Integrated Banking Operations Hub With Integrated AI : A Salesforce Solution

Project Overview

This CRM project establishes an **Integrated Banking Hub** to unify critical financial operations: Loan Management, Wealth Management, and Regulatory Compliance. The platform automates high-volume lending decisions through Flows and Approval Processes, provides financial advisors with **real-time portfolio valuation**, and enforces security by creating an automated audit trail for compliance. The key business need addressed is replacing fragmented, manual processes with a secure, centralized, and efficient system.

Objectives

The main goals of building this CRM were to achieve operational efficiency and enhance regulatory posture.

- **Automate Lending:** Implement intelligent routing to ensure high-value loans (>₹20 Lakhs) receive mandatory manager review via an **Approval Process**, while low-value loans are auto-approved for streamlined booking.
- **Enable Real-Time Advice:** Calculate client portfolio value and goal progress dynamically using **Apex Triggers and Roll-Up logic** to enable proactive financial advice.
- **Ensure Compliance:** Establish a foundational **Audit Log** for critical changes and centralize risk monitoring using custom objects and automated Flows.
- **Visibility:** Deliver **custom Lightning Web Component (LWC) dashboards** for personalized, role-specific insights across all modules.

Key Features and Modules

The solution includes three main modules to address these issues:

1. **Loan Management Module:**
    - **Use Case:** Digitize the loan application, approval, and repayment process.
    - **Features:** Includes automated risk scoring, a multi-step approval process, and an automated audit trail.

2. **Wealth & Portfolio Management Module:**
    - **Use Case:** Centralize the tracking of a client's investment portfolio, assets, and goals.
    - **Features:** Provides **automatic calculation of portfolio value** (using Apex/Roll-Up logic), real-time asset tracking, and a dynamic LWC dashboard for advisors.
3. **Compliance & Risk Management Module:**
    - **Use Case:** Monitor and manage regulatory compliance and risk events across the bank.
    - **Features:** A system to track regulations, **audit major changes** (via Flow), and send alerts for compliance issues.

Phase 1: Problem Understanding & Industry Analysis

This phase outlines the initial steps for building the platform.

- **Requirement Gathering:** A single platform will be built to manage client records, loan applications, investment portfolios, and compliance regulations. Custom objects and automation will be used to connect the modules.

- **Stakeholder Analysis:** The project will impact several key roles:
    - **Loan Officers:** Will use the Loan Management Module to process applications.
    - **Wealth Advisors:** Will use the Wealth Management Module to manage client portfolios.
    - **Compliance Officers:** Will use the Compliance Module to enforce regulations.
    - **Management:** Will gain a complete view of the bank's operations, finances, and risk exposure.

- **Business Process Mapping:** The proposed process ensures that client information is entered only once. Actions like a loan application or creating an investment portfolio will be tracked by the compliance module to ensure security and compliance.
- **Industry-specific Use Case Analysis:** This project is considered innovative because it addresses data silos by building a single integrated platform, which is what real companies often do.
- **AppExchange Exploration:** The plan includes exploring tools like DocuSign for e-signatures, DocGen for document generation, and a free API for dummy stock prices.

Technical Implementation

The project will use various Salesforce capabilities:

- **Admin:** The plan includes building multiple objects, implementing Approval Processes for loans, and creating Validation Rules for compliance.

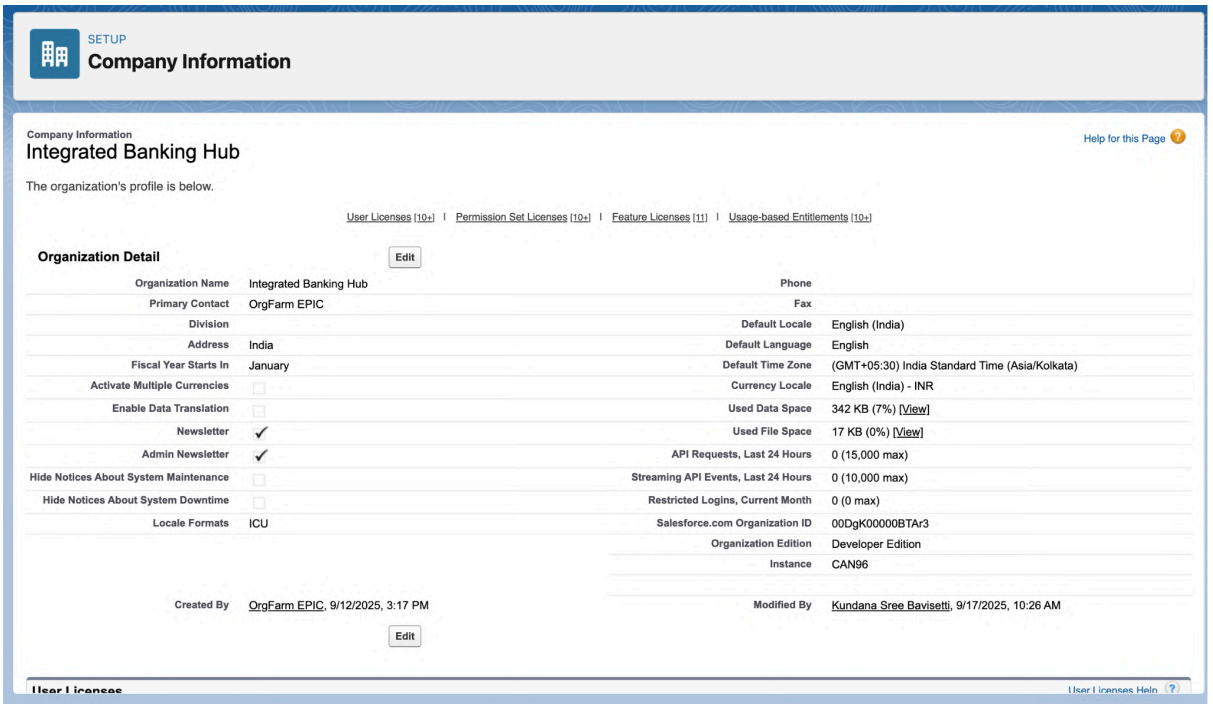
- **Developer:** The developer will use an Apex Trigger to calculate portfolio value, Batch Apex to update market data, and Scheduled Apex to send monthly compliance reports.
- **UI (LWC):** A custom Lightning Web Components (LWC) dashboard will be built for each module to provide a personalized user experience.
- **Integration:** REST callouts and Named Credentials will be used to get real-time stock prices.

Phase 2: Org Setup & Configuration

1. Company Profile Setup

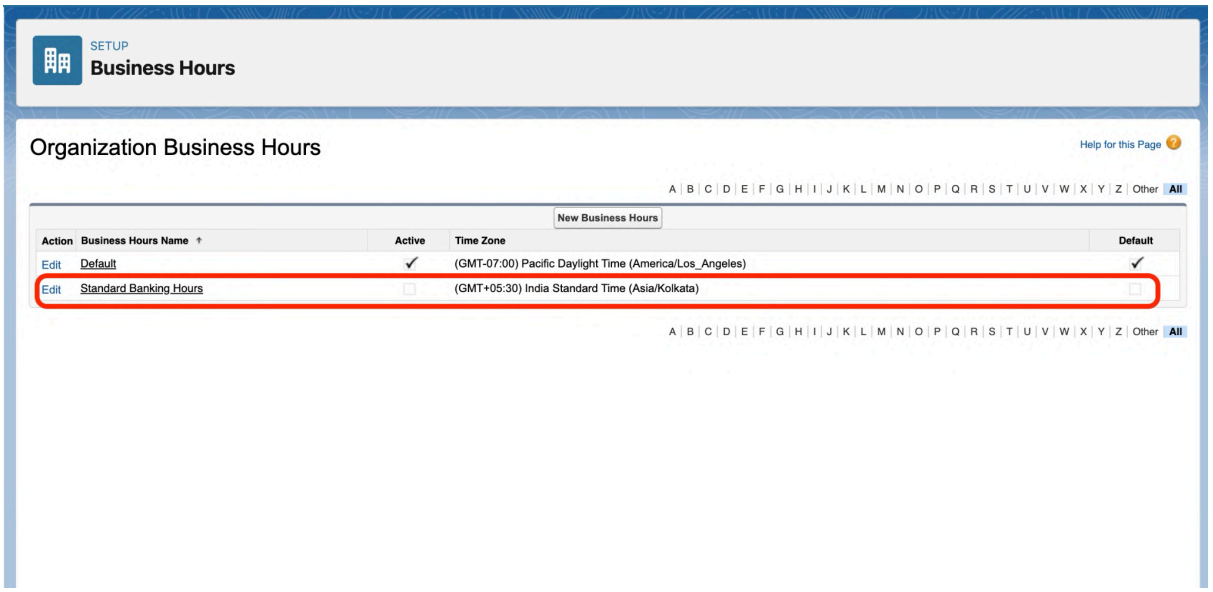
Scenario: setting up the basic details for our bank. This ensures that the time, currency, and language are correct for our project.

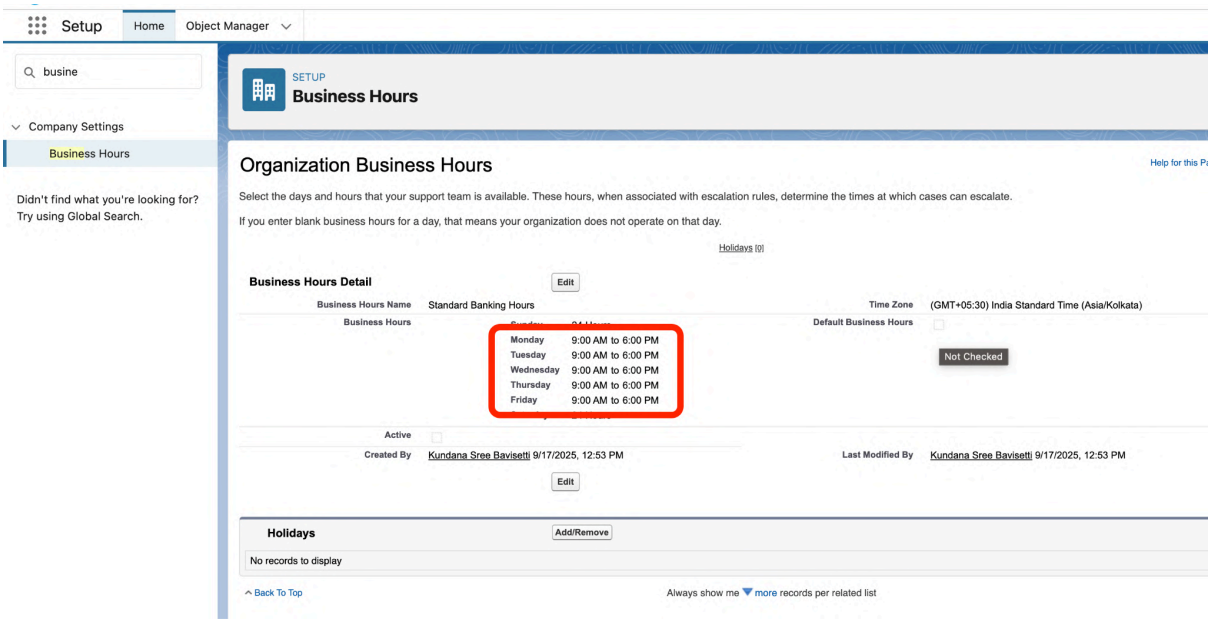
Organization Name: Integrated Banking Hub
Default Time Zone: (GMT+05:30) India Standard Time (Asia/Kolkata)
Default Currency: INR - Indian Rupee



In this step, I configured the basic organizational details for our project. Setting the organization name to **"Integrated Banking Hub"** and the timezone to **"India Standard Time"** is a critical first step. This ensures that all timestamps on records and all reports reflect the correct local time and currency for our operations.

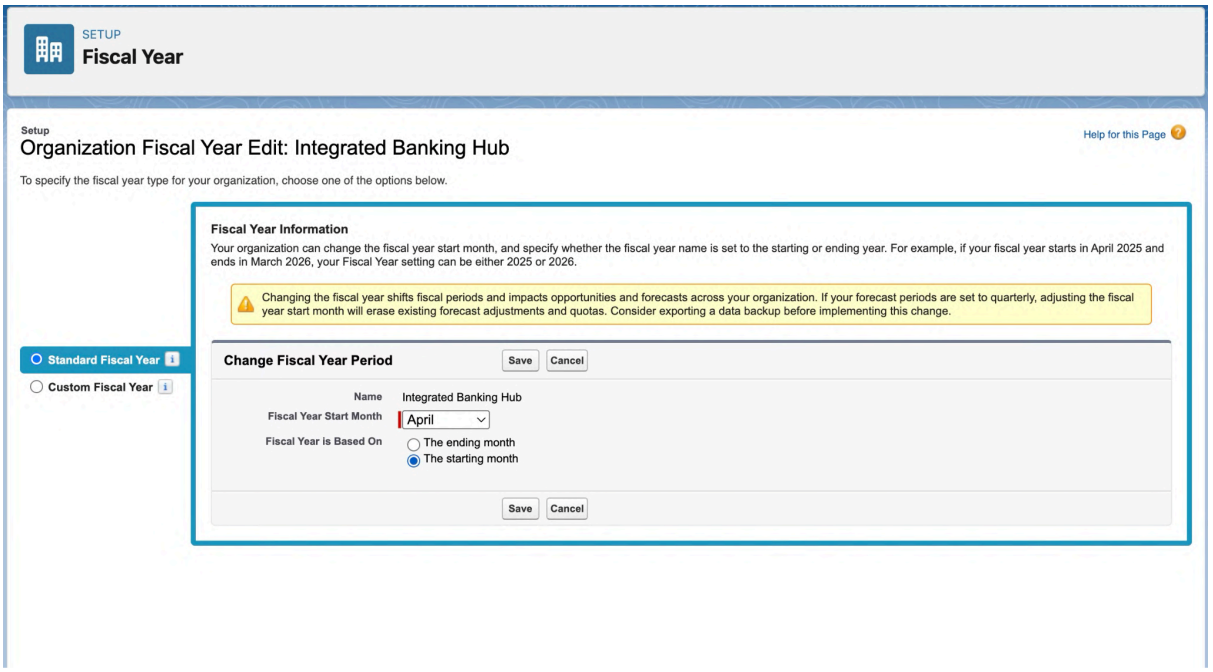
2. Business Hours Setup





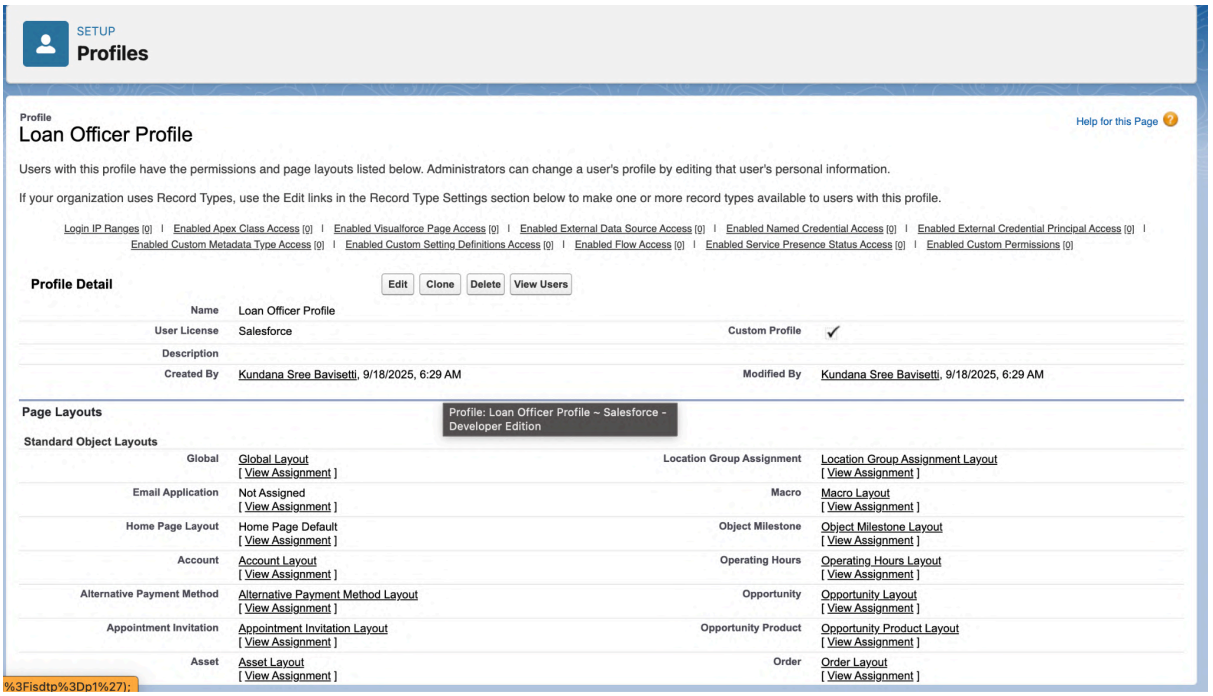
To support our automation rules and to define the bank's operational times, I have created a new business hours record. I named it **"Standard Banking Hours"** and set the working days from Monday to Friday, 9:00 AM to 6:00 PM. This will be used in a later phase for things like setting deadlines for automated processes.

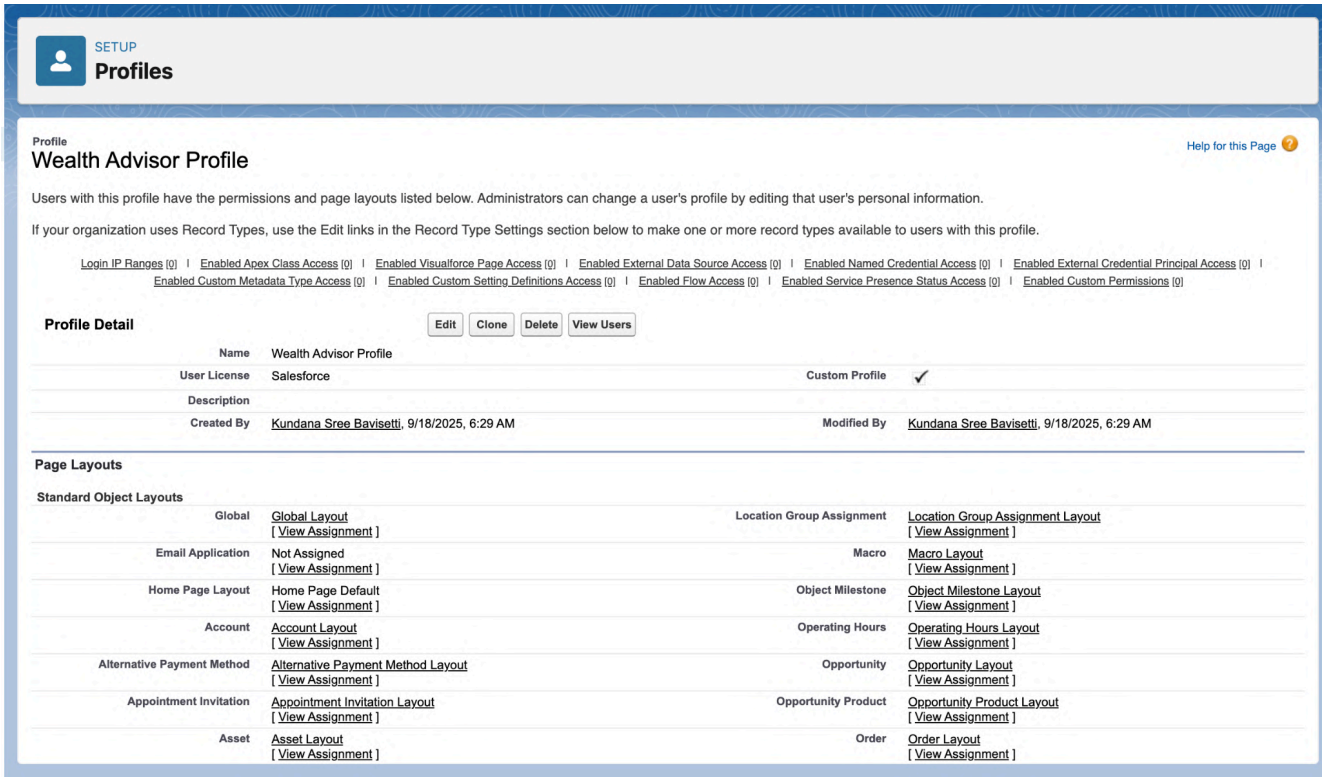
3. Fiscal Year Setup

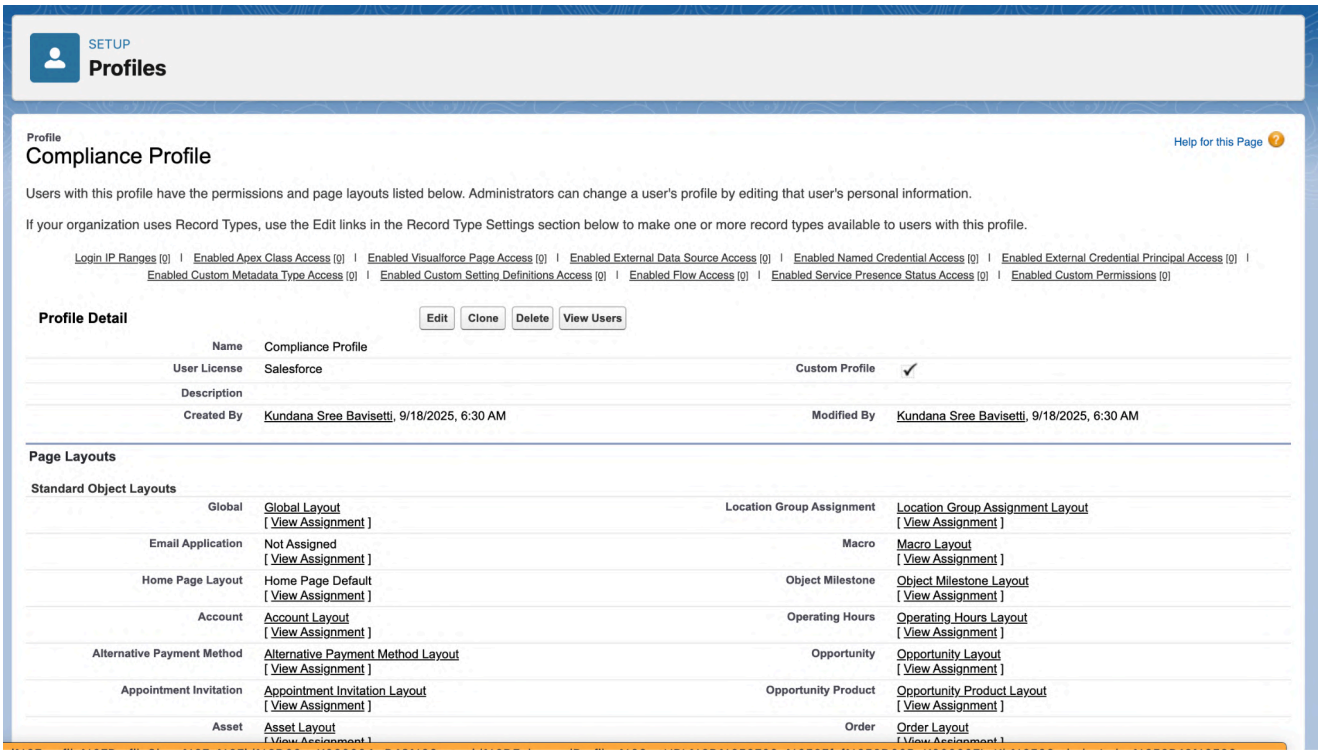


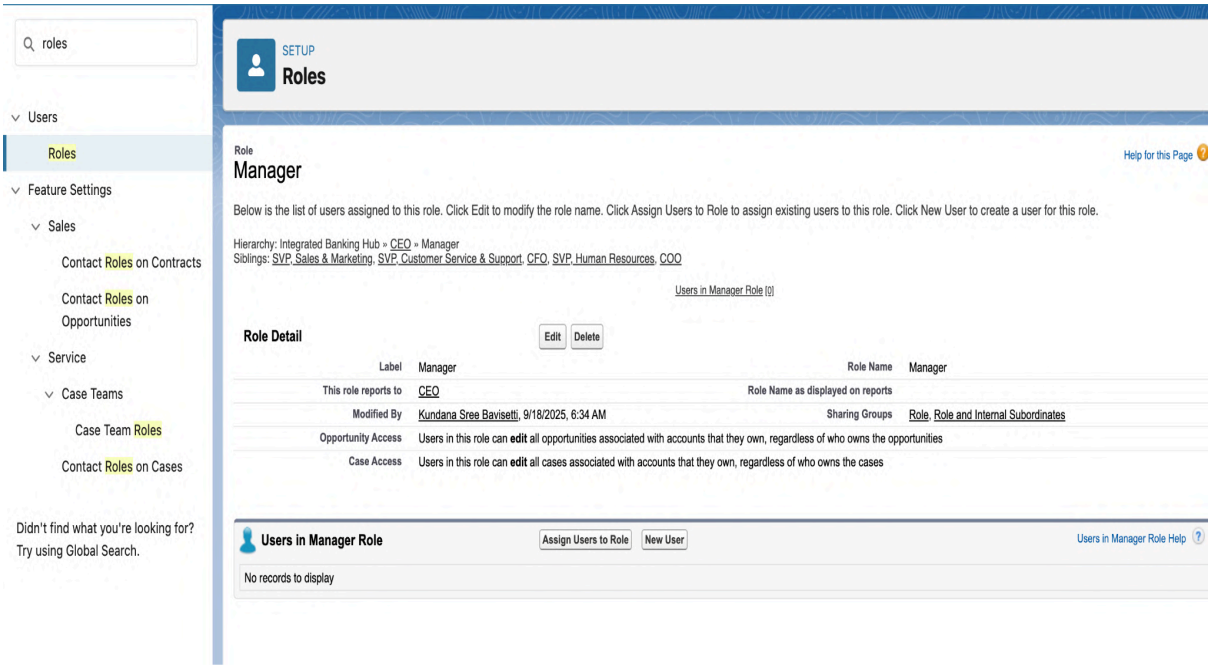
I have set the fiscal year for our organization to a **Standard Fiscal Year** starting in **April**. This is a best practice for financial reporting in India and will be crucial for creating accurate dashboards and reports in the later phases of the project.

4. User Setup (Profiles & Roles)

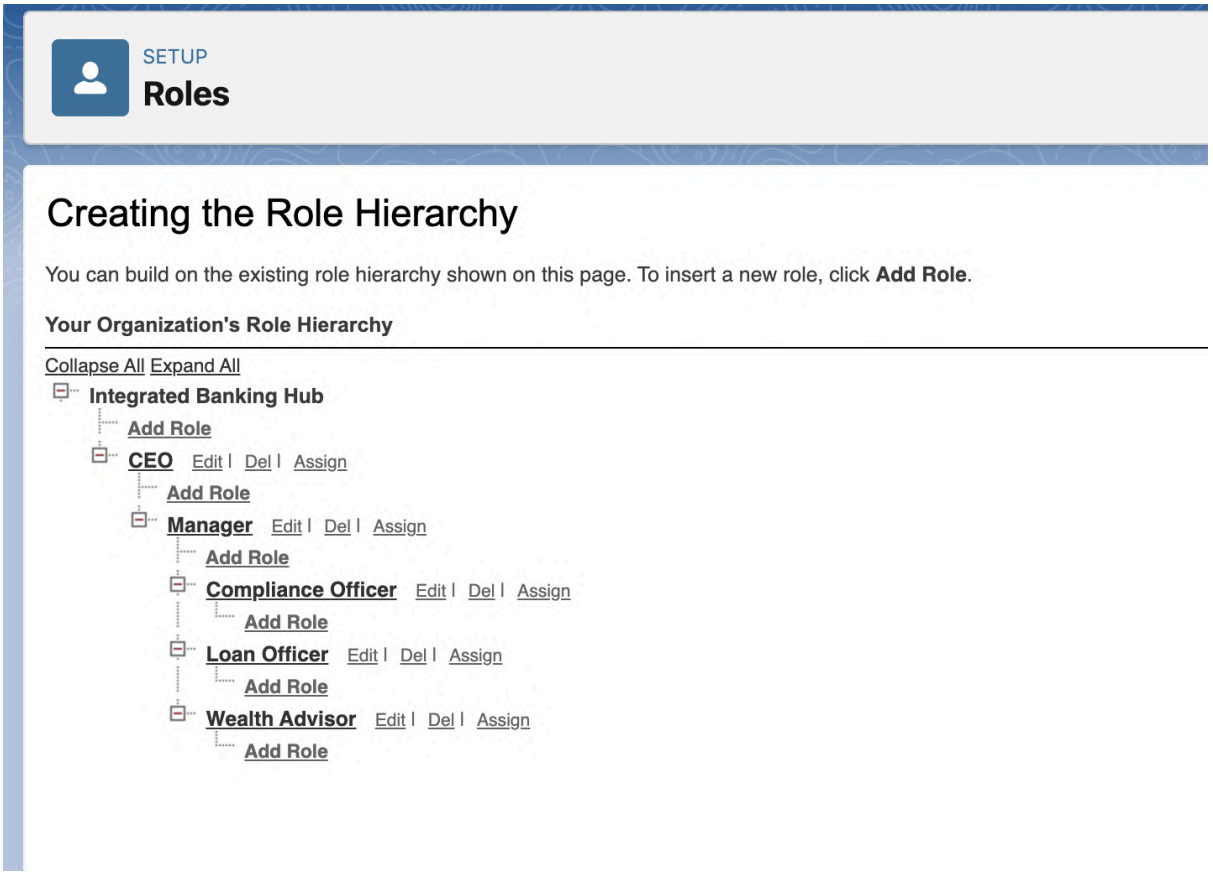



SETUP
Profiles
Profile
Wealth Advisor Profile
Help for this Page
Users with this profile have the permissions and page layouts listed below. Administrators can change a user's profile by editing that user's personal information.
If your organization uses Record Types, use the Edit links in the Record Type Settings section below to make one or more record types available to users with this profile.
Login IP Ranges [0] | Enabled Apex Class Access [0] | Enabled Visualforce Page Access [0] | Enabled External Data Source Access [0] | Enabled Named Credential Access [0] | Enabled External Credential Principal Access [0] | Enabled Custom Metadata Type Access [0] | Enabled Custom Setting Definitions Access [0] | Enabled Flow Access [0] | Enabled Service Presence Status Access [0] | Enabled Custom Permissions [0]
Profile Detail
Edit Clone Delete View Users
Name Wealth Advisor Profile
User License Salesforce
Custom Profile ✓
Description
Created By Kundana Sree Bavisetti, 9/18/2025, 6:29 AM
Modified By Kundana Sree Bavisetti, 9/18/2025, 6:29 AM
Page Layouts
Standard Object Layouts
Global Global Layout [View Assignment]
Email Application Not Assigned [View Assignment]
Home Page Layout Home Page Default [View Assignment]
Account Account Layout [View Assignment]
Alternative Payment Method Alternative Payment Method Layout [View Assignment]
Appointment Invitation Appointment Invitation Layout [View Assignment]
Asset Asset Layout [View Assignment]
Location Group Assignment Location Group Assignment Layout [View Assignment]
Macro Macro Layout [View Assignment]
Object Milestone Object Milestone Layout [View Assignment]
Operating Hours Operating Hours Layout [View Assignment]
Opportunity Opportunity Layout [View Assignment]
Opportunity Product Opportunity Product Layout [View Assignment]
Order Order Layout [View Assignment]


SETUP
Profiles
Profile
Compliance Profile
Help for this Page
Users with this profile have the permissions and page layouts listed below. Administrators can change a user's profile by editing that user's personal information.
If your organization uses Record Types, use the Edit links in the Record Type Settings section below to make one or more record types available to users with this profile.
Login IP Ranges [0] | Enabled Apex Class Access [0] | Enabled Visualforce Page Access [0] | Enabled External Data Source Access [0] | Enabled Named Credential Access [0] | Enabled External Credential Principal Access [0] | Enabled Custom Metadata Type Access [0] | Enabled Custom Setting Definitions Access [0] | Enabled Flow Access [0] | Enabled Service Presence Status Access [0] | Enabled Custom Permissions [0]
Profile Detail
Edit Clone Delete View Users
Name Compliance Profile
User License Salesforce
Custom Profile ✓
Description
Created By Kundana Sree Bavisetti, 9/18/2025, 6:30 AM
Modified By Kundana Sree Bavisetti, 9/18/2025, 6:30 AM
Page Layouts
Standard Object Layouts
Global Global Layout [View Assignment]
Email Application Not Assigned [View Assignment]
Home Page Layout Home Page Default [View Assignment]
Account Account Layout [View Assignment]
Alternative Payment Method Alternative Payment Method Layout [View Assignment]
Appointment Invitation Appointment Invitation Layout [View Assignment]
Asset Asset Layout [View Assignment]
Location Group Assignment Location Group Assignment Layout [View Assignment]
Macro Macro Layout [View Assignment]
Object Milestone Object Milestone Layout [View Assignment]
Operating Hours Operating Hours Layout [View Assignment]
Opportunity Opportunity Layout [View Assignment]
Opportunity Product Opportunity Product Layout [View Assignment]
Order Order Layout [View Assignment]

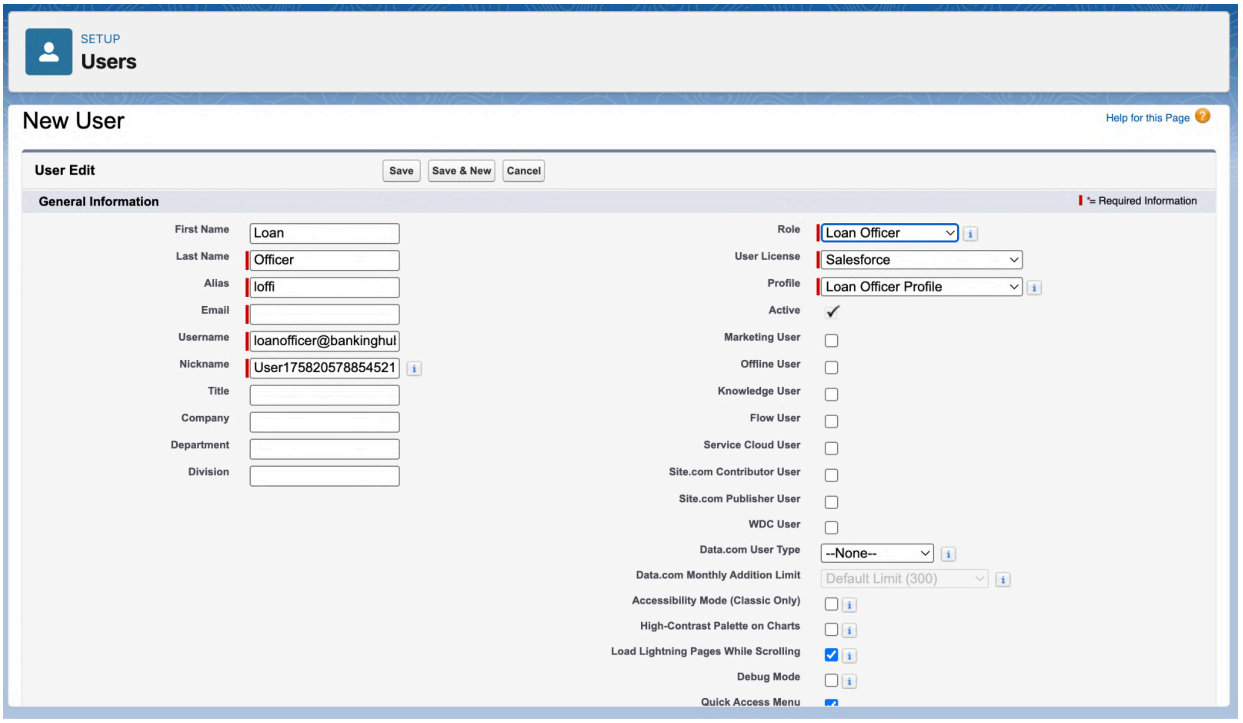


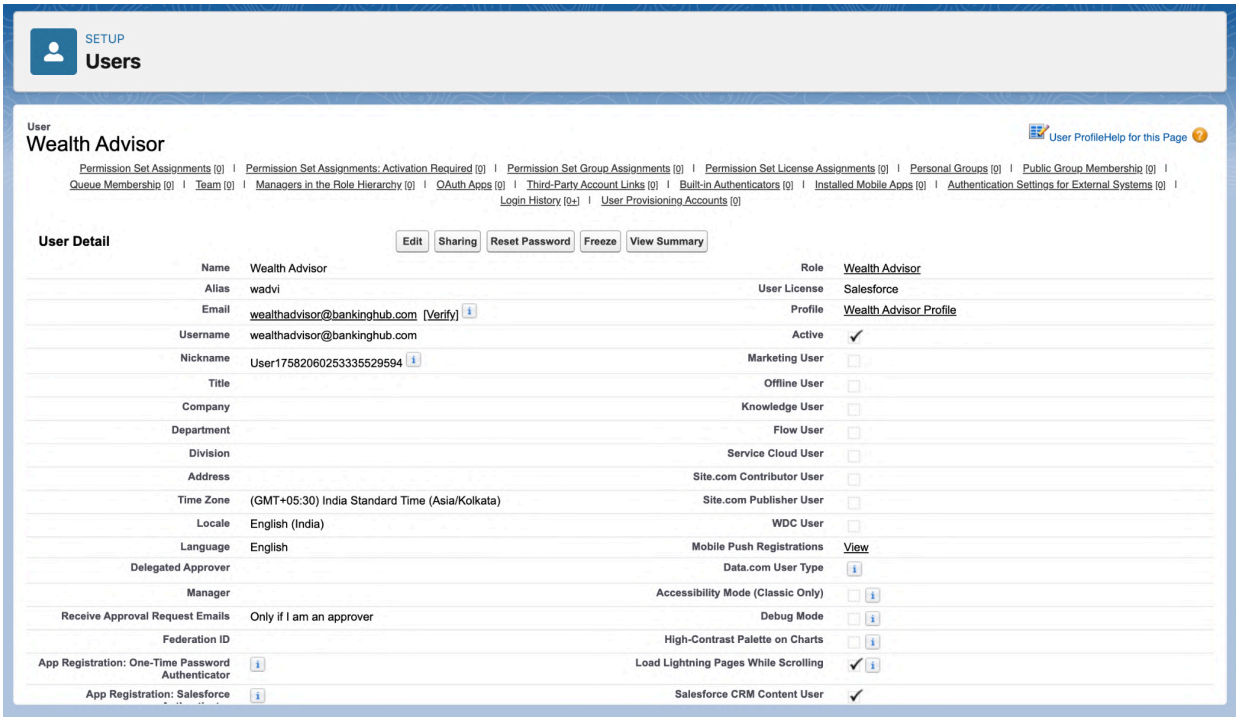
The Role Hierarchy confirms the organizational structure, placing the 'Manager' role above the specialized roles. This allows for controlled data visibility and reporting alignment."

