Project: Disaster Relief Resource Management CRM (ReliefConnect)

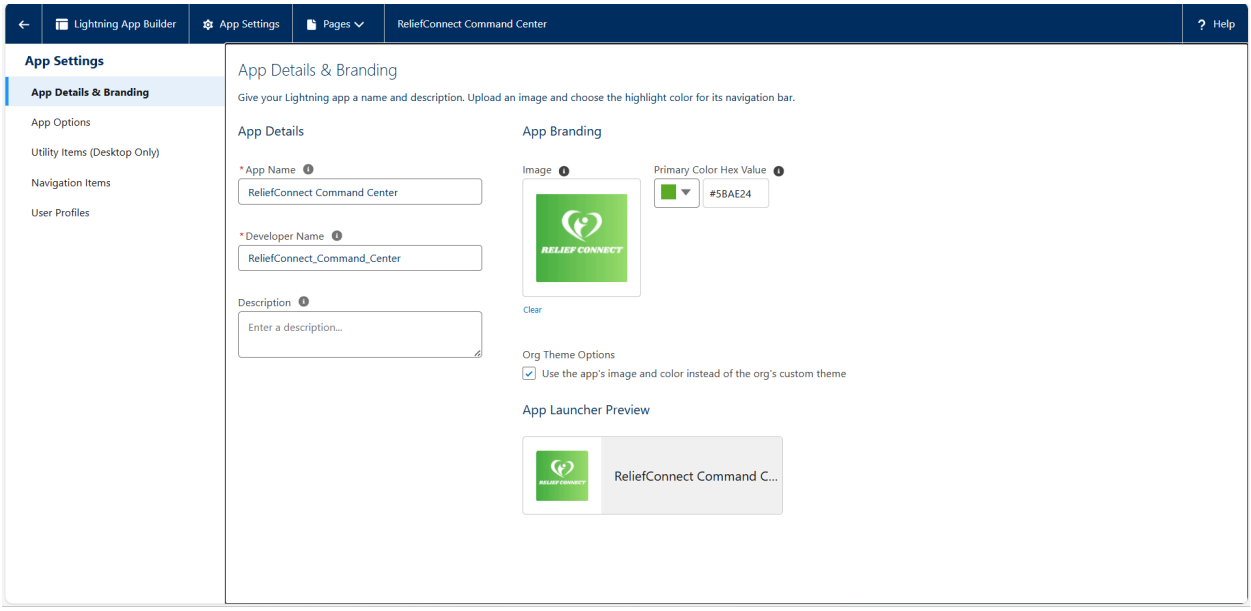
PHASE 6: User Interface Development

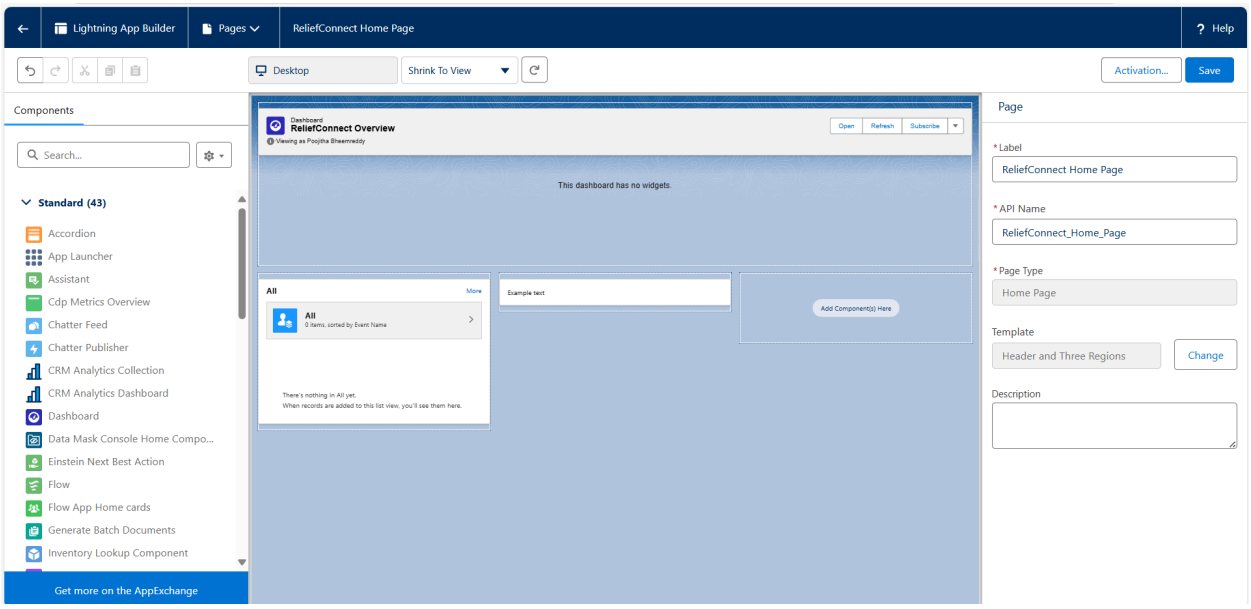
🎯 Executive Summary

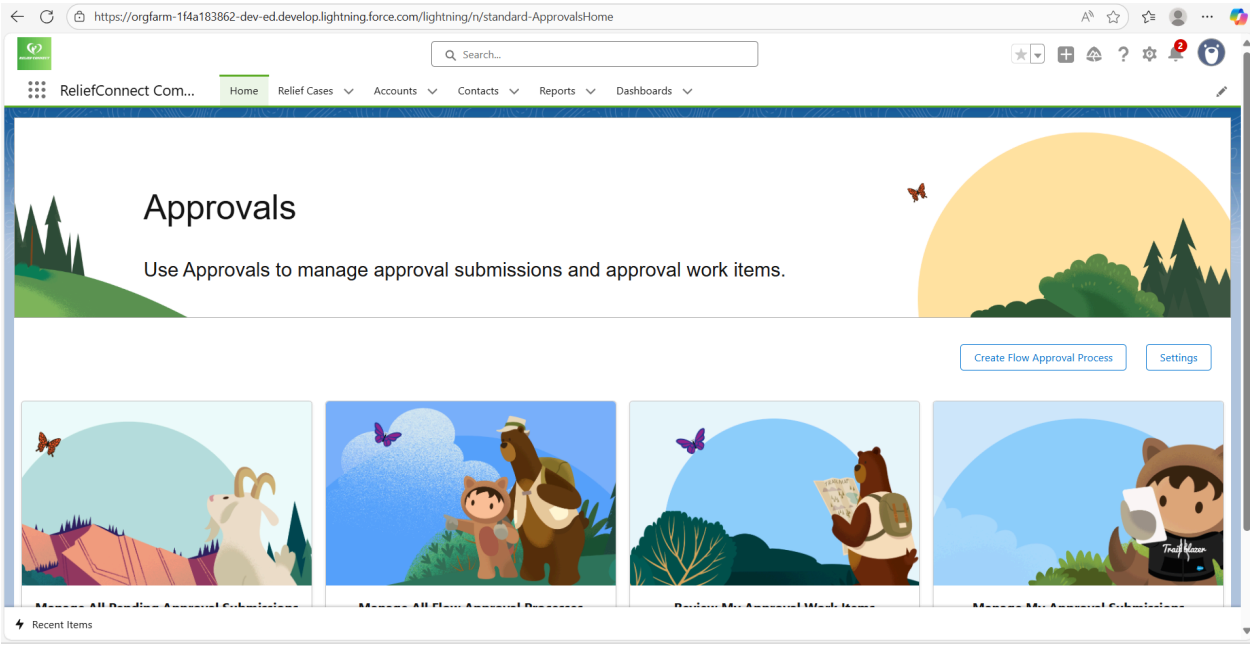
Phase 6 focused on translating the robust backend data model and automations into a clean, intuitive, and efficient user interface (UI). The primary goal was to build an experience that empowers all stakeholders, from field volunteers on mobile devices to coordinators in a command center to interact with the system effectively. This was achieved by creating a custom Lightning App, designing dynamic record and home pages to provide critical information at a glance, and developing a key Lightning Web Component (LWC) for mobile data entry that serves as the primary tool for on-the-ground data capture.

