ISTITUTO DI ISTRUZIONE SUPERIORE BELLUZZI FIORAVANTI

IIS Belluzzi-Fioravanti
Indirizzo di Informatica e Telecomunicazioni

Elaborato concernente le discipline di indirizzo

Livechat

multi-user chat application

Link Progetto

Arto Manuel, 5' Bi a.s. 19/20

Indice

Introduzione

Il progetto ha come obiettivo la realizzazione di una web app che implementa una chat multi utente con varie funzionalità aggiuntive, tra cui un sistema di autenticazione e l'utilizzo di metodologie riguardanti la sicurezza dell'applicazione.

Il prodotto finale è un'applicazione cross-platform utilizzabile da Android, IOS e Browser che, dopo una fase di login/registrazione, permette agli utenti di scambiarsi messaggi real-time in privato o in modalità broadcast attraverso una chat globale.

Sono state implementate varie funzionalità per la sicurezza del sistema, tra cui: hashing di password con *salt*, utilizzo di token per l'identificazione degli utenti all'interno della rete e SSL per lo scambio di messaggi criptati tra client e server. Inoltre sono state prese scelte architetturali riguardanti la scalabilità e l'efficienza da parte del server per alleggerire il carico di risorse.

Questi sono i due link per utilizzare l'app, da browser o installando l'APK sul proprio dispositivo android:

- Web: link
- Android: link

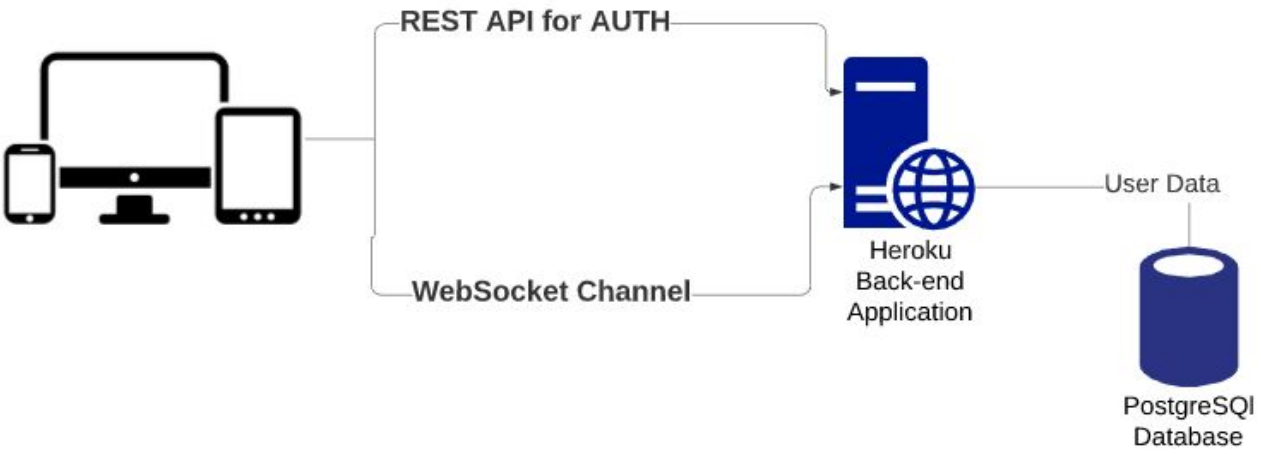
Il codice sorgente è reperibile a questo link con relative istruzioni al codice scritto e a come eseguirlo. Vi è inoltre una *gif* che illustra una simulazione del funzionamento dell'applicazione.

Tools utilizzati

Prima di introdurre l'architettura del sistema e le relative entità, mi piacerebbe illustrare i vari servizi/tecnologie che sono stati implementati per la realizzazione di questa applicazione:

- **Git** e **Github**: version control e piattaforma; hanno permesso un rapido sviluppo del software in maniera controllata e efficiente.
- **Heroku:** PaaS (Platform As A Service); servizio di hosting per applicazione back-end con una modalità free tier. Si appoggia su macchine virtuali EC2 di AWS ed è diventato famoso grazie alla sua semplicità di configurazione e scalabilità.
- **Github Pages:** servizio di hosting di siti statici.
- **Firebase Storage:** servizio di storage; utilizzato per il salvataggio delle foto profilo dei vari utenti.
- **Flutter:** UI Toolkit di Google basato su Dart; utilizzato per la parte di front-end per le piattaforme Android, IOS e Web
- **Flask:** microframework in Python; utilizzato grazie alla sua flessibilità per la realizzazione della parte di back-end
- **PostgreSQL:** ORDBMS (*Object-Relational Database Management System)*; utilizzato per la sua facile implementazione in Python e come alternativa a MySQL a puro scopo educativo.
- **SQLite:** DBMS interno alle applicazioni mobile;
- **JWT:** JsonWebToken, standard open (*RFC 7519*); utilizzato per l'autenticazione durante ogni richiesta.
- **WebSocket:** protocollo basato su TCP per una comunicazione bidirezionale
- **SPA:** *Single Page Application,* sito web nel quale i contenuti sono elaborati dinamicamente sul browser attraverso richieste HTTP eseguite in background.

Architettura



L'applicazione si basa su un'architettura a 3-tier, che favorisce una maggiore manutenibilità e scalabilità del sistema.

Le entità facenti parte dell'architettura sono:

- **Client:** relativa all'interfaccia utente, si divide in:
  - **Web Browser**: *SPA* richiesta attraverso il servizio di Github Pages
  - **Mobile**: applicazione installata sui sistemi operativi Android e IOS
- **Server**: applicazione risiedente su un server remoto con due Endpoint: uno per l'autenticazione e un altro per instaurare una comunicazione WebSocket per lo scambio di messaggi real-time
- **Database**: servizio relativo alla gestione dei dati persistenti risiedente su un altro server remoto; fornisce l'accesso ai dati dei vari utenti

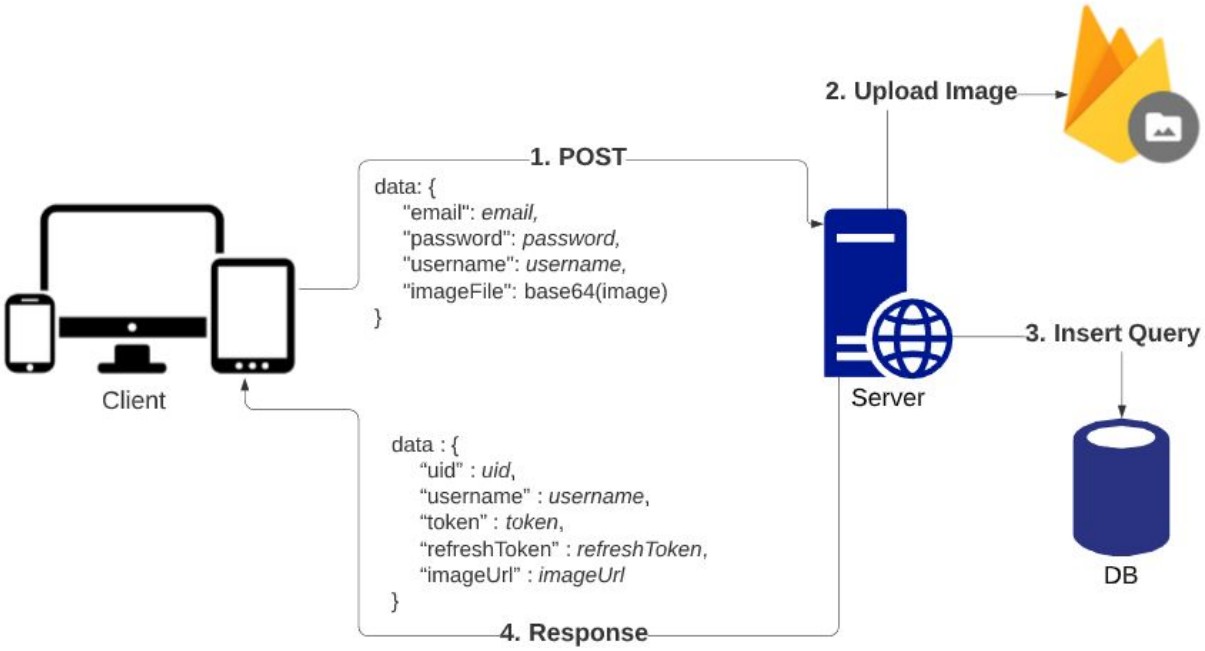
La logica di business è perfettamente distribuita tra client e server attraverso l'uso di chiamate API e scambio di messaggi in formato JSON: il client, sia da

browser che da mobile, effettua richieste al server per ottenere i dati relativi all'utente con i quali rielabora l'interfaccia grafica.

