Custom JWT Token In ASP.NET Core We API

JWTTokenPOC.zip
Download Free .NET & JAVA Files API

Introduction

In this article, I am explaining how to implement custom JWT token authentication in ASP.NET Core 3.1 API.

For this, I have created the demo application which has two controllers “AthenticateController” and “UserController”.

AthenticateController has one endpoint “AuthenticateUser”, which will authenticate the user based on the user id and password and if user is valid then it will generate the JWT Token.

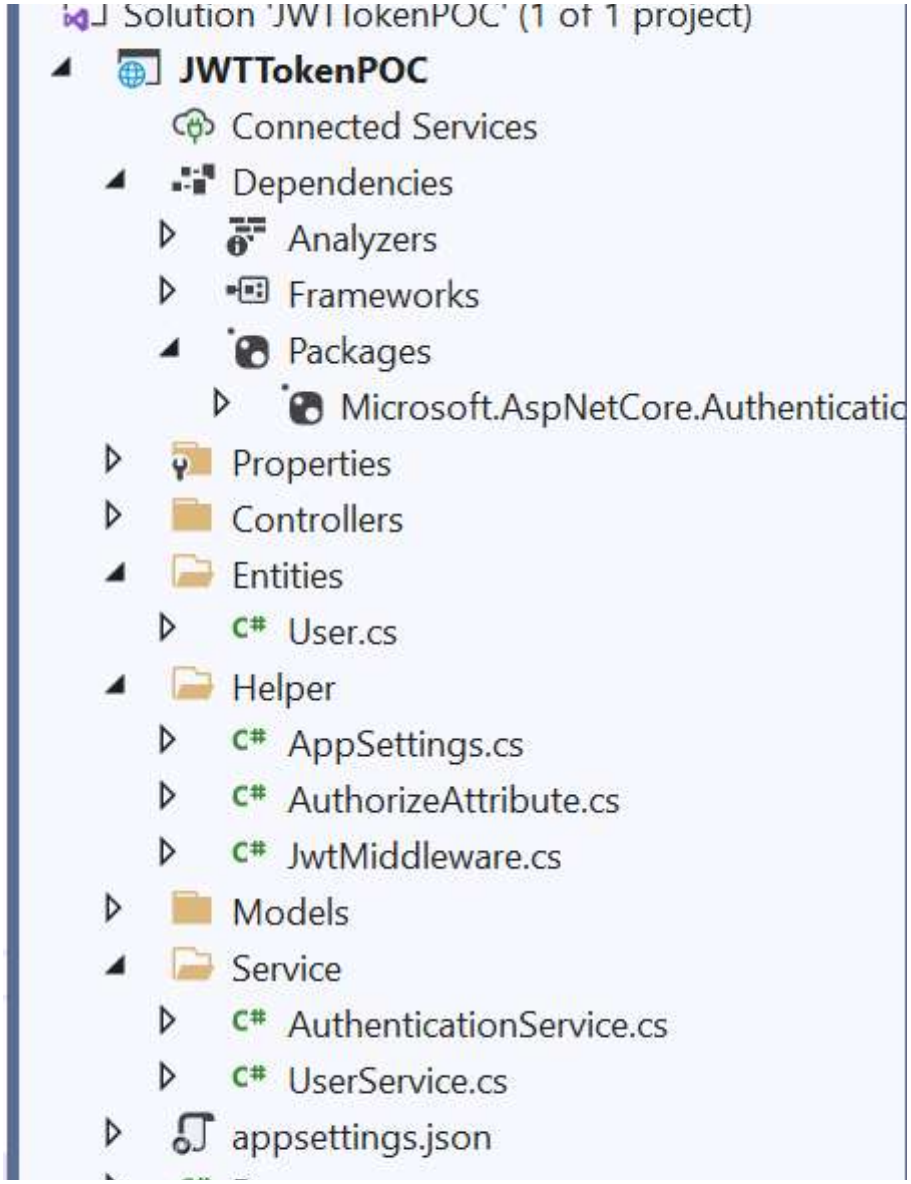
UserController has two endpoints “GetUsers” and “GetUserById”.

GetUsers

I have implemented Authorization filter to secure the endpoint and this endpoint accepts HTTP GET requests and returns a list of all the users in the application if the HTTP Authorization header contains a valid JWT token. If there is no auth token or the token is invalid, then a 401 Unauthorized response is returned.

Similarly, **GetUserById** returns user details by id if the HTTP Authorization header contains a valid JWT token.

