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 just below the call to **builder.Services.ConfigureCors()** add the following lines of code.

```
// Add & Configure Authentication
builder.Services.AddAuthentication();
// Add & Configure Authorization
builder.Services.AddAuthorization();
```

Go further down and just after the call to **UseCors**() add the following lines of code

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

Open the **CustomerRouter.cs** file and modify the app.MapGet() that calls the Get() method.

```
app.MapGet($"/{UrlFragment}", () => Get())
    .WithTags(TagName)
    .Produces(200)
    .Produces<List<Customer>>()
    .Produces(404)
    .Produces(500)
    .RequireAuthorization();
```

Try it Out

Run the application and click on the **GET /api/Customer** button and you should get a 500 error.

The error should read something like "No authenticationScheme was specified…"

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

Add JWT Package

Install the **System.IdentityModel.Tokens.Jwt (Version 6.27.0)** package.

Add Bearer Token Package

Install the **Microsoft.AspNetCore.Authentication.JwtBearer (Version 6.0.14)** package.

Add JWT Settings to AppSettings File

Open the **appsettings.Development.json** file and add the following:

```
"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:nnnn",
    "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 the file with the following code.

```
namespace AdvWorksAPI.EntityLayer;

public class JwtSettings
{
  public JwtSettings()
  {
    Key = "A_KEY_GOES_HERE";
    Issuer = "http://localhost:nnnn";
    Audience = "Audience";
    MinutesToExpiration = 1;
  }

  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** class 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;
  DefaultTitle = "Mr.";
  DefaultEmail = "LastName.FirstName@adventure-
works.com";
  JWTSettings = new();
}
```

Try it Out

Open the **CustomerRouter.cs** file and set a breakpoint on the last line of the constructor just after the setting of the **_Settings** field.

Run the application.

Hover over the **_Settings** field and ensure that the values for the JWTSettings property are set from the appsettings.Development.json file and not the hard-coded values from the constructor of the JWTSettings class.

NOTE:	Make sure the **PORT** number in the settings files is the same as the port number on your Web API project.

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.

We are going to add more to this method later.

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

Locate the previous calls you made to builder.Services.AddAuthentication() and builder.Services.AddAuthorization() and replace those lines with the following:

```
// 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 **/api/Customer** and you should now see a 401 Unauthorized status code is returned.

Click on some other API calls **other** than the **/api/Customer** to ensure everything else works.

Lab 4: 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 5: 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-d0e638bdcdcc");
          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.Ke
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 6: Add Security Router to Generate JWT Token

Right mouse-click on the \RouterClasses folder and add a new class named **SecurityTestRouter**. Replace the entire contents of the file with the following code.

```
using AdvWorksAPI.BaseClasses;
using AdvWorksAPI.EntityLayer;
using AdvWorksAPI.SecurityLayer;

namespace AdvWorksAPI.RouterClasses;

public class SecurityTestRouter : RouterBase
{
  private readonly AdvWorksAPIDefaults _Settings;

  public SecurityTestRouter(ILogger<SecurityTestRouter>
logger, AdvWorksAPIDefaults settings) : base(logger)
  {
    UrlFragment = "api/SecurityTest";
    TagName = "SecurityTest";
    _Settings = settings;
  }

  /// <summary>
  /// Add routes
  /// </summary>
  /// <param name="app">A WebApplication object</param>
  public override void AddRoutes(WebApplication app)
  {

app.MapGet($"/{UrlFragment}/AuthenticateUser/{{name}}/pa
ssword/{{password}}", (string name, string password) =>
      AuthenticateUser(name, password))
      .WithTags(TagName)
      .Produces(200)
      .Produces<AppSecurityToken>()
      .Produces(400);
  }

  protected virtual IResult AuthenticateUser(string
name, string password)
  {
    IResult ret;
    AppSecurityToken asToken;

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

    if (asToken.User.IsAuthenticated) {
      ret = Results.Ok(asToken);
    }
    else {
      ret = Results.BadRequest("Invalid User
Name/Password.");
    }

    return ret;
  }
}
```

