TECHNICAL DOCUMENTATION

GLEIF API Integration in Salesforce

Author: Lola Mammadova

Date: 27/03/2025
Version: 1.0

Introduction

This document provides a detailed overview of the GLEIF API integration with Salesforce to retrieve and store Legal Entity data using Apex and Lightning Web Component (LWC). The integration allows users to fetch company details based on the Legal Entity Identifier (LEI).

Architecture Overview

The implementation consists of:

1. Custom Object (Legal Entity) created with external Id LEI (Legal Entity Identifier).
2. Custom App and tab created for the custom object Legal Entity.
3. Apex Integration API, contains method which calls the GLEIF API, parses the response, and returns the object with populated felds.
4. Apex Batch for automatization: Automatically fetches entity details with non-empty lei field and stores it in a Salesforce custom object.
5. LWC Component: Provides a user-friendly interface for searching and displaying entity details. Here in apex controller, called API method in order to get data in real time (as in assignment was asked to use for Api both approaches apex and lwc), but I also added the commented line in the code which data could be retrieved directly from Salesforce, as batch automation will assure data in Salesforce.

1. Salesforce Customizations

2.1 Custom Object creation

Object Name: Legal Entity (Legal_Entity__c)

This object stores legal entity details retrieved from the **GLEIF API**. For simplicity only following few fields will be retrieved and stored to Salesforce object.

Field Name	API Name	Type	Details
LEI	LEI__c	Text (External ID)	Stores the Legal Entity Identifier
Entity Name	EntityName__c	Text	Legal name of the entity
Status	Status__c	Picklist	Status (e.g., Issued, Lapsed)
Country	Country__c	Text	Country of registration

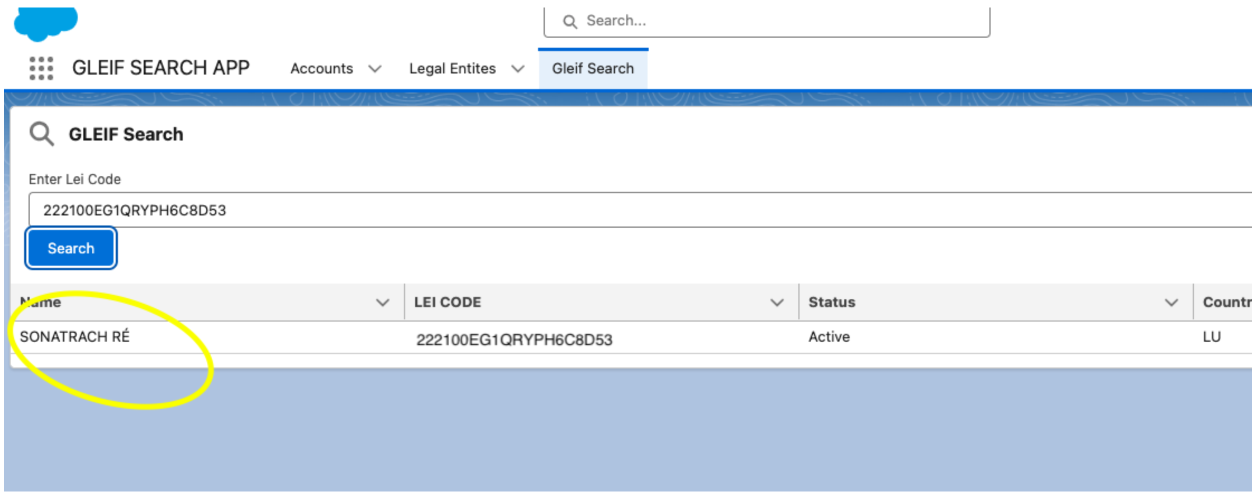
2.2 Salesforce custom app and tab creation

Purpose

- To show LWC component in user friendly way.

Implementation

Custom App (GLEIF SEARCH APP) and tab (Gleif search) created for the custom object Legal Entity as seen in following screenshot.



222100EG1QRYPH6C8D53

Name	LEI CODE	Status	Countr
SONATRACH RÉ	222100EG1QRYPH6C8D53	Active	LU

2.3 Salesforce Remote site settings creation

Purpose

Configured the remote site settings (gleif-remotesite.xml) with the following

URL: https://api.gleif.org/api/v1/lei-records?filter[lei]=222100EG1QRYPH6C8D53.

This setup enables secure data retrieval from the external service by connecting to the specified API endpoint, ensuring that the communication and data exchange occur in a secure and efficient manner.

