

Library Management Phase 7 — Integration & External Access (Implemented Parts)

Step 1: Create an External Credential (one time)

Go to **Setup** → **Quick Find** → type **External Credentials** → click **External Credentials**.

Click **New External Credential**.

Fill the form:

- **Label:** GoogleMapsAPIAuth
- **Name:** GoogleMapsAPIAuth
- **Authentication Protocol:** No Authentication

Click **Save**.

You now have an External Credential called **GoogleMapsAPIAuth**.

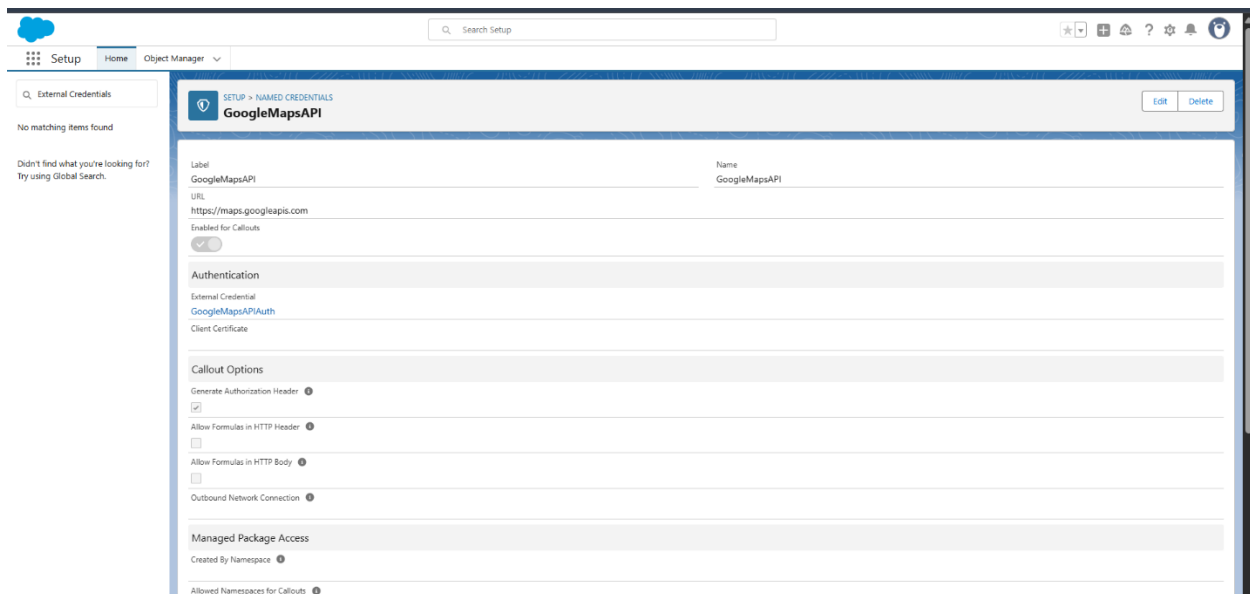
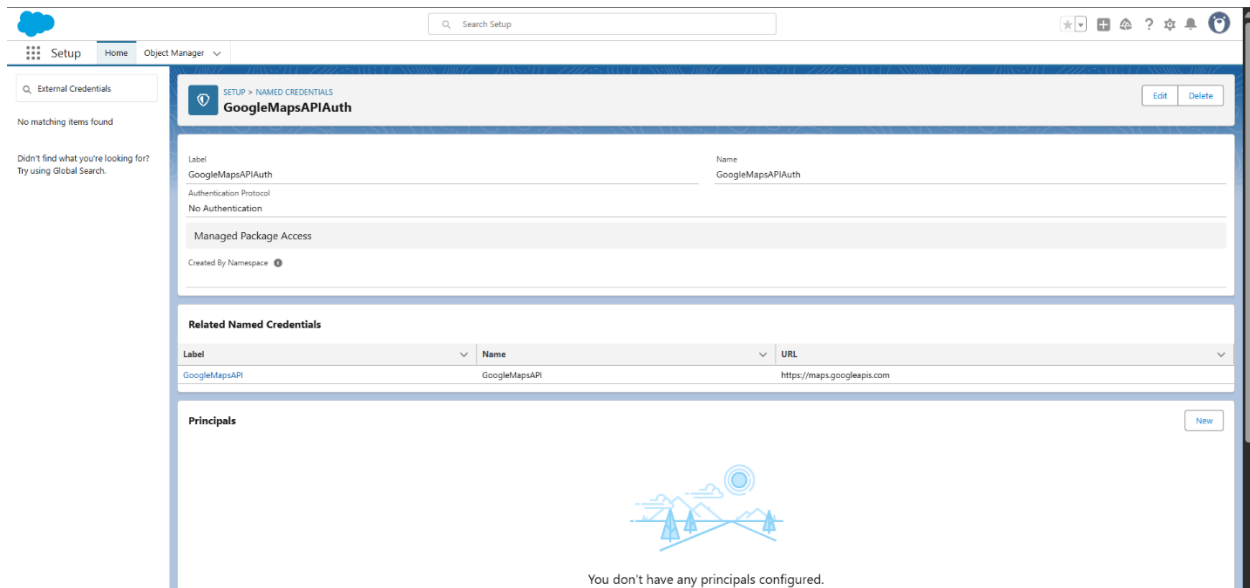
Step 2: Create the Named Credential (link to External Credential)

Go to **Setup** → **Quick Find** → **Named Credentials** → click **Named Credentials** → click **New**.

