Dynamic Route Optimization for Last-Mile Delivery in Salesforce CRM

Problem Statement

In TCS's Last-Mile operations, current route planning methods are insufficiently adaptive and often static. They fail to account for real-world conditions such as traffic fluctuations, road closures, delivery time windows, vehicle constraints, and customer availability. This leads to sub-optimal routes, increased fuel/time costs, frequent delays, missed or failed deliveries, and poor visibility into deviations. The lack of dynamic response to real-time disruptions reduces operational efficiency, degrades customer satisfaction, and limits scalability.

Phase 1: Problem Understanding & Industry Analysis

Goal: Understand what we're building and why for Last Mile CRM.

1. Requirement Gathering

To develop a dynamic and adaptive route optimization system within Salesforce CRM, a detailed requirement study was conducted across multiple operational levels. The goal was to capture both functional and non-functional requirements that would drive intelligent routing decisions.

- Predict delivery ETAs with high accuracy using live traffic and GPS data.
- Plan and reoptimize routes dynamically based on real-time conditions.
- Assign deliveries intelligently to agents considering constraints like shift times, capacity, and location.
- Provide real-time tracking and status visibility for all stakeholders.
- Alert system for delays, deviations, or high-risk deliveries.
- Digital proof of delivery with photo/signature capture.
- Seamless integration with Salesforce modules, mapping APIs, and logistics systems.

2. Stakeholder Analysis

Each stakeholder plays a unique role in ensuring successful last-mile delivery. The system's design must address their distinct requirements and expectations.

- **Operations / Logistics Manager:** Seeks visibility on performance KPIs, operational costs, and delivery efficiency.
- **Dispatch / Routing Team:** Requires dashboards to plan, monitor, and modify routes dynamically when disruptions occur.
- **Field Agents / Drivers:** Need mobile interfaces for navigation, delivery confirmation, and status updates.
- **Customer Support:** Relies on accurate, real-time delivery status to resolve customer queries efficiently.
- **Customers:** Expect timely updates, accurate ETAs, and smooth delivery experiences.
- **Salesforce Admin / IT Team:** Ensures secure system setup, scalability, and integration with Salesforce ecosystems.

3. Business Process Mapping

The proposed process leverages Salesforce CRM to automate and optimize delivery workflows:

Flow: Order received → System predicts ETA & suggests optimal route → Dispatch assigns to

agent → Agent starts journey → Real-time tracking updates CRM → Route deviations trigger alerts → Delivery completed → Feedback recorded → Data stored for continuous improvement.

Exception Management: In case of delays, system recalculates routes, reassigns deliveries if needed, and sends automated notifications to customers.

4. Industry-Specific Use Case Analysis

In logistics, real-world conditions are unpredictable. Systems must adapt dynamically to maintain efficiency and reliability. This section outlines key challenges specific to the last-mile delivery sector.

- High traffic density in urban areas significantly affects delivery times.
- Frequent changes in customer availability require dynamic re-routing.
- Vehicle limitations (capacity, fuel range) influence delivery sequence planning.
- External factors like weather or road closures often disrupt delivery schedules.
- Customer satisfaction heavily depends on real-time updates and transparency.

5. App / Solution Landscape

An evaluation of Salesforce AppExchange solutions and third-party integrations was conducted to assess capabilities for dynamic route planning and optimization.

- **Existing Solutions:** Tools like Routific, Onfleet, and Locus provide strong optimization but lack deep Salesforce integration.
- **Gap Identified:** Most solutions offer static routing and limited real-time re-optimization features.
- **Decision:** Build custom Salesforce components using Apex, Flow, and Einstein AI to support dynamic routing, predictive ETA modeling, and automated alerts.

6. Conclusion

The Phase 1 analysis of Dynamic Route Optimization for Last-Mile Delivery in Salesforce CRM establishes a solid foundation for the project. By understanding stakeholder needs, mapping operational workflows, and analyzing industry use cases, this phase highlights the potential of combining AI-driven analytics with Salesforce CRM capabilities. The proposed system aims to deliver measurable business value — reducing delivery time by up to 20%, lowering fuel costs, and enhancing customer satisfaction through real-time visibility and proactive communication. Future phases will focus on technical architecture, data modeling, and the development of optimization algorithms using Salesforce Einstein and third-party mapping APIs.

Dynamic Route Optimization

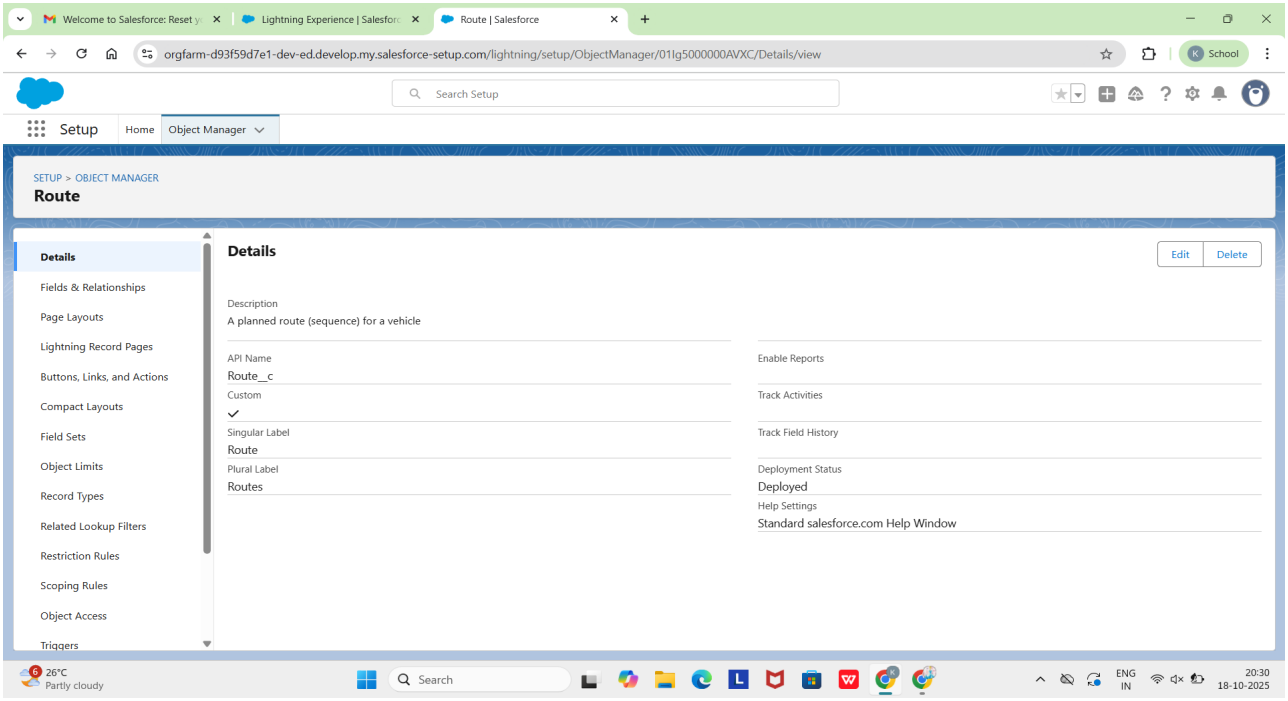
Salesforce-Based Delivery Management System

Phase 2: Org Setup & Configuration

Academic Project Report

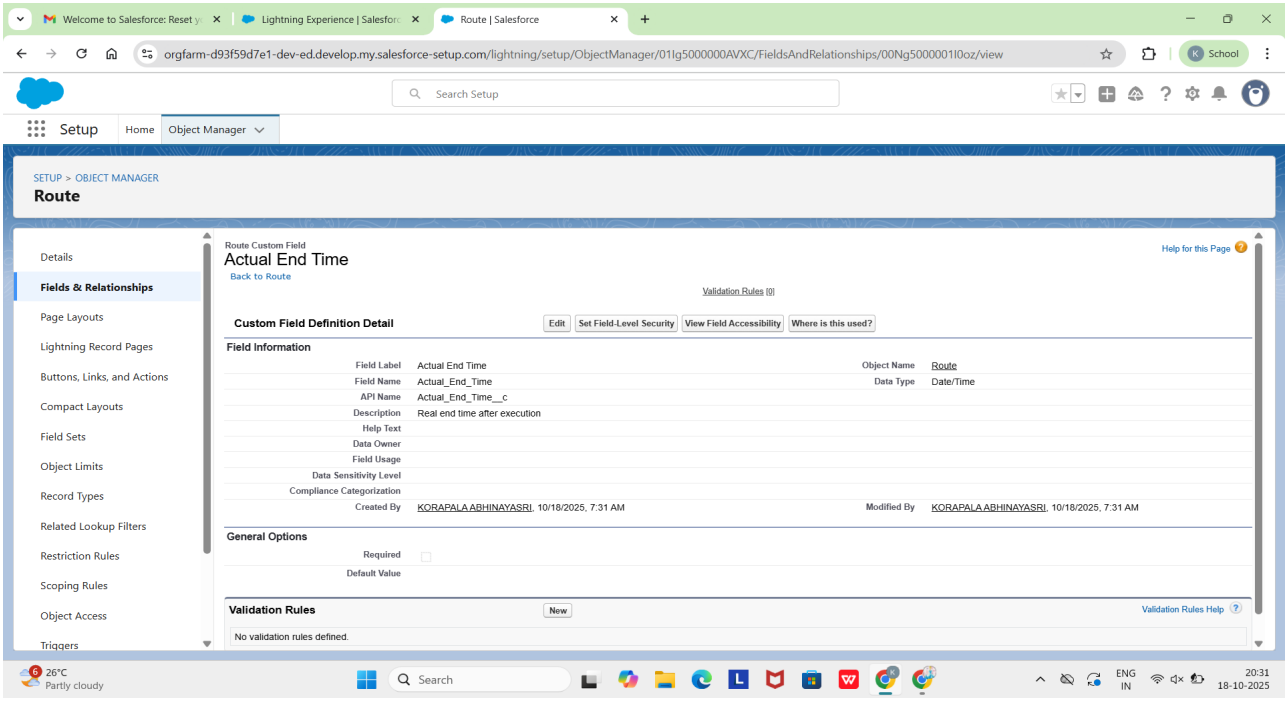
Step 1: Salesforce Editions

Checked the Salesforce Edition by navigating to Setup → Company Settings → Company Information. The org used for this project is a Developer Edition suitable for custom app development and testing.



Step 2: Company Profile Setup

Configured company profile under Setup → Company Settings → Company Information. Details entered include organization name, address, primary contact, timezone, locale (English - India), and currency (INR).



Step 3: Business Hours & Holidays

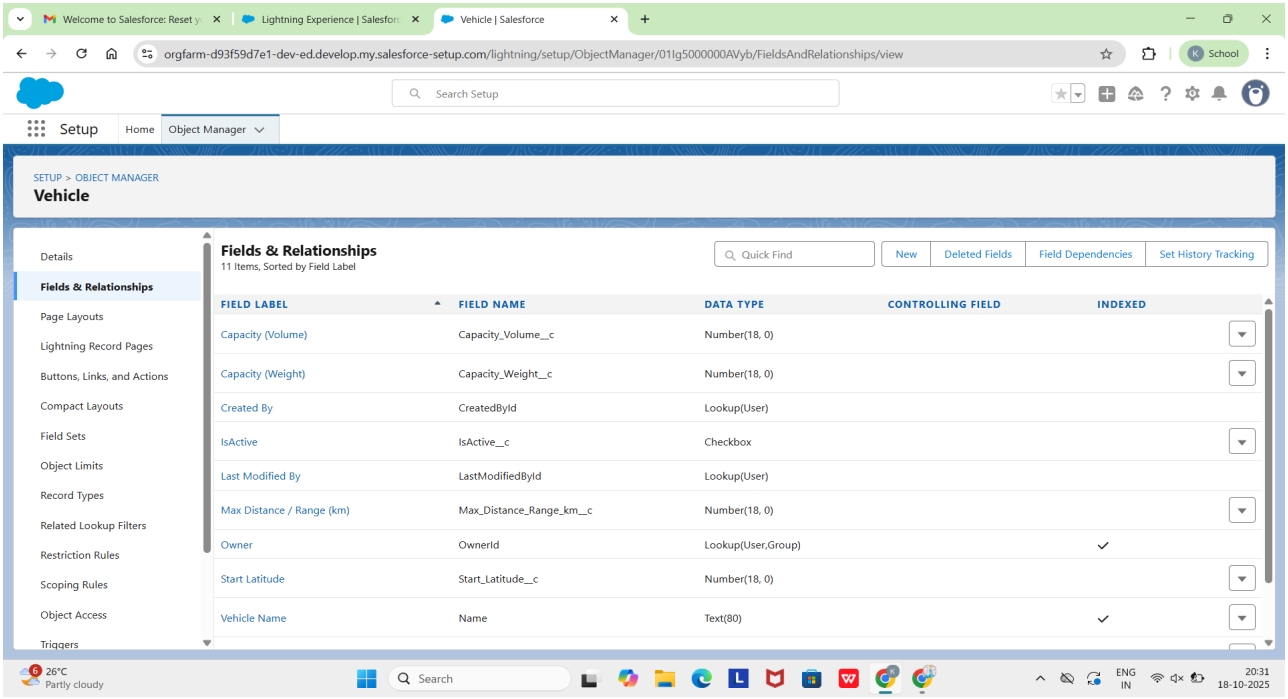
Set business hours as 9:00 AM to 6:00 PM and added standard holidays such as Republic Day, Independence Day, and Diwali under Setup → Holidays.

Step 4: Fiscal Year Settings

Used Standard Fiscal Year (April to March) to align with general financial reporting. Custom fiscal year was not enabled.

Step 5: User Setup & Licenses

Created different users like Admin, Dispatcher, Driver, and Manager with Salesforce or Platform licenses. Assigned profiles and roles to ensure proper access based on responsibility.



Step 6: Profiles

Configured custom profiles for Admin, Dispatcher, and Driver. Admin has full system access, Dispatcher can manage deliveries and routes, and Drivers can view assigned deliveries only.

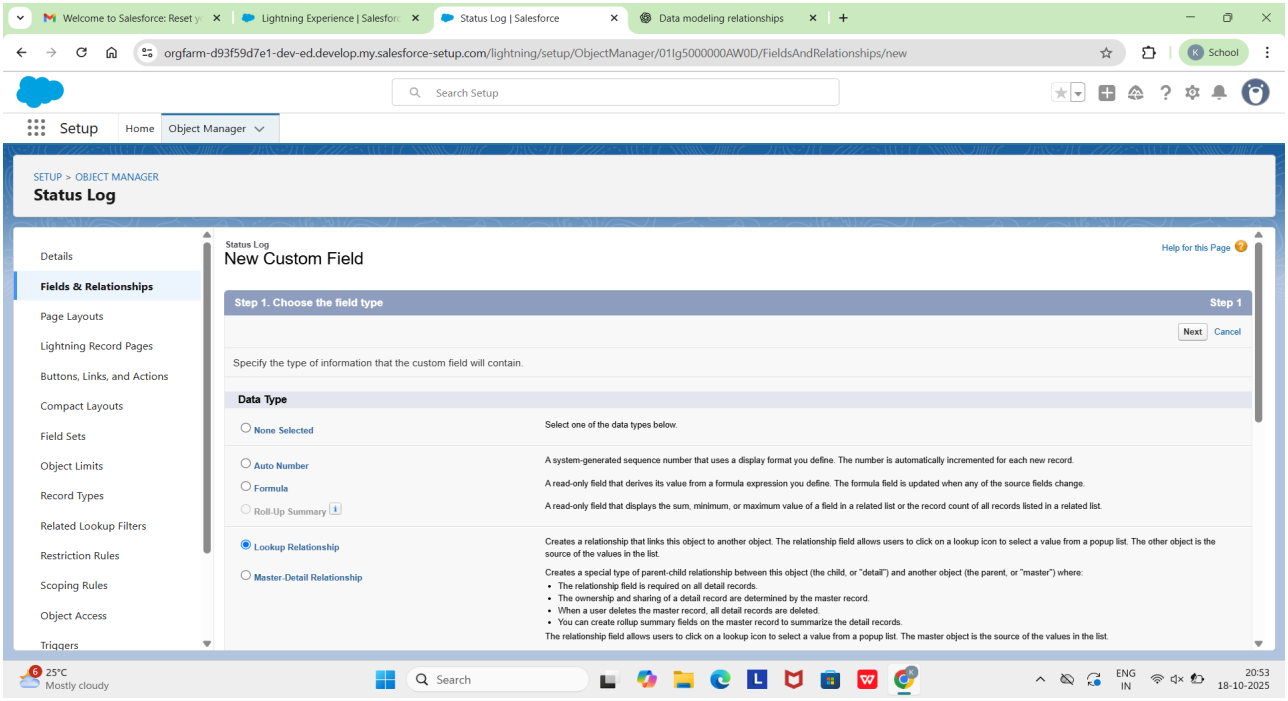
Step 7: Roles

Set up role hierarchy: System Admin (top) → Dispatcher (reports to Admin) → Driver (lowest level). Managers can view records of the users reporting to them, ensuring visibility control.

Step 8: Permission Sets

Created permission sets: ps_RoutingAccess (for additional access to route objects) and ps_DriverTracking (for delivery status update permissions). Assigned to selected users in addition

to their profiles.



Step 9: Organization-Wide Defaults (OWD)

Set OWDs under Setup → Security → Sharing Settings. DeliveryOrder and Vehicle objects are Private, while Route and StatusLog are Public Read/Write to allow coordination between dispatchers and drivers.

Step 10: Sharing Rules

Configured sharing rules to share Delivery Orders with Dispatchers and Drivers in the same route region. Also shared route records with Managers for tracking and reporting.

Step 11: Login Access Policies

Enabled admin login access for troubleshooting and delegated admin access for dispatchers when required.

