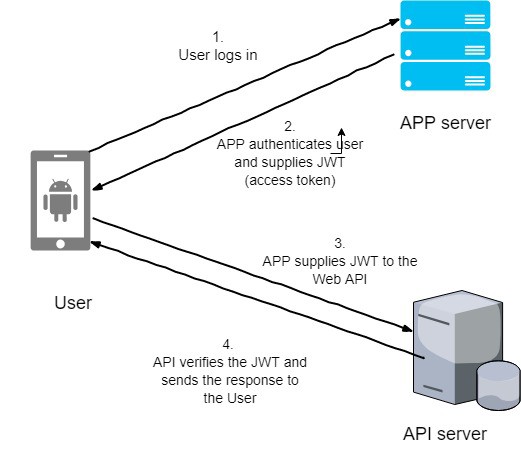
# Implementare JWT e refresh Token in .Net Core 2.2 Web API

## Che cos’è JWT

JWT, abbreviazione di Json Web Token, è esso stesso un token di accesso (una chiave privata) che viene dato all'utente autenticato che dà loro il diritto di accedere agli endpoint API.

Quindi, un utente accede alla tua app, l'app verifica che l'utente sia chi dice di essere (autentica), quindi l'app emette un token di accesso (JWT) all'utente che a sua volta utilizza per accedere alla tua API Web ma prima che l'API serva qualsiasi cosa, si assicura che l'app sia effettivamente quella che ha dato all'utente il token di accesso ... quello è il viaggio.



Un'ultima cosa:

JWT è una chiave privata generata crittografando il payload, l'intestazione e la firma utilizzando una chiave segreta. E può essere verificato utilizzando questa stessa chiave segreta.

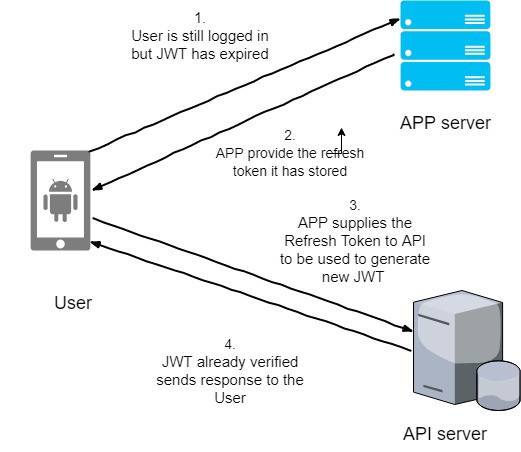
Nessuna paura di non spiegare quale sia il payload, l'intestazione e la firma, sarà chiaro in pochissimo tempo.

Ma la cosa con JWT è che scade. E quando scade, non vorresti chiedere all'utente di accedere nuovamente, giusto?

Quindi cosa farai? Aggiorna token.

## Che cos’è il refresh token?

Voglio dire, quando creiamo il JWT perché non creiamo una certa "stringa crittografata" chiamata token di aggiornamento, che viene salvata nel database dell'app utente. In modo tale che quando il JWT scade, dobbiamo solo fornire questo token di aggiornamento e se è verificato, possiamo generare un altro token di aggiornamento JWT + e il ciclo continua ...



JWT si assicura che coloro che accedono alla tua API siano autenticati, mentre il token di aggiornamento si assicura che non ti innervosisca chiedendo loro di accedere nuovamente.

Per rendere le cose molto più semplici, stiamo usando due modelli, uno per le informazioni di accesso dell'utente e l'altro per il token di aggiornamento.

Crea due classi e scrivi in questo codice:

Ora la logica

Vai alla cartella dei controller e aggiungi un'altra classe chiamata AccountsController.cs (si occupa della registrazione dell'utente in un token di accesso).

E dal momento che stiamo cercando di rendere le cose il più semplice possibile, avremmo un utente personalizzato che vogliamo accettare.

Che cosa abbiamo scritto?

