



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 multiplatforma
per l'organizzazione di eventi condivisi e
la condivisione multimediale automatica
in tempo reale

Relatore:
Chiar.mo Prof.
Michele Colajanni

Presentata da:
Giacomo Romanini

Sessione Luglio 2025

Anno Accademico 2025/2026

Abstract

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

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

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

Indice

Introduzione	1
Organizzazione dei capitoli	3
0.1 L'implementazione del database	4
0.1.1 La scelta del database	5
0.1.2 La definizione delle classi	11
0.1.3 L'integrazione con le Azure Functions: il framework .Net e l'eventual consistency	11
0.1.4 L' integrazione con C#	12

Introduzione

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

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

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

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

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

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

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



Figura 1: Il logo di Wyd

La realizzazione di un progetto come Wyd implica la risoluzione e la gestione di diverse problematiche tecniche. In primo luogo, la stabilità del programma deve essere garantita da un'infrastruttura affidabile e scalabile. La persistenza deve essere modellata per fornire alte prestazioni sia in lettura che in scrittura indipendentemente dalla quantità delle richieste, rimanendo però aggiornata e coerente. La funzionalità di condivisione degli eventi richiede inoltre l'aggiornamento in tempo reale verso tutti gli utenti coinvolti. Infine, il caricamento ed il salvataggio delle foto aggiungono la necessità di gestire richieste di archiviazione di dimensioni significative.

Organizzazione dei capitoli

Il seguente elaborato è suddiviso in cinque capitoli.

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

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

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

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

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

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

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

database nel progetto.

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

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

0.1.1 La 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. Questo è un vantaggio fondamentale in architetture distribuite e ambienti ad alta intensità di dati. La scalabilità è realizzata attraverso la partizionamento (o sharding), un meccanismo che distribuisce automaticamente i dati tra diversi nodi di archiviazione fisici.

Attraverso la denormalizzazione si possono migliorare le prestazioni di lettura, aggregando dati correlati all'interno di un singolo documento o partizione. Questo approccio semplifica la gestione delle join, che possono essere ottimizzate per risiedere all'interno della stessa partizione o in partizioni vicine, minimizzando la necessità di operazioni di lettura tra nodi distinti.

La denormalizzazione comporta intrinsecamente alcune sfide a livello di consistenza dei dati, in particolare per le operazioni di recupero che coinvolgono dati potenzialmente duplicati. Questo problema è efficacemente mitigato dall'adozione di Global Secondary Indexes (GSI). I GSI consentono di interrogare i dati su attributi che non sono la chiave primaria del documento, e risiedono quindi su partizioni differenti. Forniscono percorsi di accesso alternativi e performanti sull'intero dataset distribuito, superando le limitazioni imposte dalla distribuzione fisica delle partizioni e mantenendo un'elevata efficienza nelle query complesse.

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 un lato della relazione, mentre i Global Secondary Indexes (GSI) superano le limitazioni delle partizioni fisiche, consentendo interrogazioni efficienti e performanti dall'altro lato. 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 loro architettura, che spesso lega indici e tabelle alla stessa partizione logica, impone intrinsecamente dei vincoli sulla scalabilità orizzontale,

non solo per la difficoltà di distribuire indici e tabelle su nodi diversi, ma anche per il numero massimo di connessioni contemporanee che possono gestire, limitando la capacità di rispondere a un numero massiccio di richieste simultanee. L'implementazione dello sharding, sebbene possibile, è interamente a carico dello sviluppatore, introducendo un significativo onere di progettazione, sviluppo e manutenzione.

Inoltre, la gestione di relazioni molti-a-molti nel modello relazionale, pur essendo logicamente chiara con tabelle di giunzione, può presentare problemi di scalabilità in contesti di elevato carico. Quando una query necessita di recuperare dati da entrambi i lati di una relazione molti-a-molti, ciò implica l'esecuzione di join complesse tra più tabelle. Sebbene queste operazioni possano essere veloci su singole istanze di database ben ottimizzate, in un ambiente distribuito e con volumi di dati in crescita, queste join possono richiedere il trasferimento di grandi quantità di dati tra nodi diversi per essere risolte, introducendo latenza e overhead di rete significativi. Questo può diventare un collo di bottiglia, compromettendo le performance complessive. Infine, la mancanza di un supporto nativo per indici secondari globali (GSI) rende più complessa la gestione di query su attributi non chiave distribuiti su diverse partizioni, costringendo a soluzioni alternative che potrebbero compromettere la reattività del sistema e aumentare la complessità del codice. 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.

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 essere presentare un'architettura distribuita, con replica automatica dei dati, assicurando un'elevatissima disponibilità e resilienza. Queste





















INDICE

vengono assicurate anche in caso di interruzioni regionali, grazie a meccanismi di failover automatico. Inoltre, la distribuzione globale "turnkey" garantisce che i dati siano sempre vicini agli utenti, riducendo drasticamente la latenza a millisecondi a cifra singola (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.

