Chapter 1. INTRODUCTION

REST API

You can access your Salesforce data without using the Salesforce user interface via the REST API, which is one of the various online interfaces available. With API access, you can do whatever you want with Salesforce and integrate it into your applications.

The REST API makes use of the RESTful architecture to give a simple and uniform interface. One of the main advantages of REST API is that it does not necessitate a lot of tooling to access your data. It's easier to use than SOAP API, but it still has a lot of power. Other Salesforce APIs, such as Bulk 2.0 API, Metadata API, and Connect REST API, provide additional capability for specialized activities.

API Compatible Editions and Development Environments

You'll need a Salesforce org with API access and the API Enabled user permission within that org to utilize APIs to access your Salesforce data.

API Access

Professional Edition, Performance Edition, Enterprise Edition, Unlimited Edition, and Developer Edition all come with API access. All requests for API access in the Professional Edition must be paid and completed through your Account Executive. All development and testing should be done in Developer Edition, sandboxes, or scratch org to protect your live data. You can use this method to create a separate environment where you can test things out before implementing the modifications.

REST Resources and Requests[{1}]

The REST API is built on the use of resources, which are Salesforce data items such as records, groups of records, query results, metadata, and API information. A uniform resource identifier (URI) is assigned to each resource, and it is accessed by submitting HTTP requests to that URI.

You can conduct a variety of activities depending on the resource you wish to access and how you create an HTTP request, including:

- Determine available API versions
- Access limits for your Salesforce org

- Retrieve object metadata
- Create, read, update, and delete records
- Query and search for data

Because you can use a variety of software tools to send HTTP requests, the actual appearance of a request may differ from the cURL examples in this article. The ingredients, however, remain the same regardless of how requests are submitted. These elements are commonly used in requests.

- URI
- HTTP method
- Headers
- Request body (not required for GET requests)

Chapter 2. Comparison between REST API and SOAP API [2]

SOAP and REST APIs are not comparable in any way. However, there are a few items to consider below that will help you pick between these two web services. Here are several examples:

1. REST stands for REpresentational State Transfer and SOAP is for Simple Object Access Protocol.
2. Because SOAP is a protocol, it adheres to a tight set of guidelines to allow communication between the client and the server, whereas REST is an architectural style that does not adhere to any set of guidelines but does adhere to six limitations outlined by Roy Fielding in 2000. Uniform Interface, Client-Server, Stateless, Cacheable, Layered System, and Code on Demand are the limitations.
3. SOAP is limited to XML for transmitting information in its message format, whereas REST is not. It is up to the implementer to decide which Media-Type to use, such as XML, JSON, or plain-text. Furthermore, REST can make use of the SOAP protocol, but SOAP cannot make use of REST.
4. SOAP employs @WebService for service interfaces to business logic, whereas REST uses URIs like @Path instead of interfaces.
5. SOAP is complex to implement and consumes more bandwidth, but REST is simple to implement and consumes less bandwidth, making it ideal for mobile devices.
6. SOAP provides several advantages over REST, one of which is that SOAP transactions are ACID compliant. Some applications demand transactional capability, which SOAP provides but REST does not.
7. SOAP uses SSL (Secure Socket Layer) and WS-security for security, whereas REST uses SSL and HTTPS. SOAP is favored over REST for dealing with bank account passwords, credit card numbers, and other sensitive information. The issue with security is that it is entirely dependent on your application's requirements; you must implement security on your own. It all comes down to the protocol you choose.

Chapter 3. REST API Architectural Constraints [3]

The acronyms REST and API stand for REpresentational State Transfer and Application Program Interface, respectively. REST is a software architecture style that specifies the guidelines for developing web services. RESTful web services are online services that follow the REST architectural paradigm. It provides a uniform and predetermined set of rules for requesting systems to access and change web resources. The Hypertext Transfer Protocol (HTTP) is used to communicate in REST-based services (HTTP).
A Restful system is made up of a:

- A client who makes resource requests.
- server with the necessary resources

It is critical to establish REST APIs per industry standards, as this facilitates the development and increases client acceptance.

RESTful API Architectural Constraints:

The following are six architectural restrictions that each web service must adhere to:

- Uniform Interface
- Stateless
- Cacheable
- Client-Server
- Layered System
- Code on Demand

Code on demand is the only REST architecture requirement that is optional. A service cannot strictly be described as RESTful if it violates any other requirement.

Uniform Interface: This is a crucial distinction between a REST API and a non-REST API. It implies that, regardless of device or application type, there should be a standard manner of interacting with a server (website, mobile app).

