

Phase 2: Org Setup & Configuration

Purpose:

The purpose of this phase was to establish a secure and structured Salesforce environment where all insurance claim-related operations could be configured. This setup ensures that data security, access control, and company-wide standards are properly implemented before moving on to data modeling and automation.

(Org Setup & Configuration This phase involves preparing the Salesforce Developer Org for the application build.)

1. Salesforce Edition:

Purpose: The purpose of this phase was to establish a secure and structured Salesforce environment where all insurance claim-related operations could be configured. This setup ensures that data security, access control, and company-wide standards are properly implemented before moving on to data modelling and automation.

Implementation: -

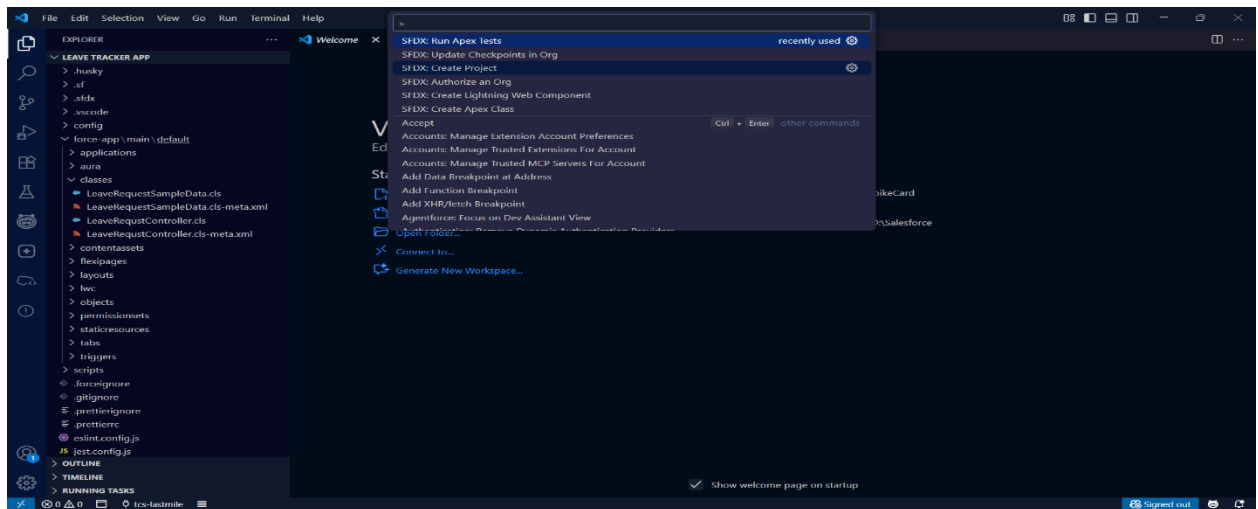
a. Developer Org/Sandbox Creation:

- A Salesforce Developer Org was created for initial configuration and testing.
- Alternatively, a Sandbox can be used to mirror production settings.

To create the org, we are using the lwc (lightning web component/vs code) and also mostly for all the things and implementation like object creation/implementation we are using vs code and then deploy to the org

Step 1: -

- On VScode open command palatte or add (>) symbol and write (SFDX: Create A PROJECT) and name it "Leave Tracker App"

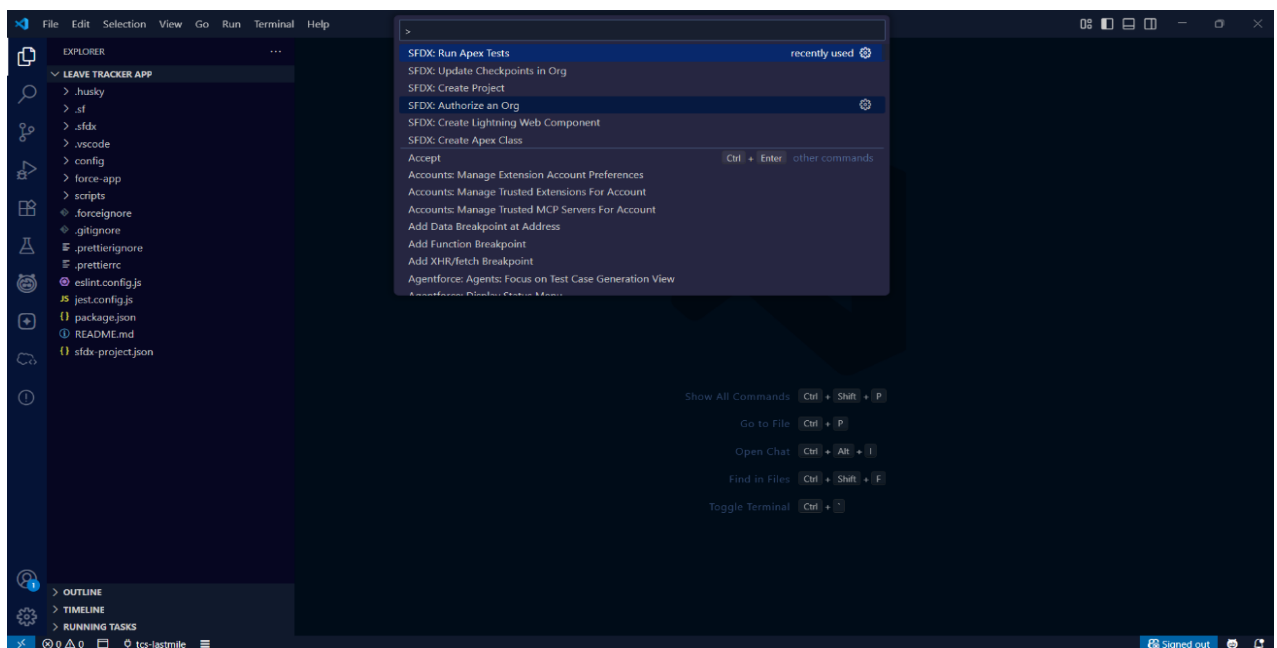


- A new project file will get created and then save it according to the system.

Step 2: -

Authorize the org:

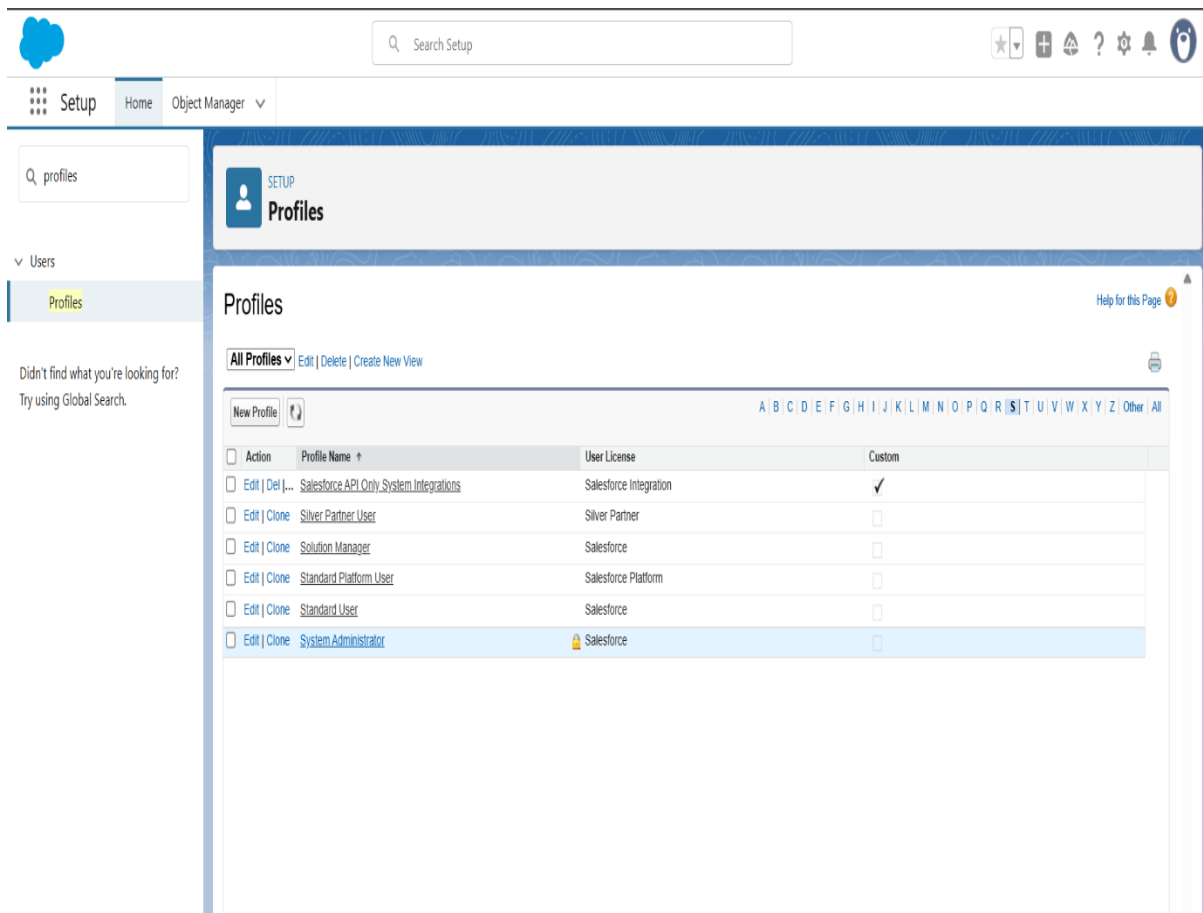
- In this we have to authorize the org to the salesforce account, for that in the same command palette search for (SFDX: Authorize an org).
- Press enter, then a new screen will show on web browser on that we have to login using the salesforce credentials.



- After connecting the account to the project, it will show how many logs have been created to the vs code, so choose according to the account you want to create the project on it.
- Now you have created the project titled “Leave Tracker App” and also the “Salesforce Edition” is also comes under the dev org setup.

2. Permission Sets:

- Customized Profiles to restrict or grant access based on responsibilities.
- Added Permission Sets for advanced tasks like claim approvals.



The screenshot displays the Salesforce Setup interface for Profiles. The left sidebar shows the navigation menu with 'Setup' selected. The main content area is titled 'Profiles' and includes a search bar and a 'New Profile' button. Below this is a table listing various profiles.

Action	Profile Name	User License	Custom
Edit Delete	Salesforce API Only System Integrations	Salesforce Integration	<input checked="" type="checkbox"/>
Edit Clone	Silver Partner User	Silver Partner	<input type="checkbox"/>
Edit Clone	Solution Manager	Salesforce	<input type="checkbox"/>
Edit Clone	Standard Platform User	Salesforce Platform	<input type="checkbox"/>
Edit Clone	Standard User	Salesforce	<input type="checkbox"/>
Edit Clone	System Administrator	Salesforce	<input type="checkbox"/>

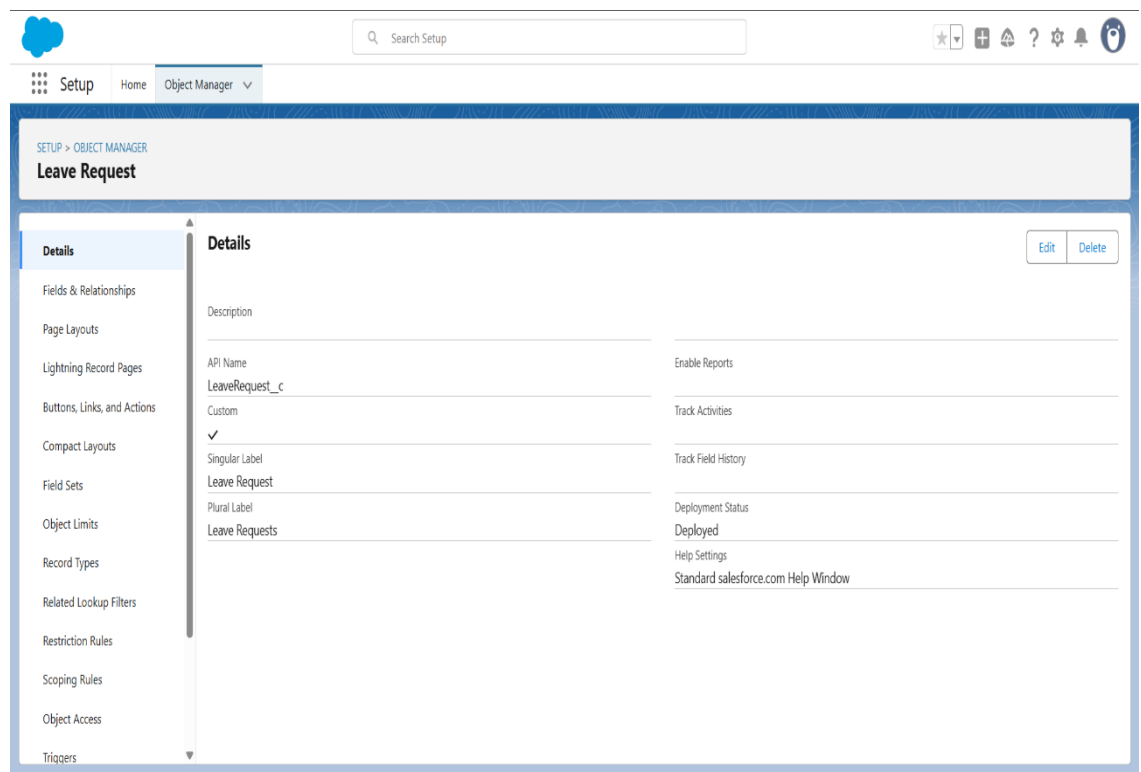
Phase 3: Data Modeling & Relationships


Purpose:

This phase focused on designing the underlying data model to store and manage information about policyholders, policies, and claims. A scalable and logical data structure was essential for enabling claim processing, reporting, and automation.








