

Project Title: Restaurant CRM (Bookings + Orders + Loyalty)

Industry: Food & Beverage / Hospitality

Project Type: B2C Salesforce CRM Implementation

Target Users: Restaurant Managers, Staff (Booking & Order team), Customers, Marketing Team

Problem Statement

A restaurant chain receives numerous bookings and food orders daily via phone, website, and in-person visits. However:

- Bookings are tracked manually in registers or spreadsheets
- Duplicate or overlapping reservations often lead to poor customer experience
- Orders are not centralized, making it hard to link them with customers
- Loyalty programs are inconsistent and manually managed
- No real-time insights into top customers, peak booking hours, or revenue trends

■ To address these challenges, the restaurant wants to implement a **Salesforce CRM solution** to:

- Automate table bookings and send instant confirmation emails/SMS
- Manage customer orders and link them to accounts/contacts
- Track loyalty points and reward repeat customers
- Generate real-time reports for bookings, orders, and customer trends
- Improve customer communication through automated notifications

Use Cases

Booking Management

- Allow staff to create table bookings (date, time, party size, table number)
- Prevent double-booking using validation rules
- Send automatic confirmation/cancellation emails to customers

Order Management

- Record dine-in and take-away orders
- Link orders to customers (new or repeat)
- Track order details (menu items, price, quantity, bill amount)

Customer Management

- Maintain profiles with contact details, visit frequency, and order history
- Categorize customers (new, regular, VIP)
- Auto-assign loyalty program tiers based on spend

Loyalty Points Program

- Assign points per order (e.g., 1 point for every ■100 spent)
- Track accumulated points and redemption history
- Auto-notify customers of available rewards

Reporting & Dashboards

- Top 10 loyal customers by spend
- Peak booking hours and busiest days
- Revenue from dine-in vs. take-away
- Order trends per menu category

RESTAURANT CRM

Company Information

This is the foundation of any Salesforce organization. The *Company Information* section stores the official details of the business such as the company name, address, time zone, default currency, and fiscal year.

- **Organization Name:** Defines how the system recognizes your company and how it will appear in system headers, licenses, and reports. For example, *Restaurant CRM Pvt Ltd*.
- **Default Time Zone:** Ensures that all date/time stamps, workflows, and automation rules work according to the company's working hours. For India, we use (GMT+05:30) India Standard Time.
- **Default Currency:** Determines the default currency for opportunities, reports, and transactions. Businesses that operate in multiple regions can enable **multi-currency**.
- **Fiscal Year:** Critical for reporting revenue and performance. Salesforce supports both **Standard Fiscal Years** (Jan–Dec) and **Custom Fiscal Years** (e.g., Apr–Mar for Indian businesses).

The screenshot shows the Salesforce Company Information page. At the top, there's a navigation bar with tabs like Setup, Home, Object Manager, and a search bar. Below that is a sidebar with sections for Company Settings (Business Hours, Calendar Settings, Public Calendars and Resources), Company Information (Data Protection and Privacy, Fiscal Year, Holidays, Language Settings, My Domain), and a Global Search bar. The main content area is titled "Company Information" and shows the profile for "MyRestaurant Pvt Ltd". It includes fields for Organization Name (MyRestaurant Pvt Ltd), Primary Contact (OrgFarm EPIC), Division (India), Address (India), Fiscal Year Starts In (January), and various checkboxes for Newsletter, Admin Newsletter, and system notices. On the right, there are sections for Phone, Fax, Default Locale (English (India)), Default Language (English), Default Time Zone ((GMT+05:30) India Standard Time (Asia/Kolkata)), Currency Locale (English (United States) - USD), Used Data Space (342 KB (7%)), Used File Space (17 KB (0%)), API Requests, Last 24 Hours (0 (15,000 max)), Streaming API Events, Last 24 Hours (0 (10,000 max)), Restricted Logins, Current Month (0 (max)), Salesforce.com Organization ID (0CgI.00000C0ckj), Organization Edition (Developer Edition), and Instance (CAN98). At the bottom, it shows the page was created by OrgFarm EPIC on 9/10/2025 at 12:17 PM and modified by PAIPETI PALLAVI on 9/25/2025 at 3:45 AM. The status bar at the bottom indicates it's a sunny day with 33°C, and the system status shows ENG IN 04:16 PM 25-09-2025.

2. Business Hours

Defining *Business Hours* in Salesforce is vital because many service-related features, like **case escalations**, **SLA timers**, and **workflow triggers**, depend on them.

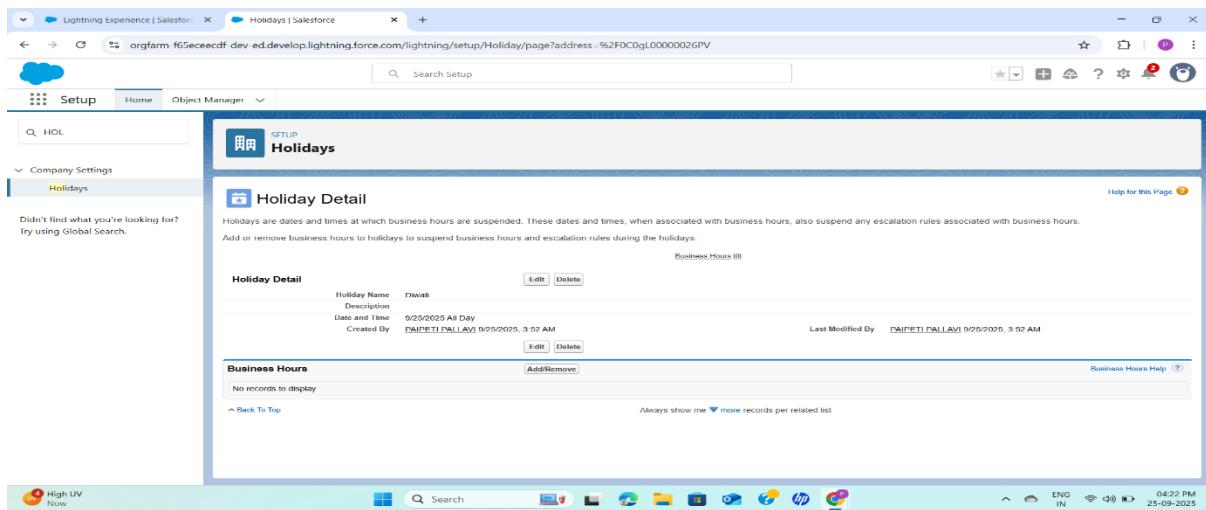
- Example: The restaurant operates from **11:00 AM to 11:00 PM** every day.
- Marking them as **default** ensures they apply across customer support cases and automation flows unless overridden.
- Having clear business hours helps track SLA commitments more accurately.

The screenshot shows the Salesforce Business Hours setup page. At the top, there are two tabs: 'Lightning Experience | Salesforce' and 'Business Hours | Salesforce'. The URL in the address bar is 'orgfarm-f65eceecdf-dev-ed.develop.lightning.force.com/lightning/setup/BusinessHours/page?address=%2F01mgL0000043mVd'. The page title is 'Business Hours'. On the left, there's a sidebar with 'Company Settings' and 'Business Hours' selected. A search bar at the top says 'Search Setup' and has 'busin' typed into it. Below the sidebar, a message says 'Didn't find what you're looking for? Try using Global Search.' The main content area is titled 'Organization Business Hours' and contains a table of 'Business Hours Detail'. The table shows 'Restaurant Working Hours' for each day of the week, with specific times listed. There are columns for 'Business Hours Name' (which is 'Restaurant Working Hours'), 'Restaurant Working Hours' (with a table of daily times), 'Time Zone' (set to '(GMT+05:30) India Standard Time (Asia/Kolkata)'), and 'Default Business Hours' (checkbox). Below the table, it says 'Active' with a checkbox, 'Created By' (PAIPETI PALLAVI), and 'Last Modified By' (PAIPETI PALLAVI). At the bottom, there's a section for 'Holidays' with a link to 'Add/Remove' and a note 'No records to display'. The status bar at the bottom shows 'High UV Now' and other system information like time and date.

3. Holidays

Holidays allow you to define non-working days in Salesforce. These are linked to business hours to ensure SLA calculations and workflows don't count those days.

- Example: *Diwali – Nov 12, 2025.*
- When a holiday is marked, Salesforce automatically pauses SLA timers and escalations on that day.
- This ensures accurate reporting and avoids penalizing the support team during holidays.

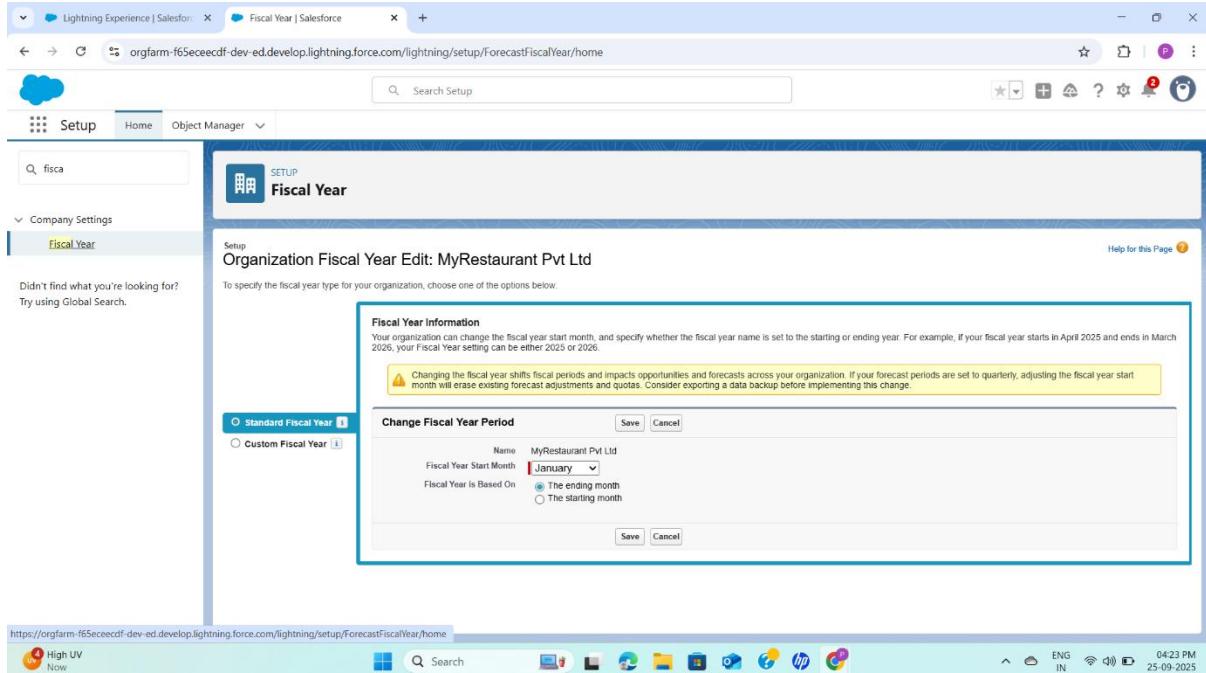


4. Fiscal Year

The *Fiscal Year* setup directly affects reporting, forecasting, and dashboards.

- **Standard Fiscal Year:** Runs Jan–Dec. Recommended for global organizations unless specified otherwise.
- **Custom Fiscal Year:** Can follow any 12-month pattern (e.g., Apr–Mar for Indian businesses). This is useful for aligning Salesforce reporting with statutory accounting requirements.

Once a fiscal year type is enabled, it affects all company reports, so the decision must be made carefully.



5. Profiles

Profiles are the backbone of Salesforce's **security model**. They define **object permissions**, **field-level access**, **page layouts**, and **app visibility**.

- **Manager Profile** → Full CRUD (Create, Read, Update, Delete) on Orders, Bookings, and Loyalty objects.
- **Host Profile** → Can create and manage bookings/orders but has limited edit access.
- **Kitchen Profile** → Read-only access to orders and related order items.

The screenshot shows the Salesforce Setup interface for managing Profiles. The left sidebar navigation includes: Salesforce Mobile App, Lightning Usage, Optimizer, Sales Cloud Everywhere, ADMINISTRATION, Users, Permission Set Groups, Permission Sets, Profiles (selected), Public Groups, Queues, Roles, User Management Settings, Users, Data, Email, PLATFORM TOOLS, Subscription Management, and Apps. The main content area is titled "SETUP Profiles". It displays "Custom Object Permissions" for various objects like Bookings, Loyalty Accounts, and Menu Items, with columns for Basic Access (Read, Create, Edit, Delete) and Data Administration (View All Records, Modify All Records, View All Fields). It also shows "Session Settings" with a session timeout of 2 hours of inactivity and a required login security level. At the bottom, a status bar indicates "Profile: Manager – Salesforce - Developer Edition", the date "25-09-2025", and the time "04:32 PM".

6. Permission Sets

Permission Sets allow granting **additional access on top of profiles** without changing the base profile.

- Example: *Order_Manager* permission set → Grants full access (Read, Create, Edit, Delete) on the *Order_c* object.
- Assignable to any user when temporary or extended access is required.
- This provides flexibility in managing security without cloning multiple profiles.

The screenshot shows the Salesforce Setup interface under the 'Permission Sets' section. The 'Order Manager' permission set is selected. The page displays various sections: 'Tab Settings' (Available tab is 'Visible'), 'Record Type Assignments' (Dine-In record type assigned), 'Object Permissions' (listing permissions like Read, Create, Edit, Delete, View All Records, Modify All Records, View All Fields for standard objects like Account, Contact, Opportunity, etc.), and 'Field Permissions' (listing field access for fields like AccountId, AccountNumber, ActivatedById, ActivatedDate). The status bar at the bottom indicates it's 04:40 PM on 25-09-2025.

7. Users

Users represent real people who log in to Salesforce. Each user must be assigned a **profile** and optionally **permission sets**.

- Example: *Manager User* with Manager Profile.
- Each user consumes a license, which defines the type of functionality they can access (e.g., Sales Cloud, Service Cloud, or Platform license).

Proper user setup ensures accountability and controlled access to the system.

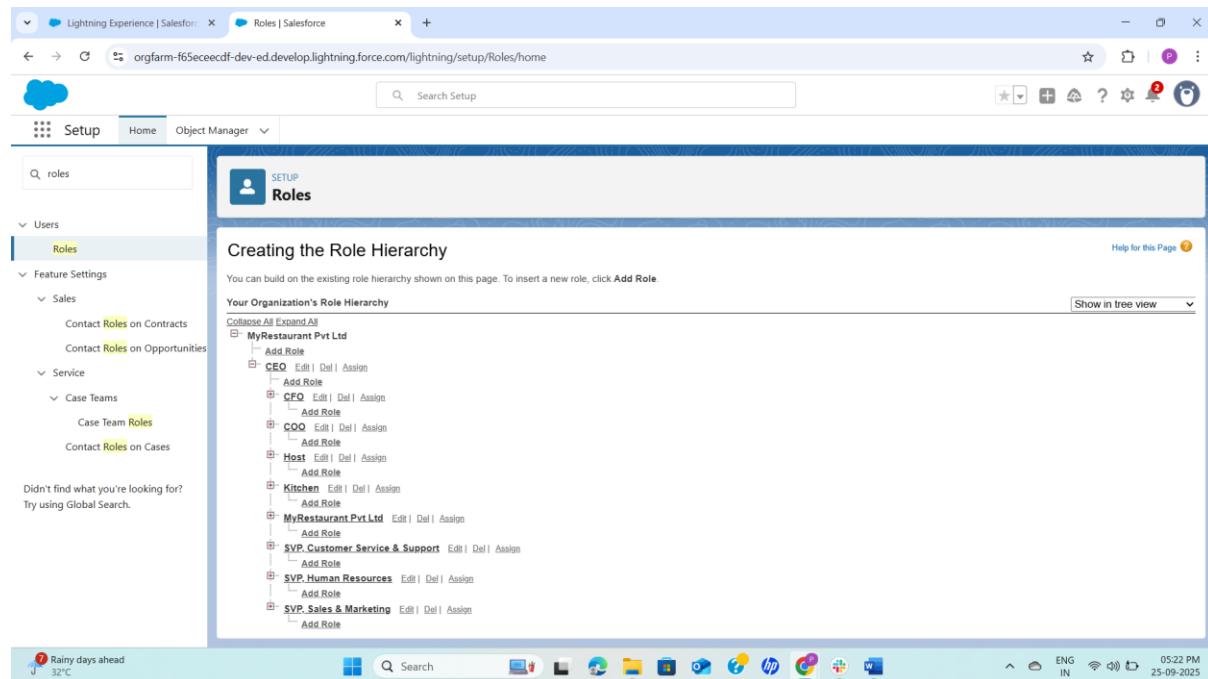
The screenshot shows the Salesforce Setup interface under the 'Users' section. A user named 'Manager' is selected. The 'User Detail' section shows details like Name (Manager), Alias (man), Email (manager@gmail.com), Username (manager@salesforce.com), and Title (User15678900244174545). The 'Profile' section shows the 'Standard Platform User' profile is active. The 'Licenses' section shows the 'Salesforce Platform' license is assigned. The status bar at the bottom indicates it's 04:43 PM on 25-09-2025.

8. Role Hierarchy

Roles control **record visibility** and define how data rolls up in the organization.

- Top role: **Restaurant Manager** → Can view and manage all subordinate data.
- Under: **Host** → Handles bookings.
- Under: **Kitchen** → Sees order details only.

The role hierarchy supports Salesforce's "**record sharing up the hierarchy**" rule, meaning managers can automatically see the records owned by their subordinates.



9. Session Settings

Session settings define the **security rules** for how long users remain logged in and how sessions are handled.

- **Force re-login after logout:** Prevents unauthorized re-entry.
- **Timeout (2 hours):** Ensures inactive sessions automatically log out for security.
- **Lock sessions to IP address:** Optional, prevents session hijacking but may inconvenience mobile/remote workers.

This setup balances **security with usability**.

