REST API in APEX

Architectural constraints/Characteristics

Stateless

Each request from client to server must contain all the information necessary to understand the request, and not use any stored context on the server.

Caching behavior

Responses are labeled as cacheable or non-cacheable.

Uniform interface

All resources are accessed with a generic interface over HTTP.

Named resources

All resources are named using a base URI that follows your Lightning Platform URI.

Layered components

allows for the existence of such intermediaries as proxy servers and gateways to exist between the client and the resources.

Authentication

Uses Oath2.0.

Support for JSON and XML

JSON is the default. You can use the `HTTP ACCEPT` header to select either JSON or XML, or append `json` or `xml` to the URI

Compression

The REST API allows the use of compression on the request and the response, using the standards defined by the HTTP 1.1 specification. Compression is automatically supported by some clients, and can be manually added to others. For better performance.

Request compression: Add **`Accept-Encoding: gzip`** or **`Accept-Encoding: deflate`** in a request Header.

Response compression : in response header **`Content-Encoding: gzip/Content-Encoding: deflate`**

Authorization Through Connected Apps and OAuth 2.0

A connected app requests access to REST API resources on behalf of the client application.

OAuth 2.0 is an open protocol that authorizes secure data sharing between applications through the exchange of tokens.
OAuth authorization flows grant a client app restricted access to REST API resources on a resource server. Each OAuth flow offers a different process for approving access to a client app, but in general the flows consist of three main steps.

1. a connected app, on behalf of a client app, requests access to a REST API resource.
2. In response, an authorizing server grants access tokens to the connected app.
3. A resource server validates these access tokens and approves access to the protected REST API resource.

Perform Cross-Origin Requests from Web Browsers

Suppose a user visits http://www.example.com and the page attempts a cross-origin request to fetch the user's data from http://service.example.com. A CORS-compatible browser will attempt to make a cross-origin request to service.example.com.

In Salesforce, add the origin serving the code to a CORS allowlist.

1. enter `CORS` in the `Quick Find` box, then select **CORS**
2. **New**
3. Enter a URL Pattern

   You can add (*) wildcard at 2nd level domain, Example : `https://*.example.com`.

If a browser that supports CORS makes a request to an origin in the allowlist, Salesforce returns the origin in the **Access-Control-Allow-Origin** HTTP header along with any additional CORS HTTP headers. If the origin isn’t included in the allowlist, Salesforce returns HTTP status code 403.

Connected App and OAuth Terminology

https://help.salesforce.com/articleView?id=sf.remoteaccess_terminology.htm&type=5

Access Token: Instead of using the user's Salesforce credentials, a consumer (connected app) can use an access token to gain access to protected resources on behalf of the user. For OAuth 2.0, the access token is a session ID and can be used directly.

Authorization Code: In Oath 2.0, The authorization code is used to obtain an access token and a refresh token. It expires after 15 minutes.

Authorization Server: server that authorizes a resource owner, and upon successful authorization, issues access tokens to the requesting consumer.

Callback URL: A callback URL is the URL that is invoked after OAuth authorization for the consume.

OAuth Endpoint: OAuth endpoints are the URLs that you use to make OAuth authorization requests to Salesforce.

Consumer : A consumer is the website or app that uses OAuth to authorize both the Salesforce user and itself on the user's behalf. Referred to as client in OAuth 2.0.

Consumer Key :A consumer uses a key to identify itself to Salesforce. Referred to as client_id in OAuth 2.0.

Consumer Secret : A consumer uses a secret to establish ownership of the consumer key. Referred to as client_secret in OAuth 2.0.

Refresh Token : Only used in OAuth 2.0, a consumer can use a refresh token to obtain a new access token, without having the end user approve the access again.

Resource Owner: The resource owner is the entity (usually the end user) that grants access to a protected resource.

Resource Server : The resource server is the server that hosts the protected resource. Your Salesforce org is the resource server that protects your data.

Token Secret : A consumer uses this secret to establish ownership of a given token, both for request tokens and access tokens.

Connected Apps

A connected app is a framework that enables an external application to integrate with Salesforce using APIs and standard protocols, such as Security Assertion Markup Language (SAML), OAuth, and OpenID Connect. Connected apps use these protocols to authorize, authenticate, and provide single sign-on (SSO) for external apps.

Developers create and configure authorization flows for connected apps, and admins set policies and permissions to control connected app usage.

