

Phase 1: Problem Understanding & Industry Analysis

Requirement Gathering

- Identify gaps in manual payment tracking.
- Understand issues in sending timely payment reminders.

Stakeholder Analysis

- Customers, bank staff, recovery agents, and managers.
- Focus on their roles and interaction with the payment system.

Business Process Mapping

- Map repayment lifecycle:
due date → reminder → payment → overdue → recovery.
- Visualize processes for loan and EMI repayments.

Industry-specific Use Case Analysis

- EMI reminders, loan repayments, credit card dues monitoring.
- Streamline overdue and recovery workflows.

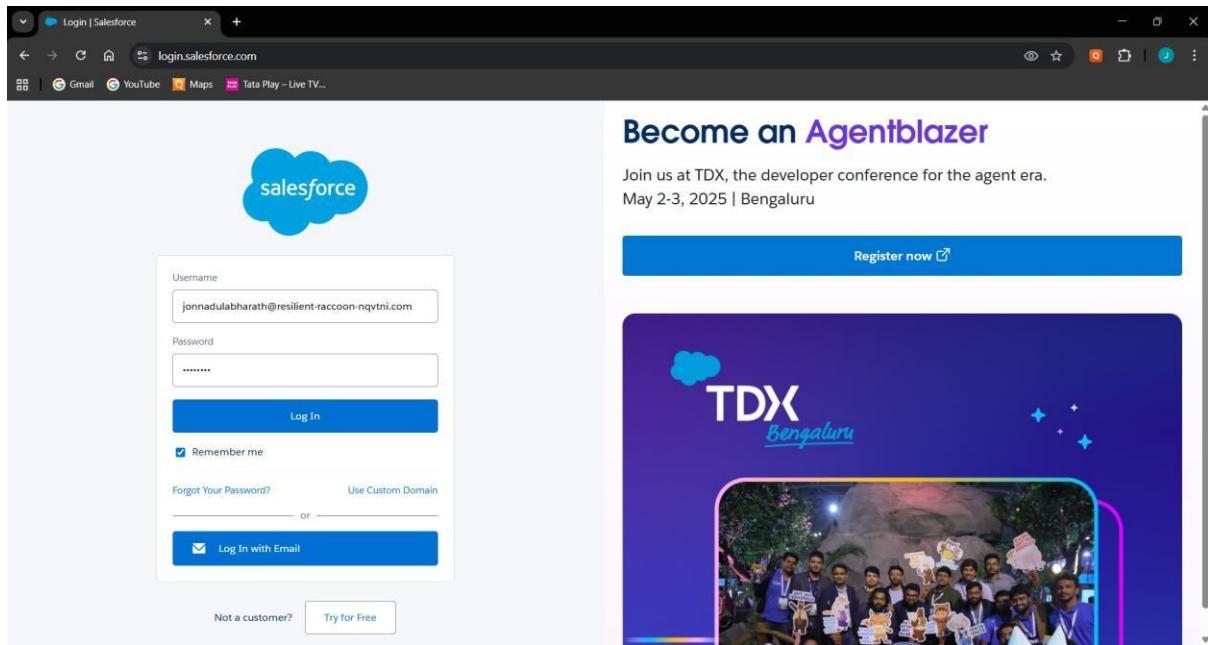
AppExchange Exploration

- Research payment reminder.
- Identify ready-made apps to speed up implementation.

Phase 2: Org Setup & Configuration

Salesforce Editions

- Use Developer Edition for initial development and testing.
- Plan for Enterprise Edition for production banking workflows.



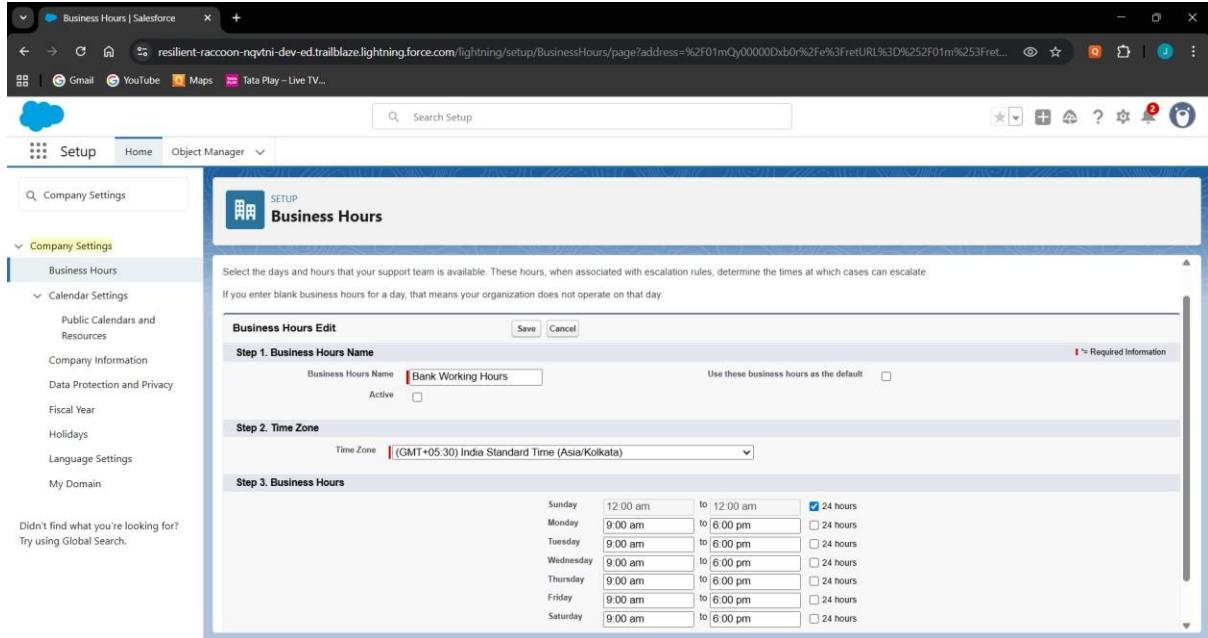
Company Profile Setup

- Set Company Name, Currency (INR/USD), and Time Zone.
- Configure organization-wide defaults.

This screenshot shows the 'Company Information' page in the Salesforce Setup. The top navigation bar includes 'Setup', 'Home', and 'Object Manager'. The left sidebar has sections like 'Company Settings' (selected) and 'Company Information'. The main content area is titled 'Company Information' and shows details for 'ABC Bank Payment Tracking'. It lists organization details such as name, primary contact (Bharath Jonnada), division (IN), fiscal year start (Custom Fiscal Year), and currency (INR). It also shows API usage statistics and a note about restricted logins. Navigation links at the bottom include 'Edit', 'Deactivate Org', and links to 'User Licenses', 'Permission Set Licenses', 'Feature Licenses', and 'Usage-based Entitlements'. A footer at the bottom provides creation and modification details.

Business Hours & Holidays

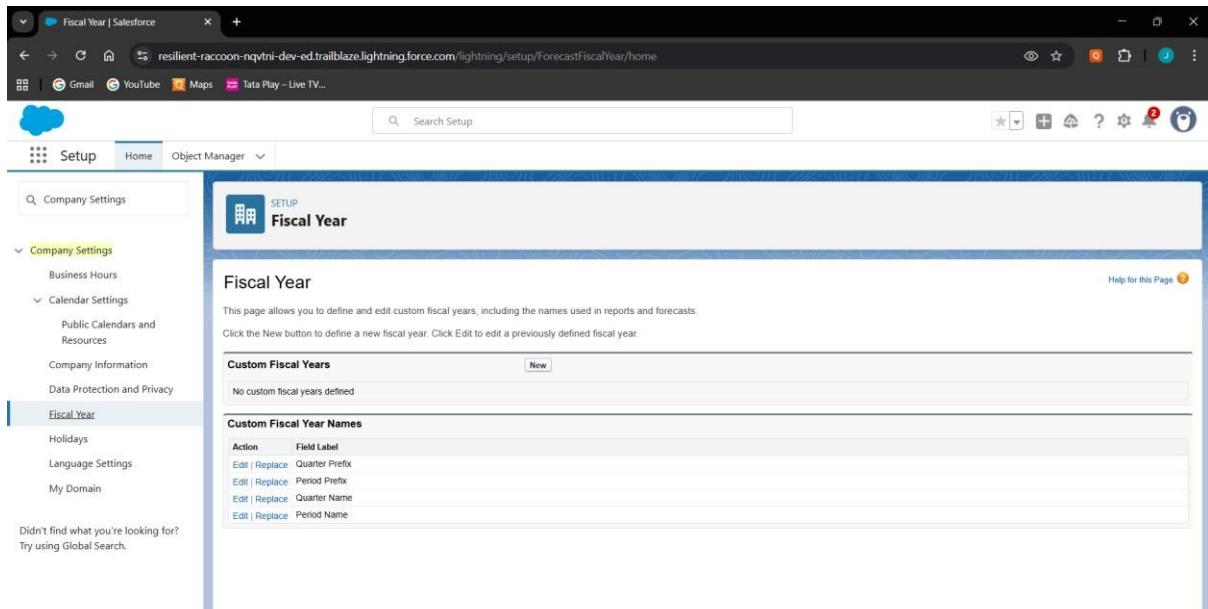
- Define working hours (e.g., Mon–Sat, 9 AM–6 PM).
- Add holidays like Independence Day and Diwali.



The screenshot shows the 'Business Hours' setup page in Salesforce. The left sidebar under 'Company Settings' has 'Business Hours' selected. The main content area is titled 'Business Hours' and contains three steps: Step 1. Business Hours Name (Bank Working Hours, Active), Step 2. Time Zone (GMT+05:30 India Standard Time (Asia/Kolkata)), and Step 3. Business Hours (a grid showing daily working hours from 9:00 am to 6:00 pm, with a '24 hours' checkbox checked for all days). A note at the top says: 'Select the days and hours that your support team is available. These hours, when associated with escalation rules, determine the times at which cases can escalate. If you enter blank business hours for a day, that means your organization does not operate on that day.'

Fiscal Year Settings

- Use standard Jan–Dec fiscal year or custom if needed.
- Align with financial reporting requirements.



The screenshot shows the 'Fiscal Year' setup page in Salesforce. The left sidebar under 'Company Settings' has 'Fiscal Year' selected. The main content area is titled 'Fiscal Year' and contains sections for 'Custom Fiscal Years' (a table showing none defined) and 'Custom Fiscal Year Names' (a table showing columns: Action, Field Label, Edit | Replace, Quarter Prefix, Period Prefix, Edit | Replace, Quarter Name, Period Name). A note at the top says: 'This page allows you to define and edit custom fiscal years, including the names used in reports and forecasts. Click the New button to define a new fiscal year. Click Edit to edit a previously defined fiscal year.'

User Setup & Licenses

- Create Loan Officer, Recovery Agent, and Manager users.
- Assign appropriate licenses (Salesforce / Salesforce Platform).

The screenshot shows the Salesforce Setup interface under the 'Users' section. A new user record is being created for 'Branch Manager'. The 'General Information' tab is selected, displaying fields for First Name ('Branch'), Last Name ('Manager'), Alias ('manager'), Email ('bankmanager.officer@bank'), Username ('bankmanager.officer@bank'), Nickname ('manager'), Title (''), Company (''), Department (''), and Division (''). On the right side, the 'Role' is set to '<None Specified>', 'User License' to 'Salesforce', 'Profile' to 'Manager Profile', and 'Active' is checked. Other optional checkboxes like Marketing User, Offline User, Knowledge User, Flow User, Service Cloud User, Site.com Contributor User, Site.com Publisher User, WDC User, and Data.com User Type are available but unchecked. A note at the bottom indicates a monthly addition limit of 300.

The screenshot shows the Salesforce Setup interface under the 'Users' section. A new user record is being created for 'Loan Officer'. The 'General Information' tab is selected, displaying fields for First Name ('Loan'), Last Name ('Officer'), Alias ('loan'), Email ('loan.officer@bankdemo.cor'), Username ('loan.officer@bankdemo.cor'), Nickname ('loanofficer'), Title (''), Company (''), Department (''), and Division (''). On the right side, the 'Role' is set to '<None Specified>', 'User License' to 'Salesforce Platform', 'Profile' to 'Standard Platform User', and 'Active' is checked. Other optional checkboxes like Marketing User, Offline User, Knowledge User, Flow User, Service Cloud User, Site.com Contributor User, Site.com Publisher User, WDC User, and Data.com User Type are available but unchecked. A note at the bottom indicates a monthly addition limit of 300.

Profiles & Roles

- Create custom profiles (Loan Officer, Manager, Recovery Agent).
- Establish role hierarchy (Manager → Loan Officer / Recovery Agent).

Permission Sets

- Create a Permission Set for payment tracking access.
- Assign Permission Set to Loan Officers and Managers.

The screenshot shows the 'Permission Sets' page in the Salesforce Setup. A permission set named 'Payment Team Access' is selected. The 'Permission Set Overview' section displays details like API Name (Payment_Team_Access), Namespace Prefix, and creation and modification history. The 'Apps' section lists various app settings such as Assigned Apps, Assigned Connected Apps, Object Settings, App Permissions, Apex Class Access, and Visualforce Page Access.

OWD & Sharing Rules

- Set custom objects (Loan, Payment, Recovery) to Private.
- Create rules to share overdue records with Recovery Agents.

The screenshot shows the 'Sharing Settings' page in the Salesforce Setup. It displays a table of sharing rules for various objects. The columns show the object name, sharing rule type (e.g., Private, Public Read Only), and sharing rule status (indicated by checkmarks). Objects listed include Quick Text Usage, Rebate Payout Snapshot, Scorecard, Seller, Service Contract, Streaming Channel, Tableau Host Mapping, Thanks, User Provisioning Request, Web Cart Document, Work Order, Work Plan, Work Plan Template, Work Step Template, Loan, Loan Agent Assignment, Payment, and Recovery.

Object	Type	Status
Quick Text Usage	Private	✓
Rebate Payout Snapshot	Private	✓
Scorecard	Private	✓
Seller	Private	✓
Service Contract	Private	✓
Streaming Channel	Public Read/Write	✓
Tableau Host Mapping	Public Read Only	✓
Thanks	Public Read Only	✓
User Provisioning Request	Private	✓
Web Cart Document	Private	✓
Work Order	Private	✓
Work Plan	Private	✓
Work Plan Template	Private	✓
Work Step Template	Private	✓
Loan	Private	✓
Loan Agent Assignment	Controlled by Parent	✓
Payment	Controlled by Parent	✓
Recovery	Private	✓

Login Access Policies

- Set session timeout to 30 minutes.
- Enable login hours (8 AM–8 PM) for users.

The screenshot shows the 'Session Settings' page in the Salesforce Setup interface. The URL is <https://resilient-raccoon-nqytni-dev-ed.trailblaze.lightning.force.com/lightning/setup/SecuritySession/home>. The left sidebar shows 'Session Management' and 'Session Settings' selected. The main content area is titled 'Session Settings' and contains sections for 'Session Timeout' (timeout value set to '30 minutes') and 'Session Settings' (checkboxes for locking sessions by IP, domain, or password reset). A note at the bottom states: "EXTENDED USE OF IE11 WITH LIGHTNING EXPERIENCE HAS NOW ENDED" and "AS OF DECEMBER 31, THE EXTENDED PERIOD HAS ENDED, AND USE OF INTERNET EXPLORER 11 (IE 11) WITH LIGHTNING EXPERIENCE IS NO LONGER SUPPORTED. ISSUES WITH PERFORMANCE OR FUNCTIONALITY".

Dev Org & Sandbox Usage

- Use Developer Org for development and testing.
- Plan sandbox usage for production-like testing later.

Deployment Basics

- Use SFDX commands to push/pull metadata.
- Prepare for deployment via Change Sets or SFDX when ready.

The screenshot shows the VS Code editor with an Apex class named 'LoanDetailsController.cls'. The code defines a static method 'getRecord' that queries a 'Loan__c' record from the database. The code editor has syntax highlighting for Apex and shows line numbers. The Explorer sidebar on the left lists files like 'force-app/main/default/classes/LoanDetailsController.cls', 'loanSummary.html', 'loanSummary.js', and 'loanSummary.js-meta.xml'. The bottom status bar indicates the file is 'Activating Extensions...'.

Phase 3: Data Modeling & Relationships

Standard & Custom Objects

- Use Account/Contact for Customers.
- Create custom objects: Loan, Payment, Recovery, Loan Agent Assignment (junction).

The image displays two side-by-side screenshots of the Salesforce Object Manager setup screen. Both screenshots show the 'Details' tab for a custom object, with the left sidebar collapsed.

Loan Object Setup:

- API Name:** Loan__c
- Singular Label:** Loan
- Plural Label:** Loans

Object Settings (Right Panel):

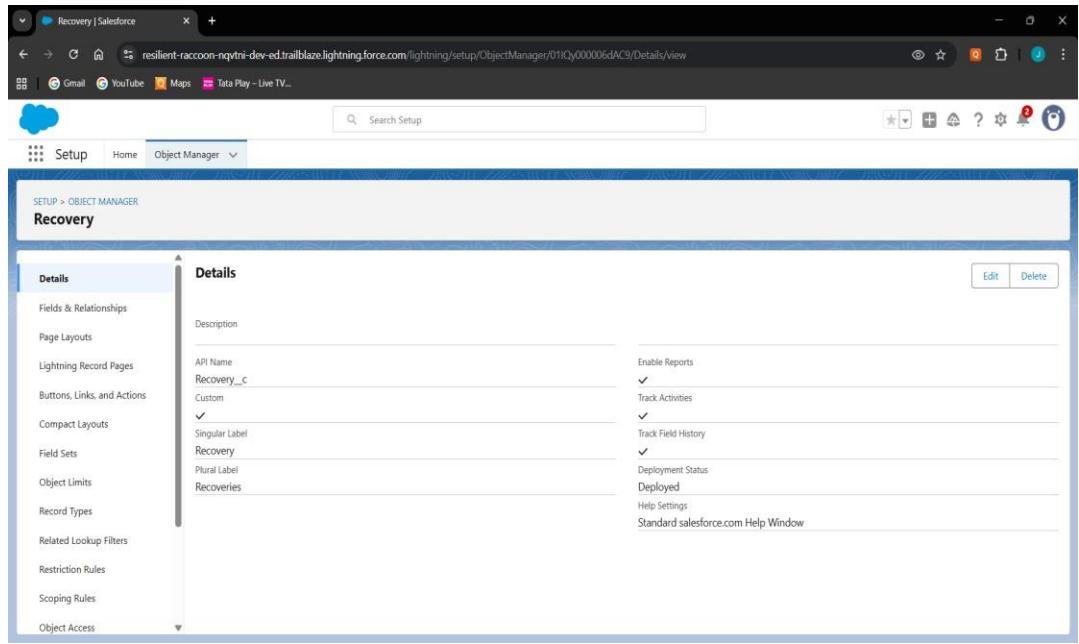
- Enable Reports: ✓
- Track Activities: ✓
- Track Field History: ✓
- Deployment Status: Deployed
- Help Settings: Standard salesforce.com Help Window

Payment Object Setup:

- API Name:** Payment__c
- Singular Label:** Payment
- Plural Label:** Payments

Object Settings (Right Panel):

- Enable Reports: ✓
- Track Activities: ✓
- Track Field History: ✓
- Deployment Status: Deployed
- Help Settings: Standard salesforce.com Help Window



Fields

- Loan: Loan Number (Auto Number), Principal Amount (Currency), Interest Rate (Percent), Term Months (Number), Start Date (Date), Status (Picklist), Customer (Lookup to Account/Contact).
- Payment: Payment Date (Date), Amount (Currency), Payment Method (Picklist: Cash/UPI/Transfer), Status (Picklist: Paid/Overdue), Loan (Master-Detail).
- Recovery: Recovery Date (Date), Amount Recovered (Currency), Status (Picklist: Assigned/InProgress/Completed), Loan (Lookup), Assigned Agent (Lookup to User).
- Loan Agent Assignment (junction): Loan (Master-Detail), Assigned Agent (Lookup to User).