The screenshot shows the 'Session Settings' page in the Salesforce Setup. The left sidebar has sections for Einstein (Einstein Assessors, Einstein Conversation Insights Assessor, Sales Cloud Einstein Assessor, Service Cloud Einstein Assessor) and Security (Session Management, Session Settings). The main content area is titled 'Session Settings' and describes setting session security and expiration timeout. It includes a 'Session Timeout' section with a dropdown set to '2 hours' and checkboxes for 'Disable session timeout warning popup' (unchecked) and 'Force logout on session timeout' (checked). Below this is a 'Session Settings' section with several checkboxes: 'Lock sessions to the IP address from which they originated' (unchecked), 'Lock sessions to the domain in which they were first used' (checked), 'Terminate all of a user's sessions when an admin resets that user's password' (unchecked), 'Force relogin after Login-As-User' (checked), 'Require HttpOnly attribute' (unchecked), 'Use POST requests for cross-domain sessions' (unchecked), 'Enforce login IP ranges on every request' (unchecked), and 'When embedding a Lightning application in a third-party site, use a session token instead of a session cookie' (unchecked). A note at the bottom states: "EXTENDED USE OF IE11 WITH LIGHTNING EXPERIENCE HAS NOW ENDED" and "AS OF DECEMBER 31, THE EXTENDED PERIOD HAS ENDED, AND USE OF INTERNET EXPLORER 11 (IE 11) WITH LIGHTNING EXPERIENCE IS NO LONGER SUPPORTED. ISSUES WITH PERFORMANCE OR FUNCTIONALITY THAT AFFECT ONLY IE 11 WILL NOT BE FIXED. PLEASE SWITCH TO A SUPPORTED BROWSER." The system status bar at the bottom shows 'Light rain Tomorrow', the date '25-09-2025', and the time '04:49 PM'.

10. Login Access Policies

This feature allows admins to log in as other users (with permission) for **troubleshooting, testing, and training**.

- Example: The admin can log in as a Kitchen User to verify if permissions and page layouts are working correctly.
- This is critical during testing before going live.

The screenshot shows the 'Login Access Policies' page in the Salesforce Setup. The left sidebar has sections for Identity (Login Flows, Login History) and Security (Login Access Policies). The main content area is titled 'Login Access Policies' and controls which support organizations users can grant login access to. It includes a 'Manage Support Options' section with a 'Save' and 'Cancel' button. Under 'Setting', it says 'Administrators Can Log in as Any User' with an unchecked checkbox. Under 'Support Organization', there is a table:

Support Organization	Packages	Available to Users	Available to Administrators Only
Salesforce.com Support	(radio button)	(radio button)	(radio button)

At the bottom is a 'Save' and 'Cancel' button. The system status bar at the bottom shows 'Light rain Tomorrow', the date '25-09-2025', and the time '04:50 PM'.

Phase 3 — Data Modelling & Relationships

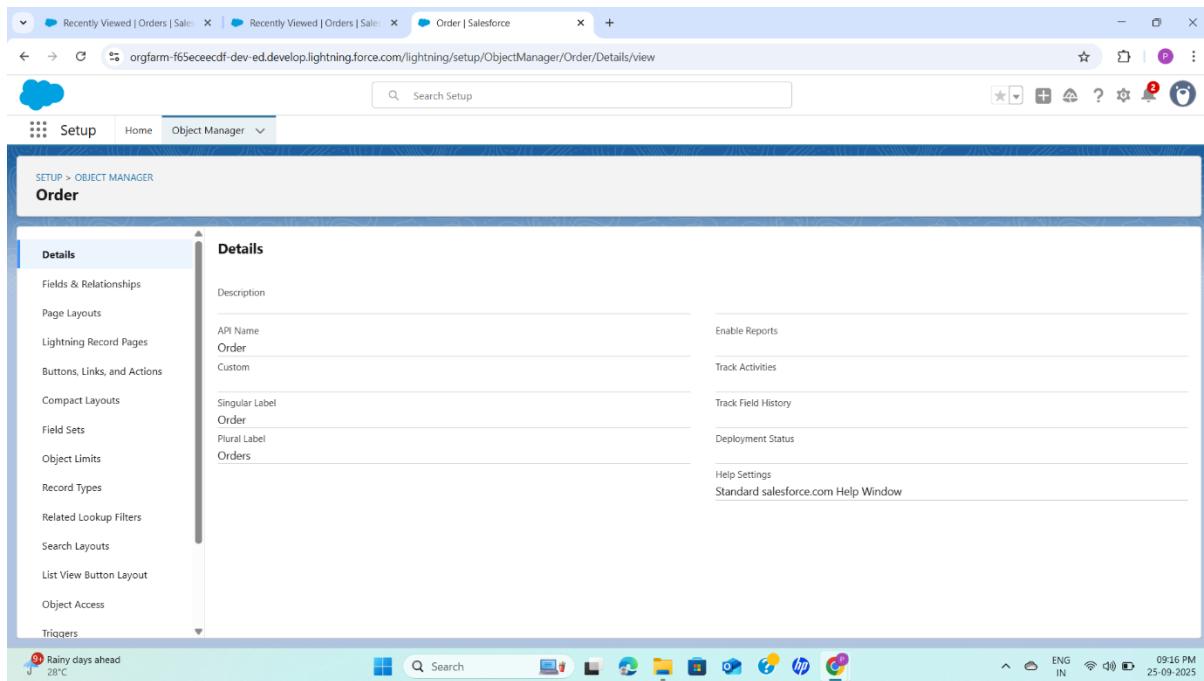
In Salesforce, data modelling is the process of defining **objects, fields, and relationships** so that business processes can be represented properly in the system. For a restaurant CRM, the core entities include Orders, Order Items, Bookings, and Loyalty Accounts.

Custom Objects

Custom objects are created to represent real-world entities. In this case:

- **Order** → Captures details of a customer's order.

These objects are configured with reporting, activities, and field history tracking enabled so data can be monitored and analysed.



The **Order object** captures all information about a customer's food order. It acts as the central business record for sales transactions in the restaurant. Each order may be tied to a **customer (Contact)**, a **Booking** (if it was pre-reserved), and contains multiple **Order Items** (individual dishes).

- The **Order Type** field differentiates whether the order is Dine-In, Takeaway, or Delivery.
- The **Status** field shows the order lifecycle (e.g., Draft → Placed → Preparing → Served → Paid → Cancelled).
- **Payment Status** ensures financial tracking, while **Total Amount** provides billing details.
- If delivery is chosen, an address is also stored.

Key Relationship:

Orders are **parents** to Order Items in a Master-Detail relationship, ensuring that if the order is deleted, all its line items are removed. This also allows roll-up summaries of order totals.

The screenshot shows the Salesforce Object Manager interface for the 'Order' object. On the left, a sidebar lists various setup options like Page Layouts, Lightning Record Pages, Buttons, etc. The main area is titled 'Fields & Relationships' and displays a table of fields. The table has columns for FIELD LABEL, FIELD NAME, DATA TYPE, CONTROLLING FIELD, and INDEXED. Key fields shown include Account Name (AccountId, Lookup(Account)), Account Number (AccountNumber, Text(40)), Activated By (ActivatedById, Lookup(User)), Activated Date (ActivatedDate, Date/Time), Bill To Contact (BillToContactId, Lookup(Contact)), Billing Address (BillingAddress, Address), Company Authorized By (CompanyAuthorizedById, Lookup(User)), Company Authorized Date (CompanyAuthorizedDate, Date), Contact (Customer__c, Lookup(Contact)), Contract End Date (ContractEndDate, Date), and Contract Name (ContractName, Text(80)). A status bar at the bottom indicates '28°C Mostly clear' and the date '25-09-2025'.

This screenshot is identical to the one above, showing the 'Fields & Relationships' section for the 'Order' object. However, the 'Total Amount' field is highlighted with a red box. This field is a Roll-Up Summary (SUM Order Item). The rest of the table and interface elements are the same as the first screenshot.

Booking → Stores reservations made by customers.

The screenshot shows the 'Object Manager' interface in Salesforce. The 'Booking' object is selected. On the left, a sidebar lists various configuration tabs: Details, Fields & Relationships, Page Layouts, Lightning Record Pages, Buttons, Links, and Actions, Compact Layouts, Field Sets, Object Limits, Record Types, Related Lookup Filters, Restriction Rules, Scoping Rules, Object Access, and Triggers. The 'Details' tab is active. The main pane displays the 'Details' section for the 'Booking' object. It includes fields for Description, API Name (Booking__c), Custom (✓), Singular Label (Booking), Plural Label (Bookings), and several checkboxes for reports and tracking. At the bottom right of the main pane are 'Edit' and 'Delete' buttons. The status bar at the bottom shows the date and time as 25-09-2025.

The **Booking object** manages restaurant reservations. It allows customers to reserve a table for a specific time slot and number of guests.

- The **Booking Time** defines when the reservation occurs.
- **Pax** indicates the number of people.
- **Status** reflects the booking lifecycle (Confirmed, Seated, NoShow, Cancelled).
- Each booking is linked to a **Contact** (customer) and to a **Restaurant Table** to manage seating capacity.

Key Role:

Bookings help prevent overbooking, improve customer service, and give staff visibility into restaurant occupancy.

The screenshot shows the 'Object Manager' interface in Salesforce, specifically the 'Fields & Relationships' section for the 'Booking' object. The sidebar on the left is identical to the previous screenshot. The main pane lists the fields and their relationships. The fields listed are: Booking DateTime, Booking Name, Contact, Created By, Last Modified By, Owner, Restaurant Table, and Status. Each field is associated with its field label, field name, data type, controlling field, and indexed status. The status bar at the bottom shows the date and time as 25-09-2025.

Loyalty Account → Tracks points and tier for customer loyalty.

The screenshot shows the Salesforce Object Manager interface for the 'Loyalty Account' object. The left sidebar lists various setup options like Details, Fields & Relationships, Page Layouts, etc. The main 'Details' tab is selected, displaying fields such as API Name (Loyalty_Account__c), Singular Label (Loyalty Account), and Plural Label (Loyalty Accounts). On the right, there are sections for Reports, Activities, Field History, Deployment Status, and Help Settings. The bottom status bar shows it's 25-09-2025 at 09:23 PM.

The **Loyalty Account object** tracks customer engagement through points and reward tiers. It is tightly linked to the **Contact** object (the customer).

- Each customer can have a **Loyalty Account** where points are credited based on orders.
- **Points Balance** keeps track of available reward points.
- **Tier** (Bronze, Silver, Gold, etc.) reflects customer loyalty level.
- **Join Date** stores when the customer enrolled in the loyalty program.

Key Role:

This object supports marketing and customer retention strategies by incentivizing repeat business.

The screenshot shows the 'Fields & Relationships' tab for the Loyalty Account object. It lists six fields: Contact (Lookup(Contact)), Created By (Lookup(User)), Last Modified By (Lookup(User)), Loyalty Account Name (Name), Owner (Lookup(User/Group)), and Points (Number(18, 0)). The 'Fields & Relationships' sidebar is also visible. The bottom status bar shows it's 25-09-2025 at 09:23 PM.

Order Item → Child of Order, representing individual menu items within the order.

The screenshot shows the Salesforce Object Manager interface for the 'Order Item' object. The left sidebar lists various configuration tabs: Fields & Relationships, Page Layouts, Lightning Record Pages, Buttons, Links, and Actions, Compact Layouts, Field Sets, Object Limits, Record Types, Related Lookup Filters, Restriction Rules, Scoping Rules, Object Access, and Triggers. The main pane displays the 'Details' tab, which contains fields for Description, API Name (Order_Item__c), Singular Label (Order Item), Plural Label (Order Items), and other settings like Enable Reports, Track Activities, and Deployment Status (Deployed). The bottom status bar shows the date and time as 09:23 PM on 25-09-2025.

The **Order Item object** represents the specific menu items within an order. It ensures detailed tracking of what was ordered, in what quantity, and at what price.

- It is **child** to Order (via Master-Detail).
- It links to the **Menu Item object**, which contains the catalog of dishes available.
- **Quantity** and **Unit Price** define the financial details, while **Line Total** (Quantity × Price) ensures accurate calculation per item.

Key Role:

Without Order Items, the Order object would only represent a lump sum; Order Items break it down into granular details for billing, kitchen preparation, and inventory.

The screenshot shows the Salesforce Object Manager interface for the 'Order Item' object, specifically the 'Fields & Relationships' tab. The left sidebar lists the same configuration tabs as the previous screenshot. The main pane displays a table of fields and their relationships. The columns include FIELD LABEL, FIELD NAME, DATA TYPE, CONTROLLING FIELD, and INDEXED. Key entries include Created By (CreatedById, Lookup(User)), is it Name (Name, Text(80)), Last Modified By (LastModifiedById, Lookup(User)), Line Total (Line_Total__c, Formula(Number)), Menu Item (Menu_Item__c, Lookup(Menu Item)), Order (Order__c, Master-Detail(Order)), Quantity (Quantity__c, Number(18, 0)), and Unit Price (Unit_Price__c, Currency(18, 0)). The bottom status bar shows the date and time as 09:24 PM on 25-09-2025.

There are some fields and relationships in the menu item that are given below

The screenshot shows the Salesforce Object Manager interface for the 'Menu Item' object. On the left, a sidebar lists various setup options like Page Layouts, Lightning Record Pages, and Field Sets. The main area is titled 'Fields & Relationships' and displays six items sorted by field label. The table has columns for FIELD LABEL, FIELD NAME, DATA TYPE, CONTROLLING FIELD, and INDEXED. The fields listed are Category (Category__c, Picklist), Created By (CreatedById, Lookup(User)), Last Modified By (LastModifiedById, Lookup(User)), Menu Item Name (Name, Text(80)), Owner (OwnerId, Lookup(User,Group)), and Price (Price__c, Currency(18, 0)).

FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
Category	Category__c	Picklist		
Created By	CreatedById	Lookup(User)		
Last Modified By	LastModifiedById	Lookup(User)		
Menu Item Name	Name	Text(80)		
Owner	OwnerId	Lookup(User,Group)		
Price	Price__c	Currency(18, 0)		

The **Restaurant Table** object represents the physical seating arrangement in the restaurant. It helps the system track which tables are available, reserved, or occupied during a booking. This ensures efficient seat management and prevents double-booking.

The screenshot shows the Salesforce Object Manager interface for the 'Restaurant Table' object. The left sidebar contains the same setup options as the previous screenshot. The main area is titled 'Details' and shows the configuration for the 'Restaurant Table'. It includes fields for Description, API Name (Restaurant_Table__c), Custom status (Custom checked), Singular Label (Restaurant Table), Plural Label (Restaurant Tables), and various tracking and deployment settings like Enable Reports, Track Activities, Track Field History, Deployment Status (Deployed), Help Settings, and Standard salesforce.com Help Window.

The **Restaurant Table** object represents the physical tables in the restaurant. It ensures that seating arrangements are managed efficiently.

- Each table has a **unique identifier (Table Number)**.

- **Capacity** defines how many guests it can accommodate.
- **Status** indicates whether a table is Available, Reserved, Occupied, or Out of Service.
- **Location** can differentiate indoor vs outdoor seating.

Key Relationship:

Bookings are linked to Restaurant Tables, ensuring that reservations are tied to real seating capacity. This avoids double-booking and helps staff assign tables efficiently

FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
Capacity	Capacity__c	Number(10, 0)		
Contact Number	Contact_Number__c	Phone		
Created By	CreatedBy	Lookup(User)		
Delivery Address	Delivery_Address__c	Long Text Area(50000)		
Last Modified By	LastModifiedBy	Lookup(User)		
Owner	OwnerId	Lookup(User,Group)		
Restaurant Table	Restaurant_Table__c	Lookup(Restaurant Table)		
Restaurant Table Name	Name	Text(80)		
Table Number	Table_Number__c	Text(10)		

Create Page Layouts

Go to **Object Manager** → **Order__c** → **Page Layouts**.

Create/Clone 2 layouts:

Order Layout – Dine-In

- Key fields: Table, Booking, Order Items, Status.

Order Layout – Delivery

- Same as above + **Delivery Address, Contact Phone, Delivery Notes**.

PAGE LAYOUT NAME	CREATED BY	MODIFIED BY
Loyalty Account Layout	PAIPETI PALLAVI, 9/23/2025, 7:42 AM	PAIPETI PALLAVI, 9/23/2025, 9:16 AM

Creating Tabs for Custom Objects in Salesforce

A **tab** in Salesforce is a user interface element that provides direct access to objects, records, or web resources from the Salesforce app's navigation bar. Think of a tab as a “doorway” into the database — without it, a custom object may exist but users cannot reach it easily from the UI.

The Booking tab makes reservations discoverable to hosts and managers. Picking an obvious Tab Style (calendar icon) is a small but important UX affordance — it reduces cognitive load so staff can scan navigation quickly. When you set Tab Visibility to “Default On” for Manager and Host but “Hidden” for Kitchen, you’re enforcing a simple principle: the host and manager need immediate access to create and change bookings, the kitchen does not.

Orders are the central operational object for the restaurant, so their tab should be universally available to everyone involved in order handling (Manager, Host, Kitchen). Setting Tab Visibility to Default On for Manager, Host and Kitchen makes sense because hosts create orders, kitchen staff need to pick up items from the order, and managers monitor/override. The Orders tab provides access to list views (e.g., “Open Orders”, “Pending Payment”) and record pages (where order details, items and status live).

Order items are often a child object shown on the Order page as a related list, so creating a standalone Order Items tab is a deliberate admin decision: it’s useful for data management, reporting ad-hoc edits, and bulk imports/cleanups but often unnecessary for everyday order-taking workflows. When you make the tab Default On for Manager and Optional or Read-Only for Host/Kitchen, you are signalling that managers are the ones who might need direct access to the raw line-item data (for auditing, refunds, adjustments) while service staff generally work from the Order parent record and its related list.

Loyalty and rewards often contain personal and behavioural data and so deserve conservative visibility. By making the Loyalty Accounts tab Default On for Manager, Read-Only for Host and Hidden for Kitchen you protect customers’ reward data while still enabling hosts to view balances when assisting customers. This map supports common workflows: marketing and managers analyse and update tiers, hosts can check balances for in-person redemptions, and kitchen staff have no operational need for that information.

