How do we resolve the captive dependency?

Service would be running days and days. But the connections wouldn’t be open for a long time.

Service injection:

* Hostesd services are registered as Singleton
* Created a scope within the hostContext in the Program.cs - CreateHostBuilder

services.AddDbContext<ContosoContext>(options =>

                   {

                       options.EnableDetailedErrors();

                       options.UseSqlServer(hostContext.Configuration.GetConnectionString("DefaultConnection"));

                   });

                   services.AddScoped(typeof(IUnitOfWork), typeof(UnitOfWork));

Every day when the while loop runs, there would be a new connection to the DB.

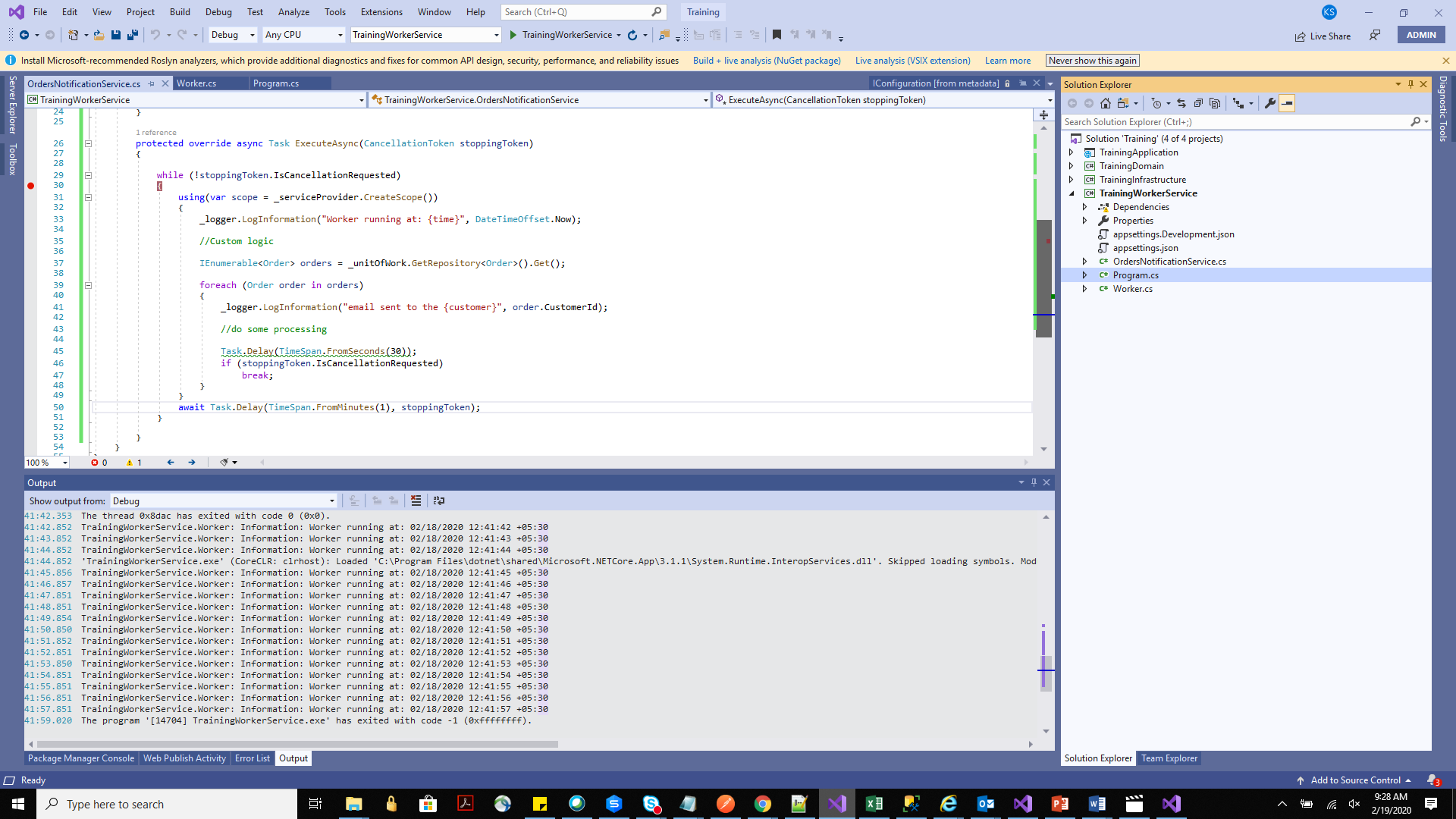
Instead of IUnitOfWork unitOfWork, now we are injecting the IserviceProvider.

And create a new scope in the while loop.

