



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 per l'organizzazione di eventi condivisi

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 e 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 di Wyd, evidenziando l’impatto dell’integrazione delle risorse cloud sul risultato finale.

Indice

Introduzione	1
Organizzazione dei capitoli	3
Capitolo 1	4
Individuazione dei requisiti e dei casi d'uso	5
I requisiti e il vocabolario	5
Casi d'uso	8
Requisiti di sicurezza	14
Analisi del problema	20
Analisi delle funzionalità	20
Ideazione dell'architettura logica	24
Capitolo 2	29
Sviluppo del client utente	31
Scelta del framework di sviluppo	32
Realizzazione delle interfacce grafiche	33
Implementazione della logica applicativa	38
Distribuzione del codice verso i dispositivi utente	44
Creazione del server principale	47
Individuazione del servizio adatto	48
Scelte progettuali derivate dall'utilizzo delle Azure Functions	50
Implementazione della logica applicativa	53
Autenticare le richieste: la scelta del servizio e la sua integrazione	57
Uno sguardo sulla sicurezza: segreti e protocolli	60
Monitoraggio dei servizi	61
Capitolo 3	63

Analisi per l'identificazione del database	65
Proprietà dei database relazionali	66
Proprietà dei database non relazionali	67
Impatto delle relazioni delle entità sulle prestazioni	68
Analisi del dominio	70
Implementazione del database	73
Scelta del database	74
Configurazione di Cosmos DB	79
Definizione delle collezioni	81
Integrazione con le Azure Functions nel framework .Net	84
Garantire la consistenza eventuale dei dati	86
Implementazione della memoria locale	93
La realizzazione della cache	94
Scelta della tecnologia per la comunicazione in tempo reale	96
Invio delle notifiche	98
Garantire l'allineamento della cache	102
Capitolo 5	103
Modalità di recupero delle immagini	104
Integrazione con il sistema	107
Scelta del servizio di persistenza	107
Procedura di salvataggio	109
Capitolo 5	113
Impostazione dei test	114
Velocità della procedura di creazione	115
Velocità in lettura	118
Velocità degli aggiornamenti	120
Garanzia di propagazione delle informazioni	122
Velocità di salvataggio delle immagini	125
Costo previsto del sistema	128
Conclusione	134
Sviluppi futuri	136

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 a 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 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 e il salvataggio delle foto aggiungono la necessità di gestire richieste di archiviazione di dimensioni significative.

Descrizione dei capitoli

L'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 a 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 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 a 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 lo sviluppo del programma vero e proprio.

L'abstract di Wyd è il seguente:

Wyd è un'applicazione che permette ai clienti di organizzare i propri impegni, siano essi confermati oppure proposti. Mette a disposizione due calendari, il primo con gli eventi in cui l'utente è convinto di partecipare, il secondo in cui vengono riuniti gli eventi a cui l'utente è stato invitato ma senza aver ancora dato conferma. L'utente ha la possibilità di creare, modificare, confermare o disdire un evento, ma anche condividerlo con altri o allegarci foto.

La condivisione di un evento può avvenire con applicazioni esterne tramite la generazione di un link o grazie all'ausilio di gruppi di profili. Inoltre, al termine di un evento, l'applicazione carica automaticamente le foto scattate durante l'evento, per allegarle a seguito della conferma dell'utente. L'utente può infatti cercare altri profili e creare gruppi con i profili trovati. Tutta l'interazione avviene tramite l'utilizzo di profili, che permettono di suddividere semanticamente gli eventi e le relazioni.

1.1 Individuazione dei requisiti e dei 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 e 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 a 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.

Da una prima analisi dell’abstract si evincono immediatamente i principali requisiti funzionali, che riguardano in generale l’esperienza utente nelle sue parti principali, dalla visualizzazione nelle schermate all’inserimento di dati e foto. Si aggiungono quindi le funzionalità dettate da necessità derivate, quali l’esigenza di autenticare l’utente o il bisogno di gestire i profili e i gruppi. Infine si analizzano le specifiche non funzionali, che sono raramente incluse nel testo ma che descrivono le caratteristiche performative e di sicurezza ritenute essenziali per il successo desiderato.

Vengono quindi introdotti i desiderata relativi all’esperienza utente, evidenziando l’intuitività e la reattività dell’applicazione, che richiedono di conseguenza velocità nel recuperare i dati. Inoltre, essendo Wyd pensata per interagire con migliaia di utenti, in simultanea e interconnessi tra loro, le caratteristiche di scalabilità vengono individuate e introdotte fin da subito. La sicurezza del sistema, escludendo l’autenticazione che impatta direttamente sull’utente, necessitando di un’analisi approfondita apposita, viene trattata in seguito.

ID	Requisiti	Tipo
R1F	Registrazione di un account tramite l'interfaccia web	Funzionale
R2F	Identificazione attraverso mail univoca e password di almeno sei caratteri	Funzionale
R3F	Visualizzazione degli eventi confermati	Funzionale
R4F	Visualizzazione degli eventi proposti	Funzionale
R5F	Creazione di un evento impostando almeno la data d'inizio e quella di fine	Funzionale
R6F	La data di fine deve essere successiva alla data d'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 1.1: 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. Questo consentirà, quando in seguito verranno citati, di evitare possibili ambiguità derivate dall'uso comune dei termini. Si specifica cosa si intende quindi per utente, profilo e gruppo, ma anche, analizzando i requisiti circoscritti in precedenza, evento confermato e proposto, o ancora email e password.

Voce	Definizione	Sinonimi
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 otto caratteri	
Credenziali	Insieme composto da email e password necessari per accedere al sistema	

Tabella 1.2: Vocabolario di Wyd

1.1.2 Casi d’uso

I casi d’uso descrivono le interazioni tra gli attori e 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, e ogni caso d’uso può essere relazionato con altri, definendo la loro relazione.

I casi d’uso possono essere collegati tra loro tramite rapporti d’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.

Per ogni azione descritta nei requisiti, intuibile dal contesto o necessaria per il soddisfacimento dei requisiti, viene introdotto un nuovo caso d’uso. Ad esempio, la creazione o la condivisione di un evento vengono estratti direttamente dalla descrizione del progetto, così come la ricerca automatica delle immagini. L’eliminazione di un evento, la registrazione di un utente o l’aggiunta di un profilo a un gruppo, per quanto non espressamente elencati tra i requisiti, sono dedotti dal contesto. L’aggiornamento da un server esterno viene introdotto per realizzare la modifica in tempo reale degli eventi.

I casi d’uso così individuati si raggruppano attorno a tre principali. VisualizzaEvento permette di visualizzare i dettagli dell’evento, ma anche di modificarli e di eseguire tutte le azioni relative. GestioneGruppi racchiude tutte le esigenze che riguardano i contatti e i gruppi, come la visualizzazione dei gruppi, la ricerca dei profili e l’aggiunta di un profilo a un gruppo. Infine GestioneProfili permette il controllo dei profili collegati all’utente, visualizzandoli e dando la possibilità di cambiare il profilo corrente.

Si distinguono rispetto ai precedenti casi d’uso e alle loro estensioni altre funzionalità il cui scopo non riguarda nessuno dei tre argomenti, o il cui funzionamento sia scorrelato. Login e Registrazione, ad esempio, vedono il loro utilizzo in maniera trasversale rispetto al resto dell’applicazione, essendo necessari per il funzionamento di tutti ma distanti da un punto di vista logico. AggiornaEvento e RecuperaImmagini eseguono indipendentemente dall’interazione utente, apportando modifiche automaticamente in base ad attori esterni.

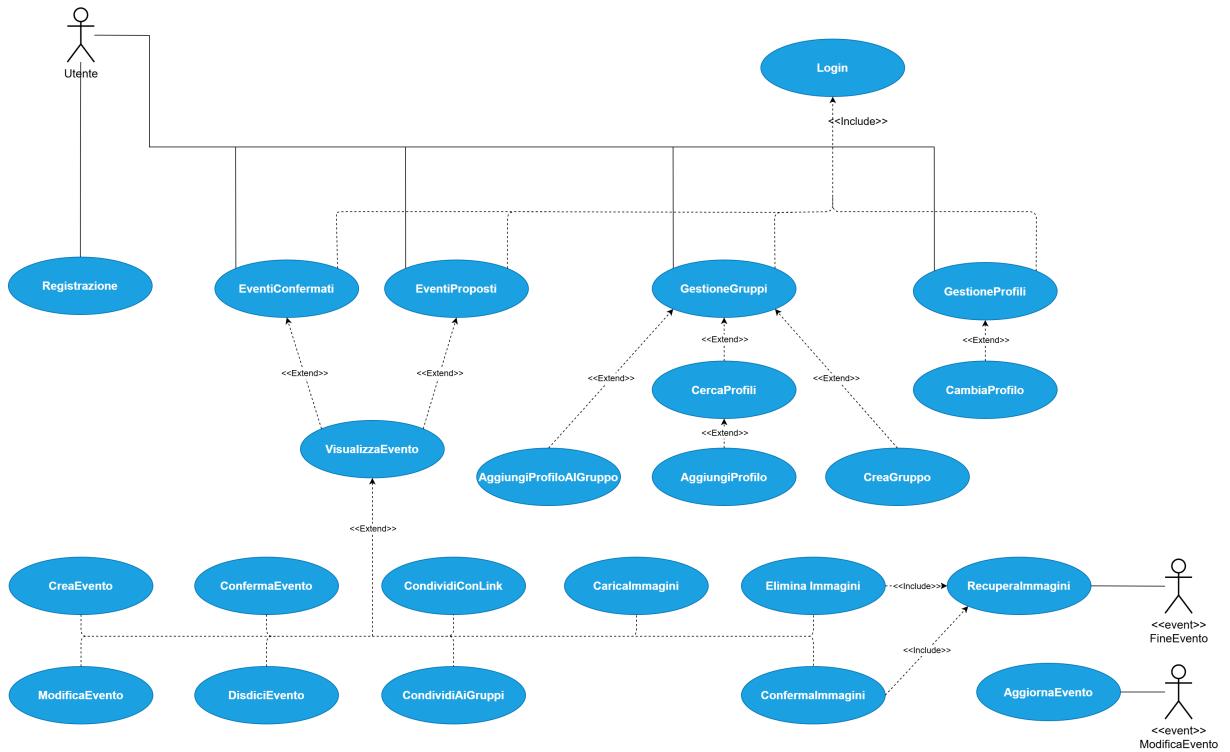


Figura 1.1: Diagramma dei casi d’uso

Per ogni caso d’uso viene poi identificato uno scenario di utilizzo, che chiarifica il contesto, il comportamento e i punti critici dell’utilizzo. Lo scenario ha il solo compito di mostrare il comportamento desiderato, senza scendere quindi in dettagli o complessità progettuali. Dallo scenario risulta quindi complesso dedurre la difficoltà implementativa del caso d’uso, ma è il tassello da cui si stabiliranno poi la coordinazione e l’interazione delle varie parti del programma.

Si riportano gli scenari di utilizzo per i principali casi d’uso di Wyd, ovvero quelli che più andranno a 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 a un unico utente, che può così in seguito creare o unirne di nuovi.

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

Tabella 1.3: 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 tutti i dispositivi di tutti gli utenti a cui l’evento è stato condiviso. Inoltre, sarà necessario inserire un controllo per evitare che due richieste simultanee causino conflitti.

Titolo	ModificaEvento
Descrizione	Salva le modifiche a un evento
Attori	Utente
Relazioni	VisualizzaEvento
Precondizioni	L’evento esiste e sono stati modificati dei dati

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

Tabella 1.4: 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 queste particolari richieste di caricamento, che generalmente necessitano di 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 sullo stesso evento risulta elevata, creando la necessità di una gestione parallela di modifiche concorrenti.

Titolo	CaricaImmagini
Descrizione	Permette all'utente di selezionare immagini da collegare all'evento, salvandole
Attori	Utente
Relazioni	VisualizzaEvento
Precondizioni	L'evento esiste

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

Tabella 1.5: Scenario del caricamento delle immagini

L’azione di recupero delle immagini facilita l’utilizzo dell’applicazione, automatizzando il procedimento di ricerca delle immagini, riducendo l’interazione utente alla sola conferma. Una 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 cernita di dati, con effetti sull’analisi tecnologica, sui processi in background e sulla gestione della memoria locale.

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

Tabella 1.6: Scenario di recupero delle immagini dal dispositivo dell’utente

1.1.3 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 rilevazione dei rischi e la successiva definizione dei requisiti necessari per evitare o minimizzare i danni è alla base della strategia di sicurezza.

La definizione dei requisiti di sicurezza deriva dall'analisi del rischio. L'analisi del rischio individua i possibili vettori di attacco e serve a orientare le risorse dove più necessario, tramite la valutazione dei beni, l'identificazione delle minacce e l'individuazione dei punti deboli delle tecnologie di cui si prevede l'utilizzo.

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. In particolare, Wyd non prevede altri sistemi diversi dai comuni sistemi informatici, ma i valori principali da proteggere risiedono nei dati degli utenti.

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 1.7: 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 e il costo di realizzazione delle contromisure necessarie. Fornisce quindi una prima analisi sulle necessità implementative.

Tutte le risorse di Wyd sono orientate agli utenti, in particolare al mantenimento e alla distribuzione dei loro dati. Per questo motivo le minacce sono relative alla confidenzialità dei dati o all'interruzione del servizio.

Minaccia	Probab.	Controllo	Fattibilità
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 d'implementazione

Tabella 1.8: 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.

Wyd prevede principalmente la comunicazione tra le applicazioni utenti e un server centrale, per cui la tecnologia da analizzare si concentra sull’architettura ma soprattutto sulle comunicazioni e sull’autenticazione.

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

Tabella 1.9: Analisi tecnologica della sicurezza

A questo punto si prevedono 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 sono molto simili alle minacce individuate in precedenza, ma vengono creati in base alla modalità di attacco, più che alla tipologia. A ogni caso d’uso malevolo ne viene corrisposto un altro che ne comporta la mitigazione. Si integrano quindi con i casi d’uso dell’applicazione, evidenziando i punti e la loro applicazione.

In Wyd sono stati individuati quattro casi d'uso malevoli, tre dei quali relativi all'integrità e alla confidenzialità dei dati, e uno relativo alla disponibilità del servizio.

Tramite la saturazione del database l'attaccante riesce a inserire quantità importanti di dati, che può comportare un rallentamento dell'applicazione temporaneo o permanente, in base alla configurazione dell'attacco. Questo è particolarmente efficace dal momento in cui si possono inserire delle foto. Per mitigare questo rischio si aggiunge un caso d'uso relativo al controllo delle dimensioni delle richieste.

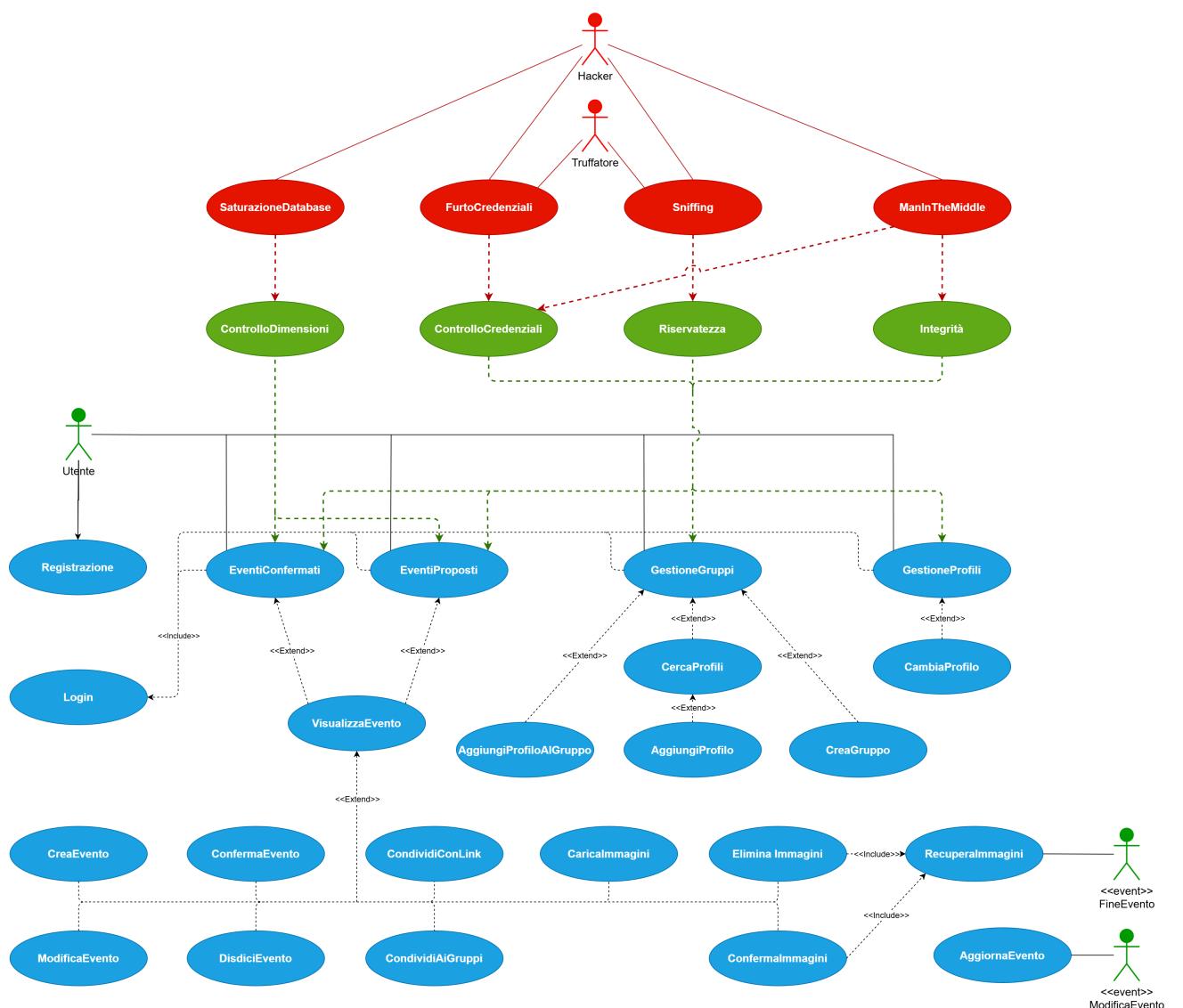


Figura 1.2: Casi d'uso relativi alla sicurezza

La confidenzialità e l'integrità dei dati sono minacciati dal furto di credenziali, che permetterebbe a un utente d'identificarsi come qualcun altro; lo sniffing mina la riservatezza delle comunicazioni, se vengono intercettate; infine tramite man in the middle un attore malevolo ha la possibilità di modificare le richieste ingannando entrambi i lati della conversazione. Si introducono i casi d'uso relativi al controllo delle credenziali, che aumenta la difficoltà di un possibile furto; la riservatezza, che permette di nascondere le comunicazioni alle parti non interessate, e l'integrità, che consente l'individuazione di eventuale manipolazione dei messaggi.

Visti i costi e appurate le risorse a disposizione sono stati quindi identificati 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 a 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

ID	Requisiti	Tipo
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 1.10: Requisiti di sicurezza

1.2 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.

In particolare, VisualizzaEvento permette di accedere alla maggior parte delle azioni che l’utente può attuare sugli eventi. Allo stesso modo GestioneGruppi e GestioneProfili permettono di eseguire le azioni correlate al loro contesto.

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 1.11: Scomposizione delle funzionalità

Di ogni funzionalità vengono evidenziati il grado di complessità, la tipologia di azione che svolgono e 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.

A parte il Login, la Registrazione e ScritturaLog, il cui servizio è diretto e uniforme in tutta l'applicazione, tutte le altre funzionalità prevedono una gestione e manipolazione di più dati, a volte in strutture complicate, a volte permettendo una modifica puntuale delle informazioni interessate.

Funzionalità	Tipo	Grado di complessità	Requisiti Collegati
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 1.12: Funzionalità

Si procede analizzando i dati che ogni funzionalità gestisce, indicandone la tipologia, la protezione richiesta e i vincoli correlati, per conoscere in maniera definitiva tutte le caratteristiche delle informazioni scambiate. L’analisi delle informazioni non viene riportata in quanto poco rilevante ai fini della tesi, ma eventuali dettagli saranno riportati quando necessario.

A seguito dell’analisi delle informazioni, si procede con l’analisi dei vincoli, in cui si chiarificano i requisiti non funzionali, evidenziandone le criticità e quali componenti ne vengono coinvolti.

Requisito	Categorie	Impatto	Funzionalità
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

Requisito	Categorie	Impatto	Funzionalità
Scalabilità delle richieste	Tempo di Risposta	Minor degradamento delle prestazioni	EventiConfermati, EventiProposti, AggiornaEvento, RecuperaImmagini

Tabella 1.13: Analisi dei vincoli

Infine si definiscono logicamente le maschere, ovvero i componenti visuali essenziali del programma. A 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.

Maschera	Informazioni	Funzionalità
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 1.14: Maschere

1.2.2 Ideazione dell’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.

Il dominio di Wyd si concentra attorno a due entità principali: Event e Profile.

Gli Event contengono tutti i dati generali degli eventi, quali l’ora d’inizio e l’ora di fine, il titolo e la descrizione. Strettamente correlate a loro ci sono le immagini, identificate con Photo. Ogni Event può avere più Photo, ma una Photo può essere relativa da un solo Event. I Profile racchiudono i dati dei profili, che devono essere identificati da un Tag unico in tutto il programma. Group rappresenta un gruppo. Più Profile possono fare parte di un Group, ma, allo stesso modo, un Profile può appartenere a più Group.

L’Account memorizza le informazioni attraverso cui l’utente può accedere e agire in quanto User, entità che descrive l’utente. L’utente ha diverse modalità di accesso, ed è per questo motivo che più Account possono essere relativi allo stesso User. Lo stesso utente può impersonare più profili, e un profilo può essere gestito da più utenti. User e Profile sono quindi connessi in una relazione molti a molti.

La maggior parte delle operazioni avrà a che fare con le due entità principali, da cui l’importanza dell’elemento che ne descrive la relazione, ovvero ProfileEvent.

ProfileEvent ha un ruolo centrale in quanto sarà l’unità interrogata sia quando si vorranno ottenere gli eventi di un determinato profilo, sia quando bisognerà recuperare i profili relativi a un evento. Contiene tutte le informazioni relative al particolare profilo sul determinato evento, rendendolo infatti l’entità più modificata di tutto il progetto.

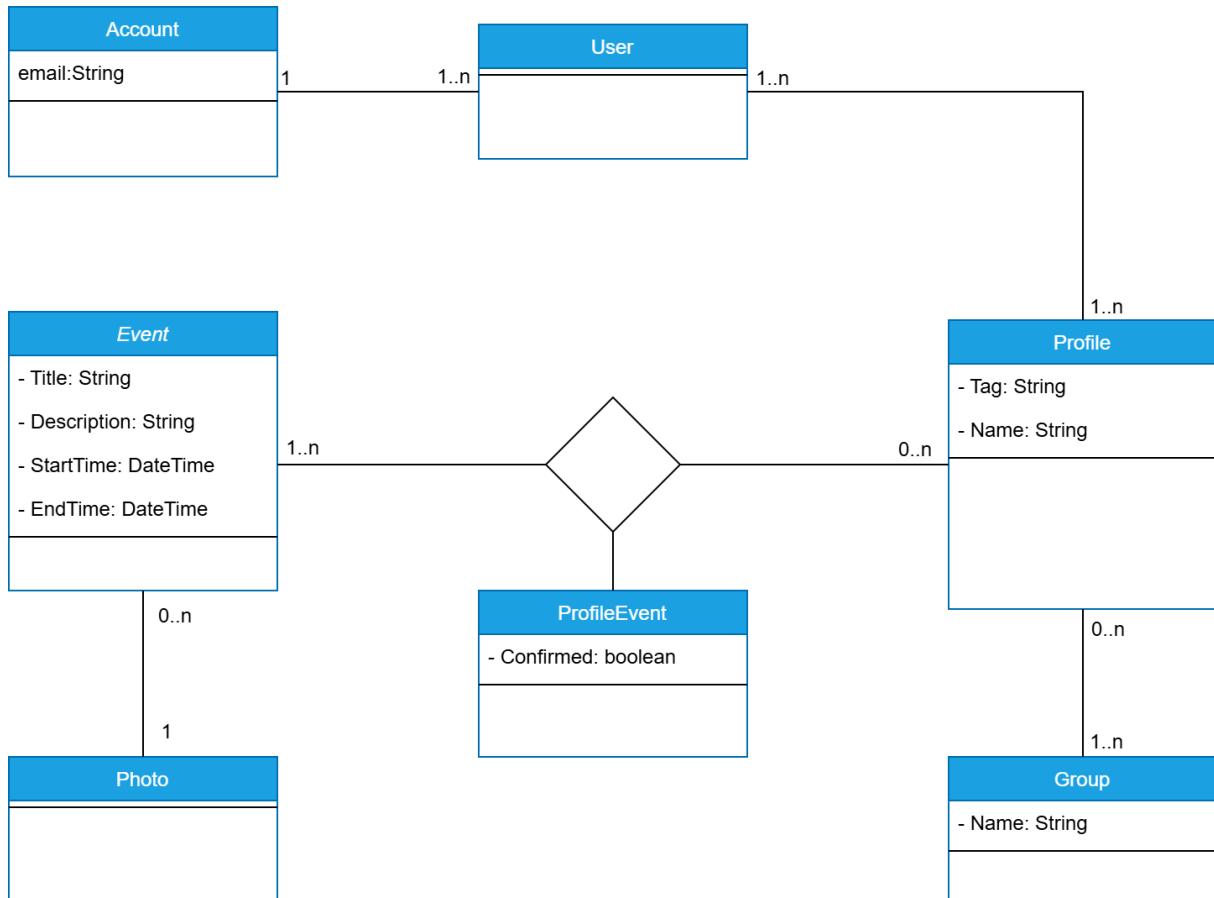


Figura 1.3: Modello del dominio

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 e alle dipendenze delle funzionalità che ricopre. Si possono così distinguere, ad esempio, package relativi a 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.

Distinguiamo i diversi package in base al loro scopo.

InterfacciaAccesso e InterfacciaUtente sono le parti che si occuperanno dell’interazione grafica con l’utente. InterfacciaAccesso deve presentare le schermate di login e di registrazione, e tutti i passaggi intermedi che saranno necessari. InterfacciaUtente si occuperà di mostrare le viste per interagire con il resto delle funzionalità dell’applicazione.

Il package del Dominio contiene la persistenza principale del progetto, con tutti i dati delle entità del dominio. Il package delle Immagini contiene i file multimediali, recuperabili tramite i metadati salvati sul Dominio. Il package Log riceve e salva le informazioni relative alle richieste svolte dai vari servizi.

GestioneAccesso segue la logica per autenticare gli utenti, collaborando con InterfacciaAccesso e il Dominio, per indirizzare l'utente a InterfacciaUtente in caso di login positivo. GestioneProfilo è il package che racchiude le funzionalità applicative del progetto. Si occupa quindi d'implementare tutta la logica relativa agli eventi, ai profili e alle loro interazioni. GestioneAggiornamenti si occupa infine di connettere GestioneProfilo e InterfacciaUtente per trasmettere le modifiche in tempo reale.

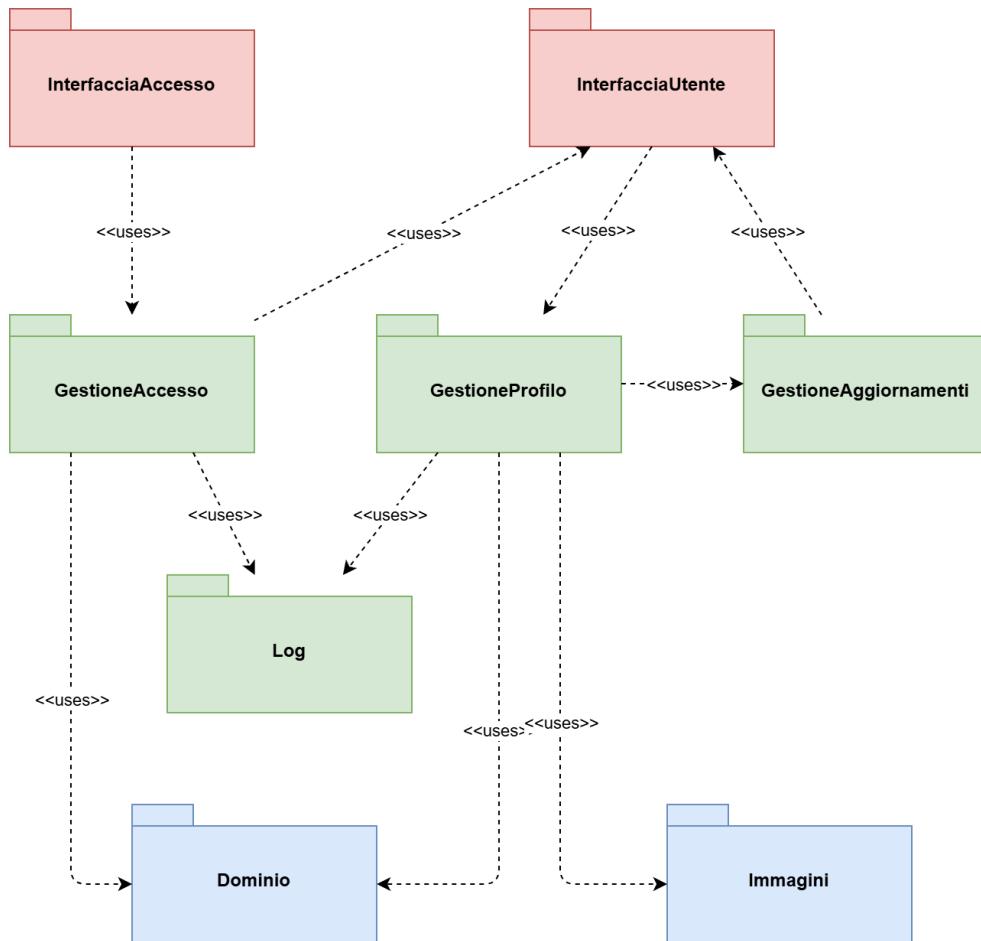


Figura 1.4: Diagramma dei Package

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.

InterfacciaUtente ha una classe per ogni funzionalità principale. Ci sono quindi le classi di ViewEventiConfermati e ViewEventiProposti, che permettono di visualizzare i relativi impegni. Attraverso queste interfacce si può interagire con ViewVisualizzaEvento, per interagire con i dettagli dell'evento. ViewGestioneGruppi presenta la lista dei gruppi con le azioni relative e permette l'accesso a ViewCercaProfili, la schermata per trovare altri profili all'interno dell'applicazione. ViewGestioneProfili è la classe che visualizza i profili dell'utente e ne permette il cambio.

