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 Luglio 2025

Anno Accademico 2025/2026

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, anche con fondi limitati.

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 ed implementative adottate nello sviluppo dell'applicativo in questione, evidenziando l'impatto dell'integrazione delle risorse cloud sul risultato finale.

Indice

Introduzione **1**

Organizzazione dei capitoli 3

1 La realizzazione della struttura centrale **4**

1.1 Lo sviluppo del client utente 6

1.1.1 La scelta del framework di sviluppo 7

1.1.2 La realizzazione delle interfacce grafiche 8

1.1.3 L'implementazione della logica applicativa 13

1.1.4 La distribuzione del codice verso i dispositivi utente 19

1.2 La creazione del server principale 22

1.2.1 La scelta del servizio adatto 23

1.2.2 Le scelte progettuali derivate dall'utilizzo delle Azure Functions . . 25

1.2.3 L'implementazione della logica applicativa 28

1.3 Autenticare le richieste: la scelta del servizio e la sua integrazione 33

1.4 Uno sguardo sulla sicurezza: segreti e protocolli 35

1.5 Il monitoraggio dei servizi 36

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 conteporanea sia l'invito sia la partecipazione.

In risposta a tali richieste è stata creata Wyd, un'applicazione che permette agli utenti 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 confermati 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. 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, la stabilità del programma deve essere garantita da un'infrastruttura affidabile e scalabile. La persistenza deve essere modellata per fornire alte prestazioni sia in lettura che in scrittura indipendentemente dalla quantità delle richieste, rimanendo però aggiornata e coerente. La funzionalità di condivisione degli eventi richiede inoltre l'aggiornamento in tempo reale verso tutti gli utenti coinvolti. Infine, il caricamento ed il salvataggio delle foto aggiungono 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.

Capitolo 1

La realizzazione della 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.

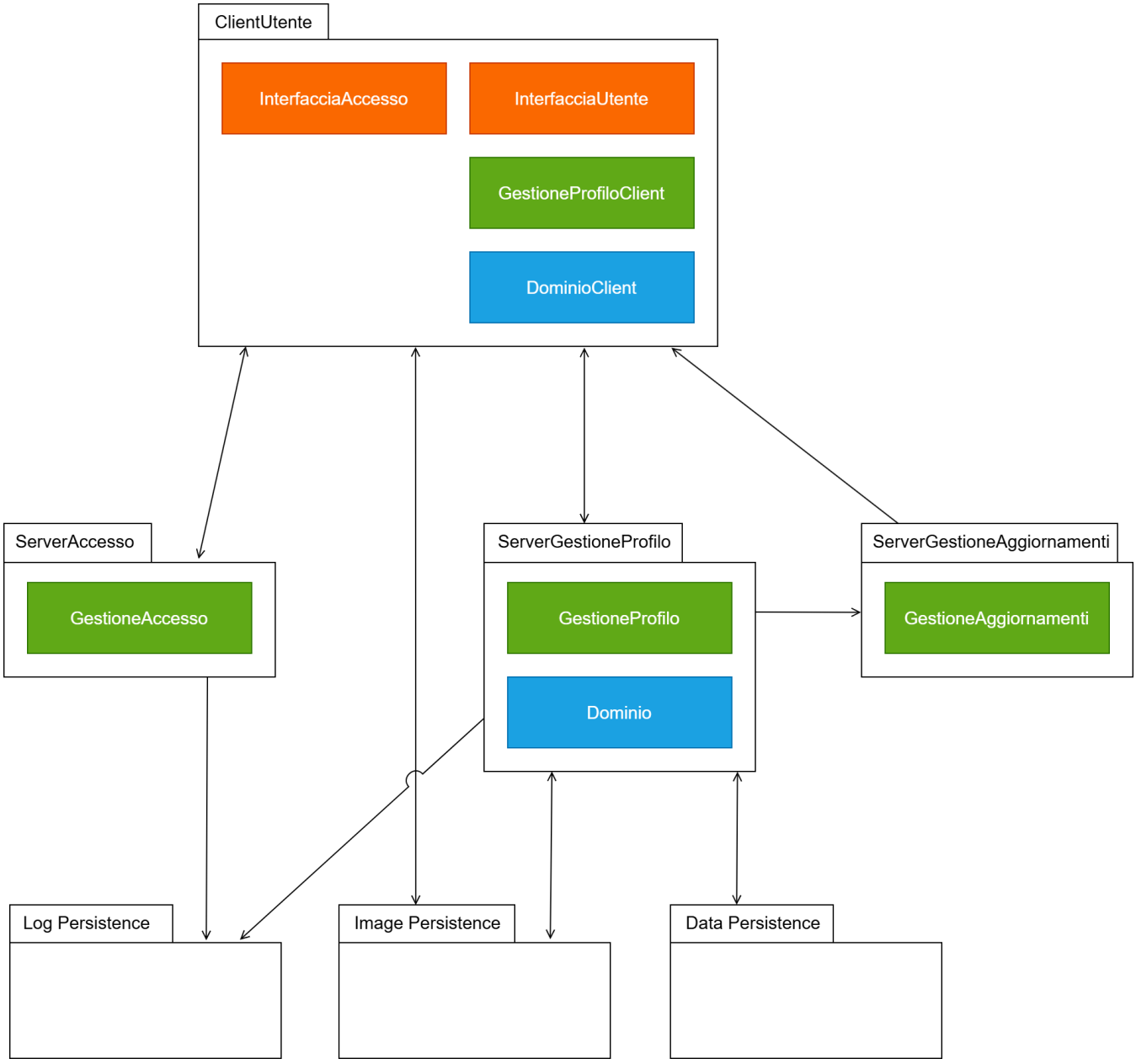


Figura 1.1: Struttura e responsabilità delle parti del progetto

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 a un servizio apposito, per migliorare sia le prestazioni che la sicurezza.

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.

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.

1.1 Lo sviluppo del client 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 mul-

timediali 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.

1.1.1 La scelta del 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 mantenibi-

Figura 1.2: Il logo di Flutter

le.

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.

1.1.2 La realizzazione delle 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.

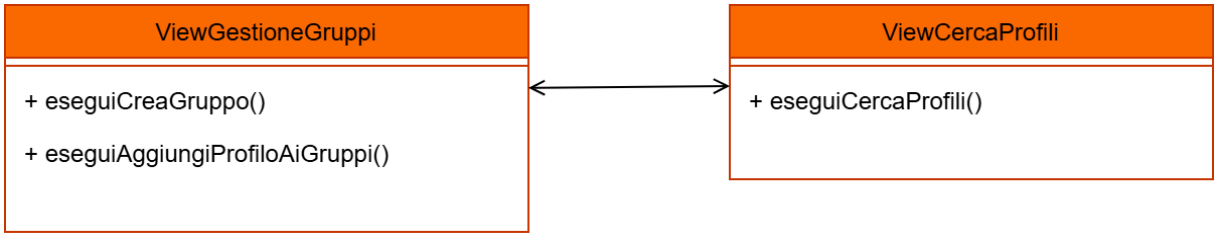
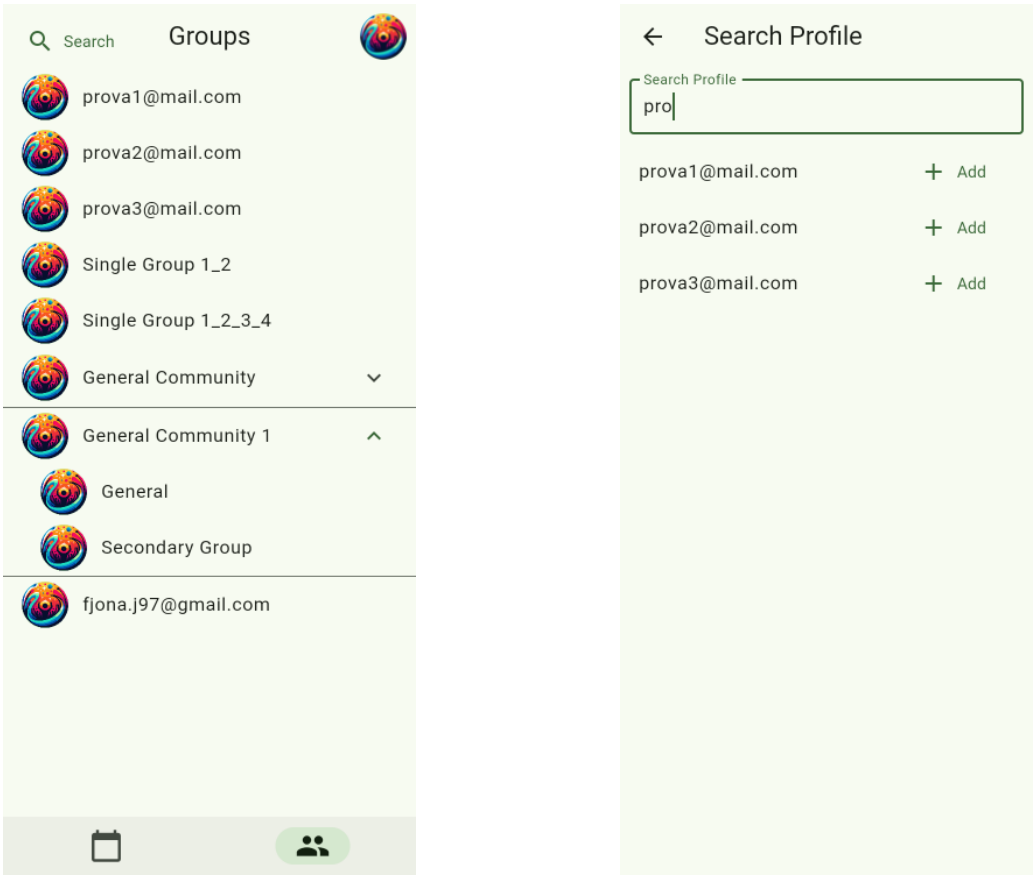


