ASP.NET Core Web API 5.0 Authentication Using JWT(JSON BASE TOKEN)

## **Introduction**

Authentication plays a very important role to keep an organization's data/network etc secure. There are many authentication approaches available in the market, but the most popular approach is “Token Based Authentication”.

In this article, we are going to discuss and implement Token Based Authentication with Asp.net Core Web API 5.0 + JWT (JSON Web Token).

We will create a simple Web API to understand JWT. We can use postman or swagger to test our Web API once it's done.

I recommend to read this [article](https://www.c-sharpcorner.com/article/introduction-of-json-web-token-jwt/) first if you are not aware of JWT and then come back for implementation.

## **What is Token-based authentication?**

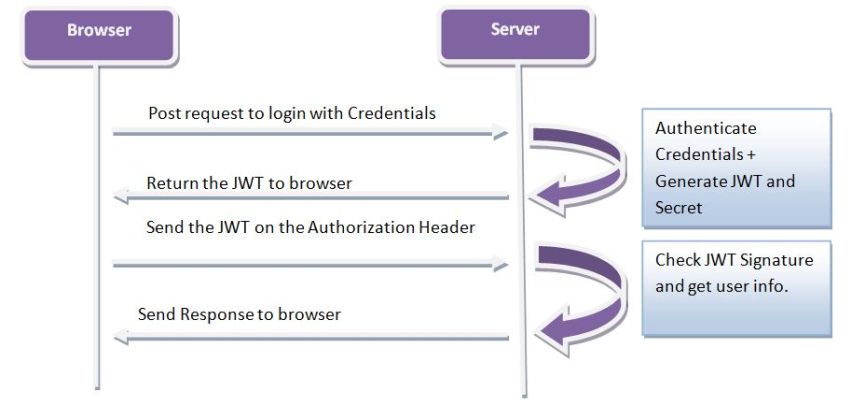
Before we start, we should know what Authentication is.

In simple words we can say, Authentication is validating the user with credentials or identity.

Now next question comes in the mind is “What is Token based authentication in Web API”?

Token Base Authentication processes,

1. The client sent a request to the server with credentials.
2. The server validates the credential and creates an Access token and sends it back to the client.
3. All subsequence requests content this token until its expired.



There are many Open standards available in the market to implement Token Based Authentication in Web API and JSON WEB TOKEN(JWT) is the most popular among them.

## **Steps to follow to create JWT Authentication in Web API**

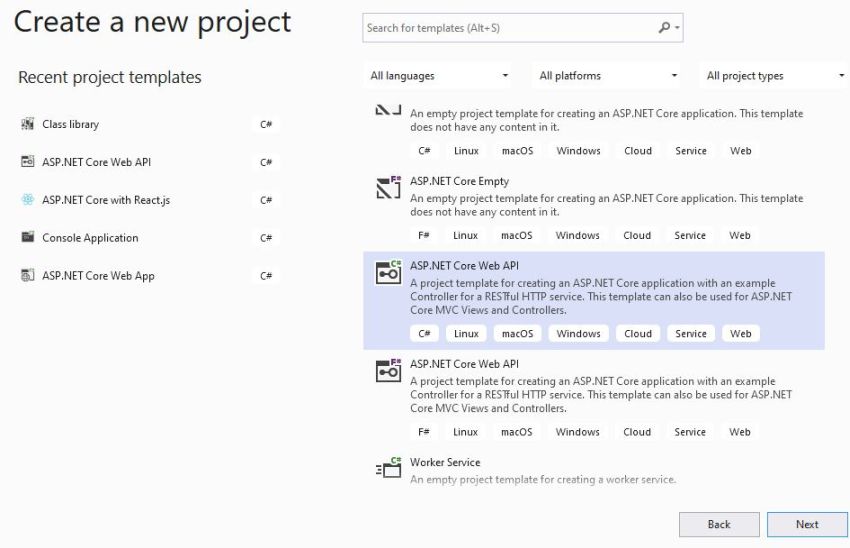
Web API Project has the below endpoints,

1. /authenticate – Authenticate Member Credential and send Access token for subsequence request.
2. /All members – Return List of Members.
3. / MemberByid /id - Members filter by id and return a single member.

