Asp Net Core 5 web API token based authentication example using JWT

NOVEMBER 29, 2020 BY MEBAKAR1005 — 2 COMMENTS

In this tutorial, we are going to cover a **web api token based authentication** example using **JWT** in **Asp.Net Core 5** using visual studio 2019. So, first-of-all, we will create a new Asp.Net Core 5 web API project and then we will see how to implement Microsoft Identity and then finally we will see how to implement token based authentication using JWT in Asp.Net Core 5 web API app.

If you have a project in **Asp.Net Core 3.1** and want to upgrade it to Asp.Net Core 5 then click here to go back to the tutorial where we have discu

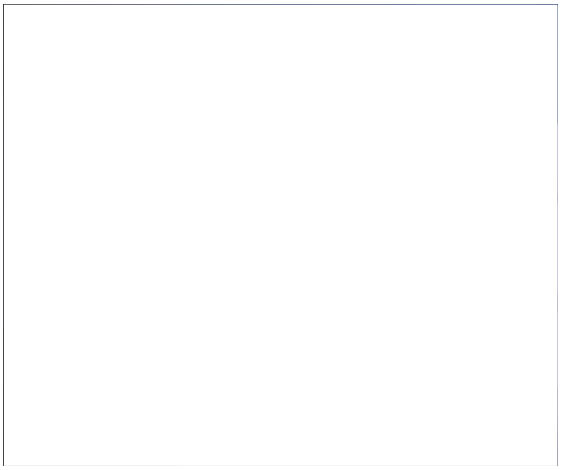
Prerequisites

There are some prerequisites for this tutorial. So, before going to next you must install these dependencies below.

- **Visual Studio 2019**: Click here to download it according to your machine and then install it. Make sure you have installed the **visual studio 2019 version 16.8** or later.
- **.NET Core 5**: Click here to download it according to your machine and install **.NET Core 5**
- **Postman:** Click here to download and then install it.
- **SQL Server**

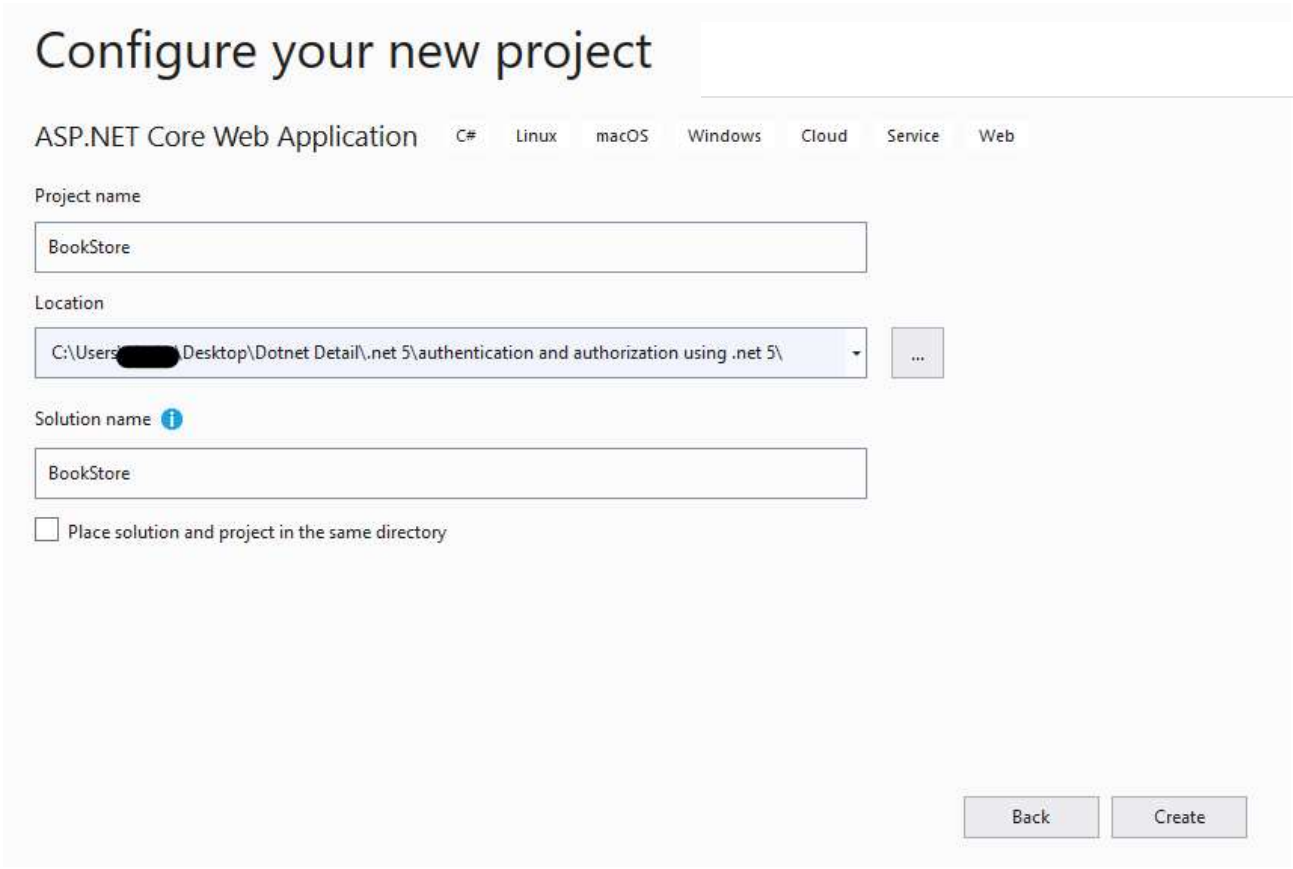
How to implement token based authentication (JWT) in Asp.Net Core 5 web API?

