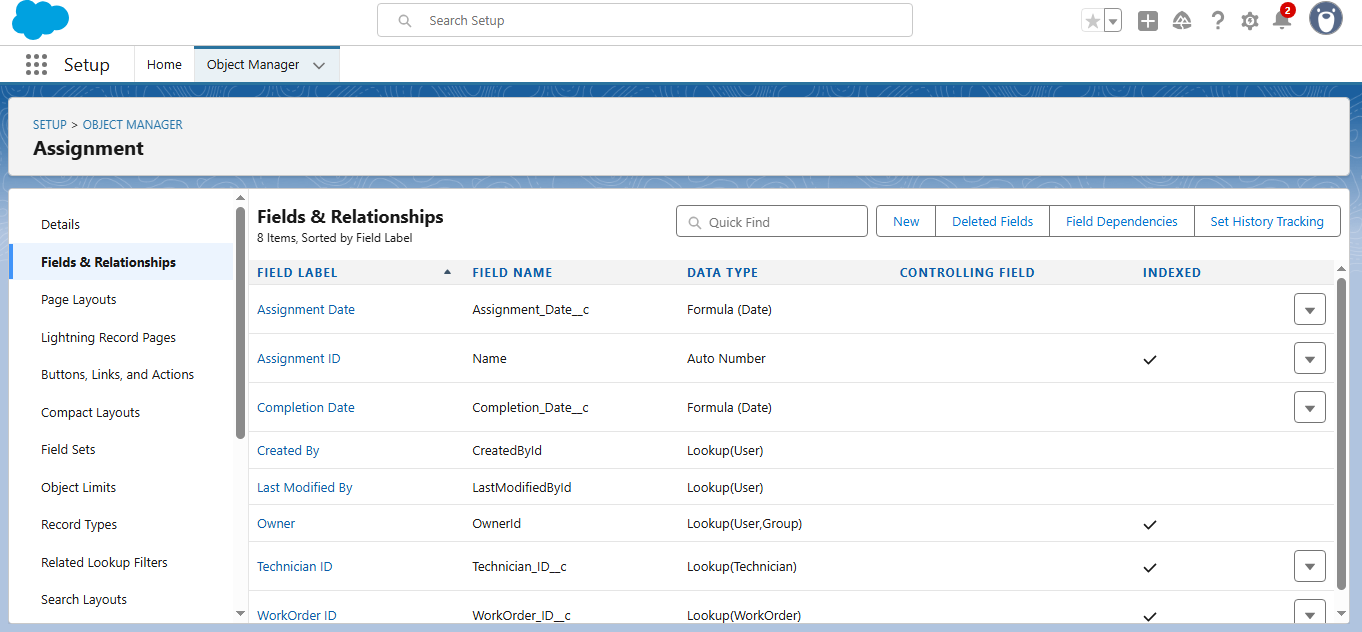
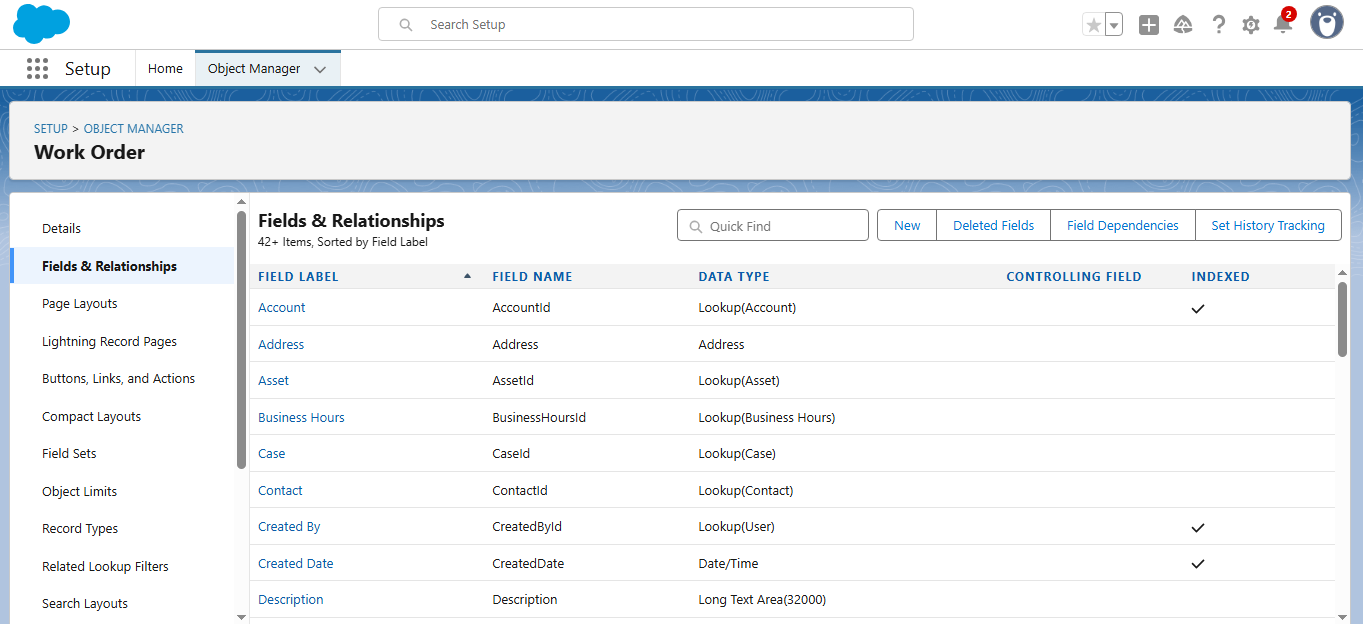
# PROJECT DEVELOPMENT PHASE

