Università di Catania

DISTRIBUTED SYSTEMS AND BIG DATA
A.A. 2025/26

Homework 1 DSBD

STUDENTI:

Alessia Fichera 1000084787

Davide Pantò 1000081854

Abstract

L’applicazione utilizza un’architettura a microservizi per garantire scalabilità e resilienza. Tra le funzionalità principali:

- gestione utenti in tempo reale (autenticazione, profili, sessioni e autorizzazioni);

- integrazione con l’API openSky Network per il recupero dei dati di volo in tempo reale.

L’utente comunica al Data Collector uno o più aeroporti di interesse, con la possibilità di modificare o aggiornare elenco di suo interesse in qualsiasi momento.

Il sistema è composto da:

• **Comunicazione gRPC (User Manager ↔ Data Collector):** I Client gRPC fungono da intermediari tra i due microservizi gestendo la comunicazione verso due server distinti; il primo verifica l'autenticazione dell'utente prima di processare le richieste sui voli, mentre il secondo si occupa della rimozione degli interessi associati a un utente specifico quando quest’ultimo viene cancellato dalla tabella Users.

• **Data Collector:** è un servizio che raccoglie in modo continuativo le informazioni sui voli dall’API esterna OpenSky Network, aggiornando nelle tabelle del DataDB. I voli in arrivo vengono salvati nella tabella **“Flight_Data_Arrives”**, mentre quelli in partenza nella tabella **“Flight_Data_Departures”**. Il servizio integra due thread:

- **Il primo**, ogni 12 ore, monitora i dati provenienti da OpenSky Network e inserisce o aggiorna le informazioni nelle due tabelle.
- Il **secondo** elimina periodicamente i record più vecchi di 10 giorni.

Questo disaccoppiamento permette all'utente di avere risposte rapide (leggendo dal DB locale) senza dover attendere la chiamata API verso OpenSky in tempo reale.

• **Database MySQL**: Sono stati implementati due database distinti.

- Il primo, denominato **UserDB**, è dedicato alla gestione delle informazioni relative agli utenti. Al suo interno sono presenti le tabelle *Users*, che archivia le credenziali degli utenti, Logged_Users, che mantiene traccia degli utenti autenticati, e una tabella di cache utilizzata per supportare il meccanismo di At-Most-Once, evitando l’esecuzione duplicata delle operazioni. A tal fine è implementato un thread che elimina periodicamente dalla cache le richieste più vecchie.
- Il secondo database, denominato **DataDB**, è invece finalizzato alla memorizzazione dei dati relativi ai voli. Contiene le tabelle Flight_Data_Arrives per i voli in arrivo, Flight_Data_Departures per i voli in partenza e *Interessi*, che registra gli aeroporti di interesse associati ai vari utenti.

Per implementare la semantica **At-Most-Once**, oltre alla persistenza nella tabella Users, viene utilizzato **Redis** come sistema di deduplicazione a bassa latenza. Al completamento con successo della richiesta di registrazione oltre al salvataggio nella tabella Users , il sistema inserisce in Redis una coppia *key–value*, dove:

- **key**: hash dell'email dell'utente (SHA-256)
- **value**: username associato
- **TTL**: 60 secondi

L'inserimento garantisce atomicamente che la chiave venga creata solo se non esiste già, consentendo un controllo efficiente delle operazioni duplicate. Durante un eventuale ritentativo della stessa richiesta, il sistema esegue una lettura in Redis (usando la key). Se la chiave è presente, la richiesta viene riconosciuta come duplicata e non viene effettuata alcuna interrogazione sulla tabella Users, evitando così accessi DB ridondanti.La gestione del TTL garantisce che Redis mantenga solo le richieste recenti, limitando l'utilizzo di memoria. Poiché la cache contiene esclusivamente chiavi temporanee relative a operazioni in corso, il consumo di memoria rimane molto ridotto rispetto al dataset persistente presente nella tabella Users, assicurando una latenza minima nell'identificazione dei duplicati.Questo approccio fornisce un meccanismo leggero ed efficiente per supportare la semantica At-Most-Once, riducendo il carico sul database e impedendo la creazione di record duplicati.

Diagramma architetturale:

Il diagramma rappresenta l'architettura dell'applicazione, mostrando le principali interazioni tra i microservizi:

- **Data Collector**: Micro Servizio che raccoglie in modo continuativo le informazioni sui voli dall'API esterna OpenSky Network, aggiornando nelle tabelle del DataDB.
- **User Manager** :Micro servizio che gestisce le richieste di registrazione, login e cancellazione dell'utente nello UserDB.
- **gRPC Service User**: Verifica l'autenticazione prima di processare richieste sui voli.
- **gRPC Service Data**: Rimuove gli interessi dell'utente quando viene cancellato dal sistema.
- **DataDB**: Contiene le tabelle con le informazioni sui voli.
- **UserDB**: Contiene le tabelle con le informazioni sugli utenti.
- **Redis:** funge da cache temporanea per intercettare i tentativi duplicati di registrazione ed evitare interrogazioni inutili al database.

