Phase 1 - Problem Understanding & Industry Analysis

Requirement Gathering

The Helping Hands Foundation project requires a Salesforce-based solution to streamline the management of volunteers, donations, and events. The primary needs identified include:

- Tracking volunteer details, skills, and availability.
- Recording and monitoring donations with amount, date, and donor details.
- Organizing events and associating them with volunteers.
- Maintaining transparency and generating reports for stakeholders.
- Ensuring secure access based on roles.

Stakeholder Analysis

Key stakeholders in the Helping Hands Foundation include:

- CEO: Oversees the overall operations and ensures alignment with the organization's mission.
- CFO: Manages financial aspects, donation tracking, and compliance.
- Financial Officer: Records transactions, monitors donation flow, and ensures proper allocation.
- Donation Manager: Handles donor engagement, donation collection, and donor relations.
- Volunteer Coordinator: Manages volunteer recruitment, assignments, and scheduling.
- Donors: External stakeholders providing financial support.
- Volunteers: Community members who actively participate in events and initiatives.

Business Process Mapping

The current manual processes include:

1. Volunteers register via paper forms or spreadsheets.
2. Donations are tracked manually in ledgers or Excel files.
3. Event planning and volunteer assignments are managed over phone calls or emails.

Proposed Salesforce Solution:

- Volunteers can be registered in Salesforce with their availability and skills.
- Donations are logged in Salesforce and linked to donors and volunteers.
- Events are created in Salesforce with capacity, location, and date, and linked to volunteers via a junction object.
- Automation rules ensure approvals, notifications, and reminders are triggered.
- Reports and dashboards provide real-time visibility into donations and volunteer activities.

Industry-specific Use Case Analysis

Non-profit organizations, like Helping Hands Foundation, face challenges in managing large amounts of donor and volunteer data. Common industry challenges include donor retention, transparent reporting, and efficient resource allocation. Salesforce Nonprofit Success Pack (NPSP) is a popular industry solution, offering capabilities like donor management, household tracking, and grant management.

For Helping Hands Foundation:

- Donor management ensures donor history and preferences are maintained.
- Volunteer management ensures events have adequate staffing.
- Donation transparency builds trust among stakeholders and encourages repeated contributions.

AppExchange Exploration

AppExchange, Salesforce's marketplace, provides pre-built solutions for nonprofits. Relevant apps include:

- Nonprofit Success Pack (NPSP): Tailored for donor and volunteer management.
- Classy for Salesforce: Helps with fundraising campaigns.
- Volunteers for Salesforce: Streamlines volunteer scheduling and hours tracking.
- FormAssembly: Simplifies data collection via online forms.

Exploring these solutions can accelerate implementation by leveraging best practices and reducing custom development efforts.

🛰 Phase 2 - Org Setup & Configuration

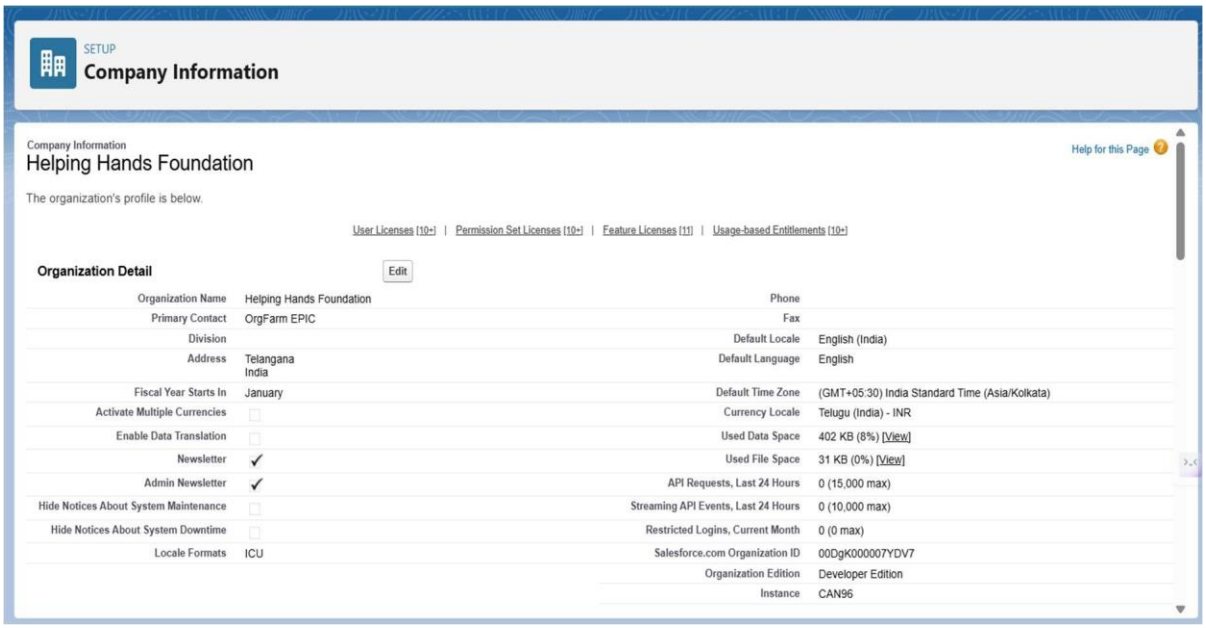
This phase covers the complete setup and configuration of the proper environment setup, security, data modeling readiness, and prepares the organization for future development, automation, and deployment.

Step 1: Salesforce Editions

We are using Salesforce Developer Edition as our working environment. This edition provides the necessary licenses, features, and tools to build and test our Helping Hands Foundation project without production costs.

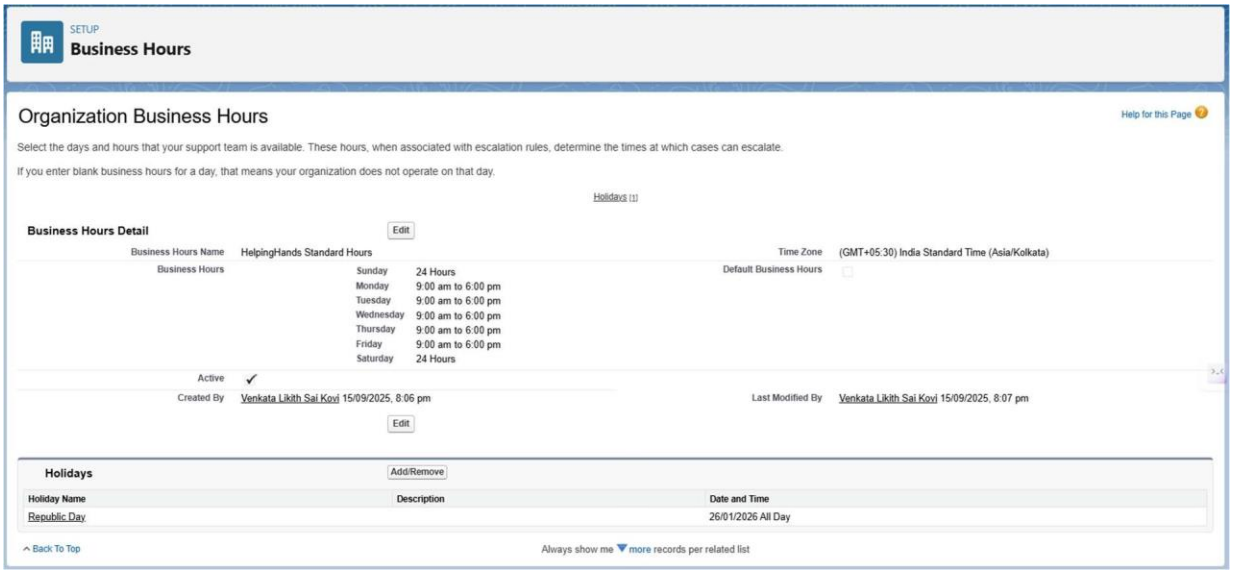
Step 2: Company Profile Setup

Configured Company Profile with organization details such as Company Name (Helping Hands Foundation), Locale, Default Language, Time Zone, and Currency to ensure consistency across the org.



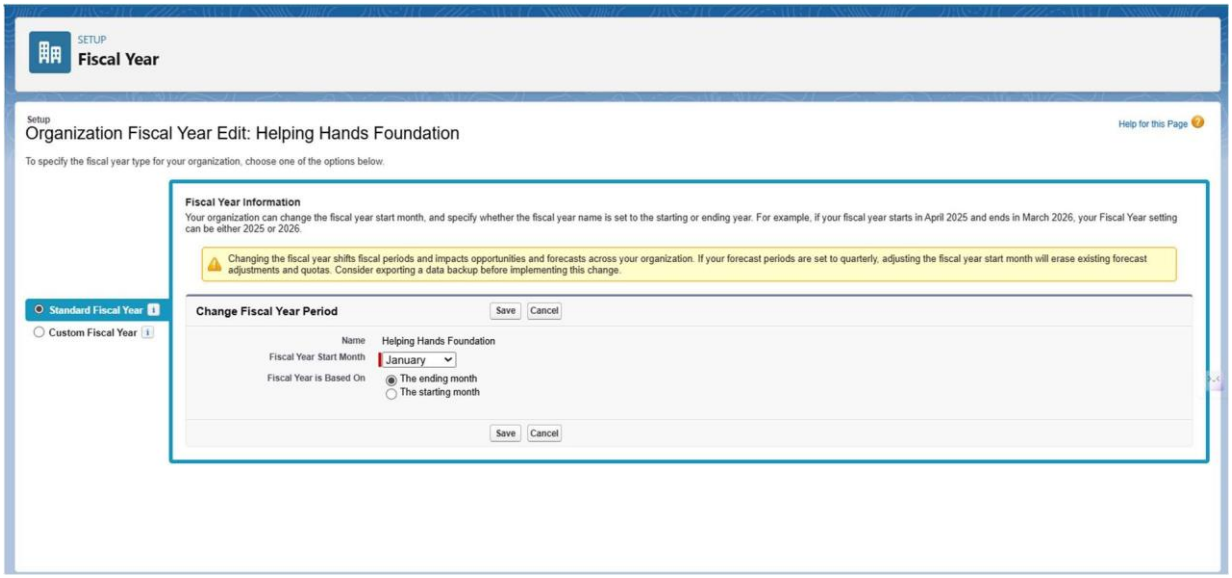
Step 3: Business Hours & Holidays

Defined standard business hours (Mon-Fri, 9 AM – 6 PM) and added holidays (Independence Day, Republic Day, New Year, etc.) to support service-level agreements and volunteer scheduling.



Step 4: Fiscal Year Settings

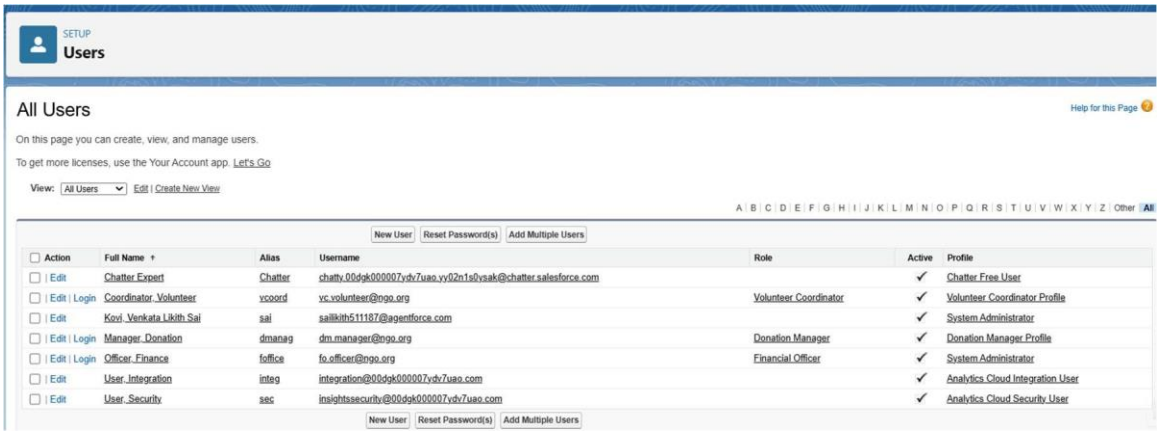
Enabled a Standard Fiscal Year setup aligned with the calendar year (Jan – Dec). This ensures proper reporting of donations, expenses, and volunteer funding cycles.



Step 5: User Setup & Licenses

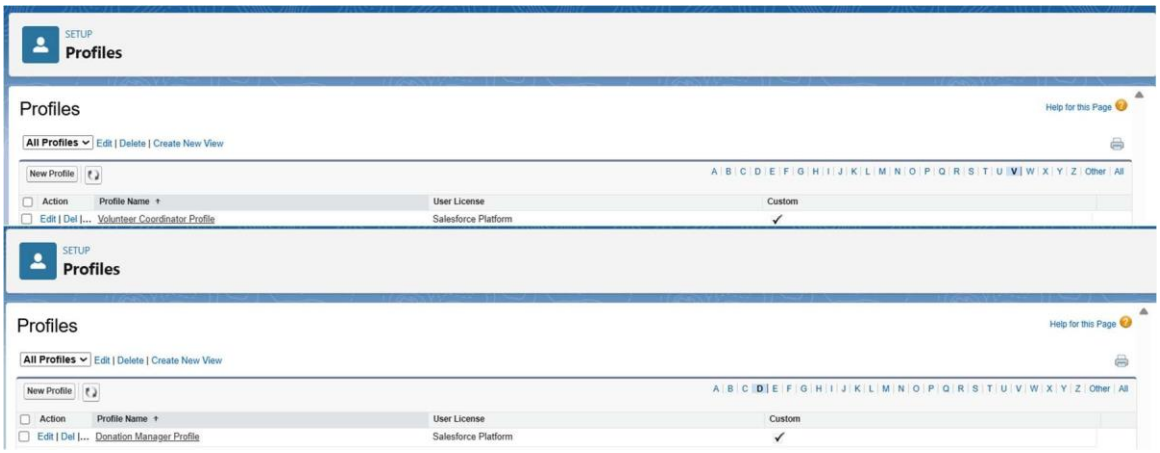
Created project-specific users and assigned appropriate licenses and profiles:

- CEO: System Administrator License
- CFO: Salesforce Platform License
- Financial Officer: System Administrator License
- Donation Manager: Salesforce Platform License
- Volunteer Coordinator: Salesforce Platform License



Step 6: Profiles

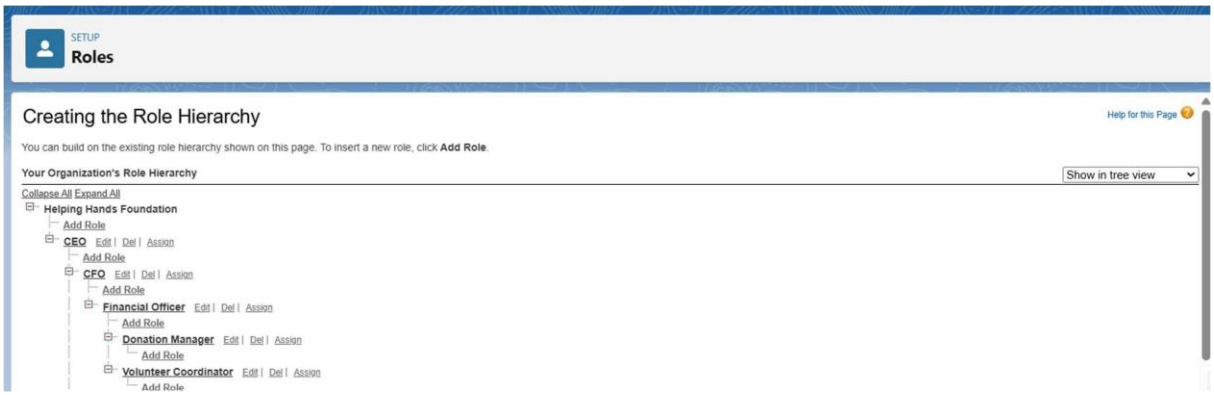
Profiles define baseline access. Created custom profiles for Donation Manager and Volunteer Coordinator with object-level and field-level access tailored to their roles.



Step 7: Roles

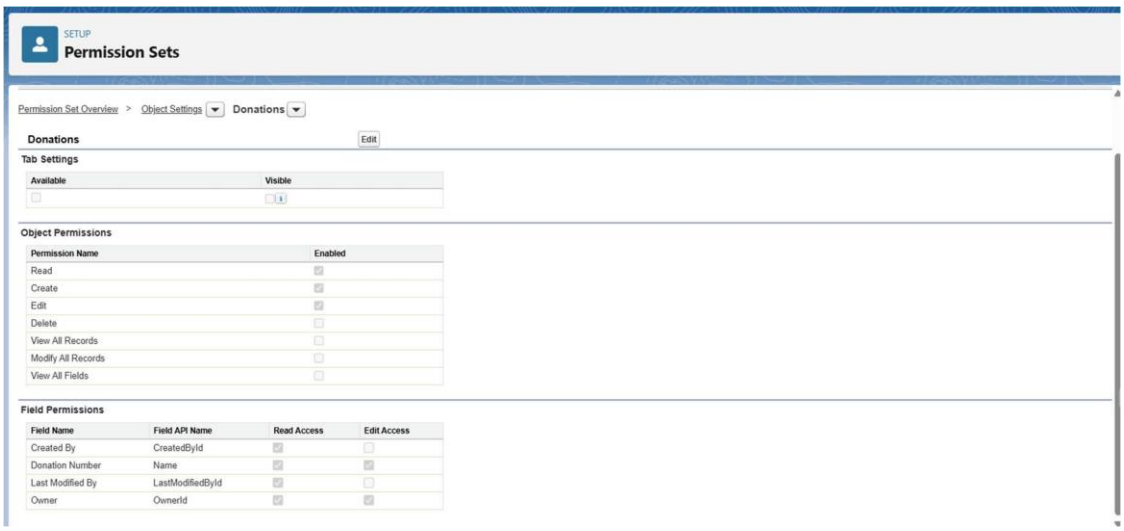
Set up the role hierarchy to mirror the organization:

- CEO
  ↳ CFO
    ↳ Financial Officer
      ↳ Donation Manager
        ↳ Volunteer Coordinator



Step 8: Permission Sets

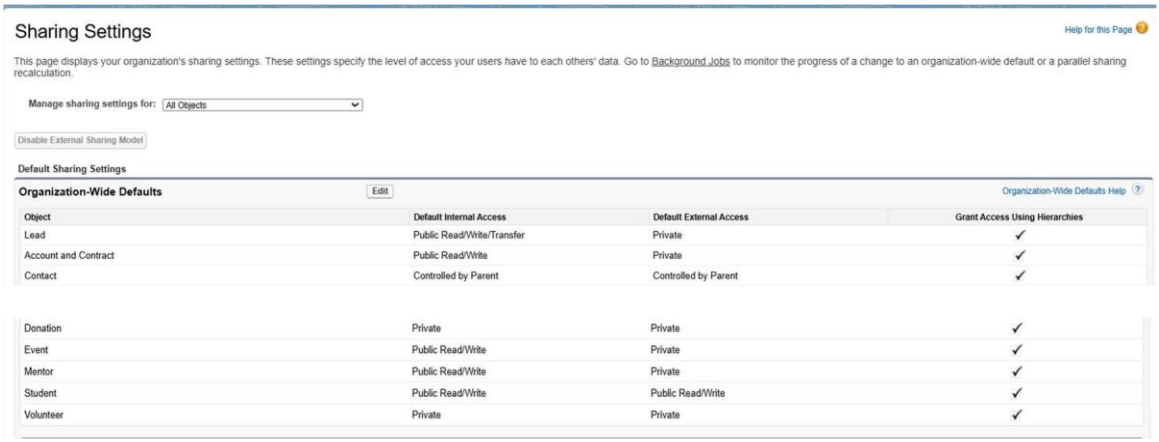
Created additional permission sets for users requiring extra access beyond their profiles, e.g., 'Edit Access and Read/Write/Edit Access' for generating advanced reports.



Step 9: Organization-Wide Defaults (OWD)

Configured OWD:

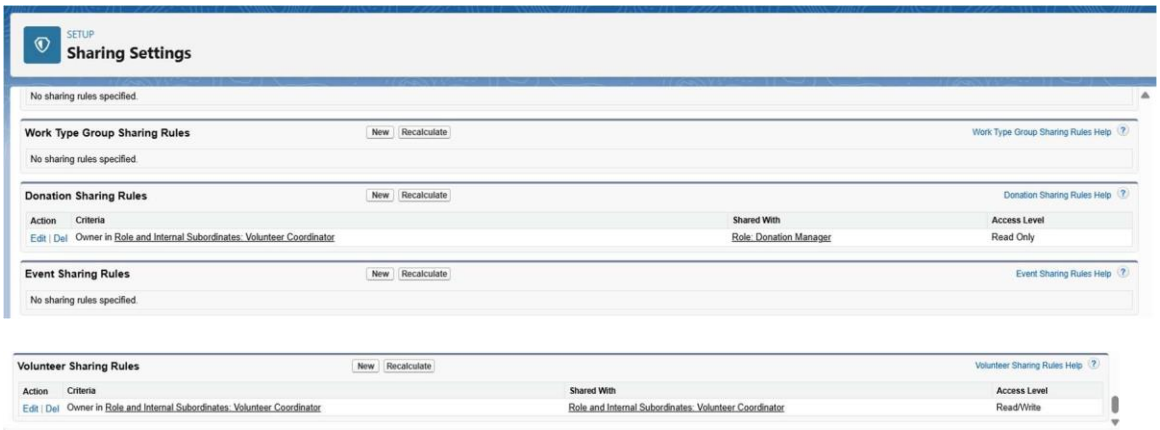
- Donations: Private (only owner & above roles can see)
- Volunteers: Private (restricted by default)
- Accounts/Contacts: Controlled by Parent
- Other standard objects: Default Salesforce settings



Step 10: Sharing Rules

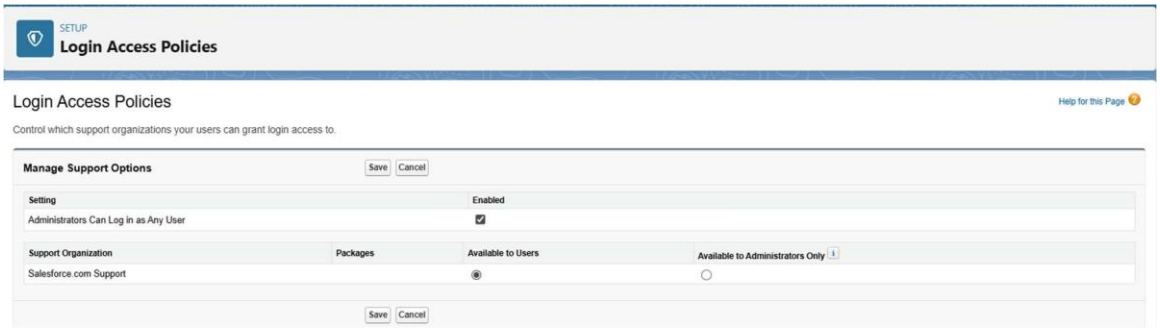
Created Sharing Rules to support collaboration:

- Donation Sharing Rule: Records owned by Volunteer Coordinators are shared with Donation Managers (Read-Only).
- Volunteer Sharing Rule: Volunteer records owned by Volunteer Coordinators are shared with all Volunteer Coordinators (Read/Write).



Step 11: Login Access Policies

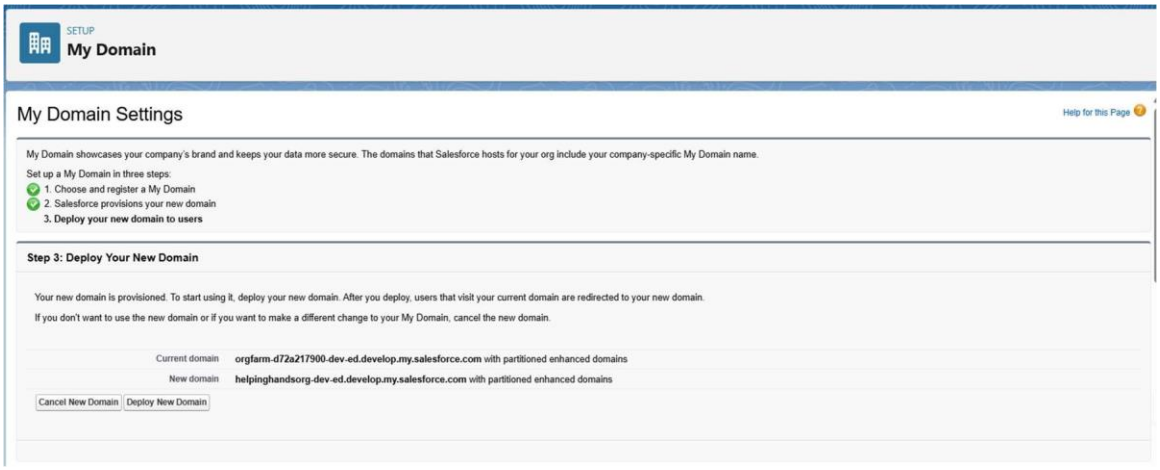
Enabled administrators to log in as any user for troubleshooting. This allows quick issue resolution and ensures smooth testing during the project.



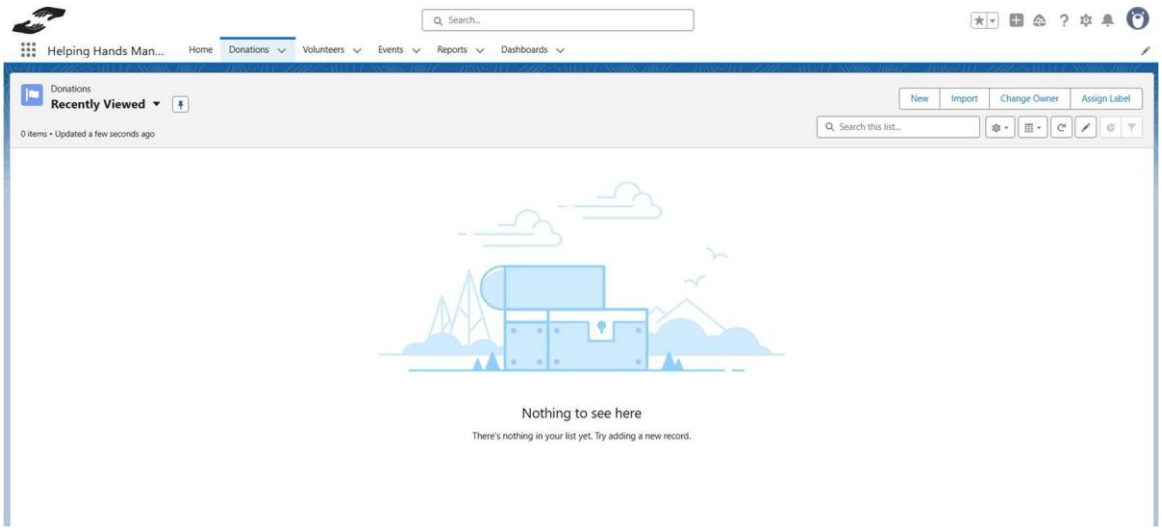
Step 12: Development Org Setup

Confirmed the Developer Org setup and enabled

My Domain :(**helpinghandsorg-dev-ed.develop.my.salesforce.com**).



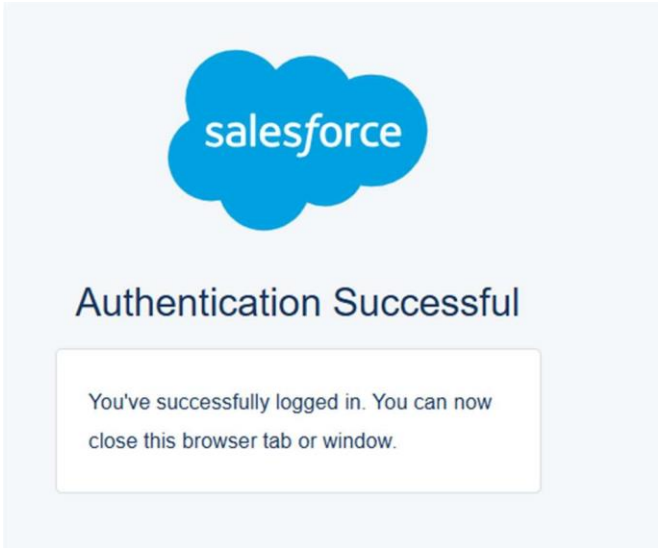
Created a dedicated app 'Helping Hands Management' for testing all project configurations and customizations.

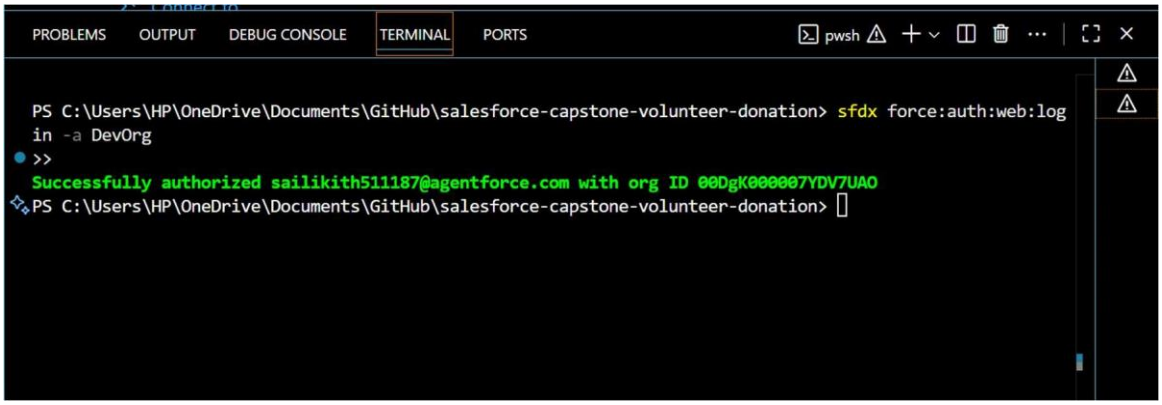


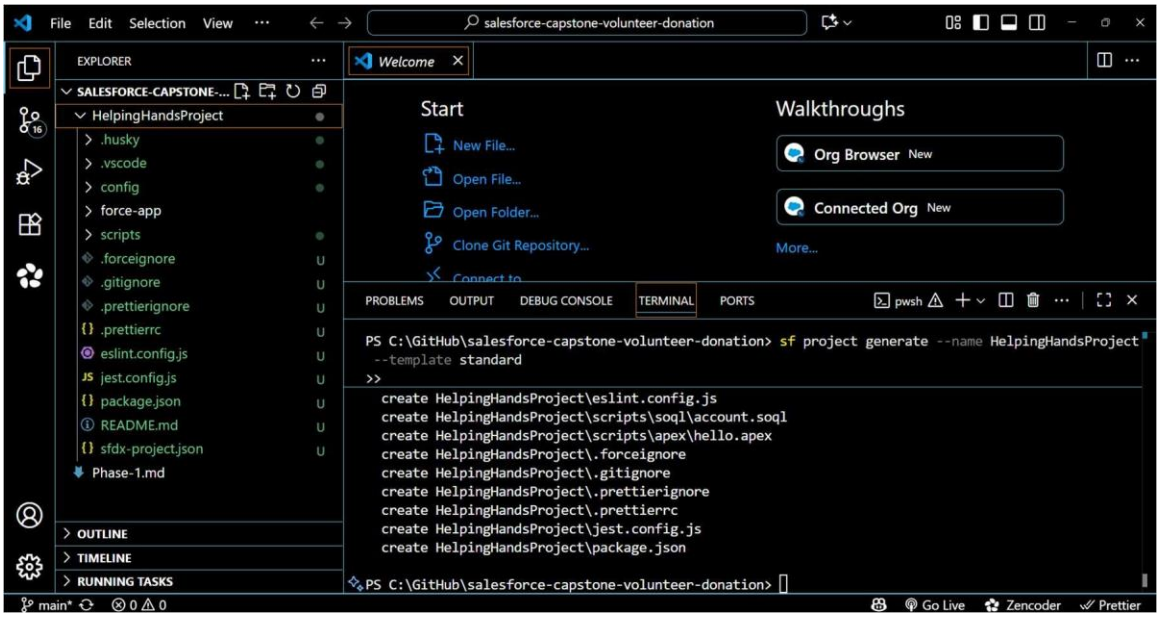
Step 13: Deployment Basics

Deployment options:

- Change Sets: For Admin-driven migration (not available in Developer Edition).
- Salesforce DX (SFDX): Setup VS Code with Salesforce CLI, connect Dev Org, retrieve and deploy metadata to another org.







For this project, deployment will be demonstrated using SFDX and a secondary Developer Org as Production simulation.

🛰 Phase 3 - Data Modeling & Relationships

---

1. Purpose & Objectives

Purpose: Develop the data model necessary to manage Volunteers, Events, Donations, Volunteer assignments, and tasks, enabling the Helping Hands Foundation to track volunteer participation, donations, and event activities.

Objectives:

- Create and configure custom objects and fields
- Establish relationships (lookup & master-detail, junction object)
- Configure Page Layouts, Compact Layouts, and Related Lists
- Verify model in Schema Builder
- Create sample data and test basic SOQL queries
- Pull metadata into SFDX and commit to Git

2. General Conventions & Notes

- All custom objects use the suffix __c in the API name (e.g., Volunteer__c).
- Use Lookup relationships for flexibility and use Master-Detail when you need roll-up summaries and ownership inheritance.
- System fields like CreatedById, LastModifiedById, OwnerId are created automatically — you don't need to add them.

