

# OAuth and OpenID


Presented By :Riya Arora


# OpenID Connect


- ▶ An identity layer that sits on top of OAuth 2.0 protocol.
- ▶ Enables client to verify identity of end user based on authentication done by authorization server
- ▶ Obtains profile info of end user
- ▶ OAuth it's an open standard for authorization provides authorization but **not authentication**
- ▶ It can extend OAuth so applications can get identity information.
- ▶ The OpenID Connect flow looks the same as OAuth.
- ▶ The only differences are, in the initial request, a specific scope of openid is used
- ▶ In the final exchange the **Client** receives both an **Access Token** and an **ID Token**

# Application of OAuth

The image shows a login interface with three social media login options at the top, each in a white box with a colored icon and text: 'Continue with Google' (Google G logo), 'Continue with Facebook' (Facebook f logo), and 'Continue with Apple' (Apple logo). Below these is a horizontal line with the word 'or' in the center. This is followed by an email login section with the label 'EMAIL' in bold, a text input field containing the placeholder 'Email', and a password section with the label 'PASSWORD' in bold, a text input field containing the placeholder 'Password'. To the right of the password field is a blue link that says 'Forgot password?'. Below the password field is a large blue button with the text 'Log in' in white. At the bottom, there is another horizontal line with the word 'or' in the center.

 Continue with Google

 Continue with Facebook

 Continue with Apple

or

**EMAIL**

**PASSWORD**

[Forgot password?](#)

**Log in**

or

# Basic Concepts

- ▶ **Participants:** End User, Relying party, Identity provider
- ▶ **Identity Tokens:** An ID Token is a specifically formatted string of characters known as a JSON Web Token
- ▶ **Claims:** The data inside the ID Token are called *claims*. Are used to retrieve info. Eg: name, email
- ▶ **Scopes:** used to request that specific sets of info made available as claim values. The granular permissions the Client wants, such as access to data or to perform actions. Eg: openid, profile , email