Abbiamo già sottolineato che avremmo un utente personalizzato, quindi request.Username == "Jane" && request.Password == "Password!"

chiave È la chiave segreta che viene utilizzata per crittografare i dati ed è ottenuta dal file di configurazione che in questo caso è il nome del file applicazioniettings.json.

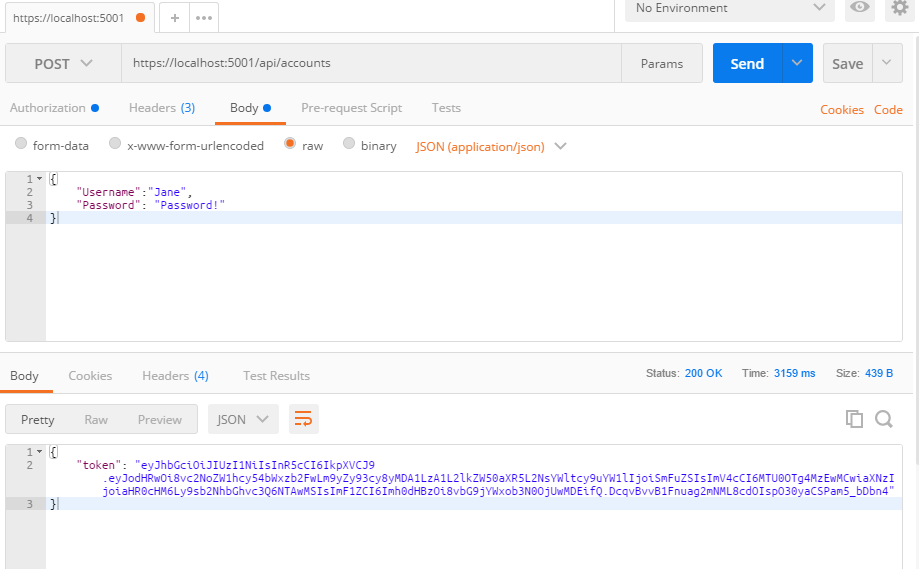
**cred** è quando stiamo creando il tipo di sicurezza che vogliamo usando l'algoritmo di sicurezza di tipo SHA256.

token è ciò di cui abbiamo parlato che rende sicura la nostra app ed è costituita dal payload che in questo caso è l'emittente, il pubblico, i reclami, scade; la firma che è signingCredentials.

Infine, aggiorna ConfigureServices e il metodo Configure della classe startup.cs come segue:

Siamo pronti e pronti a correre!

Userò Postman, ma sentiti libero di usare qualsiasi cosa che ti consenta di consegnare un payload a un endpoint API

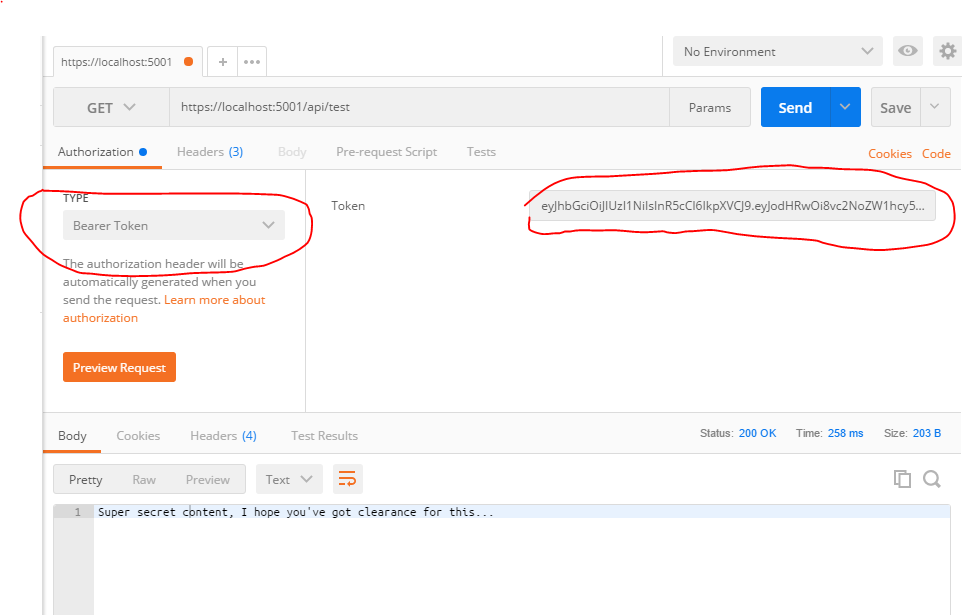


Sìì! funziona ... abbiamo creato con successo il nostro token di accesso.

