere i dati delle relazioni di un elemento solo quando strettamente necessario. La sua realizzazione tramite EFCore è attuata grazie alla proprietà virtual, che permette di gestire un oggetto con un riferimento al database richiedendo i dati delle sue relazioni solo quando viene espressamente richiesto.

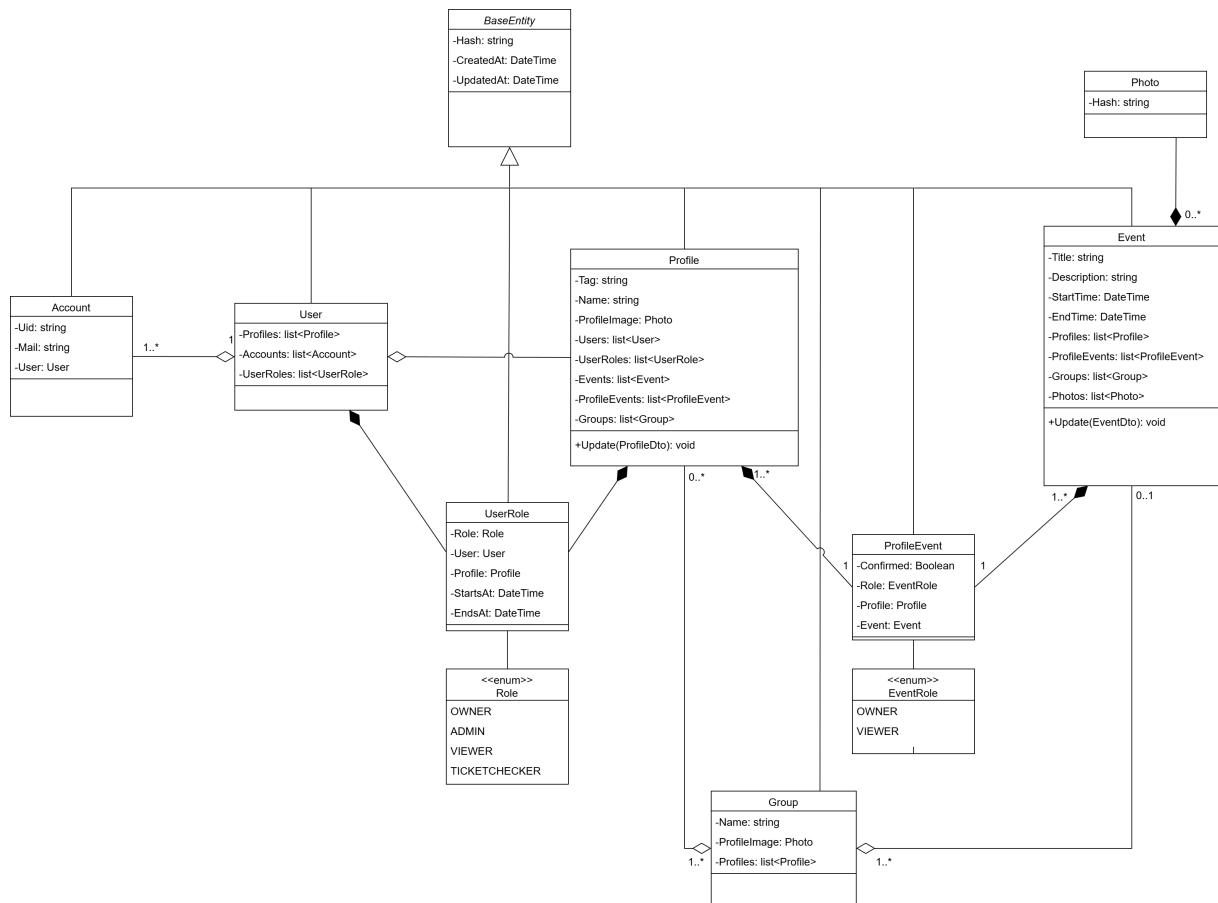


Figura 29: Modello delle classi del client

## 3.2 Cache Locale

Richiedere e ottenere dati dal server comporta ritardi che impattano sulle prestazioni. Se la tecnologia del dispositivo lo permette, è utile salvare copie delle informazioni usate più frequentemente nella memoria locale del dispositivo. Nel momento di una interazione che richiede la ricerca dei dati, si possono fornire le informazioni disponibili in copia, per poi eventualmente aggiornarle se si rivelassero imprecise. Questo permette un tempo di risposta apparente minore e una migliore esperienza utente.

### 3.2.1 Creazione della memoria locale

Non tutti i dispositivi permettono una gestione della memoria a lungo termine. In particolare, i browser web non presentano questa funzionalità, ma i dispositivi mobili e i programmi sì.

Per motivi di prestazioni, il framework Flutter nativamente mantiene lo stato di un componente solo il tempo strettamente necessario per la sua esecuzione. Il tempo di vita dello stato coincide normalmente con quello del componente a cui è associato. La normale persistenza degli elementi e dei dati ha durata limitata. Si rende necessaria la creazione e la gestione di una cache locale che permetta di mantenere i dati ricevuti dal server anche in seguito al termine di servizio dei componenti.

La cache locale viene suddivisa in base alle classi logiche del dominio, ed è accessibile tramite servizi dedicati. In particolare, grazie a componenti di tipologia provider è possibile aggiornare in automatico le parti dell'applicazione interessate dalle modifiche. Al termine di una richiesta dati al server, il provider viene notificato, salvando il dato in memoria e scatenando un aggiornamento a catena sui componenti interessati.

La cache deve essere unica e disponibile in tutto il programma. I provider attraverso

AuthProvider	UriProvider	EventProvider
-instance: AuthProvider -_auth: FirebaseAuth	-instance: UriProvider -uri: String	-instance: EventProvider -events: list<Event>
+login(String, String): void +register(String, String): void	+setUri(String): void +getUri(): String	+addAll(list<Event>): void +add(Event): void +update(Event): void +updateEvent(Event): void
UserProvider	ProfilesProvider	
-instance: UserProvider -user: User	-instance: ProfilesProvider -profiles: list<Profile>	
+updateUser(User): void	+addAll(list<Profile>): void	

Figura 30: Classi provider all'interno dell'applicazione

cui si realizza la cache locale seguiranno il pattern singleton, che garantisce l'unicità dell'elemento all'interno del programma. Una volta creati all'avvio del programma, infatti, tutti gli elementi dell'applicazione avranno accesso agli stessi dati, quando necessario.