Let us create the project using Visual Studio 2019.

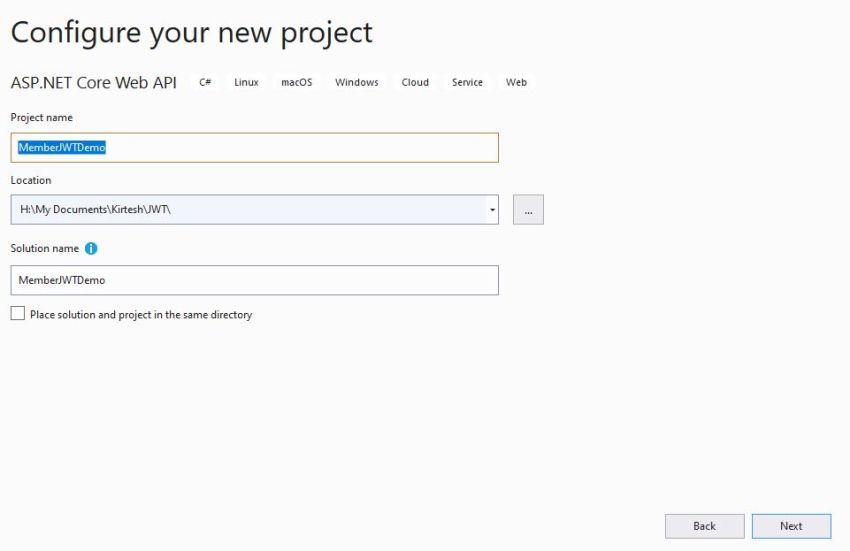
**Step 1**

Create a new project.



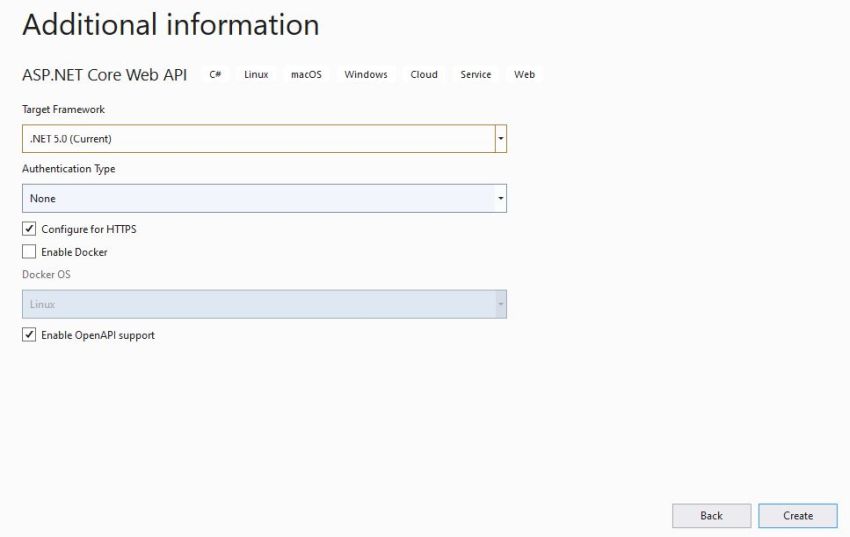
**Step 2**

Select the “Asp.Net Core Web API” template and click on the Next button.



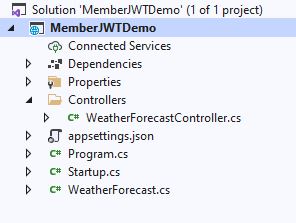
**Step 3**

Configure Project Name, location as per the above screen. Click on the Next button.



**Step 4**

Provide Target Framework (.Net 5.0) and click on create button to create a Web API project.