The four principles of Uniform Interface are as follows:

- **Resource-Based:** Individual resources are identified in requests. For example API/users.
- **Manipulation of Resources Through Representations:** The client has a representation of the resource and enough information to edit or remove it on the server if it has the authorization to do so. Example: When a user requests a list of users, they typically receive a user id and then use that id to delete or alter that specific user.
- **Self-descriptive Messages:** Each message contains sufficient information to specify how the message should be processed, allowing the server to quickly analyze the request.
- **Hypermedia as the Engine of Application State (HATEOAS):** It should include links for each response so that the client can readily find additional resources.

Stateless: It means that the state required to handle the request is included inside the request itself, and the server does not keep any session data. In REST, the client must include all information required by the server to perform the request, whether in query params, headers, or the URI. Because the server does not have to maintain, update, or communicate the session state, statelessness allows for higher availability. When a client needs to transfer a large amount of data to the server, it restricts the scope of network optimization and necessitates additional bandwidth.

Cacheable: Every response should state whether it is cacheable or not, as well as how long responses can be stored on the client-side. For every subsequent request, the client will return the data from its cache, eliminating the need to transmit the request to the server again. Caching that is well-managed reduces or eliminates some client-server interactions, enhancing availability and performance even more. However, there is a potential that the user will obtain stale data at times.

Client-Server: A client-server architecture should be used for REST applications. A client is someone who requests resources but is unconcerned about data storage, which is handled

internally by each server, whereas a server is someone who manages resources but is unconcerned about the user interface or user state. They have the ability to evolve on their own. The client does not need to understand business logic, and the server does not need to understand frontend UI.

Layered system: An application architecture must be made up of several layers. There are several intermediate servers between the client and the end server, and each layer has no knowledge of any layer other than its immediate layer. By facilitating load balancing and providing shared caches, intermediary servers can increase system availability.

Code on-demand: It's an optional feature. Servers can also provide executable code to clients, according to this. Compiled components, such as Java applets, and client-side scripts, such as JavaScript, are instances of code on demand.

CHAPTER 4. Proposed Work Flow:

Flow chart:

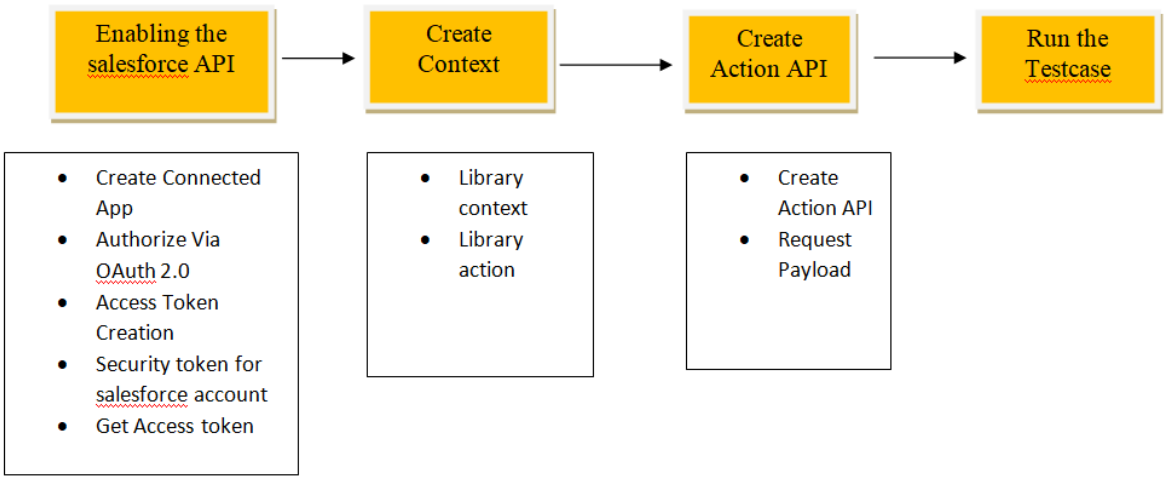


Fig1: Flowchart For Proposed work

Enabling the Salesforce API :

This step consists of the major steps which are as follows:

- Create connected app
- Authorize via OAuth 2.0
- Access Token Creation
- Security Token for salesforce account

- Get access token

Create connected App: This step configures OAuth 2.0 for Salesforce so that any client requesting data must first authenticate through the app's settings.

