M.Q.T.T. - Me Quiero Trovar un Trabajo

Progetto per Internet of Things Based Smart Systems del corso di Laurea Magistrale in Ingegneria Informatica dell'Università di Catania

Claudio Curto – Manlio Puglisi

Descrizione

Scopo del progetto è quello di realizzare un assistente virtuale personale tramite l'uso di un bot Telegram scritto in Python. Il bot dovrà fornire dei servizi all'utente tramite una subscription a specifici topic di un broker **MQTT**. Tali servizi sono:

- Posti di lavoro disponibili nella zona dell'utente;
- Meteo locale;
- News;

Il progetto gravita attorno all'utilizzo del protocollo di comunicazione **MQTT**.
Le principali tecnologie coinvolte sono Python con l'ausilio della libreria *Paho-mqtt* e JavaScript mediante il Framework Node.js e package *mqtt*.
A ciò, ci si appoggia alla piattaforma online *HiveMQTT*, che offre un Broker MQTT.
Ulteriori package di supporto sono descritti direttamente nelle sezioni riguardanti i client interessati.

MQTT

Il protocollo MQTT (Message Queue Telemetry Transport) è un protocollo di messagistica leggero di tipo *publish-subscribe,* posizionato in top-of TCP/IP. Nonostante sia stato inizialmente ideato per la comunicazione M2M, ha trovato presto applicazione per i sistemi IoT, in quanto il pattern implementativo fa in modo che si eviti il polling continuo, anche grazie ad un *terzo elemento*, ossia il *broker,* che effettivamente assume la funzione di server e si occupa di riceve e distribuire i messaggi tra i vari client.
Le strutture che vengono implementate sono:

- Publisher: è il client che si occupa della generazione del messaggio da inviare al broker. I messaggi sono divisi per *topic*, quindi un client Publisher invia un messaggio insieme al topic di appartenenza;
- Subscriber: è il client che effettua le sottoscrizioni ai *topic* (uno o più) e riceve dal broker i messaggi appartenenti ai topic a cui il client è sottoscritto;
- Broker: è il server effettivo, si occupa della ricezione e dello smistamento dei messaggi e permette anche funzioni di *retain*, ossia di mantenimento di messaggi in memoria. HiveMQTT, ad esempio, nella sua versione gratuita offre il retain dei messaggi per un massimo di 3 giorni;

Un nodo client è possibile che sia contemporaneamente publisher che subscriber.

Un topic è l'argomento/categoria a cui può essere assegnato un messaggio. Ne è ammessa la disposizione gerarchica.

Principali framework e librerie

Node.js

Nato nel 2009 per JavaScript V8 di Google Chrome, un motore per l'esecuzione di codice JavaScript sviluppato da Google, Node.js è un framework utilizzato per scrivere codice lato backend sfruttando un linguaggio di programmazione utilizzato per il frontend (quindi che tecnicamente necessiterebbe di un web browser per essere eseguito). Node.js quindi è stata una rivoluzione nell'ambito della programmazione di server e microservizi, trasformando JavaScript in un linguaggio altamente versatile (non solo per programmare server e microservizi, ma anche elaborazioni statistiche, scientifiche, etc.).

Node.js vanta di un package manager (ufficiale) molto efficiente, *npm*, che consente il download automatico di moduli/librerie/framework dal proprio database remoto e non solo. Permette il version control e una gestione delle dipendenze.

La libreria utilizzata per implementare le funzionalità di MQTT è - colpo di scena – *mqtt*, scaricabile mediante il comando CLI: **$ npm install mqtt**.

Telegram.ext e Paho-mqtt

Telegram è un'app di messaggistica istantanea che deve il suo successo a molteplici ragioni: garantisce privacy e sicurezza, permette la creazione e l'utilizzo di bot. Una tra le librerie Python più utilizzate per la stesura di bot Telegram è *telegram.ext*.