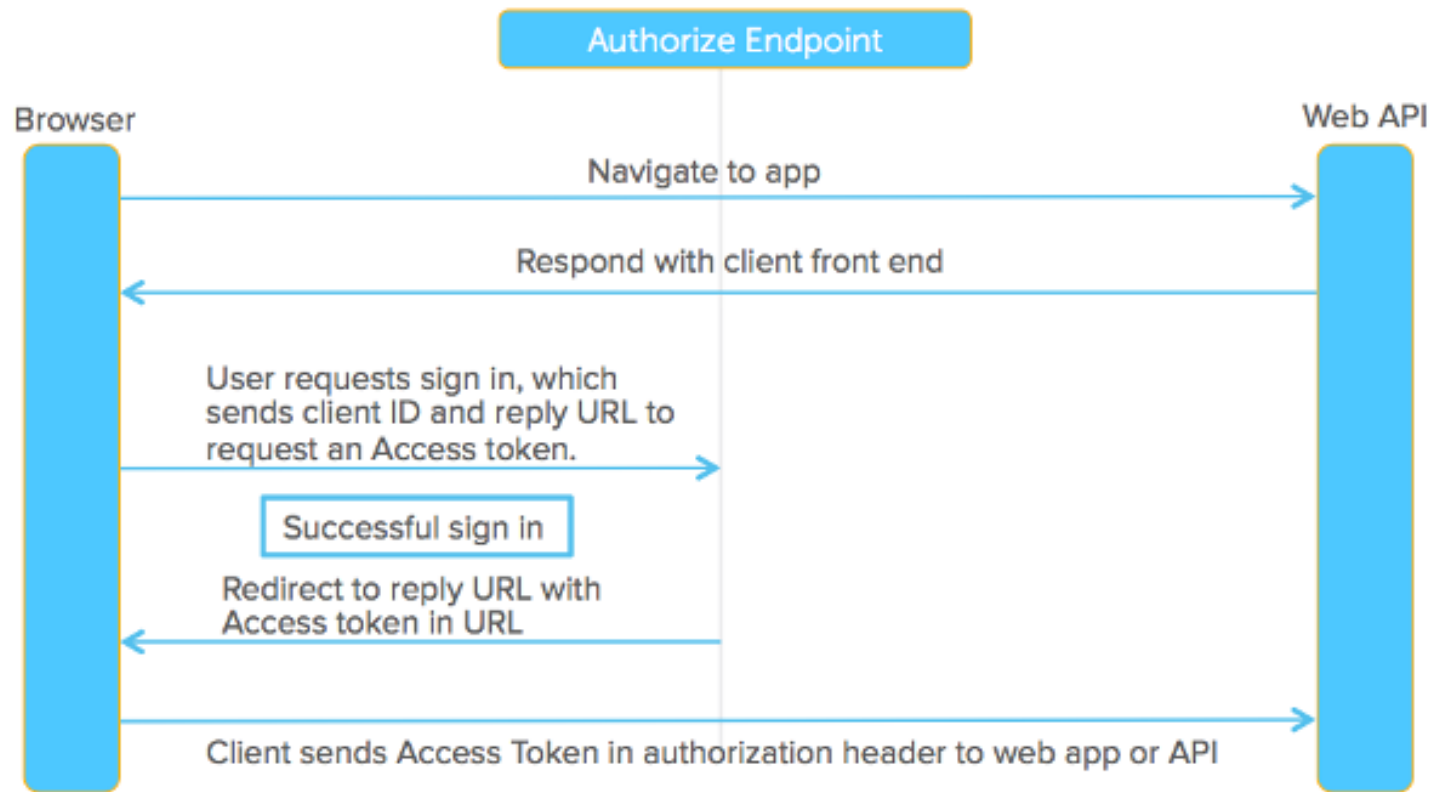
# OAuth terms

- ▶ **Resource Owner:** An entity capable of granting access to a protected resource.
- ▶ **Resource Server:** Server hosting the protected resource.Eg:Google
- ▶ **Client:** An application making protected resource requests on behalf of the resource owner and its authorization.
- ▶ **Authorization server:** Issues access tokens to client.
- ▶ **Authorization grant:**The user consents the access to resource.
- ▶ **Access Token:** The key the client will use to communicate with the Resource Server.
- ▶ **Redirect Uri:** The URL the Authorization Server will redirect the Resource Owner back to after granting permission to the Client. (Callback URL)
- ▶ **Response Type:** The type of information the Client expects to receive.
- ▶ **Authorization Code:** A short-lived temporary code the Client gives the Authorization Server , in exchange for an Access Token.

