## **NotifyMaps: Progetto Distributed Systems And Big Data**

L'obiettivo dell'applicazione è fornire all'utente finale un servizio che permetta di monitorare la durata dei percorsi alla quale l'utente si sia iscritto. L'utente mediante un applicativo desktop potrà registrarsi alla piattaforma e iscriversi a una o più tratte, potrà modificare le iscrizioni a esse e i dati relativi all'utente. La comunicazione verrà gestita mediante un client che utilizzerà l'applicativo Telegram per ricevere notifiche sullo stato dei percorsi alla quale è iscritto in modo tale da poter verificare se il tempo di percorrenza di uno di essi è nella media o vi è una congestione che aumenta il tempo di percorrenza di esso. Immagine che contiene testo, schermata, Rettangolo, diagramma

Descrizione generata automaticamente

Figura - Diagramma interazioni componenti

**Server Principale e Server Autenticazione**

Il server principale gestisce non solo la comunicazione proveniente dall’esterno mediante chiamate Rest, gestisce la comunicazione con Kafka per la persistenza e inoltro dei dati al topic relativo, la comunicazione con un database MySQL per la memorizzazione dei dati e delle relazioni tra utenti e tratte, gestisce le chiamate effettuate alle API Google di Geocoding e Routes.

Una volta avviato il server proverà a connettersi al database MySQL se si effettua correttamente la connessione egli creerà il database Route e internamente inserirà una tabella per le province, una tabella per le route dove saranno presenti tutte le tratte dove è iscritto ogni singolo utente, successivamente creerà lo slaDB. Una volta completato ciò si aprirà un server http in modo tale da poter accettare richieste REST, grazie all’avvio della goroutine in parallelo verrà gestito l’invio del messaggio al topic KAFKA una prima volta all’avvio e poi periodicamente ogni ora.

All’interno del Server è stato implementato il pattern CircuitBreaker mediante l’utilizzo del package gobreaker, la possibilità di utilizzare tutti i meccanismi di Query di un Database di tipo mysql tramite il package go-sql-driver/mysql.

Ogni connessione a un servizio/server differente ha un CircuitBreaker dichiarato con regole di visibilità del package dove si trova, sono stati utilizzati gli stessi parametri di configurazione per tutti i circuit breaker.

All’interno del file di configurazione saranno presenti tutti gli indirizzi per la connessione con i vari servizi/server e le api google utilizzate.

La struttura delle cartelle del progetto rispecchia la struttura nonché i servizi con la quale ogni server si interfaccia. Nel server main si gestirà mediante il package database verranno gestite tutte le chiamate al database per le operazioni creazione, lettura, cancellazione, attivazione e disattivazione di una tratta.

Tramite i servizi google di Geocoding possiamo andare a definire una posizione mediante una città e/o una via in modo tale da ottenere indietro una posizione in latitudine e longitudine utilizzate per utilizzare i servizi google di Route e ottenere una tratta con i relativi dati ad essa associata.

Tramite Kafka si va a gestire la comunicazione tra i vari client mobile, poiché una volta avviato il server i messaggi verranno inviati ad essi tramite il server python che potrà smistarli alle relative chat telegram.

Tramite l’apertura della porta 25536 accettiamo chiamate rest che permettono di ricevere dati dal client Postman che permette l’inoltro al serverauth per quanto riguarda le operazioni di registrazione e autenticazione, mentre se vengono fatte richieste riguardanti gli endpoint:

* deletesRoute

Tramite la funzione **HandleDeleteRouteRequest** riusciamo a cancellare una specifica Tratta prendendo in considerazione la partenza e la destinazione e l’e-mail dell’account alla quale essa è collegata.

* registerRoute

Tramite la funzione **HandleRegisterRouteRequest** riusciamo a registrarci a una specifica Tratta prendendo in considerazione la partenza e la destinazione e l’e-mail dell’account alla quale essa è collegata.

* enableRoute

Tramite la funzione **HandleEnableRouteRequest** riusciamo ad attivare la ricezione di notifiche a tutte le tratte prendendo relative all’email dell’account alla quale esse sono collegate.