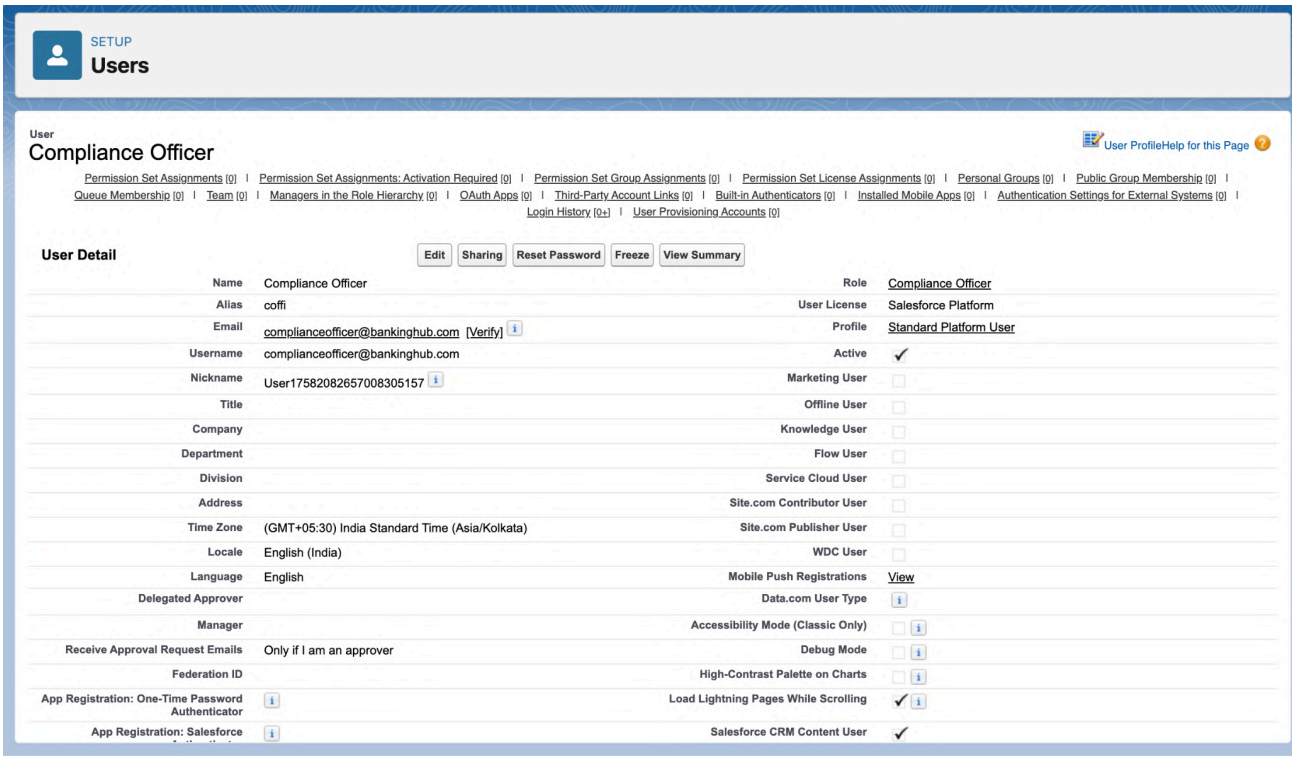


To enforce our project's security model, I have created three custom profiles: **Loan Officer Profile**, **Wealth Advisor Profile**, and **Compliance Profile**. I have also set up a clean and organized role hierarchy that reflects our project's structure, with the “Manager” role overseeing the other three specialized roles. This ensures a clear chain of command and data visibility.

User Creation

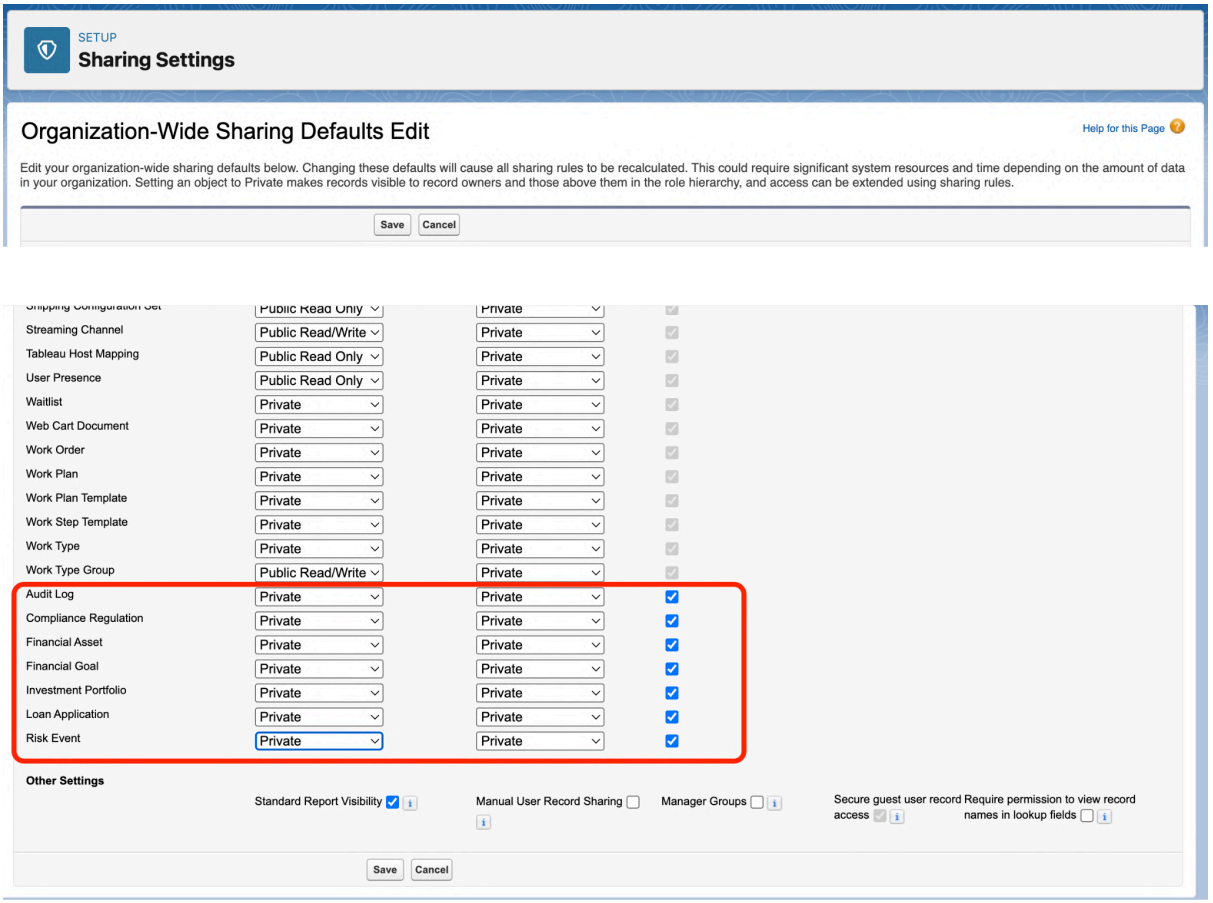






To test our application properly, I have created a user for each of the core roles in our project. Each user has been assigned the appropriate profile and role, ensuring they have the correct permissions to access the data and features relevant to their job.

5. OWD (Org-Wide Defaults) & Sharing Rules



In **Setup**, I navigated to **Sharing Settings** and clicked **Edit**.
I set the Default Internal Access for all custom objects (e.g., Loan Application, Investment Portfolio, Compliance Regulation, Collateral,Repayment Schedule, Financial Asset, Financial Goal,Risk Event, Audit Log) to **Private**.

Setting Org-Wide Defaults (OWD) to **Private** for all financial objects (e.g., Loan Application, Investment Portfolio) is a core security best practice. This 'restrictive first' approach guarantees data is only shared through controlled mechanisms like Roles or Sharing Rules, adhering to strict financial compliance rules.

Phase 3: Data Modeling & Relationships

1. Custom Objects

The following are the custom objects in each module that i have created

Loan Module Objects:

- Loan Application
- Collateral
- Repayment Schedule

Wealth Module Objects:

- Investment Portfolio
- Financial Asset
- Financial Goal

Compliance Module Objects:

- Compliance Regulation
- Risk Event
- Audit Log

2. Custom Fields

After creating the custom objects, I added the necessary fields to store specific data. As shown in the below screenshots.

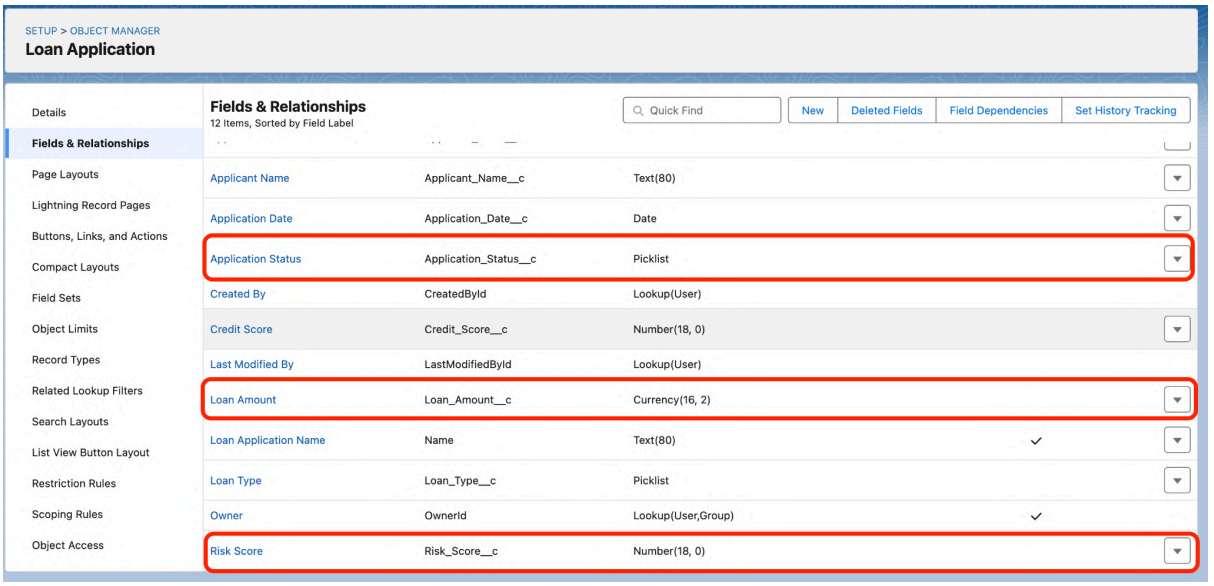
Loan Application Module:

Use Case: Stores all loan request details (type, amount, risk score) and drives the approval workflow.

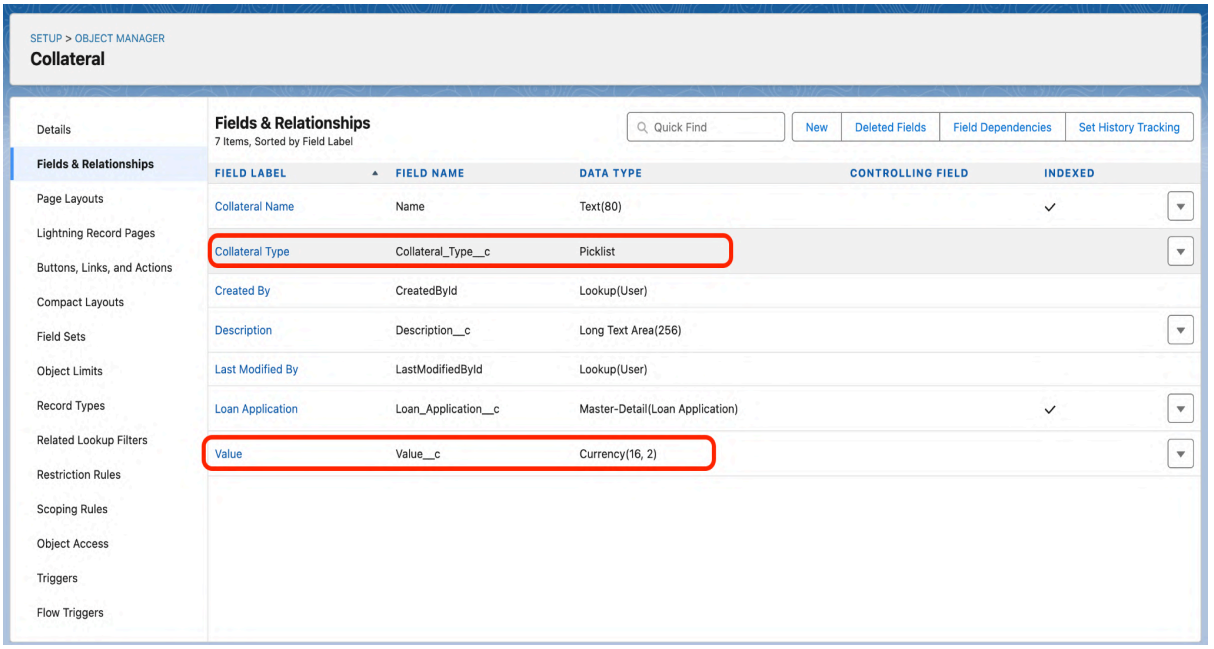
Relationship : Master-Detail to Account (Client).

- **Loan Application Object fields:**

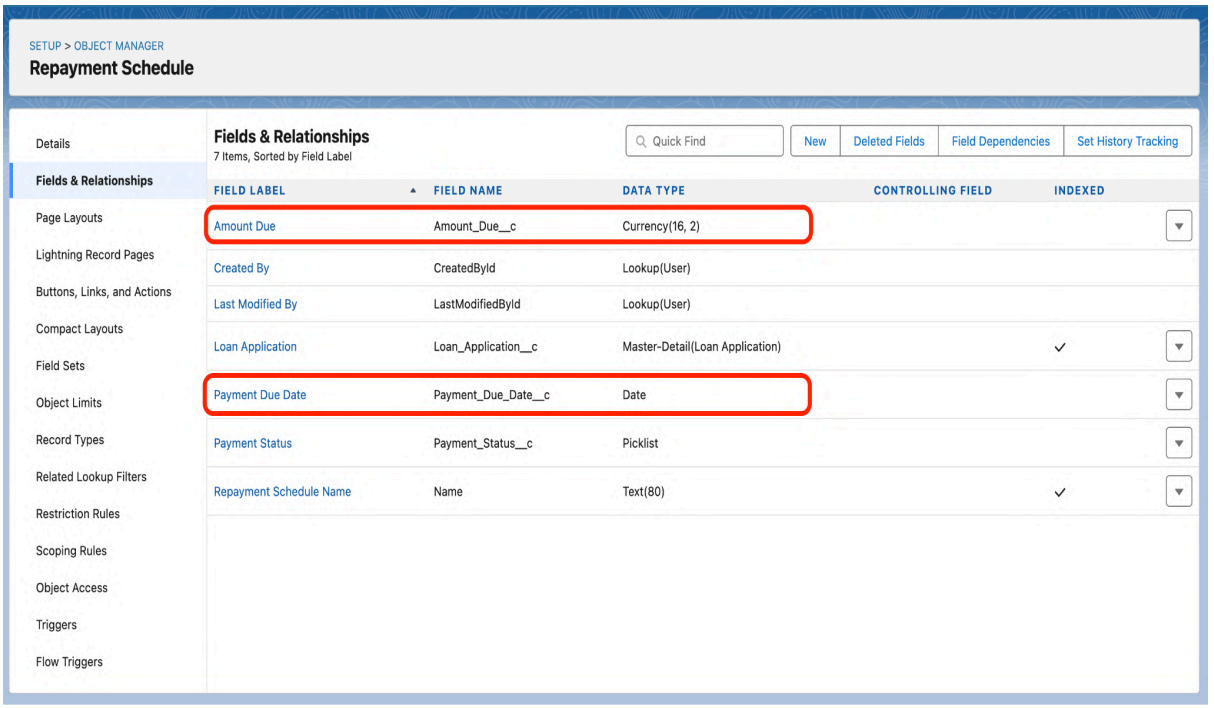
  The custom fields were engineered to capture all necessary business data. For instance, the Loan Application requires Risk_Score__c, and the Investment Portfolio requires the calculated Total_Value__c. This structure ensures data quality and completeness for subsequent automations.



- **Collateral Object fields:**

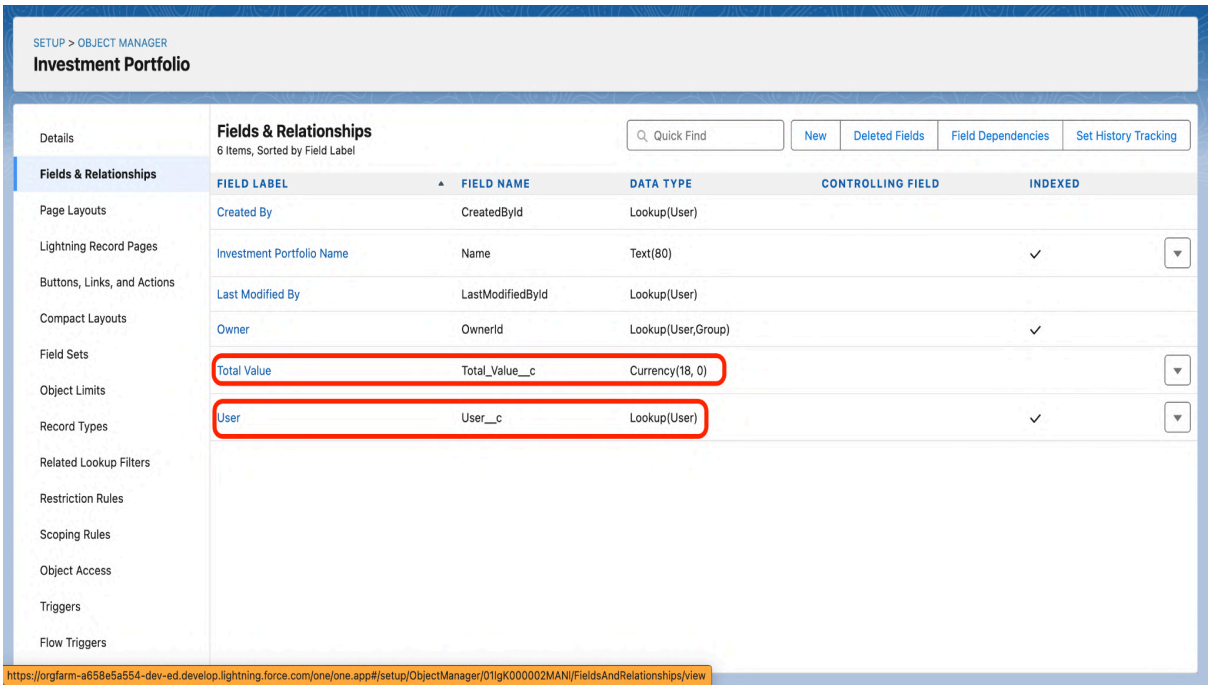


- **Repayment Schedule Object fields:**

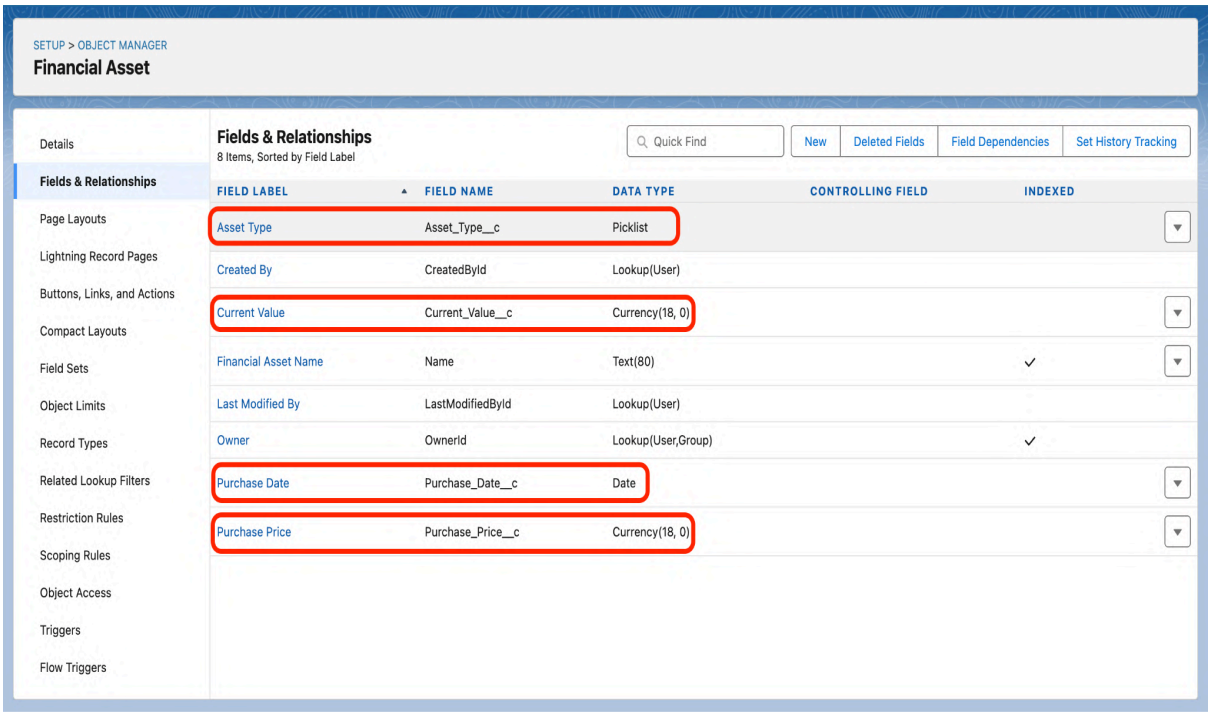


Wealth Module:

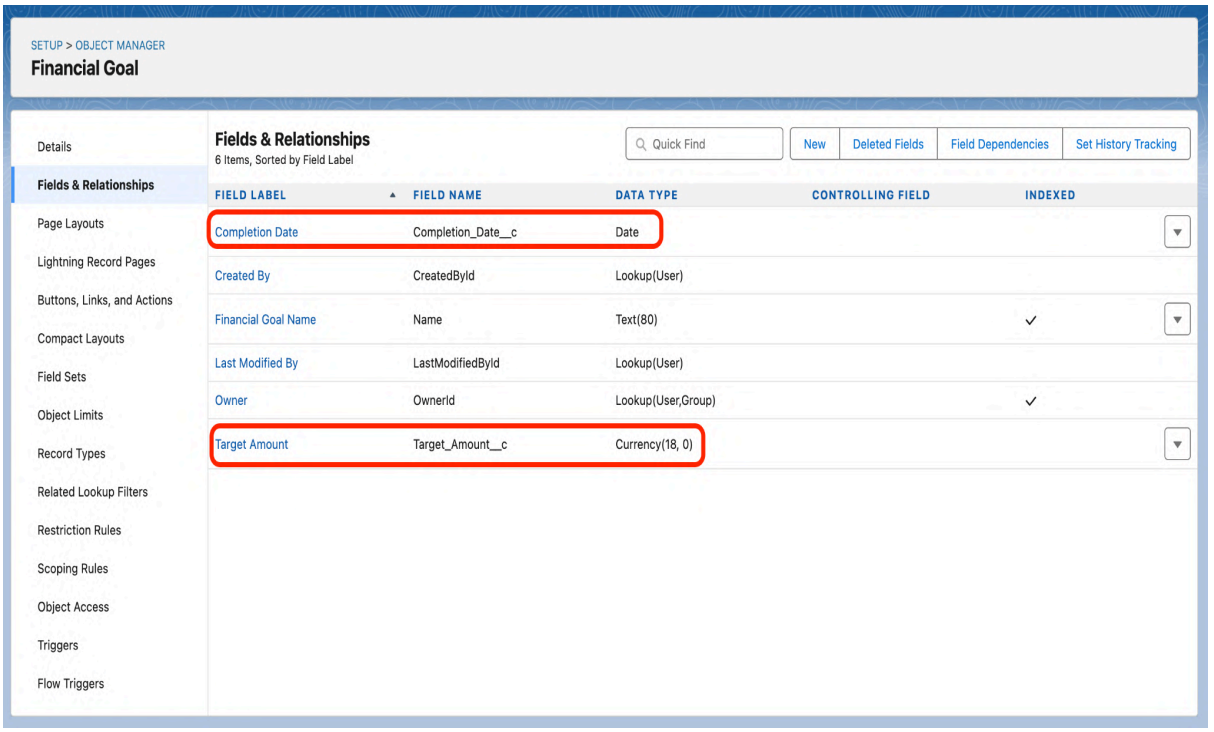
- **Investment Portfolio Object fields:**



- **Financial Asset Object fields:**

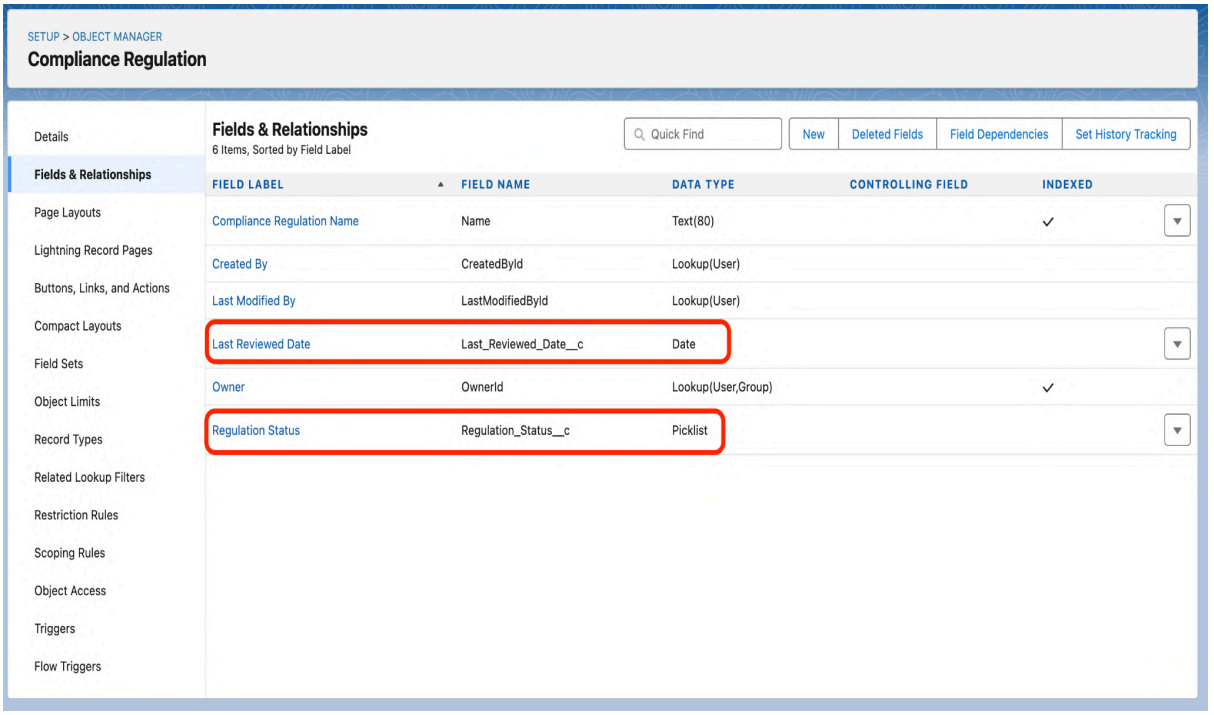


- **Financial Goal Object fields:**

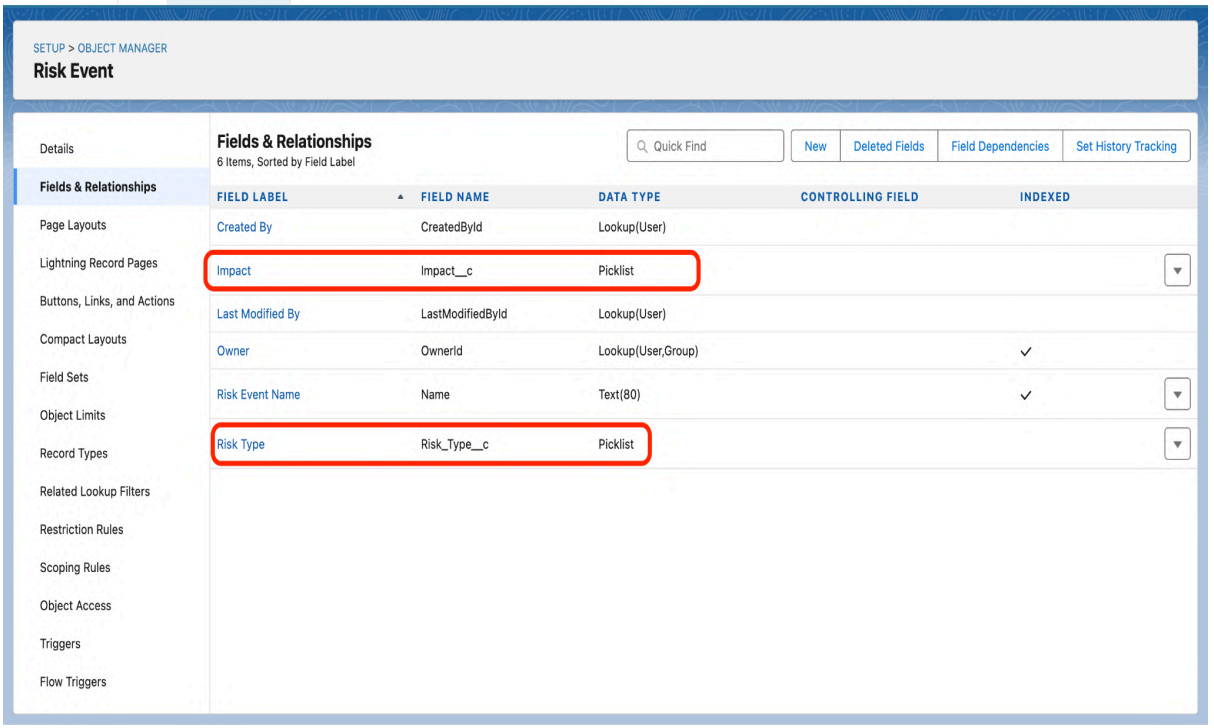


Compliance Module:

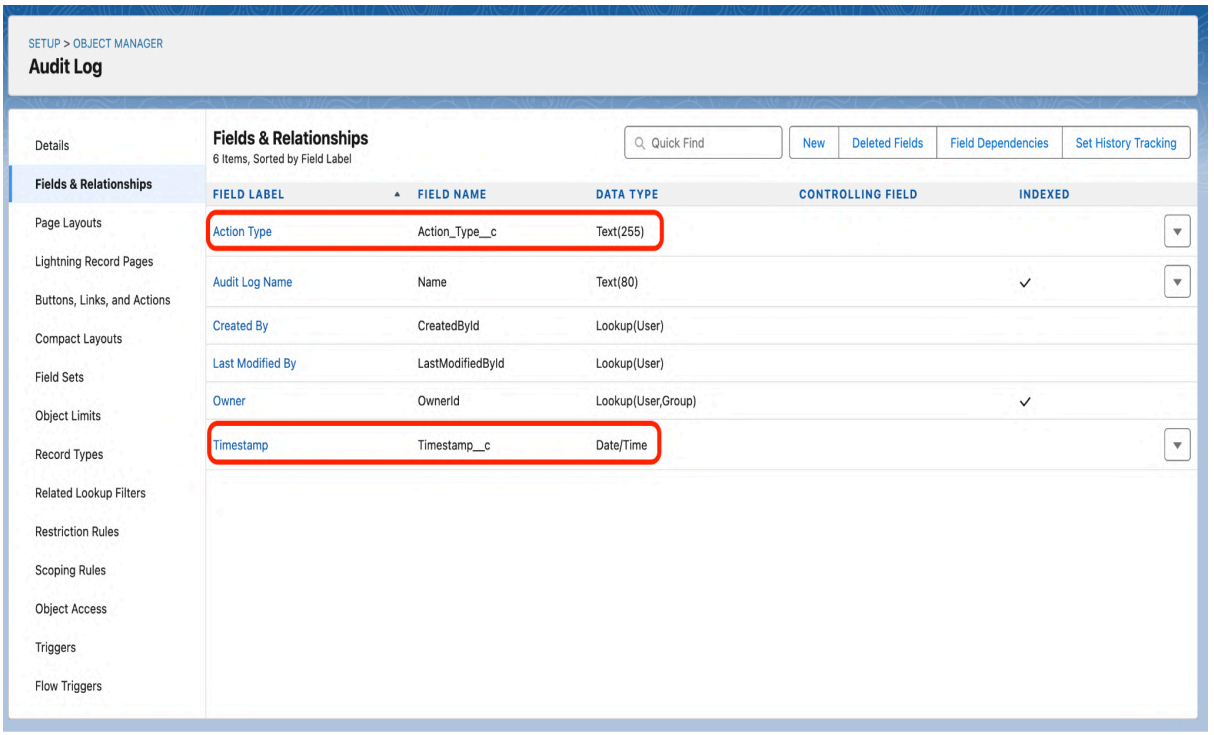
- **Compliance Regulation Object fields:**



- **Risk Event Object fields:**



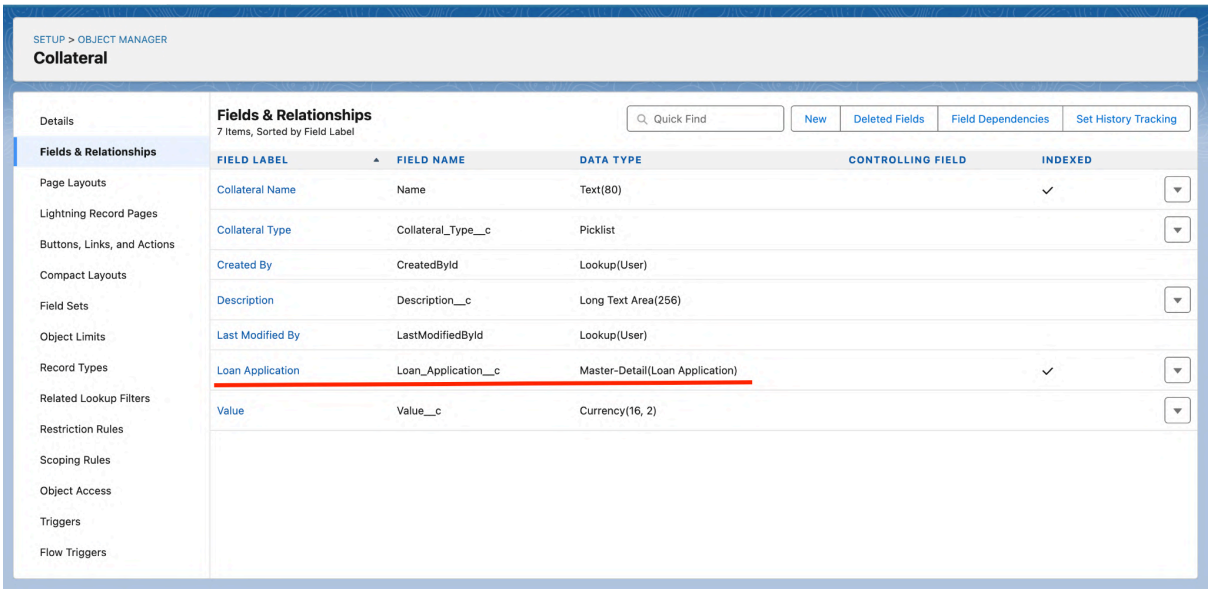
- **Audit Log Object fields:**



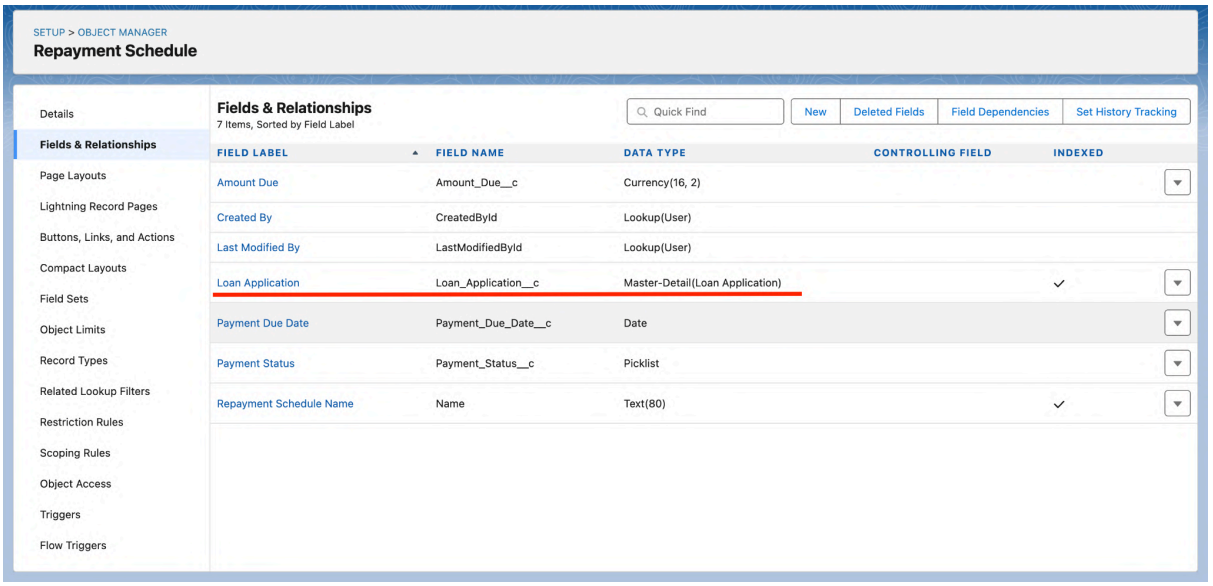
3. Relationships

To connect our different data modules, I established a series of relationships.
The use of **Master-Detail Relationships** guarantees the integrity of our financial data. A Financial Asset is dependent on its parent Investment Portfolio, ensuring that records cannot exist in isolation, which is vital for accurate roll-up summaries and reporting.

Use Case: Stores all loan request details (type, amount, risk score) and drives the approval workflow.

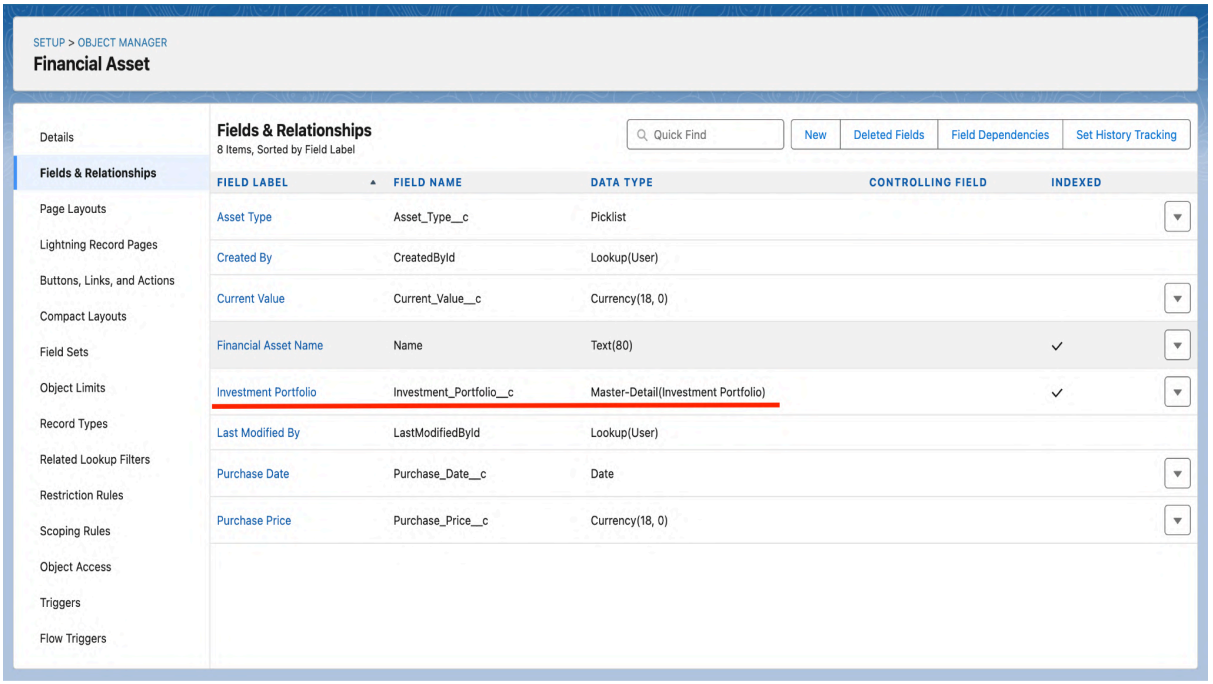


I created a **Master-Detail Relationship** between the **Loan Application** object and the **Collateral** object. This creates a strong link, so a collateral record cannot exist without a parent loan application.

