LAB1: Implementing Token Based Authentication in ASP.NET Core Web API

Task 1: In the appSettings.json add the secret key for token

```
"JWTSettings": {
    "SecretKey":
"0++yXoea/d3u/g0oTzDiuY+HZgzdF2RftLFIljv91T59xN/Fscyy8WoIhBYT7ld3HJ1gPqMAxs1sYxMRD4Rv
XQ==",
    "ExpityInMinuts": 20
  }
```

Task 2: Create a new ASP.NET Core Application and add the following code in ConfigureServices() method for JWT Validation

```
services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
            .AddJwtBearer(options =>
            {
                var secretKey =
Convert.FromBase64String(Configuration["JWTSettings:SecretKey"]);
                options.TokenValidationParameters = new TokenValidationParameters
                {
                    ValidateIssuer = false,
                    ValidateAudience = false,
                    ValidateIssuerSigningKey = true,
                    IssuerSigningKey = new SymmetricSecurityKey(secretKey)
                };
            });
```

Task 3: In the Models folder add a new class file and name it as UserModel.cs and add following classes in it

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Threading.Tasks;

namespace Core_AppJWT.Models
{


    public class LoginUser
    {
        [Required(ErrorMessage ="User Name is Must")]
        public string UserName { get; set; }

        [Required(ErrorMessage = "Password is Must")]
        public string Password { get; set; }
    }

    public class RegisterUser
    {
```

```
        [Required(ErrorMessage = "Email is Must")]
        [EmailAddress]
        public string Email { get; set; }
        [Required(ErrorMessage = "Password is Must")]
        [RegularExpression("^((?=.*?[A-Z])(?=.*?[a-z])(?=.*?[0-9])|(?=.*?[A-
Z])(?=.*?[a-z])(?=.*?[^a-zA-Z0-9])|(?=.*?[A-Z])(?=.*?[0-9])(?=.*?[^a-zA-Z0-
9])|(?=.*?[a-z])(?=.*?[0-9])(?=.*?[^a-zA-Z0-9])).{8,}$",
            ErrorMessage = "Passwords must be minimum 8 characters and can
contain upper case, lower case, number (0-9) and special character")]
        public string  Password { get; set; }
        [Compare("Password")]
        public string ConfirmPassword { get; set; }
    }
}
```

The above classes are used for Registering new user and authenticating it.

Task 4: In the project add a new folder of name Services and add a new class file of name **AuthenticationServices.cs** add the following code in it

```
using Core_AppJWT.Models;
using Microsoft.AspNetCore.Identity;
using Microsoft.Extensions.Configuration;
using Microsoft.IdentityModel.Tokens;
using System;
using System.Collections.Generic;
using System.IdentityModel.Tokens.Jwt;
using System.Linq;
using System.Security.Claims;
using System.Threading.Tasks;


namespace Core_AppJWT.Services
{
    /// <summary>
    /// The followint class is used for the following
    /// 1. The class will accept Login Details and then Issue the token based
on the login Status
    ///
    /// </summary>
    public class AuthenticationService
    {
        IConfiguration configuration;
        SignInManager<IdentityUser> signInManager;
        UserManager<IdentityUser> userManager;
        public AuthenticationService(IConfiguration configuration,
            SignInManager<IdentityUser> signInManager,
            UserManager<IdentityUser> userManager)
        {
            this.configuration = configuration;
            this.signInManager = signInManager;
            this.userManager = userManager;
        }

        public bool RegisterUser(RegisterUser register)
```

```
        {
            bool IsCreated = false;
            var registerUser = new IdentityUser() { UserName =
register.Email, Email = register.Email };
            var result = userManager.CreateAsync(registerUser,
register.Password).Result;
            if (result.Succeeded)
            {
                IsCreated = true;
            }
            return IsCreated;
        }

        public string Authenticate(LoginUser inputModel)
        {
            string jwtToken ="";
            var result =
signInManager.PasswordSignInAsync(inputModel.UserName, inputModel.Password,
false, lockoutOnFailure: true).Result;
            if (result.Succeeded)
            {
                // Read the secret key and the expiration from the
configuration
                var secretKey =
Convert.FromBase64String(configuration["JWTSettings:SecretKey"]);
                var expiryTimeSpan =
Convert.ToInt32(configuration["JWTSettings:ExpityInMinuts"]);
                // logic to get the user role
                // get the user object based on Email
                IdentityUser user = new IdentityUser(inputModel.UserName);


                // set the expity, subject, etc.
                // note that Issuer and Audience will be null because
                // there is no third-party issuer
                var securityTokenDescription = new SecurityTokenDescriptor()
                {
                    Issuer = null,
                    Audience = null,
                    Subject = new ClaimsIdentity(new List<Claim> {
                        new Claim("username",user.Id,ToString()),
                    }),
                    Expires = DateTime.UtcNow.AddMinutes(expiryTimeSpan),
                    IssuedAt = DateTime.UtcNow,
                    NotBefore = DateTime.UtcNow,
                    SigningCredentials = new SigningCredentials(new
SymmetricSecurityKey(secretKey), SecurityAlgorithms.HmacSha256Signature)
                };
                // Now generate token using JwtSecurityTokenHandler
                var jwtHandler = new JwtSecurityTokenHandler();
                var jwToken =
jwtHandler.CreateJwtSecurityToken(securityTokenDescription);
                jwtToken = jwtHandler.WriteToken(jwToken);
            }

            return jwtToken;
        }
```

```
    }
}
```

The above code uses ASP.NET Core Identity for Registering new user and authenticating it. The Authenticate method uses SecurityTokenDescriptor class to issue token based on claim information send by user.

Task 5: In the Models folder add a new class file and name it as ResponseData.cs and add a following class in it

```
public class ResponseData
    {
        public string Message { get; set; }
    }
```

The above class will be used for responding response to client.

Task 6: In the controllers folder add a new WEB API Controller and add the following code in it

```
using Core_AppJWT.Models;
using Core_AppJWT.Services;
using Microsoft.AspNetCore.Mvc;

namespace Core_AppJWT.Controllers
{
    [Route("api/[controller]/[action]")]
    [ApiController]
    public class AuthenticationController : ControllerBase
    {
        AuthenticationService authenticationService;
        public AuthenticationController(AuthenticationService
authenticationService)
        {
            this.authenticationService = authenticationService;
        }


        [HttpPost]
        public IActionResult Register(RegisterUser user)
        {
            if (ModelState.IsValid)
            {
                var IsCreated = authenticationService.RegisterUser(user);
                if (IsCreated == false)
                {
                    return Conflict("The User Already Present");
                }
                var ResponseData = new ResponseData()
                {
                     Message = $"{user.Email} User Created Successfully"
                };
                return Ok(ResponseData);
            }
            return BadRequest(ModelState);
        }
```

```
        [HttpPost]
        public IActionResult Login(LoginUser inputModel)
        {
            if (ModelState.IsValid)
            {
                var token = authenticationService.Authenticate(inputModel);
                if (token == null)
                {
                    return Unauthorized("The Authentication Failed");
                }
                var ResponseData = new ResponseData()
                {
                    Message = token
                };
                return Ok(ResponseData);
            }
            return BadRequest(ModelState);
        }
    }
}
```

The above controller contains action methods for Registering new user and managing login. The Login action method returns the token to the client. The client can use this token to authenticate user.

Task 7: In the Api Controller of the application apply the**[Authorize]** attribute and make http request to the action method of this controller by sending the token in the header.