3. Full Object & Field Definitions

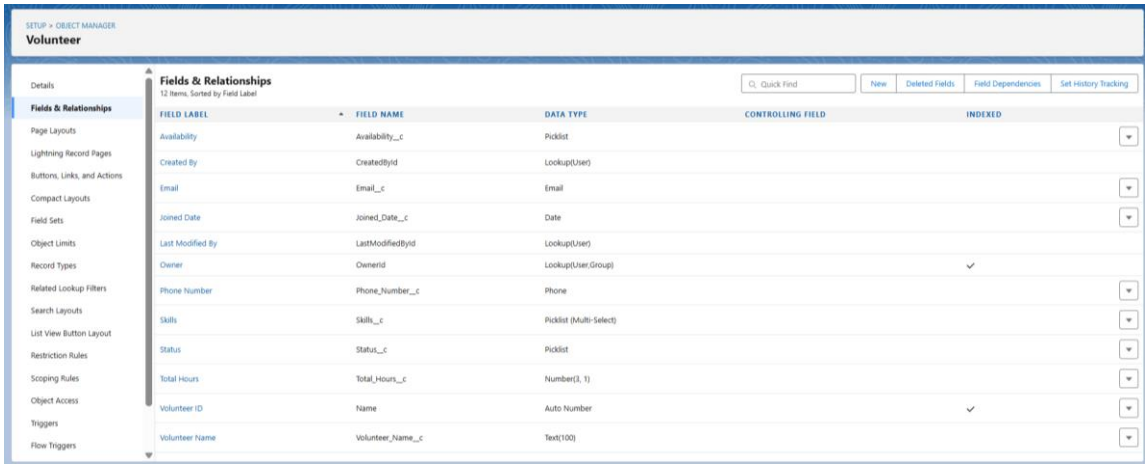
3.1 Volunteer (Volunteer__c)

Object settings

- Label: Volunteer
- Plural Label: Volunteers
- Record Name: Volunteer ID (Auto Number)
- Display Format: VOL-{0000}

Fields to create (custom object-volunteer):

Label	API Name	Data Type	Notes
Volunteer Name	Volunteer_Name__c	Text (255)	Full person name
Email	Email__c	Email	--
Phone	Phone__c	Phone	--
Skills	Skills__c	Multi-Select Picklist	e.g., Event Management; Fundraising; Teaching; Medical
Availability	Availability__c	Picklist	Values: Weekdays, Weekends, Evenings, Flexible
Joined Date	Joined_Date__c	Date	--
Total Hours	Total_Hours__c	Number (5,1)	Sum of hours contributed (see roll-up notes)
Status	Status__c	Picklist	Active, Inactive, Onboarding
Linked Account (optional)	Account__c	Lookup(Account)	If volunteers are linked to organizations



3.2 Event (Event__c)

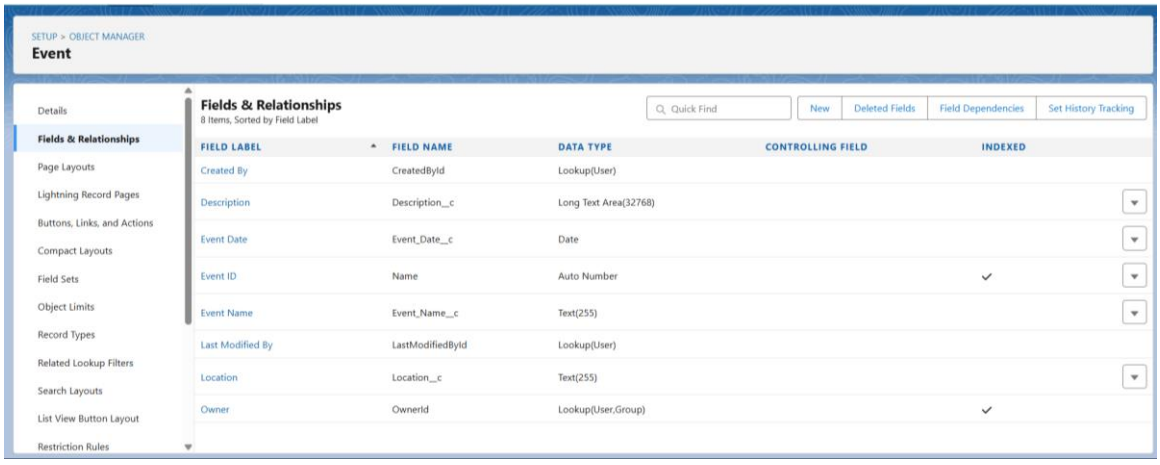
Object settings

- Label: Event
- Plural Label: Events
- Record Name: Event ID (Auto Number)

- Display Format: EVT-{0000}

Fields to create (custom object - Event):

Label	API Name	Data Type	Notes
Event Name	Event_Name__c	Text (255)	Human-readable name
Event Date	Event_Date__c	Date	--
Location	Location__c	Text (255)	--
Description	Description__c	Long Text Area	--
Event Type	Event_Type__c	Picklist	Fundraising, Awareness, Education, Other
Organizer (Volunteer)	Organizer__c	Lookup(Volunteer__c)	Who organizes the event



3.3 Donation (Donation__c)

Object settings

- Label: Donation

- Plural Label: Donations
- Record Name: Donation ID (Auto Number) — DON-{0000}

Fields present in my org:

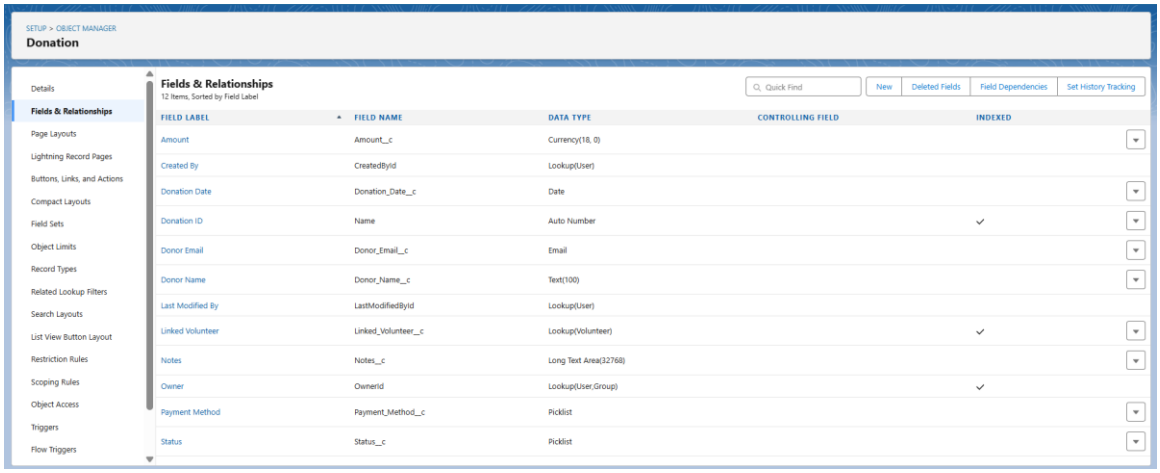
- **Donation ID** — Name (Auto Number).
- **Amount** — Amount__c — Currency.
- **Donation Date** — Donation_Date__c — Date.
- **Donor Name** — Donor_Name__c — Text(100).
- **Donor Email** — Donor_Email__c — Email.
- **Linked Volunteer** — Linked_Volunteer__c — Lookup(Volunteer__c).
- **Payment Method** — Payment_Method__c — Picklist. (Values you used e.g., Cash, Bank Transfer, Credit Card, UPI, Cheque, Other)
- **Status** — Status__c — Picklist. (e.g., Pledged, Received, Refunded)
- **Notes** — Notes__c — Long Text Area.
- **Owner** — standard OwnerId (User/Queue).
- **CreatedBy, LastModifiedBy** — system fields.

Recommended additional field(s):

- **Receipt Sent** — Receipt_Sent__c — Checkbox (default false).
- **Related Event** — Related_Event__c — Lookup(Event__c)

Validation Rule (recommended):

- **Name:** Amount_Must_Be_Positive
- **Formula:** Amount__c <= 0
- **Error message:** Donation Amount must be greater than 0.
- **Error location:** Amount__c field



3.4 Volunteer Event (Junction) (VolunteerEvent__c)

Object settings

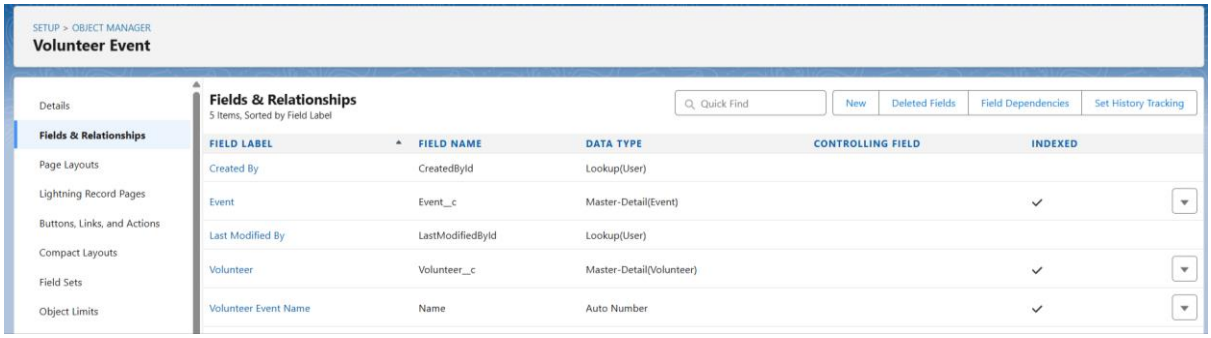
- Label: Volunteer Event
- Plural Label: Volunteer Events
- Record Name: Volunteer Event ID (Auto Number) — VE-{0000}

Relationships (required)

- Master-Detail → Volunteer (Volunteer__c) — Volunteer__c (Master-Detail)
- Master-Detail → Event (Event__c) — Event__c (Master-Detail)

Optional fields on the junction:

- Role — Role__c — Picklist (Helper, Organizer, Coordinator)
- Hours Contributed — Hours_Contributed__c — Number (5,1)
- Feedback — Feedback__c — Long Text Area



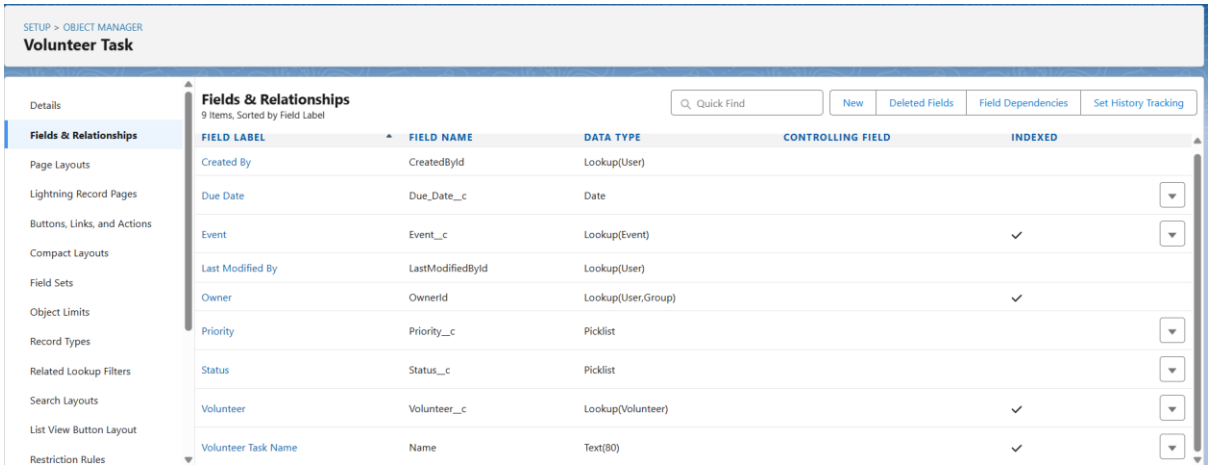
3.5 Volunteer Task (Volunteer_Task__c) — custom task object

Object settings

- Label: Volunteer Task
- Plural Label: Volunteer Tasks
- Record Name: Task ID (Auto Number) or Task Name (Text) – choose Task Name (Text) if you want descriptive titles.

Fields:

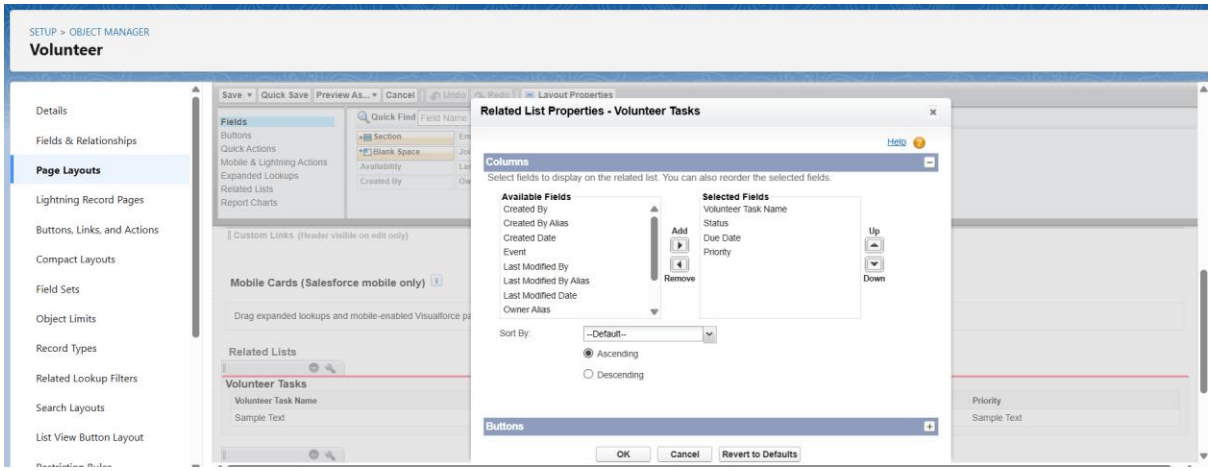
- Task Name — Name — Text (if used as Name)
- Assigned Volunteer — Assigned_Volunteer__c — Lookup(Volunteer__c)
- Related Event — Related_Event__c — Lookup(Event__c)
- Status — Status__c — Picklist (Not Started, In Progress, Completed)
- Priority — Priority__c — Picklist (High, Medium, Low)
- Due Date — Due_Date__c — Date
- Notes — Notes__c — Long Text Area



4. Page Layouts

Configure Page Layouts & Related Lists

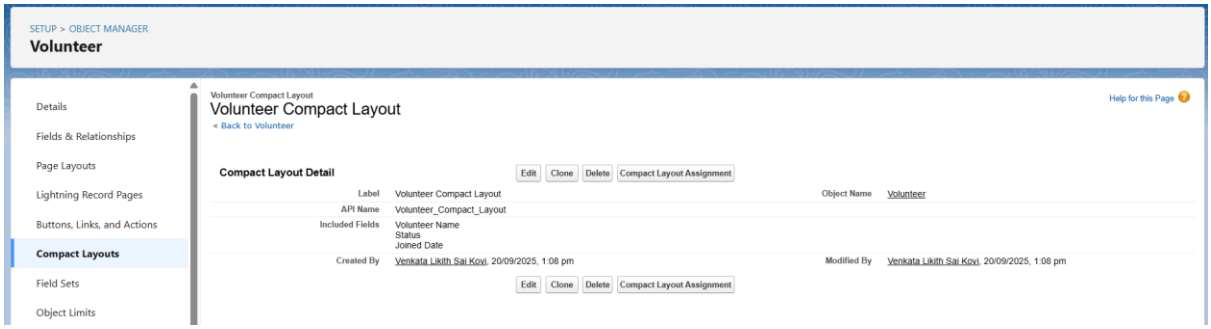
1. Object Manager → Volunteer → **Page Layouts** → Edit default layout.
2. Drag key fields at the top (Volunteer Name, Email, Phone, Status, Skills), and add related lists: Volunteer Events, Volunteer Tasks, Donations (if Donations link to Volunteer).
3. Save.
4. Repeat for Event and Donation (add related lists and important fields).



5. Compact Layouts

Configure Compact Layouts (Mobile / Highlights Panel)

1. Object Manager → Volunteer → **Compact Layouts** → New → choose key fields (Volunteer Name, Status, Joined Date).
2. Save → Click **Compact Layout Assignment** → Set as Primary.



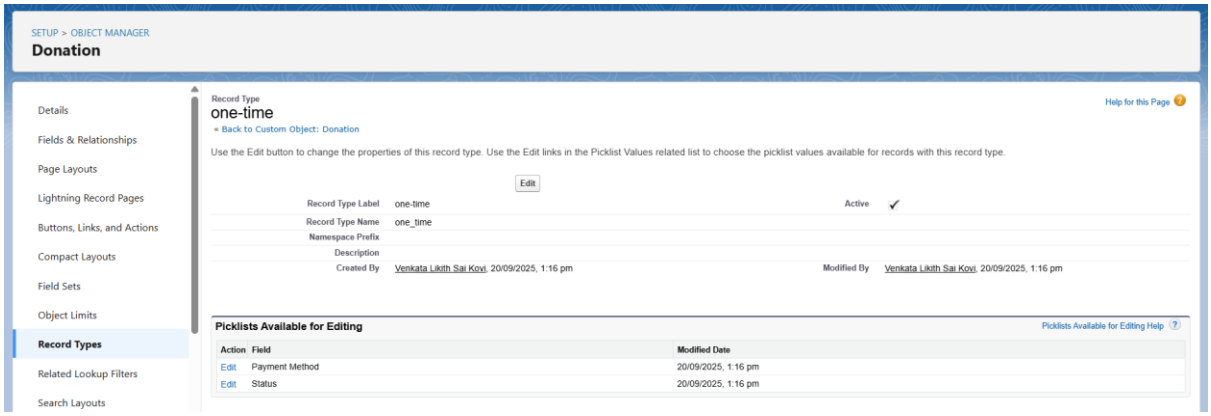
6. Field Level Security & Profile Access

1. For each field: Object Manager → {Object} → Fields & Relationships → click field → **Set Field-Level Security** → Select profiles that should have visibility/edit access → Save.
2. For each object: Object Manager → {Object} → **Object Settings** in profile to control CRUD (Create, Read, Edit, Delete). Or edit the cloned profiles created in Phase 2 and set Object Permissions.

7. Record Types & Page Layout Assignments

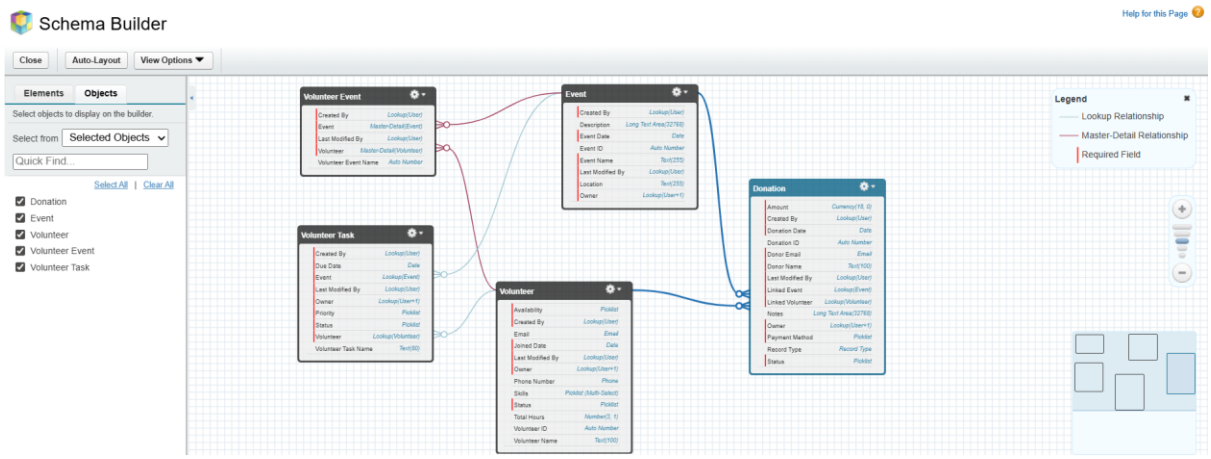
Object Manager → Donation → **Record Types** → New: One-Time, Recurring.

1. Assign different page layouts if you want different fields visible for types.
2. Save and assign to profiles.



8. Schema Builder

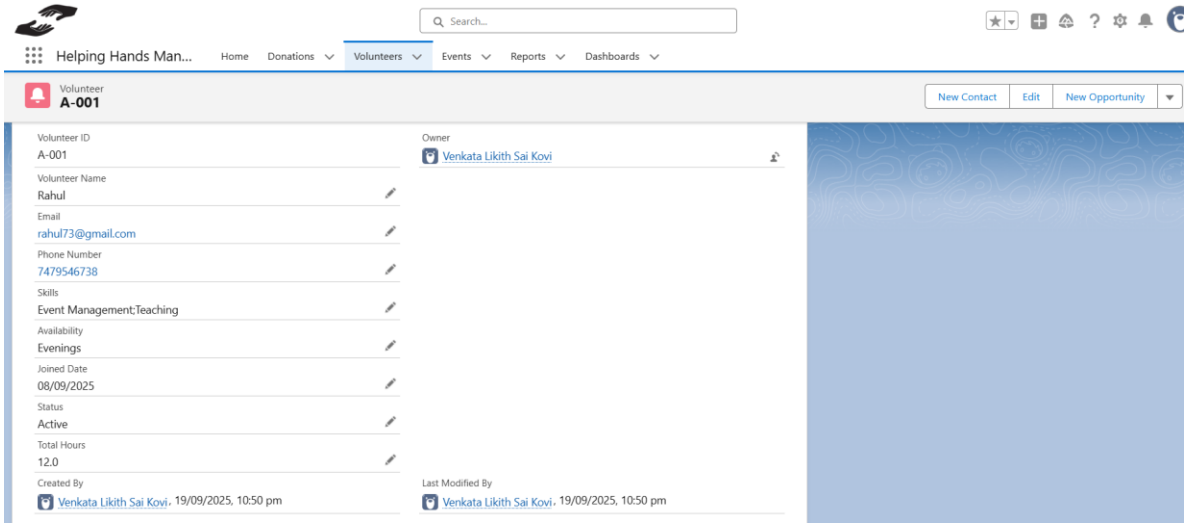
1. Setup → **Schema Builder** → Check the left panel boxes for Volunteer, Event, Donation, Volunteer Event, Volunteer Task.
2. Layout the schema on the canvas and verify relationships (look for lines representing Lookup and Master-Detail).
3. Take a screenshot of the schema diagram for documentation.



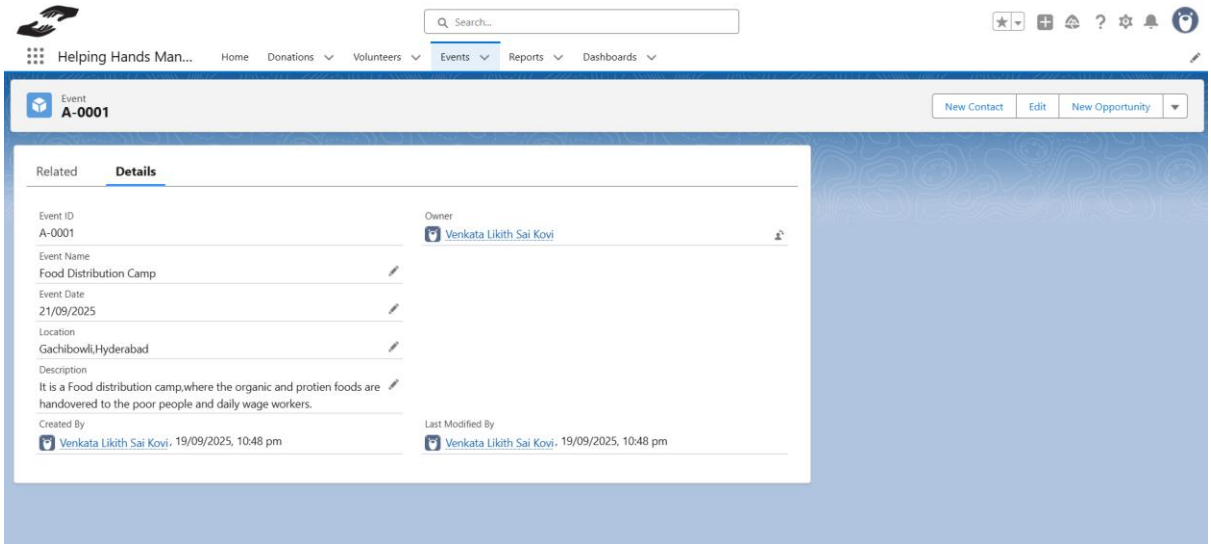
9. Sample Data (Test Records)

Create sample records to validate relationships and reporting:

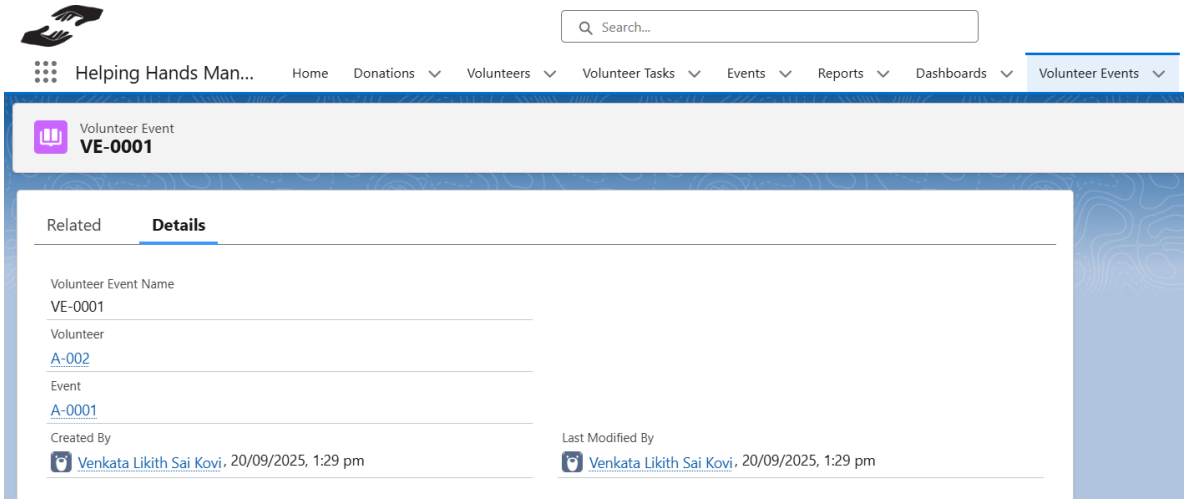
Volunteers:



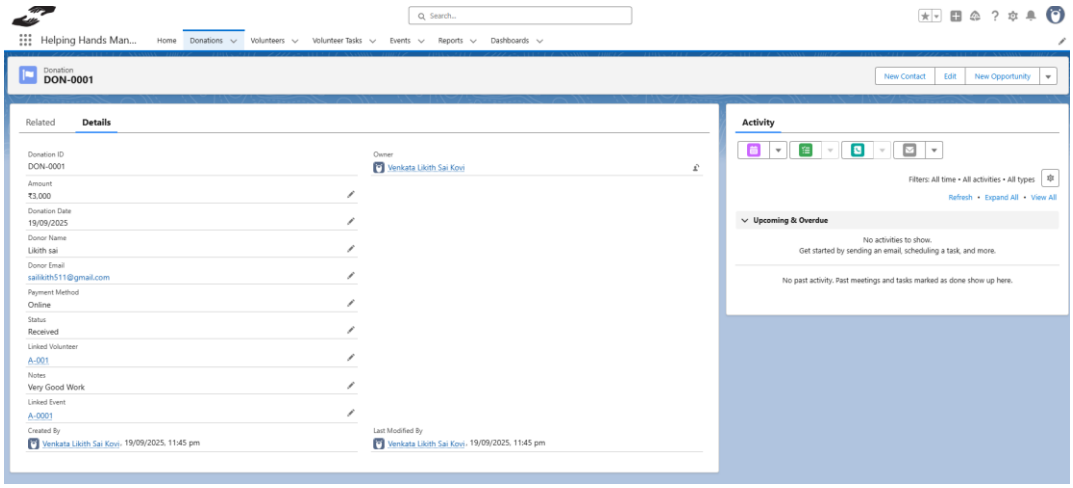
Events:



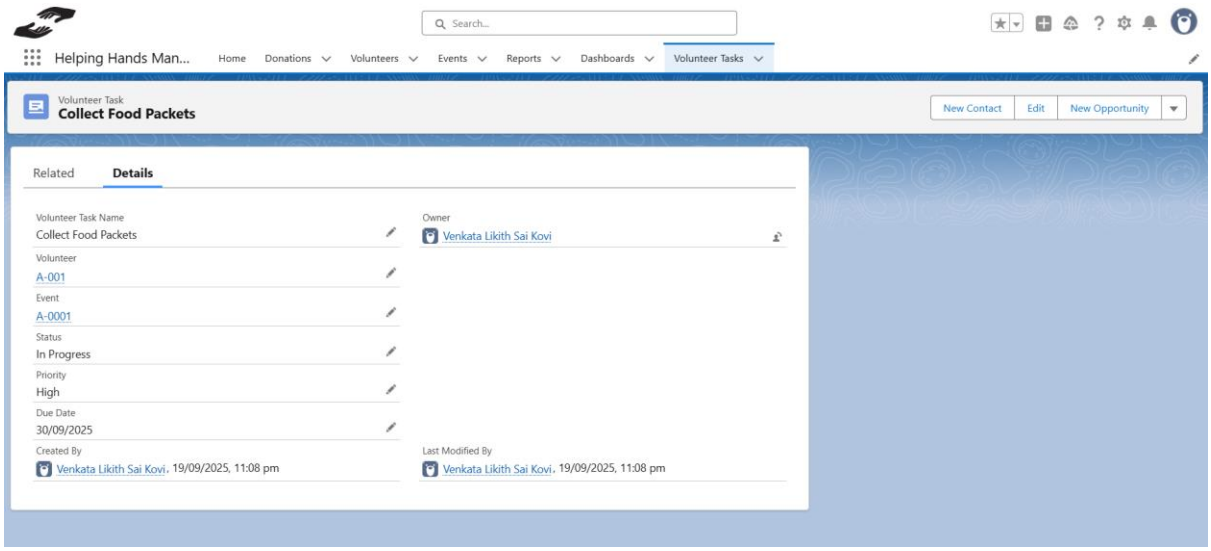
VolunteerEvent (junction):



Donations:



Volunteer Task(Food Donation):



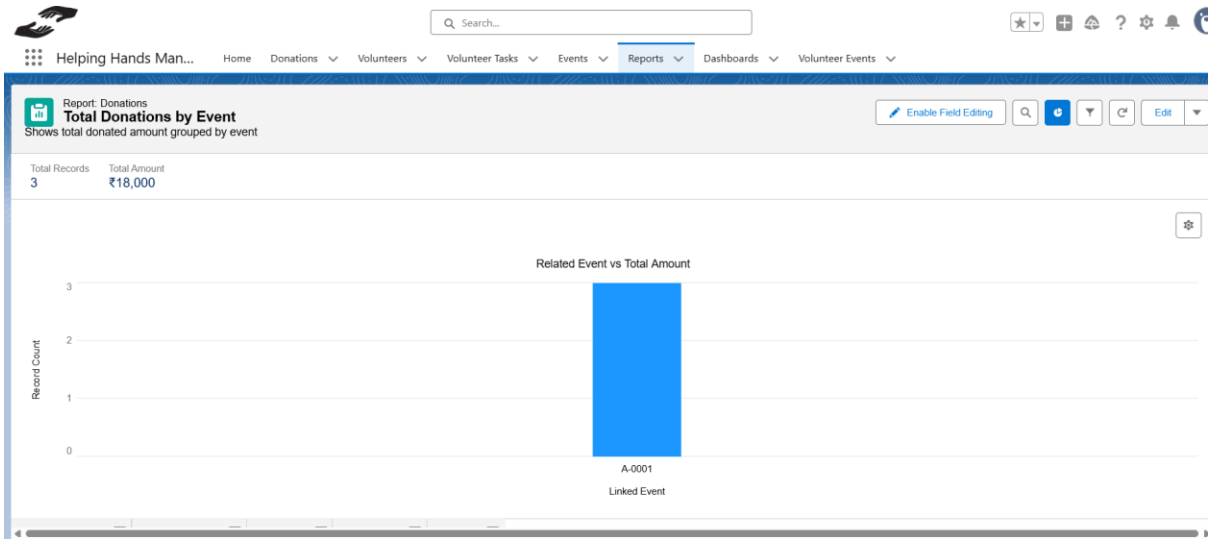
10. Reports & Dashboards

Reports to create:

1. Total Donations by Event

   — Group by Related Event, summarize Sum(Amount)

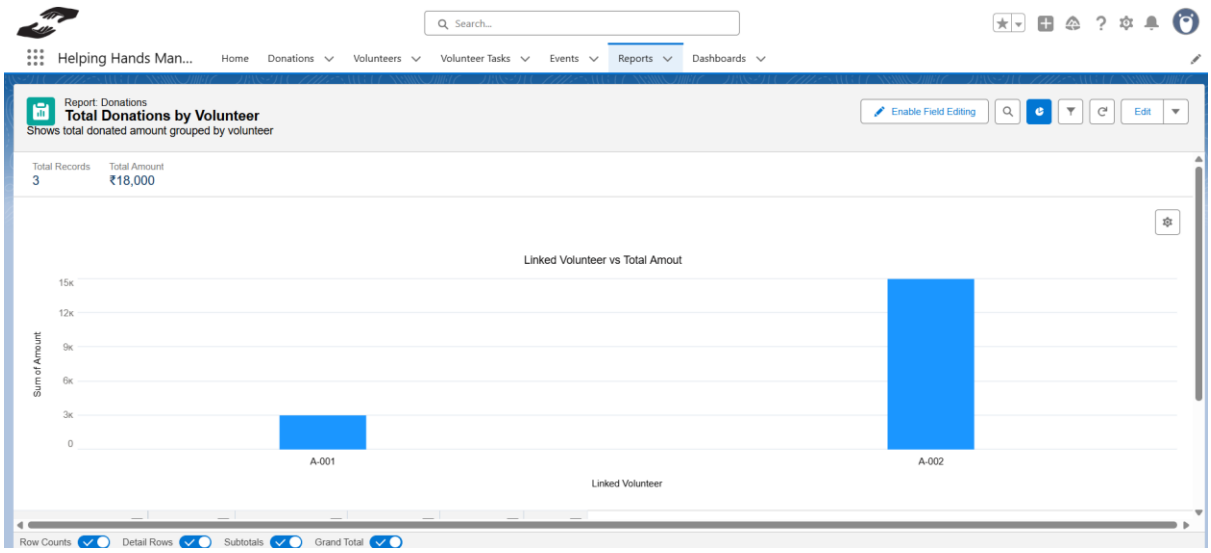
   — add chart (bar).



2. Donations by Volunteer

   — Group by Linked Volunteer

   — Sum Amount.



Dashboard:

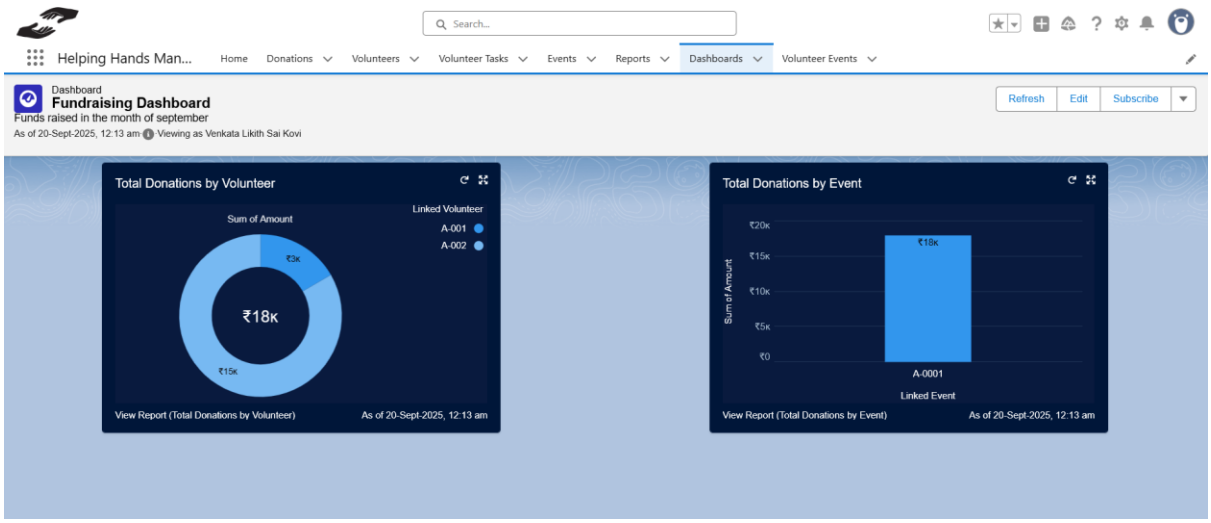
- Dashboards → New Dashboard

→ Name: Fundraising Dashboard

→ Add component: select report Total Donations by Event

→ choose Bar Chart

- Total Donations by Event
- Total Donations by Volunteer → Save.