Let's start how to implement a web API token based authentication example using **Asp.Net Core 5**.

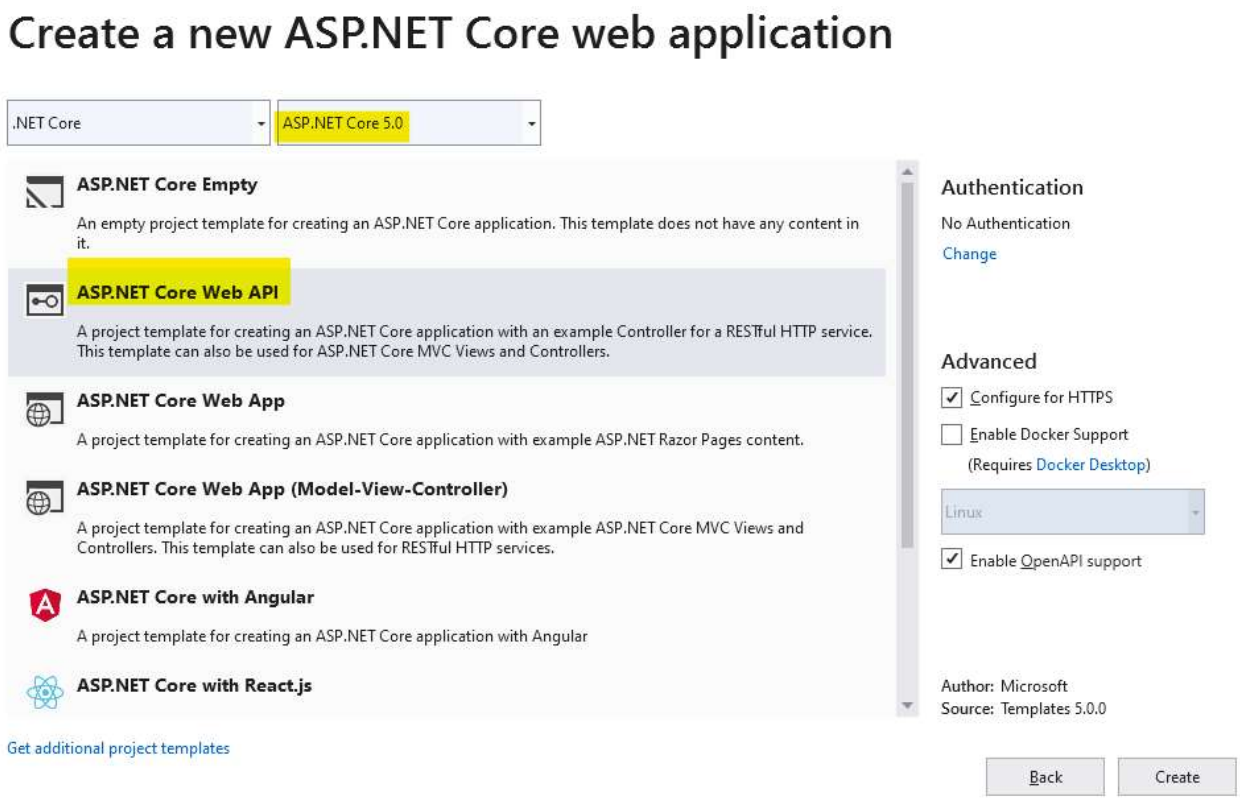


Step # 1: How to create Asp Net Core 5 web API using visual studio 2019?

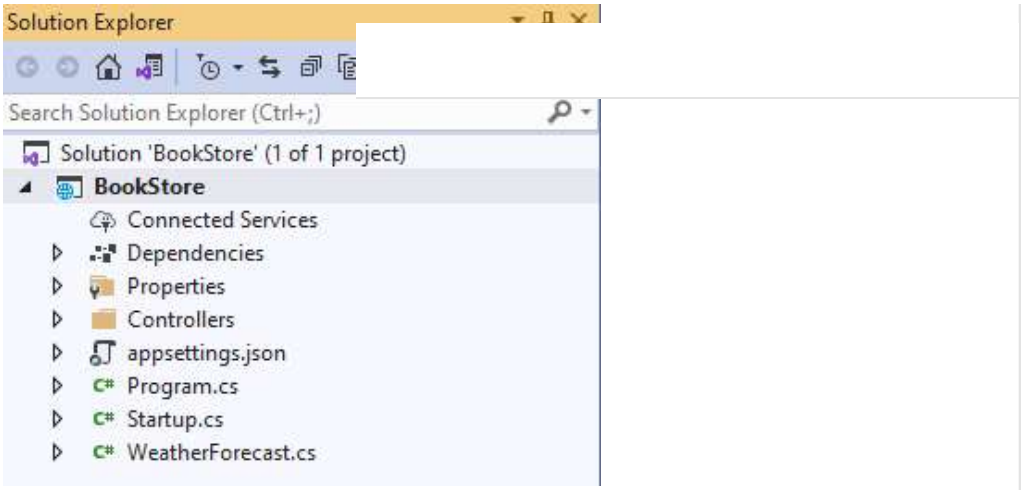
First-of-all, now in this step we will create a new asp.net core 5 web api using visual studio 2019. So, go to **Visual Studio 2019** and then click on the **Create a new project** and then select **Asp.Net Core Web Application** and then enter the name in the **Project name** field and then set **location** of project directory and then click on the **Create** button.



