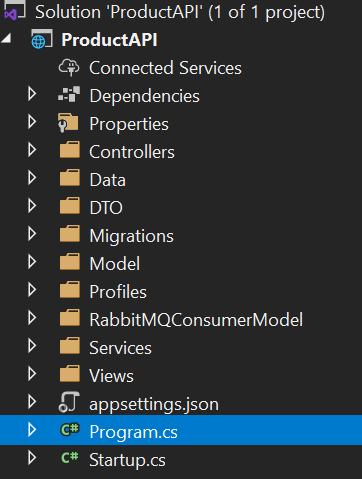
**Tools/Technologies:**

* Framework Used – ASP.NET Core
* Language Used – C#
* Service Discovery - Eureka for DotNet)
* API Gateway for implementing routing - Ocelot for DotNet
* Load balancer – Implemented in API Gateway
* Communication pattern- Implemented using Rabbit MQ
* Authentication mechanism to secure APIs- Token based Using Identity Server

I have created two microservices:

1. ProductAPI Microservices – It will be used to create a product and will be added to the Inventory
2. ProductDetailsAPI Microservices – It will be used to keep the details of each product like size/price/design. The product microservice can fetch the details of each product from product details Microservice

**Product API (Microservice - 1 )**



* In this microservice, I have created an entity product available in Model Folder of which a product table is being created in the database using Entity Framework Core.
* I have separated the data related code in the separate folder – DTO folder .Also I have created an DB Context file in the Data folder and created Migration to create a table and update database.
* In the Services Folder , the main operations are performed. I have created one Controller which will perform all the CRUD operations. There are 5 **Action Methods:**

GetProducts – Fetch all the products from the Database

GetProduct(int id) – Get a single product from the Database based on the ID passed

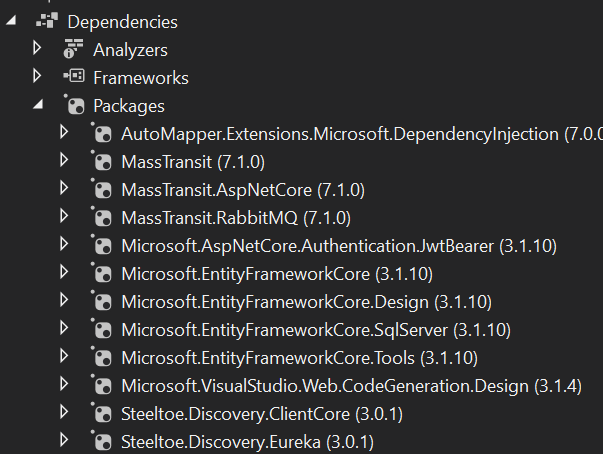
PostProduct(ProductDto product) – Post a product on to the Database

DeleteProduct(int id) – Delete a single product from the Database based on the ID passed

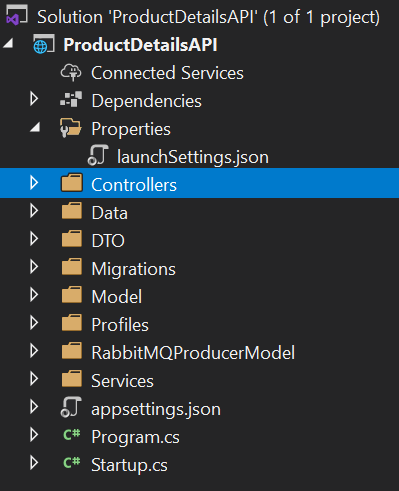
PutProduct(int? id, ProductDto product) – Update single product in the Database based on the ID passed

* All methods undergo Authentication and Authorization phase in which an Identity Server is used which provides token based authentication and claim based authorization.

.

**Nu Get Packages Used:** 

**Product Details API (Microservice-2 )**



* In this microservice, I have created an entity ProductDetail available in Model folder which a productDetails table is being created in the database using Entity Framework Core.
* I have separated the data related code in the separate folder - DTO. Also, I have created an DB Context file in the Data folder and created Migration to create a table and update database.
* I have created two Controllers –
* ProductDetailsController will perform all the CRUD operations. There are 3 **Action Methods:**

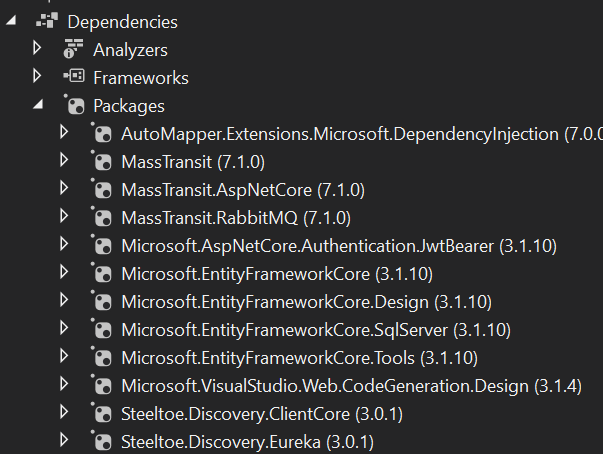
GetProductDetails – Fetch all the products from the Database

