**Custom JWT Auth System**

*This document describes the step-by-step process on how the authentication system functions within the market.io Blazor web app.*

**Step 1: Registration -> Direct Login**

Before a user can login, they must already have in account it the database.

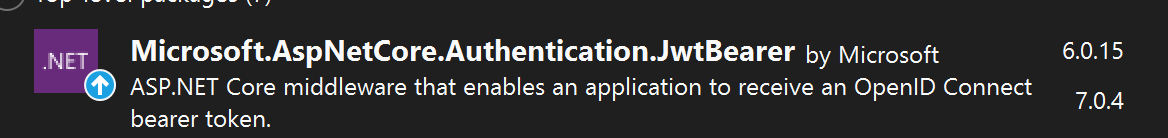
When a user completes the registration form or a user logs in with their credentials, an HTTP request is made. Inside the body of the request, the form data is sent to the web API controller where it runs validation on the user’s entered details. If validation is passed, the API will create the user inside of the database (if the user initially registered). Once registered, the user will be logged in automatically upon registration (login explained in next step)

**Step 2: Login and Token Generation**

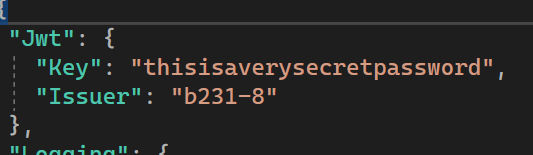
When a user logs in, a token is generated in the web API then stored in the browser’s session and cookie variables.

The ASP.NET Core Web API is configured with JWT Token generation, this is how it was setup:

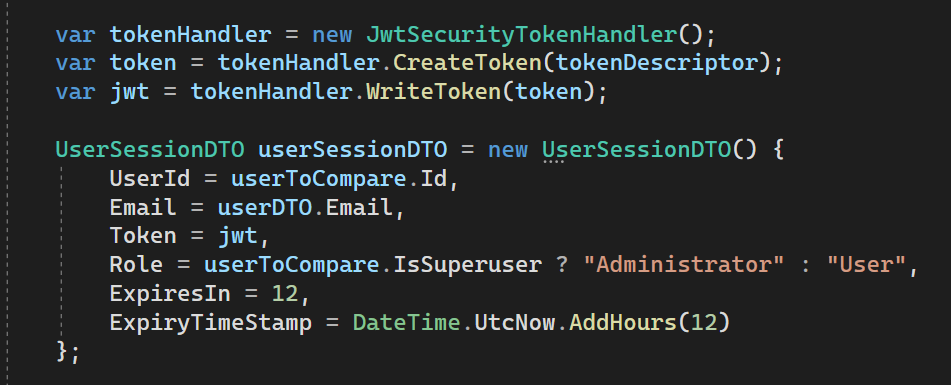
1. ASP.NET Core JWT Bearer package was installed



1. Secret key and issuer key was added to appSettings.JSON file

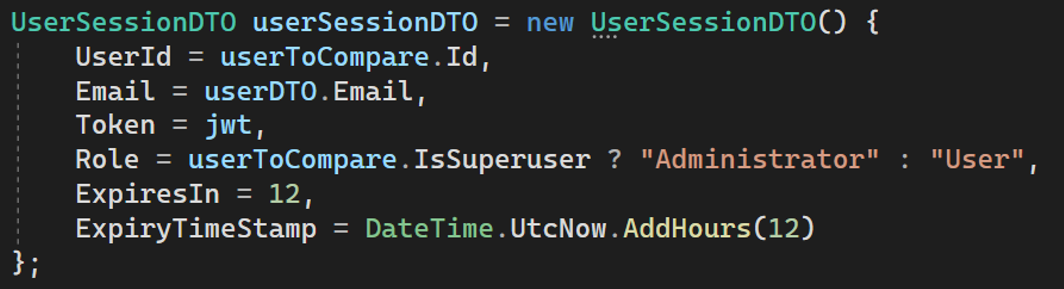


1. The login POST request handler method generates a json web token and embeds it into a token object (called UserSessionDTO) if the login credentials are valid



1. The UserSessionDTO object contains details about the session token including the user’s ID, email, role, token age, and the JWT that was just created.