Pricing : All Operating System : All Publisher Type : All Product Type : All Publisher name : All

☐ Azure services only

Showing 1 to 20 of 244 results for 'document database'. [Clear search](#)

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 Deploy...	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	Apps4Rent 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	Nuvento LLC	Gold	
 MongoDB 6 on Red Hat Enterprise Linux 9	AskforCloud LLC	MongoDB 6 on Red Hat Enterprise Linux 9	€0.035/hour

[Previous](#) Page 1 of 13 [Next](#)

Figura 2: Proposte di Azure per i database documentali

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.



Azure Cosmos

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 Rerquest 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.

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 1: 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 e 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, in una fase successiva del progetto, In caso ulteriori analisi svolte successivamente nel progetto, (grazie ai dati dell'utilizzo effettivo) non identifichino questa soluzione come la più adatta,

"Qualora, in seguito a ulteriori analisi condotte nelle fasi successive del progetto — rese possibili dai dati derivanti dall'utilizzo effettivo del sistema — emergesse che la soluzione adottata non rappresenta l'opzione più adeguata, sarà possibile valutarne l'evoluzione o la sostituzione con alternative più rispondenti ai requisiti osservati."

lo spostamento dei dati verso un altro gestore richiede uno sforzo limitato, data la compatibilità nella rappresentazione dei dati tra le diverse tecnologie di database documentali.

impostazioni di cosmos

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

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

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

Interporre una cache tra la logica applicativa e il database semplifica e riduce il numero di richieste verso il database. Il livello di caching si occupa di gestire le richieste al database fornendo e duplicando le risposte che possiede già in memoria, eventualmente concentrando le richieste in caso i dati siano invece da recuperare. Per i dati in scrittura, invece, salva temporaneamente le modifiche richieste, aggiornando subito la memoria locale, per poi apportare le modifiche al database in momenti di carico ridotto. Garantisce così un

tempo di risposta e di propagazione degli aggiornamenti ridotto, alleviando il numero di richieste al database, estendendo così le prestazioni fornite.

