

Security Lab

Perform these labs on your own computer using Visual Studio 2022 to ensure you understand the lessons presented in the corresponding videos and lectures.

Lab 1: Add Authentication

Open the **Program.cs** file and add the following line of code just before the call to the **builder.Services.ConfigureCors();** method.

```
// Add and Configure  
builder.Services.AddAuthentication();
```

Just before the call to the **app.UseAuthorization()** method add the code shown in **bold** below.

```
// Enable Authentication & Authorization Middleware  
app.UseAuthentication();  
app.UseAuthorization();
```

Open the **WeatherForecastController.cs** file and add the **[Authorize]** attribute to the **Get()** method.

```
[HttpGet(Name = "GetWeatherForecast")]  
[Authorize]  
public IEnumerable<WeatherForecast> Get()  
{  
    // REST OF THE CODE HERE  
}
```

Try it Out

Run the application and click on the **GET /WeatherForecast** button.

You should see a 500 Internal Server Error message.

Lab 2: Add JWT to Web API Project

Right mouse-click on the **AdvWorksAPI** project and select **Manage NuGet Packages...**

Click on the Browse tab.

Install the **System.IdentityModel.Tokens.Jwt** (select the latest package that targets .NET 6/7, should be around 6.27.0) package.

Install the **Microsoft.AspNetCore.Authentication.JwtBearer**

For .NET 6, select the version 6.0.14 package.

For .NET 7, select the version 7.0.3 package.

Add JWT Settings to AppSettings File

Open the **appsettings.Development.json** file and add the code shown in **bold** below:

```
{
  "AdvWorksAPI": {
    "InfoMessageDefault": "Problem Attempting to {Verb}
a Customer using the Customer API. Please Contact Your
System Administrator.",
    "DefaultTitle": "Ms.",
    "DefaultEmail": "FirstName.LastName@AdvWorks.com",
    "JwtSettings": {
      "key":
"This!Is&A*Long (Key) For#Creating (A) Symmetric*Key",
      "issuer": "http://localhost:5198",
      "audience": "AdvWorksAPI",
      "minutesToExpiration": "10"
    }
  },
}, /// REST OF THE JSON HERE
```

NOTE:	Change the PORT number in the settings to be the same as your port number on your Web API project.
--------------	--

Create a JWT Settings Class

Right mouse-click on the EntityLayer folder and add a new class named **JwtSettings**.

Replace the entire contents of this new file with the following code.

```
namespace AdvWorksAPI.EntityLayer;

public class JwtSettings
{
    public JwtSettings()
    {
        Key = "A_KEY";
        Issuer = "http://localhost:nnnn";
        Audience = "Audience";
        MinutesToExpiration = 30;
    }

    public string Key { get; set; }
    public string Issuer { get; set; }
    public string Audience { get; set; }
    public int MinutesToExpiration { get; set; }
}
```

Add JwtSettings Class to AdvWorksAPIDefaults Class

Open the **AdvWorksAPIDefaults.cs** file and add a new property.

```
public JwtSettings JWTSettings { get; set; }
```

Modify the constructor to initialize this new property.

```
public AdvWorksAPIDefaults()
{
    Created = DateTime.Now;
    InfoMessageDefault = string.Empty;
    CustomerCategoryID = 1;
    CustomerModelID = 2;
    JWTSettings = new();
}
```

Try it Out

Open the **CustomerController.cs** file and set a breakpoint on the closing brace of the constructor.

Run the application and click on the **GET /api/Customer** button.

Hover over the **_Settings** property and ensure all the settings are read in from the `appsettings.Development.json` file.

Lab 3: Register Authentication using JWT

Open the **ServiceExtension.cs** file and add a new method named `ConfigureJwtAuthentcation()`.

```
public static AuthenticationBuilder
ConfigureJwtAuthentication(this IServiceCollection
services, AdvWorksAPIDefaults settings)
{
    // Add Authentication to Services
    return services.AddAuthentication(options =>
    {
        options.DefaultAuthenticateScheme =
JwtBearerDefaults.AuthenticationScheme;
        options.DefaultChallengeScheme =
JwtBearerDefaults.AuthenticationScheme;
        options.DefaultScheme =
JwtBearerDefaults.AuthenticationScheme;
    }).AddJwtBearer(jwtOptions =>
    {
        jwtOptions.TokenValidationParameters =
            new TokenValidationParameters
            {
                ValidIssuer = settings.JWTSettings.Issuer,
                ValidAudience = settings.JWTSettings.Audience,
                IssuerSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(settings.JWT
Settings.Key)),
                ValidateIssuer = true,
                ValidateAudience = true,
                ValidateLifetime = true,
                ValidateIssuerSigningKey = true,
                ClockSkew =
TimeSpan.FromMinutes(settings.JWTSettings.MinutesToExpir
ation)
            };
    });
}
```