After clicking on the create button, then you will see a new popup as you do see below in the screenshot. Now, select **Asp.Net Core 5.0** from the dropdown and then select **Asp.Net Core Web API** template and then click on the **Create** button.



After clicking on the **Create** button, then you will see a new **Asp.Net Core 5.0 Web API** project is created with the name of **BookStore.**



Step # 2: How to setup Database and implement Microsoft Identity in Asp.Net Core 5 Web API project?

Now, in this step, we will setup a new database in **Asp.Net Core 5.0 web api** project. So, go to the project folder structure and then open **appsettings.json** file and then add a connection string as you do see in the below file code.

```
{
    "ConnectionStrings": {
        "DefaultConnection": "Data Source=DataSourceName; Initial Catalog=BookStoreDB;Integrated Se
    },
    "Logging": {
        "LogLevel": {
            "Default": "Information",
            "Microsoft": "Warning",
            "Microsoft.Hosting.Lifetime": "Information"
        }
    },
    "AllowedHosts": "*"
}
```

connection string - asp.net core 5 Authentication hosted with ♡ by GitHub view raw

Now, go to the project folder structure and then enter a new folder with the name of **Data.** Now, we will add some classes related to the databases like **ApplicationUser, BookStoreDbContext** and etc.

READ **How to add SignalR to Asp.Net Core 3.0 app using Visual Studio 2019**

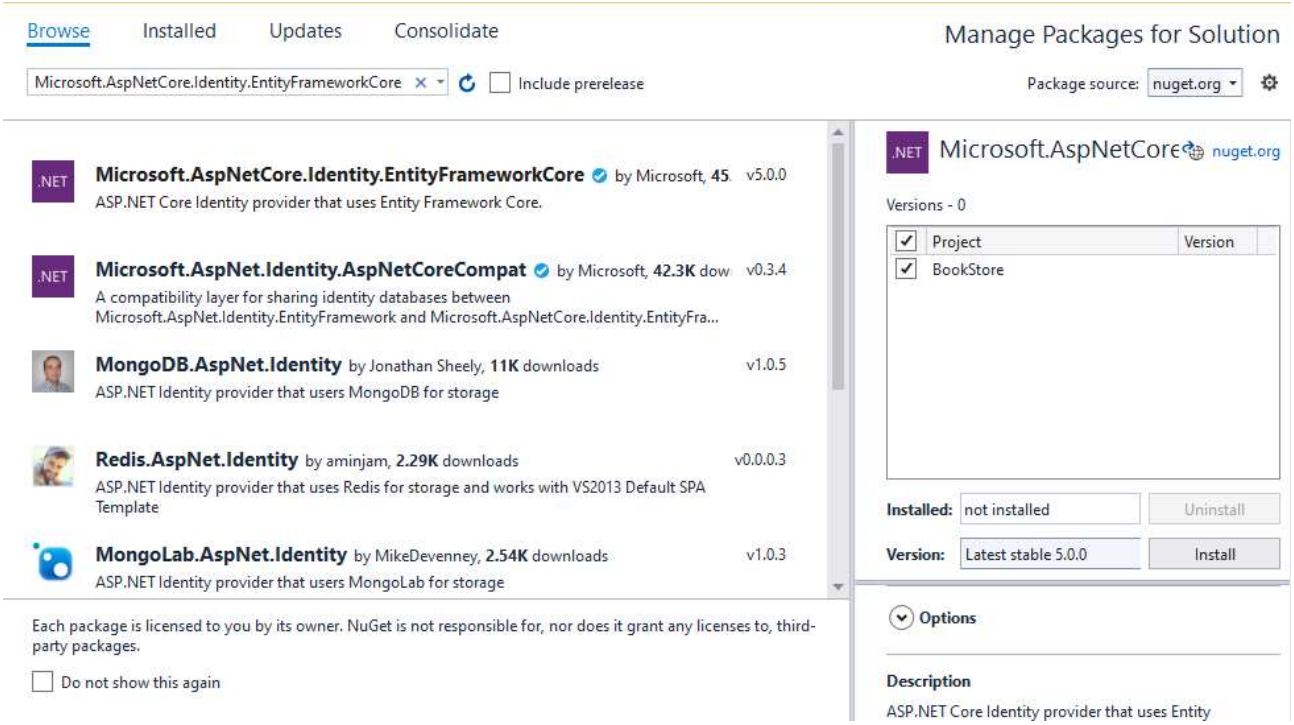
Note: – don't forget to add libraries.

```
using Microsoft.AspNetCore.Identity;
using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace BookStore.Data
{
    public class ApplicationUser : IdentityUser
    {
    }
}
```

Application User Class - Asp.Net Core 5 authentication hosted with ♡ by GitHub view raw

Now, go to **NuGet Packages** and then search for **Microsoft.AspNetCore.Identity.EntityFrameworkCore** and then select it and then install as you do see in the below file.



