**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**

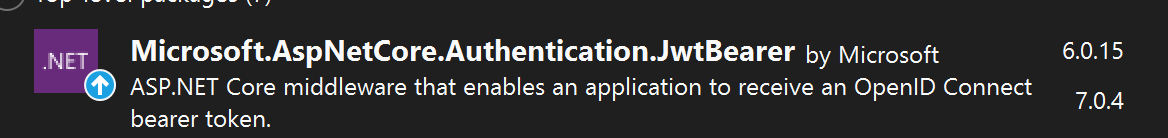
Before a user can login, you must have an account stored away in a database where you can access and compare to a user’s login request.

When a user completes the registration form from the client app’s register page, 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 users newly entered details. If validation is passed, the API will create the user inside of the database.

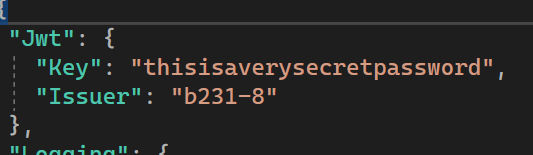
**Step 2: Login and Token Generation**

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

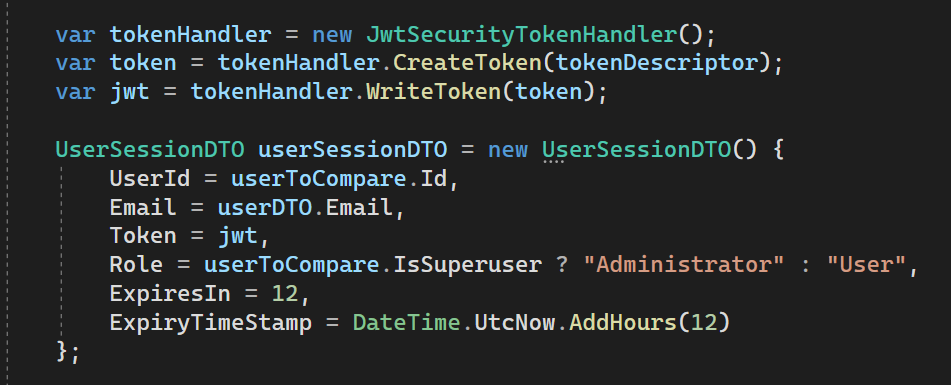
1. ASP.NET Core JWT Bearer packet was installed



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



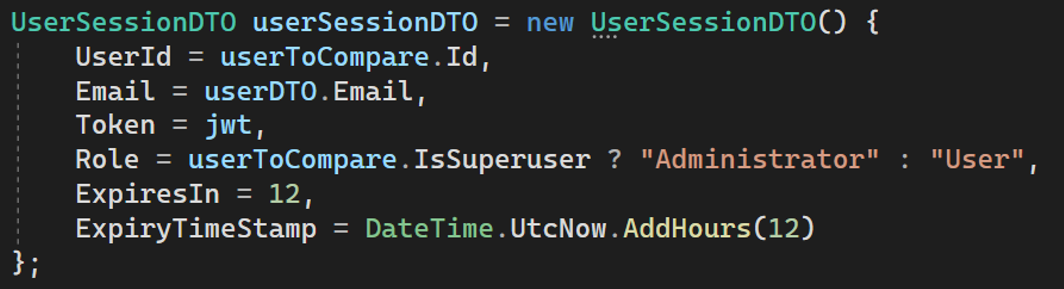
1. The login request handler method generates a json web token and embeds it into a token object (called UserSessionDTO)



Now to put it in action. When a user sends a login request from the client app to the API login handler, the credentials will be checked to see if the email and password matches a record in the database.

If it does, a json web token will be created, and will be stored in the UserSessionDTO object.

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



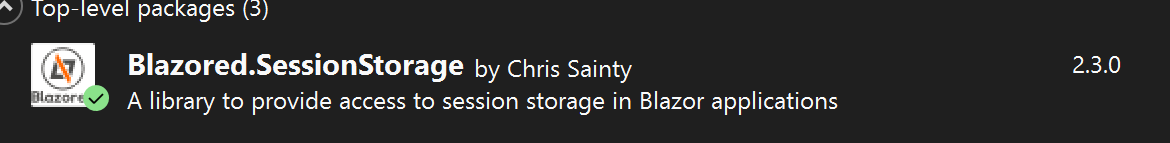
This object is then sent back as an HTTP response.

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.

