[**1. SECTION – 1 : Auth-Service-Integration-Using-API-Gateway** 1](#_Toc170980652)

# **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.

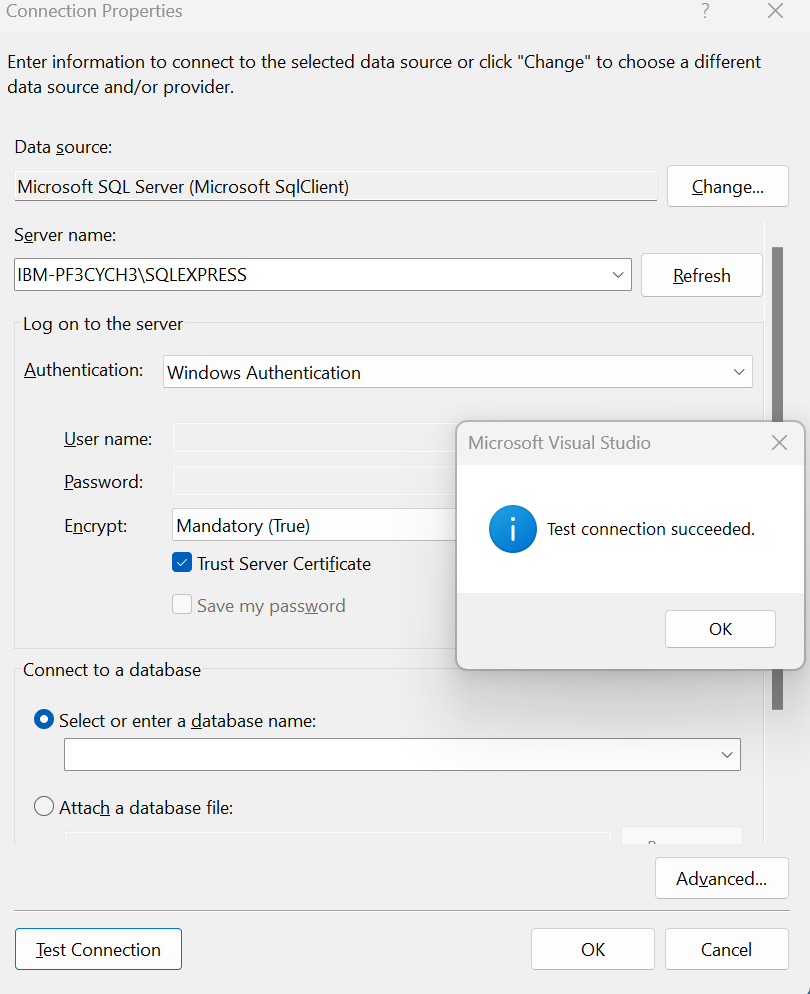
7 Install EF Core Power Tools (Visual Studio Extension)

