



**Università  
di Catania**

**ELABORATO  
DISTRIBUTED SYSTEMS  
AND BIG DATA**

**Antonio Inveninato 1000055701**

**Angelo Cocuzza 1000055700**

# Sommario

---

Introduzione .....	3
Scelte progettuali .....	3
Architettura del sistema .....	4
Auth service .....	5
Subscription service .....	7
Api service .....	8
Notifier service .....	9
SLA manager .....	10
Prometheus .....	14
Gateway .....	14
Test .....	15

# Introduzione

---

L'obiettivo del progetto è la creazione di una piattaforma che offra agli utenti la possibilità di ricevere avvisi giornalieri sulle migliori offerte di volo tramite e-mail. Dopo la registrazione o l'autenticazione, gli utenti possono sfruttare la funzione di sottoscrizione, inserendo parametri come città di partenza e destinazione, un intervallo di date desiderato per la partenza e il ritorno, oltre a un range di prezzi minimo e massimo.

Ogni giorno, viene richiamata un'API di volo (Kiwi) in base ai parametri specificati dall'utente. L'API seleziona i due migliori voli disponibili che soddisfano i criteri della sottoscrizione.

Successivamente, questi dati vengono inviati all'utente tramite e-mail, solamente se nei giorni precedenti quell'offerta non è già stata inviata, fornendo un servizio personalizzato e tempestivo per le migliori opportunità di viaggio.

## Scelte progettuali

---

L'obiettivo principale è la creazione di un'infrastruttura scalabile e modularizzata, in grado di garantire una gestione efficiente e indipendente di funzionalità specifiche attraverso microservizi distinti. L'architettura a microservizi è stata scelta strategicamente per consentire una maggiore flessibilità e adattabilità del sistema, permettendo lo sviluppo, il rilascio e la manutenzione indipendente di ciascun componente.

Nel progettare il nostro sistema, abbiamo fatto diverse scelte per garantire la robustezza e la scalabilità di un'architettura basata su microservizi.

Abbiamo optato per il linguaggio di programmazione Python insieme al framework Flask, noti per la loro chiarezza e flessibilità.

Per la persistenza dei dati, abbiamo scelto di associare un database MySQL a ciascun servizio, promuovendo l'isolamento dei dati e semplificando la manutenzione.

La comunicazione asincrona tra i servizi è stata implementata utilizzando Apache Kafka e Zookeeper. Kafka ci offre una piattaforma di streaming distribuita che gestisce la messaggistica persistente in modo affidabile, mentre Zookeeper supporta la sincronizzazione tra i nodi. La comunicazione asincrona tramite Kafka consente ai servizi di scambiare messaggi in modo efficiente e scalabile.

Per la raccolta e il monitoraggio delle metriche, abbiamo integrato il servizio Prometheus. Questo strumento ci consente di ottenere una visione dettagliata delle prestazioni del sistema, identificare eventuali problemi e monitorare il rispetto degli SLA definiti.

Abbiamo implementato un servizio SLA dedicato per gestire le metriche critiche del sistema. Questo servizio fornisce informazioni dettagliate sullo stato attuale delle metriche, identifica violazioni passate e valuta la probabilità di violazioni future.

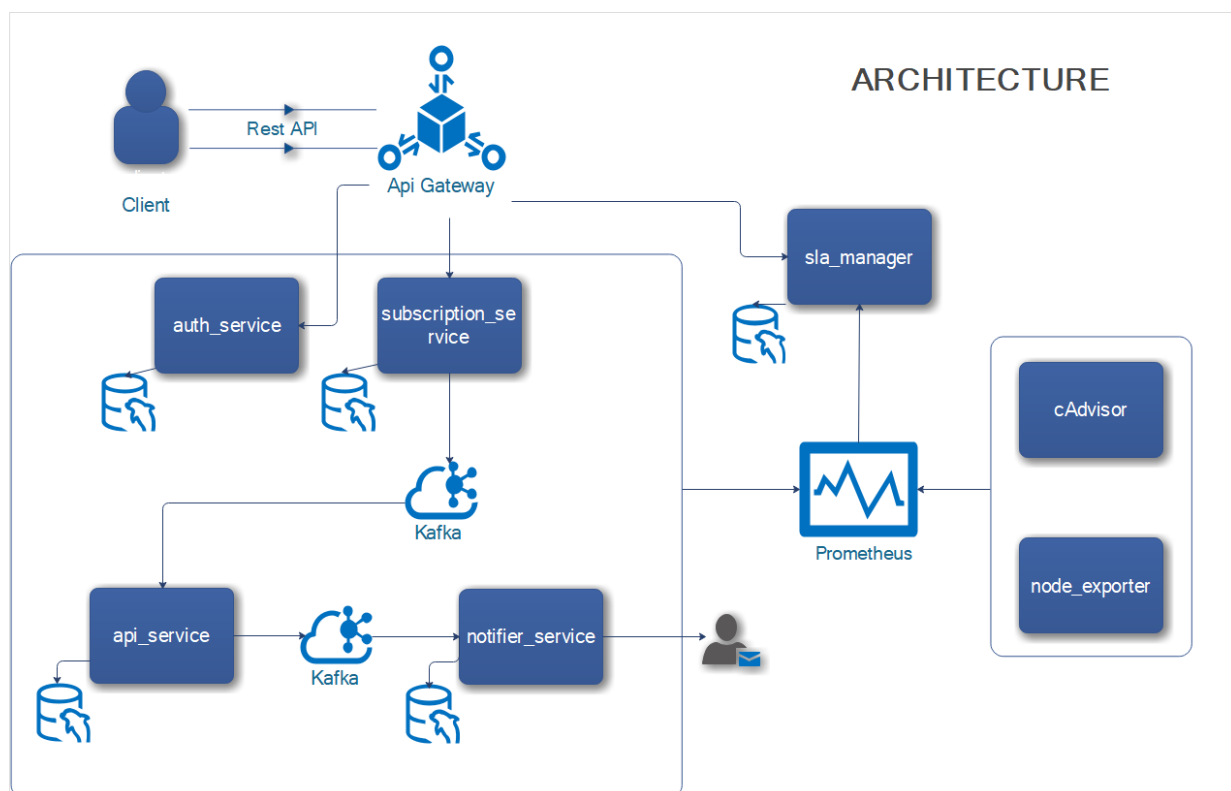
Per semplificare l'interfacciamento con i microservizi, abbiamo creato client specifici. Questi client nascondono i dettagli di implementazione e facilitano l'integrazione dei vari componenti del sistema.

Infine abbiamo incluso Nginx come componente chiave per la gestione dell'accesso alle nostre risorse. Nginx opera come un robusto gateway API, indirizzando le richieste degli utenti verso i microservizi corrispondenti. Questa scelta ci consente di implementare un routing efficace e un bilanciamento del

carico. L'integrazione di Nginx come gateway API rafforza la sicurezza, l'affidabilità e l'efficienza complessiva del nostro ambiente microservizi.

Complessivamente, queste scelte progettuali riflettono la nostra attenzione alla scalabilità, alla manutenibilità e all'efficace gestione delle metriche in un ambiente distribuito basato su microservizi. L'integrazione di tecnologie affidabili e la chiara definizione di ruoli e responsabilità contribuiscono alla creazione di un sistema robusto e flessibile.

## Architettura del sistema



**Figura 1. Architettura del sistema**

L'architettura è composta dai seguenti microservizi:

- auth\_service
- subscription\_service
- api\_service
- notifier\_service
- prometheus
- kafka
- sla\_manager
- gateway
- cAdvisor
- node-exporter

## Auth service

Il microservizio `auth_service` è progettato per gestire l'autenticazione degli utenti attraverso operazioni di registrazione e login. Queste operazioni coinvolgono un database MySQL per la memorizzazione dei dati degli utenti e l'emissione di token JWT per l'autenticazione.