📱 Lightning App Builder

- **Purpose/Rationale:** A dedicated Lightning App was essential to create a focused and branded workspace for ReliefConnect users. This prevents distraction from standard Salesforce apps (like Sales or Service) and streamlines the user experience by only presenting the navigation and tools relevant to disaster response.
- **Detailed Implementation:** Using the Lightning App Builder, I constructed the **"ReliefConnect Command Center"** app. The app was branded with a custom logo representing a helping hand and a calming blue color scheme to maintain a professional look and feel. The navigation bar was explicitly configured to include the `Home` tab, `Relief Cases`, `Accounts` (renamed to "NGOs & Agencies"), `Contacts` (for volunteers), `Reports`, and `Dashboards`.

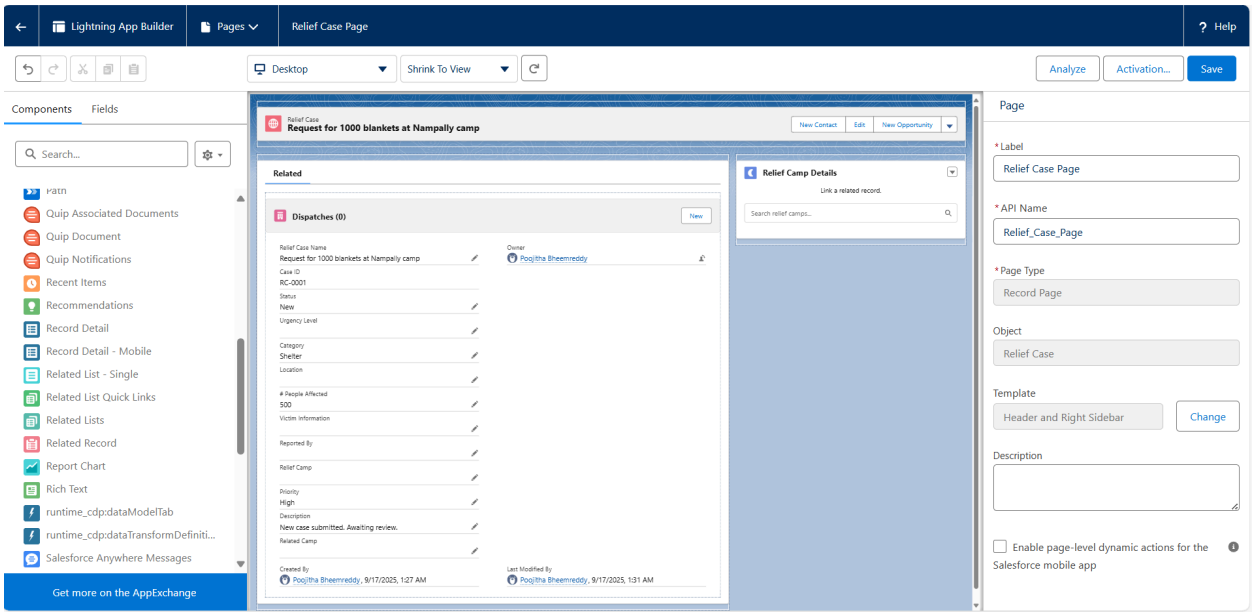


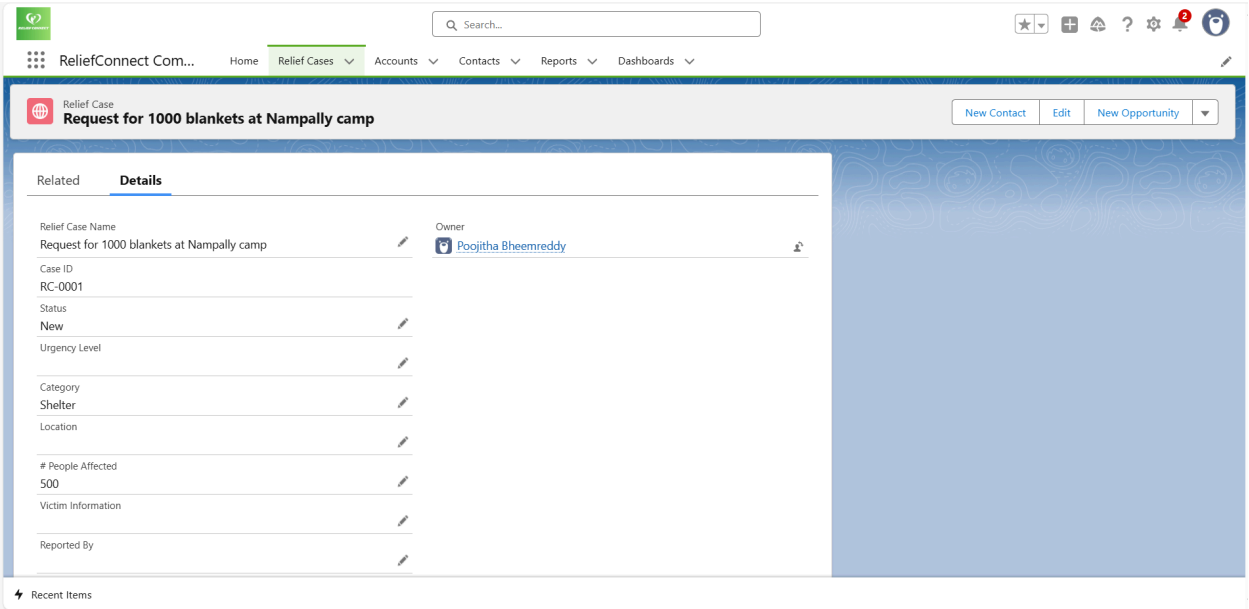

Lightning App Builder
Pages
ReliefConnect Home Page
Help
Desktop
Shrink To View
Activation...
Save
Components
Search...
Standard (43)
Accordion
App Launcher
Assistant
Cdp Metrics Overview
Chatter Feed
Chatter Publisher
CRM Analytics Collection
CRM Analytics Dashboard
Dashboard
Data Mask Console Home Compo...
Einstein Next Best Action
Flow
Flow App Home cards
Generate Batch Documents
Inventory Lookup Component
Get more on the AppExchange
Dashboard
ReliefConnect Overview
Open
Refresh
Subscribe
This dashboard has no widgets.
All
Example text
Add Component(s) Here
There's nothing in All yet.
When records are added to this list view, you'll see them here.
Page
* Label
ReliefConnect Home Page
* API Name
ReliefConnect_Home_Page
* Page Type
Home Page
Template
Header and Three Regions
Change
Description


https://orgfarm-1f4a183862-dev-ed.develop.lightning.force.com/lightning/n/standard-ApprovalsHome
Search...
ReliefConnect Com...
Home
Relief Cases
Accounts
Contacts
Reports
Dashboards
Approvals
Use Approvals to manage approval submissions and approval work items.
Create Flow Approval Process
Settings
Recent Items

