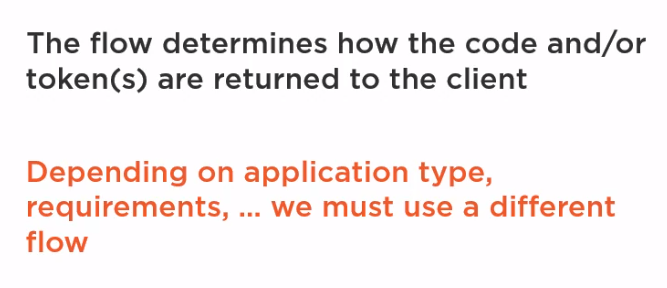
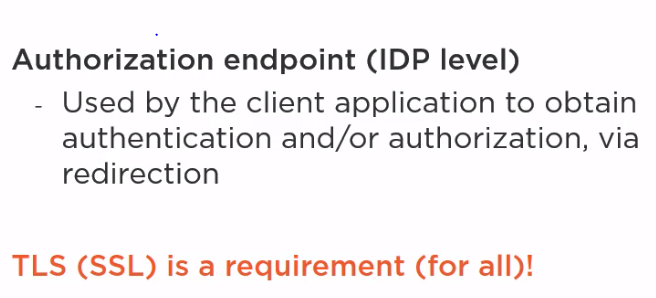
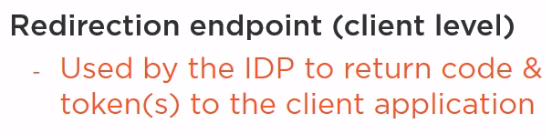
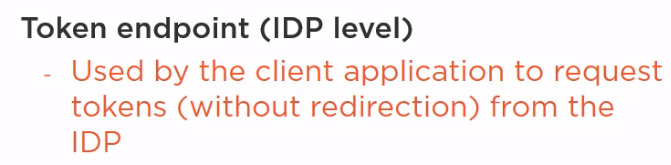
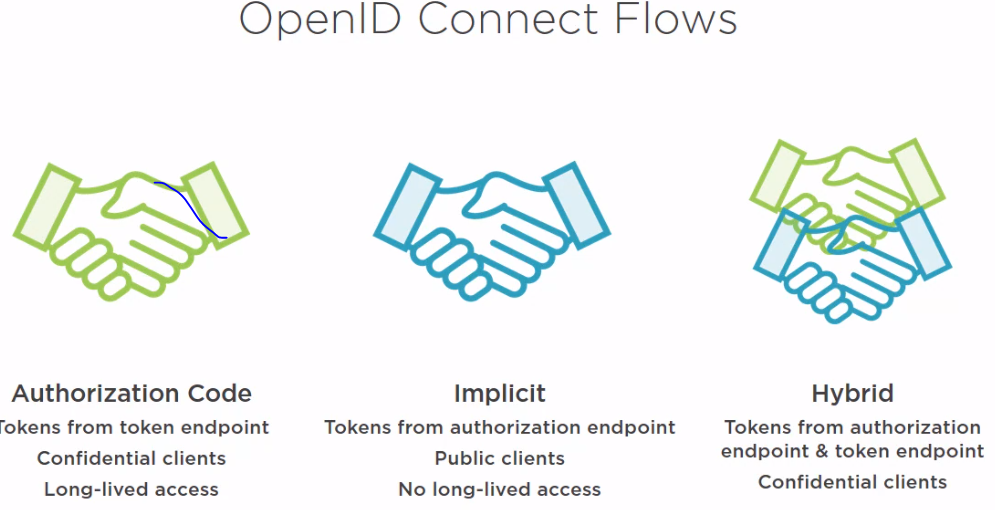
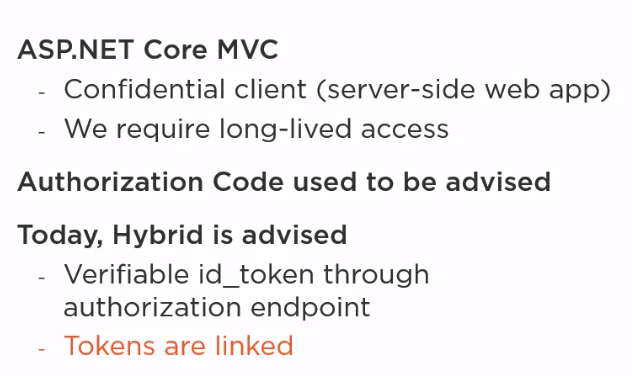
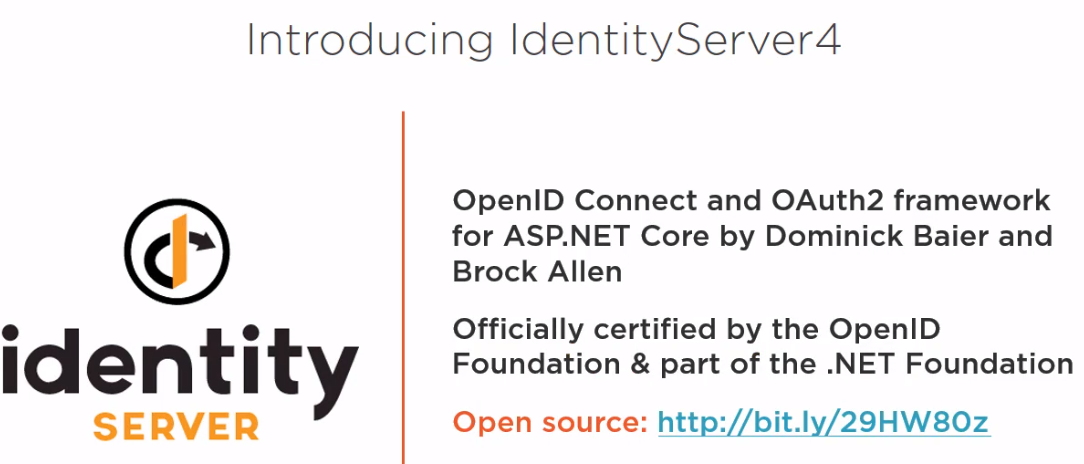
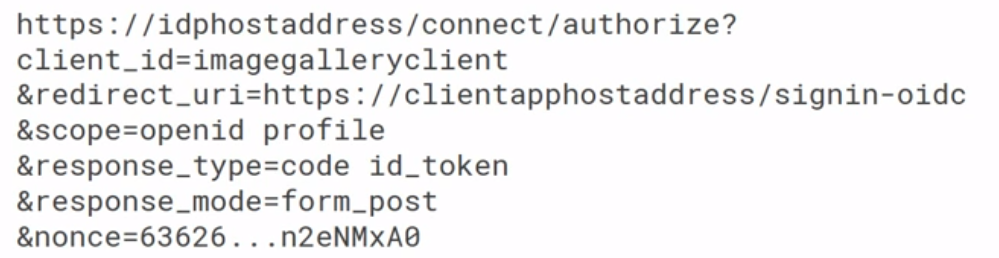
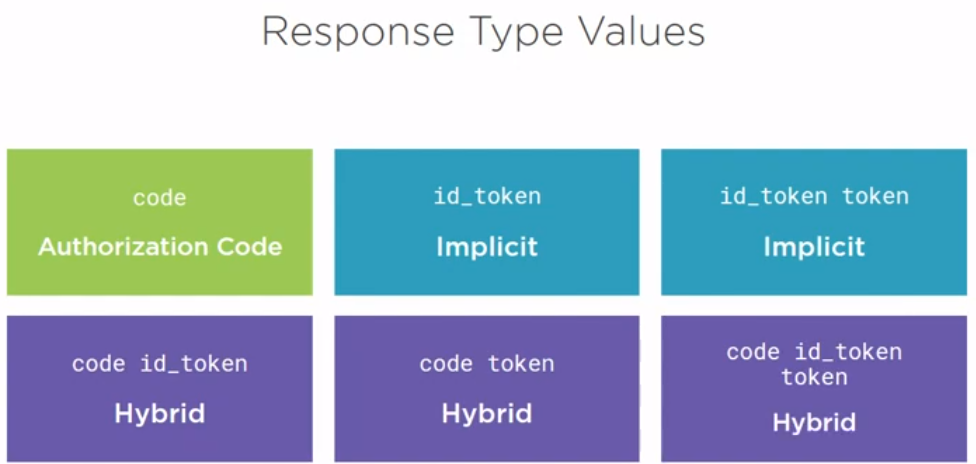
1. 
2. **There are Various End point present inside the openId Connect to handle different type of flows:**  
   Authorization endpoint is used by the client app to obtain authentication for identity tokens and/or authorization for access tokens from the user. This is done by redirecting the client application to the identity provider. So from the authorization endpoint, an authorization code or tokens can be returned to the client application. Important to know is that in OIDC, TLS, or as it's wrongly named SSL, is a requirement. In other words, traffic should always be encrypted.