### **Registrazione degli Utenti:**

Gli utenti possono registrarsi fornendo un nome utente e una password.

La password deve essere lunga almeno 8 caratteri.

Vengono registrati i tentativi di registrazione e se l'utente esiste già.

### **Autenticazione e Login:**

Gli utenti possono effettuare il login fornendo le loro credenziali.

Vengono registrati i tentativi di login, inclusi quelli che falliscono.

Se le credenziali sono valide, viene generato un token JWT che verrà utilizzato per le successive richieste autenticate.

### **Token di Autenticazione:**

Viene utilizzato JSON Web Token (JWT) per creare token di autenticazione sicuri.

I token contengono il nome utente e un tempo di scadenza.

I token vengono emessi durante la registrazione e il login.

### **Metriche di Monitoraggio:**

Il microservizio registra metriche di monitoraggio, come il numero totale di utenti registrati, tentativi di login riusciti/falliti e metriche di utilizzo delle risorse del sistema.

Utilizza Prometheus per esporre queste metriche, consentendo il monitoraggio delle prestazioni e delle risorse del microservizio.

### **Scheduler di Metriche:**

Un job schedulato periodicamente raccoglie metriche sulle risorse di sistema come l'utilizzo della CPU, la memoria e lo spazio su disco.

Queste metriche sono utili per il monitoraggio delle prestazioni e la risoluzione dei problemi.

### **Gestione degli Errori:**

L'applicazione gestisce gli errori durante il processo di registrazione e login, fornendo messaggi appropriati agli utenti.

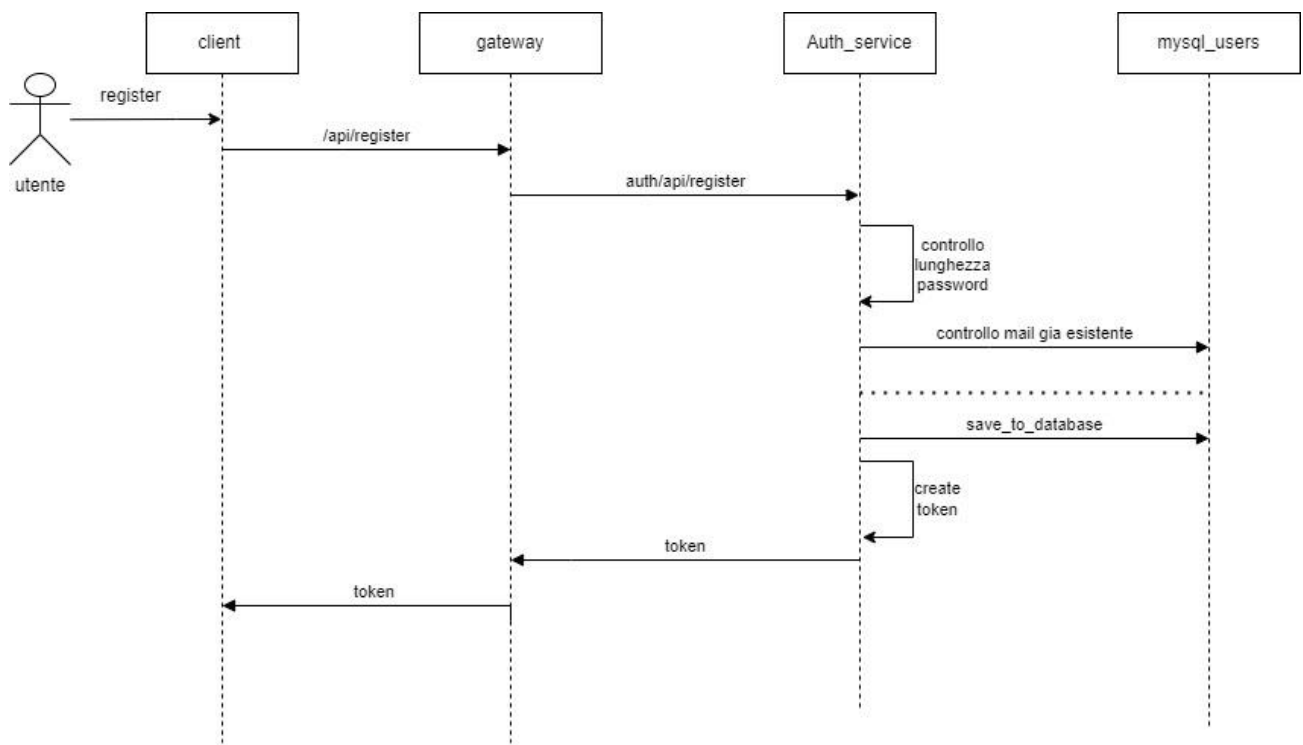
### **CORS abilitato:**

Abilita il Cross-Origin Resource Sharing (CORS) per consentire alle richieste di provenire da domini diversi.

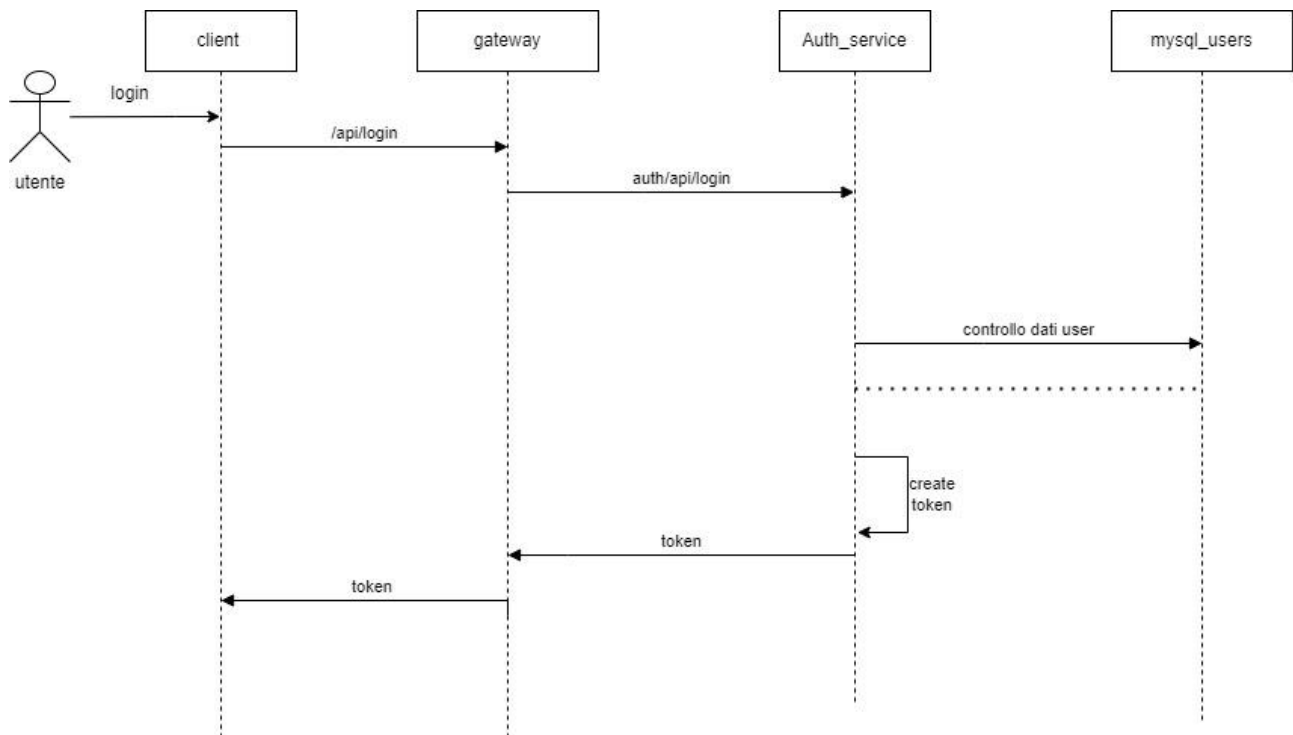
### **Database:**

Interagisce con il database `mysql_users` per memorizzare le informazioni degli utenti.