Open the **ServiceExtension.cs** file and in the AddRouterClasses() method inject the SecurityTestRouter class

```
services.AddScoped<RouterBase, SecurityTestRouter>();
```

Try it Out

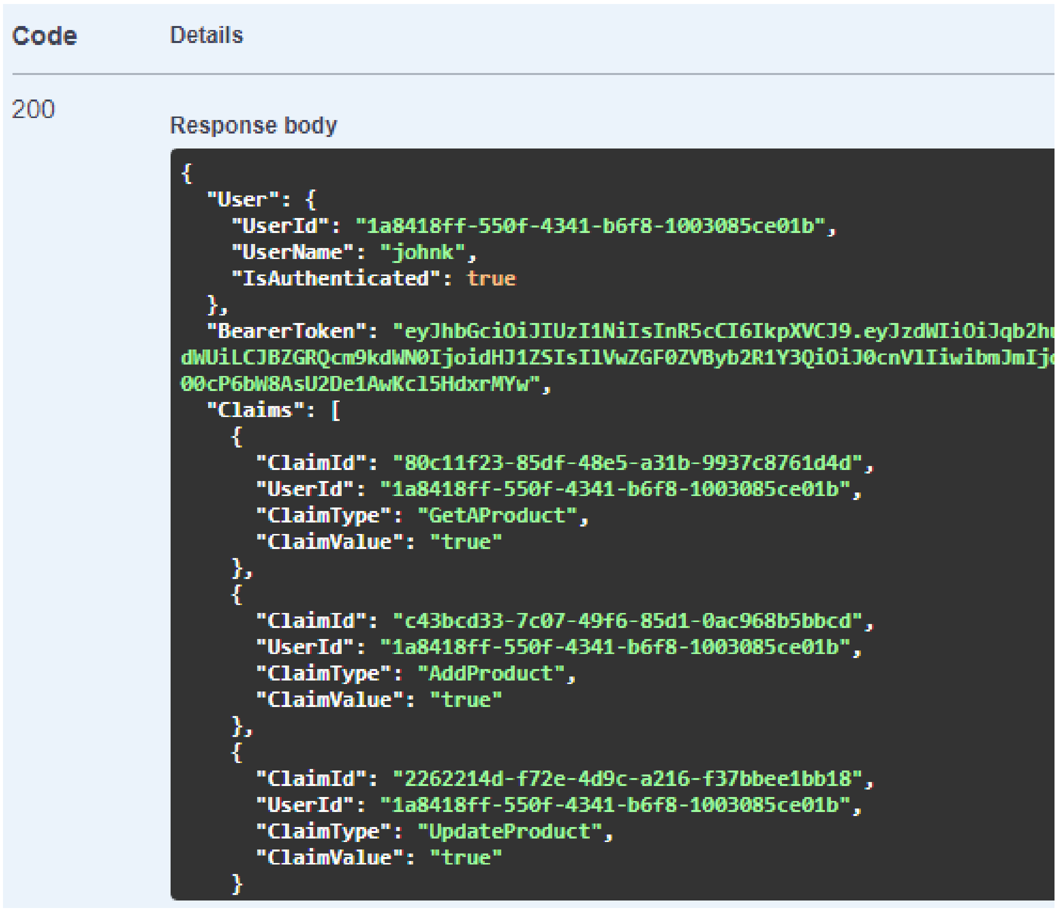
Run the application and click on the **GET /api/SecurityTest/AuthenticateUser** button.

Enter "johnk" for the **name** field.

Enter "password" for the **password** field.

Click the **Execute** button.

