Filters

* Filters inject logic into Request processing.
* Allows us to run certain actions before of after specific stages in the request pipeline.

Need

1. Avoid Duplication of code
2. Logging
3. Caching
4. Error Handling

**How its different from the middlewares**

* Execution of middleware occurs before the MVC context becomes available in the pipeline.
* Middleware doesn’t have access to the ActionExecutedContext Rather he has access to HttpContext that allows you to perform actions on the request and response.
* As model binding has not occurred yet , using middleware would not be suited to run the validation function or modifying the values.
* Middleware also run on every request regardless of which controller or action is called.

Filter Types

1. Authorization Filters

* Runs first and Determines authorization

1. Resource Filters

* Use to implement caching
* Can be used to short circuit the pipeline for performance reasons.

1. Action Filters

* Filter is used to modify the request before its received by an action method or to modify action result after it has been produced
* Can be applied only to controllers and actions.
* Altering the request or response for an action method.

1. Exception Filters

* Use to globally handle all unhandled exceptions of individual results.

1. Result Filters

* Used to alter the action result before its executed or to modify the result after execution.

Need

* Inspecting Or Altering the Result produced by an endpoint

Filters

* It can be added by type or by instance
* If an instance is added , that instance is used for every request.
* If a type is added it’s a type activated

Type activate means

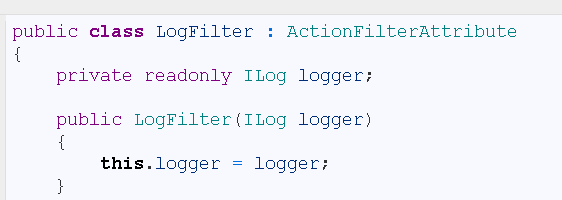
1. Ann instance is created for each request.
2. Any ctor dependencies are populated by DI.

Filters that are implemented as attributes and directly to controller classes or action methods cant have ctor dependencies provided by DI.

1. Attributes must have their constructor parameters supplied where they are applied
2. This is limitation of how it works

Following filters support constructor dependencies provided from DI

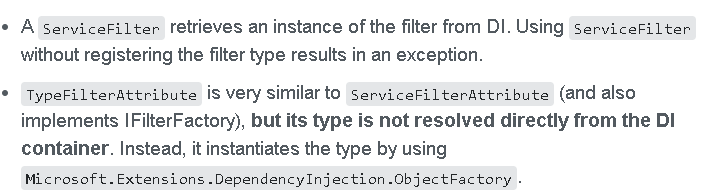
1. ServiceFilterAttribute : it takes DI



1. TypeFilterAttribute.

TypeFilterAttribute:

* No need to register with the DI container
* They can accept constructor arguments for the type.





Filter Types

1. Authorization Filters

* Runs first and use to determine current user is authorized or not.

1. Resource Filter

* Executes after Authorize
* Use to implement caching
* Can be used for performance reasons.

Real World Scenario : Response Caching

1. Action Filter

* Action filters are executed after the model binding process.
* Action Filters are used when model binding is required , which means they are used for tasks such as altering the model or enforcing validation.
* Can be applied to controllers and actions methods.
* Action Filters are provided with context data through 2 different context classes.
* Run code before or after an individual action method is called .
* Use to modify view data that a controller action returns

Real World Scenarion : When I am uploading file , I want to know the current status.

Scope

1. Global : Register filter globally
2. Action
3. Controller

Order of invocation

1. OnActionExecuting From the Global Filter
2. OnActionExecuting from the Controller filter
3. OnActionExecuting From the Action Filter
4. Action Method Execution
5. OnActionExecuted From the action filter
6. OnActionExecuted From the Global Filter
7. Exception filters : used to handle all unhandled exceptions that occur in the application.
8. Result filters :

* Executed before and after Action result is used to generate response.
* Can run code immediately after or before the execution of individual action results
* They can add behavior to view or formatter execution.
* These filters contain logic that is executed before and after a view result is executed. Like if you want to modify a view result right before the view is rendered to the browser.
* Example : You might want to modify view result right before the view is rendered to the browser.

Result filters are only executed when an action or action filter produces an action result. Result filters are not executed when:

* An authorization filter or resource filter short-circuits the pipeline.
* An exception filter handles an exception by producing an action result.

Demo Action Filter

1. Add Conditions to handle errors

public async Task<IActionResult> SaveProduct([FromBody] ProductBindingTarget target)

{

if (target == null)

{

return BadRequest("object is null");

}

if (!ModelState.IsValid)

{

return BadRequest("Model State is Invalid");

}

public async Task<IActionResult> UpdateProduct([FromBody] Product product)

{

if (product == null)

{

return BadRequest("object is null");

}

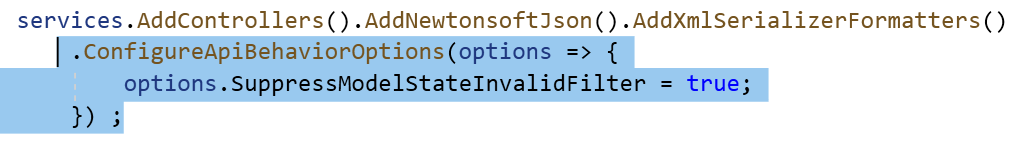
if (ModelState.IsValid)

{

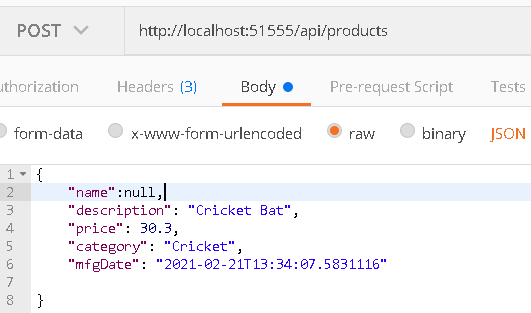
return BadRequest("ModelState is Invalid");

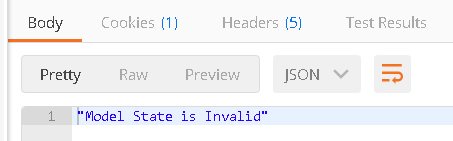
}

1. Disable Default Errors



1. Check the Output





Problem : Logic is Repeated in SaveProduct and UpdateProduct. One more thing, we want such model validation to happen before the action method gets execute. So we can create our own action filter in this case.

1. Add New CustomFilter

public class CustomActionFilter:ActionFilterAttribute

{

public override void OnActionExecuting(ActionExecutingContext context)

{

var param = context.ActionArguments.SingleOrDefault();

if (param.Value == null)

{

context.Result = new BadRequestObjectResult("Model is Null");

}

if (!context.ModelState.IsValid)

{

context.Result = new BadRequestObjectResult("Model is invalid");

}

}

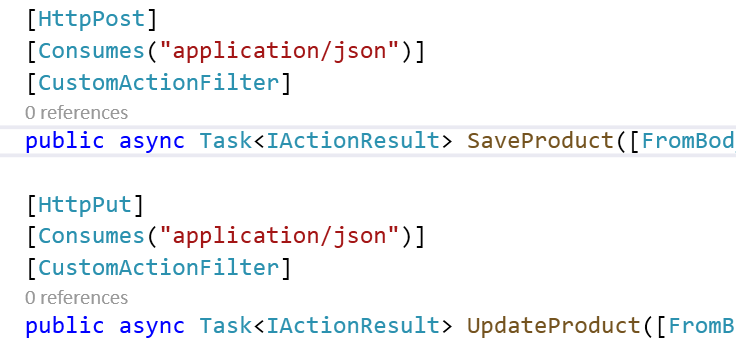
public override void OnActionExecuted(ActionExecutedContext context)

{

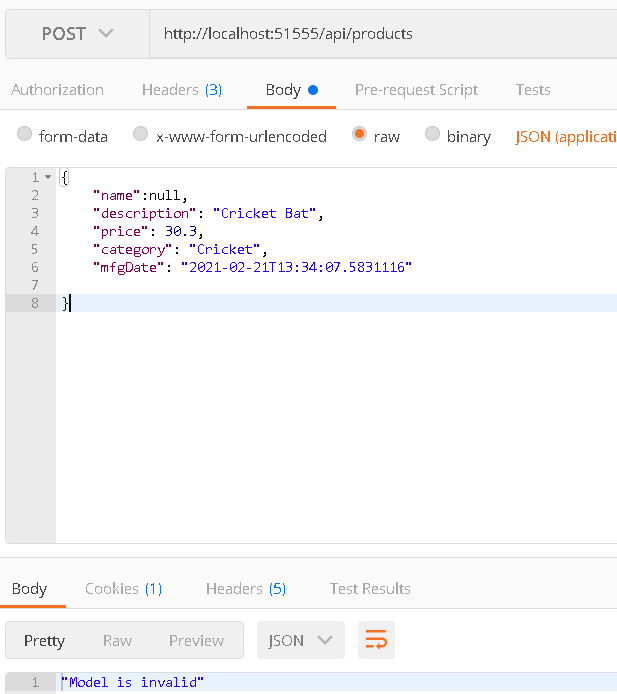
base.OnActionExecuted(context);

}

1. Apply those filter on Action Methods



1. Check the Output



Result Filter

* We can use Result filters to run code before or after the execution of controller action results.