Custom Object Tabs

Action	Label	Tab Style	Description
Edit Del	Bookings	Fan	
Edit Del	Loyalty Accounts	Flag	
Edit Del	Order Items	Apple	
Edit Del	Orders	Castle	

Web Tabs

Action	Label	Tab Style	Description
			No Web Tabs have been defined

Visualforce Tabs

Action	Label	Tab Style	Description
			No Visualforce Tabs have been defined

Lightning Component Tabs

Action	Label	Tab Style	Description
Edit	Get Started with Agentforce	Heart	

Schema Builder

Contact

- **Core Role:** Customer record.
- **Relationships:**
 - Lookup → **Loyalty Account** (one-to-one).
 - Lookup → **Order** (one-to-many).
 - Lookup → **Booking** (one-to-many).
- **Theory:**
The Contact object is the **foundation of CRM**. Every customer interaction begins here — whether booking a table, placing an order, or redeeming loyalty points.

Loyalty Account

- **Core Role:** Tracks rewards/points for each customer.
- **Relationship:** Lookup → **Contact**.
- **Theory:**
Enables the restaurant to manage customer retention by offering a loyalty program. Every order linked to a loyalty account can earn or redeem points.

Booking

- **Core Role:** Stores table reservations.
- **Relationships:**
 - Lookup → **Contact** (who made the booking).
 - Lookup → **Restaurant Table** (where the booking is seated).
- **Theory:**
Ensures efficient **table allocation and reservation management**, preventing double bookings and improving service flow.

Restaurant Table

- **Core Role:** Represents physical seating arrangements.
- **Relationship:** Lookup → **Booking**.
- **Theory:**
Allows staff to map reservations to available tables, track occupancy, and manage seating efficiently.

Order

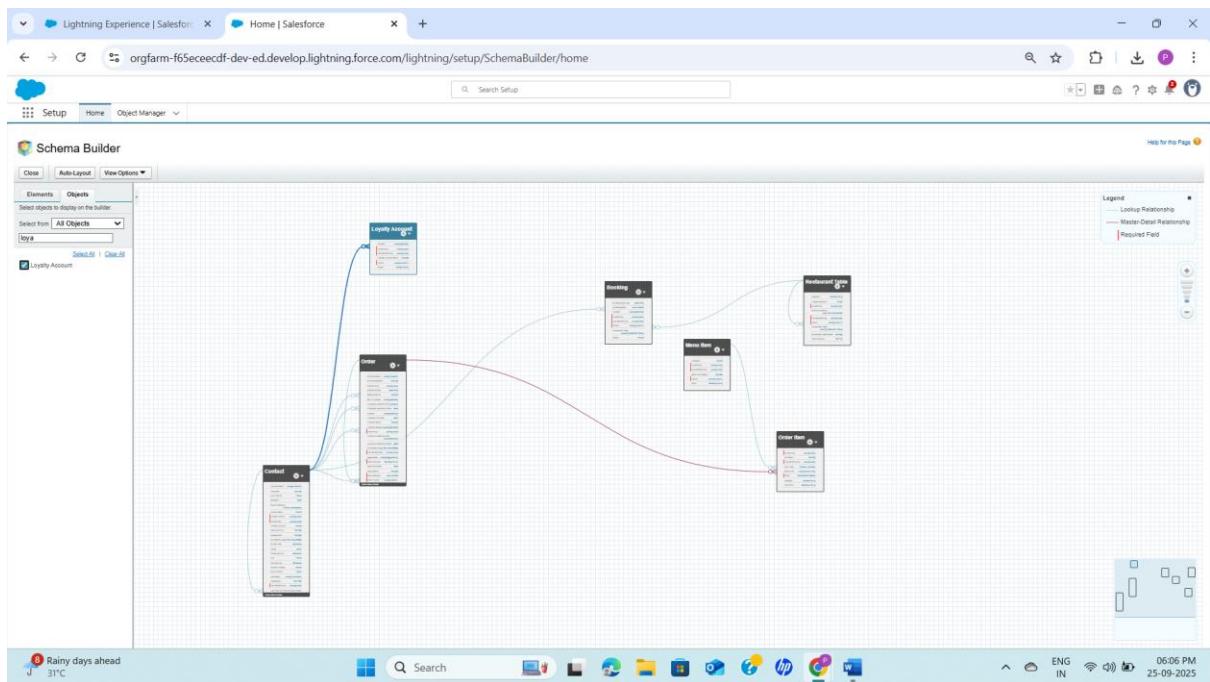
- **Core Role:** Stores all food/beverage orders.
- **Relationships:**
 - Master-Detail → **Order Item** (an order contains multiple items).
 - Lookup → **Contact** (who placed the order).
 - Lookup → **Booking** (for dine-in orders linked to reservations).
 - Lookup → **Loyalty Account** (if loyalty points are used).
- **Theory:**
The **backbone of restaurant operations** — captures both dine-in and delivery orders. It ties customer transactions to billing, loyalty points, and reporting.

Order Item

- **Core Role:** Captures individual food items within an order.
- **Relationships:**
 - Master-Detail → **Order** (parent).
 - Lookup → **Menu Item** (what was ordered).
- **Theory:**
Provides granularity. Instead of storing only the order total, the restaurant can track exactly which dishes were sold, in what quantity, and at what price.

Menu Item

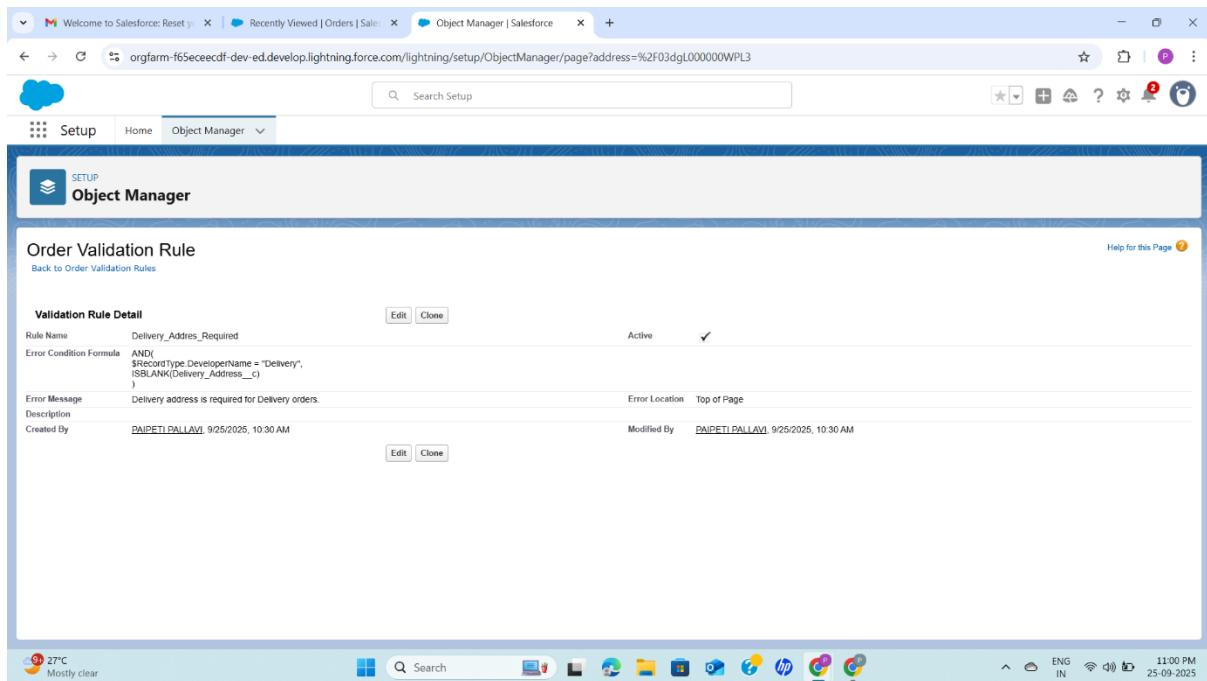
- **Core Role:** Master catalog of dishes.
- **Relationship:** Referenced by **Order Item**.
- **Theory:**
The restaurant's product database. Every order item references a menu item, ensuring consistency in pricing and reporting.



PHASE 4

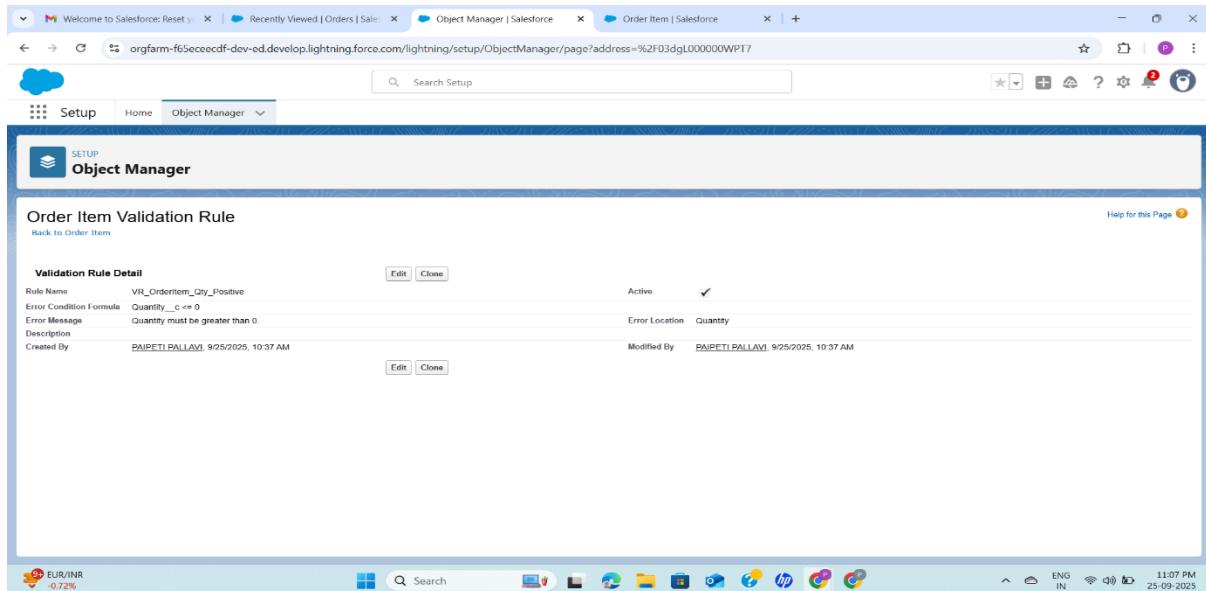
A. Delivery address required

1. Setup → Object Manager → Order__c → Validation Rules → New.
2. Fill:
 - o Rule Name: Delivery_Address_Required
 - o Error Condition Formula:
 - o AND(
 - o \$RecordType.DeveloperName = "Delivery",
 - o ISBLANK(Delivery_Address__c)
 - o)
 - o Error Message: Delivery address is required for Delivery orders.
 - o Error Location: Field → Delivery_Address__c
3. Click Save, then Activate.
4. Test: Create/update an Order record with Record Type = Delivery and no address → expect error.



B. Quantity must be > 0

1. Setup → Object Manager → Order_Item__c → Validation Rules → New.
2. Fill:
 - o Rule Name: Quantity_Positive
 - o Error Condition Formula:
 - o Quantity__c <= 0
 - o Error Message: Quantity must be greater than 0.
 - o Error Location: Field → Quantity__c
3. Save & Activate.
4. Test: Create Order Item with Quantity = 0 or negative → expect error.



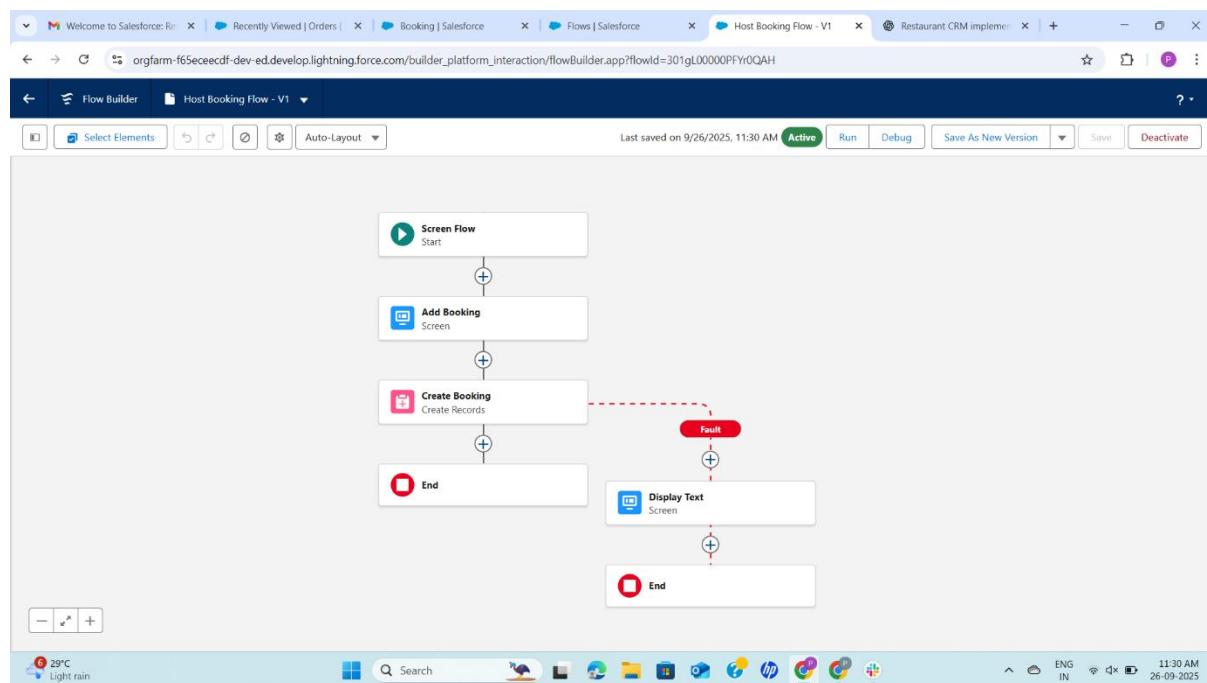
Screen Flow (Host Booking) — step-by-step

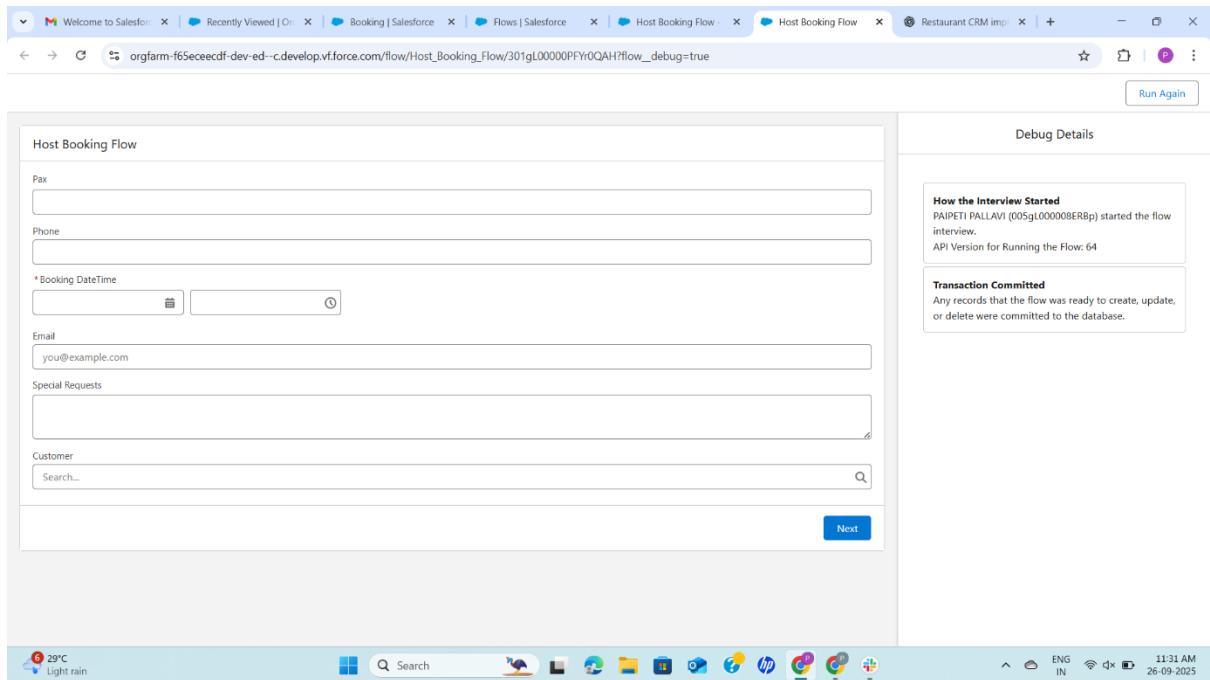
Goal: tablet-host enters booking, flow creates Booking__c, assigns table (auto or manual), notifies manager.

1. Setup → **Flow** → New Flow → choose **Screen Flow** → Create.
2. On canvas: + → **Screen**.
 - o Label: Add Booking
 - o Add components:
 - **Lookup** (Customer) → API/field: Customer__c (set Required)
 - **Number** (Pax) → Pax__c (Required)
 - **Date/Time** → Booking_DateTime__c (Required)
 - **Text Area** → Special_Requests__c
 - **Phone & Email** fields if needed
 - o Save screen.
3. **Create Records** element:
 - o Label: Create Booking Record
 - o Create one Booking__c. Map screen fields to Booking__c fields (Customer__c ← {!Customer}, etc.).
4. **Decision** element: Table Available?
 - o Option A: If you have a Table__c object, add a **Get Records** to find a table with Capacity__c >= {!Pax} and Is_Available__c = TRUE. If one found → route to Auto Assign.
 - o Option B: If complex logic required, call an **Apex Invocable** (add Action → the invocable class) which returns Table Id.
5. **Screen Confirm**:
 - o Show assigned table or a record choice to pick table manually. (Use Record Choice Set or Choice component bound to Table__c results.)