Project Architecture



Follow below steps for project set up and generate JWT token,

Create the ASP.NET Core 3.1 Web API Application. I am giving application name as “JWTTokenPOC”

Step 2

Install “Microsoft.AspNetCore.Authentication.JwtBearer” using NuGet Package manager. I have installed 3.1.26 version.

Step 3

Create new folder “Entities” inside the solution and create an entity class “User”

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Text.Json.Serialization;
namespace JWTTokenPOC.Entities {
    public class User {
        public int Id {
            get;
            set;
        }
        public string FirstName {
            get;
            set;
        }
        public string LastName {
```

```
        }
        public string Username {
            set;
        }
        [JsonIgnore]
        public string Password {
            get;
            set;
        }
    }
}
```

Step 4

Create new folder Models inside the solution and create two models AuthenticateRequest and AuthenticateResponse.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.ComponentModel.DataAnnotations;
namespace JWTTokenPOC.Models {
    public class AuthenticateRequest {
        [Required]
        public string UserName {
            get;
            set;
```

```
        [Required]
        public string Password {
            get;
        }
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using JWTTokenPOC.Entities;
namespace JWTTokenPOC.Models {
    public class AuthenticateResponse {
        public int Id {
            get;
            set;
        }
        public string FirstName {
            get;
            set;
        }
        public string LastName {
            get;
            set;
        }
        public string Username {
            get;
```

```
43          public string Token {
44              get;
45          }
46          public AuthenticateResponse(User user, string token) {
47              Id = user.Id;
48              FirstName = user.FirstName;
49              LastName = user.LastName;
50              Username = user.Username;
51              Token = token;
52          }
53      }
54  }
```

Step 5

Create new folder “Helper” inside the solution and create two helper classes “AppSettings” and “AuthorizeAttribute” in that folder.

```
1  namespace JWTTokenPOC.Helper {
2      public class AppSettings {
3          public string Key {
4              get;
5              set;
6          }
7          public string Issuer {
8              get;
9              set;
10         }
```

```
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Filters;
using JWTTokenPOC.Entities;
namespace JWTTokenPOC.Helper {
    [AttributeUsage(AttributeTargets.Class | AttributeTargets.Method)]
    public class AuthorizeAttribute: Attribute, IAuthorizationFilter {
        public void OnAuthorization(AuthorizationFilterContext context) {
            var user = context.HttpContext.Items["User"];
            if (user == null) {
                // user not logged in
                context.Result = new JsonResult(new {
                    message = "Unauthorized"
                }) {
                    StatusCode = StatusCodes.Status401Unauthorized
                };
            }
        }
    }
}
```

Step 6

Add below appsetting in appsettings.json file.

```
    "Issuer": "atul1234"
}
```

Create new folder “Service” inside the solution and create two service classes “AuthenticationService” and “UserService” in that folder.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using JWTTokenPOC.Entities;
namespace JWTTokenPOC.Service {
    public interface IUserService {
        User GetById(int id);
        IEnumerable < User > GetAll();
    }
    public class UserService: IUserService {
        // List of user
        private List < User > _users = new List < User > {
            new User {
                Id = 1, FirstName = "mytest", LastName = "User", Username = "mytestuser", Password = "test123"
            },
            new User {
                Id = 2, FirstName = "mytest2", LastName = "User2", Username = "test", Password = "test"
            }
        };
        public IEnumerable < User > GetAll() {
```

```
        public User GetById(int id) {
            return users.FirstOrDefault(x => x.Id == id);
        }
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.IdentityModel.Tokens;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using Microsoft.Extensions.Options;
using System.Text;
using JWTTokenPOC.Entities;
using JWTTokenPOC.Helper;
using JWTTokenPOC.Models;
namespace JWTTokenPOC.Service {
    public interface IAuthenticationService {
        AuthenticateResponse Authenticate(AuthenticateRequest model);
    }
    public class AuthenticationService: IAuthenticationService {
        // here I have  hardcoded user for simplicity
        private List < User > _users = new List < User > {
            new User {
                Id = 1, FirstName = "mytest", LastName = "User", Username = "mytestuser", Password = "test123"
            }
```

