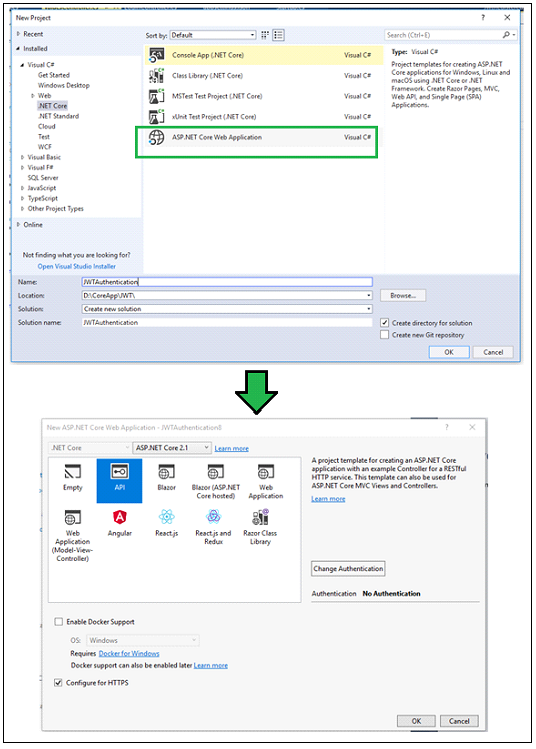
**Introduction**

[JWT (JSON web token)](https://www.c-sharpcorner.com/article/introduction-to-jwt/)has 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.

In this article, we will learn how to setup JWT with ASP.NET core web application. We can create an application using Visual Studio or using CLI (Command Line Interface).



1. 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 a JWT authentication schema by using "AddAuthentication" method and specifying JwtBearerDefaults.AuthenticationScheme. Here, we configure the authentication schema with JWT bearer options.

1. **public** **void** ConfigureServices(IServiceCollection services)
2. {
3. services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
4. .AddJwtBearer(options =>
5. {
6. options.TokenValidationParameters = **new** TokenValidationParameters
7. {
8. ValidateIssuer = **true**,
9. ValidateAudience = **true**,
10. ValidateLifetime = **true**,
11. ValidateIssuerSigningKey = **true**,
12. ValidIssuer = Configuration["Jwt:Issuer"],
13. ValidAudience = Configuration["Jwt:Issuer"],
14. IssuerSigningKey = **new** SymmetricSecurityKey(Encoding.UTF8.GetBytes(Configuration["Jwt:Key"]))
15. };
16. });
17. services.AddMvc();
18. }

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

* Validate the server (ValidateIssuer = true) that generates the token.
* Validate the recipient of token is authorized to receive (ValidateAudience = true)
* Check if token is not expired 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, I have stored these values in appsettings.json file.

**AppSetting.Json**

1. {
2. "Jwt": {
3. "Key": "ThisismySecretKey",
4. "Issuer": "Test.com"
5. }
6. }

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

1. **public** **void** Configure(IApplicationBuilder app, IHostingEnvironment env)
2. {
3. app.UseAuthentication();
4. app.UseMvc();
5. }

**Generate JSON Web Token**

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

I have created "AuthenticateUser" method, that is responsible to validate the user credential and returns to the UserModel. For demo purposes, I have returned hardcode model if the username is "Jignesh". If "AuthenticateUser" method returns user model, API generates the new token by using "GenerateJSONWebToken" method.

Here, I have created a JWT using the JwtSecurityToken class. I have created an 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.

