

# Leave Tracker Management App - Salesforce CRM Project

## Project Overview

The **Leave Tracker Management App** is a Salesforce CRM-based solution developed to track employee leave requests, balances, and approvals. It replaces manual processes with a centralized, automated system built on **Apex, Lightning Web Components (LWC)**, and Salesforce platform features. This focus on developer-centric tools ensures a highly responsive UI, streamlined workflows, and a scalable solution.

## Objectives

The primary goals of developing the Leave Tracker Management App were to:

- **Maintain a centralized leave management system:** Consolidating all employee leave data, policies, and history within the secure Salesforce environment.
- **Provide a rich, responsive User Interface (UI):** Utilize **LWC** to deliver a fast, self-service application for employees and an efficient approval console for managers.
- **Automate Data Integrity:** Implement custom validation (client-side) and server-side logic (Apex Trigger/Validation Rules) to ensure accurate date ranges and prevent conflicting entries.
- **Streamline Communication:** Implement component-to-component communication logic to ensure immediate UI updates across the application upon record save/update.
- **Support Hierarchical Management:** Enable data filtering to restrict manager views to only their direct subordinates' leave requests.

# Phase 1: Problem Understanding & Industry Analysis

## Requirement Gathering

**Use Case:** To define the functional scope of the application based on user needs, focusing on automation and UI responsiveness.

### Explanation:

Key requirements identified were:

1. A single-page application experience for leave tracking (achieved via LWC tabs)
2. Immediate, dynamic feedback on data save/update (refreshApex)
3. Robust date validation (preventing past dates or invalid ranges) before data hits the server
4. Conditional styling to clearly indicate request status.

### Functional Requirements (What it must do)

ID	Requirement Description	Stakeholders	Status	Notes / Implementation
FR-1.0	<b>Leave Request Submission:</b> Must allow employees to submit new leave requests, specifying <b>From Date</b> , <b>To Date</b> , and <b>Reason</b> .	Employee	Completed	Implemented via 'lightning-record-edit-form' in 'c:myLeaves' LWC.
FR-1.1	<b>Manager Approval/Rejection:</b> Must provide managers with a console to view and update the status of subordinate requests.	Manager	Completed	Implemented in 'c:leaveRequests' LWC with read-only fields for details and editable fields for <b>Status</b> and <b>Manager Comment</b> .
FR-1.2	<b>Hierarchical Data Filtering:</b> Managers must <b>only</b> see leave requests submitted by their direct subordinates.	Manager, HR	Completed	Enforced via <code>SOQL</code> filtering in 'LeaveRequestController.getLeavesRequest()' using the standard <code>User.ManagerId</code> field.
FR-1.3	<b>Request Edit Control:</b> Must prevent editing of leave requests once the status is marked <b>Approved</b> or <b>Rejected</b> .	Employee	Completed	Enforced via conditional rendering logic ('isEditDisabled' property) in the 'c:myLeaves' LWC data table.
FR-1.4	<b>Initial Status Defaulting:</b> All new leave requests must automatically be set to 'Pending' upon submission.	System	Completed	Enforced via JavaScript logic in the 'c:myLeaves' 'onsubmit' handler.
FR-1.5**	<b>Leave Balance Management:</b> Must support a mechanism for calculating and displaying current employee leave balances.	HR, Employee	Planned	Requires <b>Batch Apex</b> and 'Leave_Balance__c' object for future implementation (Future Scope).

## User Experience (UX) Requirements (How it must feel)

ID	Requirement Description	Priority	Status	Notes / Implementation
UX-1.0	<b>Single Page Application (SPA) Flow:</b> Navigation between tabs must occur without a page refresh.	High	Completed	Achieved using the 'lightning-tabset' component in the parent LWC.
UX-1.1	<b>Real-time Grid Update:</b> The data grid must refresh automatically when a new record is saved.	High	Completed	Implemented using <code>refreshApex</code> and the Custom Event/Public Method sibling communication pattern.
UX-1.2	<b>Immediate Validation Feedback:</b> Users must receive instant feedback for invalid date submissions (e.g., past dates, invalid ranges).	High	Completed	Implemented via the LWC <code>onsubmit</code> override and <code>ShowToastEvent</code> (client-side validation).
UX-1.3	<b>Visual Status Indication:</b> Request status must be visually clear using color coding on the data table rows.	High	Completed	Implemented using <b>SLDS conditional styling</b> ( <code>slds-theme__success</code> for Approved, <code>slds-theme__warning</code> for Rejected).

## Technical Requirements (How it will be built)

ID	Requirement Description	Component/Technology	Status	Implementation Details
TR-1.0	<b>UI Technology:</b> All user-facing components must be built using <b>Lightning Web Components (LWC)</b> .	LWC	Completed	All front-end components are LWCs.
TR-1.1	<b>Data Access Layer:</b> Server-side data must be retrieved securely using <b>Apex Controllers</b> with <code>@AuraEnabled(cacheable=true)</code> .	Apex	Completed	'LeaveRequestController' handles all <code>SOQL</code> queries.
TR-1.2	<b>Component Communication:</b> Must support non-hierarchical communication between sibling components.	Custom Events, <code>@api</code>	Completed	Uses <b>Custom Events</b> (Child $\rightarrow$ Parent) and <b>Public Methods</b> (Parent $\rightarrow$ Sibling).
TR-1.3	<b>Source Management:</b> Project development and deployment must be managed using <b>VS Code and SFDX</b> .	SFDX	Completed	Used for creating project structure, managing metadata, and deployment.
TR-1.4	<b>Code Quality:</b> All Apex code must meet the <code>75%</code> code coverage requirement.	Apex Test Class	Planned	Requires creation of comprehensive unit tests for the controller logic.

## Stakeholder Analysis

**Use Case:** To identify the primary users and define their access and process requirements within the system.

### Explanation:

- **Employees:** Need self-service application (myLeaves tab), real-time status tracking, and conditional access (only editable when status is Pending).
- **Managers:** Need approval functionality (leaveRequests tab) and restricted visibility to only their subordinate's data (enforced by Apex SOQL filtering).
- **HR/Admins:** Need configuration and full reporting access.

