[**1. SECTION – 1 : Auth-Service-Integration-Using-API-Gateway** 1](#_Toc171712147)

[**2. SECTION – 2 : Authentication at Support UI** 13](#_Toc171712148)

[**3. SECTION – 3 : Authentication and Decomposition Pattern** 24](#_Toc171712149)

# **1. SECTION – 1 : Auth-Service-Integration-Using-API-Gateway**

**Folder Structure Creation:**

1. Create New Project 🡪 select blank solution template 🡪 Give a name to empty solution as eShopFlix29Jun and select folder path.
2. Add BackendServices folder under empty solution 🡪 Right Click on folder 🡪 Add 🡪 New Project 🡪 Select ASP.NET Core Web API template 🡪 Next 🡪 Give Project name as AuthService and Location remains as it is by default selected 🡪 Next 🡪 Set the below checkbox as per screenshot 🡪 Create.
3. Same repeat the steps for CatalogService and OrderService under BackendServices folder.

A screenshot of a computer

Description automatically generated

1. Add FrontendServices folder under empty solution 🡪 Right Click on folder 🡪 Add 🡪 New Project 🡪 Select ASP.NET Core Web App (Model - View - Controller) template 🡪 Next 🡪 Name the project as WebApp and select FrontendServices folder 🡪 Next 🡪 Select the below screenshot setting 🡪 Create

A screenshot of a web application

Description automatically generated

1. Repeat the same step of 4 for the SupportApp
2. Add ApiGateways folder under empty solution 🡪 Right Click on folder 🡪 Add 🡪 New Project 🡪 Select ASP.NET Core Empty template 🡪 Next 🡪 Select path as ApiGateways folder inside empty solution 🡪 Next 🡪 Select the below checkboxes as per screen shot 🡪 Create.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Create the Database **AuthService29Jun** and Run the **Schema.sql** and SeedData.sql scripts under this DB in chronical order.

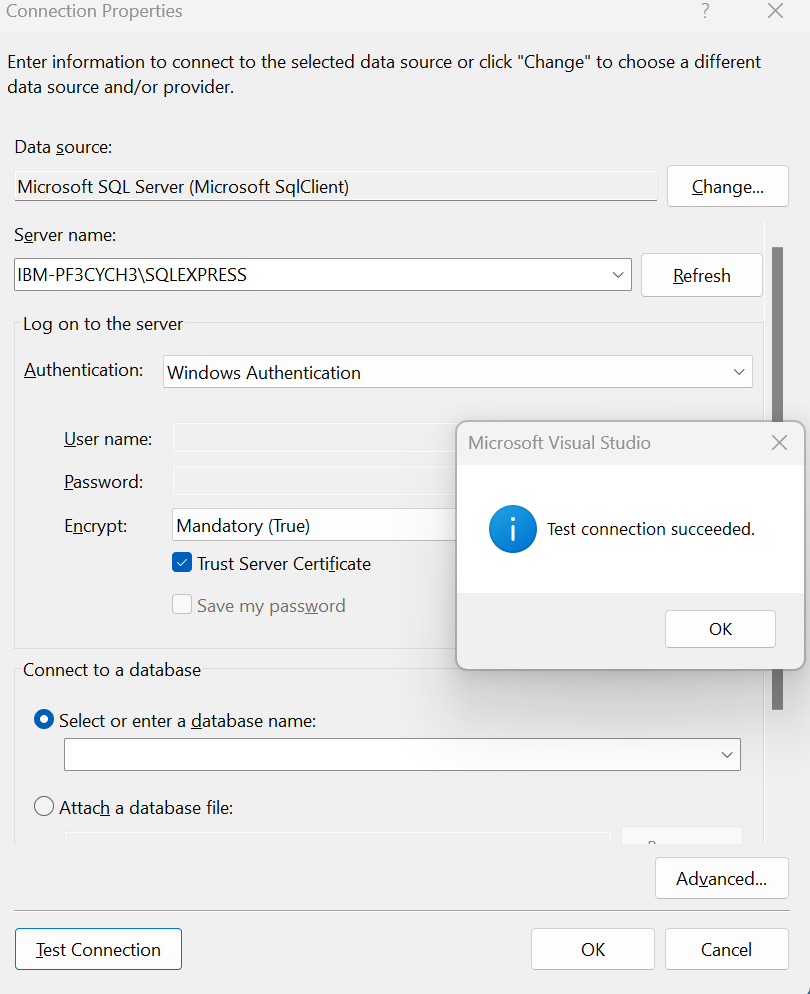
7 Install EF Core Power Tools (Visual Studio Extension)

8 Right click on AuthService project 🡪 EF Core Power Tools 🡪 Reverse Engineer.

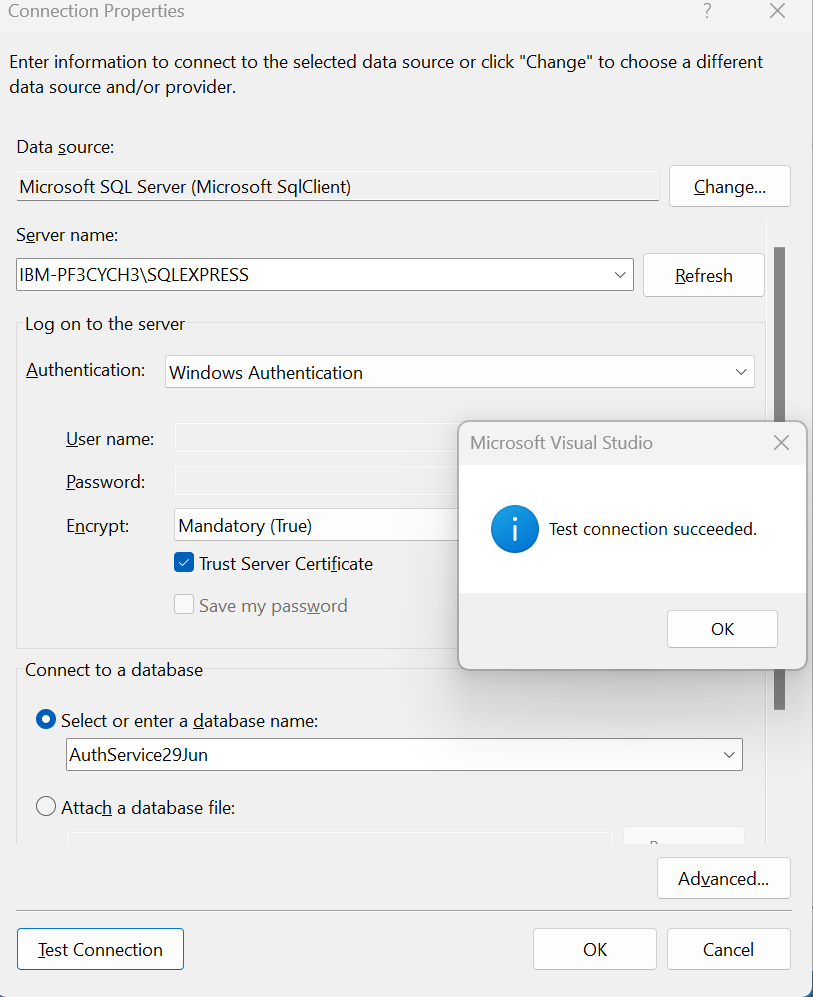
A screenshot of a computer error

Description automatically generated

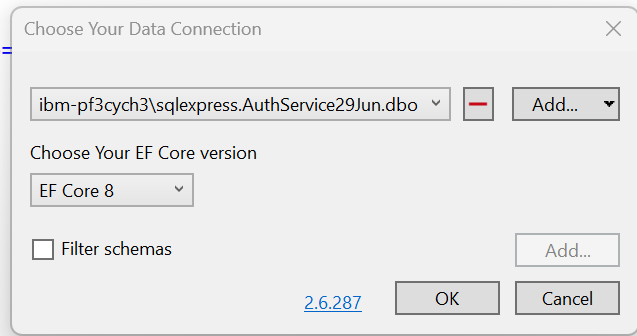
Click on Add 🡪 Add Database Connection 🡪 Check the below screen shot



Select the Database.



