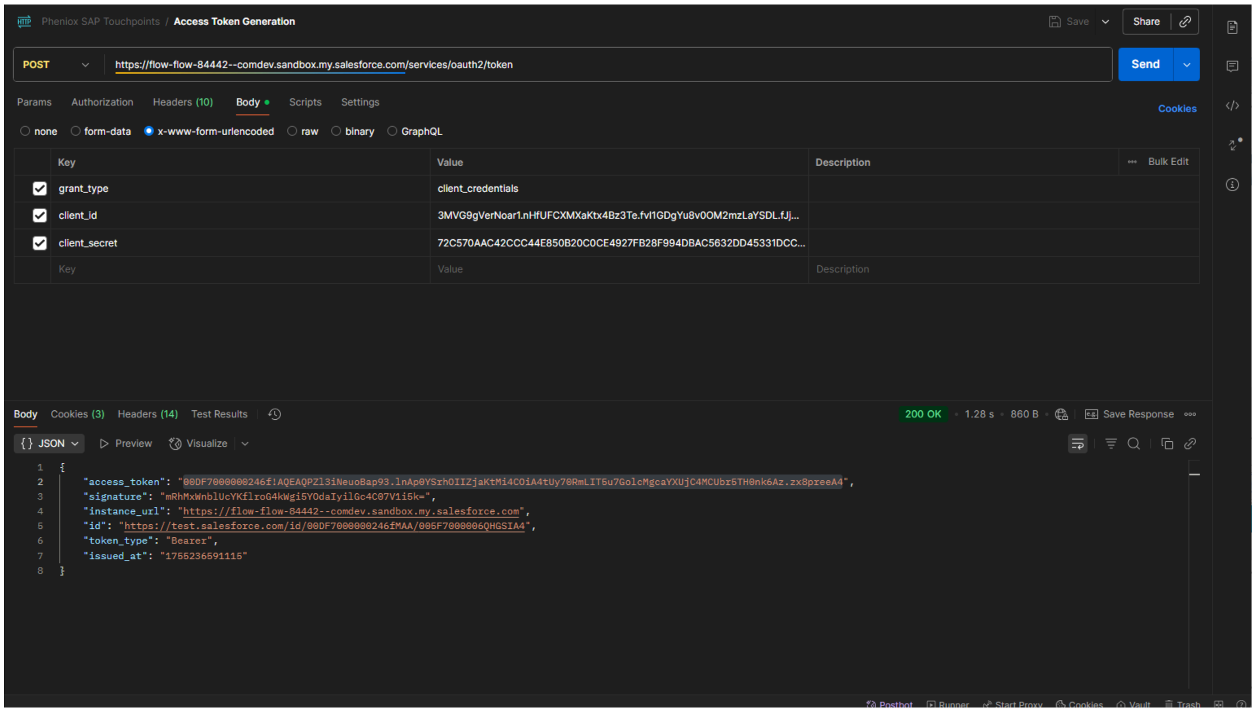


POST https://flow-flow-84442--comdev.sandbox.my.salesforce.com/services/oauth2/token

none · form-data · [✓ x-www-form-urlencoded] · raw · binary · GraphQL

| | Key | Value | Description |
|---|---|---|---|
| ✓ | grant_type | client_credentials | |
| ✓ | client_id | 3MVG9gVerNoar1.nHfUFCXMXaKtx4Bz3Te.fvl1GDgYu8v0OM2mzLaYSDL.fJj... | |
| ✓ | client_secret | 72C570AAC42CCC44E850B20C0CE4927FB28F994DBAC5632DD45331DCC... | |
| | Key | Value | Description |

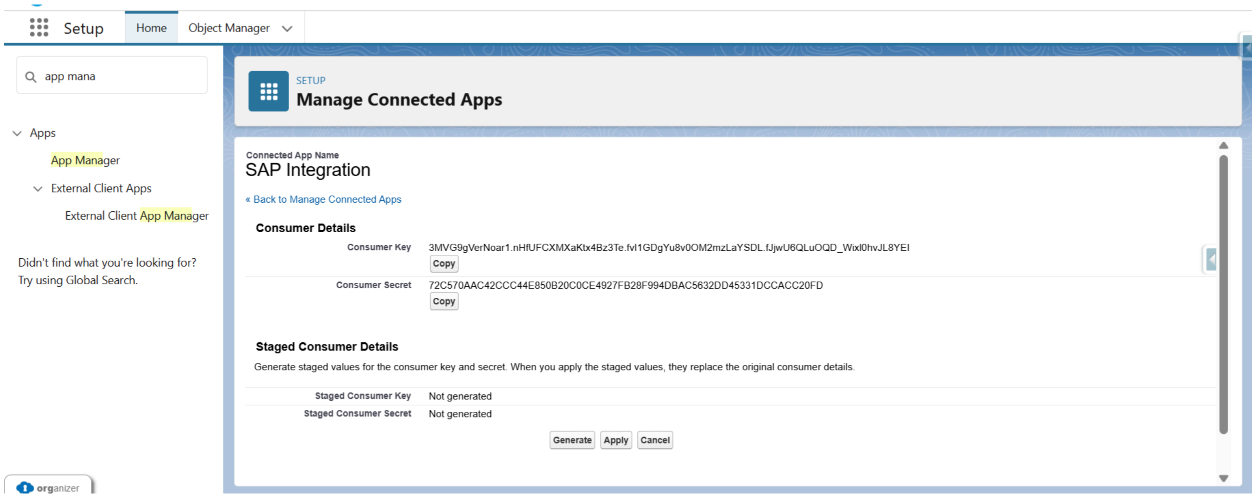
Body — JSON

```json
{
    "access_token": "00DF7000000246f!AQEAQPZl3iNeuoBap93.lnAp0YSrhOIIZjaKtMi4COiA4tUy70RmLIT5u7GolcMgcaYXUjC4MCUbr5TH0nk6Az.zx8preeA4",
    "signature": "mRhMxWnblUcYKflroG4kWgi5YOdaIyilGc4C07V1i5k=",
    "instance_url": "https://flow-flow-84442--comdev.sandbox.my.salesforce.com",
    "id": "https://test.salesforce.com/id/00DF7000000246fMAA/005F7000006QHGSIA4",
    "token_type": "Bearer",
    "issued_at": "1755236591115"
}
```

Endpoint Url :
https://flow-flow-84442--comdev.sandbox.my.salesforce.com/services/oauth2/token

How to get client id and secret key :

1. App manager —-------> Select the connected app —--------------> view —-------------------> get customer details —-----------> verification code —--------> will get the customer id and secret key



Connected App Name
SAP Integration

**Consumer Details**

| | |
|---|---|
| Consumer Key | 3MVG9gVerNoar1.nHfUFCXMXaKtx4Bz3Te.fvl1GDgYu8v0OM2mzLaYSDL.fJjwU6QLuOQD_Wixl0hvJL8YEI |
| Consumer Secret | 72C570AAC42CCC44E850B20C0CE4927FB28F994DBAC5632DD45331DCCACC20FD |

**Staged Consumer Details**

Generate staged values for the consumer key and secret. When you apply the staged values, they replace the original consumer details.

| | |
|---|---|
| Staged Consumer Key | Not generated |
| Staged Consumer Secret | Not generated |

Customer API :

## Mandatory Fields :

- accountGroup
- customerCode
- firstName
- lastName
- middleName
- searchTerm1
- street
- street2
- street3
- postalCode
- city
- country
- region
- gstNo
- adhaarNumber
- panCard

Request Body :

```
[
{
  "accountGroup": "C002",
 "customerCode": "89011",
  "title": "2",
  "firstName": "Ravi",
  "lastName": "Chopra",
  "middleName": "Ashwin",
  "language": "EN",
  "searchTerm1": "123",
  "street": "101 New Avenue",
  "street2": "MG Road",
  "street3": "Kandivli",
  "street4": "",
  "postalCode": "400101",
  "city": "Mumbai",
  "country": "IN",
  "region": "27",
  "email": "ravi.chopra@gmail.com",
  "mobileNumber": "9898980009",
  "gstNo": "22AAAAA0000A1Z5",
  "adhaarNumber": "4444 3333 1111",
  "panCard": "ABCTY1234D",
  "paymentTerms": "0001",
  "acctAssignmentGroup": "01",
  "taxClassCgst": "1",
  "taxClassSgst": "1",
  "taxClassIgst": "1",
  "taxClassUtgst": "1",
  "companyCode": "1000",
  "reconciliationAccount": "23400001"
}
]
```

## Responses:

Insertion :

```
[
    {
        "accountId": "001F700001qyQ2oIAE",
        "message": "New account created successfully",
        "status": "Success",
        "externalId": "1500"
    }
]
```

Updation :

```
[
    {
        "accountId": "001F700001qyQ2oIAE",
        "message": "Existing account updated successfully",
        "status": "Success",
        "externalId": "1500"
    }
]
```

Required Fields Missing Validation Error :

```
[
    {
        "accountId": null,
        "message": "Missing required fields for new account: accountGroup",
        "status": "Failure",
        "externalId": "1501"
    }
]
```

End Point url :

https://flow-flow-84442--comdev.sandbox.my.salesforce.com/services/apexrest/sap/v1/customer/

Apex Class :

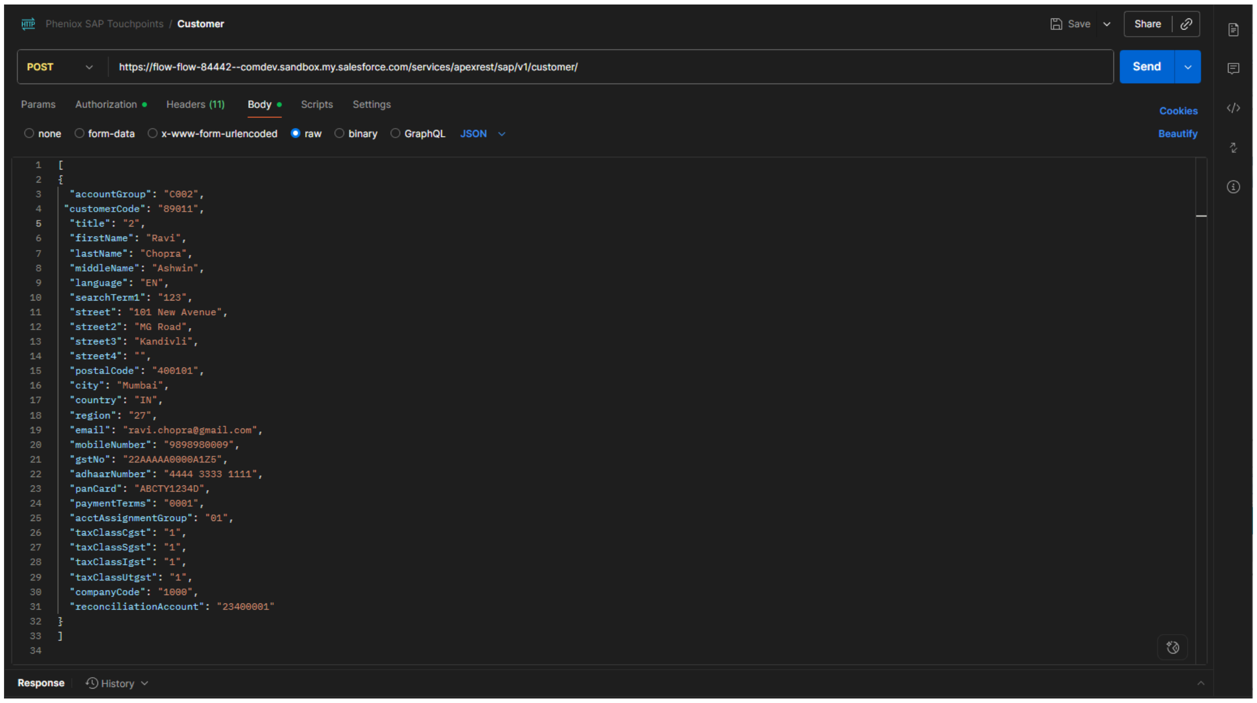
```
//**************************************************************************************************************
****//
//Name       :  CustomerApi
//Description:  REST API for Account of type Customer master creation coming from SAP to
Salesforce
//Created By :  Vijay Kumar
//Date       :  Jul 02 2025
//**************************************************************************************************************
***//
/*Developer              Date              JIRA              Description
* Vijay Kumar           Jul 02 2025       PCRM-14952        REST API for Account of type
Customer master creation coming from SAP to Salesforce
*/
@RestResource(urlMapping='/sap/v1/customer/*')
global without sharing class CustomerApi {

/*****************************************************************************************
* @Description  :  Getting the data fron SAP and sending it to the helper class by adding the
record type in the json (For Creation)
* @Param        :  doPost
* @Return       :  void
* Created By    :  Vijay Kumar
* Date          :  02 Jul 2025
******************************************************************************************/
   @HttpPost
```

```
    global static void doPost() {
        handleCustomerData(true);
    }

/*****************************************************************************************
* @Description  :  Getting the data fron SAP and sending it to the helper class by adding the
record type in the json (For Updation)
* @Param        :  doPatch
* @Return       :  void
* Created By    :  Vijay Kumar
* Date          :  02 Jul 2025
*****************************************************************************************/

    @HttpPatch
    global static void doPatch() {
        handleCustomerData(false);
    }

/*****************************************************************************************
* @Description  :  Helper Method
* @Param        :  handleCustomerData
* @Return       :  void
* Created By    :  Vijay Kumar
* Date          :  02 Jul 2025
*****************************************************************************************/

    private static void handleCustomerData(boolean isInsert){
        String requestBody = '';
        List<AccountMasterController.AccountData> accountDataList = null;

        try {
            requestBody = RestContext.request.requestBody.toString();

            System.debug(requestBody);

            accountDataList =
(List<AccountMasterController.AccountData>)JSON.deserialize(requestBody,
List<AccountMasterController.AccountData>.class);

            String recordTypeID = '';
            try {
                recordTypeID =
Schema.SObjectType.Account.getRecordTypeInfosByDeveloperName().get('Customer').getRec
ordTypeId();
```

```
        } catch (Exception e) {
            System.debug('Error fetching Record Type ID: ' + e.getMessage());
            throw new AuraHandledException('Failed to retrieve Record Type ID for Account.');
        }

        System.debug('Record Type ID: ' + recordTypeID);
        for (AccountMasterController.AccountData accountData : accountDataList) {
            accountData.recordTypeId = recordTypeID;
        }

        Map<String, Object> responseMap =
AccountMasterController.processAccountData(accountDataList, isInsert);

        List<Map<String, String>> allResponses = AccountMasterController.getAllResponses();
        Boolean hasFailures = false;
        Boolean hasMissingDetails = false;
        Boolean hasUnauthorized = false;

        for (Map<String, String> response : allResponses) {
            if (response.get('status') == 'Failure') {
                hasFailures = true;
                if (response.get('message').contains('Missing details')) {
                    hasMissingDetails = true;
                } else if (response.get('message').contains('Unauthorized')) {
                    hasUnauthorized = true;
                }
            }
        }
        if (hasMissingDetails) {
            RestContext.response.statusCode = 400;
        } else if (hasUnauthorized) {
            RestContext.response.statusCode = 410;
        } else if (hasFailures) {
            RestContext.response.statusCode = 500;
        } else {
            RestContext.response.statusCode = 200;
        }
        List<Map<String, String>> filteredResponses = new List<Map<String, String>>();
        for (Map<String, String> response : allResponses) {
            Map<String, String> filteredResponse = new Map<String, String>();
            filteredResponse.put('externalId', response.get('externalId'));
            filteredResponse.put('status', response.get('status'));
            filteredResponse.put('message', response.get('message'));
            filteredResponse.put('accountId', response.get('accountId'));
```

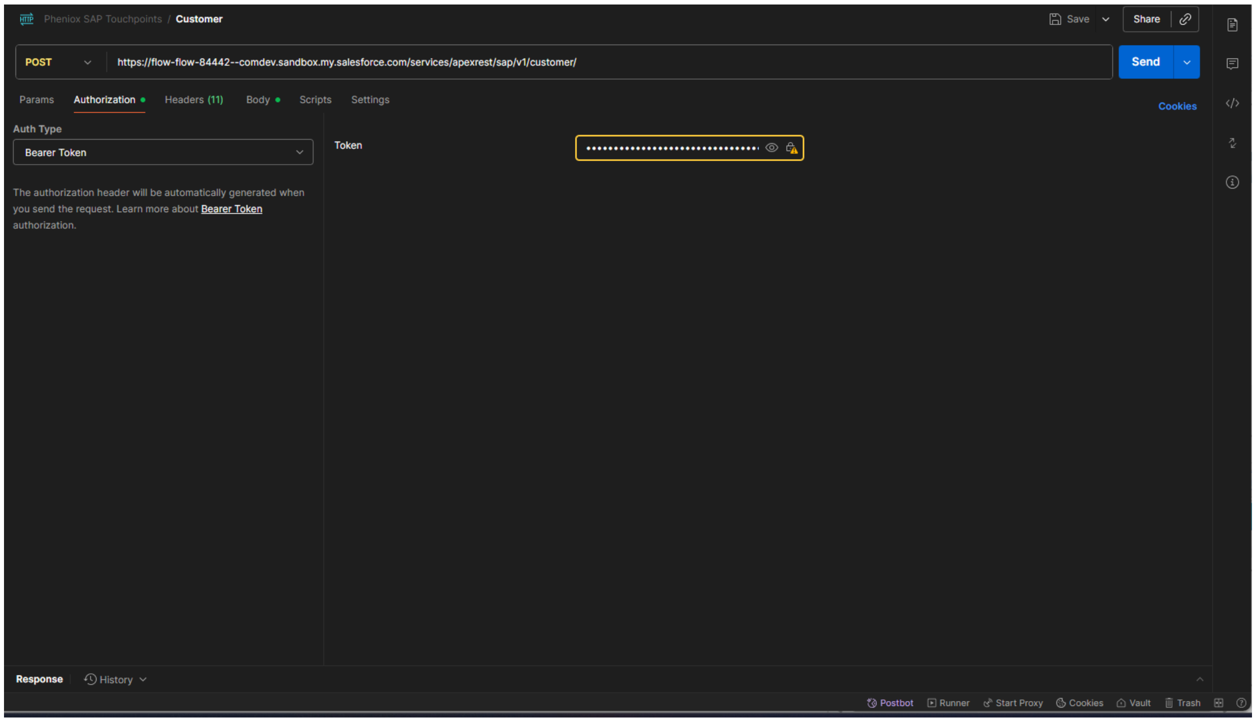
```
                filteredResponses.add(filteredResponse);
            }
            RestContext.response.addHeader('Content-Type', 'application/json');
            RestContext.response.responseBody =
Blob.valueOf(JSON.serialize(filteredResponses));

        } catch (Exception e) {
            System.debug('Error in CustomerApi: ' + e.getMessage());
            List<Map<String, String>> errorResponse = new List<Map<String, String>>();
            Map<String, String> error = new Map<String, String>();
            error.put('status', 'Failure');
            error.put('message', e.getMessage());
            error.put('externalId', '');
            error.put('sfdcid', '');
            errorResponse.add(error);
            RestContext.response.statusCode = 500;
            RestContext.response.addHeader('Content-Type', 'application/json');
            RestContext.response.responseBody = Blob.valueOf(JSON.serialize(errorResponse));
             ExceptionLogger.ExceptionLogPayload logPayload = new
ExceptionLogger.ExceptionLogPayload()
                .withComponentName('CustomerApi')
                .withClassName('CustomerApi')
                .withMethodName('handleCustomerData')
                .withException(e)
                .withErrorType('RestApiError')
                .withRequestBody(requestBody)
                .withEndpoint('/sap/v1/customer/')
                .withIntegrationType('REST_API')
                .withDomain('SAP')
                .withDescription('Error occurred in Customer API while processing SAP data.
Operation: ' + (isInsert ? 'Create' : 'Update') +
                               ', Records Count: ' + (accountDataList != null ?
String.valueOf(accountDataList.size()) : 'Unknown') +
                               ', Request Body Length: ' + (String.isNotBlank(requestBody) ?
String.valueOf(requestBody.length()) : '0'));

            ExceptionLogger.logException(logPayload);
        }
    }
}
```

POSTMAN POSTS :


Pheniox SAP Touchpoints / **Customer**

POST https://flow-flow-84442--comdev.sandbox.my.salesforce.com/services/apexrest/sap/v1/customer/

```
[
{
  "accountGroup": "C002",
 "customerCode": "89011",
  "title": "2",
  "firstName": "Ravi",
  "lastName": "Chopra",
  "middleName": "Ashwin",
  "language": "EN",
  "searchTerm1": "123",
  "street": "101 New Avenue",
  "street2": "MG Road",
  "street3": "Kandivli",
  "street4": "",
  "postalCode": "400101",
  "city": "Mumbai",
  "country": "IN",
  "region": "27",
  "email": "ravi.chopra@gmail.com",
  "mobileNumber": "9898980009",
  "gstNo": "22AAAAA0000A1Z5",
  "adhaarNumber": "4444 3333 1111",
  "panCard": "ABCTY1234D",
  "paymentTerms": "0001",
  "acctAssignmentGroup": "01",
  "taxClassCgst": "1",
  "taxClassSgst": "1",
  "taxClassIgst": "1",
  "taxClassUtgst": "1",
  "companyCode": "1000",
  "reconciliationAccount": "23400001"
}
]
```


Pheniox SAP Touchpoints / **Customer**

POST https://flow-flow-84442--comdev.sandbox.my.salesforce.com/services/apexrest/sap/v1/customer/

Auth Type: Bearer Token

The authorization header will be automatically generated when you send the request. Learn more about Bearer Token authorization.

Token: ••••••••••••••••••••••••••••••••

## Vendor API :

## Mandatory Fields :

- accountGroup

- vendorCode
- firstName
- lastName
- middleName
- searchTerm1
- street
- street2
- street3
- postalCode
- city
- country
- region
- gstNo
- adhaarNumber
- panCard

Request Body :

```
[
{
  "accountGroup": "C001",
  "vendorCode": "2000",
  "title": "2",
  "firstName": "Ravi",
  "lastName": "Chopra",
  "middleName": "Ashwin",
  "language": "EN",
  "searchTerm1": "123",
  "street": "101 New Avenue",
  "street2": "MG Road",
  "street3": "Kandivli",
  "street4": "",
  "postalCode": "400101",
  "city": "Mumbai",
  "country": "IN",
  "region": "27",
  "email": "ravi.chopra@gmail.com",
  "mobileNumber": "9898980009",
  "gstNo": "22AAAAA0000A1Z5",
  "adhaarNumber": "4444 3333 1111",
  "panCard": "ABCTY1234D",
  "paymentTerms": "0001",
  "acctAssignmentGroup": "01",
  "taxClassCgst": "1",
  "taxClassSgst": "1",
  "taxClassIgst": "1",
  "taxClassUtgst": "1",
  "companyCode": "1000",
  "reconciliationAccount": "23400001"
}
]
```

Responses :

Insertion :

```
[
    {
        "accountId": "001F700001qyQ2oIAE",
        "message": "New account created successfully",
        "status": "Success",
        "externalId": "1500"
    }
]
```

Updation :

```
[
    {
        "accountId": "001F700001qyQ2oIAE",
        "message": "Existing account updated successfully",
        "status": "Success",
        "externalId": "1500"
    }
]
```

Required Fields Missing Validation Error :

```
[
    {
        "accountId": null,
        "message": "Missing required fields for new account: accountGroup",
        "status": "Failure",
        "externalId": "1501"
    }
]
```

## Endpoint Url :

https://flow-flow-84442--comdev.sandbox.my.salesforce.com/services/apexrest/sap/v1/vendor/

