

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

The screenshot shows the 'Company Information' setup page for the organization 'Crowdsourced Smart City Issue Tracking System'. The page includes sections for Organization Detail, Licenses, and various configuration options like newsletter settings and API limits. Key details shown include the organization name, address, and contact information.

Organization Detail		Licenses	
Organization Name	Crowdsourced Smart City Issue Tracking System	Phone	(630) 086-4973
Primary Contact	Koppada Prudhvi Vinayak	Fax	
Division		Default Locale	English (India)
Address	Kakinada 533002 Andhra Pradesh India	Default Language	English
Fiscal Year Starts In	January	Default Time Zone	(GMT+05:30) India Standard Time (Asia/Kolkata)
Activate Multiple Currencies	<input type="checkbox"/>	Currency Locale	Telugu (India) - INR
Enable Data Translation	<input type="checkbox"/>	Used Data Space	402 KB (8%) [View]
Newsletter	<input checked="" type="checkbox"/>	Used File Space	17 KB (0%) [View]
Admin Newsletter	<input checked="" type="checkbox"/>	API Requests, Last 24 Hours	0 (15,000 max)
Hide Notices About System Maintenance	<input type="checkbox"/>	Streaming API Events, Last 24 Hours	0 (10,000 max)
Hide Notices About System Downtime	<input type="checkbox"/>	Restricted Logins, Current Month	0 (0 max)
Locale Formats	ICU	Salesforce.com Organization ID	00Df0000008q2U9
		Organization Edition	Developer Edition
		Instance	USA1044

Created By: OrgFarm_EPIC, 9/19/2025, 5:28 PM Modified By: Koppada.Prudhvi.Vinayak, 9/26/2025, 8:22 AM

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.

The screenshot shows the 'Business Hours' setup page in Salesforce. At the top, there's a header with a 'SETUP' button and a 'Business Hours' title. Below the header, a section titled 'Organization Business Hours' contains a note about selecting support team availability. A 'Help for this Page' link is also present. The main area is titled 'Business Hours Detail' and contains a table with columns for 'Business Hours Name', 'City Services Hours', 'Time Zone', and 'Default Business Hours'. The table shows a single row for 'Business Hours' with 'Sunday' through 'Saturday' all listed as '24 Hours'. The 'Time Zone' is set to '(GMT+05:30) India Standard Time (Asia/Kolkata)'. Below the table, there are fields for 'Active' status (checked), 'Created By' (Koppada Prudvi Vinayak), and 'Last Modified By' (Koppada Prudvi Vinayak). An 'Edit' button is located below these fields. At the bottom, there's a 'Holidays' section with an 'Add/Remove' button and a note stating 'No records to display'. Navigation links include '^ Back To Top' and 'Always show me ▾ more records per related list'.

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

The screenshot shows the 'Fiscal Year' setup page in Salesforce. At the top, there's a 'SETUP' button and a 'Fiscal Year' section. Below it, the title 'Organization Fiscal Year Edit: Crowdsourced Smart City Issue Tracking System' is displayed. A note says 'To specify the fiscal year type for your organization, choose one of the options below.' On the left, there are two radio button options: 'Standard Fiscal Year' (selected) and 'Custom Fiscal Year'. The main area contains a 'Fiscal Year Information' section with a note about changing the fiscal year start month. A 'Change Fiscal Year Period' dialog box is open, showing the 'Name' as 'Crowdsourced Smart City Issue Tracking System', 'Fiscal Year Start Month' set to 'January', and 'Fiscal Year is Based On' set to 'The ending month'. A warning message in a yellow box states: 'Changing the fiscal year shifts fiscal periods and impacts opportunities and forecasts across your organization. If your forecast periods are set to quarterly, adjusting the fiscal year start month will erase existing forecast adjustments and quotas. Consider exporting a data backup before implementing this change.'

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.*

<input type="checkbox"/> Edit Del ... <u>Department Head</u>	Salesforce
<input type="checkbox"/> Edit Del ... <u>Department Staff</u>	Salesforce

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.

Creating the Role Hierarchy

You can build on the existing role hierarchy shown on this page. To insert a new role, click **Add Role**.

Your Organization's Role Hierarchy

[Help for this Page](#)

[Show in tree view](#)

[Collapse All](#) [Expand All](#)

- **Crowdsourced Smart City Issue Tracking System**
 - └ Add Role
 - └ Add Role
 - └ **CEO** [Edit](#) | [Del](#) | [Assign](#)
 - └ Add Role
 - └ **CFO** [Edit](#) | [Del](#) | [Assign](#)
 - └ Add Role
 - └ **City Admin** [Edit](#) | [Del](#) | [Assign](#)
 - └ Add Role
 - └ **Department Head** [Edit](#) | [Del](#) | [Assign](#)
 - └ Add Role
 - └ **Department Staff** [Edit](#) | [Del](#) | [Assign](#)
 - └ Add Role
 - └ **COO** [Edit](#) | [Del](#) | [Assign](#)
 - └ Add Role
 - └ **SVP, Customer Service & Support** [Edit](#) | [Del](#) | [Assign](#)
 - └ Add Role
 - └ **SVP, Human Resources** [Edit](#) | [Del](#) | [Assign](#)
 - └ Add Role
 - └ **SVP, Sales & Marketing** [Edit](#) | [Del](#) | [Assign](#)
 - └ Add Role

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.

The screenshot shows the Salesforce Setup - Users page. At the top, there's a blue header bar with the text "SETUP" and "Users". Below the header, the page title is "All Users". On the right, there's a link "Help for this Page ?". Under the title, a message says "On this page you can create, view, and manage users." followed by "To get more licenses, use the Your Account app. [Let's Go](#)". Below this, there's a "View" dropdown set to "All Users" and a "Create New View" button. A navigation bar at the top of the list table includes links for "New User", "Reset Password(s)", and "Add Multiple Users". The main table has columns: Action, Full Name, Alias, Username, Role, Active, and Profile. The table lists eight users:

Action	Full Name	Alias	Username	Role	Active	Profile
<input type="checkbox"/>	Agarwal, Priya	PAgarwal	priya.agarwal@citizen.com		<input checked="" type="checkbox"/>	Citizen
<input type="checkbox"/>	Chatter Expert	Chatter	chatty.00dfj0000008q2u9eai.cfkchvzpzf7g@chatter.salesforce.com		<input checked="" type="checkbox"/>	Chatter Free User
<input type="checkbox"/>	EPIC_OrgFarm	OEPIC	epic.14028bef1cad@orgfarm.salesforce.com		<input checked="" type="checkbox"/>	System Administrator
<input type="checkbox"/>	Kumar, Ravi	rkuma	ravi.kumar@city.gov.in	Department Head	<input checked="" type="checkbox"/>	Department Head
<input type="checkbox"/>	Prudhi Vinayak_Koppada	vin	vinayak22436141@agentforce.com	City Admin	<input checked="" type="checkbox"/>	System Administrator
<input type="checkbox"/>	Sharma, Sita	sshar	sita.sharma@city.gov.in	Department Staff	<input checked="" type="checkbox"/>	Department Staff
<input type="checkbox"/>	User_Integration	integ	integration@00dfj0000008q2u9eai.com		<input checked="" type="checkbox"/>	Analytics Cloud Integration User
<input type="checkbox"/>	User_Security	sec	insightssecurity@00dfj0000008q2u9eai.com		<input checked="" type="checkbox"/>	Analytics Cloud Security User

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).

SETUP > OBJECT MANAGER
Department

Details		Edit Delete
Fields & Relationships	Description	
Page Layouts		
Lightning Record Pages	API Name Department__c	Enable Reports ✓
Buttons, Links, and Actions	Custom ✓	Track Activities ✓
Compact Layouts	Singular Label Department	Track Field History ✓
Field Sets	Plural Label Departments	Deployment Status Deployed
Object Limits		Help Settings Standard salesforce.com Help Window
Record Types		
Related Lookup Filters		
Search Layouts		
List View Button Layout		
Restriction Rules		
Scoping Rules		
Object Access		

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

SETUP > OBJECT MANAGER
Urban Issue

Details		Edit Delete
Fields & Relationships	Description	
Page Layouts		
Lightning Record Pages	API Name Urban_Issue__c	Enable Reports ✓
Buttons, Links, and Actions	Custom ✓	Track Activities ✓
Compact Layouts	Singular Label Urban Issue	Track Field History ✓
Field Sets	Plural Label Urban Issues	Deployment Status Deployed
Object Limits		Help Settings Standard salesforce.com Help Window
Record Types		
Related Lookup Filters		
Search Layouts		
List View Button Layout		
Restriction Rules		
Scoping Rules		
Object Access		

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

SETUP > OBJECT MANAGER
Citizen

Details	Details	Edit Delete
Fields & Relationships	Description	
Page Layouts	API Name Citizen__c	Enable Reports ✓
Lightning Record Pages	Custom ✓	Track Activities ✓
Buttons, Links, and Actions	Singular Label Citizen	Track Field History ✓
Compact Layouts	Plural Label Citizens	Deployment Status Deployed
Field Sets		Help Settings Standard salesforce.com Help Window
Object Limits		
Record Types		
Related Lookup Filters		
Search Layouts		
List View Button Layout		
Restriction Rules		
Scoping Rules		
Object Access		

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

SETUP > OBJECT MANAGER
Feedback

Details	Details	Edit Delete
Fields & Relationships	Description	
Page Layouts	API Name Feedback__c	Enable Reports ✓
Lightning Record Pages	Custom ✓	Track Activities ✓
Buttons, Links, and Actions	Singular Label Feedback	Track Field History ✓
Compact Layouts	Plural Label Feedbacks	Deployment Status Deployed
Field Sets		Help Settings Standard salesforce.com Help Window
Object Limits		
Record Types		
Related Lookup Filters		
Search Layouts		
List View Button Layout		
Restriction Rules		
Scoping Rules		
Object Access		

- **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)

SETUP > OBJECT MANAGER Department					
Details Fields & Relationships Page Layouts Lightning Record Pages Buttons, Links, and Actions Compact Layouts Field Sets Object Limits Record Types Related Lookup Filters Search Layouts List View Button Layout Restriction Rules Scoping Rules Object Access	Fields & Relationships				
	FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
	Created By	CreatedById	Lookup(User)		
	Department Code	Department_Code__c	Text(10)		
	Department Email	Department_Email__c	Email		
	Department Head	Department_Head__c	Lookup(User)		
	Department Name	Name	Text(80)		
	Last Modified By	LastModifiedById	Lookup(User)		
	Owner	OwnerId	Lookup(User,Group)		
	Phone Number	Phone_Number__c	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)

SETUP > OBJECT MANAGER Urban Issue					
Details Fields & Relationships Page Layouts Lightning Record Pages Buttons, Links, and Actions Compact Layouts Field Sets Object Limits Record Types Related Lookup Filters Search Layouts List View Button Layout Restriction Rules Scoping Rules Object Access Triggers Flow Triggers Validation Rules Conditional Field Formatting	Fields & Relationships				
	FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
	Assigned Staff	Assigned_Staff__c	Lookup(Staff)		
	Citizen	Citizen__c	Lookup(Citizen)		
	Created By	CreatedById	Lookup(User)		
	Department	Department__c	Lookup(Department)		
	Department Name	Department_Name__c	Text(80)		
	Description	Description__c	Long Text Area(32768)		
	Issue Number	Name	Auto Number		
	Issue Title	Issue_Title__c	Text(80)		
	Issue Type	Issue_Type__c	Picklist		
	Last Modified By	LastModifiedById	Lookup(User)		
	Location	Location__c	Text(255)		
	Location Type	Location_Type__c	Picklist		
	Owner	OwnerId	Lookup(User,Group)		
	Priority	Priority__c	Picklist		
	Staff	Staff__c	Lookup(Staff)		
	Status	Status__c	Picklist		

On Citizen:

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

Citizen							
Details		Fields & Relationships					
		FIELD LABEL		FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
Page Layouts		Address		Address__c	Text Area(255)		
Lightning Record Pages		Citizen Name		Name	Text(80)	✓	
Buttons, Links, and Actions		Created By		CreatedById	Lookup(User)		
Compact Layouts		Email		Email__c	Email		
Field Sets		Government ID		Government_ID__c	Text(20)		
Object Limits		Last Modified By		LastModifiedById	Lookup(User)		
Record Types		Owner		OwnerId	Lookup(User,Group)	✓	
Related Lookup Filters		Phone Number		Phone_Number__c	Phone		
Search Layouts							
List View Button Layout							

On Feedback:

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

Feedback							
Details		Fields & Relationships					
		FIELD LABEL		FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
Page Layouts		Citizen		Citizen__c	Lookup(Citizen)	✓	
Lightning Record Pages		Created By		CreatedById	Lookup(User)		
Buttons, Links, and Actions		Feedback Number		Name	Auto Number	✓	
Compact Layouts		Feedback Text		Feedback_Text__c	Long Text Area(32768)		
Field Sets		Last Modified By		LastModifiedById	Lookup(User)		
Object Limits		Owner		OwnerId	Lookup(User,Group)	✓	
Record Types		Rating		Rating__c	Picklist		
Related Lookup Filters		Urban Issue		Urban_Issue__c	Lookup(Urban Issue)	✓	
Search Layouts							
List View Button Layout							
Restriction Rules							

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.

SETUP > OBJECT MANAGER

Department

Details
Fields & Relationships
Page Layouts
Lightning Record Pages
Buttons, Links, and Actions
Compact Layouts
Field Sets
Object Limits
Record Types
Related Lookup Filters
Search Layouts
List View Button Layout
Restriction Rules
Scoping Rules
Object Access
Triggers

Save ▾ Quick Save Preview As... Cancel Undo Redo Layout Properties

Quick Find Field Name *
Fields Buttons Quick Actions Mobile & Lightning Actions Expanded Lookups Related Lists Report Charts

Salesforce Mobile and Lightning Experience Actions
Actions in this section are predefined by Salesforce. You can override the predefined actions to set a customized list of actions on Lightning Experience and mobile app pages that use this layout. If you customize the actions in the Quick Actions in the Salesforce Classic Publisher section, and have saved the layout, then this section inherits that set of actions by default when you click to override.

Department Detail
Standard Buttons Edit Delete Clone Change Owner Change Record Type Printable View Sharing Sharing Hierarchy Edit Labels Custom Buttons

Information (Header visible on edit only)
* Department Name Sample Text
Department Code Sample Text
Department Email sarah.sample@company.com
Phone Number 1-415-555-1212
Department Head Sample Text

Owner Sample Text

System Information (Header visible on edit only)
Created By Sample Text Last Modified By Sample Text

Custom Links (Header visible on edit only)

SETUP > OBJECT MANAGER

Urban Issue

Details
Fields & Relationships
Page Layouts
Lightning Record Pages
Buttons, Links, and Actions
Compact Layouts
Field Sets
Object Limits
Record Types
Related Lookup Filters
Search Layouts
List View Button Layout
Restriction Rules
Scoping Rules
Object Access
Triggers
Flow Triggers
Validation Rules

Save ▾ Quick Save Preview As... Cancel Undo Redo Layout Properties

Quick Find Field Name *
Fields Buttons Quick Actions Mobile & Lightning Actions Expanded Lookups Related Lists Report Charts

Salesforce Mobile and Lightning Experience Actions
Actions in this section are predefined by Salesforce. You can override the predefined actions to set a customized list of actions on Lightning Experience and mobile app pages that use this layout. If you customize the actions in the Quick Actions in the Salesforce Classic Publisher section, and have saved the layout, then this section inherits that set of actions by default when you click to override.

Urban Issue Detail
Standard Buttons Edit Delete Clone Change Owner Change Record Type Printable View Sharing Sharing Hierarchy Submit for Approval Edit Labels Custom Buttons

Information (Header visible on edit only)
Issue Number GEN-2004-001234
Issue Title Sample Text
Description Sample Text
Location Sample Text
Status Sample Text
Department Sample Text
Department Name Sample Text
Issue Type Sample Text
Citizen Sample Text
Priority Sample Text
Location Type Sample Text
Staff Sample Text
Assigned Staff Sample Text

Owner Sample Text

System Information (Header visible on edit only)
Created By Sample Text Last Modified By Sample Text

SETUP > OBJECT MANAGER

Citizen

Details
Fields & Relationships
Page Layouts
Lightning Record Pages
Buttons, Links, and Actions
Compact Layouts
Field Sets
Object Limits
Record Types
Related Lookup Filters
Search Layouts
List View Button Layout
Restriction Rules
Scoping Rules
Object Access
Triggers
Flow Triggers

Save ▾ Quick Save Preview As... Cancel Undo Redo Layout Properties

Quick Find Field Name *
Fields Buttons Quick Actions Mobile & Lightning Actions Expanded Lookups Related Lists Report Charts

Salesforce Mobile and Lightning Experience Actions
Actions in this section are predefined by Salesforce. You can override the predefined actions to set a customized list of actions on Lightning Experience and mobile app pages that use this layout. If you customize the actions in the Quick Actions in the Salesforce Classic Publisher section, and have saved the layout, then this section inherits that set of actions by default when you click to override.

Citizen Detail
Standard Buttons Edit Delete Clone Change Owner Change Record Type Printable View Sharing Sharing Hierarchy Edit Labels Custom Buttons

Information (Header visible on edit only)
* Citizen Name Sample Text
Email sarah.sample@company.com
Phone Number 1-415-555-1212
Address Sample Text
Government ID Sample Text

Owner Sample Text

System Information (Header visible on edit only)
Created By Sample Text Last Modified By Sample Text

Custom Links (Header visible on edit only)

Mobile Cards (Salesforce mobile only)

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.

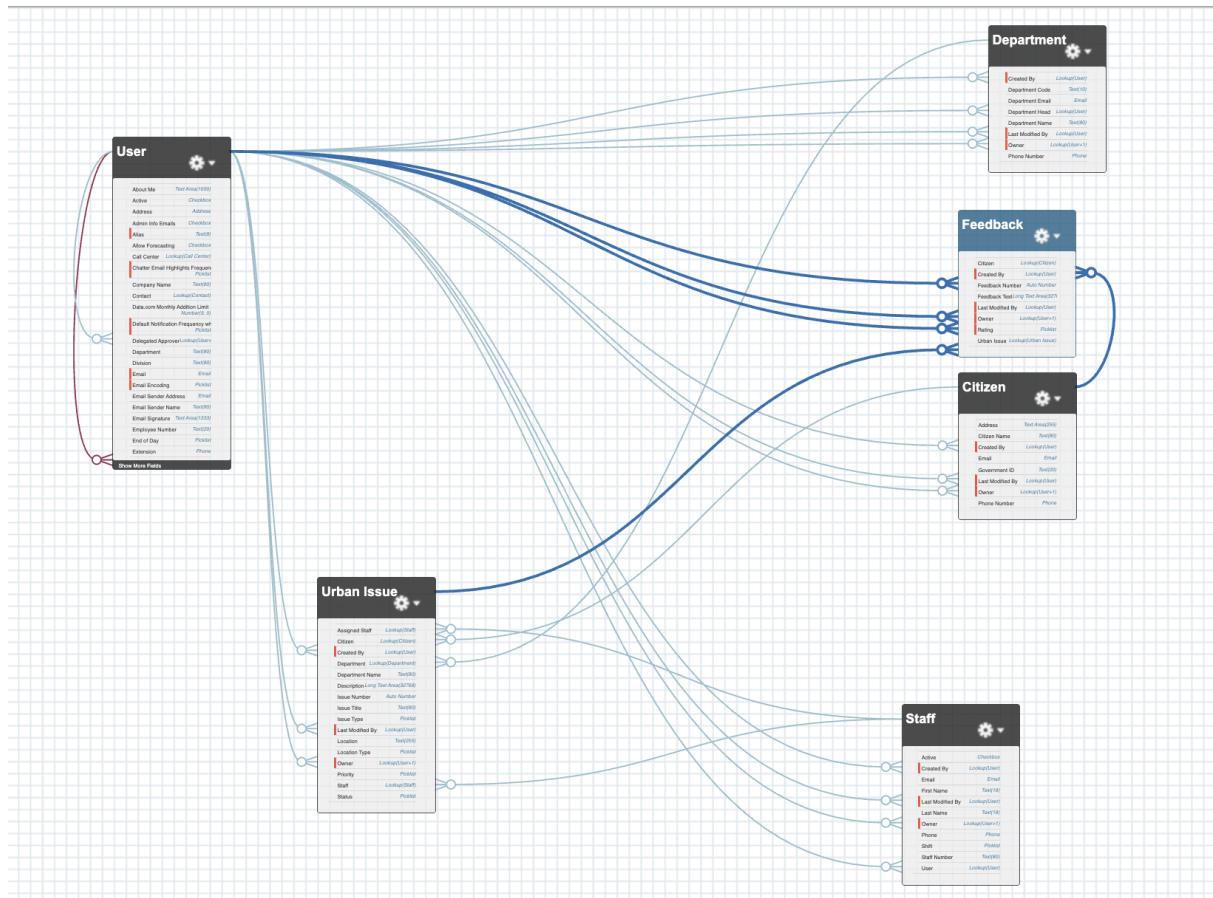
(Future enhancement: Add record types to Urban_Issue for different issue categories or reporting sources.)

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:

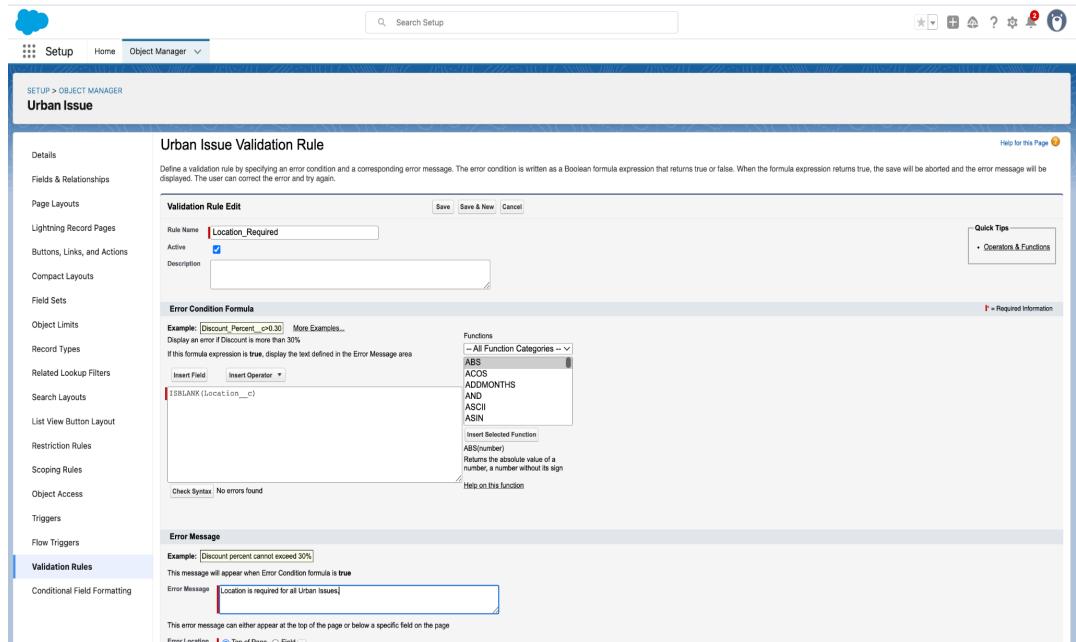
Ensure that Urban Issues have correct and complete data before saving.

Object: `Urban_Issue__c`

Implemented Examples:

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

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



* = Required Information

Information

Issue Number

Issue Title

Garbage issue

Description

(Large text area for description)

Location

(Large text area for location)

Location Type

Bus Stop/Transit Point

Issue Type

Garbage

Ø We hit a snag.

Review the errors on this page.

- Location is required for all Urban Issues.

Owner

Koppada Prudhvi Vinayak

Department

Search Departments...

Status

--None--

Priority

Urgent

Citizen

Search Citizens...

Staff

Search Staffs...

Assigned Staff

Search Staffs...



Cancel

Save & New

Save

Object: citizen__c

Implemented Examples:

- **Require_Government_ID**
 - **Formula:** ISBLANK(Government_ID_c)
 - **Error Message:** "Government ID is required".

Citizen Validation Rule

Define a validation rule by specifying an error condition and a corresponding error message. The error condition is written as a Boolean formula expression that returns true or false. When the formula expression returns true, the save displayed. The user can correct the error and try again.

Validation Rule Edit

Rule Name: **Require_Government_ID** Save Save & New Cancel

Active:

Description:

Error Condition Formula

Example: Discount_Percent_c>0.30 More Examples...

If this formula expression is true, display the text defined in the Error Message area

Insert Field Insert Operator **ISBLANK (Government_ID_c)**

Functions

-- All Function Categories --

ABS
ACOS
ADDMONTHS
AND
ASCII
ASIN

Insert Selected Function
ABS(number)
Returns the absolute value of a number, a number without its sign

Help on this function

Check Syntax

Error Message

Example: Discount percent cannot exceed 30%
This message will appear when Error Condition formula is true

Error Message: **Government ID is required.**

This error message can either appear at the top of the page or below a specific field on the page

Error Location: Top of Page Field **Government ID** i

New Citizen

* = Required Information

Information

* Citizen Name: shiva

Owner

 Koppada Prudhvi Vinayak

Email: shiva@gmail.com

Phone Number:

Address:

Government ID:

We hit a snag.

Review the following fields

• Government ID



Cancel

Save & New

Save

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"

The screenshot shows the 'Workflow Rule Detail' page for a rule named 'Urgent_Issue_Auto_In_Progress'. The rule is active and set to evaluate when a record is created or edited. It has one immediate workflow action: a 'Field Update' that sets the 'Status' field to 'In Progress' for records where the 'Priority' is 'Urgent'. A note at the bottom indicates that time triggers cannot be added to this active rule.

3. Flow Builder (Record-Triggered Flow)

Tool: Flow Builder (Record-Triggered Flow)

Objective:

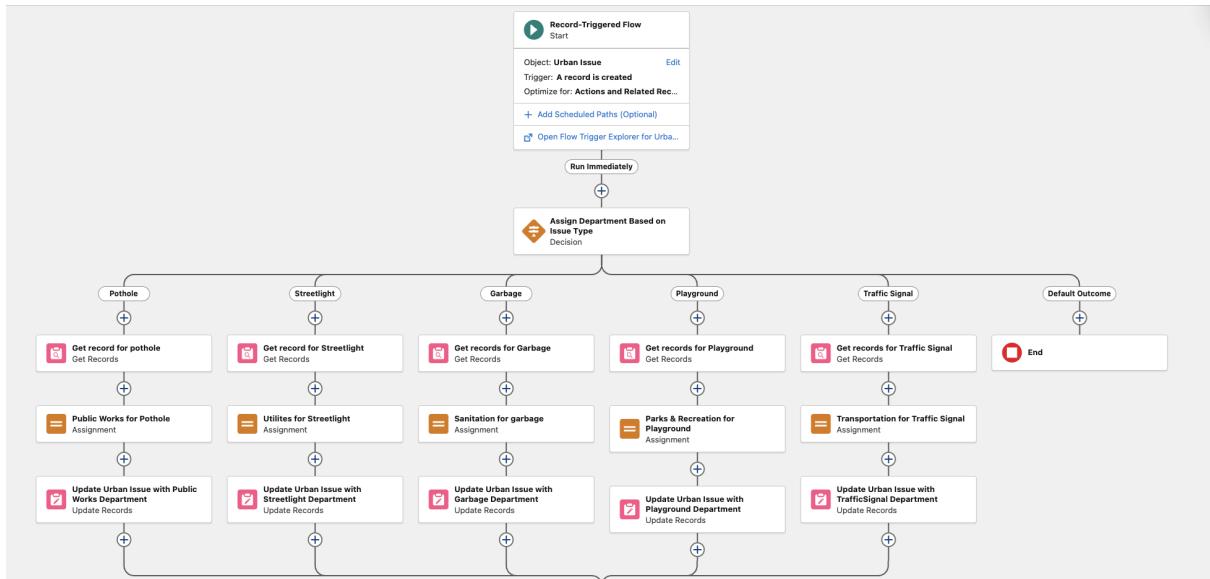
Automate core business logic for Urban Issues.

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.

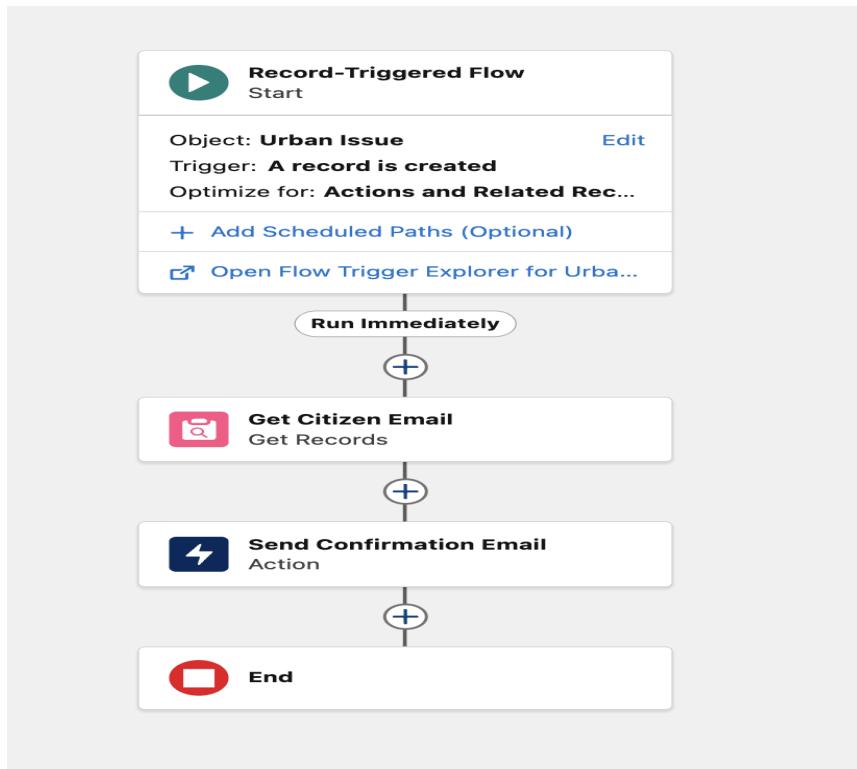
- Assignment sets Department lookup and Department Name on Urban Issue.
- Update Records element saves changes.



B. Email Notification to Citizen

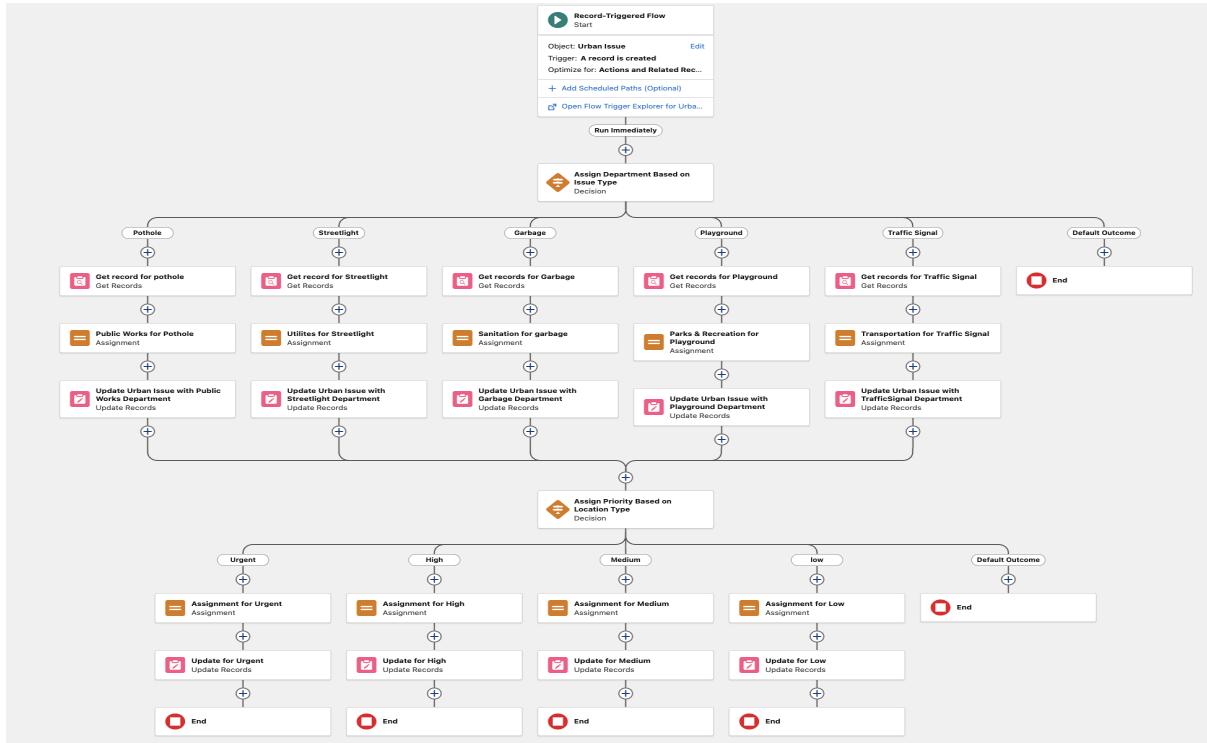
- Trigger: When Urban Issue record is created.
- Logic:

- 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".

The screenshot shows the 'Approval Processes' section of a software application. The main title is 'Urban Issue: Urgent Issue Supervisor Approval'. Below it, there's a link to 'Back to Approval Process List'. The top navigation bar includes 'Edit', 'Clone', and 'Deactivate' buttons, and a status indicator 'Active' with a checkmark.

Process Definition Detail

Process Name	Urgent Issue Supervisor Approval	Active	<input checked="" type="checkbox"/>
Unique Name	Urgent_Issue_Supervisor_Approval	Next Automated Approver Determined By	
Description			
Entry Criteria	Urban Issue: Priority EQUALS Urgent		
Record Editability	Administrator ONLY	Allow Submitters to Recall Approval Requests	<input type="checkbox"/>
Approval Assignment Email Template			
Initial Submitters	Urban Issue Owner		
Created By	Koppada Prudhvi Vinayak, 10/2/2025, 10:59 PM	Modified By	Koppada Prudhvi Vinayak, 10/2/2025, 11:31 PM

Initial Submission Actions

Action	Type	Description
	Record Lock	Lock the record from being edited
Edit	Remove	Email Alert

Approval Steps

Action	Step Number	Name	Description	Criteria	Assigned Approver	Reject Behavior
Show Actions	1	Step 1			User:Ravi Kumar	Final Rejection

Final Approval Actions

Action	Type	Description
Edit	Record Lock	Lock the record from being edited

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

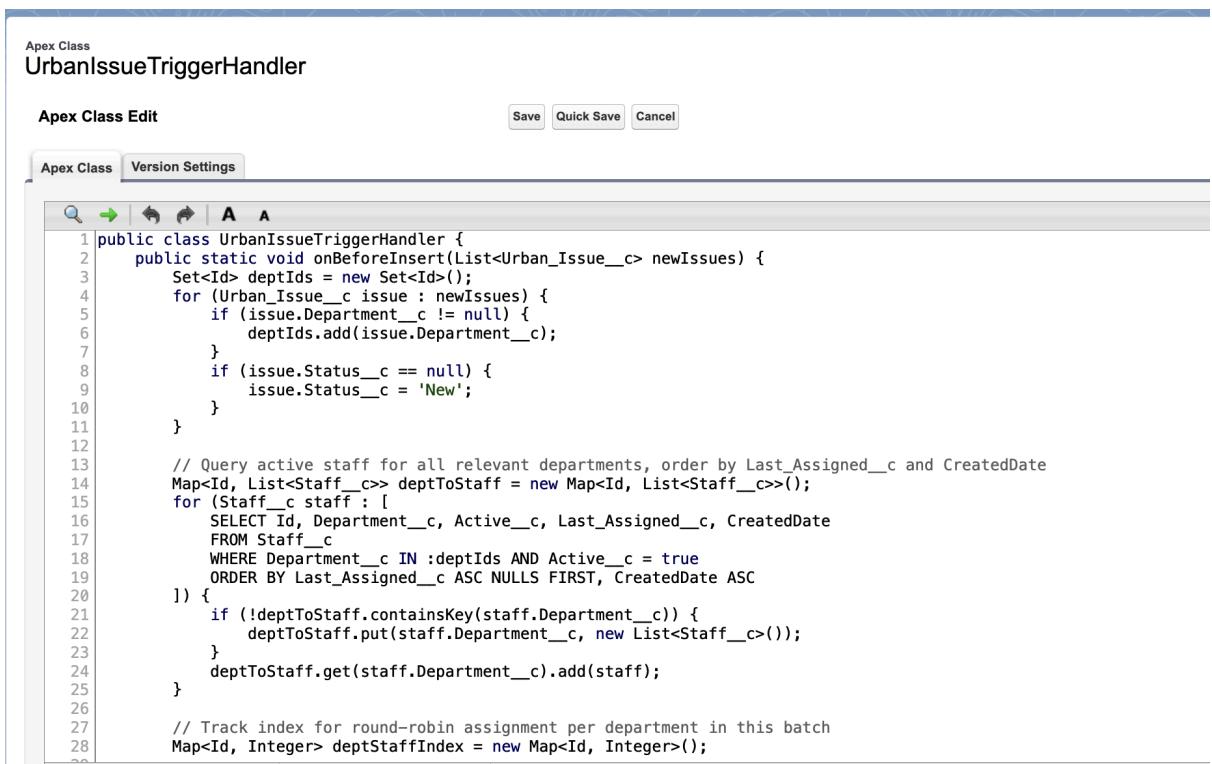
Overview

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.



The screenshot shows the Salesforce Apex Class Editor interface. The title bar says "Apex Class UrbanIssueTriggerHandler". Below it is a toolbar with "Save", "Quick Save", and "Cancel" buttons. A tab bar has "Apex Class" selected. The main area contains the Apex code for the `UrbanIssueTriggerHandler` class.

```
1 public class UrbanIssueTriggerHandler {
2     public static void onBeforeInsert(List<Urban_Issue__c> newIssues) {
3         Set<Id> deptIds = new Set<Id>();
4         for (Urban_Issue__c issue : newIssues) {
5             if (issue.Department__c != null) {
6                 deptIds.add(issue.Department__c);
7             }
8             if (issue.Status__c == null) {
9                 issue.Status__c = 'New';
10            }
11        }
12
13        // Query active staff for all relevant departments, order by Last_Assigned__c and CreatedDate
14        Map<Id, List<Staff__c>> deptToStaff = new Map<Id, List<Staff__c>>();
15        for (Staff__c staff : [
16            SELECT Id, Department__c, Active__c, Last_Assigned__c, CreatedDate
17            FROM Staff__c
18            WHERE Department__c IN :deptIds AND Active__c = true
19            ORDER BY Last_Assigned__c ASC NULLS FIRST, CreatedDate ASC
20        ]) {
21            if (!deptToStaff.containsKey(staff.Department__c)) {
22                deptToStaff.put(staff.Department__c, new List<Staff__c>());
23            }
24            deptToStaff.get(staff.Department__c).add(staff);
25        }
26
27        // Track index for round-robin assignment per department in this batch
28        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.

The screenshot shows the Apex Trigger Detail page for `UrbanIssueTrigger`. At the top, there are buttons for `Edit`, `Delete`, `Download`, and `Show Dependencies`. Below this, the trigger's details are listed:

- Name:** `UrbanIssueTrigger`
- sObject Type:** `Urban Issue`
- Status:** `Active`
- Created By:** `Koppada Prudhvi Vinayak`, `10/4/2025, 6:08 AM`
- Last Modified By:** `Koppada Prudhvi Vinayak`, `10/4/2025, 6:08 AM`
- Namespace Prefix:** None

Under the `Code Coverage` section, it shows `0% (0/4)`. The trigger's code is displayed in a code editor:

```
1trigger UrbanIssueTrigger on Urban_Issue__c (before insert, after insert) {
2    if (Trigger.isBefore && Trigger.isInsert) {
3        UrbanIssueTriggerHandler.onBeforeInsert(Trigger.new);
4    }
5    if (Trigger.isAfter && Trigger.isInsert) {
6        UrbanIssueTriggerHandler.onAfterInsert(Trigger.new);
7    }
8}
```

At the bottom of the page, there are buttons for `Edit`, `Delete`, `Download`, and `Show Dependencies`.

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.

Apex Classes

```
'8      if (issue.Status__c == null) {
9          issue.Status__c = 'New';
10     }
11 }
12 // Query active staff for all relevant departments, order by Last_Assigned__c and CreatedDate
13 Map<Id, List<Staff__c>> deptToStaff = new Map<Id, List<Staff__c>>();
14 for (Staff__c staff : [
15     SELECT Department__c, Active__c, Last_Assigned__c, CreatedDate
16     FROM Staff__c
17     WHERE Department__c IN :deptIds AND Active__c = true
18     ORDER BY Last_Assigned__c ASC NULLS FIRST, CreatedDate ASC
19 ]) {
20     if (!deptToStaff.containsKey(staff.Department__c)) {
21         deptToStaff.put(staff.Department__c, new List<Staff__c>());
22     }
23     deptToStaff.get(staff.Department__c).add(staff);
24 }
25
26 // Track index for round-robin assignment per department in this batch
27 Map<Id, Integer> deptStaffIndex = new Map<Id, Integer>();
28
29 for (Urban_Issue__c issue : newIssues) {
30     if (issue.Department__c != null && !deptToStaff.containsKey(issue.Department__c)) {
31         List<Staff__c> staffList = deptToStaff.get(issue.Department__c);
32         if (!staffList.isEmpty()) {
33             Integer idx = deptStaffIndex.containsKey(issue.Department__c) ? deptStaffIndex.get(issue.Department__c) : 0;
34             if (idx >= staffList.size()) {
35                 idx = 0; // wrap around if batch > staff count
36             }
37             Staff__c assignedStaff = staffList[idx];
38             issue.Assigned_Staff__c = assignedStaff.Id;
39             deptStaffIndex.put(issue.Department__c, idx + 1);
40         }
41     }
42 }
43 }
44
45 public static void onAfterInsert(List<Urban_Issue__c> newIssues) {
46     // Update Last_Assigned__c for each assigned staff
47     Map<Id, Staff__c> staffToUpdate = new Map<Id, Staff__c>();
48     for (Urban_Issue__c issue : newIssues) {
49         if (issue.Assigned_Staff__c != null) {
50             staffToUpdate.put(issue.Assigned_Staff__c, new Staff__c(
51                 Id: issue.Assigned_Staff__c,
52                 Last_Assigned__c = System.now()
53             ));
54         }
55     }
56     if (!staffToUpdate.isEmpty()) {
57 }
```

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).

Apex Class
UrbanIssueBatchClose

Help for this P...

Apex Class Detail

Name	UrbanIssueBatchClose	Status	Active
Namespace Prefix		Code Coverage	0% (0/8)
Created By	Koppada Prudhi Vinayak , 10/4/2025, 11:33 AM	Last Modified By	Koppada Prudhi Vinayak , 10/4/2025, 11:33 AM
Class Body Class Summary Version Settings Trace Flags			
<pre>1 global class UrbanIssueBatchClose implements Database.Batchable<SObject> { 2 global Database.QueryLocator start(Database.BatchableContext bc) { 3 // Query Urban Issues older than 30 days and not already closed via batch 4 String query = 'SELECT Id, Status__c, CreatedDate FROM Urban_Issue__c WHERE Status__c != \'Closed - Batch\' AND CreatedDate < :System.now().addDays(-30)'; 5 return Database.getQueryLocator(query); 6 } 7 8 global void execute(Database.BatchableContext bc, List<Urban_Issue__c> scope) { 9 for (Urban_Issue__c issue : scope) { 10 issue.Status__c = 'Closed - Batch'; 11 } 12 update scope; 13 } 14 15 global void finish(Database.BatchableContext bc) { 16 // Optional: Add logging or email notification if required 17 } 18 }</pre>			

Scheduled Apex

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

Apex Class
UrbanIssueBatchScheduler

Help for tl...

Apex Class Detail

Name	UrbanissueBatchScheduler	Status	Active
Namespace Prefix		Code Coverage	0% (0/3)
Created By	Koppada Prudhi Vinayak , 10/4/2025, 11:39 AM	Last Modified By	Koppada Prudhi Vinayak , 10/4/2025, 11:39 AM
Salesforce - Developer Edition Class Body Class Summary Version Settings Trace Flags			
<pre>1 global class UrbanIssueBatchScheduler implements Schedulable { 2 global void execute(SchedulableContext sc) { 3 UrbanIssueBatchClose batch = new UrbanIssueBatchClose(); 4 Database.executeBatch(batch, 200); 5 } 6 }</pre>			

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.

Apex Class
UrbanIssueTriggerHandlerTest

Help for this Page

Apex Class Detail

Name	UrbanIssueTriggerHandlerTest	Status	Active
Namespace Prefix		Created By	Koppada Prudhvi Vinayak , 10/4/2025, 6:11 AM
Last Modified By	Koppada Prudhvi Vinayak , 10/4/2025, 6:11 AM		

Class Body | Class Summary | Version Settings | Trace Flags

```

1  @isTest
2  public class UrbanIssueTriggerHandlerTest {
3      @isTest
4      static void testOnBeforeInsert() {
5          // Create a Department record that matches an Issue Type
6          Department__c dept = new Department__c(Name = 'Pothole');
7          insert dept;
8
9          // Create an Urban Issue record with Issue_Type__c matching the Department
10         Urban_Issue__c issue = new Urban_Issue__c(Issue_Type__c = 'Pothole');
11         insert issue;
12
13         // Reload issue to check Department assignment and Status
14         Urban_Issue__c insertedIssue = [
15             SELECT Id, Status__c, Department__c
16             FROM Urban_Issue__c
17             WHERE Id = :issue.Id
18         ];
19         System.assertEquals('New', insertedIssue.Status__c, 'Status should be New');
20         System.assertEquals(dept.Id, insertedIssue.Department__c, 'Department should be assigned based on Issue Type');
21     }
22 }
```

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 Name: Urban Issue Management
- * Developer Name: Urban_Issue_Management
- Description: Enter a description...

App Branding

Image: 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

Lightning App Builder | **App Settings** | **Pages** | **Urban Issue Management - Urban Issue Home Page** | ? Help

Activation... Save

Components

C. 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

Standard.RecentsItems (0)

All

Quick Create Urban Issue

Issue Type: Select an Option
Priority: Select an Option
Citizen: Search Citizens...
Create Issue

Submit a Citizen Issue

Are you a new citizen or existing citizen?
New Citizen Existing Citizen

This page has been activated for your org.

Sample Flow Report: Screen Flows

We can't draw this chart because there is no data.

View Report As of Today at 11:44 AM

> Report Chart

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

Add Filter

Lightning App Builder | **App Settings** | **Pages** | **Urban Issue Management - Urban Issue Record Page** | ? Help

Analyze Activation... Save

Components Fields

C. 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: Select an Option
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 not working
Description:
Location: Hbbbh
Location Type: Hospital Vicinity
Issue Type: Traffic Signal

Owner: Koppada Prudhvi Vinayak
Department: Transportation
Status: New
Priority: Urgent
Citizen: Teja
Staff:
Assigned Staff:
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

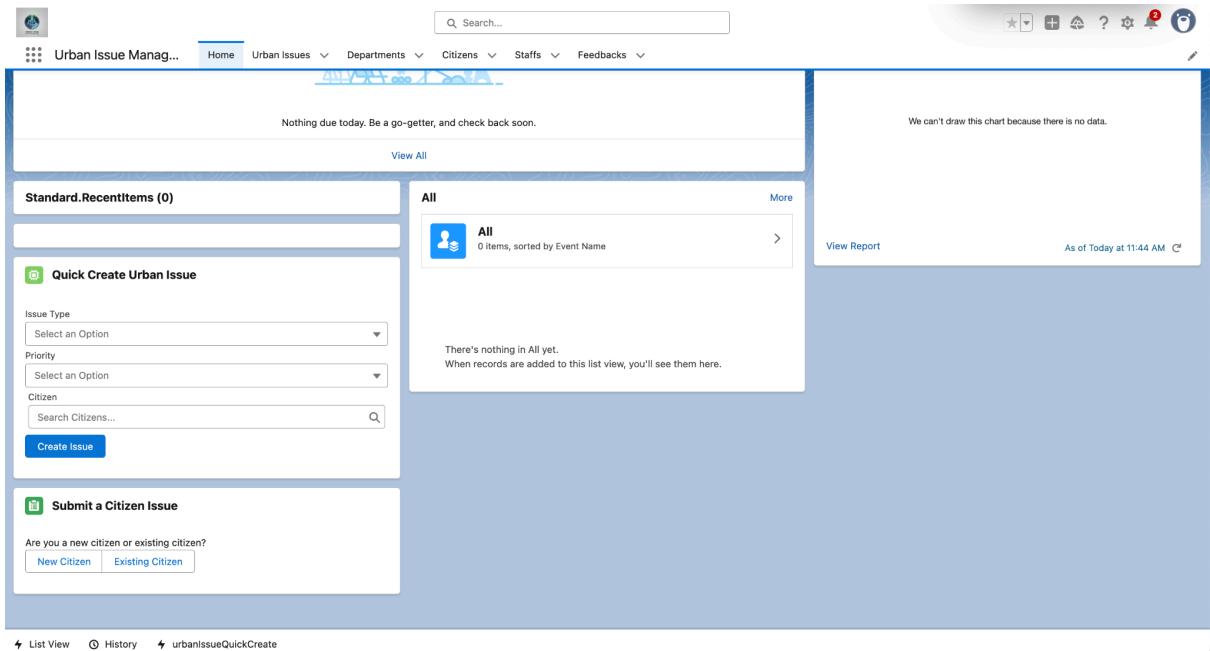
Template: Header and Left Sidebar Change

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.

```

JS urbanissueQuickCreate.js
1 import { LightningElement, track } from 'lwc';
2 import createUrbanIssue from '@salesforce/apex/UrbanIssueQuickCreateController.createUrbanIssue';
3
4 export default class UrbanIssueQuickCreate extends LightningElement {
5   @track issueType = '';
6   @track priority = '';
7   @track citizenId = '';
8   @track message = '';
9
10  issueTypeOptions = [
11    { label: 'Pothole', value: 'Pothole' },
12    { label: 'Streetlight', value: 'Streetlight' },
13    { label: 'Garbage', value: 'Garbage' },
14    { label: 'Playground', value: 'Playground' },
15    { label: 'Traffic Signal', value: 'Traffic Signal' },
16    { label: 'Other', value: 'Other' }
17  ];
18
19  priorityOptions = [
20    { label: 'Urgent', value: 'Urgent' },
21    { label: 'High', value: 'High' },
22    { label: 'Medium', value: 'Medium' },
23    { label: 'Low', value: 'Low' }
24  ];
25
26  handleIssueTypeChange(event) {
27    this.issueType = event.detail.value;
28  }
29}

PROBLEMS OUTPUT ... Filter
Salesforce metadata XML Intellisense is now available.

CitizenController.cls
1 import { LightningElement, track } from 'lwc';
2 import CitizenController from './CitizenController';
3
4 export default class CitizenController extends LightningElement {
5   @track citizens = [];
6
7   @track citizen = {
8     id: '',
9     name: '',
10    address: '',
11    email: '',
12    phone: '',
13    type: ''
14  };
15
16   ...
17 }

PROBLEMS OUTPUT ... Filter
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 PROBLEMS OUTPUT ... Filter
Salesforce CLI

citizenIssueSubmit.html
1 <template>
2   <lightning-card title="Submit a Citizen Issue" icon-name="standard:record">
3     <div>
4       <template iftrue={step1IsChoice}>
5         <div>Are you a new citizen or existing citizen?</div>
6         <lightning-button-group>
7           <lightning-button label="New Citizen" value="new" onclick={handleChoice}></lightning-button>
8           <lightning-button label="Existing Citizen" value="existing" onclick={handleChoice}></lightning-button>
9         </lightning-button-group>
10      </template>
11      <!-- Step 2: New Citizen -->
12      <template iftrue={step1IsNewCitizen}>
13        <div>Create New Citizen</div>
14        <lightning-input label="Name" value={citizenName} onchange={handleCitizenNameChange}></lightning-input>
15        <lightning-input label="Email" type="email" value={citizenEmail} onchange={handleCitizenEmail}></lightning-input>
16        <lightning-input label="Address" value={citizenAddress} onchange={handleCitizenAddressChange}></lightning-input>
17        <lightning-input label="Government ID" value={citizenGovId} onchange={handleCitizenGovIdChange}></lightning-input>
18        <lightning-input label="Phone Number" value={citizenPhone} onchange={handleCitizenPhoneChange}></lightning-input>
19        <lightning-button label="Create Citizen" variant="brand" class="slds-m-top_small slds-m-left_small" onclick={handleCreate}></lightning-button>
20      </template>
21      <!-- Step 3: Existing Citizen -->
22      <template iftrue={step1IsExistingCitizen}>
23        <div>Edit Existing Citizen</div>
24        <lightning-input label="Name" value={citizenName} onchange={handleCitizenNameChange}></lightning-input>
25        <lightning-input label="Email" type="email" value={citizenEmail} onchange={handleCitizenEmail}></lightning-input>
26        <lightning-input label="Address" value={citizenAddress} onchange={handleCitizenAddressChange}></lightning-input>
27        <lightning-input label="Government ID" value={citizenGovId} onchange={handleCitizenGovIdChange}></lightning-input>
28        <lightning-input label="Phone Number" value={citizenPhone} onchange={handleCitizenPhoneChange}></lightning-input>
29      </template>
30    </div>
31  </lightning-card>
32</template>

PROBLEMS OUTPUT ... Filter
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 PROBLEMS OUTPUT ... Filter
Salesforce CLI

```



Submit a Citizen Issue

Are you a new citizen or existing citizen?

New Citizen

Existing Citizen



Quick Create Urban Issue

Issue Type

Select an Option

Priority

Select an Option

Citizen

Search Citizens...



Create Issue

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.

↳ List View ⏺ History ↳ urbanissueQuickCreate

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
 - JS citizenissueSubmit.js
 - ↳ citizenissueSubmit.js-meta.xml
 - > urbanissueQuickCreate
 - > __tests__
 - ↳ urbanissueQuickCreate.html
 - JS urbanissueQuickCreate.js
 - ↳ urbanissueQuickCreate.js-meta.xml
 - { jsonconfig.json
 - > objects

UNTITLED-2

```

force-app > main > default > classes > • CitizenIssueRestService.cls > ↳ CitizenIssueRestService
2   global with sharing class CitizenIssueRestService {
6     global static String createIssue() {
9       Map<String, Object> params = (Map<String, Object>) JSON.deserializeUntyped(requestBody);
10
11       Urban_Issue__c issue = new Urban_Issue__c();
12       issue.Issue__Title__c = (String) params.get('Issue__Title__c');
13       issue.Description__c = (String) params.get('Description__c');
14       issue.Issue_Type__c = (String) params.get('Issue_Type__c');
15       issue.Location__c = (String) params.get('Location__c');
16       issue.Location_Type__c = (String) params.get('Location_Type__c');
17       issue.Status__c = (String) params.get('Status__c');
18       issue.Citizen__c = (String) params.get('Citizen__c'); // Pass Citizen record Id
19
20       insert issue;
21       return issue.Id;
22     }
23
24     // Get Urban Issues (GET) - optional basic implementation
25     @HttpGet
26     global static List<Urban_Issue__c> getIssues() {
27       // You can add filtering logic (by citizen, status, etc.) as needed
28       return [SELECT Id, Issue__Title__c, Description__c, Issue_Type__c, Location__c, Location_Type__c, Stat
29           FROM Urban_Issue__c LIMIT 100];
30     }
31   }

```

PROBLEMS **OUTPUT** ... Filter

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
 - JS citizenissueSubmit.js
 - ↳ citizenissueSubmit.js-meta.xml
 - > urbanissueQuickCreate
 - > __tests__
 - ↳ urbanissueQuickCreate.html
 - JS urbanissueQuickCreate.js
 - ↳ urbanissueQuickCreate.js-meta.xml
 - { jsonconfig.json
 - > objects
 - > permissionsets
 - > staticresources
 - > tabs
 - > triggers

CitizenController.cls

```

1  public with sharing class CitizenController {
2    @AuraEnabled
3    public static Id createCitizen(String name, String email, String address, String govId, String phone) {
4      Citizen__c citizen = new Citizen__c(
5        Name = name,
6        Email__c = email,
7        Address__c = address,
8        Government_ID__c = govId,
9        Phone_Number__c = phone
10      );
11      insert citizen;
12      return citizen.Id;
13    }
14
15    @AuraEnabled(cacheable=true)
16    public static List<Citizen__c> searchCitizens(String email) {
17      if (String.isBlank(email)) return new List<Citizen__c>();
18      return [SELECT Id, Name, Email__c FROM Citizen__c WHERE Email__c LIKE :('%' + email + '%') LIMIT 10];
19    }
20  }

```

PROBLEMS **OUTPUT** ... Filter

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

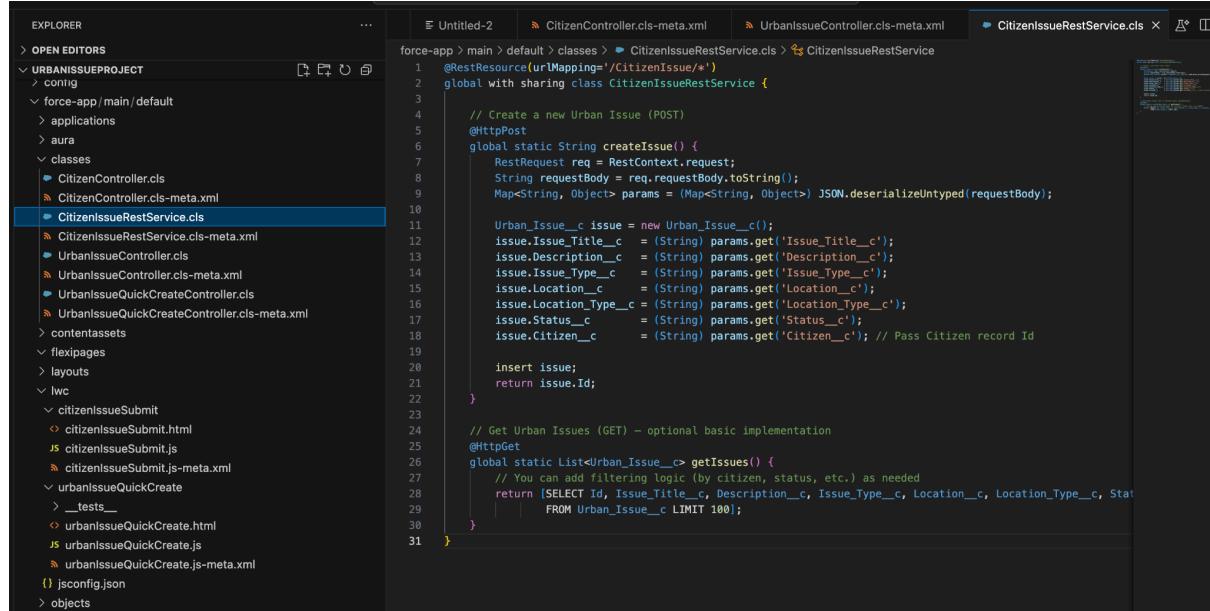
Use Case:

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.



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

- EXPLORER** pane on the left showing the project structure under `URBANISSUEPROJECT`, including files like `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`, `urbanissueQuickCreate.html`, `urbanissueQuickCreate.js`, and `urbanissueQuickCreate.js-meta.xml`.
- UNTITLED-2** editor tab showing the Apex code for `CitizenIssueRestService.cls`.
- CitizenIssueRestService.cls** code content:

```
force-app > main > default > classes > CitizenIssueRestService.cls > CitizenIssueRestService
1  @RestResource(urlMapping='/CitizenIssue/*')
2  global with sharing class CitizenIssueRestService {
3
4      // Create a new Urban Issue (POST)
5      @HttpPost
6      global static String createIssue() {
7          RestRequest req = RestContext.request;
8          String requestBody = req.requestBody.toString();
9          Map<String, Object> params = (Map<String, Object>) JSON.deserializeUntyped(requestBody);
10
11         Urban_Issue__c issue = new Urban_Issue__c();
12         issue.Issue_Title__c = (String) params.get('Issue_Title__c');
13         issue.Description__c = (String) params.get('Description__c');
14         issue.Issue_Type__c = (String) params.get('Issue_Type__c');
15         issue.Location__c = (String) params.get('Location__c');
16         issue.Location_Type__c = (String) params.get('Location_Type__c');
17         issue.Status__c = (String) params.get('Status__c');
18         issue.Citizen__c = (String) params.get('Citizen__c'); // Pass Citizen record Id
19
20         insert issue;
21         return issue.Id;
22     }
23
24     // Get Urban Issues (GET) - optional basic implementation
25     @HttpGet
26     global static List<Urban_Issue__c> getIssues() {
27         // You can add filtering logic (by citizen, status, etc.) as needed
28         return [SELECT Id, Issue_Title__c, Description__c, Issue_Type__c, Location__c, Location_Type__c, Status__c
29              | FROM Urban_Issue__c LIMIT 100];
30     }
31 }
```

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.

The screenshot shows the 'Manage External Client Apps' page in Salesforce. The app named 'UrbanIssueAPI' is selected. Key details shown include:

- Contact Email: vinayak22436@gmail.com
- App Authorization: All users can self-authorize
- Type: Local
- App Status: Enabled

The 'Settings' tab is active. Under 'Basic Information', the following fields are filled:

External Client App Name: UrbanIssueAPI	API Name: UrbanIssueAPI
Contact Email: vinayak22436@gmail.com	Distribution State: Local
Contact Phone: 6300864973	Info URL: Enter a URL...
Icon URL: Enter a URL...	Logo Image URL: Enter a URL...
Description: (empty)	

3. Named Credentials

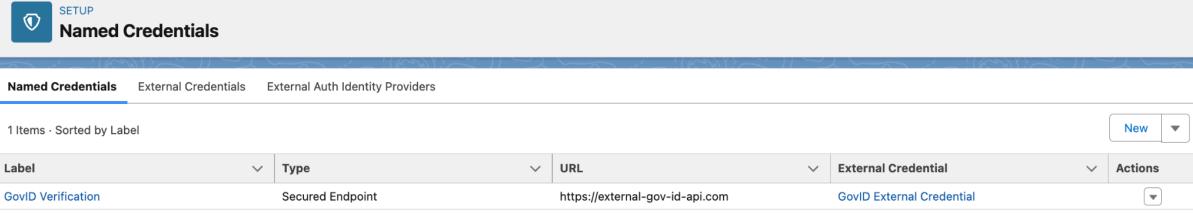
Use Case:

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.



The screenshot shows the Salesforce 'Named Credentials' setup page. At the top, there's a 'SETUP' button and a 'Named Credentials' section. Below that, there are tabs for 'Named Credentials', 'External Credentials', and 'External Auth Identity Providers'. A message '1 Items - Sorted by Label' is displayed. On the right, there are 'New' and a dropdown menu buttons. The main area is a table with columns: 'Label', 'Type', 'URL', 'External Credential', and 'Actions'. One row is visible, labeled 'GovID Verification', with 'Secured Endpoint' as the type, 'https://external-gov-id-api.com' as the URL, and 'GovID External Credential' as the external credential. There's also a small dropdown arrow next to the 'Actions' column.

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.

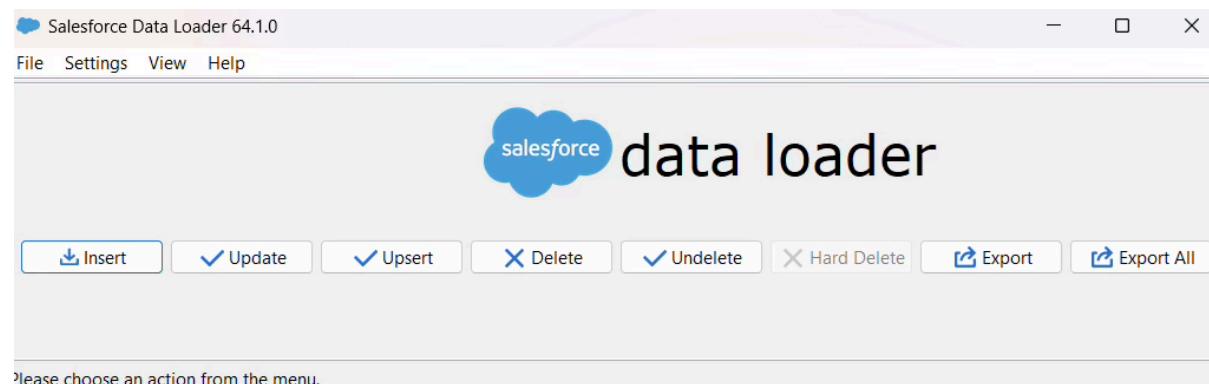
1. Data Import Wizard

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.



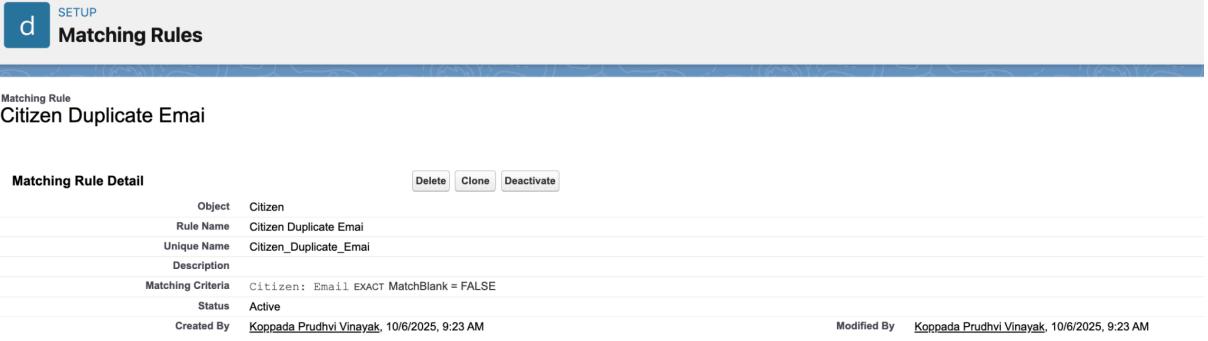
2. Duplicate Rules

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.

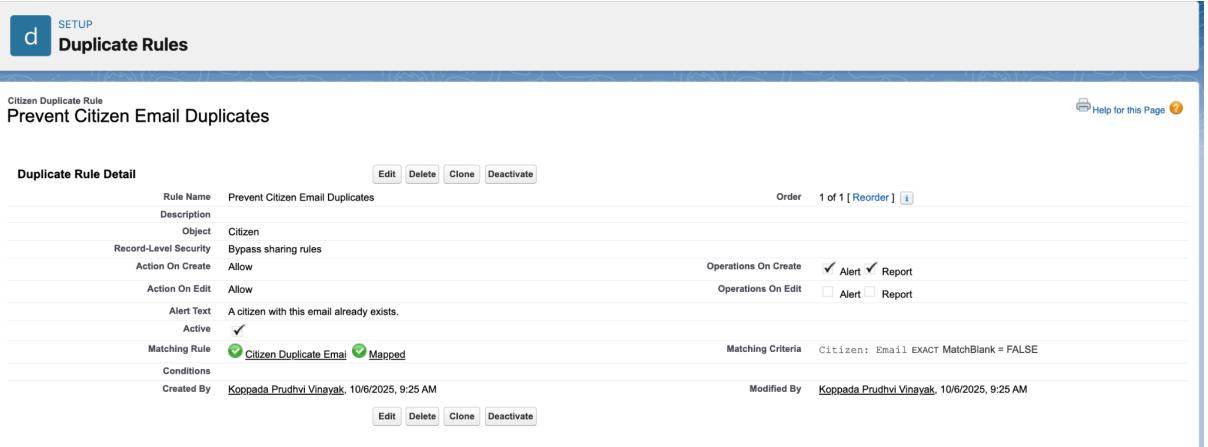
Implementation:

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.



The screenshot shows the Matching Rules page in Salesforce. A specific rule named "Citizen Duplicate Email" is selected. The rule details are as follows:

Object	Citizen
Rule Name	Citizen Duplicate Email
Unique Name	Citizen_Duplicate_Email
Description	
Matching Criteria	Citizen: Email EXACT MatchBlank = FALSE
Status	Active
Created By	Koppada Prudhvi Vinayak, 10/6/2025, 9:23 AM
Modified By	Koppada Prudhvi Vinayak, 10/6/2025, 9:23 AM



The screenshot shows the Duplicate Rules page in Salesforce. A specific rule named "Prevent Citizen Email Duplicates" is selected. The rule details are as follows:

Rule Name	Prevent Citizen Email Duplicates
Description	
Object	Citizen
Record-Level Security	Bypass sharing rules
Action On Create	Allow
Action On Edit	Allow
Alert Text	A citizen with this email already exists.
Active	✓
Matching Rule Conditions	✓ Citizen Duplicate Email Mapped
Created By	Koppada Prudhvi Vinayak, 10/6/2025, 9:25 AM
Modified By	Koppada Prudhvi Vinayak, 10/6/2025, 9:25 AM

3. Data Export & Backup

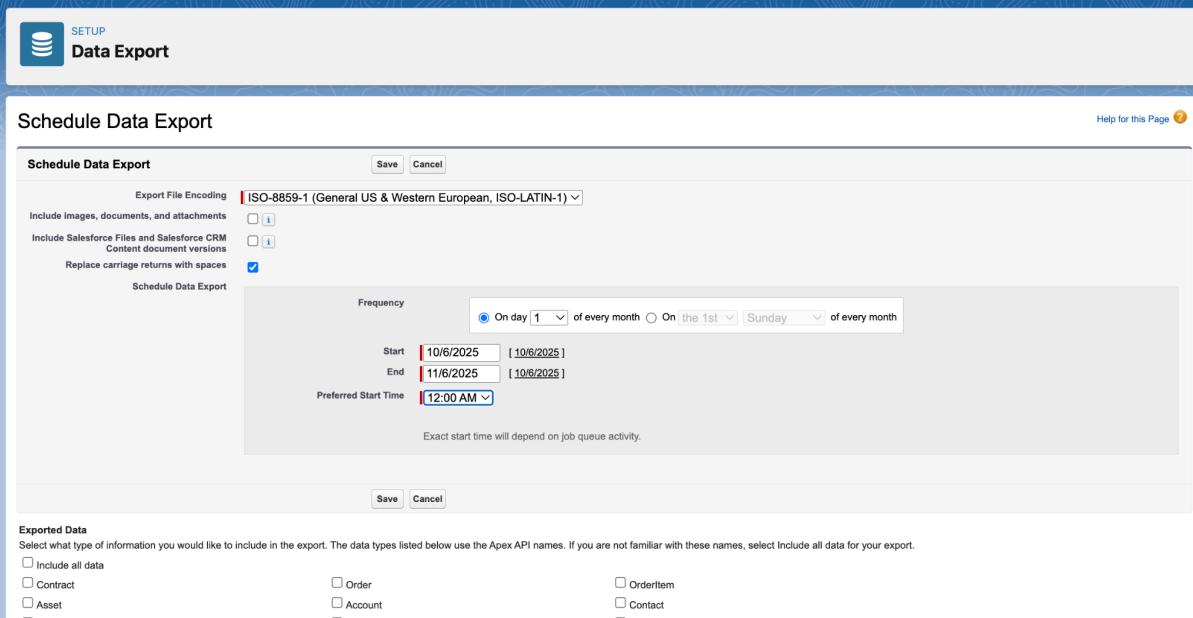
Use Case:

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.

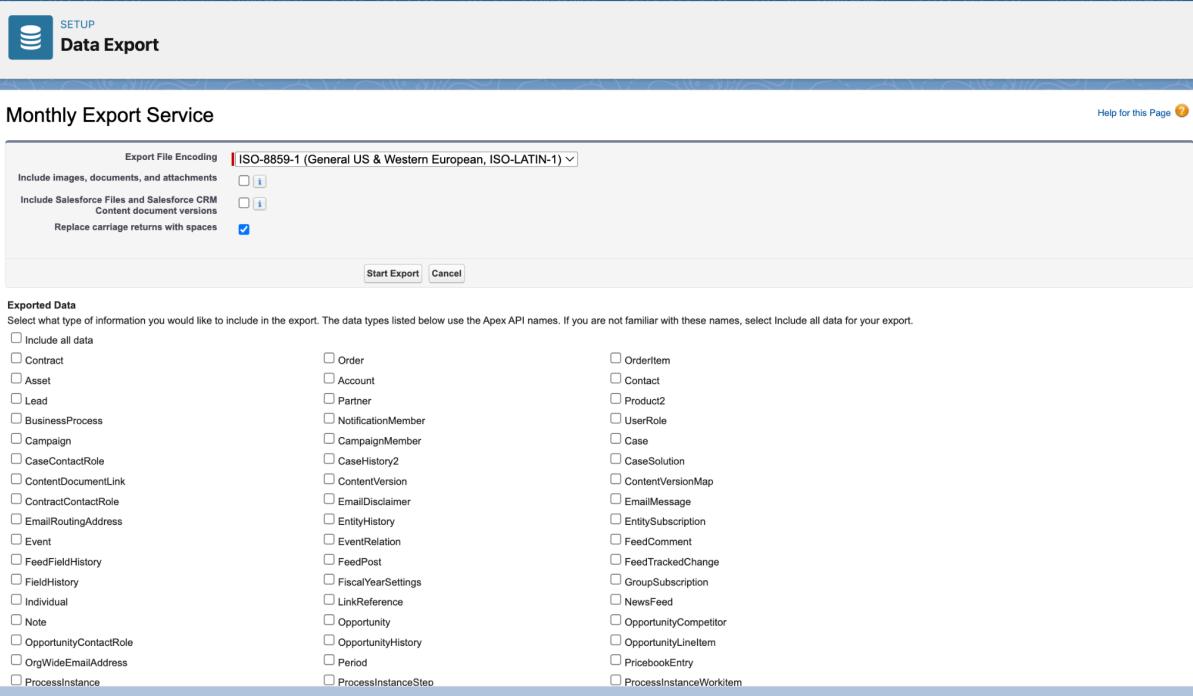


The screenshot shows the 'Schedule Data Export' page in the Salesforce Setup interface. It includes fields for export file encoding (ISO-8859-1), frequency (On day 1 of every month), and start/end dates/times. A note states: "Exact start time will depend on job queue activity." Buttons for Save and Cancel are at the bottom.

Exported Data

Select what type of information you would like to include in the export. The data types listed below use the Apex API names. If you are not familiar with these names, select Include all data for your export.

<input type="checkbox"/> Include all data	<input type="checkbox"/> Order	<input type="checkbox"/> OrderItem
<input type="checkbox"/> Contract	<input type="checkbox"/> Account	<input type="checkbox"/> Contact
<input type="checkbox"/> Asset	<input type="checkbox"/> Partner	<input type="checkbox"/> Product2
<input type="checkbox"/> Lead	<input type="checkbox"/> NotificationMember	<input type="checkbox"/> UserRole
<input type="checkbox"/> BusinessProcess	<input type="checkbox"/> CampaignMember	<input type="checkbox"/> Case
<input type="checkbox"/> Campaign		



The screenshot shows the 'Monthly Export Service' page in the Salesforce Setup interface. It includes fields for export file encoding (ISO-8859-1), frequency (On day 1 of every month), and start/end dates/times. A note states: "Exact start time will depend on job queue activity." Buttons for Start Export and Cancel are at the bottom.

Exported Data

Select what type of information you would like to include in the export. The data types listed below use the Apex API names. If you are not familiar with these names, select Include all data for your export.

<input type="checkbox"/> Include all data	<input type="checkbox"/> Order	<input type="checkbox"/> OrderItem
<input type="checkbox"/> Contract	<input type="checkbox"/> Account	<input type="checkbox"/> Contact
<input type="checkbox"/> Asset	<input type="checkbox"/> Partner	<input type="checkbox"/> Product2
<input type="checkbox"/> Lead	<input type="checkbox"/> NotificationMember	<input type="checkbox"/> UserRole
<input type="checkbox"/> BusinessProcess	<input type="checkbox"/> CampaignMember	<input type="checkbox"/> Case
<input type="checkbox"/> Campaign	<input type="checkbox"/> CaseHistory2	<input type="checkbox"/> CaseSolution
<input type="checkbox"/> CaseContactRole	<input type="checkbox"/> ContentVersion	<input type="checkbox"/> ContentVersionMap
<input type="checkbox"/> ContentDocumentLink	<input type="checkbox"/> EmailDisclaimer	<input type="checkbox"/> EmailMessage
<input type="checkbox"/> ContractContactRole	<input type="checkbox"/> EntityHistory	<input type="checkbox"/> EntitySubscription
<input type="checkbox"/> EmailRoutingAddress	<input type="checkbox"/> EventRelation	<input type="checkbox"/> FeedComment
<input type="checkbox"/> Event	<input type="checkbox"/> FeedPost	<input type="checkbox"/> FeedTrackedChange
<input type="checkbox"/> FeedFieldHistory	<input type="checkbox"/> FiscalYearSettings	<input type="checkbox"/> GroupSubscription
<input type="checkbox"/> FieldHistory	<input type="checkbox"/> LinkReference	<input type="checkbox"/> NewsFeed
<input type="checkbox"/> Individual	<input type="checkbox"/> Opportunity	<input type="checkbox"/> OpportunityCompetitor
<input type="checkbox"/> Note	<input type="checkbox"/> OpportunityHistory	<input type="checkbox"/> OpportunityLineItem
<input type="checkbox"/> OpportunityContactRole	<input type="checkbox"/> Period	<input type="checkbox"/> PricebookEntry
<input type="checkbox"/> OrgWideEmailAddress	<input type="checkbox"/> ProcessInstanceStep	<input type="checkbox"/> ProcessInstanceWorkitem
<input type="checkbox"/> ProcessInstance		

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.

```
1  @RestResource(urlMapping
2  global with sharing clas
3
4  // Create a new Urba
5  @HttpPost
6  global static String
7  RestRequest req
8  String requestBody
9  Map<String, Obj
10
```

- Open to the Side
- Open With...
- Reveal in Finder
- Open in Integrated Terminal
- Select for Compare
- Open Timeline
- Add File to Chat
- Cut ⌘ X
- Copy ⌘ C
- Copy Path ⌘ ⌥ C
- Copy Relative Path ⌘ ⌄ ⌥ C
- Rename...
- Delete ⌘ ⌄
- SFDX: Clear Code Analyzer Violations from Selected Files or Folders
- SFDX: Create OpenAPI Document from This Class (Beta)
- SFDX: Delete from Project and Org
- SFDX: Deploy This Source to Org
- SFDX: Diff File Against Org
- SFDX: Generate Manifest File
- SFDX: Retrieve This Source from Org
- SFDX: Scan Selected Files or Folders with Code Analyzer

PHASE 9 - Reporting, Dashboards & Security Review

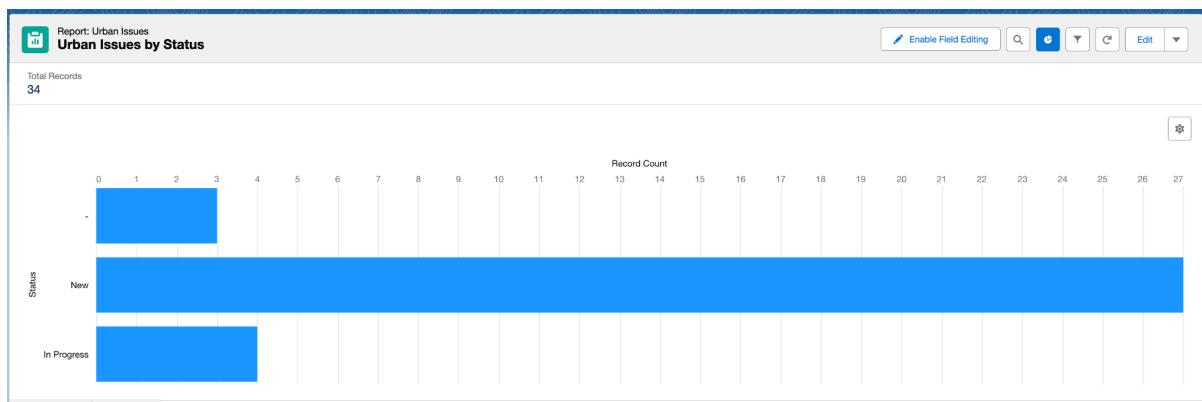
This phase focused on providing actionable insights through reporting, enabling visual representation of key metrics via dashboards, and ensuring the solution is secure and ready for deployment.

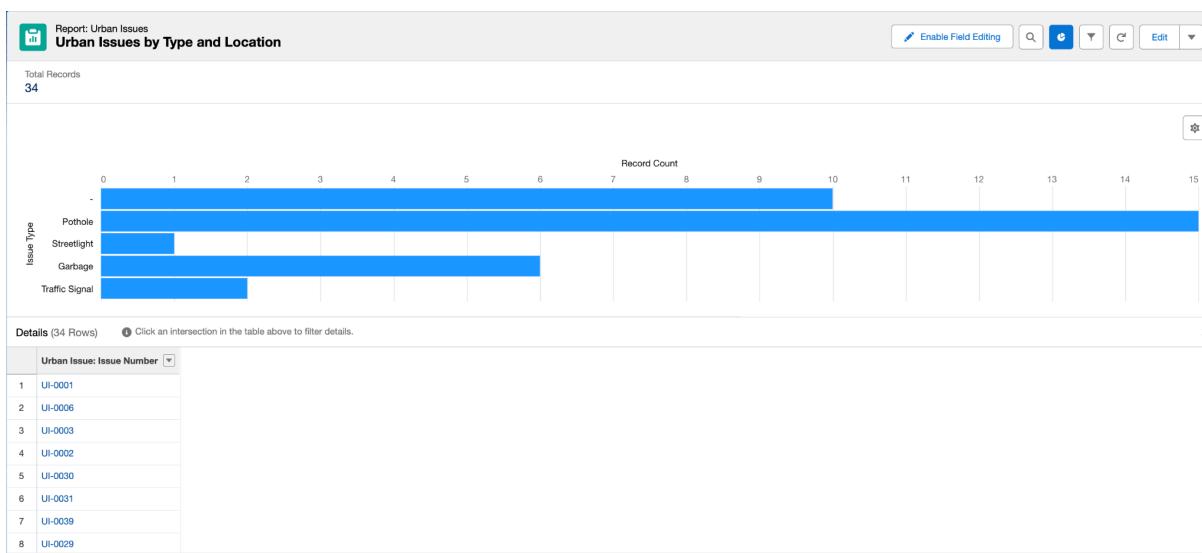
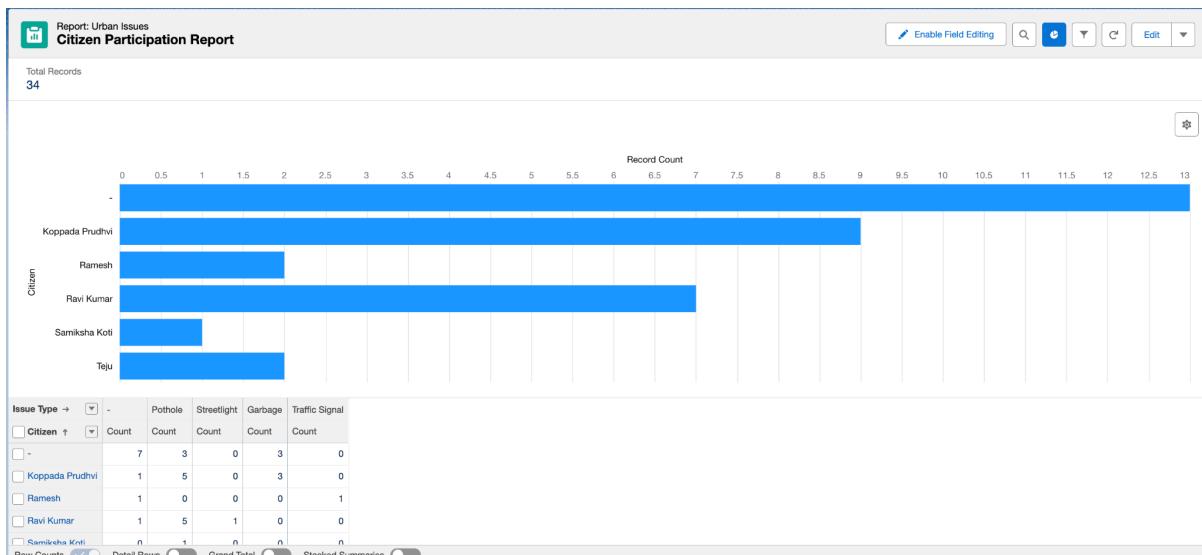
Reporting

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.



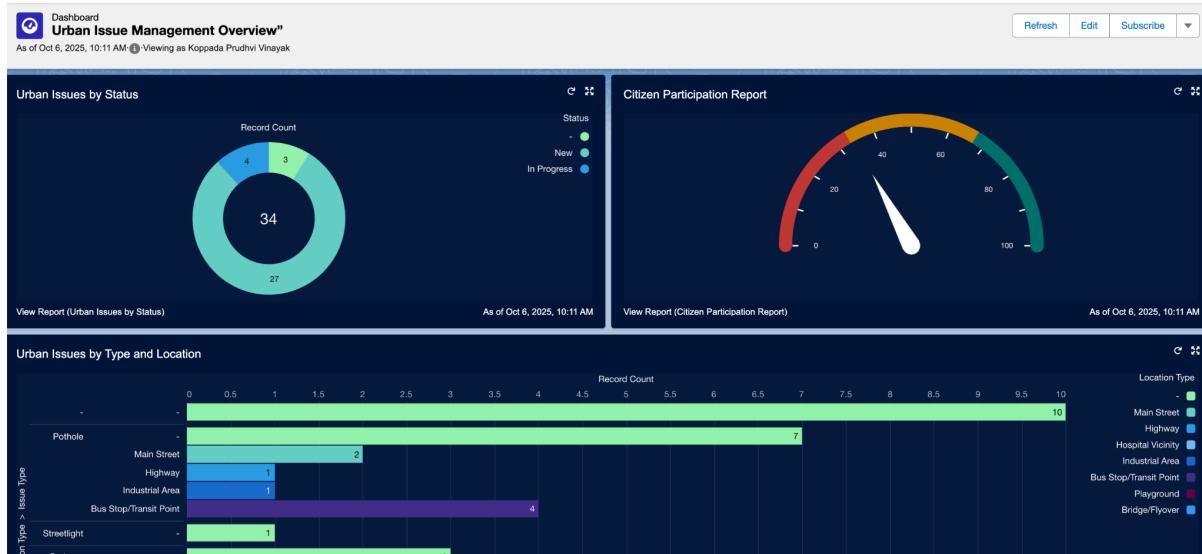


◆ Dashboards

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).

Custom Object Permissions										
	Basic Access			Data Administration						
	Read	Create	Edit	Delete	View All Records	Modify All Records	View All Fields	Read	Create	Edit
Citizens	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Departments	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Feedbacks	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
Staffs	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Urban Issues	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				

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.