How its different from OnActionExecuted and OnResultExecuted

* OnActionExecuted runs after the action method and can perform additional processing such as providing extra data to the action method, inspecting the return value or cancelling the execution of action method.
* OnResultExecuted runs after the result and can perform additional processing of the result such as Modifying of Http Response.
* OnResultExecuting after the business logic runs ad before the display logic runs.

Demo :

1. Add this Filter

public class AddHeaderAttribute : ResultFilterAttribute

{

private readonly string \_name;

private readonly string \_value;

public AddHeaderAttribute(string name, string value)

{

\_name = name;

\_value = value;

}

public override void OnResultExecuting(ResultExecutingContext context)

{

context.HttpContext.Response.Headers.Add(\_name, new string[] { \_value });

base.OnResultExecuting(context);

}

}

1. Apply this Filter on controller

[AddHeader("Author", "Swapnil")]

public class ProductController : Controller

{

private readonly IProductRepository \_repository;

1. Check the Output From Developer Tools
2. Go to dev tools
3. Network Tab
4. Click on the file that is loaded



ASP.Net Core Identity

* Membership System for web application that includes membership.
* API that supports UI login functionality
* Manage Users, Passwords, Profile data , roles , claims , tokens etc
* Users can create an account with login information stored in Identity.
* Identity is configured using SQL Server database to store user name and password and profile data , alternatively you can use AzureTableStorage
* Can use external login provider like FB,Google, MS and Twitter.

To Secure Application you can use

1. Azure Active Directory
2. Identity Server4 : OpenID Connect and OAuth2.0 Framework for ASP.Net Core

It enables

1. Authentication as a service
2. Single Sign on
3. Access Control for APIS

FAQ

1. Can we use Oracle for Storing Identity

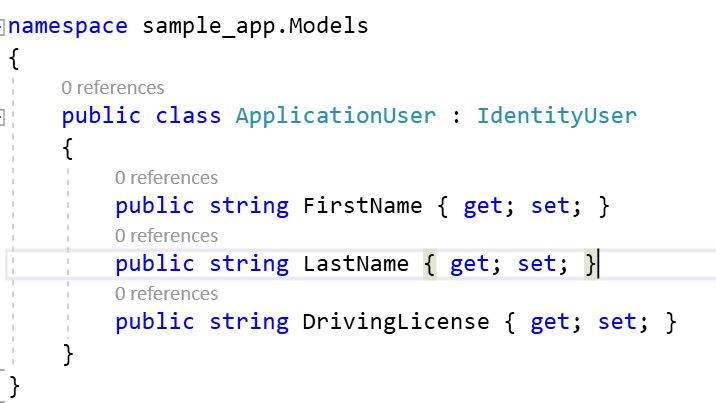
* [Using Entity Framework Core Implementation of ASP.NET Core Identity for Oracle (devart.com)](https://www.devart.com/dotconnect/oracle/docs/Identity-3-Tutorial.html#:~:text=In%20order%20to%20create%20an%20ASP.NET%20Core%20application,Web%20Application%2C%20click%20Next%2C%20and%20then%20click%20Create.)

Demo :

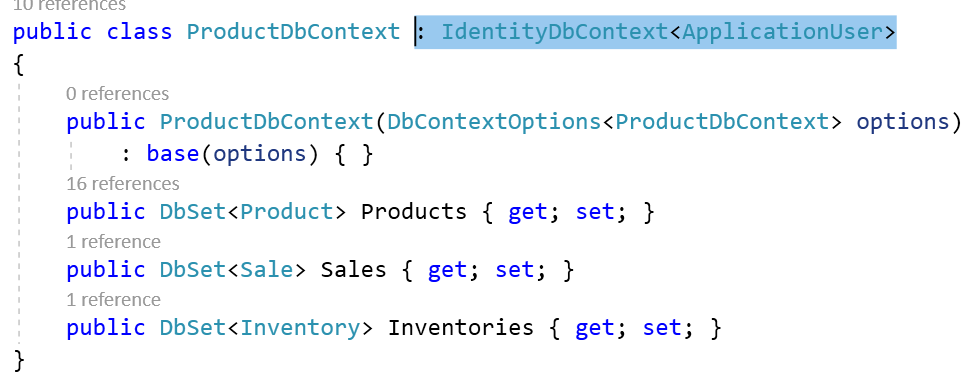
1. Add Package



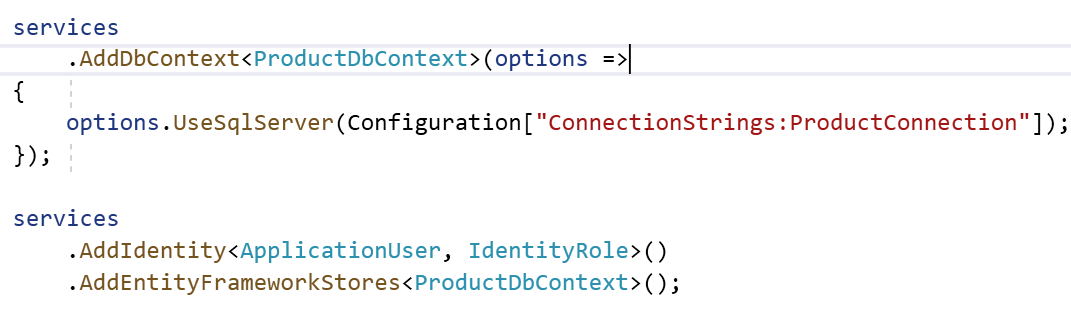
1. Add Class ApplicationUser in the model folder



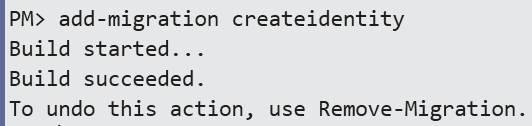
1. Inherit Context class from IdentityContext class



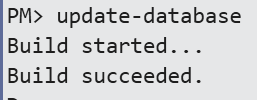
1. Register Core Identity : We have 2 extension methods
2. AddIdentity<TUser>
3. AddIdentity<TUser,TRole>: Support both users and roles both.



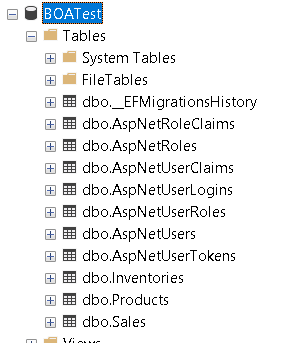
1. Add Identity Migrations



1. Update Database(Delete Database if you are facing an error)



1. This will generate following tables



1. Add class RoleConfiguration in model class

public class RoleConfiguration : IEntityTypeConfiguration<IdentityRole>

{

public void Configure(EntityTypeBuilder<IdentityRole> builder)

{

builder.HasData(

new IdentityRole

{

Name = "User",

NormalizedName = "User"

},

new IdentityRole

{

Name = "Administrator",

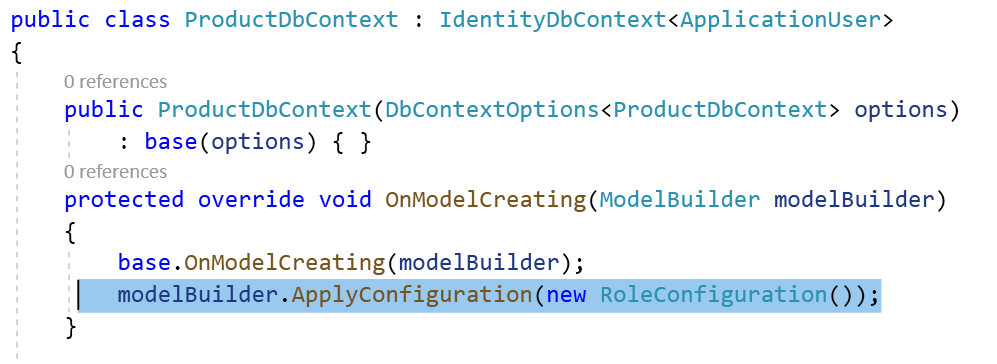
NormalizedName = "Administrator"