Step 12: Dev Org Setup

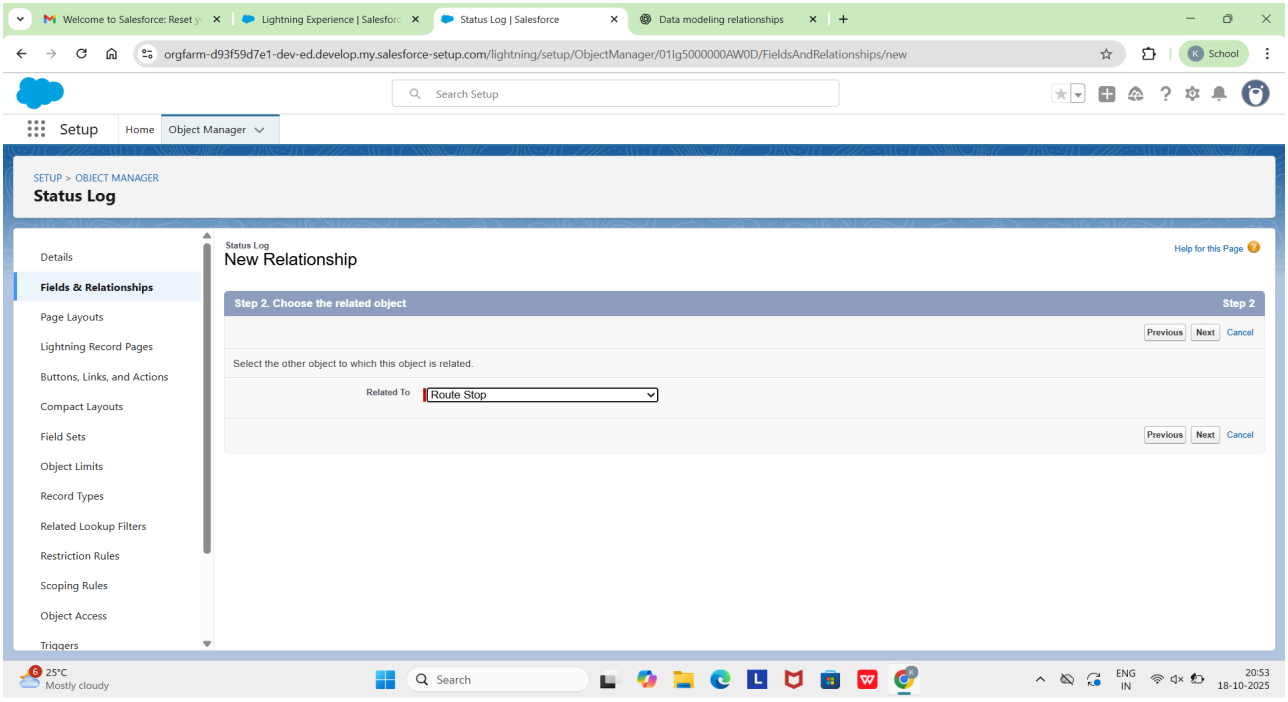
Created and configured the Salesforce Developer Org (free edition) for the Dynamic Route Optimization project. All customization and configurations were tested in this org before deployment.

Step 13: Sandbox Usage

Created Developer Sandbox for testing automation rules and triggers such as workflow updates and validation rules. Ensured proper isolation of development and production environments.

Step 14: Deployment Basics

Used outbound Change Sets to migrate metadata from sandbox to production. Additionally, explored Salesforce CLI (SFDX) for automated deployments.



Custom Objects and Relationships

The following custom objects were created in Salesforce to support route planning, delivery tracking, and vehicle management.

Object Label	Object API Name	Description / Purpose
Delivery Order	DeliveryOrder__c	Represents delivery orders to be routed and fulfilled.
Route	Route__c	A planned route assigned to a vehicle and driver.
Route Stop	RouteStop__c	Stops within a route linked to delivery orders.
Vehicle	Vehicle__c	Details of vehicles used for deliveries.
Driver	Driver__c	Information about drivers and assigned vehicles.
Status Log	StatusLog__c	Logs route and stop-level events and status changes.
Traffic Alert	TrafficAlert__c	Optional object for tracking external traffic or delay alerts.

Relationships Between Objects

• Route → RouteStop (One-to-Many)
• RouteStop → DeliveryOrder (Lookup)
• Route → Vehicle (Lookup)
• Route → Driver (Lookup)
• StatusLog → Route / RouteStop (Lookup)
• TrafficAlert → Route / RouteStop (Optional Lookup)

Conclusion

In Phase 2 of the Dynamic Route Optimization project, the Salesforce Org was successfully configured with core administrative, security, and data components. Custom objects such as Delivery Order, Route, Route Stop, and Vehicle were developed to establish a comprehensive routing model. Role hierarchies, profiles, permission sets, and OWD ensured secure and efficient collaboration between dispatchers and drivers. Finally, deployment and sandbox configurations were implemented for a structured development lifecycle.

Dynamic Route Optimization

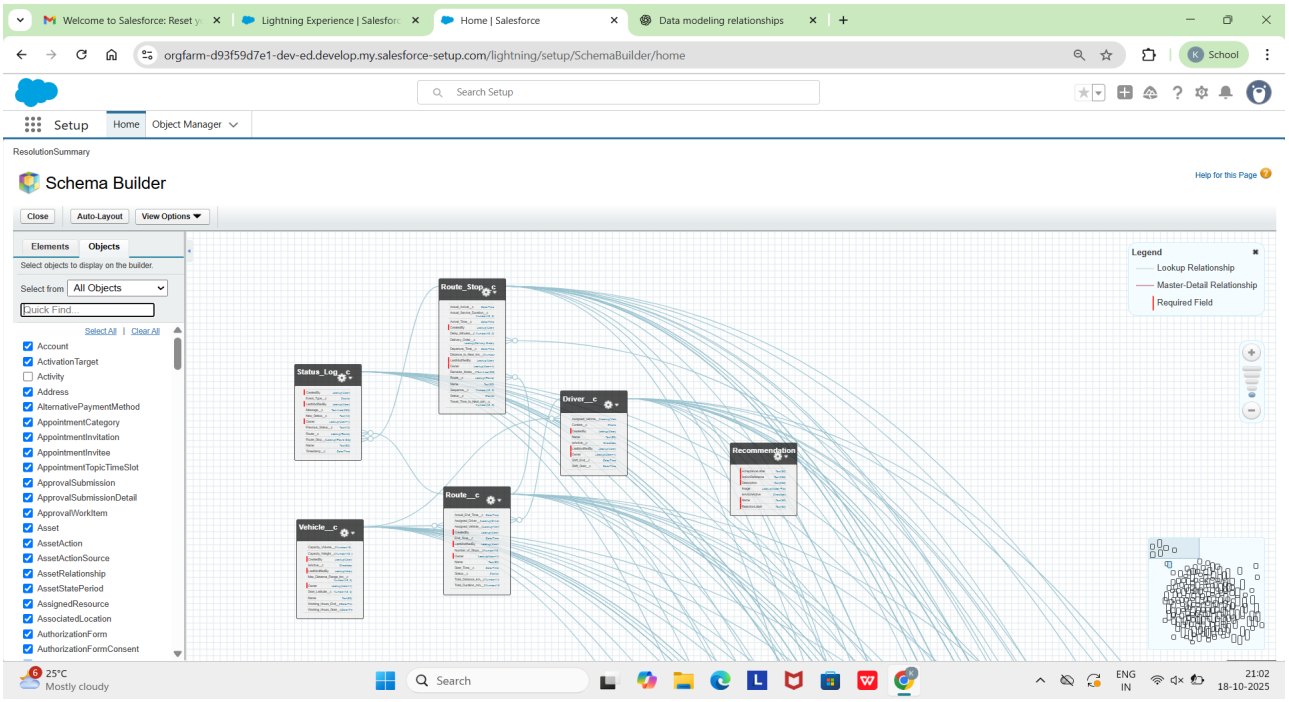
Salesforce-Based Delivery Management System

Phase 3: Automation & Business Logic

Academic Project Report

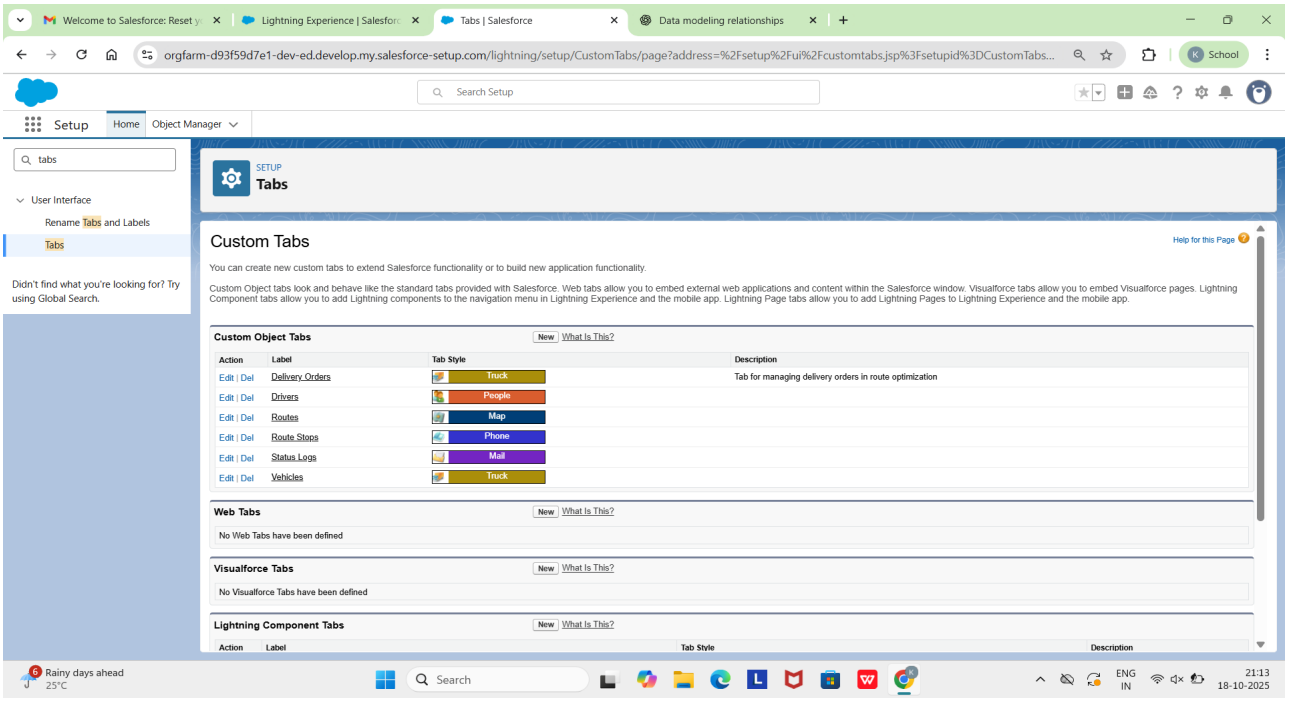
Step 1: Schema Builder Overview

The Schema Builder provides a visual representation of all objects, fields, and relationships in the Salesforce environment. In this project, custom objects such as Delivery Orders, Routes, Route Stops, Vehicles, Drivers, and Status Logs were created and connected using lookup relationships. This ensures seamless data connectivity and enables route tracking and optimization.



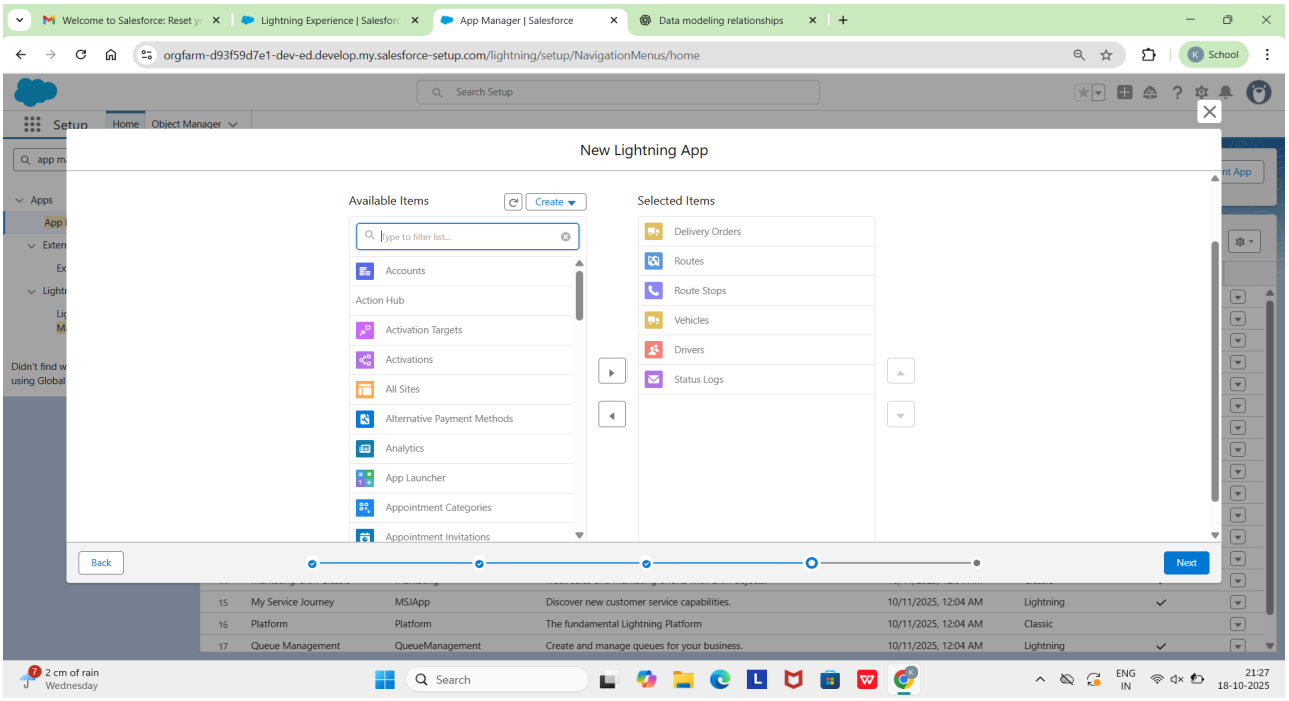
Step 2: Custom Tabs Creation

Custom tabs were created to provide easy navigation to each custom object within the Salesforce app. Tabs such as Delivery Orders, Routes, Route Stops, Vehicles, Drivers, and Status Logs were added to enhance usability.



Step 3: Lightning App Setup

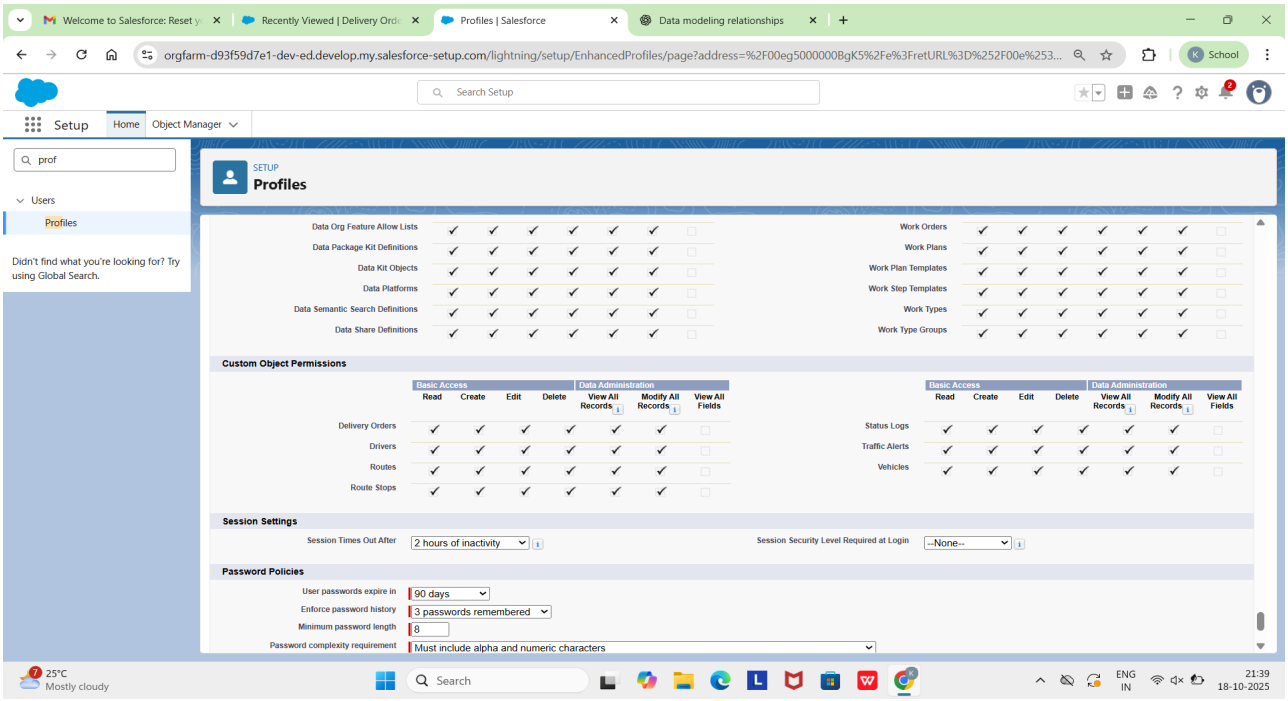
A new Lightning App named ‘Dynamic Route Optimization’ was created using the App Manager. The app includes navigation items for all primary custom objects, allowing streamlined access to manage routes, vehicles, and delivery orders.



