

Project Name : Smart Task And Email Summarizer For Executives

Phase 1: Problem Understanding & Industry Analysis

Project Title: Smart Tasks & Email Summarizer for Executives

1. Requirement Gathering

Objective: Understand what executives need in terms of email and task summarization.

Requirements:

- Summarize long emails into short key points.
- Generate action items (tasks) automatically from text.
- Provide priority tagging (urgent, important, informative).
- Integrate with Salesforce Inbox, Outlook, Gmail.
- Provide summaries on Salesforce mobile app for executives on the go.

Salesforce Tools: Einstein GPT, Apex triggers, Flows, Salesforce Email Integration APIs.

2. Stakeholder Analysis

Primary Stakeholders:

- Executives/Managers: Need quick insights instead of reading full reports.
- Sales Teams: Want summarized client communications.
- HR/Operations: Save time reading lengthy policy or vendor updates.

Secondary Stakeholders:

- IT/Admins: Configure Salesforce setup, maintain security.
- Developers: Build custom summarizer logic using Apex & LLMs.

Output: Clear view of who benefits and who maintains the solution.

3. Business Process Mapping

Current Flow (Without Summarizer):

- Executives receive 100+ emails daily.
- They spend 2–3 hours reading through reports.
- Important action points are often missed.

Proposed Flow (With Summarizer):

- Emails sync into Salesforce.
- Summarizer extracts summary + action items.
- Tasks are created automatically in Salesforce.
- Executives see short digest instead of full text.

4. Industry-specific Use Case Analysis

Finance: Summarize compliance reports, auto-create tasks for risk alerts.

Healthcare: Summarize patient case updates, highlight urgent follow-ups.

IT/Consulting: Summarize project updates & client escalations.

Sales/CRM: Summarize customer emails & create follow-up tasks.

This proves the solution is not just generic but industry-relevant.

5. AppExchange Exploration

Existing Tools:

- Einstein Activity Capture (for syncing emails).
- Summarizer Apps (some third-party AI summarizers).
- TaskRay (task management).

Gap: Most tools provide either email syncing or task management, but not AI-based summarization + automatic task creation inside Salesforce.

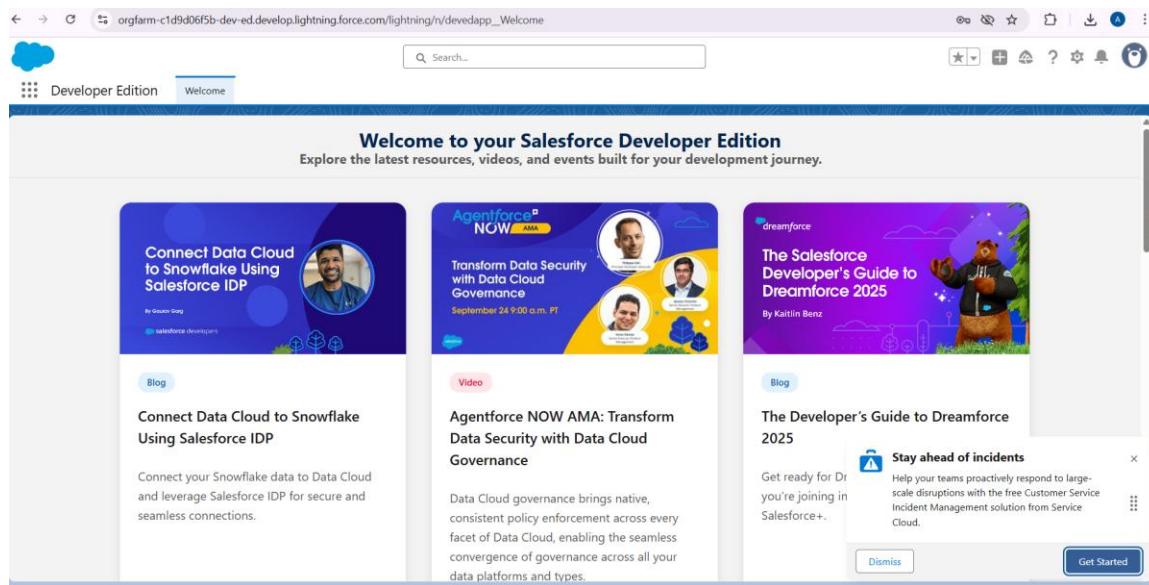
Conclusion: This project fills this gap.

Phase 2: Org Setup & Configuration - Smart Task & Email Summarizer for Executives

Purpose: This document is a step-by-step hands-on guide to set up a Salesforce Org for your project, with clear instructions, checklists, and placeholders where you should attach screenshots. Follow each step in order to minimize errors.

A. Prerequisites (Before you begin)

- A valid Salesforce account (Developer Edition or company org).
- Admin access in the org you will configure.
- Access to the email account(s) you'll link for testing (Gmail/Outlook).
- PC with a browser (Chrome recommended), and ability to install Salesforce CLI (optional but recommended).
- A directory on your PC to store project screenshots named: ProjectScreenshots_Phase2/.



1. Salesforce Editions — choose the right edition

Why this matters:

- Different Salesforce editions (Essentials, Professional, Enterprise, Unlimited) provide different features and limits. For most development and advanced automation (APIs, Sandboxes, Change Sets), Enterprise or higher is recommended; Developer Edition is fine for learning and the project prototype.

How to check your edition:

1. In Salesforce Setup, search for and open Company Information.
2. Look for Organization Edition and Organization ID.

What to capture (screenshot):

- [Insert screenshot: 1_CompanyInformation.png] — capture the Organization Edition line.

Phone	
Fax	
Default Locale	English (United States)
Default Language	English
Default Time Zone	(GMT-07:00) Pacific Daylight Time (America/Los_Angeles)
Currency Locale	English (United States) - USD
Used Data Space	342 KB (7%) [View]
Used File Space	17 KB (0%) [View]
API Requests, Last 24 Hours	39 (15,000 max)
Streaming API Events, Last 24 Hours	0 (10,000 max)
Restricted Logins, Current Month	0 (0 max)
Salesforce.com Organization ID	00DgLO0000C727q
Organization Edition	Developer Edition
Instance	CAN98
Modified By	OrgFarm EPIC , 9/25/2025, 6:14 AM

2. Company Profile Setup (Company Information)

Steps (hands-on):

1. Setup → Quick Find → Company Information.
2. Click Edit if you need to update address, phone, default locale, default time zone.
3. Confirm the Default Locale, Language, and Currency, as these affect date/number formatting in reports.

Tips:

- Use the legal company name and exact address to avoid confusion in audit logs.

Screenshot to attach:

- [Insert screenshot: 2_CompanyInformation_Edit.png] (after you click Edit showing filled fields)

The screenshot shows the Salesforce Setup interface for the organization 'EPIC OrgFarm'. The left sidebar includes links for Home, Administrator, and various management sections. The main content area displays 'Organization Detail' with fields for Organization Name (EPIC OrgFarm), Primary Contact (OrgFarm EPIC), Address (United States), Fiscal Year Starts In (January), and various checkboxes for currency, data translation, newsletter, and admin newsletter. It also shows locale formats (ICU). On the right, there's a summary of system usage including API requests, streaming API events, restricted logins, and instance details (e.g., Salesforce.com Organization ID: 00Dg1.00000C727q, Organization Edition: Developer Edition, Instance: CAN98). Navigation links at the bottom include 'User Licenses Help'.

3. Business Hours & Holidays

Purpose: Business Hours influence Escalation Rules and case assignment timing.

Steps:

1. Setup → Quick Find → Business Hours → New.
2. Create your organization's primary business hours (name, time zone, open/close times, holidays).
3. Setup → Quick Find → Holidays → New to add public holidays.

Screenshot to attach:

- [Insert screenshot: 3_BusinessHours_List.png] (list view)
- [Insert screenshot: 3_BusinessHours_Edit.png] (detail of created hours)

The screenshot shows the 'Business Hours' list in the Salesforce Setup. The header includes a 'SETUP' icon and the title 'Business Hours'. Below is a section titled 'Organization Business Hours' with a table. The table has columns for Action, Business Hours Name, Active, Time Zone, and Default. Two rows are shown: 'Default' (Active, checked, Time Zone: (GMT-07:00) Pacific Daylight Time (America/Los_Angeles), Default checked) and 'Primary Business Hours' (Active, unchecked, Time Zone: (GMT+05:30) India Standard Time (Asia/Kolkata), Default unchecked). A 'New Business Hours' button is at the top right of the table. A navigation bar at the bottom includes letters A-Z and 'All'.

Holidays

Holidays are dates and times at which business hours are suspended. Business hours are the days and hours that your support team is available.

Action	Holiday Name	Description	Date and Time
Edit Del	Dwali		9/25/2025 All Day
Edit Del	Independence Day		9/25/2025 All Day

Elapsed Holidays

No records to display

4. Fiscal Year Settings

Purpose: Set reporting quarters and fiscal calculations correctly.

Steps:

1. Setup → Quick Find → Fiscal Year.
2. If your organization follows a standard Gregorian 12-month fiscal year, use the Standard Fiscal Year option and set the start month.
3. If your business uses non-standard periods (13-week quarters or non-monthly boundaries), use Custom Fiscal Year (note: custom fiscal years are irreversible and can break some managed packages). Document this before changing.

Screenshot to attach:

- [Insert screenshot: 4_FiscalYear_Settings.png]

Fiscal Year Information

Your organization can change the fiscal year start month, and specify whether the fiscal year name is set to the starting or ending year. For example, if your fiscal year starts in April 2025 and ends in March 2026, your Fiscal Year setting can be either 2025 or 2026.

Change Fiscal Year Period

Name: EPIC OrgFarm

Fiscal Year Start Month: April

Fiscal Year is Based On: The ending month

5. User Setup & Licenses

Purpose: Create users with the correct license types for your project's needs.

Steps:

1. Setup → Quick Find → Users → Users → New User (or Activate existing users).

2. Choose the appropriate User License (Salesforce, Salesforce Platform, Chatter Free, etc.).
3. Fill in required fields: First Name, Last Name, Alias, Email, Username (must be unique across Salesforce), Nickname, Role, Profile, and Active checkbox.

Best practices:

- Reserve full Salesforce licenses for admins and power users.
- For testing, use Developer Edition or spare licenses in a sandbox.

Screenshot to attach:

- [Insert screenshot: 5_NewUser_Form.png] (after filling the form but before saving).

The screenshot shows the Salesforce 'Users' setup page. At the top, there's a navigation bar with 'SETUP' and 'Users'. Below it, the user profile for 'Alias Kim' is displayed. The 'User Detail' section contains the following information:

Field	Value	Role	
Name	Alias Kim	Role	
Alias	akim	User License	Salesforce
Email	manoharitelang18@gmail.com [Verify]	Profile	System Administrator
Username	siddhideshmukh.dev+phase2@example.com	Active	<input checked="" type="checkbox"/>
Nickname	User17588107677825593989	Marketing User	<input type="checkbox"/>
Title		Offline User	<input type="checkbox"/>
Company		Knowledge User	<input type="checkbox"/>
Department		Flow User	<input type="checkbox"/>
Division		Service Cloud User	<input type="checkbox"/>
Address	GAJANAN NAGAR BAHADURA PLOT NO 19 Nagpur 440034 Maharashtra India	Site.com Contributor User	<input type="checkbox"/>
Time Zone	(GMT+05:30) India Standard Time (Asia/Kolkata)	Site.com Publisher User	<input type="checkbox"/>
Locale	English (India)	WDC User	<input type="checkbox"/>
Language	English	Mobile Push Registrations	<input type="checkbox"/> View
Delegated Approver		Data.com User Type	<input type="checkbox"/> View
Manager		Accessibility Mode (Classic Only)	<input type="checkbox"/> View

6. Profiles

Purpose: Profiles define baseline object and system permissions.

