Notes on securing Blazor UI

* Uses JSON Web Tokens as user identities enabling reuse of the Jwt code for recording signatures during analysis.
* Secret key in this example is held in appsettings.json, not built from GUIDs
* You will need to plug in your authentication component/REST+JSON proxy that implements the IAuthenticationService interface and uses the AuthenticationRequest and AuthenticationResponse DTOs as in the existing MARS application. Note that the namespaces are different in the example, as I don’t use your client proxy to an authentication service, but a simplified class library as a mock.
* The AuthenticationRequest DTO has been decorated with data annotations so that data validation and error messages can be displayed for it in the login form where it is the model for the EditForm component.
* In Program.cs, note the different list of usings for supporting authentication, plus the setup of injectable services for ProtectedSessionStorage, cascanding authentication state, the JwtAuthStateProvider as the custom AuthenticationStateProvider. I have also made my mock AuthenticationService be the injected IAuthenticationService, but you will want to inject your remote LDAP provider.
* Make sure the fields of the JWT token you create in the LDAP provider match the fields of my mock object in terms of what is validated and what is not. Your existing code base validates nothing at present (the wrong functions were called on the JWT library).
* I’ve increased the lifetime of the token from15 to 45 minutes.
* I’ve replaced the previous JWT library with the more up to date Microsoft.IdentityModel.JsonWebTokens library as the previous library was out of date and some of its functions about to be deprecated.
* Obviously other services and middleware setup in the MARS program.cs will have to be merged with the authentication services in my simplified Program.cs.
* The JwtAuthStateProvider class is very carefully designed. To keep the session available should the SignalR circuit be compromised, browser code in Blazor tries for up to 110 seconds to re-establish a broken connection. The state provider here stores the JwtToken (or lack of it) in a scoped variable inside the state provider, but once the first page has been rendered on a browser and the SIgnalR connection first established, it also saves the token into protected session storage on the browser. This makes it still available for when a session reconnects. It means that all the time the current tab is open in the browser, the session should remain alive. Sessions do not jump across tabs in the browser. You’ll have to use the other ProtectedBrowserStorage if you need that. Note the word ‘protected’. Data stored on the browser is encrypted in the server before storing in the browser, and decrypted after loading back into the server when retrieved.
* You will need to use the <Authorized> and <NotAuthorized> tags in pages and controls to prevent illegal access when not logged in.
* I’ve not built in the role support, but this is straightforward if roles are among the claims of your JWT tokens.