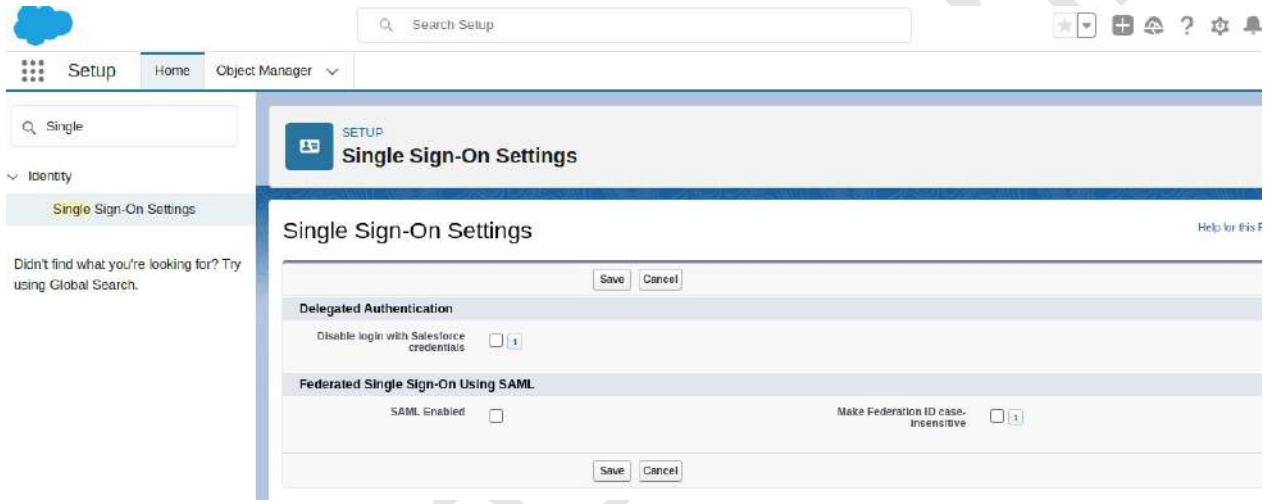
Integration

Q. What is Single Sign On ?

Ans -- > Single sign-on (SSO) allows users to access multiple applications with a single set of credentials. They do not need to remember separate user ID/password for each application. Salesforce offers various options for configuring single sign-on. This also includes:

- Federated Authentication using SAML
- Delegated Authentication
- OpenID Connect



We need to enable SAML in single sign on setting

Q. What is Identity Provider ?

Ans--> IdP stands for Identity Provider and SP stands for Service Provider.IdP is the system that authenticates user by validating the username and password and then subsequently all other applications trust IdP and allow user to access the application if the IdP asserts that the user is a valid user.IdP is the system that stores user's login name and password.

Q. What is Service Provider ?

Ans --> A service provider is a website that hosts apps. A Service Provider (SP) is the entity providing the service, typically in the form of an application.Let suppose we are accessing the google's credential to login salesforce org so Salesforce will be the Service Provider and Google will be the Identity provider because Google will validate the user and salesforce will give the service for doing work.

Q. What is Salesforce Integration ?

Ans--> Salesforce Integration is a process of connecting two or more applications.

Q. What are the Salesforce Integration Direction ?

Ans--> Integration can be two direction inbound integration or outbound Integration.

Inbound Integration: An external system initiates contact with Salesforce.

Outbound Integration: Salesforce initiates contact with an external system.

Q. What is outbound and inbound Integration in Salesforce ?

Ans - >

Inbound Web Service:

Inbound web service is when Salesforce exposes SOAP/REST web service, and any external/third party application consume it to get data from your Salesforce org. It is an Inbound call to Salesforce, but outbound call to the external system. Here, Salesforce is the publisher and external system is the consumer of web services.

Outbound Web Service:

Outbound web service is when Salesforce consume any external/third party application web service, a call needs to send to the external system. It is an Inbound call to the external system, but outbound call to Salesforce. Here, external system is the publisher of web services and Salesforce is the consumer.

Q. Different APIs in Salesforce and when we can use these Api?

Ans -

REST API

REST API is a simple and powerful web service based on RESTful principles. It exposes all sorts of Salesforce functionality via REST resources and HTTP methods. For example, you can create, read, update, and delete (CRUD) records, search or query your data, retrieve object metadata, and access information about limits in your org. REST API supports both XML and JSON.