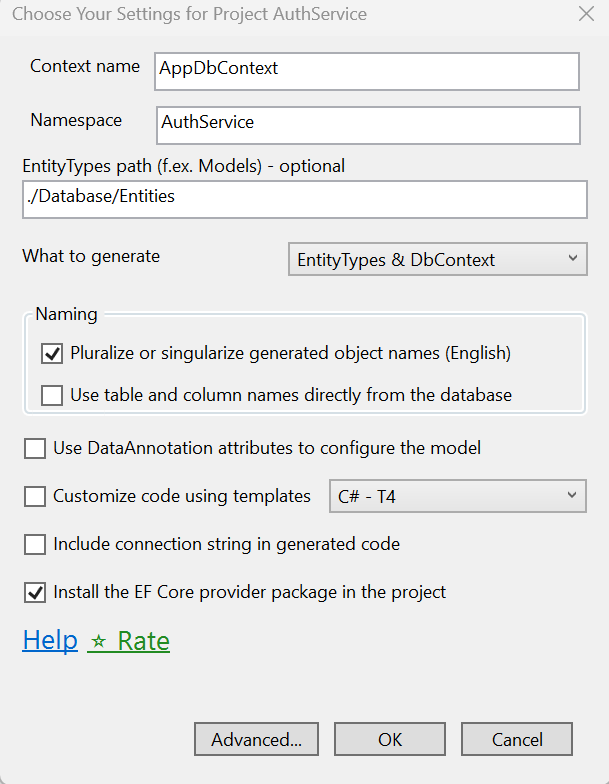
Click Ok 🡪 Click Ok ( Please check the below screen shot)



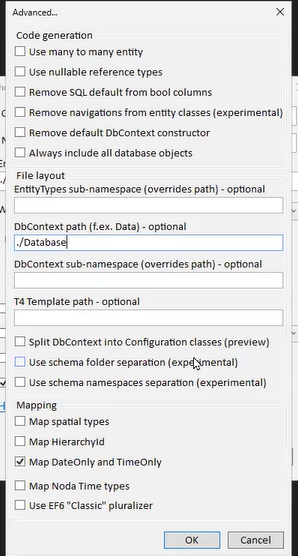
* Import all the tables from the below screenshot 🡪 Click Ok