Figura 1.3: Diagramma di dettaglio delle interfacce di gestione dei gruppi

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 1.4: 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.

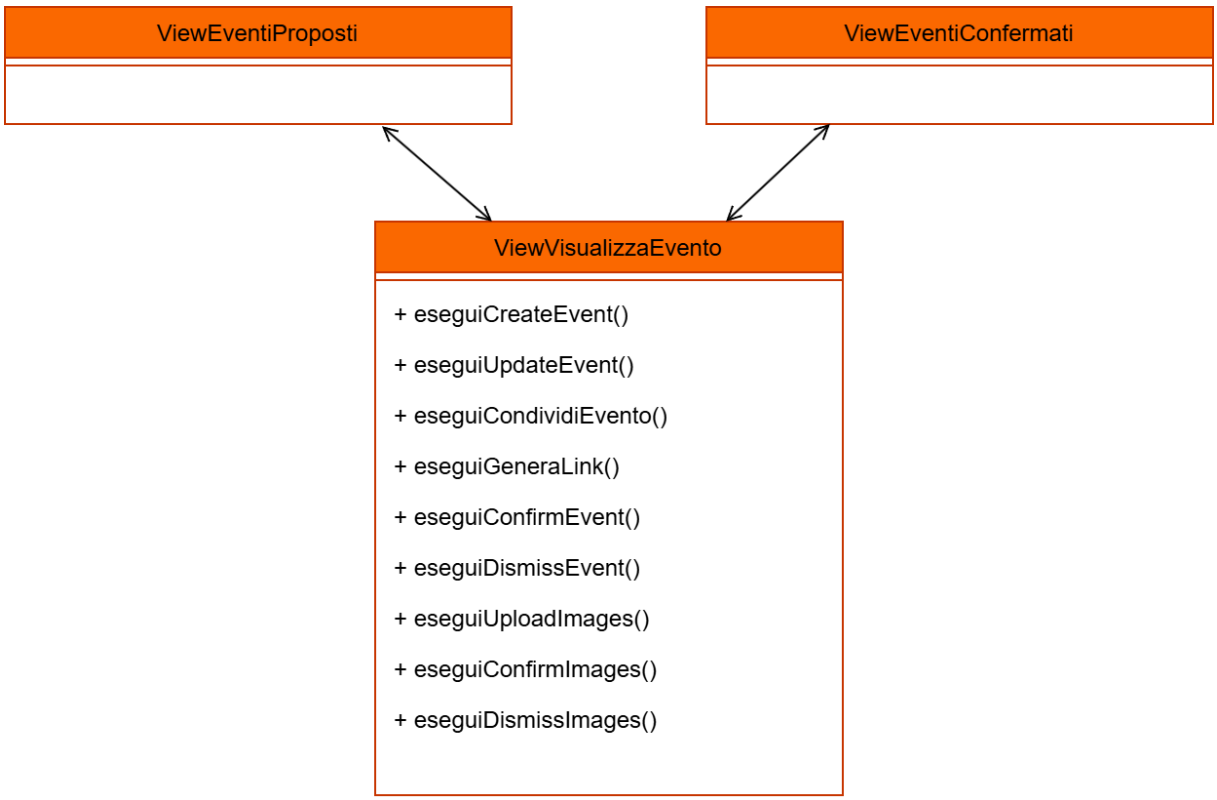
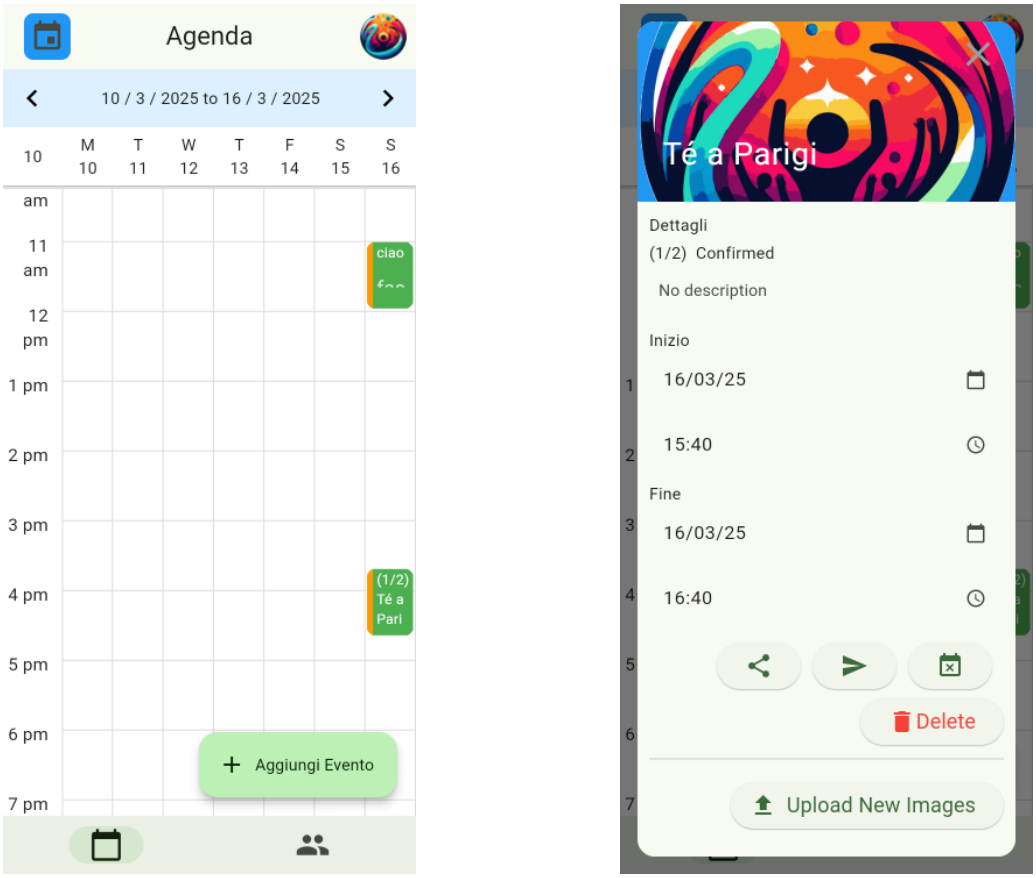


Figura 1.5: Diagramma di dettaglio delle interfacce di visualizzazione eventi

Queste due funzionalità sono al centro del servizio del sistema, ed è quindi essenziale che l'interfaccia proposta sia veloce ma sopratutto 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 1.6: Schermate degli eventi

1.1.3 L'implementazione della logica applicativa

Ogni interazione con l'utente scatena una qualche forma di elaborazione di dati. La visualizzazione di qualunque componente comporta il ritrovamento delle informazioni, 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 a 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.

