tutorialspoint

SIMPLY EASY LEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

OAuth2.0 is an open authorization protocol, which allows accessing the resources of the resource owner by enabling the client applications on HTTP services such as Facebook, GitHub, etc. It allows sharing of resources stored on one site to another site without using their credentials. It uses username and password tokens instead.

Audience

This tutorial is designed for software programmers who would like to understand the concepts of OAuth. This tutorial will give you enough understanding on OAuth from where you can take yourself to higher levels of expertise.

Prerequisites

Before proceeding with this tutorial, you should have a basic understanding of authorization and authentication of a basic client server application model.

Copyright & Disclaimer

Table of Contents

1. OAuth 2.0 – Overview

What is OAuth 2.0?

OAuth is an open authorization protocol, which allows accessing the resources of the resource owner by enabling the client applications on HTTP services such as Facebook, GitHub, etc. It allows sharing of resources stored on one site to another site without using their credentials. It uses username and password tokens instead.

OAuth 2.0 is developed by the *IETF OAuth Working Group,* published in October 2012.

Why Use OAuth 2.0?

- You can use OAuth 2.0 to read data of a user from another application.
- It supplies the authorization workflow for web, desktop applications, and mobile devices.
- It is a server side web app that uses authorization code and does not interact with user credentials.

Features of OAuth 2.0

- OAuth 2.0 is a simple protocol that allows to access resources of the user without sharing passwords.
- It provides user agent flows for running clients application using a scripting language, such as JavaScript. Typically, a browser is a user agent.
- It accesses the data using tokens instead of using their credentials and stores data in online file system of the user such as Google Docs or Dropbox account.

Advantages of OAuth 2.0

- OAuth 2.0 is a very flexible protocol that relies on SSL (Secure Sockets Layer that ensures data between the web server and browsers remain private) to save user access token.
- OAuth 2.0 relies on SSL which is used to ensure cryptography industry protocols and are being used to keep the data safe.
- It allows limited access to the user's data and allows accessing when authorization tokens expire.
- It has ability to share data for users without having to release personal information.
- It is easier to implement and provides stronger authentication.

Disadvantages of OAuth 2.0

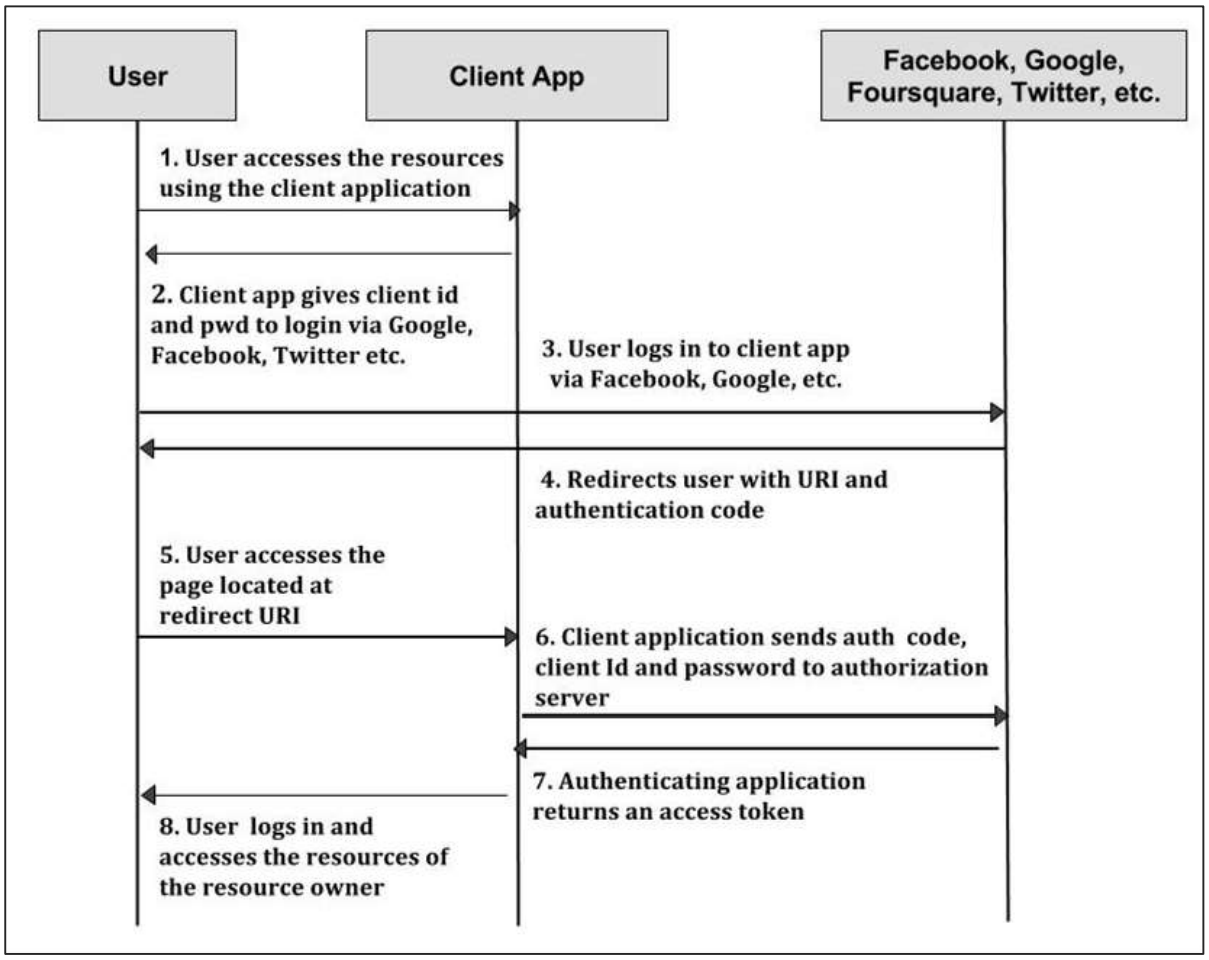
- If you are adding more extension at the ends in the specification, it will produce a wide range of non-interoperable implementations, which means you have to write separate pieces of code for Facebook, Google, etc.

- If your favorite sites are connected to the central hub and the central account is hacked, then it will lead to serious effects across several sites instead of just one.

For information about OAuth 2.0 diagram and some various concepts, refer this link.

2. OAuth 2.0 – Architecture

In this chapter, we will discuss the architectural style of OAuth 2.0.



Step 1: First, the user accesses resources using the client application such as Google, Facebook, Twitter, etc.

Step 2: Next, the client application will be provided with the client id and client password during registering the redirect URI (Uniform Resource Identifier).

Step 3: The user logs in using the authenticating application. The client ID and client password is unique to the client application on the authorization server.

Step 4: The authenticating server redirects the user to a redirect Uniform Resource Identifier (URI) using authorization code.

Step 5: The user accesses the page located at redirect URI in the client application.

Step 6: The client application will be provided with the authentication code, client id and client password, and send them to the authorization server.

Step 7: The authenticating application returns an access token to the client application.

Step 8: Once the client application gets an access token, the user starts accessing the resources of the resource owner using the client application.

OAuth 2.0 has various concepts, which are briefly explained in the following table.

Sr. No.	Concept & Description
1	**Terminology** OAuth provides some additional terms to understand the concepts of authorization.
2	**Web Server** Web server delivers the web pages and uses HTTP to serve the files that forms the web pages to the users.
3	**User-Agent** The user agent application is used by client applications in the user's device, which acts as the scripting language instance.
4	**Native Application** Native application can be used as an instance of desktop or mobile phone application, which uses the resource owner password credentials.

