Relazione homework 1

Florio Gabriele – Di Rosolini Enricomaria

Descrizione

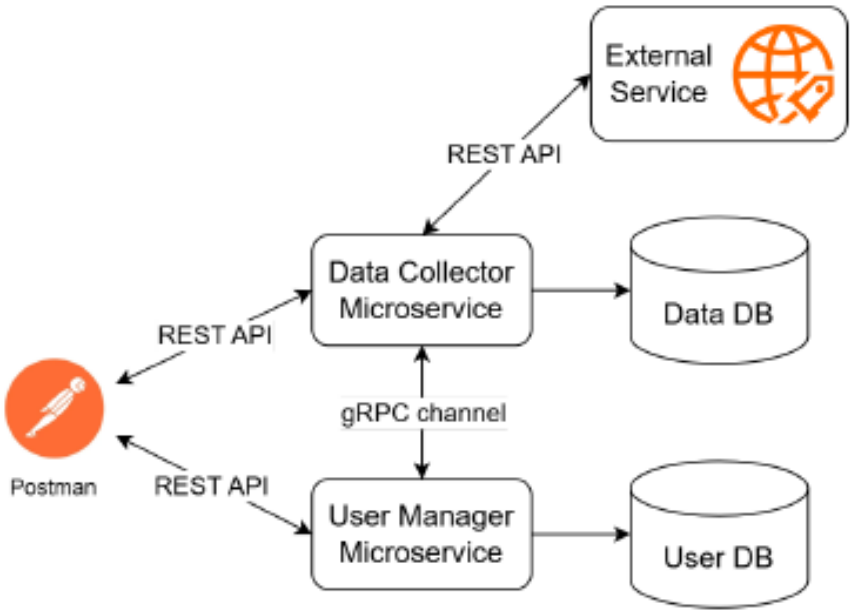
L'applicazione rappresenta lo sviluppo di un sistema distribuito composto da due microservizi, uno che gestisce gli utenti e l'altro che gestisce le informazioni riguardanti gli aeroporti/voli integrando un servizio esterno denominato OpenSky Netowork per recuperare i dati in tempo reale.

Architettura

L'architettura adottata segue il pattern a microservizi dockerizzati, dove ognuno di essi ha una responsabilità ben definita.

Il sistema è suddiviso nei seguenti blocchi:

- **User manager microservice:** espone API REST per la gestione degli utenti, come la registrazione, la cancellazione o l'ottenimento delle relative informazioni.
- **Data collector microservice:** espone API REST per la gestione delle informazioni sui voli degli aeroporti di interesse degli utenti. Inoltre, esegue dei task periodici per scaricare dati da OpenSky Network.
- **Database:** sono stati utilizzati due database separati, uno per la gestione delle informazioni degli utenti, uno per la gestione dei dati dei voli/aeroporti, in modo tale da garantire il disaccoppiamento dei dati tra i servizi.



Scelte progettuali

Per quanto riguarda la comunicazione, come si evince dal diagramma è stato adottato un approccio ibrido:

- **REST API:** utilizzato per le iterazioni tra client e i microservizi, in quanto molto semplice e ottimizzato nell'esporre risorse web.
- **gRPC:** utilizzato per la comunicazione tra i microservizi, tra il data collector e lo user manager. La scelta del gRPC è dovuta al fatto che offre delle prestazioni migliori, con una bassa latenza. Una delle scelte progettuali più significative riguarda la gestione della comunicazione gRPC. Sebbene il Data Collector debba interrogare lo User Manager per verificare l'esistenza degli utenti, il sistema è stato progettato implementando **due canali gRPC distinti**, rendendo di fatto lo User Manager un componente ibrido che agisce sia da Server che da Client a seconda del contesto.
    - **Canale 1: Data Collector (Client) - User Manager (Server) – Check user:** questo è il canale standard richiesto dalle specifiche. Quando un utente richiede di tracciare un aeroporto, il Data Collector invoca una procedura remota sullo User Manager per validare l'esistenza dell'utente prima di salvare l'interesse.
    - **Canale 2: User Manager (Client) - Data Collector (Server) - Gestione della Cancellazione:** è stato implementato un secondo canale inverso per gestire la cancellazione dell'utente. Quando viene invocata l'API DELETE /users, lo User Manager (che qui agisce da client gRPC) notifica immediatamente il Data Collector affinché rimuova tutti i dati associati a quell'utente. Questa scelta è stata preferita rispetto ad un approccio passivo (ogni qualvolta il database viene aggiornato venga controllato se effettivamente l'utente che ha mostrato interesse per uno o più aeroporti sia ancora registrato, in caso contrario vengono eliminati gli interessi) per avere una maggiore consistenza immediata tra i due microservizi, evitando
    - chiamate esterne per utenti cancellati, disaccoppiando quindi la logica di business dai cicli periodici di aggiornamento.

