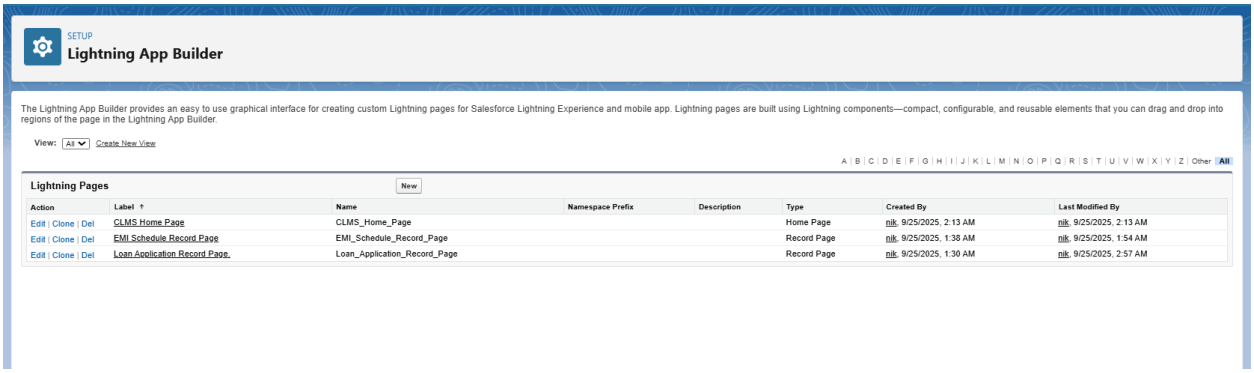
Phase 6: User Interface Development

Introduction

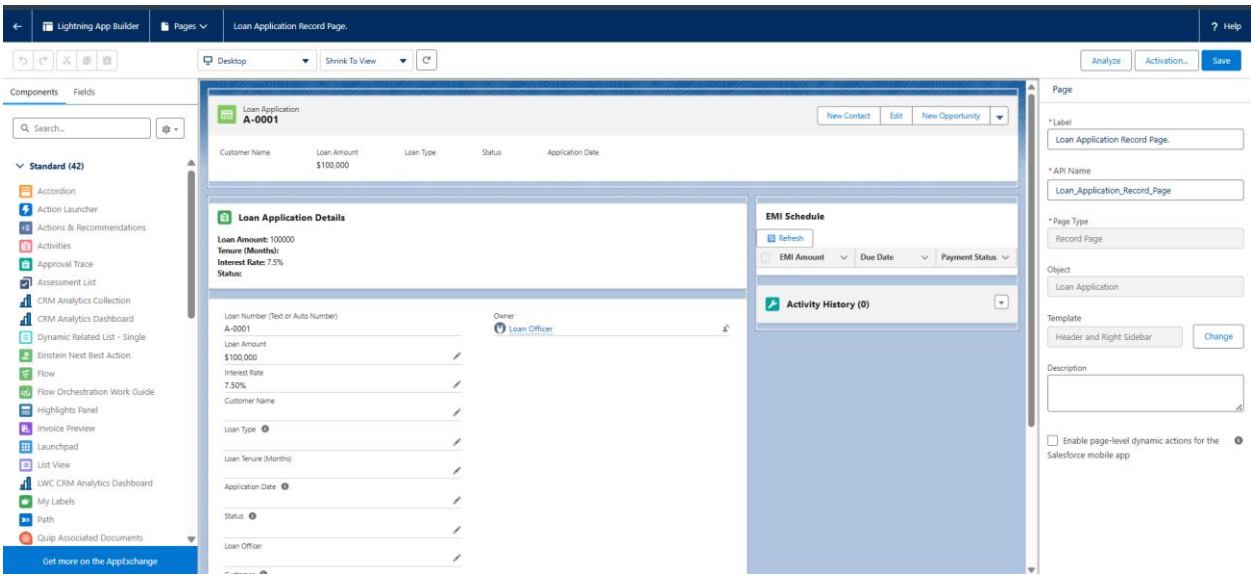
Phase 6 focused on the UI/UX layer of CLMS using Salesforce Lightning Experience.
The objective was to create user-friendly, modular, and dynamic interfaces so that Loan Officers, Credit Managers, and Branch Managers can easily interact with loan applications, EMIs, approvals, and related records.