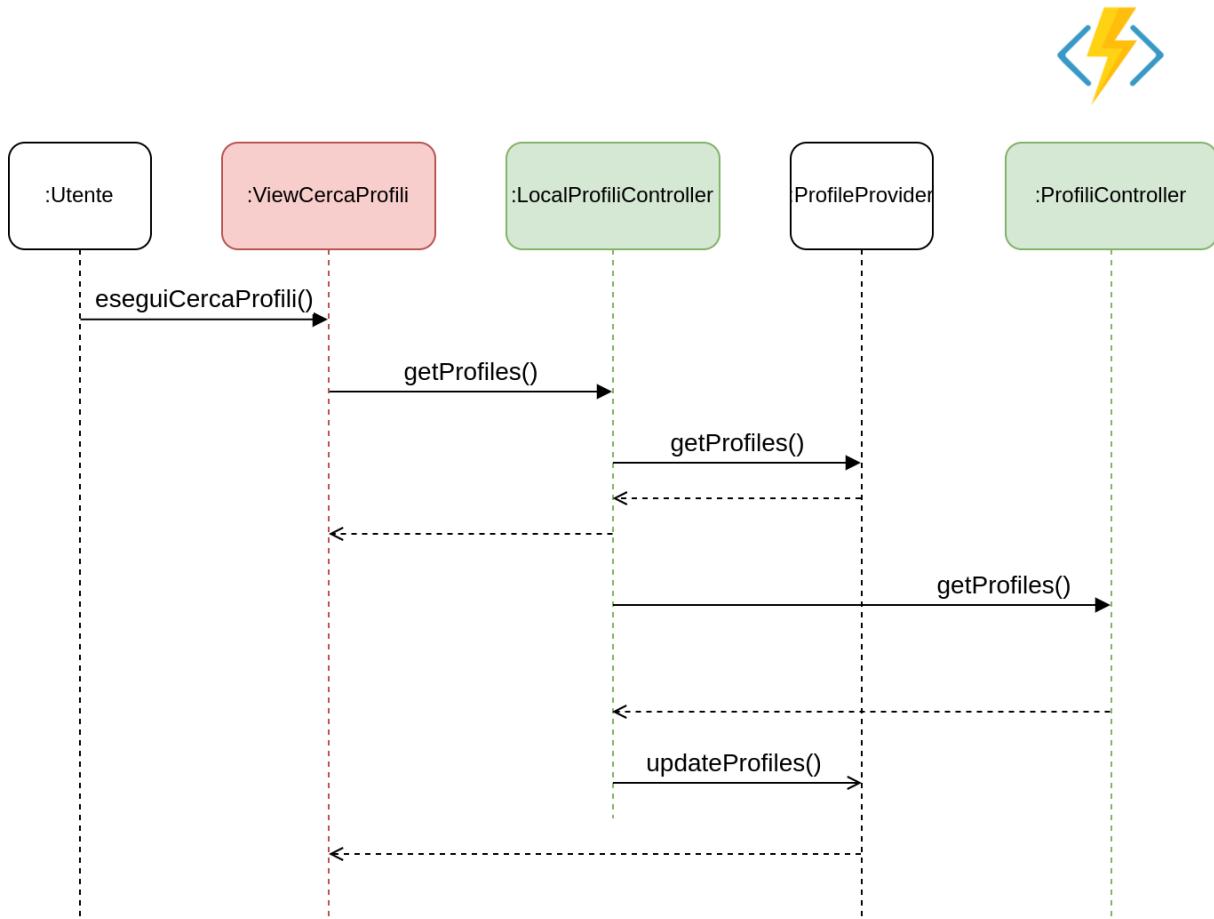


Figura 31: Esempio di interazione logica tra i componenti del client

eventualmente salvataggio delle modifiche offline per caricarle quando si torna online.

### 3.2.2 L'allineamento con la memoria centrale

il client ha la responsabilità di tenere allineata la propria cache ai valori della memoria centrale. Per quanto possa fare temporaneamente affidamento sulle risorse che ha salvato in cache per ridurre i tempi di risposta con l'utente, la loro validità dipende dalla certezza che corrispondano ai dati ufficiali.

Ogni elemento avrà associato l'ultimo momento in cui è stato aggiornato. Allo stesso modo, viene salvato l'ultimo momento in cui è stata attivamente effettuata una richiesta esplicita di aggiornamento. Ad ogni avvio dell'applicazione il client invierà una richiesta al server per ricevere tutti gli elementi che hanno subito modifiche successivamente al momento dell'ultimo aggiornamento.

La necessità di avere aggiornati tutti gli elementi della cache nel minor tempo possibile dipende anche dalla tipologia dell'elemento. Bisogna identificare gli elementi per il quale l'allineamento è critico per il corretto funzionamento dell'applicazione, come invece quelli che svolgono ruoli secondari e possono essere aggiornati anche in un secondo momento. Ad esempio, la modifica della durata di un evento è necessario venga rilevata il prima possibile, mentre la modifica della foto associata ad un profilo ha vincoli di aggiornamento in memoria locale molto più rilassati.

Nel caso di dati critici si implementa un meccanismo che esegue, periodicamente, una richiesta degli elementi modificati nell'arco di tempo dall'ultimo aggiornamento, per poi aggiornare il loro valore. Invece, nel caso di elementi secondari, si possono richiedere allineamenti direttamente nel momento in cui vengono posti espressamente in attenzione dato l'utilizzo dell'app.

### **3.3 Aggiornamenti**

Data la natura condivisa dell'applicazione risulta fondamentale che gli utenti siano aggiornati in tempo reale sulle modifiche applicate agli eventi, oltre ad essere una prerogativa di tutte le applicazioni moderne, anche per migliorare la user experience. Lo spostamento di un appuntamento, la conferma di una presenza o la modifica del luogo di appuntamento sono elementi critici che è bene che gli utenti vengano informati il prima possibile.

Il cloud fornisce strumenti per il supporto e lo sviluppo di queste funzionalità, ma, oltre a dover individuare la tecnologia più adatta, bisogna anche essere in grado di integrarla nel resto del progetto.

#### **3.3.1 Scelta della tecnologia**

La comunicazione con il server finora implementata si basa sul protocollo Hypertext Transfer Protocol (HTTP). HTTP prevede un fornitore di servizi (il server) mettere a disposizione una porta ad un indirizzo fisso rimanendo in attesa di eventuali utilizzatori (client) che, interfacciandosi attivamente alla porta disponibile, espongono le loro richieste. La riduzione delle comunicazioni al minimo indispensabile, oltre a non richiedere al server alcuna conoscenza del client, rende il protocollo pratico e scalabile.

Questa dinamica però impedisce ai client di essere notificati di eventuali modifiche apportate, a meno di richieste periodiche frequenti che comportano un sovraccarico da entrambe le parti. Inoltre, l'inversione dei ruoli non è applicabile in quanto i client cambiano costantemente l'indirizzo a loro associato, così come è impossibile distinguere se il dispositivo abbia terminato la connessione o se sia un guasto di altro tipo.

Si necessita una comunicazione che mantenga in costante contatto i client con le modifiche del server, permettendo una trasmissione attiva degli aggiornamenti. A basso livello, il protocollo che permette una comunicazione continua più adatto alle tecnologie comunemente diffuse è quello delle WebSocket. Tramite WebSocket infatti si crea un canale

diretto tra le parti che consente una comunicazione istantanea.

Alla necessità di supportare il protocollo delle WebSocket e di inviare istantaneamente i messaggi, si aggiunge la possibilità di creare molteplici canali specifici per indirizzare correttamente le comunicazioni ai soli interessati.

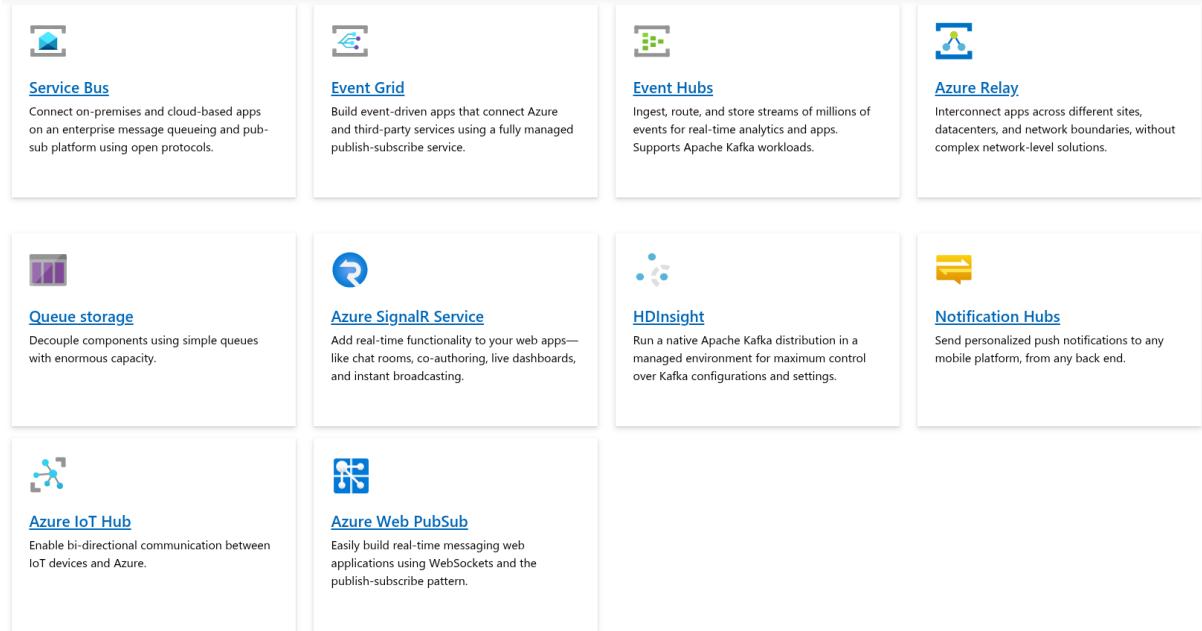


Figura 32: I servizi di comunicazione istantanea proprietari di Azure

Per individuare la tecnologia più adatta ad aggiornare gli utenti, tra le tante che offrono servizi di collegamento istantaneo tra tecnologie, è fondamentale comprendere gli scopi per cui sono nate e che problemi quindi risolvono. Infatti, ogni servizio è stato progettato per affrontare specifiche sfide che si differenziano sia per la natura dei servizi a cui si rivolgono che per modalità di approccio .