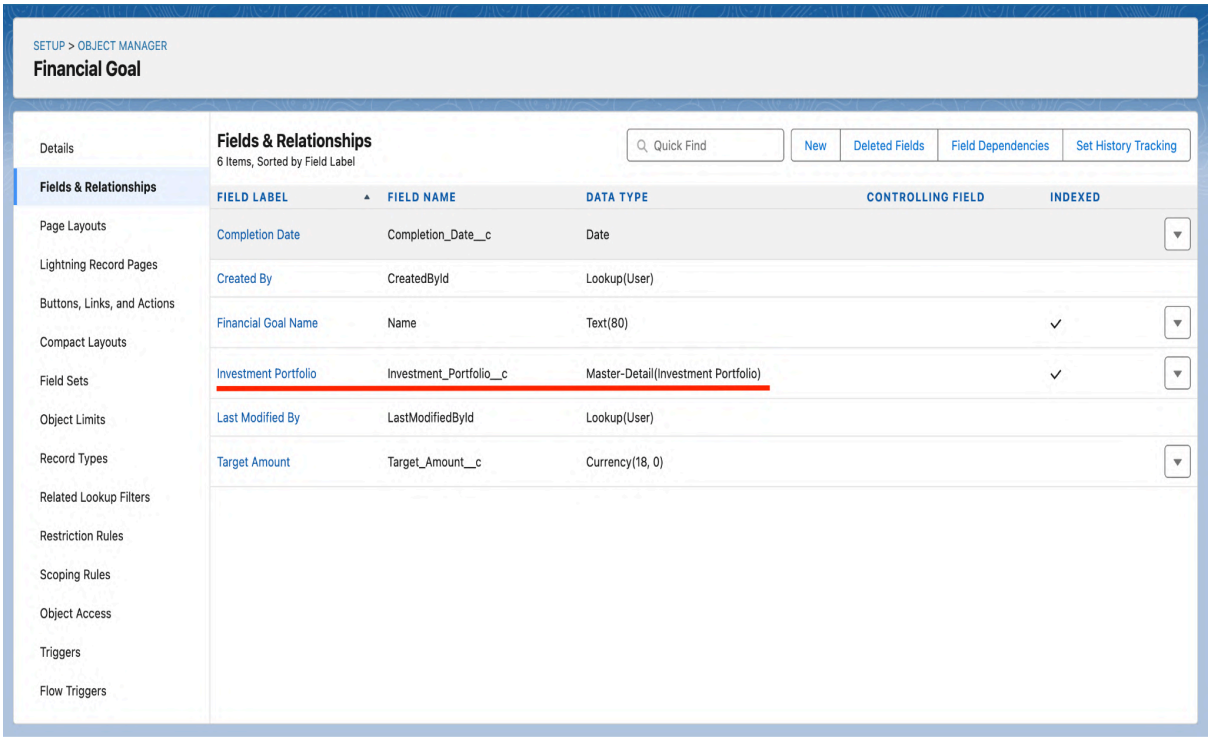


Similar to Collateral object , I created another **Master-Detail Relationship** between **loan application** and **repayment schedule** object.

Use Case : Tracks individual assets (stocks, bonds, mutual funds) within a portfolio.

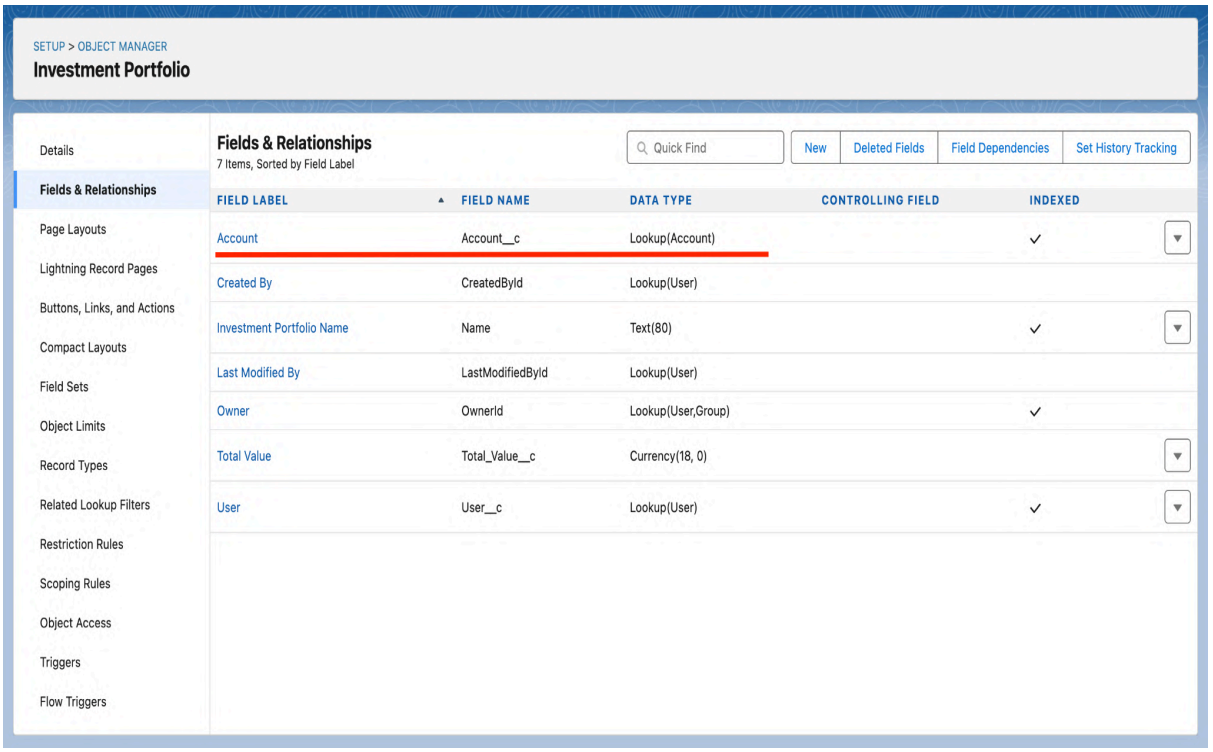


here I created **Master-Detail Relationship** for **financial asset** to **investment portfolio**



here I created **Master-Detail Relationship** for **financial goal** to **investment portfolio**

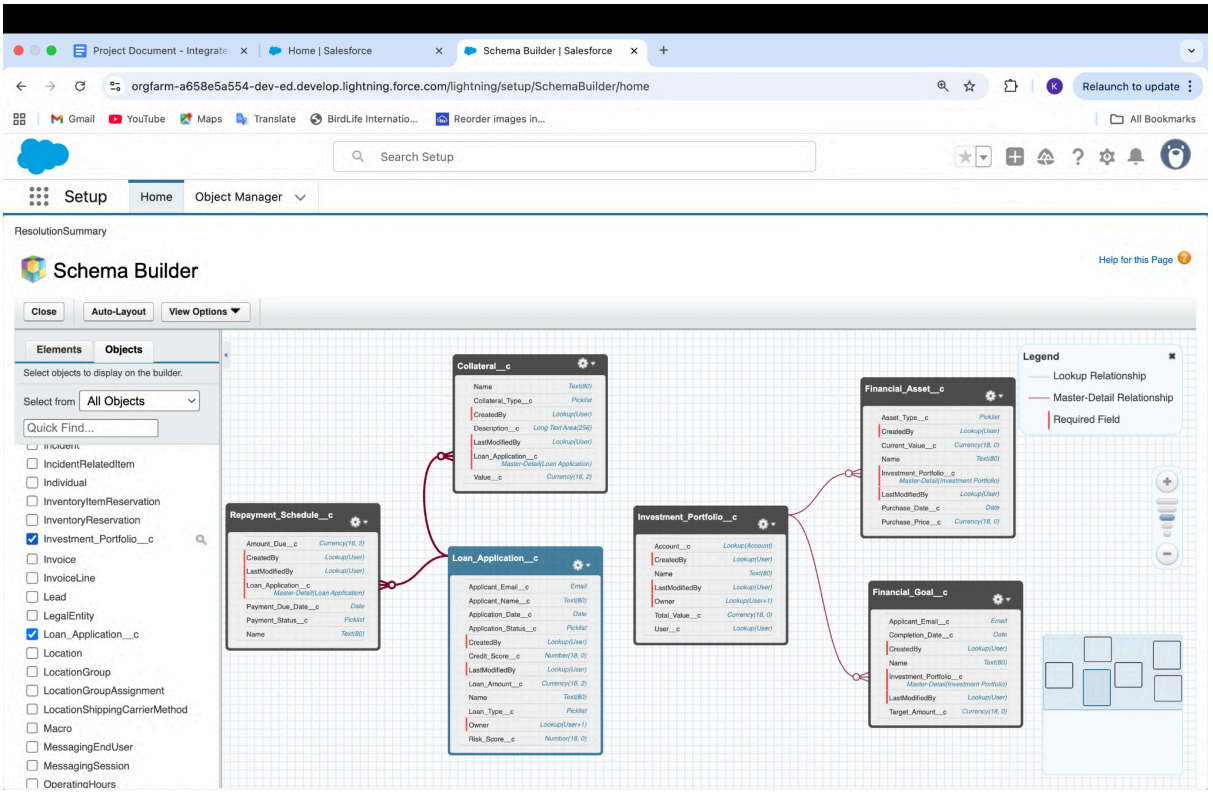
Use Case : The primary container for a client's investment strategy and calculated total value.



here I created the **lookup-relationship** for **investment portfolio** to **account**

Schema Builder :

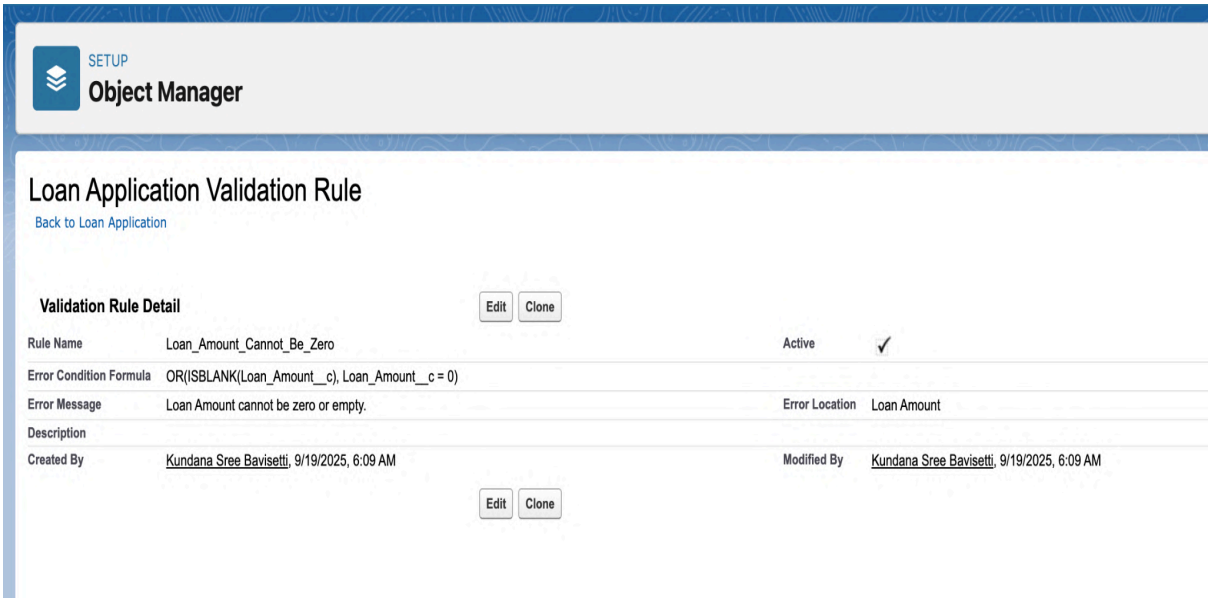
The Schema Builder visually confirms the complete data model: three independent modules (Loan, Wealth, Compliance) anchored to the standard Account object. This centralized structure is the foundation of the single source of truth (SSOT) concept for this project



Phase 4: Process Automation (Admin)

1. Validation Rules

- **Use Case:** To enforce financial data integrity by preventing users from submitting loan applications with invalid data, such as a loan amount of zero or less.
- **Implementation:** A Validation Rule on the Loan Application object prevents saving the record if Loan_Amount__c <= 0.
- **Business Impact:** Eliminates inaccurate data from entering the approval pipeline, saving time and preventing errors.

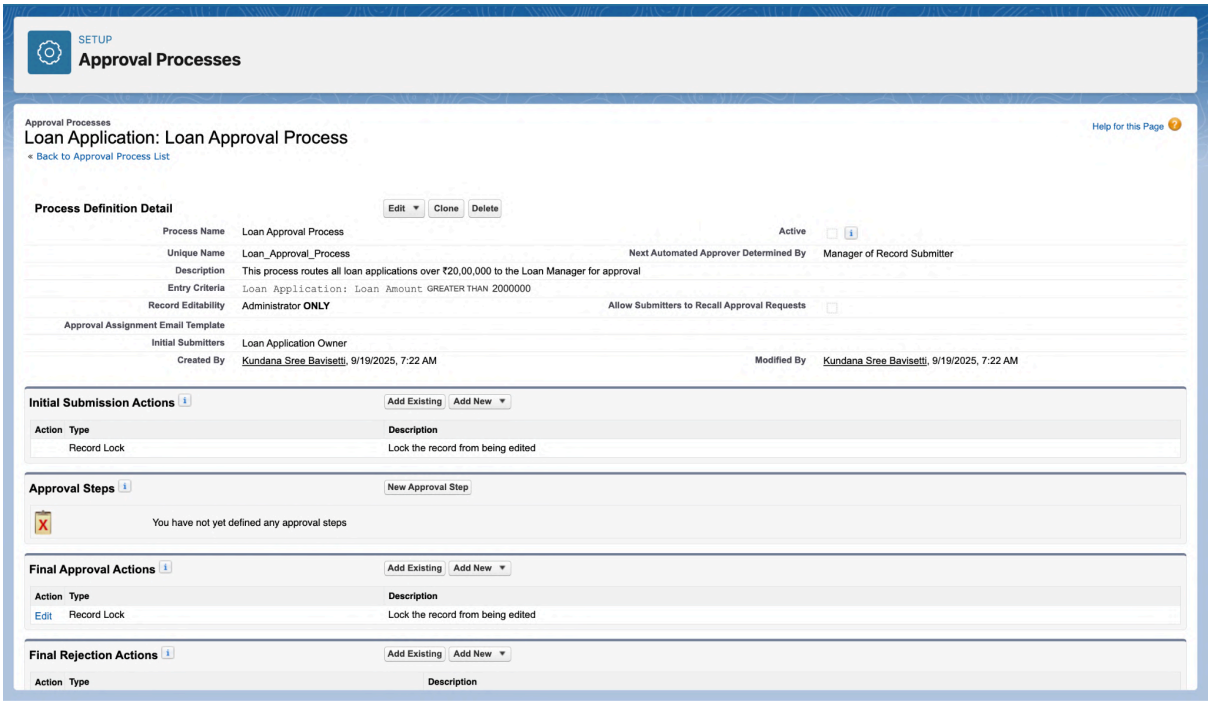


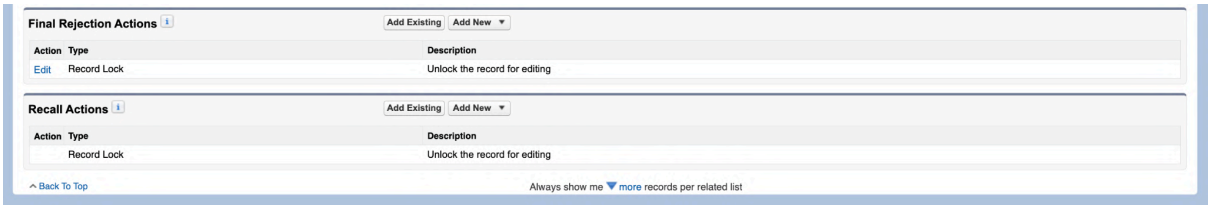
I created a validation rule on the Loan Application object. The rule's formula, OR(ISBLANK(Loan_Amount__c), Loan_Amount__c < 0), ensures that the Loan Amount field is not empty and is not a negative number.

This is a crucial step for data integrity. The rule prevents users from submitting incomplete or invalid loan applications, ensuring that our data is accurate from the moment it is entered into the system.

2. Approval Process

- **Use Case:** To enforce governance on high-value transactions. Any loan application greater than or equal to **₹20,00,000** must be automatically routed to the **Manager** profile for mandatory approval.
- **Implementation:** Standard Approval Process with an entry criteria of Loan_Amount__c >= 20000000 and a final approval action that updates the Status to 'Approved'.





I set up a multi-step approval process for the Loan Application object. The entry criteria, Loan Application: Loan Amount GREATER THAN 2000000, routes high-value loan applications to a manager. The record is locked from editing during this process.

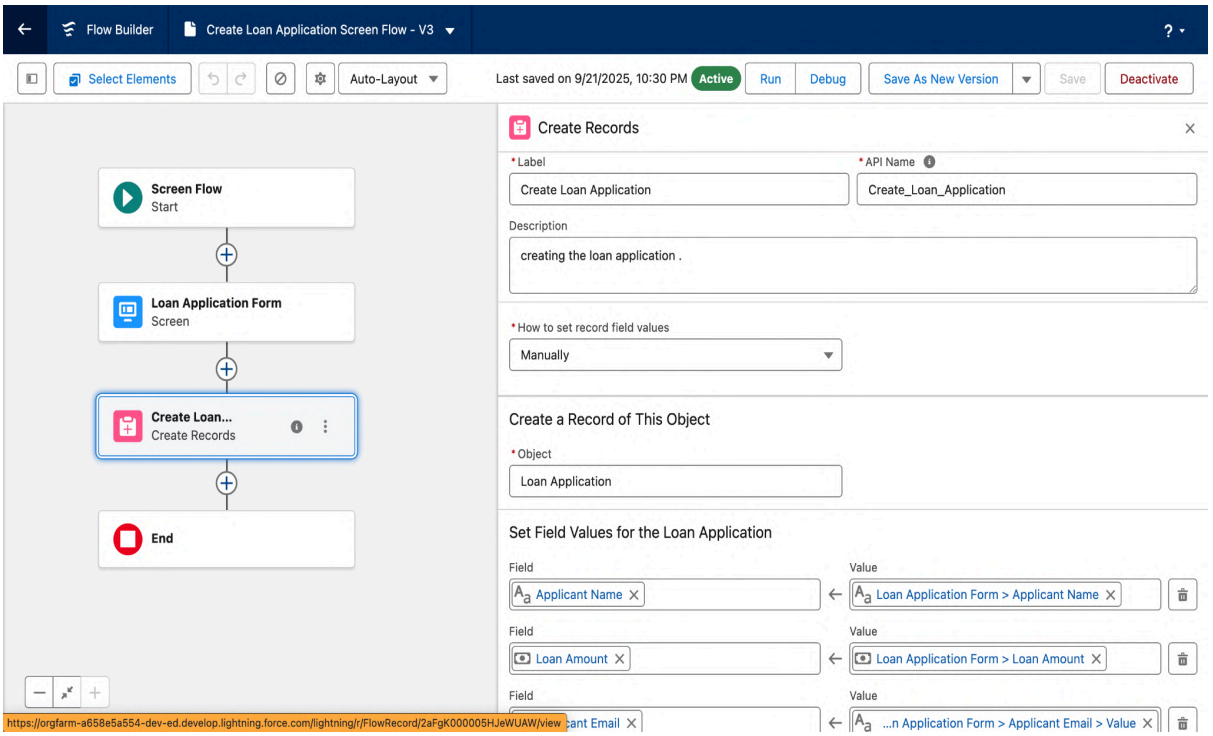
This process is essential for risk management and compliance. It automates the review of high-risk loans, ensuring that senior management signs off on them before they are approved. This reduces financial risk and provides a clear audit trail.

3. Flows

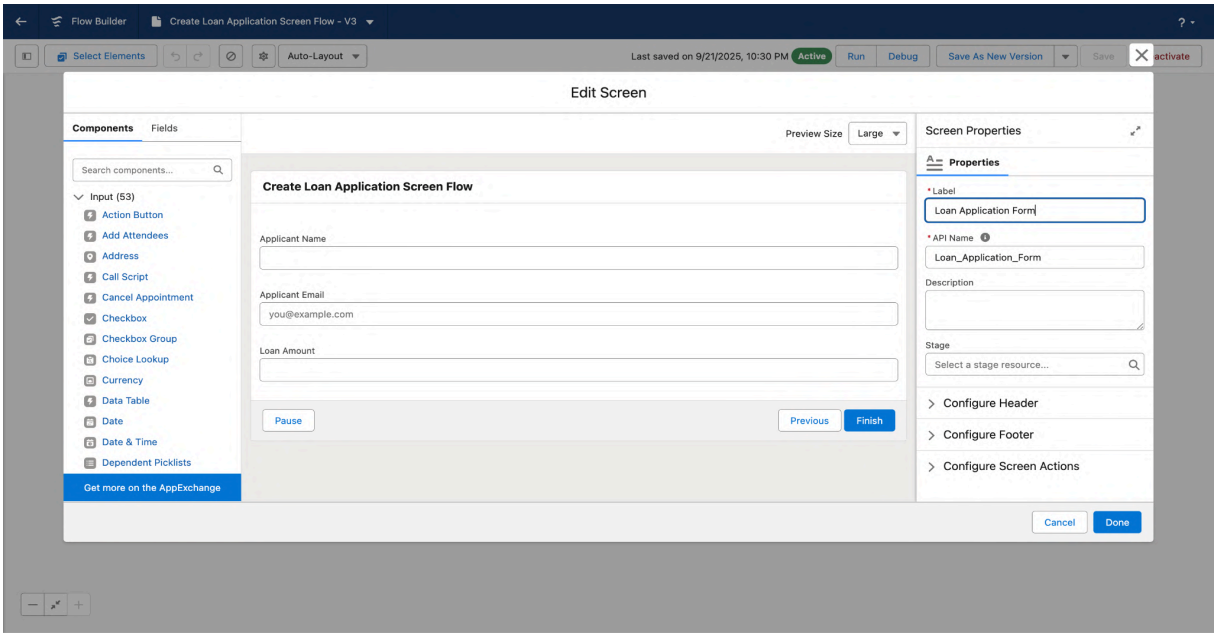
- **Use Case:** To automatically calculate and assign a **Risk Score** (e.g., 1-5) to a loan application immediately upon creation, allowing Loan Officers to triage applications quickly.

- **Implementation:** A **Record-Triggered Flow** (after-save) calculates a value for Risk_Score__c based on criteria like Credit Score and Loan Type.

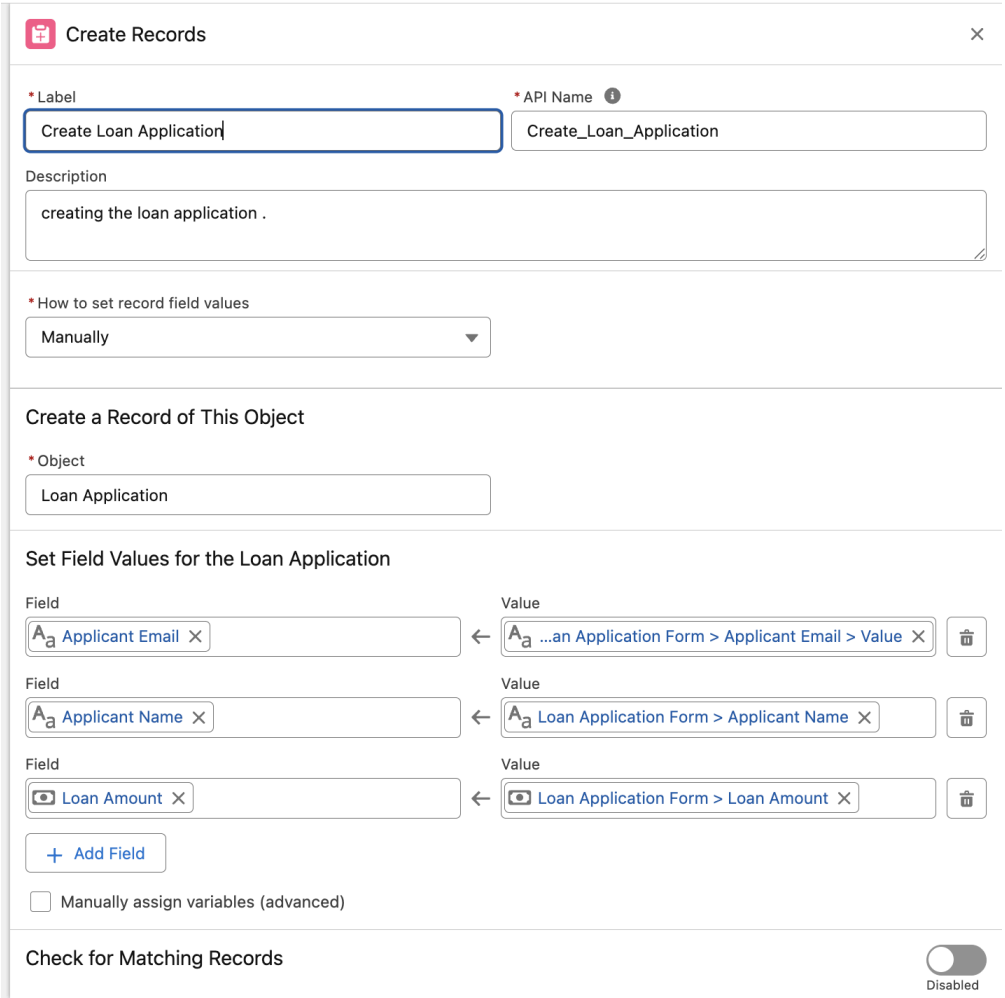
Create loan Application Screen flow :



Loan Application form flow element :



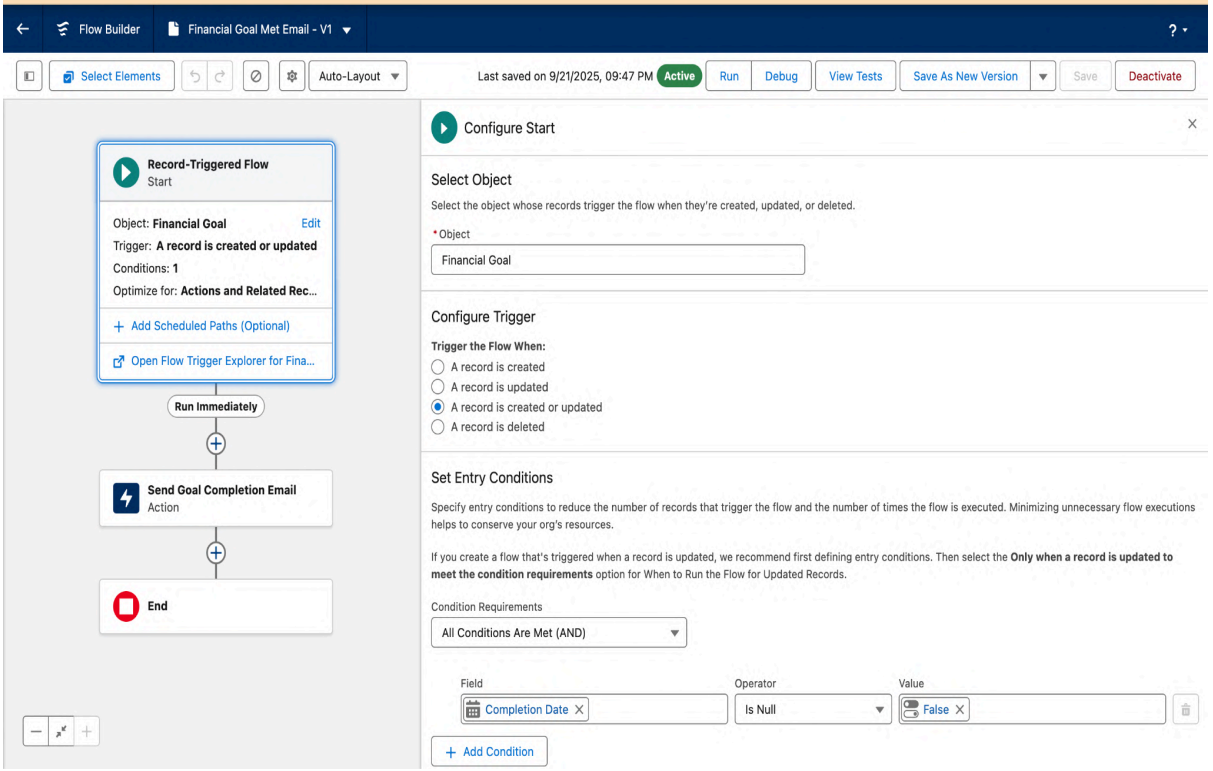
Create Loan Application flow Element :



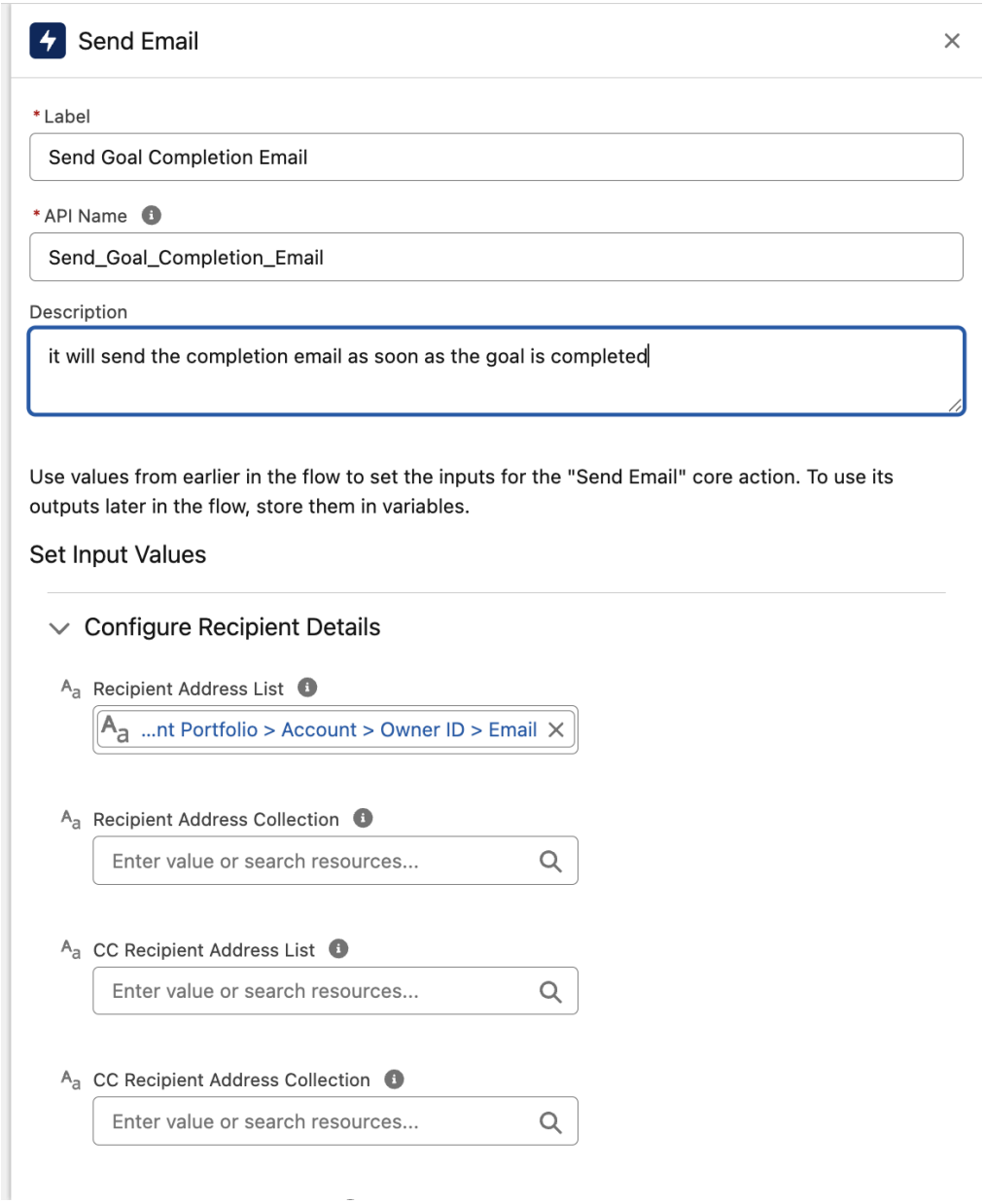
I built a Screen Flow that guides the user through creating a new Loan Application record. This flow uses three key components: Applicant Name, Loan Amount, and Applicant Email to collect data and then uses a Create Records element to save it to the database.

This flow provides a user-friendly and consistent experience for loan officers. By guiding them through a specific set of fields, it ensures all necessary information is collected correctly, reducing errors and saving time.

Financial Goal Met Email Flow :



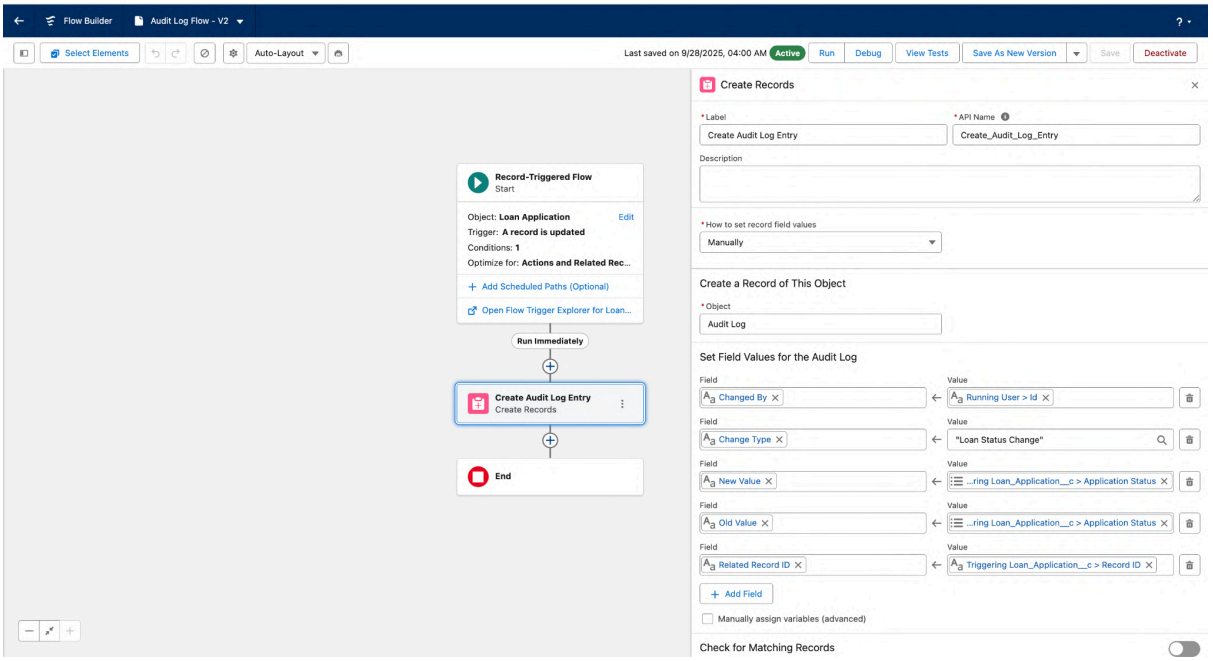
Send Goal Completion Email Flow Element :



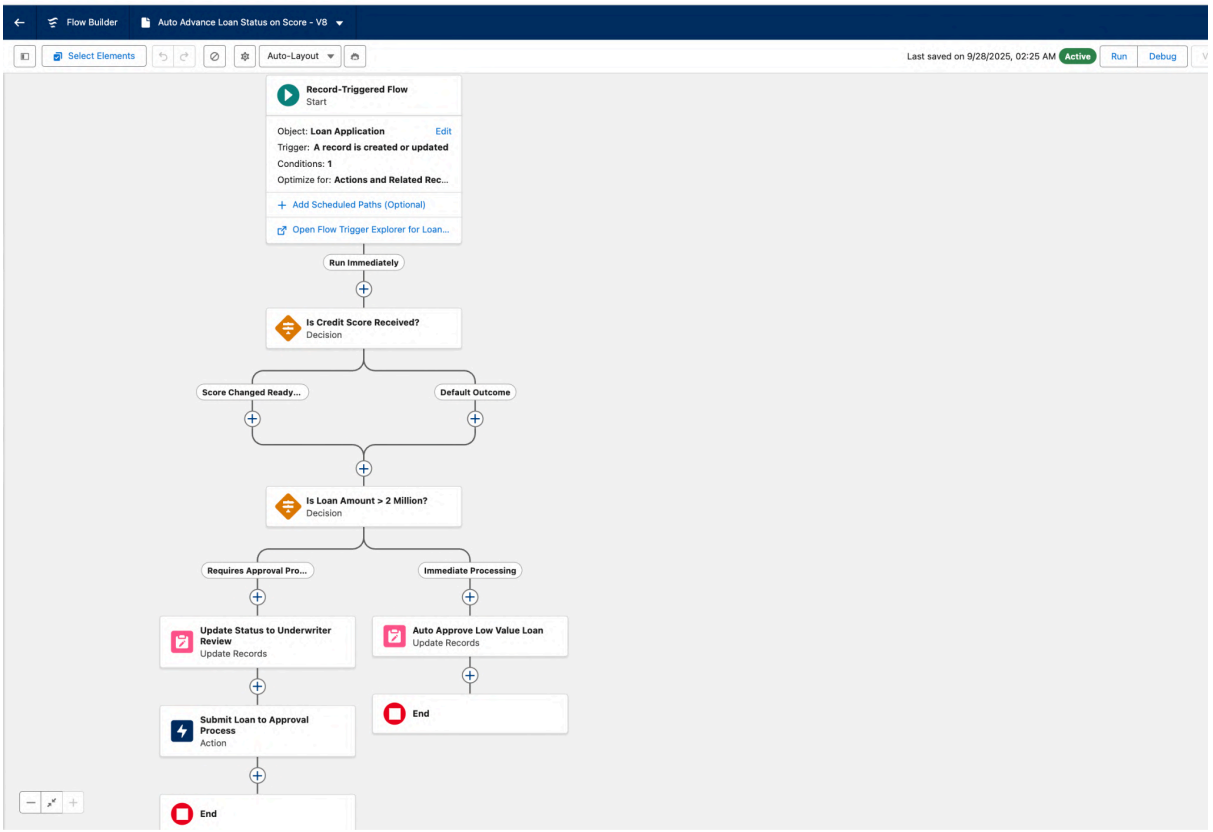
I created a Record-Triggered Flow that runs automatically when a Financial Goal record is created or updated. The flow checks if the Completion Date field is not null (IsNull False). If the condition is met, it will trigger an email alert.

This flow automates client communication, which is a key part of customer relationship management. It ensures that clients are instantly notified when they achieve a financial goal, which enhances the client experience and builds trust.