6. **Action(s):**
 - Add **Create Task or Send Custom Notification** (Action → Send Custom Notification) for manager.
 - Or use **Action → Email Alert** (if you created an Email Alert) to send email.
7. Save → **Debug** with sample values → fix mapping.
8. Save → **Activate**.
9. Expose Flow:
 - Add to Lightning App page (App Builder) or create a Quick Action that launches the flow (Object → Buttons, Links, Actions → New Action → Flow). For tablet, add to the app page or make it full-screen.

Testing: on a device, run flow, confirm Booking__c created, table assigned, notifications received.





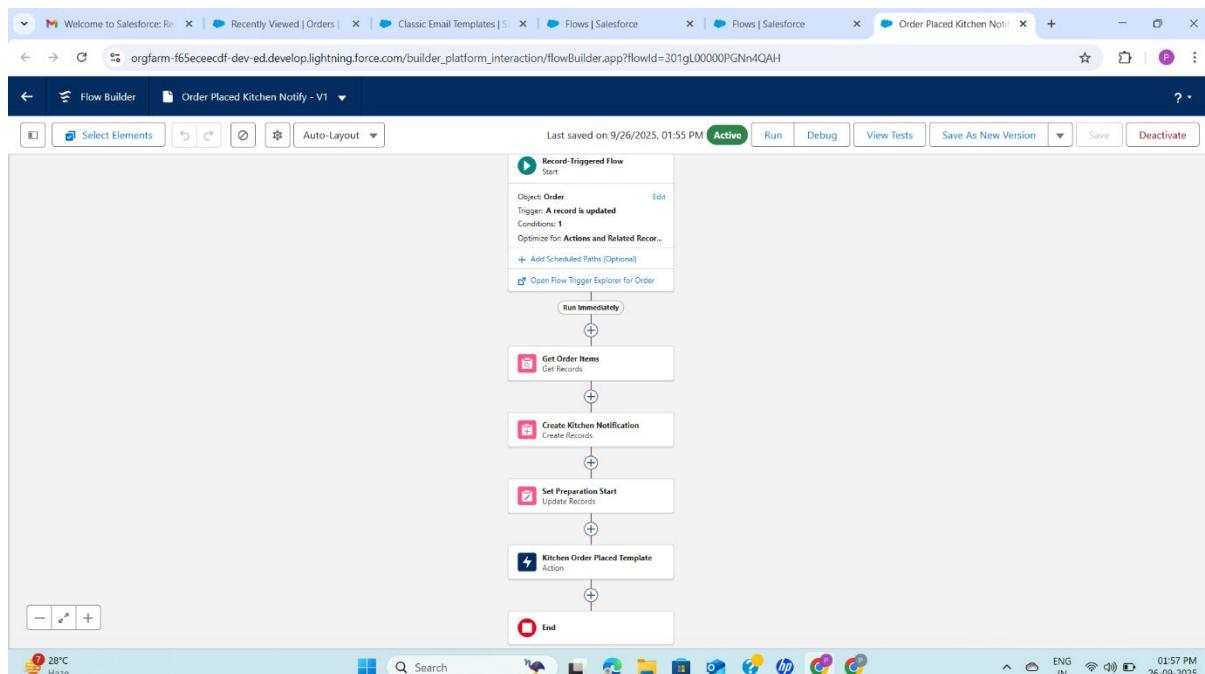
Record-Triggered Flow

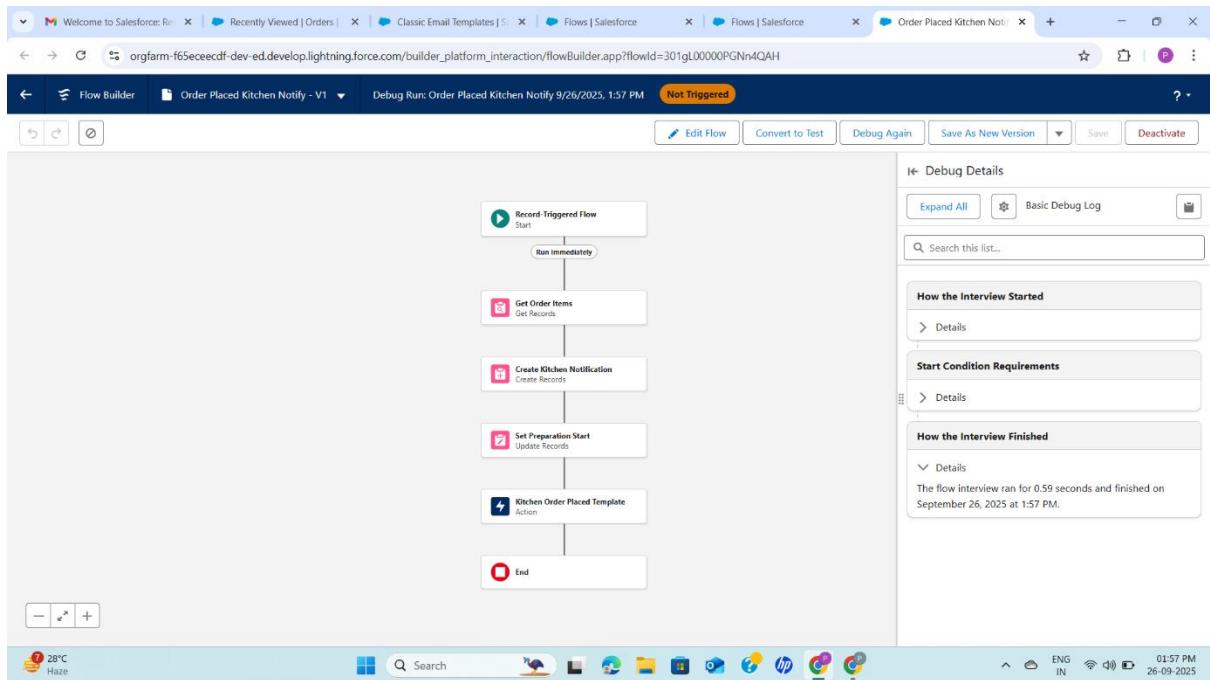
Goal: When Order__c.Status__c becomes **Placed**, notify kitchen, set Preparation_Start__c.

1. Setup → Flows → New Flow → **Record-Triggered Flow** → Create.
2. Configure start:
 - **Object:** Order__c
 - **Trigger:** A record is updated
 - **Condition Requirements:** Status__c EQUALS Placed (use picklist equals).
 - **When to Run:** After the record is saved (we will create other records & notifications).
 - **Optimize the Flow for:** Actions and Related Records.
 - Click Done.
3. First **Get Records:**
 - Label: Get Order Items
 - Object: Order_Item__c
 - Filter: Order__c = \$Record.Id
 - Get all records, store as collection.
4. (Optional) **Decision:** If no order items → end (avoid empty operations).
5. **Create Records:** create Kitchen_Order__c (or your custom notification record) — you can create one record per Order or one per item.
 - For large sets, create one summary Kitchen_Order and attach items via related list.
6. **Update Records:**
 - Label: Set Prep Start
 - Update record: choose the triggering record (Order__c where Id = \$Record.Id) → set Preparation_Start__c = \$Flow.CurrentDateTime (or use formula NOW()).
7. **Action:** Send Notification

- Option A: **Send Custom Notification** action → choose Notification Type (create one first) and set Recipient IDs (Queue/Users).
 - Option B: Use **Action** → **Email Alert** (pre-created Email Alert).
 - Option C: **Post to Chatter** action: Post to a group or user.
8. **Add Scheduled Path** (if you want follow-up/reminder or auto-cancel):
- In the Start element, click + **Add Scheduled Paths**, set label Auto Cancel If Not Confirmed, offset 24 Hours After Trigger (or X hours), then on that scheduled path add a Decision: if Order_cConfirmed_c = FALSE → **Update Records** to set Status_c = 'Cancelled' and send email.
9. Save → **Debug**: run with a test order using "Run" debug in flow (use record Id). Review Flow Interviews in Setup → Paused Flow Interviews (if scheduled).
10. Activate.

Testing: Update an Order to Placed → check Kitchen_Order created, Preparation_Start updated, notifications emailed.





Scheduled Flow

Goal: nightly job to process Loyalty_Account__c and update balances.

1. Setup → **Flows** → New Flow → **Scheduled-Triggered Flow** → Create.
2. Configure schedule:
 - **Start Date & Time:** pick next run time (e.g., tomorrow 02:00 AM)
 - **Frequency:** Daily
 - **Time Zone:** Org default (confirm timezone)
3. Canvas:
 - **Get Records:** Loyalty_Account__c → filter criteria as needed (e.g., Active = TRUE). **Store all records** into a collection.
 - **Loop** over the collection (Loop variable: loopLoyaltyAccounts).
 - In loop: **Decision** → has expiry? has activity?
 - If points to add: Add a record to a collection variable transactionsToCreate (build a record variable using Assignment).
 - Also use Assignment to compute new Points_Balance__c into a accountsToUpdate collection (create collection of records to update).
 - After Loop: **Create Records** (create many Loyalty_Transaction__c using transactionsToCreate) — this uses bulk create.
 - **Update Records** — use accountsToUpdate collection to bulk update balances.
4. Save → Debug (use a small test dataset) → Activate.
5. **Scale note:** If your org has thousands+ accounts, consider Batch Apex or break the scheduled flow to process smaller slices (e.g., criteria by last modified ranges) to avoid platform limits.

Create Notification Type (for in-app)

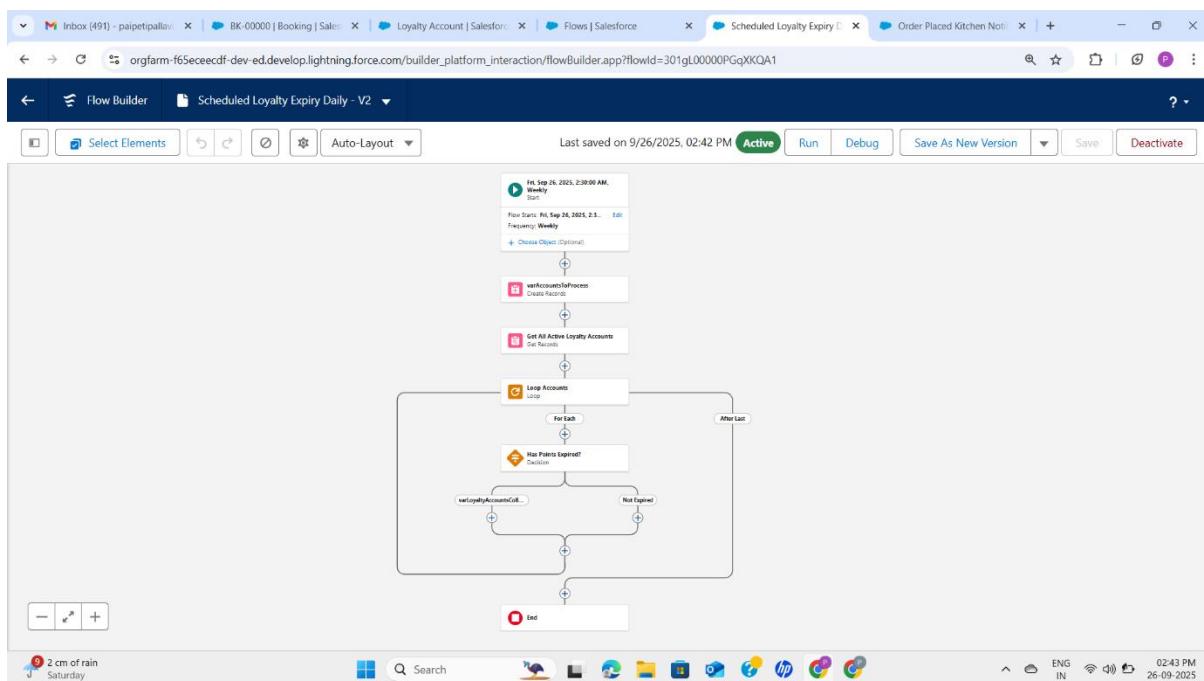
1. Setup → **Notification Builder** → **Notification Type** → New.
2. Label: Kitchen_Order_Alert, Channels: In-App and Desktop if needed. Save.

Create Email Template & Email Alert

1. Setup → **Email Templates** → **New Email Template** (Lightning) → create template using merge fields.
2. Setup → **Email Alerts** → **New Email Alert**:
 - o Object: Order__c
 - o Email Template: choose above
 - o Recipients: User, Queue, Related Users (e.g., Order.Owner)
3. Use this Email Alert in Flow:
 - o In Flow canvas: **Action** → search for **Email Alert** → choose your alert.

Send Custom Notification in Flow

1. In Flow canvas: **Action** → Send Custom Notification → choose Notification Type created earlier.
2. Supply Title, Body, Recipient Id(s) (use {!User.Id} or collection).



Created a Booking record (**Booking__c** object)

- Record Name auto-generated: **BK-00000** (probably an Auto-Number field).

Filled core Booking fields:

- **Contact** → Selected *Maria Clifton* (lookup to Contact).
- **Booking DateTime** → 9/27/2025, 12:00 PM.
- **Restaurant Table** → Linked to a Table record (lookup).
- **Status** → Set as *Pending* (picklist).
- **Account** → Left empty (lookup to Account).
- **Pax** → Empty (should store number of guests).
- **Phone, Email, Special Requests** → Empty (but available for input).

Record Owner → You (Paipeti Pallavi).

- This means you are responsible for the booking.

Activity Panel → No tasks/emails/events logged yet.

The screenshot shows a Salesforce Lightning interface for a Booking record named 'BK-00000'. The page is divided into two main sections: 'Details' on the left and 'Activity' on the right.

Details Section:

- Booking Name:** BK-00000
- Contact:** Maria Clifton
- Booking Date/Time:** 9/27/2025, 12:00 PM
- Restaurant Table:** Table
- Status:** Pending
- Account:** (No value)
- Pax:** (No value)
- Phone:** (No value)

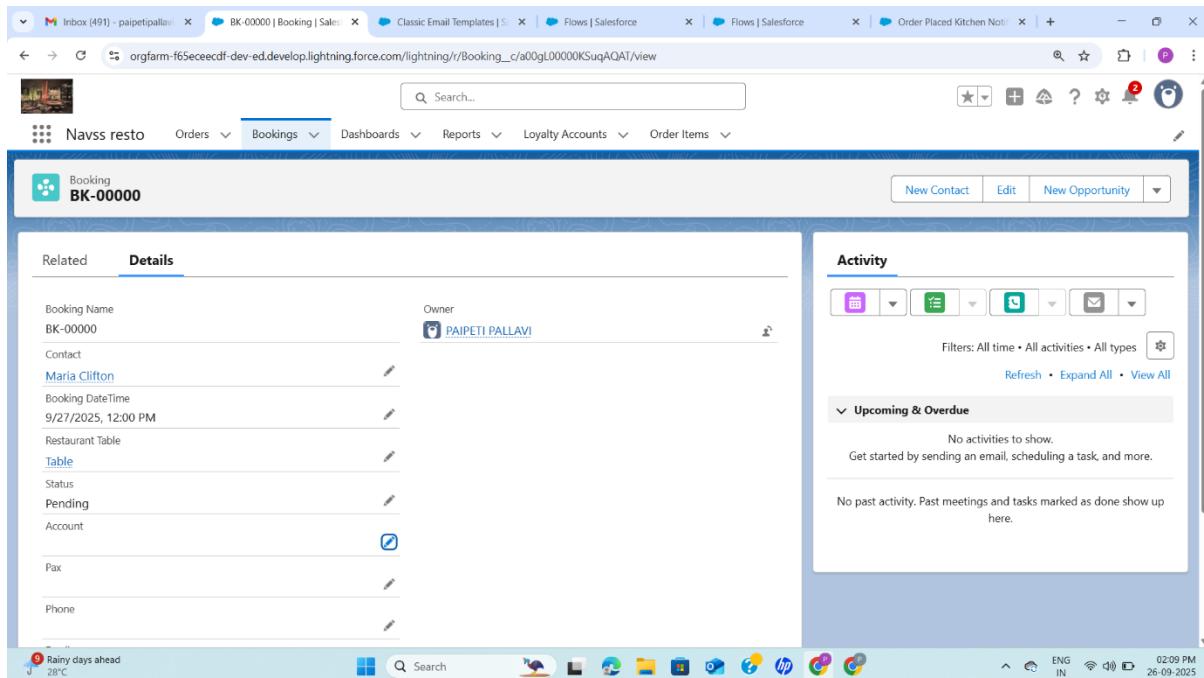
Activity Section:

- Owner:** PAIPETI PALLAVI
- Filters:** All time • All activities • All types
- Upcoming & Overdue:** No activities to show.
- No past activity:** Past meetings and tasks marked as done show up here.

The browser taskbar at the bottom shows multiple open tabs related to the application, and the system status bar indicates it's 02:09 PM on 26-09-2025.

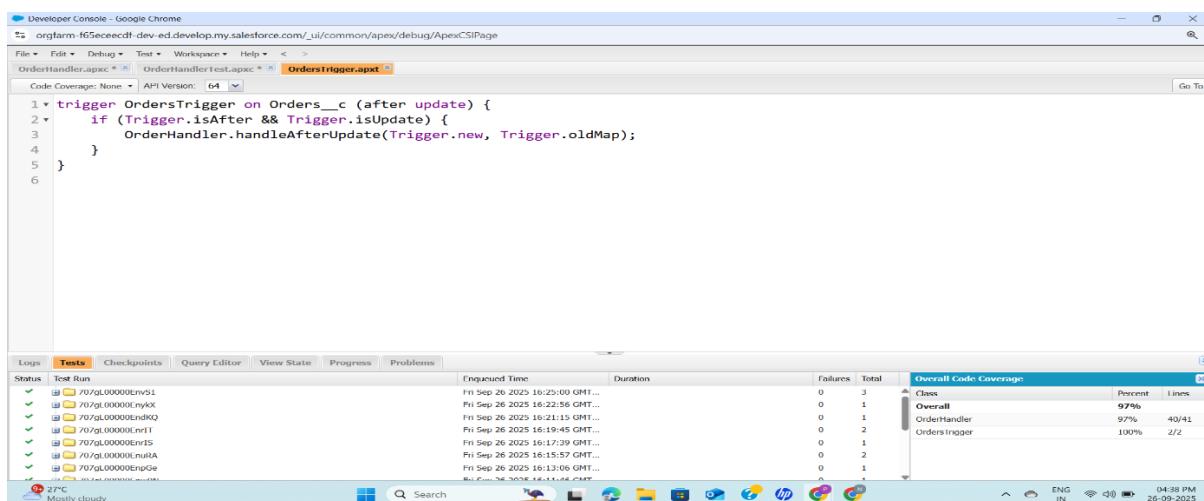
PHASE 5

Before running the apex classes the status is in pending state



I have used the apex to change the status to the confirmed state.