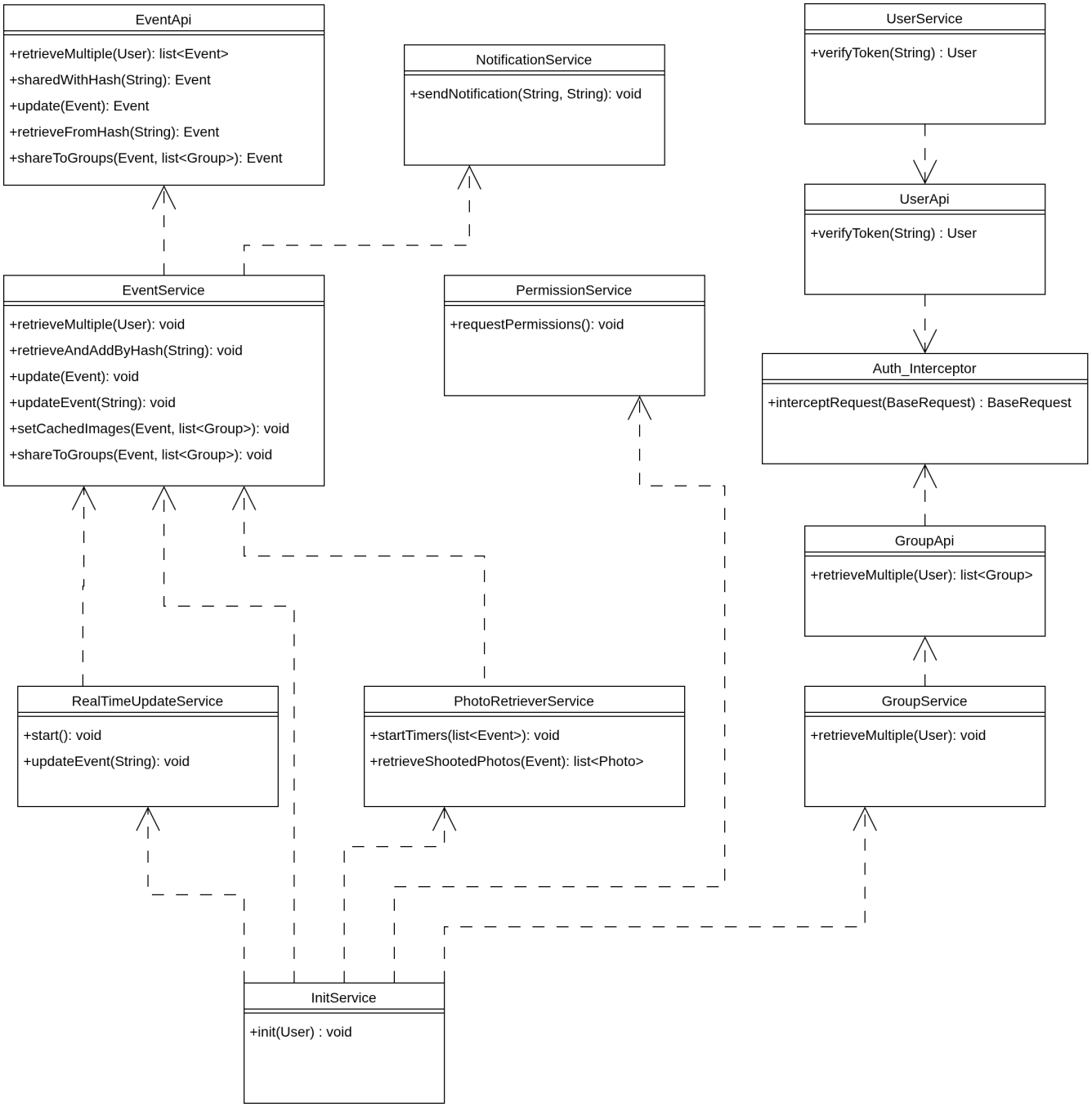


Figura 1.7: Modello delle classi del client

Non tutte le funzionalità sono però correlate direttamente al dominio. Per questo motivo si creano servizi ausiliari dedicati, anch'essi separatati 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 possegga, 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 a 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 a un canale e rimane in attesa di eventuali messaggi.

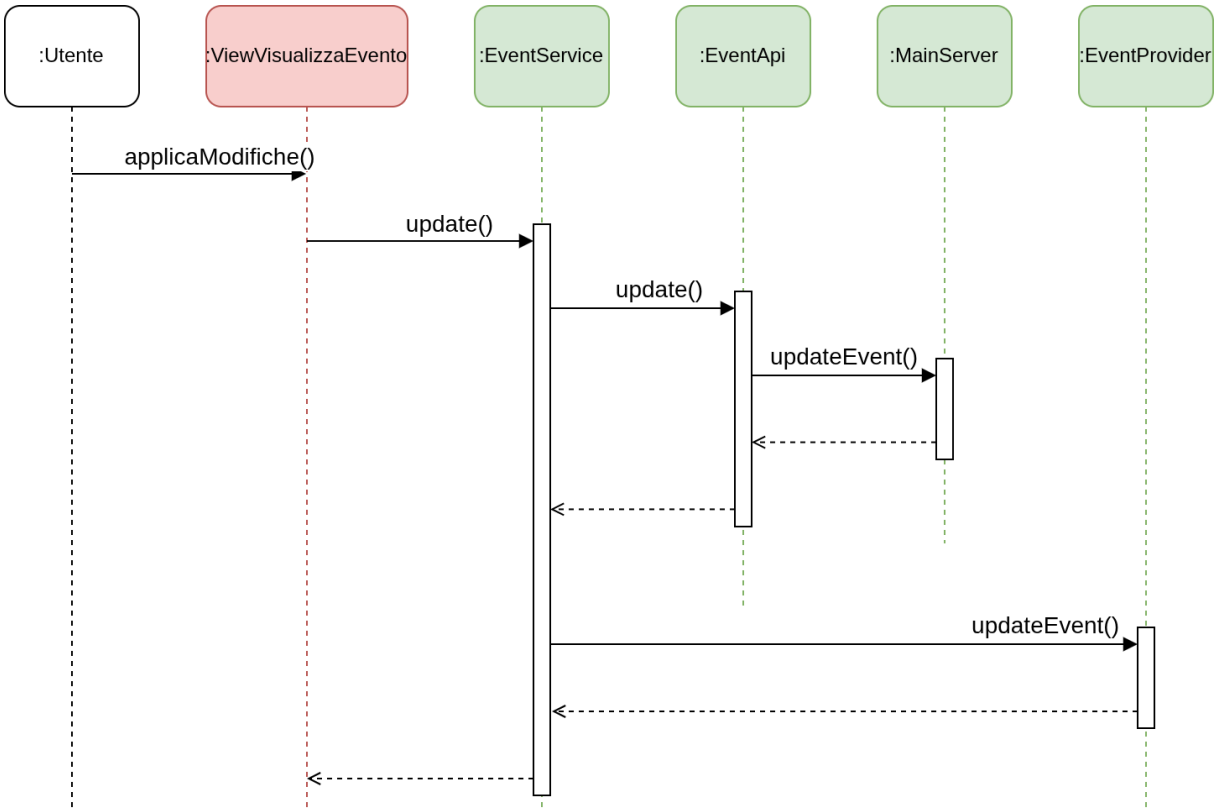


Figura 1.8: Diagramma di sequenza della modifica di un evento

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.