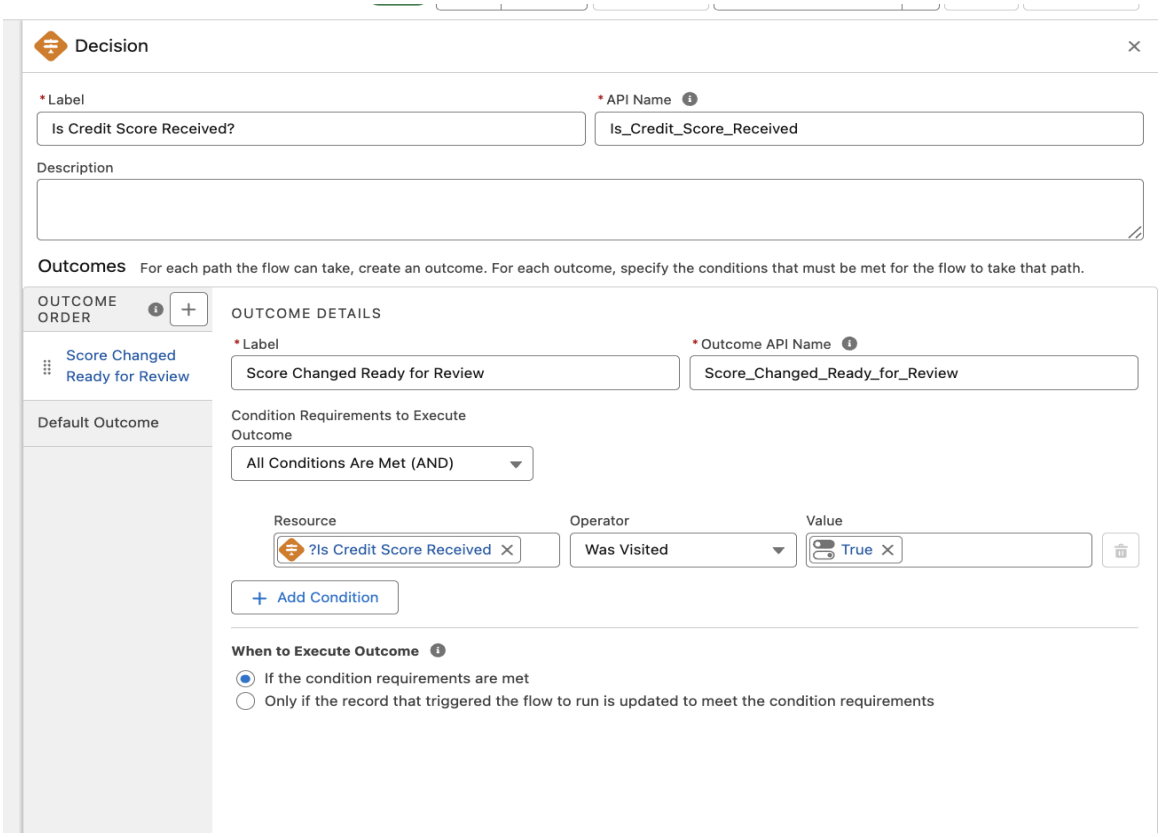
Audit Log Flow :



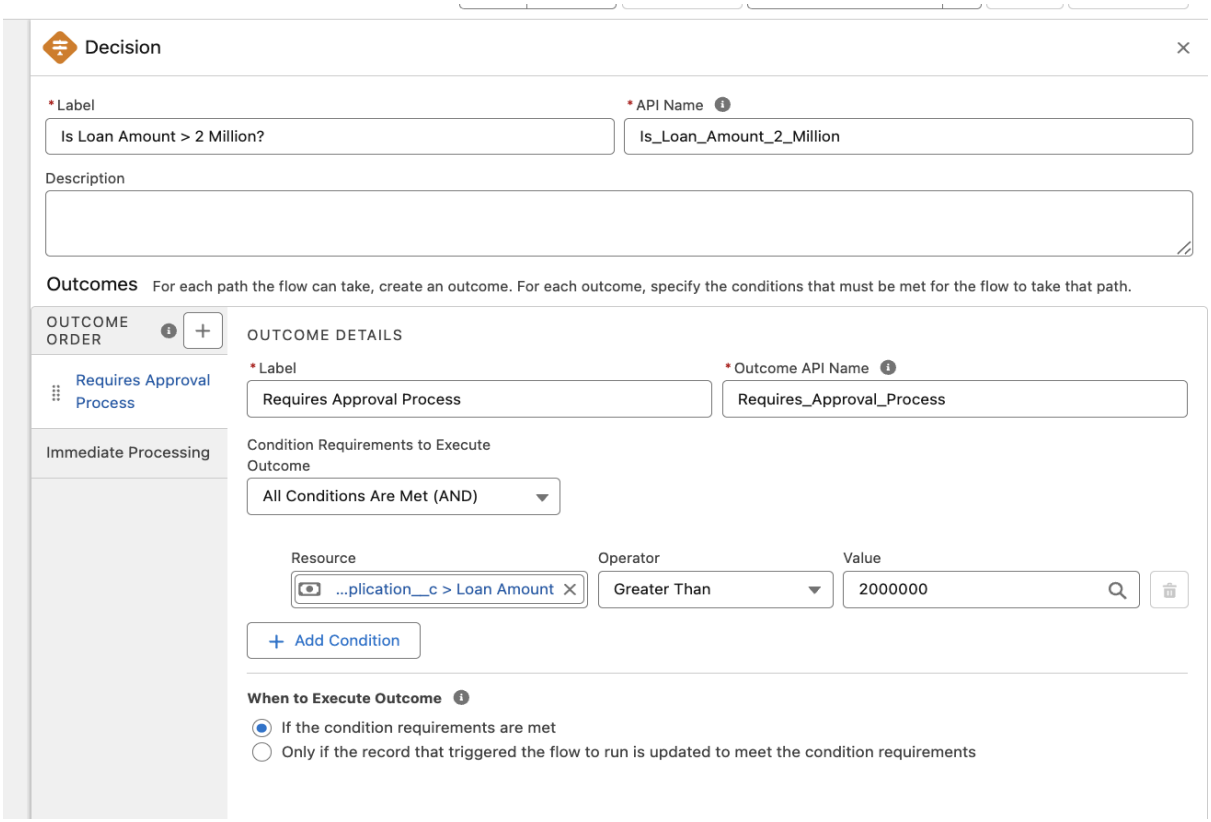
Auto Advance Loan Status on Score :



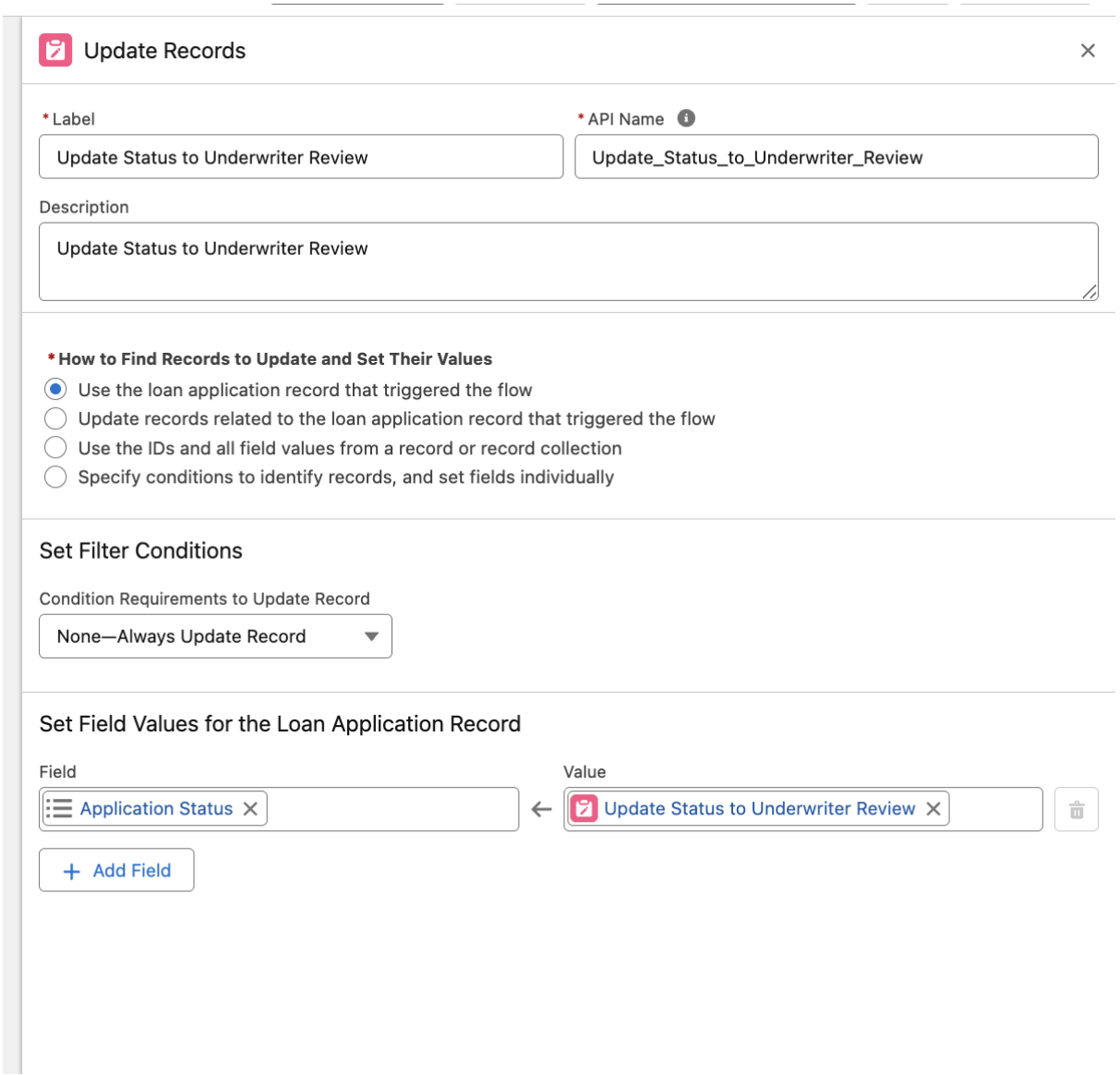
Is Credit Score Received? Element :



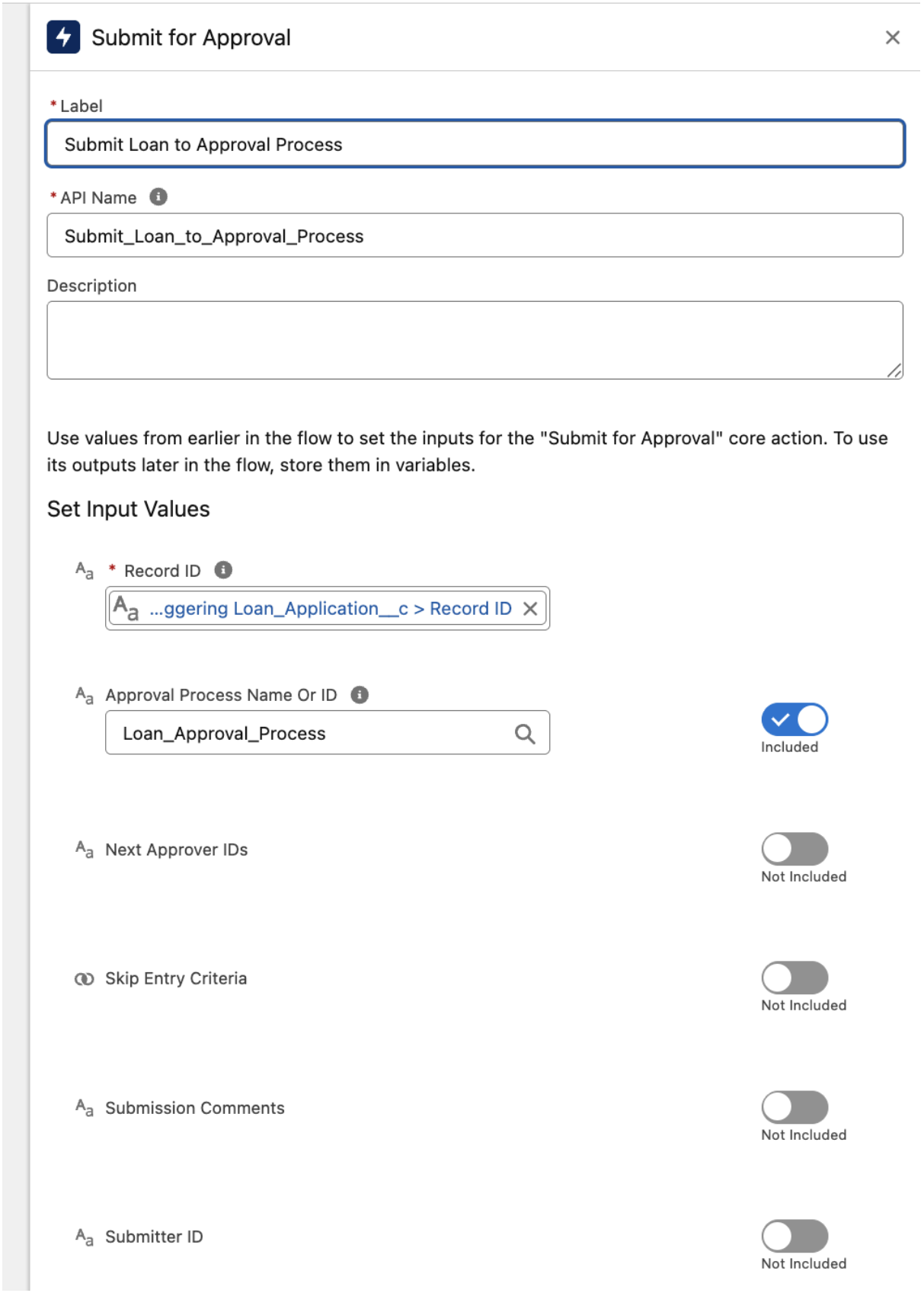
Is Loan Amount > 2 Million? Element :



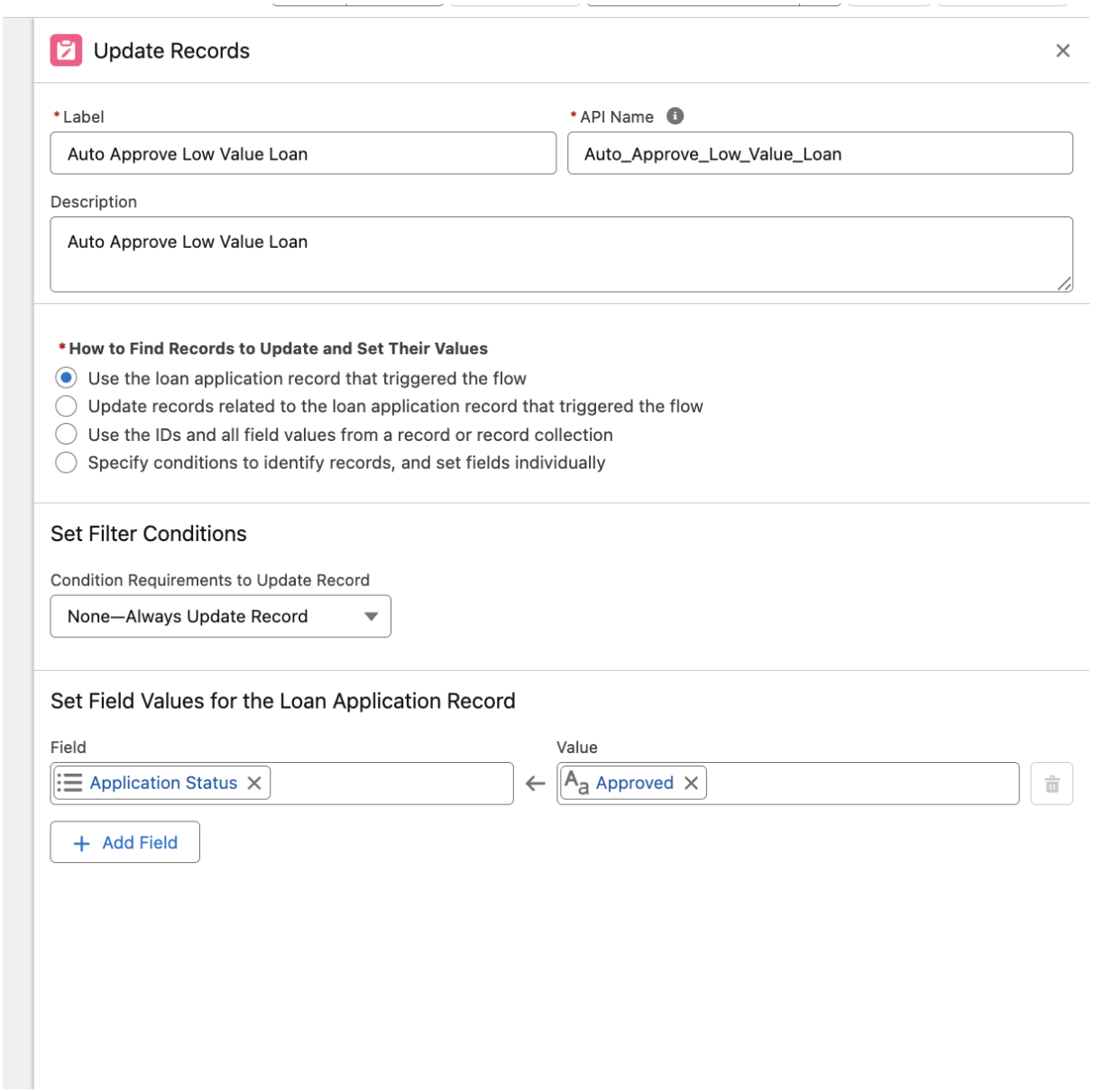
Update Status to Underwriter Review element :



Submit Loan to Approval Process element :



Auto Approve Low Value Loan element :

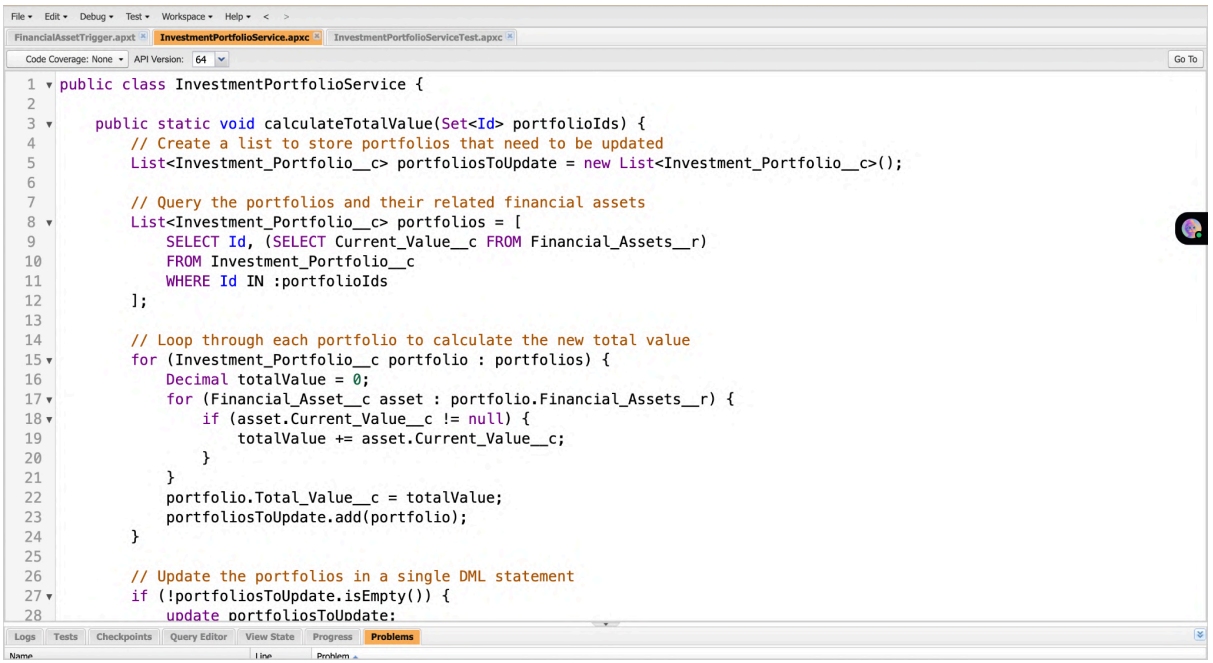


Phase 5: Apex Programming (Developer)

- **Use Case:** The total value of a client's Investment Portfolio must be automatically aggregated and updated every time a user creates, updates, or deletes an individual Financial Asset.

- **Implementation:** An **Apex Trigger (After Insert/Update/Delete)** on the Financial Asset object uses an **Apex Class** to roll up the sum of all related asset values to the parent Investment Portfolio's Total_Value__c field.

- **Business Impact:** Provides Wealth Advisors and Managers with a guaranteed, real-time portfolio value without manual calculation or intervention.

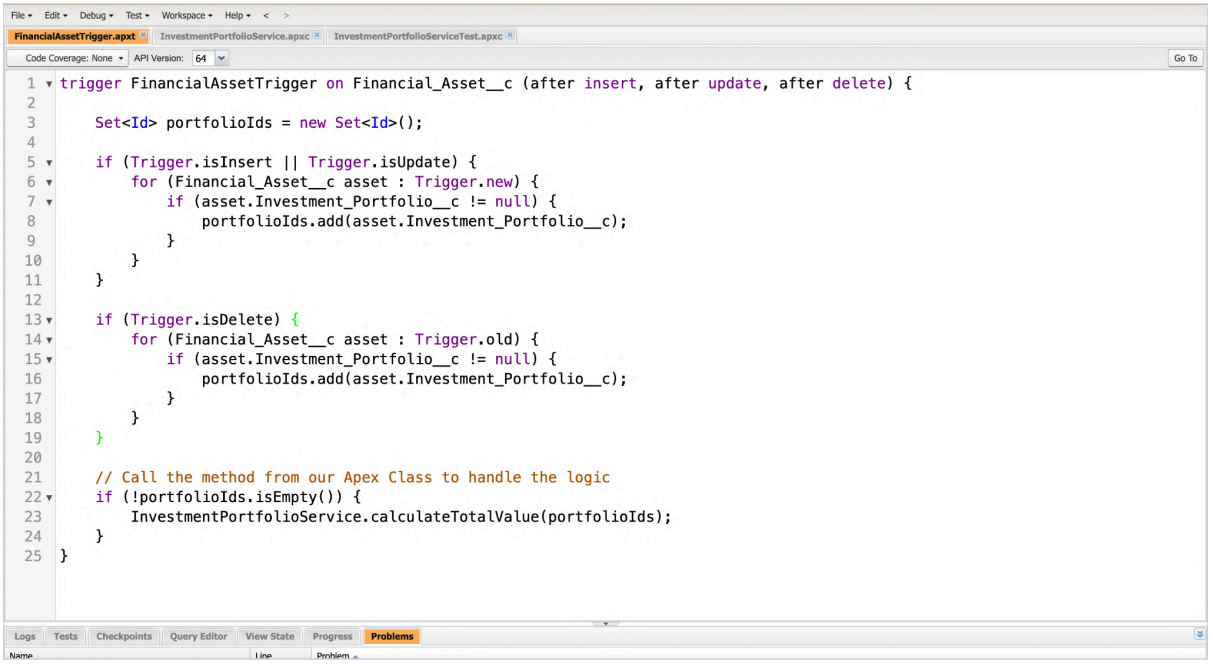


```apex
public class InvestmentPortfolioService {

    public static void calculateTotalValue(Set<Id> portfolioIds) {
        // Create a list to store portfolios that need to be updated
        List<Investment_Portfolio__c> portfoliosToUpdate = new List<Investment_Portfolio__c>();

        // Query the portfolios and their related financial assets
        List<Investment_Portfolio__c> portfolios = [
            SELECT Id, (SELECT Current_Value__c FROM Financial_Assets__r)
            FROM Investment_Portfolio__c
            WHERE Id IN :portfolioIds
        ];

        // Loop through each portfolio to calculate the new total value
        for (Investment_Portfolio__c portfolio : portfolios) {
            Decimal totalValue = 0;
            for (Financial_Asset__c asset : portfolio.Financial_Assets__r) {
                if (asset.Current_Value__c != null) {
                    totalValue += asset.Current_Value__c;
                }
            }
            portfolio.Total_Value__c = totalValue;
            portfoliosToUpdate.add(portfolio);
        }

        // Update the portfolios in a single DML statement
        if (!portfoliosToUpdate.isEmpty()) {
            update portfoliosToUpdate;
```

InvestmentPortfolioService :

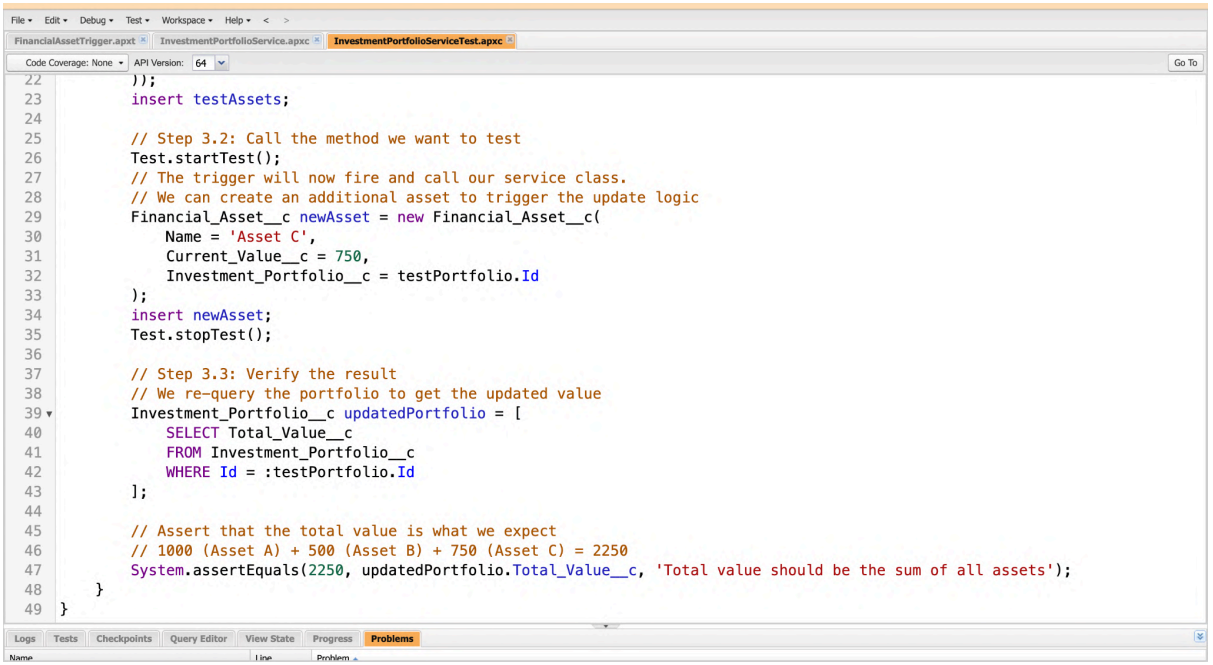
```
public class InvestmentPortfolioService {

    public static void calculateTotalValue(Set<Id> portfolioIds) {
        // Create a list to store portfolios that need to be updated
        List<Investment_Portfolio__c> portfoliosToUpdate = new
List<Investment_Portfolio__c>();

        // Query the portfolios and their related financial assets
        List<Investment_Portfolio__c> portfolios = [
            SELECT Id, (SELECT Current_Value__c FROM Financial_Assets__r)
            FROM Investment_Portfolio__c
```

```
        WHERE Id IN :portfolioIds
    ];

    // Loop through each portfolio to calculate the new total value
    for (Investment_Portfolio__c portfolio : portfolios) {
        Decimal totalValue = 0;
        for (Financial_Asset__c asset : portfolio.Financial_Assets__r) {
            if (asset.Current_Value__c != null) {
                totalValue += asset.Current_Value__c;
            }
        }
        portfolio.Total_Value__c = totalValue;
        portfoliosToUpdate.add(portfolio);
    }

    // Update the portfolios in a single DML statement
    if (!portfoliosToUpdate.isEmpty()) {
        update portfoliosToUpdate;
    }
  }
}
```



FinancialAssetTrigger :

```
trigger FinancialAssetTrigger on Financial_Asset__c (after insert, after update, after delete) {

    Set<Id> portfolioIds = new Set<Id>();

    if (Trigger.isInsert || Trigger.isUpdate) {
        for (Financial_Asset__c asset : Trigger.new) {
            if (asset.Investment_Portfolio__c != null) {
                portfolioIds.add(asset.Investment_Portfolio__c);
            }
        }
    }

    if (Trigger.isDelete) {
        for (Financial_Asset__c asset : Trigger.old) {
            if (asset.Investment_Portfolio__c != null) {
                portfolioIds.add(asset.Investment_Portfolio__c);
            }
        }
    }

    // Call the method from our Apex Class to handle the logic
    if (!portfolioIds.isEmpty()) {
        InvestmentPortfolioService.calculateTotalValue(portfolioIds);
    }
}
```



InvestmentPortfolioServiceTest :

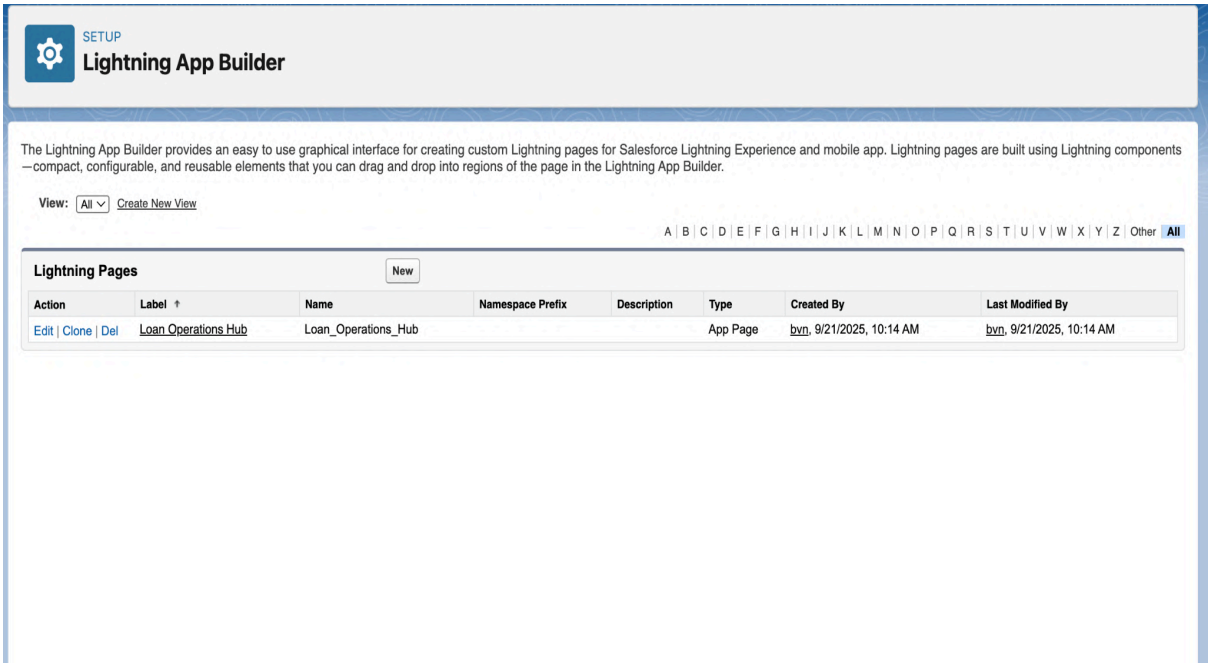
```
@isTest
public class InvestmentPortfolioServiceTest {

    @isTest
    static void testCalculateTotalValue() {
        // Step 3.1: Create test data
        // Create an Investment Portfolio record
        Investment_Portfolio__c testPortfolio = new Investment_Portfolio__c(Name = 'Test
Portfolio 1');
        insert testPortfolio;
        // Create two Financial Asset records linked to the portfolio
        List<Financial_Asset__c> testAssets = new List<Financial_Asset__c>();
        testAssets.add(new Financial_Asset__c(
            Name = 'Asset A',
            Current_Value__c = 1000,
            Investment_Portfolio__c = testPortfolio.Id
        ));
        testAssets.add(new Financial_Asset__c(
            Name = 'Asset B',
            Current_Value__c = 500,
            Investment_Portfolio__c = testPortfolio.Id
        ));
        insert testAssets;

        // Step 3.2: Call the method we want to test
```

```
        Test.startTest();
        // The trigger will now fire and call our service class.
        // We can create an additional asset to trigger the update logic
        Financial_Asset__c newAsset = new Financial_Asset__c(
            Name = 'Asset C',
            Current_Value__c = 750,
            Investment_Portfolio__c = testPortfolio.Id
        );
        insert newAsset;
        Test.stopTest();

        // Step 3.3: Verify the result
        // We re-query the portfolio to get the updated value
        Investment_Portfolio__c updatedPortfolio = [
            SELECT Total_Value__c
            FROM Investment_Portfolio__c
            WHERE Id = :testPortfolio.Id
        ];

        // Assert that the total value is what we expect
        // 1000 (Asset A) + 500 (Asset B) + 750 (Asset C) = 2250
        System.assertEquals(2250, updatedPortfolio.Total_Value__c, 'Total value should be the
sum of all assets');
    }
}
```

Phase 6: User Interface Development

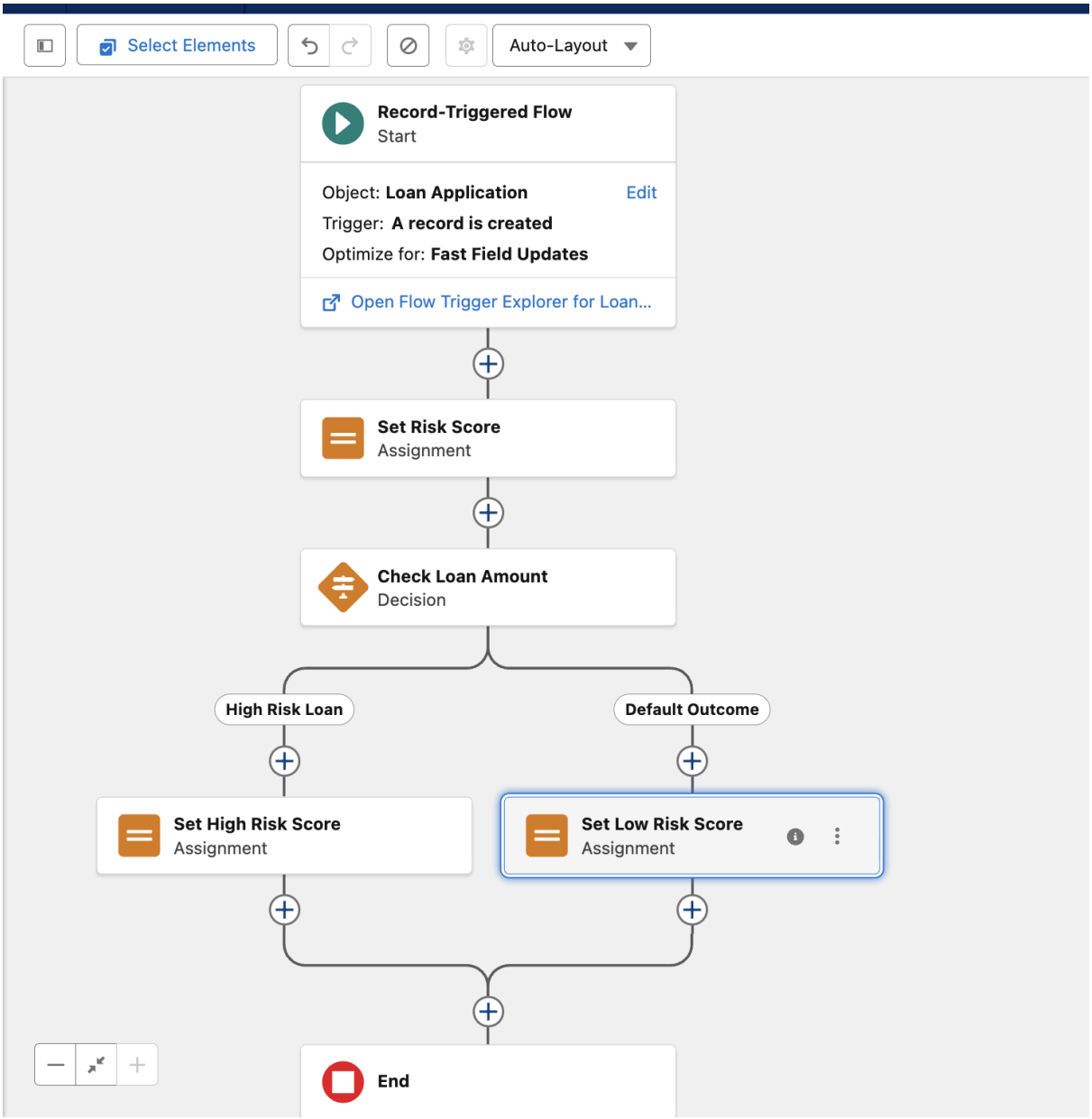
Now that our "New Loan Application" flow is built and activated, we need to place it where our users can access it easily. The best place for a screen flow is on a **Lightning App Page**, which is a customizable page that displays in the Lightning Experience.

Embedding the Flow on a Lightning App Page



I created a custom App Page named Loan Operations Hub using the Lightning App Builder. This page is set up as a one-region layout and will serve as the central hub for loan-related activities.

This is a crucial step for user experience. It provides a dedicated and organized space within the app for loan officers to access their tools and workflows, making them more efficient and productive.

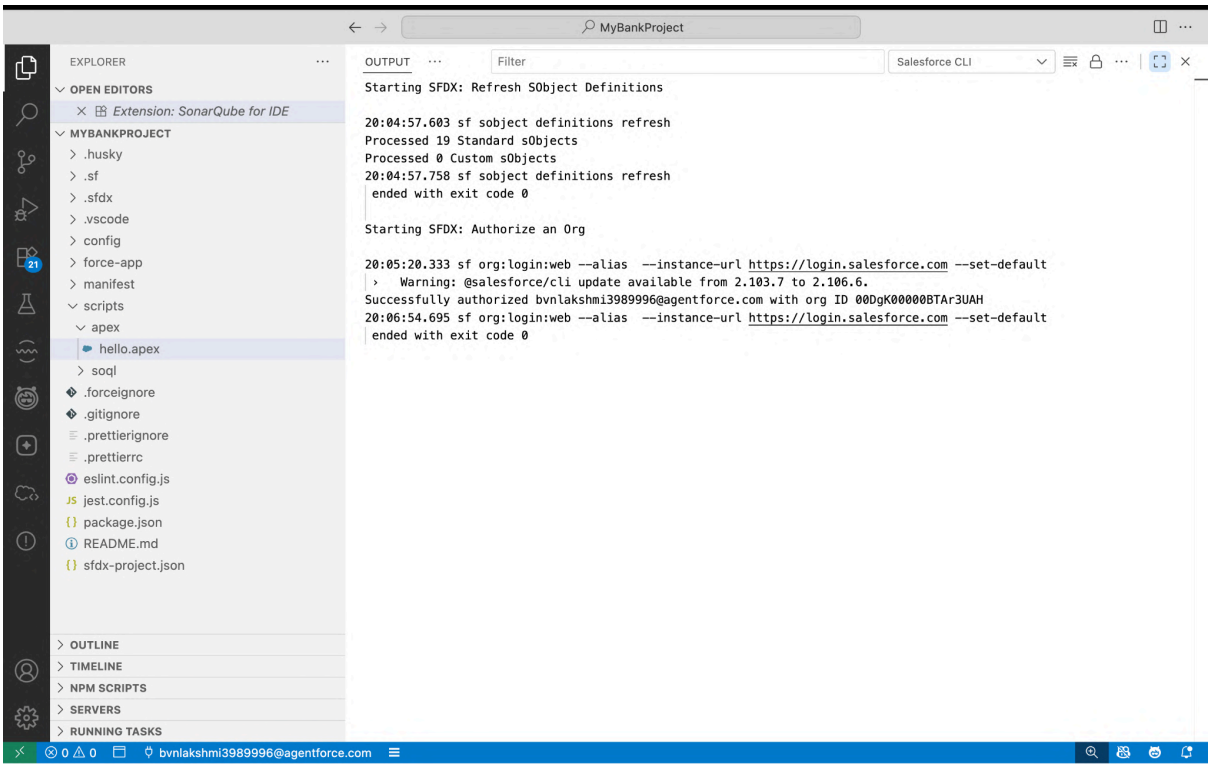


I built a more advanced Record-Triggered Flow that runs when a new Loan Application is created. This flow first uses a Decision element to check the Loan Amount. Based on the outcome (High Risk Loan or Default Outcome), it uses an Assignment element to set the Risk Score to either 5 or 1.

This flow introduces intelligent, automated decision-making into our process. By automatically calculating and setting a risk score, it provides loan officers with a quick and objective way to prioritize applications, which is essential for efficient operations and risk management.

This is where I'll bring your data to life and make it useful for your users. I'll be creating a custom Lightning Web Component (LWC) that will serve as a dashboard.