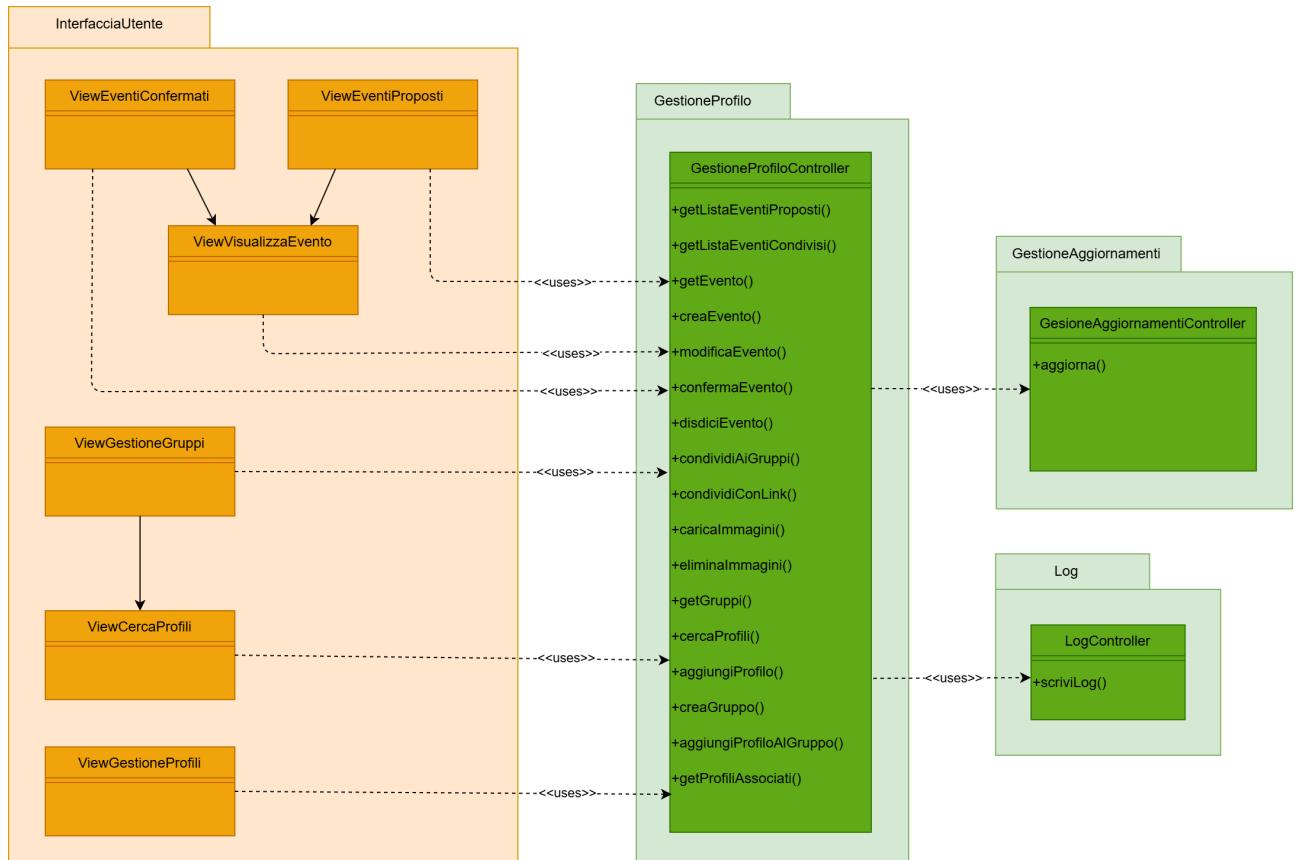


Figura 1.5: Diagramma delle classi: interfaccia utente, gestione profilo e aggiornamenti

GestioneProfilo è il package principale, a cui le classi di InterfacciaUtente si rivolgono soddisfare le richieste dell’utente. Prevede una sola classe, GestioneProfiloController, che risponde alle azioni a cui un profilo può avere accesso, ovvero tutte quelle previste. Contiene quindi le funzioni relative ai principali casi d’uso, quali a l’ottenimento degli impegni, la creazione o la conferma dell’evento o il caricamento delle immagini.

GestioneAggiornamenti ha il solo compito di aggiornare i dispositivi a seguito di una chiamata da GestioneProfilo, per cui contiene una classe con un metodo. InterfacciaAccesso prevede invece due classi, una per il Login e una per la Registrazione. GestioneAccesso ha una sola classe ma che presenta, seguendo la stessa logica, due metodi distinti. Il package Log, fornendo un solo metodo, contiene una sola classe.

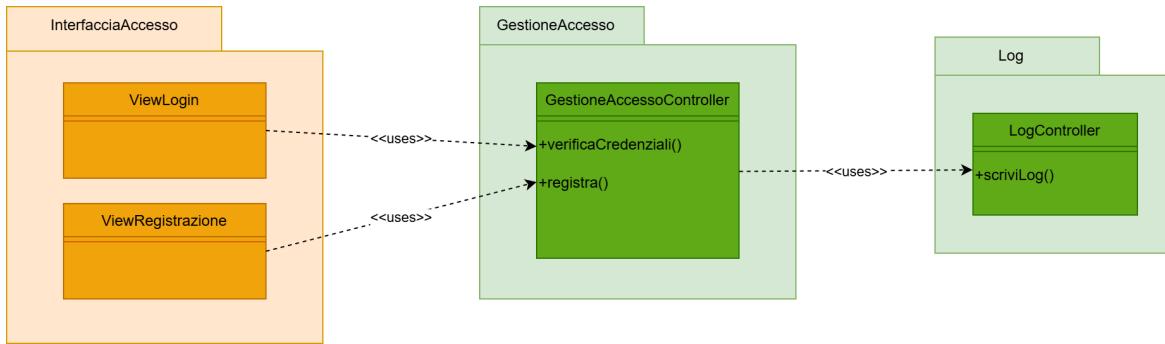


Figura 1.6: Diagramma delle classi: interfacce e gestione accesso

Capitolo 2

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.

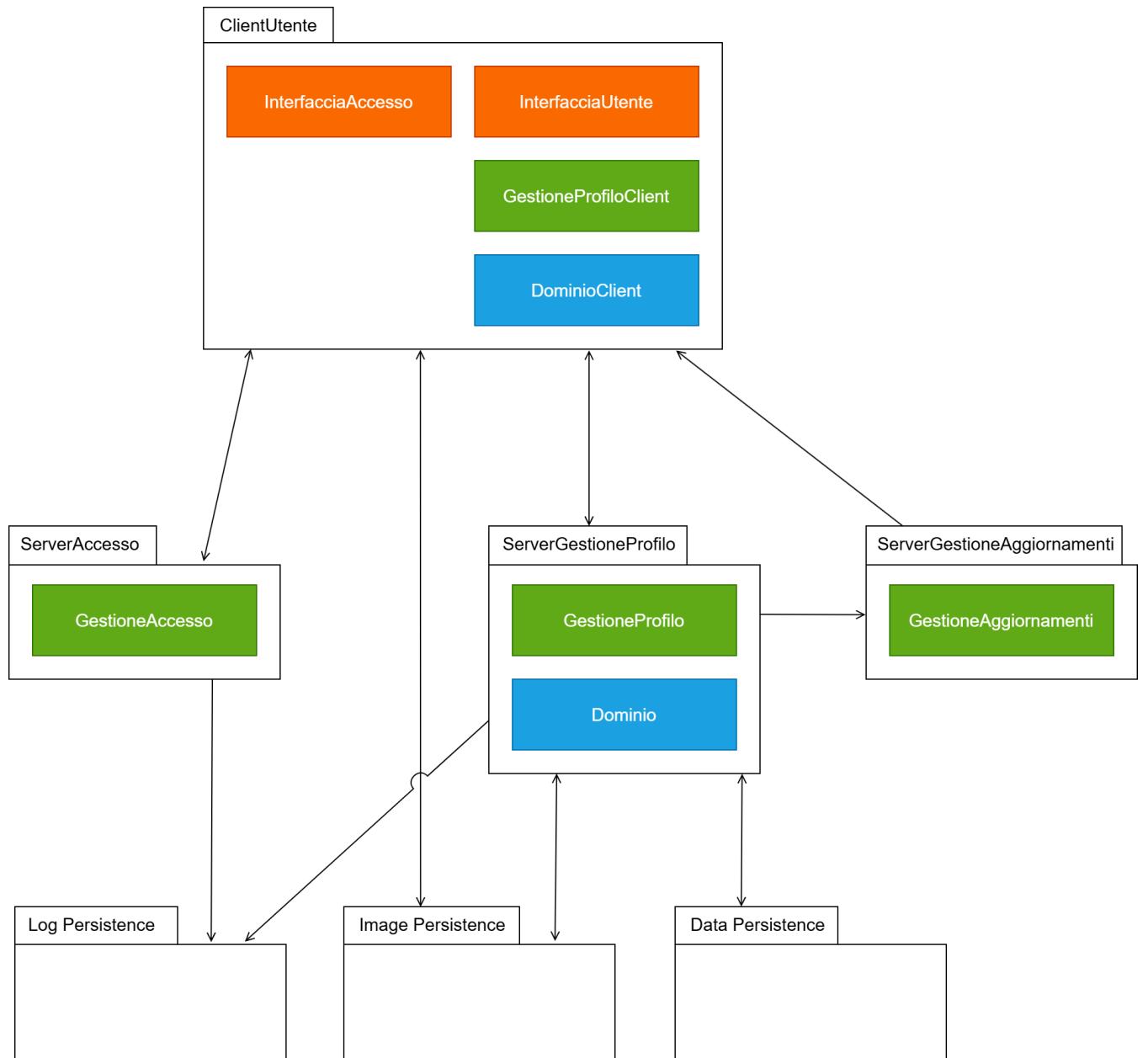


Figura 2.1: Struttura e responsabilità delle parti del progetto

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.

2.1 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.

2.1.1 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 semplificare lo sviluppo, concentrando il codice in un unico progetto e evitando quindi di dover creare una versione diversa per ogni tecnologia su cui si voglia distribuire l’applicazione. L’astrazione fornita permette inoltre di ottenere elevate prestazioni e garantire un’esperienza utente uniforme indipendentemente dal dispositivo usato. 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 mantenibile.



Flutter

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 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.

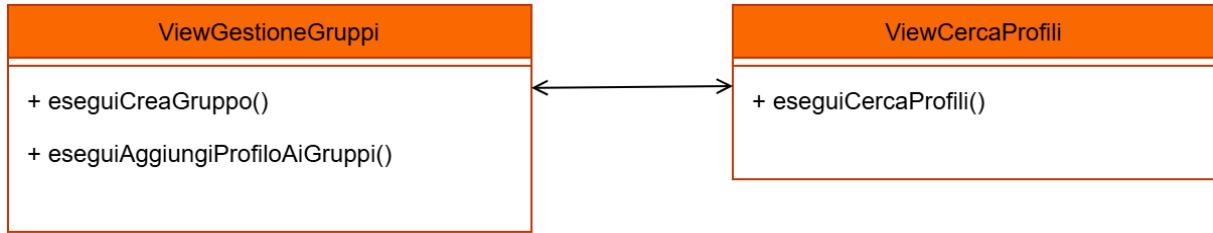


Figura 2.2: Diagramma di dettaglio delle interfacce di gestione dei gruppi

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.

The image consists of two side-by-side screenshots of a mobile application interface.

Screenshot (a) Elenco dei gruppi: This screenshot shows a list of groups and users. At the top, there is a search bar labeled "Search" and a "Groups" button. Below this is a list of items:

- prova1@mail.com
- prova2@mail.com
- prova3@mail.com
- Single Group 1_2
- Single Group 1_2_3_4
- General Community
- General Community 1
- General
- Secondary Group
- fjona.j97@gmail.com

At the bottom of the screen are three navigation icons: a square, a person icon, and another person icon.

Screenshot (b) Ricerca dei profili: This screenshot shows a search interface. At the top, there is a back arrow and a search bar labeled "Search Profile". The search bar contains the text "pro". Below the search bar is a list of profiles:

- prova1@mail.com + Add
- prova2@mail.com + Add
- prova3@mail.com + Add

(a) Elenco dei gruppi

(b) Ricerca dei profili

Figura 2.3: 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.

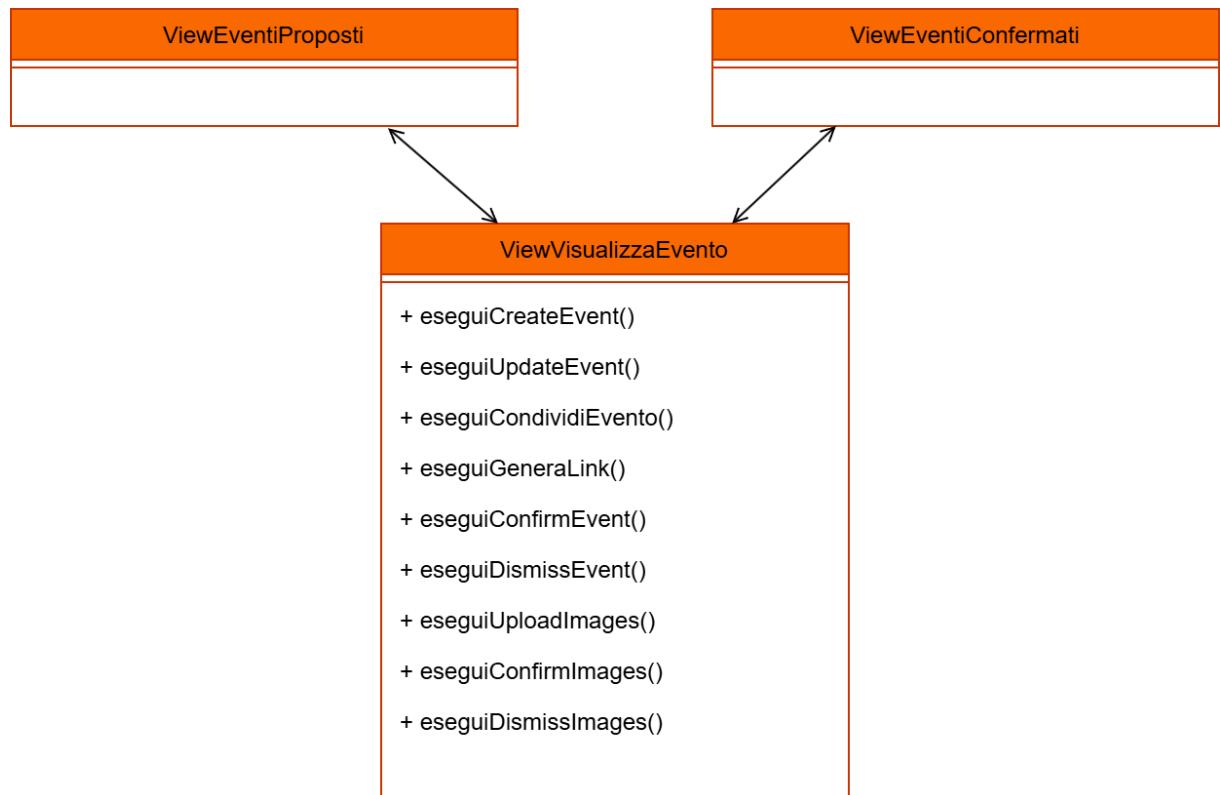


Figura 2.4: Diagramma di dettaglio delle interfacce di visualizzazione eventi

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.

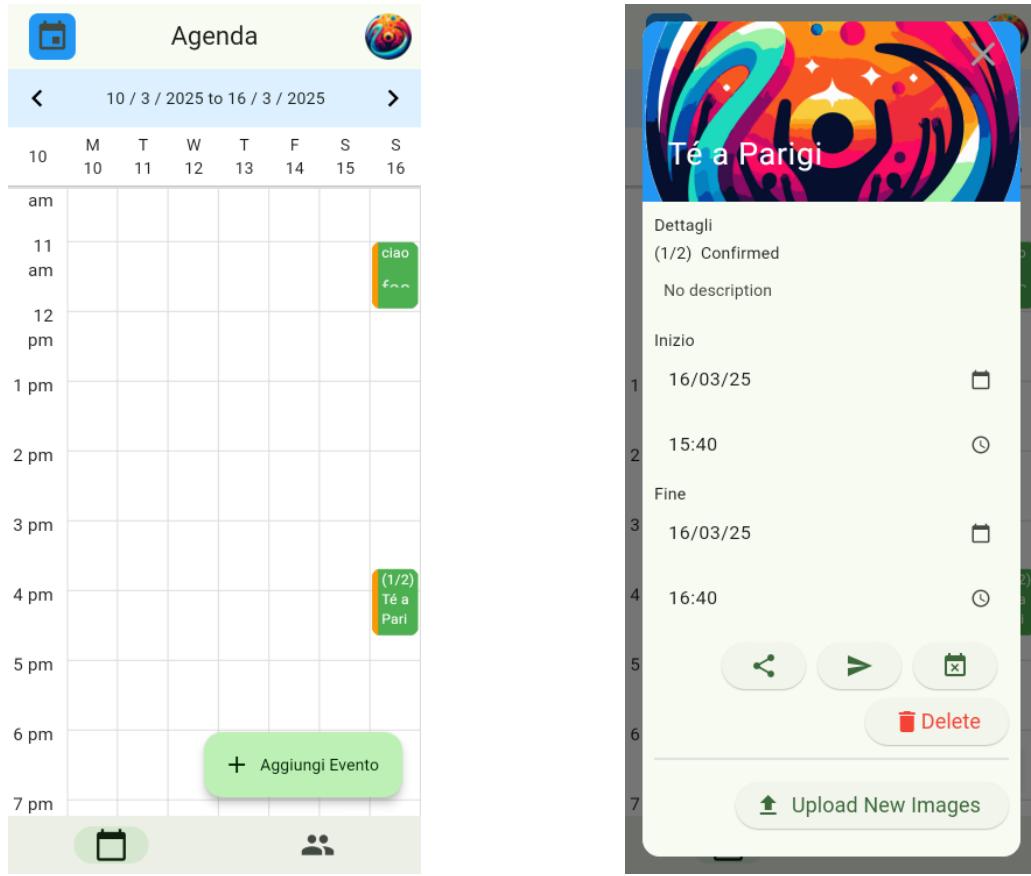


Figura 2.5: Schermate degli eventi

2.1.3 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.

2 – Capitolo 2

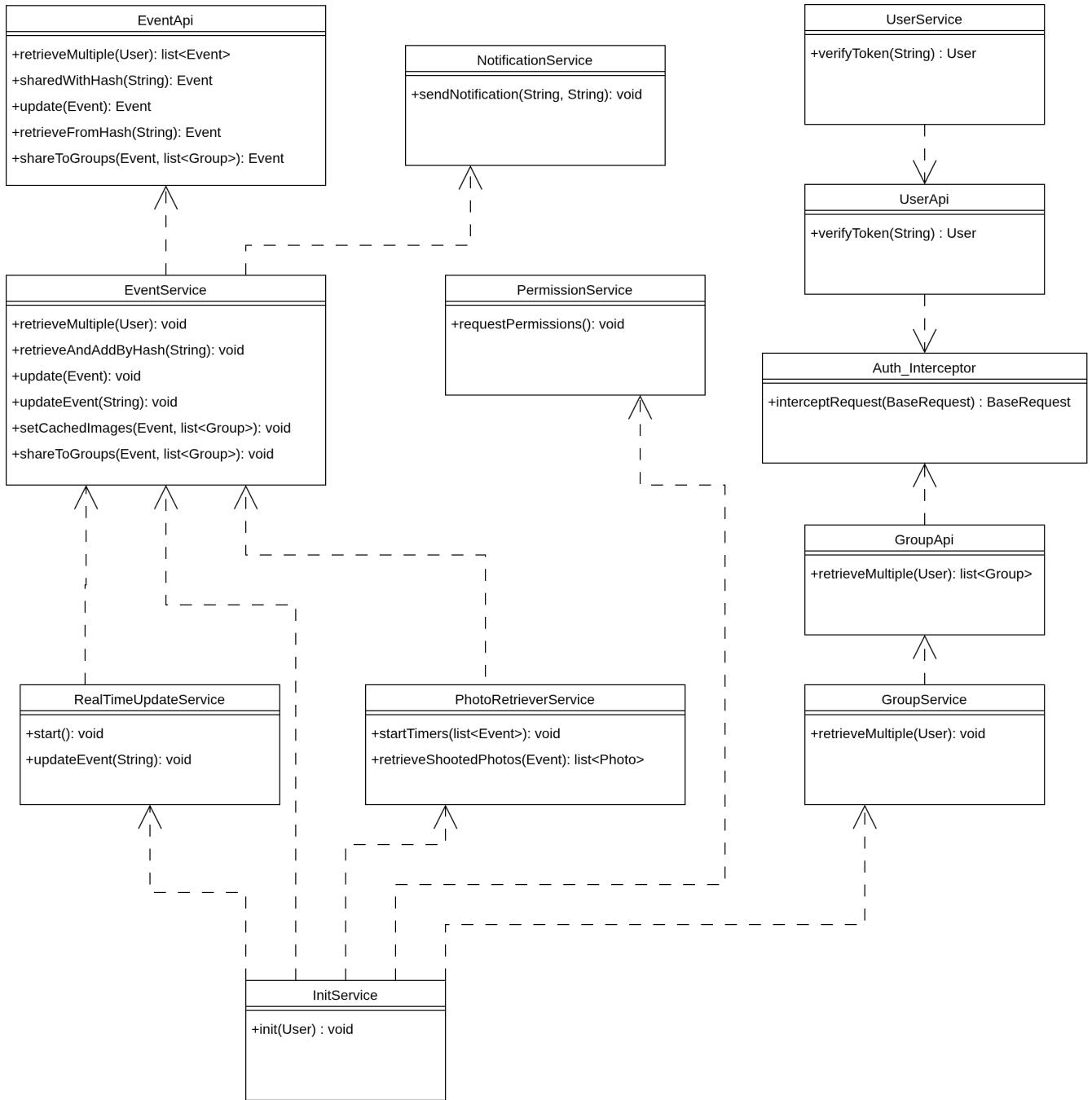


Figura 2.6: Modello delle classi del client

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 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.

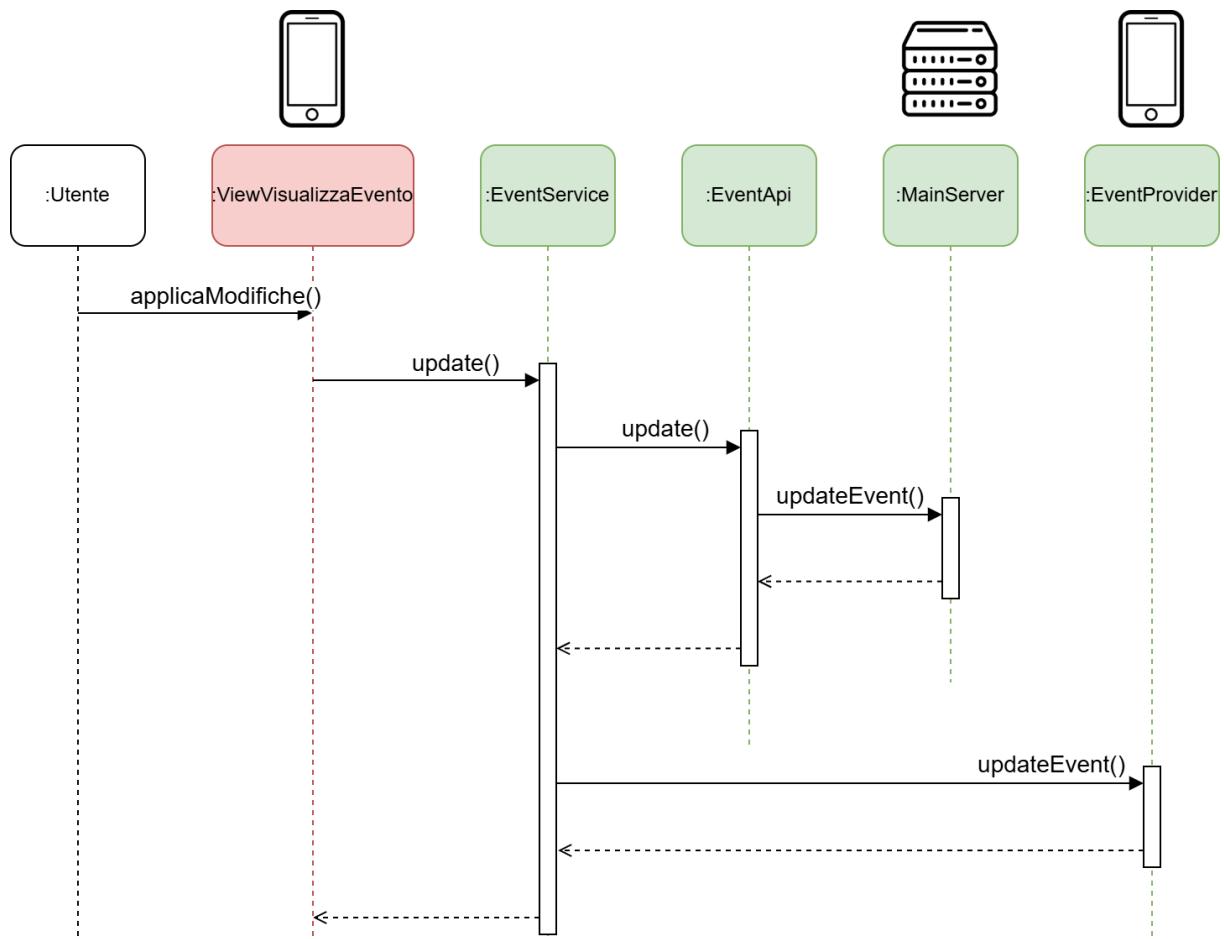


Figura 2.7: Diagramma di sequenza della modifica di un evento

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.

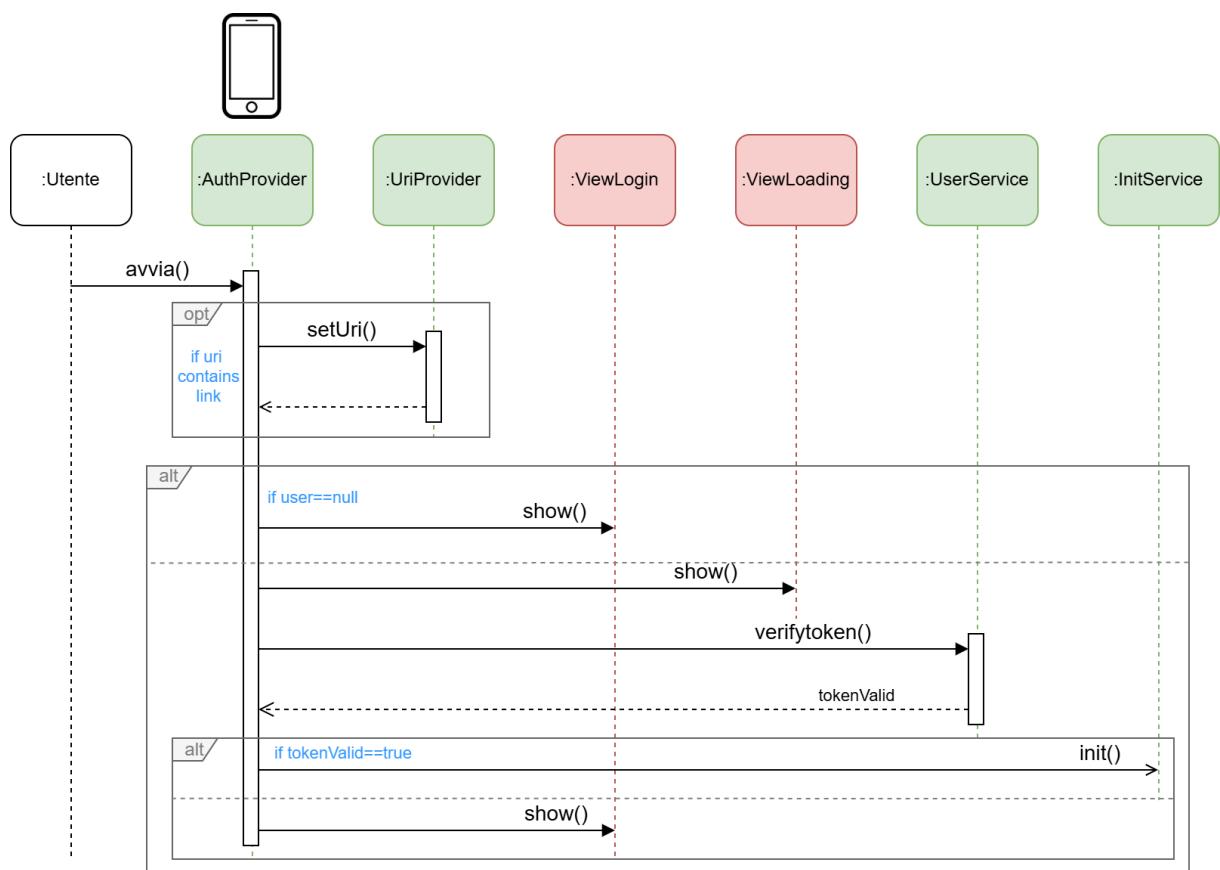


Figura 2.8: Diagramma di sequenza dell'avvio dell'applicazione

La condivisione di un evento tramite link consiste in due passaggi principali: la creazio-

ne 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.

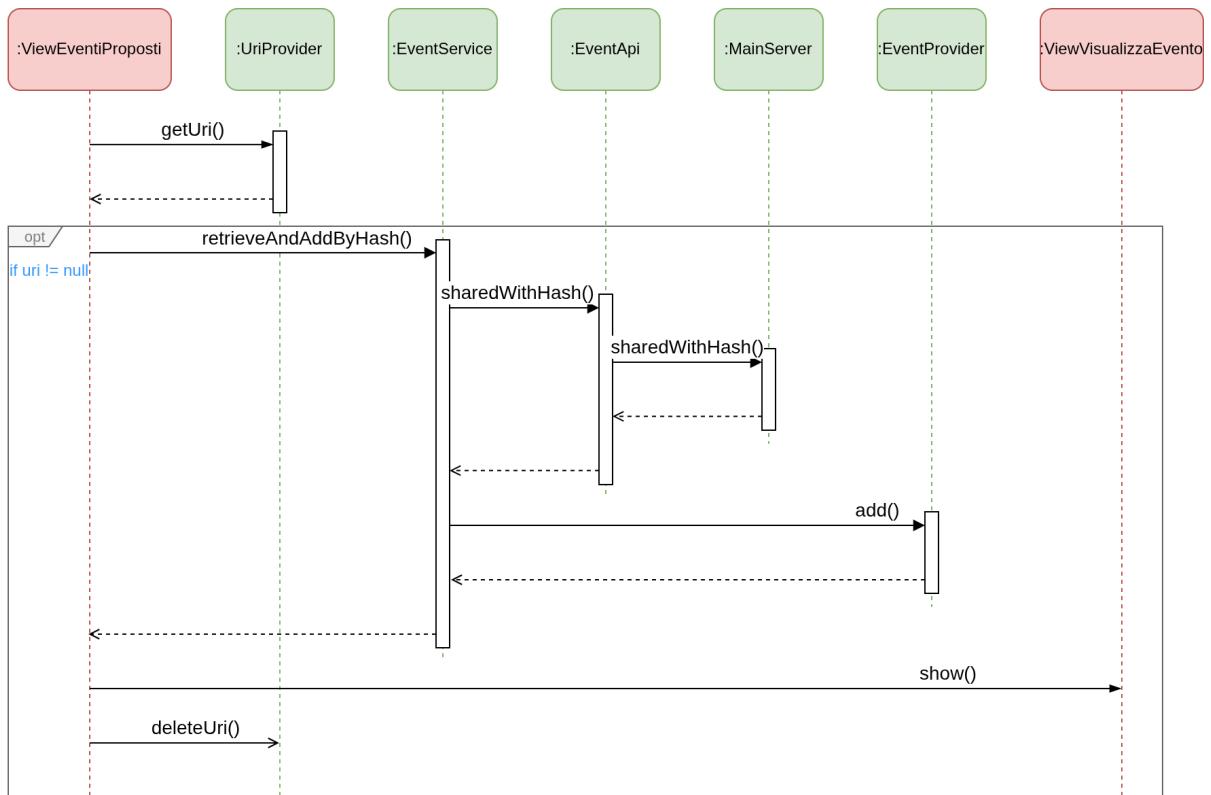


Figura 2.9: Diagramma di sequenza della visualizzazione dell'evento proposto

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.