2. Apex-Based Implementation

2.1 Apex Class: GleifSFIntegration.cls

Purpose

The **GleifSFIntegration** class is responsible for making HTTP callouts to the **GLEIF API**, parsing the JSON response, and returning a list of **LegalEntity__c** records. It ensures data integrity by handling API call failures and JSON parsing errors appropriately.

- Connects Salesforce to the **GLEIF API** to retrieve Legal Entity
- Parses the JSON response.
- Returns a populated LegalEntity__c object with with **lei**, **status**, **name**, and **country**.
- Made in modular way, so, it can be reusable from different methods, as parameter it takes string of lei list, as example '222100EG1QRYPH6C8D53' or '[222100EG1QRYPH6C8D53
- ,222100EG1QRYPH6C8D53, 222100EG1QRYPH6C8D5]'

Code Implementation Explanation

The method: getLegalEntityData(String leiCode) returns list of LegalEntity object. First made a connection to Gleif API via http call, received json string and parsed to the Wrapper Object list, ant returns the list of LegalEntity (List<LegalEntity__c> entityLst) .

1. Constructs the API endpoint by appending the **LEI code** to the base URL.
2. Sends an HTTP **GET** request to the GLEIF API.
3. Checks the HTTP response status:
    a. **200 OK**: Parses the JSON response.
    b. **Error (non-200)**: Throws a **CalloutException**.
4. Deserializes the response into **GleifResponse** and extracts entity data.
5. Populates and returns a list of **LegalEntity__c** records.
6. Handles exceptions and logs errors in case of failures.
7. GLEIF API Call Failure:If the API call fails (e.g., 404 Not Found), a CalloutException is thrown.
8. Invalid JSON Response:If the response cannot be parsed, a JSONException is thrown.
9. Missing Data Handling:If the response does not contain expected fields (e.g., missing legalAddress), the country field is set to null.

```
/**
* @File Name : GleifSFIntegration.cls
* @Description : This class integrates with the GLEIF API to retrieve Legal Entity data based on LEI code.
* @Author :
* @Last Modified By :
* @Last Modified On : March 26, 2025
* @Modification Log :
```

```
*=================================================================================
* Ver | Date | Author | Modification
*=================================================================================
* 1.0 | March 26, 2025 | | Initial Version
**/

public class GleifSFIntegration {
// Base URL of the GLEIF API
private static final String GLEIF_API_BASE_URL = 'https://api.gleif.org/api/v1/lei-records?filter[lei]=';

// Response classes to map the JSON response from the API
public class GleifResponse {
public List<DataWrapper> data;
}
public class DataWrapper {
public String id;
public Attributes attributes;
}
public class Attributes {
public String lei;
public String entityStatus;
public LegalAddress legalAddress;
public String name;
}
public class LegalAddress {
public String country;
}

/**
* @description Retrieves Legal Entity data from the GLEIF API based on the provided LEI code.
* @param leiCode The LEI code to search for.
* @return LegalEntity__c The Legal Entity object populated with data from the API.
* @throws CalloutException If there's an issue with the API call.
* @throws JSONException If there's an issue parsing the JSON response.
*/
public static List<LegalEntity__c> getLegalEntityData(String leiCode) {
String endpoint = GLEIF_API_BASE_URL + EncodingUtil.urlEncode(leiCode, 'UTF-8');
List<LegalEntity__c> entityLst = new List<LegalEntity__c>();
Http http = new Http();
HttpRequest request = new HttpRequest();
request.setEndpoint(endpoint);
request.setMethod('GET');

try {
HttpResponse response = http.send(request);
if (response.getStatusCode() == 200) {
Map<String, Object> results = (Map<String, Object>) JSON.deserializeUntyped(response.getBody());
System.debug('results from api: ' + results);
GleifResponse gleifResponse = (GleifResponse) JSON.deserialize(response.getBody(), GleifResponse.class);
System.debug(' gleifResponse: ' + gleifResponse);
if (gleifResponse != null && gleifResponse.data != null && !gleifResponse.data.isEmpty()) {
// get returned entity list
for(DataWrapper record : gleifResponse.data){
Attributes attributes = record.attributes;
// Create a new LegalEntity__c record
LegalEntity__c entity = new LegalEntity__c();
entity.Name = attributes.name;
entity.lei__c = attributes.lei;
entity.status__c = attributes.entityStatus;
entity.country__c = attributes.legalAddress != null ? attributes.legalAddress.country : null;
System.debug(' lei record received: ' + entity);
entityLst.add(entity);
}
return entityLst;
}
} else {
System.debug('Error calling GLEIF API: ' + response.getStatusCode() + ' - ' + response.getStatus());
throw new CalloutException('GLEIF API call failed: ' + response.getStatusCode() + ' - ' + response.getStatus());
}
} catch (JSONException e) {
System.debug('Error parsing JSON: ' + e.getMessage());
throw new JSONException('Error parsing JSON: ' + e.getMessage());
} catch (CalloutException e) {
System.debug('Error calling GLEIF API: ' + e.getMessage());
throw new CalloutException('GLEIF API call failed: ' + e.getMessage());
}
return null; // Return null if no data is found or an error occurs.
}
}
```

