**JWT Token Creation, Authentication And Authorization In ASP.NET Core 6.0 With Postman**

[JWTLoginAuthenticationAuthorization.zip](javascript:void(0);)

Introduction

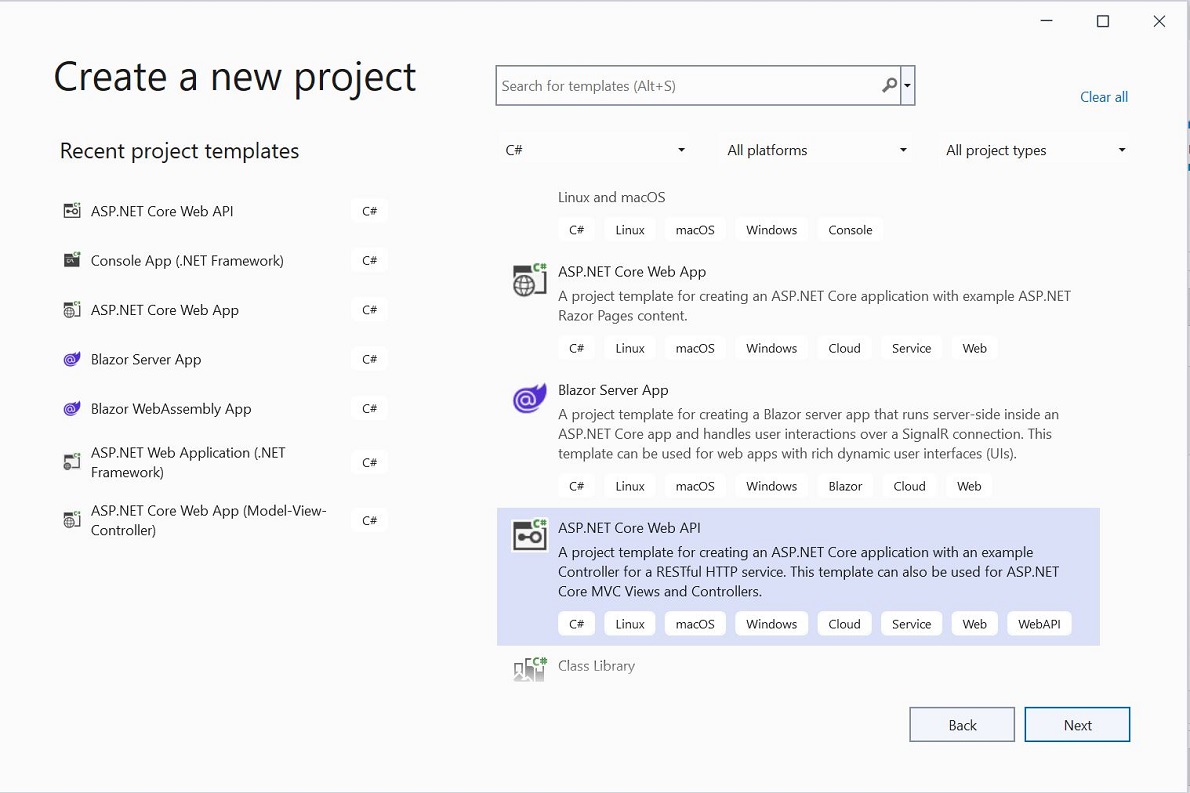
In this article, I will explain how to create the JWT token and how to Authenticate and Authorize it in very simple steps.

We will follow the below steps to JWT token creation, authentication and authorization.