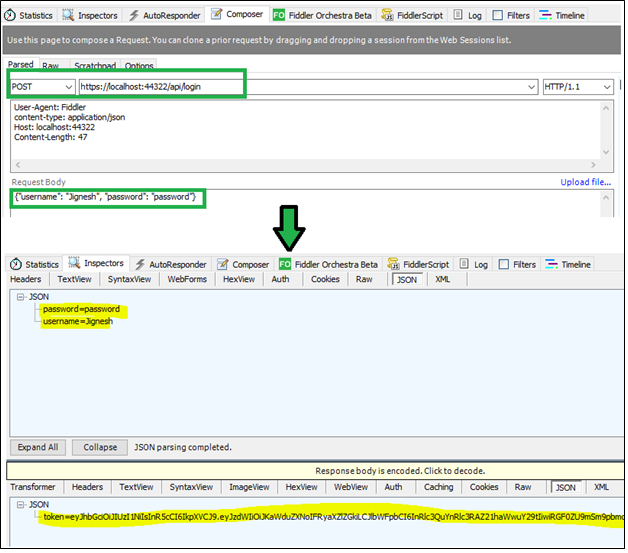
1. **using** Microsoft.AspNetCore.Authorization;
2. **using** Microsoft.AspNetCore.Mvc;
3. **using** Microsoft.Extensions.Configuration;
4. **using** Microsoft.IdentityModel.Tokens;
5. **using** System;
6. **using** System.IdentityModel.Tokens.Jwt;
7. **using** System.Security.Claims;
8. **using** System.Text;
10. **namespace** JWTAuthentication.Controllers
11. {
12. [Route("api/[controller]")]
13. [ApiController]
14. **public** **class** LoginController : Controller
15. {
16. **private** IConfiguration \_config;
18. **public** LoginController(IConfiguration config)
19. {
20. \_config = config;
21. }
22. [AllowAnonymous]
23. [HttpPost]
24. **public** IActionResult Login([FromBody]UserModel login)
25. {
26. IActionResult response = Unauthorized();
27. var user = AuthenticateUser(login);
29. **if** (user != **null**)
30. {
31. var tokenString = GenerateJSONWebToken(user);
32. response = Ok(**new** { token = tokenString });
33. }
35. **return** response;
36. }
38. **private** **string** GenerateJSONWebToken(UserModel userInfo)
39. {
40. var securityKey = **new** SymmetricSecurityKey(Encoding.UTF8.GetBytes(\_config["Jwt:Key"]));
41. var credentials = **new** SigningCredentials(securityKey, SecurityAlgorithms.HmacSha256);
43. var token = **new** JwtSecurityToken(\_config["Jwt:Issuer"],
44. \_config["Jwt:Issuer"],
45. **null**,
46. expires: DateTime.Now.AddMinutes(120),
47. signingCredentials: credentials);
49. **return** **new** JwtSecurityTokenHandler().WriteToken(token);
50. }
52. **private** UserModel AuthenticateUser(UserModel login)
53. {
54. UserModel user = **null**;
56. //Validate the User Credentials
57. //Demo Purpose, I have Passed HardCoded User Information
58. **if** (login.Username == "Jignesh")
59. {
60. user = **new** UserModel { Username = "Jignesh Trivedi", EmailAddress = "test.btest@gmail.com" };
61. }
62. **return** user;
63. }
64. }
65. }

Once, we have enabled the JWT based authentication, I have created a simple Web API method that returns a list of value strings when invoked with an HTTP GET request. Here, I have marked this method with the 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, I am Using [Fiddler](https://www.telerik.com/fiddler). First, I have requested to "API/login" method to generate the token. I have passed the following JSON in the request body.

