Adding authentication in the WebApi

Working in a minimal webapi.

To make sense, I will run three different tasks on this document.

1. Create a simple, minimal WebAPI
2. Organising the code (consult the document **Organising Your Code**)
3. Adding Swagger (consult **Adding Swagger to your code)**
4. Then, I will add the authentication and authorisation in the GET method

Create your project.

dotnet new webapi -n jwtGen

Add these NuGet packages.

cd jwtGen

dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer –version 9.0.5

dotnet add package Swashbuckle.AspNetCore --version 8.1.1

dotnet add package System.IdentityModel.Tokens.Jwt --version 8.9.0

Change your Program.cs.

Add these lines at top of the file:

using System.Text;

using Microsoft.IdentityModel.Tokens;

using Microsoft.AspNetCore.Authentication.JwtBearer;

using System.IdentityModel.Tokens.Jwt;

using jwtGen.Services;

using jwtGen.EndPoints;

After this line: builder.Services.AddOpenApi(); add the below lines:

// Load configuration

IConfiguration configuration = builder.Configuration;

// Register JwtService with configuration

builder.Services.AddSingleton<JwtService>(sp => new JwtService(configuration));

// Config Jwt token

builder.Services.AddAuthentication("Bearer")

.AddJwtBearer("Bearer", options => {

options.TokenValidationParameters = new Microsoft.IdentityModel.Tokens.TokenValidationParameters

{

ValidateIssuer = true,

ValidateAudience = true,

ValidateLifetime = true,

ValidateIssuerSigningKey = true,

ValidIssuer = configuration["jwt:Issuer"],

ValidAudience = configuration["jwt:Audience"],

IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(configuration["Jwt:SecretKey"] ?? throw new ArgumentNullException("JWT Secret Key is missing in appsettings.json")))

};

});

builder.Services.AddAuthorization();

// Register Swagger

builder.Services.AddEndpointsApiExplorer();

//builder.Services.AddSwaggerGen();

builder.Services.AddSwaggerGen(options =>

{

options.AddSecurityDefinition("Bearer", new Microsoft.OpenApi.Models.OpenApiSecurityScheme

{

Name = "Authorization",

Type = Microsoft.OpenApi.Models.SecuritySchemeType.Http,

Scheme = "Bearer",

BearerFormat = "JWT",

In = Microsoft.OpenApi.Models.ParameterLocation.Header,

Description = "Enter JWT token like: Bearer {token}"

});

options.AddSecurityRequirement(new Microsoft.OpenApi.Models.OpenApiSecurityRequirement

{

{

new Microsoft.OpenApi.Models.OpenApiSecurityScheme

{

Reference = new Microsoft.OpenApi.Models.OpenApiReference

{

Type = Microsoft.OpenApi.Models.ReferenceType.SecurityScheme,

Id = "Bearer"

}

},

new string[] {}

}

});

});

var app = builder.Build(); **// Your code already has this line. Don’t need to add**

var jwtService = app.Services.GetRequiredService<JwtService>();

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

app.MapOpenApi(); **// Your code already has this line. Don’t need to add**

}

app.UseAuthentication();

app.UseAuthorization();

app.UseHttpsRedirection(); **// Your code already has this line. Don’t need to add**

app.MapWeatherEndPoint(jwtService);

app.Run(); **// Your code already has this line. Don’t need to add**

Create your EndPoint.

mkdir EndPoints

cd EndPoints

vi WeatherEndPoint.cs

using jwtGen.Services;

using System.Text;

using Microsoft.IdentityModel.Tokens;

using Microsoft.AspNetCore.Authentication.JwtBearer;

using System.IdentityModel.Tokens.Jwt;

using Microsoft.AspNetCore.Mvc;

namespace jwtGen.EndPoints;

public static class WeatherEndPoint

{

public static **WebApplication** MapWeatherEndPoint(**this** WebApplication app, JwtService jwtService )

