Phase 3: Data Modeling & Object Creation

Leave Management using Salesforce LWC

Overview

The "Leave Management" object contains several critical fields ensuring accurate data capture, workflow automation, and an intuitive user experience. Each field is configured to meet specific business requirements and supports the intended leave management process.

The "Leave Management" object contains several critical fields ensuring accurate data capture, workflow automation, and an intuitive user experience. Each field is configured to meet specific business requirements and supports the intended leave management process.

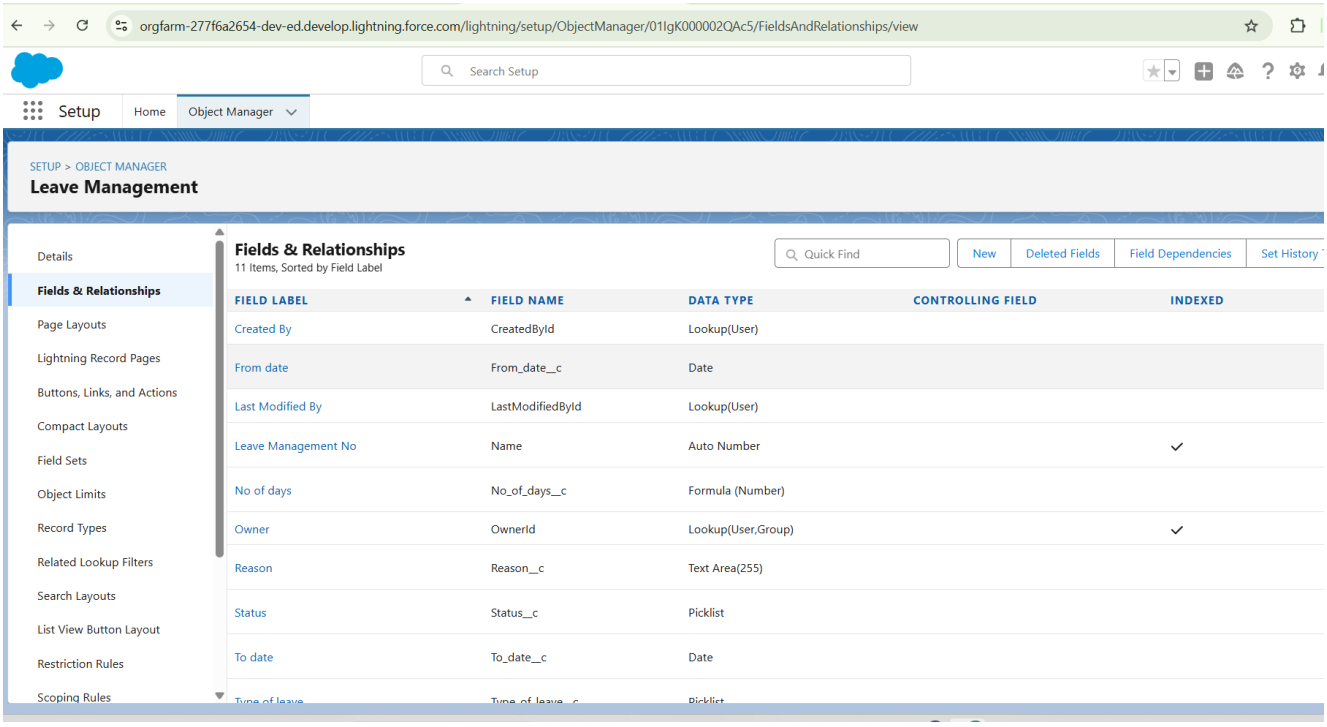
- Type of Leave
  This is a Picklist field and is mandatory for creating a record. It allows selection from manually entered leave types such as Sick or Casual, ensuring every leave request specifies its category.
- From Date
  Captured as a Date field, this is required to indicate the start date of the employee's leave period.
- To Date
  Also a required Date field, it specifies the end date of the requested leave, completing the duration definition for the absence.
- Reason
  This optional field uses a Text Area data type and allows the user to input the reason for leave, supporting up to 255 characters for details or explanations.
- User
  Configured as a Lookup to the User object, this field is not required but enables associating the leave record with any employee—including creating requests for other users if needed.
- Status
  This Picklist field is required and reflects the current workflow state of the leave request. Possible values include Pending, Approved, or Rejected, with the default set to Pending for new applications.
- Number of Days
  A Formula(Number) field, always required, calculates the total days of leave using the formula To_Date__c - From_Date__c + 1. This ensures the duration is updated automatically based on the dates provided

---

1. Add Fields to Objects

1.1 Event__c Fields

- Type_of_Leave__c (Picklist): Mandatory field to capture the type of leave (e.g., Sick, Casual).
- From_Date__c (Date): Required field indicating the start date of the leave.
- To_Date__c (Date): Required field indicating the end date of the leave.
- Reason__c (Text Area): Optional field with a maximum length of 255 characters to provide the reason for leave.
- User__c (Lookup(User)): Optional field linking to the User object to specify the employee requesting leave; enables leave application on behalf of others.
- Status__c (Picklist): Required field with values such as Pending, Approved, and Rejected; defaults to Pending for new entries.
- Number_of_Days__c (Formula - Number): Required formula field calculating the duration of the leave using the formula: To_Date__c - From_Date__c + 1.



Relationships

- User Relationship: Leave record references User object for requestor.
- The leave request may be submitted on behalf of another user via lookup.

Page Layouts & Compact Layouts

- Add all core fields to default page layout.
- Compact layout: Leave Type, From Date, To Date, Status, Number of Days.

Field-Level Security

- Ensure visibility/edit access across manager/employee profiles.
- Restrict status updates to managers only (using profile or permission set).

Validation Rules

- *Type of Leave, From Date, To Date* must always be entered.
- Formula sample: ISBLANK(TEXT(Type_of_Leave__c)) triggers "Type of Leave is required" error.

Automation & Formula Fields

- Number of Days formula: To_Date__c - From_Date__c + 1 (ensures inclusive calculation).

Object Security Permissions

- System Administrator: Full (Read, Create, Edit, Delete).
- Employees: Create & View own leave requests.
- Managers: Approve/Reject status and view subordinate records.

Deployment Checklist

- Tab created and set On for Leave Management app.
- Page layout and compact layout configured as above.
- Test: Create leave records, verify automatic field formula, status transitions, and validation

Phase 4: Business Logic, Validation & Automation

- **Validation Rules:**
  - Ensure mandatory fields are filled when creating or editing leave records.
  - Example: "Type of Leave" must not be blank.
  - Validate date logic if needed, such as "From Date" should not be later than "To Date".
  - Enforce status transitions only by authorized roles (e.g., only managers can change status from Pending to Approved or Rejected).

- **Formula Field Logic:**
  - Number_of_Days__c automatically calculates total leave days based on the formula:
    To Date−From Date+1To Date−From Date+1
  - Provides real-time update whenever dates change.

- **Approval Process (Optional):**
  - Automate manager approval workflows for leave requests.
  - Notify users upon approval or rejection using email templates or platform notifications.

- **Apex Triggers / Classes:**
  - Handle complex validation or automation that cannot be configured declaratively.
  - For example, to update related records or send custom notifications upon status changes.

- **Lightning Web Component Automation:**
  - UI buttons trigger actions such as Apply, Approve, Reject.
  - Display toast messages for success/failure feedback.
  - Dynamic UI refresh on leave record updates to ensure current status visibility.

- **Permissions & Security:**
  - Lock status picklist field to prevent unauthorized changes.
  - Configure Salesforce profiles and permission sets to enforce data and action access control.

Phase 5: Apex Programming

Overview

In this phase, customized Apex controller methods are created to efficiently handle backend queries for Lightning Web Components (LWC), enabling dynamic retrieval of leave records based on user context.

- **Class: LeaveManagementHandler**
  The class is annotated with @AuraEnabled to expose methods to Lightning components,
  with cacheable=true to optimize performance by allowing client-side caching of results.

- **Methods:**

  - **getPendingLeaves()**
    Returns the list of Leave Management records where the status is "Pending" and the requesting user is the owner (applicant).
    Fields fetched: Id, Name, Type of Leave, From Date, To Date, Reason, User (lookup), User Name, Status, Number of Days.
    This method employs a filter on the current user's ID (UserInfo.getUserId()) to show only their pending leaves.

  - **getLeavesHistory()**
    Retrieves all leave records in the system, regardless of user or status, showing complete leave history fields similar to above. Useful for system-wide reporting or admin views.

  - **getLeaves()**
    Fetches leave requests of users managed by the current user, identified by referencing the User's ManagerId against the current user's ID. This allows managers to see leave requests pending or processed for their team.

- **Debug Statements:**
  Each method includes System.debug() logs for monitoring query results during development and troubleshooting.

This Apex layer underpins the interactive, user-filtered display of leave records in the Leave Management LWC, providing scalable and secure data access for applicants and managers as demonstrated in the referenced YouTube tutorial.

This summary builds developer clarity on the Apex backend supporting the leave retrieval functionality of the project. Let me know if you want me to explain this further or generate additional Apex examples.Phase 5: Apex Programming for your Leave Management project involves creating Apex controller methods that serve Lightning Web Components (LWC) with filtered leave data based on user roles and statuses.

- The LeaveManagementHandler class is annotated with @AuraEnabled(cacheable=true) to enable invocation from LWC and cache data on the client side.
- The getPendingLeaves() method returns pending leave records submitted by the current logged-in user by filtering Status__c = 'Pending' and matching User__c with the current user ID.
- The getLeavesHistory() method fetches all leave records to provide a comprehensive leave history view.
- The getLeaves() method retrieves leave requests for users who report to (have ManagerId of) the logged-in user, allowing managers to see their team's leaves.
- These methods select key leave fields including Id, Name, Type_of_Leave, From Date, To Date, Reason, User, Status, and Number of Days.
- Debug logs are included to output the queried leave records for monitoring during development.

