Phase 1: Problem Understanding & Industry Analysis

Requirement Gathering

- Institutions often face issues where **lost belongings remain unclaimed** due to lack of a structured reporting system.
- Users need a **centralized platform** to report lost or found items, search records, and get timely notifications.
- Admins/security staff need a way to **verify claims, manage submissions, and track resolution** efficiently.

Stakeholder Analysis

- **Students/Faculty/Staff (Users)** → Report lost/found items, search the portal, receive notifications.
- **Admin/Security Staff** → Manage lost/found records, verify ownership, close cases.
- **System (Salesforce Portal)** → Stores item details, auto-suggests matches, sends alerts, generates reports.

Business Process Mapping

1. User logs into the portal → reports a lost or found item with details.
2. System stores the entry and attempts to **match lost and found records**.
3. User receives notification/email if a possible match is found.
4. Admin verifies item ownership and marks case as resolved.
5. System generates reports → insights into frequently lost items, common locations, and resolution timelines.

Industry-Specific Use Case Analysis

- In large campuses (universities, corporate offices), **misplaced items are common** and recovery is inefficient.
- Existing solutions are either **manual (notice boards, registers)** or require **heavy custom development**.

- A **Salesforce CRM-based solution** provides a **cost-effective, scalable, and mobile-ready approach** with faster adoption.

AppExchange Exploration

- Similar models exist for **case management, resource booking, and asset tracking** on Salesforce AppExchange.
- These apps inspired the **Lost and Found Portal** model, customized to the needs of institutions.

Outcome of Phase 1

The problem, stakeholders, and scope of the **"Lost and Found Portal for Institutions"** are well-defined and aligned with real-world needs.
This ensures the solution is **relevant, practical, and scalable** within an institutional environment.

Phase 2: Org Setup & Configuration

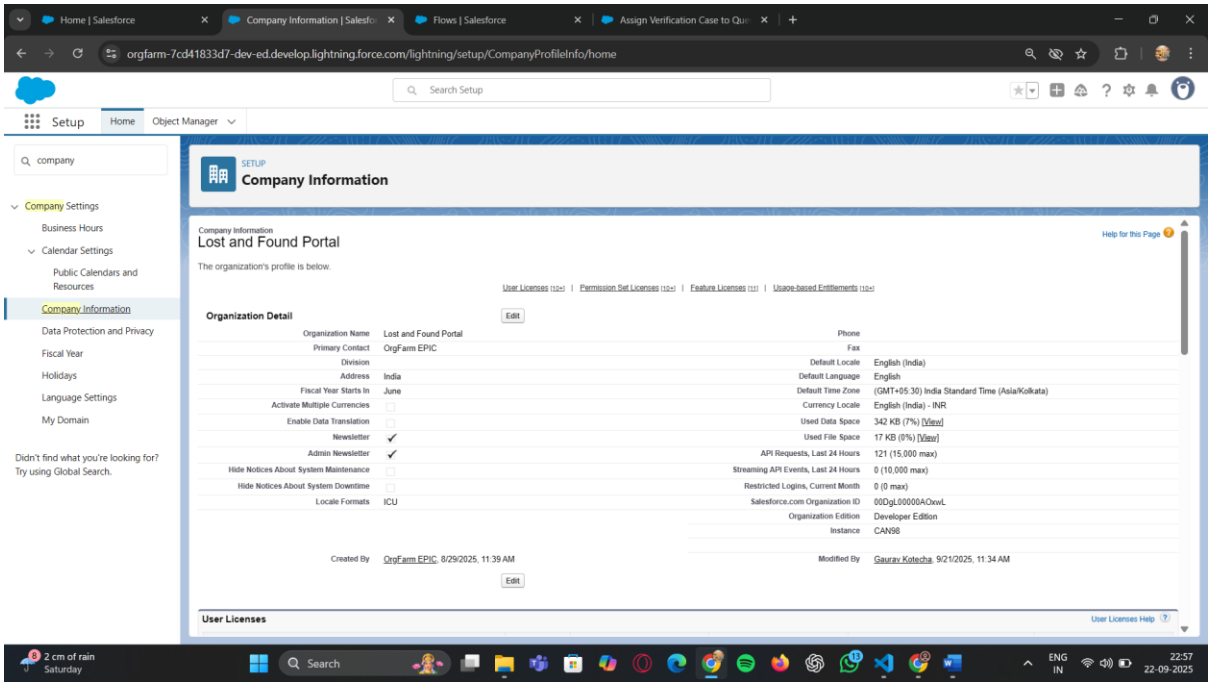
2.1 Introduction

The objective of Phase 2 was to establish the foundational Salesforce environment for the Lost and Found Portal. This involved setting up the company profile, creating a robust security and sharing model, and implementing the core automation to assign incoming cases to the appropriate team.

2.2 Company Profile & Org Defaults

The organization's basic details and default settings were configured to align with the project's operational context.

- **Salesforce Edition:** Enterprise Edition
- **Company Name:** Campus Lost & Found Services Pvt. Ltd.
- **Default Locale:** English (India)
- **Default Time Zone:** Asia/Kolkata (IST)
- **Default Currency:** INR

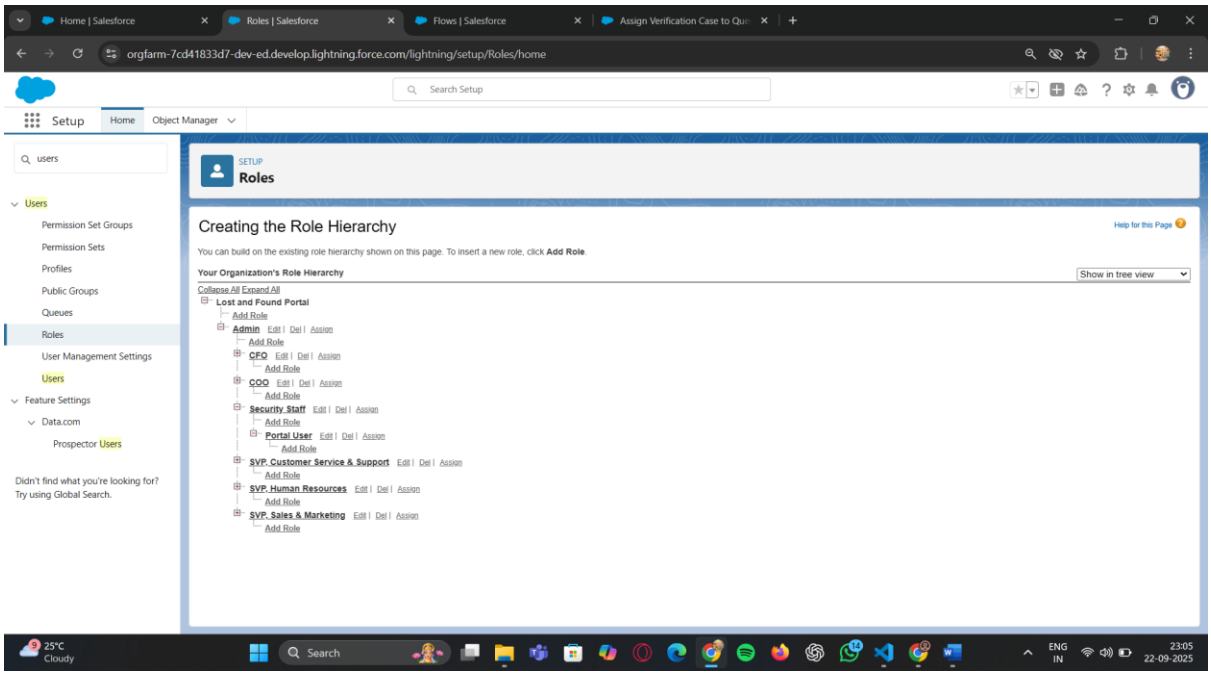


2.3 User & Security Model Setup

A security model was established to ensure users have appropriate access based on roles.