User manager

Per il microservizio user manager sono state adottate le seguenti scelte progettuali:

1. **Politica At-Most-Once:** Il sistema garantisce che una richiesta di registrazione venga elaborata al massimo una volta, in caso di richieste duplicate o ritrasmissioni, il sistema non crea duplicati ma restituisce lo stato coerente della risorsa. Questo viene gestito tramite la memoria cache all'interno della quale vengono conservati gli id delle richieste soddisfatte, conservando, inoltre, l'esito e il timestamp dell'operazione.
2. **Database MySQL**: Per la persistenza dei dati dello User Manager è stato selezionato un database relazionale MySQL. La scelta è motivata dalla natura dei dati trattati:

a. **Struttura definita**: Le informazioni utente (email, password, dati anagrafici) seguono uno schema rigido e predefinito.
b. **Integrità referenziale e Vincoli**: MySQL permette di definire vincoli forti (come PRIMARY KEY sull'email) che impediscono la duplicazione dei record, offrendo un ulteriore livello di sicurezza per la politica *at-most-once*.
c. **Transazionalità (ACID)**: Garantisce che le operazioni di registrazione e cancellazione siano atomiche e consistenti.

Data Collector

Per il microservizio data collector sono state adottare le seguenti scelte progettuali:

1. **Scheduler:** è stato implementato uno scheduler, quindi un processo parallelo che interroga ciclicamente con un intervallo di 12 ore OpenSky Network per gli aeroporti monitorati dagli utenti e salva i risultati all'interno del database.
2. **Database:** Per la persistenza dei dati si è optato per un database NoSQL, in particolare MongoDB per i seguenti motivi:
   a. **Tipo di informazioni:** i dati recuperati dalle API REST di OpenSky Network sono in formato JSON, quindi l'utilizzo di MongoDB permette di memorizzarli in maniera rapida e senza la necessità di effettuare delle operazioni di trasformazioni in tabelle relazioni.
   b. **Flessibilità:** a differenza di MySQL, MongoDB non avendo una struttura rigida permette di gestire qualunque tipo di variazione dei dati sui voli senza la necessità di modificare la struttura del database.
   c. **Velocità di scrittura:** poiché il data collector opera periodicamente e potrebbe generare una quantità di dati elevata, l'utilizzo di MongoDB facilita e accelera la scrittura all'interno del DB

   Nel microservizio user-manager serviva rigore, qui si predilige la flessibilità e la velocità di scrittura.

Lista API

User manager

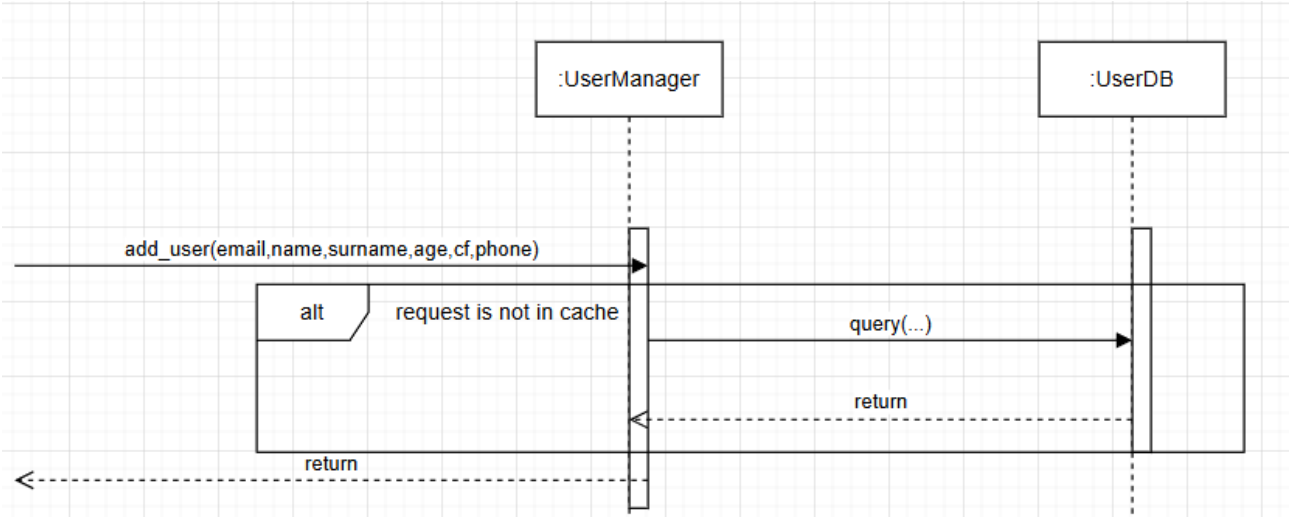
Metodo	Endpoint	Descrizione	Body (JSON)
POST	/add_user	Registrazione nuovo utente	{ "request_id": ..., "email" : "...", "name" : "...", "surname": "...", "age" : ..., "CF" : "...", "phone": "..." }
POST	/rmv_user	Rimuovi utente	{ "email" : "..." }
POST	/get_user	Ottieni informazioni utente	{ "email" : "..." }