Fill details:

- **Label:** GoogleMapsAPI
- **Name:** GoogleMapsAPI
- **External Credential:** pick **GoogleMapsAPIAuth**
- **URL:** https://maps.googleapis.com
- **Generate Authorization Header:** leave checked
- **Authentication:** nothing extra (because using API key)

Click **Save**.



```
public with sharing class LibraryMapsService {
    // Returns raw JSON string or you can parse as needed
    public static String getLocationData(String branchAddress) {
        HttpRequest req = new HttpRequest();
        req.setEndpoint(
            'callout:GoogleMapsAPI/maps/api/geocode/json?address=' +
            EncodingUtil.urlEncode(branchAddress,'UTF-8') +
            '&key=YOUR_KEY'
        );
        req.setMethod('GET');
```

```

    HttpResponse res = new Http().send(req);

    // Parse JSON if needed
    Map<String,Object> result =
        (Map<String,Object>) JSON.deserializeUntyped(res.getBody());

    // For now just return raw response
    return res.getBody();
}
}

```

The screenshot displays the Salesforce Developer Console and Setup interface. The top section shows the code editor for the `LibraryMapsService.apxc` file, which contains the following code:

```

1 public with sharing class LibraryMapsService {
2     // Returns raw JSON string or you can parse as needed
3     public static String getLocationData(String branchAddress) {
4         HttpRequest req = new HttpRequest();
5         req.setEndpoint(
6             'callout:GoogleMapsAPI/maps/api/geocode/json?address=' +
7             EncodingUtil.urlEncode(branchAddress, 'UTF-8') +
8             '&key=YOUR_KEY'
9         );
10        req.setMethod('GET');
11
12        HttpResponse res = new Http().send(req);
13
14        // Parse JSON if needed
15        Map<String,Object> result =
16            (Map<String,Object>) JSON.deserializeUntyped(res.getBody());
17
18        // For now just return raw response
19        return res.getBody();
20    }
21 }

```

The bottom section shows the Setup page for the `LibraryMapsService` Apex Class. The class is active and has a code coverage of 0% (0/11). The class body is displayed, showing the same code as in the editor.

Apex Class Detail

Name	Namespace Prefix	Status	Active
LibraryMapsService		Active	

Class Body

```

1 public with sharing class LibraryMapsService {
2     // Returns raw JSON string or you can parse as needed
3     public static String getLocationData(String branchAddress) {
4         HttpRequest req = new HttpRequest();
5         req.setEndpoint(
6             'callout:GoogleMapsAPI/maps/api/geocode/json?address=' +
7             EncodingUtil.urlEncode(branchAddress, 'UTF-8') +
8             '&key=YOUR_KEY'
9         );
10        req.setMethod('GET');
11
12        HttpResponse res = new Http().send(req);
13
14        // Parse JSON if needed
15        Map<String,Object> result =
16            (Map<String,Object>) JSON.deserializeUntyped(res.getBody());
17
18        // For now just return raw response
19        return res.getBody();
20    }
21 }

```

An LWC is always 3 separate files inside force-app/main/default/lwc/libraryMap/:

- libraryMap.html
- libraryMap.js
- libraryMap.js-meta.xml

Here's the **correct code** for your libraryMap LWC — all in one answer:

libraryMap.html

```
<template>

  <lightning-card title="Nearest Library Branch">

    <div class="slds-p-around_medium">

      <iframe

        src={mapUrl}

        width="100%"

        height="300"

        style="border:0;"

        allowfullscreen

        loading="lazy">

      </iframe>

    </div>

  </lightning-card>

</template>
```

libraryMap.js

```
import { LightningElement, api } from 'lwc';
```

```
export default class LibraryMap extends LightningElement {
```


```

@api branchAddress;

mapUrl;

connectedCallback() {
    // Directly build Google Maps iframe URL
    this.mapUrl =
        'https://www.google.com/maps/embed/v1/place?key=YOUR_KEY&q=' +
        this.branchAddress;
}
}