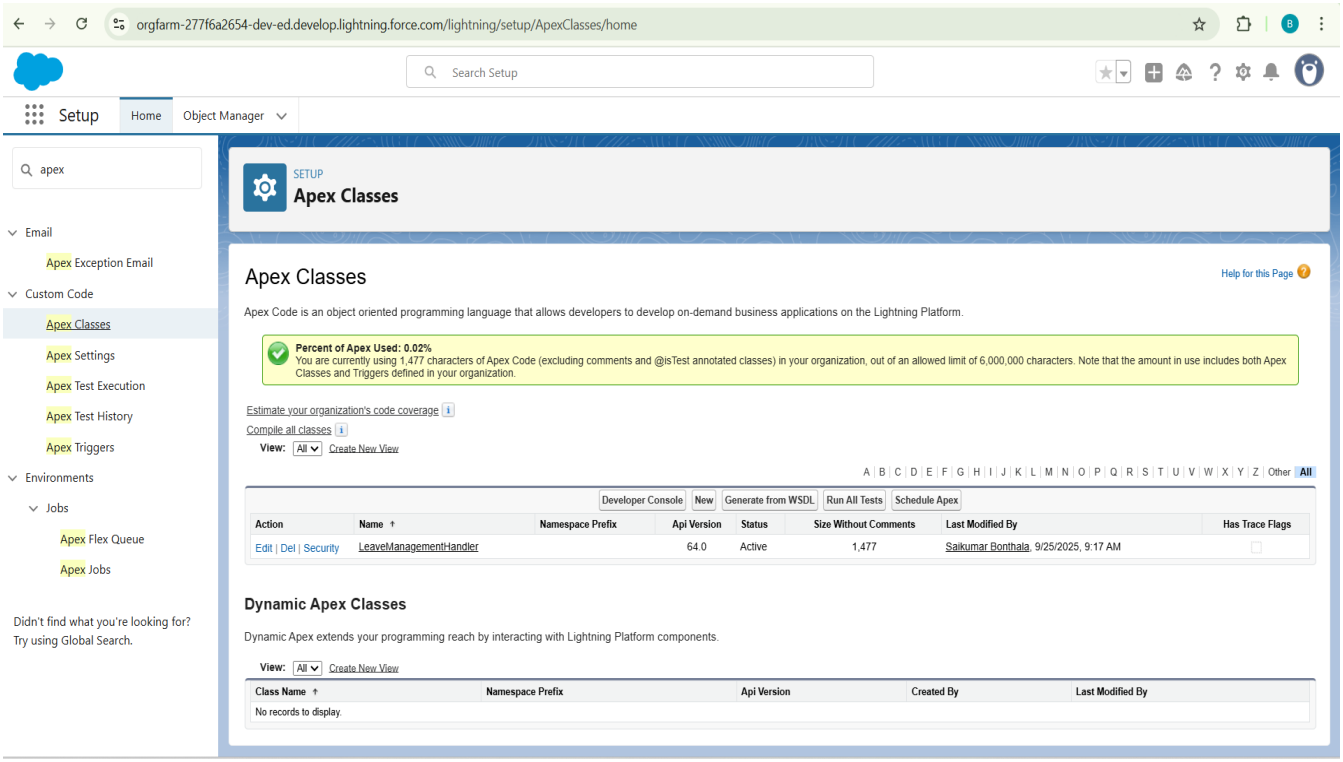
Code Summary

```
public class LeaveManagementHandler {
    @AuraEnabled(cacheable=true)
    public static List<Leave_Management__c> getPendingLeaves() {
        List<Leave_Management__c> pendingLeaves = [
            SELECT Id, Name, Type_of_leave__c, From_date__c, To_date__c,
                Reason__c, User__c, User__r.Name, Status__c, No_of_days__c
            FROM Leave_Management__c
            WHERE Status__c = 'Pending' AND User__c=: userInfo.getUserId()
        ];
        System.debug('pendingLeaves == ' + pendingLeaves);
        return pendingLeaves;
    }

    @AuraEnabled(cacheable=true)
    public static List<Leave_Management__c> getLeavesHistory() {
        List<Leave_Management__c> leavesHistory = [
            SELECT Id, Name, Type_of_leave__c, From_date__c, To_date__c,
                Reason__c, User__c, User__r.Name, Status__c, No_of_days__c
            FROM Leave_Management__c
        ];
        System.debug('leavesHistory == ' + leavesHistory);
        return leavesHistory;
    }
    @AuraEnabled(cacheable=true)
    public static List<Leave_Management__c> getLeaves() {
        List<Leave_Management__c> leaves = [
            SELECT Id, Name, Type_of_leave__c, From_date__c, To_date__c,
                Reason__c, User__c, User__r.Name, Status__c, No_of_days__c
```

```
        FROM Leave_Management__c
        WHERE User__r.ManagerId=: userInfo.getUserId()
    ];
    System.debug('leaves == ' + leaves);
    return leaves;
  }
}
```

Apex Class(LeaveManagementHandler)



Usage in Project

The LeaveManagementHandler Apex class plays a crucial role in the Leave Management Salesforce project by enabling efficient data retrieval for Lightning Web Components (LWC) and supporting role-based data access.

Phase 6: User Interface Development

In this phase, a user-friendly Lightning Experience interface was developed to enable employees and managers to manage leave requests through intuitive navigation and interactive components. The approach involved creating a dedicated Lightning App, customizing record pages, building Lightning Web Components (LWC) integrated with Apex, and configuring tabs for user convenience and enhanced usability.

1. Created Lightning App - Leave Management

- Navigated to Setup → App Manager → New Lightning App.
- Defined the app as "Leave Management," with suitable branding and standard Lightning navigation.
- Added custom object tabs for Leave Management.
- Assigned app visibility primarily to System Administrator, Manager, and Employee profiles.
- Saved and activated the app for organizational usage.

2. Customized Lightning Record Page for Leave Management

- Opened a Leave Management record and launched Edit Page in the Lightning App Builder.
- Added the Record Detail component to display key fields including Type of Leave, From Date, To Date, Status, and Number of Days.
- Included related lists or custom datatable components as needed to show leave history or pending requests.
- Activated this page layout as the default Lightning Record Page for the Leave Management object.

3. Developed Lightning Web Components for Leave Management

- Created various LWCs such as:
    - LeaveApplyForm: A modal form for employees to submit new leave requests.
    - LeavePendingList: Displays pending leave requests filtered by user or manager context.
    - LeaveHistory: Shows both employees and managers the history of leave requests.
- Designed component UI using SLDS (Salesforce Lightning Design System) for consistent look and responsive design.
- Handled state, user input, and validation within component JavaScript files.
- Incorporated imperative Apex calls to LeaveManagementHandler class to fetch filtered leave data.
- Used Lightning toast notifications to give users real-time feedback on actions like submitting or approving leaves.
- Placed these LWCs on relevant Lightning Record Pages and App Pages via Lightning App Builder.
- Activated all changes, enabling smooth leave request workflows with inline updates from both employee and manager perspectives.

leaveManagement.html

```
<template>
   <lightning-tabset>
      <lightning-tab label="Employee">
         <c-leave-employee></c-leave-employee>
      </lightning-tab>
      <lightning-tab label="Manager">
         <c-leave-management-manager></c-leave-management-manager>
      </lightning-tab>
   </lightning-tabset>
</template>
```

The provided code snippet represents a simple Lightning Web Component (LWC) template in Salesforce that implements a tabbed interface using the lightning-tabset component. It contains two tabs labeled "Employee" and "Manager," each loading a corresponding custom component. Here's an explanation and usage context within a project:

Lightning Tabset Component Template Explanation

- <lightning-tabset>:
  This is the container component that defines a tabbed interface. It manages multiple tabs and ensures only one tab content is visible at a time.
- <lightning-tab label="Employee">
  Defines the first tab labeled "Employee". When this tab is active, the custom component <c-leave-employee> is loaded and displayed. This component likely handles leave application and status for employees.
- <lightning-tab label="Manager">
  Defines the second tab labeled "Manager". It loads <c-leave-management-manager>, a custom component likely designed for managers to view and approve/reject leave requests.

Usage in Leave Management Project

- This tabset is used as a parent container for two key user interfaces in the Leave Management app:
  - The Employee tab gives end users (employees) access to leave application forms and their leave status.
  - The Manager tab provides managerial views and controls over subordinate leave requests.
- It organizes functionality cleanly in one place, improving navigation and user experience without page reloads.
- Each tab content is encapsulated within its respective component, enhancing modularity and maintainability.
- This approach follows Salesforce Lightning Design System best practices for a responsive, accessible tabbed interface in Lightning Experience.

leaveEmployee.html

```
<template>
  <lightning-card>
    <lightning-button variant="base" label={labelHideShow} onclick={handleClickButton}
      class="slds-var-m-around_medium"></lightning-button>
    <template if:true={isButtonClicked}>
      <div class="slds-box slds-var-m-around_medium">
        Leave is earned by an employee and granted by the employer to take time off from the work. The
employee
        is free to avail this leave in accordance with the company policy
      </div>
    </template>
  </lightning-card>

  <lightning-button-group class="slds-align_absolute-center">
    <lightning-button label="Apply" onclick={handleApplyLeave}></lightning-button>
    <lightning-button label="Pending" onclick={handlePendingLeave}></lightning-button>
    <lightning-button label="History" onclick={handleLeaveHistory}></lightning-button>
  </lightning-button-group></br>

  <template if:true={isApplyLeave}>
    <section role="dialog" tabindex="-1" aria-modal="true" aria-labelledby="modal-heading-01"
```

```
        class="slds-modal slds-fade-in-open slds-modal_medium">
        <div class="slds-modal__container">
            <button onclick={handleClose} class="slds-button slds-button_icon slds-modal__close slds-
button_icon-inverse">
                <lightning-icon icon-name="utility:close" alternative-text="Close" variant="inverse"></lightning-
icon>
                <span class="slds-assistive-text"> Cancel and Close</span>
            </button>
            <div class="slds-modal__header">
                <h1 id="modal-heading-01" class="slds-modal__title slds-hyphenate" tabindex="-1"> Apply
Leave
                </h1>
            </div>
            <div class="slds-modal__content slds-p-around_medium" id="modal-content-id-1">
                <lightning-record-edit-form id="recordViewForm" object-api-name="Leave_Management__c"
                    onsuccess={handleSave}>
                    <lightning-input-field field-name="User__c"></lightning-input-field>
                    <lightning-input-field field-name="Type_of_leave__c"></lightning-input-field>
                    <lightning-input-field field-name="From_date__c"></lightning-input-field>
                    <lightning-input-field field-name="To_date__c"></lightning-input-field>
                    <lightning-input-field field-name="Reason__c"></lightning-input-field>
                    <lightning-input-field field-name="Status__c"></lightning-input-field>
                    <lightning-input-field field-name="No_of_days__c"></lightning-input-field>
                    <div class="slds-align_absolute-center">
                        <lightning-button variant="brand" type="submit" label="Save" class="slds-m-left_small">
                        </lightning-button>
                        <lightning-button variant="brand" label="Cancel" onclick={handleClose}
                            class="slds-m-left_small"> </lightning-button>
                    </div>
                </lightning-record-edit-form>
            </div>
        </div>
    </section>
    <div class="slds-backdrop slds-backdrop_open" role="presentation"></div>
```

```
    </template>

    <template if:true={isPendingLeave}>
        <lightning-datatable key-field="Id" data={dataList} show-row-number-column row-number-offset={rowOffset}
            hide-checkbox-column columns={columnsList} onrowaction={handleRowAction}>
        </lightning-datatable>
    </template>

    <template if:true={isLeaveHistory}>
        <lightning-datatable key-field="Id" data={dataList} show-row-number-column row-number-offset={rowOffset}
            hide-checkbox-column columns={columnsList} onrowaction={handleRowAction}>
        </lightning-datatable>
    </template>

    <template if:true={isEdit}>
        <section role="dialog" tabindex="-1" aria-modal="true" aria-labelledby="modal-heading-02"
            class="slds-modal slds-fade-in-open slds-modal_medium">
            <div class="slds-modal__container">
                <button onclick={handleClose} class="slds-button slds-button_icon slds-modal__close slds-button_icon-inverse">
                    <lightning-icon icon-name="utility:close" alternative-text="Close" variant="inverse"></lightning-icon>
                    <span class="slds-assistive-text"> Cancel and Close</span>
                </button>
                <div class="slds-modal__header">
                    <h1 id="modal-heading-02" class="slds-modal__title slds-hyphenate" tabindex="-1"> Edit Leave
                    </h1>
                </div>
                <div class="slds-modal__content slds-p-around_medium" id="modal-content-id-2">
                    <lightning-record-edit-form id="recordViewForm1" object-api-name="Leave_Management__c"
                        onsuccess={handleSave}>
                        <lightning-input-field field-name="User__c"></lightning-input-field>
```

```
                <lightning-input-field field-name="Type_of_leave__c"></lightning-input-field>
                <lightning-input-field field-name="From_date__c"></lightning-input-field>
                <lightning-input-field field-name="To_date__c"></lightning-input-field>
                <lightning-input-field field-name="Reason__c"></lightning-input-field>
                <lightning-input-field field-name="Status__c"></lightning-input-field>
                <lightning-input-field field-name="No_of_days__c"></lightning-input-field>
                <div class="slds-align_absolute-center">
                  <lightning-button variant="brand" type="submit" label="Save" class="slds-m-left_small">
                  </lightning-button>
                  <lightning-button variant="brand" label="Cancel" onclick={handleClose}
                    class="slds-m-left_small">
                  </lightning-button>
                </div>
              </lightning-record-edit-form>
            </div>
          </div>
        </section>
        <div class="slds-backdrop slds-backdrop_open" role="presentation"></div>
    </template>
</template>
```