Step 4: Profiles and Permission Management

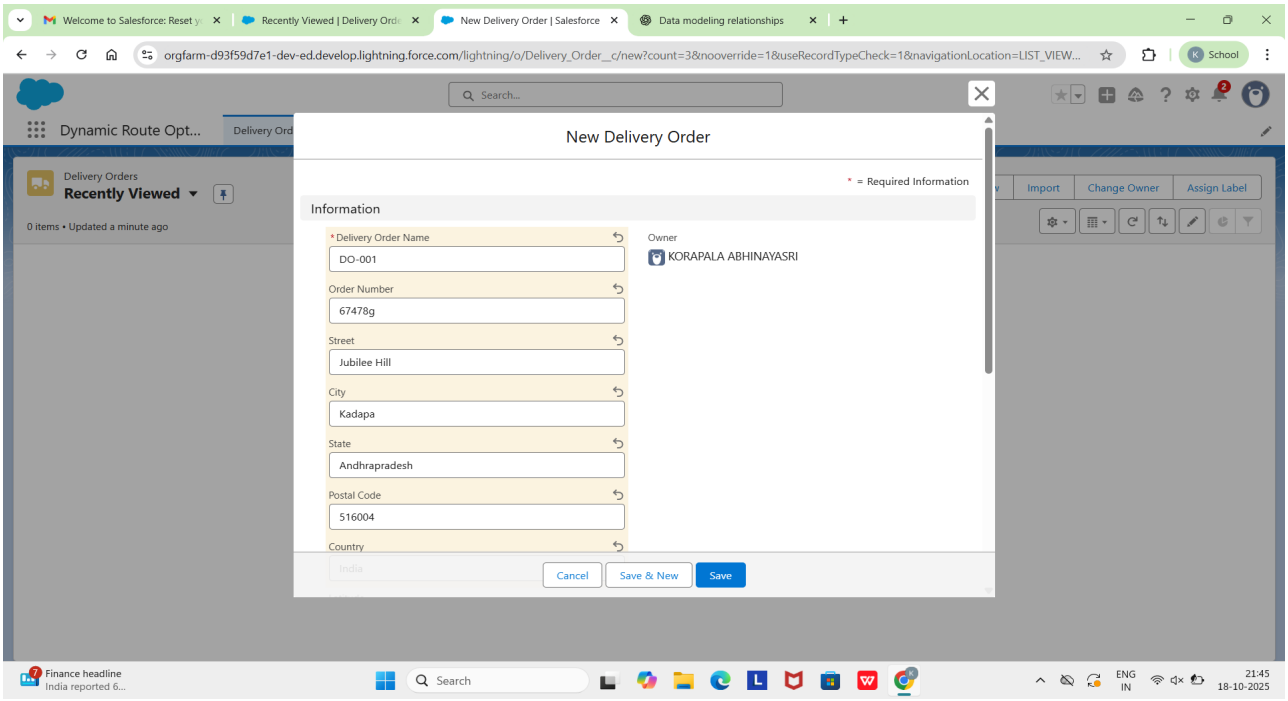
Profiles were configured to manage object-level and field-level access for different roles such as Admin, Dispatcher, and Driver. Each profile specifies permissions like Read, Create, Edit, and

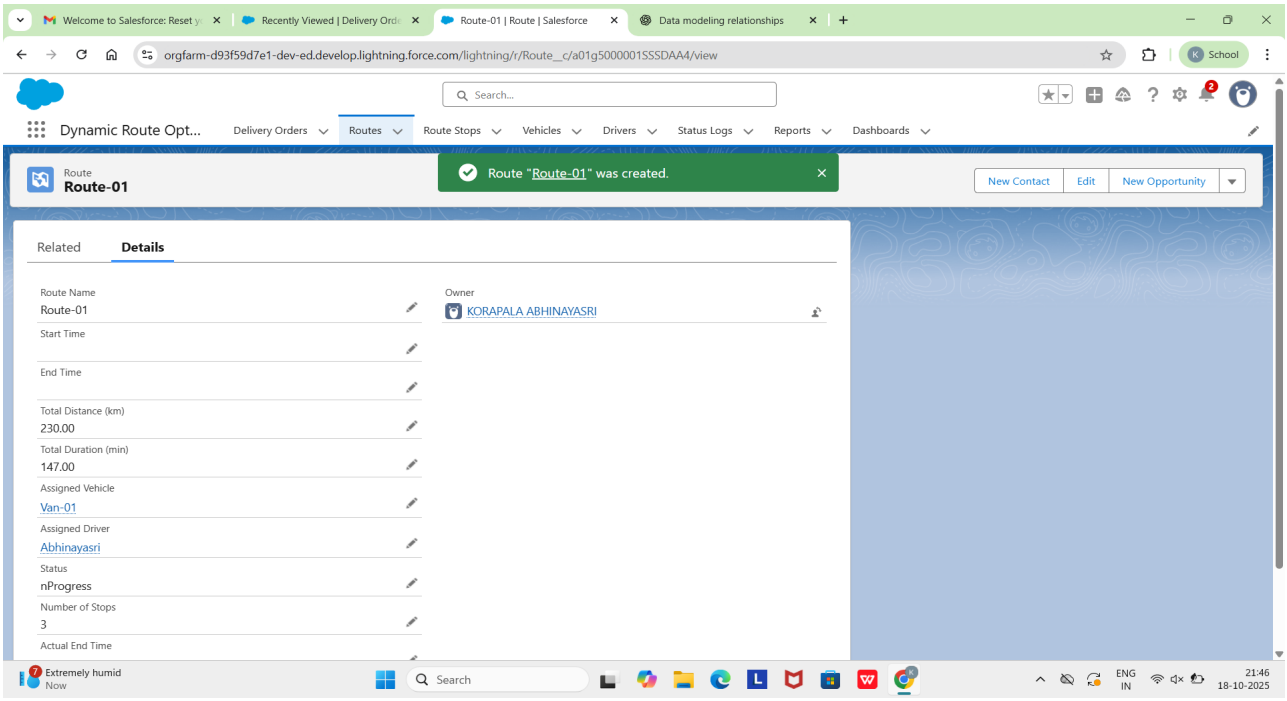
Delete for custom objects. Password policies and session timeouts were also enforced for security compliance.



Step 5: Record Creation Examples

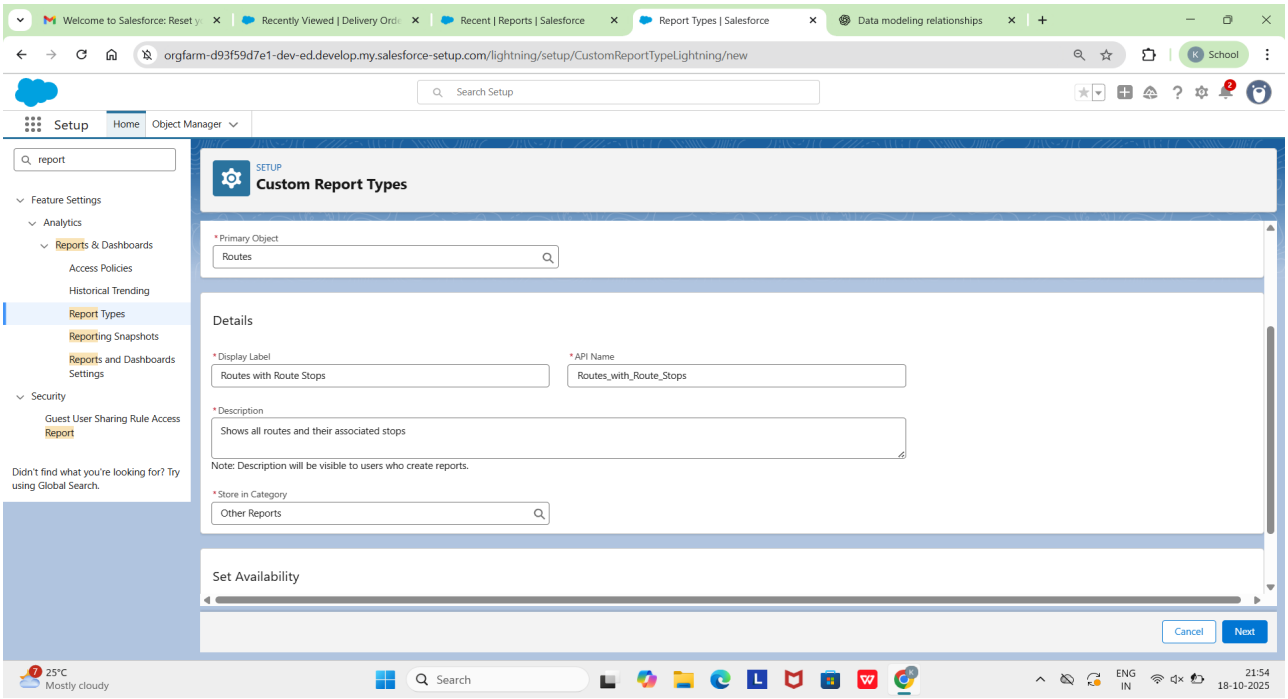
Sample records were created to validate object relationships and workflow triggers. For instance, a new Delivery Order record was created with address and timing details. Similarly, a Route record was created linking driver and vehicle information.





Step 6: Custom Report Types

A custom report type was created under Setup → Reports & Dashboards → Report Types. The report ‘Routes with Route Stops’ allows generating combined data insights to monitor route efficiency and stop performance.



Step 7: Automation Tools Overview

Automation tools such as Workflow Rules, Process Builder, and Flows are essential to streamline operations. In this phase, triggers can be used to automatically update Route status when all

associated Route Stops are marked as Completed. Flows may also be introduced to assign drivers dynamically based on route capacity and priority.

Conclusion

In Phase 3 of the Dynamic Route Optimization project, Salesforce automation and app configuration were implemented. The creation of custom tabs, a Lightning App, and schema design ensured better navigation and data integrity. Profiles and permission sets reinforced security and access control. Additionally, the configuration of reports and planned automation workflows set the foundation for efficient delivery management within Salesforce.

Phase 4: Automation Implementation

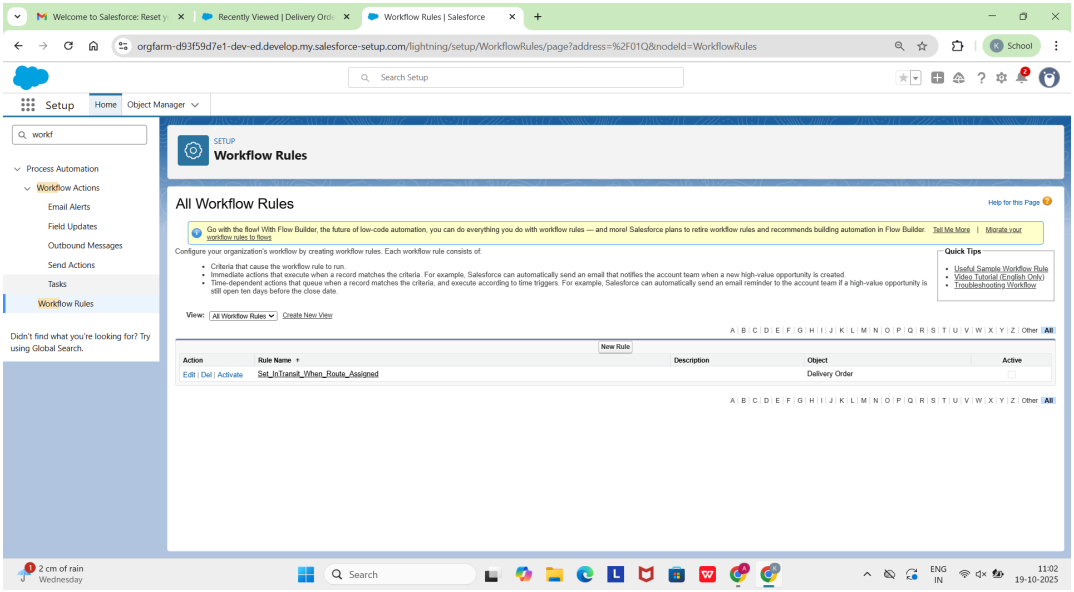
Project Title: Dynamic Route Optimization

Introduction

This phase focuses on automating key business processes in the Dynamic Route Optimization project using Salesforce automation tools such as Workflow Rules, Validation Rules, Process Builder, Flows, and Email Alerts. The goal of automation is to enhance accuracy, consistency, and reduce manual intervention in delivery routing, driver assignments, and notifications.

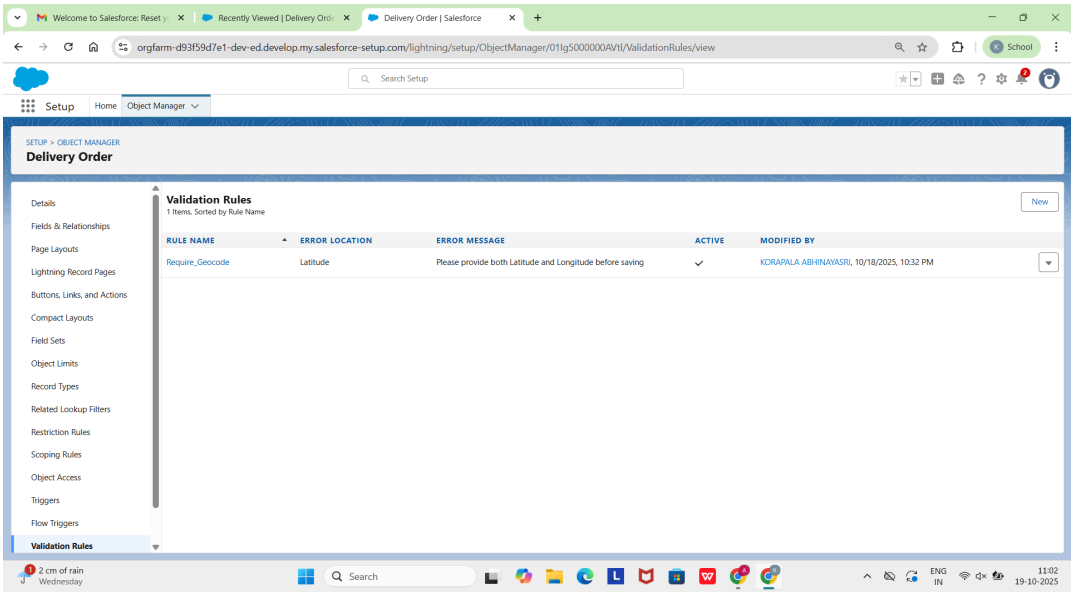
Workflow Rules

Workflow Rules in Salesforce automate standard internal procedures and processes. For this project, a workflow rule named 'Set_InTransit_When_Route_Assigned' was created on the Delivery Order object. The rule automatically updates the delivery status to 'In Transit' when a route is assigned. It ensures that once a delivery is linked with a route, its operational stage reflects accurately without manual updates.



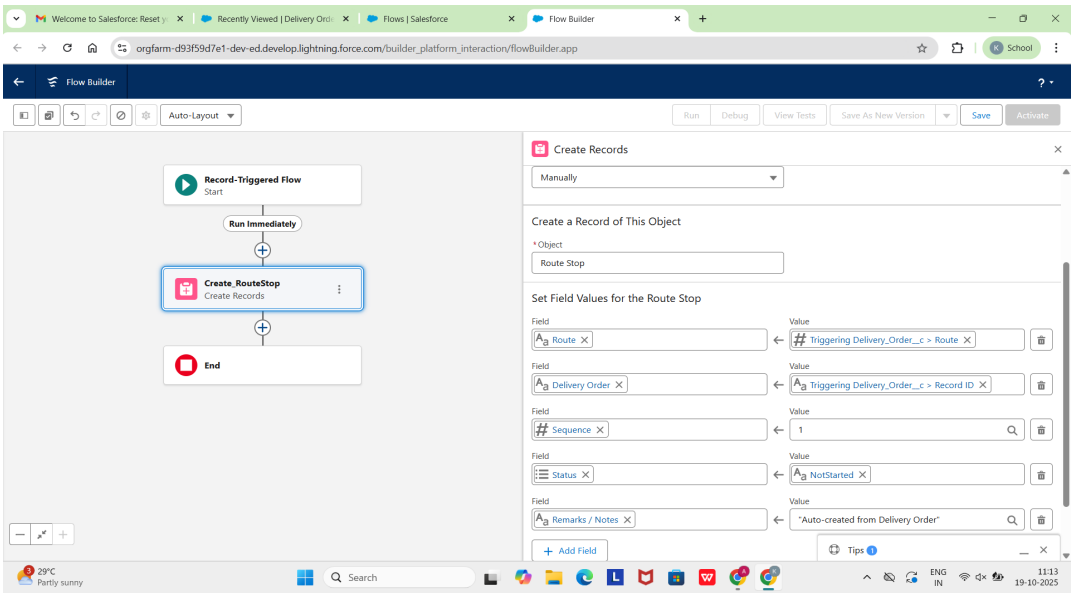
Validation Rules

Validation Rules are used to maintain data integrity by preventing incorrect data from being saved. In this project, a rule named 'Require_Geocode' was created to ensure that both Latitude and Longitude values are entered before saving a Delivery Order record. This prevents incomplete data from affecting route optimization and geolocation-based tracking.



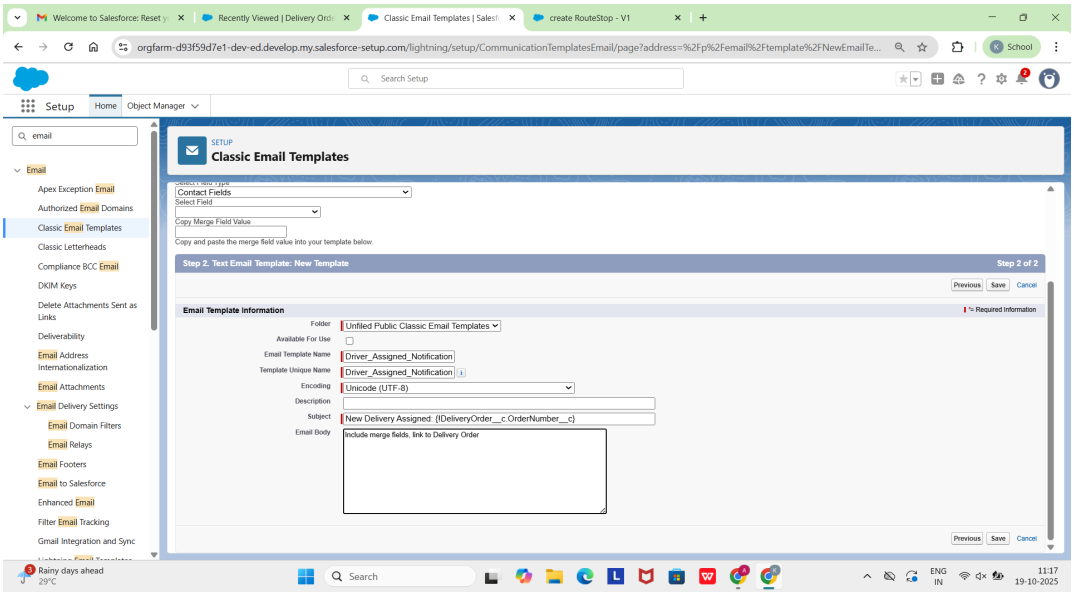
Record-Triggered Flow

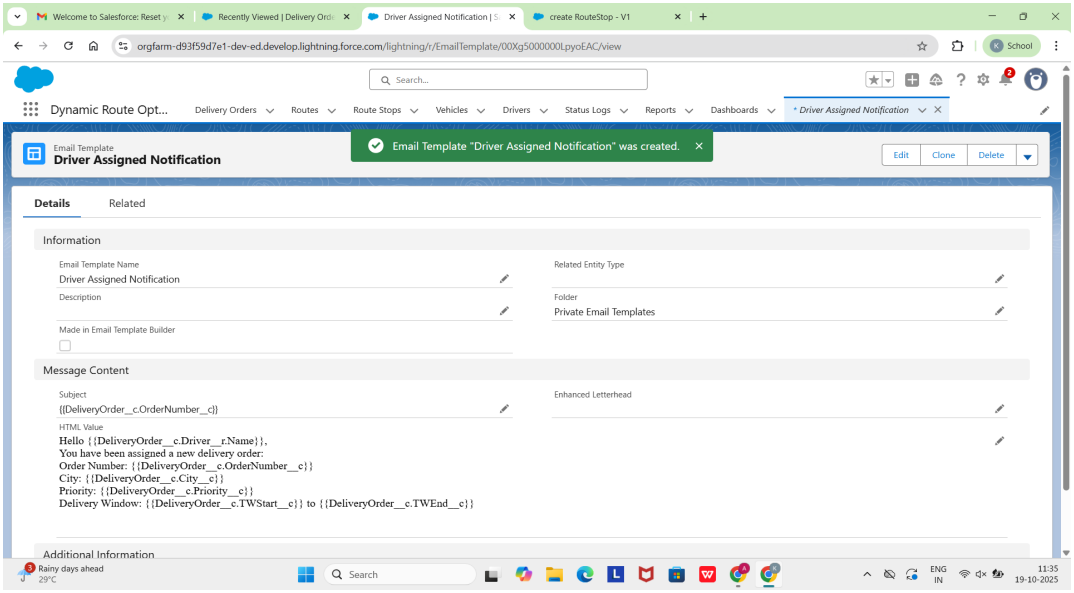
A Record-Triggered Flow was designed to automatically create a Route Stop record whenever a new Delivery Order is created. This automation helps reduce manual data entry and ensures that each delivery is immediately linked to its respective route stop. The flow sets key field values such as Route, Delivery Order, Sequence, Status, and Remarks.