Because REST API has a lightweight request and response framework and is easy to use, it’s great for writing mobile and web apps.

SOAP API

SOAP API is a robust and powerful web service based on the industry-standard protocol of the same name. It uses a Web Services Description Language (WSDL) file to rigorously define the parameters for accessing data through the API. SOAP API supports XML only. Most of the SOAP API functionality is also available through REST API. It just depends on which standard better meets your needs.

Because SOAP API uses the WSDL file as a formal contract between the API and consumer, it’s great for writing server-to-server integrations.

Bulk API

Bulk API is a specialized RESTful API for loading and querying lots of data at once. By lots, we mean 50,000 records or more. Bulk API is asynchronous, meaning that you can submit a request and come back later for the results. This approach is the preferred one when dealing with large amounts of data. There are two versions of Bulk API (1.0 and 2.0). Both versions handle large amounts of data, but we use Bulk API 2.0 in this module because it's a bit easier to use.

Bulk API is great for performing tasks that involve lots of records, such as loading data into your org for the first time.

Pub/Sub API

Use Pub/Sub API for integrating external systems with real-time events. You can subscribe to real-time events that trigger when changes are made to your data or subscribe to custom events. The APIs use a publish-subscribe, or pub/sub, model in which users can subscribe to channels that broadcast data changes or custom notifications.

The pub/sub model reduces the number of API requests by eliminating the need for making frequent API requests to get data. Pub/Sub API is great for writing apps that would otherwise need to frequently poll for changes.

When to Use Pub/Sub API
You can use Pub/Sub API to integrate external systems with real-time events. Streams of data are based on custom payloads through platform events or changes in Salesforce records through Change Data Capture. Within Salesforce, you can publish and subscribe to events with Apex triggers, Process Builder, and Flow Builder.

Pub/Sub API is built for high scale, bi-directional event integration with Salesforce. Use Pub/Sub API to efficiently publish and subscribe to binary event messages in the Apache Avro format. Pub/Sub API is based on gRPC and HTTP/2 and uses a pull-based model so you can control the subscription flow. With Pub/Sub API, you can use one of the 11 programming languages that gRPC supports.

When to Use Apex REST API
Use Apex REST API when you want to expose your Apex classes and methods so that external applications can access your code through REST architecture. Apex REST API supports both OAuth 2.0 and Session ID for authentication.

When to Use Apex SOAP API
Use Apex SOAP API when you want to expose Apex methods as SOAP web service APIs so that external applications can access your code through SOAP. Apex SOAP API supports both OAuth 2.0 and Session ID for authentication.

When to Use Tooling API
Use Tooling API to build custom development tools or apps for Platform applications. For example, you can use Tooling API to add features and functionality to your existing Platform tools and build dynamic modules into your enterprise integration tools. You can also use Tooling API to build specialized development tools for a specific application or service.

Tooling API's SOQL capabilities for many metadata types allow you to retrieve smaller pieces of metadata. Smaller retrieves improve performance, making Tooling API a good fit for developing interactive applications. Tooling API provides SOAP and REST interfaces.

When to Use GraphQL API
Build highly responsive and scalable apps by returning only the data a client needs, all in a single request. GraphQL API overcomes the challenges posed by traditional REST APIs through field selection, resource aggregation, and schema introspection. Field selection reduces the size of the payload, sending back only fields that were included in the query. Aggregations reduce round trips between the client and server, returning a set of related resources within a single response. Schema introspection enables a user to see the types, fields, and objects that the user has access to.

Q. What is connected App in Salesforce and how can we create connected app in Salesforce ?
Ans - 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.

Q. Some Scenarios for using connected app in Salesforce ?

Ans- **Integration with a Custom Marketing Automation Tool**
Implementation Steps:

Step 1: Create a Connected App in Salesforce:

1. Log in to Salesforce as an administrator.
2. Click on the gear icon in the top-right corner to access "Setup."
3. In the Quick Find box, type "App Manager" and select "App Manager."
4. Click the "New Connected App" button.
5. Configure the basic information:

- **Connected App Name**: Give your app a name (e.g., "MarketingAutomationIntegration").
- **API Name**: It will be automatically generated based on the app name.
- **Contact Email**: Provide a contact email address.
- **Enable OAuth Settings**: Check this box to enable OAuth authentication.