Data-collector

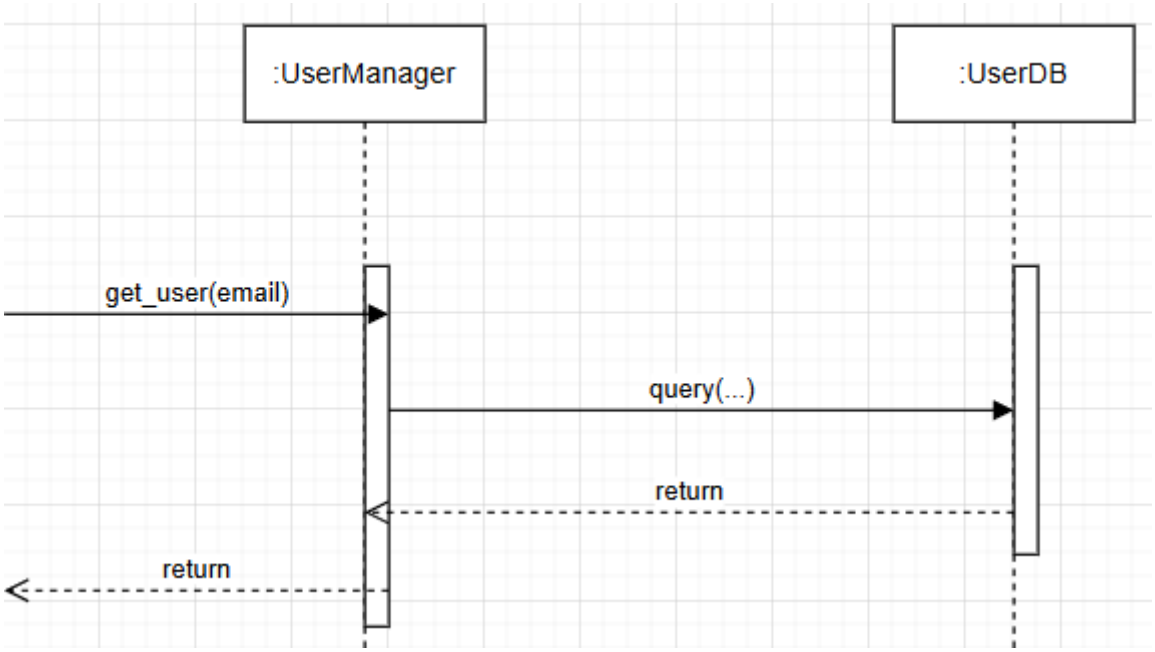
Metodo	Endpoint	Descrizione	Body (JSON)
POST	/add_interest	Aggiunge un nuovo interesse	{"email" : "...", "airport_code": "..." }
POST	/rmv_interest	Rimuove un interesse	{"email" : "...", "airport_code": "..." }
POST	/list_interest	Mostra la lista degli interessi dell'ut.	{"email" : "..." }
POST	/get_flight	Ottiene i voli in arrivo e in partenza dall'aeroporto	{"email" : "...", "airport_code": "..." }
POST	/force_update	Forza l'aggiornamento del db	//
POST	/get_last_flight	Ottiene l'ultimo volo in arrivo e in partenza dall'aeroporto	{"email" : "...", "airport_code": "..." }
POST	/average	Calcola la media voli degli ultimi x giorni	{"email" : "...", "airport_code": "..." "days": ...}
POST	/get_arrivals	Ottiene i voli in arrivo che partono da un determinato aeroporto	{"email" : "...", "airport_code_arrivals":"...", "airport_code_departures":"..." }