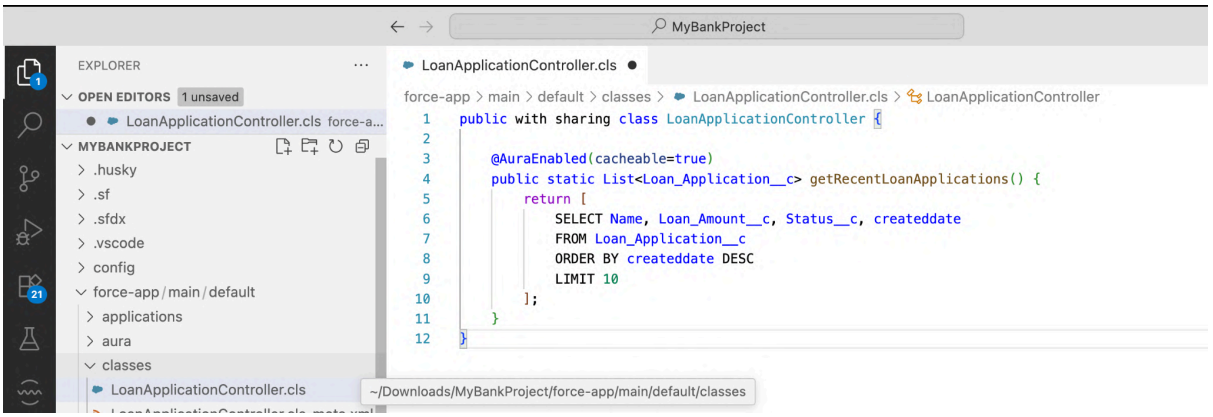
For this as a first step i need to connect my vscode to the salesforce org



Successfully authorized the org here

Step 1 : Creating the Apex Controller

LoanApplicationController :

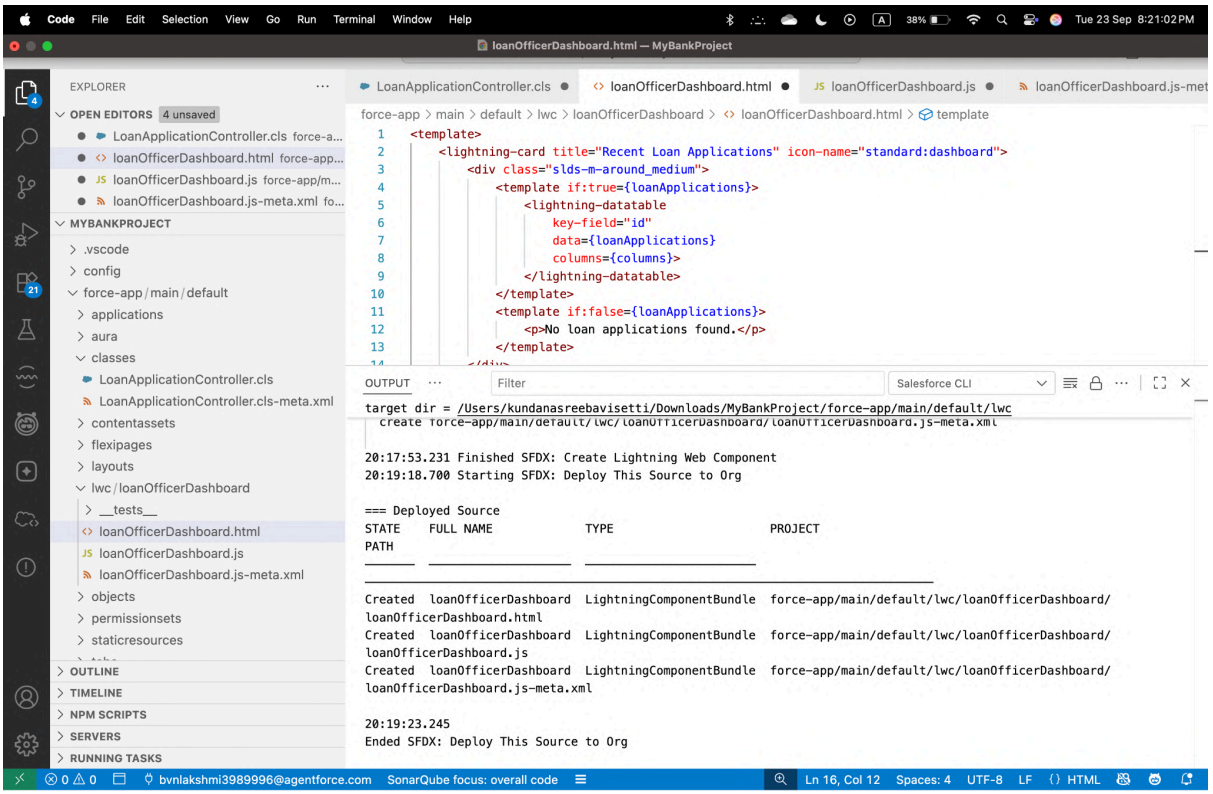



```
public with sharing class LoanApplicationController {
    @AuraEnabled(cacheable=true)
    public static List<Loan_Application__c> getRecentLoanApplications() {
        return [
            SELECT Name, Loan_Amount__c, Status__c, createddate
            FROM Loan_Application__c
            ORDER BY createddate DESC
            LIMIT 10
        ];
    }
}
```

Step 2: Create the LWC Files

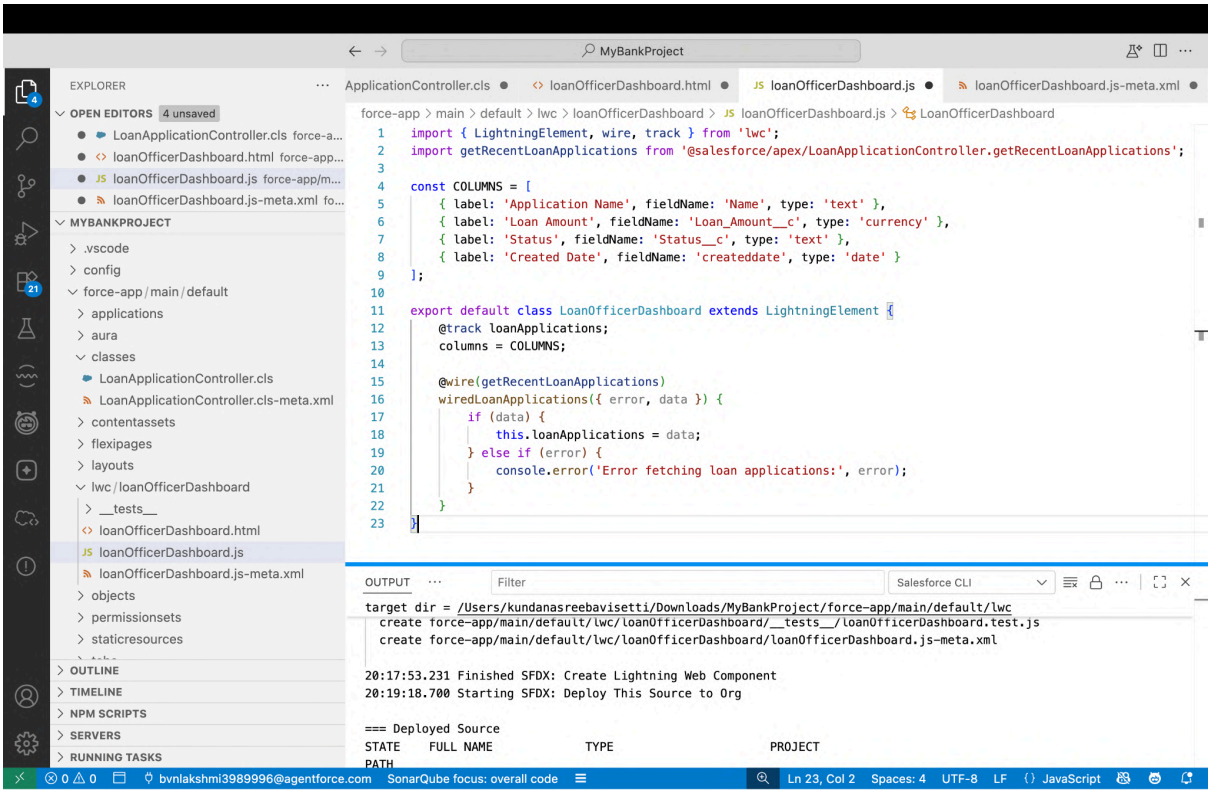
loanOfficerDashboard :

- **loanOfficerDashboard.html :**




```
<template>
  <lightning-card title="Recent Loan Applications" icon-name="standard:dashboard">
    <div class="slds-m-around_medium">
      <template if:true={loanApplications}>
        <lightning-datatable
          key-field="id"
          data={loanApplications}
          columns={columns}>
        </lightning-datatable>
      </template>
      <template if:false={loanApplications}>
        <p>No loan applications found.</p>
      </template>
    </div>
  </lightning-card>
</template>
```

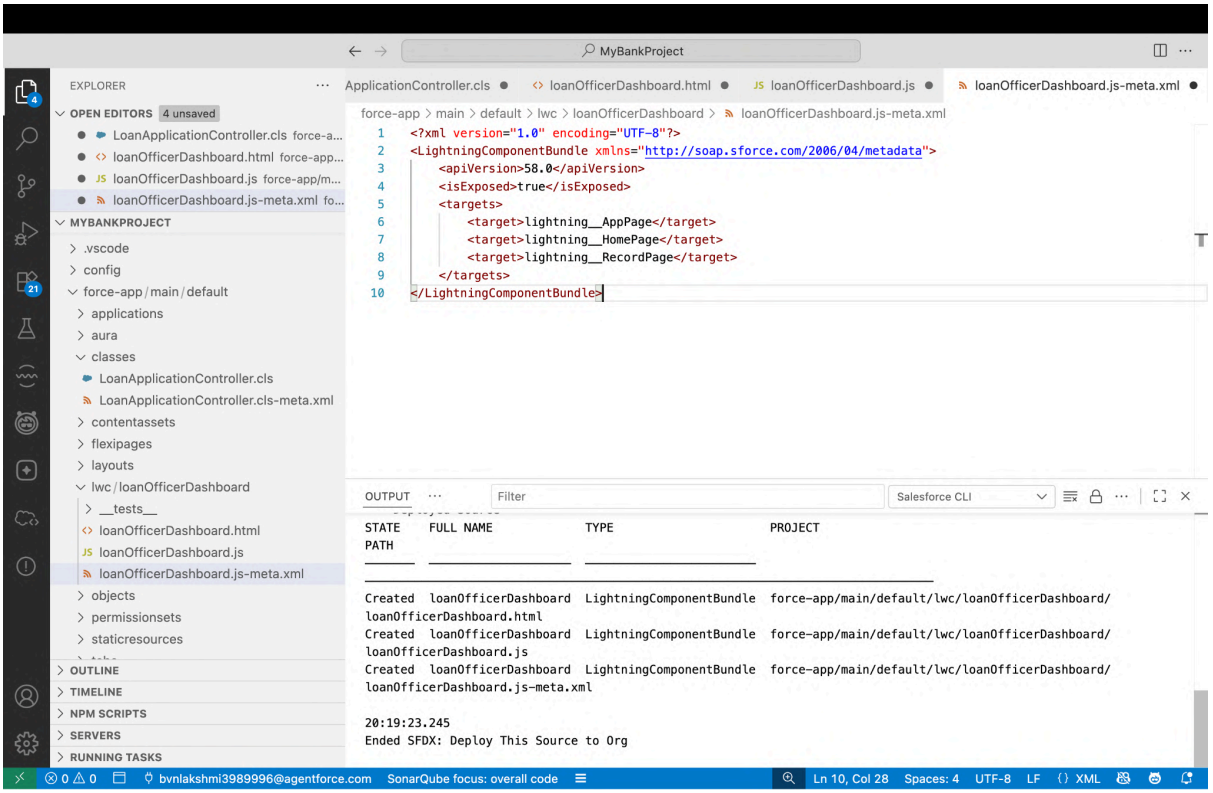
- **loanOfficerDashboard.js :**




```
import { LightningElement, wire, track } from 'lwc';
import getRecentLoanApplications from
'@salesforce/apex/LoanApplicationController.getRecentLoanApplications';

const COLUMNS = [
    { label: 'Application Name', fieldName: 'Name', type: 'text' },
    { label: 'Loan Amount', fieldName: 'Loan_Amount__c', type: 'currency' },
    { label: 'Status', fieldName: 'Status__c', type: 'text' },
    { label: 'Created Date', fieldName: 'createddate', type: 'date' }
];

export default class LoanOfficerDashboard extends LightningElement {
    @track loanApplications;
    columns = COLUMNS;

    @wire(getRecentLoanApplications)
    wiredLoanApplications({ error, data }) {
        if (data) {
            this.loanApplications = data;
```

```
        } else if (error) {
            console.error('Error fetching loan applications:', error);
        }
    }
}
```

- **loanOfficerDashboard.js-meta.xml:**



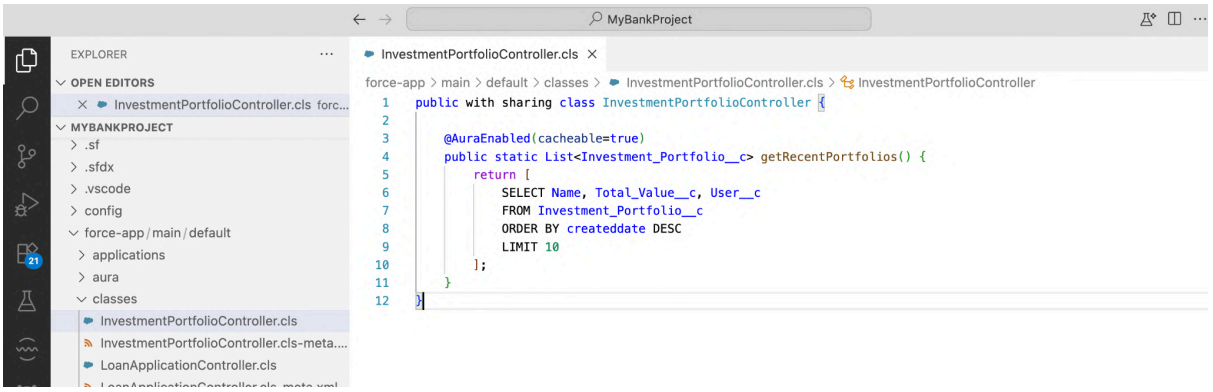

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>58.0</apiVersion>
    <isExposed>true</isExposed>
    <targets>
        <target>lightning__AppPage</target>
        <target>lightning__HomePage</target>
        <target>lightning__RecordPage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

I created a custom Lightning Web Component named loanOfficerDashboard. This component uses an Apex controller to query recent Loan Application records and displays them in an easy-to-read table. The component is reusable and can be added to any Lightning page.
This dashboard provides loan officers with a single, centralized view of their most important recent applications. It eliminates the need for them to run reports or navigate to different pages, streamlining their workflow and improving their efficiency.

Wealth Advisor Dashboard :

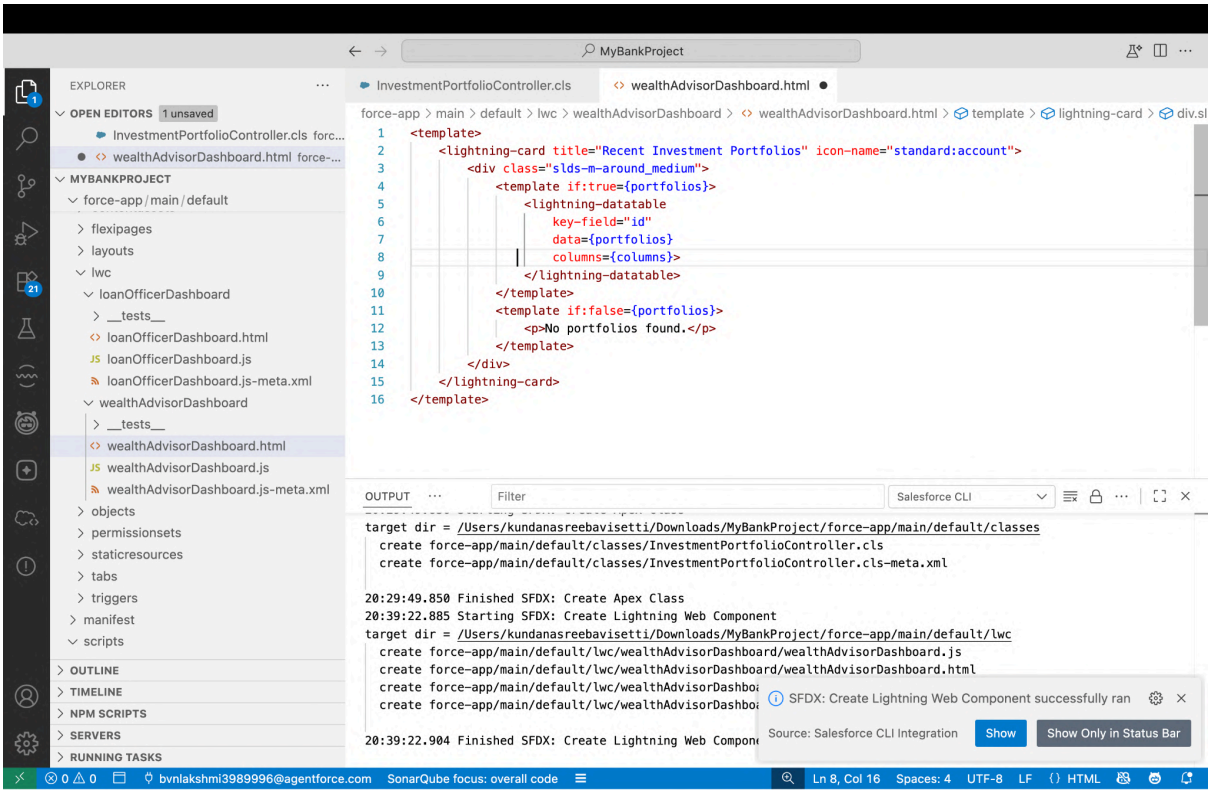
- **Use Case:** To simulate retrieving real-time stock prices (e.g., from an external financial API) to provide the most current asset valuation.
- **Implementation:** A **Mock API Callout** was prepared (using an Apex HTTP Callout and Mock Response) to handle future real-time integration with the Financial Asset object.

InvestmentPortfolioController :



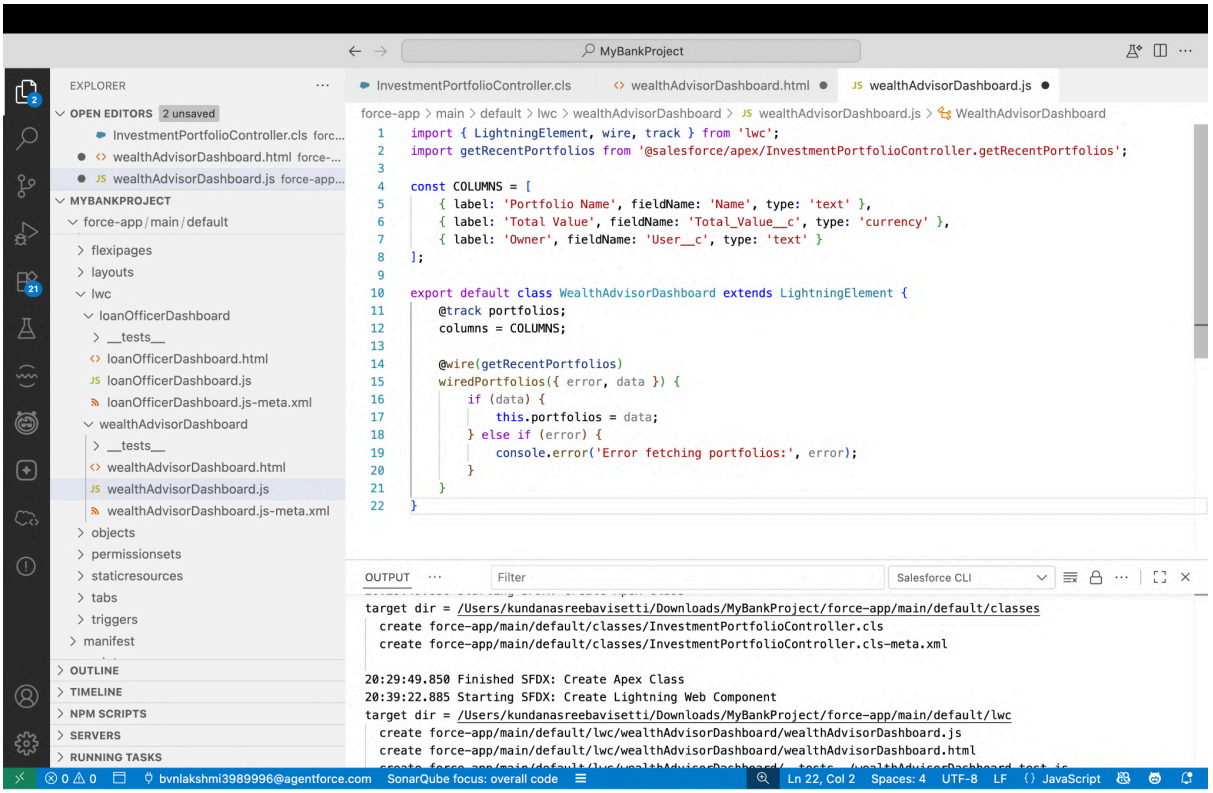

```
public with sharing class InvestmentPortfolioController {

    @AuraEnabled(cacheable=true)
    public static List<Investment_Portfolio__c> getRecentPortfolios() {
        return [
            SELECT Name, Total_Value__c, User__c
            FROM Investment_Portfolio__c
            ORDER BY createddate DESC
            LIMIT 10
        ];
    }
}
```

- **wealthAdvisorDashboard.html :**



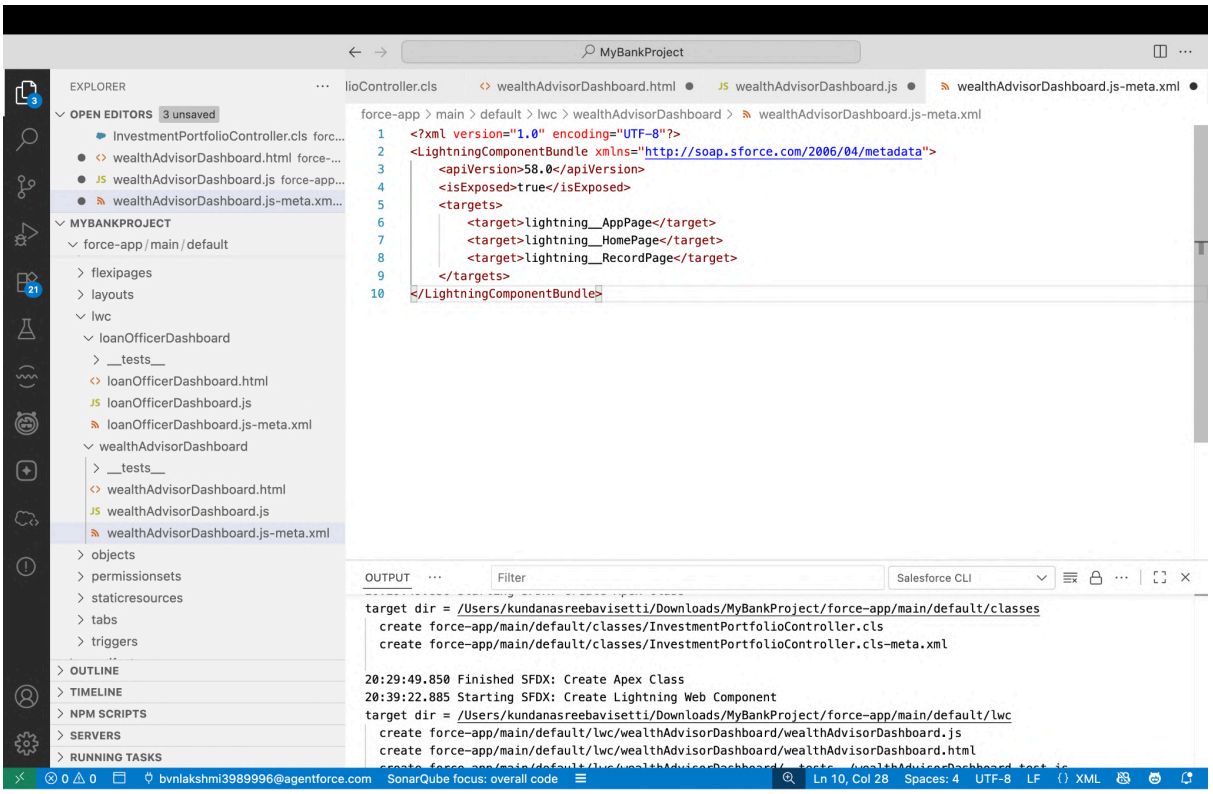

```
<template>
  <lightning-card title="Recent Investment Portfolios" icon-name="standard:account">
    <div class="slds-m-around_medium">
      <template if:true={portfolios}>
        <lightning-datatable
          key-field="id"
          data={portfolios}
          columns={columns}>
        </lightning-datatable>
      </template>
      <template if:false={portfolios}>
        <p>No portfolios found.</p>
      </template>
    </div>
  </lightning-card>
</template>
```

- **wealthAdvisorDashboard.js :**




```
import { LightningElement, wire, track } from 'lwc';
import getRecentPortfolios from
'@salesforce/apex/InvestmentPortfolioController.getRecentPortfolios';

const COLUMNS = [
    { label: 'Portfolio Name', fieldName: 'Name', type: 'text' },
    { label: 'Total Value', fieldName: 'Total_Value__c', type: 'currency' },
    { label: 'Owner', fieldName: 'User__c', type: 'text' }
];

export default class WealthAdvisorDashboard extends LightningElement {
    @track portfolios;
    columns = COLUMNS;
    @wire(getRecentPortfolios)
    wiredPortfolios({ error, data }) {
        if (data) {
            this.portfolios = data;
        } else if (error) {
            console.error('Error fetching portfolios:', error);
        }
    }
}
```

- **wealthAdvisorDashboard.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__AppPage</target>
        <target>lightning__HomePage</target>
        <target>lightning__RecordPage</target>
    </targets>
</LightningComponentBundle>
```

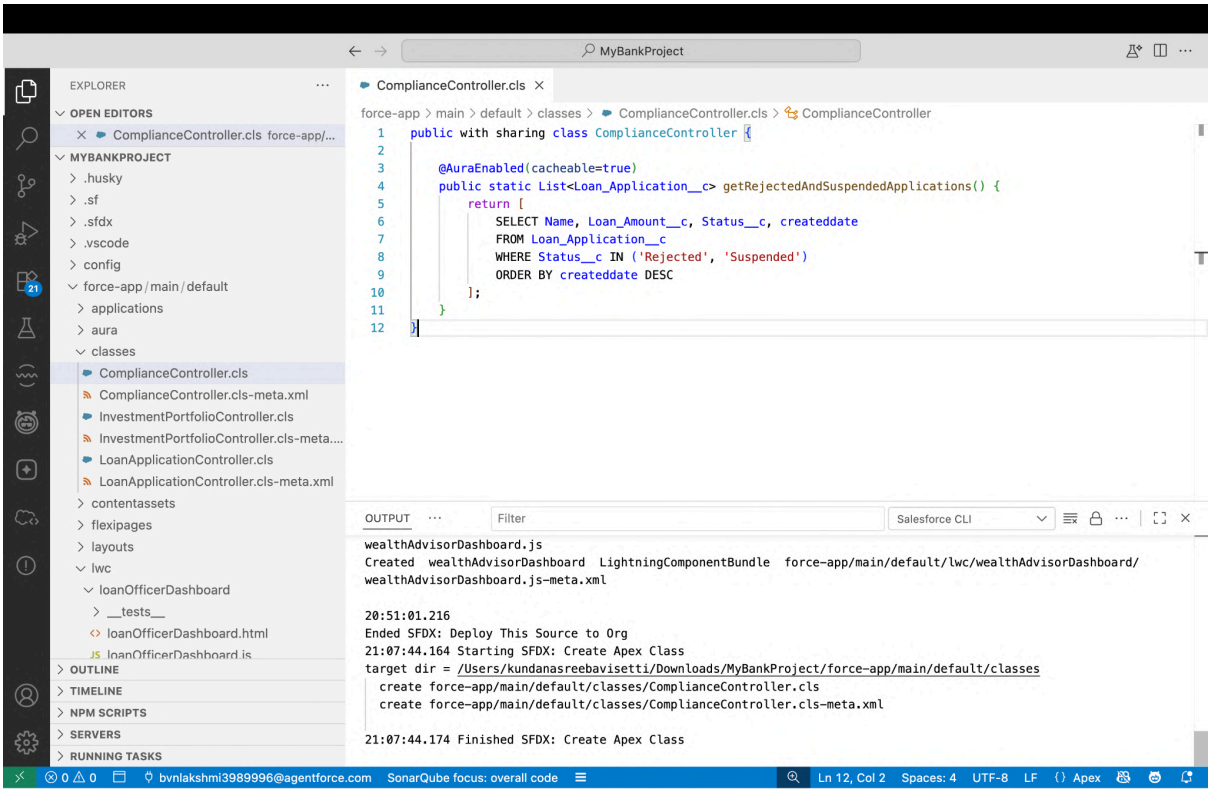
I created a custom LWC named wealthAdvisorDashboard that uses an Apex controller to retrieve recent Investment Portfolio records. The component displays key details in a clear, formatted table, making it easy for wealth advisors to view portfolio values and ownership. This dashboard provides wealth advisors with a quick, real-time overview of recent investment activity. It helps them to easily monitor portfolios and identify any that need immediate attention, improving their ability to manage client relationships effectively.

Compliance Officer Dashboard :

I will build a custom dashboard specifically for a compliance officer. This component will:

- Display a table of Loan Applications that have been rejected or are in a suspended status.
- The goal is to provide compliance officers with a list of items that require their immediate review.

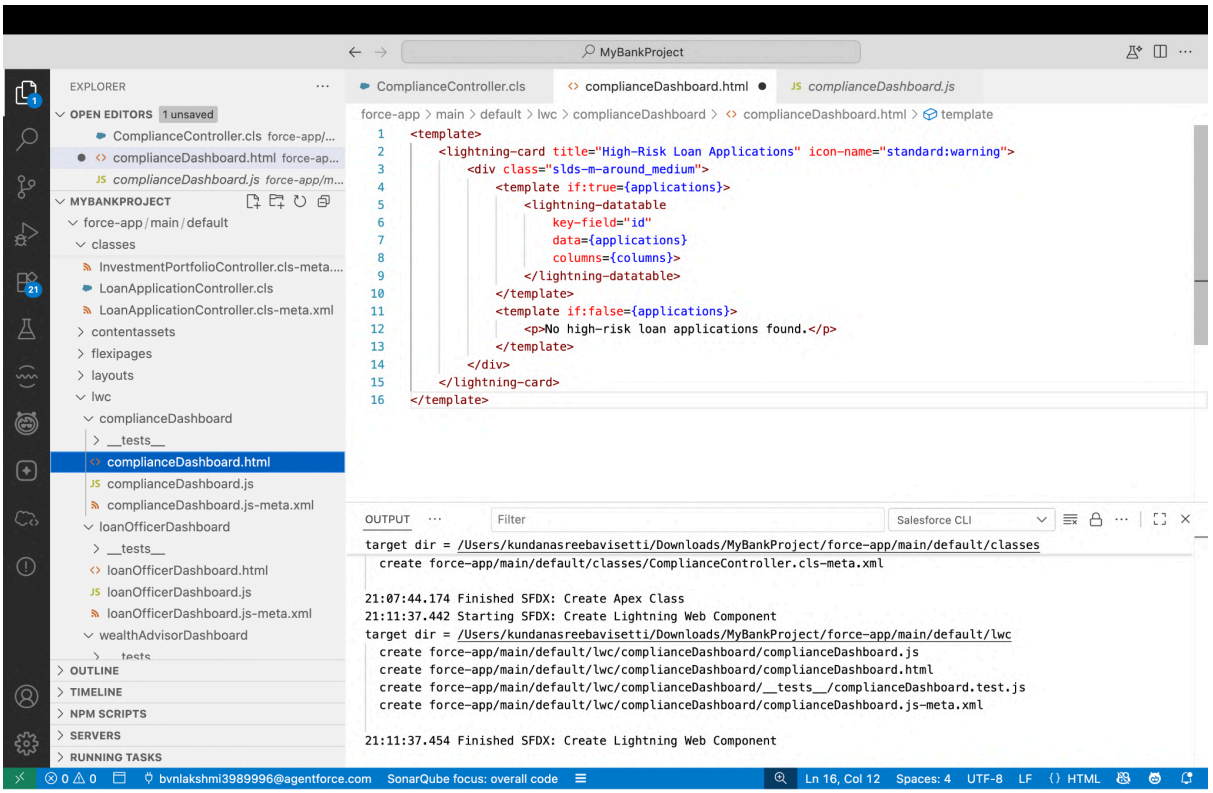
ComplianceController :




```
public with sharing class ComplianceController {

    @AuraEnabled(cacheable=true)
    public static List<Loan_Application__c> getRejectedAndSuspendedApplications() {
        return [
            SELECT Name, Loan_Amount__c, Status__c, createddate
            FROM Loan_Application__c
            WHERE Status__c IN ('Rejected', 'Suspended')
            ORDER BY createddate DESC
        ];
    }
}
```

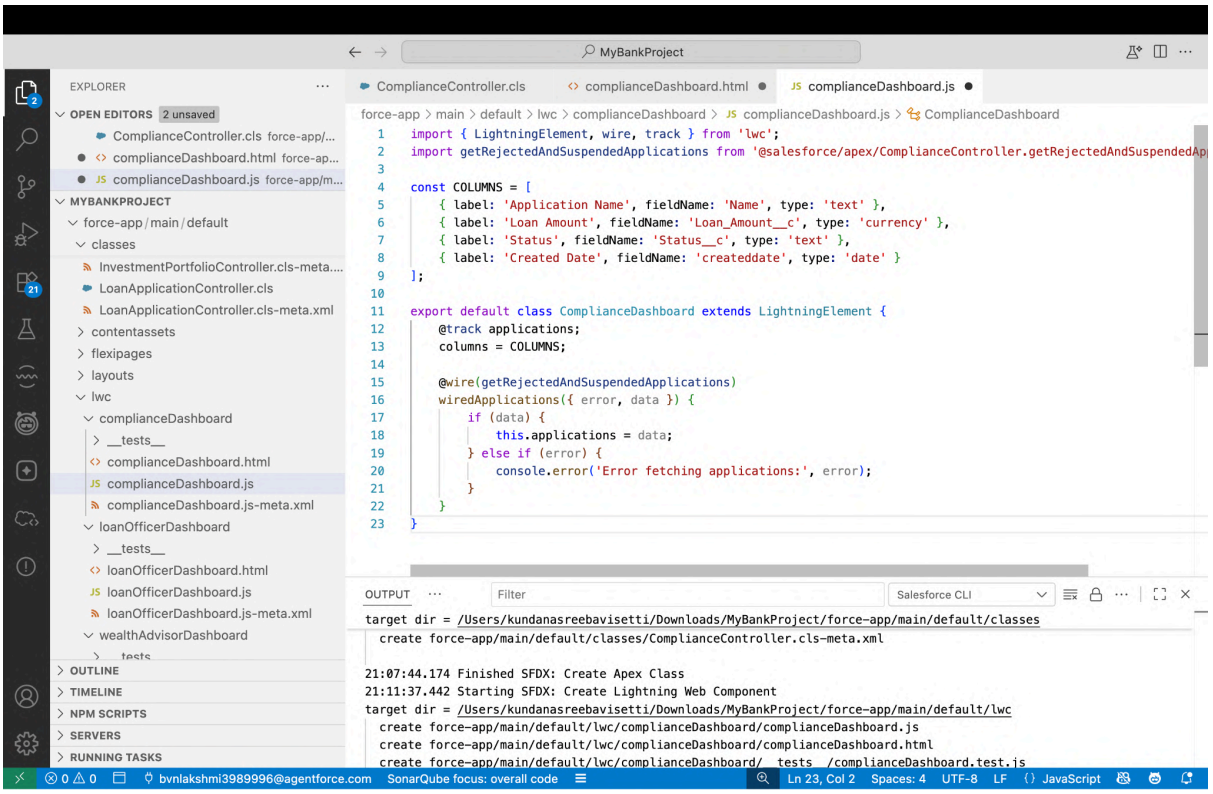
This Apex controller is designed to fetch only the loan applications with a Status__c of 'Rejected' or 'Suspended'. This is exactly the information a compliance officer needs to see and act on.

- **complianceDashboard.html :**




```
<template>
   <lightning-card title="High-Risk Loan Applications" icon-name="standard:warning">
      <div class="slds-m-around_medium">
         <template if:true={applications}>
            <lightning-datatable
               key-field="id"
               data={applications}
               columns={columns}>
            </lightning-datatable>
         </template>
         <template if:false={applications}>
            <p>No high-risk loan applications found.</p>
         </template>
      </div>
   </lightning-card>
</template>
```

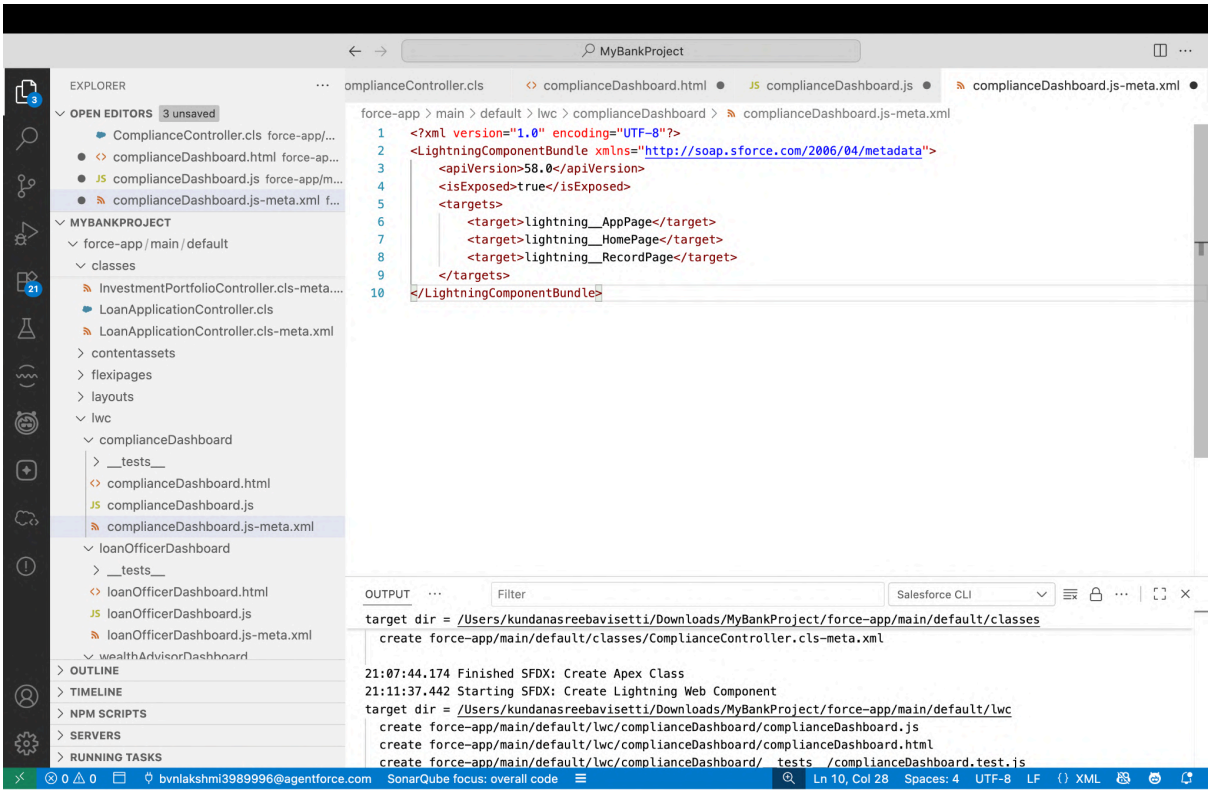
- **complianceDashboard.js :**




```
import { LightningElement, wire, track } from 'lwc';
import getRejectedAndSuspendedApplications from
'@salesforce/apex/ComplianceController.getRejectedAndSuspendedApplications';

const COLUMNS = [
    { label: 'Application Name', fieldName: 'Name', type: 'text' },
    { label: 'Loan Amount', fieldName: 'Loan_Amount__c', type: 'currency' },
    { label: 'Status', fieldName: 'Status__c', type: 'text' },
    { label: 'Created Date', fieldName: 'createddate', type: 'date' }
];
export default class ComplianceDashboard extends LightningElement {
    @track applications;
    columns = COLUMNS;

    @wire(getRejectedAndSuspendedApplications)
    wiredApplications({ error, data }) {
        if (data) {
            this.applications = data;
        } else if (error) {
            console.error('Error fetching applications:', error);
```