Fields & Relationships					
FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED	
Amount Recovered	Amount_Recovered__c	Currency(16, 2)			
Assigned Agent	Assigned_Agent__c	Lookup(User)			
Created By	CreatedById	Lookup(User)			
Last Modified By	LastModifiedById	Lookup(User)			
Loan	Loan__c	Lookup(Loan)			
Owner	OwnerId	Lookup(User, Group)			
Recovery Date	Recovery_Date__c	Date			
Recovery Number	Name	Auto Number			
Status	Status__c	Picklist			

Record Types

- Loan: Retail Loan & Business Loan.
- Configure Status picklist values per record type (e.g., Retail = Active/Closed, Business = Active/Defaulted).

The screenshot shows the Salesforce Object Manager interface for the 'Loan' object. On the left, a sidebar lists various configuration options: Details, Fields & Relationships, Page Layouts, Lightning Record Pages, Buttons, Links, and Actions, Compact Layouts, Field Sets, Object Limits, Record Types (which is selected and highlighted in blue), Related Lookup Filters, Search Layouts, List View Button Layout, and Restriction Rules. The main content area is titled 'Record Types' and displays a table with two items: 'Business Loan' and 'Retail Loan'. The table has columns for 'RECORD TYPE LABEL', 'DESCRIPTION', 'ACTIVE', and 'MODIFIED BY'. Both records were modified by 'Bharath Jonnadula' on 20/09/2025, 5:10 pm.

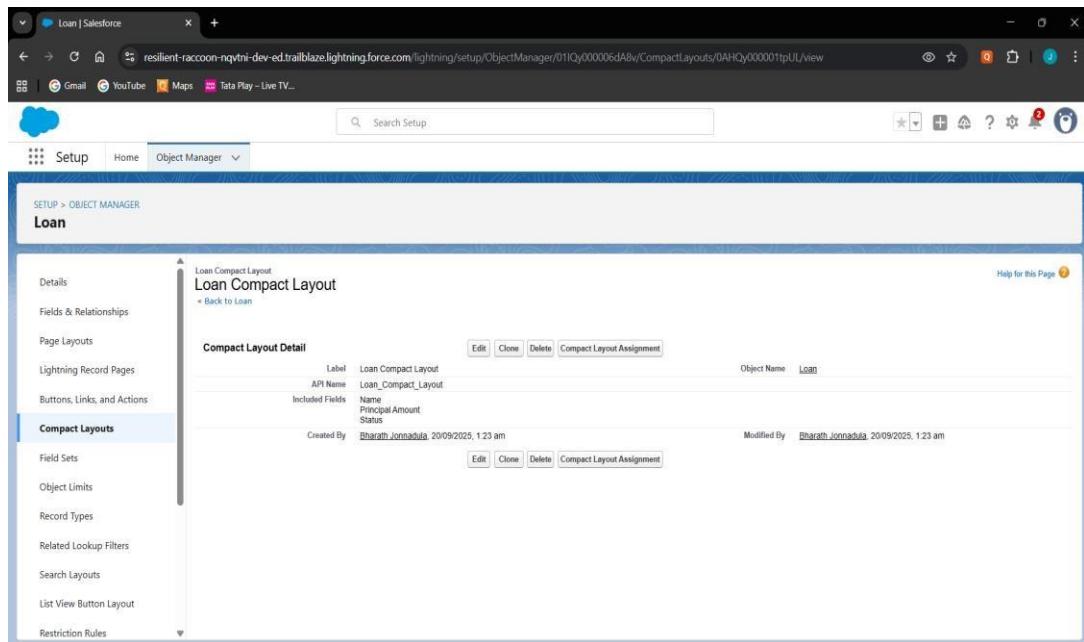
Page Layouts

- Loan Layout: sections for Basic Info, Financials, Dates & Status. Related Lists = Payments, Recoveries, Loan Agent Assignments.
- Payment Layout: fields for Date, Amount, Method, Status.
- Recovery Layout: fields for Date, Amount Recovered, Status, Assigned Agent.

The screenshot shows the Salesforce Object Manager interface for editing the 'Loan Layout'. The left sidebar shows the same list of configuration options as the previous screenshot. The main content area is titled 'Loan Layout' and includes tabs for 'Fields', 'Buttons', 'Quick Actions', 'Mobile & Lightning', 'Actions', 'Expanded Lookups', 'Related Lists', and 'Details'. A 'Layout Properties' section at the top right contains fields for 'Name' (set to 'Customer'), 'Save Form', 'Status', 'Record Type' (set to 'Business'), and 'Record Label'. Below this is a 'Loan Sample' panel with a 'Highlights Panel' and a 'Quick Actions In the Salesforce Classic Publisher' section. At the bottom, there is a note about 'Salesforce Mobile and Lightning Experience Actions'.

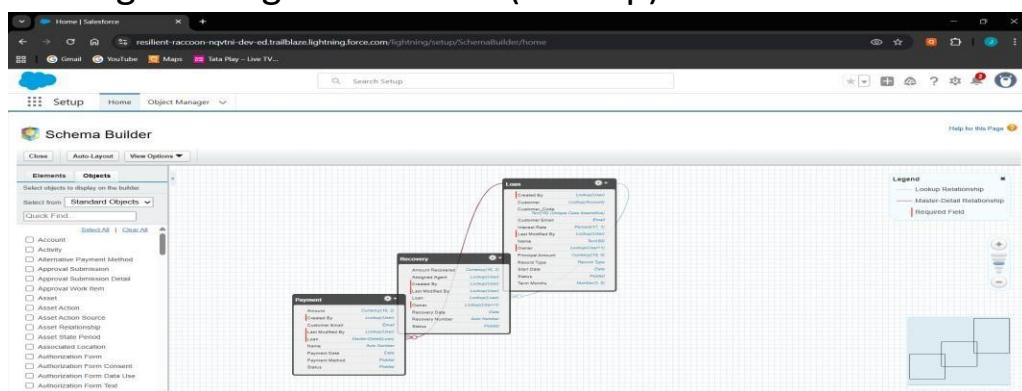
Compact Layouts

- Loan Compact Layout: Loan Number, Principal Amount, Status, Customer.
- Payment Compact Layout: Payment Date, Amount, Status.
- Recovery Compact Layout: Recovery Date, Amount Recovered, Status.



Schema Builder

- Visualize Loan, Payment, Recovery, and Loan Agent Assignment objects.
- Confirm relationships: Loan–Payment (Master-Detail), Loan–Recovery (Lookup), Loan–LoanAgentAssignment (Master-Detail), LoanAgentAssignment–User (Lookup).



Lookup vs Master-Detail vs Hierarchical

- Payment → Loan = Master-Detail (payments always tied to loans).
- Recovery → Loan = Lookup (independent audit records).
- Loan Agent Assignment → Loan = Master-Detail, → User = Lookup.
- Hierarchical not used (only for User object).

Junction Objects

- Loan Agent Assignment to handle many-to-many between Loans and Agents.
- Enables multiple agents handling multiple loans.

External Objects

- Concept: Connect to external payment system using Salesforce Connect.
- Example: Show legacy payment history without storing inside Salesforce.

Phase 4: Process Automation (Admin)

Validation Rules

- Ensure values are valid (Loan/Payment/Recovery > 0).
- Prevent negative amounts and invalid dates.

The image displays two screenshots of the Salesforce Object Manager interface, specifically focusing on the Validation Rules section for the Loan and Payment objects.

Loan Validation Rules:

RULE NAME	ERROR LOCATION	ERROR MESSAGE	ACTIVE	MODIFIED BY
Interest_Rate_Range	Interest Rate	Interest Rate must be between 0 and 100.	✓	Bharath Jonnadula, 20/09/2025, 11:44 am
Principal_Positive	Principal Amount	Principal Amount must be greater than zero.	✓	Bharath Jonnadula, 20/09/2025, 11:44 am
Term_Positive	Term Months	Loan term (Term Months) must be greater than zero.	✓	Bharath Jonnadula, 20/09/2025, 11:43 am

Payment Validation Rules:

RULE NAME	ERROR LOCATION	ERROR MESSAGE	ACTIVE	MODIFIED BY
Payment_Amount_Positive	Amount	Payment Amount must be greater than zero.	✓	Bharath Jonnadula, 20/09/2025, 11:59 am
Payment_Date_must_be_on	Payment Date	Payment Date cannot be earlier than the Loan Start Date.	✓	Bharath Jonnadula, 20/09/2025, 11:56 am
Payment_Status_required	Status	Please select a Payment Status.	✓	Bharath Jonnadula, 20/09/2025, 12:00 pm

Rule Name	Error Location	Error Message	Active	Modified By
AssignedAgent_Required	Assigned Agent	Assigned Agent must be selected when status is Assigned.	✓	Bharath Jonnadula, 20/09/2025, 12:04 pm
AssignedAgent_Required_ForAssigned	Top of Page	Please select an Assigned Agent when status is Assigned.	✓	Bharath Jonnadula, 20/09/2025, 12:13 pm
Recovery_Amount_Positive	Amount Recovered	Recovery Amount must be greater than zero.	✓	Bharath Jonnadula, 20/09/2025, 12:03 pm
RecoveryDate_NotFuture	Recovery Date	Recovery Date cannot be in the future.	✓	Bharath Jonnadula, 20/09/2025, 12:04 pm

Workflow Rules

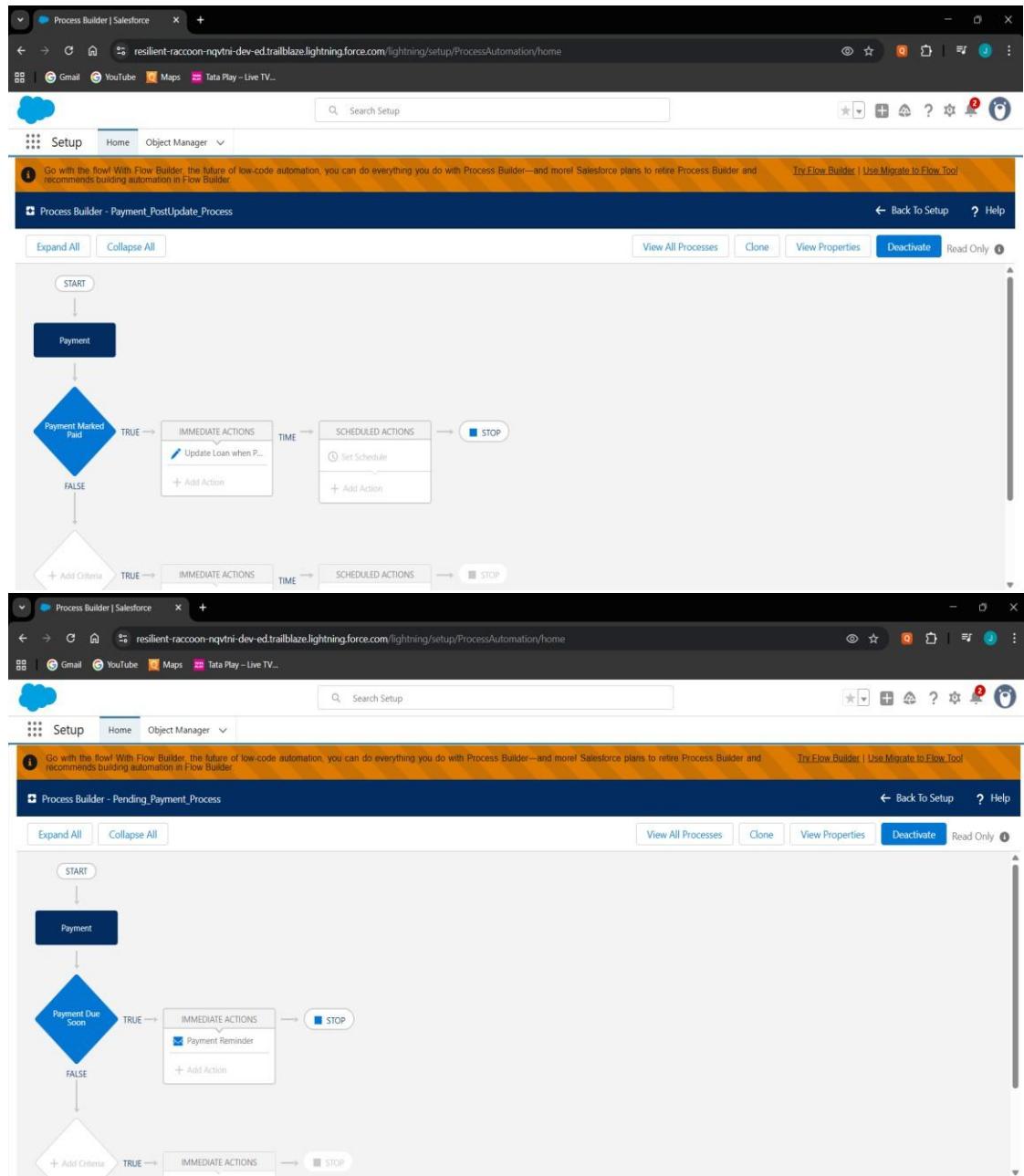
- Rule: Pending_Payment_Reminder.
- Sends reminder email for pending payments.

Action	Rule Name	Description	Object	Active
Edit Del Deactivate	Payment_Overage_Rule	created, and any time it's edited to subsequently meet criteria	Payment	✓
Edit Del Deactivate	Pending_Payment_Reminder		Payment	✓

Process Builder

- Sends email reminders for due payments.
- Updates Loan status when all Payments are paid.

- Creates follow-up task for overdue payments.



Approval Process

- Loans > 5 Lakhs require Manager approval.
- Auto email + status update on approval/rejection.

Flow Builder

- Scheduled Flow: Sends payment reminders 2 days before due.
- Record-Triggered: Marks payments overdue automatically.

- Record-Triggered: Creates Recovery record when overdue.

Email Alerts

- Payment_Reminder_Email → Customer.
- Loan_Submitted, Approved, Rejected → Manager/Officer.

The screenshot shows the Salesforce Setup interface for 'Email Alerts'. The left sidebar includes 'Process Automation' and 'Workflow Actions' sections, with 'Email Alerts' selected. The main content area displays a table titled 'All Email Alerts' with the following data:

Action	Description	Email Template Name	Object	Last Modified Date
Edit Del	Loan_Approved_Email	Loan_Approved_Template	Loan	20/09/2025
Edit Del	Loan_Rejected_Email	Loan_Rejected_Template	Loan	20/09/2025
Edit Del	Loan_Submitted_Email	Loan_Submitted_Template	Loan	20/09/2025
Edit Del	Payment_Reminder_Email	Payment_Reminder_Template	Payment	20/09/2025
Edit Del	Send_Payment_Reminder_Email	Payment_Reminder_Template	Payment	20/09/2025

Field Updates

- Auto-change Payment.Status to Overdue.
- Auto-close Loan when all Payments are Paid.

The screenshot shows the Salesforce Setup interface for 'Field Updates'. The left sidebar includes 'Process Automation' and 'Workflow Actions' sections, with 'Field Updates' selected. The main content area displays a table titled 'All Workflow Field Updates' with the following data:

Action	Name	Field to Update	Operation	Value	Last Modified Date
Edit Del	Changes the case priority to high.	Case: Priority	Value	High	15/09/2025
Edit Del	Mark_Payment_Overdue	Payment: Status	Value	Overdue	20/09/2025
Edit Del	Mark_Payment_Paid	Payment: Status	Value	Paid	20/09/2025
Edit Del	Set_Loan_Status_Approved	Loan: Status	Value	Approved	20/09/2025
Edit Del	Set_Loan_Status_Closed	Loan: Status	Value	Closed	20/09/2025
Edit Del	Set_Loan_Status_Rejected	Loan: Status	Value	Rejected	20/09/2025
Edit Del	Set_Recovery_Status_Completed	Recovery: Status	Value	Completed	20/09/2025

Tasks

- Task created when Payment is Overdue.
- Assigned to Recovery Agent with high priority.

Custom Notifications

- Sends in-app notification for Overdue Payments.

Phase 5: Apex Programming (Developer)

Classes & Objects

- Created Apex helper classes (LoanHandler, PaymentHandler, RecoveryHandler) to separate business logic from triggers.
- Used object-oriented approach for cleaner, reusable, and modular code.