You should see something that looks like the following:



Code Details

200

Response body

```
{
  "User": {
    "UserId": "1a8418ff-550f-4341-b6f8-1003085ce01b",
    "UserName": "johnk",
    "IsAuthenticated": true
  },
  "BearerToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJqb2h
dWUiLCJBZGRQcm9kdWN0IjoidHJ1ZSIsIlVwZGF0ZVByb2R1Y3QiOiJ0cnVlIiwibmJmIj
00cP6bW8AsU2De1AwKcl5HdxrMYw",
  "Claims": [
    {
      "ClaimId": "80c11f23-85df-48e5-a31b-9937c8761d4d",
      "UserId": "1a8418ff-550f-4341-b6f8-1003085ce01b",
      "ClaimType": "GetAProduct",
      "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"
    }
```

Try Invalid User

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

You should see a **400 – Bad Request** status code with a message **Invalid User Name/Password**.

Lab 7: Display Contents of JWT Token

To see what makes up the *BearerToken* property, you can decode the value at the www.jwt.io website.

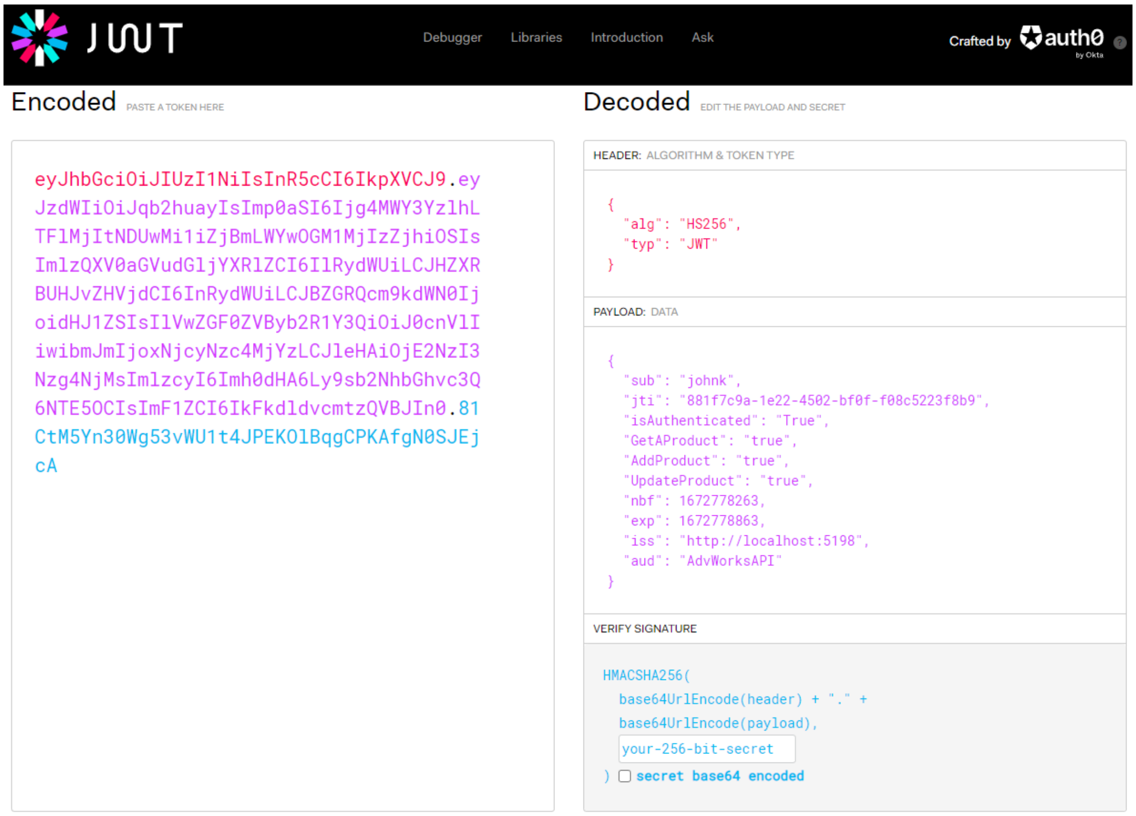
Try it Out

Copy the BearerToken value to the clipboard.