### Key Stakeholders & Responsibilities

Stakeholder Role	Primary Responsibilities	Key Application Interactions	Access Level / Data Scope
Employee	<ul style="list-style-type: none"><li>• Apply for personal leave.</li><li>• Track the status of submitted leave requests.</li><li>• View personal leave history.</li><li>• Edit pending leave requests.</li></ul>	<ul style="list-style-type: none"><li>• Submit new leave via form.</li><li>• View records in "My Leaves" tab.</li><li>• Edit (pending only) via modal.</li></ul>	<ul style="list-style-type: none"><li>• <b>Create:</b> Own Leave Requests.</li><li>• <b>Read:</b> Own Leave Requests.</li><li>• <b>Edit:</b> Own Pending Leave Requests.</li><li>• <b>Delete:</b> Own Pending Leave Requests (Future).</li></ul>
Manager	<ul style="list-style-type: none"><li>• Review and approve/reject subordinate leave requests.</li><li>• Provide comments/feedback on requests.</li><li>• Monitor team's leave patterns.</li></ul>	<ul style="list-style-type: none"><li>• View records in "Leave Requests" tab.</li><li>• Update status/comments via modal.</li><li>• Receive notifications for new requests (Future).</li></ul>	<ul style="list-style-type: none"><li>• <b>Read:</b> Subordinate's Leave Requests (filtered by <code>\$!text{ManagerId}\$</code>).</li><li>• <b>Edit:</b> Subordinate's Pending Leave Requests (Status, Manager Comment).</li></ul>

<b>HR Administrator</b>	<ul style="list-style-type: none"> <li>• Oversee overall leave policy and management.</li> <li>• Generate reports on leave utilization.</li> <li>• Manage employee leave balances (Future).</li> <li>• Override leave requests/statuses (Future).</li> </ul>	<ul style="list-style-type: none"> <li>• Access to all tabs.</li> <li>• Reporting/Dashboard features.</li> <li>• Backend configuration.</li> </ul>	<ul style="list-style-type: none"> <li>• <b>Create/Read/Edit/Delete:</b> All Leave Requests.</li> <li>• Full access to associated objects.</li> <li>• Reporting and Dashboard access.</li> </ul>
<b>System Administrator</b>	<ul style="list-style-type: none"> <li>• Configure and maintain the Salesforce Org.</li> <li>• Deploy new features and updates.</li> <li>• Manage user permissions and security.</li> <li>• Troubleshoot technical issues.</li> </ul>	<ul style="list-style-type: none"> <li>• VS Code / SFDX.</li> <li>• Salesforce Setup.</li> <li>• Developer Console.</li> </ul>	<ul style="list-style-type: none"> <li>• Full system access (profiles, FLS, metadata).</li> <li>• Developer tooling access.</li> </ul>

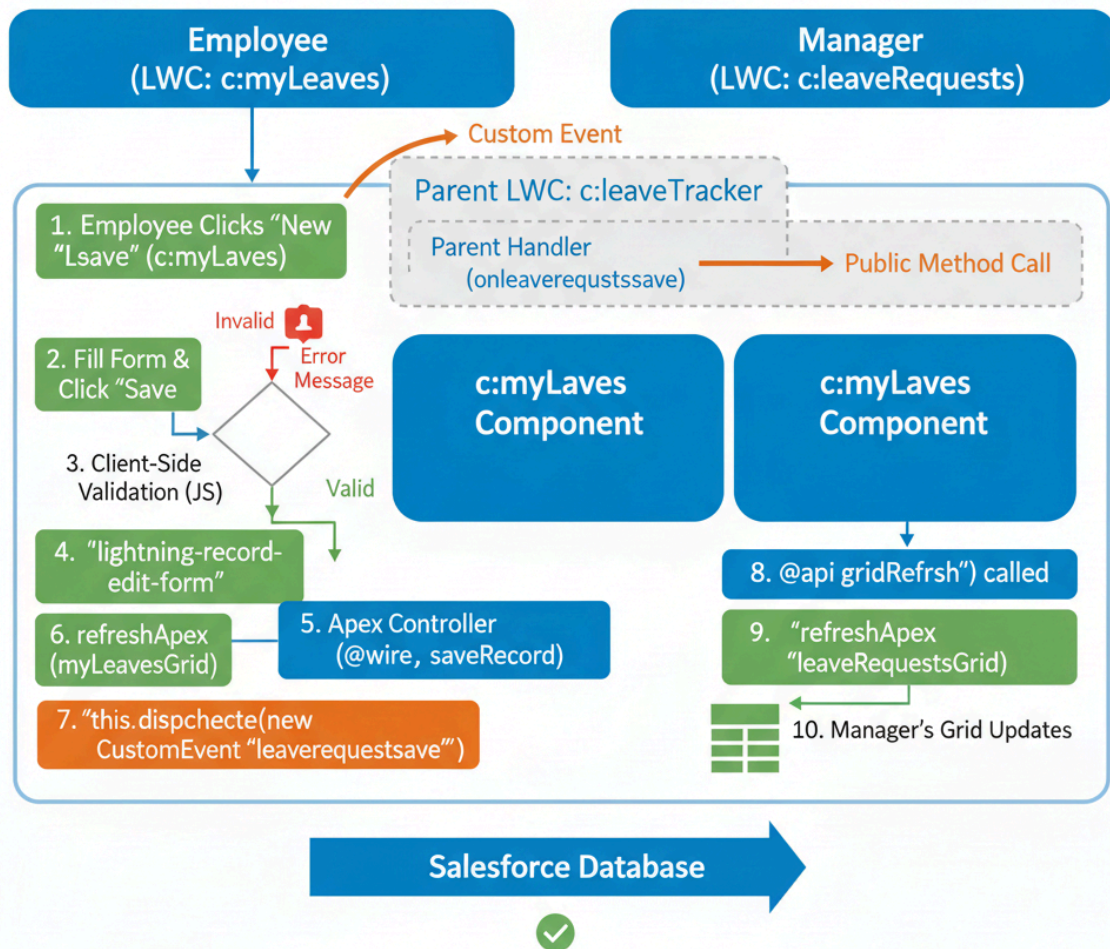
## Business Process Mapping

### Use Case:

To visualize and validate the streamlined, automated leave management process.

**Explanation:** The new process is highly automated: Employee submits via **lightning-record-edit-form** → **LWC onsubmit** validates dates → **Apex** executes → **refreshApex** updates the employee's grid → **Custom Event** notifies the Parent → **Parent** calls the Sibling's Public Method → **Manager's Grid** updates dynamically.