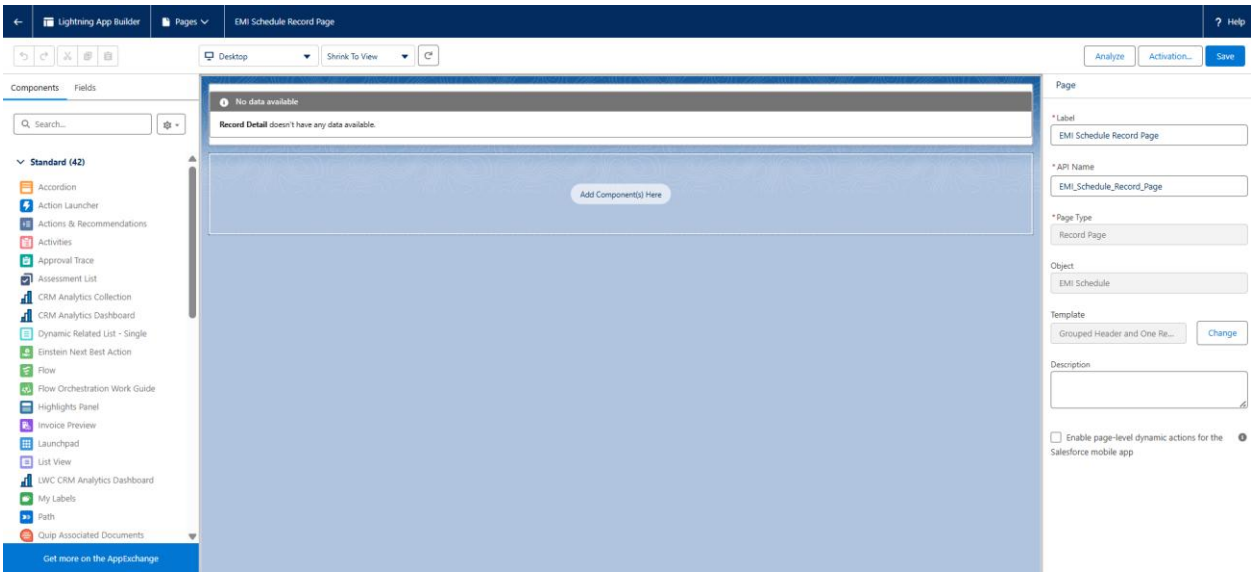
- Lightning App Builder
  - Built custom pages (App Page, Record Page, Home Page).
  - In CLMS:
    - Loan Application Record Page → shows Loan details + EMI Schedule.
    - CLMS Home Page → shows recent items, loan reports, pending approvals.
  - Outcome: Flexible layouts for different roles.



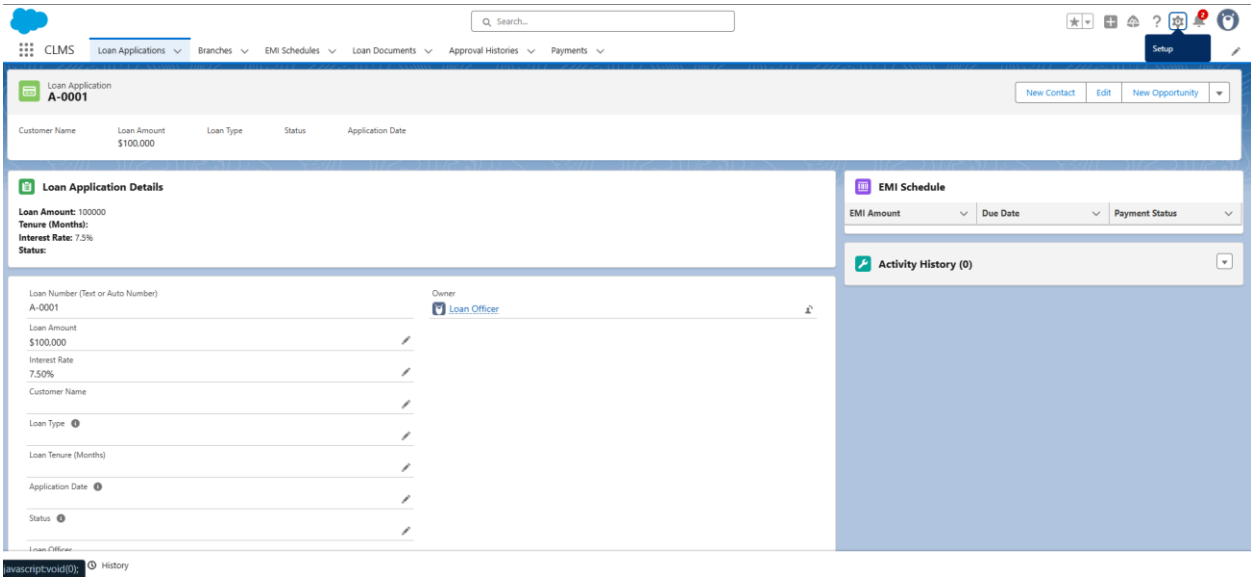
- Record Pages

- Custom record pages created for:
    - Loan Application (loan details + EMI Viewer LWC).
    - EMI Schedule (EMI details + related Loan).
    - Loan Document (document details + loan reference).
    - Approval History (role, decision, comments).
- Outcome: Each record has its own optimized Lightning layout.

