ASP.NET CORE WEB API WITH JWT

Introduction

JWT (JSON web token) become more and more popular in web development. It is an open standard which allows transmitting data between parties as a JSON object in a secure and compact way. The data transmitting using JWT between parties are digitally signed, so that it can be easily verified and trusted.

How to setup JWT with ASP.net core web application. We can create application using Visual Studio or using CLI (command Line Interface).

dotnet new webapi -n JWTAuthentication

Above command will create an ASP.net web API project with name "JWTAuthentication" in the current folder.

The first step is to configure JWT based authentication in our project. To do this, we need to register JWT authentication schema by using "AddAuthentication" method and specifying JwtBearerDefaults.AuthenticationScheme. Here, we configure the authentication schema with JWT bearer options.

```
public void ConfigureServices(IServiceCollection services)
{
   services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
   .AddJwtBearer(options =>
   {
      options.TokenValidationParameters = new TokenValidationParameters
      {
         ValidateIssuer = true,
         ValidateAudience = true,
         ValidateLifetime = true,
         ValidateIssuerSigningKey = true,
         ValidIssuer = Configuration["Jwt:Issuer"],
         ValidAudience = Configuration["Jwt:Issuer"],
            IssuerSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(Configuration["Jwt:Key"]))

        };

    });

    services.AddMvc();

}
```

In this example, we have specify which parameters must be taken into account to consider JWT as a valid. As per our code, following items consider a token valid:

* Validate the server (ValidateIssuer = true) that generate the token

* Validate the recipient of token is authorized to receive (ValidateAudience = true)

* Check token is not expire and the signing key of the issuer is valid (ValidateLifetime = true)

* Validate signature of the token (ValidateIssuerSigningKey = true)

* Additionally, we specify the values for the issuer, audience, signing key. In this example, we have stored these values in appsettings.json file.

AppSetting.Json

```
{

  "Jwt": {

    "Key": "ThisismySecretKey",

    "Issuer": "Test.com"

  }

}
```

Above mention steps used to configure JWT based authentication service. Next step is to make the authentication service available to the application. To do this, we need to call app.UseAuthentication() method in Configure method of startup class. The UseAuthentication method call before UseMvc method.

```
```

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)

{

    app.UseAuthentication();

    app.UseMvc();

}
```

Generate JSON Web Token

We have created LoginController and Login method within this controller, that responsible to generate the JWT. We have marked this method with AllowAnonymous attribute to bypass the authentication. This method expects Usermodel object for Username and Password.

We have created "AuthenticateUser" method, that is responsible to validate the user credential and returns the UserModel. For Demo pupose, We have returns hardcode model if username is "Tapan". If "AuthenticateUser" method returns user model, API generates the new token by using "GenerateJSONWebToken" method.

Here, We have created a JWT using the JwtSecurityToken class. We have created object of this class by passing some parameter to the constructor such as issuer, audience , expiration, and signature.

Finally, JwtSecurityTokenHandler.WriteToken method is used to generate the JWT. This method expects object of JwtSecurityToken class.

```
using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.Configuration;

using Microsoft.IdentityModel.Tokens;

using System;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;
using System.Text;

namespace JWTAuthentication.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class LoginController : Controller
    {
        private IConfiguration _config;

        public LoginController(IConfiguration config)
        {
            _config = config;
        }
        [AllowAnonymous]
        [HttpPost]
        public IActionResult Login([FromBody]UserModel login)
        {
            IActionResult response = Unauthorized();
            var user = AuthenticateUser(login);

            if (user != null)
            {
                var tokenString = GenerateJSONWebToken(user);
                response = Ok(new { token = tokenString });
            }

            return response;
        }


        private string GenerateJSONWebToken(UserModel userInfo)
        {
            var securityKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(_config["Jwt:Key"]));
            var credentials = new SigningCredentials(securityKey,
SecurityAlgorithms.HmacSha256);


            var token = new JwtSecurityToken(_config["Jwt:Issuer"],
              _config["Jwt:Issuer"],
              null,
              expires: DateTime.Now.AddMinutes(120),
              signingCredentials: credentials);


            return new JwtSecurityTokenHandler().WriteToken(token);
        }


        private UserModel AuthenticateUser(UserModel login)
        {
            UserModel user = null;


