**Reference URLs:**

<https://aaronparecki.com/oauth-2-simplified/>

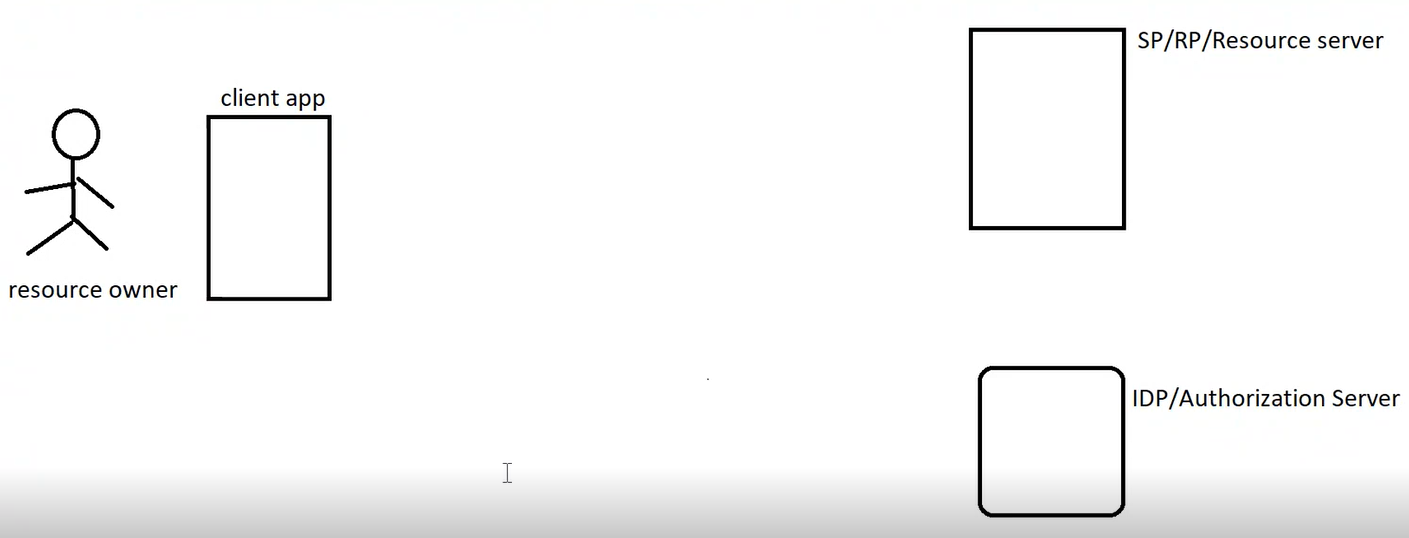
<https://aaronparecki.com/oauth/>

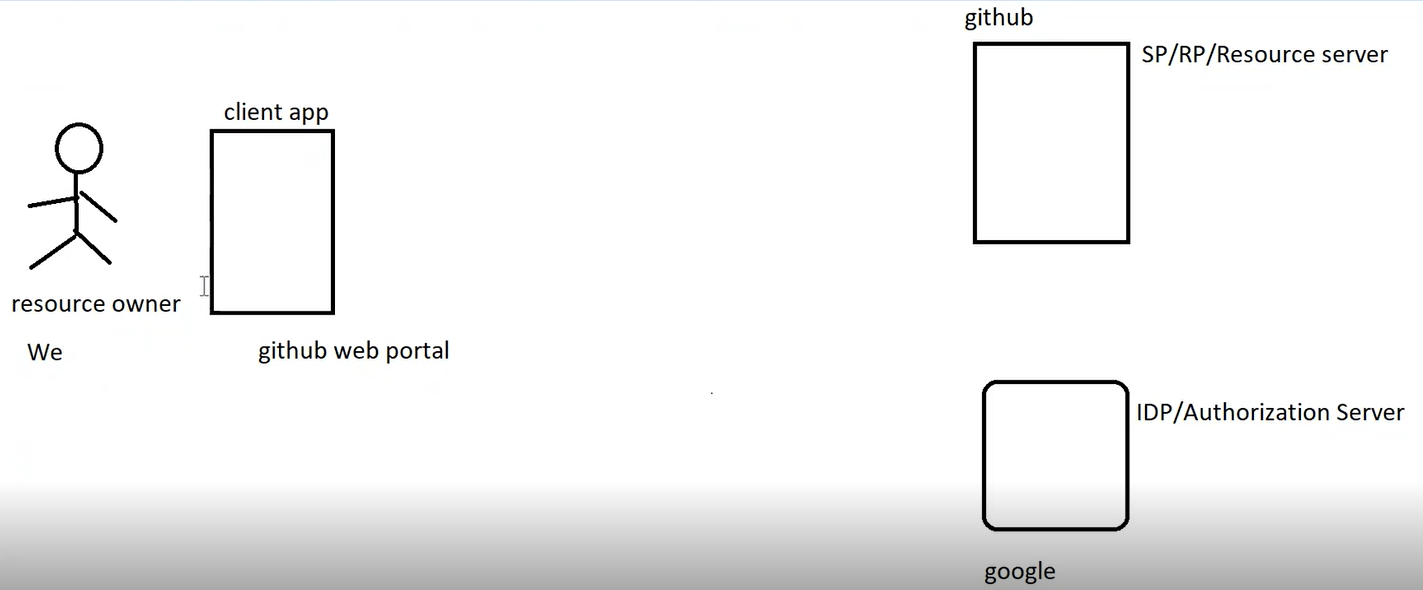
<https://athiththan11.medium.com/oauth-2-grant-types-a-story-guide-582580a3c4c2>