1. **Access Data with API Integration**: web-based or mobile applications that need to pull data from your Salesforce org, you can use connected apps as the clients to request this data. Use OAuth 2.0 **Security**.
   a. Web Server integration – Web -> connected app -> Salesforce
   b. Mobile integration -> mobile to salesforce/ can use mobile SDK instead of connected app.
   c. Server to Server integration -> no need to be real time, can do authorization and authentication in advance by sending JSON WEB TOKEN or JWT
   d. IOT integration -> for devices like TV , having limited input capabilities.

2. **Integrate Service Providers with Salesforce**: SSO
   a. You want your users to be able to log in to their app with their Salesforce credentials. Using SAML 2.0 for user **authentication**.
   b. OpenID Connect is a protocol that enables SSO between two services, like SAML2.0. It adds an authentication layer on top of OAuth 2.0 to enable secure exchange of ID tokens that contain user information alongside OAuth access tokens.

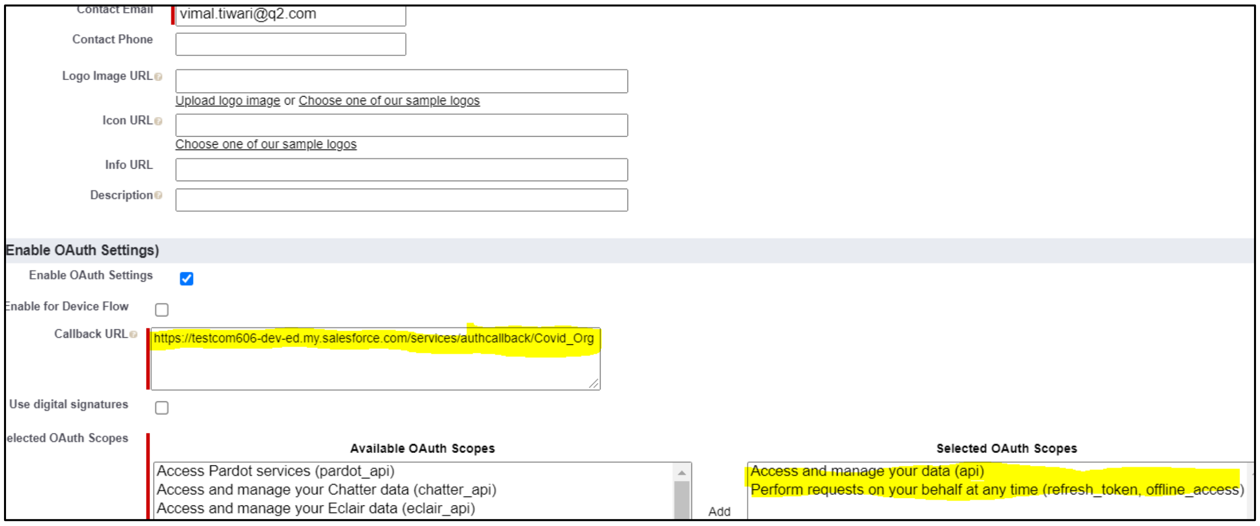
3. **Provide Authorization for External API Gateways:** Using OpenID Connect dynamic client registration, resource servers can dynamically create client apps as connected apps in Salesforce, by sending request to authorization server. The authorization server

verifies the resource server's request and creates the connected app, giving it a unique client ID and client secret.

Basic Rest Based data movement Demo using Connected App and OAuth2.0

Steps for Org 1 acting as Server.

1. In response Server i.e. your org providing service, create Apex class for [Account Rest Services]()
2. Create a connected App there.



Contact Email	vimal.tiwari@q2.com
Contact Phone	
Logo Image URL	Upload logo image or Choose one of our sample logos
Icon URL	Choose one of our sample logos
Info URL	
Description	

Enable OAuth Settings)