11. Developer: Retrieve Metadata to SFDX & Version Control

Authenticate to Dev Org

```
sfdx force:auth:web:login -a DevOrg
```

Retrieve specific custom objects

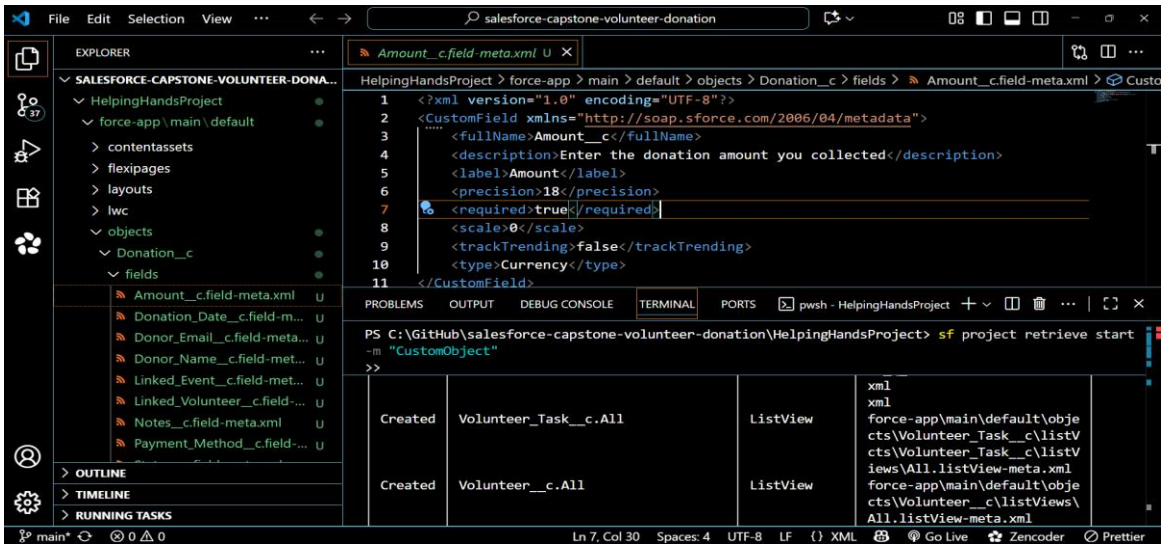
```
sfdx force:source:retrieve -m CustomObject:Volunteer__c,CustomObject:Event__c,CustomObject:Donation__c,CustomObject:VolunteerEvent__c,CustomObject:VolunteerTask__c -u DevOrg
```

Commit to Git

```
git add .
```

```
git commit -m "Phase 3: Add custom objects Volunteer, Event, Donation, VolunteerEvent, VolunteerTask"
```

```
git push origin main
```



🛰 Phase 4 - Automation, Approval Processes, and Reporting

Summary of Phase 4 Deliverables

Phase 4 implements automation (Flows), validation rules, an approval process, scheduled summary flows, email alerts and reporting dashboards. The key deliverables are:
- Validation Rules (Donation amount > 0)
- Record-triggered Flows (Orientation task for new volunteers; Large-donation email notifications)
- Approval Process for large donations (threshold ≥ ₹100,000)
- Scheduled Flow that summarises monthly donations into Donation_Summary__c
- Reports & Dashboards (Monthly Donation Summary Report, Fundraising Dashboard)
- Debug logs, test cases, and fixes for duplicate month records.

This doc reflects what was implemented in your Dev Org and includes exact resource names, labels, variables, and formulas you used.

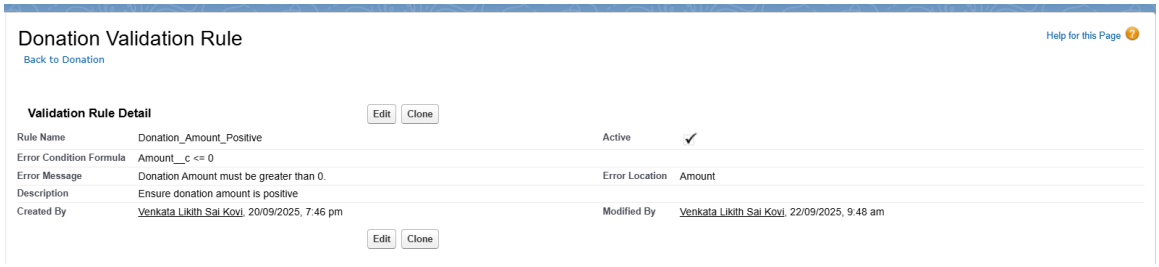
Step 1: Validation Rule — Donation Amount Must Be Greater Than Zero

Purpose: Prevent invalid donation records where the amount is zero or negative. This preserves reporting accuracy.

Steps:-

1. Click the Gear icon → Setup.
2. In Quick Find type: Object Manager → click Object Manager.
3. Find and click: Donation (Donation__c).
4. In left menu click: Validation Rules → New.
5. Fill fields exactly as below:
   - Rule Name: Validate_Donation_Amount
   - Description: Ensure donation amount is positive
   - Error Condition Formula:
     Amount__c <= 0
   - Error Message: Donation amount must be greater than 0.
   - Error Location: Field → Amount

6. Click Save, then Activate (if not active automatically).

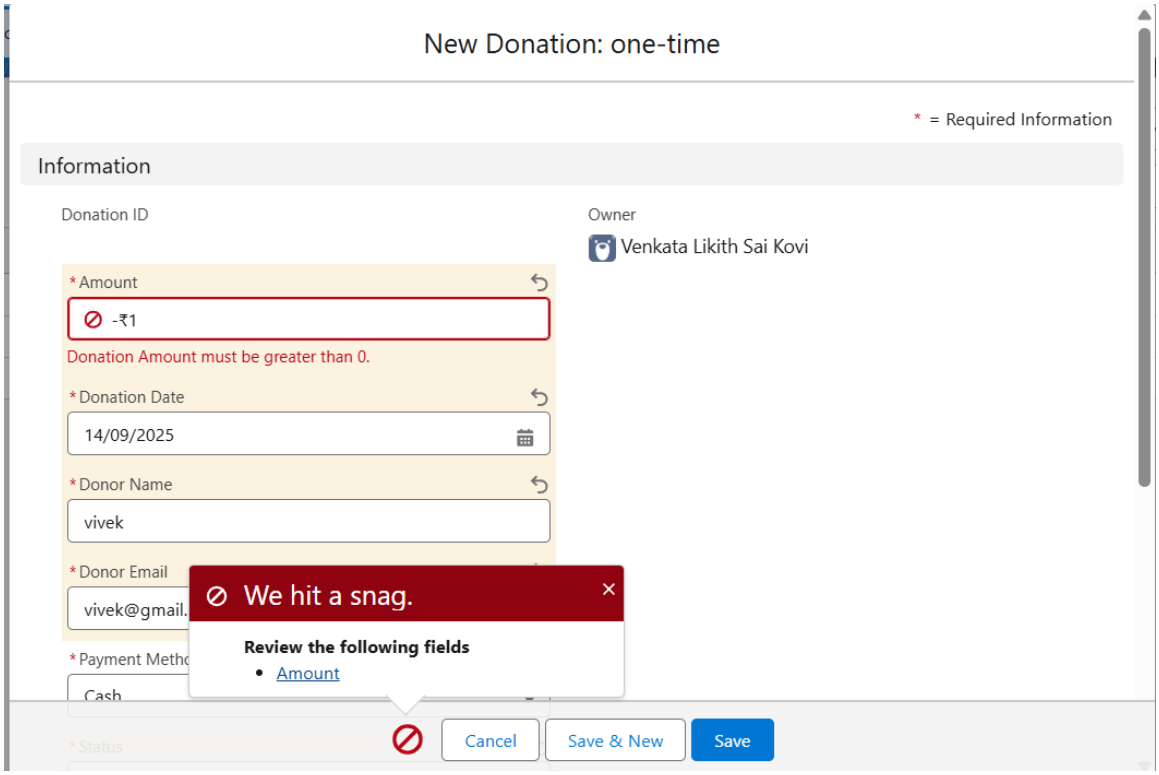


Test case :

• Create a Donation record with Amount = -1

→ Attempt to save

→ Expected Result: Validation error displayed near the Amount field.

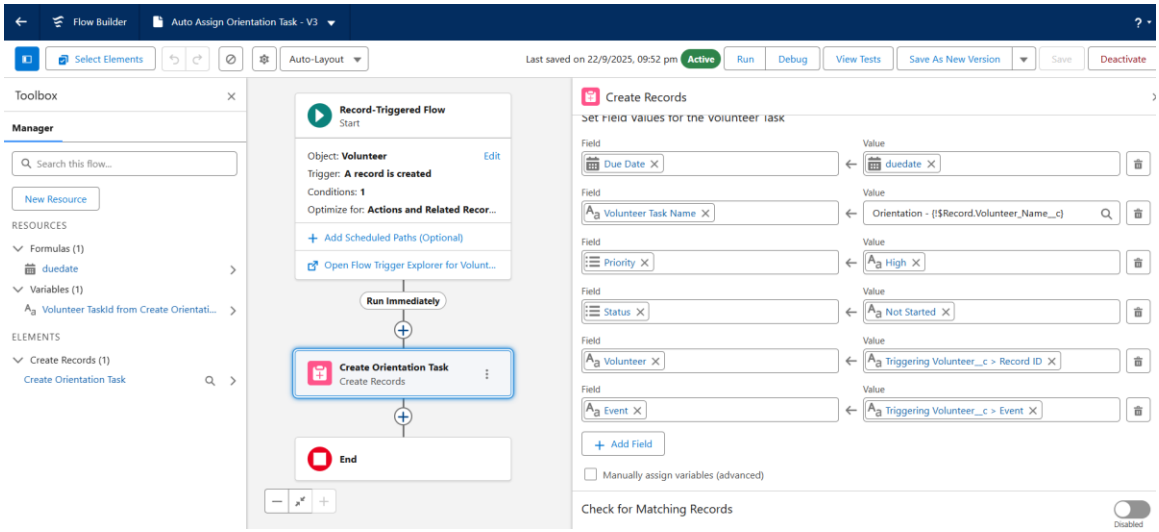


Step 2: Record-Triggered Flow — Auto-assign 'Orientation' Volunteer Task

Purpose: When a new Volunteer__c record is created, automatically create a VolunteerTask__c record for orientation so onboarding is tracked and consistent.

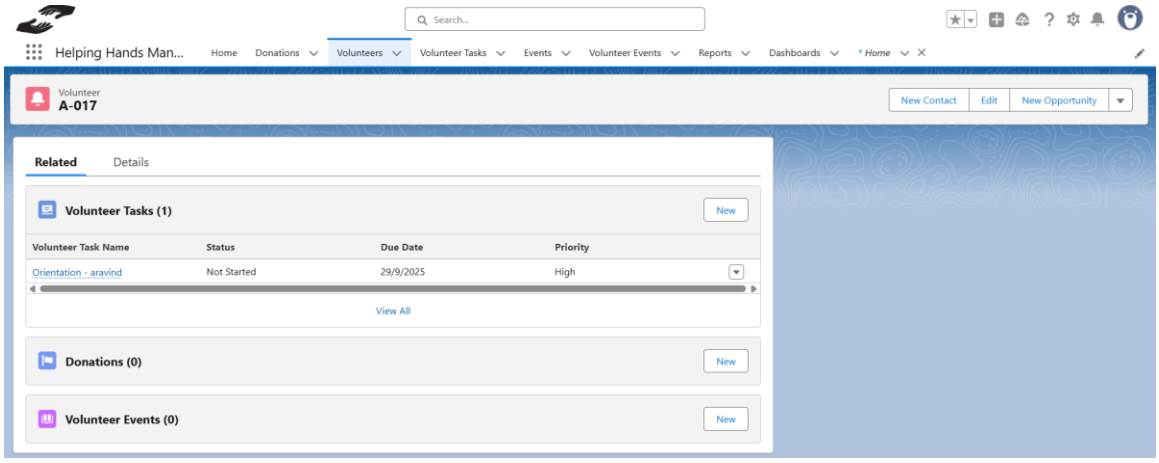
Implementation details (exact):

1. Setup → Quick Find → Flows → Flows → New Flow.
2. Choose: Record-Triggered Flow → Create.
3. Configure Start:
   - Object: Volunteer (Volunteer__c)
   - Trigger: A record is created
   - Condition Requirements: None (or Status__c = 'Onboarding' if you only want to run for that status)
   - Optimize the Flow for: Actions and Related Records
4. Click Done.
5. Add a Create Records element:
   - Label: Create Orientation Task
   - Create One Volunteer Task (VolunteerTask__c)
   - Set Field Values:
     Name (Task Name): 'Orientation - ' & {!$Record.Volunteer_Name__c}
     Assigned_Volunteer__c: {!$Record.Id}
     Status__c: Not Started
     Priority__c: High
     Due_Date__c: Use a Date Formula Resource (DueDateFormula) = TODAY() + 7
6. Save Flow Label: Auto_Assign_Orientation_Task. Save and Activate.



Test steps :

1. App Launcher → Helping Hands → Volunteers → New Volunteer.
2. Fill Volunteer_Name__c = 'Test Volunteer Onboard', other fields as needed → Save.
3. Open the volunteer record → scroll to Volunteer Tasks related list → Confirm 'Orientation - Test Volunteer Onboard' exists.

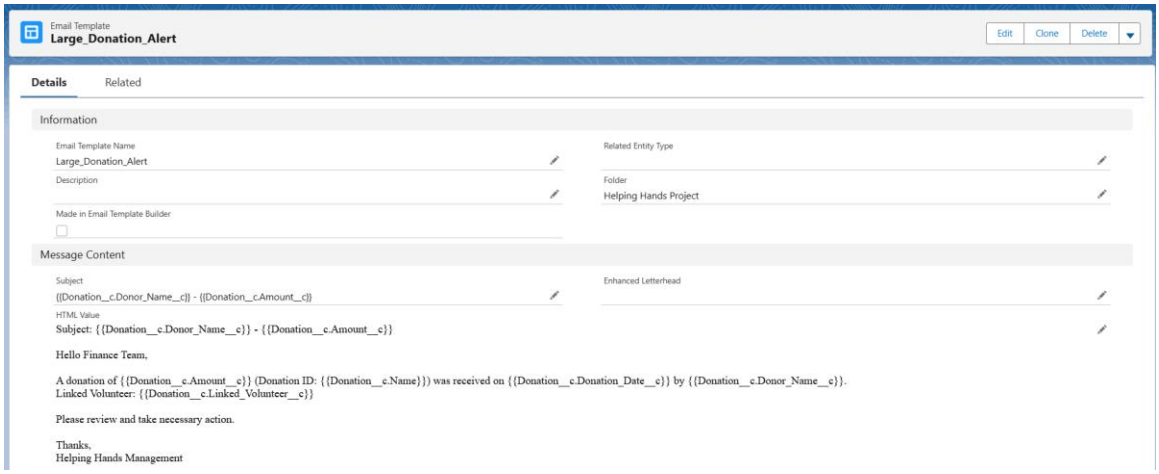


Step 3: Record-Triggered Flow — Email Alert for Large Donations (>= ₹10,000)

Purpose: Notify Finance Officer (or a Role) when any donation equal to or greater than ₹10,000 is created so that high-value gifts receive timely attention.

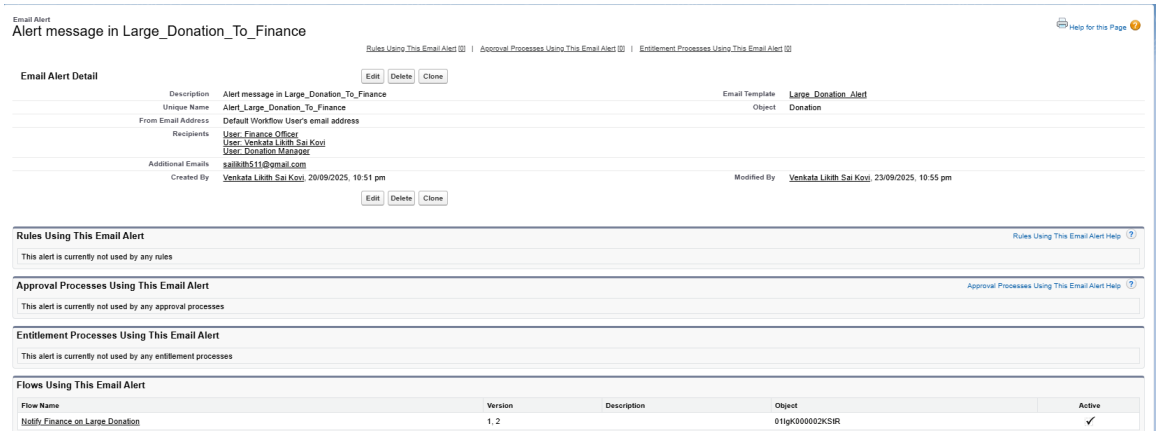
Implementation steps :

A. Create Email Template (Lightning Email Template):
1. Setup → Email Templates → New Email Template.
2. Template Name: Large Donation Alert
3. Subject: Large donation received: {!Donation__c.Donor_Name__c} - {!Donation__c.Amount__c}
4. Body: include merge fields for Donation__c.Donor_Name__c, Donation__c.Amount__c, Donation__c.Donation_Date__c and link to the Donation record.
5. Save in folder: Helping Hands Templates (or Unfiled Public Email Templates).



B. Create Email Alert:

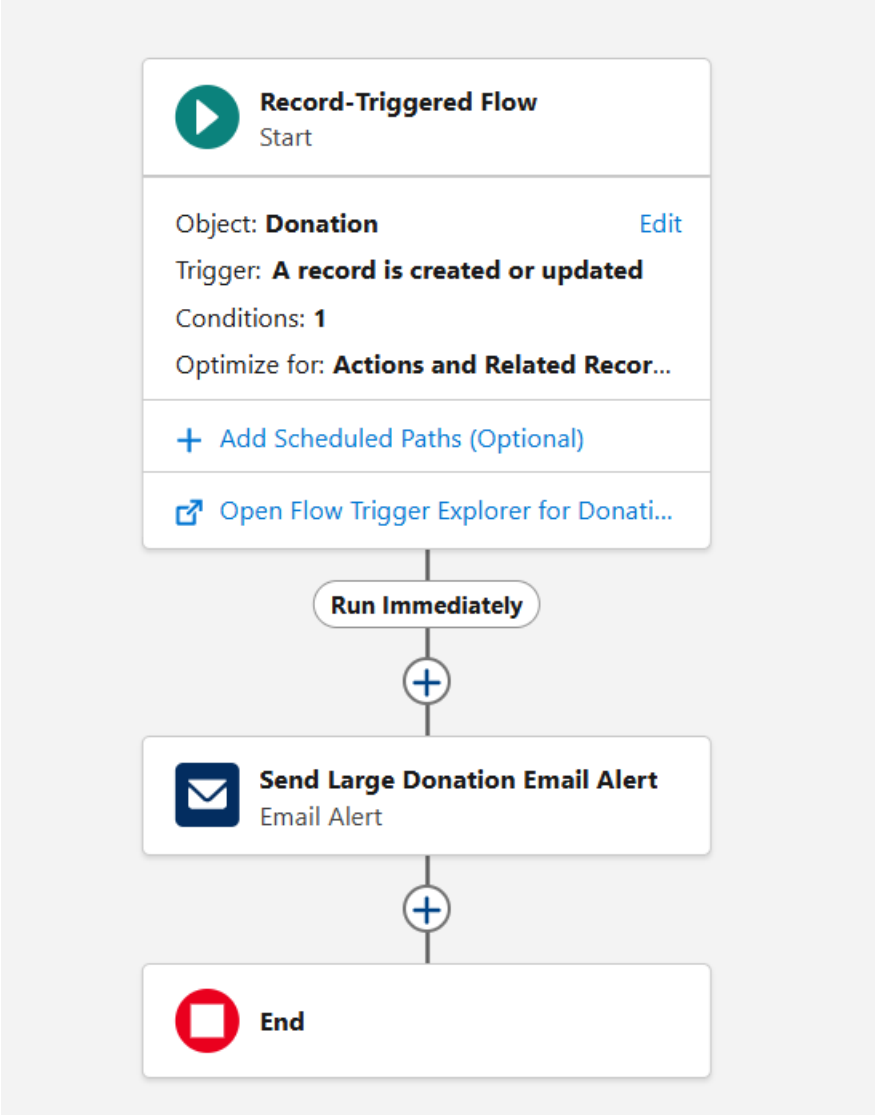
1. Setup → Quick Find → Email Alerts → New Email Alert.
2. Object: Donation
3. Email Template: Large Donation Acknowledgement
4. Recipient Type: Role → Financial Officer (or specific user)
5. Save



C. Record-Triggered Flow:

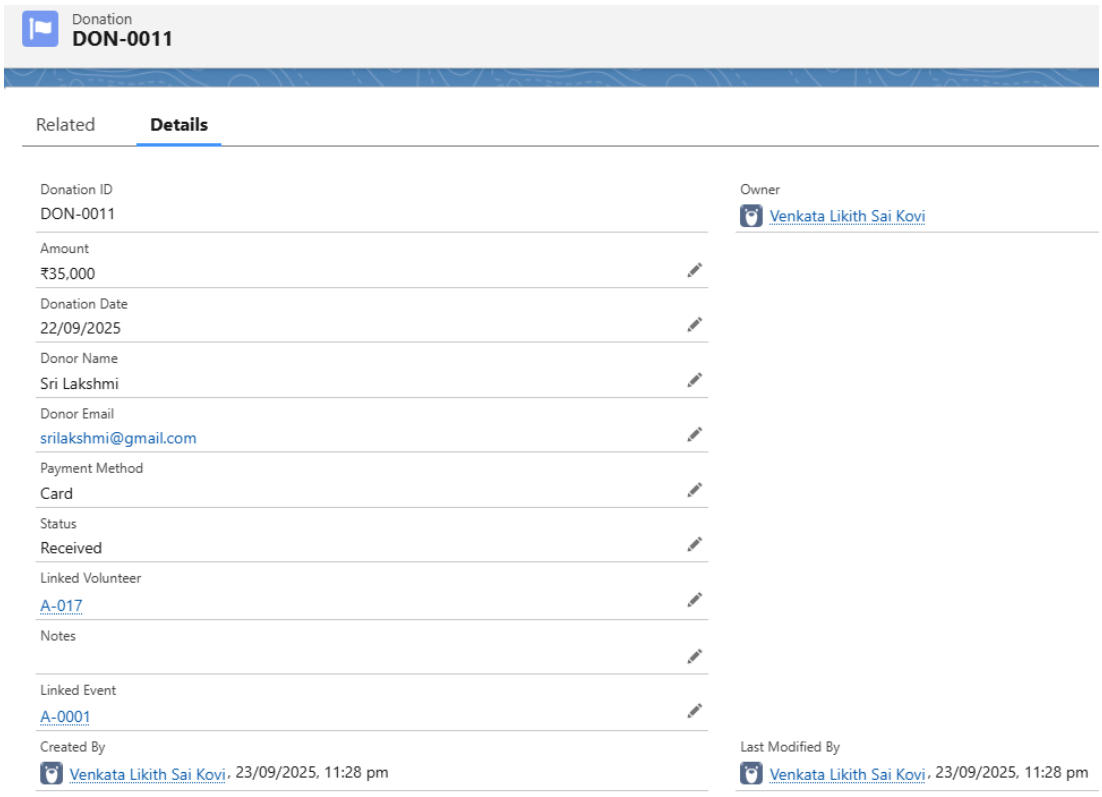
1. Setup → Flows → New Flow → Record-Triggered Flow
2. Object: Donation (Donation__c)
3. Trigger: A record is created or updated
4. Condition Requirements: Amount__c >= 10000
5. Run After Save (so email actions can execute)
6. Add an Action element → select the Email Alert

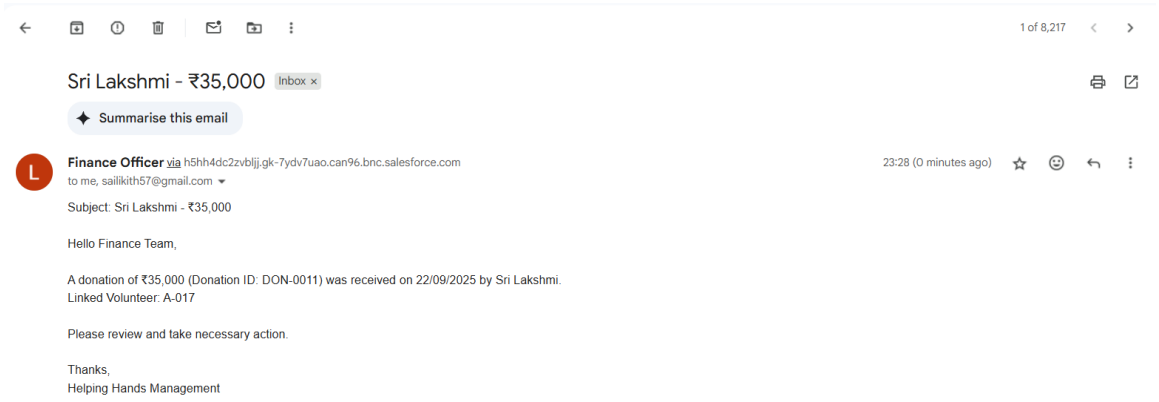
(EA_LargeDonation_NotifyFinance)
7. Save Flow Label: Notify_Finance_on_Large_Donation. Activate.



Test example :

• Create Donation record with Amount = 15000, Donor_Name__c = 'Test Donor Email'.
• Save → Check the Financial Officer's email inbox or the email logs in Salesforce.



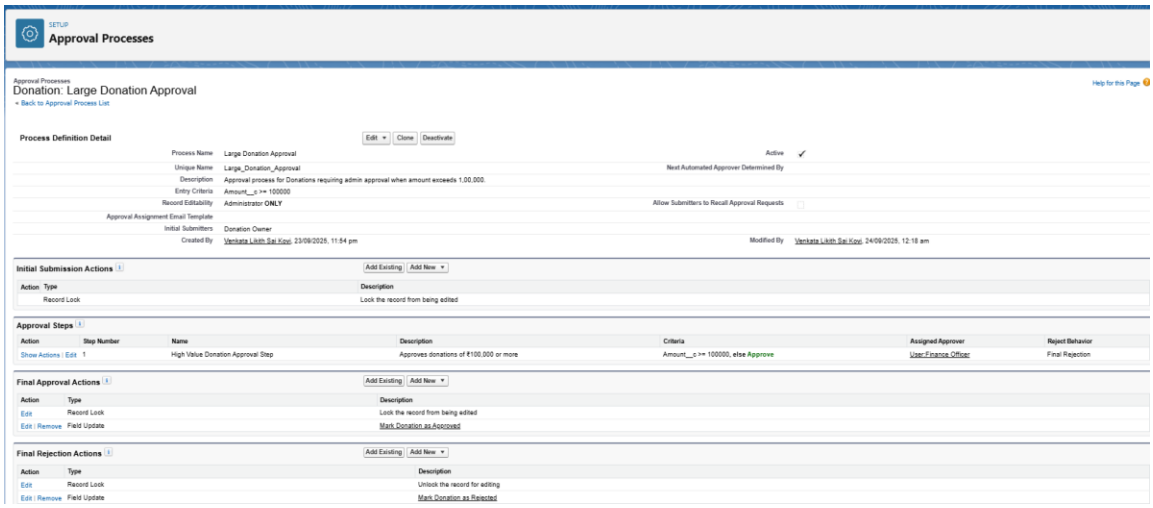


Step 4: Approval Process — Large Donation Approval (>= ₹100,000)

Purpose: Implement a formal approval process for very large donations to ensure financial oversight. You implemented an approval process that routes donations >= ₹1,00,000 to the Finance Officer.

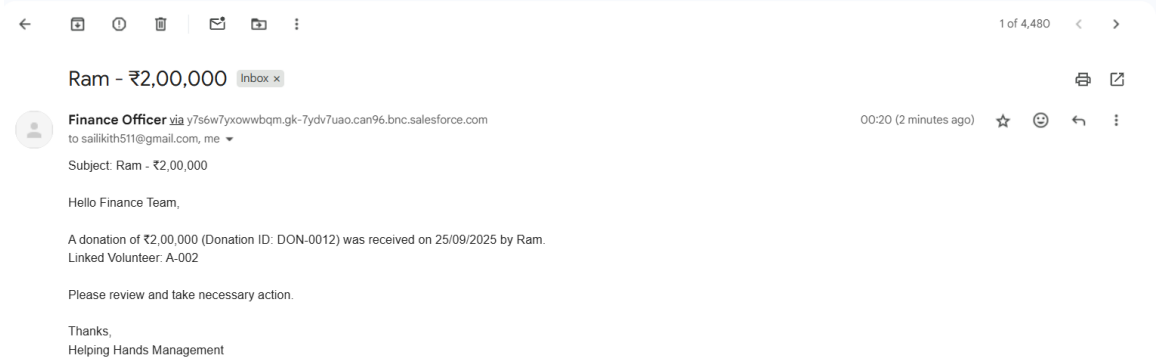
Steps performed (Jump Start Wizard approach):

1. Setup → Quick Find → Approval Processes → Approval Processes.
2. Select Object: Donation.
3. Click Create New Approval Process → Use Jump Start Wizard.
4. Fill: Name = Large Donation Approval, Description as needed.
5. Entry Criteria: Amount__c greater or equal 100000.
6. Select Approver: Automatically assign to approver(s) → User: choose Finance Officer (or CFO). For testing you used System Administrator user.
7. Final Approval Action → Add New → Field Update: set Status__c = 'Approved'
8. Final Rejection Action → Add New → Field Update: set Status__c = 'Rejected'
9. Save → Activate the approval process.



Approval Step (added):

• Created an Approval Step named 'High Value Donation Approval Step'.
 - Step Criteria: Amount__c >= 100000
 - Assigned Approver: System Administrator (for testing), later set to Financial Officer role.



Step 5: Scheduled Flow — Monthly Donation Summary (Detailed)

Purpose: Create a scheduled automation that computes monthly donation totals and stores them in Donation_Summary__c. Because Lightning Scheduled Flow may not show 'Monthly' frequency, we scheduled it Daily and used a Decision to run only on the first of the month.

Flow design steps:-
1. Flow type: Schedule-Triggered Flow (configured as Daily; Decision checks DAY(TODAY()) = 1)

2. Flow Label: Monthly Donation Summary (Daily Check)

3. Manager Resources created:
  - Variable: TotalAmount (Number, scale 2, default 0)
  - Variable: DonationCount (Number, scale 0, default 0)
  - Formula: MonthYearText (Text) = TEXT(MONTH(TODAY())) & "-" & TEXT(YEAR(TODAY()))
  - Formula (IsFirstOfMonth) or Decision: DAY(TODAY()) = 1

4. Schedule: Start Date = (for testing) 24-Sep-2025; Frequency = Daily; Start Time = as selected

5. Flow Elements and exact labels used:
  - Start (Schedule) -> Decision (Check if First Day of Month)
   Decision Outcome: Yes_Run_Summary if DAY(TODAY()) = 1 OR RunNow boolean is True

- Get Records: Get_Donations_This_Month
   * Object: Donation__c
   * Condition: Donation_Date__c equals THIS_MONTH (or use date range variables StartOfPrevMonth/EndOfPrevMonth depending on design)
   * How Many: All records; Store: All fields

- Loop: Loop_Donations (iterate Get_Donations_This_Month collection)

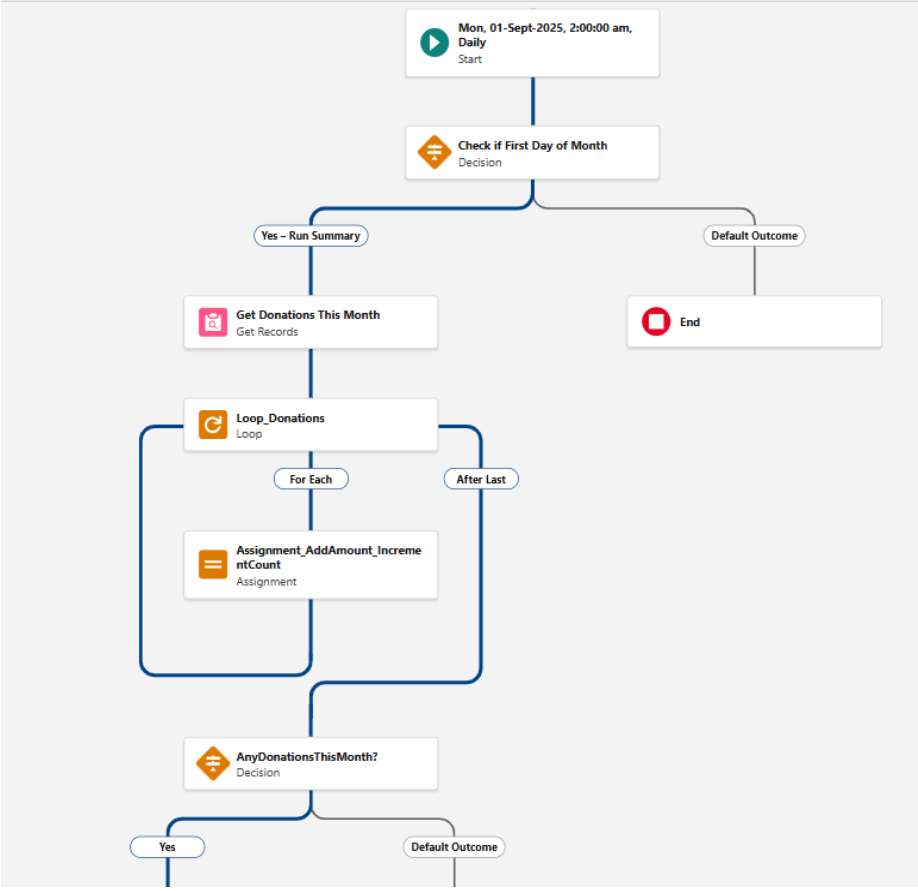
- Assignment element inside loop: Assignment_AddAmount_IncrementCount
   * TotalAmount = TotalAmount + {!Loop_Donations.Amount__c}
   * DonationCount = DonationCount + 1

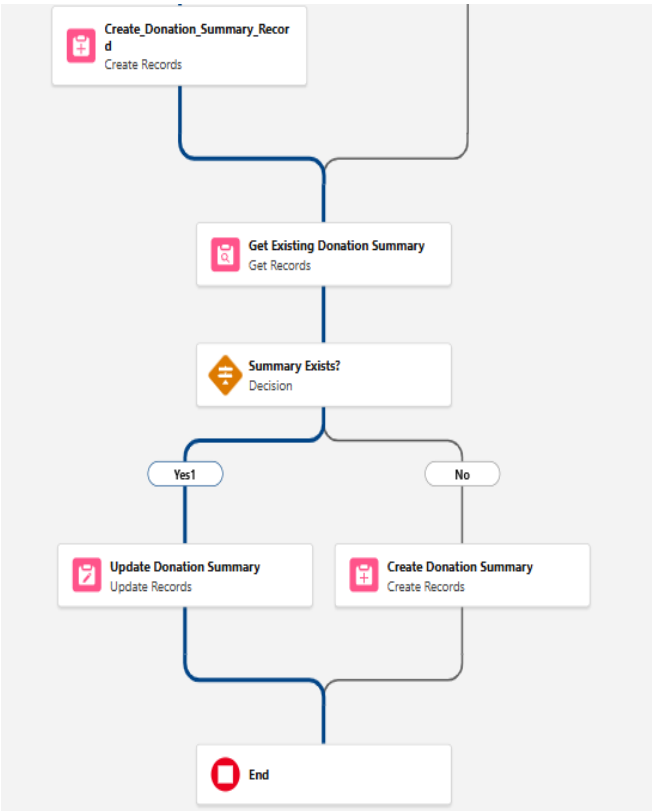
- After loop: Get Records: Get_Existing_Donation_Summary
  * Object: Donation_Summary__c
  * Condition: Month__c equals {!MonthYearText} (or {!date} depending on naming)
  * How many: Only the first record (we check existence)

- Decision: Summary Exists?
  * Yes path -> Update Donation Summary
    - Update record (where Id = {!Get_Existing_Donation_Summary.Id}) set:
      Total_Amount__c = {!TotalAmount}
      Total_Donations__c = {!DonationCount}

* No path -> Create Donation Summary
    - Create record Donation_Summary__c with fields:
      Month__c = {!MonthYearText}
      Total_Amount__c = {!TotalAmount}
      Total_Donations__c = {!DonationCount}

6. Save as 'Monthly Donation Summary (Daily Check)' and Activate.





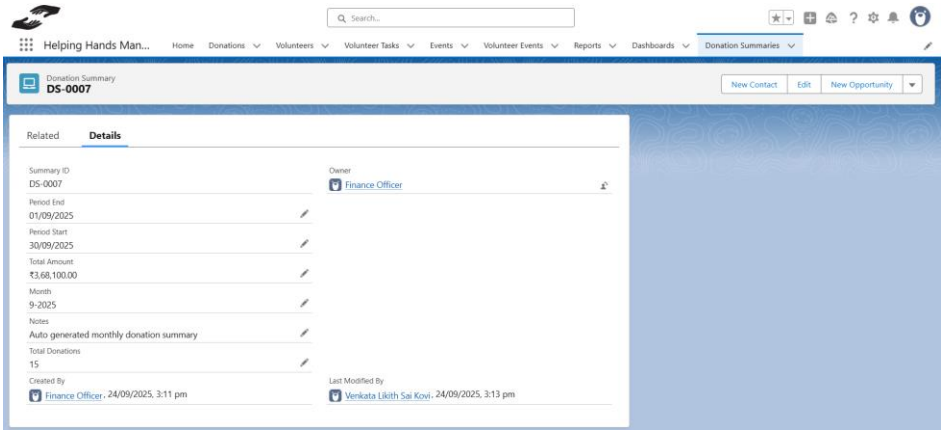
Test examples & debug steps :

• Created test donations for Sept-2025 (various amounts) and ran Debug with RunNow = True to bypass date check.