Step 2: Configure OAuth Settings:

6. In the "API (Enable OAuth Settings)" section, configure the OAuth settings:

- **Callback URL**: Provide the callback URL where your marketing automation tool will redirect users after authentication. It should be a URL within your marketing automation tool's settings.
- **Selected OAuth Scopes**: Choose the necessary OAuth scopes based on the permissions your integration needs. For example, you might need "Access and manage your data (api)" and "Perform requests on your behalf at any time (refresh_token, offline_access)."

- **Require Secret for Web Server Flow**: If your integration will use the Web Server OAuth flow, check this box.

7. Optionally, you can configure other settings, such as digital signatures or IP Relaxation settings, depending on your security requirements.

Step 3: Save the Connected App:

8. Click the "Save" button to save your connected app configuration.

Step 4: Note the Consumer Key and Secret:

9. After saving, Salesforce will generate a "Consumer Key" and "Consumer Secret." Keep these values secure as they'll be used for authentication.

Step 5: Configure Permitted Users:

10. In the "Profiles" related list, specify which user profiles or permission sets are allowed to use the connected app. Ensure that the appropriate users have access to the connected app.

Step 6: Implement Integration with the Marketing Automation Tool:

11. In your custom marketing automation tool, implement OAuth 2.0 authentication using the Salesforce connected app's Consumer Key and Secret. Follow Salesforce's OAuth 2.0 authentication flow to obtain access tokens.
12. Once you obtain an access token, use it to make authorized API requests to Salesforce. You can create and update leads, contacts, and other Salesforce objects as needed from your marketing automation tool.

Step 7: Monitor and Maintain:

13. Regularly monitor the integration's usage, and ensure that you handle token expiration and token refresh logic in your integration to maintain seamless connectivity.

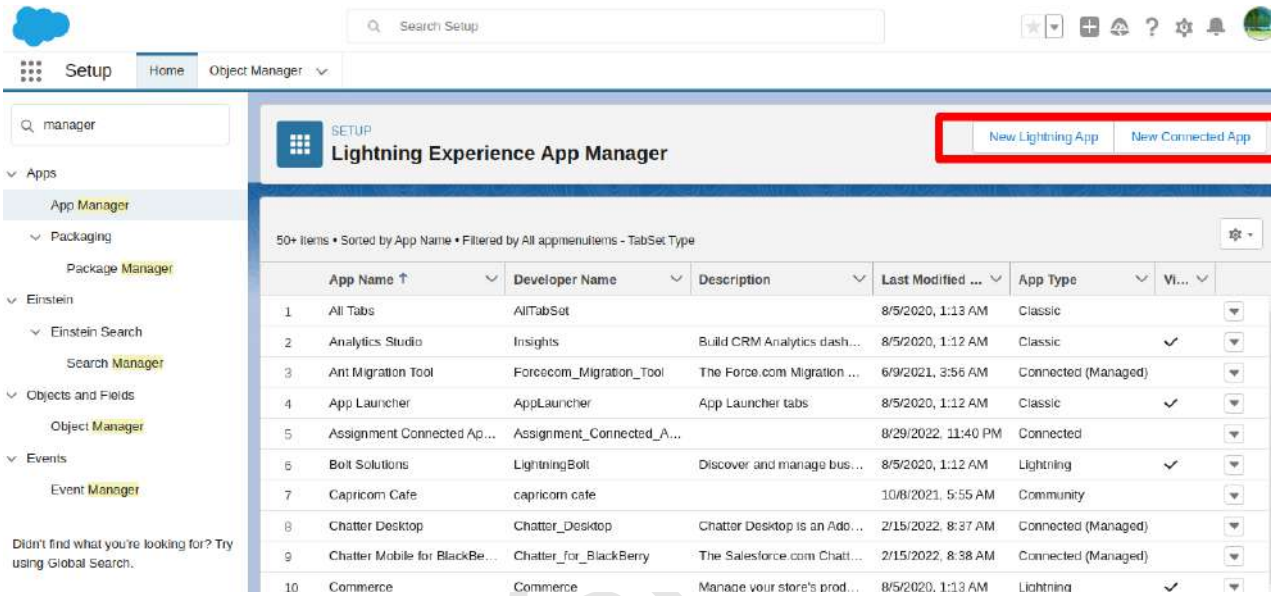
Scenario 2- External Identity Providers:
Your organization uses an external identity provider (IdP) for user authentication, and you want to connect Salesforce with this IdP.

- Solution: Create a connected app in Salesforce and configure it to use the IdP's authentication services, such as SAML or OpenID Connect, for user authentication and single sign-on.

