



# JWT Token Authentication Using The .NET Core

[WebAPI.zip](#)

[Download Free .NET & JAVA Files API](#)

We are going to discuss JWT Token Authentication and Implementation using .NET Core API 6.

Before looking into this article, visit my below blog to understand the basics and details of JWT Token Authentication and Authorization and how things work using JWT.

- [Introduction and Detail about JWT Token Authentication and Authorization](#)

Let's start the implementation of the .NET Core 6 Web API,

## Step 1

Create the .NET Core 6 Web API Application

## Step 2

Install the following NuGet Packages which we are going to use throughout the application.

- Microsoft.AspNetCore.Authentication.JwtBearer
- Microsoft.EntityFrameworkCore
- Microsoft.EntityFrameworkCore.Design
- Microsoft.EntityFrameworkCore.SqlServer
- Microsoft.EntityFrameworkCore.Tools
- Newtonsoft.Json

### Step 3

```
1 namespace WebAPI.Model
2 {
3     public class Product
4     {
5         public int ProductId { get; set; }
6         public string? ProductName { get; set; }
7         public string? ProductDescription { get; set; }
8         public int ProductCost { get; set; }
9         public int ProductStock { get; set; }
10    }
11 }
```

### Step 4

Create one DbContextClass inside the Data folder used for Database operation.

```
1 using Microsoft.EntityFrameworkCore;
2 using WebAPI.Model;
3 namespace WebAPI.Data {
4     public class DbContextClass: DbContext {
5         protected readonly IConfiguration Configuration;
6         public DbContextClass(IConfiguration configuration) {
7             Configuration = configuration;
8         }
9         protected override void OnConfiguring(DbContextOptionsBuilder options) {
```

```
--  
12     public DbSet < Product > Products {  
13         get {  
14             return _context.Products;  
15         }  
16     }  
17 }
```

## Step 5

Later on, Create ProductController inside the Controllers folder.

```
1  using Microsoft.AspNetCore.Authorization;  
2  using Microsoft.AspNetCore.Mvc;  
3  using Microsoft.EntityFrameworkCore;  
4  using WebAPI.Cache;  
5  using WebAPI.Data;  
6  using WebAPI.Model;  
7  namespace WebAPI.Controllers {  
8      [Route("api/[controller]")]  
9      [ApiController, Authorize]  
10     public class ProductController: ControllerBase {  
11         private readonly DbContextClass _context;  
12         private readonly ICacheService _cacheService;  
13         public ProductController(DbContextClass context, ICacheService cacheService) {  
14             _context = context;  
15             _cacheService = cacheService;  
16         }  
17         [HttpGet]  
18         [Route("ProductsList")]
```

```
21 productCache = _cacheService.GetData < List < Product >> ("Product");
22 if (productCache == null) {
23     // Cache miss, fetch from database
24     if (product.Count > 0) {
25         productCache = product;
26         var expirationTime = DateTimeOffset.Now.AddMinutes(3.0);
27         _cacheService.SetData("Product", productCache, expirationTime);
28     }
29 }
30 return productCache;
31 }
32 [HttpGet]
33 [Route("ProductDetail")]
34 public async Task < ActionResult < Product >> Get(int id) {
35     var productCache = new Product();
36     var productCacheList = new List < Product > ();
37     productCacheList = _cacheService.GetData < List < Product >> ("Product");
38     productCache = productCacheList.Find(x => x.ProductId == id);
39     if (productCache == null) {
40         productCache = await _context.Products.FindAsync(id);
41     }
42     return productCache;
43 }
44 [HttpPost]
45 [Route("CreateProduct")]
46 public async Task < ActionResult < Product >> POST(Product product) {
47     _context.Products.Add(product);
48     await _context.SaveChangesAsync();
49 }
```

```
51         id = product.ProductId
52     }
53     }
54     [HttpPost]
55     [Route("DeleteProduct")]
56     public async Task < ActionResult < IEnumerable < Product >>> Delete(int id) {
57         var product = await _context.Products.FindAsync(id);
58         if (product == null) {
59             return NotFound();
60         }
61         _context.Products.Remove(product);
62         _cacheService.RemoveData("Product");
63         await _context.SaveChangesAsync();
64         return await _context.Products.ToListAsync();
65     }
66     [HttpPost]
67     [Route("UpdateProduct")]
68     public async Task < ActionResult < IEnumerable < Product >>> Update(int id, Product product) {
69         if (id != product.ProductId) {
70             return BadRequest();
71         }
72         var productData = await _context.Products.FindAsync(id);
73         if (productData == null) {
74             return NotFound();
75         }
76         productData.ProductCost = product.ProductCost;
77         productData.ProductDescription = product.ProductDescription;
78         productData.ProductName = product.ProductName;
```

```
await _context.SaveChangesAsync();  
return await _context.Products.ToListAsync();
```

```
}
```

```
}
```