Add another method to configure the JWT Authorization. You will add more to this method later.

```
public static IServiceCollection
ConfigureJwtAuthorization(this IServiceCollection
services)
{
    return services.AddAuthorization();
}
```

Open the Program.cs file and locate the previous call you made to **builder.Services.AddAuthentication()** and delete that line of code.

Replace that line of code with the following lines of code.

```
// Add & Configure JWT Authentication
builder.Services.ConfigureJwtAuthentication(

builder.Configuration.GetRequiredSection("AdvWorksAPI").
Get<AdvWorksAPIDefaults>());

// Add & Configure JWT Authorization
builder.Services.ConfigureJwtAuthorization();
```

Try it Out

Run the application and click on the **GET /WeatherForecast** button.

You should still get a 500 status code, but now it just shows the path you were trying to get to.

Lab 4: Handle 401 Call

Open the **ErrorController.cs** file, locate the **StatusCodeHandler()** method and add a new case statement in the **switch()**.

```
case 401:
    msg = $"You are not authorized for this route:
'{msg}'";
    ret = StatusCode(StatusCode.Status401Unauthorized,
msg);
    break;
```

Try it Out

Run the application again and click on the **GET /WeatherForecast** button.

You should now get a 401 status code.

Lab 5: Create Security Classes

Right mouse-click on the Web API project and add a new folder named **SecurityLayer**.

Add User Class

Right mouse-click on the SecurityLayer folder and add a new class named **AppUser**.

```
using System.Text.Json.Serialization;

namespace AdvWorksAPI.SecurityLayer;

public partial class AppUser
{
    public AppUser()
    {
        UserId = Guid.NewGuid();
        UserName = string.Empty;
        Password = string.Empty;
        IsAuthenticated = false;
    }

    public Guid UserId { get; set; }
    public string UserName { get; set; }
    [JsonIgnore]
    public string Password { get; set; }
    public bool IsAuthenticated { get; set; }
}
```

Add User Claim Class

Right mouse-click on the SecurityLayer folder and add a new class named **AppUserClaim**.

```
namespace AdvWorksAPI.SecurityLayer;

public partial class AppUserClaim
{
    public AppUserClaim()
    {
        ClaimId = Guid.NewGuid();
        UserId = Guid.NewGuid();
        ClaimType = string.Empty;
        ClaimValue = string.Empty;
    }

    public Guid ClaimId { get; set; }
    public Guid UserId { get; set; }
    public string ClaimType { get; set; }
    public string ClaimValue { get; set; }
}
```

Add Security Token Class

Right mouse-click on the SecurityLayer folder and add a new class named **AppSecurityToken**.

```
namespace AdvWorksAPI.SecurityLayer;

public class AppSecurityToken
{
    public AppSecurityToken()
    {
        User = new() { UserName = "Not Authenticated" };
        BearerToken = string.Empty;
        Claims = new();
    }

    public AppUser User { get; set; }
    public string BearerToken { get; set; }
    public List<AppUserClaim> Claims { get; set; }
}
```