Un miglioramento da effettuare in futuro è quello di suddividere i due servizi relativi all'autenticazione e alla comunicazione websocket in due server distinti così da alleggerire il carico di risorse e aumentare la produttività del sistema.

Questo miglioramento è realizzabile grazie all'uso di JWT che, al contrario dei session-id che richiedono il salvataggio persistente degli id relativi a ogni connessione, permettono di identificare gli utenti all'interno della rete in maniera *stateless* così da poter utilizzare lo stesso token su più Server.

Fase di Autenticazione



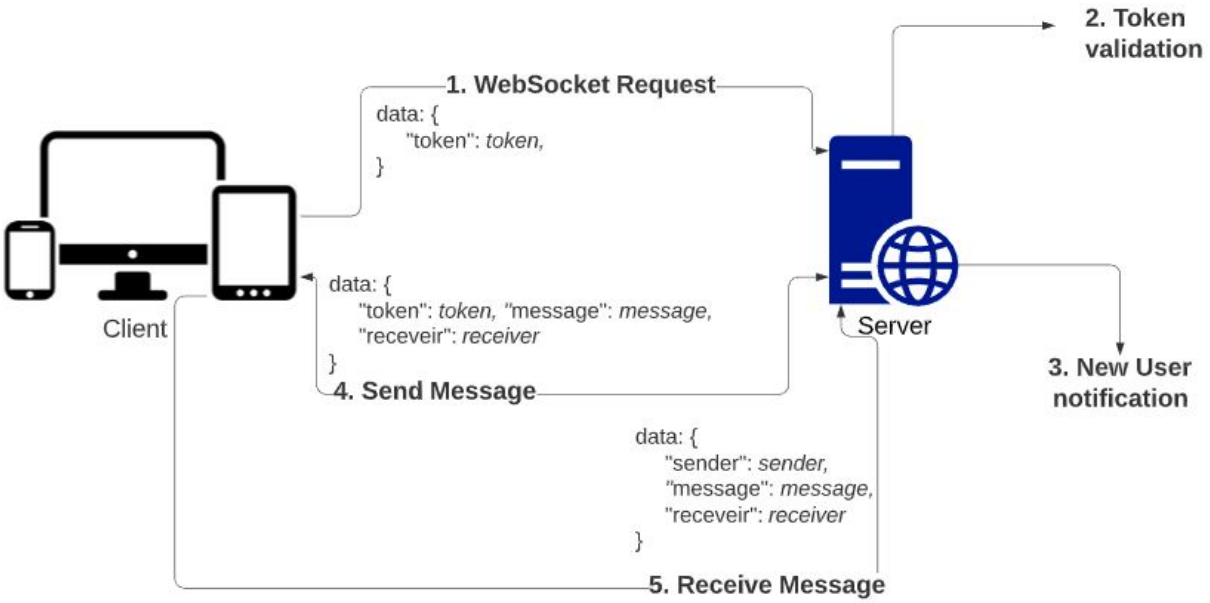
Step eseguiti durante la fase di registrazione di un nuovo utente:

1. **POST Request:** il client effettua una richiesta POST al server inviando, attraverso un formato json, i dati relativi alla registrazione *(username, email, password, Base64 dell'immagine profilo)*;

2. **Upload immagine profilo:** dopo aver verificato la correttezza dei dati, il server effettua l'upload dell'immagine profilo sul servizio di Firebase Storage;
3. **Insert Query:** il server effettua una richiesta per l'inserimento di un nuovo utente all'interno del DB. Qui viene gestito il controllo riguardante l'inserimento di username o email già esistenti, in tal caso il server risponde con un messaggio contenente l'errore avvenuto (stessa cosa avviene in caso di errore negli altri step);
4. **Response:** nel caso non vi siano stati errori il server risponde con un messaggio in formato json contenente le varie informazioni relative all'utente *(userID, username, token, refreshToken, imageUrl)*;

Durante la fase di Login l'unica differenza avviene tra lo *step 1* e lo *step 4* dove il server interroga il DB per verificare il corretto inserimento di email e password.