Implementation:

1. Custom/Standard object creation:
 - a. LeaveRequest__c (includes: From_Date__c, To_Date__c, Manager_Comment__c, Reason__c, Status__c and User__c)





Search Setup



SetupHomeObject Manager

SETUP > OBJECT MANAGER

Leave Request

Details

Fields & Relationships

Page Layouts

Lightning Record Pages

Buttons, Links, and Actions

Compact Layouts

Field Sets

Object Limits

Record Types

Related Lookup Filters

Restriction Rules

Scoping Rules

Object Access

Triggers

Fields & Relationships

10 Items, Sorted by Field Label

Quick Find

New

Deleted Fields

Field Dependencies

Set History Tracking

Created By	CreatedById	Lookup(User)		
From Date	From_Date__c	Date		
Last Modified By	LastModifiedById	Lookup(User)		
Leave Request Id	Name	Auto Number	✓	
Manager Comment	Manager_Comment__c	Text Area(255)		
Owner	OwnerId	Lookup(User.Group)	✓	
Reason	Reason__c	Text Area(255)		
Status	Status__c	Picklist		
To Date	To_Date__c	Date		
User	User__c	Lookup(User)	✓	

Phase 4: Process Automation (Admin) (Only client-side validation)

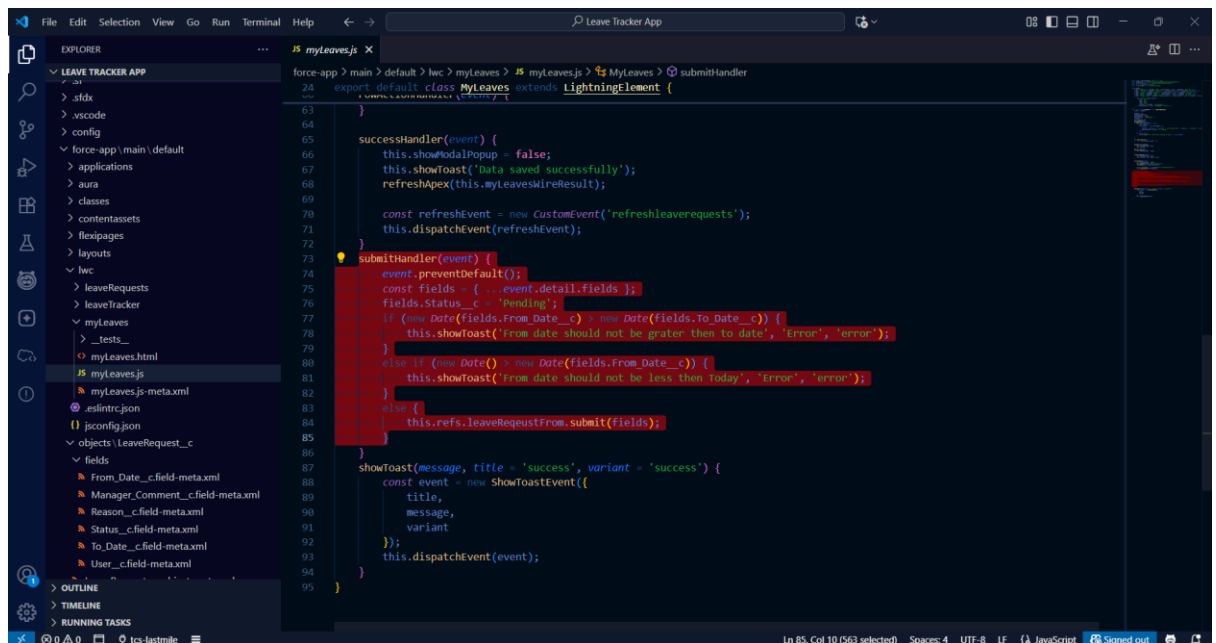
Purpose:

The goal of this phase was to automate business processes to improve efficiency, ensure compliance with rules, and streamline claim submission and approval workflows.

Implementation: -

1. Custom validation using form submit event:

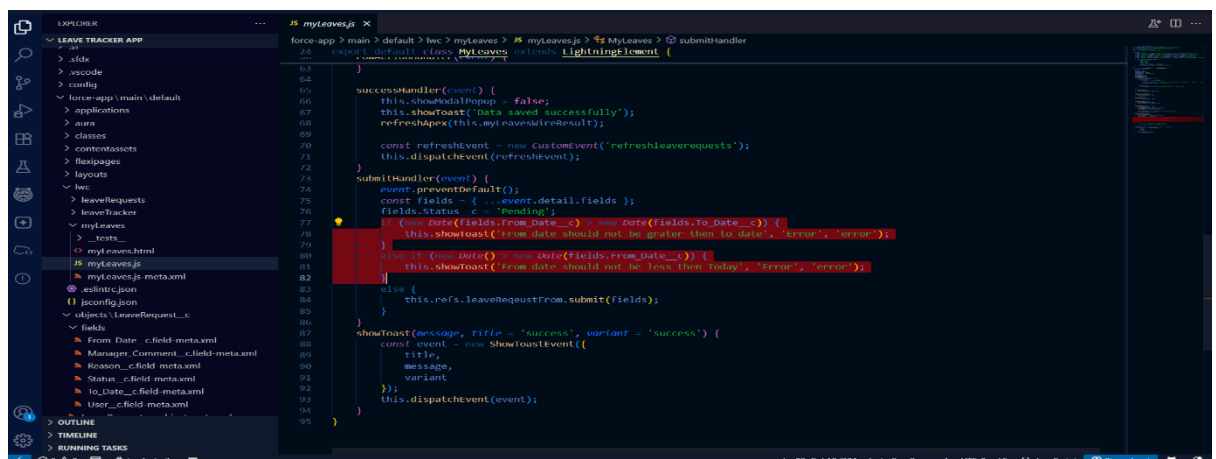
We are using lwc and Vs code to create the validation in the submit handler.



```
24 export default class MyLeaves extends LightningElement {
    63
    64
    65
    66
    67
    68
    69
    70
    71
    72
    73
    74
    75
    76
    77
    78
    79
    80
    81
    82
    83
    84
    85
    86
    87
    88
    89
    90
    91
    92
    93
    94
    95
    submitHandler(event) {
        event.preventDefault();
        const fields = { ...event.detail.fields };
        fields.Status__c = 'Pending';
        if (new Date(fields.From_Date__c) > new Date(fields.To_Date__c)) {
            this.showToast('from date should not be greater then to date', 'Error', 'error');
        }
        else if (new Date() > new Date(fields.From_Date__c)) {
            this.showToast('from date should not be less then today', 'Error', 'error');
        }
        else {
            this.refs.leaveRequestFrom.submit(fields);
        }
    }
    showToast(message, title = 'success', variant = 'success') {
        const event = new ShowToastEvent({
            title,
            message,
            variant
        });
        this.dispatchEvent(event);
    }
}
```

2. Date validation logic:

This is for the logic for the from date to date for both MyLeaves and Request.



```
24 export default class MyLeaves extends LightningElement {
    63
    64
    65
    66
    67
    68
    69
    70
    71
    72
    73
    74
    75
    76
    77
    78
    79
    80
    81
    82
    83
    84
    85
    86
    87
    88
    89
    90
    91
    92
    93
    94
    95
    submitHandler(event) {
        event.preventDefault();
        const fields = { ...event.detail.fields };
        fields.Status__c = 'Pending';
        if (new Date(fields.From_Date__c) > new Date(fields.To_Date__c)) {
            this.showToast('from date should not be greater then to date', 'Error', 'error');
        }
        else if (new Date() > new Date(fields.From_Date__c)) {
            this.showToast('from date should not be less then today', 'Error', 'error');
        }
        else {
            this.refs.leaveRequestFrom.submit(fields);
        }
    }
    showToast(message, title = 'success', variant = 'success') {
        const event = new ShowToastEvent({
            title,
            message,
            variant
        });
        this.dispatchEvent(event);
    }
}
```

```
force-app > main > default > lwc > myLeaves > JS myLeaves.js > MyLeaves > submitHandler

24 export default class MyLeaves extends LightningElement {
25
26 }
27
28 popupCloseHandler() {
29     this.showModalPopup = false;
30 }
31
32 rowActionHandler(event) {
33     this.showModalPopup = true;
34     this.recordId = event.detail.rowId;
35 }
36
37 successHandler(event) {
38     this.showModalPopup = false;
39     this.showToast('Data saved successfully');
40     refreshApex(this.myLeavesWireResult);
41
42     const refreshEvent = new CustomEvent('refreshleaveRequests');
43     this.dispatchEvent(refreshEvent);
44 }
45
46 submitHandler(event) {
47     event.preventDefault();
48
49     const fields = { ...event.detail.fields };
50     fields.Status__c = 'Pending';
51
52     if (new Date(fields.From_Date__c) > new Date(fields.To_Date__c)) {
53         this.showToast('From date should not be greater than to date', 'Error', 'error');
54     }
55     else if (new Date() > new Date(fields.From_Date__c)) {
56         this.showToast('From date should not be less than today', 'Error', 'error');
57     }
58     else {
59         this.refs.leaveRequestForm.submit(fields);
60     }
61 }
62
63 showToast(message, title = 'success', variant = 'success') {
64     const event = new ShowToastEvent({
65         title,
66     });
67 }
```

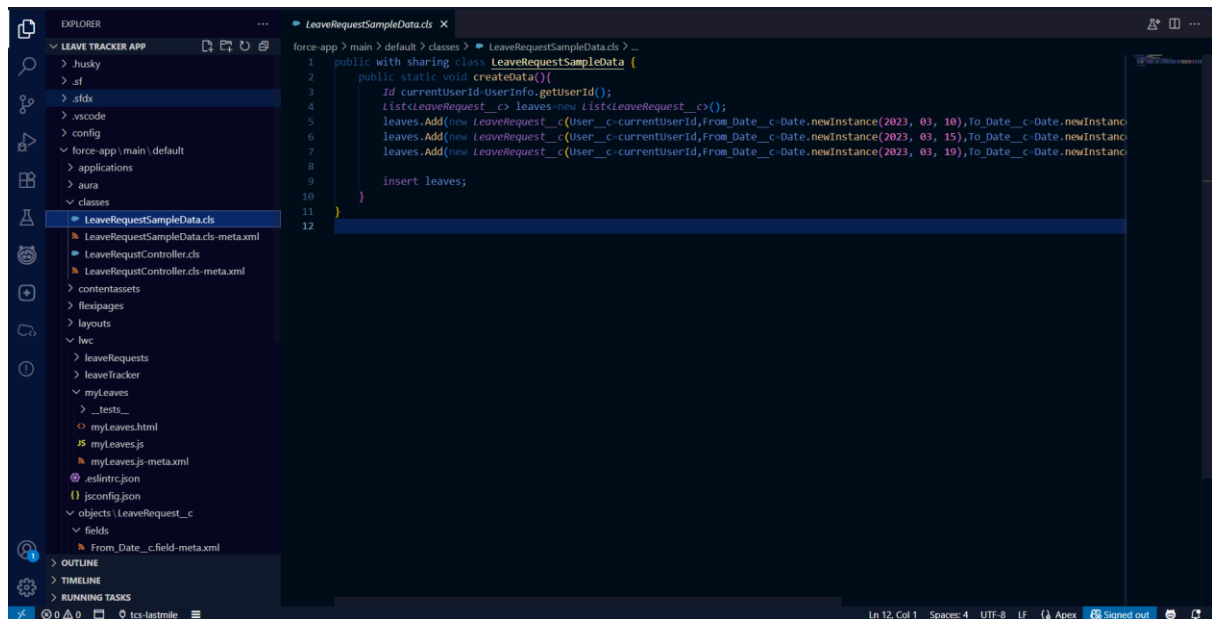
Ln 74, Col 32 (335 selected) Spaces: 4 UTF-8 LF JavaScript Signed out

Phase 5: Apex Programming (Developer)

Purpose: Custom logic was required to extend Salesforce's standard functionality. Apex programming allowed automation of tasks not possible with declarative tools.