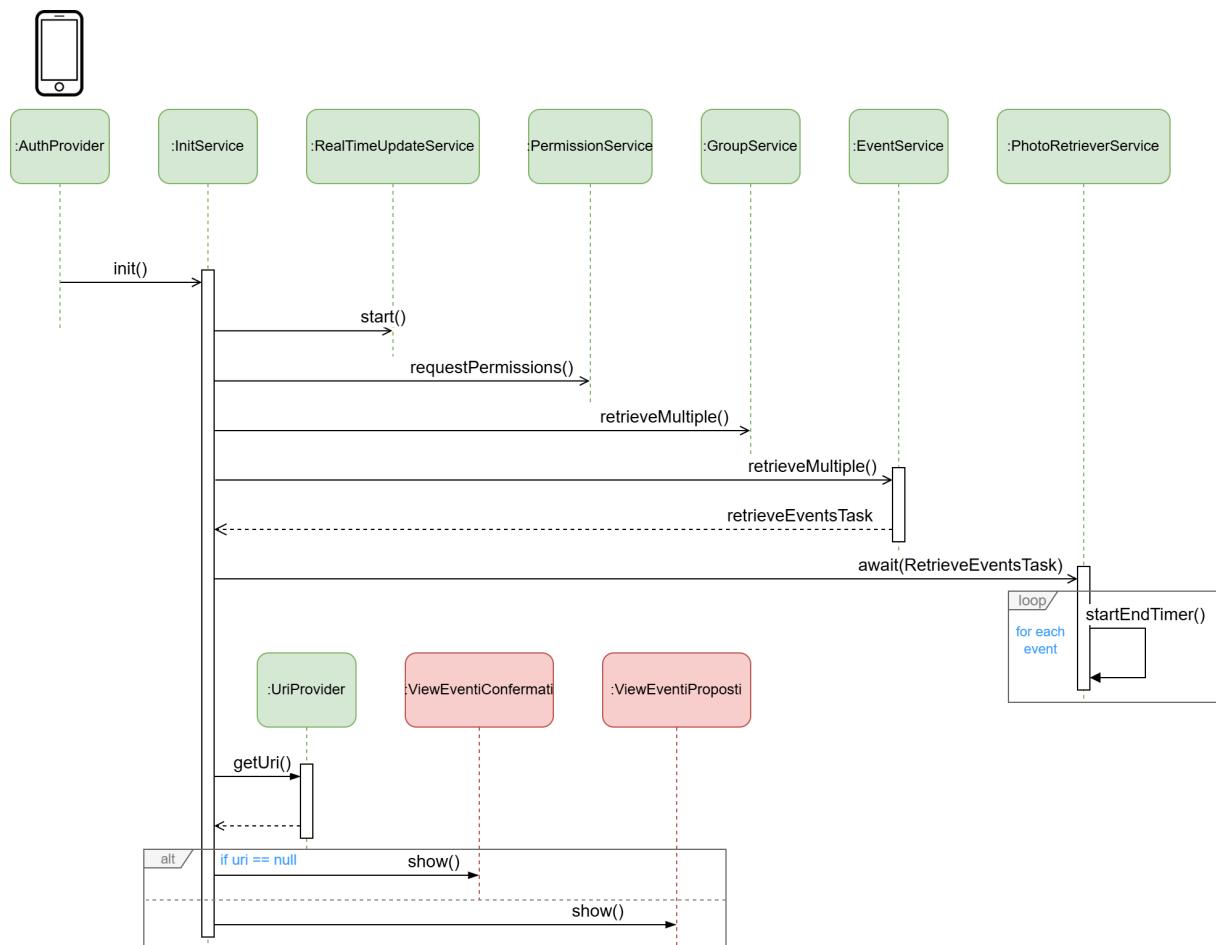


Figura 2.10: Diagramma di sequenza della fase di inizializzazione

2.1.4 Distribuzione del codice verso i dispositivi utente

Per quanto Flutter consenta di uniformare lo sviluppo e semplifichi la compilazione del codice per essere eseguito su diverse piattaforme, alcune configurazioni rimangono comunque dipendenti dalla tecnologia su cui l'applicazione viene eseguita. Di conseguenza, per ogni categoria di dispositivi per la quale si vuole distribuire l'applicazione richiede l'introduzione di una manutenzione aggiuntiva. Oltre alle modifiche derivate dalle dipendenze specifiche, bisogna anche considerare la distribuzione e la 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 virtuale apposita a soluzioni che utilizzano container. Tuttavia, Azure mette a disposizione un servizio apposito per queste precise esigenze, Azure Static Web App (SWA).



Azure Static Web
App

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.



Azure Blob Storage

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 distribuito 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.

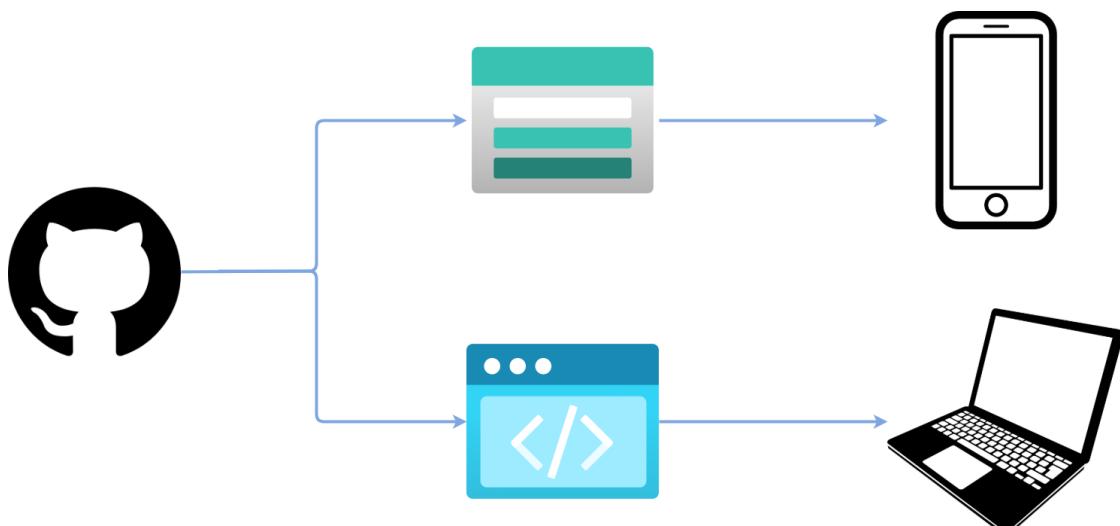


Figura 2.11: Diagramma di aggiornamento e distribuzione del client

2.2 Creazione del server principale

Il server principale è il componente con il compito di ricevere tutte le richieste dai client, 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.

2.2.1 Individuazione 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, il 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 deployment 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 2.1: 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 attivi, che, fornendo una continuità di richieste, contribuiscono a diminuire la frequenza dello spegnimento.

Essendo le differenze tra un servizio e l’altro minime, sia a livello di costi che di prestazioni, ed essendo il progetto improntato su Azure, la scelta della tecnologia su cui implementare il server principale è ricaduta sulle Azure Functions.

2.2.2 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.



Azure Functions

Innanzitutto 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 suddividerle 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.

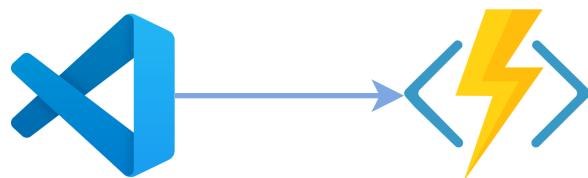


Figura 2.12: Diagramma di aggiornamento e distribuzione del server

2.2.3 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.

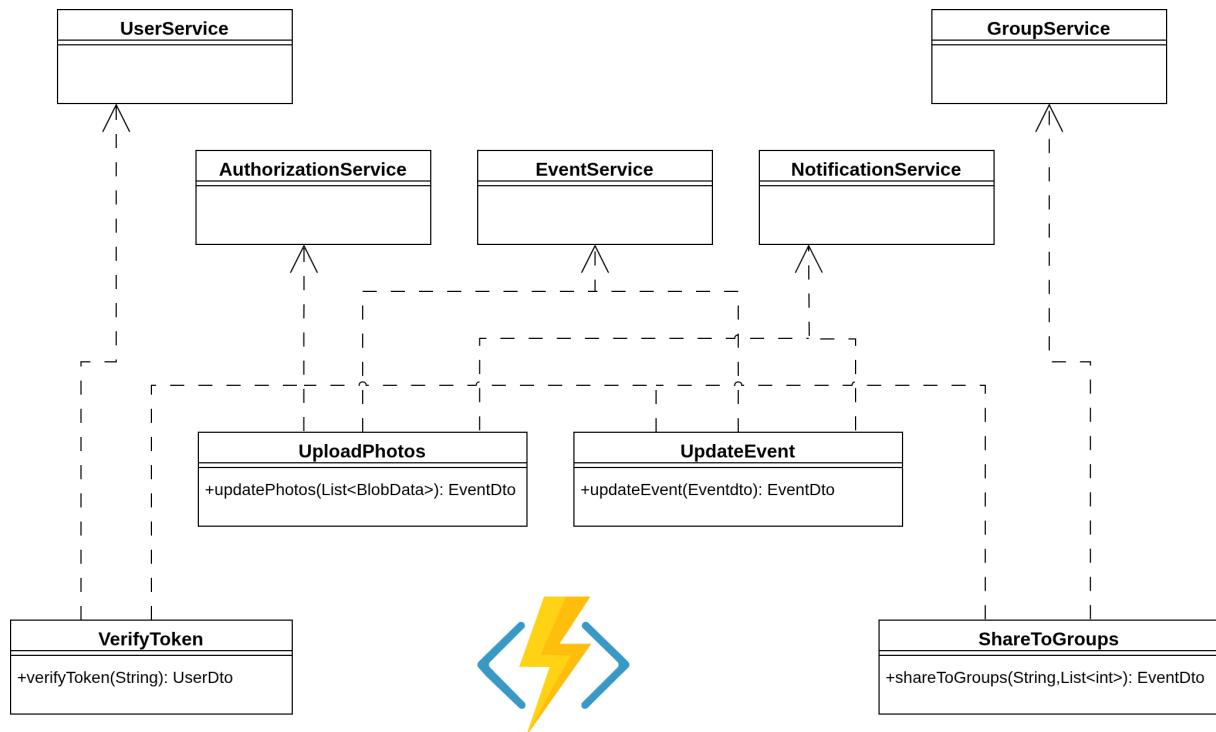


Figura 2.13: Modello delle relazioni tra Functions e i servizi usati

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.

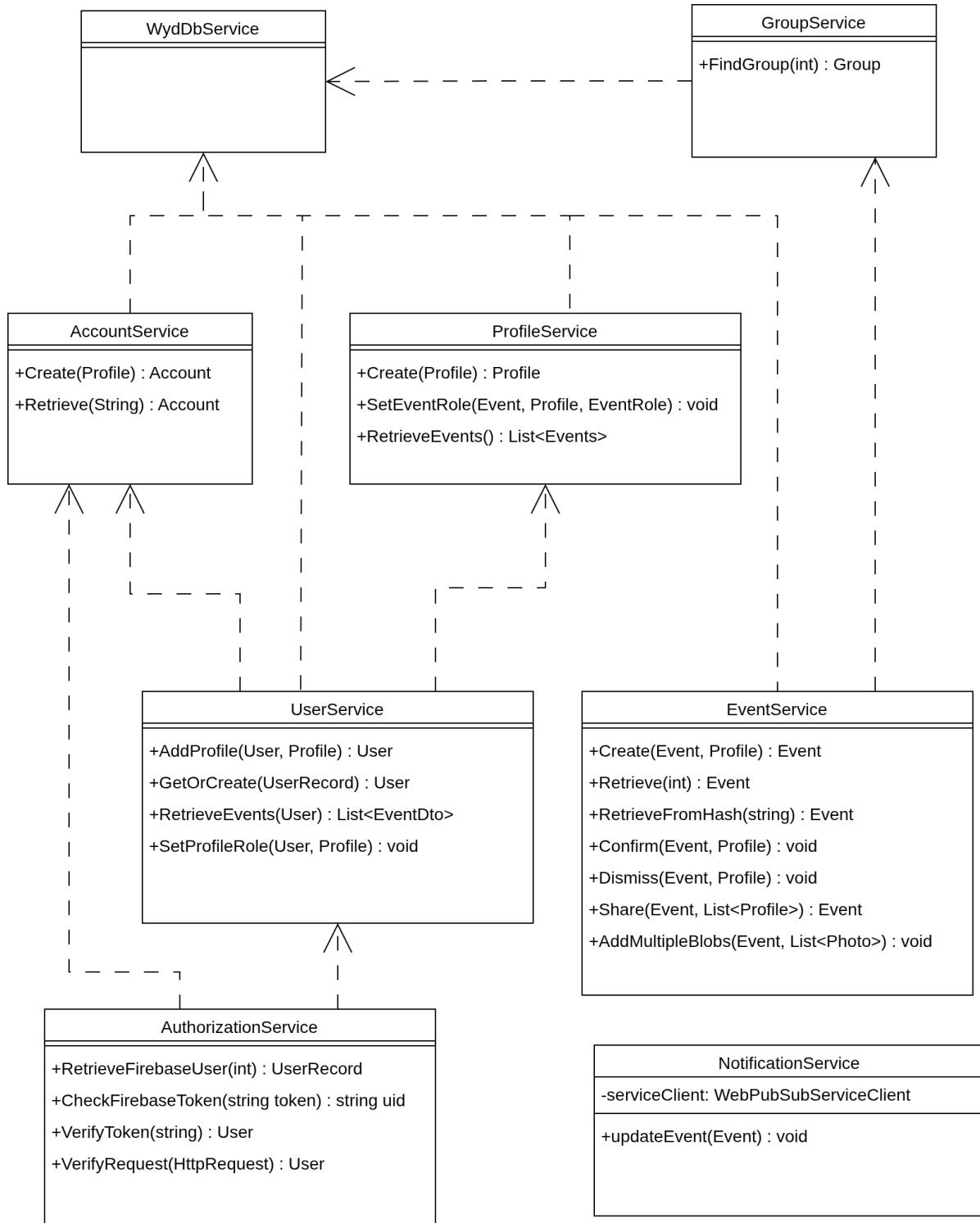


Figura 2.14: Modello delle classi del server

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

2 – Capitolo 2

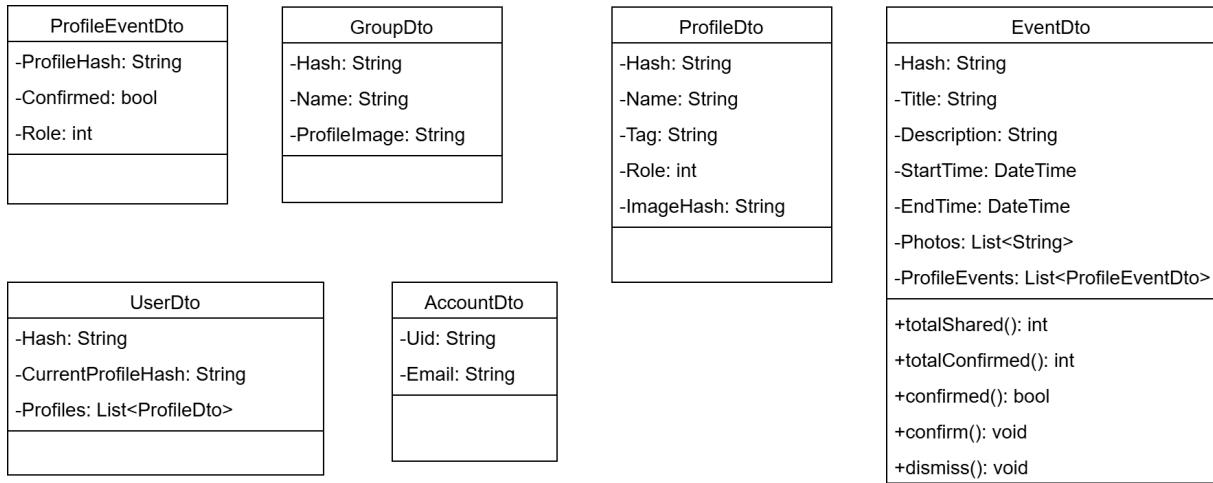


Figura 2.15: Modello delle classi dei data tranfer object

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 2.16: Test di creazione di un account

2.3 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
Authentication

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.

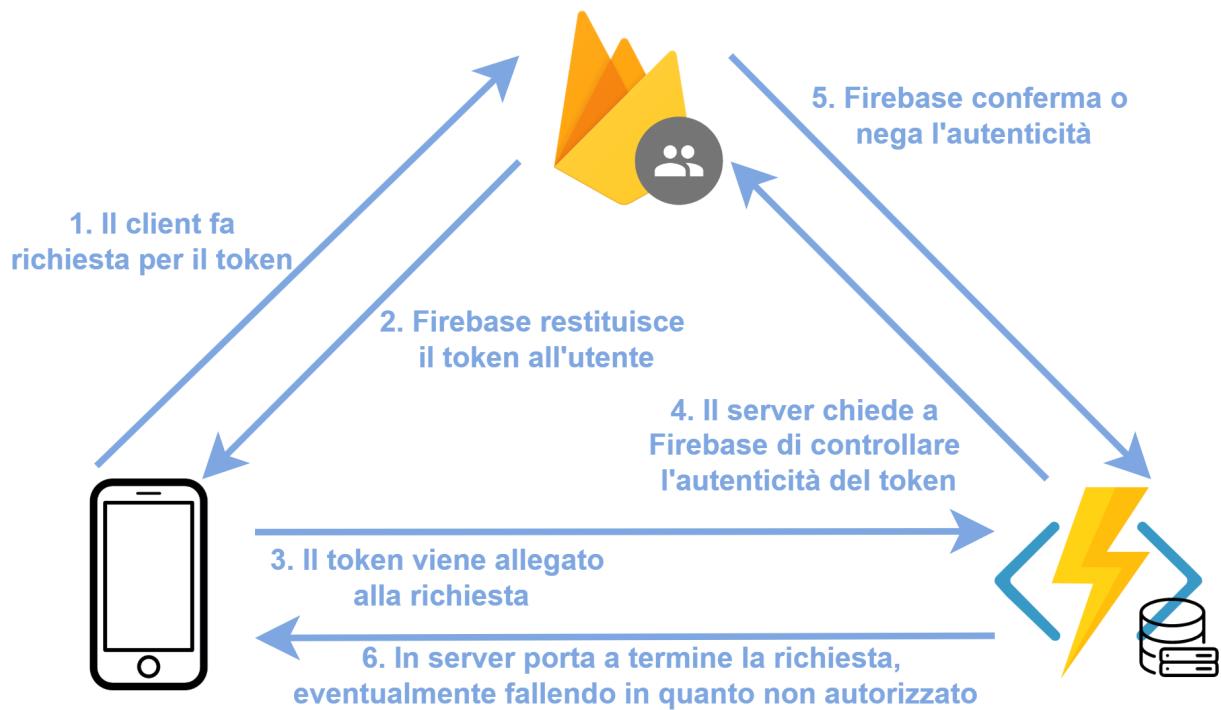


Figura 2.17: Fasi di ottenimento e uso del token

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.

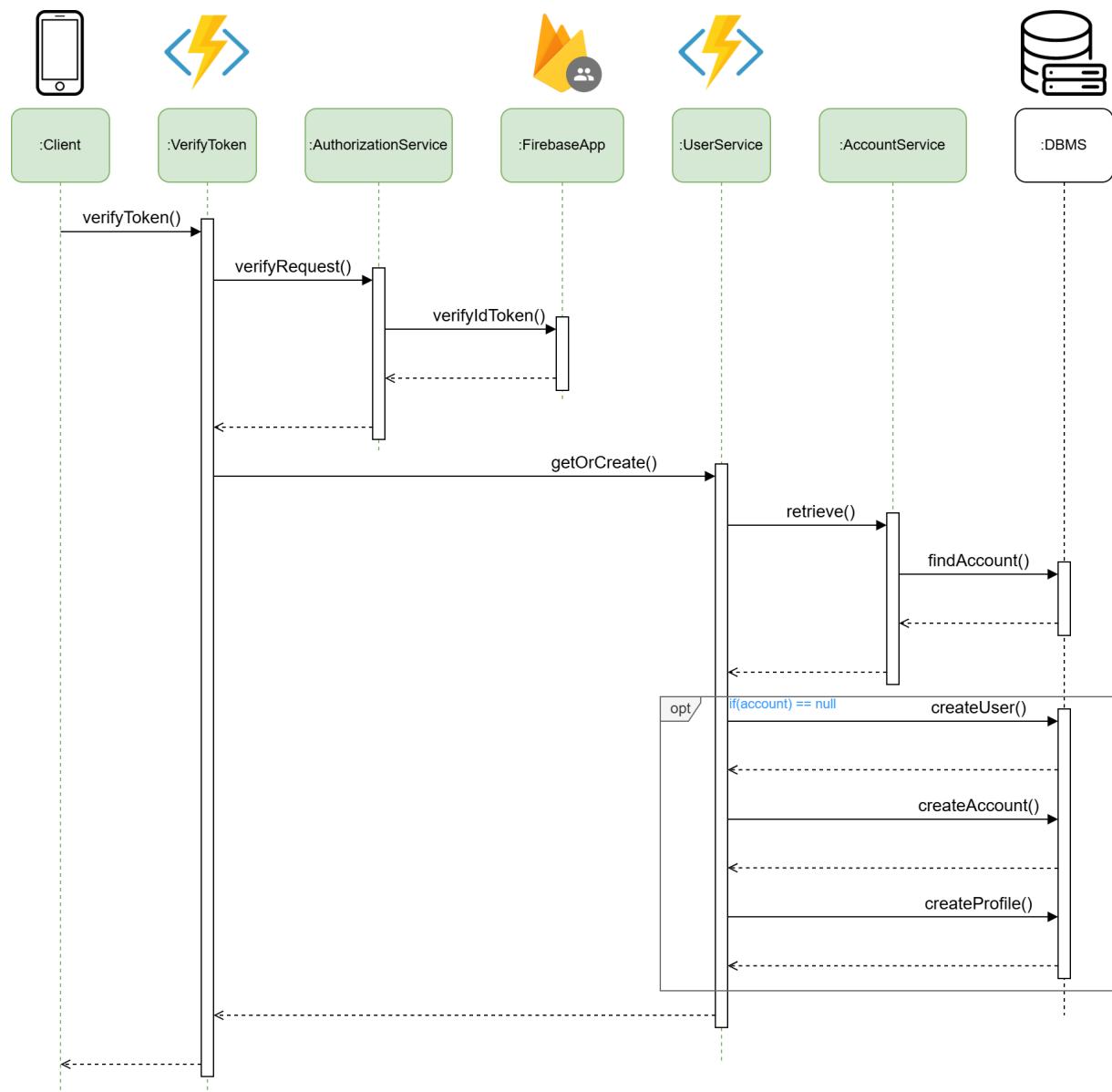


Figura 2.18: Diagramma di sequenza per la creazione di un account

2.4 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.



Azure Key Vault

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.

2.5 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.



Azure Application
Insights

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

2 – Capitolo 2

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.

Capitolo 3

Nel contesto di un sistema che prevede la gestione di eventi e impegni, è essenziale garantire all’utente di poter 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 rimane parziale, in quanto non garantisce una persistenza a lungo termine dei dati, né assicura la loro disponibilità in ogni momento, per essere in grado di sincronizzare più dispositivi. Per superare tali 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 principale. 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 ope-

rativa.

Oltre alla gestione della persistenza dei dati per il singolo utente, è necessario affrontare il problema della modifica di eventi condivisi. 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. 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 Analisi per l'identificazione del database

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 il recupero, 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 risulta più efficiente dal punto di vista prestazionale e semplifica 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, grazie a una minore complessità di sincronizzazione e di mantenimento della consistenza delle informazioni.

I database non distribuiti 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 entità in tempi estremamente rapidi, garantendo allo stesso tempo operazioni atomiche. Viceversa, i database non relazionali offrono una maggiore flessibilità strutturale, permettendo l'archiviazione di dati eterogenei e adottando un accoppiamento più debole tra i vari oggetti.

La scelta della tipologia di database più appropriata dipende direttamente dalle esigenze specifiche del progetto. Ogni prodotto è stato infatti realizzato per rispondere a una funzionalità specifica, introducendo vantaggi per un determinato caso d'uso ma comportando anche punti deboli, sia in termini di prestazioni che in termini di scalabilità. Determinare il database più adatto alle esigenze dipende quindi non solo dalle proprietà intrinseche della tecnologia, ma soprattutto di come queste riescano a risolvere i particolari problemi che il progetto presenta.

3.1.1 Proprietà dei database relazionali

I database relazionali gestiscono i dati tramite strutture chiamate schemi. Lo schema è una struttura rigida i cui campi e le relative proprietà vengono definite sin dal momento della creazione. Mantengono però un grande potere espressivo, in quanto permettono di descrivere direttamente le relazioni(e le loro proprietà) tra gli oggetti, attraverso la creazione di uno schema dedicato. Ogni elemento ha un identificativo univoco(Primary Key o PK) attraverso cui viene individuato all'interno del suo schema. Se lo schema descrive una relazione, viene identificato tramite una combinazione di identificativi derivati(Foreign Key o FK). Gli aggiornamenti agli schemi avvengono tramite un rigoroso sistema di transazioni. La transazione è il processo attraverso il quale una modifica viene portata a termine, a cui viene riservata per il tempo necessario la risorsa interessata.

Grazie alla struttura statica dei dati, unitamente alle relazioni predefinite del dominio, il tempo di recupero e analisi dei dati risulta altamente ottimizzato. L'utilizzo delle primary e foreign key consente al database di creare automaticamente indici dedicati che consentono l'individuazione di un oggetto in tempi minimi, e facilitano l'incrocio delle informazioni contenute all'interno delle relazioni. L'utilizzo delle transazioni rende i database relazionali in grado di garantire le proprietà di Atomicità, Consistenza, Isolamento e Durabilità (ACID), assicurando un'elevata affidabilità dei dati.

La rigidità del modello non permette però di avere un elemento di uno schema che presenti proprietà diverse da tutti gli altri. Non è quindi supportata l'aggiunta o la modifica di campi all'interno di un oggetto, a meno di non cambiare la definizione dell'intero schema. L'utilizzo delle transazioni introduce inoltre la necessità, durante le operazioni di scrittura, di bloccare temporaneamente le risorse interessate,facendo fallire o aspettare altre richieste simultanee sullo stesso elemento, potenzialmente influenzando le prestazioni complessive del sistema.

Per poter garantire il successo di una transazione il database deve controllare tutte le sessioni con il server, il che comporta una limitazione al numero massimo di connessioni(e quindi richieste) contemporanee possibili. Inoltre gli indici dipendono da tutti gli elementi del database, e allo stesso modo gli schemi delle relazioni sono fortemente accoppiati

con gli schemi delle entità coinvolte. Risulta quindi arduo dividere degli elementi su più tabella(sharding), operazione necessaria per la distribuzione del database su più server, aumentando di conseguenza la complessità richiesta per essere in grado di scalare orizzontalmente.

3.1.2 Proprietà dei database non relazionali

I database non relazionali, noti anche come NoSQL, si distinguono per l'adozione di modelli di archiviazione dei dati flessibili, che si discostano dalla rigida struttura tabellare dei database relazionali. Questi modelli includono approcci chiave-valore, documento, colonnare e a grafo, ciascuno ottimizzato per specifiche esigenze applicative, come la gestione di file, dati semi-strutturati, o la rappresentazione di relazioni complesse.

La loro caratteristica fondamentale è la capacità di salvare le informazioni in forme variabili, permettendo di modificare la struttura dei dati senza la necessità di riconfigurare lo schema del database. La vera forza dei database NoSQL, tuttavia, risiede nella loro predisposizione alla scalabilità orizzontale. I NoSQL sono infatti progettati per distribuire il carico su più server, consentendo di gestire volumi crescenti di dati e richieste semplicemente aggiungendo nuovi nodi al sistema.

Nonostante i vantaggi in termini di scalabilità, i database non relazionali presentano però alcune limitazioni significative. La più rilevante è la rinuncia alle proprietà ACID, alla quale si contrappone, per favorire la disponibilità e la tolleranza ai partizionamenti, una consistenza finale (eventual consistency), dove le modifiche ai dati si propagano attraverso il sistema in un certo lasso di tempo, piuttosto che essere immediatamente consistenti su tutti i nodi. Questo può portare a letture di dati non aggiornati in determinate circostanze, il che può essere un problema per applicazioni che richiedono forte consistenza, ma che deve essere comunque tenuto in considerazione durante l'analisi, per progettare un'applicazione che sia resistente a un asincronismo temporaneo dei dati.

3.1.3 Impatto delle relazioni delle entità sulle prestazioni

L’aspetto determinante alla base della 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 che ogni operazione comporta. Tra le operazioni più costose in termini di prestazioni, che più ostacola e rallenta il recupero dei dati, vi è l’operazione di unione(join). Durante l’operazione di unione vengono incrociati i dati di vari elementi per restituire un oggetto coerente che presenti tutte le proprietà necessarie, originariamente distribuite in molteplici tabelle.

Nonostante l’esecuzione di join su database sia altamente ottimizzata (particolarmente in quelli relazionali) e facilitata dall’utilizzo di indici, introduce comunque carichi computazionali di grande entità che impattano significativamente sui tempi di risposta delle richieste, crescendo proporzionalmente con la quantità dei dati presenti nel database. Per questo motivo è bene modellare il dominio nell’ottica 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 spesso un’operazione di join, ed è quindi bene eseguire un’attenta valutazione delle tipologie di accesso per ottimizzarne le prestazioni. Un’operazione di join è accettabile se il numero delle entità coinvolte è limitato o facilmente reperibile. Altrimenti, una delle strategie possibili per migliorare l’efficienza delle richieste offerte dai database non relazionali è la denormalizzazione delle entità.

La denormalizzazione consiste nel duplicare o incorporare dati correlati all’interno della stessa entità o documento, eliminando la necessità di operazioni di join complesse e costose in fase di lettura. Questo significa che, anziché avere tabelle separate tra due elementi e collegarle tramite chiavi esterne, un database denormalizzato potrebbe memorizzare direttamente un array dei dati del primo all’interno del documento del secondo. Recuperare tutti i dati necessari per una determinata operazione richiede spesso una singola lettura

da un'unica entità, e raramente un'operazione di join, riducendo drasticamente il numero di accessi al disco e le elaborazioni computazionali. Questo è particolarmente vantaggioso in scenari dove le operazioni di lettura sono molto più frequenti di quelle di scrittura.

Per ogni dato duplicato, infatti, si introduce la complessità di dover garantire la coerenza di tali dati attraverso il sistema. Se un'informazione duplicata viene modificata in una delle sue occorrenze, è essenziale che tale modifica si propaghi correttamente a tutte le altre copie per evitare che il database contenga dati incoerenti. Non esiste un meccanismo automatico che ne assicura la coerenza, e la sua creazione può diventare onerosa e complessa, soprattutto in sistemi distribuiti e con elevato volume di scritture.

In una relazione uno a molti, nel caso in cui la richiesta di quell'elemento non sia frequente ma sia invece importante restituire spesso gli elementi a lui 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.

Nel caso molti a molti bisogna considerare ancora di più la proporzione delle richieste. Le relazioni molti a molti vengono generalmente descritte da un terzo oggetto, che oltre a mantenere i riferimenti alle due entità, descrive le proprietà della relazione. Le scelte principali che si possono fare in questo caso sono due. Se la lettura è sbilanciata verso uno dei due elementi della relazione, e, allo stesso tempo, è importante che vengano restituiti gli oggetti che descrivono l'associazione assieme all'elemento stesso, è bene integrare le associazioni in quell'elemento. Altrimenti, in caso la necessità di lettura sia equiparabile da entrambe le parti, è necessario mantenere l'associazione come documento indipendente, eventualmente copiando i dati richiesti per evitare di dover recuperare il terzo elemento della relazione.