• Observed Debug log showing Decision 'Summary Exists?' = Yes and Update path executed.

• Debug outputs showed values: TotalAmount = 3,21,100.00; DonationCount = 12 (example run). Later run produced TotalAmount = 3,68,100.00; DonationCount = 15.

• Important: Debug default runs in rollback mode; uncheck 'Run flow in rollback mode' to persist changes.



Step 6: Reports & Dashboards — Detailed Build

Objective: Provide Finance and Executive stakeholders with actionable visuals summarizing donations, donors, and volunteer contributions.

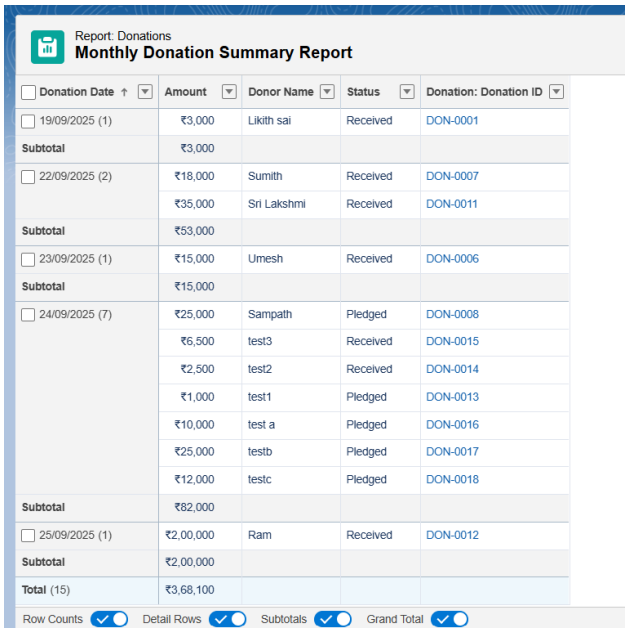
Report : Monthly Donation Summary Report (detailed build):

1. App Launcher → Reports → New Report.

2. Select Report Type: Donations (Donation__c) → Continue.

3. In the Report Builder:
   - Add Columns: Donation ID (Name), Donor_Name__c, Amount__c, Donation_Date__c, Linked_Volunteer__c, Payment_Method__c, Status__c
   - Filters: Date Range = This Year (or All Time for historical). Remove default 'My Donations'.
   - Group Rows by: Calendar Month (Donation Date) -> To do this: click Add Group -> Donation Date -> select 'Calendar Month'
   - Summaries: click the chevron on Amount__c -> Summarize -> Sum. Also add Row Count for number of donations.
   - Optional: add filter Status__c = Received to show only completed donations.

4. Save Report Name: Monthly Donation Summary Report. Folder: Helping Hands Reports.



Report: Donations
Monthly Donation Summary Report

Donation Date	Amount	Donor Name	Status	Donation: Donation ID
19/09/2025 (1)	₹3,000	Likith sai	Received	DON-0001
Subtotal	₹3,000			
22/09/2025 (2)	₹18,000	Sumith	Received	DON-0007
	₹35,000	Sri Lakshmi	Received	DON-0011
Subtotal	₹53,000			
23/09/2025 (1)	₹15,000	Umesh	Received	DON-0006
Subtotal	₹15,000			
24/09/2025 (7)	₹25,000	Sampath	Pledged	DON-0008
	₹6,500	test3	Received	DON-0015
	₹2,500	test2	Received	DON-0014
	₹1,000	test1	Pledged	DON-0013
	₹10,000	test a	Pledged	DON-0016
	₹25,000	testb	Pledged	DON-0017
	₹12,000	testc	Pledged	DON-0018
Subtotal	₹82,000			
25/09/2025 (1)	₹2,00,000	Ram	Received	DON-0012
Subtotal	₹2,00,000			
Total (15)	₹3,68,100			

Row Counts Detail Rows Subtotals Grand Total

Dashboard: **2025-September Month Donation Dashboard**

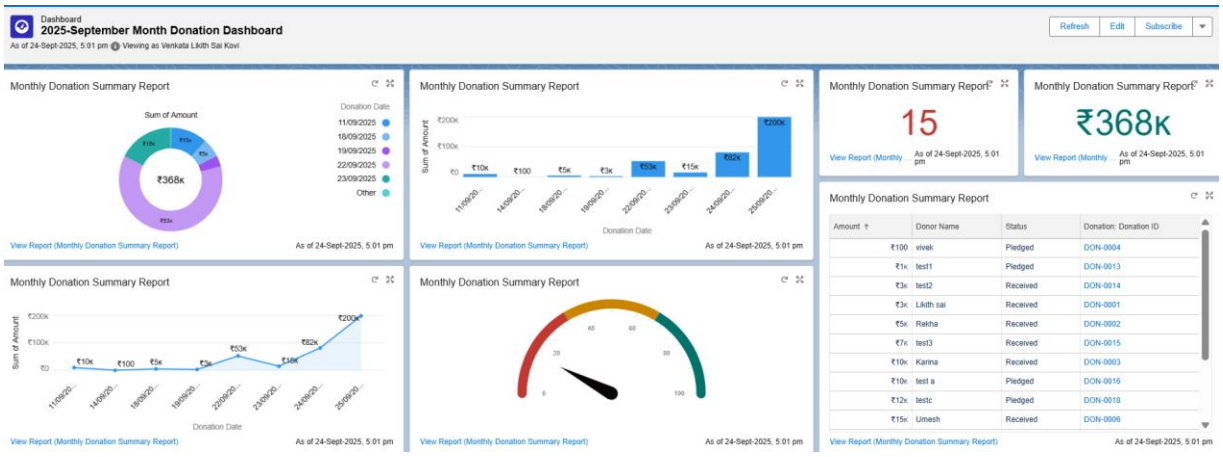
1. App Launcher → Dashboards → New Dashboard.

2. Name: Helping Hands Donation Dashboard. Folder: Helping Hands Reports.

3. Add Components:
   - Component 1: Bar Chart -> Source report: Monthly Donation Summary Report -> X-axis: Month, Y-axis: Sum of Amount
   - Component 2: Donut/Pie -> Source report: Donations by Payment Method (create an ad-hoc report grouped by Payment_Method__c)
   - Component 3: Metric tile -> Source report: Monthly Donation Summary -> configure to show total for current month
   - Component 4: Table -> Top 10 donors by amount (create a report that sorts Amount desc and limit to 10)

4. Configure component titles, display units, and filters as needed.

5. Save Dashboard and share folder permissions with Finance Officer and Admins.



🛰 Phase - 5 Apex Programming

Part 1: Creating the Apex Files

First, we need to create each of the necessary Apex files in your Salesforce org.

Resource:- using the Developer Console.

Step 1.1: Open the Developer Console

1. Log in to your Salesforce org.
2. Click the **Gear Icon** in the top-right corner.
3. Select **Developer Console** from the dropdown menu.

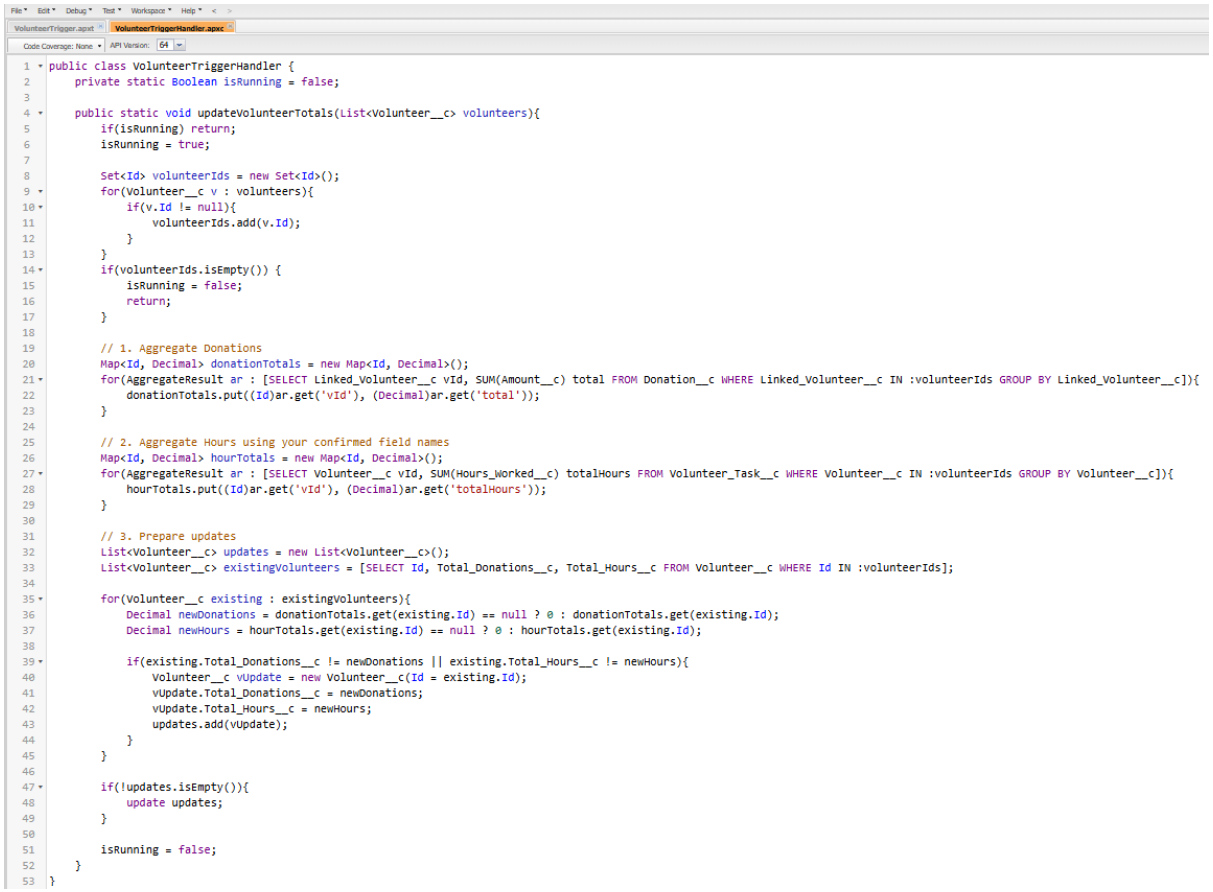
Step 1.2: Create Each Apex File

For each of the six files listed below, you will follow the same process:

- In the Developer Console, click **File → New → Apex Class** (or **Apex Trigger** for the .trigger file).
- Enter the exact file name provided.
- Delete any default text in the editor window.
- Copy and paste the complete code for that file.
- Click **File → Save** or press Ctrl + S.

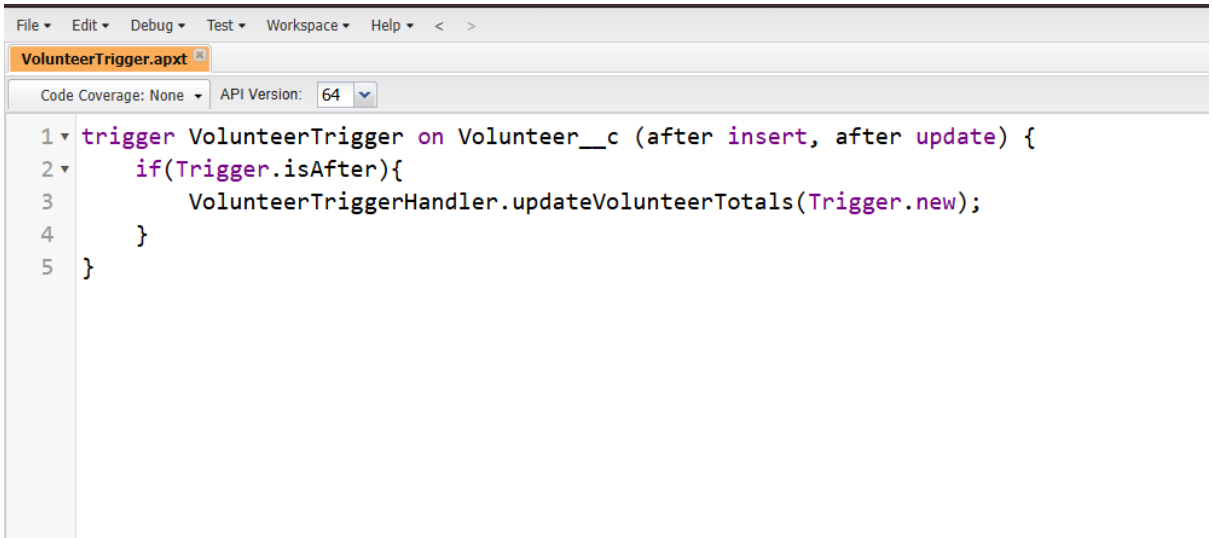
Create the files in this order:

1. **VolunteerTriggerHandler.apxc (Class)**



```
public class VolunteerTriggerHandler {
    private static Boolean isRunning = false;

    public static void updateVolunteerTotals(List<Volunteer__c> volunteers){
        if(isRunning) return;
        isRunning = true;

        Set<Id> volunteerIds = new Set<Id>();
        for(Volunteer__c v : volunteers){
            if(v.Id != null){
                volunteerIds.add(v.Id);
            }
        }
        if(volunteerIds.isEmpty()) {
            isRunning = false;
            return;
        }

        // 1. Aggregate Donations
        Map<Id, Decimal> donationTotals = new Map<Id, Decimal>();
        for(AggregateResult ar : [SELECT Linked_Volunteer__c vId, SUM(Amount__c) total FROM Donation__c WHERE Linked_Volunteer__c IN :volunteerIds GROUP BY Linked_Volunteer__c]){
            donationTotals.put((Id)ar.get('vId'), (Decimal)ar.get('total'));
        }

        // 2. Aggregate Hours using your confirmed field names
        Map<Id, Decimal> hourTotals = new Map<Id, Decimal>();
        for(AggregateResult ar : [SELECT Volunteer__c vId, SUM(Hours_Worked__c) totalHours FROM Volunteer_Task__c WHERE Volunteer__c IN :volunteerIds GROUP BY Volunteer__c]){
            hourTotals.put((Id)ar.get('vId'), (Decimal)ar.get('totalHours'));
        }

        // 3. Prepare updates
        List<Volunteer__c> updates = new List<Volunteer__c>();
        List<Volunteer__c> existingVolunteers = [SELECT Id, Total_Donations__c, Total_Hours__c FROM Volunteer__c WHERE Id IN :volunteerIds];

        for(Volunteer__c existing : existingVolunteers){
            Decimal newDonations = donationTotals.get(existing.Id) == null ? 0 : donationTotals.get(existing.Id);
            Decimal newHours = hourTotals.get(existing.Id) == null ? 0 : hourTotals.get(existing.Id);

            if(existing.Total_Donations__c != newDonations || existing.Total_Hours__c != newHours){
                Volunteer__c vUpdate = new Volunteer__c(Id = existing.Id);
                vUpdate.Total_Donations__c = newDonations;
                vUpdate.Total_Hours__c = newHours;
                updates.add(vUpdate);
            }
        }

        if(!updates.isEmpty()){
            update updates;
        }

        isRunning = false;
    }
}
```

2. VolunteerTrigger.apxt (Trigger)



File ▾ Edit ▾ Debug ▾ Test ▾ Workspace ▾ Help ▾ < >

VolunteerTrigger.apxt

Code Coverage: None ▾ API Version: 64

```
trigger VolunteerTrigger on Volunteer__c (after insert, after update) {
    if(Trigger.isAfter){
        VolunteerTriggerHandler.updateVolunteerTotals(Trigger.new);
    }
}
```

3. VolunteerTriggerTest.apxc (Class)

VolunteerTriggerTest.apxc

Code Coverage: None | API Version: 64

```
@isTest
public class VolunteerTriggerTest {
    @testSetup
    static void setupData() {
        Volunteer__c v = new Volunteer__c(
            Volunteer_Name__c = 'Test Volunteer',
            Email__c = 'vol@test.com',
            Phone_Number__c = '1234567890',
            Availability__c = 'Weekends',
            Status__c = 'Active'
        );
        insert v;

        Donation__c d1 = new Donation__c(
            Donor_Name__c = 'Donor One',
            Donor_Email__c = 'donor1@test.com',
            Amount__c = 100,
            Linked_Volunteer__c = v.Id,   // <-- ADD A COMMA HERE
            Donation_Date__c = Date.today() // <-- ADD A COMMA HERE
        );
            insert d1;
    }

    @isTest
    static void testVolunteerTotalsUpdate() {
        Volunteer__c v = [SELECT Id, Total_Donations__c, Total_Hours__c FROM Volunteer__c LIMIT 1];

        System.assertEquals(100, v.Total_Donations__c, 'Total Donations should be updated');
        System.assertEquals(0, v.Total_Hours__c, 'Total Hours should remain 0 since no Hours__c field exists on Task');
    }
}
```

4. DonationTriggerHandler.apxc (Class)

DonationTriggerHandler.apxc

Code Coverage: None | API Version: 64

```
public class DonationTriggerHandler {
    public static void processDonations(List<Donation__c> donations){
        Set<Id> volunteerIds = new Set<Id>();
        for(Donation__c d : donations){
            if(d.Linked_Volunteer__c != null){
                volunteerIds.add(d.Linked_Volunteer__c);
            }
        }

        if(!volunteerIds.isEmpty()){
            VolunteerTriggerHandler.updateVolunteerTotals(
                [SELECT Id FROM Volunteer__c WHERE Id IN :volunteerIds]
            );
        }

        // Async email
        for(Donation__c d : donations){
            if(d.Donor_Email__c != null){
                System.enqueueJob(new DonationThankYouEmailQueue(d.Id));
            }
        }
    }
}
```

5. DonationTrigger.apxt (Trigger)

DonationTrigger.apxt

Code Coverage: None API Version: 64

```
trigger DonationTrigger on Donation__c (after insert, after update) {
    if(Trigger.isAfter){
        DonationTriggerHandler.processDonations(Trigger.new);
    }
}
```

6. DonationTriggerTest.apxc (Class)

File Edit Debug Test Workspace Help < >

DonationTriggerTest.apxc

Code Coverage: None API Version: 64

```
@isTest
public class DonationTriggerTest {
    @isTest
    static void testDonationProcessingAndEmail() {
        Volunteer__c v = new Volunteer__c(
            Volunteer_Name__c = 'Donor Volunteer',
            Email__c = 'donorvol@test.com',
            Phone_Number__c = '9999999999',
            Availability__c = 'Weekdays',
            Status__c = 'Active'
        );
        insert v;

        Test.startTest();
        Donation__c d = new Donation__c(
            Donor_Name__c = 'Test Donor',
            Donor_Email__c = 'thankyou@test.com',
            Amount__c = 200,
            Linked_Volunteer__c = v.Id,
            Donation_Date__c = Date.today()
        );
        insert d;
        Test.stopTest(); // ensures async Queueable job executes

        v = [SELECT Total_Donations__c FROM Volunteer__c WHERE Id = :v.Id];
        System.assertEquals(200, v.Total_Donations__c, 'Volunteer total donations should update after donation');
    }
}
```

7. DonationThankYouEmailQueue.apxc (Class)

DonationThankYouEmailQueue.apxc

Code Coverage: None | API Version: 64

```apex
public class DonationThankYouEmailQueue implements Queueable {
    private Id donationId;
    public DonationThankYouEmailQueue(Id donationId){
        this.donationId = donationId;
    }

    public void execute(QueueableContext context){
        Donation__c d = [
            SELECT Donor_Email__c, Donor_Name__c, Amount__c
            FROM Donation__c WHERE Id = :donationId LIMIT 1
        ];

        Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();
        mail.setToAddresses(new String[]{d.Donor_Email__c});
        mail.setSubject('Thank you for your donation!');
        mail.setPlainTextBody('Dear ' + d.Donor_Name__c +
            ',\n\nThank you for your generous donation of $' + d.Amount__c +
            '. Your support is greatly appreciated.\n\n- Nonprofit Team');
        Messaging.sendEmail(new Messaging.SingleEmailMessage[]{mail});
    }
}
```

8. VolunteerTaskTriggerHandler.apxc (Class)

VolunteerTaskTriggerHandler.apxc

Code Coverage: None | API Version: 64

```apex
public class VolunteerTaskTriggerHandler {
    public static void updateVolunteerHours(List<Volunteer_Task__c> tasks){
        Set<Id> volunteerIds = new Set<Id>();
        for(Volunteer_Task__c t : tasks){
            if(t.Volunteer__c != null){
                volunteerIds.add(t.Volunteer__c);
            }
        }
        if(!volunteerIds.isEmpty()){
            VolunteerTriggerHandler.updateVolunteerTotals(
                [SELECT Id FROM Volunteer__c WHERE Id IN :volunteerIds]
            );
        }
    }
}
```

9. VolunteerTaskTrigger (Trigger)

VolunteerTaskTrigger.apxt

Code Coverage: None API Version: 64

```
trigger VolunteerTaskTrigger on Volunteer_Task__c (after insert, after update, after delete) {
    if(Trigger.isAfter){
        VolunteerTaskTriggerHandler.updateVolunteerHours(
            Trigger.isDelete ? Trigger.old : Trigger.new
        );
    }
}
```

10. VolunteerTaskTriggerTest (Class)

VolunteerTaskTrigger.apxt

Code Coverage: None API Version: 64

```
trigger VolunteerTaskTrigger on Volunteer_Task__c (after insert, after update, after delete) {
    if(Trigger.isAfter){
        VolunteerTaskTriggerHandler.updateVolunteerHours(
            Trigger.isDelete ? Trigger.old : Trigger.new
        );
    }
}
```

11. VolunteerTotalBatch.apxc (Class)

VolunteerTotalsBatch.apxc

Code Coverage: None API Version: 64

```
public class VolunteerTotalsBatch implements Database.Batchable<SObject>, Database.Stateful {
    public Database.QueryLocator start(Database.BatchableContext bc) {
        return Database.getQueryLocator('SELECT Id FROM Volunteer__c');
    }

    public void execute(Database.BatchableContext bc, List<Volunteer__c> scope) {
        VolunteerTriggerHandler.updateVolunteerTotals(scope);
    }

    public void finish(Database.BatchableContext bc) {
        System.debug('Volunteer totals recalculation completed.');
    }
}
```

12. VolunteerTotalsScheduler.cls (Class)

File Edit Debug Test Workspace Help < >

VolunteerTotalsScheduler.apxc

Code Coverage: None API Version: 64

```
public class VolunteerTotalsScheduler implements Schedulable {
    public void execute(SchedulableContext sc) {
        Database.executeBatch(new VolunteerTotalsBatch(), 200);
    }
}
```

13. **VolunteerTotalsScheduler.cls** (Class)

VolunteerBatchSchedulerTest.apxc

Code Coverage: None | API Version: 64

```
@isTest
public class VolunteerBatchSchedulerTest {
    @isTest
    static void testBatchExecution() {
        // Setup Volunteer with all required fields
        Volunteer__c v = new Volunteer__c(
            Volunteer_Name__c = 'Batch Volunteer',
            Availability__c = 'Weekends',
            Status__c = 'Active',
            Email__c = 'vol1@test.com'
        );
        insert v;

        // Setup Donation
        Donation__c d = new Donation__c(
            Donor_Name__c = 'Test Donor',
            Linked_Volunteer__c = v.Id,
            Amount__c = 500,
            Donation_Date__c = Date.today(),
            Donor_Email__c = 'donor1@test.com'
        );
        insert d;

        // Setup Volunteer Task using your confirmed field names
        Volunteer_Task__c t = new Volunteer_Task__c(
            Name = 'Test Task',
            Volunteer__c = v.Id,
            Hours_Worked__c = 8,
            Priority__c = 'High',
            Due_Date__c = Date.today().addDays(7)
        );
        insert t;

        Test.startTest();
        Database.executeBatch(new VolunteerTotalsBatch(), 1);
        Test.stopTest();

        // Re-query the volunteer to get the updated values
        v = [SELECT Total_Donations__c, Total_Hours__c FROM Volunteer__c WHERE Id = :v.Id];

        System.assertEquals(500, v.Total_Donations__c, 'Batch should recalc donations');
        System.assertEquals(8, v.Total_Hours__c, 'Batch should recalc hours');
    }

    @isTest
    static void testSchedulerExecution() {
        Test.startTest();
        String cron = '0 0 0 1 1 ? 2099';
        System.schedule('Test Volunteer Totals Scheduler', cron, new VolunteerTotalsScheduler());
        Test.stopTest();

        System.assertEquals(1, [SELECT COUNT() FROM CronTrigger WHERE CronJobDetail.Name = 'Test Volunteer Totals Scheduler']);
    }
}
```

After saving all 13 files without any errors, you are ready to run the tests.

Part 2: Running Tests and Capturing Results

Now we will execute the test classes to verify all the logic works correctly and to check your code coverage.

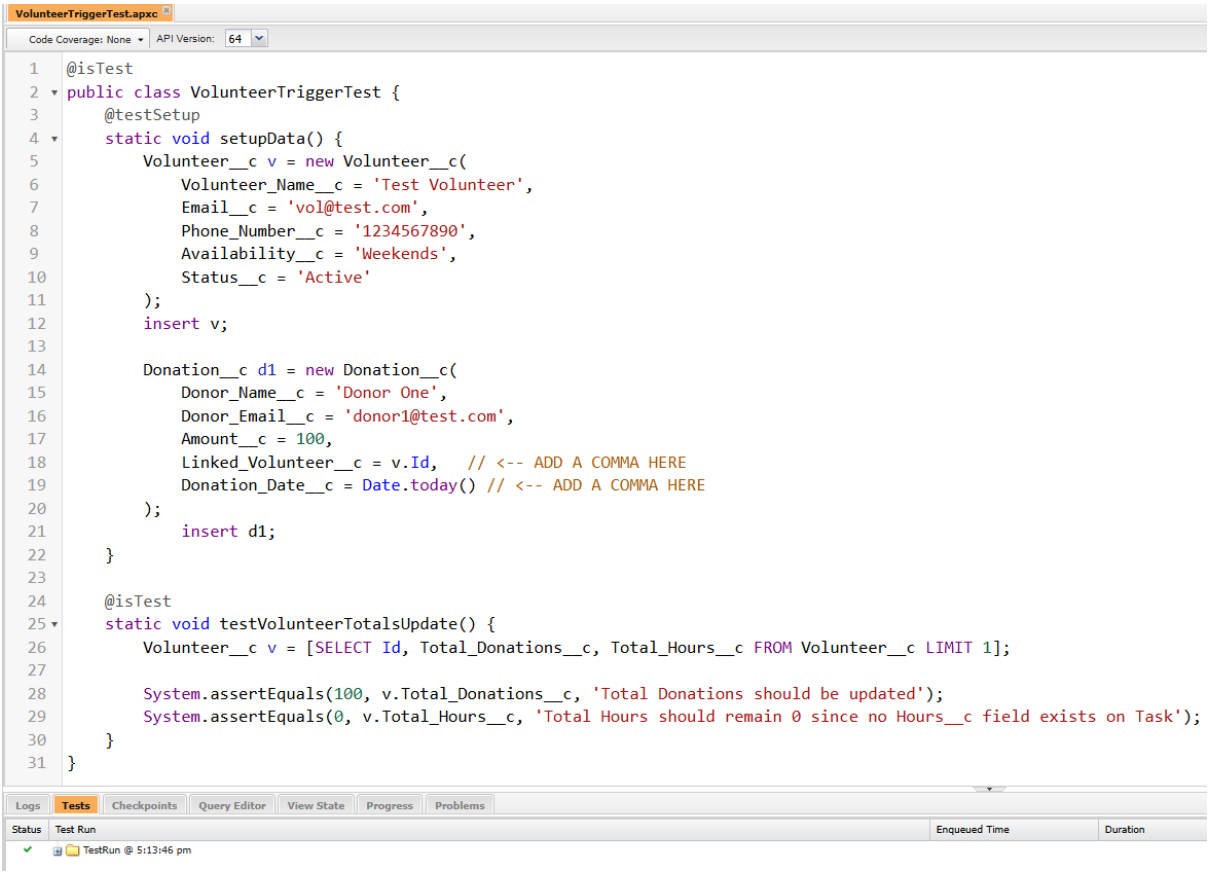
Step 2.1: Navigate to the Test Execution Tab

1. In the Developer Console, click on the **Test** tab in the menu bar.
2. Select **New Run**.

Step 2.2: Run the Trigger Test

1. From the "Test Classes" list, select **VolunteerTriggerTest**.
2. Click the **Add Selected** button to move it to the right-hand column.
3. Click the **Run** button.

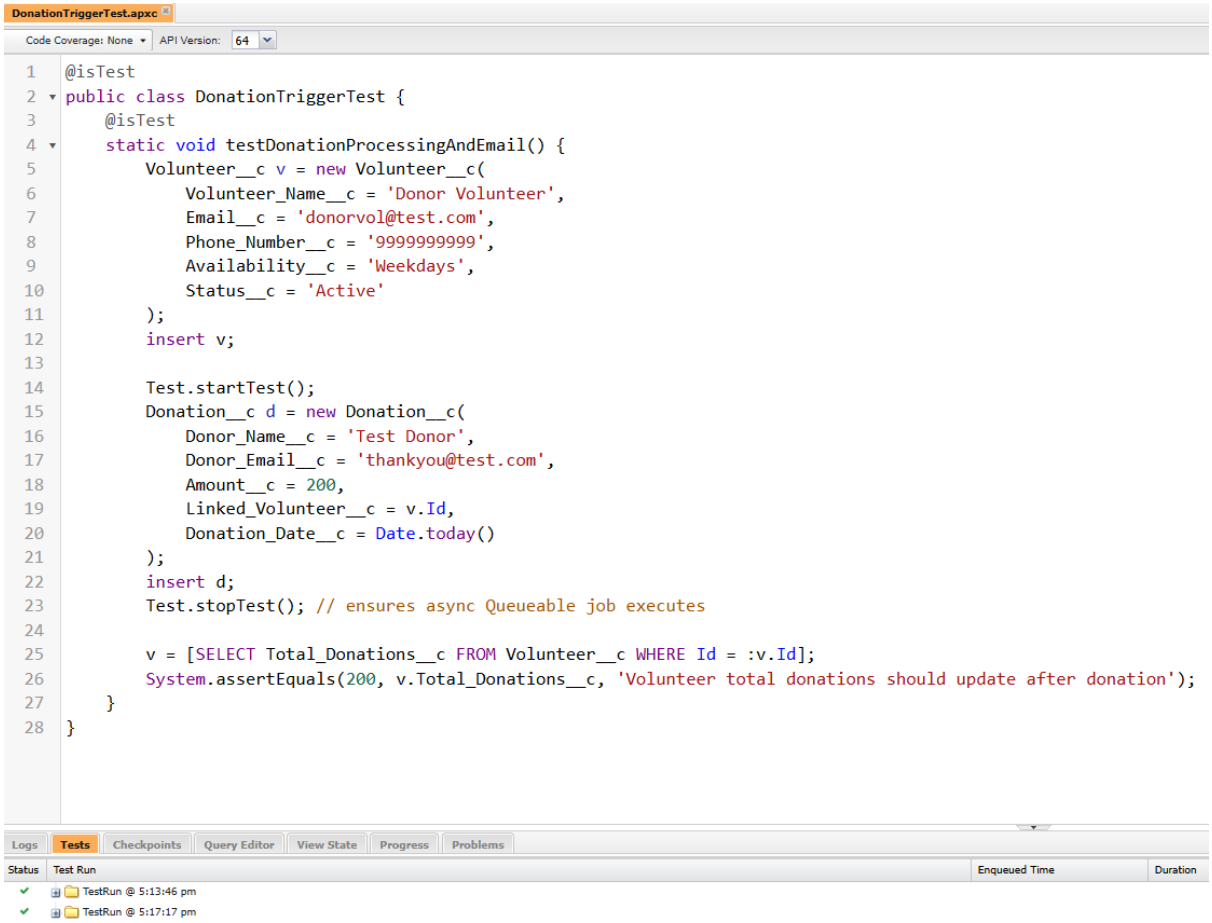
The test will execute, and the results will appear in the "Tests" tab.