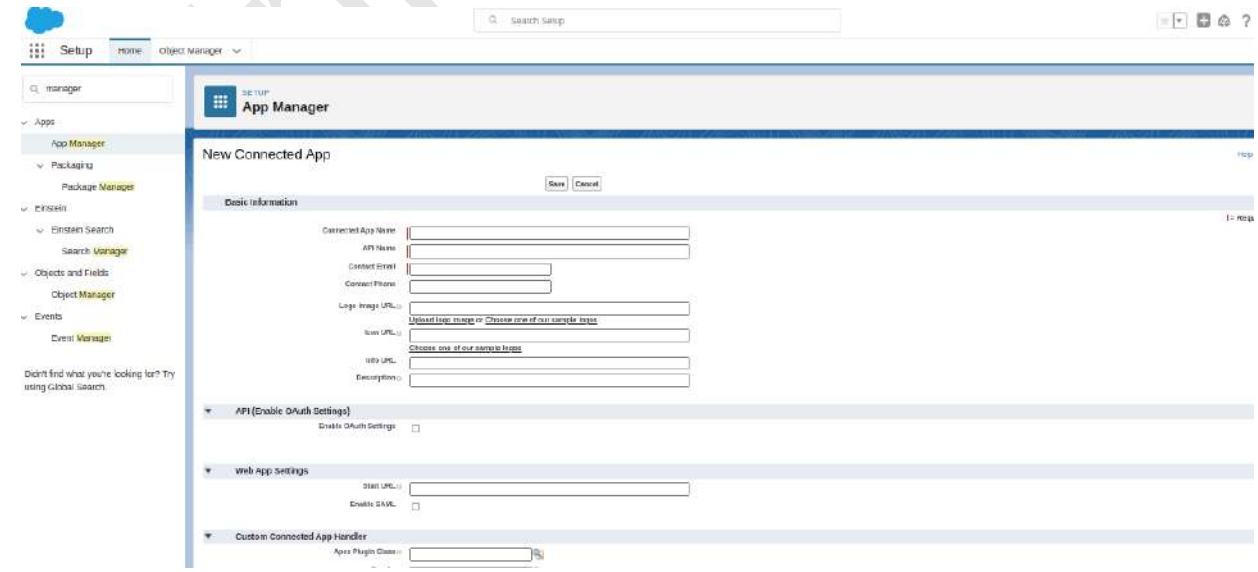
Scenario 3 - Single Sign-On (SSO):

Scenario: Your company uses Salesforce for customer relationship management (CRM) and Google Workspace (formerly G Suite) for email and collaboration. You want your Salesforce users to access Google Workspace seamlessly without having to log in separately.

Solution: Create a connected app in Salesforce and configure it to use OAuth 2.0 for SSO with Google Workspace. This way, users can log in to Salesforce and access Google Workspace resources without additional login steps.



If you want to create a connected app in Salesforce first you search on Quick find box '**App Manager'** and select this after that you can click on **New Connected App**



You need to fill the details as per your requirement

You can create connected app and try to implement in your org and refer this module for practice

https://trailhead.salesforce.com/content/learn/modules/mobile_sdk_introduction/mobilesdk_intro_security

Q. Oauth Terminologies

OAuth (Open Authorization): OAuth is an open standard protocol that enables secure authorization and authentication for granting access to resources, such as APIs, without exposing user credentials.
Client Application: The software application that wants to access protected resources on behalf of the user. In Salesforce integration, this could be a third-party app or system.

Resource Owner: The user or entity that owns the protected resource. In Salesforce, this is typically a Salesforce user whose data or resources are being accessed.

Authorization Server: In Salesforce, the authorization server is the Salesforce Identity and Access Management system. It handles user authentication and issues access tokens after the user grants permission.

Access Token: An access token is a credential used by the client application to access protected resources. In Salesforce, access tokens are short-lived and grant access to specific resources for a limited time.

Refresh Token: A refresh token is a long-lived credential that can be used to obtain a new access token when the current one expires. It is often used to maintain a long-term connection between the client application and Salesforce.

Authorization Code: In the Web Server OAuth Authentication Flow, after the user is authenticated, the authorization server issues an authorization code. This code is exchanged for an access token and a refresh token by the client application.

Consumer Key: It is value used by the consumer—in this case, the Mobile SDK app—to identify itself to Salesforce. Referred to as client_id.