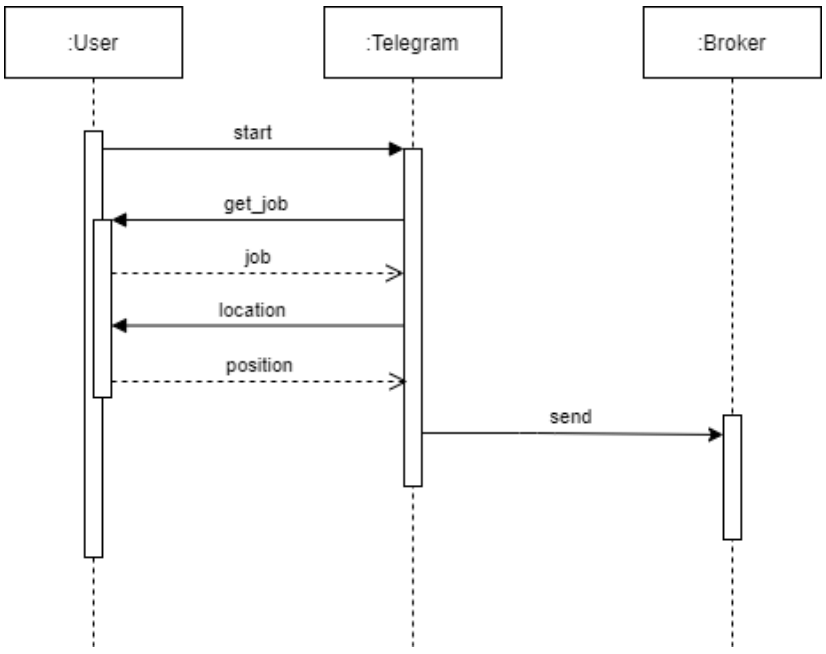
Per la realizzazione del client Telegram MQTT si è fatto uso della libreria Paho.mqtt, la quale mette a disposizione tutte le funzioni necessarie per la connessione al broker MQTT, subscription ai topic e invio di messaggi al broker stesso. Per ragioni di semplicità il client è stato implementato come classe, caratterizzata dai metodi:

- mqtt_connect(): ritorna un messaggio in console che conferma la corretta connessione al broker;
- mqtt_onmessage(): callback eseguita quando il client riceve un messaggio da un qualsiasi topic a cui è sottoscritto; a seconda del topic da cui riceve il messaggio verrà effettuata una "push" in una queue specifica, permettendo così il passaggio dei valori al bot, eseguito in un altro thread;
- publish(): metodo richiamato dal bot quando quest'ultimo viene avviato o viene richiesto un aggiornamento dei parametri (lavoro e posizione) dall'utente; in questo modo il pubblisher potrà filtrare i dati da inviare in maniera adeguata;
- run(): metodo necessario per l'avvio del thread; ivi vi sono definite tutti i passaggi necessari per instanziare il client, inviare le credenziali di accesso al broker, effettuare la connessione e tutte le sottoscrizioni ai topic specificati precedentemente;

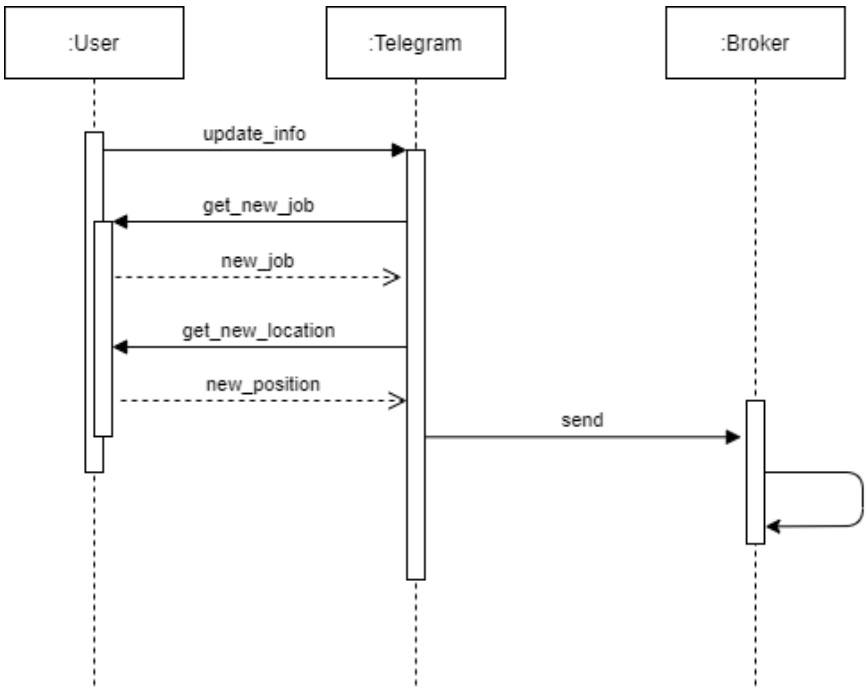
Bot Telegram

La parte relativa all'interfacciamento con l'utente è stata realizzata tramite la programmazione di un bot Telegram scritto in linguaggio Python. Si è scelta questa piattaforma grazie alla sua larga diffusione come app di messaggistica e semplicità d'uso. Il bot così realizzato si compone di tre file principali:

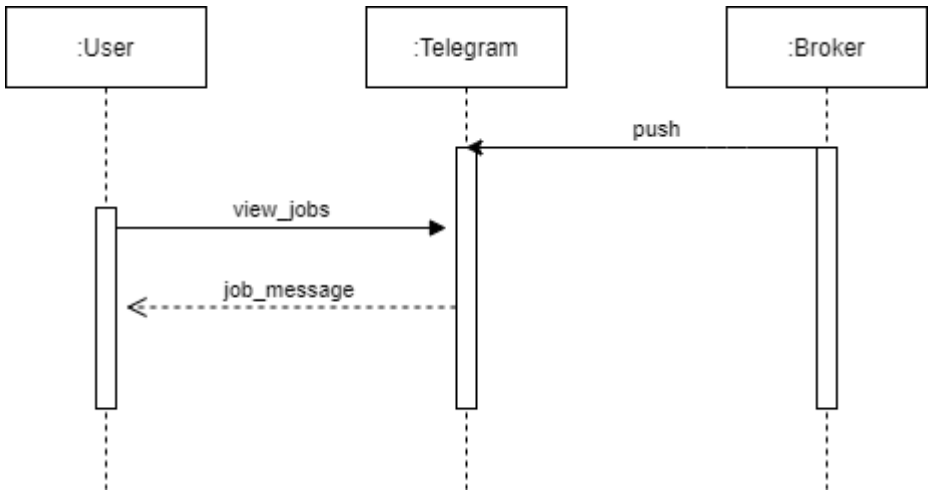
- **Main.py**, è il file da eseguire per mandare in esecuzione il bot. Il suo compito è importare i dati necessari per l'avvio del bot dal file config.py e instanziare le due classi MQTTClient e TelegramBot mandandole in run;
- **MQTTClient.py**, è il file in cui è implementata la classe MQTT, descritta in precedenza;
- **TelegramBot.py**, è l'effettivo Bot di interfacciamento con l'utente. Tramite semplici comandi testuali l'utente è in grado di richiedere i vari servizi messi a disposizione dal publisher MQTT. Sono stati definiti i seguenti comandi:
    - *start*: avvia il bot; una volta inviato inizializza una sequenza di richieste per l'utente nelle quali si richiedono informazioni relative al lavoro ricercavo e della posizione di interesse (non solo per il lavoro ma anche per gli altri servizi offerti). Una volta inseriti tutti i dati richiesti si passerà per uno stato SENDVALUES in cui tali dati verranno riformattati come json e inviati al publisher tramite il metodo publish() definito nella classe MQTTClient;



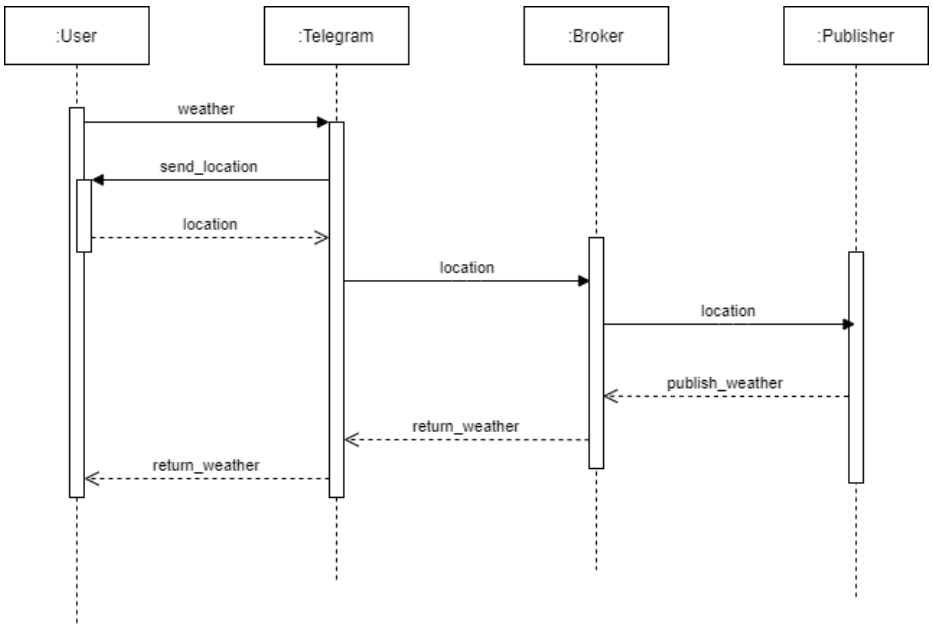
    - *aggiorna_informazioni*: per eventuale modifica dei valori inviati in fase di avvio del bot; lato implementativo richiama gli stati definiti nella sequenza di start;