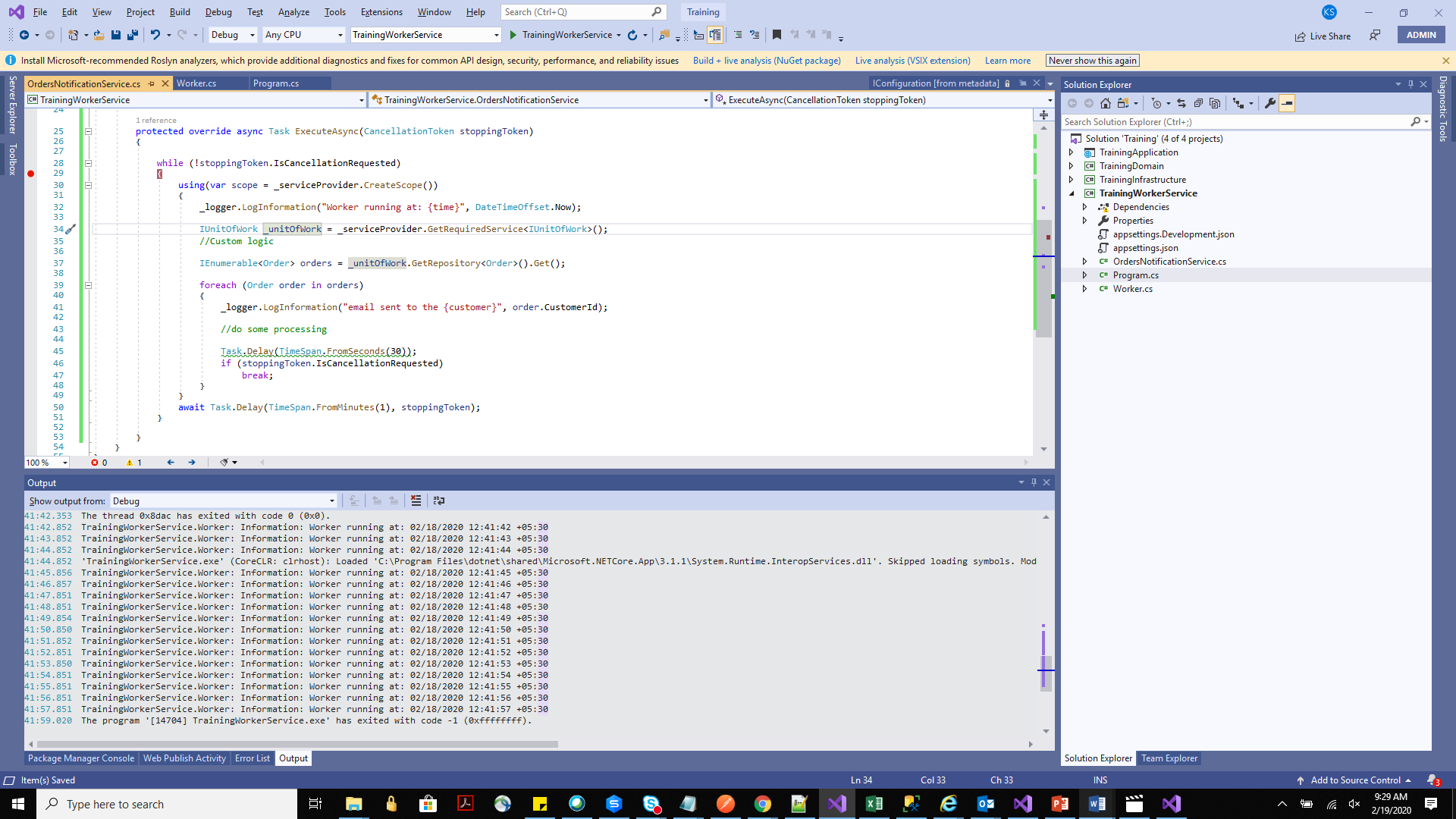
using(var scope = \_serviceProvider.CreateScope())

                {

                }



Also new instance of uoW everytime.



Issue 1 and 3 fixed.

Issue 2 -> add a delay of 1 seconds. And make it await. (1st line itself) above the while loop.

-----------logger service dependency not resolved yet.

**System.AggregateException:** 'Some services are not able to be constructed (Error while validating the service descriptor 'ServiceType: Microsoft.Extensions.Hosting.IHostedService Lifetime: Singleton ImplementationType: TrainingWorkerService.OrdersNotificationService': Unable to resolve service for type 'Microsoft.Extensions.Logging.ILogger' while attempting to activate 'TrainingWorkerService.OrdersNotificationService'.)'

InvalidOperationException: Unable to resolve service for type 'Microsoft.Extensions.Logging.ILogger' while attempting to activate 'TrainingWorkerService.OrdersNotificationService'.

**Authentication Protocols:**

* oAuth – applications can authenticate by using one another
* Suppose my Website wants to consume -> API exposed by amazon
* And This API is supposed to be used by an authorized entity.
* Today we have oAuth 2 is there. Oauth and oauth 2 are completely different.
* Different protocols both
* M to M protocol – m = machine.
* OIDC -> OpenID Connect is built on top of oAuth.
* So additional capabilities on OIDC.

User(resource owner) -> Website -> API (protected resource)

RO is the entity that can grant the access to a protected resource, typically the end user.

Client is the application requesting the access to the protected resource on behalf of the resource owner.

Resource server where the API is residing. (the protected resource).

Authorizing server is the server that authenticates the resource owner and issues Access token after getting proper authorization.

These tokens are also industry standard. JWT token.

Sometimes we may ourself have the capacity to authentication of the JWT token. It all depends on the capacity and capability of the protected resource. It may not need the Auth Server itself.

User Agent is the agent used bye the resource owner to interact with the client (eg; browser, or a native application, postman)

Eg: Azure active directory is a third party – auth server.

Auth0 is another one. Okta, Identity Server (IS). or You can implement it yourself.

There could be a gateway between the website and the Amazon API. Useful feature in the enterprise application (when multiple APIs are being called) . Important but its not the auth server. Token is still published by the auth server.

Access token is generated by the Auth server and is stolen while its being sent to the client. Using that access token anyone can access the protected resource. If it were in the session variable – then stolen.

AT if stolen – you can do malicious things till it expires. Banking and finance have lifetime as low as 15 min.

Protected resource is not going to check for anything but the access token.

Important recommendation – AT lifetime should be as short as possible. By default it is 1 hour. Max lifetime.

There is a refresh token – Allows an expired AT to be silently refreshed. Sometimes when you remember yourself on the client , then in the backend AT is still getting expired but for say 6 months (which is the validity for refresh token), AT gets refreshed in the backend.

Refresh token storing should be done with a greater precaution. Stealing AT will give someone the unautorized access for 15 min but Refresh token if stolen, malicious user can use the protected resource for 6 months.

Unless we ask for the RT, auth server will not create a new one.

ID token -> contains the information of the resource owner. It is a part of OIDC. Other 2 tokens are oAuth capabilities.