- Login to your Salesforce org via https://login.salesforce.com/
- Goto Setup->Platform Tools->Apps->App Manager click ‘New Connected App’

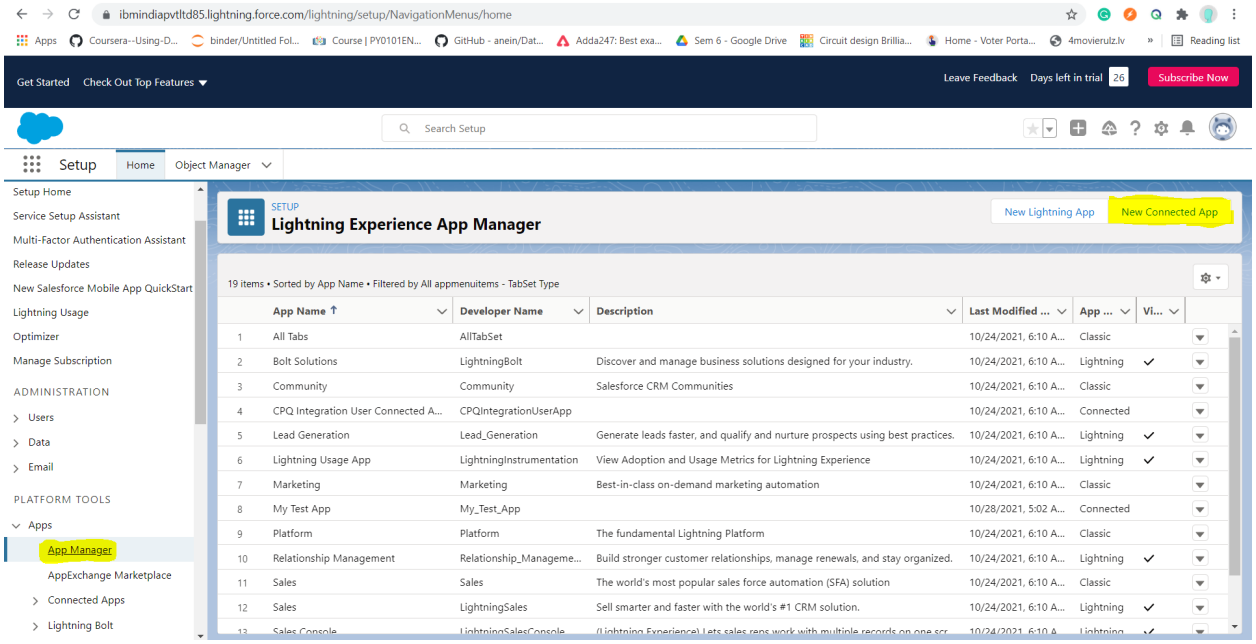


Fig 2: Setup page salesforce

- Fill in the blanks with basic information.
- Check "Enable OAuth Settings" under "API (Enable OAuth Settings)" to bring up a list of options.

- Enter a URL for a callback (make sure it is HTTPS). You can use this URL to authenticate any other user with your app and grant them access to a set of permissions that will allow them to undertake further tasks.
- In the "Selected OAuth Scopes" section, choose from a list of possible OAuth scopes. The options below will provide you access to various users' org data. Make sure you know what data you need access to and what scopes you should give it.
- Save the program.