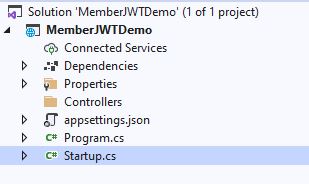
****

“MemberJWTDemo” Project has been created successfully, and default solution should look likethe above image.

The initial Project setup is ready, now follow the below steps to implement JWT Authentication.

**Step 5**

Remove default controller “WeatherForecastController.cs” and “WeatherForcast.cs” files from the project.

****

**Step 6**

We need to Enable Authentication in middleware to validate members. To do so, open the Startup.cs file and add the below code,

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Hosting;

using Microsoft.AspNetCore.HttpsPolicy;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Hosting;

using Microsoft.Extensions.Logging;

using Microsoft.OpenApi.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace MemberJWTDemo

{

public class Startup

{

public Startup(IConfiguration configuration)

{

Configuration = configuration;

}

public IConfiguration Configuration { get; }

// This method gets called by the runtime. Use this method to add services to the container.

public void ConfigureServices(IServiceCollection services)

{

services.AddControllers();

services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new OpenApiInfo { Title = "MemberJWTDemo", Version = "v1" });

});

}

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

app.UseSwagger();

app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json", "MemberJWTDemo v1"));

}

app.UseHttpsRedirection();

app.UseRouting();

app.UseAuthentication();

app.UseAuthorization();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllers();

});

}

}

}

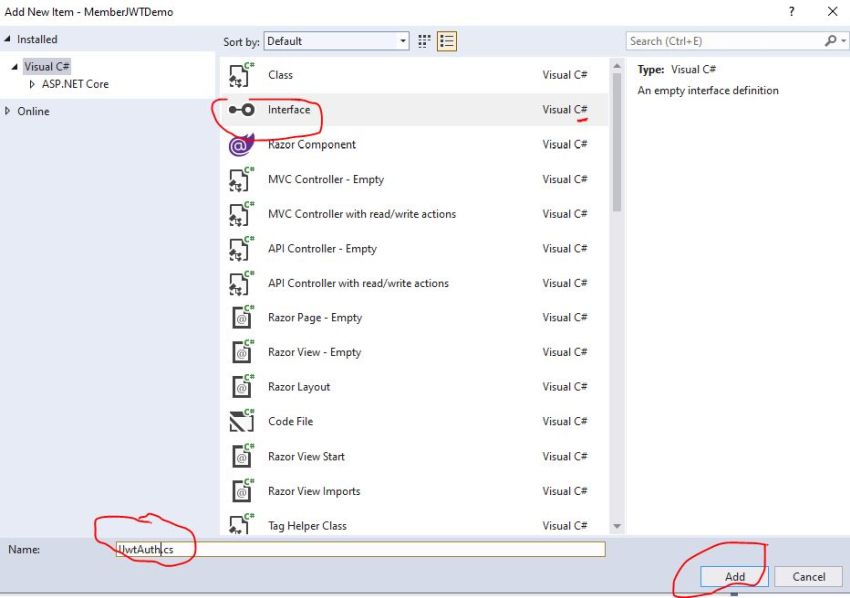
C#

Copy

To validate member’s credentials and generate JWT tokens, we need a custom authentication class.

**Step 7**

First, we will create an interface called the”IJwtAuth.cs” file,



namespace MemberJWTDemo

{

public interface IJwtAuth

{

string Authentication(string username, string password);