## Diagrammi delle interazioni:



**Figura 2. Diagramma registrazione utente**



**Figura 3. Diagramma login utente**

## Subscription service

Il microservizio è progettato per gestire le sottoscrizioni di viaggio degli utenti, consentendo agli utenti di registrare le loro preferenze di volo.

La persistenza dei dati viene gestita tramite SQLAlchemy, mentre la comunicazione asincrona avviene attraverso Kafka per l'invio di dati a sistemi esterni.

### Sottoscrizione al volo:

Gli utenti autenticati possono sottoscrivere ai voli, specificando città di partenza e destinazione, range di date di andata, ritorno e intervalli di prezzo.

### Sicurezza e Autenticazione:

L'autenticazione è gestita tramite l'utilizzo di token JWT, garantendo che solo utenti validi possano registrare le proprie preferenze di volo.

### Invio dei Dati a Kafka:

Dopo la sottoscrizione, i dati vengono convertiti in formato JSON e inviati a un sistema di messaggistica Kafka tramite il produttore Kafka integrato sul topic users.

### Monitoraggio e Metriche:

Il microservizio espone metriche Prometheus per il monitoraggio delle prestazioni, inclusi il numero di sottoscrizioni riuscite/fallite, il tempo di elaborazione delle sottoscrizioni e le metriche di utilizzo delle risorse del sistema.

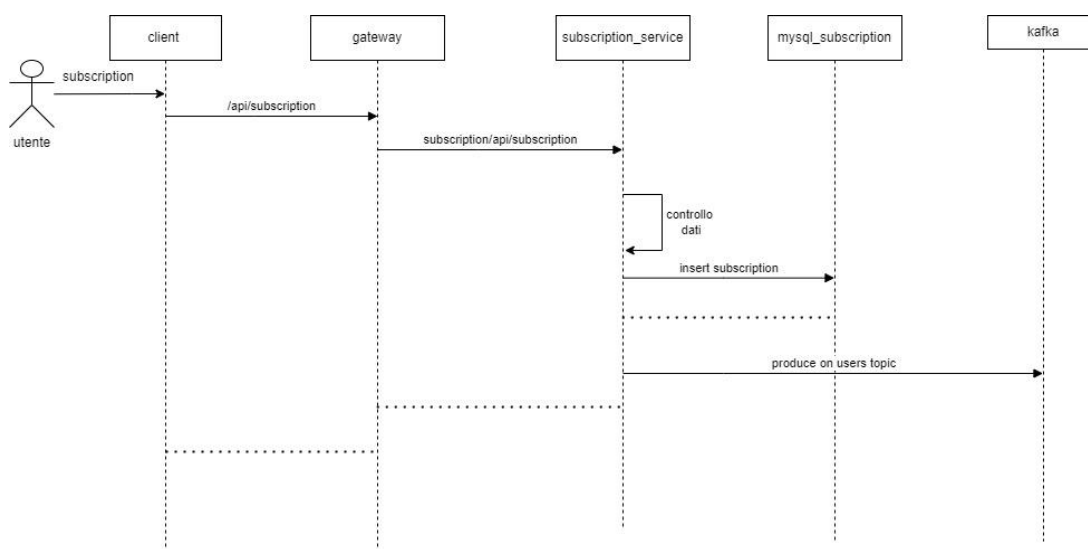
### Scheduler per le Metriche:

Un scheduler periodico esegue la raccolta di metriche sulle risorse di sistema, consentendo un monitoraggio continuo delle prestazioni.

### Database:

Interagisce con il database mysql\_subscription per memorizzare le informazioni delle sottoscrizioni degli utenti.

### Diagrammi delle interazioni:



**Figura 4. Diagramma sottoscrizione utente**

## Api service

Questo microservizio funge da consumatore e produttore in un'architettura basata su Kafka, con l'obiettivo di gestire le preferenze degli utenti per i voli e fornire dati sui voli.

### **Kafka Consumer:**

Si connette a un topic Kafka chiamato "users" come consumatore per ricevere messaggi contenenti le preferenze degli utenti.

Salva le preferenze degli utenti nel database MySQL utilizzando SQLAlchemy. Il modello SubPreferences rappresenta la struttura delle preferenze.

### **API Tequila Kiwi Integration:**

Utilizza l'API Tequila Kiwi per ottenere i codici IATA (International Air Transport Association) delle città specificate dagli utenti, dopodiché utilizza questi codici per ottenere i dati dei voli in base alle preferenze degli utenti.

### **Kafka Producer:**

Agisce come produttore inviando i dati dei voli su un altro topic Kafka chiamato "flights".

### **Monitoraggio e Metriche:**

Raccoglie metriche di sistema, come l'utilizzo della memoria, l'utilizzo della CPU e lo spazio su disco, utilizzando psutil e Prometheus.

Espone le metriche su un endpoint HTTP per il monitoraggio attraverso Prometheus.

### **Attività Periodiche e Pianificazione:**

Esegue ciclicamente attività periodiche tramite la funzione `schedule_jobs`.

Consuma messaggi da Kafka usando la funzione `consume_messages`.

Controlla l'orario corrente e, se sono le ore 8:00, esegue la funzione `flights` per ottenere e inviare dati sui voli a Kafka.

Misura le metriche di sistema tramite la funzione `measure_metrics`.

Ripete il ciclo con una pausa di 45 secondi.

### **Monitoraggio e Metriche:**

Il microservizio espone metriche Prometheus per il monitoraggio delle prestazioni.

### **Database:**

Interagisce con il database `mysql_api` per memorizzare le informazioni delle sottoscrizioni degli utenti.

### **Vantaggi di Kafka:**

Nel processo di progettazione, abbiamo considerato l'opzione di utilizzare una comunicazione gRPC sincrona per richiamare quotidianamente l'API. Tuttavia, abbiamo deciso di adottare Apache Kafka per vari motivi chiave che riflettono le esigenze del nostro sistema.

- Disaccoppiamento dei Microservizi:

Kafka offre un modello di comunicazione asincrona che favorisce il disaccoppiamento tra i microservizi. La pubblicazione dei dati su un topic consente ai servizi di produrre e consumare informazioni senza dipendenze dirette, promuovendo una migliore modularità e manutenibilità del sistema.

- Scalabilità per Sviluppi Futuri:

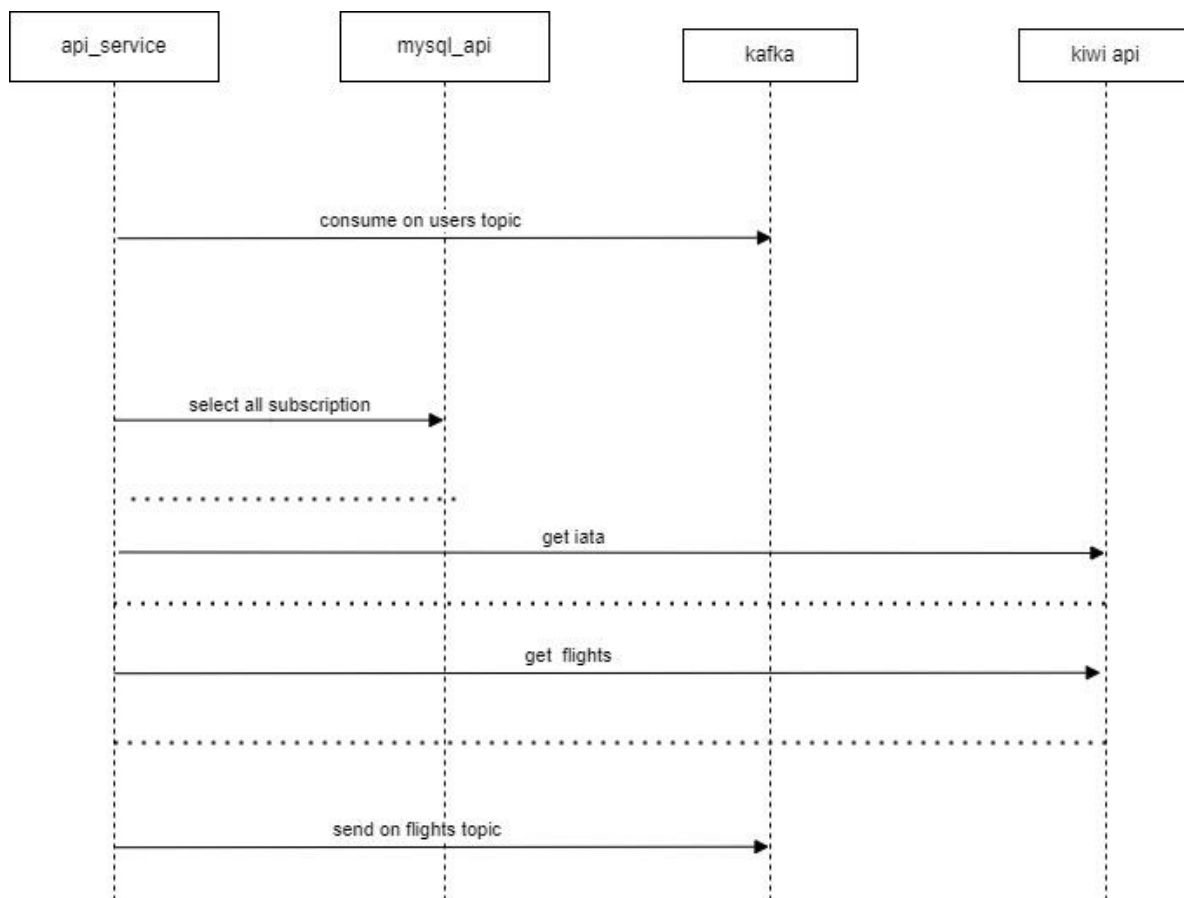


L'utilizzo di Kafka prepara il terreno per lo sviluppo futuro del sistema. La struttura di messaggistica distribuita consente a più servizi di consumare i dati pubblicati su un topic, garantendo una maggiore flessibilità e scalabilità rispetto alla comunicazione sincrona di gRPC.

- Persistenza dei Dati:

Kafka offre una persistenza superiore dei dati rispetto alla comunicazione sincrona. Una volta pubblicati i dati sul topic, essi rimangono accessibili e possono essere consumati da più servizi nel tempo.

#### Diagrammi delle interazioni:



**Figura 5. Diagramma interazione api\_service**

#### Notifier service

Questo microservizio è progettato per ricevere notifiche relative a nuove offerte di volo da un topic 'flights' Kafka, salvarle nel database MySQL e inviarle agli utenti interessati tramite e-mail.

#### Kafka Consumer:

Si connette a un topic Kafka chiamato "flights" come consumatore per ricevere messaggi contenenti dati relativi alle offerte di volo.

#### Salvataggio nel Database:

Utilizza SQLAlchemy per salvare le nuove offerte di volo nel database MySQL. Verifica se l'offerta è già presente nel database e la salva solo se è una nuova offerta o se il prezzo è migliore.

#### **Invio di E-mail di Notifica:**

Utilizza un account e-mail configurato per inviare notifiche agli utenti. Le credenziali dell'account sono ottenute da variabili d'ambiente.

Costruisce un'e-mail contenente i dettagli delle nuove offerte di volo e la invia all'utente interessato.

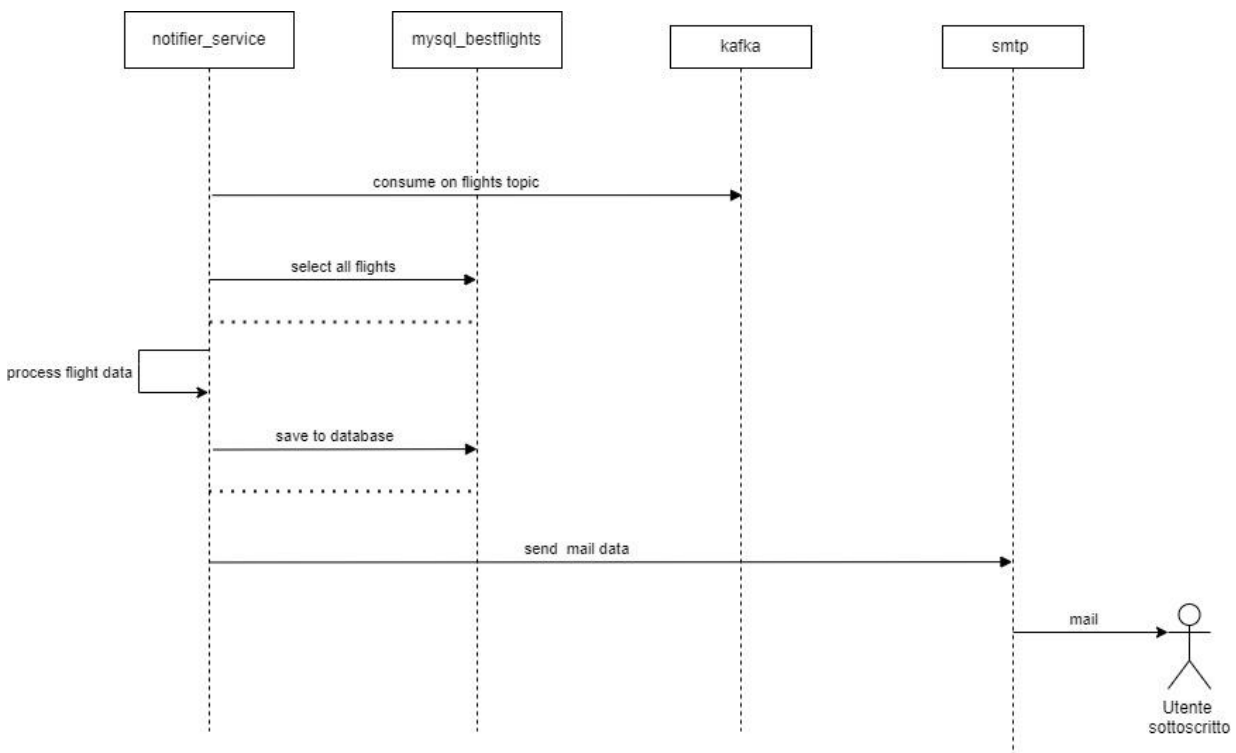
#### **Monitoraggio e Metriche:**

Raccoglie metriche di sistema, come l'utilizzo della memoria, l'utilizzo della CPU e lo spazio su disco, utilizzando psutil e Prometheus.

#### **Database:**

Interagisce con il database mysql\_bestflights per memorizzare le informazioni sui voli.

#### **Diagrammi delle interazioni:**



**Figura 6. Diagramma interazione notify\_service**

### **SLA manager**

Il microservizio offre un'infrastruttura robusta per il monitoraggio delle metriche, consentendo un'analisi approfondita delle prestazioni e facilitando la previsione di potenziali problemi.

#### **Configurazione del Database:**

Utilizza SQLAlchemy per la configurazione e l'interazione con un database MySQL, consentendo una gestione efficiente delle metriche.

Definisce un modello Metrics con colonne per il nome della metrica, i valori minimi e massimi.

**Endpoint `/api/add`:**

Implementa un endpoint POST che consente l'aggiunta di nuove metriche. I dati vengono ricevuti attraverso un modulo di form nella richiesta e vengono quindi memorizzati nel database per ulteriore monitoraggio.