Now, add another class inside the **Data** folder with the name of **BookStoreDbContext** and then write the code as you do see below in the file.

```
using Microsoft.AspNetCore.Identity.EntityFramework
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace BookStore.Data
{
    public class BookStoreDbContext : IdentityDbContext<ApplicationUser>
    {
        public BookStoreDbContext(DbContextOptions<BookStoreDbContext> options) : base(options)
        {
        }
        protected override void OnModelCreating(ModelBuilder builder)
        {
            base.OnModelCreating(builder);
        }
    }
}
```

BookStore Db Context - Asp.Net Core 5 authentication hosted with ♡ by GitHub view raw

Note: – don't forget to add libraries.

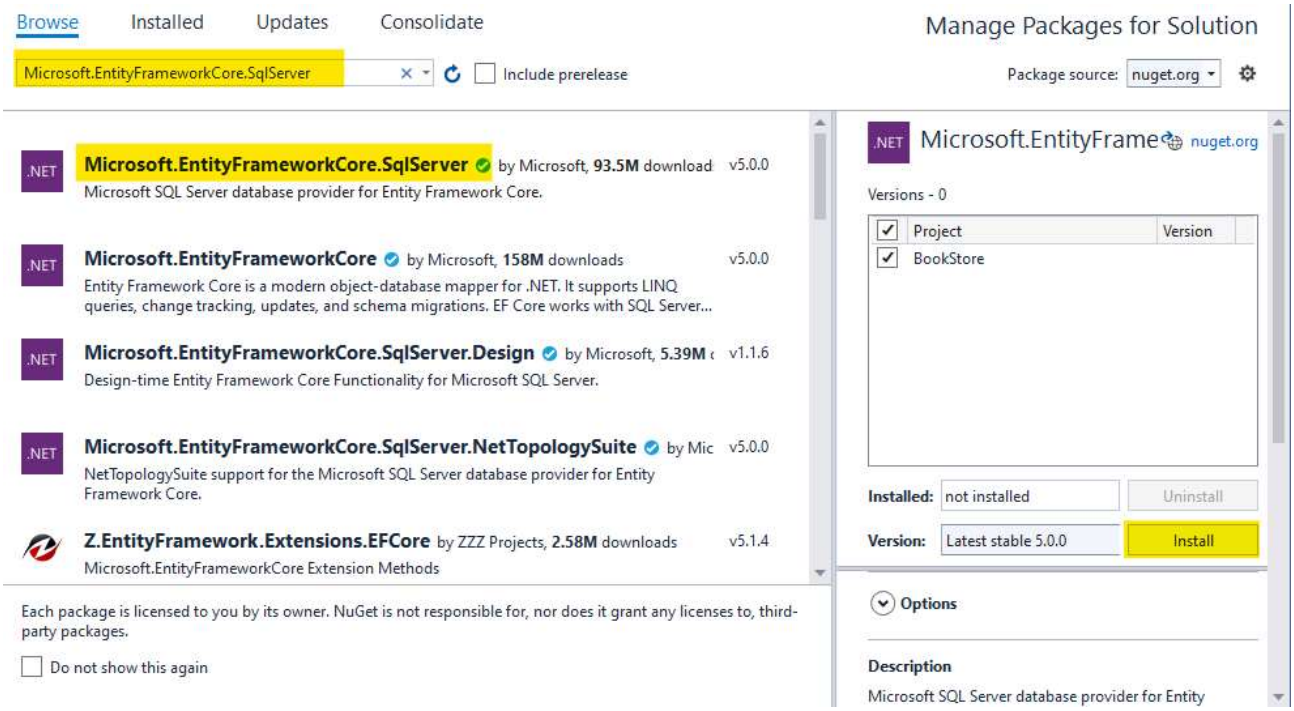
Now, we will add a new seed class which will add a demo data for the user account. So, right click on the **Data** folder and then click on the **Add** and then click on the **Class** and then enter the name of class (**E.g. SeedDB**) and then write the code as you do see below in the file.

```
using Microsoft.AspNetCore.Identity;
using Microsoft.Extensions.DependencyInjection;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace BookStore.Data
{
    public class SeedDB
    {
        public static void Initialize(IServiceProvider serviceProvider)
        {
            var context = serviceProvider.GetRequiredService<BookStoreDbContext>();
            var userManager = serviceProvider.GetRequiredService<UserManager<ApplicationUser>>();
            context.Database.EnsureCreated();
            if (!context.Users.Any())
            {
                ApplicationUser user = new ApplicationUser()
                {
```

```
                    Email = "test@gmail.com",
                    UserName = "test",
                    SecurityStamp = Guid.NewGuid().ToString()
                };
                userManager.CreateAsync(user, "Test@123");
            }
        }
    }
}
```

SeedDB - Asp.Net Core 5 authentication hosted with ♡ by GitHub view raw

Now, go to **NuGet Packages** and then search for **Microsoft.EntityFrameworkCore.SqlServer** and then install it as you do see below in the screenshot.