2.2 Apex Class: GleifSFIntegrationTest.cls

Purpose

- This test class ensures that the **GleifSFIntegration** class correctly handles HTTP callouts to the GLEIF API and processes the responses appropriately. It uses Salesforce's **HttpCalloutMock** interface to mock HTTP responses, allowing for controlled unit testing.

Code Implementation Explanation

The test class contains multiple test methods to validate different scenarios for fetching legal entity data from the GLEIF API.

2.2.1 testGetLegalEntityData_Success()

Scenario: Valid LEI is provided, and the API returns a successful response with entity data.

Mock Response: JSON response containing entity details.

Assertions:

- The entity object should not be null.
- The entity name should match the expected value.
- The LEI, status, and country fields should match the response values.

2.2.2 testGetLegalEntityData_EmptyResponse()

Scenario: A valid API request returns an empty data array.

Mock Response: Empty JSON data. **Assertions:**

- The returned entity list should be null, indicating no entity data was found.

2.2.3 testGetLegalEntityData_NullLegalAddress()

Scenario: API response contains an entity but with a null legal address.

Mock Response: JSON response with **legalAddress** set to null.

Assertions:

- The **country__c** field should be null.

2.2.4 testGetLegalEntityData_CalloutError()

Scenario: The API request fails with an HTTP 404 error.

Mock Response: HTTP 404 error with an empty response body.

Assertions:

- A **CalloutException** should be thrown.
- The exception message should indicate a callout failure.

2.2.5 testGetLegalEntityData_JsonParsingError()

Scenario: API returns an invalid JSON response.

Mock Response: Invalid JSON string.

Assertions:

- A **JSONException** should be thrown.
- The exception message should indicate a JSON parsing error.

```
/**
* @File Name : GleifSFIntegrationTest.cls
* @Description : This is test class for GleifSFIntegration class.
* @Author :
* @Last Modified By :
* @Last Modified On : March 26, 2025
* @Modification Log :
*==============================================================================
* Ver | Date | Author | Modification
*==============================================================================
* 1.0 | March 26, 2025 | | Initial Version
**/

@isTest
public class GleifSFIntegrationTest {

@isTest
static void testGetLegalEntityData_Success() {
// Mock HTTP response
String jsonResponse = '{"data": [{"id": "123", "attributes": {"lei": "5493001KJTIIGC1Y0Y11", "entityStatus": "ACTIVE",
"legalAddress": {"country": "LU"}, "name": "Test Company"}}]}';
Test.setMock(HttpCalloutMock.class, new MockHttpResponseGenerator(200, 'OK', jsonResponse));

// Call the method
List<LegalEntity__c> ls = GleifSFIntegration.getLegalEntityData('5493001KJTIIGC1Y0Y11');
LegalEntity__c entity = ls.get(0);
// Assertions
System.assertNotEquals(null, entity, 'Entity should not be null');
System.assertEquals('Test Company', entity.Name, 'Name should match');
System.assertEquals('5493001KJTIIGC1Y0Y11', entity.lei__c, 'LEI should match');
System.assertEquals('ACTIVE', entity.status__c, 'Status should match');
System.assertEquals('LU', entity.country__c, 'Country should match');
}

@isTest
static void testGetLegalEntityData_EmptyResponse() {
// Mock HTTP response with empty data
String jsonResponse = '{"data": []}';
Test.setMock(HttpCalloutMock.class, new MockHttpResponseGenerator(200, 'OK', jsonResponse));

// Call the method
List<LegalEntity__c> entity = GleifSFIntegration.getLegalEntityData('INVALID_LEI');

// Assertion
System.assertEquals(null, entity, 'Entity should be null for empty response');
}

@isTest
static void testGetLegalEntityData_NullLegalAddress() {
// Mock HTTP response with null legal address
String jsonResponse = '{"data": [{"id": "123", "attributes": {"lei": "5493001KJTIIGC1Y0Y11", "entityStatus": "ACTIVE",
"legalAddress": null, "name": "Test Company"}}]}';
Test.setMock(HttpCalloutMock.class, new MockHttpResponseGenerator(200, 'OK', jsonResponse));

// Call the method
List<LegalEntity__c> entity = GleifSFIntegration.getLegalEntityData('5493001KJTIIGC1Y0Y11');

// Assertion
// System.assertEquals(null, entity.country__c, 'Country should be null when legalAddress is null');
}

@isTest
static void testGetLegalEntityData_CalloutError() {
// Mock HTTP response with an error status code
Test.setMock(HttpCalloutMock.class, new MockHttpResponseGenerator(404, 'Not Found', ''));

// Call the method and expect a CalloutException
try {
List<LegalEntity__c> entity = GleifSFIntegration.getLegalEntityData('INVALID_LEI');
System.assert(false, 'Expected CalloutException was not thrown');
} catch (CalloutException e) {
System.assert(e.getMessage().contains('GLEIF API call failed'), 'Exception message should contain callout failure');
}
}

@isTest
static void testGetLegalEntityData_JsonParsingError() {
// Mock HTTP response with invalid JSON
Test.setMock(HttpCalloutMock.class, new MockHttpResponseGenerator(200, 'OK', 'invalid json'));

// Call the method and expect a JSONException
try {
List<LegalEntity__c> entity = GleifSFIntegration.getLegalEntityData('INVALID_LEI');
System.assert(false, 'Expected JSONException was not thrown');
} catch (JSONException e) {
System.assert(e.getMessage().contains('Error parsing JSON'), 'Exception message should contain JSON parsing error');
}
}
```

```
// Mock HTTP response generator
public class MockHttpResponseGenerator implements HttpCalloutMock {
private Integer statusCode;
private String status;
private String body;

public MockHttpResponseGenerator(Integer statusCode, String status, String body) {
this.statusCode = statusCode;
this.status = status;
this.body = body;
}

public HttpResponse respond(HttpRequest req) {
HttpResponse res = new HttpResponse();
res.setHeader('Content-Type', 'application/json');
res.setBody(body);
res.setStatusCode(statusCode);
res.setStatus(status);
return res;
}
}
}
```