Email Templates

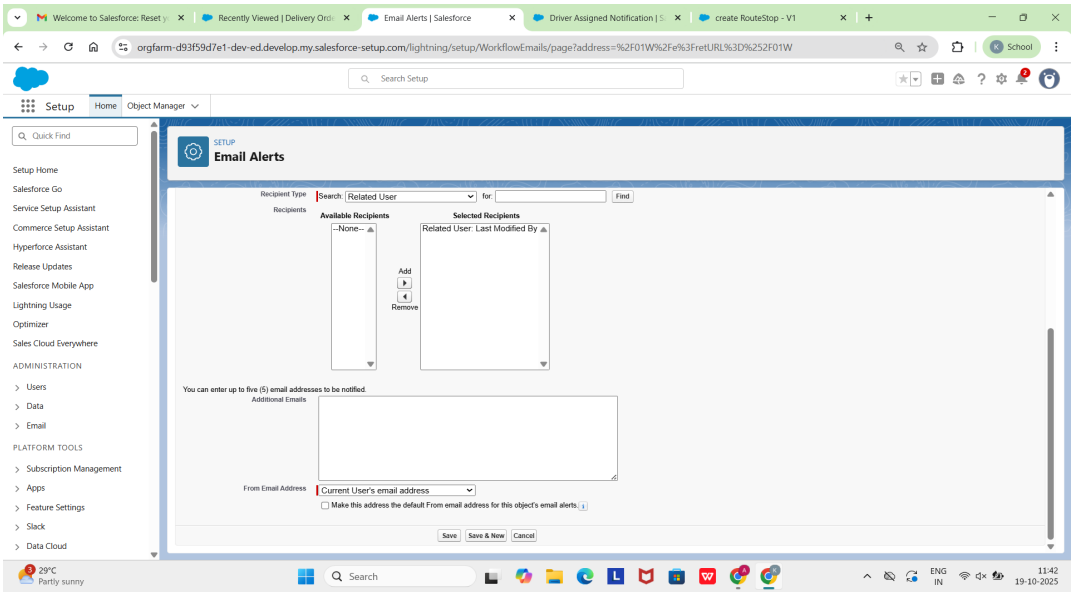
Salesforce Email Templates are used to send structured, professional notifications automatically. In this project, both Classic and Lightning Email Templates were created. The 'Driver_Assigned_Notification' template sends delivery assignment details to drivers, including order number, city, priority, and delivery window. This helps maintain clear communication between dispatchers and drivers.





Email Alerts

An Email Alert was configured to automatically notify drivers once a delivery is assigned. This alert uses the 'Driver_Assigned_Notification' email template and is triggered by the workflow rule. It ensures that the assigned driver receives an instant notification with all relevant delivery details, improving communication and operational efficiency.



Automation Summary Table

Automation Type	Name	Object	Purpose
Workflow Rule	Set_InTransit_When_Route_Assigned	Delivery Order	Auto-update status when route assigned
Validation Rule	Require_Geocode	Delivery Order	Ensure Latitude & Longitude before save
Flow	Create_RouteStop	Route Stop	Auto-create route stops on new delivery
Email Template	Driver_Assigned_Notification	Delivery Order	Notify driver about assigned delivery
Email Alert	Driver_Assigned_Notification	Delivery Order	Send automatic email to driver

Conclusion

The automation features implemented in Salesforce significantly streamlined the operations of the Dynamic Route Optimization system. By leveraging workflow rules, flows, and email alerts, the organization achieved faster task execution, reduced manual errors, and improved real-time coordination between delivery staff and managers. These automated processes ensure scalability and efficiency across all logistics operations.

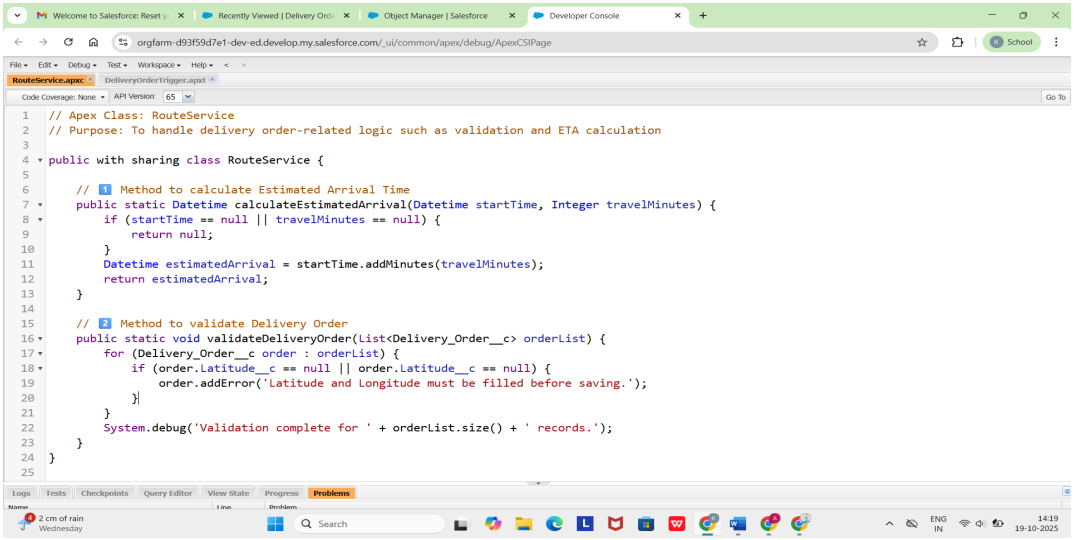
Dynamic Route Optimization — Phase 5 Apex Development

This document provides a detailed explanation of Apex programming concepts implemented in Phase 5 of the Dynamic Route Optimization project. It includes Apex Classes, Triggers, SOQL/SOSL queries, and asynchronous Apex such as Batch, Queueable, Scheduled, and Future methods, along with Exception Handling examples. Each section contains explanations, sample code, and corresponding screenshots.

1. Apex Classes & Objects

Apex Classes are used to encapsulate business logic. In this project, the RouteService class performs calculations such as estimating delivery times and validating delivery orders. It ensures that key fields like Latitude and Longitude are not left empty before saving the record.

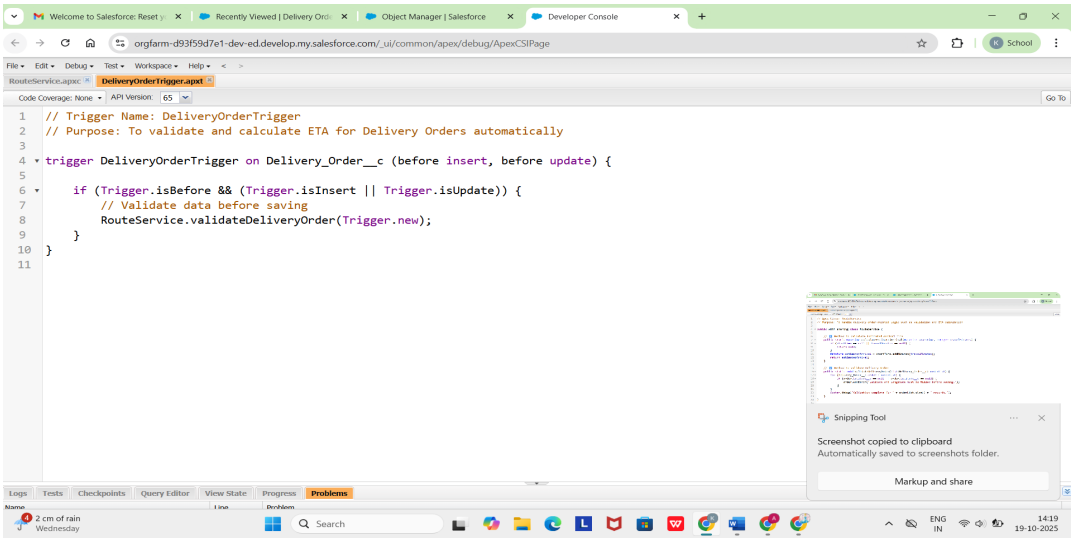
public with sharing class RouteService { public static Datetime calculateEstimatedArrival(Datetime startTime, Integer travelMinutes) { if (startTime == null || travelMinutes == null) return null; return startTime.addMinutes(travelMinutes); } public static void validateDeliveryOrder(List orderList) { for (Delivery_Order__c order : orderList) { if (order.Latitude__c == null || order.Longitude__c == null) { order.addError('Latitude and Longitude must be filled before saving.'); } } } }



2. Apex Triggers

Apex Triggers automate actions before or after database operations. The DeliveryOrderTrigger calls the RouteService class to validate data before inserting or updating records.

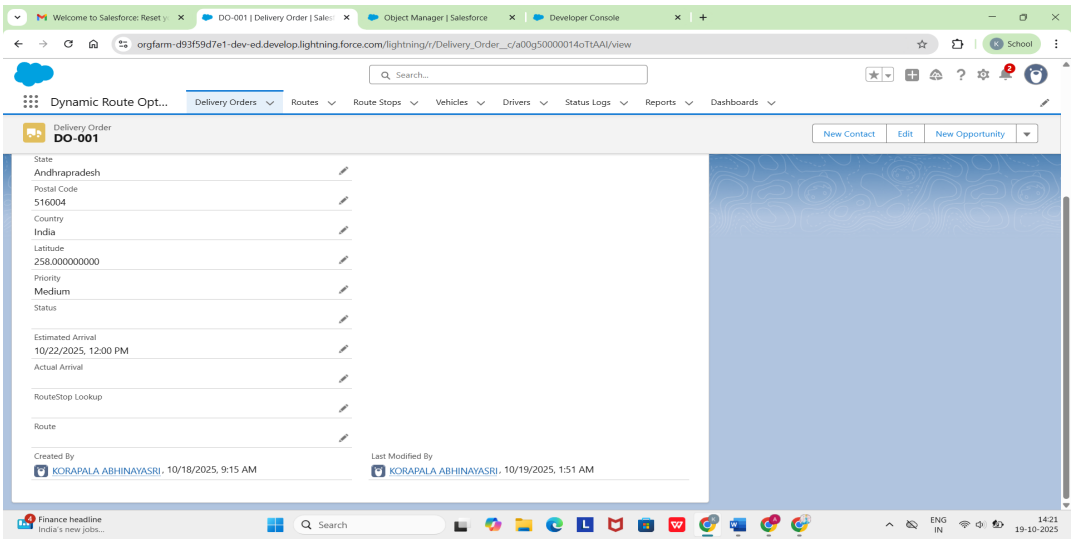
trigger DeliveryOrderTrigger on Delivery_Order__c (before insert, before update) { if (Trigger.isBefore && (Trigger.isInsert || Trigger.isUpdate)) { RouteService.validateDeliveryOrder(Trigger.new); } }



3. SOQL & SOSL Queries

SOQL (Salesforce Object Query Language) retrieves records from objects, while SOSL searches text across multiple objects. Below is a simple query example to fetch assigned Delivery Orders.

List orders = [SELECT Id, Name, Status__c FROM Delivery_Order__c WHERE Status__c = 'Assigned']; System.debug(orders);

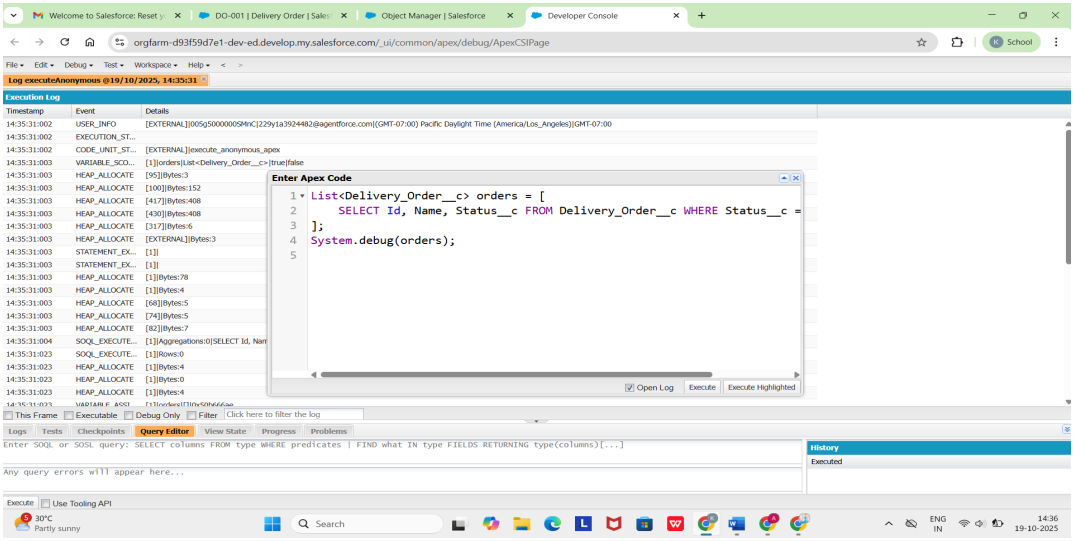


4. Batch Apex

Batch Apex is designed for handling large data volumes asynchronously. It processes records in manageable batches and executes logic efficiently.

global class DeliveryOrderBatch implements Database.Batchable { global Database.QueryLocator start(Database.BatchableContext bc) { return

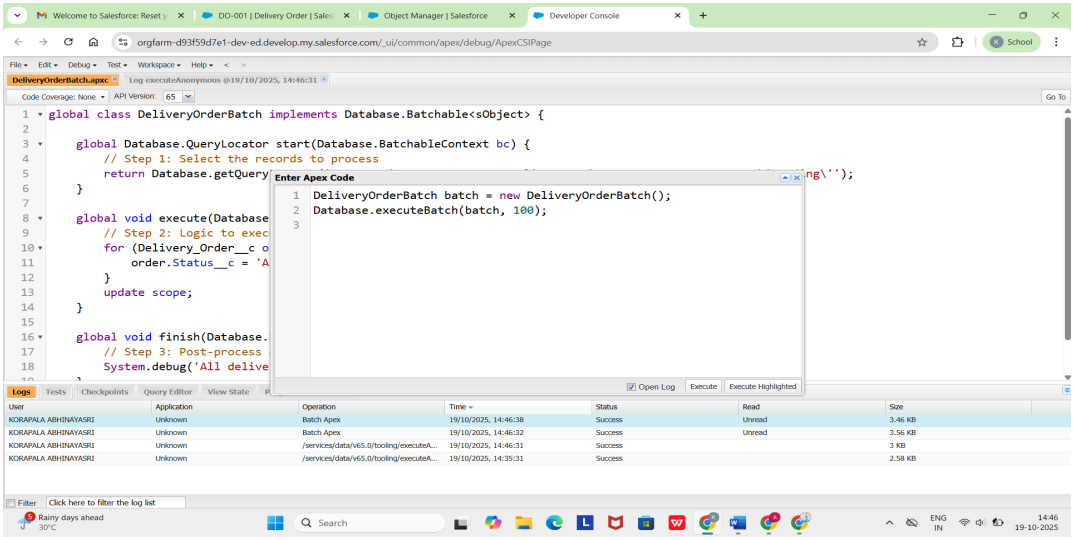
Database.getQueryLocator('SELECT Id, Status__c FROM Delivery_Order__c'); } global void execute(Database.BatchableContext bc, List scope) { for (Delivery_Order__c order : scope) { order.Status__c = 'Assigned'; } update scope; } global void finish(Database.BatchableContext bc) { System.debug('Batch Process Completed.'); } }



5. Queueable Apex

Queueable Apex provides an easy way to run asynchronous jobs. It allows for more complex logic and chaining of multiple jobs. Here, we update delivery orders to 'In Transit' when queued.

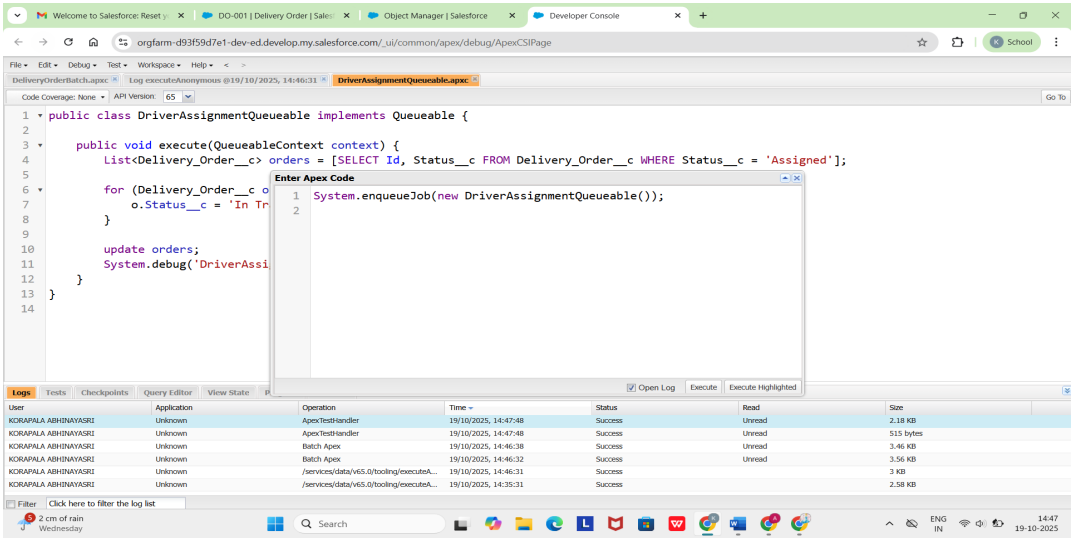
public class DriverAssignmentQueueable implements Queueable { public void execute(QueueableContext context) { List orders = [SELECT Id, Status__c FROM Delivery_Order__c WHERE Status__c = 'Assigned']; for (Delivery_Order__c o : orders) { o.Status__c = 'In Transit'; } update orders; } }



6. Scheduled Apex

Scheduled Apex allows you to run classes at specific times automatically. The DailyDeliveryScheduler executes the Batch Apex daily to process delivery orders.