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

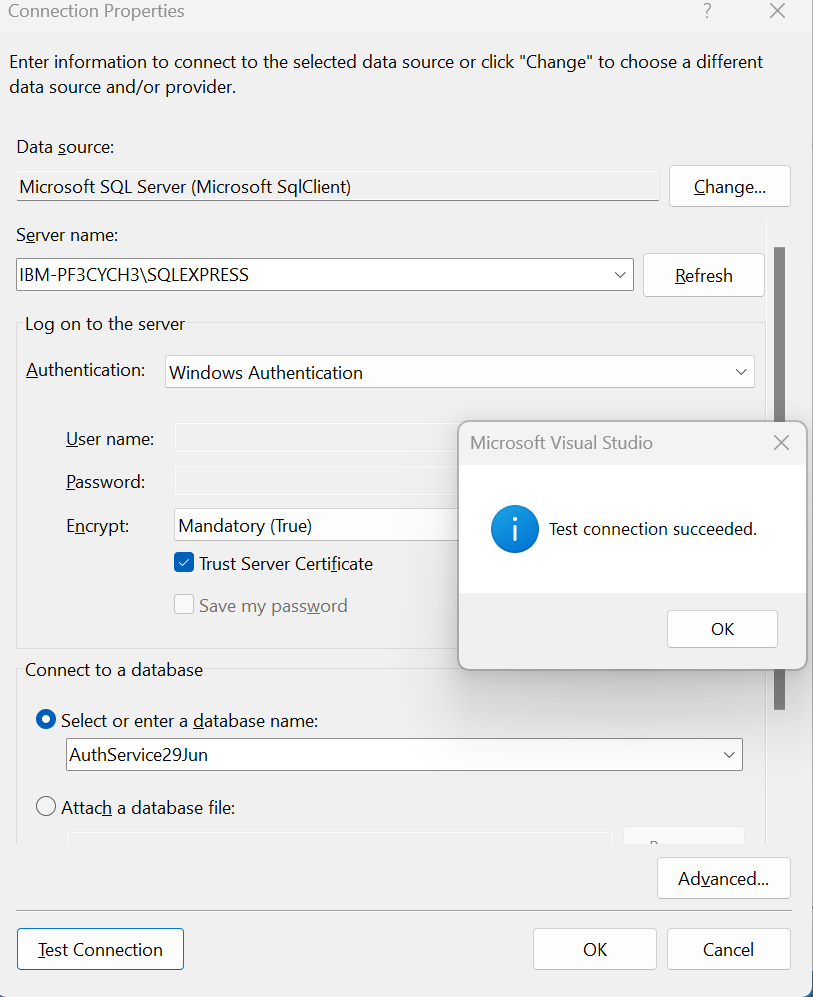
A screenshot of a computer error

Description automatically generated

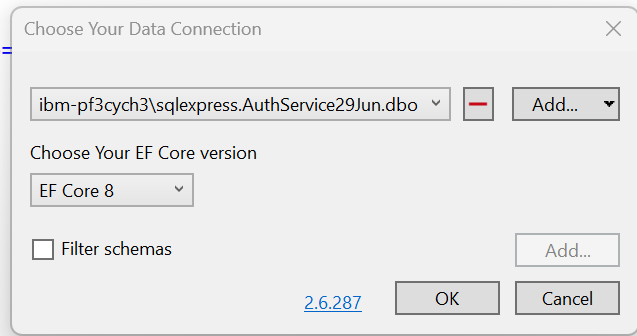
Click on Add 🡪 Add Database Connection 🡪 Check the below screen shot



Select the Database.



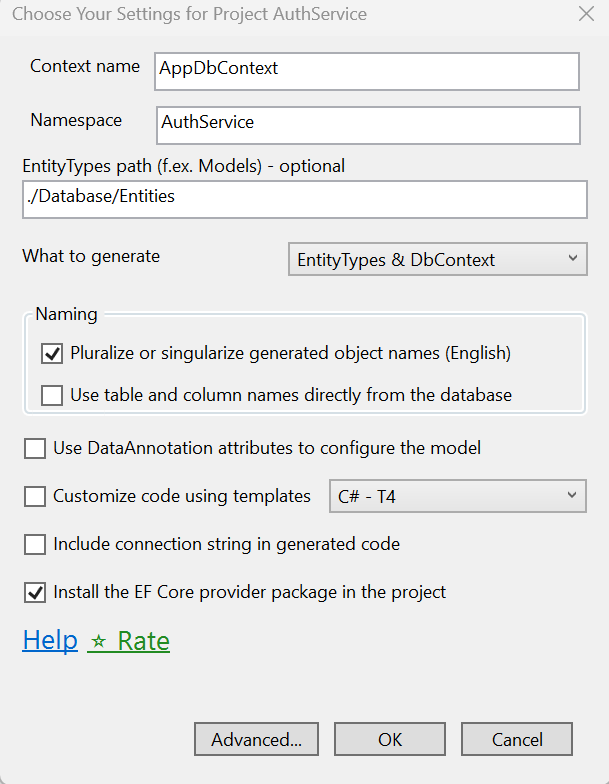
Click Ok 🡪 Click Ok ( Please check the below screen shot)