This Lightning Web Component (LWC) template embodies the core user interaction for your Leave Management project, providing multiple key UI elements and functionalities as described below:

Detailed UI Component Breakdown for Leave Management

- Information Toggle Button:
  A button that dynamically toggles the display of an informational text box detailing the company's leave policy. This helps users understand leave usage without cluttering the interface.
- Navigation Button Group (Apply, Pending, History):
  - Apply: Opens a modal form allowing employees to submit new leave requests.
  - Pending: Displays a data table listing all leave requests currently pending approval.
  - History: Shows a comprehensive leave record history in a similar table format.
- Apply Leave Modal:
  A modal dialog for creating leave records using lightning-record-edit-form bound to the

Leave_Management__c object. It includes fields for User, Type of Leave, From Date, To Date, Reason, Status, and Number of Days. Submission triggers success handling logic.

- Pending Leave Data Table:
  Shows pending leave requests with appropriate columns and supports row actions.
- Leave History Data Table:
  Displays all leave records, enabling users or managers to review past leaves.
- Edit Leave Modal:
  Similar to the apply modal, this dialog is for editing existing leave records. It supports field modifications and cancel/save functionality.
- Modal and Styling:
  SLDS styling ensures consistent Salesforce UI appearance. Accessibility attributes (aria-modal, roles) are included for screen readers and accessibility compliance.

Role in Leave Management Salesforce Project

This template provides a seamless, intuitive UI for employees and managers to interact with leave data and workflows. It encapsulates:

- Leave application creation
- Viewing and processing pending requests
- History review
- Inline editing of leave records

This LWC structure enhances user experience through dynamic view switching, modals, and data presentation aligned with Salesforce Lightning best practices, as demonstrated in the project tutorial video.This Lightning Web Component template provides a user interface for your Leave Management project with the following functionalities:

- A toggle button shows or hides an informational box describing leave policy.
- A button group with "Apply", "Pending", and "History" buttons for switching views.
- A modal dialog for applying leave with fields bound to the Leave_Management__c object, supporting Save and Cancel actions.
- A datatable to display pending leave requests when the Pending view is active.
- A datatable to display leave history when the History view is active.
- A modal dialog for editing existing leave records, with fields bound similarly and Save/Cancel functionality.

employeeLeave.js

```
import { LightningElement, track } from 'lwc';
import { ShowToastEvent } from 'lightning/platformShowToastEvent';
import getPendingLeaves from '@salesforce/apex/LeaveManagementHandler.getPendingLeaves';
import getLeavesHistory from '@salesforce/apex/LeaveManagementHandler.getLeavesHistory';

export default class LeaveEmployee extends LightningElement {
    @track isButtonClicked = true;
    @track labelHideShow = 'Hide';
    @track isApplyLeave = false;
    @track isPendingLeave = false;
    @track isLeaveHistory = false;
    @track isEdit = false;
    @track recordId = '';
    @track dataList = [];
    @track columnsList = [
        { label: 'User', fieldName: 'userName' },
        { label: 'Type of leave', fieldName: 'Type_of_leave__c' },
        { label: 'From date', fieldName: 'From_date__c' },
        { label: 'To date', fieldName: 'To_date__c' },
        { label: 'Reason', fieldName: 'Reason__c' },
        {
            label: 'Status', fieldName: 'Status__c', cellAttributes: {
                class: {
                    fieldName: 'statuscss'
                }
            }
        },
        { label: 'No of days', fieldName: 'No_of_days__c' },
        {
            type: 'button',
            typeAttributes: {
                label: 'Edit',
                disabled : {fieldName : 'isEditdisabled'}
```

```
        }
    }
];

handleClickButton() {
    this.isButtonClicked = !this.isButtonClicked;
    this.labelHideShow = this.isButtonClicked ? 'Hide' : 'Show';
}

handleApplyLeave() {
    this.isApplyLeave = true;
    this.isLeaveHistory = false;
    this.isPendingLeave = false;
}

handleClose() {
    this.isApplyLeave = false;
    this.isEdit = false;
}

handleSave(event) {
    const eve = new ShowToastEvent({
        title: 'Success',
        message: 'Record Created/Updated Successfully : Id = ' + event.detail.id,
        variant: 'success'
    });
    this.dispatchEvent(eve);
    this.isApplyLeave = false;
    this.isEdit = false;
    this.isLeaveHistory = false;
    this.isPendingLeave = false;
}

handlePendingLeave() {
```

```
        this.isPendingLeave = true;
        this.isApplyLeave = false;
        this.isLeaveHistory = false;
        getPendingLeaves()
            .then((result) => {
                console.log('result==', JSON.stringify(result));
                this.dataList = result.map(a => {
                    return {
                        ...a, userName: a.User__r != undefined ? a.User__r.Name : '',
                        isEditdisabled : a.Status__c !='Pending',
                        statuscss: a.Status__c == 'Approved' ? 'slds-theme_success' : a.Status__c == 'Rejected' ? 'slds-theme_error' : 'slds-theme_warning'
                    }
                });
            })
            .catch((error) => {
                this.dispatchEvent(
                    new ShowToastEvent({
                        title: 'Error loading pending leaves',
                        message: error.body ? error.body.message : error.message,
                        variant: 'error'
                    })
                );
            });
    }

    handleLeaveHistory() {
        this.isLeaveHistory = true;
        this.isPendingLeave = false;
        this.isApplyLeave = false;
        getLeavesHistory()
            .then((result) => {
                this.dataList = result.map(a => {
                    return {
```

```
                ...a, userName: a.User__r != undefined ? a.User__r.Name : '',
                isEditdisabled : a.Status__c !='Pending',
                statuscss: a.Status__c == 'Approved' ? 'slds-theme_success' : a.Status__c == 'Rejected' ? 'slds-theme_error' : 'slds-theme_warning'
            };
        });
      })
      .catch((error) => {
        this.dispatchEvent(
          new ShowToastEvent({
            title: 'Error loading leaves history',
            message: error.body ? error.body.message : error.message,
            variant: 'error'
          })
        );
      });
  }

  handleRowAction(event) {
    this.isEdit = true;
    this.recordId = event.detail.row.Id;
    console.log(' this.recordId', this.recordId);
  }
}
```

This Lightning Web Component JavaScript controller, named LeaveEmployee, manages the employee-facing UI logic for the Leave Management project. Here is a detailed explanation of its usage and functionality:

LeaveEmployee LWC JavaScript Controller Overview

- Reactive Properties:

  Manages interaction and state visibility using tracked properties (@track):
  - isButtonClicked and labelHideShow toggle display of leave info text.
  - Booleans isApplyLeave, isPendingLeave, isLeaveHistory, isEdit control visibility of related UI sections/modal dialogs.
  - recordId stores the current record id for editing.
  - dataList stores fetched leave records for display in tables.
  - columnsList defines columns for the leave record datatables including conditional styling and edit buttons.
- Button Handlers:
  - handleClickButton(): Toggles leave policy info visibility and button label between "Show" and "Hide".
  - handleApplyLeave(): Shows the leave application modal.
  - handlePendingLeave(): Shows the pending leave table and fetches pending leaves via Apex getPendingLeaves().
  - handleLeaveHistory(): Shows leave history table and fetches leave records via Apex getLeavesHistory().
- Data Fetching and Mapping:

  In handlePendingLeave and handleLeaveHistory, data returned from Apex is mapped to add display-friendly fields:
  - User name from related User object.
  - Disabling edit button if status is not "Pending".
  - Conditional cell styling based on status (Approved = success, Rejected = error, else warning).
- Row Action Handler:
  - handleRowAction(event): Opens the edit modal dialog for the clicked row, setting recordId for editing.
- Save and Close Handlers:
  - handleSave(event): Shows a success toast on save, closes modals, and resets view states.
  - handleClose(): Closes modals without saving.
- Error Handling:
  - Displays error toast with message from Apex for data loading failures.

leaveManagementManager.html

```
<template>
   <lightning-card>
      <lightning-datatable key-field="id" data={dataList} show-row-number-column row-number-
offset={rowOffset}
         hide-checkbox-column columns={columnsList} onrowaction={handleRowAction}>
      </lightning-datatable>
   </lightning-card>
</template>
```

This template snippet represents a Lightning Web Component (LWC) UI that renders a data table inside a Salesforce Lightning Card. Here's an explanation of its usage and functionality in the Leave Management project:

Lightning Data Table Component Usage

- Wrapper Component:
  The lightning-card provides a consistent Salesforce card UI container with standard padding and shadow.
- Data Table (lightning-datatable):
  Displays tabular data for leave records with these key configurations:
  - key-field="id": Uniquely identifies each row using the record Id.
  - data={dataList}: Binds to a tracked property holding leave record data fetched from Apex.
  - show-row-number-column: Enables a sequential row number column for easy record identification.
  - row-number-offset={rowOffset}: Supports pagination by offsetting row numbers.
  - hide-checkbox-column: Removes selection checkboxes for simpler UI and no bulk row selection.
  - columns={columnsList}: Columns are dynamically set, typically including User, Type of Leave, Dates, Status, and actions like Edit.
  - onrowaction={handleRowAction}: Event handler bound to row button clicks, such as opening the edit modal for the selected leave record.