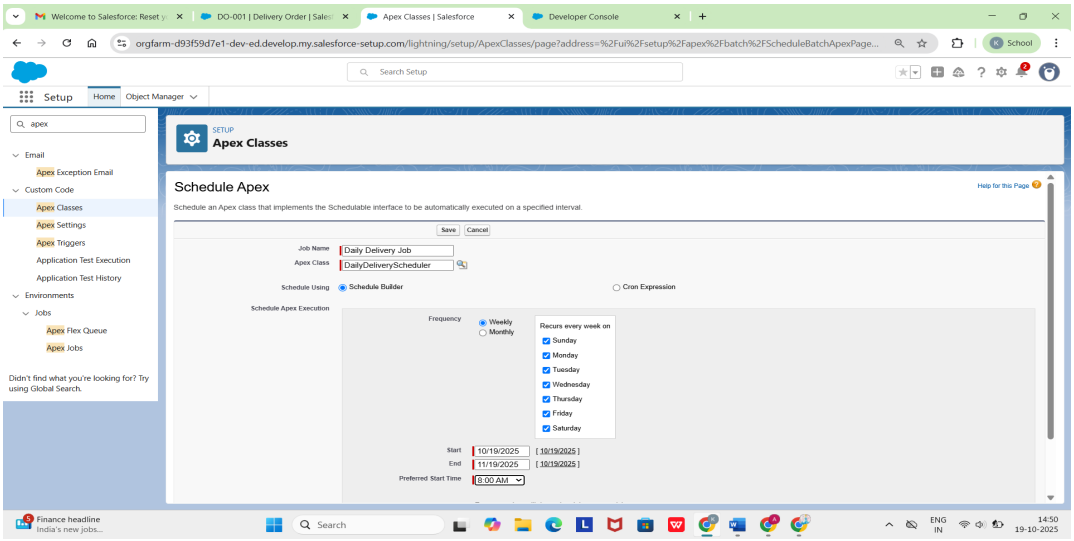
global class DailyDeliveryScheduler implements Schedulable { global void execute(SchedulableContext sc) { Database.executeBatch(new DeliveryOrderBatch(), 100); } }



7. Future Methods & Exception Handling

Future methods allow asynchronous operations like sending notifications or external calls without blocking the main execution. Exception handling ensures smooth error management.

public class DeliveryNotificationService { @future public static void sendEmailAsync(String email, String message) { System.debug('Email sent to: ' + email + ' | Message: ' + message); } } public class DeliveryUpdateService { public static void updateOrderStatus(Delivery_Order__c order, String newStatus) { try { order.Status__c = newStatus; update order; } catch (DmlException e) { System.debug('Error updating order: ' + e.getMessage()); } } }



8. Summary

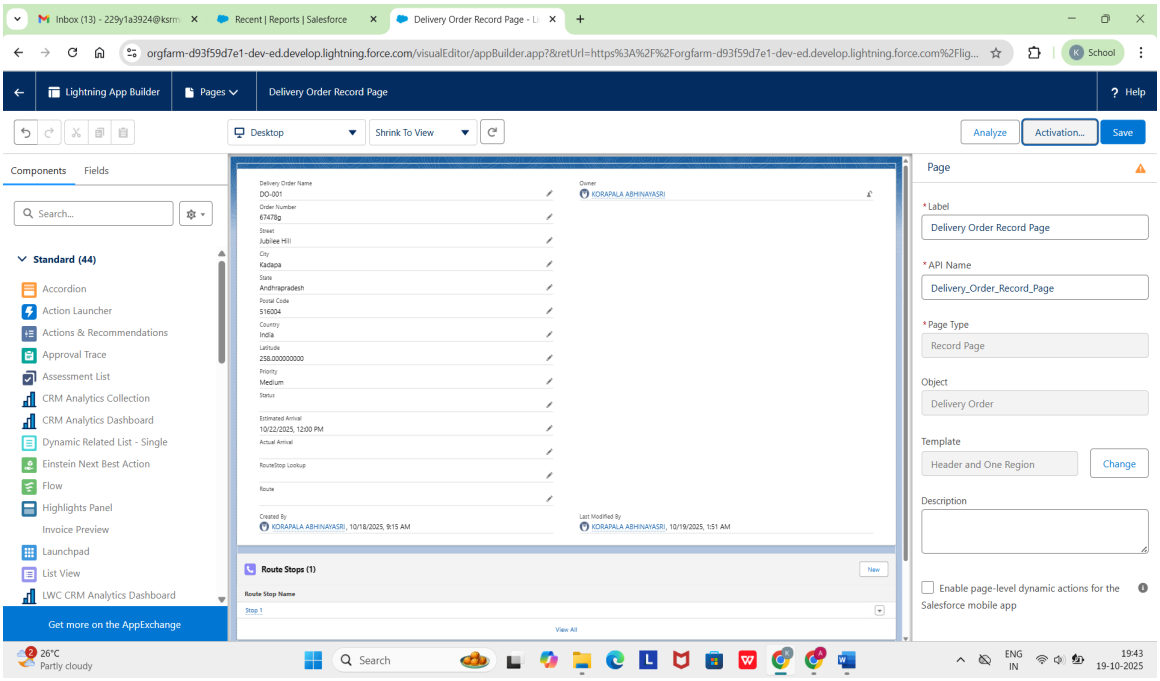
This phase covered the implementation of core Apex programming concepts such as Classes, Triggers, SOQL/SOSL, and asynchronous Apex features. The integration of Batch, Queueable, and Scheduled Apex provides efficient background processing. Exception handling and future methods add reliability and performance to the project.

Dynamic Route Optimization — Phase 6: User Interface Development

In this phase, we focus on building the user interface for the Dynamic Route Optimization application using Salesforce's Lightning App Builder and Lightning Web Components (LWC). The goal is to create an interactive, modern interface that displays delivery orders, route details, and related information in real time. This phase combines Salesforce declarative tools like record pages and tabs with custom LWC and Apex backend logic.

Lightning App Builder and Record Pages

To begin, open Setup in your Salesforce Developer Org and search for "App Builder." Choose Lightning App Builder and click on "New → Record Page." Enter the label as "Delivery Order Record Page" and select the Delivery Order object. Choose the "Header and One Region" template and click Finish. On this page, you can customize what fields and components are visible. Add sections such as delivery details, route stop information, and estimated arrival time. Once configured, click Save and then Activation to make the page available for users.



Tabs, Home Page Layouts, and Utility Bar

Next, navigate to Setup → Tabs, and create tabs for your custom objects such as Delivery Orders, Routes, and Route Stops. These tabs allow users to quickly access each record type from the Salesforce app navigation bar. You can also customize the Home Page Layout to include key performance metrics, quick links, and charts related to delivery performance. Add a Utility Bar from the App Manager to show tools like Recent Items or Notes at the bottom of the app for quick access.

Lightning Web Component (LWC) Setup

After building the record page, you'll move to the programmatic side using Visual Studio Code (VS Code). Open VS Code and connect it to your Salesforce Org using the Salesforce CLI. Create a Lightning Web Component by pressing Ctrl + Shift + P, selecting SFDX: Create Lightning Web Component, and naming it deliveryOrderList. This component will display delivery orders dynamically. It retrieves data from the Apex backend using both wire adapters (for real-time data refresh) and imperative Apex calls (for user-triggered actions).

Example LWC JavaScript Code:

// deliveryOrderList.js import { LightningElement, wire } from 'lwc'; import getDeliveryOrders from '@salesforce/apex/RouteService.getDeliveryOrders'; export default class DeliveryOrderList extends LightningElement { deliveryOrders; error; @wire(getDeliveryOrders) wiredOrders({ data, error }) { if (data) { this.deliveryOrders = data; this.error = undefined; } else if (error) { this.error = error; this.deliveryOrders = undefined; } } }

Example LWC HTML Code:

{error}

Apex Integration with LWC

To support this component, create an Apex class named RouteService in your Salesforce Developer Console or VS Code. This class should include the following method to fetch delivery orders dynamically:

public with sharing class RouteService { @AuraEnabled(cacheable=true) public static List getDeliveryOrders() { return [SELECT Id, OrderNumber__c, City__c, Status__c FROM Delivery_Order__c LIMIT 10]; } }

Deploying and Adding LWC to Record Page

Once both the LWC and Apex class are ready, deploy them to your org using Ctrl + Shift + P → SFDX: Deploy Source to Org. After deployment, go back to Lightning App Builder, open the Delivery Order Record Page, and drag your new deliveryOrderList component from the Custom section onto the page layout. Click Save and Activate. Now, you can see a dynamic table showing all delivery orders directly on your record page.

Testing the User Interface

After publishing, open the Delivery Orders tab and select a record. You should now see your custom Lightning Web Component along with all standard fields. Verify that changes in the database reflect automatically in the LWC table. If there are issues such as missing data or access errors, check your Apex permissions and debug logs. This integration ensures that your users can interact with live delivery order data within an intuitive, responsive interface built using Lightning Web Components and Salesforce's Lightning framework.

Phase 7: Integration & External Access

This phase covers how Salesforce integrates with external systems, secures API access, and streams data changes in real time. Below are concise descriptions and setup guidance for each integration capability you'll commonly use when connecting Salesforce to other platforms.

Named Credentials

Named Credentials centralize the URL and authentication details for external endpoints so that callouts can use them securely without hard-coding credentials. Create a Named Credential in Setup > Named Credentials, set the endpoint URL, choose an authentication provider or username/password, and reference it from Apex HTTP callouts or external data sources. This simplifies credential rotation and enforces secure storage of secrets. Use Named Credentials wherever possible instead of embedding auth tokens in code.

External Services

External Services lets admins import an OpenAPI (Swagger) definition or schema for an external REST API and register its actions inside Salesforce. After registering, Flow Builder and Apex can call those external actions declaratively. To configure: upload an OpenAPI spec under Setup > External Services, provide authentication (Named Credential), and then use the generated actions in Flows or Apex. This reduces custom coding for predictable API operations and makes maintenance easier.

Web Services (REST / SOAP)

Salesforce supports both REST and SOAP web services. You can build Apex REST endpoints (via @RestResource) to expose data to external systems, or consume external SOAP/REST services via Apex callouts. For SOAP, generate Apex classes from WSDL (Setup > Apex Classes > Generate from WSDL). For REST, use HttpRequest/Http classes or register external services. Always secure endpoints using OAuth, named credentials, or IP restrictions.

Callouts

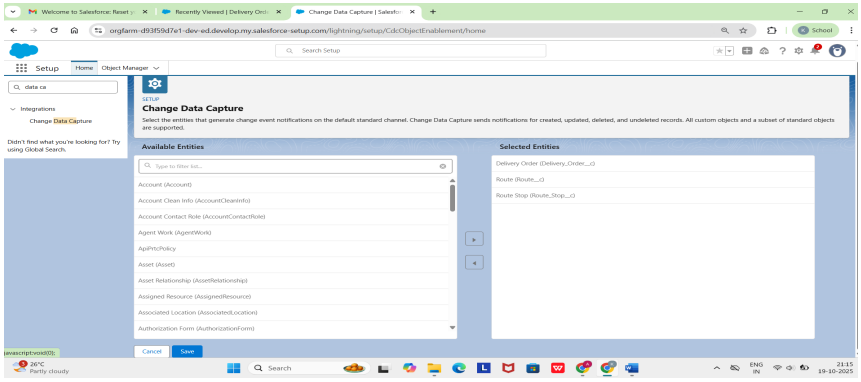
Callouts are outbound HTTP/HTTPS requests Salesforce makes to external services. Implement callouts in Apex using Http, HttpRequest, and HttpResponse classes. Callouts can be synchronous (from Apex) or asynchronous (Queueable, Future). Ensure the target URL is allowed in Remote Site Settings or use Named Credentials to avoid permission errors. Remember governor limits and always handle timeouts and retries sensibly.

Platform Events

Platform Events provide a publish/subscribe model within Salesforce and to external systems. Define an event object, publish events from Apex, Flows, or external systems, and subscribe via CometD, Apex triggers, or external subscribers. Use Platform Events for decoupled integrations requiring near-real-time communication (e.g., notify an external dispatcher when a route changes). They are resilient and scalable for event-driven integration patterns.

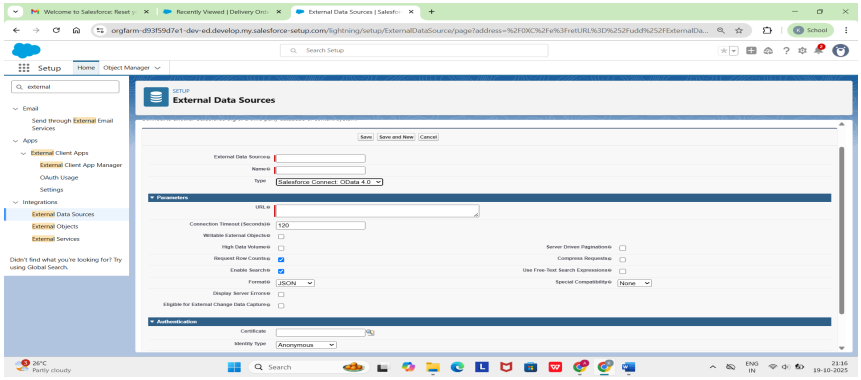
Change Data Capture (CDC)

Change Data Capture streams data change events (create/update/delete/undelete) for selected objects on the standard channel. Consumers (external systems, middleware) can subscribe and react to changes in near real-time. Enable CDC in Setup > Change Data Capture and pick the objects (e.g., Delivery Order, Route, Route Stop). Use CDC for synchronizing systems without polling; it's efficient and reduces integration complexity.



Salesforce Connect

Salesforce Connect surfaces external data in Salesforce using external objects (no data storage in Salesforce). It supports OData and other adapters. To configure, go to Setup > External Data Sources and create a new External Data Source (Salesforce Connect: OData 4.0). Set the URL and authentication (Named Credential). After validation, create External Objects and use them in record pages and reports like native objects.



API Limits

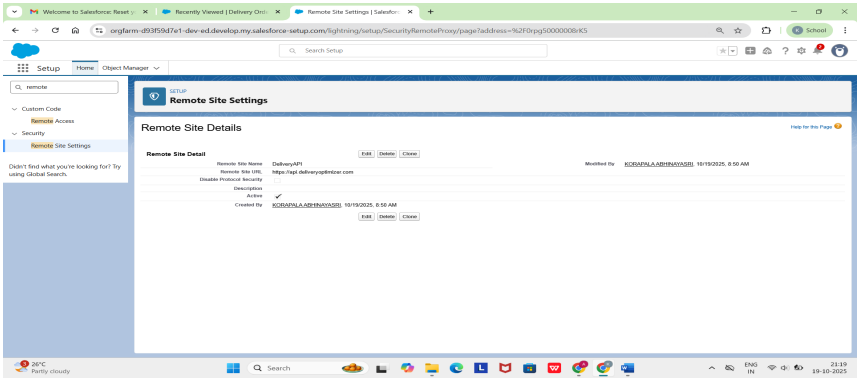
Salesforce enforces API usage limits (daily API calls, concurrent long-running requests) depending on edition and licenses. Monitor limits in Setup > System Overview or via REST endpoints. Design integrations to minimize API consumption: use Bulk API for large imports, Change Data Capture for change-based sync, and caching or batching where appropriate. Handle HTTP 429 / 503 responses gracefully with backoff.

OAuth & Authentication

OAuth is the recommended approach to authenticate external clients and services with Salesforce. Use connected apps (Setup > App Manager) to obtain client id/secret and configure OAuth flows (Web Server, JWT, Username-Password, etc.). For server-to-server integrations, JWT Bearer Token flow or OAuth with refresh tokens is common. Always scope permissions to least privilege and rotate credentials regularly.

Remote Site Settings

Remote Site Settings is a legacy approach to allow Apex callouts by registering external endpoints (Setup > Remote Site Settings). Enter the remote URL and activate it so Apex HttpRequest calls succeed. If using Named Credentials, Remote Site Settings is not required; Named Credentials encapsulate the endpoint and optional auth for safer callouts.



Conclusion

These integration features form a robust toolkit for connecting Salesforce with external applications. Combine change streaming (CDC, Platform Events), secure authentication (Named Credentials, OAuth), and efficient data access (Salesforce Connect) to build scalable, secure, and maintainable integrations. Always account for API limits, secure credentials, and prefer declarative options when available.

Phase 8: Data Management & Deployment

Objective: In this phase, we focus on managing data efficiently and deploying customizations from one Salesforce environment to another. This ensures smooth migration, data consistency, and secure backups, forming the foundation for stable and scalable CRM operations.

1. Data Import Wizard

Purpose: Simplifies importing data such as Accounts, Contacts, Leads, or Custom Object records directly through Salesforce UI.

1 Navigate to Setup → Data Import Wizard → Launch Wizard.

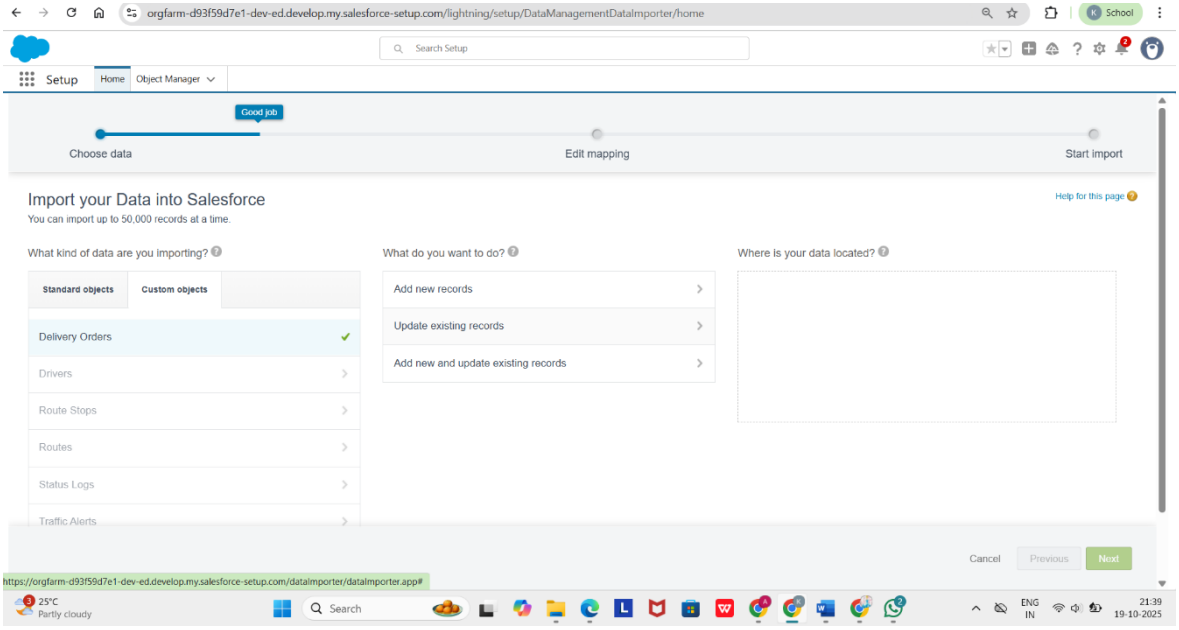
2 Choose the object (e.g., Account, Contact, or Custom Object).