}

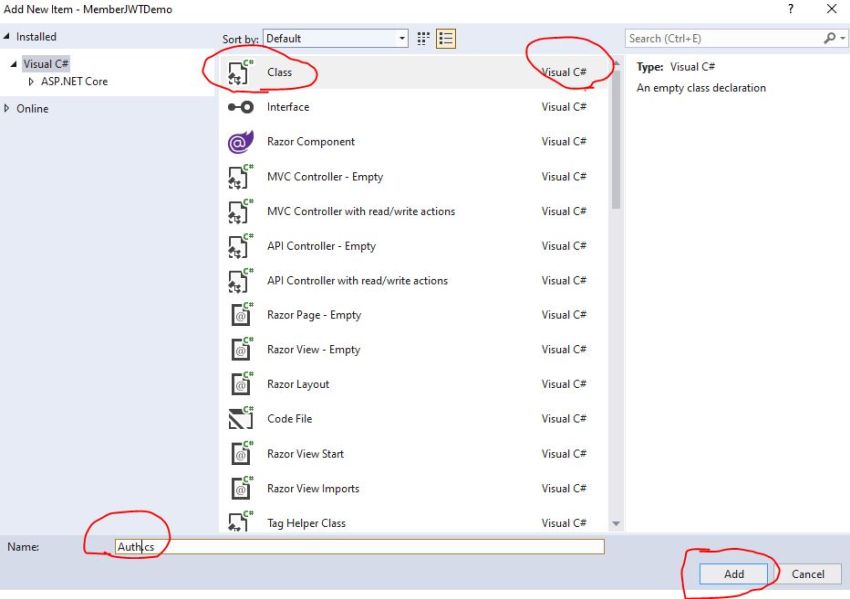
}

C#

Copy

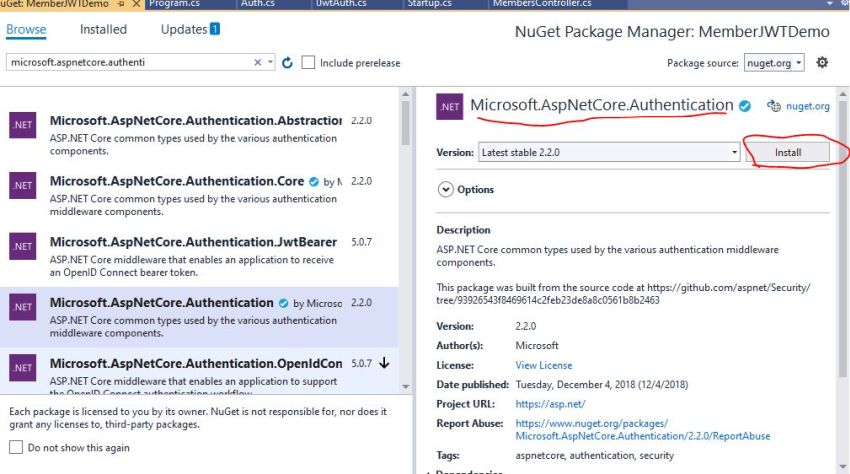
**Step 8**

Now add a new class called “Auth” and Implement the “IAuth” interface.



**Step 9**

We need to add Nuget package “Microsoft.AspNetCore.Authentication” for Authentication.

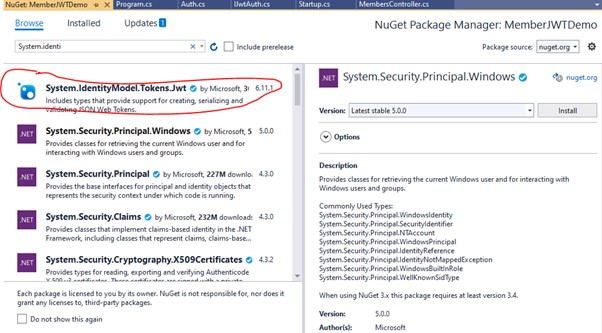


We will keep the username and password hardcoded for demo purposes. In the real-world scenario, it will come from a database.

If the user is authenticated successfully then we will create a JWT token.

**Step 10**

To create JWT Token we need to install Nuget package “System.IdentityModel.Tokens.Jwt”.

