Phase 1: Problem Understanding & Industry Analysis

Requirement Gathering

The Urban Issue Management System is designed to streamline and automate the process of reporting, tracking, and resolving urban infrastructure issues (such as potholes, streetlight outages, garbage problems, playground maintenance, and traffic signal malfunctions). The primary goal is to provide a responsive, transparent, and efficient mechanism for citizens to report problems, for staff to manage and resolve them, and for supervisors and city management to monitor performance and trends.

Key requirements identified include:

- Citizens need a simple and accessible way to submit urban issues, specifying details such as location, type, and description.
- Each issue must be routed to the appropriate department for resolution based on type and location.
- Staff and supervisors require a centralized platform to track the progress of each reported issue.
- Automated assignment of priority and escalation for urgent or critical issues is essential.
- Timely notifications and communications (e.g., email alerts) are required for both citizens and staff.
- The system must support reporting and analytics for operational oversight and continuous improvement.

Stakeholder Analysis

The main stakeholders for this project are:

- Citizens: Individuals who identify and report urban issues within the city limits. Their experience should be straightforward and transparent, with notification at each major status change.
- City Staff: Employees responsible for investigating and resolving reported issues. This includes field workers, department representatives, and supervisors.
- Supervisors/Managers: Oversee department performance, approve certain issues (especially urgent or costly ones), and manage escalations.

- City Management: Responsible for oversight, policy, and resource allocation, requiring dashboards and analytics.
- IT/Admin Team: Maintains the system, adapts processes, and manages integrations with other city systems.

Understanding the needs and workflows of each group is essential to the solution's success.

Business Process Mapping

The current (as-is) process is typically manual or semi-automated, involving phone calls, paper forms, or siloed apps. Issues can be lost, delayed, or tracked inefficiently. The proposed (to-be) process, enabled by Salesforce, is as follows:

1. Citizen submits a new urban issue via a portal or mobile app.
2. The system captures details, assigns a unique issue number, and automatically routes it to the relevant department based on the type and location.
3. Priority is automatically set according to predefined rules (e.g., urgent for main roads or hazardous locations).
4. Staff receive notification and a task for follow-up.
5. Supervisors are notified for approval if the issue is urgent or high-cost.
6. Issue status updates are communicated back to the citizen.
7. The system provides real-time dashboards and reports for management oversight.

This streamlined process reduces delays, increases accountability, and enhances citizen satisfaction.

Industry-Specific Use Case Analysis

Municipalities worldwide face similar challenges in managing urban infrastructure issues. Common pain points include lack of transparency, slow response times, poor communication with citizens, and difficulty in tracking performance. Industry best practices increasingly focus on digital transformation, citizen engagement, mobile accessibility, and data-driven insights.

Our solution draws inspiration from leading smart-city initiatives, leveraging Salesforce's platform capabilities for automation, notification, and analytics. The system is designed to be scalable, adaptable for future needs (such as IoT sensor integration or AI-driven prioritization), and compliant with municipal data policies.

AppExchange Exploration

To accelerate delivery and maximize value, the project explored relevant Salesforce AppExchange solutions for:

- Citizen service request management
- Field service scheduling and optimization
- GIS and mapping integrations
- Prebuilt dashboards and analytics for city operations

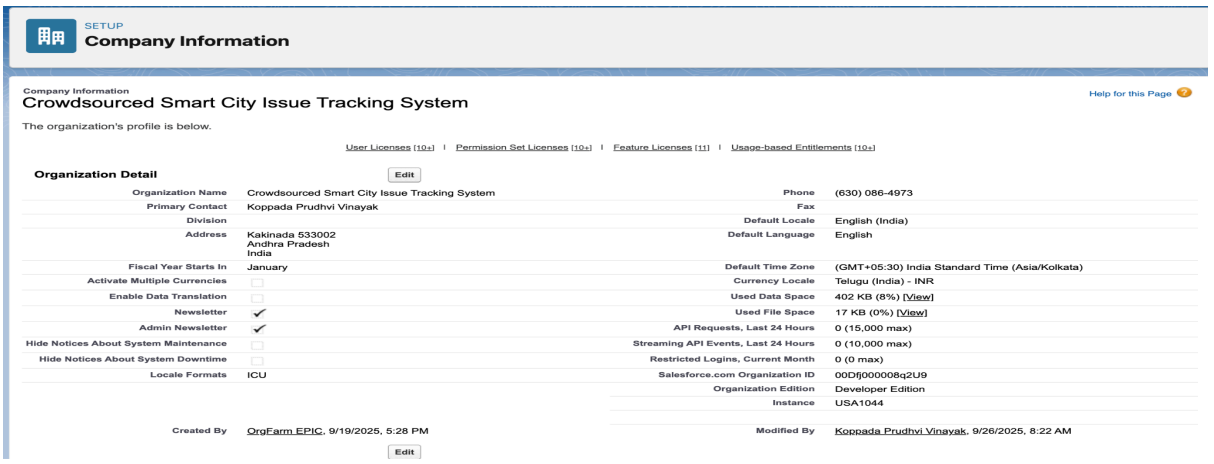
While several apps offer useful features, most are either too generic or lack the specific process customizations required for our city's needs. As a result, a custom Salesforce implementation, with selective use of AppExchange components for notifications or mapping, is chosen as the optimal path.

PHASE 2 – Org Setup & Configuration (Urban Issue Management System

Company Profile Setup

The foundational org settings were configured under Setup → Company Information → Edit for the Urban Issue Management System.

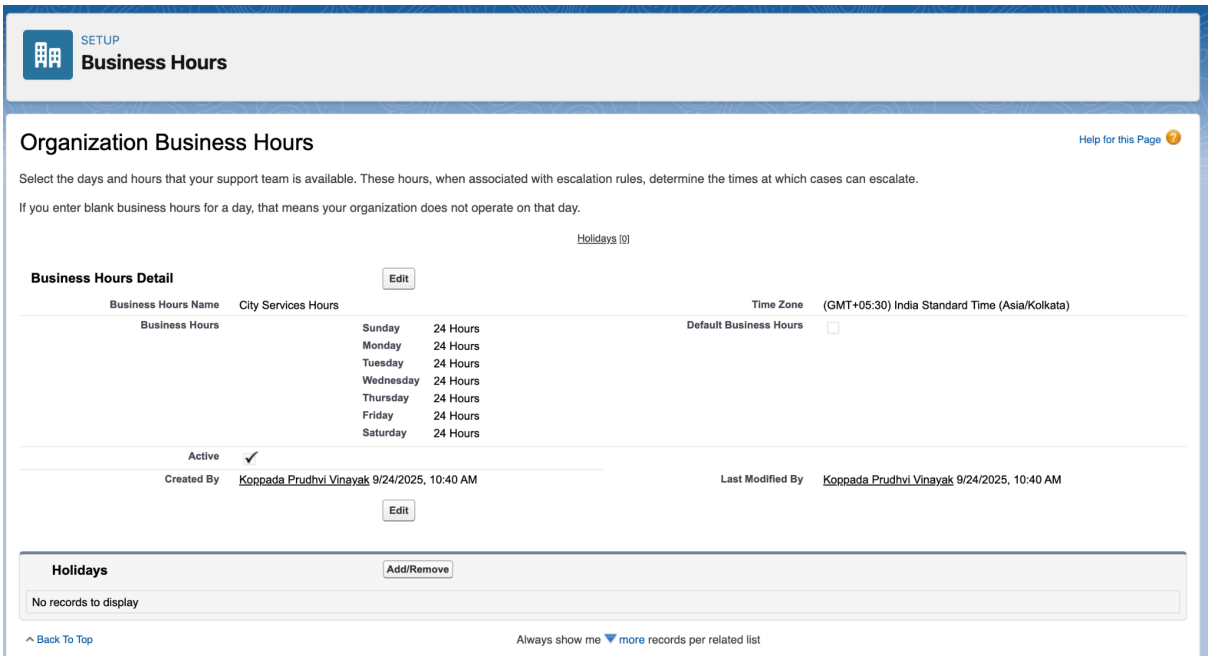
- Name: Urban Issue Management System
- Time Zone: GMT+05:30 Asia/Kolkata
- Locale: English (India)
- Language: English, Currency: INR



Business Hours & Holidays

Standard business hours were configured to match city/municipal operating hours for case routing and SLA calculations.

- Path: Setup → Business Hours → New
- Name: Urban Issue Standard Hours
- Time Zone: GMT+05:30 Asia/Kolkata
- Working Hours: Monday–Friday, 9:00 AM–6:00 PM
- Holidays: City government holidays and national holidays added for accurate service calculations.

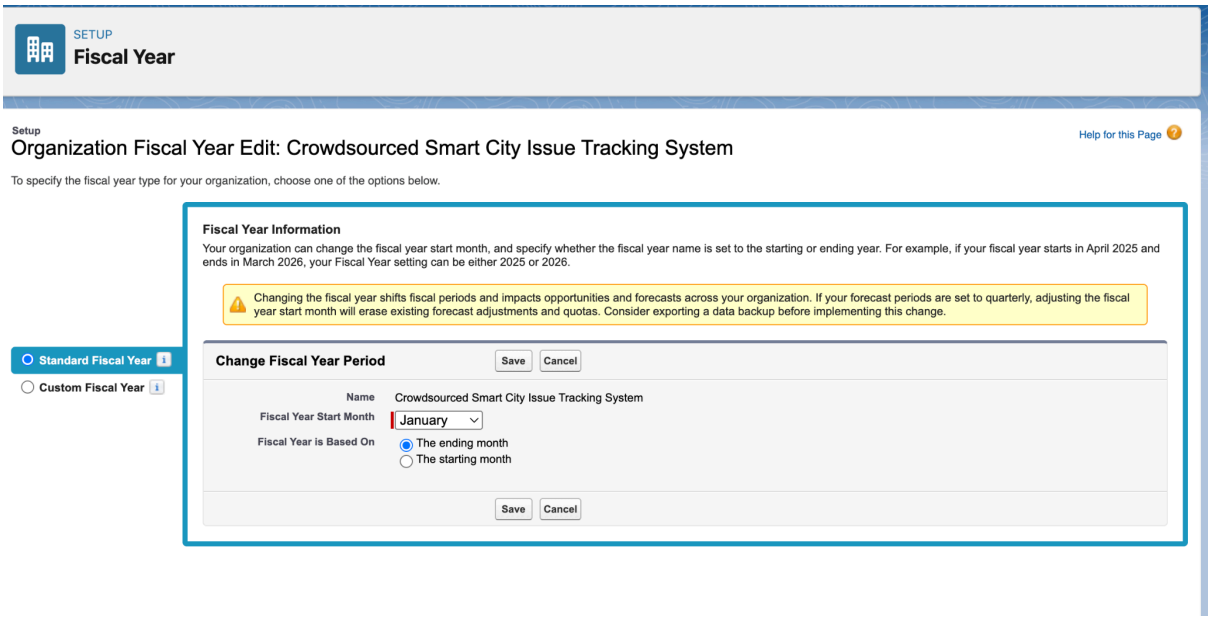


Fiscal Year Setup

The fiscal year was set to standard to align with municipal reporting cycles.

- Path: Setup → Fiscal Year
- Type: Standard Fiscal Year

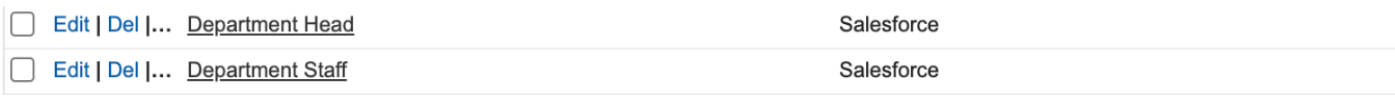
- Start Month: January



User Setup (Profiles, Roles, Permission Sets, Users)

Profiles

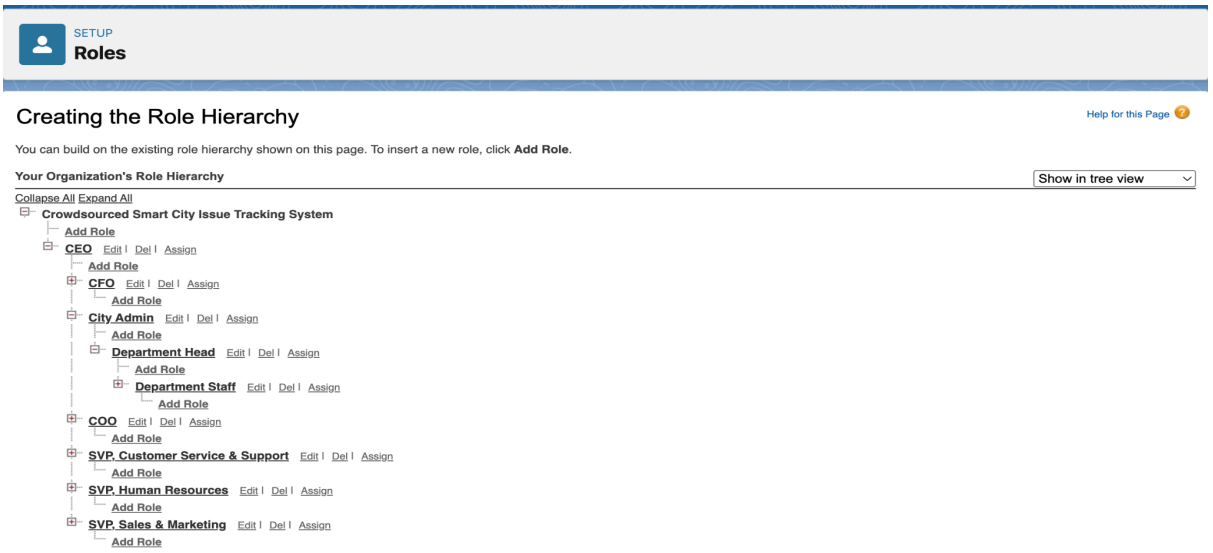
- *Setup → Users → Profiles.*
- *Cloned "Standard User" profile as Department Staff.*
- *Cloned again as Department Head.*
- *(Optional) Created "Citizen" profile for future Experience Cloud/portal use.*
- *Assigned these profiles to the corresponding users.*



Roles

- *Setup → Users → Roles.*
- *Created the following hierarchy:*
  - *City Admin (top, reports to CEO)*
  - *Department Head (reports to City Admin)*

    - *Department Staff (reports to Department Head)*
- *Assigned these roles to the users created earlier.*

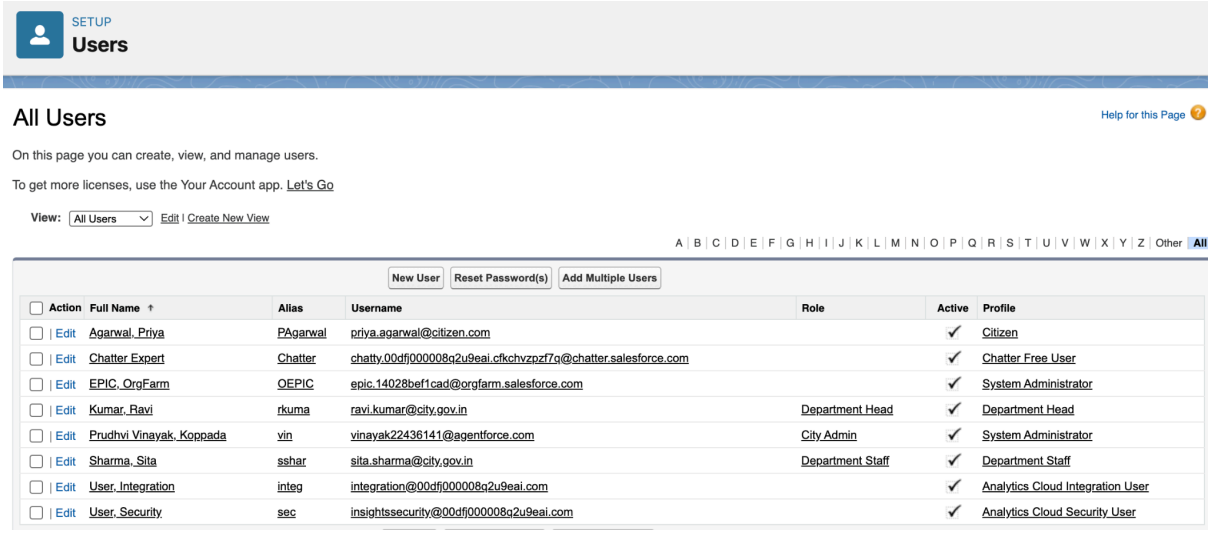


Permission Sets

- *Setup → Users → Permission Sets.*
- *Created Department Extended Access permission set.*
    - *Checked "Export Reports" and "Run Reports" under System Permissions.*
- *Assigned this permission set to department users (e.g., Ravi Kumar, Sita Sharma).*

User Setup & Licenses

- Setup → Users → Users.
- Created users for:
    - System Administrator (yourself)
    - Department Heads (e.g., Ravi Kumar)
    - Department Staff (e.g., Sita Sharma)
- Assigned “Standard User” or custom profile (created in next step).
- Set role to be assigned after roles were created.



Org-Wide Defaults (OWD)

- Setup → Security → Sharing Settings.
- Set Org-Wide Defaults (Default Internal Access) for:
    - Department: Public Read Only
    - Urban Issue: Private
    - Citizen: Private
    - Feedback: Private
- Saved the changes.
- Note: Default External Access left unchanged (no community/portal users yet).

Login Access Policies

- Setup → Security → Login Access Policies.
- Enabled “Administrators Can Log in as Any User”.

Dev Org Setup

- *All steps performed in a Salesforce Developer Edition org.*

Sandbox Usage

- Developer Edition org used; no sandboxes available.
- Noted in docs that for production, sandboxes and change sets/SFDX would be used for testing/deployment.

PHASE 3 – Data Modeling & Relationships (Urban Infrastructure Issue Reporting)

Standard & Custom Objects

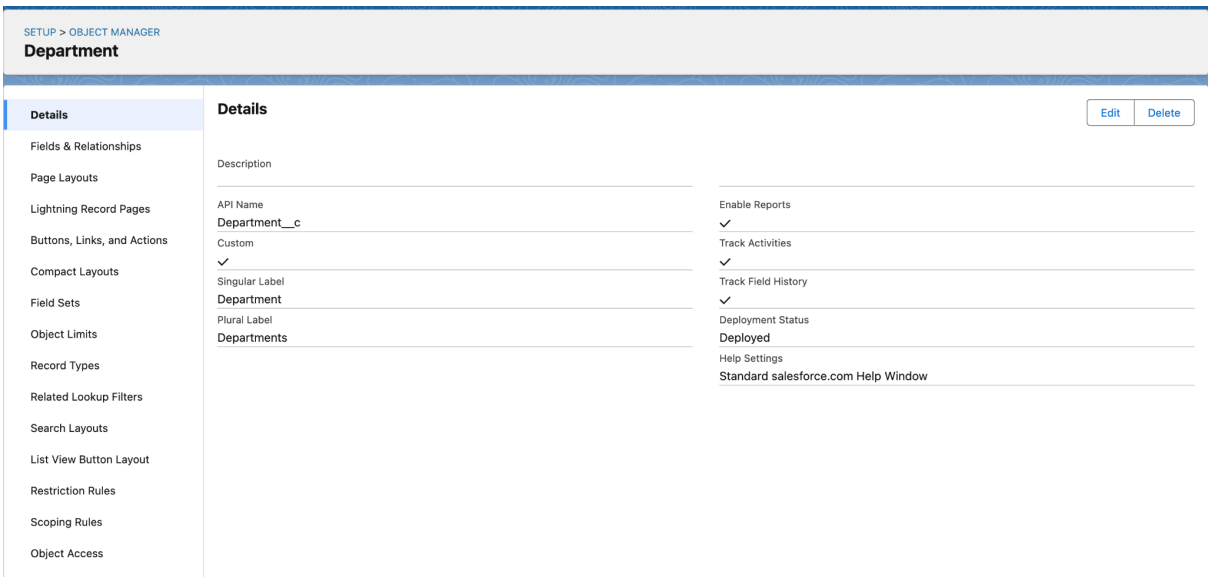
The data model leverages both standard and custom objects to create a scalable, government-centric application.