```
55          public AuthenticationService(IOptions < AppSettings > appSettings) {
56              appSettings = appSettings.Value;
57          }
58          public AuthenticateResponse Authenticate(AuthenticateRequest model) {
59              var user = _users.SingleOrDefault(x => x.Username == model.UserName && x.Password == model.Passwo
60              // return null if user not found
61              if (user == null) return null;
62              // authentication successful so generate jwt token
63              var token = generateJwtToken(user);
64              // Returns User details and Jwt token in HttpResponse which will be user to authenticate the user
65              return new AuthenticateResponse(user, token);
66          }
67          //Generate Jwt Token
68          private string generateJwtToken(User user) {
69              var securityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_appSettings.Key));
70              var credentials = new SigningCredentials(securityKey, SecurityAlgorithms.HmacSha256);
71              // Here you  can fill claim information from database for the users as well
72              var claims = new [] {
73                  new Claim("id", user.Id.ToString()),
74                      new Claim(JwtRegisteredClaimNames.Sub, "atul"),
75                      new Claim(JwtRegisteredClaimNames.Email, ""),
76                      new Claim("Role", "Admin"),
77                      new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString())
78              };
79              var token = new JwtSecurityToken(_appSettings.Issuer, _appSettings.Issuer, claims, expires: DateT
80              return new JwtSecurityTokenHandler().WriteToken(token);
81          }
```

Now inside Helper folder create a JwtMiddleware class. This class will be used to validate the token and it will be registered as middleware.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.Options;
using Microsoft.IdentityModel.Tokens;
using JWTTokenPOC.Service;
using System.IdentityModel.Tokens.Jwt;
using System.Text;
namespace JWTTokenPOC.Helper {
    public class JwtMiddleware {
        private readonly RequestDelegate _next;
        private readonly AppSettings _appSettings;
        public JwtMiddleware(RequestDelegate next, IOptions < AppSettings > appSettings) {
            _next = next;
            _appSettings = appSettings.Value;
        }
        public async Task Invoke(HttpContext context, IUserService userService) {
            var token = context.Request.Headers["Authorization"].FirstOrDefault()?.Split(" ").Last();
            if (token != null)
                //Validate the token
```

```
        }
        private void attachUserToContext(HttpContext context, IUserService userService, string token) {
            try {
                var tokenHandler = new JwtSecurityTokenHandler();
                var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_appSettings.Key));
                tokenHandler.ValidateToken(token, new TokenValidationParameters {
                    ValidateIssuerSigningKey = true,
                        ValidateAudience = true,
                        ValidateLifetime = true,
                        IssuerSigningKey = key,
                        ValidIssuer = _appSettings.Issuer,
                        ValidAudience = _appSettings.Issuer,
                        // set clockskew to zero so tokens expire exactly at token expiration time.
                        ClockSkew = TimeSpan.Zero
                }, out SecurityToken validatedToken);
                var jwtToken = (JwtSecurityToken) validatedToken;
                var userId = int.Parse(jwtToken.Claims.First(x => x.Type == "id").Value);
                // attach user to context on successful jwt validation
                context.Items["User"] = userService.GetById(userId);
            } catch (Exception) {
                // do nothing if jwt validation fails
                // user is not attached to context so request won't have access to secure routes
            }
        }
    }
}
```

Now open the statrtup.cs file. In the ConfigureServices method, add CORS policy and add the services a

```
    services.AddControllers();
    // configure to get Appsetting section from appsetting.json
    services.Configure < AppSettings > (Configuration.GetSection("AppSettings"));
    services.AddScoped < IUserService, UserService > ();
    services.AddScoped < IAuthenticationService, AuthenticationService > ();
}
```

Step 10

In the Configure method, set CORS policy and register the JWT middleware as below.