****

**Step 11**

Now we will write the below code in Auth class - Authentication method to create token after authenticating.

using Microsoft.IdentityModel.Tokens;

using System;

using System.IdentityModel.Tokens.Jwt;

using System.Text;

using System.Security.Claims;

namespace MemberJWTDemo

{

public class Auth : IJwtAuth

{

private readonly string username = "kirtesh";

private readonly string password = "Demo1";

private readonly string key;

public Auth(string key)

{

this.key = key;

}

public string Authentication(string username, string password)

{

if (!(username.Equals(username) || password.Equals(password)))

{

return null;

}

// 1. Create Security Token Handler

var tokenHandler = new JwtSecurityTokenHandler();

// 2. Create Private Key to Encrypted

var tokenKey = Encoding.ASCII.GetBytes(key);

//3. Create JETdescriptor

var tokenDescriptor = new SecurityTokenDescriptor()

{

Subject = new ClaimsIdentity(

new Claim[]

{

new Claim(ClaimTypes.Name, username)

}),

Expires = DateTime.UtcNow.AddHours(1),

SigningCredentials = new SigningCredentials(

new SymmetricSecurityKey(tokenKey), SecurityAlgorithms.HmacSha256Signature)

};

//4. Create Token

var token = tokenHandler.CreateToken(tokenDescriptor);

// 5. Return Token from method

return tokenHandler.WriteToken(token);

}

}

}

C#

Copy

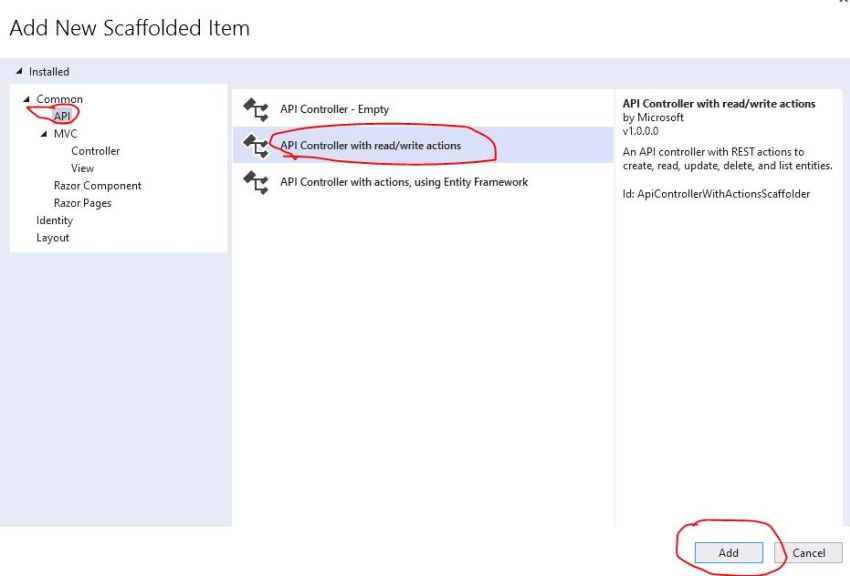
In the above code,

* Create Security Handler – “token handler”.
* Once Token Handler is created, we will encrypt the key into bytes.
* Now we will create a token descriptor.
  + **Subject –** New Claim identity
  + **Expired –**When it will be expired.
  + **SigningCredentical –** Private key + Algorithm
* Now we will create a token using the “create token” method of the token handler.
* Return token from Authentication method.

Now we will create a controller and use this authentication method.