```

 libraryMap.js-meta.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>59.0</apiVersion>
    <isExposed>true</isExposed>
    <targets>
        <target>lightning__RecordPage</target>
        <target>lightning__AppPage</target>
        <target>lightning__HomePage</target>
    </targets>
    <targetConfigs>
        <targetConfig targets="lightning__RecordPage,lightning__AppPage,lightning__HomePage">
            <property name="branchAddress" type="String" label="Branch Address"
description="Address to show on map"/>

```

</targetConfig>

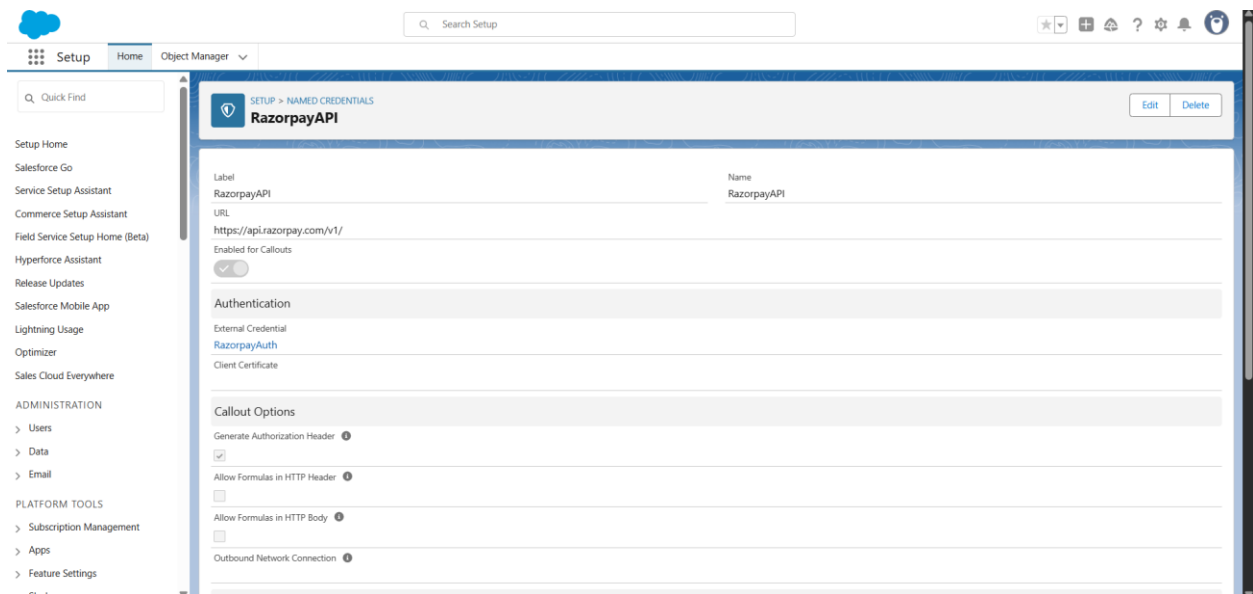
</targetConfigs>

</LightningComponentBundle>

Step 3:

Create a Named Credential

1. In Setup, search **Named Credentials** → **New**.
2. Fill:
 - **Label:** RazorpayAPI
 - **Name:** RazorpayAPI
 - **External Credential:** **RazorpayAuth** (the one you just made)
 - **URL:** <https://api.razorpay.com/v1/>
 - Leave “Generate Authorization Header” checked.
3. Save.



public with sharing class RazorpayPaymentService {

```

public static String createPaymentLink() {

    HttpRequest req = new HttpRequest();

    req.setEndpoint('callout:RazorpayAPI/payment_links'); // note plural endpoint

    req.setMethod('POST');

    req.setHeader('Content-Type','application/json');

    req.setBody('{'+

        '"amount": 1000,' +

        '"currency": "INR",' +

        '"customer": {"email": "test@example.com"}'+

        '});

    HttpResponse res = new Http().send(req);

    System.debug('Response: ' + res.getBody());

    return res.getBody();

}
}

```

The screenshot shows the Salesforce Setup interface. On the left is a navigation menu with categories like Email, Custom Code, Apex Classes, Apex Settings, Apex Test Execution, Apex Test History, Apex Triggers, Environments, and Jobs. The 'Apex Classes' section is selected. The main content area displays the details for the 'RazorpayPaymentService' class. At the top, there are tabs for 'Class Body', 'Class Summary', 'Version Settings', and 'Trace Flags'. The 'Class Body' tab is active, showing the following code:

```

1 public with sharing class RazorpayPaymentService {
2     public static String createPaymentLink() {
3         HttpRequest req = new HttpRequest();
4         req.setEndpoint('callout:RazorpayAPI/payment_links'); // note plural endpoint
5         req.setMethod('POST');
6         req.setHeader('Content-Type','application/json');
7         req.setBody('{'+
8             '"amount": 1000,' +
9             '"currency": "INR",' +
10            '"customer": {"email": "test@example.com"}'+
11            '});
12     }
13     HttpResponse res = new Http().send(req);
14     System.debug('Response: ' + res.getBody());
15     return res.getBody();
16 }
17 }

```

Below the code, there are buttons for 'Edit', 'Delete', 'Download', 'Security', and 'Show Dependencies'. The class is listed as 'Active' with '0% (0/0)' code coverage. The creator is 'Manish.Umaliya' and the creation date is '9/25/2025, 10:57 PM'. The last modified date is also '9/25/2025, 10:58 PM' by 'Manish.Umaliya'.