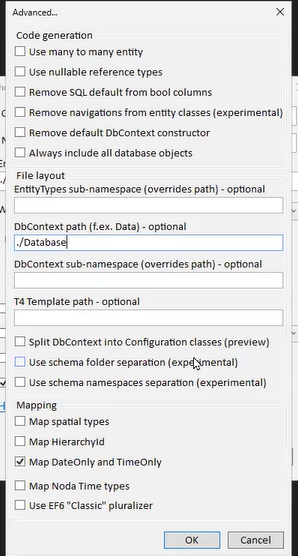
* Import all the tables from the below screenshot 🡪 Click Ok

A screenshot of a computer

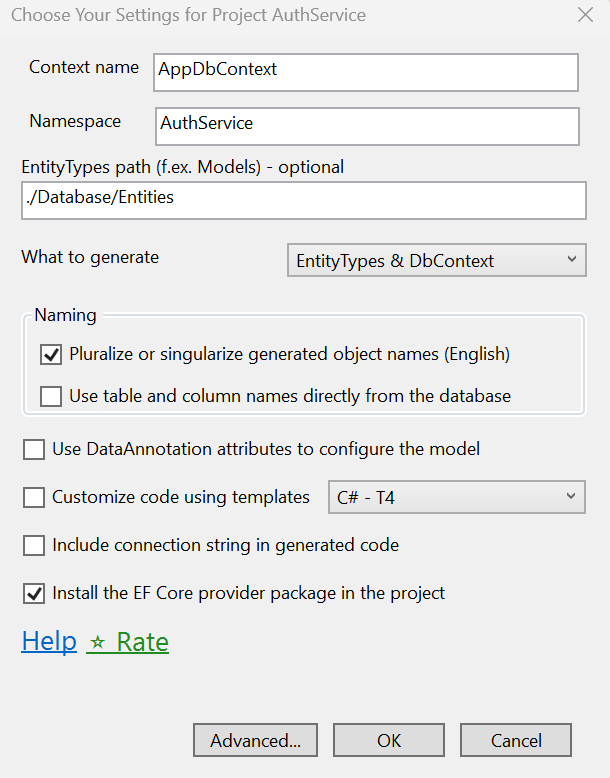
Description automatically generated



Click on Advanced 🡪 Make the below screenshot setting



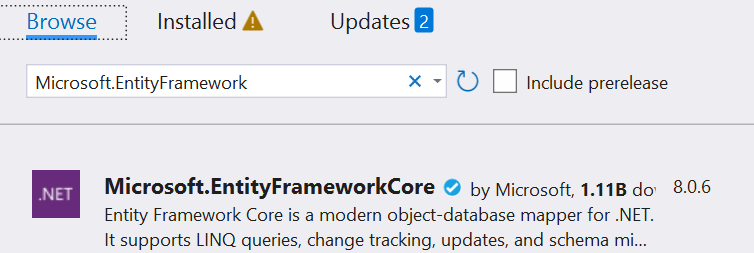
Click Ok 🡪 Click Ok of below window