**Endpoint `/api/status`:**

Fornisce un endpoint GET per recuperare lo stato corrente di tutte le metriche. Viene effettuata un'interrogazione a Prometheus per ottenere i valori attuali delle metriche, e si verifica se rientrano nei limiti specificati.

**Endpoint `/api/singlestatus`:**

Implementa un endpoint POST per ottenere lo stato di una singola metrica. Interroga Prometheus per recuperare il valore attuale della metrica specificata.

**Endpoint `/api/violations`:**

Fornisce un endpoint GET per ottenere il conteggio delle violazioni delle metriche in periodi di tempo specificati (1, 3, 6 ore). Utilizza i dati storici delle metriche e rileva violazioni in base ai limiti definiti.

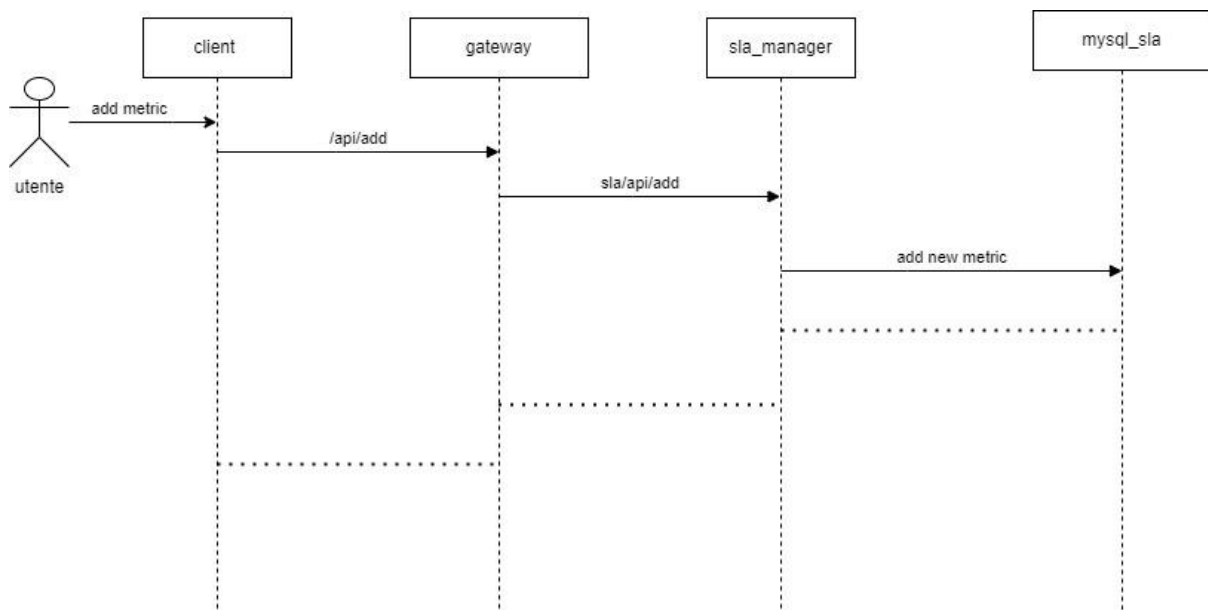
**Endpoint `/api/probability`:**

Implementa un endpoint POST per calcolare la probabilità di violazione in un periodo futuro specificato (6 ore). Utilizza i dati storici delle metriche e una funzione di previsione per stimare la probabilità di violazione.

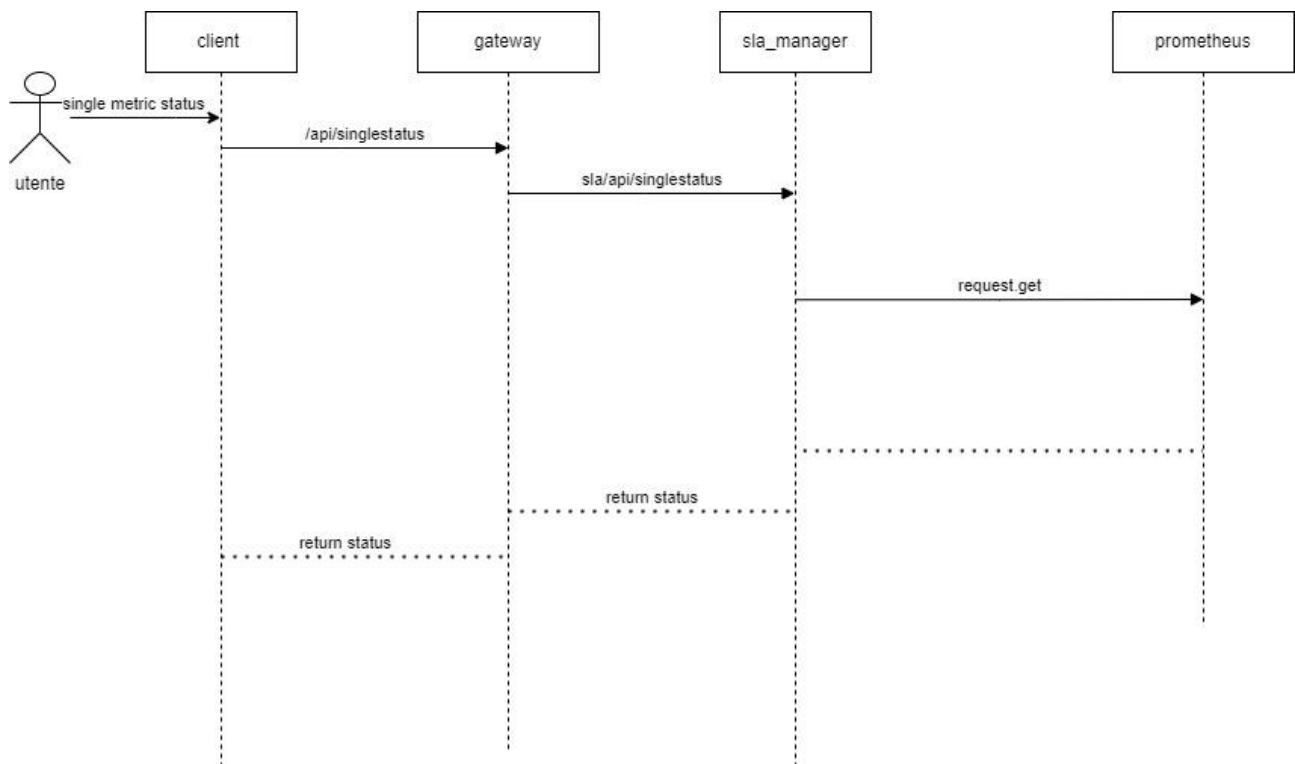
**Database:**

Interagisce con il database mysql\_sla per memorizzare le metriche.