La natura degli attori per cui il servizio si specializza determina le prestazioni di scalabilità e le integrazioni supportate. Bisogna quindi considerare la località e la natura delle risorse: in-premise o sul cloud, se appartengono alla stessa piattaforma o se devono comunicare internamente. Ad esempio, un servizio pensato per collegare tantissimi dispositivi distribuiti con limitato potere computazionale, come nel caso dell'Internet of Things, fornirà supporto a connessioni esterne e a protocolli standard, e prevederà un'elevata quantità di richieste di limitate dimensioni e frequenza. Viceversa, la necessità di creare una comunicazione tra un numero ristretto di server con prestazioni elevate comporta la creazione di flussi di dati importanti, magari gestiti internamente all'ambiente cloud, astraendo la tecnologia necessaria.

I servizi si differenziano però anche per le caratteristiche delle connessioni gestite. Proprietà fondamentale è la natura delle comunicazioni. I canali possono essere infatti unidirezionali, permettere la comunicazione da entrambe le parti o implementati come flussi di eventi, in cui le comunicazioni possono essere inviate e ricevute da molteplici attori, senza che il destinatario sia noto al mittente. Inoltre, alcuni servizi offrono la possibilità di individuare categorie di clienti specifiche a cui eventualmente inviare notifiche mirate. Infine, bisogna prendere in considerazione la necessità di persistenza delle comunicazioni, che fornisce, oltre all'aggiornamento in tempo reale, anche la possibilità di recuperare modifiche passate.

Nel progetto si necessita di un servizio che supporti le WebSocket e che permetta di creare una molteplicità di canali unidirezionali differenti. In particolare, deve essere il più indipendente possibile dagli attori con cui comunica per poter garantire il maggior supporto possibile. Dovendo coprire solo le notifiche di aggiornamento, senza responsabilità di rintracciabilità dei dati, la presenza della persistenza non è necessaria.

Gratuito per le prime 20 connessioni, ma eventualmente scalabile per soddisfare ulteriori carichi, il servizio individuato per la gestione delle notifiche in tempo reale è Azure Web Pub Sub (AWPS). Permette infatti la creazione di canali tramite WebSockets e l'integrazione con le Azure Functions. Supporta la creazione di canali, sia unidirezionali che bidirezionali, su cui pubblicare eventi, a cui gli utenti possono collegarsi per ricevere gli aggiornamenti. Non prevede l'utilizzo di persistenza ma gestisce completamente la scalabilità e l'affidabilità del sistema.

### 3.3.2 Integrazione

L'integrazione di Azure Web Pub Sub deve avvenire sia con il server che con i devices degli utenti. Seguendo il modello publish subscribe, ogni client si connetterà ad un canale in sola lettura, ricevendo tutti i dati che verranno pubblicati su di esso. Il server avrà il compito di interfacciarsi con il servizio per pubblicare i dati sui canali interessati.

La scelta della definizione del canale deriva da un'ulteriore analisi del dominio. Il soggetto interessato alle modifiche sottoposte a notifica è il profilo. Se però si creassero i canali in relazione ai profili ogni dispositivo (che riassume l'interazione di un utente) dovrebbe mantenere una connessione per ogni profilo collegato all'utente. La creazione di un

canale per ogni device allo stesso modo risulta estremamente inefficiente, in quanto, oltre ad introdurre nuovi requisiti per garantire la tracciabilità dei dispositivi, ne richiederebbe di creazione e gestione in numero elevato. Per queste ragioni i canali verranno creati uno per utente, garantendo inoltre che gli unici utenti a ricevere le notifiche ne possiedano effettivamente l'accesso adeguato.

A seguito di una richiesta che comporta la notifica ai profili interessati, il server avrà il compito di interfacciarsi con AWPS per affidargli le comunicazioni relative. Tuttavia AWPS non supporta la capacità di unire gli elementi in base alle loro relazioni, per cui la responsabilità di trovare gli utenti interessati ricade sul server. Ad esempio, la modifica di un evento comporta la notifica a tutti i profili relativi, e quindi una comunicazione a tutti gli utenti che ne possiedono i permessi di notifica per la particolare azione su detti profili.

L'operazione di ottenimento degli utenti coinvolti data l'azione svolta (nell'esempio, la modifica di un evento) verrà eseguita in un'altra Azure Functions dedicata che si occuperà poi, per ogni utente coinvolto, di comunicare al server AWPS il messaggio da notificare. L'eventuale fallimento della operazione viene inserito tra i log e risulterà durante i monitoraggi, senza coinvolgere la funzione principale.

La ricezione della notifica sul dispositivo dell'utente può comportare la richiesta al server dell'elemento modificato. La scelta di recuperare i dati tramite il server invece di includere i dati direttamente all'interno della notifica permette di uniformare il formato delle notifiche, semplificando la loro gestione e velocizzando l'invio; di diminuisce il volume dei dati trasmessi tramite WebSocket, garantendo la scalabilità delle notifiche; di migliorare la gestione degli errori da parte del server rendendoli più affidabili e di garantisce la possibilità di recuperare solo l'ultimo aggiornamento, riducendo il consumo totale dei dati.