PostProduct(ProductDetailDto product) – Post a product on to the Database

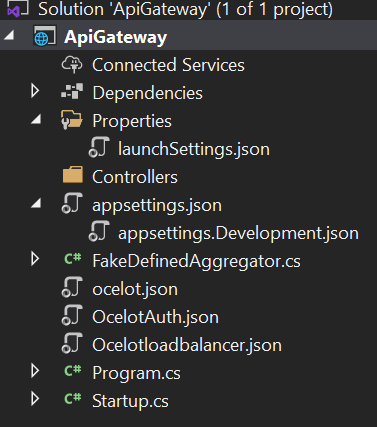
DeleteProduct(int id) – Delete a single product from the Database based on the ID passed

* TestRabbitController – It acts as producer to send the message to the Product Detail Service .
* All methods undergo Authentication and Authorization phase in which an Identity Server is used which provides token based authentication and claim based authorization.

**Nu Get Packages Used:**

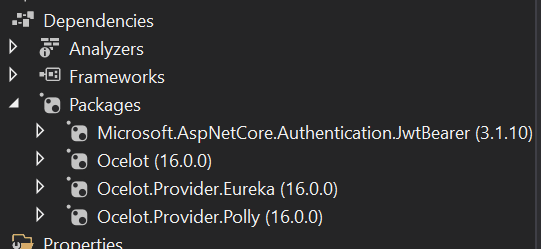


**API Gateway**

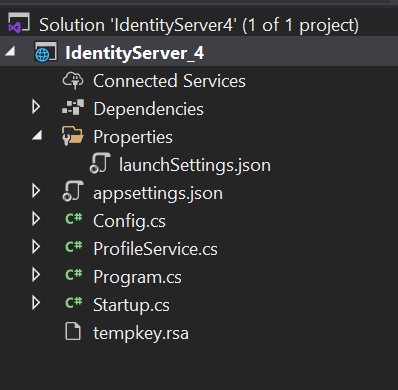


* It is implementing the API Gateway which is the communication between both the API’s as a single endpoint.
* In this, we are using Ocelot API Gateway as a single endpoint for both API and also implementing authorization and authentication for both API.
* OcelotAuth.json is used to configure the both API.
* In this API Gateway , I implemented the required Load balancer with Round Robin as an algorithm, Service discovery using Eureka Serve.
* For Security purpose, JWT bearer tokens are used to make gateway more secure and policy based authorization is performed.
* FakeDefinedAggregator is used for API composition.

**NuGet Packages Used:**



**IdentityServer4 ( IdentityServer)**

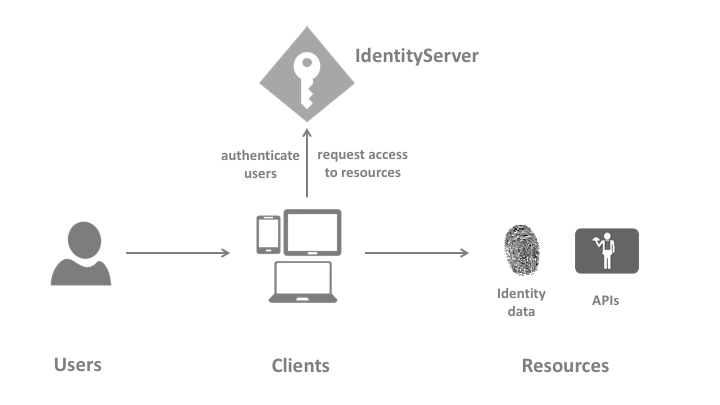


IdentityServer is an OpenID Connect provider - it implements the OpenID Connect and OAuth 2.0 protocol. It is a piece of software that issues security tokens to clients.

IdentityServer has a number of jobs and features - including:

* protect your resources
* authenticate users using a local account store or via an external identity provider
* provide session management and single sign-on
* manage and authenticate clients
* issue identity and access tokens to clients
* validate tokens

The following is the pictorial view of how Identity Server works:



**User**

A user is a human that is using a registered client to access resources.

**Client**

A client is a piece of software that requests tokens from IdentityServer - either for authenticating a user (requesting an identity token) or for accessing a resource (requesting an access token). A client must be first registered with IdentityServer before it can request tokens.

Examples for clients are web applications, native mobile or desktop applications, SPAs, server processes etc.

**Resources**

Resources are something you want to protect with IdentityServer - either identity data of your users, or APIs. Every resource has a unique name - and clients use this name to specify to which resources they want to get access to.

**Identity data**

Identity information (aka claims) about a user, e.g. name or email address.

**APIs**

APIs resources represent functionality a client wants to invoke - typically modelled as Web APIs, but not necessarily.