📄 Record Pages

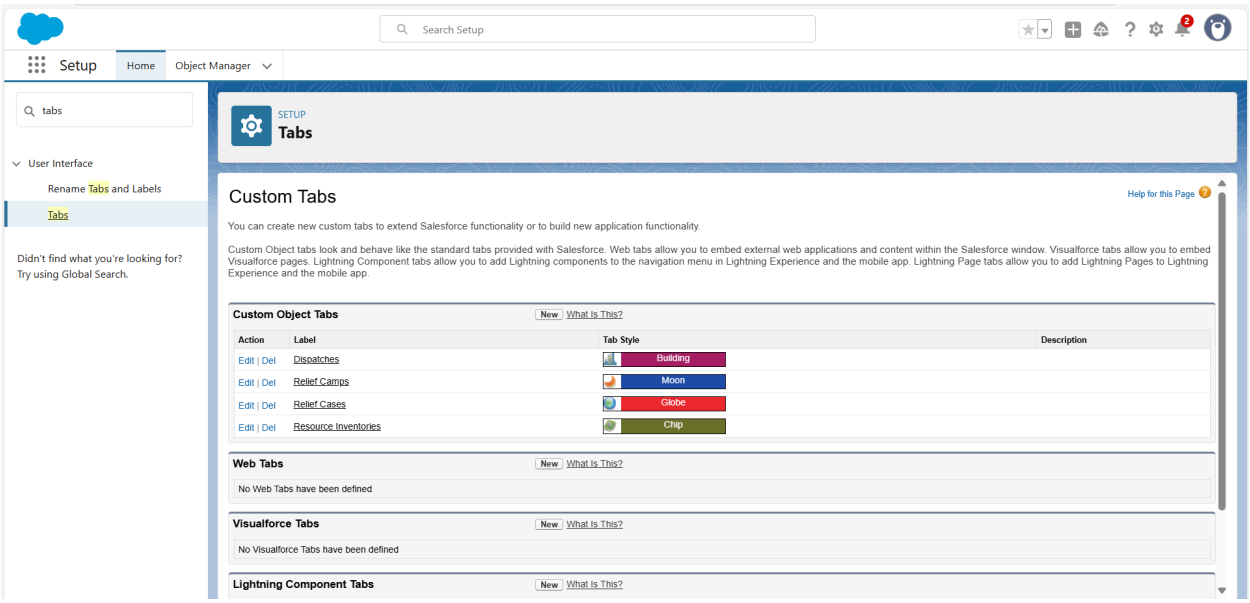
- **Purpose/Rationale:** The standard Salesforce record page is generic. A custom record page was designed for the `Relief_Case__c` object to present the most critical information in a prioritized and actionable format for a busy relief coordinator.
- **Detailed Implementation:** The `Relief Case` Lightning Record Page was built with a three-column layout.
    - The **Highlights Panel** at the top was configured to always show the Case Name, Status, Priority, and # of People Affected for immediate triage.
    - A **Tabs component** in the main region separates the core `Details` of the case from related lists like `Dispatch Records`, `Activity History`, and `Chatter`, reducing clutter.
    - A **Related Record component** was placed in the sidebar to display key fields from the parent `Relief Camp` record (like Camp Capacity and the Onsite Lead's contact info), eliminating the need for users to click away to find related information.

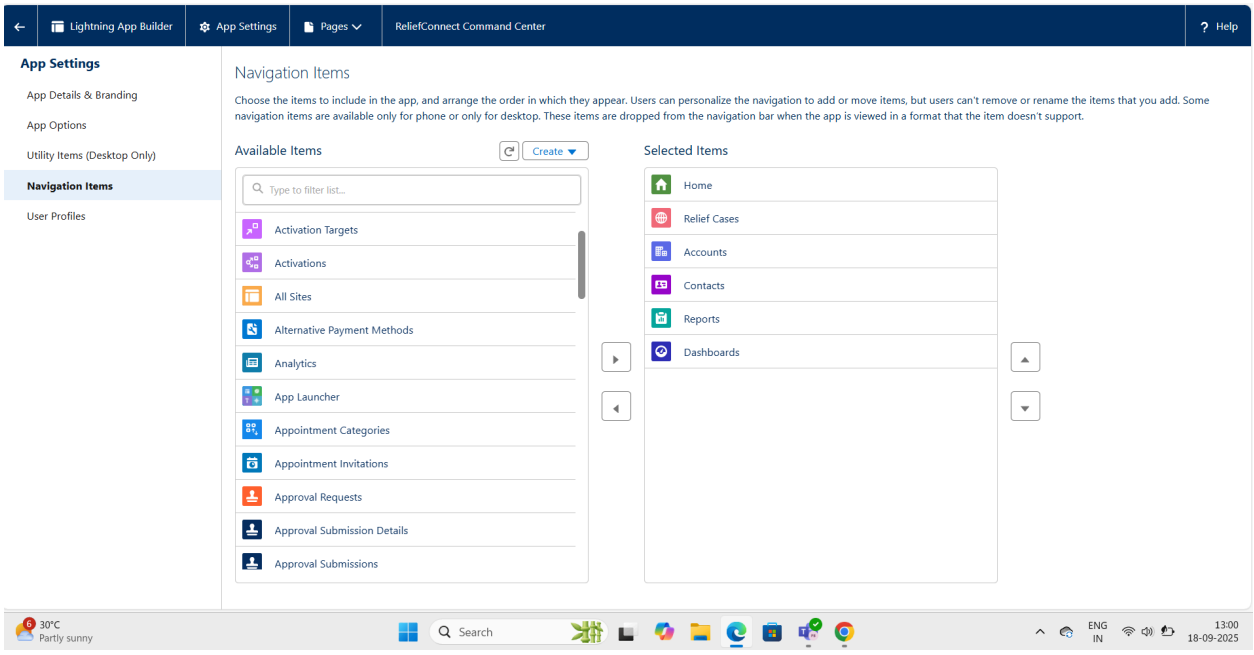




📑 Tabs

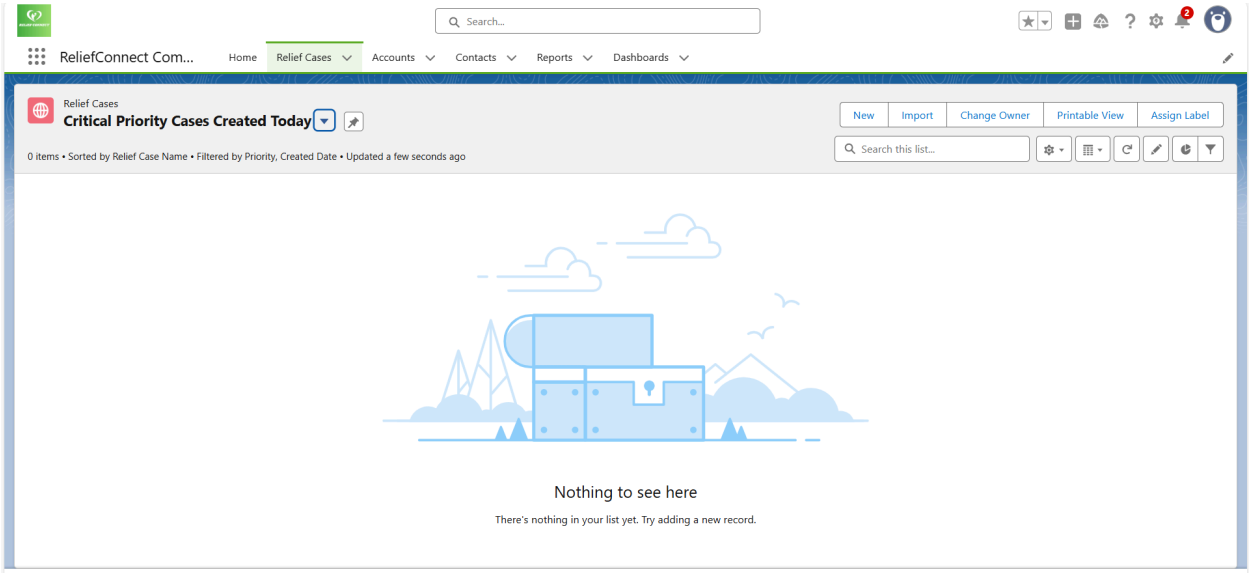
- **Purpose/Rationale:** Tabs are the primary navigation mechanism for users to access list views of records. Creating custom tabs for our custom objects is a fundamental step to make them accessible within the app.
- **Detailed Implementation:** Custom tabs were created for all primary custom objects: `Relief Case`, `Resource Inventory`, `Dispatch`, and `Relief Camp`. I configured the default list view for the `Relief Cases` tab to be "My Open Cases," ensuring that when a user clicks the tab, they are immediately presented with their most urgent workload.