Scopes: Scopes define the specific permissions and access rights requested by the client application. In Salesforce, scopes can control the level of access to objects and data.

Redirect URI (Callback URL): When the user is authenticated and grants permissions, the authorization server redirects the user's browser to a specific URL (the redirect URI) with the authorization code. In Salesforce integrations, this URL is often provided by the client application.

JWT (JSON Web Token): JWT is a compact, URL-safe means of representing claims to be transferred between two parties. In Salesforce, JWTs are used in the JWT Bearer Token OAuth Authentication Flow for secure communication.

Connected App: In Salesforce, a connected app represents the client application that wants to integrate with Salesforce using OAuth. Connected apps define various settings, including OAuth settings, to control the integration.

User-Agent Flow: Also known as the Implicit Flow, this OAuth flow is used for single-page applications and mobile apps where the access token is returned directly to the user's browser or app.

Username-Password Flow: This OAuth flow allows the client application to directly exchange the user's Salesforce credentials for an access token. It's generally discouraged due to security concerns.

Q. Oauth 1.O vs Oauth 2.O ?

Better support for non-browser applications
OAuth 1.0 has been designed focusing on the interactions of inbound and outbound messages in web client applications. Therefore, it is inefficient for non-browser clients. OAuth 2.0 has addressed this issue by introducing more authorization flows for different client needs that do not use web UIs.

Reduced complexity in signing requests
OAuth 1.0 needs to generate a signature on every API call to the server resource and that should be matched with the signature generated at the receiving endpoint in order to have access for the client. OAuth 2.0 do not need to generate signatures. It uses TLS/SSL (HTTPS) for communication.

The separation of roles
Handling resource requests and handling user authorization can be decoupled in OAuth 2.0. It has clearly defined the roles involved in communication which are client, resource owner, resource server, and authorization server.

The short-lived access token and the refresh token
In OAuth 1.0, access tokens can be stored for a year or more. But in OAuth 2.0, access tokens can contain an expiration time, which improves the security and reduces the chances of illegal access. And it offers a refresh token which can be used to get a new access token at the access token expiration without reauthorizing.

Q. What is Rest API ?

Ans -> The Salesforce REST API lets you integrate with Salesforce applications using simple HTTP methods, in either JSON or XML formats, making this an ideal API for developing mobile applications or external clients.

HTTP Method		Description
GET	>>>>	Retrieve data identified by a URL.
POST	>>>>	Create a resource or post data to the server.
DELETE	>>>>	Delete a resource identified by a URL.
PUT	>>>>	Create or replace the resource sent in the request body.
PATCH	>>>>	Partial update to an existing resource,

Rest Api Callout ---> Code

Piece of code --> Retrieve the data from third party (Get the Data)

```
Http http = new Http();
HttpRequest request = new HttpRequest();
request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals');
request.setMethod('GET');
HttpResponse response = http.send(request);
// If the request is successful, parse the JSON response.
if(response.getStatusCode() == 200) {
    // Deserialize the JSON string into collections of primitive data types.
    Map<String, Object> results = (Map<String, Object>)
JSON.deserializeUntyped(response.getBody());
    // Cast the values in the 'animals' key as a list
    List<Object> animals = (List<Object>) results.get('animals');
    System.debug('Received the following animals:');
    for(Object animal: animals) {
        System.debug(animal);
    }
}
```

Piece of code --> Send Data to a Service (Post)

```
Http http = new Http();
HttpRequest request = new HttpRequest();
request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals');
request.setMethod('POST');
request.setHeader('Content-Type', 'application/json;charset=UTF-8');
// Set the body as a JSON object
request.setBody('{"name":"mighty moose"}');
HttpResponse response = http.send(request);
// Parse the JSON response
if(response.getStatusCode() != 201) {
    System.debug('The status code returned was not expected: ' + response.getStatusCode() + ' ' +
response.getStatus());
} else {
    System.debug(response.getBody());
}
```

now you can test callout with the help of HttpCalloutMock

Test a Callout with HttpCalloutMock

To test your POST callout, we provide an implementation of the HttpCalloutMock interface. This interface enables you to specify the response that's sent in the respond method. Your test class instructs the Apex runtime to send this fake response by calling Test.setMock again. For the first argument, pass HttpCalloutMock.class. For the second argument, pass a new instance of

AnimalsHttpCalloutMock, which is your interface implementation of HttpCalloutMock. (We'll write AnimalsHttpCalloutMock in the example after this one.)