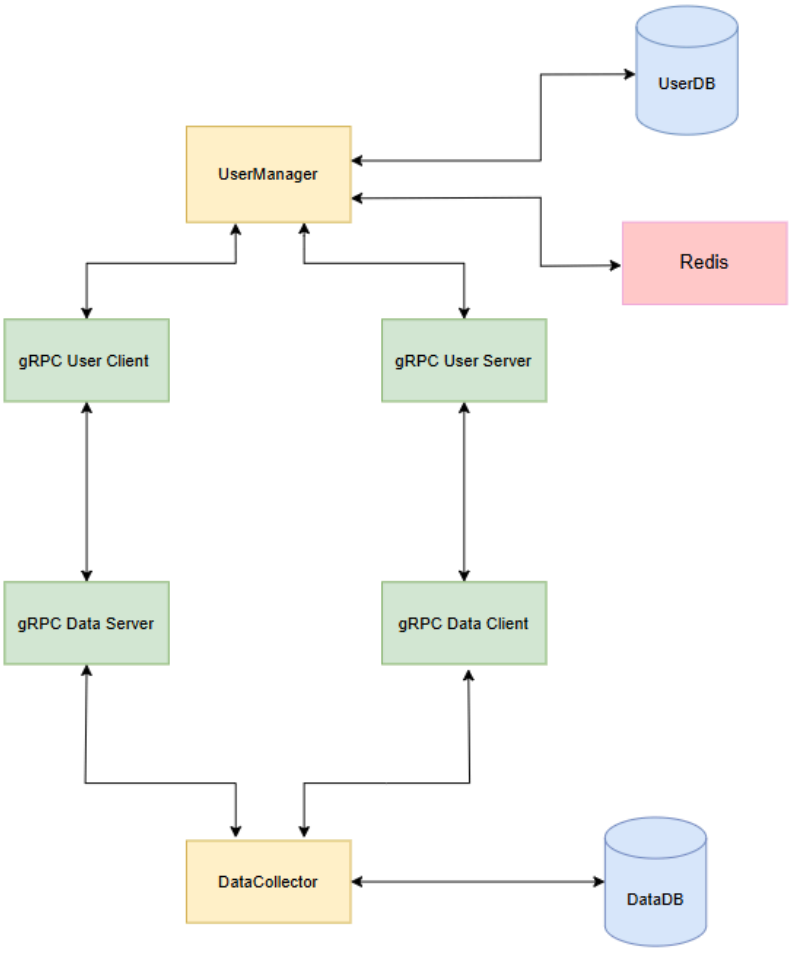
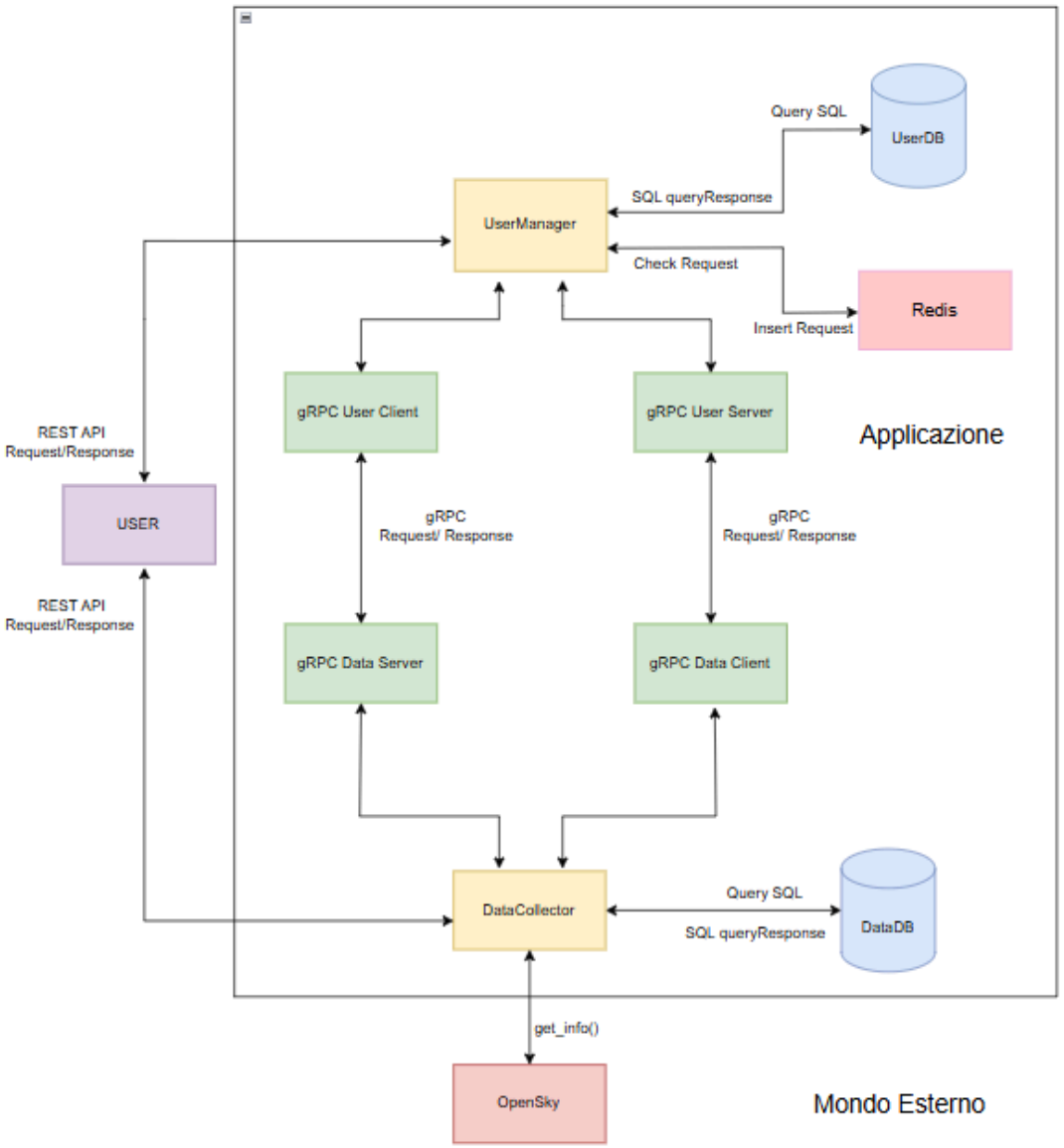