Terminology

Following is the explanation of OAuth 2.0 terms:

Authentication

Authentication is a process of identifying an individual, usually based on a username and password. It is about knowing that the user is the owner of the account on the web and desktop computers.

Federated Authentication

Many applications have their own username and passwords. Some applications depend on other services for verification of the user's identity. A federated identity management system provides a single access to multiple systems. This is known as federated authentication.

Authorization

Authorization is the process of giving someone the permission to do something. It needs the valid user's identification to check whether that user is authorized or not.

Delegated Authorization

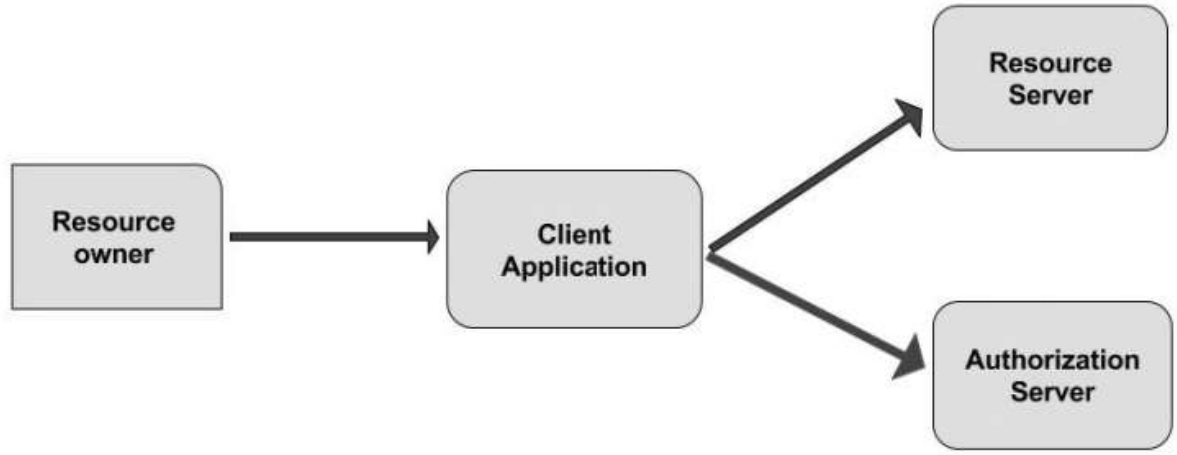
Delegated authorization is the process of giving one’s credentials to other user to perform some actions on behalf of that user.

Roles

OAuth defines the following roles:

- Resource Owner
- Client Application
- Resource Server
- Authentication Server

The roles are illustrated in the following figure:



- **Resource Owner**: Resource owner is defined as an entity having the ability to grant access to their own data hosted on the resource server. When the resource owner is a person, it is called the end-user.

- **Client Application**: Client is an application making protected resource requests to perform actions on behalf of the resource owner.

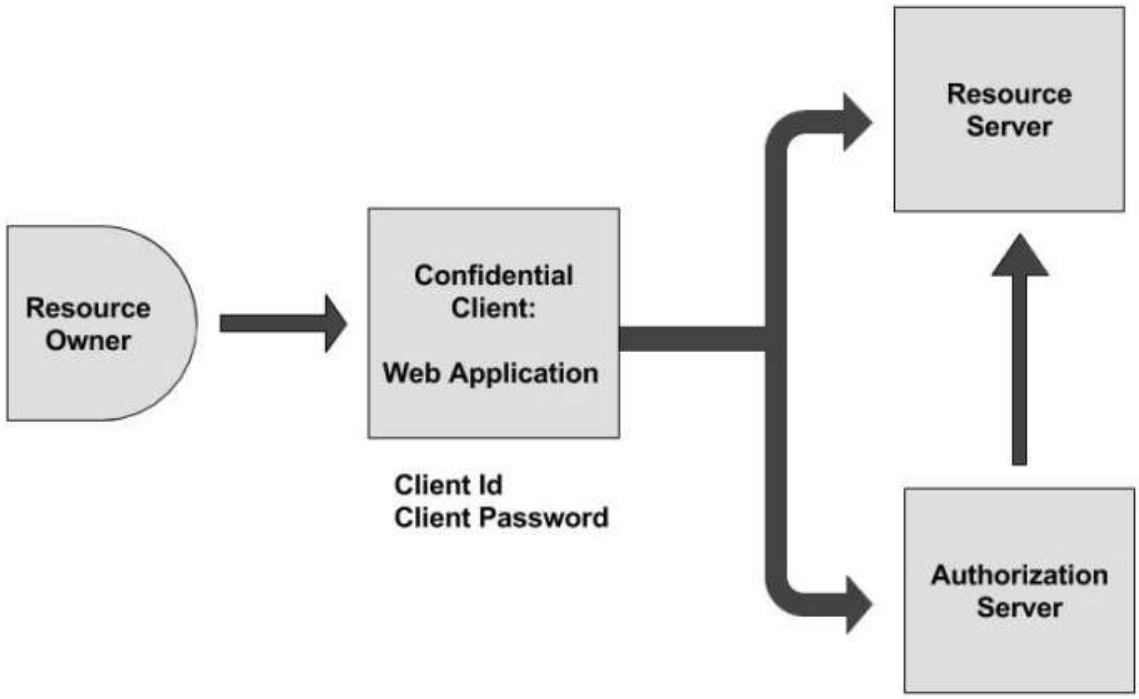
- **Resource Server**: Resource server is API server that can be used to access the user's information. It has the capability of accepting and responding to protected resource requests with the help of access tokens.

- **Authentication Server**: The authentication server gets permission from the resource owner and distributes the access tokens to clients, to access protected resource hosted by the resource server.

Web Server

The web server is a computer system that delivers the web pages to the users by using HTTP. The client ID and password is stored on the web application server, whenever the application wants to access the resource server. The client ID and password which is stored on the web application server is intended to be kept secret.

The following figure depicts the Confidential Client Web Application Server:

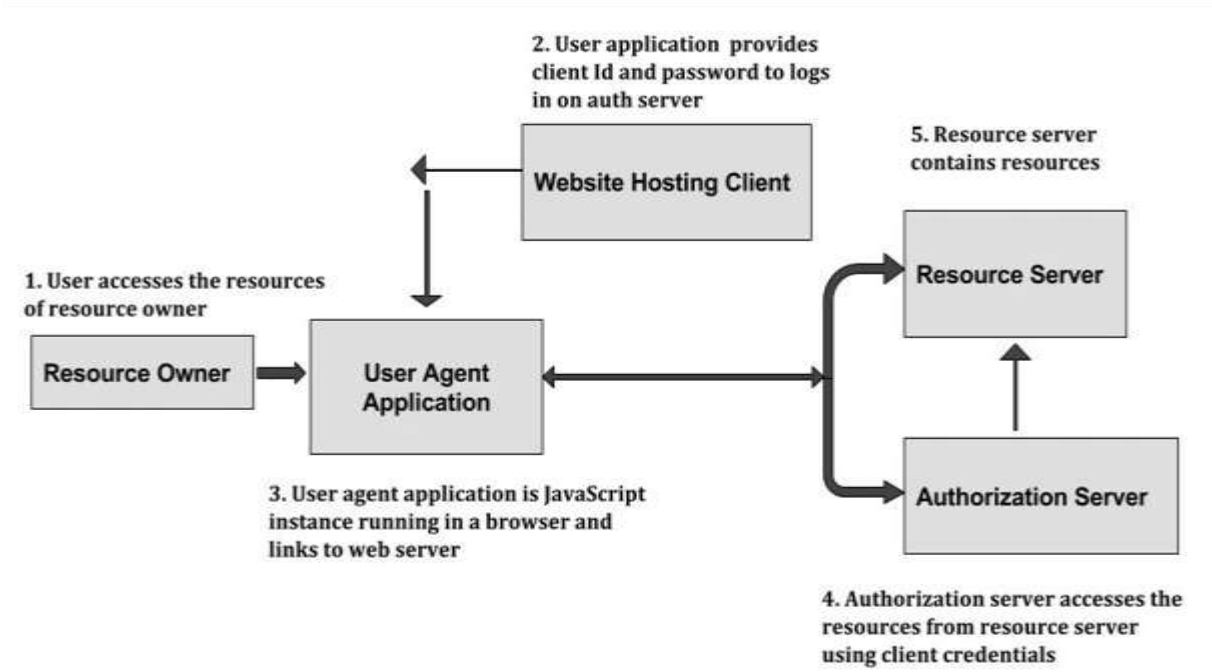


- In the above figure, the resource owner allows the confidential client to access the data that is hosted on the resource server, where client ID and password are kept confidential on the server.
- The client ID and password is unique to the client application on the authorization server.
- The resource server is a server, which hosts the resources such as Facebook, Twitter, Google, etc. These resources are stored on the resource server and are accessed by the client application and the resource owner owns these resources.
- The resources of the resource owner are then accessed by the authorization server using confidential client web application.

User Agent

The user agent application is used by the client applications in the user's device, which acts as the scripting language instance such as JavaScript running in a browser. You can store the user agent application on a web server.

The following diagram shows the architecture of the client user agent application.



Step 1: First, the user accesses the resources of the resource owner by using authenticating application such as Google, Facebook, Twitter, etc.

Step 2: Next, the user application provides the client Id and client password to log on to the authorization server.

Step 3: Then, the user agent application provides an instance of a JavaScript application running in a browser and links to the web server.

Step 4: The authorization server allows access to the resources from the resource server using the client credentials.

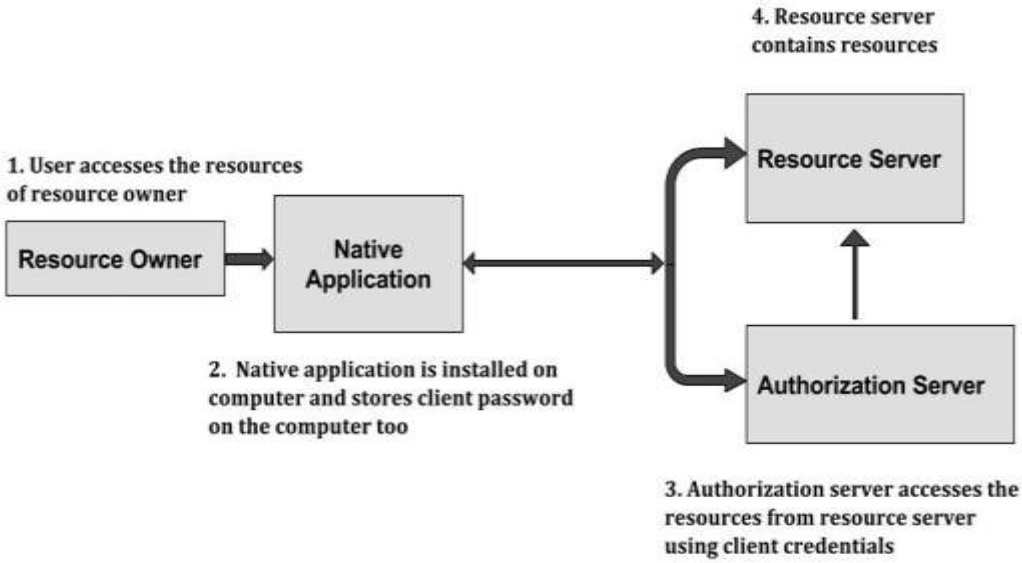
Step 5: The resource server contains the resources, which are owned by the resource owner.

Native Application

Native application can be used as instance of desktop or mobile phone application, which uses the resource owner credentials. It is a public client installed that executes on the resource's owner device.

The authentication credentials used by the application are included in the application code. Hence, do not use the native application that runs in the external user agents.

The following diagram shows the architecture of the client native application:



Step 1: First, the user accesses the resources of the resource owner by using authenticating application such as Google, Facebook, Twitter, etc.

Step 2: Next, the native application uses client Id and client password to log on to the authorization server. The native application is an instance of desktop or mobile phone application, which is installed on the user computer and stores the client password on the computer or device.

Step 3: The authorization server allows accessing the resources from the resource server using the client credentials.

Step 4: The resource server contains the resources, which are owned by the resource owner.

3. OAuth 2.0 – Client Credentials

The client credentials can be used as an authorization grant when the client is the resource owner, or when the authorization scope is limited to protected resources under the control of the client.

- The client requests an access token only with the help of client credentials.
- The client credentials authorization flow is used to acquire access token to authorize API requests.
- Using client credentials authorization, access token which is acquired, only grants permission for your client application to search and get catalog documents.

The following figure depicts the Client Credentials Flow.

The flow illustrated in the above figure consists of the following steps:

- **Step 1**: The client authenticates with the authorization server and makes a request for access token from the token endpoint.
- **Step 2**: The authorization server authenticates the client and provides access token if it's valid and authorized.