3.1.4 Analisi del dominio

Il dominio descrive i componenti dell'applicazione e le loro relazioni. Ne vengono espresse le dipendenze, i rapporti reciproci e la cardinalità delle relazioni. La sua analisi, integrata con la previsione del carico delle richieste, permette di fornire un quadro dettagliato sulle necessità relative, per poter poi definire la tipologia delle strutture in cui salvare i dati e le caratteristiche richieste al database.

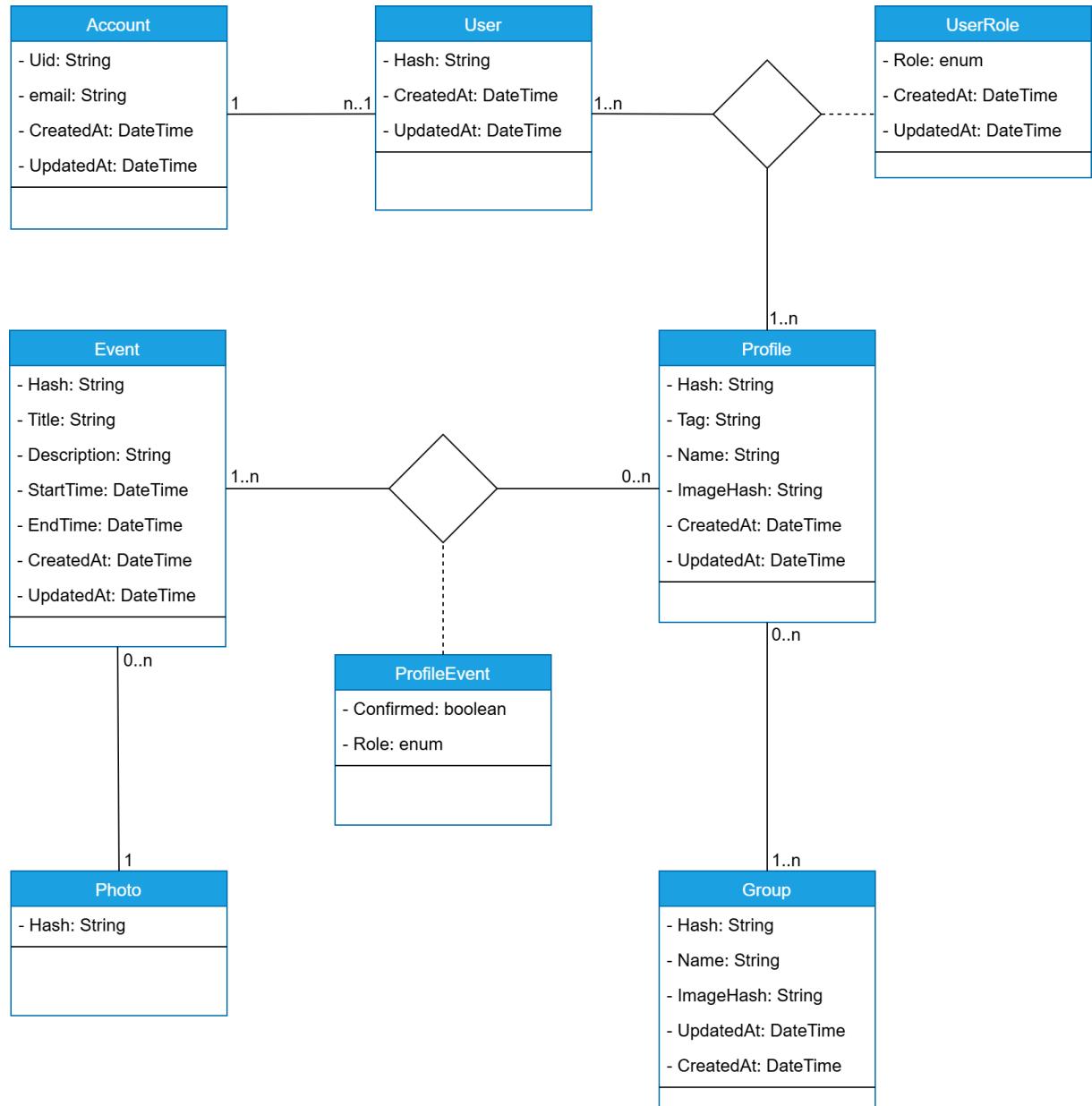


Figura 3.1: Diagramma del dominio di Wyd

Le entità Account, User e Profile, assieme alla relazione UserRole, descrivono le specifiche di autenticazione, identificazione dell’utente e i suoi ruoli sui profili associati. I loro dati vengono recuperati solamente una volta nel ciclo di vita del programma, a seguito del login dell’utente. Vengono poi salvati in memoria locale, riducendo il numero di richieste relative successive a un controllo su eventuali aggiornamenti. Si prevede che le modifiche a questi elementi siano sporadiche.

Allo stesso modo gli oggetti Group vengono recuperati solo all’avvio dell’applicazione, e, salvo rari aggiornamenti, non comportano ulteriori richieste. Le richieste di lettura e scrittura previste per questi elementi sono quindi in quantità esigua, e influiscono secondariamente sulle performance del sistema.

La maggioranza delle richieste verterà sull’ottenimento dei dati relativi agli Event e ai Profile. Gli Event descrivono gli impegni ai quali i profili partecipano. Essendo la loro relazione di tipologia molti a molti, si introduce l’entità ProfileEvent che descrive l’associazione evento-profilo, indicando che un evento è condiviso con un profilo. La proprietà più importante di ProfileEvent è sicuramente Confirmed, una variabile booleana che esprime la partecipazione o meno di un profilo all’evento. È ragionevole prevedere che la cardinalità dei profili associati a un evento non superi l’ordine delle centinaia. Agli eventi che si associano ai profili l’ordine di grandezza invece previsto risiede nelle migliaia. Ci sono molteplici situazioni da tenere in considerazione analizzando questa relazione.

Sicuramente bisogna considerare il recupero specifico delle entità Event e Profile. Questo accade quando un utente decide di entrare nel dettaglio di uno dei due elementi, comportando la richiesta di tutti i dati dell’oggetto. Consistono in un’operazione di lettura tutto sommato semplice: essendo l’utente in possesso dell’identificativo dell’elemento, la ricerca risulta diretta, e i dati da recuperare esigui.

La conferma o la disdetta a un evento vede la modifica di un ProfileEvent. Questa operazione di per sé risulta veloce in quanto necessita di recuperare un solo elemento per poi modificarlo. Allo stesso modo la modifica di un evento comporta l’aggiornamento di un solo oggetto Event, a seguito però del recupero del ProfileEvent associato, necessario per controllare i permessi del profilo sull’evento. Entrambe le operazioni richiedono però

l’invio di una notifica verso tutti i profili interessati, che necessita il recupero di tutti i ProfileEvent associati ogni volta che una di queste accade.

La probabilità che un evento o un qualunque ProfileEvent a esso associato venga modificato risulta quindi complessivamente elevata e, di conseguenza, l’operazione di recupero degli identificativi di tutti i profili associati all’evento è da considerarsi frequente. Questa operazione, in base all’implementazione, può risultare costosa: una separazione, un accoppiamento debole o la mancanza di indici tra Event e ProfileEvent determinano la necessità di ricercare le entità in tutto il database.

Quando un utente accede per la prima volta su un dispositivo è necessario ottenere gli eventi associati ai suoi profili. Allo stesso modo, a ogni avvio dell’applicazione si recuperano gli eventi che hanno subito modifiche dall’ultimo accesso. Inoltre, per poter garantire(all’interno di un determinato vincolo temporale) la consistenza dei dati anche a livello locale, il client applica una strategia di long polling per ottenere gli eventi che sono stati modificati dall’ultimo aggiornamento noto. L’operazione di ritrovamento degli eventi a partire dal profilo risulta quindi centrale e frequente, per quanto possa accettare un tempo di esecuzione leggermente più lungo.

Funzionalità	Frequenza	Complessità
Recupero di un Event	Media	Semplice
Recupero di un Profile	Media	Semplice
Modifica di un Event	Media	Semplice
Conferma/disdedda di un Event	Alta	Semplice
Ritrovamento dei Profile associati a un Event	Alta	Complessa
Ritrovamento degli Event associati a un Profile	Alta	Complessa

Tabella 3.1: Funzionalità principali tra Event e Profile

La relazione tra Event e Photo impatta sulle prestazioni del sistema solo in casi particolari e verrà affrontato nei capitoli successivi.

3.2 Implementazione del database

Nelle sezioni precedenti si è discusso delle proprietà offerte dai diversi tipi di database e delle necessità che il dominio impone sul sistema. La scelta del database deriva quindi dall’incrocio di tutte queste condizioni, individuando la tecnologia che meglio riesce a rispondere alle esigenze del progetto. Ogni tipologia di database comporta un approccio differente alle informazioni, implicando una strategia di salvataggio e manipolazione dei dati propria. Le strutture che modellano le entità devono quindi essere create per sfruttare nella maniera più efficiente possibile i vantaggi offerti dalla tecnologia scelta.

Una volta scelto il database e le strutture in base alla modalità che più si addicono alle esigenze del progetto, l’utilizzo di servizi in cloud comporta una maggiore attenzione anche alle proprietà legate al mantenimento del servizio, dalle quali derivano le proprietà di scalabilità e affidabilità. 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 determinano infatti 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 su più 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 a un punto precedente rispetto all’avvenimento di eventuali errori o guasti del sistema.

3.2.1 Scelta del database

Viste le necessità del progetto in ambito di scalabilità e le caratteristiche del dominio, si individua nei database documentali la tecnologia più adatta per gestire la persistenza centrale dell’applicazione.

I database documentali, facenti parte della categoria dei database non relazionali, rappresentano un paradigma di gestione dei dati che organizza le informazioni in documenti. Ogni documento è un’unità autonoma che incapsula la descrizione di un’entità, contenendo le sue proprietà. Tali documenti sono logicamente raggruppati in collezioni. All’interno di una collezione, ciascun documento è univocamente identificato da un proprio identificativo, garantendo l’accesso diretto e la manipolazione individuale.

Un aspetto distintivo e strategicamente rilevante dei database documentali è la loro intrinseca capacità di supportare la scalabilità orizzontale in modo nativo. Data la natura dei documenti e il loro accoppiamento debole con gli altri elementi, la separazione delle collezioni risulta particolarmente semplificata, favorendo un partizionamento dei dati, essenziale per distribuire il database su diversi nodi fisici di archiviazione. Questo vantaggio è fondamentale in architetture distribuite e ambienti ad alta intensità di dati.

Un altro vantaggio derivato dall'utilizzo di un database non relazionale è la propensione verso la denormalizzazione degli elementi del dominio, sia strutturale che logica. La denormalizzazione consiste nel riportare le stesse proprietà degli elementi in più collezioni, riducendo così, se correttamente ottimizzata, il numero di join o richieste al database necessario per soddisfare le richieste. Infatti, vista la riduzione di efficienza nell'incrocio tra entità, dovuta a una difficoltà intrinseca nell'ottimizzare le join tra documenti, si propende, invece che a incrociare i dati, a duplicarli sui vari documenti. Le operazioni di join sono così agevolate, in quanto possono essere ottimizzate per far risiedere la risorsa voluta all'interno dello stesso documento o nella stessa partizione dell'elemento di partenza, minimizzando la necessità di operazioni di giunzione di tabelle o di lettura tra nodi distinti. Questo permette di aggregare i dati attorno agli elementi chiave su cui vertono le richieste, favorendo la divisione dei dati anche in caso di relazioni complesse.

La denormalizzazione comporta però intrinsecamente alcune sfide a livello di consistenza dei dati, in particolare per le operazioni di modifica che coinvolgono informazioni duplicate. Infatti, pur se si modificasse atomicamente il documento direttamente coinvolto dalla modifica, bisognerebbe comunque aggiornare tutte le altre parti che includono quella proprietà. Oltre a dover progettare l'applicazione affinché sia resiliente all'inconsistenza temporanea, ad esempio mostrando dati "vecchi" per un breve periodo prima che le modifiche vengano propagate, o implementando logiche di compensazione che possano correggere eventuali discrepanze, è quindi necessario implementare la logica per garantire che la modifica venga propagata correttamente. Esistono diverse strategie gestire le informazioni duplicate in modo efficace e mitigare l'impatto. Una delle soluzioni avviene tramite trigger a livello di database, che scatena la chiamata che corregge poi i dati ove necessario. Le code di messaggistica prevedono invece un orchestratore per la distribuzione degli aggiornamenti, che riceve notifiche delle variazioni e le elabora per poi applicare le modifiche. Un'altra tecnica è l'applicazione di servizi in background che periodicamente scansionano e sincronizzano i dati. Infine, si possono adottare timestamp o numeri di versione su ciascun documento o campo denormalizzato, permettendo alle applicazioni di determinare la versione più recente di un dato, risolvendo i conflitti quando si presentano aggiornamenti concorrenti o ritardati.

Implementando automaticamente e nativamente la scalabilità orizzontale, il database relazionale ci permette quindi di gestire con efficienza l’incremento dei volumi di dati e dei carichi di lavoro senza interventi complessi. Fornisce inoltre un supporto diretto all’esigenza dell’architettura riguardo alla necessità di letture performanti da entrambi i lati di relazioni molti-a-molti: attraverso il partizionamento strategico, i dati correlati possono essere collocati in partizioni vicine per ottimizzare le letture da entrambi i lati della relazione. Infine, un’attenta progettazione del modello di dati, che include una denormalizzazione strategica e l’utilizzo degli indici, garantirà un tempo di recupero ridotto per le informazioni, massimizzando la reattività del sistema e l’efficienza complessiva.

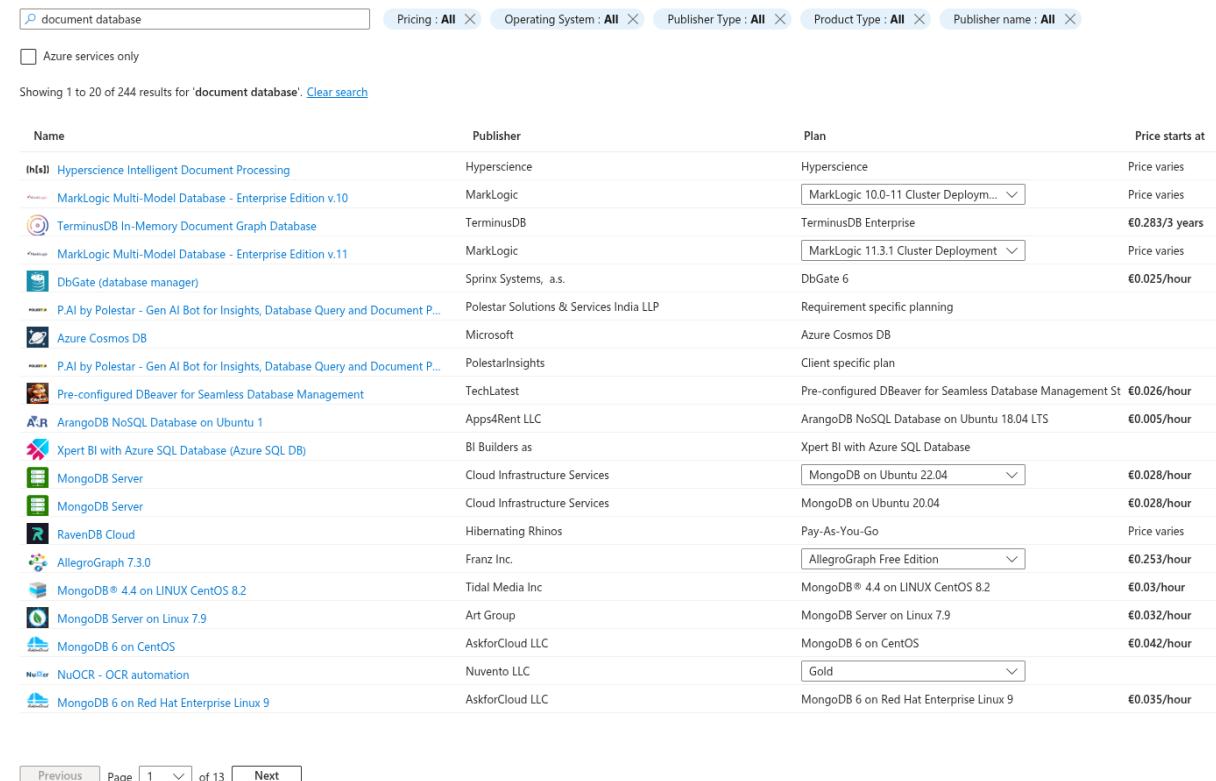
Un confronto con il paradigma relazionale evidenzia le ragioni della sua esclusione per le esigenze del nostro progetto. Sebbene i database relazionali siano soluzioni consolidate per la gestione di dati strutturati, presentano delle limitazioni che non si allineano con i requisiti di scalabilità richiesti. La necessità di controllare le transazioni al fine di garantire le proprietà ACID influenza il numero massimo di connessioni contemporanee che possono gestire, limitando la capacità di rispondere a un numero massiccio di richieste simultanee. Inoltre la loro architettura non prevede una separazione fisica di schemi in relazione tra loro, legandole alla stessa partizione logica. Questo impone intrinsecamente dei vincoli sulla scalabilità orizzontale, in quanto l’implementazione dello sharding, sebbene possibile, viene lasciata interamente a carico dello sviluppatore, introducendo un significativo onere di progettazione, sviluppo e manutenzione.

La gestione di relazioni molti-a-molti nel modello relazionale si pone infatti in diretto contrasto con la denormalizzazione dei dati, utilizzando un’unica tabella di giunzione per descriverne il rapporto. Se si provasse a dividere in partizioni gli schemi relazionali, dalla distribuzione della tabella di giunzione ne conseguirebbe uno svantaggio, indipendentemente dalla strategia usata. Separando la tabella in base a un elemento si andrebbe infatti a compromettere l’abilità di ritrovare i dati in base all’altro elemento e viceversa: per quanto si avrebbero tutti i dati di un elemento dell’associazione sulla stessa partizione, sfruttando al massimo la velocità di unione tra schemi dei relazionali, per eseguire la ricerca in senso inverso bisognerebbe invece allargare la richiesta a tutte le partizioni

3 – Capitolo 3

del sistema. In un ambiente distribuito e con volumi di dati in crescita, le join richiedono quindi l’analisi e il trasferimento di grandi quantità di dati tra nodi diversi, riducendo le performance complessive. Questi fattori combinati ci hanno portato a escludere il modello relazionale.

Essendo il progetto già improntato sulla piattaforma Azure, la ricerca verte inizialmente tra le opzioni che mette a disposizione. Azure offre un’ampia scelta di database documentali che possono essere integrati con il resto dell’ecosistema. Tuttavia, Azure presenta un servizio completamente gestito e nativo per i database non relazionali chiamato Azure Cosmos DB. Garantendo la massima interoperabilità all’interno dell’ecosistema, si procede analizzando le proprietà e i vantaggi offerti da Cosmos DB.



The screenshot shows the Azure Marketplace search results for 'document database'. The search bar at the top contains the query 'document database'. Below the search bar are several filter buttons: 'Pricing : All', 'Operating System : All', 'Publisher Type : All', 'Product Type : All', and 'Publisher name : All'. There is also a checkbox labeled 'Azure services only' which is unchecked. The results section displays 20 out of 244 items, each with a small icon, the product name, the publisher, the plan, and the price start. The results include various document databases like Hyperscience Intelligent Document Processing, MarkLogic Multi-Model Database - Enterprise Edition v.10, TerminusDB In-Memory Document Graph Database, MarkLogic Multi-Model Database - Enterprise Edition v.11, DbGate (database manager), P.AI by Polestar - Gen AI Bot for Insights, Database Query and Document P..., Azure Cosmos DB, P.AI by Polestar - Gen AI Bot for Insights, Database Query and Document P..., Pre-configured DBeaver for Seamless Database Management, ArangoDB NoSQL Database on Ubuntu 1, Xpert BI with Azure SQL Database (Azure SQL DB), MongoDB Server, MongoDB Server, RavenDB Cloud, AllegroGraph 7.3.0, MongoDB® 4.4 on LINUX CentOS 8.2, MongoDB Server on Linux 7.9, MongoDB 6 on CentOS, NuOCR - OCR automation, and MongoDB 6 on Red Hat Enterprise Linux 9. The 'Azure Cosmos DB' entry is highlighted in blue, indicating it is the selected item.

Name	Publisher	Plan	Price starts at
Hyperscience Intelligent Document Processing	Hyperscience	Hyperscience	Price varies
MarkLogic Multi-Model Database - Enterprise Edition v.10	MarkLogic	MarkLogic 10.0-11 Cluster Deploym...	Price varies
TerminusDB In-Memory Document Graph Database	TerminusDB	TerminusDB Enterprise	€0.283/3 years
MarkLogic Multi-Model Database - Enterprise Edition v.11	MarkLogic	MarkLogic 11.3.1 Cluster Deployment	Price varies
DbGate (database manager)	Sprinx Systems, a.s.	DbGate 6	€0.025/hour
P.AI by Polestar - Gen AI Bot for Insights, Database Query and Document P...	Polestar Solutions & Services India LLP	Requirement specific planning	
Azure Cosmos DB	Microsoft	Azure Cosmos DB	
P.AI by Polestar - Gen AI Bot for Insights, Database Query and Document P...	PolestarInsights	Client specific plan	
Pre-configured DBeaver for Seamless Database Management	TechLatest	Pre-configured DBeaver for Seamless Database Management St	€0.026/hour
ArangoDB NoSQL Database on Ubuntu 1	App4Rent LLC	ArangoDB NoSQL Database on Ubuntu 18.04 LTS	€0.005/hour
Xpert BI with Azure SQL Database (Azure SQL DB)	BI Builders as	Xpert BI with Azure SQL Database	
MongoDB Server	Cloud Infrastructure Services	MongoDB on Ubuntu 22.04	€0.028/hour
MongoDB Server	Cloud Infrastructure Services	MongoDB on Ubuntu 20.04	€0.028/hour
RavenDB Cloud	Hibernating Rhinos	Pay-As-You-Go	Price varies
AllegroGraph 7.3.0	Franz Inc.	AllegroGraph Free Edition	€0.253/hour
MongoDB® 4.4 on LINUX CentOS 8.2	Tidal Media Inc	MongoDB® 4.4 on LINUX CentOS 8.2	€0.03/hour
MongoDB Server on Linux 7.9	Art Group	MongoDB Server on Linux 7.9	€0.032/hour
MongoDB 6 on CentOS	AskforCloud LLC	MongoDB 6 on CentOS	€0.042/hour
NuOCR - OCR automation	Nuveto LLC	Gold	
MongoDB 6 on Red Hat Enterprise Linux 9	AskforCloud LLC	MongoDB 6 on Red Hat Enterprise Linux 9	€0.035/hour

Figura 3.2: Proposte di Azure per i database documentali

Azure Cosmos DB si distingue per la sua capacità di scalare orizzontalmente in maniera illimitata, consentendo di gestire volumi di dati e carichi di lavoro molto elevati, fino a milioni di richieste al secondo, grazie alla possibilità di distribuire il carico su più regioni Azure. È stato infatti ideato per presentare un’architettura distribuita, con replica au-

tomatica dei dati, assicurando un'elevatissima disponibilità e resilienza. Queste vengono assicurate anche in caso di interruzioni regionali, grazie a meccanismi di failover automatico. Inoltre, la distribuzione globale garantisce che i dati siano sempre vicini agli utenti, riducendo drasticamente la latenza a millisecondi (con SLA del 99.999% di disponibilità per account multi-regione). Consente l'indicizzazione attraverso più partizioni in maniera automatica e personalizzabile ottimizzando le query, riducendo la complessità e migliorando le prestazioni, senza richiedere oneri di gestione manuale degli indici.

Pur essendo focalizzato sui database documentali, Cosmos DB è però una soluzione multi-modello e multi-API. Supporta infatti, oltre alla sua API nativa per NoSQL (che usa il modello a documenti JSON), anche API compatibili con MongoDB, Apache Cassandra, Apache Gremlin (per i grafi) e Azure Table. Questa versatilità permette agli sviluppatori di utilizzare strumenti familiari, semplificando la migrazione di applicazioni esistenti o lo sviluppo di nuove con la flessibilità di scegliere il modello di dati più appropriato.

A livello di costi è difficile portare un'analisi precisa, in quanto tutti i competitor presentano servizi con capacità, disponibilità e integrazione differenti. I modelli di pagamento che utilizzano metriche di utilizzo diverse, rendendo necessarie ulteriori analisi che dipendono anche dall'effettiva tipologia e quantità delle richieste che vertono sul database. Di seguito viene riportata una tabella per comparazione i costi delle alternative principali. Cosmos usa come metrica le Request Units(RU) per quantificare l'impatto di una richiesta sul database. Le RU rappresentano un'astrazione delle risorse di sistema (CPU, I/O, memoria). Ogni operazione consuma RU, proporzionalmente alla sua complessità, dimensione e al carico computazionale richiesto.



Azure Cosmos

Servizio	Costo ogni milione di scritture (normalizzate a 1 KB)	Costo ogni milione di scritture (normalizzate a 1 KB)	Costo di manutenzione GB/mese
AWS DynamoDB	\$1.25	\$0.25	\$0.25
Google Cloud Firestore	\$0.90	\$0.30	\$0.156
Azure Cosmos DB	In base al consumo di RU, \$5.84 al mese ogni 100RU/s	In base al consumo di RU, \$5.84 al mese ogni 100RU/s	\$0.25 (Transazionale)

Tabella 3.2: Costi dei principali database documentali gestiti in Cloud

La difficoltà di stabilirne il costo in una fase iniziale è mitigata però dalla presenza di un piano gratuito perenne. Cosmos DB offre infatti una quota gratuita di risorse iniziali, per tutta la durata dell'utilizzo. Il piano prevede 25 GigaByte di memoria gratuita, a cui si aggiungono 1000 RU/s offerti per ogni categoria di operazione, suddivise in lettura, scrittura ed eliminazione. Dato lo stadio iniziale del progetto queste caratteristiche sono state considerate sufficienti, permettendo di sfruttare e testare le capacità di distribuzione. Nel caso in cui, sfruttando dati derivati dall'utilizzo effettivo dell'applicazione, un'analisi condotta durante fasi successive del progetto faccia emergere che Cosmos DB non rappresenti l'opzione più adeguata, lo spostamento dei dati verso un altro gestore comporterà uno sforzo limitato, data la compatibilità nella rappresentazione dei dati tra le diverse tecnologie di database documentali.

3.2.2 Configurazione di Cosmos DB

La creazione di una nuova istanza di Cosmos DB richiede la definizione delle impostazioni di funzionamento, che ne determinano le proprietà, a livello di disponibilità, ridondanza, sicurezza e resilienza ai guasti.

La prima impostazione riguarda la distribuzione geografica dei dati. Cosmos distribuisce sue risorse in zone, che vengono raggruppate in regioni. L'opzione di usare delle availability zones duplica i dati su più zone all'interno della stessa regione. Questo crea ridondanza dei dati per una maggiore resistenza ai guasti, e così facendo aumenta la disponibilità dei

dati, che da un disponibilità garantita iniziale per la zona singola del 99.99% (sulle scritture) sale al 99.995%, evitando la perdita dei dati e delle funzionalità in caso una zona non sia più raggiungibile. L'aggiunta di ulteriori regioni diminuirebbe il rischio indisponibilità a causa di guasti (i dati verrebbero duplicati anche tra le varie regioni), ma aumenterebbe la complessità e il costo dell'applicazione. Per una fase iniziale si è optato per garantire una ridondanza a livello di zone, selezionando quindi l'opzione delle availability zones, rimanendo però all'interno di una regione singola.

Basics

Subscription	Dipartimento di Informatica - Scienza e Ingegneria
Resource Group	SRS2024-Giacomo-Romanini-new
Location	Italy North
Account Name	(new) wyddocuments
API	Azure Cosmos DB for MongoDB
Capacity mode	Provisioned throughput
Geo-Redundancy	Disable
Multi-region Writes	Disable
Availability Zones	Enable

Figura 3.3: Impostazioni generali di Azure Cosmos DB

Volendo inizialmente rimanere in una singola regione, l'impostazione "Geo-Redundancy" non è stata abilitata. Allo stesso modo le scritture su più regioni non sono state abilitate, preferendo un approccio centralizzato.

Come modello di costo Azure propone due modalità distinte: Serverless e Provisioned throughput. Nella modalità serverless il servizio scala in base alle richieste, e considerando quindi solo l'effettivo utilizzo di risorse richieste. Adatto a carichi di richieste improvvisi e sbilanciati, non supporta l'esecuzione su più regioni. Il provisioned throughput invece richiede una previsione delle risorse necessarie, per metterle a disposizione e farsi pagare di conseguenza, che siano state usate o meno. Questo necessita quindi di una stima del carico previsto, per poi impostare le risorse. Azure propone però un metodo chiamato autoscale, che permette di modificare automaticamente (all'interno di un intervallo pre-

definito) le risorse messe a disposizione in base al carico del momento, per poi considerare il massimo valore raggiunto in quell'ora. Questo permette di evitare l'implementazione di una strategia, mantenendo comunque i costi legati al consumo effettivo delle risorse. Prevedendo un carico di richieste non troppo elevato, ma anche costante e poco variegato, e considerando un tempo di latenza minore e il supporto a regioni multiple, si è scelta una strategia di pagamento di tipologia provisioned throughput, che verrà poi impostata per scalare automaticamente in un intervallo di risorse inizialmente contenuto.

Per migliorare la sicurezza del database e garantire il controllo degli accessi il database è stato inserito all'interno di una rete virtuale privata, limitando l'accesso ai soli altri nodi che ne fanno parte, isolandolo verso l'esterno. Verranno quindi aggiunti alla rete solo i servizi che devono comunicare con Cosmos DB, in particolare Azure Functions e il KeyVault(che contiene le chiavi per la cifratura dei dati). Nonostante la ridondanza nelle zone garantisca il ritrovamento dei dati anche in caso di indisponibilità di un'istanza del database, si è scoperti qualora avvengano modifiche erronee o malintenzionate.

Il servizio di backup proposto da Azure permette di salvare lo stato dei dati fino a un certo momento nel passato, in maniera tale garantire il ritorno a una situazione precedente in caso ci si accorga siano avvenute modifiche non desiderate. Azure permette sia di personalizzare la tipologia di backup, in termini di durata, grandezza dei salvataggi e numero di copie, sia di seguire delle opzioni già configurate. Nel nostro caso si è scelto di applicare il servizio incluso gratuitamente che mantiene le modifiche avvenute negli ultimi sette giorni.

3.2.3 Definizione delle collezioni

Stabilita la tipologia e il comportamento del servizio che ospita il database, bisogna definire la struttura il cui le classi vengono salvate, in maniera da sfruttare al meglio le sue proprietà, allineando i dati alle procedure di lettura e scrittura, per ottimizzare il carico e il tempo di risposta.

Grazie alle proprietà di accoppiamento debole e al supporto alla denormalizzazione delle entità, la libertà di modellazione del dominio fornita da Cosmos DB è massima. Par-

tendo dalle entità principali, per ognuna di esse è stata associata una collezione. Sono state così definite quindi le entità di User, Profile, Event, Image e Group, che rispondono agli omonimi elementi del dominio. A Event e Profile si aggiungono ProfileDetails ed EventDetails, che contengono i dati particolari di questi elementi. Gli Account sono gestiti internamente da Firebase Authentication, e la loro relazione con gli utenti è stata mappata in un insieme all'interno di User tramite i loro Uid.

