Assignment 12 – JWT  
COS318 – Web Programming

You are approaching another test chamber. Cake and grief counseling will be available at the conclusion of the assignment. There is a good chance that the cake is a lie. In this assignment you will be using JWT to validate requests against a server. Most of the JWT generation and validation code is provided for you. Your job is to use it in your controllers, filters, and javascript.

This assignment will have one WebApi project with two controllers and an html page with javascript that will make requests to the controllers. The javascript must keep track of a current JWT that can be sent along with requests to the server.

1. **(10 Points) UserDatabase/UserModel**
   1. Create a model class to store individual users and a database to store list of users. Users should be stored with two strings: username and password.
   2. The database should be initialized with a few usernames and passwords. One of those users must be the username “glados” with the password “cake”.
   3. UserDatabase should have a method for checking if a user exists in the database as well as a method to validate a user’s password.
2. **(20 Points) LoginController**
   1. Create a controller that will allow a user to login. It should accept POST requests with a username and password sent as JSON.
   2. The LoginController should check if the username and password combination being requested exists in the database.
      1. If the user doesn’t exist or their password doesn’t match, return appropriate status codes.
      2. If the user does exist, use ISecurityProvider (found on the class github) to generate and return a token to the user in a JSON document with a single key, named “token”. The token should be generated with a single claim, “username” with a value equal to the username sent to the server.
3. **(10 Points) GladosController**
   1. Create a controller that has a single GET method which returns a random quote from Glados.
      1. There should be at least five quotes. Hint: Don’t do too much work in the controller here. Use a service.
   2. Requests to GladosController should only succeed if there is a valid token sent in the request in the appropriate header and format.
4. **(20 Points) AuthorizationFIlter**
   1. Create a filter that will validate tokens sent in the Authorization header. Verify the header is in the correct format and use ISecurityProvider to validate the token. Stop the request and return the appropriate status code if the token is not validated.
   2. This filter should only be applied to the GladosController.
5. **(20 Points) Html and Javascript**
   1. Create an html page that can make requests to the server. It should have buttons for Login, Logout, and Quote. It should also have appropriate input fields for username and password.
      1. Login should send username and password to the LoginController. On a successful response, it should store the token in the browser’s local storage. Hint: local storage is not the same thing as a javascript variable. Local storage is persisted even if the browser is closed and reopened.
      2. Quote should request a quote from GladosController. Since GladosController is protected with AuthorizationFilter, you will need to send along the token in the Authorization header.
      3. Logout should delete the stored token out of the local storage.
   2. The html page should display if the user is currently logged in.
   3. The html page must show error messages and status codes when requests fail (such as not having a correct token)
   4. Store the user’s login token using local storage.
6. **(20 Points)** Code style, formatting, completeness, and quality.

Stretch Levels

If you already have a lot of experience with JWT and authorization, or if you just want to use momentum to your advantage to reach high ledges, try to complete these stretch levels for a reputation bonus. If you try for the stretch levels, make sure to type it in the comments on Moodle so I don’t miss it.

**Glados Level**

Add some CSS to your page to make it look nicer. Background colors, font colors, or anything that looks good.

**Wheatley Level**

Modify your server to use a constant public and private key (Hint: from an xml string) so that if your server is restarted, your token in local storage will continue to be valid (until it expires).

**Companion Cube Level**

Update SecurityProvider to support validation of another issuer, “www.tokensareneat.com”. When tokens are created, randomly choose which issuer is saved in the token. Make sure both issuers validate correctly when they are in tokens. Do the same thing with audience but use “www.ilovetokens.com” for the second audience.

The Rules

1. No inline styles or inline javascript.
2. Error messages must be “in-page” i.e. no pop-ups or alerts.
3. Any resources not created by you (images, javascript libraries, etc.) must be referenced using a CDN or URL, not directly included in your assignment submission.
4. All requests that submit a body to your server must have their entities validated with appropriate annotations, such as MinLength, Range, or Required.
5. The root path of your server must display the main page of your application.
6. Service/data/model classes must not have any http, request, or response references.
7. Controller entity classes must not be used directly to store data on the server; translate them into a model (data storage) class before saving the data. Conversely, controllers must not send any model classes to the user; translate them into controller entity classes before sending the response.
8. All service class instances must be obtained using dependency injection.
9. You may not use any synchronous methods in your C# code wherever there is an async option.
10. All controllers (and their corresponding entities) must enforce the usage of an api version. Your namespace and folder structure for controllers and entities must contain the api version.