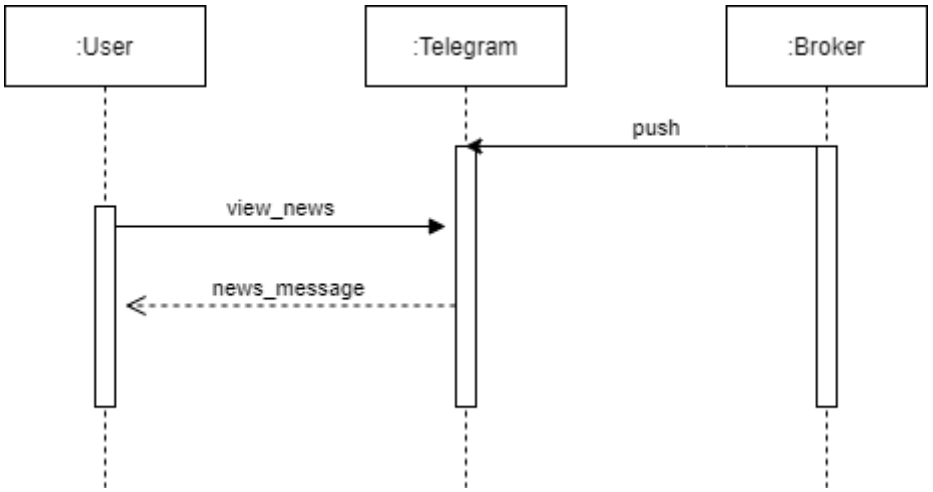
- *Lavori*: ritorna i link recuperati dal Broker in base alle informazioni indicate in precedenza;



- *Meteo*: è la chiamata per cui prima verrà richiesto all'utente di inviare la propria posizione geografica e ritornerà, sulla base di quest'ultima, informazioni su condizioni atmosferiche e temperatura;



- *Notizie*: ritorna i link alle notizie delle testate nazionali;



- *annulla*: annullamento dell'operazione di aggiornamento delle informazioni;

Si è deciso inoltre di porre i dati sensibili del progetto in un file di configurazione esterno.

Publisher Nodes

Un nodo MQTT può essere sia Publisher sia Subscriber. Nella fattispecie, i nodi che effettivamente si occupano della pubblicazione sul broker di informazioni richieste dal client Telegram basandosi sulla sua posizione GPS, devono essere sottoscritti al topic '*user/info'*, assumendo funzione di Subscriber.

Scendendo ad un livello più specifico, le informazioni che arrivano da un utente pubblicate nel topic 'user/info' riguardano il tipo di lavoro che cerca e la sua posizione espressa in *latitudine* e *longitudine*.

I nodi Publisher progettati sono tre:

- **LinkedinPublisher.js**, Publisher per la ricerca di lavoro. Assume la funzione di Subscriber al topic ‘user/info’;

- **WeatherPublisher.js**, Publisher per ottenere informazioni metereologiche basate sulla posizione del client. Assume la funzione di Subscriber al topic ‘user/info’;

- **NewsPublisher.js**, Publisher per l’ottenimento delle principali notizie provenienti da ogni parte del mondo – in italiano.

Il nodo Publisher viene visto più come un *wrapper* MQTT per l’implementazione di servizi modulari definiti all’esterno del nodo stesso.
Questa scelta è principalmente dovuta non solo per una questione di ordine e pulizia del codice, ma anche per garantire scalabilità, in quanto si possono aggiungere o rimuovere informazioni da far pubblicare dal nodo senza intaccare né il topic di destinazione né la struttura delle funzionalità chiave del nodo – quali Connect al Broker o funzioni event driven.

L’idea chiave rimane garantire l’elevata riutilizzabilità del codice.