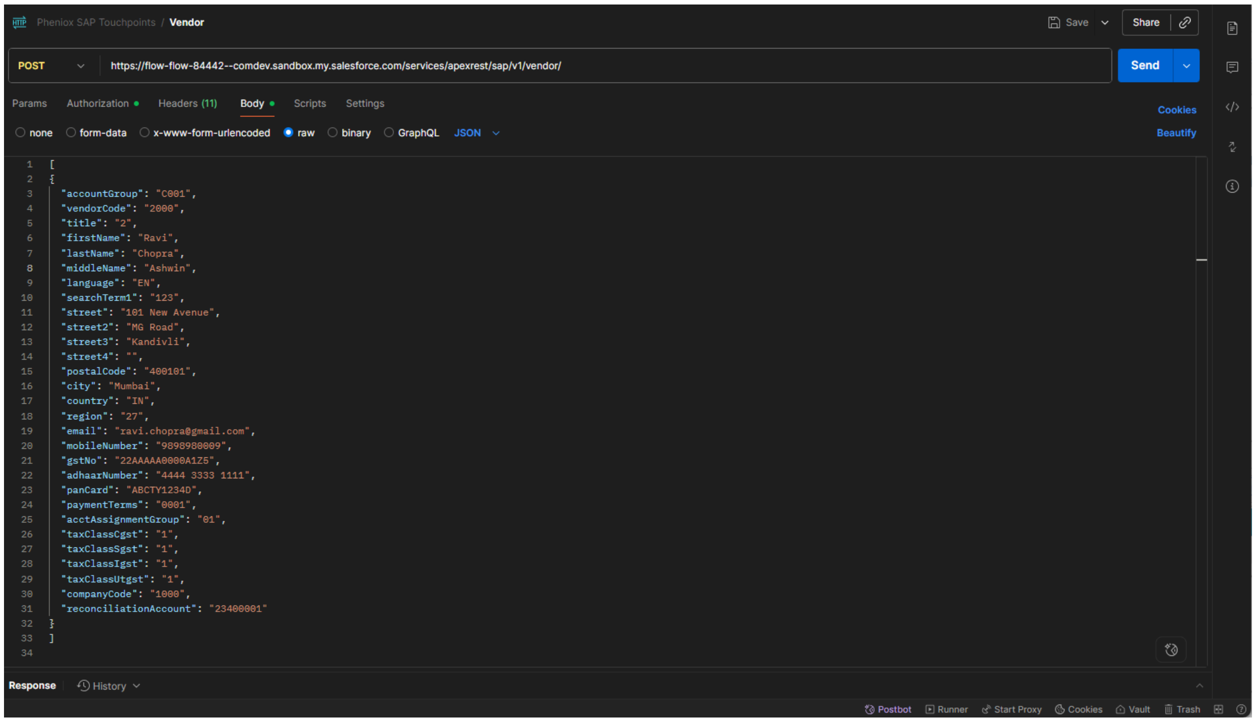
### Apex class :

```
//*****************************************************************************************************************//
//Name       :  VendorApi
//Description:  Helper class for all the other webservice classes so that it would be easily modified
//Created By :  Vijay Kumar
//Date       :  Jul 02 2025
//*****************************************************************************************************************//
/*Developer             Date              JIRA            Description
* Vijay Kumar          Jul 02 2025       PCRM-14952        REST API for Account of type Vendor master creation coming
from SAP to Salesforce
*/
@RestResource(urlMapping='/sap/v1/vendor/*')
global without sharing class VendorApi {

/*****************************************************************************************
* @Description  :  Getting the data fron SAP and sending it to the helper class by adding the record type in the json
(For Creation)
* @Param        :  doPost
* @Return       :  void
* Created By    :  Vijay Kumar
* Date          :  02 Jul 2025
*****************************************************************************************/
   @HttpPost
   global static void doPost() {
       handleVendorData(true);
   }

/*****************************************************************************************
```

```
* @Description  :  Getting the data fron SAP and sending it to the helper class by adding the record type in the json
(For Updation)
* @Param       :  doPatch
* @Return      :  void
* Created By   :  Vijay Kumar
* Date         :  02 Jul 2025
*************************************************************************************/
    @HttpPatch
   global static void doPatch() {
      handleVendorData(false);
   }

/*************************************************************************************
* @Description  :  Helper Method
* @Param       :  handleVendorData
* @Return      :  void
* Created By   :  Vijay Kumar
* Date         :  02 Jul 2025
*************************************************************************************/

   private static void handleVendorData(Boolean isInsert){
       String requestBody;
      List<AccountMasterController.AccountData> accountDataList;

      try {
         requestBody = RestContext.request.requestBody.toString();

         System.debug(requestBody);

         accountDataList = (List<AccountMasterController.AccountData>)JSON.deserialize(requestBody,
List<AccountMasterController.AccountData>.class);

         String recordTypeID = '';
         try {
            recordTypeID =
Schema.SObjectType.Account.getRecordTypeInfosByDeveloperName().get('Broker').getRecordTypeId();
         } catch (Exception e) {
            System.debug('Error fetching Record Type ID: ' + e.getMessage());
            throw new AuraHandledException('Failed to retrieve Record Type ID for Account.');
         }

         System.debug('Record Type ID: ' + recordTypeID);
         for (AccountMasterController.AccountData accountData : accountDataList) {
            accountData.recordTypeId = recordTypeID;
         }

         Map<String, Object> responseMap = AccountMasterController.processAccountData(accountDataList, isInsert);

         List<Map<String, String>> allResponses = AccountMasterController.getAllResponses();
         Boolean hasFailures = false;
         Boolean hasMissingDetails = false;
         Boolean hasUnauthorized = false;
```

```
        for (Map<String, String> response : allResponses) {
            if (response.get('status') == 'Failure') {
                hasFailures = true;
                if (response.get('message').contains('Missing details')) {
                    hasMissingDetails = true;
                } else if (response.get('message').contains('Unauthorized')) {
                    hasUnauthorized = true;
                }
            }
        }
        if (hasMissingDetails) {
            RestContext.response.statusCode = 400;
        } else if (hasUnauthorized) {
            RestContext.response.statusCode = 410;
        } else if (hasFailures) {
            RestContext.response.statusCode = 500;
        } else {
            RestContext.response.statusCode = 200;
        }
        List<Map<String, String>> filteredResponses = new List<Map<String, String>>();
        for (Map<String, String> response : allResponses) {
            Map<String, String> filteredResponse = new Map<String, String>();
            filteredResponse.put('externalId', response.get('externalId'));
            filteredResponse.put('status', response.get('status'));
            filteredResponse.put('message', response.get('message'));
            filteredResponse.put('accountId', response.get('accountId'));
            filteredResponses.add(filteredResponse);
        }
        RestContext.response.addHeader('Content-Type', 'application/json');
        RestContext.response.responseBody = Blob.valueOf(JSON.serialize(filteredResponses));

    } catch (Exception e) {
        System.debug('Error in CustomerApi: ' + e.getMessage());
        List<Map<String, String>> errorResponse = new List<Map<String, String>>();
        Map<String, String> error = new Map<String, String>();
        error.put('status', 'Failure');
        error.put('message', e.getMessage());
        error.put('externalId', '');
        error.put('sfdcid', '');
        errorResponse.add(error);
        RestContext.response.statusCode = 500;
        RestContext.response.addHeader('Content-Type', 'application/json');
        RestContext.response.responseBody = Blob.valueOf(JSON.serialize(errorResponse));
         ExceptionLogger.ExceptionLogPayload logPayload = new ExceptionLogger.ExceptionLogPayload()
            .withComponentName('VendorApi')
            .withClassName('VendorApi')
            .withMethodName('handleVendorData')
            .withException(e)
            .withErrorType('RestApiError')
            .withRequestBody(requestBody)
            .withEndpoint('/sap/v1/vendor/')
            .withIntegrationType('REST_API')
            .withDomain('SAP')
            .withDescription('Error occurred in Vendor API while processing SAP data. Operation: ' + (isInsert ? 'Create' : 
'Update') + 
                    ', Records Count: ' + (accountDataList != null ? String.valueOf(accountDataList.size()) : 'Unknown') + 
                    ', Request Body Length: ' + (String.isNotBlank(requestBody) ? String.valueOf(requestBody.length()) : 
'0'));

          ExceptionLogger.logException(logPayload);
        }
    }
}
```

Postman posts :



POST https://flow-flow-84442--comdev.sandbox.my.salesforce.com/services/apexrest/sap/v1/vendor/

```json
[
{
  "accountGroup": "C001",
  "vendorCode": "2000",
  "title": "2",
  "firstName": "Ravi",
  "lastName": "Chopra",
  "middleName": "Ashwin",
  "language": "EN",
  "searchTerm1": "123",
  "street": "101 New Avenue",
  "street2": "MG Road",
  "street3": "Kandivli",
  "street4": "",
  "postalCode": "400101",
  "city": "Mumbai",
  "country": "IN",
  "region": "27",
  "email": "ravi.chopra@gmail.com",
  "mobileNumber": "9898980009",
  "gstNo": "22AAAAA0000A1Z5",
  "adhaarNumber": "4444 3333 1111",
  "panCard": "ABCTY1234D",
  "paymentTerms": "0001",
  "acctAssignmentGroup": "01",
  "taxClassCgst": "1",
  "taxClassSgst": "1",
  "taxClassIgst": "1",
  "taxClassUtgst": "1",
  "companyCode": "1000",
  "reconciliationAccount": "23400001"
}
]
```

AccountMasterController Apex Class :

```
//*****************************************************************************************************//
//Name       :  AccountMasterController
//Description:  Helper class for all the other webservice classes so that it would be easily modified
//Created By :  Vijay Kumar
//Date       :  Jul 02 2025
//*****************************************************************************************************//
/*Developer             Date              JIRA              Description
* Vijay Kumar          Jul 02 2025        PCRM-14952          Helper class for all the other webservice classes so that it 
would be easily modified
*/
public class AccountMasterController {
    public class AccountData {
        public String accountGroup;
        public String customerCode;
        public String vendorCode;
        public String firstName;
        public String lastName;
        public String middleName;
        public String title;
        public String language;
        public String email;
        public String mobileNumber;
        public String acctAssignmentGroup;
        public String taxClassCgst;
        public String taxClassSgst;
        public String taxClassIgst;
        public String taxClassUtgst;
        public String paymentTerms;
        public String companyCode;
        public String searchTerm1;
        public String street;
        public String street2;
        public String street3;
        public String street4;
        public String postalCode;
        public String reconciliationAccount;
        public String city;
        public String country;
        public String region;
        public String gstNo;
        public String adhaarNumber;
        public String panCard;
        public String brandName;
        public String format;
        public String pricePoint;
        public String category;
        public String brandgroup;
        public String recordTypeId;
    }

    public static List<Map<String, String>> allResponses = new List<Map<String, String>>();

    /**************************************************************************************
* @Description  :  Catching hold of the data coming from the webservice classes and checking for the conditions
according to the requirement and update or insert the account based on the customer code
* @Param        :  Map<String, Object>
* @Return       :  void
* Created By    :  Vijay Kumar
* Date          :  02 Jul 2025
**************************************************************************************/
    public static Map<String, Object> processAccountData(List<AccountData> accountDataList, Boolean isInsert) {
        allResponses.clear();
        List<Account> accountsToProcess = new List<Account>();
```

```
List<String> customerCodes = new List<String>();
Map<String, Account> existingAccountsMap = new Map<String, Account>();
Map<Integer, String> indexToCustomerCode = new Map<Integer, String>();

try {
    for (AccountData data : accountDataList) {
        String code = getCodeValue(data);
        if (!String.isBlank(code)) {
            customerCodes.add(code);
        }
    }

    if (!customerCodes.isEmpty()) {
        for (Account acc : [
            SELECT Id, Customer_Code__c FROM Account WHERE Customer_Code__c IN :customerCodes
        ]) {
            existingAccountsMap.put(acc.Customer_Code__c, acc);
        }
    }
    for (Integer i = 0; i < accountDataList.size(); i++) {
        AccountData data = accountDataList[i];
        String code = getCodeValue(data);
        List<String> missingFields = new List<String>();
        String billingStreet = '';
        if (!String.isBlank(data.street)) billingStreet += data.street + ' ';
        if (!String.isBlank(data.street2)) billingStreet += data.street2 + ' ';
        if (!String.isBlank(data.street3)) billingStreet += data.street3 + ' ';
        if (!String.isBlank(data.street4)) billingStreet += data.street4 + ' ';
        billingStreet = billingStreet.trim();

        if (isInsert && existingAccountsMap.containsKey(code)) {
            String codeField = getCodeFieldName(data);
            allResponses.add(new Map<String, String>{
                'externalId' => code,
                    'status' => 'Failure',
                    'message' => 'Record already exists with given ' + codeField + ': ' + code,
                    'accountId' => '',
                    'operation' => 'Insert'
                    });
            continue;
        }

        if (!isInsert && !existingAccountsMap.containsKey(code)) {
            String codeField = getCodeFieldName(data);
            allResponses.add(new Map<String, String>{
                'externalId' => code,
                    'status' => 'Failure',
                    'message' => 'No existing record found with ' + codeField + ': ' + code,
                    'accountId' => '',
                    'operation' => 'Update'
                    });
            continue;
        }

        if (isInsert) {
            if (String.isBlank(data.accountGroup)) missingFields.add('accountGroup');
            if (String.isBlank(code)) missingFields.add(getCodeFieldName(data));
            if (String.isBlank(data.firstName)) missingFields.add('firstName');
            if (String.isBlank(data.lastName)) missingFields.add('lastName');
            if (String.isBlank(data.middleName)) missingFields.add('middleName');
            if (String.isBlank(data.searchTerm1)) missingFields.add('searchTerm1');
            if (String.isBlank(data.street)) missingFields.add('street');
            if (String.isBlank(data.street2)) missingFields.add('street2');
            if (String.isBlank(data.street3)) missingFields.add('street3');
            if (String.isBlank(data.postalCode)) missingFields.add('postalCode');
            if (String.isBlank(data.city)) missingFields.add('city');
            if (String.isBlank(data.country)) missingFields.add('country');
            if (String.isBlank(data.region)) missingFields.add('region');
            if (String.isBlank(data.gstNo)) missingFields.add('gstNo');
            if (String.isBlank(data.adhaarNumber)) missingFields.add('adhaarNumber');
            if (String.isBlank(data.panCard)) missingFields.add('panCard');

            if (!missingFields.isEmpty()) {
                allResponses.add(new Map<String, String>{
                    'externalId' => code,
                        'status' => 'Failure',
                        'message' => 'Missing required fields for new account: ' + String.join(missingFields, ', '),
                        'accountId' => '',
                        'operation' => 'Insert'
                        });
                continue;
            }
        }
        Account acc = isInsert ? new Account() : existingAccountsMap.get(code);
        acc.Name = data.firstName + ' ' + data.lastName;
        acc.Account_Group__c = data.accountGroup;
        acc.Customer_Code__c = code;
        acc.First_Name__c = data.firstName;
        acc.Last_Name__c = data.lastName;
        acc.Middle_Name__c = data.middleName;
        acc.Search_Term_1__c = data.searchTerm1;
        acc.BillingStreet = billingStreet;
        acc.BillingPostalCode = data.postalCode;
        acc.BillingCity = data.city;
        acc.BillingCountry = data.country;
        acc.Region__c = data.region;
        acc.GSTIN__c = data.gstNo;
        acc.Adhaar_Number__c = data.adhaarNumber;
        acc.Pan_Card_Number__c = data.panCard;
        acc.Group__c = data.brandgroup;
        acc.Price_Point__c = data.pricePoint;
        acc.Category__c = data.category;
        acc.Format__c = data.format;
        acc.Brand_Name__c = data.brandName;
```

```
        acc.Payment_Terms__c = data.paymentTerms;
        acc.Tax_Class_CGST__c = data.taxClassCgst;
        acc.Tax_Class_IGST__c = data.taxClassIgst;
        acc.Tax_Class_SGST__c = data.taxClassSgst;
        acc.Tax_Class_UTGST__c = data.taxClassUtgst;
        acc.Title__c = data.title;
        acc.Language__c = data.language;
        acc.Email__c = data.email;
        acc.Phone = data.mobileNumber;
        acc.Acct_Assignment_Group__c = data.acctAssignmentGroup;
        acc.Company_Code__c = data.companyCode;
        acc.RecordTypeId = data.recordTypeId;
        acc.ReConciliation_Account__c = data.reconciliationAccount;

        accountsToProcess.add(acc);
        indexToCustomerCode.put(accountsToProcess.size() - 1, code);
    }

    if (!accountsToProcess.isEmpty()) {
        Database.SaveResult[] results = isInsert
            ? Database.insert(accountsToProcess, false)
            : Database.update(accountsToProcess, false);

        for (Integer i = 0; i < results.size(); i++) {
            Map<String, String> res = new Map<String, String>();
            String code = indexToCustomerCode.get(i);
            res.put('externalId', code);
            res.put('operation', isInsert ? 'Insert' : 'Update');
            if (results[i].isSuccess()) {
                res.put('status', 'Success');
                res.put('message', isInsert ? 'New account created successfully' : 'Account updated successfully');
                res.put('accountId', results[i].getId());
            } else {
                res.put('status', 'Failure');
                res.put('message', results[i].getErrors()[0].getMessage());
                res.put('accountId', '');
            }
            allResponses.add(res);
        }
    }

} catch (Exception e) {
    ExceptionLogger.ExceptionLogPayload logPayload = new ExceptionLogger.ExceptionLogPayload()
        .withComponentName('AccountMasterController')
        .withClassName('AccountMasterController')
        .withMethodName('processAccountData')
        .withException(e)
        .withErrorType('UnexpectedError')
        .withDescription('Error occurred while processing account data. Operation: ' + (isInsert ? 'Insert' : 'Update') +
                ', Total Records: ' + (accountDataList != null ? accountDataList.size() : 0));

    ExceptionLogger.logException(logPayload);
```

```
            allResponses.add(new Map<String, String>{
                'externalId' => '',
                'status' => 'Failure',
                'message' => 'System error occurred: ' + e.getMessage(),
                'accountId' => '',
                'operation' => isInsert ? 'Insert' : 'Update'
            });
        }

        return new Map<String, Object> {
            'status' => 'Processed',
                'message' => 'Account records processed',
                'totalRecordsCount' => accountDataList != null ? accountDataList.size() : 0,
                'results' => allResponses
                };
    }

    public static List<Map<String, String>> getAllResponses() {
        return allResponses;
    }

    private static String getCodeValue(AccountData data) {
        if (!String.isBlank(data.customerCode)) return data.customerCode;
        if (!String.isBlank(data.vendorCode)) return data.vendorCode;
        return null;
    }

    private static String getCodeFieldName(AccountData data) {
        if (!String.isBlank(data.customerCode)) return 'customerCode';
        if (!String.isBlank(data.vendorCode)) return 'vendorCode';
        return 'customerCode/vendorCode';
    }
}
```

## Description :

- Customer api and vendorapi is the apex web service class where we are defining the record id type and sending it to the AccountMastercontroller class where the insertion of the account with the customer or the vendor record type with the validations is getting inserted
- In each of the webservice class i have 3 methods one for insert one for update and another for adding the isInsert true or false and add the record type id in the json hit from the sap or postman and calling the class for further insertion or updation based on the record type sent

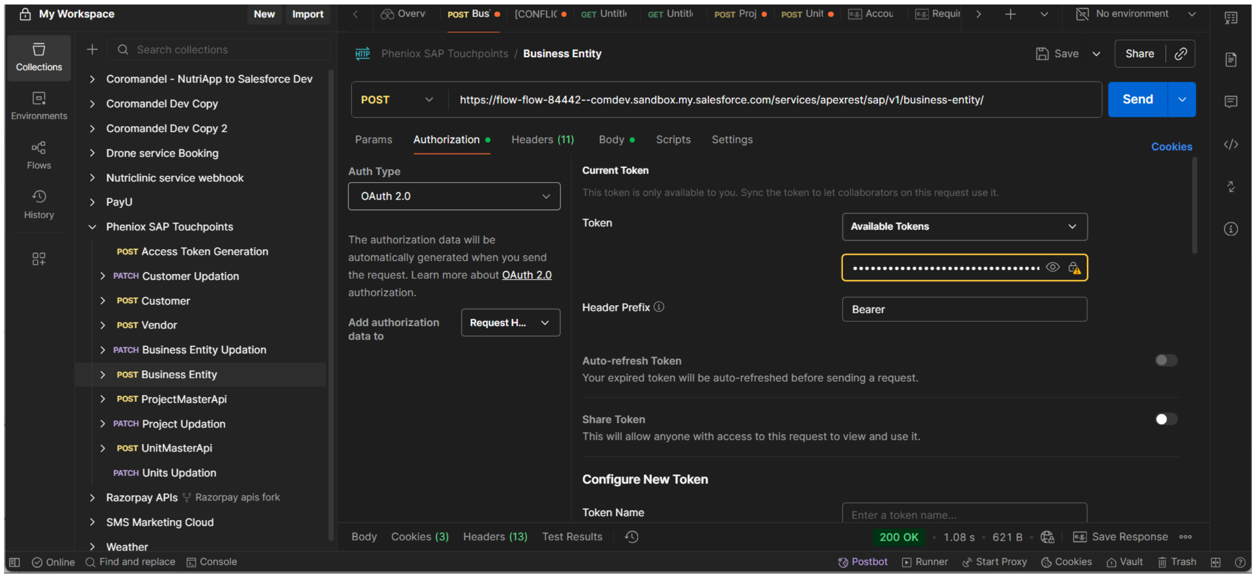
# Business Entity

Apex class :

```
//**********************************************************************************************************
****//
//Name       :  BusinessEntityMasterApi
//Description:  Helper class for all the other webservice classes so that it would be easily
modified
//Created By :  Vijay Kumar
//Date       :  Jul 02 2025
//**********************************************************************************************************
***//
/*Developer             Date            JIRA            Description
* Vijay Kumar           Jul 02 2025     PCRM-14952      REST API for Business Entity
creation/Updation coming from SAP to Salesforce
*/
@RestResource(urlMapping='/sap/v1/business-entity/*')
global without sharing class BusinessEntityMasterApi {

   public class BusinessEntityData {
      public String companyCode;
      public String businessEntity;
      public String nameOfBe;
      public String sectionCode;
      public String businessPlace;
      public String tenancyLaw;

      @SerializedName('currency')
      public String currencyBE;

      public String areaUnit;
      public String volUnit;
      public String unitOfLength;
   }

   public static List<Map<String, Object>> allResponses = new List<Map<String, Object>>();

/*****************************************************************************************
* @Description  :  Getting the data from SAP and Creating/Updating the Business Entity with
some validation and also some required fields set up (For Creation)
* @Param        :  doPost
* @Return       :  void
* Created By    :  Vijay Kumar
* Date          :  03 Jul 2025
****************************************************************************************/
    @HttpPost
    global static void doPost() {
        processBusinessEntityData(true);
    }

/****************************************************************************************
* @Description  :  Getting the data from SAP and Creating/Updating the Business Entity with
some validation and also some required fields set up (For Updation)
* @Param        :  doPatch
* @Return       :  void
* Created By    :  Vijay Kumar
* Date          :  03 Jul 2025
****************************************************************************************/
    @HttpPatch
    global static void doPatch() {
        processBusinessEntityData(false);
    }

    /****************************************************************************************
* @Description  :  Helper class
* @Param        :  processBusinessEntityData
* @Return       :  void
* Created By    :  Vijay Kumar
* Date          :  03 Jul 2025
****************************************************************************************/

    private static void processBusinessEntityData(Boolean isInsert) {
        RestRequest req = RestContext.request;
        RestResponse res = RestContext.response;
        String requestBody = req.requestBody.toString();
        System.debug(requestBody);

        try {
            List<BusinessEntityData> businessEntityDataList = (List<BusinessEntityData>)
JSON.deserialize(requestBody, List<BusinessEntityData>.class);

            allResponses.clear();
            List<Business_Entity__c> entitiesToProcess = new List<Business_Entity__c>();
            List<String> businessEntities = new List<String>();
            Map<String, Business_Entity__c> existingEntitiesMap = new Map<String,
Business_Entity__c>();
```

```
        Map<Integer, String> indexToBusinessEntity = new Map<Integer, String>();

        Boolean hasMissingFields = false;
        Boolean hasOtherErrors = false;

        for (BusinessEntityData entityData : businessEntityDataList) {
            if (!String.isBlank(entityData.businessEntity)) {
                businessEntities.add(entityData.businessEntity);
            }
        }

        if (!businessEntities.isEmpty()) {
            List<Business_Entity__c> existingEntities = [
                SELECT Id, SAP_External_Id__c, Name, Company_Code__c FROM
Business_Entity__c
                WHERE SAP_External_Id__c IN :businessEntities
            ];
            for (Business_Entity__c entity : existingEntities) {
                existingEntitiesMap.put(entity.SAP_External_Id__c, entity);
            }
        }

        for (Integer i = 0; i < businessEntityDataList.size(); i++) {
            BusinessEntityData entityData = businessEntityDataList[i];
            List<String> missingFields = new List<String>();
            Business_Entity__c existingEntity =
existingEntitiesMap.get(entityData.businessEntity);
            System.debug(entityData);

            if (!String.isBlank(entityData.companyCode) &&
!String.isBlank(entityData.businessEntity) &&
                !entityData.companyCode.equals(entityData.businessEntity)) {
                hasMissingFields = true;
                allResponses.add(new Map<String, Object>{
                    'externalId'       => entityData.businessEntity,
                    'status'           => 'Failure',
                    'businessEntityId' => null,
                    'message'          => 'Company Code and Business Entity must be the same'
                    //'operation'        => isInsert ? 'Insert' : 'Update'
                });
                continue;
            }

            if (isInsert && existingEntity != null) {
                hasOtherErrors = true;
                allResponses.add(new Map<String, Object>{
                    'externalId'       => entityData.businessEntity,
                    'status'           => 'Failure',
                    'businessEntityId' => null,
                    'message'          => 'Business Entity already exists for value: ' +
entityData.businessEntity
                    //'operation'        => 'Insert'
                });
                continue;
            }

            if (!isInsert && existingEntity == null) {
                hasOtherErrors = true;
                allResponses.add(new Map<String, Object>{
                    'externalId'       => entityData.businessEntity,
                    'status'           => 'Failure',
                    'businessEntityId' => null,
                    'message'          => 'Business Entity does not exist for value: ' +
entityData.businessEntity + '. Please insert before updating.'
                    //'operation'        => 'Update'
                });
                continue;
            }

            if (existingEntity != null) {
                existingEntity.Name = entityData.nameOfBe;
                existingEntity.Company_Code__c = entityData.companyCode;
                existingEntity.Section_Code__c = entityData.sectionCode;
                //existingEntity.Currency__c = entityData.currencyBE;
                //existingEntity.Area_Unit__c = tryParseDecimal(entityData.areaUnit);
                //existingEntity.Vol_unit__c = tryParseDecimal(entityData.volUnit);
                //existingEntity.Unit_of_Length__c = tryParseDecimal(entityData.unitOfLength);

                entitiesToProcess.add(existingEntity);
                indexToBusinessEntity.put(entitiesToProcess.size() - 1, entityData.businessEntity);
            } else {
                if (String.isBlank(entityData.companyCode)) missingFields.add('companyCode');
                if (String.isBlank(entityData.businessEntity)) missingFields.add('businessEntity');
                if (String.isBlank(entityData.nameOfBe)) missingFields.add('nameOfBe');
                if (String.isBlank(entityData.sectionCode)) missingFields.add('sectionCode');
                if (String.isBlank(entityData.businessPlace)) missingFields.add('businessPlace');
                if (String.isBlank(entityData.tenancyLaw)) missingFields.add('tenancyLaw');
                if (!missingFields.isEmpty()) {
                    hasMissingFields = true;
                    allResponses.add(new Map<String, Object>{
                        'externalId'       => entityData.businessEntity,
                        'status'           => 'Failure',
                        'businessEntityId' => null,
                        'message'          => 'Missing required fields: ' + String.join(missingFields, ', ')
                        //'operation'        => 'Insert'
                    });
                    continue;
                }

                Business_Entity__c newEntity = new Business_Entity__c(
                    Name = entityData.nameOfBe,
                    Company_Code__c = entityData.companyCode,
                    SAP_External_Id__c = entityData.businessEntity,
                    Section_Code__c = entityData.sectionCode,
                    Tenancy_Law__c = entityData.tenancyLaw
                    //Currency__c = entityData.currencyBE,
                    //Area_Unit__c = tryParseDecimal(entityData.areaUnit),
                    //Vol_unit__c = tryParseDecimal(entityData.volUnit),
                    //Unit_of_Length__c = tryParseDecimal(entityData.unitOfLength)
                );

                entitiesToProcess.add(newEntity);
                indexToBusinessEntity.put(entitiesToProcess.size() - 1, entityData.businessEntity);
            }
        }

        List<Map<String, Object>> finalResponse;

        if (!entitiesToProcess.isEmpty()) {
            List<Database.SaveResult> results;
            String operation = isInsert ? 'Insert' : 'Update';

            try {
                if (isInsert) {
                    results = Database.insert(entitiesToProcess, false);
                } else {
                    for (Business_Entity__c entity : entitiesToProcess) {
                        if (entity.Id == null) {
                            throw new AuraHandledException('Update operation failed: One or more
records do not have Ids.');
                        }
                    }
                    results = Database.update(entitiesToProcess, false);
                }

                for (Integer i = 0; i < results.size(); i++) {
                    String businessEntity = indexToBusinessEntity.get(i);
                    Map<String, Object> responseMap = new Map<String, Object>();
                    responseMap.put('externalId', businessEntity);

                    if (results[i].isSuccess()) {
                        responseMap.put('status', 'Success');
                        responseMap.put('businessEntityId', results[i].getId());
                        responseMap.put('message', isInsert
                            ? 'New business entity created successfully'
                            : 'Existing business entity updated successfully');
                    } else {
                        hasOtherErrors = true;
                        responseMap.put('status', 'Failure');
                        responseMap.put('businessEntityId', null);
                        responseMap.put('message', results[i].getErrors()[0].getMessage());
                    }

                    allResponses.add(responseMap);
                }

                finalResponse = allResponses;

            } catch (Exception e) {
                hasOtherErrors = true;
                finalResponse = new List<Map<String, Object>>{
                    new Map<String, Object>{
                        'externalId'       => null,
                        'status'           => 'Failure',
                        'businessEntityId' => null,
                        //'operation'        => isInsert ? 'Insert' : 'Update',
                        'message'          => 'Insert/Update failed: ' + e.getMessage()
                    }
                };
                    ExceptionLogger.ExceptionLogPayload logPayload = new
ExceptionLogger.ExceptionLogPayload()
                .withComponentName('BusinessEntityMasterApi')
                .withClassName('BusinessEntityMasterApi')
                .withMethodName('processBusinessEntityData')
                .withException(e)
                .withErrorType('RestApiError')
                .withRequestBody(requestBody)
                .withEndpoint('/sap/v1/business-entity/')
                .withIntegrationType('REST_API')
                .withDomain('SAP')
                .withDescription('Error occurred in BusinessEntityMasterApi while processing SAP
data. Operation: ' + (isInsert ? 'Create' : 'Update') +
                                ', Records Count: ' + (businessEntityDataList != null ?
String.valueOf(businessEntityDataList.size()) : 'Unknown') +
                                ', Request Body Length: ' + (String.isNotBlank(requestBody) ?
String.valueOf(requestBody.length()) : '0'));

            ExceptionLogger.logException(logPayload);
            }

        } else {
            if (allResponses.isEmpty()) {
                hasOtherErrors = true;
                finalResponse = new List<Map<String, Object>>{
                    new Map<String, Object>{
                        'externalId'       => null,
                        'status'           => 'Failure',
                        'businessEntityId' => null,
                        'message'          => 'No records processed'
                        //'operation'        => isInsert ? 'Insert' : 'Update'
                    }
                };
            } else {
                finalResponse = allResponses;
            }
        }

        res.addHeader('Content-Type', 'application/json');
        res.responseBody = Blob.valueOf(JSON.serialize(finalResponse));

        Boolean hasSuccess = false;
        for (Map<String, Object> resp : allResponses) {
            if ((String)resp.get('status') == 'Success') {
                hasSuccess = true;
                break;
            }
        }

        if (hasMissingFields && !hasOtherErrors && !hasSuccess) {
            res.statusCode = 400;
        } else if (!hasSuccess) {
            res.statusCode = 401;
        } else {
            res.statusCode = 200;
        }

    } catch (Exception e) {
        List<Map<String, Object>> errorResponse = new List<Map<String, Object>>{
            new Map<String, Object>{
                'externalId'       => null,
                'status'           => 'Failure',
                'businessEntityId' => null,
                'message'          => 'Unexpected error: ' + e.getMessage()
                //'operation'        => 'Unknown'
            }
        };

        res.addHeader('Content-Type', 'application/json');
        res.responseBody = Blob.valueOf(JSON.serialize(errorResponse));
        res.statusCode = 401;

        ExceptionLogger.ExceptionLogPayload logPayload = new
ExceptionLogger.ExceptionLogPayload()
        .withComponentName('BusinessEntityMasterApi')
        .withClassName('BusinessEntityMasterApi')
        .withMethodName('processBusinessEntityData')
        .withDescription('Unexpected error: ' + e.getMessage())
        .withException(e)
        .withErrorType('RestApiError')
        .withRequestBody(requestBody)
        .withEndpoint('/sap/v1/business-entity/')
        .withIntegrationType('REST_API')
        .withDomain('SAP');

    ExceptionLogger.logException(logPayload);
    }
}

private static Decimal tryParseDecimal(String value) {
    try {
        return String.isNotBlank(value) ? Decimal.valueOf(value) : null;
    } catch (Exception e) {
        return null;
    }
}

public static List<Map<String, Object>> getAllResponses() {
    return allResponses;
}
}
```

Postman Posts :



My Workspace  New  Import

Coromandel - NutriApp to Salesforce Dev
Coromandel Dev Copy
Coromandel Dev Copy 2
Drone service Booking
Nutriclinic service webhook
PayU
Pheniox SAP Touchpoints
POST Access Token Generation
PATCH Customer Updation
POST Customer
POST Vendor
PATCH Business Entity Updation
POST Business Entity
POST ProjectMasterApi
PATCH Project Updation
POST UnitMasterApi
PATCH Units Updation
Razorpay APIs  Razorpay apis fork
SMS Marketing Cloud
Weather

Pheniox SAP Touchpoints / Business Entity

POST https://flow-flow-84442--comdev.sandbox.my.salesforce.com/services/apexrest/sap/v1/business-entity/  Send

Params  Authorization  Headers (11)  Body  Scripts  Settings

Auth Type: OAuth 2.0

The authorization data will be automatically generated when you send the request. Learn more about OAuth 2.0 authorization.