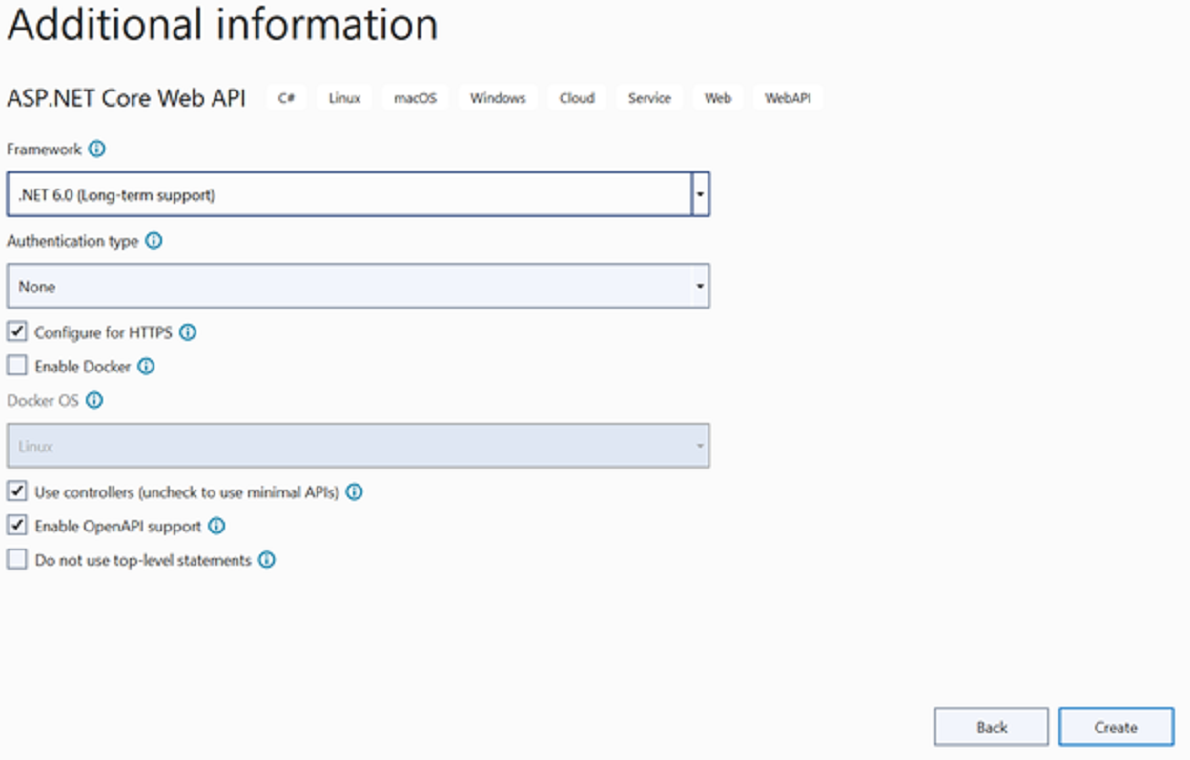
* ASP.Net Core API Application
* Add required packages.
* Add Key, Issuer and Audience in appsettings.cs
* Register JWT Token for Authentication in Startup.cs file.
* Create Models (UserLogin, UserModel and UserConstant)
* Create Login API Controller (Authenticate user and generate token)
* Create User API Controller to authorize user role.
* Test the API endpoint in Postman with Token.

1. Add ASP.Net Core API Application

Open visual studio 2022 click on create new project --> Select ASP.Net Core Web API --> Next



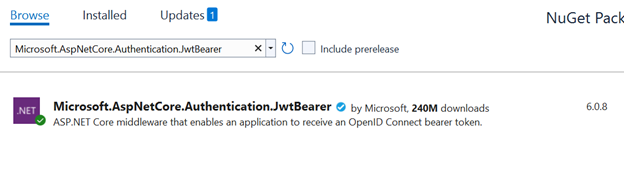
Give desired project and solution name --> Next --> select framework .Net 6.0 --> Create

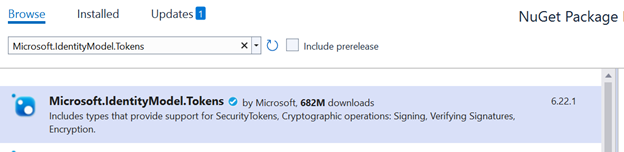


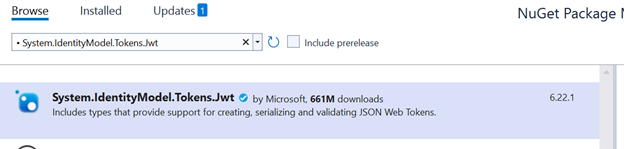
2. Add Nuget Packages

Add the following packages from nuget package manager.

* Microsoft.AspNetCore.Authentication.JwtBearer
* Microsoft.IdentityModel.Tokens
* System.IdentityModel.Tokens.Jwt







3. Add setting in appsetting.json

Open appsetting.json and add following Key, Issuer and Audience

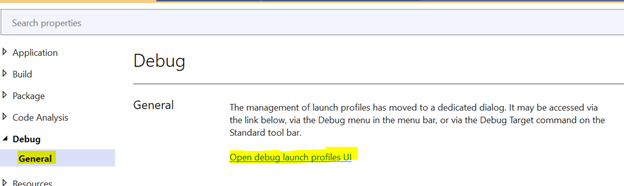


\* To generate the random key use

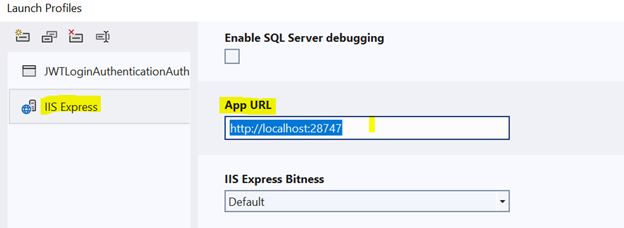
https://www.random.org/strings

\* For issuer and audience local URL follow the below steps

Project properties --> Debug --> General --> Open Debug Launch Profile UI



Select IIS Express and pick the App URL



"Jwt": {

"Key": "ACDt1vR3lXToPQ1g3MyN", //Generate random String from <https://www.random.org/strings>