Role in Leave Management UI

This data table is used to show lists of leave requests (pending or history views) with enhanced readability and interaction capabilities. It enables:

- Clear structured representation of multiple leave records.
- Interactive row actions for editing or approving leave.
- Integration with Apex backend by binding data dynamically.
- Consistent styling and accessibility by Salesforce Lightning standards.

This approach matches the project's functional requirements demonstrated in the YouTube tutorial, providing a scalable, user-friendly data display component for leave management.This template snippet represents the usage of a Lightning Data Table component inside a Lightning Card in your Leave Management LWC. It serves to display a structured, interactive table of leave records with these features:

- Wrapped inside a lightning-card for consistent Salesforce UI styling.
- lightning-datatable configured with:
    - key-field="id" for unique row identification.
    - Dynamic data binding via data={dataList} containing leave records.
    - Row numbering enabled with show-row-number-column and controlled pagination with row-number-offset={rowOffset}.
    - Disabled checkbox column for simplified UI.
    - Columns dynamically provided via columns={columnsList}, typically including user name, leave type, dates, status, and action buttons like Edit.
    - onrowaction={handleRowAction} event to handle row button clicks, e.g., opening an edit form.

leaveManagementManager.js

```
import { LightningElement, track } from 'lwc';
import { ShowToastEvent } from 'lightning/platformShowToastEvent';
import getLeaves from '@salesforce/apex/LeaveManagementHandler.getLeaves';

export default class LeaveManagementManager extends LightningElement {
    @track recordId = '';
    @track dataList = [];
    @track columnsList = [
        { label: 'User', fieldName: 'userName' },
```

```
    { label: 'Type of leave', fieldName: 'Type_of_leave__c' },
    { label: 'From date', fieldName: 'From_date__c' },
    { label: 'To date', fieldName: 'To_date__c' },
    { label: 'Reason', fieldName: 'Reason__c' },
    {
        label: 'Status', fieldName: 'Status__c', cellAttributes: {
            class: {
                fieldName: 'statuscss'
            }
        }
    },
    { label: 'No of days', fieldName: 'No_of_days__c' },
    {
        type: 'button',
        typeAttributes: {
            label: 'Edit',
            //disabled: { fieldName: 'isEditdisabled' }
        }
    }
];

connectedCallback() {
    this.handleLoad();
}

handleLoad() {
    getLeaves()
        .then((result) => {
            console.log('result==', JSON.stringify(result));
            this.dataList = result.map(a => {
                return {
                    ...a,
                    userName: a.User__r != undefined ? a.User__r.Name : '',
                    //isEditdisabled: a.Status__c != 'Pending',
```

```
                statuscss: a.Status__c == 'Approved' ? 'slds-theme_success' :
                    a.Status__c == 'Rejected' ? 'slds-theme_error' :
                        'slds-theme_warning'
            }
        });
    })
    .catch((error) => {
        this.dispatchEvent(
            new ShowToastEvent({
                title: 'Error loading leaves',
                message: error.body ? error.body.message : error.message,
                variant: 'error'
            })
        );
    });
}
handleRowAction(event) {
    this.isEdit = true;
    this.recordId = event.detail.row.Id;
}
handleClose() {
    this.isEdit = false;
}
handleSave(event) {
    const eve = new ShowToastEvent({
        title: 'Success',
        message: 'Record Created/Updated Successfully : Id = ' + event.detail.id,
        variant: 'success'
    });
    this.dispatchEvent(eve);
    this.isEdit = false;
    window.location.reload();
}
}
```

This JavaScript class defines an LWC named LeaveManagementManager for the manager interface in your Leave Management Salesforce project. Here's a detailed explanation of its functionality:

LeaveManagementManager LWC JavaScript Controller

- Tracked Properties:
  - recordId: Tracks the ID of the selected leave record for editing.
  - dataList: Holds the fetched leave records to display in the data table.
  - columnsList: Defines the data table columns:
    - User name extracted from related User record.
    - Leave details: Type, From Date, To Date, Reason, Status, Number of Days.
    - Status includes conditional CSS styling for visual feedback: green for Approved, red for Rejected, yellow for Warning.
    - Edit button for each row to allow record editing.
- Lifecycle Hook:
  - connectedCallback(): Calls handleLoad() immediately after the component is inserted in DOM.
- Main Methods:
  - handleLoad():

    Calls Apex method getLeaves() to fetch leave requests for employees reporting to the logged-in manager.

    Maps results to add user-friendly fields and status styling.

    Handles errors by showing toast notifications.
  - handleRowAction(event):

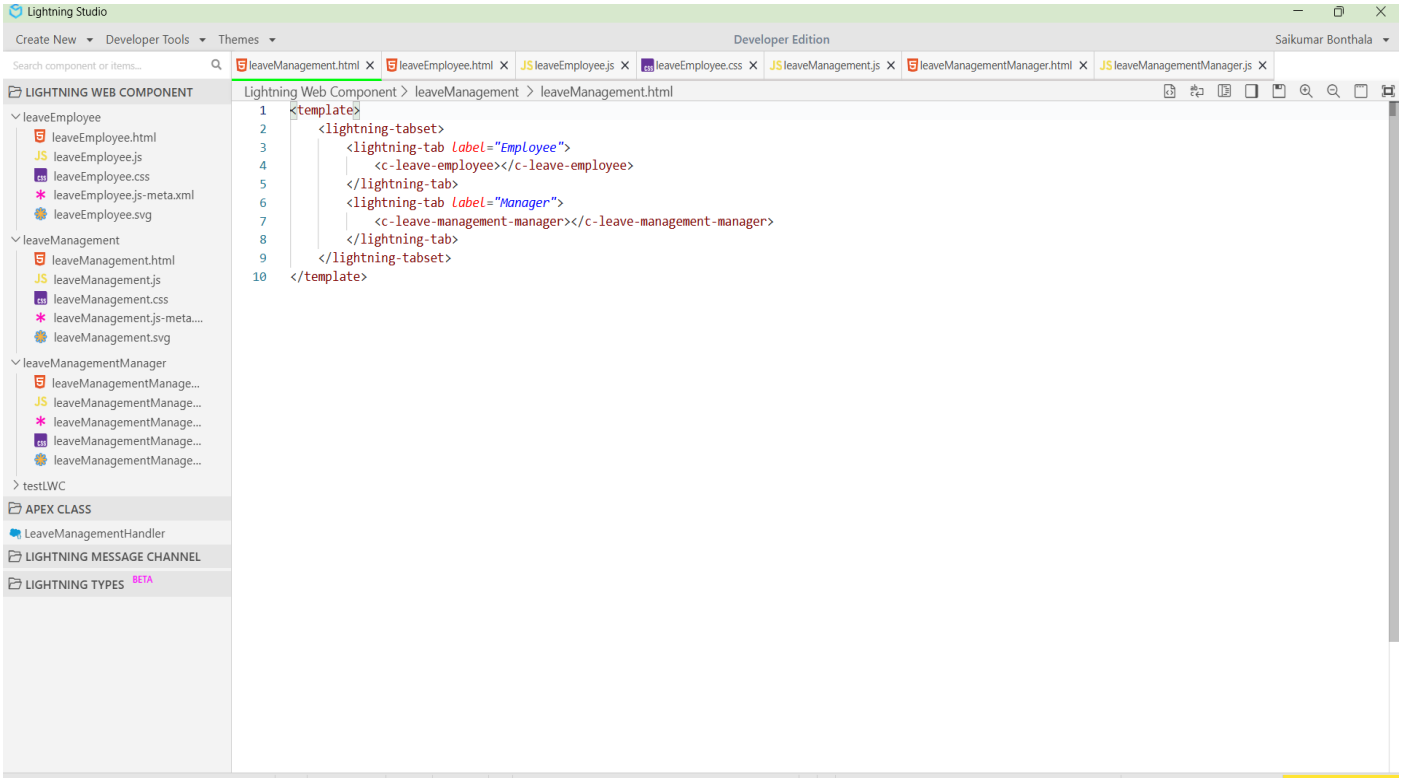
    Triggered when a data table row button is clicked.

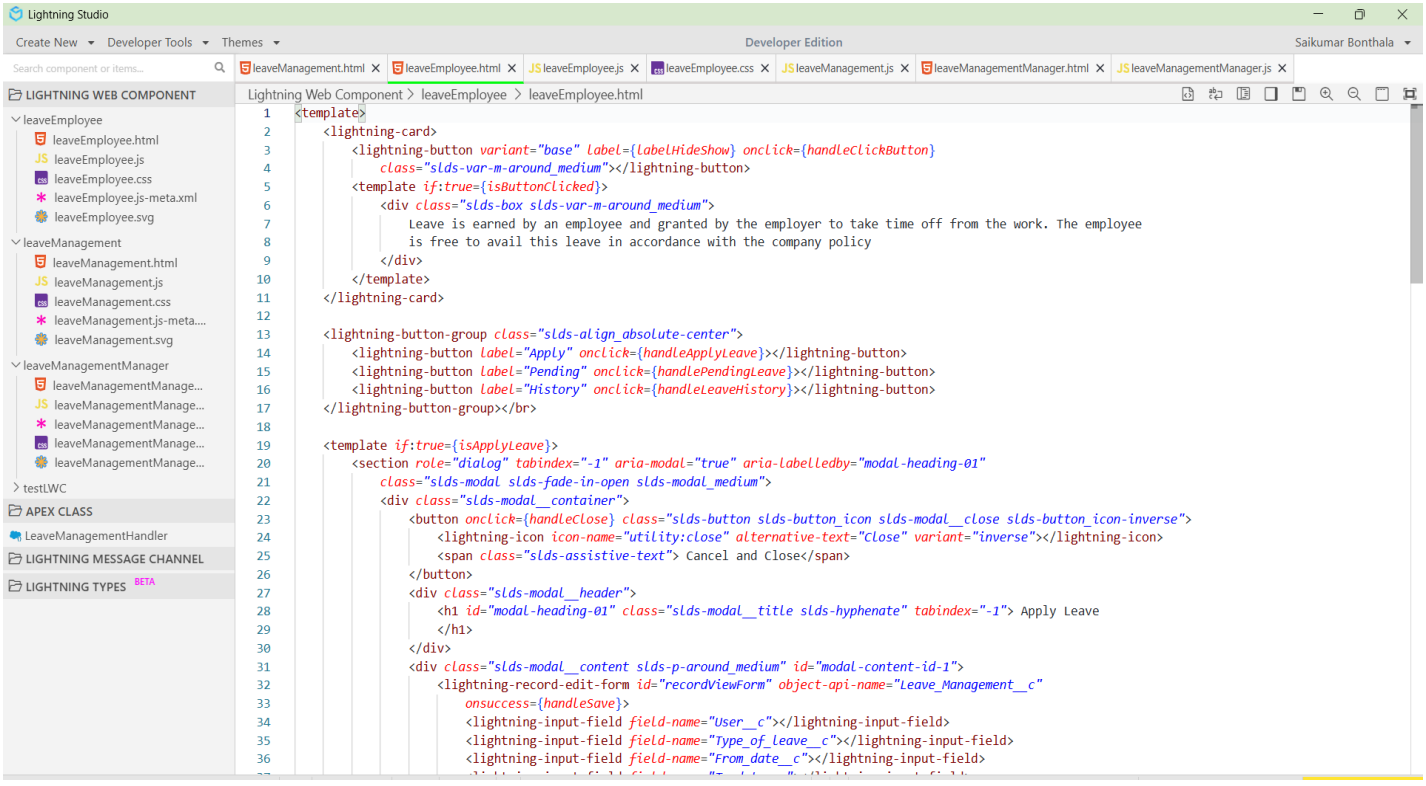
    Sets recordId and toggles edit modal display.
  - handleClose():

    Closes the edit modal.
  - handleSave(event):

    Fires a success toast on record creation or updates.