1. {"username": "Jignesh", "password": "password"}

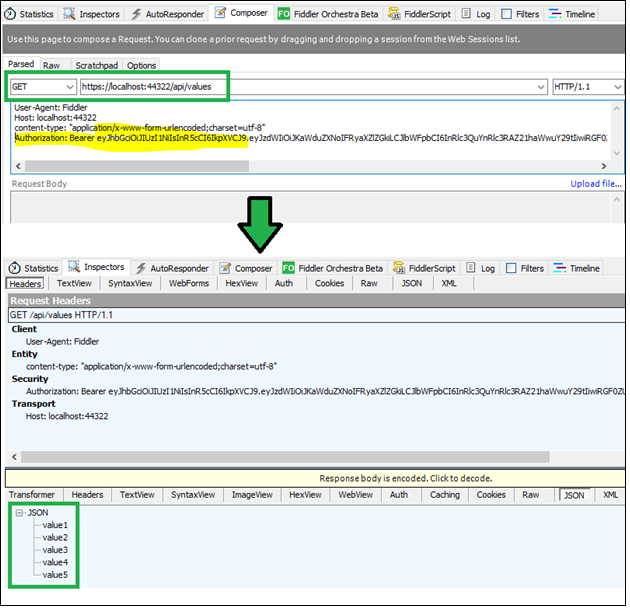


As a response, we will get the JSON like the following,

1. {
2. "token" : "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJKaWduZXNoIFRyaXZlZGkiLCJlbWFpbCI6InRlc3QuYnRlc3RAZ21haWwuY29tIiwiRGF0ZU9mSm9pbmciOiIwMDAxLTAxLTAxIiwianRpIjoiYzJkNTZjNzQtZTc3Yy00ZmUxLTgyYzAtMzlhYjhmNzFmYzUzIiwiZXhwIjoxNTMyMzU2NjY5LCJpc3MiOiJUZXN0LmNvbSIsImF1ZCI6IlRlc3QuY29tIn0.8hwQ3H9V8mdNYrFZSjbCpWSyR1CNyDYHcGf6GqqCGnY"
3. }

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

1. [HttpGet]
2. [Authorize]
3. **public** ActionResult<IEnumerable<**string**>> Get()
4. {
5. **return** **new** **string**[] { "value1", "value2", "value3", "value4", "value5" };
6. }
7. Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJKaWduZXNoIFRyaXZlZGkiLCJlbWFpbCI6InRlc3QuYnRlc3RAZ21haWwuY29tIiwiRGF0ZU9mSm9pbmciOiIwMDAxLTAxLTAxIiwianRpIjoiYzJkNTZjNzQtZTc3Yy00ZmUxLTgyYzAtMzlhYjhmNzFmYzUzIiwiZXhwIjoxNTMyMzU2NjY5LCJpc3MiOiJUZXN0LmNvbSIsImF1ZCI6IlRlc3QuY29tIn0.8hwQ3H9V8mdNYrFZSjbCpWSyR1CNyDYHcGf6GqqCGnY



**Handle Claims with JWT**

Claims are data contained by the token. They are information about the user which helps us to authorize the access to a resource. They could be Username, 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 the above example, if we want to pass the claims to our token then the claim information needs to add GenerateJSONWebToken method of Login controller. In the following example, I have added a username, email address, and date of joining as claimed into the token.

1. **private** **string** GenerateJSONWebToken(UserModel userInfo)
2. {
3. var securityKey = **new** SymmetricSecurityKey(Encoding.UTF8.GetBytes(\_config["Jwt:Key"]));
4. var credentials = **new** SigningCredentials(securityKey, SecurityAlgorithms.HmacSha256);
6. var claims = **new**[] {
7. **new** Claim(JwtRegisteredClaimNames.Sub, userInfo.Username),
8. **new** Claim(JwtRegisteredClaimNames.Email, userInfo.EmailAddress),
9. **new** Claim("DateOfJoing", userInfo.DateOfJoing.ToString("yyyy-MM-dd")),
10. **new** Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString())
11. };
13. var token = **new** JwtSecurityToken(\_config["Jwt:Issuer"],
14. \_config["Jwt:Issuer"],
15. claims,
16. expires: DateTime.Now.AddMinutes(120),
17. signingCredentials: credentials);
19. **return** **new** JwtSecurityTokenHandler().WriteToken(token);
20. }

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 to filter the data. In the following example, I have to change the list of values if the user spends more than 5 years with the company.

1. [HttpGet]
2. [Authorize]
3. **public** ActionResult<IEnumerable<**string**>> Get()
4. {
5. var currentUser = HttpContext.User;
6. **int** spendingTimeWithCompany = 0;
8. **if** (currentUser.HasClaim(c => c.Type == "DateOfJoing"))
9. {
10. DateTime date = DateTime.Parse(currentUser.Claims.FirstOrDefault(c => c.Type == "DateOfJoing").Value);
11. spendingTimeWithCompany = DateTime.Today.Year - date.Year;
12. }
14. **if**(spendingTimeWithCompany > 5)
15. {
16. **return** **new** **string**[] { "High Time1", "High Time2", "High Time3", "High Time4", "High Time5" };
17. }
18. **else**
19. {
20. **return** **new** **string**[] { "value1", "value2", "value3", "value4", "value5" };
21. }
22. }