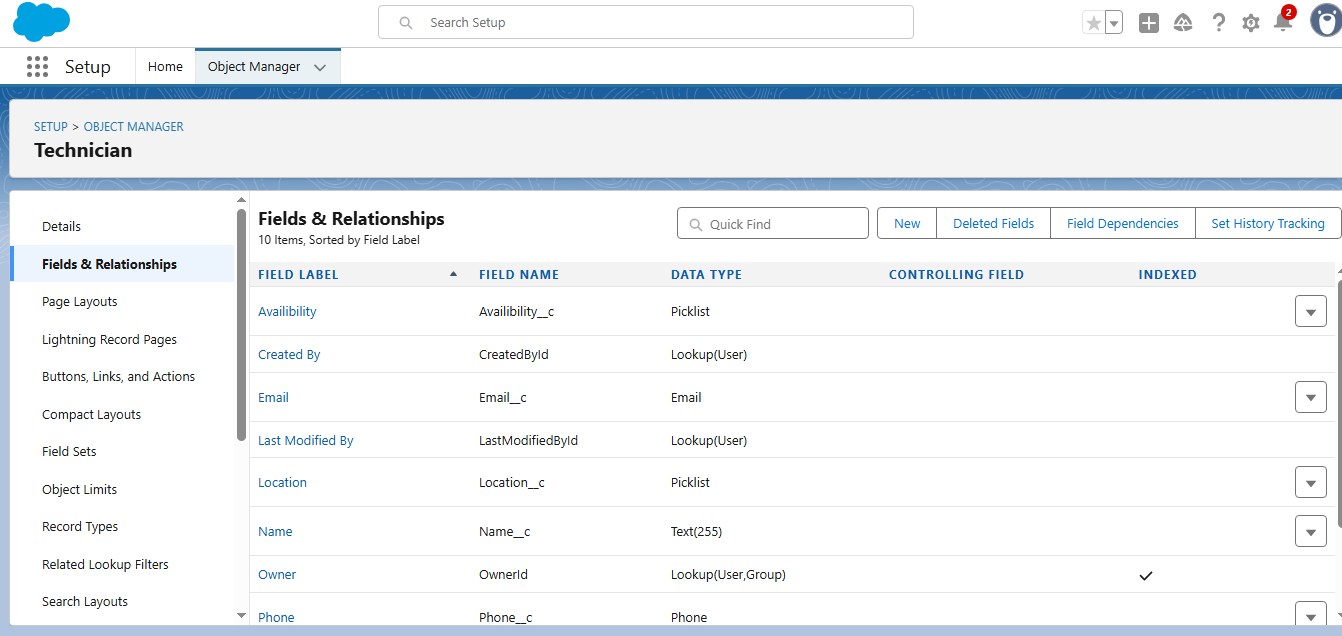
|  |  |
| --- | --- |
| **Date** | **28-06-2025** |
| **Team Id** | **LTVIP2025TMID31548** |
| **Project Name** | **field service workorder optimization** |
| **College Name** | **Ideal Institute Of Technology** |

## **.Created developer org and explored platform features**

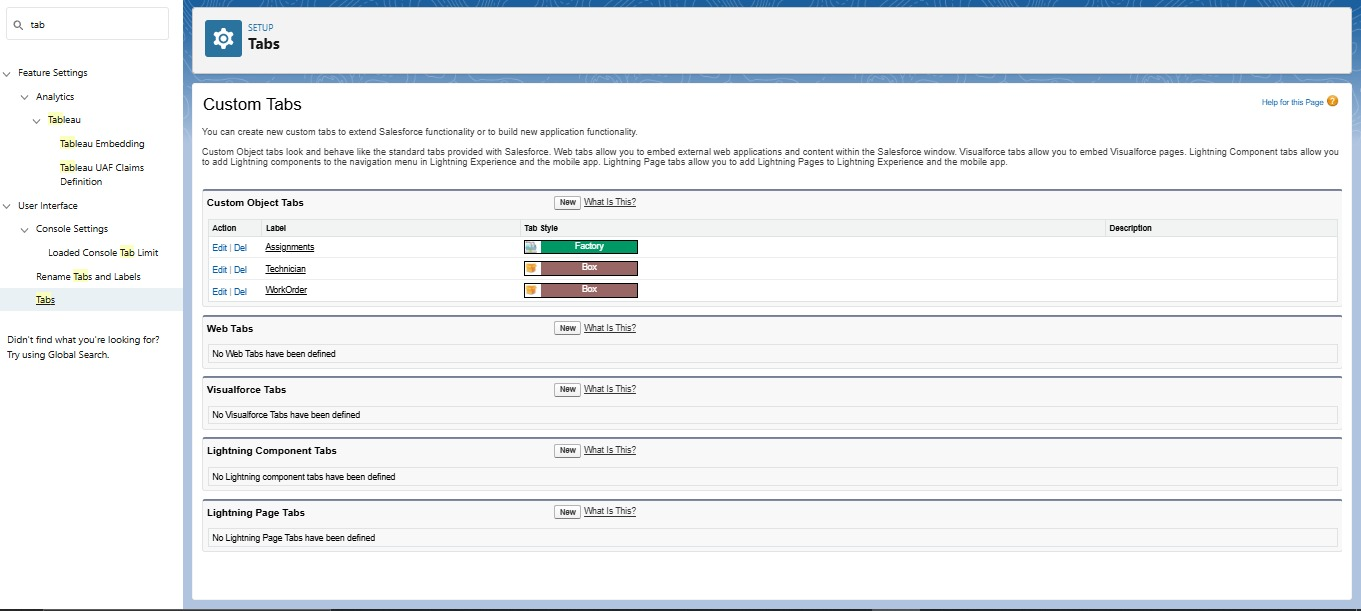
* **Created three custom objects: Technician (stores technician details), WorkOrder (captures service task info), and Assignment (links Technician to WorkOrder using relationships), with Assignment having lookup or master-detail fields to both Technician and WorkOrder.**