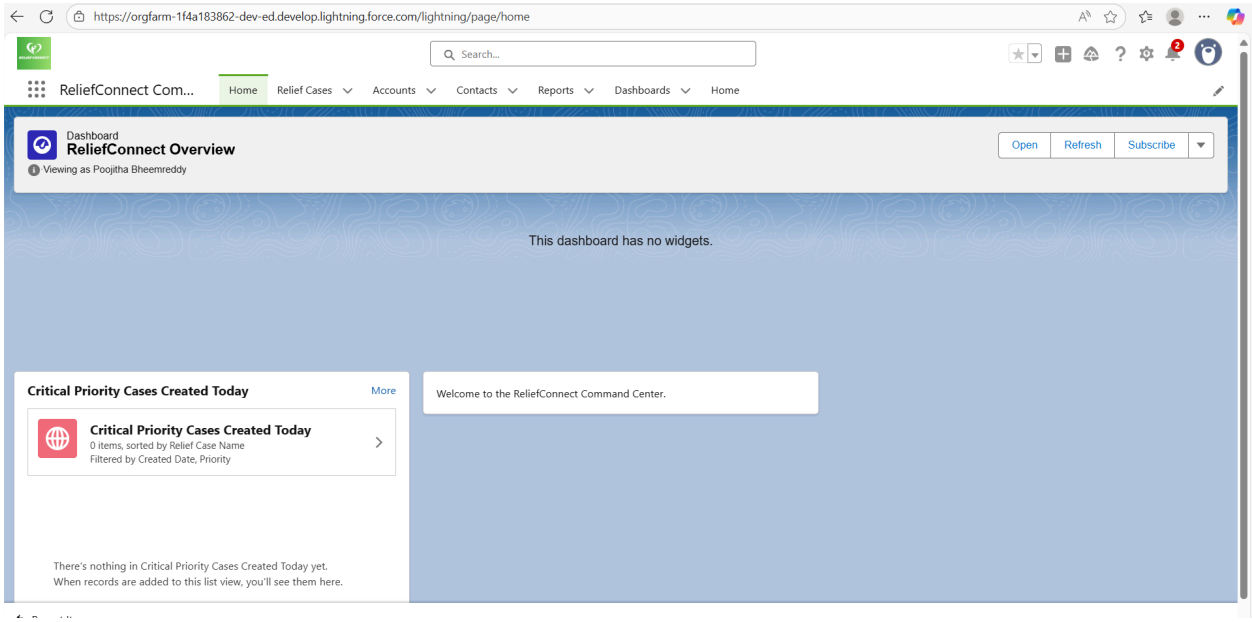




🏠 Home Page Layouts

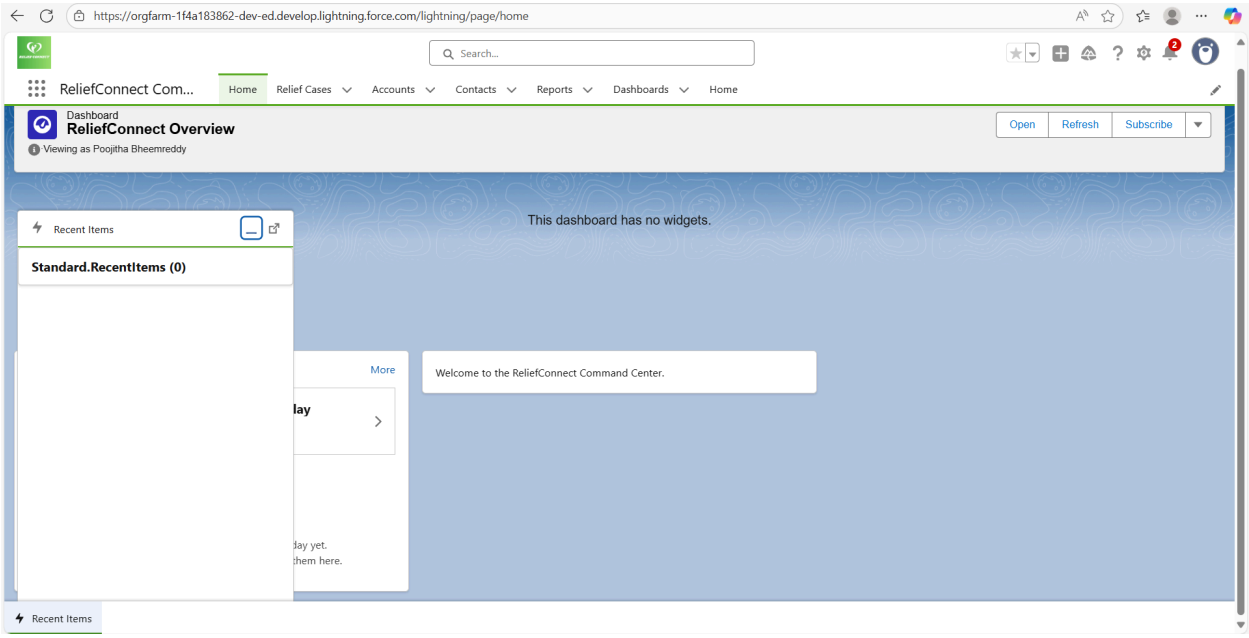
- **Purpose/Rationale:** The Home Page is the first screen users see upon logging in. It was designed to be an actionable command center, providing immediate situational awareness rather than a generic welcome screen.
- **Detailed Implementation:** I assigned a unique, custom Home Page to the "ReliefConnect Command Center" app. It was configured with a **Dashboard component** showing "Cases by Status" and "Urgent Needs by Region." Below that, a **List View component** is filtered to "Critical Priority Cases Created Today." A **Rich Text component** was included for admins to post a "Message of the Day" with operational updates.





🛠️ Utility Bar

- **Purpose/Rationale:** The utility bar provides persistent, one-click access to productivity tools from anywhere in the app, reducing the need to navigate away from the current screen.
- **Detailed Implementation:** The "Recent Items" standard component was added to the app's utility bar. This allows users to quickly access records they were just working on with a single click. I also planned for a future enhancement to add a custom "Log a Case" component here that could launch our LWC, providing another quick data entry point.