Implementation:

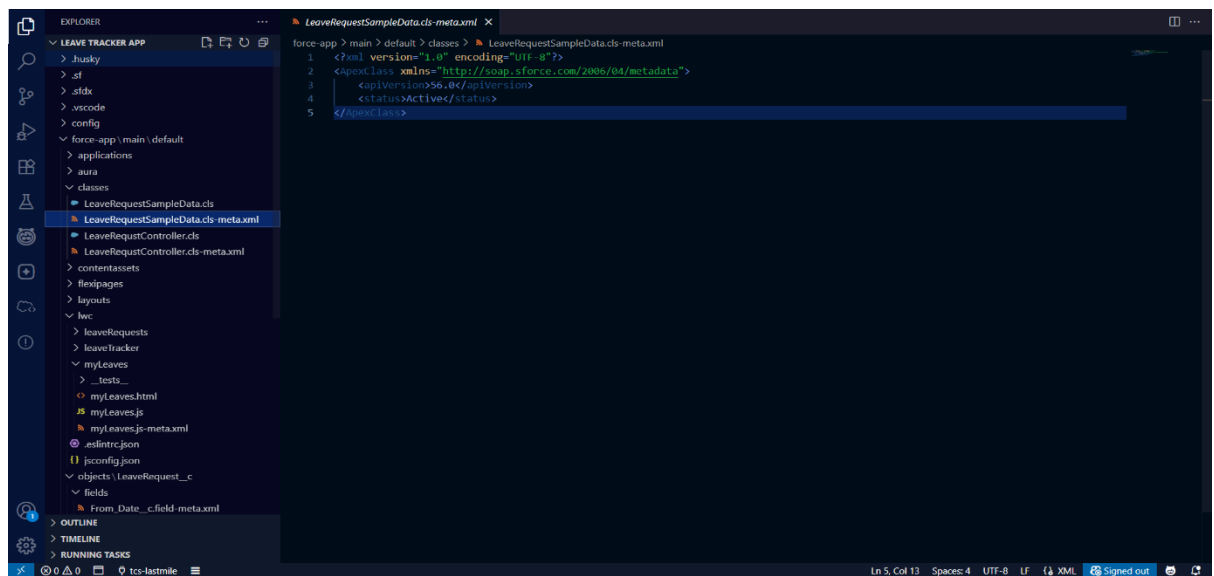
1. Classes & Object (Apex classes setup:
 - a. LeaveRequestSampleData
 - b. LeaveRequestController



The screenshot shows the Visual Studio Code editor with the Explorer pane on the left displaying the project structure. The file 'LeaveRequestSampleData.cls' is selected. The main editor area shows the following Apex code:

```
1 public with sharing class LeaveRequestSampleData {  
2     public static void createData(){  
3         Id currentUserId = UserInfo.getUserId();  
4         List<LeaveRequest__c> leaves = new List<LeaveRequest__c>();  
5         leaves.add(new LeaveRequest__c(User__c: currentUserId, From_Date__c: Date.newInstance(2023, 03, 10), To_Date__c: Date.newInstance(2023, 03, 15), Status__c: 'On Leave'));  
6         leaves.add(new LeaveRequest__c(User__c: currentUserId, From_Date__c: Date.newInstance(2023, 03, 15), To_Date__c: Date.newInstance(2023, 03, 19), Status__c: 'On Leave'));  
7         insert leaves;  
8     }  
9 }  
10  
11  
12
```

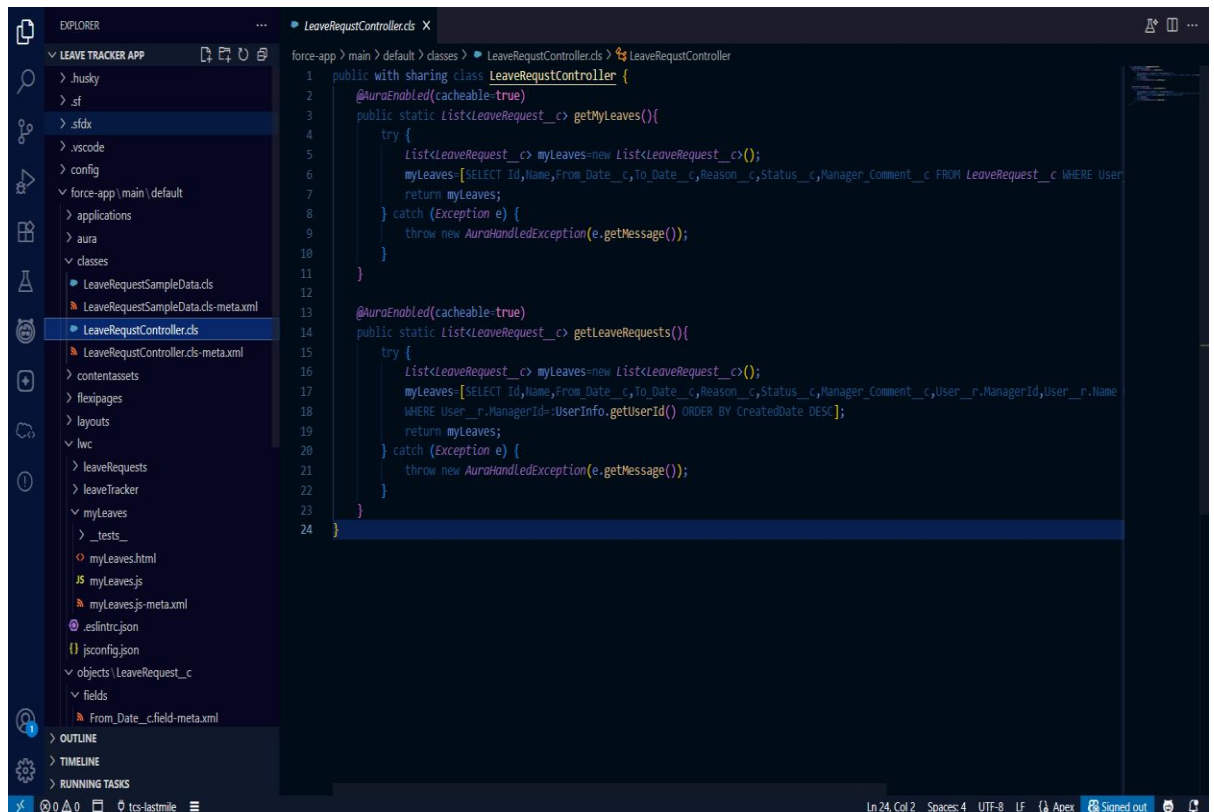
LeaveRequestSampleData.cls



The screenshot shows the Visual Studio Code editor with the Explorer pane on the left displaying the project structure. The file 'LeaveRequestSampleData.cls-meta.xml' is selected. The main editor area shows the following XML metadata:

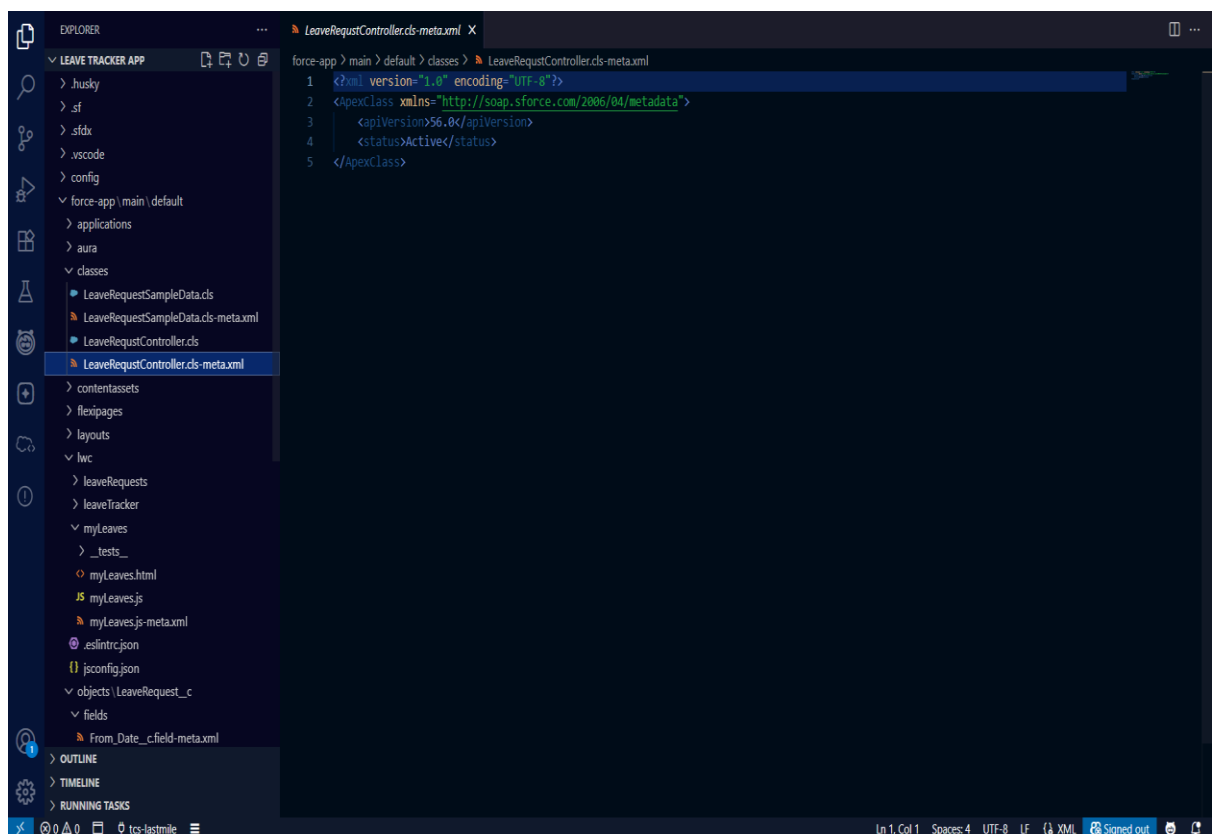
```
1 <?xml version="1.0" encoding="UTF-8"?>  
2 <ApexClass xmlns="http://soap.sforce.com/2006/04/metadata">  
3     <apiVersion>56.0</apiVersion>  
4     <status>Active</status>  
5 </ApexClass>
```

LeaveRequestSampleData.cls-meta.xml



```
1 public with sharing class LeaveRequestController {
2     @AuraEnabled(cacheable=true)
3     public static List<LeaveRequest__c> getMyLeaves(){
4         try {
5             List<LeaveRequest__c> myLeaves=new List<LeaveRequest__c>();
6             myLeaves=[SELECT Id,Name,From_Date__c,To_Date__c,Reason__c,Status__c,Manager_Comment__c FROM LeaveRequest__c WHERE User
7             return myLeaves;
8         } catch (Exception e) {
9             throw new AuraHandledException(e.getMessage());
10        }
11    }
12
13    @AuraEnabled(cacheable=true)
14    public static List<LeaveRequest__c> getLeaveRequests(){
15        try {
16            List<LeaveRequest__c> myLeaves=new List<LeaveRequest__c>();
17            myLeaves=[SELECT Id,Name,From_Date__c,To_Date__c,Reason__c,Status__c,Manager_Comment__c,User__r.ManagerId,User__r.Name
18            WHERE User__r.ManagerId=:UserInfo.getUserId() ORDER BY CreatedDate DESC];
19            return myLeaves;
20        } catch (Exception e) {
21            throw new AuraHandledException(e.getMessage());
22        }
23    }
24 }
```