A screenshot of a computer

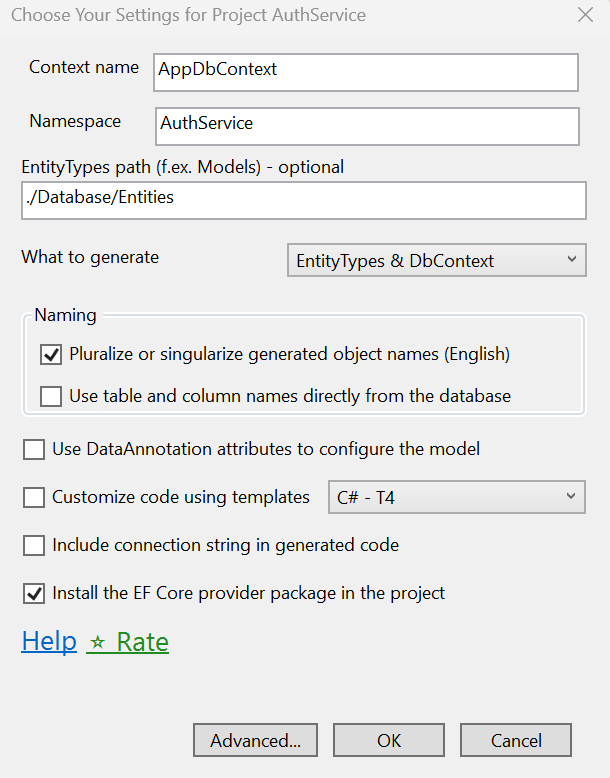
Description automatically generated



Click on Advanced 🡪 Make the below screenshot setting



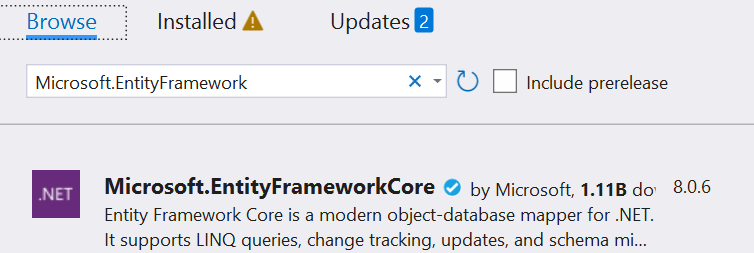
Click Ok 🡪 Click Ok of below window



9. Specify the connection string in appsettings.json file of AppService project.

10. Add the Repository and Its implementation in AuthService project.

11. Install the below EF Core dependency.



A close-up of a logo

Description automatically generated

A screenshot of a computer

Description automatically generated

12. Create the Auth Controller and call repository method in Controller and add services to the container as below.

A screenshot of a computer program

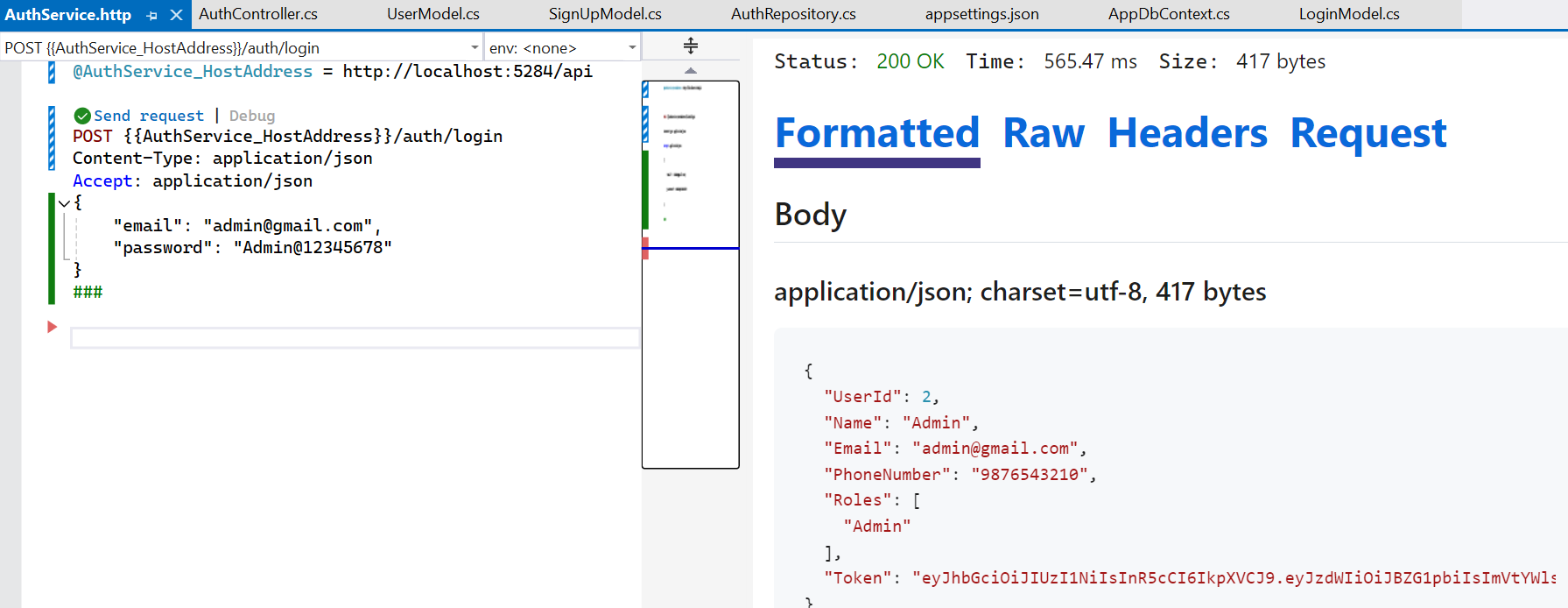
Description automatically generated

Test api using new feature of .Net Core 8.0 as below

A screenshot of a computer

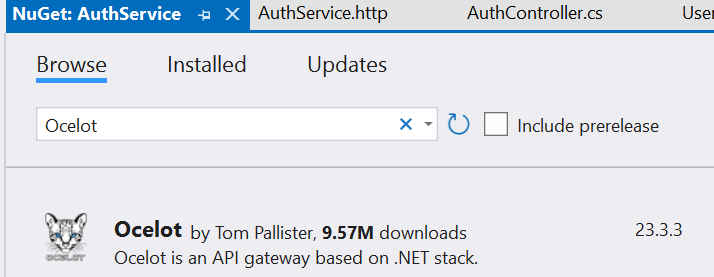
Description automatically generated

After add on token logic in AuthRepository.cs file



**Ocelot API Gateway**

13. Install Ocelot api gateway and JWT Token dependancy in ApiGateways 🡪 ApiGateway.Web



A close up of a screen

Description automatically generated

14. Add ocelot.json file in ApiGateways 🡪 ApiGateway.Web and mention launchSettings.json file https section of ApiGateway.web 7160 port with below ocelot.json file.

"GlobalConfiguration": {

"BaseUrl": "https://localhost:7160"

}

Refer the official site for API Gateway.

<https://ocelot.readthedocs.io/en/latest/introduction/gettingstarted.html>

15. Mention AuthRepository launchSettings.json file https section 7203 port in ApiGateways 🡪 ApiGateway.Web -- > ocelot.json in below section.

"DownstreamHostAndPorts": [

{

"Host": "localhost",

"Port": 7203

}

]

“DownstreamPathTemplate” => API endpoint path.

"UpstreamPathTemplate" => Frontend expose endpoint path.

In Program.cs file add the ocelot.json file for load ocelot configuration and service dependencies and ocelot middleware as below.

builder.Configuration.AddJsonFile("ocelot.json");

builder.Services.AddOcelot();

app.UseOcelot().Wait();

The final code of program.cs is as below.

using Ocelot.DependencyInjection;

using Ocelot.Middleware;

var builder = WebApplication.CreateBuilder(args);

builder.Configuration.AddJsonFile("ocelot.json");

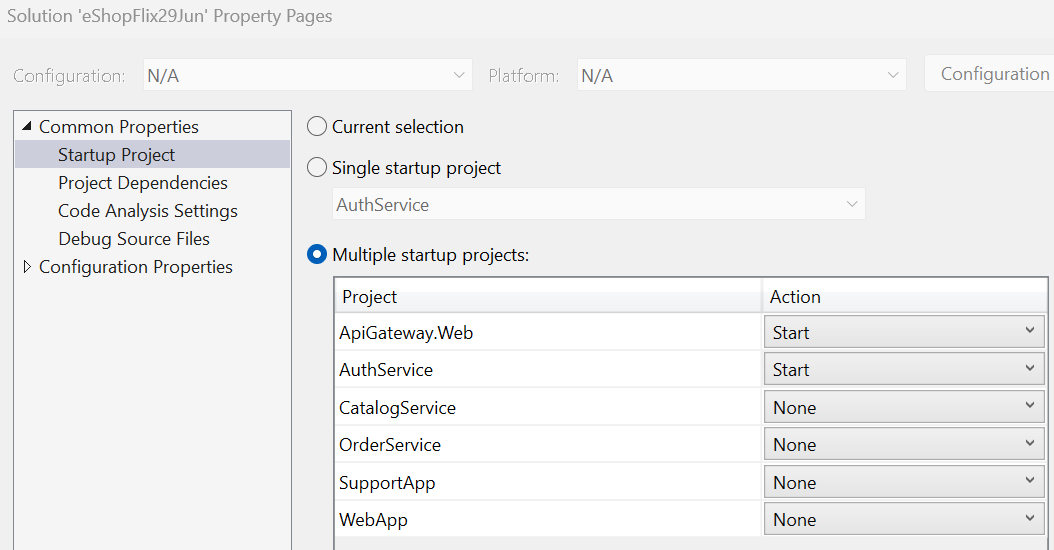
builder.Services.AddOcelot();

var app = builder.Build();

app.UseOcelot().Wait();

app.Run();