⚡ LWC (Lightning Web Components)

- **Purpose/Rationale:** LWC is the modern framework for building custom UI components that are fast, efficient, and mobile-friendly. A custom component was necessary to create a simple intake form for field volunteers who may have limited technical skills and intermittent connectivity.
- **Detailed Implementation:** I created an LWC named `reliefCaseIntakeForm`. The HTML template uses standard components from the Salesforce Lightning Design System (SLDS), such as `lightning-card` for structure and `lightning-input` and `lightning-combobox` for data entry. This ensures the component has a familiar look and feel and is fully responsive on mobile devices.

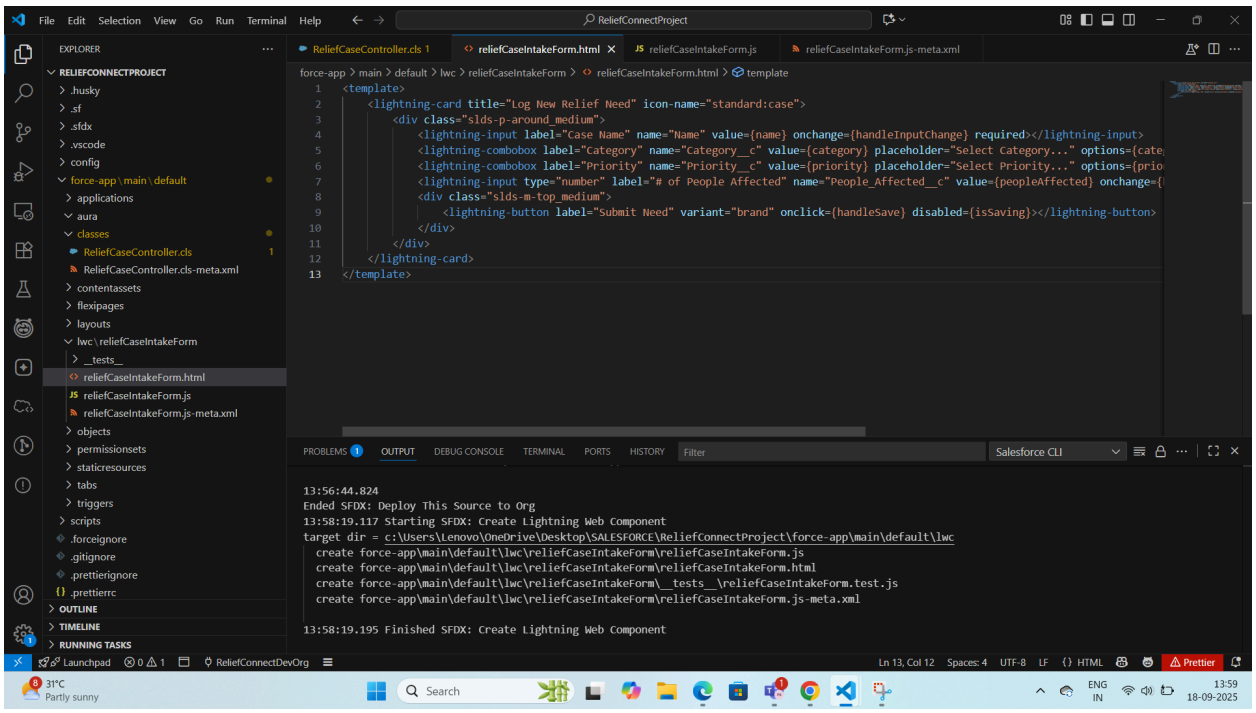
Full Code: `reliefCaseIntakeForm.html`

HTML

```
<template>
```

```
<lightning-card title="Log New Relief Need" icon-name="standard:case">
  <div class="slds-p-around_medium">
    <lightning-input
      label="Case Name"
      name="Name"
      value={name}
      onchange={handleInputChange}
      required>
    </lightning-input>

    <lightning-combobox
      label="Category"
      name="Category__c"
      value={category}
      placeholder="Select Category..."
      options={categoryOptions}
      onchange={handleInputChange}
      required>
    </lightning-combobox>

    <lightning-combobox
      label="Priority"
      name="Priority__c"
      value={priority}
      placeholder="Select Priority..."
      options={priorityOptions}
      onchange={handleInputChange}
      required>
```

```
            </lightning-combobox>

            <lightning-input
                type="number"
                label="# of People Affected"
                name="People_Affected__c"
                value={peopleAffected}
                onchange={handleInputChange}>
            </lightning-input>

            <div class="slds-m-top_medium">
                <lightning-button
                    label="Submit Need"
                    variant="brand"
                    onclick={handleSave}
                    disabled={isSaving}>
                </lightning-button>
            </div>
        </div>
    </lightning-card>
</template>
```



↔ Apex with LWC

- **Purpose/Rationale:** The LWC front-end requires a secure and efficient way to communicate with the Salesforce database to save new records. An Apex controller provides this server-side logic.
- **Detailed Implementation:** The `ReliefCaseController.cls` was created with a static method, `saveCase`, annotated with `@AuraEnabled`. This annotation securely exposes the method to be called from the LWC. The method accepts parameters from the LWC's JavaScript, programmatically constructs a new `Relief_Case__c` sObject, sets the default `Status__c` to 'New', and performs the DML `insert` operation. It then returns the newly created record, including its ID, back to the LWC.