# OAuth flow

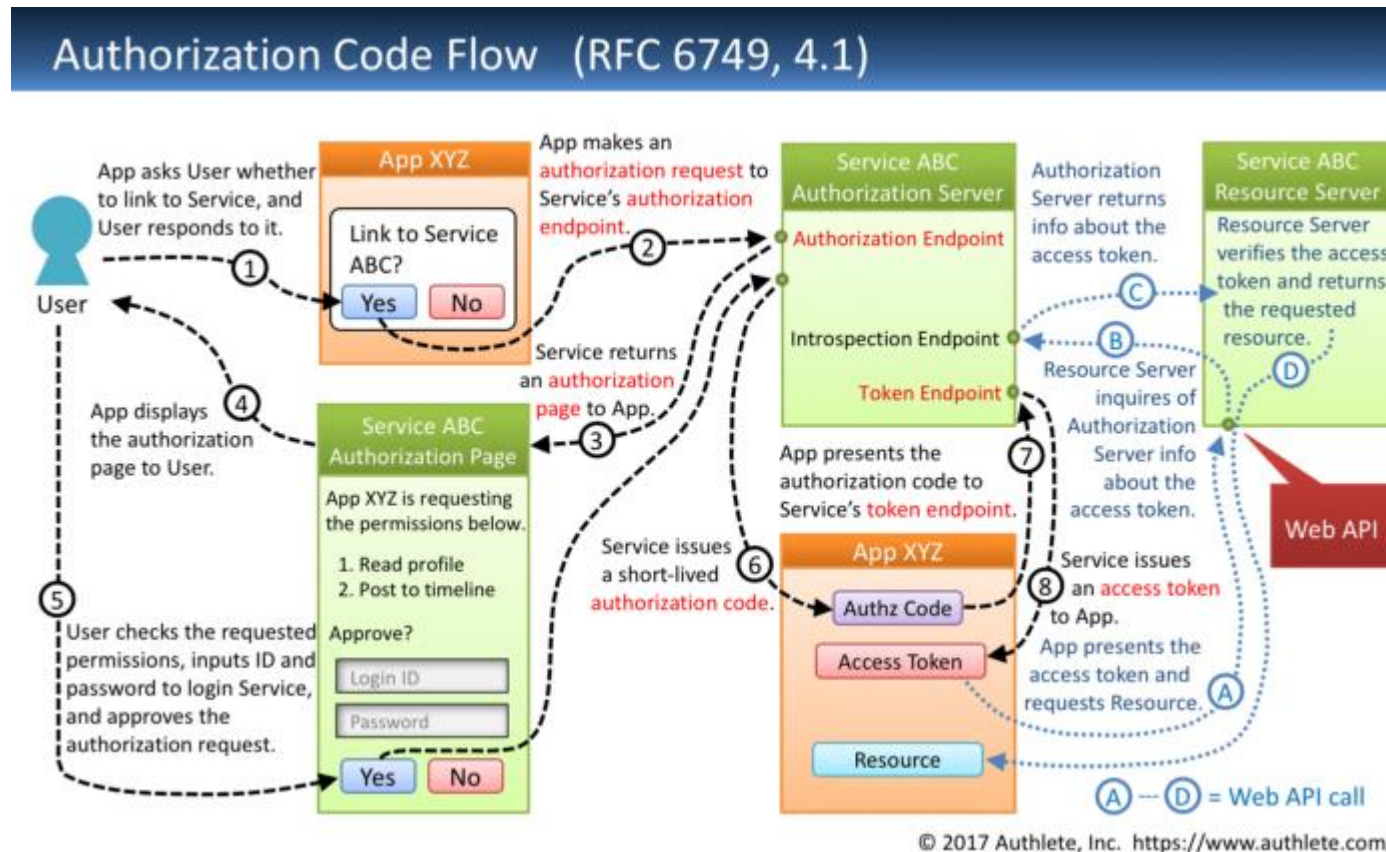
1. **Resource Owner**, want to allow the **Client**, to access contacts so they can send invitations to all your friends.
2. The **Client** redirects your browser to the **Authorization Server** and includes with the request the **Client ID**, **Redirect URI**, **Response Type**, and one or more **Scopes** it needs.
3. The **Authorization Server** verifies who you are, and if necessary prompts for a login.
4. The **Authorization Server** presents you with a **Consent** form based on the **Scopes** requested by the **Client**. You grant (or deny) permission.
5. The **Authorization Server** redirects back to **Client** using the **Redirect URI** along with an **Authorization Code**.
6. The **Client** contacts the **Authorization Server** directly (does not use the **Resource Owner's** browser) and securely sends its **Client ID**, **Client Secret**, and the **Authorization Code**.
7. The **Authorization Server** verifies the data and responds with an **Access Token**.
8. The **Client** can now use the **Access Token** to send requests to the **Resource Server** for your contacts.

