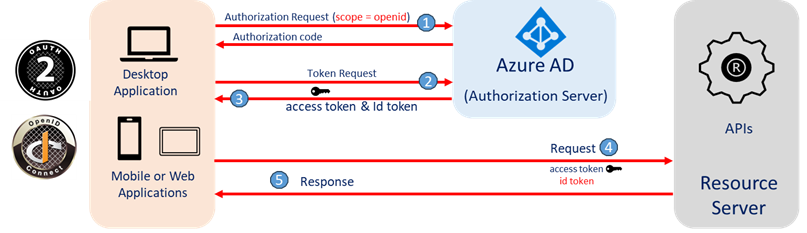
**Part 5: OpenID Connect (OIDC) with Azure AD**



OpenID Connect or OIDC is a protocol which provides identity as a service. OpenID Connect is built directly on OAuth 2.0 and all concepts, flows, endpoints, and tokens of OAuth 2.0 also apply in the context of OpenID Connect. It allows end-user data in the form of claims or attributes to be passed in a secure manner from a provider to a client. OpenID Connect delivers claims of the end-user either as a RESTful UserInfo API endpoint or as a standardised identity token.

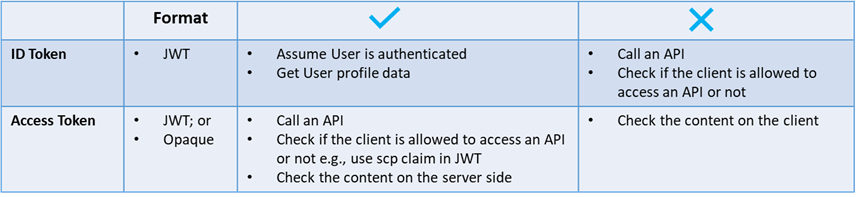
Identity token is issued together with an access and optionally a refresh token. The identity token contains the claims and attributes of the end-user in an encoded form. This design allows the third-party app to access the claims offline. The app only needs the identity token and the cryptographic keys to validate the token.

This token is provided by the token endpoint in a JSON web token (JWT) format. JWT tokens are created by putting mandatory and optional claims into a JSON object, Base64URL-encoding this JSON object, and signing the resulting string using one of the cryptographic algorithms defined in the JWT standard.

The resource server protects the access to the API. Any access to these APIs needs to be authorized using an access token. The access token is meant to be read and validated by the resource server and verifies the access token before granting access.

The resource server verifies the access token before granting access. If the access token is valid, the requested resource is returned. If the access token is invalid, in general the status code 401 (Unauthorized) is returned. However, if the access token is expired, the status code 403 (Forbidden) is returned.

An ID token contains information about authenticated user and is intended to be read by the client. ID tokens are [JWTs](https://oauth.net/2/jwt/) on the other hand Access tokens [can be JWTs](https://oauth.net/2/jwt-access-tokens/) or may also be an opaque i.e., random string. The ID token may also contain information about the authenticated user such as their name or email address, in the form of claims along with other mandatory claims.



OpenID Connect extends the authorization code flow by specifying the parameter *scope=openid* in the token request. The client tells the authorization server to run the OpenID Connect protocol.

**Sequence Diagram**

Following sequence diagrams explain the OIDC flows.