});

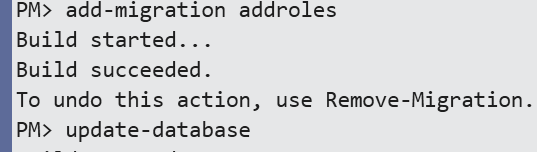
}

}

1. Configure Context class to Add Default Roles.



1. Add migration and update database



Demo : How to Create Registration Page

1. Add RegisterViewModel Class

public class RegisterViewModel

{

public string FirstName { get; set; }

public string LastName { get; set; }

[Required(ErrorMessage = "Email is required")]

[EmailAddress]

public string Email { get; set; }

[Required(ErrorMessage = "Password is required")]

[DataType(DataType.Password)]

public string Password { get; set; }

[DataType(DataType.Password)]

[Compare("Password", ErrorMessage = "The password and confirmation password do not match.")]

public string ConfirmPassword { get; set; }

public string DrivingLicense { get; set; }

}

1. Add Register.cshtml

@model sample\_app.ViewModels.RegisterViewModel

<h1>Register</h1>

<hr />

<div class="row">

<div class="col-md-4">

<form **asp-action**="Register">

<div **asp-validation-summary**="All" class="text-danger"></div>

<div class="form-group">

<label **asp-for**="FirstName" class="control-label"></label>

<input **asp-for**="FirstName" class="form-control" />

<span **asp-validation-for**="FirstName" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="LastName" class="control-label"></label>

<input **asp-for**="LastName" class="form-control" />

<span **asp-validation-for**="LastName" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="Email" class="control-label"></label>

<input **asp-for**="Email" class="form-control" />

<span **asp-validation-for**="Email" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="Password" class="control-label"></label>

<input **asp-for**="Password" class="form-control" />

<span **asp-validation-for**="Password" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="ConfirmPassword" class="control-label"></label>

<input **asp-for**="ConfirmPassword" class="form-control" />

<span **asp-validation-for**="ConfirmPassword" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="DrivingLicense" class="control-label"></label>

<input **asp-for**="DrivingLicense" class="form-control" />

</div>

<div class="form-group">

<input type="submit" value="Create" class="btn btn-primary" />

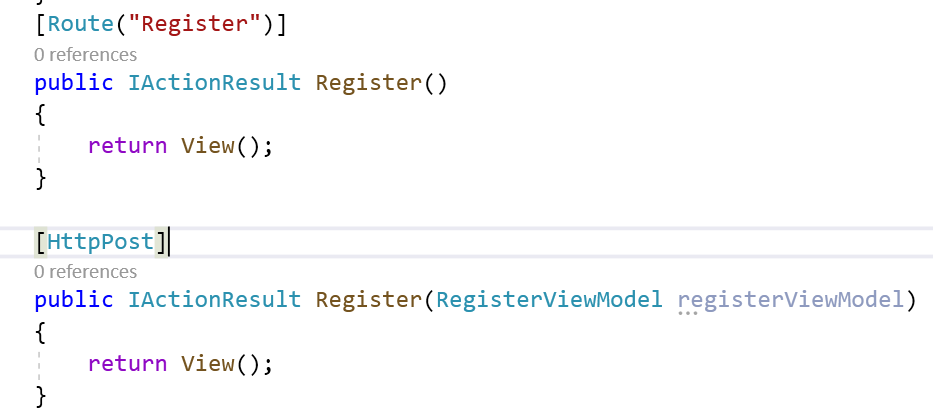
</div>

</form>

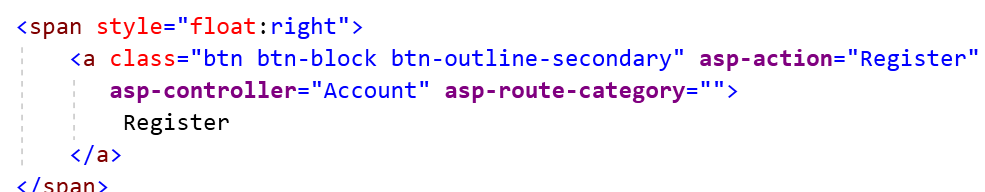
</div>

</div>

1. Add Action Method in AccountController



1. Add a Link for Register in Layout Page



1. Add code for HttpPost for Register Action

[HttpPost]

[Route("Register")]

public async Task<IActionResult> Register(RegisterViewModel registerViewModel)

{

if (!ModelState.IsValid)

{

return View(registerViewModel);

}

// Convert Register ViewModel to ApplicationUser

var user = \_mapper.Map<ApplicationUser>(registerViewModel);

var result = await \_userManager.CreateAsync(user, registerViewModel.Password);

if (!result.Succeeded)

{

foreach (var error in result.Errors)

{

ModelState.TryAddModelError(error.Code, error.Description);

}

return View(registerViewModel);

}

await \_userManager.AddToRoleAsync(user, "User");

return RedirectToAction("Index", "Home");

}

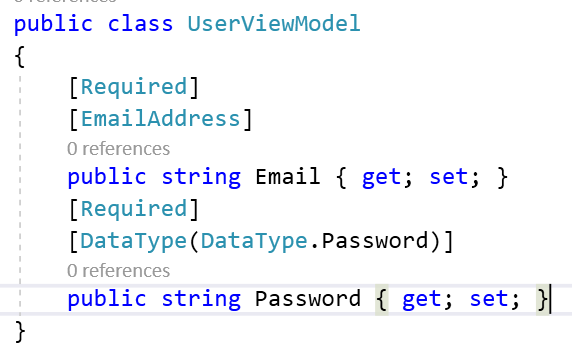
1. Add Mapping for UserName



1. Check the Output

Implement Login

1. Add UserLoginViewModel



1. Add UI for Login that is Login.cshtml

@model sample\_app.ViewModels.UserViewModel

<h1>Login</h1>

<div class="row">

<div class="col-md-4">

<form **asp-action**="Login">

<div **asp-validation-summary**="ModelOnly" class="text-danger"></div>

<div class="form-group">

<label **asp-for**="Email" class="control-label"></label>

<input **asp-for**="Email" class="form-control" />

<span **asp-validation-for**="Email" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="Password" class="control-label"></label>

<input **asp-for**="Password" class="form-control" />

<span **asp-validation-for**="Password" class="text-danger"></span>

</div>

<div class="form-group">

<input type="submit" value="Log In" class="btn btn-primary" />

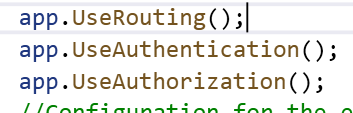
</div>

</form>

</div>

</div>

1. Add Middlewares



1. Add Action code for the same

[Route("Login")]

[HttpPost]

public async Task<IActionResult> Login(UserViewModel userViewModel)

{

if (!ModelState.IsValid)

{

return View(userViewModel);

}

var result = await \_signInManager.PasswordSignInAsync(userViewModel.Email,

userViewModel.Password, false, false);

if (result.Succeeded)

{

return RedirectToAction("Index", "Home");

}

else

{

ModelState.AddModelError("", "Ïnvalid UserName and Password");

return View();

}

}

1. Go to\_Layout.cshtml page and add links

<span style="float:right">

<a class="btn btn-block btn-outline-secondary" **asp-action**="Register"

**asp-controller**="Account" **asp-route-category**="">

Register

</a>

</span>

<span style="float:right">

<a class="btn btn-block btn-outline-secondary" **asp-action**="Login"

**asp-controller**="Account" **asp-route-category**="">

Login

</a>

</span>

<span style="float:right">

<a class="btn btn-block btn-outline-secondary" **asp-action**="Login"

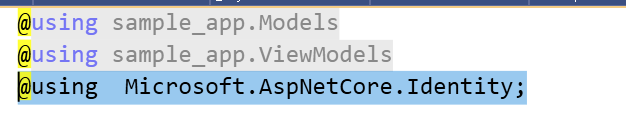
**asp-controller**="Account" **asp-route-category**="">