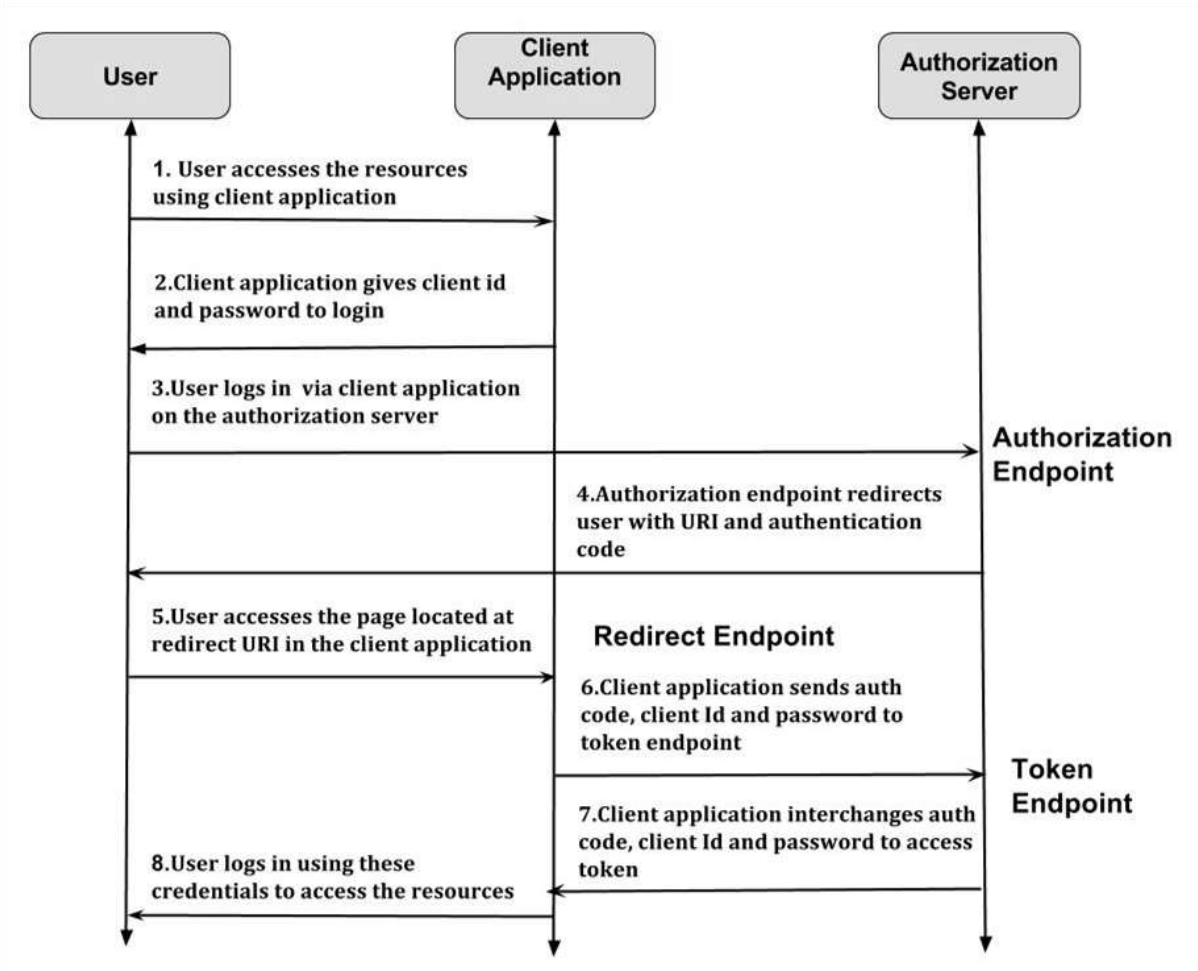
The following table lists the concepts of Client Credentials.

Sr. No.	Concept & Description
1	**Obtaining End-User Authorization** The authorization end point is typically URI on the authorization server in which the resource owner logs in and permits to access the data to the client application.
2	**Authorization Response** The authorization response can be used to get the access token for accessing the owner resources in the system using the authorization code.
3	**Error Response and Codes** The authorization server responds with a HTTP 400 or 401 (bad request) status codes, if an error occurs during authorization.

Obtaining End-User Authorization

The authorization end points are the URL's which makes an authentication request on the authorization server, in which the resource owner logs in and permits to access the data to the client application. For instance, address of JSP page, PHP page, etc.

The authorization end user can be described as shown in the following diagram.



The authorization endpoint can be defined in three ways:

- Authorization Endpoint
- Redirect Endpoint
- Token Endpoint

Authorization Endpoint

- Authorization endpoint can be used to interact with the resource owner who permits the authorization to access the resource of the resource owner.

- First, the user accesses the resources of the resource owner by using the client application. The client application will be provided with the client id and client password during registering the redirect URI (Uniform Resource Identifier).

- Next, the user can login via client application on the authorization server. which contains Authorization Endpoint.

- Authorization endpoint redirects the user with URI (Uniform Resource Identifier) and authentication code to the user.

Redirect Endpoint

- The user accesses the page located at redirect URI (Uniform Resource Identifier) in the client application.

- The client application provides client id, client password and authentication code to the authorization server.

Token Endpoint

- At this point, the client application interchanges the client id, client password and authorization code to obtain an access token.

- The client application sends these credentials to the user along with the token. Once the user receives the token, it can be sent to the access resources such as Facebook, Google, etc. to access the resources in the system, related to the logged in users.

Authorization Response

The authorization response can be used to get the access token for accessing the owner resources in the system using the authorization code. The access token is given by the authorization server when it accepts the client ID, client password and authorization code sent by the client application.

The authorization code will be issued by the authorization server, which allows accessing the request by using the following parameters:

- **Code**: It is a required parameter that specifies the authorization code produced by the authorization server. The lifetime of the authorization code is maximum 10 minutes and the authorization code cannot be used more than once. The authorization server rejects the request and cancels all tokens that are issued previously based on the authorization code, if the client application uses the authorization code more than once.

- **State**: It is a required parameter, if the authorization code is available in the authorization request.

The authorization server provides authorization code and grants access to the client application by using the following format:

```
"application/x-www-form-urlencoded"
```

It is the default MIME (Multipurpose Internet Mail Extensions) type of your request, which must be encoded in a such way that control names and values are escaped, space characters are replaced by the '+' sign, name/value pairs are separated from each other by '&', etc.

Error Response and Codes

The authorization server responds with HTTP 400 or 401 status codes. Here, two cases take place, if an error occurs during the authorization. In the first case, the client is not identified or recognized. In the second case, something else fails in spite of the client being identified exactly. In such a case, an error response is sent back to the client as follows:

- **error_description**: It is an optional human readable error description in a language specified by Content-Language header, which is meant for the developer and not the end user.

- **error_uri**: It is an optional link to a human-readable web page along with information about an error that can be helpful for problem solving.

- **error**: It is a set of predefined error codes.

Following is the description of error codes and equivalent HTTP status codes.

400 Errors

The following table shows 400 errors with description.

Sr. No.	Error & Description
1	**unsupported_over_http** OAuth 2.0 only supports the calls over https.
2	**version_rejected** If an unsupported version of OAuth is supplied.
3	**parameter_absent** If a required parameter is missing from the request.
4	**parameter_rejected** When a given parameter is too long.
5	**invalid_client** When an invalid client ID is given.
6	**invalid_request** When an invalid request parameter is given.
7	**unsupported_response_type** When a response type provided does not match that particular request.
8	**unsupported_grant_type** When a grant type is provided that does not match a particular request.
9	**invalid_param** When an invalid request parameter is provided.
10	**unauthorized_client** When the client is not given the permission to perform some action.

11	**access_denied** When the resource owner refuses the request for authorization.
12	**server_error** This error displays an unexpected error.

401 Errors

The following table shows 401 errors with description.

Sr. No.	Error & Description
1	**token_expired** When the provided token expires.
2	**invalid_token** When the provided token is invalid.
3	**invalid_callback** When the provided URI with the request does not match the consumer key.
4	**invalid_client_secret** When the provided client server is invalid.
5	**invalid_grant** When the provided token has either expired or is invalid.

4. OAuth 2.0 – Obtaining an Access Token

An access token is a string that identifies a user, an application, or a page. The token includes information such as when the token will expire and which app created that token.

- First, it is necessary to acquire OAuth 2.0 client credentials from API console.
- Then, the access token is requested from the authorization server by the client.
- It gets an access token from the response and sends the token to the API that you wish to access.

You must send the user to the authorization endpoint at the beginning. Following is an example of a dummy request.

```
    https://public-
api.example.com/oauth2/authorize?client_id=your_client_id&redirect_uri=your_url
&response_type=code
```

Following are the parameters and their descriptions.

- **client_id**: It should be set to the client id of your application.
- **redirect_uri**: It should be set to the URL. After the request is authorized, the user will be redirected back.
- **response_type**: It can either be a code or a token. The code must be used for server side applications, whereas the token must be used for client side applications. In server side applications, you can make sure that the secrets are saved safely.