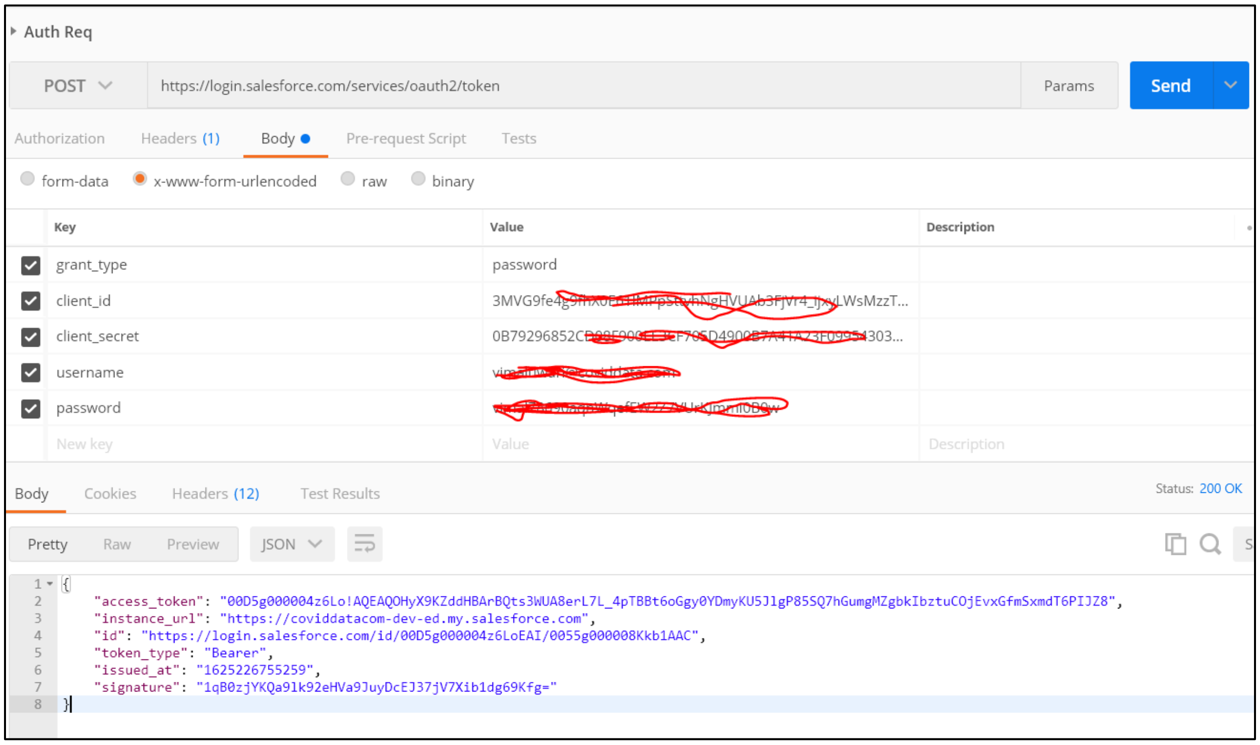
Enable OAuth Settings	✓
Enable for Device Flow	
Callback URL	https://testcom606-dev-ed.my.salesforce.com/services/authcallback/Covid_Org
Use digital signatures	

Selected OAuth Scopes

Available OAuth Scopes	Selected OAuth Scopes
Access Pardot services (pardot_api)	Access and manage your data (api)
Access and manage your Chatter data (chatter_api)	Perform requests on your behalf at any time (refresh_token, offline_access)
Access and manage your Eclair data (eclair_api)	

3. Use [Basic Tutorial of Rest API with Oauth2.0]() for creation and testing Rest Services.

Auth :