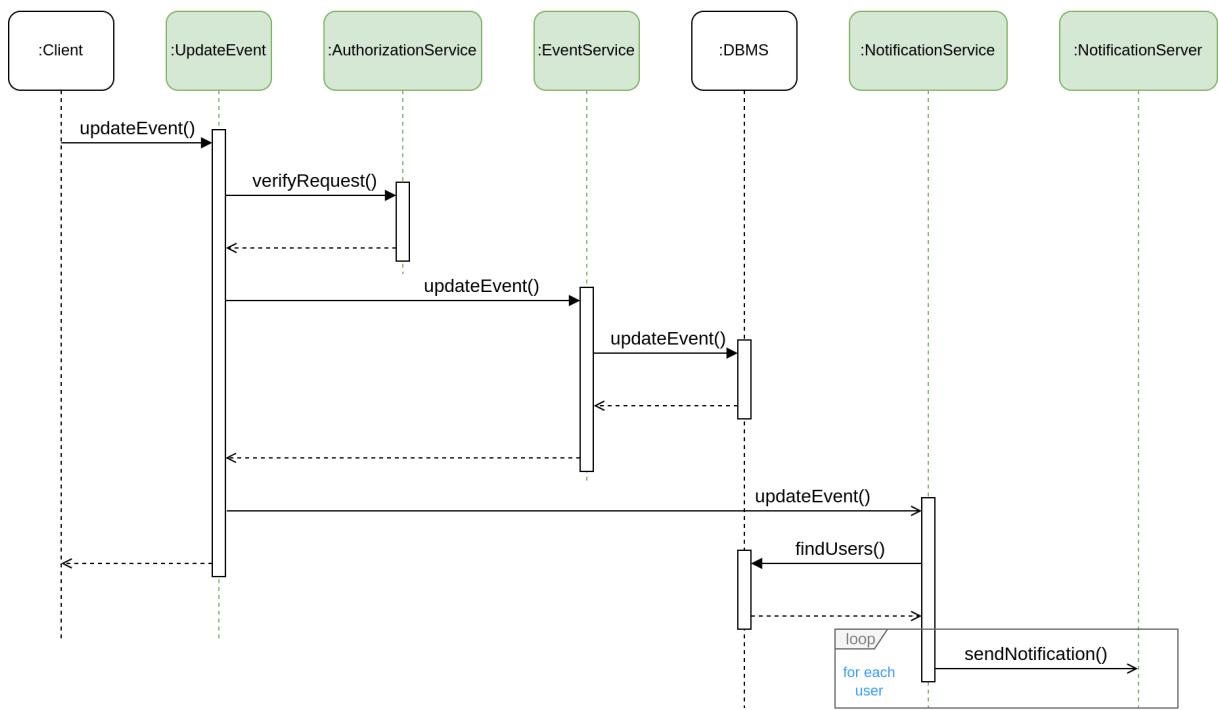


Figura 33: Interazione delle Azure Functions con AWPS

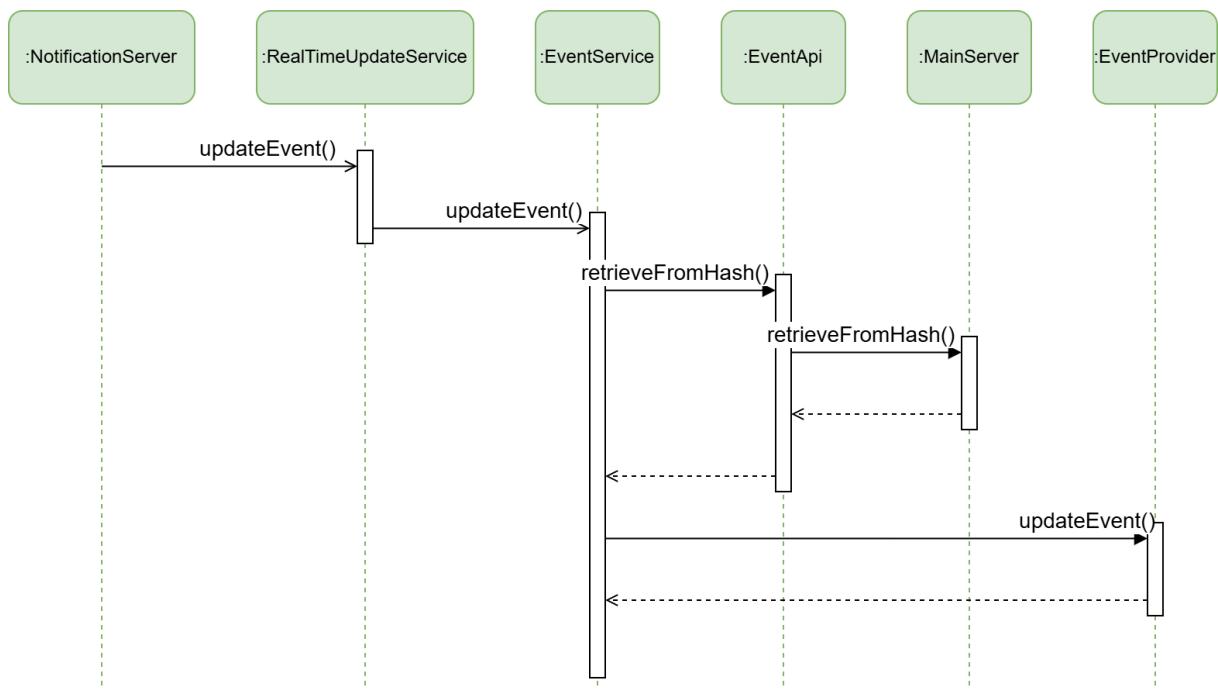


Figura 34: Interazione tra AWPS e un client

## 4 Gestione dei dati multimediali

I file multimediali rappresentano un elemento centrale all'interno di un'applicazione orientata alla condivisione sociale, contribuendo significativamente all'esperienza utente e all'interazione tra i partecipanti. La possibilità di acquisire e condividere contenuti visivi, come immagini e video, consente di documentare eventi e attività, favorendo una memoria collettiva e rafforzando il legame tra gli utenti. In particolare, l'integrazione di materiale multimediale associato a eventi condivisi permette di preservare una rappresentazione più completa e dettagliata dell'esperienza vissuta, migliorando l'engagement e la partecipazione all'interno della piattaforma, rappresentando uno dei punti di forza dell'applicazione.

Tuttavia, la gestione dei file multimediali introduce complessità operative sia per gli utenti sia per il sistema stesso. Da un lato, la selezione e l'invio dei contenuti possono rappresentare un onere significativo per l'utente, aumentando l'attrito nell'utilizzo dell'applicazione. Per ottimizzare il processo e migliorare l'usabilità, è essenziale semplificare al massimo l'interazione richiesta, automatizzando il recupero dei dati e limitando il ruolo dell'utente alla semplice conferma dei contenuti selezionati. Questo approccio non solo semplifica l'esperienza d'uso, ma la rende più intuitiva e fruibile, contribuendo anche a un incremento del tasso di adozione e della frequenza di utilizzo dell'applicazione.

Parallelamente, la memorizzazione e il trasferimento di file multimediali pongono sfide significative a livello infrastrutturale, in quanto tali dati presentano un impatto rilevante sulle risorse computazionali e sulla gestione dello storage. Il volume elevato di richieste di caricamento e accesso ai file può infatti compromettere le prestazioni del sistema, introducendo ritardi causati dal traffico di azioni che potrebbero influire negativamente sulle altre operazioni dell'applicazione. Per garantire un'archiviazione efficiente e scalabile, è quindi necessario implementare una strategia di gestione della memoria che separi il salvataggio dei file multimediali dal database principale, evitando di sovraccaricare il server applicativo. Questa soluzione deve inoltre garantire un aggiornamento tempestivo delle informazioni, assicurando la sincronizzazione tra i dati archiviati e le modifiche effettuate dagli utenti.

Per affrontare tali problematiche, un primo esame osserverà le modalità di recupero dei file multimediali, con un focus sulle tecniche di selezione e rilevamento automatico delle immagini, nonché sui vincoli normativi e di sicurezza che ne regolano l'utilizzo. In un secondo tempo l'analisi si concentrerà invece sulle strategie di salvataggio e gestione dello storage, analizzando le diverse tipologie di archiviazione disponibili e le soluzioni implementate per garantire scalabilità, efficienza e riduzione dell'impatto sulle prestazioni del sistema.

## 4.1 Recupero

L'aggiunta di immagini a un evento prevede una fase preliminare di recupero, che consente all'utente di selezionare i file multimediali da associare. Il sistema offre due modalità principali di acquisizione: la selezione manuale da parte dell'utente, che può scegliere le immagini direttamente dalla memoria del dispositivo, e in alternativa, disponibile sui dispositivi mobili, un meccanismo automatizzato, che identifica le foto scattate durante lo svolgimento dell'evento.

L'implementazione di questa funzionalità automatica di analisi della galleria per individuare i file multimediali desiderati richiede l'accesso alla galleria fotografica del dispositivo, un'operazione subordinata al consenso esplicito dell'utente. Al primo avvio dell'applicazione perciò, il sistema richiede l'autorizzazione per accedere ai file multimediali, unitamente al permesso per la gestione delle notifiche. Nel caso in cui l'utente neghi l'accesso, la richiesta verrà riproposta ogni qualvolta il sistema rilevi la necessità di accedere alla galleria per il recupero delle immagini.