Following table lists the concepts of Client Credentials.

Sr. No.	Concept & Description
1	**Authorization Code** The authorization code allows accessing the authorization request and grants access to the client application to fetch the owner resources.
2	**Resource Owner Password Credentials** The resource owner password credentials include only one request and one response, and is useful where the resource owner has a good relationship with the client.
3	**Assertion** Assertion is a package of information that makes the sharing of identity and security information across various security domains possible.
4	**Refresh Token**

	The refresh tokens are used to acquire a new access tokens, which carries the information necessary to get a new access token.
5	**Access Token Response** Access token is a type of token that is assigned by the authorization server.
6	**Access Token Error Response and Codes** If the token access request, which is issued by the authorization server is invalid or unauthorized, then the authorization server returns an error response.

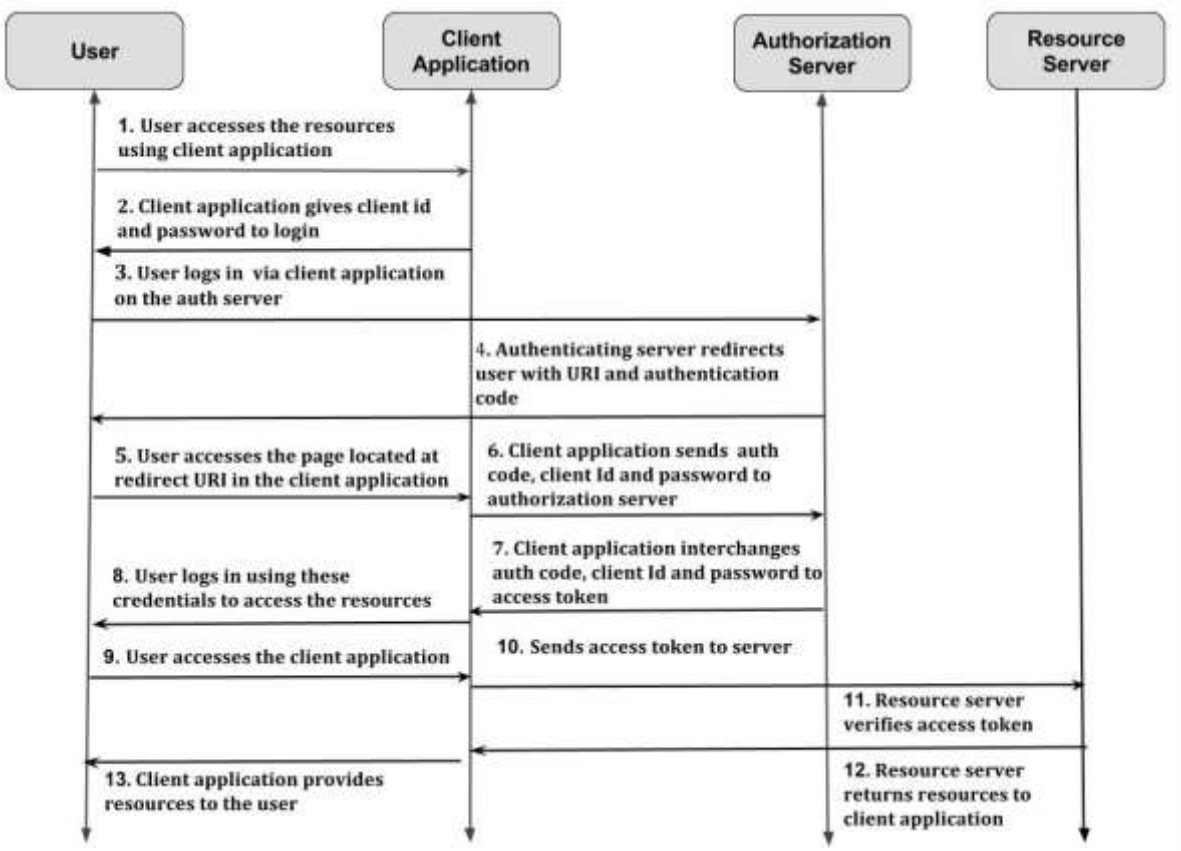
Authorization Code

The authorization code will be issued by the authorization server which allows accessing the authorization request and grants access to the client application to fetch the owner resources.

- The resource owner can be redirected to the client application with the authorization code by directing the owner to the authorization server using the client application.

- The important role of the authorization code is to authenticate the client and access the token directly without passing it to the owner's user agent.

The following diagram shows the process of authorization code.



Step 1: First, the user accesses the resources of the resource owner by using the client application.

Step 2: Next, the client application will be provided with the client id and client password during registering the redirect URI (Uniform Resource Identifier).

Step 3: Then, the user logs in via the client application on the authorization server such as Google, Facebook, Twitter, etc.

Step 4: The authenticating server redirects the user to a redirect Uniform Resource Identifier (URI) using the authorization code which the owner of the client application registers the redirect URI.

Step 5: After registration, the user accesses the redirect URI from the client application.

Step 6: The client application will be provided with the authentication code, client id and client password, and sends them to the authorization server.

Step 7: The client ID and client password is unique to the client application on the authorization server. The authorization server sends an access token to the client application.

Step 8: The user will be allowed to login to the application.

Step 9: The user logs in and accesses the client application using these credentials via the authorization server.

Step 10: It then sends an access token to the resource server.

Step 11: Resource server is the server hosting resources such as Facebook, Google, Twitter, etc. which verifies the access token.

Step 12: Next, the client application accesses the resources stored on the resource server. The resource server returns the resources to the client application.

Step 13: Next, the client application provides the resources to the user.

Resource Owner Password Credentials

The resource owner password credentials include only one request and one response. This grant type is useful where the resource owner has a good relationship with the client, when there are no authorization grant types available.

The resource owner password credentials can be used to grant authorization to access the token. This type of grant type removes the storing of resource owner credentials for future use by exchanging credentials with the access token or the refreshing token.

The resource owner password credentials grant request contains the following parameters:

- **grant_type**: It is a required parameter used to set the password.
- **username**: It is a required parameter that specifies the resource owner username.
- **password**: It is a required parameter that specifies the resource owner password.
- **scope**: It is an optional parameter that specifies the scope of the request and authorization.

The resource owner password credentials grant response contains the following JSON structure.

```
{
   "access_token"  : ". . .",
   "token_type"    : ". . .",
   "expires_in"    : ". . . ",
   "refresh_token" : ". . .",
}
```

- **access_token**: It is a required parameter in which the authorization server accesses the token.
- **token_type**: It is a required parameter which is assigned by the authorization server and specifies the type of token.
- **expires_in**: It is a recommended parameter that specifies the duration of access token expiry.
- **refresh_token**: It provides a refresh token if the access token expires, to get the new access token using the authorization grant.

Assertion

Assertion is a package of information that makes sharing of identity and security information easier across various security domains.

It holds the data about the subject, circumstances under which assertion is considered valid, such as where and when it can be used.