2.3.1 Roles

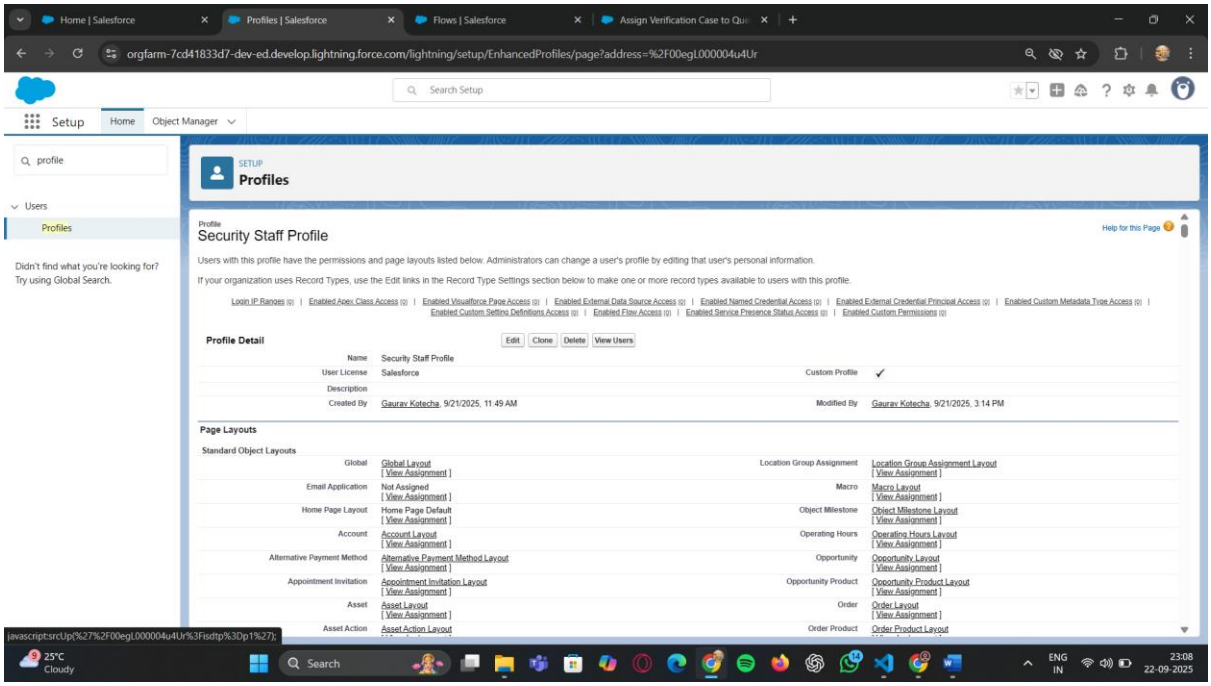
A role hierarchy was created to reflect the authority structure within the institution.



2.3.2 Profiles

A custom profile was created for the security team to grant them specific permissions to the Lost and Found objects.

- **Profile Name:** Security Staff Profile
- **Method:** Cloned from the standard "Standard User" profile.
- **Key Permissions:** Granted full Create, Read, Edit, and Delete (CRUD) access on the Lost Item, Found Item, and Verification Case objects.



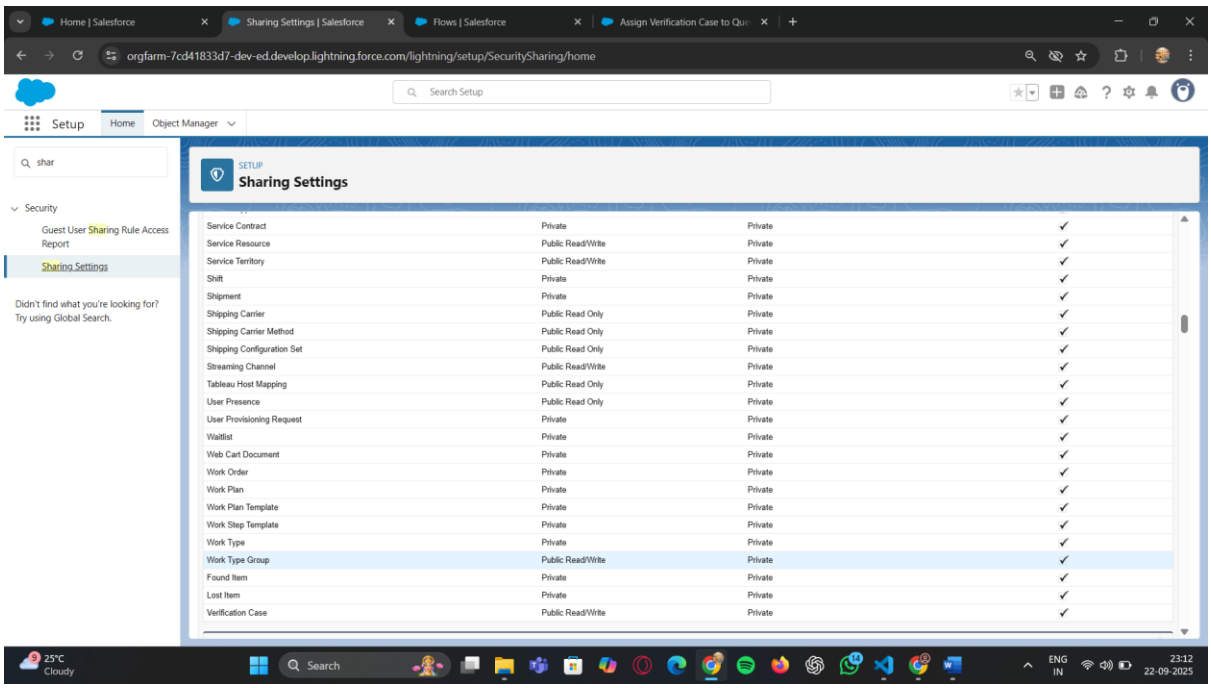
2.4 Data Sharing & Visibility

Default record visibility was set to private, with sharing rules created to grant access where neede

2.4.1 Organization-Wide Defaults (OWD)

The default internal access for the custom objects was set to the most restrictive level.

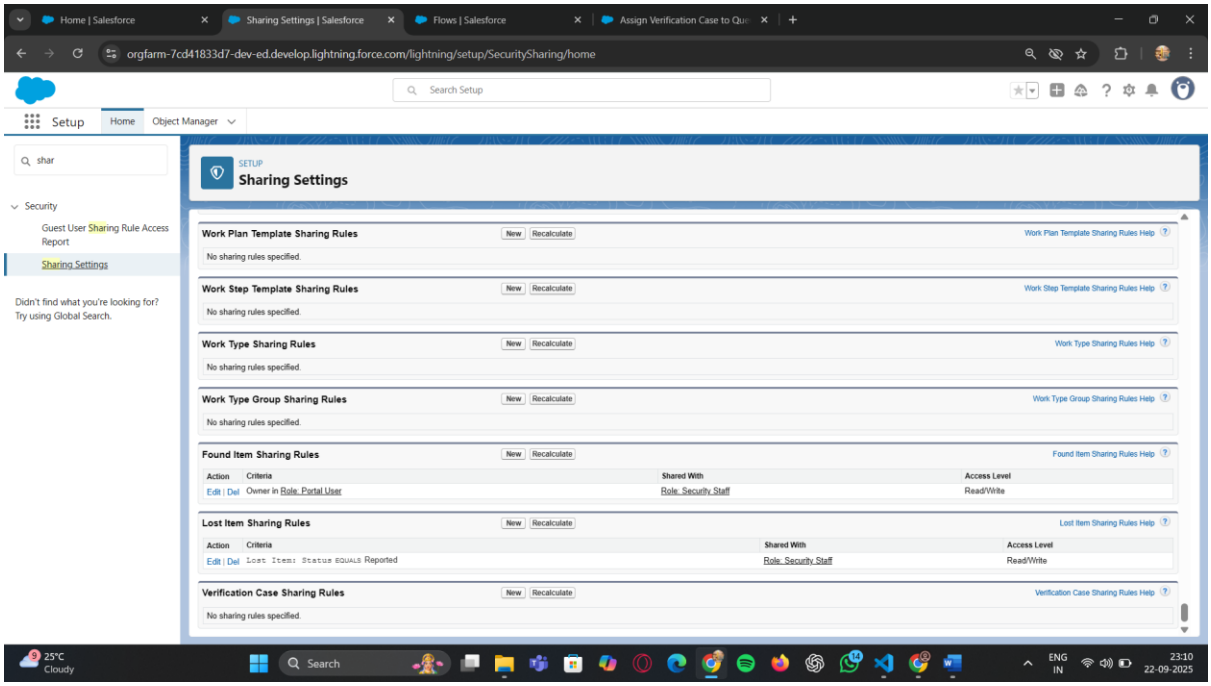
- **Lost Item:** Private
- **Found Item:** Private
- **Verification Case:** Private



2.4.2 Sharing Rules

Rules were created to grant the Security Staff access to records they need to manage.

- **Rule 1:** Shared Lost Item records with the Security Staff role when the Status field equals "Reported".
- **Rule 2:** Shared Found Item records owned by Portal Users with the Security Staff role.