Test.setMock(HttpCalloutMock.class, new AnimalsHttpCalloutMock());

Now add the class that implements the HttpCalloutMock interface to intercept the callout. If an HTTP callout is invoked in test context, the callout is not made. Instead, you receive the mock response that you specify in the respond method implementation in AnimalsHttpCalloutMock.

Apex Class

AnimalsCallouts

```
public class AnimalsCallouts {
    public static HttpResponse makeGetCallout() {
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals');
        request.setMethod('GET');
        HttpResponse response = http.send(request);
        // If the request is successful, parse the JSON response.
        if(response.getStatusCode() == 200) {
            // Deserializes the JSON string into collections of primitive data types.
            Map<String, Object> results = (Map<String, Object>)
JSON.deserializeUntyped(response.getBody());
            // Cast the values in the 'animals' key as a list
            List<Object> animals = (List<Object>) results.get('animals');
            System.debug('Received the following animals:');
            for(Object animal: animals) {
                System.debug(animal);
            }
        }
        return response;
    }
    public static HttpResponse makePostCallout() {
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals');
        request.setMethod('POST');
        request.setHeader('Content-Type', 'application/json;charset=UTF-8');
        request.setBody('{"name":"mighty moose"}');
        HttpResponse response = http.send(request);
        // Parse the JSON response
        if(response.getStatusCode() != 201) {
            System.debug('The status code returned was not expected: ' +
                response.getStatusCode() + ' ' + response.getStatus());
        } else {
            System.debug(response.getBody());
        }
        return response;
    }
```

```
}
```

AnimalsCalloutsTest

```
@isTest
private class AnimalsCalloutsTest {
    @isTest static  void testGetCallout() {
        // Create the mock response based on a static resource
        StaticResourceCalloutMock mock = new StaticResourceCalloutMock();
        mock.setStaticResource('GetAnimalResource');
        mock.setStatusCode(200);
        mock.setHeader('Content-Type', 'application/json;charset=UTF-8');
        // Associate the callout with a mock response
        Test.setMock(HttpCalloutMock.class, mock);
        // Call method to test
        HttpResponse result = AnimalsCallouts.makeGetCallout();
        // Verify mock response is not null
        System.assertNotEquals(null,result, 'The callout returned a null response.');
        // Verify status code
        System.assertEquals(200,result.getStatusCode(), 'The status code is not 200.');
        // Verify content type
        System.assertEquals('application/json;charset=UTF-8',
          result.getHeader('Content-Type'),
          'The content type value is not expected.');
        // Verify the array contains 3 items
        Map<String, Object> results = (Map<String, Object>)
            JSON.deserializeUntyped(result.getBody());
        List<Object> animals = (List<Object>) results.get('animals');
        System.assertEquals(3, animals.size(), 'The array should only contain 3 items.');
    }
}
```

HttpCalloutMock

```
@isTest
global class AnimalsHttpCalloutMock implements HttpCalloutMock {
    // Implement this interface method
    global HTTPResponse respond(HTTPRequest request) {
        // Create a fake response
        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json');
        response.setBody('{"animals": ["majestic badger", "fluffy bunny", "scary bear", "chicken",
"mighty moose"]}');
        response.setStatusCode(200);
        return response;
    }
}
```

Create Custom Rest API In Salesforce

Sometimes we need to do some customization in OOB REST API for some complex implementation.

	Use	Action
@RestResource(urlMapping=“ url”)	Defines the class as a custom Apex endpoint	
@HttpGet	Defines the function to be called via Http Get- Used to retrieve a record	Read
@HttpDelete	Used to delete a record	Delete
@HttpPost	Used to create a record	Create
@HttpPatch	Used to partially update a record	Upsert
@HttpPut	Used to fully update a record	Update

MyFirstRestAPIClass

RestContext --> To access the RestRequest and RestResponse objects in an Apex REST method.

```
@RestResource(urlMapping='/api/Account/*')
global with sharing class MyFirstRestAPIClass
{
   @HttpGet
   global static Account doGet() {
      RestRequest req = RestContext.request;
      RestResponse res = RestContext.response;
      String AccNumber = req.requestURI.substring(req.requestURI.lastIndexOf('/')+1);
      Account result = [SELECT Id, Name, Phone, Website FROM Account WHERE
AccountNumber = :AccNumber ];
      return result;
   }

   @HttpDelete
   global static void doDelete() {
      RestRequest req = RestContext.request;
      RestResponse res = RestContext.response;
      String AccNumber = req.requestURI.substring(req.requestURI.lastIndexOf('/')+1);
      Account result = [SELECT Id, Name, Phone, Website FROM Account WHERE
AccountNumber = :AccNumber ];
      delete result;
   }

   @HttpPost
   global static String doPost(String name,String phone,String AccountNumber ) {
      Account acc = new Account();
      acc.name= name;
      acc.phone=phone;
      acc.AccountNumber =AccountNumber ;
      insert acc;
```

```
        return acc.id;
    }

}
```