## Step 6

Now, we are going to use the Redis cache inside this application. If you understand how distributed Redis Cache work for that check my following blog

- [Implementation of Redis Cache in .NET Core Web API](#)

## Step 7

Create the Cache folder inside the solution and create a few classes for Redis and Connection Helper. So, First, create ICacheService and CacheService for Redis Cache.

```
1 namespace WebAPI.Cache  
2 {  
3     public interface ICacheService  
4     {  
5         /// <summary>  
6         /// Get Data using key  
7         /// </summary>  
8         /// <typeparam name="T"></typeparam>  
9         /// <param name="key"></param>  
10        /// <returns></returns>  
11        T GetData<T>(string key);  
12    }
```

```
15    /// </summary>
16    /// <param name="key"></param>
17
18    /// <param name="value"></param>
19    /// <param name="expirationTime"></param>
20    /// <returns></returns>
21    bool SetData<T>(string key, T value, DateTimeOffset expirationTime);
22
23    /// <summary>
24    /// Remove Data
25    /// </summary>
26    /// <param name="key"></param>
27    /// <returns></returns>
28    object RemoveData(string key);
29 }
30 }
```

Next, Create CacheService class for Redis Cache-related functionality.

```
1  using Newtonsoft.Json;
2  using StackExchange.Redis;
3  namespace WebAPI.Cache {
4      public class CacheService: ICacheService {
5          private IDatabase _db;
6          public CacheService() {
7              ConfigureRedis();
8          }
9          private void ConfigureRedis() {
10             _db = ConnectionHelper.Connection.GetDatabase();
```

```
13         var value = _db.StringGet(key);
14         if (!string.IsNullOrEmpty(value))
15         {
16             }
17             return default;
18         }
19         public bool SetData < T > (string key, T value, DateTimeOffset expirationTime) {
20             TimeSpan expiryTime = expirationTime.DateTime.Subtract(DateTime.Now);
21             var isSet = _db.StringSet(key, JsonConvert.SerializeObject(value), expiryTime);
22             return isSet;
23         }
24         public object RemoveData(string key) {
25             bool _isKeyExist = _db.KeyExists(key);
26             if (_isKeyExist == true) {
27                 return _db.KeyDelete(key);
28             }
29             return false;
30         }
31     }
32 }
```

## Step 8

Create ConfigurationManager class which we use to configure the appsetting.json file

```
1 namespace WebAPI {
2     static class ConfigurationManager {
3         public static IConfiguration AppSetting {
4             get;
```



```
AppSetting = new ConfigurationBuilder().SetBasePath(Directory.GetCurrentDirectory()).AddJsonSources("AppSettings", "AppSettings.json").Build();
```

```
}
```

## Step 9

Next, Create ConnectionHelper class inside the cache folder to get the RedisURL and configure that into the application.

```
1 using StackExchange.Redis;
2 namespace WebAPI.Cache {
3     public class ConnectionHelper {
4         static ConnectionHelper() {
5             ConnectionHelper.lazyConnection = new Lazy < ConnectionMultiplexer > (() => {
6                 return ConnectionMultiplexer.Connect(ConfigurationManager.AppSettings["RedisURL"]);
7             });
8         }
9         private static Lazy < ConnectionMultiplexer > lazyConnection;
10        public static ConnectionMultiplexer Connection {
11            get {
12                return lazyConnection.Value;
13            }
14        }
15    }
16 }
```

## Step 10

Now, we are going to create Login and JWTTokenResponse class for the JWT Authentication part

```
1  public string ? Username {
2      get;
3      set;
4  }
5
6  }
7  public string ? Password {
8      get;
9      set;
10 }
11 }
12 }
```