Logout

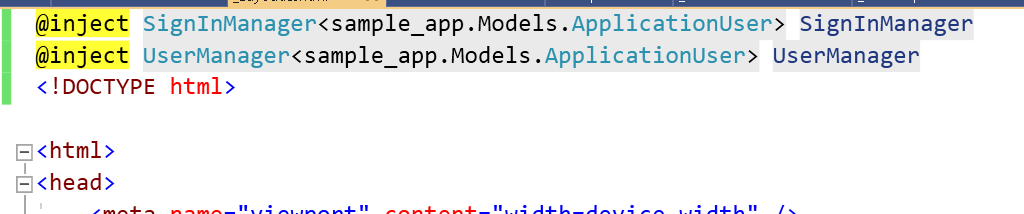
</a>

</span>

1. Check the Output
2. Enable Register Links when the user is not logged in
3. Go to \_viewImports.cshtml and add Identity Namespace

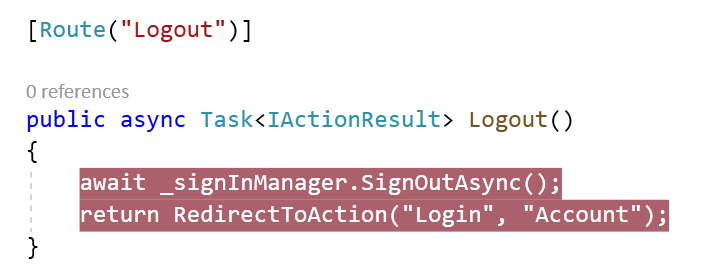


1. Add Conditions in \_Layout.cshtml





1. Check the Output
2. Add Code for Logout



1. Check the Output

Add returnURL for our application

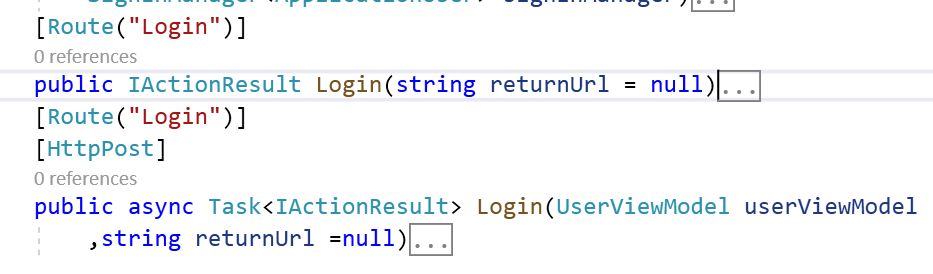
1. Add Authorize attribute in HomeController



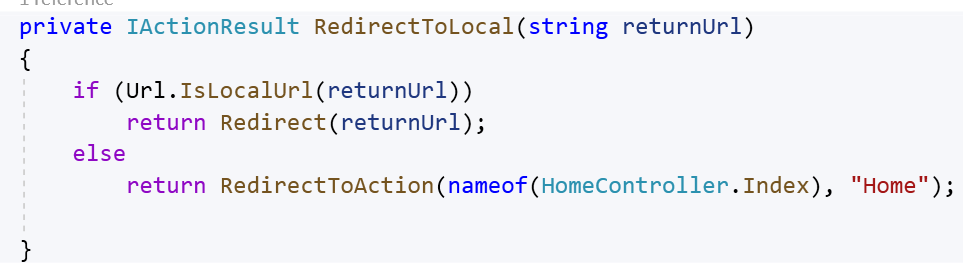
1. Check the Output

It will show the error

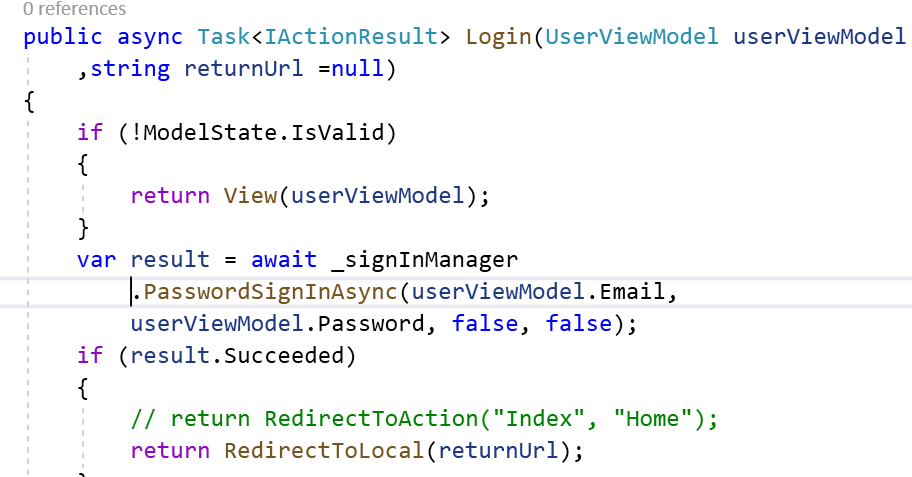
1. Add Code in AccountController to Work with Return URL



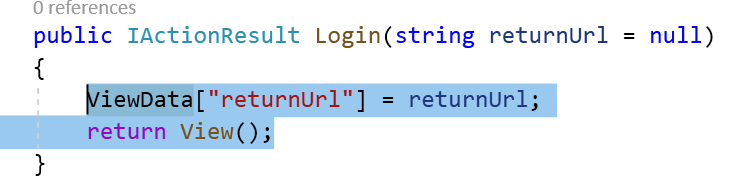
1. Add a Method to RedirectToLocal ,to handle redirection conditionally



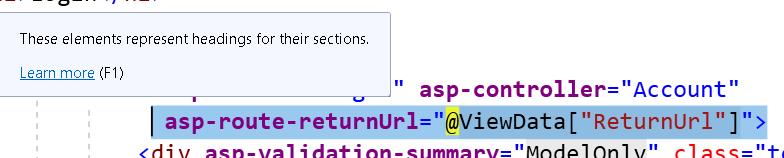
1. Modify the Login Action for Redirecting to Local



1. Modify Login action to store return URL in ViewData



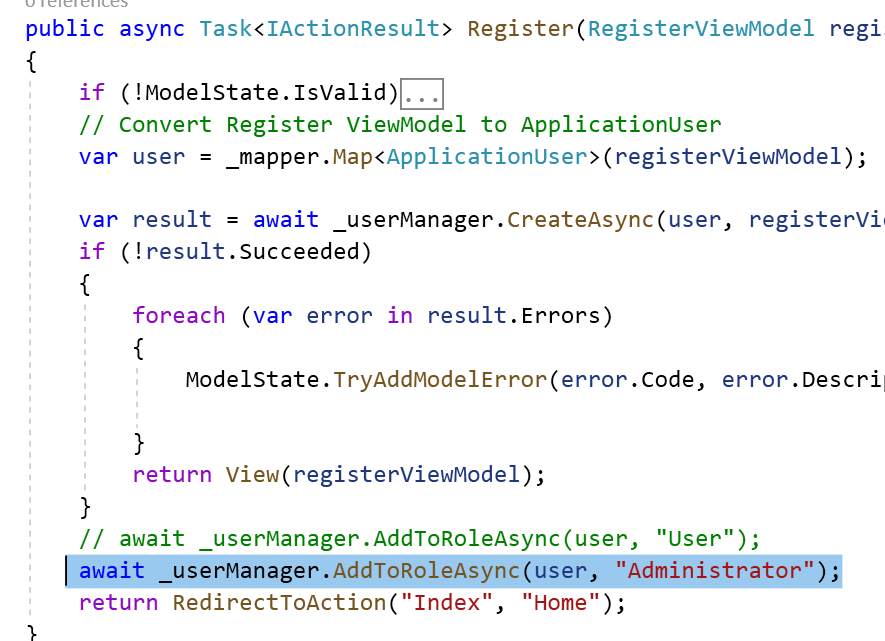
1. Send the Return URL using Login.cshtml



1. Check the Output

Problem : Lets say we want to allow ProductController , Add, Delete and Update function to be available only to administrator

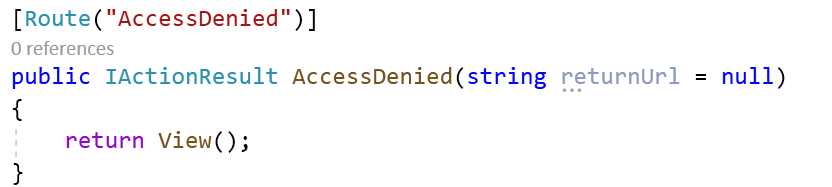
1. Add New User with Administrator Roles



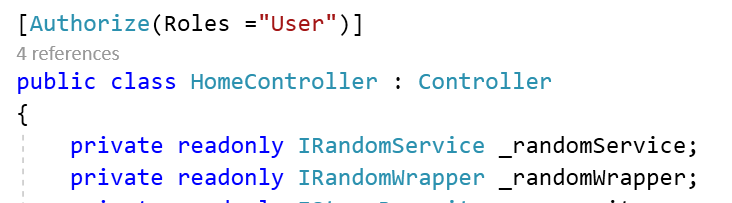
1. Authorize Action with those roles



1. Add accessdenied page



1. Check the Output. By Signing in as administrator or user and check how its working
2. Add Roles at the top of controller for user



1. Check the Output

Working of JSON Web Token

Concept

A user enters its username, password and presses the login button. After pressing the login button, a client (eg web browser) sends the user’s data to the server’s **API** endpoint.