Now, we will add services to **Startup.cs** class. So, go to the project folder structure and then open the **Startup.cs** class and then write the code as you do see in the below file's **line # 33 to 36** and **line # 52.**

```
using BookStore.Data;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.HttpsPolicy;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
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 BookStore
{
    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.AddDbContext<BookStoreDbContext>(options => options.UseSqlServer(Configuration
            services.AddIdentity<ApplicationUser, IdentityRole>()
                .AddEntityFrameworkStores<BookStoreDbContext>()
                .AddDefaultTokenProviders();
            services.AddSwaggerGen(c =>
            {
                c.SwaggerDoc("v1", new OpenApiInfo { Title = "BookStore", Version = "v1" });
            });
        }

        // This method gets called by the runtime. Use this method to configure the HTTP request pi
        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
                app.UseSwagger();
                app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json", "BookStore v1")
            }
            SeedDB.Initialize(app.ApplicationServices.GetRequiredService<IServiceScopeFactory>().Cr
            app.UseHttpsRedirection();
            app.UseRouting();
            app.UseAuthorization();
            app.UseEndpoints(endpoints =>
            {
                endpoints.MapControllers();
            });
        }
    }
```

```
62  }
```

Startup.cs for services - asp.net core 5 authentication hosted with ♡ by GitHub view raw

Let's understand the above code.

Line # 33: In this line, we are adding the service for SQL server db.

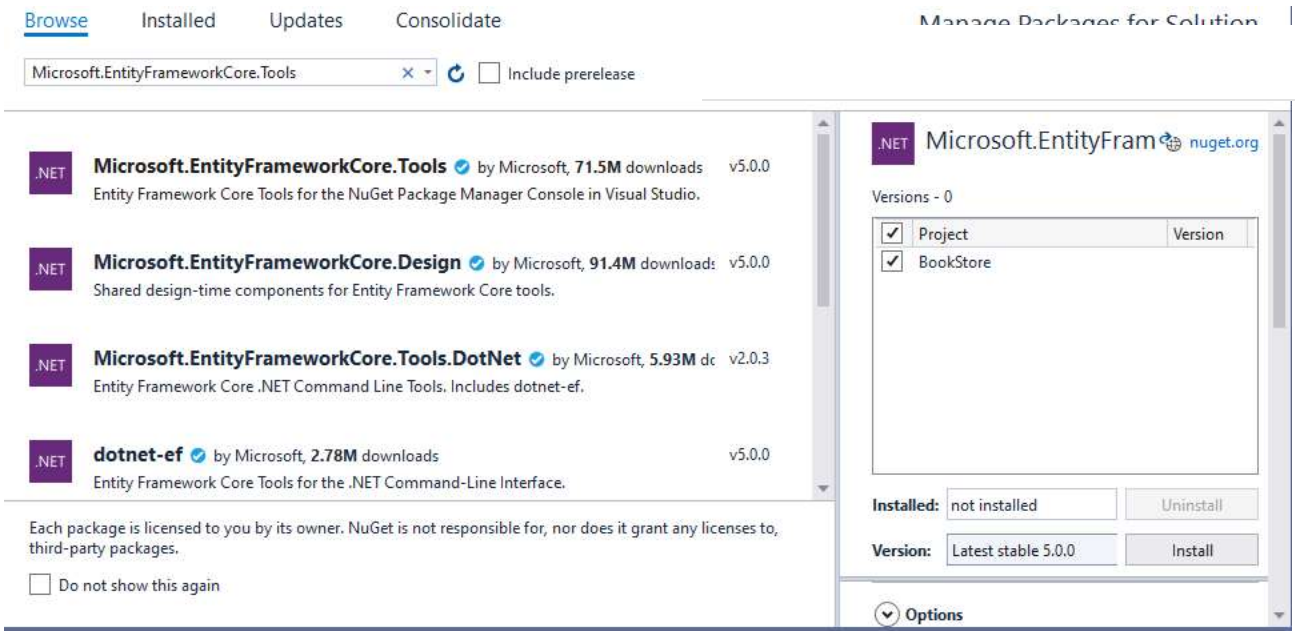
Line # 34 to 36: In this line, we are adding service for Microsoft Identity.

Line # 56: In this line, we are initializing the **SeedDB** class to create a new user.

Add Entity Framework Core Migrations

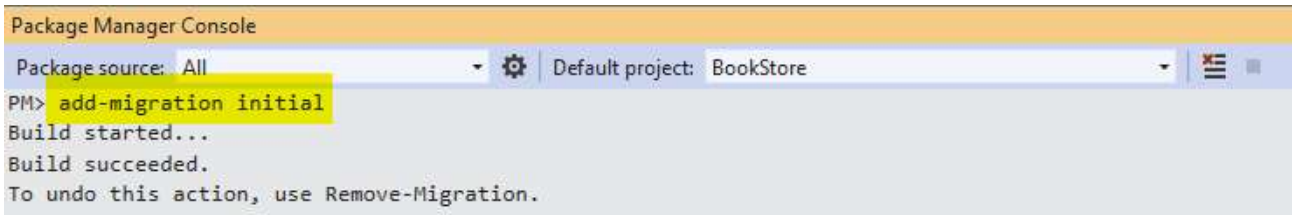
Now, we will run the migration. So, go to **Tools** and then **NuGet Package Manager** and then go to **Manage NuGet Packages for solutions...** and then search for **Microsoft.EntityFrameworkCore.Tools** and then install it as you do see below in the screenshot.