  
Redirection Endpoint  
We're talking about redirection here. The client redirects to the authorization endpoint at level of the identity provider and the identity provider redirects back to the client. The URI or the IDP redirects the client back to what's called the redirection endpoint  
  
  
Token EndPoint -   
From this endpoint, which lives at level of the identity provider, client applications can programmatically request tokens and that's typically done via an HTTP post without redirection.  
  


1. **Different Type of Flows Present Inside OpenId Connect and OAuth2.0**The first one is the OpenID Connect authorization code flow. It returns an authorization code from the authorization endpoint and tokens from the token endpoint. This authorization code can be seen as a short-lived single use credential used to verify that the user who logged in at level of the identity provider is the same one who started the flow at level of the web app. The authorization code flow is suitable for confidential clients and it allows long-lived access. The implicit flow returns all tokens from the authorization endpoint and there is no authorization code. The token endpoint is induced. It's suitable for public clients. There is no client authentication as public clients can safely store client secrets anyway. Therefore, there is also no long-lived access through the fresh tokens allowed. The implicit flow may be used by confidential clients as well. Lastly, the hybrid flow. This one returns some tokens from the authorization endpoint and others from the token endpoint. It's just a mix of the authorization code and implicit grant or flow.  
   
2. Refresh tokens aren't allowed with an implicit flow, so we won't choose that. And that leaves us with two flows, the authorization code and the hybrid flow. Before OpenID Connect was conceived, the authorization code flow was the flow of choice. For that flow, as we learned, all tokens are returned from the token endpoint, but the hybrid flow has an advantage over that one. It allows us to get an identity token from the authorization endpoint first and we can then verify that before continuing with additional roundtrips to get an access token. Next to that, OpenID Connect links the identity token to an optional access token and these are reasons to choose OpenID Connect over regular OAuth2.  
   
3. 
4. IdentityResources map to scopes that give access to identity related information. Later, we'll also encounter API resources. Those map to scopes that give access to API resources.
5. <http://localhost:53533/.well-known/openid-configuration>-  
   It's this discovery document that's read by other pieces of middleware when we they want to find out where, for example, those endpoints can be found.
6. There is the endpoint itself to the authorization URI at the level of the IDP. We recognize that from the discovery document. Then there is the client id. That's the identifier of the client application. To redirect URI value is the redirection endpoint at the level of the client app. We also see scopes. The application requests access to the openid and profile scopes, so it wants access to the user's identifier and the profile related claims like given name and family name. We'll cover response mode and nonce soon, but the part we want to focus on now is the response type. It's currently set to code id\_token, but what does that mean? Well, it's the response type value that determines the flow that's used.  
     
   The value of the response type inside the end point determines which type of flow is requested from the identity data provider(IDP)  
     
   
7. Our web app creates an authentication request with response type code id\_token and other parameters like scopes and so on as we just see on the slides. The web app sends the request. At the level of the IDP, a user authenticates, for example, by providing his or her username and password combination. From that moment on, the user is authenticated at level of the IDP. The identity provider optionally asks the user for consent, ie. it'll ask you if you want to allow the application to get your profile information. The identity provider sends us back to the web application via URI redirection or by a form post with the authorization code and the identity token, that's the response type we asked for. The WebClient then validates the identity token it got back from the authorization endpoint. If it's invalid, execution of the flow stops. If it's valid, the flow continues. The middleware calls the token endpoint through the backchannel. For this, it passes through the authorization code, clientid, and clientsecret, ie. the client application has to authenticate itself. In the response, we get back an identity token, this is validated as well. Amongst other things, which we'll cover later on, one of the validation steps is to check if this token matches the user in the previous token, and at this time, the middleware extracts the user's identifier from the identity token. Now this may look a bit weird, after all, we've already got an identity token from the authorization endpoints, why not just use that? Well, the token endpoint requires client authentication with the secret that's not exposed so that adds an additional layer of security. Moreover, I deliberately skipped on a few things to keep this simple. In reality, access tokens and refresh tokens can be returned from this token endpoint as well, so it's not just a matter of getting a second identity token. But don't worry about that for now, we're covering all of that later on in the course. We're currently focusing on the identity token and this identity token after it has been validated has proved that the user is who he or she says he or she is. So it can be used by applications for that purpose. A difficult thing to do with such a token is create a claims identity from it and use that to sign into our ASP. NET Core MVC app.  
   