Also, Create JWTTokenResponse class for token

```
1 namespace WebAPI.Model {
2     public class JWTTokenResponse {
3         public string ? Token {
4             get;
5             set;
6         }
7     }
8 }
```

## Step 11

Later, Create AuthenticationController inside the Controllers for Authentication of User.

```
1 using Microsoft.AspNetCore.Mvc;
2 using Microsoft.IdentityModel.Tokens;
3 using System.IdentityModel.Tokens.Jwt;
4 using System.Security.Claims;
```

```
7 namespace WebAPI.Controllers {  
8     [Route("api/[controller]")]  
  
10    public class AuthenticationController: ControllerBase {  
11        [HttpPost("login")]  
12        public IActionResult Login([FromBody] Login user) {  
13            if (user is null) {  
14                return BadRequest("Invalid user request!!!");  
15            }  
16            if (user.UserName == "Jaydeep" && user.Password == "Pass@777") {  
17                var secretKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(ConfigurationManager.AppSettings["JWT:Va  
18                var signinCredentials = new SigningCredentials(secretKey, SecurityAlgorithms.HmacSha256  
19                var tokenOptions = new JwtSecurityToken(issuer: ConfigurationManager.AppSettings["JWT:Va  
20                var tokenString = new JwtSecurityTokenHandler().WriteToken(tokenOptions);  
21                return Ok(new JWTTokenResponse {  
22                    Token = tokenString  
23                });  
24            }  
25            return Unauthorized();  
26        }  
27    }  
28 }
```

- As you see above class we take the Username and Password from the User, then take the secret key which we put inside the appsettings.json file
- Next, create signing credentials using a secret key using HMAC SHA256 Crypto Algorithm for encoded string.
- Later on, we put a few attributes while creating tokens like signing credentials, expiration time, issuer, audience, and different types of claims as per our needs and requirement.

## Step 12

```
1  {
2      "Logging": {
3          "LogLevel": {
4              "Default": "Information",
5              "Microsoft.AspNetCore": "Warning"
6          }
7      },
8      "AllowedHosts": "*",
9      "RedisURL": "127.0.0.1:6379",
10     "JWT": {
11         "ValidAudience": "http://localhost:7299",
12         "ValidIssuer": "http://localhost:7299",
13         "Secret": "JWTAuthentication@777"
14     },
15     "ConnectionStrings": {
16         "DefaultConnection": "Data Source=Server;Initial Catalog=JWTDemo;User Id=sa;Password=****;"
17     }
18 }
```

## Step 13

Next, register all servers related to JWT Authentication, Swagger UI for Authentication, CORS Policy, and Cache Services inside the Program class as shown below

```
1  using Microsoft.AspNetCore.Authentication.JwtBearer;
2  using Microsoft.IdentityModel.Tokens;
```



```
35         new List < string > ()
36     },
37     };
38 });
39 builder.Services.AddScoped < ICacheService, CacheService > ();
40 builder.Services.AddDbContext < DbContextClass > ();
41 builder.Services.AddAuthentication(opt => {
42     opt.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
43     opt.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
44 }).AddJwtBearer(options => {
45     options.TokenValidationParameters = new TokenValidationParameters {
46         ValidateIssuer = true,
47         ValidateAudience = true,
48         ValidateLifetime = true,
49         ValidateIssuerSigningKey = true,
50         ValidIssuer = ConfigurationManager.AppSettings["JWT:ValidIssuer"],
51         ValidAudience = ConfigurationManager.AppSettings["JWT:ValidAudience"],
52         IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(ConfigurationManager.Ap
53     });
54 });
55 var app = builder.Build();
56 // Configure the HTTP request pipeline.
57 if (app.Environment.IsDevelopment()) {
58     app.UseSwagger();
59     app.UseSwaggerUI(options => {
60         options.SwaggerEndpoint("/swagger/V1/swagger.json", "Product WebAPI");
61     });
62 }
```

```
65 | app.UseAuthorization();  
66 | app.MapControllers();
```

```
68 | app.Run();
```

## Step 14

Finally, execute the following command in the Package Manager Console for the entity framework for data migration and database update.