This object is then sent back as an HTTP response.

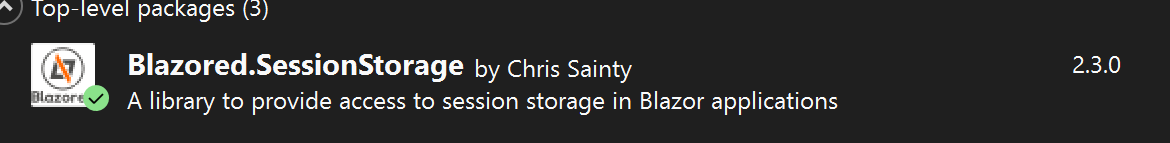


1. From there, the client app will receive the user session object and set it as a cookie / session variable inside of the user’s browser. (Explained in next step)

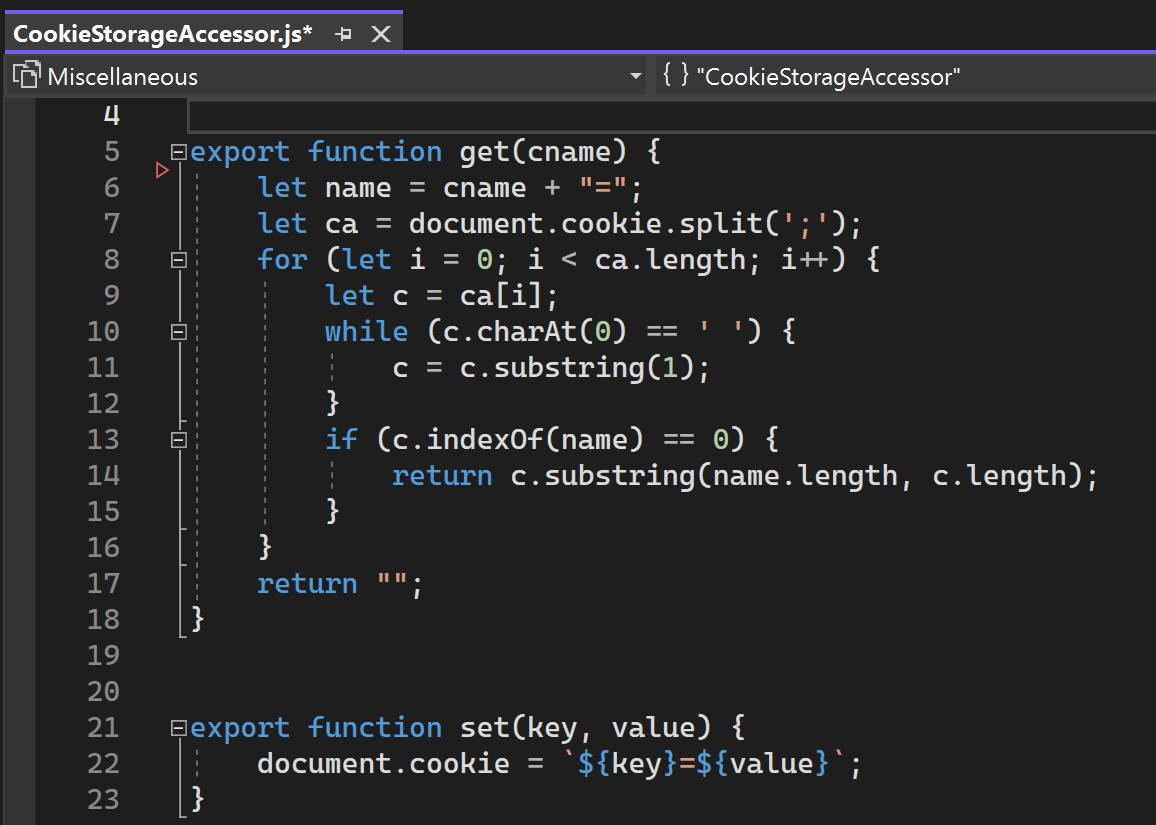
**Step 3: How the Client Handles the Token Object Upon Login**