DIAGRAMMA DELLE INTERAZIONI

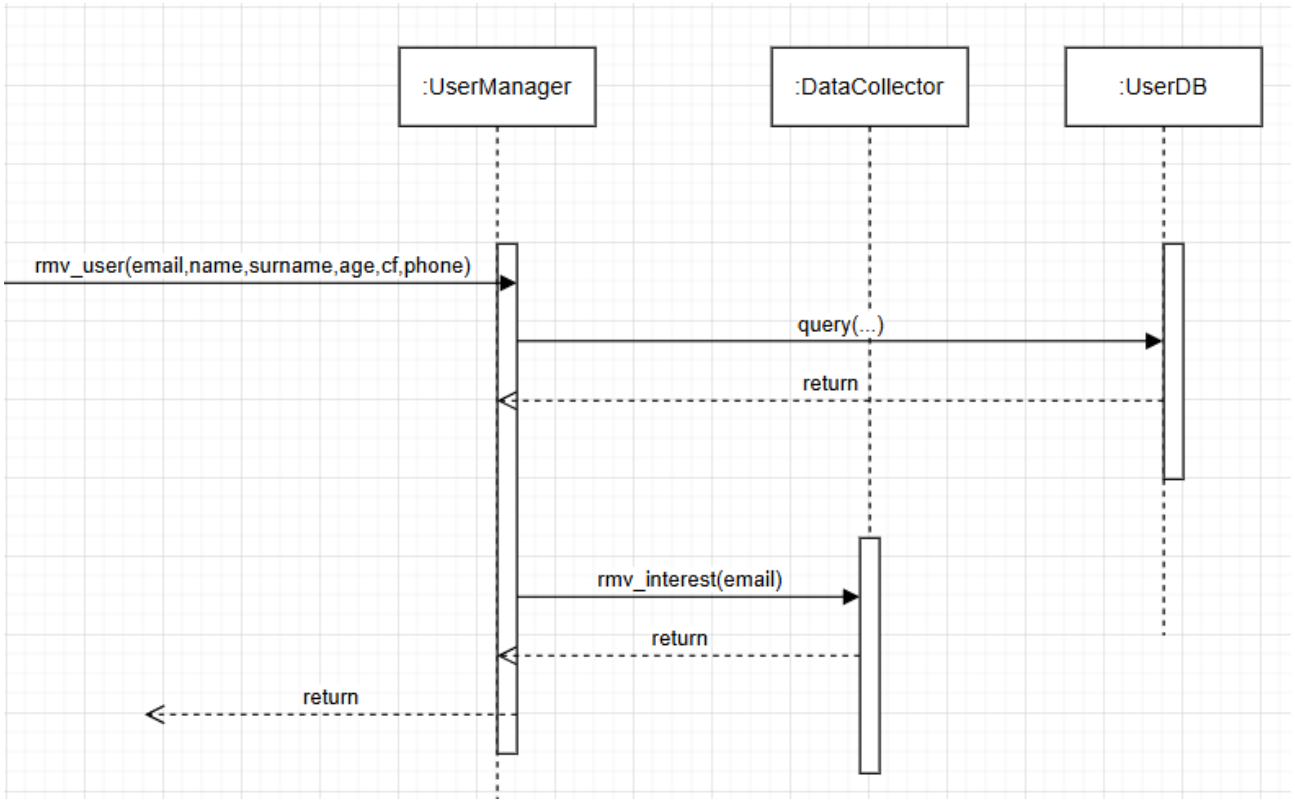
Add_user