Steps (create/adjust profile):

1. Setup → Quick Find → Profiles → click the profile name (or New Profile to clone an existing one).
2. Configure Object Settings, Field-Level Security, Login Hours, and IP Ranges as needed.
3. Keep profiles minimal: give only baseline permissions necessary.

Screenshot to attach:

- [Insert screenshot: 6_Profiles_List.png]
- [Insert screenshot: 6_Profile_ObjectSettings.png]

Profile
Executive_Profile

Users with this profile have the permissions and page layouts listed below. Administrators can change a user's profile by editing that user's personal information.

If your organization uses Record Types, use the Edit links in the Record Type Settings section below to make one or more record types available to users with this profile.

[Login IP Ranges](#) | [Enabled Apex Class Access](#) | [Enabled Visualforce Page Access](#) | [Enabled External Data Source Access](#) | [Enabled Named Credential Access](#) | [Enabled External Credential Principal Access](#) | [Enabled Custom Metadata Type Access](#) | [Enabled Custom Setting Definitions Access](#) | [Enabled Flow Access](#) | [Enabled Service Presence Status Access](#) | [Enabled Custom Permissions](#)

Profile Detail	
Name	Executive_Profile
User License	Salesforce
Description	
Created By	Anjali Deshmukh, 9/25/2025, 7:47 AM
Modified By	Anjali Deshmukh, 9/25/2025, 12:49 PM

Page Layouts

Standard Object Layouts	Global	Location Group Assignment	Location Group Assignment Layout
Email Application	Not Assigned [View Assignment]	Macro	Macro Layout [View Assignment]
Home Page Layout	Home Page Default [View Assignment]	Object Milestone	Object Milestone Layout [View Assignment]
Account	Account Layout [View Assignment]	Operating Hours	Operating Hours Layout [View Assignment]
Alternative Payment Method	Alternative Payment Method Layout [View Assignment]	Opportunity	Opportunity Layout [View Assignment]

password emails

[Edit](#) [Clone](#) [Delete](#) [View Users](#)

Login Hours

Day	Start Time	End Time
Sunday	All Day	All Day
Monday	8:30 PM PDT	5:30 AM PDT
Tuesday	8:30 PM PDT	5:30 AM PDT
Wednesday	8:30 PM PDT	5:30 AM PDT
Thursday	8:30 PM PDT	5:30 AM PDT
Friday	8:30 PM PDT	5:30 AM PDT
Saturday	8:30 PM PDT	5:30 AM PDT

Login IP Ranges

Action	IP Start Address	IP End Address	Description
Edit Del	47.11.14.83	47.11.14.93	

7. Roles (Role Hierarchy)

Purpose: Roles control record visibility via the role hierarchy.

Steps:

1. Setup → Quick Find → Roles → Set Up Roles.
2. Create a clear hierarchy that reflects your organization's reporting lines (e.g., CEO > VP Sales > Regional Manager > Sales Rep).
3. Assign users to roles when creating user records.

Tips:

- Keep the hierarchy focused on data visibility, not org chart details that don't affect record sharing.

Screenshot to attach:

- [Insert screenshot: 7_Roles_Hierarchy.png]

 SETUP

Roles

- [VP, Marketing](#) [Edit](#) | [Del](#) | [Assign](#)
 - └ [Add Role](#)
- [Marketing Team](#) [Edit](#) | [Del](#) | [Assign](#)
 - └ [Add Role](#)
- [VP, North American Sales](#) [Edit](#) | [Del](#) | [Assign](#)
 - └ [Add Role](#)
- [Director, Channel Sales](#) [Edit](#) | [Del](#) | [Assign](#)
 - └ [Add Role](#)
- [Channel Sales Team](#) [Edit](#) | [Del](#) | [Assign](#)
 - └ [Add Role](#)
- [Director, Direct Sales](#) [Edit](#) | [Del](#) | [Assign](#)
 - └ [Add Role](#)
- [Eastern Sales Team](#) [Edit](#) | [Del](#) | [Assign](#)
 - └ [Add Role](#)
- [Western Sales Team](#) [Edit](#) | [Del](#) | [Assign](#)
 - └ [Add Role](#)
- [VP](#) [Edit](#) | [Del](#) | [Assign](#)
 - └ [Add Role](#)
- [Manager](#) [Edit](#) | [Del](#) | [Assign](#)
 - └ [Add Role](#)
- [Executive](#) [Edit](#) | [Del](#) | [Assign](#)
 - └ [Add Role](#)
- [Developer](#) [Edit](#) | [Del](#) | [Assign](#)
 - └ [Add Role](#)

The screenshot shows the Salesforce Setup - Users page. At the top, there is a blue header bar with a user icon and the text "SETUP" and "Users". Below the header, the page title is "All Users" and there is a "Help for this Page" link with a question mark icon. A brief description states: "On this page you can create, view, and manage users." It also mentions: "To get more licenses, use the Your Account app. [Let's Go](#)". Below this, there is a "View" dropdown set to "All Users" with options to "Edit" or "Create New View". A navigation bar below the dropdown includes links for letters A through Z and an "Other" link, with "All" being the selected option. At the top right of the main content area, there are three buttons: "New User", "Reset Password(s)", and "Add Multiple Users". The main content is a table listing six user records:

Action	Full Name ↑	Alias	Username	Role	Active	Profile
Edit	Chatter Expert	Chatter	chatty.00dg 00000c727quab.hskqljzuhoo@chatter.salesforce.com	Chatter Admin	<input checked="" type="checkbox"/>	Ch Us
Edit	Deshmukh, Anjali	des	deshmukhanjali2004174@agentforce.com	Executive	<input checked="" type="checkbox"/>	Sy Ad
Edit	EPIC_OrgFarm	OEPIC	epic.9b778aa5a14@orgfarm.salesforce.com	Manager	<input checked="" type="checkbox"/>	Sy Ad
Edit	Kim_Alias	akim	siddhideshmukh.dev+phase2@example.com	Developer	<input checked="" type="checkbox"/>	Sy Ad
Edit	User_Integration	integ	integration@00dg 00000c727quab.com	VP	<input checked="" type="checkbox"/>	An Ck Int Us
Edit	User_..	sec	insightssecurity@00dal00000c727auab.com		<input checked="" type="checkbox"/>	An Ck ..

8. Permission Sets

Purpose: Permission Sets are additive permissions applied to users without changing their profile.

Steps:

1. Setup → Quick Find → Permission Sets → New.
2. Name the permission set (e.g., 'Summarizer_Admin_Access'), set License (if required).
3. Add granular permissions: Object Settings, Field Permissions, Apex Class Access, Visualforce Page Access, and System Permissions (e.g., 'Modify All Data' only if necessary).
4. Assign permission sets to users via the Permission Set Assignment related list on the user record or via the Permission Set Assign page.

Best practice:

- Use permission sets to grant temporary or additional rights; use profiles for baseline.

Screenshot to attach:

- [Insert screenshot: 8_PermissionSet_Detail.png]
- [Insert screenshot: 8_PermissionSet_Assign.png]

The screenshot shows the Salesforce Setup interface for managing Permission Sets. The top navigation bar includes 'SETUP' and the 'Permission Sets' icon. The main title is 'Permission Set' followed by the name 'Summarizer_Admin_Access'. On the right, there are links for 'Video Tutorial' and 'Help for this Page'.

Below the title is a search bar with placeholder 'Find Settings...' and several action buttons: 'Clone', 'Delete', 'Edit Properties', 'Manage Assignments', and 'View Summary'.

The breadcrumb navigation shows 'Permission Set Overview > Object Settings > Custom Summarizers'.

The 'Custom Summarizers' section has a tab labeled 'Tab Settings' with an 'Edit' button. Below it is a table with two columns: 'Available' and 'Visible'. The 'Visible' column contains three checkboxes: one is checked, and the other two are empty.

The 'Object Permissions' section lists various permissions with checkboxes for 'Enabled'. Most permissions have their checkboxes checked, except for 'View All Records' and 'Modify All Records' which are unchecked.

Permission Name	Enabled
Read	<input checked="" type="checkbox"/>
Create	<input checked="" type="checkbox"/>
Edit	<input checked="" type="checkbox"/>
Delete	<input checked="" type="checkbox"/>
View All Records	<input type="checkbox"/>
Modify All Records	<input type="checkbox"/>
Allow All Fields	<input type="checkbox"/>

Obj... 1 assignments were successful. X

... > PERMISSION SET 'SUMMARIZER_ADMIN_ACCESS' > MANAGE ASSIGNMENTS
Summarizer_Admin_Access

Assignment Summary

Full Name	User License	Expires On	Time Zone	Status
Alias Kim	Salesforce			Success

Developer Edition Welcome

Custom Summarizers Recently Viewed

0 items • Updated a few seconds ago Search this list...



Nothing to see here

There's nothing in your list yet. Try adding a new record.

9. Organization-Wide Defaults (OWD)

Purpose: OWD sets the baseline record visibility for objects and is the most restrictive layer in the sharing model. You should plan OWD carefully before creating many records.

Steps:

1. Setup → Quick Find → Sharing Settings.
2. Click Edit in the Organization-Wide Defaults area and set default access per object (Private, Public Read Only, Public Read/Write, Controlled by Parent).

Tips:

- Start with restrictive defaults (Private) and then open visibility using Role Hierarchy, Sharing Rules, and Permission Sets.

Screenshot to attach:

- [Insert screenshot: 9_SharingSettings_OWD.png]

Object	Default Internal Access	Default External Access	Grant Access Using Hierarchies
Lead	Public Read/Writer/Transfer	Private	✓
Account and Contract	Public Read/Write	Private	✓
Contact	Controlled by Parent	Controlled by Parent	✓
Order	Controlled by Parent	Controlled by Parent	✓

Sharing Settings

[Help for this Page](#)

This page displays your organization's sharing settings. These settings specify the level of access your users have to each others' data. Go to [Background Jobs](#) to monitor the progress of a change to an organization-wide default or a parallel sharing recalculation.

Object	Default Internal Access	Default External Access	Grant Access Using Hierarchies
Task	Public Read/Write	Private	✓

10. Sharing Rules

Purpose: Use sharing rules to expand access beyond OWD for particular groups or criteria.

Types of sharing rules:

- Owner-based sharing: share records owned by users in a role/public group.
- Criteria-based sharing: share records matching certain field criteria.