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env) {
    app.UseRouting();
    // set global cors policy
    app.UseCors(x => x.AllowAnyOrigin().AllowAnyMethod().AllowAnyHeader());
    // Custom jwt auth middleware to authenticate the token
    app.UseMiddleware < JwtMiddleware > ();
    app.UseEndpoints(endpoints => {
        endpoints.MapControllers();
    });
}
```

Step 11

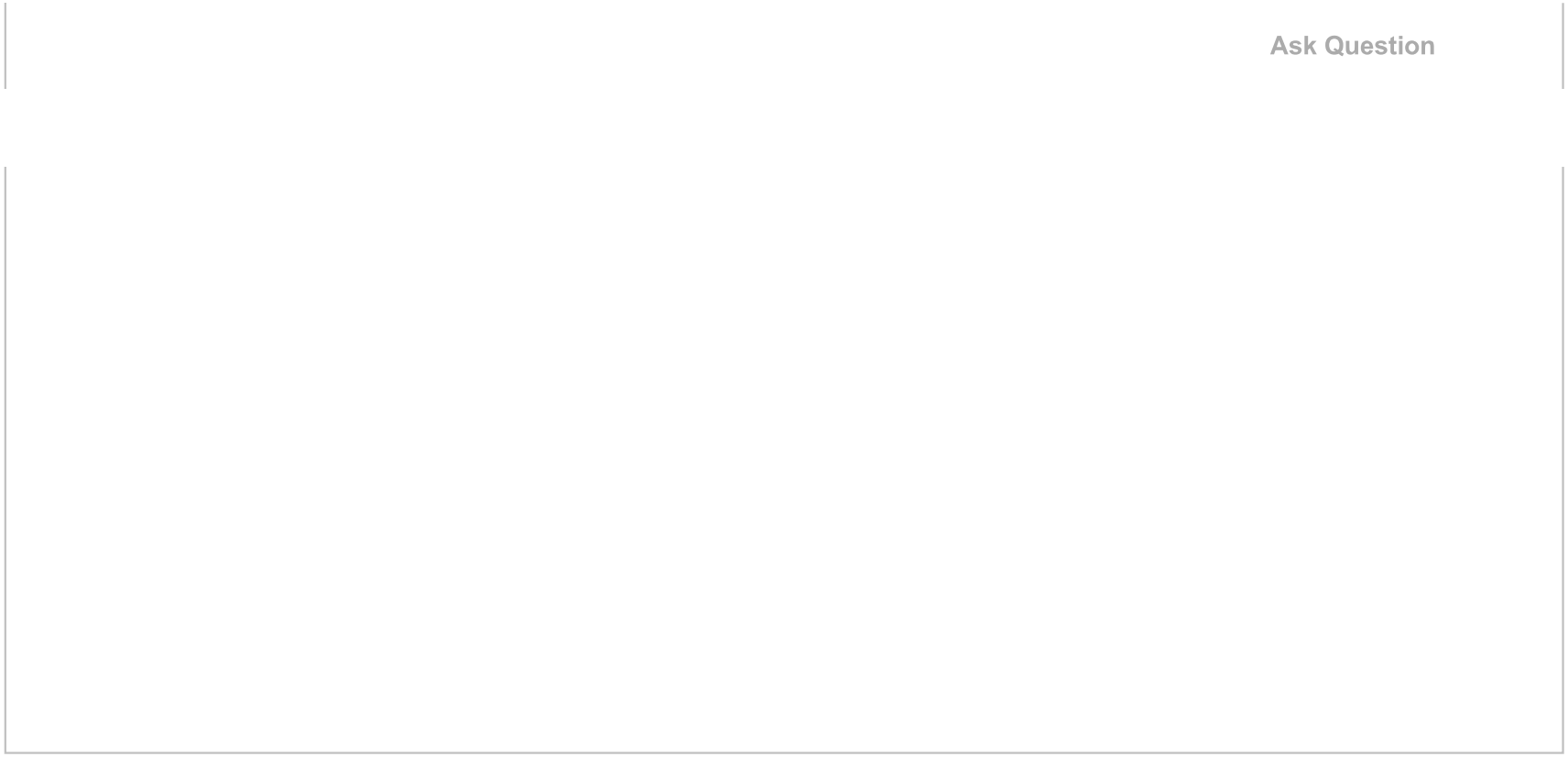
Now build and run the application.

Open Postman tool and generate the JWT token as below:

- Change the http request method to "GET" with the dropdown selector on the left of the URL input field.
- In the URL field enter the address to the route of your local API *http://localhost:60119/Authenticate/authenticate*.
- Select the "Body" tab below the URL field, change the body type radio button to "raw", and change the format dropdown **selector** to "JSON (application/json)".
- Enter a JSON object containing the test username and password in the "Body" text