# Automated Leave Process Flow

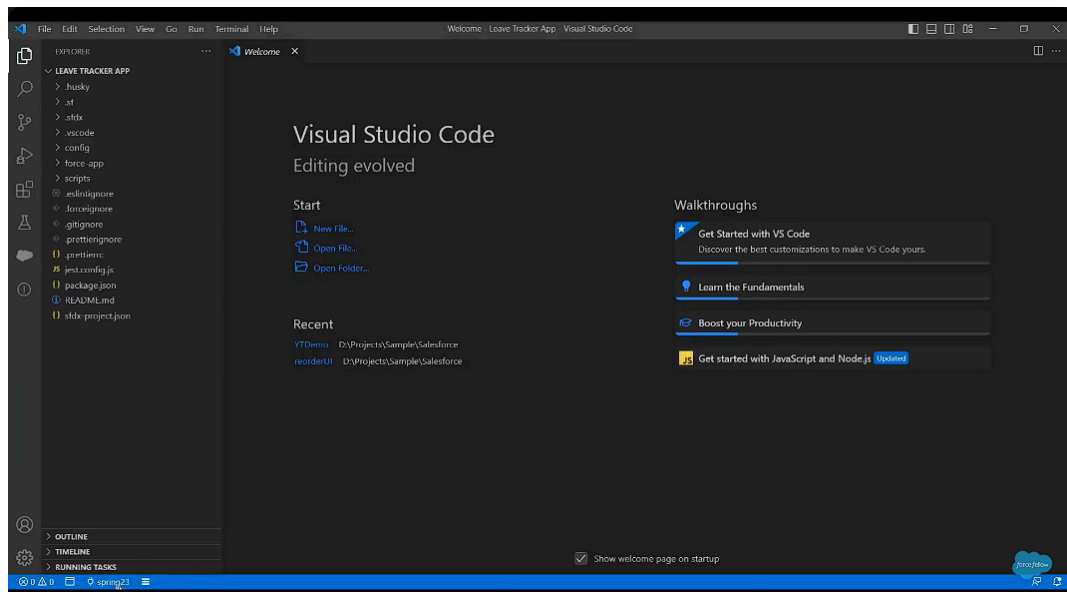


# Phase 2: Org Setup & Configuration

## Salesforce Editions & Dev Org Setup

**Use Case:** Establishing the development environment for building and deploying LWC and Apex code

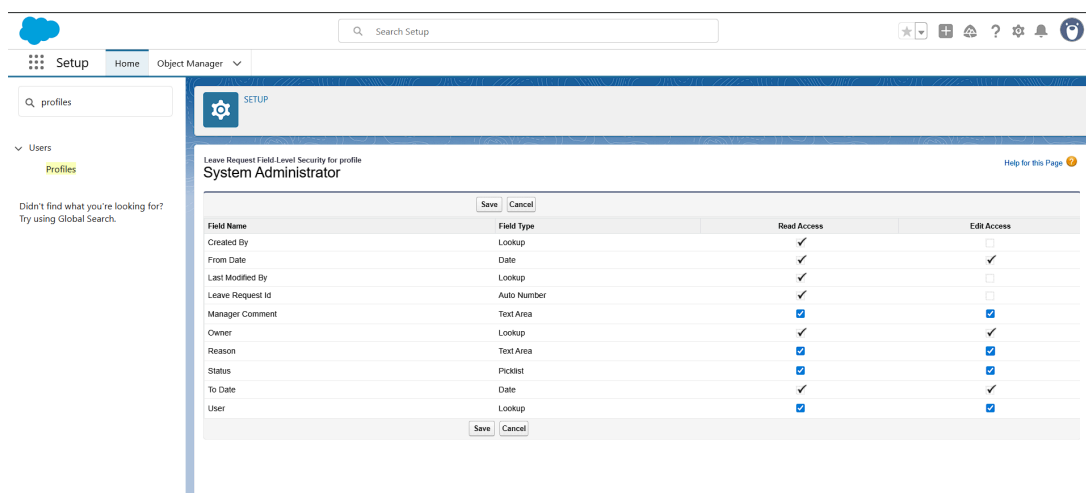
**Explanation:** A Developer Edition Org was utilized. The project was created and managed using SFDX in VS Code, ensuring a secure, version-controlled environment and enabling efficient deployment of all custom metadata.



## User Profiles & Access Control

**Use Case:** Assigning necessary permissions for the development user to perform CRUD operations on the custom object.

**Explanation:** The System Administrator Profile was configured to have full Object Permissions and Field-Level Security (FLS) on the Leave\_Request\_\_c custom object. While demoing the Manager functionality, it was assumed the users had the appropriate standard Manager lookup field populated to facilitate the SOQL filtering (Phase 5).





## Lightning App and App Page Creation

**Use Case:** Creating a dedicated container for the application components and user navigation.

**Explanation:** A custom Lightning App called "Leave Tracker Management App" was created. A Lightning App Page called "Leave Tracker" was created in the App Builder, and the parent LWC (c:leaveTracker) was placed on this page, defining the initial UI.

The screenshot displays the Lightning App Builder interface for the 'Leave Tracker' app. The top navigation bar includes 'Lightning App Builder', 'Pages', and 'Leave Tracker'. The main workspace shows a preview of the 'My Leaves' page, which contains a table of leave requests. The table has columns for Request Id, From Date, To Date, Reason, Status, and Manager Comment. The table lists four requests: A0009 (medical reason, Pending), A0000 (For personal reason, Approved), A0001 (Test, Pending), and A0002 (For personal reason, Rejected). The left sidebar shows the 'Components' panel with a search bar and a list of components including Rich Text, runtime\_cdpdataModelTab, Scheduled Service Appointments, Tableau Pulse, Tableau View, Tabs, and Visualforce. The right sidebar shows the 'Page' configuration panel with fields for Label (Leave Tracker), API Name (Leave\_Tracker), Page Type (App Page), Template (One Region), and Description. There is also an 'Actions' section with a 'Select...' button.

Request Id	From Date	To Date	Reason	Status	Manager Comment
A0009	2025-09-20	2025-10-06	medical reason	Pending	
A0000	2023-03-10	2023-03-11	For personal reason	Approved	
A0001	2023-03-15	2023-03-15	Test	Pending	
A0002	2023-03-19	2023-03-19	For personal reason	Rejected	

## Phase 3: Data Modeling & Relationships

### Custom Object: Leave\_Request\_\_c

**Use Case:** Defining the core data structure to store all leave application details.

**Explanation:** The Leave\_Request\_\_c object was created via metadata deployment. This object stores essential data points for the application.

The screenshot displays the Salesforce Setup interface for the 'Leave Request' object. The top navigation bar includes 'Setup', 'Home', and 'Object Manager'. The main workspace shows the 'Leave Request' object configuration page. The left sidebar shows the 'Details' section with a list of configuration options: Fields & Relationships, Page Layouts, Lightning Record Pages, Buttons, Links, and Actions, Compact Layouts, Field Sets, Object Limits, Record Types, Related Lookup Filters, Search Layouts, List View Button Layout, Restriction Rules, and Scoping Rules. The right sidebar shows the 'Details' section with fields for API Name (LeaveRequest\_\_c), Custom, Singular Label (Leave Request), Plural Label (Leave Requests), Enable Reports, Track Activities, Track Field History, Deployment Status, Help Settings, and a link to the Standard salesforce.com Help Window.



## Custom Fields & Lookup

**Use Case:** Capturing specific application data and linking the request to the submitting employee and the manager via the User object.

**Explanation:** Key fields include From\_Date\_\_c, To\_Date\_\_c, Status\_\_c, and Manager\_Comment\_\_c. A Lookup Relationship was defined from Leave\_Request\_\_c to the standard User object (User\_\_c) to identify the employee who applied for the leave.

**Setup** | Home | Object Manager ▾

---

SETUP > OBJECT MANAGER  
**Leave Request**

Details

**Fields & Relationships**  
 10 Items. Sorted by Field Label

Quick Find

FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
Created By	CreatedById	Lookup(User)		
From Date	FromDate__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	ToDate__c	Date		
User	User__c	Lookup(User)		✓

## Data Transformation for UI

**Use Case:** Creating derived fields in the JavaScript layer to handle styling and related object data display.

**Explanation:** The Apex SOQL queries used a Relationship Query (User\_\_r.Name) to get the manager name, but this data was mapped in the LWC's JavaScript to a simple property (e.g., username) to be compatible with the lightning-datatable column structure.