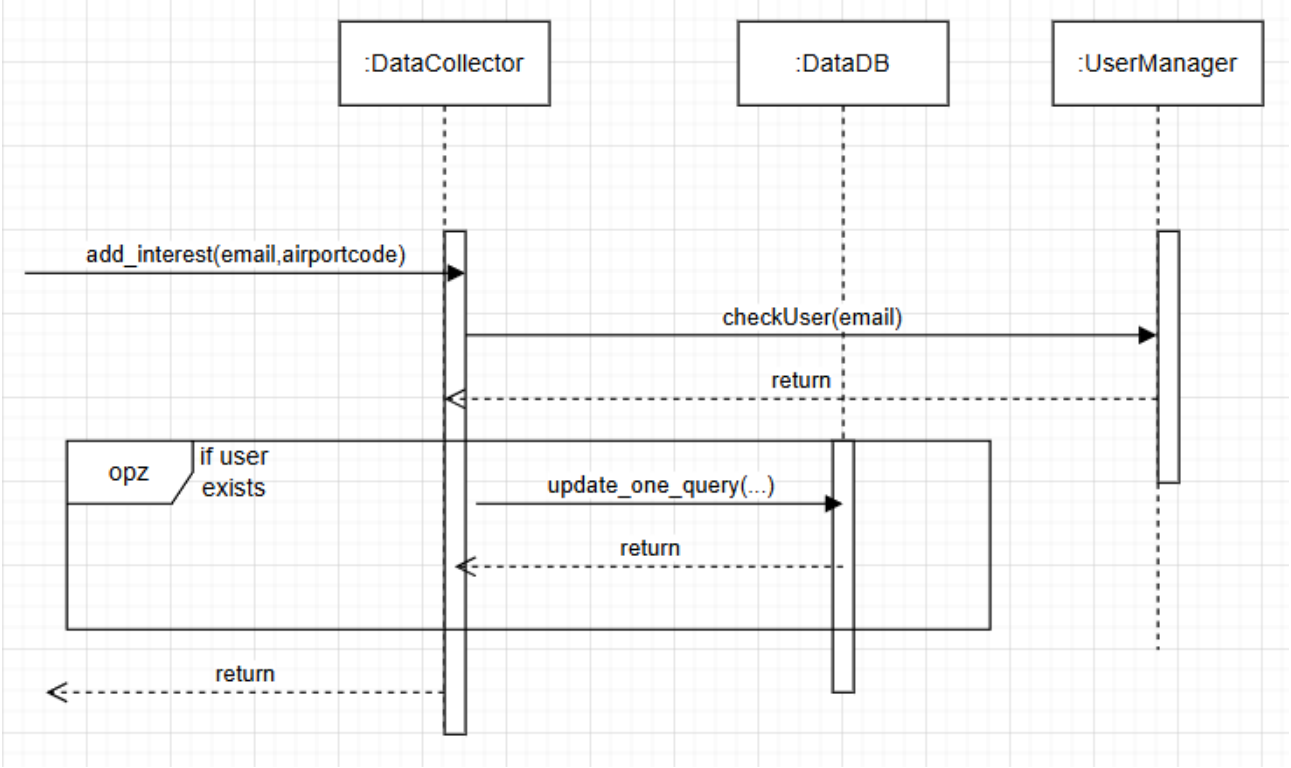
Get_user



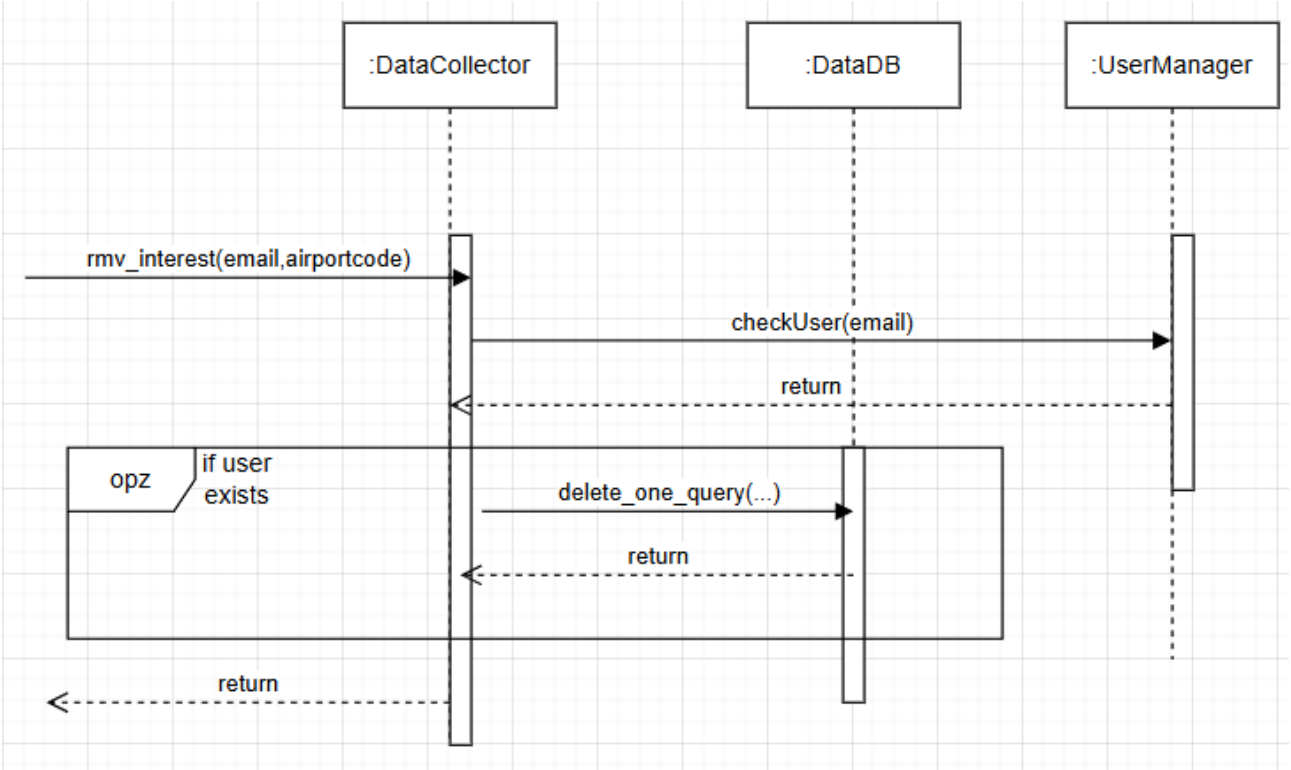
Rmv_user