Run both the project i.e ApiGateway.Web and AuthService

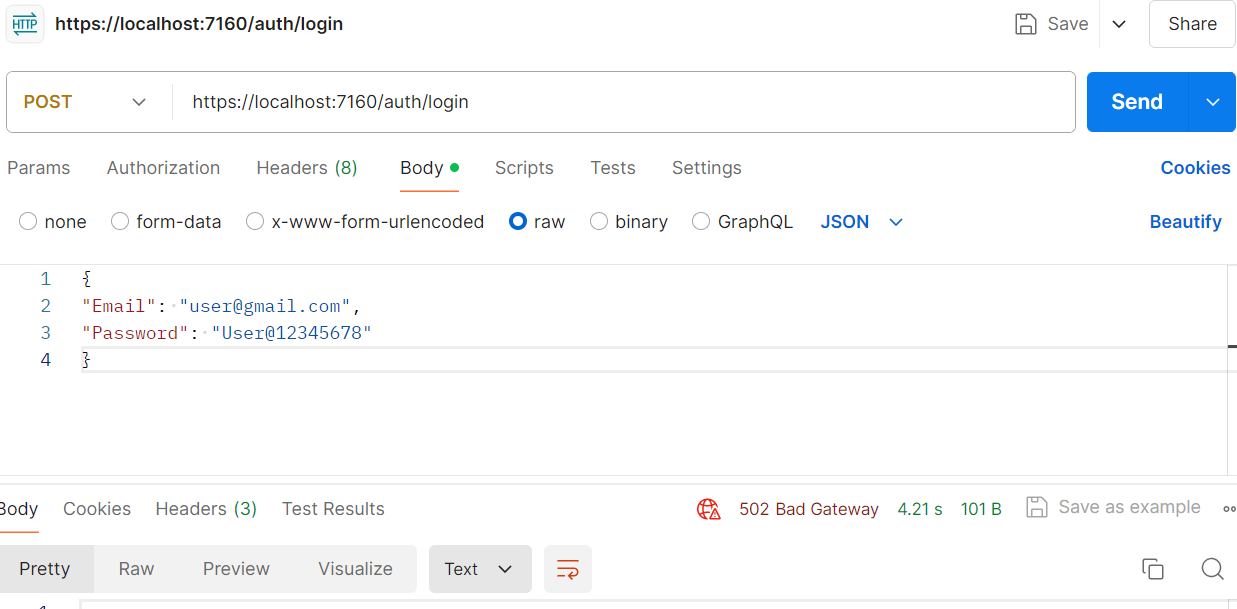


16. Download the Postman app from the below url and install it and login as google authentication.

<https://www.postman.com/downloads/?utm_source=postman-home>

A screenshot of a computer

Description automatically generated



17. FrontendServices 🡪 WebApp 🡪 Add the AccountController.cs file and its respective Action method with view and modify the Views🡪 Shared 🡪 \_Layout.cshtml file

18. Define the API Gateway address in appsettings.json file of WebApp project.

"ApiGatewayAddress": <https://localhost:7160/>

19. Used API Gateway inside program.cs file.

//Services

builder.Services.AddHttpClient<AuthService>(client =>

{

client.BaseAddress = new Uri(builder.Configuration["ApiGatewayAddress"]);

});

20. Adding the routing for area in program.cs file

app.MapControllerRoute(

name: "areas",

pattern: "{area:exists}/{controller=Home}/{action=Index}/{id?}"

);

21. Add the View in Area same as per the Root level view of WebApp.

22. Authenticating the user @ UI level by using Authentication cookie using GenerateTicket() method.

23. Add a CurrentUser property in BaseController.cs file (For controller level).

24. Same above logic we can add in UI level as a Helper and mention the

@inherits WebApp.Helpers.BaseViewPage<LoginModel> in \_ViewImports.cshtml file of Areas.

25. Add Service for AddAuthentication before AddHttpClient and auth middleware app.UseAuthentication(); in program.cs file before app.UseAuthorization().

26. Set the ApiGateway.Web, AuthService and WebApp Project as start up project.

# **2. SECTION – 2 : Authentication at Support UI**

1. Create SupportApp project same as per the WebApp project.
2. Define LoginModel.cs file in Model folder and Create AccountController.cs file. Add UserModel.cs file under Model folder.
3. Define Login Action Method and based on that create login razor view.(Right click and add view)
4. A screenshot of a computer

   Description automatically generated
5. Program.cs file, set the default route to as below

app.MapControllerRoute(

name: "default",

pattern: "{controller=Account}/{action=Login}/{id?}");

1. Define Login Action Method of http POST verb.
2. Add HttpClients Folder and add AuthService.cs file inside it.
3. Add Helpers folder and add BaseViewPage.cs file inside it.
4. Add Admin Area

A black line with white background

Description automatically generated

1. Define Admin Area route in Program.cs file

app.MapControllerRoute(

name: "areas",

pattern: "{area:exists}/{controller=Home}/{action=Index}/{id?}"

);

1. Define Home and Base controller under Admin Area. Decorate Base controller with Admin Area and Inherit Base controller from Home controller.
2. Add the view for the index method of the Home controller.
3. Copy the Shared folder and \_viewImports.cshtml and \_viewStart.cshtml from SupportApp parent folder into Admin Area View folder.
4. Import BaseViewPage under \_viewImports.cshtml for both Admin Area and Parent view.

@inherits SupportApp.Helpers.BaseViewPage<TModel>

1. Add the API Gateway configuration in Program.cs file and Authentication Middleware.

builder.Services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)

.AddCookie(options =>

{

options.Cookie.Name = "WebAppCookie";

options.LoginPath = "/Account/Login";

options.AccessDeniedPath = "/Account/UnAuthorize";

});

//Services

builder.Services.AddHttpClient<AuthService>(client =>

{

client.BaseAddress = new Uri(builder.Configuration["ApiGatewayAddress"]);

});

1. Add API Gateway address in the appsettings.json file is as below

"ApiGatewayAddress": <https://localhost:7160/>

**CatalogService.cs**

1. Create Product and Category API Controller, Its Interfaces (IProductService.cs and ICategoryService.cs) and its Implementation (ProductService.cs and CategoryService.cs)
2. Login on SQL Server (Windows Authentication) and create Database named as **CatalogService06Jul**
3. Run Schema.sql and SeedData.sql file in under created database.
4. Use DB first approach for **CatalogService06Jul** database using EF Core Power Tools same as we did in the WebApp Project.
5. Import the tables and click Ok

A screenshot of a computer

Description automatically generated

* Advnaced 🡪 Do the below setting and path
* A screenshot of a computer

  Description automatically generated
* Ok 🡪

1. Intall Nueget Packages (EFCore and EFCore.Sql) if tool is failed.
2. Add a connectionstring in appsettings.json file and add services dependancy for product and category and JsonSerializerOptions.

// Add services to the container.

builder.Services.AddDbContext<AppDbContext>(

options => options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));

builder.Services.AddScoped<ICategoryService, CategoryService>();

builder.Services.AddScoped<IProductService, ProductService>();

builder.Services.AddControllers().AddJsonOptions(options =>

{

options.JsonSerializerOptions.PropertyNamingPolicy = null;

});

1. Add a root folder (wwwroot) folder for product images. To serve a images we need to add a middleware.

app.UseStaticFiles();

1. Integrate CatalogService with API Gateway. Below is the public one (can access anyone)

{

"DownstreamPathTemplate": "/api/product/{everything}",

"DownstreamScheme": "https",

"DownstreamHostAndPorts": [

{

"Host": "localhost",

"Port": 7108

}

],

"UpstreamPathTemplate": "/catalog/{everything}",

"UpstreamHttpMethod": [ "GET" ]

},

1. Integrate CatalogService with API Gateway. Below is the private one. (For protect backend api). AuthenticationOptions is the, provide security for each service layer, because each service layer have separate security code. Here we added it for now CatalogService service.