The image shows a screenshot of a code editor (likely VS Code) with a project explorer on the left, a code editor in the center, and a terminal on the right.

**Project Explorer (Left):**

- force-app/main/default
  - components
    - leaveRequests
  - classes
  - components
  - layouts
  - lwc
    - leaveRequests
  - myLeaves
    - \_\_init\_\_
    - leaveRequests.html
    - leaveRequests.js
    - leaveRequests.js-meta.xml
  - myLeavesTracker
    - \_\_init\_\_
    - leaveTracker.html
    - leaveTracker.js
    - leaveTracker.js-meta.xml
  - myLeaves
    - \_\_init\_\_
    - myLeaves.html
    - myLeaves.js
    - myLeaves.js-meta.xml
  - jsconfig.json
  - objects/leaveRequest\_c
    - fields
    - leaveRequest\_\_object-meta.xml
  - permissionsets
  - staticresources
  - tabs
  - triggers
  - scripts
  - apex
  - tool

**Code Editor (Center):**

```
1 // LeaveTrackerApp
2
3 import { LightningElement, track } from 'lwc';
4
5 export default class LeaveRequests extends LightningElement {
6   @wire({})
7   myLeaves;
8
9   if (result.data) {
10     this.leaveRequests = result.data.map(a => ({
11       ...a,
12       username: a.User__r.Name,
13       calendar: a.Status_c === 'Approved' ? 'slds-theme_success' : a.Status_c === 'Rejected' ? 'slds-theme_warning' : '',
14       isDisabled: a.Status_c !== 'Pending'
15     }));
16   }
17
18   if (result.error) {
19     console.log('Error occurred while fetching my leaves - ', result.error);
20   }
21 }
22
23 get noRecordsFound() {
24   return this.leaveRequests.length === 0;
25 }
26
27 handleRequestClickHandler() {
28   this.showModalPopup = true;
29   this.recordId = '';
30 }
31
32 popupCloseHandler() {
33   this.showModalPopup = false;
34 }
35
36 rowActionHandler(event) {
37   this.showModalPopup = true;
38   this.recordId = event.detail.rowId;
39 }
40
41 successHandler(event) {
42   this.showModalPopup = false;
43   this.showToast('Data saved successfully');
44   this.refreshGrid();
45 }
46
47 @api
48 refreshGrid() {
49   refresh Apex({this.leaveRequestsWireResult});
50 }
51
52 showToast(message, title = 'Success', variant = 'success') {
53   const event = new ShowToastEvent({
54     title,
55     message,
56     variant,
57     mode: 'toast'
58   });
59   this.dispatchEvent(event);
60 }
```

**Terminal (Right):**

```
npm install
```

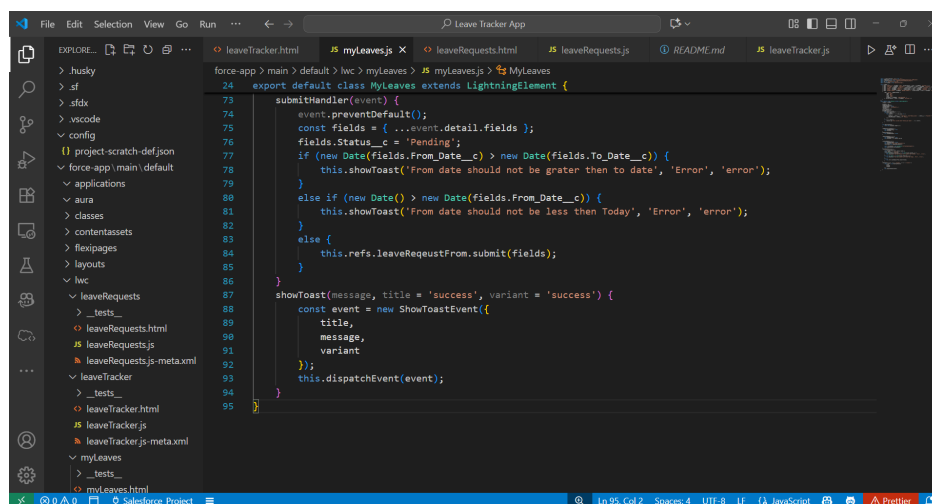
## Phase 4: Process Automation (Admin)

### Client-Side Validation on Submit

**Use Case:** Providing immediate user feedback and preventing invalid data from reaching the server by overriding the form submission.

**Explanation:** The onsubmit event handler in the myLeaves LWC was implemented using `event.preventDefault()` to stop the default form submission. Custom JavaScript validation logic was executed to check for:

1. **Past Date Check:** From Date should not be less than Today.
2. **Date Range Check:** From Date should not be greater than To Date.

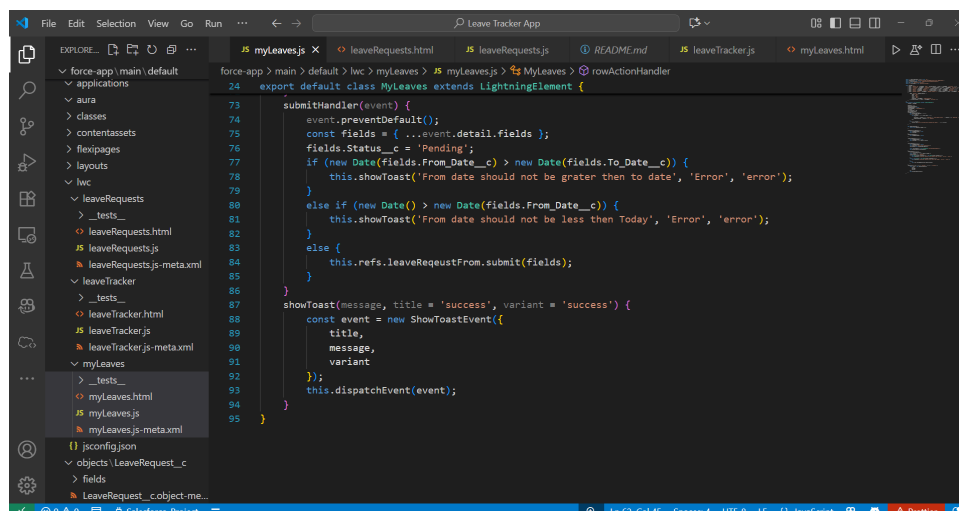


```
force-app > main > default > lwc > myLeaves > JS myLeaves.js > MyLeaves
24 export default class MyLeaves extends LightningElement {
73   submitHandler(event) {
74     event.preventDefault();
75     const fields = {...event.detail.fields};
76     fields.Status__c = 'Pending';
77     if (new Date(fields.From_Date__c) > new Date(fields.To_Date__c)) {
78       this.showToast('From date should not be greater then to date', 'Error', 'error');
79     }
80     else if (new Date() > new Date(fields.From_Date__c)) {
81       this.showToast('From date should not be less then Today', 'Error', 'error');
82     }
83     else {
84       this.refs.leaveRequestFrom.submit(fields);
85     }
86   }
87   showToast(message, title = 'success', variant = 'success') {
88     const event = new ShowToastEvent({
89       title,
90       message,
91       variant
92     });
93     this.dispatchEvent(event);
94   }
95 }
```