Steps:

1. Setup → Quick Find → Sharing Settings → scroll to the object → New under Sharing Rules.

2. Choose rule type, criteria or owner, and specify target users (Roles, Roles and Subordinates, Public Groups).

Screenshot to attach:

- [Insert screenshot: 10_SharingRule_Create.png]
- [Insert screenshot: 10_SharingRule_List.png]

The screenshot shows the 'Sharing Settings' page under the 'SETUP' tab. It includes sections for 'Other Settings' (with checkboxes for Manager Groups, Secure guest user record access, and Require permission to view record names in lookup fields), 'Sharing Rules' (Task Sharing Rules table showing 'Owner in Role: VP' shared with 'Role: Executive' at 'Read/Write' level), and 'Sharing Overrides' (Profiles That Override Task Sharing section).

Action	Criteria	Shared With	Access Level
Edit Del	Owner in Role: VP	Role: Executive	Read/Write

11. Login Access Policies & Security

Items to configure:

- Session Settings (Timeouts): Setup → Quick Find → Session Settings.
- Login IP Ranges for Profiles: in Profile settings → Login IP Ranges.
- Login Hours for Profiles: in Profile settings → Login Hours.
- Connected App policies (for integrations).
- Two-Factor Authentication (2FA) enforcement for admins (recommended in production).

Screenshot to attach:

- [Insert screenshot: 11_SessionSettings.png]
- [Insert screenshot: 11_Profile_LoginIP.png]



SETUP

Session Settings

Session Settings

[Help for this Page](#)

Set the session security and session expiration timeout for your organization.

Session Timeout

Timeout Value

- Disable session timeout warning popup
- Force logout on session timeout

Session Settings

- Lock sessions to the IP address from which they originated
- Lock sessions to the domain in which they were first used
- Terminate all of a user's sessions when an admin resets that user's password [i](#)
- Force relogin after Login-As-User
- Require HttpOnly attribute
- Use POST requests for cross-domain sessions
- Enforce login IP ranges on every request [i](#)
- When embedding a Lightning application in a third-party site, use a session token instead of a session cookie.

Extended use of IE11 with Lightning Experience

EXTENDED USE OF IE11 WITH LIGHTNING EXPERIENCE HAS NOW ENDED



SETUP

Profiles

Require a minimum 1 day password lifetime	<input type="checkbox"/>
Don't immediately expire links in forgot password emails	<input type="checkbox"/>

[Edit](#) [Clone](#) [Delete](#) [View Users](#)

Login Hours		Edit	Delete	Login Hours Help ?
Day		Start Time	End Time	
Sunday		7:30 PM PDT	5:30 AM PDT	
Monday		7:30 PM PDT	5:30 AM PDT	
Tuesday		7:30 PM PDT	5:30 AM PDT	
Wednesday		7:30 PM PDT	5:30 AM PDT	
Thursday		7:30 PM PDT	5:30 AM PDT	
Friday		7:30 PM PDT	5:30 AM PDT	
Saturday		7:30 PM PDT	5:30 AM PDT	

Login IP Ranges		New	Login IP Ranges Help ?
Action	IP Start Address	IP End Address	Description
Edit Del	157.33.253.0	157.33.253.1	

Enabled Apex Class Access		Edit	Enabled Apex Class Access Help ?
---------------------------	--	----------------------	--

The screenshot shows the Salesforce Setup interface under the 'Profiles' tab. It displays various permission settings:

- Data Share Targets:** Data Share Targets (✓), Data Share Target Connection (✓).
- Work Types:** Work Types (✓), Work Type Groups (✓).
- Custom Object Permissions:**

Object	Basic Access				Data Administration			Basic Access				Data Administration		
	Read	Create	Edit	Delete	View All Records	Modify All Records	View All Fields	Read	Create	Edit	Delete	View All Records	Modify All Records	View All Fields
Cases	✓				✓			✓				✓		
Custom Summarizers	✓				✓			✓				✓		
EmailMessages	✓							✓				✓		
Tasks	✓							✓				✓		
- Session Settings:** Session Times Out After: 2 hours of inactivity; Session Security Level Required at Login.

12. Dev Org Setup (Developer Edition)

Why use a Developer Org:

- Developer Edition is free and provides most features for development and testing without impacting production.

Steps:

1. Go to <https://developer.salesforce.com> and sign up for a Developer Edition if you don't already have one.
2. Note the username and password in a secure place.

Smart Task & Email Summarizer For Executives — Phase 3: Data Modeling & Relationships (Hands-On Guide)

Goal: Build the Salesforce data structure for a Smart Task & Email Summarizer for Executives. This guide walks you through objects, fields, relationships, record types, layouts, compact layouts, schema builder usage, choice of relationship types, automation hints (Flows), validation rules, test cases, and where/how to attach screenshots to the DOCX.

Quick Checklist (at-a-glance)

1. Create custom objects: Email__c, Summary__c, Action_Item__c (Task-like), Feedback__c (optional).
2. Create fields (exact types & API names listed below).
3. Create Lookup relationships between objects and to Contact/User.
4. Create Record Types for Summary: Auto_Summary vs Manual_Edit; and for Action_Item: Action_Item vs Follow_Up.
5. Create Page Layouts & Lightning record pages; add related lists and quick actions.
6. Create Compact Layouts for mobile.
7. Use Schema Builder to visualize relationships.
8. Add validation rules, formula fields, and sample Flows to generate Action Items from summaries.
9. Run the testing plan and capture screenshots into the screenshots/ folder; insert them into this doc.

Prerequisites

- Salesforce Developer Edition or Sandbox with Lightning Experience.
- Admin rights to create objects, fields, record types, layouts and Flows.
- API access for integrations (e.g., Gmail/Microsoft Graph) if ingesting emails automatically.
- (Optional) OpenAI/GPT integration credentials and a middleware (Heroku/Apex/Platform Events) for summarization.

1) Objects & API names (recommended)

Standard object: Contact (executives, senders, recipients) and User (internal users).

Custom objects to create:

A) Email

- Label: Email
- API Name: Email__c
- Record Name: Auto Number (EMAIL-{0000}) or Text (Subject snippet)

B) Summary

- Label: Summary

- API Name: Summary__c
- Record Name: Auto Number (SUM-{0000})

C) Action Item (for generated tasks / follow-ups)

- Label: Action Item
- API Name: Action_Item__c
- Record Name: Auto Number (AI-{0000})

D) Feedback (optional — capture user feedback on summary quality)

- Label: Feedback
- API Name: Feedback__c
- Record Name: Auto Number

The screenshot shows two related pages from the Salesforce setup interface.

Object Manager Page:

- Header: Setup > Object Manager
- Search bar: ema
- Create button: Create ▾
- Table of objects:

LABEL	API NAME	TYPE	DESCRIPTION	LAST MODIFIED	DEPLOYED
Contact Point Email	ContactPointEmail	Standard Object			
Email Message	EmailMessage	Standard Object			
EmailMessage	EmailMessage__c	Custom Object		9/25/2025	✓
List Email	ListEmail	Standard Object			

Summary Object Details Page:

- Header: SETUP > OBJECT MANAGER
- Section: Summary
- Left sidebar (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
- Right panel (Details):

Description	
API Name	Summary__c
Custom	✓
Singular Label	Summary
Plural Label	Summaries
Enable Reports	✓
Track Activities	
Track Field History	✓
Deployment Status	Deployed
Help Settings	
Standard salesforce.com Help Window	
- Buttons: Edit | Delete

Action Items

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	<p>Details</p> <p>Description</p> <hr/> <p>API Name <u>Action_Item__c</u></p> <p>Custom ✓</p> <p>Singular Label Action Items</p> <p>Plural Label Action Items</p> <hr/> <p>Enable Reports ✓</p> <p>Track Activities ✓</p> <p>Track Field History</p> <hr/> <p>Deployment Status Deployed</p> <p>Help Settings</p> <p>Standard salesforce.com Help Window</p>
<input type="button" value="Edit"/> <input type="button" value="Delete"/>	

2) Fields (types, lengths & API names)

Email object fields (Email__c):

- Subject — Subject__c — Text (255)
- Body — Body__c — Long Text Area (32768)
- Sender Email — Sender_Email__c — Email
- Sender Contact — Sender_Contact__c — Lookup(Contact)
- Received Date — Received_Date__c — DateTime
- Thread Id — Thread_Id__c — Text (100) — for deduping/conversation grouping
- Attachment Links — Attachment_Links__c — Long Text Area
- Processing Status — Processing_Status__c — Picklist — Pending, Processing, Summarized, Error
- Source System — Source_System__c — Picklist — Gmail, Outlook, Manual Upload, API
- External Id — External_Id__c — Text (100) — set External ID if syncing from outside

Summary object fields (Summary__c):

- Short Summary — Short_Summary__c — Text (500) — short headline summary
- Full Summary — Full_Summary__c — Long Text Area (32768) — full AI-generated summary
- Key Action Items — Key_Action_Items__c — Long Text Area — list each action item on new line
- Confidence Score — Confidence_Score__c — Percent (3,2) — confidence from AI or heuristic
- Generated By — Generated_By__c — Picklist — AI, Human, Hybrid
- Source Email — Source_Email__c — Lookup(Email__c)
- Related Contact — Related_Contact__c — Lookup(Contact) — primary recipient/executive
- Status — Summary_Status__c — Picklist — Draft, Ready, Sent, Reviewed
- Created By System At — Generated_At__c — DateTime

Action Item object fields (Action_Item__c):

- Title — Title__c — Text (255)
- Description — Description__c — Long Text Area
- Due Date — Due_Date__c — Date
- Assigned To — Assigned_To__c — Lookup(User)
- Status — AI_Status__c — Picklist — Open, In Progress, Done, Deferred
- Priority — Priority__c — Picklist — Low, Medium, High
- Source Summary — Source_Summary__c — Lookup(Summary__c)
- Related Email — Related_Email__c — Lookup(Email__c) — optional

Feedback object fields (Feedback__c) — optional:

- Rating — Rating__c — Number (1,0)
- Comments — Comments__c — Long Text Area
- Related Summary — Related_Summary__c — Lookup(Summary__c)
- Submitted By — Submitted_By__c — Lookup(User or Contact)