OrderTrigger.trigger



OrderHandler.apxc

Developer Console - Google Chrome
 orgfarm-f65ecccdf-dev-ed.develop.my.salesforce.com/_ui/common/apex/debug/ApexCSIPage

File • Edit • Debug • Test • Workspace • Help • < >

OrderHandler.apxc OrderHandlerTest.apxc OrdersTrigger.apxt

Code Coverage: None • API Version: 64 • Go To

```

1 * public class OrderHandler {
2   public static void handleAfterUpdate(List<Orders__c> newList, Map<Id, Orders__c> oldMap) {
3     // 1. Collect completed orders (status changed to Completed)
4     List<Orders__c> completedOrders = new List<Orders__c>();
5     for (Orders__c o : newList) {
6       if (o.Status__c == 'Completed' && oldMap.get(o.Id).Status__c != 'Completed') {
7         completedOrders.add(o);
8       }
9     }
10
11   if (completedOrders.isEmpty()) {
12     return;
13   }
14
15   // 2. Collect related Contact Ids
16   Set<Id> contactIds = new Set<Id>();
17   for (Orders__c o : completedOrders) {

```

Logs Tests Checkpoints Query Editor View State Progress Problems

Status	Test Run	Enqueued Time	Duration	Failures	Total	Overall Code Coverage
✓	707gl.00000EnvS1	Fri Sep 26 2025 16:25:00 GMT...		0	3	Class
✓	707gl.00000EnvKX	Fri Sep 26 2025 16:22:56 GMT...		0	1	Overall
✓	707gl.00000EnvRQ	Fri Sep 26 2025 16:21:15 GMT...		0	1	OrderHandler
✓	707gl.00000EnvTT	Fri Sep 26 2025 16:19:45 GMT...		0	2	OrdersTrigger
✓	707gl.00000EnvTS	Fri Sep 26 2025 16:17:39 GMT...		0	1	
✓	707gl.00000EnvRA	Fri Sep 26 2025 16:15:57 GMT...		0	2	
✓	707gl.00000EnvGe	Fri Sep 26 2025 16:13:06 GMT...		0	1	

27°C Mostly cloudy 04:34 PM ENG IN 26-09-2025

Developer Console - Google Chrome
 orgfarm-f65ecccdf-dev-ed.develop.my.salesforce.com/_ui/common/apex/debug/ApexCSIPage

File • Edit • Debug • Test • Workspace • Help • < >

OrderHandler.apxc OrderHandlerTest.apxc OrdersTrigger.apxt

Code Coverage: None • API Version: 64 • Go To

```

18   if (o.Contact__c != null) {
19     contactIds.add(o.Contact__c);
20   }
21
22
23   // 3. Calculate points per contact
24   Map<Id, Integer> contactToPoints = new Map<Id, Integer>();
25   for (Orders__c o : completedOrders) {
26     if (o.Contact__c != null && o.Total_Amounts__c != null) {
27       // FIX: Use Math.floor and intValue() to avoid "Illegal assignment from Decimal to Integer"
28       Integer pts = Math.floor(o.Total_Amounts__c / 10).intValue();
29
30       contactToPoints.put(
31         o.Contact__c,
32         contactToPoints.containsKey(o.Contact__c)
33           ? contactToPoints.get(o.Contact__c) + pts
34           : pts

```

Logs Tests Checkpoints Query Editor View State Progress Problems

Status	Test Run	Enqueued Time	Duration	Failures	Total	Overall Code Coverage
✓	707gl.00000EnvS1	Fri Sep 26 2025 16:25:00 GMT...		0	3	Class
✓	707gl.00000EnvKX	Fri Sep 26 2025 16:22:56 GMT...		0	1	Overall
✓	707gl.00000EnvRQ	Fri Sep 26 2025 16:21:15 GMT...		0	1	OrderHandler
✓	707gl.00000EnvTT	Fri Sep 26 2025 16:19:45 GMT...		0	2	OrdersTrigger
✓	707gl.00000EnvTS	Fri Sep 26 2025 16:17:39 GMT...		0	1	
✓	707gl.00000EnvRA	Fri Sep 26 2025 16:15:57 GMT...		0	2	
✓	707gl.00000EnvGe	Fri Sep 26 2025 16:13:06 GMT...		0	1	

27°C Mostly cloudy 04:35 PM ENG IN 26-09-2025

Developer Console - Google Chrome
 orgfarm-f65eccecd1-dev-ed.develop.my.salesforce.com/_ui/common/apex/debug/ApexCSIPage

File ▾ Edit ▾ Debug ▾ Test ▾ Workspace ▾ Help ▾ < >

OrderHandler.apxc * OrderHandlerTest.apxc OrdersTrigger.apxt

Code Coverage: None API Version: 64 Go To

```

35      );
36  }
37 }
38
39 // 4. Query existing Loyalty Accounts
40 Map<Id, Loyalty_Account__c> contactToAccount = new Map<Id, Loyalty_Account__c>();
41 for (Loyalty_Account__c la : [
42     SELECT Id, Contact__c, Points__c
43     FROM Loyalty_Account__c
44     WHERE Contact__c IN :contactIds
45 ]) {
46     contactToAccount.put(la.Contact__c, la);
47 }
48
49 // 5. Prepare inserts & updates
50 List<Loyalty_Account__c> toInsert = new List<Loyalty_Account__c>();
51 List<Loyalty_Account__c> toUpdate = new List<Loyalty_Account__c>();

```

Logs Tests Checkpoints Query Editor View State Progress Problems

Status	Test Run	Enqueued Time	Duration	Failures	Total	Overall Code Coverage
✓	707gl.00000Efv5I	Fri Sep 26 2025 16:25:00 GMT...		0	3	Class Overall 97%
✓	707gl.00000EnyKX	Fri Sep 26 2025 16:22:56 GMT...		0	1	Percent 40/41 Lines
✓	707gl.00000EndQ	Fri Sep 26 2025 16:21:15 GMT...		0	1	OrderHandler 97% 40/41
✓	707gl.00000EvrTT	Fri Sep 26 2025 16:19:45 GMT...		0	2	OrdersTrigger 100% 2/2
✓	707gl.00000EnvIS	Fri Sep 26 2025 16:17:39 GMT...		0	1	
✓	707gl.00000EnuRA	Fri Sep 26 2025 16:15:57 GMT...		0	2	
✓	707gl.00000EngGe	Fri Sep 26 2025 16:13:06 GMT...		0	1	
	707gl.00000Evn...AN	Fri Sep 26 2025 16:11:46 GMT...		0	1	

27°C Mostly cloudy Search ENG IN 04:35 PM 26-09-2025

Developer Console - Google Chrome
 orgfarm-f65eccecd1-dev-ed.develop.my.salesforce.com/_ui/common/apex/debug/ApexCSIPage

File ▾ Edit ▾ Debug ▾ Test ▾ Workspace ▾ Help ▾ < >

OrderHandler.apxc * OrderHandlerTest.apxc OrdersTrigger.apxt

Code Coverage: None API Version: 64 Go To

```

52 for (Id contactId : contactToPoints.keySet()) {
53     Integer earnedPoints = contactToPoints.get(contactId);
54     if (contactToAccount.containsKey(contactId)) {
55         Loyalty_Account__c la = contactToAccount.get(contactId);
56         la.Points__c += earnedPoints;
57         toUpdate.add(la);
58     } else {
59         Loyalty_Account__c la = new Loyalty_Account__c(
60             Contact__c = contactId,
61             Points__c = earnedPoints
62         );
63         toInsert.add(la);
64     }
65 }
66 if (!toInsert.isEmpty()) {
67     insert toInsert;
68 }

```

Logs Tests Checkpoints Query Editor View State Progress Problems

Status	Test Run	Enqueued Time	Duration	Failures	Total	Overall Code Coverage
✓	707gl.00000Efv5I	Fri Sep 26 2025 16:25:00 GMT...		0	3	Class Overall 97%
✓	707gl.00000EnyKX	Fri Sep 26 2025 16:22:56 GMT...		0	1	Percent 40/41 Lines
✓	707gl.00000EndQ	Fri Sep 26 2025 16:21:15 GMT...		0	1	OrderHandler 97% 40/41
✓	707gl.00000EvrTT	Fri Sep 26 2025 16:19:45 GMT...		0	2	OrdersTrigger 100% 2/2
✓	707gl.00000EnvIS	Fri Sep 26 2025 16:17:39 GMT...		0	1	
✓	707gl.00000EnuRA	Fri Sep 26 2025 16:15:57 GMT...		0	2	
✓	707gl.00000EngGe	Fri Sep 26 2025 16:13:06 GMT...		0	1	
	707gl.00000Evn...AN	Fri Sep 26 2025 16:11:46 GMT...		0	1	

27°C Mostly cloudy Search ENG IN 04:36 PM 26-09-2025

OrderHandlerTest.apxc

Developer Console - Google Chrome
 orgfarm-f65ecccdf-dev-ed.develop.my.salesforce.com/_ui/common/apex/debug/ApexCSIPage

File ▾ Edit ▾ Debug ▾ Test ▾ Workspace ▾ Help ▾ < >

OrderHandler.apcpx * OrderHandlerTest.apcpx OrdersTrigger.apcpx

Code Coverage: None API Version: 64 Go To

```

1  @isTest
2  private class OrderHandlerTest {
3
4      @testSetup
5      static void setupData() {
6          // Create test contact
7          Contact c = new Contact(
8              LastName = 'Test User',
9              Email = 'test@example.com'
10         );
11         insert c;
12
13         // Create initial order
14         Orders__c o = new Orders__c(
15             Name = 'Test Order',
16             Status__c = 'Pending',
17             Total_Amounts__c = 250,
18         );
19         insert o;
20
21     }
22
23     @isTest
24     static void testLoyaltyAccountCreation() {
25         Orders__c o = [SELECT Id, Status__c FROM Orders__c LIMIT 1];
26         o.Status__c = 'Completed';
27         update o;
28
29         List<Loyalty_Account__c> laList = [
30             SELECT Id, Points__c, Contact__c
31             FROM Loyalty_Account__c
32             LIMIT 1
33         ];
34         System.assertEquals(1, laList.size(), 'Loyalty account should be created');
  
```

Logs Tests Checkpoints Query Editor View State Progress Problems

Status	Test Run	Enqueued Time	Duration	Failures	Total	Overall Code Coverage
✓	707gl.00000EnvS1	Fri Sep 26 2025 16:25:00 GMT...		0	3	Class: 97%
✓	707gl.00000EnvKX	Fri Sep 26 2025 16:22:56 GMT...		0	1	Overall: 97%
✓	707gl.00000EnvQ	Fri Sep 26 2025 16:21:15 GMT...		0	1	OrderHandler: 97%
✓	707gl.00000EnvTT	Fri Sep 26 2025 16:19:45 GMT...		0	2	OrdersTrigger: 100%
✓	707gl.00000EnvTS	Fri Sep 26 2025 16:17:39 GMT...		0	1	
✓	707gl.00000EnvRA	Fri Sep 26 2025 16:15:57 GMT...		0	2	
✓	707gl.00000EnvGe	Fri Sep 26 2025 16:13:06 GMT...		0	1	

27°C Mostly cloudy 04:37 PM ENG IN 26-09-2025

Developer Console - Google Chrome
 orgfarm-f65ecccdf-dev-ed.develop.my.salesforce.com/_ui/common/apex/debug/ApexCSIPage

File ▾ Edit ▾ Debug ▾ Test ▾ Workspace ▾ Help ▾ < >

OrderHandler.apcpx * OrderHandlerTest.apcpx OrdersTrigger.apcpx

Code Coverage: None API Version: 64 Go To

```

18     Contact__c = c.Id
19     );
20     insert o;
21 }
22
23     @isTest
24     static void testLoyaltyAccountCreation() {
25         Orders__c o = [SELECT Id, Status__c FROM Orders__c LIMIT 1];
26         o.Status__c = 'Completed';
27         update o;
28
29         List<Loyalty_Account__c> laList = [
30             SELECT Id, Points__c, Contact__c
31             FROM Loyalty_Account__c
32             LIMIT 1
33         ];
34         System.assertEquals(1, laList.size(), 'Loyalty account should be created');
  
```

Logs Tests Checkpoints Query Editor View State Progress Problems

Status	Test Run	Enqueued Time	Duration	Failures	Total	Overall Code Coverage
✓	707gl.00000EnvS1	Fri Sep 26 2025 16:25:00 GMT...		0	3	Class: 97%
✓	707gl.00000EnvKX	Fri Sep 26 2025 16:22:56 GMT...		0	1	Overall: 97%
✓	707gl.00000EnvQ	Fri Sep 26 2025 16:21:15 GMT...		0	1	OrderHandler: 97%
✓	707gl.00000EnvTT	Fri Sep 26 2025 16:19:45 GMT...		0	2	OrdersTrigger: 100%
✓	707gl.00000EnvTS	Fri Sep 26 2025 16:17:39 GMT...		0	1	
✓	707gl.00000EnvRA	Fri Sep 26 2025 16:15:57 GMT...		0	2	
✓	707gl.00000EnvGe	Fri Sep 26 2025 16:13:06 GMT...		0	1	

27°C Mostly cloudy 04:37 PM ENG IN 26-09-2025

Developer Console - Google Chrome

File Edit Debug Test Workspace Help < >

OrderHandler.apxc OrderHandlerTest.apxc OrdersTrigger.apxt

Code Coverage: None API Version: 64 Go To

```

35
36     System.assertEquals(25, laList[0].Points__c.intValue(), 'Expected 25 points for 250 total');
37
38     @isTest
39     static void testLoyaltyAccountUpdate() {
40         // Complete first order
41         Orders__c o = [SELECT Id, Status__c FROM Orders__c LIMIT 1];
42         o.Status__c = 'Completed';
43         update o;
44
45         Loyalty_Account__c la = [SELECT Id, Points__c, Contact__c FROM Loyalty_Account__c LIMIT 1];
46         Integer initialPoints = la.Points__c.intValue(); // Fixed Decimal → Integer
47
48         // Create second order for same contact
49         Contact c = [SELECT Id FROM Contact LIMIT 1];
50         Orders__c o2 = new Orders__c(
51             Name = 'Second Order',

```

Logs Tests Checkpoints Query Editor View State Progress Problems

Status	Test Run	Enqueued Time	Duration	Failures	Total	Overall Code Coverage
✓	707gl000000Ev51	Fri Sep 26 2025 16:29:00 GMT...		0	1	Overall 97%
✓	707gl000000enyXK	Fri Sep 26 2025 16:22:56 GMT...		0	1	OrderHandler 97% 40/41
✓	707gl000000EndkQ	Fri Sep 26 2025 16:21:15 GMT...		0	1	OrdersTrigger 100% 2/2
✓	707gl000000EnvIT	Fri Sep 26 2025 16:19:45 GMT...		0	2	
✓	707gl000000EnvIS	Fri Sep 26 2025 16:17:39 GMT...		0	1	
✓	707gl000000EnvRA	Fri Sep 26 2025 16:15:57 GMT...		0	2	
✓	707gl000000EnvGe	Fri Sep 26 2025 16:13:06 GMT...		0	1	

27°C Mostly cloudy Search ENG IN 04:37 PM 26-09-2025

Developer Console - Google Chrome

File Edit Debug Test Workspace Help < >

OrderHandler.apxc OrderHandlerTest.apxc OrdersTrigger.apxt

Code Coverage: None API Version: 64 Go To

```

52         Status__c = 'Pending',
53         Total_Amounts__c = 100,
54         Contact__c = c.Id
55     );
56     insert o2;

58     // Complete second order
59     o2.Status__c = 'Completed';
60     update o2;

62     // Verify points updated correctly
63     Loyalty_Account__c la2 = [
64         SELECT Id, Points__c
65         FROM Loyalty_Account__c
66         WHERE Contact__c = :c.Id
67         LIMIT 1
68 ];

```

Logs Tests Checkpoints Query Editor View State Progress Problems

Status	Test Run	Enqueued Time	Duration	Failures	Total	Overall Code Coverage
✓	707gl000000Ev51	Fri Sep 26 2025 16:25:00 GMT...		0	3	Class 97%
✓	707gl000000enyXK	Fri Sep 26 2025 16:22:56 GMT...		0	1	Overall 97%
✓	707gl000000EndkQ	Fri Sep 26 2025 16:21:15 GMT...		0	1	OrderHandler 97% 40/41
✓	707gl000000EnvIT	Fri Sep 26 2025 16:19:45 GMT...		0	2	OrdersTrigger 100% 2/2
✓	707gl000000EnvIS	Fri Sep 26 2025 16:17:39 GMT...		0	1	
✓	707gl000000EnvRA	Fri Sep 26 2025 16:15:57 GMT...		0	2	
✓	707gl000000EnvGe	Fri Sep 26 2025 16:13:06 GMT...		0	1	

27°C Mostly cloudy Search ENG IN 04:37 PM 26-09-2025

Developer Console - Google Chrome
 orgfarm-f65ceecdf-dev-ed.develop.my.salesforce.com/_ui/common/apex/debug/ApexCSIPage

File ▾ Edit ▾ Debug ▾ Test ▾ Workspace ▾ Help ▾ < >

OrderHandler.apxc * OrderHandlerTest.apxc OrdersTrigger.apxt

Code Coverage: None API Version: 64 Go To

```

69     System.assertEquals(initialPoints + 10, la2.Points__c.intValue(), 'Points should increase by 10');
70 }
71 @isTest
72 static void testNoLoyaltyWhenNotCompleted() {
73     Contact c = [SELECT Id FROM Contact LIMIT 1];
74     Orders__c o = new Orders__c(
75         Name = 'Uncompleted Order',
76         Status__c = 'Pending',
77         Total_Amounts__c = 500,
78         Contact__c = c.Id
79 );
80     insert o;
81     Test.startTest();
82     o.Status__c = 'Pending'; // no change
83     update o;
84     Test.stopTest();
85     List<Loyalty_Account__c> laList = [