Auth Req

POST https://login.salesforce.com/services/oauth2/token

Body: x-www-form-urlencoded

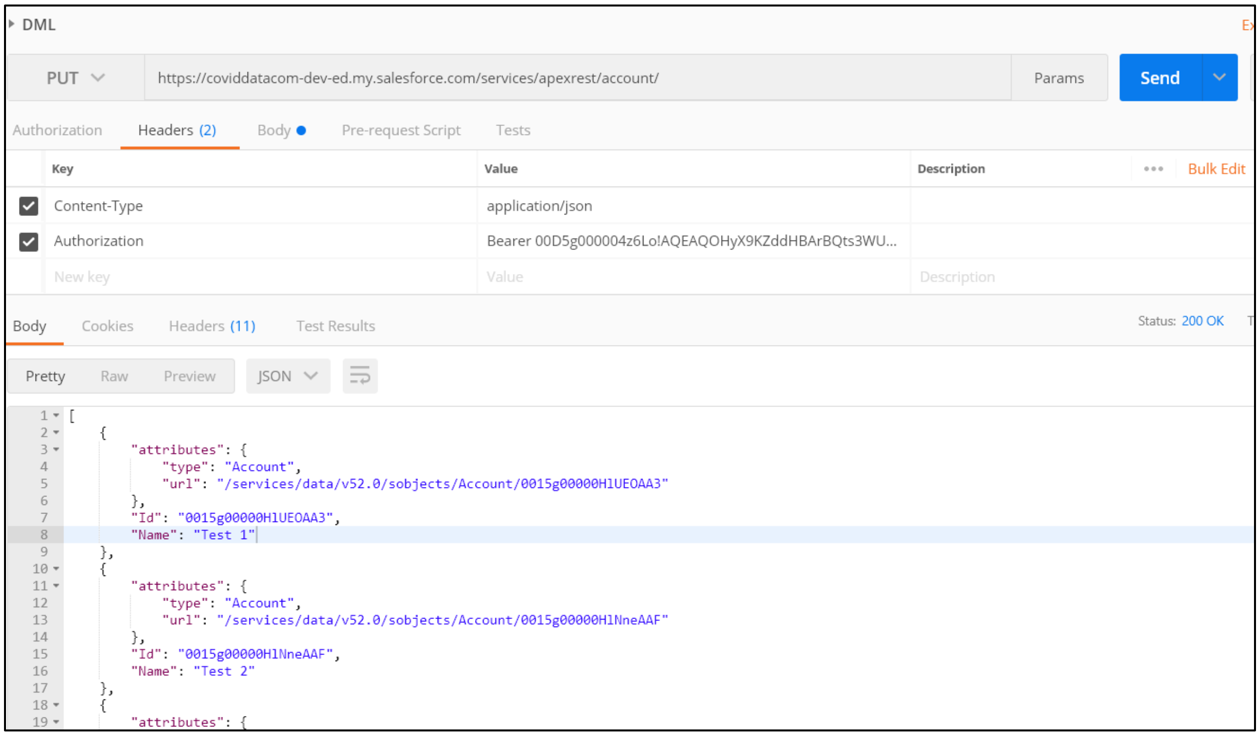
Key	Value	Description
✓ grant_type	password	
✓ client_id	3MVG9fe4g[redacted]WsMzzT…	
✓ client_secret	0B79296852C[redacted]303…	
✓ username	[redacted]	
✓ password	[redacted]	

Status: 200 OK

```
{
    "access_token": "00D5g000004z6Lo!AQEAQOHyX9KZddHBArBQts3WUA8erL7L_4pTBBt6oGgy0YDmyKU5J1gP85SQ7hGumgMZgbkIbztuCOjEvxGfmSxmdT6PIJZ8",
    "instance_url": "https://coviddatacom-dev-ed.my.salesforce.com",
    "id": "https://login.salesforce.com/id/00D5g000004z6LoEAI/0055g000008Kkb1AAC",
    "token_type": "Bearer",
    "issued_at": "1625226755259",
    "signature": "1qB0zjYKQa91k92eHVa9JuyDcEJ37jV7Xib1dg69Kfg="
}
```

Rest Testing

In authorization : Bearer <access-token>// bearer is a prefix



DML

PUT https://coviddatacom-dev-ed.my.salesforce.com/services/apexrest/account/

Headers (2)

Key	Value	Description
✓ Content-Type	application/json	
✓ Authorization	Bearer 00D5g000004z6Lo!AQEAQOHyX9KZddHBArBQts3WU…	

Status: 200 OK

```
[
    {
        "attributes": {
            "type": "Account",
            "url": "/services/data/v52.0/sobjects/Account/0015g00000H1UEOAA3"
        },
        "Id": "0015g00000H1UEOAA3",
        "Name": "Test 1"
    },
    {
        "attributes": {
            "type": "Account",
            "url": "/services/data/v52.0/sobjects/Account/0015g00000H1NneAAF"
        },
        "Id": "0015g00000H1NneAAF",
        "Name": "Test 2"
    },
    {
        "attributes": {
```