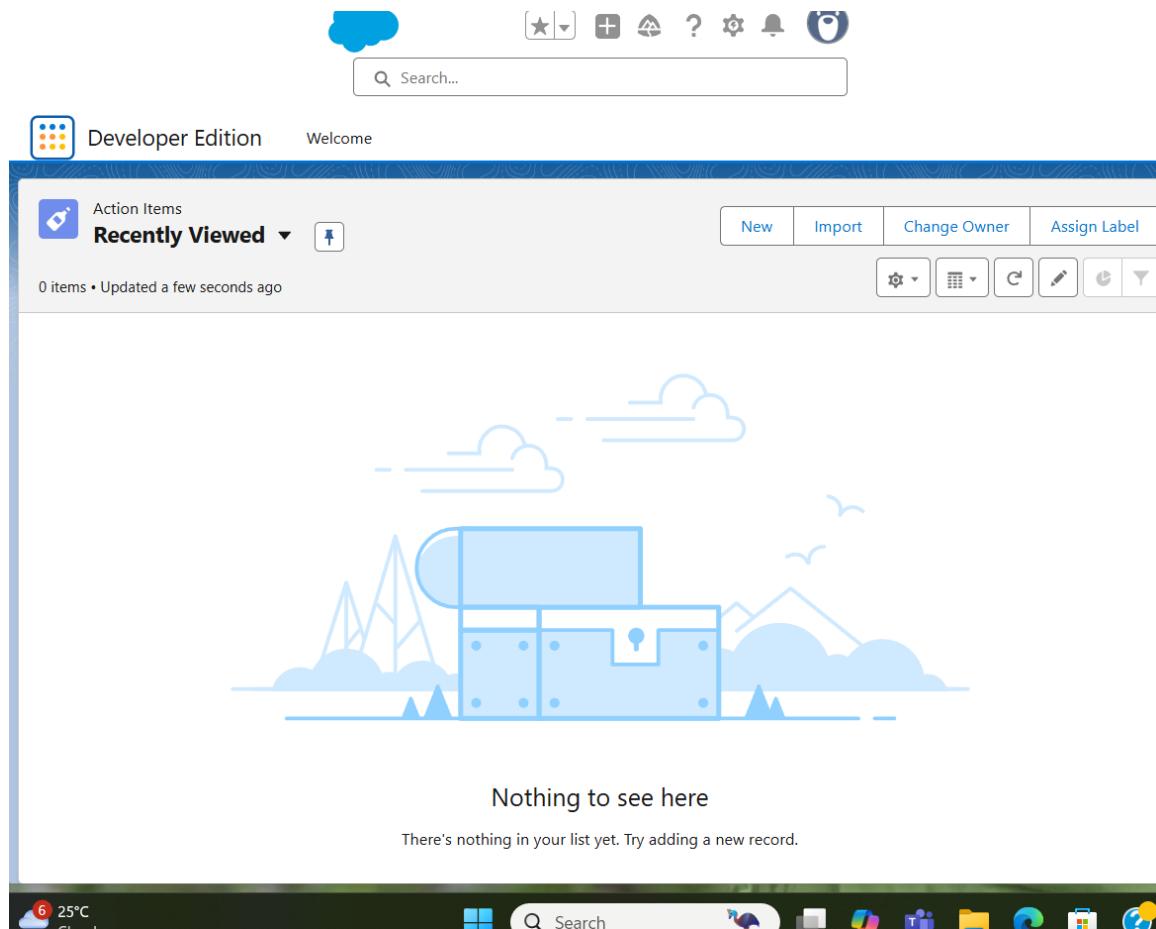
3) Relationships & Data Model (recommended)

Recommended relationships (Lookup unless noted):

- Summary__c → Email__c (Lookup) — each summary points to the source email.
- Summary__c → Contact (Lookup) — primary executive/recipient the summary is for.
- Action_Item__c → Summary__c (Lookup) — tasks generated from a summary.
- Action_Item__c → User (Assigned_To__c) (Lookup to User).
- Email__c → Contact (Sender_Contact__c) (Lookup) — optional if sender is in Contacts.

If you want cascading deletes or roll-up summary fields (e.g., number of action items per summary), consider Master-Detail between Action_Item__c and Summary__c. Otherwise keep Lookup to preserve loose coupling.

Visual tip: Use Schema Builder (Setup → Schema Builder) to drag these objects and create the Lookup links visually.



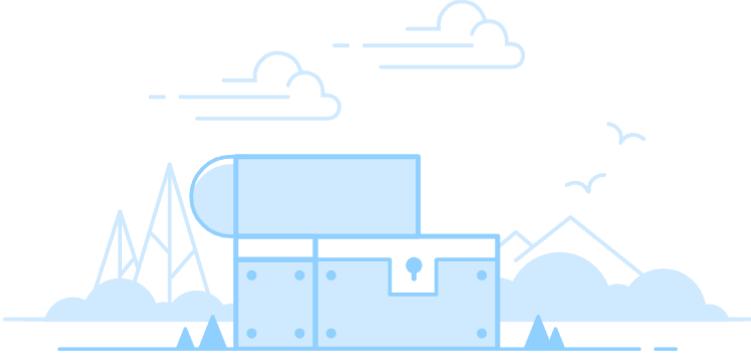
Developer Edition Welcome

EmailMessages
Recently Viewed ▾

New Import Change Owner Assign Label

Search this list...

0 items • Updated a few seconds ago



A light blue illustration of a modern building with a glass facade and a flat roof. It is surrounded by stylized white clouds and small blue birds flying in the sky.

Developer Edition Welcome

Summaries
Recently Viewed ▾

New Import Change Owner Assign Label

Search...

0 items • Updated a few seconds ago



A light blue illustration of a modern building with a glass facade and a flat roof. It is surrounded by stylized white clouds and small blue birds flying in the sky.

Nothing to see here

4) Useful Formula Fields & Example

A) Action_Item__c: Days Until Due — formula (Number):
IF(ISBLANK(Due_Date__c), NULL, Due_Date__c - TODAY())

B) Summary__c: Has_Action_Items (Checkbox formula):

NOT(ISBLANK(Key_Action_Items__c))

C) Summary__c: Short_Summary__c auto-generated via automation — not a formula if relying on external AI; use Apex/Flow to populate.

The screenshot shows the Salesforce Object Manager interface. The top navigation bar includes 'Setup', 'Home', and 'Object Manager'. Below the navigation is a breadcrumb trail 'SETUP > OBJECT MANAGER' and the page title 'Action Items'. On the left, a sidebar lists various setup categories: Details, Fields & Relationships (which is selected), Page Layouts, Lightning Record Pages, Buttons, Links, and Actions, Compact Layouts, Field Sets, Object Limits, Record Types, Related Lookup Filters, Restriction Rules, and Scoping Rules. The main content area displays the 'Fields & Relationships' section for the Action Items object. It features a table with columns: FIELD LABEL, FIELD NAME, DATA TYPE, CONTROLLING FIE..., and INDEXED. The table contains the following data:

FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIE...	INDEXED
Action Items Name	Name	Text(80)		✓
Created By	CreatedById	Lookup(User)		
Days Until Due	Days_Until_Due__c	Formula (Number)		
Due Date	Due_Date__c	Date		
Last Modified By	LastModifiedById	Lookup(User)		
Owner	OwnerId	Lookup(User,Group)		✓

SETUP > OBJECT MANAGER

Summary

Details	Quick Find	New	Deleted Fields	Field Dependencies	Set History
Fields & Relationships					
Page Layouts					
Lightning Record Pages					
Buttons, Links, and Actions					
Compact Layouts					
Field Sets					
Object Limits					✓
Record Types					✓
Related Lookup Filters					
Restriction Rules					

5) Validation Rules (samples)

A) Prevent setting Summary Status to Ready if Full Summary is blank:

Formula:

AND(ISPICKVAL(Summary_Status__c, "Ready"), ISBLANK(Full_Summary__c))

Error: "Full Summary must be present before marking Ready."

B) Action Item Due Date cannot be in the past when creating:

Formula:

AND(

NOT(ISBLANK(Due_Date__c)),

Due_Date__c < TODAY()

)

Error: "Due Date cannot be in the past."

The screenshot shows the Salesforce Object Manager interface. At the top, there's a blue header bar with the word 'SETUP' and a gear icon. Below it is a white header with the title 'Object Manager'. The main content area has a light gray background. At the top left of this area, the title 'Action Items Validation Rule' is displayed. To its right is a small blue link labeled 'Help for'. Below the title, there's a blue link 'Back to Action Items'. The main body of the page is titled 'Validation Rule Detail'. It contains several sections: 'Rule Name' (DueDate_Not_Past), 'Active' (with a checked checkbox), 'Error Condition Formula' (AND(NOT(ISBLANK(Due_Date__c)), Due_Date__c < TODAY())), 'Error Message' (Due Date cannot be in the past), 'Description' (empty), 'Created By' (Anjali Deshmukh, 9/26/2025, 2:33 AM), 'Modified By' (Anjali Deshmukh, 9/26/2025, 2:33 AM), 'Error Location' (empty), and 'Due Date' (empty). At the bottom of the detail section are two buttons: 'Edit' and 'Clone'.

6) Record Types & Page Layouts

Record Types (recommended):

- Summary__c: Auto_Summary (default) and Manual_Edit — use Manual_Edit for human-edited summaries.
- Action_Item__c: Action_Item and Follow_Up (differences could change required fields or page layout).

Page Layouts & Lightning Pages:

- Email Layout: Subject, Sender, Received Date, Processing Status, Body (collapsed), Related Summaries (related list).
- Summary Layout: Short Summary, Full Summary, Key Action Items, Confidence Score, Related Email (lookup), Related Action Items.
- Action Item Layout: Title, Source Summary, Description, Due Date, Assigned To, Status, Priority.

Add Quick Actions:

- From Email record: 'Create Summary' (pre-fills Email lookup), 'Create Manual Summary'.
- From Summary record: 'Create Action Items' (Flow button to parse Key Action Items and create Action_Item__c records).

Summary

- 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

Record Type
Auto_Summary

[Help for this Page](#)
[« Back to Custom Object: Summary](#)

Use the Edit button to change the properties of this record type. Use the Edit links in the Picklist Values related list to choose the picklist values available for records with this record type.

[Edit](#)

Record Type Label	Auto_Summary	Active	<input checked="" type="checkbox"/>
Record Type Name	Auto_Summary		
Namespace Prefix			
Description			
Created By	Anjali Deshmukh, 9/26/2025, 2:37 AM	Modified By	Anjali Deshmukh, 9/26/2025, 2:37 AM

Picklists Available for Editing

[Picklists Available for Editing Help](#)

Action	Field	Modified Date
Edit	Summary Status	9/26/2025, 2:37 AM

Action Items

- 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

Record Type
Action_Item

[Help for this Page](#)
[« Back to Custom Object: Action Items](#)

Use the Edit button to change the properties of this record type. Use the Edit links in the Picklist Values related list to choose the picklist values available for records with this record type.

[Edit](#)

Record Type Label	Action_Item	Active	<input checked="" type="checkbox"/>
Record Type Name	Action_Item		
Namespace Prefix			
Description			
Created By	Anjali Deshmukh, 9/26/2025, 2:38 AM	Modified By	Anjali Deshmukh, 9/26/2025, 2:38 AM

7) Compact Layouts (mobile)

Compact layouts to show on mobile where space is limited:

- Email compact: Subject, Sender_Email__c, Processing_Status__c
- Summary compact: Short_Summary__c, Confidence_Score__c, Summary_Status__c

Assign these under the object's compact layout settings and on the lightning record page.

SETUP > OBJECT MANAGER

EmailMessage

Details

Fields & Relationships

Page Layouts

Lightning Record Pages

Buttons, Links, and Actions

Compact Layouts

Field Sets

Object Limits

Record Types

Related Lookup Filters

Search Layouts

EmailMessage Compact Layout
EmailMessage

« Back to EmailMessage

Compact Layout Detail

Label	EmailMessage	Object Name	EmailMessage
API Name	EmailMessage		
Included Fields	Sender Subject Processing Status		
Created By	Anjali Deshmukh, 9/26/2025, 3:00 AM	Modified By	Anjali Deshmukh, 9/26/2025, 3:00 AM

Edit Clone Delete Compact Layout Assignment

Setup | Home | Object Manager

SETUP > OBJECT MANAGER

Summary

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

Summary Compact Layout
Summary

Help for this Page ?