```
{
    "username": "mytestuser",
    "password": "test123"
}
```

Click the "Send" button, you should receive a "200 OK" response with the user details including a JWT token in the response body, make a copy of the token value because we'll be using it in the next step to make an authenticated request.



Step 13

Now investigate the body section there is “token” attribute. This is the JWT token that you got, and this token will be used to validate the user and get User data from User controller

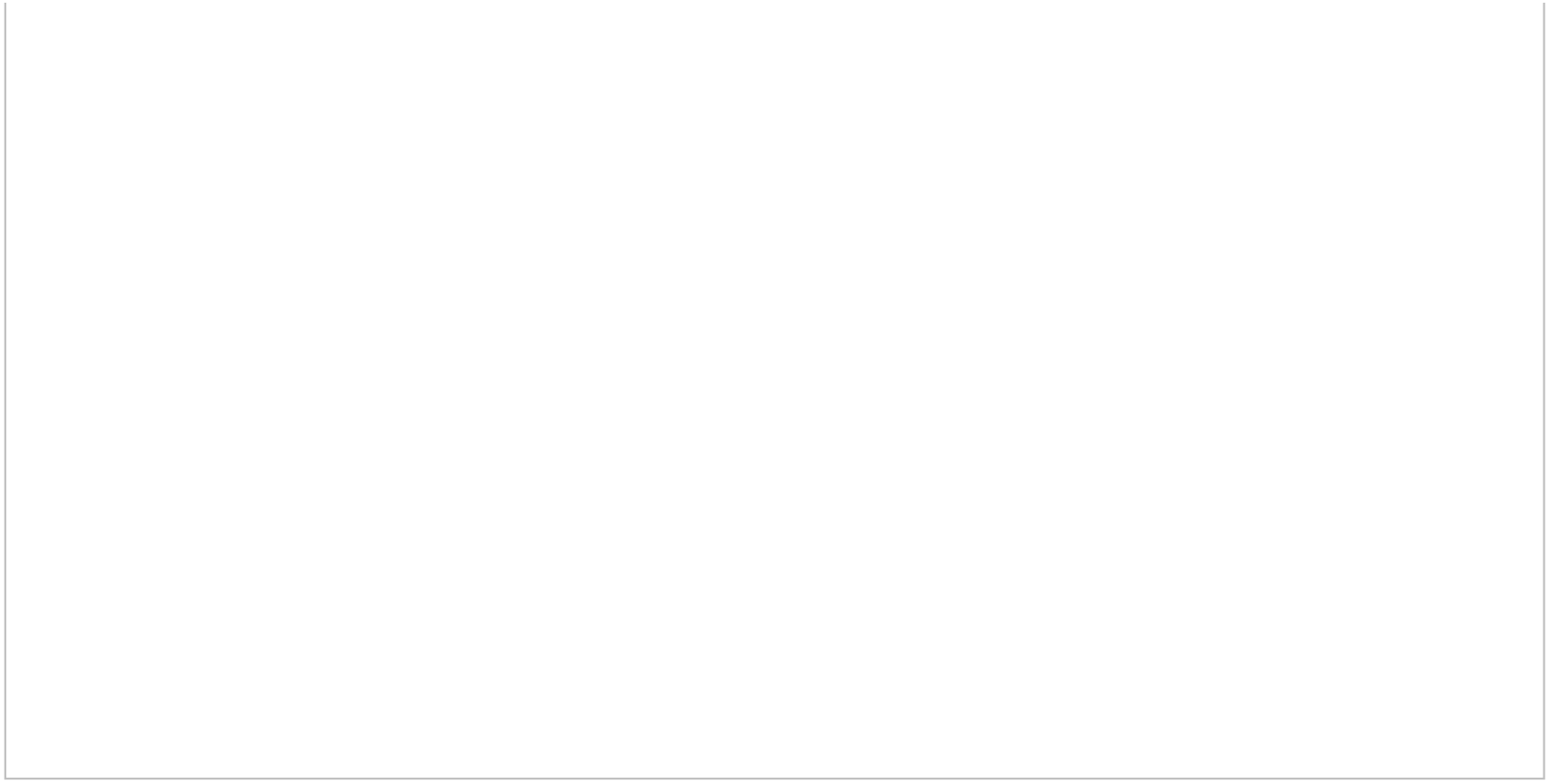
Step 14

To make an authenticated request using the JWT token from the previous step, follow these steps:

- Open a new request tab by clicking the plus (+) button at the end of the tabs.
- Change the http request method to "GET" with the dropdown selector on the left of the URL input field.

JWT token from the previous authenticate step into the "Token" field.

- Click the "Send" button, you should receive a "200 OK" response containing a JSON array with all the user records in the system.



ASP.NET Core Web API JWT Token

RECOMMENDED EBOOK

SIMILAR ARTICLES

JWT Authentication In ASP.NET Core

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

ASP.NET Core Web API - Creating And Validating JWT (JSON Web Token)

Authentication And Authorization In ASP.NET Core Web API With JSON Web Tokens

ASP.NET Web API 2 - Creating And Validating JWT (JSON Web Token)

Atul Sharma

1576 **26.7k**

View All Comments

6 **1**