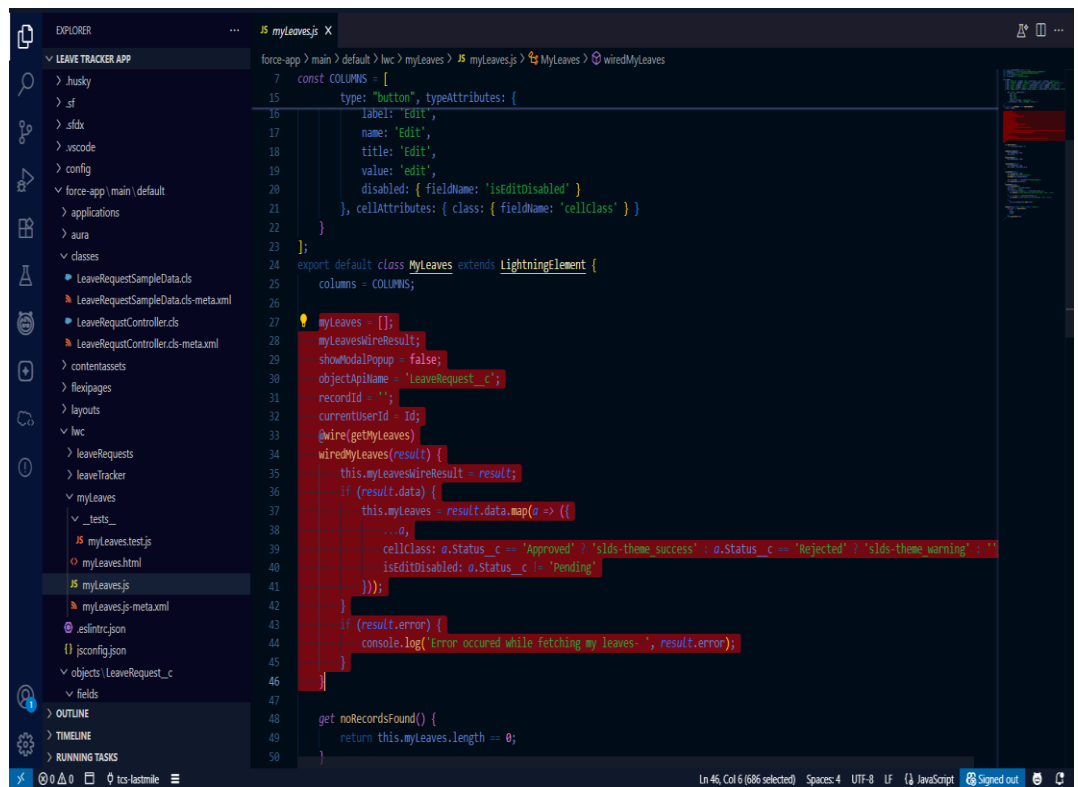
LeaveRequestController.cls



```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <ApexClass xmlns="http://soap.sforce.com/2006/04/metadata">
3     <apiVersion>56.0</apiVersion>
4     <status>Active</status>
5 </ApexClass>
```

LeaveRequestController.cls-meta.xml

- Now to connect to the objects/classes the (@wire) is used to connect



```
force-app > main > default > lwc > myLeaves > JS myLeaves.js > MyLeaves > wiredMyLeaves
7  const COLUMNS = [
15    type: 'button', typeAttributes: {
16      label: 'Edit',
17      name: 'Edit',
18      title: 'Edit',
19      value: 'edit',
20      disabled: { fieldName: 'isEditDisabled' }
21    }, cellAttributes: { class: { fieldName: 'cellClass' } }
22  }
23 ];
24 export default class MyLeaves extends LightningElement {
25   columns = COLUMNS;
26
27   myLeaves = [];
28   myLeavesWireResult;
29   showModalPopup = false;
30   objectApiName = 'LeaveRequest_c';
31   recordId = '';
32   currentUserId = Id;
33   @wire(getMyLeaves)
34   wiredMyLeaves(result) {
35     if (this.myLeavesWireResult = result) {
36       if (result.data) {
37         this.myLeaves = result.data.map(a => ({
38           id: a.Id,
39           cellClass: a.Status_c == 'Approved' ? 'slds-theme success' : a.Status_c == 'Rejected' ? 'slds-theme warning' : '',
40           isEditDisabled: a.Status_c != 'Pending'
41         }));
42       }
43       if (result.error) {
44         console.log('error occurred while fetching my leaves -', result.error);
45       }
46     }
47
48     get noRecordsFound() {
49       return this.myLeaves.length == 0;
50     }
51   }
52 }
```

(note: the LeaveRequestSampleData is a dummy data for testing the app)

Phase 6: User Interface Development

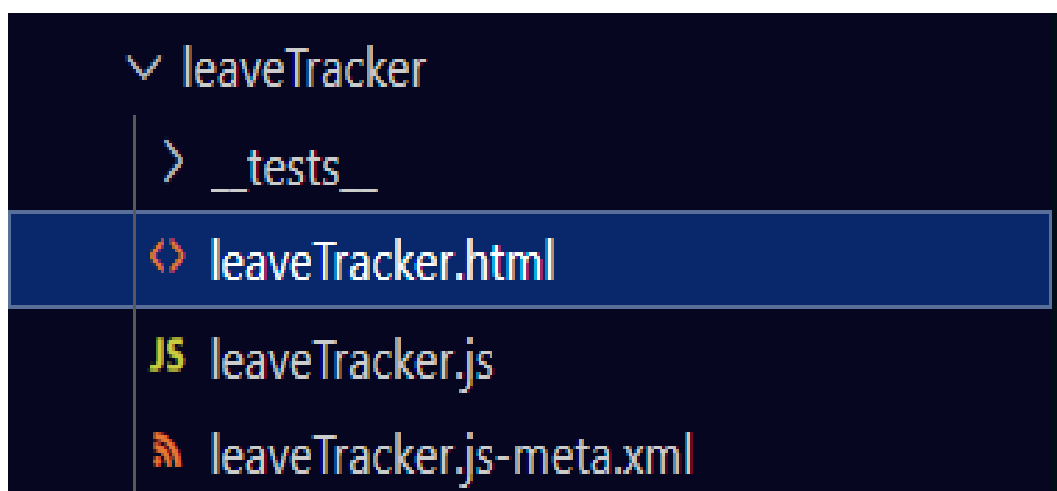
Purpose:

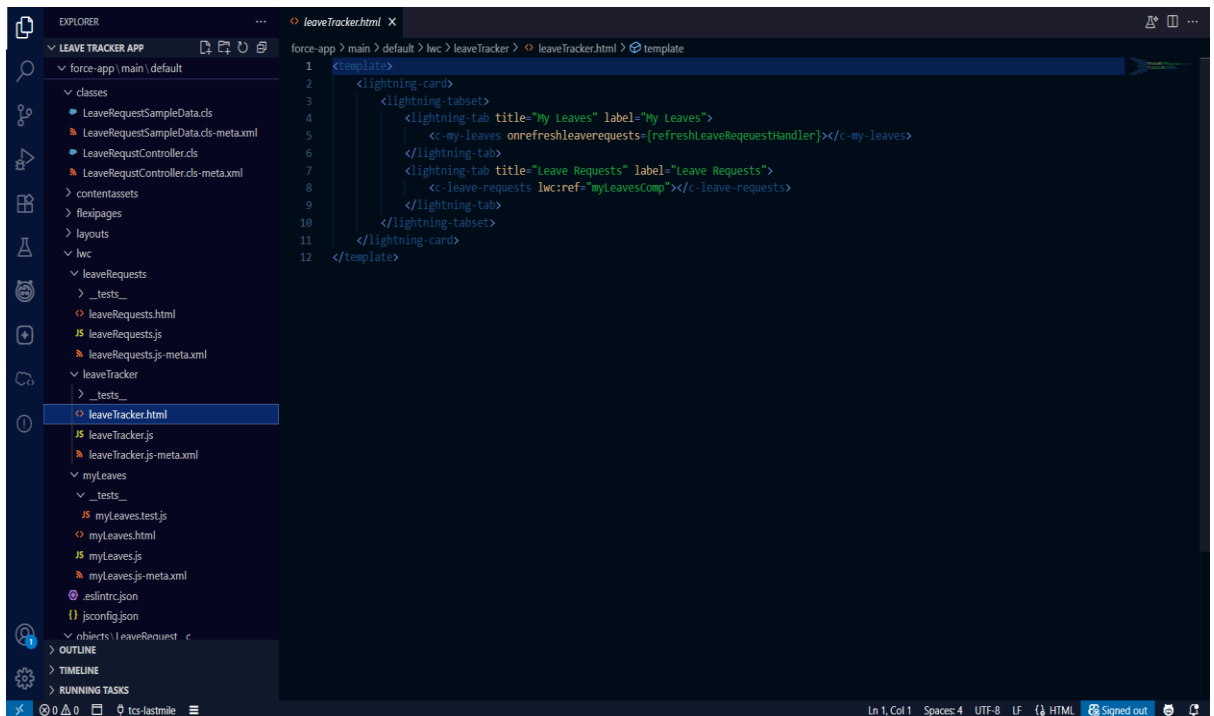
The purpose of this phase was to design an interactive and user-friendly interface for agents, managers, and policyholders using Salesforce Lightning Web Components (LWC). This ensures that claim submissions, claim tracking, and managerial dashboards are intuitive, efficient, and aligned with Salesforce Lightning Design System (SLDS) standards.

Implementation:

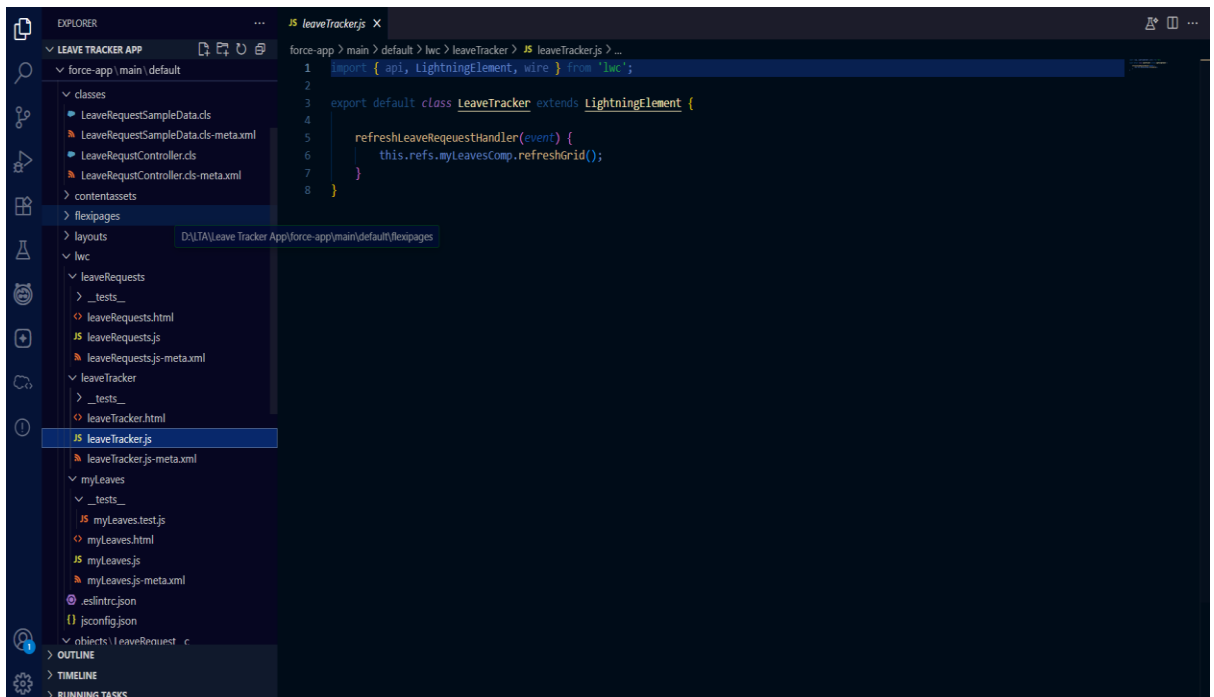
1. Parent component using LWC:

- Using LWC (lightning web component) the parent component is created the project
- The parent component is 'leave Tracker'.
- SFDX: Create Lightning Web Component from the command palette and give the name "leaveTracker".