LoanHandler.cls

```
public class LoanHandler {

    public static void beforeInsert(List<Loan__c>
newLoans) {
        for (Loan__c loan : newLoans) {
            if (loan.Status__c == null) {
                loan.Status__c = 'Pending Approval';
            }
        }
    }

    public static void beforeUpdate(List<Loan__c>
newLoans, Map<Id, Loan__c> oldMap) {
        for (Loan__c loan : newLoans) {
            Loan__c oldLoan = oldMap.get(loan.Id);
            if (loan.Principal_Amount__c != oldLoan.Principal_Amount__c) {
                loan.Status__c = 'Modified';
            }
        }
    }

    public static void afterInsert(List<Loan__c> newLoans)
    {
        System.debug('New Loans inserted: ' + newLoans);
    }
}
```

```
public static void afterUpdate(List<Loan__c>
newLoans, Map<Id, Loan__c> oldMap) {
    System.debug('Loans updated: ' + newLoans);
}
```

```
public static void afterDelete(List<Loan__c> oldLoans)
{
    System.debug('Loans deleted: ' + oldLoans);
}
```

The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows the project structure under "PAYMENTTRACKINGINBANKING".
- Code Editor:** Displays the Apex class `LoanHandler.cls` with its static methods `beforeInsert` and `beforeUpdate`.
- Terminal:** Shows the command line output of a deployment command.
- Status Bar:** Shows file paths, line numbers, and other development information.

```
1  public class LoanHandler {
2
3      public static void beforeInsert(List<Loan__c> newLoans) {
4          for (Loan__c loan : newLoans) {
5              if (loan.Status__c == null) {
6                  loan.Status__c = 'Pending Approval';
7              }
8          }
9      }
10
11     public static void beforeUpdate(List<Loan__c> newLoans, Map<Id, Loan__c> oldMap) {
12         for (Loan__c loan : newLoans) {
13             Loan__c oldLoan = oldMap.get(loan.Id);
14             if (loan.Principal_Amount__c != oldLoan.Principal_Amount__c) {
15                 loan.Status__c = 'Modified';
16             }
17         }
18     }
19 }
```

```
Microsoft Windows [Version 10.0.26100.6584]
(c) Microsoft Corporation. All rights reserved.

D:\PaymentTrackinginBanking>sf project deploy start --source-dir force-app/main/default/classes --target-org DevOrg
* History restored

Microsoft Windows [Version 10.0.26100.6584]
(c) Microsoft Corporation. All rights reserved.
```

PaymentHandler.cls

```
public class PaymentHandler {  
  
    public static void beforeInsert(List<Payment__c> newPayments) {  
        for (Payment__c p : newPayments) {  
            if (p.Status__c == null) p.Status__c = 'Pending';  
        }  
    }  
  
    public static void beforeUpdate(List<Payment__c> newPayments,  
        Map<Id, Payment__c> oldMap) {  
        for (Payment__c p : newPayments) {  
            Payment__c oldP = oldMap.get(p.Id);  
            if (p.Amount__c != oldP.Amount__c) {  
                p.Status__c = 'Revised';  
            }  
        }  
    }  
  
    public static void afterInsert(List<Payment__c> newPayments) {  
        updateLoanStatus(newPayments);  
    }  
  
    public static void afterUpdate(List<Payment__c> newPayments,  
        Map<Id, Payment__c> oldMap) {  
        updateLoanStatus(newPayments);  
    }  
  
    public static void afterDelete(List<Payment__c> oldPayments) {  
        updateLoanStatus(oldPayments);  
    }  
  
    private static void updateLoanStatus(List<Payment__c> payments) {  
        Set<Id> loanIds = new Set<Id>();  
        for (Payment__c p : payments) if (p.Loan__c != null)  
            loanIds.add(p.Loan__c);  
        if (loanIds.isEmpty()) return;  
    }  
}
```

```

Map<Id, Integer> totalPayments = new Map<Id, Integer>();
for (AggregateResult ar : [
    SELECT Loan__c loanId, COUNT(Id) totalCount
    FROM Payment__c WHERE Loan__c IN :loanIds
    GROUP BY Loan__c
]) {
    totalPayments.put((Id) ar.get('loanId'), (Integer) ar.get('totalCount'));
}

Map<Id, Integer> paidPayments = new Map<Id, Integer>();
for (AggregateResult ar : [
    SELECT Loan__c loanId, COUNT(Id) paidCount
    FROM Payment__c WHERE Loan__c IN :loanIds AND Status__c
    = 'Paid'
    GROUP BY Loan__c
]) {
    paidPayments.put((Id) ar.get('loanId'), (Integer) ar.get('paidCount'));
}

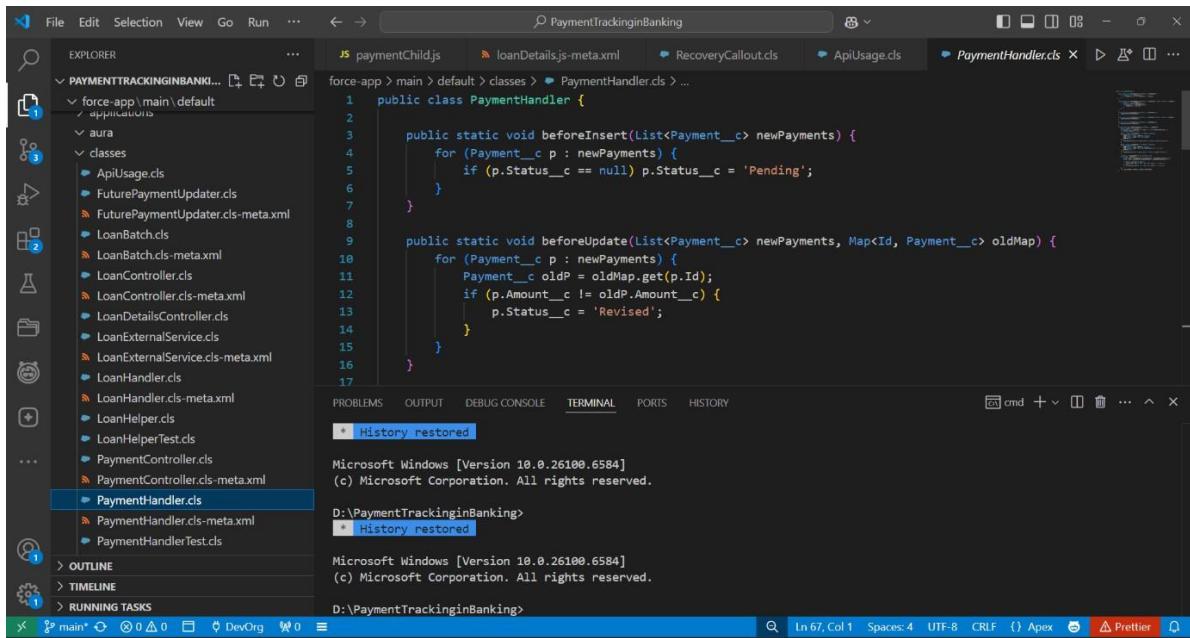
List<Loan__c> loansToUpdate = new List<Loan__c>();
for (Id lid : loanIds) {
    Integer total = totalPayments.containsKey(lid) ?
totalPayments.get(lid) : 0;
    Integer paid = paidPayments.containsKey(lid) ?
paidPayments.get(lid) : 0;

    if (total > 0 && total == paid) {
        loansToUpdate.add(new Loan__c(Id = lid, Status__c =
'Closed'));
    } else {
        loansToUpdate.add(new Loan__c(Id = lid, Status__c =
'Active'));
    }
}
if (!loansToUpdate.isEmpty()) update loansToUpdate;

```

```
}
```

```
}
```



```
PaymentHandler.cls
```

```
1 public class PaymentHandler {  
2     public static void beforeInsert(List<Payment__c> newPayments) {  
3         for (Payment__c p : newPayments) {  
4             if (p.Status__c == null) p.Status__c = 'Pending';  
5         }  
6     }  
7     public static void beforeUpdate(List<Payment__c> newPayments, Map<Id, Payment__c> oldMap) {  
8         for (Payment__c p : newPayments) {  
9             Payment__c oldP = oldMap.get(p.Id);  
10            if (p.Amount__c != oldP.Amount__c) {  
11                p.Status__c = 'Revised';  
12            }  
13        }  
14    }  
15 }  
16 }  
17 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS HISTORY

```
* History restored  
Microsoft Windows [Version 10.0.26100.6584]  
(c) Microsoft Corporation. All rights reserved.  
D:\PaymentTrackinginBanking  
* History restored  
Microsoft Windows [Version 10.0.26100.6584]  
(c) Microsoft Corporation. All rights reserved.  
D:\PaymentTrackinginBanking
```

Ln 67, Col 1 Spaces: 4 UTF-8 CRLF {} Apex ⚡ Prettier

RecoveryHandler.cls

```
public class RecoveryHandler {  
  
    public static void beforeInsert(List<Recovery__c> newRecoveries) {  
        for (Recovery__c r : newRecoveries) {  
            if (r.Status__c == null) r.Status__c = 'Assigned';  
        }  
    }  
  
    public static void beforeUpdate(List<Recovery__c> newRecoveries,  
        Map<Id, Recovery__c> oldMap) {  
        for (Recovery__c r : newRecoveries) {  
            Recovery__c oldR = oldMap.get(r.Id);  
            if (r.Amount_Recovered__c != oldR.Amount_Recovered__c) {  
                r.Status__c = 'Updated';  
            }  
        }  
    }  
}
```

```

public static void afterInsert(List<Recovery__c> newRecoveries) {
    System.debug('New Recoveries: ' + newRecoveries);
}

public static void afterUpdate(List<Recovery__c> newRecoveries,
Map<Id, Recovery__c> oldMap) {
    System.debug('Updated Recoveries: ' + newRecoveries);
}

public static void afterDelete(List<Recovery__c> oldRecoveries) {
    System.debug('Deleted Recoveries: ' + oldRecoveries);
}

```

The screenshot shows the Visual Studio Code (VS Code) interface with the following details:

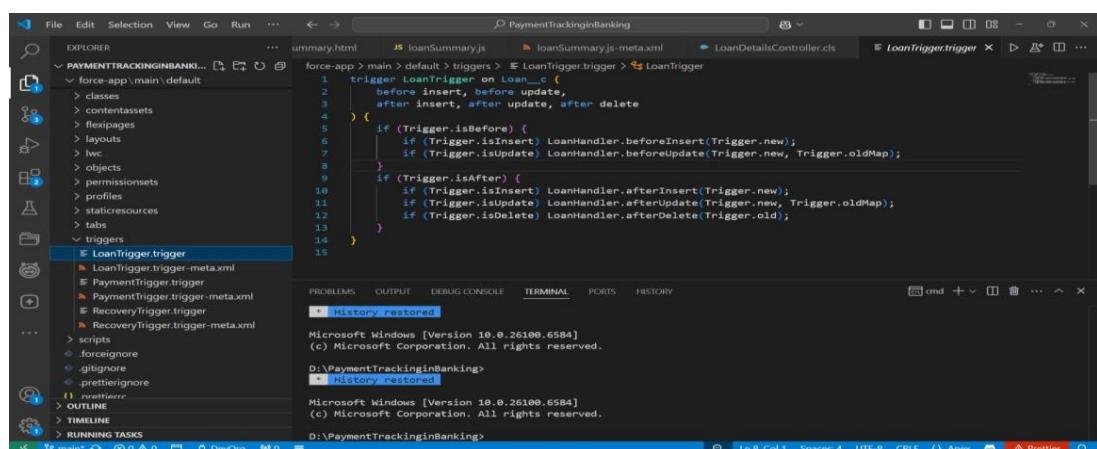
- File Explorer:** On the left, it shows the project structure under "PAYOUTTRACKINGINBANKING". The "classes" folder contains several Apex classes: PaymentController.cls, PaymentHandler.cls, PaymentHandlerTest.cls, PaymentHelper.cls, PaymentReminderQueue.cls, PaymentReminderQueue.cls-meta.xml, PaymentSearchDemo.cls, PaymentSearchDemo.cls-meta.xml, PaymentSummaryController.cls, PaymentSummaryController.cls-meta.xml, RecoveryCallout.cls, RecoveryHandler.cls, RecoveryHandler.cls-meta.xml, RecoveryHandlerTest.cls, RecoveryHelper.cls, Test_PaymentHandlers.cls, and Test_RecoveryHandler.cls.
- Code Editor:** The main area displays the Apex code for the RecoveryHandler class. It includes three static methods: beforeInsert, beforeUpdate, and afterDelete. The beforeInsert method sets the Status__c field to 'Assigned' if it is null. The beforeUpdate method checks if the Amount_Recovered__c field has changed and updates the Status__c field to 'Updated' if it has. The afterDelete method prints a debug message.
- Terminal:** At the bottom, the terminal window shows the command `sf project deploy start --source-dir force-app/main/default/classes` being run in the directory `D:\PaymentTrackinginBanking`. The output indicates that history was restored.
- Bottom Status Bar:** The status bar at the bottom shows the current file is "main.sfdx", there are 0 errors and 0 warnings, the target org is "DevOrg", and the file has 0 changes.

Apex Triggers

- Implemented triggers for Loan, Payment, and Recovery to handle automation like updating statuses and roll-ups.

LoanTrigger.trigger

```
trigger LoanTrigger on Loan__c (  
    before insert, before update,  
    after insert, after update, after delete  
) {  
  
    if (Trigger.isBefore) {  
  
        if (Trigger.isInsert) LoanHandler.beforeInsert(Trigger.new);  
  
        if (Trigger.isUpdate) LoanHandler.beforeUpdate(Trigger.new,  
            Trigger.oldMap);  
  
    }  
  
    if (Trigger.isAfter) {  
  
        if (Trigger.isInsert) LoanHandler.afterInsert(Trigger.new);  
  
        if (Trigger.isUpdate) LoanHandler.afterUpdate(Trigger.new,  
            Trigger.oldMap);  
  
        if (Trigger.isDelete) LoanHandler.afterDelete(Trigger.old);}}}
```

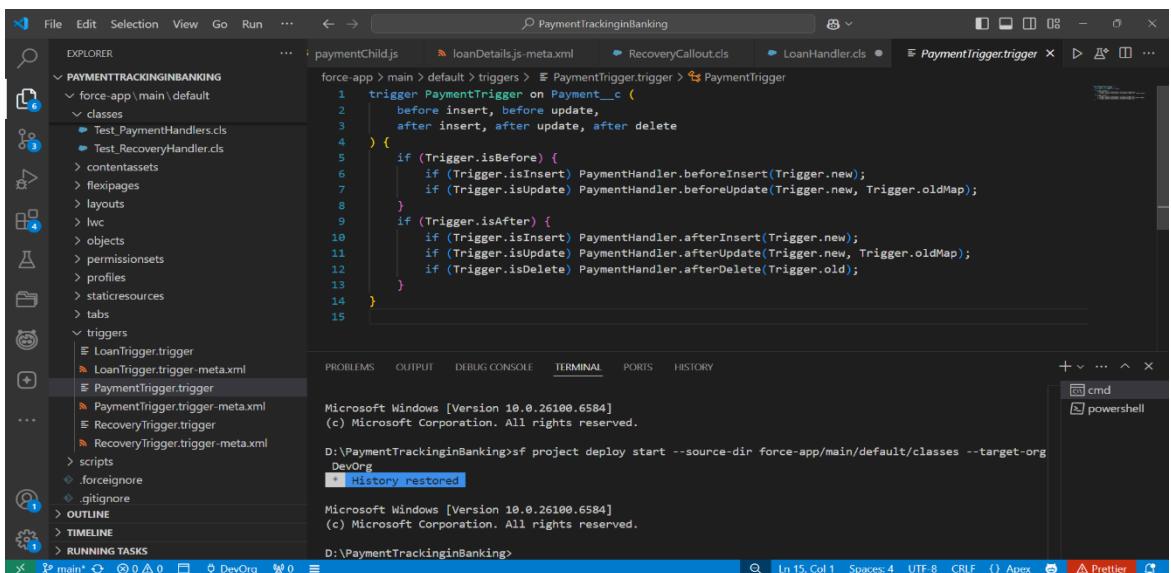


The screenshot shows the Visual Studio Code (VS Code) interface with the following details:

- Explorer View:** Shows the project structure under "PAYMENTTRACKINGINBANKING/main/default". The "triggers" folder is expanded, and "LoanTrigger.trigger" is selected, indicated by a blue outline.
- Editor View:** The main editor area displays the Apex code for "LoanTrigger.trigger". The code handles four trigger types: before insert, before update, after insert, and after update. It uses the "LoanHandler" class to perform specific actions based on the trigger type and record status (isBefore or isAfter).
- Terminal View:** The bottom right terminal window shows the output of running the code in a Windows environment, indicating "History restored" twice.
- Status Bar:** The bottom status bar shows the file name "main", line "Ln 8, Col 1", spaces "Spaces: 4", encoding "UTF-8", and file type "Apex".

PaymentTrigger.trigger

```
trigger PaymentTrigger on Payment__c (
    before insert, before update,
    after insert, after update, after delete
) {
    if (Trigger.isBefore) {
        if (Trigger.isInsert) PaymentHandler.beforeInsert(Trigger.new);
        if (Trigger.isUpdate)
            PaymentHandler.beforeUpdate(Trigger.new, Trigger.oldMap);
    }
    if (Trigger.isAfter) {
        if (Trigger.isInsert) PaymentHandler.afterInsert(Trigger.new);
        if (Trigger.isUpdate)
            PaymentHandler.afterUpdate(Trigger.new, Trigger.oldMap);
        if (Trigger.isDelete) PaymentHandler.afterDelete(Trigger.old);
    }
}
```



The screenshot shows the Salesforce IDE interface. The top navigation bar includes File, Edit, Selection, View, Go, Run, and other standard options. The main area is titled "PaymentTrackinginBanking". The left sidebar, "EXPLORER", displays the project structure under "PAYMENTTRACKINGINBANKING". It lists "force-app > main > default > triggers > PaymentTrigger.trigger" and its associated files: "PaymentTrigger.js", "PaymentTrigger.trigger-meta.xml", and "PaymentTrigger.trigger-meta.xml". Other items in the sidebar include "Test_PaymentHandlers.cls", "Test_RecoveryHandler.cls", "contentassets", "flexipages", "lwc", "objects", "permissionsets", "profiles", "staticresources", "tabs", and "triggers". The bottom status bar shows file paths like "D:\PaymentTrackinginBanking\sf", deployment status, and terminal output. The bottom right corner has icons for cmd and powershell.

RecoveryTrigger.trigger

```
trigger RecoveryTrigger on Recovery__c (
    before insert, before update,
    after insert, after update, after delete
) {

    if (Trigger.isBefore) {

        if (Trigger.isInsert) RecoveryHandler.beforeInsert(Trigger.new);

        if (Trigger.isUpdate)
            RecoveryHandler.beforeUpdate(Trigger.new, Trigger.oldMap);

    }

    if (Trigger.isAfter) {

        if (Trigger.isInsert) RecoveryHandler.afterInsert(Trigger.new);

        if (Trigger.isUpdate)
            RecoveryHandler.afterUpdate(Trigger.new, Trigger.oldMap);

        if (Trigger.isDelete) RecoveryHandler.afterDelete(Trigger.old);