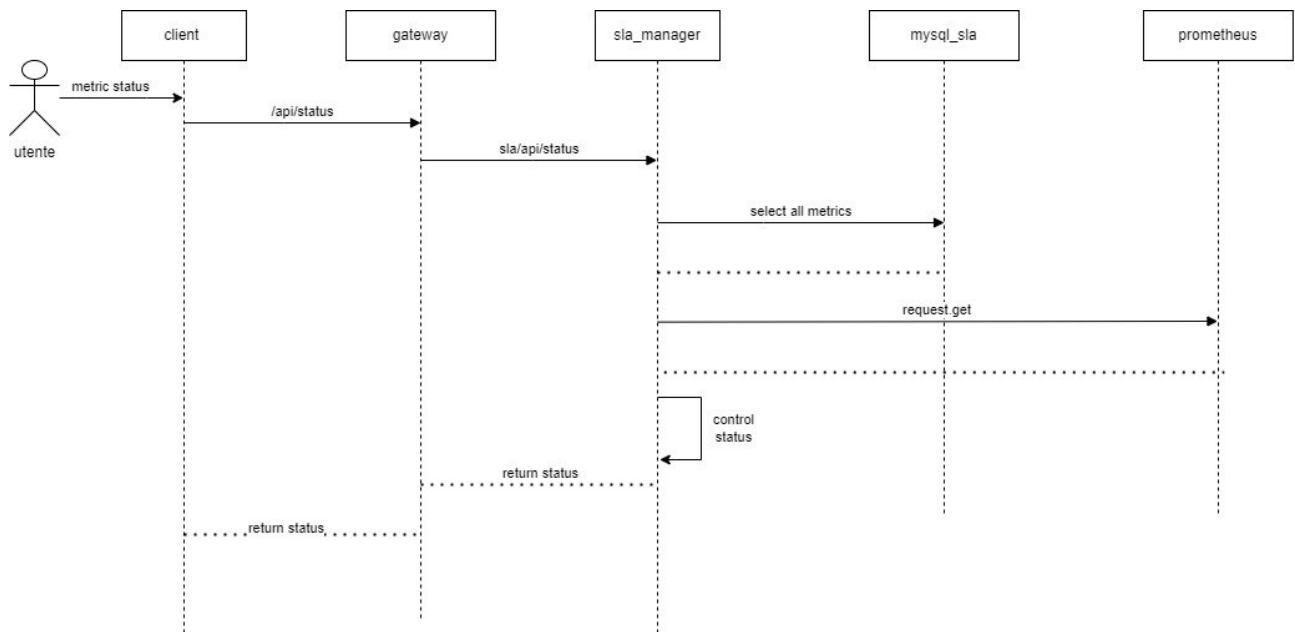
## Diagrammi delle interazioni:



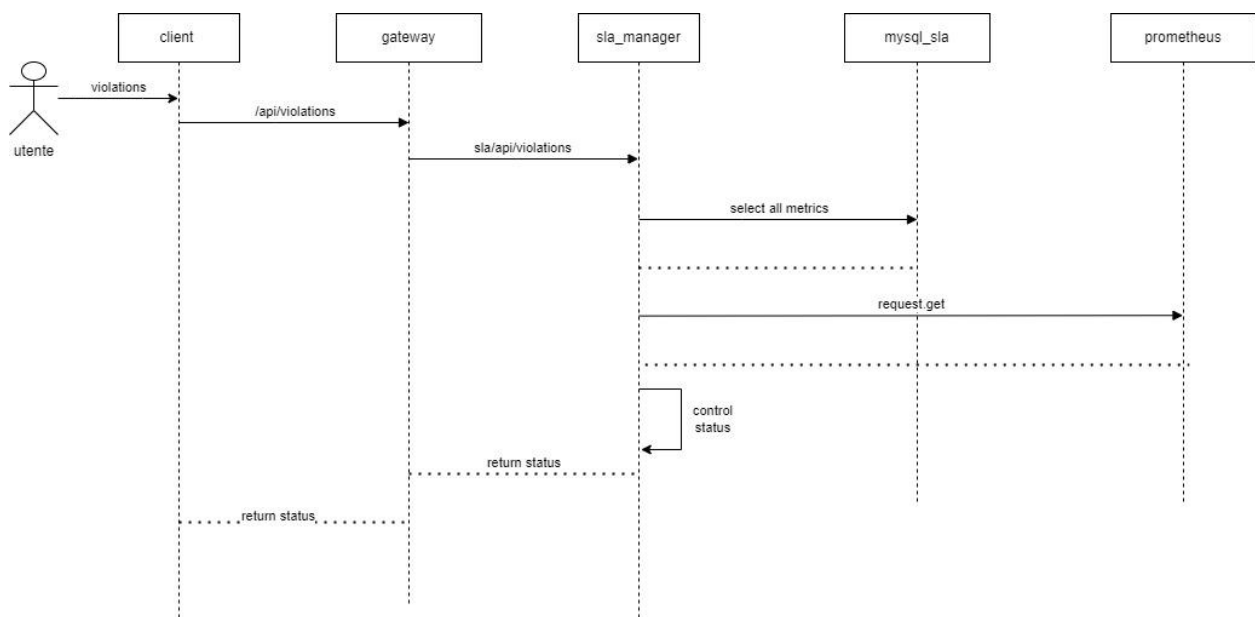
**Figura 7. Diagramma endpoint `/api/add`**



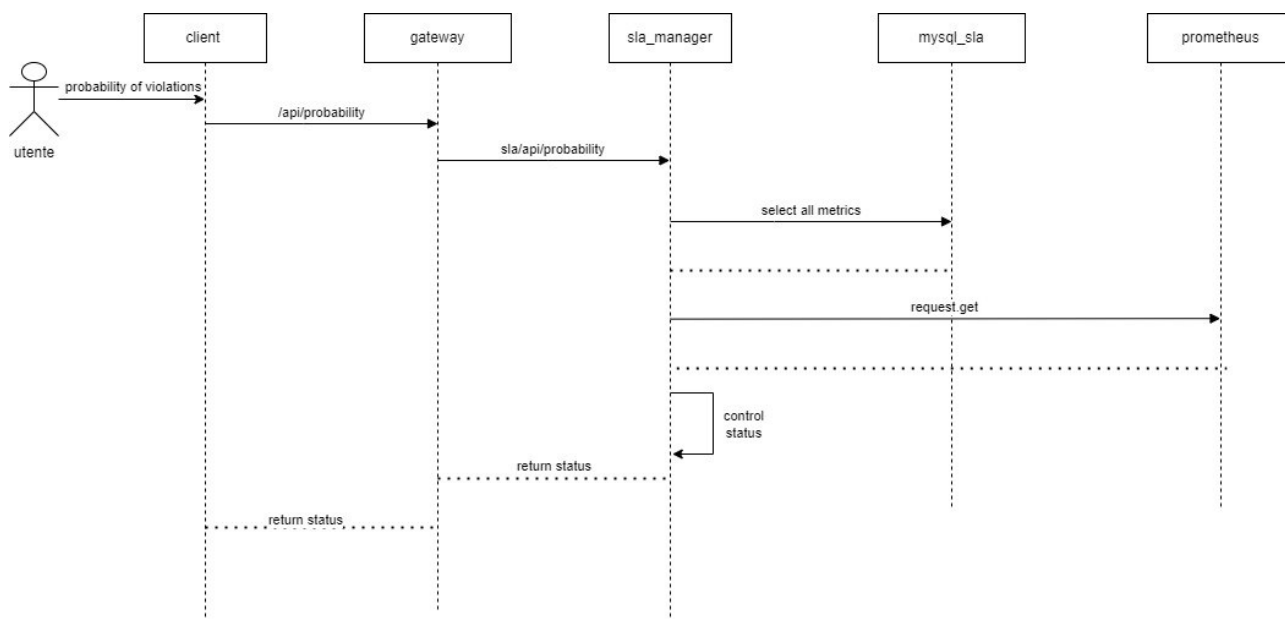
**Figura 8. Diagramma endpoint `/api/singlestatus`**



**Figura 9. Diagramma endpoint /api/status**



**Figura 10. Diagramma endpoint /api/violations**



**Figura 11. Diagramma endpoint /api/probability**

## Prometheus

Viene utilizzato Prometheus per monitorare i diversi microservizi

I microservizi `auth_service`, `subscription_service`, `api_service`, e `notifier_service` espongono endpoint metrici che forniscono informazioni sulle prestazioni e lo stato del servizio.

Prometheus esegue lo scraping periodico di questi endpoint per raccogliere metriche e mantenerle in un database.

In particolare è stato definito un file `.YML` che descrive la configurazione di un server Prometheus per il monitoraggio di diversi job tramite la raccolta di metriche. L'intervallo di raccolta delle metriche è di 15 secondi per tutti i job. Ogni job raccoglie metriche da servizi specifici attraverso le rispettive porte specificate.