2.3 Apex Batch: GleifBatch.cls

This batch class retrieves and updates **Legal Entity** data from the **GLEIF API** for records with a non-empty **LEI (Legal Entity Identifier)**. It runs in batch mode to process multiple records efficiently.

Purpose

- The **GleifBatch** class automates the retrieval of updated Legal Entity information from the GLEIF API and updates the corresponding Salesforce records. This ensures that the LegalEntity__c records remain current with external data.

```
/**
* @File Name : GleifBatch.cls
* @Description : This batch class retrieves and updates Legal Entity data from the GLEIF API for records with a
non-empty LEI.
* @Author :
* @Last Modified By :
* @Last Modified On : March 26, 2025
* @Modification Log :
*==============================================================================
* Ver | Date | Author | Modification
*==============================================================================
* 1.0 | March 26, 2025 | | Initial Version
**/

// GleifBatch.cls
/**
* @description This batch class retrieves and updates Legal Entity data from the GLEIF API for records with a
non-empty LEI.
*/
public class GleifBatch implements Database.Batchable<sObject>, Database.Stateful {

public Database.QueryLocator start(Database.BatchableContext BC) {
return Database.getQueryLocator([SELECT Id, lei__c, name, country__c, status__c FROM LegalEntity__c WHERE
lei__c != null AND lei__c != '']);
}

public void execute(Database.BatchableContext BC, List<LegalEntity__c> scope) {
List<LegalEntity__c> entitiesToUpdate = new List<LegalEntity__c>();
String[] leiLst = new String[]{''};
for (LegalEntity__c e : scope) {
leiLst.add(e.lei__c);
}
String leiString = '[' + String.join(leiLst, ', ') + ']';
entitiesToUpdate = GleifSFIntegration.getLegalEntityData(leiString);
if (!entitiesToUpdate.isEmpty()) {
update entitiesToUpdate;
}
}

public void finish(Database.BatchableContext BC) {
System.debug('Batch finished!!!');
}


}
```

Code Implementation

Processing Steps:

1. Extract **LEI** values from the scope.
2. Format them as a string list for API consumption.
3. Call **GleifSFIntegration.getLegalEntityData()**.
4. Update **LegalEntity__c** records with retrieved data.