    }

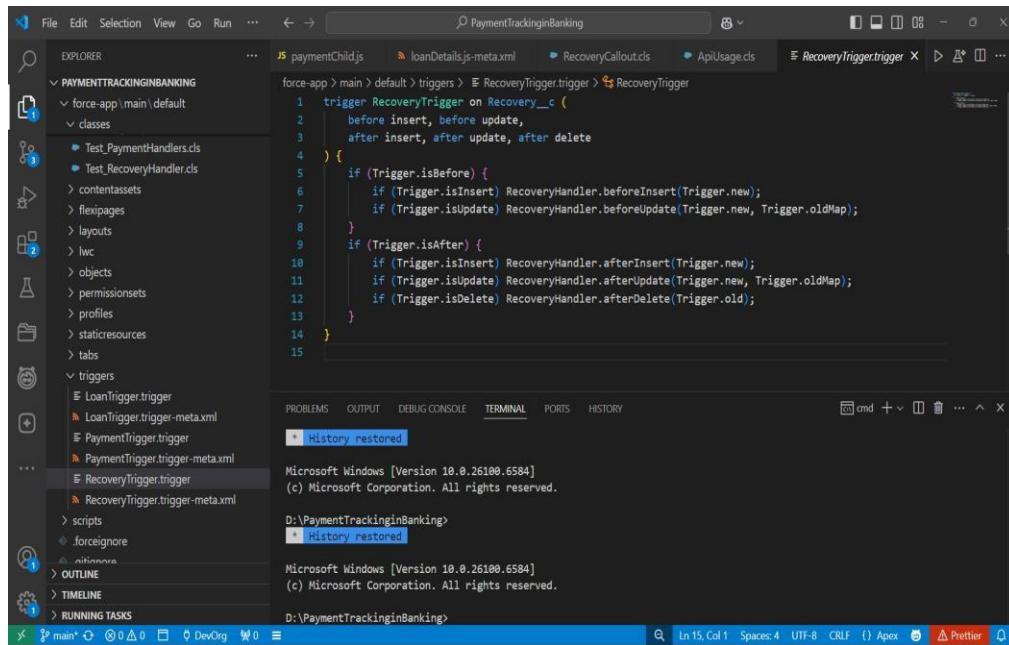
}
```

The screenshot shows the Salesforce IDE interface with the following details:

- Editor Tab:** Displays the Apex trigger code for `RecoveryTrigger`. The code handles both `before` and `after` triggers on the `Recovery__c` object. It includes logic for insert, update, and delete operations, calling methods in the `RecoveryHandler` class.
- Terminal Tab:** Shows the command-line output of a deployment process. It includes the command `sf project deploy start --source-dir force-app/main/default/classes --target-org DevOrg`, the message "History restored.", and two Microsoft Windows version messages.

Trigger Design Pattern

- Applied Trigger Handler Pattern to keep trigger code clean and delegate logic to handler classes.
- Ensured only one trigger per object with proper event handling.



The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure for "PAYMENTTRACKINGINBANKING".
- Editor:** Displays the code for "RecoveryTrigger.trigger".
- Terminal:** Shows two terminal windows:
 - Top window: Microsoft Windows [Version 10.0.26100.6584]
D:\PaymentTrackinginBanking> * History restored*
 - Bottom window: Microsoft Windows [Version 10.0.26100.6584]
D:\PaymentTrackinginBanking> * History restored*
- Status Bar:** Shows the current file is "main* 0 0 0 0 DevOrg 0 0", line 15, column 1, spaces: 4, UTF-8, CRLF, and the file type is Apex.

SOQL & SOSL

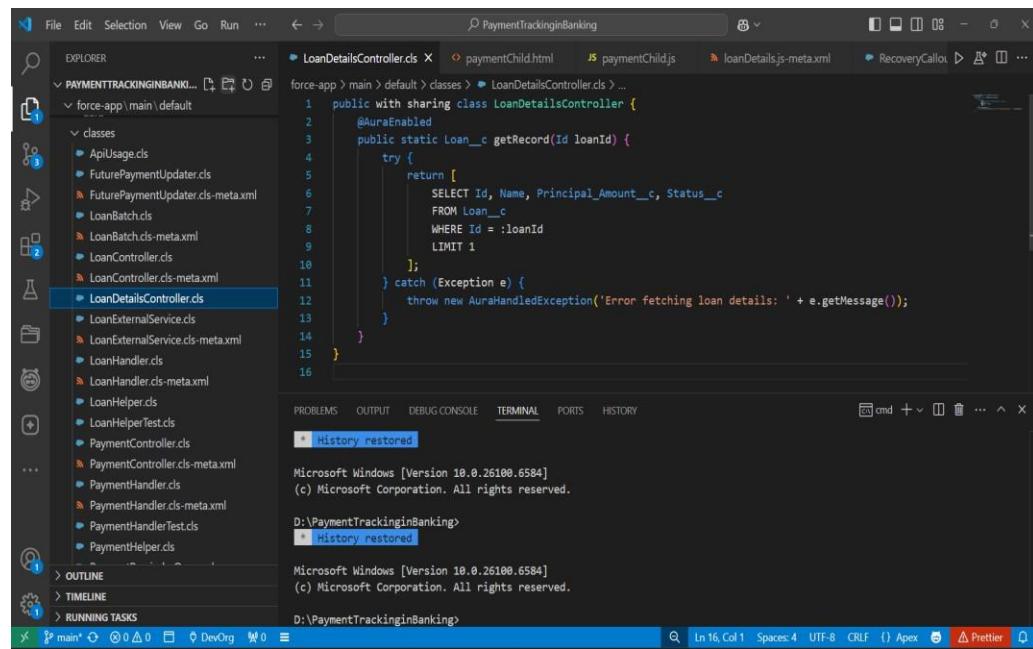
- Used SOQL in classes to query Loans, Payments, and Recoveries.
- Implemented simple search functionality to fetch customer records.

LoanDetailsController.cls

```
public with sharing class LoanController {  
    @AuraEnabled(cacheable=true)  
    public static Loan__c getLoan(String loanId) {  
        return [SELECT Name, Principal_Amount__c FROM  
        Loan__c WHERE Id = :loanId LIMIT 1];  
    }  
}
```

```
@AuraEnabled
```

```
public static Boolean markLoanClosed(String loanId) {  
    try {  
        Loan__c L = [SELECT Id, Status__c FROM Loan__c WHERE  
        Id = :loanId LIMIT 1];  
        L.Status__c = 'Closed';  
        update L;  
        return true;  
    } catch (Exception e) {  
        throw new AuraHandledException('Cannot close loan: ' +  
        e.getMessage());  
    }  
}
```



The screenshot shows the Visual Studio Code (VS Code) interface with the following details:

- File Path:** PAYMENTTRACKINGINBANKING\force-app\main\default\classes\LoanDetailsController.cls
- Editor Content:** The code for the `LoanDetailsController` class, which includes a static method `markLoanClosed`.
- Explorer Sidebar:** Shows the project structure with files like `ApiUsage.cls`, `FuturePaymentUpdater.cls`, `LoanBatch.cls`, `LoanBatch.cls-meta.xml`, `LoanController.cls`, `LoanController.cls-meta.xml`, `LoanExternalService.cls`, `LoanExternalService.cls-meta.xml`, `LoanHandler.cls`, `LoanHandler.cls-meta.xml`, `LoanHelper.cls`, `LoanHelperTest.cls`, `PaymentController.cls`, `PaymentController.cls-meta.xml`, `PaymentHandler.cls`, `PaymentHandler.cls-meta.xml`, `PaymentHandlerTest.cls`, and `PaymentHelper.cls`.
- Terminal:** Displays the Windows command prompt output, showing history restoration and copyright information.
- Status Bar:** Shows the current file is `main*`, has 0 errors and 0 warnings, and is connected to `DevOrg`.

PaymentsController.cls

```
public class PaymentController {  
    @AuraEnabled(cacheable=true)  
    public static List<Payment__c> getPayments(Id loanId) {  
        if (loanId == null) return new List<Payment__c>();  
        return [  
            SELECT Name, Amount__c, Status__c,  
            Payment_Date__c  
            FROM Payment__c  
            WHERE Loan__c = :loanId  
            ORDER BY Payment_Date__c DESC  
        ];  
    }  
}
```

The screenshot shows the Visual Studio Code (VS Code) interface with the following details:

- Editor:** The main editor area displays the `PaymentController.cls` Apex class code.
- Explorer:** The left sidebar shows the project structure under `PAYMENTTRACKINGINBANKING`, specifically the `force-app/main/default/classes` folder, where `PaymentController.cls` is listed.
- Terminal:** The bottom right terminal window shows the deployment command and its output:

```
Microsoft Windows [Version 10.0.26100.6584]  
(c) Microsoft Corporation. All rights reserved.  
D:\PaymentTrackinginBanking>sf project deploy start --source-dir force-app/main/default/classes --target-org DevOrg  
+ History restored
```

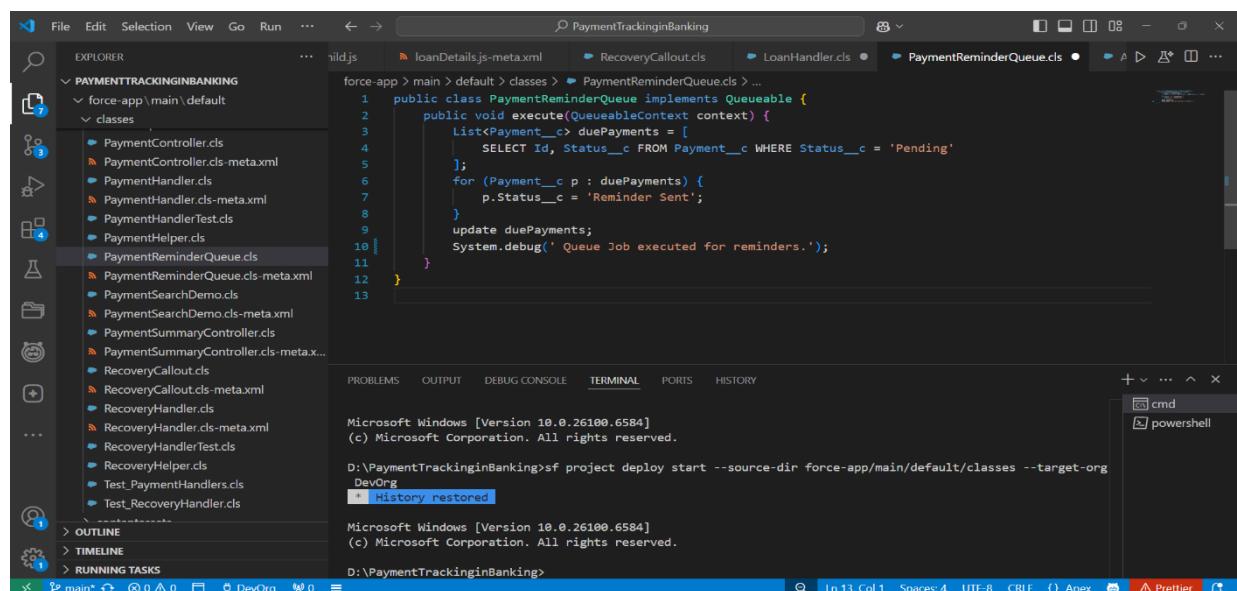
```
Microsoft Windows [Version 10.0.26100.6584]  
(c) Microsoft Corporation. All rights reserved.  
D:\PaymentTrackinginBanking>
```

Collections: List, Set, Map

- Applied Lists to hold multiple Payments for bulk processing.
- Used Maps for Loan → Payments mapping to handle roll-ups efficiently.

PaymentReminderQueue.cls

```
public class PaymentReminderQueue implements Queueable {
    public void execute(QueueableContext context) {
        List<Payment__c> duePayments = [
            SELECT Id, Status__c FROM Payment__c WHERE
            Status__c = 'Pending'
        ];
        for (Payment__c p : duePayments) {
            p.Status__c = 'Reminder Sent';
        }
        update duePayments;
        System.debug(' Queue Job executed for reminders.');
    }
}
```



```
1  public class PaymentReminderQueue implements Queueable {
2      public void execute(QueueableContext context) {
3          List<Payment__c> duePayments = [
4              SELECT Id, Status__c FROM Payment__c WHERE
5              Status__c = 'Pending'
6          ];
7          for (Payment__c p : duePayments) {
8              p.Status__c = 'Reminder Sent';
9          }
10         update duePayments;
11         System.debug(' Queue Job executed for reminders.');
12     }
13 }
```

Microsoft Windows [Version 10.0.26100.6584]
© Microsoft Corporation. All rights reserved.
D:\PaymentTrackinginBanking>sf project deploy start --source-dir force-app/main/default/classes --target-org DevOrg
* History restored

Microsoft Windows [Version 10.0.26100.6584]
© Microsoft Corporation. All rights reserved.
D:\PaymentTrackinginBanking>

Control Statements

- Implemented IF conditions for validation.

RecoveryHandler.cls

```
public class RecoveryHandler {  
  
    public static void beforeInsert(List<Recovery__c> newRecoveries) {  
  
        for (Recovery__c r : newRecoveries) {  
  
            if (r.Status__c == null) r.Status__c = 'Assigned';  
  
        }  
  
    }  
  
  
    public static void beforeUpdate(List<Recovery__c> newRecoveries,  
        Map<Id, Recovery__c> oldMap) {  
  
        for (Recovery__c r : newRecoveries) {  
  
            Recovery__c oldR = oldMap.get(r.Id);  
  
            if (r.Amount_Recovered__c != oldR.Amount_Recovered__c) {  
  
                r.Status__c = 'Updated';  
  
            }  
  
        }  
  
    }  
  
  
    public static void afterInsert(List<Recovery__c> newRecoveries) {  
  
        System.debug('New Recoveries: ' + newRecoveries);  
  
    }  
}
```

```

public static void afterUpdate(List<Recovery__c> newRecoveries,
Map<Id, Recovery__c> oldMap) {
    System.debug('Updated Recoveries: ' + newRecoveries);
}

public static void afterDelete(List<Recovery__c> oldRecoveries) {
    System.debug('Deleted Recoveries: ' + oldRecoveries);
}

```

```

1  public class RecoveryHandler {
2
3      public static void beforeInsert(List<Recovery__c> newRecoveries) {
4          for (Recovery__c r : newRecoveries) {
5              if (r.Status__c == null) r.Status__c = 'Assigned';
6          }
7      }
8
9      public static void beforeUpdate(List<Recovery__c> newRecoveries, Map<Id, Recovery__c> oldMap) {
10         for (Recovery__c r : newRecoveries) {
11             Recovery__c oldR = oldMap.get(r.Id);
12             if (r.Amount_Recovered__c != oldR.Amount_Recovered__c) {
13                 r.Status__c = 'Updated';
14             }
15         }
16     }
17 }

```

The screenshot shows the VS Code interface with the following details:

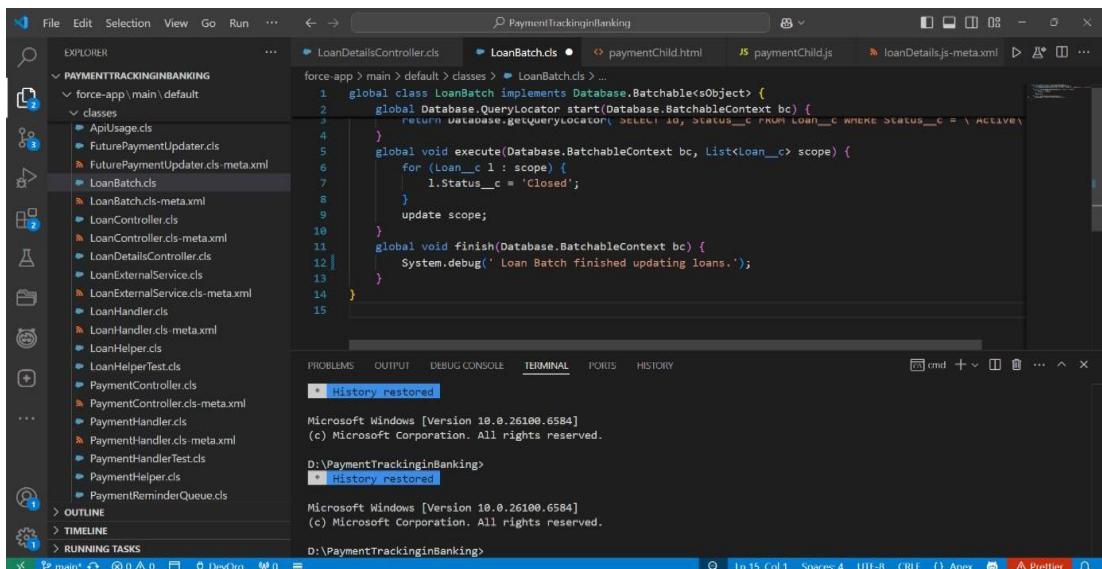
- File Explorer:** Shows the project structure under "PAYOUTTRACKINGINBANKING". The "classes" folder contains several Apex classes like PaymentController.cls, PaymentHandler.cls, etc.
- Editor:** The "RecoveryHandler.cls" file is open, displaying the provided Apex code.
- Terminal:** Two terminal windows are visible, both showing the message "* History restored".
- Status Bar:** Shows the file path "D:\PaymentTrackinginBanking", line 30, column 1, spaces 4, and encoding CRLF.

Batch Apex

- Created simple batch class for recalculating loan balances.
- Enabled processing of large datasets asynchronously.

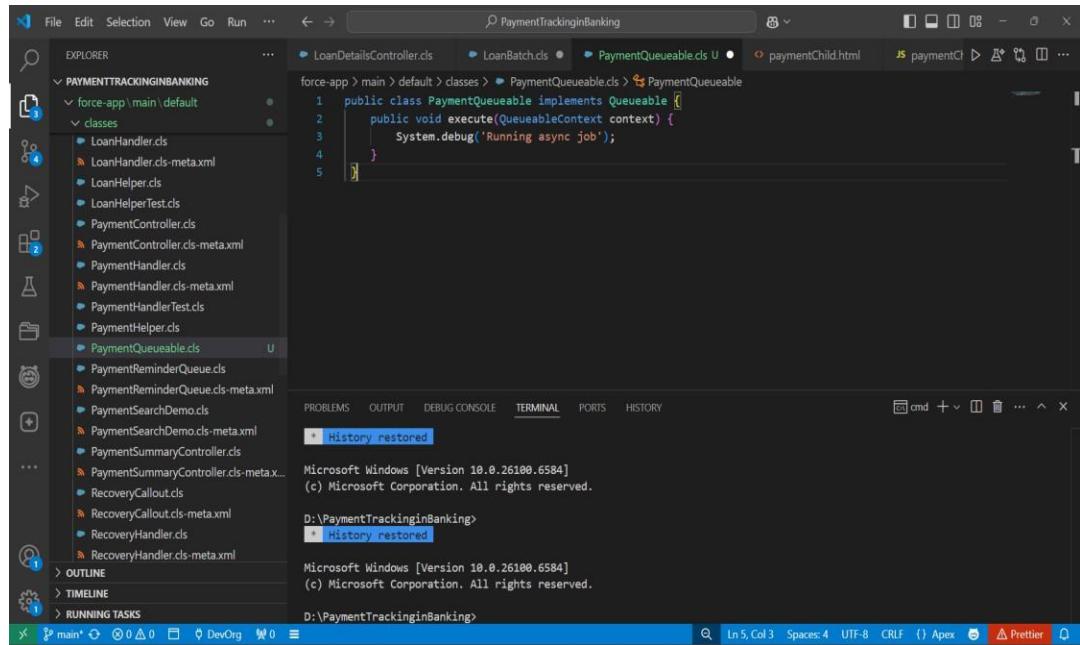
LoanBatch.cls