{

app.MapPost("/weatherforecast", ([FromBody] LoginInfo loginInfo ) =>

{

if(loginInfo.LoginName == "Admin" && loginInfo.Password == "Admin123")

{

var token = jwtService.GenerateToken(loginInfo.LoginName);

return Results.Ok(new {Token = token });

}

return Results.Unauthorized();

});

app.MapGet("/weatherforecast", () =>

{

return WeatherService.GetWeatherforecast();

})

**.RequireAuthorization();** // This ensures authentication is required for thi;

return app;

}

}

public class LoginInfo

{

public string? LoginName {get; set; }

public string? Password {get; set; }

}

Create you Services

mkdir Services

cd Services

**vi JwtService.cs**

using System.Security.Claims;

using System.IdentityModel.Tokens.Jwt;

using Microsoft.IdentityModel.Tokens;

using System.Text;

using Microsoft.Extensions.Configuration;

namespace jwtGen.Services;

public class JwtService

{

private readonly string \_SECRETKEY;

public JwtService(IConfiguration configuration)

{

\_SECRETKEY = configuration["Jwt:SecretKey"] ?? throw new ArgumentNullException("JWT secret key is missing in appsettings.json");

}

public string GenerateToken(string username)

{

var claim = new[]

{

new Claim(JwtRegisteredClaimNames.Sub, username),

new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString())

};

var key= new SymmetricSecurityKey(Encoding.UTF8.GetBytes(\_SECRETKEY));

var cred = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

var token = new JwtSecurityToken(

issuer: "http://localhost:5032",

audience: "http://localhost:5032",

claims: claim,

expires: DateTime.Now.AddHours(1),

signingCredentials: cred);

return new JwtSecurityTokenHandler().WriteToken(token);

}

}

**vi WeatherService.cs**

namespace jwtGen.Services;

public class WeatherService

{

private static readonly string[] summaries = new[]

{

"Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot", "Sweltering", "Scorching"

};

public static IEnumerable<WeatherForecast> GetWeatherforecast()

{

return Enumerable.Range(1, 5).Select(index => new WeatherForecast

(

DateOnly.FromDateTime(DateTime.Now.AddDays(index)),

Random.Shared.Next(-20, 55),

summaries[Random.Shared.Next(summaries.Length)]

))

.ToArray();

}

public record WeatherForecast(DateOnly Date, int TemperatureC, string? Summary)

{

public int TemperatureF => 32 + (int)(TemperatureC / 0.5556);

}

}

Test your WebApi

Run the WebApi

vagrant@ubuntu-jammy:~/Projects/jwtGen$ dotnet run

Using launch settings from /home/vagrant/Projects/jwtGen/Properties/launchSettings.json...

Building...

info: Microsoft.Hosting.Lifetime[14]

Now listening on: http://**0.0.0.0**:5032

info: Microsoft.Hosting.Lifetime[0]

Application started. Press Ctrl+C to shut down.

info: Microsoft.Hosting.Lifetime[0]

Hosting environment: Development

info: Microsoft.Hosting.Lifetime[0]