```

Logs Tests Checkpoints Query Editor View State Progress Problems

Status	Test Run	Enqueued Time	Duration	Failures	Total	Overall Code Coverage
✓	707gl.00000EnvS1	Fri Sep 26 2025 16:25:00 GMT...		0	3	Class Overall
✓	707gl.00000EnvXK	Fri Sep 26 2025 16:22:56 GMT...		0	1	97%
✓	707gl.00000EndRQ	Fri Sep 26 2025 16:21:15 GMT...		0	1	OrderHandler
✓	707gl.00000EnvTT	Fri Sep 26 2025 16:19:45 GMT...		0	2	OrdersTrigger
✓	707gl.00000EnvTS	Fri Sep 26 2025 16:17:39 GMT...		0	1	100% 2/2
✓	707gl.00000EnvRA	Fri Sep 26 2025 16:15:57 GMT...		0	2	
✓	707gl.00000EnvGe	Fri Sep 26 2025 16:13:06 GMT...		0	1	
✓	707gl.00000EnvCN	Fri Sep 26 2025 16:11:44 GMT...		0	1	

27°C Mostly cloudy 04:38 PM ENG IN 26-09-2025

Developer Console - Google Chrome
 orgfarm-f65ceecdf-dev-ed.develop.my.salesforce.com/_ui/common/apex/debug/ApexCSIPage

File ▾ Edit ▾ Debug ▾ Test ▾ Workspace ▾ Help ▾ < >

OrderHandler.apxc * OrderHandlerTest.apxc OrdersTrigger.apxt

Code Coverage: None API Version: 64 Go To

```

75     Name = 'Uncompleted Order',
76     Status__c = 'Pending',
77     Total_Amounts__c = 500,
78     Contact__c = c.Id
79 );
80     insert o;
81     Test.startTest();
82     o.Status__c = 'Pending'; // no change
83     update o;
84     Test.stopTest();
85     List<Loyalty_Account__c> laList = [
86         SELECT Id FROM Loyalty_Account__c WHERE Contact__c = :c.Id
87     ];
88     System.assertEquals(0, laList.size(), 'No loyalty account should be created if order not completed');
89 }
90 }
91 
```

Logs Tests Checkpoints Query Editor View State Progress Problems

Status	Test Run	Enqueued Time	Duration	Failures	Total	Overall Code Coverage
✓	707gl.00000EnvS1	Fri Sep 26 2025 16:25:00 GMT...		0	3	Class Overall
✓	707gl.00000EnvXK	Fri Sep 26 2025 16:22:56 GMT...		0	1	97%
✓	707gl.00000EndRQ	Fri Sep 26 2025 16:21:15 GMT...		0	1	OrderHandler
✓	707gl.00000EnvTT	Fri Sep 26 2025 16:19:45 GMT...		0	2	OrdersTrigger
✓	707gl.00000EnvTS	Fri Sep 26 2025 16:17:39 GMT...		0	1	100% 2/2
✓	707gl.00000EnvRA	Fri Sep 26 2025 16:15:57 GMT...		0	2	
✓	707gl.00000EnvGe	Fri Sep 26 2025 16:13:06 GMT...		0	1	
✓	707gl.00000EnvCN	Fri Sep 26 2025 16:11:44 GMT...		0	1	

27°C Mostly cloudy 04:38 PM ENG IN 26-09-2025

After executing the apex codes the status is changed to the confirmed

The screenshot shows the Salesforce Lightning interface for a booking record. The top navigation bar includes tabs for 'Booking | Salesforce' and 'Order | Salesforce'. The URL is https://orgfarm-f65ecccdf-dev-ed.lightning.force.com/lightning/r/Booking_c/a00gL00000KSuqAQAT/view. The header features a search bar and various icons. The main content area has a title 'Booking BK-00000' and tabs for 'Related' and 'Details'. Under 'Details', fields include 'Booking Name' (BK-00000), 'Owner' (PAIPETI PALLAVI), 'Contact' (Maria Clifton), 'Booking Date/Time' (9/27/2025, 12:00 PM), 'Restaurant Table' (Table), 'Status' (Confirmed), and 'Account' (Account). Below these are sections for 'Pax' and 'Phone'. To the right is an 'Activity' sidebar with a weather forecast (27°C, Mostly cloudy), a toolbar with icons for calendar, file, message, etc., and a section for 'Upcoming & Overdue' activities. The bottom of the screen shows the Windows taskbar with various pinned icons and system status.

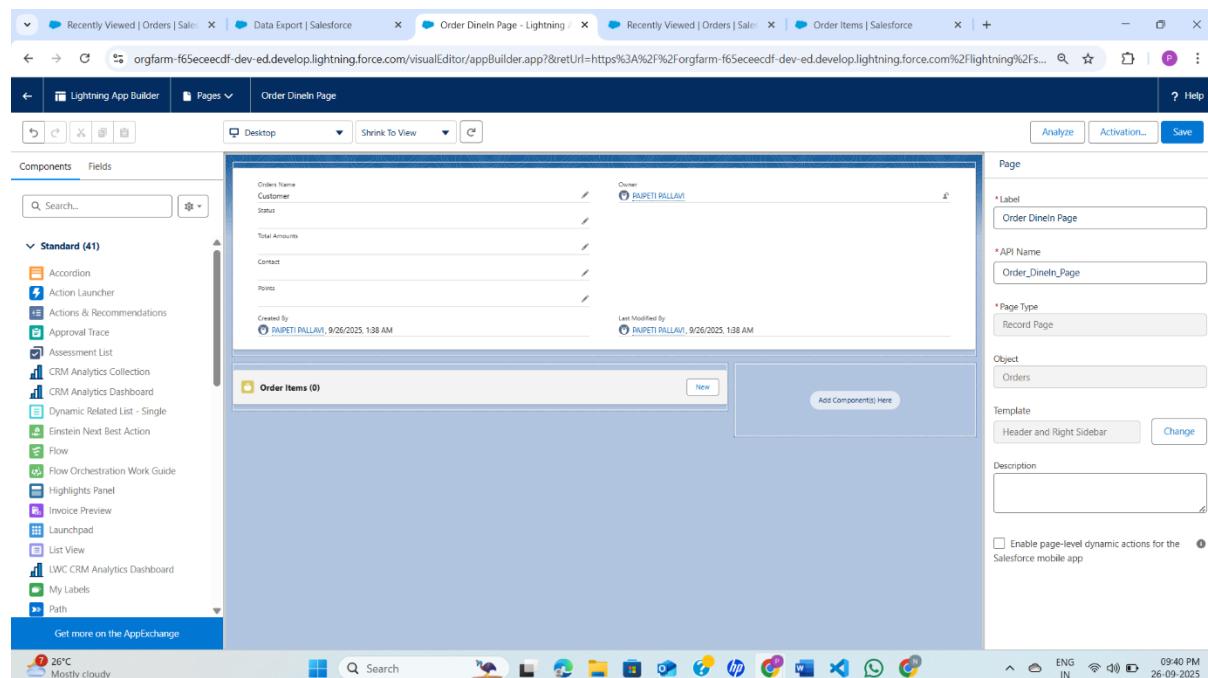
PHASE 6

LIGHTNING WEB COMPONENTS

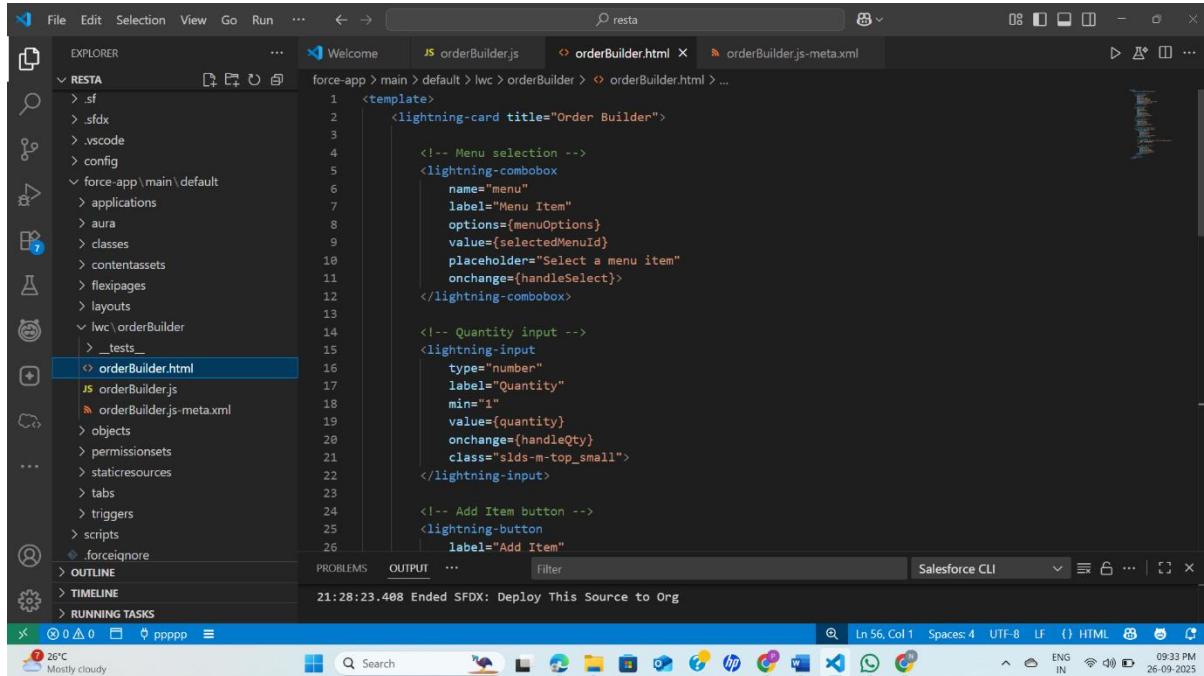
Before creating the LWC components I have created the lightning app builder

Page Details (Right Panel)

- Label:**
The display name of the page → Order DineIn Page.
- API Name:**
Internal developer name → Order_DineIn_Page.
(This name is used in code/configuration, not shown to end-users.)
- Page Type:**
Record Page → means this layout is tied to a Salesforce Object's record (in your case, the **Orders** object).
Example: When a user opens a record from the **Orders** object, this page layout will load.
- Object:**
Orders → This page is specifically for **Order** records.
- Template:**
Header and Right Sidebar → The structure of the page, defining where components can be placed.
(Here, you see a header region, a sidebar, and the main body.)
- Description:**
Optional text area where you can describe the purpose of this page. Currently blank.
- Dynamic Actions toggle:**
Enable page-level dynamic actions for the Salesforce mobile app → lets you control actions dynamically (which buttons users see, based on conditions).

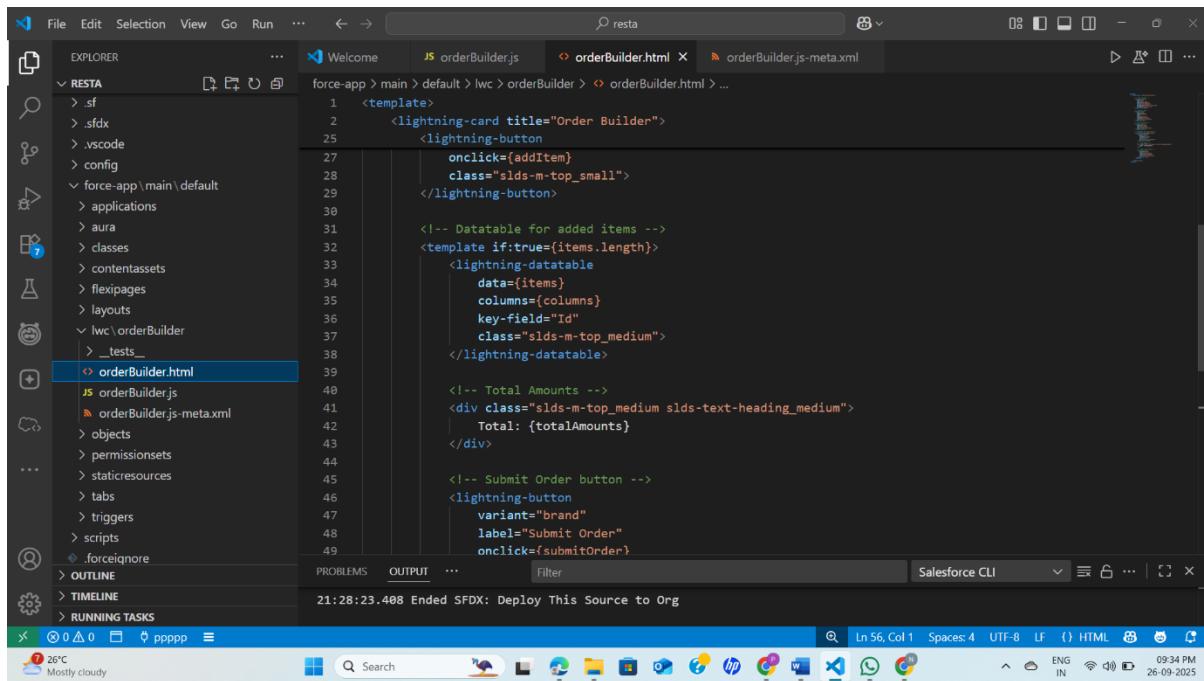


After creating the lightning app builder I have created the orderBuilder and deployed that to the org



The screenshot shows the Visual Studio Code interface with the following details:

- Explorer View:** Shows the project structure under the RESTA folder, including .sf, .sfdx, .vscode, config, force-app\main\default, applications, aura, classes, contentassets, flexipages, layouts, lwc\orderBuilder, _tests_, orderBuilder.html, orderBuilder.js, and orderBuilder.js-meta.xml.
- Editor View:** Displays the code for `orderBuilder.html`. The code includes a lightning-card component with a menu selection, a lightning-input component for quantity, and a lightning-button for adding items. It also includes a lightning-data-table for displaying added items and a lightning-button for submitting the order.
- Status Bar:** Shows the message "21:28:23.408 Ended SFDX: Deploy This Source to Org".
- System Tray:** Shows the date and time as 26-09-2025, and the weather as 26°C Mostly cloudy.



The screenshot shows the Visual Studio Code interface with the following details:

- Explorer View:** Shows the project structure under the RESTA folder, including .sf, .sfdx, .vscode, config, force-app\main\default, applications, aura, classes, contentassets, flexipages, layouts, lwc\orderBuilder, _tests_, orderBuilder.html, orderBuilder.js, and orderBuilder.js-meta.xml.
- Editor View:** Displays the code for `orderBuilder.html`. The code includes a lightning-card component with a menu selection, a lightning-input component for quantity, and a lightning-button for adding items. It also includes a lightning-data-table for displaying added items and a lightning-button for submitting the order. The code is identical to the previous screenshot but includes additional lines for the submit button and total amounts.
- Status Bar:** Shows the message "21:28:23.408 Ended SFDX: Deploy This Source to Org".
- System Tray:** Shows the date and time as 26-09-2025, and the weather as 26°C Mostly cloudy.

The screenshot shows the Visual Studio Code interface with the following details:

- Explorer View:** Shows the project structure under the RESTA folder, including .sf, .sfdx, .vscode, config, force-app\main\default, lwc\orderBuilder, and _tests_. The orderBuilder.html file is selected.
- Editor View:** Displays the code for orderBuilder.html. The code is an LWC component template. It includes a lightning-card with a title "Order Builder". Inside, there's a template if block for items, a comment for total amounts, a div for the total amount, a lightning-button for submit, and a closing lightning-card.
- Bottom Status Bar:** Shows the status "21:28:23.408 Ended SFDX: Deploy This Source to Org".
- System Tray:** Shows the date and time as 26-09-2025, 09:34 PM.