Authentication Cookie -> User access the website for the first time. First time the user is directed to the auth server, the user gives the credential and auth returns the tokens (can be multiple tokens – depending on how you have implemented it and how many tokens are you using), the tokens are retuned in the form of Auth cookies.

These tokens can also be stored in the cache – persistent cache, SQL DB.. etc or the cookies. That’s the responsibility of the client.

One use case -> auth server can be implemented in the website … but if on a different server, multiple websites can use this auth server.

Single Sign Out -> you need to get the user logged out from every client (phone, tablet, desktop, laptop)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Client Credentials -> read slide – good diagram.

Let code now for this authentication:

Training\_MVC\_Client – this is the project as a webservice

Lets add a auth server proj as well. If from scratch then 6 months. Its not just validating users and issuing tokens. It has to have revocation endpoint. What token and when for which user. For .net we have a open source auth server -> Identity server.

<http://quartzsystems.com/downloads/core3/identityserver.txt>

auth server has to trusted the client. So the trusted clients needs to be configured in the auth server.

Client uses Clientid and client secret to the auth server to get the token. One client can have an array of client secrets.

Also add services related to the identity server in startup.config

services.AddIdentityServer()

              .AddDeveloperSigningCredential()

              .AddInMemoryApiResources(IdentityServerConfig.GetApiResources())

              .AddInMemoryClients(IdentityServerConfig.GetClients());

And add the middleware components for authorization just after the routing middleware.

app.UseIdentityServer();

app.UseAuthorization();

The below needs to be added to the Order API project:

Microsoft.AspNetCore.Authentication.JwtBearer has nothing to do with Identity Server in the Auth project.

In the startup register the below services:

services.AddAuthorization();

        services.AddAuthentication("Bearer")

            .AddJwtBearer("Bearer", options =>

            {

                options.Authority = "https://localhost:44378"; // this should match with the host of the auth server.

                options.Audience = "orders.api";

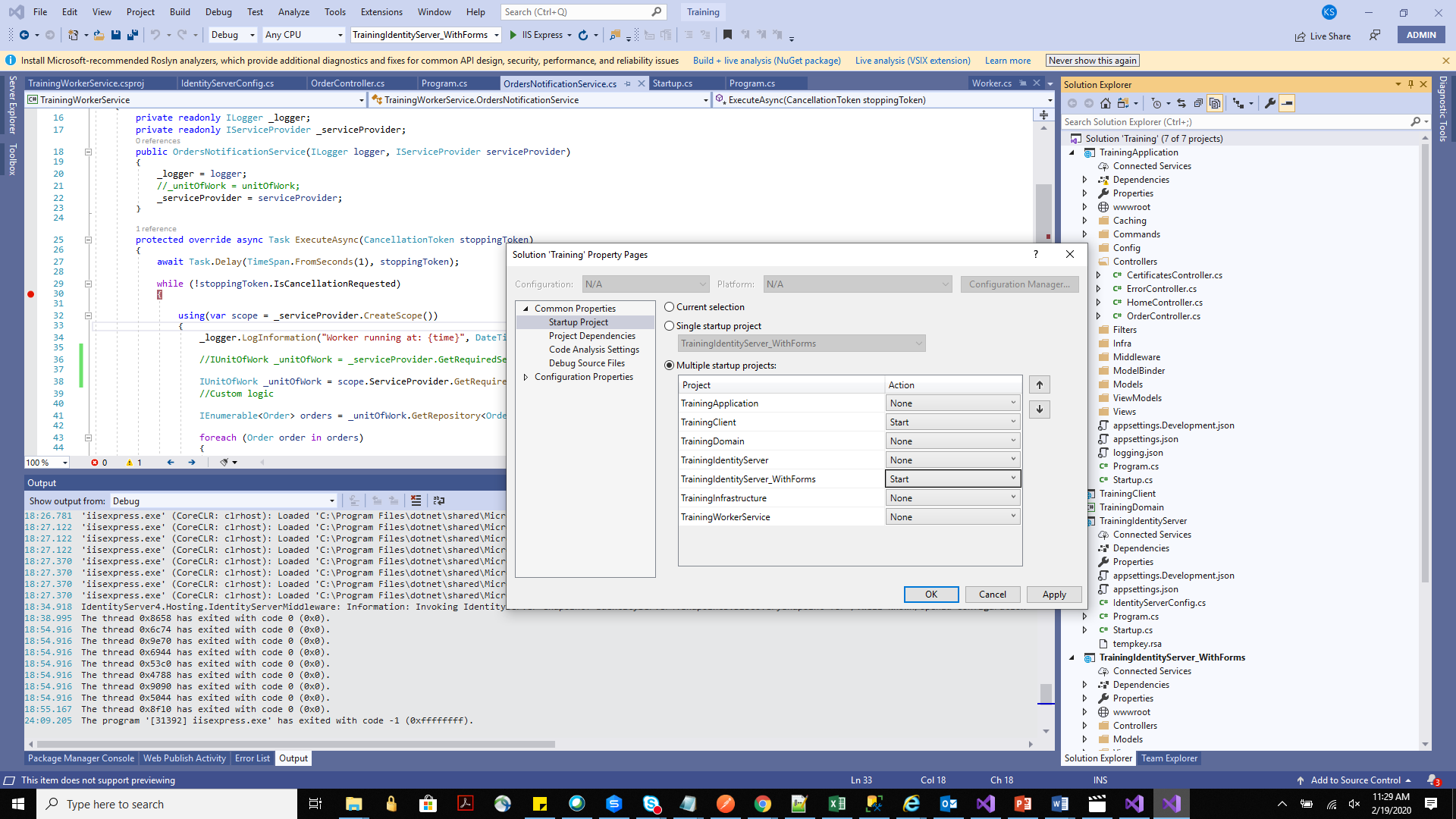
            });

JWT bearer can go ahead and verify any token issued from Okta or any other auth server as well.