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

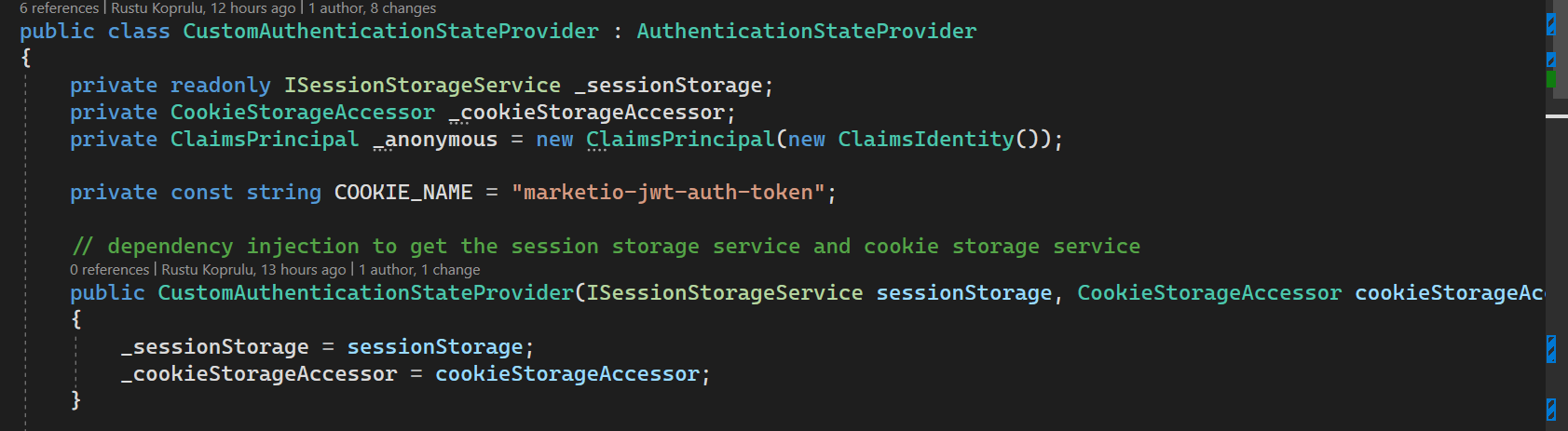
The client Blazor app has a few important dependencies that manage the token session.

1. Blazored.SessionStorage nuget package



1. C# 🡨🡪 JavaScript cookie logic to get and set token in browser cookie store

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

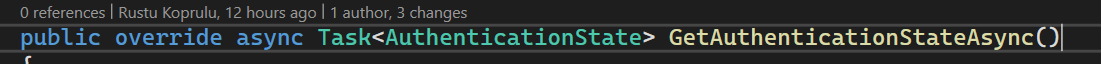


Quick notes:

* The class uses dependency injection to initialize the session and cookie storage helpers
* ClaimsPrincipal is used to set the authentication state which is an ASP.NET built in system
* The \_anonymous object you see above is simple set when there is no token available (giving the user the un authenticated state where they are not logged in)

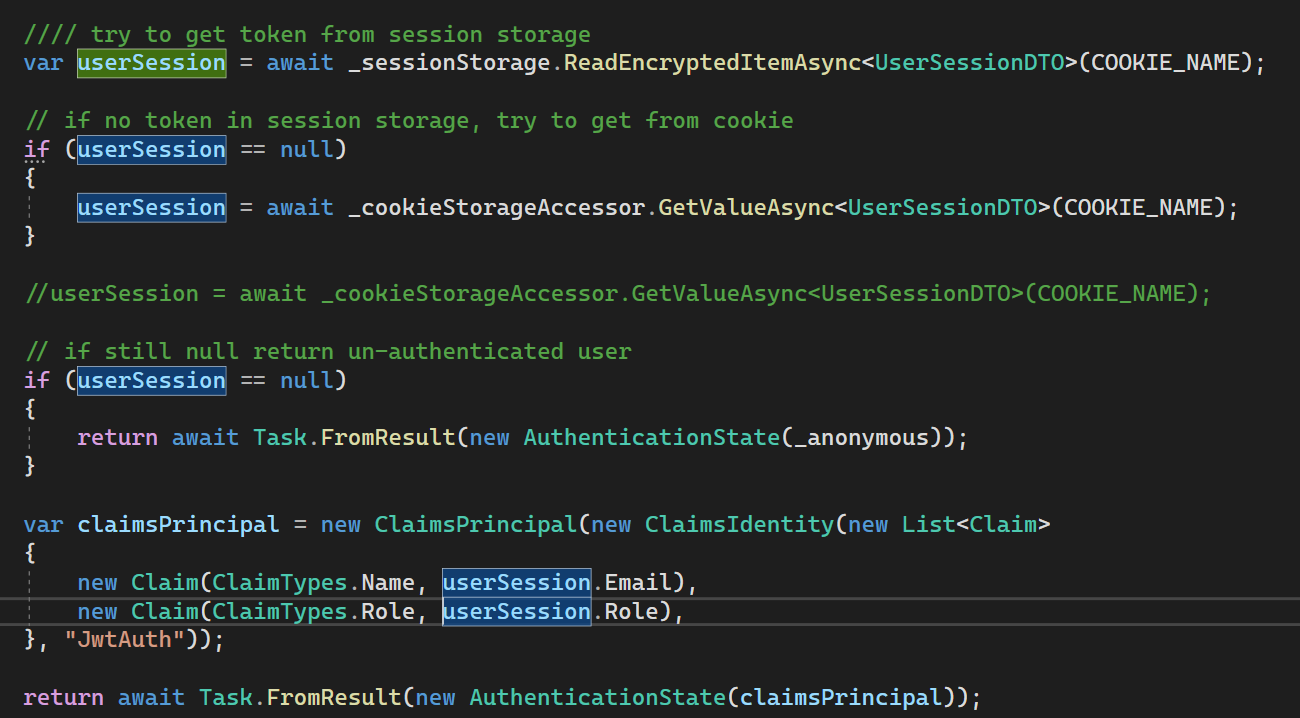
This class inherits from ‘AuthenticationStateProvider’ which is an ASP.NET class that handles authentication and authorization in the background.

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



This method is automatically called before a user loads in 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 object was found, create a claim and set that that as the authentication state for the app.

If not, the ‘\_anonymous’ user object is used.