    Closes the edit modal and reloads the page to refresh data.


Lightning Studio
Create New
Developer Tools
Themes
Developer Edition
Saikumar Bonthala
leaveManagement.html
leaveEmployee.html
leaveEmployee.js
leaveEmployee.css
leaveManagement.js
leaveManagementManager.html
leaveManagementManager.js
LIGHTNING WEB COMPONENT
leaveEmployee
leaveEmployee.html
leaveEmployee.js
leaveEmployee.css
leaveEmployee.js-meta.xml
leaveEmployee.svg
leaveManagement
leaveManagement.html
leaveManagement.js
leaveManagement.css
leaveManagement.js-meta....
leaveManagement.svg
leaveManagementManager
leaveManagementManage...
leaveManagementManage...
leaveManagementManage...
leaveManagementManage...
leaveManagementManage...
testLWC
APEX CLASS
LeaveManagementHandler
LIGHTNING MESSAGE CHANNEL
LIGHTNING TYPES
BETA
Lightning Web Component > leaveManagement > leaveManagement.html
<template>
<lightning-tabset>
<lightning-tab label="Employee">
<c-leave-employee></c-leave-employee>
</lightning-tab>
<lightning-tab label="Manager">
<c-leave-management-manager></c-leave-management-manager>
</lightning-tab>
</lightning-tabset>
</template>


Lightning Studio
Create New
Developer Tools
Themes
Developer Edition
Saikumar Bonthala
leaveManagement.html
leaveEmployee.html
leaveEmployee.js
leaveEmployee.css
leaveManagement.js
leaveManagementManager.html
leaveManagementManager.js
LIGHTNING WEB COMPONENT
leaveEmployee
leaveEmployee.html
leaveEmployee.js
leaveEmployee.css
leaveEmployee.js-meta.xml
leaveEmployee.svg
leaveManagement
leaveManagement.html
leaveManagement.js
leaveManagement.css
leaveManagement.js-meta....
leaveManagement.svg
leaveManagementManager
leaveManagementManage...
leaveManagementManage...
leaveManagementManage...
leaveManagementManage...
leaveManagementManage...
testLWC
APEX CLASS
LeaveManagementHandler
LIGHTNING MESSAGE CHANNEL
LIGHTNING TYPES
BETA
Lightning Web Component > leaveEmployee > leaveEmployee.html
<template>
<lightning-card>
<lightning-button variant="base" label={labelHideShow} onclick={handleClickButton}
class="slds-var-m-around_medium"></lightning-button>
<template if:true={isButtonClicked}>

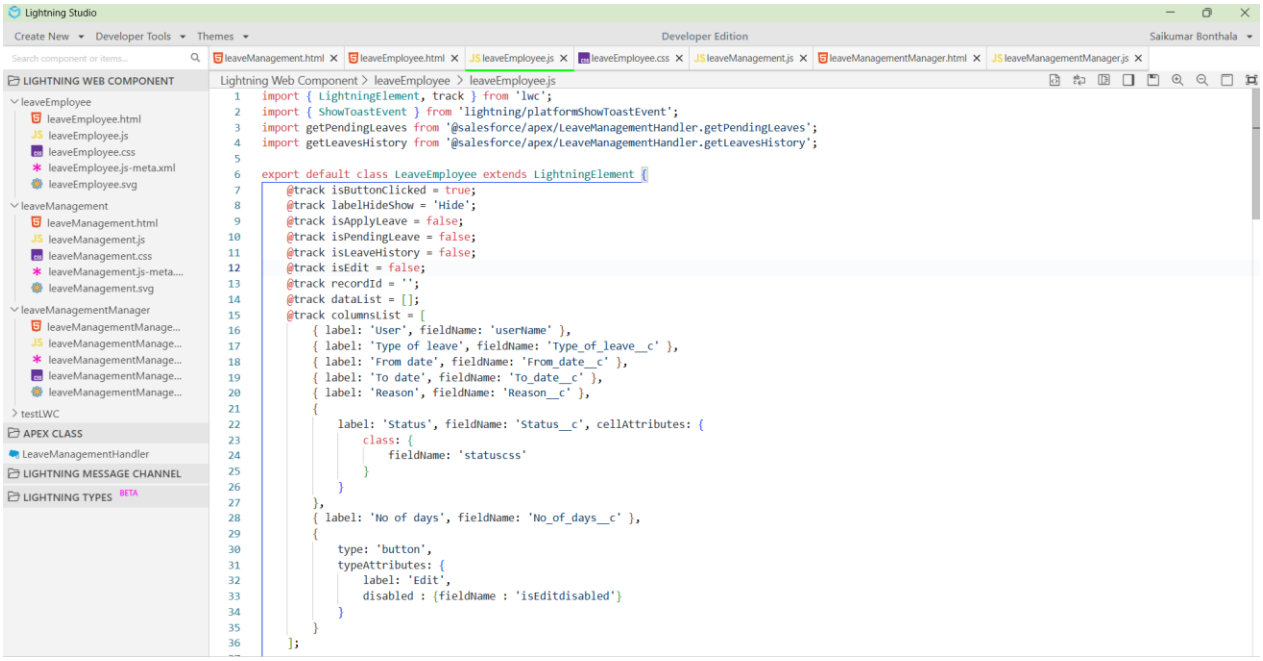
Leave is earned by an employee and granted by the employer to take time off from the work. The employee
is free to avail this leave in accordance with the company policy

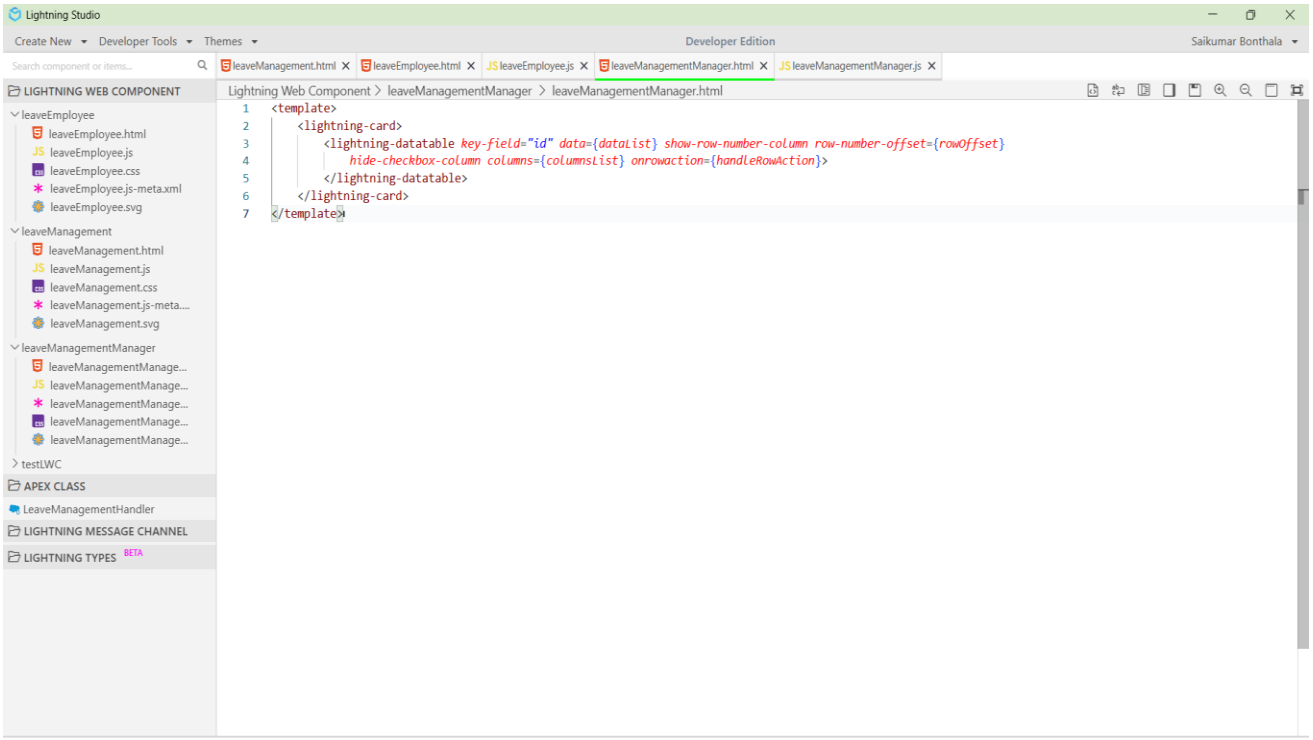
</template>
</lightning-card>
<lightning-button-group class="slds-align_absolute-center">
<lightning-button label="Apply" onclick={handleApplyLeave}></lightning-button>
<lightning-button label="Pending" onclick={handlePendingLeave}></lightning-button>
<lightning-button label="History" onclick={handleLeaveHistory}></lightning-button>
</lightning-button-group></br>
<template if:true={isApplyLeave}>
<section role="dialog" tabindex="-1" aria-modal="true" aria-labelledby="modal-heading-01"
class="slds-modal slds-fade-in-open slds-modal_medium">