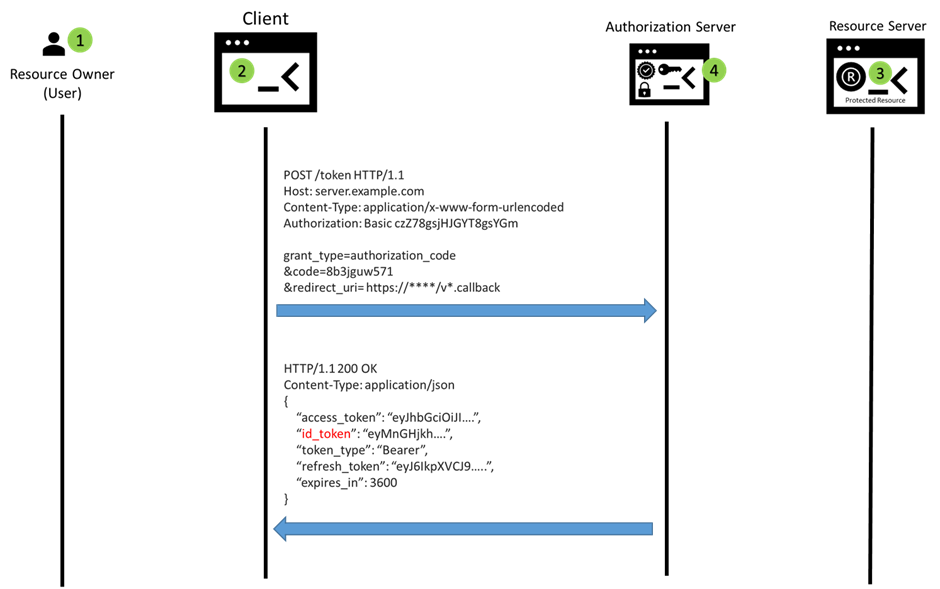
**Authorization End Point**

OpenID Connect extends the authorization code flow. The only difference here is that the OAuth scopes openid, profile and email (both profile and email are not mandatory) are requested. The scopes are separated by a space.



**Token End Point**

The token endpoint for OpenID Connect is based on the basic OAuth token endpoint. The result delivered by this endpoint is an extension of the JSON object containing access token and refresh token. The JSON object additionally contains an id token.



**Register an Application with Azure AD**

As mentioned earlier OpenID Connect is an identity layer on top of OAuth 2.0 protocol.

All the concepts, flows, endpoints, and tokens of OAuth 2.0 also apply in the context of OpenID Connect. OpenID Connect requires one obligatory scope i.e., *openid* in addition to optional scopes such as email or address. The obligatory scope must be present, and it will deliver a basic set of claims.

*Note: APIs are still protected by OAuth, which means that a valid OAuth access token with correct OAuth scopes required.*

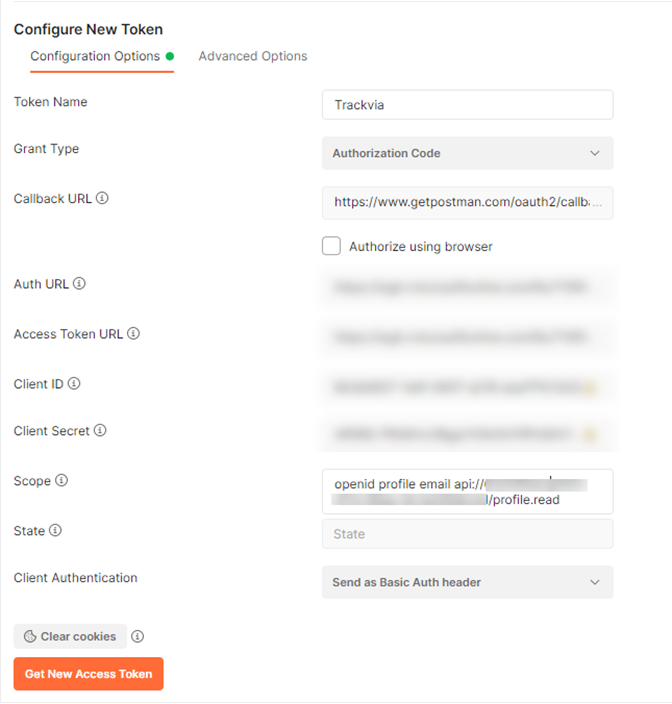
**Register an application**

In [Part 2B](https://medium.com/@shoaib.alam/part-2b-oauth-2-0-authorization-code-grant-with-azure-ad-d97b213ecc9) I already registered two applications DemoClientApp01 and DemoWebApp in Azure AD. I am going to use the same application to get access token and ID token.

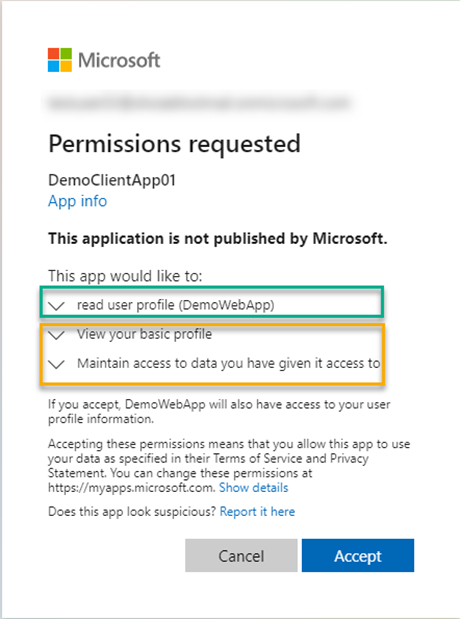
I also assume that Postman is downloaded and installed. In this part we are going to use Authorization Code as a grant type. Rest of the parameters are as follows:

* Token Name: *Any Suitable Name for a Token*
* Grant Type: Authorization Code
* CallBack URL: It is the Redirect URI provided above. For Postman it could be <https://www.getpostman.com/oauth2/callback>
* Auth URL: https://login.microsoftonline.com/<<your tenant ID>>/oauth2/v2.0/authorize
* Access Token URL: https://login.microsoftonline.com/<<your tenant ID>>/oauth2/v2.0/token
* Client ID: 8\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
* Client Secret: z\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
* Scope: openid

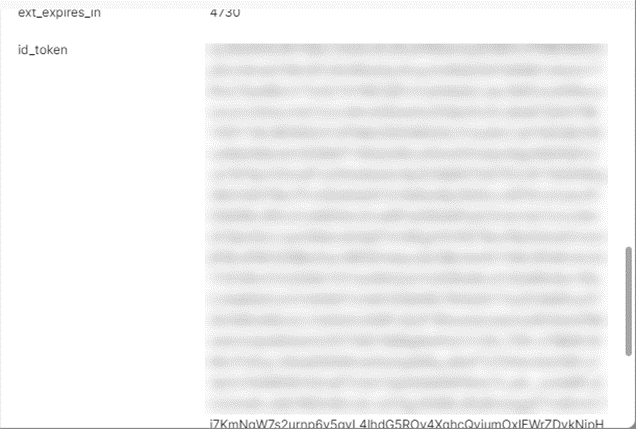
An openid scope is added in the request. You can also add optional scope email, profile etc. For refresh token offline\_access and you can also provide scopes for access token from [Part 2B](https://medium.com/@shoaib.alam/part-2b-oauth-2-0-authorization-code-grant-with-azure-ad-d97b213ecc9).



Now Click the New Access Token Button. A login screen will pop up asking to enter username and password. Once the user is authenticated successfully, it will ask for a permission to allows you to sign into the app with your account and let the app read your profile.



Once you Accept it a new Tokens are issued by Azure AD. There is an additional token called id\_token along with access token added in the response.



**ID Token by Azure AD**

The ID token called id\_token is issued in addition to an access token in a token endpoint response.

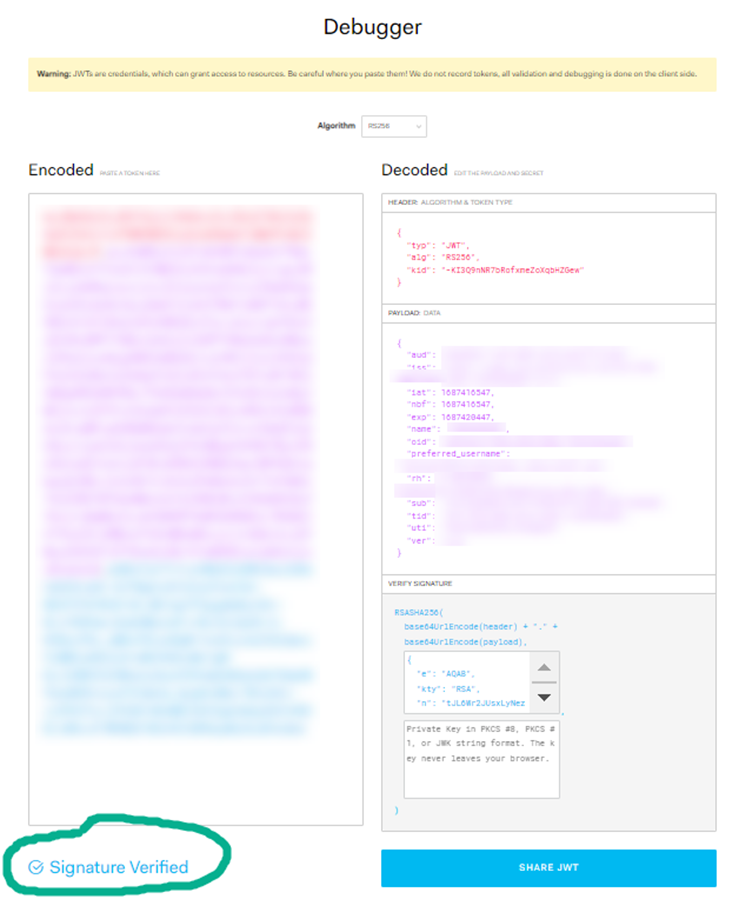
An ID token is encoded as a JSON Web Token (JWT), as a standard format. As mentioned in part 2, JWT typically looks like the xxxxx.yyyyy.zzzzz composed of three parts: Header, Payload and Signature.