Step 2.3: Run the Trigger Test

1. From the "Test Classes" list, select **DonationTriggerTest**.
2. Click the **Add Selected** button to move it to the right-hand column.
3. Click the **Run** button.

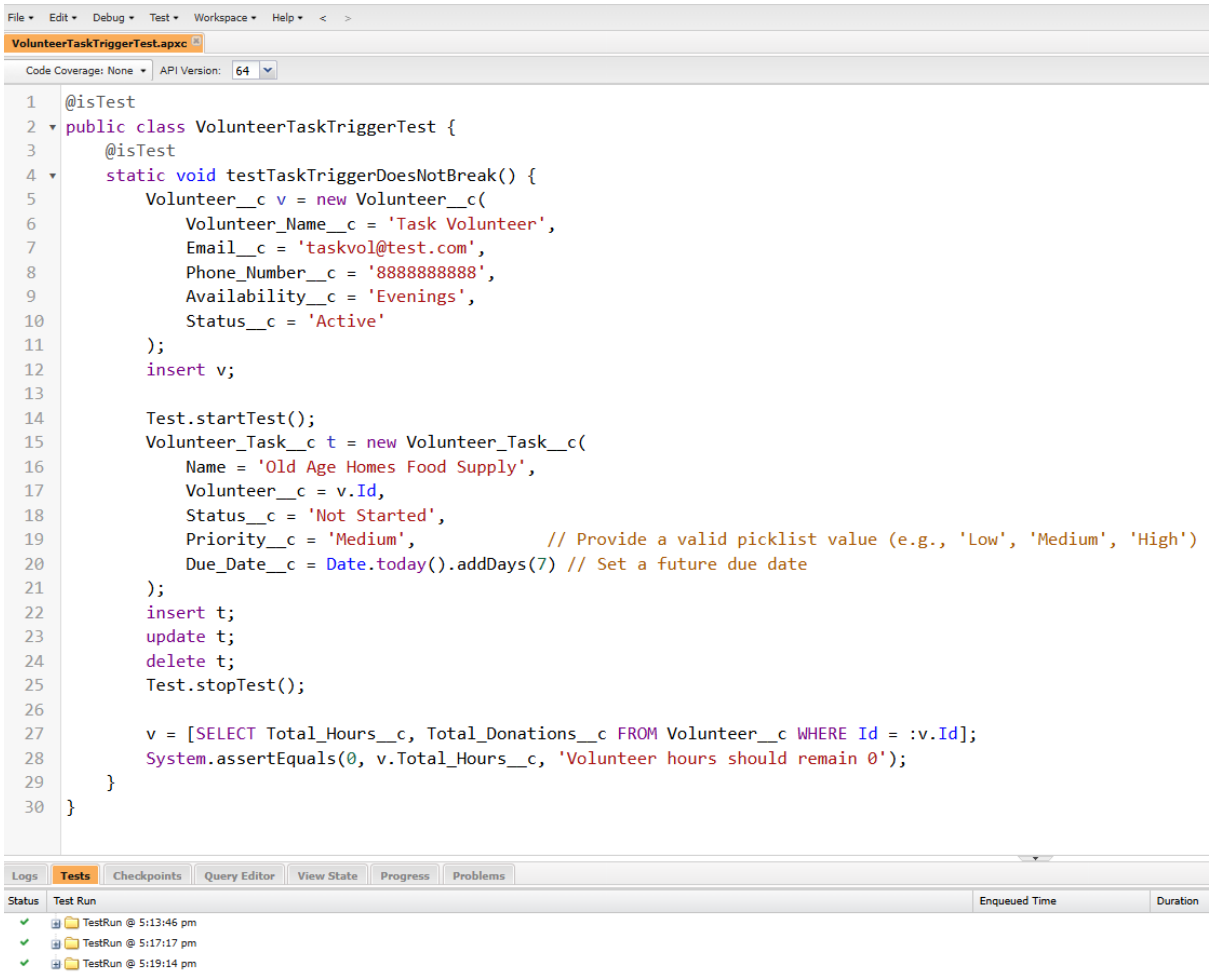
The test will execute, and the results will appear in the "Tests" tab.



Step 2.3: Run the Trigger Test

1. From the "Test Classes" list, select **VolunteerTaskTriggerTest**.
2. Click the **Add Selected** button to move it to the right-hand column.
3. Click the **Run** button.

The test will execute, and the results will appear in the "Tests" tab.



Step 2.4: Run the Batch and Scheduler Test

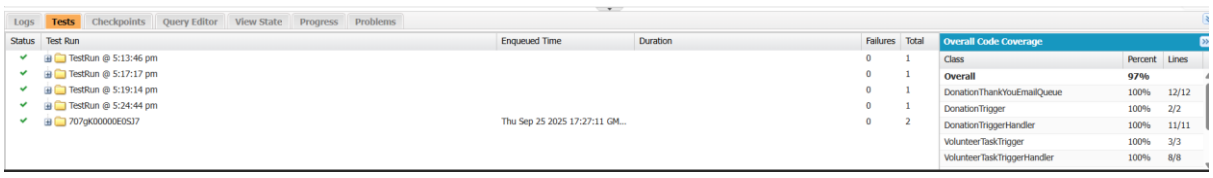
1. Just like before, click **Test → New Run**.
2. This time, select **VolunteerBatchSchedulerTest** from the list.
3. Click **Add Selected**, then click **Run**.

```
@isTest
public class VolunteerBatchSchedulerTest {
    @isTest
    static void testBatchExecution() {
        // Setup Volunteer with all required fields
        Volunteer__c v = new Volunteer__c(
            Volunteer_Name__c = 'Batch Volunteer',
            Availability__c = 'Weekends',
            Status__c = 'Active',
            Email__c = 'vol1@test.com'
        );
        insert v;

        // Setup Donation
        Donation__c d = new Donation__c(
            Donor_Name__c = 'Test Donor',
            Linked_Volunteer__c = v.Id,
            Amount__c = 500,
            Donation_Date__c = Date.today(),
            Donor_Email__c = 'donor1@test.com'
        );
        insert d;

        // Setup Volunteer Task using your confirmed field names
        Volunteer_Task__c t = new Volunteer_Task__c(
            Name = 'Test Task',
            Volunteer__c = v.Id,
            Hours_Worked__c = 8,
            Priority__c = 'High',
            Due_Date__c = Date.today().addDays(7)
        );
        insert t;

        Test.startTest();
        Database.executeBatch(new VolunteerTotalsBatch(), 1);
        Test.stopTest();

        // Re-query the volunteer to get the updated values
        v = [SELECT Total_Donations__c, Total_Hours__c FROM Volunteer__c WHERE Id = :v.Id];

        System.assertEquals(500, v.Total_Donations__c, 'Batch should recalc donations');
        System.assertEquals(8, v.Total_Hours__c, 'Batch should recalc hours');
    }

    @isTest
    static void testSchedulerExecution() {
        Test.startTest();
        String cron = '0 0 0 1 1 ? 2099';
        System.schedule('Test Volunteer Totals Scheduler', cron, new VolunteerTotalsScheduler());
        Test.stopTest();

        System.assertEquals(1, [SELECT COUNT() FROM CronTrigger WHERE CronJobDetail.Name = 'Test Volunteer Totals Scheduler']);
    }
}
```

Step 2.4: Check Overall Code Coverage

After all tests have passed, you can view your organization's overall Apex code coverage.

1. In the "Tests" tab of the Developer Console, look at the top-right corner. You will see "Overall Code Coverage".
2. Click the name of a class to see which lines were covered by your tests.



Part 3: Scheduling the Nightly Job

The final step is to run the script that officially schedules your batch job to run every night.

Step 3.1: Run the Anonymous Apex Script

1. In the Developer Console, click **Debug → Open Execute Anonymous Window**.
2. Copy and paste the following script into the window:

Apex

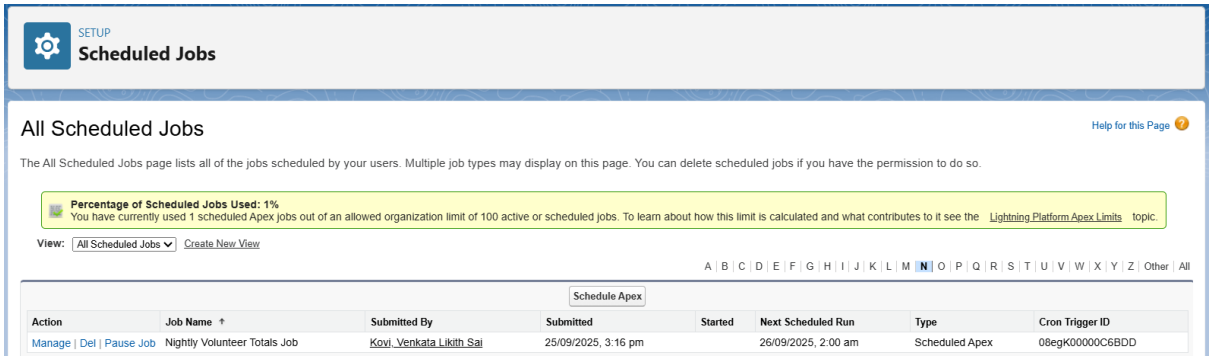
```
VolunteerTotalsScheduler job = new VolunteerTotalsScheduler();

String cron = '0 0 2 * * ?'; // Runs daily at 2 AM

System.schedule('Nightly Volunteer Totals Job', cron, job);
```

3. Click the **Execute** button at the bottom right.

Step 3.2: Verify the Job is Scheduled

1. Go back to the main Salesforce window (not the Developer Console).
2. Click the **Gear Icon** and select **Setup**.
3. In the "Quick Find" box on the left, type Scheduled Jobs.
4. Click on the **Scheduled Jobs** link under the "Jobs" heading.

You will see a list of all scheduled jobs in your org.

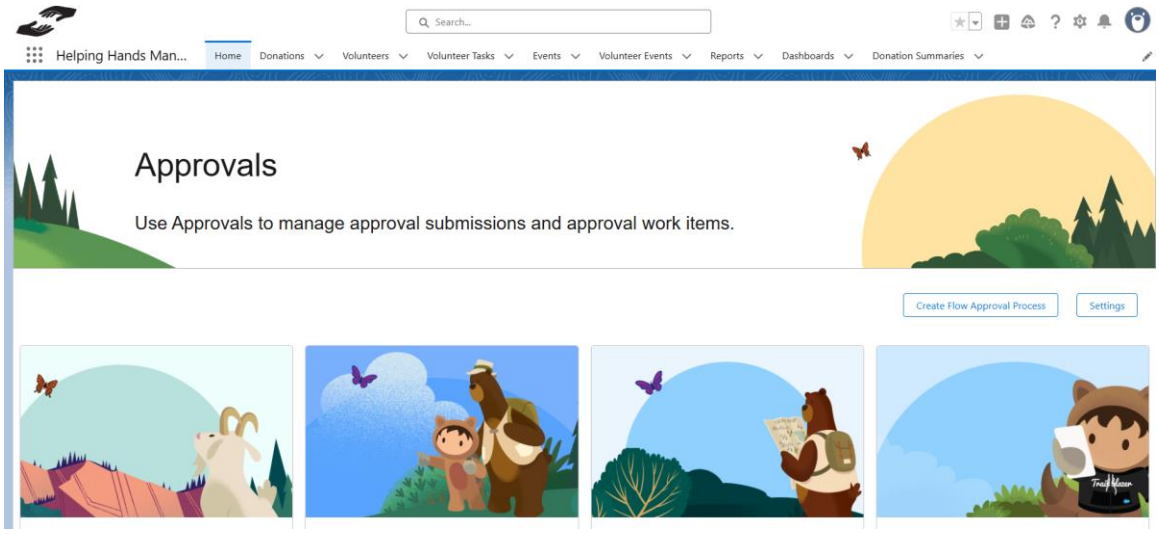


Phase 6 - User Interface Development

In this phase, we will focus on building the User Interface (UI) within the Salesforce platform. The goal is to make it easy for users to view, create, and interact with the Donation Summary and Donor information without using external tools like VS Code. This phase ensures that everything is built directly within Salesforce using Lightning App Builder, Lightning Record Pages, and Custom Tabs.

Step 1: Open Lightning App Builder

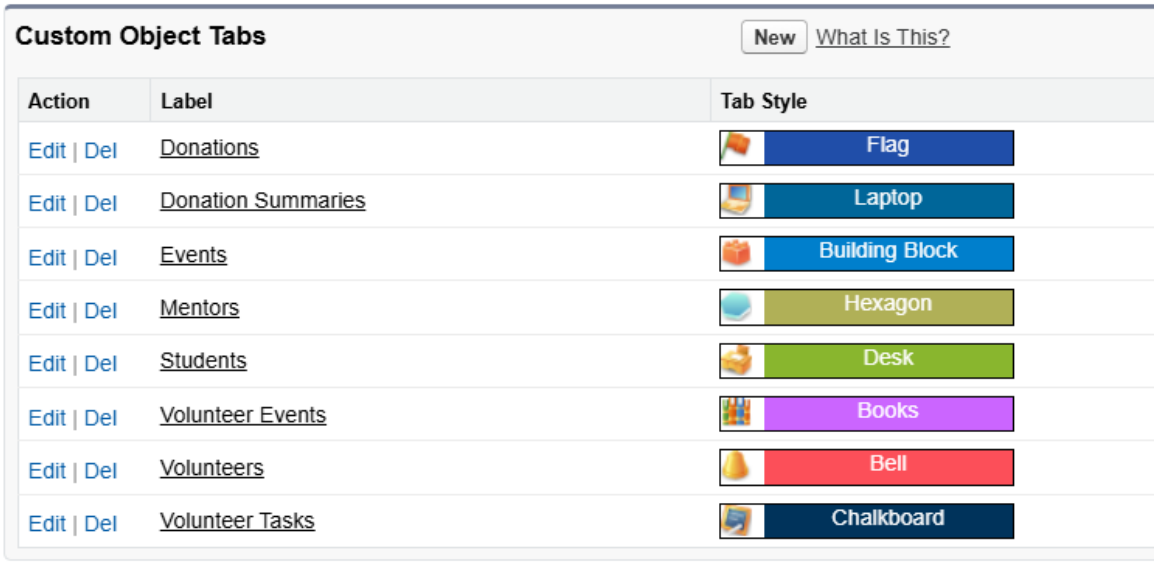
1. Go to **Setup** (gear icon → Setup).
2. In the **Quick Find** box, type **Lightning App Builder**.
3. Click **New**.
4. Choose **App Page** → Next.
5. Name it: Helping Hands Management
6. Select **One Region Layout** (simple layout to start).
7. Click **Finish**.



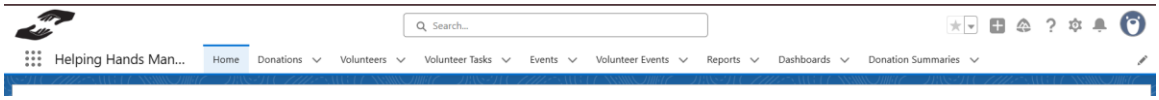
Step 2: Add Custom Tabs for Objects

1. Go to **Setup → Tabs** (Quick Find → Tabs).
2. Under **Custom Object Tabs → New**.
3. Create Tabs for:
    - Donation
    - Volunteer
    - Event
    - Donation Summary

Select a Tab Style (e.g., “Money Bag” for Donation).
4. Save each.



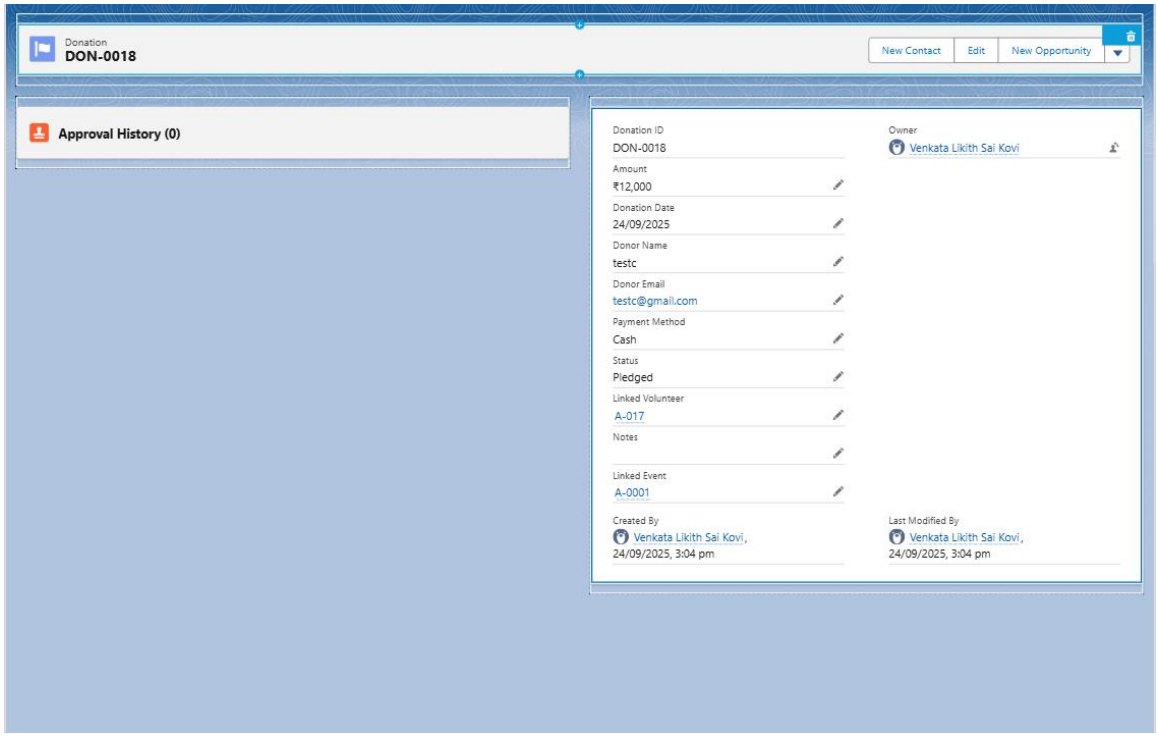
Tabs Output:-



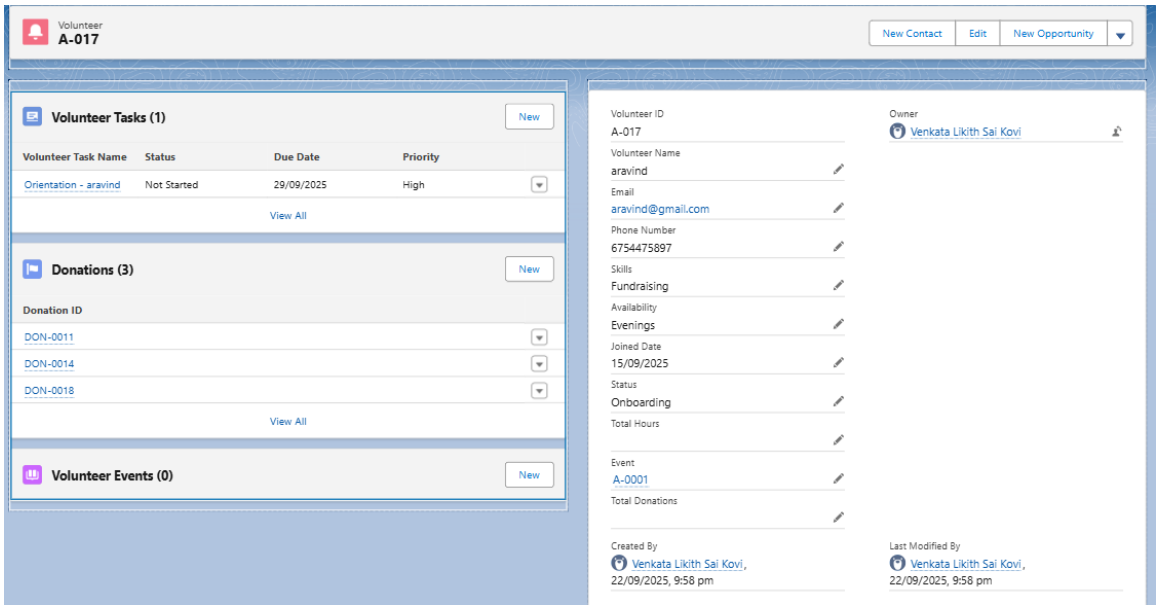
Step 3: Add Record Pages

1. Go to **Object Manager → Donation**.
2. Under **Lightning Record Pages → New**.
3. Select **Record Page** → Name: Donation Record Page.
4. Choose **Header and 2 Columns** layout.
5. Drag **Highlights Panel**, **Related Lists**, and **Details** into the layout.
6. Save & Activate → Assign as **Org Default**.

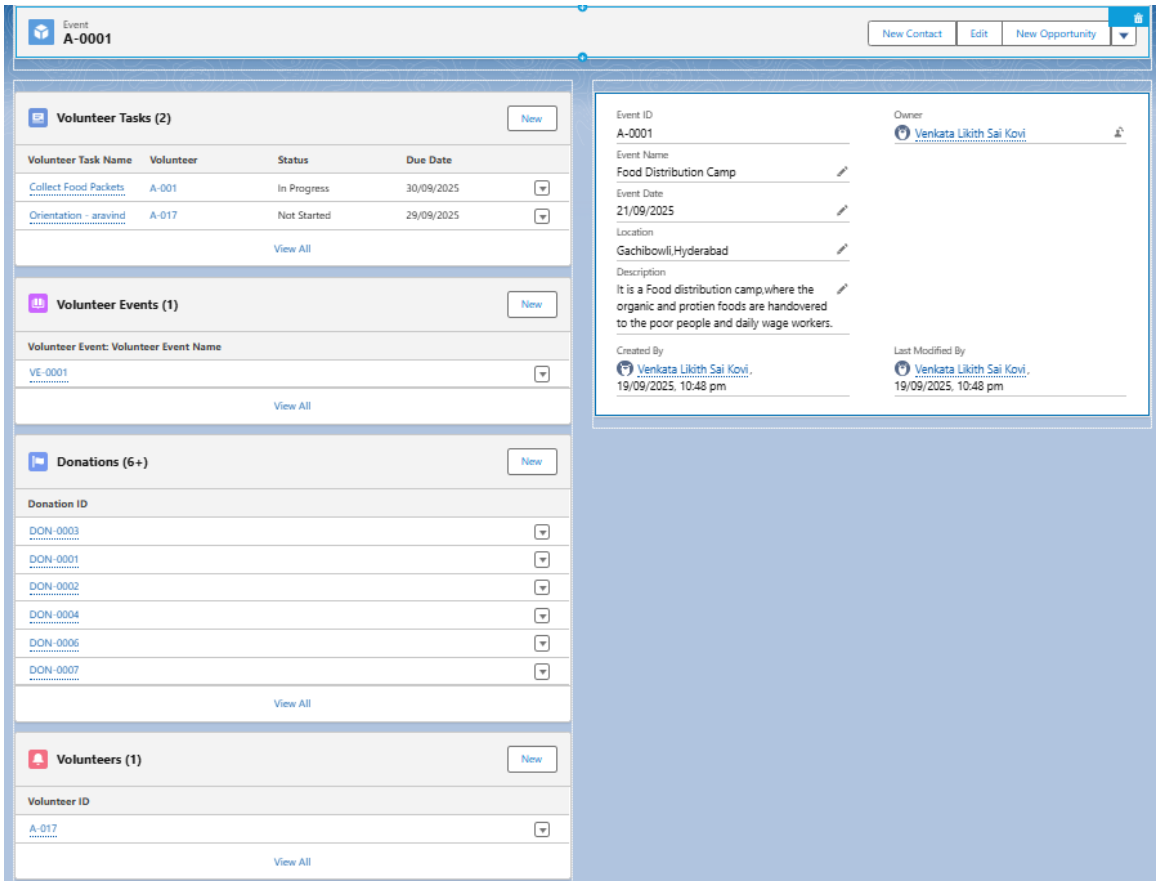
Donation Record Page



Volunteer Record Page



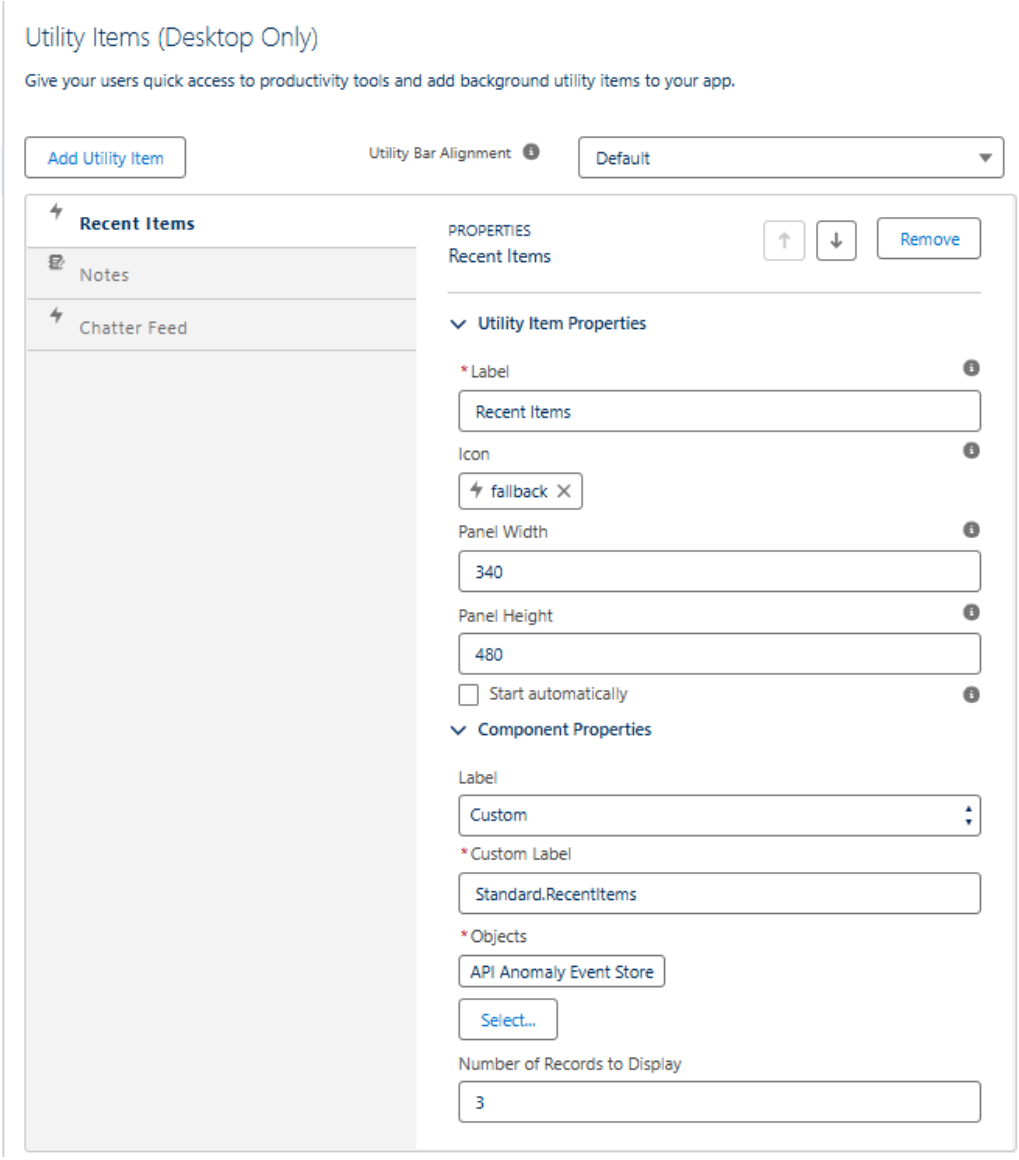
Event Record Page



Step 4: Add Utility Bar (for Quick Access)

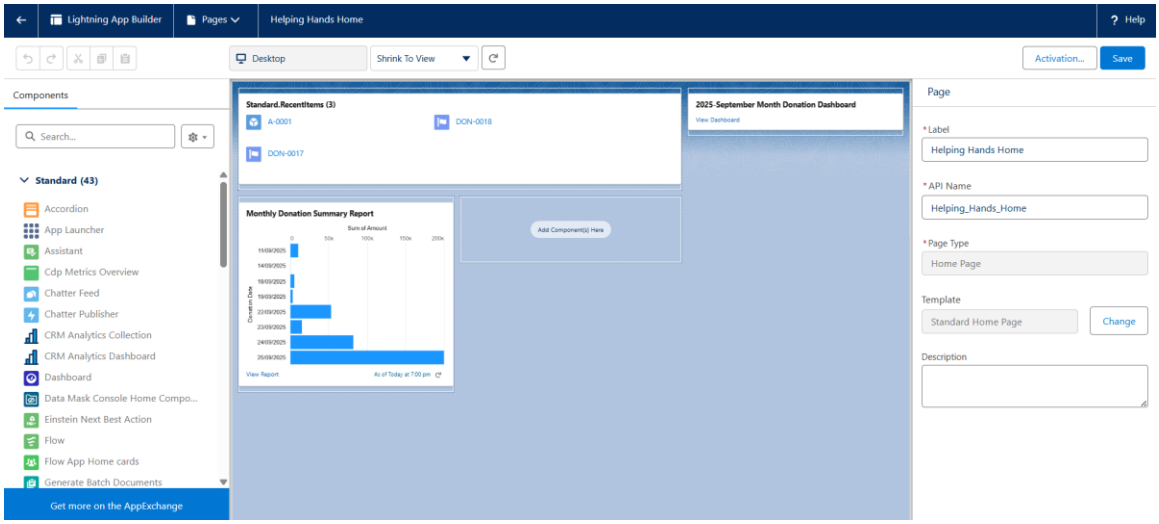
1. Go to **App Manager → Helping Hands App → Edit**.
2. Click **Utility Items** (left panel).
3. Add:
    - **Recent Items**
    - **Notes**
    - **Chatter Feed**

Set **Start Automatically → On**.



Step 5: Create a Custom Home Page

1. Go to **Setup → Lightning App Builder → New → Home Page**.
2. Name it: Helping Hands Home.
3. Choose **Standard Home Page**.
4. Drag components:
    - **Recent Items (Donations, Events)**
    - **Reports (Donation Summary Report)**
    - **Dashboard (Donation Trends)**
5. Save & Activate → Assign to Helping Hands App.



Step 6: Basic LWC (Lightning Web Component)

1. Open **Setup → Developer Console**.
2. Click **File → New → Lightning Component**.
3. Name: donationSummaryCard.
4. In the markup, paste simple code:

donationSummaryCardController.js | donationSummaryCard.cmp | DonationController.apxc

```
<aura:component controller="DonationController" implements="flexipage:availableForAllPageTypes" access="global">
    <!-- attributes to hold values -->
    <aura:attribute name="totalAmount" type="String" />
    <aura:attribute name="totalDonations" type="String" />
    <aura:attribute name="monthLabel" type="String" />

    <!-- init handler -->
    <aura:handler name="init" value="{!this}" action="{!c.doInit}"/>

    <lightning:card title="Donation Summary">
        <div class="slds-p-around_medium">
            <p><strong>Month:</strong> {!v.monthLabel}</p>
            <p><strong>Total Donation Amount:</strong> {!v.totalAmount}</p>
            <p><strong>Total Donations:</strong> {!v.totalDonations}</p>
        </div>
    </lightning:card>
</aura:component>
```

5. We will create an Apex class that returns the most recent Donation_Summary__c record.

1. Setup (⚙) → Quick Find → **Apex Classes** → click **Apex Classes**.
2. Click **New** (top-right).
3. In the editor, paste the exact code below.
4. Click **Save**.

Apex class:

donationSummaryCardController.js | donationSummaryCard.cmp | DonationController.apxc

```
({
    doInit : function(component, event, helper) {
        var action = component.get("c.getLatestDonationSummary");
        action.setCallback(this, function(response) {
            var state = response.getState();
            if (state === "SUCCESS") {
                var data = response.getReturnValue();
                if (data) {
                    component.set("v.totalAmount", data.Total_Amount__c ? data.Total_Amount__c : '0');
                    component.set("v.totalDonations", data.Total_Donations__c ? data.Total_Donations__c : '0');
                    component.set("v.monthLabel", data.Month__c ? data.Month__c : 'N/A');
                } else {
                    component.set("v.totalAmount", '0');
                    component.set("v.totalDonations", '0');
                    component.set("v.monthLabel", 'N/A');
                }
            } else {
                // handle errors for debugging
                var errors = response.getError();
                console.error('Error from server-side action', errors);
                component.set("v.totalAmount", 'Error');
                component.set("v.totalDonations", 'Error');
                component.set("v.monthLabel", 'Error');
            }
        });
        $A.enqueueAction(action);
    }
})
```

6. In Apex Controller:

File ▾ Edit ▾ Debug ▾ Test ▾ Workspace ▾ Help ▾ < >

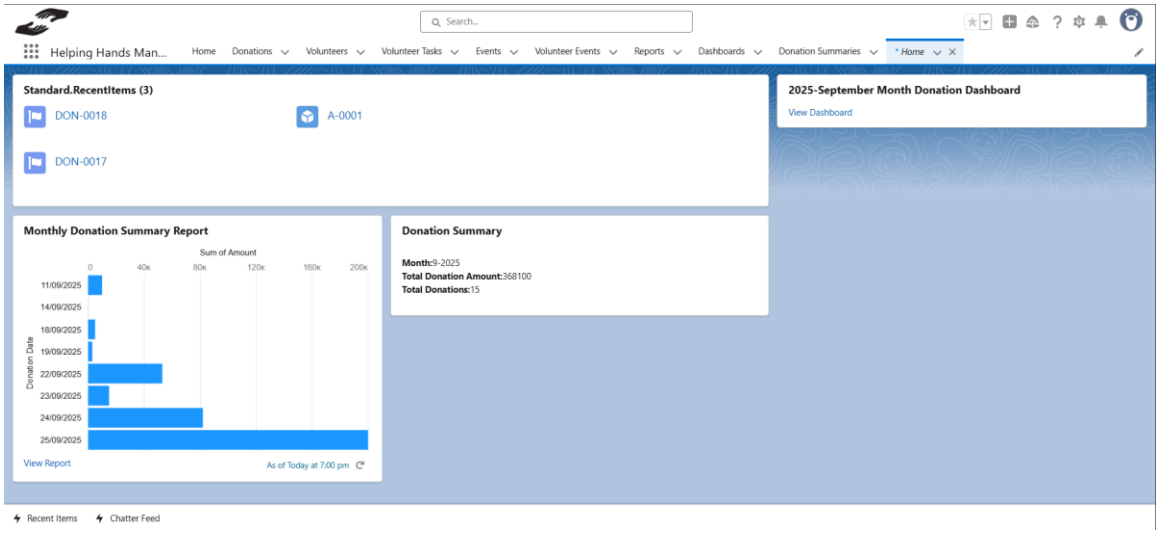
donationSummaryCardController.js | donationSummaryCard.cmp | DonationController.apxc

Code Coverage: None ▾ API Version: 64

```
1  public with sharing class DonationController {
2      @AuraEnabled(cacheable=true)
3      public static Donation_Summary__c getLatestDonationSummary() {
4          List<Donation_Summary__c> results = [
5              SELECT Id, Month__c, Total_Amount__c, Total_Donations__c
6              FROM Donation_Summary__c
7              ORDER BY CreatedDate DESC
8              LIMIT 1
9          ];
10         if (results.isEmpty()) {
11             return null;
12         }
13         return results[0];
14     }
15 }
```

Step 7: Place LWC on the Home Page

1. Go to **Lightning App Builder → Home Page → Edit**.
2. Drag **donationSummaryCard** LWC into the Home Page.
3. Save & Activate.