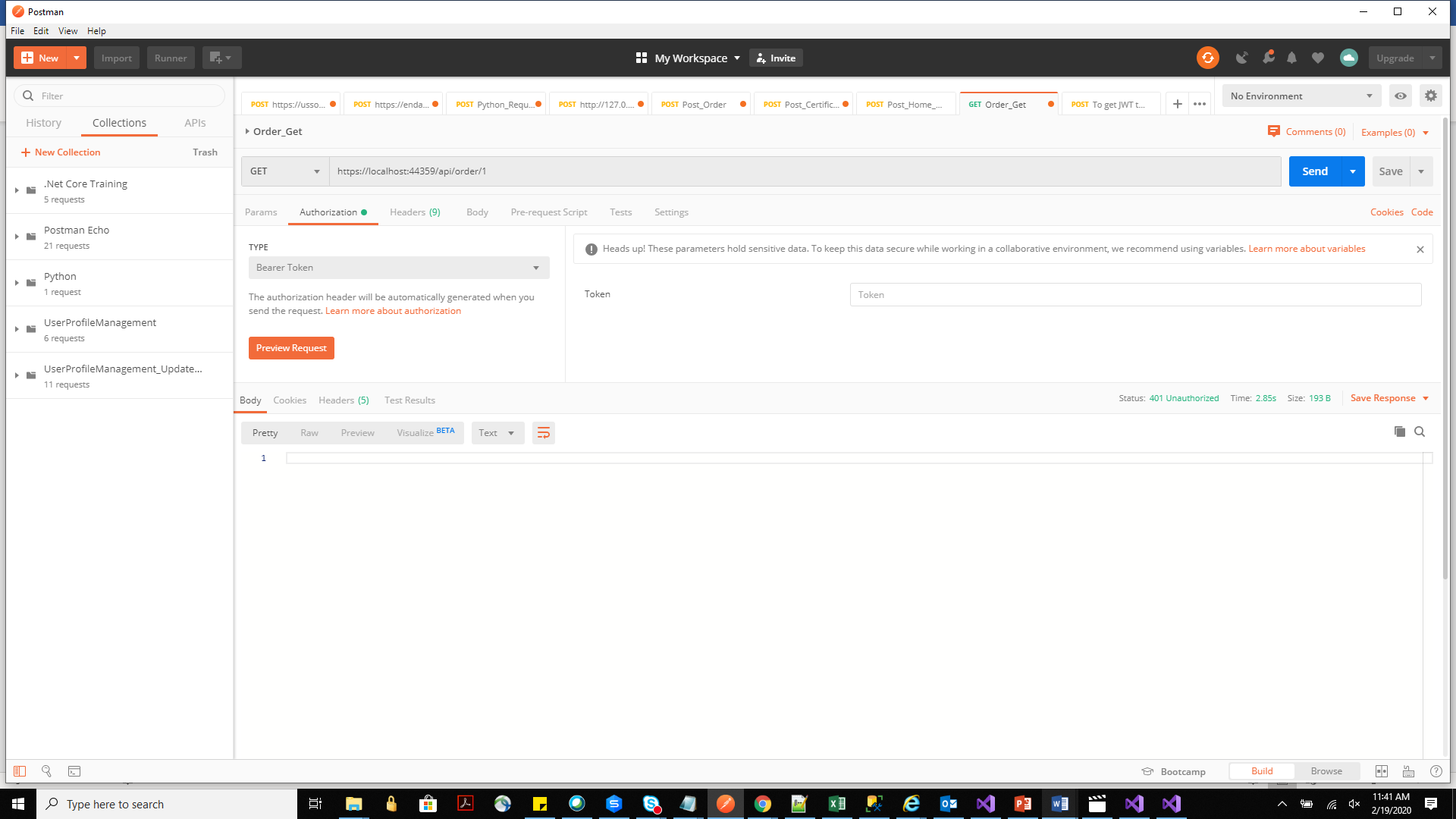
The Protected resource (in our case API) needn’t go to auth server to identify the token is valid or not. It can inspect it by itself.

Instead of JWT you can use reference token as well

2 of the projects should be start up.