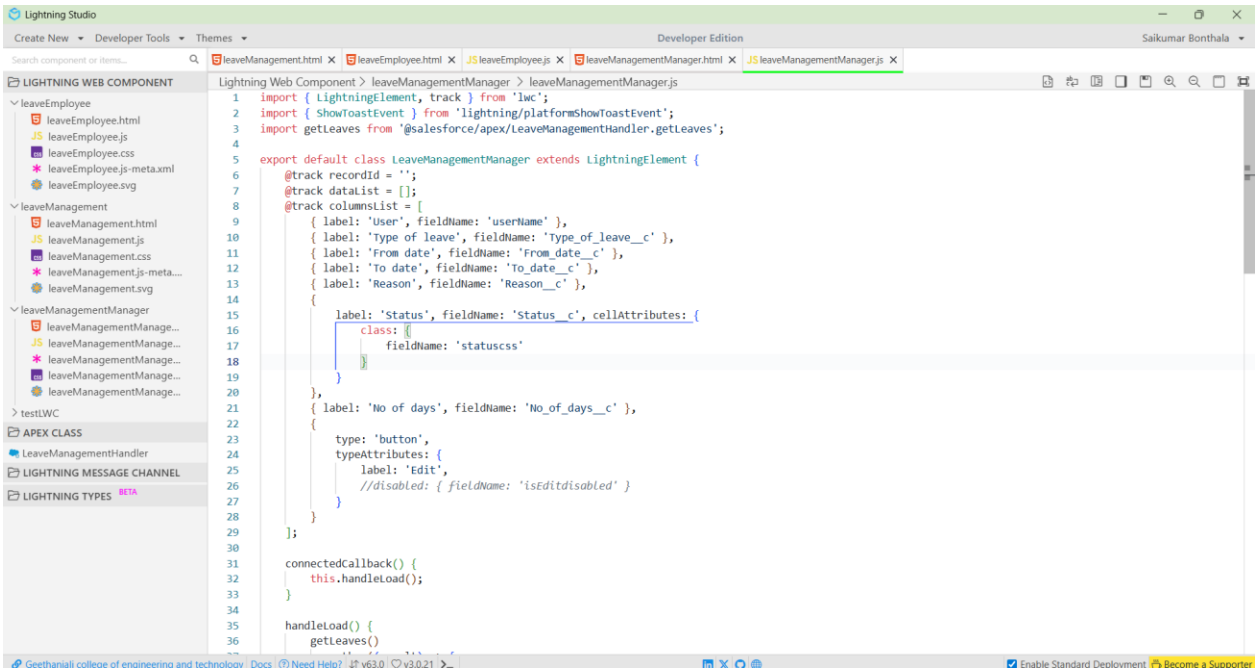
<button onclick={handleClose} class="slds-button slds-button_icon slds-modal__close slds-button_icon-inverse">
<lightning-icon icon-name="utility:close" alternative-text="Close" variant="inverse"></lightning-icon>
 Cancel and Close
</button>

<h1 id="modal-heading-01" class="slds-modal__title slds-hyphenate" tabindex="-1"> Apply Leave
</h1>

<lightning-record-edit-form id="recordViewForm" object-api-name="Leave_Management__c"
onsuccess={handleSave}>
<lightning-input-field field-name="User__c"></lightning-input-field>
<lightning-input-field field-name="Type_of_leave__c"></lightning-input-field>
<lightning-input-field field-name="From_date__c"></lightning-input-field>


Lightning Studio
Create New
Developer Tools
Themes
Developer Edition
Saikumar Bonthala
leaveManagement.html
leaveEmployee.html
leaveEmployee.js
leaveEmployee.css
leaveManagement.js
leaveManagementManager.html
leaveManagementManager.js
LIGHTNING WEB COMPONENT
leaveEmployee
leaveEmployee.html
leaveEmployee.js
leaveEmployee.css
leaveEmployee.js-meta.xml
leaveEmployee.svg
leaveManagement
leaveManagement.html
leaveManagement.js
leaveManagement.css
leaveManagement.svg
leaveManagementManager
testLWC
APEX CLASS
LeaveManagementHandler
LIGHTNING MESSAGE CHANNEL
LIGHTNING TYPES
BETA
Lightning Web Component > leaveEmployee > leaveEmployee.js
import { LightningElement, track } from 'lwc';
import { ShowToastEvent } from 'lightning/platformShowToastEvent';
import getPendingLeaves from '@salesforce/apex/LeaveManagementHandler.getPendingLeaves';
import getLeavesHistory from '@salesforce/apex/LeaveManagementHandler.getLeavesHistory';
export default class LeaveEmployee extends LightningElement {
@track isButtonClicked = true;
@track labelHideShow = 'Hide';
@track isApplyLeave = false;
@track isPendingLeave = false;
@track isLeaveHistory = false;
@track isEdit = false;
@track recordId = '';
@track dataList = [];
@track columnsList = [
{ label: 'User', fieldName: 'userName' },
{ label: 'Type of leave', fieldName: 'Type_of_leave__c' },
{ label: 'From date', fieldName: 'From_date__c' },
{ label: 'To date', fieldName: 'To_date__c' },
{ label: 'Reason', fieldName: 'Reason__c' },
{
label: 'Status', fieldName: 'Status__c', cellAttributes: {
class: {
fieldName: 'statuscss'
}
}
},
{ label: 'No of days', fieldName: 'No_of_days__c' },
{
type: 'button',
typeAttributes: {
label: 'Edit',
disabled : {fieldName : 'isEditdisabled'}
}
}
];


Lightning Studio
Create New
Developer Tools
Themes
Developer Edition
Saikumar Bonthala
leaveManagement.html
leaveEmployee.html
leaveEmployee.js
leaveManagementManager.html
leaveManagementManager.js
LIGHTNING WEB COMPONENT
leaveEmployee
leaveEmployee.html
leaveEmployee.js
leaveEmployee.css
leaveEmployee.js-meta.xml
leaveEmployee.svg
leaveManagement
leaveManagement.html
leaveManagement.js
leaveManagement.css
leaveManagement.svg
leaveManagementManager
testLWC
APEX CLASS
LeaveManagementHandler
LIGHTNING MESSAGE CHANNEL
LIGHTNING TYPES
BETA
Lightning Web Component > leaveManagementManager > leaveManagementManager.html
<template>
<lightning-card>
<lightning-datatable key-field="id" data={dataList} show-row-number-column row-number-offset={rowOffset}
hide-checkbox-column columns={columnsList} onrowaction={handleRowAction}>
</lightning-datatable>
</lightning-card>
</template>



Phase 7: Integration & External Access

The goal of this phase is to extend the Leave Management system by integrating with external systems and tools to enhance automation, validation, and user productivity beyond native Salesforce capabilities.

7.1 Implemented Feature: Custom Link for External Leave Policy Lookup

- Component: Custom Detail Page Button — Leave Policy Web Search
- Purpose:

  Enables employees and managers to quickly access additional external resources and information about leave policies and regulations without leaving Salesforce.
- Implementation Details:
  - Object: Leave_Management__c
  - Display Type: Detail Page Button on Leave Management record pages.
  - Behavior: Opens an external link in a new browser tab or window.
  - Content Source: URL with dynamic merge field.
  - URL Formula:

    "https://www.google.com/search?q=" & TEXT(Leave_Management__c.Type_of_leave__c) & "+leave policy"

    - This URL dynamically includes the current leave type in the web search query, making it context-aware and useful for quick policy reference.
- User Experience:
  With one click, users can validate or enrich their understanding of leave policies related to their specific leave requests, improving decision making and compliance.

7.2 Future Enhancements

To further mature into an enterprise-grade leave management platform, future integration efforts can focus on:

- HR System API Integration:
  Seamless syncing of leave requests, approvals, and balances with external HR payroll or workforce management systems.
- Employee Calendar Synchronization:
  Real-time update of leave approvals directly reflected in employees' and managers' digital calendars (e.g., Outlook, Google Calendar).
- Notification Services:
  Integration with external communication platforms (e.g., Slack, Microsoft Teams) for automated leave status alerts and reminders.
- Compliance Automation:
  External compliance and audit tool callouts to validate policy adherence and produce audit reports.

Phase 8 – Data Management & Deployment

Objective

This phase focuses on ensuring robust data handling and systematically deploying the Leave Management solution to production or target environments to meet organizational needs.

8.1 Data Management

- **Data Migration:**
  - Preparation of any initial leave data imports using Salesforce Data Loader or Data Import Wizard.
  - Validation of migrated data ensuring consistency of key fields such as Type of Leave, Dates, Status, and User associations.
- **Data Quality & Maintenance:**
  - Implement duplicate detection rules to prevent multiple overlapping leave requests for the same period.
  - Periodic data clean-up strategies to archive or delete obsolete leave records.
- **Data Security:**
  - Apply profile and permission set based access controls to ensure that employees only see their own leaves, and managers see their team's leaves.
  - Field-level security configurations for sensitive data fields like Reason or Status.

8.2 Deployment Strategy

- **Metadata Deployment:**
  - Use change sets or Salesforce DX to deploy custom objects, fields, Apex classes, Lightning Web Components, and other metadata.
  - Validate deployments in a Partial Copy or Full sandbox before production.
- **User Acceptance Testing (UAT):**
  - Engage end users in sandbox or UAT environments to verify business processes and UI workflows.
  - Collect feedback and resolve issues before final deployment.
- **Production Rollout:**
  - Schedule deployment during low-business hours.
  - Communicate rollout plans and training guides to users.
  - Monitor post-deployment for issues, data integrity, and user adoption.

Phase 9: Reporting, Dashboards & Security Review

Objective

The objective of this phase is twofold:

1. Transform raw Leave Management data into actionable insights through reports and dashboards.
2. Review and validate the application's security settings to ensure leave data (requests, approvals, and statuses) is accessible only to authorized users.

9.1 Reports

Reports were created in Salesforce to analyze leave data, apply filters, group results, and provide summaries useful to employees, managers, and HR administrators.

- **Report 1: Leaves by Type and Status**
  - Purpose: Provide an overview of leave requests by type (Sick, Casual, etc.) and their current status (Pending, Approved, Rejected).
  - Report Type: Summary Report based on the Leave Management custom object.
  - Configuration: Group by Type of Leave and Status fields; summary of record counts.
  - Visualization: Bar chart to display counts of different leave types and approval statuses.
- **Report 2: Leave Utilization by User**
  - Purpose: Measure the total days of leave used by each employee.
  - Report Type: Summary Report using Leave Management records.
  - Configuration: Group by User, aggregate the sum of Number of Days.
  - Visualization: Number widget or chart to show leave utilization per employee.
- **Report 3: Leave Approval Status Overview**
  - Purpose: Track how many leave requests are pending, approved, or rejected within a time period.
  - Report Type: Matrix Report on Leave Management object.
  - Configuration: Group rows by Status and columns by date ranges or users.
  - Visualization: Donut chart showing proportion of approval statuses.