Scambio di messaggi



Step eseguiti durante la fase di scambio messaggi:

1. **WebSocket Request:** il client effettua una richiesta di *WebSocket upgrade* al server per instaurare una comunicazione bidirezionale;
2. **Token Validation:** il server verifica la validità del token e, in caso di successo, instaura la connessione;
3. **New User Notification:** il server aggiorna tutti i client dei dati relativi agli utenti online;
4. **Send Message:** dopo che la connessione è stata stabilita l'utente potrà inviare dei messaggi privati a singoli utenti o messaggi pubblici in una chat globale. Per identificare il mittente si utilizza il token;
5. **Receive Message:** ogni client può ricevere messaggi privati o pubblici;

Front End

La parte di Front end è rappresentata da due applicazioni, una per Android e IOS, e una SPA per l'utilizzo da browser, il tutto interamente realizzato in Flutter.

Flutter

Flutter è un framework open-source di google basato su dart, un linguaggio di programmazione simil java molto potente e innovativo che dallo stesso codebase permette di realizzare applicazioni native per Android, IOS e Desktop sfruttando una compilazione *JIT* e un compilatore *AOT*. Per il Web dart implementa un compilatore *dart2js* per la produzione di codice javascript eseguibile sul browser.

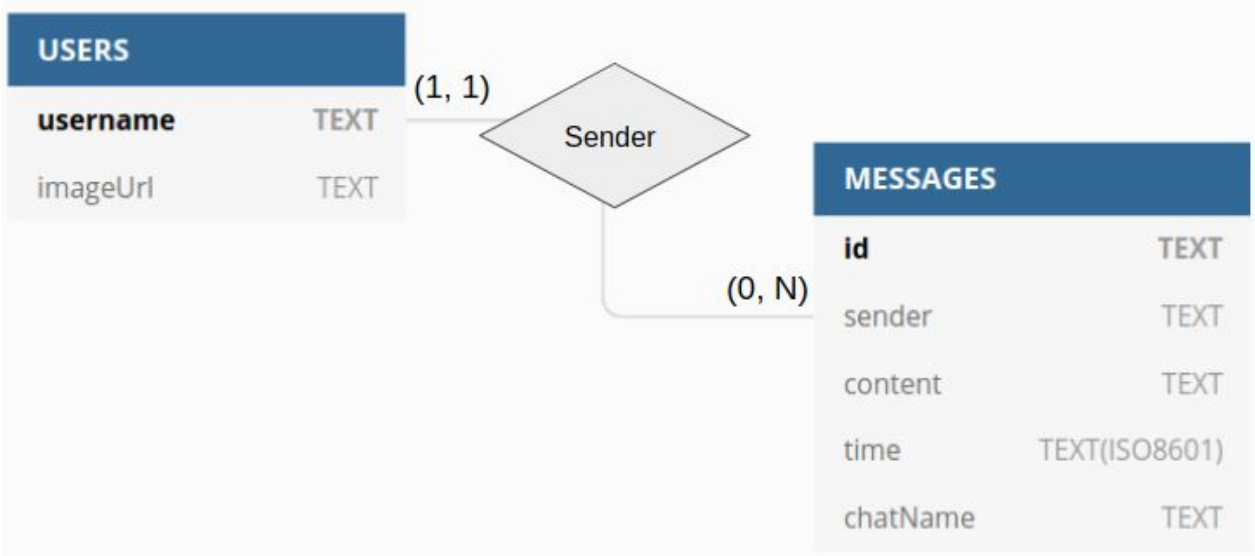
In mercato vi sono vari framework che permettono la realizzazione di applicazioni eseguibili su diverse piattaforme utilizzando un solo linguaggio, ma Flutter sta guadagnando molta popolarità grazie soprattutto alle ottime performance che riesce a ottenere confronto ai suoi rivali, primo tra questi è *React Native* di Facebook. A differenza di React, dove ogni elemento durante la compilazione viene trasformato nel corrispettivo elemento android, ios o web; l'engine di Flutter controlla ogni pixel dello schermo sul quale "disegna" gli elementi a runtime. Flutter permette perciò di ottenere una performance costante di 60 fps e implementa inoltre altre funzionalità tra cui caching e modalità di rebuild dinamiche.