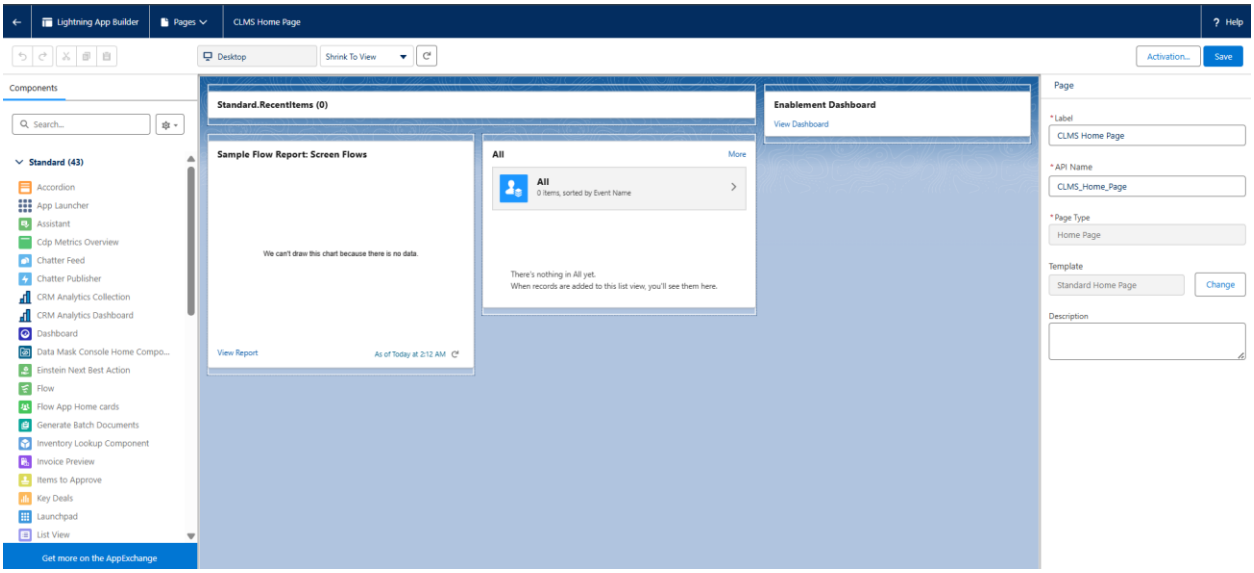




- Tabs
  - Added CLMS navigation tabs:
    - Loan Application, EMI Schedule, Payment, Loan Document, Approval History.
  - Outcome: Quick switching between CLMS objects.



- Home Page Layouts
  - Created CLMS Home Page via App Builder.
  - Components added:
    - Recent Items → quick access to loans.
    - Report Chart → Loans Approved This Month.
    - List View → Pending approvals.
  - Outcome: Centralized dashboard.



- Utility Bar
  - Added Utility Bar in CLMS App.
  - Items: Notes, Recent Items.
  - Outcome: Quick actions available across the app.

- Lightning Web Components (LWCs)

Developed LWCs for CLMS:

1. loanApplicationViewer → displays loan details.

2. emiScheduleViewer → lists EMI records + refresh option.

- Apex with LWC
  - Added Apex methods in LoanHelper.cls:
    - calculateEMI() → EMI logic.
    - getEmiByLoan() → fetch EMI schedules.
  - Used in LWC: emiScheduleViewer uses @wire + imperative calls.

- Events in LWC
  - Added event handling in LWCs:
    - Example: Refresh button triggers EMI list reload.
  - Outcome: LWCs became interactive and dynamic.

- Wire Adapters
  - Used @wire with getEmiByLoan.
  - Auto-updates EMI list when Loan Application changes.