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

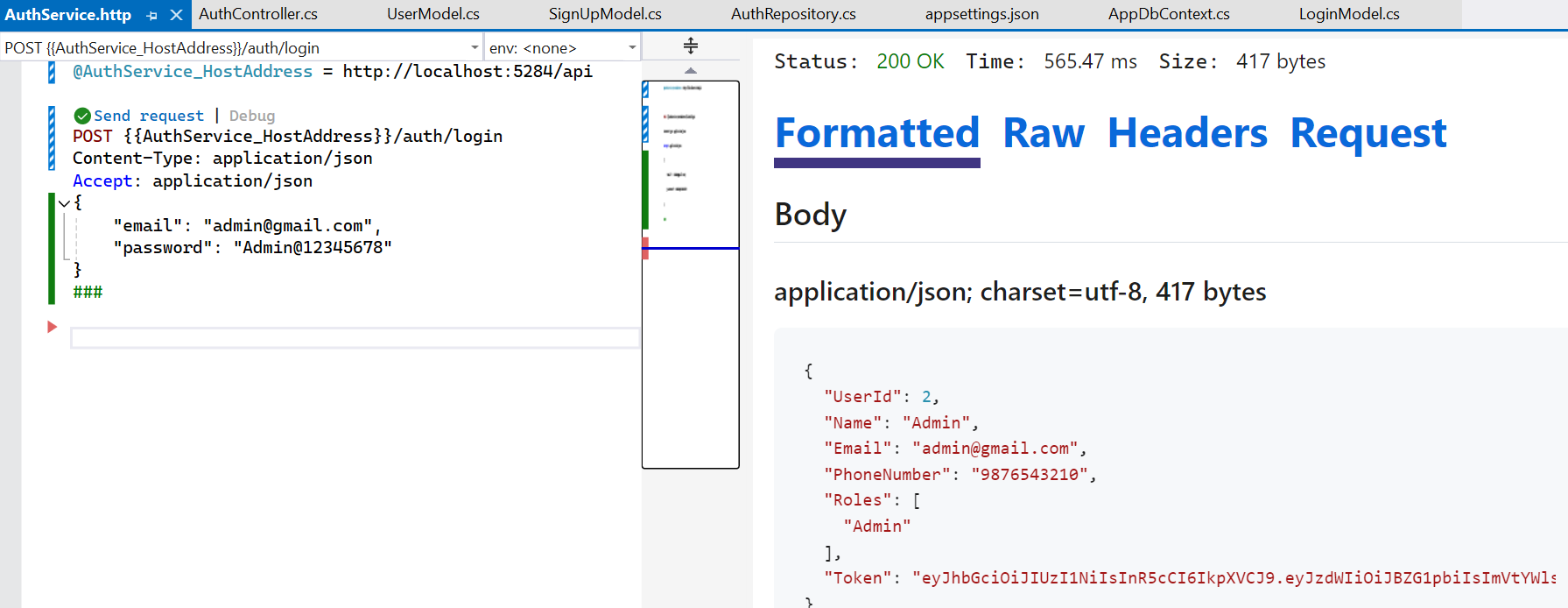
Description automatically generated

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

A screenshot of a computer

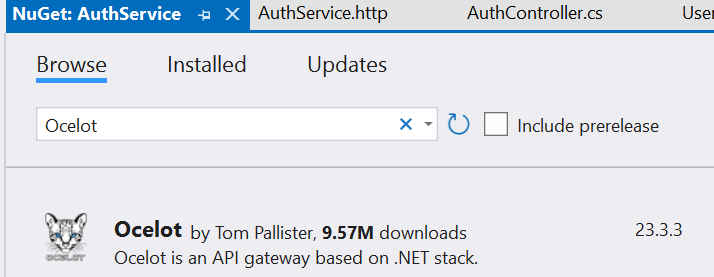
Description automatically generated

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");

