# Leave Management App

Prepared by: SHAIK BAJI

**Project Title**: Leave Management App
**Industry**: Human Resources (HR) / Workforce Management
**Project Type**: Custom CRM Application (Built on Salesforce Platform using Lightning Web Components, custom objects, and Apex)
**Target Users**: Employees, managers, and HR administrators

**Problem Statement**:
Traditional leave management processes are often manual, fragmented, and inefficient, leading to challenges such as delayed leave approvals, lack of visibility into employee leave statuses, inconsistent tracking of leave requests, and difficulties in ensuring compliance with organizational policies. These inefficiencies can result in employee dissatisfaction, administrative errors, and disrupted workforce planning.

## **Phase 1**: Project Setup and Initialization

**Objective**: Set up the development environment and create the Salesforce project structure.

- **Tasks**:
  - Installed Visual Studio Code (VS Code) with Salesforce CLI and extensions.
  - Created a new Salesforce project using the command palette (Ctrl+Shift+P) and selected the "SFDX: Create Project" option with the standard template.
  - Named the project "LeaveTrackerApp."
  - Authorized the Salesforce org using the command palette (SFDX: Authorize an Org) with the alias "LiveProject" and default developer org settings.
  - Verified successful org connection in VS Code.

## **Phase 2:** Object Creation

**Objective**: Create the Leave_Request__c custom object to store leave request data.

- **Tasks**:
  - Downloaded the Leave_Request__c object metadata from the provided GitHub repository.
  - Copied the object folder into the project's force-app/main/default/objects directory.
  - Deployed the object to the Salesforce org using SFDX: Deploy Source to Org.
  - Verified the object in Salesforce Setup under Object Manager.
  - Created fields: From_Date__c (Date), To_Date__c (Date), Reason__c (Text), Status__c (Picklist: Pending, Approved, Rejected), Manager_Comment__c (Text), and User__c (Lookup to User).
  - Assigned object and field-level permissions to the System Administrator profile manually via Setup.

## **Phase 3:** Apex Classes Development

**Objective**: Develop Apex classes to handle data operations and create sample data.

- **Tasks**:
  - Copied two Apex classes from the GitHub repository: LeaveRequestSampleData and LeaveRequestController.
  - LeaveRequestSampleData.cls:
    - Contains a static method createData() to generate three sample leave requests for the current user.
    - Executed the method via VS Code's SFDX: Execute Anonymous Apex with Currently Selected Text to populate dummy data.
  - LeaveRequestController.cls:
    - Contains two @AuraEnabled methods:
      - getMyLeaves(): Retrieves leave requests for the current user with fields Id, Name, From_Date__c, To_Date__c, Reason__c, Status__c, and Manager_Comment__c.
      - getLeaveRequests(): Retrieves leave requests where the user's manager is the current user, including additional fields like User__r.Name and User__r.ManagerId.
  - Deployed both classes to the org using SFDX: Deploy Source to Org.

## **Phase 4**: Lightning Web Components Creation

**Objective**: Create LWC components for the application's UI.

- **Tasks**:
  - Created three LWC components using SFDX: Create Lightning Web Component:
    - leaveTracker: Parent component to host tabs.
    - myLeaves: Child component for the "My Leaves" tab.
    - leaveRequest: Child component for the "Leave Requests" tab.
  - Updated the leaveTracker component's meta XML file to set isExposed to true and added lightning__AppPage as a target.
  - Deployed all components to the org.

## **Phase 5**: Lightning App and Page Setup

**Objective**: Create a Lightning App and App Page to host the LWC components.

- **Tasks**:
  - Navigated to Setup > App Manager and created a new Lightning App named "Leave Tracker App."
  - Selected the System Administrator profile for access and skipped optional settings.
  - In Lightning App Builder, created a new App Page named "Leave Tracker" with a single-column layout.
  - Dragged the leaveTracker component onto the page, saved, and activated it.
  - Added the page to the "Leave Tracker App" as a tab.
  - Verified the app in the App Launcher, ensuring the leaveTracker component displayed correctly.

## **Phase 6**: Implementing Tabs in Parent Component

**Objective**: Add tabs to the leaveTracker component to switch between "My Leaves" and "Leave Requests."

- **Tasks**:
  - Modified leaveTracker.html to include a lightning-tabset with two tabs:
    - "My Leaves" tab calling the c-my-leaves component.
    - "Leave Requests" tab calling the c-leave-request component.
  - Added labels for tabs and sample text in child components (myLeaves.html and leaveRequest.html) for initial testing.
  - Redeployed the components and verified tab functionality in the org.