### Status Defaulting

**Use Case:** Automatically setting the initial lifecycle state for all newly created leave requests.

**Explanation:** Within the overridden onsubmit handler, before allowing the form to submit, the Status\_\_c field was explicitly set to 'Pending'. This ensures all new requests enter the correct lifecycle state automatically.



```
force-app > main > default > lwc > myLeaves > JS myLeaves.js > MyLeaves > rowActionHandler
24 export default class MyLeaves extends LightningElement {
73   submitHandler(event) {
74     event.preventDefault();
75     const fields = {...event.detail.fields};
76     fields.Status__c = 'Pending';
77     if (new Date(fields.From_Date__c) > new Date(fields.To_Date__c)) {
78       this.showToast('From date should not be greater then to date', 'Error', 'error');
79     }
80     else if (new Date() > new Date(fields.From_Date__c)) {
81       this.showToast('From date should not be less then Today', 'Error', 'error');
82     }
83     else {
84       this.refs.leaveRequestFrom.submit(fields);
85     }
86   }
87   showToast(message, title = 'success', variant = 'success') {
88     const event = new ShowToastEvent({
89       title,
90       message,
91       variant
92     });
93     this.dispatchEvent(event);
94   }
95 }
```

# Conditional Grid Styling and Controls

Use Case: Visually communicating the status and controlling user interaction on the data table based on the record's status.

Explanation:

- 1. **Styling:** SLDS theme classes (slds-theme\_success for Approved, slds-theme\_warning for Rejected) were mapped to the data array in JavaScript to provide conditional row backgrounds.
- 2. **Edit Control:** A boolean property (isEditDisabled) was calculated in JavaScript to be True if Status\_\_c is not 'Pending', dynamically disabling the edit button in the data table column configuration.

LEAVE TRACKER

Leave Tracker Mana...Leave Tracker

Q Search...

★

+

🔒

?

⚙

🔔

👤

Leave Tracker

My Leaves

Leave Requests

Request Id

From Date

To Date

Reason

Status

Manager Comment

A0009

2025-09-30

2025-10-06

medical reason

Pending

Edit

A0000

2023-03-10

2023-03-11

For personal reason

Approved

Edit

A0001

2023-03-15

2023-03-15

Test

Pending

Edit

A0002

2023-03-19

2023-03-19

For personal reason

Rejected

Edit

Data Table showing green/yellow rows and disabled Edit button

## Phase 5: Apex Programming (Developer)

### Apex Controller (LeaveRequestController)

**Use Case:** Providing secure and efficient server-side data fetching for both employee and manager views.

**Explanation:** The class contains key @AuraEnabled(cacheable=true) methods:

1. **getMyLeaves():** Filters Leave\_Request\_\_c records where User\_\_c=Current User Id.
2. **getLeavesRequest():** Filters Leave\_Request\_\_c records using a semi-join or relationship query to find users whose ManagerId equals the Current User Id, effectively implementing the Manager's hierarchical security.

```
force-app > main > default > classes > LeaveRequestController.cls > ...
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__c =:UserInfo.getUserId() ORDER BY CreatedDate DESC];
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> getLeavesRequests() {
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 FROM LeaveRequest__c
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 Apex Code for getLeavesRequest()

### Handling Related Field Access

**Use Case:** Transforming complex SOQL relationship results into simple properties for LWC display

**Explanation:** In methods like getLeavesRequest(), the query returns User\_\_r.Name. In the LWC JavaScript, this path is accessed, and the value is assigned to a flat property (e.g., username) to be bound directly to the lightning-datatable column, solving the LWC constraint on relationship field display.

```
force-app > main > default > lwc > leaveRequests > JS leaveRequests.js > LeaveRequests
25  export default class LeaveRequests extends LightningElement {
26
27      //
28      leavesRequests = [];
29      leavesRequestsWireResult;
30      showModalPopup = false;
31      objectApiName = 'LeaveRequest__c';
32      recordId = '';
33      currentUserId = Id;
34      @wire(getLeavesRequests)
35      wiredMyLeaves(result) {
36          this.leavesRequestsWireResult = result;
37          if (result.data) {
38              this.leavesRequests = result.data.map(a => ({
39                  ...a,
40                  username: a.User__r.Name,
41                  cellClass: a.Status__c == 'Approved' ? 'slds-theme_success' : a.Status__c == 'Rejected' ? 'slds-theme_warning' : '',
42                  isEditDisabled: a.Status__c != 'Pending'
43              }));
44          }
45          if (result.error) {
46              console.log('Error occurred while fetching my leaves - ', result.error);
47          }
48      }
29 }
```

Placeholder for LWC JS code snippet showing assignment of User\_\_r.Name to a flat property

## Phase 6: User Interface Development

### Parent LWC: `leaveTracker` (Communication Hub)

**Use Case:** Creating the component hierarchy and serving as the communication bridge between sibling components.

**Explanation:** This component houses the tab structure and is primarily used to listen for the Custom Event (`onleaverequestsave`) fired by the `c:myLeaves` component. Upon receiving the event, it calls the public method (`gridRefresh()`) on the `c:leaveRequests` component, ensuring both grids update seamlessly.

```
force-app > main > default > lwc > leaveTracker > leaveTracker.html > template
1  <template>
2    <lightning-card>
3      <lightning-tabset>
4        <lightning-tab title="My Leaves" label="My Leaves">
5          <c-my-leaves onrefreshleaverequests={refreshLeaveRequeestHandler}></c-my-leaves>
6        </lightning-tab>
7        <lightning-tab title="Leave Requests" label="Leave Requests">
8          <c-leave-requests lwc:ref="myLeavesComp"></c-leave-requests>
9        </lightning-tab>
10     </lightning-tabset>
11   </lightning-card>
12 </template>
```

`leaveTracker.html` Code showing `onleaverequestsave`

### Child LWC: `myLeaves` (Add/Edit Form)

**Use Case:** Providing the employee interface for submitting and editing leave.

**Explanation:** This component implements the modal using SLDS markup and utilizes `lightning-record-edit-form` with the `record-id` attribute, allowing it to handle both Add (blank ID) and Edit (existing ID) modes seamlessly. `refreshApex` is called in the `onsuccess` handler to update the local grid.