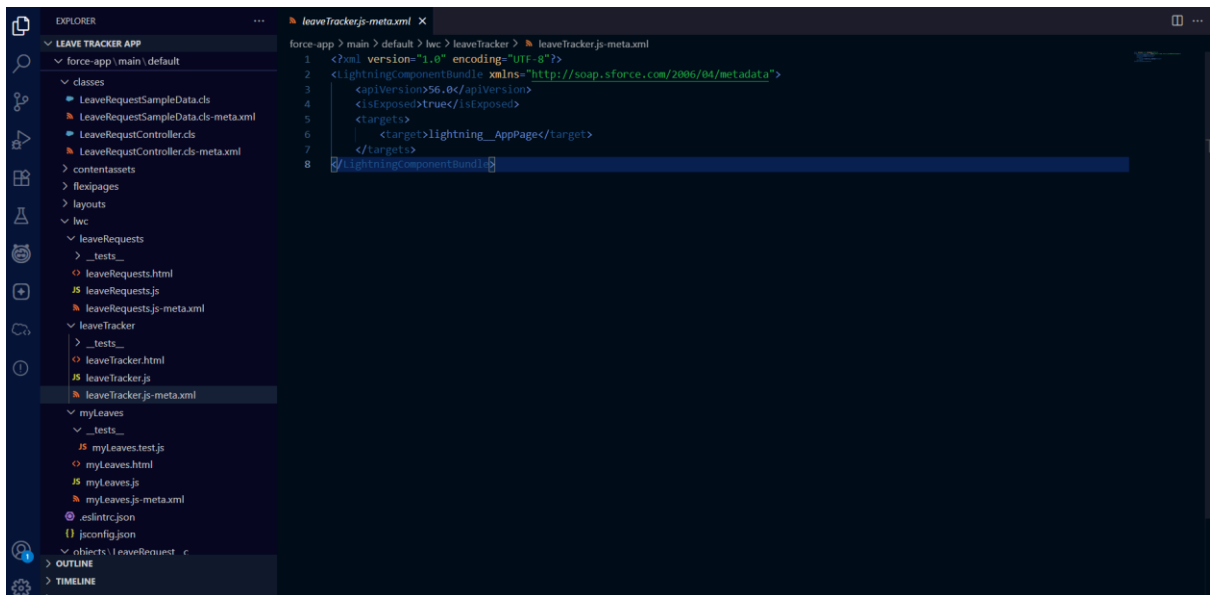




leaveTracker.html



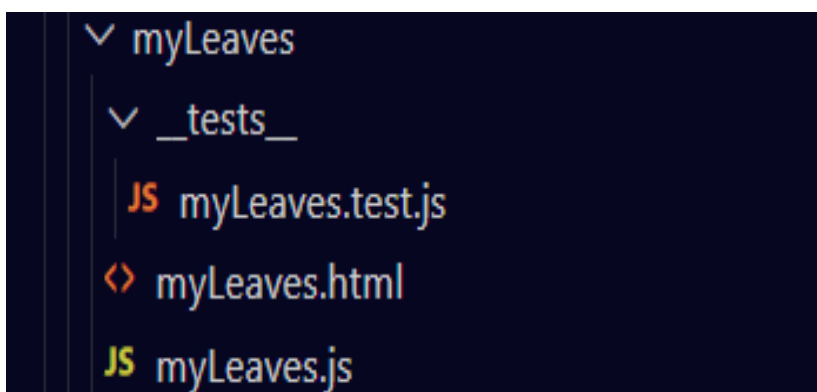
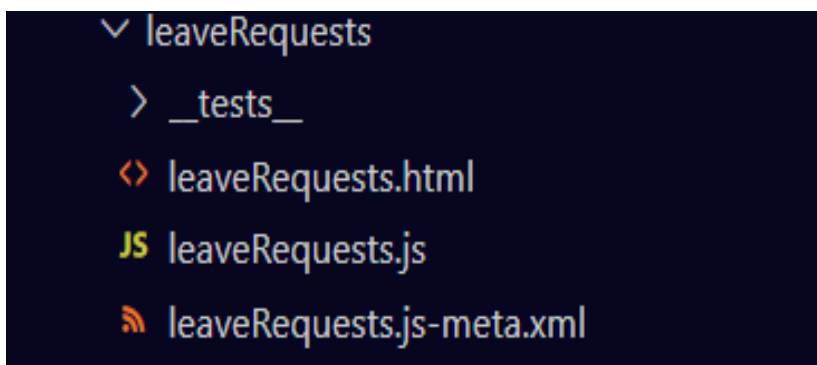
leaveTracker.js



leaveTracker.js-meta.xml

2. Child component using LWC:

- a. leaveRequests
- b. myLeaves



- TheScreenShots for leaveRequests including its code:

```
force-app > main > default > lwc > leaveRequests > leaveRequests.html > template > template > section.slds-modal.slds-fade-in-open.slds-modal_small > div.slds-modal_container > button.slds-button.slds-button_icon.slds-modal_close

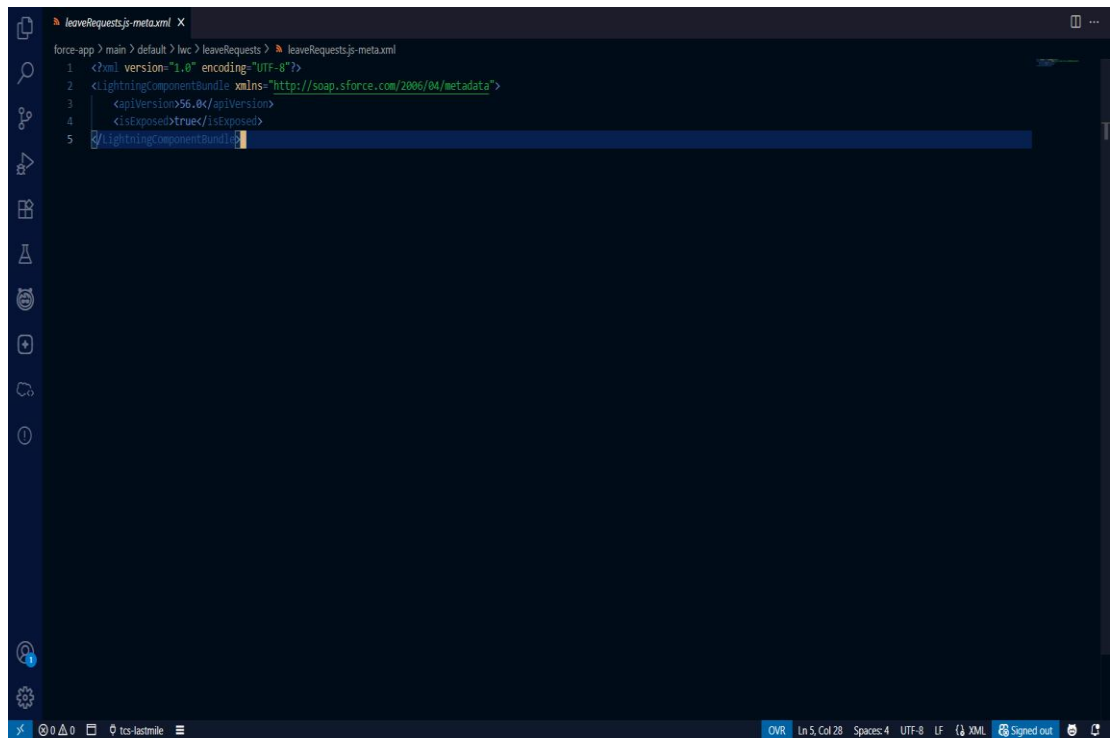
1 <template>
2 <lightning-card>
3 <lightning-datatable key-field="Id" data={leavesRequests} columns={columns}
4 onrowaction={rowActionHandler}></lightning-datatable>
5
6 <template lwc:if={noRecordsFound}>
7 <div class="slds-align_absolute-center slds-p-around_small slds-text-heading_medium">No Records Found</div>
8 </template>
9 </lightning-card>
10
11 <template lwc:if={showModalPopup}>
12 <section role="dialog" tabindex="-1" aria-modal="true" aria-labelledby="modal-heading-01"
13 class="slds-modal slds-fade-in-open slds-modal_small">
14 <div class="slds-modal_container">
15 <button onclick={popupCloseHandler}
16 class="slds-button slds-button_icon slds-modal_close slds-button_icon-inverse">
17 <lightning-icon icon-name="utility:close" alternative-text="close" variant="inverse"
18 size="small"></lightning-icon>
19 <span class="slds-assistive-text">Cancel and close</span>
20 </button>
21 <div class="slds-modal_header">
22 <h1 id="modal-heading-01" class="slds-modal_title slds-hyphenate">Leave Request</h1>
23 </div>
24 <div class="slds-modal_content slds-p-around_medium" id="modal-content-id-1">
25 <lightning-record-edit-form object-api-name={objectApiName} record-id={recordId}
26 onsuccess={successHandler} lwc:ref="leaveRequestForm">
27 <lightning-output-field field-name="User_c"></lightning-output-field>
28 <lightning-output-field field-name="From Date_c"></lightning-output-field>
29 <lightning-output-field field-name="To Date_c"></lightning-output-field>
30 <lightning-output-field field-name="Reason_c"></lightning-output-field>
31 <lightning-input-field field-name="Status_c"></lightning-input-field>
32 <lightning-input-field field-name="Manager Comment_c"></lightning-input-field>
33
34 <div class="slds-var-m-top_medium">
35 <lightning-button variant="brand" type="submit" label="Save">
36 </lightning-button>
37
```

leaveRequests.html

```
force-app > main > default > lwc > leaveRequests > JS leaveRequests.js > @COLUMNS > typeAttributes

1 import { api, LightningElement, wire } from "lwc";
2 import getLeaveRequests from "@salesforce/apex/leaveRequestController.getLeaveRequests";
3 import { ShowToastEvent } from "lightning/platformShowToastEvent";
4 import Id from "@salesforce/user/Id";
5 import { refreshApex } from "@salesforce/apex";
6
7 const COLUMNS = [
8 { label: 'Request Id', fieldName: 'Name', cellAttributes: { class: { fieldName: 'cellClass' } } },
9 { label: 'User', fieldName: 'userName', cellAttributes: { class: { fieldName: 'cellClass' } } },
10 { label: 'From Date', fieldName: 'From Date_c', cellAttributes: { class: { fieldName: 'cellClass' } } },
11 { label: 'To Date', fieldName: 'To Date_c', cellAttributes: { class: { fieldName: 'cellClass' } } },
12 { label: 'Reason', fieldName: 'Reason_c', cellAttributes: { class: { fieldName: 'cellClass' } } },
13 { label: 'Status', fieldName: 'Status_c', cellAttributes: { class: { fieldName: 'cellClass' } } },
14 { label: 'Manager Comment', fieldName: 'Manager Comment_c', cellAttributes: { class: { fieldName: 'cellClass' } } },
15 {
16 type: "button", typeAttributes: {
17 label: 'Edit',
18 name: 'edit',
19 title: 'Edit',
20 value: 'edit',
21 disabled: { fieldName: 'isEditDisabled' },
22 cellAttributes: { class: { fieldName: 'cellClass' } }
23 }
24 ];
25 export default class LeaveRequests extends LightningElement {
26 columns = COLUMNS;
27
28 leavesRequests = [];
29 leavesRequestsWireResult;
30 showModalPopup = false;
31 objectApiName = 'LeaveRequest_c';
32 recordId = '';
33 currentUserId = Id;
34 @wire(getLeaveRequests)
35 wiredMyLeaves(result) {
36 this.leavesRequestsWireResult = result;
37 if (result.data) {
```

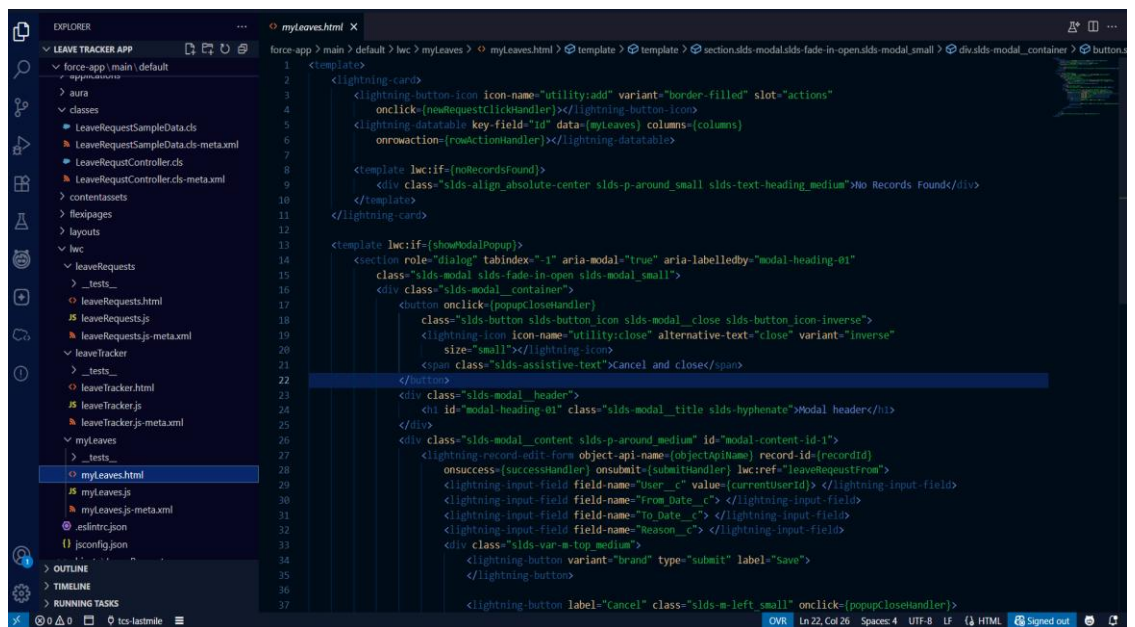
leaveRequests.js



```
force-app > main > default > lwc > leaveRequests > leaveRequests.js-meta.xml
1  <?xml version="1.0" encoding="UTF-8"?>
2  <lightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
3    <apiVersion56.0/>
4    <isExposed>true</isExposed>
5  </lightningComponentBundle>
```