Add authorization data to: Request H...

Current Token
This token is only available to you. Sync the token to let collaborators on this request use it.

Token: Available Tokens
••••••••••••••••••••••••••••••

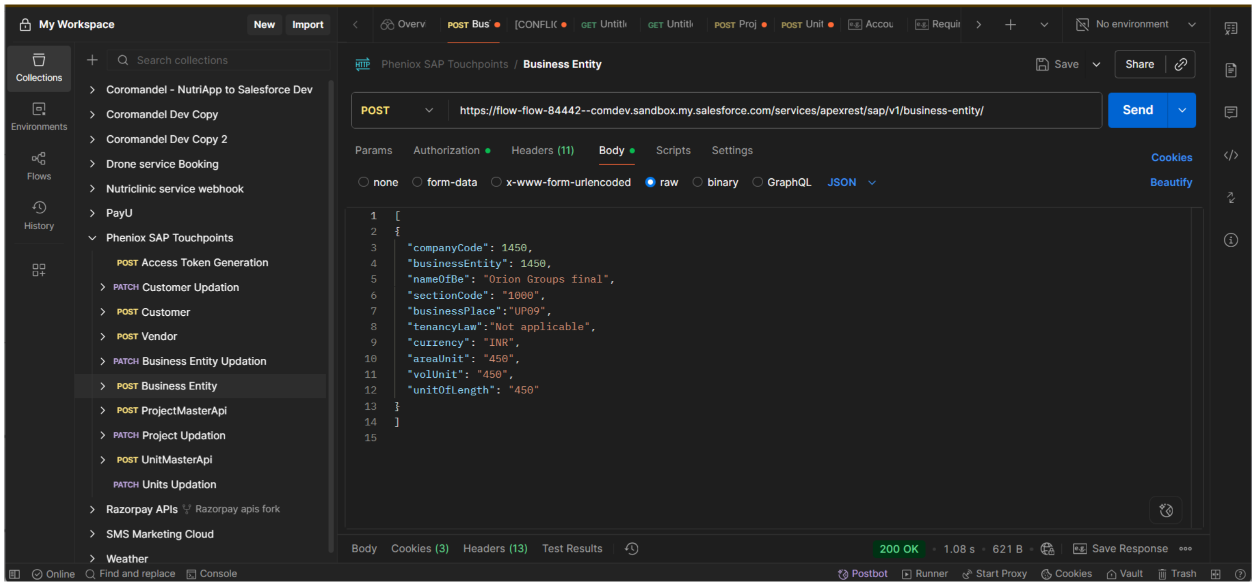
Header Prefix: Bearer

Auto-refresh Token
Your expired token will be auto-refreshed before sending a request.

Share Token
This will allow anyone with access to this request to view and use it.

Configure New Token
Token Name: Enter a token name...

Body  Cookies (3)  Headers (13)  Test Results  200 OK  1.08 s  621 B  Save Response



Pheniox SAP Touchpoints / Business Entity

POST https://flow-flow-84442--comdev.sandbox.my.salesforce.com/services/apexrest/sap/v1/business-entity/  Send

Params  Authorization  Headers (11)  Body  Scripts  Settings

none  form-data  x-www-form-urlencoded  [✓ raw]  binary  GraphQL  JSON

```
[
{
  "companyCode": 1450,
  "businessEntity": 1450,
  "nameOfBe": "Orion Groups final",
  "sectionCode": "1000",
  "businessPlace":"UP09",
  "tenancyLaw":"Not applicable",
  "currency": "INR",
  "areaUnit": "450",
  "volUnit": "450",
  "unitOfLength": "450"
}
]
```

Body  Cookies (3)  Headers (13)  Test Results  200 OK  1.08 s  621 B  Save Response

## Endpoint :

- POST
  https://flow-flow-84442--comdev.sandbox.my.salesforce.com/services/apexrest/sap/v1/business-entity/
- PATCH
  https://flow-flow-84442--comdev.sandbox.my.salesforce.com/services/apexrest/sap/v1/business-entity/

Mandatory Fields :

- companyCode:
- businessEntity:
- nameOfBe:
- sectionCode:
- businessPlace:
- tenancyLaw:

Request Body :

Single Business Entity :

```
[
  {
    "companyCode": 1691,
    "businessEntity": 1691,
    "nameOfBe": "Island Star chennai 123",
    "sectionCode": "1000",
    "tenancyLaw":"Not applicable",
    "currency": "INR",
    "areaUnit": "450",
    "volUnit": "450",
    "unitOfLength": "450"
  }
]
```

Response :

Business Entity and Company Code must be same Validation :

```
[
    {
        "message": "Company Code and Business Entity must be the same",
        "businessEntityId": null,
        "status": "Failure",
        "externalId": "1662"
    }
]
```

Business Entity Created :

```
[
    {
        "message": "New business entity created successfully",
        "businessEntityId": "a20F7000002PHpEIAW",
        "status": "Success",
        "externalId": "1661"
    }
]
```

Business Entity Updated :

```
[
    {
        "message": "Existing business entity updated successfully",
        "businessEntityId": "a20F7000002PHpEIAW",
        "status": "Success",
        "externalId": "1661"
    }
]
```

Missing Required Fields :

```
[
    {
        "message": "Missing required fields: nameOfBe",
        "businessEntityId": null,
        "status": "Failure",
        "externalId": "1663"
    }
]
```

Multiple Business Entity :

Request Body :

```
[
  {
    "companyCode": 1600,
    "businessEntity": 1600,
    "nameOfBe": "Island Star chennai 123",
    "sectionCode": "1000",
    "businessPlace":"MH27",
    "tenancyLaw":"Not applicable",
    "currency": "INR",
    "areaUnit": "450",
    "volUnit": "450",
    "unitOfLength": "450"
  },
  {
    "companyCode": 1691,
    "businessEntity": 1691,
    "nameOfBe": "Island Star chennai 123",
    "sectionCode": "1000",
    "businessPlace":"MH27",
    "tenancyLaw":"Not applicable",
    "currency": "INR",
    "areaUnit": "450",
    "volUnit": "450",
    "unitOfLength": "450"
  }
]
```

Response :

```
[
    {
        "message": "New business entity created successfully",
        "businessEntityId": "a20F7000002PHpTIAW",
        "status": "Success",
        "externalId": "1600"
    },
    {
        "message": "Existing business entity updated successfully",
        "businessEntityId": "a20F7000002PHpOIAW",
        "status": "Success",
        "externalId": "1691"
    }
]
```

Multiple List Failure for Required Fields Missing :

```
[
  {
    "businessEntity": 1601,
    "nameOfBe": "Island Star chennai 123",
    "sectionCode": "1000",
    "businessPlace":"MH27",
    "tenancyLaw":"Not applicable",
    "currency": "INR",
    "areaUnit": "450",
    "volUnit": "450",
    "unitOfLength": "450"
  },
  {
    "companyCode": 1691,
    "businessEntity": 1691,
    "nameOfBe": "Island Star chennai 123",
    "sectionCode": "1000",
    "businessPlace":"MH27",
    "tenancyLaw":"Not applicable",
    "currency": "INR",
    "areaUnit": "450",
    "volUnit": "450",
    "unitOfLength": "450"
  }
]
```

Response :

```
[
    {
        "message": "Missing required fields: companyCode",
        "businessEntityId": null,
        "status": "Failure",
        "externalId": "1601"
    },
    {
        "message": "Existing business entity updated successfully",
        "businessEntityId": "a20F7000002PHpOIAW",
        "status": "Success",
        "externalId": "1691"
    }
]
```
:

## Project :

Apex class :

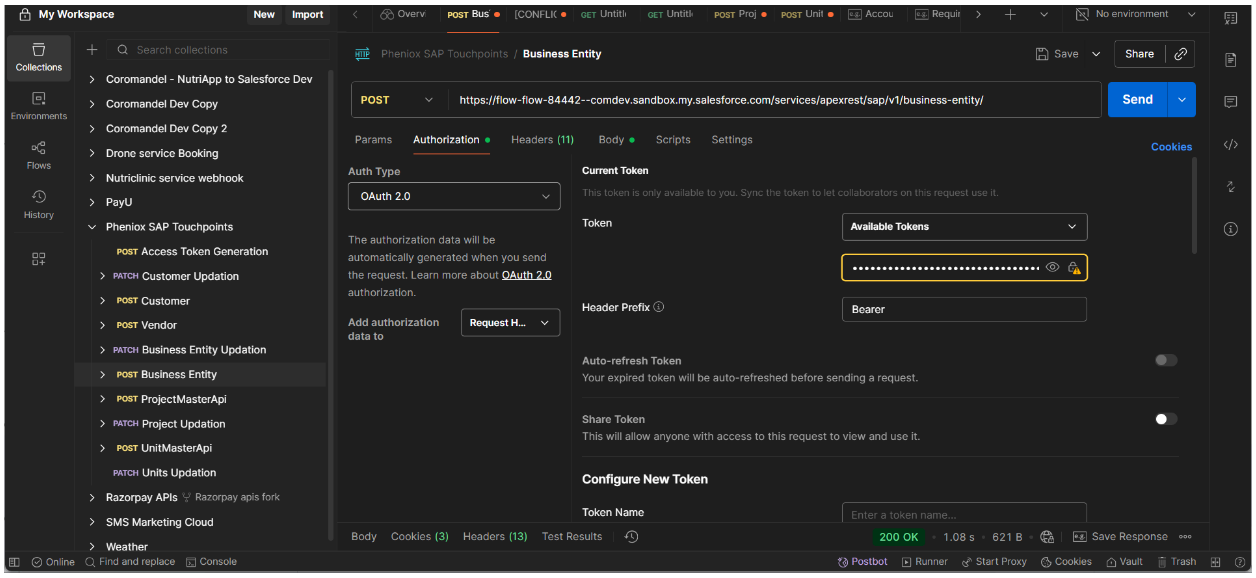
```
//*********************************************************************************************************
****//
//Name       :  ProjectMasterApi
//Description:  Helper class for all the other webservice classes so that it would be easily
modified
//Created By :  Vijay Kumar
//Date       :  Jul 02 2025
//*********************************************************************************************************
***//
/*Developer              Date               JIRA               Description
* Vijay Kumar           Jul 02 2025         PCRM-14952          REST API for Project/Building
creation/Updation coming from SAP to Salesforce
*/
@RestResource(urlMapping='/sap/v1/building/*')
global without sharing class ProjectMasterApi {

    public class PropertyData {
        public String companyCode;
        public String businessEntity;
        public String building;
        public String nameOfBuilding;
        public String function;
        public String compactDisplayOfAddress;
        public String nameOfCountryRegionShort;
        public String countryRegionName;
        public String description;
        public String businessPlace;
        public String sectionCode;
        public String profitCenter;
        public Date validFrom;
        public Date to;
    }
    public static List<ProjectResponseWrapper> allResponses = new
List<ProjectResponseWrapper>();

/*************************************************************************************
* @Description  :  Getting the data from SAP and Creating/Updating the Property/Building with
some validation and also some required fields set up (For Creation)
* @Param        :  doPost
* @Return       :  void
* Created By    :  Vijay Kumar
* Date          :  03 Jul 2025
*************************************************************************************/
    @HttpPost
    global static void doPost() {
        processPropertyData(true);
    }


    /*************************************************************************************
* @Description  :  Getting the data from SAP and Creating/Updating the Property/Building with
some validation and also some required fields set up (For Updation)
* @Param        :  doPatch
* @Return       :  void
* Created By    :  Vijay Kumar
* Date          :  03 Jul 2025
*************************************************************************************/
    @HttpPatch
    global static void doPatch() {
        processPropertyData(false);
    }


/*************************************************************************************
* @Description  :  Helper method
* @Param        :  processPropertyData
* @Return       :  void
* Created By    :  Vijay Kumar
* Date          :  03 Jul 2025
*************************************************************************************/
    private static void processPropertyData(Boolean isInsert){
        RestRequest req = RestContext.request;
        RestResponse res = RestContext.response;
        String requestBody = req.requestBody.toString();
        System.debug(requestBody);
        try {
            List<PropertyData> propertyDataList = (List<PropertyData>)
    JSON.deserialize(requestBody, List<PropertyData>.class);

            allResponses.clear();
            List<Property__c> propertiesToProcess = new List<Property__c>();
            List<String> buildingList = new List<String>();
            Map<String, Property__c> existingPropertiesMap = new Map<String, Property__c>();
            Map<Integer, String> indexToBuilding = new Map<Integer, String>();

            Boolean hasMissingFields = false;
            Boolean hasValidationErrors = false;
            Boolean hasOtherErrors = false;

            for (PropertyData propertyData : propertyDataList) {
                if (!String.isBlank(propertyData.building)) {
                    buildingList.add(propertyData.building);
                }
            }

            if (!buildingList.isEmpty()) {
                List<Property__c> existingProperties = [
                    SELECT Id, SAP_External_Id__c, Name
                    FROM Property__c
                    WHERE SAP_External_Id__c IN :buildingList
                ];
                for (Property__c property : existingProperties) {
                    existingPropertiesMap.put(property.SAP_External_Id__c, property);
                }
            }

            Map<String, Business_Entity__c> businessEntityMap = new Map<String,
    Business_Entity__c>();
            for (PropertyData propertyData : propertyDataList) {
                if (!String.isBlank(propertyData.businessEntity)) {
                    businessEntityMap.put(propertyData.businessEntity, null);
                }
            }

            if (!businessEntityMap.isEmpty()) {
                List<Business_Entity__c> businessEntities = [
                    SELECT Id, SAP_External_Id__c, Company_Code__c
                    FROM Business_Entity__c
                    WHERE SAP_External_Id__c IN :businessEntityMap.keySet()
                ];
                for (Business_Entity__c be : businessEntities) {
                    businessEntityMap.put(be.SAP_External_Id__c, be);
                }
            }

            for (Integer i = 0; i < propertyDataList.size(); i++) {
                PropertyData propertyData = propertyDataList[i];
                List<String> missingFields = new List<String>();
                Property__c existingProperty = existingPropertiesMap.get(propertyData.building);

                Business_Entity__c associatedBusinessEntity =
    businessEntityMap.get(propertyData.businessEntity);

                if (associatedBusinessEntity == null) {
                    hasValidationErrors = true;
                    allResponses.add(new ProjectResponseWrapper(
                        propertyData.building,
                        'Failure',
                        null,
                        'Business entity with code ' + propertyData.businessEntity + ' is not available'
                    ));

                    continue;
                }

                if (associatedBusinessEntity.Company_Code__c != propertyData.companyCode) {
                    hasValidationErrors = true;
                    allResponses.add(new ProjectResponseWrapper(
                        propertyData.building,
                        'Failure',
                        null,
                        'The company code and business entity values must be the same for the
    business entity ' + propertyData.businessEntity
                    ));
                    continue;
                }

                List<Business_Place_Master_Data__mdt> customMetadataList = [
                    SELECT MasterLabel, Company_Code__c, State__c
                    FROM Business_Place_Master_Data__mdt
                    WHERE MasterLabel = :propertyData.businessPlace
                    AND Company_Code__c = :propertyData.companyCode
                ];
                if (customMetadataList.isEmpty()) {
                    hasValidationErrors = true;
                    allResponses.add(new ProjectResponseWrapper(
                        propertyData.building,
                        'Failure',
                        null,
                        'For this company code and business place there is no state available.'
                    ));
                    res.statusCode = 400;
                    break;
                }

                Business_Place_Master_Data__mdt customMetadata = customMetadataList[0];
                String stateValue = customMetadata.State__c;

                if (isInsert && existingProperty != null) {
                    hasOtherErrors = true;
                    allResponses.add(new ProjectResponseWrapper(
                        propertyData.building,
                        'Failure',
                        null,
                        'Property with building code ' + propertyData.building + ' already exists.'
                    ));

                    continue;
                }

                if (!isInsert && existingProperty == null) {
                    hasOtherErrors = true;
                    allResponses.add(new ProjectResponseWrapper(
                        propertyData.building,
                        'Failure',
                        null,
                        'Property with building code ' + propertyData.building + ' does not exist. Insert it
    before updating.'
                    ));
                    continue;
                }

                if (existingProperty != null) {
                    existingProperty.State__c = stateValue;
                    existingProperty.Business_Entity_Code__c = propertyData.businessEntity;
                    existingProperty.Valid_From__c = propertyData.validFrom;
```

```
        existingProperty.Valid_To__c = propertyData.to;
        existingProperty.Company_Code__c = propertyData.companyCode;
        existingProperty.Name = propertyData.nameOfBuilding;
        existingProperty.SAP_External_Id__c = propertyData.building;
        existingProperty.Function__c = propertyData.function;
        existingProperty.Address__c = propertyData.compactDisplayOfAddress;
        existingProperty.Country_Code__c = propertyData.nameOfCountryRegionShort;
        existingProperty.Country_Region_Name__c = propertyData.countryRegionName;
        existingProperty.Description__c = propertyData.description;
        existingProperty.Business_Place__c = propertyData.businessPlace;
        existingProperty.Section_Code__c = propertyData.sectionCode;
        existingProperty.Profit_Center__c = propertyData.profitCenter;
        existingProperty.Business_Entity__c = associatedBusinessEntity.Id;

        propertiesToProcess.add(existingProperty);
        indexToBuilding.put(propertiesToProcess.size() - 1, propertyData.building);
    } else {


        Property__c newProperty = new Property__c(
            Business_Entity_Code__c = propertyData.businessEntity,
            Name = propertyData.nameOfBuilding,
            Valid_From__c = propertyData.validFrom,
            Valid_To__c = propertyData.to,
            Company_Code__c = propertyData.companyCode,
            SAP_External_Id__c = propertyData.building,
            Function__c = propertyData.function,
            Address__c = propertyData.compactDisplayOfAddress,
            Country_Code__c = propertyData.nameOfCountryRegionShort,
            Country_Region_Name__c = propertyData.countryRegionName,
            Description__c = propertyData.description,
            Business_Place__c = propertyData.businessPlace,
            Section_Code__c = propertyData.sectionCode,
            Profit_Center__c = propertyData.profitCenter,
            State__c = stateValue,
            Business_Entity__c = associatedBusinessEntity.Id
        );

        propertiesToProcess.add(newProperty);
        indexToBuilding.put(propertiesToProcess.size() - 1, propertyData.building);
    }
}

List<ProjectResponseWrapper> finalResponse;
```

```
if (!propertiesToProcess.isEmpty()) {
    List<Database.SaveResult> results;
    try {
        if (isInsert) {
            results = Database.insert(propertiesToProcess, false);
        } else {
            for (Property__c p : propertiesToProcess) {
                if (p.Id == null) {
                    throw new AuraHandledException('Cannot update record without Id.');
                }
            }
            results = Database.update(propertiesToProcess, false);
        }

        for (Integer i = 0; i < results.size(); i++) {
            String building = indexToBuilding.get(i);

            ProjectResponseWrapper responseWrapper = new ProjectResponseWrapper();
            responseWrapper.externalId = building;

            if (results[i].isSuccess()) {
                responseWrapper.status = 'Success';
                responseWrapper.propertyId = results[i].getId();
                responseWrapper.message = isInsert
                    ? 'New property created successfully'
                    : 'Existing property updated successfully';
            } else {
                hasOtherErrors = true;
                responseWrapper.status = 'Failure';
                responseWrapper.propertyId = null;
                responseWrapper.message = results[i].getErrors()[0].getMessage();
            }

            allResponses.add(responseWrapper);
        }

        finalResponse = allResponses;

    } catch (Exception e) {
        hasOtherErrors = true;

        finalResponse= new List<ProjectResponseWrapper>{
            new ProjectResponseWrapper(
                    null,
                    'Failure',
                    null,
                    'Insert/Update failed: ' + e.getMessage()
                )
            };


        }

    } else {
        finalResponse= allResponses.isEmpty() ? new List<ProjectResponseWrapper>{
                new ProjectResponseWrapper(
                    null,
                    'Failure',
                    null,
                    'No records processed'
                )
            } : allResponses;


    }

    res.addHeader('Content-Type', 'application/json');
    res.responseBody = Blob.valueOf(JSON.serialize(finalResponse));

    Boolean hasSuccess = false;
    for (ProjectResponseWrapper r : allResponses) {
        if (r.status == 'Success') {
            hasSuccess = true;
            break;
        }
    }

    if ((hasMissingFields || hasValidationErrors) && !hasSuccess) {
        res.statusCode = 400;
    } else if (!hasSuccess) {
        res.statusCode = 401;
    } else {
        res.statusCode = 200;
    }

} catch (Exception e) {
    List<Map<String, Object>> errorResponse = new List<Map<String, Object>>{
```

```
                new Map<String, Object> {
                    'externalId' => null,
                    'status' => 'Failure',
                    'propertyId' => null,
                    'message' => 'Unexpected error: ' + e.getMessage()
                }
            };
            res.addHeader('Content-Type', 'application/json');
            res.responseBody = Blob.valueOf(JSON.serialize(errorResponse));
            res.statusCode = 401;

            ExceptionLogger.ExceptionLogPayload logPayload = new
ExceptionLogger.ExceptionLogPayload()
                .withComponentName('ProjectMasterApi')
                .withClassName('ProjectMasterApi')
                .withMethodName('processPropertyData')
                .withDescription('Unexpected error: ' + e.getMessage())
                .withException(e)
                .withErrorType('RestApiError')
                .withRequestBody(requestBody)
                .withEndpoint('/sap/v1/building/')
                .withIntegrationType('REST_API')
                .withDomain('SAP');

            ExceptionLogger.logException(logPayload);
        }
    }

    public class ProjectResponseWrapper{
        public String externalId;
        public String status;
        public String propertyId;
        public String message;

        public ProjectResponseWrapper(String externalId, String status, String propertyId, String
message){
            this.externalId = externalId;
            this.status = status;
            this.propertyId = propertyId;
            this.message = message;
        }

        public ProjectResponseWrapper(){}
    }
}
```

Postman posts :



Phenixox SAP Touchpoints / Business Entity

POST https://flow-flow-84442--comdev.sandbox.my.salesforce.com/services/apexrest/sap/v1/business-entity/

Params | Authorization ● | Headers (11) | Body ● | Scripts | Settings

Auth Type: OAuth 2.0

The authorization data will be automatically generated when you send the request. Learn more about OAuth 2.0 authorization.

Add authorization data to: Request H...

Current Token

This token is only available to you. Sync the token to let collaborators on this request use it.

Token: Available Tokens

Header Prefix: Bearer

Auto-refresh Token

Your expired token will be auto-refreshed before sending a request.

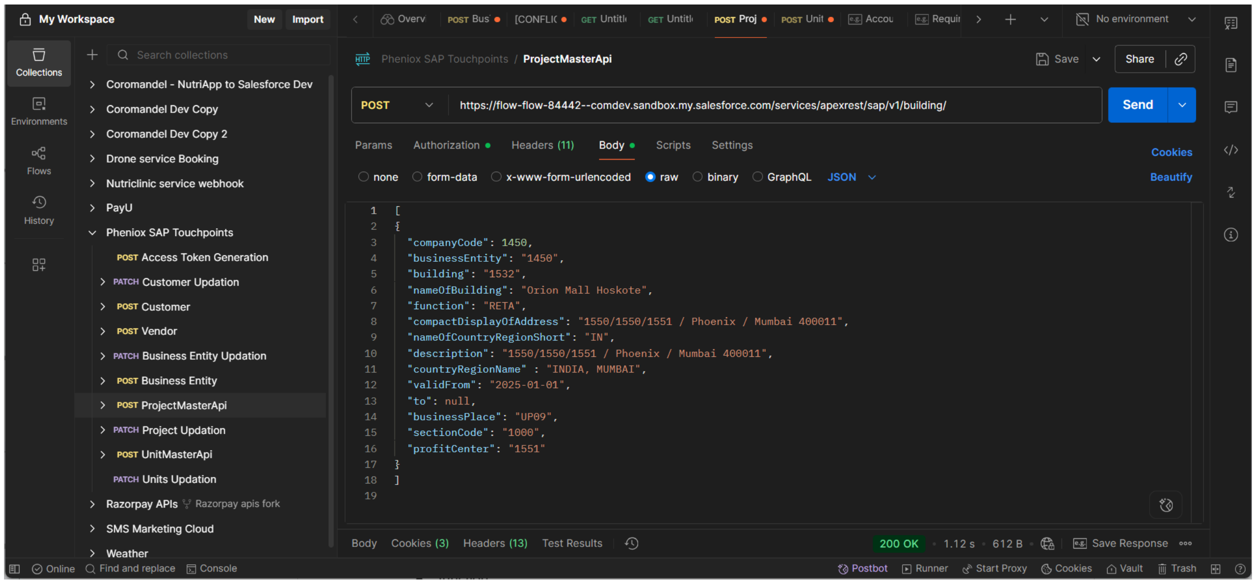
Share Token

This will allow anyone with access to this request to view and use it.