JWT ID token carries profile information about end-users that is authenticated in an OpenID Connect flow. Depending upon scopes, this security token may contain information about the user as JWT claims.

ID token should not be used to gain access to an API.

**Decode JWT Access Token**

Let’s decode the id\_token by opening the website jwt.io. The copy and paste the token acquire in the previous step.



The ID token contains a set of claims about the authentication session, including an identifier for the user (*sub*), the identifier for the identity provider that issued the token (*iss*), and the identifier of the client for which this token was created (*aud*). Additionally, the ID token contains information about the token’s own validity time window (with the *exp* and *iat* claims).

**Add Optional Claims**

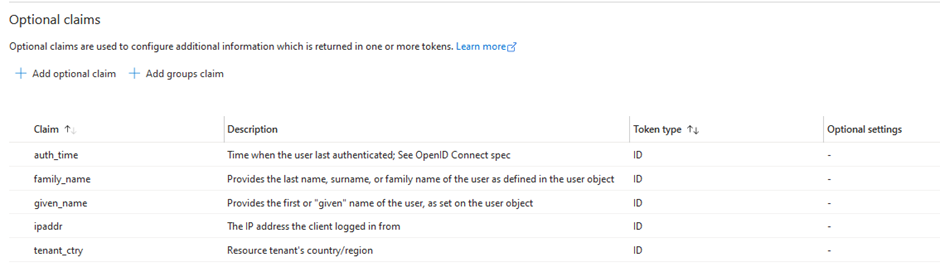
You can configure optional claims for your application through the Azure portal or application manifest. You can configure an application so that optional claims can be returned in ID tokens, access tokens, or SAML 2 tokens. Here we are going to add optional claims in ID token via Azure Portal.

Get Shoaib Alam’s stories in your inbox

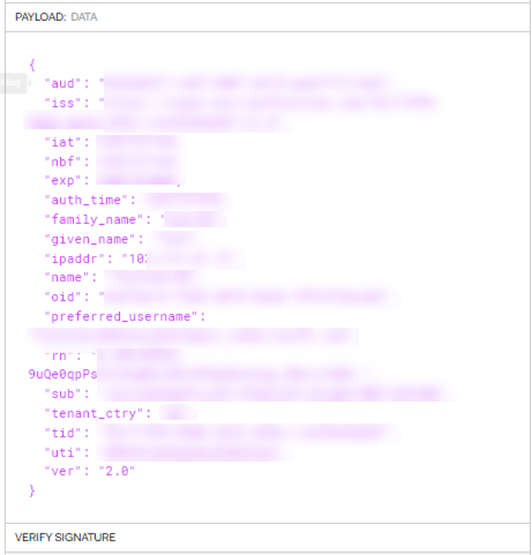
Join Medium for free to get updates from this writer.

Subscribe

Add an optional claim from Azure Active directory by navigating to DemoClientApp01 and going to **Token configuration > Add optional claim** and follow the prompts.



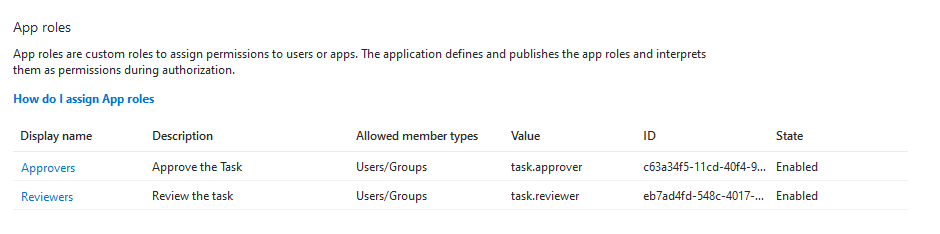
Get a new token and when you decode it these optional claims are present in the id\_token.



**Azure AD Roles for User/Groups for Role Based Access Control**

You can assign a role to a group or user for effective Role Based Access Control (RBAC). Creating groups with specific roles and responsibilities allows to manage access to resources and applications. By assigning roles to groups, you can ensure that only those with the appropriate level of access can perform certain actions or access sensitive data.