{

"DownstreamPathTemplate": "/api/product/{everything}",

"DownstreamScheme": "https",

"DownstreamHostAndPorts": [

{

"Host": "localhost",

"Port": 7108

}

],

"AuthenticationOptions": {

"AuthenticationProviderKeys": "eShopFlixProduct@12345678"

},

"RouteClaimsRequirement": {

"Roles": "Admin"

},

"UpstreamPathTemplate": "/product/{everything}",

"UpstreamHttpMethod": [ "GET","POST","Put","Delete" ]

},

1. Add the authentication service and JWT key in ocelot.json (ApiGateway.Web) project.

"Jwt": {

"Key": "My$ecretKeyF0rAuth0123467890XYZKjkjkl8726826823",

"Issuer": "www.authapi.com",

"Audience": "www.apigateway.com"

},

"Keys": {

"CatalogService": "eShopFlixProduct@12345678"

}

1. Add the authentication for CatalogService in Program.cs file of ApiGateway.Web project.

var catalogAuthKey = builder.Configuration["Keys:CatalogService"];

builder.Services.AddAuthentication()

.AddJwtBearer(catalogAuthKey, options =>

{

options.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuer = true,

ValidateAudience = true,

ValidateLifetime = true,

ValidateIssuerSigningKey = true,

ValidIssuer = builder.Configuration["Jwt:Issuer"],

ValidAudience = builder.Configuration["Jwt:Audience"],

IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(builder.Configuration["Jwt:Key"]))

};

});

* Here service auth key (catalogAuthKey) i.e builder.Configuration["Keys:CatalogService"];

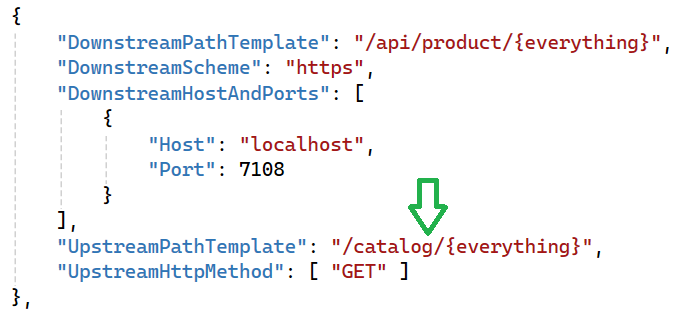
Its for **specific** service key. We can keep **one or different** auth key for all service layer.

* Recommended is use different auth key for each service layer but token is common every where.
* JWT part is common for all the service layer.

A screenshot of a computer

Description automatically generated

Public path for catalog (will open for anyone)