< Back to Summary

Compact Layout Detail		Edit	Clone	Delete	Compact Layout Assignment
Label	Summary				Object Name Summary
API Name	Summary				
Included Fields	Summary Status Summary Name				
Created By	Anjali Deshmukh, 9/26/2025, 3:01 AM				Modified By Anjali Deshmukh, 9/26/2025, 3:01 AM

[Edit](#) [Clone](#) [Delete](#) [Compact Layout Assignment](#)

Setup | Home | Object Manager

SETUP > OBJECT MANAGER

Action Items

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

Action Items Compact Layout
Action Items

Help for this Page ?

< Back to Action Items

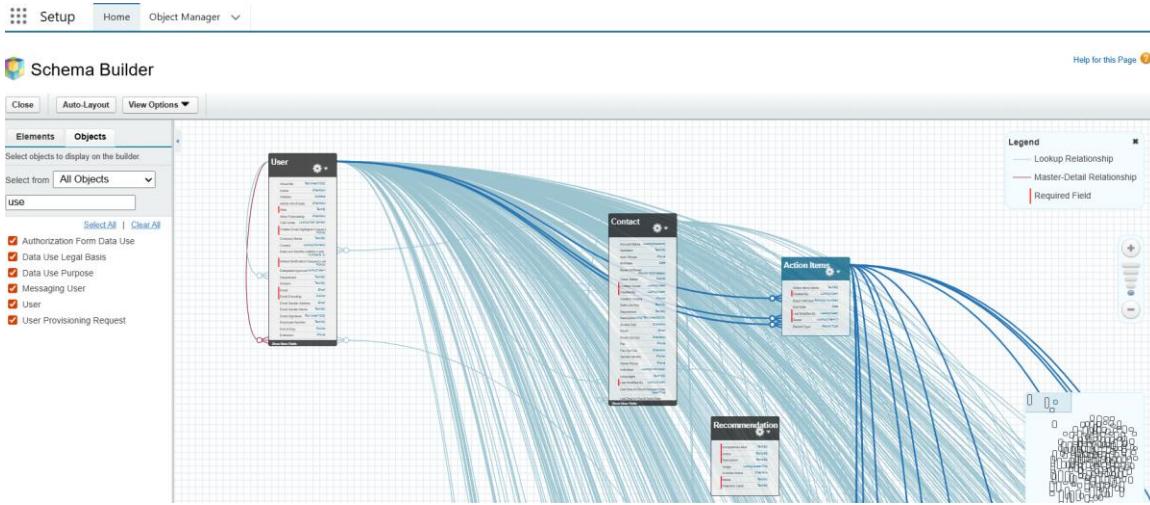
Action Items Compact Layout Detail

Compact Layout Detail		Edit	Clone	Delete	Compact Layout Assignment
Label	Action Items				Object Name Action Items
API Name	Action_Items				
Included Fields	Due Date Days Until Due				
Created By	Anjali Deshmukh, 9/26/2025, 3:02 AM				Modified By Anjali Deshmukh, 9/26/2025, 3:02 AM

[Edit](#) [Clone](#) [Delete](#) [Compact Layout Assignment](#)

8) Schema Builder — steps

1. Setup → Schema Builder. Use the 'Objects' panel to show Email, Summary, Action Item, Contact, User.
2. Drag objects to the canvas and create Lookup relationships visually, or create fields first in Object Manager and then refresh.
3. Use Filters to reduce clutter and take a screenshot once complete.



9) Lookup vs Master-Detail vs Hierarchical (guidance)

- Use Lookup for loose coupling: Email ↔ Summary, Summary ↔ Contact, Action Item ↔ Summary (unless you need roll-ups).
- Use Master-Detail when you need roll-up summary fields, strict ownership, and cascade-delete (e.g., if Action Items must be deleted with the Summary).
- Hierarchical relationships are only for User-to-User relationships and are not needed here.
- Recommendation: Start with Lookup; move to Master-Detail only if roll-ups are required and data-model adjustments are acceptable.

Developer Edition Welcome

EmailMessage
anajlidesjmukf@gmail.com

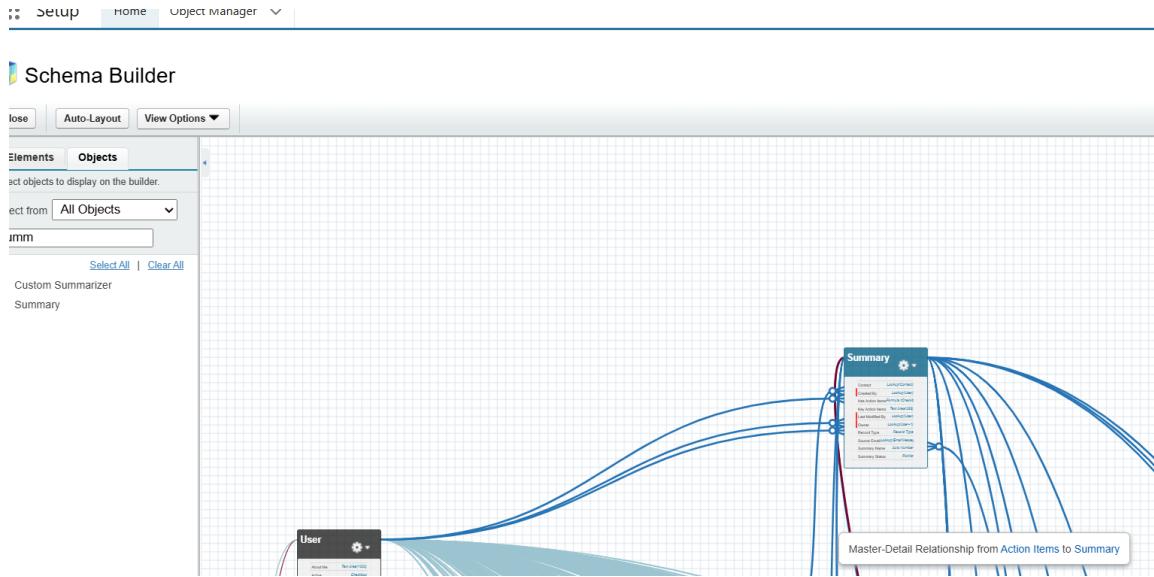
New Contact Edit New Opportunity ▾

Related	Details
EmailMessage Name anajlidesjmukf@gmail.com	Owner  Anjali Deshmukh 
Source_Email	
Created By  Anjali Deshmukh, 9/26/2025, 4:38 AM	Last Modified By  Anjali Deshmukh, 9/26/2025, 4:38 AM

SETUP > OBJECT MANAGER

Action Items

Details	Quick Find	New	Deleted Fields	Field Dependencies	Set History
Fields & Relationships					
Page Layouts	Action Items Name	Name	Text(80)		✓
Lightning Record Pages	Created By	CreatedBy	Lookup(User)		
Buttons, Links, and Actions	Days Until Due	Days_Until_Due_c	Formula (Number)		
Compact Layouts	Due Date	Due_Date_c	Date		
Field Sets	Last Modified By	LastModifiedBy	Lookup(User)		
Object Limits	Record Type	RecordTypeId	Record Type	✓	
Record Types	Source Summary	Source_Summary_c	Master-Detail(Summary)		✓
Related Lookup Filters					
Restriction Rules					



10) Junction Objects & External Objects

- Junction: If one summary needs to be related to many contacts and you need a normalized many-to-many, create Summary_Contact__c as a junction.
- External Objects: If emails are stored in an external system and you don't want to sync them into Salesforce, use External Data Sources / External Objects. Alternatively, store a lightweight Email__c record with an External_Id and a link to the original system for full access.

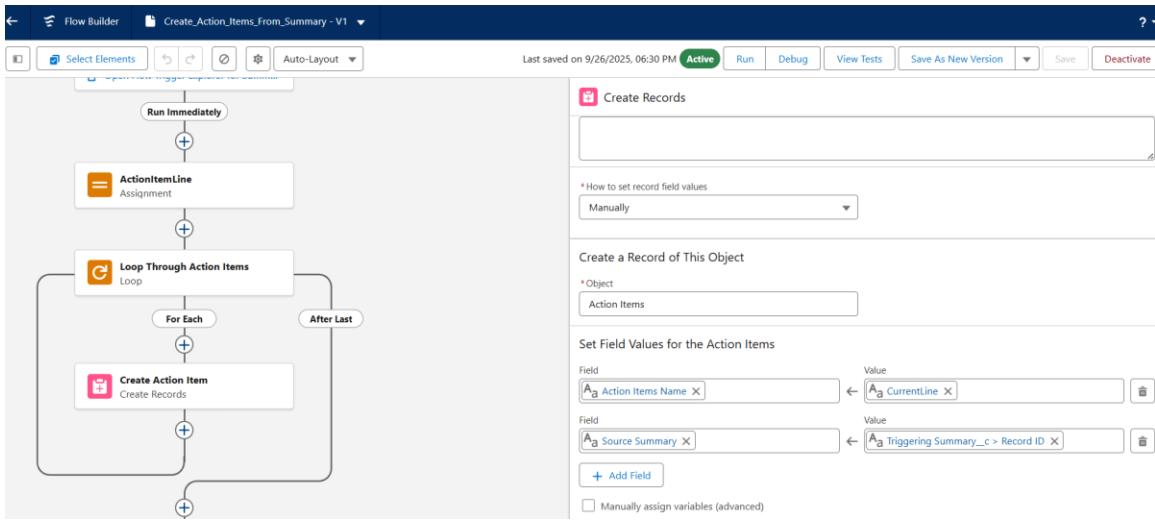
Integration tip: Consider using Platform Events or Middleware to stream incoming emails into Salesforce quickly.

11) Automation blueprint (Flow) — create Action Items from Key Action Items

Use a Record-Triggered Flow on Summary__c when Summary_Status__c changes to 'Ready':

1. Trigger: When Summary__c is updated and Summary_Status__c equals 'Ready'.
2. Get element: Parse Key_Action_Items__c (split by newline) — use Apex or a Collection loop in Flow to split lines.
3. For Each action text: Create Action_Item__c record with Title (first 60 chars), Description (full text), Source_Summary__c = current Summary Id.
4. Optionally assign to default user or use a mapping logic to assign based on keywords.
5. Add error handling path to log failures in a custom Platform Event or Feedback__c.

If you need advanced NLP splitting, perform it outside Salesforce and send ready-to-create Action Items via API.



12) Testing Plan & Sample Records

Create sample records to verify behavior. Example test data:

Emails:

- Email 1: Subject='Weekly Exec Briefing: Q3 results', Sender='ceo@example.com', Received Date=2025-09-24, Processing Status='Pending'
- Email 2: Subject='Client meeting follow-up', Sender='pm@example.com', Received Date=2025-09-25, Processing Status='Pending'

Summaries:

- Summary A: Short='Q3 highlights', Full='Sales up 12%... Action items: 1) Review pricing. 2)

Schedule meeting with ops.' Confidence 0.82, Generated_By='AI'

Action Items:

- AI-1: Title='Review pricing', Due Date='2025-10-01', Assigned To=executive assistant,

Status='Open' (created by Flow from Summary A)

Developer Edition Welcome

The screenshot shows a Salesforce interface for managing EmailMessages. At the top, there's a navigation bar with icons for Home, App Launcher, and Help. Below it is a header with the title "EmailMessages" and a dropdown menu "Recently Viewed". On the right side of the header are buttons for "New", "Import", "Change Owner", and "Assign Label". Below the header is a search bar with placeholder text "Search this list..." and several filter and sorting icons.