Home | Salesforce
Sharing Settings | Salesforce
Flows | Salesforce
Assign Verification Case to Que
orgfarm-7cd41833d7-dev-ed.develop.lightning.force.com/lightning/setup/SecuritySharing/home
Search Setup
Setup
Home
Object Manager
shar
Security
Guest User Sharing Rule Access Report
Sharing Settings
Didn't find what you're looking for? Try using Global Search.
SETUP
Sharing Settings
Work Plan Template Sharing Rules
New
Recalculate
Work Plan Template Sharing Rules Help
No sharing rules specified.
Work Step Template Sharing Rules
Work Step Template Sharing Rules Help
No sharing rules specified.
Work Type Sharing Rules
Work Type Sharing Rules Help
No sharing rules specified.
Work Type Group Sharing Rules
Work Type Group Sharing Rules Help
No sharing rules specified.
Found Item Sharing Rules
Found Item Sharing Rules Help
Action
Criteria
Shared With
Access Level
Edit | Del
Owner in Role: Portal User
Role: Security Staff
Read/Write
Lost Item Sharing Rules
Lost Item Sharing Rules Help
Action
Criteria
Shared With
Access Level
Edit | Del
Lost Item: Status EQUALS Reported
Role: Security Staff
Read/Write
Verification Case Sharing Rules
Verification Case Sharing Rules Help
No sharing rules specified.
25°C
Cloudy
Search
ENG
IN
23:10
22-09-2025

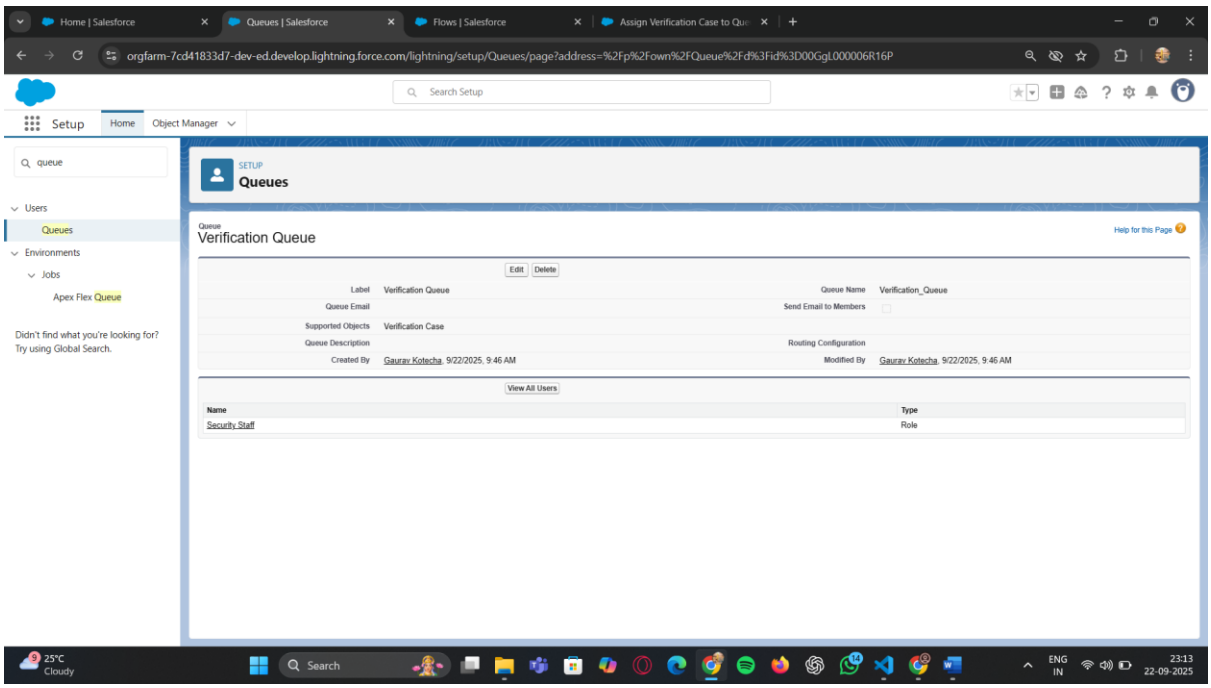
2.5 Automation Setup

Automation was built to streamline the case assignment process. Due to a platform bug, a workaround using Apex was required.

2.5.1 Queue

A queue was created to act as a central holding point for all new verification cases.

- **Queue Name:** Verification Queue
- **Supported Object:** Verification Case
- **Members:** Role: Security Staff



2.5.2 Invocable Apex (Workaround for Platform Bug)

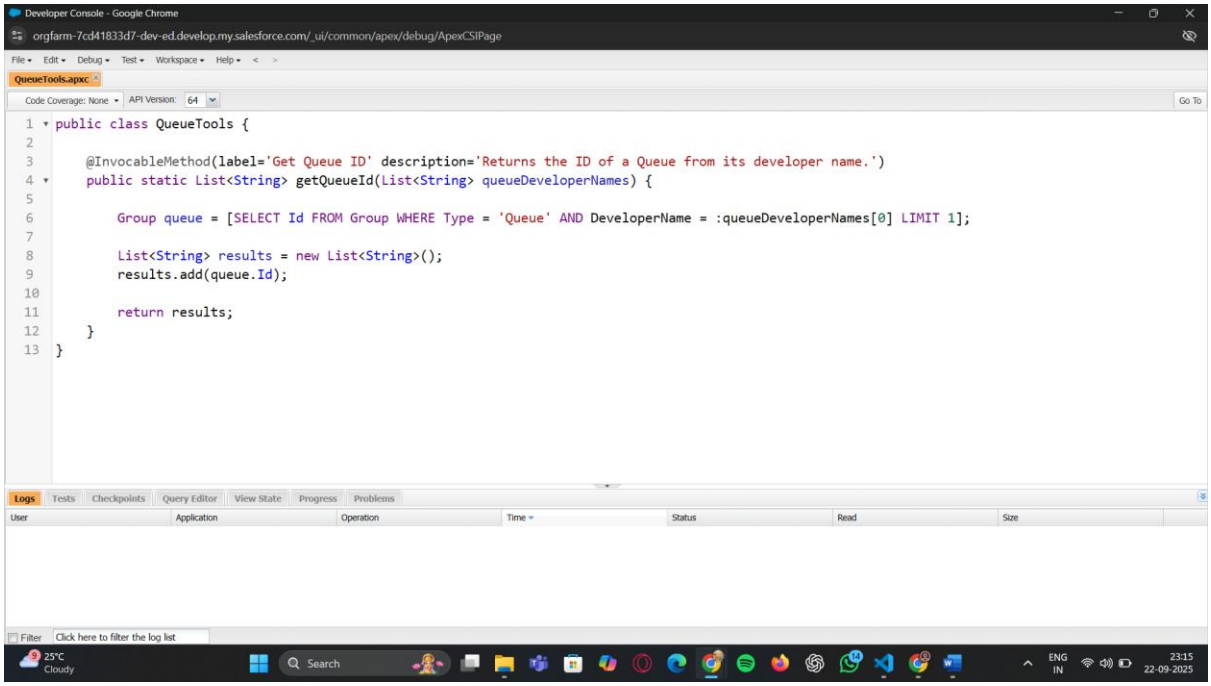
A bug in the Flow Builder prevented the direct selection of the Queue ID. To bypass this, a simple Apex class was created to find and return the Queue ID to the Flow.

- **Apex Class Name:** QueueTools
- **Code:**

Java

```
public class QueueTools {

    @InvocableMethod(label='Get Queue ID' description='Returns the ID of a Queue from its developer name.')

    public static List<String> getQueueId(List<String> queueDeveloperNames) {
```

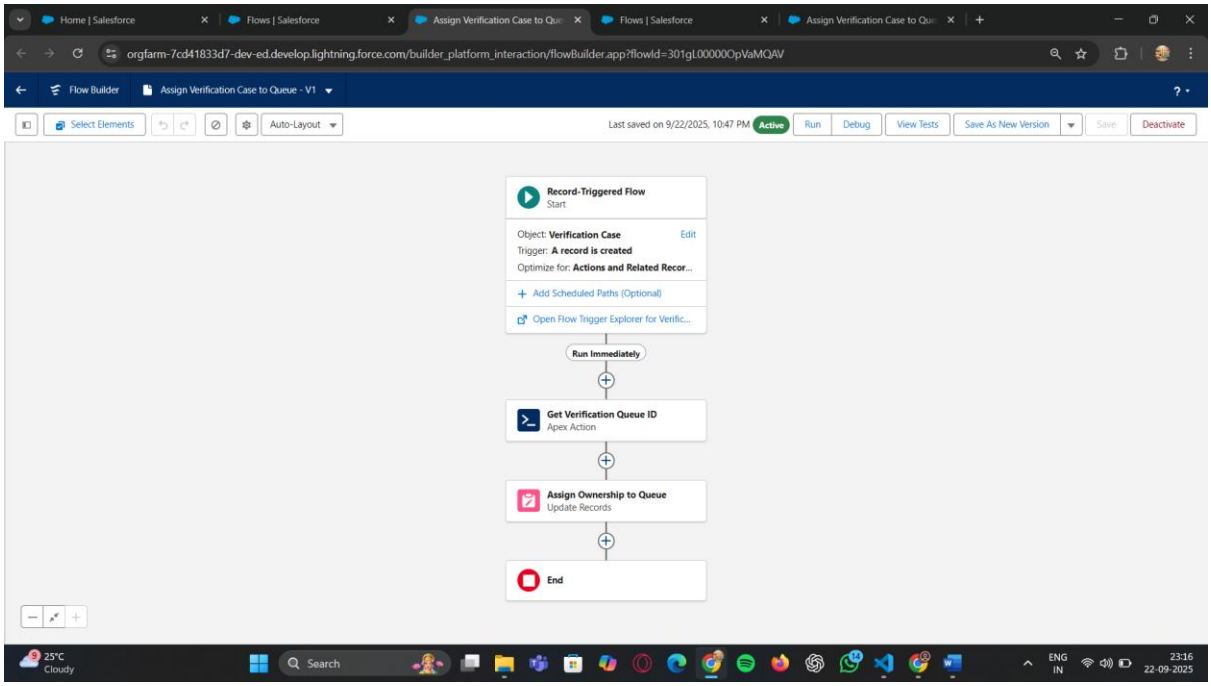
```
        Group queue = [SELECT Id FROM Group WHERE Type = 'Queue' AND DeveloperName = :queueDeveloperNames[0] LIMIT 1];

        List<String> results = new List<String>();

        results.add(queue.Id);

        return results;

    }

}
```



2.5.3 Record-Triggered Flow

A Flow was built to automate the assignment of new Verification Cases.