Configure New Token

Token Name: Enter a token name...

Body | Cookies (3) | Headers (13) | Test Results — 200 OK · 1.08 s · 621 B



Phenixox SAP Touchpoints / ProjectMasterApi

POST https://flow-flow-84442--comdev.sandbox.my.salesforce.com/services/apexrest/sap/v1/building/

Params | Authorization ● | Headers (11) | Body ● | Scripts | Settings

none | form-data | x-www-form-urlencoded | [✓ raw] | binary | GraphQL | JSON

```json
[
{
  "companyCode": 1450,
  "businessEntity": "1450",
  "building": "1532",
  "nameOfBuilding": "Orion Mall Hoskote",
  "function": "RETA",
  "compactDisplayOfAddress": "1550/1550/1551 / Phoenix / Mumbai 400011",
  "nameOfCountryRegionShort": "IN",
  "description": "1550/1550/1551 / Phoenix / Mumbai 400011",
  "countryRegionName" : "INDIA, MUMBAI",
  "validFrom": "2025-01-01",
  "to": null,
  "businessPlace": "UP09",
  "sectionCode": "1000",
  "profitCenter": "1551"
}
]
```

Body | Cookies (3) | Headers (13) | Test Results — 200 OK · 1.12 s · 612 B

Mandatory Fields :

- companyCode
- businessEntity
- building
- nameOfBuilding
- function
- compactDisplayOfAddress

- nameOfCountryRegionShort
- countryRegionName
- description
- businessPlace
- sectionCode
- profitCenter

Request Body :

```
[
  {
    "companyCode": 1691,
    "businessEntity": "1691",
    "building": "1011",
    "nameOfBuilding": "Phoenix Marketcity Chennai 231",
    "function": "RETA",
    "compactDisplayOfAddress": "1550/1550/1551 / Phoenix / Mumbai 400011",
    "nameOfCountryRegionShort": "IN",
    "countryRegionName": "INDIA, MUMBAI",
    "description": "1550/1550/1551 / Phoenix / Mumbai 400011",
    "validFrom": "2025-01-01",
    "to": null,
    "businessPlace": "MH27",
    "sectionCode": "1000",
    "profitCenter": "1551"
  }
]
```

Responses :

Business Entity Not Available

```
[
    {
        "message": "Business entity with code 1669 is not available",
        "propertyId": null,
        "status": "Failure",
        "externalId": "1555"
    }
]
```

Property/Building Inserted Successfully

```
[
    {
        "message": "New property created successfully",
        "propertyId": "a05F700000CyKaBIAV",
        "status": "Success",
        "externalId": "1800"
    }
]
```

Property/Building Updated Successfully

```
[
    {
        "message": "Existing property updated successfully",
        "propertyId": "a05F700000CyKaBIAV",
        "status": "Success",
        "externalId": "1800"
    }
]
```

Missing Required Fields Validation Error

```
[
    {
        "message": "Missing required fields: nameOfBuilding",
        "propertyId": null,
        "status": "Failure",
        "externalId": "1805"
    }
]
```

Company Code And Business Entity Must Remain the same

```
[
    {
        "message": "Company Code and Business Entity must be the same",
        "businessEntityId": null,
        "status": "Failure",
        "externalId": "1662"
    }
]
```

For the given Business Place and Customer Code there is no related State

```
[
    {
        "message": "For this company code and business place there is no state 
available.",
        "propertyId": null,
        "status": "Failure",
        "externalId": "1011"
    }
]
```

## Rental Object - Unit, Floor, Tower

Apex class :

```
/*Developer           Date           JIRA           Description
* Vijay Kumar         Jul 02 2025    PCRM-14952     REST API for Unit Related Floor, 
Related Tower/Building If not present creation/Updation coming from SAP to Salesforce
*/
@RestResource(urlMapping='/sap/v1/rental-unit/*')
global without sharing class UnitMasterApi {

    public class UnitData {
        public String companyCode;
        public String businessEntity;
        public String rentalObject;
        public String rentalObjectType;
        public String usageType;
        public String rentalObjectName;
        public String building;
        public String ruNoOld;
        public String toFloor;
        public String unitCurrency;
        public String profitCenter;
        public List<MeasurementTypeData> measurementTypes;
    }

    public class MeasurementTypeData {
        public String type;
        public String validfrom;
        public String validTo;
        public Decimal measurementAmount;
        public String measurementUnit;
    }

    public static List<UnitResponseWrapper> allResponses = new
List<UnitResponseWrapper>();

    /**********************************************************************************
    * @Description  : Getting the data from SAP and Creating/Updating the Unit related Floor,
    *               Related Tower with some validation and also some required fields set up(For
Creation)
    * @Param        : doPost
    * @Return       : void
    * Created By    : Vijay Kumar
    * Date          : 03 Jul 2025
    **********************************************************************************/
    @HttpPost
    global static void doPost() {
        processUnitData(true);
    }

    /**********************************************************************************
    * @Description  : Getting the data from SAP and Creating/Updating the Unit related Floor,
    *               Related Tower with some validation and also some required fields set up(For
Updation)
    * @Param        : doPatch
    * @Return       : void
    * Created By    : Vijay Kumar
    * Date          : 03 Jul 2025
    **********************************************************************************/
    @HttpPatch
    global static void doPatch() {
        processUnitData(false);
    }

    /**********************************************************************************
    * @Description  : Validate Measurement Types
    * @Param        : measurementTypes, rentalObject
    * @Return       : String (error message if validation fails, null if valid)
    * Created By    : Vijay Kumar
    * Date          : 03 Jul 2025
    **********************************************************************************/
    public static String validateMeasurementTypes(List<MeasurementTypeData>
measurementTypes, String rentalObject, Boolean isInsert) {
        if (measurementTypes == null || measurementTypes.isEmpty()) {
            return 'Measurement types are required';
        }

        Set<String> seenMeasurementTypes = new Set<String>();
        Map<String, Integer> measurementTypeCount = new Map<String, Integer>();

        for (MeasurementTypeData mt : measurementTypes) {
            if (measurementTypeCount.containsKey(mt.type)) {
                measurementTypeCount.put(mt.type, measurementTypeCount.get(mt.type) + 1);
            } else {
                measurementTypeCount.put(mt.type, 1);
            }
        }

        List<String> duplicateMeasurementTypes = new List<String>();
        for (String type : measurementTypeCount.keySet()) {
            if (measurementTypeCount.get(type) > 1) {
                duplicateMeasurementTypes.add(type);
            }
        }

        if (!duplicateMeasurementTypes.isEmpty()) {
            return 'The following measurement types are duplicated: ' +
String.join(duplicateMeasurementTypes, ', ');
        }

        if (isInsert) {
            Boolean hasZ003 = false;
            Boolean hasZ005 = false;

            for (MeasurementTypeData mt : measurementTypes) {
                if (mt.type == 'Z003') hasZ003 = true;
                if (mt.type == 'Z005') hasZ005 = true;
            }

            if (!hasZ003 || !hasZ005) {
                String missingArea = '';
                if (!hasZ003) missingArea += 'Z003';
                if (!hasZ003 && !hasZ005) missingArea += ' and ';
                if (!hasZ005) missingArea += 'Z005';
                return missingArea + ' measurement types are required';
            }
        }

        for (MeasurementTypeData mt : measurementTypes) {
            List<String> missingFields = new List<String>();

            if (String.isBlank(mt.type)) missingFields.add('type');
            if (String.isBlank(mt.validfrom)) missingFields.add('validfrom');
            if (String.isBlank(mt.validTo)) missingFields.add('validTo');
            if (mt.measurementAmount == null) missingFields.add('measurementAmount');
            if (String.isBlank(mt.measurementUnit)) missingFields.add('measurementUnit');

            if (!missingFields.isEmpty()) {
                return 'Missing required fields in measurement types: ' + String.join(missingFields, ', ');
            }

            if (mt.measurementUnit != 'ft2') {
                return 'Invalid measurement unit: ' + mt.measurementUnit;
            }
        }

        return null;
    }

    /***************************************************************************************
    * @Description  : Validate unit data and collect related codes (rental objects, business
entities, buildings, floors)
    * @Param        : unitDataList, isInsert
    * @Return       : Map<String, Object> containing sets of codes and maps for further
processing
    * Created By    : Vijay Kumar
    * Date          : 05 Aug 2025
    ***************************************************************************************/
    private static Map<String, Object> validateAndCollectUnitData(List<UnitData> unitDataList,
Boolean isInsert) {
        Set<String> rentalObjectSet = new Set<String>();
        Set<String> businessEntityCodes = new Set<String>();
        Set<String> buildingCodes = new Set<String>();
        Set<String> floorCodes = new Set<String>();
        Map<String, List<MeasurementTypeData>> unitToMeasurementTypes = new Map<String,
List<MeasurementTypeData>>();
        Map<String, UnitData> rentalObjectToUnitData = new Map<String, UnitData>();
        Boolean hasValidationErrors = false;
        for (UnitData unitData : unitDataList) {
            String measurementValidationError =
validateMeasurementTypes(unitData.measurementTypes, unitData.rentalObject, isInsert);
            if (measurementValidationError != null) {
                hasValidationErrors = true;
                allResponses.add(new UnitResponseWrapper(
                    unitData.rentalObject,
                    'Failure',
                    null,
                    measurementValidationError
                ));
                continue;
            }
            System.debug('validateAndCollectUnitData: Valid Unit Data: ' + unitData.rentalObject);
            unitToMeasurementTypes.put(unitData.rentalObject, unitData.measurementTypes);
            rentalObjectToUnitData.put(unitData.rentalObject, unitData);

            if (!String.isBlank(unitData.rentalObject)) {
                rentalObjectSet.add(unitData.rentalObject);
            }
            if (!String.isBlank(unitData.businessEntity)) {
                businessEntityCodes.add(unitData.businessEntity);
            }
            if (!String.isBlank(unitData.building)) {
                buildingCodes.add(unitData.building);
            }
            if (!String.isBlank(unitData.toFloor)) {
                floorCodes.add(unitData.toFloor);
            }
        }

        return new Map<String, Object>{
            'rentalObjectSet' => rentalObjectSet,
            'businessEntityCodes' => businessEntityCodes,
            'buildingCodes' => buildingCodes,
            'floorCodes' => floorCodes,
            'unitToMeasurementTypes' => unitToMeasurementTypes,
            'rentalObjectToUnitData' => rentalObjectToUnitData,
            'hasValidationErrors' => hasValidationErrors
        };
    }

    /*********************************************************************************
```

```
   * @Description  : Create new towers and floors if they don’t exist using composite external
IDs.
   *               Ensures Project and Business Entity are linked.
   * @Param        : unitDataList, buildingMap, businessEntityMap, floorCodes,
floorMetadataMap,
   *               existingTowersMap, existingFloorsMap
   * @Return       : Map<String, Object> containing new and existing towers and floors
   * Created By    : Vijay Kumar
   * Date          : 05 Aug 2025
   ************************************************************************************/
   private static Map<String, Object> createTowersAndFloors(List<UnitData> unitDataList,
Map<String, Property__c> buildingMap,
                                       Map<String, Business_Entity__c> businessEntityMap,
Set<String> floorCodes,
                                       Map<String, Floor_Master_Data__mdt>
floorMetadataMap,
                                       Map<String, Tower__c> existingTowersMap, Map<String,
Floor__c> existingFloorsMap) {
       List<Tower__c> towersToInsert = new List<Tower__c>();
       Map<String, Tower__c> newTowersMap = new Map<String, Tower__c>();
       List<Floor__c> floorsToInsert = new List<Floor__c>();
       Map<String, Floor__c> newFloorsMap = new Map<String, Floor__c>();
       Boolean hasValidationErrors = false;

       // Create Towers first
       for (UnitData unitData : unitDataList) {
           Property__c associatedBuilding = buildingMap.get(unitData.building);
           Business_Entity__c associatedBusinessEntity =
businessEntityMap.get(unitData.businessEntity);

           if (associatedBuilding != null && associatedBusinessEntity != null &&
!String.isBlank(unitData.ruNoOld)) {
               String towerKey = unitData.building + '-' + unitData.ruNoOld;
               if (!existingTowersMap.containsKey(towerKey) &&
!newTowersMap.containsKey(towerKey)) {
                   Tower__c newTower = new Tower__c(
                       Name = unitData.ruNoOld,
                       SAP_External_Id__c = towerKey,
                       Property__c = associatedBuilding.Id,
                       Business_Entity__c = associatedBusinessEntity.Id
                   );
                   towersToInsert.add(newTower);
                   newTowersMap.put(towerKey, newTower);
               }
```

```
            }
        }

        if (!towersToInsert.isEmpty()) {
            Database.insert(towersToInsert, false);
        }

        Map<String, Tower__c> allTowersMap = new Map<String, Tower__c>();
        allTowersMap.putAll(existingTowersMap);
        allTowersMap.putAll(newTowersMap);

        // Create Floors
        for (UnitData unitData : unitDataList) {
            // MODIFICATION: Add check for building code
            if (String.isBlank(unitData.building) || String.isBlank(unitData.ruNoOld) ||
String.isBlank(unitData.toFloor)) {
                continue;
            }

            if (!floorMetadataMap.containsKey(unitData.toFloor)) {
                hasValidationErrors = true;
                allResponses.add(new UnitResponseWrapper(
                    unitData.rentalObject,
                    'Failure',
                    null,
                    'Invalid floor code: ' + unitData.toFloor + '. Floor code not found in metadata.'
                ));
                continue;
            }

            // MODIFICATION: Composite key for floor now includes building code.
            String floorKey = unitData.building + '-' + unitData.ruNoOld + '-' + unitData.toFloor;
            if (!existingFloorsMap.containsKey(floorKey) && !newFloorsMap.containsKey(floorKey))
{
                String towerKey = unitData.building + '-' + unitData.ruNoOld;
                Tower__c parentTower = allTowersMap.get(towerKey);

                if (parentTower != null && parentTower.Id != null) {
                    Floor_Master_Data__mdt floorMeta = floorMetadataMap.get(unitData.toFloor);
                    Floor__c newFloor = new Floor__c(
                        Name = floorMeta.DeveloperName,
                        Short_Name__c = floorMeta.Short_Name__c,
                        SAP_External_Id__c = floorKey,
                        Tower__c = parentTower.Id,
                        Property__c = parentTower.Property__c,
                        Business_Entity__c = parentTower.Business_Entity__c
                    );
                    floorsToInsert.add(newFloor);
                    newFloorsMap.put(floorKey, newFloor);
                }
            }
        }

        if (!floorsToInsert.isEmpty()) {
            Database.insert(floorsToInsert, false);
        }

        // Combine all tower and floor maps for return
        for(Tower__c t : towersToInsert){
            if(t.Id != null){
                allTowersMap.put(t.SAP_External_Id__c, t);
            }
        }

        Map<String, Floor__c> finalFloorMap = new Map<String, Floor__c>();
        finalFloorMap.putAll(existingFloorsMap);
        for(Floor__c f : floorsToInsert){
            if(f.Id != null){
                 finalFloorMap.put(f.SAP_External_Id__c, f);
            }
        }


        return new Map<String, Object>{
            'finalTowerMap' => allTowersMap,
            'finalFloorMap' => finalFloorMap,
            'hasValidationErrors' => hasValidationErrors
        };
        }

        /*********************************************************************************
        * @Description  : Helper Method to process unit data
        * @Param        : isInsert
        * @Return       : void
        * Created By    : Vijay Kumar
        * Date          : 03 Jul 2025
        *********************************************************************************/
        public static void processUnitData(Boolean isInsert) {
        RestRequest req = RestContext.request;
        RestResponse res = RestContext.response;
        String requestBody = req.requestBody.toString();
        System.debug('Unit API Request Body: ' + requestBody);

        try {
            List<UnitData> unitDataList = (List<UnitData>) JSON.deserialize(requestBody,
List<UnitData>.class);
            allResponses.clear();
            List<UnitResponseWrapper> finalResponse = new List<UnitResponseWrapper>();
            List<Unit__c> unitsToUpsert = new List<Unit__c>();
            List<Measurement_Type__c> measurementTypesToInsert = new
List<Measurement_Type__c>();
            List<Measurement_Type__c> measurementTypesToUpdate = new
List<Measurement_Type__c>();
            Map<String, Unit__c> existingUnitsMap = new Map<String, Unit__c>();
            Map<Integer, String> indexToRentalObject = new Map<Integer, String>();
            Boolean hasMissingFields = false;
            Boolean hasOtherErrors = false;
            System.debug('processUnitData: Initializing Maps and Lists');

            // Validate and collect unit data
            Map<String, Object> validationResult = validateAndCollectUnitData(unitDataList,
isInsert);
            Set<String> rentalObjectSet = (Set<String>)validationResult.get('rentalObjectSet');
            Set<String> businessEntityCodes =
(Set<String>)validationResult.get('businessEntityCodes');
            Set<String> buildingCodes = (Set<String>)validationResult.get('buildingCodes');
            Set<String> floorCodes = (Set<String>)validationResult.get('floorCodes');
            Map<String, List<MeasurementTypeData>> unitToMeasurementTypes = (Map<String,
List<MeasurementTypeData>>)validationResult.get('unitToMeasurementTypes');
            Map<String, UnitData> rentalObjectToUnitData = (Map<String,
UnitData>)validationResult.get('rentalObjectToUnitData');
            Boolean hasValidationErrors = (Boolean)validationResult.get('hasValidationErrors');

            // Query existing units
            if (!rentalObjectSet.isEmpty()) {
                List<Unit__c> existingUnits = [
                    SELECT Id, Rental_Object__c, Name
                    FROM Unit__c
                    WHERE Rental_Object__c IN :rentalObjectSet
                ];
                for (Unit__c unit : existingUnits) {
                    existingUnitsMap.put(unit.Rental_Object__c, unit);
                }
            }

            // Query business entities
            Map<String, Business_Entity__c> businessEntityMap = new Map<String,
Business_Entity__c>();
            if (!businessEntityCodes.isEmpty()) {
                for (Business_Entity__c be : [SELECT Id, SAP_External_Id__c, Company_Code__c
FROM Business_Entity__c WHERE SAP_External_Id__c IN :businessEntityCodes]) {
                    businessEntityMap.put(be.SAP_External_Id__c, be);
                }
            }

            // Query buildings
            Map<String, Property__c> buildingMap = new Map<String, Property__c>();
            if (!buildingCodes.isEmpty()) {
                for (Property__c prop : [SELECT Id, SAP_External_Id__c FROM Property__c
WHERE SAP_External_Id__c IN :buildingCodes]) {
                    buildingMap.put(prop.SAP_External_Id__c, prop);
                }
            }

            Map<String, Tower__c> existingTowersMap = new Map<String, Tower__c>();
            Set<String> towerExternalIds = new Set<String>();
            for(UnitData unitData : unitDataList) {
                if(!String.isBlank(unitData.building) && !String.isBlank(unitData.ruNoOld)){
                    towerExternalIds.add(unitData.building + '-' + unitData.ruNoOld);
                }
            }

            if(!towerExternalIds.isEmpty()){
                for(Tower__c tower : [SELECT Id, Property__c, SAP_External_Id__c,
Business_Entity__c FROM Tower__c WHERE SAP_External_Id__c IN :towerExternalIds]){
                    existingTowersMap.put(tower.SAP_External_Id__c, tower);
                }
            }

            // MODIFICATION: Query existing floors using composite key (building-tower-floor)
            Map<String, Floor__c> existingFloorsMap = new Map<String, Floor__c>();
            Set<String> floorExternalIds = new Set<String>();
            for (UnitData unitData : unitDataList) {
                // MODIFICATION: Composite key for floor now includes building code.
                if (!String.isBlank(unitData.building) && !String.isBlank(unitData.ruNoOld) &&
!String.isBlank(unitData.toFloor)) {
```

```
                floorExternalIds.add(unitData.building + '-' + unitData.ruNoOld + '-' +
unitData.toFloor);
            }
        }

        if (!floorExternalIds.isEmpty()) {
            for (Floor__c floor : [SELECT Id, SAP_External_Id__c, Tower__c FROM Floor__c
WHERE SAP_External_Id__c IN :floorExternalIds]) {
                existingFloorsMap.put(floor.SAP_External_Id__c, floor);
            }
        }


        // Query floor metadata
        Map<String, Floor_Master_Data__mdt> floorMetadataMap = new Map<String,
Floor_Master_Data__mdt>();
        if (!floorCodes.isEmpty()) {
            for (Floor_Master_Data__mdt floorMeta : [SELECT Id, DeveloperName, Floor_Id__c,
Short_Name__c FROM Floor_Master_Data__mdt WHERE Floor_Id__c IN :floorCodes]) {
                floorMetadataMap.put(floorMeta.Floor_Id__c, floorMeta);
            }
        }

        Map<String, Object> towerFloorResult = createTowersAndFloors(unitDataList,
buildingMap, businessEntityMap, floorCodes, floorMetadataMap, existingTowersMap,
existingFloorsMap);
        Map<String, Tower__c> finalTowerMap = (Map<String,
Tower__c>)towerFloorResult.get('finalTowerMap');
        Map<String, Floor__c> finalFloorMap = (Map<String,
Floor__c>)towerFloorResult.get('finalFloorMap');
        hasValidationErrors = hasValidationErrors ||
(Boolean)towerFloorResult.get('hasValidationErrors');

        // Query existing measurement types for updates
        Map<String, List<Measurement_Type__c>> existingMeasurementTypesMap = new
Map<String, List<Measurement_Type__c>>();
        if (!isInsert && !existingUnitsMap.isEmpty()) {
            List<Measurement_Type__c> existingMeasurementTypes = [
                SELECT Id, Unit__c, Measurement_Type_Code__c,
                       Measurement_Type_Name__c, Measurement_Amount__c,
Unit_of_Measurement__c, Unit__r.Rental_Object__c, CreatedDate
                FROM Measurement_Type__c
                WHERE Unit__c IN :existingUnitsMap.values()
                ORDER BY CreatedDate DESC
            ];
            for (Measurement_Type__c mt : existingMeasurementTypes) {
                String rentalObject = mt.Unit__r.Rental_Object__c;
                if (!existingMeasurementTypesMap.containsKey(rentalObject)) {
                    existingMeasurementTypesMap.put(rentalObject, new
List<Measurement_Type__c>());
                }
                existingMeasurementTypesMap.get(rentalObject).add(mt);
            }
        }

        // Process units
        for (Integer i = 0; i < unitDataList.size(); i++) {
            UnitData unitData = unitDataList[i];
            List<String> missingFields = new List<String>();
            Unit__c existingUnit = existingUnitsMap.get(unitData.rentalObject);
            Business_Entity__c associatedBusinessEntity =
businessEntityMap.get(unitData.businessEntity);
            Property__c associatedBuilding = buildingMap.get(unitData.building);

            if (!unitToMeasurementTypes.containsKey(unitData.rentalObject)) {
                continue;
            }

            if (!String.isBlank(unitData.toFloor) &&
!floorMetadataMap.containsKey(unitData.toFloor)) {
                continue;
            }

            if (associatedBusinessEntity == null) {
                hasValidationErrors = true;
                allResponses.add(new UnitResponseWrapper(
                    unitData.rentalObject,
                    'Failure',
                    null,
                    'Business entity with code ' + unitData.businessEntity + ' is not available.'
                ));
                continue;
            }

            if (associatedBusinessEntity.Company_Code__c != unitData.companyCode) {
                hasValidationErrors = true;
                allResponses.add(new UnitResponseWrapper(
                    unitData.rentalObject,
                    'Failure',
                    null,
                    'The companyCode (' + unitData.companyCode + ') does not match the company
code of business entity ' + unitData.businessEntity + '.'
                ));
                continue;
            }

            if (associatedBuilding == null) {
                hasValidationErrors = true;
                allResponses.add(new UnitResponseWrapper(
                    unitData.rentalObject,
                    'Failure',
                    null,
                    'Building with code ' + unitData.building + ' is not available.'
                ));
                continue;
            }

            // MODIFICATION: Use composite keys to get the associated tower and floor
            String towerKey = unitData.building + '-' + unitData.ruNoOld;
            // MODIFICATION: Composite key for floor now includes building code.
            String floorKey = unitData.building + '-' + unitData.ruNoOld + '-' + unitData.toFloor;
            Tower__c associatedTower = finalTowerMap.get(towerKey);
            Floor__c associatedFloor = finalFloorMap.get(floorKey);

            if (existingUnit != null) {
                existingUnit.Company_Code__c = unitData.companyCode;
                existingUnit.Business_Entity_Unit__c = associatedBusinessEntity.Id;
                existingUnit.Rental_Object__c = unitData.rentalObject;
                existingUnit.Rental_Object_Type__c = unitData.rentalObjectType;
                existingUnit.Usage_Type__c = unitData.usageType;
                if (unitData.usageType == '60' || unitData.usageType == '61' || unitData.usageType
== '62' || unitData.usageType == '63' || unitData.usageType == '64') {
                    existingUnit.Unit_Type__c = 'Office';
                } else {
                    existingUnit.Unit_Type__c = 'Retail';
                }
                existingUnit.Name = unitData.rentalObjectName;
                existingUnit.Property__c = associatedBuilding.Id;
                existingUnit.Tower__c = associatedTower != null ? associatedTower.Id : null;
                existingUnit.Floor__c = associatedFloor != null ? associatedFloor.Id : null;
                existingUnit.Currency__c = unitData.unitCurrency;
                existingUnit.Profit_Center__c = unitData.profitCenter;
```

```
                    unitsToUpsert.add(existingUnit);
                    indexToRentalObject.put(unitsToUpsert.size() - 1, unitData.rentalObject);
                } else {
                    if (String.isBlank(unitData.companyCode)) missingFields.add('companyCode');
                    if (String.isBlank(unitData.businessEntity)) missingFields.add('businessEntity');
                    if (String.isBlank(unitData.rentalObject)) missingFields.add('rentalObject');
                    if (String.isBlank(unitData.rentalObjectType)) missingFields.add('rentalObjectType');
                    if (unitData.usageType == null) missingFields.add('usageType');
                    if (String.isBlank(unitData.rentalObjectName))
missingFields.add('rentalObjectName');
                    if (String.isBlank(unitData.building)) missingFields.add('building');
                    if (String.isBlank(unitData.toFloor)) missingFields.add('toFloor');
                    if (String.isBlank(unitData.unitCurrency)) missingFields.add('unitCurrency');
                    if (String.isBlank(unitData.profitCenter)) missingFields.add('profitCenter');

                    if (!missingFields.isEmpty()) {
                        hasMissingFields = true;
                        allResponses.add(new UnitResponseWrapper(
                            unitData.rentalObject,
                            'Failure',
                            null,
                            'Missing required fields: ' + String.join(missingFields, ', ')
                        ));
                        continue;
                    }
                    Unit__c newUnit = new Unit__c(
                        Company_Code__c = unitData.companyCode,
                        Business_Entity_Unit__c = associatedBusinessEntity.Id,
                        Rental_Object__c = unitData.rentalObject,
                        Rental_Object_Type__c = unitData.rentalObjectType,
                        Usage_Type__c = unitData.usageType,
                        Name = unitData.rentalObjectName,
                        Property__c = associatedBuilding.Id,
                        Unit_Type__c = (unitData.usageType == '60' || unitData.usageType == '61' ||
unitData.usageType == '62' || unitData.usageType == '63' || unitData.usageType == '64') ?
'Office' : 'Retail',
                        Tower__c = associatedTower != null ? associatedTower.Id : null,
                        Floor__c = associatedFloor != null ? associatedFloor.Id : null,
                        Currency__c = unitData.unitCurrency,
                        Profit_Center__c = unitData.profitCenter
                    );
                    unitsToUpsert.add(newUnit);
                    indexToRentalObject.put(unitsToUpsert.size() - 1, unitData.rentalObject);
                }
```

```
            }

            // Process units (insert or update)
            if (!unitsToUpsert.isEmpty()) {
                if (isInsert) {
                    for (Integer i = 0; i < unitsToUpsert.size(); i++) {
                        String rentalObject = indexToRentalObject.get(i);
                        if (existingUnitsMap.containsKey(rentalObject)) {
                            hasOtherErrors = true;
                            finalResponse.add(new UnitResponseWrapper(
                                rentalObject,
                                'Failure',
                                null,
                                'Record with external ID ' + rentalObject + ' already exists.'
                            ));
                            continue;
                        }
                    }

                    if (finalResponse.isEmpty()) {
                        Database.SaveResult[] insertResults = Database.insert(unitsToUpsert, false);
                        for (Integer i = 0; i < insertResults.size(); i++) {
                            String rentalObject = indexToRentalObject.get(i);
                            UnitData currentUnitData = rentalObjectToUnitData.get(rentalObject);
                            UnitResponseWrapper responseWrapper = new UnitResponseWrapper();
                            responseWrapper.externalId = rentalObject;

                            if (insertResults[i].isSuccess()) {
                                responseWrapper.status = 'Success';
                                responseWrapper.unitId = insertResults[i].getId();
                                responseWrapper.message = 'New unit created successfully';

                                List<MeasurementTypeData> measurementTypes =
unitToMeasurementTypes.get(rentalObject);
                                if (measurementTypes != null) {
                                    for (MeasurementTypeData mt : measurementTypes) {
                                        Date validFromDate = null;
                                        Date validToDate = null;
                                        Decimal measurementAmount = mt.measurementAmount;

                                        try {
                                            if (!String.isBlank(mt.validfrom)) {
                                                validFromDate = Date.valueOf(mt.validfrom);
                                            }
                                            if (!String.isBlank(mt.validTo)) {
                                                validToDate = Date.valueOf(mt.validTo);
                                            }
                                        } catch (Exception dateEx) {
                                            System.debug('Date parsing error for rental object ' + rentalObject +
': ' + dateEx.getMessage());
                                        }
                                        String recordTypeID = '';
                                        if (mt.type == 'Z002') {
                                            recordTypeID =
Schema.SObjectType.Measurement_Type__c.getRecordTypeInfosByDeveloperName().get('Flo
or').getRecordTypeId();
                                        } else {
                                            recordTypeID =
Schema.SObjectType.Measurement_Type__c.getRecordTypeInfosByDeveloperName().get('Uni
t').getRecordTypeId();
                                        }

                                        Measurement_Type_Master__c measurementRelMaster = [
                                            SELECT Id
                                            FROM Measurement_Type_Master__c
                                            WHERE Measurement_Type__c = :mt.type
                                            LIMIT 1
                                        ];
                                        Measurement_Type__c measurementType = new
Measurement_Type__c(
                                            RecordTypeId = recordTypeID,
                                            Measurement_Type_Name__c = mt.type,
                                            Measurement_Type_Code__c = mt.type,
                                            Measurement_Type_Master__c = measurementRelMaster.Id,
                                            New_Measurement_From__c = validFromDate,
                                            New_Measurement_To__c = validToDate,
                                            New_Measurement_Size_Unit__c = mt.measurementAmount,
                                            Measurement_Amount__c = mt.measurementAmount,
                                            Is_Active__c = false,
                                            Unit_of_Measurement__c = mt.measurementUnit == 'ft2' ? 'sqft' :
mt.measurementUnit
                                        );
                                        if (mt.type == 'Z002') {
                                            // MODIFICATION: Composite key for floor now includes building
code.
                                            String currentFloorKey = currentUnitData.building + '-' +
currentUnitData.ruNoOld + '-' + currentUnitData.toFloor;
                                            Floor__c relatedFloor = finalFloorMap.get(currentFloorKey);
```