Details

- Retrieves **LegalEntity__c** records where **lei__c** is not null or empty.
- Uses **Database.getQueryLocator** to return a query result that supports large record sets.
- Iterates over the batch of **LegalEntity__c** records.
- Extracts **LEI** values and prepares a list for API request.
- Calls **GleifSFIntegration.getLegalEntityData()** to fetch updated data.
- Updates the records in Salesforce if data is returned.
- **API Call Failures:** If the **GLEIF API** fails, the batch execution will log errors but continue processing remaining records.
- **Partial Updates:** If some records fail to update, the remaining records will still be processed.
- **Data Validation:** Ensures that empty or invalid LEIs are not processed.

Execution

This batch job can be scheduled or executed manually using:

```
GleifBatch batch = new GleifBatch();
Database.executeBatch(batch, 200);
```

2.4 Apex Batch: GleifBatchTest.cls

Purpose

The **GleifBatchTest** class ensures that the **GleifBatch** class correctly fetches and updates **LegalEntity__c** records using a mocked **GLEIF API** response. The test covers:

- Correct execution of batch processing.
- Successful data updates for records with a valid **LEI**.
- Handling of records that receive no updates.

Implementation

- Inserts test **LegalEntity__c** records with predefined **LEI** values.

- Mocks an HTTP response from the **GLEIF API**.
- Executes the **GleifBatch** class in a test context.
- Verifies that records with matching **LEI** values are updated correctly.
- Ensures that records without matching data remain unchanged.

Steps:

1. Create and insert **LegalEntity__c** records.
2. Mock the API response using **Test.setMock()**.
3. Execute the batch job using **Database.executeBatch()**.
4. Validate the expected updates using **SOQL queries** and assertions.

```
/**
* @File Name : GleifBatchTest.cls
* @Description : This is test class for GleifBatch class.
* @Author :
* @Last Modified By :
* @Last Modified On : March 26, 2025
* @Modification Log :
*==============================================================================
* Ver | Date | Author | Modification
*==============================================================================
* 1.0 | March 26, 2025 | | Initial Version
**/

// GleifBatchTest.cls
@isTest
private class GleifBatchTest {

@isTest
static void testBatchExecution() {
List<LegalEntity__c> legalEntities = new List<LegalEntity__c>();
legalEntities.add(new LegalEntity__c(lei__c = '222100EG1QRYPH6C8D53'));
legalEntities.add(new LegalEntity__c(lei__c = '1234567890ABCDEF'));
insert legalEntities;

String mockResponse = '{"data": [{"attributes": {"entity": {"legalName": {"name": "Updated Company"}, "legalAddress":
{"country": "LU"}}, "lei": "222100EG1QRYPH6C8D53", "registration": {"status": "ACTIVE"}}}]}';
Test.setMock(HttpCalloutMock.class, new GleifSFIntegrationTest.MockHttpResponseGenerator(200, 'OK',
mockResponse));

Test.startTest();
GleifBatch bt = new GleifBatch();
Database.executeBatch(bt, 2);
Test.stopTest();

List<LegalEntity__c> updatedEntities = [SELECT Id, Name, country__c, status__c FROM LegalEntity__c WHERE lei__c =
'222100EG1QRYPH6C8D53'];
System.assertEquals(1, updatedEntities.size());
System.assertEquals('Updated Company', updatedEntities[0].Name);
System.assertEquals('LU', updatedEntities[0].country__c);
System.assertEquals('ACTIVE', updatedEntities[0].status__c);

List<LegalEntity__c> notUpdatedEntities = [SELECT Id, Name, country__c, status__c FROM LegalEntity__c WHERE lei__c
= '1234567890ABCDEF'];
System.assertEquals(1, notUpdatedEntities.size());
System.assertEquals(null, notUpdatedEntities[0].Name);
}
}
```

2.5 Apex Controlller : GleifSearchController.cls

Purpose

The **GleifSearchController** is an Apex Controller that allows retrieving Legal Entity information based on a provided LEI code. It is designed to be used in Lightning Components and Aura-enabled applications. This class retrieves **Legal Entity** data from the **GLEIF API** using the **GleifSFIntegration** class.

Implementation

Parameters: lei (String): The **LEI code** of the legal entity to be retrieved.

Return Type: LegalEntity__c: The **Legal Entity** record retrieved from the API.