Ora usiamolo

Creiamo un controller che richiede il token di accesso per visualizzare il suo risultato:

solo un controller che restituisce una stringa se viene fornito il token di accesso



pollice su! abbiamo fatto un ottimo lavoro.

Ora, se lo vogliamo, potremmo fermarci qui, ma dovremmo affrontare la rabbia dei nostri utenti quando dobbiamo dire loro di accedere dopo che il token di accesso è scaduto.

Quindi, non ci fermiamo, stiamo andando avanti ... aggiorna i token sulla strada da percorrere!

Aggiungiamo la funzionalità di aggiornamento token.

Ricordi ancora la nostra classe refreshToken? Questo è quello che stiamo usando ora.

Innanzitutto, impostiamo una funzionalità di database per aiutare a memorizzare il nostro `refreshToken.

Per fare ciò, creare una classe AppDbContext.cs e aggiornare il metodo ConfigureServices del file startup.cs

Quindi, aggiorniamo il nostro AccountController per includere un metodo RefreshToken aggiorna il nostro metodo di accesso e includiamo \_db nel controller:

Abbiamo quasi finito ...

La novità è questa:

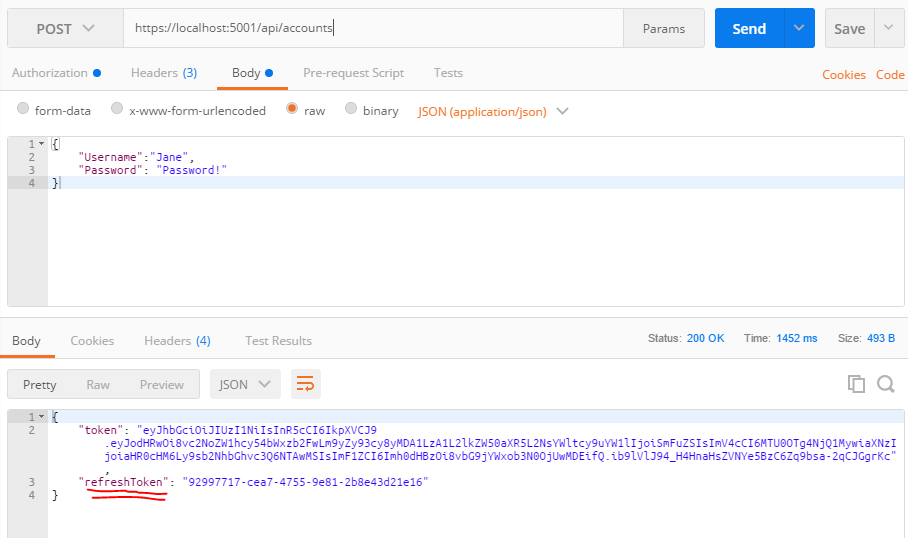
\_refreshTokenObj viene creato all'accesso e aggiunto al database

E per il metodo RefreshToken, il parametro di stringa refreshToken viene inviato ad esso e verifica se nel database è presente una corrispondenza, se esiste crea un nuovo JWT e un nuovo refreshToken e il ciclo inizia.

Ora proviamo di nuovo ...