Open a browser window and go to www.jwt.io.

Paste your token into the box labeled "Encoded"

You should immediately see the payload data with all your data as shown below.


JWT
Debugger
Libraries
Introduction
Ask
Crafted by auth0 by Okta
Encoded PASTE A TOKEN HERE
Decoded EDIT THE PAYLOAD AND SECRET
HEADER: ALGORITHM & TOKEN TYPE
PAYLOAD: DATA
VERIFY SIGNATURE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiJqb2huayIsImp0aSI6Ijg4MWY3YzlhL
TFlMjItNDUwMi1iZjBmLWYwOGM1MjIzZjhiOSIs
ImlzQXV0aGVudGljYXRlZCI6IlRydWUiLCJHZXR
BUHJvZHVjdCI6InRydWUiLCJBZGRQcm9kdWN0Ij
oidHJ1ZSIsIlVwZGF0ZVByb2R1Y3QiOiJ0cnVlI
iwibmJmIjoxNjcyNzc4MjYzLCJleHAiOjE2NzI3
Nzg4NjMsImlzcyI6Imh0dHA6Ly9sb2NhbGhvc3Q
6NTE5OCIsImF1ZCI6IkFkdldvcmtzQVBJIn0.81
CtM5Yn30Wg53vWU1t4JPEKOlBqgCPKAfgN0SJEj
cA
```

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

```
{
  "sub": "johnk",
  "jti": "881f7c9a-1e22-4502-bf0f-f08c5223f8b9",
  "isAuthenticated": "True",
  "GetAProduct": "true",
  "AddProduct": "true",
  "UpdateProduct": "true",
  "nbf": 1672778263,
  "exp": 1672778863,
  "iss": "http://localhost:5198",
  "aud": "AdvWorksAPI"
}
```

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

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

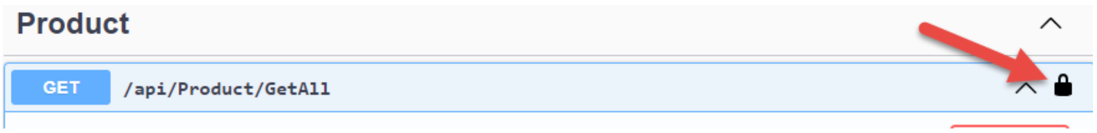
```
// Configure Open API (Swagger)
// More Info: https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();
```

Change these lines of code to be 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/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 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 **change** it to look like the following:

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

Open the **CustomerRouter.cs** file and locate the app.MapGet() method that calls the Get() method. Within the RequireAuthorization() method you previously added, insert the string "GetCustomers".

```
.RequireAuthorization("GetCustomersClaim");
```

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

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 **403 – Forbidden** error because johnk does not have the **GetCustomersClaim**.

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.

Demo 10: Add 401/403 Status Code

Open the **ErrorRouter.cs** file

```
protected virtual IResult StatusCode(int code,
HttpContext context)
{
  string msg = string.Empty;

  // Get some path information
  var feature =
context.Features.Get<IStatusCodeReExecuteFeature>();
  if (feature != null) {
    msg = feature.OriginalPathBase
        + feature.OriginalPath
        + feature.OriginalQueryString;
  }

  switch (code) {
    case 401:
      msg = $"You are not authorized for this route:
'{msg}'";
      break;
    case 403:
      msg = $"You are forbidden from accessing this
route: '{msg}'";
      break;
    case 404:
      msg = $"API Route Was Not Found: '{msg}'";
      break;
    default:
      msg = $"Status Code Not Handled: '{code}'";
      break;
  }

  return Results.Problem(msg, statusCode: code);
}
```

Change the StatusCode() method to return different Results based on the code