```
global class LoanBatch implements Database.Batchable<sObject> {
    global Database.QueryLocator start(Database.BatchableContext bc)
    {
        return Database.getQueryLocator('SELECT Id, Status__c FROM
Loan__c WHERE Status__c = \'Active\'');
    }
    global void execute(Database.BatchableContext bc, List<Loan__c>
scope) {
        for (Loan__c l : scope) {
            l.Status__c = 'Closed';
        }
        update scope;
    }
    global void finish(Database.BatchableContext bc) {
        System.debug(' Loan Batch finished updating loans.');
    }
}
```



Queueable Apex

- Used Queueable Apex for lightweight async operations like notifications.
- Provides better chaining support compared to future methods.



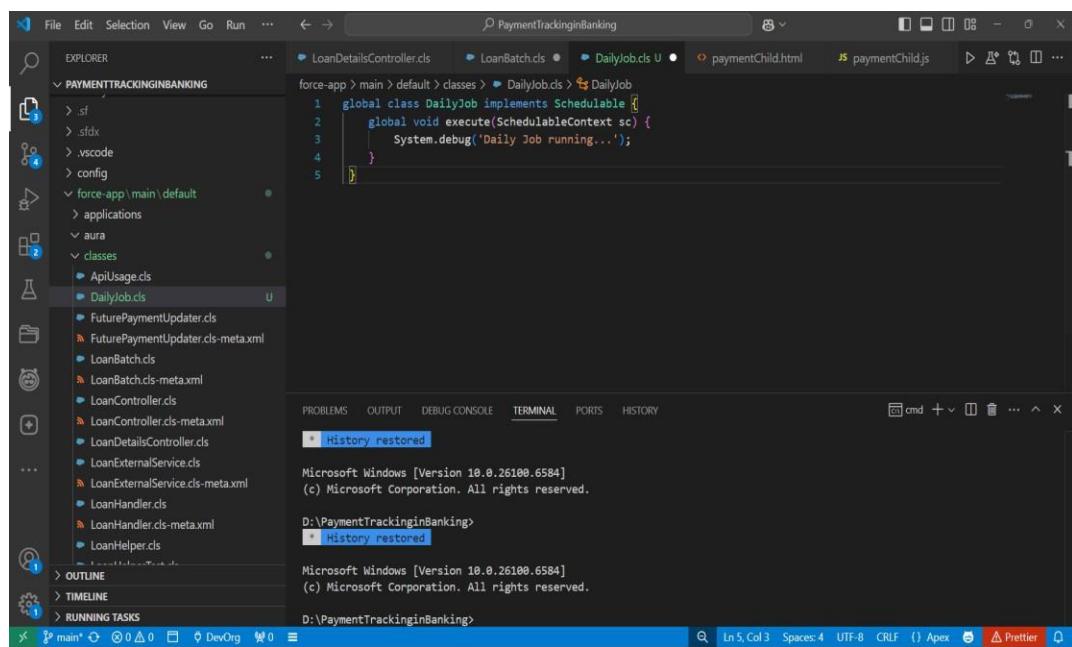
The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows the project structure under "PAYMENTTRACKINGINBANKING". The "classes" folder contains several files, and "PaymentQueueable.cls" is currently selected.
- Code Editor:** Displays the code for "PaymentQueueable.cls":

```
1 public class PaymentQueueable implements Queueable {
2     public void execute(QueueableContext context) {
3         System.debug('Running async job');
4     }
5 }
```
- Terminal:** Shows the command history restored on Microsoft Windows 10.
- Status Bar:** Shows the file path "D:\PaymentTrackinginBanking", line 5, column 3, and other development settings.

Scheduled Apex

- Built a scheduler to run daily overdue payment checks.
- Automated reminders and recovery creation without manual intervention.



The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows the project structure under "PAYMENTTRACKINGINBANKING". The "classes" folder contains several files, and "DailyJob.cls" is currently selected.
- Code Editor:** Displays the code for "DailyJob.cls":

```
1 global class DailyJob implements Schedulable {
2     global void execute(SchedulableContext sc) {
3         System.debug('Daily Job running...');
4     }
5 }
```
- Terminal:** Shows the command history restored on Microsoft Windows 10.
- Status Bar:** Shows the file path "D:\PaymentTrackinginBanking", line 5, column 3, and other development settings.

Future Methods

- Implemented future call for sending async email notifications.

Exception Handling

- Created a centralized ErrorHandler class to log and handle exceptions.
- Ensured smooth error reporting without stopping automation.

LoanHelper.cls

```
public class LoanHelper {  
  
    public static void beforeInsert(List<Loan__c> newLoans) {  
  
        try {  
  
            for (Loan__c loan : newLoans) {  
  
                if (loan.Amount__c <= 0) {  
  
                    throw new PaymentTrackingException('Loan Amount must  
be greater than 0.');                }  
  
            }  
  
        } catch (Exception ex) {  
  
            String msg = ErrorHandler.logError(ex,  
'LoanHelper.beforeInsert');  
  
            ErrorHandler.notifyAdmin('Loan Insert Error', msg);  
  
            throw ex;  
  
        }  
  
    }  
}
```

```
1 public class LoanHelper {
2     public static void beforeInsert(List<Loan__c> newLoans) {
3         try {
4             for (Loan__c loan : newLoans) {
5                 if (loan.Amount__c <= 0) {
6                     throw new PaymentTrackingException('Loan Amount must be greater than 0.');
7                 }
8             }
9         } catch (Exception ex) {
10            String msg = ErrorHandler.logError(ex, 'LoanHelper.beforeInsert');
11            ErrorHandler.notifyAdmin('Loan Insert Error', msg);
12            throw ex;
13        }
14    }
15 }
16
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS HISTORY

Microsoft Windows [Version 10.0.26100.6584]
(c) Microsoft Corporation. All rights reserved.

D:\PaymentTrackinginBanking>sf project deploy start --source-dir force-app/main/default/classes --target-org DevOrg
+ History restored

Microsoft Windows [Version 10.0.26100.6584]
(c) Microsoft Corporation. All rights reserved.

D:\PaymentTrackinginBanking>

PaymentHelper.cls

```
public class PaymentHandler {
    public static void beforeInsert(List<Payment__c> newPayments) {
        try {
            for (Payment__c pay : newPayments) {
                if (pay.Payment_Date__c == null) {
                    throw new PaymentTrackingException('Payment Date is required.');
                }
            }
        } catch (Exception ex) {
            String msg = ErrorHandler.logError(ex, 'PaymentHandler.beforeInsert');
            ErrorHandler.notifyAdmin('Payment Insert Error', msg);
            throw ex;
        }
    }

    public static void afterUpdate(List<Payment__c> updatedPayments)
    {
        try {
```

```

        for (Payment__c pay : updatedPayments) {
            if (pay.Status__c == 'Overdue') {
                System.debug('Payment ' + pay.Id + ' is overdue. Creating
recovery...');
            }
        }
    } catch (Exception ex) {
        String msg = ErrorHandler.logError(ex,
'PaymentHandler.afterUpdate');
        ErrorHandler.notifyAdmin('Payment Update Error', msg);
    }
}
}

```

The screenshot shows the Salesforce Dev Console interface. On the left is the Explorer sidebar with project files like LoanBatch.cls, PaymentHelper.cls, and Test_PaymentHandlers.cls. The main area displays the code for PaymentHelper.cls:

```

1  public class PaymentHandler {
2      public static void beforeInsert(List<Payment__c> newPayments) {
3          try {
4              for (Payment__c pay : newPayments) {
5                  if (pay.Payment_Date__c == null) {
6                      throw new PaymentTrackingException('Payment Date is required.');
7                  }
8              }
9          } catch (Exception ex) {
10             String msg = ErrorHandler.logError(ex, 'PaymentHandler.beforeInsert');
11             ErrorHandler.notifyAdmin('Payment Insert Error', msg);
12             throw ex;
13         }
14     }
15
16     public static void afterUpdate(List<Payment__c> updatedPayments) {
17         try {

```

Below the code, the terminal window shows deployment logs:

```

Microsoft Windows [Version 10.0.26100.6584]
(c) Microsoft Corporation. All rights reserved.

D:\PaymentTrackinginBanking>sf project deploy start --source-dir force-app/main/default/classes --target-org
DevOrg
[*] History restored

Microsoft Windows [Version 10.0.26100.6584]
(c) Microsoft Corporation. All rights reserved.

D:\PaymentTrackinginBanking>

```

The status bar at the bottom indicates the file is main*, line 13, column 10, with 4 spaces, using UTF-8 encoding, and is an Apex file.

Test Classes

- Wrote test classes for each handler & trigger to validate functionality.

LoanHelperTest.cls

```
@isTest
public class LoanHelperTest {

    @isTest
    static void testLoanBeforeInsert() {
        Loan__c loan = new Loan__c(
            Principal_Amount__c = 50000,
            Term_Months__c = 12,
            Interest_Rate__c = 10
        );

        Test.startTest();
        insert loan;
        Test.stopTest();

        System.assertEquals(null, loan.Id);
    }
}
```

The screenshot shows the Salesforce IDE interface. The top navigation bar includes File, Edit, Selection, View, Go, Run, and other options. The title bar says "PaymenttrackinginBanking". The left sidebar has sections for EXPLORER, PAYMENTTRACKINGINBANKING, force-app, and main. In the EXPLORER section, "classes" is expanded, showing files like LoanBatch.cls, LoanController.cls, PaymentController.cls, etc., and "LoanHelperTest.cls" is selected. The main editor area displays the Apex code for LoanHelperTest.cls. The code defines a class with a test method named testLoanBeforeInsert(). This method creates a new Loan__c record with specific values for Principal_Amount__c, Term_Months__c, and Interest_Rate__c. It then uses Test.startTest(), insert loan;, Test.stopTest(), and System.assertEquals(null, loan.Id) to verify the record was inserted correctly. The code editor shows syntax highlighting and line numbers from 1 to 19.

```
File Edit Selection View Go Run ...
PaymenttrackinginBanking
EXPLORER
PAYMENTTRACKINGINBANKING
force-app / main / default
classes
  LoanBatch.cls
  LoanBatch.cls-meta.xml
  LoanController.cls
  LoanController.cls-meta.xml
  LoanDetailsController.cls
  LoanExternalService.cls
  LoanExternalService.cls-meta.xml
  LoanHandler.cls
  LoanHandler.cls-meta.xml
  LoanHelper.cls
  LoanHelperTest.cls
  PaymentController.cls
  PaymentController.cls-meta.xml
  PaymentHandler.cls
  PaymentHandler.cls-meta.xml
  PaymentHandlerTest.cls
  PaymentHelper.cls
  PaymentReminderQueue.cls
  PaymentReminderQueue.cls-meta.xml
  PaymentSearchDemo.cls
  PaymentSearchDemo.cls-meta.xml
OUTLINE
TIMELINE
RUNNING TASKS
main* 0 0 0 DevOrg 0 0
Ln 10, Col 11 Spaces: 4 UTF-8 CRLF {} Apex Prettier
```

```
1  @isTest
2  public class LoanHelperTest {
3
4      @isTest
5      static void testLoanBeforeInsert() {
6          Loan__c loan = new Loan__c(
7              Principal_Amount__c = 50000,
8              Term_Months__c = 12,
9              Interest_Rate__c = 10
10         );
11
12         Test.startTest();
13         insert loan;
14         Test.stopTest();
15
16         System.assertEquals(null, loan.Id);
17     }
18 }
```

Test_PaymentHandlers.cls

```
@IsTest
```

```
private class Test_PaymentHandlers {
```

```
    @IsTest static void testPaymentInsertAndLoanClose() {
```

```
        Loan__c loan = new Loan__c(Name='TestLoan',  
        Principal_Amount__c = 1000.00, Status__c='Active');  
        insert loan;
```

```
        Payment__c p1 = new Payment__c(Name='P1', Loan__c = loan.Id,  
        Payment_Date__c = Date.today().addDays(1), Amount__c = 500,  
        Status__c='Pending');
```

```
        Payment__c p2 = new Payment__c(Name='P2', Loan__c = loan.Id,  
        Payment_Date__c = Date.today().addDays(2), Amount__c = 500,  
        Status__c='Pending');
```

```
        insert new List<Payment__c>{p1,p2};
```

```
        p1.Status__c = 'Paid';
```

```
        p2.Status__c = 'Paid';
```

```
        update new List<Payment__c>{p1,p2};
```

```
        Loan__c refreshed = [SELECT Id, Status__c FROM Loan__c WHERE  
        Id = :loan.Id];
```

```
        System.assertEquals('Closed', refreshed.Status__c);
```

```
}
```

```
@IsTest static void testPaymentOverdueBeforeInsert() {
```

```
    Loan__c loan = new Loan__c(Name='L2',  
    Principal_Amount__c=1000.00);  
    insert loan;
```

```

        Payment__c p = new Payment__c(Name='past', Loan__c =
loan.Id, Payment_Date__c = Date.today().addDays(-2), Amount__c
= 100, Status__c=null);
        insert p;
        Payment__c got = [SELECT Id, Status__c FROM Payment__c
WHERE Id = :p.Id];
        System.assertEquals('Overdue', got.Status__c);
    }
}

```

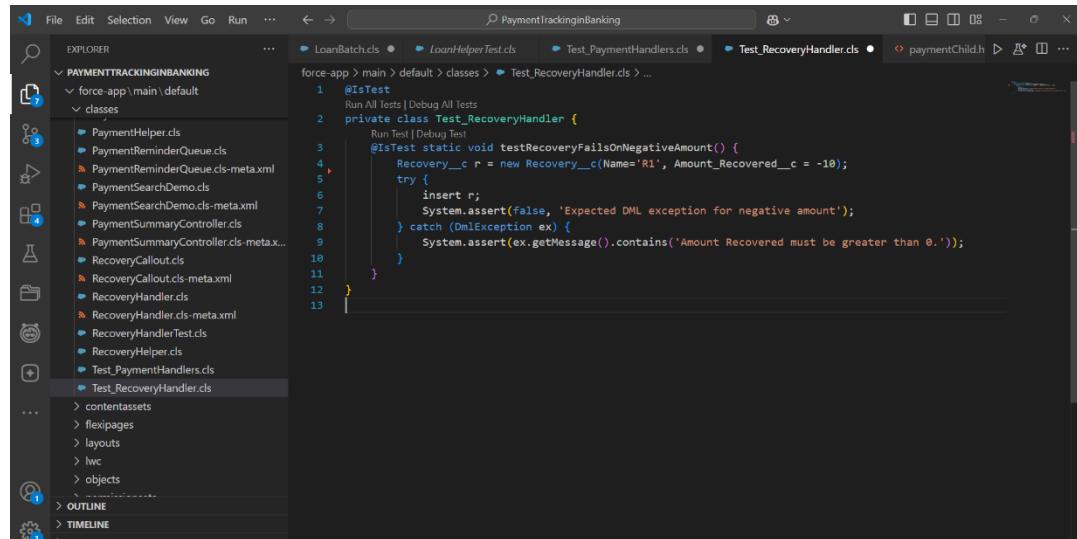
```

1  @IsTest
2  private class Test_PaymentHandlers {
3      @IsTest static void testPaymentInsertAndLoanClose() {
4          Loan__c loan = new Loan__c(Name='TestLoan', Principal_Amount__c = 1000.00, Status__c='Active');
5          insert loan;
6
7          Payment__c p1 = new Payment__c(Name='P1', Loan__c = loan.Id, Payment_Date__c = Date.today().addDays(-2));
8          Payment__c p2 = new Payment__c(Name='P2', Loan__c = loan.Id, Payment_Date__c = Date.today().addDays(-2));
9
10         insert new List<Payment__c>{p1,p2};
11
12         p1.Status__c = 'Paid';
13         p2.Status__c = 'Paid';
14         update new List<Payment__c>{p1,p2};
15
16         Loan__c refreshed = [SELECT Id, Status__c FROM Loan__c WHERE Id = :loan.Id];
17         System.assertEquals('Closed', refreshed.Status__c);
18     }
19
20     @IsTest static void testPaymentOverdueBeforeInsert() {
21         Loan__c loan = new Loan__c(Name='L2', Principal_Amount__c=1000.00);
22         insert loan;
23
24         Payment__c p = new Payment__c(Name='past', Loan__c = loan.Id, Payment_Date__c = Date.today().addDays(-2));
25
26         System.assertEquals('Overdue', p.Status__c);
27     }
}

```

Test_RecoveryHandlers.cls

```
@IsTest
private class Test_RecoveryHandler {
    @IsTest static void testRecoveryFailsOnNegativeAmount() {
        Recovery__c r = new Recovery__c(Name='R1',
        Amount_Recovered__c = -10);
        try {
            insert r;
            System.assert(false, 'Expected DML exception for negative
amount');
        } catch (DmlException ex) {
            System.assert(ex.getMessage().contains('Amount Recovered
must be greater than 0.'));
        }
    }
}
```



```
}
```