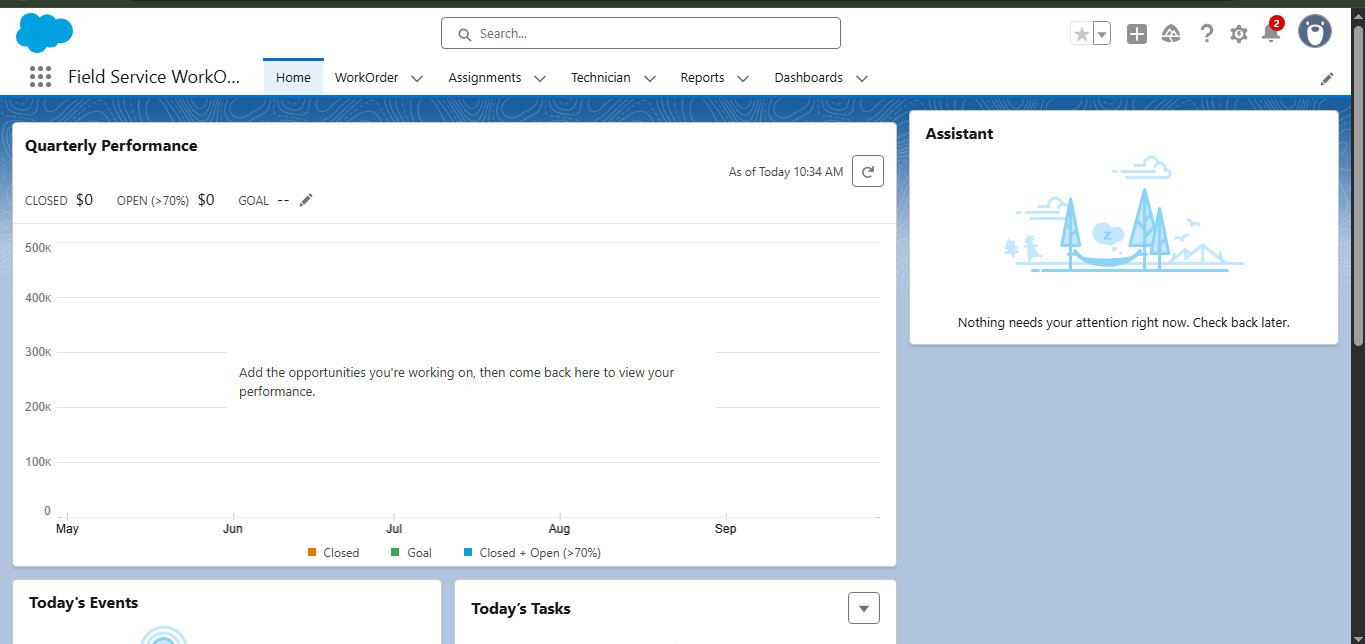


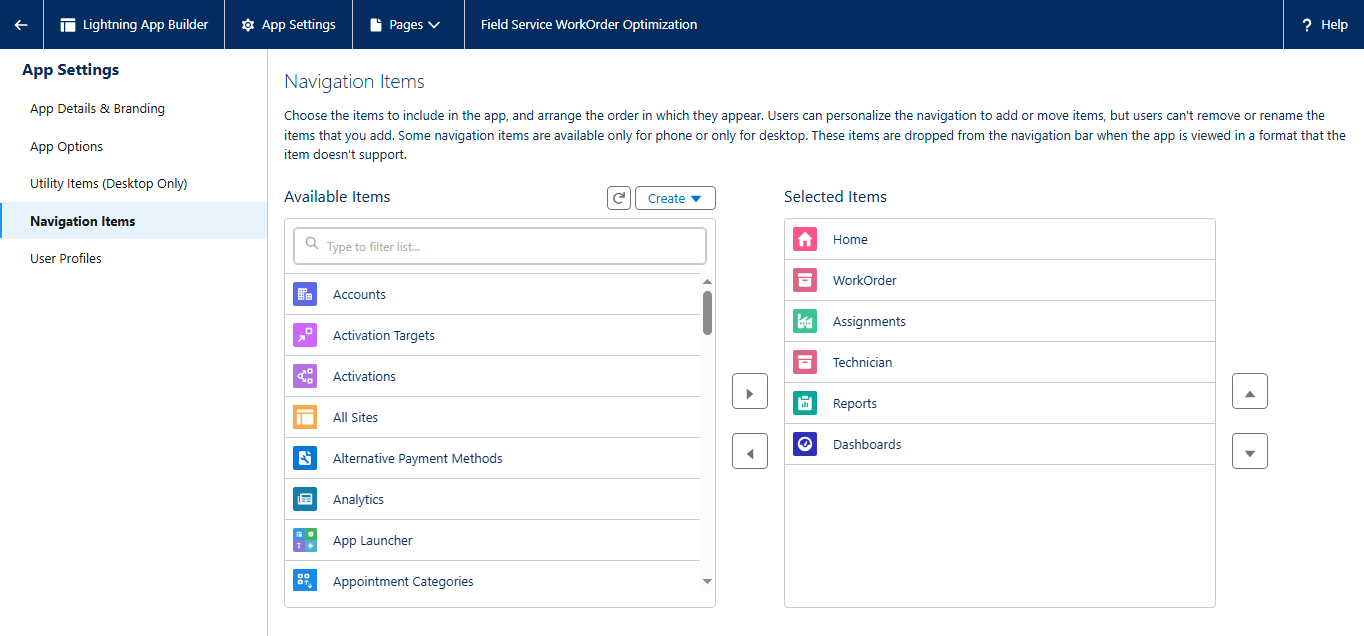


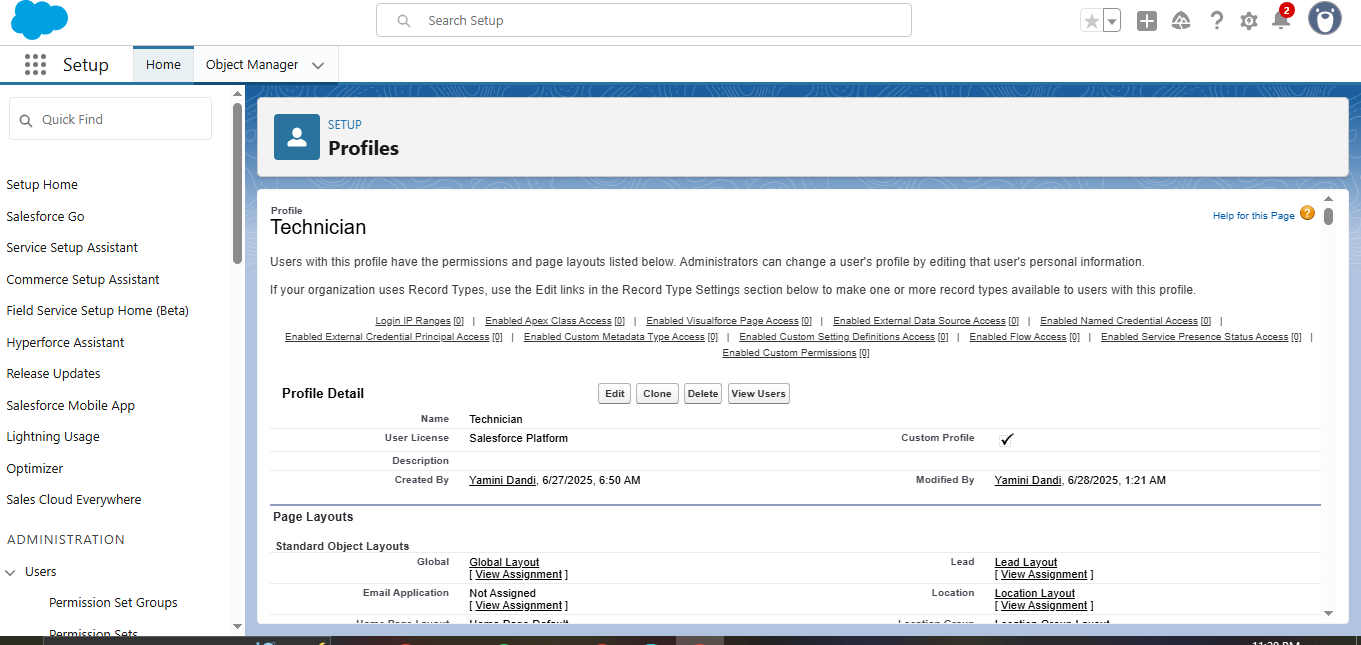
* **Created core objects and tabs for Technician, WorkOrder, Assignment, and ServiceTerritory.**