- Imperative Apex Calls
  - Refresh button → calls Apex imperatively → reloads EMI list.
  - LOAN APPLICATION
    VIEWER

loanApplicationViewer.js-meta.xml | loanApplicationViewer.html | loanApplicationViewer.js

force-app > main > default > lwc > loanApplicationViewer > loanApplicationViewer.js-meta.xml > ...

```xml
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>64.0</apiVersion>
    <isExposed>true</isExposed>
    <targets>
        <target>lightning__RecordPage</target>
        <target>lightning__AppPage</target>
    </targets>
    <targetConfigs>
        <targetConfig targets="lightning__RecordPage">
            <objects>
                <object>Loan_Application__c</object>
            </objects>
        </targetConfig>
    </targetConfigs>
</LightningComponentBundle>
```

loanApplicationViewer.js-meta.xml | loanApplicationViewer.html | loanApplicationViewer.js

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

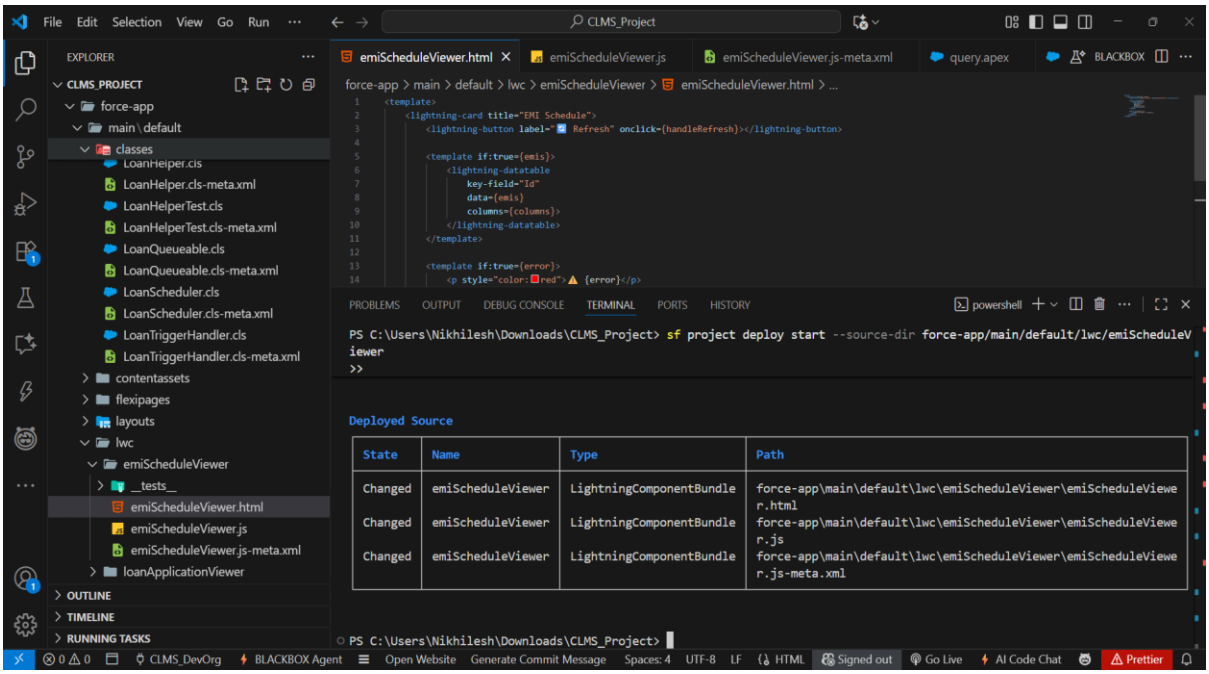
```html
<template>
    <lightning-card title="Loan Application Details" icon-name="standard:record">
        <div class="slds-m-around_medium">
            <p><b>Loan Amount:</b> {loanAmount}</p>
            <p><b>Tenure (Months):</b> {tenure}</p>
            <p><b>Interest Rate:</b> {interestRate}%</p>
            <p><b>Status:</b> {status}</p>
        </div>
    </lightning-card>
</template>
```

loanApplicationViewer.js-meta.xml | loanApplicationViewer.html | loanApplicationViewer.js