Lab 6: Create a Security Manager Class

Right mouse-click on the SecurityLayer folder and add a new class named **SecurityManager**.

```
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;
using AdvWorksAPI.EntityLayer;
using Microsoft.IdentityModel.Tokens;

namespace AdvWorksAPI.SecurityLayer;

public class SecurityManager
{
    #region AuthenticateUser Method
    public AppSecurityToken AuthenticateUser(string name,
string password, JwtSettings settings)
    {
        AppSecurityToken asToken;

        // Validate the user passed in
        // Create the AppSecurityToken object
        asToken = ValidateUser(name, password);

        if (asToken.User.IsAuthenticated) {
            // Load User Claims into Security Token
            LoadUserClaims(asToken);

            // Build Application Security Token
            SetJwtToken(settings, asToken);
        }

        return asToken;
    }
    #endregion

    #region ValidateUser Method
    protected AppSecurityToken ValidateUser(string name,
string password)
    {
        AppSecurityToken asToken = new();

        // Validate User - HARD CODED FOR NOW
        // TODO: Authenticate against a data store
        switch (name.ToLower()) {
            case "pauls":
                if (password == "password") {
                    asToken.User.UserName = name;
                    asToken.User.UserId = new Guid("4df9b2b3-e497-
407f-8b84-d0e638bdcddc");
                    asToken.User.IsAuthenticated = true;
                }
            }
        }
    }
}
```

```
        }
        break;

    case "johnk":
        if (password == "password") {
            asToken.User.UserName = name;
            asToken.User.UserId = new Guid("1a8418ff-550f-
4341-b6f8-1003085ce01b");
            asToken.User.IsAuthenticated = true;
        }
        break;
    }

    return asToken;
}
#endregion

#region LoadUserClaims
protected void LoadUserClaims(AppSecurityToken
asToken)
{
    // Get Claims for a user - HARD CODED FOR NOW
    // TODO: Get Claims from a Data Store
    switch (asToken.User.UserName.ToLower()) {
        case "pauls":
            // Add GetCustomers
            asToken.Claims.Add(new AppUserClaim()
            {
                UserId = asToken.User.UserId,
                ClaimType = "GetCustomers",
                ClaimValue = "true"
            });
            // Add GetACustomer
            asToken.Claims.Add(new AppUserClaim()
            {
                UserId = asToken.User.UserId,
                ClaimType = "GetACustomer",
                ClaimValue = "true"
            });
            // Add Search
            asToken.Claims.Add(new AppUserClaim()
            {
                UserId = asToken.User.UserId,
                ClaimType = "Search",
                ClaimValue = "true"
            });
            break;
    }
}
```

```
case "johnk":
    // Add GetACustomer
    asToken.Claims.Add(new AppUserClaim()
    {
        UserId = asToken.User.UserId,
        ClaimType = "GetACustomer",
        ClaimValue = "true"
    });
    // Add AddCustomer
    asToken.Claims.Add(new AppUserClaim()
    {
        UserId = asToken.User.UserId,
        ClaimType = "AddCustomer",
        ClaimValue = "true"
    });
    // Add UpdateCustomer
    asToken.Claims.Add(new AppUserClaim()
    {
        UserId = asToken.User.UserId,
        ClaimType = "UpdateCustomer",
        ClaimValue = "true"
    });
    break;
}
}
#endregion

#region SetJwtToken
protected void SetJwtToken(JwtSettings settings,
AppSecurityToken asToken)
{
    // Build JWT claims
    List<Claim> claims = BuildJWTClaims(asToken);

    SecurityTokenDescriptor tokenDescriptor = new()
    {
        Expires =
DateTime.UtcNow.AddMinutes(settings.MinutesToExpiration)
,
        Issuer = settings.Issuer,
        Audience = settings.Audience,
        SigningCredentials = new SigningCredentials
            (new
SymmetricSecurityKey(Encoding.ASCII.GetBytes(settings.Key
y))),
        SecurityAlgorithms.HmacSha512Signature),
```

```
// Add Claims
Subject = new ClaimsIdentity(claims)
};

var tokenHandler = new JwtSecurityTokenHandler();
var bearerToken =
tokenHandler.WriteToken(tokenHandler.CreateToken(tokenDe
scriptor));

// Create a string representation of the Jwt token
// Stored into BearerToken property
asToken.BearerToken = bearerToken;
}
#endregion

#region BuildJWTClaims Method
protected List<Claim> BuildJWTClaims(AppSecurityToken
asToken)
{
    // Create standard JWT claims
    List<Claim> ret = new()
    {
        // Add Unique User Name
        new Claim(JwtRegisteredClaimNames.Sub,
asToken.User.UserName),
        // Add Unique JWT Token Identifier
        new Claim(JwtRegisteredClaimNames.Jti,
Guid.NewGuid().ToString()),
        // Add IsAuthenticated Claim
        new Claim("IsAuthenticated",
asToken.User.IsAuthenticated.ToString())
    };

    // Add Custom Claims for your Application
    foreach (var item in asToken.Claims) {
        ret.Add(new Claim(item.ClaimType,
item.ClaimValue));
    }

    return ret;
}
#endregion
}
```

Lab 7: Add Security Controller to Generate JWT Token

Right mouse-click on the Controllers folder and add a new class named **SecurityTestController**.