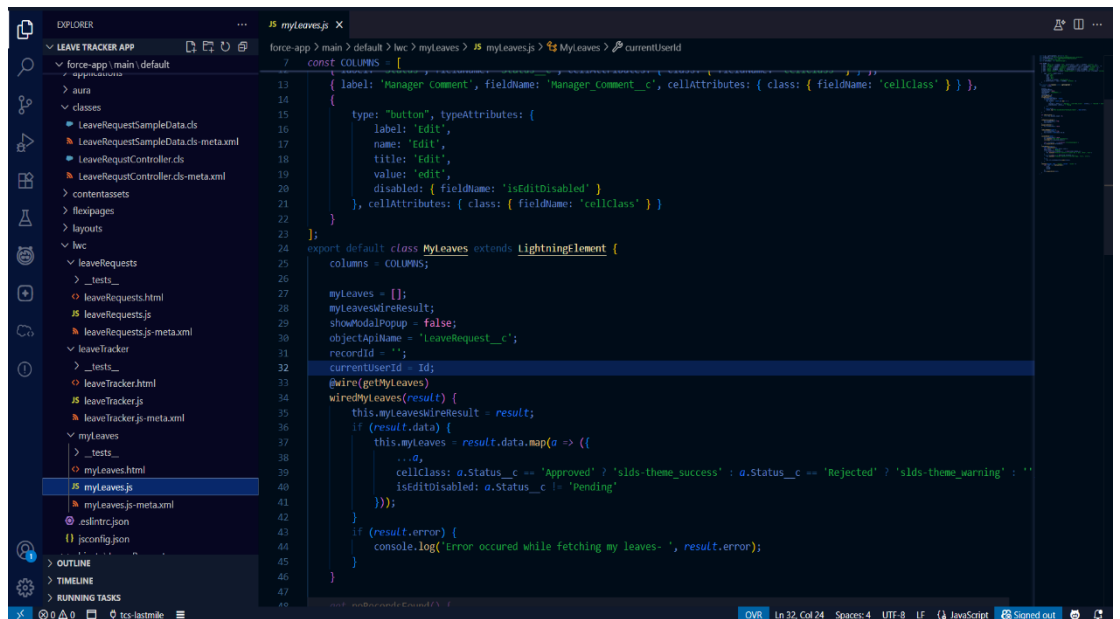
leaveRequests.js-meta.xml

- The ScreenShots for myLeaves:

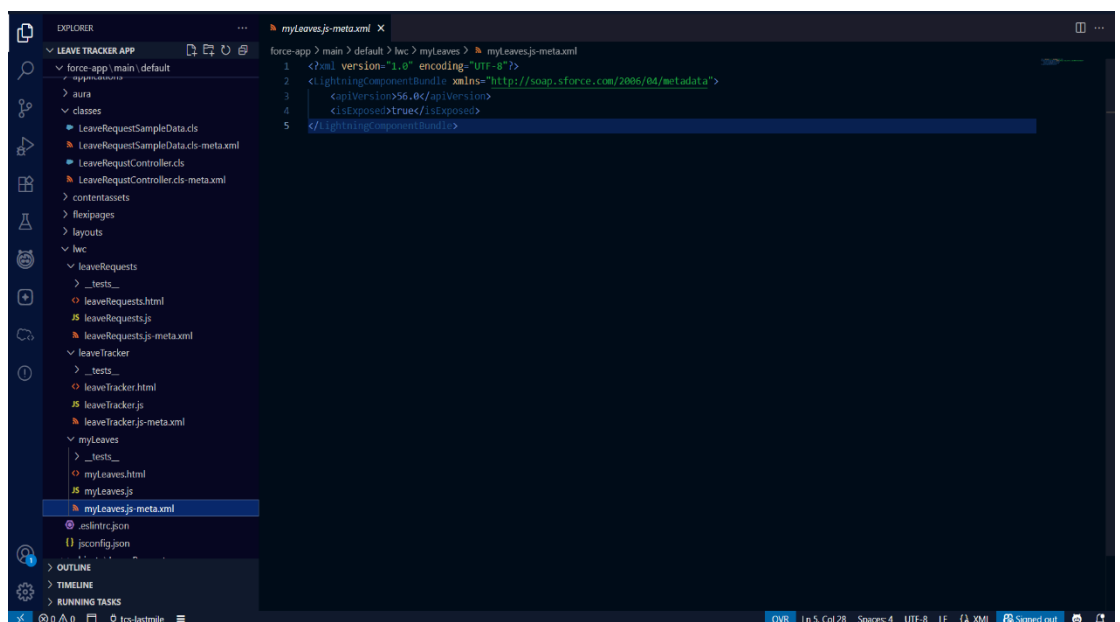


```
force-app > main > default > lwc > myLeaves > myLeaves.html > template > template > section.slds-modal.slds-fade-in-open.slds-modal_small > div.slds-modal_container > button
1  <template>
2  <lightning-card>
3    <lightning-button-icon icon-name="utility:add" variant="border-filled" slot="actions"
4      onclick={newRequestClickHandler}></lightning-button-icon>
5    <lightning-datatable key-field="Id" data={myLeaves} columns={columns}
6      onrowaction={rowActionHandler}></lightning-datatable>
7
8    <template lwc:if={noRecordsFound}>
9      <div class="slds-align_absolute-center slds-p-around_small slds-text-heading_medium">No Records Found</div>
10    </template>
11  </lightning-card>
12
13  <template lwc:if={showModalPopup}>
14    <section role="dialog" tabindex="-1" aria-modal="true" aria-labelledby="modal-heading-01"
15      class="slds-modal slds-fade-in-open slds-modal_small"
16      <div class="slds-modal_container">
17        <button onclick={popupCloseHandler}
18          class="slds-button slds-button-icon slds-modal_close slds-button-icon-inverse">
19          <lightning-icon icon-name="utility:close" alternative-text="close" variant="inverse"
20            size="small"></lightning-icon>
21          <span class="slds-assistive-text">Cancel and close</span>
22        </button>
23        <div class="slds-modal_header">
24          <h1 id="modal-heading-01" class="slds-modal_title slds-hyphenate">Modal header</h1>
25        </div>
26        <div class="slds-modal_content slds-p-around_medium" id="modal-content-id-1">
27          <lightning-record-edit-form object-api-name={objectApiName} record-id={recordId}>
28            <onsuccess={successHandler} onsubmit={submitHandler} lwc:ref="leaveRequestForm">
29              <lightning-input-field field-name="User_c" value={currentUserId}></lightning-input-field>
30              <lightning-input-field field-name="From_Date_c"></lightning-input-field>
31              <lightning-input-field field-name="To_Date_c"></lightning-input-field>
32              <lightning-input-field field-name="Reason_c"></lightning-input-field>
33            <div class="slds-var-w-top_medium">
34              <lightning-button variant="brand" type="submit" label="Save">
35            </lightning-button>
36
37          <lightning-button label="Cancel" class="slds-w-left_small" onclick={popupCloseHandler}>
```

myLeaves.html



myLeaves.js



myLeaves.js-meta.xml

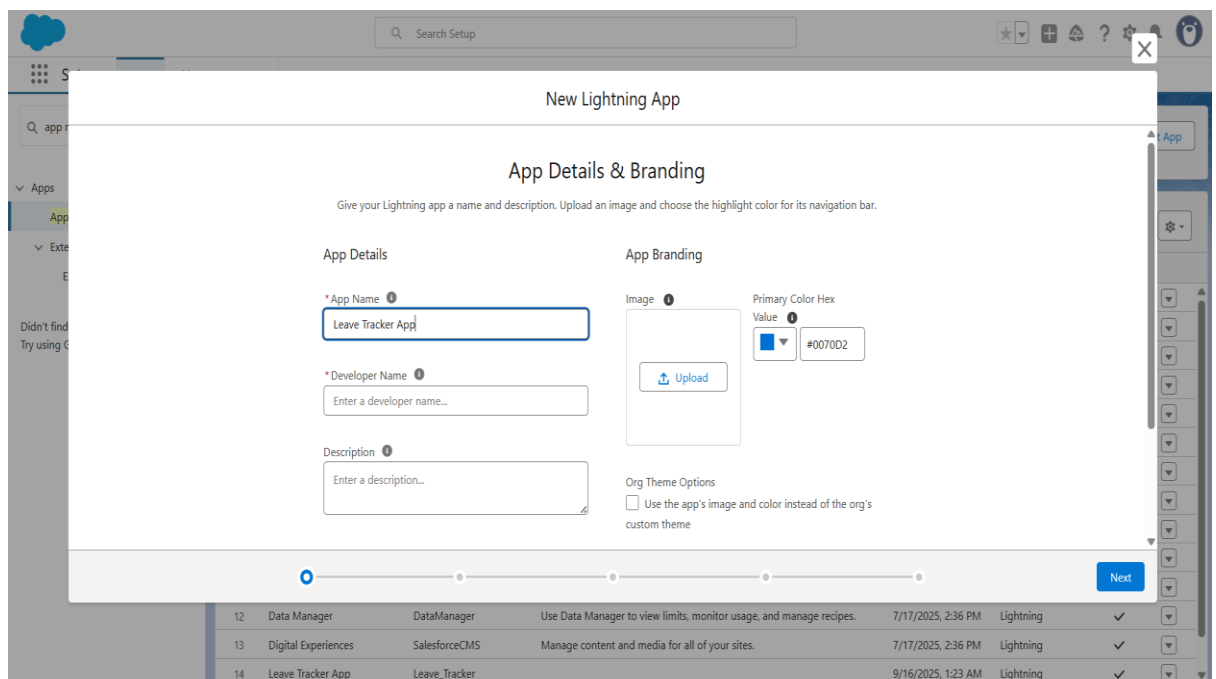
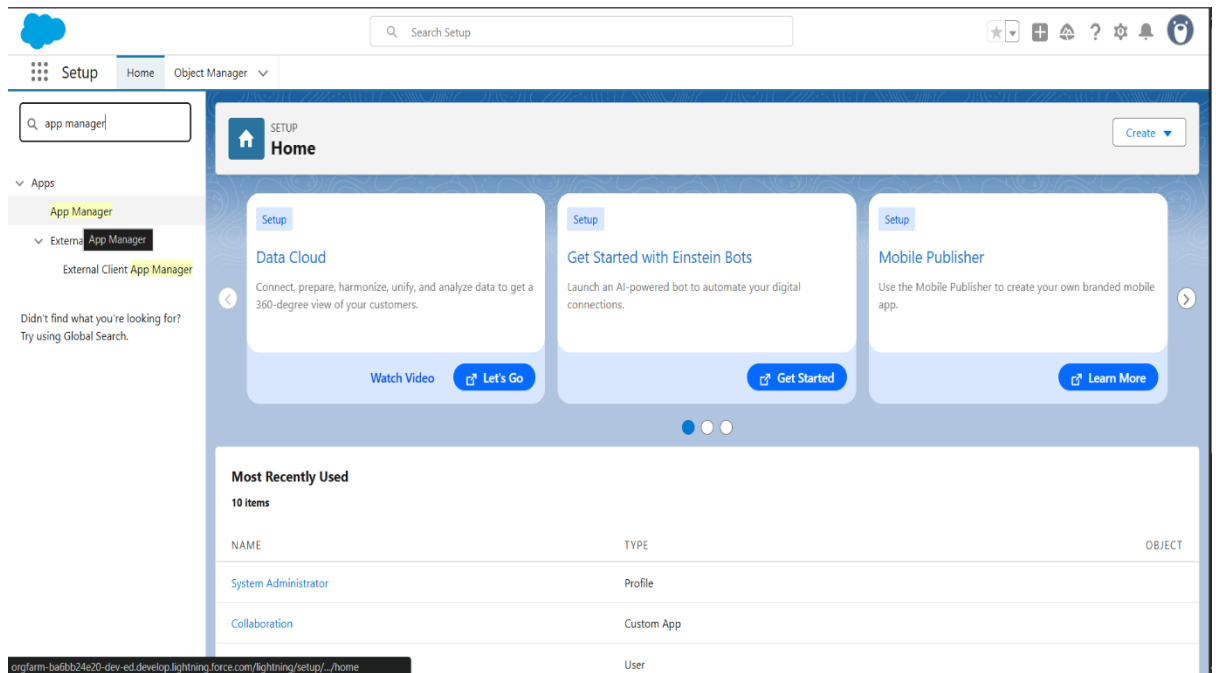
(note: All the 'Parent and 'Child' component's "-meta.xml" file's [isExposed> </isExposed>] should be 'true'.)