The main area displays a list titled "Recently Viewed" with 2 items updated a few seconds ago. The list includes columns for a checkbox, the EmailMessage Name, and a dropdown menu. The items listed are:

	EmailMessage Name	
1	siddhideshmukh@gmail.com	▼
2	anajlidesjmukf@gmail.com	▼

Phase 4 — Process Automation (Admin)

Project: Smart Email & Task Summarizer for Executives

Overview (keep it minimal)

This document lists only the necessary, step-by-step actions to implement Phase 4 (Process Automation) for the Booking / Rental Booking object in Salesforce. Follow the steps in your Sandbox/Developer org, capture screenshots at the indicated steps, paste them into the screenshot placeholders, and then run the tests.

Quick Checklist (must-do items)

- Create Validation Rule: End Date cannot be before Start Date (covers empty-checks).
- Create Record-Triggered Flow (Before-Save) to calculate Total Amount.
- Create Approval Process (or Flow-based approval) for bookings > ₹50,000.
- Add Final Approval Actions: Field Update (Status = Confirmed), Email Alert to customer, Create Task for Agent, Send Custom Notification to Agent.
- Create Screen Flow for manual booking creation (optional fast UI).
- Create Email Template(s) and Email Alerts used by the Approval Process/Flow.
- Create Notification Type and use Flow action 'Send Custom Notification'.
- Test end-to-end in Sandbox, capture screenshots for every completed step, embed them in this docx.

Detailed Steps (necessary)

1) Validation Rule — End Date must be after Start Date

Where: Setup → Object Manager → Rental Booking (or Booking) → Validation Rules → New

Create a new validation rule with:

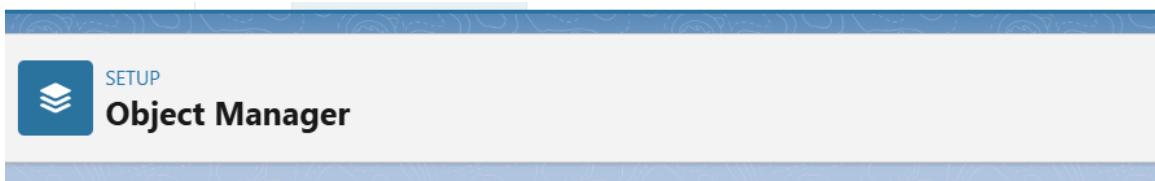
1. Rule Name: End_Date_After_Start_Date (or similar)
2. Formula (use this exact formula):

```
AND (
    NOT (ISBLANK(Start_Date__c)) ,
    NOT (ISBLANK(End_Date__c)) ,
    End_Date__c < Start_Date__c
)
```

Error Message: "End Date cannot be before Start Date."

Error Location: End Date (or Top of Page)

Screenshot: Take a screenshot after saving the validation rule — filename: 01_ValidationRule.png. Paste it below the "Validation Rule" heading in this docx and upload the same file to the Booking record Files related list.



2) Record-Triggered Flow — Calculate Total Amount (Before-Save)

Where: Setup → Flow → New Flow → Record-Triggered Flow

Trigger: Object = Rental Booking; Trigger the Flow when: A record is created or updated; Run the Flow: Before Save (fast field update)

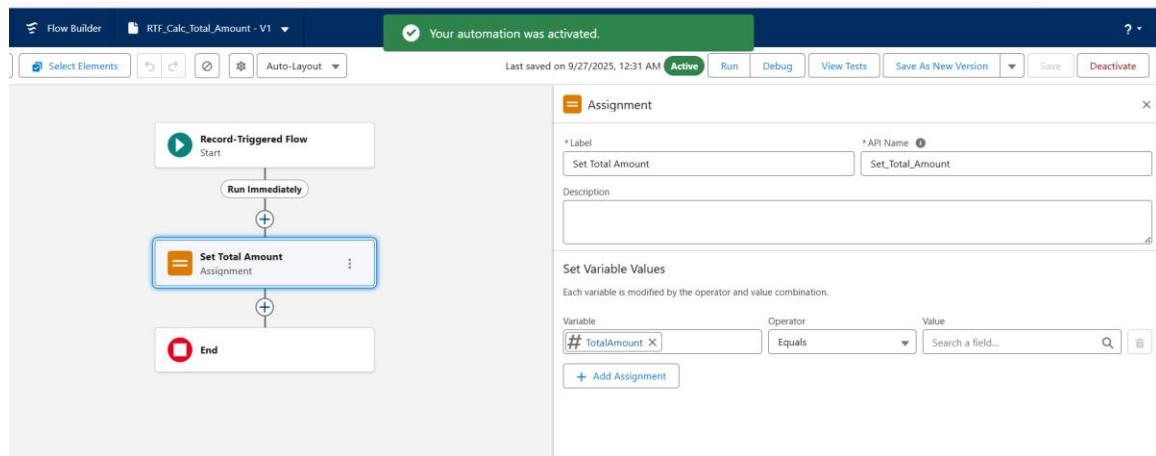
Purpose: Compute and set Total_Amount__c = (number-of-days) × Daily_Rate__c. Use a Formula Resource or Assignment.

3. Steps (quick):
4. Start: Select object 'Rental Booking'. Choose 'A record is created or updated'. Choose 'Before Save'.
5. Create a Formula Resource (Number) named Days_Booked: $\{!\$Record.End_Date_c\} - \{!\$Record.Start_Date_c\} + 1$
6. Create a Formula Resource (Currency/Number) named Calculated_Total: Days_Booked * $\{!\$Record.Daily_Rate_c\}$
7. Assignment: Set $\$Record.Total_Amount_c = Calculated_Total$ (or directly assign expression to the field in Before Save update).
8. Save → Activate the Flow.

Notes: Use checks to avoid negative or blank dates. Add a Decision element only if you want to skip calculation when Start or End is blank.

Test: Create a booking with Start = 2025-10-01, End = 2025-10-03, Daily Rate = 1500 → Total = 3 × 1500 = 4500.

Screenshot: 02_Flow_BeforeSave_Calc.png — capture the Flow canvas showing the Formula Resource and Assignment. Paste it in this docx.



3) Approval Process — Bookings > ₹50,000

Where: Setup → Process Automation → Approval Processes → Create New Approval Process → For Rental Booking object

Entry Criteria: Total_Amount__c > 50000 (or formula)

Steps:

9. Create the approval process using "Use Standard Setup Wizard" (faster).
10. Step 1: Initial Submitters: Record Owner and users higher in role hierarchy (adjust as needed).
11. Step 2: Approval Steps: Send to manager (use Manager field or a lookup to Approver).
12. Final Approval Actions: Field Update (Booking_Status__c = "Confirmed"), Email Alert to Customer, Create Task for Agent, Send Custom Notification to Agent.
13. Final Rejection Actions: Field Update (Booking_Status__c = "Rejected") and Email Alert to customer (optional).

Tip: If you prefer Flow-based approvals (more flexible), you can model approval steps with Flows. Recent releases improved Flow Approvals.

Screenshot: 03_Approval_Process.png — capture the approval process summary page after creation. Paste it here and upload that image to the Approval Process documentation folder and Booking record Files.

The screenshot shows the 'Approval Processes' page in Salesforce. The process is named 'Booking Approval - >50k'. It has the following details:

- Process Name:** Booking Approval - >50k
- Unique Name:** Booking_Approval_50k
- Description:** Approval for bookings where Total_Amount__c exceeds \$50,000
- Entry Criteria:** Rental Booking: Total Amount GREATER THAN 9/26/2025
- Record Editability:** Administrator ONLY
- Approval Assignment Email Template:** Sales_New_Customer_Email
- Initial Submitters:** Rental Booking Owner
- Created By:** Anjali Deshmukh, 9/26/2025, 12:15 PM
- Modified By:** Anjali Deshmukh, 9/26/2025, 12:17 PM
- Active:**
- Next Automated Approver Determined By:**
- Allow Submitters to Recall Approval Requests:**

Initial Submission Actions:

Action Type	Description
Record Lock	Lock the record from being edited

Approval Steps:

Action	Step Number	Name	Description	Criteria	Assigned Approver	Reject Behavior
Show Actions Edit Del	1	Manager Approval			Manually Chosen	Final Rejection

4) Email Templates and Email Alerts

Where: Setup → Email Templates → New Lightning Template (or Classic if your org uses classic templates)

Create two templates (examples):

14. Booking Approval Template (used for Final Approval) — merge fields: {!Booking.Name}, {!Booking.Start_Date__c}, {!Booking.End_Date__c}, {!Booking.Total_Amount__c}, {!Contact.Name}
15. Booking Rejection Template — short text explaining reason and next steps.

Create Email Alerts (Setup → Email Alerts) that reference the templates. Use Email Alerts in Approval Process final actions or Flow "Send Email" action.

Screenshot: 04_EmailTemplate.png

Email Template
Booking Approval Template

Edit Clone Delete ▾

Details

Related

Information

Email Template Name
Booking Approval Template

Description
Template to notify customer and agent when booking is approved

Made in Email Template Builder

Related Entity Type
Rental Booking

Folder
Public Email Templates

Message Content

Subject
Booking Approval Request for

HTML Value
Dear {Rental_Booking__c.Customer__r.Name},
Your booking has been approved! Here are the details:
Booking Name: {Rental_Booking__c.Name}

Enhanced Letterhead



SETUP

Email Alerts

Email Alert

Send email to customer when booking is approved



Help for this Page

[Rules Using This Email Alert \[0\]](#) | [Approval Processes Using This Email Alert \[0\]](#) |
[Entitlement Processes Using This Email Alert \[0\]](#)

Email Alert Detail

[Edit](#) [Delete](#) [Clone](#)