**Federated Security:**

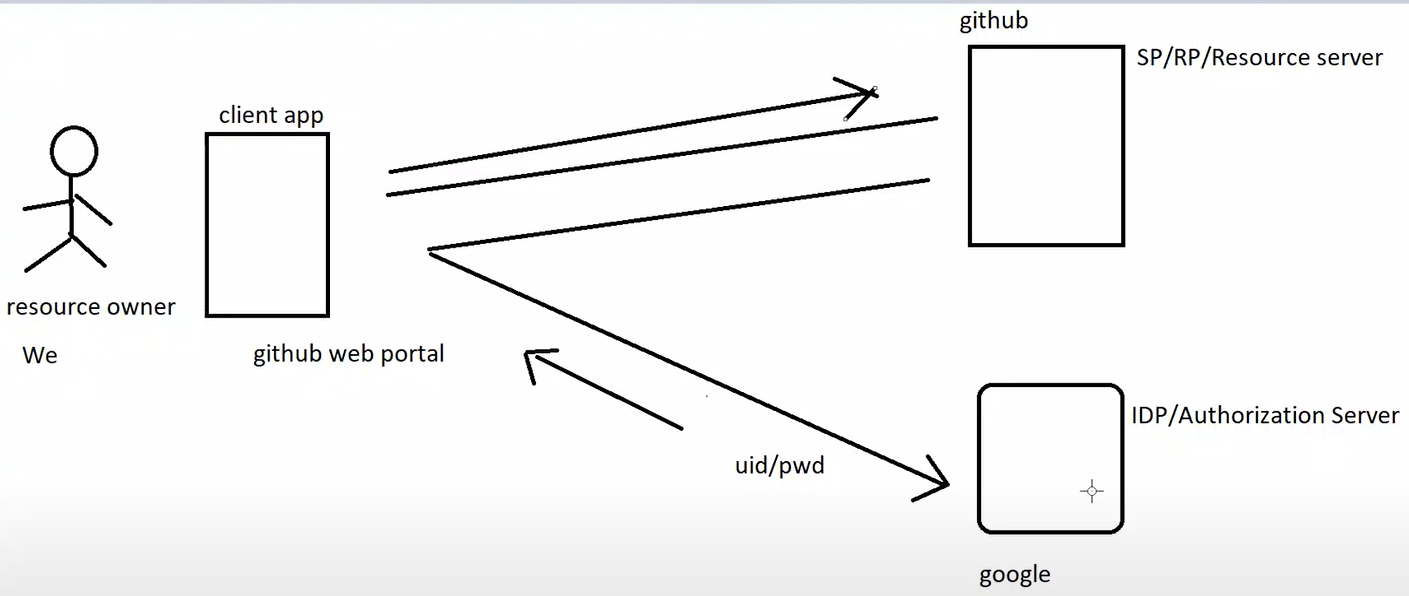
**Session 1:**

**Parties:**





**How it Works?**





For two applications to communicate with each other, they should sync up with each other. Things like protocol of communication, data interchange format etc. should be same.

There are multiple stakeholders to implement federated security:

**OAuth (Open Authorization):** Standard and Specification that application should follow to implement federated security.

OAuth specifies how the communication should happen between the entities so that federated security can be implemented.

**Google Search: OAuth Aaron Parecki**

**Aaron Parecki:** One of the founding members/contributors in OAuth specification.

**Reference URL:** https:// 

OAuth library is available in almost all technologies.

Grant Types are also called Work Flows.

Grant Type is practical situation/scenario in OAuth.

response\_type=code means flow in the URL diagram will be followed.

client\_id: Client Id of the app. E.g. – Medium.com

redirect\_uri: Response will come to this URL.

scope\_id: Details sought about end user. E.g. – FirstName, LastName, ContactNo, Email etc.

state: Random value that service provider should generate and send to IDP.

**Client Credentials Grant Type:**



Session 2:

**Demo:**

**IDP Project:**



OAuth Library created by Microsoft for .NET Core applications.

IdentityServer as an external service has been configured and added as a middleware.