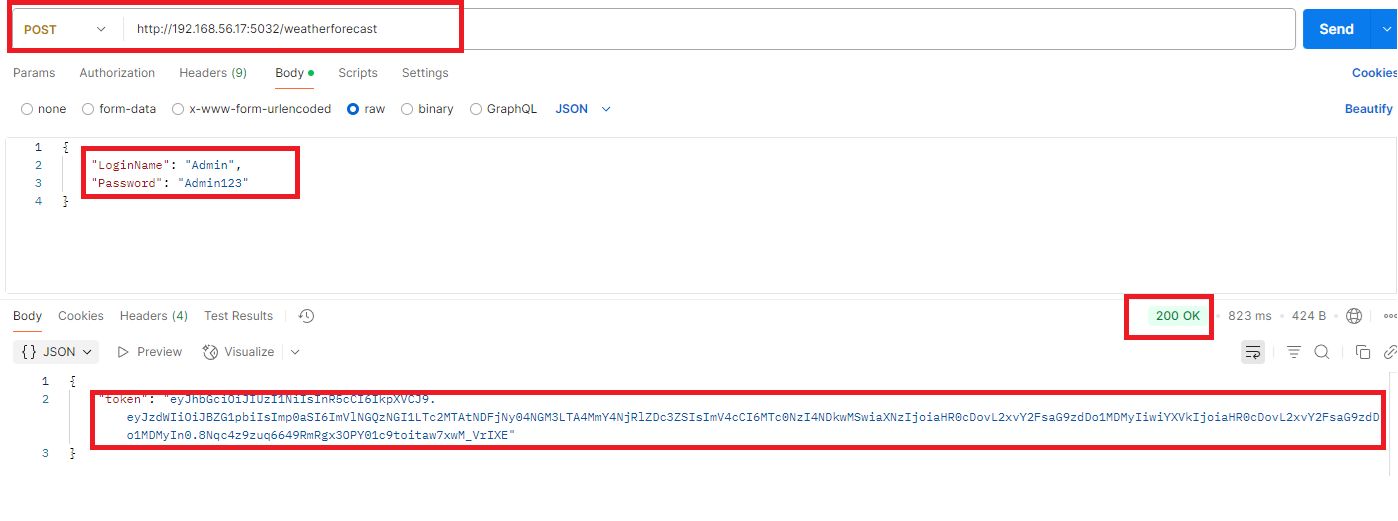
Content root path: /home/vagrant/Projects/jwtGen

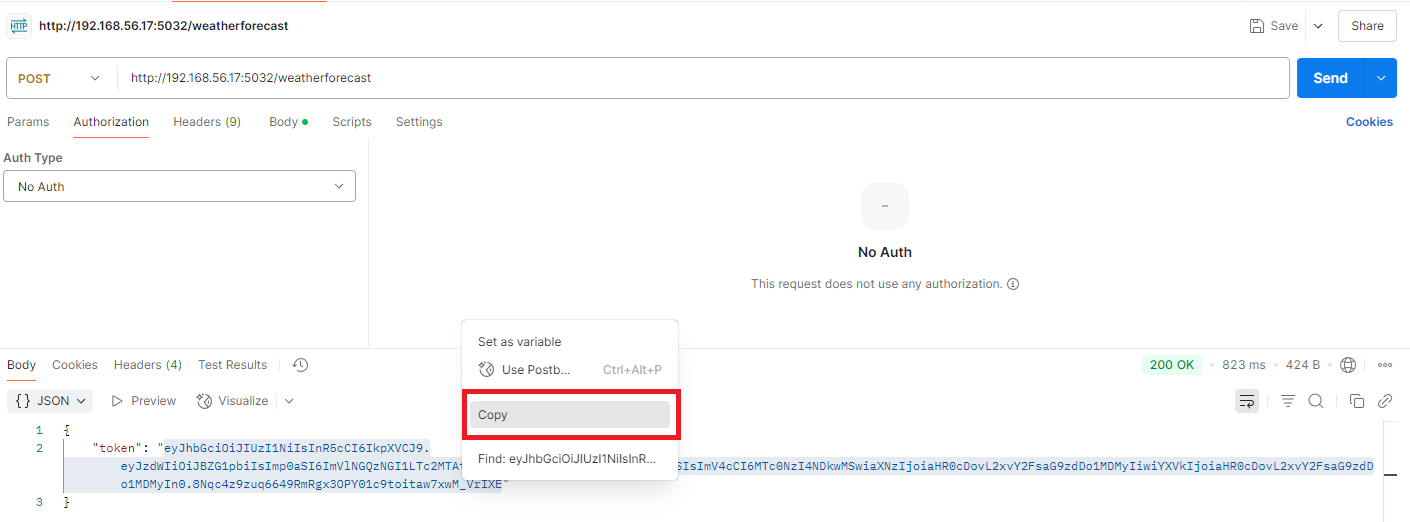
**Note**: Since I am running my development environment on my Linux VM, I edited the file Properties/launchSettings.json, changing the ‘localhost’ to 0.0.0.0. This allowed me to access this WebApi with a browser in my Windows 11 host.