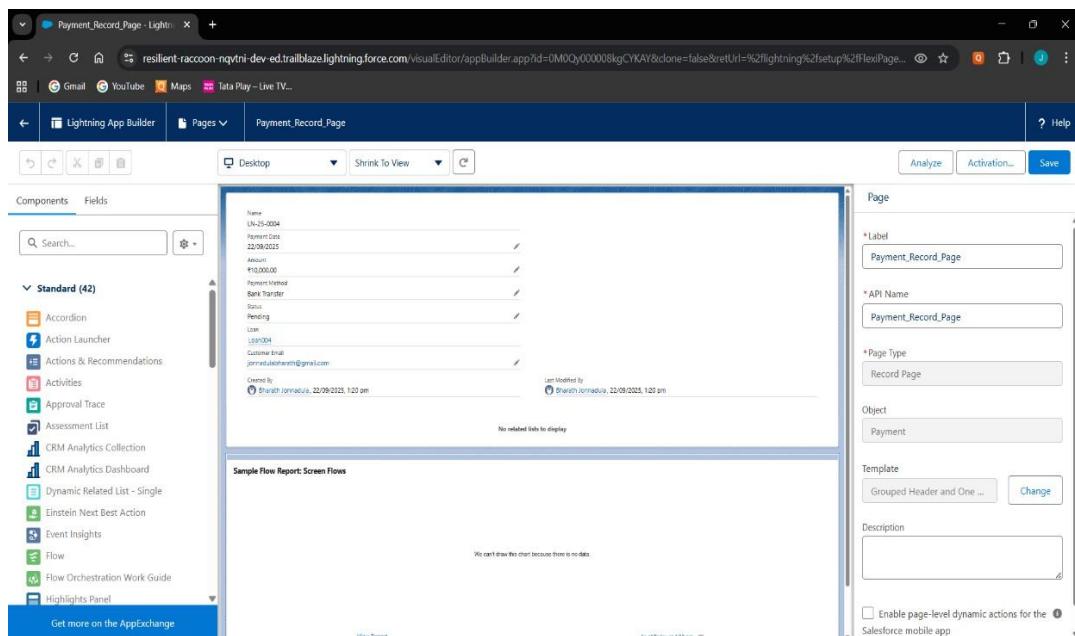
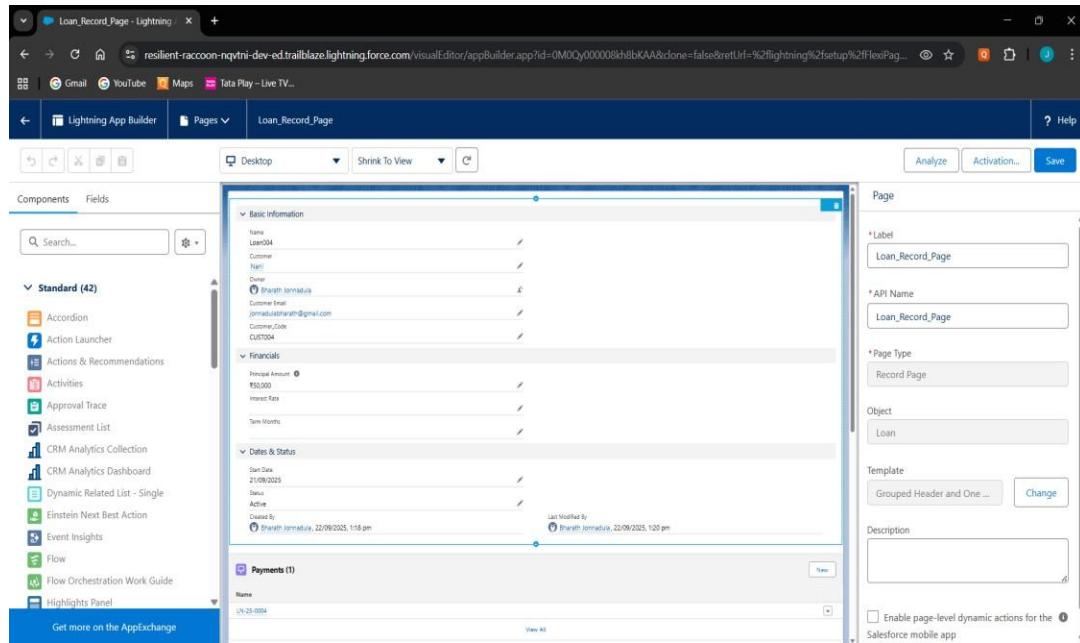
Asynchronous Processing

- Combined Batch, Queueable, Scheduled, and Future methods for async operations.

Phase 6: User Interface Development

Lightning App Builder

- Created a custom Loan_Record_Page, Payment_Home_Page, Payment_Record_Page, Recovery_record_Page

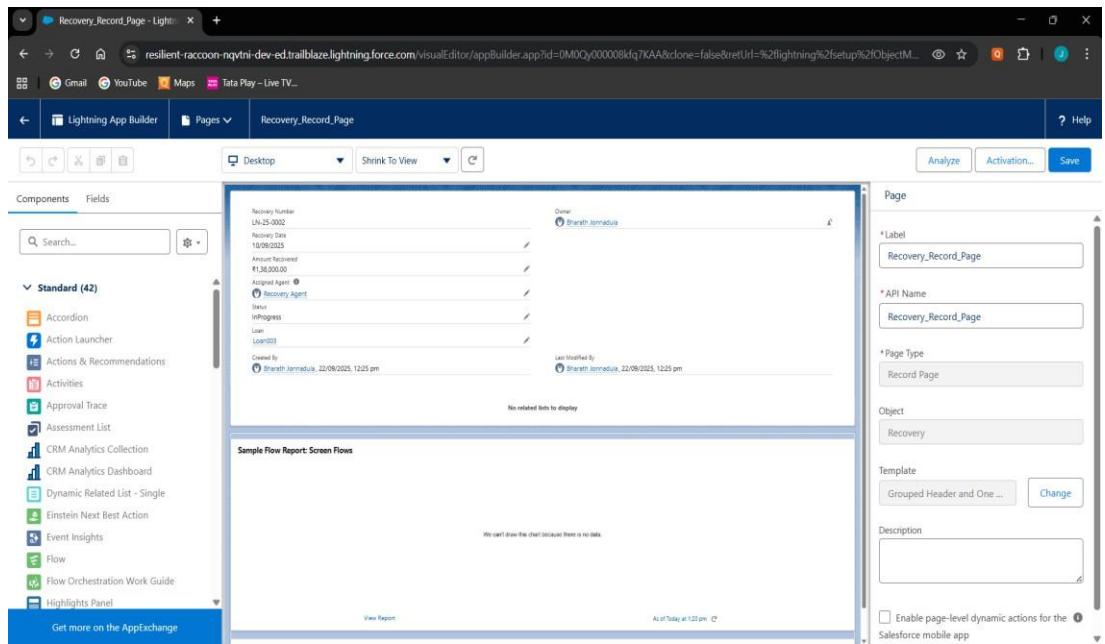


Record Pages

- Built custom Record Pages for Loan and Payment objects using Lightning App Builder.

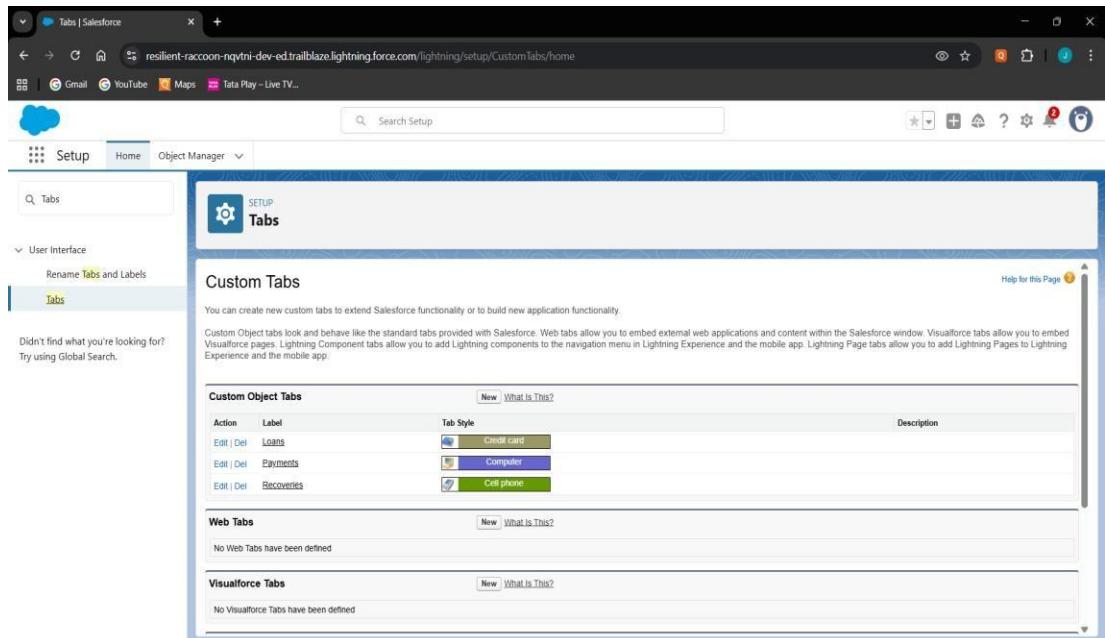
The screenshot shows the Lightning App Builder interface for creating a custom Record Page named "Loan_Record_Page". The "Basic Information" section includes fields for Name (User004), Owner (Bharath Jommalu), and various financial details like Principal Amount (\$50,000) and Interest Rate. The "Payments" section lists one payment entry. The "Page" configuration panel on the right specifies the Label as "Loan_Record_Page", API Name as "Loan_Record_Page", and Page Type as "Record Page". The Object is set to "Loan". The Template is "Grouped Header and One ...". A checkbox for "Enable page-level dynamic actions for the Salesforce mobile app" is checked.

The screenshot shows the Lightning App Builder interface for creating a custom Record Page named "Payment_Record_Page". The "Basic Information" section includes fields for Name (LN-25-0004), Payment Date (22/09/2025), and amount (\$10,000.00). The "Page" configuration panel on the right specifies the Label as "Payment_Record_Page", API Name as "Payment_Record_Page", and Page Type as "Record Page". The Object is set to "Payment". The Template is "Grouped Header and One ...". A checkbox for "Enable page-level dynamic actions for the Salesforce mobile app" is checked.



Tabs

- Added custom tabs for Loan, Payment, and Recovery objects.



Home Page Layouts

- Customized the Home Page with components like Recent Loans, Pending Payments

The screenshot shows the Salesforce Lightning App Builder interface. On the left, there's a sidebar with 'Feature Settings' expanded, showing 'Service' (Field Service), 'Field Service Mobile', 'Field Service Mobile App Builder', and 'User Interface' (Lightning App Builder selected). Below the sidebar, a message says 'Didn't find what you're looking for? Try using Global Search.' The main area is titled 'Lightning App Builder' and contains a section titled 'Lightning Pages'. It lists five pages with the following details:

Action	Label	Name	Namespace Prefix	Description	Type	Created By	Last Modified By
Edit Clone Del	Loan_Record_Page	Loan_Record_Page			Record Page	Bjorn	Bjorn
Edit Clone Del	Payment_Home_Page	Payment_Home_Page			Home Page	Bjorn	Bjorn
Edit Clone Del	Payment_Record_Page	Payment_Record_Page			Record Page	Bjorn	Bjorn
Edit Clone Del	Recovery_Record_Page	Recovery_Record_Page			Record Page	Bjorn	Bjorn

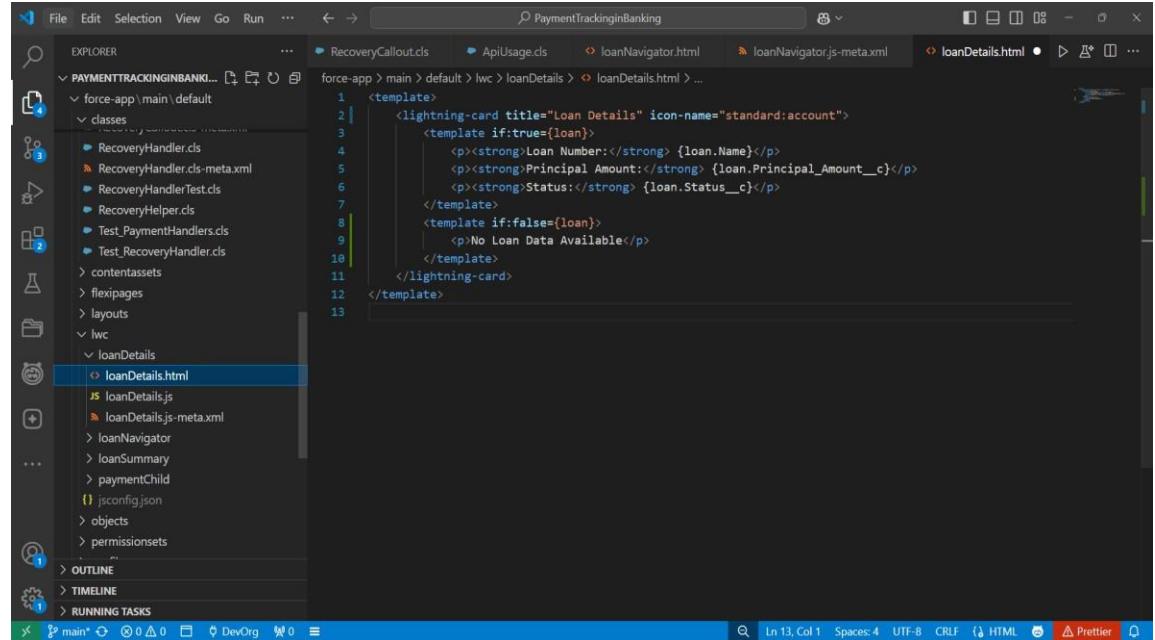
Utility Bar

- Switch between loan & payment records.

The screenshot shows the 'App Settings' page in the Lightning App Builder. The left sidebar has 'Navigation Items' selected under 'App Settings'. The main area is titled 'Navigation Items' and contains a section for 'Available Items' and 'Selected Items'. The 'Available Items' list includes: Accounts, All Sites, Alternative Payment Methods, Analytics, App Launcher, Approval Requests, Approval Submission Details, Approval Submissions, Approval Work Items, and Asset Action Sources. The 'Selected Items' list contains: Dashboards, Reports, Loans, Payments, and Recoveries. Navigation arrows on the right side of the 'Selected Items' list allow users to move items between the two lists.

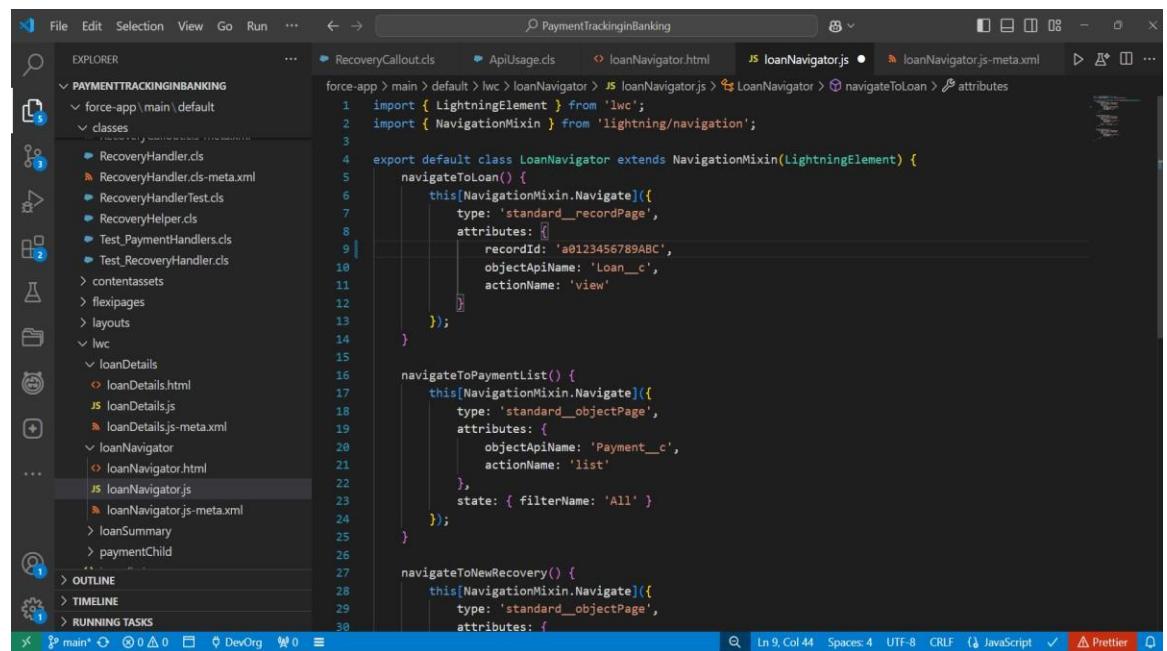
LWC (Lightning Web Components)

- Built LWC components (loanSummary, loanDetails, paymentChild).



The screenshot shows the VS Code interface with the title bar "PaymentTrackinginBanking". The Explorer sidebar on the left shows a project structure under "PAYMENTTRACKINGINBANKING". In the center, the code editor displays the template for an LWC component named "loanDetails.html". The template uses the lightning-card element to display loan details, including the loan number, principal amount, and status. A conditional template is shown for cases where no loan data is available.

```
<template>
  <lightning-card title="Loan Details" icon-name="standard:account">
    <template if:true={loan}>
      <p><strong>Loan Number:</strong> {loan.Name}</p>
      <p><strong>Principal Amount:</strong> {loan.Principal_Amount__c}</p>
      <p><strong>Status:</strong> {loan.Status__c}</p>
    </template>
    <template if:false={loan}>
      <p>No Loan Data Available</p>
    </template>
  </lightning-card>
</template>
```



The screenshot shows the VS Code interface with the title bar "PaymentTrackinginBanking". The Explorer sidebar on the left shows a project structure under "PAYMENTTRACKINGINBANKING". In the center, the code editor displays a JavaScript class named "loanNavigator.js". The class extends the NavigationMixin and includes methods for navigating to a loan record page, a payment list, and a new recovery record. It uses the LightningElement and NavigationMixin from the lwc library.