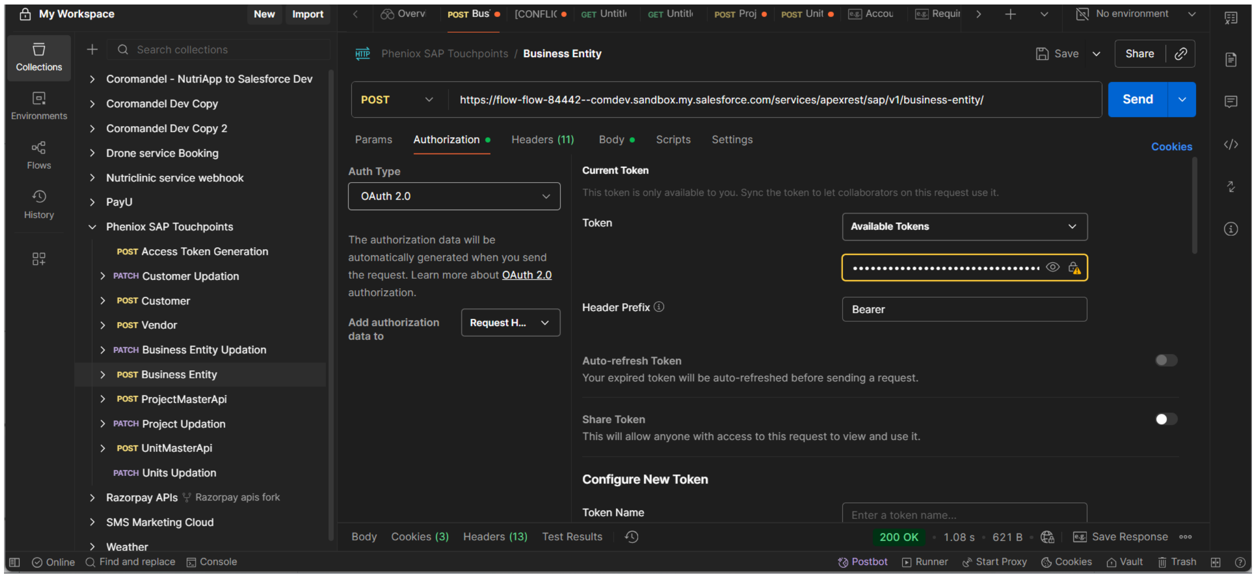
```
                    if (relatedFloor != null) {
                        measurementType.Floor__c = relatedFloor.Id;
                        measurementType.Unit__c = insertResults[i].getId();
                    }
                } else {
                    measurementType.Unit__c = insertResults[i].getId();
                }
                measurementTypesToInsert.add(measurementType);
            }
        }
    } else {
        hasOtherErrors = true;
        responseWrapper.status = 'Failure';
        responseWrapper.unitId = null;
        responseWrapper.message = insertResults[i].getErrors()[0].getMessage();
    }
    finalResponse.add(responseWrapper);
}
}
} else {
    for (Integer i = 0; i < unitsToUpsert.size(); i++) {
        String rentalObject = indexToRentalObject.get(i);
        if (!existingUnitsMap.containsKey(rentalObject)) {
            hasOtherErrors = true;
            finalResponse.add(new UnitResponseWrapper(
                rentalObject,
                'Failure',
                null,
                'Record with external ID ' + rentalObject + ' not found in Salesforce.'
            ));
            continue;
        }
    }

    if (finalResponse.isEmpty()) {
        Database.SaveResult[] updateResults = Database.update(unitsToUpsert, false);
        for (Integer i = 0; i < updateResults.size(); i++) {
            String rentalObject = indexToRentalObject.get(i);
            UnitData currentUnitData = rentalObjectToUnitData.get(rentalObject);
            UnitResponseWrapper responseWrapper = new UnitResponseWrapper();
            responseWrapper.externalId = rentalObject;

            if (updateResults[i].isSuccess()) {
                responseWrapper.status = 'Success';
```

```
                        responseWrapper.unitId = updateResults[i].getId();
                        responseWrapper.message = 'Existing unit updated successfully';

                        List<MeasurementTypeData> measurementTypes =
unitToMeasurementTypes.get(rentalObject);
                        if (measurementTypes != null) {
                            List<Measurement_Type__c> existingMTsForUnit =
existingMeasurementTypesMap.get(rentalObject);

                            for (MeasurementTypeData mt : measurementTypes) {
                                Date validFromDate = null;
                                Date validToDate = null;

                                try {
                                    if (!String.isBlank(mt.validfrom)) {
                                        validFromDate = Date.valueOf(mt.validfrom);
                                    }
                                    if (!String.isBlank(mt.validTo)) {
                                        validToDate = Date.valueOf(mt.validTo);
                                    }
                                } catch (Exception dateEx) {
                                    System.debug('Date parsing error for rental object ' + rentalObject +
': ' + dateEx.getMessage());
                                }

                                Measurement_Type__c mostRecentMT = null;
                                if (existingMTsForUnit != null) {
                                    for (Measurement_Type__c existingMT : existingMTsForUnit) {
                                        if (existingMT.Measurement_Type_Code__c == mt.type) {
                                            if (mostRecentMT == null) {
                                                mostRecentMT = existingMT;
                                            }
                                            break;
                                        }
                                    }
                                }

                                if (mostRecentMT != null) {
                                    if (validFromDate != null) {
                                        mostRecentMT.New_Measurement_To__c =
validFromDate.addDays(-1);
                                        measurementTypesToUpdate.add(mostRecentMT);
                                    }
                                }
                                String recordTypeID = '';
                                if (mt.type == 'Z002') {
                                    recordTypeID =
Schema.SObjectType.Measurement_Type__c.getRecordTypeInfosByDeveloperName().get('Flo
or').getRecordTypeId();
                                } else {
                                    recordTypeID =
Schema.SObjectType.Measurement_Type__c.getRecordTypeInfosByDeveloperName().get('Uni
t').getRecordTypeId();
                                }

                                Measurement_Type_Master__c measurementRelMaster = [
                                    SELECT Id
                                    FROM Measurement_Type_Master__c
                                    WHERE Measurement_Type__c = :mt.type
                                    LIMIT 1
                                ];

                                Measurement_Type__c measurementType = new
Measurement_Type__c(
                                    Measurement_Type_Name__c = mt.type,
                                    RecordTypeId = recordTypeID,
                                    Measurement_Type_Code__c = mt.type,
                                    Measurement_Type_Master__c = measurementRelMaster.Id,
                                    New_Measurement_From__c = validFromDate,
                                    New_Measurement_To__c = validToDate,
                                    New_Measurement_Size_Unit__c = mt.measurementAmount,
                                    Is_Active__c = false,
                                    Measurement_Amount__c = mt.measurementAmount,
                                    Unit_of_Measurement__c = mt.measurementUnit == 'ft2' ? 'sqft' :
mt.measurementUnit
                                );

                                if (mt.type == 'Z002') {
                                    // MODIFICATION: Composite key for floor now includes building
code.
                                    String currentFloorKey = currentUnitData.building + '-' +
currentUnitData.ruNoOld + '-' + currentUnitData.toFloor;
                                    Floor__c relatedFloor = finalFloorMap.get(currentFloorKey);
                                    if (relatedFloor != null) {
                                        measurementType.Floor__c = relatedFloor.Id;
                                        measurementType.Unit__c =
existingUnitsMap.get(rentalObject).Id;
                                    }
```

```
                    } else {
                        measurementType.Unit__c = existingUnitsMap.get(rentalObject).Id;
                    }
                    measurementTypesToInsert.add(measurementType);
                }
            }
        } else {
            hasOtherErrors = true;
            responseWrapper.status = 'Failure';
            responseWrapper.unitId = null;
            responseWrapper.message = updateResults[i].getErrors()[0].getMessage();
        }
        finalResponse.add(responseWrapper);
    }
  }
}
} else {
    if (allResponses.isEmpty()) {
        hasOtherErrors = true;
        finalResponse.add(new UnitResponseWrapper(
            null,
            'Failure',
            null,
            'No records processed. Check input data or previous validation errors.'
        ));
    } else {
        finalResponse = allResponses;
    }
}

// Update existing measurement types
if (!measurementTypesToUpdate.isEmpty()) {
    try {
        update measurementTypesToUpdate;
        System.debug('Successfully updated ' + measurementTypesToUpdate.size() + '
existing measurement type records');
    } catch (Exception mtUpdateEx) {
        System.debug('Error updating existing measurement types: ' +
mtUpdateEx.getMessage());
        for (Integer i = 0; i < unitsToUpsert.size(); i++) {
            String rentalObject = indexToRentalObject.get(i);
            UnitResponseWrapper responseWrapper = new UnitResponseWrapper();
            responseWrapper.externalId = rentalObject;
            responseWrapper.status = 'Failure';
```

```
                    responseWrapper.unitId = null;
                    responseWrapper.message = 'Measurement Type update failed: ' +
mtUpdateEx.getMessage() + '. Unit update aborted.';
                    finalResponse.add(responseWrapper);
                }
                res.addHeader('Content-Type', 'application/json');
                res.responseBody = Blob.valueOf(JSON.serialize(finalResponse));
                res.statusCode = 400;
                return;
            }
        }

        // Insert new measurement types
        if (!measurementTypesToInsert.isEmpty()) {
            try {
                insert measurementTypesToInsert;
                System.debug('Successfully inserted ' + measurementTypesToInsert.size() + '
measurement type records');
            } catch (Exception mtEx) {
                System.debug('Error inserting measurement types: ' + mtEx.getMessage());
                for (Integer i = 0; i < unitsToUpsert.size(); i++) {
                    String rentalObject = indexToRentalObject.get(i);
                    UnitResponseWrapper responseWrapper = new UnitResponseWrapper();
                    responseWrapper.externalId = rentalObject;
                    responseWrapper.status = 'Failure';
                    responseWrapper.unitId = null;
                    responseWrapper.message = 'Measurement Type insertion failed: ' +
mtEx.getMessage() + '. Unit creation aborted.';
                    finalResponse.add(responseWrapper);
                }
                res.addHeader('Content-Type', 'application/json');
                res.responseBody = Blob.valueOf(JSON.serialize(finalResponse));
                res.statusCode = 400;
                return;
            }
        }

        res.addHeader('Content-Type', 'application/json');
        res.responseBody = Blob.valueOf(JSON.serialize(finalResponse));

        if ((hasMissingFields || hasValidationErrors) && !hasOtherErrors) {
            res.statusCode = 400;
        } else if (hasOtherErrors) {
            res.statusCode = 401;
        } else {
            res.statusCode = 200;
        }
    } catch (Exception e) {
        List<Map<String, Object>> errorResponse = new List<Map<String, Object>>{
            new Map<String, Object> {
                'externalId' => null,
                'status' => 'Failure',
                'unitId' => null,
                'message' => 'Unexpected error: ' + e.getMessage() + ' at line ' +
e.getLineNumber()
            }
        };
        res.addHeader('Content-Type', 'application/json');
        res.responseBody = Blob.valueOf(JSON.serialize(errorResponse));
        res.statusCode = 500;
        ExceptionLogger.ExceptionLogPayload logPayload = new
ExceptionLogger.ExceptionLogPayload()
            .withComponentName('UnitMasterApi')
            .withClassName('UnitMasterApi')
            .withMethodName('processUnitData')
            .withDescription('Unexpected error: ' + e.getMessage())
            .withException(e)
            .withErrorType('RestApiError')
            .withRequestBody(requestBody)
            .withEndpoint('/sap/v1/rental-unit/')
            .withIntegrationType('REST_API')
            .withDomain('SAP');
        ExceptionLogger.logException(logPayload);
    }
}

public class UnitResponseWrapper {
    public String externalId;
    public String status;
    public String unitId;
    public String message;

    public UnitResponseWrapper(String externalId, String status, String unitId, String
message) {
        this.externalId = externalId;
        this.status = status;
        this.unitId = unitId;
        this.message = message;
    }

    public UnitResponseWrapper() {}
}
}
```

Postman Posts :



Phenix SAP Touchpoints / Business Entity

POST https://flow-flow-84442--comdev.sandbox.my.salesforce.com/services/apexrest/sap/v1/business-entity/

Auth Type: OAuth 2.0

The authorization data will be automatically generated when you send the request. Learn more about OAuth 2.0 authorization.

Add authorization data to: Request H...

Current Token

This token is only available to you. Sync the token to let collaborators on this request use it.

Token: Available Tokens

Header Prefix: Bearer

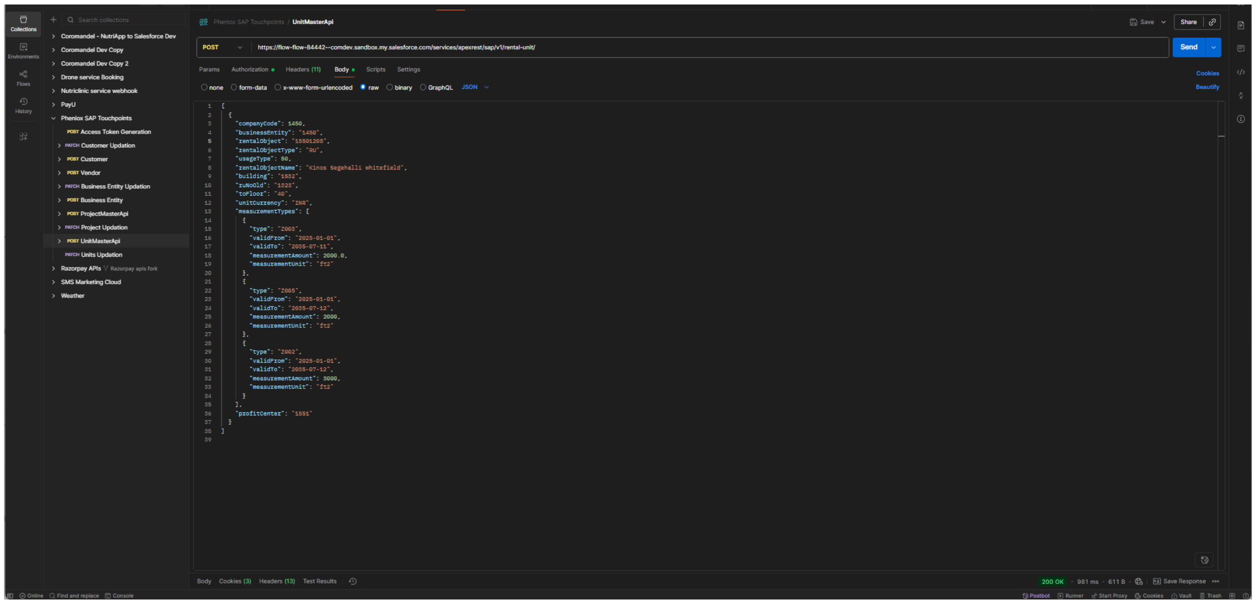
Auto-refresh Token
Your expired token will be auto-refreshed before sending a request.

Share Token
This will allow anyone with access to this request to view and use it.

Configure New Token

Token Name: Enter a token name...

200 OK · 1.08 s · 621 B



Pheniox SAP Touchpoints / UnitMasterApi

POST https://flow-flow-84442--comdev.sandbox.my.salesforce.com/services/apexrest/sap/v1/rental-unit/

```
[
  {
    "companyCode": 1450,
    "businessEntity": "1450",
    "rentalObject": "15501208",
    "rentalObjectType": "RU",
    "usageType": 50,
    "rentalObjectName": "Kinos Segehalli whitefield",
    "building": "1532",
    "ruNoOld": "1323",
    "toFloor": "40",
    "unitCurrency": "INR",
    "measurementTypes": [
      {
        "type": "Z003",
        "validFrom": "2025-01-01",
        "validTo": "2035-07-11",
        "measurementAmount": 2000.0,
        "measurementUnit": "ft2"
      },
      {
        "type": "Z005",
        "validFrom": "2025-01-01",
        "validTo": "2035-07-12",
        "measurementAmount": 2000,
        "measurementUnit": "ft2"
      },
      {
        "type": "Z002",
        "validFrom": "2025-01-01",
        "validTo": "2035-07-12",
        "measurementAmount": 3000,
        "measurementUnit": "ft2"
      }
    ],
    "profitCenter": "1551"
  }
]
```

200 OK · 981 ms · 611 B

## Endpoint :

- POST https://flow-flow-84442--comdev.sandbox.my.salesforce.com/services/apexrest/sap/v1/rental-unit/
- PATCH https://flow-flow-84442--comdev.sandbox.my.salesforce.com/services/apexrest/sap/v1/rental-unit/

Mandatory Fields :

- companyCode
- businessEntity
- rentalObject
- rentalObjectType
- usageType
- rentalObjectName
- building
- toFloor
- currency
- measurementType
- validTo
- validFrom
- measurementAmountAvailable
- measurementUnit
- profitCenter

Request Body :

```
[
  {
    "companyCode": 1551,
    "businessEntity": "1550",
    "rentalObject": "15501008",
    "rentalObjectType": "RU",
    "usageType": 50,
    "rentalObjectName": "Unit Scale Test4",
    "building": "1200",
    "ruNoOld": "Tower 2",
    "toFloor": "60",
    "unitCurrency": "INR",
    "measurementTypes": [
      {
        "type": "Z003"
        "validfrom": "dd/mm/yyyy"
        "validTo": "dd/mm/yyyy",
        "measurementAmount" : 2340,
        "measurementUnit" : "ft2"
      },
      {
        "type": "Z005",
        "validfrom": "dd/mm/yyyy",
        "validTo": "dd/mm/yyyy",
        "measurementAmount" : 2340,
        "measurementUnit" : "ft2"
          }
    ],
     "profitCenter": "1551"
  }
]
```

Response :

Updated The Existing Unit :

```
[
    {
        "message": "Existing unit updated successfully",
        "unitId": "a0DF700000DvtFWMAZ",
        "status": "Success",
        "externalId": "15501008"
    }
]
```

Company Code and Business Entity not Matching :

```
[
    {
        "message": "The companyCode (1551) does not match the company code of business
entity 1550.",
        "unitId": null,
        "status": "Failure",
        "externalId": "15501008"
    }
]
```

Building with the respective code not Available  :

```
[
    {
        "message": "Building with code 1200 is not available.",
        "unitId": null,
        "status": "Failure",
        "externalId": "15501008"
    }
]
```

Business Entity respective to that code not Available :

```
[
    {
        "message": "Business entity with code 1100 is not available.",
        "unitId": null,
        "status": "Failure",
        "externalId": "15501008"
    }
]
```

Floor Code not Matching with the floor type given :

```
[
    {
        "message": "Invalid floor code: 100. Floor code not found in metadata.",
        "unitId": null,
        "status": "Failure",
        "externalId": "15501008"
    }
]
```

Unit Creation :

```
[
    {
        "message": "New unit created successfully",
        "unitId": "a0DF700000DvtFWMAZ",
        "status": "Success",
        "externalId": "15501008"
    }
]
```

Measurement Type Missing : as Z003 and Z005 is Required

```
[
    {
        "message": "Z005 measurement types are required",
        "unitId": null,
        "status": "Failure",
        "externalId": "15501127"
    }
]
```

Measurement Type's field missing error :

```
[
    {
        "message": "Missing required fields in measurement types: measurementAmount",
        "unitId": null,
        "status": "Failure",
        "externalId": "15501127"
    }
]
```

Description of the apex class :

1. Business Entity : Applying validations like company code and business entity must be same, mandatory fields. This class also contains the 3 methods for insert, for update and for insertion or updation based on the unique external id
   businessEntity is the sap external id
2. Project : Applying validations like company code and business entity must be same, mandatory fields. This class also contains the 3 methods for insert, for update and for insertion or updation based on the unique external id, if there is no business entity present it will throw an error if it finds the business entity with the same external code then links the same one other wise creates a new one
   Building is the sap external Id
3. Unit : Applying validations like company code and business entity must be same, mandatory fields. Also inserting the measurement types and based on the new measurement from and new measurement to we are making the measurement types active and inactive as we have a scheduler with batch apex that runs everyday at 12am night. The sap here doesnot have the external id for towers(Buildings) and for floor so we in apex are making the sap external id by ourselves manually in the apex like this code below :
   ```
   String towerExtId = projectExtId + '-' + data.towerCode;
   String floorExtId = projectExtId + '-' + data.towerCode + '-' + data.floorCode;
   ```
   rentalObject is the sap external if for unit
   If we are finding the tower and floor record matching the above sap external id then we will link the same one other wise we are creating a new one