9.2 Dashboard

- Leave Management Dashboard

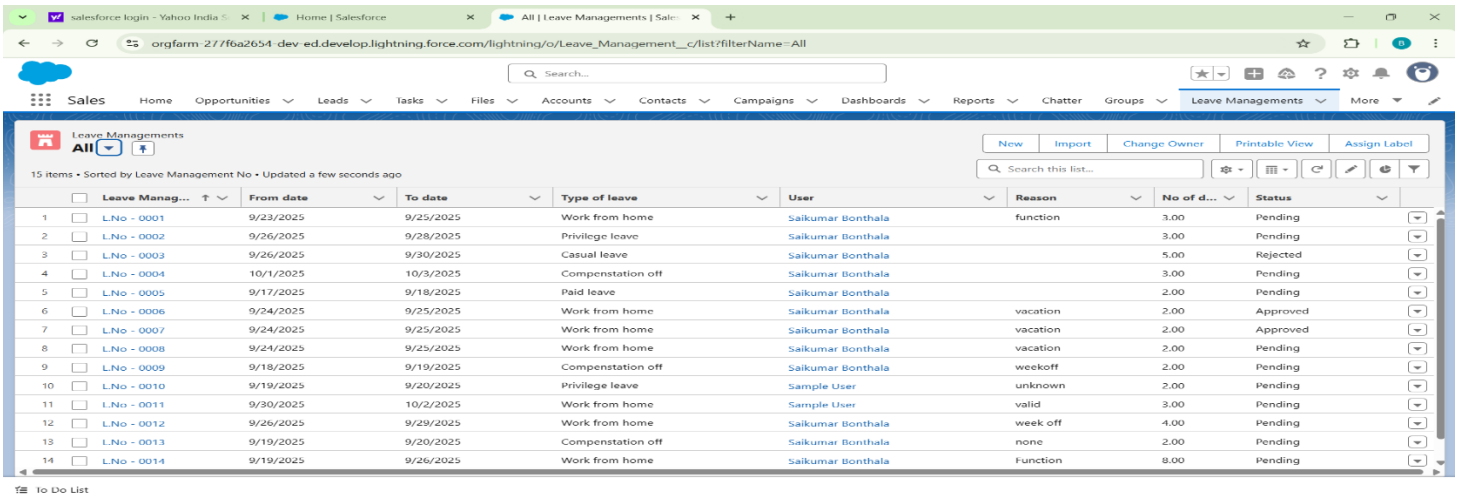
  Consolidates the vital leave management KPIs into a unified and visual dashboard:

  - Bar chart for Leaves by Type and Status.
  - Number widget for Leave Utilization by User.
  - Donut chart for Leave Approval Status Overview.

    This dashboard empowers HR, managers, and employees to instantly assess leave trends and operational bottlenecks.

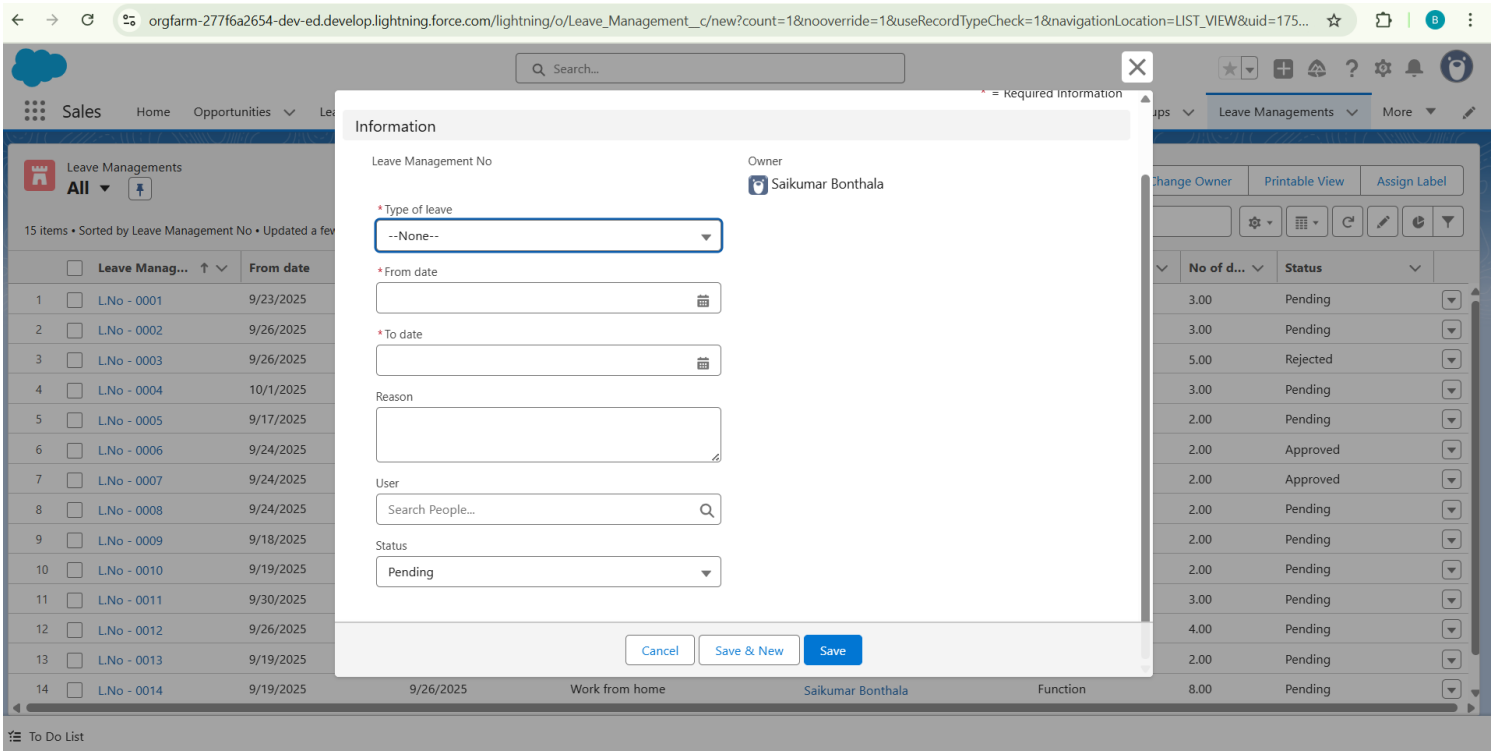
9.3 Security Review

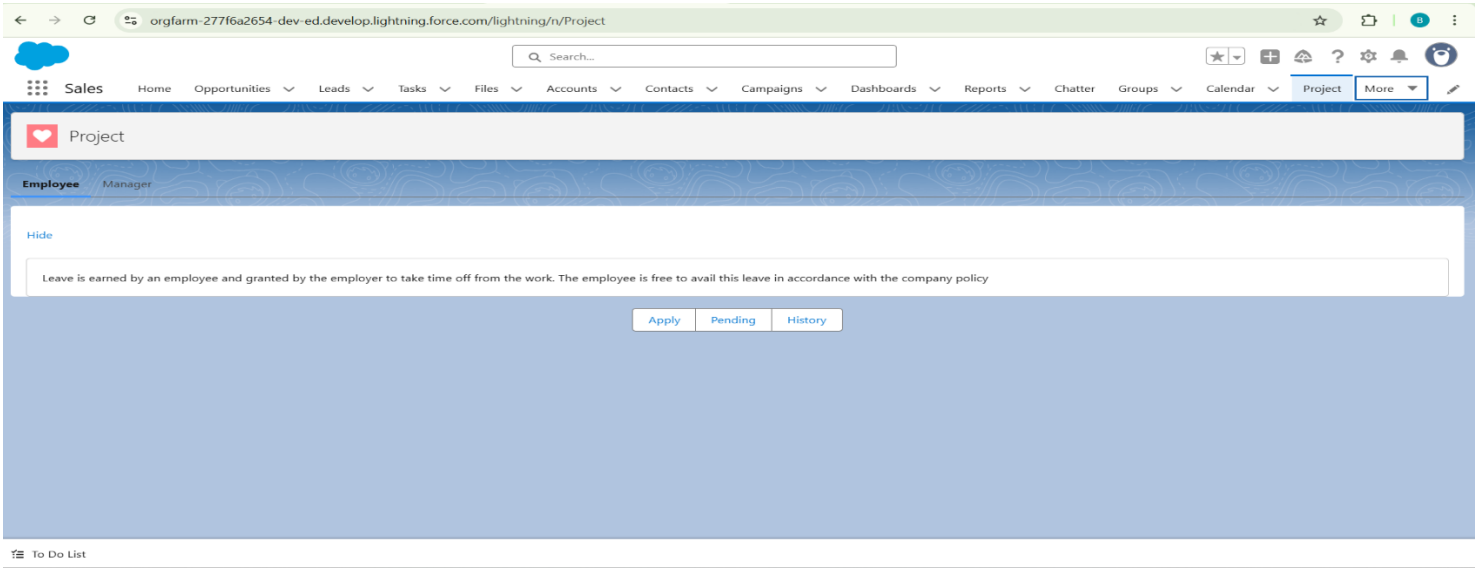
- Sharing Settings:
  - Org-wide defaults set to Private for the Leave Management object.
  - Sharing rules implemented to allow employees access to their own leave records and managers access to their team's records.
- Field-Level Security (FLS):
  - Sensitive fields such as Reason and Status are restricted to authorized profiles.
  - Standard employees typically have read/write access only to their leave requests.
- Session & Login Policies:
  - Standard Salesforce session timeout and password complexity policies enforced.
  - External IP restrictions recommended for production environments.
- Audit Trail:
  - Salesforce Audit Trail reviewed to monitor configuration changes and maintain compliance



This screen lists all Leave Management records, showing the org-wide leave database for administrators or managers.