When the server validates the user’s credentials and confirms that the user is valid, it’s going to send an encoded JWT to the client. JSON web token is basically a JavaScript object that can contain some attributes of the logged-in user. It can contain a username, user subject, user roles or some other useful information.

At the client-side, we store the JWT in the browser’s local storage to remember the user’s login session. We may also use the information from the JWT to enhance the security of our application as well.

JSON web tokens enable a secure way to transmit data between two parties in the form of a JSON object. It’s an open standard and it’s a popular mechanism for web authentication

Json web token consists of

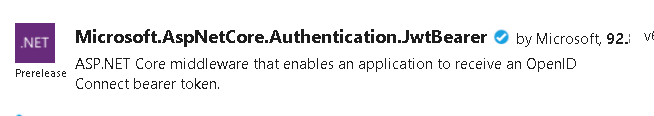
1. Header : Standard Part of JWT , JSON object encoded in the base 64 format .It contains information like type of token and name of algo
2. Payload : Contains user related information , like user id m subject and the information about user role.
3. Signature : Server uses the signature part to verify whether token contains valid information , the information w hich the server is issuing . It’s a digial signature that gets generated by combining the header and payload together. Its based on the secret key that only server knows.
4. So if malicious use try to modify the values in the payload they to recreate the signature, they need secret key which only server has.

Demo : How to Perform JWT Authentication in Web API

1. Add Identity Package



1. Add JWTBearer Package



1. Add New Class ApplicationUser

public class ApplicationUser : IdentityUser

{

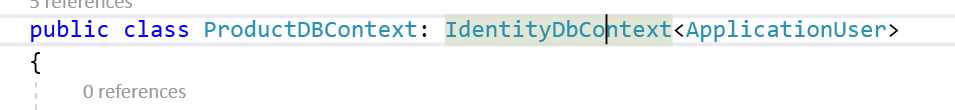
public string FirstName { get; set; }

public string LastName { get; set; }

public string DrivingLicense { get; set; }

}

1. Modify the ProductDBContext to work with IdentityDBContext



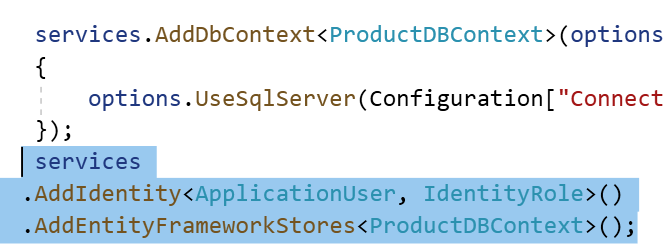
1. Change the connection string to the app that has identity tables

"ConnectionStrings": {

"ProductConnection": "Data Source=SWAPNIL-PC\\SQLEXPRESS;Initial Catalog=BOATest;Integrated Security=true"

},

1. Add Configuration for Authentication for ProductWebAPI



1. Add JwtAuthenticaion Support:

We need to add authentication middleware first.

Issure :Actual server that created the token

Audience : Receiver of token

Lifetime : Whether token has not expired

Key : Valid and is trusted by the server

services.AddAuthentication(opt =>

{

opt.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;

opt.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;

})

.AddJwtBearer(options =>

{

options.TokenValidationParameters = new Microsoft.IdentityModel.Tokens.TokenValidationParameters

{

ValidateIssuer = true,

ValidateAudience = true,

ValidateLifetime = true,

ValidateIssuerSigningKey = true,

ValidIssuer = "http://localhost:5000",

ValidAudience = "http://localhost:5000",

IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("MySecucretKey@845"))

};

});

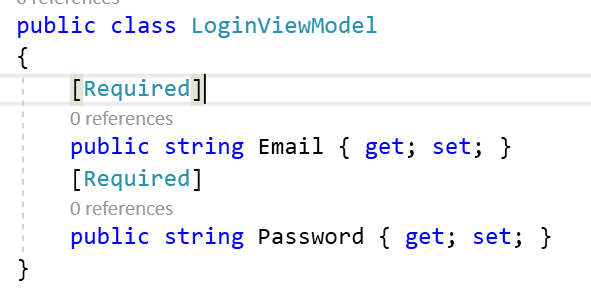
1. Add Authorization

app.UseRouting();

app.UseAuthentication();

app.UseAuthorization();

1. Add LoginViewModel



1. Go to AccountController and add SignInManager in Constructor

private readonly SignInManager<ApplicationUser> \_signInManager;

public AccountController(SignInManager<ApplicationUser> signInManager)

{

\_signInManager = signInManager;

}

1. Create method that validates user

public async Task<bool> UserExists(LoginViewModel userViewModel)

{

var result = await \_signInManager

.PasswordSignInAsync(userViewModel.Email,

userViewModel.Password, false, false);

return (result.Succeeded == true ? true : false);

}

1. Call this Method in Login Action

[HttpPost, Route("login")]

public IActionResult Login([FromBody] LoginViewModel user)

{

if (user == null)

{

return BadRequest("Invalid Client Request");

}

if (UserExists(user).Result)

{

var secretKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("MySecucretKey@845"));

var signinCredentials = new SigningCredentials(secretKey, SecurityAlgorithms.HmacSha256);

var claims = new[]

{

new Claim(JwtRegisteredClaimNames.Sub,"ProductAPIServiceAccessToken"),

new Claim(JwtRegisteredClaimNames.Jti,Guid.NewGuid().ToString()),

new Claim(JwtRegisteredClaimNames.Iat,DateTime.Now.ToString()),

new Claim("Email",user.Email)

};

var tokenOptions = new JwtSecurityToken(

issuer: "http://localhost:5000",

audience: "http://localhost:5000",

claims: claims,

expires: DateTime.Now.AddSeconds(30),

signingCredentials: signinCredentials

);

var tokenString = new JwtSecurityTokenHandler().WriteToken(tokenOptions);

return Ok(new { Token = tokenString });

}

else

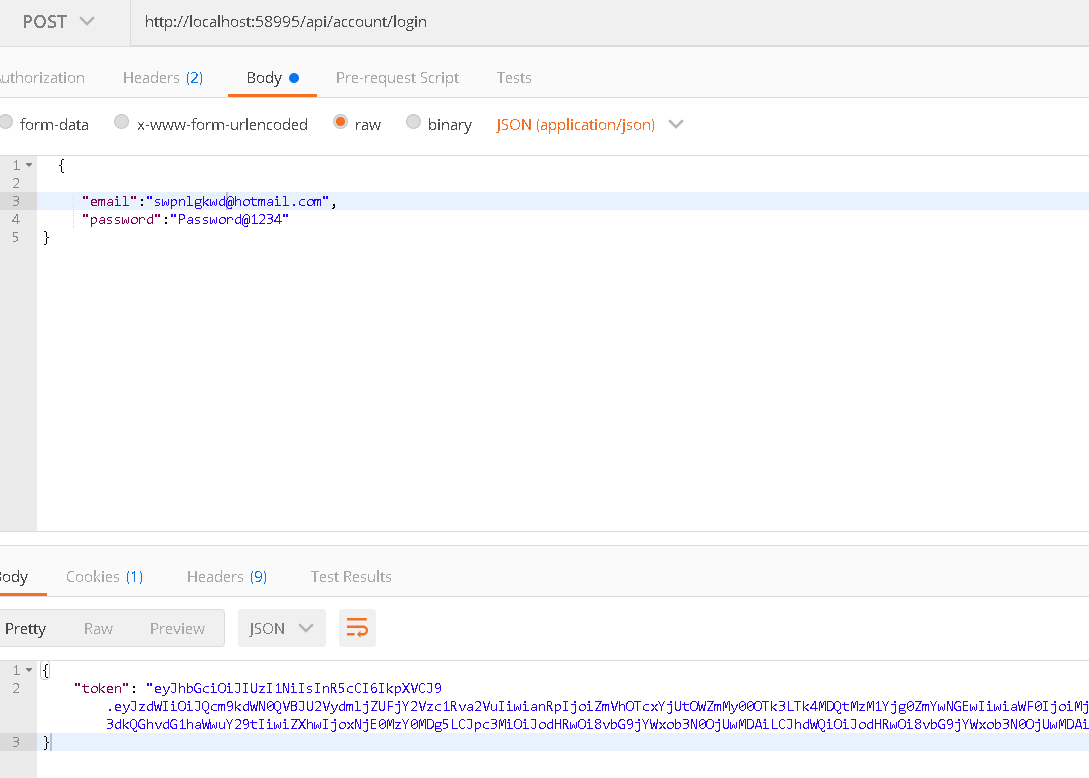
{

return Unauthorized();