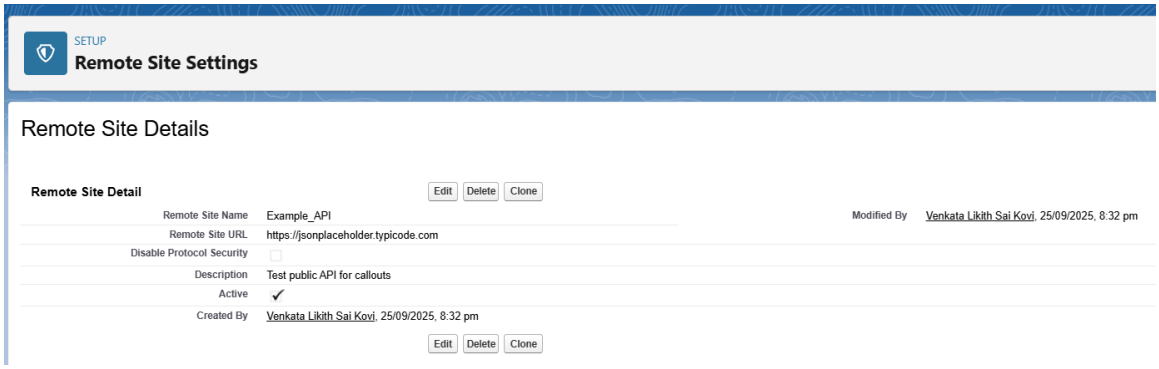
🛰 Phase 7 - Integration & External Access

Contents

A. Remote Site Settings (Quick Allow List)

Purpose: If you will call external endpoints with simple HTTP requests (and are not using Named Credentials), you must register the external domain in Remote Site Settings.

1. Click the Gear icon → Setup.
2. In Quick Find box type 'Remote Site Settings' → Click 'Remote Site Settings'.
3. Click New Remote Site.
4. Fill the form exactly:
   - Remote Site Name: Example_API
   - Remote Site URL: https://jsonplaceholder.typicode.com
   - Active: checked
   - Description: Test public API for callouts
5. Click Save.

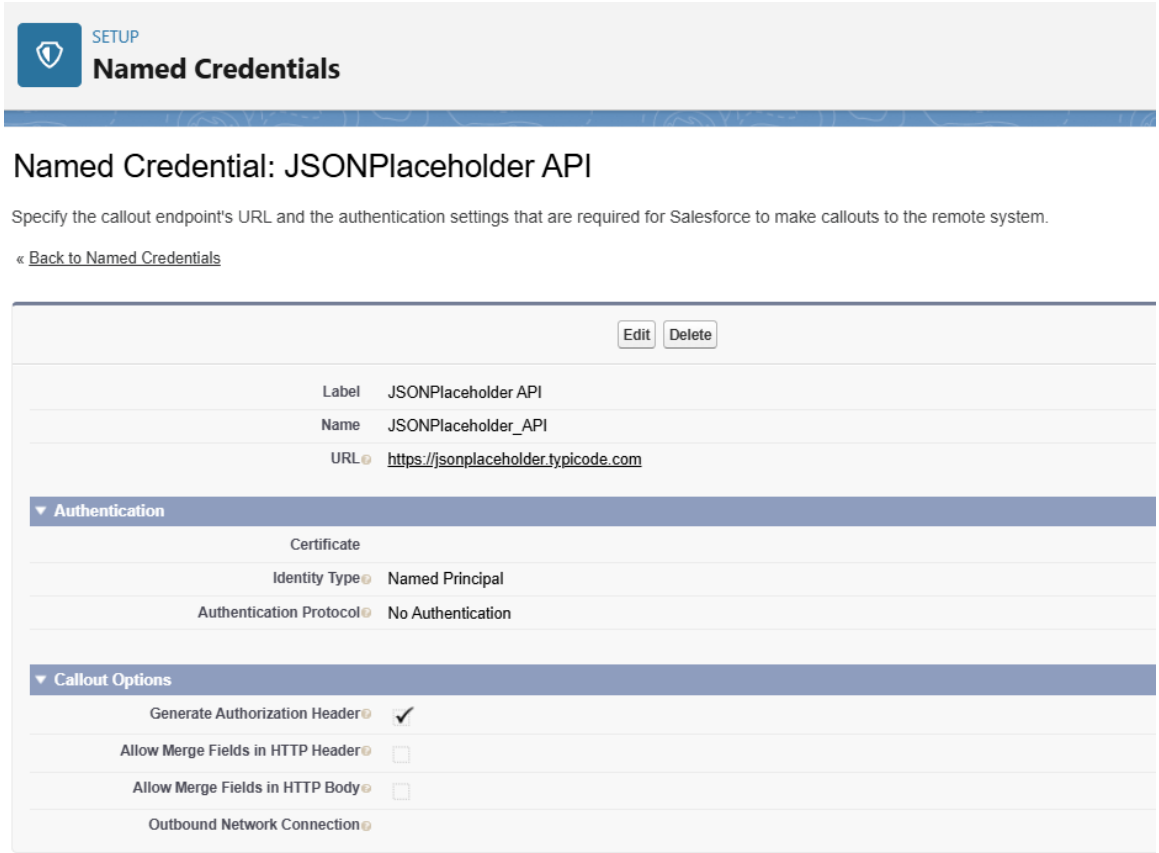


B. Named Credentials (Preferred for Secure Callouts)

Purpose: Named Credentials centralize endpoint URL and authentication. Use them instead of Remote Site Settings whenever possible — they simplify authentication and secure credentials.

Exact steps to create a Named Credential (no auth example):

1. Setup → Quick Find → Named Credentials → Click Named Credentials.
2. Click New Named Credential.
3. Fill values:
   - Label: JSONPlaceholder API
   - Name: JSONPlaceholder_API
   - URL: https://jsonplaceholder.typicode.com
   - Identity Type: Named Principal
   - Authentication Protocol: No Authentication
   - Allow Merge Fields in HTTP Header: unchecked
4. Click Save.



C. Connected App & Auth Provider (for OAuth 2.0)

Purpose: Use a Connected App to allow external systems to authenticate with Salesforce, or to connect Salesforce to an external OAuth provider via Named Credential + Auth Provider.

Create a Connected App (for external OAuth client):

1. Setup → Quick Find → App Manager → Click New Connected App (top-right).

2. Fill the basics:
   - Connected App Name: HelpingHands Integration App
   - API Name: HelpingHands_Integration_App
   - Contact Email: your-email@example.com

3. Under 'API (Enable OAuth Settings)'

check the box: 'Enable OAuth Settings'.
   - Callback URL: https://login.salesforce.com/services/oauth2/success
   - Selected OAuth Scopes: Add 'Full access (full)', 'Perform requests on your behalf at any time (refresh_token, offline_access)'

4. Save the Connected App. Wait ~2–10 minutes for the app to be active.

5. After saving, you will get Consumer Key (Client Id) and Consumer Secret (Client Secret) on the page — copy them for external use.

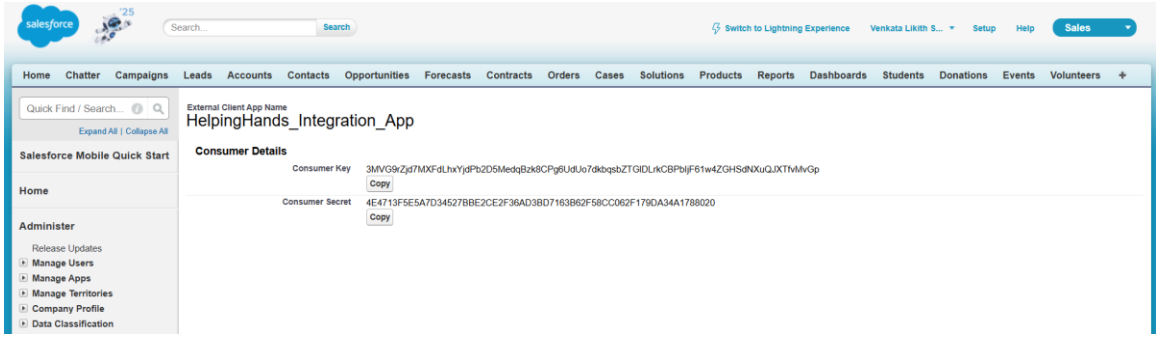
Create an Auth. Provider (if you want Salesforce to use an OAuth provider like Google or Custom OAuth):

1. Setup → Quick Find → Auth. Providers → New.

2. Provider Type: select the provider (e.g., Google) or choose 'OpenID Connect'/'OAuth 2.0' for custom providers.

3. Fill name and the client Id/secret if required.

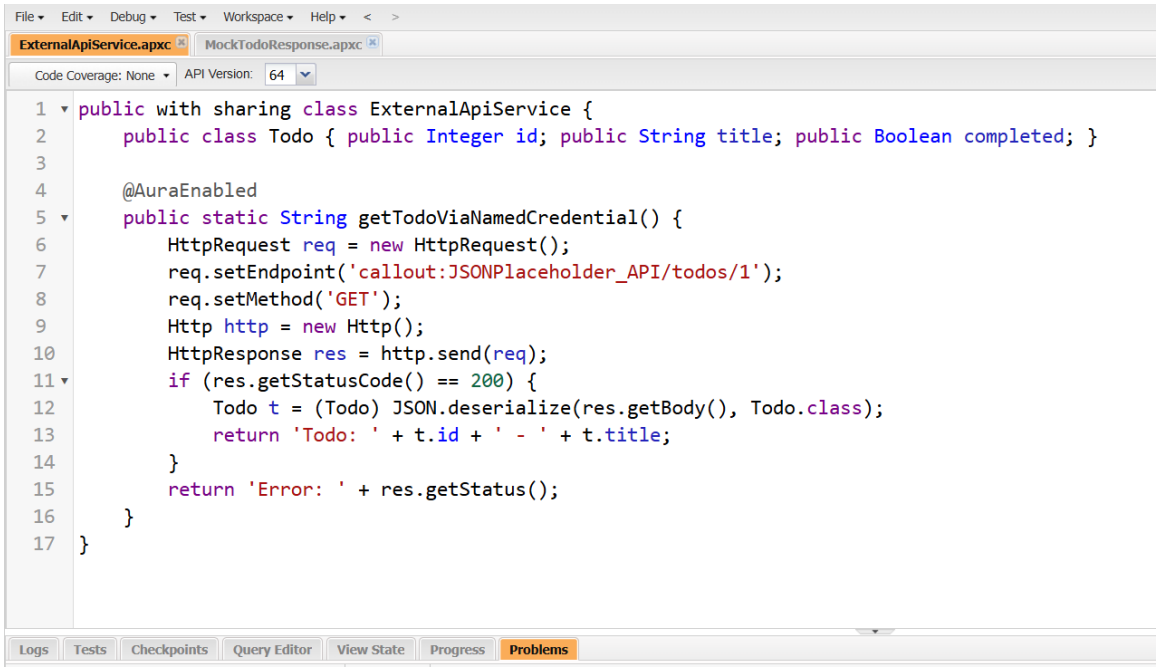
4. Save.



D. Apex HTTP Callouts and Testing (Developer Console)

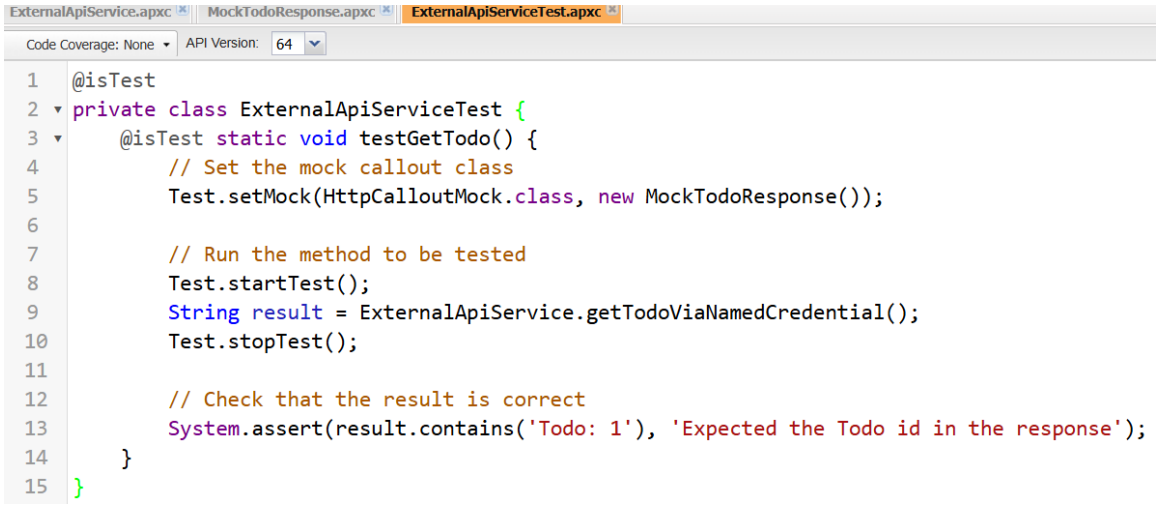
Purpose: This section shows a complete pattern for doing a callout from Apex using a Named Credential and how to test it safely with HttpCalloutMock so tests run without issues.

Step D.1 — Create an Apex class that performs the callout:(MockTodoResponse.apxc)



Step D.2 — Create an HttpCalloutMock for tests(MockTodoResponse.apxc)

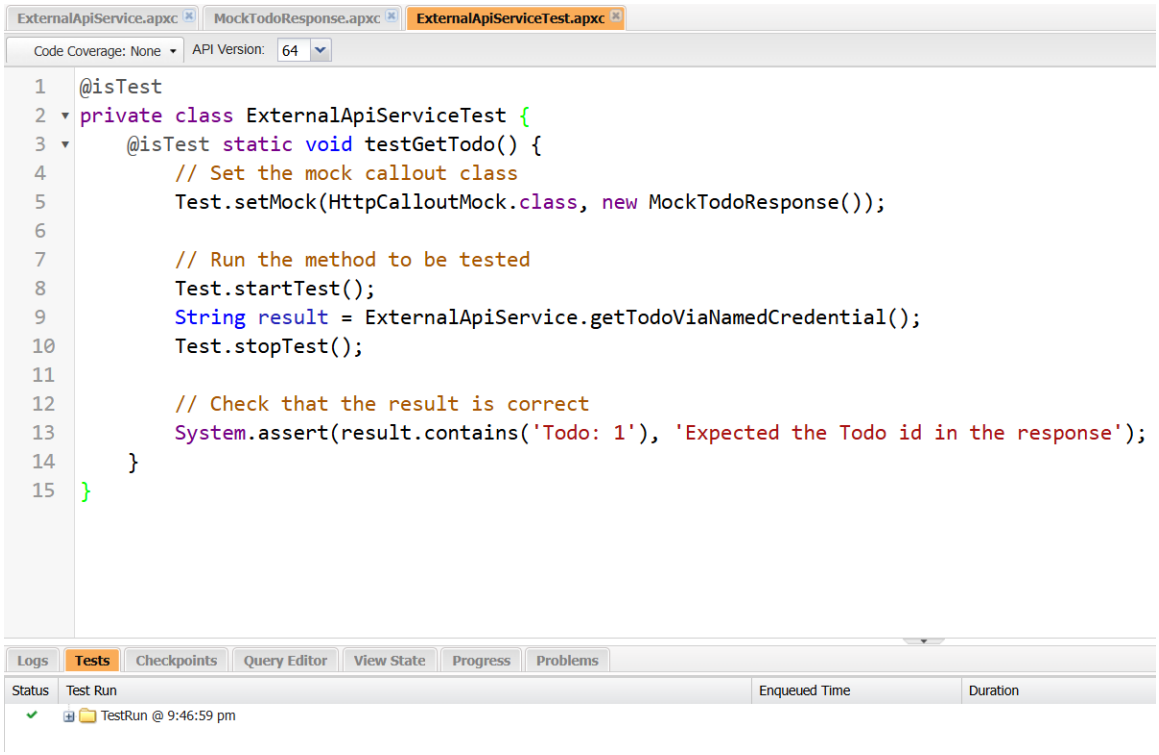
Step D.3 — Create a Test Class that uses the mock and validates behavior:



Step D.4.

open Developer Console → File → New → Apex Class (MockTodoResponse) → Save; then create ExternalApiService class and save;

Then create ExternalApiServiceTest class and save. Run the Apex test via Setup → Apex Test Execution or Developer Console → Test → New Run.



E. External Services (Register an OpenAPI / Swagger)

Purpose: External Services lets you import a REST API described by an OpenAPI (Swagger) schema and then use the generated actions in declarative tools (Flows).

Steps to Register the External Service

Your guide doesn't provide the required API specification, so I've created a simple one for you to use.

1. Go to **Setup**, use the Quick Find box to search for External Services, and open it.
2. Click the **New External Service** button.
3. On the first screen, select **From API Specification** and click **Next**.
4. Fill out the registration details:
    - **External Service Name**: JSONPlaceholder_Service

- **Named Credential**: Select the JSONPlaceholder_API you created earlier.
- For the **Service Schema**, choose the option Complete JSON. This will open a text box.

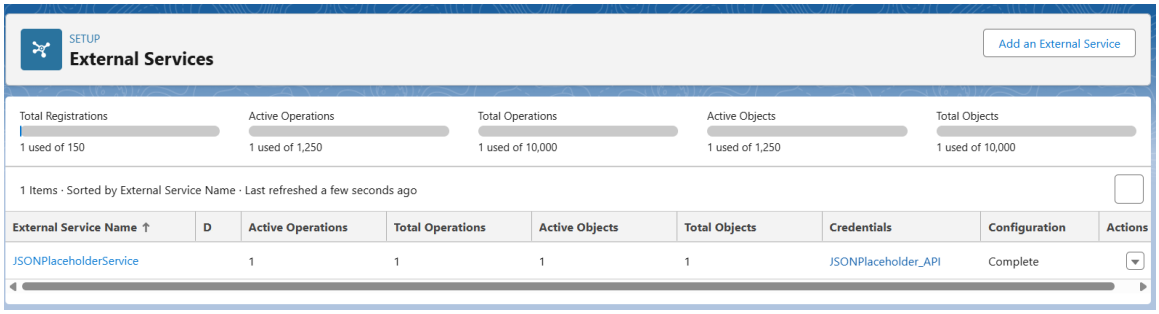
5. JSON

```
{
 "openapi": "3.0.0",
 "info": {
  "title": "JSONPlaceholder Todos API",
  "version": "1.0.0"
 },
 "paths": {
  "/todos/{id}": {
   "get": {
    "summary": "Get a single todo item by ID",
    "parameters": [
     {
      "name": "id",
      "in": "path",
      "required": true,
      "schema": {
       "type": "integer"
      }
     }
    ],
    "responses": {
     "200": {
```

```
        "description": "A single todo item",
        "content": {
          "application/json": {
            "schema": {
              "$ref": "#/components/schemas/Todo"
            }
          }
        }
      }
    }
  }
},
"components": {
  "schemas": {
    "Todo": {
      "type": "object",
      "properties": {
        "userId": { "type": "integer" },
        "id": { "type": "integer" },
        "title": { "type": "string" },
        "completed": { "type": "boolean" }
      }
    }
  }
}
```

```
}
```

6. Save



F. Salesforce Connect — External Data Source & External Objects

Purpose: Salesforce Connect lets you surface external data as external objects without copying data into Salesforce.

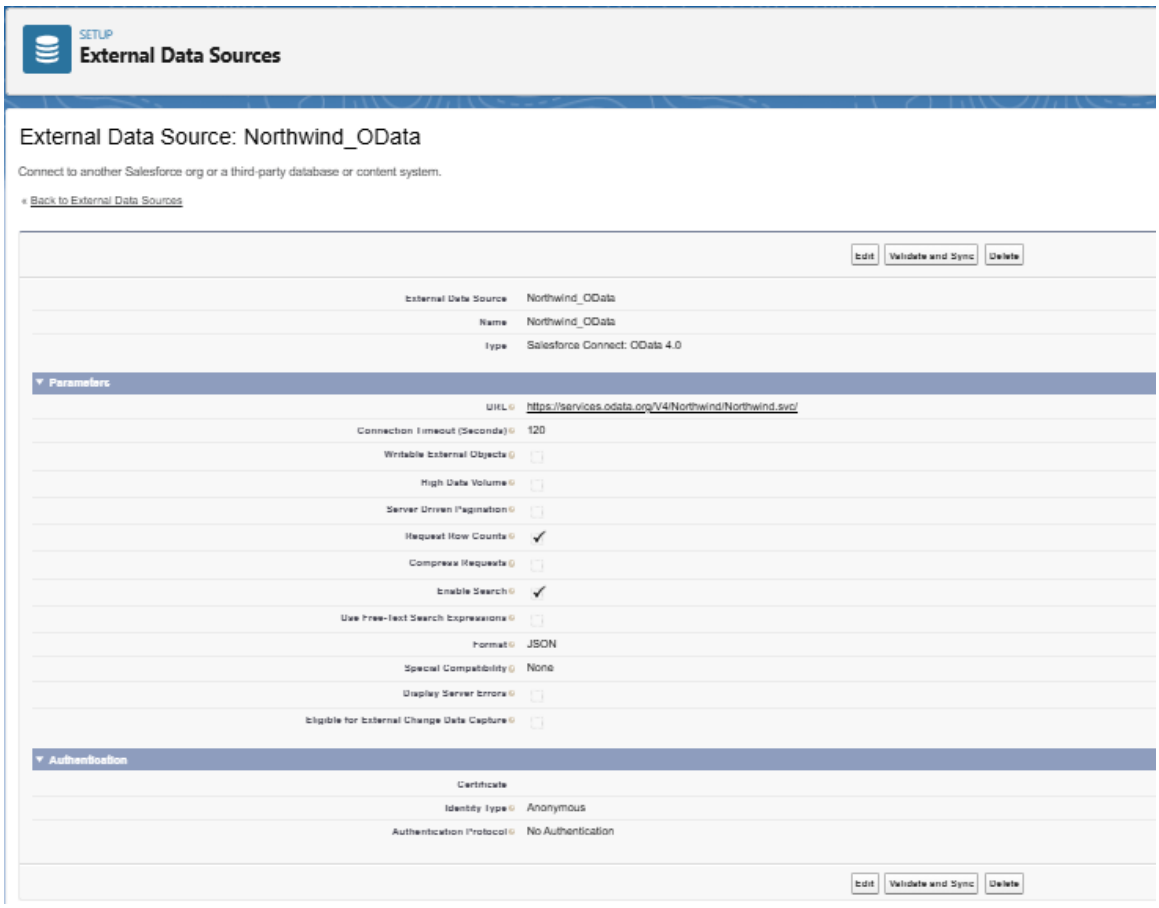
1. Setup → Quick Find → External Data Sources → New External Data Source.

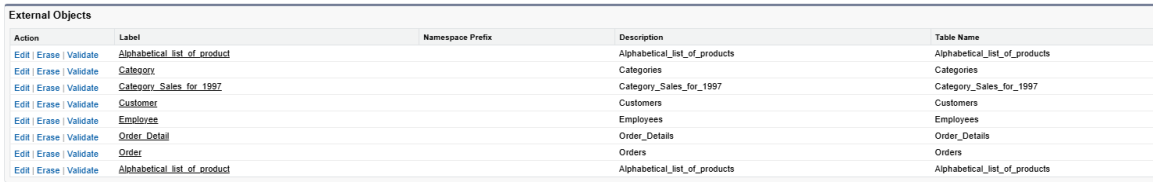
2. Fill values:
   - External Data Source Name: Northwind_OData
   - Type: OData 4.0 (or 2.0 depending on the endpoint)
   - URL: https://services.odata.org/V4/Northwind/Northwind.svc/
   - Identity Type: Anonymous (or Named Principal with Named Credential if auth required)

3. Click Save and then click 'Validate and Sync'.

4. In the Sync UI, select the Entities you want to sync (Products, Orders, etc.) and click 'Sync'.

5. Salesforce will create External Objects (e.g., Product__x) that map to external entities.

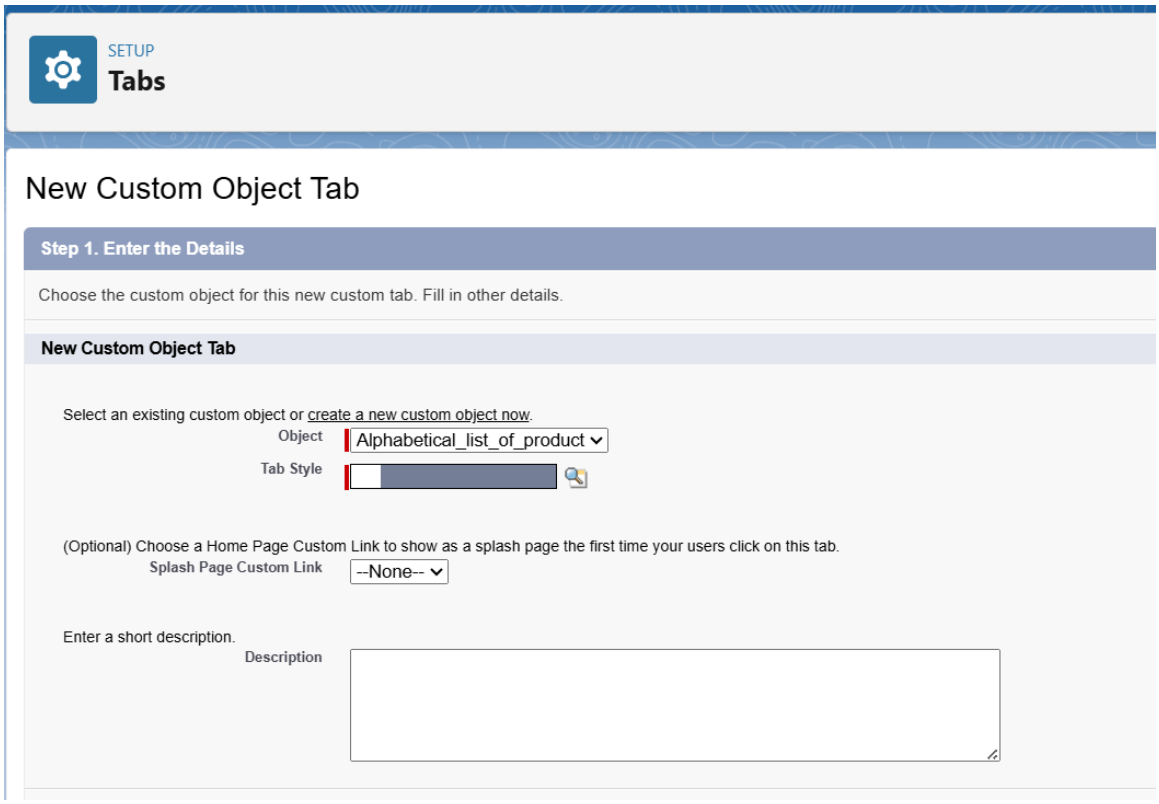




Test Example:

Click the new External Object tab (create a Tab via Setup → Tabs → New → External Object Tab) or browse via App Launcher to view records.

You can also create external lookup relationships between standard/custom objects and external objects.



Custom Object Tabs

New | What Is This?

Action	Label	Tab Style
Edit | Del	Alphabetical_list_of_products	Airplane
Edit | Del	Donations	Flag
Edit | Del	Donation Summaries	Laptop
Edit | Del	Events	Building Block
Edit | Del	Mentors	Hexagon
Edit | Del	Students	Desk
Edit | Del	Volunteer Events	Books
Edit | Del	Volunteers	Bell
Edit | Del	Volunteer Tasks	Chalkboard

G. Platform Events — Publish & Subscribe

Purpose: Platform Events deliver custom event messages within Salesforce (or to external systems). They are great for near real-time integration patterns.

Create a Platform Event: exact clicks:

1. Setup → Quick Find → Platform Events → Click Platform Events.
2. Click 'New Platform Event'.
3. Fill:
   - Label: Volunteer_Update
   - Plural Label: Volunteer_Updates
   - Object Name: Volunteer_Update__e
   - Publish Behavior: Publish After Commit (recommended)
4. Click Save.

Add fields to the Platform Event: e.g., VolunteerId__c (Text), Update_Type__c (Text), Notes__c (Long Text).

Publish a Platform Event via Apex (Developer Console):

Subscribe to Platform Events with an Apex trigger:

VolunteerUpdateEventTrigger.apxt *

Code Coverage: None | API Version: 64

```
trigger VolunteerUpdateEventTrigger on Volunteer_Update__e (after insert) {
    for (Volunteer_Update__e ev : Trigger.New) {
        System.debug('Platform Event received: ' + ev);
    }
}
```

5. Publish an Event to Test It

Now I manually publish one event to see if your trigger is working.

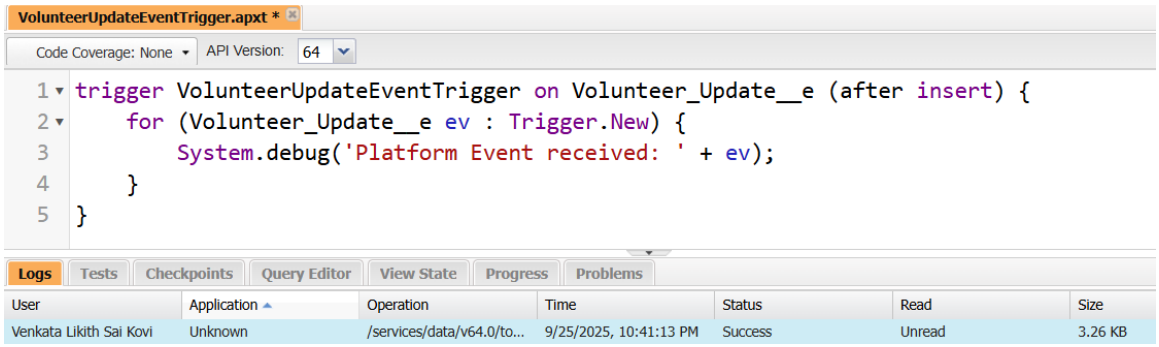
1. In the **Developer Console**, click **Debug** > **Open Execute Anonymous Window**.
2. In the window that pops up, paste the following code. (The VolunteerId__c is just a placeholder, you don't need to change it).

Apex

```
Volunteer_Update__e ev = new Volunteer_Update__e(

  VolunteerId__c = 'a0B...',
```

```
    Update_Type__c = 'Assigned',
    Notes__c = 'Auto-assigned task'
);
Database.SaveResult sr = EventBus.publish(ev);
```

6. Check the **Open Log** box at the bottom and click **Execute**.



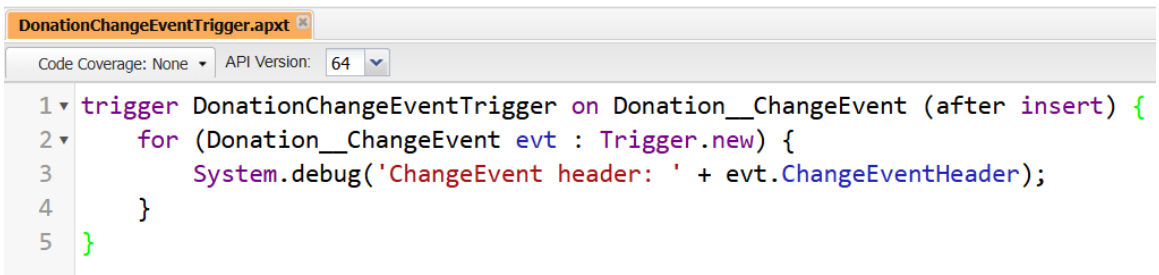
H. Change Data Capture (CDC) — Enable & Subscribe

Purpose: CDC publishes change events for record lifecycle changes (create/update/delete/undelete).

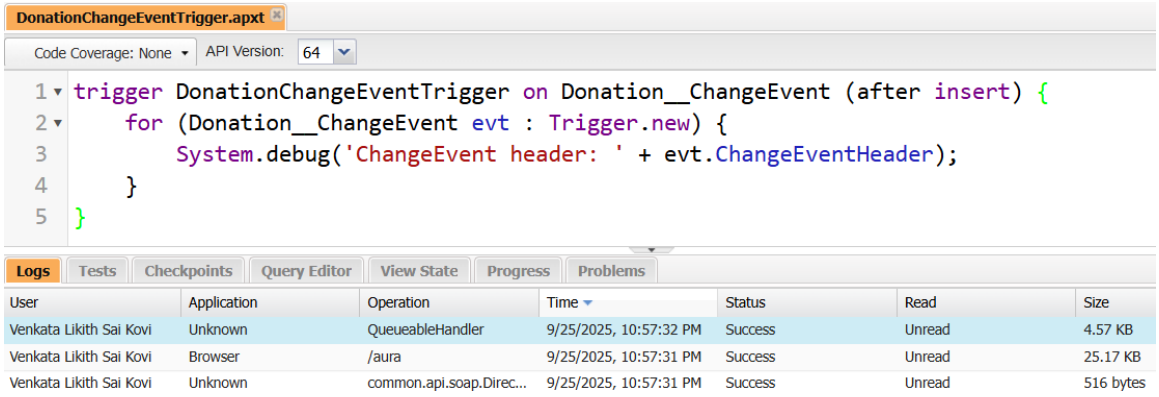
Enable CDC for Donation__c :

1. Setup → Quick Find → Change Data Capture → Click 'Change Data Capture'.
2. From the list of objects, check 'Donation' (Donation__c) and click Save.
3. Now Salesforce will produce Donation__ChangeEvent events whenever Donation__c records change.

Subscribe in Apex by creating a trigger for the change event (Developer Console → New → Apex Trigger):



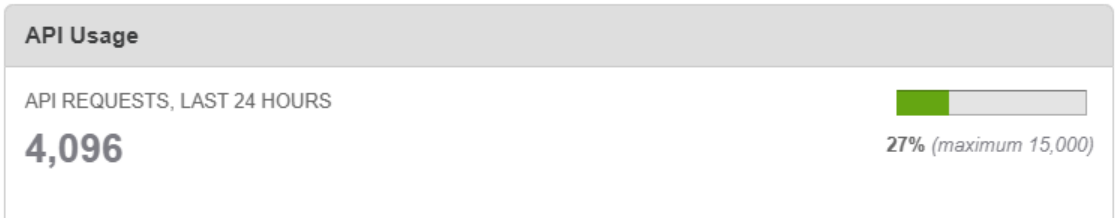
Test: Update a Donation record. Then check Debug Logs or the trigger's System.debug output to confirm the change event fired.