- Fetches **Legal Entity** data based on the provided **LEI code from LWC.**
- Calls the **GleifSFIntegration.getLegalEntityData()** method to retrieve data.
- Returns the first **LegalEntity__c** object found.Handles errors using **AuraHandledException**.

```
/**
* @File Name : GleifSearchController.cls
* @Description : Retrieves Legal Entity data using the GleifSFIntegration class.
* @Author :
* @Last Modified By :
```

```
* @Last Modified On : March 26, 2025
* @Modification Log :
*==============================================================================
* Ver | Date | Author | Modification
*==============================================================================
* 1.0 | March 26, 2025 | | Initial Version
**/

public with sharing class GleifSearchController {

/**
* @description Retrieves Legal Entity data based on the provided LEI code.
* @param lei The LEI code to search for.
* @return LegalEntity__c The Legal Entity object populated with data from the API.
*/
@AuraEnabled(cacheable=true)
public static LegalEntity__c getLegalEntity(String lei) {
// return [SELECT Id, Name, lei__c,status__c, country__c FROM LegalEntity__c WHERE lei__c =: lei];
System.debug('print lei : ' + lei);
try{
List<LegalEntity__c> ls = GleifSFIntegration.getLegalEntityData(lei);
return ls.get(0); // return only searching entity
}catch(Exception e){
throw new AuraHandledException('Error getting entity');
}
}


}
```

The **GleifSearchController** provides a streamlined way to retrieve **Legal Entity** data based on an **LEI code**. It ensures:

- Efficient API calls through the **GLEIF API**.
- Compatibility with **Lightning Components**.
- Proper error handling to avoid application crashes

2.6 Apex Test class: GleifSearchControllerTest.cls

This is the test class for the **GleifSearchController** Apex controller. It verifies the functionality of the **getLegalEntity** method by simulating an API response from the **GLEIF API** and ensuring that the method correctly processes and returns the expected **LegalEntity__c** records.

Purpose

This test class ensures that the **GleifSearchController** retrieves and processes **GLEIF API** data correctly by performing unit tests using mock HTTP responses.

Description:

- This test method verifies the **getLegalEntity** method by
- Simulating a valid HTTP response using **Test.setMock**.
- Calling **getLegalEntity** with a specific **LEI**
- Asserting that the returned **LegalEntity__c** object contains the correct data.
- If the **getLegalEntity** method correctly processes the API response, the assertions will pass.
- If there is an issue with JSON parsing or incorrect API handling, the test will fail, providing feedback for debugging.

```
/**
* @File Name : GleifSearchControllerTest.cls
* @Description : Test class for GleifSearchController
* @Author :
* @Last Modified By :
* @Last Modified On : March 26, 2025
* @Modification Log :
*==============================================================================
* Ver | Date | Author | Modification
*==============================================================================
* 1.0 | March 26, 2025 | | Initial Version
**/
@isTest
private class GleifSearchControllerTest {

@isTest
static void testGetLegalEntity() {
String mockResponse = '{"data": [{"attributes": {"entity": {"legalName": {"name": "Test Company"}, "legalAddress":
{"country": "LU"}}, "lei": "222100EG1QRYPH6C8D53", "registration": {"status": "ISSUED"}}}]}';
Test.setMock(HttpCalloutMock.class, new GleifSFIntegrationTest.MockHttpResponseGenerator(200, 'OK',
mockResponse));

LegalEntity__c legalEntity = GleifSearchController.getLegalEntity('222100EG1QRYPH6C8D53');
System.assertEquals('Test Company', legalEntity.Name);
System.assertEquals('222100EG1QRYPH6C8D53', legalEntity.lei__c);
System.assertEquals('LU', legalEntity.country__c);
System.assertEquals('ISSUED', legalEntity.status__c);
}
}
```

3. Lightning Web Component (LWC) Implementation

3.1 Component Name: searchGleifRecords

Purpose

GleifSearchComp is a Lightning Web Component (LWC) that enables users to search for Global Legal Entity Identifier Foundation (GLEIF) records using a Legal Entity Identifier (LEI). The search results are displayed in a **lightning-datatable** format.

- Accepts an **LEI code** as input from the user.
- Calls an **Apex method (searchGleifRecords)** to fetch entity data from the backend.
- Displays the retrieved data in a **datatable**.
- Implements basic **error handling and validation**.

Html Structure:

- A **lightning-card** with title "GLEIF Search".
- A **lightning-input** for user input (LEI code), restricted to 20 characters.
- A **lightning-button** for triggering the search.
- A **lightning-datatable** to display results (conditionally rendered).
- An **error message** section (conditionally displayed).