"Issuer": "<http://localhost:28747/>", //Project Property-> Debug-> IIS-->App URL (you can local host url as well)

"Audience": "<http://localhost:28747/>"

},

C#

Copy

4. Register JWT token for Authentication in Program.cs file

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.IdentityModel.Tokens;

using System.Text;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();

// Learn more about configuring Swagger/OpenAPI at <https://aka.ms/aspnetcore/swashbuckle>

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

//JWT Authentication

builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme).AddJwtBearer(options => {

options.TokenValidationParameters = new TokenValidationParameters {

ValidateIssuer = true,

ValidateAudience = true,

ValidateLifetime = true,

ValidateIssuerSigningKey = true,

ValidIssuer = builder.Configuration["Jwt:Issuer"],

ValidAudience = builder.Configuration["Jwt:Audience"],

IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(builder.Configuration["Jwt:Key"]))

};

});

var app = builder.Build();

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment()) {

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection();

app.UseAuthentication();

app.UseAuthorization();

app.MapControllers();

app.Run();

C#

Copy

5. Create Models (UserLogin, UserModel and UserConstant)

Add a new folder with Models name and create UserLogin, UserModel and UserConstant classes.

namespace JWTLoginAuthenticationAuthorization.Models

{

public class UserModel

{

public string Username { get; set; }

public string Password { get; set; }

public string Role { get; set; }

}

}

C#

Copy

namespace JWTLoginAuthenticationAuthorization.Models

{

public class UserLogin

{

public string Username { get; set; }

public string Password { get; set; }

}

}

C#

Copy

namespace JWTLoginAuthenticationAuthorization.Models

{

// We are not taking data from data base so we get data from constant

public class UserConstants

{

public static List<UserModel> Users = new()

{

new UserModel(){ Username="naeem",Password="naeem\_admin",Role="Admin"}

};

}

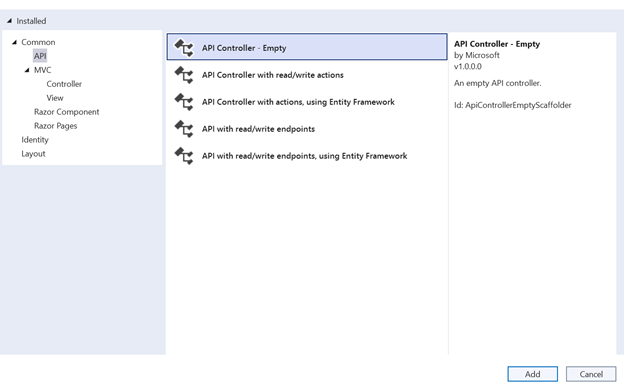
}

C#

Copy

6. Create LoginAPI Controller (Authenticate user and generate token)

Add a new Empty API controller name “LoginController” in controller folder.



Here creates one Post Action method for Login and two methods for Authenticating the user credentials and Generate the token (if user is authenticated).

using JWTLoginAuthenticationAuthorization.Models;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.IdentityModel.Tokens;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

namespace JWTLoginAuthenticationAuthorization.Controllers

{

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

[ApiController]

public class LoginController : ControllerBase

{

private readonly IConfiguration \_config;

public LoginController(IConfiguration config)

{

\_config = config;

}

[AllowAnonymous]

[HttpPost]

public ActionResult Login([FromBody] UserLogin userLogin)

{

var user = Authenticate(userLogin);

if (user != null)

{

var token = GenerateToken(user);

return Ok(token);

}

return NotFound("user not found");

}

// To generate token

private string GenerateToken(UserModel user)

{

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

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

var claims = new[]

{

new Claim(ClaimTypes.NameIdentifier,user.Username),

new Claim(ClaimTypes.Role,user.Role)

};

var token = new JwtSecurityToken(\_config["Jwt:Issuer"],

\_config["Jwt:Audience"],

claims,

expires: DateTime.Now.AddMinutes(15),

signingCredentials: credentials);

return new JwtSecurityTokenHandler().WriteToken(token);

}

//To authenticate user

private UserModel Authenticate(UserLogin userLogin)

{

var currentUser = UserConstants.Users.FirstOrDefault(x => x.Username.ToLower() ==

userLogin.Username.ToLower() && x.Password == userLogin.Password);

if (currentUser != null)

{

return currentUser;

}

return null;

}

}

}

C#

Copy

7. Create User API Controller to authorize user role

Add new empty API controller named “UserController.cs” in controller folder.