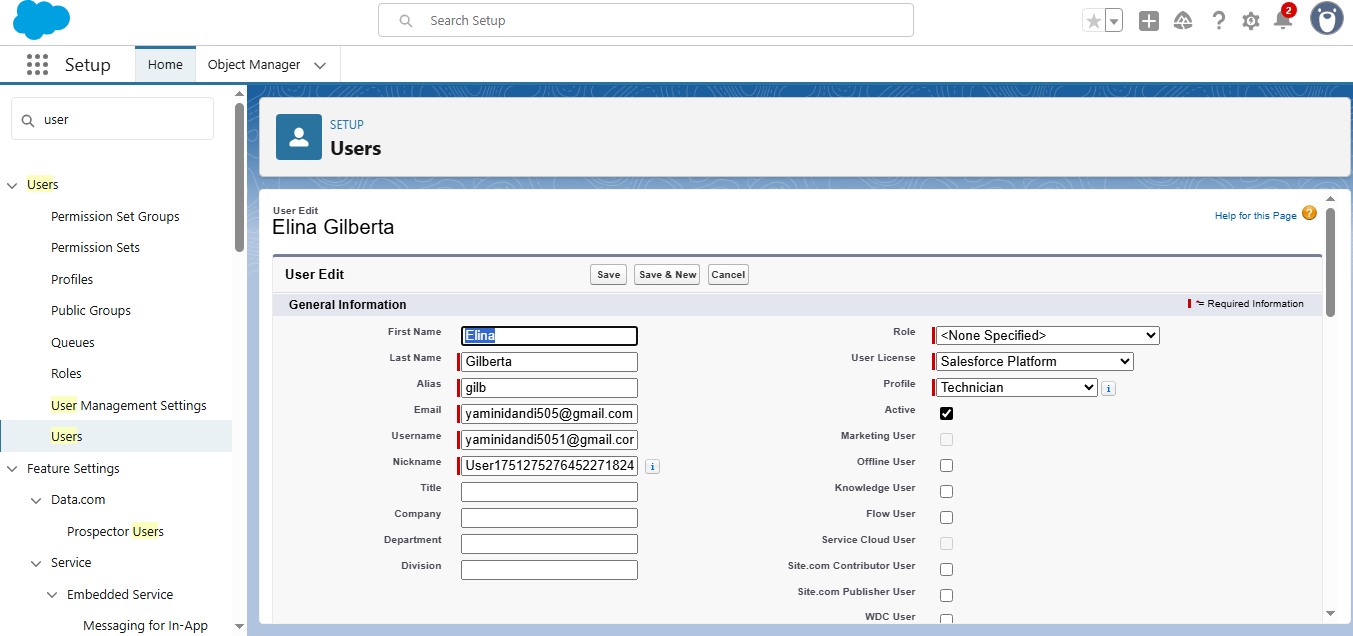


* **Developed a Lightning App**





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

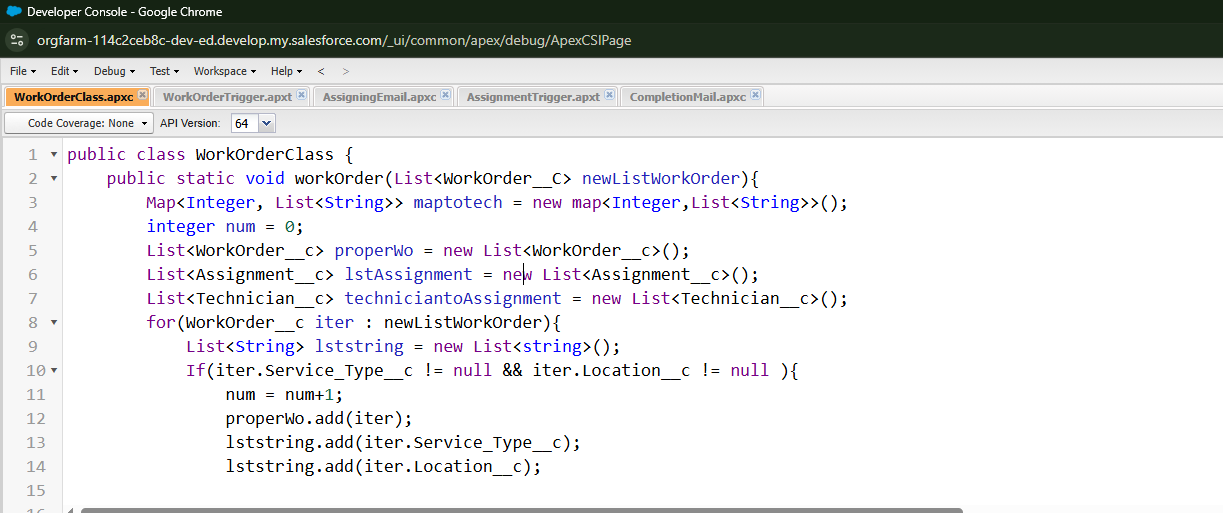


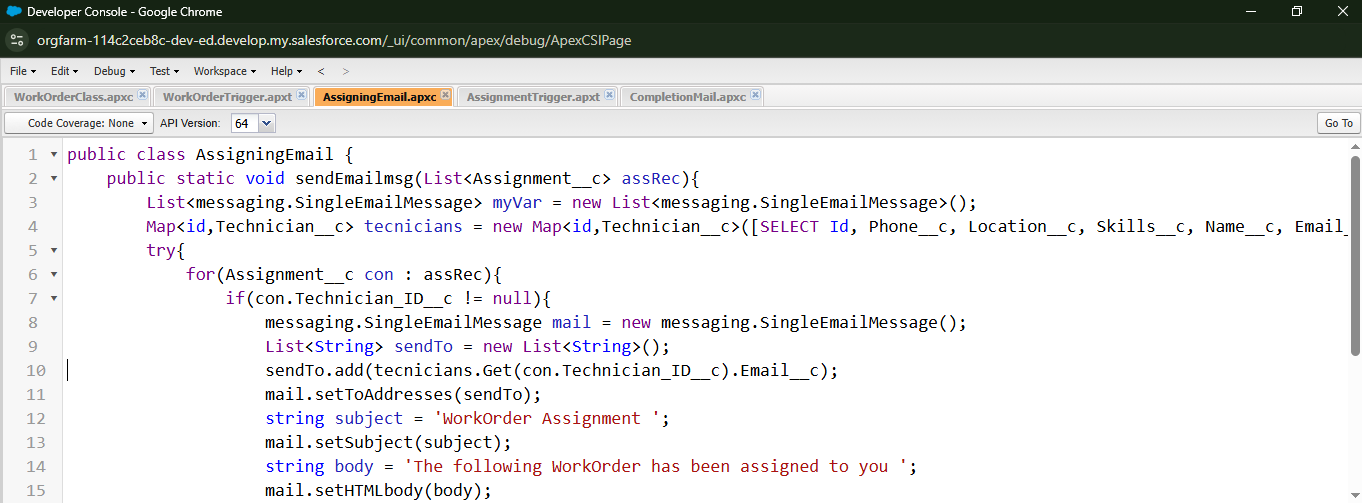