SearchGleifRecords.html

```
<template>
<lightning-card title="GLEIF Search" icon-name="utility:search">
<div class="slds-m-around_medium">
<lightning-input
label="Enter Lei Code"
value={searchKey}
onchange={handleInputChange}
max-length="20">
</lightning-input>
<lightning-button
label="Search"
variant="brand"
onclick={handleSearch}
class="slds-m-top_medium">
</lightning-button>
</div>

<template if:true={gleifData}>
<lightning-datatable
key-field="Id"
data={gleifData}
columns={columns}
hide-checkbox-column="true">
</lightning-datatable>
</template>

<template if:true={error}>
<p class="slds-text-color_error">{error}</p>
</template>
</lightning-card>
</template>
```

Javascript Structure:

Imports:

- **LightningElement, track** from 'lwc' - For component state management.
- **searchGleifRecords** from '@salesforce/apex/GleifSearchController.fetchLei' - Calls Apex to fetch GLEIF records.

Tracked Properties:

- @track searchKey → Stores the user input (LEI code).
- @track gleifData → Holds the retrieved records.
- @track error → Stores any error messages.

- columns → Defines the columns for the datatable.

Methods:

- **handleInputChange(event)**: Updates searchKey when the user enters input.
- **handleSearch()**: Calls Apex, retrieves data, and handles errors.

SearchGleifRecords.js

```
/**
* @description GleifSearchComp is a Lightning Web Component that allows users to search for GLEIF records
* using a Legal Entity Identifier (LEI). It displays the search results in a datatable.
*/
import { LightningElement, track } from 'lwc';
import searchGleifRecords from '@salesforce/apex/GleifSearchController.fetchLei';


// Define the columns for the datatable.
const COLUMNS = [
{ label: 'Name', fieldName: 'Name' },
{ label: 'LEI CODE', fieldName: 'LEI__c' },
{ label: 'Status', fieldName: 'Status__c' },
{ label: 'Country', fieldName: 'Country__c' }
];


export default class GleifSearchComp extends LightningElement {
// Track the search key entered by the user.
@track searchKey = '';
// Track the data retrieved from Apex.
@track gleifData;
// Track any errors that occur during the search.
@track error;
// Assign the defined columns to the datatable.
columns = COLUMNS;


// Handle input change event and update the search key.
handleInputChange(event) {
this.searchKey = event.target.value;
}


// Handle the search button click.
handleSearch() {
// Validate the length of the search key.
if (this.searchKey.length > 20) {
```

```
this.error = 'Input exceeds 20 characters. Please shorten it.';
return; // Exit the function if validation fails.
}
// Clear any previous errors.
this.error = null;
// Call the Apex method to search for GLEIF records.
searchGleifRecords({ lei: this.searchKey })
.then(result => {
// Assign the result to the gleifData property.
this.gleifData = result;
})
.catch(error => {
// Handle any errors that occur during the Apex call.
this.error = 'Error retrieving data.';
// Clear the gleifData property.
this.gleifData = null;
});
}
}
```

3.2 Component Name: searchGleifRecords.test.js

This is a Jest-based test suite designed to validate the behavior of the GleifSearchComp Lightning Web Component (LWC). The component interacts with the GLEIF API via an Apex method and displays the results in a datatable. This test suite covers multiple scenarios, such as input validation, successful Apex responses, and error handling.

Implementation:

- **Unit testing**: Verifies the functionality of the GleifSearchComp component.
- **Mocking**: Mocks the Apex call to the backend (fetchLei), using jest.mock to simulate the API response.
- **Test scenarios**: Includes tests for input changes, search functionality, success/failure of the Apex method, and validation for input constraints.

Imports:

- **createElement**: Utility from LWC to create an instance of the component.
- **GleifSearchComp**: The Lightning Web Component being tested.
- **searchGleifRecords**: The Apex method that queries the GLEIF API.