I metodi implementati dai client Publisher sono:

- Client.connect**(obj),** a cui viene passato un oggetto contenente le informazioni necessarie per effettuare una connect mqtt, quindi informazioni riguardanti l’host del broker, username e password, e tipologia di protocollo (si usa *mqtts*);

- Client.subscribe(*‘topic’*) è il metodo che permette al client Publisher di sottoscriversi al topic utilizzato per inoltrare le informazioni del client Telegram ai client Publisher, che dovranno elaborare tali informazioni prima di poter effettuare una publish;

- Client.on(“**connect**”, …) e Client.on(“**error**”, …) sono gli event emitter che in base al realizzarsi degli eventi in grassetto permettono di effettuare una funzione di *callback* - nel codice, per questi due eventi sono implementate delle semplici funzioni di log;

- Client.on(“**message**”, …) è l’effettivo wrapper per l’elaborazione delle informazioni richieste dall’utente. Dato che per i nodi **LinkedinPublisher** e **WeatherPublisher** risultano essere necessarie le informazioni dell’utente - e quindi dato che al momento della stesura di tale relazione sono gli unici due Publisher ad essere sottoscritti ai topic, rispettivamente, l’uno a ‘*user/info*’ e l’altro a ‘*user/position*’ - all’arrivo delle tipologie di messaggio (ossia le informazioni utente), avvieranno procedure di elaborazione per poi effettuare una…

- … Client.publish(‘**topic**‘, …), ossia l’invio vero e proprio dei messaggi al broker. I topic dei tre client Publisher allo stato attuale sono:
    - ‘node/jobs’, topic utilizzato per pubblicare i lavori;
    - ‘node/weather’, topic per l’invio delle condizioni metereologiche;
    - ‘node/news’, topic per l’inoltro ogni 4 ore delle notizie più recenti.

All’interno delle publish, vengono definiti anche altri parametri, tra cui funzioni di callback (che anche qui sono semplici log per indicare l’avvenuto invio del messaggio) e **QoS**, settata di default a 0, che è stata aumentata a 2: per una questione di design dei messaggi che riceve il client Telegram, i messaggi devono essere ricevuti una e una volta sola. Questo perché il client Telegram pusha i messaggi ricevuti dal broker in una queue e, su richiesta dell’utente, tali messaggi devono essere estrapolati dalla queue – in base al topic – e inviati in chat. Dato che si tratta di una queue, il primo messaggio inviato in chat sarebbe il primo ad essere ricevuto, quindi, se venisse inviato un duplicato di tale messaggio, alla successiva richiesta il client Telegram invierebbe in chat il duplicato del messaggio precedente *e non* una risposta specifica alla richiesta.
Altro parametro, definito soltanto nel client **NewsPublisher** è il flag **retain** settato a *true*, in modo tale che se l’utente utilizza la chiamata per ottenere le ultime news, prima che siano passate le quattro ore del CronJob del nodo per il refresh di queste ultime, possa comunque ottenere delle notizie abbastanza recenti.

Servizi d’appoggio per i nodi Publisher

I moduli di supporto ai client Producer sono quattro:

- **LinkedinScraper.js** è il modulo più importante, in quanto è, insieme a *NearestCity.js*, parte logica principale del progetto. Sfruttando il package *puppeteer.js*, questo modulo è in grado di simulare il comportamento di un web browser, scaricando in locale le pagine ricercate e potendo interagire con queste – letteralmente ci sono metodi per effettuare mouse click, scrivere testo e input da tastiera – purché vengano applicati agli elementi della pagina ottenuti tramite *querySelector*, metodi JavaScript per effettuare ricerche di elementi con determinati attributi (quali classi, id, tag html, attributi custom...).
  Il funzionamento consiste nell’invocare la funzione *getJobLinks* che, presi in ingresso due stringe quali la qualifica lavorativa e la *città* dove si cerca il lavoro, effettuerà delle ricerche su Linkedin e restituirà un massimo di 15 link che reindirizzeranno alle offerte di lavoro che si trovano sul sito riguardanti la qualifica cercata.
  Se non viene specificata la qualifica – e quindi si ottiene un valore di stringa vuota ‘ ‘– verranno restituiti link di qualsiasi offerta di lavoro per la zona desiderata;

- **NearestCity.js** è un modulo che implementa una funzione di supporto per LinkedinScraper.js: presi in ingresso latitudine e longitudine, restituisce la città più vicina a tali coordinate nel raggio di 25 Km;

- **Weather.js** è il modulo in cui è implementata la funzione che, avente in ingresso latitudine tramite le API offerte da Open Weather Map (openweathermap.org), restituisce temperatura e descrizione delle condizioni atmosferiche in base alla posizione ottenuta in ingresso;

- **GoogleNews.js** è il modulo più semplice: sfrutta il package *google-news-json* per ottenere un elenco delle ultime news provenienti dal feed di

Google. Allo stato attuale, risponde inviando solo le ultime 5 – per questioni di praticità e risparmio di payload, dato che si sarebbero ottenuti messaggi troppo lunghi con troppe news e troppi link.