2 Methods are used for JWT storage (Session and Cookie)

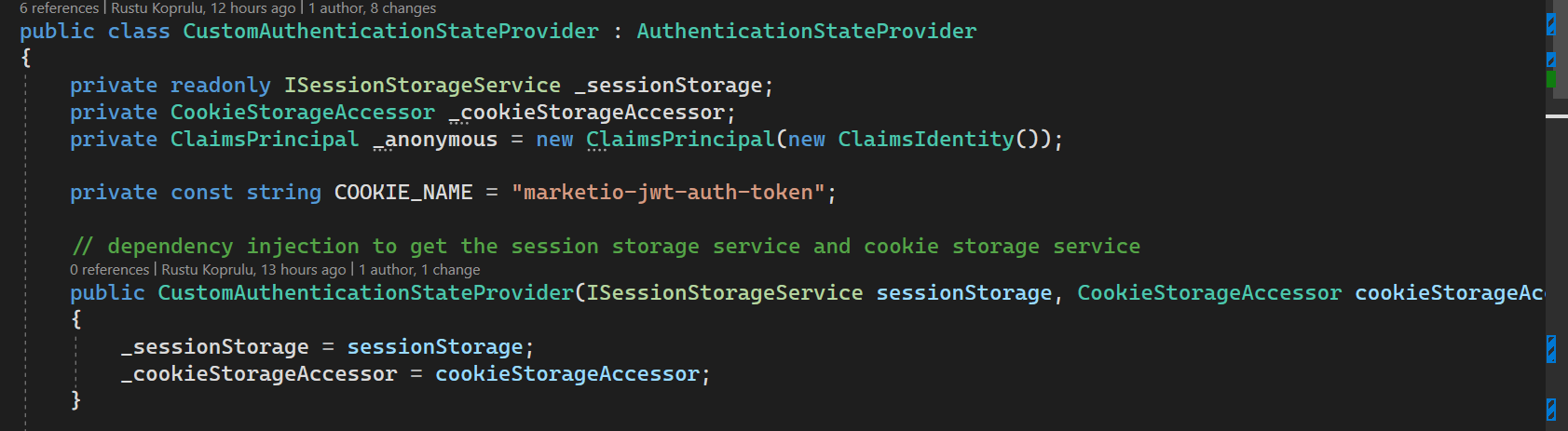
1. Blazored.SessionStorage



1. Javascript code for cookie logic to get and set token in browser cookie store



These two methods are responsible for storing the token locally, there is a class that manages authentication and authorization throughout the app.

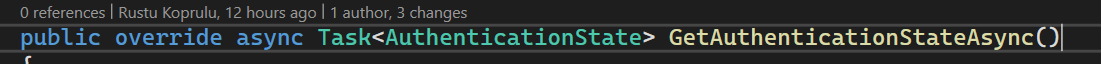


Quick notes:

* The class uses dependency injection to initialize the session and cookie storage helpers (ISessionStorageService from BlazoredStorage package, and CookieStorageAccessor which can call the JS function using C# - JavaScript runtime)
* ClaimsPrincipal is used to set the authentication state which is an ASP.NET built in system
* The \_anonymous object you see above is simply set when there is no token available (giving the user the un-authenticated state where they are not logged in)