The screenshot shows the Visual Studio Code interface with the following details:

- Explorer View:** Shows the project structure under the RESTA folder, including .sf, .sfdx, .vscode, config, force-app\main\default, lwc\orderBuilder, and _tests_. The orderBuilder.js file is selected.
- Editor View:** Displays the code for orderBuilder.js. It imports LightningElement, track, createRecord, and various schema fields from @salesforce/schema. It defines a class OrderBuilder extending LightningElement with hardcoded menu items and tracks items.
- Bottom Status Bar:** Shows the status "21:28:23.408 Ended SFDX: Deploy This Source to Org".
- System Tray:** Shows the date and time as 26-09-2025, 09:34 PM.

```
force-app > main > default > lwc > orderBuilder > JS orderBuilder.js > ...
15  export default class OrderBuilder extends LightningElement {
27      @track totalAmounts = 0;
28
29      quantity = 1;
30      selectedMenuItem;
31
32      columns = [
33          { label: 'Item', fieldName: 'Name', type: 'text' },
34          { label: 'Qty', fieldName: 'Quantity', type: 'number' },
35          { label: 'Unit Price', fieldName: 'UnitPrice', type: 'currency' },
36          { label: 'Line Total', fieldName: 'LineTotal', type: 'currency' }
37      ];
38
39      handleSelect(event) {
40          this.selectedMenuItem = event.detail.value;
41      }
42
43      handleQty(event) {
44          this.quantity = parseInt(event.detail.value, 10) || 1;
45      }
46
47      addItem() {
48          const selected = this.menuItems.find(m => m.Id === this.selectedMenuItem);
49          if (!selected) {
50              alert('Please select a menu item.');
51              return;
52          }
53
54          const newItem = {
55              Id: Date.now().toString(),
56              Name: selected.Name,
57              MenuItemId: selected.Id,
58              Quantity: this.quantity,
59              UnitPrice: selected.Price__c,
60              LineTotal: this.quantity * selected.Price__c
61          };
62
63          this.items = [...this.items, newItem];
64          this.computeTotal();
65      }
66
67      computeTotal() {
68          this.totalAmounts = this.items.reduce((sum, item) => sum + item.LineTotal, 0);
69      }
70
71      async submitOrder() {
72          if (this.items.length === 0) {
73              alert('Add at least one item to submit order.');
74              return;
75          }
76      }
77  }
78
79  
```

PROBLEMS OUTPUT ... Filter

21:28:23.408 Ended SFDX: Deploy This Source to Org

Salesforce CLI

Ln 117, Col 1 Spaces: 4 UTF-8 LF () JavaScript

26-09-2025

```
force-app > main > default > lwc > orderBuilder > JS orderBuilder.js > ...
47      addItem() {
48          const selected = this.menuItems.find(m => m.Id === this.selectedMenuItem);
49          if (!selected) {
50              alert('Please select a menu item.');
51              return;
52          }
53
54          const newItem = {
55              Id: Date.now().toString(),
56              Name: selected.Name,
57              MenuItemId: selected.Id,
58              Quantity: this.quantity,
59              UnitPrice: selected.Price__c,
60              LineTotal: this.quantity * selected.Price__c
61          };
62
63          this.items = [...this.items, newItem];
64          this.computeTotal();
65      }
66
67      computeTotal() {
68          this.totalAmounts = this.items.reduce((sum, item) => sum + item.LineTotal, 0);
69      }
70
71      async submitOrder() {
72          if (this.items.length === 0) {
73              alert('Add at least one item to submit order.');
74              return;
75          }
76      }
77  }
78
79  
```

PROBLEMS OUTPUT ... Filter

21:28:23.408 Ended SFDX: Deploy This Source to Org

Salesforce CLI

Ln 117, Col 1 Spaces: 4 UTF-8 LF () JavaScript

26-09-2025

The screenshot shows the Visual Studio Code interface with the 'orderBuilder.js' file open in the center editor tab. The file contains JavaScript code for a LightningElement named 'OrderBuilder'. The code includes two main sections: creating an 'Orders__c' record and creating multiple 'Order_Items__c' records. It uses async/await syntax and the Salesforce SFDX API. The code editor has syntax highlighting for JavaScript and Salesforce-specific objects like 'TOTAL_FIELD' and 'STATUS_FIELD'. The status bar at the bottom indicates the file was deployed successfully at 21:28:23.408.

```
force-app > main > default > lwc > orderBuilder > JS orderBuilder.js > ...
15  export default class OrderBuilder extends LightningElement {
71    async submitOrder() {
77      try {
78        // 1. Create Orders__c record
79        const ordersFields = {};
80        ordersFields[TOTAL_FIELD.fieldApiName] = this.totalAmounts;
81        ordersFields[STATUS_FIELD.fieldApiName] = 'Placed';
82
83        const ordersRec = await createRecord({
84          apiName: ORDERS_OBJECT.objectApiName,
85          fields: ordersFields
86        });
87
88        // 2. Create Order_Items__c records
89        for (let it of this.items) {
90          const itemFields = {};
91          itemFields[MENU_LOOKUP.fieldApiName] = ordersRec.id;
92          itemFields[ITEM_LOOKUP.fieldApiName] = it.menuId;
93          itemFields[QTY_FIELD.fieldApiName] = it.Quantity;
94          itemFields[UNIT_PRICE_FIELD.fieldApiName] = it.UnitPrice;
95
96          await createRecord({
97            apiName: ORDER_ITEMS_OBJECT.objectApiName,
98            fields: itemFields
99          });
100
101      }
102
103      alert('Order submitted successfully!');
104
105      // Reset
106      this.items = [];
107      this.totalAmounts = 0;
108      this.quantity = 1;
109      this.selectedMenuItem = null;
110
111    } catch (error) {
112      alert('Error: ' + error.body.message);
113    }
114  }
115
116
117
```

This screenshot is identical to the one above, showing the 'orderBuilder.js' file in VS Code. The cursor is positioned on the closing brace of the 'submitOrder' function on line 113. The status bar at the bottom again shows the deployment message from 21:28:23.408.

```
force-app > main > default > lwc > orderBuilder > JS orderBuilder.js > OrderBuilder > submitOrder
15  export default class OrderBuilder extends LightningElement {
71    async submitOrder() {
77      try {
78        // 1. Create Orders__c record
79        const ordersFields = {};
80        ordersFields[TOTAL_FIELD.fieldApiName] = this.totalAmounts;
81        ordersFields[STATUS_FIELD.fieldApiName] = 'Placed';
82
83        const ordersRec = await createRecord({
84          apiName: ORDERS_OBJECT.objectApiName,
85          fields: ordersFields
86        });
87
88        // 2. Create Order_Items__c records
89        for (let it of this.items) {
90          const itemFields = {};
91          itemFields[MENU_LOOKUP.fieldApiName] = ordersRec.id;
92          itemFields[ITEM_LOOKUP.fieldApiName] = it.menuId;
93          itemFields[QTY_FIELD.fieldApiName] = it.Quantity;
94          itemFields[UNIT_PRICE_FIELD.fieldApiName] = it.UnitPrice;
95
96          await createRecord({
97            apiName: ORDER_ITEMS_OBJECT.objectApiName,
98            fields: itemFields
99          });
100
101      }
102
103      alert('Order submitted successfully!');
104
105      // Reset
106      this.items = [];
107      this.totalAmounts = 0;
108      this.quantity = 1;
109      this.selectedMenuItem = null;
110
111    } catch (error) {
112      alert('Error: ' + error.body.message);
113    }
114  }
115
116
117
```

The screenshot shows the Visual Studio Code interface with the following details:

- Explorer:** Shows the project structure under "RESTA". The file "orderBuilder.js-meta.xml" is selected.
- Editor:** Displays the XML code for "orderBuilder.js-meta.xml".
- Terminal:** Shows the command "21:28:23.408 Ended SFDX: Deploy This Source to Org".

The terminal window displays the deployment results:

```
21:28:19.899 Starting SFDX: Deploy This Source to Org
== Deployed Source
STATE    FULL NAME      TYPE          PROJECT PATH
Created   orderBuilder  LightningComponentBundle  force-app\main\default\lwc\orderBuilder\orderBuilder.html
Created   orderBuilder  LightningComponentBundle  force-app\main\default\lwc\orderBuilder\orderBuilder.js
Created   orderBuilder  LightningComponentBundle  force-app\main\default\lwc\orderBuilder\orderBuilder.js-meta.xml
21:28:23.408 Ended SFDX: Deploy This Source to Org
```

PHASE 7

Named Credentials

1. Label & Name

- o Label: PaymentGateway → Human-readable name
- o Name: PaymentGateway → API reference for Apex callouts

2. URL

- o External system endpoint URL, e.g., <https://api.paymentgateway.com>
- o This will be the base URL for all callouts using this Named Credential

3. Identity Type

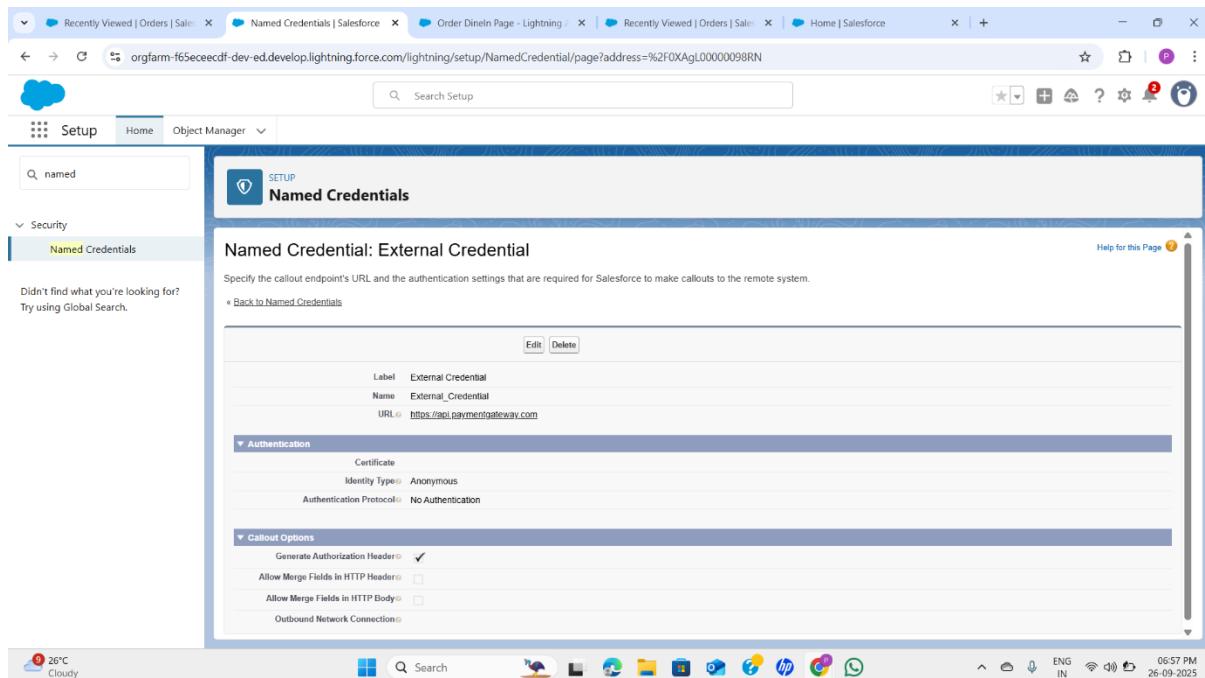
- o Choose **Named Principal** → Single set of credentials used by all users
- o Optionally, you could choose **Per User** if each Salesforce user needs separate authentication

4. Authentication

- o If you have created an **External Credential** (OAuth, AWS SigV4), select it here
- o Otherwise, choose **Password Authentication** and enter username/password or token

5. Generate Authorization Header (optional)

- o For OAuth or token-based auth, Salesforce automatically adds the Authorization header to requests



Use Platform Events for Real-Time Notifications

Notify external systems when orders are updated

1. Create a Platform Event

- Setup → Platform Events → New Platform Event
- Name: OrderStatusEvent
- Fields:
 - OrderId (Text)
 - Status (Picklist: Pending, Completed, Cancelled)
 - CustomerId (Text)

- 2. □ Platform Events are a **Salesforce event-driven messaging mechanism**.
- 3. □ They allow **Salesforce to notify external systems** (POS, delivery platforms, analytics tools) **in real-time** when something changes in Salesforce.
- 4. □ Think of it as **Salesforce “publishing” an event** that subscribers can react to immediately.

□ Real-Time Communication

- External systems can subscribe via **CometD protocol** or **Streaming API**.
- Notifications are near-instant, no need for polling Salesforce.

□ Custom Event Structure

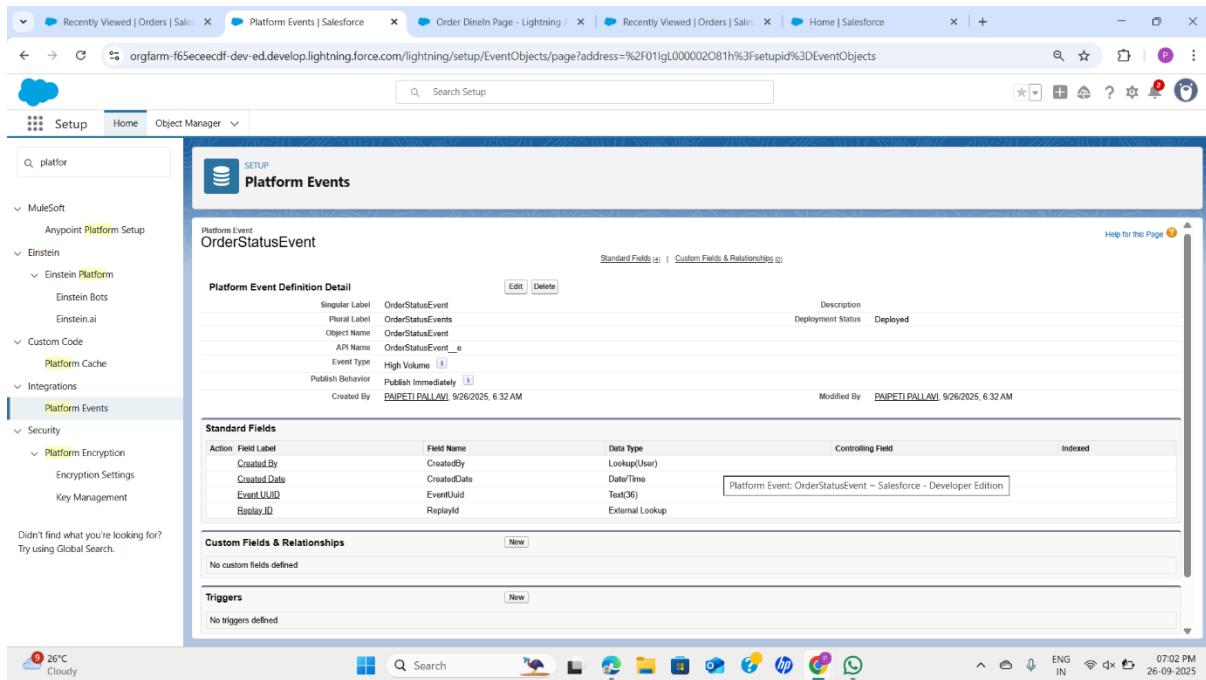
- You define which fields are sent in the event.
- Only necessary data is included, making it lightweight.

□ Reliable Delivery

- Events can be replayed if an external system was temporarily disconnected.

□ Supports Standard & Custom Subscribers

- Subscribers can be **Apex triggers, Lightning components, external middleware**, or other apps.



Change Data Capture

Automatically streams record changes for standard/custom objects

1. Setup → Change Data Capture → Select Object (e.g., Order__c)
 2. Enable CDC → External system can subscribe for **insert, update, delete** events.
1. **Change Data Capture (CDC)** is a Salesforce **event-driven mechanism** that automatically **streams changes of Salesforce records** to external systems in real-time.
 2. It is **especially useful for integrations**, so external systems (like POS, ERP, or analytics platforms) **stay in sync** with Salesforce without constant polling.

Real-Time Streaming

- Sends events immediately when a record is **created, updated, deleted, or undeleted**.
- Reduces the need for batch jobs or polling APIs.

Works with Standard & Custom Objects

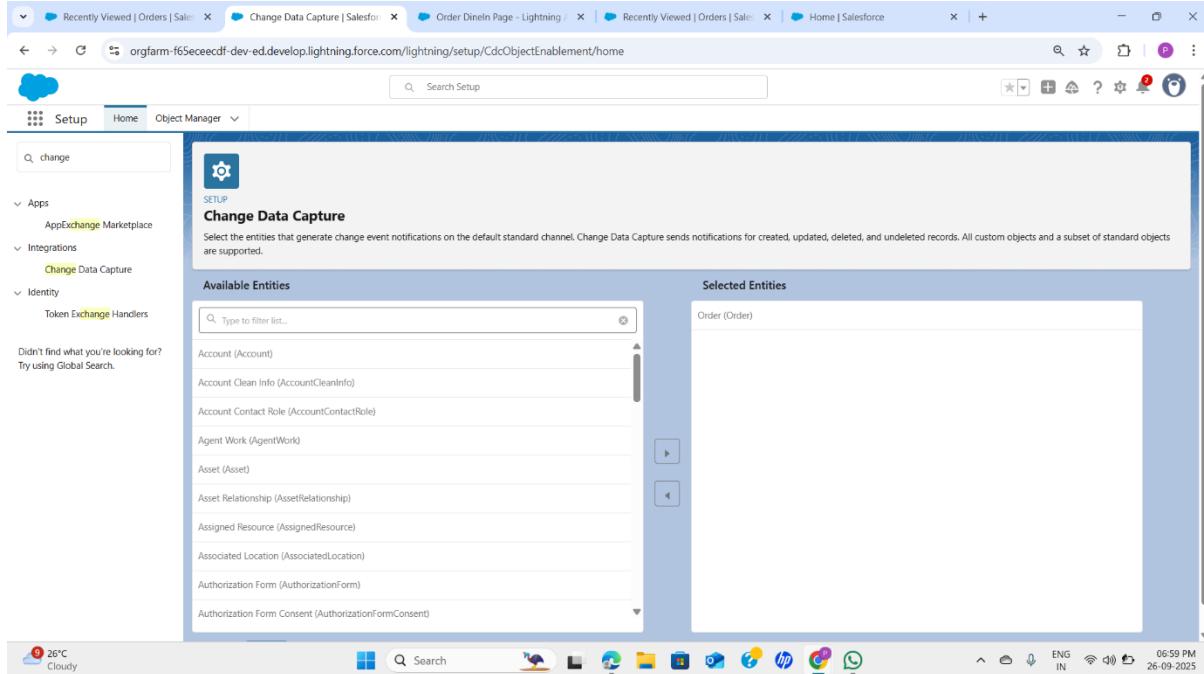
- Example objects:
 - Standard: Account, Contact, Order
 - Custom: Order__c, Booking__c

Reliable & Secure

- Events are delivered via **CometD protocol** or **Streaming API**.
- Can be **subscribed to by external systems** using secure authentication.

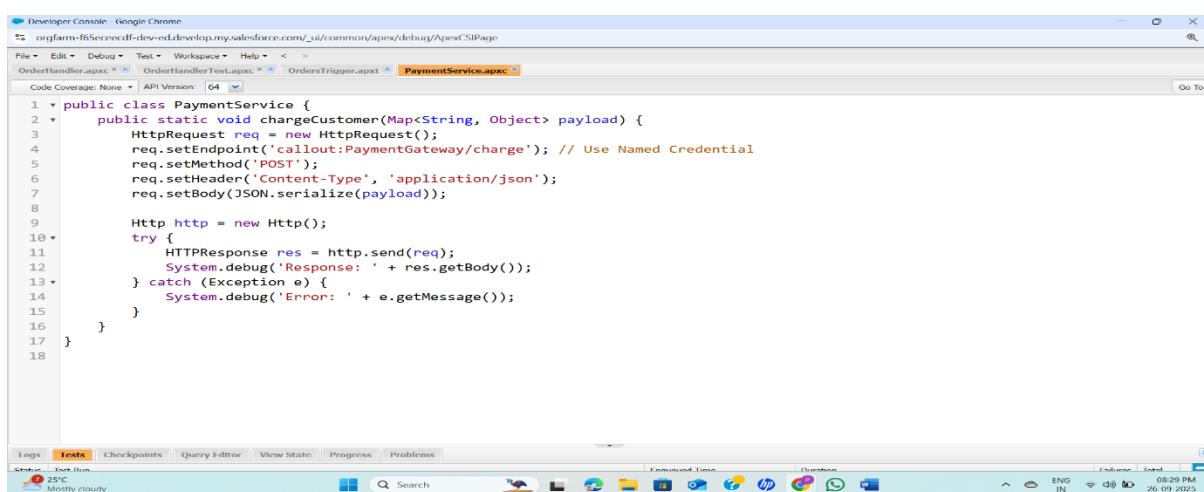
Supports Bulk Changes

- Multiple changes can be sent in a single event batch, optimizing API usage.