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

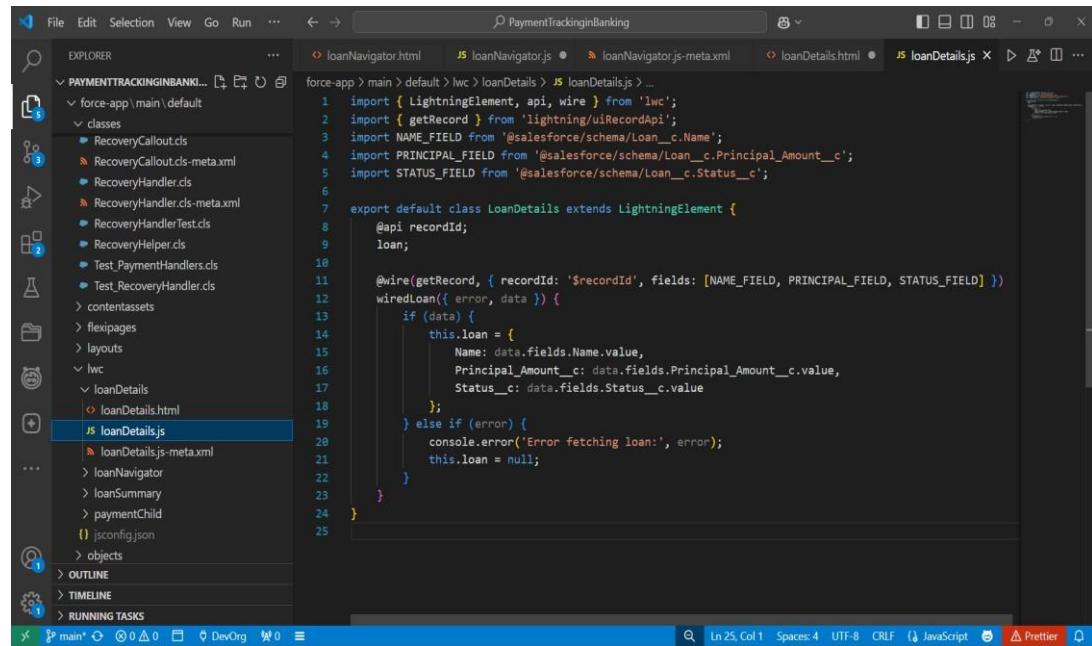
export default class LoanNavigator extends NavigationMixin(LightningElement) {
  navigateToLoan() {
    this[NavigationMixin.Navigate]({
      type: 'standard__recordPage',
      attributes: {
        recordId: 'a0123456789ABC',
        objectApiName: 'Loan__c',
        actionName: 'view'
      }
    });
  }

  navigateToPaymentList() {
    this[NavigationMixin.Navigate]({
      type: 'standard__objectPage',
      attributes: {
        objectApiName: 'Payment__c',
        actionName: 'list'
      },
      state: { filterName: 'All' }
    });
  }

  navigateToNewRecovery() {
    this[NavigationMixin.Navigate]({
      type: 'standard__objectPage',
      attributes: {
        objectApiName: 'Recovery__c',
        actionName: 'new'
      }
    });
  }
}
```

Wire Adapters

- Used @wire(getRecord) to fetch Loan fields directly in LWC.



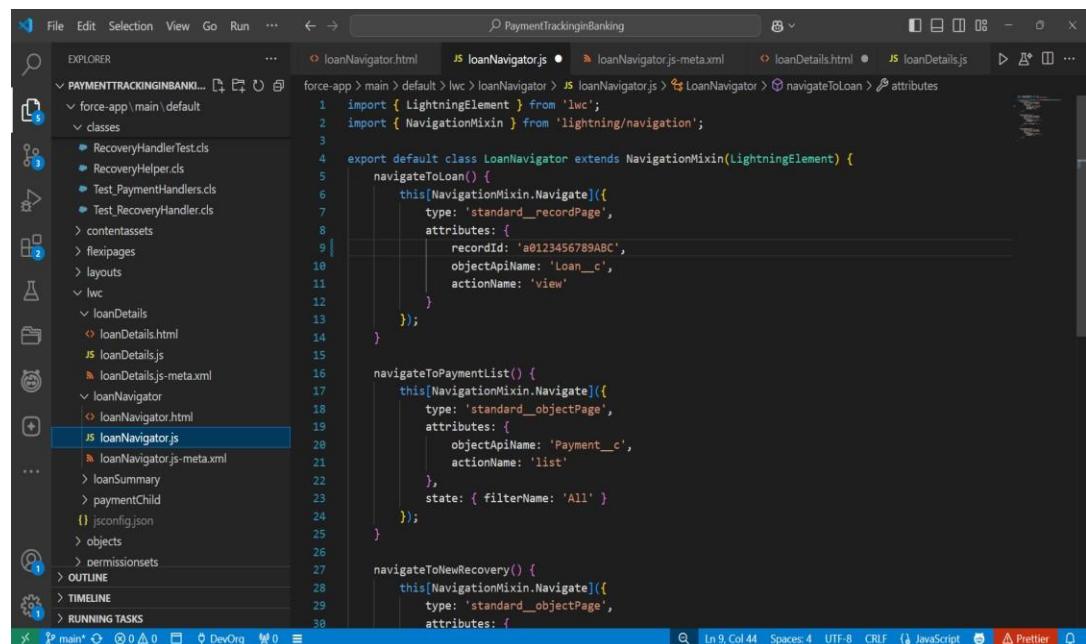
```
import { LightningElement, api, wire } from 'lwc';
import { getRecord } from 'lightning/uiRecordApi';
import NAME_FIELD from '@salesforce/schema/Loan__c.Name';
import PRINCIPAL_FIELD from '@salesforce/schema/Loan__c.Principal_Amount__c';
import STATUS_FIELD from '@salesforce/schema/Loan__c.Status__c';

export default class LoanDetails extends LightningElement {
    @api recordId;
    loan;

    @wire(getRecord, { recordId: '$recordId', fields: [NAME_FIELD, PRINCIPAL_FIELD, STATUS_FIELD] })
    wiredLoan({ error, data }) {
        if (data) {
            this.loan = {
                Name: data.fields.Name.value,
                Principal_Amount__c: data.fields.Principal_Amount__c.value,
                Status__c: data.fields.Status__c.value
            };
        } else if (error) {
            console.error('Error fetching loan:', error);
            this.loan = null;
        }
    }
}
```

Navigation Service

- Implemented NavigationMixin to let users navigate from Payment → Loan detail page.
- Improves user experience with one-click record redirection.



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

export default class LoanNavigator extends NavigationMixin(LightningElement) {
    navigateToLoan() {
        this[NavigationMixin.Navigate]({
            type: 'standard__recordPage',
            attributes: {
                recordId: 'a0123456789ABC',
                objectApiName: 'Loan__c',
                actionName: 'view'
            }
        });
    }

    navigateToPaymentList() {
        this[NavigationMixin.Navigate]({
            type: 'standard__objectPage',
            attributes: {
                objectApiName: 'Payment__c',
                actionName: 'list'
            },
            state: { filterName: 'All' }
        });
    }

    navigateToNewRecovery() {
        this[NavigationMixin.Navigate]({
            type: 'standard__objectPage',
            attributes: {
                objectApiName: 'RecoveryCallout__c'
            }
        });
    }
}
```

Phase 7: Integration & External Access

Named Credentials

- Used to securely store API endpoints + authentication details for external systems.

The screenshot shows the 'Named Credentials' setup page in Salesforce. The 'BankAPI' credential is selected. The configuration includes:

- Label:** BankAPI
- URL:** https://api.mybank.com
- Enabled for Callouts:** Checked
- Authentication:** External Credential (BankAPI_Credential) and Client Certificate
- Callout Options:** Generate Authorization Header (checked), Allow Formulas in HTTP Header (unchecked), and Allow Formulas in HTTP Body (unchecked)

External Services

- Lets Salesforce call external APIs using schema (Swagger/OpenAPI).

The screenshot shows the 'External Services' setup page in Salesforce. The 'BankPaymentService' integration is selected. The configuration includes:

- Service name:** BankPaymentService
- Creation source:** From API specification
- Description:** External Banking API for Loan Validation and Payments
- Type:** OpenApi
- File Name:** BankAPI_OpenAPI.json
- Named credentials:** BankAPI

Operations: 2 items - Sorted by Operation Name

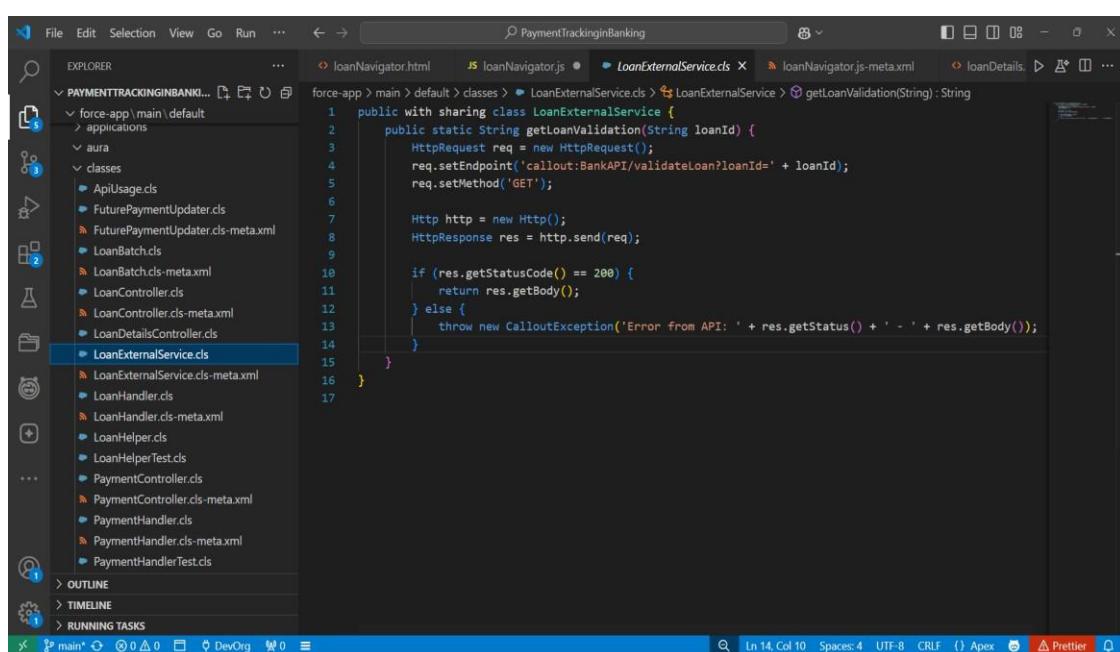
Operation Name ↑	Description	Input parameters	Output parameters
getValidateLoan	Validate Loan	loanId	responseCode, 200, default
postMakePayment	Make a Payment	loanId, amount	responseCode, 200, default

Web Services (REST/SOAP)

- Allow external system to fetch Loan details.

LoanExternalService.cls

```
public with sharing class LoanExternalService {  
  
    public static String getLoanValidation(String loanId) {  
  
        HttpRequest req = new HttpRequest();  
  
        req.setEndpoint('callout:BankAPI/validateLoan?loanId=' + loanId);  
  
        req.setMethod('GET');  
  
        Http http = new Http();  
  
        HttpResponse res = http.send(req);  
  
        if (res.getStatusCode() == 200) {  
  
            return res.getBody();} else {  
  
            throw new CalloutException('Error from API: ' + res.getStatus() + ' - ' + res.getBody());}}}
```

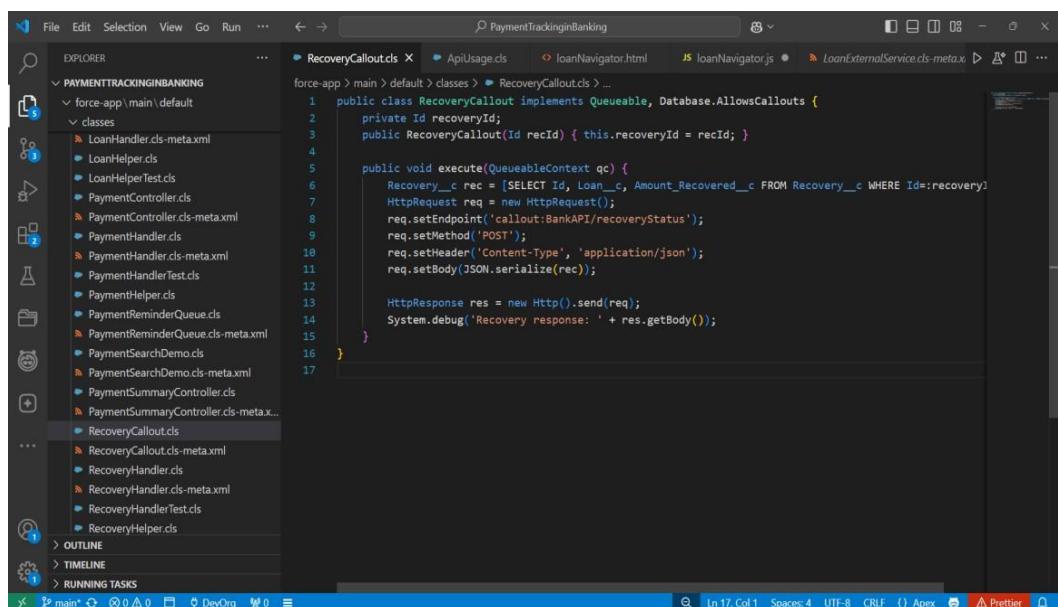


Callouts

- Used when Salesforce sends data to external services.

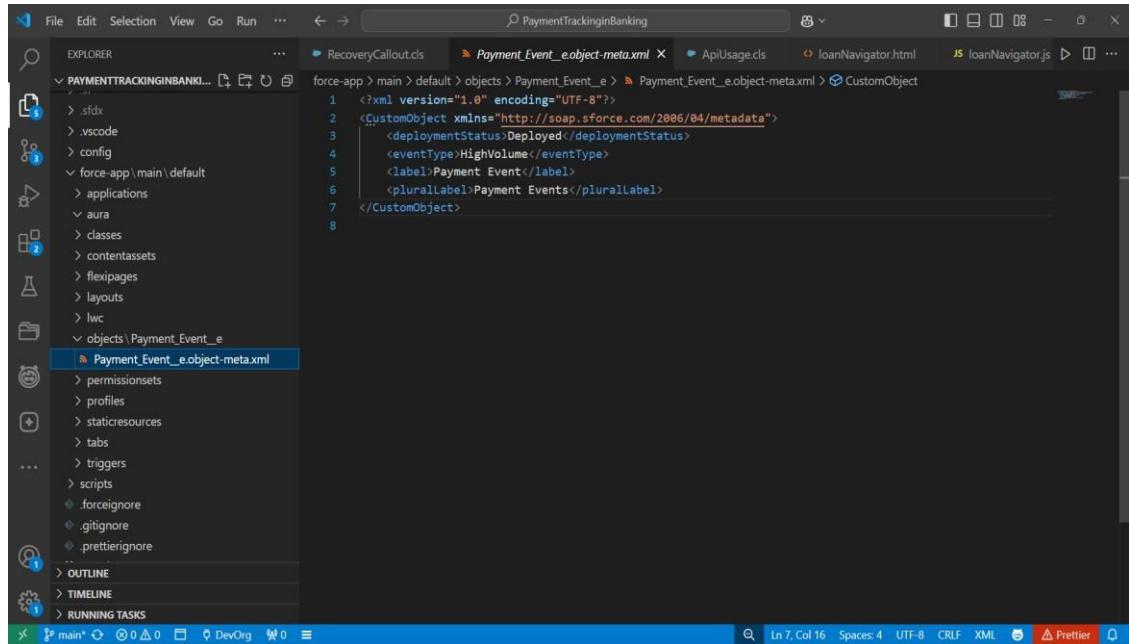
RecoveryCallout.cls

```
public class RecoveryCallout implements Queueable,  
    Database.AllowsCallouts {  
  
    private Id recoveryId;  
  
    public RecoveryCallout(Id recId) { this.recoveryId = recId; }  
  
    public void execute(QueueableContext qc) {  
  
        Recovery__c rec = [SELECT Id, Loan__c, Amount_Recovered__c  
        FROM Recovery__c WHERE Id=:recoveryId];  
  
        HttpRequest req = new HttpRequest();  
  
        req.setEndpoint('callout:BankAPI/recoveryStatus');  
        req.setMethod('POST');  
  
        req.setHeader('Content-Type', 'application/json');  
  
        req.setBody(JSON.serialize(rec));  
  
        HttpResponse res = new Http().send(req);  
  
        System.debug('Recovery response: ' + res.getBody());}}}
```



Platform Events

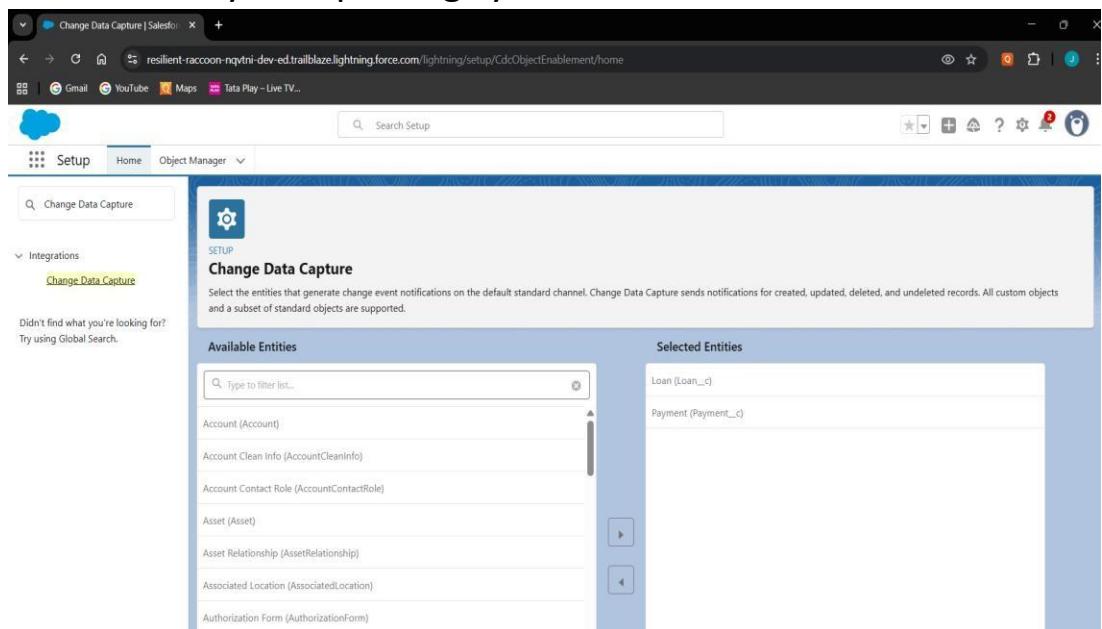
- Publish event when a Payment is Overdue (integrates with external collectors).



```
<?xml version="1.0" encoding="UTF-8"?>
<CustomObject xmlns="http://soap.sforce.com/2006/04/metadata">
    <deploymentStatus>Deployed</deploymentStatus>
    <eventType>HighVolume</eventType>
    <label>Payment Event</label>
    <pluralLabel>Payment Events</pluralLabel>
</CustomObject>
```

Change Data Capture (CDC)

- Streams real-time changes of Salesforce records to external systems.
- Any update in Payment_c record can be pushed automatically to reporting system.