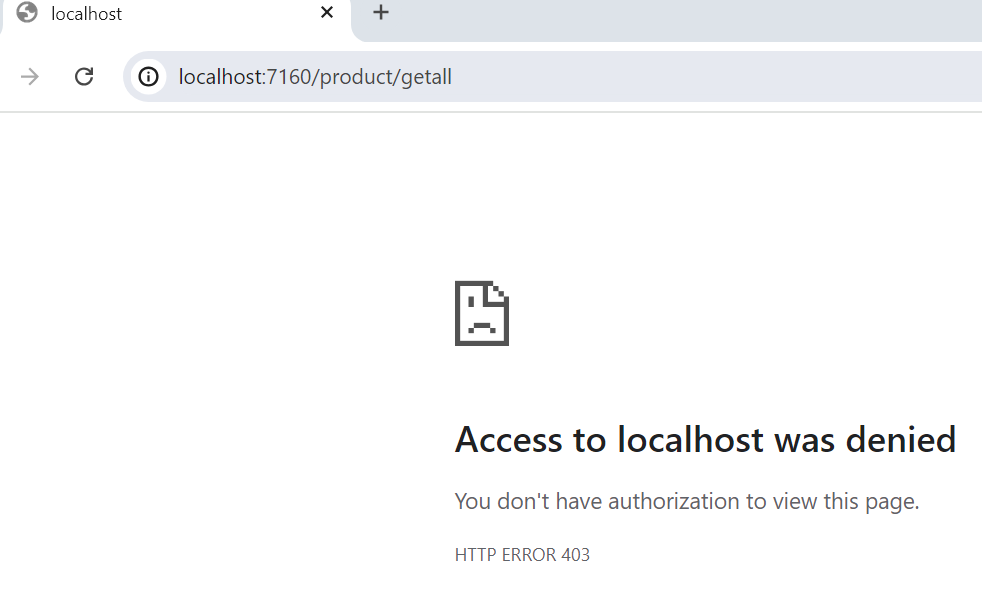


A screenshot of a computer screen

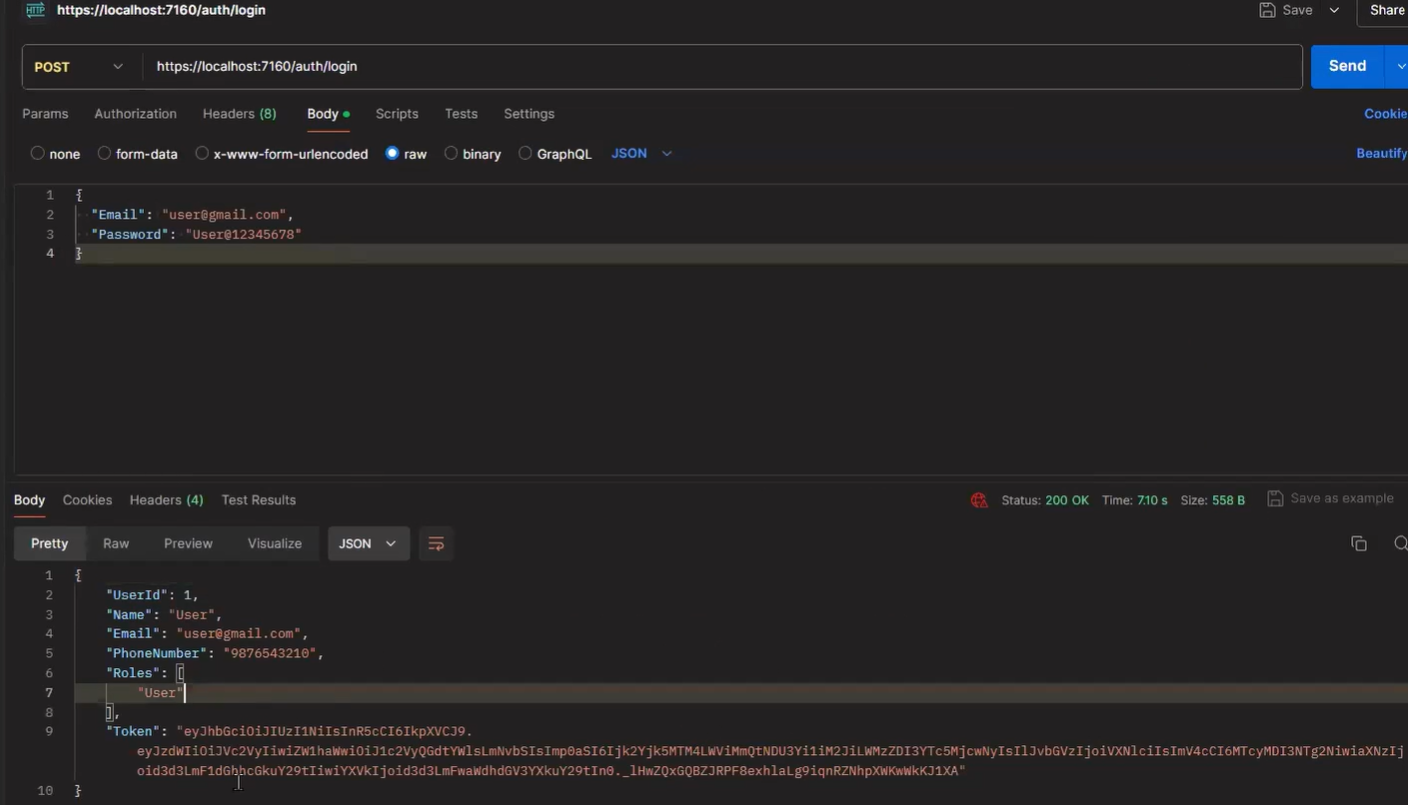
Description automatically generated

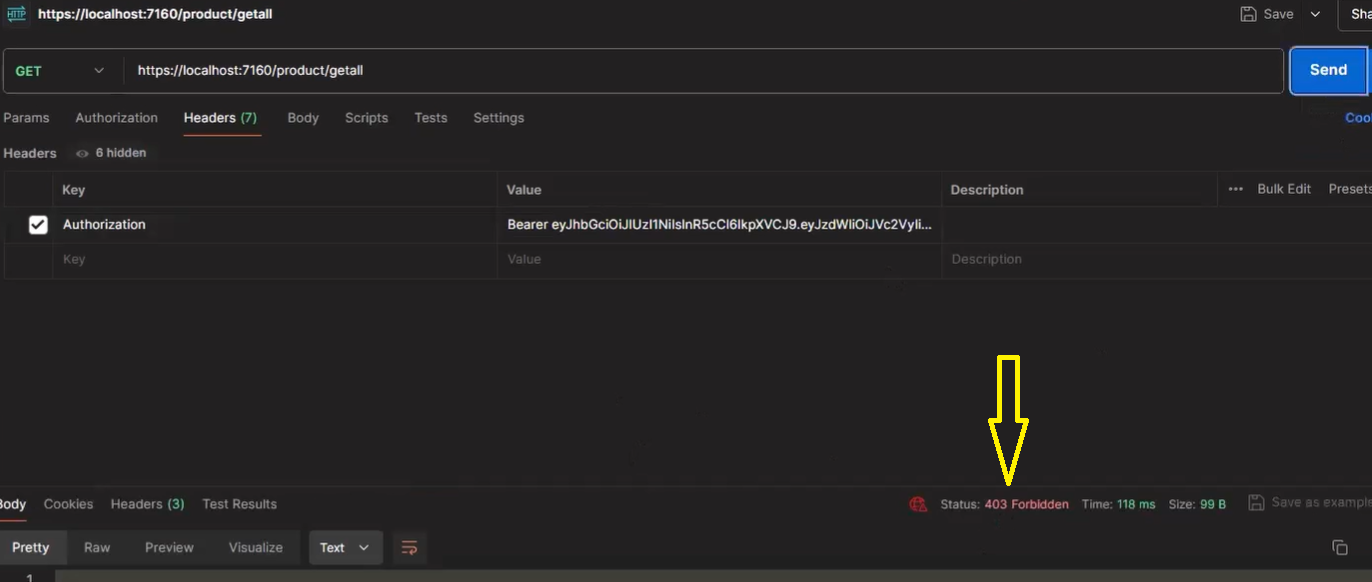
Protected path for Product ( Except Admin role , it protect the endpoint for all users)