I. API Limits & Monitoring

Purpose: Understand API usage and limits so you design efficient integrations.
How to check API usage in Salesforce (click-by-click):

1. Setup → Quick Find → System Overview → Click System Overview. Review 'API Requests, Last 24 Hours' and other stats.



2. To get programmatic view of limits, you can call the REST endpoint /services/data/vXX.0/limits (use Workbench or Postman with Salesforce authentication).

Quick steps to call the REST limits endpoint (Workbench method):

1. Go to https://workbench.developerforce.com → Login with your dev org credentials.

2. Navigate to Utilities → REST Explorer.
3. Use GET /services/data/v54.0/limits (replace v54.0 with your API version).
4. Review the JSON result which shows Concurrent Async, Daily API Calls, etc.

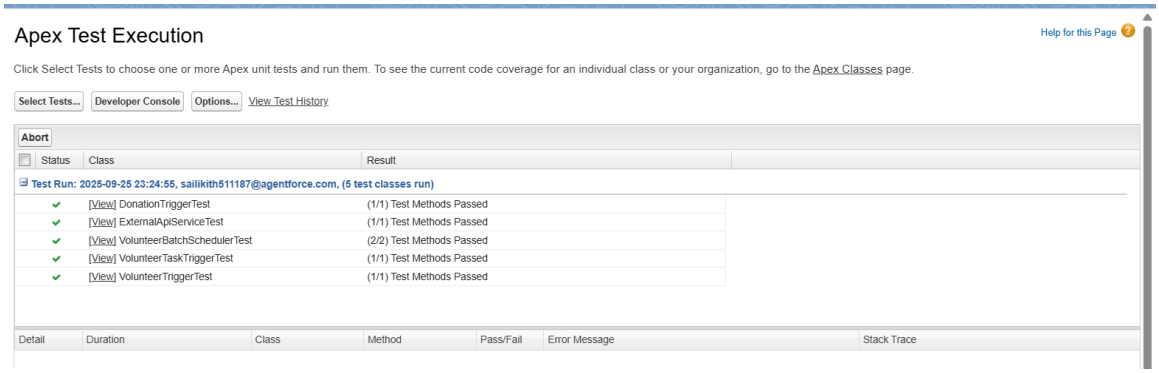
🛬 Phase 8 - Testing & Deployment

1. Unit Testing with Apex

Step 1: Go to Setup → Apex Test Execution.

Step 2: Click 'Select Tests' → choose all test classes (DonationTriggerTest, VolunteerTriggerTest, etc.) and Phase-7 (HttpCalloutMock tests).

Step 3: Run them and wait for results.



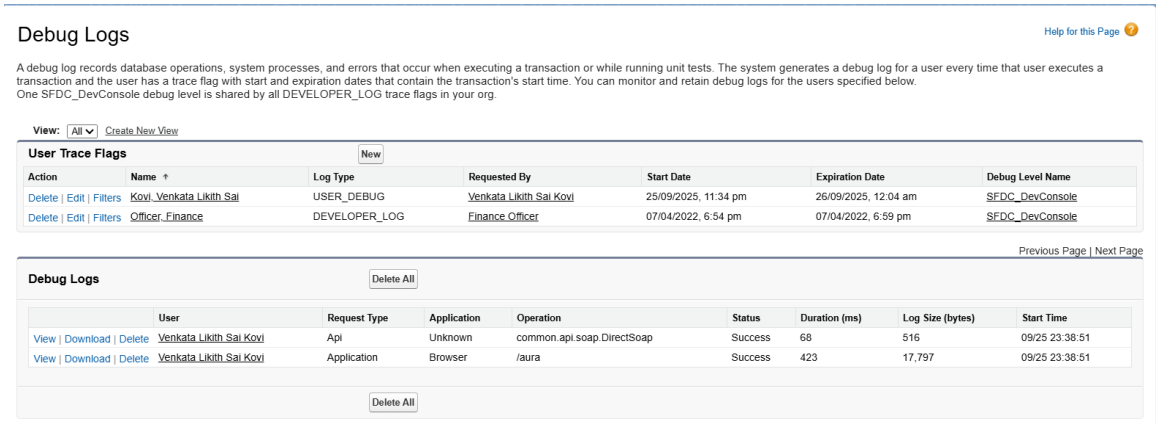
2. Debugging with Debug Logs

Step 1: In Setup → Debug Logs.

Step 2: Add yourself as a Monitored User.

Step 3: Run a test or perform actions (like creating a Donation record).

Step 4: Open the generated log → filter for USER_DEBUG to check values.



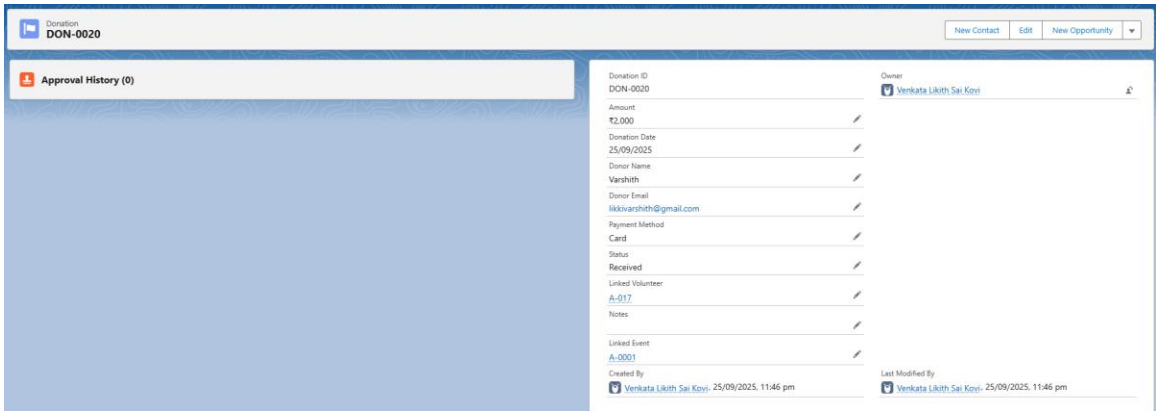
3. Validation of Automation

This process involves manually testing the key features you've built to ensure they work as a user would expect them to.

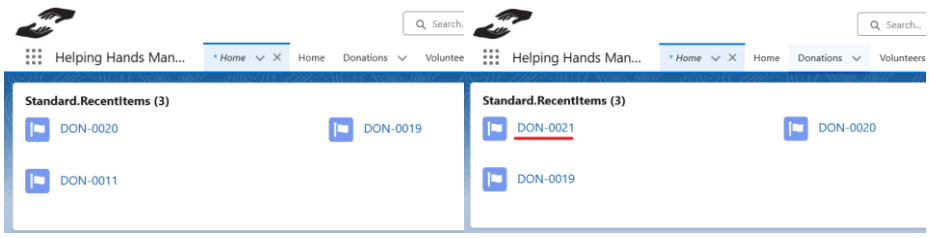
Test 1: Flow & Lightning Component

This test validates that creating a donation correctly updates the summary object and that the Lightning component shows the new data.

- **Action:** Navigate to the Donations tab and create a new **Donation record**. Fill in all the necessary details, such as the amount and related account.
- **What to Check:**
    - Go to the **Donation Summary** object and find the relevant summary record. Verify that its values have been correctly updated to include your new donation.



    - Open the page where the **Donation Summary Card** Lightning component is located and check that it displays the updated values.

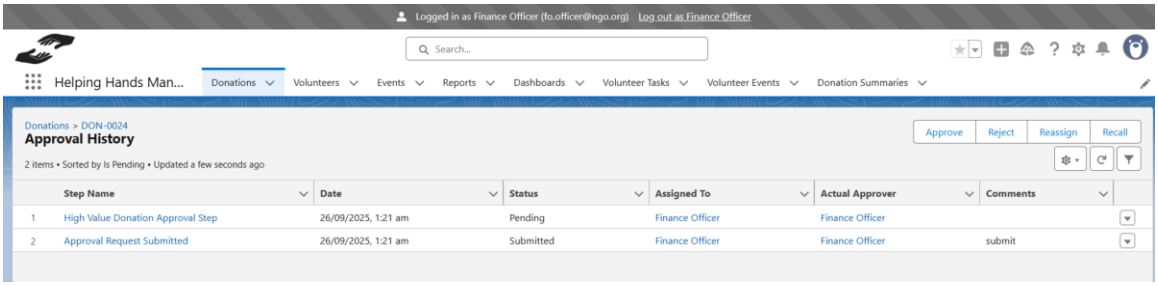


Donation Summary record after your test donation.

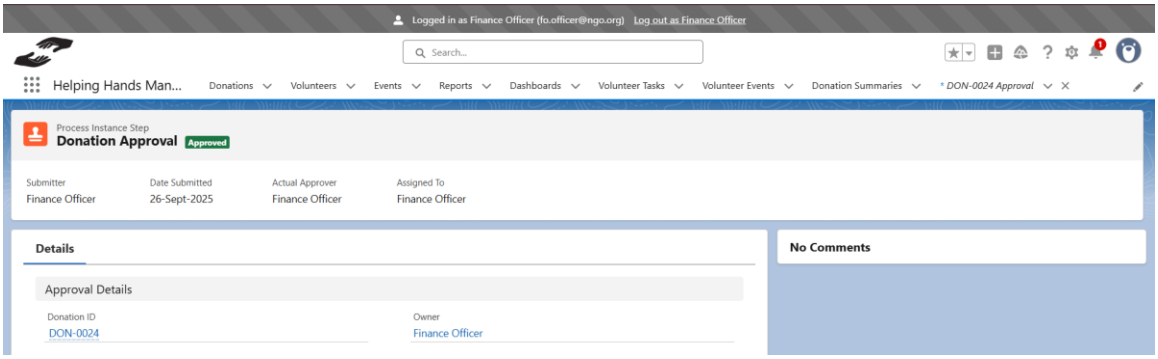
Test 2: Approval Process

This test ensures that the approval workflow functions correctly.

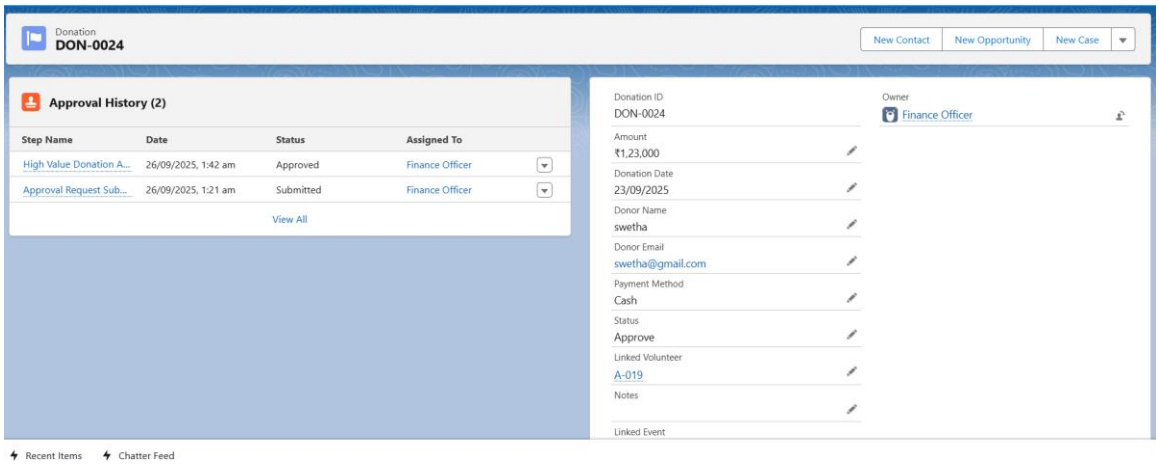
- **Action:** Find a Donation record that meets the criteria for your approval process. Submit it for approval. Then, as the designated approver, find the approval request and **approve the Donation**.



- **What to Check:** After approving, view the Donation record and confirm that its status or relevant fields have been updated to reflect the approval.



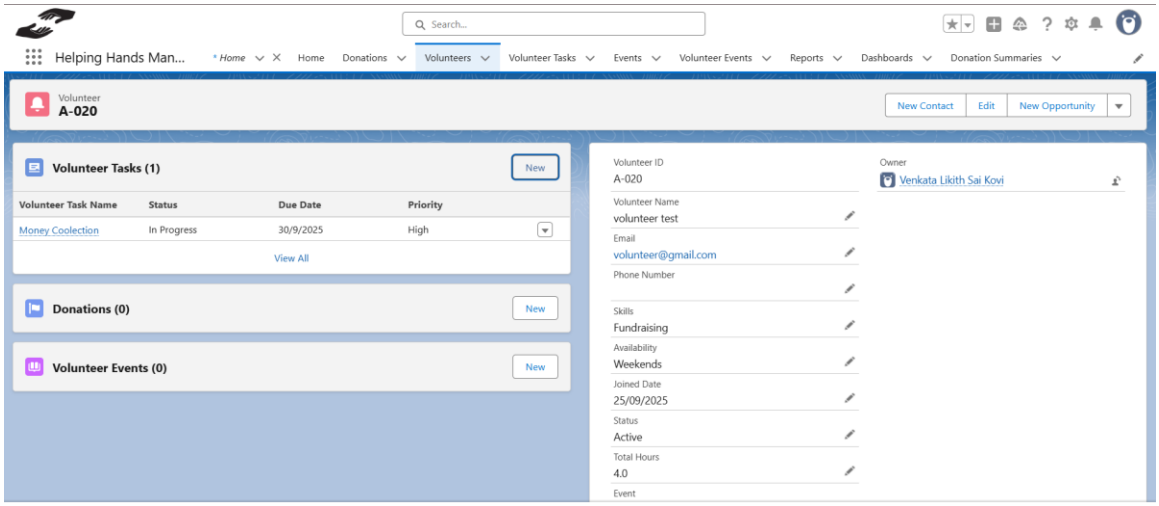
Result at User's Side:-



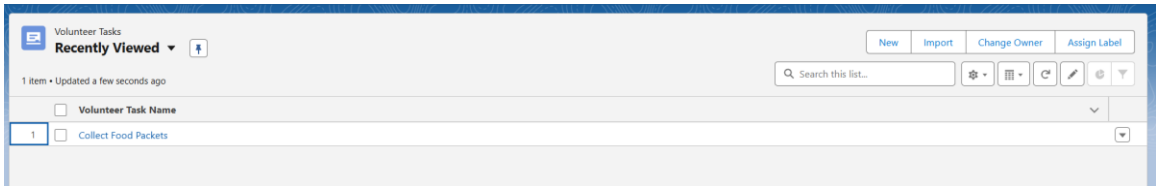
Test 3: Trigger Logic

This test verifies that your trigger's cleanup logic is working when a record is deleted.

- **Action:** First, create a sample Volunteer record and ensure it has related records (like Volunteer Tasks). Then, go to that Volunteer record and **delete it**.

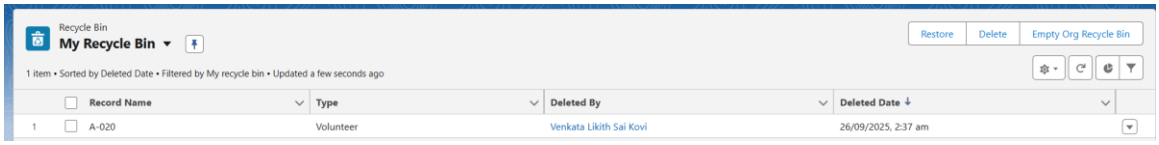


- **What to Check:** After deletion, confirm that the trigger's cleanup logic has run successfully. For example, check to see if the related Volunteer Task records were also deleted or reassigned as expected



The volunteer task has also been deleted after the deletion of volunteer A-020.

Recycle Bin Check:-

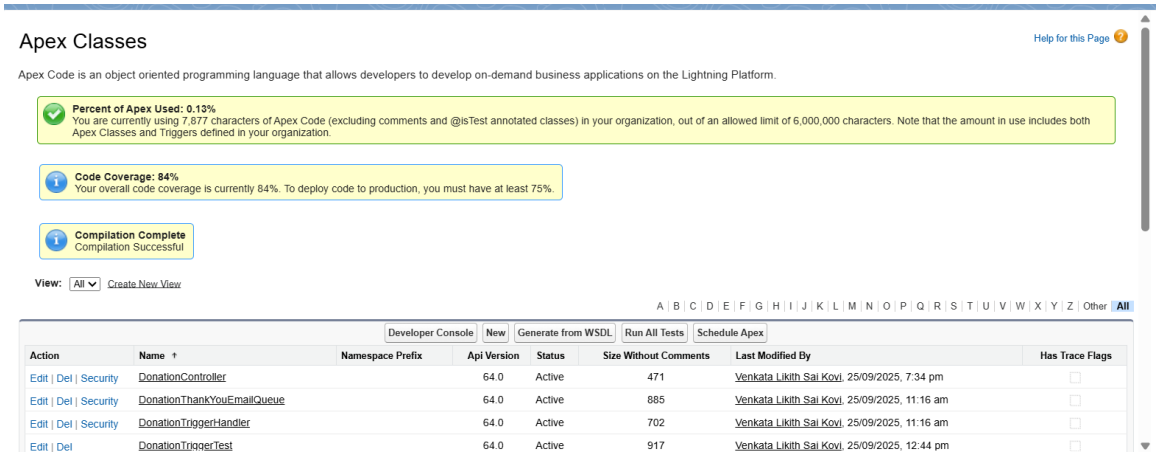


4. Code Coverage Requirement

Salesforce requires ≥75% code coverage for deployment.

Step 1: Go to Setup → Apex Classes → Code Coverage.

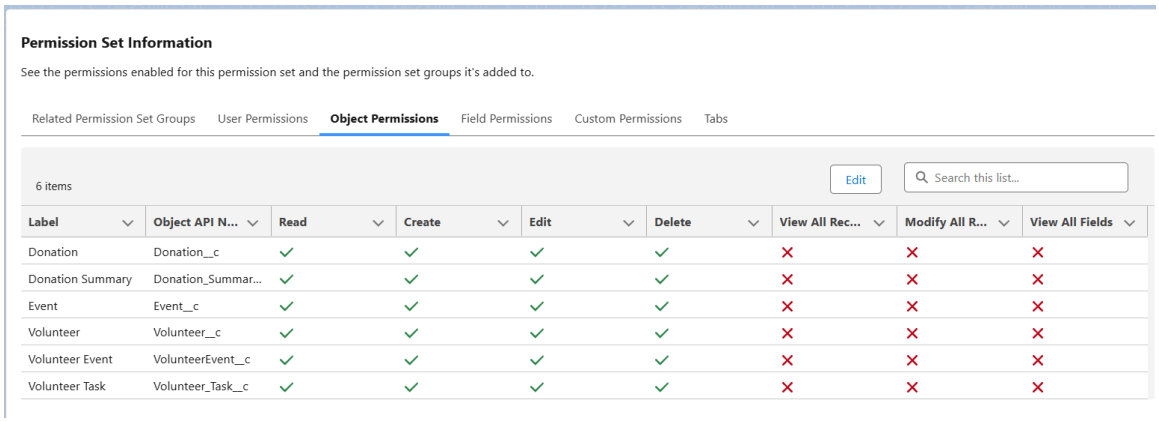
Step 2: Ensure each trigger/class has coverage.



5. Post-Deployment Steps

Step 1: Assign Permissions:
- Create a HelpingHands User Permission Set.
- Add CRUD access to all HelpingHands objects.

Step 2: Test with a normal user:
- Log in as them → check if they can Create/Update Donation.



Permission Set Information

See the permissions enabled for this permission set and the permission set groups it's added to.

Related Permission Set Groups | User Permissions | **Object Permissions** | Field Permissions | Custom Permissions | Tabs

6 items — Edit — Search this list...

Label	Object API N...	Read	Create	Edit	Delete	View All Rec...	Modify All R...	View All Fields
Donation	Donation__c	✓	✓	✓	✓	✗	✗	✗
Donation Summary	Donation_Summar...	✓	✓	✓	✓	✗	✗	✗
Event	Event__c	✓	✓	✓	✓	✗	✗	✗
Volunteer	Volunteer__c	✓	✓	✓	✓	✗	✗	✗
Volunteer Event	VolunteerEvent__c	✓	✓	✓	✓	✗	✗	✗
Volunteer Task	Volunteer_Task__c	✓	✓	✓	✓	✗	✗	✗

6. Data Import Wizard

The Data Import Wizard is a tool built directly into Salesforce Setup. It's designed for simple data imports of up to 50,000 records. Think of it as the perfect tool for a non-technical user, like a manager, who needs to upload a list of new leads or contacts from a spreadsheet.

Importing New Donations

Your task is to import a small list of new Donation records using the Data Import Wizard.

Step 1: Create Your CSV File

First, you need a data file to import. Open a plain text editor (like Notepad or TextEdit) and paste the following text. Save the file on your computer with the name donations.csv.

Code snippet

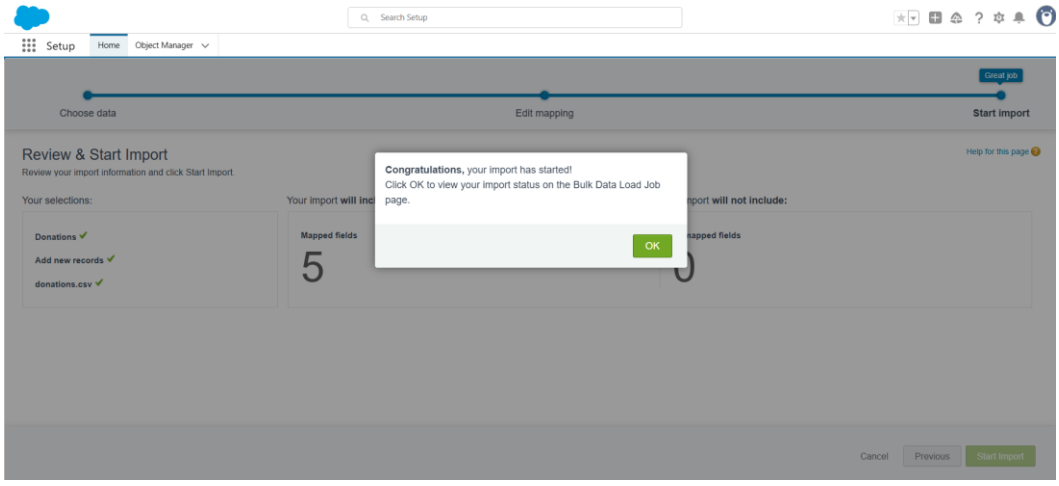
```
Donor_Name__c,Donor_Email__c,Amount__c,Payment_Method__c,Status__c

Alice Johnson,alice.j@example.com,250,Card,Received

Bob Williams,bob.w@example.com,500,Cash,Received

Charlie Brown,charlie.b@example.com,75,Online,Pledged
```

Note: The column headers in the CSV must exactly match the API names of your fields on the Donation object.

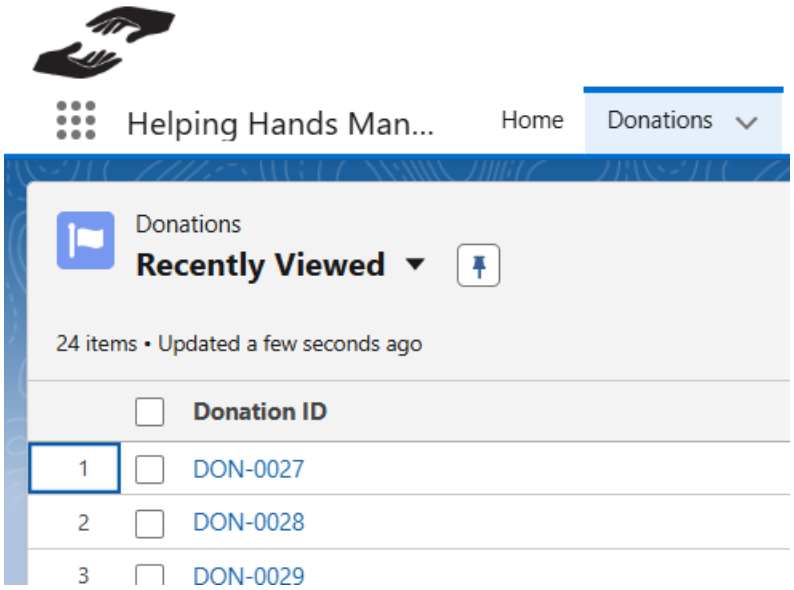
Step 2: Use the Data Import Wizard

1. Go to Setup and type Data Import Wizard in the Quick Find box.
2. Click Launch Wizard!
3. On the first screen, choose the Donations custom object.
4. Choose to Add new records. You can leave the other options as they are.
5. Upload your donations.csv file.
6. On the next screen, Salesforce will ask you to map your columns. It should automatically match them, but ensure that Donor_Name__c from your CSV maps to the Donor Name field in Salesforce, and so on.
7. Click Next, then Start Import.

You'll receive an email when the import is complete. Go to your Donations tab to see the three new records.



Results

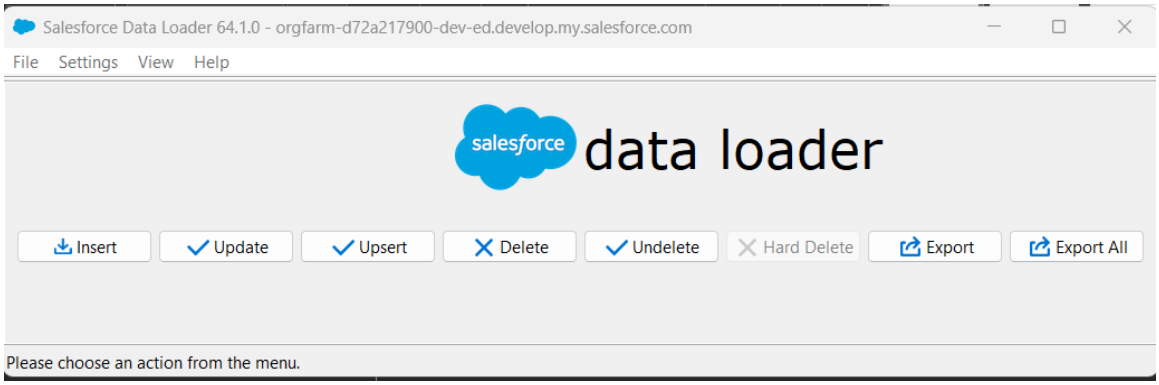


7. Data Loader

Step 1: Install Data Loader

Data Loader is a program you need to download and install on your computer.

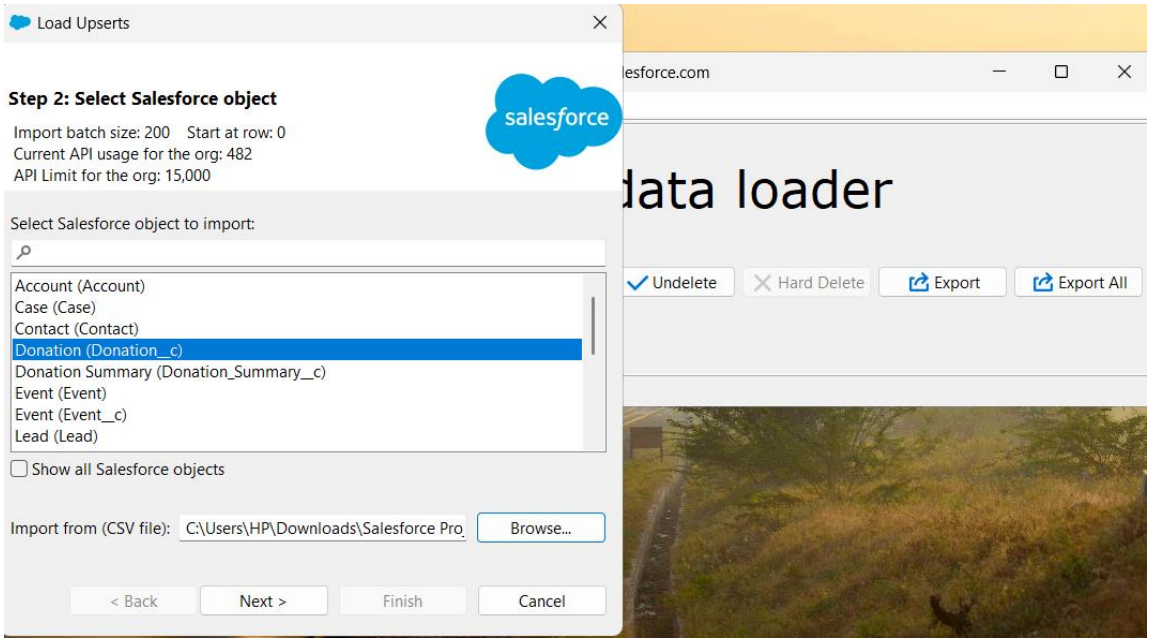
1. In Salesforce Setup, use the Quick Find box to search for and open Data Loader.
2. On this page, you'll find links to download the installer for Windows or macOS.
3. Download the correct version for your computer and follow the on-screen instructions to install it. You may also need to install a Java runtime if prompted.



Step 2: Perform the Upsert

Once Data Loader is installed, you can perform the upsert operation.

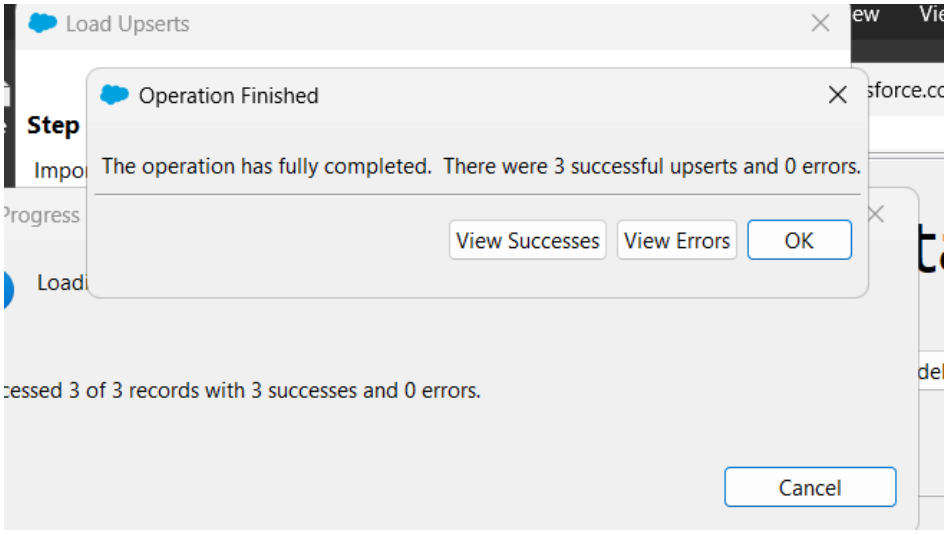
1. Open Data Loader and click Upsert.
2. Log in:
    - Environment: Production (Developer orgs use this)
    - Click Log in and use the username and password for your project org.
3. Choose SObject and CSV:
    - Select the Donation (Donation__c) object.
    - Browse for and select your donations_upsert.csv file.



4. Map External ID: For an upsert, Data Loader needs a way to match existing records. Since you don't have an External ID field, you'll use the Salesforce ID itself. Select the Id field from the list.

5. Map Fields: On the mapping screen, click Create or Edit a Map. Drag and drop the field names from the top box to the bottom box to match your CSV columns to the Salesforce fields.

6. Run the Operation: Click Next, choose a directory to save the success and error files, and click Finish.



ID	Donor_Name__c	Donor_Email__c	Amount__c	Payment_Method__c	Status__c	STATUS
a02gK000005pyQfQAI	Alice Johnson	alice.j@example.com	250	Card	Received	Item Created
a02gK000005pyQgQAI	Bob Williams	bob.w@example.com	500	Cash	Received	Item Created
a02gK000005pyQhQAI	Charlie Brown	charlie.b@example.com	75	Online	Pledged	Item Created

8. Duplicate Rules

After importing a lot of data, one of the biggest challenges is preventing duplicate records. Duplicate Rules are Salesforce's built-in tool to automatically catch and block duplicates as they are being created.

This feature works in two parts:

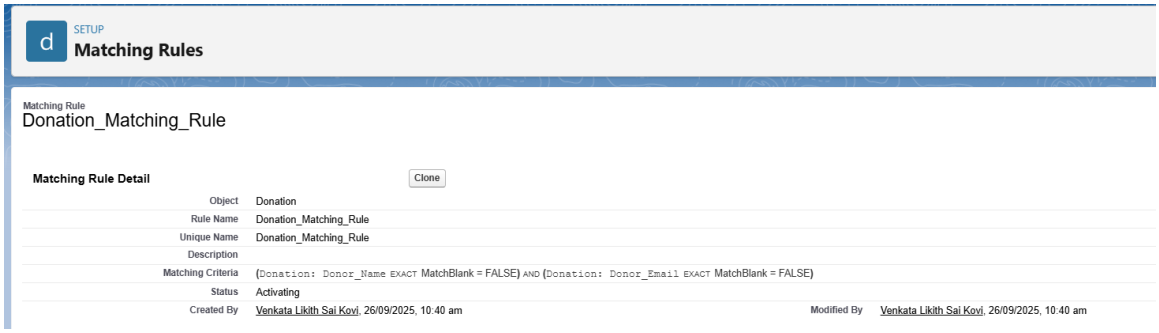
1. **Matching Rule**: Defines *what counts* as a duplicate (e.g., two records with the same email).
2. **Duplicate Rule**: Defines *what happens* when a duplicate is found (e.g., block the user or just show a warning).

Prevent Duplicate Donations

Our goal is to create a rule that blocks users from creating a new Donation if another Donation with the same **Donor Email** and **Donation Date** already exists.

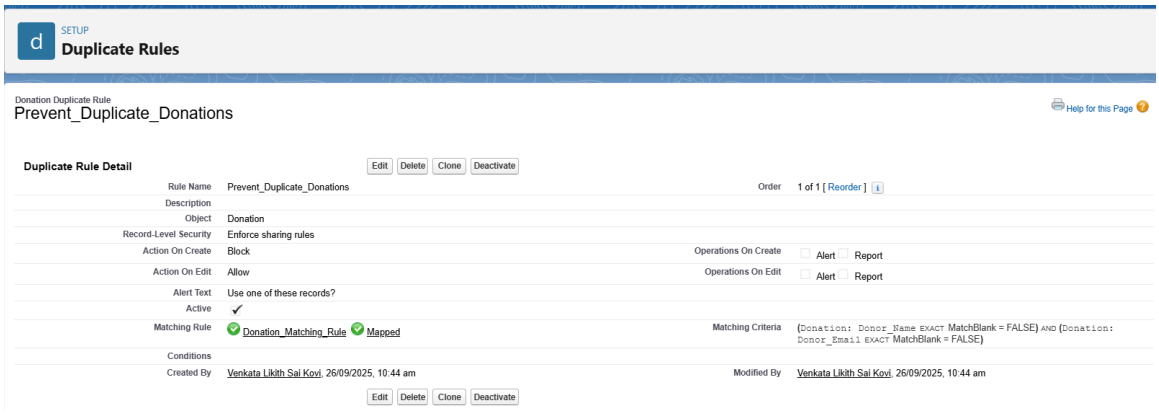
Step 1: Create the Matching Rule

1. Go to **Setup** and search for **Matching Rules**.
2. Click **New Rule** and select the **Donation** object.
3. Name the rule Donation_Matching_Rule.
4. For the criteria, select the **Donor Email** and **Donation Date** fields. Set the Matching Method for both to **Exact**.
5. **Save** and **Activate** the rule.