Steps for Org 2 acting as Client

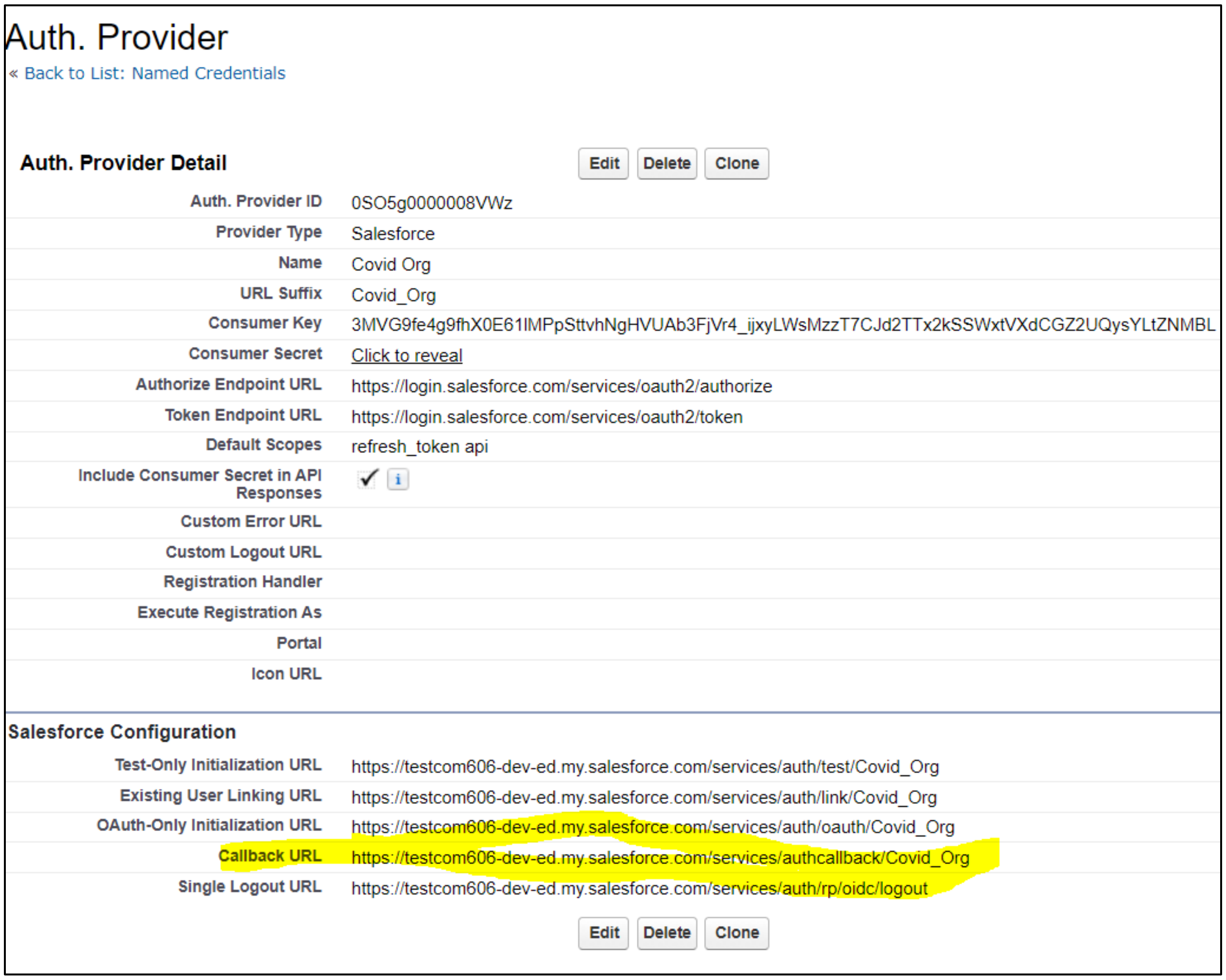
1. Create Authorization Provider – req. in Named Credentials
    a. Setup -> Auth. Provider
    b. Default scope – **refresh_token full**

refresh_token means – It will get updated token each time the existing token expires for authentication.

full means – full access,

Api – Api access.

Call back url – Need to be updated in call back field of connected app in org1.