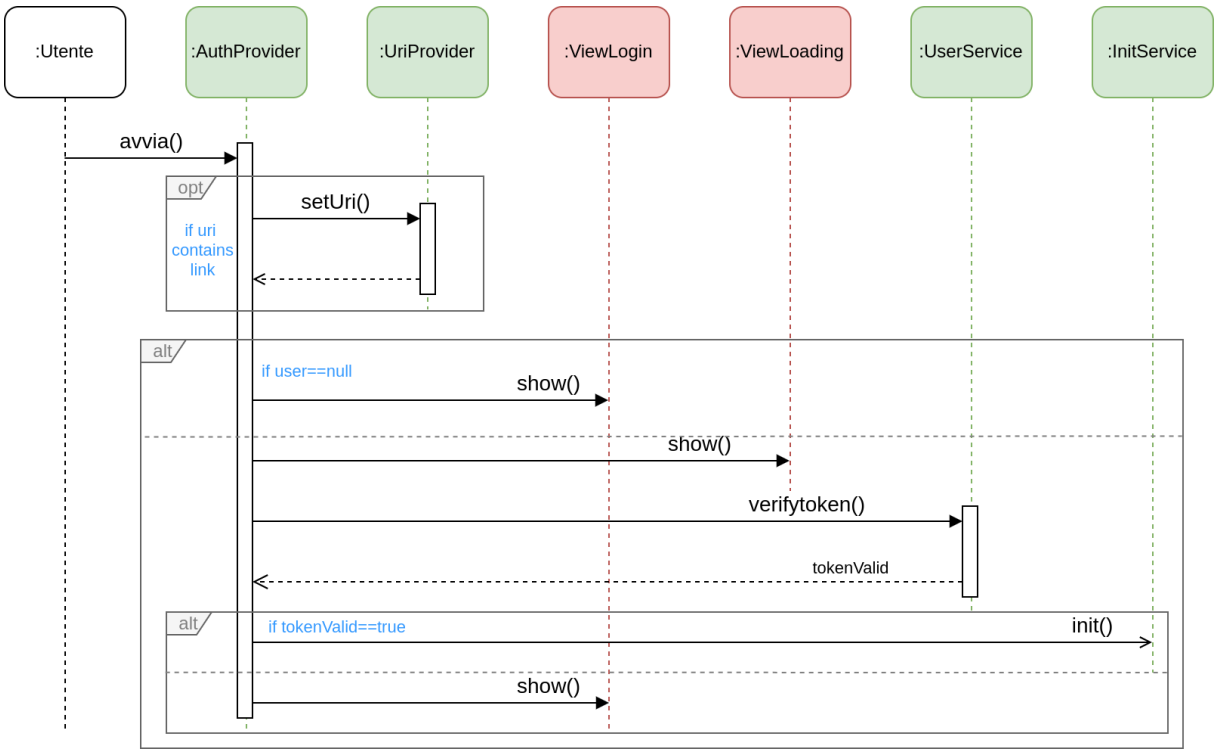


Figura 1.9: Diagramma di sequenza dell'avvio dell'applicazione

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.

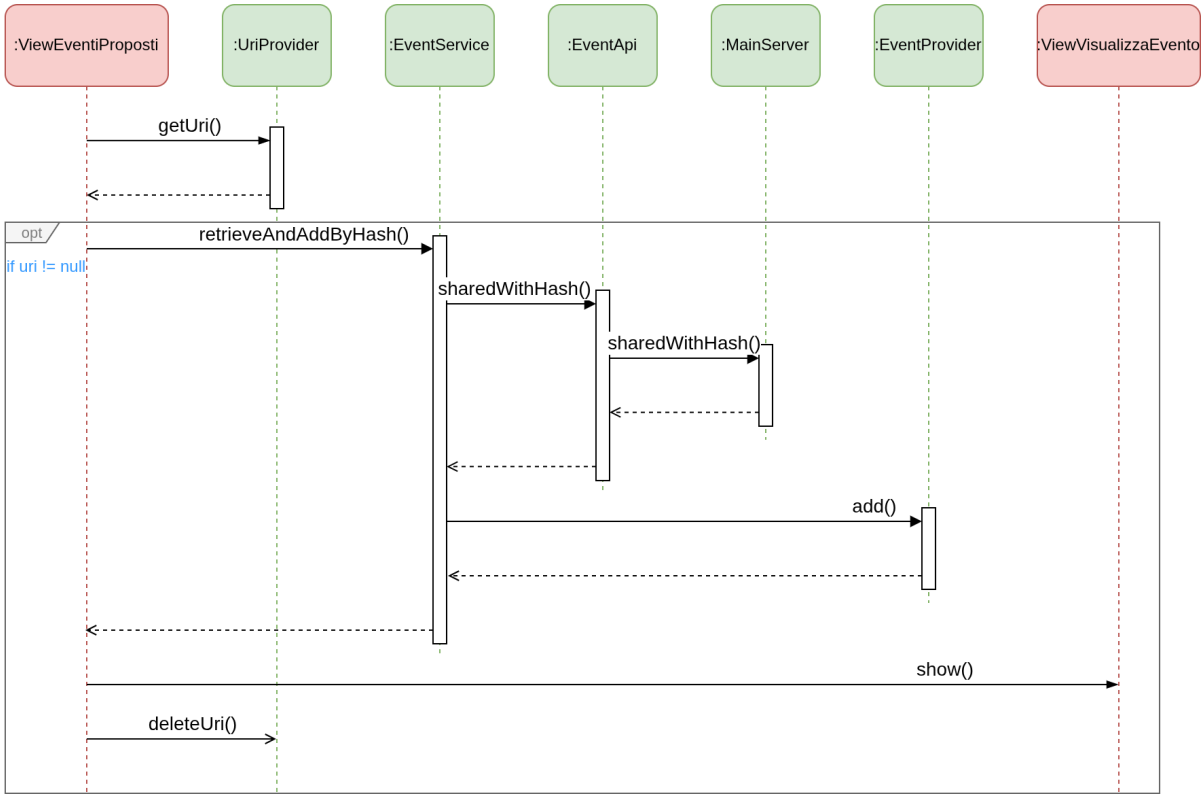


Figura 1.10: Diagramma di sequenza della visualizzazione degli eventi proposti

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.

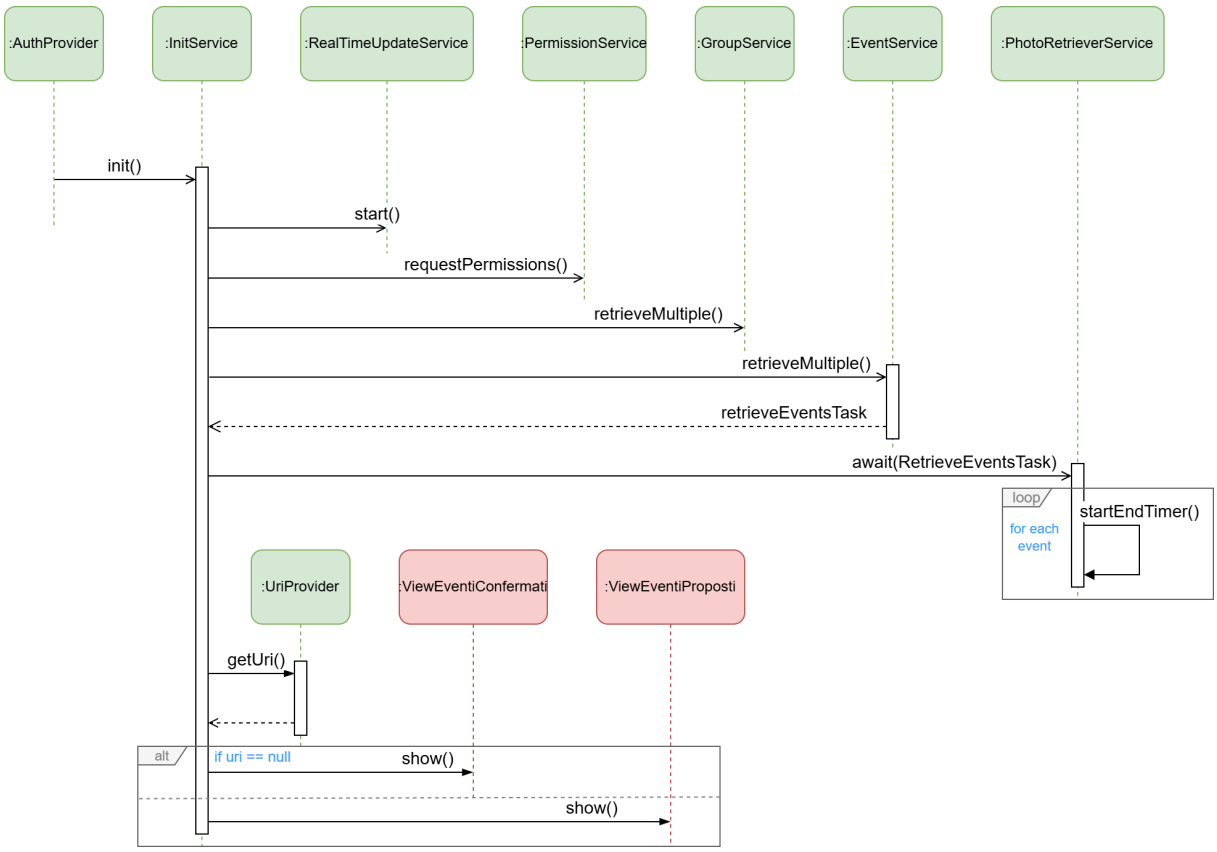


Figura 1.11: Diagramma di sequenza della fase di inizializzazione

1.1.4 La distribuzione del codice verso i dispositivi utente

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. Per questi motivi, nella 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, con l'obiettivo di estendere il servizio alle altre tecnologie in un secondo momento.