## Gateway

Viene creato il file di configurazione che consente a Nginx di definire un server block che ascolta sulla porta 80 e gestisce il reverse proxy per tre upstream:

Upstream "auth": Questo upstream è configurato per instradare le richieste verso il servizio `auth_service` sulla porta 5000. Puoi facilmente aggiungere altri server a questo upstream se necessario.

Upstream "subscription": Similarmente, questo upstream instrada le richieste verso il servizio `subscription_service` sulla porta 5000.

Upstream "sla": Questo upstream instrada le richieste verso il servizio `sla_manager` sulla porta 5000.

Il blocco di server gestisce quindi i percorsi delle richieste per ciascun upstream specificando il prefisso di percorso tramite la direttiva `location`.

`location /auth/`: Tutte le richieste con il prefisso `/auth/` vengono instradate verso l'upstream "auth".

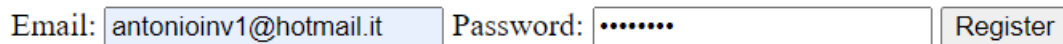
location /subscription/: Tutte le richieste con il prefisso "/subscription/" vengono instradate verso l'upstream "subscription".

location /sla/: Tutte le richieste con il prefisso "/sla/" vengono instradate verso l'upstream "sla".

## Test

---

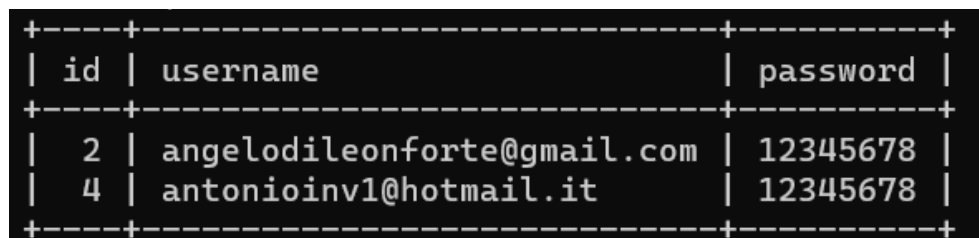
Per la fase di testing, saranno eseguite richieste attraverso i diversi client creati al fine di verificare la corretta comunicazione con i vari servizi. In particolare, verrà valutata l'efficacia della comunicazione di ciascun servizio con il proprio database. La prima attività coinvolge la creazione di un nuovo utente attraverso l'inserimento di dati specifici, con il microservizio responsabile di tale operazione identificato come 'auth\_service'.



Email:  Password:

**Figura 12. Client per la registrazione dell'utente**

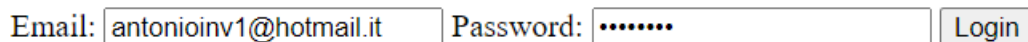
Una volta fatta la registrazione si può vedere che l'utente è correttamente salvato nel database nella seguente figura.



id	username	password
2	angelodileonforte@gmail.com	12345678
4	antonioinv1@hotmail.it	12345678

**Figura 13. Aggiunta dell'utente nel database**

Si può quindi fare il login con le stesse credenziali.



Email:  Password:

**Figura 14. Client login dell'utente**

Successivamente si viene reindirizzati al client di sottoscrizione dove si può anche vedere che è stato correttamente creato un token per mantenere l'accesso.

← → ↺ 🏠 Archivio C:/nginx-1.24.0/html/subscription.html?token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImFudG9uaW9pbmYxQGHvdG1haWwuaXQlClleHBpcmF0aW9u... ☆

## Flights Preferences

Inserisci le tue preferenze

Città di partenza:

Città di arrivo:

Data di partenza da (DD/MM/YYYY):

Data di partenza a (DD/MM/YYYY):

Data di ritorno da (DD/MM/YYYY):

Data di ritorno a (DD/MM/YYYY):

Prezzo minimo:

Prezzo massimo:

**Figura 15. Client sottoscrizione dell'utente**

Una volta inviati i dati, anche questi vengono correttamente salvati nel database come mostrato nella seguente figura.

```
mysql> select * from user_preferences;
```

id	user_id	city_from	city_to	date_from	date_to	return_from	return_to	price_from	price_to
4	angelodileonforte@gmail.com	Catania	Budapest	05/06/2024	06/06/2024	12/06/2024	14/06/2024	10	500
6	antonioinv1@hotmail.it	Catania	Milano	05/06/2024	06/06/2024	12/06/2024	14/06/2024	10	500

**Figura 16. Aggiunta sottoscrizione nel database**

Successivamente viene fatta una produce sul topic users di kafka.

Il messaggio generato viene consumato da api service e nella figura è mostrato il log relativo.

```
api_service | ERROR:root:{'user_id': 'antonioinv1@hotmail.it', 'city_from': 'Catania', 'city_to': 'Milano', 'date_from': '05/06/2024', 'date_to': '06/06/2024', 'return_from': '12/06/2024', 'return_to': '14/06/2024', 'price_from': '10', 'price_to': '500'}
```

**Figura 17. Log della consumazione di un messaggio in api\_service**

Questi dati vengono salvati nel database di api in modo che, quando arrivi l'orario determinato vengano fatte le corrispondenti chiamate all'api di kiwi. Inoltre questi dati vengono prodotti su un topic kafka chiamato flights .

Il notifier che è sottoscritto a tale topic consuma questi messaggi, come mostrato nella seguente immagine.

```
notifier_service | DEBUG:root:process_flight_data: {'id': '0a66034e4da74dae6847099a_0|0a66034e4da74dae6847099a_1', 'flyFrom': 'CTA', 'flyTo': 'LIN', 'cityFrom': 'Catania', 'cityTo': 'Milano', 'cityCodeFrom': 'CTA', 'cityCodeTo': 'MIL', 'countryFrom': {'code': 'IT', 'name': 'Italy'}, 'countryTo': {'code': 'IT', 'name': 'Italy'}, 'local_departure': '2024-06-05T17:15:00.000Z', 'utc_departure': '2024-06-05T15:00.000Z', 'local_arrival': '2024-06-05T19:05:00.000Z', 'utc_arrival': '2024-06-05T17:05:00.000Z', 'total': 12900, 'price': 113, 'conversion': {'EUR': 113}, 'fare': {'adults': 113, 'children': 113, 'infants': 113}, 'price_dropdown': {'base_fare': 113, 'fees': 0}, 'bags_price': {'1': 143}, 'baglimit': {'hand_height': 35, 'hand_length': 55, 'hand_weight': 8, 'hand_width': 25, 'hold_dimensions_sum': 158, 'hold_height': 52, 'hold_length': 78, 'hold_weight': 23, 'hold_width': 28, 'personal_item_height': 36, 'personal_item_length': 45, 'personal_item_weight': 3, 'personal_item_width': 20}, 'availability': {'seats': None}, 'airlines': ['AZ'], 'route': [{'id': '0a66034e4da74dae6847099a_0', 'combination_id': '0a66034e4da74dae6847099a_1', 'flyFrom': 'CTA', 'flyTo': 'LIN', 'cityFrom': 'Catania', 'cityCodeFrom': 'CTA', 'cityTo': 'Milano', 'cityCodeTo': 'MIL', 'local_departure': '2024-06-05T17:15:00.000Z', 'utc_departure': '2024-06-05T15:00.000Z', 'local_arrival': '2024-06-05T19:05:00.000Z', 'utc_arrival': '2024-06-05T17:05:00.000Z', 'airline': 'AZ', 'flight_no': 1720, 'operating_carrier': 'AZ', 'operating_flight_no': '1720', 'fare_basis': 'FOWLGN7', 'fare_category': 'M', 'fare_classes': 'F', 'return': 1, 'bags_recheck_required': False, 'vi_connection': False, 'guarantee': False, 'equipment': None, 'vehicle_type': 'aircraft'}, {'id': '0a66034e4da74dae6847099a_1', 'combination_id': '0a66034e4da74dae6847099a_1', 'flyFrom': 'LIN', 'flyTo': 'CTA', 'cityFrom': 'Milano', 'cityCodeFrom': 'MIL', 'cityTo': 'Catania', 'cityCodeTo': 'CTA', 'local_departure': '2024-06-12T08:35:00.000Z', 'utc_departure': '2024-06-12T06:35:00.000Z', 'local_arrival': '2024-06-12T10:20:00.000Z', 'utc_arrival': '2024-06-12T08:20:00.000Z', 'airline': 'AZ', 'flight_no': 1727, 'operating_carrier': 'AZ', 'operating_flight_no': '1727', 'fare_basis': 'FOWLGN7', 'fare_category': 'M', 'fare_classes': 'F', 'return': 1, 'bags_recheck_required': False, 'vi_connection': False, 'guarantee': False, 'equipment': None, 'vehicle_type': 'aircraft'}], 'booking_token': 'G6T-I7nkHqV83_e48dad45c9SveoUadFXdVMYq1fctE1LcxZWjJVJmUjmg2iFQbB8d7nhKMg36WcKcWGXF21STRQsy0Et9l6Mf2ySkMmAzHs31k0cINydeM-vTb82njdB8EUKAPfxxtsonD3n4q66AR1KEOLyavXV1vx4mVeyVvL5AvHDAYXTqv8yYkqaYLo88500zETHuSfCmSnrsyeyjgpU2kMN36E7glC8QFr2yGaRuU5uLDEQs7dYHyj0WhGy9DTd8q6M9mc2DRkdB10gogoyqLWCTRoDAhZyWmNXiHAHgd3z_g2EqnOXA4R3l3p3q3XqiC4NRxmq2Bv3hQrmalzo75Spzcqf-Veg_yD2ncN8oXpd0kWH_vjtAGSXWm8BM9djc00K28RV52QwaRGzS0dBo1LNx12B5moYGeytXMTNth3123PCaqGTL624kGaRrwhSfQPIGmcl9f7FF1o1SdFF2xHvJBfZhd2g4GnH4bH5KYdYdqm03_ahck_yQx6CSD33WLWxwDA8MulDwrc8vNj8wBoAazpYUN318sbE-YTUxysodUxSt7phzphBRXqtenogOkcBQApTo8Ne14w==', 'facilitated_booking_available': True, 'pnr_count': 1, 'has_airport_change': False, 'technical_stops': 0, 'throw_away_ticketing': False, 'hidden_city_ticketing': False, 'virtual_interlining': False, 'user_id': 'antonioinv1@hotmail.it'}
```