SearchGleifRecords.test.js

```
// gleifSearchComp.test.js
import { createElement } from 'lwc';
import GleifSearchComp from 'c/gleifSearchComp';
import searchGleifRecords from '@salesforce/apex/GleifSearchController.fetchLei';

// Mock Apex wire adapter
jest.mock(
'@salesforce/apex/GleifSearchController.fetchLei',
() => {
const { createApexTestWireAdapter } = require('@salesforce/sfdx-lwc-jest');
return {
default: createApexTestWireAdapter(jest.fn())
};
},
{ virtual: true }
);

describe('c-gleif-search-comp', () => {
afterEach(() => {
// The jsdom instance is shared across test cases in a single file so reset the DOM
while (document.body.firstChild) {
document.body.removeChild(document.body.firstChild);
}
jest.clearAllMocks();
});

it('should update searchKey when input changes', () => {
const element = createElement('c-gleif-search-comp', {
is: GleifSearchComp
});
document.body.appendChild(element);

const inputElement = element.shadowRoot.querySelector('lightning-input');
inputElement.value = 'test';
inputElement.dispatchEvent(new CustomEvent('change'));

return Promise.resolve().then(() => {
expect(element.searchKey).toBe('test');
});
});

it('should call Apex method with correct searchKey on handleSearch', () => {
const element = createElement('c-gleif-search-comp', {
is: GleifSearchComp
});
document.body.appendChild(element);

element.searchKey = 'testLei';
element.handleSearch();

return Promise.resolve().then(() => {
expect(searchGleifRecords.mock.calls.length).toBe(1);
expect(searchGleifRecords.mock.calls[0][0]).toEqual({ lei: 'testLei' });
});
});

it('should set gleifData on successful Apex call', () => {
const element = createElement('c-gleif-search-comp', {
is: GleifSearchComp
});
document.body.appendChild(element);

const mockData = [{ Name: 'Test Name', LEI__c: 'testLei', Status__c: 'Active', Country__c: 'US' }];
searchGleifRecords.mockResolvedValue(mockData);

element.searchKey = 'testLei';
element.handleSearch();

return Promise.resolve().then(() => {
expect(element.gleifData).toEqual(mockData);
});
});

it('should set error on failed Apex call', () => {
const element = createElement('c-gleif-search-comp', {
is: GleifSearchComp
});
document.body.appendChild(element);

searchGleifRecords.mockRejectedValue(new Error('Test Error'));

element.searchKey = 'testLei';
element.handleSearch();

return Promise.resolve().then(() => {
expect(element.error).toBe('Error retrieving data.');
expect(element.gleifData).toBeNull();
});
});

it('should set error when searchKey length exceeds 20', () => {
const element = createElement('c-gleif-search-comp', {
is: GleifSearchComp
});
document.body.appendChild(element);

element.searchKey = 'thisIsALongSearchKeyThatExceeds20Characters';
element.handleSearch();
```

```
return Promise.resolve().then(() => {
expect(element.error).toBe('Input exceeds 20 characters. Please shorten it.');
});
});
});
```

This Jest test validates the primary functionality of the GleifSearchComp Lightning Web Component, ensuring that it behaves correctly in different scenarios, such as:

- Updating the search key input.
- Calling the Apex method with the correct search key.
- Handling both successful and failed responses from the Apex method.
- Validating the input length constraint.

The tests cover edge cases and simulate realistic user interactions to ensure robust functionality of the component.

3.3 How to Use the LWC Component

1. Navigate to the **GLEIF Search** component.
2. Enter a valid **LEI Code**.
3. Click **Search**.
4. View results in the table.
5. If an error occurs, an error message will be displayed.

4. Deployment and GitHub Versioning

4.1 GitHub Repository

- Created a public repository : https://github.com/lola-gliefSF/gliefrepo
- Pushed Salesforce project. (jsdx)
- Created following 2 new branches:
  - apex-glief for Apex development.
  - lwc-glief for LWC development.

1) Initialized repository

```
git init
git remote add origin https://github.com/lola-gliefSF/gliefrepo.git
git add .
git commit -m "My initial comments"
git push origin main
```

2) Created branches

Create a new branch for **Apex**, with: `git checkout -b apex- implementation`
Create a new branch for **LWC**, with: `git checkout -b lwc- implementation`

Pushed each branch to **GitHub**, with :

```
git push origin apex- implementation
git push origin lwc- implementation
```

3) Committed changes to the corresponding branches

```
git checkout lwc- implementation

git add .

git commit -m "Commited LWC component"

git push origin lwc- implementation


git checkout apex- implementation

git add .

git commit -m "Commited Apex classes"

git push origin apex- implementation
```

3) Merged changes to the main branche

```
git checkout main

git pull origin main

git merge lwc- implementation

git push origin main

git merge apex- implementation

git push origin main
```

2) Deploy to Salesforce

Created a scratch org to deploy the project: https://force-force-3426-dev-ed.scratch.my.salesforce.com/

Authenticate to Salesforce org with: sfdx force:auth:web:login -a https://force-force-3426-dev-ed.scratch.my.salesforce.com

Deploy Metadata: sfdx force:source:deploy -p force-app/main/default/ -u https://force-force-3426-dev-ed.scratch.my.salesforce.com/

Run Apex tests after deployment: sfdx force:apex:test:run -u https://force-force-3426-dev-ed.scratch.my.salesforce.com/ -r human -w 10

5. Conclusion

This implementation enables the efficient and reusable integration of the GLEIF API with Salesforce using Apex and LWC, ensuring a seamless and user-friendly experience.