READ **How to add Authentication to React app using Asp.Net Core 3 in VS2019**



So, go to **Package Manager Console** and then run the below command and then you will see the migration folder will be created.

```
add-migration initial
```



Now, run below command.

```
update-database
```

```
PM> update-database
Build started...
Build succeeded.
Security Warning: The negotiated TLS 1.0 is an insecure protocol and is supported for backward compatibility only. The recommended
 protocol version is TLS 1.2 and later.
Security Warning: The negotiated TLS 1.0 is an insecure protocol and is supported for backward compatibility only. The recommended
 protocol version is TLS 1.2 and later.
Security Warning: The negotiated TLS 1.0 is an insecure protocol and is supported for backward compatibility only. The recommended
 protocol version is TLS 1.2 and later.
Done.
```

Now, the database and identity implementation part is completed.

Step # 3: How to implement token based authentication using JWT in Asp.Net Core 5 web api?

Now, in this step, we will see how to implement **token based authentication** using **JWT** in **Asp.Net Core 5.0 web API.**

So, go to project folder structure and then add a new folder with the name of **Models.** Then right click on the **Models** folder and then click on the **Add** and then click on th

then enter the name of class (**E.g. LoginModel**) and then click on the **Add** button. Now, write some properties as you do see below in the file.

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Threading.Tasks;

namespace BookStore.Models
{
    public class LoginModel
    {
        [Required(ErrorMessage = "User Name is required")]
        public string Username { get; set; }

        [Required(ErrorMessage = "Password is required")]
        public string Password { get; set; }
    }
}
```

Login Model - Asp.Net Core 5 authentication hosted with ♡ by GitHub view raw

Now, go to the project folder structure and then right click on the **Controllers** folder and then select **Add** and then click on the **Controller...** and then select the **MVC Controller – Empty** from the middle pane and then click on the **Add** button and then enter the name of the controller (**E.g. AuthenticateController**) and then click the **Add** button. So, it will create a new controller class with the name of **AuthenticateController.cs.**

Now, write the code as you do see below in the file.

```
using BookStore.Data;
using BookStore.Models;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
```

```
using Microsoft.IdentityModel.Tokens;
using System;
using System.Collections.Generic;
using System.IdentityModel.Tokens.Jwt;
using System.Linq;
using System.Security.Claims;
using System.Text;
using System.Threading.Tasks;

namespace BookStore.Controllers
{
    [Route("api/[controller]")]
    public class AuthenticateController : Controller
    {
        private UserManager<ApplicationUser> userManager;

        public AuthenticateController(UserManager<ApplicationUser> userManager)
        {
            this.userManager = userManager;
        }

        [HttpPost]
        [Route("login")]
        public async Task<IActionResult> Login([FromBody] LoginModel model)
        {
            var user = await userManager.FindByNameAsync(model.Username);
            if (user != null && await userManager.CheckPasswordAsync(user, model.Password))
            {
                var authClaims = new[]
                {
                    new Claim(JwtRegisteredClaimNames.Sub, user.UserName),
                    new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString())
                };
                var authSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("7S79jvOkEdwoR
                var token = new JwtSecurityToken(
                    issuer: "https://dotnetdetail.net",
                    audience: "https://dotnetdetail.net",
                    expires: DateTime.Now.AddDays(5),
                    claims: authClaims,
                    signingCredentials: new Microsoft.IdentityModel.Tokens.SigningCredentials(authS
                    );
                return Ok(new
                {
                    token = new JwtSecurityTokenHandler().WriteToken(token),
                    expiration = token.ValidTo
                });
            }
            return Unauthorized();
        }
```

```
54
55        }
56    }
```

Authenticate Controller - AspNet Core 5 Authentication hosted with ♡ by GitHub view raw

Let's understand the above code.

Line # 16: In this line, we are declaring the route path for API.

Line # 19: In this line, we are injecting the User Manager.

Line # 30: In this line, we are getting the specific user.

Line # 31: In this line, we are checking the user value and password. If the condition is true, then we will generate the token otherwise it will return an Unauthorized response.

Line # 33 to 37: In this block of code, we are creating claims.

Line # 38: In this line, we are creating the signing key.

READ **Entity Framework Core 3.0 Bulk Insert Update and Delete - Asp.Net Core 3.0**

Line # 39 to 45: In this block of code, we are generating the token using JWT.

Line # 46 to 50: In this block of code, we are returning the status with token and expiration time after generating the token successfully.

Now, go to the project folder structure and then go to **Controllers** folder and then open the **WeatherForecastController.cs** and then just put the following code as you do see below in the file's **line # 11**.