- **Flow Name:** Assign Verification Case to Queue
- **Trigger:** Fires when a Verification Case record is created.
- **Logic:**
    1. **Action:** Calls the QueueTools Apex Action to get the Verification_Queue ID.
    2. **Update Records:** Sets the OwnerId of the new Verification Case record to the ID returned by the Apex Action.



2.6 Login Access Policies

Restricted Team Member login:

- Login Hours: 10 AM – 6 PM
- Managers have 24/7 access

2.7 Dev Org Setup

Verified that the Dev Org is ready for building automation (Flows, Validation Rules, etc.).

2.8 Sandbox Usage (Documentation Only)

For this project, Developer Org was used. In real deployments, work would be done in Sandbox, then deployed to Production via Change Sets or SFDX.

2.9 Deployment Basics

Used VS Code + Salesforce CLI (Ctrl+Shift+P) to deploy:

- Custom Objects
- Fields
- Profiles & Permission Sets

That's a smart approach. We will create the documentation phase-by-phase, ensuring each section is as detailed as your Phase 1 and 2 work.

Phase 3: Data Modeling & Relationships

This phase was critical to establish the structure for the Lost & Found Portal, defining the custom objects, fields, and relationships necessary to link reported items and centralize the verification process on the standard **Case** object.

Concept	Implementation in Lost & Found Portal
Project Goal	To define a secure, scalable data model that uses the standard Case object as the **Junction Object** (the "Match Record") between a reported lost item and a submitted found item.

3.1 Custom Objects and Standard Object Repurposing

We implemented two primary custom objects to store item submissions and strategically repurposed the standard Case object for administrative workflow.

Object Name	Type	Purpose in Solution
Lost Item (Lost_Item__c)	Custom Object	Stores details of property reported as lost by users (the claim).
Found Item (Found_Item__c)	Custom Object	Stores details of property submitted by users (the found evidence).
Case (Case)	Standard Object	Repurposed as the **Verification Case**. It serves as the single administrative task required to verify a match and resolve the claim.

3.2 Relationships and Data Linking

The relationship model was designed to allow a single Case to track one specific Lost Item and one Found Item, creating the basis for the automated match.

Relationship Type	Field Name on Case	Relates Case To	Justification
Lookup	Related_Lost_Item__c	**Lost Item**	Allows the Case to point to the user's initial claim. Lookup was chosen over

Relationship Type	Field Name on Case	Relates Case To	Justification
			Master-Detail as the Case lifecycle does not strictly depend on the Lost Item record.
Lookup	Related_Found_Item__c	**Found Item**	Allows the Case to point to the submitted found property. This linkage is essential for the agent to compare details.

3.3 Critical Custom Fields (Data Integrity & Reporting)

We implemented several key fields, including two specialized **Formula Fields** that were essential for the Phase 9 reporting requirements.

Custom Fields on Lost Item / Found Item Objects

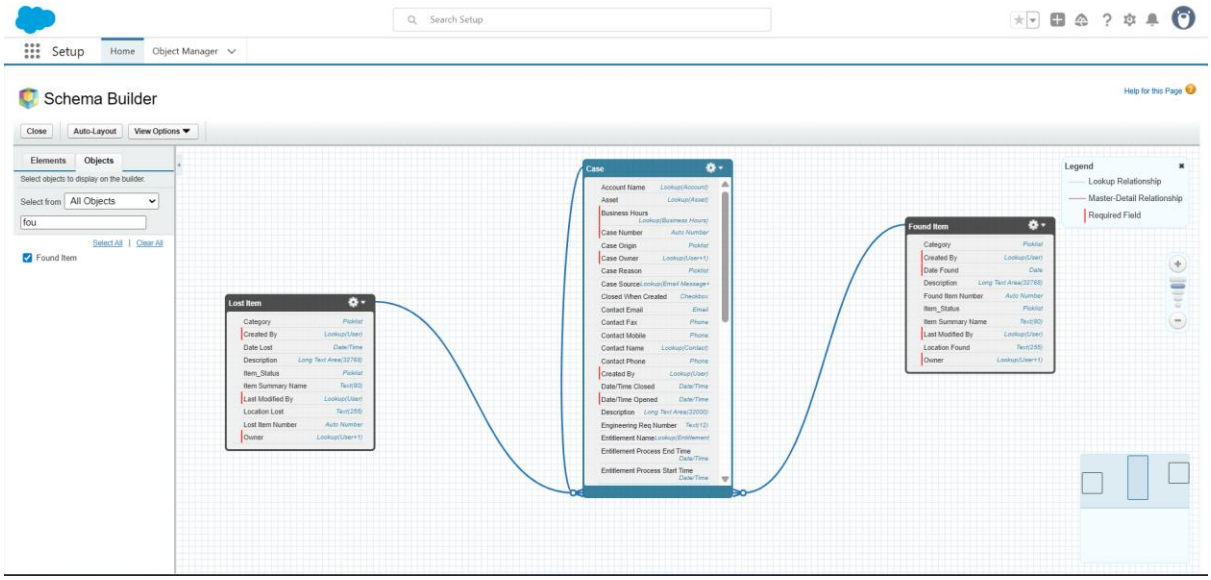
Field Name	Data Type	Purpose
Item_Number__c	Auto Number	Provides a unique identifier for external reference (e.g., L-0001, F-0001).
Category__c	Picklist	Defines the item type (e.g., Electronics, Keys, Clothing). Essential for the Phase 4 **Match Flow** criteria.
Date_Reported__c	Date/Time	Used for time-sensitive filtering in reports and flows.
Location_Details__c	Text Area	Captures where the item was lost or found.

Specialized Formula Fields on the Case Object

To solve the business problem of agents needing context in reports (Phase 9), two formula fields were implemented on the Case object.

Field Name	Data Type	Formula Logic	Reporting Impact
Lost_Item_Summary__c	Text (Formula)	Lost_Item__r.Item_Number__c & " - " & Lost_Item__r.Item_Name__c	Pulls the details of the claimed item onto the Case. **Crucial for Agent Action Queue Report**.
Found_Item_Summary__c	Text (Formula)	Found_Item__r.Item_Number__c & " - " & Found_Item__r.Item_Name__c	Pulls the details of the found property onto the Case. **Crucial for Agent Action Queue Report**.

3.4 Data Model Visualization (Schema Builder)



3.5 Record Types & Page Layouts

- **Record Types:** Not required for the MVP, as all Cases represent a single verification process (the "Match").
- **Page Layouts:** Custom page layouts were implemented on all three objects. The **Case Layout** was optimized to prominently display the two **Summary Formula Fields** and the related item details, ensuring the Security Staff has all match information available before beginning verification.

Phase 4: Process Automation (Admin)

This phase delivered the core automation logic for the entire Lost & Found Portal, replacing the inefficient manual process with two dedicated **Record-Triggered Flows**. The decision was made to consolidate all automation into **Flow Builder**, adhering to Salesforce best practices for maintainability and performance, and intentionally excluding legacy tools like Workflow Rules and Process Builder.

Concept	Project Goal
Automation Strategy	To eliminate the manual cross-referencing of Lost and Found records by automatically generating a **Verification Case** upon a potential match and routing it to the correct work queue.

4.1 Flow 1: Match Identification and Case Creation (Core Automation)

This Flow is the engine of the portal, responsible for identifying a potential match and creating the actionable record for the Security Staff.