Replace the entire contents of this new file with the following code.

```

using AdvWorksAPI.BaseClasses;
using AdvWorksAPI.EntityLayer;
using AdvWorksAPI.SecurityLayer;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Options;

namespace AdvWorksAPI.Controllers;

[Route("api/[controller]")]
[ApiController]
public class SecurityTestController : ControllerBaseAPI
{
    private readonly AdvWorksAPIDefaults _Settings;

    public
    SecurityTestController(IOptionsMonitor<AdvWorksAPIDefaults> defaults, ILogger<SecurityTestController> logger) :
    base(logger)
    {
        _Settings = defaults.CurrentValue;
    }

    [HttpGet()]
    [Route("AuthenticateUser/{name}/password/{password}")]
    [ProducesResponseType(StatusCodes.Status200OK)]

    [ProducesResponseType(StatusCodes.Status401Unauthorized)]
    public ActionResult<AppSecurityToken>
    AuthenticateUser(string name, string password)
    {
        ActionResult<AppSecurityToken> ret;
        AppSecurityToken asToken;

        asToken = new
        SecurityManager().AuthenticateUser(name, password,
        _Settings.JWTSettings);

        if (asToken.User.IsAuthenticated) {
            ret = StatusCode(StatusCodes.Status200OK,
asToken);
        }
        else {
            ret =
            StatusCode(StatusCodes.Status401Unauthorized, "Invalid
            User Name/Password.");
        }
    }
}

```

```
        return ret;
    }
}
```

Try it Out

Run the application and click on the **GET** `/api/SecurityTest/AuthenticateUser/{name}/password/{password}` button.

Enter **johnk** into the **name** field.

Enter **password** into the **password** field.

Click on the **Execute** button.

You should see something that looks like the following:

Code	Details
200	<p>Response body</p> <pre>{ "User": { "UserId": "1a8418ff-550f-4341-b6f8-1003085ce01b", "UserName": "johnk", "IsAuthenticated": true }, "BearerToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJqb2h1dWUiLCJlb2ZGRQcm9kdWN0IjoidHJ1ZSIsIlVwZGF0ZVB2R1Y3QiOiJ0cnV1IiwibmJmIjE00cP6bW8AsU2De1AwKc15HdxrMYw", "Claims": [{ "ClaimId": "80c11f23-85df-48e5-a31b-9937c8761d4d", "UserId": "1a8418ff-550f-4341-b6f8-1003085ce01b", "ClaimType": "GetAPProduct", "ClaimValue": "true" }, { "ClaimId": "c43bcd33-7c07-49f6-85d1-0ac968b5bbcd", "UserId": "1a8418ff-550f-4341-b6f8-1003085ce01b", "ClaimType": "AddProduct", "ClaimValue": "true" }, { "ClaimId": "2262214d-f72e-4d9c-a216-f37bbee1bb18", "UserId": "1a8418ff-550f-4341-b6f8-1003085ce01b", "ClaimType": "UpdateProduct", "ClaimValue": "true" }] }</pre>

Now try entering a bad name such as "asdf" with a bad password.

You should see a **401 – Not Authorized** status code.

Lab 8: Add JWT Token in Swagger

You need to add some options to the Swagger generation to be able to enter a bearer token.