* disableRoute

Tramite la funzione **HandleDisableRouteRequest** riusciamo a disattivare la ricezione di notifiche a tutte le tratte relative all’email dell’account alla quale esse sono collegate.

* getRoute

Tramite la funzione **HandleGetRoute** riusciamo a disattivare la ricezione di notifiche a tutte le tratte relative all’email dell’account alla quale esse sono collegate.

* getprovince

Tramite la funzione **HandleGetProvince** riusciamo a ottenere una lista contenente tutte le province presente nel database.

* authentication

Tramite la funzione **HandleAuthRequest** effettuiamo una basic authentication così da inoltrare i dati al server di autenticazione che si occuperà di confermare o negare il login dell’utente

* register

Tramite la funzione **HandleRegisterRequest** effettuiamo un inoltro dei dati al server di autenticazione che si occuperà di registrare o meno in caso di errore l’utente.

* Gettg

Tramite la funzione **HandleGetTg** andiamo a selezionare tutte le route che hanno le notifiche abilitate e la inoltro al server auth successivamente tale lista contenente le e-mail verrà filtrata in modo tali da avere email univoche e otterremo il corrispondente id\_tg da esse.

* getuserdata

Tramite la funzione **HandleGetUserData** andiamo a utilizzare tale funzione per il login mediante il cliente telegram poiché ci permette di aggiornare il campo id\_tg una volta che viene effettuato il login per la prima volta dall’app dopo aver effettuato la registrazione sul client C# e ritorna una lista delle tratte a cui è iscritto.

* updateuserdata

Tramite la funzione **HandleUpdateUserData** andiamo a utilizzare tale funzione per aggiornare i dati.

* deleteuser

Tramite la funzione **HandleDeleteUserData** andiamo a utilizzare tale funzione per eliminare i dati delle route e dell’utente.

Le precedenti funzioni descritte che riguardano l’utente vengono dopo essere state ricevute dal server principale inoltrate al server Auth alla porta 8081.

I relativi endpoint ci permettono di ricevere i dati e aggiornare/aggiungere/eliminare i dati attualmente presenti sul database di autenticazione.

All’interno del Server auth è stato implementato il pattern CircuitBreaker mediante l’utilizzo del package gobreaker per la connessione con il database.



Figura - Diagramma Componenti Microservizi

**Server Manager Notification e Server Command Telegram**

* **Server Command Telegram**

Il bot gestisce un processo di autenticazione tramite una conversazione di login, connettendosi a un server di notifiche per verificare le credenziali dell'utente. Una volta effettuato il login l’utente potrà ricevere le informazioni sul traffico a cui ha effettuato l’iscrizione sul client postman.

L’utente mediante il comando di start avvia la conversazione col bot, il quale replica con un messaggio di benvenuto e con un pulsante di login.

L’utente una volta premuti il pulsante di login entra in una conversazione col bot dove vengono chiesti in sequenza E-mail e Password per effettuare l’accesso al servizio. Durante tale conversazione è possibile interrompere l’inserimento delle credenziali digitando un comando specifico.

Una volta effettuato l’accesso sarà possibile interrompere il servizio con il comando di logout.

* **Connessione al Notification server**

Quando l’utente inserisce le credenziali di accesso o vuole eseguire il logout viene stabilita una connessione al Notification server tramite richieste POST. Nelle richieste vengono incapsulati l’e-mail, la password e l’ID della chat di Telegram in modo da poter contattare l’utente per restituire i messaggi del servizio e comunicazioni di vario tipo, come la notifica di server momentaneamente non disponibile.

Una volta ricevuta una risposta dal server, questa, viene analizzata e restituita al bot in modo da poter rispondere all’utente con l’eventuale messaggio ricevuto dal server (autorizzato, logout effettuato, utente non trovato, ...).

* **Server Flask - Manager Notification**

Il server Flask implementa delle route per gestire le richieste di autenticazione e logout da parte degli utenti tramite richieste POST.

Durante l’avvio il server aspetta che il server principale si avvia con successo per effettuare una richiesta di tipo GET per recuperare eventuali utenti che precedentemente avevano fatto l’accesso, questo meccanismo viene utilizzato per notificare gli utenti di ripetere l’operazione di login dal momento che il server ha avuto un crash inaspettato.