**Config.cs:**

**Inputs given to Identity Server:**

**Ids:** All the details IDP/Authorization Server maintains about end user. E.g. – Google tracks out online activity.

**OpenId:** Metadata like SSID etc.

**Profile:** First Name and Last Name

**Address:** Permanent Address

**Additional Information:** Information that IDP has about end users. Like role assigned to resource owner, role to which end user belongs, hobbies etc.

**Apis:** List of backend services that can be authorized by IDP.

Multiple Apis can be specified using comma separated list.

**OpenId:** How authentication should work in federated security

**OAuth:** How authorization should work in federated security.

**OAuth 2.0 : OAuth + OpenId**

OpenId is merged with OAuth.

**Clients:** Contains registration code.

Currently details are stored in memory.

**TestUser.cs**

given\_name, family\_name : Profile

address: address

role: roles

IDP application has Controllers for different purpose.

**Client Application:**

Authentication is configured in ConfigureServices() method and Injected in the Pipeline in Configure() method.

OAuth 2.0 used OpenId for authentication and add authorization to it.

IDP address is specified in AddOpenIdConnect() method.

options.Scope.Add("address");

options.Scope.Add("roles");

Address and Role of the end user is supposed to be sent in the token.

options.Scope.Add("imagegalleryapi");

Details about this API should be there. This code corresponds to the following code in IDP Application:

public static IEnumerable<ApiResource> Apis =>

new ApiResource[]

{

new ApiResource(

"imagegalleryapi",

"Image Gallery API",

new List<string>() { "role" })

};

Lot of Metadata in the token is created. Below code in Client Application removes these details/metadata:

options.ClaimActions.DeleteClaim("sid");

options.ClaimActions.DeleteClaim("idp");

options.ClaimActions.DeleteClaim("s\_hash");

options.ClaimActions.DeleteClaim("auth\_time");

options.ClaimActions.MapUniqueJsonKey("role", "role");

The above code maps the role.

public static IEnumerable<IdentityResource> Ids =>

new IdentityResource[]

{

new IdentityResources.OpenId(),

new IdentityResources.Profile(),

new IdentityResources.Address(),

new IdentityResource(

"roles",

"Your role(s)",

new List<string>() { "role" })

};

The token will contain the data element named role. That data element will be mapped to second role parameter in MapUniqueJsonKey() method.

It relates with the following code:

options.TokenValidationParameters = new TokenValidationParameters

{

NameClaimType = JwtClaimTypes.GivenName,

RoleClaimType = JwtClaimTypes.Role

};

User.Identity.Name : Name of the logged in user/Name of the Resource Owner.

GivenName is NameClaimType. Indirectly what does it mean?

User.Identity.Name should be polulated by GivenName.

role in MapUniqueJsonKey() method should be populated by JwtClaimTypes.Role.

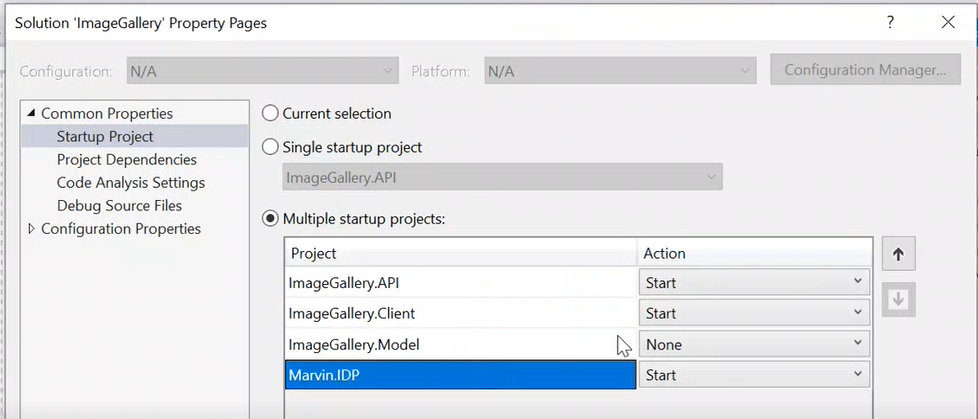
ClientApplication is configured to use Cookie Based Authentication and it is ready to Communicate with IDP.

GalleryController:

AddImage() method can only be accessed by people belonging to PayingUser role.

InTestUser.cs file, there is one free user and one PayingUser.

Set multiple startup projects as shown below:



Run the project.

After hitting the Client URL, observe that you are redirected to 44318 (IDP URL).

When request comes, application tries to find cookie or token. If it doesn’t find either of them, then user is redirected to IDP.

Token is obtained by UseAuthentication() middleware and once the token is obtained, token is saved as a cookie and that cookie is sent to the browser.

Whatever operations user performs on the website after that, user is identified using that cookie.

ImageController is like a backend service to GalleryController.

HttpClientFactory is used for creating HttpClient instances. It is configured in ConfigureServices() method.