Standard Objects:

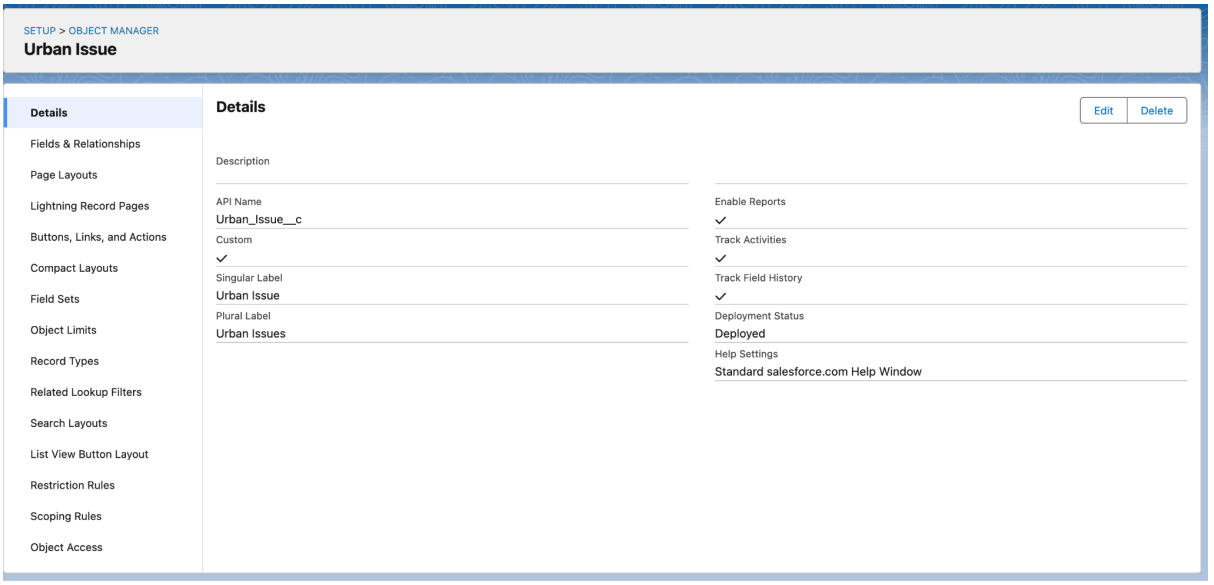
- User: Represents city staff, department heads, and system administrators who manage issues and oversee departments.

Custom Objects:
Four custom objects form the backbone of the solution:

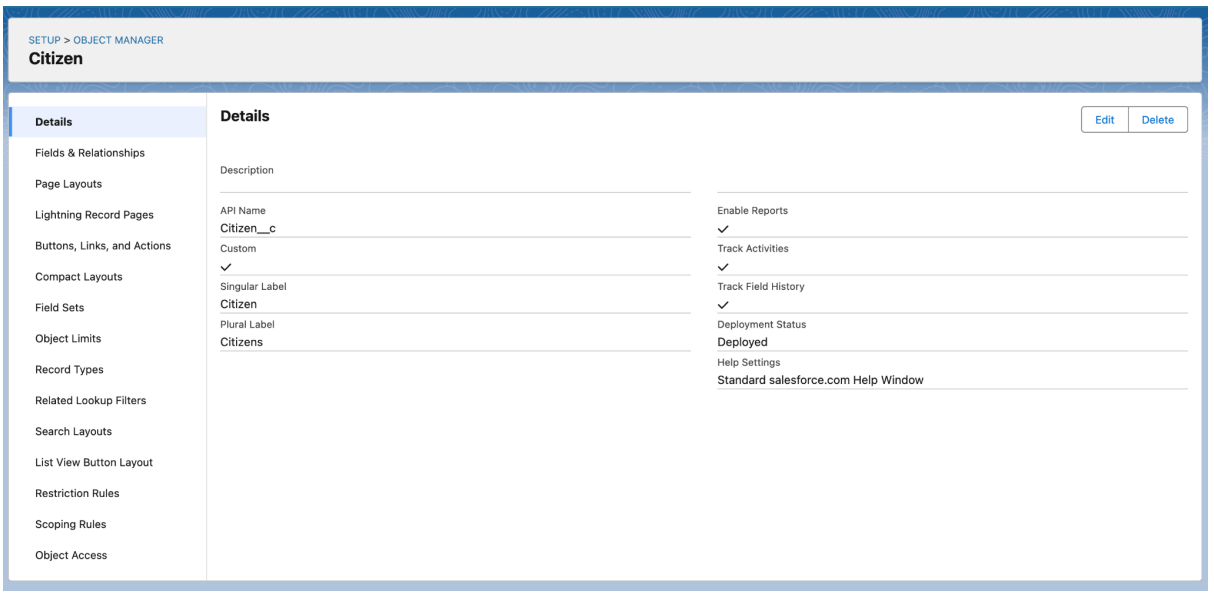
- Department: Represents each city or municipal department responsible for handling reported issues (e.g., Sanitation, Public Works, Utilities).



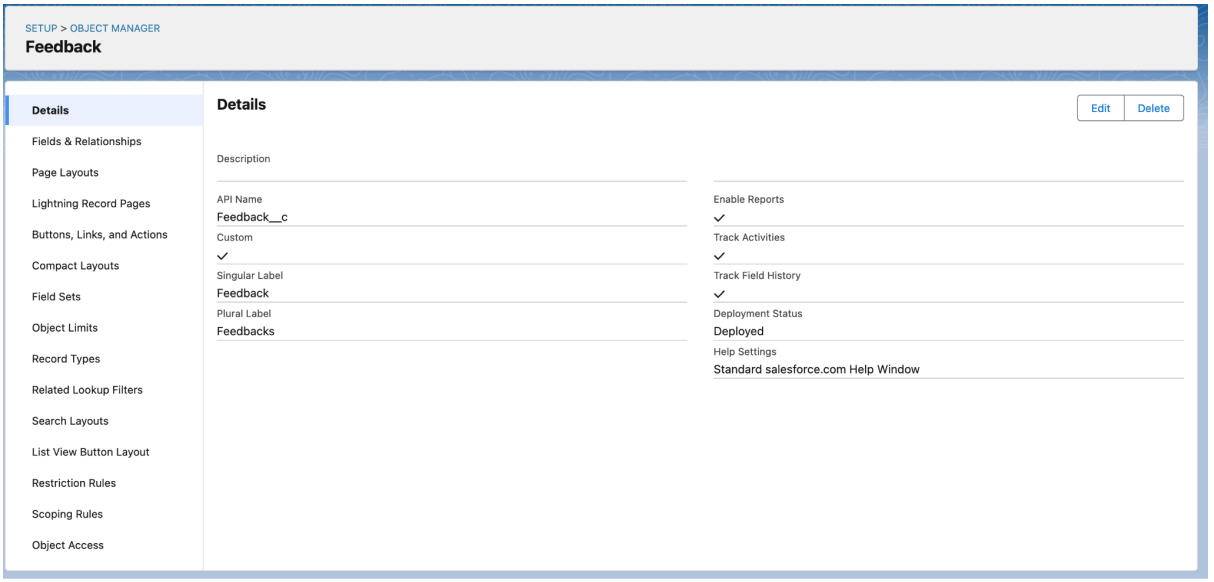
- Urban_Issue: Captures all reported urban issues, including details like status, type, location, and assigned department.



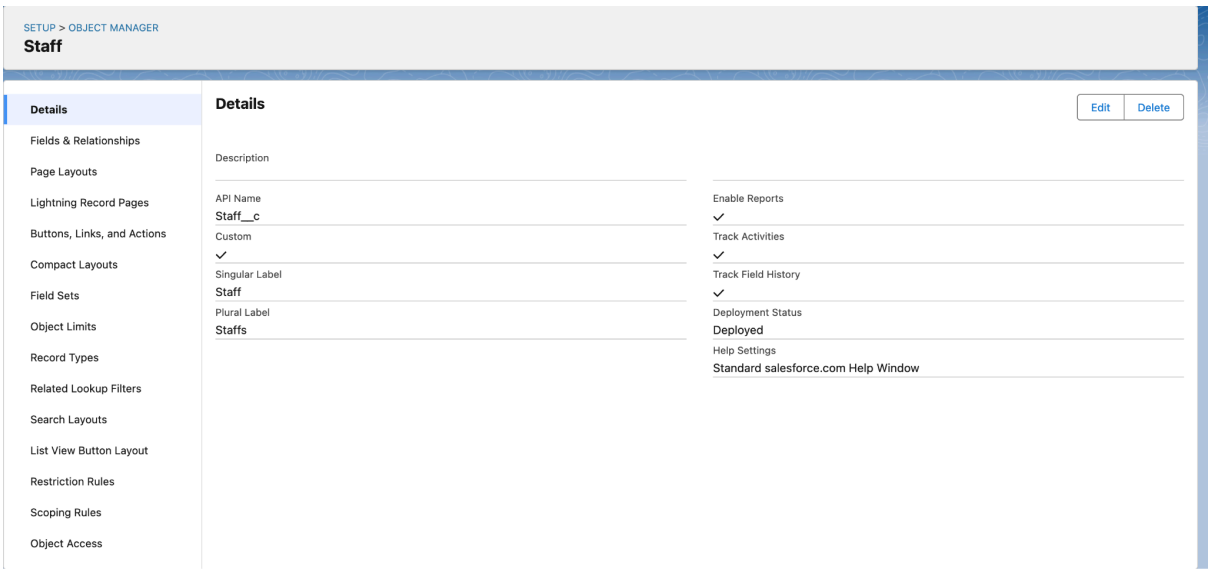
- Citizen: Stores information about citizens who report issues (name, contact details, government ID, etc.).



- Feedback: Collects feedback and ratings from citizens about the quality and timeliness of issue resolution.



- Staff: Helps to Track the staff information.



Fields & Relationships

Custom fields capture essential data, and lookup relationships create the necessary links between objects.

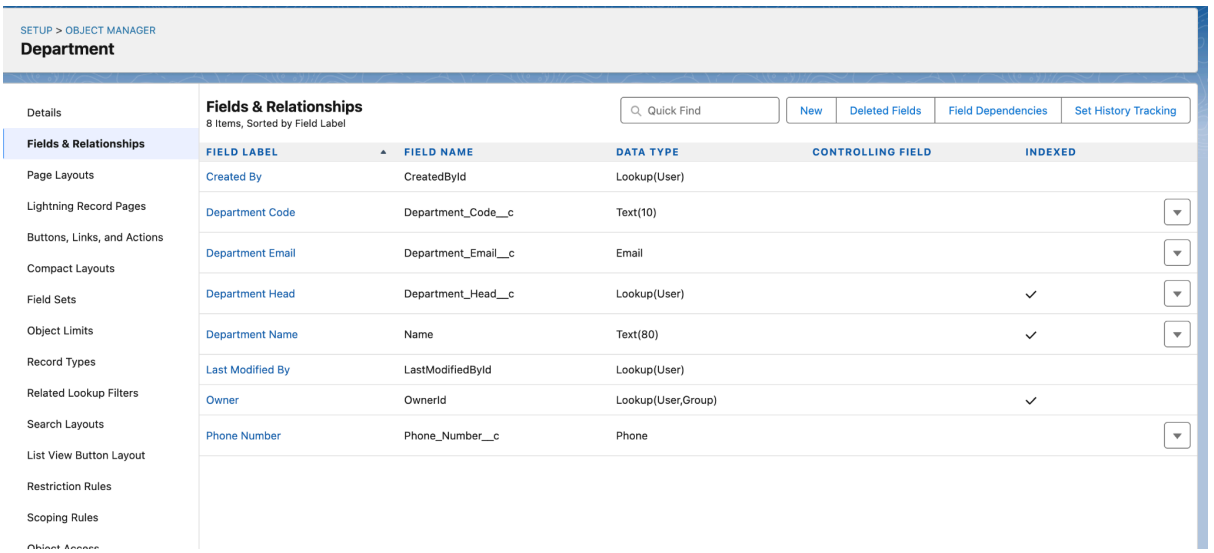
Relationships:

- Urban_Issue → Department: Lookup relationship; each urban issue is assigned to a department based on type/location.
- Urban_Issue → Citizen: (Future/optional) Lookup relationship, if issues are to be directly linked to the reporting citizen (not yet implemented).
- Feedback → Citizen: Lookup relationship; feedback is tied to the citizen who submitted it.
- Feedback → Urban_Issue: Lookup relationship; feedback is tied to the resolved urban issue.
- Department → Department Head: Lookup to User; identifies the supervisor/manager for each department.

Custom Fields:

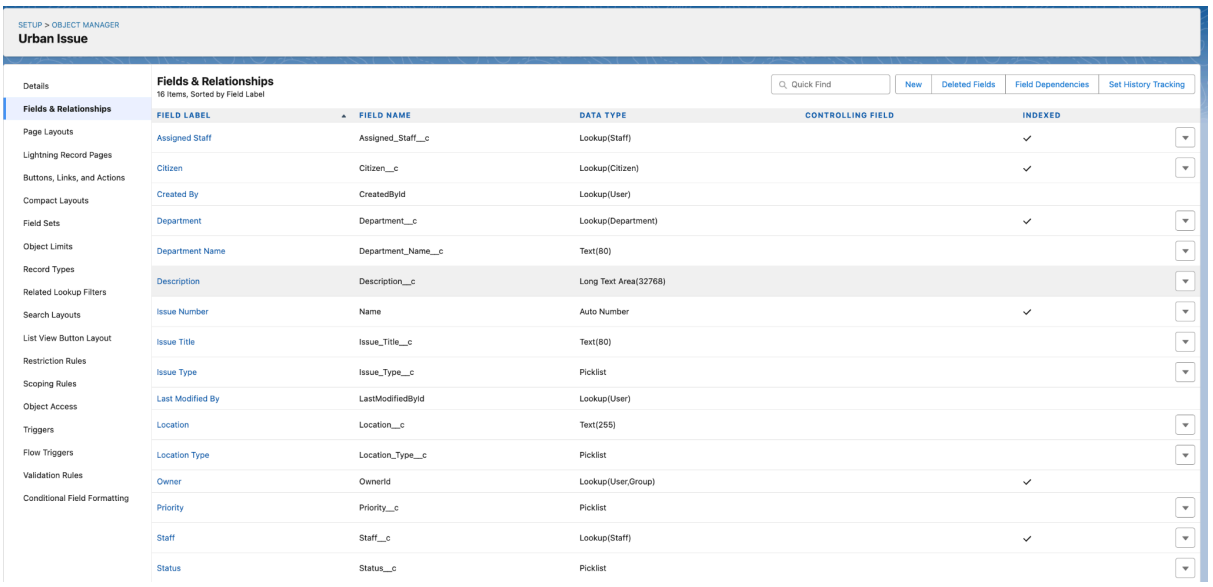
On Department:

- Department_Code (Text)
- Department_Email (Email)
- Department_Head (Lookup to User)
- Phone_Number (Phone)



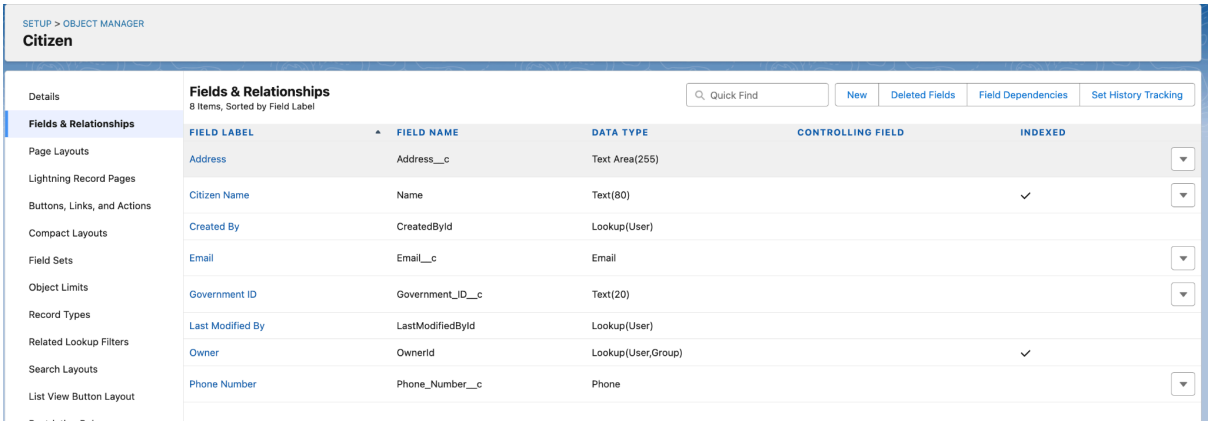
On Urban_Issue:

- Issue_Title (Text)
- Description (Long Text Area)
- Location (Text)
- Status (Picklist: New, In Progress, Resolved, Closed)
- Department (Lookup to Department)
- Department_Name (Text, for sharing rule workaround)



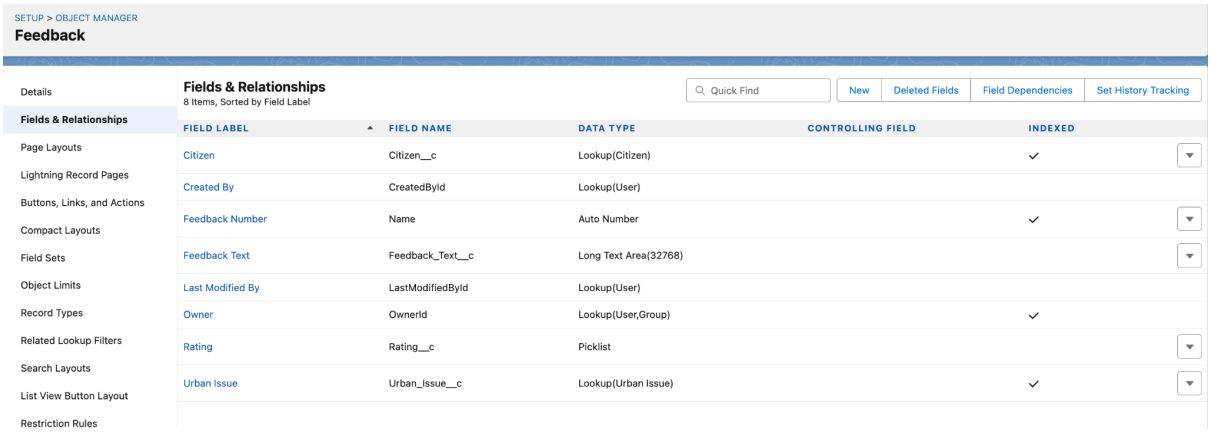
On Citizen:

- Email (Email)
- Phone_Number (Phone)
- Address (Text Area)
- Government_ID (Text)



On Feedback:

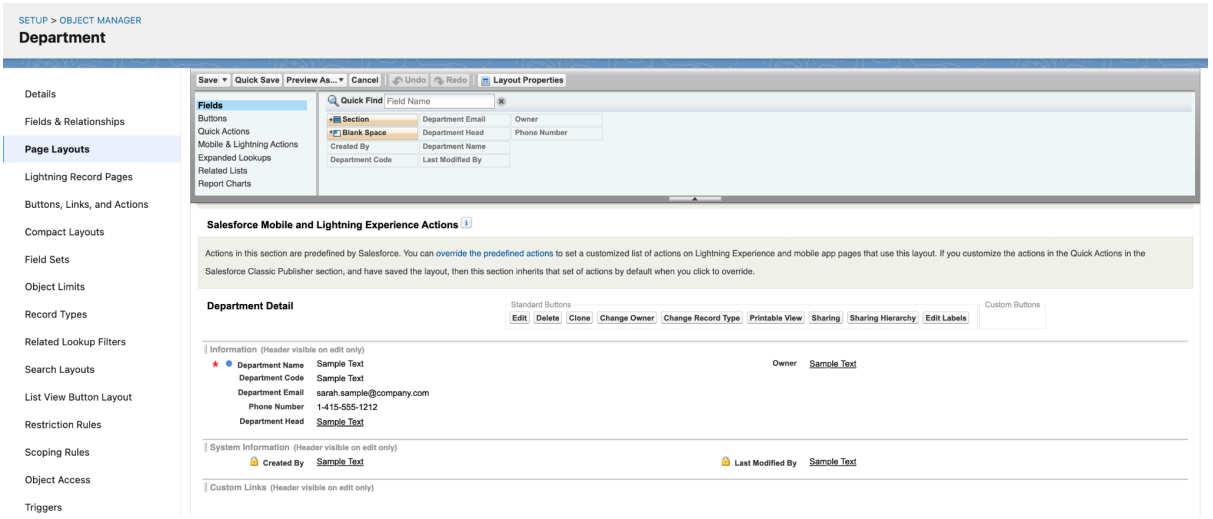
- Feedback_Text (Long Text Area)
- Rating (Picklist: 1, 2, 3, 4, 5)
- Citizen (Lookup to Citizen)
- Urban_Issue (Lookup to Urban_Issue)