Funzionalità aggiuntive

Oltre alle varie funzionalità di UX visualizzabili attraverso l'uso dell'applicazione, sono state implementate due modalità di salvataggio persistente riguardanti l'autenticazione dell'utente e le chat:

- **Auth**: ogni qualvolta si effettui il login, le informazioni relative all'autenticazione (token, refreshToken, username, userID) vengono salvate in memoria con modalità *SharedPreferences* (struttura chiave/valore) così da effettuare un auto login all'avvio dell'app se il token è ancora valido
- **Chat** (*solo mobile*): ogni chat viene salvata su un database SQLite interno

Database SQLite



Modello concettuale

Il DB contiene due tabelle *Users* e *Messages* e una relazione che identifica il mittente di ogni messaggio.

SQLite, al contrario di qualsiasi altro DBMS, non necessita di essere eseguito su un server ma è incorporabile all'interno di applicazioni. Permette di creare un DB utilizzando un unico file.

In ciascuna applicazione mobile è salvato un file DB per ogni utente che abbia effettuato l'accesso. Questo permette di utilizzare la stessa applicazioni da più utenti mantenendo divise le chat.

Le principali query utilizzate sono di *INSERT*, per l'inserimento di nuovi utenti o messaggi, e di *SELECT,* per il recupero totale durante l'avvio dell'app.

Un futuro aggiornamento è quello di inserire delle funzionalità di statistica che permetteranno all'utente di ricevere più informazioni riguardanti i propri messaggi:

Ecco alcuni esempi di funzionalità con relativa query:

- Visualizzare il numero di messaggi ricevuti da un utente specifico: SELECT count(*) from MESSAGES WHERE sender = ***USERNAME***
- Visualizzare il numero di messaggi ricevuti in un periodo di tempo: SELECT count(*) from MESSAGES WHERE time BETWEEN ***DATE1*** AND ***DATE2***
- Ricerca di una stringa nel contenuto di ogni messaggio ricevuto: SELECT * from MESSAGES WHERE content like "%***STRING***%"

Back End

Il server è stato implementato utilizzando Python e il microframework Flask che grazie alla sua incredibile flessibilità permette di realizzare applicazioni lato server di qualsiasi genere. Per questa applicazione il risultato è un server REST API con endpoint per l'autenticazione, una connessione websocket per la comunicazione bidirezionale e una connessione al DBMS PostgreSQL per il salvataggio delle credenziali.

WebSocket

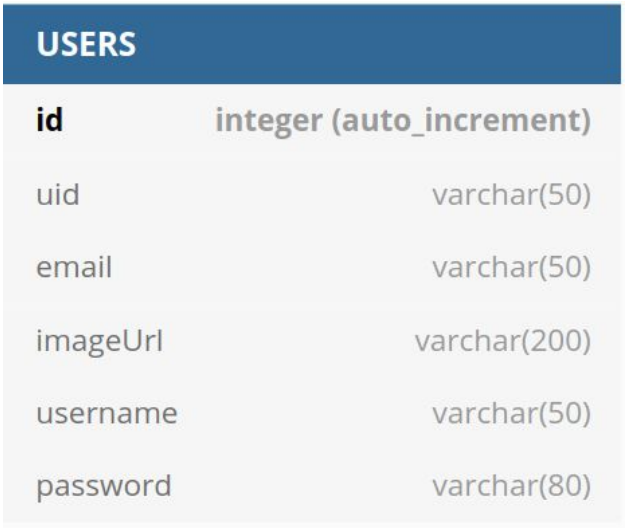
Uno dei problemi incontrati durante la progettazione dell'applicazione riguardava la funzionalità di messaggistica istantanea: HTTP, essendo un protocollo *stateless* in cui le richieste partono dal client, non fornisce l'invio di messaggi dal server al client.

Una prima soluzione era quella di effettuare richieste http secondo modalità di *long-polling*: il client effettua la richiesta e il server risponde solamente all'arrivo di un nuovo messaggio. Questa soluzione risulta essere inefficace per vari motivi tra cui l'eccessivo carico di risorse richiesto dal server per mantenere aperte le varie richieste.