**Apex Code: Apex Classes**

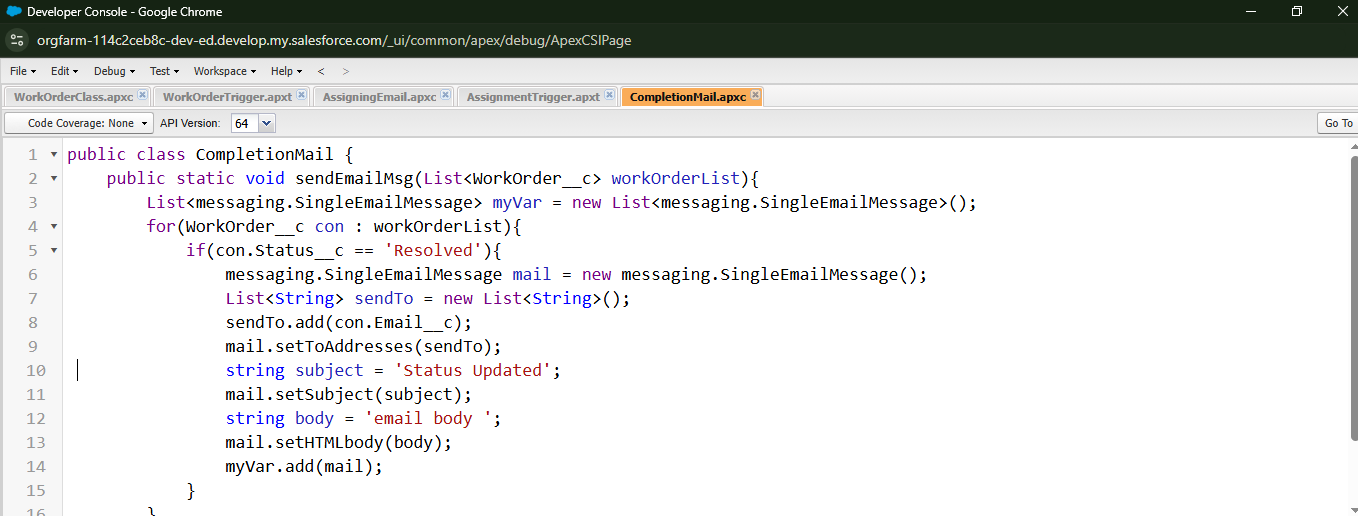
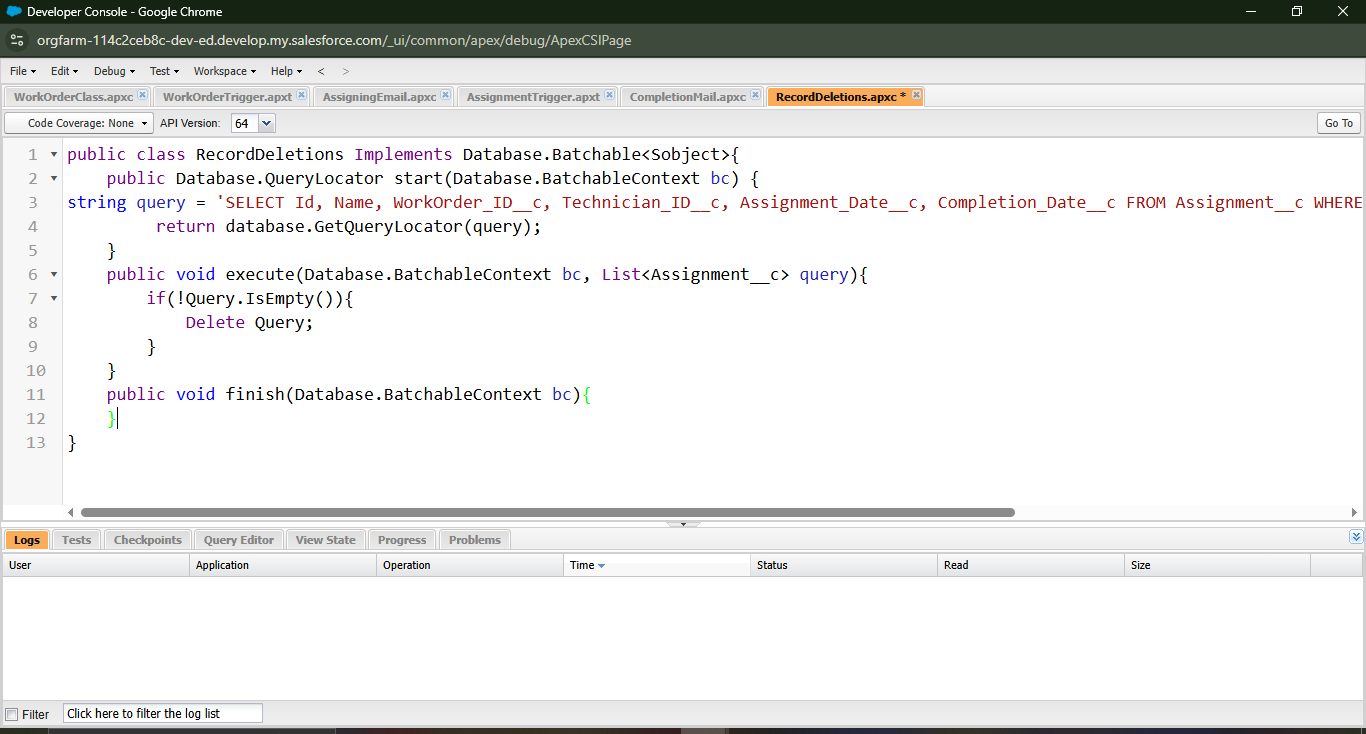
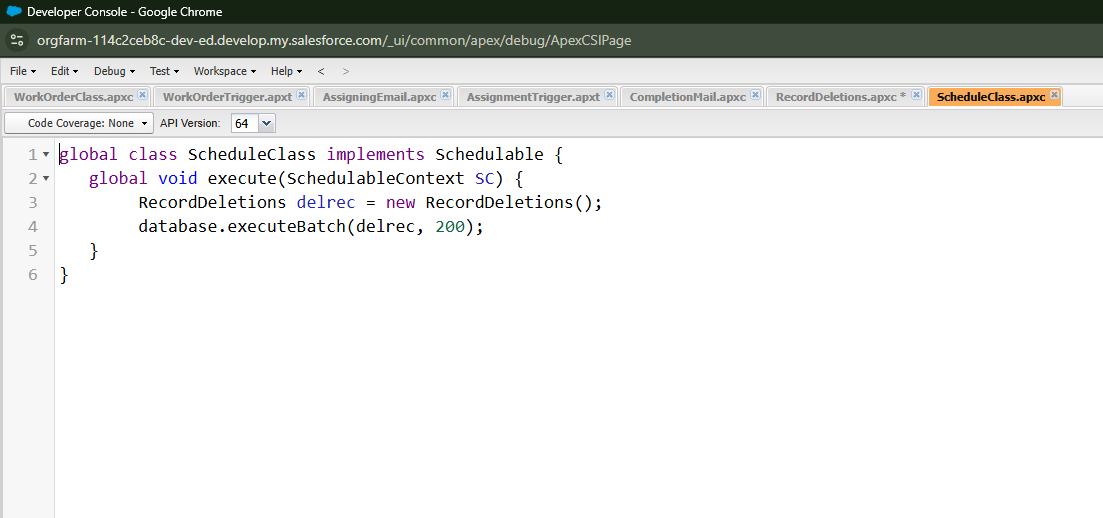
The project incorporates five Apex classes designed to automate and streamline key processes:

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.**

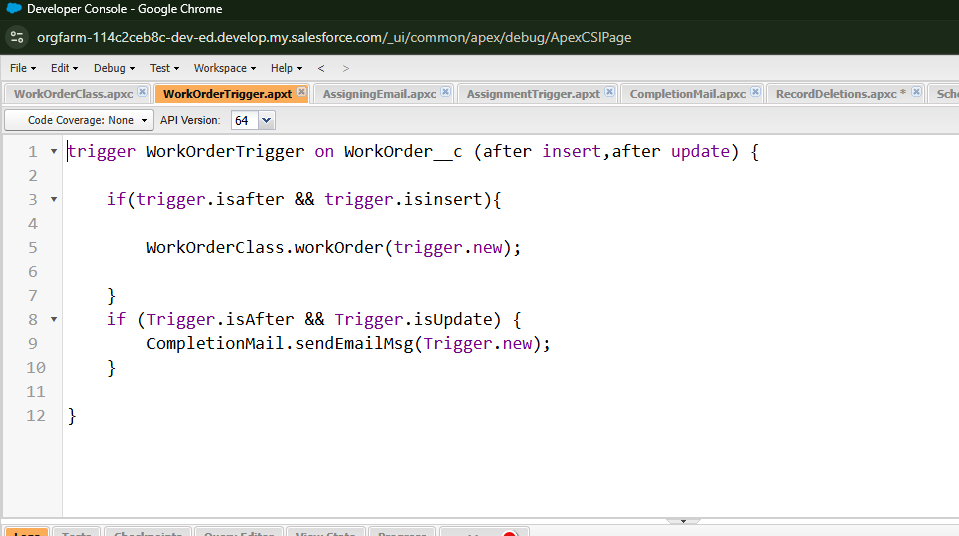


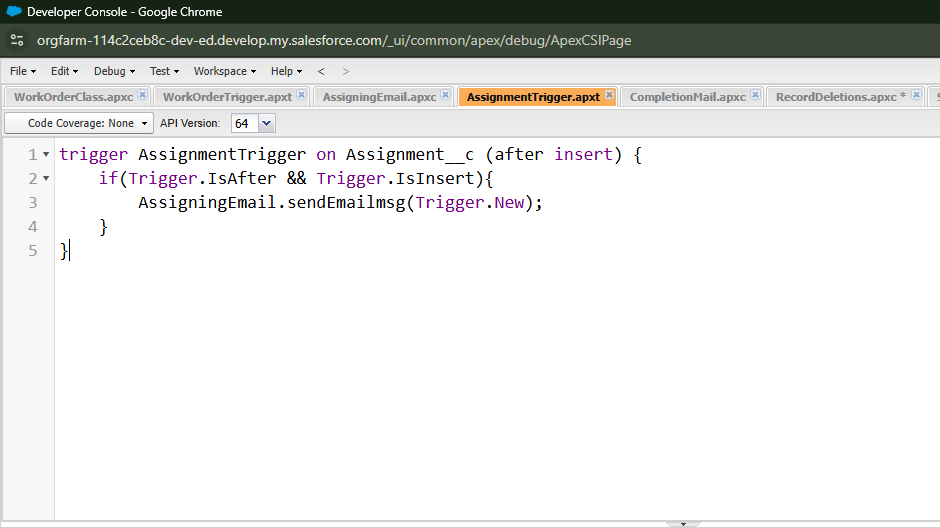




**Apex Triggers Implemented**