3 Select the operation — Add new, Update existing, or Upsert.

4 Upload your CSV file and map fields correctly.

5 Run the import and review the success/error files.

Use Case: Ideal for smaller datasets under 50,000 records. Best used for quick imports where mapping and deduplication are simple. It is helpful when end-users or admins need to upload data without developer assistance.



2. Data Loader

Purpose: A client application for bulk import, export, update, or delete operations.

1 Download and install Salesforce Data Loader.

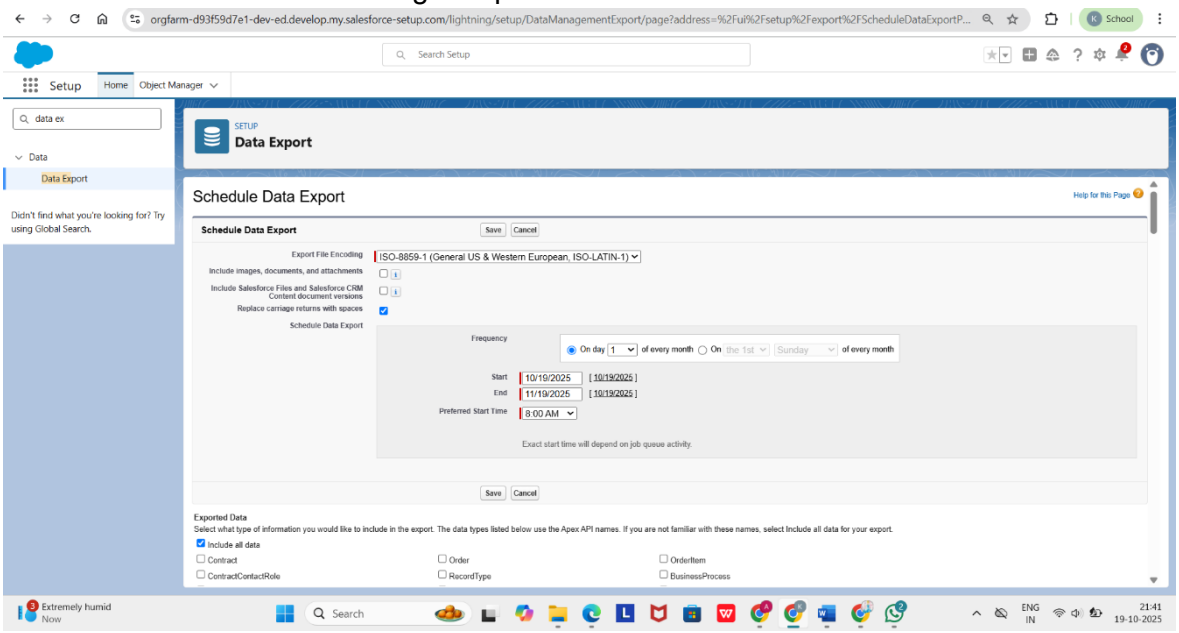
2 Login using Salesforce credentials or OAuth.

3 Choose the operation (Insert, Update, Upsert, Delete, Export).

4 Select the CSV file and map fields.

5 Run the operation and review success/error logs.

Use Case: Used when large volumes of data need to be migrated, such as initial data loads, mass updates, or regular data synchronization. It allows automation and provides better control over error handling and performance.



3. Duplicate & Matching Rules

Purpose: Prevent creation of duplicate records and maintain clean data.

1 Go to Setup → Matching Rules → New and define matching criteria (e.g., Email, Phone).

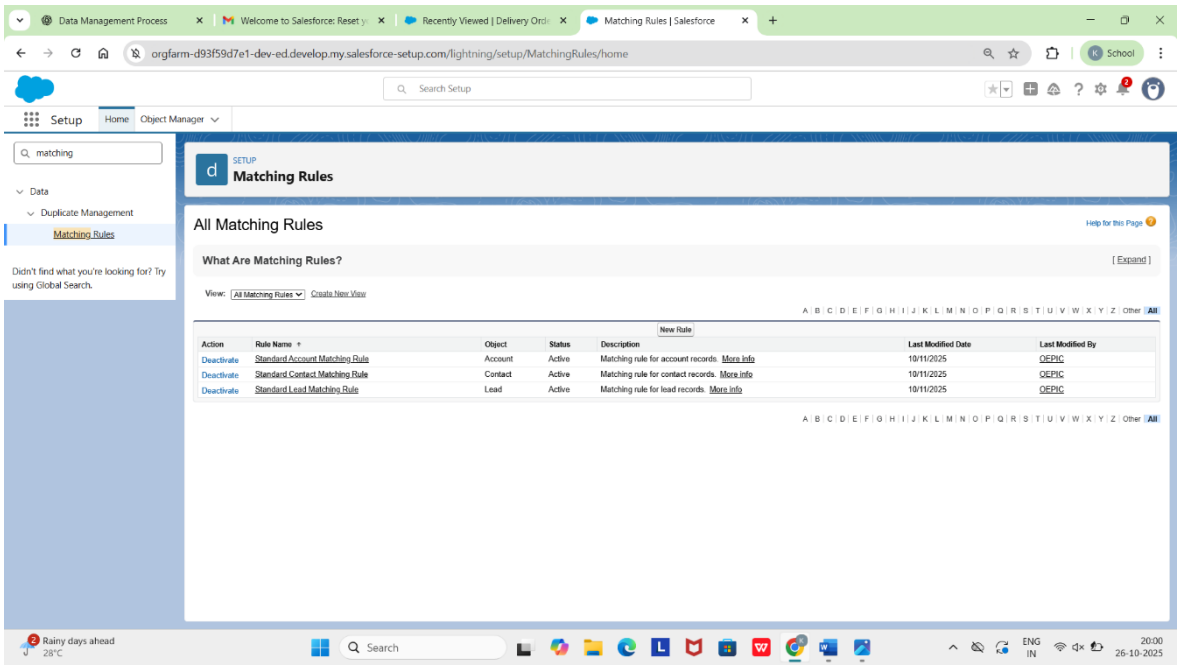
2 Create a Duplicate Rule using the matching rule and set behavior (Block, Alert, or Report).

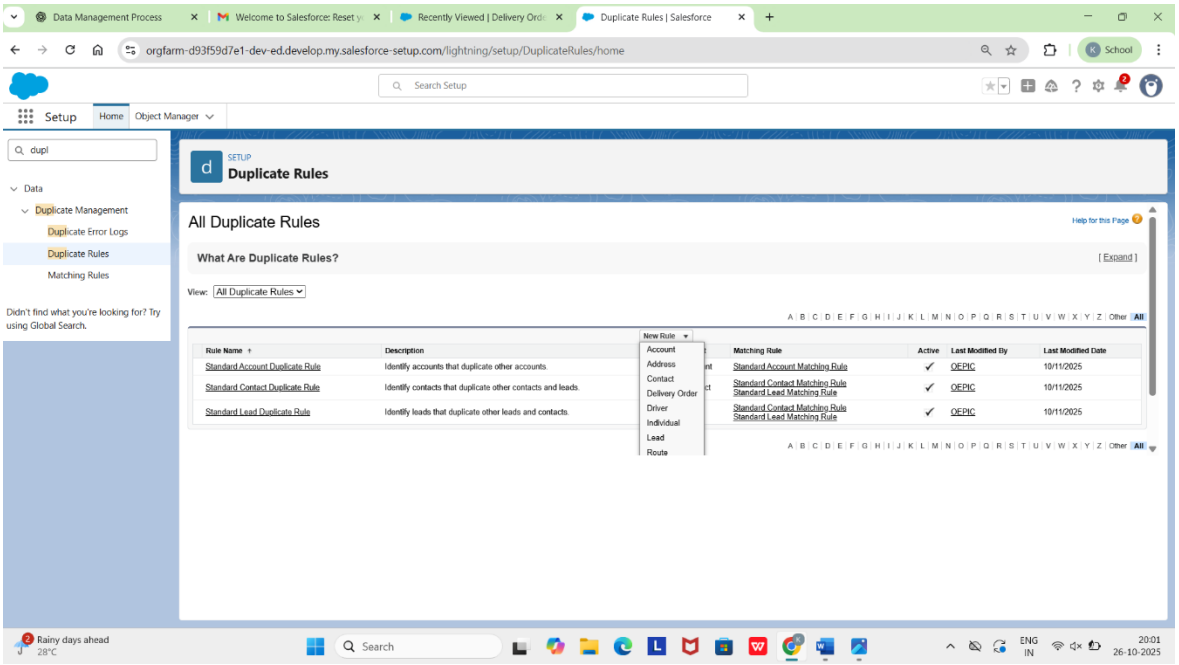
3 Activate both rules.

4 Test by creating records to ensure duplicates are flagged.

Use Case: Keeps CRM data clean by automatically identifying potential duplicates during record creation or update. This improves data integrity, reporting accuracy, and customer

satisfaction.





4. Data Export & Backup

Purpose: To ensure data safety and compliance by creating regular backups.

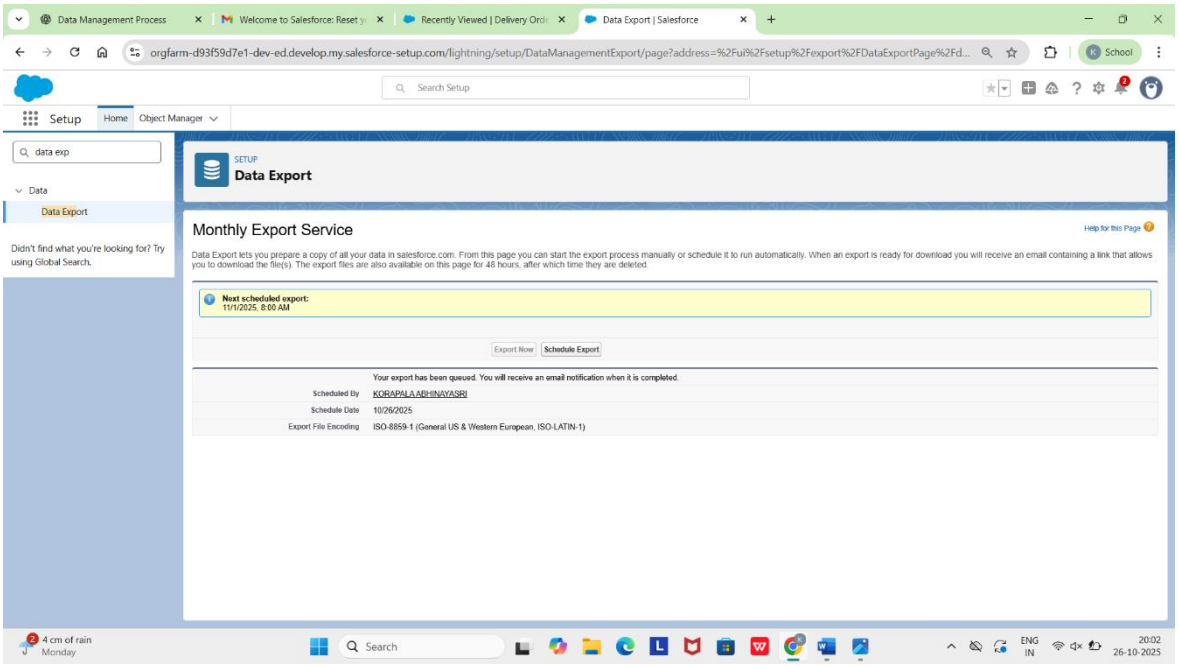
1 Navigate to Setup → Data Export.

2 Choose objects to include and decide whether to include attachments/files.

3 Click Export Now or Schedule Export for weekly/monthly backups.

4 Download the ZIP file once available.

Use Case: Ensures business continuity by maintaining secure backups of all Salesforce data. This process helps restore accidentally deleted records, recover from corruption, and meet data retention or compliance requirements.



5. Change Sets

Purpose: To deploy metadata (custom objects, triggers, layouts, etc.) between connected Salesforce orgs (e.g., Sandbox → Production).

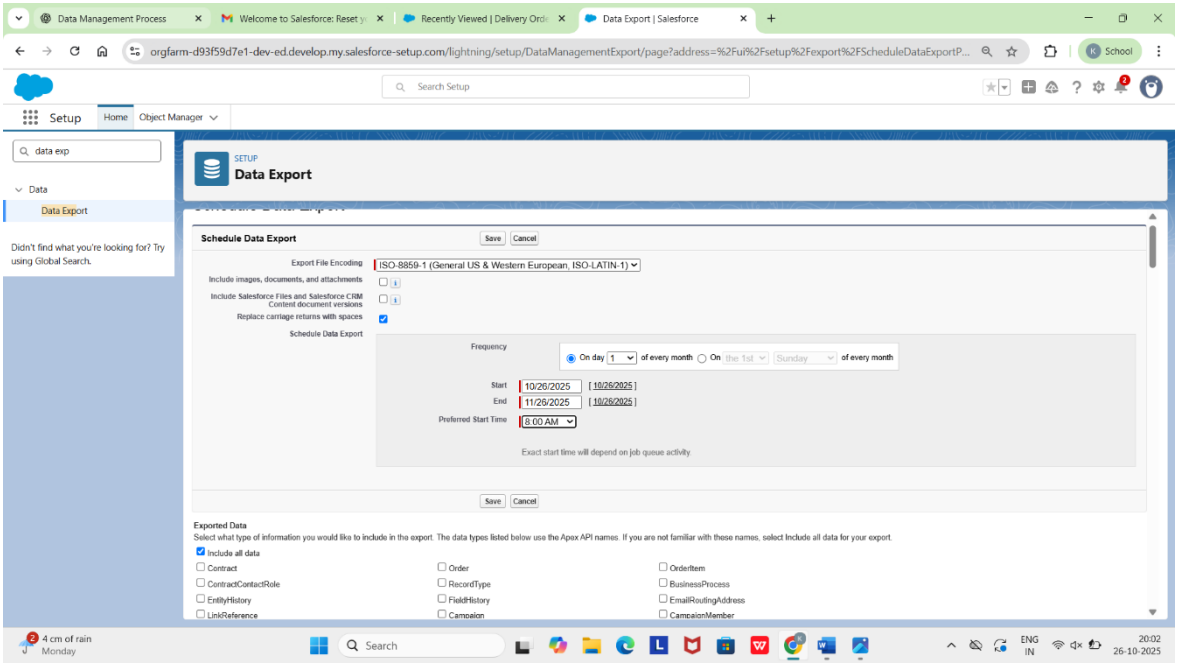
1 In Sandbox, go to Setup → Outbound Change Sets → New.

2 Add components (Apex Classes, Page Layouts, etc.).

3 Upload to target org.

4 In Production, go to Inbound Change Sets, validate, and deploy.

Use Case: Simplifies deployment between environments while maintaining control and traceability. It is ideal for admins and smaller teams who prefer a point-and-click migration method for tested configurations.



6. Managed vs Unmanaged Packages

Purpose: Packaging enables distribution and reusability of Salesforce components.

Use Case: Managed packages are used by developers and ISVs to distribute upgradable solutions through AppExchange. Unmanaged packages are used when sharing source code or configurations for learning, customization, or client-specific implementations.

7. ANT Migration Tool

Purpose: A Java-based command-line tool for retrieving and deploying metadata via XML manifests.

1 Install Java and Apache Ant.

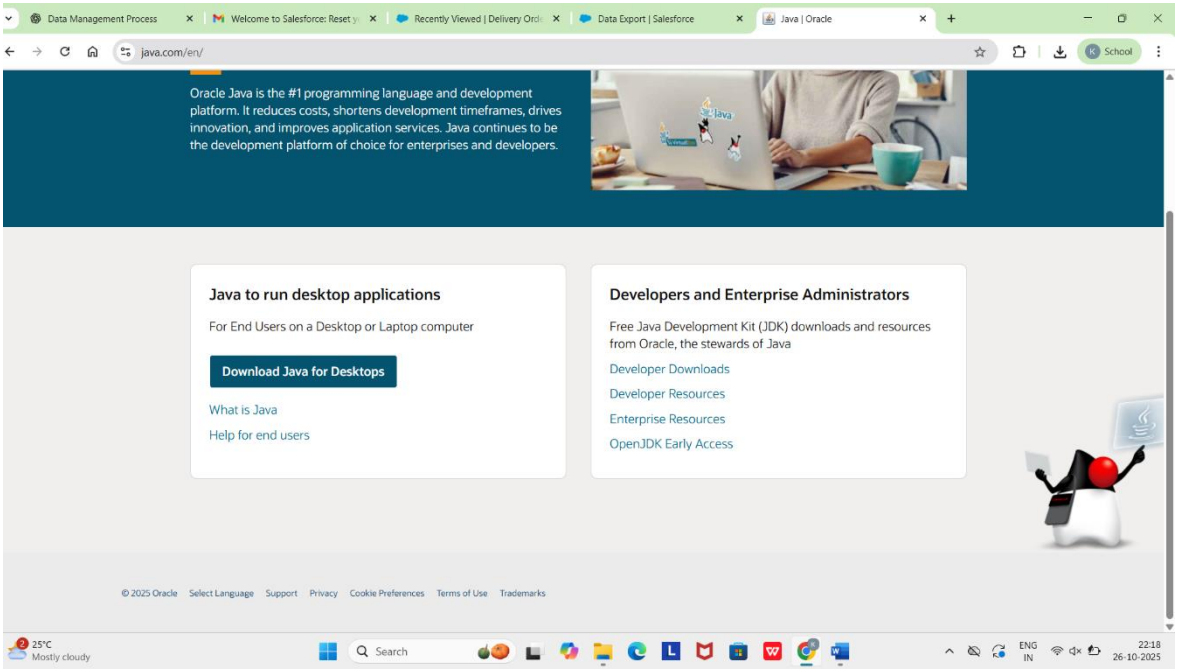
2 Download the Salesforce ANT Migration Tool.