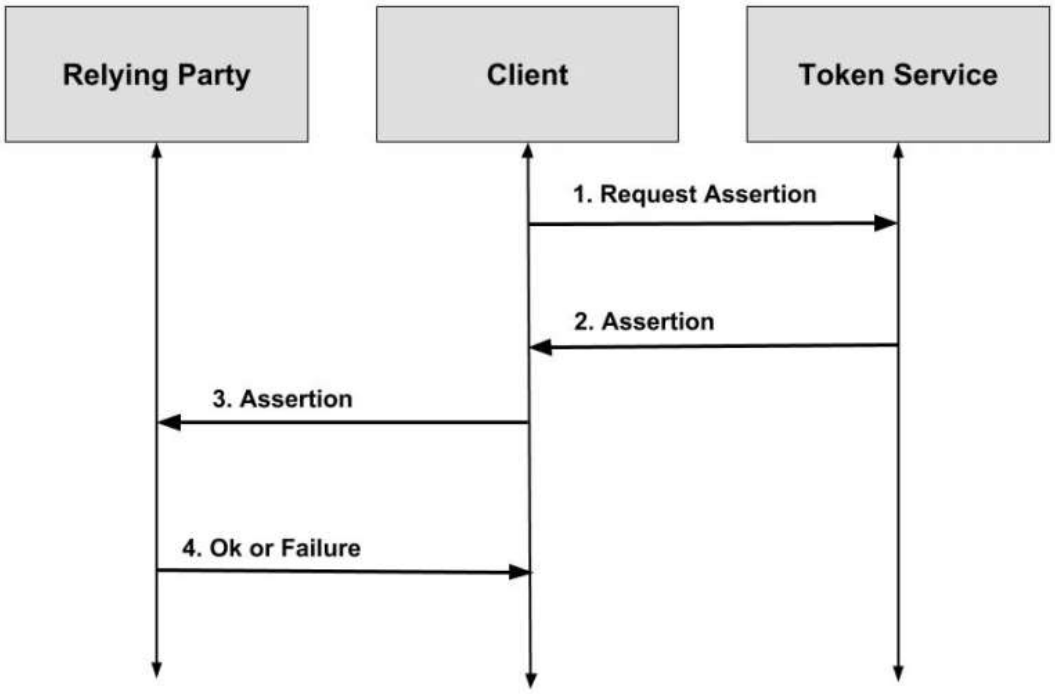
Entity that creates or protects the assertion is called as the **Issuer**.

Entity that consumes the assertion depending on its information is called as the **Relying Party**.

Assertions are of two general types. They are:

- **Bearer Assertions**: Any entity can use assertion to obtain access to the associated resources, wherein an entity can be in-charge of bearer assertion.
- **Holder-of-key Assertions**: In this case, the entity must demonstrate the possession of additional cryptographic material, if it wants to access associated resources.

The following figure depicts the third party created assertion.



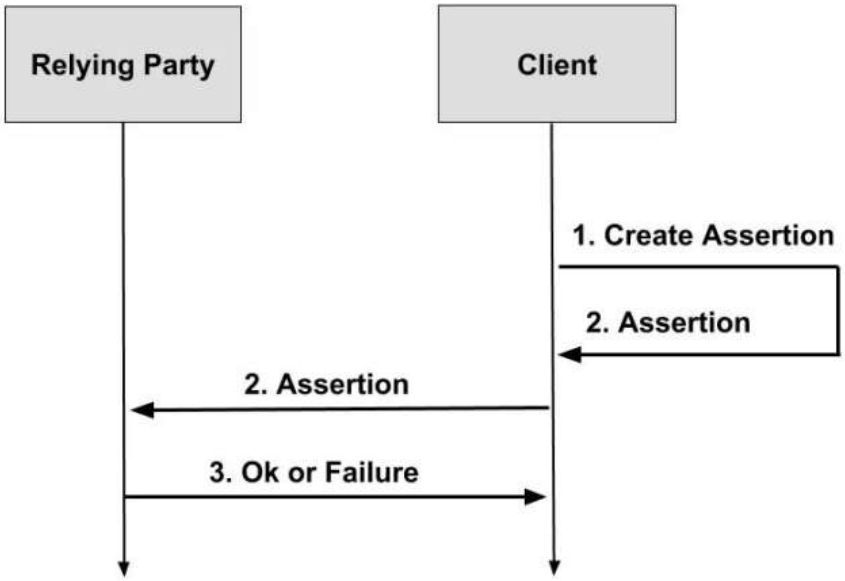
Step 1: This is the first case where the client first requests assertion from third party entity, which is usually known as the "token service" or "security token service". Token service is capable of issuing, renewing, validating, and transforming security tokens to the client.

Step 2: The token service fulfils the client's request by granting assertion.

Step 3: A trust relationship exists between the token service and the relying party. The client then issues assertion to the relying party.

Step 4: The relying party validates assertion and notifies the client about the status.

The following figure depicts the self-issued assertion.



Step 1: This is the second case in which the client itself creates assertion locally. It doesn't have to request for assertion from the third party entity.

Step 2: The client then issues the created assertion to the relying party.

Step 3: The relying party validates assertion and notifies the client about the status.

Using Assertions as Authorization Grants

Using the following HTTP request parameters, the client includes the assertion and related information when using the assertions as authorization grants.

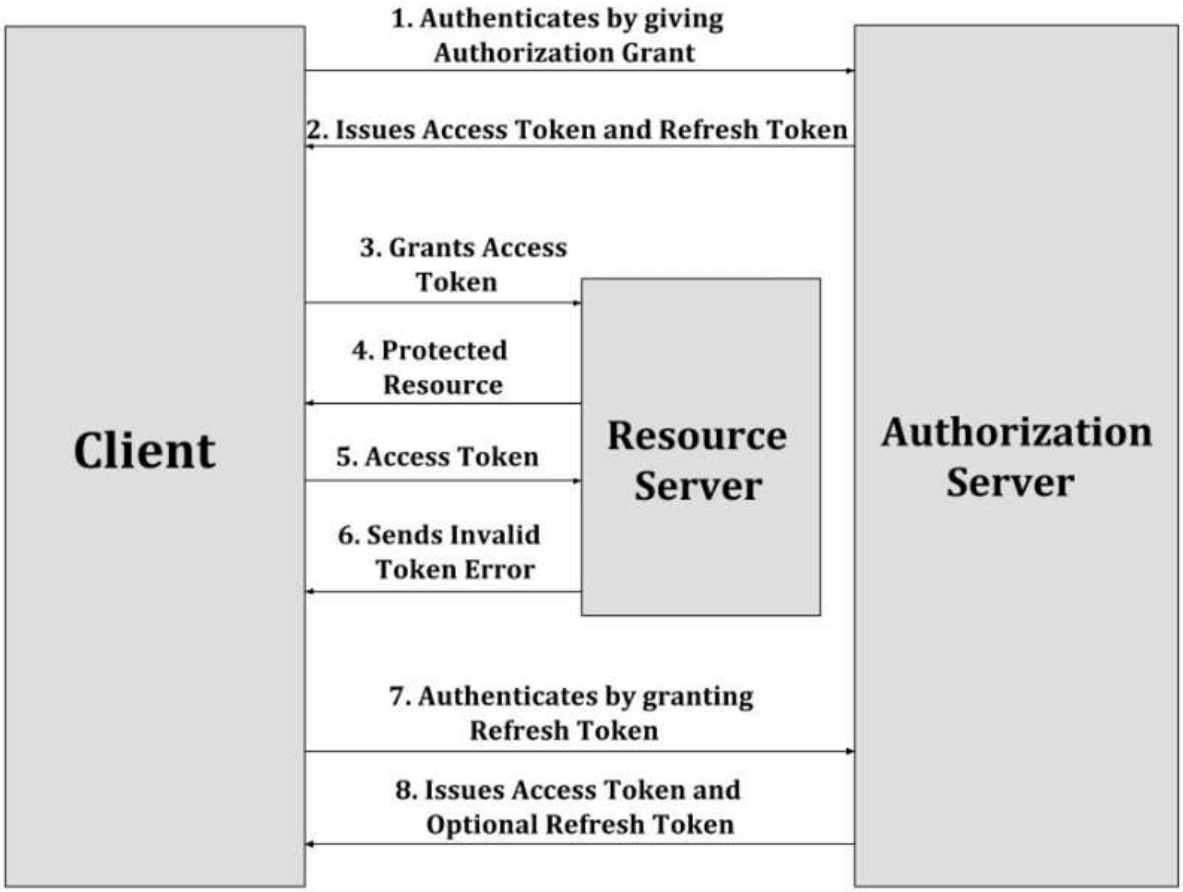
- **grant_type**: The format of the assertion that is defined by the authorization server.
- **assertion**: A particular serialization of the assertion defined by the profile documents.
- **scope**: The authorization of the token is previously granted through some out-of-band mechanism while exchanging the assertions for access tokens. In that case, the scope that is requested must be equal to or less than the original scope granted to the authorized accessor.