Add_interest



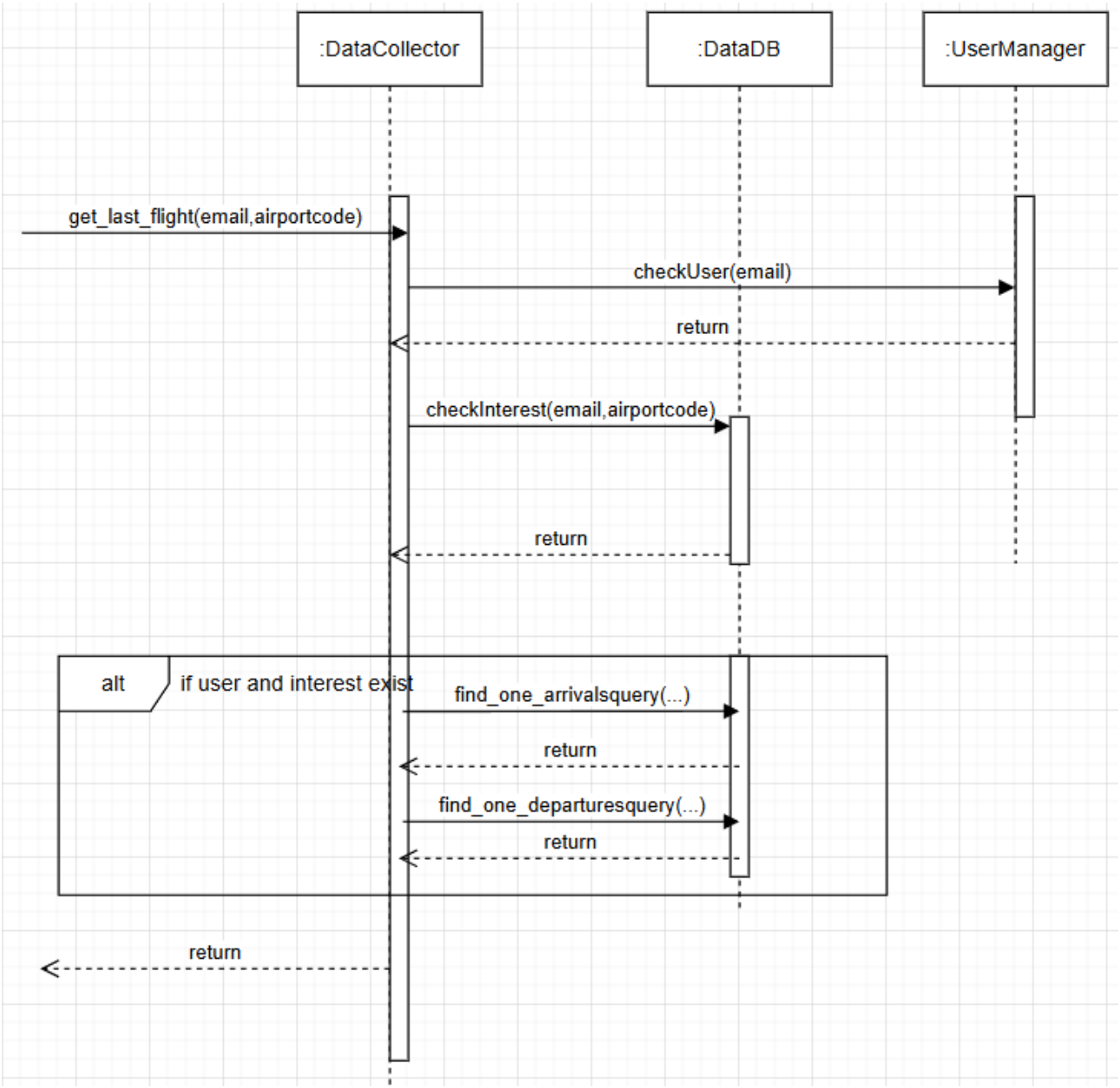
Rmv_interest



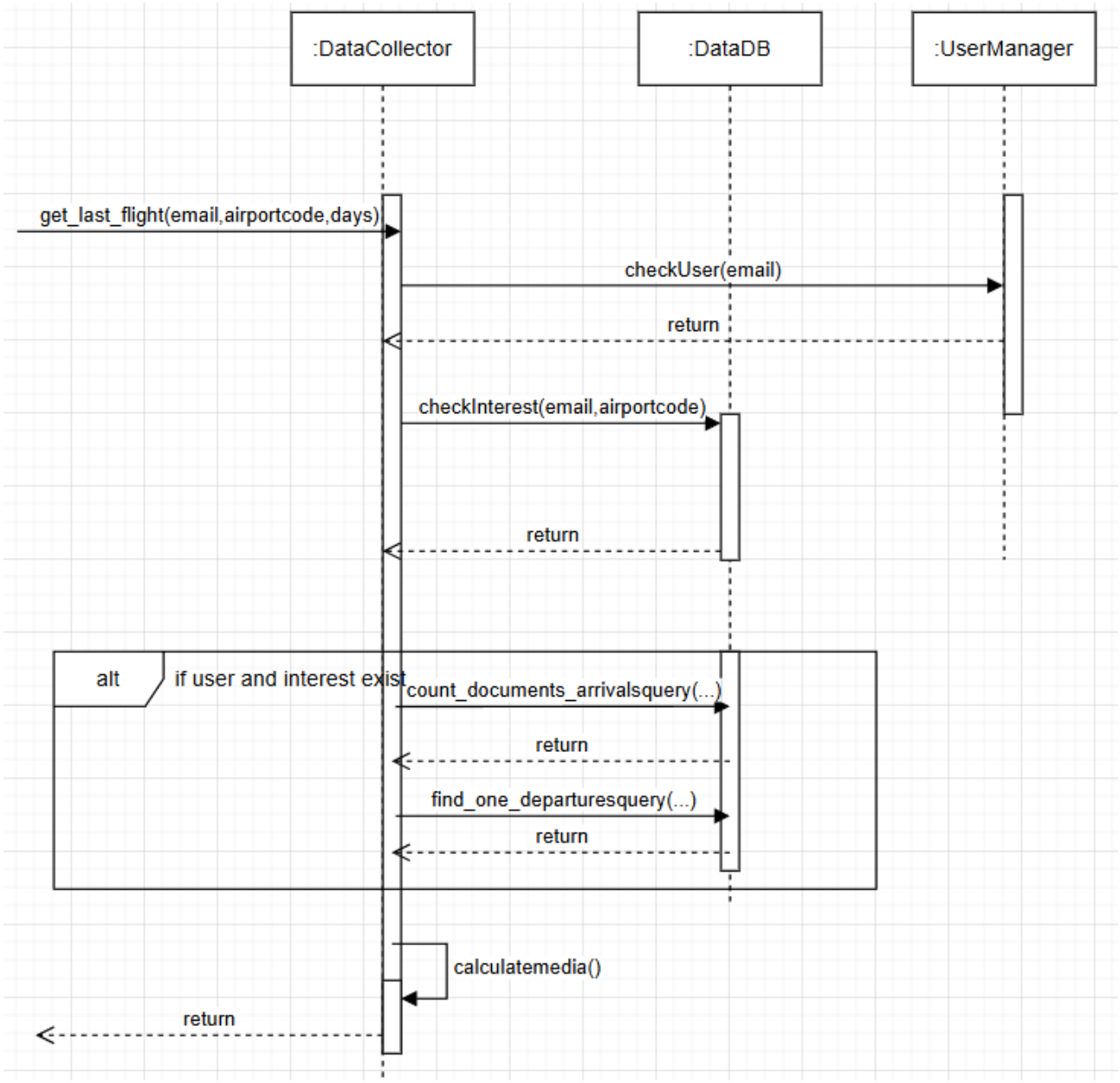
List_interest

Diagramma simile ai precedenti. Cambia solo la query con il db.

