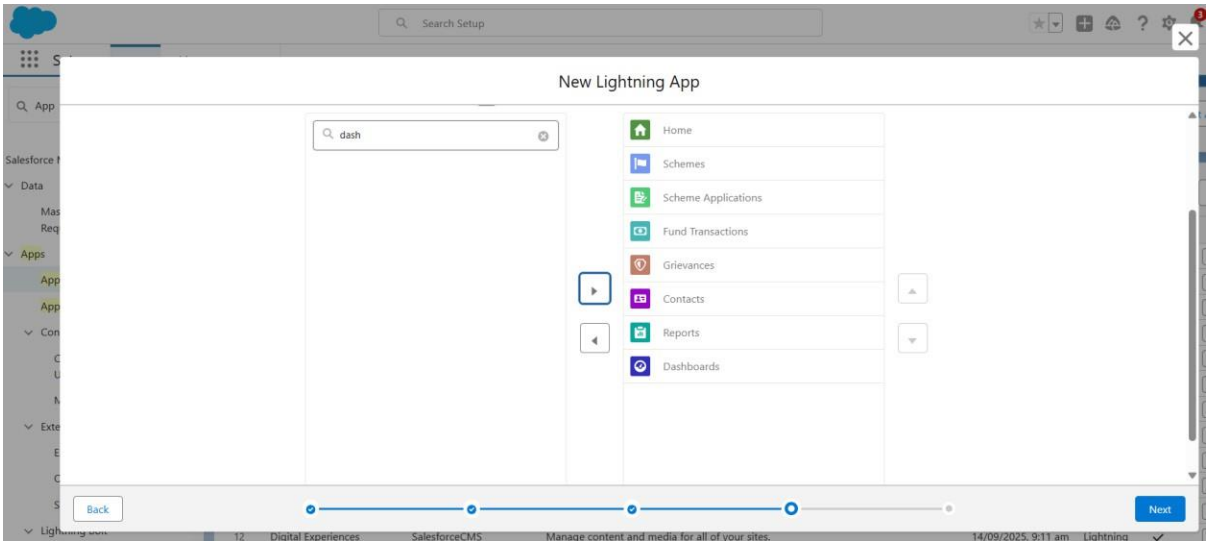
Phase 6: User Interface Development

In Phase 6, the user interface of the Salesforce application was customized to create an intuitive and user-friendly environment for managing **government schemes, citizen applications, and fund disbursements** through the **Government Schemes Portal app**. The design focused on making essential information easily accessible for **Officers, Managers, and Administrators**.

□ Lightning App Builder

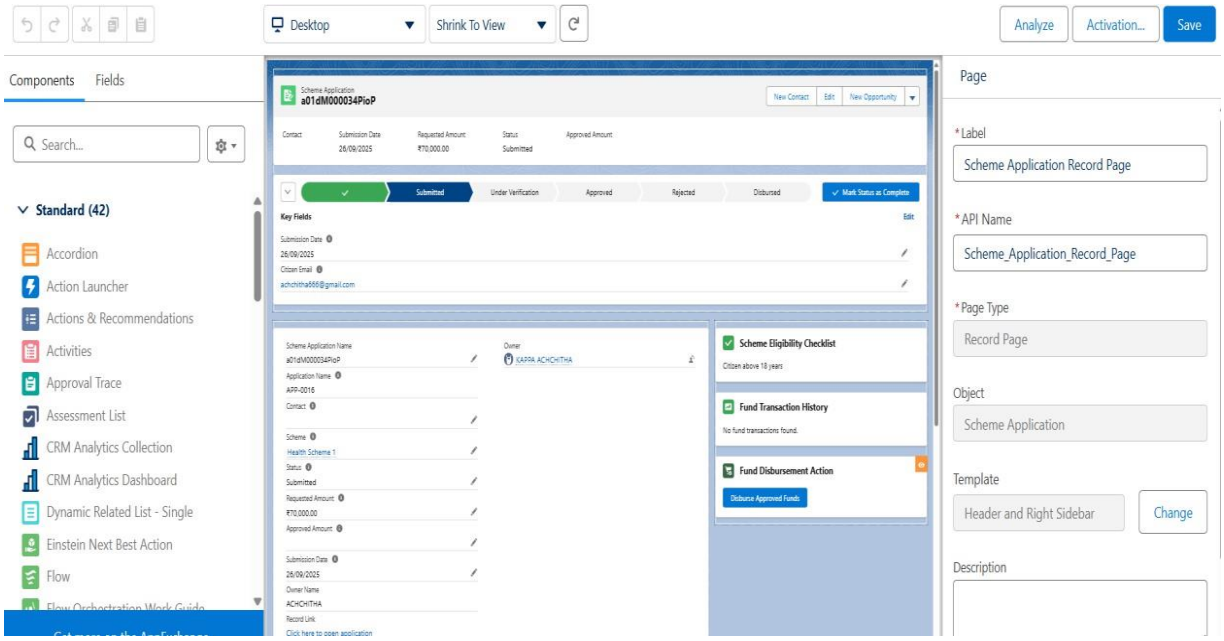
- A new custom Lightning App named **Government Schemes Portal** was created.
- Navigation Items added:
    - Home
    - Schemes
    - Scheme Applications
    - Fund Transactions
    - Grievances
    - Contacts
    - Reports
    - Dashboards