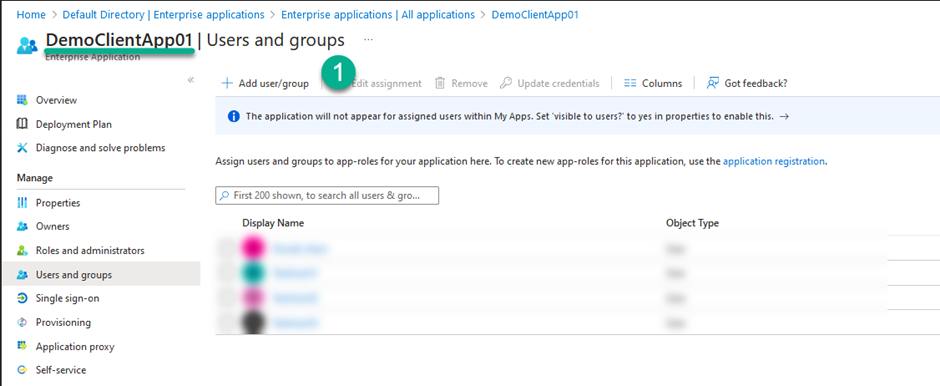
From Azure Active directory navigate to DemoClientApp01 and add app role from **App Roles > Create app role**and create Reviewers and Approvers roles.



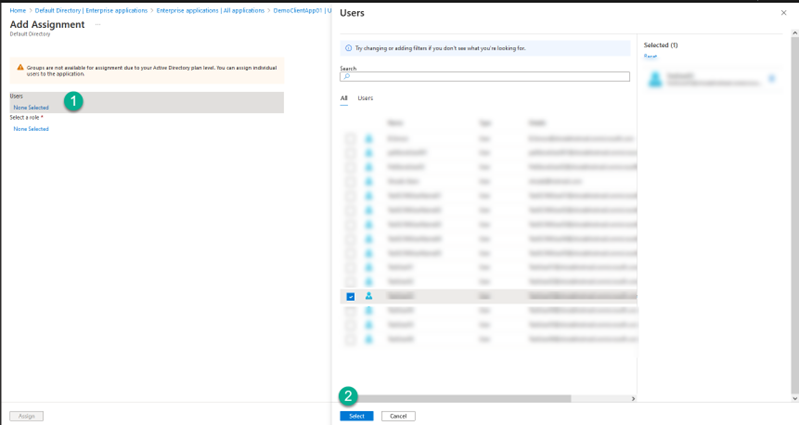
*Assign Role to a User*

Group-based role assignment requires Azure Active Directory Premium P1 or P2 edition. I am only going to show how to assign a role assignment to a User.

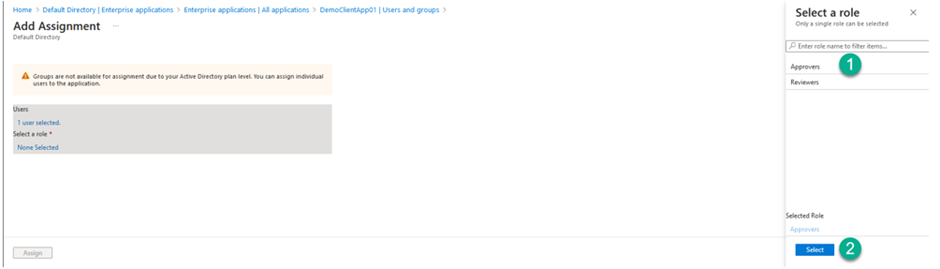
To assign a user (or group) to a role, from **Enterprise applications** select DemoClientApp01. Browse to **Azure Active Directory** > **Users and groups**, and then select **Add user/group**.