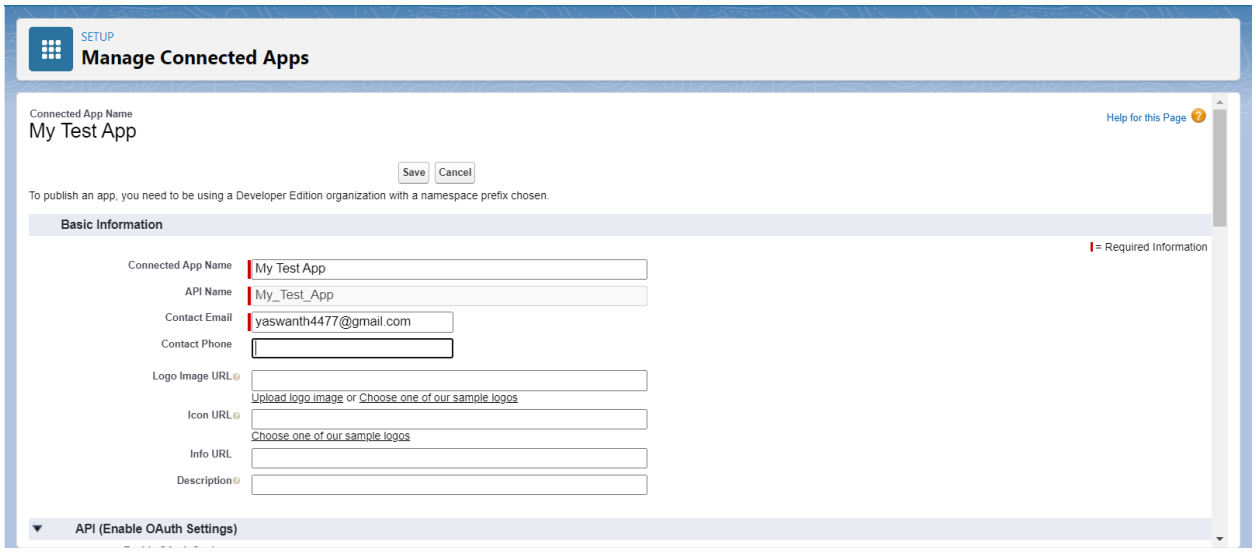


Fig 3: Basic information page

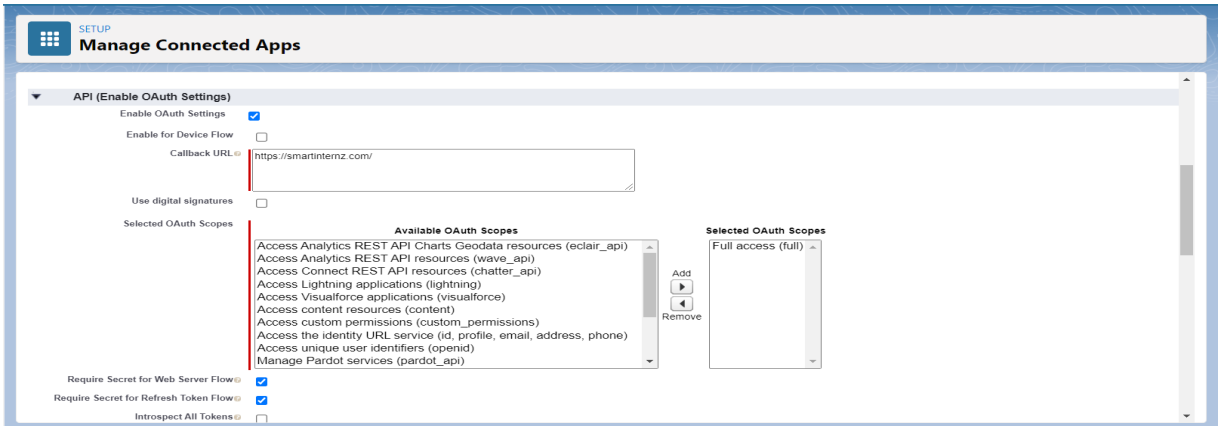


Fig 4: API(enable OAuth Setting)

- After saving, you'll have a 'Consumer Key' and a 'Consumer Secret' in your app. To establish a connection link, you'll need these.

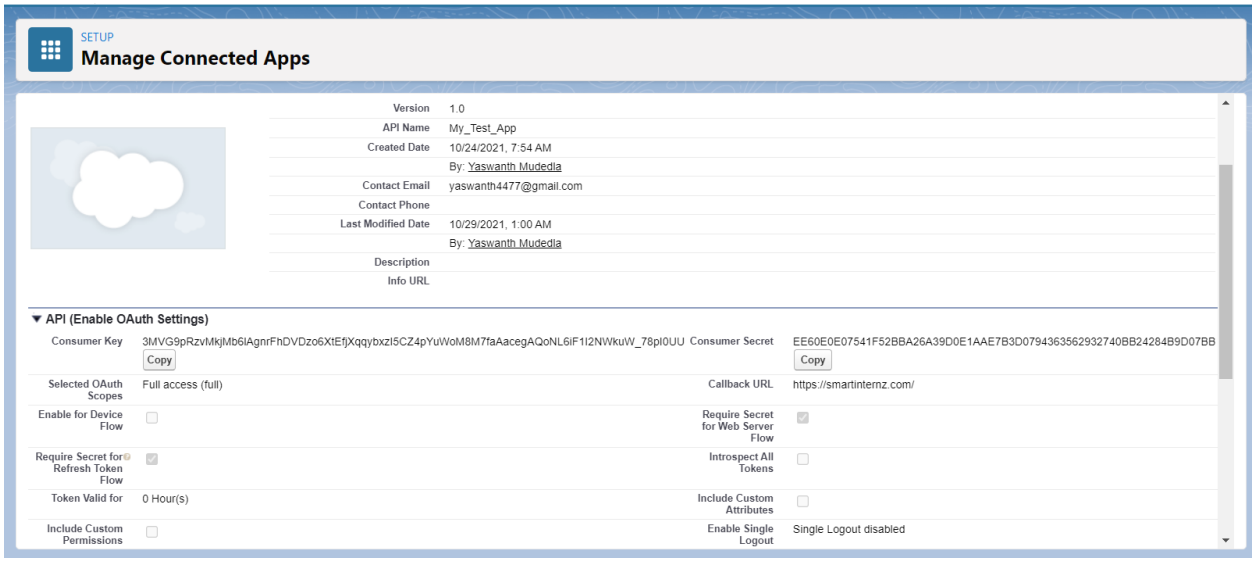


Fig 5: Connected App details

The following are the multiple OAuth 2.0 authentication flows (sets of stages) that you can use to authenticate your application and Salesforce:

- Web server flow, where the server can keep the customer's secret safe. (An authorization code grant type is used.)
- Applications that can't securely store the consumer secret employ the user-agent flow. (Implicit grant type is used.)
- The application has direct access to the user's credentials via a username-password sequence.