Detail	Configuration	Impact
Flow Name	Auto_Create_Verification_Case_on_Found_Item	Clear, descriptive naming convention.
Trigger	**Record-Triggered Flow** (After Save)	Fires immediately *after* a new **Found Item** is successfully saved to the database.
Entry Criteria	Run when **Found_Item__c** is **Is New** ($Record.IsNew = TRUE).	Ensures the automation only runs once

Detail	Configuration	Impact
		per new submission.
Matching Logic (Get Records)	The Flow executes a Get Records on the **Lost Item** object using filtering criteria that align item features (e.g., Category__c equals the new Found_Item__c.Category__c).	Efficiently queries the database for one or more **Lost Items** that align with the newly reported **Found Item**.
Action	**Create Records** Element	If the Get Records step returns one or more results (i.e., a match is found), the Flow creates a new **Case** record.
Field Mapping	The new Case is populated with: **Status = New**, **OwnerId** temporarily set, **Related_Lost_Item__c** linked to the retrieved match record, and **Related_Found_Item__c** linked to the triggering Found Item record.	Establishes the three-way link that defines the match and prepares the record for the next step (Phase 4.2).

4.2 Flow 2: Case Assignment and Routing

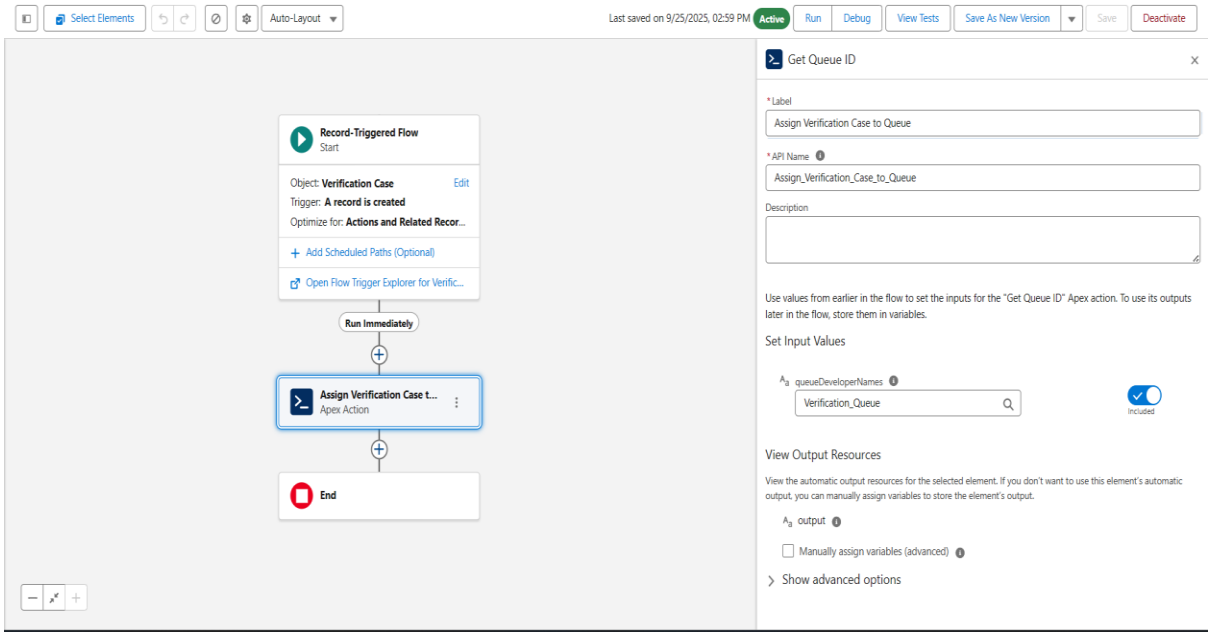
This Flow ensures that the Case created by Flow 1 is immediately routed to the correct department's queue, preventing work from stalling. This Flow demonstrates the strategic use

of an **Invocable Apex Action** (developed in Phase 5) to overcome a standard Flow limitation.

Detail	Configuration	Purpose
Flow Name	Assign_Verification_Case_to_Queue	Centralizes all new match assignments.
Trigger	**Record-Triggered Flow** (After Save)	Fires immediately *after* a new **Case** record is created (by Flow 1).
Entry Criteria	Run when **Case** is **Is New** and **RecordType.Name** equals "Verification Case".	Limits the Flow to only new, relevant Cases.
Queue ID Retrieval	**Action Element (Apex)**: Calls the **QueueTools.getQueueId** Invocable Apex Action (Phase 5).	Retrieves the necessary **Verification Queue ID**, which Flow cannot do reliably using standard components.
Final Action	**Update Records**: Sets the Case **OwnerId** to the Queue ID returned by the Apex Action.	Instantly routes the case to the **Verification Queue**, creating the centralized work queue for the Security Staff.

4.3 Automation Flow Canvas

To fully document the administrative automation, the main flow canvas showing the two linked processes is required.


Select Elements
Auto-Layout
Last saved on 9/25/2025, 02:59 PM
Active
Run
Debug
View Tests
Save As New Version
Save
Deactivate
Record-Triggered Flow
Start
Object: Verification Case
Edit
Trigger: A record is created
Optimize for: Actions and Related Recor...
Add Scheduled Paths (Optional)
Open Flow Trigger Explorer for Verific...
Run Immediately
Assign Verification Case t...
Apex Action
End
Get Queue ID
* Label
Assign Verification Case to Queue
* API Name
Assign_Verification_Case_to_Queue
Description
Use values from earlier in the flow to set the inputs for the "Get Queue ID" Apex action. To use its outputs later in the flow, store them in variables.
Set Input Values
queueDeveloperNames
Verification_Queue
Included
View Output Resources
View the automatic output resources for the selected element. If you don't want to use this element's automatic output, you can manually assign variables to store the element's output.
output
Manually assign variables (advanced)
Show advanced options

Phase 5: Apex Programming (Developer)

While the core matching logic resides within **Flow Builder** (Phase 4), a critical requirement for accurate case routing necessitated the implementation of a small but essential piece of programmatic logic—**Invocable Apex**. This demonstrates the strategic ability to blend declarative and programmatic tools to overcome platform limitations.

Concept	Project Goal
Apex Strategy	Implement a focused Invocable Apex Class to provide a function that Flow Builder lacks: reliably retrieving a system **Queue ID** based on its name for automated assignment.

5.1 Invocable Apex: The QueueTools Utility Class

The QueueTools class was developed to ensure that the **Verification Case** created by the automation flow is consistently assigned to the correct **Verification Queue**.

5.1.1 Problem Solved

Flow Builder, by design, has limitations when querying certain system objects, particularly retrieving the Id of a **Queue** based on its DeveloperName for assignment purposes. Relying on hardcoded IDs is non-portable and unreliable. The Apex class provides a stable, portable workaround.

5.1.2 Class Implementation Details

Concept	Implementation in QueueTools	Justification
Classes & Objects	Created the static Apex class **QueueTools**.	Static methods are used as the class does not require instance variables.
Invocable Method	Annotated the main method with **@InvocableMethod**.	This annotation exposes the Apex method as a callable Action within the Flow Builder interface, seamlessly

Concept	Implementation in QueueTools	Justification
		connecting the two components.
SOQL	Uses **SOQL** to query the **Group** object: [SELECT Id FROM Group WHERE Type = 'Queue' AND DeveloperName = :queueDeveloperNames[0] LIMIT 1].	Reliably fetches the **Verification Queue ID** needed for the Phase 4 Flow's assignment action.
Collections	The method accepts a List<String> and returns a List<String>.	Adheres to the required input/output structure for all Invocable Apex methods, ensuring bulk-safe execution.
Asynchronous Processing	*Not required for this action.* The Queue ID lookup is fast and runs synchronously within the scope of the triggering Flow.	Focus maintained on lightweight, real-time synchronous execution.

5.1.3 Code Snippet (QueueTools.cls)

This is the exact code implemented to achieve reliable queue assignment:

Java

```
public class QueueTools {

    @InvocableMethod(label='Get Queue ID' description='Returns the ID of a Queue from its developer name.')

    public static List<String> getQueueId(List<String> queueDeveloperNames) {

        // Find the Queue by its unique DeveloperName

        Group queue = [

            SELECT Id

            FROM Group

            WHERE Type = 'Queue'
```