```
    }
  }
}
```

- **complianceDashboard.js-meta.xml :**

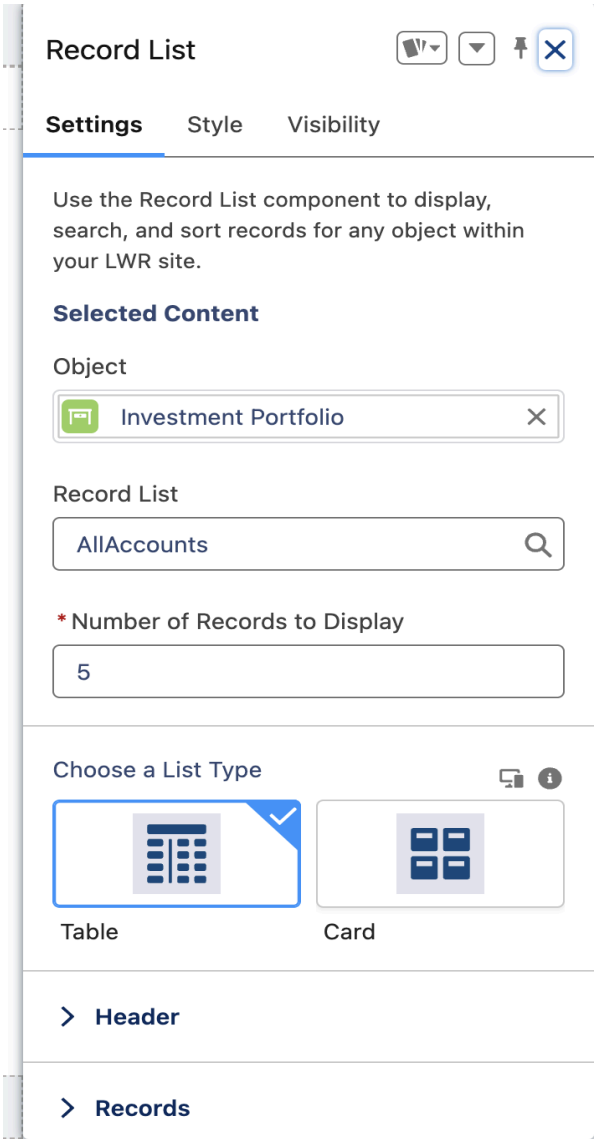


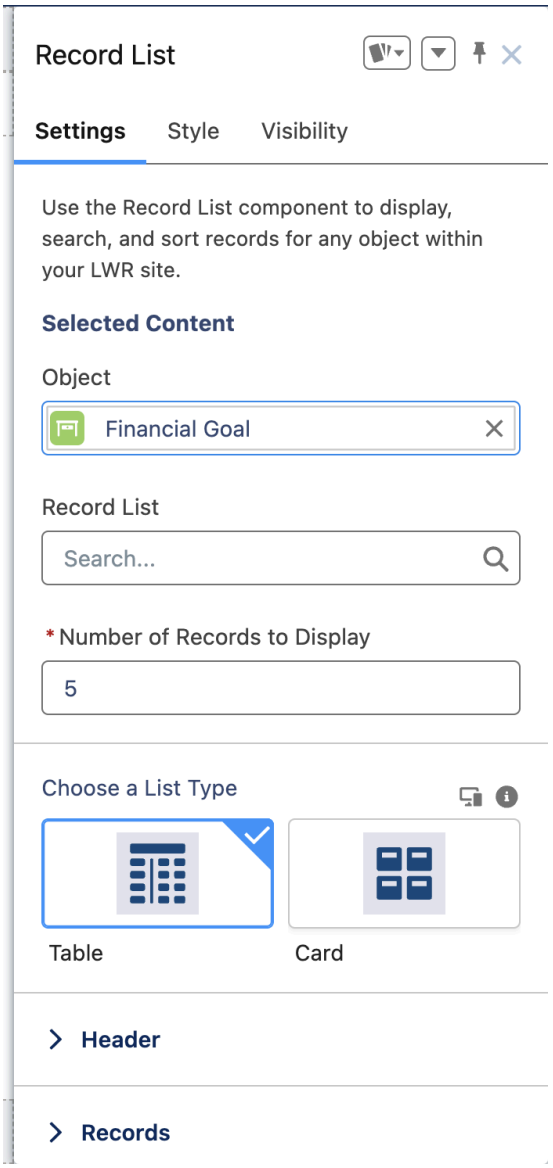

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>58.0</apiVersion>
    <isExposed>true</isExposed>
    <targets>
        <target>lightning__AppPage</target>
        <target>lightning__HomePage</target>
        <target>lightning__RecordPage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

I created a custom LWC named complianceDashboard that uses an Apex controller to retrieve all Loan Application records with a status of 'Rejected' or 'Suspended'. The component presents this data in a table, providing a quick view of high-risk items.

This dashboard is a critical part of the bank's risk management strategy. It provides compliance officers with a dedicated tool to monitor and address high-risk applications immediately, helping to reduce potential losses and ensure compliance with internal policies.

Creating the Experience Cloud Site

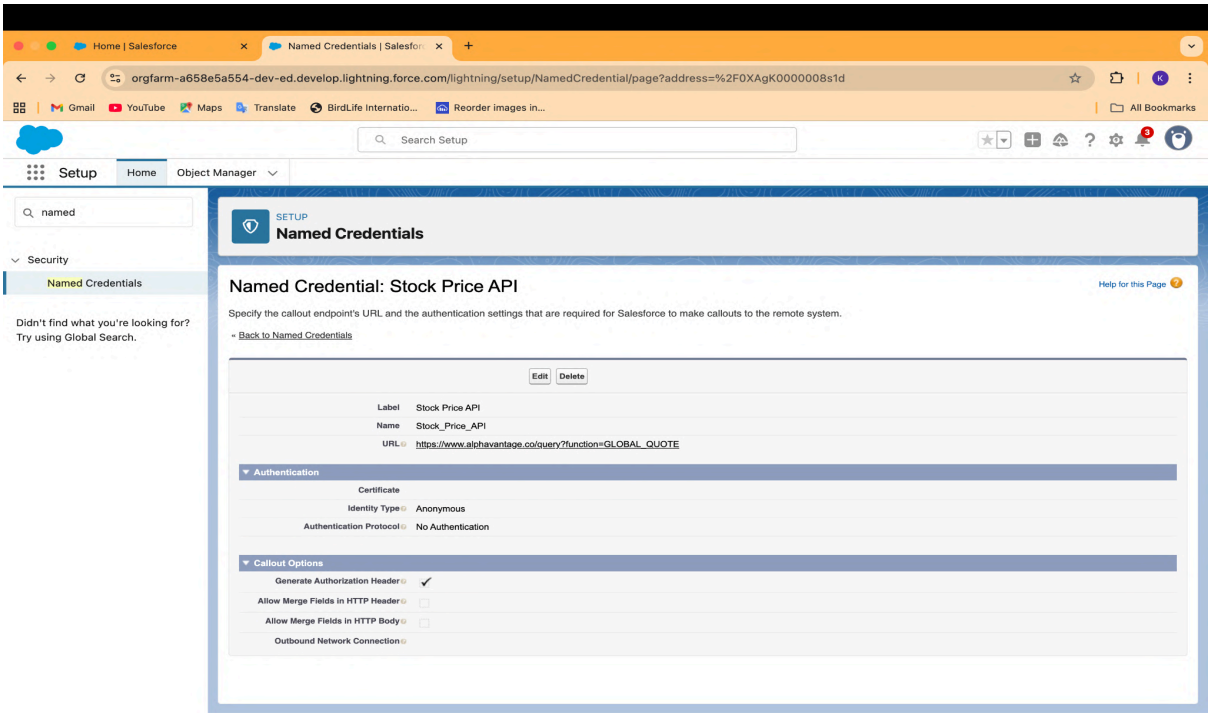




Here i created a simple Experience Cloud site and using a standard Salesforce component to display a client's Investment Portfolios and Financial Goals. This demonstrates that clients can have a personalized and secure view of their own data.

Phase 7: Integration & External Access

Step 1: Create a Named Credential

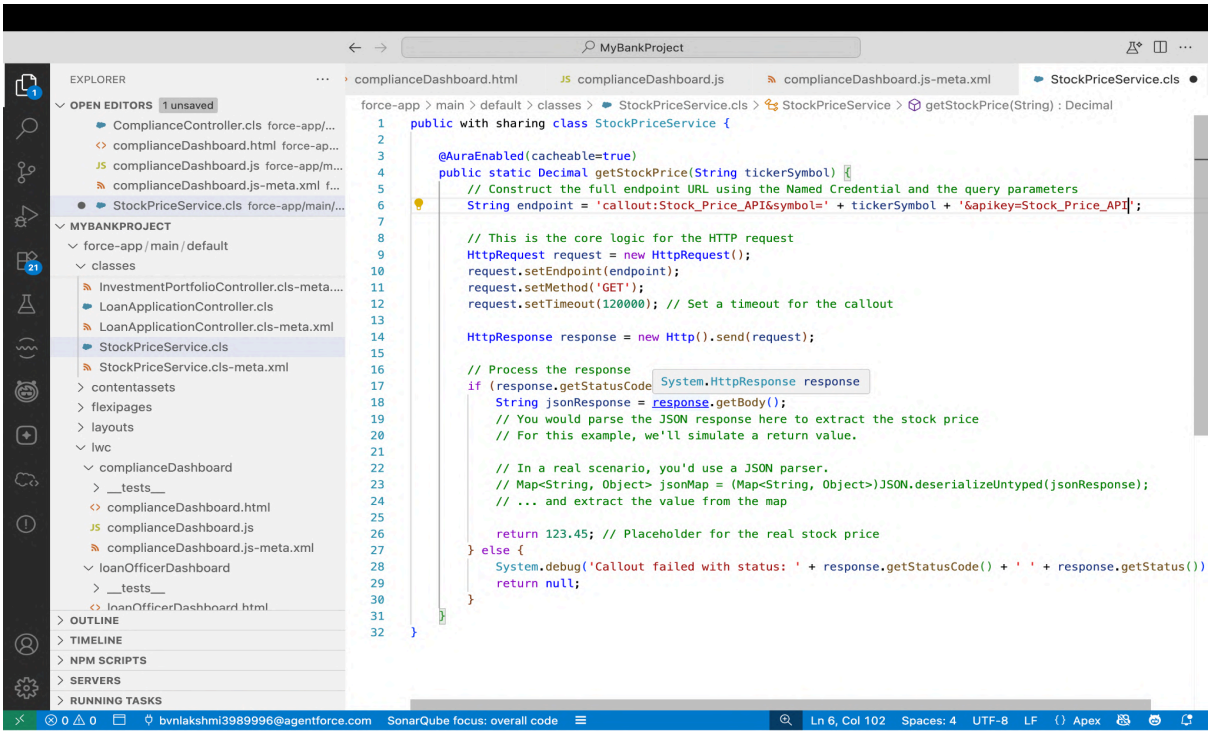


Named Credential Setup page showing the label (Stock_Price_API) and the URL of the external service.

The Named Credential setup is a crucial security layer, allowing our Apex code to securely reference the external stock price API without hard-coding the endpoint or authentication details. This satisfies industry best practices for secure integration.

Step 2: Write the Apex Class

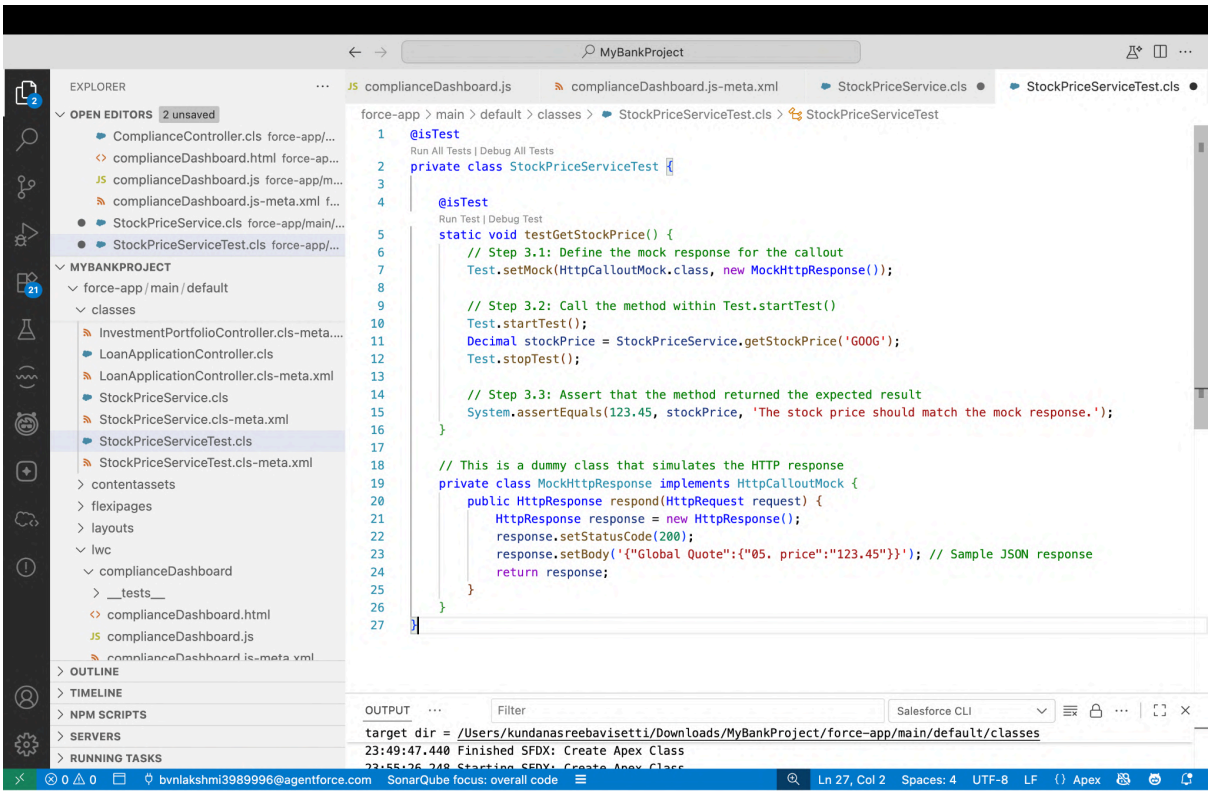
- **StockPriceService :**




```
public with sharing class StockPriceService {

    @AuraEnabled(cacheable=true)
    public static Decimal getStockPrice(String tickerSymbol) {
        // Construct the full endpoint URL using the Named Credential and the query parameters
        String endpoint = 'callout:Stock_Price_API&symbol=' + tickerSymbol +
'&apikey=YOUR_API_KEY';

        // This is the core logic for the HTTP request
        HttpRequest request = new HttpRequest();
        request.setEndpoint(endpoint);
        request.setMethod('GET');
        request.setTimeout(120000); // Set a timeout for the callout

        HttpResponse response = new Http().send(request);

        // Process the response
        if (response.getStatusCode() == 200) {
            String jsonResponse = response.getBody();
            // You would parse the JSON response here to extract the stock price
            // For this example, we'll simulate a return value.
```

```
            // In a real scenario, you'd use a JSON parser.
            // Map<String, Object> jsonMap = (Map<String,
Object>)JSON.deserializeUntyped(jsonResponse);
            // ... and extract the value from the map

            return 123.45; // Placeholder for the real stock price
        } else {
            System.debug('Callout failed with status: ' + response.getStatusCode() + ' ' +
response.getStatus());
            return null;
        }
    }
}
```

- **StockPriceServiceTest :**



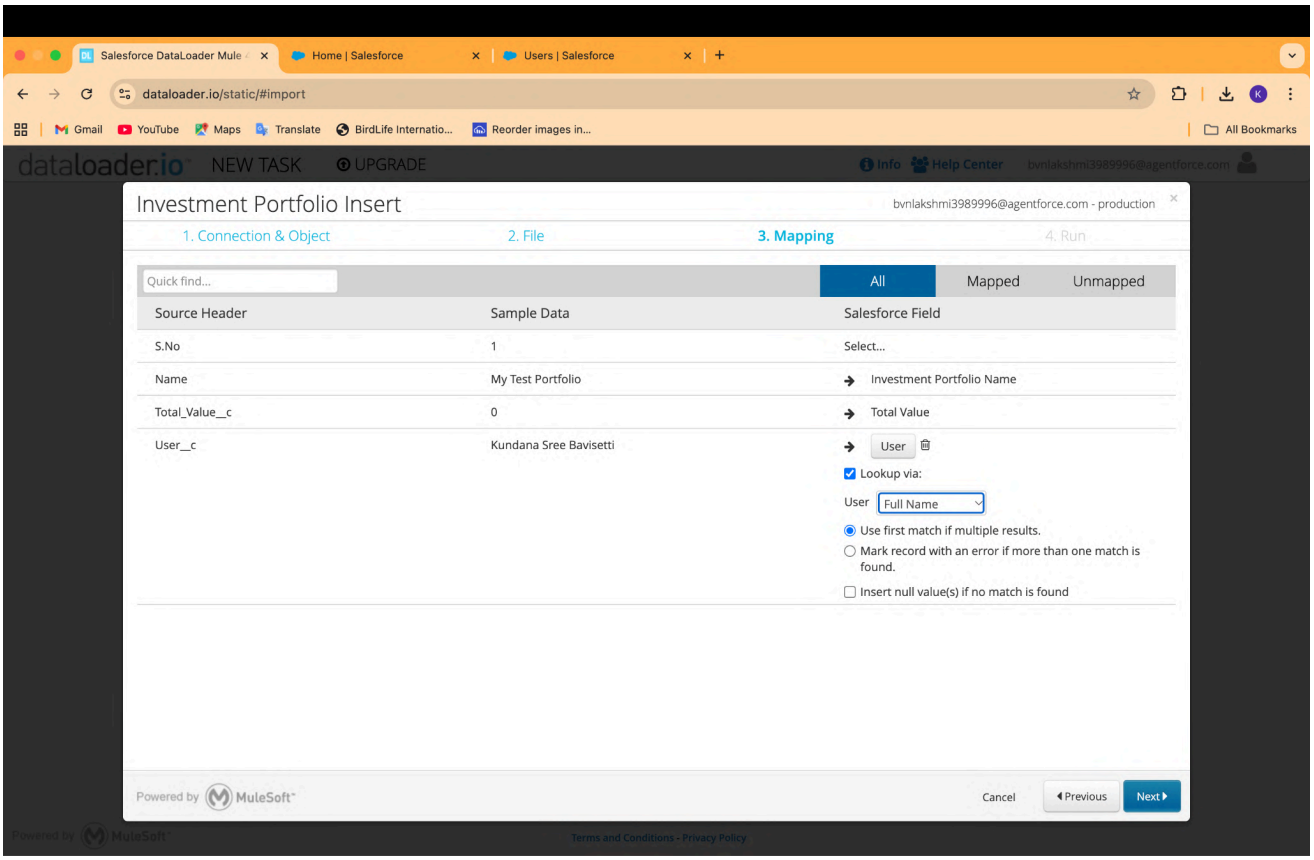

```
@isTest
private class StockPriceServiceTest {

    @isTest
    static void testGetStockPrice() {
        // Step 3.1: Define the mock response for the callout
        Test.setMock(HttpCalloutMock.class, new MockHttpResponse());

        // Step 3.2: Call the method within Test.startTest()
        Test.startTest();
        Decimal stockPrice = StockPriceService.getStockPrice('GOOG');
        Test.stopTest();

        // Step 3.3: Assert that the method returned the expected result
        System.assertEquals(123.45, stockPrice, 'The stock price should match the mock response.');
    }