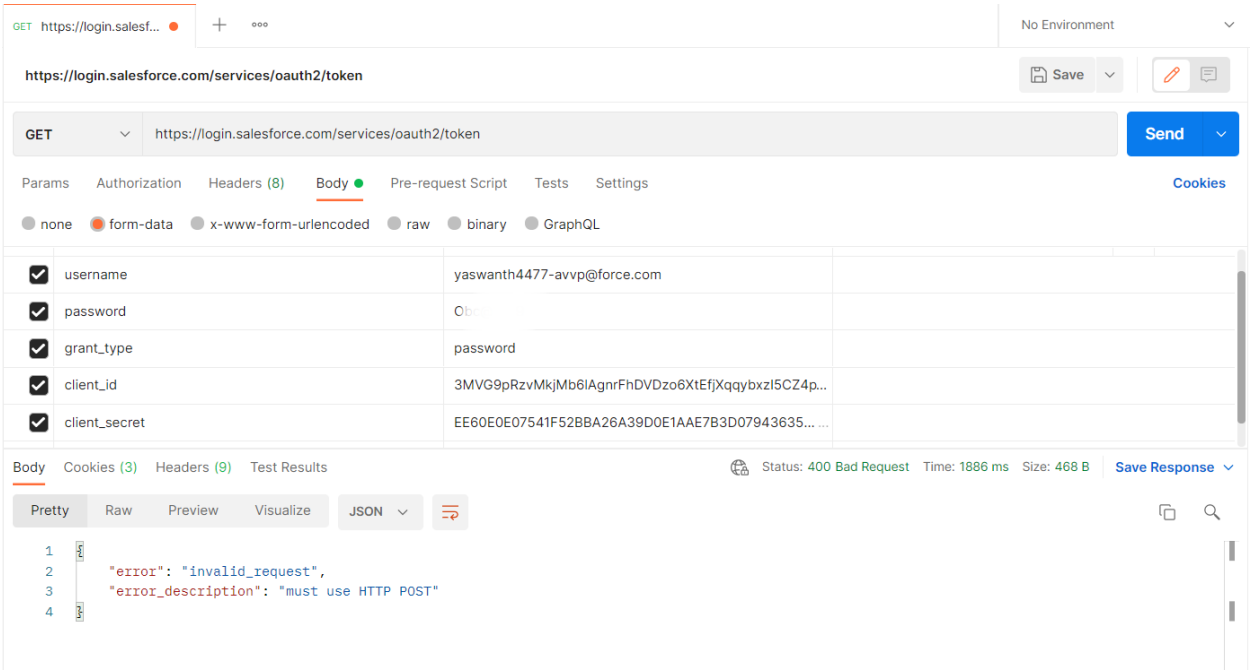
To authenticate the associated app user with Salesforce, we'll use Web server flow. Web Server Flow is suggested for any server application since it employs a client secret as an extra authorization parameter to prevent impersonating servers.

Access Token Creation

Create a **POST** request to the endpoint: https://login.salesforce.com/services/oauth2/token with the following parameters

- Username:
- Password:
- grant_type: password
- client_id: <client_id>
- client_secret: <client_secret>

As a result, your organization is not authenticated due to security reasons As in the below figure.



Security Token For Your Salesforce Account

Salesforce sends an email message from support@salesforce.com with the subject: salesforce.com security token confirmation to the email address connected with the account after you generate one. This email message contains the account's Security Token and is the only place where the Security Token value may be found. When you change your account password, the security token is likewise regenerated (and the old one expires), and you receive a similar email.

Get Access Token

After getting the security token at the email address provided during registration. Add it to the password and send it. So, then you can able to access your Salesforce org with the call our response.