Refresh Token

Refresh tokens are the credentials that can be used to acquire new access tokens.

- The lifetime of a refresh token is much longer compared to the lifetime of an access token.
- Refresh tokens can also expire but are quiet long-lived.
- When current access tokens expire or become invalid, the authorization server provides refresh tokens to the client to obtain new access token.

The following figure illustrates the process of refreshing an expired Access Token.



Step 1: First, the client authenticates with the authorization server by giving the authorization grant.

Step 2: Next, the authorization server authenticates the client, validates the authorization grant and issues the access token and refresh token to the client, if valid.

Step 3: Then, the client requests the resource server for protected resource by giving the access token.

Step 4: The resource server validates the access token and provides the protected resource.

Step 5: The client makes the protected resource request to the resource server by granting the access token, where the resource server validates it and serves the request, if valid. This step keeps on repeating until the access token expires.

Step 6: If the access token expires, the client authenticates with the authorization server and requests for new access token by providing refresh token. If the access token is invalid, the resource server sends back the invalid token error response to the client.

Step 7: The client authenticates with the authorization server by granting the refresh token.

Step 8: The authorization server then validates the refresh token by authenticating the client and issues a new access token, if it is valid.

Access Token Response

Access token is a type of token that is assigned by the authorization server. The authorization server issues the access token if the access token request is valid and authorized. If the token access request is invalid or unauthorized, then the authorization server returns an error response.

The access token is given by the authorization server when it accepts the client ID, client password and authorization code sent by the client application. Once the user receives the token, it can be sent to the access resources such as Facebook, Google, etc. to access the resources in the system, related to the logged in users.

The access token response contains the following JSON structure.

```
{
   "access_token"  : ". . .",
   "token_type"    : ". . .",
   "expires_in"    : ". . . ",
   "refresh_token" : ". . .",
}
```

- **access_token**: It is a required parameter in which the authorization server accesses the token.
- **token_type**: It is a required parameter which is assigned by the authorization server and specifies the type of token.
- **expires_in**: It is a recommended parameter that specifies the duration of access token expiry.
- **refresh_token**: It provides a refresh token, if the access token expires, to get the new access token using the authorization grant.

Access Token Error Response and Codes

Access token is a type of token that is assigned by the authorization server. The authorization server issues the access token, if the access token request is valid and authorized. If the token access request is invalid or unauthorized, then the authorization server returns an error response.

For information on access token response, click this link

Access Token Response

Access token is a type of token that is assigned by the authorization server. The authorization server issues the access token, if the access token request is valid and authorized. If the token access request is invalid or unauthorized, then the authorization server returns an error response.

The access token is given by the authorization server when it accepts the client ID, client password and authorization code sent by the client application. Once the user receives the token, it can be sent to the access resources such as Facebook, Google, etc. to access the resources in the system, related to the logged in users.

The access token response contains the following JSON structure.

```
{
   "access_token"  : ". . .",
   "token_type"    : ". . .",
   "expires_in"    : ". . . ",
   "refresh_token" : ". . .",
}
```

- **access_token**: It is a required parameter in which the authorization server accesses the token.
- **token_type**: It is a required parameter which is assigned by the authorization server and specifies the type of token.
- **expires_in**: It is a recommended parameter that specifies the duration of access token expiry.
- **refresh_token**: It provides a refresh token if the access token expires, to get the new access token using the authorization grant.

Error Response

The application can handle error response by sending them to ***redirect_uri***.

For instance:

```
GET
http://www.site.com/?error=access_denied&error_description=the+user+canceled+au
thentication
```

The above URI contains the following parameters:

- **error**: It specifies the error code if there is an invalid request, invalid client, invalid grant, or unauthorized client.
- **error_description**: It defines the detail description of the error.

Following are the various error codes, which can occur when there are errors at the authorization endpoint.

Sr. No.	Error & Description	Error Code
1	**invalid_request** This error occurs when there is a missing parameter that includes multiple credentials, unsupported parameter value.	400
2	**unauthorized_client** The unauthorized client is not allowed to access the authorization grant type.	401
3	**access_denied** It specifies the user will have no access permission to files or subfolders.	401
4	**unsupported_response_type** It specifies the response type is not supported by the authorization server.	415
5	**server_error** This error code is mainly used when 500 internal server cannot be returned to the client by using HTTP redirect.	500
6	**temporarily_unavailable** It specifies that the server is unable to handle the request during overloading of server or during server maintenance.	503

5. OAuth 2.0 – Accessing a Protected Resource

The client provides an access token to the resource server to access protected resources. The resource server must validate and verify that the access token is valid and has not expired.

There are two standard ways of sending credentials:

- **Bearer Token**: The access token can only be placed in POST request body or GET URL parameter as a fallback option in the authorization HTTP header.

  They are included in the authorization header as follows:

```
Authorization: Bearer [token-value]
```

  For example:

```
GET/resource/1 HTTP /1.1
Host: example.com
Authorization: Bearer abc...
```

- **MAC:** A cryptographic **Message Authentication Code** (MAC) is computed using the elements of the request and is sent to the authorization header. Upon receiving the request, the MAC is then compared and computed by the resource owner.

The following table shows the concepts of accessing protected resource.

Sr. No.	Concept & Description
1	**Authenticated Requests** It is used to get the authorization code token for accessing the owner resources in the system.
2	**WWW-Authenticate Response Header Field** The resource server includes the "WWW-Authenticate" response header field, if the protected resource request contains an invalid access token.

Authenticated Requests

The authenticated request can be used to get the authorization code token for accessing the owner resources in the system. The request made to the authorization endpoint results in the user authentication and provides clear credentials when sending a request to the authorization endpoint.

The authenticated request contains the following parameters:

- **response_type**: It is a required parameter used to set the value as 'code' which is used for requesting the authorization code. If there is no 'response_ type' parameter in the authorization request, then the authorization server returns an error response. The authorization request may fail due to invalid or mismatch redirect URI or an invalid client identifier.

- **client_id**: It is a required parameter that identifies the client, which is assigned by the authorization server. This is unique to the authorization server. The authorization server may take any type of credentials by gathering its security requirements. The client application should not use more than one authentication method in each request.

- **redirect_uri**: It is an optional parameter, which includes redirection URI with the authorization request. When the authorization request includes the redirection URI, it matches the value of the registered redirection URIs.