- add-migration "First"
- update-database

## Step 15

Run the application and create the token after providing credentials and put it into the Authorize tab inside swagger UI as shown in the below image



Select a definition Product WebAPI

Ask Question

Product WebAPI

Authorize 

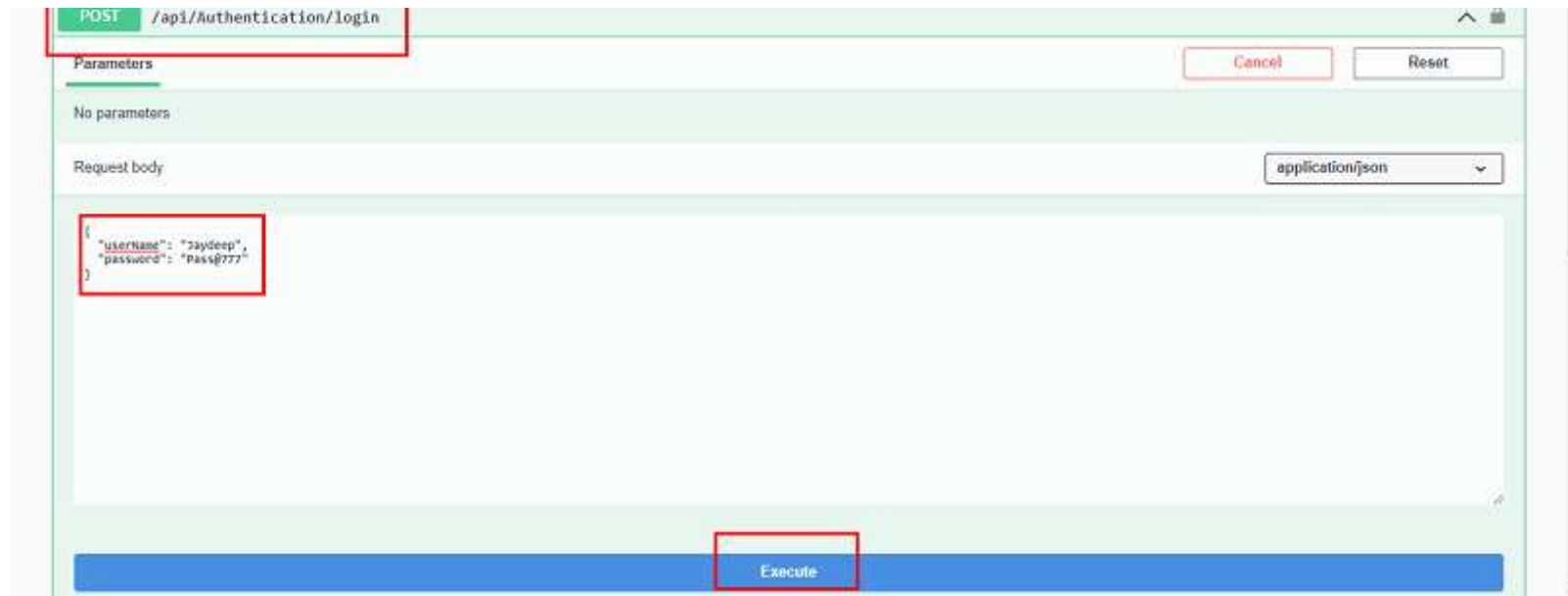
### Authentication

**POST** /api/Authentication/login

### Product

**GET** /api/Product/ProductsList**GET** /api/Product/ProductDetail**POST** /api/Product/CreateProduct**POST** /api/Product/DeleteProduct**POST** /api/Product/UpdateProduct