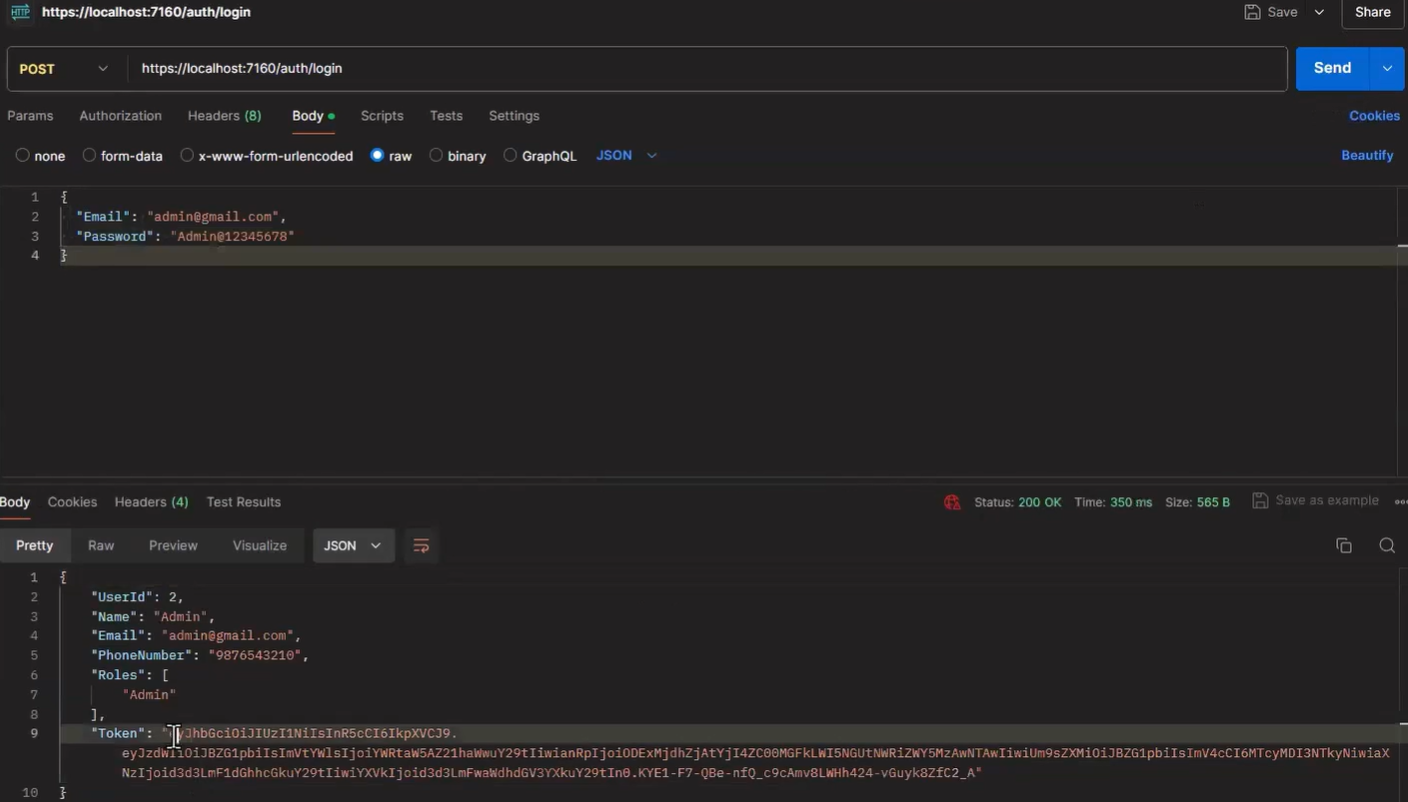


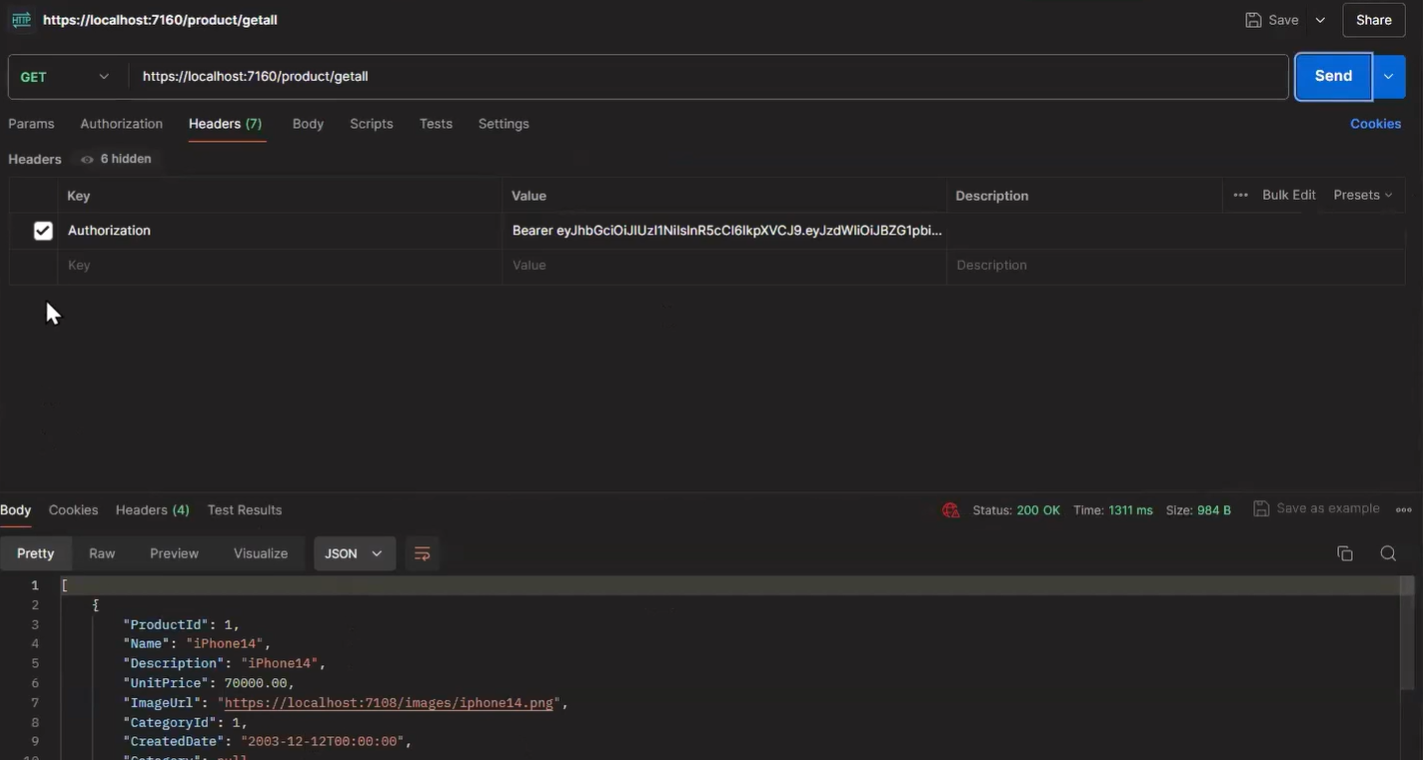
Check the flow through postman for User Role.





Check the flow through postman for Admin Role.





Suppose there are multiple Admin roles like Super Admin and Admin then the roles will be configure as below.

"RouteClaimsRequirement": {

"Roles": [ "Admin", "SuperAdmin" ]

},

1. Add the ProductModel.cs file in Model Folder and ProductService.cs file in HttpClients folder.
2. Register the ProductService in Program.cs file of SupportApp Project.

builder.Services.AddHttpClient<ProductService>(client =>

{

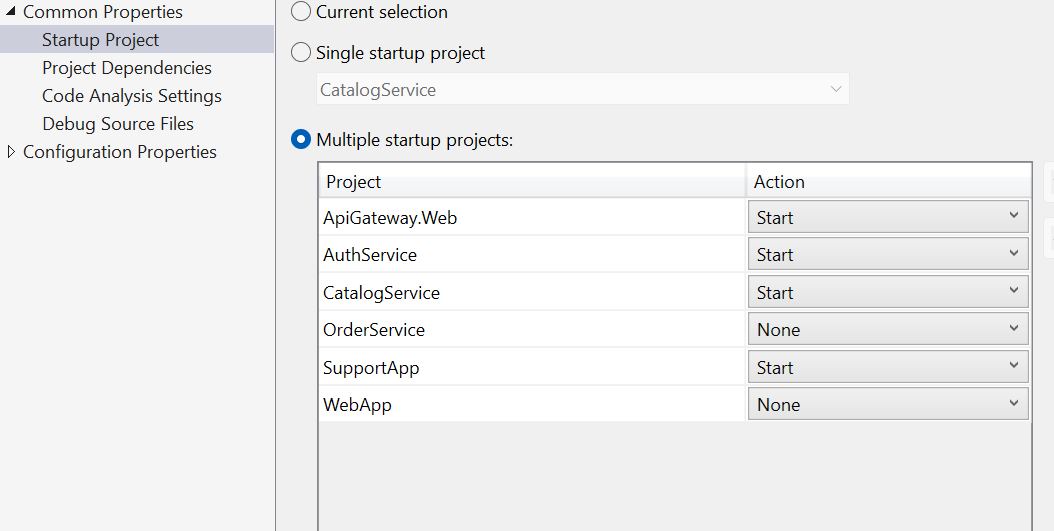
client.BaseAddress = new Uri(builder.Configuration["ApiGatewayAddress"]);

});

1. Create the Product controller and inject the ProductService and create the list razor view for Index action method.
2. Get the ClaimsPrincipal for the user associated with the executing action.(Area BaseController and Parent BaseController). AccountController is also inherited from BaseController so it can check the CurrentUser in Login Action Method for http Get method.
3. Inject **HttpContextAccessor** object in Program.cs file. For this have one built in service as below.

builder.Services.AddSingleton<IHttpContextAccessor, HttpContextAccessor>();