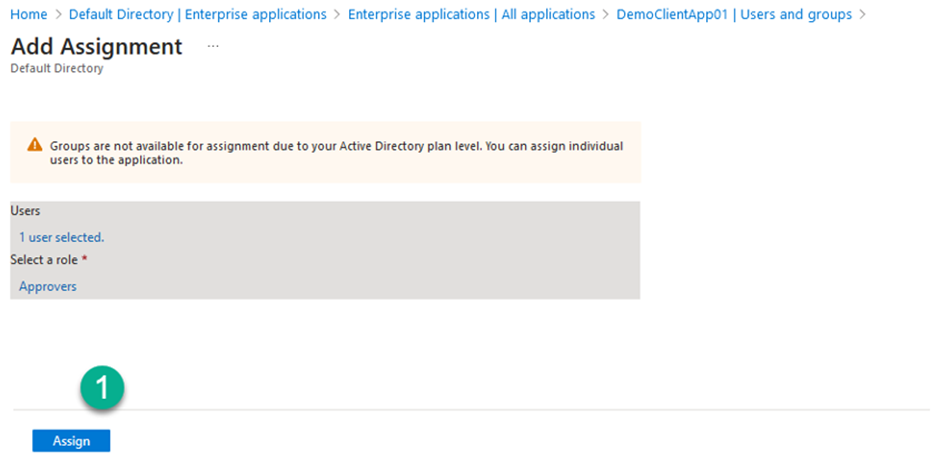
On the **Add Assignment** pane select the User.



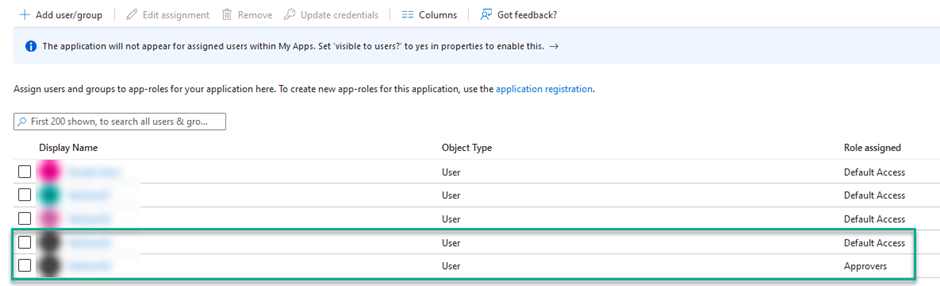
On the **Add Assignment** pane select Approvers Role for a selected user and press the **Select** button.



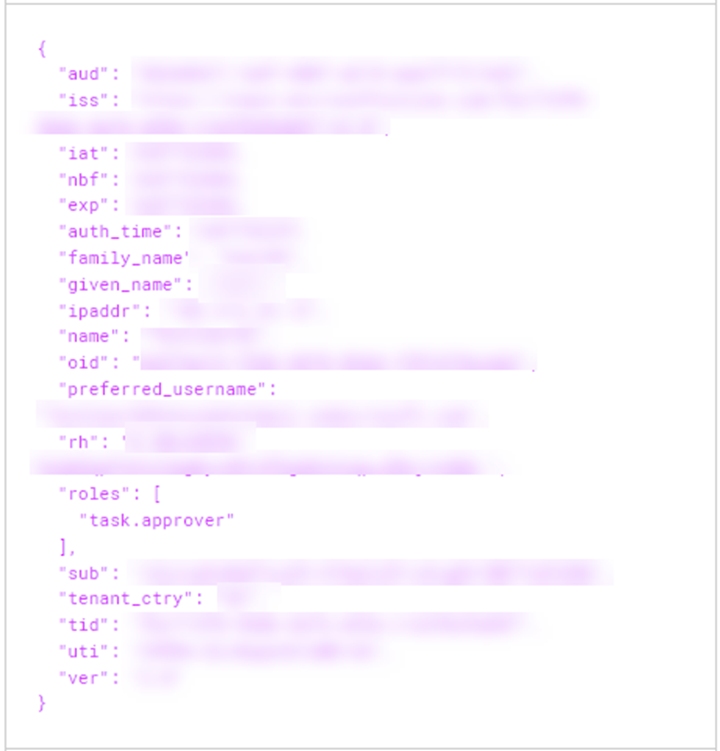
Now an Approvers Role is selected for a user. Finally, assign the role by pressing **Assign** button.



An Approvers Role is assigned to a user along with Default Access.



An Approvers Role is assigned to a user along with Default Access. Using Postman get a new id\_token and decode it. A task.approver role will be present in id\_token.



**Get access and ID token Http GET and POST Request**

**Authorization code using HTTP GET method and authorization end point**

An alternate method to get an OAuth token is to use HTTP GET and POST methods. To start the OAuth process, send an HTTP GET request to the authorization server. The resource owner authenticates to authorization server and authorizes the client.

The authorization server redirects the user to a redirect\_uri with authorization code.