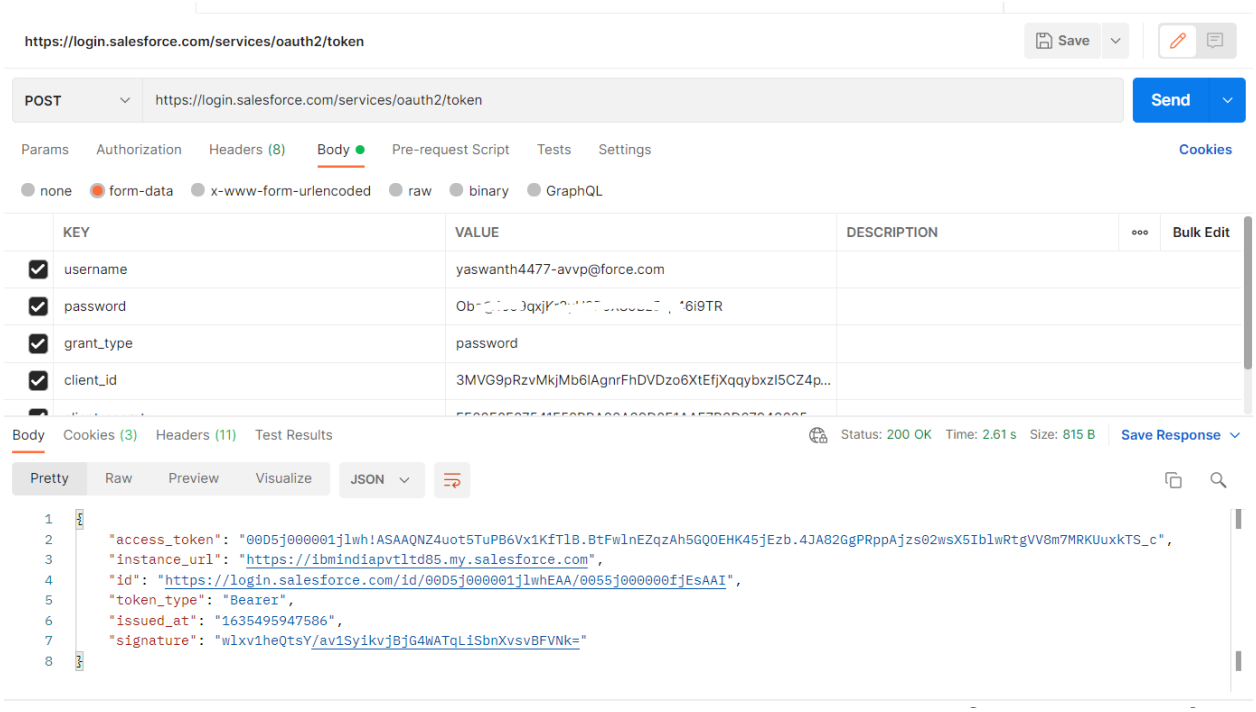


Fig 6: Response for Request

Create Context{6}:

Library Context

- Create a new context from View and name it as Salesforce API Login
- Create a new action Salesforce Action & Select Libray action.

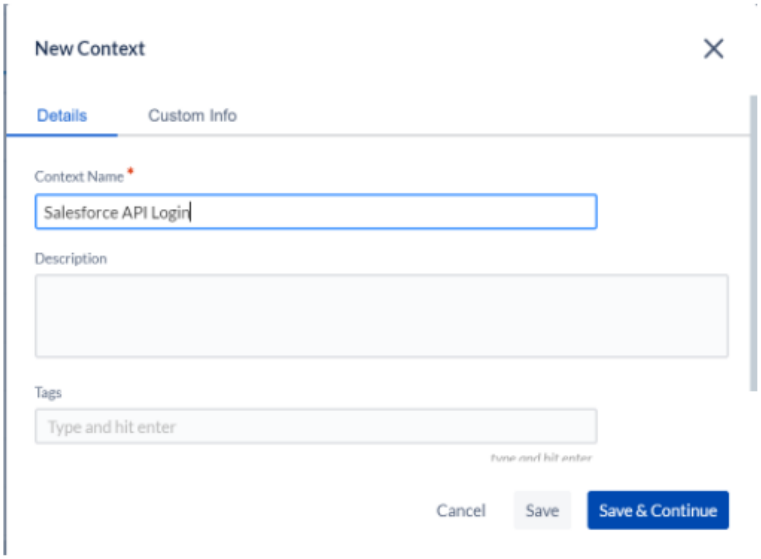


Fig 7: New Context

Library Action

Create a new action for the salesforce API Context.