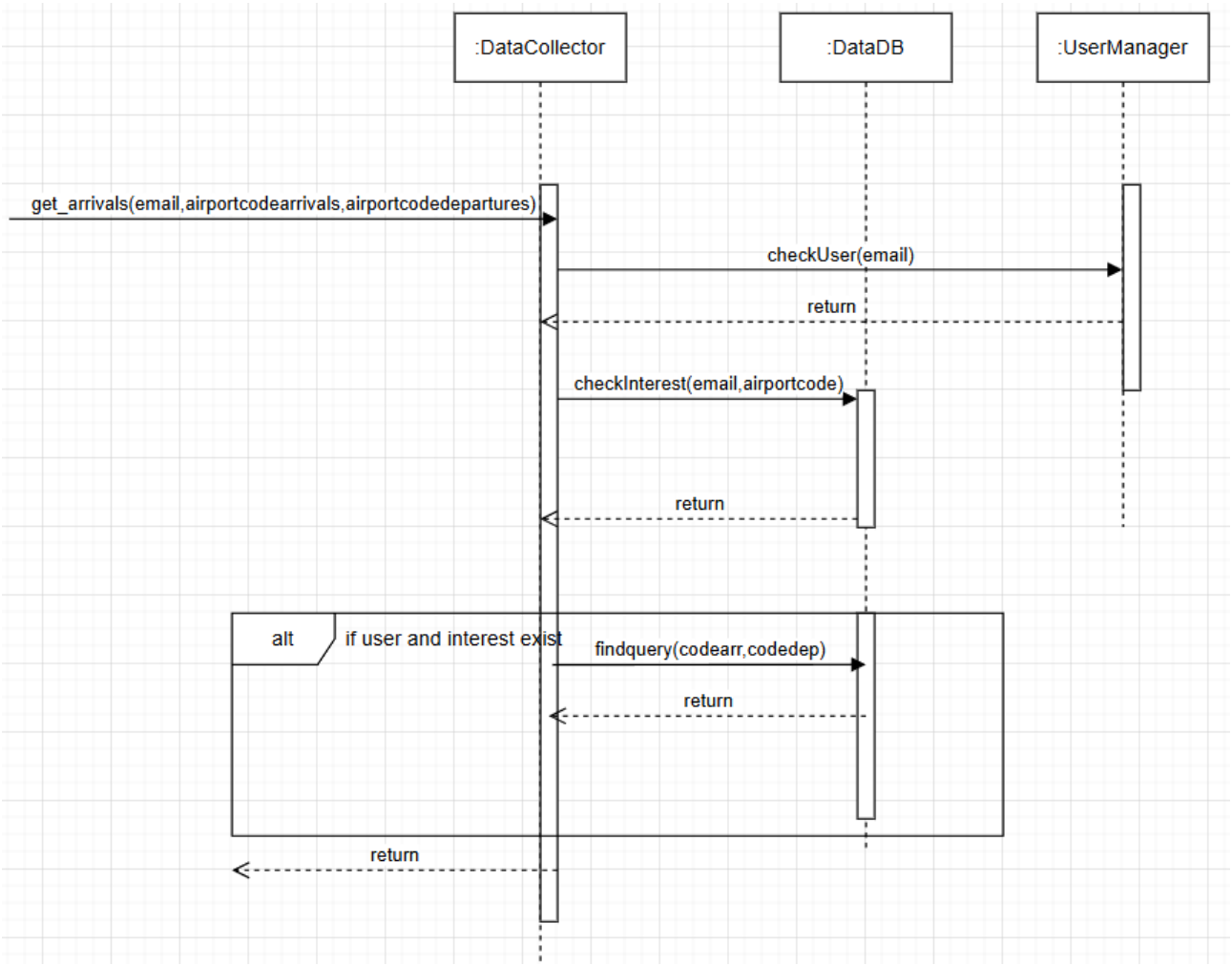
Get_last_flight



Get_average_flights



Get_arrivals



Get_flight

Diagramma delle interazioni simile ai precedenti

Update_flight_data