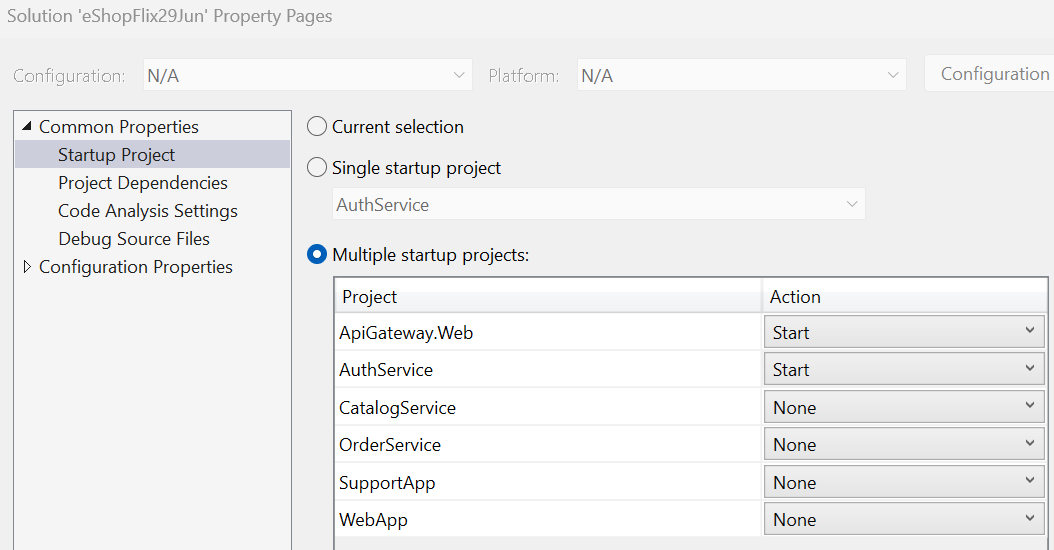
builder.Services.AddOcelot();

var app = builder.Build();

app.UseOcelot().Wait();

app.Run();

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

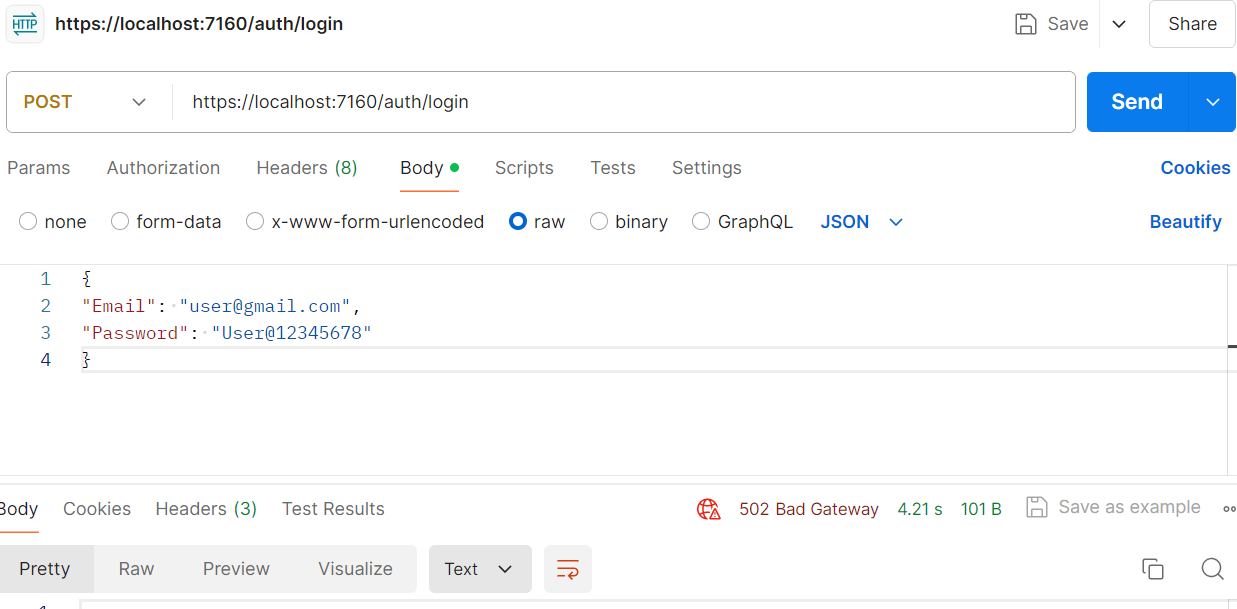


16. Download the Postman app from the below url and install it and login as google authentication.

<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.