Al termine di ogni evento, non appena possibile, l'applicazione avvia automaticamente un'analisi della galleria locale, individuando le immagini scattate durante tutta la durata dell'evento. Se il sistema rileva la presenza di contenuti pertinenti, ne memorizza temporaneamente i riferimenti in una memoria locale per poi inviare una notifica all'utente, informandolo del ritrovamento delle immagini. A questo punto, l'utente ha la possibilità di esaminare le immagini suggerite, escluderne alcune o confermarne l'intero set per il caricamento.

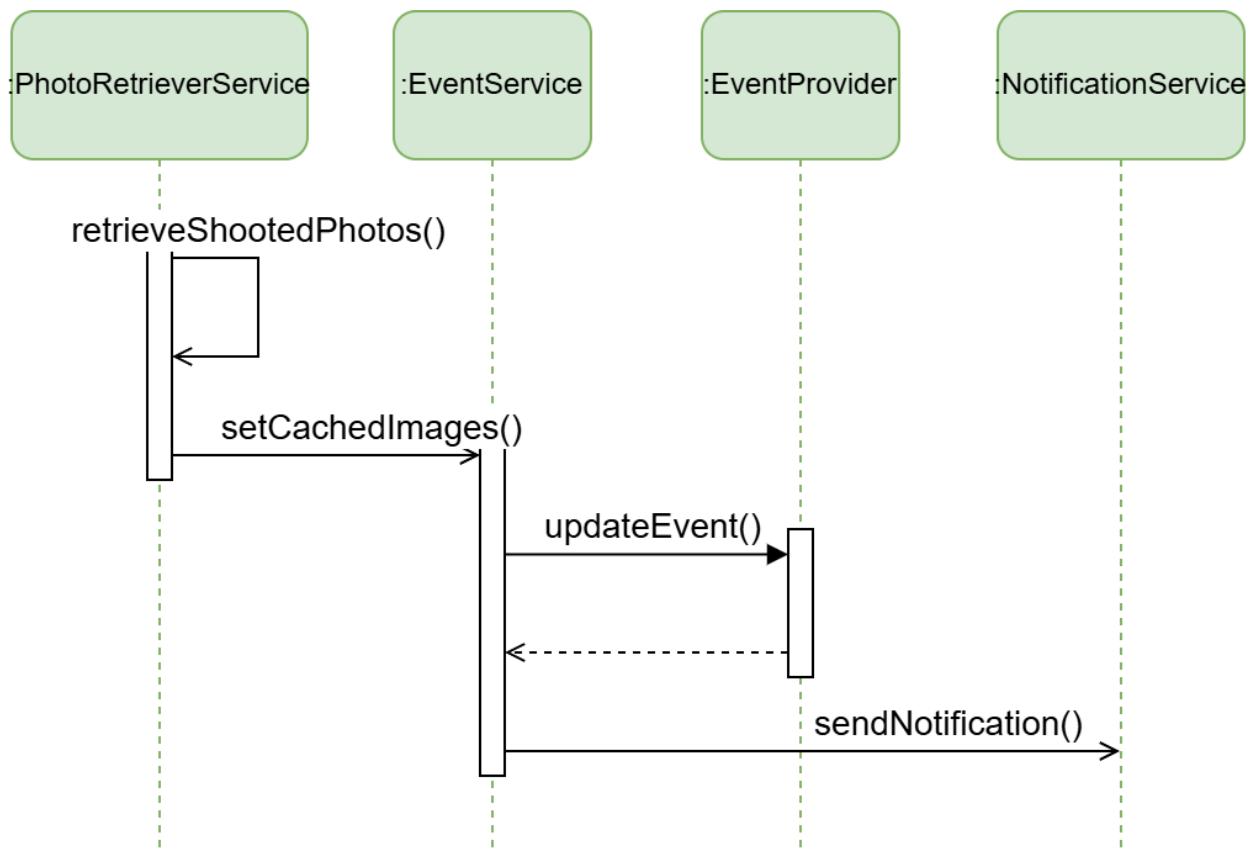


Figura 35: Interazione tra i componenti per il recupero delle immagini

Questa fase di conferma, oltre a garantire la trasparenza del servizio nei confronti dell'utente, riducendo il rischio di errori o caricamenti indesiderati, presenta anche vantaggi in termini di ottimizzazione delle prestazioni.

Da un punto di vista normativo, la procedura di recupero e selezione automatica delle immagini è esplicitamente descritta nelle condizioni d'uso dell'applicazione, alle quali l'utente deve aderire con accettazione espressa prima di utilizzare il servizio. Tuttavia, la fase di conferma dell'utente non rappresenta un obbligo giuridico, poiché la responsabilità della pubblicazione di contenuti multimediali ricade sul soggetto che realizza la fotografia. In conformità con la normativa vigente in materia di tutela dell'immagine (art. 10 c.c. e artt. 96-97 della Legge n. 633/1941) e protezione dei dati personali (Regolamento UE 2016/679 – GDPR), chi scatta una fotografia è tenuto a ottenere il consenso delle persone ritratte prima di procedere alla sua pubblicazione.

Come già affrontato nel capitolo precedente, le operazioni che coinvolgono la modifica di uno stesso componente del sistema sono soggette a vincoli di concorrenza per l'accesso alla risorsa di interesse. La conclusione di un evento condiviso tra più utenti potrebbe generare richieste simultanee per l'aggiunta di immagini associate a un medesimo evento. L'introduzione della fase di selezione introduce un ritardo nella fase di caricamento, dilatando la distribuzione temporale delle richieste e riducendo la probabilità di collisioni dovute a operazioni concorrenti sullo stesso elemento. L'attesa della conferma dell'utente prevede infatti un ritardo fisiologico tra la fine dell'evento e l'effettivo caricamento delle immagini, contribuendo a distribuire le richieste nel tempo e limitando il rischio di congestione del server dovuta a operazioni simultanee su un singolo evento.

Una volta completata la selezione da parte dell'utente, le immagini vengono inviate al server, che provvede al loro salvataggio e all'associazione con l'evento corrispondente.

## 4.2 Salvataggio

A differenza dei dati tradizionalmente scambiati all'interno del sistema, i file multimediali presentano dimensioni significativamente superiori, con diversità che si manifestano su ordini di grandezza rilevanti. Un salvataggio di tali file nel flusso di dati standard, allineandoli agli elementi logici, comporterebbe un rallentamento generale delle operazioni e un impatto significativo sulle prestazioni complessive del sistema. Per questo motivo, è necessaria una gestione della memoria specificamente progettata per l'archiviazione e il recupero di contenuti multimediali.

Inoltre, Le dimensioni delle immagini e dei video influenzano direttamente il tempo di elaborazione e il volume delle richieste, aumentando il carico computazionale su tutti i componenti del sistema. In aggiunta, la possibilità di allegare più file a un singolo evento implica che i tempi di caricamento più elevati dei file multimediali possano prolungare sensibilmente la durata delle transazioni necessarie per la modifica degli eventi, incidendo sulla reattività del sistema.

#### 4.2.1 Persistenza

La visualizzazione dei file multimediali riveste un’importanza secondaria rispetto ad altre funzionalità offerte dall’applicazione. Di conseguenza, è possibile accettare un maggiore tempo di caricamento, a condizione che ciò contribuisca a ridurre la latenza delle operazioni invece più rilevanti. Il salvataggio dei file multimediali direttamente nel database centrale comporterebbe un aumento significativo del volume delle richieste, determinando un maggiore impiego di risorse computazionali e un incremento dei tempi di caricamento. Questo fenomeno potrebbe incidere negativamente sulle prestazioni complessive del sistema, penalizzando l’esecuzione simultanea di altre operazioni.

Per ottimizzare la gestione dei file multimediali, si adotta una distinzione della relazione tra l’oggetto logico e l’evento dai dati binari che lo compongono. In questo modo, la relazione tra il file e l’evento associato viene mantenuta indipendentemente dai dati binari che lo compongono. Una volta recuperati i riferimenti ai file multimediali associati all’evento in questione, sarà possibile ottenere poi i loro contenuti binari in un secondo momento, solo quando necessario. Il modello del dominio illustrato in precedenza evidenzia la relazione logica tra gli eventi e i file associati (Photo).