**Scenarios: (Happy path)**

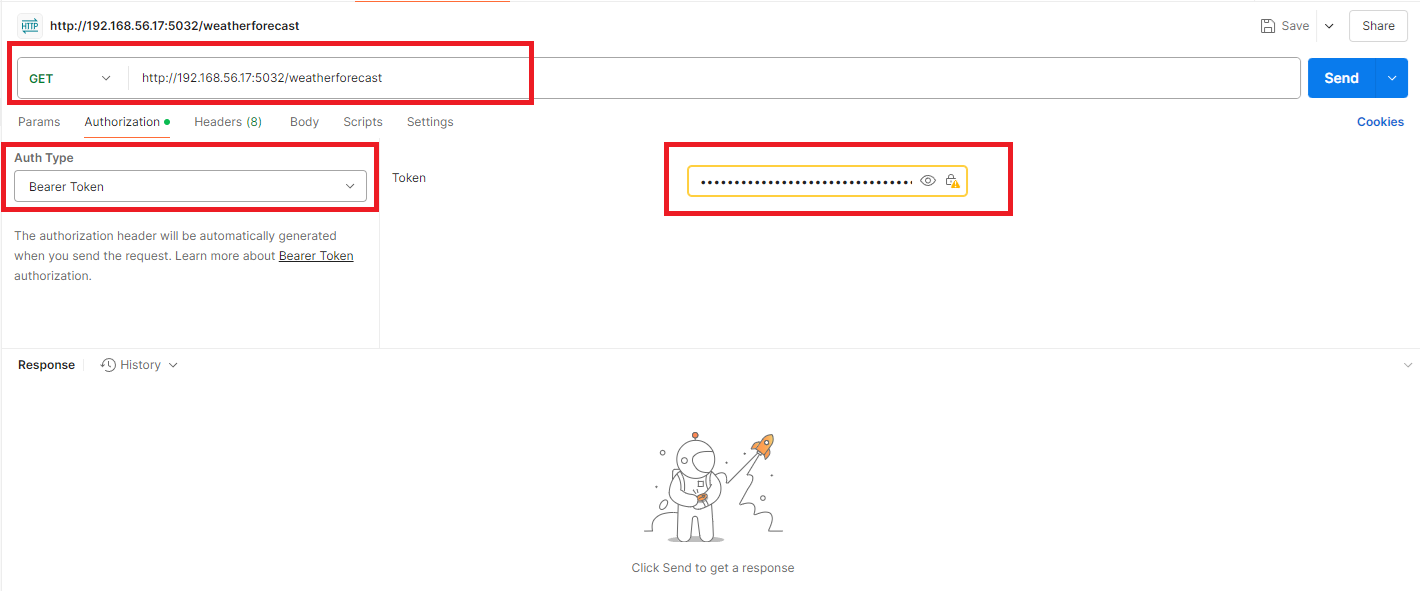
1. Use Postman or Swagger to call the POST method, providing in the JSON format the username: Admin and Password: Admin123
   1. The answer for this request will be a token that you need to use in your GET request for weatherforecast.
2. Use Postman or Swagger to call the GET method, providing the Token returned in the previous call.
   1. The answer for this request will be a list of weather forecasts
3. Getting the token. Calling the POST method

**Note:** This method doesn’t have ‘**RequireAuthorization’**





1. Obtaining the weather forecast by calling the GET method.



Using Swagger.

The process with Swagger is similar.

You will notice the ‘Authorize’ button added to the Swagger UI.