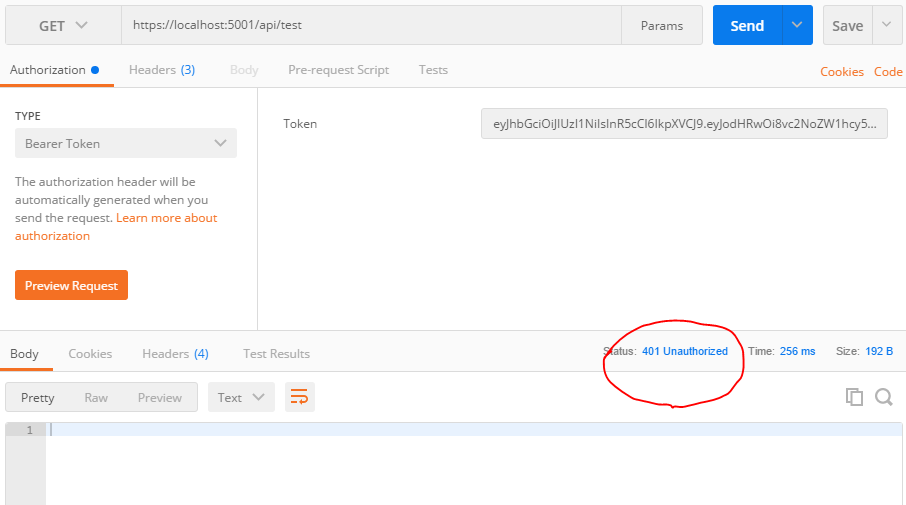
Prima prova il token di accesso precedente, dovrebbe darti un errore 401 non autorizzato

Quindi, eseguiamo di nuovo l'accesso, ma questa volta se il token di accesso scade utilizzeremo il token di aggiornamento ...

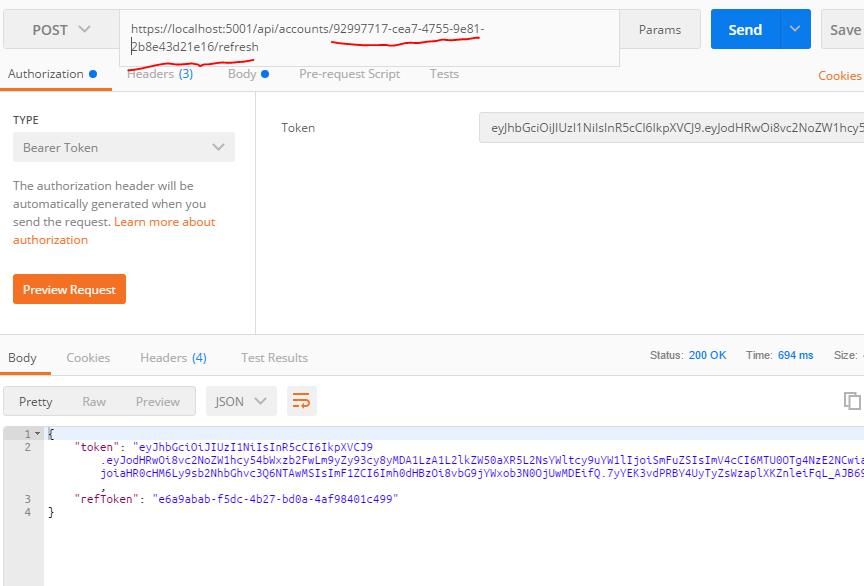


eccoci qui! sembra già vivo !!

Attendere circa 1 minuto, quindi provare a colpire il controller di test



andiamo sul percorso refreshToken {refreshToken} / refresh



lì mandiamo un JWT appena generato che possiamo usare per continuare ad accedere alla nostra API.

Conclusione

Questo è ovviamente un tutorial semplice JWT e token di aggiornamento, ma l'obiettivo qui è capire come funziona e non scrivere un codice pronto per la produzione.

Puoi armeggiarlo includendo cose come expirationTime per il token di aggiornamento o usando un algoritmo crittografato per generare una stringa refreshToken e non un Guid.

Il punto è che mi diverto. Ecco il link https://github.com/mrsimi/jwt-refreshtoken-webapi al codice completo su github.