Auth. Provider

« Back to List: Named Credentials

Auth. Provider Detail

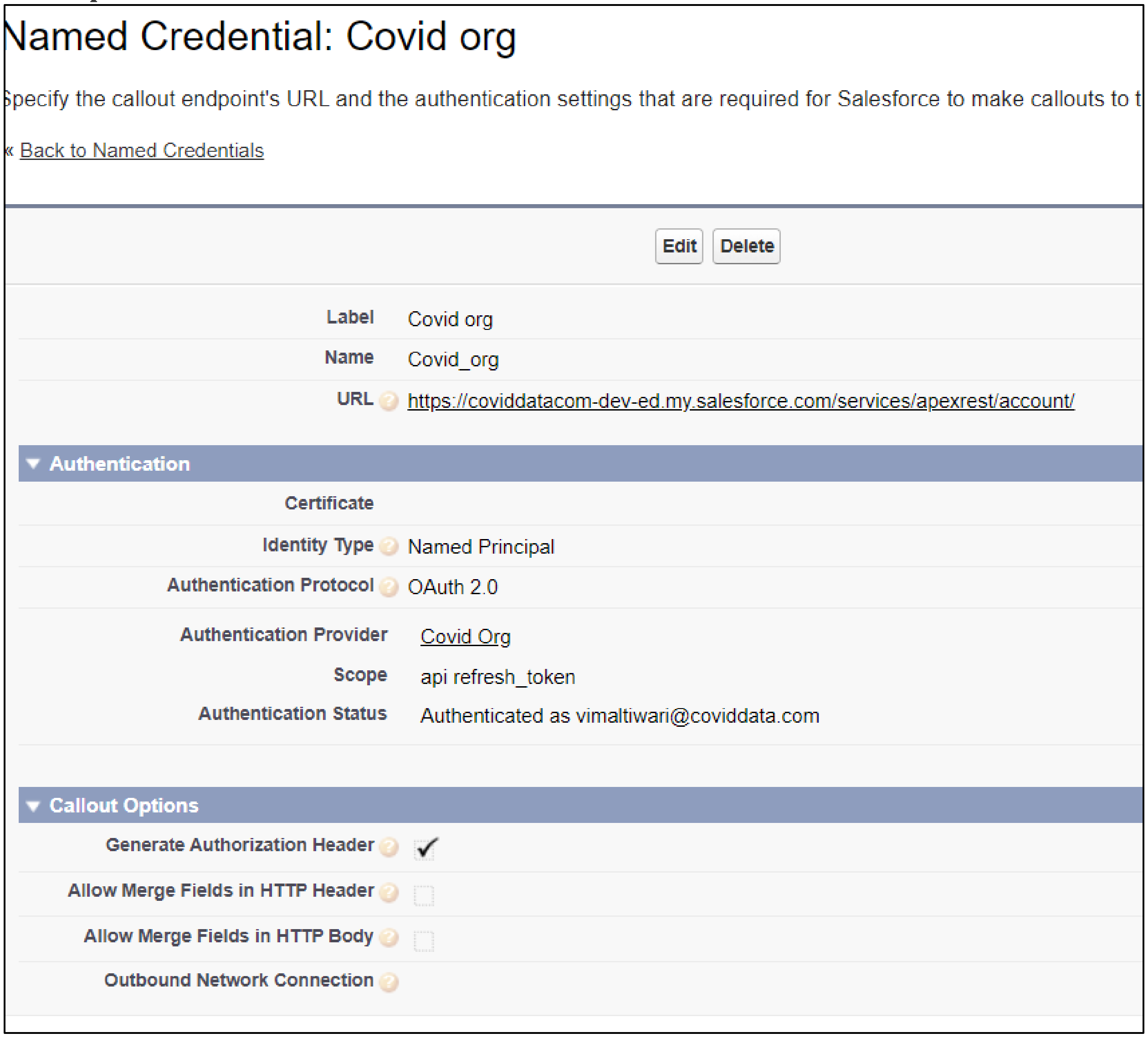
Auth. Provider ID	0SO5g0000008VWz
Provider Type	Salesforce
Name	Covid Org
URL Suffix	Covid_Org
Consumer Key	3MVG9fe4g9fhX0E61lMPpSttvhNgHVUAb3FjVr4_ijxyLWsMzzT7CJd2TTx2kSSWxtVXdCGZ2UQysYLtZNMBL
Consumer Secret	Click to reveal
Authorize Endpoint URL	https://login.salesforce.com/services/oauth2/authorize
Token Endpoint URL	https://login.salesforce.com/services/oauth2/token
Default Scopes	refresh_token api
Include Consumer Secret in API Responses	✓
Custom Error URL	
Custom Logout URL	
Registration Handler	
Execute Registration As	
Portal	
Icon URL	

Salesforce Configuration

Test-Only Initialization URL	https://testcom606-dev-ed.my.salesforce.com/services/auth/test/Covid_Org
Existing User Linking URL	https://testcom606-dev-ed.my.salesforce.com/services/auth/link/Covid_Org
OAuth-Only Initialization URL	https://testcom606-dev-ed.my.salesforce.com/services/auth/oauth/Covid_Org
Callback URL	https://testcom606-dev-ed.my.salesforce.com/services/authcallback/Covid_Org
Single Logout URL	https://testcom606-dev-ed.my.salesforce.com/services/auth/rp/oidc/logout

2. Create **Named Credentials**

a. Provide Auth. Provider for Oauth 2.0
b. Identity type –
    i. Per user – Api callout based on calling user's context
    ii. Named Principal – One user will be used for all callouts, might be the owner.
c. URL : the base url.
d. Authentication protocol – **if it is selected, that means system will take care of authentication. No need to do callout for authentication explicitly**.
e. Authentication status should not be pending. If it is there, setup is not complete.
f. **Generate authorization header** need to be checked, **if any Authentication protocol is selected.**



Named Credential: Covid org

pecify the callout endpoint's URL and the authentication settings that are required for Salesforce to make callouts to t