□ Record Pages

The **Scheme Application Record Page** was customized to display important details and related records:

- **Highlights Panel**: shows key details of each application.
- **Path**: displays the current **Status__c** of the application (e.g., Submitted, Verified, Approved, Rejected).
- **Record Detail**: includes applicant details, scheme information, and approved amount.
- **Related Lists**: show Fund Transactions, Grievances, and other linked records.



Custom **Lightning Web Components (LWCs)** were added to the sidebar:

1. **applicationChecklist** → Displays the eligibility criteria of the selected scheme.
2. **fundTransactionHistory** → Shows past fund transactions linked to the application.
3. **disburseFundsButton** → Allows officers to disburse funds for approved applications.

□ Tabs

Navigation tabs were added in the **Government Schemes Portal app** to provide quick access to:

- Home
- Schemes
- Scheme Applications
- Fund Transactions
- Grievances

- Contacts
- Reports
- Dashboards

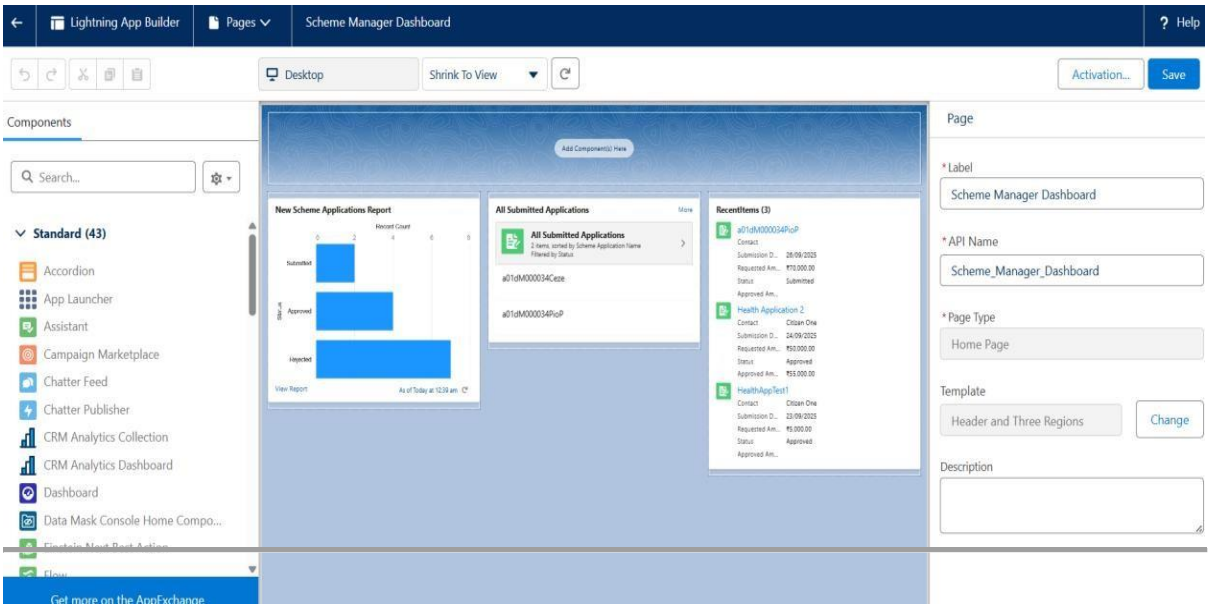
This ensures officers can move seamlessly between different areas of the system.