La grafica è stata sviluppata in maniera statica, ovvero non dipende direttamente da nessuna informazione specifica dell'utente. L'interfaccia sviluppata infatti non prevede la creazione dinamica di contenuti: gli elementi visuali che vengono restituiti rimangono invariati indipendentemente dall'utente che ne effettua la richiesta. I dati visualizzati che interessano l'utente corrente (eventi confermati o proposti, gruppi, profili e via dicendo) sono recuperati dalla memoria locale del dispositivo o tramite un server terzo. Questa scelta consente di rendere l'interfaccia grafica completamente indipendente dall'identità o dal ruolo dell'utente, consentendo una separazione netta delle responsabilità dei componenti.

Da un punto di vista della distribuzione, le due tecnologie per cui l'interfaccia è stata realizzata funzionano in modalità completamente diversa. L'interfaccia web prevede l'utilizzo di un browser utente che si connette a un server apposito per ottenere la grafica da visualizzare. L'interfaccia Android (come qualunque interfaccia realizzata per essere eseguita direttamente su un dispositivo) richiede invece la creazione di un applicativo apposito, da scaricare e installare sul dispositivo.

Per la fruizione del codice web si necessita di un server relativamente semplice, che, grazie alla staticità del sito, restituisca solo gli elementi necessari per la visualizzazione e l'elaborazione locale, senza bisogno di modificarli. La creazione di questo tipo di risorsa usando tecnologie in cloud può avvenire in vari modi, dall'utilizzo di una macchina vir-

tuale apposita a soluzioni che utilizzano container. Tuttavia, Azure mette a disposizione un servizio apposito per queste precise esigenze, Azure Static Web App (ASWA).

Questa risorsa, oltre a integrarsi direttamente con il resto dell'ambiente Azure, permettendo quindi (ad esempio) il collegamento con servizi di monitoraggio delle performance, garantisce disponibilità e scalabilità in ogni momento, facendosi carico di gestire la capacità computazionale richiesta. Il consumo delle risorse è infatti calcolato in base alla bandwidth, ovvero la quantità di dati che viene fornita dal server. Il piano gratuito prevede i primi cento giga byte di bandwidth inclusi, che rende il servizio vantaggioso, oltre che pratico.

La distribuzione dell'applicativo Android necessita di un modo per scaricare il file di installazione del programma. In attesa della pubblicazione dell'applicativo sull'App Store di Android, che fornirebbe agli utenti la possibilità di trovarlo e installarlo direttamente, ma che richiede ulteriori passaggi di certificazione e autenticazione, è stato necessario trovare un'altra soluzione. Anche in questo caso, le opzioni per fornire un file tramite cloud ce ne sono tante, ma la più semplice ed efficace è attraverso l'utilizzo di Azure Blob Storage.

Questo servizio consente l'archiviazione e la distribuzione di file di varie tipologie, fornendo un link diretto per il loro recupero. Ha un costo che varia in base all'utilizzo, aggirandosi attorno ai due centesimi per giga byte. Bilanciando la temporaneità del servizio, la ridotta quantità di download inizialmente previsti e la complessità che verrebbe introdotta in caso di utilizzo di un altro servizio, non è stato ritenuto necessario approfondire ulteriormente la ricerca.

Nonostante l'esecuzione del codice presenti alcune dipendenze in base al dispositivo, la distribuzione non ha introdotto nessuna necessità. Nessuna modifica del codice è stata quindi dovuta alla scelta della tecnologia usata per la loro distribuzione.

Per garantire un processo di aggiornamento efficiente e automatizzato, sia il codice di-

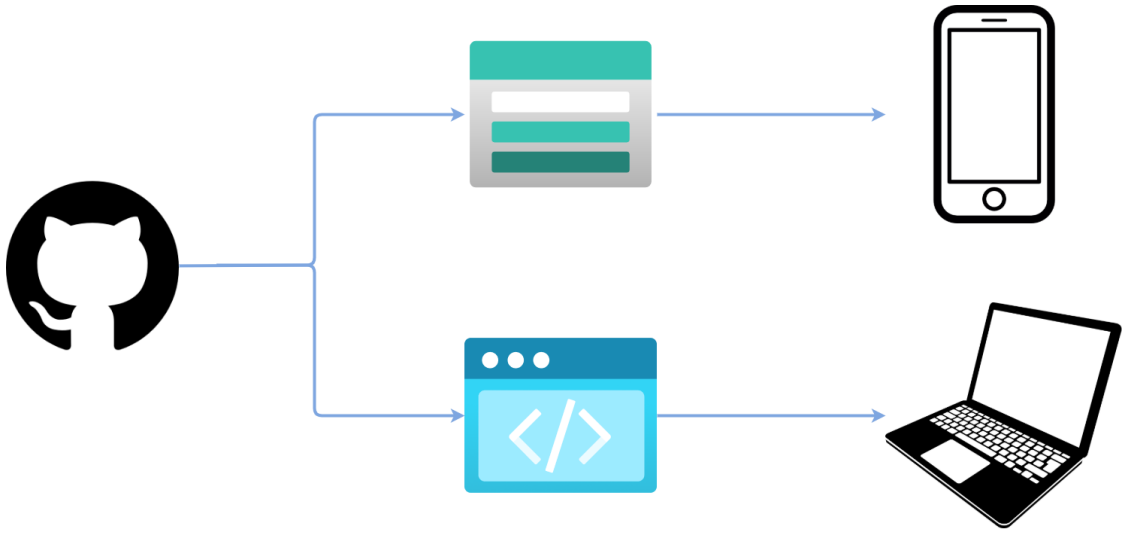


Figura 1.12: Diagramma di aggiornamento e distribuzione del client

stribuito sulla web app che l'applicativo ospitato nel container vengono gestiti tramite GitHub Actions. Le Github Actions sono funzionalità offerte da Github, che è il sito dove viene salvato il codice. In particolare, a ogni nuova versione del codice sia la Static Web App che il Blob Storage vengono notificati, modificando il loro contenuto. Questo consente di offrire un servizio aggiornato riducendo al minimo i tempi richiesti per applicare le modifiche. ASWA gestisce in autonomia il collegamento con la repository Github, mentre per aggiornare l'applicazione su Android è stato necessario crearne una appositamente.

In caso di aggiornamento, la fruizione dell'interfaccia tramite browser non necessita di nessuna manutenzione in quanto si aggiorna automaticamente a ogni accesso. Per l'applicativo su dispositivo mobile è invece necessaria una nuova installazione manuale. Per questa ragione, gli utenti dell'applicazione verranno notificati tempestivamente ogni volta che sarà disponibile una nuova versione.

1.2 La creazione del server principale

Il server principale ha il compito di ricevere tutte le richieste dai vari client, per poi elaborarle e restituire le informazioni volute, eventualmente a seguito di un'interrogazione verso la persistenza centrale. Per poter essere in grado di rispondere a un numero sempre maggiore di utenti, idealmente senza che questo impatti sul tempo di risposta o sulle performance in generale, deve essere implementato in maniera tale da poter permettere un'esecuzione distribuita, riducendo al minimo le dipendenze che possano minarne la duplicazione.

Per soddisfare questo tipo di esigenza esistono servizi definiti come Function as a Service (FaaS). I FaaS sono servizi serverless, ovvero la loro gestione hardware è completamente delegata al gestore del Cloud; il cui scopo è quello di duplicare ed eseguire il progetto, suddiviso e implementato attraverso molteplici Function. Ogni Function consiste in un'unità indipendente di codice, e in quanto tale crea la possibilità di essere eseguita in un ambiente di esecuzione unico per ogni richiesta. Questa caratteristica, combinata con la virtualizzazione dell'ambiente di esecuzione, consente una scalabilità potenzialmente illimitata.