```
force-app > main > default > lwc > myLeaves > myLeaves.js > MyLeaves > wiredMyLeaves
24 export default class MyLeaves extends LightningElement {
64
65   successHandler(event) {
66     this.showModalPopup = false;
67     this.showToast('Data saved successfully');
68     refreshApex(this.myLeavesWireResult);
69
70     const refreshEvent = new CustomEvent('refreshleaverequests');
71     this.dispatchEvent(refreshEvent);
72   }
73   submitHandler(event) {
74     event.preventDefault();
75     const fields = { ...event.detail.fields };
76     fields.Status__c = 'Pending';
77     if (new Date(fields.From_Date__c) > new Date(fields.To_Date__c)) {
78       this.showToast('From date should not be grater then to date', 'Error', 'error');
79     }
80     else if (new Date() > new Date(fields.From_Date__c)) {
81       this.showToast('From date should not be less then Today', 'Error', 'error');
82     }
83     else {
84       this.refs.leaveRequestFrom.submit(fields);
85     }
86   }
87   showToast(message, title = 'success', variant = 'success') {
88     const event = new ShowToastEvent({
89       title,
90       message,
91       variant
92     });
93     this.dispatchEvent(event);
94   }
95 }
```

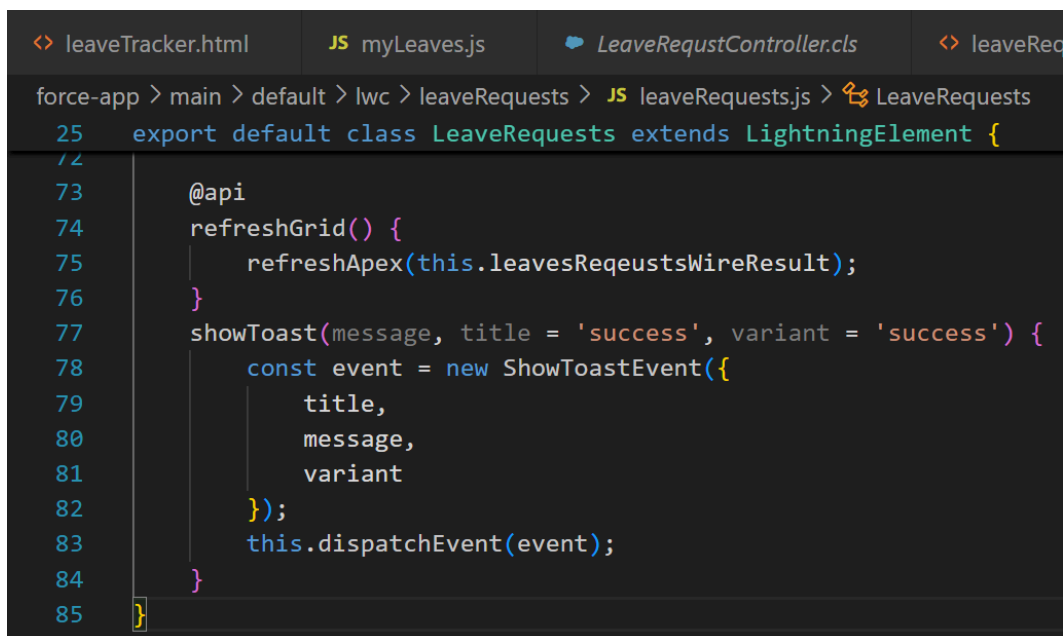
`myLeaves.js` Code showing `refreshApex` and `CustomEvent`

## Child LWC: `leaveRequests` (Manager View)

**Use Case:** Displaying subordinate requests and providing the manager with an interface to approve/reject.

**Explanation:** This component:

1. Calls `getLeavesRequest()` to filter data by `ManagerId`.
2. Implements the public method `@api gridRefresh()` which is called by the Parent component to invoke `refreshApex` and update the grid when a new request is created by a subordinate.
3. Uses `lightning-record-edit-form` for manager updates, but displays all non-editable fields using `lightning-output-field` for a read-only view.



```
<> leaveTracker.html JS myLeaves.js LeaveRequestController.cls <> leaveRec
force-app > main > default > lwc > leaveRequests > JS leaveRequests.js > LeaveRequests
25 export default class LeaveRequests extends LightningElement {
72
73   @api
74   refreshGrid() {
75     refreshApex(this.leavesRequestsWireResult);
76   }
77   showToast(message, title = 'success', variant = 'success') {
78     const event = new ShowToastEvent({
79       title,
80       message,
81       variant
82     });
83     this.dispatchEvent(event);
84   }
85 }
```

Insert Screenshot: `leaveRequests.js` Code showing `@api gridRefresh()`

# Phase 7: Integration & External Access

## Sibling Component Communication

**Use Case:** Implementing real-time, non-hierarchical data refresh between components on the same page.

**Explanation:** Communication was achieved by:

1. **Child (A):** Dispatches a **Custom Event** on save (leaverequestsave).
2. **Parent:** Listens to the event and uses the **@ref attribute** to get a reference to **Child (B)** (c:leaveRequests).
3. **Parent:** Calls the **@api gridRefresh()** public method defined on **Child (B)**. This method in turn executes **refreshApex** to update the manager's grid.