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

0.1.2 La definizione delle classi

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

0.1.3 L'integrazione con le Azure Functions: il framework .Net e l'eventual consistency

0.1.4 L' integrazione con C#

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

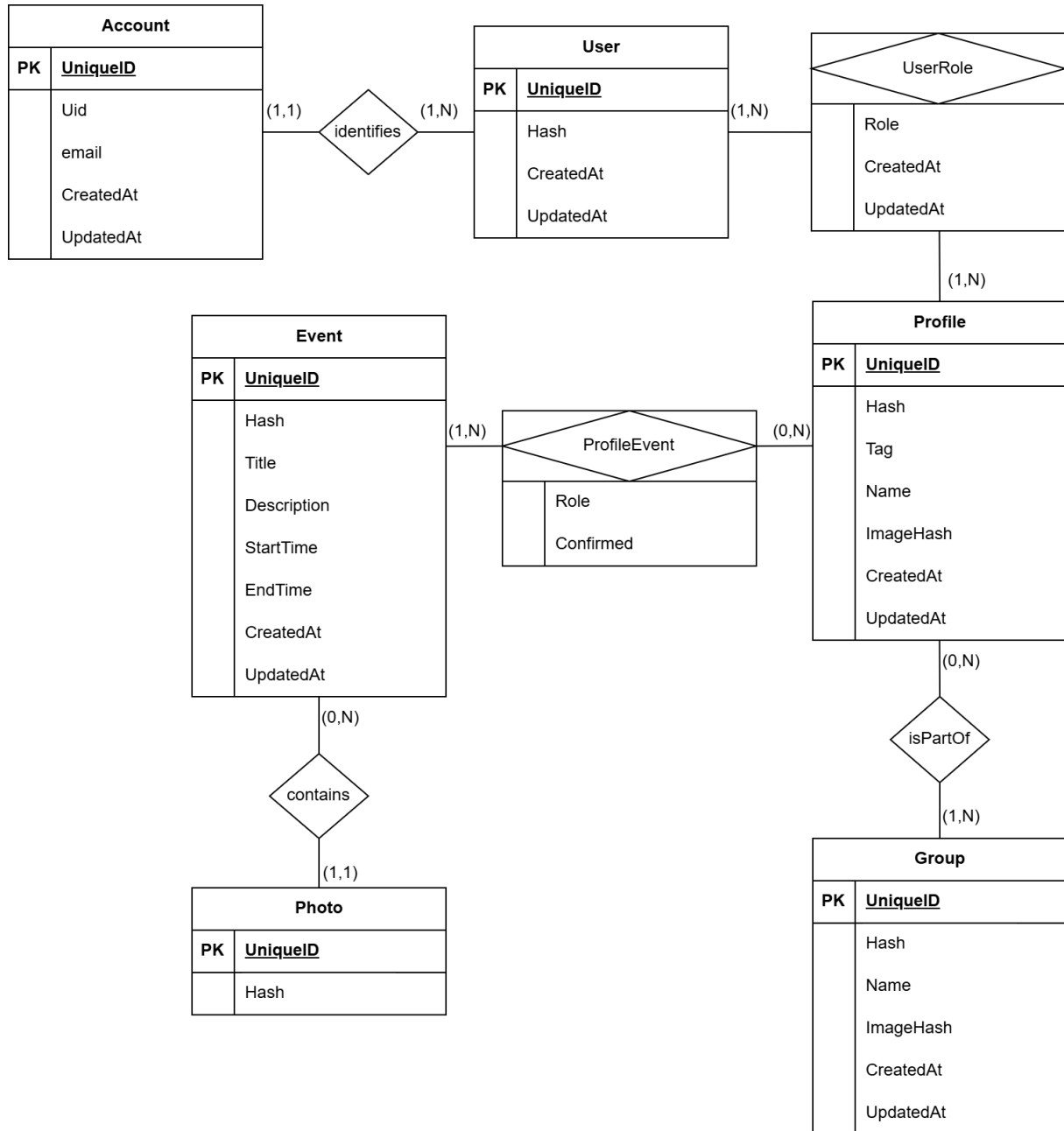


Figura 3: Diagramma Entità - Relazione del dominio

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

INDICE

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

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

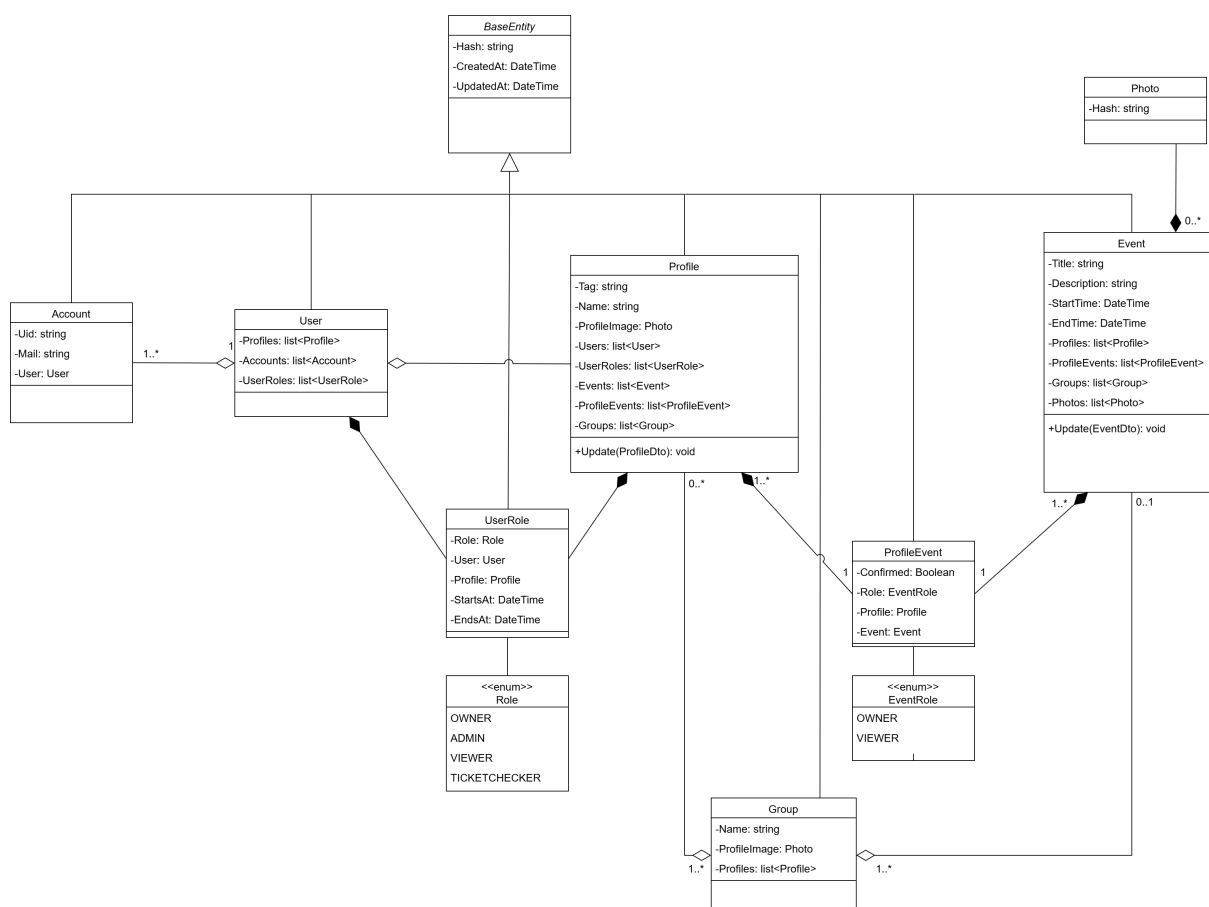


Figura 4: Modello delle classi del client

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