            //Validate the User Credentials
            //Demo Purpose, We have Passed HardCoded User Information
            if (login.Username == "Tapan")
            {
                user = new UserModel { Username = "Tapan", EmailAddress =
"tapanmentor@gmail.com" };
            }
            return user;
        }
    }
}
```

Once we have enable the JWT based authentication, We have created a simple Web API method that returns a list of value string when invoke with HTTP GET request. Here, We have marked this method with Authorize attribute, so that this endpoint will trigger the validation check of the token passed with HTTP request.

If we call this method without token, we will get 401 (UnAuthorizedAccess) HTTP status code as a response. If we want to bypass the authentication for any method, we can mark that method with AllowAnonymous attribute.

To test the created Web API, We can Use Fiddler (https://www.telerik.com/fiddler) or Postman. First, We have requested to "api/login" method to generate the token. We have pass following, JSON in request body.

```
{"username": "Tapan", "password": "password"}
```

As a response, we will get the JSON like following:

```
{
        "token" :
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJKaWduZXNoIFRyaXZlZGkiLCJl
bWFpbCI6InRlc3QuYnRlc3RAZ21haWwuY29tIiwiRGF0ZU9mSm9pbmciOiIwMDAxLTA
xLTAxIiwianRpIjoiYzJkNTZjNzQtZTc3Yy00ZmUxLTgyYzAtMzlhYjhmNzFmYzUzIiwiZXh
wIjoxNTMyMzU2NjY5LCJpc3MiOiJUZXN0LmNvbSIsImF1ZCI6IlRlc3QuY29tIn0.8hwQ3
H9V8mdNYrFZSjbCpWSyR1CNyDYHcGf6GqqCGnY"
}
```

Now We will try to get list of the values by passing this token into authentication HTTP header. Following is my Action method defination.

```
[HttpGet]
[Authorize]
```

```
public ActionResult<IEnumerable<string>> Get()

{

    return new string[] { "value1", "value2", "value3", "value4", "value5" };