Allo stesso modo, laddove siano presenti relazioni uno a molti con cardinalità contenute, è stato possibile integrarle all'interno degli oggetti stessi, duplicando i collegamenti. Si evita così la necessità di cercare e recuperare i dati, rendendoli già direttamente disponibili. È il caso delle immagini, in cui viene salvato il riferimento dell'Event in Image, o di User e Profile, i quali contengono entrambi i corrispettivi riferimenti. UserRole è stato integrato solo in ProfileDetails, essendo necessario per controllare i permessi di un utente verso un profilo, e non il contrario.

La relazione tra profili e gruppi vede invece cardinalità elevate: non è infatti posto un limite al numero di gruppi a cui un profilo può fare parte. Non richiedendo però dettagli particolari o interazioni complicate (in questa fase del progetto), è stato considerato sufficiente mappare le relazioni tramite liste all'interno di Group e ProfileDetails.

Come analizzato nelle sezioni precedenti, le richieste del progetto vertono sulla relazione tra eventi e profili. In particolare, le richieste che sono previste più frequenti e quindi impattanti sono quelle di recupero dei dati generali delle entità su entrambi i versi della relazione (eventi relativi ai profili o profili collegati agli eventi), ma anche di conferma di un evento. Per prima cosa, vista la centralità delle richieste su queste entità, sono state separate le proprietà di dettaglio da quelle essenziali, salvandole in ProfileDetails ed EventDetails.

Oltre a migliorare il tempo di recupero delle informazioni che si ritiene necessario essere immediatamente disponibili, si evita così che le operazioni che vertono su dati secondari impattino sul flusso principale delle richieste. Riducendo la mole di dati da recuperare e analizzare, si rendono le query relazionali meno pesanti e più veloci.



Figura 3.4: Modello delle collezioni e il loro partizionamento

Prevedendo richieste di conferma sullo stesso evento molto vicine tra loro si è deciso di non integrare la relazione nelle entità esistenti, bensì di esternarla in documenti dedicati, per evitare modifiche concorrenti sullo stesso Event. Si introduce così ProfileEvent, che presenta tutte le proprietà della relazione tra i due elementi. Venendo partizionato in base all'hash del profilo corrispondente, assolve il compito di recuperare gli eventi associati al profilo. Include quindi il collegamento all'evento e la conferma o meno della partecipazione del profilo all'evento, ma anche la data dell'ultimo aggiornamento del profilo associato, in maniera tale da consentire di controllare direttamente se un evento è stato modificato rispetto all'ultimo aggiornamento, e deve quindi essere caricato.

Si prevede però che la quantità di richieste sia comunque importante anche dal lato opposto della relazione, ovvero per ottenere i profili correlati agli eventi. Non è accettabile una ricerca che attraversi le partizioni per recuperare queste informazioni. Per questo motivo viene creato EventProfile, che, partizionato assieme agli eventi, mantiene le informazioni minime necessarie per mostrare i profili collegati, in maniera da limitare la quantità di dati da mantenere consistente.

La distribuzione dei dati supporta così la scalabilità totale del sistema. Ogni richiesta di dati infatti non incrocia mai più di due elementi, e rimane in ogni caso limitata alla sua partizione. Lo stesso può essere affermato per le richieste di modifica, le quali vertono su un unico elemento principale considerato come fonte di verità, in base al quale le eventuali copie dovranno essere aggiornate. Se è vero che le richieste sono state rese rapide sia in lettura che in scrittura, questo è reso possibile grazie alla denormalizzazione, che deve però prevedere un procedimento per l'allineamento dei dati copiati.

3.2.4 Integrazione con le Azure Functions nel framework .Net

L'utilizzo di un database documentale comporta un sviluppi implementativi specifici. Il polimorfismo dei documenti e la denormalizzazione delle entità rendono la rappresentazione logica degli elementi e la loro relazione reciproca di secondaria importanza. L'approccio a oggetti del Framework .Net assume quindi minore rilevanza, in quanto si rende meno

necessaria la capacità di descrivere e astrarre le entità logiche attraverso classi di codice. Si distinguono però due tipologie di richieste principali, che determinano l'approccio apportato verso i dati: la recupera di un elemento e la sua modifica.

Quando è necessario recuperare un elemento dal database risulta comodo convertirlo in un oggetto. Questo consente di gestire le informazioni attraverso un'astrazione logica, con tutti i vantaggi relativi. La sua utilità risalta soprattutto nel caso in cui la lettura dell'elemento non ha il solo fine di essere restituito al richiedente, ma deve venire analizzato per procedere con la logica applicativa. La rappresentazione dei documenti in oggetti comporta infatti un codice più pulito e ordinato, interagendo con le proprietà e i metodi della classe, senza entrare nella complessità computazionale dell'elaborazione del documento. La conversione viene svolta automaticamente dal framework, e permette di associare ai dati il codice strettamente relativo, creando metodi appositi.

Ogni elemento del dominio avrà quindi una classe associata, con tutti i campi previsti dal modello. Il contesto logico così introdotto agevola l'interazione tra i servizi e gli oggetti, riducendo la probabilità di errori e semplificandone l'individuazione. Anche il testing viene favorito, dando la possibilità di analizzare il codice senza la necessità di interrogare il database o di simularlo. Permette inoltre di automatizzare la conversione dei dati verso i DataTransferObject(DTO), necessari per uniformare le risposte verso i client, che contribuiscono a separare la logica di business da quella che gestisce la comunicazione.

Nel caso la richiesta richieda invece la modifica di un elemento la conversione del documento in oggetto risulta poco efficiente. La creazione di un nuovo oggetto comporta infatti l'intera lettura dell'elemento associato. Questo può essere utile se è necessario mantenere un riferimento logico, se si useranno le informazioni che contiene o se bisognerà restituire al mittente l'oggetto aggiornato. Nella maggior parte dei casi invece si vuole semplicemente applicare una modifica, lasciando che sia il sistema a propagarla. In tal caso, Cosmos DB permette le cosiddette patch update, ovvero modifiche limitate al campo interessato, senza richiedere la lettura e la sovrascrittura dell'intero documento. Per queste occasioni vengono implementati appositi metodi che, preso in ingresso i dati da modificare, interagiscono con il database per aggiornare le sole parti coinvolte.

La comunicazione con il database viene astratta tramite un servizio dedicato chiamato WydDbService. WydDbService ha il compito di gestire le richieste e le sessioni con il database, nascondendo le complessità implementative e di interazione. In particolare, la connessione con il database comporta la richiesta verso Azure Key Vault per il recupero delle chiavi di accesso, per poi usarle per stabilire un canale attraverso il quale applicare le modifiche. Mette inoltre a disposizione un’interfaccia per nascondere tutte le logiche di basso livello dovute dall’interazione specifica di CosmosDB. Verrà utilizzato dagli altri servizi tramite dependency injection, automatizzandone la creazione e l’utilizzo, e di conseguenza la connessione e le richieste al database.

3.2.5 Garantire la consistenza eventuale dei dati

Avendo salvato le entità del dominio sotto forma di documenti denormalizzati la modifica del campo di un elemento potrebbe dover comportare la necessità di propagare gli aggiornamenti su tutti gli altri componenti in cui tale dato è duplicato.

Potendo accettare un ritardo nell’allineamento dei dati, è possibile separare la modifica del documento principale dalla sua distribuzione sul resto del database. Affidare il compito di aggiornare tutti i componenti secondari coinvolti alla stessa funzione risulta svantaggioso sotto molti aspetti. Si introdurrebbe infatti un ritardo nell’esecuzione, dovuto all’attesa dell’applicazione delle ulteriori modifiche, che devono essere controllate e gestite in caso falliscano. Complicare la funzione va inoltre contro i principi stessi dei servizi serverless, che prevedono invece strutture semplici con responsabilità precise e limitate. La loro efficienza deriva infatti dalla possibilità di eseguire in maniera autonoma compiti specifici. L’unico requisito necessario per permettere a una funzione di delegare ad altre le modifiche derivate è la garanzia che la loro esecuzione venga controllata e che quindi il loro successo sia assicurato.

L’introduzione di una funzione con il ruolo di orchestratore non sposterebbe di molto il problema. Nonostante soddisfi la necessità di affidare a diverse funzioni le diverse parti del problema e sia in grado di controllarne il risultato, potendone così garantire il succes-

so, implica comunque l’attesa del loro completamento. Aggiungerebbe inoltre un ritardo dovuto al tempo necessario per la sua stessa esecuzione, che aspetta e controlla le funzioni che ha chiamato. Questo ritardo è negativamente influenzato dall’accoppiamento debole tra l’orchestratore e le funzioni figlie, che non assicura l’immediata ripresa del padre al termine dell’esecuzione di una funzione da lui chiamata. Le dipendenze che verrebbero così introdotte tra l’orchestratore, la funzione originale e quelle necessarie per la consistenza dei dati, comporterebbero solo rischi di fallimento e l’allungamento dei tempi di risposta.

A questo scopo, Azure Cosmos DB mette a disposizione uno strumento apposito, chiamato Change Feed. Change Feed è un meccanismo tramite il quale è possibile associare le modifiche di un elemento del database all’esecuzione di una funzione. Il Change Feed viene definito come trigger all’interno di una funzione, e viene associato a una collezione di documenti. Ogni volta che un documento di questa collezione subisce un aggiornamento si crea in automatico un log che, all’interno della partizione coinvolta, viene salvato in una nuova collezione dedicata chiamata "lease". Il Change Feed è dunque il meccanismo che, nel momento in cui un log viene aggiunto alla sua lease, fa partire autonomamente la funzione associata.

Una volta invocata, la funzione legge dal lease tutti i log che non sono ancora stati processati per poi propagare le modifiche ove necessario. L’avvenuto successo dell’elaborazione del log viene registrato grazie a dei token di aggiornamento, che permettono di sapere in quale punto del lease sono presenti i log che non sono ancora stati processati. La persistenza dei log sui lease garantisce che la modifica sia presa in carico da una funzione almeno una volta, mentre la presenza dei token di aggiornamento assicura che la funzione venga eseguita di nuovo in seguito a un errore o a un guasto del sistema, soddisfando così i requisiti di invocazione. Questo meccanismo è intrinsecamente scalabile. Essendo infatti i lease associati alle partizioni in cui risiedono i documenti, è possibile dividere (e quindi distribuire, quando necessario) il carico su più funzioni.

Il Change Feed viene usato, ad esempio, nel caso in cui venga cambiata un’informazione importante di un evento. Questa operazione comporta l’aggiornamento di UpdatedDate,

campo che memorizza il momento in cui è stata apportata l’ultima modifica. UpdatedDate è stata denormalizzata all’interno dei ProfileEvent, e la sua modifica deve essere quindi propagata. La propagazione di questo aggiornamento sarà delegata a una seconda funzione. L’invocazione di questa funzione verrà associata al lease relativo alla collezione degli Event, nel caso siano presenti log non ancora elaborati. In questa maniera i compiti vengono efficacemente suddivisi in due funzioni, indipendenti tra loro. La prima funzione avrà il solo compito di cambiare la proprietà dell’oggetto, informando il client riguardo al successo dell’operazione. Una seconda funzione, attivata in autonomia dal Change Feed, si occuperà poi di aggiornare tutti i ProfileDetails relativi, senza andare a impattare sulle prestazioni o sulle dipendenze della prima.

Ci sono situazioni in cui il Change Feed non può essere usato. È ad esempio il caso della conferma di un evento. L’elemento primario coinvolto in questa operazione è ProfileEvent, che sarà modificato dalla funzione iniziale. La conferma di partecipazione a un evento da parte di un profilo viene considerato come modifica all’evento stesso, che comporta quindi l’aggiornamento del documento Event. Se però si associasse una funzione anche per le modifiche a ProfileEvent tramite Change Feed, si verrebbe a creare una ricorsione infinita per il quale la modifica a un ProfileEvent comporta la modifica a un Event, che a sua volta comporta una modifica a ProfileEvent, e via dicendo. Per quanto sia possibile aggiungere all’interno delle funzioni controlli che analizzino ogni volta l’origine della richiesta, questo porterebbe a una maggiore complessità e a un aumento del carico implementativo, introducendo nuovi dati all’interno di ogni aggiornamento. Si è quindi deciso di utilizzare il ChangeFeed solo per le entità la cui modifica influenza molteplici altri documenti, delegando in maniera differente l’eventuale aggiornamento di un altro elemento singolo.

Ci sono diversi modi che le Azure Functions possono sfruttare per invocare un’altra funzione, senza passare per l’utilizzo di un orchestratore.

Una prima soluzione può essere l’invio di una nuova richiesta HTTP al server stesso, al quale è associata la funzione voluta. Per quanto risulti veloce e facile da implementare, questa soluzione non garantisce l’effettiva presa in carico, esecuzione e completamento

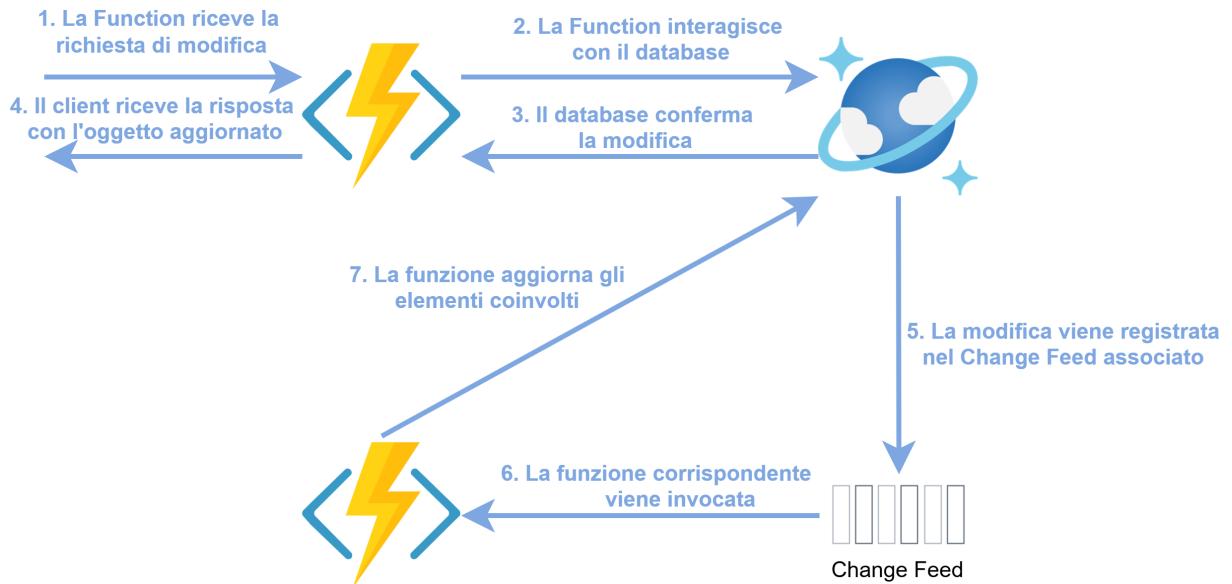


Figura 3.5: Aggiornamento dei documenti tramite Change Feed

della richiesta, a meno di controllarne la risposta. Il controllo della risposta comporta però l’attesa della richiesta, procedimento che mina la finalità stessa dell’operazione.

Azure mette invece a disposizione servizi basati su code ed eventi proprio per mettere in comunicazione diverse parti di uno stesso sistema. Tra le funzionalità offerte da Azure e quindi direttamente integrabili con le Azure Functions troviamo Azure Queue Storage, Azure Service Bus e Azure Event Hub.

Azure Queue Storage è un servizio di code di messaggi offerto come parte di Azure Storage, che si occupa del salvataggio di informazioni che non rientrano nei casi d’uso dei database tradizionali. Le funzioni hanno quindi la possibilità di aggiungere alla coda i propri messaggi, che verranno poi elaborati da un’altra funzione grazie a un trigger associato. La dimensione massima che i messaggi possono avere è di 64 KB, e rimangono in memoria per un tempo prestabilito, che può essere configurato. Le code sono persistenti e altamente disponibili, assicurando che i messaggi non vadano persi in caso di fallimento del consumer, ma non presentano ulteriori funzionalità associate.

La sua semantica di consegna garantisce che il messaggio venga elaborato almeno una volta, il che significa che la sua consegna potrebbe avvenire più volte, in caso di errori o di timeout di elaborazione da parte della prima funzione che lo ha preso in carico. Pre-

senta quindi un servizio semplice e veloce, estremamente robusto ma anche scalabile. La sua semplicità, sebbene contribuisca a ridurne il costo, ne comporta però la mancanza di ulteriori proprietà avanzate. Ad esempio, Azure Queue Storage non presenta la capacità di gestire i messaggi in caso la loro esecuzione continui a fallire, che può essere desiderata in caso sia necessario assicurare che l'operazione giunga al suo termine.

Azure Service Bus è invece un servizio di messaggistica che supporta scenari più complessi rispetto a Queue Storage. Offre due tipologie principali di comunicazione: code e argomenti. Le code di Service Bus supportano le stesse funzionalità del servizio precedente, a cui però se ne aggiungono di più avanzate quali l'ordinamento, le sessioni per il raggruppamento di messaggi correlati, la funzionalità di "dead-letter queue" integrata per la gestione automatica dei messaggi non elaborabili e l'elaborazione transazionale.

Gli argomenti(topic) prevedono un pattern publish/subscribe, nel quale più fruitori possono collegarsi allo stesso argomento per poi venire tutti notificati in contemporanea in caso venga inviato un messaggio a esso inerente. Essendo stato progettato per soddisfare carichi di lavoro aziendale, oltre a fornire garanzie di consegna più robuste e integrare meccanismi di gestione degli errori, garantisce un'elevata scalabilità. Risulta però più costoso e relativamente più complesso da utilizzare.

Azure Event Hub è un servizio di ingestione di dati altamente scalabile e a bassa latenza, progettato per lo streaming di grandi volumi di eventi da diverse fonti. Non è una coda di messaggi nel senso tradizionale, ma piuttosto un "broker di eventi", in cui gli eventi vengono aggiunti a una coda distribuita su più partizioni dove rimangono disponibili per i consumer per un periodo configurabile (fino a 90 giorni). I consumer leggono gli eventi dalla partizione mantenendo ognuno il proprio progresso, il che permette a più consumer group di leggere gli stessi eventi indipendentemente. Anche in questo caso, la semantica garantisce che il messaggio venga elaborato almeno una volta.

Presenta quindi elevate caratteristiche di scalabilità, fornendo una bassa latenza, un throughput estremamente elevato e la possibilità di parallelizzare il consumo dei messaggi. Pur mantenendo in memoria i dati, e quindi assicurando che sia sempre possibile applicare

le modifiche, tutta la logica necessaria per controllare l'esecuzione, ritentare ed eliminare il messaggio rimane però a carico dello sviluppatore.

Caratteristica	Queue Storage	Service Bus	Event Hub
Tipo di servizio	Coda di messaggi semplice	Coda/Broker di messaggi	Flusso di eventi
Modalità di fruizione	Competing Customers (Point-to-point)	Competing Customers (Point-to-point) Publish/Subscribe (argomenti)	Publish/ Subscribe (molti a molti)
Caratteristiche	Semplice, scalabile e robusto ma nessun controllo degli errori	Affidabile, scalabile, garantisce il controllo dell'esecuzione, ordina i messaggi	Scalabile, robusto, ordinato ma nessun controllo sull'esecuzione
Costo	€0,04 GB/mese + €0,0004 ogni diecimila operazioni	€0,044 ogni milione di operazioni (piano Basic con solo le code)	€0,014/ora per ogni Unità di throughput + €0,025 ogni milione di eventi

Tabella 3.3: Proprietà dei servizi Azure per la propagazione interna di messaggi

Viste queste considerazioni, per garantire la consistenza degli oggetti nei casi in cui non sia possibile sfruttare il Change Feed si è scelto di adottare Azure Service Bus. La scelta deriva principalmente dalla suo supporto nativo al controllo dell'esecuzione delle funzioni, grazie all'utilizzo di una dead-letter queue. La dead-letter queue è un contenitore in cui vengono inseriti tutti i messaggi la cui elaborazione è stata tentata e fallita una determinata quantità di volte. Garantisce quindi, in aggiunta a ulteriori tentativi in caso di insuccesso, il monitoraggio di tutte le operazioni che non riescono a essere portate a termine. Questo permette di assicurare che gli aggiornamenti necessari per allineare i documenti siano sempre controllati.



Service Bus

Tornando all'esempio precedente, nel quale era necessario propagare la conferma di un evento, la modifica di Event e di EventProfile verrà eseguita da due funzioni dedicate. Al termine della modifica di ProfileEvent, la funzione inserirà in coda al Service Bus un messaggio con le informazioni necessarie alle altre due, per poi inviare la risposta al client. Le due funzioni verranno quindi invocate, aggiornando gli elementi coinvolti. Nel caso di Event, il Change Feed associato aggiornerà di conseguenza tutti i suoi ProfileEvent, senza però creare ricorsioni.

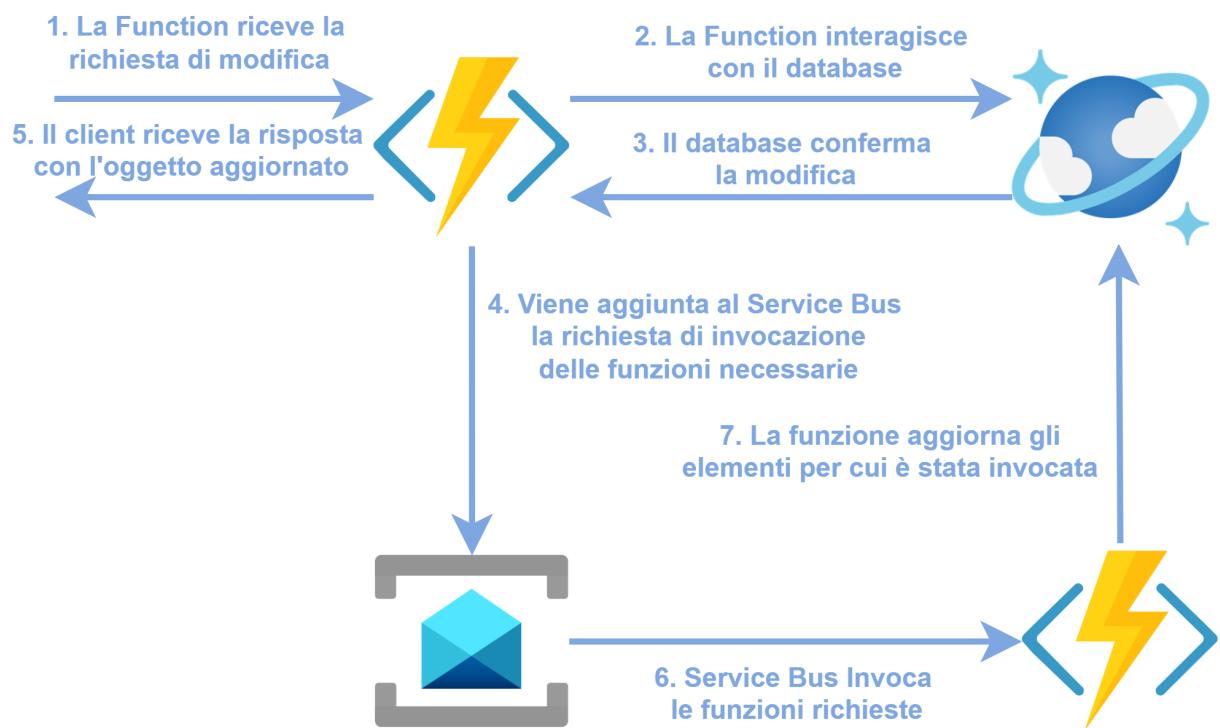


Figura 3.6: Fasi di aggiornamento tramite Service Bus

3.3 Implementazione della memoria locale

Per poter visualizzare un elemento sul dispositivo utente è necessario resuperare le informazioni relative. Attualmente questa funzionalità è realizzata effettuando una chiamata al server ogni volta che viene richiesto un dato. Chiedere, ottenere ed elaborare dati dal server richiede però tempo e risorse, causando ritardi che impattano sulle prestazioni e sull’esperienza utente. Inoltre, questo rende l’utilizzo l’applicazione strettamente dipendente dalla sua connessione al server, che diventa così inutilizzabile in situazioni in cui il dispositivo non abbia modo di comunicare.

È quindi utile salvare copie degli elementi usati più frequentemente nella memoria locale del dispositivo, nel caso in cui la tecnologia del dispositivo lo permetta. In questa maniera, nel momento in cui un’interazione richieda la visualizzazione dei dati, si possono fornire immediatamente le informazioni disponibili in copia, senza la necessità di effettuare un’ulteriore richiesta. Questo permette la diminuzione del carico del server, una maggiore reattività del prodotto e una migliore esperienza utente.

La creazione e la persistenza delle copie deve essere gestita in maniera trasparente rispetto alla logica grafica, creando nell’applicazione utente un livello intermedio tra la visualizzazione delle informazioni e la comunicazione con il server. Questo livello sarà incaricato di gestire la persistenza e la modalità di recupero dei dati, fornendoli quando richiesti, nascondendo la complessità derivata dalla necessità di allineare le due memorie.

Inoltre, data la natura condivisa dell’applicazione risulta fondamentale che l’aggiornamento avvenga quanto più possibile in tempo reale rispetto modifiche applicate: oltre a essere una prerogativa di tutte le applicazioni moderne, ne viene influenzata anche la user experience. Lo spostamento di un appuntamento, la conferma di una presenza o la modifica del luogo di appuntamento sono elementi critici per i quali gli utenti devono venire informati il prima possibile.

Tutto questo comporta sfide progettuali che permettano di garantire allo stesso tempo sia la correttezza dei dati presenti che il loro allineamento, all’interno del minor ritardo possibile.

3.3.1 La realizzazione della cache

Per motivi di prestazioni, Flutter mantiene nativamente lo stato di un componente solo per il tempo strettamente necessario per la sua fruizione. Normalmente il tempo di vita del suo stato coincide quindi con quello del componente a cui è associato. Se si corre lasse questa logica con il mantenimento dei dati, la persistenza locale degli elementi e dei dati avrebbe durata limitata, comportando la richiesta delle informazioni al server ogni volta che si vuole visualizzare un elemento. Si rende perciò necessaria la creazione e la gestione di una memoria locale indipendente dalle logiche grafiche, che permetta di mantenere i dati ricevuti dal server anche in seguito al termine del servizio dei componenti.

Gli elementi mantenuti in cache devono essere unici e disponibili in tutto il programma. I dati verranno salvati in collezioni di oggetti corrispondenti alle classi logiche del dominio, la cui gestione universale sarà affidata a servizi dedicati. L'esecuzione dei servizi dovrà essere indipendente dall'interfaccia, fornendo l'accesso agli elementi delle collezioni, ma gestendo anche le loro modifiche. In caso alcuni elementi subiscano dei cambiamenti (dal dispositivo stesso o tramite notifica), sarà inoltre compito loro aggiornare i componenti coinvolti. Per venire incontro a queste necessità, Flutter mette a disposizione dei componenti di tipologia provider, ai quali i vari componenti grafici possono essere associati.

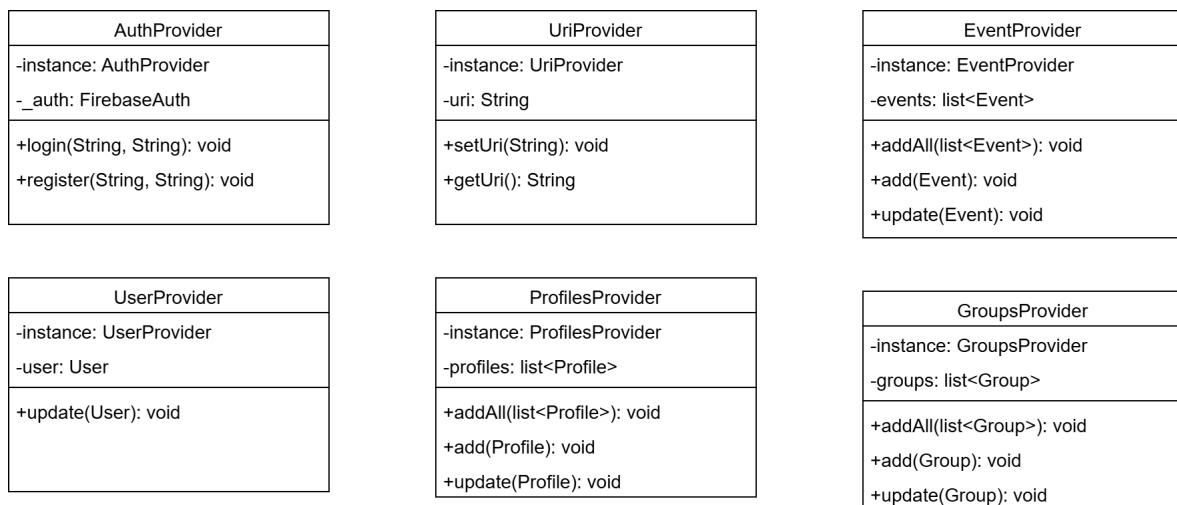


Figura 3.7: Classi provider all'interno dell'applicazione

Grazie all'associazione che si viene così a creare tra componenti grafici e provider, questi

ultimi hanno la possibilità di ricaricare attivamente parti del progetto, aggiornandole nel momento in cui avviene una modifica che li coinvolge. Questo genera una pulizia generale del codice, integrando e nascondendo tutta la logica di collegamento e distribuzione degli eventi. Per ogni entità principale del dominio è stata associata una collezione, che viene gestita da un provider apposito. Nel momento in cui un componente deve visualizzare un elemento, dichiara la sua dipendenza con il servizio provider relativo, a cui farà richiesta per i dati. Nel caso in cui i dati siano già presenti in memoria verranno restituiti subito. Altrimenti la grafica li sostituirà temporaneamente con elementi neutri, in attesa della risposta. All’arrivo della risposta il provider notifica l’elemento grafico, che si aggiorna di conseguenza. L’unicità delle collezioni e dei rapporti con il server è ottenuta grazie all’implementazioni dei provider come classi singleton, pattern di sviluppo che garantisce la cardinalità di una classe in al massimo un’istanza per ogni momento.