- **scope**: It is an optional parameter that specifies the scope of the request. The authorization grant can be used as client credentials, when the authorization scope is restricted to control the protected resources of the client. The scope parameter should not include the resource owner information because they may communicate with the insecure channel or can be stored insecurely.

- **state**: It is an optional parameter. The state value can be used when redirecting the user agent back to the client by using the authorization server. If the authorization request includes state value, then it returns the exact value from the client.

WWW-Authenticate Response Header Field

The resource server must include the HTTP **"WWW-Authenticate"** response header field, if the protected resource request contains an access token that is invalid or if the access token is malformed.

"WWW-Authenticate" header field uses the following format:

```
challenge              =   "OAuth" RWS token-challenge
token-challenge        =   realm
                           [CS error]
                           [CS error-uri]
                           [CS scope]
                           [CS 1#auth –param]
```

```
    error                    =   "error"  "=" <"> token <">
    error-desc               =   "error_description" "=" quoted-string
    error-uri                =   "error_uri" = <"> URI-Reference <">
    scope              =   quoted-value / <"> quoted-value *(1*SP quoted-value)
<">
    quoted-value             =   1* quoted-char
```

where,

- **realm**: It is an attribute which specifies the scope of protection and is displayed to the users so that they know which username and password to use. This attribute must appear only once.

- **error**: It is an attribute used to provide a client the specific reason why the access request was declined.

- **error_description**: It is an attribute that provides a human-readable text that can be used to help in understanding the error that occurred.

- **error_uri**: It is an attribute that provides a URI to identify a human-readable web page along with the information about the error that has occurred.

- **scope**: It is an attribute which specifies the required scope of the access token in order to access the requested resource.

6. OAuth 2.0 – Extensibility

There are two ways in which the access token types can be defined:

- By registering in the access token type's registry.
- By using a unique absolute URI (Uniform Resource Identifier) as its name.

Defining New Endpoint Parameters

Parameter names must obey the param-name ABNF (Augmented Backus-Naur Form is a metalanguage based on Backus-Naur Form consisting of its own syntax and derivation rules) and the syntax of parameter values must be well-defined.

```
param-name = 1* name-char
name-char = "-" / "." / "_" / DIGIT / ALPHA
```

Defining New Authorization Grant Types

New authorization grant types can be assigned a distinct absolute URI for use, with the help of "grant_type" parameter. The extension grant type must be registered in the OAuth parameters registry, if it requires additional token endpoint parameters.

Defining New Authorization Endpoint Response Types

```
response-type = response-name *(SP response-name)
response-name = 1* response-char
response-char = "_" / DIGIT / ALPHA
```

The response type is compared as space-delimited list of values, if it has one or more space characters where the order of the values does not matter and only one order of value can be registered.

Defining Additional Error Codes

The extension error codes must be registered, if the extensions they use are either a registered access token, or a registered endpoint parameter. The error code must obey the error ABNF (Augmented Backus-Naur Form) and when possible it should be prefixed by a name identifying it.

```
error = 1 * error_char
error-char =  %x20-21 / %x23-5B / 5D-7E
```

7. OAuth 2.0 – IANA Considerations

IANA stands for **I**nternet **A**ssigned **N**umbers **A**uthority which provides the information about the registration values related to the **R**emote **A**uthentication **D**ial **I**n **U**ser **S**ervice (RADIUS).

IANA includes the following considerations:

OAuth Access Token Types Registry

OAuth access tokens are registered by experts with required specification. If they are satisfied with the registration, only then they will publish the specification. The registration request will be sent to the @ietf.org for reviewing with the subject ("Request for access token type: example"). Experts will either reject or accept the request within 14 days of the request.

Registration Template

The registration template contains the following specifications:

- **Type Name**: It is the name of the request.
- **Token Endpoint Response Parameters**: The additional access token response parameter will be registered separately in OAuth parameters registry.
- **HTTP Authentication Scheme**: The HTTP authentication scheme can be used to authenticate the resources by using the access token.
- **Change Controller**: Give the state name as "IETF" for standard track RFCs, and for others use the name of the responsible party.
- **Specification Document**: The specification document contains the parameter that can be used to retrieve a copy of the document.

OAuth Parameters Registry

OAuth parameters registry contains registration of authorization endpoint request or response, token endpoint request or response by the experts with the required specification. The registration request will be sent to the experts and if they are satisfied with registration, then they will publish the specification.

Registration Template

The registration template contains specifications such as *Type Name*, *Change Controller* and *Specification Document* as defined in the above *OAuth Access Token Types Registry* section, except the following specification:

Parameter Usage Location: It specifies the location of the parameter such as authorization request or response, token request or response.

Initial Registry Contents

The following table shows OAuth parameters registry containing the initial contents:

Sr. No.	Parameter Name & Usage Location	Change Controller	Specification Document
1	**client_id** authorization request, token request	IETF	**RFC 6749**
2	**client_secret** token request	IETF	**RFC 6749**
3	**response_type** authorization_request	IETF	**RFC 6749**
4	**redirect_uri** authorization request, authorization	IETF	**RFC 6749**
5	**scope** authorization request or response, token request or response	IETF	**RFC 6749**
6	**state** authorization request or response	IETF	**RFC 6749**
7	**code** token request, authorization response	IETF	**RFC 6749**
8	**error_description** authorization response, token response	IETF	**RFC 6749**
9	**error_uri** authorization response, token response	IETF	**RFC 6749**
10	**grant_type** token request	IETF	**RFC 6749**
11	**access_token** authorization response, token response	IETF	**RFC 6749**

12	**token_type** authorization response, token response	IETF	**RFC 6749**
13	**expires_in** authorization response, token response	IETF	**RFC 6749**
14	**username** token request	IETF	**RFC 6749**
15	**Password** token request	IETF	**RFC 6749**
16	**refresh_token** token request, token response	IETF	**RFC 6749**

OAuth Authorization Endpoint Response Type Registry

This can be used to define OAuth Authorization Endpoint Response Type Registry. The response types are registered by experts with the required specification and if they are satisfied with the registration, only then they will publish the specification. The registration request will be sent to the @ietf.org for reviewing. The experts will either reject or accept the request within 14 days of the request.

Registration Template

The registration template contains specifications such as *Type Name*, *Change Controller* and *Specification Document* as defined in the above *OAuth Access Token Types Registry* section.

Initial Registry Contents

The following table shows the authorization endpoint response type registry containing the initial contents.

Sr. No.	Parameter Name	Change Controller	Specification Document
1	code	IETF	**RFC 6749**
2	token	IETF	**RFC 6749**

OAuth Extensions Error Registry

This can be used to define OAuth Extensions Error Registry. The error codes along with protocol extensions such as grant types, token types, etc. are registered by experts with the required specification. If they are satisfied with the registration, then they will publish the specification. The registration request will be sent to the @ietf.org for reviewing with subject ("Request for error code: example"). Experts will either reject or accept the request within 14 days of the request.

Registration Template

The registration template contains specifications such as *Change Controller* and *Specification Document* as defined in the above *OAuth Access Token Types Registry* section, except the following specifications:

- **Error Name**: It is the name of the request.
- **Error Usage Location**: It specifies the location of the error such as authorization code grant error response, implicit grant response or token error response, etc. which specifies where the error can be used.
- **Related Protocol Extension**: You can use protocol extensions such as extension grant type, access token type, extension parameter, etc.