L'indipendenza della Function dipende dalla relazione con le altre parti del progetto e dalla sua natura stateless. L'esecuzione di una Function infatti, oltre a dover essere limitata rispetto a qualunque dipendenza logica che possa dover essere condivisa, deve essere svincolata dalle informazioni sullo stato o sulla sessione. Queste caratteristiche, per loro natura, non possono essere garantite dal servizio, e sono quindi una responsabilità di chi deve svilupparle.

Per assicurarne l'autonomia, le Function dovranno essere implementate seguendo il principio di singola responsabilità. Ogni Function adempirà un solo compito specifico, creando una restrizione delle dipendenze e garantendo il minimo utilizzo di risorse necessarie per rispondere alla richiesta. Inoltre, si semplifica così la creazione e la manutenzione del codice, riducendo il controllo dell'esecuzione all'esito del tentativo di risoluzione del singolo problema.

1.2.1 La scelta del servizio adatto

I gestori in cloud che offrono servizi FaaS sono limitati. Escludendo i fornitori improntati allo sviluppo di applicazioni principalmente front-end, i fornitori sul mercato con servizi testati e maturi sono Amazon Web Services con AWS Lambda, Azure con Azure Functions e Google Cloud Provider con Google Cloud Functions. Tuttavia, queste tre soluzioni si assomigliano particolarmente.

Nonostante siano state tutte implementate usando una tecnologia proprietaria, le proprietà computazionali, di supporto ai linguaggi e di costo risultano molto simili.
Presentano tutti una granularità relativa alla scalabilità delle richieste a livello di funzione, ovvero permettono di duplicare il solo codice necessario per l'esecuzione della funzione richiesta. Il supporto ai linguaggi è molto esteso, ma offrono tutti comunque la possibilità di implementare un runtime personalizzato, di fatto supportando tutte le tecnologie.

	AWS Lambda	**Azure Functions**	**Google Cloud Functions**
Linguaggi supportati	Node.js, Python, Java, C#, Go, PowerShell, Ruby, PHP	Node.js, Python, Java, C#, PowerShell, F#, PHP	Node.js, Python, Java, C#, Go, F#, Visual Basic
Runtime Personalizzato	Sì, tramite i custom deplyment packages o AWS Lambda Layers	Sì, grazie ai custom handlers	Sì, tramite immagini Docker personalizzate
Massimo tempo di esecuzione	15 minuti	Dai 10 ai 60 minuti, in base al piano di pagamento	60 minuti per le richieste HTTP, 9 minuti per le richieste a eventi
Massima memoria dedicata a funzione	10 GB	Da 1.5 GB a 14 GB in base al piano di pagamento	4 GB

	AWS Lambda	**Azure Functions**	**Google Cloud Functions**
Tempo medio di attesa prima di essere spento	Dai 5 ai 7 minuti	Tra i 20 e i 30 minuti	15 minuti
Tempo medio di cold-startup	Generalmente sotto il secondo	Non oltre i 5 secondi	Da mezzo secondo a 2 secondi
Orchestrazione	Sì, tramite AWS Step Functions	Sì, tramite Durable Azure Functions	Sì, tramite GCP workflow
Costi			
Richieste gratuite mensili	400.000 GB-seconds	400.000 GB-seconds	400.000 GB-seconds
Tempo di esecuzione gratuita al mese	1 milione	1 milione	2 milioni
Costo della richiesta al consumo	$0.20 per milione	$0.20 per milione	$0.40 per milione
Costo di esecuzione al consumo	$0.000016 per GB-seconds	$0.000016 per GB-seconds	$0.0000125 per GB-seconds
Arrotondamento della durata	1 millisecondo	1 millisecondo	100 millisecondi

Tabella 1.1: Caratteristiche delle principali FaaS

La maggiore differenza tra queste soluzioni risulta nel tempo necessario in caso di start up, ma va notato che, oltre a essere un caso particolare del funzionamento, dipende molto dal linguaggio di programmazione, dalle dipendenze e dalla dimensione del progetto. Viene inoltre mitigato dal tempo di attesa prima di spegnere le istanze e dalla quantità degli utenti, che contribuiscono a diminuire la frequenza dello spegnimento.

Essendo le differenze tra un servizio e l'altro minime, sia a livello di costi che di pre-

stazioni, ed essendo il progetto improntato su Azure, la scelta della tecnologia su cui implementare il server principale è ricaduta sulle Azure Functions.

1.2.2 Le scelte progettuali derivate dall'utilizzo delle Azure Functions

L'utilizzo di un servizio FaaS comporta un approccio particolare per la scrittura del codice. A differenza di un programma normale, dove l'esecuzione del programma è indipendente dalla ricezione di un evento, nelle FaaS l'invocazione di un processo avviene esplicitamente a partire da un fattore esterno (che sia una richiesta HTTP o il messaggio in una coda). Non bisogna più preoccuparsi di come far interagire le parti dell'applicazione, ma piuttosto di come rispondere nella maniera più efficiente possibile a tanti particolari problemi. Ogni funzione dovrà essere implementata come entità autonoma rispetto al resto del sistema, con l'unica responsabilità di rispondere a un solo incarico specifico.

La difficoltà principale del procedimento sussiste nell'individuare i singoli compiti in cui suddividere l'applicazione. Raramente una richiesta può essere soddisfatta in un unico passaggio, e non è quindi automatico che a una Function corrisponda una sola parte di codice, soprattutto in quanto alcune richieste potrebbero avere alcune parti in comune (ad esempio, l'autenticazione è necessaria alla maggior parte delle richieste). Nel caso in cui una richiesta si componga di più passaggi logici, per poterli racchiudere in un'unica Function bisogna analizzarne la relazione. Innanzi tutto ogni passaggio deve essere implementato sempre in maniera indipendente e stateless come richiesto alle Function. A questo punto è necessario però che tutti i passaggi siano in diretta successione, e che il fallimento di uno solo di questi comporti il fallimento di tutta la funzione. In caso contrario, è sconsigliato raggrupparle in un'unica Function, quanto piuttosto suddividerli in altre piccole Function (con le stesse proprietà di cui sopra), per poi coordinarle tramite l'utilizzo di un orchestratore o di code di eventi.

L'orchestratore è una Function che ha la caratteristica di poter invocare e controllare altre Function. Consente di gestire efficacemente scenari in cui è richiesta un'esecuzione particolare delle operazioni, sia essa sequenziale o parallela, 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 orchestratore e le Function in esecuzione causano un tempo di risposta delle richieste più elevato.

Integrato all'interno delle Azure Functions, Azure Durable Function consente la creazione di un orchestratore, incaricato di gestire l'ordine, lo stato, il ciclo di vita e le risposte delle varie Function coinvolte nell'elaborazione della richiesta, mantenendo un'architettura indipendente e scalabile.

Il coordinamento tramite code di eventi invece prevede, durante l'esecuzione di una prima Function, l'invio di un evento al sistema. L'evento verrà aggiunto in coda, per poi invocare una seconda Function nel momento in cui sarà preso in carico. Questo consente di separare logicamente le Function tra loro senza introdurre ulteriori logiche e garantendo un tempo di risposta alla prima Function minimo, ma introduce alcune problematiche. Disaccoppiando le due Function la prima non ha conoscenza sull'esito della seconda, ed è quindi necessario che la Function invocata dalla coda non svolga un compito essenziale o che siano previste logiche di rilancio, controllo o segnalazione dell'esito. Inoltre l'invocazione verrà così considerata come doppia, andando a influire sui costi totali.

Come linguaggio di programmazione per lo sviluppo delle Function 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.

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 creando 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 nel tempo, grazie al più ampio numero di versioni del framework .Net a disposizione. Il modello in-process, differentemente, è 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.

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 e intuitivo del codice, gestito interamente dal programma.

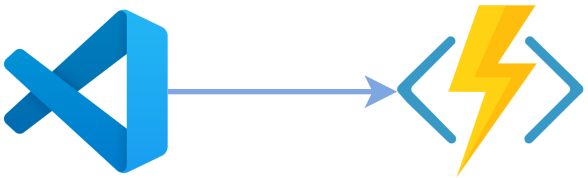


Figura 1.13: Diagramma di aggiornamento e distribuzione del server

1.2.3 L'implementazione della logica applicativa

Per quanto ogni Function ricopra un unico compito, alcune parti della sua risoluzione possono essere condivise con altre. Per questo motivo, la logica applicativa è stata suddivisa in metodi che risolvono una specifica esigenza logica, che saranno poi inclusi nelle Function quando necessario. 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.