Open the **ServiceExtension.cs** file and add a new method.

```
public static IServiceCollection ConfigureOpenAPI(this
IServiceCollection services)
{
    // Configure Open API (Swagger)
    // More Info: https://aka.ms/aspnetcore/swashbuckle
    services.AddEndpointsApiExplorer();
    return services.AddSwaggerGen(options =>
    {
        options.AddSecurityDefinition("Bearer", new
OpenApiSecurityScheme
        {
            Scheme = "Bearer",
            BearerFormat = "JWT",
            In = ParameterLocation.Header,
            Name = "Authorization",
            Description = "Bearer Authentication with JWT
Token",
            Type = SecuritySchemeType.Http
        });
        options.AddSecurityRequirement(new
OpenApiSecurityRequirement
        {
            {
                new OpenApiSecurityScheme
                {
                    Reference = new OpenApiReference
                    {
                        Id = "Bearer",
                        Type = ReferenceType.SecurityScheme
                    }
                },
                new List<string>()
            }
        });
    });
}
```

Open the **Program.cs** file and locate the following lines

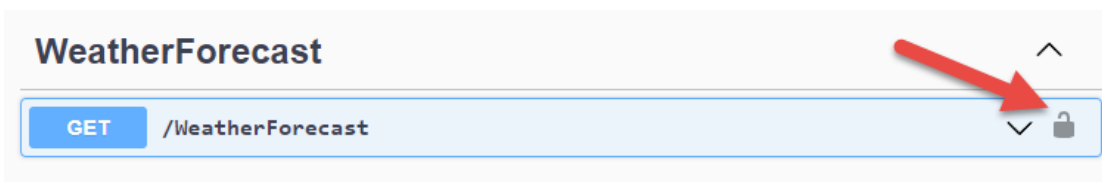
```
// Configure Open API (Swagger)
// More Info: https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();
```

Change the above lines of code to the following:

```
// Add & Configure Open API (Swagger)
builder.Services.ConfigureOpenAPI();
```

Try it Out

Run the application and you should now see a lock icon on each API call.



Click on the **GET /api/SecurityTest/AuthenticateUser** button and enter the following inputs.

```
Name: johnk
Password: password
```

Copy just the bearer token to the clipboard.

Click on the **lock icon** next to the **GET /api/WeatherForecast**.

Copy the bearer token into the input field and click the **Authorize** button.

Click the Close button.

Click the **GET /api/WeatherForecast** button and the call should now work.

Lab 9: Add Claims to Token and Secure an Endpoint

Open the **ServiceExtension.cs** file and locate the `ConfigureJwtAuthorization()` method and replace the return statement with the following code in **bold**.

```
public static IServiceCollection
ConfigureJwtAuthorization(this IServiceCollection
services)
{
    return services.AddAuthorization(options =>
    {
        options.AddPolicy("GetCustomers", policy =>
policy.RequireClaim("GetCustomers"));
        options.AddPolicy("GetACustomer", policy =>
policy.RequireClaim("GetACustomer"));
        options.AddPolicy("Search", policy =>
policy.RequireClaim("Search"));
        options.AddPolicy("AddCustomer", policy =>
policy.RequireClaim("AddCustomer"));
        options.AddPolicy("UpdateCustomer", policy =>
policy.RequireClaim("UpdateCustomer"));
    });
}
```

Open the **CustomerController.cs** file and locate the **Get()** method
Add an **[Authorize]** attribute that looks like the following:

```
[Authorize(Policy = "GetCustomersPolicy")]
```

Try it Out

Click on the **GET /api/SecurityTest/AuthenticateUser** button and enter the following inputs.

```
Name: johnk
Password: password
```

Copy just the bearer token to the clipboard.

Click on the **lock icon** next to the **GET /api/Customer**.

Copy the bearer token into the input field and click the **Authorize** button.

Click the **Close** button.

Click the **GET /api/Customer** button and you should get a 500 Error.

When using Claims, if you are not authorized to make that call a 403 – Forbidden status code is returned.

Open the **ErrorController.cs** file and add a new switch statement to handle the 403 status code.

```
case 403:
    msg = $"You are forbidden from accessing this route:
    '{msg}'";
    ret = StatusCode(StatusCode.Status403Forbidden, msg);
    break;
```

Try the above steps again using johnk's bearer token. You should now receive a 403 status code and the error message you just added to the switch statement.

Enter a Valid User

Click on the lock icon next to the **GET /api/Customer**

Click the Logout button.

Click on the **GET /api/SecurityTest/AuthenticateUser** button and enter the following inputs.

```
Name: pauls
Password: password
```

Copy just the bearer token to the clipboard.

Click on the lock icon next to the **GET /api/Customer**.

Copy the bearer token into the input field and click the **Authorize** button

Click the **Close** button.

Click the **GET /api/Customer** button and you should now get a list of customers.