3. Lightning App Creation/Implimentation:

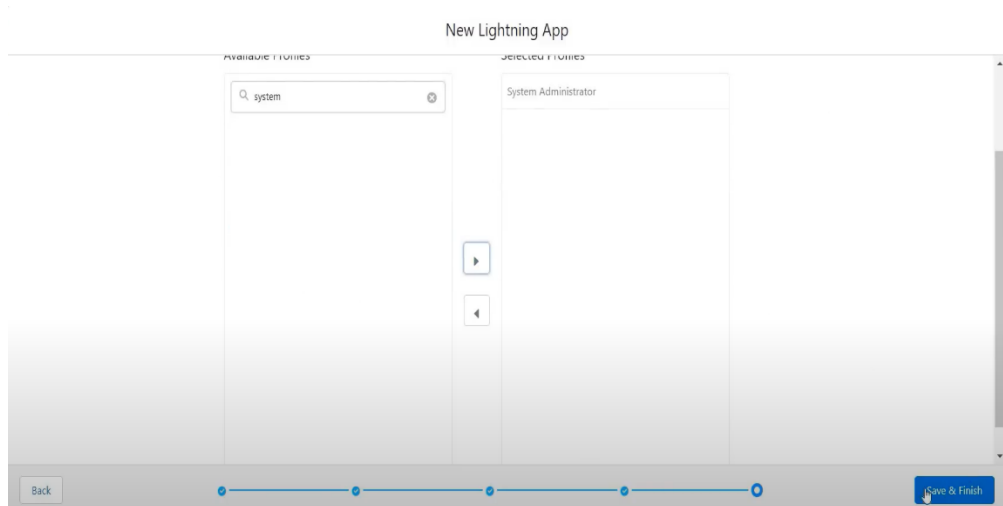
After completion of the codes and deploying it on the salesforce org we have to create a Lightning App and add the “custom component” developed using LWC and VScode in it.

Step 1:

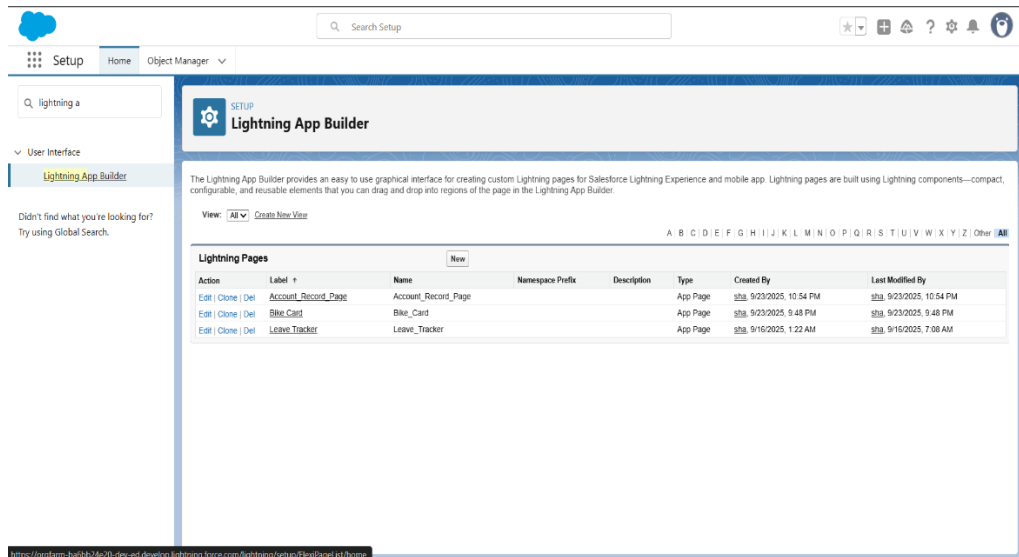
Goto setup→App manager→New Lightning App



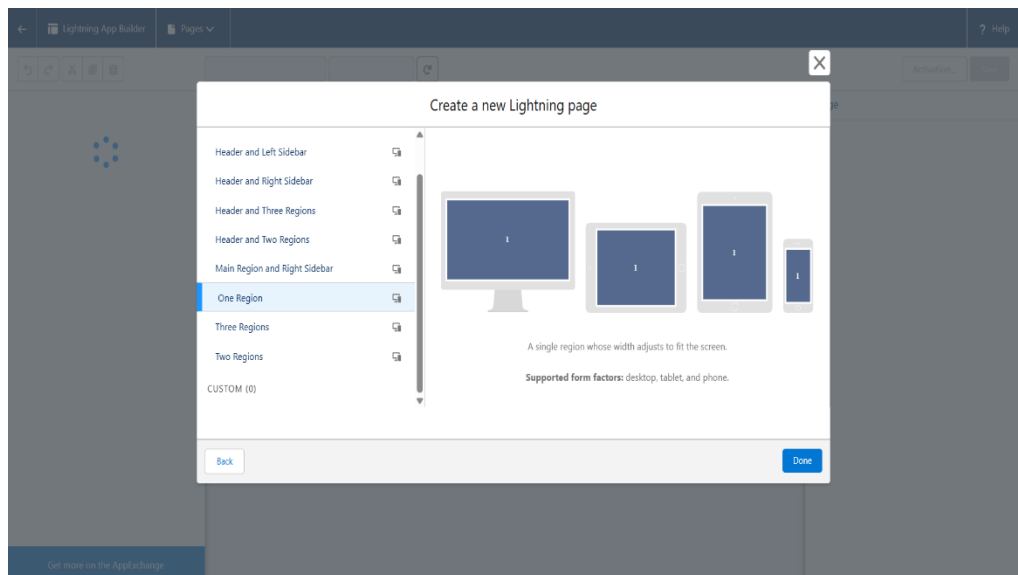
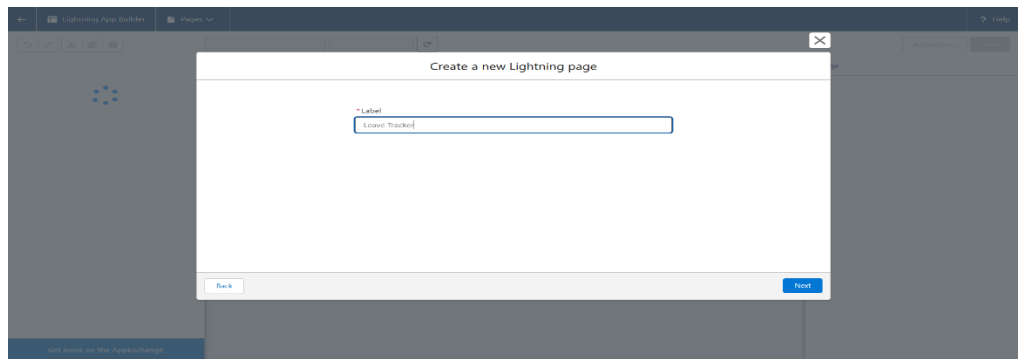
- After inserting the App name “Leave Tracker App” click on Next→Next→Next→Next→ now select “System Administrator” and click Save & Finish.



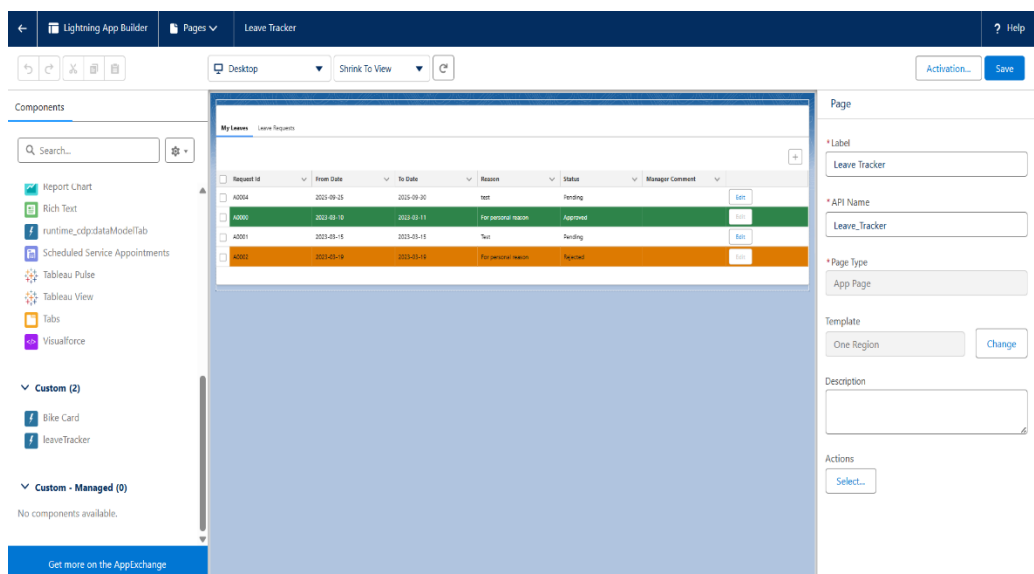
- Now goto home→ Lightning App Builder→ New and label it “leave Tracker”. (select the App page before it then hit Next then select One region).



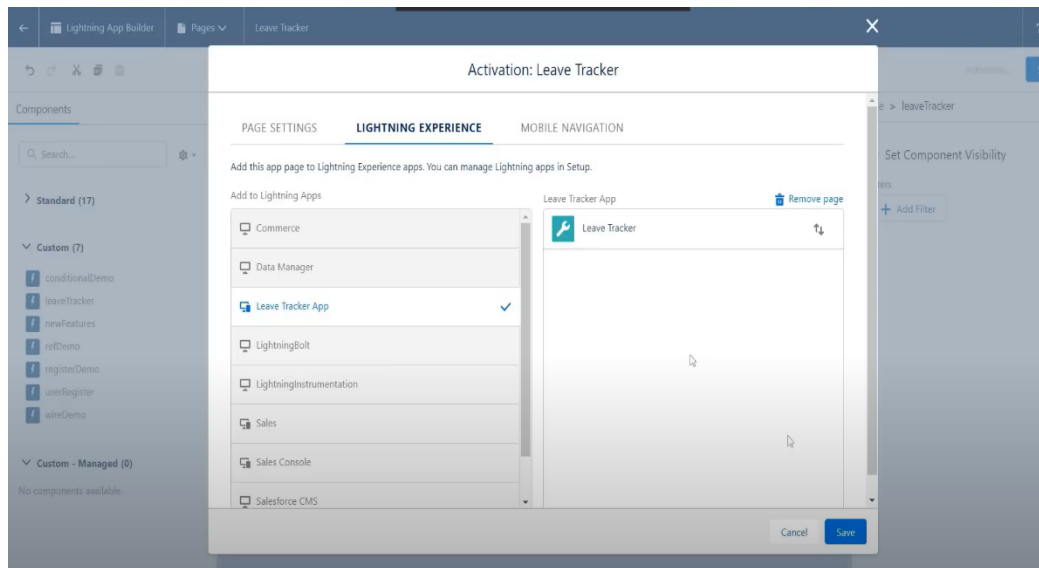
<https://orgtam-bulbb/mc20-dev-ed.develop.lightning.force.com/lightning/setup/firstPageLst/home>



- Now drag and drop the Custom 'leave Tracker' and then save and activate.



- Now save it and activate it (also in the Lightning Experience add this to Leave Tracker App we created Earlier).



- Also I implemented the some other things like:
 - a. Wire decorator implementation
 - b. Edit button click handler
 - c. Add button implementation
 - d. Grid auto-refresh using refreshApex (imperative Apex call)
 - e. Custom event creation and dispatch
 - f. Component communication

These all are implemented using VS code and then deployed using [SFDX: Deploy this source to org].