« Back to Named Credentials

Label	Covid org
Name	Covid_org
URL	https://coviddatacom-dev-ed.my.salesforce.com/services/apexrest/account/

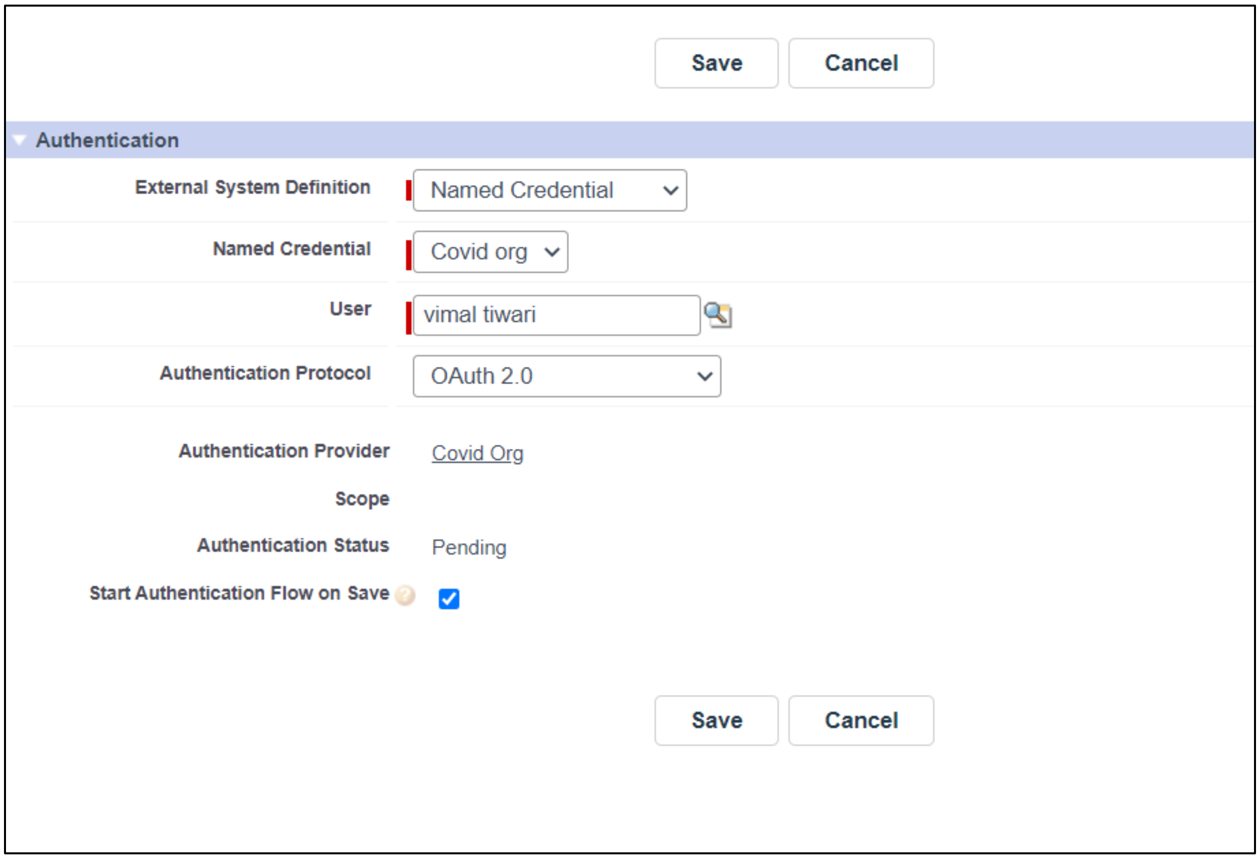
Authentication

Certificate	
Identity Type	Named Principal
Authentication Protocol	OAuth 2.0
Authentication Provider	Covid Org
Scope	api refresh_token
Authentication Status	Authenticated as vimaltiwari@coviddata.com

Callout Options

Generate Authorization Header	✓
Allow Merge Fields in HTTP Header	☐
Allow Merge Fields in HTTP Body	☐
Outbound Network Connection	

3. Give Access of Named Credential to Users – Need in case of **Per user** setting. I have used Permission set.
4. Authentication Settings for External Systems – **Needed in Per user setting**.
    a. Go to my settings -> personal -> Authentication Settings for External Systems
    b. Click new , add NC as external system definition.