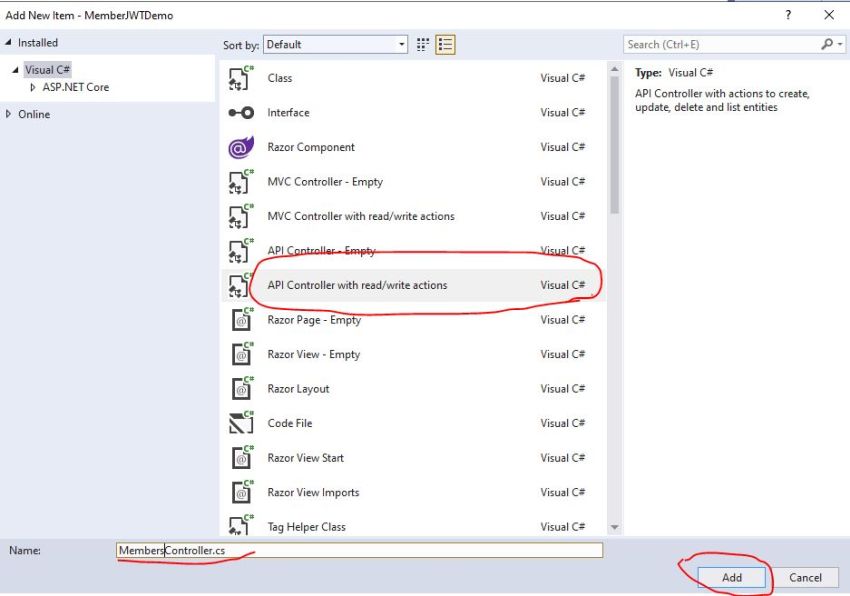
**Step 12**

Create New Controller “Members” with HTTP POST endpoint called Authentication.



**Step 13**

Click on Add and Provide controller name in the below screen,



using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using System.Collections.Generic;

// For more information on enabling Web API for empty projects, visit <https://go.microsoft.com/fwlink/?LinkID=397860>

namespace MemberJWTDemo.Controllers

{

[Authorize]

[Route("api/[controller]")]

[ApiController]

public class MembersController : ControllerBase

{

private readonly IJwtAuth jwtAuth;

private readonly List<Member> lstMember = new List<Member>()

{

new Member{Id=1, Name="Kirtesh" },

new Member {Id=2, Name="Nitya" },

new Member{Id=3, Name="pankaj"}

};

public MembersController(IJwtAuth jwtAuth)

{

this.jwtAuth = jwtAuth;

}

// GET: api/<MembersController>

[HttpGet]

public IEnumerable<Member> AllMembers()

{

return lstMember;

}

// GET api/<MembersController>/5

[HttpGet("{id}")]

public Member MemberByid(int id)

{

return lstMember.Find(x => x.Id == id);

}

[AllowAnonymous]

// POST api/<MembersController>

[HttpPost("authentication")]

public IActionResult Authentication([FromBody]UserCredential userCredential)

{

var token = jwtAuth.Authentication(userCredential.UserName, userCredential.Password);

if (token == null)

return Unauthorized();

return Ok(token);

}

}

}

C#

Copy

In the above code,

1. The authentication method took the user name and password from the body.
2. Pass credential to the jwtAuth. Authentication method to get token.
3. Return token.
4. Add attributes [AllowAnonymous] as this method can be handled by any user.
5. Add [Authorize] attributes to Member controller.
6. Add “jwtAuth” in the constructor.

**Step 14**

Create UserCredential Class as below,

namespace MemberJWTDemo.Controllers

{

public class UserCredential

{

public string UserName { get; set; }

public string Password { get; set; }

}

}

C#

Copy

**Step 15**

Create Member model class as below,

namespace MemberJWTDemo

{

public class Member

{

public int Id { get; set; }

public string Name { get; set; }

}

}

C#

Copy

We need to add dependency in the Startup.cs file. Also, we will add JETBearar to decrypt the key.

**Step 16**

Install “Microsoft.AspNetCore.Authentication.JwtBearer” NuGet package.

**Step 17**

Add the below code in the startup.cs file,

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Hosting;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Hosting;

using Microsoft.IdentityModel.Tokens;

using Microsoft.OpenApi.Models;

using System.Text;

namespace MemberJWTDemo

{

public class Startup

{

public Startup(IConfiguration configuration)

{

Configuration = configuration;

}

public IConfiguration Configuration { get; }

// This method gets called by the runtime. Use this method to add services to the container.

public void ConfigureServices(IServiceCollection services)

{

services.AddControllers();

var key = "This is my first Test Key";

services.AddAuthentication(x =>

{

x.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;

x.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;