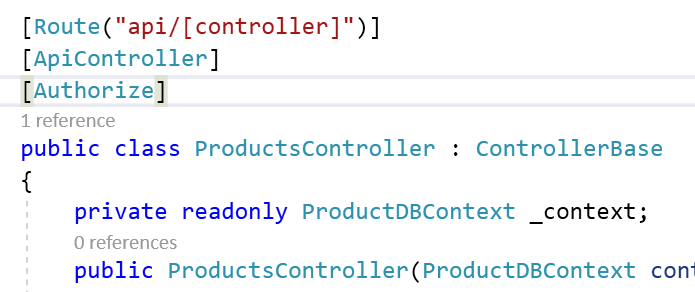
}

}

1. Go to Postman and Pass Data



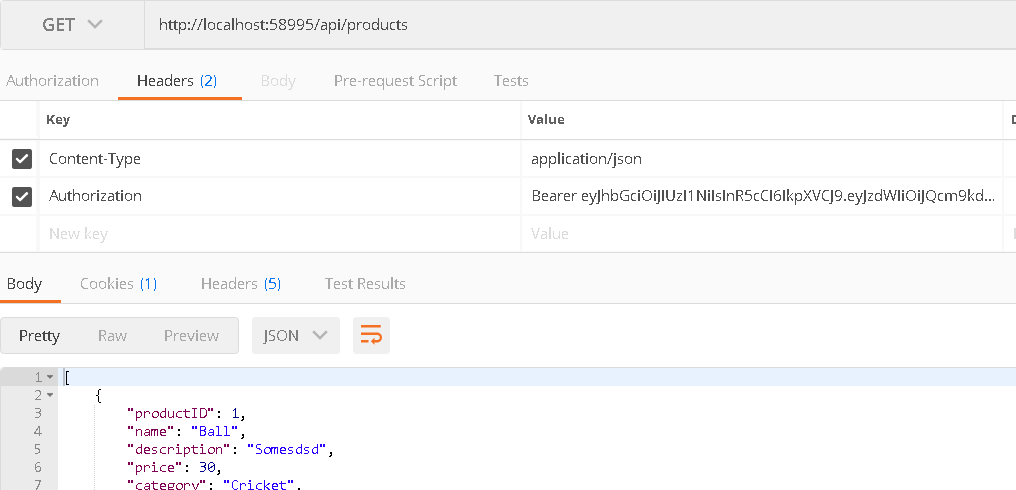
1. Add Authorize attribute on productcontroller



1. Try to access productcontroller

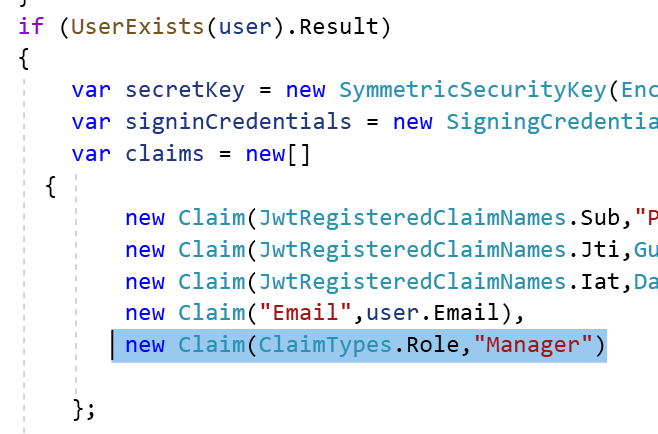
It will give error

1. Pass Token

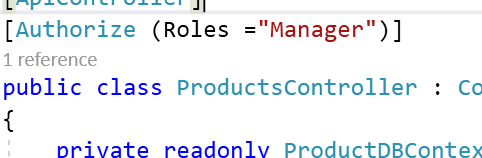


Add New Role Manager in the Claim Type

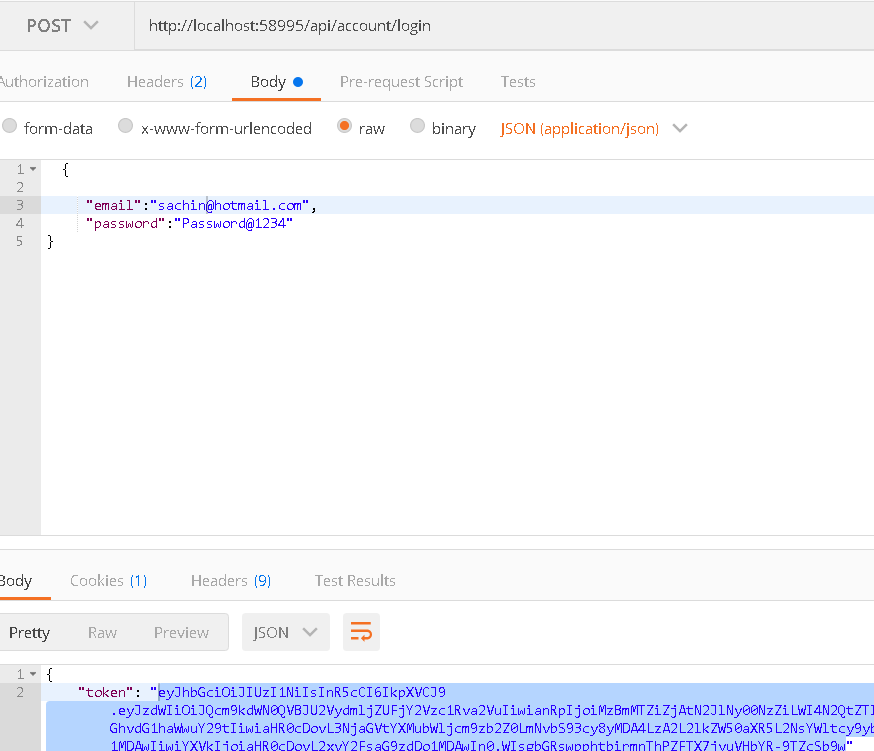
1. Add Claim Type



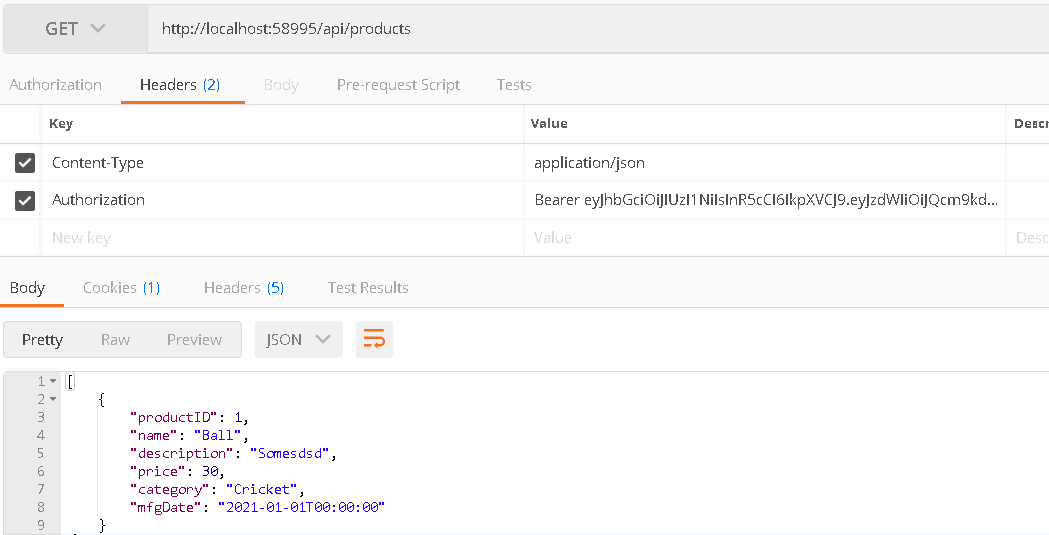
1. Add Roles to ProductCtrl



1. Create a token with Manager Role



1. Access Product with the token



What is Azure Active Directory : [Quickstart: Add sign-in with Microsoft to an ASP.NET Core web app - Microsoft identity platform | Microsoft Docs](https://docs.microsoft.com/en-us/azure/active-directory/develop/quickstart-v2-aspnet-core-webapp)

* Microsoft based cloud based identity and access management service.
* Helps employees sign in and access resources in

1. External Resources such as Microsoft 365 , Azure Portal and Thousands of SAAS applications.
2. Internal resources such as apps on your corporate network and intranet ,along with any cloud apps developed by your own organization.