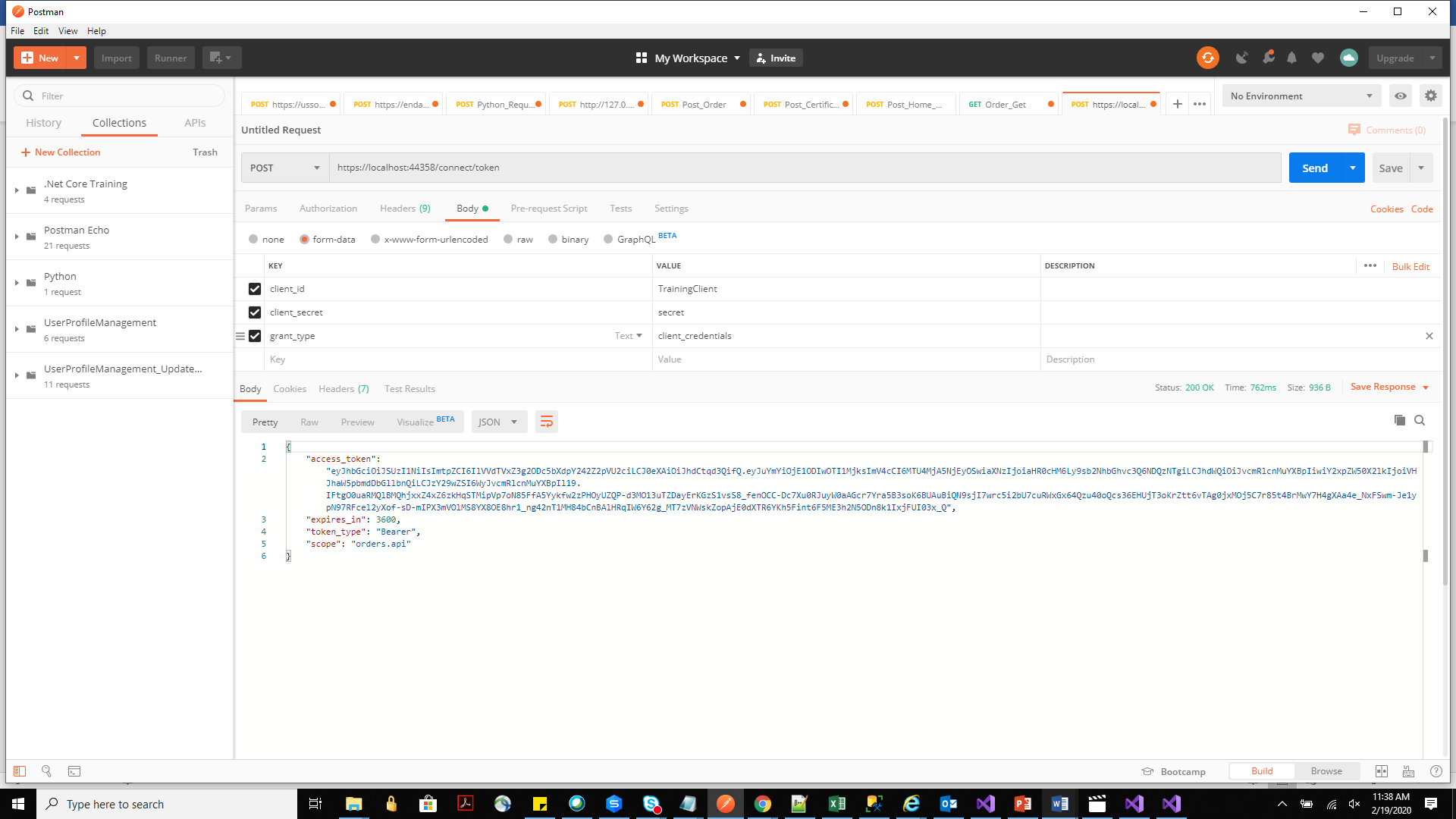


Below gives 401 and we need the authorization. -



<https://localhost:44358/connect/token> - hit this to get the token. Host is the auth server host. Make a post request and tell the flow (whether implicit or client) using grant\_type.

And it returns the jWT bearer token.



{

    "access\_token": "eyJhbGciOiJSUzI1NiIsImtpZCI6IlVVdTVxZ3g2ODc5bXdpY242Z2pVU2ciLCJ0eXAiOiJhdCtqd3QifQ.eyJuYmYiOjE1ODIwOTI1MjksImV4cCI6MTU4MjA5NjEyOSwiaXNzIjoiaHR0cHM6Ly9sb2NhbGhvc3Q6NDQzNTgiLCJhdWQiOiJvcmRlcnMuYXBpIiwiY2xpZW50X2lkIjoiVHJhaW5pbmdDbGllbnQiLCJzY29wZSI6WyJvcmRlcnMuYXBpIl19.IFtgO0uaRMQlBMQhjxxZ4xZ6zkHqSTMipVp7oN85FfA5Yykfw2zPHOyUZQP-d3MO13uTZDayErKGzS1vsS8\_fenOCC-Dc7Xu0RJuyW0aAGcr7Yra5B3soK6BUAuBiQN9sjI7wrc5i2bU7cuRWxGx64Qzu40oQcs36EHUjT3oKrZtt6vTAg0jxMOj5C7r85t4BrMwY7H4gXAa4e\_NxFSwm-Je1ypN97RFcel2yXof-sD-mIPX3mVOlMS8YX8OE8hr1\_ng42nT1MH84bCnBAlHRqIW6Y62g\_MT7zVNWskZopAjE0dXTR6YKh5Fint6F5ME3h2N5ODn8k1IxjFUI03x\_Q",