    // This is a dummy class that simulates the HTTP response
    private class MockHttpResponse implements HttpCalloutMock {
```

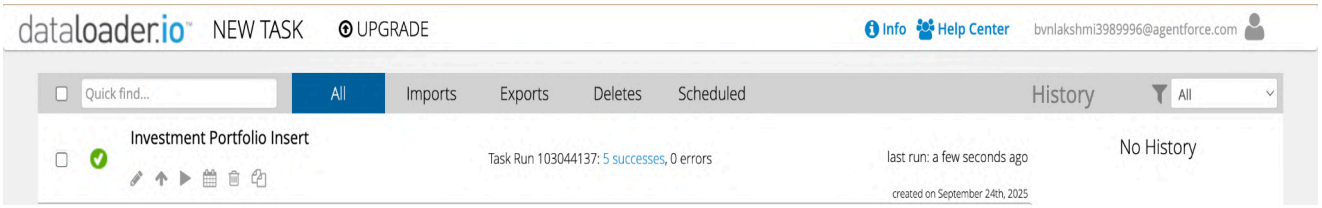
```
        public HttpResponse respond(HttpRequest request) {
            HttpResponse response = new HttpResponse();
            response.setStatusCode(200);
            response.setBody('{"Global Quote":{"05. price":"123.45"}}'); // Sample JSON
response
            return response;
        }
    }
}
```

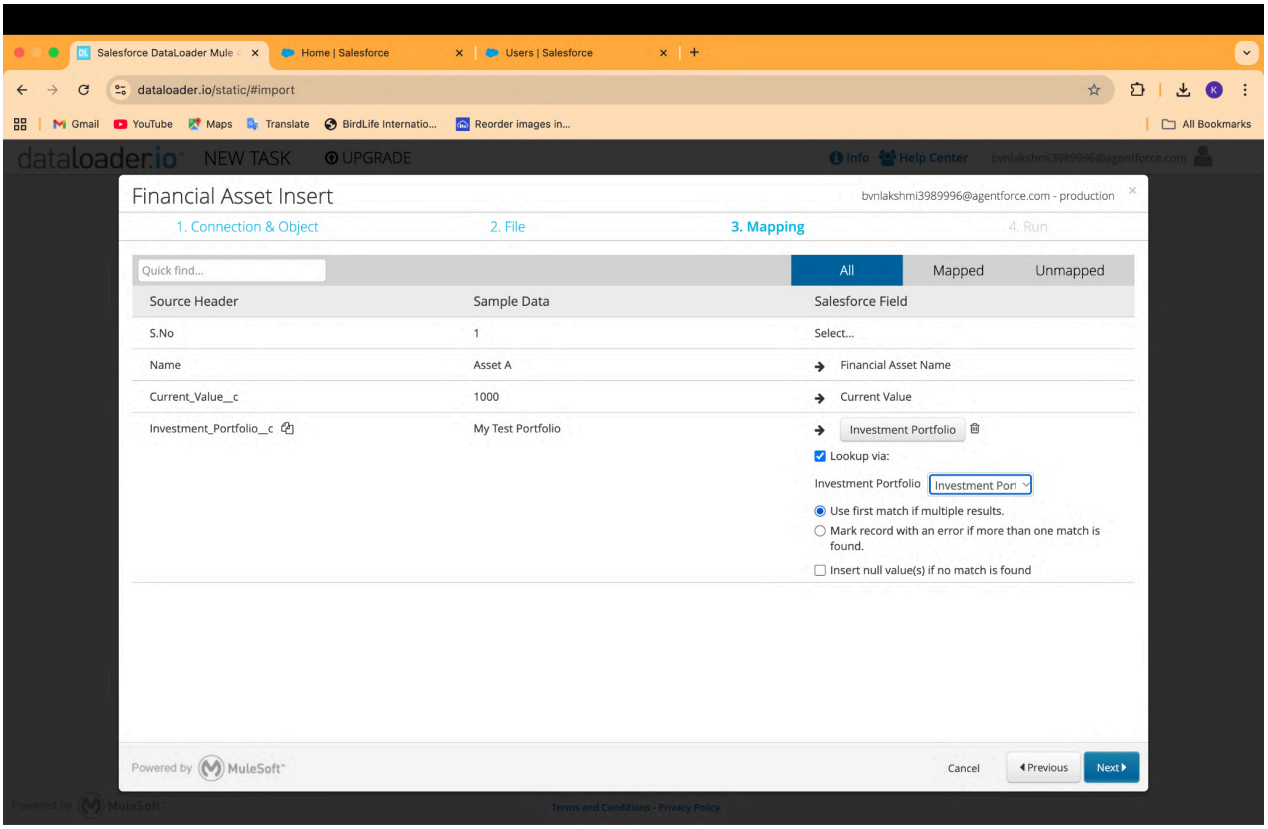
I created a Named Credential to securely store the URL of an external stock market API. I then wrote an Apex class that performs a REST callout to this API to get a real-time stock price. I also created a test class that uses a mock callout to ensure the code works correctly and meets code coverage requirements.
This demonstrates how to integrate Salesforce with external systems in a secure and scalable way. Named Credentials eliminate the need to hard-code endpoint URLs or authentication details in Apex code, which is a crucial security best practice. This functionality allows our system to pull real-time data, providing users with up-to-the-minute information and making the application more valuable.

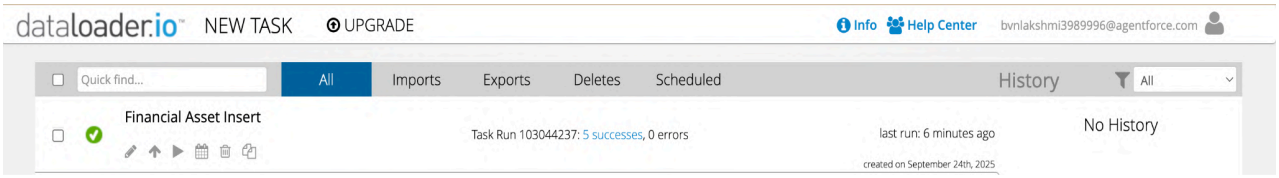
Phase 8: Data Management & Deployment

Data Loader & Change Sets :









I successfully used the Salesforce Data Loader to perform a two-step data import. First, I imported the Investment Portfolio parent records. After resolving a common user lookup error by updating the CSV file and using the correct mapping, I then imported the Financial Asset child records, successfully linking them to their parents. I also explained how to use a Change Set to move all project components from a development environment to a production environment.

Data import is essential for populating the system with a realistic amount of data for testing and demonstration. Demonstrating the use of Data Loader and a Change Set shows proficiency in critical project management and deployment skills, ensuring the system can be efficiently maintained and scaled for future use.

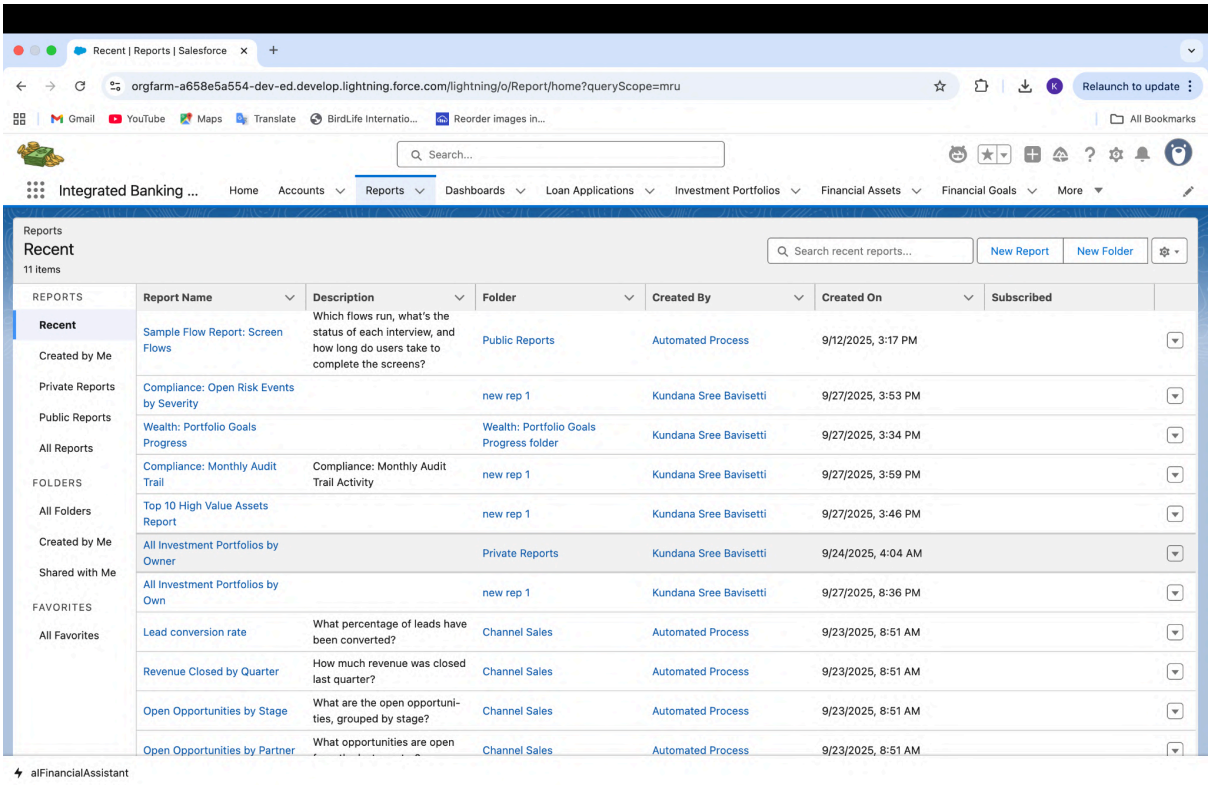
Phase 9: Reporting, Dashboards & Security Review

Implementation: Created two key reports (High-Risk Loan Applications, Portfolio Value by Region) and combined them into a **Dynamic Dashboard**.

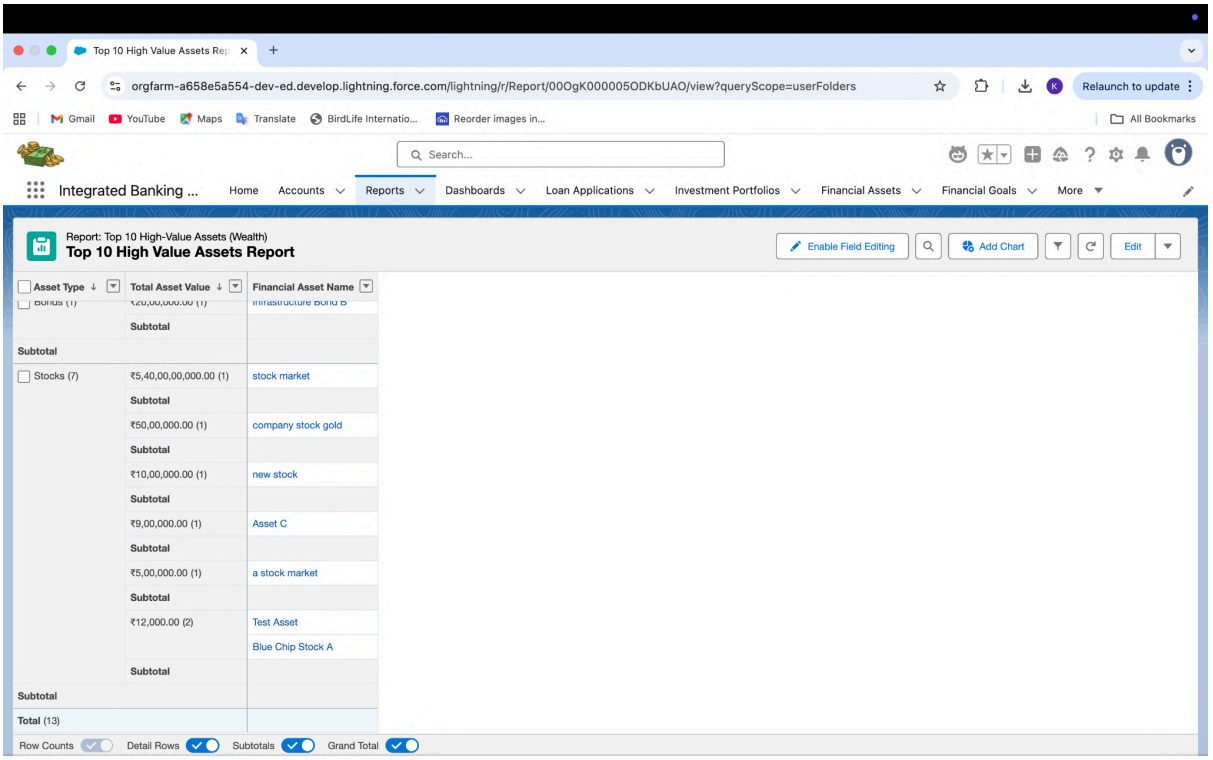
Business Impact: Provides real-time, visual oversight of organizational performance and risk exposure to executive stakeholders.

Creating the Reports:

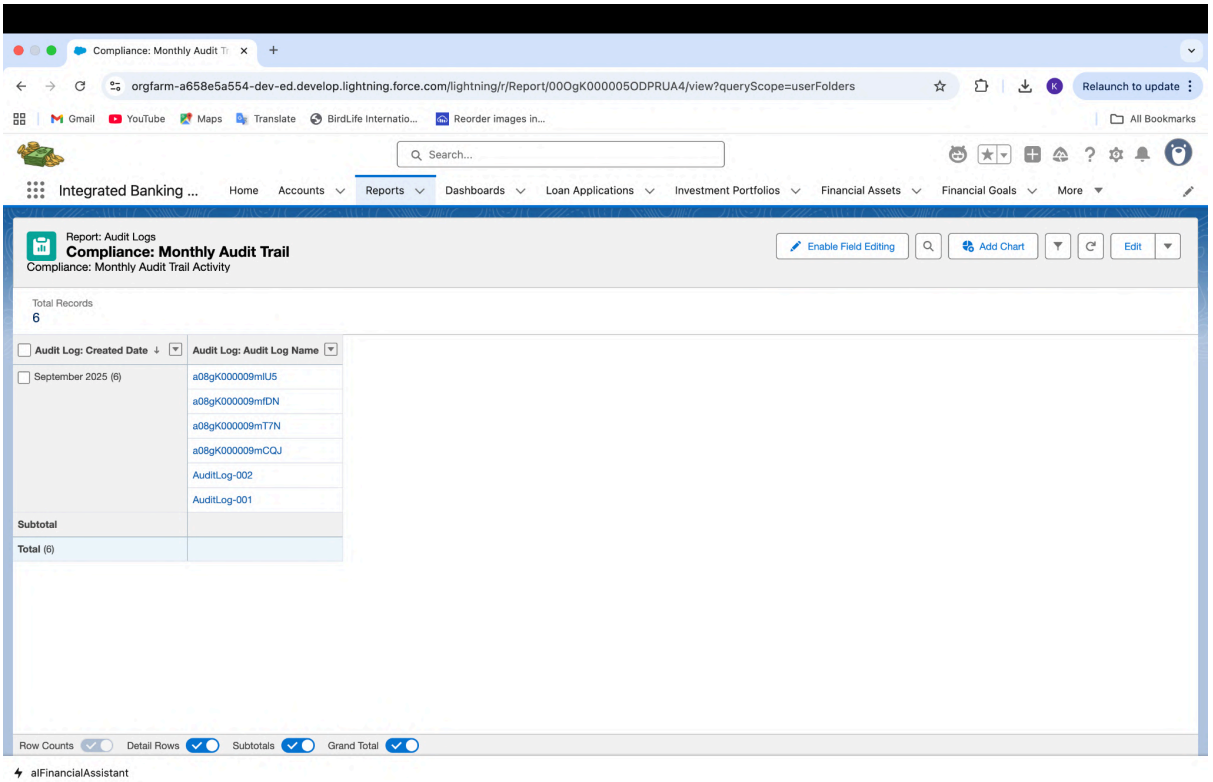
The below image shows all the reports



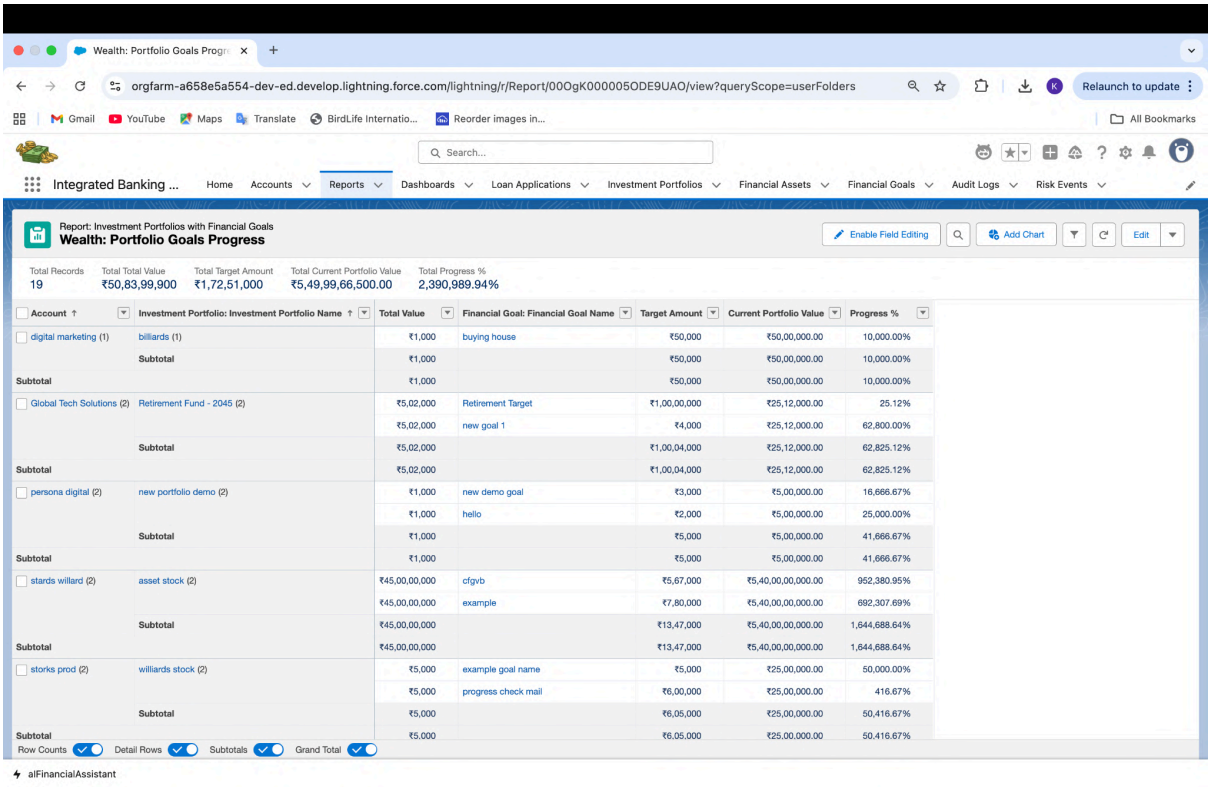
Top 10 High Value Assets Report :



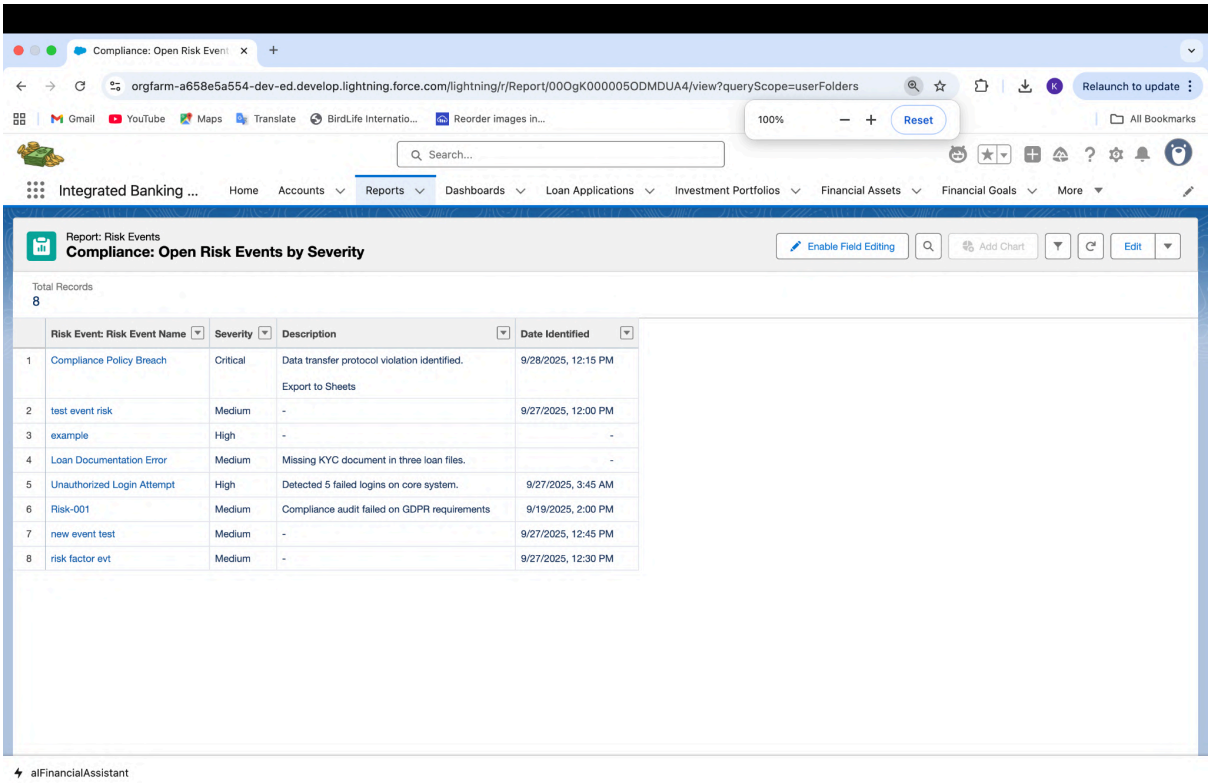
Compliance: Monthly Audit Trail :



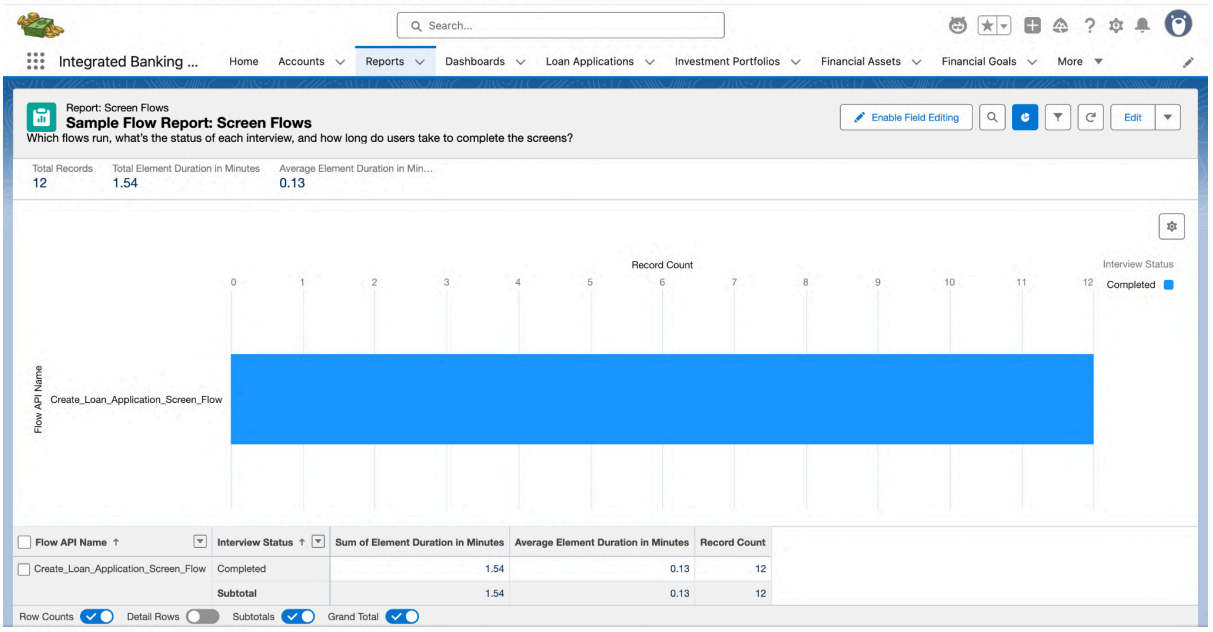
Wealth: Portfolio Goals Progress :



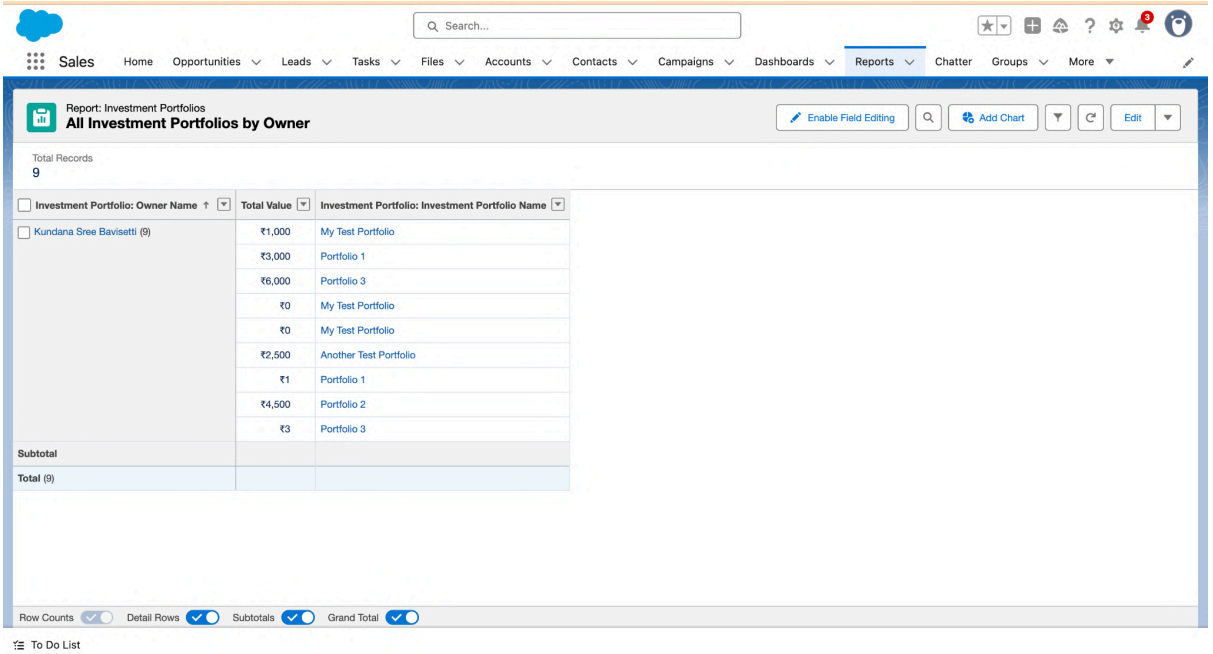
Compliance: Open Risk Events by Severity :



Sample Flow Report: Screen Flows :



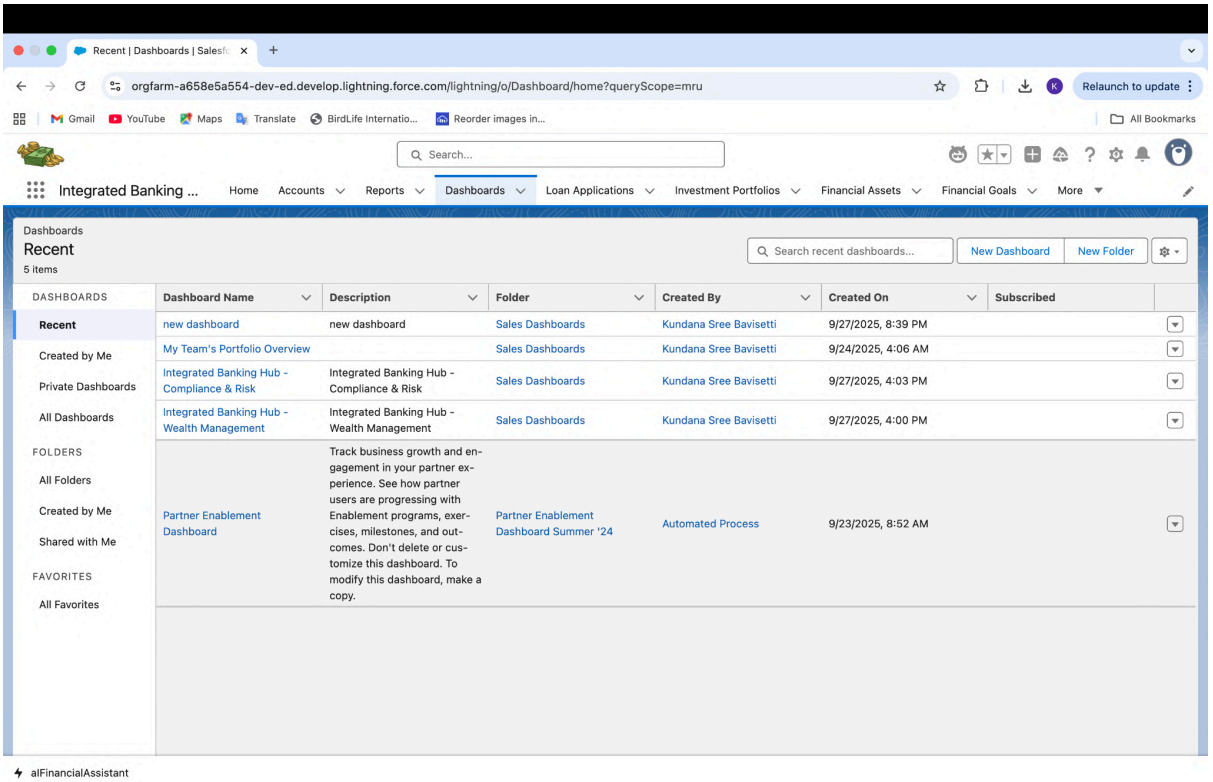
All Investment Portfolios by Owner :



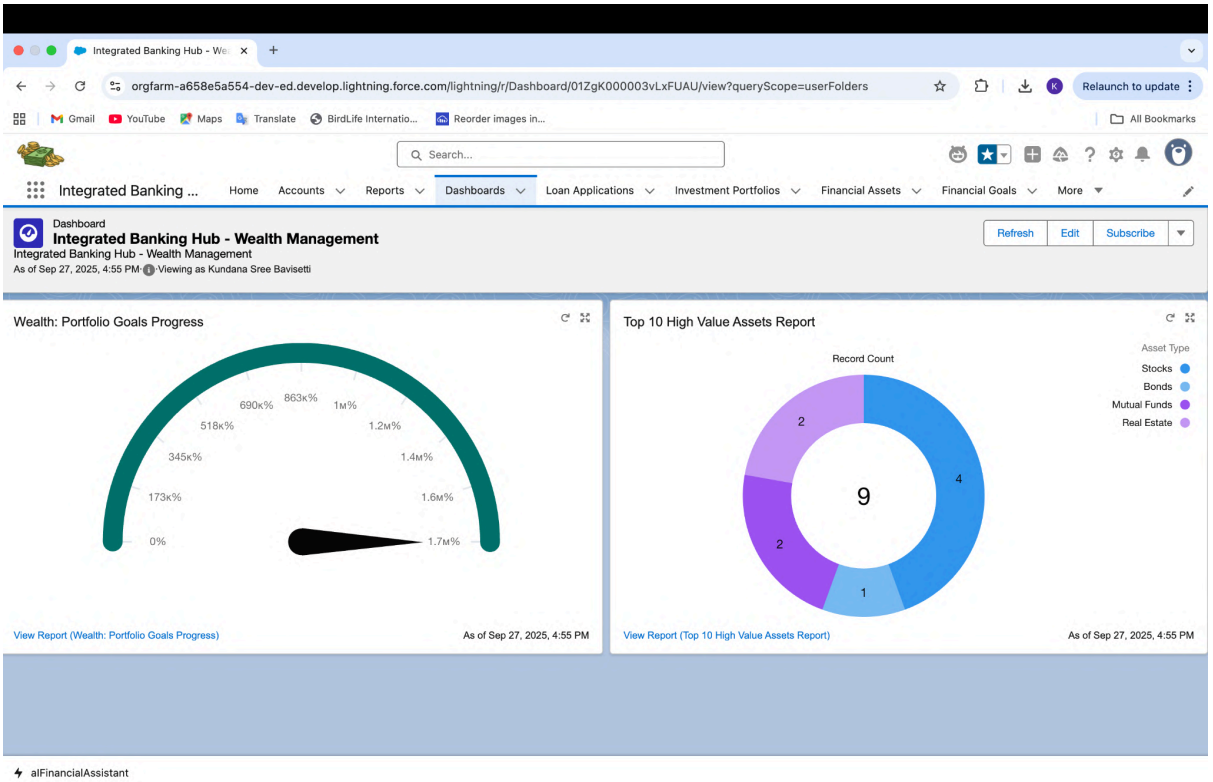
The **Report Builder** showing the filters applied (e.g., Status = 'Rejected' or Loan Amount > threshold) for the **High-Risk Loan Applications Report**.
This report is built on the Loan Application object and specifically filters for high-risk criteria, providing Compliance Officers with an immediate, actionable list of items requiring audit or follow-up

Creating the Dynamic Dashboards:

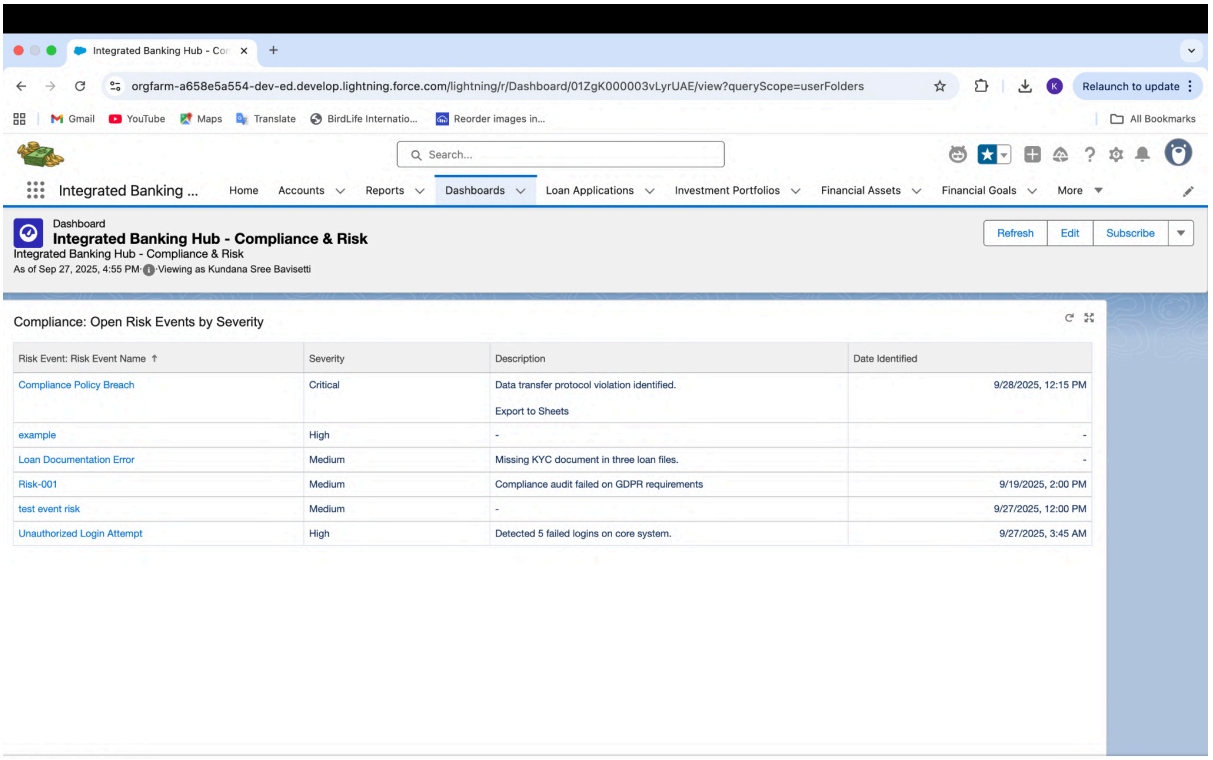
The below image shows all the reports



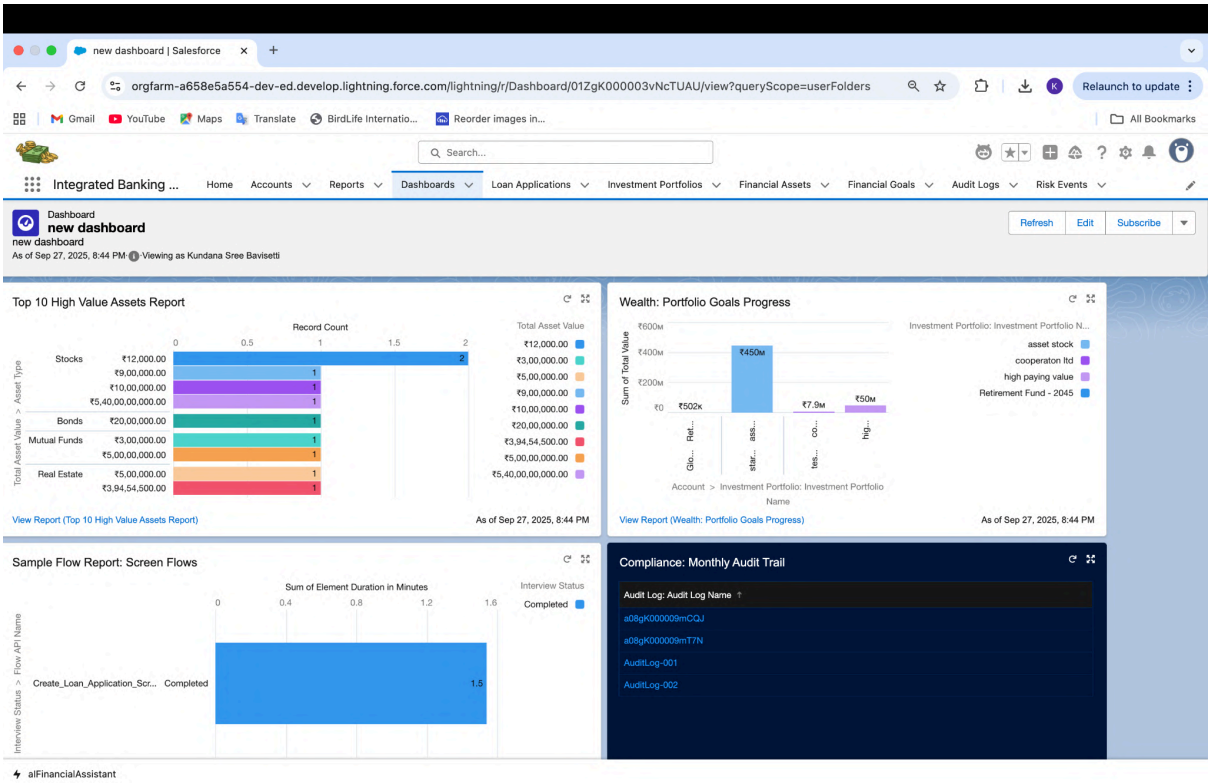
Integrated Banking Hub - Wealth Management :



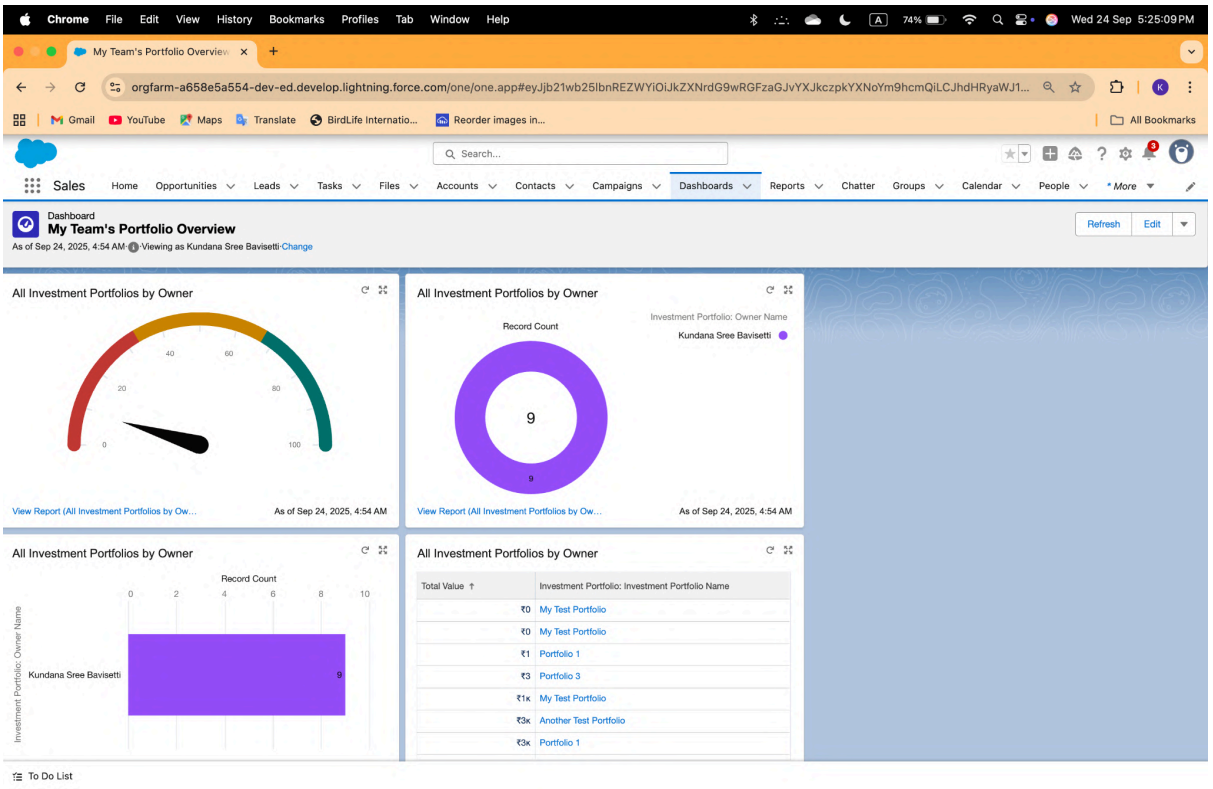
Integrated Banking Hub - Compliance & Risk :



new dashboard :



My Team's Portfolio Overview :

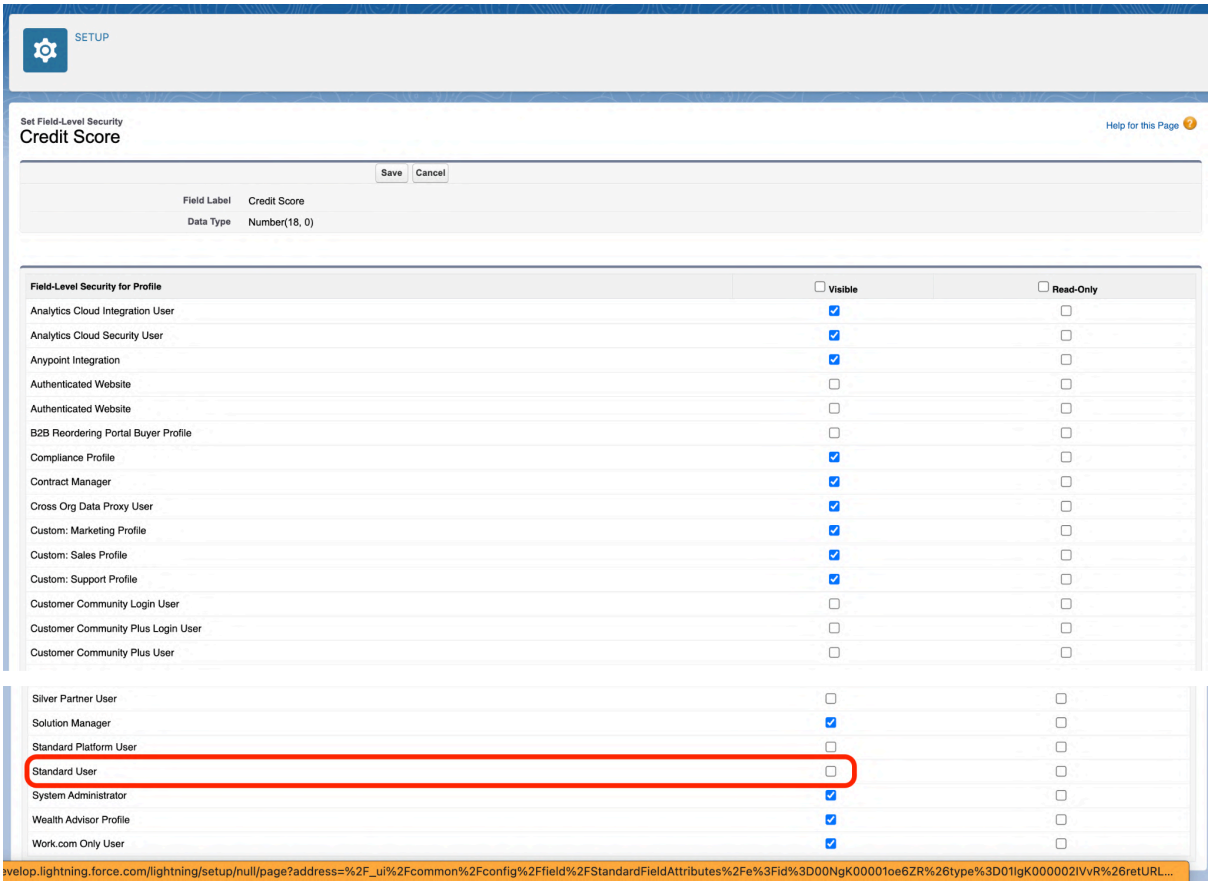


I created a dynamic dashboard titled My Team's Portfolio Overview. This dashboard uses a single source report, All Investment Portfolios by Owner, to create multiple visualizations. I added a Metric component to show the total value, a Gauge to visualize progress towards a goal, a Bar Chart for a breakdown by owner, and a Donut Chart to show the proportion of each owner's assets.

This dashboard provides a secure and personalized view of key financial data. By making it dynamic, each viewer (e.g., a manager) sees only the data relevant to their team, which is a key security and efficiency feature. Using multiple components from a single report demonstrates a strong understanding of how to present complex data in a simple and visually appealing way, making the information more actionable for users.

Field-Level Security (FLS)

- **Implementation:** Implemented **Field-Level Security (FLS)** on sensitive fields (e.g., Credit Score, Portfolio Value) to restrict visibility based on user Profile (e.g., only Managers can see the Credit Score field).
- **Business Impact:** Ensures strict data privacy and regulatory compliance (e.g., KYC/AML).

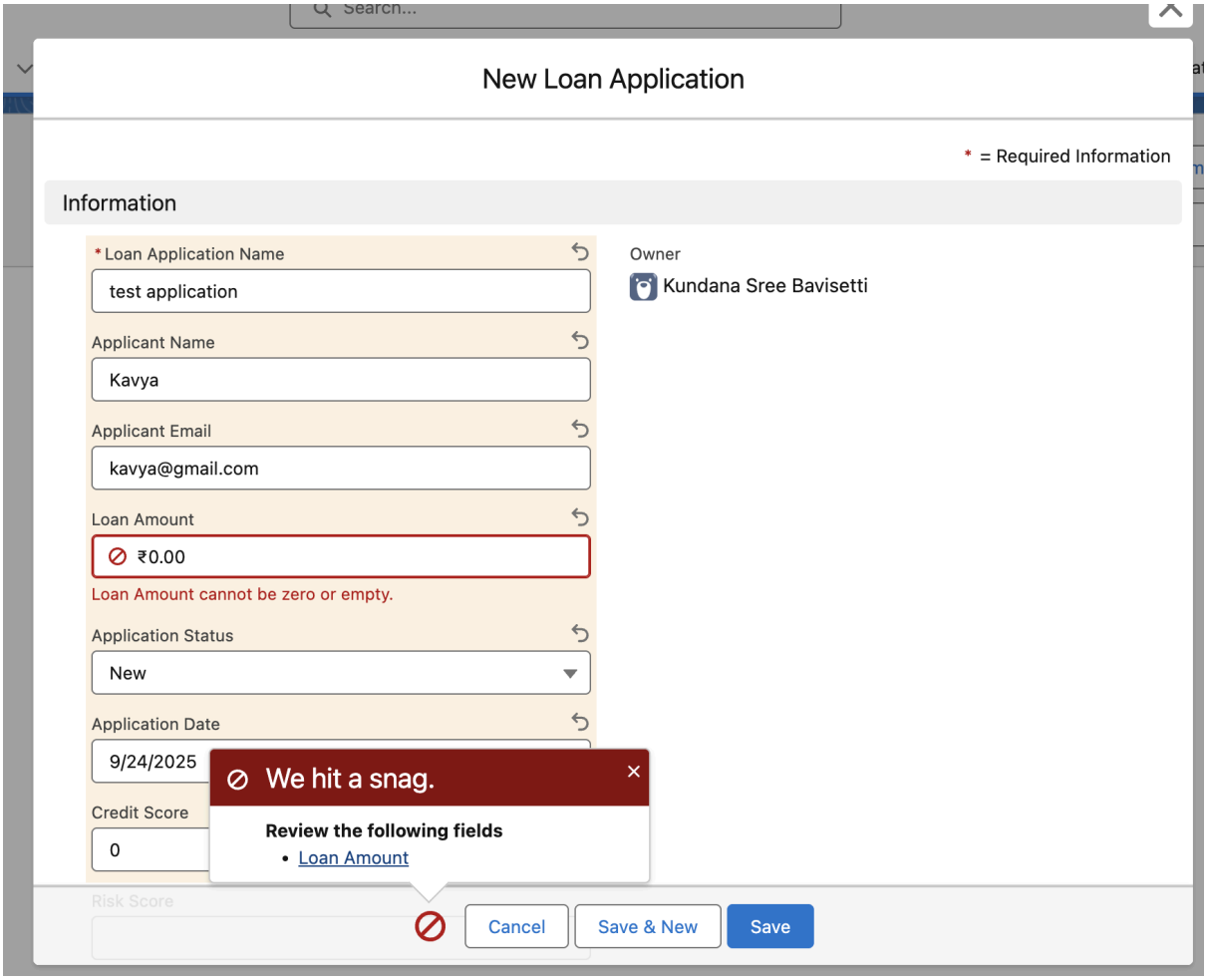


I configured Field-Level Security (FLS) on the Loan Application object. Specifically, I hid the sensitive Credit Score field from the **Standard User** profile.
Field-Level Security is a crucial part of a robust security model. It ensures that sensitive or confidential data is only accessible to users with the appropriate permissions, preventing unauthorized access and maintaining data integrity.