```
JS leaveTracker.js X
force-app > main > default > lwc > leaveTracker > JS leaveTracker.js > LeaveTracker
1 import { LightningElement } from 'lwc';
2
3 export default class LeaveTracker extends LightningElement {
4
5     refreshLeaveRequestHandler(event) {
6         this.refs.myLeavesComp.refreshGrid();
7     }
8 }
```

leaveTracker.js Handler calling the public method via @ref

## Email Alerts & Platform Events (Future Scope)

**Use Case:** Decoupling communication to external systems or providing formal notifications.

**Explanation:** While not fully implemented, the structured approach allows for future integration: Email Alerts would notify managers of new requests, and a Platform Event (Leave\_Status\_Change\_\_e) would be published on final approval to notify an external HRIS/Payroll system (decoupled integration).

The screenshot shows the Salesforce Setup interface. On the left, a navigation menu includes 'Setup', 'Home', 'Object Manager', and a search bar. The main content area is titled 'Platform Events' and shows the configuration for a 'Platform Event' named 'Leave Status Update'. The configuration details include:

- Platform Event Definition Detail:** Singular Label: Leave Status Update, Plural Label: Leave Status Updates, Object Name: Leave\_Status\_Update, API Name: Leave\_Status\_Update\_\_e, Event Type: High Volume, Publish Behavior: Publish After Commit, Created By: Sarthi Thakral, 9/28/2025, 6:58 AM, Modified By: Sarthi Thakral, 9/28/2025, 6:58 AM.
- Standard Fields:** A table with columns: Action, Field Label, Field Name, Data Type, Controlling Field, Indexed. It lists fields like Created By, Created Date, Event UUID, and Related ID.
- Custom Fields & Relationships:** A table with columns: Action, Field Label, API Name, Data Type, Indexed, Controlling Field, Modified By. It lists custom fields like Employee ID, Leave Request ID, and New Status.

Placeholder for Platform Event Definition

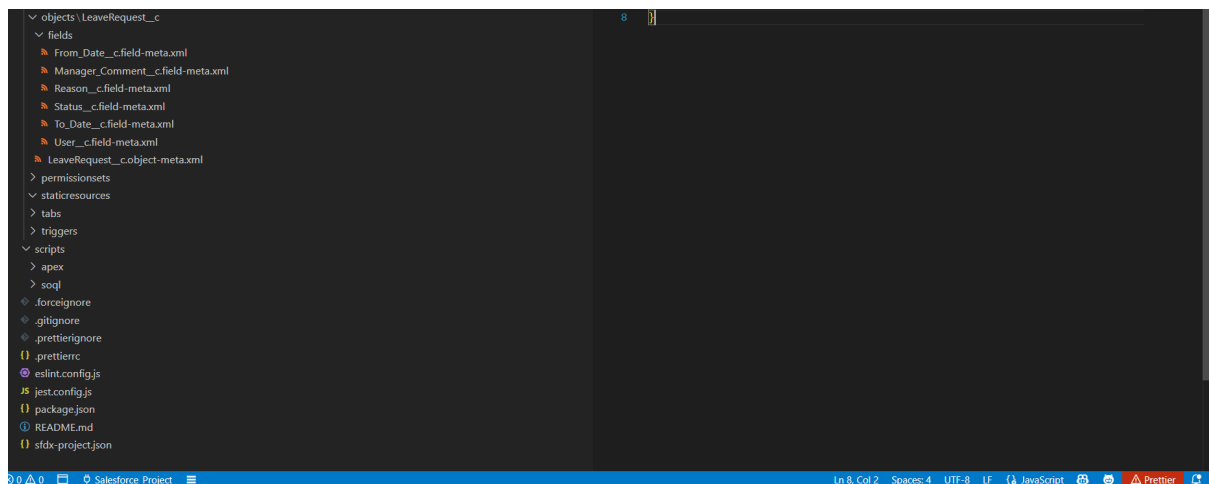
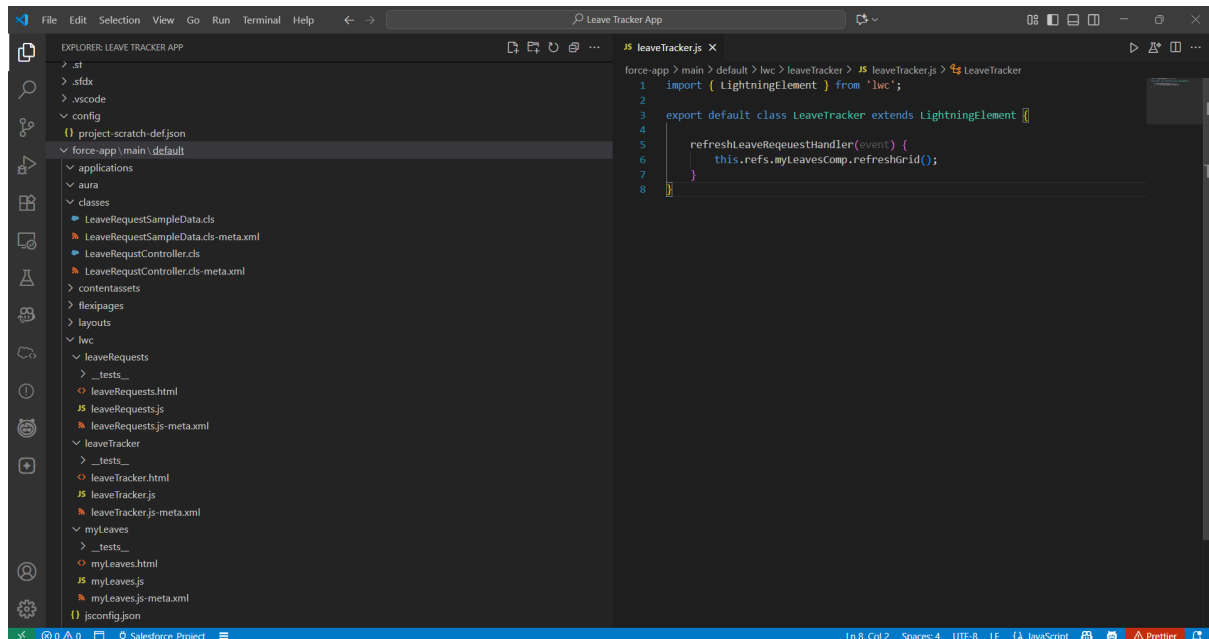


# Phase 8: Data Management & Deployment

## VS Code & SFDX

**Use Case:** Managing source code, metadata, and deployment lifecycle.

**Explanation:** VS Code and SFDX were essential. All LWC, Apex Classes, and the custom object schema were deployed using SFDX: Deploy Source to Org. This ensures all code deployed is version-controlled and adheres to modern development standards.

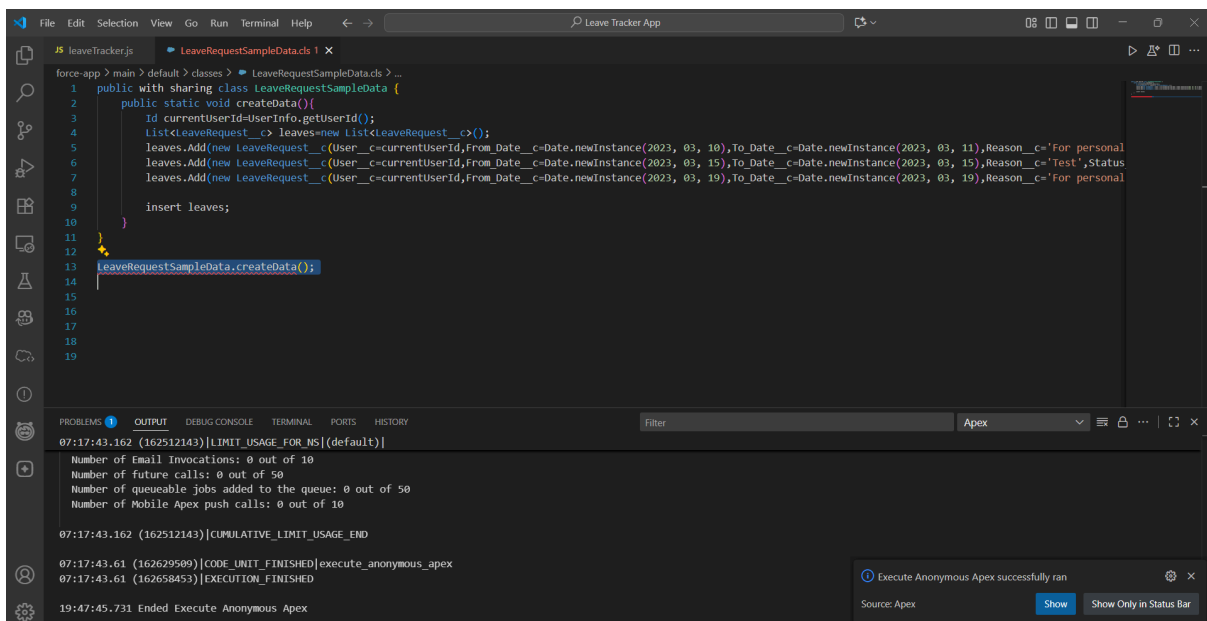
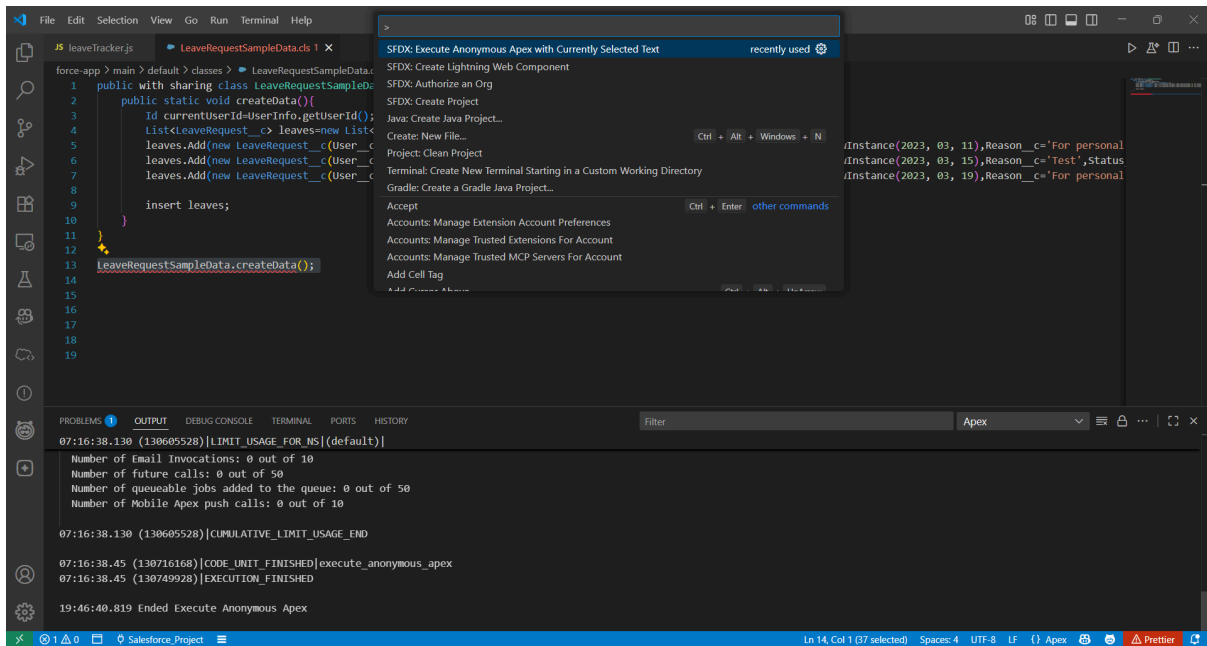


VS Code SFDX Project Structure showing LWC and Classes folders

## Data Management via Execute Anonymous

**Use Case:** Quickly seeding the Org with test data for continuous development and testing of the UI components.

**Explanation:** The Execute Anonymous Apex feature in VS Code was used to call the `LeaveRequestSampleData.createData()` method, ensuring the application always has test data available with different statuses to validate the grid logic.



VS Code Debug Log showing successful Execute Anonymous run

## Phase 9: Reporting, Dashboards & Security Review

### Security Review (Hierarchical Filtering)

**Use Case:** Enforcing data visibility restrictions based on the organization's reporting structure.

**Explanation:** Security for the leaveRequests component is enforced at the Apex layer (server-side security) using SOQL that filters records based on the standard ManagerId field on the User record. This ensures that a manager cannot view the leave requests of peers or employees not directly reporting to them.