□ Home Page Layouts

A custom **Home Page** was created for the app using **Lightning App Builder**.

- **Left Column** → Report Chart showing the count of Scheme Applications grouped by Status.
- **Middle Column** → List View showing "All Submitted Applications".
- **Right Column** → Recent Applications (recently created/modified records).

This dashboard-style layout gives officers a quick overview of scheme performance and pending work.



□ Utility Bar

The Utility Bar in the **Government Schemes Portal** can provide quick access to frequently used tools such as Notes or Recent Records. For simplicity, default utilities were used.

□ Lightning Web Components (LWC)

Unlike the demo project, this system required **custom LWCs** for handling scheme workflows:

- **applicationChecklist** (uses Apex wire to fetch scheme eligibility criteria).

force-app > main > default > lwc > applicationChecklist > applicationChecklist.html > ...

```
<template>
    <lightning-card title="Scheme Eligibility Checklist" icon-name="standard:task2">
        <div class="slds-m-around_medium">
            <template if:true={checklist}>
                <p>{checklist}</p>
            </template>
            <template if:true={error}>
                <p class="slds-text-color_error">{error}</p>
            </template>
        </div>
    </lightning-card>
</template>

```

force-app > main > default > lwc > applicationChecklist > applicationChecklist.js > ...

```
import { LightningElement, api, wire } from 'lwc';
import getApplicationChecklist from '@salesforce/apex/SchemeApplicationController.getApplicationChecklist';

export default class ApplicationChecklist extends LightningElement {
    @api recordId;
    checklist;
    error;

    @wire(getApplicationChecklist, { applicationId: '$recordId' })
    wiredChecklist({ error, data }) {
        if (data) {
            this.checklist = data;
            this.error = undefined;
        } else if (error) {
            this.error = 'Could not load eligibility criteria.';
            this.checklist = undefined;
        }
    }
}

```

- **fundTransactionHistory** (uses Apex wire to fetch related Fund Transactions).

force-app > main > default > lwc > fundTransactionHistory > fundTransactionHistory.html > ...

```
<template>
    <lightning-card title="Fund Transaction History" icon-name="standard:investment_account">
        <div class="slds-m-around_medium">
            <template if:true={transactions}>
                <template for:each={transactions} for:item="tx">
                    <p key={tx.Id} class="slds-border_bottom slds-p-bottom_x-small slds-m-bottom_x-small">
                        <b>{tx.Name}:</b> An amount of ${tx.Amount__c} was {tx.Payment_Status__c} on {tx.Payment_Date__c}.
                    </p>
                </template>
            </template>
            <template if:true={error}>
                <p class="slds-text-color_error">{error}</p>
            </template>
            <template if:false={transactions}>
                <p>No fund transactions found.</p>
            </template>
        </div>
    </lightning-card>
</template>

```

force-app > main > default > lwc > fundTransactionHistory > JS fundTransactionHistory.js > ...

```
import { LightningElement, api, wire } from 'lwc';
import getFundTransactions from '@salesforce/apex/SchemeApplicationController.getFundTransactions';

export default class FundTransactionHistory extends LightningElement {
    @api recordId;
    transactions;
    error;

    @wire(getFundTransactions, { applicationId: '$recordId' })
    wiredTransactions({ error, data }) {
        if (data) {
            this.transactions = data.length > 0 ? data : false;
            this.error = undefined;
        } else if (error) {
            this.error = 'Failed to load fund transactions.';
            this.transactions = undefined;
        }
    }
}