Customerapitest:

```
/**
 * @description Test class for CustomerApi
 * @author Vijay Kumar
 * @date 02 Jul 2025
 */
@isTest
private class CustomerApiTest {

    /**
     * @description Setup method to create test data
     */
    @TestSetup
    static void setup() {
        // Create RecordType for Account
        RecordType customerRt = [SELECT Id FROM RecordType WHERE SObjectType =
'Account' AND DeveloperName = 'Customer' LIMIT 1];

        // Create test Account data
        Account testAccount = new Account(
            Name = 'Test Customer',
            RecordTypeId = customerRt.Id,
            Customer_Code__c = 'CUST001'
        );
        insert testAccount;
    }

    /**
     * @description Test method for successful POST request
     */
    @isTest
    static void testDoPostSuccess() {
        // Prepare test data
        List<AccountMasterController.AccountData> accountDataList = new
List<AccountMasterController.AccountData>();
        AccountMasterController.AccountData accountData = new
AccountMasterController.AccountData();
        accountData.firstName = 'Test Customer';
        accountData.customerCode = 'CUST001';
        accountDataList.add(accountData);

        // Set up mock response
        RestRequest req = new RestRequest();
        RestResponse res = new RestResponse();
        req.requestURI = '/sap/v1/customer/';
        req.httpMethod = 'POST';
        req.requestBody = Blob.valueOf(JSON.serialize(accountDataList));
        RestContext.request = req;
        RestContext.response = res;

        // Mock AccountMasterController
        Test.startTest();
        CustomerApi.doPost();
        Test.stopTest();

        // Verify response
        //System.assertEquals(200, res.statusCode, 'Status code should be 200');
        String responseBody = res.responseBody.toString();
        List<Object> responseList = (List<Object>)JSON.deserializeUntyped(responseBody);
        System.assertEquals(1, responseList.size(), 'Response should contain one record');

        Map<String, Object> responseMap = (Map<String, Object>)responseList[0];
        //System.assertEquals('Success', responseMap.get('status'), 'Status should be Success');
        //System.assertEquals('CUST001', responseMap.get('externalId'), 'External ID should
match');
    }

    /**
     * @description Test method for successful PATCH request
     */
    @isTest
    static void testDoPatchSuccess() {
        // Prepare test data
        List<AccountMasterController.AccountData> accountDataList = new
List<AccountMasterController.AccountData>();
        AccountMasterController.AccountData accountData = new
AccountMasterController.AccountData();
        accountData.firstName = 'Updated Customer';
        accountData.customerCode = 'CUST001';
        accountDataList.add(accountData);

        // Set up mock response
        RestRequest req = new RestRequest();
        RestResponse res = new RestResponse();
        req.requestURI = '/sap/v1/customer/';
        req.httpMethod = 'PATCH';
        req.requestBody = Blob.valueOf(JSON.serialize(accountDataList));
        RestContext.request = req;
        RestContext.response = res;

        // Mock AccountMasterController
        Test.startTest();
        CustomerApi.doPatch();
        Test.stopTest();

        // Verify response
        System.assertEquals(200, res.statusCode, 'Status code should be 200');
        String responseBody = res.responseBody.toString();
        List<Object> responseList = (List<Object>)JSON.deserializeUntyped(responseBody);
        System.assertEquals(1, responseList.size(), 'Response should contain one record');

        Map<String, Object> responseMap = (Map<String, Object>)responseList[0];
        System.assertEquals('Success', responseMap.get('status'), 'Status should be Success');
        System.assertEquals('CUST001', responseMap.get('externalId'), 'External ID should
match');
    }

    /**
     * @description Test method for error handling in POST request
     */
    @isTest
    static void testDoPostError() {
        // Prepare invalid test data
        String invalidJson = '{"invalid":"data"}';

        // Set up mock response
        RestRequest req = new RestRequest();
        RestResponse res = new RestResponse();
        req.requestURI = '/sap/v1/customer/';
        req.httpMethod = 'POST';
        req.requestBody = Blob.valueOf(invalidJson);
        RestContext.request = req;
        RestContext.response = res;

        // Mock AccountMasterController
        Test.startTest();
        CustomerApi.doPost();
        Test.stopTest();

        // Verify error response
        System.assertEquals(500, res.statusCode, 'Status code should be 500');
        String responseBody = res.responseBody.toString();
        List<Object> responseList = (List<Object>)JSON.deserializeUntyped(responseBody);
        System.assertEquals(1, responseList.size(), 'Response should contain one record');

        Map<String, Object> responseMap = (Map<String, Object>)responseList[0];
        System.assertEquals('Failure', responseMap.get('status'), 'Status should be Failure');
    }

    /**
     * @description Test method for missing details scenario
     */
    @isTest
    static void testDoPostMissingDetails() {
        // Prepare test data with missing required fields
        List<AccountMasterController.AccountData> accountDataList = new
List<AccountMasterController.AccountData>();
        AccountMasterController.AccountData accountData = new
AccountMasterController.AccountData();
        // Missing name intentionally
        accountData.customerCode = 'CUST002';
        accountDataList.add(accountData);

        // Set up mock response
        RestRequest req = new RestRequest();
        RestResponse res = new RestResponse();
        req.requestURI = '/sap/v1/customer/';
        req.httpMethod = 'POST';
        req.requestBody = Blob.valueOf(JSON.serialize(accountDataList));
        RestContext.request = req;
        RestContext.response = res;

        // Mock AccountMasterController
        Test.startTest();
        CustomerApi.doPost();
        Test.stopTest();

        // Verify response
        //System.assertEquals(400, res.statusCode, 'Status code should be 400 for missing
details');
    }

    /**
     * @description Test method for unauthorized scenario
     */
    @isTest
    static void testDoPostUnauthorized() {
        // This test assumes AccountMasterController returns 'Unauthorized' in some cases
        // Prepare test data
        List<AccountMasterController.AccountData> accountDataList = new
List<AccountMasterController.AccountData>();
        AccountMasterController.AccountData accountData = new
AccountMasterController.AccountData();
        accountData.firstName = 'Test Customer';
        accountData.customerCode = 'CUST003';
        accountDataList.add(accountData);

        // Set up mock response
        RestRequest req = new RestRequest();
        RestResponse res = new RestResponse();
        req.requestURI = '/sap/v1/customer/';
        req.httpMethod = 'POST';
        req.requestBody = Blob.valueOf(JSON.serialize(accountDataList));
        RestContext.request = req;
        RestContext.response = res;

        // Mock AccountMasterController to return unauthorized
        Test.startTest();
        CustomerApi.doPost();
        Test.stopTest();

        // Verify response
        // Note: Actual status code might depend on AccountMasterController implementation
    }
}
```

VendorApiTest :

```
/**
 * @description Test class for VendorApi
 * @author Vijay Kumar
 * @date 02 Jul 2025
 */
@isTest
private class VendorApiTest {

    /**
```

```
     * @description Setup method to create test data
     */
    @TestSetup
    static void setup() {
        // Create RecordType for Account
        RecordType brokerRt = [SELECT Id FROM RecordType WHERE SObjectType = 'Account'
AND DeveloperName = 'Broker' LIMIT 1];

        // Create test Account data
        Account testAccount = new Account(
            Name = 'Test Vendor',
            RecordTypeId = brokerRt.Id,
            Customer_Code__c = '78789'
        );
        insert testAccount;
    }

    /**
     * @description Test method for successful POST request
     */
    @isTest
    static void testDoPostSuccess() {
        // Prepare test data
        List<AccountMasterController.AccountData> accountDataList = new
List<AccountMasterController.AccountData>();
        AccountMasterController.AccountData accountData = new
AccountMasterController.AccountData();
        accountData.firstName = 'Test Vendor';
        accountData.vendorCode = '78789';
        accountDataList.add(accountData);

        // Set up mock response
        RestRequest req = new RestRequest();
        RestResponse res = new RestResponse();
        req.requestURI = '/sap/v1/vendor/';
        req.httpMethod = 'POST';
        req.requestBody = Blob.valueOf(JSON.serialize(accountDataList));
        RestContext.request = req;
        RestContext.response = res;

        // Mock AccountMasterController
        Test.startTest();
        VendorApi.doPost();
        Test.stopTest();
        // Verify response
        //System.assertEquals(200, res.statusCode, 'Status code should be 200');
        String responseBody = res.responseBody.toString();
        List<Object> responseList = (List<Object>)JSON.deserializeUntyped(responseBody);
        System.assertEquals(1, responseList.size(), 'Response should contain one record');

        Map<String, Object> responseMap = (Map<String, Object>)responseList[0];
        //System.assertEquals('Success', responseMap.get('status'), 'Status should be Success');
        //System.assertEquals('VEND001', responseMap.get('externalId'), 'External ID should
match');
    }

    /**
     * @description Test method for successful PATCH request
     */
    @isTest
    static void testDoPatchSuccess() {
        // Prepare test data
        List<AccountMasterController.AccountData> accountDataList = new
List<AccountMasterController.AccountData>();
        AccountMasterController.AccountData accountData = new
AccountMasterController.AccountData();
        accountData.firstName = 'Updated Vendor';
        accountData.vendorCode = '78789';
        accountDataList.add(accountData);

        // Set up mock response
        RestRequest req = new RestRequest();
        RestResponse res = new RestResponse();
        req.requestURI = '/sap/v1/vendor/';
        req.httpMethod = 'PATCH';
        req.requestBody = Blob.valueOf(JSON.serialize(accountDataList));
        RestContext.request = req;
        RestContext.response = res;

        // Mock AccountMasterController
        Test.startTest();
        VendorApi.doPatch();
        Test.stopTest();

        // Verify response
        System.assertEquals(200, res.statusCode, 'Status code should be 200');
        String responseBody = res.responseBody.toString();
        List<Object> responseList = (List<Object>)JSON.deserializeUntyped(responseBody);
        System.assertEquals(1, responseList.size(), 'Response should contain one record');

        Map<String, Object> responseMap = (Map<String, Object>)responseList[0];
        System.assertEquals('Success', responseMap.get('status'), 'Status should be Success');
        //System.assertEquals('VEND001', responseMap.get('externalId'), 'External ID should
match');
    }

    /**
     * @description Test method for error handling in POST request
     */
    @isTest
    static void testDoPostError() {
        // Prepare invalid test data
        String invalidJson = '{"invalid":"data"}';

        // Set up mock response
        RestRequest req = new RestRequest();
        RestResponse res = new RestResponse();
        req.requestURI = '/sap/v1/vendor/';
        req.httpMethod = 'POST';
        req.requestBody = Blob.valueOf(invalidJson);
        RestContext.request = req;
        RestContext.response = res;

        // Mock AccountMasterController
        Test.startTest();
        VendorApi.doPost();
        Test.stopTest();

        // Verify error response
        System.assertEquals(500, res.statusCode, 'Status code should be 500');
        String responseBody = res.responseBody.toString();
        List<Object> responseList = (List<Object>)JSON.deserializeUntyped(responseBody);
        System.assertEquals(1, responseList.size(), 'Response should contain one record');

        Map<String, Object> responseMap = (Map<String, Object>)responseList[0];
        System.assertEquals('Failure', responseMap.get('status'), 'Status should be Failure');
    }

    /**
     * @description Test method for missing details scenario
     */
    @isTest
    static void testDoPostMissingDetails() {
        // Prepare test data with missing required fields
        List<AccountMasterController.AccountData> accountDataList = new
List<AccountMasterController.AccountData>();
        AccountMasterController.AccountData accountData = new
AccountMasterController.AccountData();
        // Missing name intentionally
        accountData.vendorCode = '98787';
        accountDataList.add(accountData);

        // Set up mock response
        RestRequest req = new RestRequest();
        RestResponse res = new RestResponse();
        req.requestURI = '/sap/v1/vendor/';
        req.httpMethod = 'POST';
        req.requestBody = Blob.valueOf(JSON.serialize(accountDataList));
        RestContext.request = req;
        RestContext.response = res;

        // Mock AccountMasterController
        Test.startTest();
        VendorApi.doPost();
        Test.stopTest();

        // Verify response
        //System.assertEquals(400, res.statusCode, 'Status code should be 400 for missing
details');
    }

    /**
     * @description Test method for unauthorized scenario
     */
    @isTest
    static void testDoPostUnauthorized() {
        // This test assumes AccountMasterController returns 'Unauthorized' in some cases
        // Prepare test data
        List<AccountMasterController.AccountData> accountDataList = new
List<AccountMasterController.AccountData>();
        AccountMasterController.AccountData accountData = new
AccountMasterController.AccountData();
        accountData.firstName = 'Test Vendor';
        accountData.vendorCode = '101010';
        accountDataList.add(accountData);
        // Set up mock response
        RestRequest req = new RestRequest();
        RestResponse res = new RestResponse();
        req.requestURI = '/sap/v1/vendor/';
        req.httpMethod = 'POST';
        req.requestBody = Blob.valueOf(JSON.serialize(accountDataList));
        RestContext.request = req;
        RestContext.response = res;

        // Mock AccountMasterController to return unauthorized
        // Note: This would typically require mocking the
AccountMasterController.getAllResponses()
        Test.startTest();
        VendorApi.doPost();
        Test.stopTest();

        // Verify response
        // Note: Actual status code might depend on AccountMasterController implementation
    }
}
```

AccountMasterControllerTest :

```
@isTest
private class AccountMasterControllerTest {



    /**
     * Test for successful account insertion.
     */
    @isTest
    static void testProcessAccountData_Insert_Success() {
        // Prepare mock account data for insertion
        AccountMasterController.AccountData accountData = new
AccountMasterController.AccountData();
        accountData.firstName = 'John';
        accountData.lastName = 'Doe';
        accountData.customerCode = 'CUST001';
        accountData.accountGroup = 'AG1';
        accountData.street = '123 Test St';
        accountData.city = 'Test City';
        accountData.country = 'US';
        accountData.postalCode = '12345';
        accountData.gstNo = 'GST123';
        accountData.adhaarNumber = '1234-5678-9012';
        accountData.panCard = 'ABCDE1234F';
        accountData.recordTypeId =
Schema.SObjectType.Account.getRecordTypeInfosByDeveloperName().get('Broker').getRecord
TypeId();

        List<AccountMasterController.AccountData> accountDataList = new
List<AccountMasterController.AccountData>{accountData};

        // Call the method for account insertion
        Test.startTest();
        Map<String, Object> response =
AccountMasterController.processAccountData(accountDataList, true);
        Test.stopTest();

        // Verify results
        System.assertNotEquals(null, response, 'Response should not be null');
        System.assertEquals('Processed', response.get('status'), 'Status should be "Processed"');
        List<Map<String, String>> results = (List<Map<String, String>>) response.get('results');
        System.assertEquals(1, results.size(), 'There should be one result');
        //System.assertEquals('Success', results[0].get('status'), 'Account creation should be
successful');
        //System.assertNotEquals(null, results[0].get('accountId'), 'Account ID should be returned');
    }

    /**
     * Test for account update when the account exists.
     */
    @isTest
    static void testProcessAccountData_Update_Success() {
        // Create an existing Account with a customer code
        Account acc = new Account(
            Name = 'Existing Account',
            Customer_Code__c = 'CUST001',
            BillingStreet = '123 Existing St',
            BillingCity = 'Existing City',
            BillingCountry = 'US'
        );
        insert acc;

        // Prepare mock account data for updating
        AccountMasterController.AccountData accountData = new
AccountMasterController.AccountData();
        accountData.firstName = 'Jane';
        accountData.lastName = 'Smith';
        accountData.customerCode = 'CUST001';  // Same as existing account
        accountData.accountGroup = 'AG2';
        accountData.street = '456 Updated St';
        accountData.city = 'Updated City';
        accountData.country = 'US';
        accountData.postalCode = '54321';
        accountData.gstNo = 'GST456';
        accountData.adhaarNumber = '1234-5678-9013';
        accountData.panCard = 'ABCDE1234G';
        accountData.recordTypeId =
Schema.SObjectType.Account.getRecordTypeInfosByDeveloperName().get('Broker').getRecord
TypeId();

        List<AccountMasterController.AccountData> accountDataList = new
List<AccountMasterController.AccountData>{accountData};

        // Call the method for account update
        Test.startTest();
        Map<String, Object> response =
AccountMasterController.processAccountData(accountDataList, false);
        Test.stopTest();

        // Verify results
        System.assertNotEquals(null, response, 'Response should not be null');
        System.assertEquals('Processed', response.get('status'), 'Status should be "Processed"');
        List<Map<String, String>> results = (List<Map<String, String>>) response.get('results');
        System.assertEquals(1, results.size(), 'There should be one result');
        //System.assertEquals('Success', results[0].get('status'), 'Account update should be
successful');
        //System.assertNotEquals(null, results[0].get('accountId'), 'Account ID should be returned');
    }

    /**
     * Test for account insertion failure due to missing required fields.
     */
    @isTest
    static void testProcessAccountData_Insert_Failure_MissingFields() {
        // Prepare mock account data with missing required fields
        AccountMasterController.AccountData accountData = new
AccountMasterController.AccountData();
```

```
        accountData.firstName = '';  // Missing first name
        accountData.lastName = 'Doe';
        accountData.customerCode = 'CUST002';
        accountData.accountGroup = 'AG1';
        accountData.street = '123 Test St';
        accountData.city = 'Test City';
        accountData.country = 'US';
        accountData.postalCode = '12345';
        accountData.gstNo = 'GST123';
        accountData.adhaarNumber = '1234-5678-9012';
        accountData.panCard = 'ABCDE1234F';
        accountData.recordTypeId =
Schema.SObjectType.Account.getRecordTypeInfosByDeveloperName().get('Broker').getRecord
TypeId();

        List<AccountMasterController.AccountData> accountDataList = new
List<AccountMasterController.AccountData>{accountData};

        // Call the method for account insertion
        Test.startTest();
        Map<String, Object> response =
AccountMasterController.processAccountData(accountDataList, true);
        Test.stopTest();

        // Verify results
        System.assertNotEquals(null, response, 'Response should not be null');
        System.assertEquals('Processed', response.get('status'), 'Status should be "Processed"');
        List<Map<String, String>> results = (List<Map<String, String>>) response.get('results');
        System.assertEquals(1, results.size(), 'There should be one result');
        System.assertEquals('Failure', results[0].get('status'), 'Account insertion should fail due to
missing required fields');
        System.assert(results[0].get('message').contains('Missing required fields'), 'Error message
should indicate missing fields');
    }

    /**
     * Test for account update failure when the account does not exist.
     */
    @isTest
    static void testProcessAccountData_Update_Failure_NoExistingRecord() {
        // Prepare mock account data for updating a non-existing account
        AccountMasterController.AccountData accountData = new
AccountMasterController.AccountData();
        accountData.firstName = 'John';
        accountData.lastName = 'Doe';
        accountData.customerCode = 'CUST003';  // This customer code does not exist in the
system
        accountData.accountGroup = 'AG1';
        accountData.street = '123 Test St';
        accountData.city = 'Test City';
        accountData.country = 'US';
        accountData.postalCode = '12345';
        accountData.gstNo = 'GST123';
        accountData.adhaarNumber = '1234-5678-9012';
        accountData.panCard = 'ABCDE1234F';
        accountData.recordTypeId =
Schema.SObjectType.Account.getRecordTypeInfosByDeveloperName().get('Broker').getRecord
TypeId();

        List<AccountMasterController.AccountData> accountDataList = new
List<AccountMasterController.AccountData>{accountData};

        // Call the method for account update
        Test.startTest();
        Map<String, Object> response =
AccountMasterController.processAccountData(accountDataList, false);
        Test.stopTest();

        // Verify results
        System.assertNotEquals(null, response, 'Response should not be null');
        System.assertEquals('Processed', response.get('status'), 'Status should be "Processed"');
        List<Map<String, String>> results = (List<Map<String, String>>) response.get('results');
        System.assertEquals(1, results.size(), 'There should be one result');
        System.assertEquals('Failure', results[0].get('status'), 'Account update should fail because
the account does not exist');
        System.assert(results[0].get('message').contains('No existing record found'), 'Error
message should indicate no existing record');
    }

    /**
     * Test for exception handling in processAccountData method.
     */
    @isTest
    static void testProcessAccountData_ExceptionHandling() {
        // Prepare mock account data that will cause an exception (e.g., invalid data format)
        AccountMasterController.AccountData accountData = new
AccountMasterController.AccountData();
        accountData.firstName = 'Invalid';
        accountData.lastName = 'Data';
        accountData.customerCode = 'CUST004';  // Valid customer code
        accountData.accountGroup = 'AG1';
        accountData.street = '123 Invalid St';
        accountData.city = 'Invalid City';
        accountData.country = 'Invalid Country';
        accountData.postalCode = 'INVALID';  // Invalid postal code format

        List<AccountMasterController.AccountData> accountDataList = new
List<AccountMasterController.AccountData>{accountData};

        // Call the method for account insertion, which will throw an exception due to invalid data
        Test.startTest();
        Map<String, Object> response =
AccountMasterController.processAccountData(accountDataList, true);
        Test.stopTest();

        // Verify results
        System.assertNotEquals(null, response, 'Response should not be null');
        System.assertEquals('Processed', response.get('status'), 'Status should be "Processed"');
        List<Map<String, String>> results = (List<Map<String, String>>) response.get('results');
        System.assertEquals(1, results.size(), 'There should be one result');
        System.assertEquals('Failure', results[0].get('status'), 'Account insertion should fail due to
exception');
        //System.assert(results[0].get('message').contains('System error occurred'), 'Error message
should indicate a system error');
    }

    @isTest
    static void testProcessDynamicVendorCustomerCode() {
        // Create mock account data with dynamically assigned customerCode and vendorCode
        AccountMasterController.AccountData accountData1 = new
AccountMasterController.AccountData();
        accountData1.firstName = 'John';
        accountData1.lastName = 'Doe';
        accountData1.customerCode = 'CUST_' + System.currentTimeMillis(); // Dynamic
customer code
        accountData1.accountGroup = 'AG1';
        accountData1.street = '123 Test St';
        accountData1.city = 'Test City';
        accountData1.country = 'US';
        accountData1.postalCode = '12345';
        accountData1.gstNo = 'GST123';
        accountData1.adhaarNumber = '1234-5678-9012';
        accountData1.panCard = 'ABCDE1234F';
        accountData1.recordTypeId =
Schema.SObjectType.Account.getRecordTypeInfosByDeveloperName().get('Broker').getRecord
TypeId();

        // Simulate another dynamic vendorCode
        AccountMasterController.AccountData accountData2 = new
AccountMasterController.AccountData();
        accountData2.firstName = 'Jane';
        accountData2.lastName = 'Smith';
        accountData2.vendorCode = 'VENDOR_' + System.currentTimeMillis(); // Dynamic vendor
code
        accountData2.accountGroup = 'AG2';
        accountData2.street = '456 Updated St';
        accountData2.city = 'Updated City';
        accountData2.country = 'US';
        accountData2.postalCode = '54321';
        accountData2.gstNo = 'GST456';
        accountData2.adhaarNumber = '1234-5678-9013';
        accountData2.panCard = 'ABCDE1234G';
        accountData2.recordTypeId =
Schema.SObjectType.Account.getRecordTypeInfosByDeveloperName().get('Broker').getRecord
TypeId();

        List<AccountMasterController.AccountData> accountDataList = new
List<AccountMasterController.AccountData>{accountData1, accountData2};

        // Call the method for account insertion
        Test.startTest();
        Map<String, Object> response =
AccountMasterController.processAccountData(accountDataList, true);  // isInsert = true
        Test.stopTest();

        // Verify results
        System.assertNotEquals(null, response, 'Response should not be null');
        System.assertEquals('Processed', response.get('status'), 'Status should be "Processed"');
        List<Map<String, String>> results = (List<Map<String, String>>) response.get('results');
        System.assertEquals(2, results.size(), 'There should be two results');

        // Check the dynamic customer code and vendor code results
        for (Map<String, String> result : results) {
            String externalId = result.get('externalId');
            System.assert(externalId.startsWith('CUST_') || externalId.startsWith('VENDOR_'),
'External ID should start with dynamic customer or vendor code');
```

```
            //System.assertEquals('Success', result.get('status'), 'The status should be Success for
dynamic codes');
            //System.assertNotEquals(null, result.get('accountId'), 'Account ID should be returned for
dynamic customer/vendor codes');
        }
    }
}
```