Successivamente è stata implementata la tecnologia WebSocket che permette di instaurare una comunicazione bidirezionale tra client e server riducendo al minimo la latenza.

Per l'implementazione si è fatto uso della libreria *Socket IO* che include varie funzionalità aggiuntive tra cui l'auto riconnessione.

Database PostgreSQL



Il DB, risiedente su un altro server, gestisce una sola tabella contenente i dati di tutti gli utenti.

Esempio dati utenti:

id	uid	email	imageurl	username	password
1	a5792762-f03c-4b4e-a1c3-0b299eba92fd	email1@test.com	*link1*	user1	SHA256$AmXognZN$749479bfebf1a922b920b0301ad9b46b621bcd7e67aded5d9ee1e3831c8b2bc1
2	4e5838c4-b73a-48f4-a95a-7f3e00b6566a	email2@test.com	*link2*	user2	SHA256$Vcsgt9C8$5d841f5982003c145122985b6df152c211028e30026bc17d571acfc53df44801

Particolare attenzione al metodo di salvataggio della password: una buona norma è sempre quella di non salvare mai in chiaro le password così da proteggere il profilo degli utenti anche dopo un eventuale attacco al DB.

In questo caso è stata anche implementata una modalità di *salt* che permette di ottenere da due password uguali, due hash completamente differenti.

La struttura finale risulta essere: **method$salt$password**

Durante la fase di login il server non farà altro che ricalcolare l'*hash* utilizzando lo stesso *salt* e, se i due *hash* combaciano, la password inserita è esatta.

Le uniche query utilizzate riguardano l'inserimento di un nuovo utente o il recupero dei dati relativi a uno specifico utente.

Heroku

Heroku è un PAAS (Platform As A Service) che permette di effettuare il deploy di applicazioni back end senza preoccuparsi della configurazione. Si appoggia su macchine virtuali EC2 di AWS ed il tutto viene gestito attraverso dei "dyno" che rappresentano le risorse messe a disposizione per la macchina.

Per scalare l'applicazione basta aumentare il numero di dyno dedicato.

Inoltre Heroku fornisce una serie di add-on installabili; l'unico utilizzato è stato quello che implementa un server con un DBMS PostgreSQL installato.

Il sistema è inoltre protetto da attacchi DDoS.

Per eseguire un'applicazione su Heroku basta seguire dei semplici step di configurazione iniziale e successivamente effettuare l'upload del codice tramite git.

Ipotesi di lavoro e analisi degli aspetti critici identificati

L'implementazione del progetto è stata preceduta da una fase di progettazione in cui sono stati identificati la maggior parte dei problemi del sistema.

Primo tra questi risultava essere la ricerca di una tecnologia in grado di fornire una comunicazione bidirezionale tra client e server. Dopo una fase di ricerca in rete e studio dell'implementazione del protocollo WebSocket, il problema è stato risolto.

Sempre durante la fase di progettazione si è tenuto conto dei possibili attacchi contro il sistema, tra cui la alterazione del campo mittente di un messaggio. Il problema è stato risolto attraverso l'uso di JWT; preferiti alle session-id per favorire la scalabilità del sistema.

Questi due sono risultati essere gli aspetti critici più rilevanti e sono stati risolti già durante la fase di progettazione.

Durante l'implementazione non vi sono stati numerosi ostacoli e quelli incontrati sono stati completamente risolti.

Il prodotto è ancora in uno stato di *Beta* e i miglioramenti da effettuare sono numerosi sia nella parte di Back end che nella parte di Front end.

Uno di questi riguarda l'implementazione della crittografia end-to-end cosicché neanche il server possa leggere i messaggi inviati dagli utenti.

Approfondimenti

JWT

L'introduzione di token come metodo di identificazione è nato dopo aver identificato un problema di sicurezza nell'applicazione.

Nell'invio di un nuovo messaggio il json da inviare senza token risultava essere:

data: {
"sender": ***username*,**
"message": *message,*
"receiver": *receiver*
}

→ Un malintenzionato può falsificare il messaggio semplicemente inserendo nel tag "**sender**" l'username di un altro utente

Perciò l'unico modo per evitare questa falla di sicurezza era implementare un metodo che permettesse di identificare il mittente a ogni richiesta.