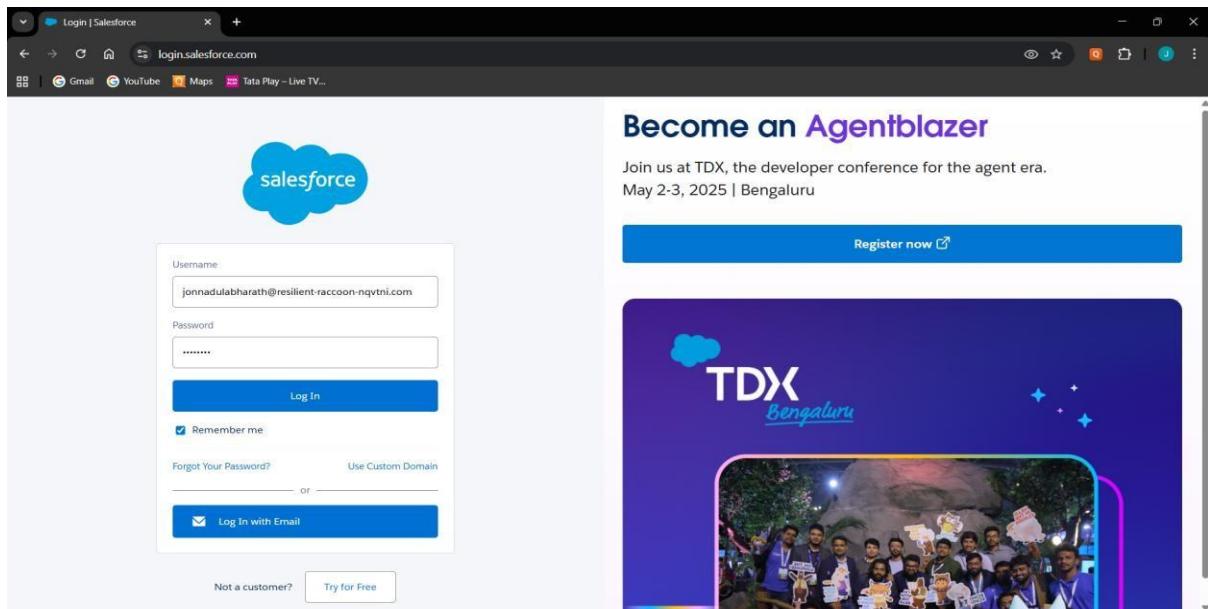


The screenshot shows the Salesforce Setup interface with the following details:

- Page Title:** Change Data Capture | Salesforce
- Header:** Search Setup
- Left Navigation:** Setup, Home, Object Manager
- Section:** Integrations, Change Data Capture
- Content:**
 - Available Entities:** A list of standard and custom objects including Account, Asset, and Payment.
 - Selected Entities:** A list of entities selected for Change Data Capture, including Loan and Payment.

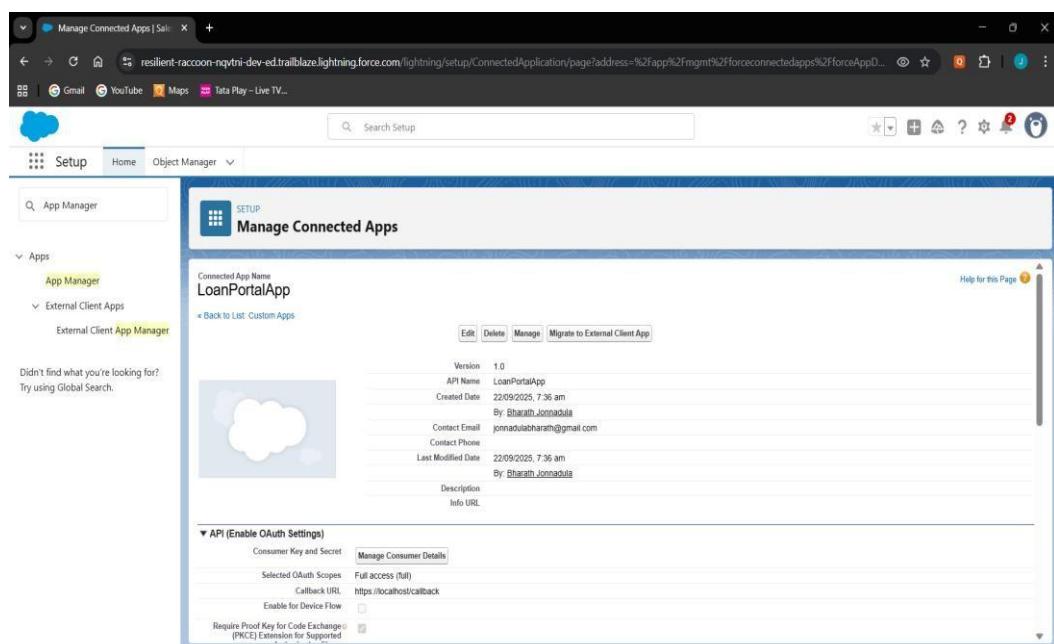
Salesforce Connect

- Provides real-time view of external data without storing in Salesforce.



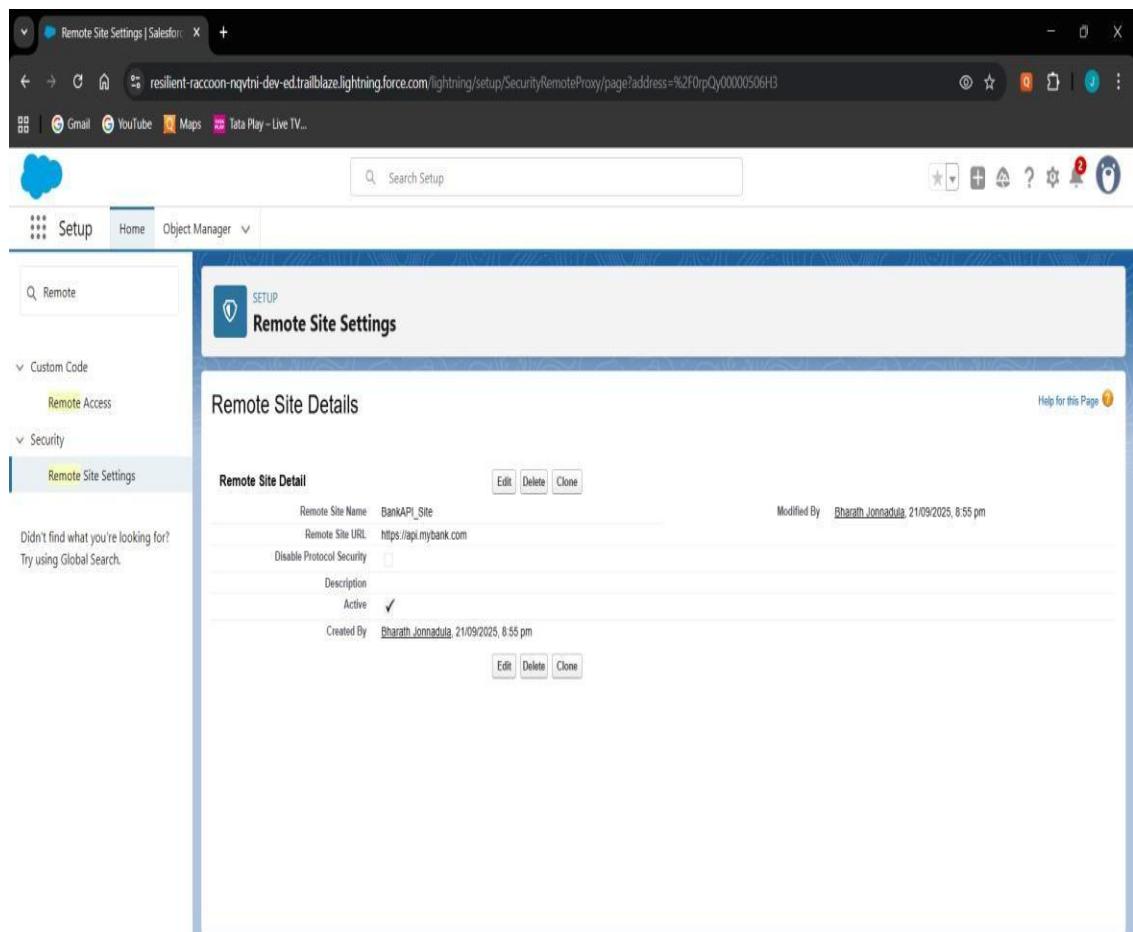
OAuth & Authentication

- Enables secure access for external apps to Salesforce APIs.



Remote Site Settings

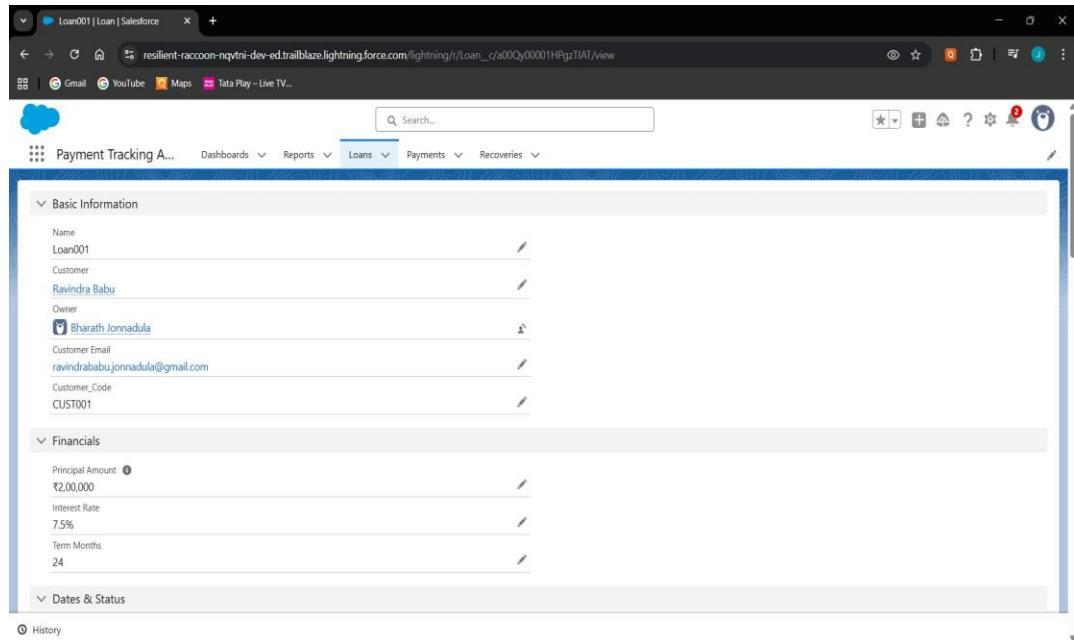
- Required to allow outbound callouts to untrusted domains.
- Add `https://api.bank.com` as a Remote Site Setting before making callouts.



Phase 8: Data Management & Deployment

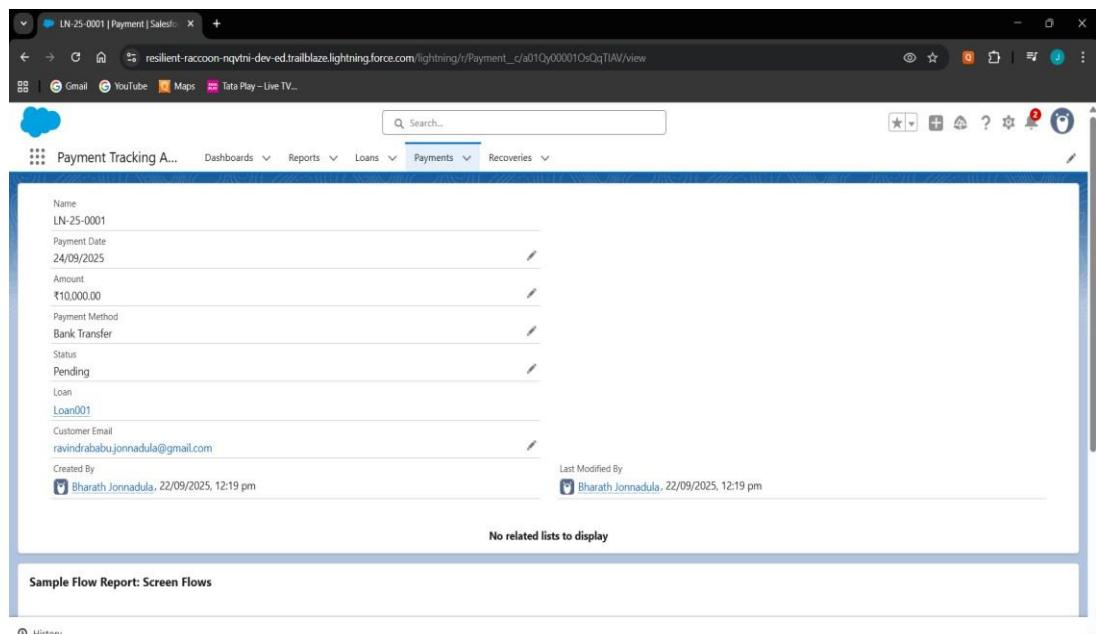
Data Import Wizard

- Used Data Import Wizard to insert sample data into Loan, Payment, and Recovery objects.



The screenshot shows the 'Basic Information' section of a new loan record. The fields are as follows:

Field	Value
Name	Loan001
Customer	Ravindra Babu
Owner	Bharath Jonnadula
Customer Email	ravindrababu.jonnadula@gmail.com
Customer Code	CUST001



The screenshot shows the 'Payments' section of a new payment record. The fields are as follows:

Field	Value
Name	LN-25-0001
Payment Date	24/09/2025
Amount	₹10,000.00
Payment Method	Bank Transfer
Status	Pending
Loan	Loan001
Customer Email	ravindrababu.jonnadula@gmail.com
Created By	Bharath Jonnadula, 22/09/2025, 12:19 pm
Last Modified By	Bharath Jonnadula, 22/09/2025, 12:19 pm

No related lists to display

Sample Flow Report: Screen Flows

The screenshot shows a Salesforce Lightning page for a Recovery record. The record details are as follows:

- Recovery Number: LN-25-0001
- Recovery Date: 05/09/2025
- Amount Recovered: ₹90,000.00
- Assigned Agent: Recovery Agent
- Status: InProgress
- Loan: Loan002
- Created By: Bharath Jonnadula, 22/09/2025, 12:24 pm
- Last Modified By: Bharath Jonnadula, 22/09/2025, 12:24 pm

The page also displays a message: "No related lists to display". Below the main content, there is a section titled "Sample Flow Report: Screen Flows".

Data Loader

- Data Loader for bulk upload, but not required since records were limited.

Data Export & Backup

- Verified Data Export option in Setup for backup.

The screenshot shows the "Data Export" setup page under the "Monthly Export Service". The configuration includes:

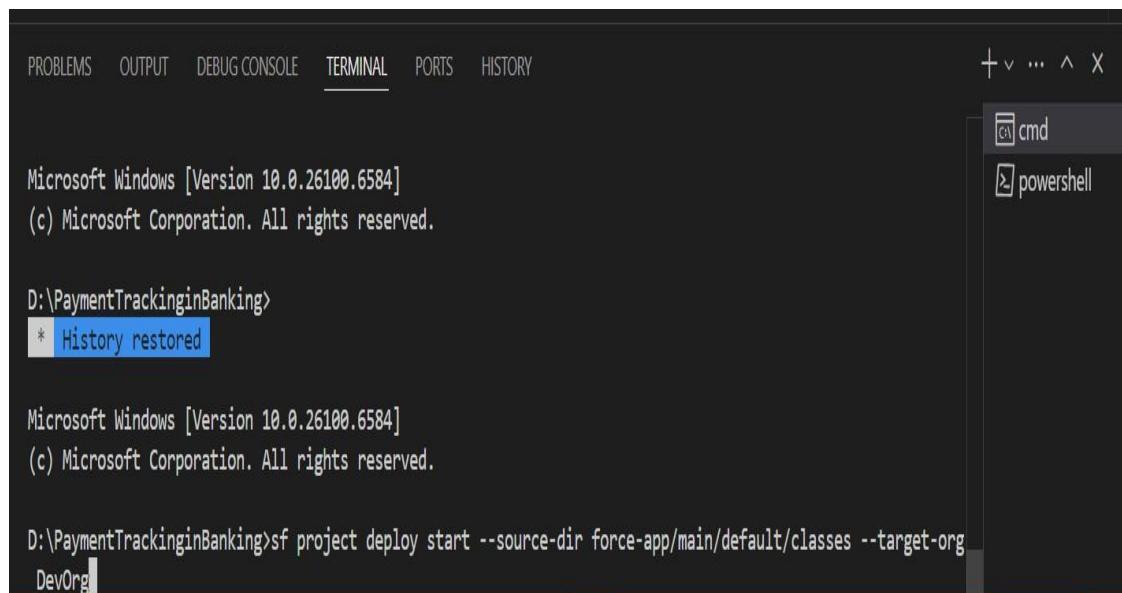
- Export File Encoding: ISO-8859-1 (General US & Western European, ISO-LATIN-1)
- Include images, documents, and attachments: Unchecked
- Include Salesforce Files and Salesforce CRM Content document versions: Unchecked
- Replace carriage returns with spaces: Checked

The "Exported Data" section allows selecting data types to include in the export. The "Include all data" checkbox is checked, and the following data types are selected:

- Contract
- Asset
- Lead
- BusinessProcess
- Campaign
- CaseContactRole
- ContentDocumentLink
- ContractContactRole
- Order
- Account
- Partner
- NotificationMember
- CampaignMember
- CaseHistory2
- ContentVersion
- EmailDisclaimer
- OrderItem
- Contact
- Product2
- UserRole
- Case
- CaseSolution
- ContentVersionMap
- EmailMessage

VS Code & SFDX

- Used VS Code with SFDX CLI for writing and deploying Apex Classes, Triggers, and LWC.
- Successfully pushed/pulled metadata between local project and Salesforce Org



The screenshot shows the VS Code interface with the Terminal tab selected. The terminal window displays the following command-line session:

```
Microsoft Windows [Version 10.0.26100.6584]
(c) Microsoft Corporation. All rights reserved.

D:\PaymentTrackinginBanking>
* History restored

Microsoft Windows [Version 10.0.26100.6584]
(c) Microsoft Corporation. All rights reserved.

D:\PaymentTrackinginBanking>sf project deploy start --source-dir force-app/main/default/classes --target-org DevOrg
```

The terminal also shows a dropdown menu for selecting between 'cmd' and 'powershell' as the terminal type.

