# Authentication Process

### Process

When user completes the login process, and they are authenticated. Our Rest API will issue the client application an access token and a refresh token.

* An access token is given a short time before it expires for example 5 to 15 minutes.
* And a refresh token is given a longer duration before it expires possibly several hours, a day or even days.

### XSS and CSRF Attacks

While no security measures are perfect, we do want to consider the risks of cross-site scripting and cross-site request forgery.

References:

* https://owasp.org/www-community/attacks/xss/
* <https://owasp.org/www-community/attacks/csrf>

### Where to store access token?

Our API will send and receive access tokens as JSON data to avoid the previously mentioned risk. It is recommended for frontend client application to only store access tokens in memory so they will be automatically lost when the app is closed. They should not be stored in local storage or in a cookie. Essentially if you store it somewhere with JavaScript, a hacker can also retrieve it with JavaScript. Just keep access tokens in memory which you might also refer to as the current application state.

### Store refresh token in cookie

Our API will issue a refresh token in an httpOnly cookie. This type of cookie is not accessible with JavaScript. Refresh tokens do need to have expiration which will require users to log in again.

Refresh tokens should not have the ability to issue new refresh tokens because that essentially grants indefinite access if a refresh token falls into the wrong hands.

Refresh tokens are just like JWT access tokens but have different expiry time.

### Access token and refresh token overview

So, the overall access token process involves issuing an access token during user authorization. The user's application can then access our Rest API protected routes with the access tokens until it expires. Our API will verify the access token with middleware, every time the access token is used to make a request. When the access token does expire, the user's application will need to send their refresh token to our API refresh endpoint to get a new access token. Of course, the refresh token is also issued during user authorization.

Our Rest API refresh endpoint will verify the token and cross-reference the refresh token in our database too. Storing a reference to the refresh token in the database will allow refresh tokens to be terminated early if the user decides to logout and again refresh tokens need to be allowed to expire so indefinite access cannot be gained.

### Generate secrets for access token and refresh token

=> import secrets  
=> secrets.token\_urlsafe (128)  
=> Examples,

=> access\_token\_secret = \_0PZjqLMCOvzxgRGRvKfB5Bbw-ixpcRQlVwkne4aDvFX0c\_B1vqhNV6NTIzoscDEvGQWjHbUrV\_lukJUZBO4kcUk0BN5ERLoUjTa6yqaqpAsjpqBZoFNDlH47NC0Ht4rSc4UhWHTbJOfTe4B01KigCCXRNhdgHXUeIKTP9lKp2Y

=> refresh\_token\_secret = 0a7Bgfi46sEFLluJkwCa7rAnigg6cJ0IE4xWP3t2gYaxDDDk23-hj2vgfwjx5MvK7maOCSXqCCLImfkdaqcJLxP1b7ZennDi5Lv2UP0tN75sSwLSmqgoOfYwRgJ-uE4oiQtZy10axPtSsh2RvNXTyxoHxxlAy-XFnyeXFDJknUw