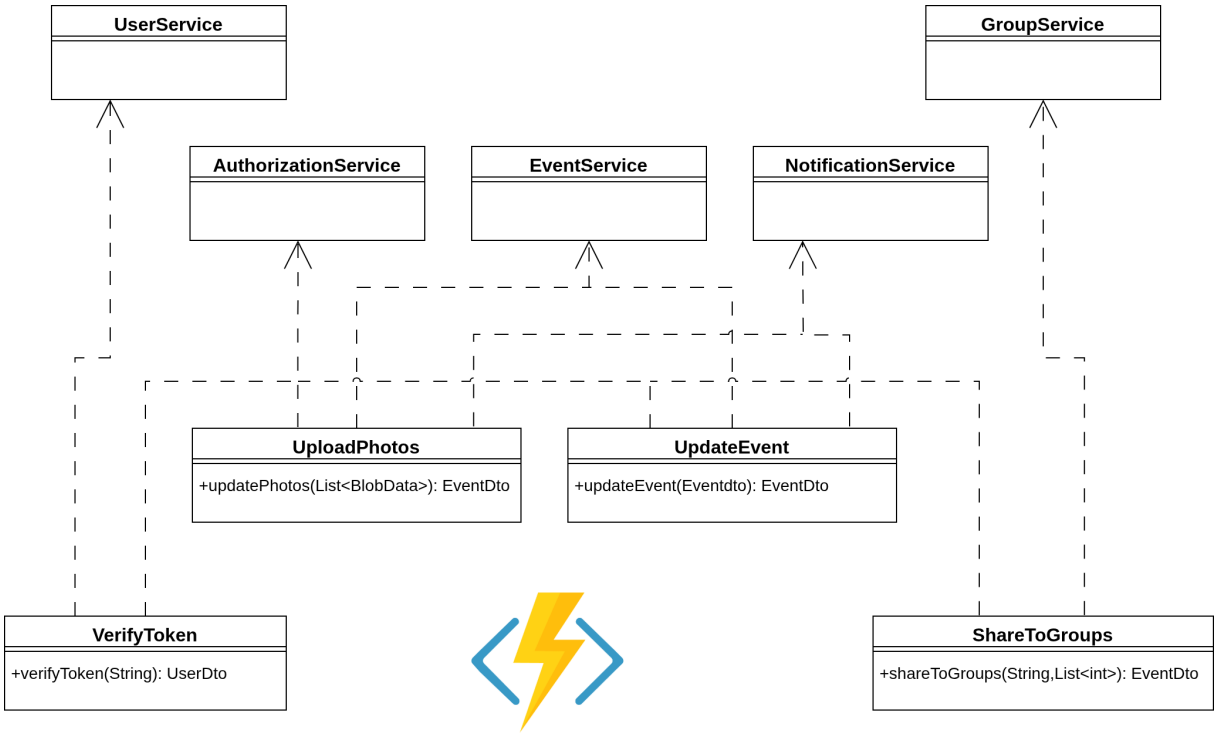


Figura 1.14: Modello delle relazioni tra Functions e i servizi usati

Con la 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 WydDbService 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. L'implementazione 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.

La maggior parte delle Function hanno il compito di rispondere a una richiesta REST, in cui l'utente cerca di recuperare dei dati o di modificarli. Ogni Function di questo tipo seguirà in generale lo stesso procedimento. Il primo passo consiste nell'autenticare l'utente che fa la richiesta, analizzando il relativo token. Si controlla poi che l'utente abbia i permessi necessari per eseguire l'operazione desiderata. Se è tutto in regola, si procede a elaborare i dati di ingresso, se necessario, tramutandoli in oggetti logici. Si esegue quindi l'azione voluta, la vera responsabilità della Function. In base alla natura dell'operazione, potrebbe essere opportuno inviare le notifiche degli aggiornamenti avvenuti. Infine, si invia la risposta dell'operazione avvenuta a buon fine, con in allegato i dati eventualmente necessari. Tutto il processo viene inserito in un blocco che permette l'identificazione di errori e la relativa risposta.

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.

