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 database. 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.
- I**l 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.

Avendo implementato la politica At-Most-Once per gestire le richieste in modo sicuro.Quando il client invia una richiesta di registrazione e questa va a buon fine,le informazioni dell'utente vengono salvate nella tabella Users, mentre nella tabella cache viene memorizzata solo la richiesta con gli attributi email hash e timestamp. In caso di ri-tentativo, il sistema verifica se l'email hash è già presente nella cache senza interrogare nuovamente la tabella Users. Il sistema utilizza inoltre un thread che periodicamente cancella dalla cache le richieste vecchie di 1 minuto basandosi sul timestamp. Poiché la cache contiene solo le richieste recenti, le sue dimensioni sono significativamente inferiori rispetto alla tabella Users, riducendo l'overhead di memoria e garantendo un controllo rapido delle operazioni duplicate. Questo meccanismo assicura che ogni operazione venga eseguita al massimo una volta, evitando duplicazioni e spreco di risorse dovuto a interrogazioni ripetute alla tabella Users.

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.

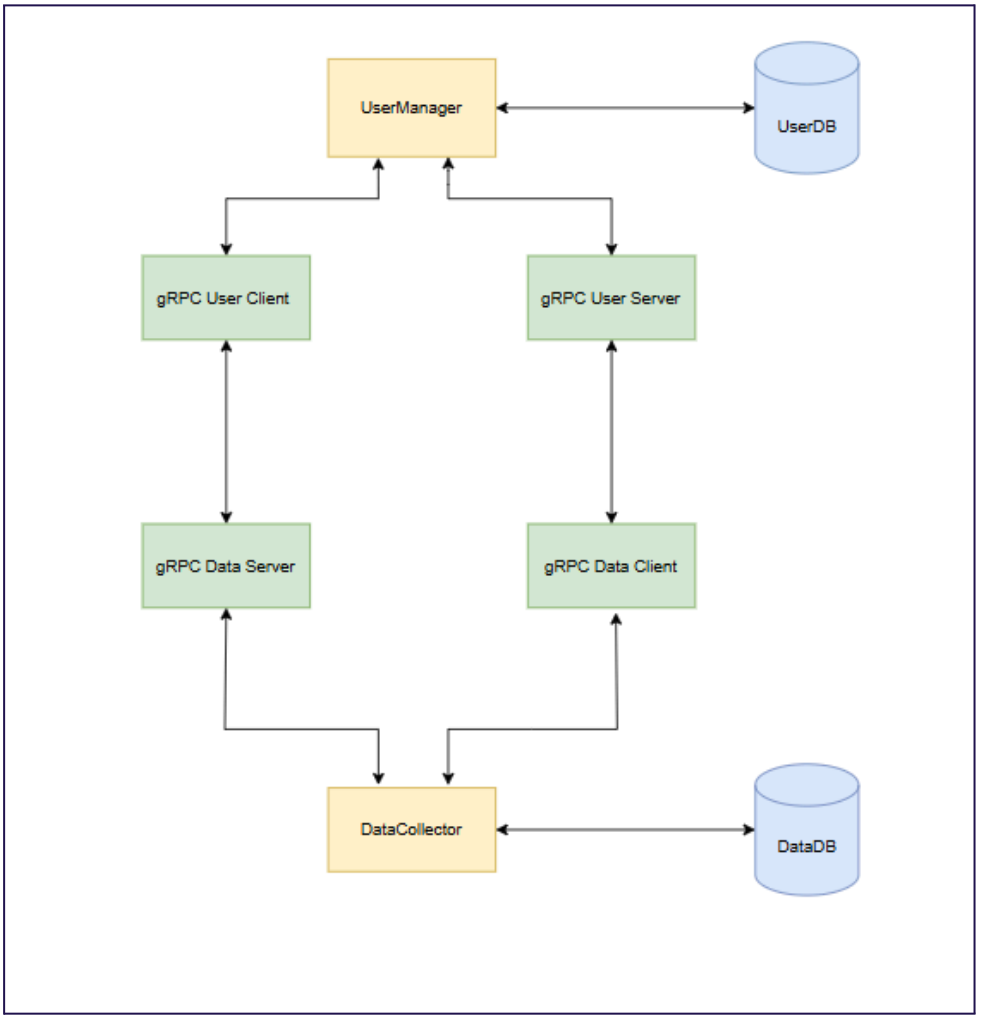
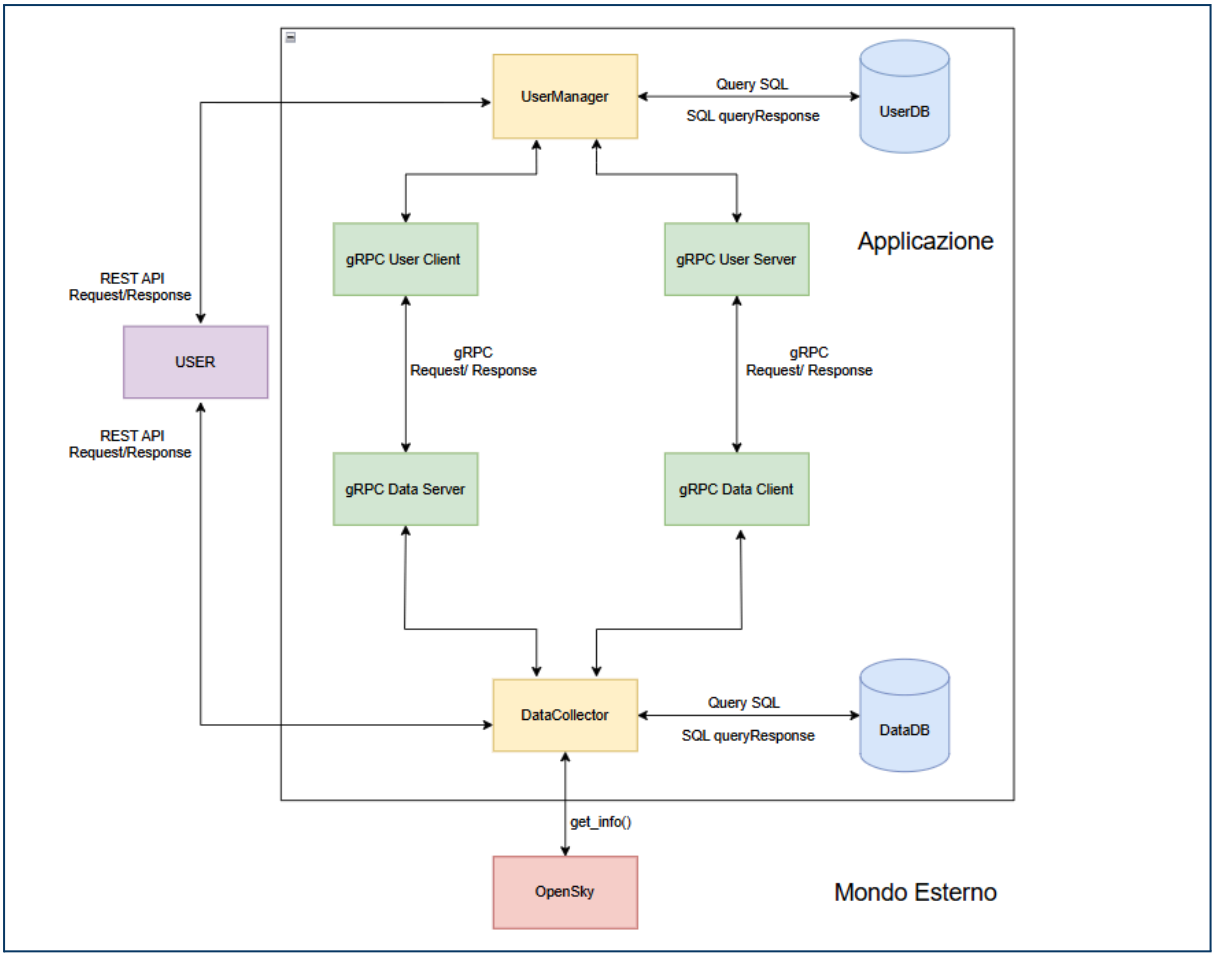


Diagramma delle Interazioni

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.



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)