# OAuth flow



# OAuth flows

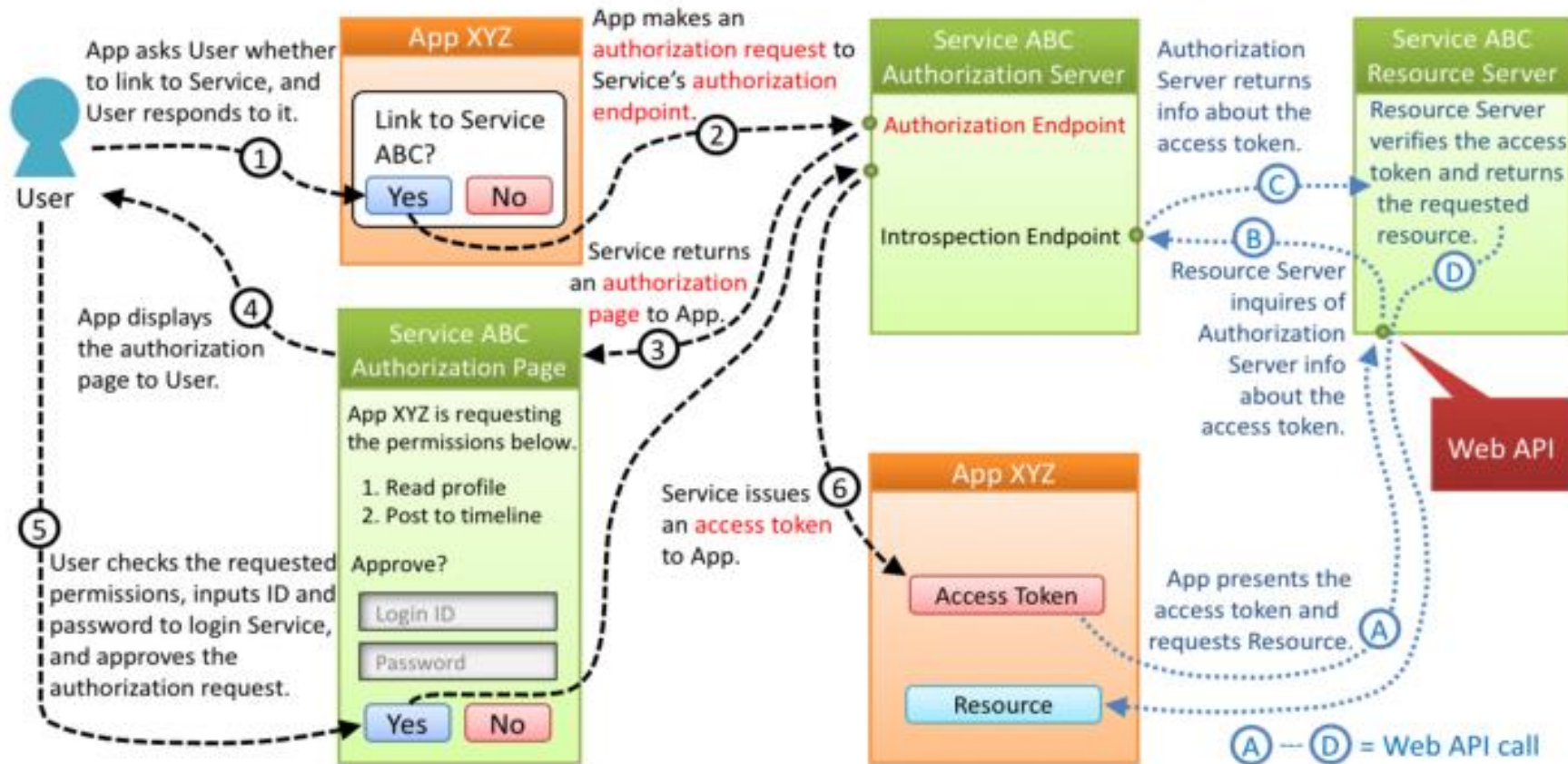
- ▶ OpenID Connect defines The following authentication flows:
- ▶ Authorization code flow:
- ▶ A client application (a) makes an authorization request to an authorization endpoint, (b) receives a short-lived authorization code, (c) makes a token request to a token endpoint with the authorization code, and (d) gets an access token.





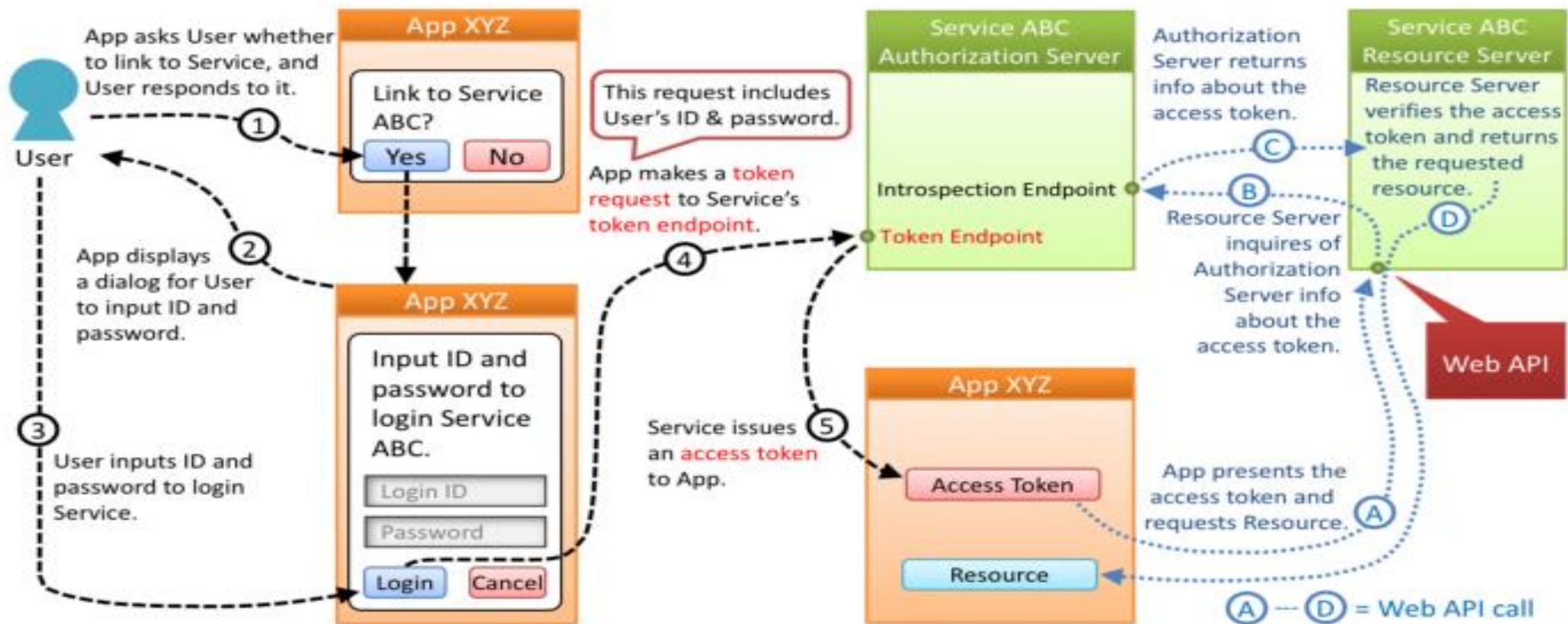
- **Implicit flow:**
- A client application (a) makes an authorization request to an authorization endpoint and (b) gets an access token directly from the authorization endpoint.