}).AddJwtBearer(x =>

{

x.RequireHttpsMetadata = false;

x.SaveToken = true;

x.TokenValidationParameters = new Microsoft.IdentityModel.Tokens.TokenValidationParameters

{

ValidateIssuerSigningKey = true,

ValidateIssuer = false,

ValidateAudience = false,

IssuerSigningKey = new SymmetricSecurityKey(Encoding.ASCII.GetBytes(key))

};

});

services.AddSingleton<IJwtAuth>(new Auth(key));

services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new OpenApiInfo { Title = "MemberJWTDemo", Version = "v1" });

})

}

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

app.UseSwagger();

app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json", "MemberJWTDemo v1"));

}

app.UseHttpsRedirection();

app.UseRouting();

app.UseAuthentication();

app.UseAuthorization();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllers();

});

}

}

}

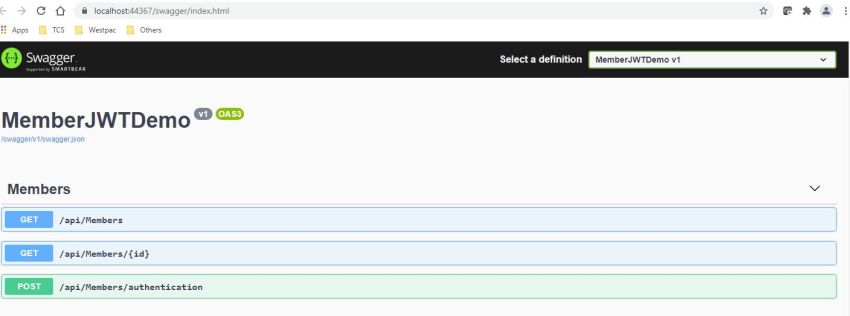
C#

Copy

We are done with code.

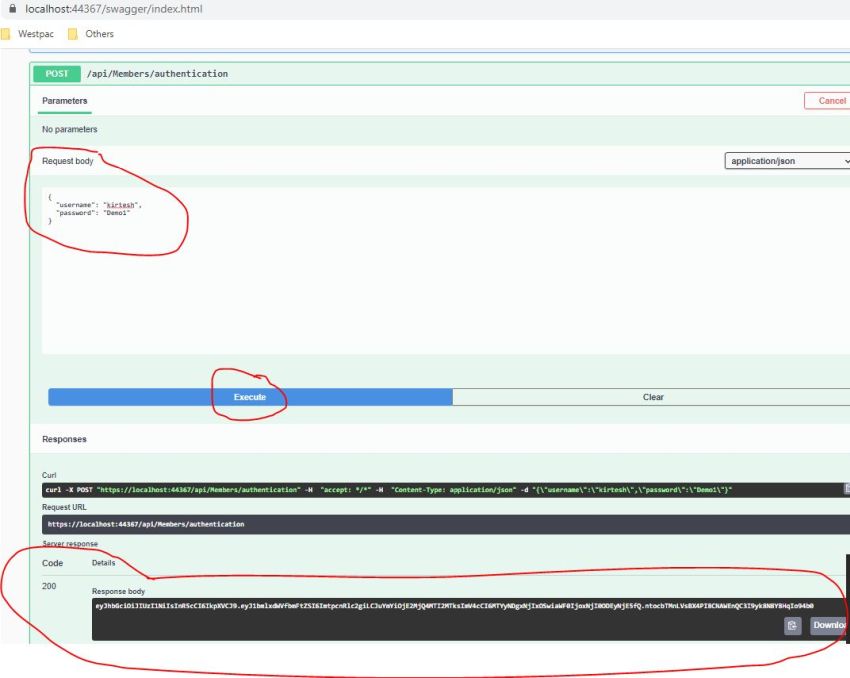
**Step 18**

Now we will call API endpoint using Postman or swagger etc.



**Step 19**

First, we will call post method /API/Members/authentication,



The token is created successfully. Now in the subsequence request, we have to pass this token in header(**Key -** authorized, and **value** -token)