(note: I have pushed all the codes in the GitHub Repository, so you will be able to see these implementations in the code).

Phase 7: Integration & External Access (not applicable for the project/only Conceptually)

Purpose:

This phase aimed to connect Salesforce with external systems, such as a policyholder portal and third-party payment services, ensuring seamless data exchange and notification delivery.

1. Request and Reply Integration:

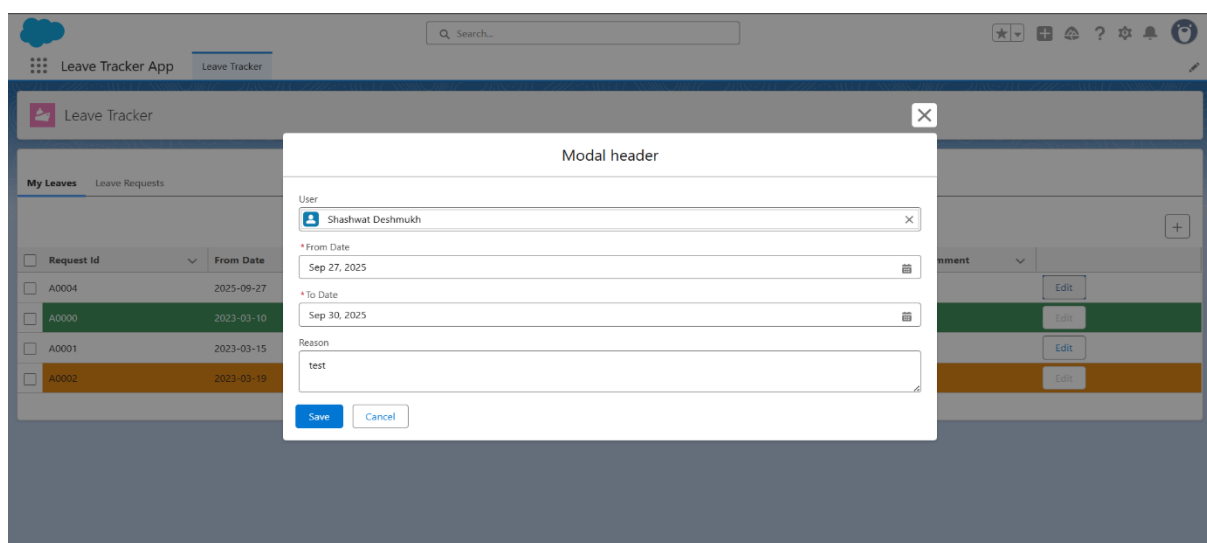
The Leave Tracker App implements Request & Reply pattern for real-time interactions with external systems. When employees submit leave requests, the system can instantly verify leave balances from external payroll systems and return approval status.

2. Batch Data Synchronization:

For large-scale data operations, the app utilizes Batch Data Synchronization pattern to sync employee data, leave policies, and historical leave records. This ensures data consistency across multiple systems without impacting real-time user experience.

3. Data Virtualization with Salesforce Connect

External employee data from HRMS systems can be accessed in real-time using Salesforce Connect without storing duplicate records. This zero-copy approach maintains data integrity while providing unified access to employee information.





<input type="checkbox"/>	A0020	Orgfarm EPIC	2025-09-27	2025-09-29	uo	Pending	Edit
<input type="checkbox"/>	A0019	Orgfarm EPIC	2025-09-25	2025-09-25	t	Pending	Edit
<input type="checkbox"/>	A0018	Orgfarm EPIC	2025-09-18	2025-09-21	..	Pending	Edit
<input type="checkbox"/>	A0017	Orgfarm EPIC	2025-09-18	2025-09-19	you	Pending	Edit
<input type="checkbox"/>	A0016	Orgfarm EPIC	2025-09-18	2025-09-22	dexter	Pending	Edit
<input type="checkbox"/>	A0015	Orgfarm EPIC	2025-09-17	2025-09-30	rest	Pending	Edit
<input type="checkbox"/>	A0014	Orgfarm EPIC	2025-09-20	2025-09-21	domain	Approved	approved!!! Edit
<input type="checkbox"/>	A0013	Orgfarm EPIC	2025-09-17	2025-09-27	Pending	Edit
<input type="checkbox"/>	A0012	Orgfarm EPIC	2025-09-18	2025-09-26	emergency		Edit
<input type="checkbox"/>	A0011	Orgfarm EPIC	2025-09-17	2025-09-26	..	Pending	Edit
<input type="checkbox"/>	A0010	Orgfarm EPIC	2025-09-17	2025-09-27	leave	Rejected	can't Edit
<input type="checkbox"/>	A0009	Orgfarm EPIC	2025-09-17	2025-09-26	.	Pending	Edit
<input type="checkbox"/>	A0008	Orgfarm EPIC	2025-09-18	2025-09-25	test3	Approved	approved! Edit
<input type="checkbox"/>	A0007	Orgfarm EPIC	2025-09-18	2025-09-21	test2	Approved	yes ok Edit
<input type="checkbox"/>	A0006	Orgfarm EPIC	2025-09-18	2025-09-30	best	Pending	Edit
<input type="checkbox"/>	A0005	Orgfarm EPIC	2025-09-17	2025-09-24	test	Rejected	nope Edit

Phase 8: Data Management & Deployment

Purpose:

The purpose of this phase was to ensure accurate data migration, prevent duplication, and deploy the solution across different Salesforce environments securely.

1. VS Code & SFDX:

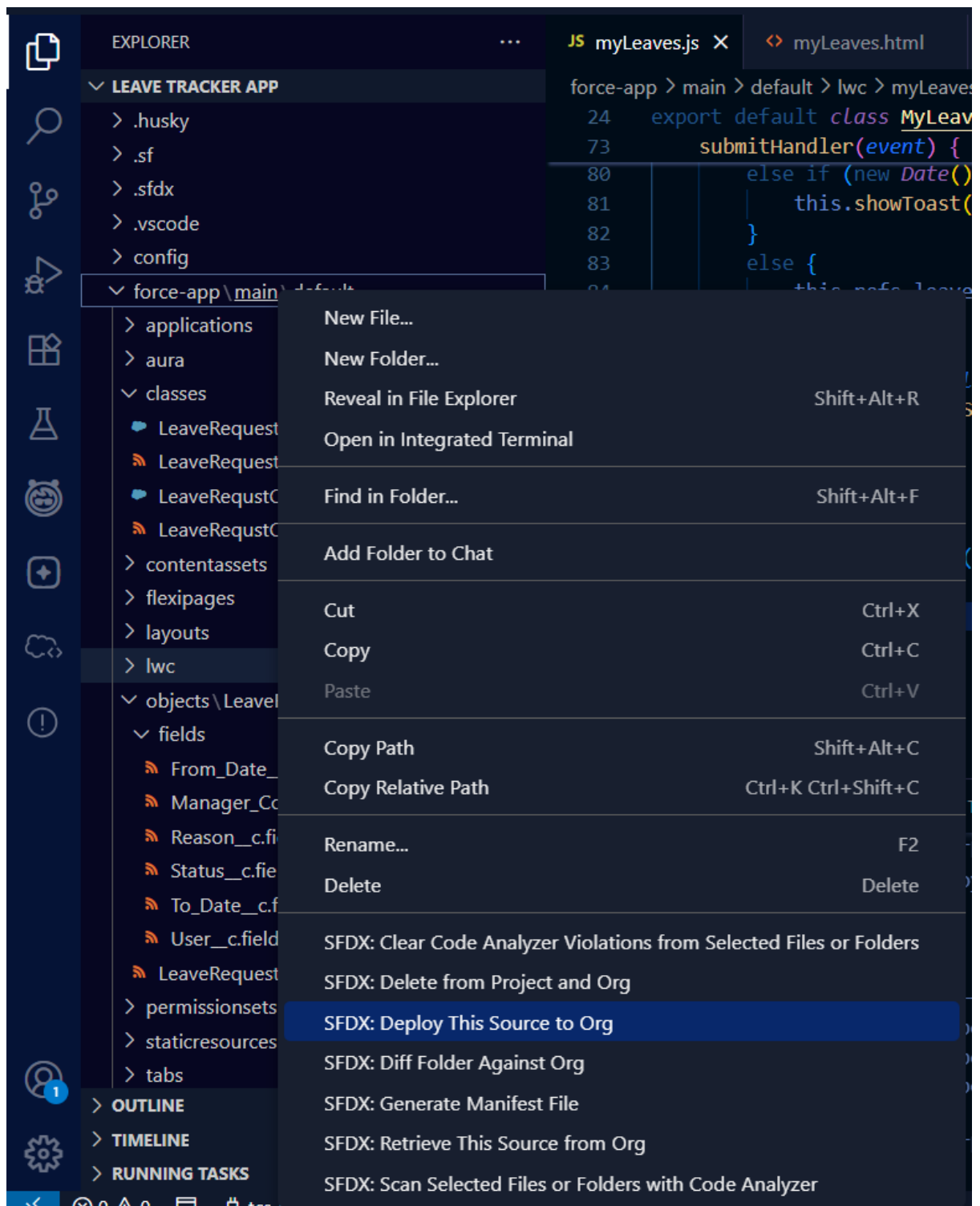
VS Code (Visual Studio Code) is a free, open-source code editor developed by Microsoft that serves as the primary development environment for Salesforce developers. It's lightweight, extensible, and available across Windows, Linux, and macOS platforms.

SFDX (Salesforce DX) is Salesforce's modern development platform that includes a commandline interface (CLI) and development tools designed for source-driven development. It represents the official future of Salesforce development, offering enhanced capabilities for working with metadata, scratch orgs, and version control systems.

2. How they work together:

VS Code and SFDX integrate seamlessly through the Salesforce Extension Pack, which brings SFDX functionality directly into the VS Code interface. This combination enables developers to create projects, authorize orgs, retrieve source code, deploy changes, and manage Salesforce metadata all within a unified development environment.

After Completing the code and debugging all the errors it is then deployed to the salesforce org using [SFDX Deploy this source to org]



Phase 9: Reporting, Dashboards & Security Review

Purpose:

The goal of this phase was to provide managers and agents with actionable insights through Salesforce reports and dashboards, enabling better decision-making and performance tracking.

1. Reports & Dashboards:

Leave Summary Reports: Create tabular reports showing employee leave requests by status (Pending, Approved, Rejected) and summary reports grouped by department and manager. Include matrix reports for comparing leave usage across different time periods.

Leave Analytics Dashboard: Build dashboards displaying key metrics like total leave requests, pending approvals, and manager workload distribution. Add dynamic dashboard components that filter data based on user roles.

Custom Report Types: Create report types combining Leave Request records with User data to enable detailed leave pattern analysis.

2. Security Implementations:

Sharing Settings: Configure Organization-Wide Defaults for Leave Request object to Private, with sharing rules allowing managers to see their team's requests and HR to access all records.

Field Level Security: Restrict access to sensitive fields like Manager Comments to appropriate user profiles only. Ensure employees can only edit their own pending requests.

Audit Trail: Enable field history tracking on critical fields like Status and Approval Date for compliance monitoring.

Phase 10: Final Presentation & Demo Day

Purpose:

The final phase ensured smooth knowledge transfer to stakeholders through demos, documentation, and training. It prepared the project for production use.

1. Project Presentation:

Business Impact Overview: Present how the Leave Tracker App transformed manual leave processes into an automated Lightning Web Component solution. Highlight key improvements like real-time updates and streamlined approval workflows.

Technical Showcase: Demonstrate the LWC architecture including wire decorators, component communication between "My Leaves" and "Leave Requests" tabs, and Lightning Data Services integration.

2. Live Demo:

End-to-End Walkthrough: Show complete leave request process from employee submission through manager approval, highlighting modal popups, form validations, and automatic grid refresh without page reload.

Component Communication Demo: Demonstrate how creating requests in one tab automatically updates the other tab through custom events and parent-child component interaction.

3. Documentation & Feedback:

Handoff Materials: Provide technical documentation covering component structure, deployment using VS Code/SFDX, and administrator configuration guides.

User Training: Create step-by-step guides for both employees and managers covering the new Lightning Web Component interface.

Code Repository: Upload clean, documented code to GitHub demonstrating LWC best practices, wire adapters, and component communication patterns.

Demo Video: Create screen recording showcasing the Leave Tracker App functionality with technical explanations suitable for portfolio presentation.

This simplified approach focuses on the core deliverables that directly relate to the Leave Management System demonstrated in the video, emphasizing practical implementation over extensive theoretical details.