```

- **disburseFundsButton** (uses imperative Apex call to disburse funds securely).

force-app > main > default > lwc > disburseFundsButton > <> disburseFundsButton.html > ...

```
<template>
    <lightning-card title="Fund Disbursement Action" icon-name="standard:checkout">
        <div class="slds-m-around_medium">
            <lightning-button
                label="Disburse Approved Funds"
                variant="brand"
                onclick={handleDisburseClick}
                disabled={isLoading}>
            </lightning-button>
        </div>
    </lightning-card>
</template>

```

force-app > main > default > lwc > disburseFundsButton > JS disburseFundsButton.js > ...

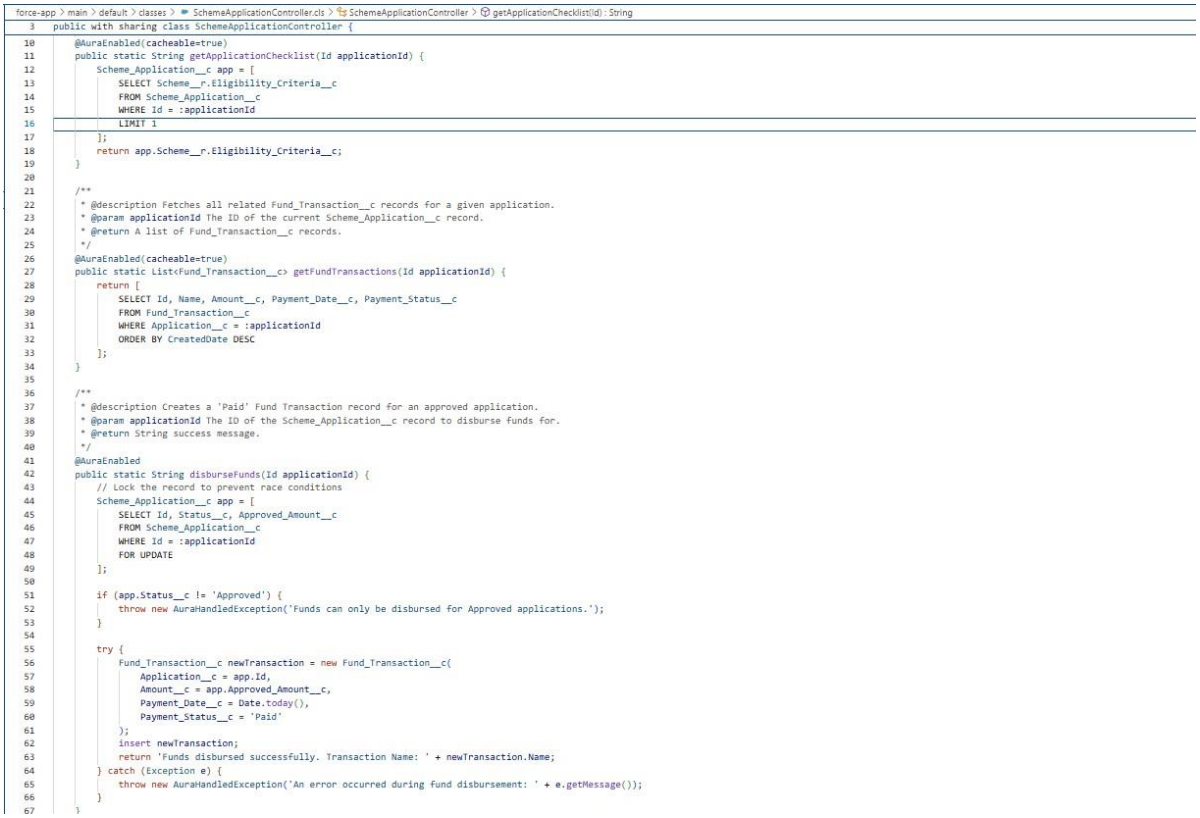
```
import { LightningElement, api } from 'lwc';
import { ShowToastEvent } from 'lightning/platformShowToastEvent';
import disburseFunds from '@salesforce/apex/SchemeApplicationController.disburseFunds';

export default class DisburseFundsButton extends LightningElement {
    @api recordId;
    isLoading = false;

    async handleDisburseClick() {
        this.isLoading = true;
        try {
            const result = await disburseFunds({ applicationId: this.recordId });
            this.showToast('Success', result, 'success');
        } catch (error) {
            const errorMessage = error.body ? error.body.message : error.message;
            this.showToast('Error', 'Fund Disbursement Failed: ' + errorMessage, 'error');
        } finally {
            this.isLoading = false;
        }
    }

    showToast(title, message, variant) {
        const event = new ShowToastEvent({ title, message, variant });
        this.dispatchEvent(event);
    }
}