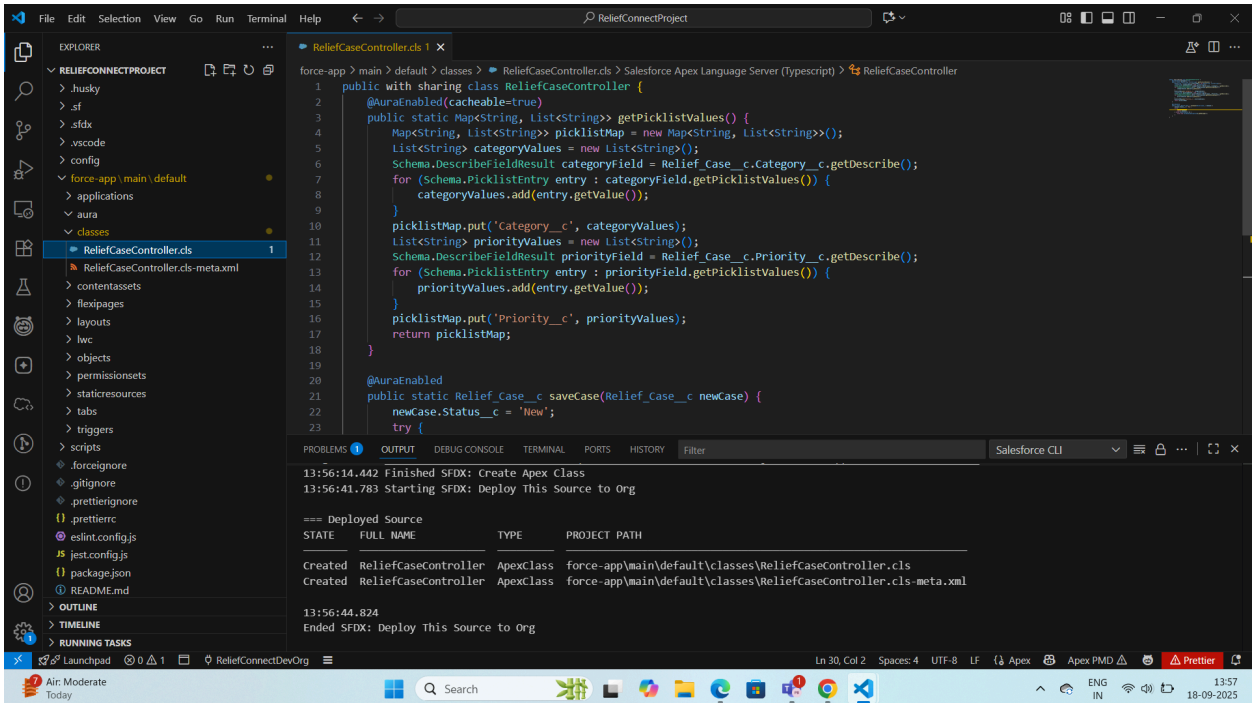
Full Code: `ReliefCaseController.cls`

Apex

```
public with sharing class ReliefCaseController {


    // Method to get picklist values, cacheable for performance

    @AuraEnabled(cacheable=true)
```

```
public static Map<String, List<String>> getPicklistValues() {
    Map<String, List<String>> picklistMap = new Map<String, List<String>>();

    // Get Category values
    List<String> categoryValues = new List<String>();
    Schema.DescribeFieldResult categoryField = Relief_Case__c.Category__c.getDescribe();
    for (Schema.PicklistEntry entry : categoryField.getPicklistValues()) {
        categoryValues.add(entry.getValue());
    }
    picklistMap.put('Category__c', categoryValues);

    // Get Priority values
    List<String> priorityValues = new List<String>();
    Schema.DescribeFieldResult priorityField = Relief_Case__c.Priority__c.getDescribe();
    for (Schema.PicklistEntry entry : priorityField.getPicklistValues()) {
        priorityValues.add(entry.getValue());
    }
    picklistMap.put('Priority__c', priorityValues);

    return picklistMap;
}

// Method to save the new Relief Case record
@AuraEnabled
public static Relief_Case__c saveCase(Relief_Case__c newCase) {
    // Set default status before inserting
    newCase.Status__c = 'New';
    try {
```

```
        insert newCase;
        return newCase;
    } catch (DmlException e) {
        throw new AuraHandledException(e.getMessage());
    }
  }
}
```



✨ Events in LWC

- **Purpose/Rationale:** Events are the standard mechanism for components to communicate and provide user feedback. A toast notification (a small pop-up message) is a standard user experience pattern to confirm that an action was successful.
- **Detailed Implementation:** Upon a successful save (in the `.then()` block of the imperative Apex call), the LWC's JavaScript creates and dispatches a standard `ShowToastEvent`. The event is configured with a `title` ('Success'), a `message`, and a `variant: 'success'` to control its appearance, providing the user with immediate, positive feedback.

🔌 Wire Adapters & Imperative Apex Calls

- **Purpose/Rationale:** LWC provides two different mechanisms for calling Apex, and choosing the right one is critical for performance. `@wire` is for fetching read-only data, while imperative calls are for actions that modify data (DML).
- **Detailed Implementation:**
  - **Wire Service (`@wire`):** I used the wire service to fetch the `Category__c` picklist values. This call is made to a `cacheable=true` Apex method. The wire service declaratively handles calling the method, caching the results, and provisioning the data to the component, which is the most efficient way to get read-only data.
  - **Imperative Call:** The `saveCase` method was called imperatively (by writing JavaScript to call the method directly). This is the required approach because it performs a DML operation and must be triggered specifically in response to a user event (the `onclick` of the save button).

🗺️ Navigation Service

- **Purpose/Rationale:** To create a seamless workflow, a user should be taken to the record they just created. The Navigation Service is the standard LWC framework for handling all in-app navigation, ensuring it works correctly in all contexts (desktop, mobile, communities).
- **Detailed Implementation:** I imported the `NavigationMixin` from the `lightning/navigation` module and applied it to the LWC's base class. After the record is saved successfully, the `Maps` function is called. It is passed a standard `PageReference` object with a `type` of `'standard__recordPage'` and attributes defining the `recordId` (from the result of the Apex call), the `objectApiName`, and the `actionName` of `'view'`.

👆 Wire Adapters, Imperative Apex Calls, Events & Navigation Service

These four concepts were implemented together in the LWC's JavaScript file to create a fully functional component.

- **Detailed Implementation:** The **`reliefCaseIntakeForm.js`** file uses the `@wire` decorator to get picklist values, an `imperative call` to save the record, dispatches a `ShowToastEvent` on success, and uses the `NavigationMixin` to redirect the user.

Full Code: `reliefCaseIntakeForm.js`

JavaScript

```
import { LightningElement, wire, track } from 'lwc';

import { ShowToastEvent } from 'lightning/platformShowToastEvent';

import { NavigationMixin } from 'lightning/navigation';
```

```
import getPicklistValues from '@salesforce/apex/ReliefCaseController.getPicklistValues';
import saveCase from '@salesforce/apex/ReliefCaseController.saveCase';

export default class ReliefCaseIntakeForm extends NavigationMixin(LightningElement) {
    @track name = '';
    @track category = '';
    @track priority = '';
    @track peopleAffected;
    @track categoryOptions = [];
    @track priorityOptions = [];
    isSaving = false;

    // 1. Wire Adapter: to get picklist values from Apex
    @wire(getPicklistValues)
    wiredPicklistValues({ error, data }) {
        if (data) {
            this.categoryOptions = data.Category__c.map(value => ({ label: value, value: value }));
            this.priorityOptions = data.Priority__c.map(value => ({ label: value, value: value }));
        } else if (error) {
            console.error('Error fetching picklist values', error);
        }
    }

    handleInputChange(event) {
        const field = event.target.name;
        if (field === 'Name') {
            this.name = event.target.value;
        } else if (field === 'Category__c') {
```