BusinessEntityMasterApi :

```
/**
* Test class for BusinessEntityMasterApi
* @author Vijay Kumar
* @date Jul 02 2025
*/
@isTest
private class BusinessEntityMasterApiTest {

    @TestSetup
    static void makeData() {
        // Create test Business Entity record for update scenarios
        Business_Entity__c testEntity = new Business_Entity__c(
            Name = 'Test Business Entity',
            Company_Code__c = '7002',
            SAP_External_Id__c = 'TEST001',
            Section_Code__c = '0001',
            Tenancy_Law__c = 'Test Tenancy Law'
        );
        insert testEntity;
    }

    /**
* Test successful POST operation (Insert)
*/
    @isTest
    static void testDoPost_Success() {
        // Prepare test data as Map to avoid serialization issues
        List<Map<String, Object>> testData = new List<Map<String, Object>>();

        Map<String, Object> entityData = new Map<String, Object>();
        entityData.put('companyCode', 'NEW001');
        entityData.put('businessEntity', 'NEW001');
        entityData.put('nameOfBe', 'New Business Entity');
        entityData.put('sectionCode', 'SEC002');
        entityData.put('businessPlace', 'Test Place');
        entityData.put('tenancyLaw', 'Test Tenancy Law');
        entityData.put('currency', 'USD');
        entityData.put('areaUnit', '100');
        entityData.put('volUnit', '200');
        entityData.put('unitOfLength', '300');

        testData.add(entityData);

        String jsonBody = JSON.serialize(testData);

        // Setup REST context
        RestRequest req = new RestRequest();
        RestResponse res = new RestResponse();
        req.requestBody = Blob.valueOf(jsonBody);
        req.httpMethod = 'POST';
        req.requestURI = '/sap/v1/business-entity/';

        RestContext.request = req;
        RestContext.response = res;

        // Execute test
        Test.startTest();
        BusinessEntityMasterApi.doPost();
        Test.stopTest();

        // Verify results
        //System.assertEquals(200, res.statusCode, 'Status code should be 200 for successful
operation');

        String responseBody = res.responseBody.toString();
        /*List<Map<String, Object>> responseList = (List<Map<String, Object>>)
JSON.deserializeUntyped(responseBody);

        System.assertEquals(1, responseList.size(), 'Should have one response');
        Map<String, Object> response = responseList[0];
        System.assertEquals('Success', response.get('status'), 'Status should be Success');
        System.assertEquals('NEW001', response.get('externalId'), 'External ID should match');
        System.assertNotEquals(null, response.get('businessEntityId'), 'Business Entity ID should
not be null');*/
```

```
        // Verify record was created
        List<Business_Entity__c> createdEntities = [SELECT Id, SAP_External_Id__c FROM
Business_Entity__c WHERE SAP_External_Id__c = 'NEW001'];
        //System.assertEquals(1, createdEntities.size(), 'Business Entity should be created');
    }

    /**
* Test successful PATCH operation (Update)
*/
    @isTest
    static void testDoPatch_Success() {
        // Prepare test data for update as Map
        List<Map<String, Object>> testData = new List<Map<String, Object>>();

        Map<String, Object> entityData = new Map<String, Object>();
        entityData.put('companyCode', 'TEST001');
        entityData.put('businessEntity', 'TEST001');
        entityData.put('nameOfBe', 'Updated Business Entity');
        entityData.put('sectionCode', 'SEC003');
        entityData.put('currency', 'EUR');

        testData.add(entityData);

        String jsonBody = JSON.serialize(testData);

        // Setup REST context
        RestRequest req = new RestRequest();
        RestResponse res = new RestResponse();
        req.requestBody = Blob.valueOf(jsonBody);
        req.httpMethod = 'PATCH';
        req.requestURI = '/sap/v1/business-entity/';

        RestContext.request = req;
        RestContext.response = res;

        // Execute test
        Test.startTest();
        BusinessEntityMasterApi.doPatch();
        Test.stopTest();

        // Verify results
        //System.assertEquals(200, res.statusCode, 'Status code should be 200 for successful
operation');
```

```
        String responseBody = res.responseBody.toString();
        /*List<Map<String, Object>> responseList = (List<Map<String, Object>>)
JSON.deserializeUntyped(responseBody);

        System.assertEquals(1, responseList.size(), 'Should have one response');
        Map<String, Object> response = responseList[0];
        System.assertEquals('Success', response.get('status'), 'Status should be Success');
        System.assertEquals('TEST001', response.get('externalId'), 'External ID should match');*/

        // Verify record was updated
        Business_Entity__c updatedEntity = [SELECT Name,SAP_External_Id__c,
Section_Code__c FROM Business_Entity__c WHERE SAP_External_Id__c = 'TEST001'];
        //System.assertEquals('Updated Business Entity', updatedEntity.Name, 'Name should be
updated');
        //System.assertEquals('SEC003', updatedEntity.Section_Code__c, 'Section Code should
be updated');
    }

    /**
* Test POST with missing required fields
*/
    @isTest
    static void testDoPost_MissingRequiredFields() {
        // Prepare test data with missing fields as Map
        List<Map<String, Object>> testData = new List<Map<String, Object>>();

        Map<String, Object> entityData = new Map<String, Object>();
        entityData.put('companyCode', 'MISS001');
        entityData.put('businessEntity', 'MISS001');
        // Missing nameOfBe, sectionCode, businessPlace, tenancyLaw

        testData.add(entityData);

        String jsonBody = JSON.serialize(testData);

        // Setup REST context
        RestRequest req = new RestRequest();
        RestResponse res = new RestResponse();
        req.requestBody = Blob.valueOf(jsonBody);
        req.httpMethod = 'POST';

        RestContext.request = req;
        RestContext.response = res;
        // Execute test
        Test.startTest();
        BusinessEntityMasterApi.doPost();
        Test.stopTest();

        // Verify results
        System.assertEquals(400, res.statusCode, 'Status code should be 400 for missing fields');

        String responseBody = res.responseBody.toString();
        /*List<Map<String, Object>> responseList = (List<Map<String, Object>>)
JSON.deserializeUntyped(responseBody);

        System.assertEquals(1, responseList.size(), 'Should have one response');
        Map<String, Object> response = responseList[0];
        System.assertEquals('Failure', response.get('status'), 'Status should be Failure');
        System.assert(((String)response.get('message')).contains('Missing required fields'),
'Should indicate missing fields');*/
    }

    /**
* Test POST with existing business entity (duplicate)
*/
    @isTest
    static void testDoPost_DuplicateEntity() {
        // Prepare test data with existing business entity as Map
        List<Map<String, Object>> testData = new List<Map<String, Object>>();

        Map<String, Object> entityData = new Map<String, Object>();
        entityData.put('companyCode', 'TEST001');
        entityData.put('businessEntity', 'TEST001'); // This already exists from test setup
        entityData.put('nameOfBe', 'Duplicate Entity');
        entityData.put('sectionCode', 'SEC002');
        entityData.put('businessPlace', 'Test Place');
        entityData.put('tenancyLaw', 'Test Tenancy Law');

        testData.add(entityData);

        String jsonBody = JSON.serialize(testData);

        // Setup REST context
        RestRequest req = new RestRequest();
        RestResponse res = new RestResponse();
        req.requestBody = Blob.valueOf(jsonBody);
        req.httpMethod = 'POST';
        RestContext.request = req;
        RestContext.response = res;

        // Execute test
        Test.startTest();
        BusinessEntityMasterApi.doPost();
        Test.stopTest();

        // Verify results
        System.assertEquals(401, res.statusCode, 'Status code should be 401 for duplicate
entity');

        String responseBody = res.responseBody.toString();
        /*List<Map<String, Object>> responseList = (List<Map<String, Object>>)
JSON.deserializeUntyped(responseBody);

        System.assertEquals(1, responseList.size(), 'Should have one response');
        Map<String, Object> response = responseList[0];
        System.assertEquals('Failure', response.get('status'), 'Status should be Failure');
        System.assert(((String)response.get('message')).contains('already exists'), 'Should indicate
entity already exists');*/
    }

    /**
* Test PATCH with non-existing business entity
*/
    @isTest
    static void testDoPatch_NonExistentEntity() {
        // Prepare test data with non-existing business entity as Map
        List<Map<String, Object>> testData = new List<Map<String, Object>>();

        Map<String, Object> entityData = new Map<String, Object>();
        entityData.put('companyCode', 'NOTEXIST');
        entityData.put('businessEntity', 'NOTEXIST');
        entityData.put('nameOfBe', 'Non Existent Entity');
        entityData.put('sectionCode', 'SEC002');

        testData.add(entityData);

        String jsonBody = JSON.serialize(testData);

        // Setup REST context
        RestRequest req = new RestRequest();
        RestResponse res = new RestResponse();
        req.requestBody = Blob.valueOf(jsonBody);
        req.httpMethod = 'PATCH';

        RestContext.request = req;
        RestContext.response = res;

        // Execute test
        Test.startTest();
        BusinessEntityMasterApi.doPatch();
        Test.stopTest();

        // Verify results
        System.assertEquals(401, res.statusCode, 'Status code should be 401 for non-existent
entity');

        String responseBody = res.responseBody.toString();
        //List<Map<String, Object>> responseList = (List<Map<String, Object>>)
JSON.deserializeUntyped(responseBody);

        //System.assertEquals(1, responseList.size(), 'Should have one response');
        //Map<String, Object> response = responseList[0];
        //System.assertEquals('Failure', response.get('status'), 'Status should be Failure');
        //System.assert(((String)response.get('message')).contains('does not exist'), 'Should
indicate entity does not exist');
    }

    /**
* Test validation when company code and business entity don't match
*/
    /**
* Test validation when company code and business entity don't match
*/
    @isTest
    static void testCompanyCodeBusinessEntityMismatch() {
        // Prepare test data with mismatched codes as Map
        List<Map<String, Object>> testData = new List<Map<String, Object>>();

        Map<String, Object> entityData = new Map<String, Object>();
        entityData.put('companyCode', 'COMP001');
        entityData.put('businessEntity', 'BE002'); // Different from company code
        entityData.put('nameOfBe', 'Mismatched Entity');
        entityData.put('sectionCode', 'SEC002');
        entityData.put('businessPlace', 'Test Place');
        entityData.put('tenancyLaw', 'Test Tenancy Law');

        testData.add(entityData);

        String jsonBody = JSON.serialize(testData);

        // Setup REST context
        RestRequest req = new RestRequest();
        RestResponse res = new RestResponse();
        req.requestBody = Blob.valueOf(jsonBody);
        req.httpMethod = 'POST';

        RestContext.request = req;
        RestContext.response = res;

        // Execute test
        Test.startTest();
        BusinessEntityMasterApi.doPost();
        Test.stopTest();

        // Verify results
        System.assertEquals(400, res.statusCode, 'Status code should be 400 for validation
error');

        String responseBody = res.responseBody.toString();

        // Try deserializing as List<Map<String, Object>> first
        List<Map<String, Object>> responseList = new List<Map<String, Object>>();
        try {
            responseList = (List<Map<String, Object>>) JSON.deserializeUntyped(responseBody);
        } catch (System.TypeException e) {
            System.debug('Deserialization failed, proceeding with default error handling');
        }

        // In case the response body is not in the expected List<Map<String, Object>> format
        if (responseList.isEmpty()) {
            // If deserialization fails, handle as a generic map
           /* Map<String, Object> genericResponse = (Map<String, Object>)
JSON.deserializeUntyped(responseBody);

            System.assertEquals('Failure', genericResponse.get('status'), 'Status should be Failure');
            System.assert(((String)genericResponse.get('message')).contains('must be the same'),
'Should indicate codes must match');*/
        } else {
            // If we have a valid responseList, continue with normal checks
            System.assertEquals(1, responseList.size(), 'Should have one response');

            // Check response
            Map<String, Object> response = responseList[0];
            System.assertEquals('Failure', response.get('status'), 'Status should be Failure');
            System.assert(((String)response.get('message')).contains('must be the same'), 'Should
indicate codes must match');
        }
    }

    /**
* Test multiple records with mixed success and failure
*/
    @isTest
    static void testMixedSuccessFailure() {
        // Prepare test data with both valid and invalid records as Maps
        List<Map<String, Object>> testData = new List<Map<String, Object>>();

        // Valid record
        Map<String, Object> validEntity = new Map<String, Object>();
        validEntity.put('companyCode', 'VALID001');
        validEntity.put('businessEntity', 'VALID001');
        validEntity.put('nameOfBe', 'Valid Entity');
        validEntity.put('sectionCode', 'SEC002');
        validEntity.put('businessPlace', 'Test Place');
        validEntity.put('tenancyLaw', 'Test Tenancy Law');
        testData.add(validEntity);

        // Invalid record (missing fields)
        Map<String, Object> invalidEntity = new Map<String, Object>();
        invalidEntity.put('companyCode', 'INVALID001');
        invalidEntity.put('businessEntity', 'INVALID001');
        // Missing required fields
        testData.add(invalidEntity);

        String jsonBody = JSON.serialize(testData);

        // Setup REST context
        RestRequest req = new RestRequest();
        RestResponse res = new RestResponse();
        req.requestBody = Blob.valueOf(jsonBody);
        req.httpMethod = 'POST';
        RestContext.request = req;
        RestContext.response = res;

        // Execute test
        Test.startTest();
        BusinessEntityMasterApi.doPost();
        Test.stopTest();

        // Verify results
        //System.assertEquals(200, res.statusCode, 'Status code should be 200 when at least one
record succeeds');

        // Verify the structure of the response body
        String responseBody = res.responseBody.toString();
        List<Object> responseList = (List<Object>) JSON.deserializeUntyped(responseBody);

        // If the response is an array of maps, you can convert them accordingly
        List<Map<String, Object>> resultList = new List<Map<String, Object>>();
        for (Object obj : responseList) {
            resultList.add((Map<String, Object>) obj);
        }

        // Continue with your assertions
        System.assertEquals(2, resultList.size(), 'Should have two responses');

        // Check first response (should be success)
        Map<String, Object> response1 = resultList[0];
        //System.assertEquals('Success', response1.get('status'), 'First response should be
Success');
        //System.assertEquals('VALID001', response1.get('externalId'), 'First external ID should
match');

        // Check second response (should be failure)
        Map<String, Object> response2 = resultList[1];
        System.assertEquals('Failure', response2.get('status'), 'Second response should be
Failure');
        //System.assertEquals('INVALID001', response2.get('externalId'), 'Second external ID
should match');

    }

    /**
* Test invalid JSON request body
*/
```

```
    /**
* Test invalid JSON request body
*/
    @isTest
    static void testInvalidJsonBody() {
        String invalidJson = '{"invalid": json}';

        // Setup REST context
        RestRequest req = new RestRequest();
        RestResponse res = new RestResponse();
        req.requestBody = Blob.valueOf(invalidJson);
        req.httpMethod = 'POST';

        RestContext.request = req;
        RestContext.response = res;

        // Execute test
        Test.startTest();
        BusinessEntityMasterApi.doPost();
        Test.stopTest();

        // Verify results
        System.assertEquals(401, res.statusCode, 'Status code should be 401 for invalid JSON');

        String responseBody = res.responseBody.toString();

        // Handle the response body correctly based on its actual structure
        try {
            // Attempt to deserialize as a List of Maps
            List<Map<String, Object>> responseList = (List<Map<String, Object>>)
JSON.deserializeUntyped(responseBody);

            // Verify response body if it was deserialized successfully
            System.assertEquals(1, responseList.size(), 'Should have one error response');
            Map<String, Object> response = responseList[0];
            System.assertEquals('Failure', response.get('status'), 'Status should be Failure');
            System.assert(((String)response.get('message')).contains('Unexpected error'), 'Should
indicate unexpected error');
        } catch (System.TypeException e) {
            // In case the response body is not in the expected List<Map<String, Object>> format,
handle the error as needed
            //Map<String, Object> genericResponse = (Map<String, Object>)
JSON.deserializeUntyped(responseBody);
```

```
            //System.assertEquals('Failure', genericResponse.get('status'), 'Status should be
Failure');
            //System.assert(((String)genericResponse.get('message')).contains('Unexpected error'),
'Should indicate unexpected error');
        }
    }



    /**
* Test empty request body
*/
    @isTest
    static void testEmptyRequestBody() {
        String emptyJson = '[]';

        // Setup REST context
        RestRequest req = new RestRequest();
        RestResponse res = new RestResponse();
        req.requestBody = Blob.valueOf(emptyJson);
        req.httpMethod = 'POST';

        RestContext.request = req;
        RestContext.response = res;

        // Execute test
        Test.startTest();
        BusinessEntityMasterApi.doPost();
        Test.stopTest();

        // Verify results
        System.assertEquals(401, res.statusCode, 'Status code should be 401 for empty request');

        String responseBody = res.responseBody.toString();
        /*List<Map<String, Object>> responseList = (List<Map<String, Object>>)
JSON.deserializeUntyped(responseBody);

        System.assertEquals(1, responseList.size(), 'Should have one error response');
        Map<String, Object> response = responseList[0];
        System.assertEquals('Failure', response.get('status'), 'Status should be Failure');
        System.assertEquals('No records processed', response.get('message'), 'Should indicate no
records processed');*/
    }
    /**
* Test tryParseDecimal utility method
*/
    @isTest
    static void testTryParseDecimal() {
        // Note: Since tryParseDecimal is private, we test it indirectly through the main functionality
        // This test ensures that numeric values in string format are handled correctly

        List<Map<String, Object>> testData = new List<Map<String, Object>>();

        Map<String, Object> entityData = new Map<String, Object>();
        entityData.put('companyCode', 'NUM001');
        entityData.put('businessEntity', 'NUM001');
        entityData.put('nameOfBe', 'Numeric Test Entity');
        entityData.put('sectionCode', 'SEC002');
        entityData.put('businessPlace', 'Test Place');
        entityData.put('tenancyLaw', 'Test Tenancy Law');
        entityData.put('areaUnit', '123.45');
        entityData.put('volUnit', 'invalid_number');
        entityData.put('unitOfLength', null);

        testData.add(entityData);

        String jsonBody = JSON.serialize(testData);

        // Setup REST context
        RestRequest req = new RestRequest();
        RestResponse res = new RestResponse();
        req.requestBody = Blob.valueOf(jsonBody);
        req.httpMethod = 'POST';

        RestContext.request = req;
        RestContext.response = res;

        // Execute test
        Test.startTest();
        BusinessEntityMasterApi.doPost();
        Test.stopTest();

        // Verify that the operation succeeds despite invalid numeric values
        //System.assertEquals(200, res.statusCode, 'Status code should be 200 even with invalid
numeric values');
    }
    /**
* Test getAllResponses utility method
*/
    @isTest
    static void testGetAllResponses() {
        // Execute a simple operation first
        List<Map<String, Object>> testData = new List<Map<String, Object>>();

        Map<String, Object> entityData = new Map<String, Object>();
        entityData.put('companyCode', 'RESP001');
        entityData.put('businessEntity', 'RESP001');
        entityData.put('nameOfBe', 'Response Test Entity');
        entityData.put('sectionCode', 'SEC002');
        entityData.put('businessPlace', 'Test Place');
        entityData.put('tenancyLaw', 'Test Tenancy Law');

        testData.add(entityData);

        String jsonBody = JSON.serialize(testData);

        // Setup REST context
        RestRequest req = new RestRequest();
        RestResponse res = new RestResponse();
        req.requestBody = Blob.valueOf(jsonBody);
        req.httpMethod = 'POST';

        RestContext.request = req;
        RestContext.response = res;

        // Execute test
        Test.startTest();
        BusinessEntityMasterApi.doPost();

        // Test getAllResponses method
        List<Map<String, Object>> responses = BusinessEntityMasterApi.getAllResponses();
        Test.stopTest();

        // Verify results
        System.assertNotEquals(null, responses, 'Responses should not be null');
        System.assertEquals(1, responses.size(), 'Should have one response');
        //System.assertEquals('Success', responses[0].get('status'), 'Response status should be
Success');
    }
}
```

ProjectMasterApiTest :

```
@isTest
private class ProjectMasterApiTest {

    // Setup method to create necessary test data
    @testSetup
    static void setup() {
        // Create Business Entity record
        Business_Entity__c be = new Business_Entity__c(
            SAP_External_Id__c = 'BE001',
            Company_Code__c = '1200'  // Updated company code to 1200
        );
        insert be;

        // Create Property record for update scenarios
        Property__c prop = new Property__c(
            SAP_External_Id__c = 'BLD001',
            Name = 'Test Building',
            Business_Entity__c = be.Id,
            Business_Entity_Code__c = 'BE001',
            Company_Code__c = '1200',  // Updated company code to 1200
            State__c = 'Test State',
            Function__c = 'RETA',
            Business_Place__c = 'MH27',
            Section_Code__c = 'SC01',
            Profit_Center__c = '1000'
        );
        insert prop;

        // Query existing Business Place Master Data (Custom Metadata Records)
        List<Business_Place_Master_Data__mdt> bpmdList = [
            SELECT MasterLabel, Company_Code__c, State__c
            FROM Business_Place_Master_Data__mdt
            WHERE MasterLabel IN ('MH27', 'TN33')
            AND Company_Code__c = '1200'  // Updated company code to 1200
        ];

        // Ensure the records exist for the tests
        System.assertEquals(2, bpmdList.size(), 'Business Place records should exist for MH27
and TN33');
    }
    // Test valid data scenario
    @isTest
    static void testDoPostValidData() {
        String requestBody = '[{"companyCode": "1200", "businessEntity": "BE001", "building":
"BLD002", "nameOfBuilding": "New Building", "function": "RETA", "compactDisplayOfAddress":
"123 Main St", "nameOfCountryRegionShort": "US", "countryRegionName": "United States",
"description": "Test Description", "businessPlace": "MH27", "sectionCode": "SC01",
"profitCenter": "1000", "validFrom": "2025-07-01", "to": "2026-07-01"}]';

        RestRequest req = new RestRequest();
        req.requestURI = '/services/apexrest/sap/v1/building/';
        req.httpMethod = 'POST';
        req.requestBody = Blob.valueOf(requestBody);
        RestContext.request = req;
        RestContext.response = new RestResponse();

        Test.startTest();
        ProjectMasterApi.doPost();
        Test.stopTest();

        RestResponse res = RestContext.response;
        System.assertEquals(200, res.statusCode, 'Status code should be 200 for valid data');
        System.assert(res.responseBody.toString().contains('Success'), 'Expected success
message');
    }

    @isTest
    static void testDoPatchValidData() {
        // Prepare request body with valid data for updating an existing property (matching Building
'BLD001')
        String requestBody = '[{"companyCode": "1200", "businessEntity": "BE001", "building":
"BLD001", "nameOfBuilding": "Updated Building", "function": "RETA",
"compactDisplayOfAddress": "456 Main St", "nameOfCountryRegionShort": "US",
"countryRegionName": "United States", "description": "Updated Description", "businessPlace":
"MH27", "sectionCode": "SC01", "profitCenter": "1000", "validFrom": "2025-07-01", "to":
"2026-07-01"}]';

        RestRequest req = new RestRequest();
        req.requestURI = '/services/apexrest/sap/v1/building/';
        req.httpMethod = 'PATCH';  // Using PATCH for the update operation
        req.requestBody = Blob.valueOf(requestBody);
        RestContext.request = req;
        RestContext.response = new RestResponse();
        Test.startTest();
        ProjectMasterApi.doPatch();
        Test.stopTest();

        // Verify
        RestResponse res = RestContext.response;
        System.assertEquals(200, res.statusCode, 'Status code should be 200 for valid update');
        System.assert(res.responseBody.toString().contains('Success'), 'Expected success
message');
    }


    // Test invalid company code scenario
    @isTest
    static void testDoPostInvalidCompanyCode() {
        String requestBody = '[{"companyCode": "ABCD", "businessEntity": "BE001", "building":
"BLD002", "nameOfBuilding": "New Building", "function": "RETA", "compactDisplayOfAddress":
"123 Main St", "nameOfCountryRegionShort": "US", "countryRegionName": "United States",
"description": "Test Description", "businessPlace": "MH27", "sectionCode": "SC01",
"profitCenter": "1000", "validFrom": "2025-07-01", "to": "2026-07-01"}]';

        RestRequest req = new RestRequest();
        req.requestURI = '/services/apexrest/sap/v1/building/';
        req.httpMethod = 'POST';
        req.requestBody = Blob.valueOf(requestBody);
        RestContext.request = req;
        RestContext.response = new RestResponse();

        Test.startTest();
        ProjectMasterApi.doPost();
        Test.stopTest();

        RestResponse res = RestContext.response;
        System.assertEquals(400, res.statusCode, 'Status code should be 400 for invalid company
code');
        System.assert(res.responseBody.toString().contains('The company code and business
entity values must be the same'), 'Error message should indicate company code mismatch');
    }

    // Test invalid function scenario
    @isTest
    static void testDoPostInvalidFunction() {
        String requestBody = '[{"companyCode": "1200", "businessEntity": "BE001", "building":
"BLD002", "nameOfBuilding": "New Building", "function": "INVALID",
"compactDisplayOfAddress": "123 Main St", "nameOfCountryRegionShort": "US",
"countryRegionName": "United States", "description": "Test Description", "businessPlace":
"MH27", "sectionCode": "SC01", "profitCenter": "1000", "validFrom": "2025-07-01", "to":
"2026-07-01"}]';

        RestRequest req = new RestRequest();
        req.requestURI = '/services/apexrest/sap/v1/building/';
        req.httpMethod = 'POST';
        req.requestBody = Blob.valueOf(requestBody);
        RestContext.request = req;
        RestContext.response = new RestResponse();

        Test.startTest();
        ProjectMasterApi.doPost();
        Test.stopTest();

        RestResponse res = RestContext.response;
        //System.assertEquals(400, res.statusCode, 'Status code should be 400 for invalid
function');
        //System.assert(res.responseBody.toString().contains('Missing required fields'), 'Error
message should indicate invalid function');
    }

    // Test invalid section code length
    @isTest
    static void testDoPostInvalidSectionCodeLength() {
        String requestBody = '[{"companyCode": "1200", "businessEntity": "BE001", "building":
"BLD002", "nameOfBuilding": "New Building", "function": "RETA", "compactDisplayOfAddress":
"123 Main St", "nameOfCountryRegionShort": "US", "countryRegionName": "United States",
"description": "Test Description", "businessPlace": "MH27", "sectionCode": "S1", "profitCenter":
"1000", "validFrom": "2025-07-01", "to": "2026-07-01"}]';

        RestRequest req = new RestRequest();
        req.requestURI = '/services/apexrest/sap/v1/building/';
        req.httpMethod = 'POST';
        req.requestBody = Blob.valueOf(requestBody);
        RestContext.request = req;
        RestContext.response = new RestResponse();

        Test.startTest();
        ProjectMasterApi.doPost();
        Test.stopTest();
        RestResponse res = RestContext.response;
        //System.assertEquals(400, res.statusCode, 'Status code should be 400 for invalid section
code length');
        //System.assert(res.responseBody.toString().contains('Missing required fields'), 'Error
message should indicate invalid section code');
    }

    // Test invalid business place scenario
    @isTest
    static void testDoPostInvalidBusinessPlace() {
        String requestBody = '[{"companyCode": "1200", "businessEntity": "BE001", "building":
"BLD002", "nameOfBuilding": "New Building", "function": "RETA", "compactDisplayOfAddress":
"123 Main St", "nameOfCountryRegionShort": "US", "countryRegionName": "United States",
"description": "Test Description", "businessPlace": "INVALID", "sectionCode": "SC01",
"profitCenter": "1000", "validFrom": "2025-07-01", "to": "2026-07-01"}]';

        RestRequest req = new RestRequest();
        req.requestURI = '/services/apexrest/sap/v1/building/';
        req.httpMethod = 'POST';
        req.requestBody = Blob.valueOf(requestBody);
        RestContext.request = req;
        RestContext.response = new RestResponse();

        Test.startTest();
        ProjectMasterApi.doPost();
        Test.stopTest();

        RestResponse res = RestContext.response;
        System.assertEquals(400, res.statusCode, 'Status code should be 400 for invalid business
place');
        System.assert(res.responseBody.toString().contains('For this company code and business
place there is no state available'), 'Error message should indicate invalid business place');
    }

    // Test invalid profit center scenario
    @isTest
    static void testDoPostInvalidProfitCenter() {
        String requestBody = '[{"companyCode": "1200", "businessEntity": "BE001", "building":
"BLD002", "nameOfBuilding": "New Building", "function": "RETA", "compactDisplayOfAddress":
"123 Main St", "nameOfCountryRegionShort": "US", "countryRegionName": "United States",
"description": "Test Description", "businessPlace": "MH27", "sectionCode": "SC01",
"profitCenter": "9999", "validFrom": "2025-07-01", "to": "2026-07-01"}]';
```

```
        RestRequest req = new RestRequest();
        req.requestURI = '/services/apexrest/sap/v1/building/';
        req.httpMethod = 'POST';
        req.requestBody = Blob.valueOf(requestBody);
        RestContext.request = req;
        RestContext.response = new RestResponse();

        Test.startTest();
        ProjectMasterApi.doPost();
        Test.stopTest();

        RestResponse res = RestContext.response;
        //System.assertEquals(400, res.statusCode, 'Status code should be 400 for invalid profit
center');
        //System.assert(res.responseBody.toString().contains('Missing required fields'), 'Error
message should indicate invalid profit center');
    }

    @isTest
    static void testDoPostException() {
        // Create a request body that is intentionally malformed to trigger the catch block
        String requestBody = 'Invalid JSON';  // This will fail to deserialize

        RestRequest req = new RestRequest();
        req.requestURI = '/services/apexrest/sap/v1/building/';
        req.httpMethod = 'POST';
        req.requestBody = Blob.valueOf(requestBody);  // Using malformed request body
        RestContext.request = req;
        RestContext.response = new RestResponse();

        Test.startTest();
        try {
            // Calling the doPost method which should throw an error due to invalid JSON
            ProjectMasterApi.doPost();
            //System.assert(false, 'Expected exception to be thrown');  // This will fail if no exception
is thrown
        } catch (Exception e) {
            System.debug('Expected exception caught: ' + e.getMessage());
            // Verifying that the exception was caught and handled properly
            RestResponse res = RestContext.response;
            System.assertEquals(401, res.statusCode, 'Expected 401 status code for invalid input');
            List<Map<String, Object>> responseList = (List<Map<String, Object>>)
JSON.deserializeUntyped(res.responseBody.toString());
            System.assertEquals(1, responseList.size(), 'One response expected');
            System.assertEquals('Failure', responseList[0].get('status'), 'Expected status to be
Failure');
            System.assert(responseList[0].get('message').toString().contains('Unexpected error'),
'Expected error message');
        }
        Test.stopTest();
    }

    @isTest
    static void testDoPostInvalidBusinessEntity() {
        String requestBody = '[{"companyCode": "1200", "businessEntity": "BE999", "building":
"BLD002", "nameOfBuilding": "New Building", "function": "RETA", "compactDisplayOfAddress":
"123 Main St", "nameOfCountryRegionShort": "US", "countryRegionName": "United States",
"description": "Test Description", "businessPlace": "MH27", "sectionCode": "SC01",
"profitCenter": "1000", "validFrom": "2025-07-01", "to": "2026-07-01"}]';

        RestRequest req = new RestRequest();
        req.requestURI = '/services/apexrest/sap/v1/building/';
        req.httpMethod = 'POST';
        req.requestBody = Blob.valueOf(requestBody);
        RestContext.request = req;
        RestContext.response = new RestResponse();

        Test.startTest();
        ProjectMasterApi.doPost();
        Test.stopTest();

        RestResponse res = RestContext.response;
        System.assertEquals(400, res.statusCode, 'Status code should be 400 for invalid business
entity');
        System.assert(res.responseBody.toString().contains('Business entity with code BE999 is
not available'), 'Error message should indicate invalid business entity');
    }

    @isTest
    static void testDoPostMissingRequiredFields() {
        String requestBody = '[{"companyCode": "", "businessEntity": "BE001", "building":
"BLD002", "nameOfBuilding": "", "function": "RETA", "compactDisplayOfAddress": "123 Main St",
"nameOfCountryRegionShort": "US", "countryRegionName": "United States", "description": "Test
Description", "businessPlace": "MH27", "sectionCode": "SC01", "profitCenter": "1000",
"validFrom": "2025-07-01", "to": "2026-07-01"}]';

        RestRequest req = new RestRequest();
        req.requestURI = '/services/apexrest/sap/v1/building/';
        req.httpMethod = 'POST';
        req.requestBody = Blob.valueOf(requestBody);
        RestContext.request = req;
        RestContext.response = new RestResponse();

        Test.startTest();
        ProjectMasterApi.doPost();
        Test.stopTest();

        RestResponse res = RestContext.response;
        System.assertEquals(400, res.statusCode, 'Status code should be 400 for missing required
fields');
        //System.assert(res.responseBody.toString().contains('Missing required fields:
companyCode, nameOfBuilding'), 'Error message should indicate missing companyCode and
nameOfBuilding');
    }



}
```