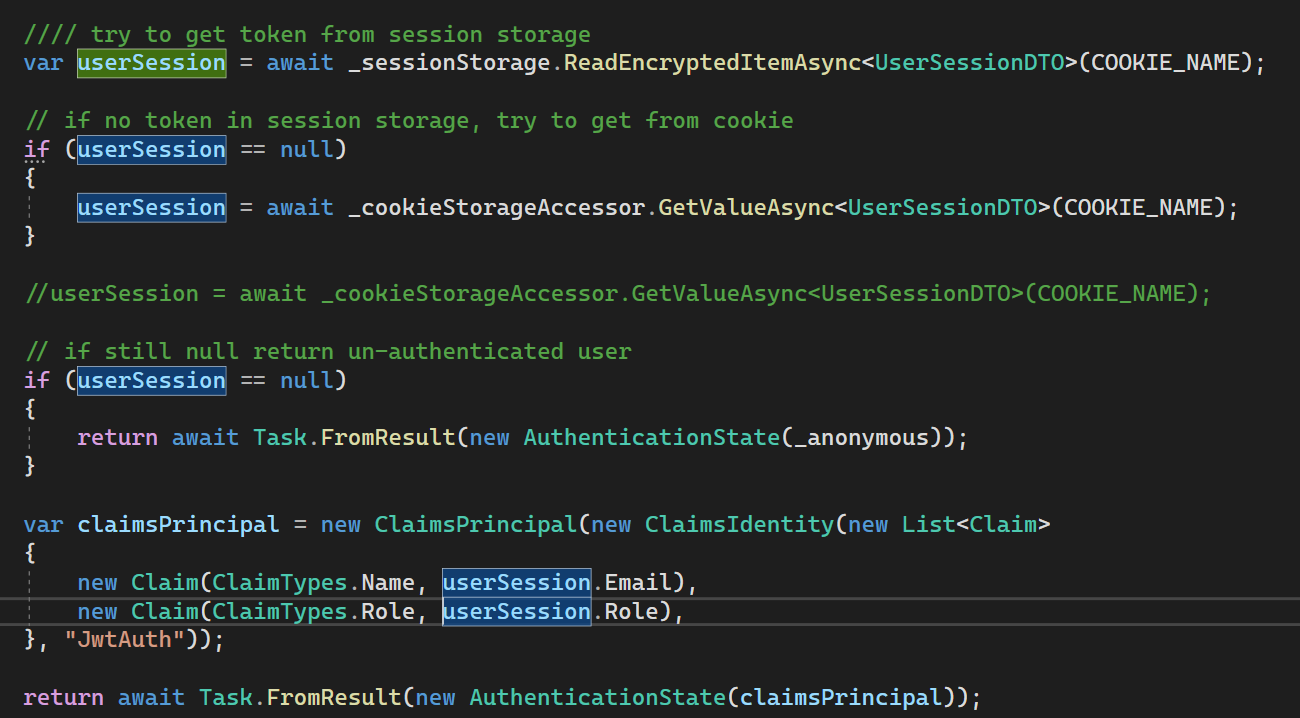
This class inherits from ‘AuthenticationStateProvider’ which is an ASP.NET class that handles authentication and authorization in the background.

Because I am inheriting from this class, I can override the ‘GetAuthenticationStateAsync()’