3 Configure build.properties with org credentials.

4 Edit package.xml to define components.

5 Run commands: ant retrieve / ant deploy.

Use Case: Suitable for automated deployments where consistency and repeatability are required. It enables version control integration and bulk metadata transfers across multiple Salesforce environments.



8. Visual Studio Code (VS Code) & Salesforce CLI (SFDX)

Purpose: The modern development and deployment environment for Salesforce developers.

1 Install VS Code and Salesforce CLI (SFDX).

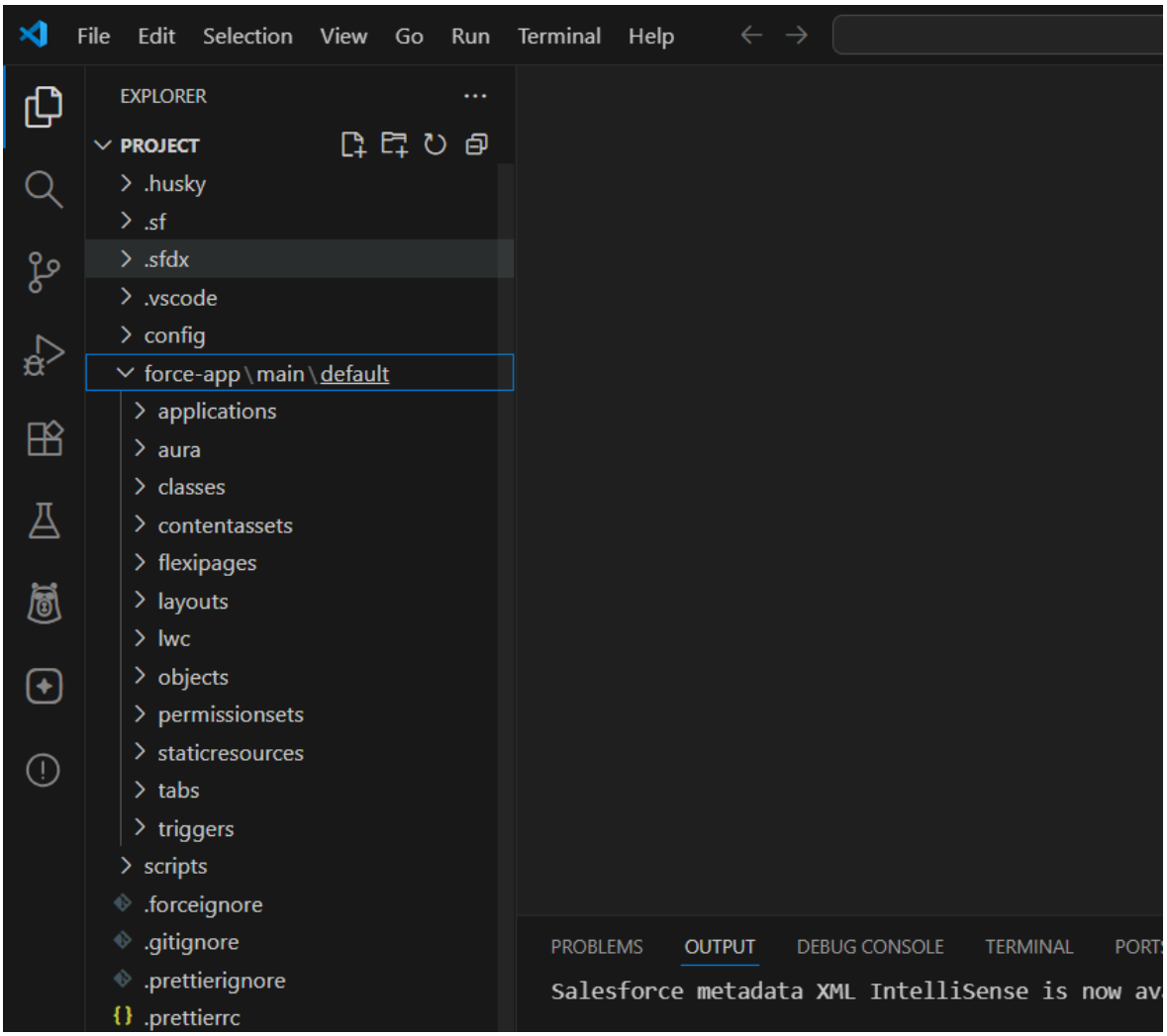
2 Create a new project using: sfdx force:project:create -n MyProject.

3 Authorize your org using: sfdx auth:web:login -a DevHubAlias.

4 Retrieve or deploy metadata using CLI commands.

5 Use Git for version control and CI/CD integrations.

Use Case: Provides a unified development platform supporting source tracking, testing, and deployment. Ideal for continuous integration (CI) and continuous delivery (CD) pipelines, improving collaboration among development teams.



Conclusion: Data Management & Deployment are vital for ensuring Salesforce environments remain synchronized, reliable, and secure. By leveraging tools like Data Loader, Change Sets, and SFDX, organizations can automate deployment, prevent errors, and maintain data integrity across all environments.

Salesforce Reports, Dashboards, and Security Review Document

1. Reports (Tabular, Summary, Matrix, Joined)

In this stage, we review and validate the various types of reports configured in the system: **Tabular, Summary, Matrix, and Joined**.

- **Tabular reports** provide flat lists—useful for delivery orders lists and data exports.

- **Summary reports** allow grouping and subtotals (e.g., grouping DeliveryOrders by city or driver).

- **Matrix reports** enable two-dimensional grouping (e.g., number of stops by vehicle by time window).

- **Joined reports** let you combine multiple report types into one (e.g., combining Route, Vehicle, and DeliveryOrder data).

During this phase we:

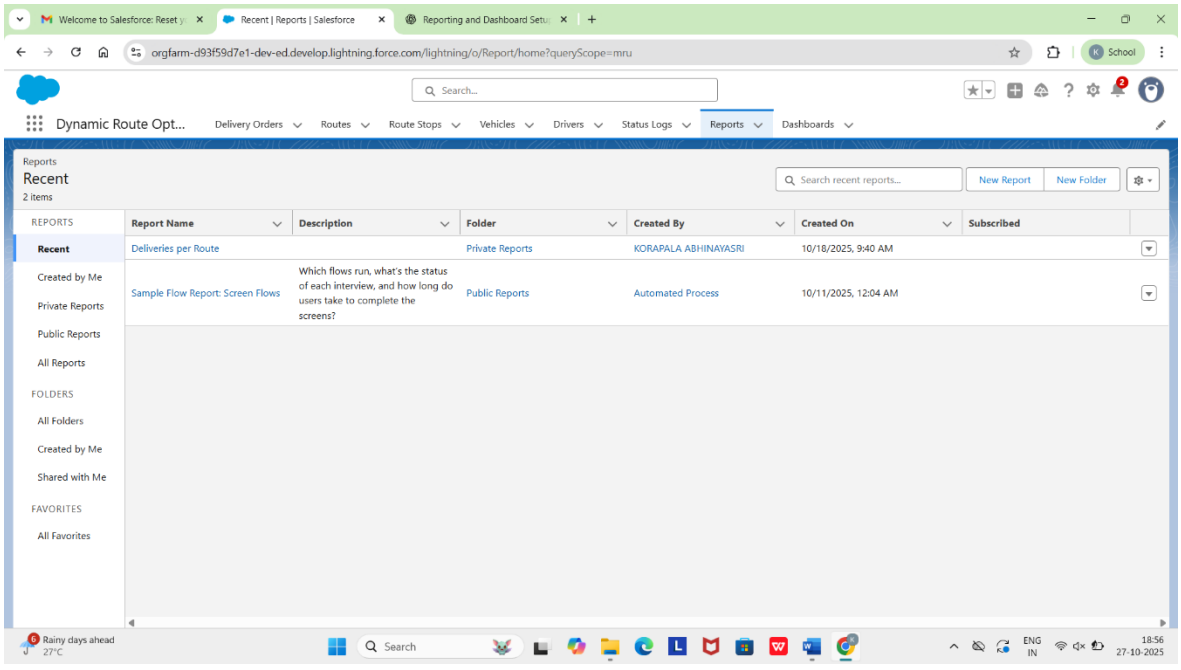
- Ensure each report type aligns with business metrics (e.g., on-time vs delayed orders).

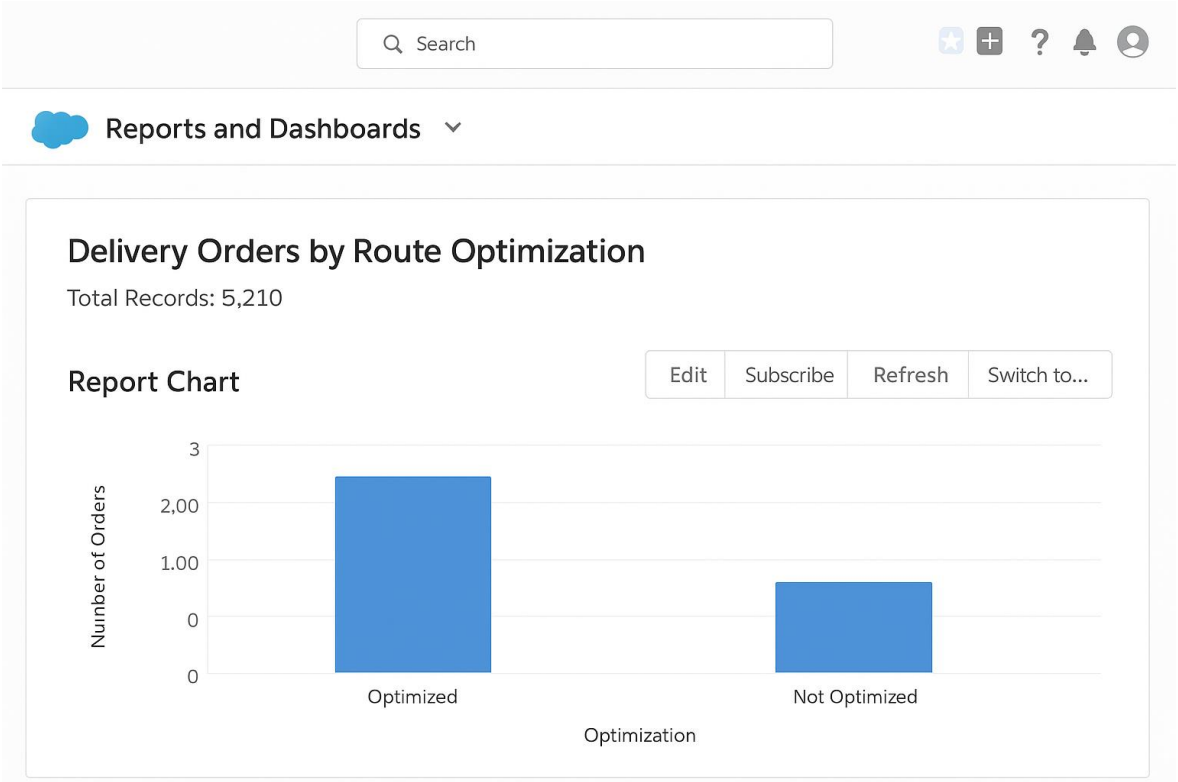
- Confirm filters, groupings, and charts display correctly for end-users.

- Optimize for performance and ensure correct fields/objects are included.

- Document and classify reports by use-case (Dispatcher, Manager, Executive).

-





-

2. Report Types

Report Types in Salesforce define which objects and fields are available for reporting.

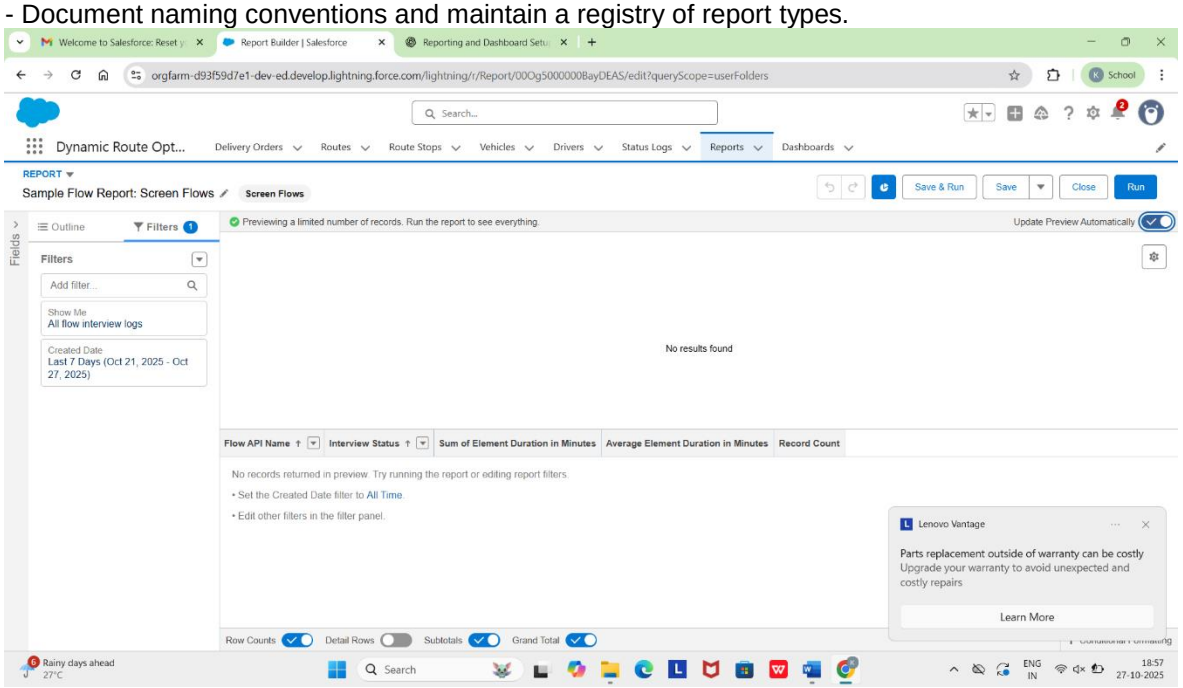
During this phase we:

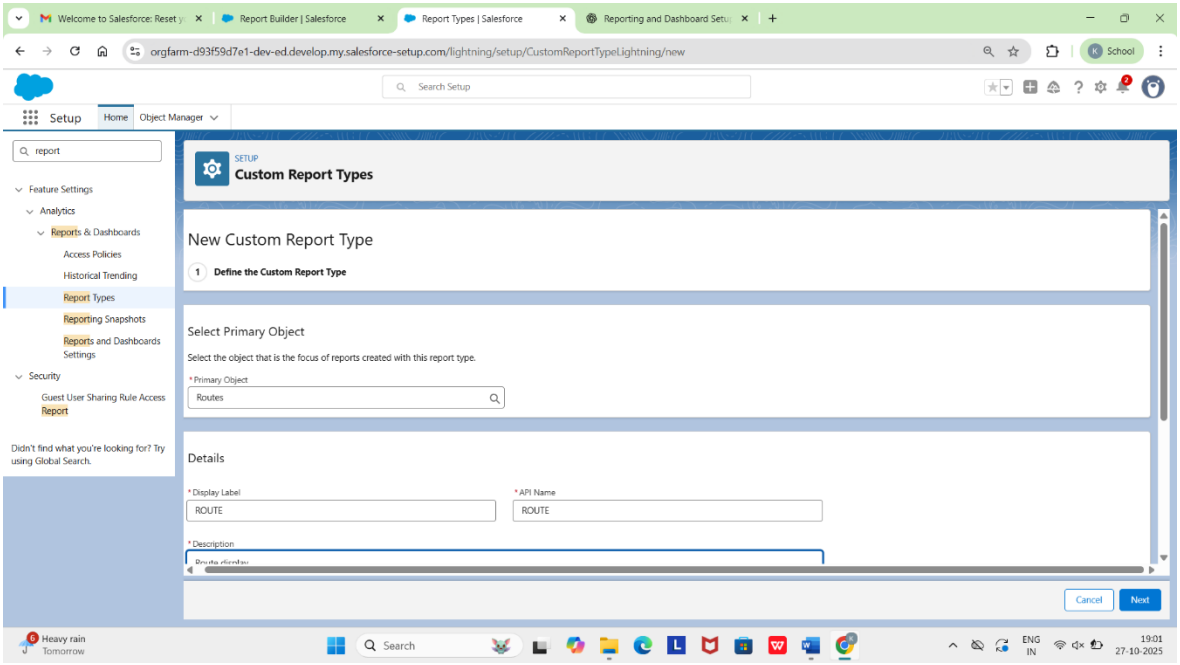
- Review existing report types and ensure they include custom objects (DeliveryOrder, RouteStop, Vehicle, etc.).

- Create or modify custom report types to support new use-cases (e.g., “Delivery Orders with Route Stops and Vehicle Assignment”).

- Test each report type for correct object relationships and field availability.

- Document naming conventions and maintain a registry of report types.