## Implicit Flow (RFC 6749, 4.2)



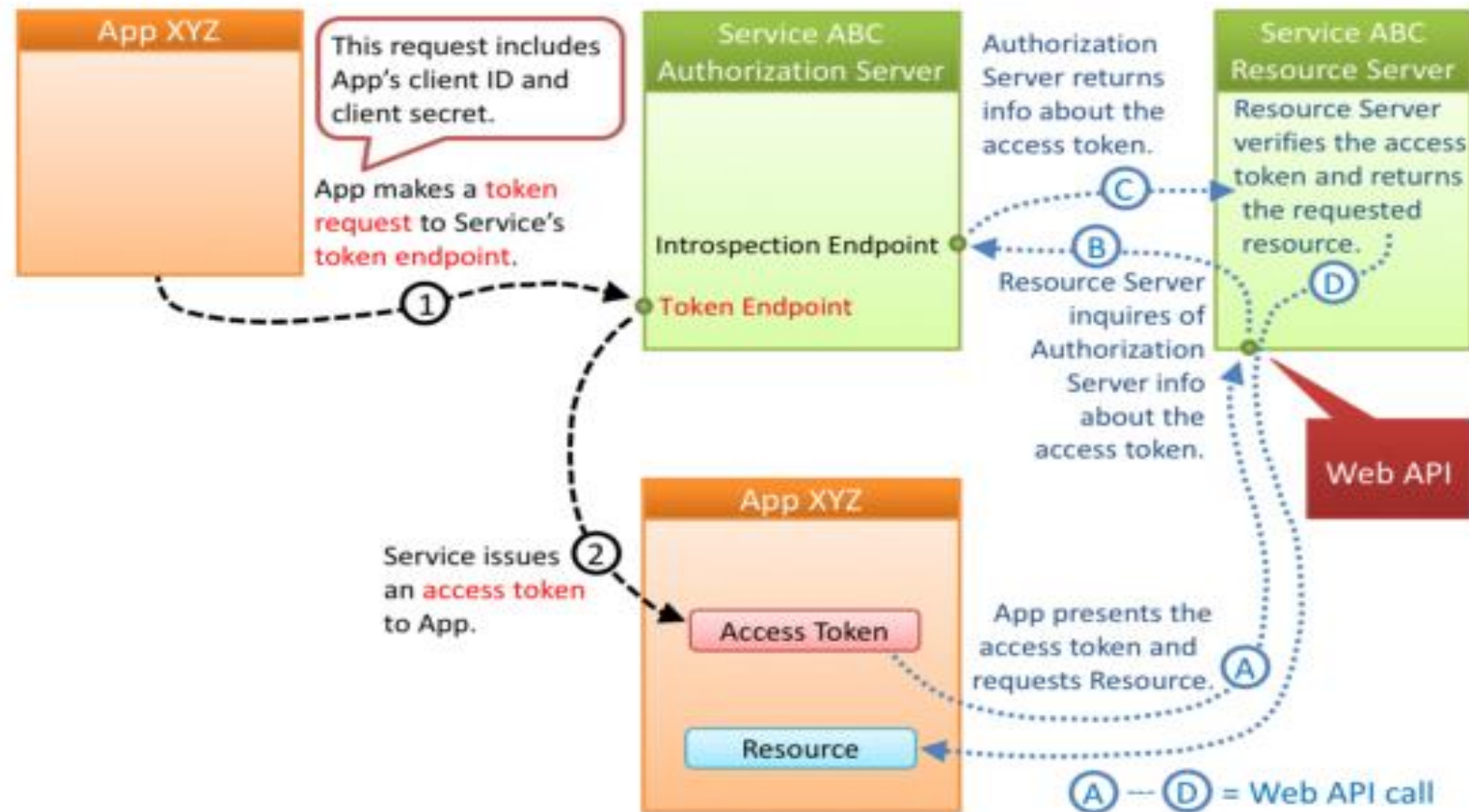
- ▶ Resource Owner Password credentials flow
- ▶ A client application (a) makes a token request to a token endpoint and (b) gets an access token. In this flow, a client application accepts a user's ID and password although the primary purpose of OAuth 2.0 is to give limited permissions to a client application WITHOUT revealing the user's credentials to the client application.