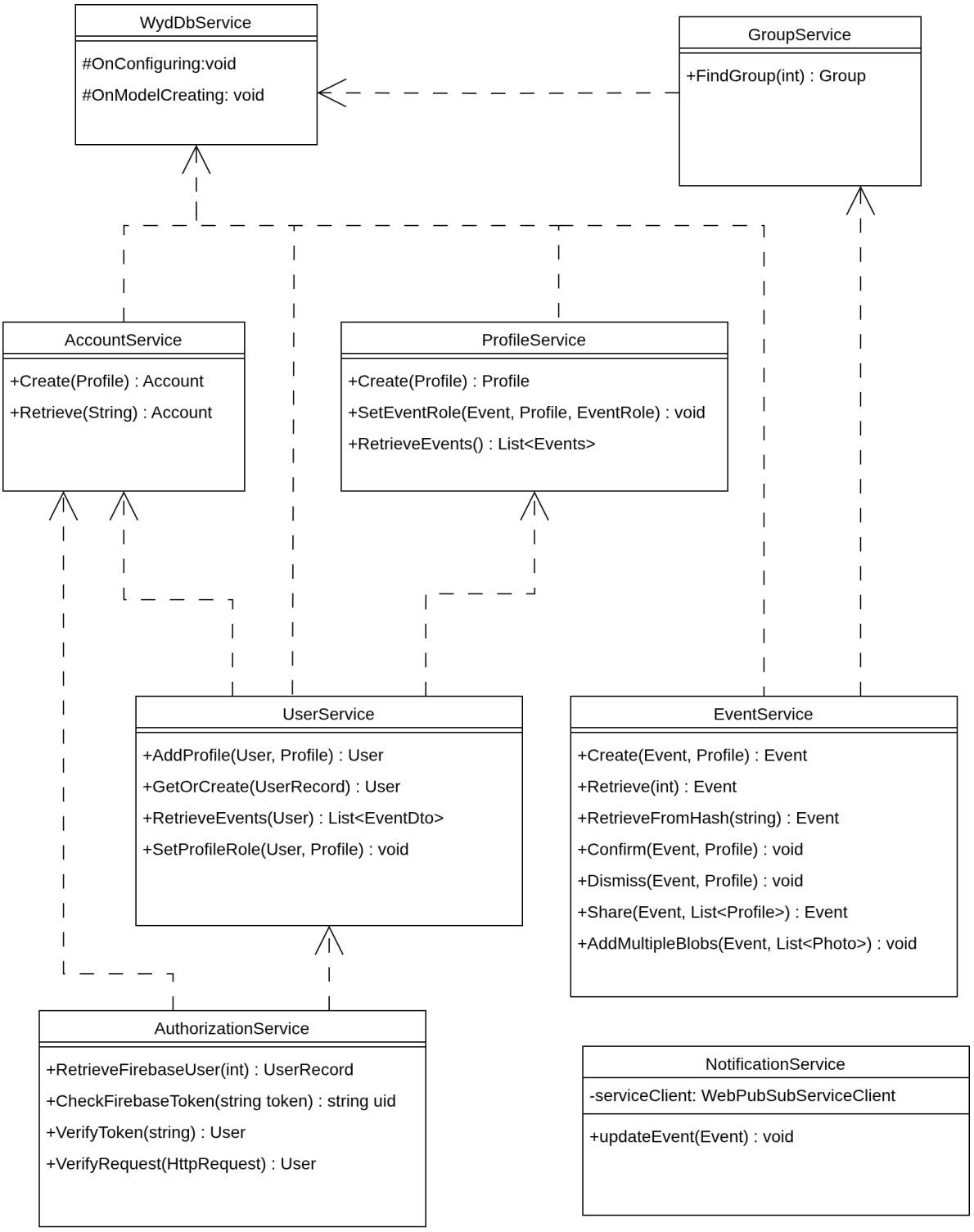


Figura 1.15: Modello delle classi del server

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

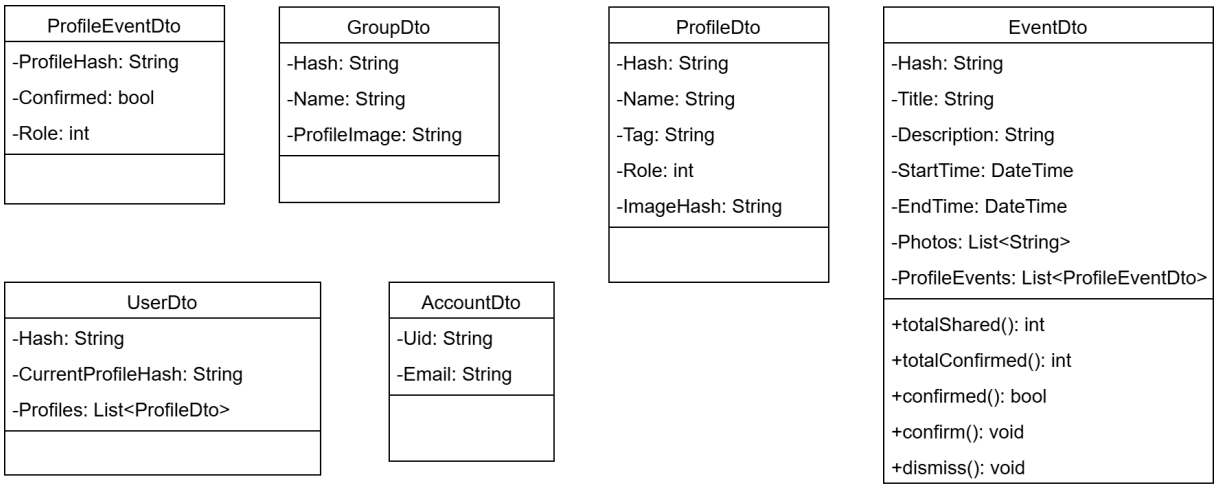


Figura 1.16: Modello delle classi dei data tranfer object

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.

```
[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 1.17: Test di creazione di un account

1.3 Autenticare le richieste: la scelta del servizio e la sua integrazione

Poiché la modalità di autenticazione rappresenta un elemento importante per l'esperienza utente, in quanto deve assicurare un accesso sicuro all'applicazione mantenendone la semplicità, la facilità del processo di autenticazione deve essere garantita. L'applicazione deve consentire la possibilità di registrarsi creando un nuovo account dedicato, ma è altrettanto essenziale che permetta agli utenti di farlo anche tramite il proprio servizio di autenticazione preferito, migliorando sicuramente l'usabilità e l'apprezzamento. 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 quindi ricaduta su Firebase Authentication, il servizio di autenticazione di Google Cloud Provider, che garantisce sia la possibilità di creare account dedicati che di collegarsi attraverso altri servizi di autenticazione. Presenta librerie di integrazione sia tramite Flutter che tramite C# che risultano facili da utilizzare, oltre a fornire una piattaforma di gestione con un'interfaccia chiara e intuitiva. Dal punto di vista economico, il servizio risulta vantaggioso, essendo gratuito fino ai cinquantamila utenti mensili attivi.

Firebase si integra facilmente con Flutter, fornendo una libreria che gestisce completamente l'ottenimento e il mantenimento dei token di autenticazione, a partire dalle credenziali o dalle verifiche precedenti. Per ogni richiesta che richiede identificazione un servizio apposito intercetta il messaggio, recuperando il token e allegandoglielo. Alla ricezione del messaggio, il server estrae il token dalla richiesta, per poi contattare Firebase grazie l'astrazione fornita dalla libreria. Firebase controlla il token e, se corretto, ne restituisce

i dati dell'account relativo.

Uno dei requisiti del progetto prevede che ogni account sia associato in modo univoco a un singolo utente. Durante la fase di registrazione, 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.

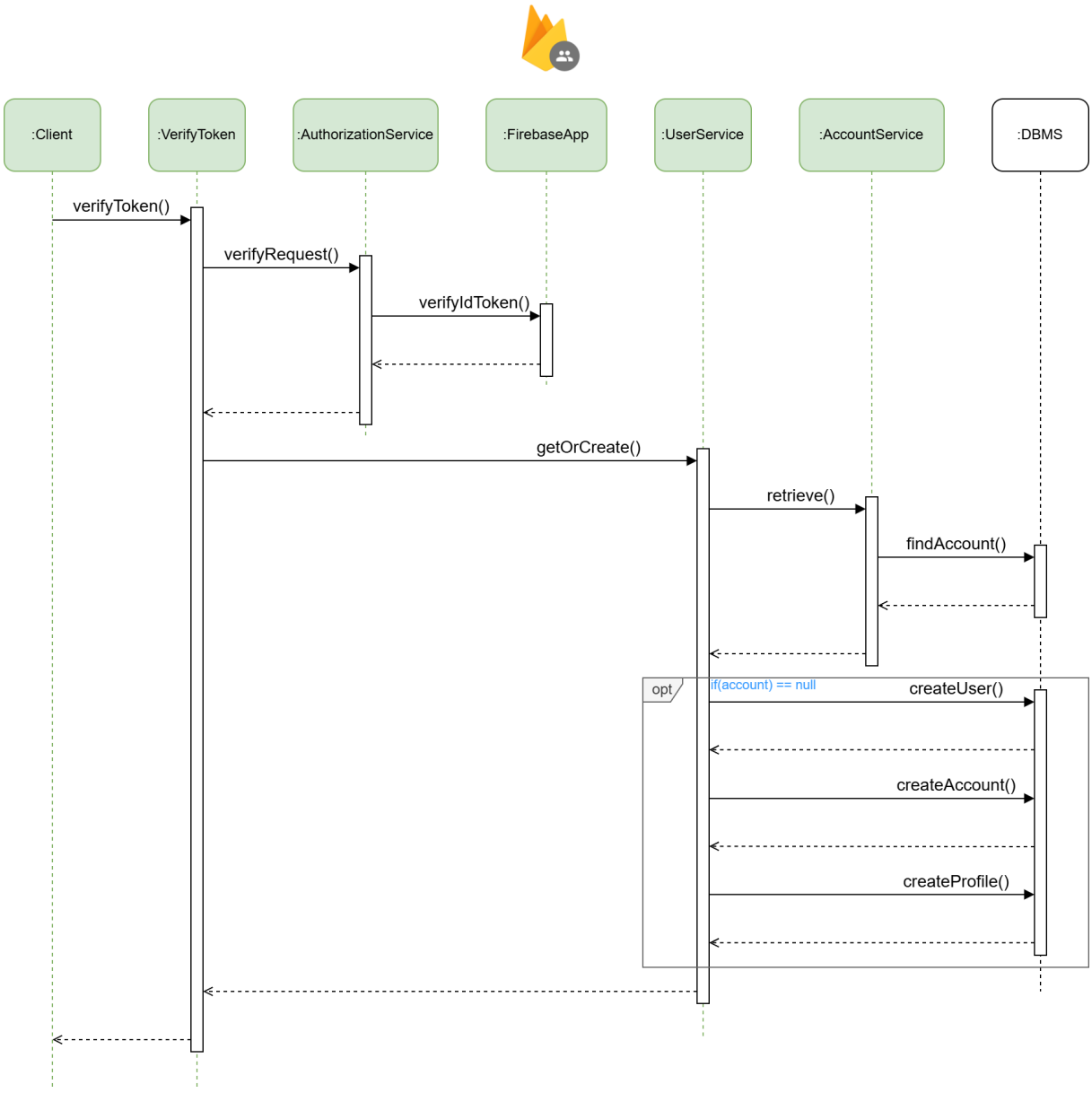


Figura 1.18: Diagramma di sequenza per la creazione di un account

1.4 Uno sguardo sulla sicurezza: segreti e protocolli

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, separando di fatto la logica implementativa dai segreti necessari per la sua esecuzione, riducendo così il rischio di una perdita delle chiavi derivata da un errore dello 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 a 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 recupero delle immagini (disponibili in teoria pubblicamente), avviene grazie a un link univoco dato dalla combinazione degli identificativi dell'evento e dell'immagine. Utilizzando i codici di hash diventa molto complicato ritrovare le immagini senza essere a conoscenza dei codici, che non avendo natura incrementale ma distribuita rende indovinare l'unica strategia per trovare un link valido.

1.5 Il monitoraggio dei servizi

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

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

Per monitorare le Azure Functions sarà invece necessario affiancargli un'istanza di Azure Application Insights, servizio nato appositamente per controllare il funzionamento e la risposta dei servizi Azure. Una volta collegato il servizio, infatti, 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. La funzione non dovrà fare altro che chiamare il metodo apposito per generare e salvare un log. Tali log saranno poi consultabili e analizzabili tramite l'interfaccia fornita da Azure Application Insight.