Una volta che il server è avviato e riceve una richiesta di login, questo si occupa di inoltrare i valori ottenuti dal bot al server principale. Se la richiesta è andata a buon fine, quindi il server principale mi restituisce un valore che indica che l’utente è stato trovato nel database utenti, il server manda un’altra richiesta POST al server principale, dove manda solo l’e-mail dell’utente, per recuperare i vari percorsi a cui è iscritto.

Se anche questa richiesta va a buon fine viene effettuata un’iscrizione al topic di kafka che da quel momento in poi controlla periodicamente il topic. Quando il consumer kafka nota che è stato inserito un messaggio nel topic questo viene analizzato e trasmette ai vari utenti online le informazioni sulle tratte a cui sono iscritti.

Questo procedimento continua fino a quando non viene richiamata la route di logout. Questa come il login effettua una richiesta POST al server principale, dove viene passata l’e-mail, per notificare che l’utente ha chiesto di interrompere il servizio. A questo punto il server flask aggiorna la lista degli utenti online presente nel consumer kakfa eliminando l’utente e quindi impedendo l’arrivo di nuovi messaggi.

* **Circuit Breaker**

Per tutte le richieste verso i server, sia dal bot al server flask e dal server flask al server principale, viene utilizzato il meccanismo di circuit breaker per individuare malfunzionamenti e disabilitare momentaneamente l’accesso ad un servizio non funzionante.

* **Prometheus**

Le metriche esposte su prometheus sono:

* **Message\_sent** che mi permette di andare a salvare il numero di messaggi inviati agli utenti. Questa metrica viene utilizzata per andare a valutare se abbiamo la necessità di aggiungere un ulteriore consumer kafka per la gestione degli utenti o se vi è la necessita di aggiungere un ulteriore server per distribuire il carico di lavoro.
* **request\_time\_to\_main** permette di salvare ogni qual volta si effettua un API request al server principale il quantitativo di tempo che la richiesta impiega in secondi.

Questa metrica ci permette da andare a verificare il tempo di risposta tra il Manager Notification e il server Principale, cosi da andare a verificare se in uno specifico momento esso è sovraccaricato di richieste, così da poter valutare possibili modifiche future al server attraverso l’allocazione di ulteriori risorse a tale server o il partizionamento del lavoro su diversi server.

* **SLA manager**

Lo SLA manager è un server flask che permette di recuperare le metriche esposte su prometheus. Le route di questo server permetto di salvare su un database apposito le metriche di cui si vogliono recuperare le informazioni, mediante la route **/Create\_Update**. Inoltre, qualora si voglia aggiornare il valore di violazione della metrica sarà possibile richiamare questa route con passando il nome della metrica con il nuovo valore di violazione.

Mediante la route **/actual\_value** sarà possibile recuperare il valore di violazione nel database e recuperare la metrica richiesta da prometheus. Una volta recuperata si andrà a valutare se il valore attuale della metrica è stata violata oppure no. Questa route tornerà il valore della metrica e dirà se questa è stata violata.

La route **/remove\_sla\_metrics \_value** permetterà di rimuovere dal database la metrica salvata.

La route **/violations** permette di recuperare i valori di una metrica per un quantitativo di ore stabilito, che sarà passato nella richeista. Una volta recuperati questi valori si andranno a verificare quanti di questi hanno avuto una violazione.

La route **/reevaluate\_model** permette di andare ad analizzare per un quantitativo di minuti tutti i valori della metrica che si sta analizzando. In questo modo verranno adattati modelli a serie temporali, esaminando le componenti cicliche, oltre a eseguire adattamenti polinomiali. Il processo è progettato per fornire informazioni dettagliate sulle tendenze. I coefficienti polinomiali e i coefficienti periodici verranno salvati nel database e recuperati in seguito.

La route **/probability** permette di valutare quanto è probabile che i valori siano al di fuori di determinati limiti, basandosi su una funzione di trend, creata con i coefficienti recuperati dal database, e utilizzando una distribuzione gaussiana. La probabilità viene calcolata attraverso l'integrazione della superficie definita dalla distribuzione gaussiana e dai limiti specificati. Questa funzione restituirà la probabilità che ci sia una violazione nei prossimi minuti.