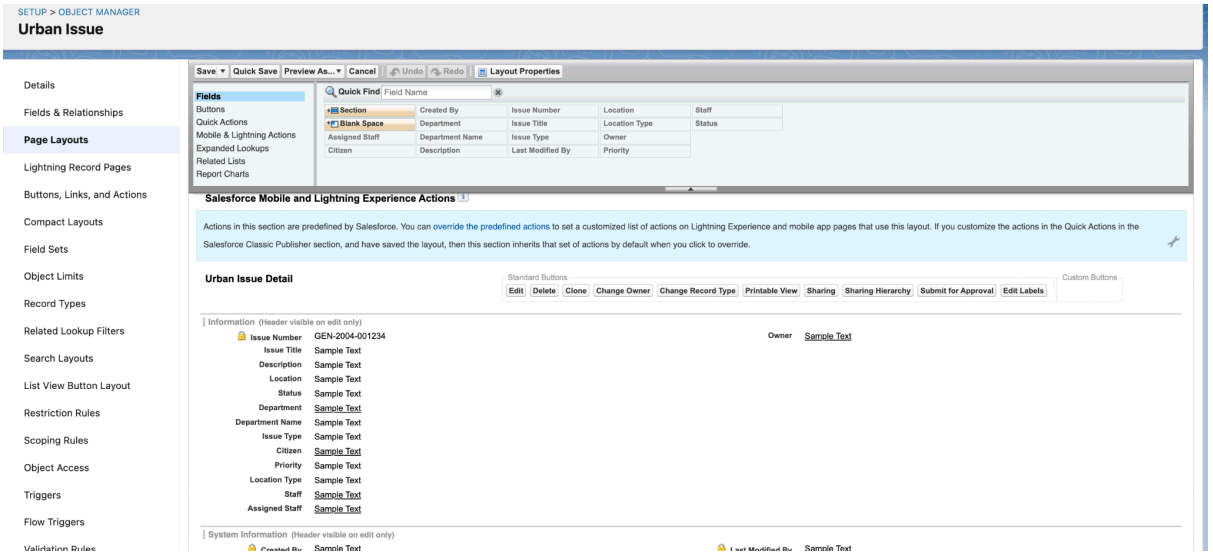


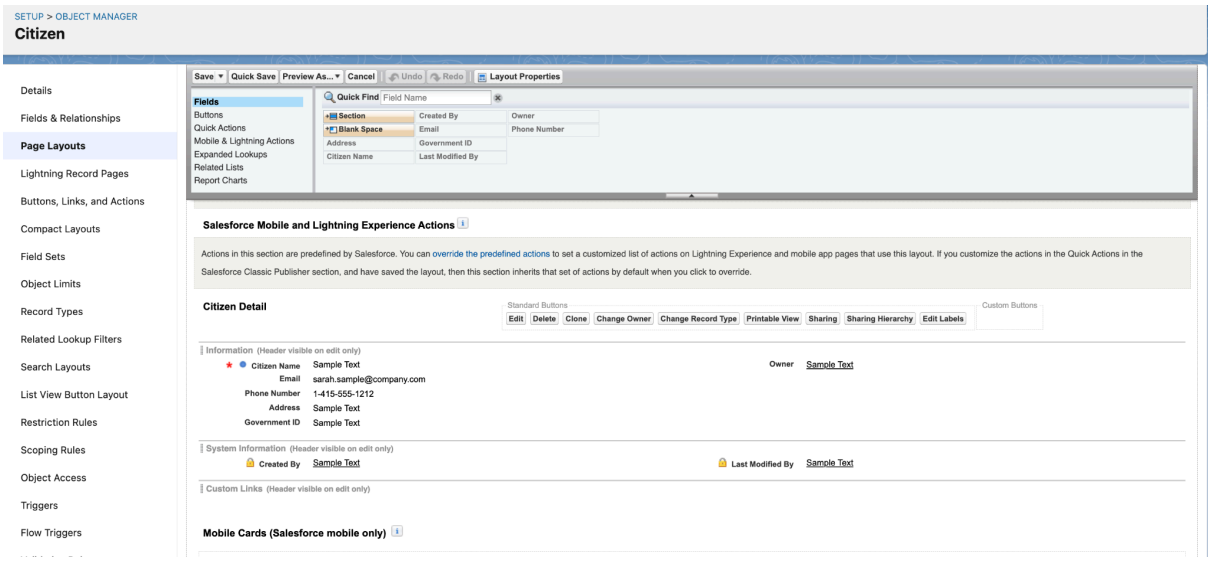
User Interface & Layouts

To ensure a user-friendly and efficient on-screen experience:

Page Layouts:
A dedicated page layout was created and customized for each custom object (Department, Urban Issue, Citizen, Feedback), organizing fields for easy data entry and review.







Compact Layouts:
A compact layout was configured for each custom object to display key fields in the highlights panel and on mobile.

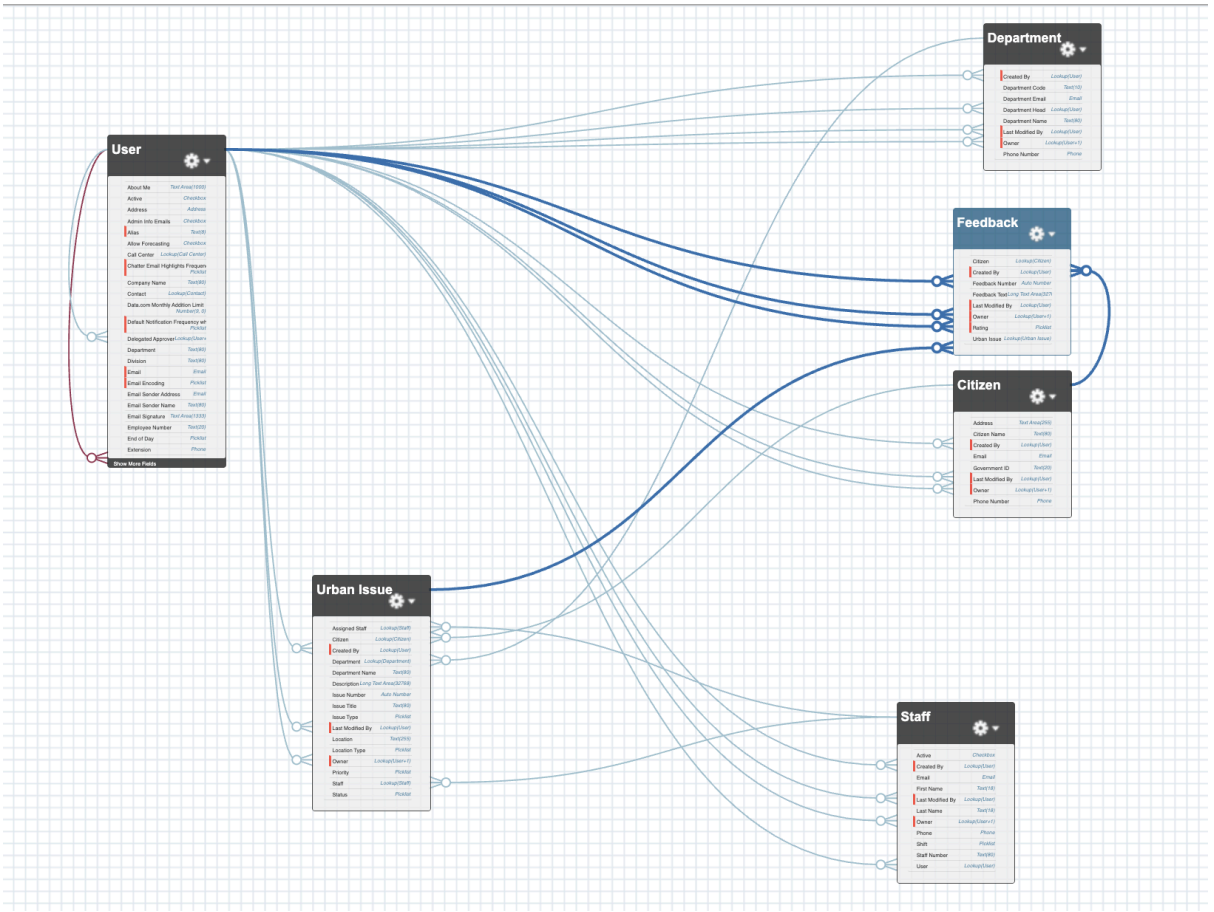
- *Example (Urban_Issue):* Compact layout includes Issue_Title, Status, and Department.

Record Types:
A single record type is used for each object at this stage.
(Future enhancement: Add record types to Urban_Issue for different issue categories or reporting sources.)

Data Architecture

Schema Builder:
The data model was visualized using Schema Builder, clearly showing:

- Department at the center with lookups from Urban_Issue.
- Citizen object connected to Feedback (and optionally Urban_Issue).
- Feedback object with lookups to both Citizen and Urban_Issue.



PHASE 4 - Process Automation (Admin)

The goal of this phase is to use Salesforce's declarative tools to automate the core business logic, enforce data quality, and handle notifications for the Urban Issue Management project.

---

1. Validation Rules

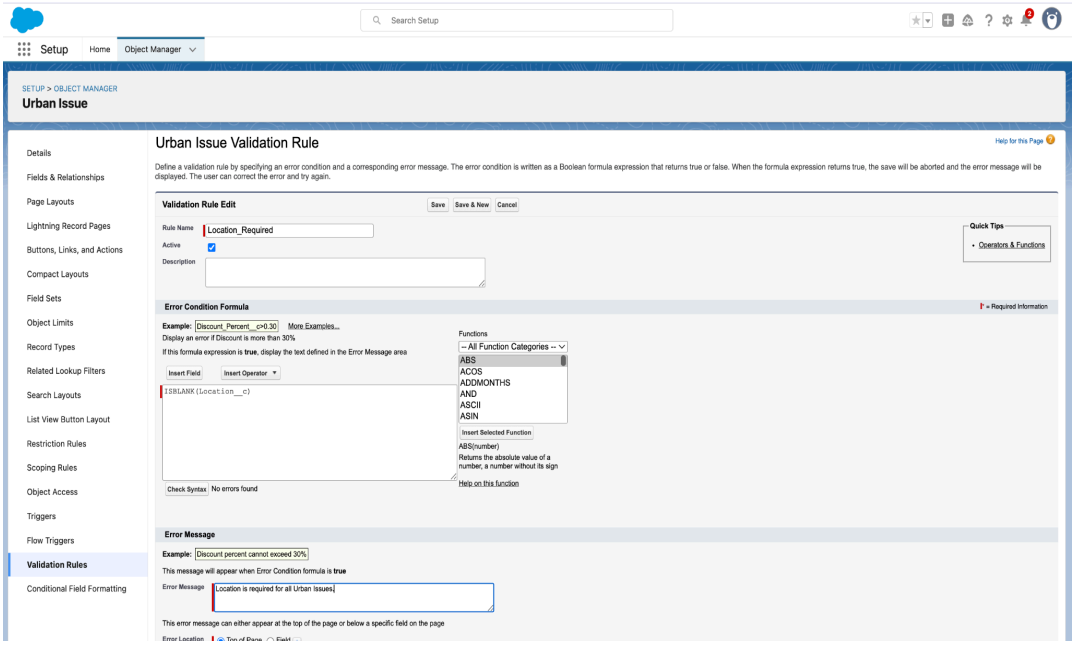
Purpose:
Ensure that Urban Issues have correct and complete data before saving.

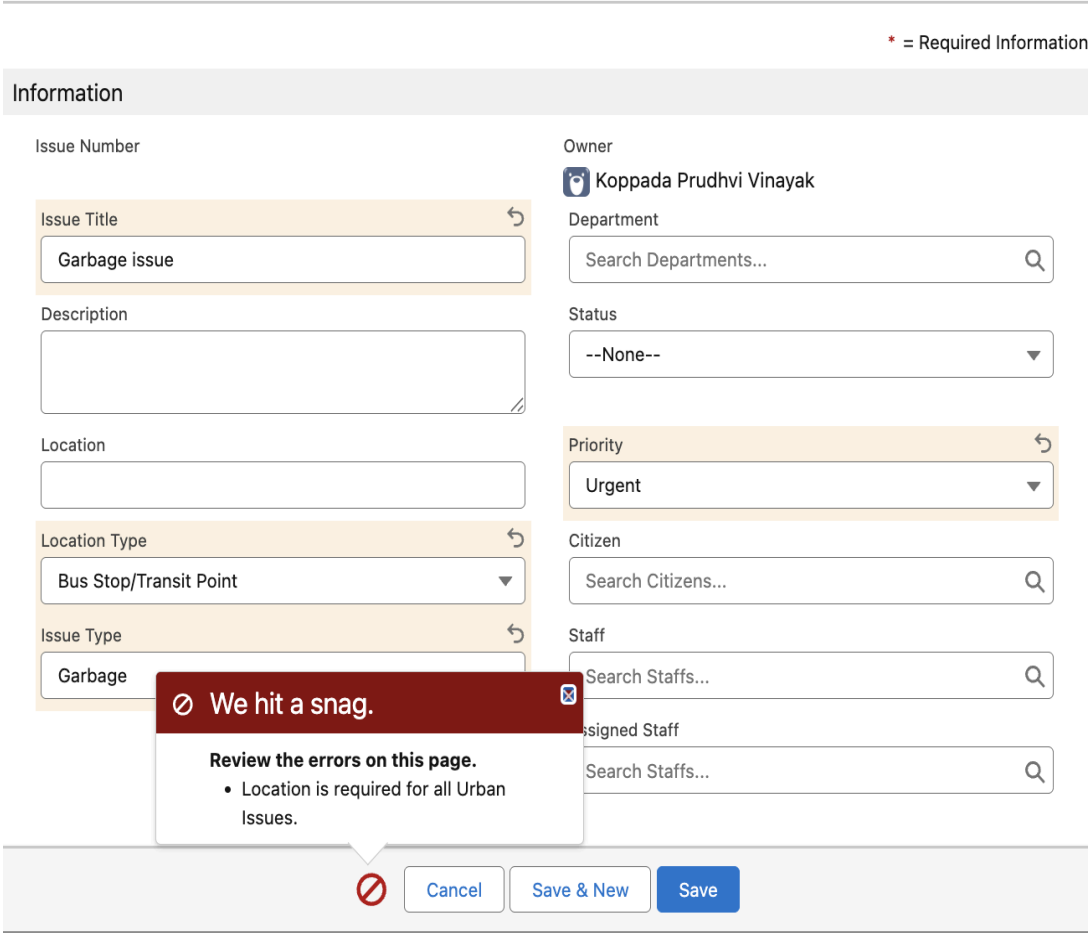
Object: Urban_Issue__c

Implemented Examples:

- Location Required
    - Formula: `ISBLANK(Location__c)`

- Error Message: "Location is required for all Urban Issues."

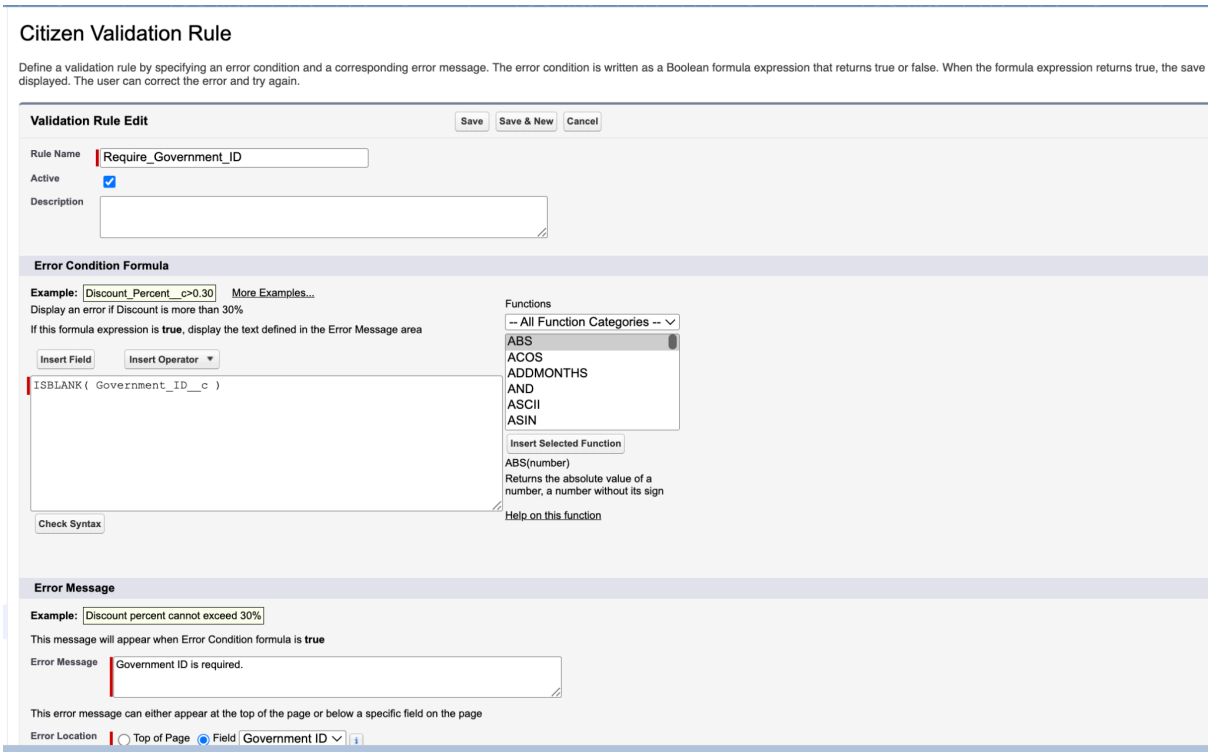


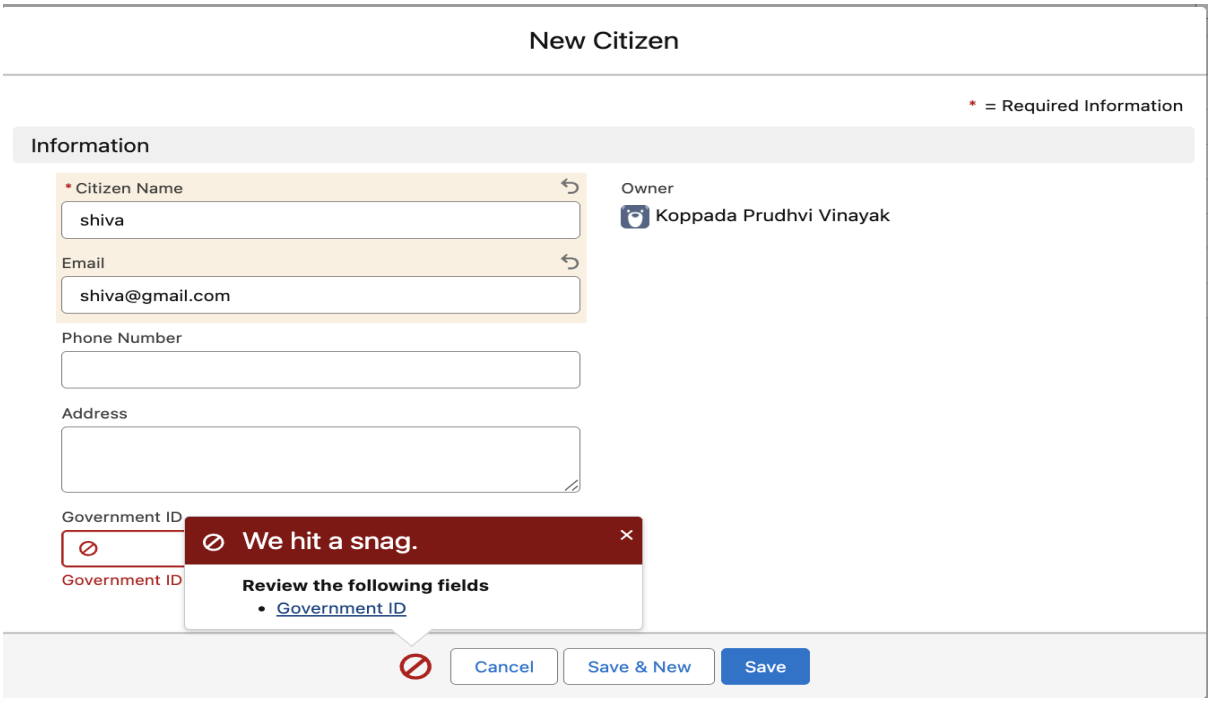


Object: citizen__c

Implemented Examples:

- Require_Government_ID
    - Formula: `ISBLANK( Government_ID__c )`
    - Error Message: “Government ID is required".





