NTC Lego Technical Document for Search,

Authorization, and JWT functionality

Table of Contents

[General Overview 1](#_Toc121937208)

[Search 2](#_Toc121937209)

[Overview 2](#_Toc121937210)

[Functionality 2](#_Toc121937211)

[Items.razor.cs 2](#_Toc121937212)

[Search.cs 2](#_Toc121937213)

[Authorization 3](#_Toc121937214)

[Overview 3](#_Toc121937215)

[Functionality 3](#_Toc121937216)

[LogInLogOutButton.razor.cs 3](#_Toc121937217)

[Login.razor.cs 3](#_Toc121937218)

[CustomAuthStateProvider.cs 3](#_Toc121937219)

[JSON Web Tokens 4](#_Toc121937220)

[Overview 4](#_Toc121937221)

[Functionality 4](#_Toc121937222)

[JWTUtil.cs 4](#_Toc121937223)

# General Overview

This document will cover the functionality of the following topics:

* The search bar located in the Items.razor view
* How user roles and authorization are handled during Login/Logout
* The creation and storage of JSON Web Tokens

All the functionality described in this document was implemented by following along with these three tutorials – *THANK YOU Patrick God for these tutorials!*

Search - <https://www.youtube.com/watch?v=s7VVGe2D9lo>

Authentication - <https://www.youtube.com/watch?v=Yh16E2u2pio>

JWT/Registration/Login - <https://www.youtube.com/watch?v=v7q3pEK1EA0>

# Search

## Overview

Graphical user interface, text, application, email

Description automatically generatedText

Description automatically generated

The search bar located on the Items page is handled using the Blazored.Typeahead nuget package. The search functionality uses a string value so items can be searched by using either the ItemName, ItemId, or ItemTypeId.

## Functionality

### Items.razor.cs

In the code-behind for the Items.razor view, the SearchItem method will execute the search functionality handled by the Search class (see [Search.cs](#_Search.cs)), and the HandleSearch method will redirect the user to the ItemView page of the selected item. The <SelectedTemplate> tags and the other templates indicate what will appear in the search bar during a search. ‘SearchMethod’ executes the search functionality while there is text in the search bar. ‘TValue’ and ‘TItem’ describe the type of the object that is being searched for and ‘Value’ is the contents of the search bar. ‘ValueChanged’ executes the HandleSearch method and uses ‘ValueExpression’ to pass in the parameter for HandleSearch().

### Search.cs

The Search class acts as a service class to connect to the AdminController with an API call. The SearchItems method within the AdminController then calls the SearchItems method in the DataService class, which returns up to 10 results that meet the specified criteria.

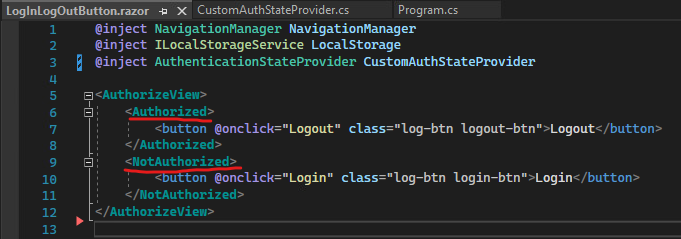
Text

Description automatically generated

# Authorization

## Overview

The roles and authorization of the users is authenticated via the CustomAuthStateProvider.cs found in the Client side of the project and works with the <AuthorizeView> tags throughout the project. The image below shows an example of the partial view for the Login and Logout buttons and how authorization will determine which button is shown.



## Functionality

### LogInLogOutButton.razor.cs

In the code-behind for the partial view LogInLogOutButton.razor, the Login method redirects the user to the login page (see [Login.razor.cs](#_Login.razor.cs)), and the Logout method will remove the JWT (‘token’) and the ‘userID’ from the applications local storage before updating the current AuthenticationState and redirecting back to the home page (see [CustomAuthStateProvider.cs](#_CustomAuthStateProvider.cs)).

### Login.razor.cs

In the code-behind for the Login.razor page, the HandleLogin() method is used to verify that the entered credentials exist/are valid, and then store their JWT (‘token’) and ‘userID’ in the local storage of the application. The ‘userID’ is used for identifying the appropriate cart, and the ‘token’ is used in the CustomAuthStateProvider class.

Table

Description automatically generated

### CustomAuthStateProvider.cs

The CustomAuthStateProvider class is made up of only 3 methods. GetAuthenticationStateAsync() is the main method that sets the authorization state of the current user based on the JWT token that is stored in the local storage of the application. It creates a ClaimsIdentity by parsing the stored JWT string and passing it into a new ClaimsPrinciple by using a key that is set in the appsettings.json file in the Server project folder (see [JWTUtil.cs](#_JWTUtil.cs)), which is finally used as the parameter for the AuthenticationState that is eventually returned to the views to determine the user’s authorization.

# JSON Web Tokens

## Overview

The JSON Web Tokens are encrypted and stored in the local storage of the application (see [Login.razor.cs](#_Login.razor.cs)) to be used for setting user roles and authorization during login. The JWT is created in the JWTUtil class by its only method called CreateToken and is currently set to expire after 24 hours.

## Functionality

### JWTUtil.cs

The JWTUtil class service as a utility class to create a JWT when a user logs in. It first creates a list of claims the ‘Name’ and ‘Role’ of the user. It then encrypts the signing credentials of the JWT with an additional password that is set as the ‘key’, which will be required in the ParseClaimsFromJwt method in the CustomAuthStateProvider class. *NOTE: There is currently no functionality in the program to set a user to an Admin role – this must be done in the database.*

JWTUtil.cs

Text

Description automatically generated

appsettings.json

Text

Description automatically generated

CustomAuthStateProvider.cs

Text

Description automatically generated