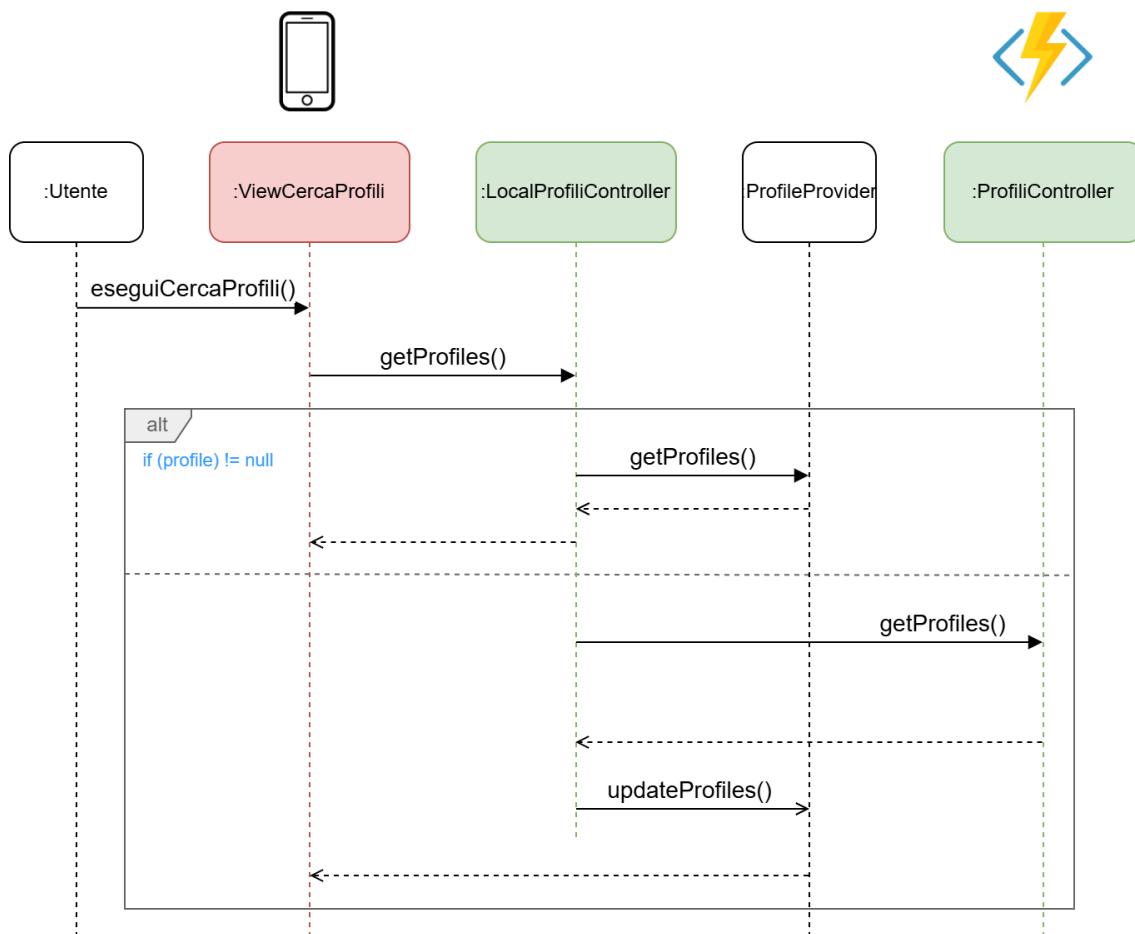


Figura 3.8: Esempio di interazione logica tra i componenti del client

I vantaggi dell’implementazione di una memoria locale derivano dalla sua capacità di rimanere aggiornata con il server principale. Risulta infatti inutile facilitare l’accesso a dei dati se questi, una volta forniti, risultano scorretti. Per essere sicuri che la cache possa quindi essere considerata valida in ogni momento è necessario implementare strategie che, all’interno di un intervallo di tolleranza, garantiscano l’allineamento con i dati presenti sul server. La principale fonte di aggiornamenti viene fornita direttamente dal server che, attraverso un processo di comunicazioni in tempo reale, invia delle notifiche ai client ogni qual volta vengano interessati da una modifica.

3.3.2 Scelta della tecnologia per la comunicazione in tempo reale

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 a 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 abbia subito 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 più adatto per permettere una comunicazione continua tra le 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.

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 le loro modalità di utilizzo.

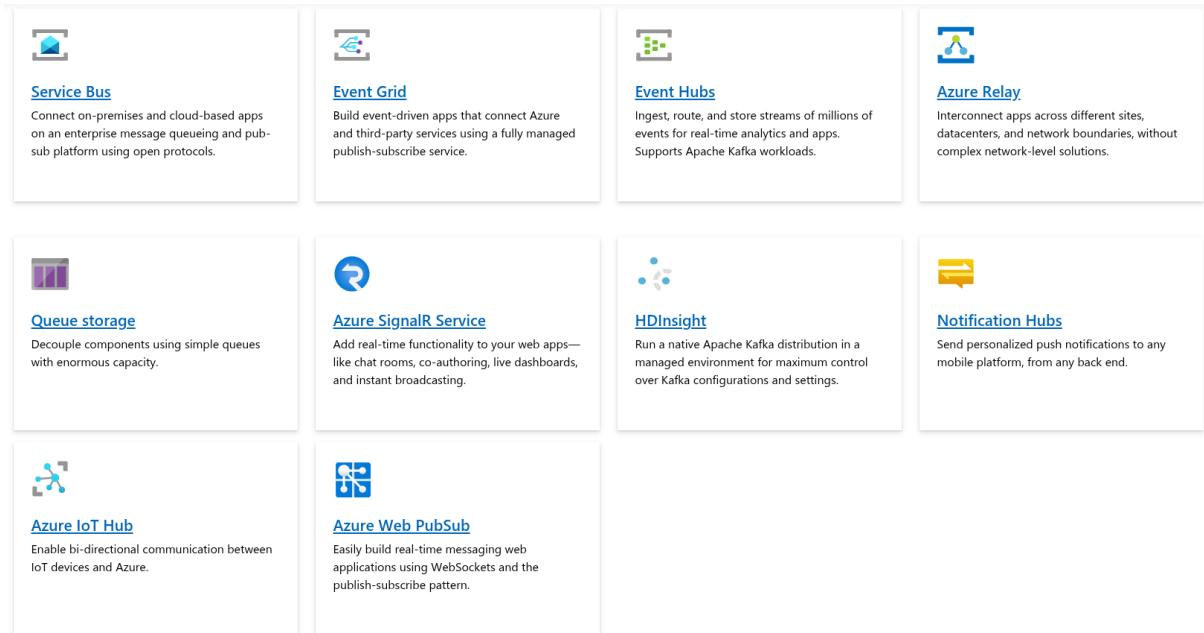


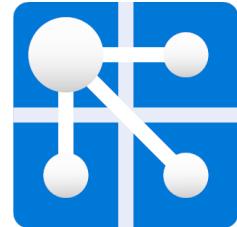
Figura 3.9: I servizi di comunicazione istantanea proprietari di Azure

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 il supporto alla persistenza, inviando direttamente i messaggi senza mantenerli in memoria, ma gestisce la scalabilità del sistema.



Azure PubSub

3.3.3 Invio delle notifiche

L’integrazione di Azure Web Pub Sub deve avvenire sia a livello server che con i device degli utenti. Seguendo il modello publish subscribe, ogni client creerà una connessione con AWPS in sola lettura, ricevendo tutti i dati che lo interessano. Il server avrà il compito di interfacciarsi con il servizio per pubblicare i dati sui canali. Un canale è

un contenitore logico che rappresenta un argomento a cui un utente può essere interessato. Un dispositivo può essere associato a più canali, pur mantenendo la stessa connessione.

Le modifiche verranno inviate ai canali, che le propagheranno a loro volta a tutti i dispositivi collegati. La scelta della definizione dell'elemento in base al quale il canale viene creato deriva da un'ulteriore analisi del dominio. Il soggetto interessato alle modifiche sottoposte a notifica è il profilo. Dunque verranno creati i canali relativamente ai profili. Un dispositivo descrive però l'interazione di un utente. Per questo motivo, una volta creata la connessione con il servizio, il dispositivo si iscrive ai canali dei profili associati al suo utente.

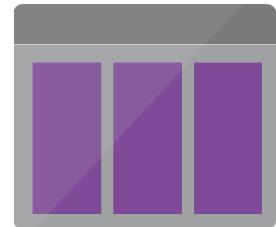
Se però si creassero i canali in relazione ai profili ogni dispositivo (che riassume l'interazione di un utente) dovrebbe mantenere una connessione per ogni profilo collegato all'utente. La creazione di un canale per ogni device allo stesso modo risulta estremamente inefficiente, in quanto, oltre a introdurre 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 posseggano effettivamente l'accesso adeguato.

A seguito di una richiesta che comporta una modifica che necessita di essere propagata 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, se non quelle tra gli utenti e i profili definite implicitamente dai canali. Per questo motivo, è molto probabile che l'operazione di notifica richieda una richiesta al database, per recuperare i profili coinvolti. Ad esempio, la modifica di un evento comporta la notifica a tutti i profili relativi, e quindi sarà necessario recuperare tutti i profili associati a quell'evento.

Vista la responsabilità precisa e considerato che l'effettivo invio delle notifiche non è un'operazione essenziale per il successo di una richiesta, si delega questo compito a un'altra funzione. Questa funzione avrà quindi il compito di recuperare i profili coinvolti per mettersi poi in contatto con AWPS e per pubblicare le notifiche sui relativi canali. A

differenza delle funzioni chiamate per garantire la consistenza della persistenza principale, dove era essenziale garantire il controllo e il successo dell'operazione, in questo caso si preferisce ottenere uno svolgimento veloce senza che questo richieda tutto il carico aggiuntivo derivato dal controllo del risultato, considerando accettabile la sua eventuale perdita o fallimento.

Queste considerazioni hanno portato alla scelta di Azure Storage Queue come strumento di delega per le operazioni di notifica. Offre infatti un servizio veloce ed economico, sebbene non fornisca nativamente un meccanismo di controllo che garantisca il successo dell'operazione legata al messaggio. Al termine della sua esecuzione la funzione principale aggiungerà quindi a una coda dedicata un messaggio con le informazioni relative alla modifica da lei apportata. Una seconda funzione, la cui attivazione è collegata all'aggiunta di oggetti in quella coda, leggerà il messaggio, per poi eseguire il recupero dei profili e il collegamento con AWPS.



Azure Storage Queue

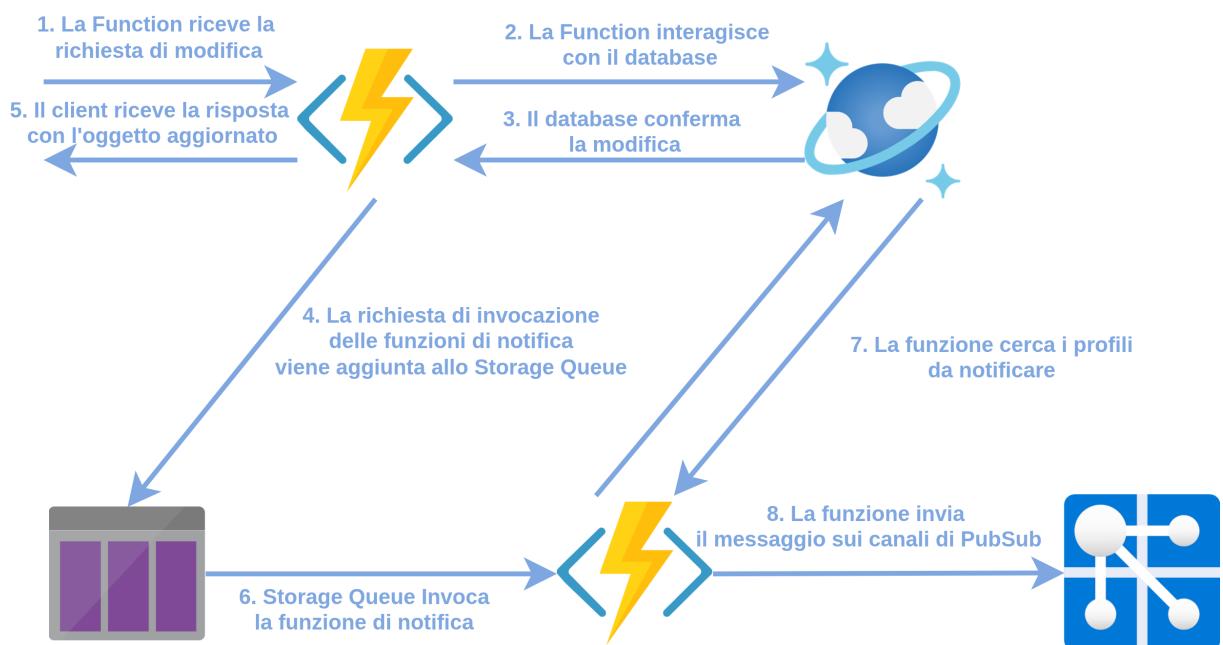


Figura 3.10: Interazione delle Azure Functions con la Queue

Nell’ottica di rendere il processo di invio delle notifiche il più veloce ed efficiente possibile, si è deciso di includere nel messaggio le informazioni minime sull’accaduto. Questo permette di uniformare il formato delle notifiche, semplificando la loro gestione e velocizzando l’invio, diminuendo il volume dei dati trasmessi tramite WebSocket. Alla ricezione della notifica il client apporterà, se possibile, gli aggiustamenti dovuti. In caso di modifiche importanti sarà però sua responsabilità recuperare i dati aggiornati direttamente dalla memoria centrale. Questa strategia risulta particolarmente efficace soprattutto in caso di numerose modifiche ravvicinate sullo stesso elemento, che possono essere raggruppate per poi eseguire la richiesta di aggiornamento una sola volta.

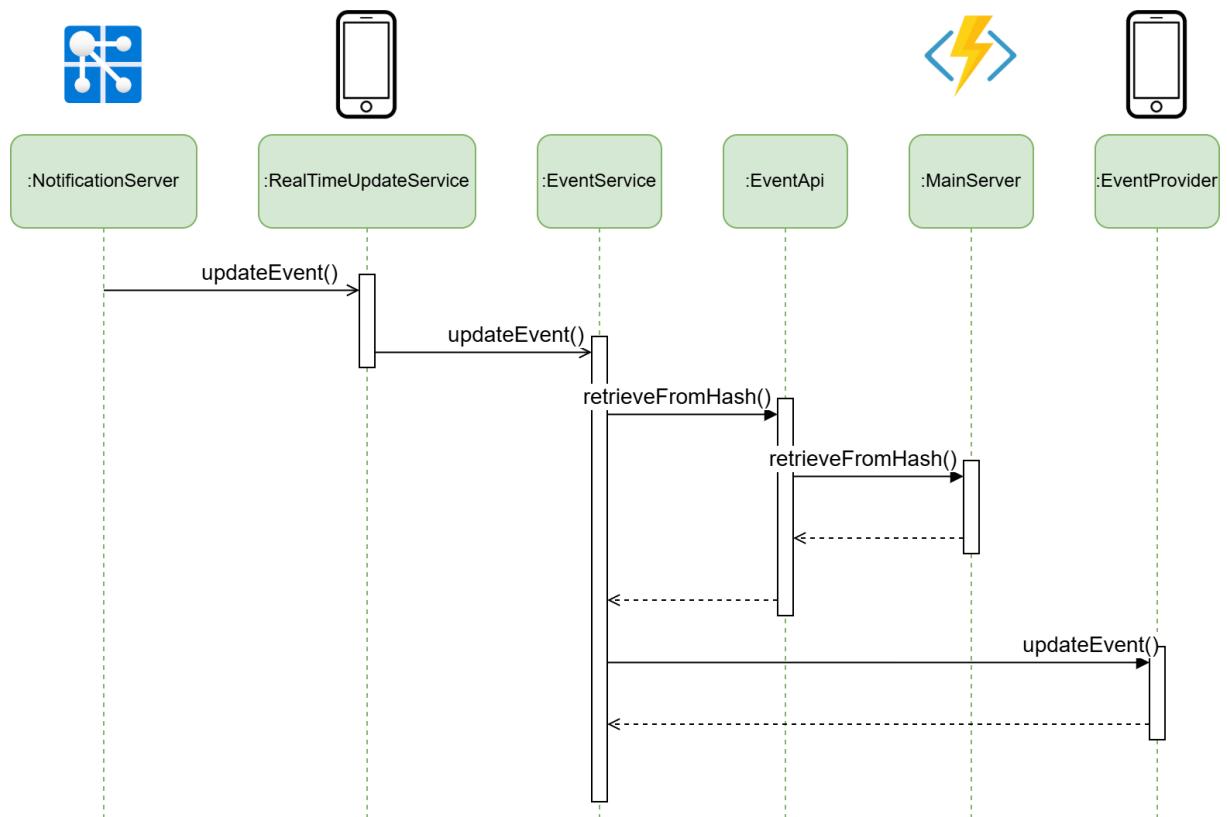


Figura 3.11: Interazione tra AWPS e un client per il recupero di un Event

Per poter assicurare però che nessuna notifica sia andata persa si necessita l’implementazione di un processo che si confronti con il server per recuperare le modifiche non ricevute.

3.3.4 Garantire l'allineamento della cache

Il client ha la responsabilità di tenere allineata la propria cache ai valori della memoria centrale. La presenza di dati aggiornati in cache permette infatti, oltre a garantire tempi di reazione minimi, di evitare di ricorrere a richiedere dati al server per ogni interazione utente. La validità delle risorse che ha salvato in cache, nonostante la ricezione di notifiche, dipende dalla certezza che corrispondano integralmente ai dati ufficiali.

La ricezione degli aggiornamenti tramite WebSocket e la loro dipendenza dalle Azure Storage Queue non garantiscono la loro consegna. Come discusso nella precedente sezione infatti, non sono previsti ulteriori tentativi di esecuzione nel caso in cui la funzione incaricata dell'invio fallisca. Inoltre, la tecnologia WebSocket, pur essendo veloce ed efficiente, può subire interruzioni o perdite dimostrandosi non completamente affidabile. Per poter quindi considerare allineata la cache alla memoria centrale, si necessita di un processo che assicuri l'aggiornamento dei dati in memoria.

A ogni elemento viene associato il momento relativo al suo ultimo aggiornamento noto. Si tiene inoltre memoria dell'ultimo momento in cui è stata attivamente effettuata una richiesta esplicita di aggiornamento. A 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. Alla ricezione della risposta si aggiorna il momento dell'ultima richiesta.

Da quel momento in poi, periodicamente, verrà inviata una richiesta contenente sia gli identificativi che il momento dell'ultimo aggiornamento degli elementi le cui modifiche sono giunte al dispositivo (grazie alle notifiche) durante l'ultimo intervallo. Il server, ricevendo queste informazioni, controllerà se combaciano con tutte le entità associate al profilo della richiesta. In caso trovi incongruenze, ovvero emergano elementi non presenti tra quelli ricevuti o con data di modifica successiva a quella dichiarata, restituirà sudetti elementi con gli ultimi dati aggiornati.

Capitolo 5

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 il sistema sia per gli utenti stessi. 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 Modalità di recupero delle immagini

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. Per questo motivo, al primo avvio dell'applicazione, 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.

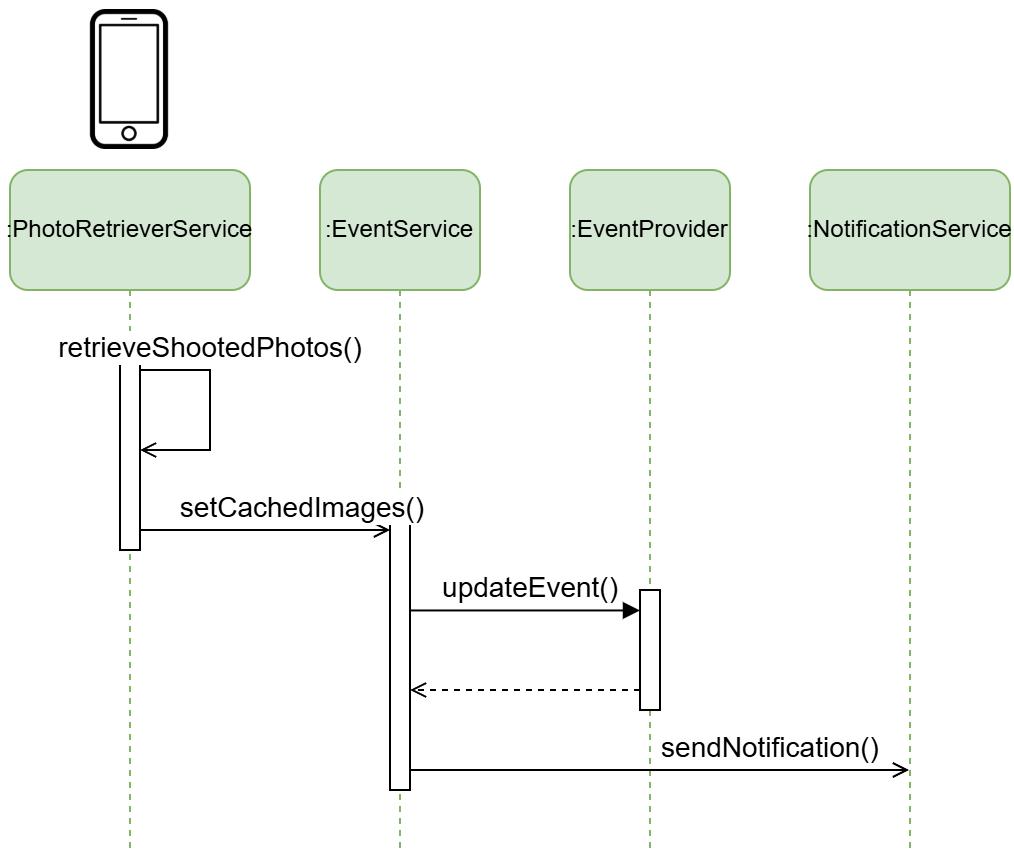


Figura 4.1: Interazione tra i componenti per il recupero delle immagini

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 devono essere salvate sul server, per essere rese disponibili anche agli altri profili con cui l’evento è condiviso.

4.2 Integrazione con il sistema

A differenza dei dati usualmente scambiati all'interno del sistema, i file multimediali presentano dimensioni significativamente superiori, con una differenza che si manifesta su ordini di grandezza rilevanti. Salvare questi file nella stessa modalità degli altri dati, affiancandoli quindi agli elementi logici, comporterebbe un impatto significativo sulla dimensione totale del database, andando a influenzare tutte le operazioni eseguite su di esso, che dovrebbero recuperare informazioni su un volume di dati molto più vasto del necessario. Il rallentamento generale delle operazioni comporterebbe 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. Infatti la possibilità di allegare più file a un singolo evento implica che i tempi di caricamento 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 Scelta del servizio di 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 è deciso di distinguere il dato bi-

nario dalla sua rappresentazione logica. In questo modo, la relazione tra il file e l’evento associato può essere mantenuta indipendentemente dai dati binari che lo compongono. Una volta recuperati i riferimenti logici ai file multimediali associati all’evento interessato, sarà possibile ottenere in un secondo momento 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 (Image).

Considerando la necessità e la possibilità di archiviare i file multimediali su risorse differenti dal database centrale, è fondamentale individuare la soluzione più adatta per la 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 a 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. Oltretutto, i costi sono elevati, particolarmente nel caso di grandi moli di dati.

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 dipendenza 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. I container creati sono relativi alle categorie per le quali le immagini vengono salvate. Ad esempio, ci sarà un container relativo alle immagini degli eventi, così come uno per quelle legate ai profili.



Azure Blob Storage

All'interno del nome del blob si possono però aggiungere percorsi logici, separati dal carattere "\\". Le immagini vengono quindi identificate univocamente attraverso la combinazione del container, dell'hash dell'elemento a cui fanno riferimento (l'evento, il profilo...) e del loro hash, concatenati per formare il percorso finale in cui sarà possibile recuperarle.

4.2.2 Procedura di salvataggio

Una volta inizializzato Azure Blob Storage, le Azure Functions potranno connettersi per creare e gestire i container. La richiesta parte dal client verso le Azure Functions, a seguito della selezione dei file multimediali da parte dell'utente. La richiesta include, oltre all'hash dell'evento interessato, la lista delle estensioni dei file che si vuole salvare.

Ricevuta la lista delle estensioni, la funzione chiamata esegue una verifica dei permessi di accesso necessari, per poi connettersi a Cosmos DB, dopo aver controllato che le

estensioni siano tra quelle accettate dal sistema, essersi accertato che l'evento esista e aver generato gli hash per le future immagini. Inserisce quindi nel database i nuovi riferimenti logici alle immagini, che includono, oltre al loro hash, il riferimento all'evento, la loro estensione e lo stato di elaborazione, inizializzato a "Created". Si connette quindi ad Azure Blob Storage per ottenere un Shared Access Signarure (SAS) token.

Il SAS token è un codice che viene usato per fornire permessi su servizi a entità terze. Chiunque sia in possesso del token ha la possibilità di eseguire le modifiche a esso collegate. È infatti possibile definire i permessi, la durata e la località da associare al token. In particolare, essendo le immagini salvate in un percorso logico che segue l'evento, il token richiesto dalla funzione e generato dal ABS avrà la possibilità di creare e scrivere file sul container "eventi", nel percorso dell'evento corrispondente. La validità di utilizzo ha durata dieci minuti.

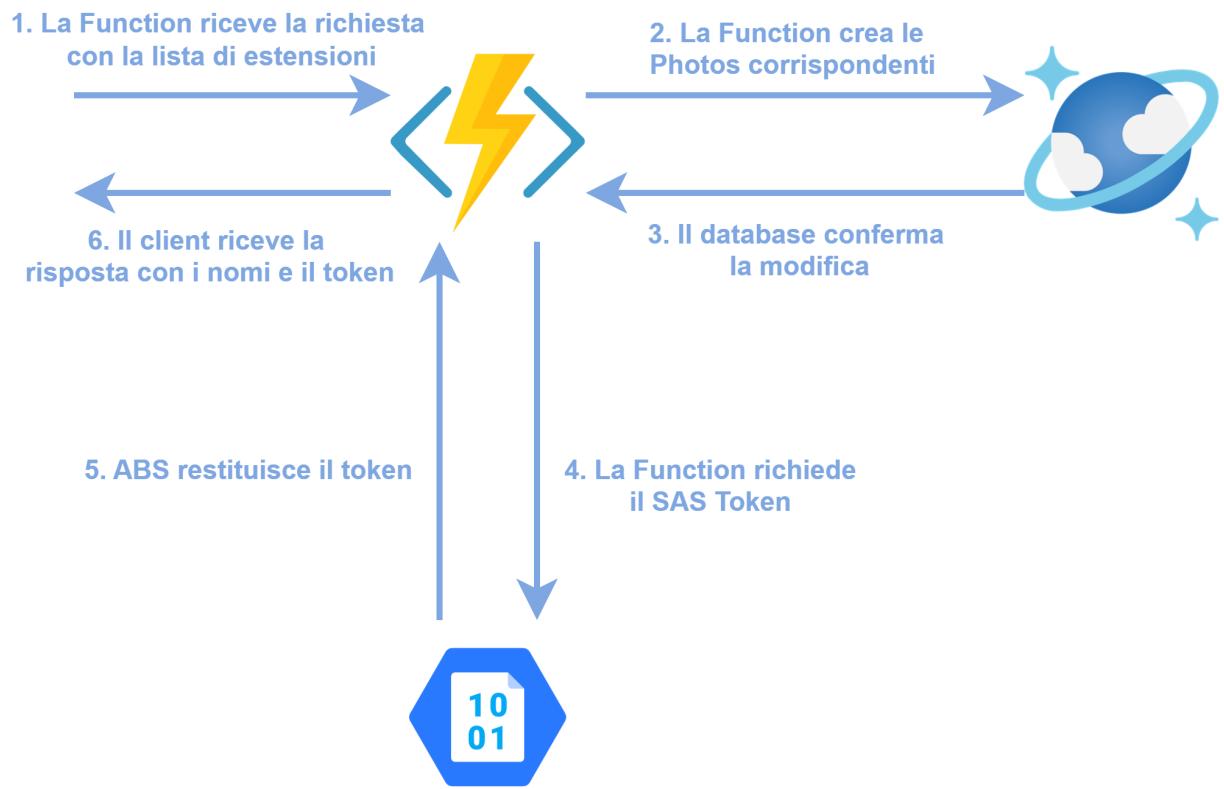


Figura 4.2: Interazione logica per l'ottenimento del token

La risposta verso il client sarà quindi composta, oltre che dal token di accesso, anche dai nomi delle immagini, affiancate dalle loro estensioni. Ricevuta la risposta il client

procederà, in simultanea, a comprimere i file selezionati, per ridurre 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. Nel momento in cui finisce la procedura di compressione, il device contatterà il Blob Storage e, usando il SAS token, caricherà le immagini usando il nome associato con l'estensione corretta.

La terminazione del caricamento è collegata all'esecuzione di una Azure Function. Il ruolo di questa funzione è quello di controllare che il file caricato corrisponda a un'immagine logica sul database. Controllerà quindi che il nome del file e il percorso associato corrispondano al nome dell'immagine e all'evento relativo, assicurandosi inoltre che l'estensione non sia cambiata e che il suo stato sia lo stesso con cui è stato inizializzato. In caso uno di questi parametri non coincida, procederà eliminando di entrambe le risorse, ovvero sia l'immagine logica sul database che il blob caricato. Altrimenti, se tutto risulta corretto, procederà ad aggiornare lo stato dell'immagine a "Uploaded".

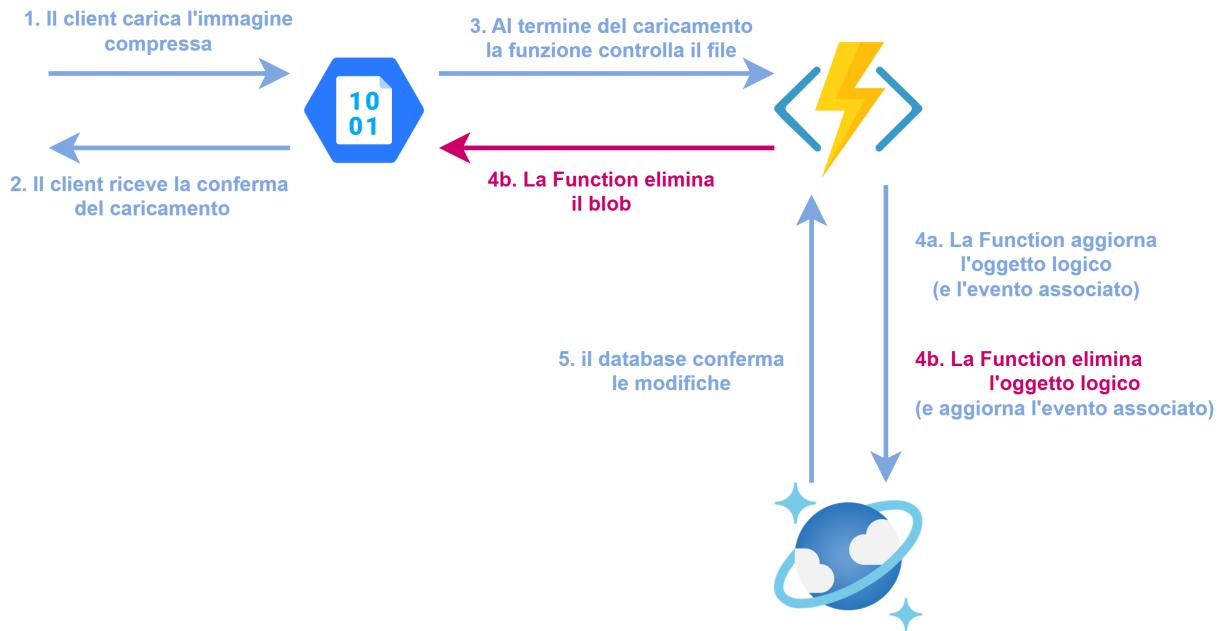


Figura 4.3: Interazione logica per il caricamento dell'immagine

Il caricamento di un'immagine costituisce una modifica all'evento. Per questo motivo è necessario aggiornare la data di aggiornamento dell'evento. Per evitare che troppe richieste ricadano sullo stesso evento, sarà solo l'ultima funzione chiamata a effettuare la modifica. A seguito delle operazioni precedenti la funzione controllerà infatti se, fra tutte

quelle associate all’evento, ci siano immagini con lo stato ancora in "Started". In caso contrario si deduce che la funzione corrente è l’ultima ad aver effettuato la modifica, e sarà lei quindi ad aggiornare l’evento relativo. L’aggiornamento dell’evento comporta la propagazione della notifica a tutti i profili coinvolti.

Essendo le immagini così create indipendenti tra loro, la loro modifica non richiede il blocco o il controllo di altre entità. Andando a modificare singolarmente le entità logiche delle immagini, questa operazione risulta altamente parallelizzabile, garantendo così la scalabilità delle richieste.