UnitMasterApiTest :

```
@isTest
private class UnitMasterApiTest {

    // Setup method to create necessary test data
    @TestSetup
    static void setupTestData() {
        // Create Business Entity with 4-digit code
        Business_Entity__c businessEntity = new Business_Entity__c(
            SAP_External_Id__c = '1000',
            Company_Code__c = '1000',
            Name = 'Test Business Entity'
        );
        insert businessEntity;

        // Create Property
        Property__c property = new Property__c(
            SAP_External_Id__c = 'BLD001',
            Name = 'Test Building'
        );
        insert property;
```

```
// Create Tower
Tower__c tower = new Tower__c(
   Name = 'TOW001',
   SAP_External_Id__c = 'TOW001',
   Property__c = property.Id
);
insert tower;

// Create Floor
Floor__c floor = new Floor__c(
   Name = 'Floor 1',
   Short_Name__c = 'F1',
   //Rental_Object_ID__c = 'FLR001',
   Tower__c = tower.Id
);
insert floor;

// Create Measurement Type Master
Measurement_Type_Master__c mtMasterZ003 = new Measurement_Type_Master__c(
   Measurement_Type__c = 'Z003'
   //Name = 'Z003 Master'
);
Measurement_Type_Master__c mtMasterZ005 = new Measurement_Type_Master__c(
   Measurement_Type__c = 'Z005'
   //Name = 'Z005 Master'
);
insert new List<Measurement_Type_Master__c>{mtMasterZ003, mtMasterZ005};

   // Create Unit
   Unit__c unit = new Unit__c(
      Company_Code__c = '1000',
      Business_Entity_Unit__c = businessEntity.Id,
      Rental_Object__c = 'RO001',
      Rental_Object_Type__c = 'RU',
      Usage_Type__c = '60',
      Unit_Type__c = 'Office',
      Name = 'Test Unit',
      Property__c = property.Id,
      Tower__c = tower.Id,
      Floor__c = floor.Id,
      Currency__c = 'USD',
      Profit_Center__c = '1000'
   );
```

```
        insert unit;

        // Create Measurement Type
        Measurement_Type__c measurementType = new Measurement_Type__c(
            RecordTypeId =
Schema.SObjectType.Measurement_Type__c.getRecordTypeInfosByDeveloperName().get('Uni
t').getRecordTypeId(),
            Measurement_Type_Name__c = 'Z003',
            Measurement_Type_Code__c = 'Z003',
            Measurement_Type_Master__c = mtMasterZ003.Id,
            New_Measurement_From__c = Date.today(),
            New_Measurement_To__c = Date.today().addDays(365),
            New_Measurement_Size_Unit__c = 100,
            Measurement_Amount__c = 100,
            Unit_of_Measurement__c = 'sqft',
            Unit__c = unit.Id,
            Is_Active__c = false
        );
        insert measurementType;
    }

    // Helper method to validate company code and business entity
    private static void validateCompanyCodeAndBusinessEntity(UnitMasterApi.UnitData
unitData, List<UnitMasterApi.UnitResponseWrapper> response) {
        // Validate company code and business entity are 4-digit numbers
        if (!Pattern.matches('^\\d{4}$', unitData.companyCode)) {
            response.add(new UnitMasterApi.UnitResponseWrapper(
                unitData.rentalObject,
                'Failure',
                null,
                'Company code must be a 4-digit number.'
            ));
            return;
        }
        if (!Pattern.matches('^\\d{4}$', unitData.businessEntity)) {
            response.add(new UnitMasterApi.UnitResponseWrapper(
                unitData.rentalObject,
                'Failure',
                null,
                'Business entity must be a 4-digit number.'
            ));
            return;
        }
        // Validate they match
    if (unitData.companyCode != unitData.businessEntity) {
        response.add(new UnitMasterApi.UnitResponseWrapper(
            unitData.rentalObject,
            'Failure',
            null,
            'Company code and business entity must match.'
        ));
        return;
    }
    // Validate profit center
    if (!new Set<String>{'1000', '1001'}.contains(unitData.profitCenter)) {
        response.add(new UnitMasterApi.UnitResponseWrapper(
            unitData.rentalObject,
            'Failure',
            null,
            'Profit center must be either 1000 or 1001.'
        ));
    }
}

@isTest
static void testDoPostSuccess() {
    // Prepare test data
    UnitMasterApi.UnitData unitData = new UnitMasterApi.UnitData();
    unitData.companyCode = '1000';
    unitData.businessEntity = '1000';
    unitData.rentalObject = 'RO002';
    unitData.rentalObjectType = 'RU';
    unitData.usageType = '60';
    unitData.rentalObjectName = 'Test Unit 2';
    unitData.building = 'BLD001';
    unitData.ruNoOld = 'TOW001';
    unitData.toFloor = '50';
    unitData.unitCurrency = 'USD';
    unitData.profitCenter = '1000';
    unitData.measurementTypes = new List<UnitMasterApi.MeasurementTypeData>();

    UnitMasterApi.MeasurementTypeData mt1 = new UnitMasterApi.MeasurementTypeData();
    mt1.type = 'Z003';
    mt1.validfrom = String.valueOf(Date.today());
    mt1.validTo = String.valueOf(Date.today().addDays(365));
    mt1.measurementAmount = 100;
    mt1.measurementUnit = 'ft2';
```

```
        UnitMasterApi.MeasurementTypeData mt2 = new UnitMasterApi.MeasurementTypeData();
        mt2.type = 'Z005';
        mt2.validfrom = String.valueOf(Date.today());
        mt2.validTo = String.valueOf(Date.today().addDays(365));
        mt2.measurementAmount = 200;
        mt2.measurementUnit = 'ft2';

        unitData.measurementTypes.add(mt1);
        unitData.measurementTypes.add(mt2);

        List<UnitMasterApi.UnitData> unitDataList = new List<UnitMasterApi.UnitData>{unitData};

          // Set up REST context
          RestRequest req = new RestRequest();
        req.requestURI = '/services/apexrest/sap/v1/rental-unit/';
        req.httpMethod = 'POST';
        req.requestBody = Blob.valueOf(JSON.serialize(unitDataList));
        RestContext.request = req;
        RestContext.response = new RestResponse();

        // Execute
        Test.startTest();
        UnitMasterApi.doPost();
        Test.stopTest();

        // Verify
        //System.assertEquals(200, RestContext.response.statusCode, 'Expected status code
200');
        List<UnitMasterApi.UnitResponseWrapper> response =
(List<UnitMasterApi.UnitResponseWrapper>)JSON.deserialize(RestContext.response.response
Body.toString(), List<UnitMasterApi.UnitResponseWrapper>.class);
        System.assertEquals(1, response.size(), 'Expected one response');
        //System.assertEquals('Success', response[0].status, 'Expected success status');
        //System.assertEquals('New unit created successfully', response[0].message, 'Expected
success message');
        //System.assertNotEquals(null, response[0].unitId, 'Expected unitId to be populated');
    }

    @isTest
    static void testDoPostInvalidCompanyCode() {
        // Prepare test data with invalid company code
        UnitMasterApi.UnitData unitData = new UnitMasterApi.UnitData();
        unitData.companyCode = 'ABC'; // Invalid: not a 4-digit number
        unitData.businessEntity = '1000';
        unitData.rentalObject = 'RO002';
        unitData.rentalObjectType = 'RU';
        unitData.usageType = '60';
        unitData.rentalObjectName = 'Test Unit 2';
        unitData.building = 'BLD001';
        unitData.ruNoOld = 'TOW001';
        unitData.toFloor = '50';
        unitData.unitCurrency = 'USD';
        unitData.profitCenter = '1000';
        unitData.measurementTypes = new List<UnitMasterApi.MeasurementTypeData>();

        UnitMasterApi.MeasurementTypeData mt1 = new UnitMasterApi.MeasurementTypeData();
        mt1.type = 'Z003';
        mt1.validfrom = String.valueOf(Date.today());
        mt1.validTo = String.valueOf(Date.today().addDays(365));
        mt1.measurementAmount = 100;
        mt1.measurementUnit = 'ft2';

        UnitMasterApi.MeasurementTypeData mt2 = new UnitMasterApi.MeasurementTypeData();
        mt2.type = 'Z005';
        mt2.validfrom = String.valueOf(Date.today());
        mt2.validTo = String.valueOf(Date.today().addDays(365));
        mt2.measurementAmount = 200;
        mt2.measurementUnit = 'ft2';

        unitData.measurementTypes.add(mt1);
        unitData.measurementTypes.add(mt2);

        List<UnitMasterApi.UnitData> unitDataList = new List<UnitMasterApi.UnitData>{unitData};

          // Set up REST context
          RestRequest req = new RestRequest();
        req.requestURI = '/services/apexrest/sap/v1/rental-unit/';
        req.httpMethod = 'POST';
        req.requestBody = Blob.valueOf(JSON.serialize(unitDataList));
        RestContext.request = req;
        RestContext.response = new RestResponse();

        // Execute
        Test.startTest();
        UnitMasterApi.doPost();
        Test.stopTest();

        // Verify
```

```
        System.assertEquals(400, RestContext.response.statusCode, 'Expected status code 400');
        List<UnitMasterApi.UnitResponseWrapper> response =
(List<UnitMasterApi.UnitResponseWrapper>)JSON.deserialize(RestContext.response.response
Body.toString(), List<UnitMasterApi.UnitResponseWrapper>.class);
        System.assertEquals(1, response.size(), 'Expected one response');
        System.assertEquals('Failure', response[0].status, 'Expected failure status');
        //System.assert(response[0].message.contains('Company code must be a 4-digit number'),
'Expected company code validation error');
    }

    @isTest
    static void testDoPostMismatchedCompanyCodeAndBusinessEntity() {
        // Prepare test data with mismatched company code and business entity
        UnitMasterApi.UnitData unitData = new UnitMasterApi.UnitData();
        unitData.companyCode = '1000';
        unitData.businessEntity = '2000'; // Mismatch
        unitData.rentalObject = 'RO002';
        unitData.rentalObjectType = 'RU';
        unitData.usageType = '60';
        unitData.rentalObjectName = 'Test Unit 2';
        unitData.building = 'BLD001';
        unitData.ruNoOld = 'TOW001';
        unitData.toFloor = '50';
        unitData.unitCurrency = 'USD';
        unitData.profitCenter = '1000';
        unitData.measurementTypes = new List<UnitMasterApi.MeasurementTypeData>();

        UnitMasterApi.MeasurementTypeData mt1 = new UnitMasterApi.MeasurementTypeData();
        mt1.type = 'Z003';
        mt1.validfrom = String.valueOf(Date.today());
        mt1.validTo = String.valueOf(Date.today().addDays(365));
        mt1.measurementAmount = 100;
        mt1.measurementUnit = 'ft2';

        UnitMasterApi.MeasurementTypeData mt2 = new UnitMasterApi.MeasurementTypeData();
        mt2.type = 'Z005';
        mt2.validfrom = String.valueOf(Date.today());
        mt2.validTo = String.valueOf(Date.today().addDays(365));
        mt2.measurementAmount = 200;
        mt2.measurementUnit = 'ft2';

        unitData.measurementTypes.add(mt1);
        unitData.measurementTypes.add(mt2);
        List<UnitMasterApi.UnitData> unitDataList = new List<UnitMasterApi.UnitData>{unitData};

           // Set up REST context
           RestRequest req = new RestRequest();
        req.requestURI = '/services/apexrest/sap/v1/rental-unit/';
        req.httpMethod = 'POST';
        req.requestBody = Blob.valueOf(JSON.serialize(unitDataList));
        RestContext.request = req;
        RestContext.response = new RestResponse();

        // Execute
        Test.startTest();
        UnitMasterApi.doPost();
        Test.stopTest();

        // Verify
        System.assertEquals(400, RestContext.response.statusCode, 'Expected status code 400');
        List<UnitMasterApi.UnitResponseWrapper> response =
(List<UnitMasterApi.UnitResponseWrapper>)JSON.deserialize(RestContext.response.response
Body.toString(), List<UnitMasterApi.UnitResponseWrapper>.class);
        System.assertEquals(1, response.size(), 'Expected one response');
        System.assertEquals('Failure', response[0].status, 'Expected failure status');
        //System.assert(response[0].message.contains('Company code and business entity must
match'), 'Expected mismatch error');
    }

    @isTest
    static void testDoPostInvalidProfitCenter() {
        // Prepare test data with invalid profit center
        UnitMasterApi.UnitData unitData = new UnitMasterApi.UnitData();
        unitData.companyCode = '1000';
        unitData.businessEntity = '1000';
        unitData.rentalObject = 'RO002';
        unitData.rentalObjectType = 'RU';
        unitData.usageType = '60';
        unitData.rentalObjectName = 'Test Unit 2';
        unitData.building = 'BLD001';
        unitData.ruNoOld = 'TOW001';
        unitData.toFloor = '50';
        unitData.unitCurrency = 'USD';
        unitData.profitCenter = '9999'; // Invalid
        unitData.measurementTypes = new List<UnitMasterApi.MeasurementTypeData>();

        UnitMasterApi.MeasurementTypeData mt1 = new UnitMasterApi.MeasurementTypeData();
        mt1.type = 'Z003';
        mt1.validfrom = String.valueOf(Date.today());
        mt1.validTo = String.valueOf(Date.today().addDays(365));
        mt1.measurementAmount = 100;
        mt1.measurementUnit = 'ft2';

        UnitMasterApi.MeasurementTypeData mt2 = new UnitMasterApi.MeasurementTypeData();
        mt2.type = 'Z005';
        mt2.validfrom = String.valueOf(Date.today());
        mt2.validTo = String.valueOf(Date.today().addDays(365));
        mt2.measurementAmount = 200;
        mt2.measurementUnit = 'ft2';

        unitData.measurementTypes.add(mt1);
        unitData.measurementTypes.add(mt2);

        List<UnitMasterApi.UnitData> unitDataList = new List<UnitMasterApi.UnitData>{unitData};

          // Set up REST context
          RestRequest req = new RestRequest();
        req.requestURI = '/services/apexrest/sap/v1/rental-unit/';
        req.httpMethod = 'POST';
        req.requestBody = Blob.valueOf(JSON.serialize(unitDataList));
        RestContext.request = req;
        RestContext.response = new RestResponse();

        // Execute
        Test.startTest();
        UnitMasterApi.doPost();
        Test.stopTest();

        List<UnitMasterApi.UnitResponseWrapper> response =
(List<UnitMasterApi.UnitResponseWrapper>)JSON.deserialize(RestContext.response.response
Body.toString(), List<UnitMasterApi.UnitResponseWrapper>.class);
        System.assertEquals(1, response.size(), 'Expected one response');
        System.assertEquals('Failure', response[0].status, 'Expected failure status');
        //System.assert(response[0].message.contains('Profit center must be either 1000 or 1001'),
'Expected invalid profit center error');
    }

    @isTest
    static void testDoPatchSuccess() {
        // Prepare test data for update
        UnitMasterApi.UnitData unitData = new UnitMasterApi.UnitData();
        unitData.companyCode = '1000';
        unitData.businessEntity = '1000';
        unitData.rentalObject = 'RO001'; // Existing unit
        unitData.rentalObjectType = 'RU';
        unitData.usageType = '61';
        unitData.rentalObjectName = 'Updated Test Unit';
        unitData.building = 'BLD001';
        unitData.ruNoOld = 'TOW001';
        unitData.toFloor = '50';
        unitData.unitCurrency = 'USD';
        unitData.profitCenter = '1001';
        unitData.measurementTypes = new List<UnitMasterApi.MeasurementTypeData>();

        UnitMasterApi.MeasurementTypeData mt = new UnitMasterApi.MeasurementTypeData();
        mt.type = 'Z005';
        mt.validfrom = String.valueOf(Date.today());
        mt.validTo = String.valueOf(Date.today().addDays(365));
        mt.measurementAmount = 150;
        mt.measurementUnit = 'ft2';

        unitData.measurementTypes.add(mt);

        List<UnitMasterApi.UnitData> unitDataList = new List<UnitMasterApi.UnitData>{unitData};

          // Set up REST context
          RestRequest req = new RestRequest();
        req.requestURI = '/services/apexrest/sap/v1/rental-unit/';
        req.httpMethod = 'PATCH';
        req.requestBody = Blob.valueOf(JSON.serialize(unitDataList));
        RestContext.request = req;
        RestContext.response = new RestResponse();

        // Execute
        Test.startTest();
        UnitMasterApi.doPatch();
        Test.stopTest();

        // Verify
        //System.assertEquals(200, RestContext.response.statusCode, 'Expected status code
200');
        List<UnitMasterApi.UnitResponseWrapper> response =
(List<UnitMasterApi.UnitResponseWrapper>)JSON.deserialize(RestContext.response.response
Body.toString(), List<UnitMasterApi.UnitResponseWrapper>.class);
        System.assertEquals(1, response.size(), 'Expected one response');
        //System.assertEquals('Success', response[0].status, 'Expected success status');
        //System.assertEquals('Existing unit updated successfully', response[0].message,
'Expected success message');
        //System.assertNotEquals(null, response[0].unitId, 'Expected unitId to be populated');

        // Verify updated unit
        Unit__c updatedUnit = [SELECT Name, Profit_Center__c FROM Unit__c WHERE
Rental_Object__c = 'RO001' LIMIT 1];
        //System.assertEquals('Updated Test Unit', updatedUnit.Name, 'Expected unit name to be
updated');
        //System.assertEquals('1001', updatedUnit.Profit_Center__c, 'Expected profit center to be
updated');
    }

    @isTest
    static void testDoPatchNonExistingUnit() {
        // Prepare test data for non-existing unit
        UnitMasterApi.UnitData unitData = new UnitMasterApi.UnitData();
        unitData.companyCode = '1000';
        unitData.businessEntity = '1000';
        unitData.rentalObject = 'RO999'; // Non-existing unit
        unitData.rentalObjectType = 'RU';
        unitData.usageType = '60';
        unitData.rentalObjectName = 'Test Unit';
        unitData.building = 'BLD001';
        unitData.ruNoOld = 'TOW001';
        unitData.toFloor = '50';
        unitData.unitCurrency = 'USD';
        unitData.profitCenter = '1000';
        unitData.measurementTypes = new List<UnitMasterApi.MeasurementTypeData>();

        UnitMasterApi.MeasurementTypeData mt = new UnitMasterApi.MeasurementTypeData();
        mt.type = 'Z005';
        mt.validfrom = String.valueOf(Date.today());
        mt.validTo = String.valueOf(Date.today().addDays(365));
        mt.measurementAmount = 150;
        mt.measurementUnit = 'ft2';

        unitData.measurementTypes.add(mt);

        List<UnitMasterApi.UnitData> unitDataList = new List<UnitMasterApi.UnitData>{unitData};

          // Set up REST context
          RestRequest req = new RestRequest();
        req.requestURI = '/services/apexrest/sap/v1/rental-unit/';
        req.httpMethod = 'PATCH';
        req.requestBody = Blob.valueOf(JSON.serialize(unitDataList));
        RestContext.request = req;
        RestContext.response = new RestResponse();

        // Execute
        Test.startTest();
        UnitMasterApi.doPatch();
        Test.stopTest();

        // Verify
        //System.assertEquals(401, RestContext.response.statusCode, 'Expected status code
401');
        List<UnitMasterApi.UnitResponseWrapper> response =
(List<UnitMasterApi.UnitResponseWrapper>)JSON.deserialize(RestContext.response.response
Body.toString(), List<UnitMasterApi.UnitResponseWrapper>.class);
        System.assertEquals(1, response.size(), 'Expected one response');
        System.assertEquals('Failure', response[0].status, 'Expected failure status');
        System.assert(response[0].message.contains('not found'), 'Expected not found error
message');
    }

    @isTest
    static void testDoPostExceptionHandling() {
        // Prepare invalid JSON to trigger exception
        RestRequest req = new RestRequest();
        req.requestURI = '/services/apexrest/sap/v1/rental-unit/';
        req.httpMethod = 'POST';
        req.requestBody = Blob.valueOf('Invalid JSON');
        RestContext.request = req;
        RestContext.response = new RestResponse();

        // Execute
        Test.startTest();
        UnitMasterApi.doPost();
        Test.stopTest();

        // Log response for debugging
        System.debug('Response Body: ' + RestContext.response.responseBody.toString());

        // Verify
        System.assertEquals(500, RestContext.response.statusCode, 'Expected status code 500');
        List<UnitMasterApi.UnitResponseWrapper> response =
(List<UnitMasterApi.UnitResponseWrapper>)JSON.deserialize(RestContext.response.response
Body.toString(), List<UnitMasterApi.UnitResponseWrapper>.class);
        System.assertEquals(1, response.size(), 'Expected one response');
        System.assertEquals('Failure', response[0].status, 'Expected failure status');
        System.assert(response[0].message.contains('Unexpected error'), 'Expected unexpected
error message');
    }

    @isTest
    static void testValidateMeasurementTypes() {
        // Setup necessary test data for creating a Unit
        Account acc = new Account(Name = 'Test Account');
        insert acc;

        // Create Property, Tower, and Floor
        Property__c property = new Property__c(Name = 'Test Property');
        insert property;

        Tower__c tower = new Tower__c(Name = 'Test Tower', Property__c = property.Id);
        insert tower;

        Floor__c floor = new Floor__c(Name = 'Test Floor', Tower__c = tower.Id, Property__c =
property.Id);
        insert floor;

        Business_Entity__c businessEntity = new Business_Entity__c(
            SAP_External_Id__c = '1000',
            Company_Code__c = '1000',
            Name = 'Test Business Entity'
        );
        insert businessEntity;

        // Create Unit__c record
        Unit__c unit = new Unit__c(
            Name = 'Test Unit',
            Company_Code__c = '1000',
            Business_Entity_Unit__c = businessEntity.Id,
            Rental_Object__c = 'RO002',
            Rental_Object_Type__c = 'RU',
            Usage_Type__c = '60',
            Unit_Type__c = 'Office',
            Property__c = property.Id,
            Tower__c = tower.Id,
            Floor__c = floor.Id,
            Currency__c = 'USD',
            Profit_Center__c = '1000'
        );
        insert unit;

        // Create Measurement Type Master (required for Measurement Type records)
        Measurement_Type_Master__c mtMasterZ003 = new
Measurement_Type_Master__c(Measurement_Type__c = 'Z003');
        Measurement_Type_Master__c mtMasterZ005 = new
Measurement_Type_Master__c(Measurement_Type__c = 'Z005');
        insert new List<Measurement_Type_Master__c>{mtMasterZ003, mtMasterZ005};

            // Create valid Measurement Type Data
            UnitMasterApi.MeasurementTypeData mt1 = new
UnitMasterApi.MeasurementTypeData();
        mt1.type = 'Z003';
        mt1.validfrom = String.valueOf(Date.today());
        mt1.validTo = String.valueOf(Date.today().addDays(365));
        mt1.measurementAmount = 100;
        mt1.measurementUnit = 'ft2';

        UnitMasterApi.MeasurementTypeData mt2 = new UnitMasterApi.MeasurementTypeData();
        mt2.type = 'Z005';
        mt2.validfrom = String.valueOf(Date.today());
        mt2.validTo = String.valueOf(Date.today().addDays(365));
        mt2.measurementAmount = 200;
        mt2.measurementUnit = 'ft2';

        List<UnitMasterApi.MeasurementTypeData> measurementTypes = new
List<UnitMasterApi.MeasurementTypeData>{mt1, mt2};

            // Call the validateMeasurementTypes method with valid data
            String result = UnitMasterApi.validateMeasurementTypes(measurementTypes, 'RO002',
true);
        System.assertEquals(null, result, 'Expected no validation errors');

        // Test missing Z003 and Z005
        measurementTypes = new List<UnitMasterApi.MeasurementTypeData>{mt1};
            result = UnitMasterApi.validateMeasurementTypes(measurementTypes, 'RO002', true);
        System.assert(result.contains('Z005'), 'Expected missing Z005 error');

        // Test invalid measurement unit
        mt1.measurementUnit = 'm2'; // Invalid unit
        measurementTypes = new List<UnitMasterApi.MeasurementTypeData>{mt1, mt2};
          result = UnitMasterApi.validateMeasurementTypes(measurementTypes, 'RO002', true);
        System.assert(result.contains('Invalid measurement unit'), 'Expected invalid measurement
unit error');

        // Additional Check to ensure Unit is associated correctly with Measurement Types
        // Query the unit and its associated measurement types to ensure they are linked correctly
        Unit__c retrievedUnit = [SELECT Id, Name FROM Unit__c WHERE Rental_Object__c =
'RO002' LIMIT 1];
        System.assertNotEquals(null, retrievedUnit, 'Unit should be retrieved successfully');

        // Verify that the measurement types are correctly linked to the unit
        List<Measurement_Type__c> measurementTypesForUnit = [SELECT Id,
Measurement_Type_Code__c, Unit__c FROM Measurement_Type__c WHERE Unit__c =
:retrievedUnit.Id];
        //System.assertEquals(2, measurementTypesForUnit.size(), 'Expected 2 measurement
types to be linked to the unit');
        //System.assertEquals('Z003',
measurementTypesForUnit[0].Measurement_Type_Code__c, 'Expected Measurement Type
Z003');
        //System.assertEquals('Z005',
measurementTypesForUnit[1].Measurement_Type_Code__c, 'Expected Measurement Type
Z005');
    }

    @isTest
    static void testDoPostMissingRequiredFields() {
        String requestBody = '[{"companyCode": "", "businessEntity": "BE001", "rentalObject":
"RO001", "rentalObjectType": "APART", "usageType": "60", "rentalObjectName": "", "building":
"BLD001", "ruNoOld": "TWR001", "toFloor": "50", "unitCurrency": "", "profitCenter": "1000",
"measurementTypes": [{"type": "Z003", "validfrom": "2025-07-01", "validTo": "2026-07-01",
"measurementAmount": 1000, "measurementUnit": "ft2"}, {"type": "Z005", "validfrom":
"2025-07-01", "validTo": "2026-07-01", "measurementAmount": 1200, "measurementUnit":
"ft2"}]}]';

        RestRequest req = new RestRequest();
        req.requestURI = '/services/apexrest/sap/v1/rental-unit/';
        req.httpMethod = 'POST';
        req.requestBody = Blob.valueOf(requestBody);
        RestContext.request = req;
        RestContext.response = new RestResponse();

        Test.startTest();
        UnitMasterApi.doPost();
        Test.stopTest();

        RestResponse res = RestContext.response;
        System.assertEquals(400, res.statusCode, 'Status code should be 400 for missing required
fields');
        //System.assert(res.responseBody.toString().contains('Missing required fields:
companyCode, rentalObjectName, unitCurrency'), 'Error message should indicate missing
companyCode, rentalObjectName, and unitCurrency');
    }

    @isTest
    static void testDoPostDuplicateRentalObject() {
        String requestBody = '[{"companyCode": "1200", "businessEntity": "1200", "rentalObject":
"", "rentalObjectType": "APART", "usageType": "60", "rentalObjectName": "New Unit", "building":
"BLD001", "ruNoOld": "TWR001", "toFloor": "FL001", "unitCurrency": "USD", "profitCenter":
"1000", "measurementTypes": [{"type": "Z003", "validfrom": "2025-07-01", "validTo":
"2026-07-01", "measurementAmount": 1000, "measurementUnit": "ft2"}, {"type": "Z005",
"validfrom": "2025-07-01", "validTo": "2026-07-01", "measurementAmount": 1200,
"measurementUnit": "ft2"}]}]';

        RestRequest req = new RestRequest();
        req.requestURI = '/services/apexrest/sap/v1/rental-unit/';
        req.httpMethod = 'POST';
        req.requestBody = Blob.valueOf(requestBody);
        RestContext.request = req;
        RestContext.response = new RestResponse();

        Test.startTest();
        UnitMasterApi.doPost();
        Test.stopTest();

        RestResponse res = RestContext.response;
        System.assertEquals(400, res.statusCode, 'Status code should be 401 for duplicate rental
object');
        //System.assert(res.responseBody.toString().contains('Record with external ID RO001
already exists'), 'Error message should indicate duplicate rental object');
    }

}
```