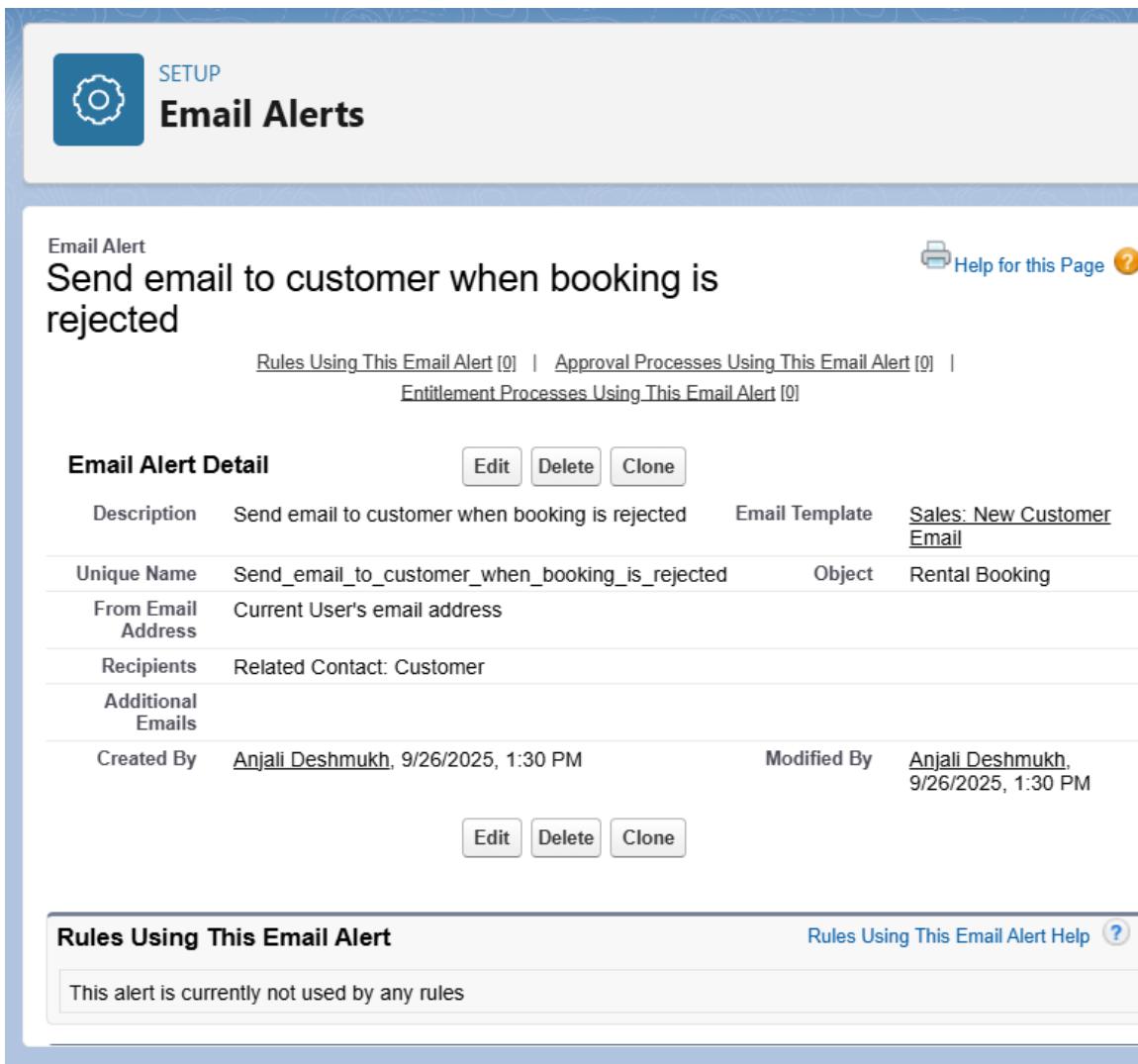
Description	Send email to customer when booking is approved	Email Template	Sales: New Customer Email
Unique Name	Send_email_to_customer_when_booking_is_approved	Object	Rental Booking
From Email Address	Current User's email address		
Recipients	Related Contact: Customer		
Additional Emails			
Created By	Anjali Deshmukh , 9/26/2025, 1:28 PM	Modified By	Anjali Deshmukh , 9/26/2025, 1:28 PM

[Edit](#) [Delete](#) [Clone](#)

Rules Using This Email Alert

[Rules Using This Email Alert Help](#)

This alert is currently not used by any rules



The screenshot shows the 'Email Alerts' setup page in Salesforce. At the top, there's a blue header bar with the word 'SETUP' and a gear icon. Below it, the main title 'Email Alerts' is displayed. On the left, a sidebar lists categories like 'Email Alert', 'Send email to customer when booking is rejected', 'Rules Using This Email Alert [0]', 'Approval Processes Using This Email Alert [0]', and 'Entitlement Processes Using This Email Alert [0]'. The main content area is titled 'Email Alert Detail' and contains a table with the following data:

Email Alert Detail		Edit	Delete	Clone
Description	Send email to customer when booking is rejected	Email Template	Sales: New Customer Email	
Unique Name	Send_email_to_customer_when_booking_is_rejected	Object	Rental Booking	
From Email Address	Current User's email address			
Recipients	Related Contact: Customer			
Additional Emails				
Created By	Anjali Deshmukh , 9/26/2025, 1:30 PM	Modified By	Anjali Deshmukh , 9/26/2025, 1:30 PM	
		Edit	Delete	Clone

Below this, there's a section titled 'Rules Using This Email Alert' with a note: 'This alert is currently not used by any rules'. There's also a link 'Rules Using This Email Alert Help'.

5) Field Update, Task Creation, and Custom Notification

Field Update (Final Approval Action): Set Booking_Status__c = "Confirmed".

Task Creation (Final Approval Action): Create a Task with the following sample fields:

Subject: Prepare car for booking {!Booking.Name} — use merge fields;

Assigned To: Agent (lookup on Booking record), Due Date: {!Booking.Start_Date__c} - 1 day,
Status: Not Started, Priority: High.

Custom Notification:

1. Setup → Platform Tools → Notifications → Notification Builder (or Notification Types) → New Notification Type → Name: Agent Booking Approved.

2. In Flow (After-Save or Final Approval Action), add Action: Send Custom Notification. Pick the Notification Type, Title, Body and set Recipient = Booking.Agent__c.

Screenshot: 05_Task_Notification.png

The screenshot shows the Salesforce Setup interface for 'Approval Processes'. The top navigation bar includes 'Record Lock' and 'Lock the record from being edited'. The main area displays four sections: 'Approval Steps', 'Final Approval Actions', 'Final Rejection Actions', and 'Recall Actions'. Each section contains a table with columns for Action, Step Number, Name, Description, Criteria, Assigned Approver, Reject Behavior, Type, and Description.

Action	Step Number	Name	Description	Criteria	Assigned Approver	Reject Behavior
Show Actions Edit Del	1	Manager Approval			Manually Chosen	Final Rejection

Action	Type	Description
Edit	Record Lock	Lock the record from being edited
Edit Remove	Field Update	Update Booking Status to Confirmed

Action	Type	Description
Edit	Record Lock	Unlock the record for editing

Action	Type	Description
	Record Lock	Unlock the record for editing



SETUP

Tasks

Task

repare car for booking {Booking.Name}

 Help for this Page [?](#)[Rules Using This Task \[0\]](#) | [Approval Processes Using This Task \[1\]](#) | [Entitlement Processes Using This Task \[0\]](#)

Workflow Task Detail

[Edit](#) [Delete](#) [Clone](#)

Object	Rental Booking	Status	Not Started
Assigned To	User : Anjali Deshmukh	Priority	High
Subject	repare car for booking {Booking.Name}		
Unique Name	repare_car_for_booking_Booking_Name		
Due Date	Rental Booking: Start Date		
Comments			
Created By	Anjali Deshmukh, 9/26/2025, 1:46 PM	Modified By	Anjali Deshmukh, 9/26/2025, 1:47 PM

[Edit](#) [Delete](#) [Clone](#)

Rules Using This Task

[Rules Using This Task Help](#)

This task is currently not used by any rules

Approval Processes Using

[Approval Processes Using This Task Help](#)



SETUP

Custom Notifications



When you create and use custom notifications, the title and body of the custom push notification may be saved to and processed by Google, Microsoft and/or Apple. Salesforce is not responsible for the privacy and security practices of third-party systems or applications like Google Cloud Messaging or Apple Push Notification Service.

Custom Notification Types

[New](#)

Send custom notifications using [Flows](#) or [Process Builder](#)

NOTIFICATION NAME	API NAME	NAMESPACE
Agent Booking Approved	Agent_Booking_Approved	
enablement_coaching_feedback_ready	enablement_coaching_feedback_ready	

Phase 5 — Apex Programming (Developer) Smart Task & Email Summarizer for Executives

Purpose

This document is a step-by-step, copy-ready developer guide for Phase 5 of the "Smart Task & Email Summarizer for Executives" project.

It contains architecture notes, Apex code samples (Trigger, Handler, Service, Batch, Queueable, Scheduled, @future), test classes, best practices and a deployment checklist.

Use the code samples as starting points — adapt field/API names exactly to your org (this guide assumes objects named Booking__c and Car__c and fields like Start_Date__c, End_Date__c, Car__c lookup, Booking_Status__c, Daily_Rate__c, Total_Amount__c).

Quick checklist (must-do, prioritized)

- Create / verify custom objects and fields: Car__c (Daily_Rate__c, Status__c), Booking__c (Car__c lookup, Start_Date__c, End_Date__c, Booking_Status__c, Total_Amount__c, Discount__c).
- Implement Trigger + Trigger Handler (bulkified) to prevent overlapping bookings.
- Create BookingService for reusable logic (date helpers, queries, email senders, callouts).
- Write Batch Apex to mark overdue rentals nightly.
- Create Queueable job to calculate discounts for bulk updates.
- Write Scheduled Apex to run the morning report and/or schedule the batch.
- Implement @future(callout=true) or better: use Queueable with callout for insurance API.
- Write robust Test Classes (use Test.startTest()/stopTest(), @testSetup, HttpCalloutMock for callout tests).
- Create Named Credential / Remote Site for external callouts and update Remote Site Settings or use Named Credentials.
- Deploy to QA sandbox and run all tests; confirm 75%+ coverage.

SETUP > OBJECT MANAGER

Rental Booking

Details	Fields & Relationships				
	FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIE...	IND
Page Layouts	Agent	Agent_c	Lookup(User)		✓
Lightning Record Pages	Booking Status	Booking_Status_c	Picklist		
Buttons, Links, and Actions	Car	Car_c	Lookup(Car)		✓
Compact Layouts	Created By	CreatedBy	Lookup(User)		
Field Sets	Customer	Customer_c	Lookup(Contact)		✓
Object Limits	Daily_Rate	Daily_Rate_c	Currency(2, 2)		
Record Types	Discount	Discount_c	Percent(16, 2)		
Related Lookup Filters					
Search Layouts					
List View Button Layout					

SETUP > OBJECT MANAGER

Car

Details	Fields & Relationships				
	FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIE...	IND
Page Layouts	Car Name	Name	Text(80)		✓
Lightning Record Pages	Created By	CreatedBy	Lookup(User)		
Buttons, Links, and Actions	Daily Rate	Daily_Rate_c	Currency(16, 2)		
Compact Layouts	End Date	End_Date_c	Date		
Field Sets	Last Modified By	LastModifiedBy	Lookup(User)		
Object Limits	Owner	OwnerId	Lookup(User,Group)		✓
Record Types	Start Date	Start_Date_c	Date		
Related Lookup Filters	Status	Status_c	Picklist		
Search Layouts					
List View Button Layout					

Architecture & important notes

- Use Booking__c and Car__c as canonical names in this guide. Change them if your org uses different API names.
- Bulkify everything: never run SOQL/DML inside loops. Use Maps/Sets/Lists.
- Avoid callouts inside triggers. Queue the work and call out asynchronously (@future or Queueable callout).
- Store configuration (manager emails, API endpoints) in Custom Metadata / Custom Settings or Named Credential for safe changes without code edits.
- Use descriptive Booking_Status__c picklist values such as 'Confirmed', 'Cancelled', 'Returned', 'Overdue'.
- Use seeAllData=false in tests and create test records programmatically.