### Do not store access token in local storage, session storage

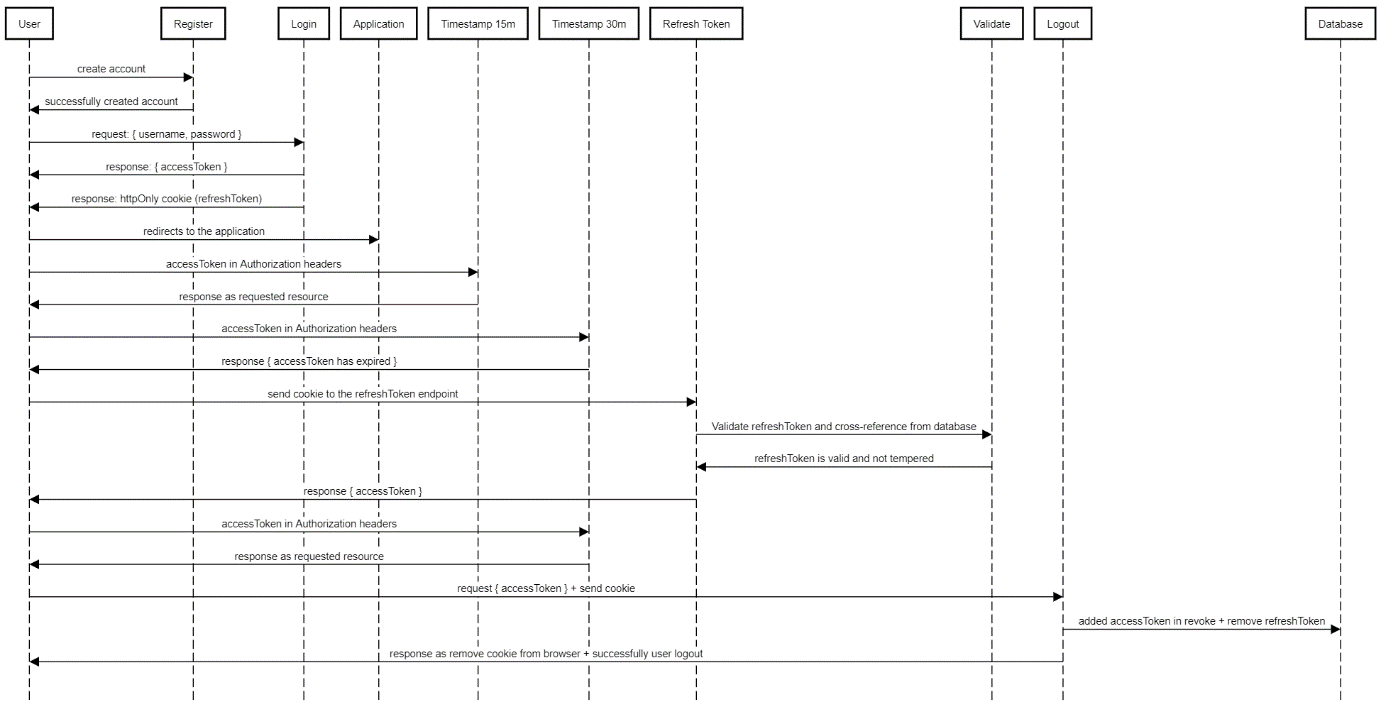
As a full stack developer, you really want to store the access token in memory. It’s not secure in local storage and any cookie that you can access with JavaScript. So, anything that really that JavaScript can access where you would store it, it’s not really that secure. So, if you store the access token in memory which has a very short life span, you are not storing in anywhere vulnerable. For the refresh token we want to send it as a cookie and I know that the cookie could be vulnerable to JavaScript but if we set the cookie as httpOnly, it is not accessible via JavaScript.

While it’s not hundred percent secure in cookie but it’s much more secure that storing your refresh token in local storage or cookie that is accessible from JavaScript.

### What happens if refresh token gets expired?

It means that user’s session has expired. Now user must re-authenticate again.

### Sequence diagram



#### Source code:

User -> Register : create account

Register -> User : successfully created account

User -> Login : request: { username, password }

Login -> User : response: { accessToken }

Login -> User : response: httpOnly cookie (refreshToken)

User -> Application: redirects to the application

User -> Timestamp 15m : accessToken in Authorization headers

Timestamp 15m -> User : response as requested resource

User ->Timestamp 30m : accessToken in Authorization headers

Timestamp 30m -> User : response { accessToken has expired }

User -> Refresh Token : send cookie to the refreshToken endpoint

Refresh Token -> Validate : Validate refreshToken and cross-reference from database

Validate -> Refresh Token : refreshToken is valid and not tempered

Refresh Token -> User : response { accessToken }

User -> Timestamp 30m : accessToken in Authorization headers

Timestamp 30m -> User : response as requested resource

User -> Logout : request { accessToken } + send cookie

Logout -> Database : added accessToken in revoke + remove refreshToken

Logout -> User : response as remove cookie from browser + successfully user logout