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:**

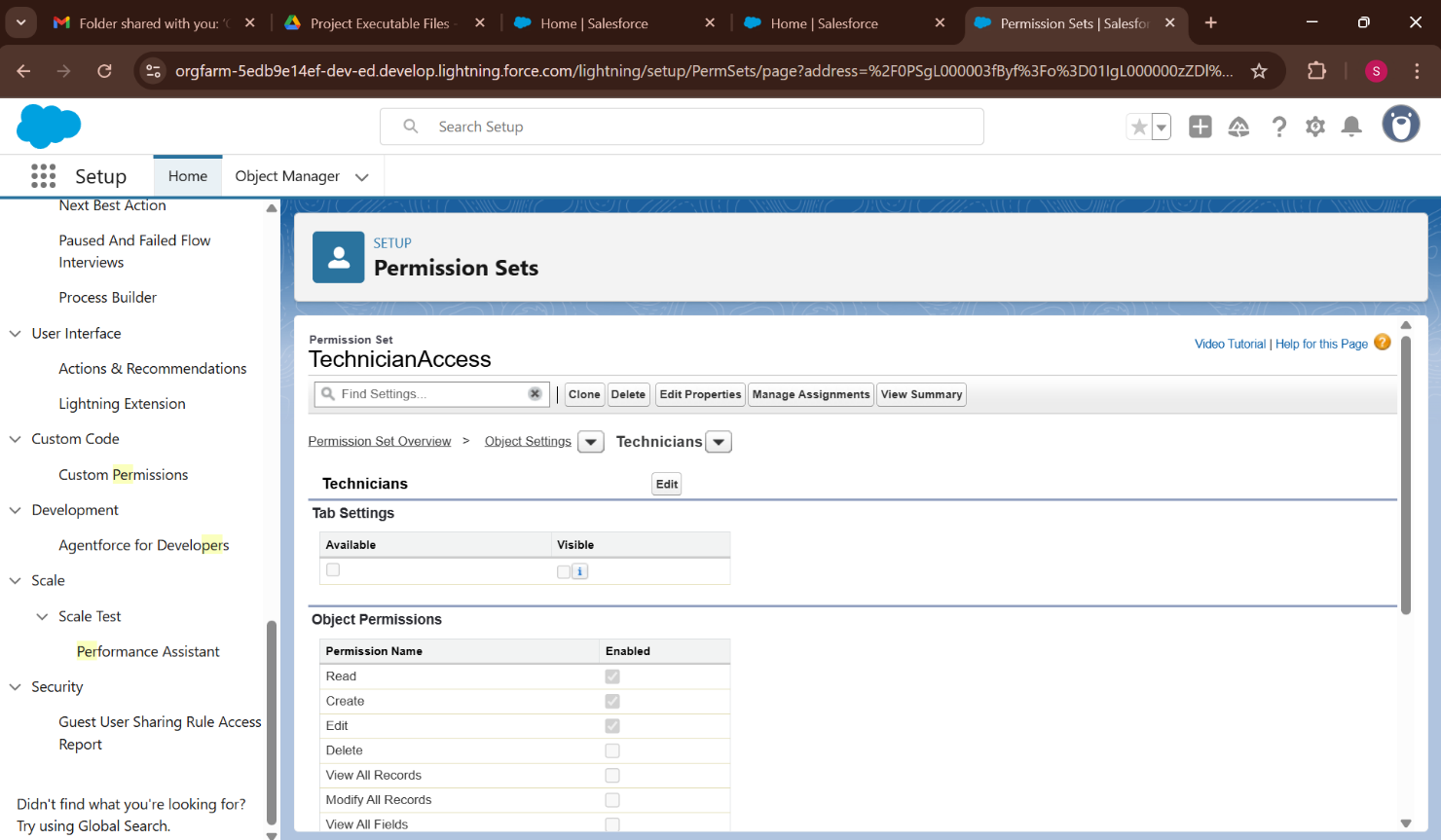
OR(

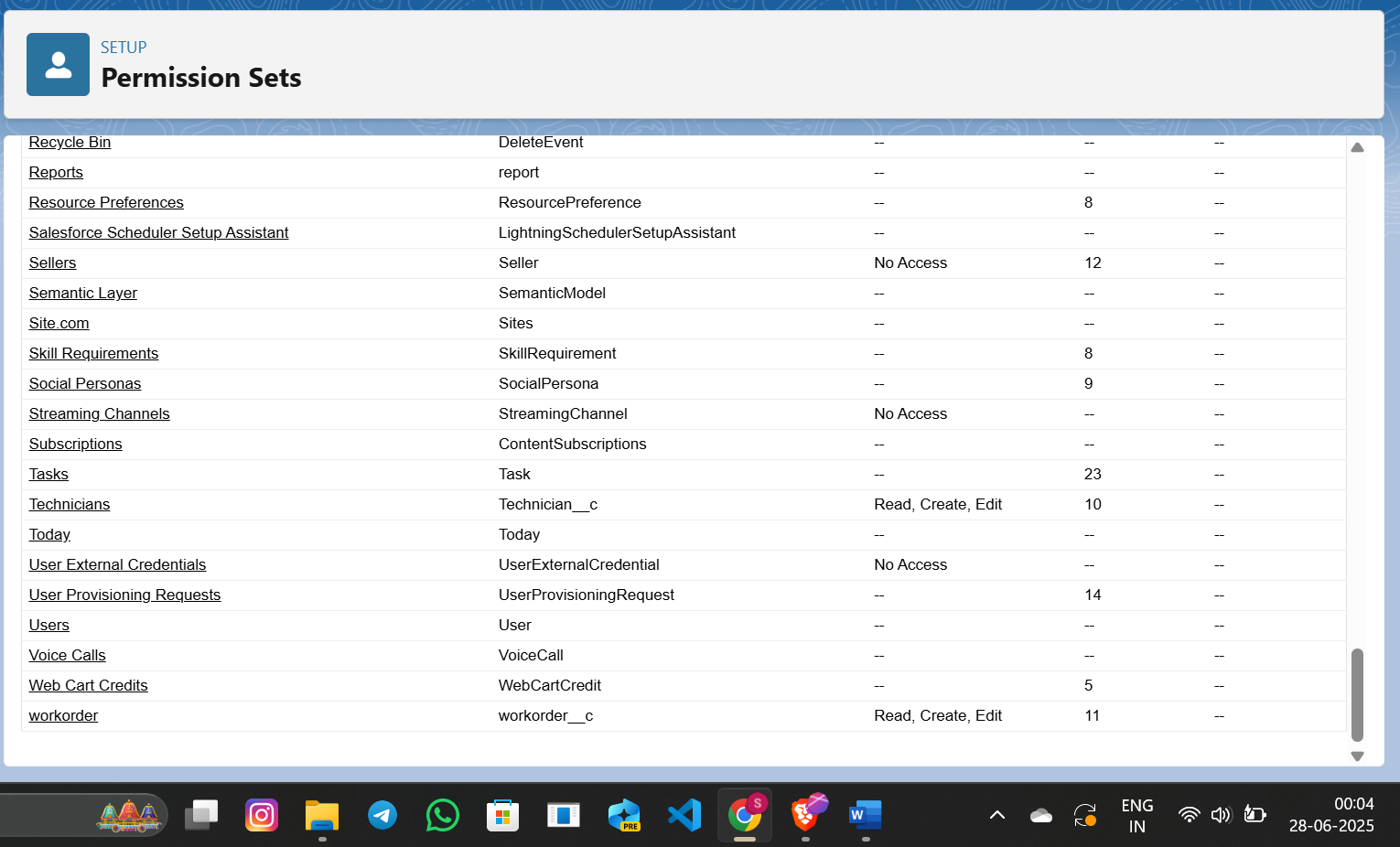
ISBLANK(Status\_\_c),

ISBLANK(Location\_\_c),

ISBLANK(Service\_Type\_\_c)

)