Test Class for REST API

MyFirstRestAPIClassTest

```
@IsTest
private class MyFirstRestAPIClassTest {

    static testMethod void testGetMethod(){
        Account acc = new Account();
        acc.Name='Test';
        acc.AccountNumber ='12345';
        insert acc;

        RestRequest request = new RestRequest();
        request.requestUri ='/services/apexrest/api/Account/12345';
        request.httpMethod = 'GET';
        RestContext.request = request;
        Account acct = MyFirstRestAPIClass.doGet();
        System.assert(acct != null);
        System.assertEquals('Test', acct.Name);

    }

    static testMethod void testPostMethod(){
        RestRequest request = new RestRequest();
        request.requestUri ='/services/apexrest/api/Account/12345';
        request.httpMethod = 'POST';
        RestContext.request = request;
        String strId = MyFirstRestAPIClass.doPost('Amit','2345678','12345');
        System.assert(strId !=null );
    }

    static testMethod void testDeleteMethod(){
        Account acc = new Account();
        acc.Name='Test';
        acc.AccountNumber ='12345';
        insert acc;

        RestRequest request = new RestRequest();
        request.requestUri ='/services/apexrest/api/Account/12345';
        request.httpMethod = 'DELETE';
        RestContext.request = request;
        MyFirstRestAPIClass.doDelete();

        List<Account> ListAcct = [SELECT Id FROM Account WHERE Id=:acc.id];
```

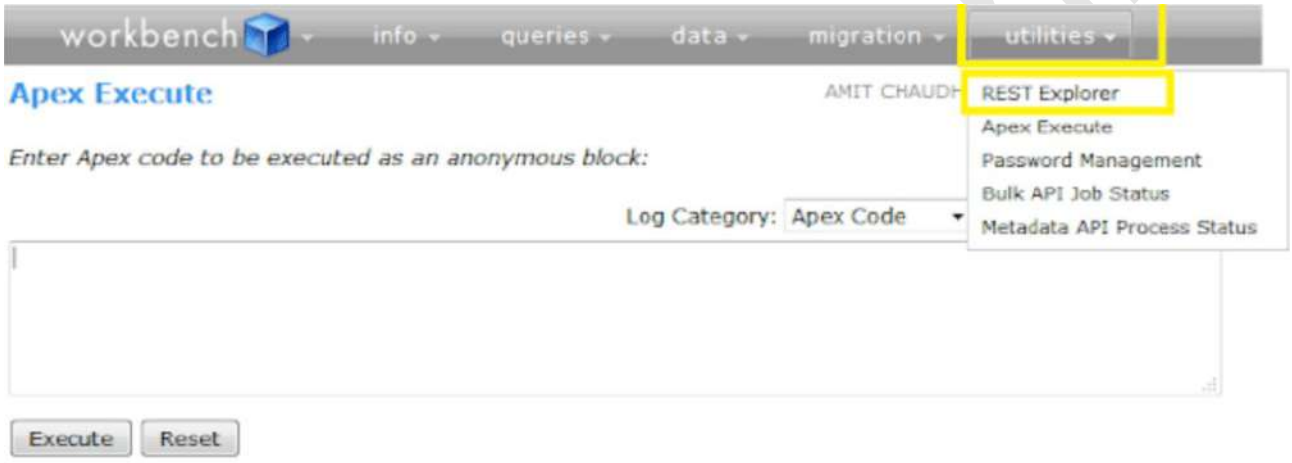
```
        System.assert(ListAcct.size() ==0 );
    }

}
```