}
```

```
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJKaWduZXNoIFRyaXZlZGkiLCJlb
WFpbCI6InRlc3QuYnRlc3RAZ21haWwuY29tIiwiRGF0ZU9mSm9pbmciOiIwMDAxLTAx
LTAxIiwianRpIjoiYzJkNTZjNzQtZTc3Yy00ZmUxLTgyYzAtMzlhYjhmNzFmYzUzIiwiZXh
wIjoxNTMyMzU2NjY5LCJpc3MiOiJUZXN0LmNvbSIsImF1ZCI6IlRlc3QuY29tIn0.8hwQ3
H9V8mdNYrFZSjbCpWSyR1CNyDYHcGf6GqqCGnY
```

Handle Claims with JWT

Claims is data that contains by the token. They are information about the user which help us to authorize the access to a resource. They could be User name, email address, role or any other information. We can add claims information to the JWT so that they are available when checking for authorization.

In above example, If we want to pass the claims to our token then the claim information need to add GenerateJSONWebToken method of Login controller. In following example, We have added use name, email address and date of joining as claim into the token.

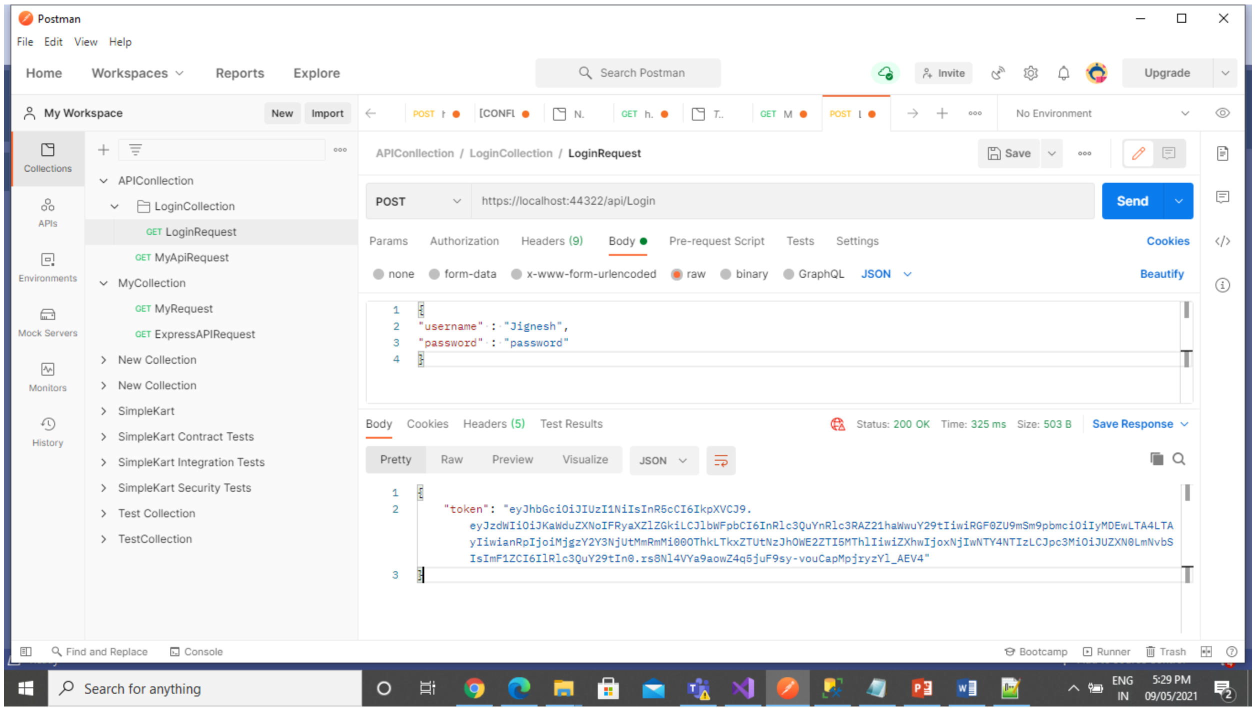
```
private string GenerateJSONWebToken(UserModel userInfo)

{

    var securityKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(_config["Jwt:Key"]));

    var credentials = new SigningCredentials(securityKey,
SecurityAlgorithms.HmacSha256);
```

```
    var claims = new[] {
        new Claim(JwtRegisteredClaimNames.Sub, userInfo.Username),
        new Claim(JwtRegisteredClaimNames.Email, userInfo.EmailAddress),
        new Claim("DateOfJoing", userInfo.DateOfJoing.ToString("yyyy-MM-dd")),
        new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString())
    };

    var token = new JwtSecurityToken(_config["Jwt:Issuer"],
        _config["Jwt:Issuer"],
        claims,
        expires: DateTime.Now.AddMinutes(120),
        signingCredentials: credentials);