External System Definition	Named Credential
Named Credential	Covid org
User	vimal tiwari
Authentication Protocol	OAuth 2.0
Authentication Provider	Covid Org
Scope	
Authentication Status	Pending
Start Authentication Flow on Save	✓

5. It's time to write some apex code and do some fun.
   Create new class **AccountRestClientWithOAuth2** and use its methods to get and put data.

For custom Authorization and callouts:

1. Create remote site setting for base URL.
2. AccountRestClient
    a. Call setAccessToken to get access token and set it in **platform cache and set time to live.** It is a field in seconds which can save the token expire time, based on server's details. Ex. 10 min, 5 min etc.
    b. For each callout, check if token is there, then go for other methods

c. Else refresh -> call the setAccessToken again and get new token, save it again in cache.

Reference :

https://www.jitendrazaa.com/blog/salesforce/login-to-salesforce-from-salesforce-using-authentication-provider/#more-4516

https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_callouts_named_credentials.htm

https://salesforce.stackexchange.com/questions/102484/use-of-named-credentials-seem-to-be-tied-to-external-data-source

Integration Framework

Overview: Since from UI, we can do real time callouts, because we can prevent user actions, for process based callouts, can't do it real time. It can be near real time. The framework takes callout request from any context and saves it. Then there will be some process which takes the request Queue and process it.

Classes Skeleton

```
global class ResponseWrapper{
        global String jsonResponse;
        global String returnCode;
}

global class RequestWrapper{
        global String endpoint;
        global String body;
}

global abstract class AbstractHTTPServiceClass{
        global abstract ResponseWrapper process(String httpType, RequestWrapper rw);
}


public class SampleHTTPServiceClass{
        public ResponseWrapper run(String httpType, RequestWrapper rw){
                if(httpType == 'GET'){
                        return getRecord(rw);
                }else if .....
        }
        @httpGet
        public ResponseWrapper getRecord(RequestWrapper rw){
                Http h =new Http();
                HttpRequest req = new HttpRequest();
                req.setEndpoint(rw.endpoint);
                HttpResponse res = h.send(req);
                return new ResponseWrapper(res.getBody());
        }
}
```

Data model

1. **Integration Log** – to save the details of an integration callout that is to be called.
    a. Service config – Unique name of corresponding service meta data record.
        i. Can create String like "Service.ClassName.Method" etc
    b. Log Status – New, In Progress, Success, Failed.
    c. Retry count.
    d. Status code
    e. Child records - Notes and Attachment
        i. Save Request as file – Request in Json converted in file
        ii. Save Response as file - Response in Json converted in file

2. **Service Configuration** – Custom Meta data to save callout information.
    a. Callout Service Class Name
    b. Callout /Http method type – Get/Post/PUT etc
    c. END-POINT – Named credential name.
    d. Relative URL – service URL that gets added after End point.
    e. Max Retry count
    f. Request Wrapper class
    g. Response Wrapper class

3. **Exception log**– log exception per callout
    a. Error
    b. Error type
    c. Callout Id – Integration log id

Execution

1. **Add in Queue - From the Context, where Callout needs to be made**,
    a. Create Integration log by putting required Values.
        i. Provide the meta data records' unique name to it.
        ii. From the meta data,
            1. get the request Wrapper class Name,
            2. create JSON from wrapper.
            3. attach it on integration log as file.
        iii. Mark status as In-Progress. As we cannot update it before Callout In same context.

2. **Process Queue** – Some process which takes the set of requests and process them one by one. For example, Batch, **Queuable** etc
    a. Get the set of Logs which are not successful yet.
    b. For each Integration log – that is in In-Progress.

```
        Try{

            i. Take the meta data record from Pre-queried Meta data list.
           ii. If retry count on log < Max retry in Meta data
               1. Get the service class name.
               2. Get the instance of Service class using Reflection.
                  (Type.forName)
               3. Call the Service Class process method
                      a. pass the method type from Meta data record.
                      b. Pass the Request Json from Attachment file.
               4. The method will do Callout, and returns a response.
               5. If the return code is 200,
                      a. update records status to Success.
                      b. De-serialize the response in Response Wrapper class
                         from meta data, Convert into JSON and save it on log's
                         attachment.
               6. Else
                      a. increase the retry count.
                      b. Change status of log to FAIL.
               7. Save the return code on Log record.
    }catch(Exception e){
            Use platform event to Create exception log record. And publish it.
    }
```

    c. Once the loop gets over. We can again go for retrying the failed logs. For this, don't do step 6b . Let the log be in IN-Progress, and again trigger the same process.
    d. For some response codes, we don't want to do anything such as 404, 403. In those cases, remove them from iterating list based on status codes.