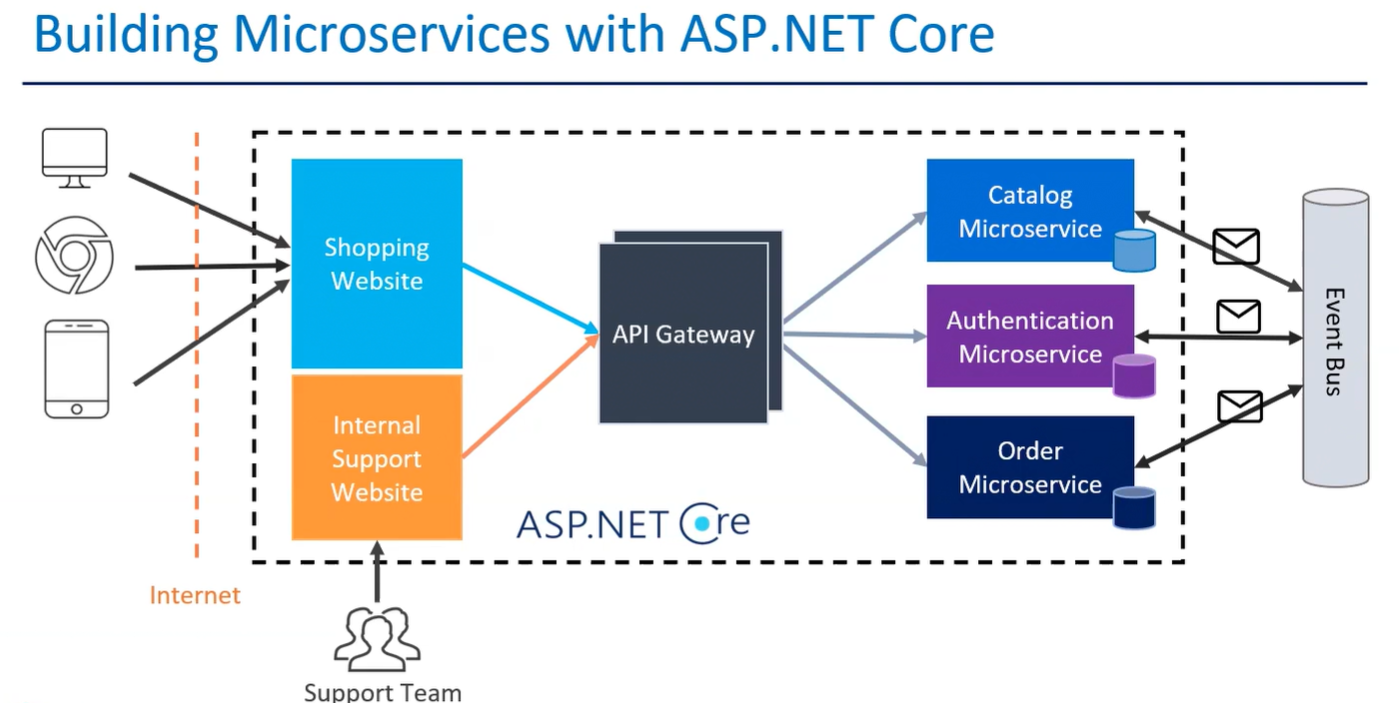
1. Access the HttpContextAccessor anywhere for reusability purpose in the application, we have to create the BaseService in the HttpClients Folder and inherit the ProductService from the BaseService.
2. Start up below the projects and login as admin.



A screenshot of a computer

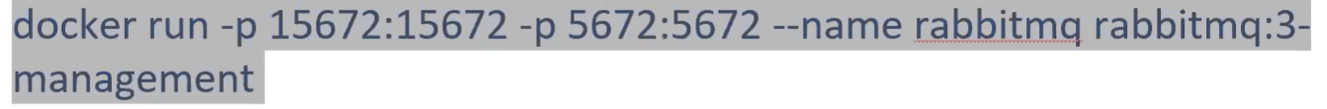
Description automatically generated

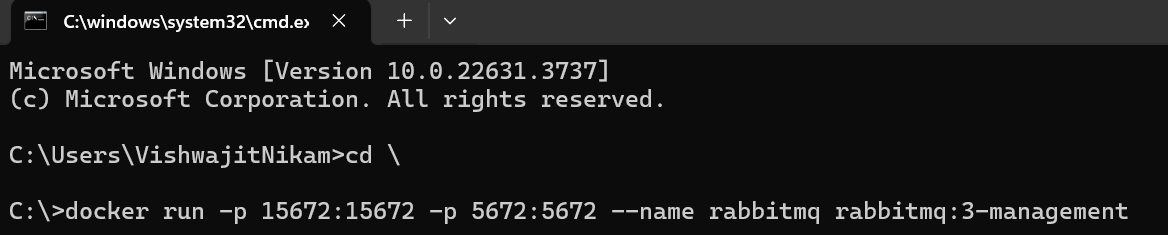
1. As per below diagram we developed till now the AuthService,CatalogService, WebApp, SupportApp.

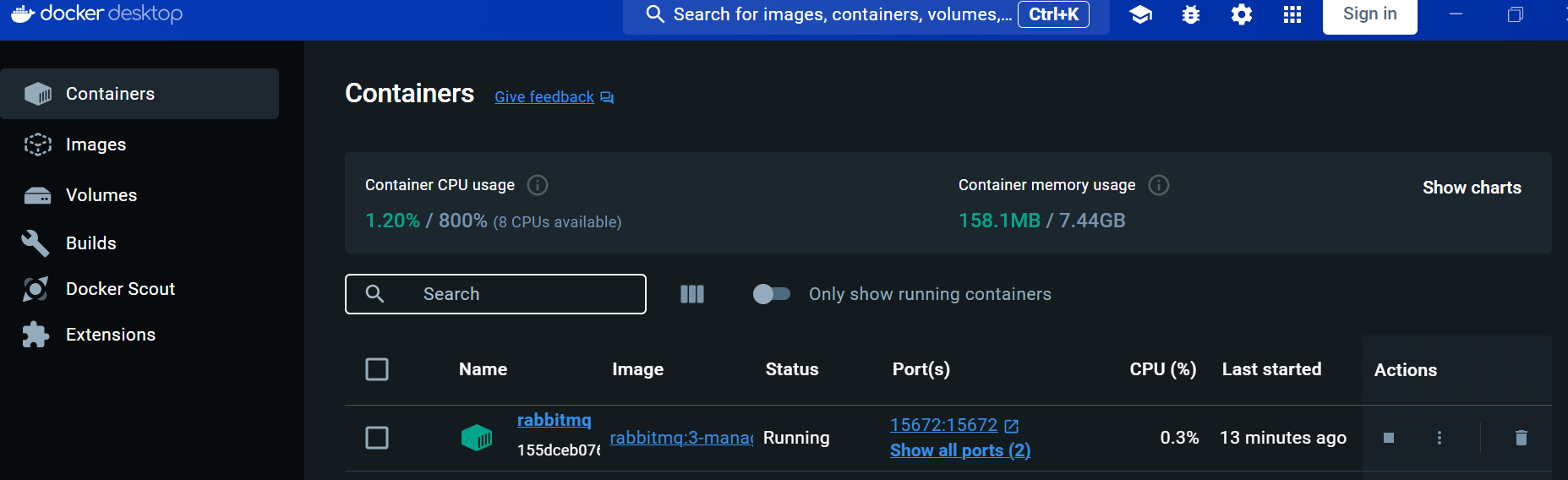


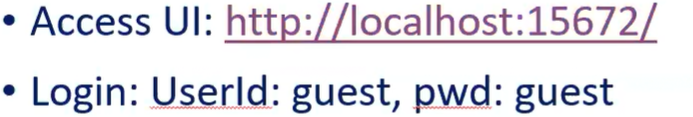
# **3. SECTION – 3 : Authentication and Decomposition Pattern**

1. Open the Docker Desktop from windows menu and run the below command in Command Prompt.









A screenshot of a computer

Description automatically generated