Diagramma delle Interazioni

Il diagramma mostra un'architettura a microservizi composta da **UserManager** e **DataCollector**, che espongono interfacce **REST API** verso l'utente (mondo esterno) e comunicano tra loro tramite protocollo **gRPC**.

- **Comunicazione gRPC (Bidirezionale):**
  - **User Client → Data Server:** Lo User Manager invia comandi al Data Collector (cancellazione dati).
  - **Data Client → User Server:** Il Data Collector interroga lo User Manager (verifica autenticazione).
- **Database Dedicati:** Ogni servizio gestisce il proprio archivio (**UserDB** e **DataDB**) tramite query SQL.
- **OpenSky:** Servizio esterno interrogato dal DataCollector per reperire le informazioni sui voli in tempo reale.
- **Redis:** funge da cache temporanea per intercettare i tentativi duplicati di registrazione ed evitare interrogazioni inutili al database.



Lista delle API Data Collector:

Nome API	Descrizione	Messaggio di Richiesta	Messaggio di Risposta
sendInterest()	Registra l'interesse di un utente per un aeroporto e una modalità* specificata.	**SendInterestRequest** email (string) token (string) airport_code (string) mode (boolean)	**SendInterestResponse** message (string)
delete_interest()	Rimuove l'interesse registrato di un utente.	**DeleteInterestRequest** email (string) token (string) airport_code (string) mode (boolean)	**DeleteInterestResponse** message (string)
get_info()	Recupera le informazioni sui voli per un determinato aeroporto e modalità*.	**GetInfoRequest** email (string) token (string) airport_code (string) mode (boolean)	**GetInfoResponse** count (int) voli (list of objects): partenza (string) ora_arrivo (string) ora_partenza (string) arrivo (string) codice (string)
get_last_one()	Recupera l'ultimo volo in arrivo e in partenza disponibile per l'aeroporto.	**GetLastValueRequest** email (string) token (string) airport_code (string)	**GetLastValueResponse** count (int) voli (list of objects): partenza (string) ora_arrivo (string) ora_partenza (string) arrivo (string) codice (string)
get_avgs()	Calcola la media dei voli (arrivi e partenze) negli ultimi N giorni.	**GetAveragesRequest** email (string) token (string) airport_code (string) n_days (int/string)	**GetAveragesResponse** media arrivi (double) media partenze (double)

*modalità: si intende aeroporto di arrivo o di partenza.

Lista delle API User Manager:

Nome API	Descrizione	Messaggio di Richiesta	Messaggio di Risposta
login()	Esegue l'autenticazione di un utente verificando le credenziali (email e password).	**LoginRequest** email (string) password (string)	**LoginResponse** message (string)
registrazione()	Registra un nuovo utente nel sistema salvando email, username e password (con hashing).	**RegistrationRequest** email (string) username (string) password (string)	**RegistrationResponse** message (string)
cancellazione()	Elimina definitivamente un account utente e i relativi dati (sessione e interessi), previa verifica della password.	**DeleteAccountRequest** email (string) password (string)	**DeleteAccountResponse** message (string)