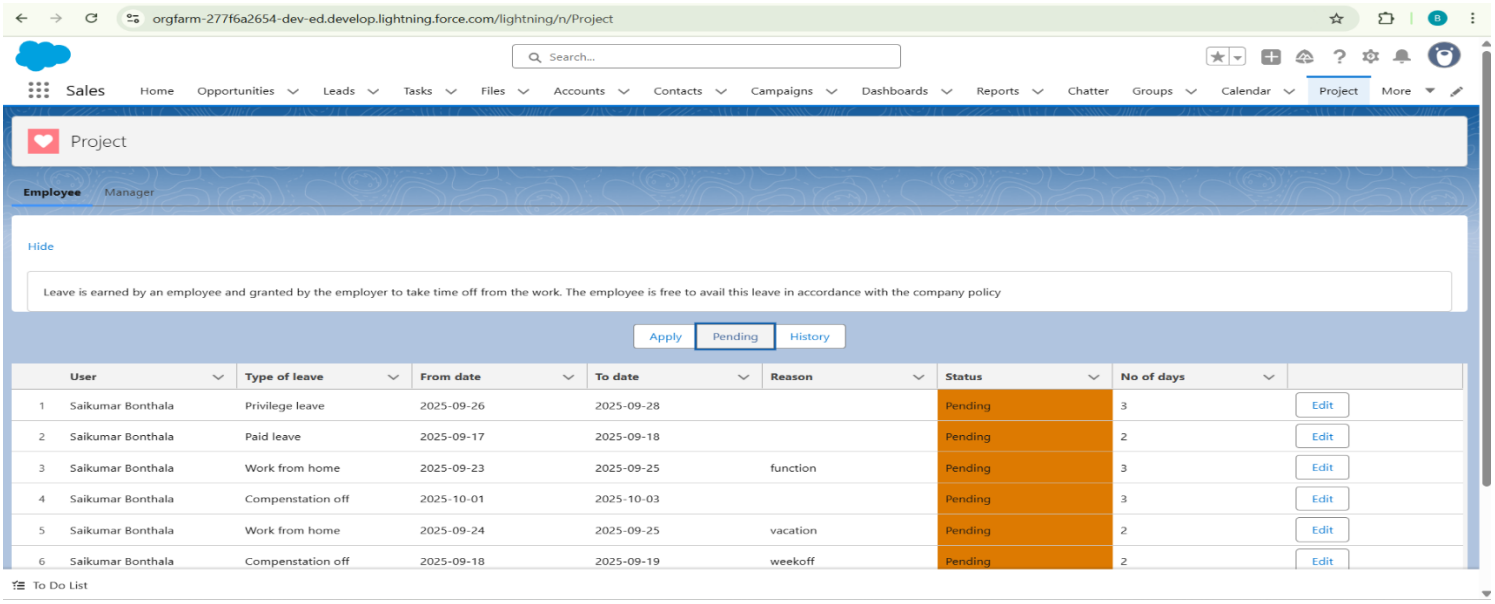
- Columns include Leave Management No, From date, To date, Type of leave, User, Reason, Number of days, and Status.
- The view is sorted and filterable, giving a complete overview of leave activity across the organization for effective tracking and reporting.





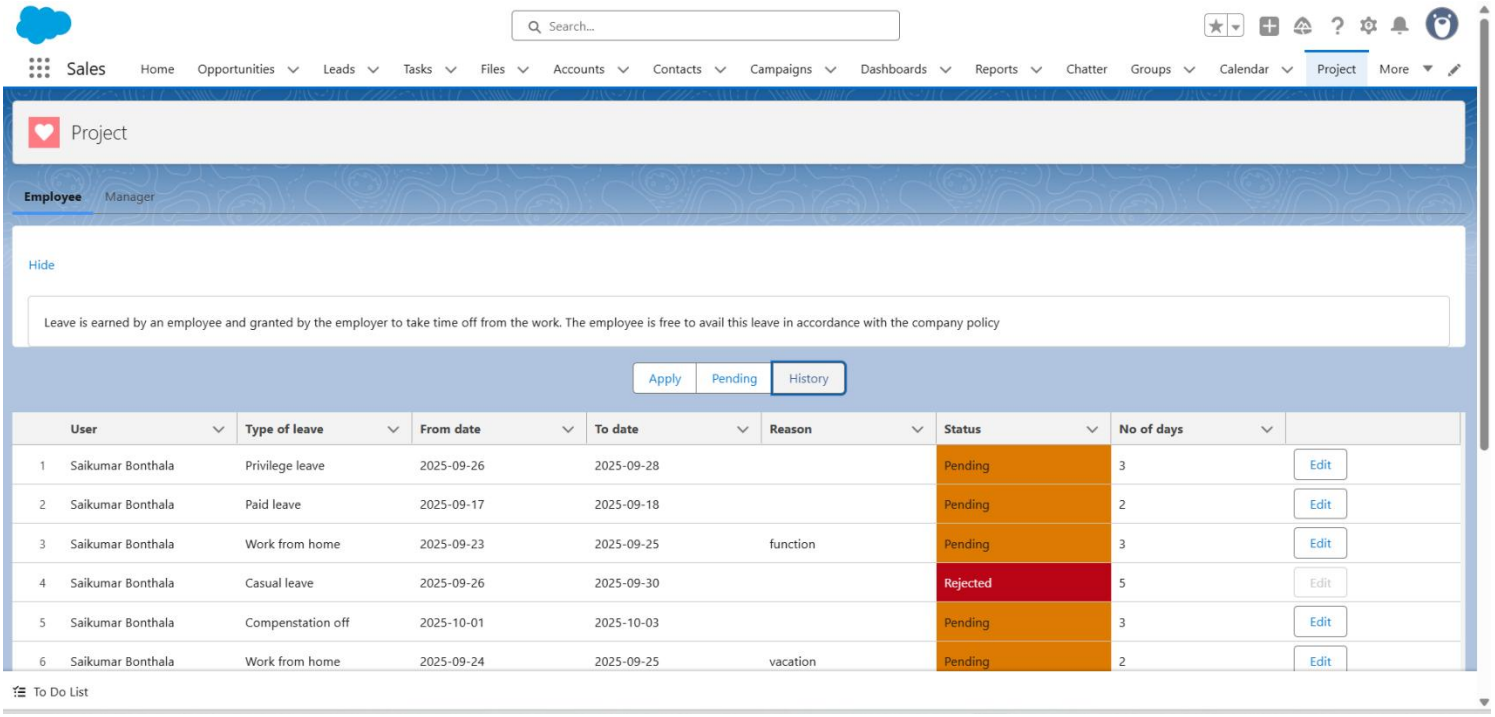
This image shows "Employee" tab in the Leave Management app interface, where employees interact with their leave dashboard.

- An informational box (“Leave is earned by an employee…”) can be toggled to help users understand company policy.
- Action buttons for "Apply," "Pending," and "History" enable users to submit a new request, view pending requests, or review past leave records.
- The UI, built with SLDS styling, ensures a clear and accessible user experience.



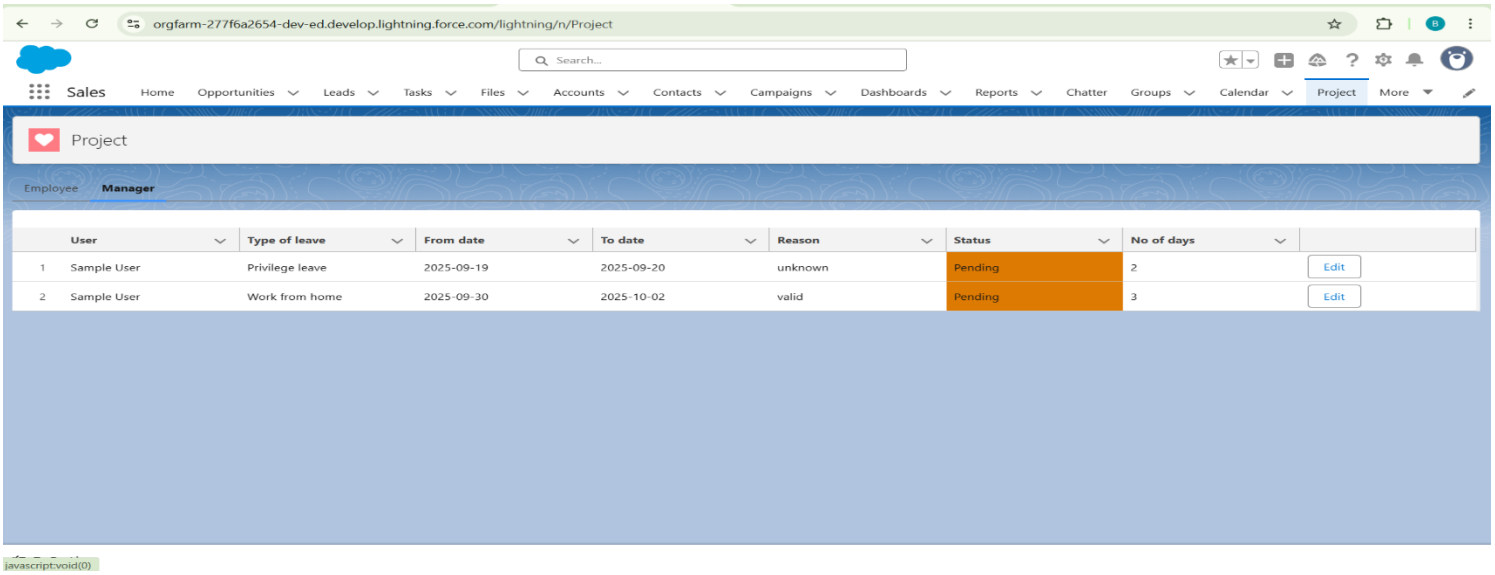
In this image, the "Employee" tab displays the pending leave requests for the current user.

- The table lists each leave request, showing who requested it, type, dates, reason, approval status, and duration.
- Status coloring (orange for pending) visually differentiates requests awaiting manager action.
- Edit buttons allow users to update details for requests that are still in a "Pending" state.



This screenshot shows the "Employee" tab with the "History" view active.

- Users see all of their past leave requests, with status indicators (orange for pending, red for rejected) providing immediate feedback.
- The table design supports easy history review and potential corrections or audits of past requests.
- Edit options are disabled for rejected or non-pending requests to protect workflow integrity.



This "Manager" tab of the Project page, likely rendered using a Lightning Web Component in a tabset.

- The table presents leave requests submitted by employees reporting to the manager.
- Key columns include User, Type of Leave, Dates, Reason, Status (with colored backgrounds indicating state: orange for pending), and the number of days requested.
- Edit buttons allow managers to update or approve leave requests, enforcing workflow management from the managerial interface.

CONCLUSION:

The Leave Management system implemented using Salesforce Lightning Web Components and Apex has successfully streamlined the employee leave request and approval process within the organization. By leveraging Salesforce's cloud platform capabilities, the project provided a centralized and responsive solution for managing leave types, tracking leave durations, and capturing reasons and statuses in real time.

Key achievements include:

- Intuitive user interfaces tailored for both employees and managers, facilitating leave applications, status tracking, and approvals.
- Robust backend logic using Apex to filter and deliver role-based leave data securely.
- Comprehensive data modeling with custom objects and fields matching business leave policies.
- Automation of workflows with validation rules and user notifications improving process efficiency.
- Integration of reporting and dashboards to offer actionable insights into leave trends and utilization.
- Strong security measures ensuring data confidentiality via sharing rules, permission sets, and field-level security.
- A foundation built for scalability, with considerations for external integrations and future enhancements.

Overall, this project demonstrates how Salesforce technology enables organizations to digitize and optimize leave management, reducing manual effort, minimizing errors, and enhancing user satisfaction. The modular, best-practice design offers extensibility to evolve with organizational needs, driving more effective HR operations and workforce management.