    return new JwtSecurityTokenHandler().WriteToken(token);
}
```

The claims are an array of key-value pair. The keys may be values of a JwtRegisteredClaimNames structure (it provides names for public standardized claims) or custom name (such as DateOfJoining in above example).

This claims can be used in filter the data. In following example, We have change the list of values if user spends more than 5 years with company.

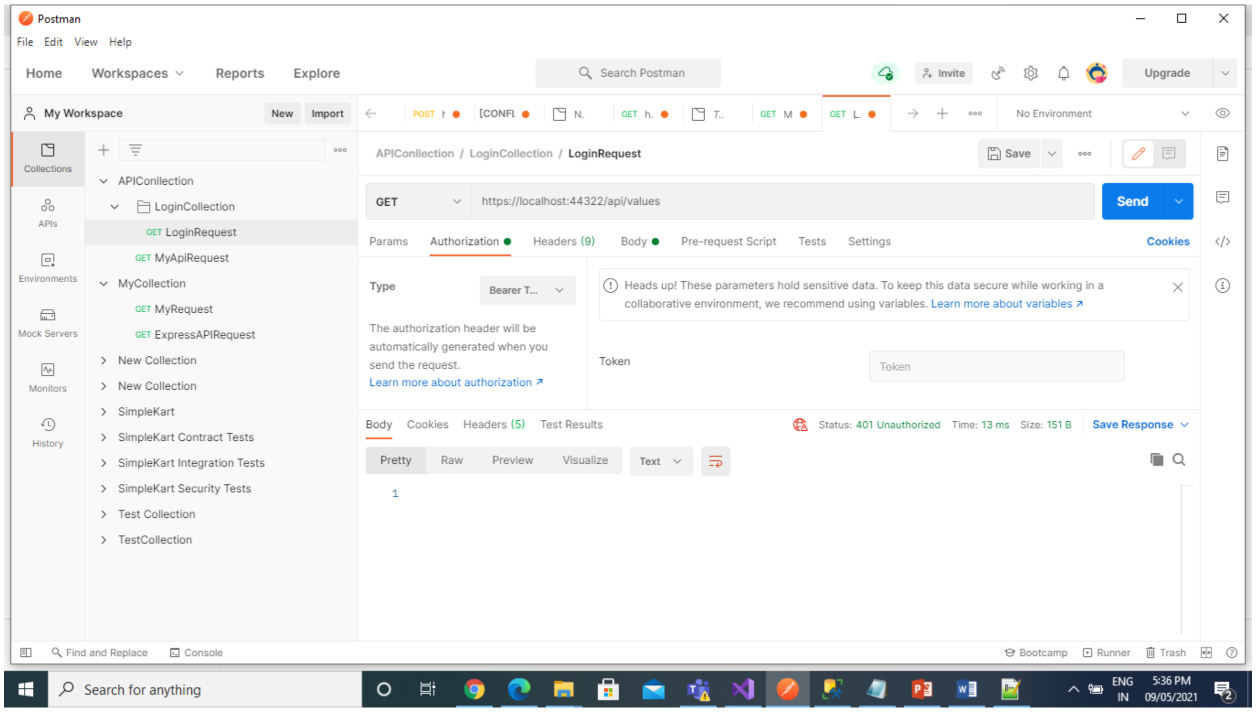
```
[HttpGet]
[Authorize]
public ActionResult<IEnumerable<string>> Get()
{
    var currentUser = HttpContext.User;
    int spendingTimeWithCompany = 0;


    if (currentUser.HasClaim(c => c.Type == "DateOfJoing"))

    {

        DateTime date = DateTime.Parse(currentUser.Claims.FirstOrDefault(c => c.Type
== "DateOfJoing").Value);

        spendingTimeWithCompany = DateTime.Today.Year - date.Year;

    }


    if(spendingTimeWithCompany > 5)

    {

        return new string[] { "High Time1", "High Time2", "High Time3", "High Time4",
"High Time5" };

    }

    else

    {

        return new string[] { "value1", "value2", "value3", "value4", "value5" };

    }

}
```


POST https://localhost:44322/api/Login

```
{
"username" : "Jignesh",
"password" : "password"
}
```

Status: 200 OK Time: 325 ms Size: 503 B