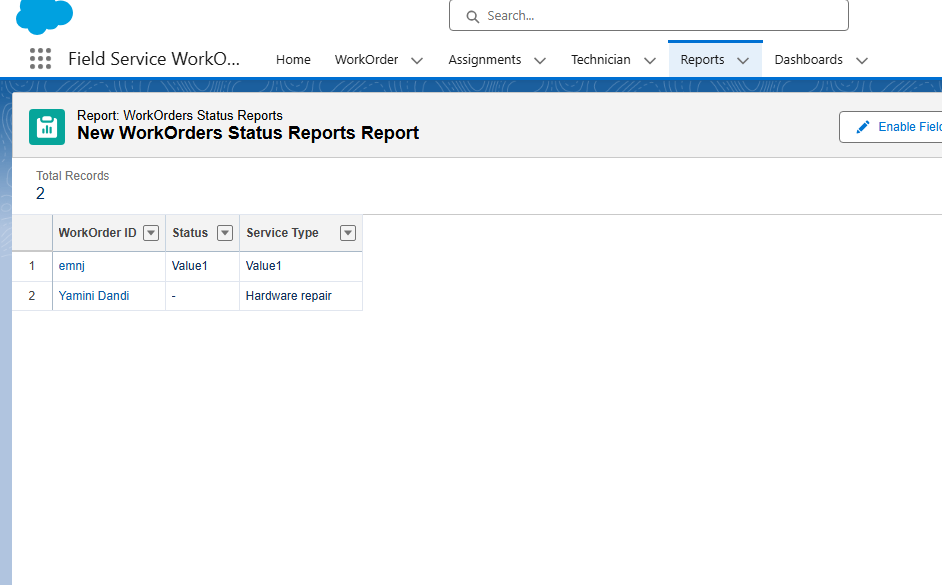


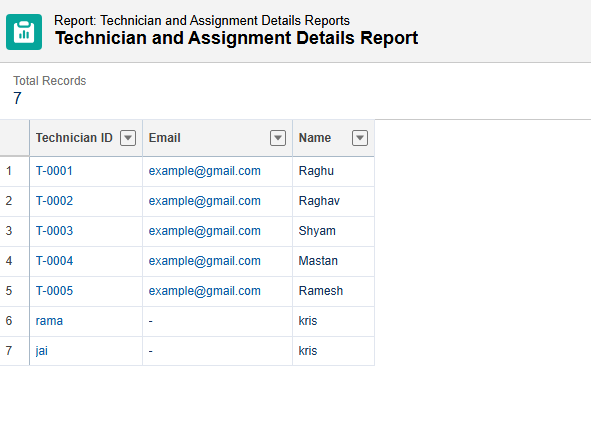


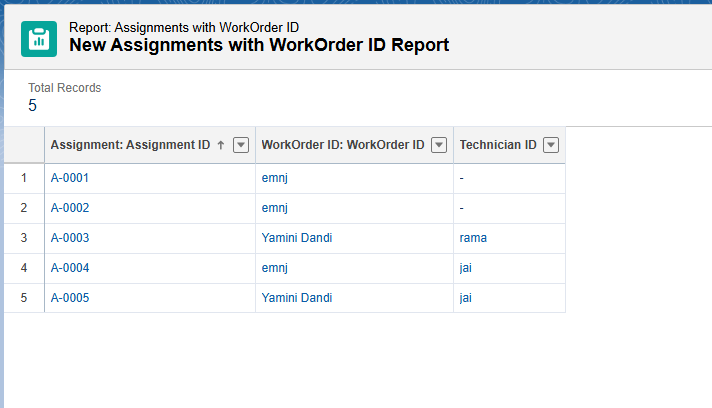
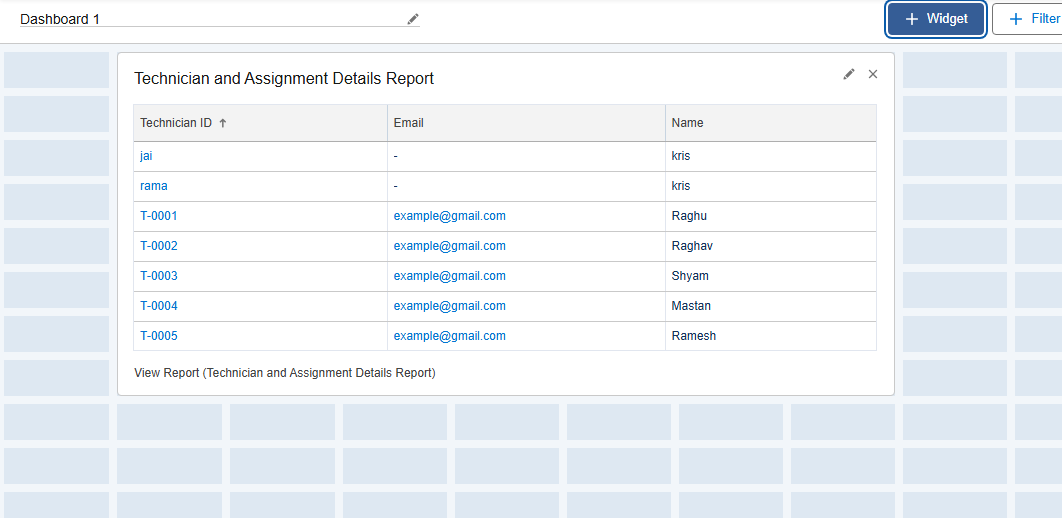
**Reports**

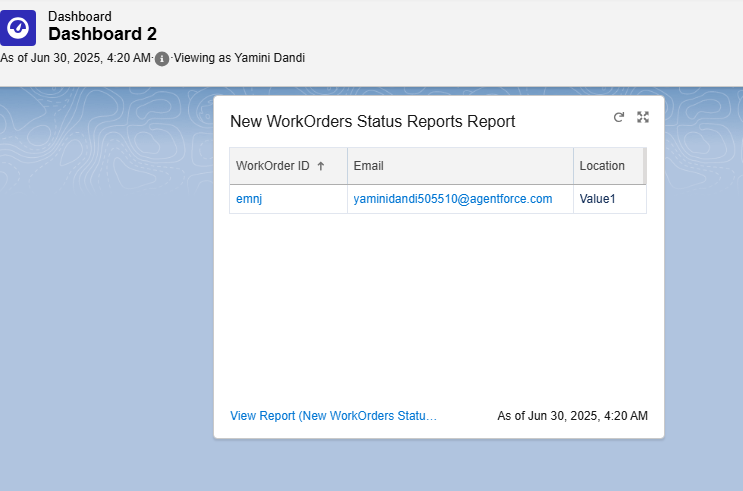
To support effective monitoring of work orders and technician activities, the following reports were created and stored under **Private Reports**:

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);

}

}

}

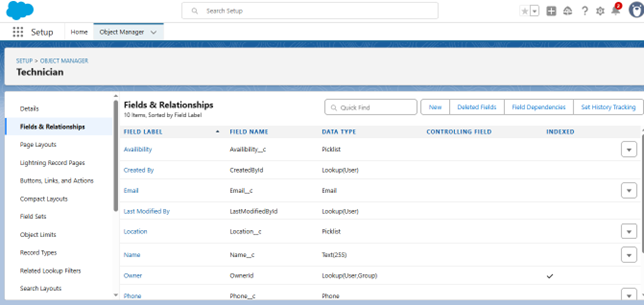
**DATASET**

**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.

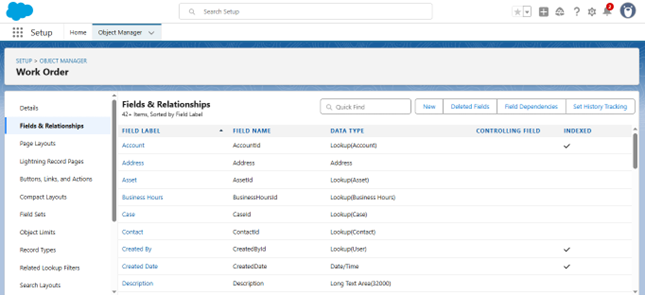
**Fields Used:**

* **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:**

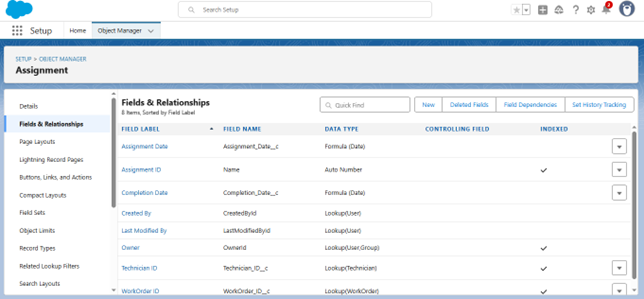
* **WorkOrder ID** *(Auto Number)*
* **Location** *(Picklist: Nasik, Warangal, Nanded)*
* **Email** *(Email)*
* **Priority** *(Picklist: High)*
* **Status** *(Picklist: Pending, In Progress, Resolved)*
* **Date** *(Date)*
* **Last Modified Date** *(Lookup)*

**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.