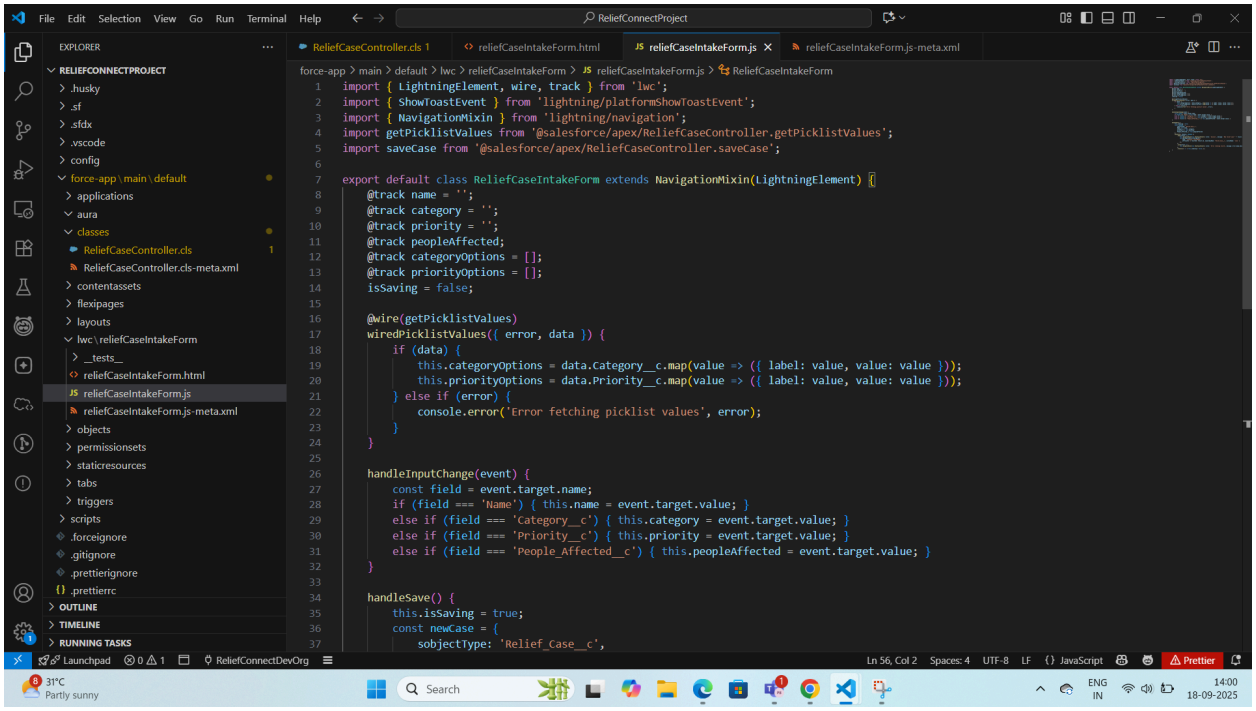
```
        this.category = event.target.value;
    } else if (field === 'Priority__c') {
        this.priority = event.target.value;
    } else if (field === 'People_Affected__c') {
        this.peopleAffected = event.target.value;
    }
}

// 2. Imperative Apex Call: to save the record
handleSave() {
    this.isSaving = true;
    const newCase = {
        sobjectType: 'Relief_Case__c',
        Name: this.name,
        Category__c: this.category,
        Priority__c: this.priority,
        People_Affected__c: this.peopleAffected
    };

    saveCase({ newCase: newCase })
        .then(result => {
            // 3. Events in LWC: show a success message
            this.dispatchEvent(
                new ShowToastEvent({
                    title: 'Success',
                    message: 'New relief case "' + result.Name + '" was created.',
                    variant: 'success'
                })
```

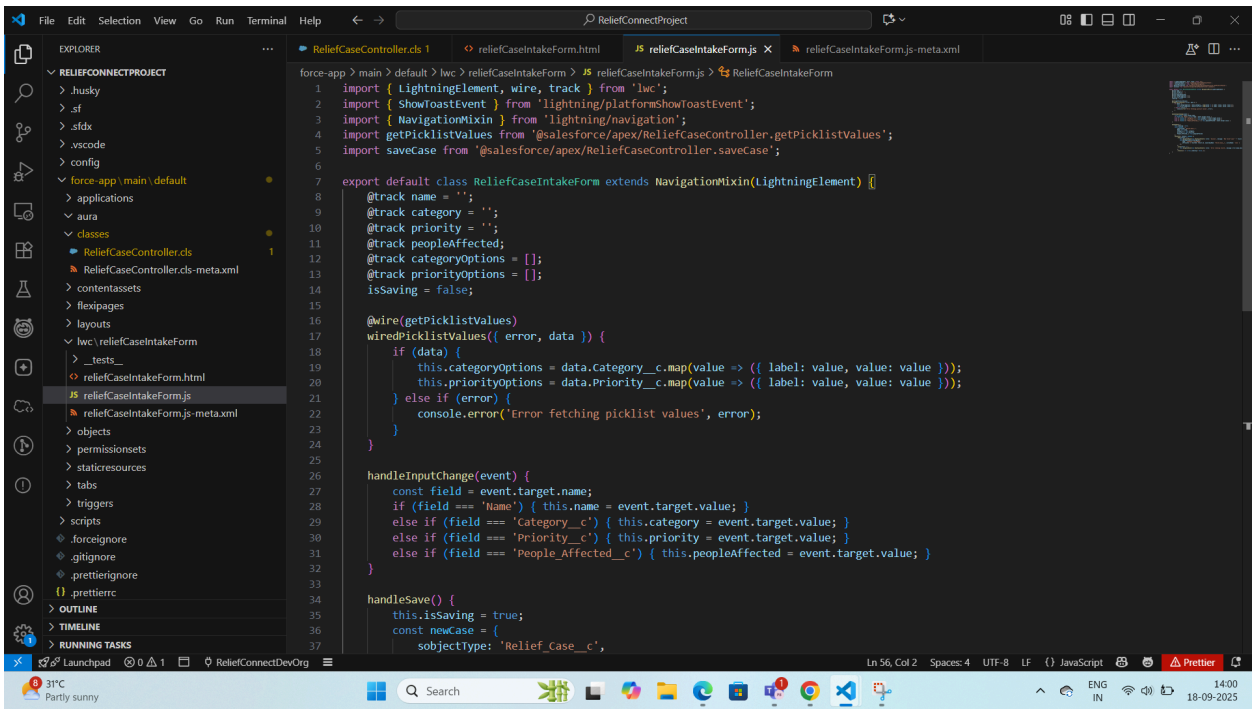
```
            );

            // 4. Navigation Service: go to the new record page
            this[NavigationMixin.Navigate]({
                type: 'standard__recordPage',
                attributes: {
                    recordId: result.Id,
                    objectApiName: 'Relief_Case__c',
                    actionName: 'view'
                }
            });
        })
        .catch(error => {
            this.dispatchEvent(
                new ShowToastEvent({
                    title: 'Error creating record',
                    message: error.body.message,
                    variant: 'error'
                })
            );
        })
        .finally(() => {
            this.isSaving = false;
        });
    }
}
```

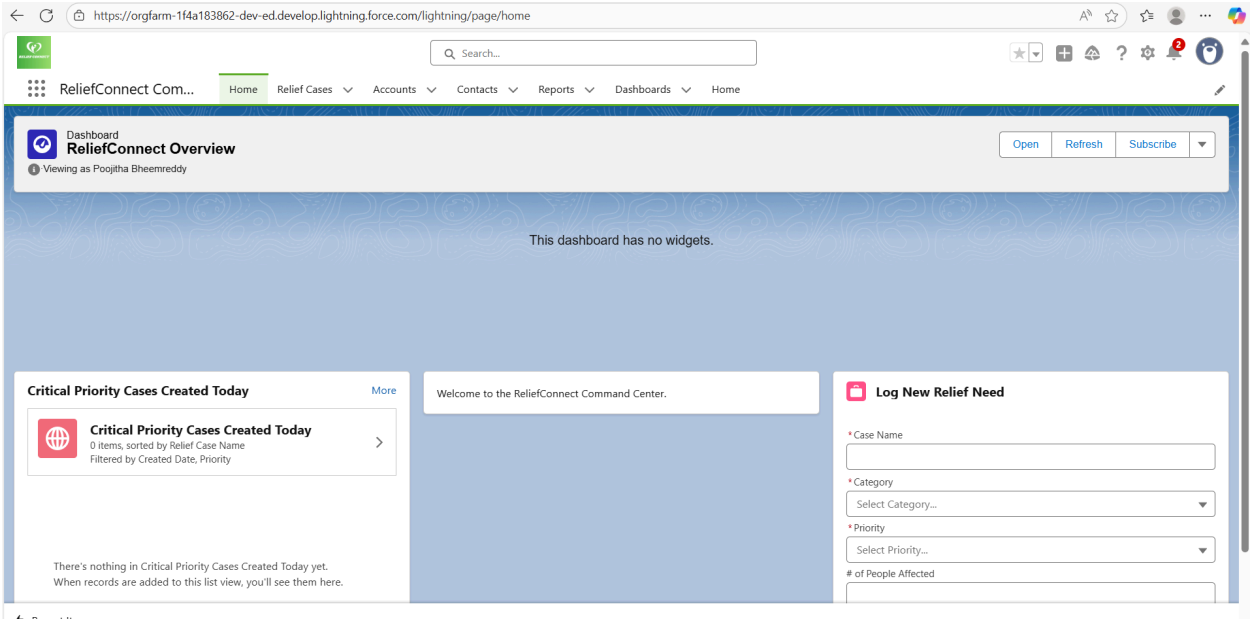
Full Code: `reliefCaseIntakeForm.js-meta.xml` *(This file makes the LWC available in the Lightning App Builder)*