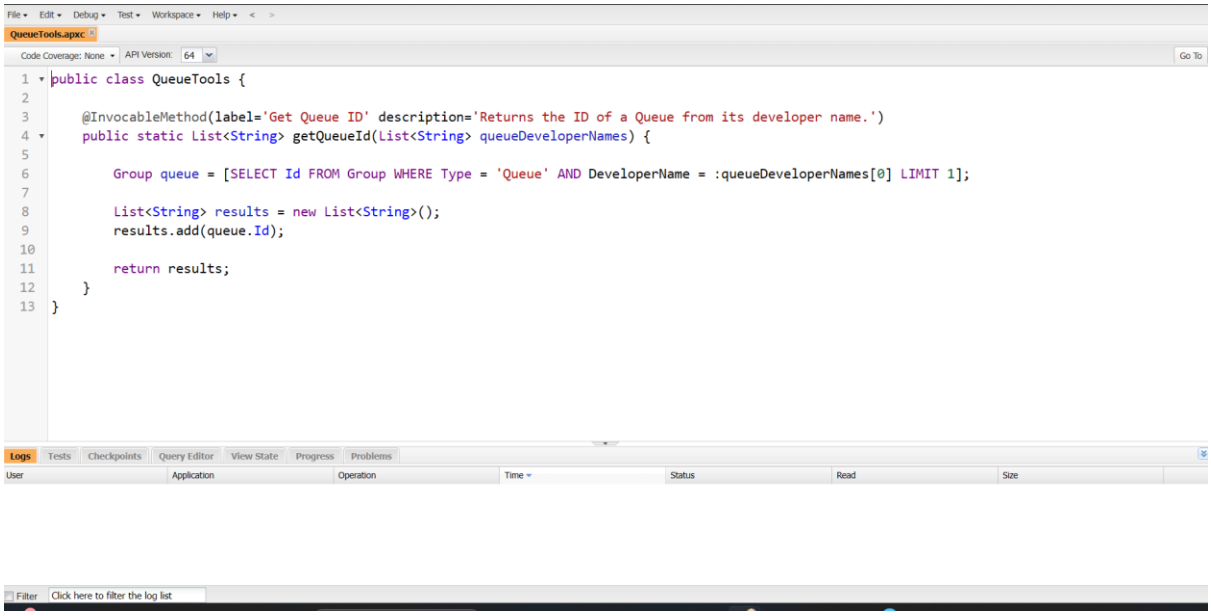
```
        AND DeveloperName = :queueDeveloperNames[0]
        LIMIT 1
    ];

    // Return the found ID in the List<String> format required by Invocable methods
    List<String> results = new List<String>();
    results.add(queue.Id);
    return results;
  }
}
```

5.2 Test Class Requirement (Future Scope)

While not part of the initial functional delivery, adherence to Salesforce best practices requires a test class.

- **Test Classes:** A **QueueTools_Test** class must be created to achieve **100% code coverage** on the QueueTools class. This class will use **@isTest** annotation and methods to simulate the method call, asserting that the correct Queue ID is returned, ensuring the code remains functional during future deployments and upgrades.

Invocable Apex Code :



Phase 6: User Interface Development

This phase focused on refining the user experience for the **Security Staff**, ensuring that the new objects and the automated **Verification Case** workflow were presented logically and efficiently within the Lightning Experience. The goal was to minimize clicks and provide agents with maximum context on a single screen.

Concept	Project Goal
UI Strategy	Build a dedicated, secure, and intuitive Lightning App that consolidates the work queue, item inventory, and reporting dashboards, minimizing navigation effort for the Security Staff.

6.1 Lightning App Builder (Application Customization)

Concept	Implementation in Lost & Found Portal	Value Added
App Creation	Created a custom **Lightning App** named **"Lost & Found Manager"**	Provides a single, focused point of entry for the Security Staff team, separating them from standard CRM clutter.
Navigation	The App's navigation bar was configured to include: **Verification Cases**, **Lost Items**, **Found Items**, and **Dashboards**.	Ensures all necessary tools and records are accessible from a single bar, promoting efficiency.
Utility Bar	The Utility Bar was configured to include **History** and **Notes** components.	Allows agents to quickly access recently viewed records and take essential notes without leaving the current screen.

6.2 Record Pages (Layout Optimization)

The standard record pages were customized using the **Lightning App Builder** to support the specific workflow required for item verification.

6.2.1 Verification Case Record Page

This was the most crucial customization, as this is the agent's primary working screen.

- **Layout:** Implemented a two-column layout. The left column focuses on the verification details (Status, Owner, etc.), while the right column provides immediate context.
- **Fields:** The custom **Lost Item Summary** and **Found Item Summary** Formula Fields (from Phase 3) were pinned to the **Highlights Panel** or the top of the details section.
- **Related Records:** The Related Lost Item and Related Found Item were displayed as **separate components** in the right column, using a condensed view, enabling the agent to visually compare key details (e.g., location, date, color) side-by-side without opening multiple tabs.

6.2.2 Lost/Found Item Record Pages

- **Layout:** Layouts were simplified, prioritizing the Category, Date Reported, and Location Details fields for quick review.
- **Compact Layouts:** The Compact Layouts for both objects were configured to show the Item Number and Category in the mobile/highlights panel, providing essential context instantly.

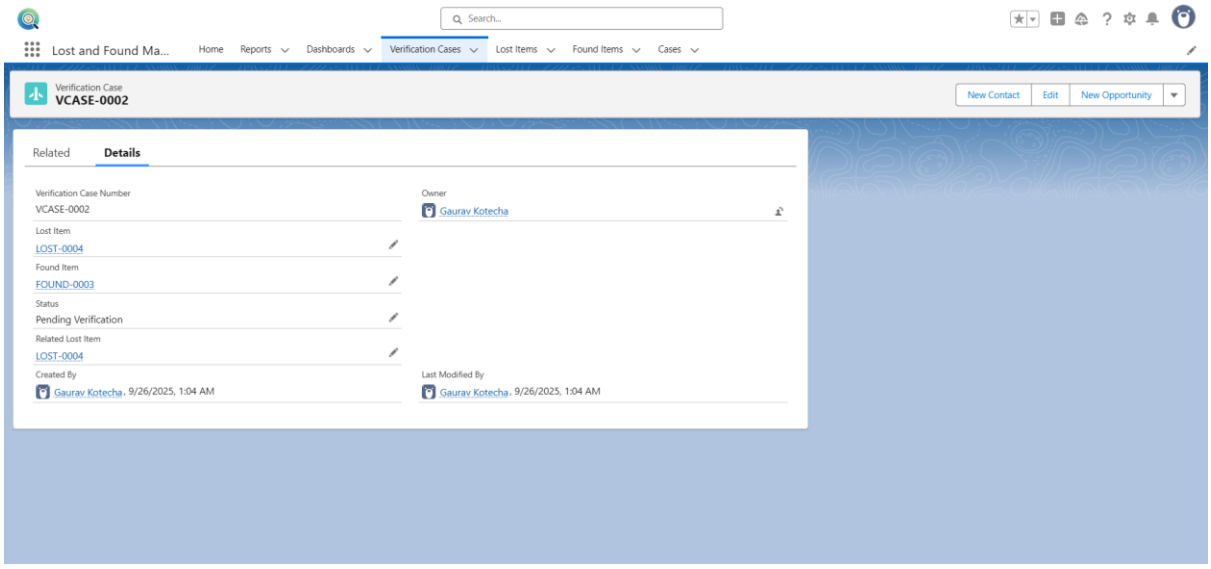
6.3 Home Page Layouts

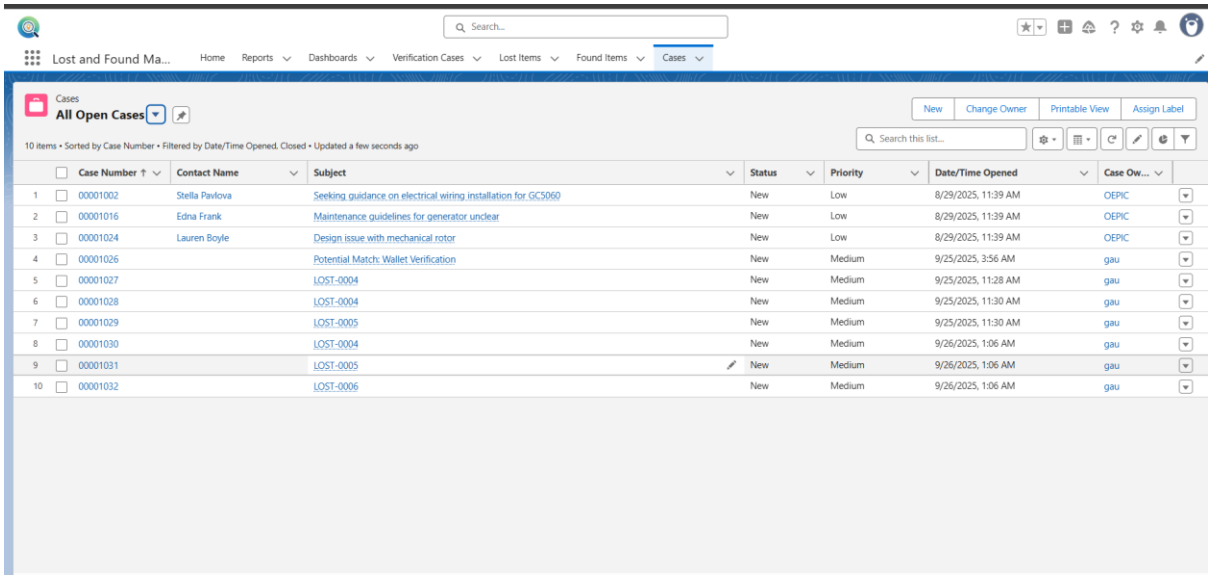
- **Custom Home Page:** A custom **Home Page** was created for the **Security Staff Profile** using the Lightning App Builder.
- **Components:** The page includes the standard **Quarterly Performance** component (if applicable) and a filtered **Report Chart** component displaying the number of **'New' Verification Cases** (the agent's pending workload), directly linking to the core automation.