```
1  using Microsoft.AspNetCore.Authorization;
2  using Microsoft.AspNetCore.Mvc;
3  using Microsoft.Extensions.Logging;
4  using System;
5  using System.Collections.Generic;
6  using System.Linq;
7  using System.Threading.Tasks;
8
9  namespace BookStore.Controllers
10 {
11     [Authorize]
12     [ApiController]
13     [Route("[controller]")]
14     public class WeatherForecastController : ControllerBase
15     {
16         private static readonly string[] Summaries = new[]
17         {
18             "Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot", "Sweltering",
19         };
20
21         private readonly ILogger<WeatherForecastController> _logger;
22
23         public WeatherForecastController(ILogger<WeatherForecastController> logger)
24         {
25             _logger = logger;
26         }
27
28         [HttpGet]
29         public IEnumerable<WeatherForecast> Get()
30         {
31             var rng = new Random();
32             return Enumerable.Range(1, 5).Select(index => new WeatherForecast
33             {
34                 Date = DateTime.Now.AddDays(index),
35                 TemperatureC = rng.Next(-20, 55),
36                 Summary = Summaries[rng.Next(Summaries.Length)]
37             })
38             .ToArray();
39         }
40     }
41 }
```

Authorize in weatherforecast contoller - asp.net core 5 authentication hosted with ♡ by GitHub view raw

Now, we will add authentication service in the **startup** class as we do see in the below file's **line # 40 to 58** and **line # 77.**

```
1  using BookStore.Data;
2  using Microsoft.AspNetCore.Authentication.JwtBearer;
3  using Microsoft.AspNetCore.Builder;
4  using Microsoft.AspNetCore.Hosting;
5  using Microsoft.AspNetCore.HttpsPolicy;
6  using Microsoft.AspNetCore.Identity;
7  using Microsoft.AspNetCore.Mvc;
8  using Microsoft.EntityFrameworkCore;
9  using Microsoft.Extensions.Configuration;
10 using Microsoft.Extensions.DependencyInjection;
11 using Microsoft.Extensions.Hosting;
12 using Microsoft.Extensions.Logging;
13 using Microsoft.IdentityModel.Tokens;
14 using Microsoft.OpenApi.Models;
15 using System;
16 using System.Collections.Generic;
17 using System.Linq;
18 using System.Text;
19 using System.Threading.Tasks;
20
21 namespace BookStore
22 {
23     public class Startup
24     {
25         public Startup(IConfiguration configuration)
26         {
27             Configuration = configuration;
28         }
29
30         public IConfiguration Configuration { get; }
31
32         // This method gets called by the runtime. Use this method to add services to the container
33         public void ConfigureServices(IServiceCollection services)
34         {
35             services.AddControllers();
36             services.AddDbContext<BookStoreDbContext>(options => options.UseSqlServer(Configuration
37             services.AddIdentity<ApplicationUser, IdentityRole>()
38                 .AddEntityFrameworkStores<BookStoreDbContext>()
39                 .AddDefaultTokenProviders();
40             services.AddAuthentication(options =>
41             {
42                 options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
43                 options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
44                 options.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
45             })
46             .AddJwtBearer(options =>
```

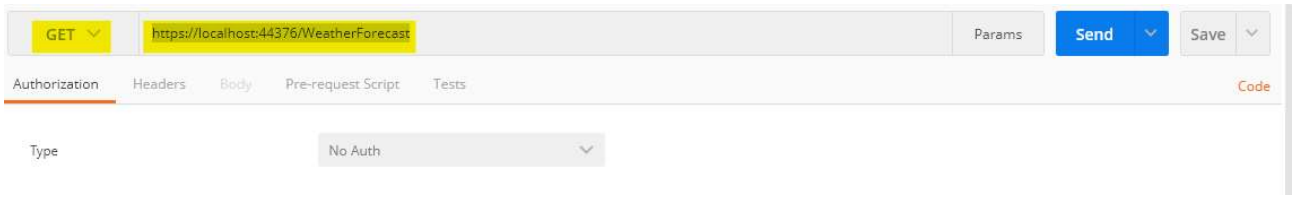
```
            {
                options.SaveToken = true;
                options.RequireHttpsMetadata = false;
                options.TokenValidationParameters = new Microsoft.IdentityModel.Tokens.TokenValidat
                {
                    ValidateIssuer = true,
                    ValidateAudience = true,
                    ValidAudience = "https://dotnetdetail.net",
                    ValidIssuer = "https://dotnetdetail.net",
                    IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("7S79jvOkEdw
                };
            });
            services.AddSwaggerGen(c =>
            {
                c.SwaggerDoc("v1", new OpenApiInfo { Title = "BookStore", Version = "v1" });
            });
        }