L’accesso in lettura ai file multimediali in ABS risulta pubblico per impostazione pre-definita, poiché la mancata definizione di ruoli comporta l’assenza di controlli esplicativi sulle autorizzazioni delle richieste. Questo aspetto pone delle problematiche sulla privacy delle immagini, che viene però mitigato grazie all’uso di hash randomici sufficientemente lunghi, che riducono drasticamente la probabilità di collisione. Infatti, 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 il successo un attacco altamente improbabile. Inoltre, anche in caso di compromissione di un hash, e quindi la possibilità di recupero di un’immagine da parte di un attore il cui accesso non dovrebbe essere permesso, l’accesso sarebbe limitato a una singola immagine, senza fornire ulteriori informazioni sugli altri file memorizzati.

Capitolo 5

Per poter definire l'efficienza e l'effettiva scalabilità del sistema creato è necessario misurare le sue prestazioni. A questo fine, vengono eseguiti dei test che effettuano un elevato numero di richieste, per valutare il suo comportamento in momenti di grande utilizzo e verificare così che sia garantita la qualità del servizio, anche in situazioni di stress.

Nell'ottica di ottenere dei risultati che descrivano fedelmente il comportamento dell'applicazione, evitando però di eseguire un test su ogni funzione implementata, sono state selezionate alcune funzionalità. La scelta di queste funzionalità deriva dalla capacità di poter ricondurre le loro prestazioni a tutto il resto del sistema, in base sia alla similitudine nel comportamento, che all'utilizzo dei vari servizi, che permette di misurarne la qualità della cooperazione.

In particolare, le funzionalità di creazione e selezione degli eventi permettono di valutare direttamente le prestazioni in scrittura e lettura dei dati sul database, influenzando Azure Functions e Cosmos. L'aggiornamento di un dato di un evento, richiedendo una scrittura parziale, consente di misurare anche questa particolarità dei database non relazionali. Ancora più rilevanti però sono le conseguenze che scatena. Per garantire la consistenza la funzione aggiunge infatti un messaggio in coda al Service Bus, che verrà poi preso in carico da un'altra Azure Function. Infine, la richiesta di caricamento delle immagini ci permette di valutare il rapporto con Azure Storage Container, ultimo componente fondamentale dell'applicazione.

5.1 Impostazione dei test

Per l’implementazione dei test è stato usato Azure Load Testing. Azure Load Testing(ALT) è un servizio che dà la possibilità di eseguire delle richieste secondo un carico impostabile, per poi monitorarne e mostrare i risultati. Questo permette di testare le prestazioni dei servizi direttamente all’interno della piattaforma. Integrandosi con l’ambiente Azure consente inoltre di associare al test il monitoraggio di ulteriori risorse, allineando ai risultati complessivi le prestazioni riportate dai singoli componenti, permettendo analisi più comprehensive e puntuali.



Azure Load Testing

Il carico viene definito tramite diversi parametri che possono essere impostati attraverso l’interfaccia o in un file di configurazione. Vengono così stabiliti la durata del test, il numero di istanze (ovvero server fisici su cui verrà eseguito il test) e il numero di thread per istanza. Per thread si intende un processo che, in parallelo con gli altri, esegue le richieste, fornendo un’idea generale sulla quantità delle richieste che verranno create. Se le istanze sono più di una, si può decidere di distribuirle su server in diverse posizioni geografiche, per ottenere informazioni relative alla qualità del servizio in diverse parti del globo.

Azure presenta tre modalità secondo le quali il carico può variare: lineare, in cui il carico varia stabilmente, scalare, nel quale il carico aumenta ogni intervallo di una quantità stabilita e picco, in cui il carico viene concentrato in un breve intervallo di tempo. Una volta stabilita la modalità, sarà necessario definire le caratteristiche che descrivono il suo comportamento. Ad esempio, la modalità lineare necessita di sapere in quanto tempo arrivare al carico massimo. In ogni caso è richiesta la durata del test.

I test possono essere eseguiti tramite url o usando un file di configurazione. Il primo caso necessita che la funzione da testare risponda a una richiesta HTTP. Il test prevede semplicemente la sua invocazione per il numero desiderato di volte. Permette di essere configurata completamente tramite interfaccia grafica, e risponde alla maggioranza delle necessità per testare prestazioni generiche. Nel caso in cui si voglia però misurare fun-

zionalità più complesse, che richiedono l’interazione con file esterni, la modifica in corso delle richieste o l’esecuzione di più richieste in successione, è necessario utilizzare un file apposito.

All’interno del file di configurazione bisogna indicare, tramite un apposito codice, il carico voluto e le richieste da eseguire. Oltre a fornire una maggiore precisione nella definizione del carico, la possibilità di impostare la modalità del test consente un’elevata libertà nel stabilire le proprietà delle richieste e le loro eventuali interazioni. Questo permette, ad esempio, di caricare e leggere file, che possono contenere informazioni utili per variare le richieste o possono essere usati per scambiare dati. Si può inoltre simulare un’interazione più complessa, in cui si aspetta una richiesta per ricevere il risultato, per poi usarlo per inviarne una seconda.

Il file può essere sviluppato usando Apache JMeter o Locust come codici di configurazione. I test sono stati implementati in JMeter, essendo quello utilizzato di default da Azure. L’interfaccia di Azure Load Testing per la creazione del test tramite del file di configurazione prevede il suo inserimento, assieme a tutti gli eventuali file allegati di cui necessita. Prevede inoltre l’impostazione del numero di istanze per il quale si vuole distribuire il test, nelle quali verrà duplicato.

Per ogni test di entrambe le modalità si possono infine specificare quali altre risorse del progetto monitorare, affiancando alle prestazioni generali del test le prestazioni specifiche dei singoli componenti.

5.2 Velocità della procedura di creazione

Il primo test si concentra sulla velocità di creazione di elementi. La creazione è un’operazione che richiede l’inizializzazione logica di oggetti in un’Azure function, per poi procedere a salvarli sul database. Per quanto l’interazione logica comporti un tempo limitato, il salvataggio dei dati sul database è un’azione che richiede normalmente tempi significativi, in quanto comporta la scrittura diretta sul disco.

In particolare, si è scelto di testare la creazione di un nuovo evento, vista sia la sua centralità nell'applicazione che la sua complessità logica. La creazione di un nuovo evento infatti comporta, oltre alla creazione dell'elemento Event stesso, l'inizializzazione dei ProfileEvent ed EventProfile associati. Il loro salvataggio avviene in seguito alla creazione dell'oggetto Event, per cui è introdotta una logica transazionale che garantisce l'atomicità della richiesta. L'esecuzione di questo test consente quindi di analizzare la capacità del sistema nel ricevere ed eseguire scritture complesse sul database, senza il coinvolgimento di ulteriori risorse.

La funzione di creazione di un evento è semplice, richiedendo un solo passaggio e presentando dei dati in ingresso che possono essere prestabiliti. Viene quindi usato un test che invoca la funzione tramite url, con un carico di richieste della durata di due minuti, che aumenta linearmente nei primi trenta secondi per poi stabilizzarsi. Questo permette di analizzare il sistema sia in base al numero di richieste in contemporanea, che durante un periodo di richieste alto ma costante. Al test vengono affiancate le metriche delle Azure Functions, monitorate attraverso le Application Insights, per misurare i risultati lato server, assieme a quelle del database Cosmos, per determinare con più facilità la causa di eventuali errori.

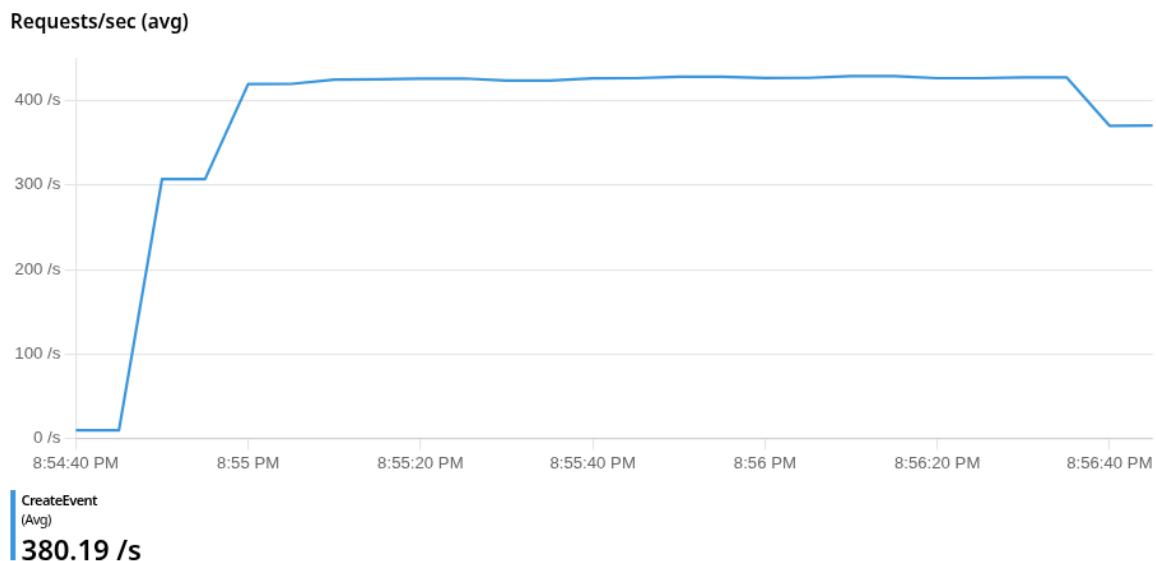


Figura 5.1: Carico delle richieste per la creazione di eventi

Il test così creato ha portato alla creazione di 49425 Event nell’arco di due minuti, sostenendo con successo un carico di, mediamente, 386 richieste al secondo. Tutte le richieste sono andate a buon fine, con un tempo di risposta di 243 ms. Questa durata è però comprensiva del tempo di trasmissione dei dati. Analizzando i dati registrati da Application Insights, si nota che il tempo medio delle richieste si abbassa a 133 ms.



Figura 5.2: Tempo medio delle richieste lato server

Si può quindi affermare che il sistema è in grado di affrontare efficacemente richieste di scritture sul database continue e parallele, anche complesse o multiple, limitate, per ora solamente dalle risorse dei servizi. Il tempo maggiore rilevato all’inizio delle richieste è riconducibile al tempo di avviamento delle risorse, che rimane stabile una volta inizializzato.

Load 49.42K Total requests	Duration 2 minutes 8 seconds	Response time 243.00 ms 90th percentile response time	Error percentage 0 % Aggregate requests which failed	Throughput 386.13 /s Request rate
---	--	--	---	--

Figura 5.3: Risultati del test di creazione

5.3 Velocità in lettura

L’operazione di recupero delle informazioni è centrale per fornire all’utente un’esperienza fluida e consistente. Risulta quindi fondamentale che il tempo di identificazione e lettura dei dati sia il più basso possibile. Sebbene la scalabilità sia facilitata dalla natura dell’azione, che può sfruttare copie e può avvenire in contemporanea sullo stesso elemento, è bene poter garantire che la risposta a operazioni di lettura anche complesse non venga degradata in base alla quantità delle richieste.

Tra le varie richieste dirette di dati, che quindi non comportano aggiornamenti o ulteriori processi, è stato scelto il recupero degli eventi relativi al profilo corrente. Questa operazione viene eseguita ogni volta che l’utente chiede dati relativi a un intervallo, quando non sono presenti dati in memoria locale. Oltre a essere importante per il funzionamento generale dell’applicazione, è anche una delle richieste di lettura più complesse. Deve infatti recuperare i ProfileEvent relativi al profilo in quell’intervallo, per poi recuperare gli EventProfile associati da cui poi ricavare l’Event corrispondente. Per ogni Event così trovato deve poi essere creato un EventDto che contenga sia le informazioni dell’Event che di ProfileEvent. Tutte queste operazioni sono definite in un’Azure Function, ma delegate direttamente a Cosmos. Il numero degli eventi trovati viene limitato a 40.

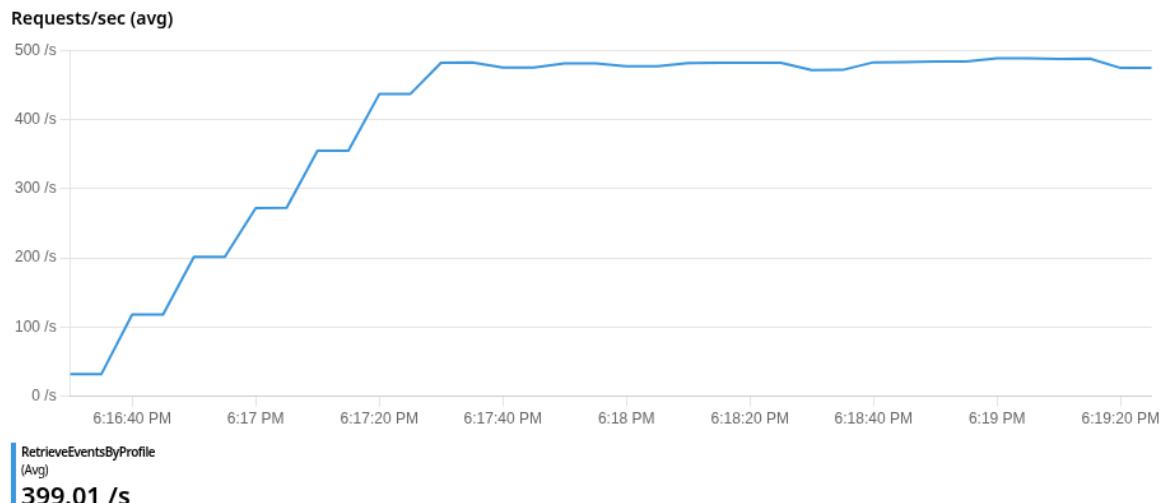


Figura 5.4: Carico delle richieste per il recupero di eventi

Anche in questo caso, vista la semplicità della richiesta è sufficiente una richiesta tramite url. Impostato l'identificativo del profilo all'interno della richiesta, il test procederà inviando richieste seguendo un carico della durata di tre minuti, impiegando i primi trenta secondi a raggiungere linearmente il carico massimo di circa 480 richieste al secondo. Influenzando solo le Azure Functions e Cosmos, verranno monitorate, come in precedenza, solo le Application Insights e Cosmos stesso.

Il test ha comportato la lettura di quasi settantamila richieste, con una media di quattrocento richieste al secondo. Non sono stati riscontrati errori. Il tempo di risposta totale medio è risultato di duecentoquindici millisecondi per richiesta, che scende però sotto i cento se si considera il tempo di esecuzione effettivamente impiegato dal server.

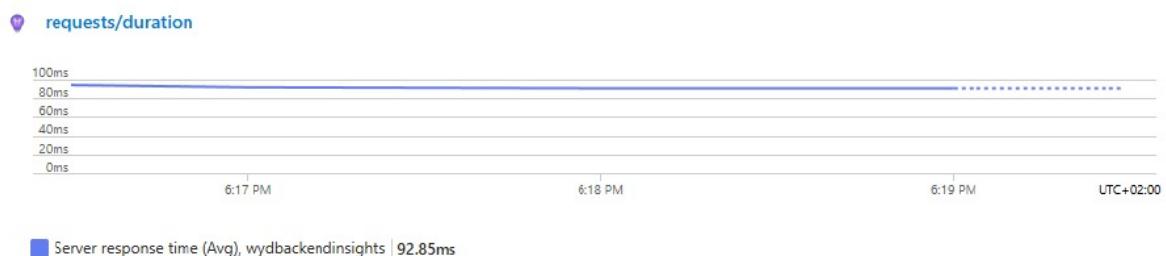


Figura 5.5: Tempo medio delle richieste lato server

Il successo di questo test conferma le capacità del sistema di rispondere a carichi elevati di richieste in lettura contemporanee, mantenendo le prestazioni costanti sia in caso di aumento di carico che in caso di richieste frequenti.

Load 71.82K	Duration 2 minutes 59 seconds	Response time 215.00 ms 90th percentile response time	Error percentage 0 % Aggregate requests which failed	Throughput 401.24 /s Request rate
-----------------------	---	--	---	--

Figura 5.6: Risultati del test di creazione

5.4 Velocità degli aggiornamenti

L’aggiornamento di un dato su un database non relazionale è un’operazione che si distingue dalle precedenti per due principali caratteristiche. La prima peculiarità consiste nella necessità di ritrovare i dati, comportando una ricerca sul database. La seconda consiste nella possibilità di modificare parti del documento senza sovrascriverlo completamente. Per mettere alla prova queste capacità si è deciso di monitorare le risposte alla modifica di un evento. Oltre a essere la funzione di aggiornamento probabilmente più richiesta, permette di testare le prestazioni delle code create per la propagazione delle modifiche.

Una volta garantita l’applicazione delle modifiche sull’Event coinvolto, la funzione deve inoltre assicurare che queste vengano propagate a tutti gli elementi sulle quali sono state denormalizzate le sue informazioni: in questo caso, tutti i ProfileEvent a esso associati. I componenti principali coinvolti rimangono Azure Function e Cosmos database, a cui si aggiunge però Azure Service Bus, che prende in carico il messaggio con cui verrà poi eseguita la funzione responsabile di sincronizzare le informazioni con le altre entità coinvolte nel database. La funzione principale dovrà quindi connettersi con il database per effettuare le modifiche, per poi inviare il messaggio al Service Bus, assicurandosi che sia stato in carico. Nel caso una di queste due fasi non vada a buon fine, l’intera funzione è da considerarsi fallita.

La funzione principale risponde a una chiamata HTTP per apportare le modifiche richieste con un aggiornamento parziale sul database. L’impostazione del test tramite url comporterebbe però la convergenza di tutte le richieste sullo stesso Event, compromettendo l’affidabilità dei risultati, in quanto rappresentazione di un caso molto particolare quanto improbabile. Per risolvere questo problema è stato un file JMeter che, importando da un file l’identificativo dell’Event, garantisce che le richieste siano distribuite su Event differenti. Il file allegato contiene tutti gli identificativi degli eventi creati nel primo test, e il file di configurazione è stato impostato per associare una sola richiesta allo stesso Id. Questo permette un miglior controllo per i risultati del test successivo.

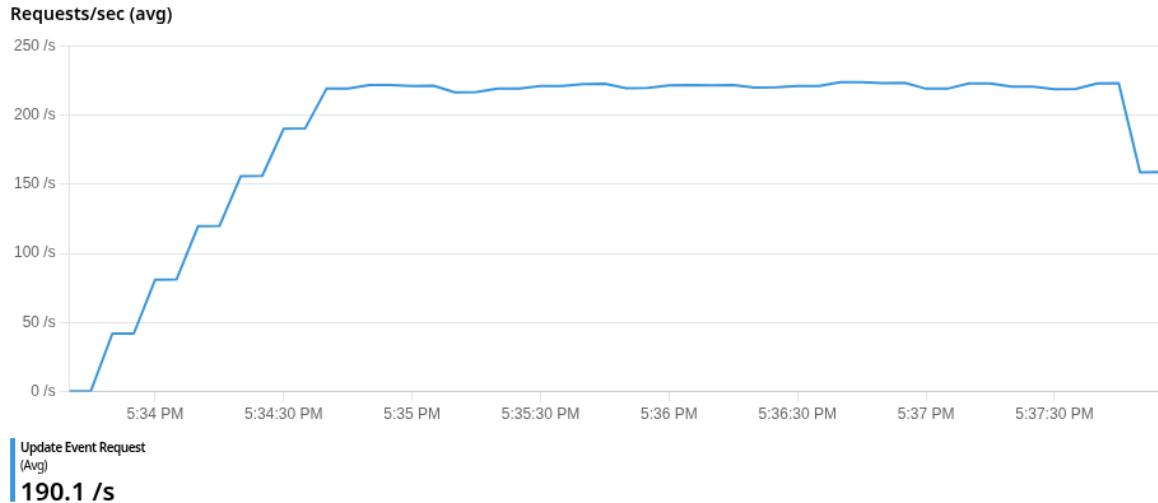


Figura 5.7: Carico delle richieste per l'aggiornamento di eventi

La durata del test dipende dal tempo necessario per processare tutti gli identificativi presenti sul file allegato. Come per gli altri casi, si è però deciso di aumentare il carico linearmente per i primi trenta secondi, per testare sia la risposta in caso di aumento del carico che sotto a stress elevato ma costante. Al monitoraggio di Application Insights e di Cosmos viene aggiunto quello di Service Bus, per assicurarsi che effettivamente i messaggi vengano aggiunti alla coda dedicata.

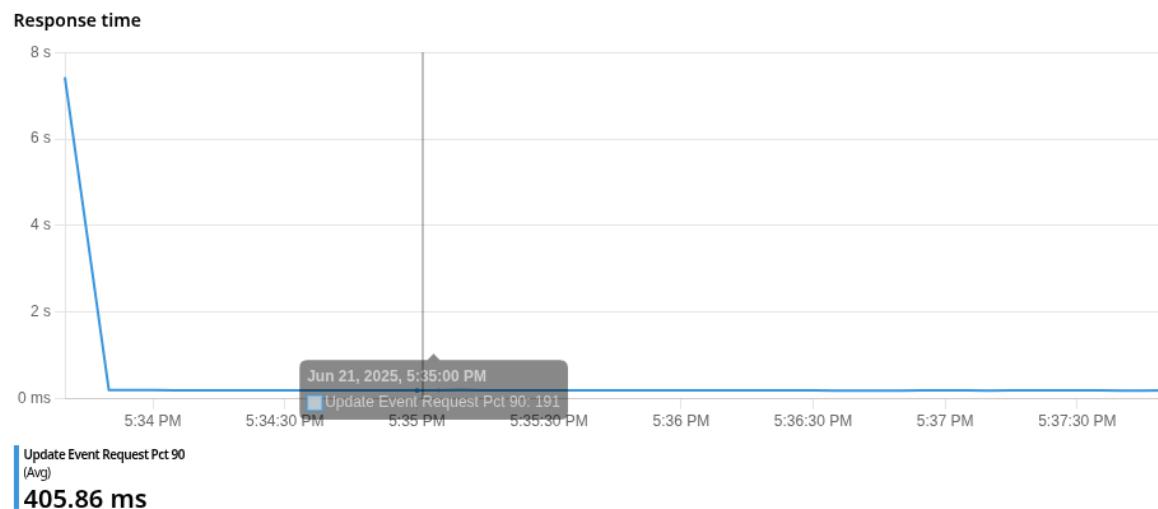


Figura 5.8: Durata delle richieste di modifica dell'evento

Il test ha così prodotto un carico medio di duecento richieste al secondo distribuite in quattro minuti e mezzo, con un picco che si è mantenuto costante negli ultimi tre minuti di duecentoventi richieste al secondo. Nonostante un tempo iniziale elevato dovuto all'inizializzazione del server, che impatta sul tempo medio della risposta, il tempo medio totale richiesto per l'esecuzione della richiesta è rimasto costante sui duecento millisecondi, che includono però anche il tempo di trasmissione dei dati e della risposta. Tutte le richieste sono andate a buon fine.

Load 49.43K Total requests	Duration 4 minutes 17 seconds	Response time 192.00 ms 90th percentile response time	Error percentage 0 % Aggregate requests which failed	Throughput 192.32 /s Request rate
---	---	--	---	--

Figura 5.9: Risultati del test di aggiornamento

Le prestazioni riscontrate risultano in linea con gli altri risultati, con un tempo medio di esecuzione effettiva intorno ai centocinquanta millisecondi, garantendo anche in questo caso un'interazione fluida. L'aumento o la frequenza della quantità delle richieste non impatta sul funzionamento o sul ritardo della risposta, che rimangono inalterati durante tutta la durata del test. Il tempo effettivo impiegato dalle Azure Function non è stato analizzato, in quanto comprensivo della durata delle funzioni che, in contemporanea, stavano propagando le modifiche sul ProfileEvent.

5.5 Garanzia di propagazione delle informazioni

La presenza di elementi denormalizzati comporta la necessità di propagare le modifiche sugli elementi coinvolti indirettamente. Potendo accettare un ritardo tra la modifica dell'elemento principale e l'aggiornamento delle sue controparti, questo compito viene delegato a un'altra funzione, riducendo il carico di lavoro (e quindi il tempo di risposta) della prima. La separazione di questi compiti può essere però sfruttata solo se viene garantita l'esecuzione della funzione secondaria incaricata di propagare gli aggiornamenti.

5 – Capitolo 5

Service Bus è stato scelto proprio per la sua proprietà di assicurare che il messaggio venga preso in carico da una funzione, per poi controllarne l'esecuzione e ripeterla in caso di fallimento. Dai grafici relativi all'Application Insights si nota una maggiore quantità di esecuzioni, così come un tempo medio superiore a quello previsto per l'aggiornamento. Questo porta a dedurre che le funzioni vengano effettivamente invocate, e che il loro tempo di esecuzione sia superiore.

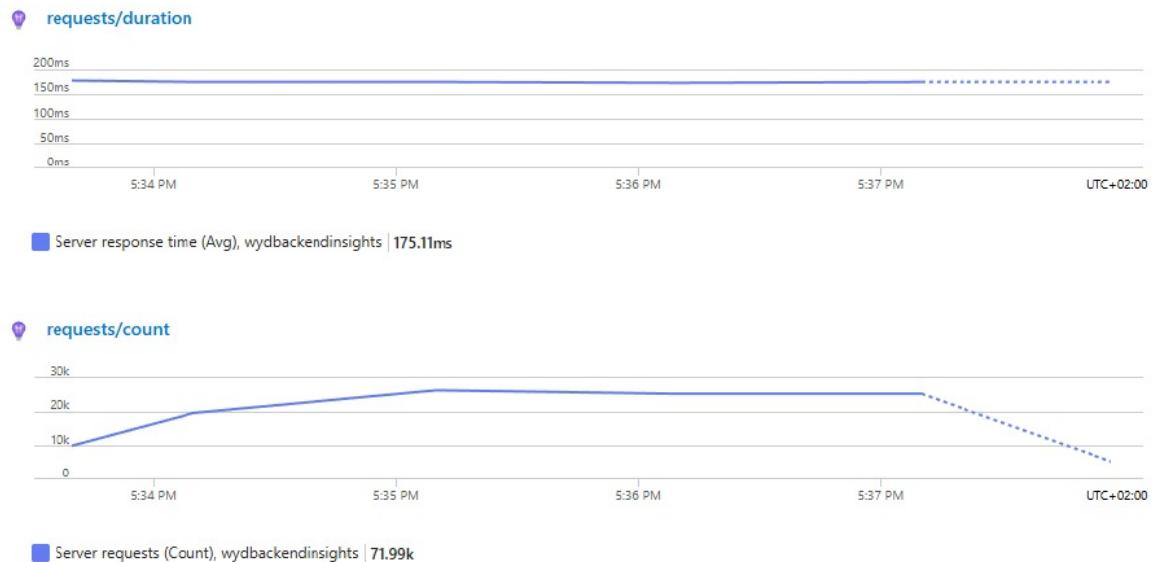


Figura 5.10: Durata e quantità delle Azure Function durante l'aggiornamento

Il monitoraggio di Service Bus conferma ulteriormente la ricezione dei messaggi e l'effettiva presa in carico della funzione.

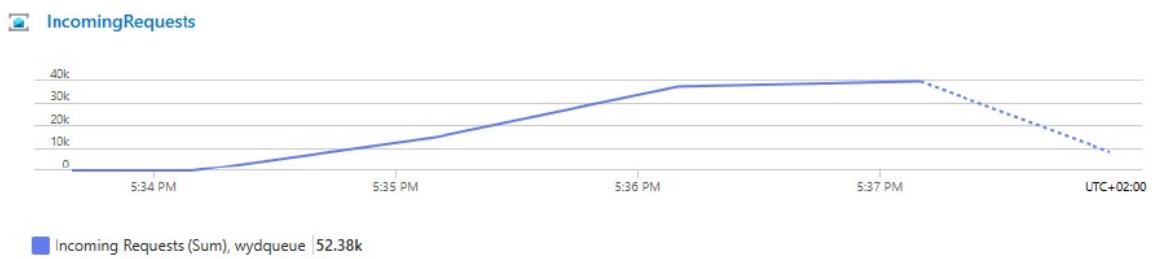
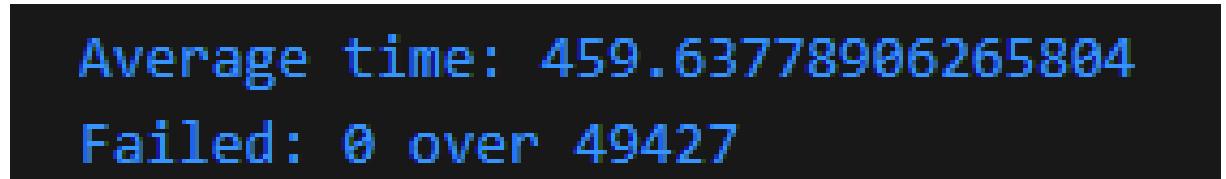


Figura 5.11: Somma dei messaggi ricevuti da Service Bus

Nonostante la conferma che il collegamento tra Service Bus e Azure Functions funzioni correttamente in su entrambi i versi, è necessario assicurare che gli aggiornamenti siano sempre terminati con successo. Per controllare che tutti gli aggiornamenti dei ProfileEvent siano andati a buon fine, il messaggio inviato sul Service Bus che notifica l'aggiornamento dell'Event contiene anche il timestamp del momento di attuazione della modifica.

La funzione incaricata di allineare i ProfileEvent salverà quindi, oltre al momento corrente in cui l'aggiornamento avviene, anche il timestamp dell'aggiornamento dell'Event. Questo consente, in un secondo tempo, sia di certificare che l'aggiornamento sia avvenuto correttamente, sia di calcolare il tempo medio che trascorre tra la modifica dell'Event e l'effettiva propagazione verso i ProfileEvent.

Tramite un programma creato appositamente, a termine dell'operazione di aggiornamento degli Event, sono stati recuperati tutti i ProfileEvent a essi relativi. Per verificare il successo del processo di propagazione è stato controllato che il momento di aggiornamento del ProfileEvent fosse successivo a quello dell'Event, per poi calcolarne la differenza.



```
Average time: 459.63778906265894
Failed: 0 over 49427
```

Figura 5.12: Risultati del programma di controllo sulla propagazione degli aggiornamenti

I risultati del programma hanno successivamente confermato l'affidabilità del processo di propagazione dell'aggiornamento tramite Service Bus, con un tempo medio di cinquecento millisecondi.

5.6 Velocità di salvataggio delle immagini

L'ultimo servizio che non è ancora stato testato, tra i principali che compongono l'applicazione, è il gestore dei file multimediali: Azure Storage Container. Azure Storage Container ha la responsabilità di generare i Token di sicurezza, controllarli e gestire il caricamento (e il ritrovamento) delle immagini. Svolge un ruolo centrale durante la fase di caricamento dei file multimediali, a seguito del termine dell'evento.

Il caricamento delle immagini avviene in due fasi. Il client contatta una Azure Function indicando l'estensione dei file che vuole caricare e l'identificativo dell'evento correlato. La funzione invocata crea le istanze sul database, per poi contattare Azure Storage Container e ottenere il token che permette all'utente di caricare i file. Ricevuto il token e gli identificativi delle immagini, il client contatta direttamente Azure Storage Container per salvare direttamente i file.

Dati i due passaggi e la necessità di allegare i file da caricare, il test è stato implementato in JMeter. Il test prevede due richieste, la prima verso Azure Function e la seconda verso Azure Storage Container. Tra la prima e la seconda richiesta verrà analizzata la risposta per estrarre il token e l'identificativo delle immagini create, per poi caricare i file allegati e impostare correttamente il corpo della richiesta successiva.

