1.4.2 OpenID Connect

Since OAuth 2.0 is for authorization, but not authentication, OpenID Connect adds an authentication layer on top of OAuth 2.0. OpenID Connect has enabled a whole ecosystem of websites to no longer need to deal with user management and authenticating users. OpenID Connect is useful to have a centralized solution for authentication supporting single sign-on and increases security as applications do not have access to the user credentials directly. Furthermore, it enables the use of stronger authentication, such as OTP or WebAuthn, without the need to support it directly within applications.

OpenID Connect (OIDC) involves three (3) main roles in its protocol:

1. **End user**: It is the same as the resource owner in OAuth 2.0. This is simply the human being who is logging in or being authenticated.
2. **Relying Party (RP):** The application or website that needs to verify or authenticate the end user.
3. **OpenID Provider (OP):** The identity provider that is authenticating the user, which is the role of Keycloak.

In OpenID Connect, the RP asks the OP for the user's identity and can also obtain a token since it builds on OAuth 2.0. OpenID Connect uses OAuth 2.0's Authorization Code grant but adds `scope=openid` to make it an **authentication request** instead of just an **authorization request**. There are two (2) flows in OpenID Connect:

1. Authorization code flow: This follows the OAuth 2.0 Authorization Code flow, returning an authorization code that can be exchanged for an ID token, access token, and refresh token.

2. Hybrid flow: The ID token and authorization code are returned together in the initial request.

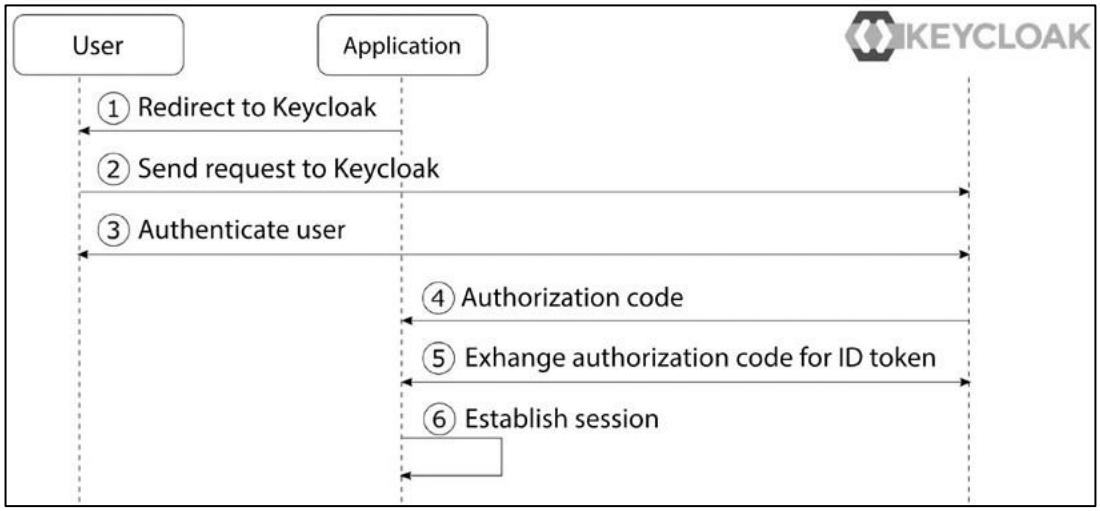

Figure 3: Simplified OpenID Connect Authorization Code Flow

The steps in the diagram are as follows:

1. The app prepares a request to authenticate and asks the user's browser to be redirected to Keycloak.
2. The browser redirects the user to Keycloak's login page (authorization endpoint).
3. If the user is not logged in yet, Keycloak will ask them to log in.
4. The app gets an authorization code from Keycloak as a response.
5. The app then exchanges this authorization code with Keycloak for an ID token and an access token.
6. The app uses the ID token to identify the user and start a logged-in session.