    "expires\_in": 3600,

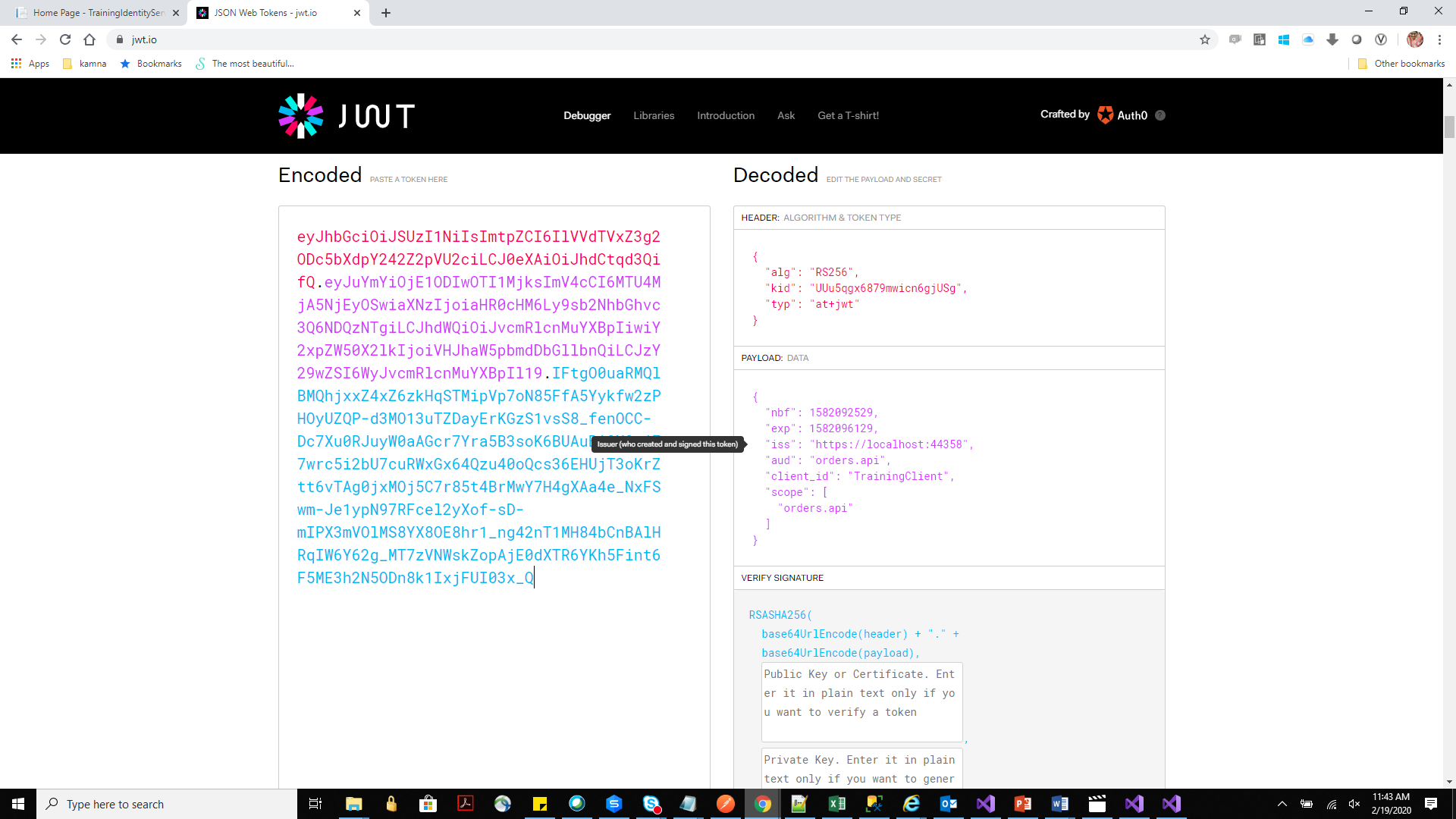
    "token\_type": "Bearer",

    "scope": "orders.api"

}

Still 401 even after passing with token.

Jwt.io to validate the token.



See the iss 44358 but I had mentioned 44378 in the API controller.

Token valid for 1 hour so use the same token.

The above token doesn’t contain the resource owner information.

Still issue – mistake is not adding use authentication in the pipeline in the order API startup.

Also Order controller needs the [Authorize] attribute.

Now we are going to learn granty-type -> Resource owner.

Previously we learn client-credential.

What if the client and the resource owner is a different entity. User accessing a website and then that website in turn calls oAuth server to get the token.

See slide – beautiful diagram.

IdentityServer4.AccessTokenValidation now on the web api project. Enough to validate the access token.

.AddIdentityServerAuthentication("Bearer", options =>

{

options.Authority = "https://localhost:44378";

options.ApiName = "orders.api";

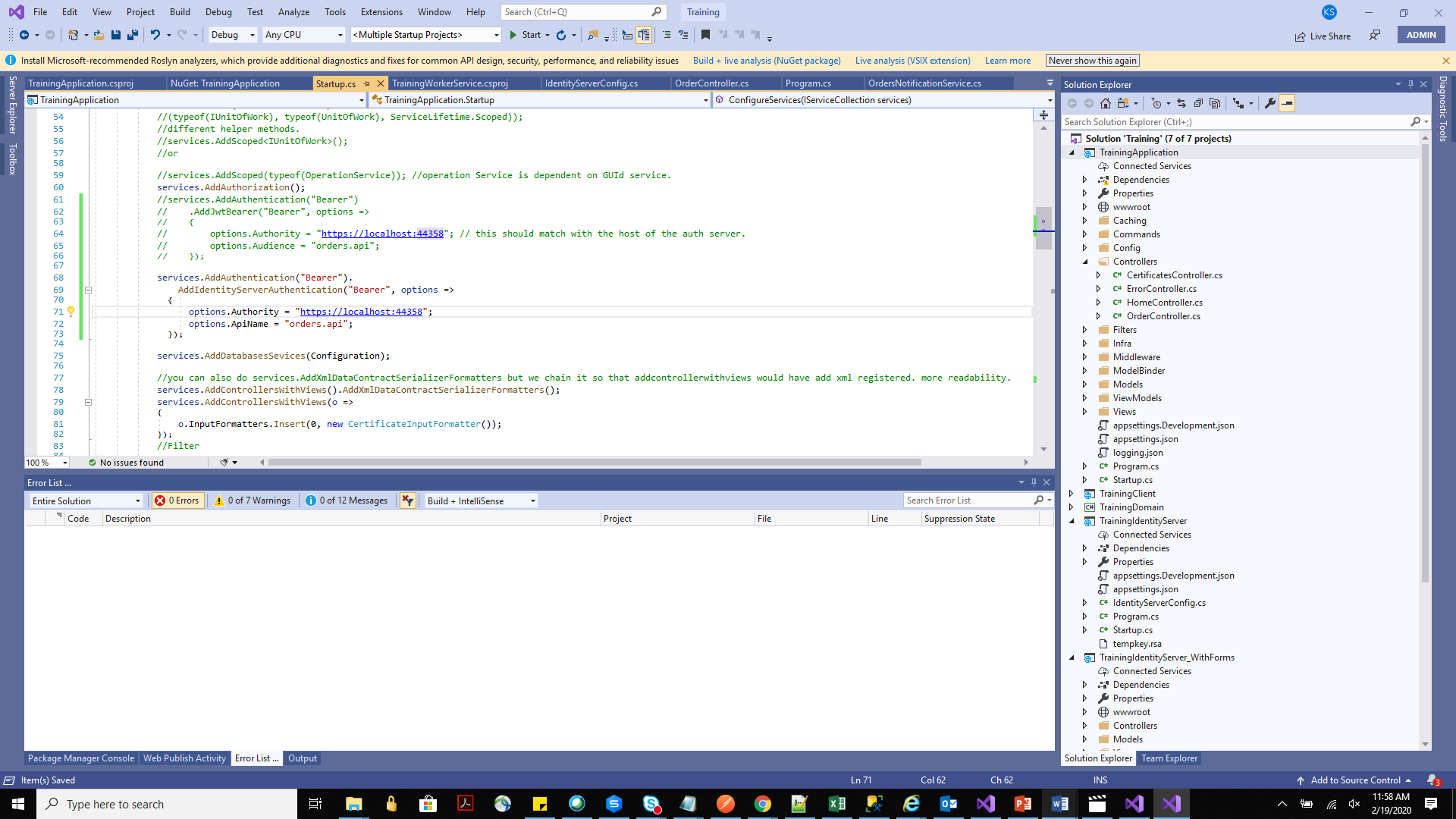
})