* For an Identity Provider to know that a user has access to a particular app. Both app and user should be registered with an identity provider.

What is **OpenID** Connect

* Simple Identity Layer on top of OAuth2.0 protocol it allows client to verify the identiy of the end user based on authentication performed by an authorization server and obtain basic information about end user in an REST like manner.
* OpenID Connect lets developers authenticate their users across websites and apps without having to own and manage password files. For the app builder,.
* OpenID Connect allows for clients of all types, including browser-based JavaScript and native mobile apps, to launch sign-in flows and receive verifiable assertions about the identity of signed-in users.

What is **OAuth 2.0**

* Framework designed to support authentication and authorization protocols.
* Provides a variety of standardized message flow based on JSON.
* OAuth is an Open Standard Authorization protocol or framework that provides applications the ability for secure designated access.

You can tell Facebook that its ok for ESPN.com to access your profile or post updates to your timeline without having to give ESPN your Facebook password. This minimizes risk in the event ESPN suffers a breach and your FB pws remains safe.

* Allows you to approve one application interacting with another on your behalf without giving away your password.
* Facebook apps are a good OAuth use case example. Say you’re using an app on Facebook, and it asks you to share your profile and pictures. Facebook is, in this case, the service provider: it has your login data and your pictures. The app is the consumer, and as the user, you want to use the app to do something with your pictures. You specifically gave this app access to your pictures, which OAuth is managing in the background.
* Your smart home devices – toaster, thermostat, security system, etc. – probably use some kind of login data to sync with each other and allow you to administer them from a browser or client device. These devices use what OAuth calls confidential authorization. That means they hold onto the secret key information, so you don’t have to log in over and over again.

Azure AD is intended for

1. App Developers

* Adding for **Single Sign** on to your app, allowing it to work with users pre-existing credentials.

Web App Sign In Flow

* Web APP determines whether user is Authenticated
* If the user isnot authenticated the web app delegates to Azure AD
* User Login and asked to consent to the access that client app needs. Microsoft Identity Platform can deliver tokens representing the access that user has consented to.

Once User is Authenticated

1. MS Identity Platform sends a token to the Web APP
2. Cookie saved , that contains the identity of the user .
3. Web app validates the token and then web app displays web page.

Security Tokens

1. Access Tokens

* Issued by authorization server as part of OAuth 2.0 flow.
* Contains information about the user and resource for which the token is intended.
* This tokens are used for access web api and other protected resources.
* Access tokens are passed to web api as the bearer token in the authorization header.

1. ID tokens

* As a Part of OpenID connect flow.
* They can be sent along side or instead of an access token. ID tokens are used by the client to authenticate the user.

What all things are present inside Claims

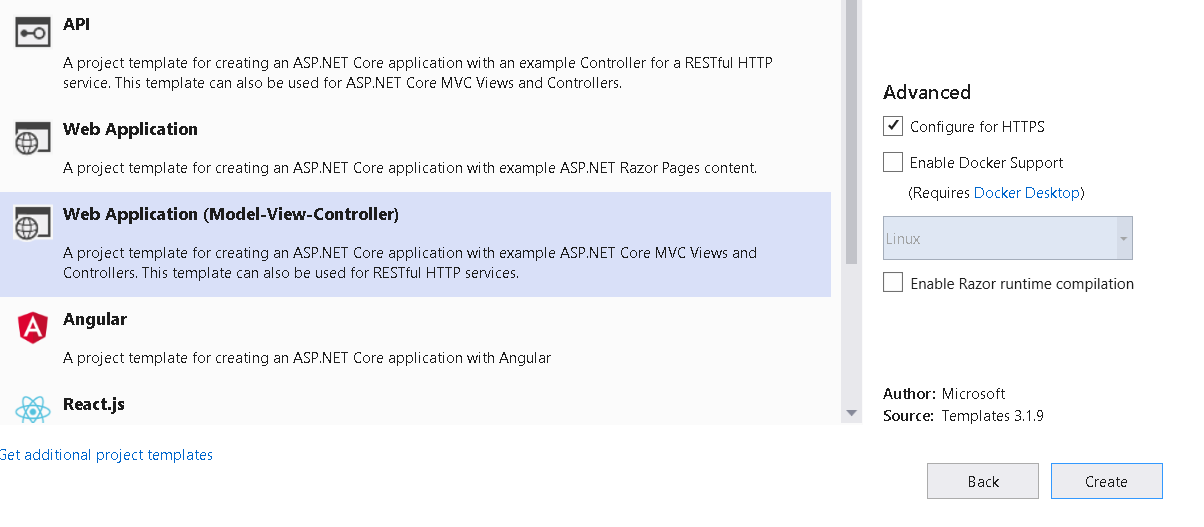
* [Provide optional claims to Azure AD apps - Microsoft identity platform | Microsoft Docs](https://docs.microsoft.com/en-us/azure/active-directory/develop/active-directory-optional-claims#v20-specific-optional-claims-set)

Demo : How to Authenticate App using Azure AD

1. Create new project

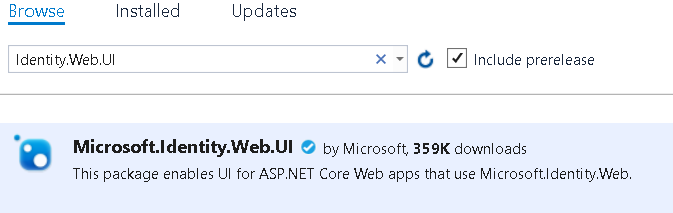


1. Add new MVC



1. Add New package

* [Microsoft.Identity.Web](https://www.nuget.org/packages/Microsoft.Identity.Web) - The main package. Required by all apps that use Microsoft Identity Web.
* [Microsoft.Identity.Web.UI](https://www.nuget.org/packages/Microsoft.Identity.Web.UI) - Optional. Adds UI for user sign-in and sign-out and an associated controller for web apps.



1. Change application url port to 44321



1. Change ConfigureService section and add following code to enable Identity

services.AddAuthentication(OpenIdConnectDefaults.AuthenticationScheme)

.AddMicrosoftIdentityWebApp(Configuration.GetSection("AzureAd"));

services.AddControllersWithViews(options =>

{

var policy = new AuthorizationPolicyBuilder()

.RequireAuthenticatedUser().Build();

options.Filters.Add(new AuthorizeFilter(policy));

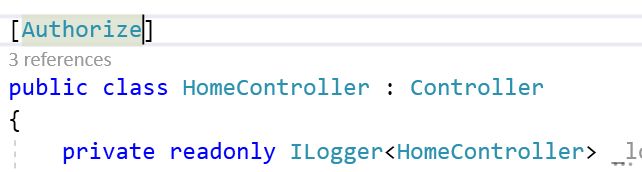
});

services.AddRazorPages().AddMicrosoftIdentityUI();