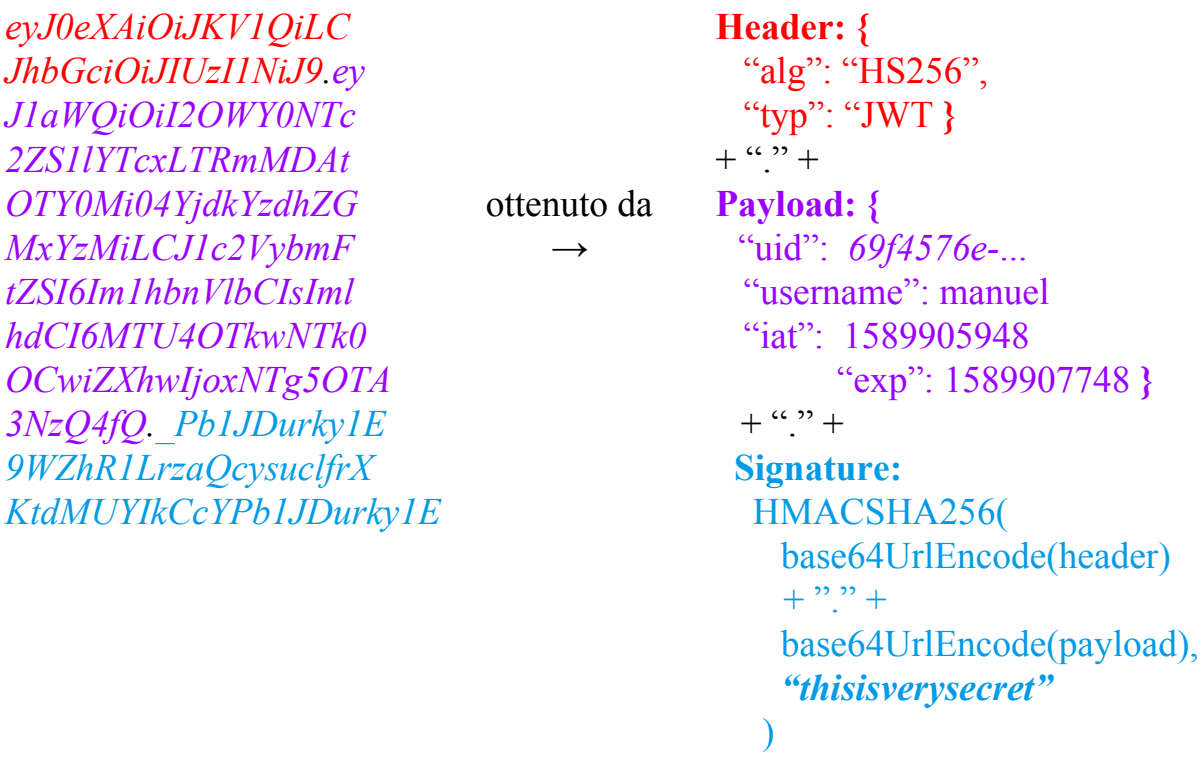
I Json Web Token sono un protocollo (**RFC 7519)** per la trasmissione di informazioni sicure attraverso oggetti json.

Ogni token è firmato con una chiave segreta che conosce solo il server.
Sono composti da 3 parti:

- **Header:** contiene le informazioni riguardanti il tipo di token, in questo caso JWT, e il tipo di algoritmo di firma utilizzato come "HMACSHA256" o "RSA";
- **Payload:** può essere inserito qualsiasi dato ritenuto necessario; implementa inoltre dati standard tra cui la data di scadenza, la data di registrazione ecc..;

- **Signature:** firma ottenuta dall'encode dell'header e del payload più la chiave segreta, nota solo al server;

Esempio di token:



Perciò a ogni richiesta il server verifica la validità del token (firma e data di scadenza) e se quest'ultimo è valido allora l'utente è verificato. In caso contrario la richiesta viene respinta.

Per puro scopo illustrativo la scadenza è di 30 minuti per il token e di 4 ore per il refreshToken. Dopo che l'utente effettua l'accesso viene inizializzato un timer di 30 minuti e al termine di esso viene effettuata una richiesta per il rinnovo del token utilizzando il refreshToken.

Se l'applicazione rimane chiusa per più di 4 ore, l'utente è automaticamente disconnesso.