The HTTP GET methods required the followings query parameters:

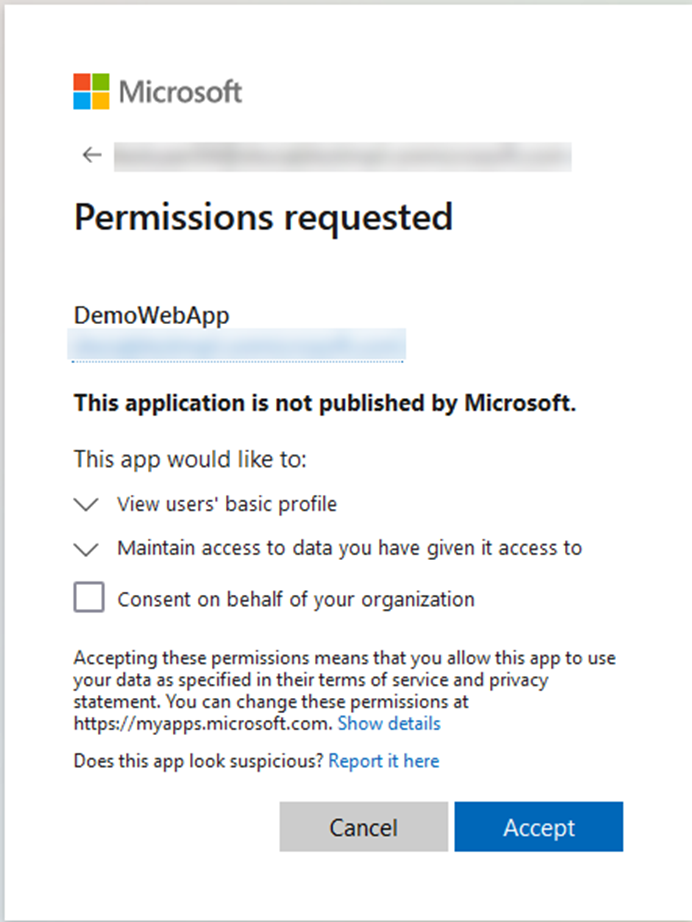
* response\_type
* client\_id
* redirect\_uri
* scope

The value of scope must include openid for ID token. It might look like:

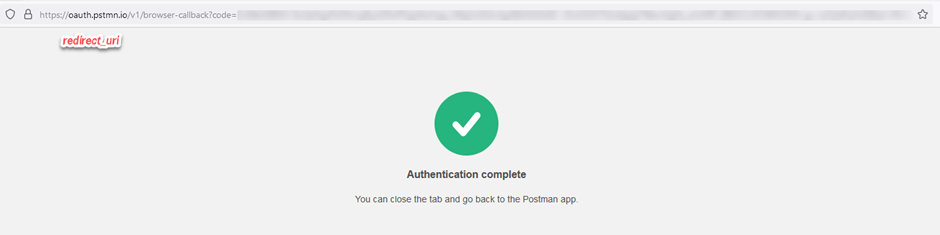
* scope= openid email profile offline\_access
* offline\_access for refresh token



Copy and paste the above URL in the browser. A login screen will pop up asking to enter username and password. Once the user is authenticated successfully, it will ask for a permission to allows you to sign into the app with your account and let the app read your profile.



Once you Accept the required permission it will redirects to redirect\_url and generate a code as a query parameter.

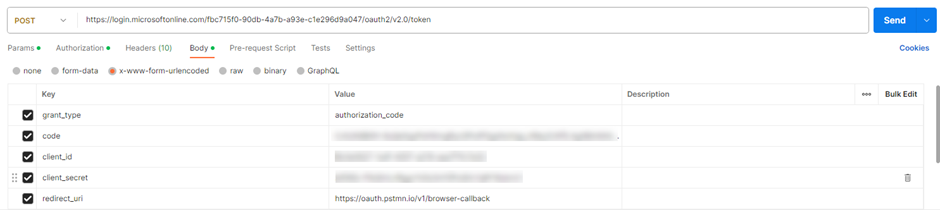


Extract the code from the query parameter.

**Access token using HTTP POST method and token end point**

Open Postman and create the HTTP POST request which requires the followings for Body. It is recommended to create the POST request upfront as authorization code is short lived and might expire by the time you finish creating the POST request (step 4).

* grant\_type
* code
* client\_id
* client\_secret
* redirect\_uri



Send the POST request to the token end point and it will generate an access token, id token and refresh token.