## **Phase 7**: My Leaves Tab Implementation

**Objective**: Implement the "My Leaves" tab with a data table, modal popup, and client-side validations.

- **Tasks**:
  - **HTML (myLeaves.html)**:
    - Added a lightning-card with a lightning-datatable to display leave requests.
    - Configured the datatable with key-field="Id", data={myLeaves}, and columns={columns}.
    - Added a conditional div with SLDS classes to display "No records found" when myLeaves is empty, using lwc:if directive and a noRecords getter.
    - Created a modal popup using SLDS markup with a lightning-record-edit-form for adding/editing leave requests.
    - Included fields: User__c, From_Date__c, To_Date__c, Reason__c, with Save and Cancel buttons.
    - Added a lightning-button-icon for adding new requests.
  - **JavaScript (myLeaves.js)**:
    - Imported getMyLeaves Apex method and wired it to fetch leave data.
    - Defined columns for the datatable, including an edit button with conditional disabling based on Status__c.
    - Added conditional row styling using cellAttributes with SLDS classes (slds-theme_success for Approved, slds-theme_warning for Rejected).
    - Implemented modal visibility with a showModalPopup property.
    - Set the current user as the default for User__c using @salesforce/user/Id.
    - Added client-side validations in the onsubmit handler to check:
      - From_Date__c is not in the past.
      - From_Date__c is not greater than To_Date__c.
    - Used refreshApex to auto-refresh the grid after saving.
    - Handled onsuccess to show a success toast and close the modal.
    - Added rowaction handler for edit button clicks to populate the form with record data.
  - **Deployment**:
    - Deployed and tested the component, ensuring the grid displayed data, validations worked, and the modal functioned correctly.

**Phase 8**: Leave Requests Tab Implementation

**Objective**: Implement the "Leave Requests" tab for managers to review subordinate leave requests.

- **Tasks**:
  - **HTML (leaveRequest.html)**:
    - Copied myLeaves.html and removed the add button.
    - Updated the modal title to "Leave Request Details."
    - Changed User__c, From_Date__c, To_Date__c, and Reason__c to lightning-output-field for read-only display.
    - Added Status__c and Manager_Comment__c as lightning-input-field for editing.
    - Removed the onsubmit handler as no validations were needed.
  - **JavaScript (leaveRequest.js)**:
    - Replaced getMyLeaves with getLeaveRequests Apex method.
    - Updated property names from myLeaves to leaveRequests for consistency.
    - Added a Username column by mapping User__r.Name to a new username property in the data.
    - Added a public refreshGrid method with @api decorator to refresh the grid.
  - **Deployment**:
    - Deployed and tested, ensuring the grid displayed subordinate leave requests and the modal allowed status and comment updates.

**Phase 9**: Inter-Component Communication

**Objective**: Enable automatic grid refresh in the "Leave Requests" tab when a new leave is created in "My Leaves."

- **Tasks**:
  - In myLeaves.js, created a custom event leaverequestsave and dispatched it in the onsuccess handler.
  - In leaveTracker.html, added an onleaverequestsave listener to the c-my-leaves component.
  - In leaveTracker.js, defined the handler to call the refreshGrid method of the c-leave-request component using the lwc:ref attribute.
  - Corrected an initial error where the lwc:ref was incorrectly applied to c-my-leaves instead of c-leave-request.
  - Deployed and tested, ensuring the "Leave Requests" grid refreshed automatically after a new leave was saved.

**Phase 10**: Testing and Finalization

**Objective**: Perform comprehensive testing and finalize the application.

- **Tasks**:
  - Tested the "My Leaves" tab:
    - Verified grid display with conditional row colors (green for Approved, yellow for Rejected).
    - Confirmed edit button was disabled for non-Pending requests.
    - Tested modal popup for adding/editing with pre-filled values and validations.
    - Ensured grid auto-refreshed after saving.

- Tested the "Leave Requests" tab:
  - Verified grid displayed subordinate requests with the Username column.
  - Confirmed read-only fields and editable Status__c and Manager_Comment__c in the modal.
  - Ensured grid auto-refreshed after new leave creation.
- Set up a test user (e.g., "Manoj") with the current user as the manager in Setup > Users.
- Created additional leave requests to validate grid updates and manager approvals.
- Fixed minor issues, such as the To_Date__c field spelling in leaveRequest.js.
- Documented the project and ensured all components were deployed.