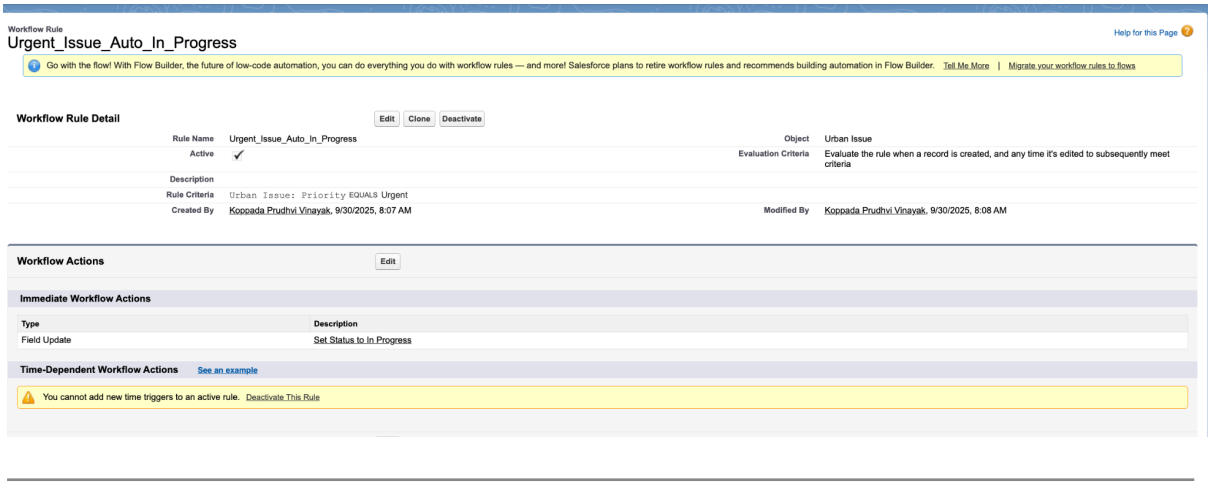
2. Workflow Rules

Purpose:
Automate simple field updates without code.

Object: Urban_Issue__c

Implemented Example:

- Auto-update Status to "In Progress" when Priority = Urgent
    - Rule Criteria: `Priority__c equals 'Urgent'`
    - Workflow Action: Field Update → Status = "In Progress"



3. Flow Builder (Record-Triggered Flow)

Tool: Flow Builder (Record-Triggered Flow)

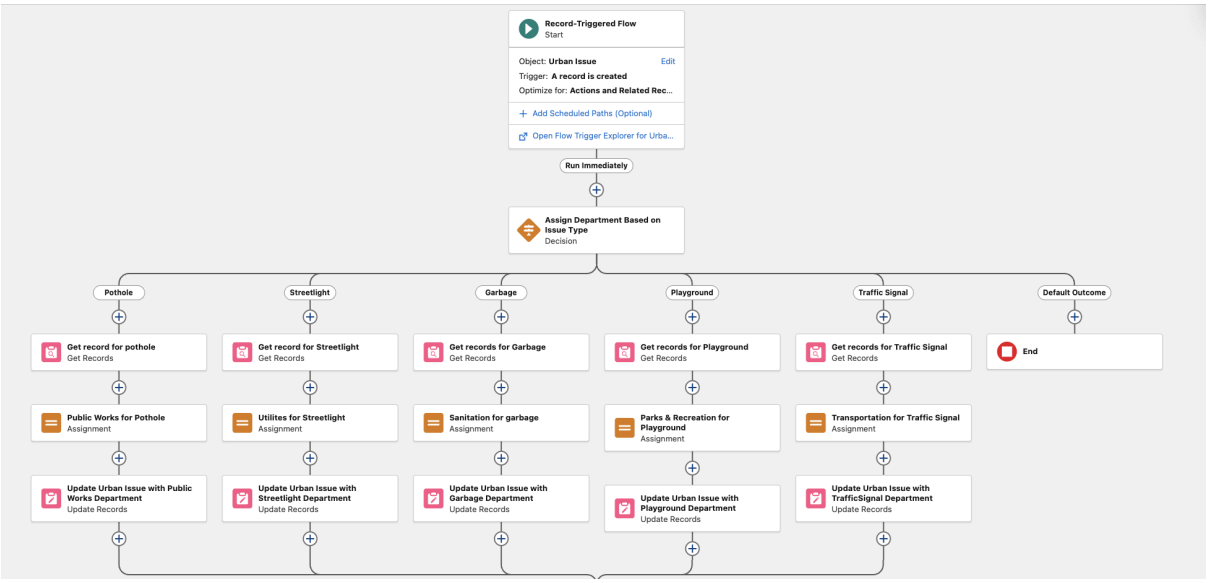
Objective:
Automate core business logic for Urban Issues.

Implemented Flows:

A. Auto-Assignment of Issues to Departments

- Trigger: When Urban Issue record is created.
- Logic:
    - Decision element checks Issue Type.
    - For each Issue Type, uses Get Records to find the correct Department.

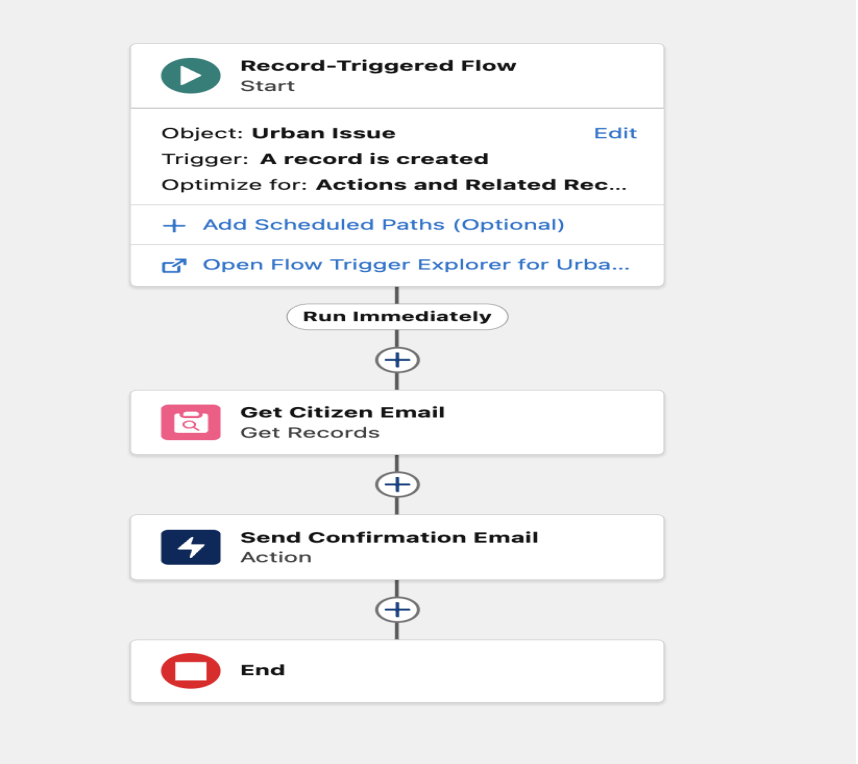
- Assignment sets Department lookup and Department Name on Urban Issue.
- Update Records element saves changes.



B. Email Notification to Citizen

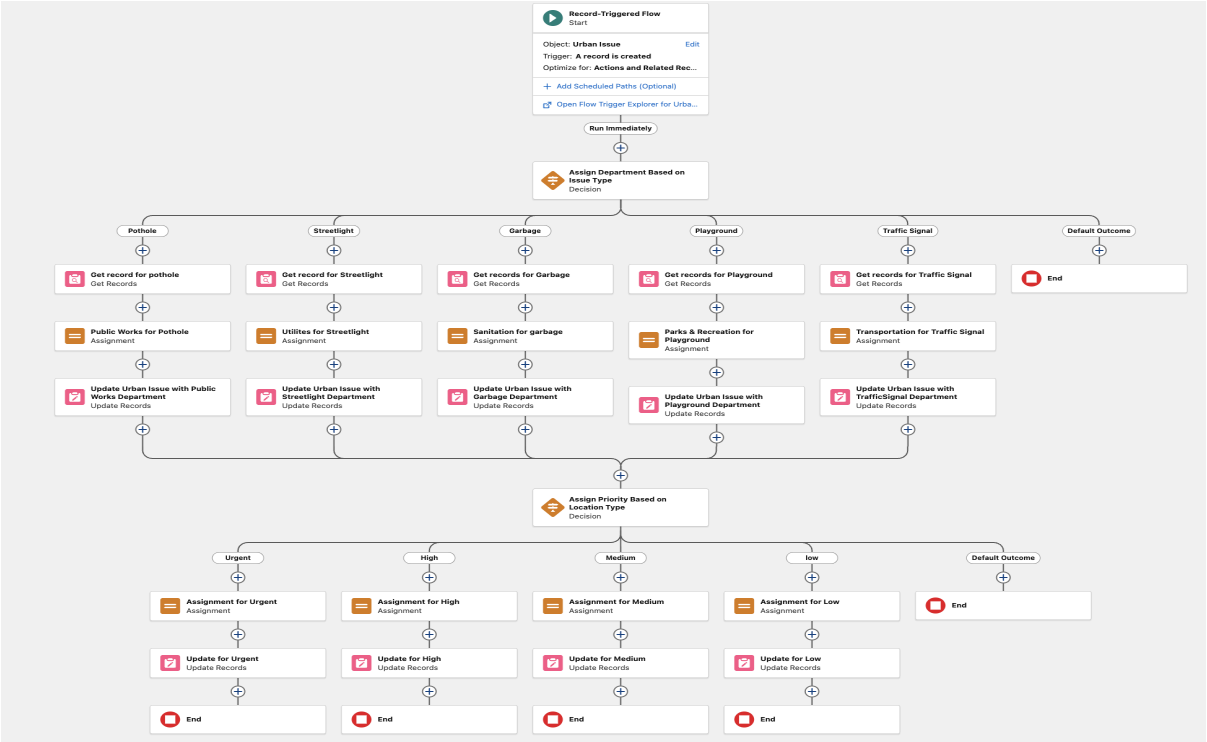
- Trigger: When Urban Issue record is created.
- Logic:

- Action: Sends a confirmation email (using Send Email or Email Alert) to the citizen’s email address with the Issue reference number.



C. Priority Assignment Using Location Type

- Trigger: After department assignment in the same flow.
- Logic:
  - Decision element evaluates Location Type.
  - Assignment sets Priority to "Urgent", "High", "Medium", or "Low" per the mapping.
  - Update Records saves the new Priority.



---

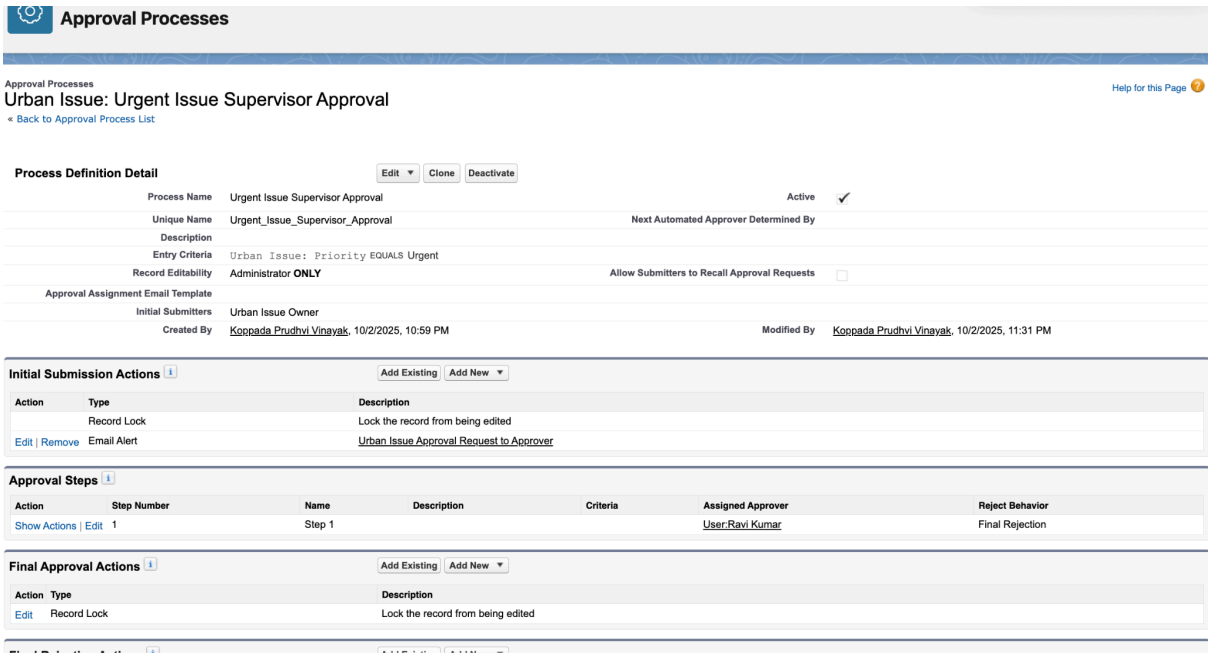
4. Approval Process

Purpose:
Require supervisor approval for critical Urban Issues.

Object: Urban_Issue__c

Implemented Example:

- Entry Criteria: `Priority = Urgent`
- Approver: Supervisor (selected automatically or by submitter)
- Actions:
    - On Submission: Notifies approver by email.
    - On Approval: Updates Status to "Approved".
    - On Rejection: Updates Status to "Rejected".



5. Email Alerts

Purpose:
Send email notifications as part of the Approval Process and Flows.

Implementation:

- Created Email Templates for approval requests and confirmation emails.
- Email Alerts configured in Approval Process:
    - On submission: Notifies approver.
    - On approval/rejection: Notifies submitter.
- Email sent to citizen on issue submission (via Flow).

> Thank you Koppada Prudhvi for submitting your urban issue report. Your reference number is UI-0031
> We will keep you updated as your issue progresses.
>
> Regards,
> City Infrastructure Team

---

6. Field Updates

Purpose:
Update key fields as part of automation.

Implementation:

- Used in Workflow Rule and Approval Process to update Status.
- Used in Flows to set Department, Department Name, and Priority.

---

7. Tasks

Purpose:
Create follow-up activities for urgent/high Urban Issues.

Status:
Attempted in Process Builder, but skipped for now due to cross-object assignment limitations.
Future Recommendation: Implement via Flow for advanced logic.

Phase 5: Apex Programming – Implementation Documentation

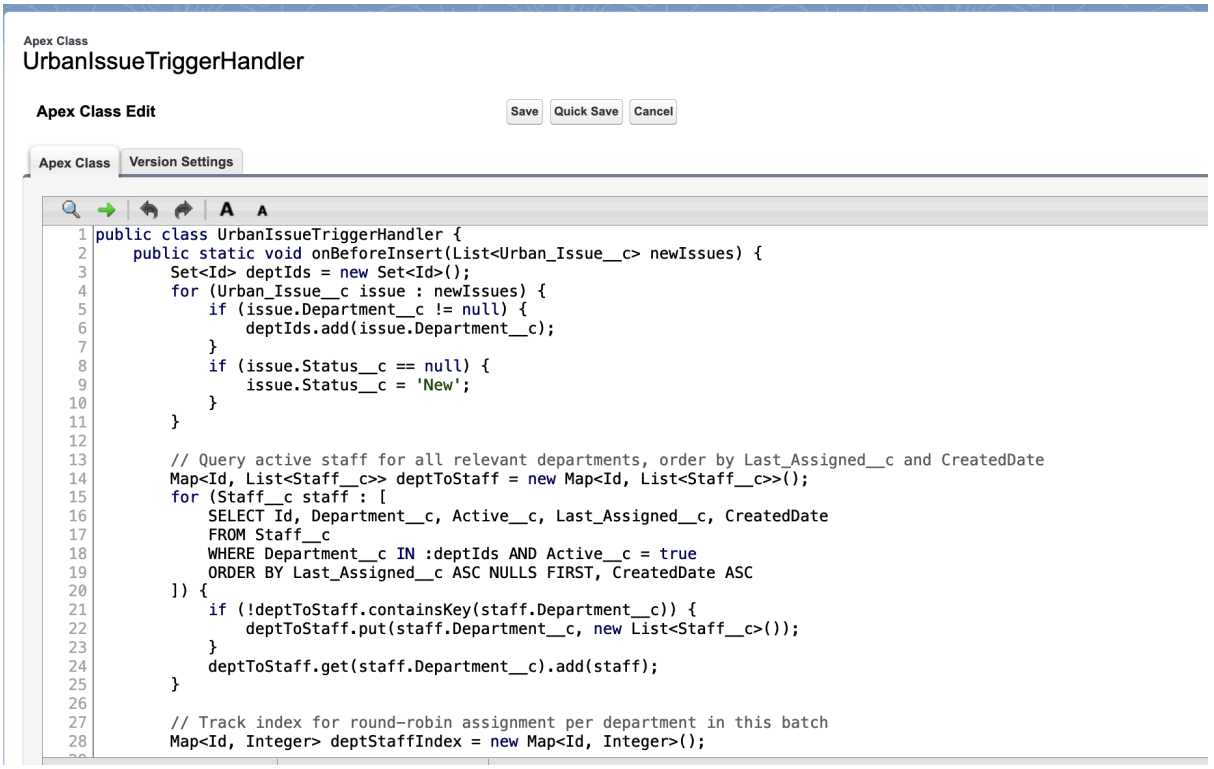
Overview

In Phase 5, we implemented advanced Salesforce automation for the Urban Issue tracking system using Apex programming. This phase covers the creation of custom logic for business processes that go beyond point-and-click tools, ensuring robust, scalable, and maintainable solutions.

What Has Been Implemented

Classes & Objects

- Implemented the `UrbanIssueTriggerHandler` Apex class to encapsulate all trigger-related business logic.
- Custom objects such as Urban Issue, Staff, and Department were used for data modeling.



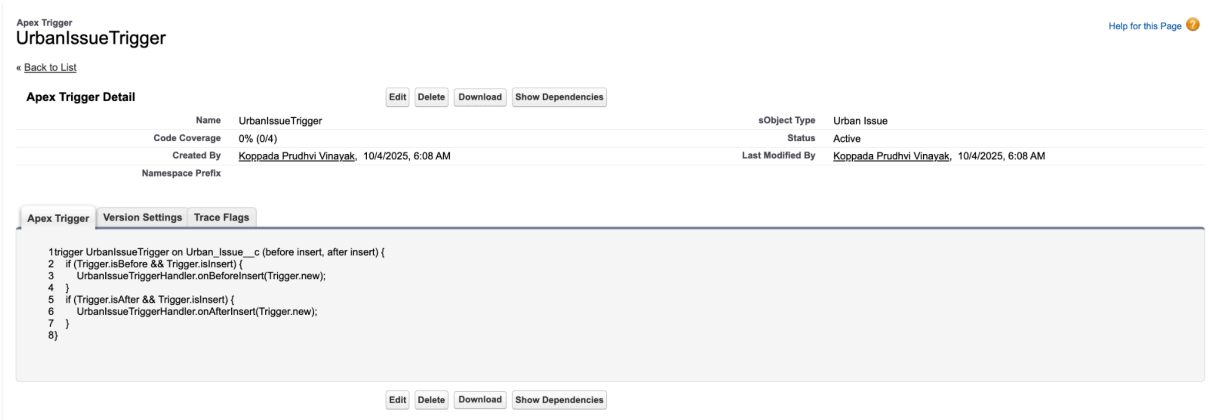

```apex
public class UrbanIssueTriggerHandler {
    public static void onBeforeInsert(List<Urban_Issue__c> newIssues) {
        Set<Id> deptIds = new Set<Id>();
        for (Urban_Issue__c issue : newIssues) {
            if (issue.Department__c != null) {
                deptIds.add(issue.Department__c);
            }
            if (issue.Status__c == null) {
                issue.Status__c = 'New';
            }
        }

        // Query active staff for all relevant departments, order by Last_Assigned__c and CreatedDate
        Map<Id, List<Staff__c>> deptToStaff = new Map<Id, List<Staff__c>>();
        for (Staff__c staff : [
            SELECT Id, Department__c, Active__c, Last_Assigned__c, CreatedDate
            FROM Staff__c
            WHERE Department__c IN :deptIds AND Active__c = true
            ORDER BY Last_Assigned__c ASC NULLS FIRST, CreatedDate ASC
        ]) {
            if (!deptToStaff.containsKey(staff.Department__c)) {
                deptToStaff.put(staff.Department__c, new List<Staff__c>());
            }
            deptToStaff.get(staff.Department__c).add(staff);
        }

        // Track index for round-robin assignment per department in this batch
        Map<Id, Integer> deptStaffIndex = new Map<Id, Integer>();
```

Apex Triggers

- Developed the `UrbanIssueTrigger` on the Urban Issue object to automate logic during record insert operations.
- The trigger uses both before and after insert contexts as required.

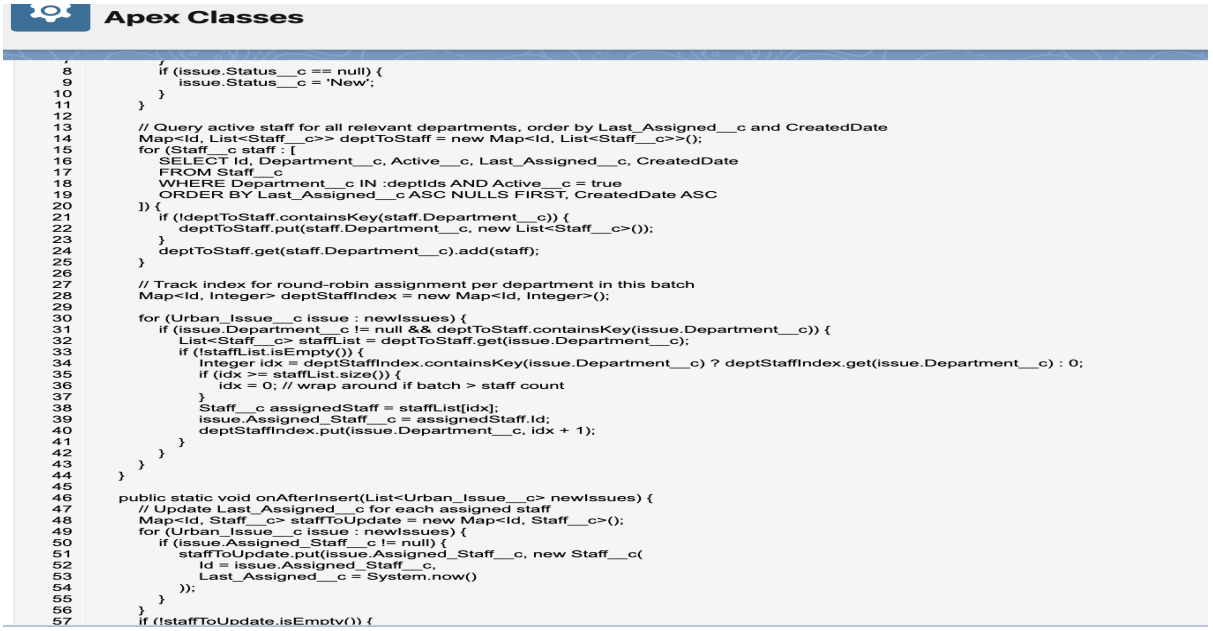


Trigger Design Pattern

- Adopted the trigger handler pattern by separating logic into the handler class, keeping triggers clean and maintainable.
- All trigger events delegate processing to the handler class methods.

SOQL (Salesforce Object Query Language)

- Utilized SOQL within the handler class to efficiently query related Staff and Department data needed for automation and assignment rules.



Collections: List, Set, Map

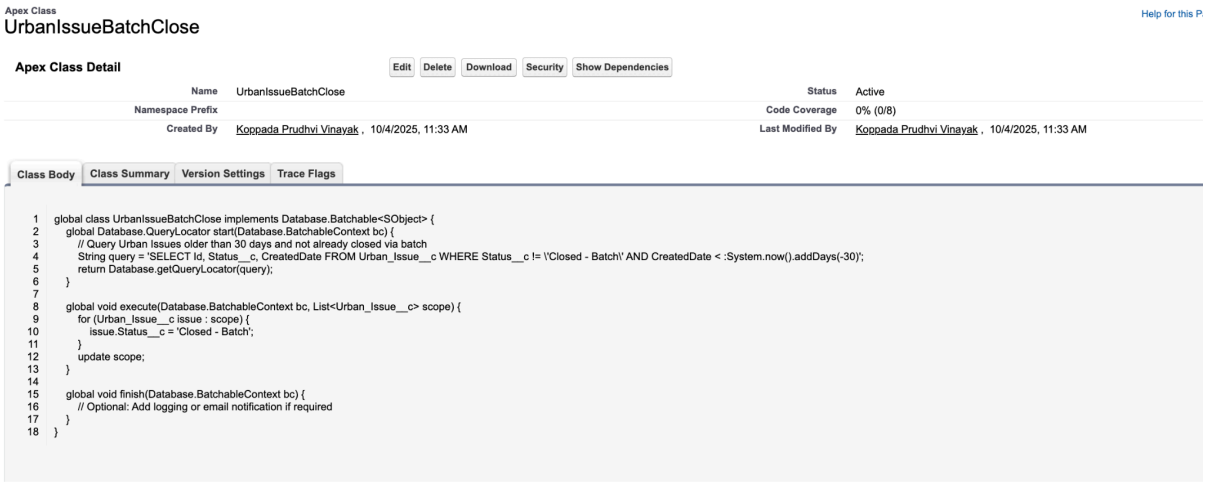
- Used Lists, Sets, and Maps in the handler for bulk processing, round-robin staff assignment, and efficient data handling.

Control Statements

- Incorporated for-loops, if-else statements, and other control flow structures to implement business logic in Apex.

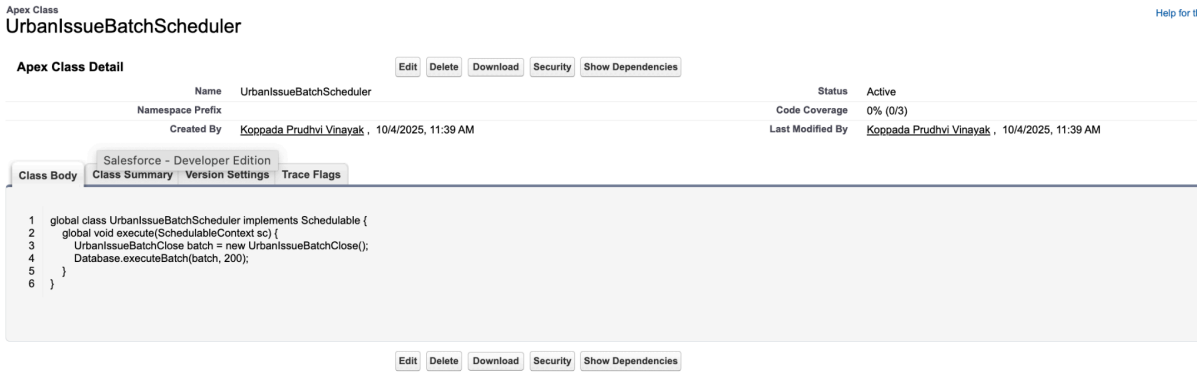
Batch Apex

- Created the `UrbanIssueBatchClose` Batch Apex class to process and update large volumes of Urban Issue records (e.g., closing issues older than 30 days in bulk).



Scheduled Apex

- Implemented the `UrbanIssueBatchScheduler` class, enabling the batch job to run automatically on a scheduled basis (e.g., weekly or nightly cleanup).

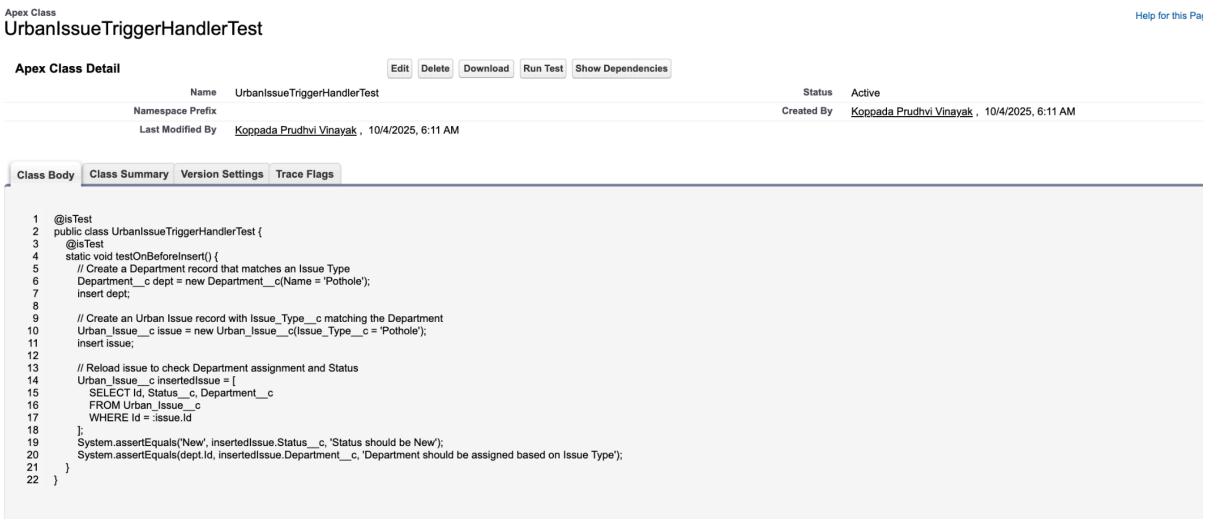


Exception Handling

- Applied try-catch blocks within handler methods to gracefully handle errors and maintain data integrity.

Test Classes

- Developed comprehensive test classes such as `UrbanIssueTriggerHandlerTest` to ensure logic coverage, data integrity, and successful deployment to production.



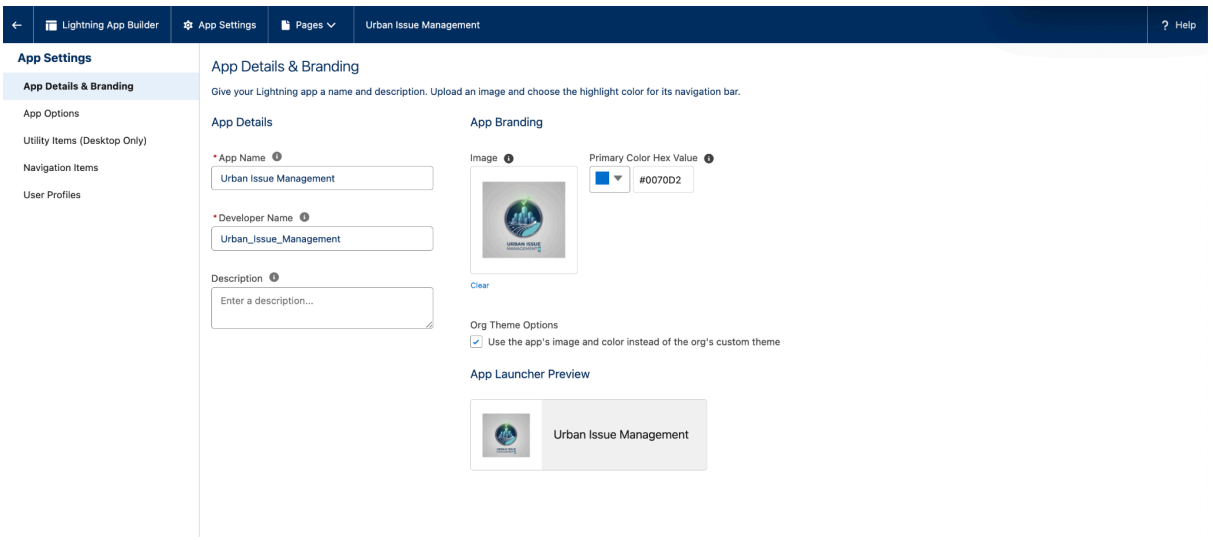
PHASE 6 - User Interface Development

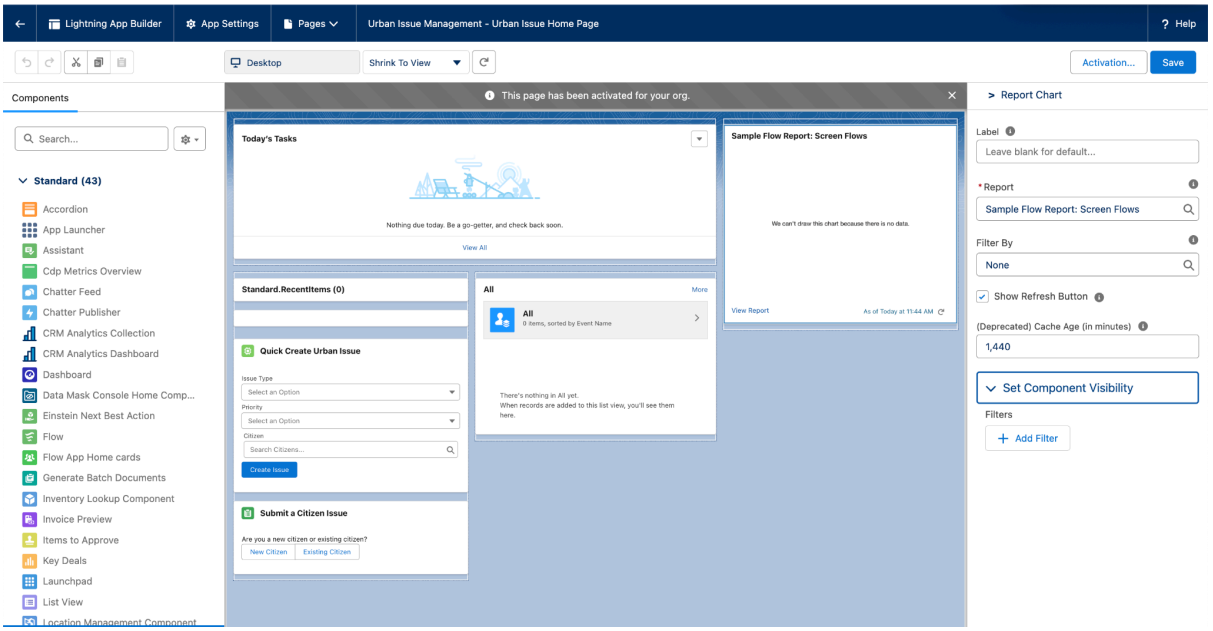
This phase focuses on building a responsive and intuitive user interface using the Lightning App Builder, custom record pages, tabs, the utility bar, and custom Lightning Web Components (LWC) for the Urban Issue Management system.

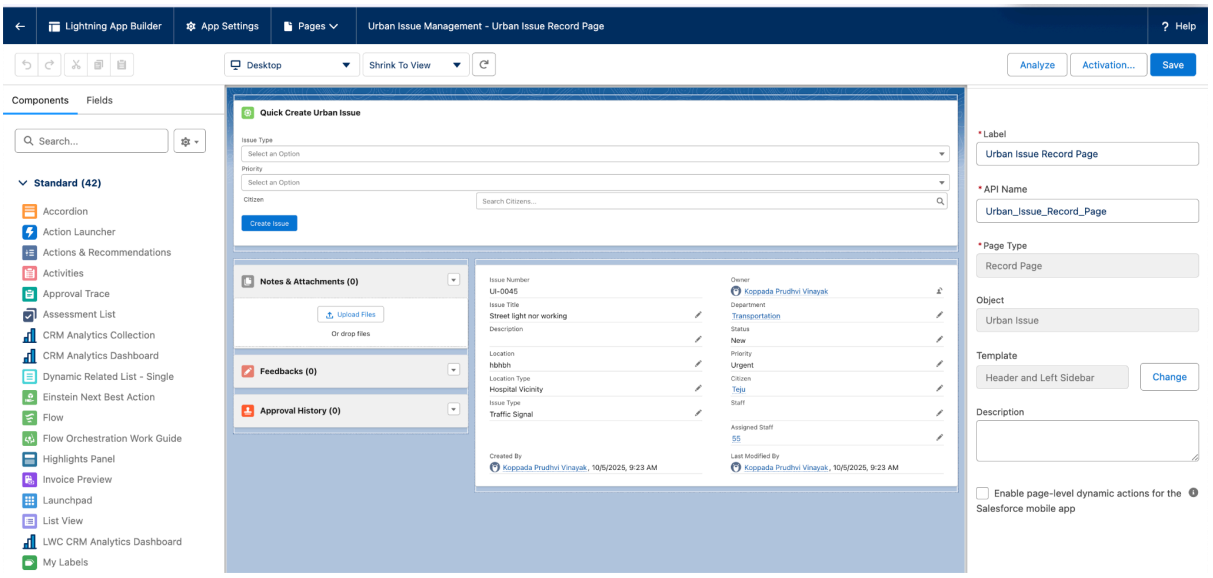
Lightning App Builder

The Lightning App Builder was the central tool for assembling the app's user interface declaratively.

- App Creation: A dedicated Salesforce Lightning App was created to serve as the workspace for managing Urban Issues and Citizens.
- Home Page & Record Pages: The Home Page and Record Pages for custom objects (Citizen, Urban Issue) were customized to include relevant components and layouts for optimal user experience.
- Utility Bar: The app's utility bar was set up to provide always-accessible tools. A custom utility item can be included, such as the Citizen Issue Submit component, for quick access to core functionality.

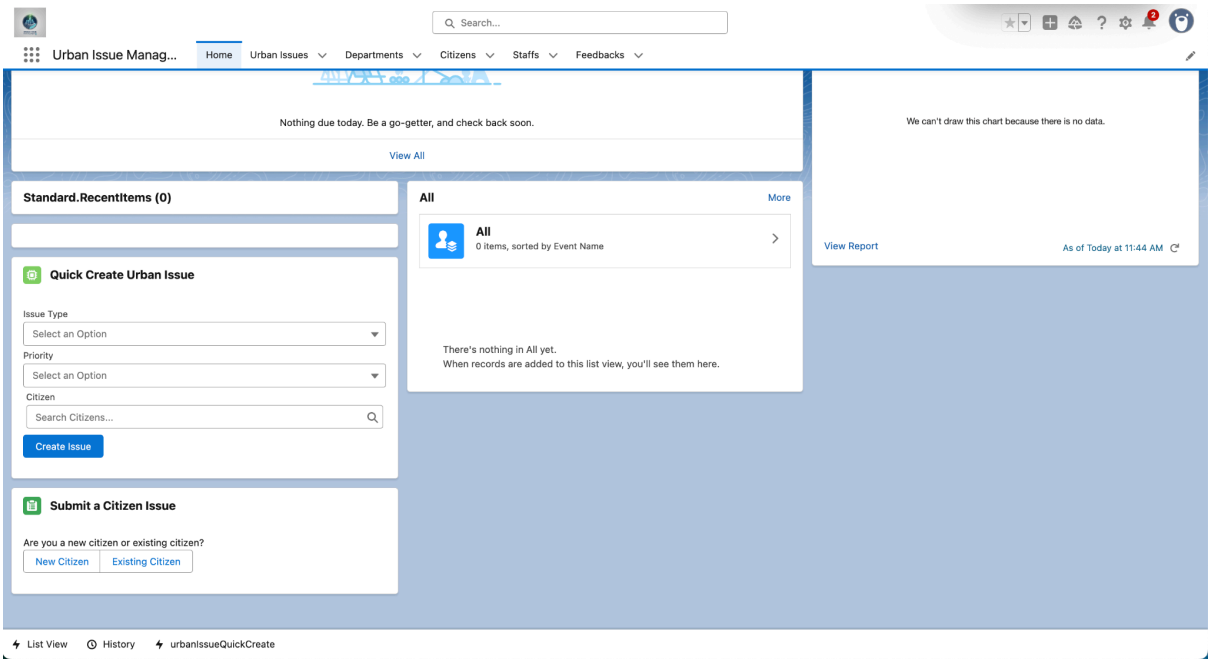

Lightning App Builder
App Settings
Pages
Urban Issue Management
Help
App Settings
App Details & Branding
App Options
Utility Items (Desktop Only)
Navigation Items
User Profiles
App Details & Branding
Give your Lightning app a name and description. Upload an image and choose the highlight color for its navigation bar.
App Details
App Branding
*App Name
Urban Issue Management
*Developer Name
Urban_Issue_Management
Description
Enter a description...
Image
Clear
Primary Color Hex Value
#0070D2
Org Theme Options
Use the app's image and color instead of the org's custom theme
App Launcher Preview
Urban Issue Management


Lightning App Builder
App Settings
Pages
Urban Issue Management - Urban Issue Home Page
Help
Desktop
Shrink To View
Activation...
Save
Components
This page has been activated for your org.
Report Chart
Search...
Standard (43)
Accordion
App Launcher
Assistant
Cdp Metrics Overview
Chatter Feed
Chatter Publisher
CRM Analytics Collection
CRM Analytics Dashboard
Dashboard
Data Mask Console Home Comp...
Einstein Next Best Action
Flow
Flow App Home cards
Generate Batch Documents
Inventory Lookup Component
Invoice Preview
Items to Approve
Key Deals
Launchpad
List View
Location Management Component
Today's Tasks
Nothing due today. Be a go-getter, and check back soon.
View All
Sample Flow Report: Screen Flows
We can't draw this chart because there is no data.
Standard.RecentItems (0)
All
More
0 items, sorted by Event Name
There's nothing in All yet.
When records are added to this list view, you'll see them here.
View Report
As of Today at 11:44 AM
Quick Create Urban Issue
Issue Type
Select an Option
Priority
Citizen
Search Citizens...
Create Issue
Submit a Citizen Issue
Are you a new citizen or existing citizen?
New Citizen
Existing Citizen
Label
Leave blank for default...
*Report
Sample Flow Report: Screen Flows
Filter By
None
Show Refresh Button
(Deprecated) Cache Age (in minutes)
1,440
Set Component Visibility
Filters
Add Filter


Lightning App Builder
App Settings
Pages
Urban Issue Management - Urban Issue Record Page
Help
Desktop
Shrink To View
Analyze
Activation...
Save
Components
Fields
Search...
Standard (42)
Accordion
Action Launcher
Actions & Recommendations
Activities
Approval Trace
Assessment List
CRM Analytics Collection
CRM Analytics Dashboard
Dynamic Related List - Single
Einstein Next Best Action
Flow
Flow Orchestration Work Guide
Highlights Panel
Invoice Preview
Launchpad
List View
LWC CRM Analytics Dashboard
My Labels
Quick Create Urban Issue
Issue Type
Select an Option
Priority
Citizen
Search Citizens...
Create Issue
Notes & Attachments (0)
Upload Files
Or drop files
Feedbacks (0)
Approval History (0)
Issue Number
UI-0045
Issue Title
Street light nor working
Description
Location
hbhbh
Location Type
Hospital Vicinity
Issue Type
Traffic Signal
Owner
Koppada Prudhvi Vinayak
Department
Transportation
Status
New
Priority
Urgent
Citizen
Teja
Staff
Assigned Staff
55
Created By
Koppada Prudhvi Vinayak, 10/5/2025, 9:23 AM
Last Modified By
Koppada Prudhvi Vinayak, 10/5/2025, 9:23 AM
*Label
Urban Issue Record Page
*API Name
Urban_Issue_Record_Page
*Page Type
Record Page
Object
Urban Issue
Template
Header and Left Sidebar
Change
Description
Enable page-level dynamic actions for the Salesforce mobile app

Record Pages & Tabs

- Enhanced Record Pages: Custom Record Pages were configured for the Citizen and Urban Issue objects. These pages display key fields, related lists, and surface custom LWCs for data entry or quick actions.
- Custom Tabs: Tabs for "Citizens," "Urban Issues," and other objects were added to the app’s navigation, making it easy for users to switch between major data entities..

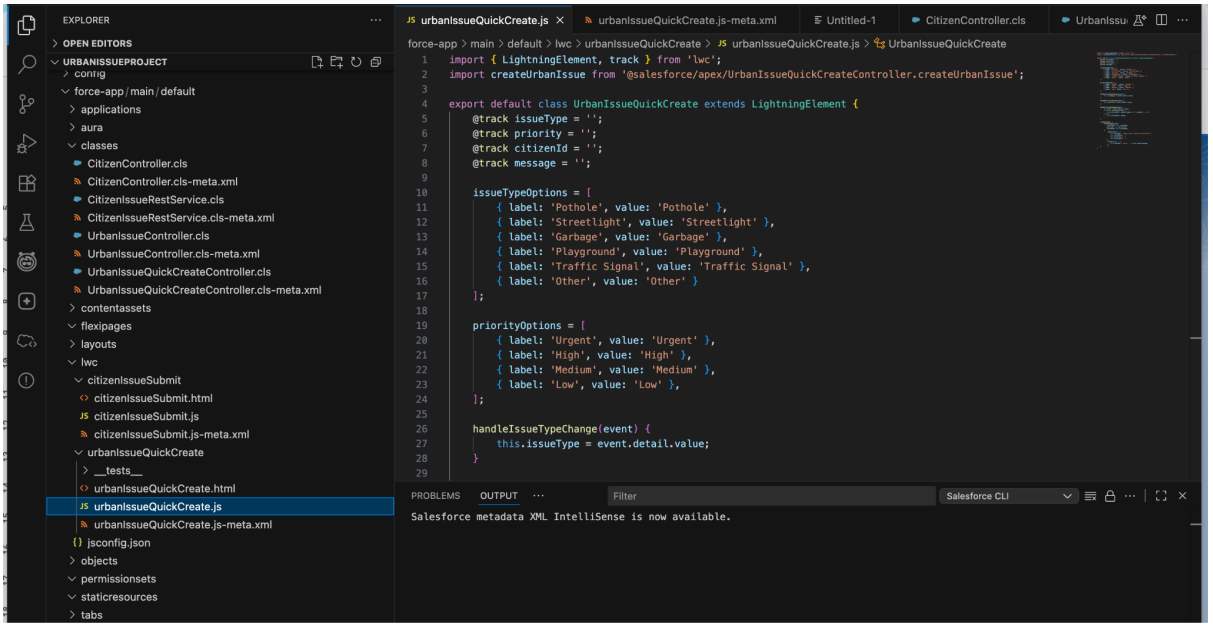


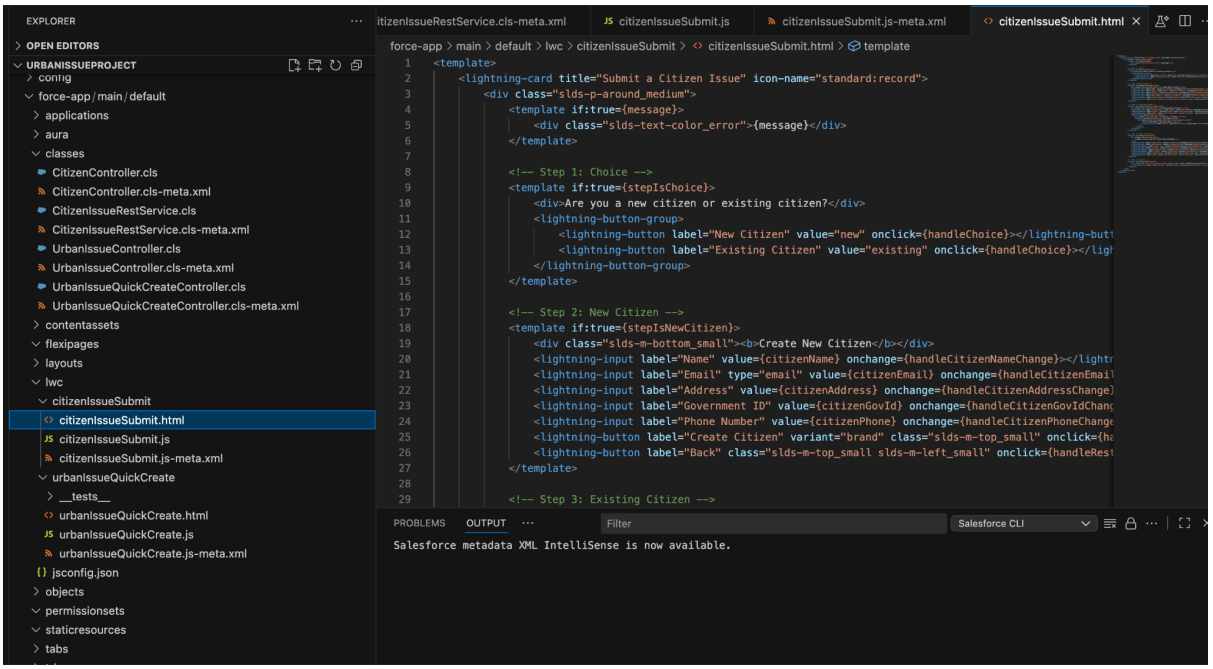
Lightning Web Components (LWC)

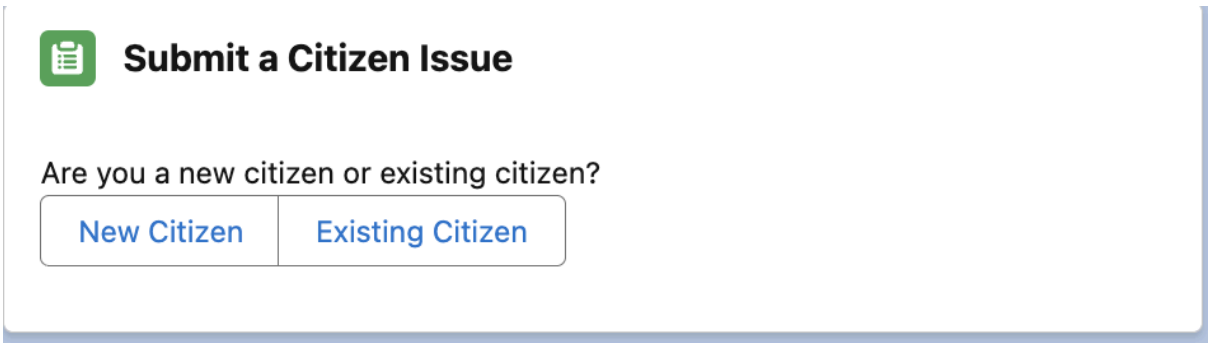
Custom Lightning Web Components were developed to streamline and enhance user interactions:

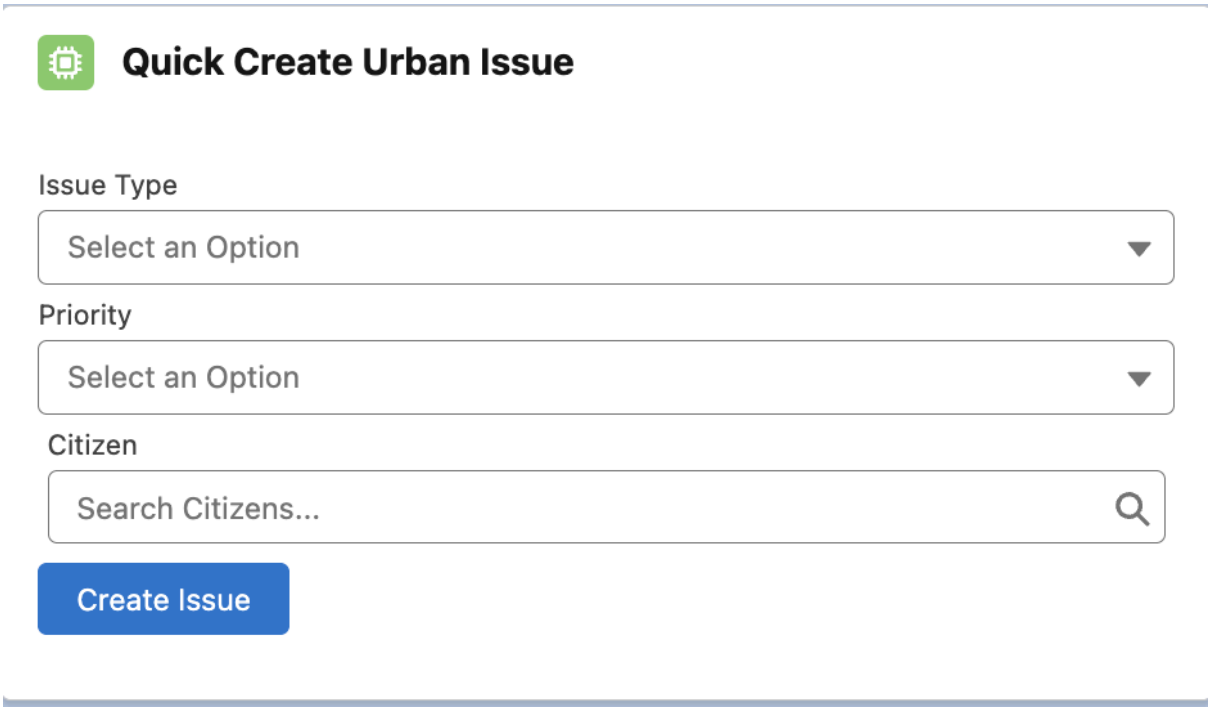
- Citizen Issue Submit:
  A multi-step LWC that guides users through either creating a new Citizen or selecting an existing one, then submitting a new Urban Issue. The form dynamically presents fields for both entities and performs validation before saving.
- Apex with LWC:
  These LWCs communicate with Apex controllers to create, search, and relate records. Apex methods are @AuraEnabled and securely handle DML operations.

- Imperative Apex Calls:
  The LWC uses imperative Apex calls for actions such as creating a Citizen or Urban Issue, and for searching existing Citizens based on input.
- Picklist Support:
  The Urban Issue section of the LWC features dynamic picklists for fields like Issue Type and Location Type, mirroring business requirements.





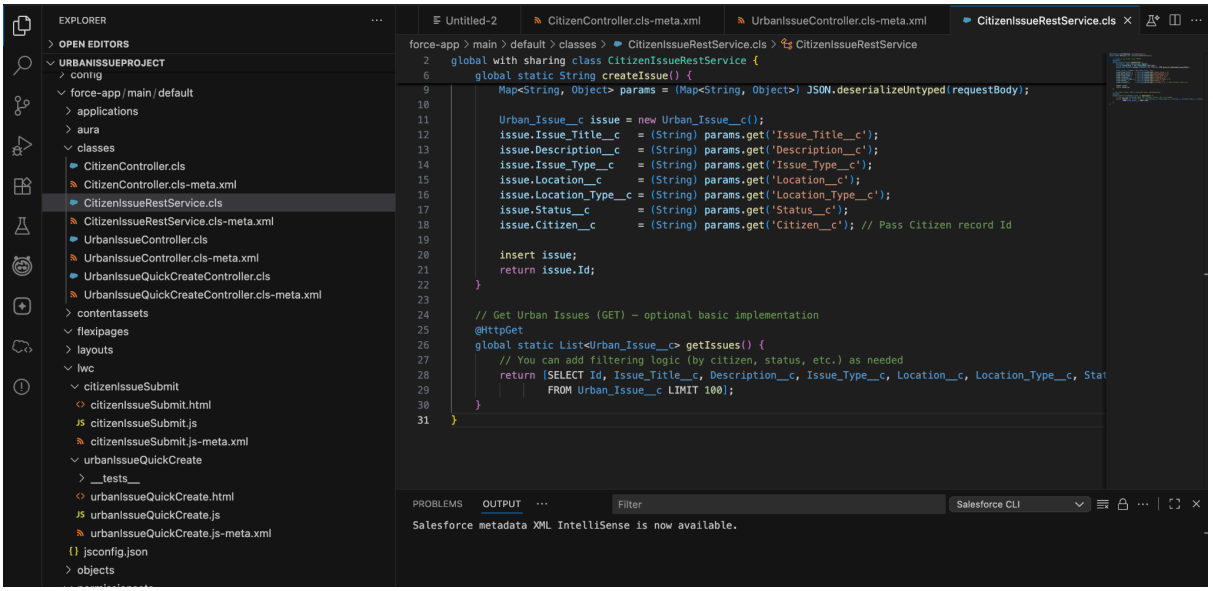


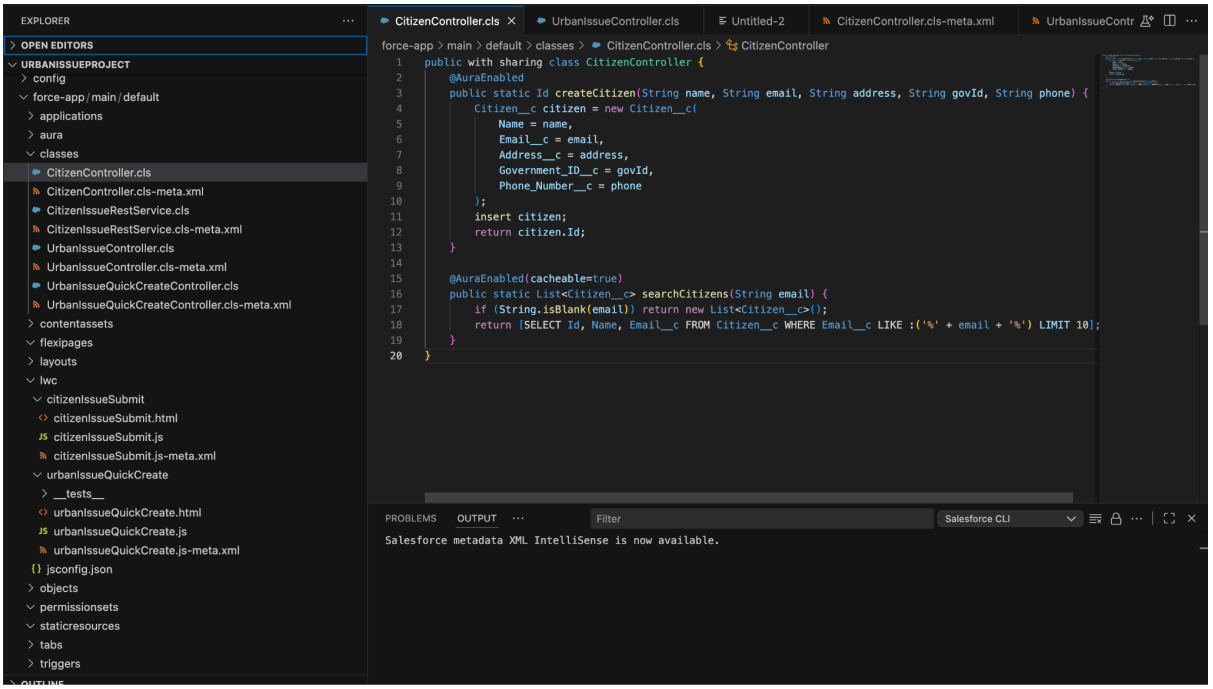


Utility Bar

- Utility Bar is set up in the App Manager for your custom app.
- Standard utilities (like History, Notes) and/or custom LWCs (like Citizen Issue Submit) can be accessed at the bottom of the screen.

Apex with LWC & Imperative Apex Calls


EXPLORER
OPEN EDITORS
URBANISSUEPROJECT
config
force-app / main / default
applications
aura
classes
CitizenController.cls
CitizenController.cls-meta.xml
CitizenIssueRestService.cls
CitizenIssueRestService.cls-meta.xml
UrbanIssueController.cls
UrbanIssueController.cls-meta.xml
UrbanIssueQuickCreateController.cls
UrbanIssueQuickCreateController.cls-meta.xml
contentassets
flexipages
layouts
lwc
citizenIssueSubmit
citizenIssueSubmit.html
citizenIssueSubmit.js
citizenIssueSubmit.js-meta.xml
urbanIssueQuickCreate
__tests__
urbanIssueQuickCreate.html
urbanIssueQuickCreate.js
urbanIssueQuickCreate.js-meta.xml
jsconfig.json
objects
Untitled-2
CitizenController.cls-meta.xml
UrbanIssueController.cls-meta.xml
CitizenIssueRestService.cls
force-app › main › default › classes › CitizenIssueRestService.cls › CitizenIssueRestService
global with sharing class CitizenIssueRestService {
global static String createIssue() {
Map<String, Object> params = (Map<String, Object>) JSON.deserializeUntyped(requestBody);
Urban_Issue__c issue = new Urban_Issue__c();
issue.Issue_Title__c = (String) params.get('Issue_Title__c');
issue.Description__c = (String) params.get('Description__c');
issue.Issue_Type__c = (String) params.get('Issue_Type__c');
issue.Location__c = (String) params.get('Location__c');
issue.Location_Type__c = (String) params.get('Location_Type__c');
issue.Status__c = (String) params.get('Status__c');
issue.Citizen__c = (String) params.get('Citizen__c'); // Pass Citizen record Id
insert issue;
return issue.Id;
}
// Get Urban Issues (GET) – optional basic implementation
@HttpGet
global static List<Urban_Issue__c> getIssues() {
// You can add filtering logic (by citizen, status, etc.) as needed
return [SELECT Id, Issue_Title__c, Description__c, Issue_Type__c, Location__c, Location_Type__c, Stat
FROM Urban_Issue__c LIMIT 100];
}
}
PROBLEMS
OUTPUT
Filter
Salesforce CLI
Salesforce metadata XML IntelliSense is now available.


EXPLORER
OPEN EDITORS
URBANISSUEPROJECT
config
force-app / main / default
applications
aura
classes
CitizenController.cls
CitizenController.cls-meta.xml
CitizenIssueRestService.cls
CitizenIssueRestService.cls-meta.xml
UrbanIssueController.cls
UrbanIssueController.cls-meta.xml
UrbanIssueQuickCreateController.cls
UrbanIssueQuickCreateController.cls-meta.xml
contentassets
flexipages
layouts
lwc
citizenIssueSubmit
citizenIssueSubmit.html
citizenIssueSubmit.js
citizenIssueSubmit.js-meta.xml
urbanIssueQuickCreate
__tests__
urbanIssueQuickCreate.html
urbanIssueQuickCreate.js
urbanIssueQuickCreate.js-meta.xml
jsconfig.json
objects
permissionsets
staticresources
tabs
triggers
CitizenController.cls
UrbanIssueController.cls
Untitled-2
CitizenController.cls-meta.xml
UrbanIssueContr
force-app › main › default › classes › CitizenController.cls › CitizenController
public with sharing class CitizenController {
@AuraEnabled
public static Id createCitizen(String name, String email, String address, String govId, String phone) {
Citizen__c citizen = new Citizen__c(
Name = name,
Email__c = email,
Address__c = address,
Government_ID__c = govId,
Phone_Number__c = phone
);
insert citizen;
return citizen.Id;
}
@AuraEnabled(cacheable=true)
public static List<Citizen__c> searchCitizens(String email) {
if (String.isBlank(email)) return new List<Citizen__c>();
return [SELECT Id, Name, Email__c FROM Citizen__c WHERE Email__c LIKE :('%' + email + '%') LIMIT 10];
}
}
PROBLEMS
OUTPUT
Filter
Salesforce CLI
Salesforce metadata XML IntelliSense is now available.

Phase 7: Integration & External Access — Implementation Documentation

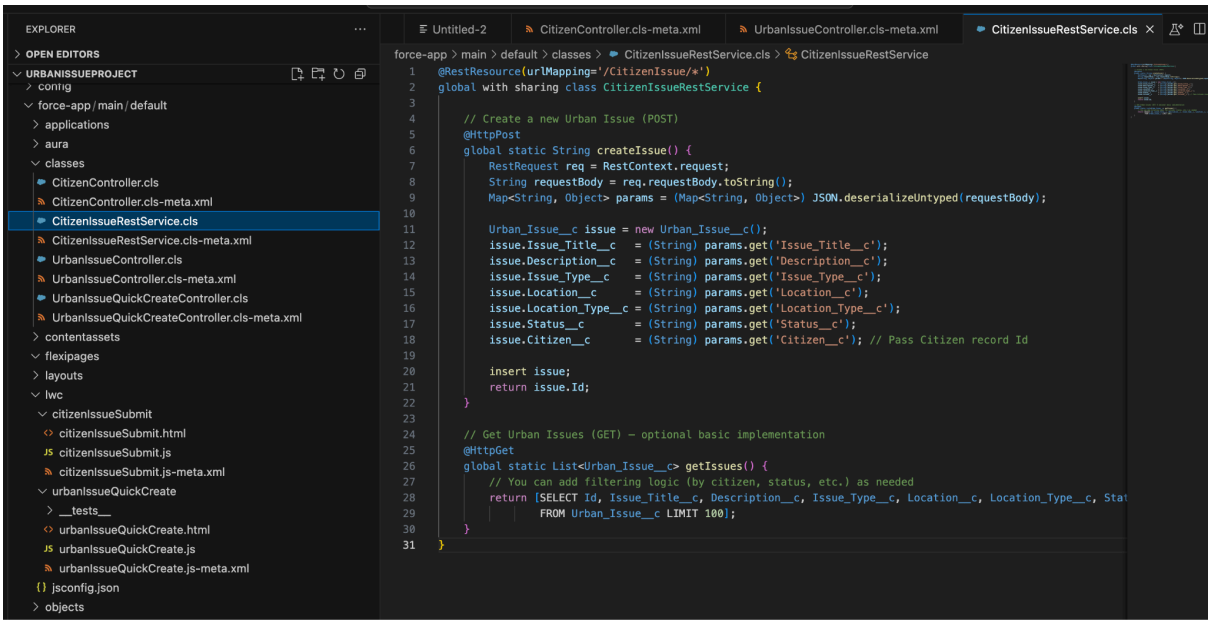
This document details each integration and external access concept implemented in Phase 7 of the Urban Issue Management Salesforce project. For each, a clear use case is provided, followed by a summary of the implementation and notes on where to add screenshots.

---

1. Apex REST Web Service

Use Case:
To allow external platforms such as a government website, mobile app, or third-party service to create or view Urban Issues directly in Salesforce, we exposed a secure REST API. This enables seamless integration with external systems, automating issue submissions and status tracking without requiring direct Salesforce access for external users.

Implementation:
We developed an Apex class (`CitizenIssueRestService`) annotated with `@RestResource`, exposing REST endpoints to create and query Urban Issues. External systems can send HTTP requests with issue data, and Salesforce processes them securely.

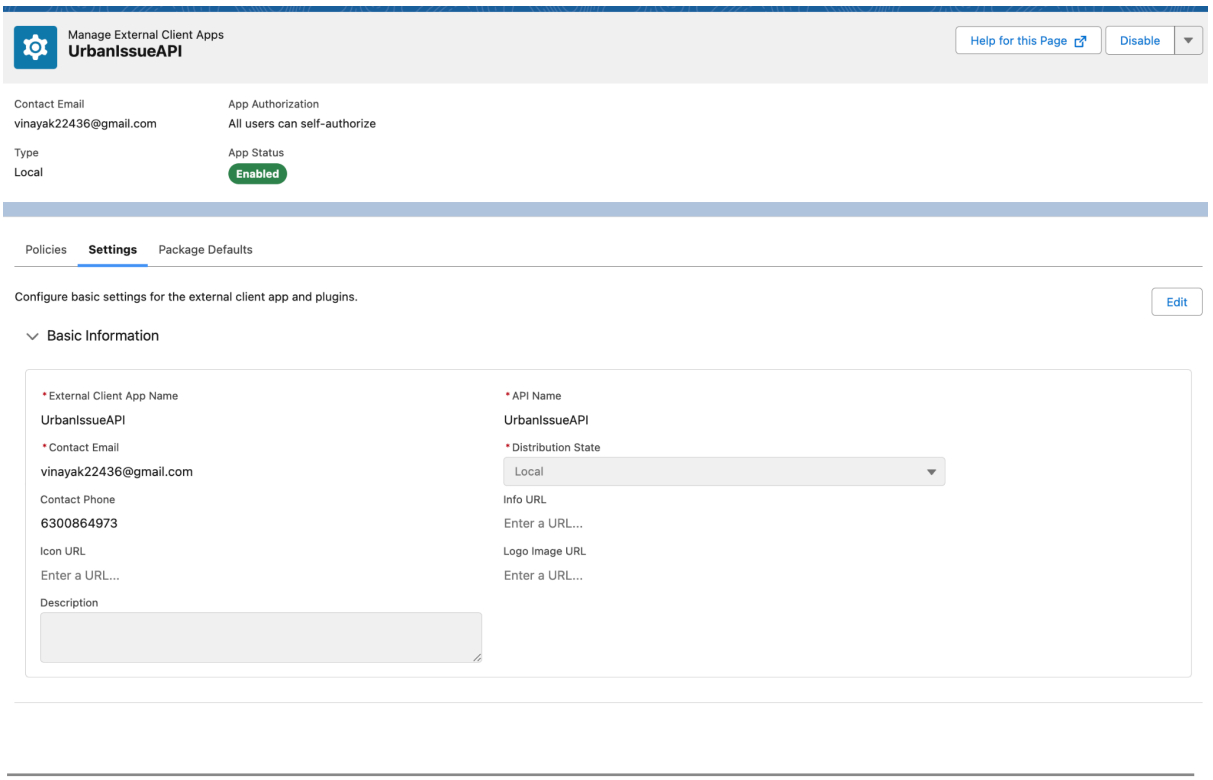


---

2. OAuth & Authentication

Use Case:
To ensure that only authorized and authenticated external applications can access our Salesforce REST API, we implemented OAuth 2.0 authentication. This protects sensitive data and restricts API access to trusted integrations.

Implementation:
A Connected App was configured in Salesforce Setup, with OAuth enabled. The Connected App provides a Client ID/Secret and defines OAuth scopes, allowing external apps to request and use access tokens for secure API communication.



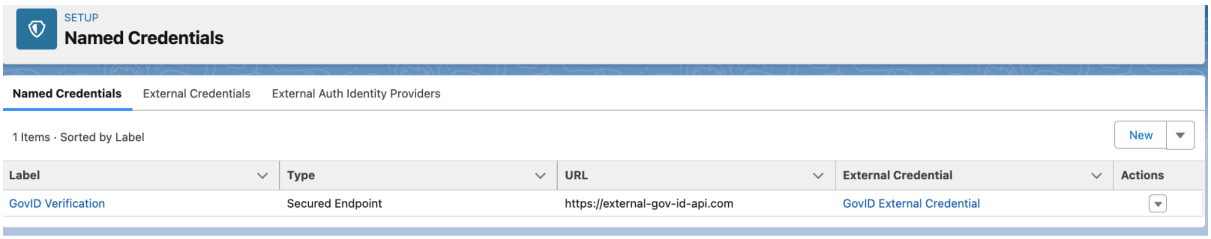
---

3. Named Credentials

Use Case:
When Salesforce needs to connect to external APIs (e.g., for government ID validation or external data checks), using Named Credentials allows us to securely store authentication details, simplify callout management, and centralize credential updates.

Implementation:
A Named Credential (`GovID_Verification`) was created, referencing an External Credential and Principal which store the authentication method and details for the

external service. This setup ensures callouts are secure, manageable, and compliant with Salesforce best practices.



Callouts

Use Case:
(Prepared, not yet implemented) To allow Salesforce to send data or validate information with external systems (such as sending government ID validation requests or notifications) via HTTP callouts. The infrastructure is in place for future expansion as business needs require.

Implementation:
Named Credentials have been configured for outbound authentication. When needed, Apex code can use these credentials to make secure callouts to external services.

Phase 8: Data Management & Deployment — Implementation Documentation

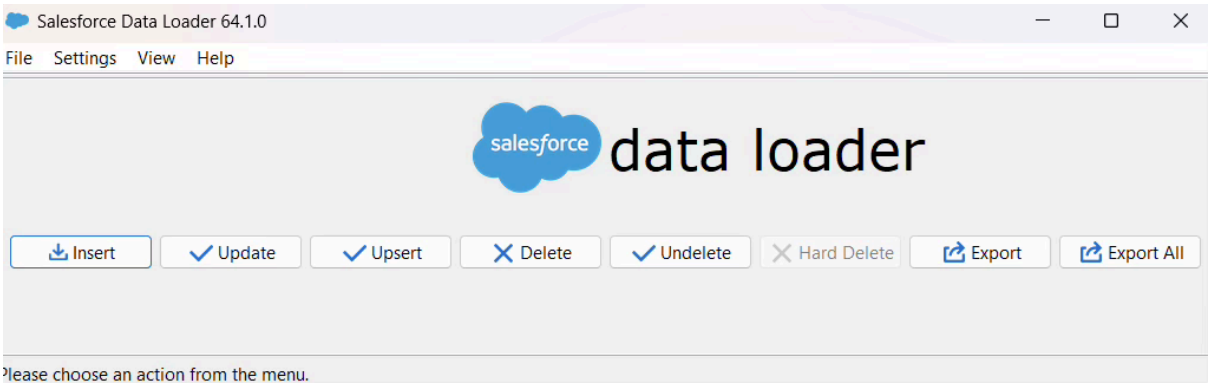
This document details each data management and deployment concept implemented in Phase 8 of the Urban Issue Management Salesforce project. For each, a clear use case is provided, followed by a summary of the implementation and notes on where to add screenshots for your records.

---

1. Data Import Wizard

Use Case:
To quickly and easily load bulk records (such as Citizens and Urban Issues) into Salesforce from CSV files, especially during initial setup or when migrating data from legacy systems. The Data Import Wizard provides a user-friendly interface for admins to map CSV columns to Salesforce fields and verify data before insertion.

Implementation:
We used the Salesforce Data Import Wizard to import new Citizen and Urban Issue records. This process involved preparing a CSV file, launching the wizard, selecting the target object, mapping fields, and confirming the import.

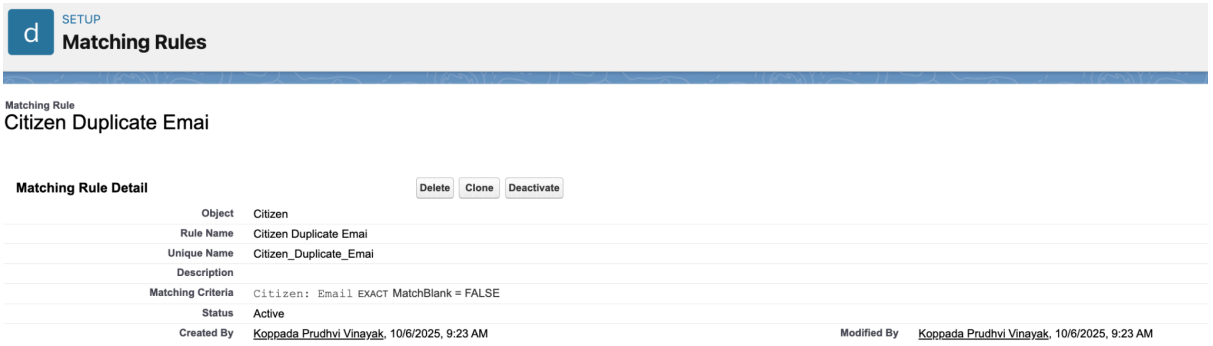


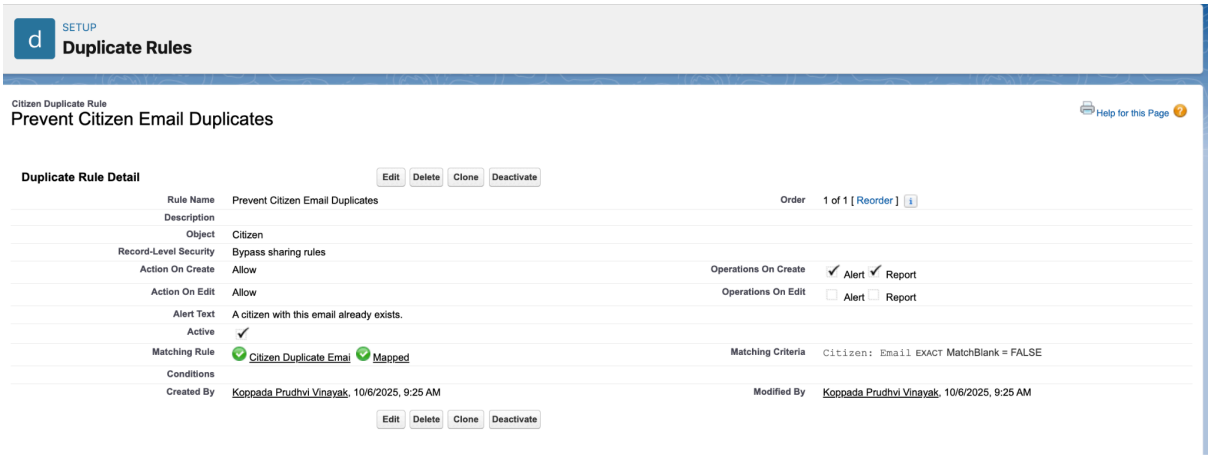
---

2. Duplicate Rules

Use Case:
To maintain high data quality by detecting and preventing duplicate records, such as Citizens with the same email or government ID. Duplicate Rules ensure that each record in the system is unique according to your chosen criteria, reducing confusion and redundant entries.

Implementation:
We created custom Matching Rules based on unique fields (like Email__c or Government_ID__c) and then set up Duplicate Rules for the Citizen object. The rules were configured to either block duplicate entries or alert users when a potential duplicate is detected during data entry or import.



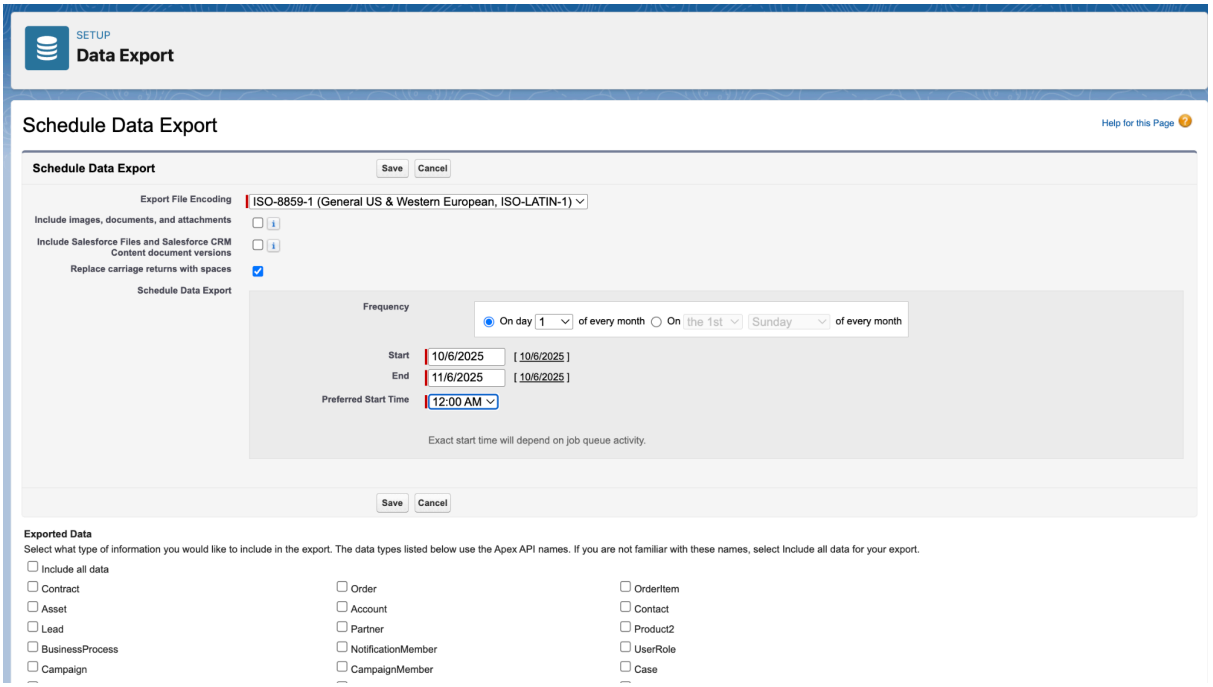


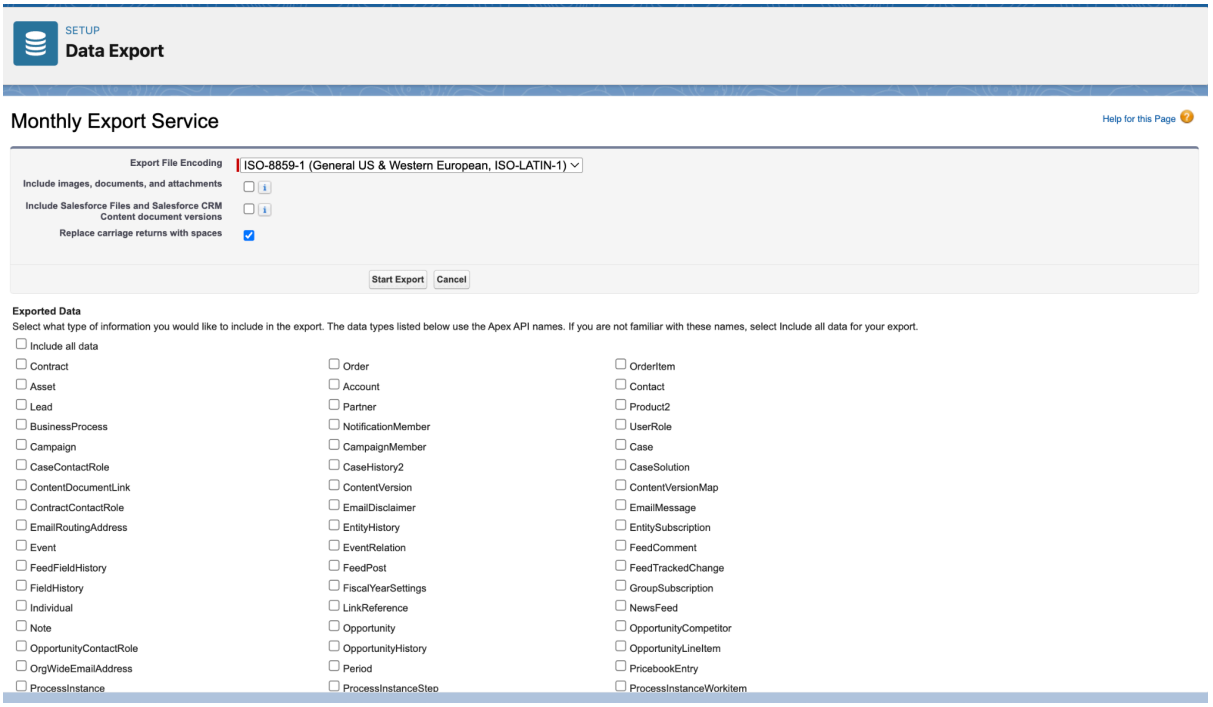
3. Data Export & Backup

Use Case:
To regularly back up Salesforce data for safety, compliance, and disaster recovery. Data Export ensures that all business-critical records can be restored or analyzed offline if needed.

Implementation:
We scheduled regular data exports using Salesforce's built-in Data Export tool. This included selecting all relevant objects (e.g., Citizen, Urban Issue) and setting a

backup schedule (weekly or monthly). We also performed on-demand exports as needed for immediate backups.





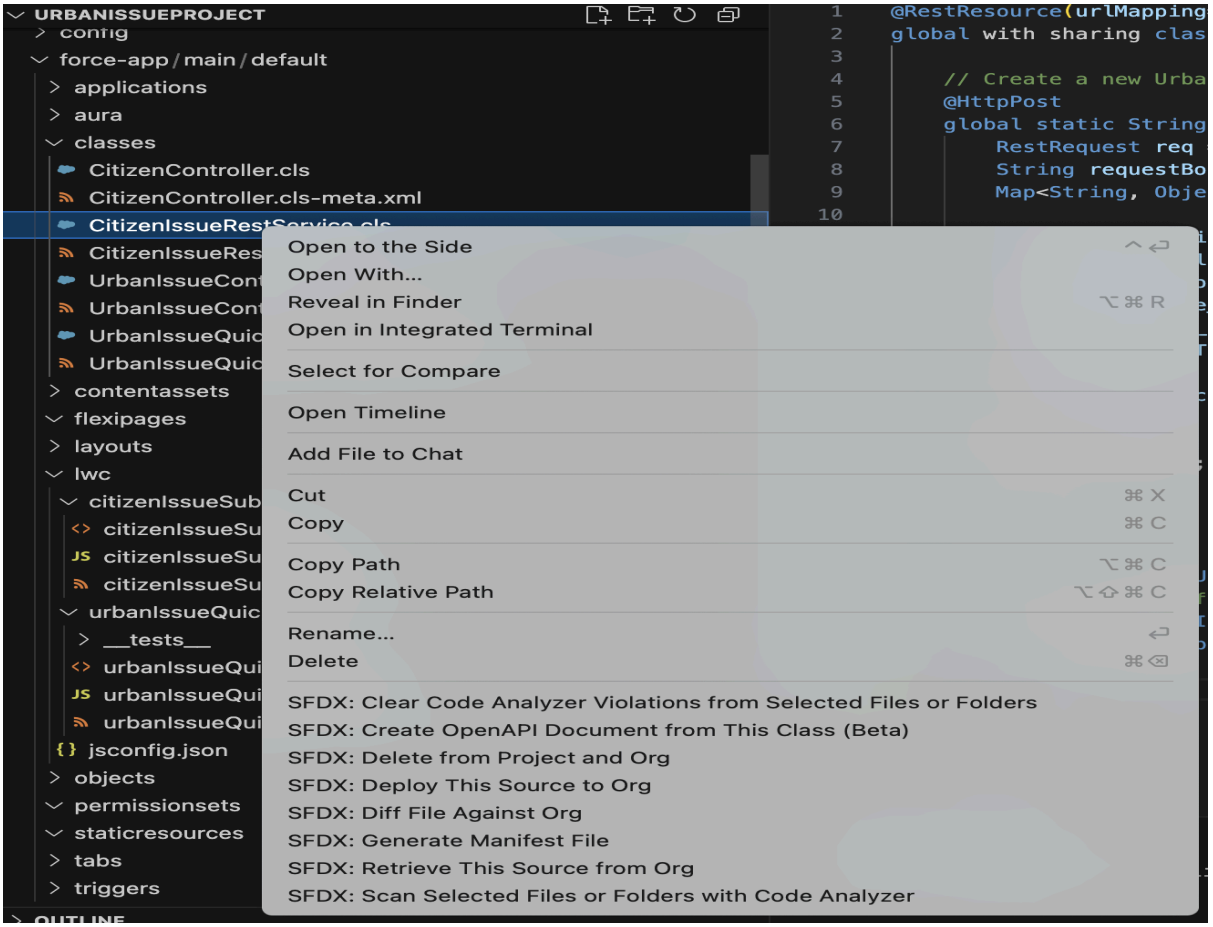
---

5. VS Code & SFDX Use Case:

For advanced/developer-driven projects, VS Code with Salesforce DX (SFDX) allows

for source-driven development, version control, and more flexible deployments. SFDX is essential for teams practicing modern DevOps or continuous integration.

Implementation:
We used VS Code with the Salesforce Extensions Pack to retrieve and deploy metadata, manage source code, and run SFDX commands for deployments between orgs.



PHASE 9 - Reporting, Dashboards & Security Review

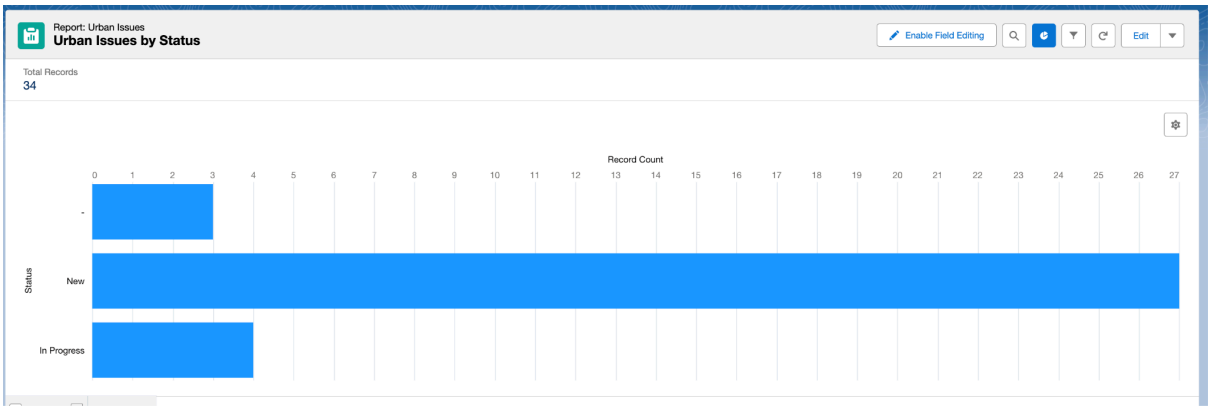
This phase focused on providing actionable insights through reporting, enabling visual representation of key metrics via dashboards, and ensuring the solution is secure and ready for deployment.

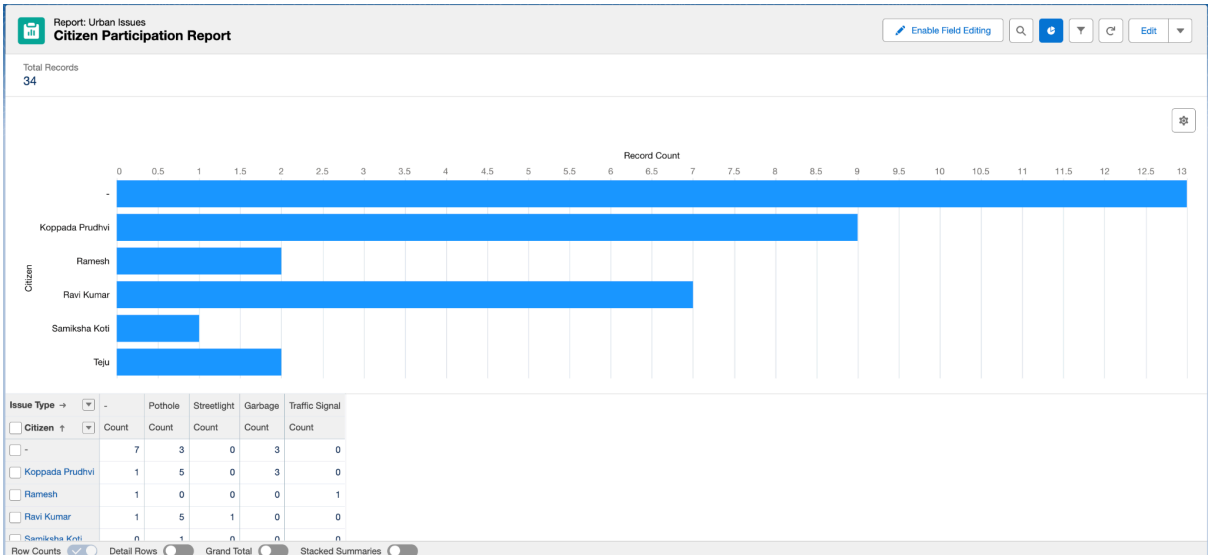
Reporting

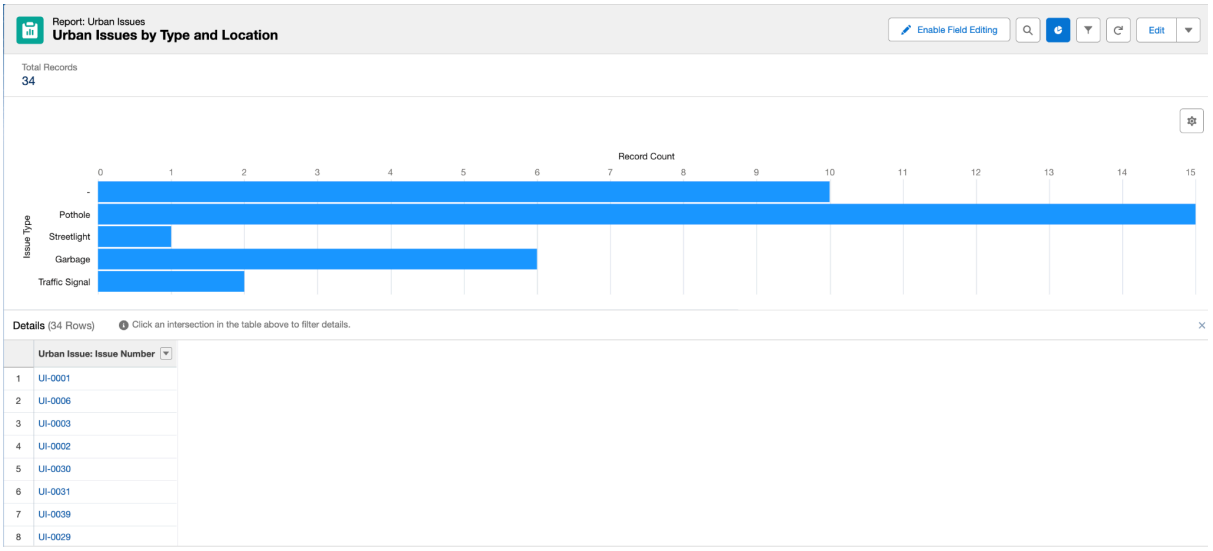
Reports were created to provide visibility into the Urban Issue Management process and citizen engagement. These reports empower teams to monitor trends, identify bottlenecks, and make data-driven decisions.

Steps Taken:

- Created a custom report type, “Urban Issues with Citizens,” to enable combined reporting across Urban Issue and Citizen records.
- Built the following reports:
    - Urban Issues by Status: Shows the count of issues in each status (Open, In Progress, Closed, etc.) to monitor workflow and backlog.
    - Urban Issues by Type and Location: Displays the distribution of reported issues by type across various locations, highlighting hotspots or recurring problems.
- Grouped and filtered data as needed for actionable views.
- Added charts to reports for quick visualization.





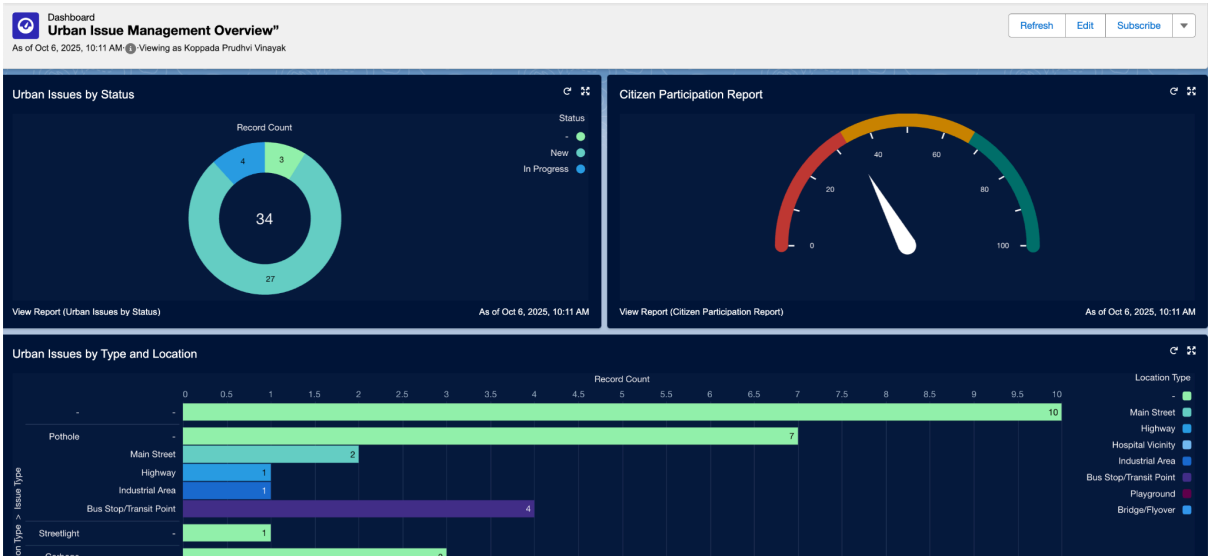


♦ Dashboards

Dashboards were implemented to provide at-a-glance insights for stakeholders, combining multiple reports and key performance indicators onto a single screen.

Steps Taken:

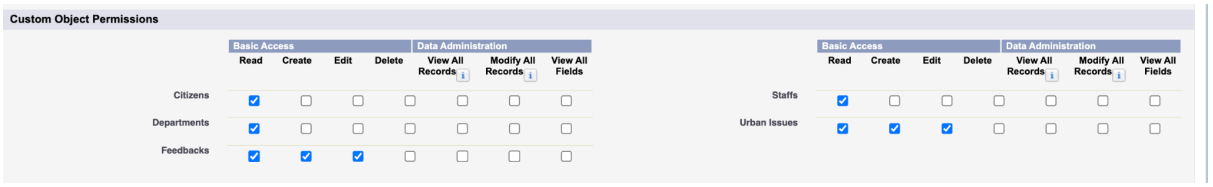
- Created a dashboard titled “Urban Issue Management Overview.”
- Added components using the previously built reports:
  - Pie chart for Urban Issues by Status.
  - Stacked bar chart for Urban Issues by Type and Location.
- Arranged and formatted dashboard components for clarity and impact.



Security Review

A final review of the security settings was conducted to ensure the principle of least privilege was enforced across all user profiles in the Urban Issue Management application.

Field Level Security:
The field-level security for the Citizen, Urban Issue, and related custom profiles was meticulously reviewed and configured. Access to sensitive fields—such as government identification numbers and contact details—was restricted so that users can only view or edit information relevant to their specific roles (e.g., city staff, department head).



Sharing Settings:
The Organization-Wide Defaults (OWD) and Sharing Rules for the Citizen and Urban Issue objects were examined to confirm that data visibility is appropriately limited. Only authorized users can access or modify records, with sharing rules established to grant broader access only when necessary and as defined in earlier phases.

Citizen	Private	Private	✓
Department	Public Read/Write	Private	✓
Feedback	Private	Private	✓
Staff	Public Read/Write	Private	✓
Urban Issue	Private	Private	✓

Phase 10: Quality Assurance Testing & Conclusion

This phase ensures that every Salesforce feature in the Urban Issue Management System functions as intended, is user-friendly, and meets business requirements. Comprehensive testing validates system reliability, data integrity, and user experience prior to go-live.

Quality Assurance Testing

Testing Approach:
For each feature—such as record creation, approval processes, automated flows, triggers, validation rules, reporting, and dashboard visuals—test cases were prepared. Each test case documents the scenario, input steps, expected outcome, actual outcome, and includes screenshots as evidence.

Sample Test Case Structure

Use Case / Scenario:
Citizen submits a new Urban Issue, triggering automatic assignment and notification.

Test Steps (with Input):

1. Navigate to the “Submit Issue” Lightning Web Component or Urban Issue record page.
2. Enter Citizen details (e.g., Name: Priya Singh, Email: priya@email.com, Gov ID: GOV999).
3. Enter Urban Issue details (Type: Pothole, Location: MG Road, Description: “Large pothole near intersection”).
4. Click “Submit”.

Expected Result:

- A new Citizen record is created (if new).
- An Urban Issue record is created with assigned Department and auto-generated Issue Number.
- Citizen receives a confirmation email with Issue reference.

- Status is set to “New”.

Actual Result:

- [Insert screenshot of input form]
- [Insert screenshot of created records]
- [Insert screenshot of confirmation email]

Use Case / Scenario:
Approval Process for Urgent Urban Issue

Test Steps (with Input):

1. Create a new Urban Issue with Priority = “Urgent”.
2. Submit for approval.

Expected Result:

- Supervisor receives approval request email.
- Approval status is “Pending”.
- Upon approval, status updates to “Approved” and assigned staff is notified.

Actual Result:

- [Insert screenshot of approval email]
- [Insert screenshot of status updates in record]

Use Case / Scenario:
Validation Rule – Government ID Required

Test Steps (with Input):

1. Attempt to create a Citizen record without a Government ID.

Expected Result:

- Error message: “Government ID is required”.

Actual Result:

- [Insert screenshot of error message]

Use Case / Scenario:
Duplicate Rule – Prevent Duplicate Citizens

Test Steps (with Input):

1. Attempt to create a Citizen with an email that already exists in the system.

Expected Result:

- System blocks creation and displays duplicate warning.

Actual Result:

- [Insert screenshot of warning]

Use Case / Scenario:
Data Export & Backup

Test Steps (with Input):

1. Schedule or run an on-demand export for the Urban Issue and Citizen objects.

Expected Result:

- Export completes and downloadable .zip with CSVs is available.

Actual Result:

- [Insert screenshot of export scheduling]
- [Insert screenshot of download link]

Use Case / Scenario:
Reporting and Dashboards

Test Steps (with Input):

1. Run the “Urban Issues by Status” report.
2. View the “Urban Issue Management Overview” dashboard.

Expected Result:

- Report displays accurate counts grouped by status.
- Dashboard components reflect latest data.

Actual Result:

- [Insert screenshot of report output]
- [Insert screenshot of dashboard]

Conclusion

The Urban Issue Management System's implementation in Salesforce successfully addresses the city's need for a transparent, efficient, and citizen-focused platform for reporting and tracking urban infrastructure issues. The project's phased approach ensured robust data modeling, automation, integration, quality assurance, and user-centric design.

Key achievements include:

- Streamlined citizen issue reporting and tracking
- Automated assignment, approval, and notifications reducing manual workload
- Data quality enforced through validation and duplicate rules
- Reliable reporting and dashboards for city management oversight
- Secure, role-based access and compliance with data privacy standards
- Scalable architecture ready for future enhancements (IoT, analytics, etc.)

Future Enhancements

Chatbot Integration:
Add an AI-powered chatbot to help citizens report issues, get instant updates, and answer FAQs, improving user engagement and support.

- AI Suggestions:
  Use AI/ML (e.g., Salesforce Einstein) to auto-suggest issue priorities, predict escalations, and recommend assignments, making resolution faster and smarter.
- Mobile App:
  Develop a mobile app with GPS, photo upload, and push notifications for easier field reporting by both citizens and staff.
- IoT & GIS Integration:
  Connect IoT sensors and mapping (GIS) to auto-detect issues and provide real-time location analytics.
- Community Portal:
  Enable forums, voting, and transparent feedback for greater citizen participation and trust.

- Advanced Analytics:
  Add more powerful dashboards, trend analysis, and export options for better city management decisions.
- Multi-language & Accessibility:
  Support additional languages and accessibility features to serve all citizens.

These enhancements will make the system smarter, more responsive, and ready for future city needs.