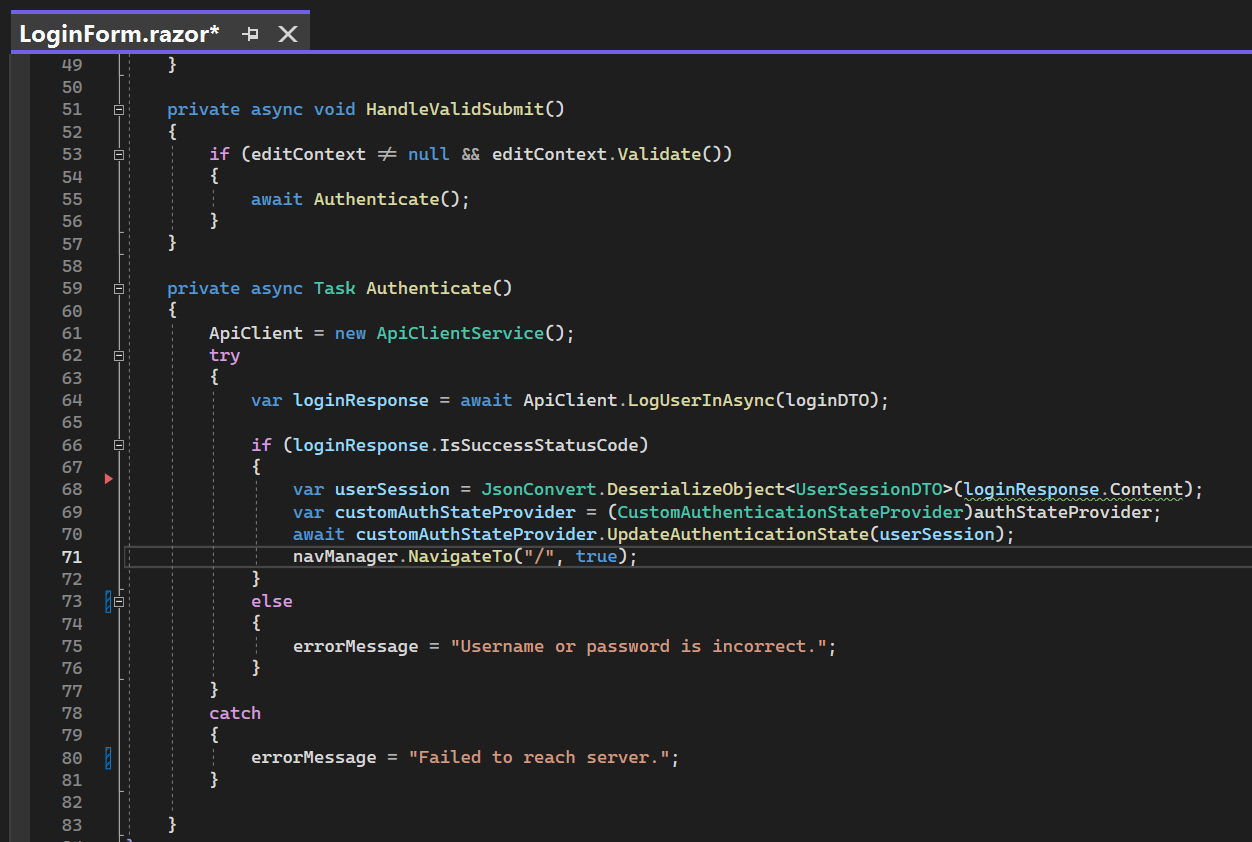
This method is automatically called before a user starts the Blazor site in their browser and returns an authentication state to the client app.

What my implementation is, is I am reading the token object from my cookie and session storage then setting the auth state using the data in the token object.



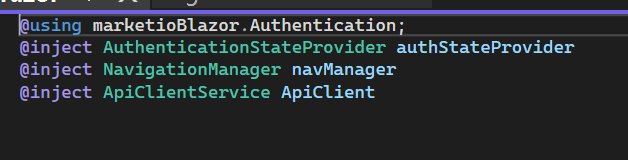
If a valid token is found, create a claim and set that that as the authentication state for the app. If not, the ‘\_anonymous’ user object is used.

**Examples: How This is Used Inside the App**

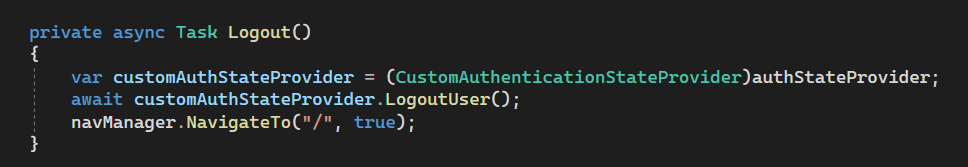
Example 1: Logging the user in:

1. When email + password is submitted, send a request to Login POST handler in API
2. Receive the UserSessionDTO object from API
3. Update the Auth state with UserSessionDTO object

Note: Make sure the following are injected into your razor component to use them

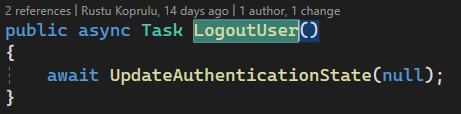


Example 2: Logging the user out



* This method clears the JWT token and returns the user to the landing page

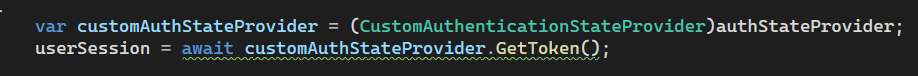
Closer look at LogoutUser() method:



* This simply passes in a null auth state which will automatically assign the \_anonymous Object if you recall from step 3.

Example 3: Getting the Users session object

If you want to access the session object (maybe to compare User Id, or email with other API data) here is how you do that:



This is the object that gets returned from GetToken() method:

