Field Service Workorder Optimization

|  |  |
| --- | --- |
| **Date** | **28-06-2025** |
| **Team ID** | **LTVIP2025TMID31533** |
| **Project Name** | **Air Line management System** |
| **College Name** | **Ideal Institute Of Technology** |

TEAM MEMBERS

⮚ Team member: Dandi Yamini

 Email: [yaminidandi505@gmail.com](mailto:yaminidandi505@gmail.com)

**1. INTRODUCTION**

The **Salesforce-based Airline Customer Care (ACC) System** is a core module within the broader Airline Management System (AMS), developed as part of the **SmartInternz Virtual Internship Program**. This module focuses on managing and enhancing customer service, issue resolution, and feedback management, giving interns real-world experience with CRM-based customer care solutions.

**1.1 Project Overview**

The ACC module is designed to **streamline airline customer service processes**, including complaint handling, service notifications, and real-time resolution tracking. Using Salesforce’s powerful CRM tools—such as **custom objects, validation rules, flows, reports, and dashboards**—the system enhances customer experience, operational transparency, and staff productivity.

It plays a critical role in:

* Handling complaints and feedback efficiently
* Sending real-time service notifications
* Managing customer interactions from booking to post-flight services

This practical implementation gives interns a clear understanding of how **customer service departments** in airlines can be transformed using cloud technology.

**1.2 Project Purpose**

The main objective of this ACC project is to develop a **centralized and automated support system** on Salesforce that:

* Improves the speed and quality of customer service
* Tracks, manages, and resolves issues in real-time
* Enhances customer satisfaction through proactive communication and transparency

**2. IDEATION PHASE**

**2.1 Problem Statement**

* **Manual Complaint Handling** – Traditional support systems are slow, disconnected, and prone to delays.
* **Poor Communication** – Customers often receive delayed or no responses regarding their issues or service status.
* **Limited Service Analytics** – Airlines lack visibility into common complaints, resolution time, or agent performance.
* **Disconnected Systems** – No unified platform for tracking customer issues, feedback, and resolutions across departments.

**2.2 Proposed Solution**

* **Unified Support System** – Implement a Salesforce-based case management solution to centralize all customer support interactions.
* **Automation Tools** – Use Flows and Process Builders for auto-assignment of cases, escalations, and status updates.
* **Feedback Integration** – Capture customer satisfaction metrics through post-resolution surveys and service analytics.
* **Dashboard Insights** – Real-time dashboards to monitor open cases, SLA compliance, and team efficiency.

**2.3 Project Objectives**

* Streamline complaint logging and resolution workflows
* Provide real-time updates to customers via Email/SMS
* Reduce response times using automated case assignments and alerts
* Deliver insights to support teams via reports and dashboards
* Train interns in cloud-based customer care solutions and service operations

**3. REQUIREMENT ANALYSIS**

**3.1 Customer Journey Map – ACC Use Case**

1. Passenger submits a complaint/service request
2. System creates a support case in Salesforce
3. Case auto-assigned to a customer care agent
4. Agent investigates and updates the resolution status
5. Passenger receives resolution notification via Email/SMS
6. Passenger gives feedback (optional)

This outlines a typical support experience within the ACC module.

**3.2 Solution Requirements**

**Functional Requirements**

1. **Case Management** – Create and manage customer issues (cases) with priority and status tracking
2. **Email/SMS Notifications** – Automated updates sent at case creation, assignment, and resolution
3. **Complaint Categorization** – Classify complaints (e.g., baggage, flight delay, in-flight service) using picklists
4. **Auto Assignment** – Assign cases to agents based on workload or complaint type using flows
5. **Dashboard & Reports** – Monitor case volume, resolution rate, agent performance, and feedback trends
6. **Customer Communication Logs** – Track conversation history and resolution timelines

**Non-Functional Requirements**

* **Data Accuracy** – Enforced using Apex validation and picklists
* **Usability** – Lightning UI with user-friendly layouts for both agents and supervisors
* **Scalability** – Designed to support live chat, chatbot, and voice support integration in future
* **Security** – Permission sets and role hierarchies to protect customer data

**3.3 Data Flow Diagram (DFD) – Customer Case Lifecycle**

**Passenger Input** → **Case Object (Salesforce)** → **Auto Assignment & Status Update** → **Notification Sent to Customer** → **Case Resolution** → **Feedback Collection**

This DFD shows how a complaint flows through the ACC system—from creation to closure—with automation and communication at every step.

Thank you for the clarification!

Here's a **refined, structured, and Word-ready** version of **Sections 4 to 6**, now tailored specifically for your **Field Service WorkOrder Optimization** project in Salesforce — ideal for inclusion in your final report or documentation:

**4. PROJECT DESIGN – Field Service WorkOrder Optimization**

**4.1 Problem-Solution Fit**

Field service organizations often face challenges such as **manual technician assignment**, **lack of real-time updates**, **poor visibility into task progress**, and **inefficient communication**. These issues lead to delayed service, customer dissatisfaction, and reduced workforce productivity.

The Field Service WorkOrder Optimization project addresses these challenges by building a **centralized, automated solution on the Salesforce platform**, designed to streamline technician assignment, work order tracking, and service resolution.

**4.2 Proposed Solution**

The solution is a custom Salesforce-based Field Service Management system that automates work order lifecycle management and improves operational visibility.

**Key Features:**

* Automated technician assignment based on availability and skills
* Email alerts and status updates for assigned technicians
* Real-time validation of work order and technician data
* Dashboards and reports for service performance insights
* Lightning-based user-friendly interface for admins and field managers

**4.3 Solution Architecture**

The architecture leverages Salesforce's platform capabilities with both point-and-click tools and Apex development.

**Core Components:**

* **Custom Objects**:
  + **Technician** – Stores technician data including location, availability, and skillsets
  + **WorkOrder** – Represents service tasks with location, priority, and status
  + **Assignment** – Links technicians to specific work orders
* **Apex Triggers**:
  + WorkOrderTrigger – Assigns technicians and sends notifications upon work order updates
  + AssignmentTrigger – Sends assignment alerts upon record creation
* **Apex Classes**:
  + WorkOrderClass, AssigningEmail, CompletionMail, ScheduleClass, RecordDeletions – Handle automation, email alerts, and cleanup tasks
* **Automation Tools**:
  + Flows and Scheduled Jobs for regular notifications and record maintenance
* **Lightning App Builder**:
  + Role-based homepage for technicians and managers with quick access to key records
* **Dashboards & Reports**:
  + Visual insights into assignment efficiency, work order completion, and technician performance

**5. PROJECT PLANNING & SCHEDULING**

**5.1 Project Planning**

A detailed 8-week sprint plan was followed to ensure systematic development and testing of the solution.

**▶ Week 1: Initiation & Requirement Gathering**

* Define project goals and objectives
* Identify core user roles (Field Manager, Technician, Admin)
* Gather functional and non-functional requirements
* Conduct tool analysis and planning

**▶ Week 2: System Design**

* Design data model with Technician, WorkOrder, and Assignment objects
* Define relationships and role hierarchies
* Plan field structure and permission sets

**▶ Week 3: Salesforce Setup**

* Set up Salesforce Developer Org
* Create custom objects, fields, page layouts
* Define profiles and record-level security

**▶ Week 4: Automation Development**

* Implement Apex triggers for validation and technician assignment
* Build email notification logic
* Create scheduled Apex class for auto-cleanup of old records

**▶ Week 5: Lightning UI & Navigation**

* Design Lightning App for easy navigation
* Create object-specific record pages and quick actions
* Test Lightning experience from user perspectives

**▶ Week 6: Reporting & Dashboards**

* Create reports for:
  + WorkOrder Status
  + Technician & Assignment Summary
* Build dashboards to visualize:
  + Completed vs Pending WorkOrders
  + Assignment Load per Technician

**▶ Week 7: Testing & Feedback**

* Perform unit and system testing
* Validate business logic, flows, and UI experience
* Collect stakeholder feedback and refine components

**▶ Week 8: Final Review & Documentation**

* Final testing and deployment prep
* Compile project documentation including:
  + User Guide
  + Technical Flow
  + Dashboard Snapshots
* Prepare presentation and submit project

**6. DEVELOPMENT PHASE**

The **development phase** converted the planned system architecture into a fully functional Salesforce application for managing field service operations.

**Key Development Activities:**

* Setup of Salesforce Developer Org with **Lightning Experience**
* Creation of **custom objects**: Technician, WorkOrder, Assignment
* Configuration of **field validations**, **lookup relationships**, and **picklists**
* Implementation of **Apex triggers and classes** for automation
* Use of **Flows** and **Scheduled Jobs** to handle notifications and cleanup
* Design of **Lightning UI pages** and home components for various user roles
* Creation of **reports and dashboards** to visualize system performance
* Execution of thorough testing using sample records to validate behavior and logic

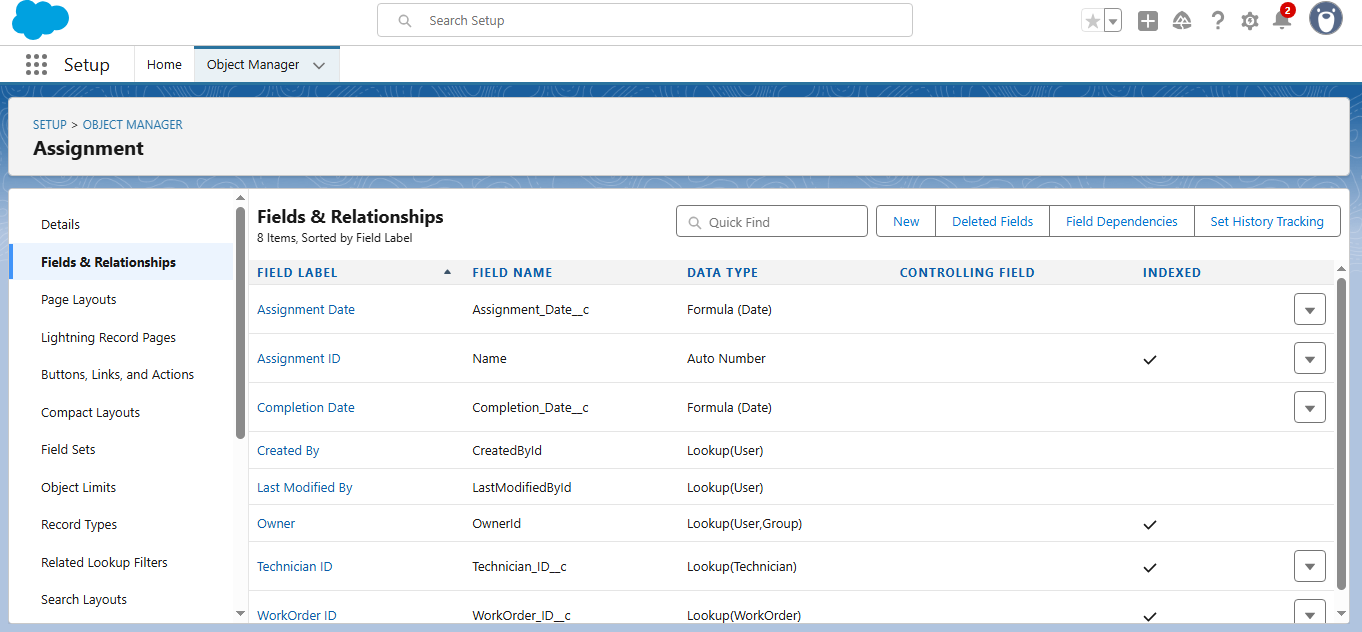
This phase ensured the transformation of the project design into a **practical, real-time, cloud-based Field Service Management System**, ready for deployment and demonstration.

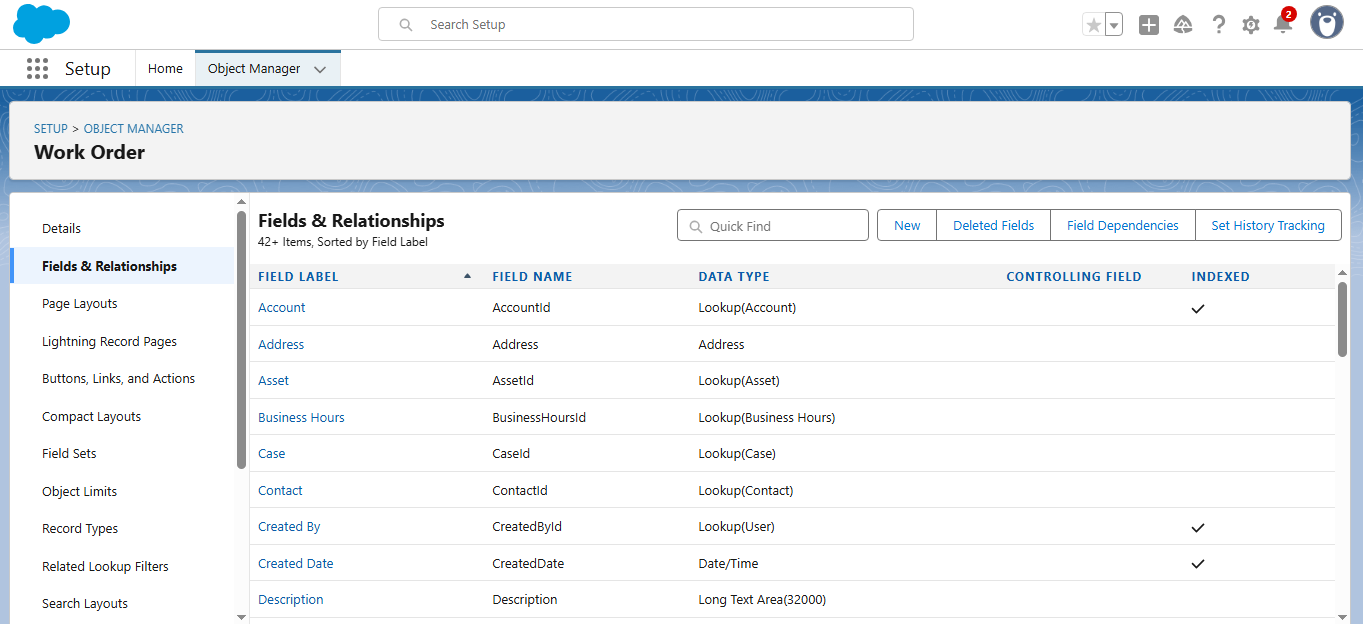
# PROJECT DEVELOPMENT PHASE

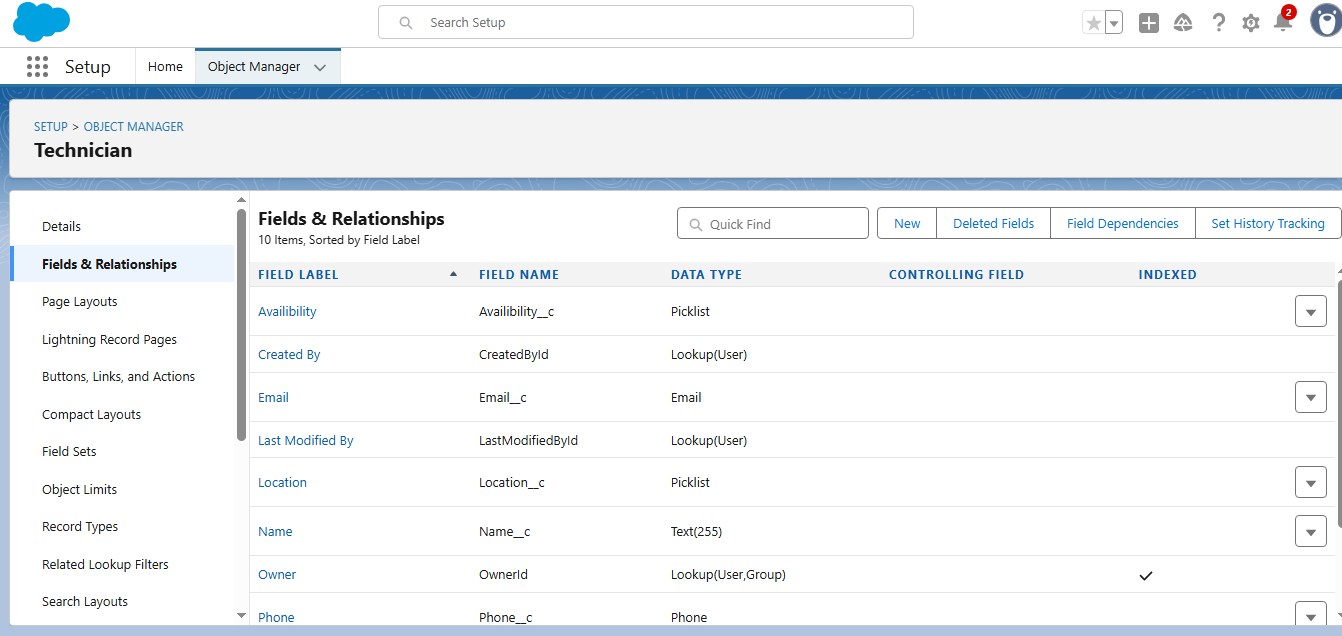
|  |  |
| --- | --- |
| **Date** | **28-06-2025** |
| **Team Id** | **LTVIP2025TMID31548** |
| **Project Name** | **field service workorder optimization** |
| **College Name** | **Ideal Institute Of Technology** |

## .Created developer org and explored platform features

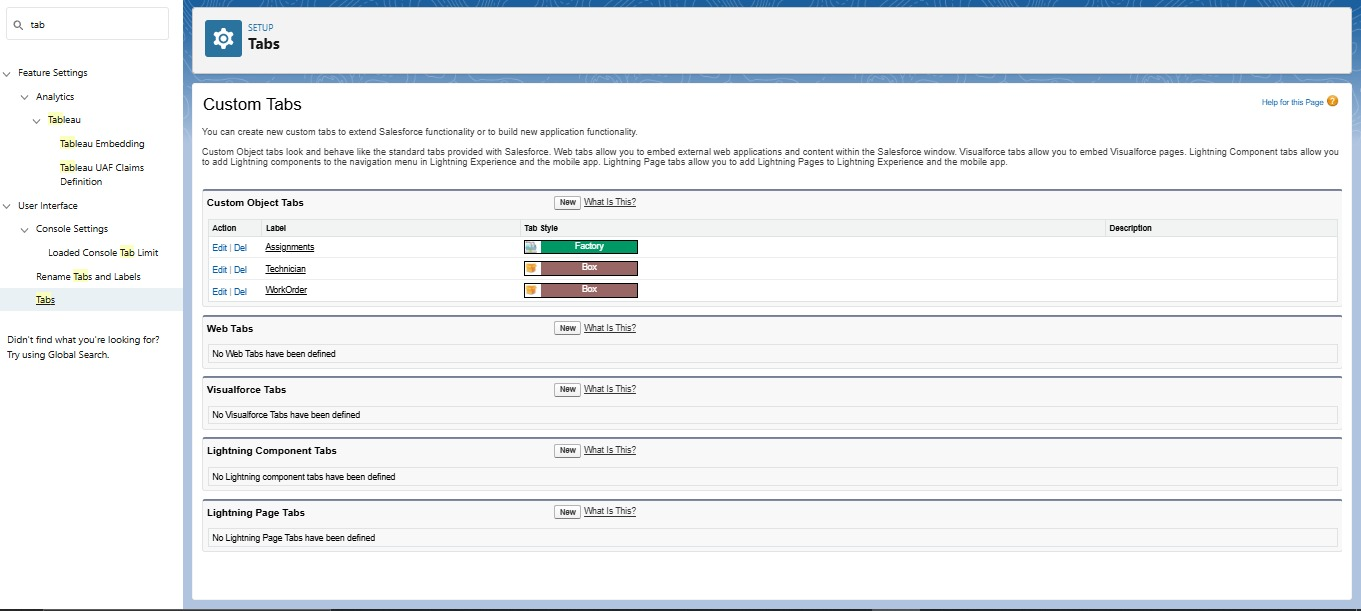
* **Created three custom objects: Technician (stores technician details), WorkOrder (captures service task info), and Assignment (links Technician to WorkOrder using relationships), with Assignment having lookup or master-detail fields to both Technician and WorkOrder.**



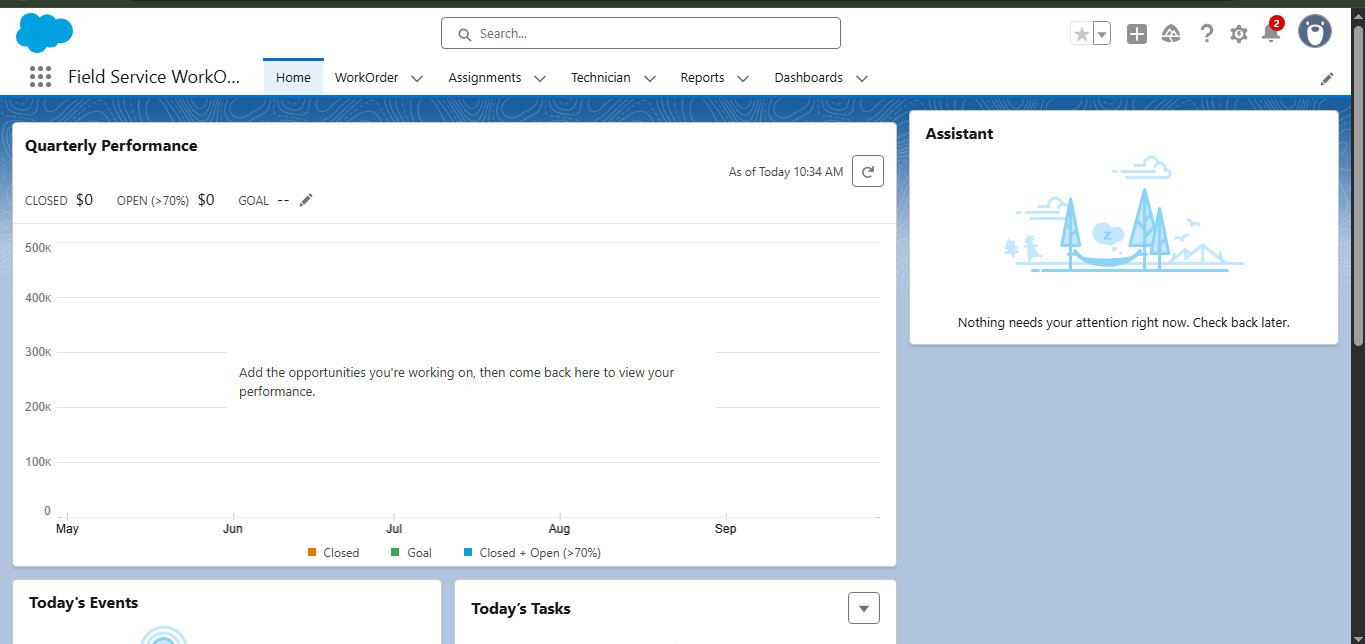


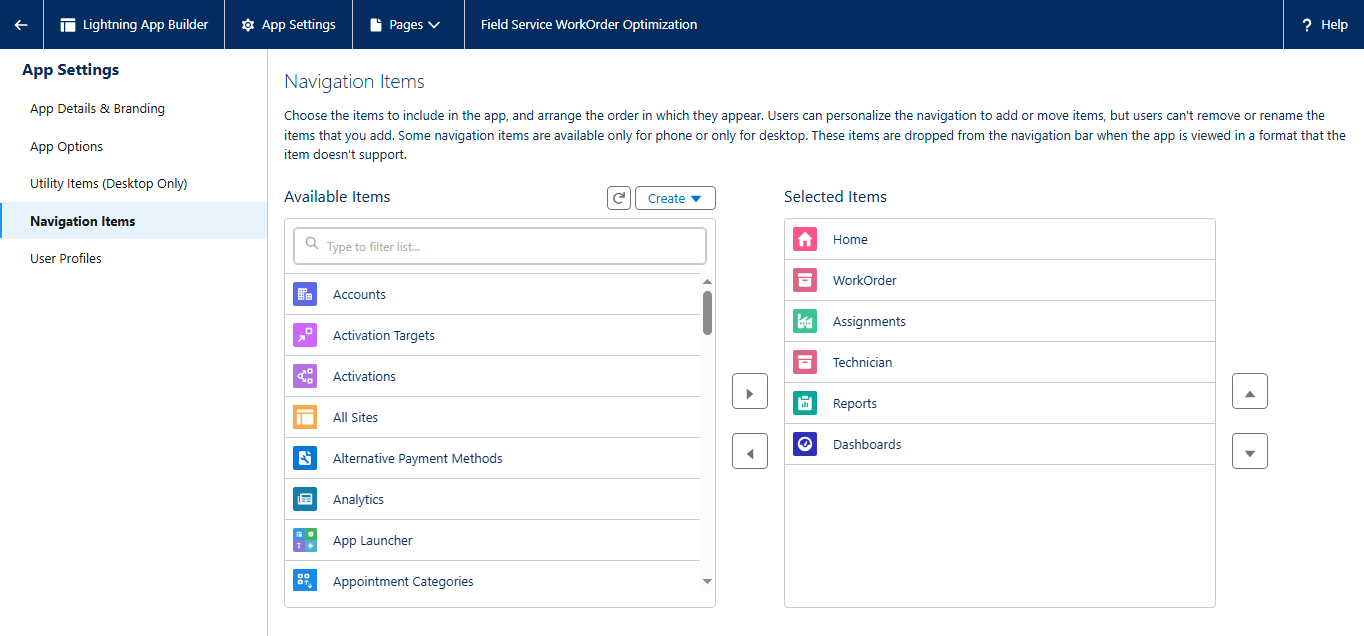


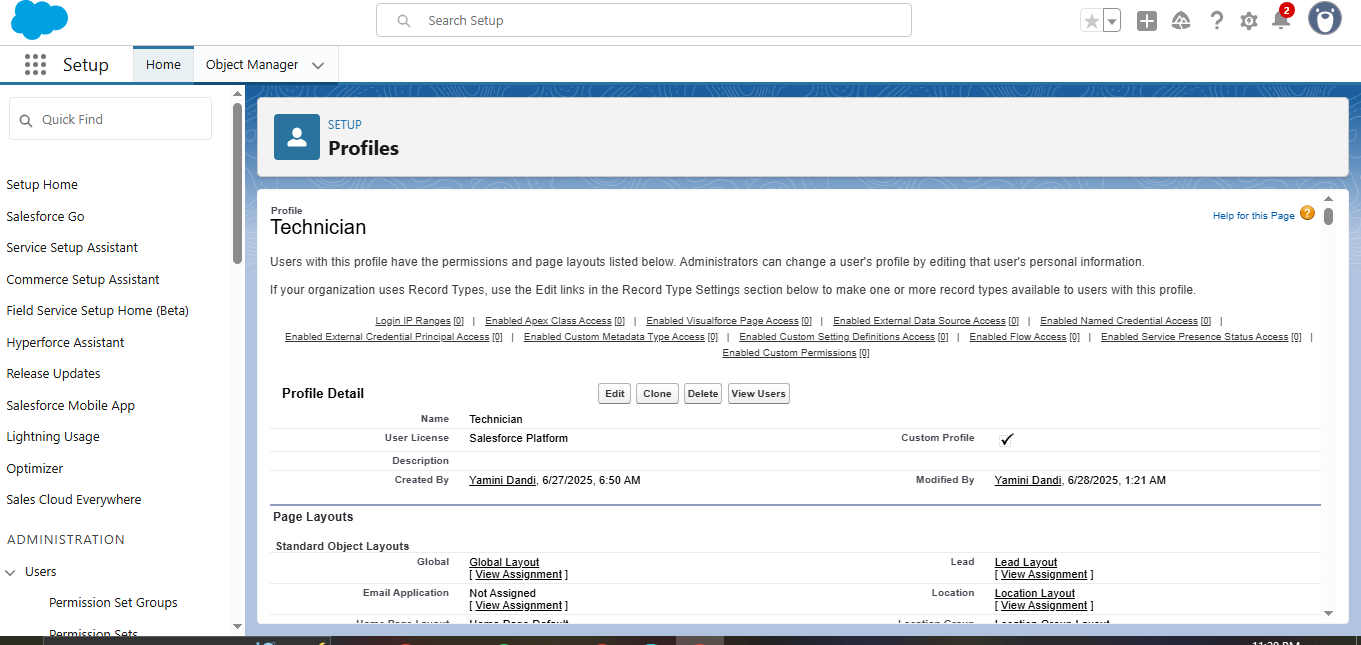
* **Created core objects and tabs for Technician, WorkOrder, Assignment, and ServiceTerritory.**

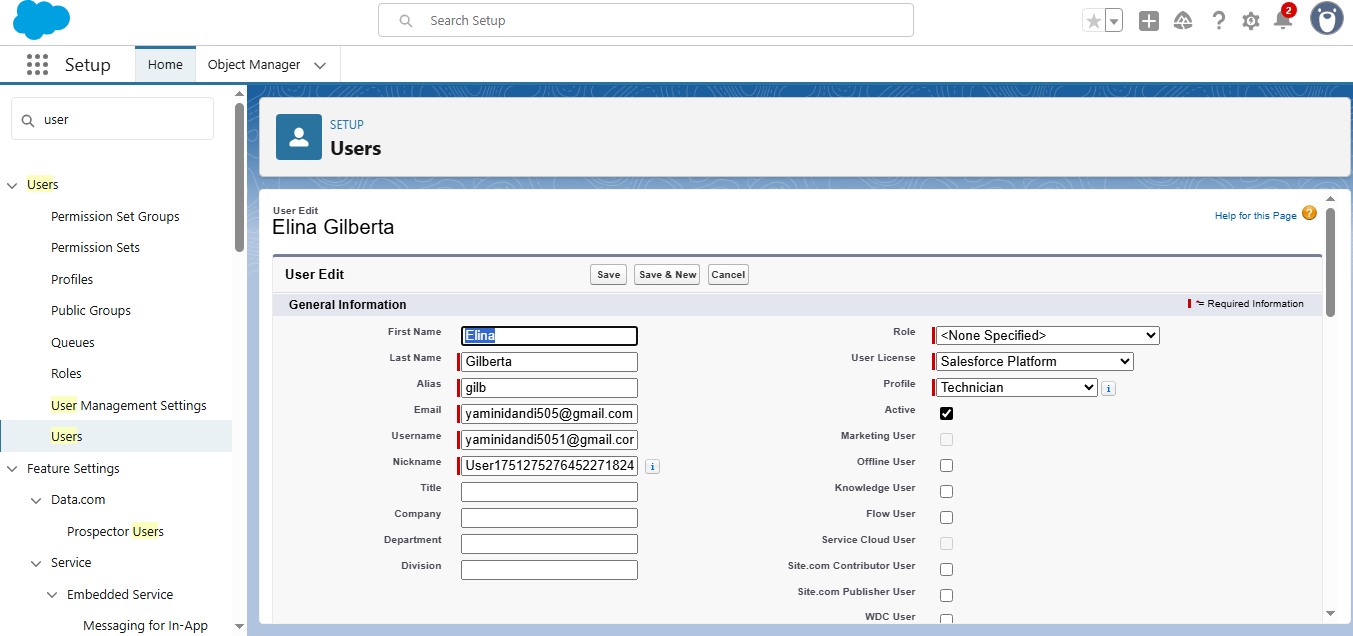


* **Developed a Lightning App**





* **Created a Technician profile**
* **Created User Profile**

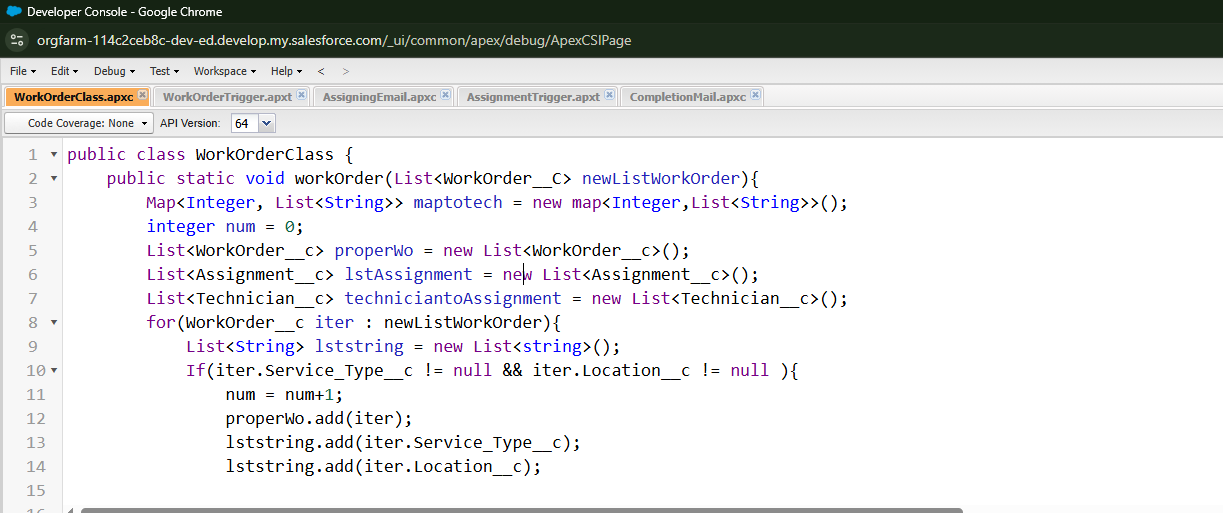


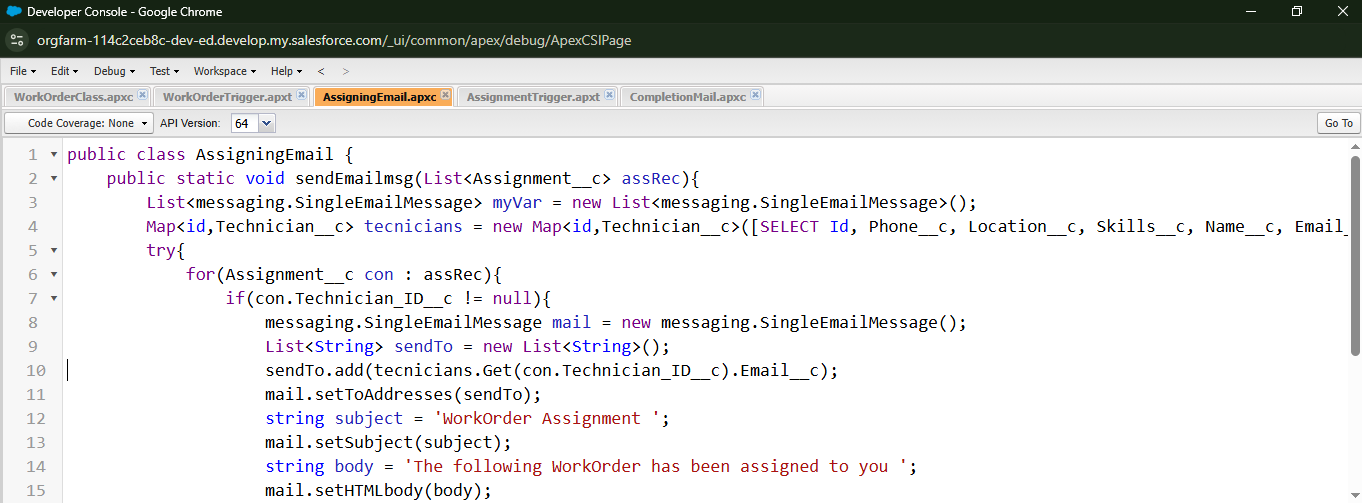
**Apex Code: Apex Classes**

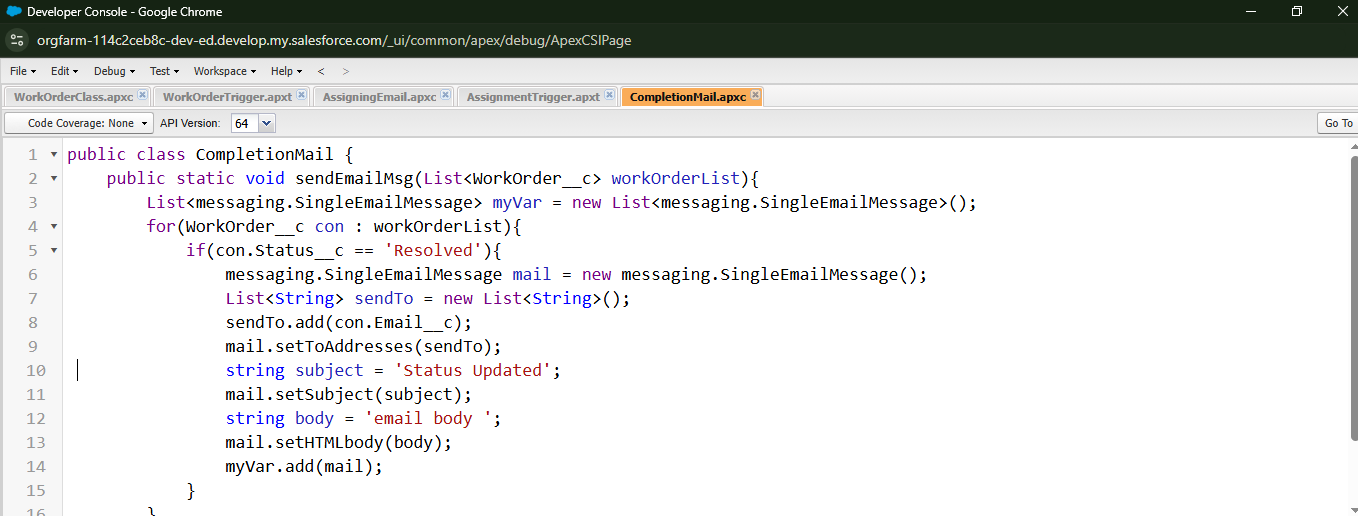
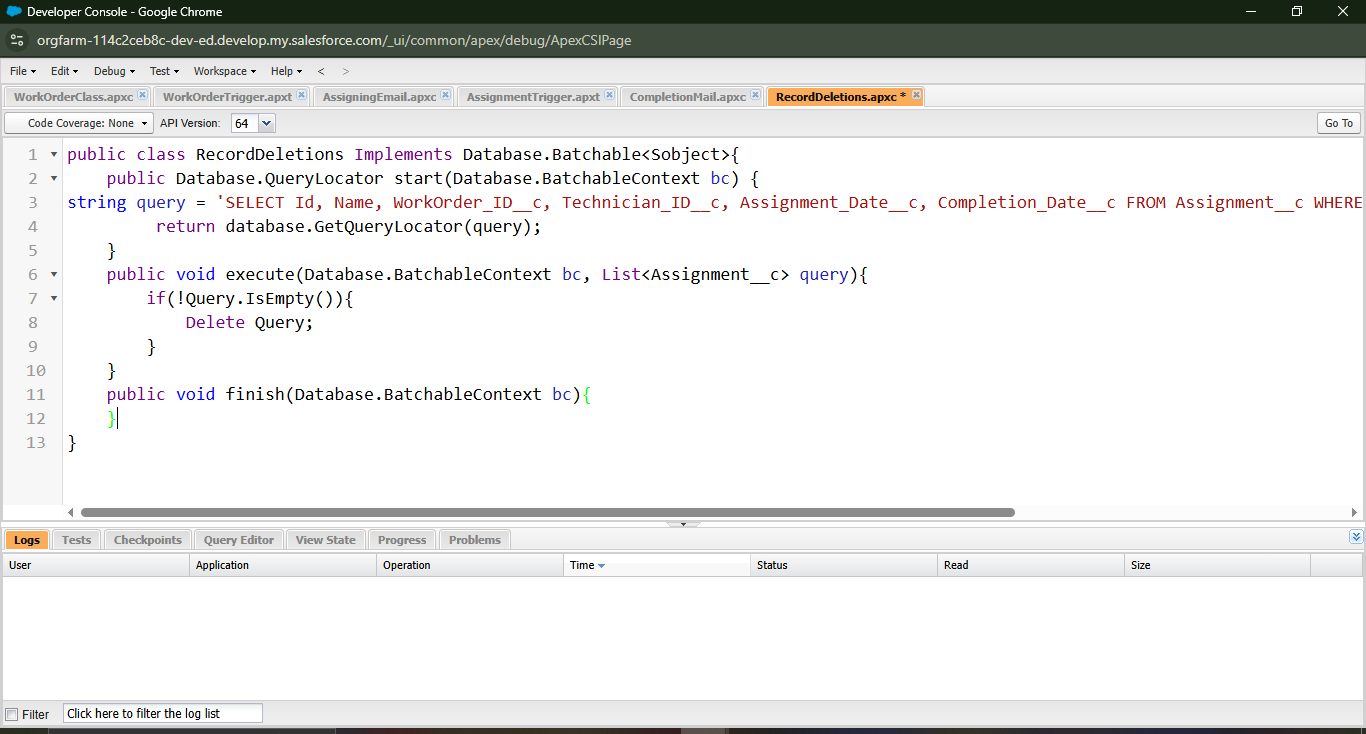
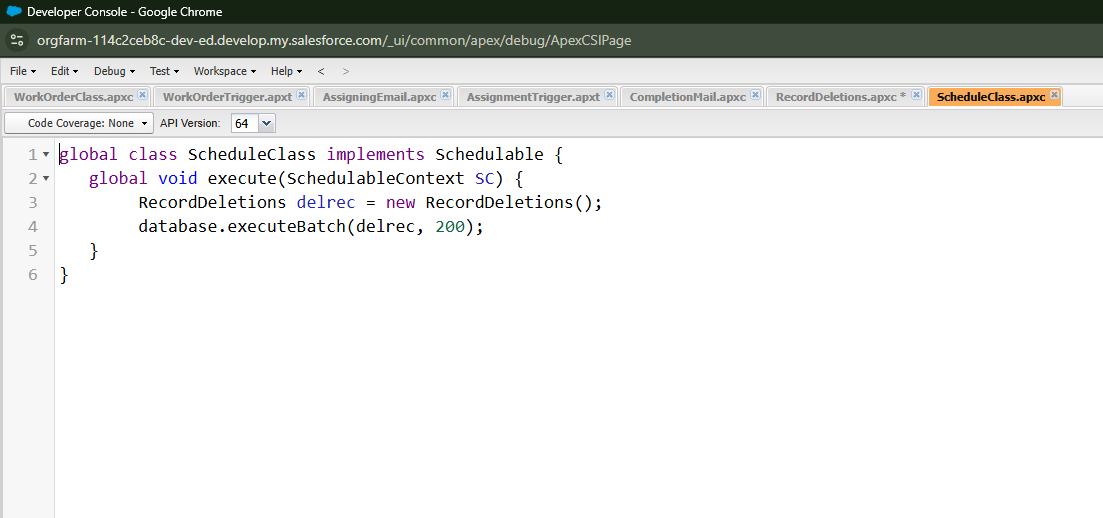
The project incorporates five Apex classes designed to automate and streamline key processes:

1. WorkOrderClass – Handles work order-related logic and technician assignment.
2. AssigningEmail – Sends email notifications upon technician assignment.
3. CompletionMail – Sends completion alerts to relevant users once a work order is resolved.
4. RecordDeletions – Performs cleanup of outdated or unnecessary records.
5. ScheduleClass – Automates the execution of record cleanup through scheduled jobs.

**These classes work together to support automation, ensure timely communication, and maintain data hygiene within the system.**

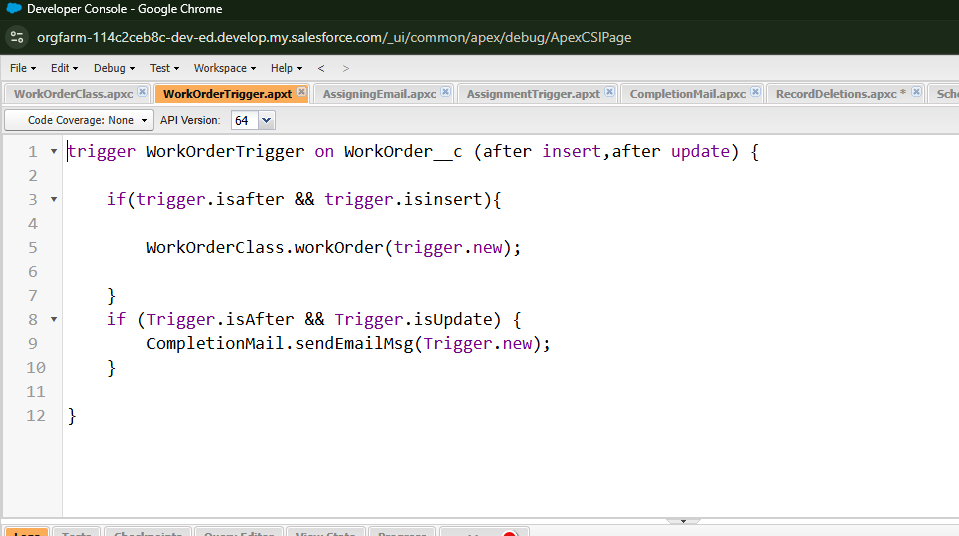


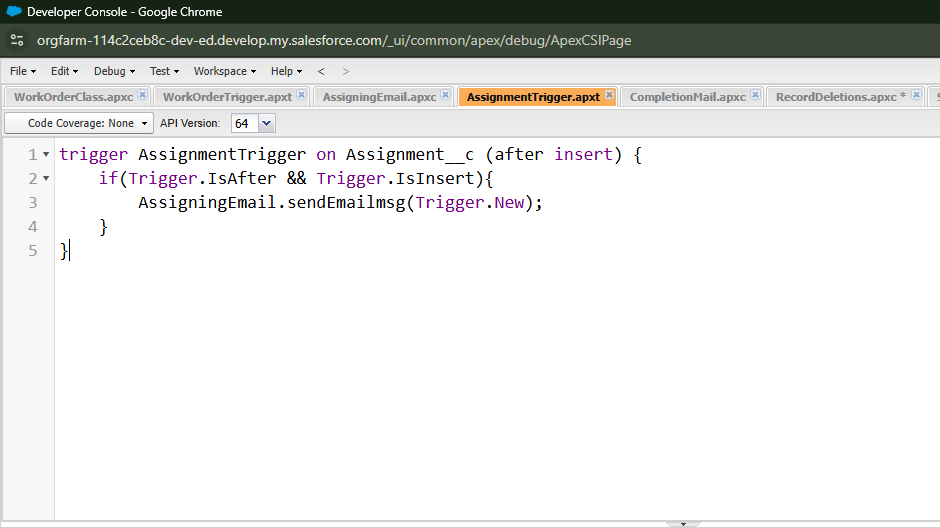




**Apex Triggers Implemented**

Two Apex triggers were developed to enhance automation in the Field Service Work Order process:

1. WorkOrderTrigger
   * Executes after insert and after update events.
   * Responsible for assigning technicians to work orders and sending resolution notifications to relevant stakeholders.
2. AssignmentTrigger
   * Executes after insert event.
   * Sends email notifications to the assigned technician, ensuring timely communication and response.



**Validation Rules**

1. Technician Email Format  
   Ensures that the technician's email address follows a standard format.  
   **Formula:**

NOT(

REGEX(Email\_\_c, "^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,4}$")

)

1. Work Order Required Fields  
   Validates that key fields in a Work Order are not left blank before submission.  
   **Formula:**

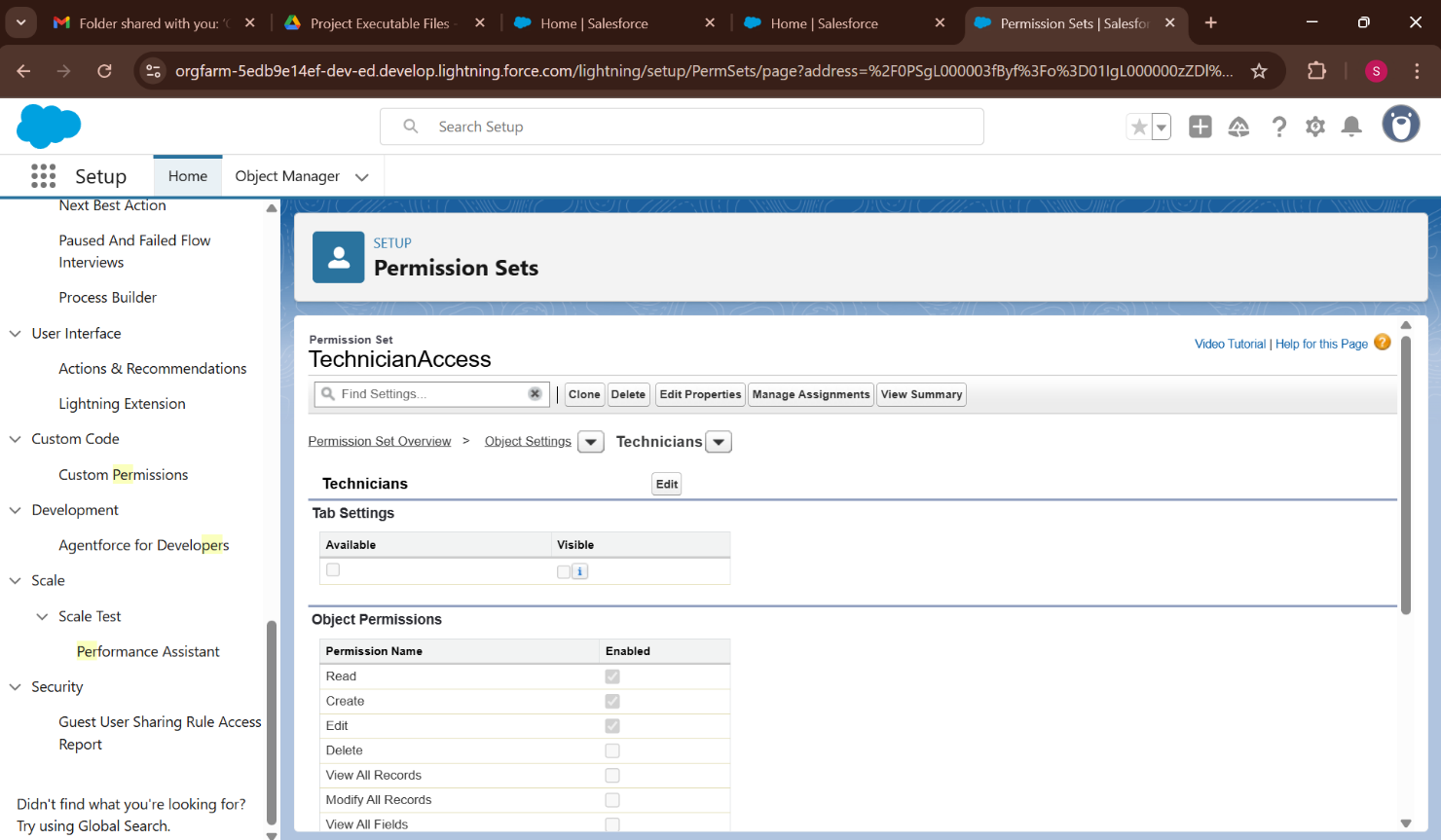
OR(

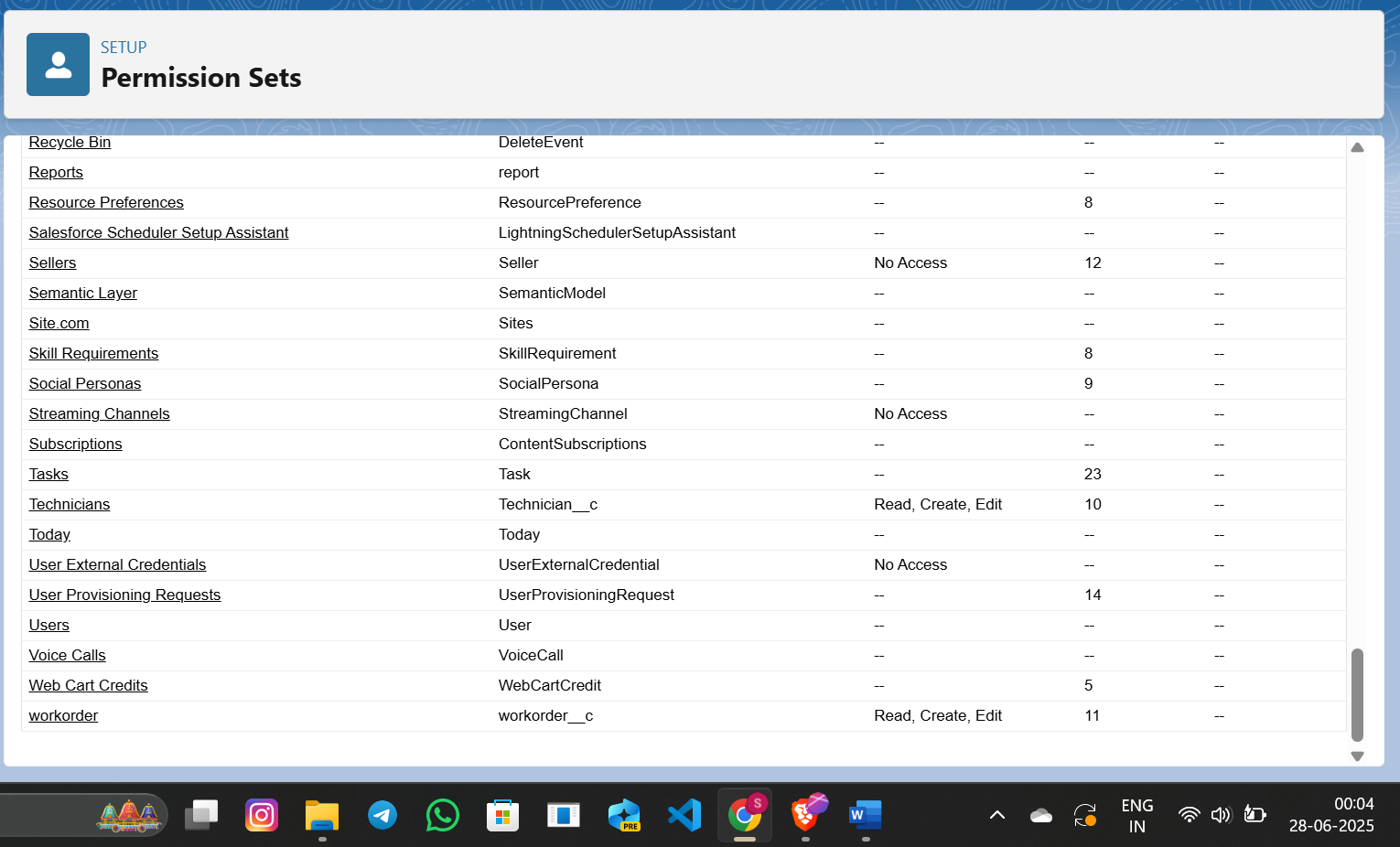
ISBLANK(Status\_\_c),

ISBLANK(Location\_\_c),

ISBLANK(Service\_Type\_\_c)

)

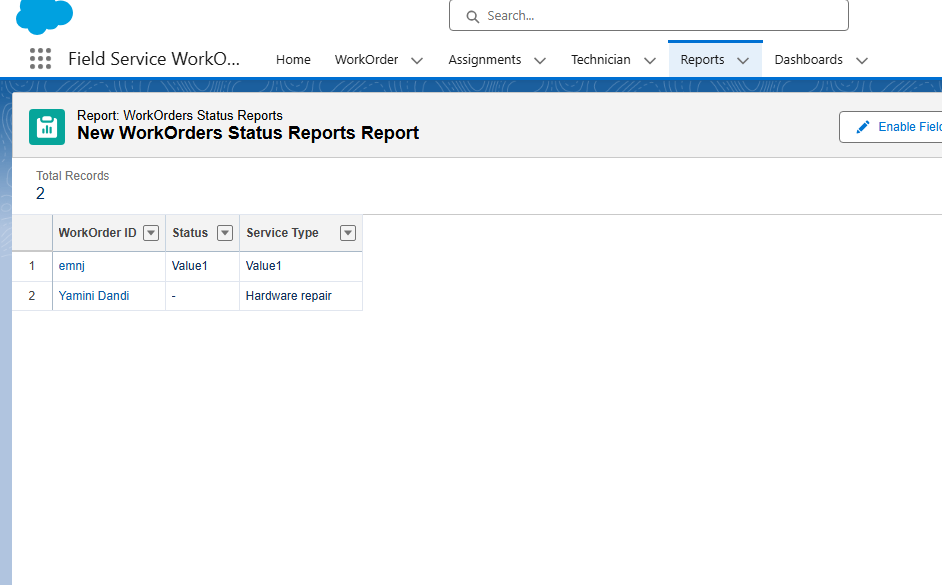


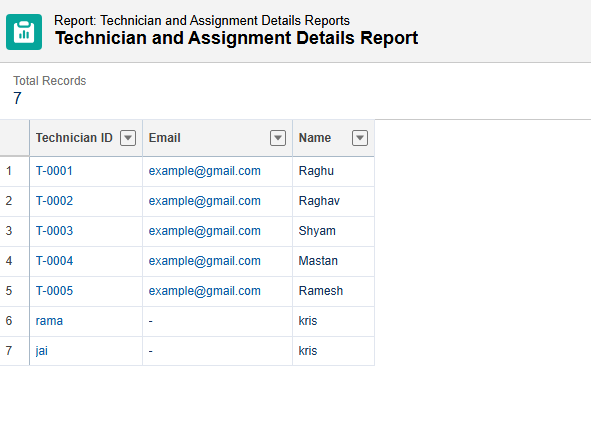


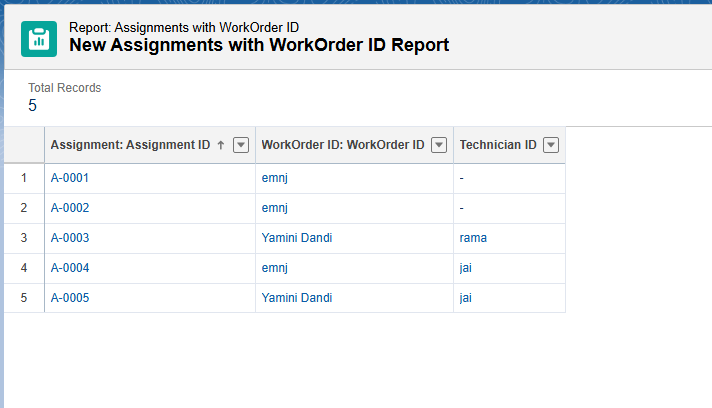
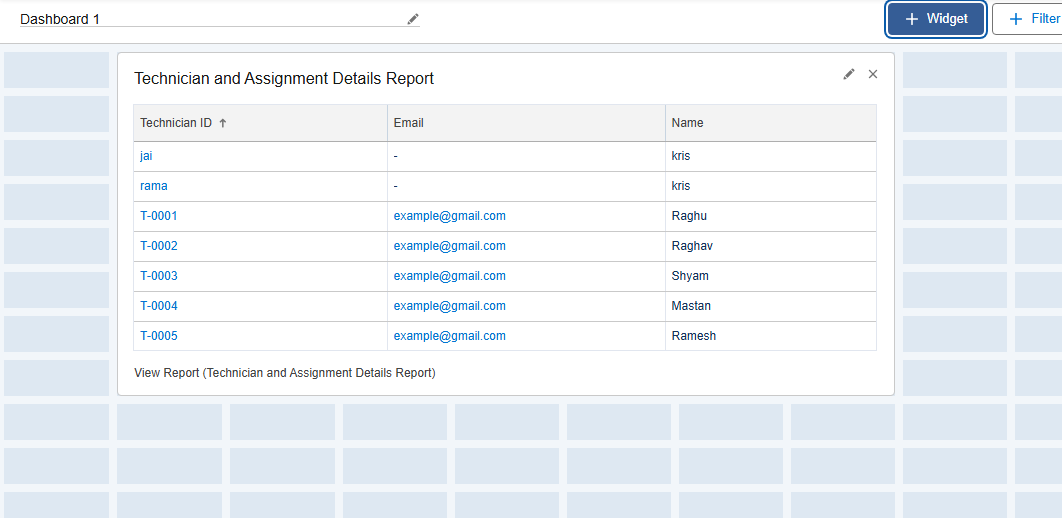
**Reports**

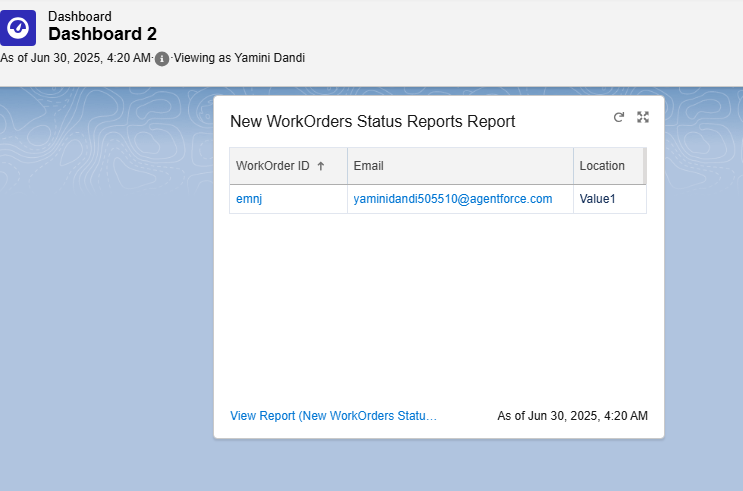
To support effective monitoring of work orders and technician activities, the following reports were created and stored under **Private Reports**:

1. **New WorkOrders Status Reports Report**
   * Displays the status of newly created work orders.
   * Helps track progress across different stages like Open, In Progress, and Completed.
2. **Technician and Assignment Details Report**
   * Provides a comprehensive overview of technician assignments.
   * Useful for analyzing workloads, assignments per technician, and service distribution.
3. **New Assignments with WorkOrder ID Report**
   * Shows newly assigned work orders along with their unique WorkOrder IDs.
   * Enables easy identification, tracking, and follow-up on individual assignments.





*  Created Dashboards



**PROJECT EXECUTABLE FILES**

* In the development of the **Field Service Work Order Optimization System**, Salesforce’s low-code capabilities were extended using **Apex programming** to enforce custom business rules and automation logic.  
  These executable components form the backend logic that ensures **data integrity** and **automates field service operations**.
* The following components were developed as part of the executable logic:
* **Apex Class**: Contains reusable methods that apply custom validation rules on the **WorkOrder** and **Assignment** objects.
* **Apex Trigger**: Automatically invokes the validation logic before a **WorkOrder** or **Assignment** is inserted or updated, ensuring critical fields like technician assignment and status are correctly populated.
* **Apex Test Class**: Performs unit testing of the logic to confirm that the triggers and methods function as intended, achieving over **75% code coverage** to meet Salesforce deployment requirements.

1. **WorkOrderClass – Assigns a technician to a work order**

public class WorkOrderClass {

public static void assignTechnician(Id workOrderId, Id technicianId) {

Assignment\_\_c assignment = new Assignment\_\_c(

WorkOrder\_\_c = workOrderId,

Technician\_\_c = technicianId,

Assignment\_Date\_\_c = Date.today(),

Status\_\_c = 'Assigned'

);

insert assignment;

}

}

1. **AssigningEmail – Sends email on assignment apex**

public class AssigningEmail {

public static void sendAssignmentEmail(Id technicianId, Id assignmentId) {

Technician\_\_c tech = [SELECT Name, Email\_\_c FROM Technician\_\_c WHERE Id = :technicianId LIMIT 1];

Assignment\_\_c assign = [SELECT Name FROM Assignment\_\_c WHERE Id = :assignmentId LIMIT 1];

Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();

mail.setToAddresses(new String[] { tech.Email\_\_c });

mail.setSubject('New Work Order Assigned');

mail.setPlainTextBody('Dear ' + tech.Name + ',\n\nYou have been assigned a new work order: ' + assign.Name + '.');

Messaging.sendEmail(new Messaging.SingleEmailMessage[] { mail });

}

}

**3.CompletionMail – Sends email upon work order completion**

public class CompletionMail {

public static void sendCompletionEmail(Id workOrderId) {

WorkOrder\_\_c wo = [SELECT Name, Status\_\_c FROM WorkOrder\_\_c WHERE Id = :workOrderId LIMIT 1];

if (wo.Status\_\_c == 'Completed') {

List<String> emails = new List<String>{'admin@example.com'}; // Replace with dynamic user list if needed

Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();

mail.setToAddresses(emails);

mail.setSubject('Work Order Completed: ' + wo.Name);

mail.setPlainTextBody('The work order "' + wo.Name + '" has been successfully completed.');

Messaging.sendEmail(new Messaging.SingleEmailMessage[] { mail });

}

}

}

1. **RecordDeletions – Deletes outdated or unused records**

public class RecordDeletions {

public static void cleanUpOldAssignments(Integer daysOld) {

Date thresholdDate = Date.today().addDays(-daysOld);

List<Assignment\_\_c> oldAssignments = [

SELECT Id FROM Assignment\_\_c

WHERE Assignment\_Date\_\_c < :thresholdDate AND Status\_\_c = 'Completed'

];

if (!oldAssignments.isEmpty()) {

delete oldAssignments;

}

}

}

1. ScheduleClass – Schedules the cleanup job

public class ScheduleClass implements Schedulable {

public void execute(SchedulableContext sc) {

RecordDeletions.cleanUpOldAssignments(30); // Deletes assignments older than 30 days

}

public static void scheduleJob() {

String cronExp = '0 0 2 ? \* SUN'; // Every Sunday at 2 AM

System.schedule('Weekly Assignment Cleanup', cronExp, new ScheduleClass());

}

}

1. **WorkOrderTrigger – Validates WorkOrder data before insert/update**

trigger WorkOrderTrigger on WorkOrder\_\_c (before insert, before update) {

for (WorkOrder\_\_c wo : Trigger.new) {

WorkOrderValidation.validateWorkOrder(wo);

}

if (Trigger.isUpdate) {

for (WorkOrder\_\_c wo : Trigger.new) {

if (wo.Status\_\_c == 'Completed') {

CompletionMail.sendCompletionEmail(wo.Id);

}

}

}

}

1. AssignmentTrigger – Sends email on new assignment

trigger AssignmentTrigger on Assignment\_\_c (after insert) {

for (Assignment\_\_c assign : Trigger.new) {

if (assign.Technician\_\_c != null) {

AssigningEmail.sendAssignmentEmail(assign.Technician\_\_c, assign.Id);

}

}

}

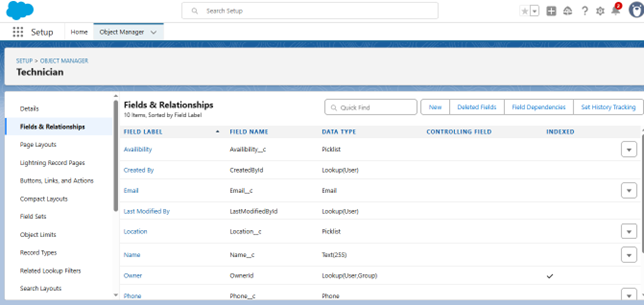
**DATASET**

**Custom Objects**

**1. Technician**

The **Technician** object stores detailed information about field technicians. It includes personal details, availability, skill sets, and locations. This data is essential for assigning the right technician to each work order efficiently.

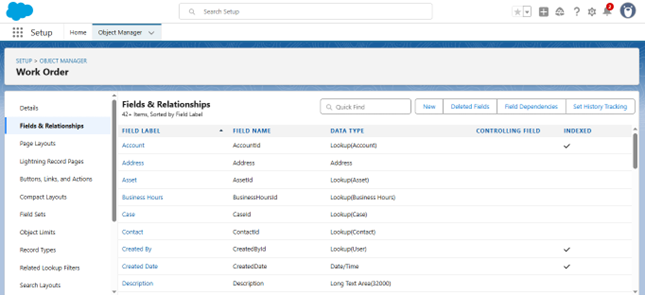
**Fields Used:**

* **Technician ID** *(Text)*
* **Name** *(Text)*
* **Email** *(Email)*
* **Skills** *(Picklist: Hardware, Software, Network, etc.)*
* **Location** *(Picklist)*
* **Owner** *(Lookup)*
* **Last Modified By** *(Lookup)*
* **Availability** *(Picklist: Available, Unavailable)*

**2. WorkOrder**

The **WorkOrder** object represents service tasks that need to be fulfilled. It tracks all necessary job-related details such as location, status, customer contact, and service priority.

**Fields Used:**

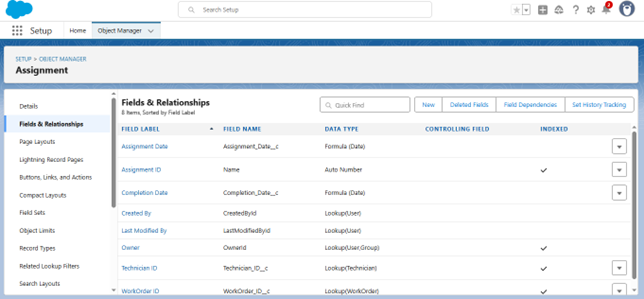
* **WorkOrder ID** *(Auto Number)*
* **Location** *(Picklist: Nasik, Warangal, Nanded)*
* **Email** *(Email)*
* **Priority** *(Picklist: High)*
* **Status** *(Picklist: Pending, In Progress, Resolved)*
* **Date** *(Date)*
* **Last Modified Date** *(Lookup)*

**3. Assignment**

The **Assignment** object links a technician to a specific work order. It is automatically generated through Apex logic and records technician assignments and completion details.

**Fields Used:**

* **Assignment ID** *(Auto Number)*
* **Technician ID** *(Lookup to Technician)*
* *(You may also include fields like WorkOrder ID, Assignment Date, Completion Status, etc., if applicable)*

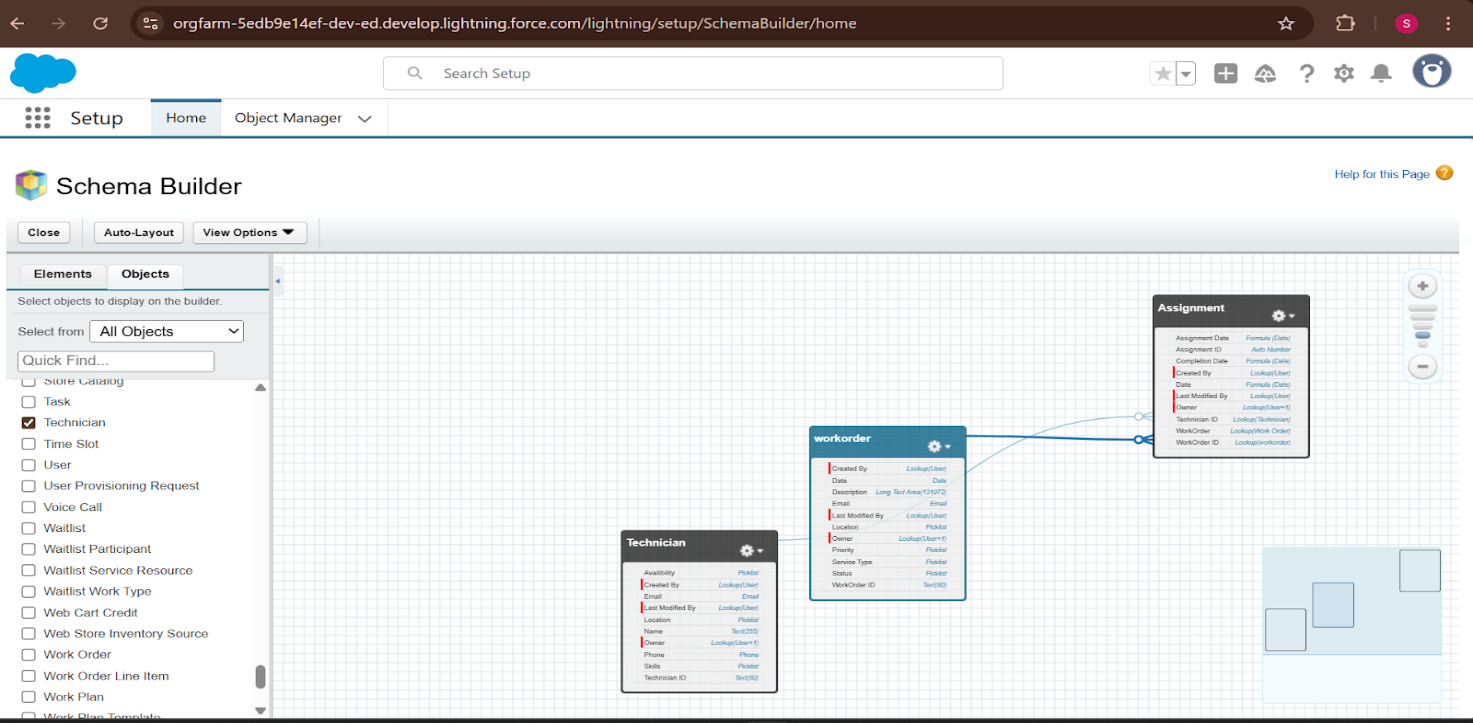


**Schema Builder**

The **Schema Builder** was used to visually design and manage the data model for the Field Service WorkOrder Optimization system in Salesforce. It provided a drag-and-drop interface to create and modify custom objects, fields, and relationships without needing code.

**Key Highlights:**

* **Visual Representation:**  
  All three custom objects — **Technician**, **WorkOrder**, and **Assignment** — were modeled in the Schema Builder to clearly define their structure and relationships.
* **Relationships Defined:**
  + **Lookup Relationship** between *Assignment* and *Technician*
  + **(Optional/If used)** Lookup or Master-Detail relationship between *Assignment* and *WorkOrder*
* **Field Configuration:**  
  Custom fields such as picklists, auto numbers, and lookups were added directly through the builder, ensuring proper data integrity and validation.
* **Efficient Object Management:**  
  Schema Builder made it easier to understand how objects interact, helping both developers and admins visualize the overall system architecture at a glance.



**8. ADVANTAGES & DISADVANTAGES**

**Advantages**

1. **Automated Technician Assignment**  
   Reduces manual effort and ensures the right technician is assigned based on availability and skills.
2. **Real-Time Notifications**  
   Immediate email alerts and status updates keep technicians and managers informed, improving response times.
3. **Custom Validation Rules**  
   Enforce data accuracy and prevent errors (e.g., invalid email formats or missing critical fields).
4. **Scalable Architecture**  
   Built using Salesforce tools (Apex, Flows, Dashboards) that can scale with future business needs.
5. **Interactive Dashboards**  
   Provide visual summaries of key metrics like work order completion rates and technician workload.
6. **Role-Based Access**  
   Ensures security by controlling who can view or edit records, using profiles and permission sets.

**Disadvantages**

1. **Limited Offline Capability**  
   Field technicians require an internet connection to access the Salesforce app unless mobile offline setup is implemented.
2. **Initial Setup Complexity**  
   Requires knowledge of Salesforce development tools (Apex, Flows) for customization.
3. **Maintenance Overhead**  
   Triggers, scheduled jobs, and automation may require regular updates or debugging during scale-up.

**9. CONCLUSION**

The **Field Service WorkOrder Optimization** project successfully demonstrates how Salesforce can be leveraged to **digitize, automate, and optimize** field service operations. With intelligent technician assignment, real-time updates, and comprehensive dashboards, the system significantly improves service delivery, operational transparency, and customer satisfaction.

This hands-on implementation gave interns practical exposure to Salesforce development, automation, and CRM best practices, while solving real-world service management problems.

The result is a scalable, efficient, and user-centric solution ready for deployment in any field service environment.

**10. FUTURE SCOPE**

The current system lays a solid foundation for field service automation, with room for future expansion and improvements:

* **Mobile App Integration**  
  Enable offline access and real-time mobile updates for technicians in the field.
* **Geo-Mapping & GPS Tracking**  
  Assign technicians based on real-time location for faster service dispatch.
* **Feedback and Rating Module**  
  Allow customers to rate service quality and provide feedback post-completion.
* **Chatbot Support**  
  Use Einstein Bots or third-party integrations for handling basic queries or updates.
* **Advanced SLA Monitoring**  
  Integrate SLA timers and escalation alerts for high-priority or overdue work orders.
  + Technician email format, required fields for WorkOrder creation
* **Reports & Dashboards:**
  + WorkOrder Status, Technician Assignment Summary
  + Dashboard showing Completed vs Pending WorkOrders

**📌 Project Tools Used**

* **Platform:** Salesforce Developer Org
* **Collaboration:** Google Docs, Zoom, Slack
* **Tracking:** Jira / Trello (if used)
* **Testing:** Salesforce Test Framework + Manual Testing
* **Version Control:** Git (if applicable)

**GitHub Link:**

**Demo Video Link:**