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

```
import { LightningElement, api, wire } from 'lwc';
import { getRecord } from 'lightning/uiRecordApi';

import LOAN_AMOUNT from '@salesforce/schema/Loan_Application__c.Loan_Amount__c';
import TENURE from '@salesforce/schema/Loan_Application__c.Loan_Tenure_Months__c';
import INTEREST_RATE from '@salesforce/schema/Loan_Application__c.Interest_Rate__c';
import STATUS from '@salesforce/schema/Loan_Application__c.Status__c';

export default class LoanApplicationViewer extends LightningElement {
    @api recordId; // record Id comes from Salesforce Record Page

    loanAmount;
    tenure;
    interestRate;
    status;

    @wire(getRecord, {
        recordId: '$recordId',
        fields: [LOAN_AMOUNT, TENURE, INTEREST_RATE, STATUS]
    })
    wiredLoan({ error, data }) {
        if (data) {
            this.loanAmount = data.fields.Loan_Amount__c.value;
            this.tenure = data.fields.Loan_Tenure_Months__c.value;
            this.interestRate = data.fields.Interest_Rate__c.value;
            this.status = data.fields.Status__c.value;
        } else if (error) {
            console.error('Error fetching Loan Application:', error);
        }
    }
}

```

- EMISCHEDULER VIEWER



emiScheduleViewer.html | emiScheduleViewer.js | emiScheduleViewer.js-meta.xml

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

```
<template>
    <lightning-card title="EMI Schedule">
        <lightning-button label="Refresh" onclick={handleRefresh}></lightning-button>

        <template if:true={emis}>
            <lightning-datatable
                key-field="Id"
                data={emis}
                columns={columns}>
            </lightning-datatable>
        </template>

        <template if:true={error}>
            <p style="color:red">⚠ {error}</p>
        </template>
    </lightning-card>
</template>
```

emiScheduleViewer.html | emiScheduleViewer.js | emiScheduleViewer.js-meta.xml

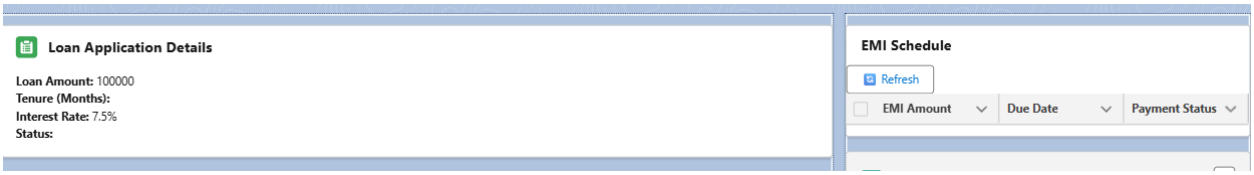
force-app > main > default > lwc > emiScheduleViewer > emiScheduleViewer.js > ...

```
import { LightningElement, api, track, wire } from 'lwc';
import getEmiByLoan from '@salesforce/apex/LoanHelper.getEmiByLoan';

export default class EmiScheduleViewer extends LightningElement {
    @api recordId;
    @track emis = [];
    @track error;

    columns = [
        { label: 'EMI Amount', fieldName: 'EMI_Amount__c', type: 'currency' },
        { label: 'Due Date', fieldName: 'Due_Date__c', type: 'date' },
        { label: 'Payment Status', fieldName: 'Payment_Status__c', type: 'text' }
    ];

    // Wire → auto fetch
    @wire(getEmiByLoan, { loanId: '$recordId' })
    wiredEmis({ data, error }) {
        if (data) {
            this.emis = data;
        } else if (error) {
            this.error = error;
        }
    }

    // Imperative → refresh button
    handleRefresh() {
        getEmiByLoan({ loanId: this.recordId })
            .then(result => {
                this.emis = result;
                this.error = undefined;
            })
```

emiScheduleViewer.html | emiScheduleViewer.js | emiScheduleViewer.js-meta.xml

force-app > main > default > lwc > emiScheduleViewer > emiScheduleViewer.js-meta.xml > ...

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>58.0</apiVersion>
    <isExposed>true</isExposed>
    <targets>
        <target>lightning__RecordPage</target>
    </targets>
    <targetConfigs>
        <targetConfig targets="lightning__RecordPage">
            <objects>
                <object>Loan_Application__c</object>
            </objects>
        </targetConfig>
    </targetConfigs>
</LightningComponentBundle>
```



- Navigation Service
  - Added record navigation inside LWCs.
  - Example: From EMI Schedule → back to Loan Application.

---

3. Outcomes

- Built modular Lightning Pages for CLMS.

- Created custom LWCs integrated with Apex.
- Enhanced usability with Home Page + Utility Bar.
- Added auto-refresh (wire) and manual refresh (imperative Apex).
- Implemented navigation & events for smooth user experience.

---

4. Conclusion

Phase 6 delivered a modern Lightning Experience UI for CLMS.
Users can now work with loans, EMIs, and approvals in a single consolidated workspace.
With LWCs and Apex integration, the UI is interactive, scalable, and user-friend