Considerando la necessità e la possibilità di archiviare i file multimediali su risorse differenti dal database centrale, è fondamentale individuare la soluzione più adatta alla loro persistenza. I principali servizi cloud per l’archiviazione di file multimediali si suddividono in tre categorie: Object Storage, File Storage e Block Storage. Gli Object Storage gestiscono i file in un unico livello, con la possibilità di aggiungere metadati agli oggetti. A ciascun elemento viene associato un identificativo univoco che ne consente il recupero. L’accesso ai dati avviene tipicamente tramite API RESTful, che oltre ad offrire la possibilità di gestire i permessi, garantisce l’utilizzo su ampia scala. La presenza di un unico livello di indirizzamento permette una scalabilità pressoché illimitata, e un costo variabile in base alla quantità di dati memorizzati. I File Storage organizzano i file in una struttura gerarchica di cartelle e sottocartelle, semplificando la gestione dei file e il controllo degli accessi. Oltre a facilitare un controllo ulteriore agli utenti, questa soluzione è compatibile con protocolli di accesso particolari. Tuttavia, la sua capacità e scalabilità,

così come il costo effettivo, sono legati alla struttura dei file e alla capacità prevista dal piano selezionato. Infine, i Block Storage gestiscono la memoria tramite la suddivisione dei dati in blocchi logici, salvati separatamente e ognuno dotato di identificativo univoco. Questa tecnologia offre elevate prestazioni per il recupero e la modifica dei dati, ma i costi aumentano all’incremento della quantità di dati presenti. La scalabilità è quindi limitata alla capacità assegnata al volume. Oltre, i costi sono elevati, particolarmente riguardo moli di grandi entità.

Tra queste soluzioni, la categoria degli Object Storage risulta la più adatta alle esigenze del progetto di salvataggio dei file multimediali. La sua scalabilità illimitata consente di gestire grandi volumi di elementi con una ridotta interdipendenza tra loro. Inoltre, l’identificazione univoca di ciascun oggetto garantisce una rapida individuazione dei dati e un’efficiente risposta prestazionale a numerose richieste contemporanee.

Nel contesto di Azure, il servizio di Object Storage fornito è rappresentato da Azure Blob Storage (ABS). ABS adotta un’organizzazione centrata su Container, entità logiche che raggruppano più file multimediali e introducono un livello di indirizzamento aggiuntivo. L’accesso in lettura ai dati avviene tramite protocollo API RESTful, con autenticazione per l’aggiunta di nuovi elementi. Per ogni evento viene creato un Container dedicato, contenente le immagini corrispondenti.

Terminata la selezione dei file multimediali, prima dell’invio al server, i dispositivi client eseguono la compressione delle immagini, riducendo il consumo di banda e il volume dei dati totali trasmessi. Questa strategia consente di diminuire il carico computazionale sul server, migliorando l’efficienza complessiva del sistema.

Il server, una volta ricevuta la richiesta, esegue una verifica dei permessi di accesso necessari e procede con il caricamento delle immagini nel Container associato all’evento. Al termine dell’operazione, il database viene aggiornato con i riferimenti ai nuovi file multimediali, e gli utenti vengono notificati della modifica avvenuta.

La visualizzazione di un evento con immagini allegate comporta la richiesta parallela

da parte delle singole immagini del dispositivo client verso ABS. Le immagini vengono identificate univocamente attraverso la combinazione dell'hash dell'immagine con quello del Container associato all'evento.

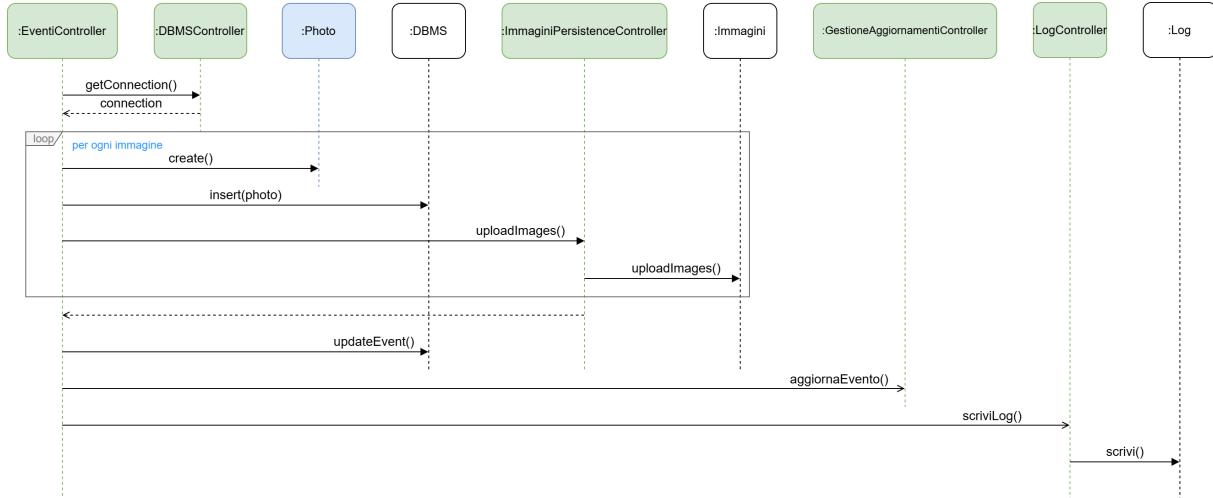


Figura 36: Interazione progettuale del server per il caricamento delle immagini

Tuttavia, l'accesso in lettura ai file multimediali in ABS risulta pubblico per impostazione predefinita, poiché la mancata definizione di ruoli comporta l'assenza di controlli esplicativi sulle autorizzazioni delle richieste. Questo aspetto viene mitigato mediante l'uso di hash randomici sufficientemente lunghi, che riducono drasticamente la probabilità di collisione. Senza la conoscenza dell'hash corretto, un accesso non autorizzato alle immagini richiederebbe tentativi casuali estremamente numerosi nella speranza di trovare una combinazione corretta, rendendo un attacco altamente improbabile. Inoltre, anche in caso di compromissione di un hash, l'accesso sarebbe limitato a una singola immagine, senza fornire ulteriori informazioni sugli altri file memorizzati.

#### 4.2.2 Concorrenza

Il caricamento delle immagini dalle Azure Functions al Container associato all'evento rappresenta un'operazione relativamente onerosa in termini di tempo, soprattutto se confrontata con le altre operazioni eseguite dal sistema. La gestione delle connessioni tra il server e il database relazionale richiede dunque scelte di dominio e sviluppo mirate a garantire un uso ottimale delle risorse, minimizzando il tempo di blocco e migliorando l'efficienza complessiva.

Un elemento centrale nel processo di invio dati è la gestione dell'hash dell'elemento Photo, che deve corrispondere al file caricato. L'hash viene generato al momento della creazione dell'oggetto, ma il suo salvataggio nel database avviene solo dopo il completamento del caricamento del file multimediale. Questa strategia evita operazioni di scrittura e cancellazione superflue, ottimizzando le prestazioni del sistema. Ne consegue che il momento della creazione dell'oggetto Photo deve essere logicamente distinto dal suo effettivo salvataggio nel database.

Per realizzare un'astrazione ad alto livello della relazione con il database, Entity Framework Core (EF Core) fornisce una rappresentazione logica degli elementi, mantenendo un collegamento con le relative controparti fisiche. Grazie a questa caratteristica, è possibile creare un oggetto senza doverlo immediatamente memorizzare, consentendo un controllo più preciso e immediato sul flusso dei dati.