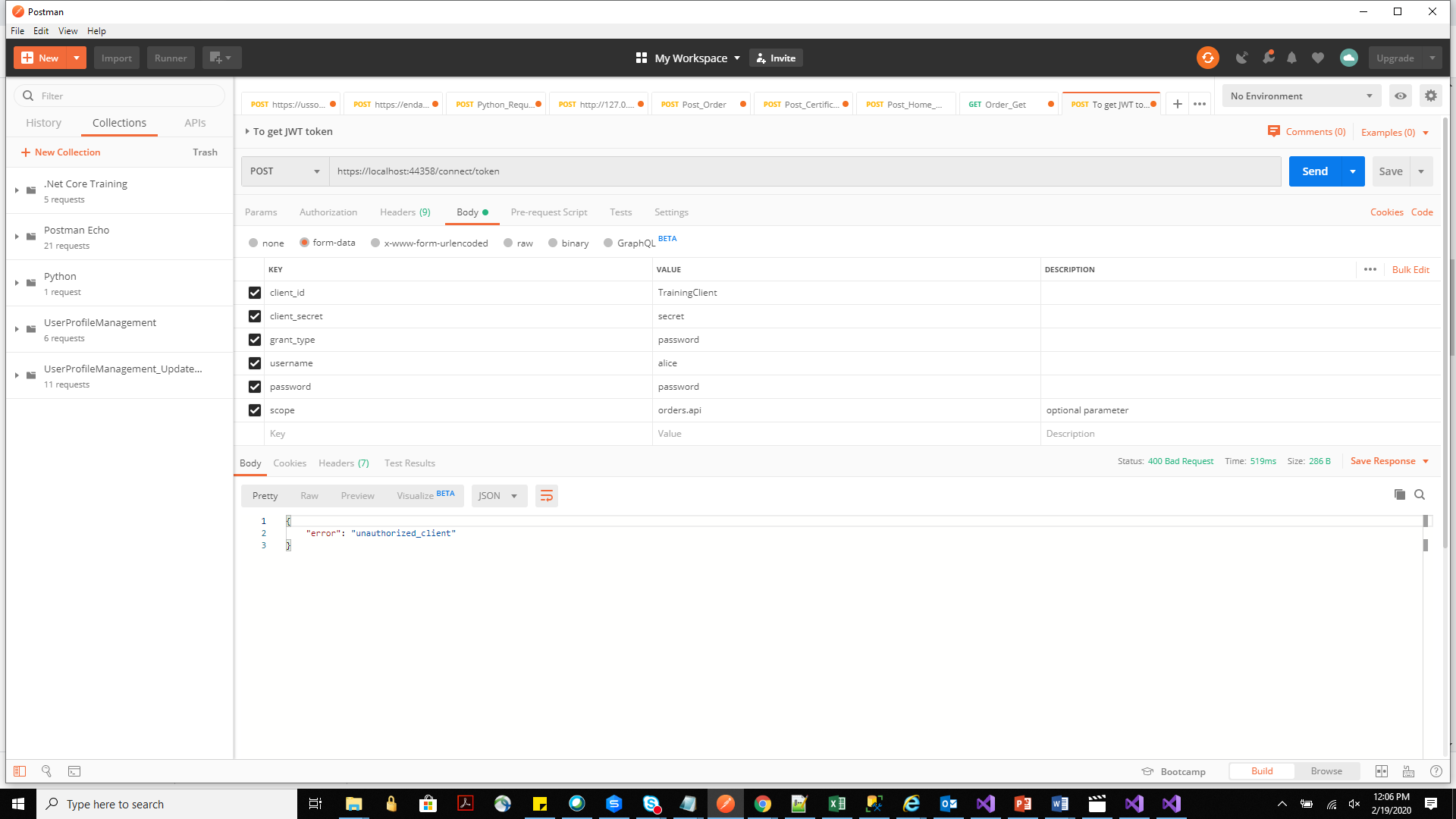
This instead of the earlier service for client credential. See the snippet below.

Previously there was no round trip to the identity server. Now there is.

Lets add some dummy users as well to the IdentityServerConfig of the auth server project.

.AddTestUsers(IdentityServerConfig.GetUsers());  to be added to the startup of the auth server proj.