```

□ Apex with LWC

Apex methods were created in **SchemeApplicationController** to:

- Retrieve eligibility criteria.
- Retrieve fund transactions.
- Disburse approved funds.

These methods are invoked directly from LWCs, ensuring smooth officer workflows.



```
force-app > main > default > classes > SchemeApplicationController.cls > SchemeApplicationController > getApplicationChecklist(Id) : String
3   public with sharing class SchemeApplicationController {
10      @AuraEnabled(cacheable=true)
11      public static String getApplicationChecklist(Id applicationId) {
12          Scheme_Application__c app = [
13              SELECT Scheme__r.Eligibility_Criteria__c
14              FROM Scheme_Application__c
15              WHERE Id = :applicationId
16              LIMIT 1
17          ];
18          return app.Scheme__r.Eligibility_Criteria__c;
19      }
20
21      /**
22       * @description Fetches all related Fund_Transaction__c records for a given application.
23       * @param applicationId The ID of the current Scheme_Application__c record.
24       * @return A list of Fund_Transaction__c records.
25       */
26      @AuraEnabled(cacheable=true)
27      public static List<Fund_Transaction__c> getFundTransactions(Id applicationId) {
28          return [
29              SELECT Id, Name, Amount__c, Payment_Date__c, Payment_Status__c
30              FROM Fund_Transaction__c
31              WHERE Application__c = :applicationId
32              ORDER BY CreatedDate DESC
33          ];
34      }
35
36      /**
37       * @description Creates a 'Paid' Fund Transaction record for an approved application.
38       * @param applicationId The ID of the Scheme_Application__c record to disburse funds for.
39       * @return String success message.
40       */
41      @AuraEnabled
42      public static String disburseFunds(Id applicationId) {
43          // Lock the record to prevent race conditions
44          Scheme_Application__c app = [
45              SELECT Id, Status__c, Approved_Amount__c
46              FROM Scheme_Application__c
47              WHERE Id = :applicationId
48              FOR UPDATE
49          ];
50
51          if (app.Status__c != 'Approved') {
52              throw new AuraHandledException('Funds can only be disbursed for Approved applications.');
53          }
54
55          try {
56              Fund_Transaction__c newTransaction = new Fund_Transaction__c(
57                  Application__c = app.Id,
58                  Amount__c = app.Approved_Amount__c,
59                  Payment_Date__c = Date.today(),
60                  Payment_Status__c = 'Paid'
61              );
62              insert newTransaction;
63              return 'Funds disbursed successfully. Transaction Name: ' + newTransaction.Name;
64          } catch (Exception e) {
65              throw new AuraHandledException('An error occurred during fund disbursement: ' + e.getMessage());
66          }
67      }
```

□ Events and Wire Adapters in LWC

- **Wire Adapters** were used in LWCs to fetch application data reactively (e.g., eligibility checklist and transaction history).
- **Imperative Apex Calls** were used in the disburse button to execute fund disbursement only when triggered by the officer.
- **Toast Events** notify officers of success or failure during fund disbursement.

□ Navigation Service

Standard navigation via tabs and record pages was sufficient. Programmatic navigation was not required in this phase.