**Figura 18. Log della consumazione di un messaggio in notifier\_service**

Successivamente verifica le condizioni e li salva nel proprio database, infine li invia tramite e-mail all'utente, nella seguente immagine possiamo vedere i dati filtrati e la composizione della email.



```

notifier_service | DEBUG:root:ciaio {'id': '0a66034e4da74dae6847099a_0', 'combination_id': '0a66034e4da74dae6847099a', 'flyFrom': 'CTA', 'flyTo': 'LIN',
'cityFrom': 'Catania', 'cityCodeFrom': 'CTA', 'cityTo': 'Milan', 'cityCodeTo': 'MIL', 'local_departure': '2024-06-05T17:15:00.000Z', 'utc_departure': '2024-
06-05T15:15:00.000Z', 'local_arrival': '2024-06-05T19:05:00.000Z', 'utc_arrival': '2024-06-05T17:05:00.000Z', 'airline': 'AZ', 'flight_no': 1720, 'operatin
g_carrier': 'AZ', 'operating_flight_no': '1720', 'fare_basis': 'FOWMLGN7', 'fare_category': 'M', 'fare_classes': 'F', 'return': 0, 'bags_recheck_required': F
alse, 'vi_connection': False, 'guarantee': False, 'equipment': None, 'vehicle_type': 'aircraft'}
notifier_service | DEBUG:root:IL BODY È : Ci sono nuove offerte di volo disponibili per le tue richieste:
notifier_service | Andata:
notifier_service | Citta di partenza Catania
notifier_service | Aeroporto di partenza CTA
notifier_service | Aeroporto di arrivo LIN
notifier_service | Citta di arrivo Milan
notifier_service | Data di partenza 2024-06-05T17:15:00.000Z
notifier_service | Ritorno:
notifier_service | Citta di partenza Milan
notifier_service | Aeroporto di partenza LIN
notifier_service | Aeroporto di arrivo CTA
notifier_service | Citta di arrivo Catania
notifier_service | Data di ritorno 2024-06-12T08:35:00.000Z
notifier_service | Prezzo 113
notifier_service | DEBUG:root:Email inviata con successo!

```

**Figura 19. Log della creazione e invio dell'email**



**Figura 20. Struttura dell'email ricevuta dall'utente**

Per quanto riguarda l'sla\_manager si mostra il funzionamento dei vari endpoint.

Per prima cosa viene aggiunta una metrica con Add metric.

Metric name:  Min Value:  Max Value:  X Minutes:

**Figura 21. Client per sla\_manager**

id	metric_name	min_value	max_value
1	subscription_processing_time_seconds	10	25
2	cpu_usage_percent_notifier	1	100
3	cpu_usage_percent_api	1	100
4	cpu_usage_percent_auth	1	100
5	cpu_usage_percent_subscription	5	50

**Figura 22. Visualizzazione delle metriche nel database**

Mediante la Metric Status si ottiene una condizione booleana di tutte le metriche salvate, che indica se lo stato della metrica è compreso o meno all'interno dell'intervallo definito durante l'aggiunta della metrica.

```

Metrica: cpu_usage_percent_api Valore: true sla.html:83
Metrica: cpu_usage_percent_auth Valore: false sla.html:83
Metrica: cpu_usage_percent_notifier Valore: true sla.html:83
Metrica: cpu_usage_percent_subscription Valore: false sla.html:83

```

**Figura 23. Risultato della Metric Status**

Utilizzando Single Status si ottiene lo stato di una metrica.

```

▼ [{"...}] ⓘ
  ▼ 0:
    ▶ metric: {__name__: 'cpu_usage_percent_subscription', instance: '...'}
    ▼ value: Array(2)
      0: 1706711702.853
      1: "0.9"
      length: 2
    ▶ [[Prototype]]: Array(0)
    ▶ [[Prototype]]: Object
    length: 1
    ▶ [[Prototype]]: Array(0)

```

**Figura 24. Risultato della Single Status**

Mediante Violations visualizziamo le violazioni generate in 1,3,6 ore delle varie metriche

```

{"message": "Violazioni generate", "success": true, "violations": [{"1_hours": {"cpu_usage_percent_api": 6, "cpu_usage_percent_auth": 18, "cpu_usage_percent_notifier": 6, "cpu_usage_percent_subscription": 225, "subscription_processing_time_seconds": 241}, {"3_hours": {"cpu_usage_percent_api": 9, "cpu_usage_percent_auth": 20, "cpu_usage_percent_notifier": 9, "cpu_usage_percent_subscription": 705, "subscription_processing_time_seconds": 721}, {"6_hours": {"cpu_usage_percent_api": 9, "cpu_usage_percent_auth": 20, "cpu_usage_percent_notifier": 12, "cpu_usage_percent_subscription": 1284, "subscription_processing_time_seconds": 1340}}]}

```

**Figura 25. Risultato della Violations**

Con l'uso di Probability of violations si ottiene la probabilità di violazione nei prossimi X minuti.

```

{"message": "Probabilità di violazione.", "..."}
  message: "Probabilità di violazione."
  ▼ probability: {cpu_usage_percent_api: 0.3323093396431875,
    cpu_usage_percent_api: 0.3323093396431875
    cpu_usage_percent_auth: 0.6015981141361331
    cpu_usage_percent_notifier: 0.330883131735934
  }

```

**Figura 26. Risultato della Probability of violations**