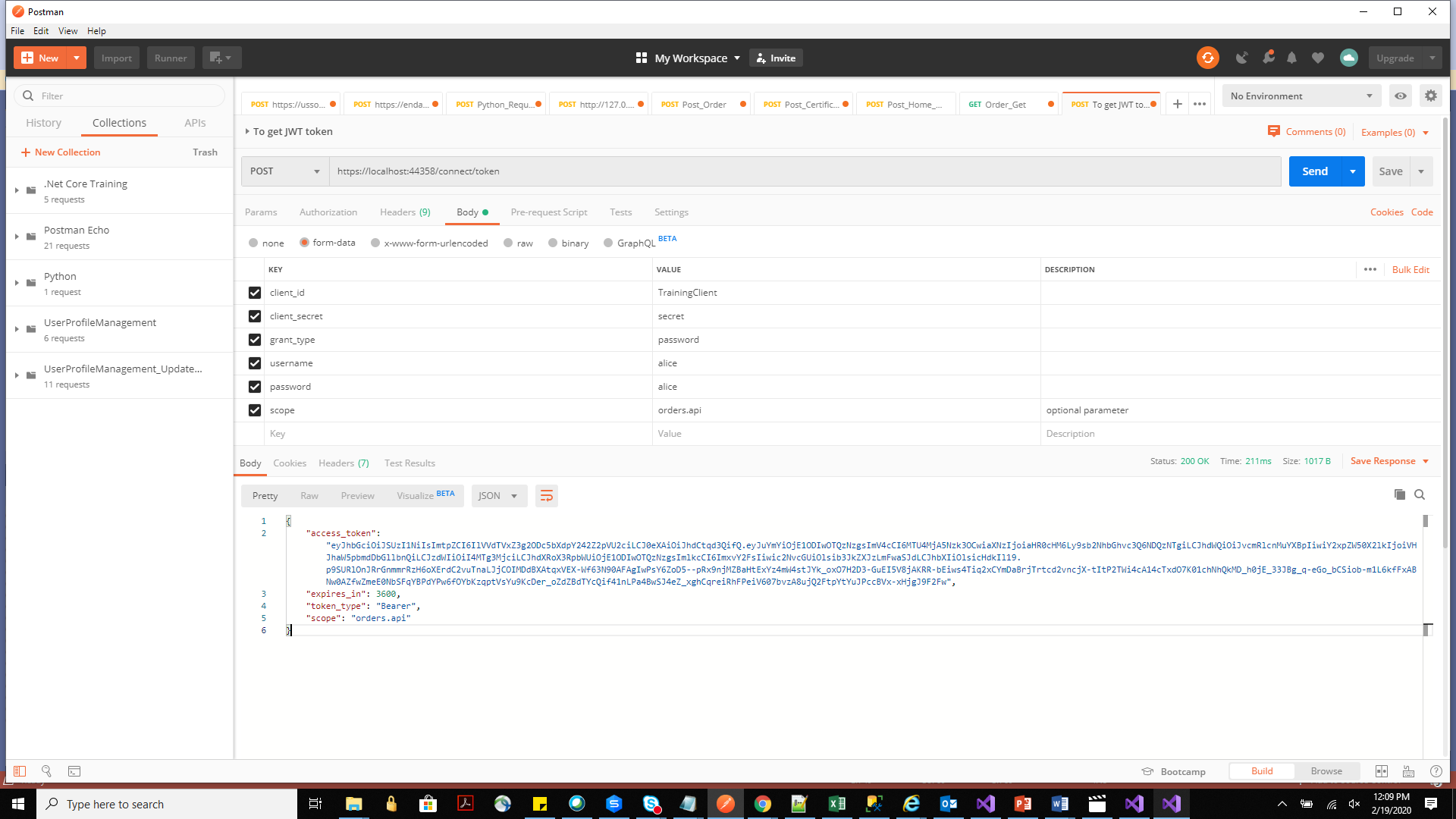




Scope is an optional parameter.

AllowedGrantTypes = GrantTypes.ResourceOwnerPassword,

Previously it was client\_credential. You can add as mmany grant types



(silly mistake .. password in the form data should be as per the pwd for the user in the indentiserverconfig cs file). Inspect this token and compare with the one from the client credential.

This grant type is not a very popular one – as the client needs the access to the client’s credential.

See postman asked for it.. bad practice.

Website asked for the username and pwd. And gives it to the auth server. If valid credential then it gets the auth token.

Lets see the next flow – much better way. – highly secure – most popular oAuth flow – Authorization code flow

Next one is client credential popularity wise. Then implicit… in the last is the resource owner pwd.

See the slide – beautiful diagram.

Login form is coming from the Auth server. Website is not asking for the client credential.

Like in myntra is you are trying to login, you can choose to login via facebook, so auth server is fb. And myntra doesn’t ask for fb credential, instead directs us to the auth server of FB and there you do the authentication.

Identity server comes with a set of templates. Run the below for the auth server project.

dotnet new -i identityserver4.templates

dotnet new is4ui -n IS4UI

New folder named IS4UI – contains files --- remove these folders from the folder to the main folder.

And permanently delete the folder.

Myntra had to be registered with facebook and then Myntra becomes the redirect UR Iwhen the authorization code is retuned to. Registered under the new client of the auth server.

The below code in the startup.configure.

.AddInMemoryIdentityResources(IdentityServerConfig.GetIdentityResources());

Understand the below snippet,

#### Authentication settings in web client

JwtSecurityTokenHandler.DefaultMapInboundClaims = false;

//We are using a cookie to locally sign-in the user (via "Cookies" as the DefaultScheme),

//and we set the DefaultChallengeScheme to oidc because when we need the user to login, we will be using the OpenID Connect protocol.

services.AddAuthentication(options =>

{

options.DefaultScheme = CookieAuthenticationDefaults.AuthenticationScheme;

options.DefaultChallengeScheme = OpenIdConnectDefaults.AuthenticationScheme;

})

//add the handler that can process cookies.

.AddCookie(CookieAuthenticationDefaults.AuthenticationScheme)

//configure the handler that perform the OpenID Connect protocol.

.AddOpenIdConnect(OpenIdConnectDefaults.AuthenticationScheme, options =>

{

//The Authority indicates where the trusted token service is located.

options.Authority = "https://localhost:44378";

//identify this client via the ClientId and the ClientSecret.

options.ClientId = "MVC Client";

options.ClientSecret = "secret";

options.ResponseType = "code";

//whether id and access tokens should be saved in the browser (in the auth cookie)

options.SaveTokens = true;

});

OpenIdConnectDefaults – takes care of the round trip for picking the auth code and then combining it with the client id and credential to get the tokens.

You can decide where to save the tokens (ID token and the other one) Here we are doing on the auth cookie.

You can even use the in-memory cache to save the tokens.