```
LeaveRequestController.cls X
force-app > main > default > classes > LeaveRequestController.cls > ...
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__c=:UserInfo.getUserId() ORDER BY CreatedDate DESC];
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 FROM LeaveRequest__c
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 }
```

Apex SOQL query showing ManagerId filtering

## Phase 10: Quality Assurance Testing

### Testing Approach

The QA process relied heavily on **Functional Testing** of the LWC UI and **System Integration Testing** to validate the complex sibling component communication pattern.

### Test Case 1: LWC Client-Side Date Validation

Use Case / Scenario	Test Steps (with input)	Expected Result	Actual Result (with Screenshot)
<b>Past Date Prevention</b>	1. Open New Request. 2. Select From_Date as yesterday. 3. Click Save.	Validation logic should prevent the default save action and display the Toast error: "from date should not be less than today."	Error toast message displayed successfully. [Insert Screenshot: Toast Message Error on Past Date]

### Test Case 2: Sibling Communication & Auto-Refresh

Use Case / Scenario	Test Steps (with input)	Expected Result	Actual Result (with Screenshot)
<b>Real-Time Grid Update</b>	1. Log in as Employee A. 2. Manager B is logged in and viewing the Leave Requests tab. 3. Employee A submits a new request.	1. Employee A's grid refreshes via refreshApex. 2. Manager B's grid (Sibling) updates automatically (without manual refresh) via the Custom Event/Public Method call.	Manager's grid shows the new entry immediately. [Insert Screenshot: Manager Grid showing new request after auto-refresh]

### Test Case 3: Manager Approval and Data Service

Use Case / Scenario	Test Steps (with input)	Expected Result	Actual Result (with Screenshot)
<b>Manager Update via Form</b>	1. Manager B opens a subordinate's Pending request for edit. 2. Changes Status__c to 'Approved'. 3. Clicks Save.	1. lightning-record-edit-form updates the database. 2. Manager's grid refreshes via refreshApex. 3. The row's background color changes to <b>Green</b> .	Row is successfully updated to Green and the Edit button is disabled. [Insert Screenshot: Manager Grid Row turns Green on Approval]

## Conclusion

The **Leave Tracker Management App** successfully transforms a manual process into a highly efficient, automated system using the Salesforce platform. The implementation successfully leverages **Apex, LWC, Custom Events, and refreshApex** to deliver a responsive, secure, and intuitive user experience for both employees and managers, meeting all the specified objectives and demonstrating proficiency in advanced Salesforce development techniques.

### Future Scope & Enhancements

1. **Batch & Scheduled Apex:** Implement the planned **Scheduled Apex** job to automate monthly leave accrual and balance updates, ensuring financial compliance.
2. **LWC for Mass Action:** Develop a dedicated **LWC** component for Managers to mass-approve/reject multiple requests from a single screen on the `leaveRequests` tab.
3. **Calendar Integration:** Fully implement **Named Credential** and Apex callouts to external APIs (e.g., Google/Outlook Calendar) to automatically block employee calendars upon leave approval.