        // This method gets called by the runtime. Use this method to configure the HTTP request pi
        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
                app.UseSwagger();
                app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json", "BookStore v1")
            }
            SeedDB.Initialize(app.ApplicationServices.GetRequiredService<IServiceScopeFactory>().Cr
            app.UseHttpsRedirection();
            app.UseRouting();
            app.UseAuthentication();
            app.UseAuthorization();
            app.UseEndpoints(endpoints =>
            {
                endpoints.MapControllers();
            });
        }
    }
}
```

startup with authorization - asp.net core 5 authentication hosted with ♡ by GitHub view raw

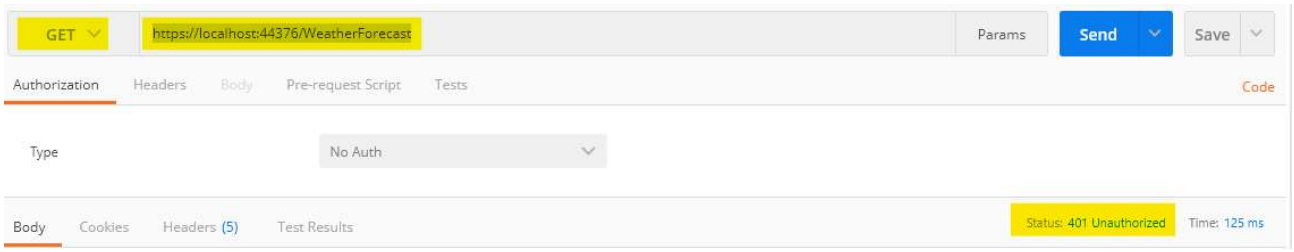
Test Project

Now, in this step, we will test our project and we will see the output using the **postman.** So, run your project by pressing **f5** or just clicking on the **IIS Express.**

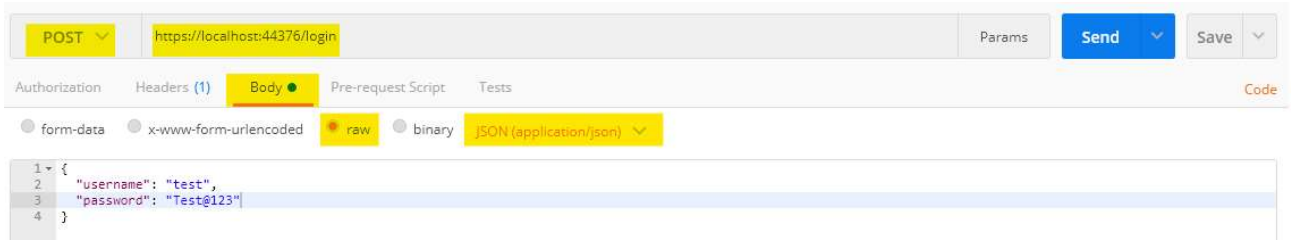
Now, open the postman and then enter the API path (**E.g. https://localhost:44376/WeatherForecast**) using the get request and then click on the **Send** button as you do see below in the screenshot.



After clicking on the **Send** button, then you will see the out with **unauthorized** status as you do see below in the screenshot.



Now, we will generate the token using the login method by passing the **username** and **password** as you do see below in the screenshot.



If the above login request is generated successfully, then you will see the output as you do see below in the screenshot.

Now, copy the token value and then enter the url (**https://localhost:44376/weatherforecast**) with **Get** method and then go to **Headers** tab and then enter the **key (Authorization)** and then **Value(Bearer <Token Value>)** and then click on the **Send** button.

After clicking on the **Send** button, then you will see the output with 200 status code as you do see below in the screenshot.

Congratulation, we have successfully created a **web api token based authentication** using **JWT** in **Asp.Net Core 5** with Visual Studio 2019

Note:- If you found this blog helpful then: Buy me a coffee

« How to upgrade Asp.Net Core 3.1 App to Asp.Net Core 5

Build a Microservice using Asp Net Core 5 and Docker »

FILED UNDER: ASP.NET CORE, ASP.NET CORE 5, AUTHENTICATION AND AUTHORIZATION, SECURITY, WEB API
TAGGED WITH: .NET CORE 5, ASP.NET CORE 5, ASP.NET CORE AUTHENTICATION, AUTHENTICATION, AUTHENTICATION AND AUTHORIZATION, WEB API

Comments

Sándor Hatvani says
FEBRUARY 3, 2021 AT 1:41 PM

Hi,
Thank you for you article. By the way I would ask you if authentication and authorization can happen by on-premise AD instead of EF?
Maybe you could suggest how to do it, please? ☺

Reply

Nick Taglianetti says
APRIL 16, 2021 AT 12:27 AM

Thank you for this tutorial. This is the first token based authentication tutorial that I've successfully implemented for a .NET Core 3.1 Web API solution to actually get a token back after building the required authentication controller. I was able to adapt it simply by installing the specified Nuget packages with the latest 3.1.x targeted framework. HOWEVER, I am getting a 401 error when trying to send a request to one of my endpoints whose controller is decorated with the [Authorize] attribute. Any ideas what I could be missing? I'm getting the error when testing with both the Swagger (swashbuckle) UI and via Postman. I followed this guide exactly other than using the .NET Core 3.1 Nuget packages. Any advice would be greatly appreciated!

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

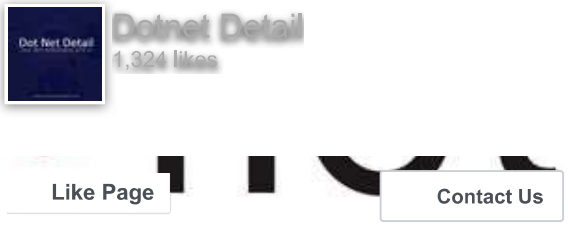
Notify me of follow-up comments by email.

Notify me of new posts by email.

POST COMMENT

Get Notifications

JOIN US



SUBSCRIBE TO BLOG VIA EMAIL

Enter your email address to subscribe.

Email Address

Get Notifications

RECENT POSTS

- Top LINQ interview questions and answers.
- Difference between .NET Core and .NET Framework
- Top 10 Programming Languages of the Future.
- Top MVC Interview Questions & Answers
- 20 best Data Science Books: Beginner to Advanced Level
- 10 Best Machine Learning Laptops