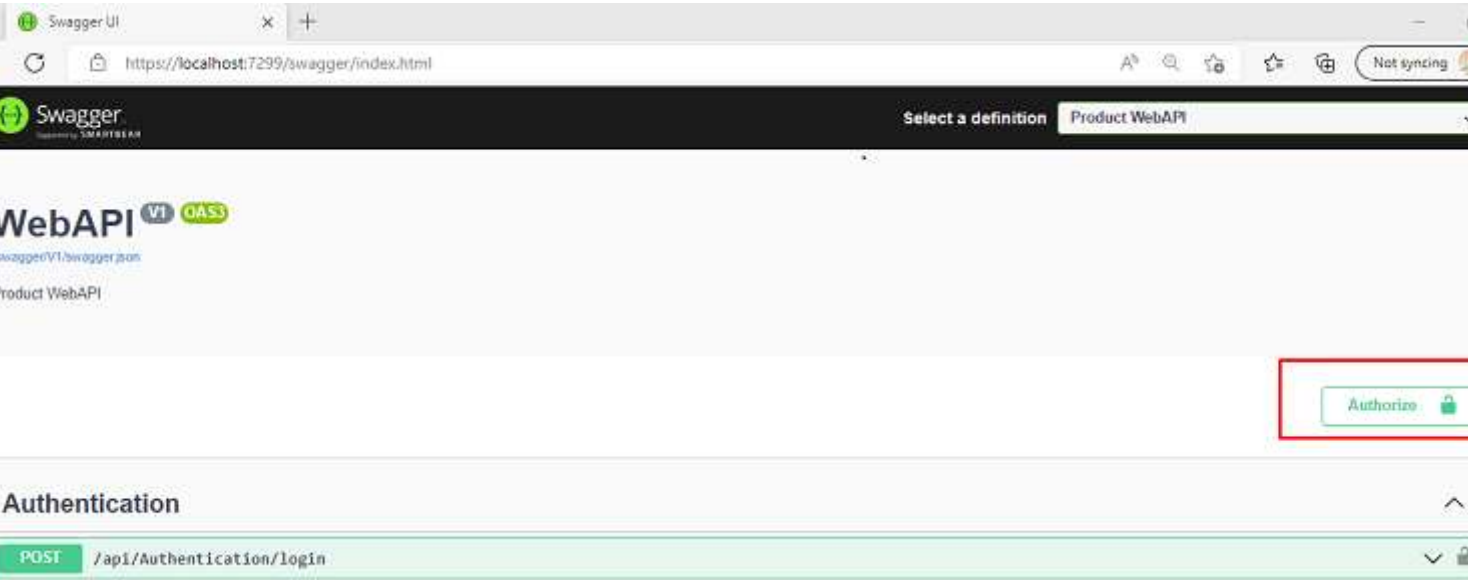


The screenshot shows a REST client interface with the following elements:

- Method and URL:** A green header bar contains the text "POST /api/Authentication/login".
- Parameters:** A section labeled "Parameters" with a sub-label "No parameters". It includes "Cancel" and "Reset" buttons.
- Request body:** A section labeled "Request body" with a dropdown menu set to "application/json".
- JSON Body:** A text area containing the JSON object: 

```
{  "username": "jaydeep",  "password": "Pass@777"}

```
- Execute Button:** A large blue button at the bottom center labeled "Execute".



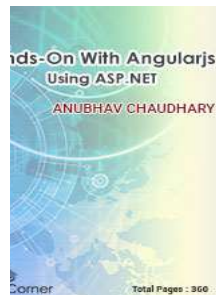
So, this is all about JWT Authentication in .NET Core 6 Web API.

I hope you understand the things I discussed and got an idea of how things work.

Happy Coding!

[.NET](#)[.NET Core](#)[ASP.NET](#)[C#](#)[C# Corner](#)

## OUR BOOKS



Jaydeep Patil *TOP 500*

C# Corner MVP | Fullstack Developer with 2.5+ yrs of experience in .NET Core API, Angular 8+, SQL Server, Docker, Kubernetes, and Azure

10 4

Ask Question



Follow Comments



Hi, thanks for the great tutorial. Please just add the step on how to setup Redis and updating the RedisURL setting in appsettings.json. It had me stuck for a bit.

Kutlo

Nov 10, 2022

NA 80 0

2 1 Reply



<https://www.c-sharpcorner.com/article/implementation-of-the-redis-cache-in-the-net-core-api/>

Jaydeep Patil

Nov 11, 2022

258 9.6k 529.4k

1



How to enable run in offline mode

hnieef

Jul 14, 2022

NA 247 0

1 0 Reply



Well Explain.....Keep posting

Hamid Khan

Jun 29, 2022

328 7.4k 1.7m

2 0 Reply

## FEATURED ARTICLES

Static Abstract Interface Members In C# 11 And Curiously Recurring Template Pattern

Containerize A .NET 6 App With Docker

Introduction To Homomorphic Encryption With Microsoft SEAL

How To Use ClickUp Efficiently For Project Management?



Learn C# 8.0

### CHALLENGE YOURSELF



Angular 13

### GET CERTIFIED



JavaScript Developer