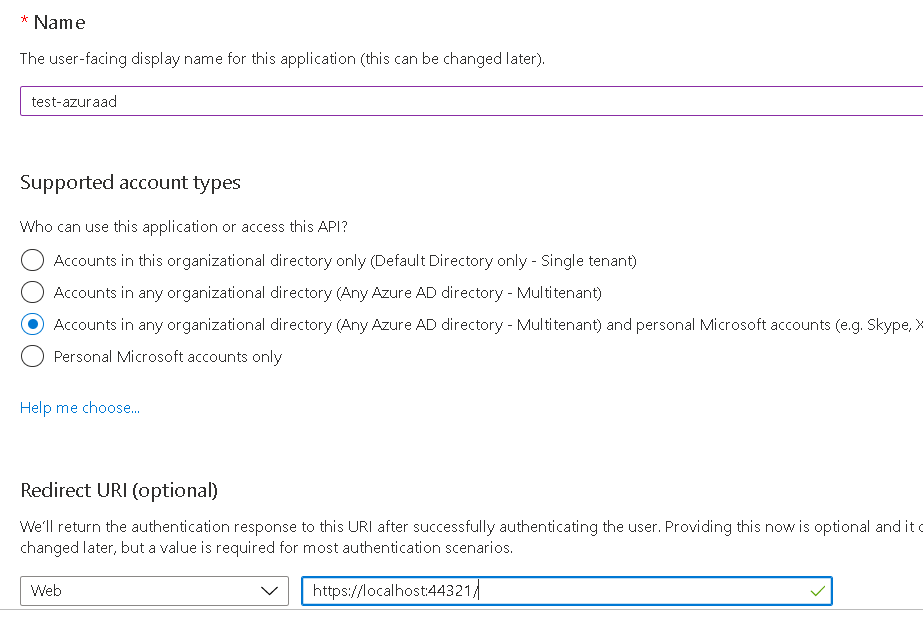
1. Enable Authentication in Request pipeline

App.UseAuthentication()

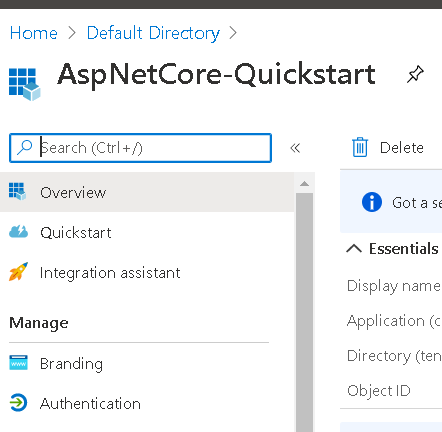
1. Add Authorize attribute to HomeCtrl



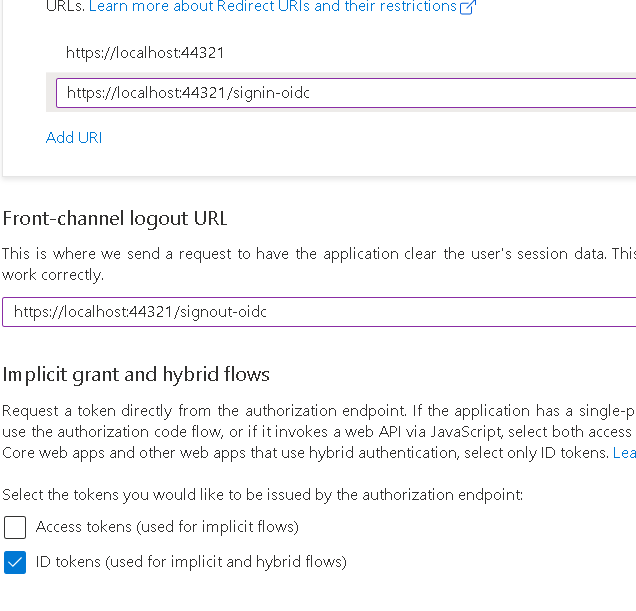
1. Register this app to Azure active directory



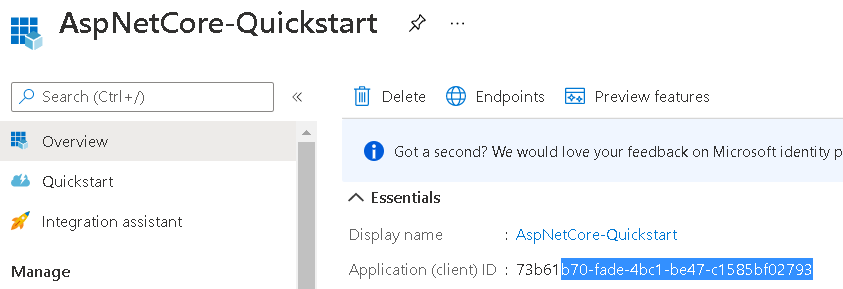
1. . G to Authentication option of the App



1. Configure RedirectURL : <https://localhost:44321/> and Signout URL : https://localhost:44321/signin-oidc



1. Go to Appsettings and add ApplicationID



1. Add Application in appsettings.json of your application

"AzureAd": {

"Instance": "https://login.microsoftonline.com/",

"ClientId": "0f2728a0-379f-44ff-a688-6704ce47c4c8",

"TenantId": "common"

},

1. Add Identity in Layout cshtml

@if (User.Identity.IsAuthenticated)

{

<li class="nav-item">

<span class="navbar-text text-dark">Hello @User.Identity.Name!</span>

</li>

<li class="nav-item">

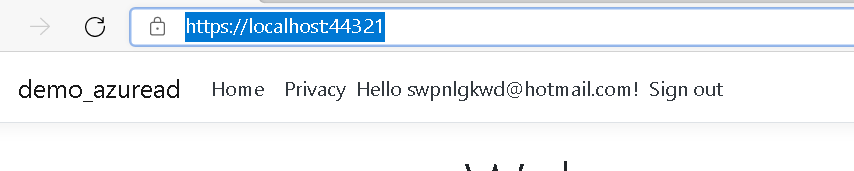
<a class="nav-link text-dark" **asp-area**="MicrosoftIdentity"

**asp-controller**="Account" **asp-action**="SignOut">Sign out</a>

</li>

}

1. Check the Output



Cross Site Scripting

* Injection of malicious code in a web application, usually JS.
* Attackers try to inject code into web application. this code often saved in a database and will impact all the users.
* If attacker manage to inject JS into the product name on some site. Every user who opens the infected product would load the malicious code.

Problem

* Attackers could read your cookies and gain access to private accounts
* Redirect to malicious sites

How to Prevent it in ASP.Net core

1. Validate user input, reject or sanitize unknown character.
2. Use HttpOnly for cookies so its not readable By JS
3. Before putting untrusted data inside an HTML element ensure its HTML encoded. It encodes additional characters such as “or ‘
4. Before Putting untrusted data into JS place the data in an HTML element whose contents you retrieve at runtime. JS encoded takes dangerous characters for JS and replaces them with hex. Replace < Would be encoded as \u003c
5. Before Putting untrusted data into query string ensure its URL encoded.

Preventing Attack

1. Use CreateElement Or textContent to prevent code from being exposed to DOM based XSS

@{

var untrustedInput = "<script>alert(1)</script>";

}

<div id="injectedData"

data-untrustedinput="@untrustedInput" />

<div id="scriptedWrite" />

<div id="scriptedWrite-html5" />

<script>

var injectedData = document.getElementById("injectedData");

// All clients

var clientSideUntrustedInputOldStyle =

injectedData.getAttribute("data-untrustedinput");

// HTML 5 clients only

var clientSideUntrustedInputHtml5 =

injectedData.dataset.untrustedinput;

// Put the injected, untrusted data into the scriptedWrite div tag.

// Do NOT use document.write() on dynamically ge

nerated data as it

// can lead to XSS.

document.getElementById("scriptedWrite").innerText += clientSideUntrustedInputOldStyle;

// Or you can use createElement() to dynamically create document elements

// This time we're using textContent to ensure the data is properly encoded.

var x = document.createElement("div");

x.textContent = clientSideUntrustedInputHtml5;

document.body.appendChild(x);

// You can also use createTextNode on an element to ensure data is properly encoded.

var y = document.createElement("div");

y.appendChild(document.createTextNode(clientSideUntrustedInputHtml5));

document.body.appendChild(y);

</script>

1. Run the application

=============================================================

* FromServicesAttribute
* FromRouteAttribute
* FromQueryAttribute
* FromBodyAttribute
* FromFormAttribute

1. FromForm

* Use to Select form data as the source of binding data. Name of parameter is used to locate a form value by default

1. FromRoute : Use to select the routing system as the source of binding data. The name of the parameters is used to locate a route data.
2. FromQuery : Use as Querystring as the source of binding data.
3. FromHeader : Use request header as the source of binding data.
4. FromBody : Use to specify request body as the source of binding data. Which is required when you want to receive data from requests that are not form encoded.
5. FromService : Gets the values from DI

When to use which : [Parameter Binding in ASP.NET Web API - ASP.NET 4.x | Microsoft Docs](https://docs.microsoft.com/en-us/aspnet/web-api/overview/formats-and-model-binding/parameter-binding-in-aspnet-web-api#using-frombody)