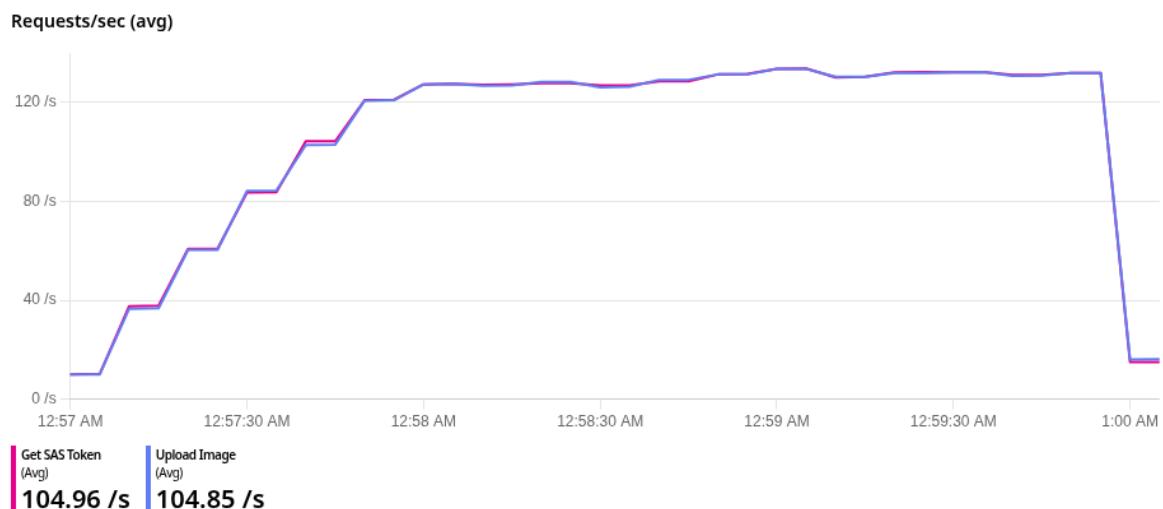


Figura 5.13: Carico delle richieste di caricamento delle immagini

Il carico è stato impostato per aumentare gradualmente nei primi trenta secondi, per poi continuare stabilmente fino a una durata massima di tre minuti. L'immagine allegata ha una dimensione di quattro megabyte, in linea con la dimensione media di un'immagine effettuata da uno smartphone moderno. In aggiunta alle metriche delle Azure Functions e del database, si monitora anche Azure Storage Container.

Load 39.87K Total requests	Duration 3 minutes 1 seconds	Response time 610.00 ms 90th percentile response time	Error percentage 0 % Aggregate requests which failed	Throughput 220.25 /s Request rate
---	--	--	---	--

Figura 5.14: Risultati del caricamento delle immagini

Il test separa le due richieste, dando informazioni dettagliate sul comportamento di entrambe, ma unendo infine i risultati. Risultano quindi quasi quarantamila richieste effettuate in tre minuti, equamente suddivise tra richieste alle Functions e richieste di caricamento. Le richieste al secondo calcolate sono da considerarsi la somma delle due, per cui, su un totale di duecentoventi richieste medie al secondo, il carico effettivamente ricevuto dallo Storage Container è di circa centodieci richieste al secondo.

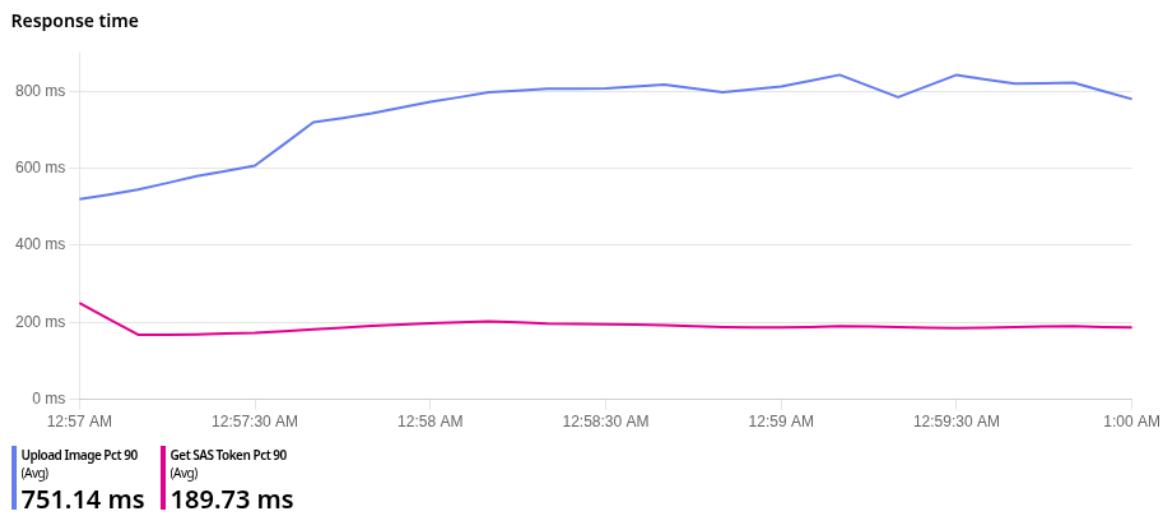


Figura 5.15: Tempi di risposta delle due richieste

La richiesta alle Azure Function risulta affidabile, mantenendo un tempo di risposta costante nonostante il carico, in linea con le richieste di scrittura sul database. Questo indica che l'operazione di generazione del token non produce un impatto significativo sulle prestazioni e presenta le stesse caratteristiche di scalabilità del resto del sistema. Analizzando invece il tempo di risposta del caricamento delle immagini, più alto a causa della dimensione della richiesta, ma ampiamente accettabile, si nota però che varia in base al carico delle richieste.

L'aumento del tempo di salvataggio delle immagini in linea con il carico comporta il pericolo che il sistema non risulti scalabile. Per quanto sia positivo che il tempo di risposta non vari nonostante un periodo prolungato di richieste, indicando una distribuzione delle risorse equilibrata e ben gestita, la correlazione tra quantità di richieste e latenza del sistema potrebbe risultare problematica se continuasse ad aumentare senza mai stabilizzarsi.

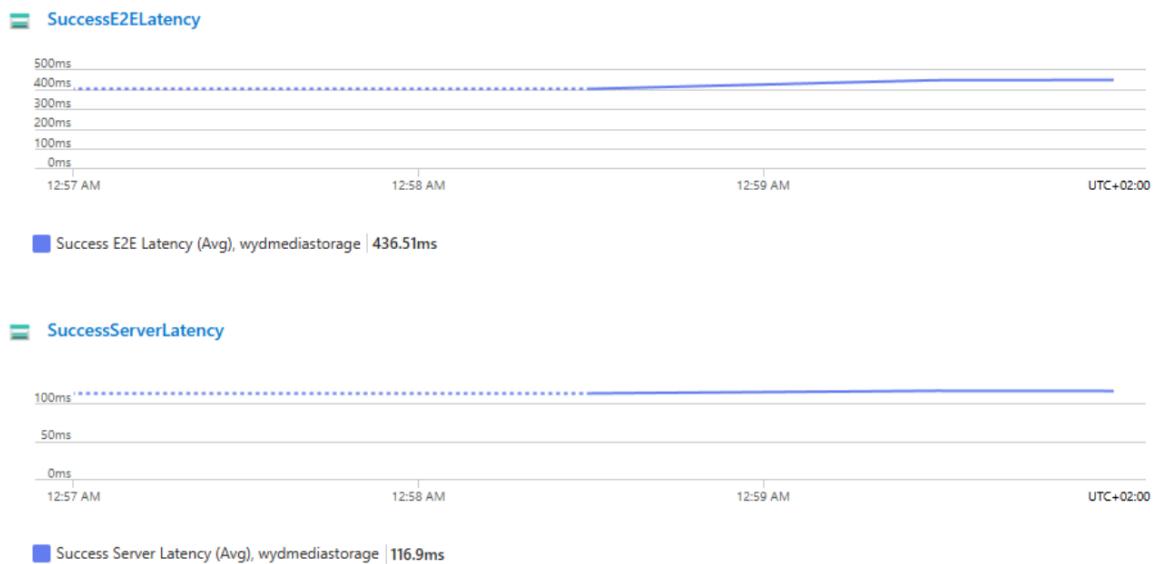


Figura 5.16: Dettaglio del tempo di risposta di Azure Storage Container

In generale, il caricamento delle immagini ha dato risultati positivi, garantendo la resilienza al caricamento simultaneo di file multimediali da parte di molti utenti in contemporanea, ma è necessario approfondire ulteriormente le prestazioni di caricamento delle immagini, per verificare ulteriormente le proprietà di scalabilità fornite da Azure Storage Service.

5.7 Costo previsto del sistema

Sebbene lo scopo di questa tesi sia evidenziare l'effetto delle scelte implementative sulla scalabilità del sistema, l'aspetto economico non può essere ignorato durante la realizzazione di un progetto. Si cerca quindi di prevedere la spesa effettiva delle risorse utilizzate in base a un utilizzo possibile, soffermandosi maggiormente sulla dinamica di definizione dei parametri richiesti rispetto all'effettiva necessità, che può variare in base al mercato dell'applicazione o alla sua maturità.

Essendo l'applicazione in fase prototipale, si può solamente provare a dedurre l'effettivo utilizzo delle risorse proporzionalmente al carico previsto. Per il calcolo del costo del sistema si è deciso di considerare un carico di cinquemila utenti attivi giornalmente, che indica un utilizzo consistente e maturo dell'applicazione.

Per stimare il costo del sistema viene usato uno strumento offerto da Azure chiamato Pricing Simulator. Questo strumento, accessibile online, permette di calcolare il costo dell'utilizzo delle risorse in base all'utilizzo che si prevede di farne. Per ogni servizio, in base alle sue peculiarità, viene chiesto l'inserimento di dati specifici per il suo utilizzo. Per alcuni servizi, in cui era stato adottato il piano di utilizzo più economico, è stato necessario assumere l'adozione di un piano che permetta effettivamente di gestire il carico previsto. I costi vengono calcolati mensilmente.

Azure Static Web App prevede due tipologie di piani, gratuito o standard. Il piano gratuito fornisce 100 gigabyte di bandwidth e un massimo di 0,5 gigabyte di memoria. La dimensione attuale dell'applicazione risulta di circa 20 megabyte, che viene però compressa dal sistema. Il limite della memoria non risulta quindi un problema, ma bisogna verificare che non si superi la quantità fornita di bandwidth.

La dimensione dei dati ricevuti dal browser in fase di caricamento del sito ammonta a un massimo di 10 megabyte. Questa quantità si riduce notevolmente se il sito è già stato caricato in cache, a un massimo stimato di 250 kilobyte. Considerando che il metodo di fruizione principale del progetto avvenga tramite applicazione, si stimano 2500 utenti mensili da browser. Stimando un utilizzo quotidiano dell'applicazione, il primo giorno

sarà necessario scaricare l'intero sito, mentre le volte successive solo aggiornare la cache, che porta il calcolo della quantità di dati effettivamente trasferita al seguente:

$$2500 \times (10 \text{ Mb} + 250 \text{ Kb} \times 29) = 43 \text{ Gb.}$$

I quarantatré gigabyte così previsti rientrano ampiamente nei cento offerti, permettendo l'utilizzo del piano gratuito.

Per stimare il costo delle Azure Functions è necessario calcolare le interazioni totali mensili con gli utenti. Queste vengono previste per 200 richieste giornaliere, che includono sia richieste in scrittura che in lettura. Si calcolano così un milione di invocazioni giornaliere, per un totale di trenta milioni di chiamate mensili.

Dai test si evince una durata media delle funzioni che si aggira sui 200 millisecondi. Per stare sul sicuro, la durata prevista per richiesta viene arrotondata a 250 millisecondi. Allo stesso modo, si prevede un utilizzo medio di 0.5 gigabyte di memoria per ogni invocazione. Nel calcolo bisogna considerare le risorse gratuite e un costo di € 0,000015 per GB/s e € 0,117 per milione di richieste, portando la spesa totale prevista attorno ai € 60.

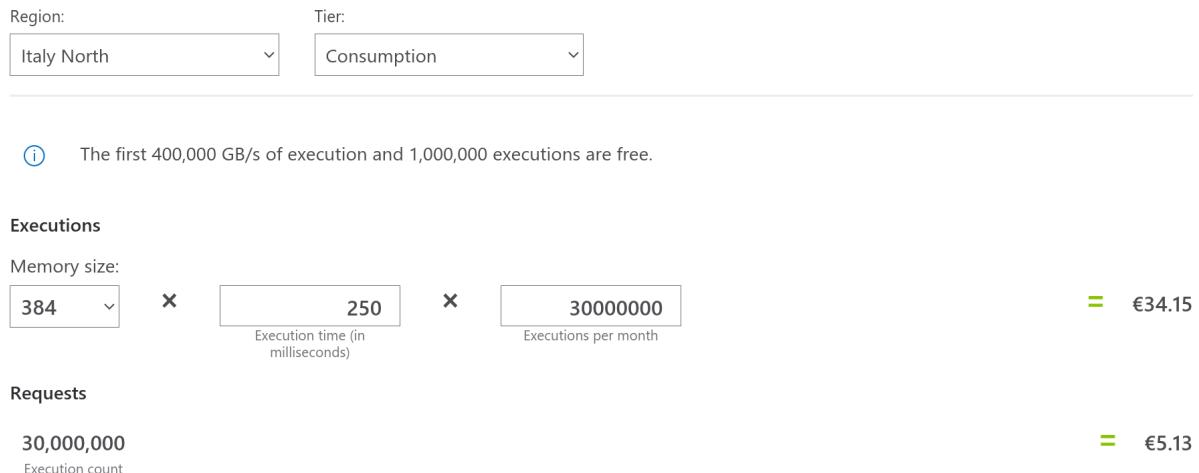


Figura 5.17: Calcolo per il costo delle Azure Functions

Il costo del database, implementato con Azure Cosmos DB, varia in base alle Request Units al secondo(RU/s) usate. Ogni richiesta consuma una certa quantità di RU, in base al carico computazionale necessario. Bisogna quindi stimare le richieste al secondo, e il

loro consumo di Request Unit. Considerando che la quasi totalità delle richieste verso le Azure Functions implicano un accesso al database, le invocazioni per secondo possono essere calcolate dividendo le richieste giornaliere per i secondi di una giornata, facendo così risultare una media di 12 richieste al secondo.

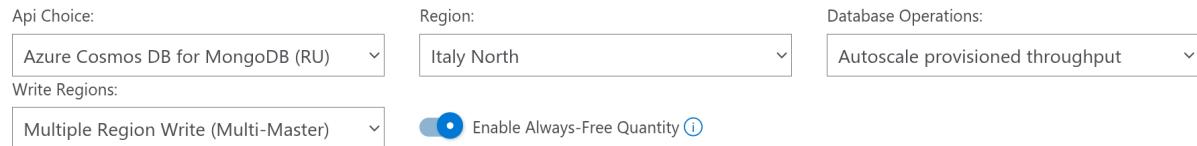


Figura 5.18: Impostazioni del server Cosmos

Il consumo medio di RU registrato durante i test varia tra i 10 e i 20 RU per richiesta. Per sicurezza, si considera il consumo medio di un'operazione sul database di 30 RU. Risulta così una media di 360 RU/s, che non fornisce però informazioni sul picco del carico. Utilizzando un piano di tipologia autoscale, che modifica in automatico le risorse del database in base al carico effettivo, bisogna stabilire il carico massimo che si prevede il database debba sostenere, tenendo però presente che la quantità minima di risorse sempre stanziate sarà del suo 10%.

Non essendo in possesso di dati relativi all'effettivo utilizzo dell'applicazione, Stimiamo un utilizzo massimo pari a 5 volte quello previsto, pari a 60 richieste al secondo. Questo porta a un massimo di 1800 RU/s, con una media prevista del 20%. Sottraendo i primi 1000 RU/s e moltiplicando i restanti per il costo orario di € 0,014, il costo totale previsto per l'utilizzo di Azure Cosmos si aggira intorno ai € 17.

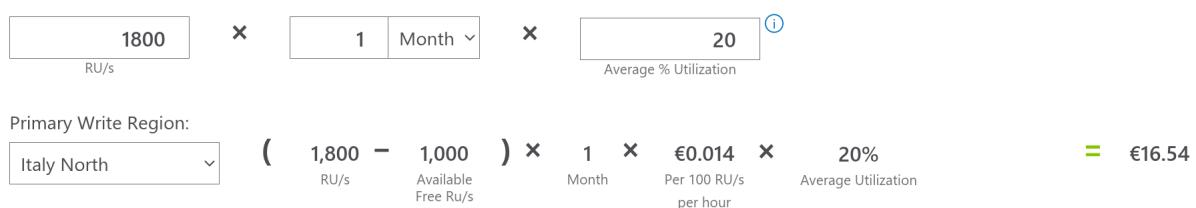


Figura 5.19: Calcolo per il costo di Cosmos

Per poter garantire la gestione totale dell'esecuzione dei messaggi è necessario impostare il piano Standard di Service Bus. Prevede un costo di base orario di € 0,012, più € 0,71 per ogni milione di messaggi, includendo però nel prezzo i primi 13 milioni. Stimando le operazioni in scrittura (che sono quelle che richiedono la propagazione) a un terzo di quelle totali, si prevedono 9 milioni di messaggi mensili, riducendo la spesa al solo costo della macchina, che ammonta così a circa € 9.

Le operazioni di aggiornamento sono le stesse che richiedono l'invio delle notifiche, scatenate tramite messaggio inviato su una Azure Storage Queue. Ogni richiesta comporta una scrittura sulla coda, che verrà presa in carico da una funzione, con un'operazione di lettura. Questo comporta 9 milioni di operazioni in lettura e altrettante in scrittura. A un costo di € 0,0035 ogni 10.000 operazioni, comporta una spesa di € 6,40, a cui si aggiunge € 1 per la capacità della coda che, per sicurezza, è stata stimata a 25 gigabyte. Questa quota difficilmente viene raggiunta in quanto un messaggio, dopo essere stato letto, dovrebbe essere eliminato.

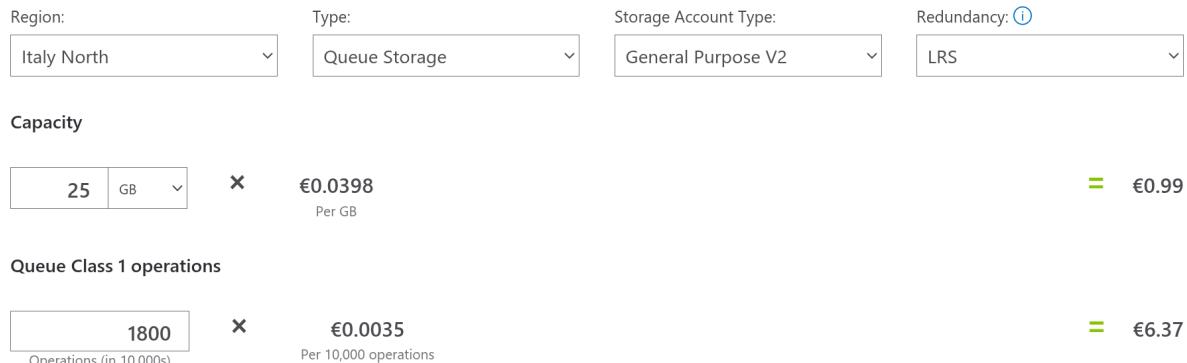


Figura 5.20: Calcolo per il costo di Azure Storage Queue

Per l'invio delle notifiche si utilizza Azure PubSub. La versione gratuita non è più sufficiente, potendo garantire un massimo di 20 connessioni contemporanee. Si stima però che, modificando il contratto a Standard, una istanza sia sufficiente, la quale, con un costo orario di € 0,0594, ammonta da sola a € 44 mensili. Stimando la dimensione media dei messaggi delle notifiche a 1 Kb, e prevedendo una media di 10 utenti attivi per ogni notifica, la dimensione totale dei messaggi così inviati con PubSub dovrebbe ammontare attorno a $9.000.000 \times 1 \text{ Kb} \times 10 = 90 \text{ Gb}$. Il costo per l'invio dei messaggi risultà così

pari a € 13, per un totale di circa € 56 mensili.

Per il calcolo del costo di Azure Blob Storage, che viene usato per salvare le immagini, viene stimato un carico di un’immagine al giorno per utente, o di 30 immagini al mese. Azure Blob Storage prevede un costo in base alla dimensione totale dei dati salvati, a cui si aggiunge il costo computazionale necessario per aggiungere o trovare le immagini. Considerata una dimensione media per immagine (compressa) di 200 Kb, ogni mese vengono aggiunti $30 \times 5000 \times 200 \text{ Kb} = 30 \text{ Gb}$. Dopo un anno di utilizzo stimiamo quindi una dimensione totale di 360 Gb, per una spesa mensile di € 6.

Si stima inoltre che un utente guardi giornalmente 3 eventi nel dettaglio, i quali sono mediamente condivisi con 10 utenti. Considerando il carico di una immagine al giorno per utente, ogni utente esegue, al giorno, 3 richieste di elenco e 30 di lettura. Si calcolano così 150.000 richieste di scrittura, 450.000 richieste di elenco, e 4.500.000 richieste di lettura. Con un prezzo di € 0,047 ogni 10.000 invocazioni per i primi due casi, e di € 0,004 ogni 10.000 invocazioni per le letture, il costo computazione ammonta a € 4,50. La spesa totale per il servizio viene quindi stimata a € 10,5.

Per il Key Vault bisogna stimare la frequenza con cui le Functions necessitano di recuperare i segreti per accedere agli altri servizi. Si stima che, come minimo, ogni richiesta abbia bisogno di almeno un segreto, ad esempio per accedere al database. Le Azure Functions sono però strutturate per riutilizzare l’ambiente di esecuzione, se le risorse computazionali lo permettono. Questo consente di riutilizzare i segreti recuperati dalle funzioni invocate precedentemente. Si stima quindi che il Key Vault venga interrogato una volta ogni dieci richieste, per un totale di 3 milioni al mese. Per un costo di € 0,027 ogni 10.000 invocazioni, la spesa totale di Azure Key Vault viene prevista per circa € 8.

L’ultima risorsa utilizzata all’interno dell’ecosistema Azure è Virtual Network, il cui utilizzo è però gratuito. Infine, il servizio per l’autenticazione, Firebase Authentication, risulta anch’esso gratuito, non superando la soglia dei 50.000 utenti attivi mensili.

Static Web Apps		Free tier		Upfront: €0.00	Monthly: €0.00
Azure Functions		Consumption tier, Pay as you go		Upfront: €0.00	Monthly: €39.29
Azure Cosmos DB		Azure Cosmos DB for MongoDB API		Upfront: €0.00	Monthly: €16.54
Service Bus		Standard tier: Messaging API		Upfront: €0.00	Monthly: €8.68
Storage Accounts		Queue Storage, General Purpose		Upfront: €0.00	Monthly: €7.37
Azure Web PubSub		Standard, 1 Unit(s) x 1 Hours		Upfront: €0.00	Monthly: €56.23
Storage Accounts		Block Blob Storage, General Purpose		Upfront: €0.00	Monthly: €10.58
Key Vault		Vault: 3,000,000 operations, ...		Upfront: €0.00	Monthly: €7.96

Figura 5.21: Riassunto dei costi previsti delle risorse Azure

Il costo totale previsto per fornire il servizio a 5000 utenti ammonta quindi a circa € 150. Viste tutte le stime assunte sull'utilizzo del sistema questa cifra è da considerarsi indicativa. Fornisce però una dimensione iniziale sul costo da prevedere per l'esecuzione del sistema. Come appena presentato, il prezzo può infatti essere correlato al numero di utenti attivi, tenendo però presente la scalabilità specifica di ogni servizio.

L'analisi di questo calcolo evidenzia quanto sia significativo l'impatto di Azure PubSub sul totale. Visto il ruolo nel sistema, il suo utilizzo risulta sicuramente degno di revisione, per trovare soluzioni alternative che possano fornire un servizio di pari qualità, a un prezzo minore.

Conclusione

L'obiettivo di questa tesi era mostrare a che livello e in che misura il requisito della scalabilità e l'utilizzo delle risorse in cloud impattano sulla realizzazione di un'applicazione.

Durante la fase di analisi è stato evidenziato quali fossero i punti critici del progetto. Si sono quindi affiancate a ogni componente scelto le necessità a cui dovevano rispondere, eventualmente approfondendo le molteplici offerte presenti sul mercato. La scelta è stata determinata dalle tecnologie e funzionalità che presentavano, influenzando in maniera decisiva l'approccio da usare nella progettazione del codice. Sono state quindi affiancate le scelte implementative implicate dall'utilizzo dei vari servizi.

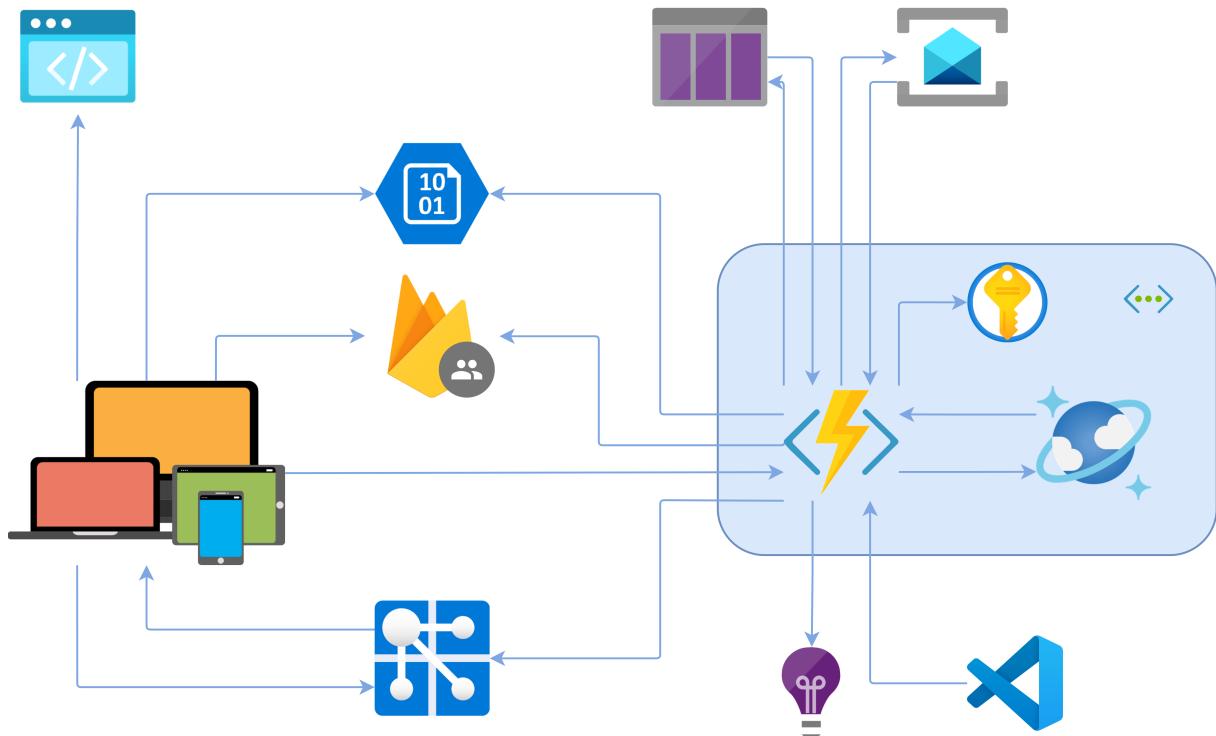


Figura 5.22: Grafico dell'architettura di WYD

La fase di realizzazione ha visto lo sviluppo di un client utente intuitivo e di un server sca-

labile ed efficiente, sfruttando la tecnologia serverless. Una particolare attenzione è stata prestata ai meccanismi di autenticazione, alla sicurezza dei dati e al monitoraggio continuo dei servizi. La gestione della persistenza è stata affrontata attraverso l'implementazione di un database non relazionale distribuito, scelta che ha comportato il principale impatto sulle dinamiche del progetto. Gli è stato inoltre affiancato lo sviluppo di una memoria locale per la cache e la comunicazione in tempo reale, elementi cruciali per la reattività dell'applicativo. Un capitolo significativo è stato dedicato al recupero e al salvataggio dei dati multimediali, essenziali per la funzionalità di condivisione in tempo reale di Wyd.

L'integrazione tra le tecnologie cloud ha richiesto un approccio progettuale orientato alla separazione delle responsabilità, alla gestione della consistenza eventuale e alla resilienza in caso di errori o richieste concorrenti. Ciò ha condotto all'adozione di pattern moderni, come la denormalizzazione, i trigger su modifiche, l'uso di code per la propagazione degli eventi e notifiche in tempo reale attraverso canali dedicati.

I risultati ottenuti durante i test di carico hanno confermato la qualità delle scelte architettoniche adottate: il sistema dimostra di sostenere centinaia di richieste al secondo senza degrado percettibile delle prestazioni, mantenendo al tempo stesso un basso tempo di latenza.

In definitiva, il percorso intrapreso ha mostrato come la realizzazione di un'applicazione distribuita e scalabile non sia il frutto di un processo lineare, ma di un dialogo costante tra le esigenze funzionali, i vincoli tecnologici e le opportunità offerte dai servizi cloud-native. Ogni tecnologia adottata ha influito sulle logiche implementative, imponendo precise scelte progettuali, sia in termini di prestazioni che di costi.

L'auspicio è che le considerazioni sviluppate possano offrire un contributo utile non solo alla valutazione di Wyd come progetto, ma anche come punto di partenza per l'approfondimento dei principi e delle pratiche legate allo sviluppo di applicazioni moderne, condivise e scalabili.

5.8 Sviluppi futuri

Wyd può essere ampliata nelle sue funzionalità, con conseguenze sull’infrastruttura che possono essere nulle o molto impattanti.

L’architettura attuale guadagnerebbe sicuramente in prestazioni dall’introduzione di una cache tra le Azure Functions e il database. È inoltre bene rivedere l’utilizzo delle websocket per le informazioni in tempo reale, considerato l’impatto previsto di Azure PubSub sui costi totali.

L’implementazione di eventi pubblici è sicuramente il principale passo logico successivo per rendere Wyd un’applicazione di ampio utilizzo. Questo comporta la realizzazione di una piattaforma dedicata, che riceva gli eventi e ne gestisca la condivisione, tenendo traccia di tutti i rapporti tra profili che seguono e profili che possono pubblicare.

Con la creazione di eventi pubblici ha senso prevedere un sistema per gestire la prenotazione e l’organizzazione degli eventi, dalle liste di attesa alle vendite dei biglietti. La necessità di garantire la robustezza del servizio, introducendo logiche di controllo dei ruoli e di traffico monetario, comporta requisiti specifici di affidabilità e consistenza a cui l’infrastruttura dovrà rispondere.

Vista la crescente attenzione alla privacy, alla protezione dei dati e alla dipendenza dovuta all’utilizzo di servizi esterni, si prevede di sviluppare un applicativo gemello che possa essere distribuito su un unico server dedicato, per fornire la possibilità a terzi di eseguire su macchine proprie. Il programma vedrebbe quindi un’infrastruttura fisica completamente diversa, ma la sua implementazione continuerebbe a seguire tutti i principi adottati per mantenere il codice scalabile, fornendo così un prodotto comunque resistente a carichi differenti, seppure ridotti. Questo diminuirebbe inoltre il numero di modifiche da introdurre nel codice.

Infine, ulteriori sviluppi potranno comprendere, in base alle necessità degli utenti:

- La visualizzazione degli impegni degli altri profili
- L’implementazione di una chat per ogni gruppo

- 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
- Una funzionalità di ricerca degli eventi o dei profili pubblici

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>

<https://learn.microsoft.com/en-us/azure/reliability/reliability-cosmos-db-nosql> <https://learn.microsoft.com/gb/azure/cosmos-db/throughput-serverless> <https://learn.microsoft.com/en-gb/azure/cosmos-db/provision-throughput-autoscale> <https://learn.microsoft.com/en-us/azure/azure-functions/functions-dotnet-dependency-injection> <https://azure.microsoft.com/en-us/pricing/calculator/>