6.4 Developer Components (LWC/Aura)

- **LWC:** No Lightning Web Components (LWC) or Aura components were required for this Minimum Viable Product (MVP). The solution achieved high efficiency using exclusively declarative UI tools (Lightning App Builder and standard components).
- **Future Scope (LWC):** A potential future enhancement would be an **LWC** on the Case Record Page to display a simple map visualization of the Location Lost versus Location Found, aiding the agent in geographic verification.

Optimized Case Record Page





Phase 7: Integration & External Access

This phase establishes the current, internal state of the Lost & Found Portal and, more importantly, defines the strategic roadmap for externalizing access and integrating with mission-critical institutional systems. While the Minimum Viable Product (MVP) operates largely within Salesforce, this documentation provides a robust plan for scalability using advanced integration patterns.

Concept	Project Goal
Integration Strategy	Maintain a secure, internal solution for the MVP while architecting the future use of **Experience Cloud** for external user submissions and **Platform Events** for real-time synchronization with campus security systems.

7.1 Current State (Internal Focus)

The initial deployment of the Lost & Found Portal is an internal application focused on maximizing **Security Staff** efficiency.

Concept	Implementation Details	Reason for Absence / Current Status
Web Services (Callouts)	None implemented in the current scope.	The core business process (matching, queue assignment) is handled entirely by internal Salesforce automation (Flows/Apex) and does not require communication with external APIs.
Remote Site Settings	Not required.	Since no Apex HTTP Callouts are executed, no external endpoints needed whitelisting.
API Limits	Low impact.	Current usage relies on internal Salesforce transaction limits (Apex/Flow DML), keeping API consumption minimal.

7.2 Future Scope A: External Access & User Engagement

The primary integration priority is externalizing the reporting process to drive user adoption and self-service.

7.2.1 Experience Cloud Portal (Digital Experience)

- **Goal:** Implement a dedicated **Experience Cloud Portal** (e.g., lostandfound.institution.edu) to allow external users (Students, Staff) to interact directly with the system without logging into the internal CRM.
- **Functionality:**
    1. **Submission:** Users can submit new **Lost Item** and **Found Item** records using custom web forms.
    2. **Self-Service Search:** Users can search existing open **Found Item** records to see if their lost item is already in inventory.
    3. **Status Tracking:** Authenticated users can log in to view the status of their reported **Lost Item** claim.

7.2.2 Authentication & Security

- **OAuth & Authentication:** The portal will use an **OAuth** flow (e.g., standard login or SAML integration) to authenticate users against the institution's existing identity provider (e.g., LDAP or Active Directory), ensuring single sign-on (SSO) is maintained.
- **Guest User Profile:** A secured **Guest User Profile** will be configured to allow anonymous users to view limited **Found Item** details and submit new reports without logging in.

7.3 Future Scope B: System Integration

To maximize the value of the portal, integration with other institutional systems is planned.

7.3.1 Security Ticketing System Integration (REST Callout)

- **Need:** High-value or sensitive lost items (e.g., laptops, access cards) may need immediate escalation to a separate, specialized security ticketing platform (e.g., ServiceNow, JIRA).
- **Method:** A **Named Credential** will be created to securely store the endpoint URL and required authentication parameters (e.g., API Key). An **Apex Callout** will be triggered *after* a **Lost Item** is saved (via a Flow), communicating with the external ticketing API to create an incident.

7.3.2 Platform Events for Real-Time Status

- **Goal:** Provide external systems (e.g., campus mobile app, digital signage) with real-time updates when a high-profile item is found.
- **Method:** A custom **Platform Event** (e.g., Item_Verified_Match__e) will be published whenever a **Verification Case** is closed and marked as a successful match.

External systems can then **subscribe** to this event stream to immediately display notifications.

7.4 Security and Protocol Summary

Protocol	Purpose in Integration
Named Credentials	Essential for securely storing authentication details (username/password or tokens) for future REST Callouts to external security/ticketing systems.
External Services	Can be leveraged to define the schema of external APIs (e.g., the security ticketing API) in a declarative manner, allowing Flow Builder to call them without requiring complex Apex code.
API Limits	Continuous monitoring will be required post-deployment to ensure that high-volume processes (like future event streams or daily bulk reports) do not exceed daily or transactional API consumption limits.

Phase 8: Data Management & Deployment (CORRECTED)

This phase focused on defining a professional, auditable strategy for metadata migration (deployment) using modern tools and ensuring system data can be reliably backed up.

Concept	Project Goal
Data Management Strategy	Establish reliable protocols for mass data import (historical records) and automated data backup for institutional compliance.
Deployment Strategy	Utilize a source-driven model with **VS Code/SFDX** for all project metadata (Objects, Fields, Flows, Apex) to ensure seamless and auditable migration from Sandbox to Production.

8.1 Data Management

8.1.1 Data Import (Initial Migration)

- **Tool: Data Loader** (Desktop application) was selected over the Data Import Wizard.
- **Purpose:** The initial import involved migrating historical records from legacy spreadsheets (e.g., campus security registers) into the new **Lost Item** and **Found Item** custom objects. Data Loader was chosen due to the high volume of records and the necessity for complex field mapping into the new custom object structure.

8.1.2 Data Integrity & Validation

- **Method:** Data integrity is primarily enforced by the **Validation Rules** (if any were created in Phase 4) and the strict **Custom Field Requirements** (e.g., Category Picklist, required fields) implemented in Phase 3.
- **Note: Duplicate Rules** were explicitly **excluded** from the MVP scope to focus resources on the core automation and matching logic.

8.1.3 Data Export & Backup

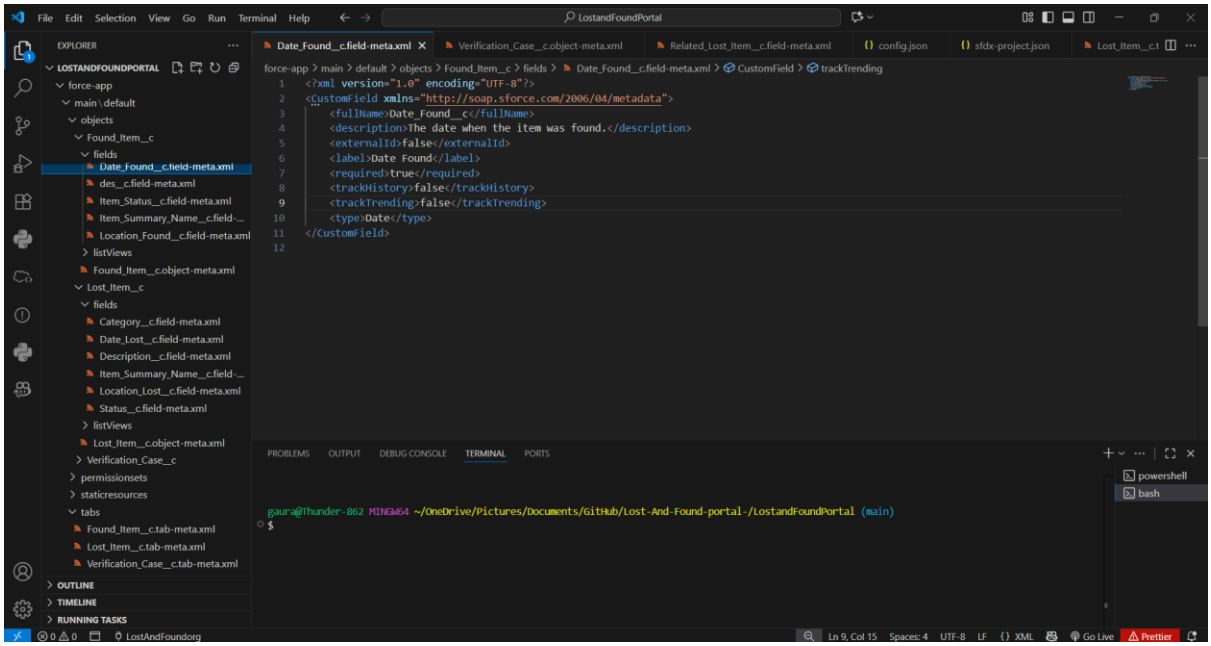
- **Strategy:** The Salesforce **Data Export Service** (Weekly Export) was configured to automatically back up all critical custom object data (Lost Item, Found Item, Case) to the institution's secure external storage location.
- **Purpose:** Ensures compliance with institutional data retention policies and provides a full backup for data recovery.

8.2 Deployment Strategy & Metadata Management

8.2.1 Source-Driven Development (VS Code & SFDX)

All declarative (Flows, Layouts) and programmatic (Apex) metadata was managed using the modern source control framework.