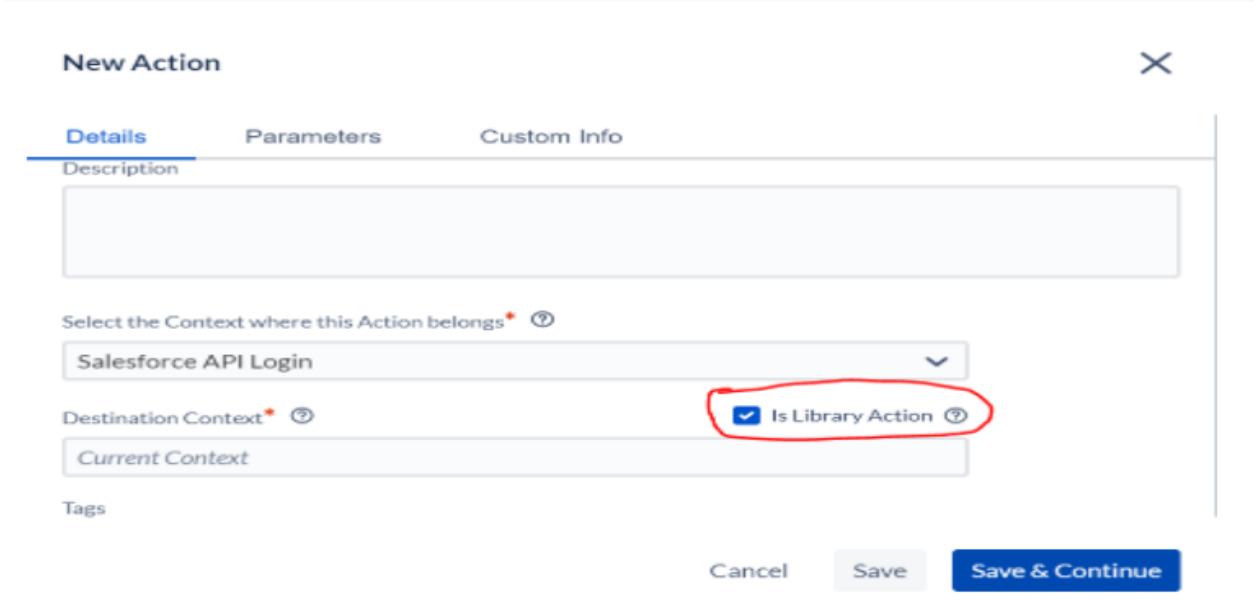


Fig 8: Enabling the Library Action

An **action**[5] is a useful function that a user can execute on a single page (context). The logic for interacting with the browser or performing validations is included in the action.

Search for flights, go to the home page, check your purchase order, and so on.

The element from the Context's repository is used by the Action, which belongs to the Context.

The **Library Action**[4] is a generic Action with no context relevance. Any Action can call this Action, and it can be used anywhere in a Scenario. Not only can you place a Library Action anywhere in a Scenario, but it also has no bearing on the steps that can be performed after it. A Library Action cannot contain statements that interact with the user interface (web, mobile, TE, desktop, etc.) or relate to any UI Elements. Library Actions are often best suited for non-UI test functions such as API, Database, Message Queue, and so on, where you want to achieve modularity while avoiding any contextual constraints.

Create Action API

- Select a **REST API** from Action Library
- Enter the information which is required.
- EndPoint **URL**: https://login.salesforce.com/services/oauth2/token
- Method: **POST**
- Click on **Next**

Request Payload

Invoke ReST Request (POST) With Form Data Type Payload

Invokes a ReSTful POST service with a Form input data payload. Note that the connection name must have already been defined (and authenticated, if required) before executing this command. The Reference Name provided with this Request should be used for subsequent validation of Response.

Next Steps: Select the Payload Type: application/ x- www-form-urlencoded & pass the required details in request body