3. Dashboards

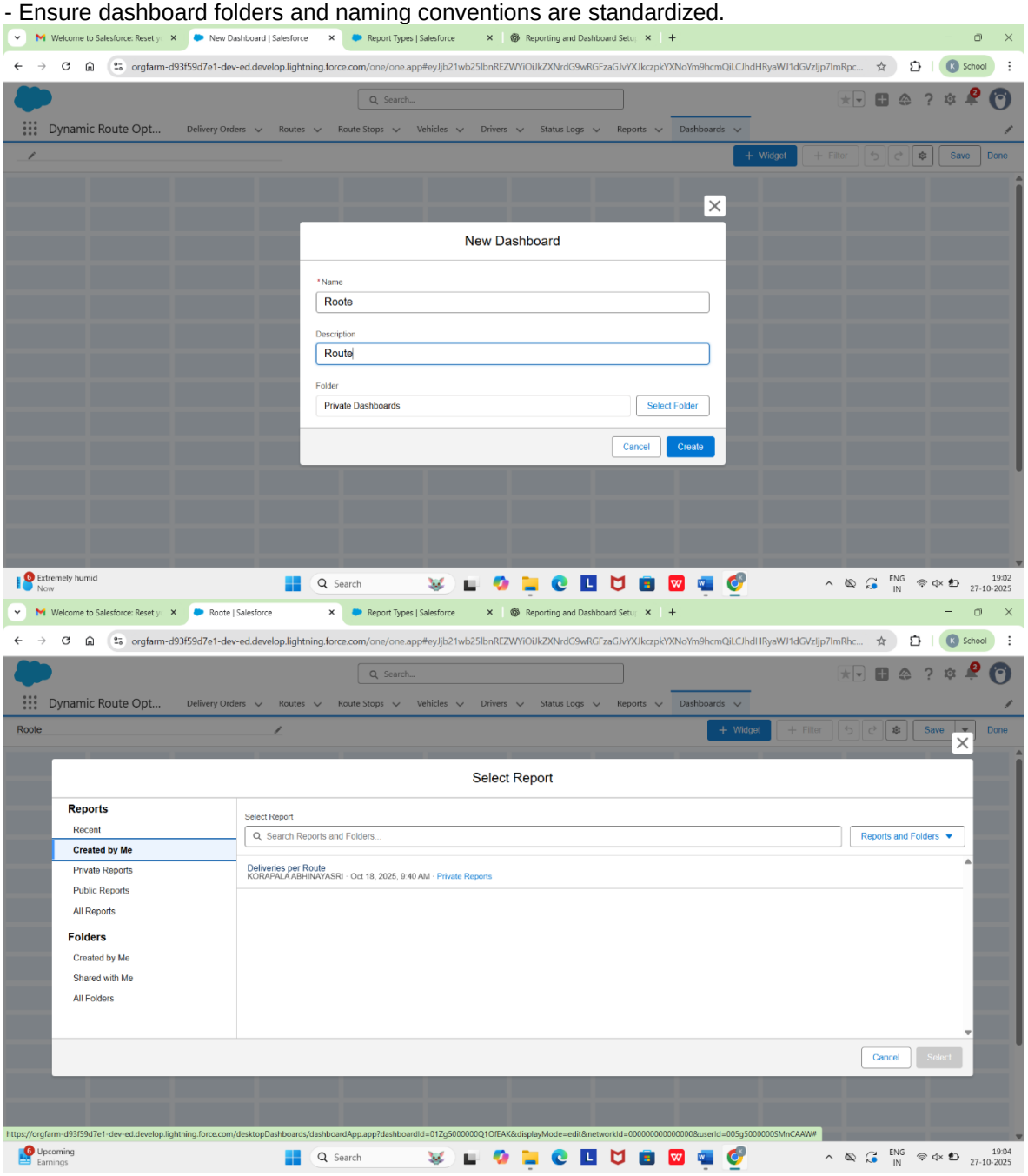
Dashboards visualize key performance indicators.

During this phase:

- Consolidate dashboard components (charts, gauges, tables) for routing performance metrics.
- Validate that components reflect accurate data.

- Check layout, color themes, and responsiveness.

- Ensure dashboard folders and naming conventions are standardized.



4. Dynamic Dashboards

Dynamic dashboards allow personalized views for users (drivers, dispatchers, managers).

During this phase:

- Identify dashboards that should be dynamic and set “Run as Logged-in User”.

- Verify user permissions and sharing settings.

- Review limits (e.g., Developer Edition supports 3 dynamic dashboards).

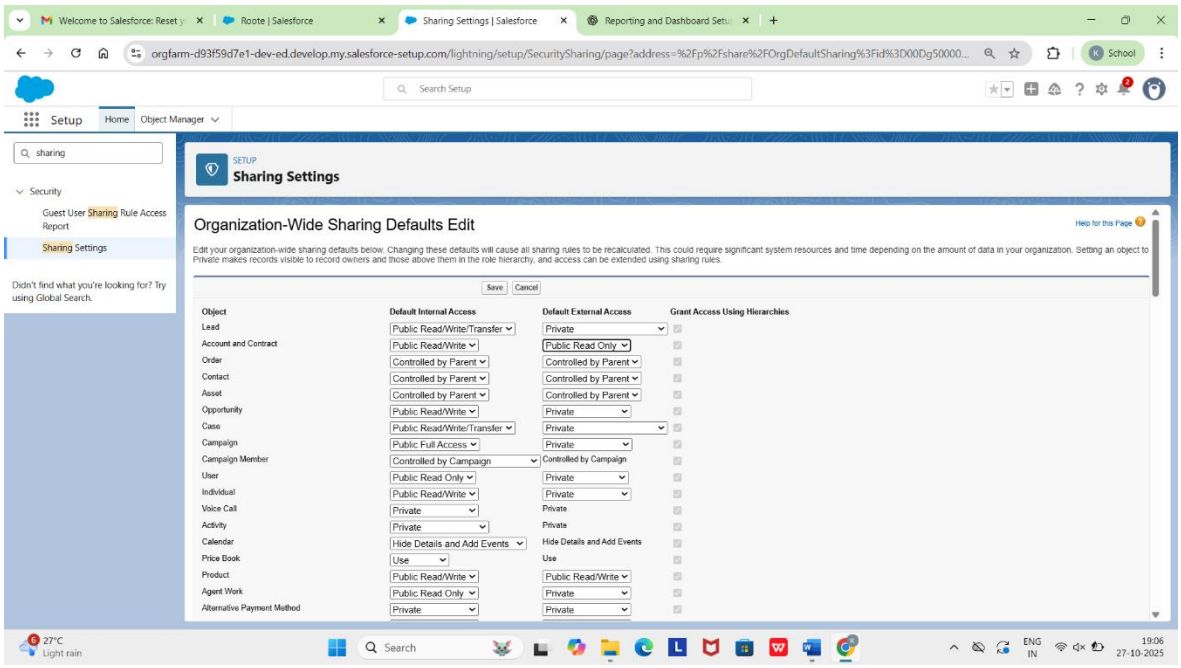
- Test across profiles to ensure correct data filtering.

5. Sharing Settings

Sharing settings control record visibility across users.

During this phase:

- Review OWD for custom objects (DeliveryOrder__c, Route__c, etc.).
- Validate Sharing Rules, Manual Sharing, and Apex Sharing.
- Ensure role hierarchy aligns with business needs.
- Test visibility for roles (driver, manager, operations head).

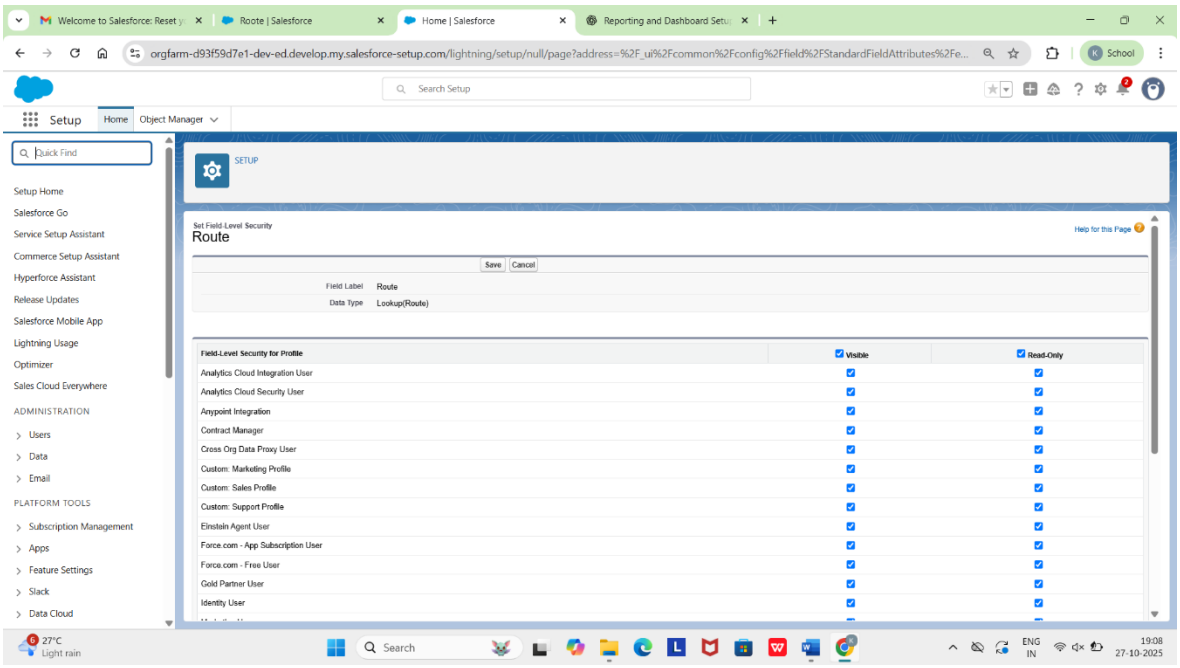


6. Field-Level Security

Defines who can view/edit specific fields.

During this phase:

- Hide or restrict sensitive fields (driver contact, vehicle timings).
- Validate mandatory fields are visible and required for relevant users.
- Review and adjust profiles/permission sets.
- Ensure compliance with data policies.



7. Session Settings

Controls login session security.

During this phase:

- Verify session timeout (e.g., 30 minutes for mobile users).

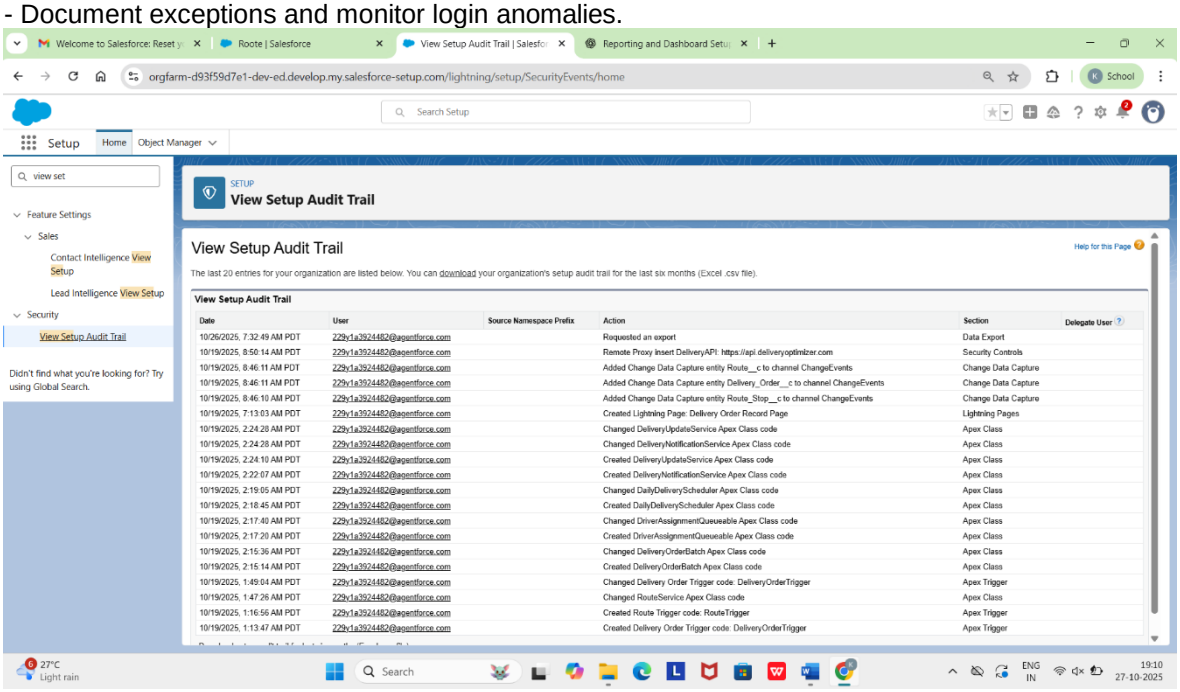
- Check IP restrictions for sensitive profiles.
- Confirm MFA and session-based permissions.
- Document exceptions and compensating controls.

8. Login IP Ranges

Defines trusted IP ranges for login.

During this phase:

- Review profiles with locked IP ranges.
- Validate restrictions for admin and operations profiles.
- Recommend VPN/secure network access.
- Document exceptions and monitor login anomalies.

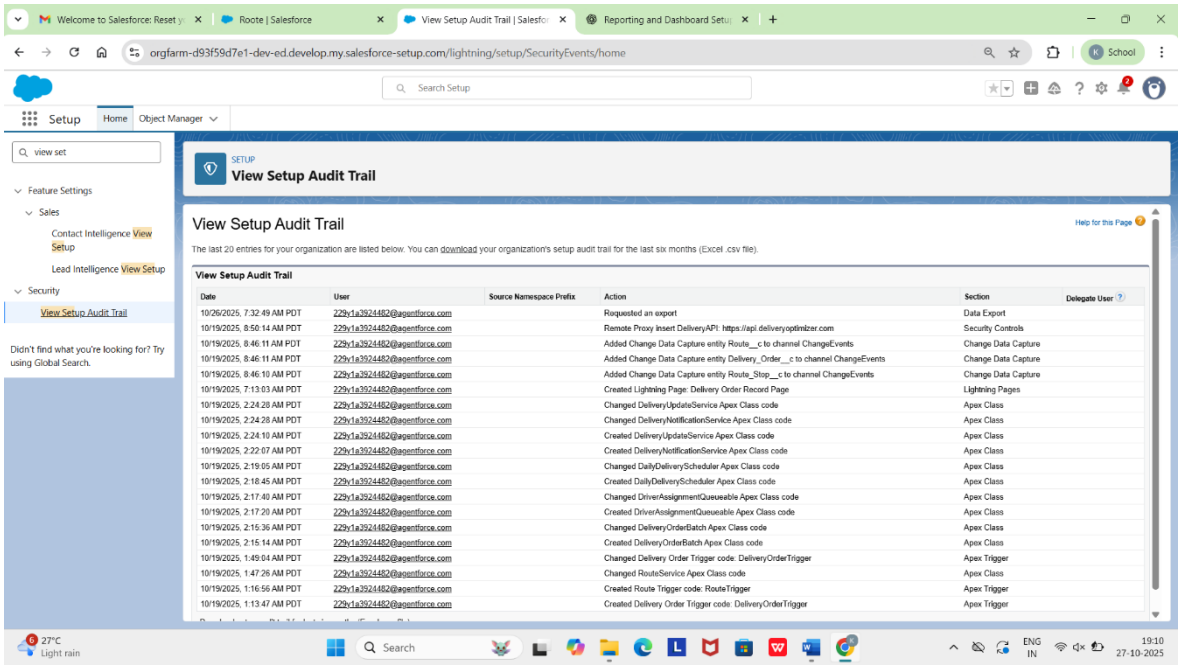


9. Audit Trail

Tracks metadata and data changes.

During this phase:

- Review 180-day Setup Audit Trail logs.
- Ensure Field History Tracking is enabled on key fields.
- Review privileged changes and validate approvals.
- Establish audit log governance and anomaly reporting.


Welcome to Salesforce: Reset y
Roote | Salesforce
View Setup Audit Trail | Salesfo
Reporting and Dashboard Setu
orgfarm-d93f59d7e1-dev-ed.develop.my.salesforce-setup.com/lightning/setup/SecurityEvents/home
School
Search Setup
Setup
Home
Object Manager
view set
Feature Settings
Sales
Contact Intelligence View Setup
Lead Intelligence View Setup
Security
View Setup Audit Trail
Didn't find what you're looking for? Try using Global Search.
SETUP
View Setup Audit Trail
View Setup Audit Trail
Help for this Page
The last 20 entries for your organization are listed below. You can download your organization's setup audit trail for the last six months (Excel .csv file).
View Setup Audit Trail
Date
User
Source Namespace Prefix
Action
Section
Delegate User
10/26/2025, 7:32:49 AM PDT
229y1a3924482@agentforce.com
Requested an export
Data Export
10/19/2025, 8:50:14 AM PDT
229y1a3924482@agentforce.com
Remote Proxy insert DeliveryAPI: https://api.deliveryoptimizer.com
Security Controls
10/19/2025, 8:46:11 AM PDT
229y1a3924482@agentforce.com
Added Change Data Capture entity Route__c to channel ChangeEvents
Change Data Capture
10/19/2025, 8:46:11 AM PDT
229y1a3924482@agentforce.com
Added Change Data Capture entity Delivery_Order__c to channel ChangeEvents
Change Data Capture
10/19/2025, 8:46:10 AM PDT
229y1a3924482@agentforce.com
Added Change Data Capture entity Route_Stop__c to channel ChangeEvents
Change Data Capture
10/19/2025, 7:13:03 AM PDT
229y1a3924482@agentforce.com
Created Lightning Page: Delivery Order Record Page
Lightning Pages
10/19/2025, 2:24:28 AM PDT
229y1a3924482@agentforce.com
Changed DeliveryUpdateService Apex Class code
Apex Class
10/19/2025, 2:24:28 AM PDT
229y1a3924482@agentforce.com
Changed DeliveryNotificationService Apex Class code
Apex Class
10/19/2025, 2:24:10 AM PDT
229y1a3924482@agentforce.com
Created DeliveryUpdateService Apex Class code
Apex Class
10/19/2025, 2:22:07 AM PDT
229y1a3924482@agentforce.com
Created DeliveryNotificationService Apex Class code
Apex Class
10/19/2025, 2:19:05 AM PDT
229y1a3924482@agentforce.com
Changed DailyDeliveryScheduler Apex Class code
Apex Class
10/19/2025, 2:18:45 AM PDT
229y1a3924482@agentforce.com
Created DailyDeliveryScheduler Apex Class code
Apex Class
10/19/2025, 2:17:40 AM PDT
229y1a3924482@agentforce.com
Changed DriverAssignmentQueueable Apex Class code
Apex Class
10/19/2025, 2:17:20 AM PDT
229y1a3924482@agentforce.com
Created DriverAssignmentQueueable Apex Class code
Apex Class
10/19/2025, 2:15:36 AM PDT
229y1a3924482@agentforce.com
Changed DeliveryOrderBatch Apex Class code
Apex Class
10/19/2025, 2:15:14 AM PDT
229y1a3924482@agentforce.com
Created DeliveryOrderBatch Apex Class code
Apex Class
10/19/2025, 1:49:04 AM PDT
229y1a3924482@agentforce.com
Changed Delivery Order Trigger code: DeliveryOrderTrigger
Apex Trigger
10/19/2025, 1:47:26 AM PDT
229y1a3924482@agentforce.com
Changed RouteService Apex Class code
Apex Class
10/19/2025, 1:16:56 AM PDT
229y1a3924482@agentforce.com
Created Route Trigger code: RouteTrigger
Apex Trigger
10/19/2025, 1:13:47 AM PDT
229y1a3924482@agentforce.com
Created Delivery Order Trigger code: DeliveryOrderTrigger
Apex Trigger
27°C
Light rain
Search
ENG
IN
19:10
27-10-2025