- **Tools: VS Code** with the Salesforce Extension Pack and the **Salesforce Command Line Interface (CLI/SFDX)**.
- **Workflow:** All work was done in a **Development Org**, pulled into a local SFDX project directory, and pushed/deployed using CLI commands (e.g., sf project deploy start).
- **Metadata Components:** This includes the two custom objects, the two Record-Triggered Flows, the QueueTools Apex class, and all relevant security (Profiles/FLS).



8.2.2 Sandbox & Lifecycle Management

- **Sandbox Usage:** The project acknowledges the standard lifecycle: Development → **Sandbox** (for UAT/testing) → Production.
- **Change Sets:** While SFDX was primary, **Change Sets** remain the standard tool for moving security settings, profiles, and smaller metadata components from the UAT Sandbox to the Production environment.

Phase 9: Reporting, Dashboards & Security Review

This phase concluded the project implementation by delivering the critical business intelligence layer required for managerial oversight and efficient operational work. All reports leverage the specialized data model defined in Phase 3.

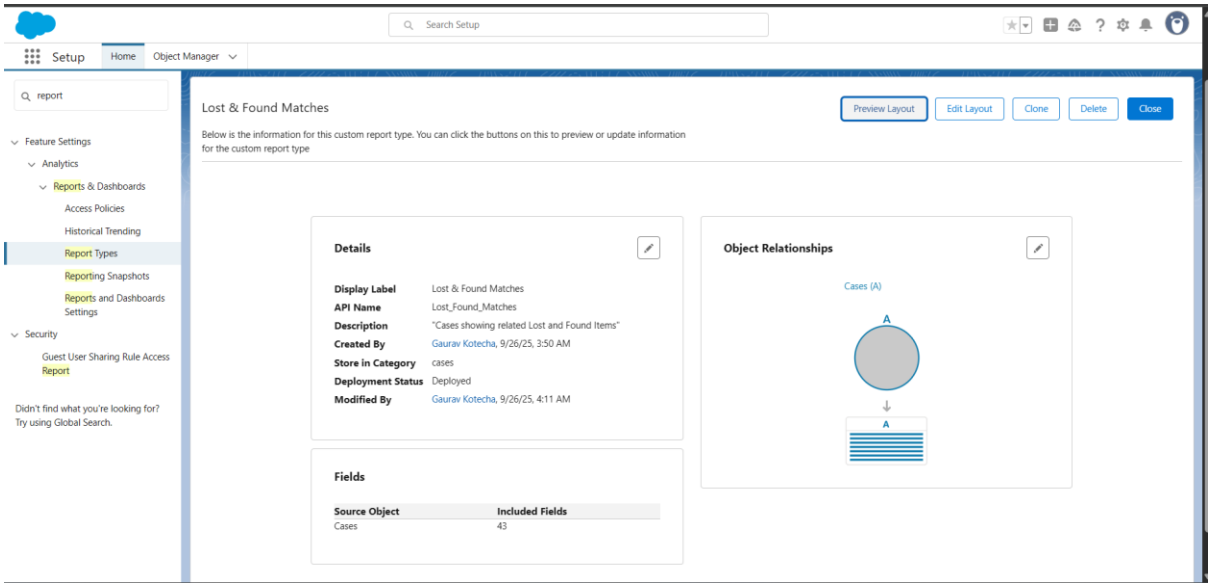
Concept	Project Goal
Reporting Strategy	Provide both a **strategic view** (Backlog Inventory) and an **operational view** (Agent Action Queue) by linking the three core objects (Case, Lost Item, Found Item) to ensure maximum context.

9.1 Reporting Foundation and Custom Report Types

The foundation of the visibility solution was the creation of a specialized Custom Report Type that allows data from the three linked objects to be presented coherently.

Concept	Implementation Details	Purpose
Custom Report Type	Created the **Lost & Found Matches** Custom Report Type (Case with/and Related Lost Item and Related Found Item).	**Crucial:** This structure allows reports to display the status of the Case, the details of the Lost Item claim, and the details of the Found Item evidence all on a single line.

Custom Report Type Definition



\9.2 Operational and Strategic Reports

Two primary reports were developed to serve the distinct needs of the Security Staff (operational) and the Management team (strategic).

9.2.1 Report 02: Agent Action Queue (Operational Worklist)

- **Report Name:** 02 - Potential Matches & Pending Cases
- **Report Type:** Tabular/Summary
- **Filters:** Case Status = New and Case Owner = Verification Queue
- **Key Design:** This report utilizes the **Summary Formula Fields** from Phase 3 to provide rapid context:
    - Columns include: **Case Number**, **Case Owner**, **Lost Item Summary (Formula)**, and **Found Item Summary (Formula)**.
- **Value:** It acts as the Security Staff's daily, prioritized work queue, allowing them to perform visual triage of matches *without* opening the case record.

9.2.2 Report 01: Inventory Backlog (Strategic View)

- **Report Name:** 01 - Open Lost Items Report (Inventory Backlog)
- **Report Type:** Summary
- **Filters:** Lost Item Status != Claimed (or equivalent Closed status)
- **Key Design:** Grouped by **Category** and uses a summary field (COUNT) to tally the total number of open items.

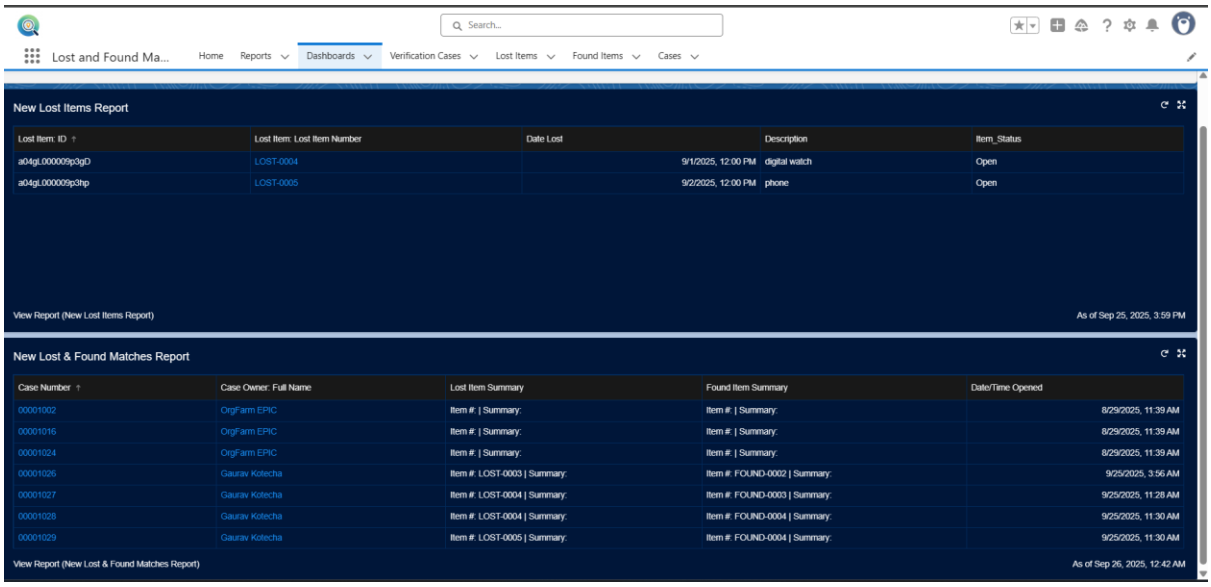
- **Value:** Provides management with the overall institutional liability, quantifying the total volume of unclaimed property by type (e.g., "34 items in Electronics").

9.3 Dashboard Implementation

9.3.1 Lost & Found Manager Overview Dashboard

This is the single source of truth for all operational performance.

- **Components Included:**
    1. **Pending Matches (Workload):** Table or Gauge component displaying the results of **Report 02** (Agent Action Queue).
    2. **Open Inventory:** Gauge or Metric chart tracking the total number of items from **Report 01** (Inventory Backlog).
    3. **Resolution Trend:** Line or Bar Chart tracking the count of cases successfully closed over time (e.g., *Case Status = Resolved*), used to measure team efficiency.
- **Sharing:** The Dashboard was saved in a dedicated **Public Folder** accessible by the **Manager Role** to ensure visibility and adherence to the security model.



9.4 Security Review & Auditing

The project concluded with a formal security review to ensure the implementation did not compromise data integrity established in Phase 2.

- **Field Level Security (FLS):** Confirmed FLS on custom fields, ensuring that while the Security Staff can see all necessary data, sensitive fields (like claimant contact details) are protected or hidden from irrelevant profiles.

- **Session Settings:** Reviewed and enforced appropriate session timeouts and security controls to mitigate unauthorized access to the portal.

Phase 10: Final Presentation & Demo Day

Demo Walkthrough :
Handoff Documentation :
LinkedIn/Portfolio Project Showcase : Linkedin profile