Step 2: Create the Duplicate Rule

1. Go to **Setup** and search for **Duplicate Rules**.
2. Click **New Rule** and select the **Donation** object.
3. Name the rule Prevent_Duplicate_Donations.
4. Under "Matching Rule," select the Donation_Matching_Rule you just created.
5. Under "Actions," for the "Action on Create" option, select **Block**.
6. **Save** and **Activate** the rule.



9. Data Export and Backup

Just as it's important to get data into Salesforce, you also need ways to get it out for reporting, archiving, or backup purposes. Salesforce provides two primary methods for this.

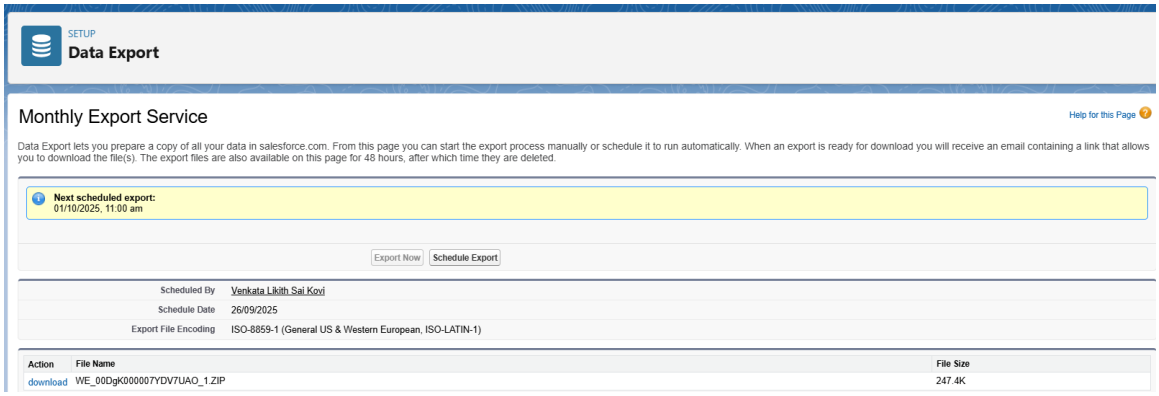
Method 1: Data Export Service (Weekly/Monthly Backup)

This is a built-in service for creating a complete backup of your org's data. You can schedule it to run automatically every week or month.

Practice:

1. Go to **Setup** and search for **Data Export**.

2. Click **Export Now**. (If that's not available, click **Schedule Export**).
3. You can leave the default settings to "Include all data".
4. Click **Start Export**.



Method 2: Data Loader (Specific Export)

You use the same Data Loader application to export specific data using a query. This is useful when you need a particular set of records right away.

Practice:

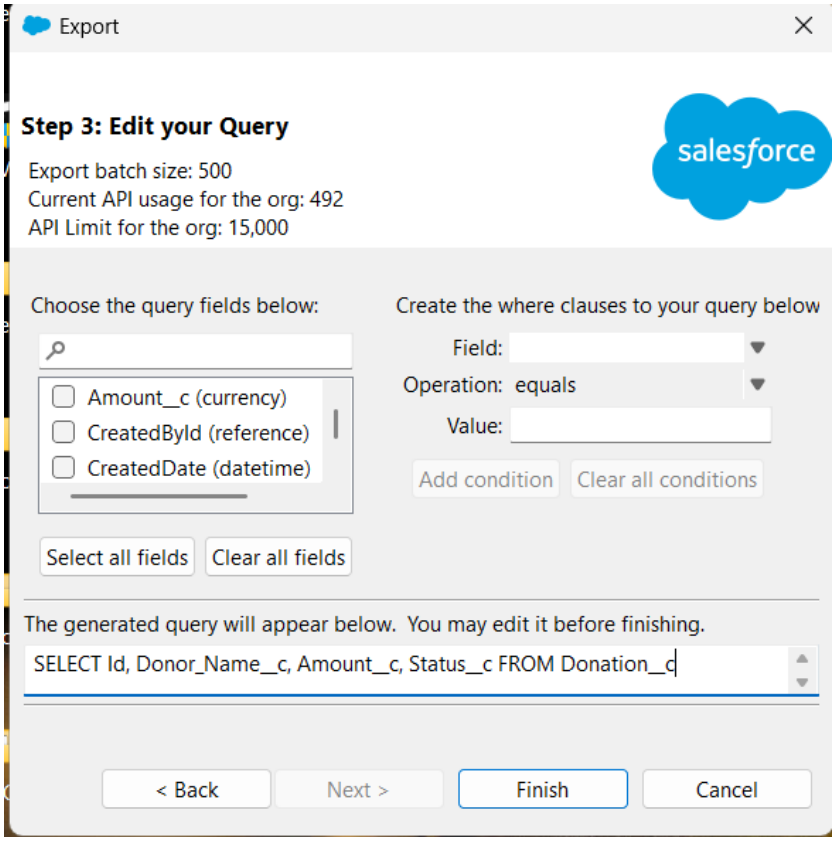
1. Open **Data Loader** and click **Export**.
2. Log in and select the **Donation (Donation__c)** object.
3. Choose a name and location for the export file (e.g., donations_export.csv).
4. Data Loader will now show you a query editor. You can write a SOQL query here to select exactly what you want. Copy and paste the following query:

SQL

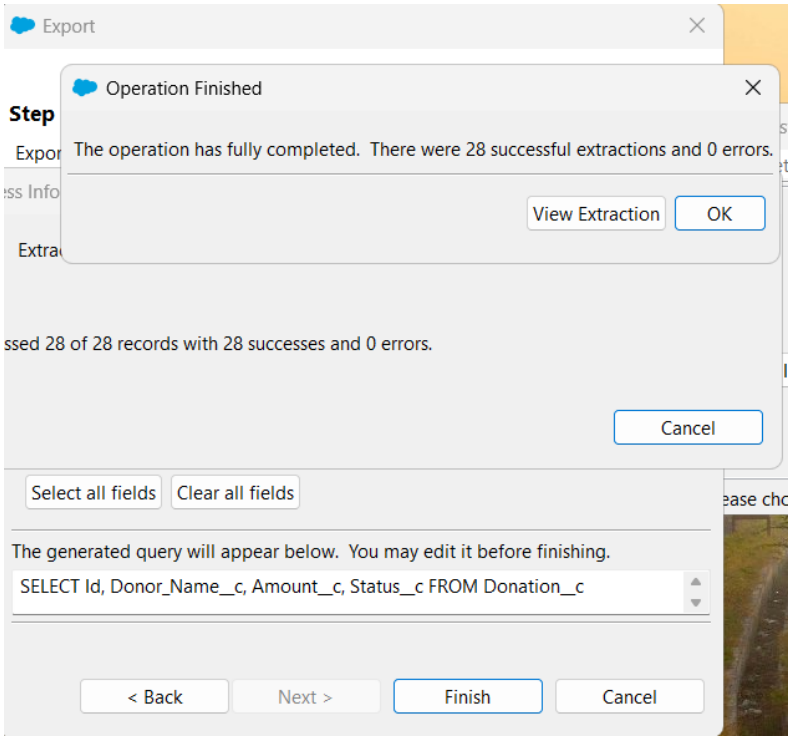
```
SELECT Id, Donor_Name__c, Amount__c, Status__c FROM Donation__c
```

5. Click **Finish** and then **Yes**.

The operation will run, and the donations_export.csv file will be saved to your computer. Open it to see the data you exported


Export
Step 3: Edit your Query
Export batch size: 500
Current API usage for the org: 492
API Limit for the org: 15,000
salesforce
Choose the query fields below:
Amount__c (currency)
CreatedById (reference)
CreatedDate (datetime)
Select all fields
Clear all fields
Create the where clauses to your query below
Field:
Operation: equals
Value:
Add condition
Clear all conditions
The generated query will appear below. You may edit it before finishing.
SELECT Id, Donor_Name__c, Amount__c, Status__c FROM Donation__c
< Back
Next >
Finish
Cancel

Result:-


Export
Operation Finished
The operation has fully completed. There were 28 successful extractions and 0 errors.
View Extraction
OK
ssed 28 of 28 records with 28 successes and 0 errors.
Cancel
Select all fields
Clear all fields
The generated query will appear below. You may edit it before finishing.
SELECT Id, Donor_Name__c, Amount__c, Status__c FROM Donation__c
< Back
Next >
Finish
Cancel

10. Packages

Unmanaged vs. Managed Packages

While Change Sets are used to move metadata between your *own connected orgs*, Packages are used to bundle and distribute your customizations to *other, unrelated* Salesforce orgs.

If a Change Set is like an internal company mail envelope sent between two departments, a Package is like a product you put in a box to ship to any customer in the world.

There are two main types of packages.

Unmanaged Packages

Think of these as an open-source template.

- **Purpose**: Best for one-time distributions or for sharing code templates.
- **Key Feature**: Once an unmanaged package is installed, all of its components (objects, code, etc.) are fully unlocked and can be edited by the administrator of that org. They essentially become part of the new org.
- **Upgrades**: They cannot be upgraded. To release a new version, the installer has to uninstall the old package and install the new one.
- **Use Case**: You create a set of useful reports and dashboards and want to share them with another department to use as a starting point.

Managed Packages

Think of these as a professional, sealed software product, like an app you'd buy.

- **Purpose**: Used by Salesforce partners to sell and distribute applications on the AppExchange.
- **Key Feature**: The components are "locked" and cannot be viewed or edited by the installer. This protects the developer's intellectual property.
- **Upgrades**: They are fully upgradeable. The developer can push new versions and bug fixes to customers who have the package installed.
- **Use Case**: You install an application like DocuSign or a financial calculator from the AppExchange.

11. ANT Migration Tool

The Ant Migration Tool is a command-line utility for moving metadata between your local computer and a Salesforce org. It's a step up from the point-and-click interface of Change Sets and requires a more technical setup.

Think of it this way: if Change Sets are like mailing a package at the post office, the Ant Migration Tool is like using a professional shipping service where you define everything in a manifest (package.xml) and use commands to script the entire process.

The High-Level Process

1. **Define package.xml**: You create an XML file that lists every single component you want to move (e.g., ApexClass, CustomObject, Flow).
2. **Retrieve**: You run a command in your computer's terminal to **retrieve** the metadata from a Salesforce org. This downloads all the components as source files into a folder on your computer.
3. **Deploy**: You then run another command to **deploy** those files from your computer to a different target org.

Why Use It?

It's more powerful than Change Sets because it's scriptable, repeatable, and can be integrated into automated systems (CI/CD). For many years, this was the primary tool for Salesforce developers.

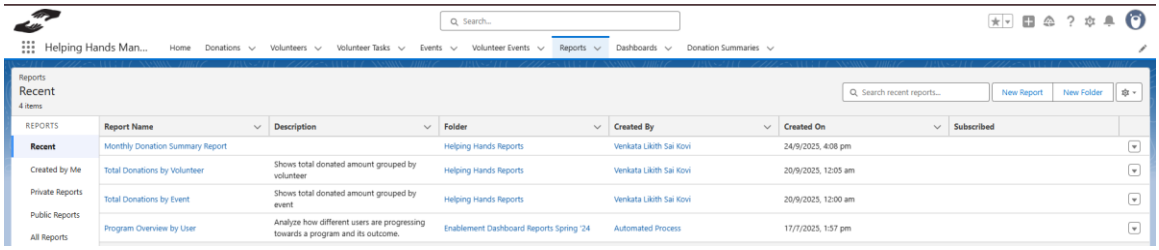
While the Ant Migration Tool is still used in some cases, it has been largely replaced by the more modern **VS Code & SFDX**

🛬 Phase 9: Reporting, Dashboards & Security Review

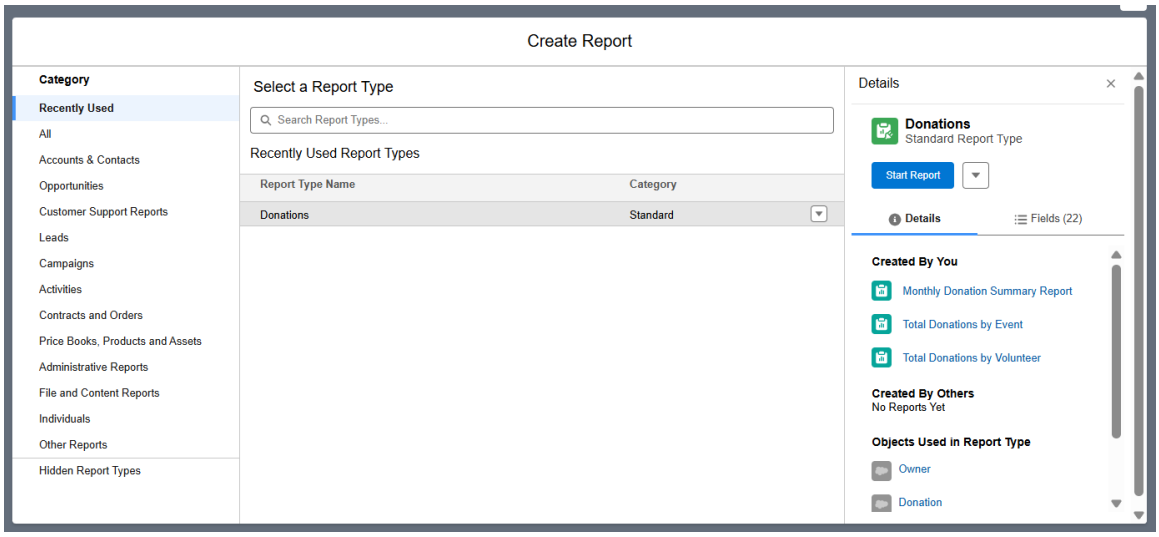
This phase focuses on building meaningful reports and dashboards for the Helping Hands Foundation application and performing a final security review of the Salesforce org.

Step 1: Create Reports

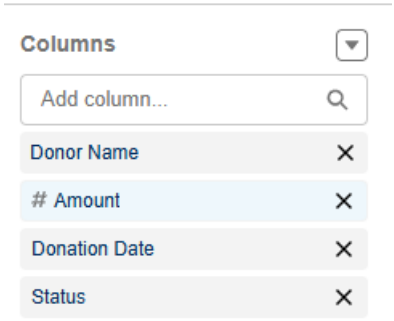
1. Navigate to Reports → App Launcher → Search 'Reports' → Open Reports tab.



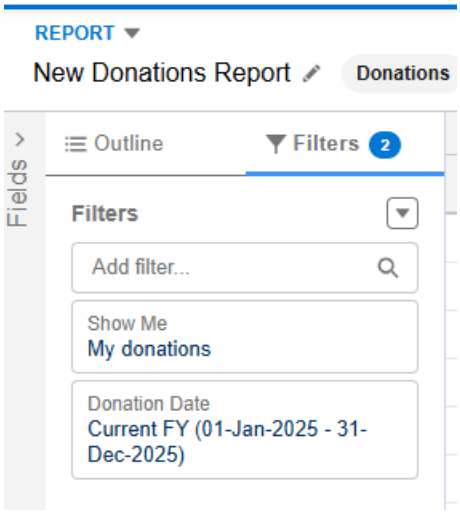
2. Create a New Report → Select object: Donations → Continue.



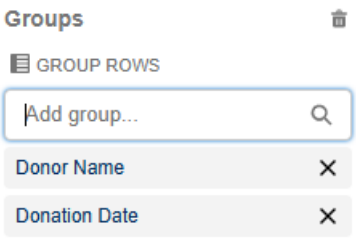
3. Add Columns → Donor Name, Donation Amount, Donation Date, Payment Status.



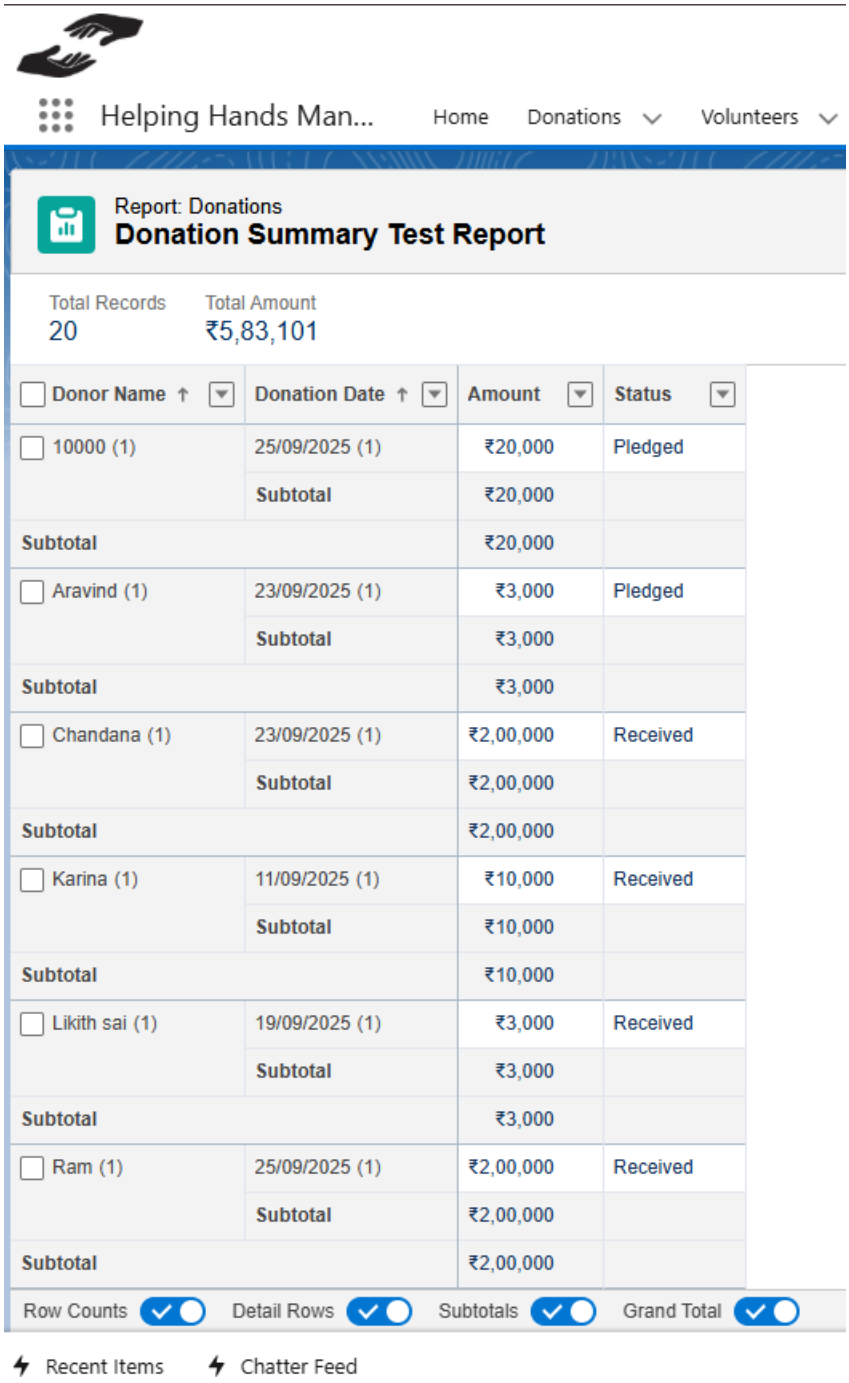
4. Apply Filters → Donation Date = Current Fiscal Year.



5. Group Records → Group by Donor Name, Group by Month (Donation Date).

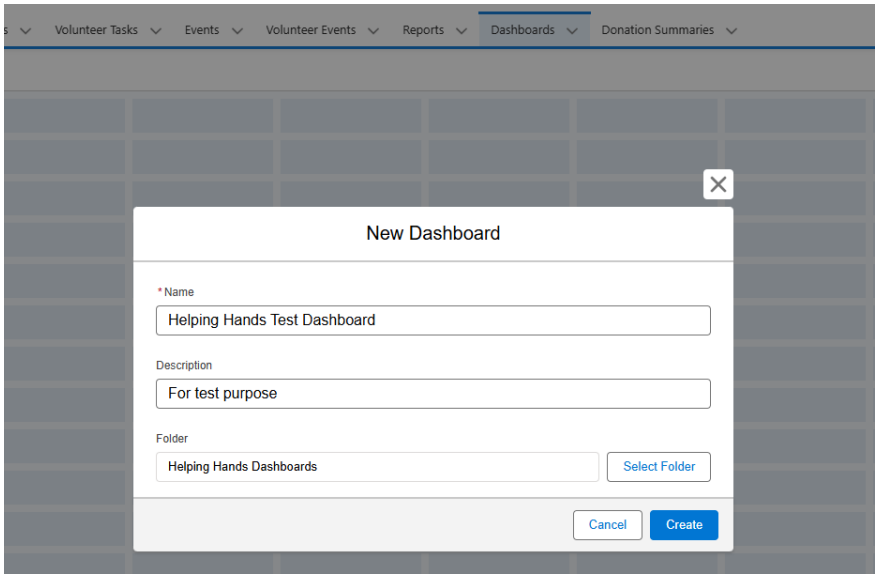


6. Save & Run Report → Name: Donation Summary Test Report → Save to Helping Hands Reports Folder.



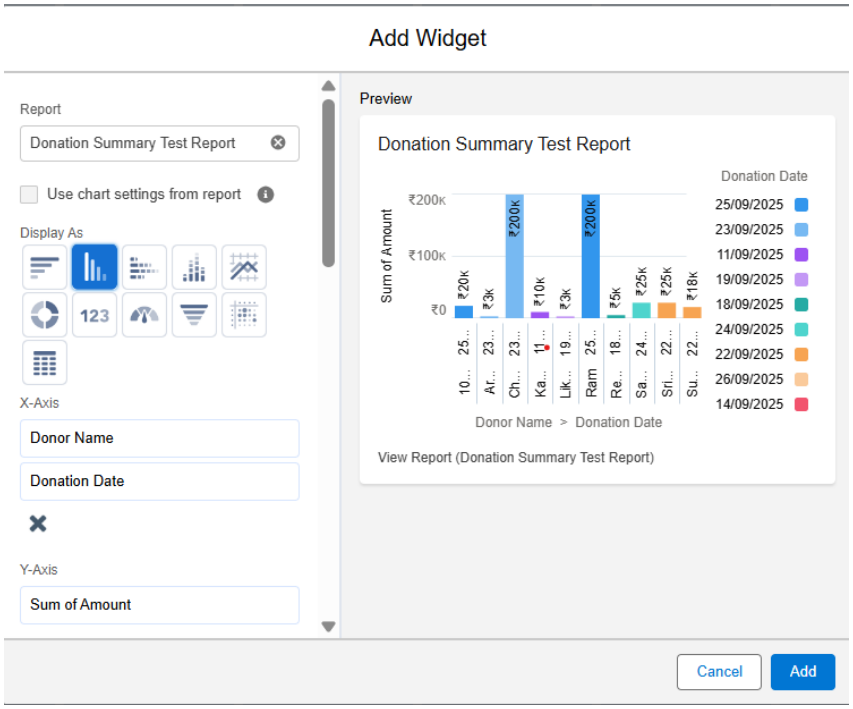
Step 2: Create Dashboard

1. Navigate to Dashboards tab → New Dashboard → Name: Helping Hands Test Dashboard.

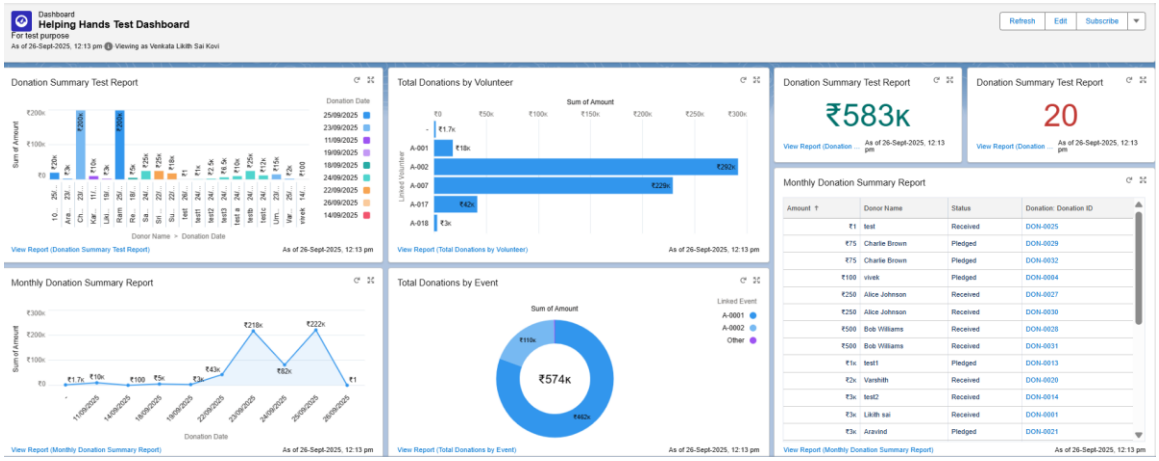


2. Add Components → Link reports to visualizations:
   - Donations by Month (Bar Chart)
   - Top Donors (Donut Chart)
   - Total Donations (Number widget)

3. Configure Dashboard → Running User = Financial Officer → Save & Refresh.

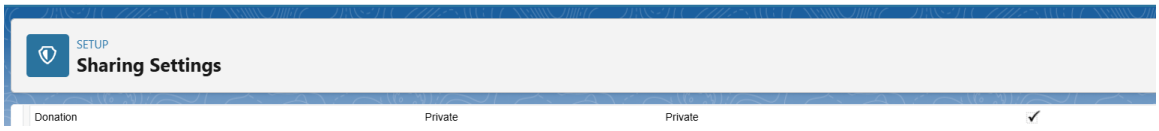


4. Create Dynamic Dashboard → Run as logged-in user.

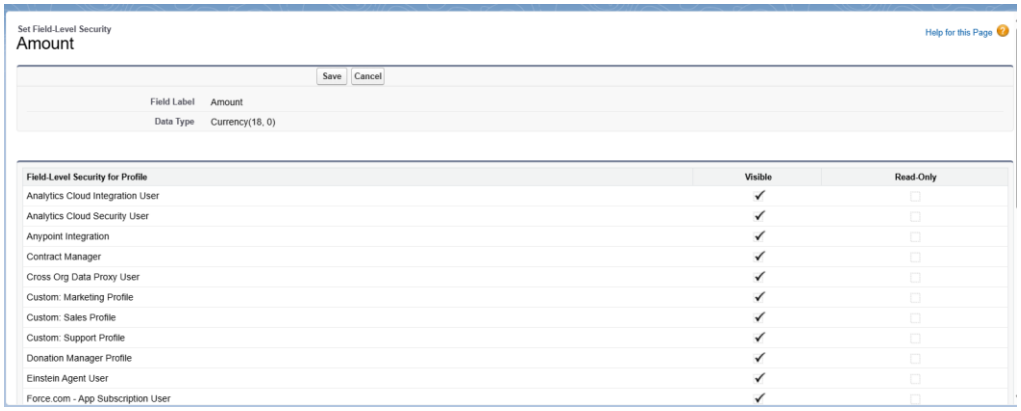


Step 3: Security Review

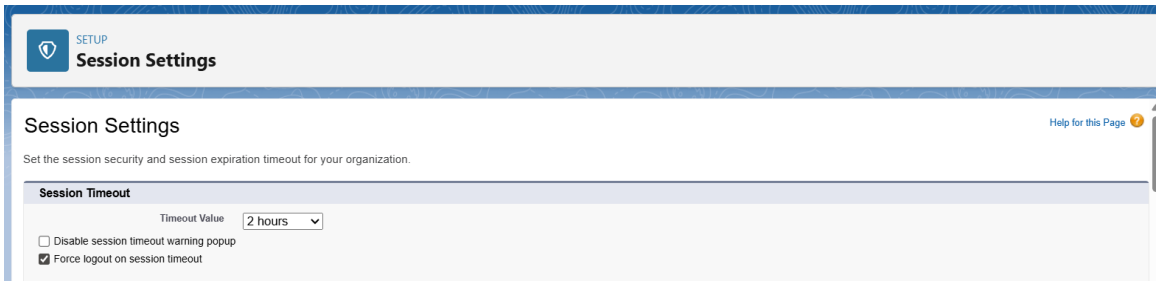
1. Sharing Settings : Setup → Sharing Settings → Ensure Donations = Private, Volunteer = Public Read Only.



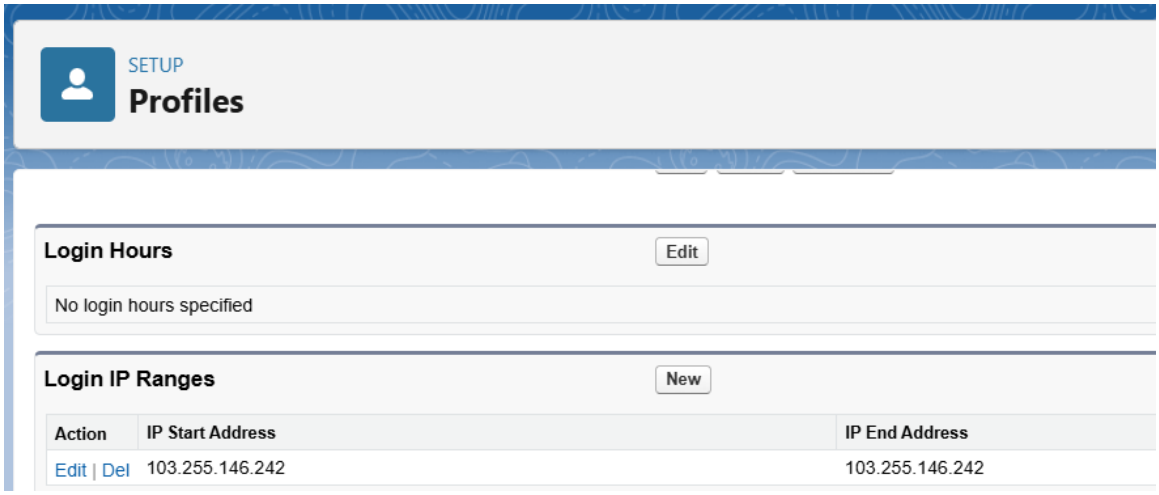
2. Field-Level Security : Setup → Object Manager → Fields → Set Field-Level Security for sensitive fields.



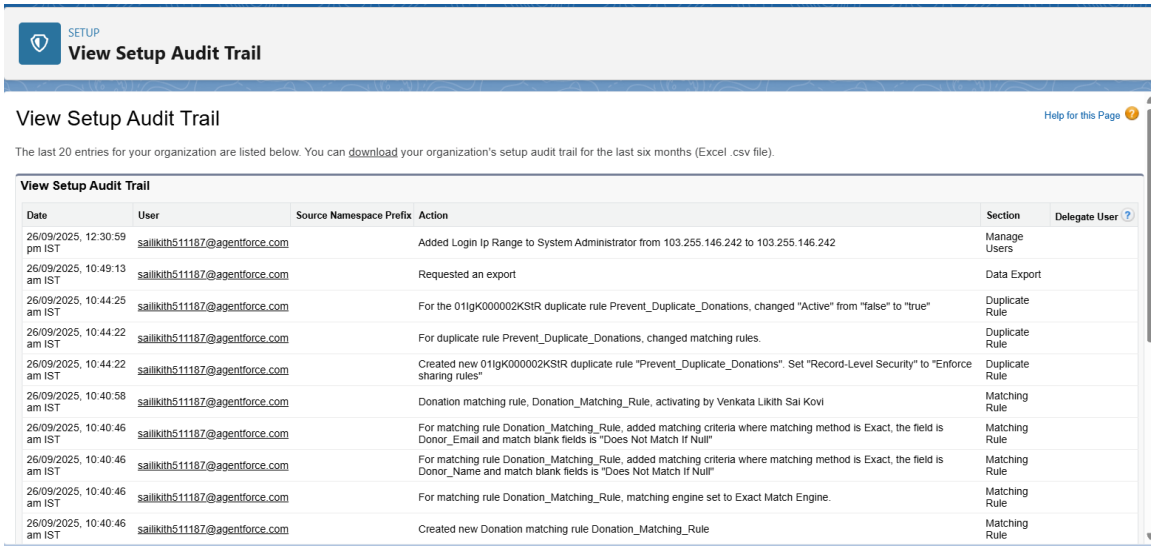
3. Session Settings : Setup → Session Settings → Timeout 2 hrs, Require HTTPS.



4. Login IP Ranges : Setup → Profiles → System Administrator → Add allowed IPs.

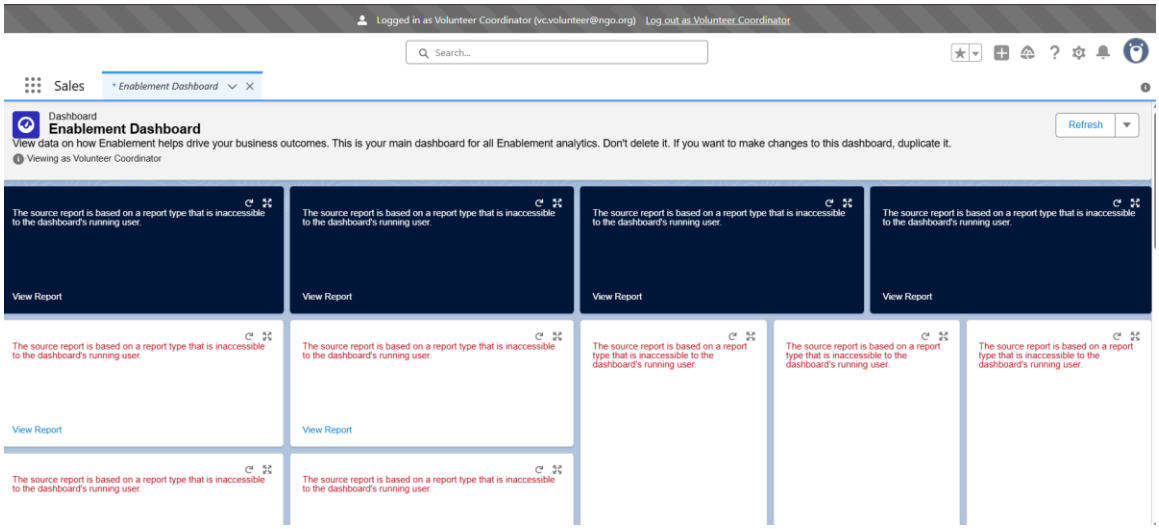


5. Audit Trail : Setup → View Setup Audit Trail → Review all admin actions.



Step 4: Testing Reports & Security

1. Login as Volunteer Coordinator → Open Dashboard → Check restricted data hidden.



2. Login as Financial Officer → Verify donation details are visible.