Here we will authorize the endpoint on the behalf of role.

namespace JWTLoginAuthenticationAuthorization.Controllers

{

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

[ApiController]

public class UserController : ControllerBase

{

//For admin Only

[HttpGet]

[Route("Admins")]

[Authorize(Roles = "Admin")]

public IActionResult AdminEndPoint()

{

var currentUser = GetCurrentUser();

return Ok($"Hi you are an {currentUser.Role}");

}

private UserModel GetCurrentUser()

{

var identity = HttpContext.User.Identity as ClaimsIdentity;

if (identity != null)

{

var userClaims = identity.Claims;

return new UserModel

{

Username = userClaims.FirstOrDefault(x => x.Type == ClaimTypes.NameIdentifier)?.Value,

Role = userClaims.FirstOrDefault(x => x.Type == ClaimTypes.Role)?.Value

};

}

return null;

}

}

}

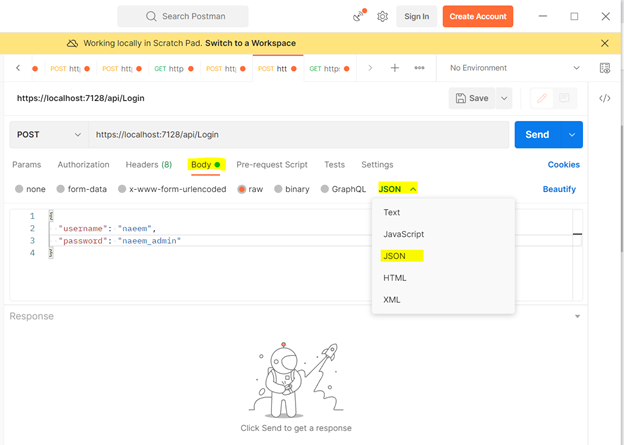
C#

Copy

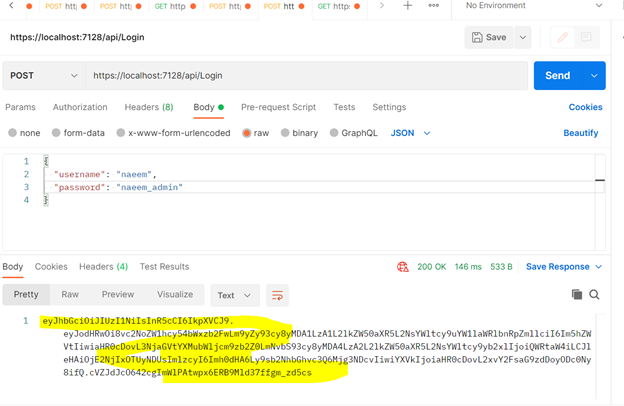
8. Test the API endpoint in Postman with Token

Run the application and copy the URL domain from the browser.

Now open the Postman, give the URL with correct API route and select post request --> Body --> Json --> give the value of Username and Password

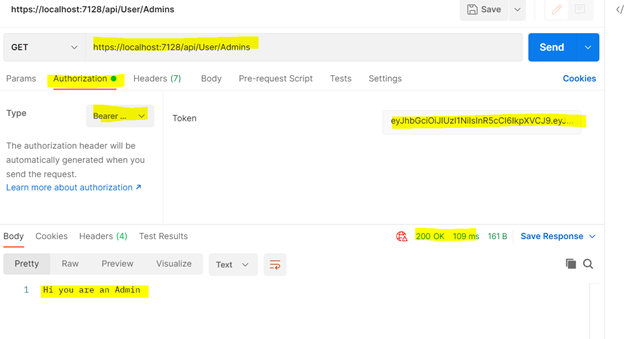


After clicking on send button we will get the JWT token in response.



Now copy this token and add a new Get request in postman and add the JWT token Authorization Tab --> Select Bearer --> Insert token and click on send button to test the authorization with given token.

If the token is not valid token then we will get 401 Error otherwise will get the bolow result.



Summary

So we created the token and did the authentication on the behalf of username and password then check the user authorization.