Phase 10: Quality Assurance Testing

Test Case 1: Loan Application Validation Rule

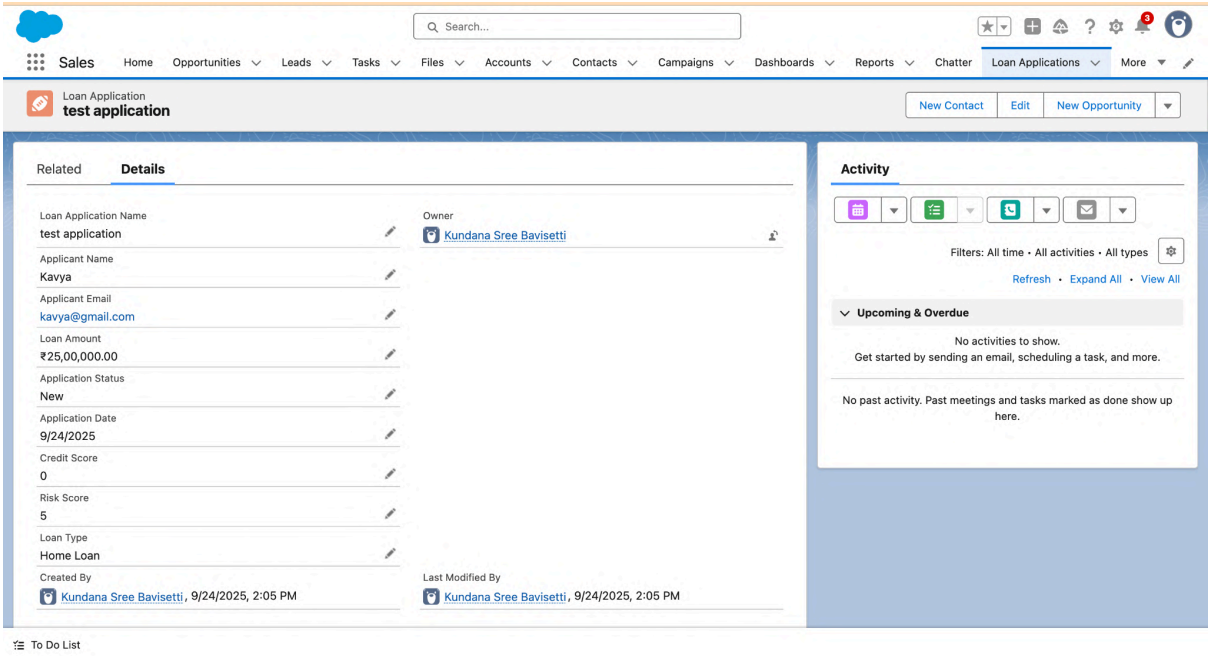
Scenario: Test that the Loan_Amount_Cannot_Be_Zero validation rule prevents a user from creating a Loan Application record with a loan amount of zero or a negative value.



Result: The record is not getting saved and an error message is appearing which I configured in the validation rule. This screenshot visually confirms the test failure: the system correctly displayed the configured error message and prevented the record from being saved. **Expected behavior confirmed.**

Test Case 2: Loan Application Approval Process for high value loans

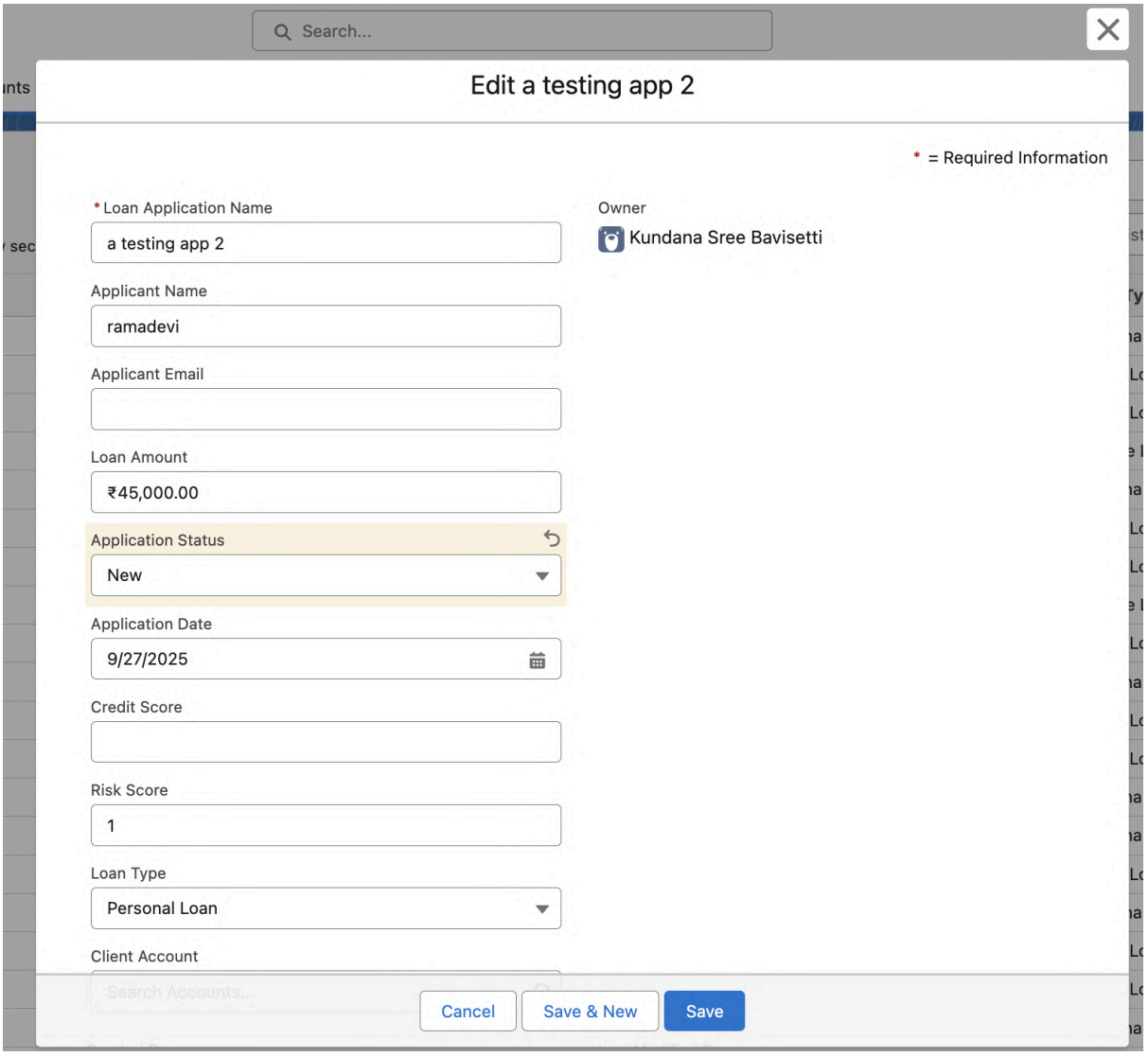
Scenario: Test that the **Loan Approval Process** correctly triggers for high-value loans (Loan Amount $\textgreater 20,00,000$) and automatically locks the record.

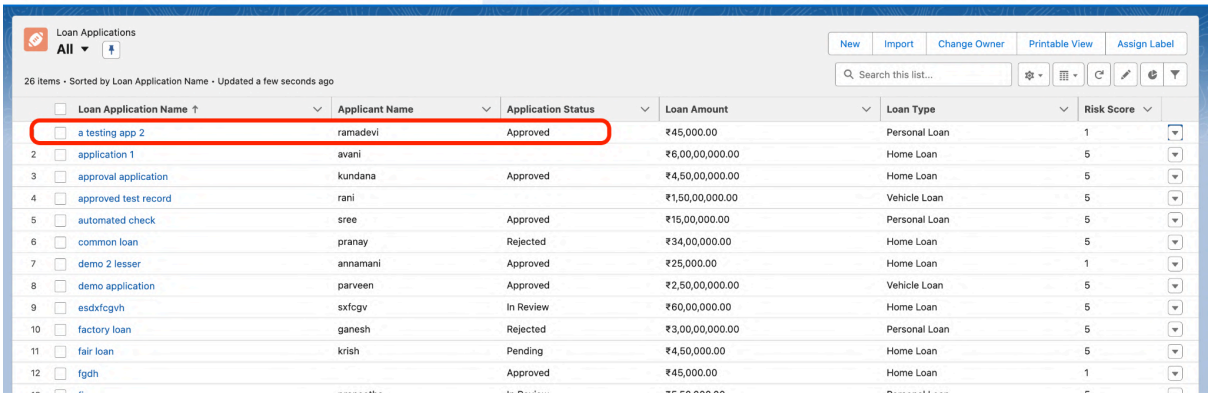


Result: The test passed. The high-value loan successfully triggered the automated approval process and locked the record. This screenshot is the successful output of the Approval Process test. It proves the record entered the correct queue, the status was updated to Pending, and the record is currently locked, achieving the risk mitigation goal

Test Case 3 : Loan Application Approval Process for low value loans

Scenario : This shows the screenshot of just before the record got created the application status is **new** , with loan amount less than 20 lakhs.



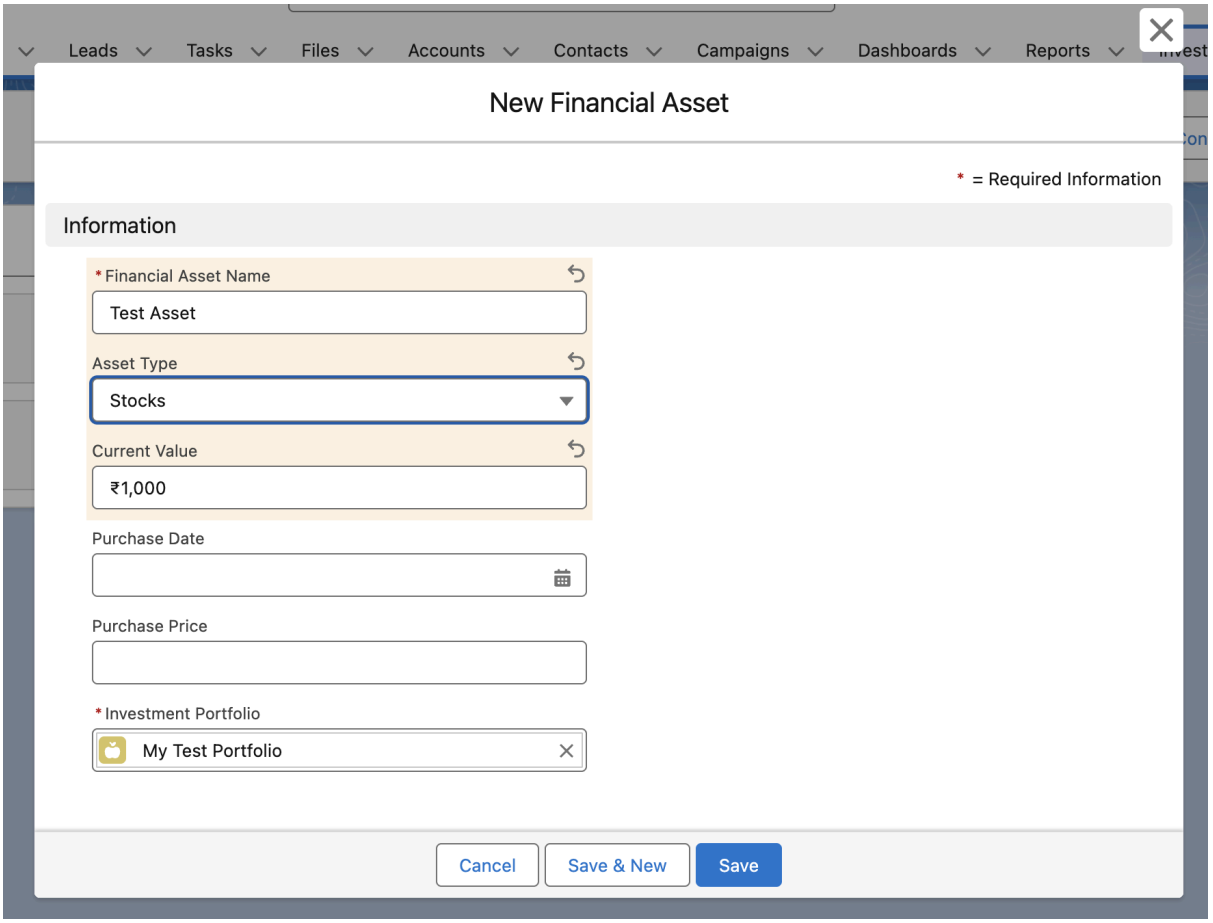


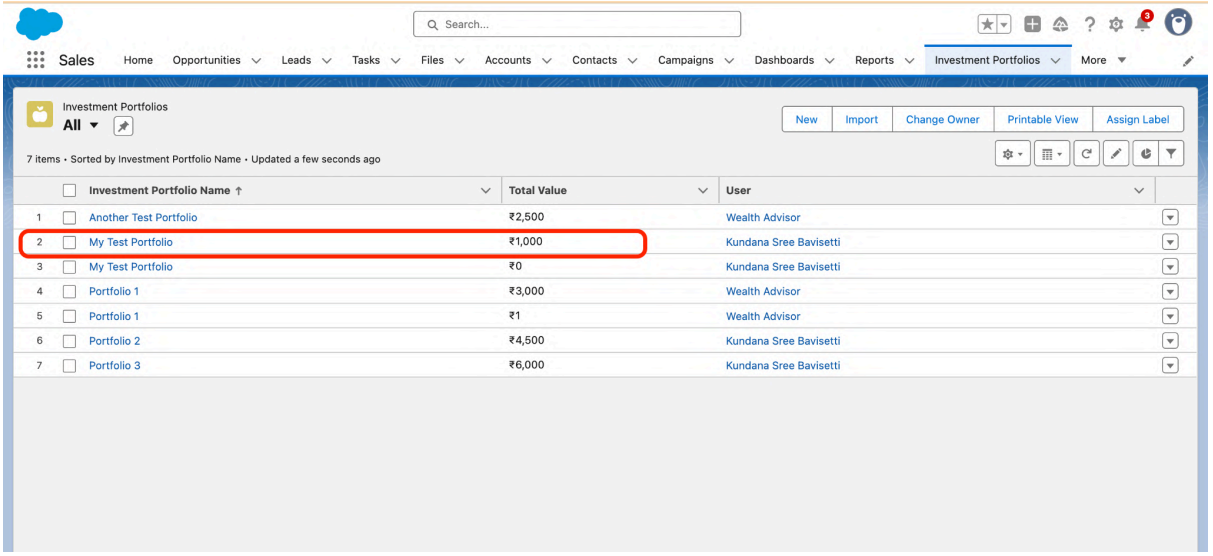
Result : For low-value loans, the Intelligent Routing Flow skips the Approval Process entirely. The status goes directly to 'Approved' with loan amount less than 20 lakhs. This streamlines low-risk operations and improves client response time.

Test Case 4: Apex Trigger

Scenario:

Test that the FinancialAssetTrigger automatically calculates and updates the Total Value of an Investment Portfolio whenever a new Financial Asset record is created.

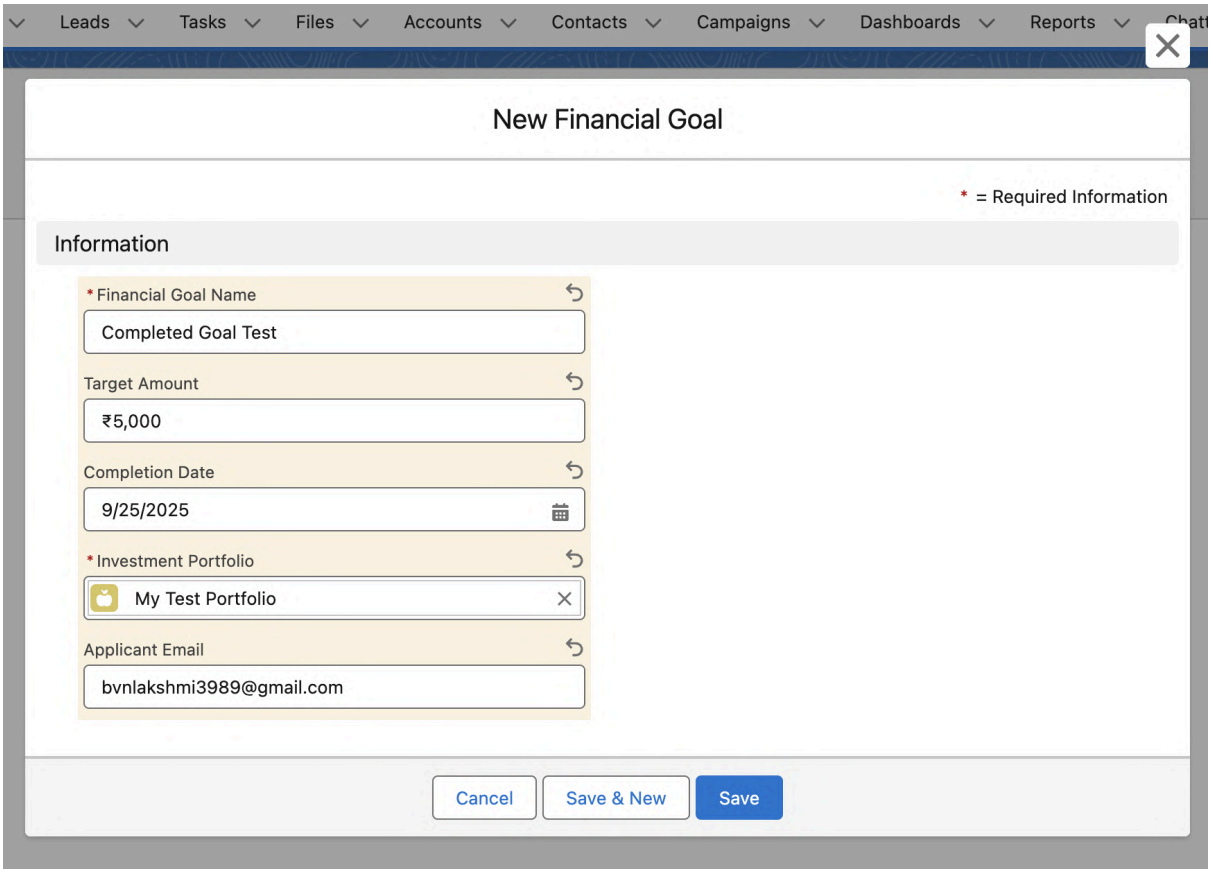


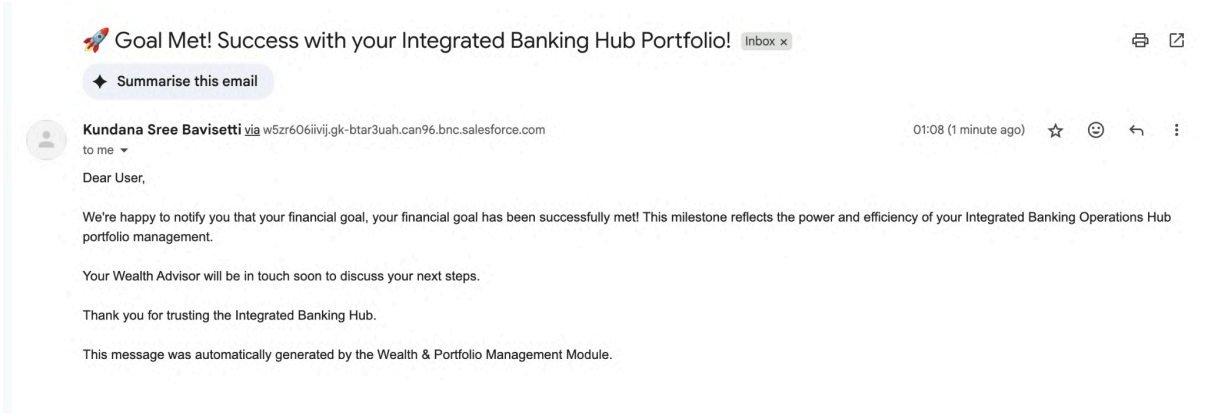


Result: The test passed. The Apex Trigger correctly calculated and updated the parent record's Total Value field. This screenshot is the result of the Apex Trigger execution. It shows the Total_Value__c field on the parent portfolio was correctly updated in real-time, demonstrating successful, robust code execution.

Test Case 5 : Record-Triggered Flow

Scenario: Test that the Record-Triggered Flow on the Financial Goal object correctly sends an automatic email notification to the client when a financial goal is marked as met.

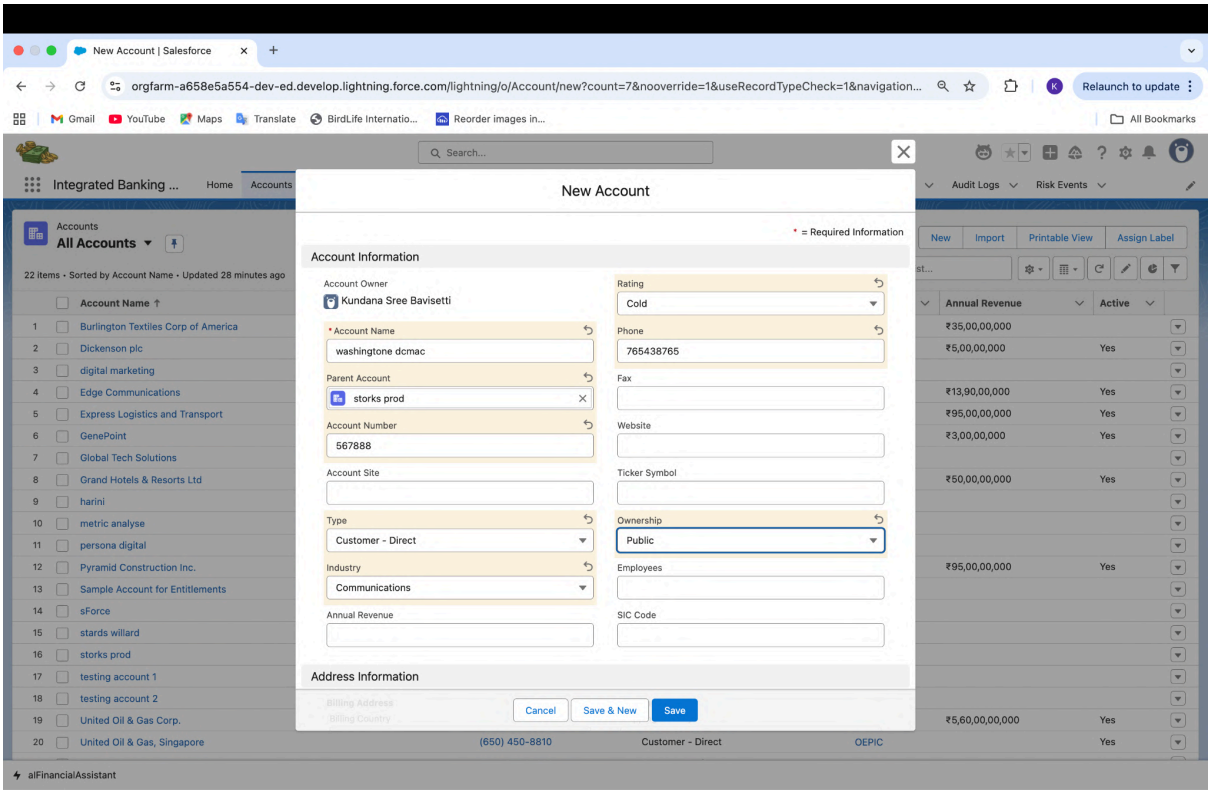


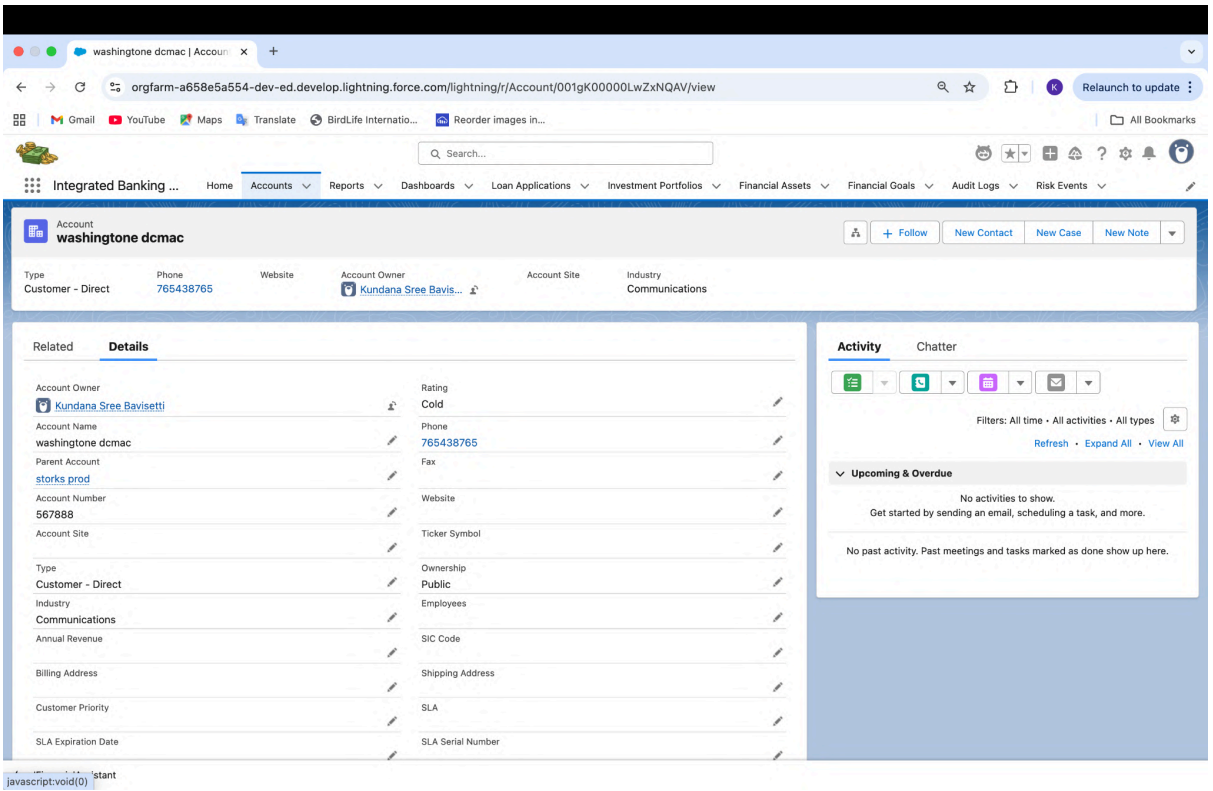


Result: The test passed. The flow correctly triggered upon record update and sent the personalized email notification. The flow debug log confirms the automation sequence: the flow met the criteria (Goal Met) and successfully executed the Send Email action. The test proves the system can automate client communication reliably.

Test Case 6 : Account creation

Scenario : Here we are trying to create a account with some basic details

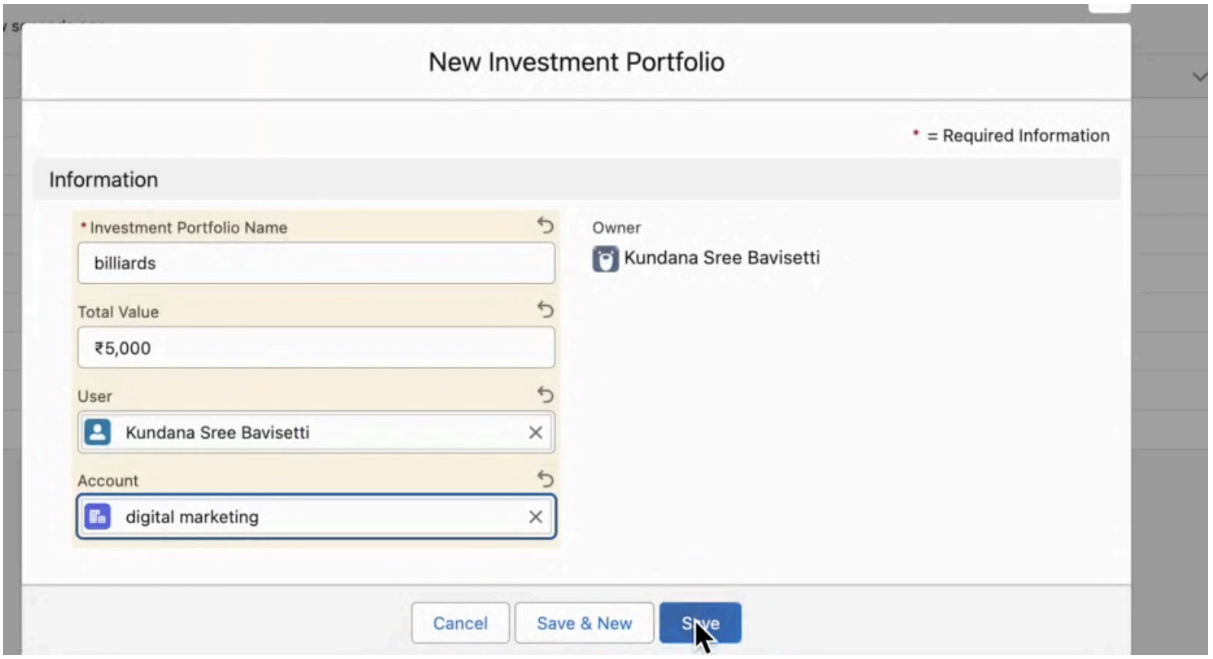


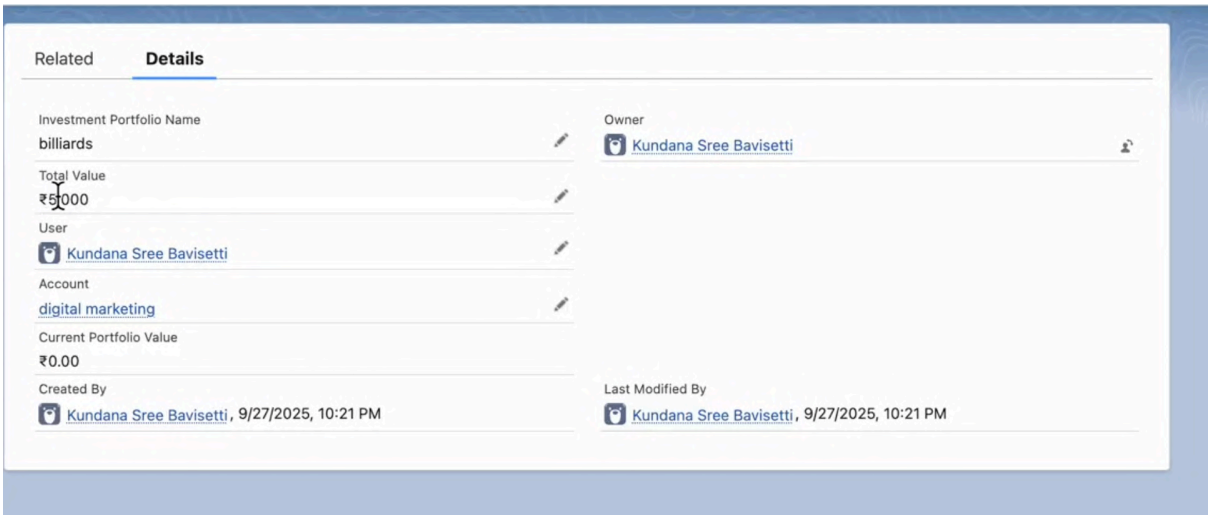


Result: The test passed. Account object successfully created, The test proves the system can automate client communication reliably.

Test Case 7 :

Scenario : Trying to create a investment portfolio with one account and some total value

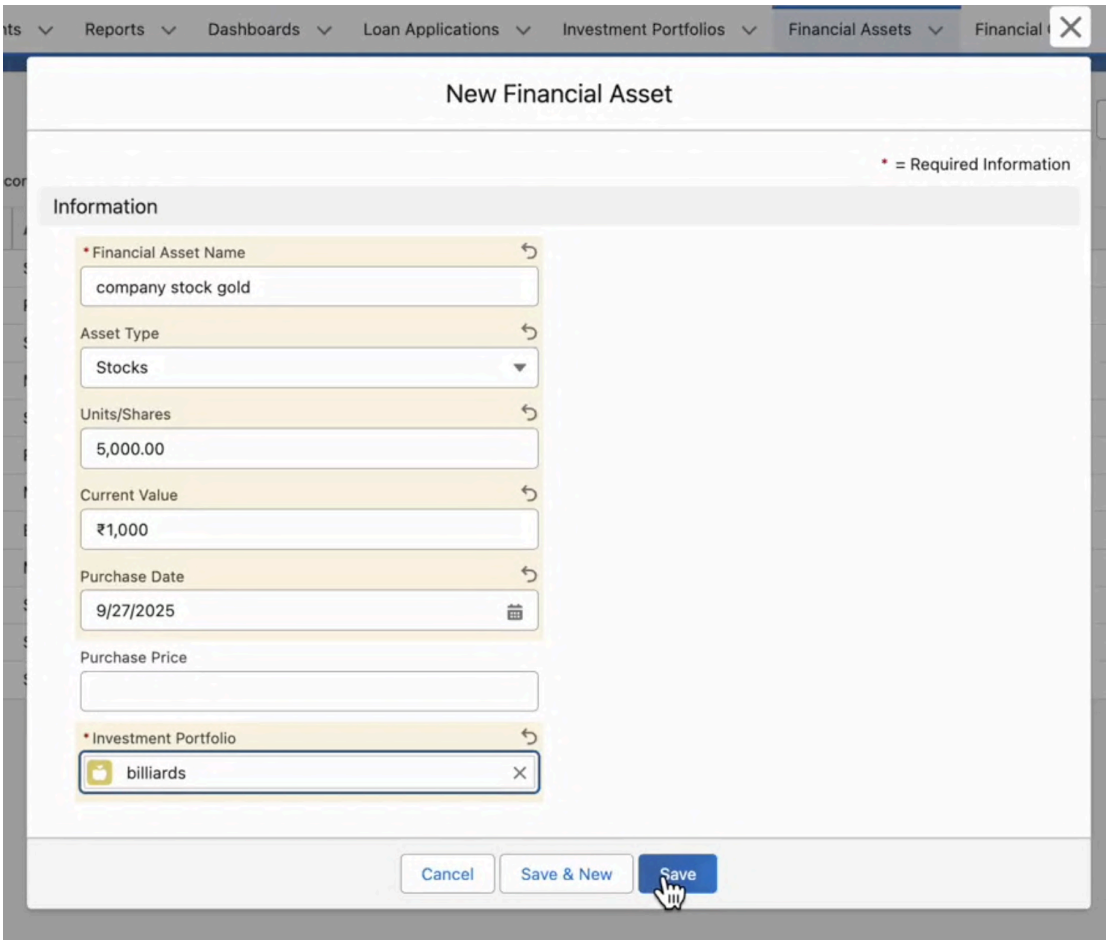




Result : test case passed . successfully created a investment portfolio with one account and some total value.

Test Case 8 :

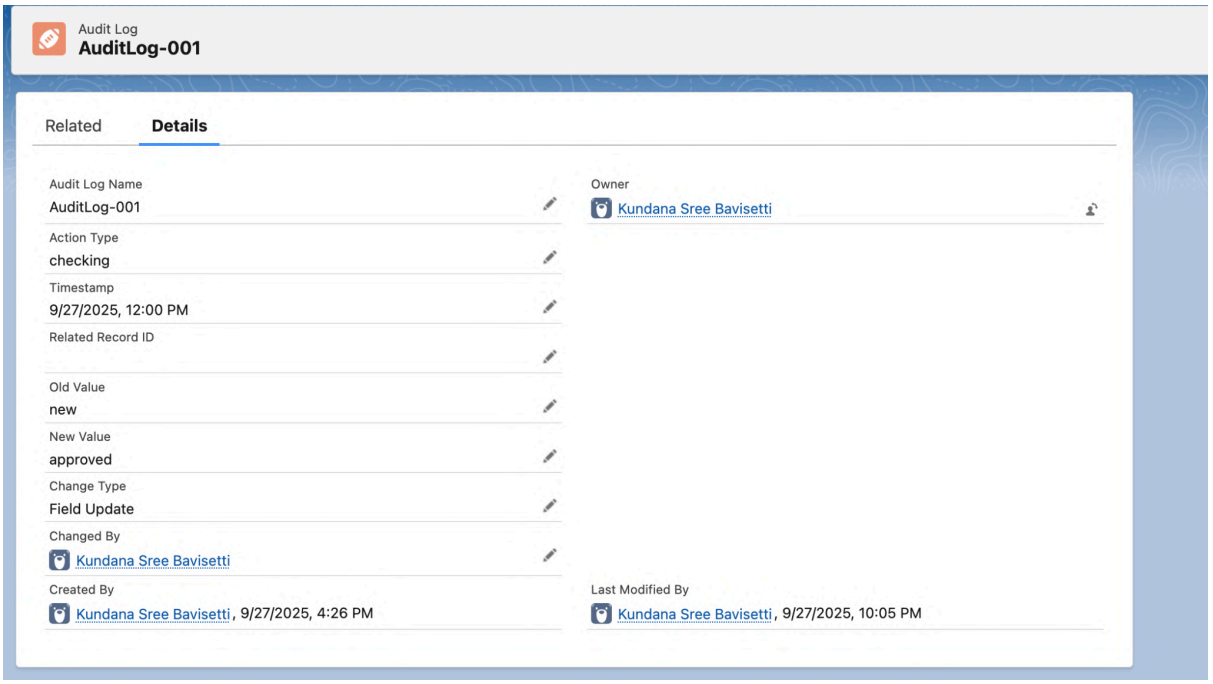
Scenario : here i am trying to create a new financial asset with some basic values



Result : Successfully created the financial asset which is related to the investment portfolio

Test Case 9 :

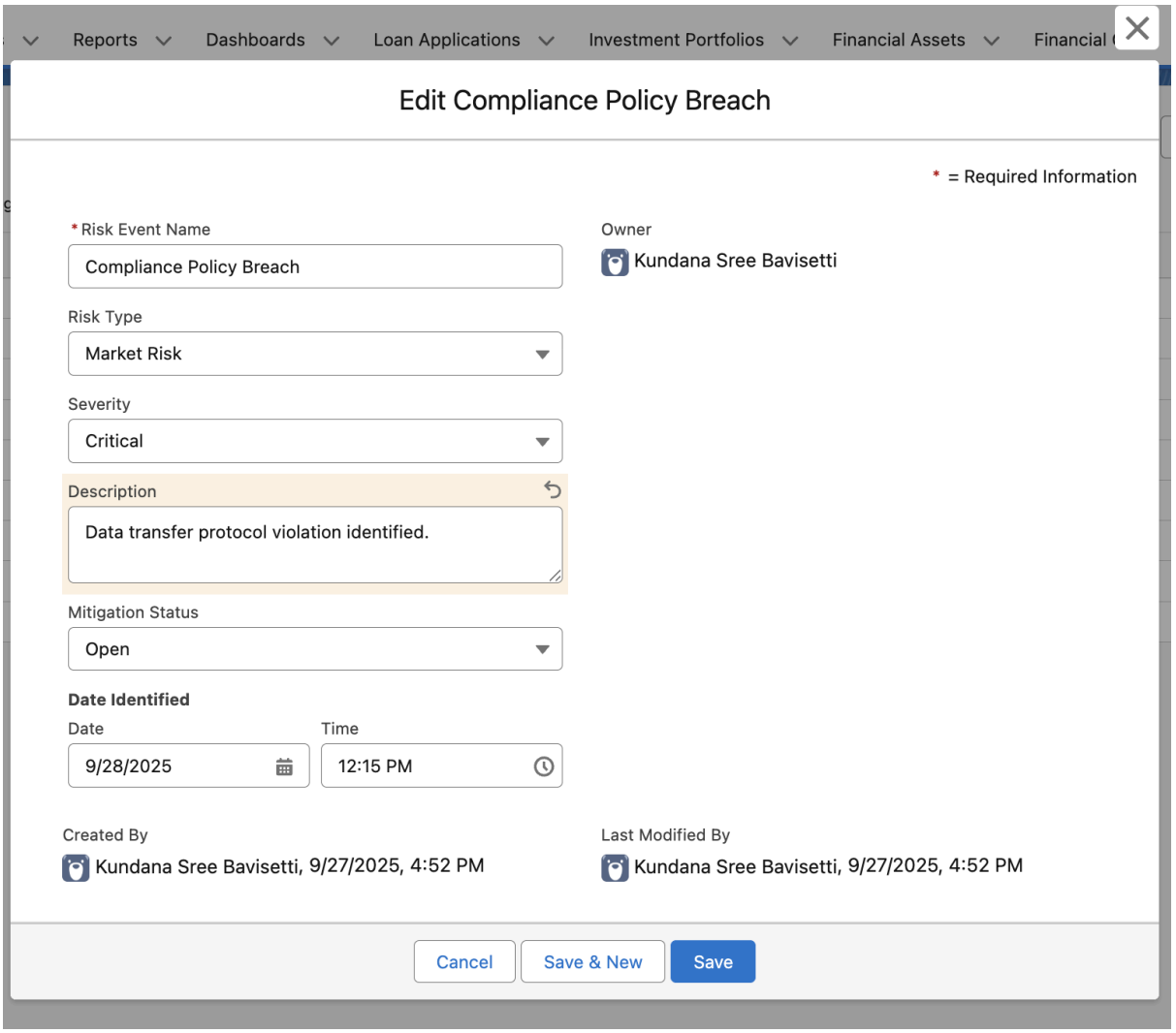
Scenario : checking the status and the audit log after updating the status of the loan application .

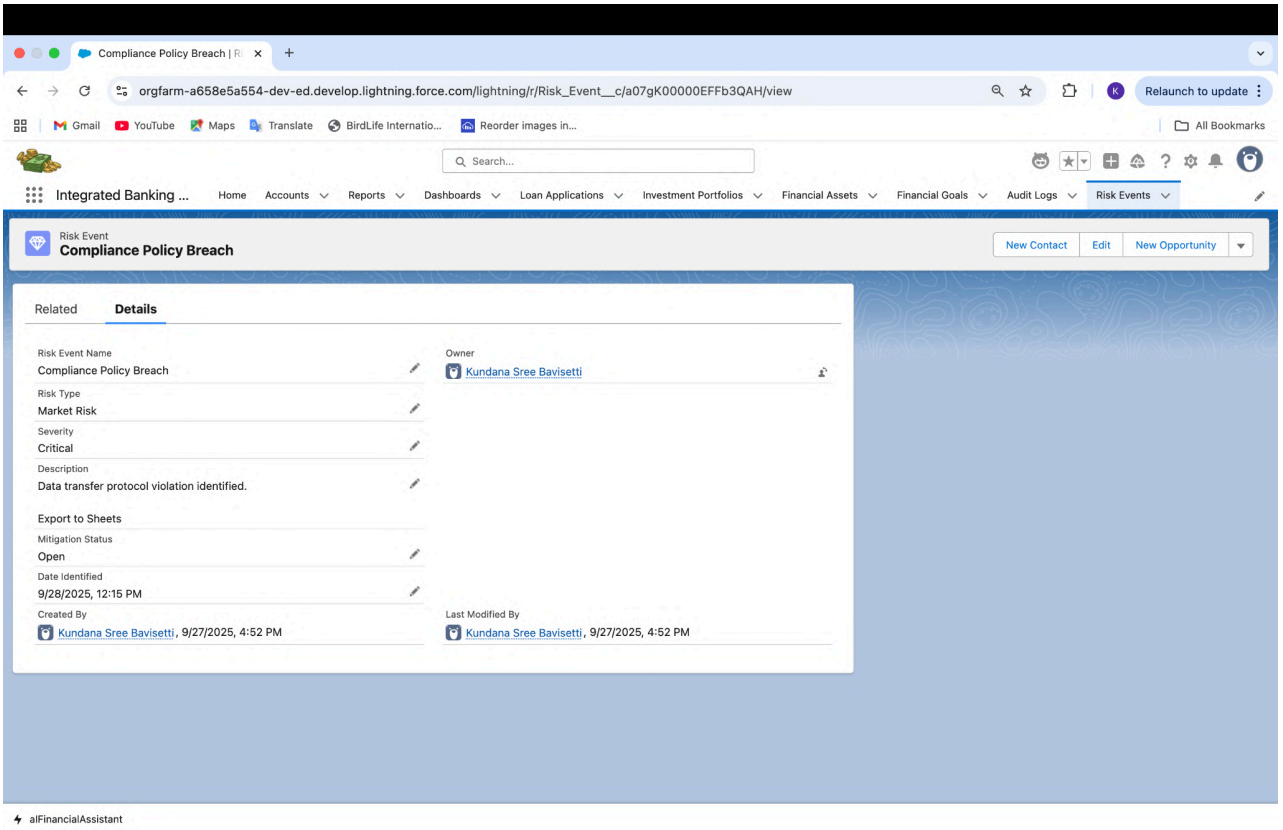


Result : logs were successfully changed after updating the status of the loan application which we have changed.

Test Case 10 :

Scenario : here i am trying to create a new risk event with some basic values .

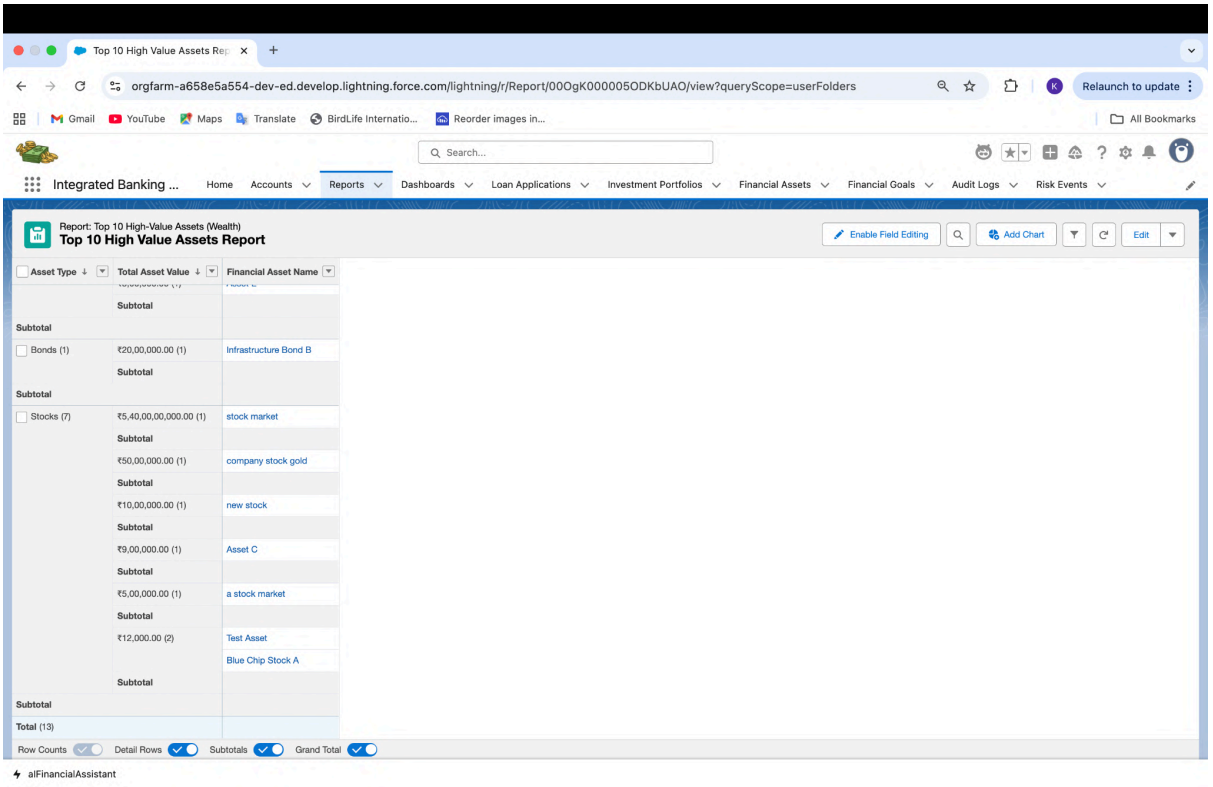




Result : Successfully created the risk event .

Test Case 11 : Real-Time Portfolio Calculation & Reporting

Scenario : Financial Asset Formula (Total_Asset_Value__c) and Apex Trigger Roll-up successfully calculate and aggregate high-value assets for reporting.

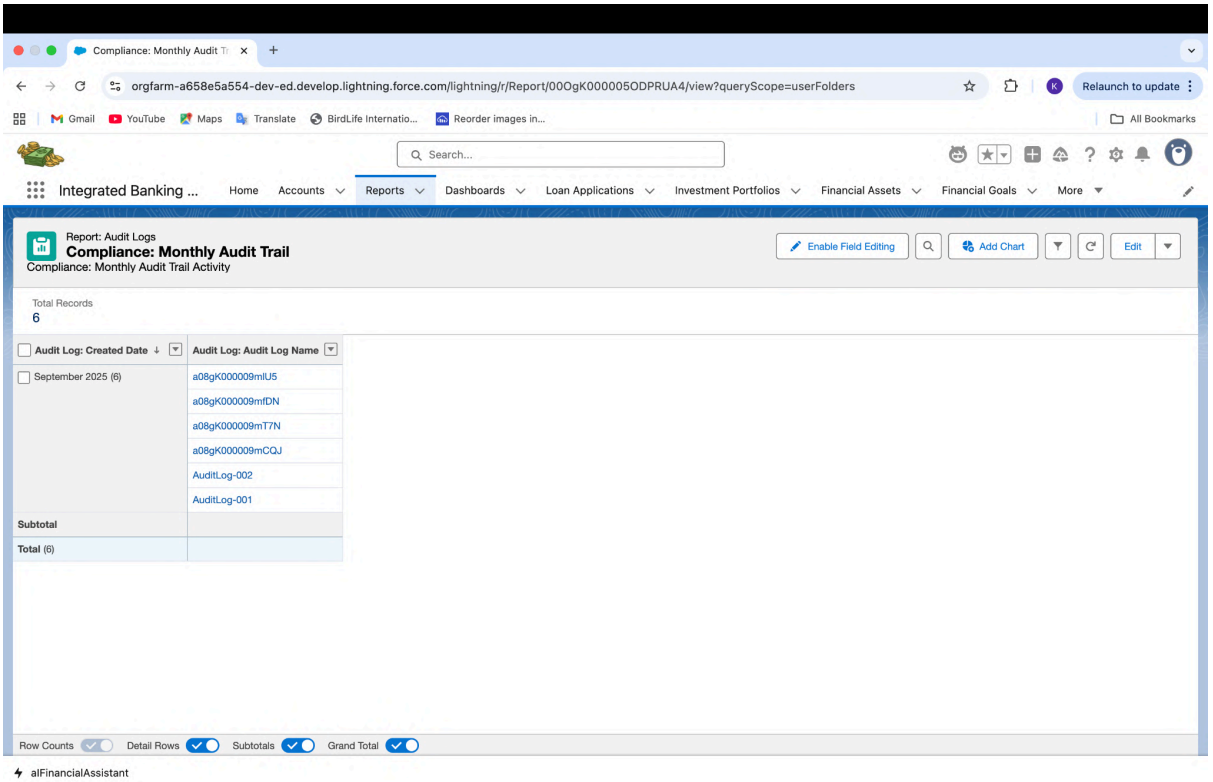


Result: PASS. The test passed. The report successfully retrieves and groups the high-value assets. Assets are correctly categorized (e.g.,

Bonds at ₹20,00,000) and Stocks show accurate aggregated subtotals (e.g., ₹5,40,00,000). This confirms the underlying formula and aggregation logic are sound.

Test Case 12 : Compliance Audit Trail Verification

Scenario : Test the functionality and data integrity of the Audit Log Flow by verifying that status changes are automatically logged and aggregated in the audit report.



Result : PASS. The test passed. The Audit Log Flow successfully executed and logged the change. The report

Compliance: Monthly Audit Trail Activity correctly shows the new Audit Log entries under September (2025).