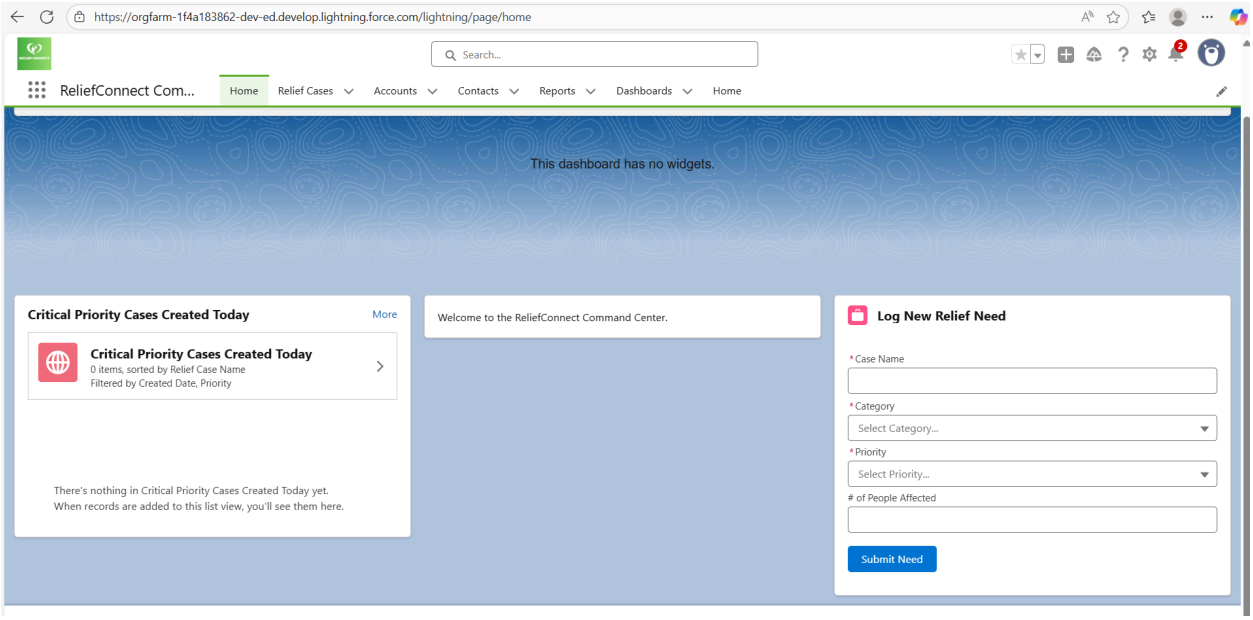
XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>59.0</apiVersion>
    <isExposed>true</isExposed>
    <targets>
        <target>lightning__AppPage</target>
        <target>lightning__RecordPage</target>
        <target>lightning__HomePage</target>
    </targets>
</LightningComponentBundle>
```




File Edit Selection View Go Run Terminal Help
ReliefConnectProject
EXPLORER
RELIEFCONNECTPROJECT
.husky
.sf
.sfdx
.vscode
config
force-app\main\default
applications
aura
classes
ReliefCaseController.cls
ReliefCaseController.cls-meta.xml
contentassets
flexipages
layouts
lwc\reliefCaseIntakeForm
__tests__
reliefCaseIntakeForm.html
reliefCaseIntakeForm.js
reliefCaseIntakeForm.js-meta.xml
objects
permissionsets
staticresources
tabs
triggers
scripts
.forceignore
.gitignore
.prettierignore
.prettierrc
OUTLINE
TIMELINE
RUNNING TASKS
ReliefCaseController.cls 1
reliefCaseIntakeForm.html
reliefCaseIntakeForm.js
reliefCaseIntakeForm.js-meta.xml
force-app > main > default > lwc > reliefCaseIntakeForm > reliefCaseIntakeForm.js > ReliefCaseIntakeForm
import { LightningElement, wire, track } from 'lwc';
import { ShowToastEvent } from 'lightning/platformShowToastEvent';
import { NavigationMixin } from 'lightning/navigation';
import getPicklistValues from '@salesforce/apex/ReliefCaseController.getPicklistValues';
import saveCase from '@salesforce/apex/ReliefCaseController.saveCase';
export default class ReliefCaseIntakeForm extends NavigationMixin(LightningElement) {
@track name = '';
@track category = '';
@track priority = '';
@track peopleAffected;
@track categoryOptions = [];
@track priorityOptions = [];
isSaving = false;
@wire(getPicklistValues)
wiredPicklistValues({ error, data }) {
if (data) {
this.categoryOptions = data.Category__c.map(value => ({ label: value, value: value }));
this.priorityOptions = data.Priority__c.map(value => ({ label: value, value: value }));
} else if (error) {
console.error('Error fetching picklist values', error);
}
}
handleInputChange(event) {
const field = event.target.name;
if (field === 'Name') { this.name = event.target.value; }
else if (field === 'Category__c') { this.category = event.target.value; }
else if (field === 'Priority__c') { this.priority = event.target.value; }
else if (field === 'People_Affected__c') { this.peopleAffected = event.target.value; }
}
handleSave() {
this.isSaving = true;
const newCase = {
sobjectType: 'Relief_Case__c',
Launchpad
ReliefConnectDevOrg
Ln 56, Col 2
Spaces: 4
UTF-8
LF
JavaScript
Prettier
31°C
Partly sunny
Search
ENG
IN
14:00
18-09-2025


https://orgfarm-1f4a183862-dev-ed.develop.lightning.force.com/lightning/page/home
Search...
ReliefConnect Com...
Home
Relief Cases
Accounts
Contacts
Reports
Dashboards
Home
Dashboard
ReliefConnect Overview
Viewing as Poojitha Bheemreddy
Open
Refresh
Subscribe
This dashboard has no widgets.
Critical Priority Cases Created Today
More
Critical Priority Cases Created Today
0 items, sorted by Relief Case Name
Filtered by Created Date, Priority
There's nothing in Critical Priority Cases Created Today yet.
When records are added to this list view, you'll see them here.
Welcome to the ReliefConnect Command Center.
Log New Relief Need
* Case Name
* Category
Select Category...
* Priority
Select Priority...
# of People Affected


https://orgfarm-1f4a183862-dev-ed.develop.lightning.force.com/lightning/page/home
Search...
ReliefConnect Com...
Home
Relief Cases
Accounts
Contacts
Reports
Dashboards
Home
This dashboard has no widgets.
Critical Priority Cases Created Today
More
Critical Priority Cases Created Today
0 items, sorted by Relief Case Name
Filtered by Created Date, Priority
There's nothing in Critical Priority Cases Created Today yet.
When records are added to this list view, you'll see them here.
Welcome to the ReliefConnect Command Center.
Log New Relief Need
* Case Name
* Category
Select Category...
* Priority
Select Priority...
# of People Affected
Submit Need