```
{
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
        eyJzdWIiOiJKaWduZXNoIFRyaXZlZGkiLCJlbWFpbCI6InRlc3QuYnRlc3RAZ21haWwuY29tIiwiRGF0ZU9mSm9pbmciOiIyMDEwLTA4LTA
        yIiwianRpIjoiMjgzY2Y3NjUtMmRmMi00OThkLTkxZTUtNzJhOWE2ZTI5MThlIiwiZXhwIjoxNjIwNTY4NTIzLCJpc3MiOiJUZXN0LmNvbS
        IsImF1ZCI6IlRlc3QuY29tIn0.rs8Nl4VYa9aowZ4q5juF9sy-vouCapMpjryzYl_AEV4"
}
```


GET https://localhost:44322/api/values

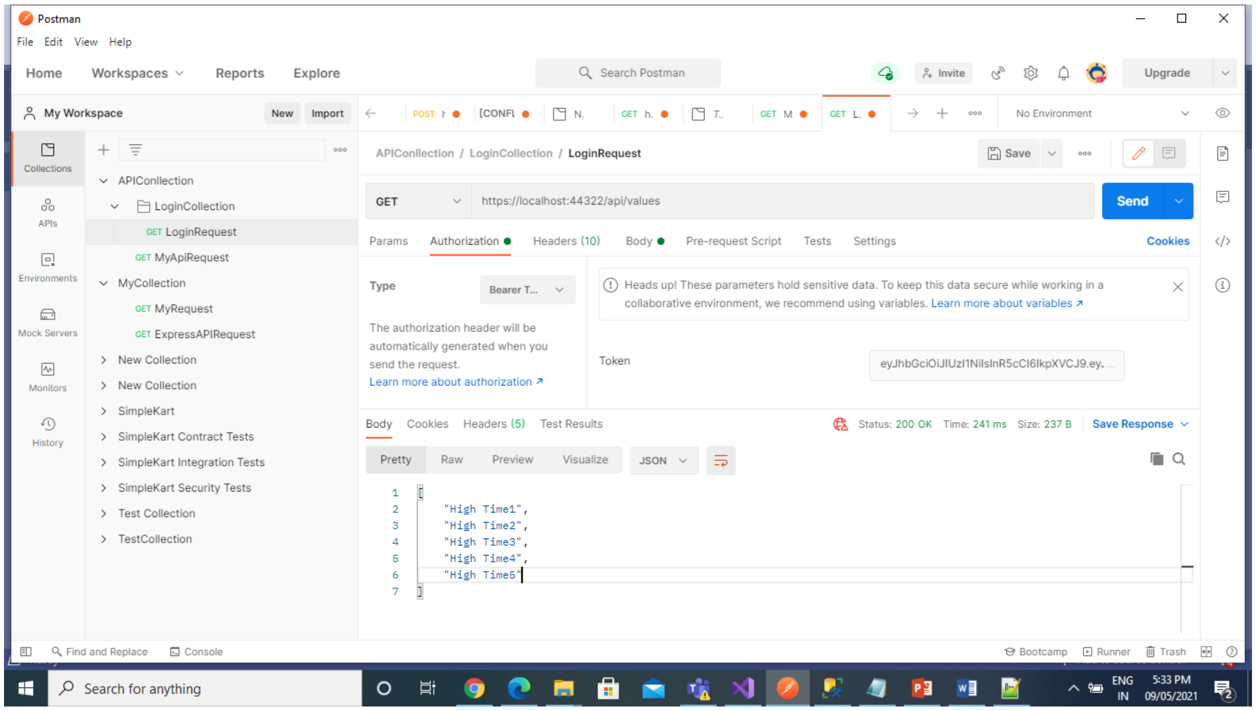
Authorization — Type: Bearer T...

The authorization header will be automatically generated when you send the request.

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables.

Token: Token

Status: 401 Unauthorized Time: 13 ms Size: 151 B



GET https://localhost:44322/api/values

Type: Bearer T...

The authorization header will be automatically generated when you send the request.
Learn more about authorization ↗

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. Learn more about variables ↗

Token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey...

Status: 200 OK Time: 241 ms Size: 237 B

```
[
    "High Time1",
    "High Time2",
    "High Time3",
    "High Time4",
    "High Time5"
]
```

ENG IN 5:33 PM 09/05/2021

Summary

JWT is very famous in web development. It is an open standard which allows transmitting data between parties as a JSON object in a secure and compact way.