Replace the keys with your respective IDs. & Click on Send Request

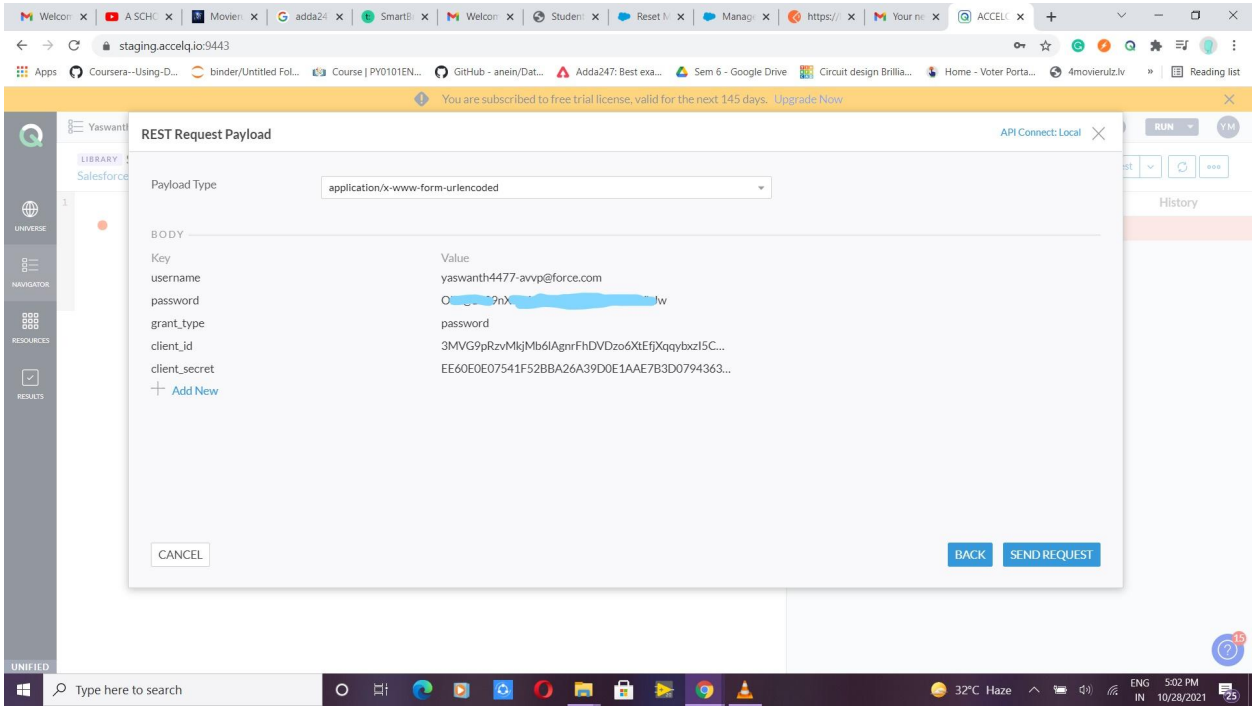


Fig 9: Payload Details

Run The Test Case

Once after successful creation of Action. Run the test case and check.

Chapter 5. RESULT:

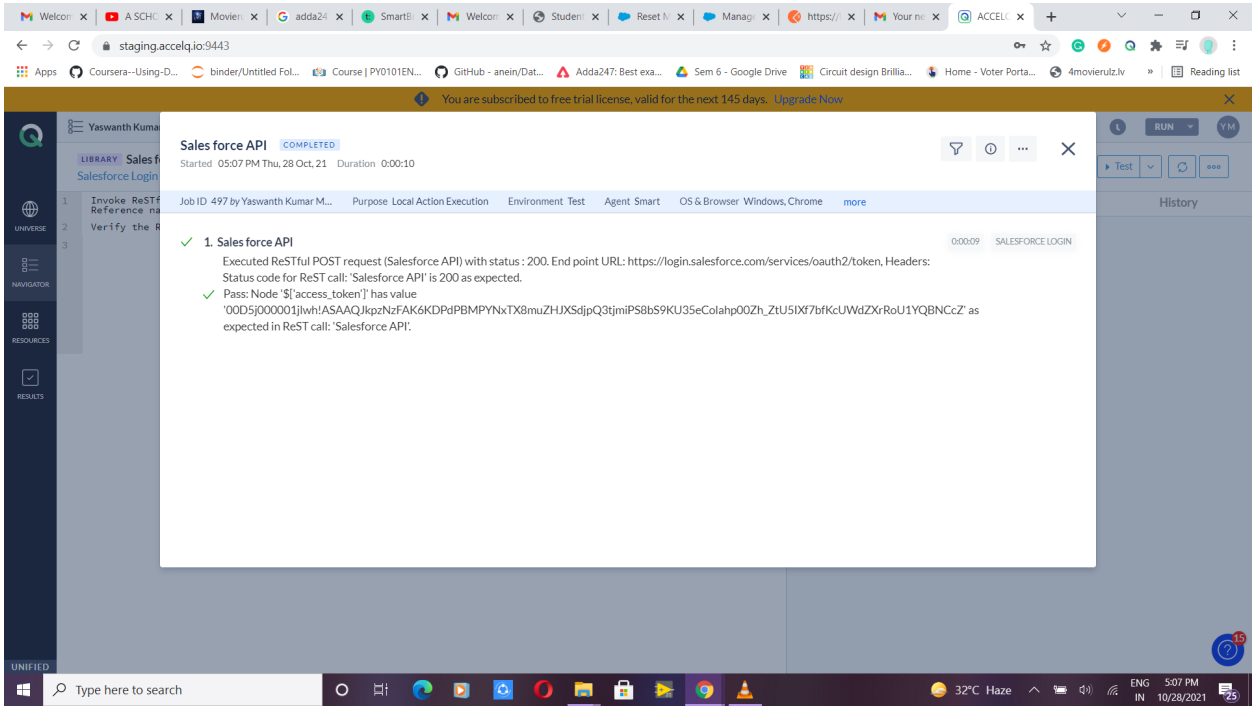


Fig 10: The result page

The above fig is the result for the test case mentioned in the previous steps and also the result shows the verification of the POST request with Status: 200 and also for the verification of the access token which is a regex expression and the test conditions are also passed successfully.

BIBILOGRAPHY :

{1} https://blog.bessereau.eu/assets/pdfs/api_rest.pdf
{2} https://www.geeksforgeeks.org/difference-between-rest-api-and-soap-api/
{3} https://www.geeksforgeeks.org/rest-api-architectural-constraints/
{4} https://support.accelq.com/hc/en-us/articles/360062073612-Library-Action
{5} https://support.accelq.com/hc/en-us/articles/115010131527-What-is-an-Action-
{6} https://support.accelq.com/hc/en-us/articles/360023969851-Creating-a-Context