## Resource Owner Password Credentials Flow (RFC 6749, 4.3)



- ▶ Client credentials flow
- ▶ A client application (a) makes a token request to a token endpoint and (b) gets an access token. In this flow, user authentication is not performed and client application authentication only is performed.

## Client Credentials Flow (RFC 6749, 4.4)





# OAuth tokens

- ▶ Tokens are retrieved from endpoints on the authorization server.
- ▶ Access Token : Represents an authorization that a client may have to do something on behalf of user
- ▶ The token that the client uses to access the Resource Server (API).
- ▶ They're meant to be short-lived.
- ▶ Can be used by a specific client app
- ▶ It has a time out.
- ▶ Refresh Token: Allows clients to obtain a fresh access token without reobtaining authorization from resource owner.
- ▶ This is much longer-lived; days, months, years.
- ▶ This can be used to get new token

```
{  
  "access_token": <access_token>,  
  "token_type": "Bearer",  
  "expires_in": 3600,  
  "refresh_token": <refresh_token>  
}
```

# Grants Types in OAuth

- ▶ Methods to get access tokens from the authorization server are called **grants**.
- ▶ The four basic grant types are:
  1. Authorization Code
  2. Implicit
  3. Resource Owner Credentials
  4. Client Credentials
- ▶ Grant types are chosen based on access token owner and client type.
- ▶ Client credentials are used on batch processes without a resource owner.
- ▶ Resource owner credentials is used when client app is trustworthy.
- ▶ Authorization code is used when client app delegates resource owner credentials to authorization server and consent is needed
- ▶ Implicit is used when client app is a user agent based app

# OpenID Connect ID Token

- ▶ Issued by identity provider
- ▶ Contains attributes or claims about end user { Claims }
  - Subject
  - Issuing Authority
  - Audience
  - Issue Date
  - Expiration Date
- ID Token is encoded as a JSON Web Token(JWT) and is digitally signed.

# JWT

- ▶ A JSON Web Token (JWT) is an open standard ([RFC 7519](#)) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object.
- ▶ This information can be verified and trusted because it is digitally signed.
- ▶ JWTs can be signed using a secret or a public/private key pair.
- ▶ **Structure:**
- ▶ **1.Header:** The header *typically* consists of two parts: the type of token, which is JWT, and the hashing algorithm
- ▶ **2.Payload:** The second part of the token is the payload, which contains the claims.
- ▶ **3.Signature:** To create the signature part, you have to take the encoded header, the encoded payload, a secret, the algorithm specified in the header, and sign that.

# JWT

## How does a JWT look like?



### Header:

```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```

### Payload:

```
{  
  "sub": "123456",  
  "name": "xyz",  
  "email": "xyz@gmail.com"  
}
```

### Signature:

```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  secret)
```





- ▶ Authentication: Answering who am I with a proof of identity.
- ▶ Authorization: What a user is allowed to do.