Execute Your Apex REST Class In Workbench

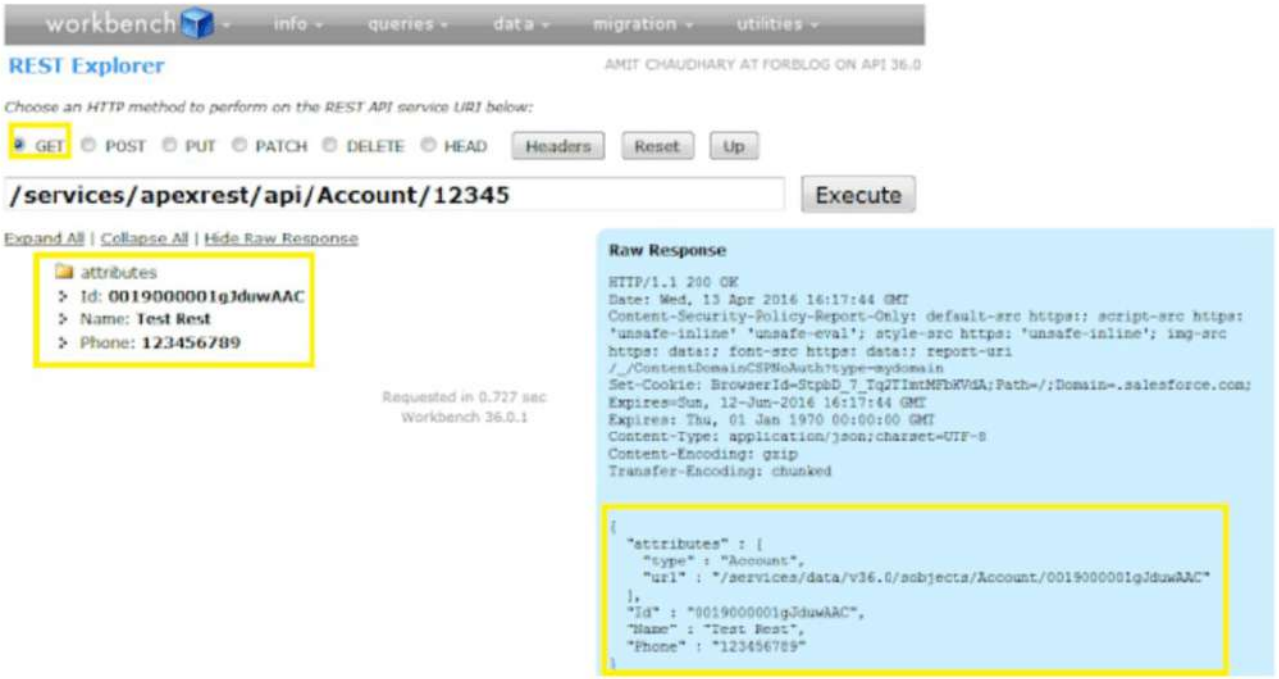
Step 1:- Open and log in.
Step 2:- Select Environment as Production and select the checkbox to agree on the terms and conditions then select log in with Salesforce

Step 3:- In the Workbench tool select Utilities > REST Explorer



Step 4:- In the REST Explorer window paste the following URL in the box

Method:- Get
URL:- /services/apexrest/api/Account/12345


workbench
info
queries
data
migration
utilities
REST Explorer
AMIT CHAUDHARY AT FORBLOG ON API 36.0
Choose an HTTP method to perform on the REST API service URI below:
GET
POST
PUT
PATCH
DELETE
HEAD
Headers
Reset
Up
/services/apexrest/api/Account/12345
Execute
Expand All | Collapse All | Hide Raw Response
attributes
Id: 0019000001gJduwAAC
Name: Test Rest
Phone: 123456789
Requested in 0.727 sec
Workbench 36.0.1
Raw Response
HTTP/1.1 200 OK
Date: Wed, 13 Apr 2016 16:17:44 GMT
Content-Security-Policy-Report-Only: default-src https:; script-src https: 'unsafe-inline' 'unsafe-eval'; style-src https: 'unsafe-inline'; img-src https: data:; font-src https: data:; report-uri /_/ContentDomainCSPNoAuth?type=mydomain
Set-Cookie: BrowserId=StpbD_7_TqJTImtMFbKVdA;Path=/;Domain=.salesforce.com; Expires=Sun, 12-Jun-2016 16:17:44 GMT
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Type: application/json;charset=UTF-8
Content-Encoding: gzip
Transfer-Encoding: chunked
{
"attributes" : [
"type" : "Account",
"url" : "/services/data/v36.0/sobjects/Account/0019000001gJduwAAC"
],
"Id" : "0019000001gJduwAAC",
"Name" : "Test Rest",
"Phone" : "123456789"
}