**Identity Token**

An identity token represents the outcome of an authentication process. It contains at a bare minimum an identifier for the user (called the sub aka subject claim) and information about how and when the user authenticated. It can contain additional identity data.

**Access Token**

An access token allows access to an API resource. Clients request access tokens and forward them to the API. Access tokens contain information about the client and the user (if present). APIs use that information to authorize access to their data.

Config.cs – contains all the configuration required.

It has a list of Clients – Clients

It defines several scopes – ApiScopes

It defines test users provided by the Identity server4 – TestUsers

**Use Case:**

Policy Based Authorization is performed in which two types of test users are created : Admin and User

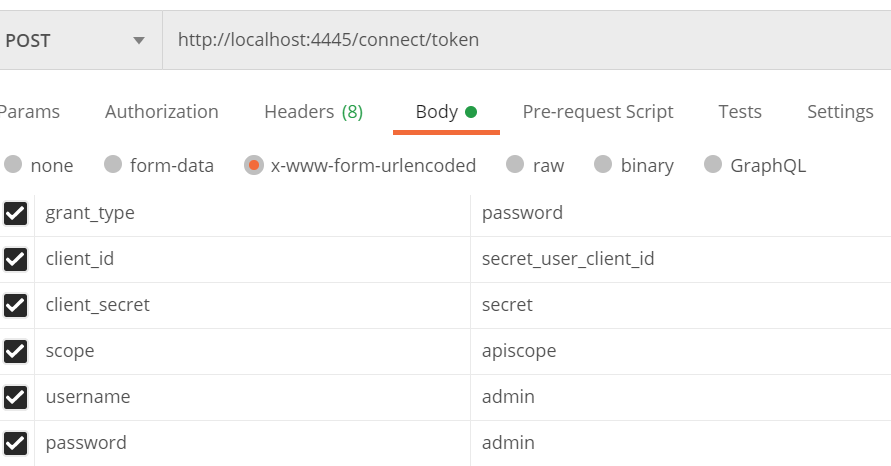
Admin has claim- CanUpdateData

User has claim – CanReadData

Scopes – Apisocope

1. **Admin can** **add/remove new products to inventory**

* Admin can generate the JWT token using the following information to the given URL: [**https://localhost:44350/connect/token**](https://localhost:44350/connect/token)



* Admin can use this token send to the given URL:

[**http://localhost:7000/api/product**](http://localhost:7000/api/product)

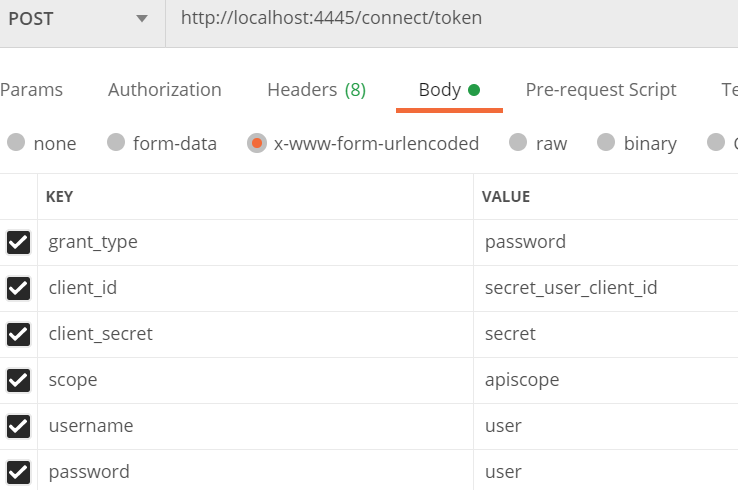
* Admin can add/remove new products to inventory

1. **Admin can add/remove product details like size/price/design**

* Admin can generate the JWT token using the following information to the given URL: [**https://localhost:44350/connect/token**](https://localhost:44350/connect/token)
* Admin can use this token send to the given URL: [**http://localhost:7000/api/productdetails**](http://localhost:7000/api/productdetails)
* Admin can add/remove product details like size/price/design

1. **User can view all the products list at any page**

* User can generate the JWT token using the following information to the given URL: [**https://localhost:44350/connect/token**](https://localhost:44350/connect/token)



* User can use this token send to the given URL:

[**http://localhost:7000/api/product**](http://localhost:7000/api/product) **,**

[**http://localhost:7000/api/productdetails**](http://localhost:7000/api/productdetails)

* User can view all the products and product details list at any page

1. **Product service can fetch detail about the product like size/price/design from another service (like price service or Product detail service)**

* For the Implementation of the message control , RabbitMQ and MassTransient is used.
* As ProductAPI is fetching the data from Product Details API.
* A OrderConsumer class is used which is Inherited from IConsumer which take Order class which is the duplicate class of ProductDetails entity in the Product Details API.
* TestRabbitController is created in the Product Details API for sending the data to Product API