La relazione uno a molti tra gli eventi e le immagini è implementata mappando fisicamente sugli oggetti Photo, i quali contengono un riferimento all'identificativo dell'evento associato. Al momento del salvataggio, viene modificata esclusivamente la tabella relativa a Photo; tuttavia, tale operazione implica un blocco in scrittura anche sull'oggetto Event. Ciò avviene poiché l'oggetto Event è coinvolto a livello logico, come dimostrato anche dalla presenza di una lista virtuale contenente le immagini associate tra i suoi attributi.

Per migliorare l'efficienza del caricamento, la trasmissione dei file multimediali verso il Container avviene in parallelo, riducendo il tempo complessivo necessario per completare l'operazione. Tuttavia, l'inserimento immediato di ciascun oggetto Photo nel database

al termine di ogni caricamento comporterebbe un rischio di conflitti sull'oggetto Event e un potenziale sovraccarico del database. Per mitigare questi problemi, gli oggetti logici Photo delle trasmissioni avvenute con successo vengono temporaneamente conservati in memoria per essere salvati successivamente in un'unica operazione.

L'inserimento simultaneo di tutti gli elementi Photo validi all'interno di una stessa transazione consente di ottimizzare l'impatto sul database, riducendo i tempi di blocco sull'oggetto Event e minimizzando il rischio di collisioni tra richieste concorrenti. Questo approccio garantisce una maggiore scalabilità e una gestione più efficiente delle risorse del sistema coinvolte.

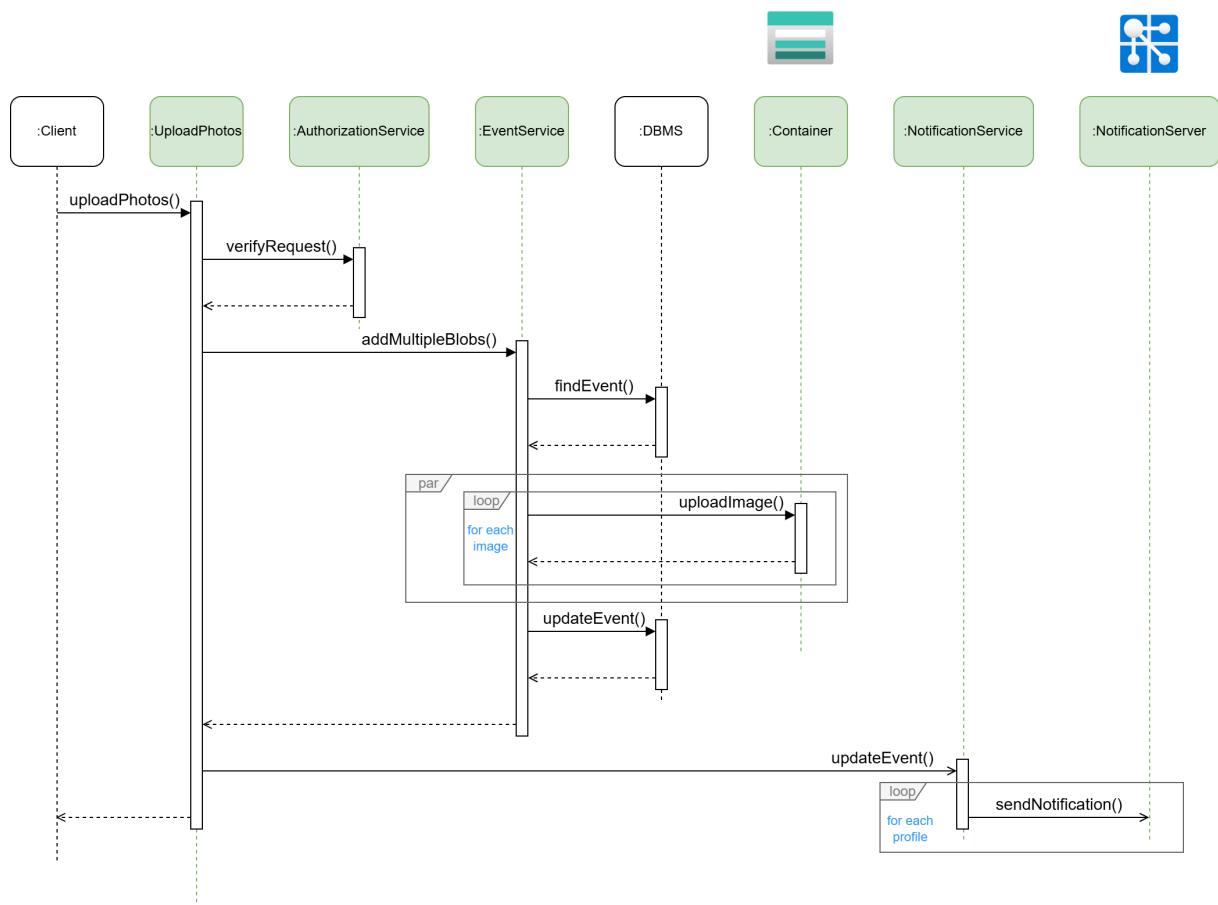


Figura 37: Interazione logica del server per il caricamento delle immagini

## 5 Risultati

### 5.0.1 Velocità in lettura

Statistics				
Load	Duration	Response time	Error percentage	Throughput
103777 Total requests	4 mins, 45 secs	158.00 ms 90th percentile response time	0 % Aggregate requests which failed	364.13 /s Request rate