Apex Callouts Using Named Credentials

- An **Apex callout** is when Salesforce (using Apex code) makes an **HTTP request** to an external system's API (e.g., Payment Gateway, Twilio, POS).
- Used for:
 - Sending data (POST) → e.g., payment requests
 - Fetching data (GET) → e.g., order status from POS
 - Updating data (PATCH/PUT)
 - Deleting data (DELETE)



PHASE 8

Data Export

Go to Setup

- Click the gear icon () in the top-right corner of Salesforce.
- Click **Setup**.

Search for Data Export

- In the Quick Find box (top-left), type: Data Export.
- Click **Data Export** under *Data Management*.

Choose Export Type

- **Export Now** → one-time backup.
- **Schedule Export** → automatically back up weekly/monthly.
 - Only available in certain Salesforce editions.

Select Objects to Backup

- Check “Select All” to include all standard and custom objects.
- **Include attachments, documents, Salesforce Files** if needed.

Start the Export

- Click **Start Export**.
- Salesforce will prepare a ZIP file with your data.
- **Note:** This may take hours depending on how much data you have.

Download & Store Safely

- You will get an email when the export is ready.
- Go to the link, download the ZIP file.

- Store it securely: local machine, external drive, or cloud storage.

The screenshot shows the 'Data Export' page in Salesforce. At the top, there's a search bar with 'data ex' and navigation links for 'Setup', 'Home', and 'Object Manager'. Below the header, a sidebar on the left has 'Data' and 'Data Export' selected. A message says 'Didn't find what you're looking for? Try using Global Search.' The main content area is titled 'Monthly Export Service'. It displays a yellow banner stating 'Next scheduled export: A data export is currently in progress for your organization.' Below this, there are buttons for 'Export Now' and 'Schedule Export'. Underneath, it shows export details: 'Scheduled By: PAMELA LALI', 'Schedule Date: 9/26/2025', and 'Export File Encoding: ISO-8859-1 (General US & Western European, ISO-LATIN-1)'. The bottom right corner of the page has a link 'Help for this Page'.

- After this export I got the email like this.

The screenshot shows an email in the inbox. The subject is 'Your Organization Data Export has completed - MyRestaurant Pvt Ltd'. The sender is 'Do not reply <noreply@salesforce.com>' with a note 'to me'. The email body says 'The export of your organization's data has been completed. Please click on the following link within the next 48 hours to receive the export.' followed by a link: <https://orgfarm-f65ecccdf-dev-ed.develop.my.salesforce.com/ui/setup/export/DataExportPage/d>. The footer of the email includes 'Thank you, Salesforce' and standard reply, forward, and smiley face buttons. The top right of the email view shows '1 of 1,066' and various icons for printing, deleting, and navigating.

Data Import

Go to Setup → Data Import Wizard

- Click the gear icon → Setup → Quick Find: Data Import Wizard → Launch Wizard.

Choose Object

- Select the object you want to import (e.g., Contacts, Accounts, Orders).

Select Operation

- **Add New Records** → only insert.
 - **Update Existing Records** → update by matching Salesforce ID or External ID.
 - **Upsert** → create or update using External ID (recommended if updating historical data).

Upload CSV File

- Prepare CSV: include required fields and External ID fields if needed.
 - Click **Choose CSV file** and upload it.

Map Fields

- Salesforce will attempt to auto-map fields from your CSV.
 - Verify all important fields (Name, Email, Order Number, Dates, etc.) are mapped correctly.

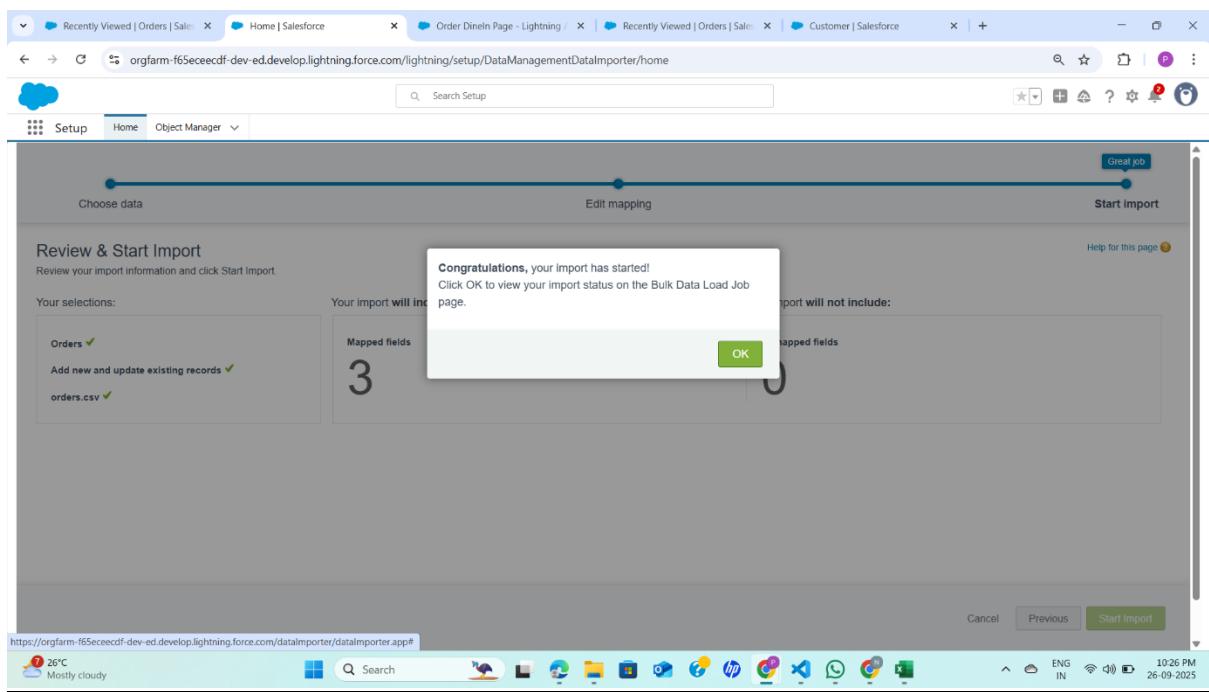
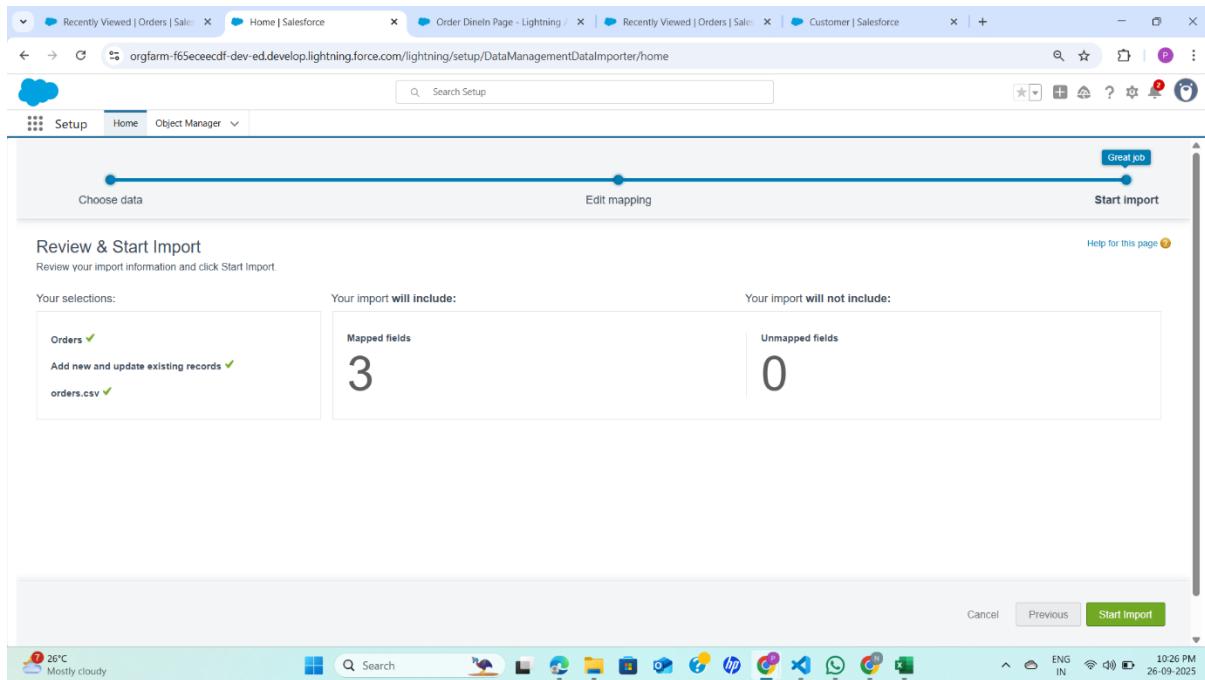
Run Import

- Click **Start Import**.
 - Wait for it to finish; you can **monitor status** in Data Import Wizard.

Check Results

- Download **Success** and **Error** CSVs.
 - Fix any errors (required fields missing, duplicate rule conflicts, invalid picklist values) and re-import if needed.

I have created the csv file for the orders



After importing I have received the email like this

Salesforce import of "orders.csv" has finished. 2 rows were processed.

to me

Your Orders import is complete. Here are your results:

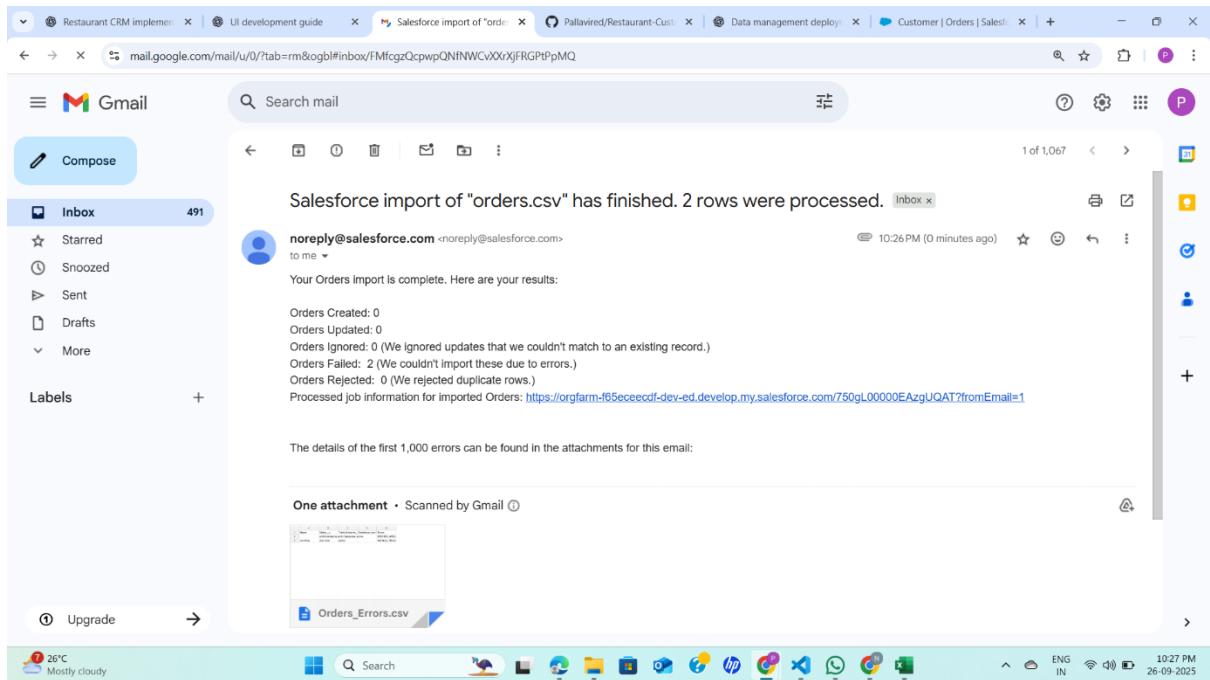
Orders Created: 0
Orders Updated: 0
Orders Ignored: 0 (We ignored updates that we couldn't match to an existing record.)
Orders Failed: 2 (We couldn't import these due to errors.)
Orders Rejected: 0 (We rejected duplicate rows.)

Processed job information for imported Orders: <https://orgfarm-f65ecccdf-dev-ed.develop.my.salesforce.com/750gL00000EAzgUQAT?fromEmail=1>

The details of the first 1,000 errors can be found in the attachments for this email:

One attachment • Scanned by Gmail

Orders_Errors.csv



PHASE 9

REPORTS AND DASHBOARDS:

What is a Report in Salesforce?

A **Report** is a list of records that meet certain criteria, displayed in rows and columns. It helps you **analyze data, track performance, and make decisions**.

For example:

- A report on **Orders** could show: *Order Number, Account, Order Amount, Status*.
- A report on **Customers** could show: *Customer Name, Total Purchases, Loyalty Points*.

Key Elements of a Report

1. **Report Type**
 - Defines *which object(s)* you're reporting on (e.g., Orders, Accounts, Contacts, or Orders with Order Items).
 - Determines what fields are available to add in your report.
2. **Columns**
 - The fields you choose to display (e.g., Order Number, Account Name, Order Start Date).
3. **Filters**
 - Narrow down the data (e.g., *Status = Completed, Created Date = Last 30 Days*).
4. **Grouping**
 - Organize records by field values (e.g., group Orders by *Status* or *Account*).
5. **Summaries**
 - Totals, averages, minimum, maximum values on numeric fields (e.g., total Order Amount).
6. **Report Formats** (in Lightning)
 - **Tabular** → Simple list (like an Excel table).
 - **Summary** → Grouped rows, with subtotals.
 - **Matrix** → Group rows *and* columns (like a pivot table).
 - **Joined** → Multiple report blocks combined for comparison.

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

- Header:** The top navigation bar includes tabs for "Recently Viewed", "Orders", "Home | Salesforce", "Order DineIn Page - Lightning", "Report Builder | Salesforce", and "Created by Me | Reports".
- Page Title:** The page title is "Created by Me | Reports".
- Search Bar:** A search bar at the top right contains the placeholder "Search...".
- Top Navigation:** The main navigation bar includes links for "Navss resto", "Orders", "Bookings", "Dashboards", "Reports" (selected), "Loyalty Accounts", "Order Items", and "More".
- Left Sidebar:** A sidebar on the left lists report categories: "RECENT", "Created by Me" (selected), "Private Reports", "Public Reports", and "All Reports". It also lists "FOLDERS" and "All Folders".
- Report List:** The main content area displays a table of reports. The columns are "Report Name", "Description", "Folder", "Created By", "Created On", and "Subscribed".

Report Name	Description	Folder	Created By	Created On	Subscribed
New Bookings	Private Reports	PAIPETI PALLAVI	9/26/2025, 10:24 AM	▼	
New Orders Report	Private Reports	PAIPETI PALLAVI	9/26/2025, 10:18 AM	▼	
- Bottom Navigation:** The bottom navigation bar includes icons for Home, Search, and various system functions like File, Help, and User.
- System Status:** The bottom left shows system status: "6 26°C Mostly cloudy".
- Page Footer:** The bottom right shows the date and time: "26-09-2025".

Dashboards

- Purpose:

This dashboard provides insights into how enablement activities (such as training, guidance, or support programs) drive business outcomes. It is designed to be the central place for monitoring enablement analytics.

- **Features:**

- Tracks business impact of enablement strategies.
 - Shows KPIs aligned to business performance.
 - Helps leadership and managers evaluate enablement ROI.

- **Notes:** This dashboard should not be deleted. If changes are required, it must be duplicated before editing.

Recently Viewed | Orders | Sales | Home | Salesforce | Order Dinelin Page - Lightning | Report Builder | Salesforce | All Dashboards | Dashboards | +

orgfarm-f65eceeccdf-dev-ed.develop.lightning.force.com/lightning/o/

Dashboard/home?queryScope=everything

Search... Star 2 +1 ? ! IN

Navss resto Orders Bookings Dashboards Reports Loyalty Accounts Order Items * JohnDoe | Customer

Dashboards All Dashboards 5 items

Search all dashboards... New Dashboard New Folder

DASHBOARDS	Dashboard Name	Description	Folder	Created By	Created On	Subscribed
Recent						
Created by Me	Enablement Dashboard	View data on how Enablement helps drive your business outcomes. This is your main dashboard for all Enablement analytics. Don't delete it. If you want to make changes to this dashboard, duplicate it.	Enablement Dashboard Spring '24	Automated Process	9/18/2025, 12:17 PM	
Private Dashboards						
All Dashboards						
FOLDERS						
All Folders	Enablement Dashboard	View data on how Enablement helps drive your business outcomes. This is your main dashboard for all Enablement analytics. Don't delete it. If you want to make changes to this dashboard, duplicate it.	Enablement Dashboard Summer '24	Automated Process	9/18/2025, 12:17 PM	
Created by Me						
Shared with Me						
FAVORITES						
All Favorites	Kitchen	Items related to the kitchen	Enablement Dashboard Summer '24	PAIPETI PALLAVI	9/26/2025, 10:30 AM	
	Manager	Manager dashboard related to the restaurant menu and the cost	Private Dashboards	PAIPETI PALLAVI	9/26/2025, 10:28 AM	
	Pallavi		Private Dashboards	PAIPETI PALLAVI	9/26/2025, 10:26 AM	

6 26°C Mostly cloudy ENG IN 🔍 11:01 PM 26-09-2025