Figura 38: 158 ms su una media di 364 richieste al secondo

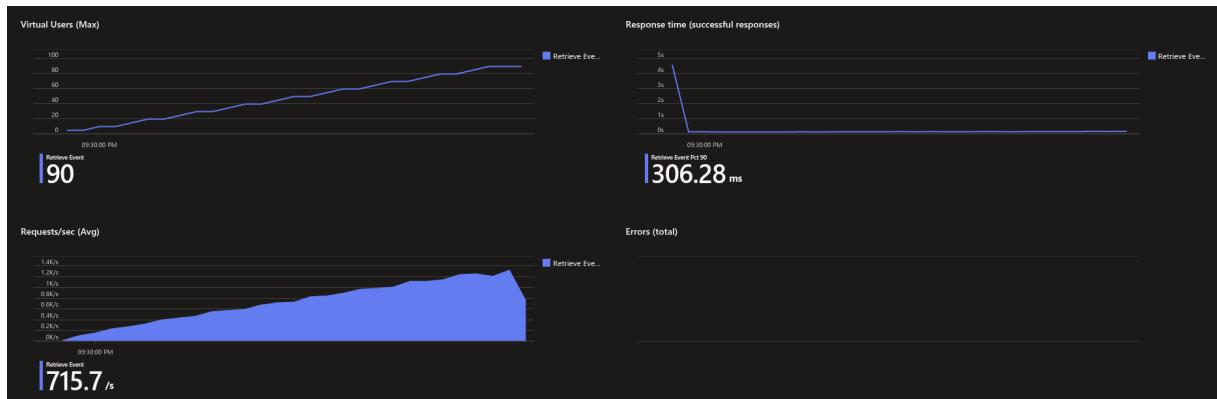


Figura 39: il tempo di risposta non varia in base al numero di richieste

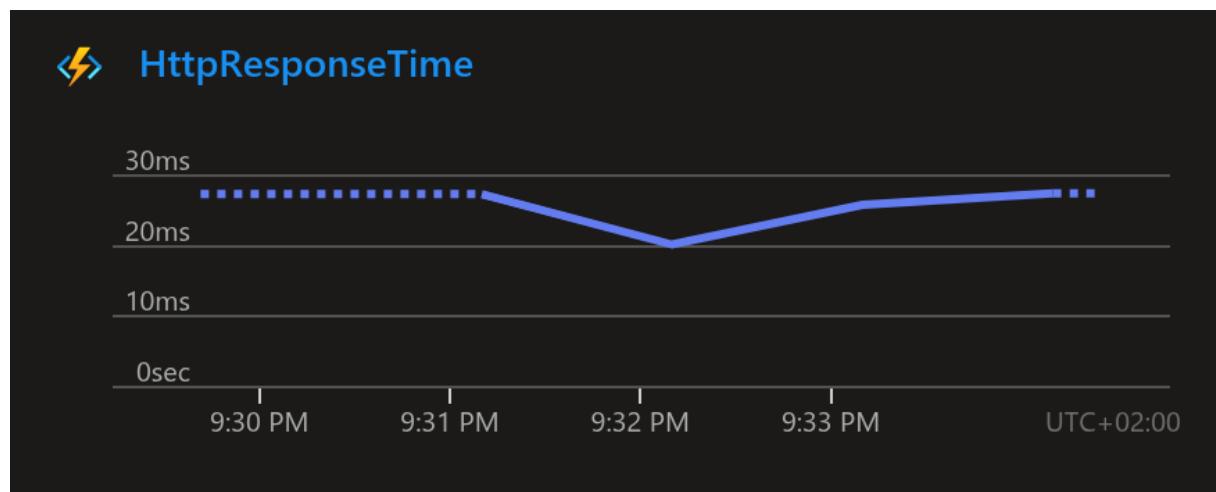


Figura 40: Dettaglio della velocità senza tempo di trasmissione

### 5.0.2 Caricamento di immagini concorrenti

Statistics				
Load	Duration	Response time	Error percentage	Throughput
2561	2 mins, 5 secs	647.00 ms	0 %	20.49 /s
Total requests		90th percentile response time	Aggregate requests which failed	Request rate

Figura 41: 2 MB di immagini in 600 ms 20 volte al secondo

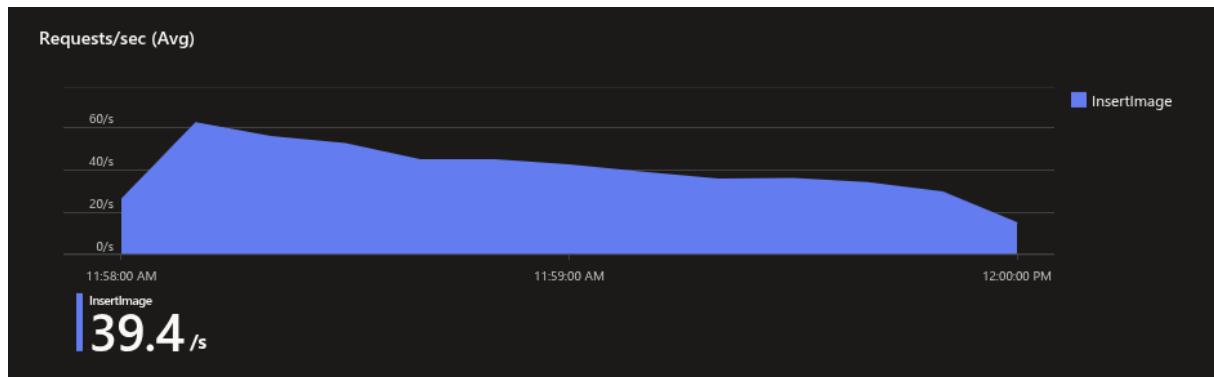


Figura 42: Andamento delle richieste

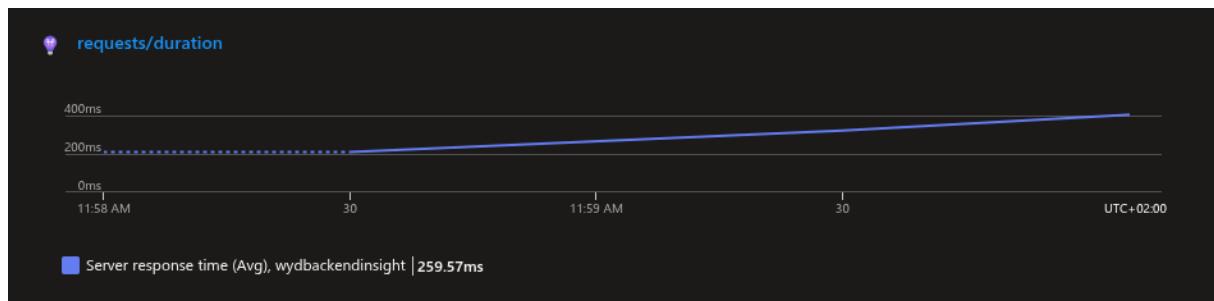


Figura 43: Dettaglio della velocità richiesta dal server

## 6 Conclusione

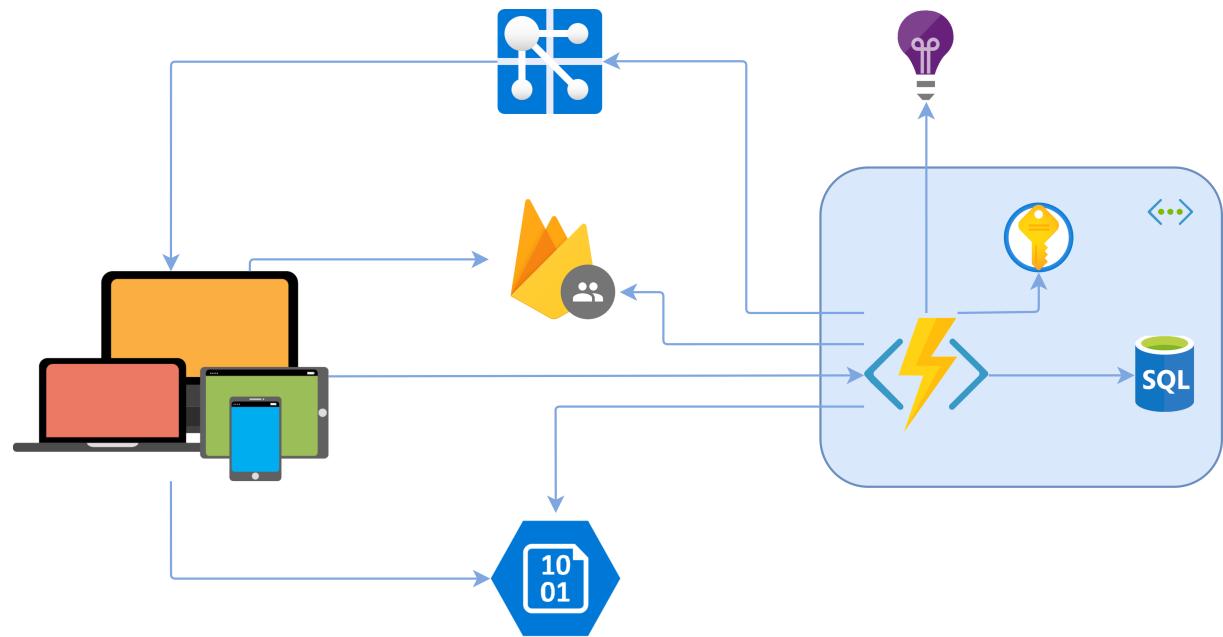


Figura 44: Grafico dell'architettura finale

## 6.1 Sviluppi futuri

Gli sviluppi futuri potranno comprendere, in base a decisioni di marketing:

- La visualizzazione degli impegni degli altri profili
- L'implementazione di una chat per ogni gruppo
- Sviluppo di strumenti utili all'organizzazione dei gruppi, quali:
  - form per combinare le disponibilità reciproche
  - appunti condivisi(liste della spesa o note su chi porta cosa)
  - calcolo delle spese compiute da ciascun componente
- La creazione di profili pubblici che possono essere seguiti
- La creazione di eventi pubblici
- Una funzionalità di ricerca degli eventi o dei profili pubblici
- Supporto alla gestione di prenotazione e organizzazione degli eventi, dalle liste di attesa alla vendita dei biglietti
- La possibilità per le aziende di gestire in locale il proprio server e i relativi dati

## **7 Fonti bibliografiche e sitografia**

Object Management Group, OMG Unified Modelling Language Version 2.5.1, December 2017, <https://www.omg.org/spec/UML/2.5.1/PDF>