Usage: In Apex, query the custom metadata:



SETUP

Custom Metadata Types

2.57 AM

2.57 AM

Standard Fields

Action	Field Label	Field Name	Data Type	Indexed
	<u>Created By</u>	CreatedBy	Lookup(User)	
Edit	<u>Custom Metadata Record Name</u>	DeveloperName	Text(40)	
Edit	<u>Label</u>	MasterLabel	Text(40)	
	<u>Last Modified By</u>	LastModifiedBy	Lookup(User)	
Edit	<u>Namespace Prefix</u>	NamespacePrefix	Text	
Edit	<u>Protected Component</u>	IsProtected	Checkbox	

Custom Fields

[New](#)

Action	Field Label	API Name	Data Type	Field Manageability	Index
Edit Del	<u>Cron_Expression__c</u>	Cron_Expression_c_c	Text(255)	Upgradable	
Edit Del	<u>Insurance_Endpoint__c</u>	Insurance_Endpoint_c_c	Text(255)	Upgradable	
Edit Del	<u>Manager_Email__c</u>	Manager_Email_c_c	Text(255)	Upgradable	

App Config Detail		Edit	Delete	Clone
Label	Default	<input type="checkbox"/> Protected Component		
App Config Name	Default	<input type="checkbox"/> Namespace Prefix		
Manager_Email__c	Anujdeshmukh18@gmail.com			
Insurance_Endpoint__c	https://api.insurance.example			
Cron_Expression__c	0 30 7 * * ?			
Created By	Anjali Deshmukh, 9/27/2025, 3:01 AM	Last Modified By	Anjali Deshmukh, 9/27/2025, 3:01 AM	
		Edit	Delete	Clone

1) Trigger + Trigger Handler — prevent overlapping bookings (bulkified)

Create a before insert, before update trigger that delegates to a handler class.

```
trigger BookingTrigger on Booking__c (before insert, before update) {
    if (Trigger.isBefore) {
        if (Trigger.isInsert || Trigger.isUpdate) {
            BookingTriggerHandler.preventOverlappingBookings(Trigger.new,
Trigger.isUpdate ? Trigger.oldMap : null);
        }
    }
}

public with sharing class BookingTriggerHandler {
    // Prevent overlapping bookings for same car. Bulk-safe.
    public static void preventOverlappingBookings(List<Booking__c> newBookings,
Map<Id, Booking__c> oldMap) {
```

```

// Collect involved car Ids (avoid nulls)
Set<Id> carIds = new Set<Id>();
for (Booking__c b : newBookings) {
    if (b.Car__c != null) carIds.add(b.Car__c);
}
if (carIds.isEmpty()) return;

// Query existing bookings for those cars (exclude Cancelled). Exclude
records that are being updated by checking oldMap keys.
List<Booking__c> existing = [
    SELECT Id, Start_Date__c, End_Date__c, Booking_Status__c, Car__c
    FROM Booking__c
    WHERE Car__c IN :carIds AND Booking_Status__c != 'Cancelled'
];

// Group existing bookings by car
Map<Id, List<Booking__c>> existingByCar = new Map<Id,
List<Booking__c>>();
for (Booking__c e : existing) {
    if (!existingByCar.containsKey(e.Car__c))
existingByCar.put(e.Car__c, new List<Booking__c>());
    existingByCar.get(e.Car__c).add(e);
}

// Validate each incoming booking for overlap
for (Booking__c nb : newBookings) {
    // Basic date validation
    if (nb.Start_Date__c == null || nb.End_Date__c == null) continue;
    if (nb.Start_Date__c > nb.End_Date__c) {
        nb.addError('Start Date cannot be after End Date.');
        continue;
    }
    List<Booking__c> listForCar = existingByCar.get(nb.Car__c);
    if (listForCar == null) continue;
    for (Booking__c ex : listForCar) {
        // If updating, ignore comparing to the same record
        if (oldMap != null && ex.Id == nb.Id) continue;
        if (datesOverlap(nb.Start_Date__c, nb.End_Date__c,
ex.Start_Date__c, ex.End_Date__c)) {
            nb.addError('This booking overlaps with existing booking
with Id: ' + ex.Id);
            break;
        }
    }
}

// Helper to determine interval overlap (Date objects)
public static Boolean datesOverlap(Date aStart, Date aEnd, Date bStart,
Date bEnd) {
    if (aStart == null || aEnd == null || bStart == null || bEnd == null)
return false;
    return !(aEnd < bStart || aStart > bEnd);
}

```

```

1 public with sharing class BookingTriggerHandler {
2     public static void preventOverlappingBookings(List<Booking__c> newList, Map<Id, Booking__c> oldList) {
3         if (newList == null || newList.isEmpty()) return;
4
5         // ✅ Correct types
6         Set<Id> carIds = new Set<Id>();
7         Set<Id> updatingIds = new Set<Id>();
8         Date minStart = null;
9         Date maxEnd = null;
10
11        for (Booking__c b : newList) {
12            if (b.Car__c != null) carIds.add(b.Car__c); // ✅ Car__c is Id
13            if (b.Id != null) updatingIds.add(b.Id); // ✅ Id is Id
14
15            if (b.Start_Date__c != null) {
16                minStart = (minStart == null) ? b.Start_Date__c :
17                                (b.Start_Date__c < minStart ? b.Start_Date__c : minStart);
18            }
19        }

```



```

1 trigger BookingTrigger on Booking__c (before insert, before update) {
2     if (Trigger.isBefore) {
3         BookingTriggerHandler.preventOverlappingBookings(Trigger.new, Trigger.isUpdate ? Trigger.old : null);
4     }
5 }

```

2) BookingService — reusable logic & queries

```

public with sharing class BookingService {
    // Example SOQL for available cars
    public static List<Car__c> getAvailableCars() {
        return [SELECT Id, Name, Status__c, Daily_Rate__c FROM Car__c WHERE
Status__c = 'Available'];
    }

    // Send morning email with today's rentals (simple example)
    public static void sendDailyRentalsEmail() {
        Date today = Date.today();
        List<Booking__c> todays = [SELECT Id, Name, Start_Date__c, End_Date__c,

```

```

Car__c, Booking_Status__c FROM Booking__c WHERE Start_Date__c = :today];
if (todays.isEmpty()) return;

String body = 'Today\'s Bookings:\n';
for (Booking__c b : todays) {
    body += b.Name + ' (' + String.valueOf(b.Car__c) + ')'
Start: ' + String.valueOf(b.Start_Date__c) + '\n';
}

Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();
mail.setToAddresses(new List<String>{ 'manager@example.com' });
Replace or look up managers dynamically
mail.setSubject('Today\'s Rentals (' + String.valueOf(today) + ')');
mail.setPlainTextBody(body);
Messaging.sendEmail(new List<Messaging.SingleEmailMessage>{ mail });
}

// Example @future call to external insurance API (async callout)
@future(callout=true)
public static void callInsuranceApi(Set<Id> bookingIds) {
    Http http = new Http();
    for (Id bId : bookingIds) {
        try {
            HttpRequest req = new HttpRequest();
            req.setEndpoint('https://example-insurance.com/api/quote');
            req.setMethod('POST');
            req.setHeader('Content-Type', 'application/json');
            req.setBody('{ "bookingId": "' + bId + '"}');
            HttpResponse res = http.send(req);
            // Optionally parse res.getBody()
        } catch (Exception e) {
            System.debug('Insurance call failed for ' + bId + ': ' +
e.getMessage());
        }
    }
}

```

```

1 public with sharing class BookingService {
2
3     // 1. Query available cars
4     public static List<Car__c> getAvailableCars() {
5         return [
6             SELECT Id, Name, Status__c, Daily_Rate__c
7             FROM Car__c
8             WHERE Status__c = 'Available'
9         ];
10    }
11
12    // 2. Send morning email with today's rentals
13    public static void sendDailyRentalsEmail() {
14        Date today = Date.today();
15
16        //  Correct type: Booking__c (not Summary__c)
17        List<Booking__c> todays = [
18            SELECT Id, Name, Start_Date__c, End_Date__c, Car__c, Booking_Status__c
19            FROM Booking__c
20            WHERE Start_Date__c = :today
21        ];
22
23        ...
24    }
25
26    ...
27
28 }

```

Logs Tests Checkpoints Query Editor View State Progress Problems

3) Batch Apex — nightly job to mark overdue rentals

```

global class MarkOverdueBatch implements Database.Batchable<sObject> {

    global Database.QueryLocator start(Database.BatchableContext BC) {
        // Find bookings ended before today and not yet marked Returned or
        Overdue
        return Database.getQueryLocator([
            SELECT Id, End_Date__c, Booking_Status__c
            FROM Booking__c
            WHERE End_Date__c < :Date.today() AND Booking_Status__c NOT IN
('Returned', 'Overdue')
        ]);
    }

    global void execute(Database.BatchableContext BC, List<Booking__c> scope) {
        List<Booking__c> updates = new List<Booking__c>();
        for (Booking__c b : scope) {
            b.Booking_Status__c = 'Overdue';
            updates.add(b);
        }
        if (!updates.isEmpty()) update updates;
    }

    global void finish(Database.BatchableContext BC) {
        // Optional: notify admins or chain Queueables
    }
}

```

```

1 global class MarkOverdueBatch implements Database.Batchable<sObject>, Database.Stateful {
2     global Database.QueryLocator start(Database.BatchableContext BC) {
3         return Database.getQueryLocator([
4             SELECT Id, End_Date__c, Booking_Status__c
5             FROM Booking__c
6             WHERE End_Date__c < :Date.today() AND Booking_Status__c NOT IN ('Returned', 'Overdue')
7         ]);
8     }
9
10    global void execute(Database.BatchableContext BC, List<Booking__c> scope) {
11        List<Booking__c> updates = new List<Booking__c>();
12        for (Booking__c b : scope) {
13            b.Booking_Status__c = 'Overdue';
14            updates.add(b);
15        }
16        if (!updates.isEmpty()) update updates;
17    }
18
19    global void finish(Database.BatchableContext BC) {
20        // optional: notify admins, enqueue another job, or log summary

```

4) Queueable — async discount calculation for bulk rentals

```

public class DiscountQueueable implements Queueable {
    private Set<Id> bookingIds;

    public DiscountQueueable(Set<Id> bookingIds) {
        this.bookingIds = bookingIds;
    }

    public void execute(QueueableContext qc) {
        List<Booking__c> bookings = [SELECT Id, Start_Date__c, End_Date__c,
Total_Amount__c, Car__r.Daily_Rate__c FROM Booking__c WHERE Id IN :bookingIds];
        List<Booking__c> updates = new List<Booking__c>();
        for (Booking__c b : bookings) {
            Integer days = b.Start_Date__c.daysBetween(b.End_Date__c) + 1;
            Decimal rate = (b.Car__r != null && b.Car__r.Daily_Rate__c != null)
? b.Car__r.Daily_Rate__c : 0;
            Decimal total = rate * days;
            Decimal discountPct = 0;
            if (days >= 30) discountPct = 0.20;
            else if (days >= 7) discountPct = 0.10;
            b.Total_Amount__c = total * (1 - discountPct);
            updates.add(b);
        }
        if (!updates.isEmpty()) update updates;
    }
}

```

```

1 public class DiscountQueueable implements Queueable {
2     private Set<Id> bookingIds;
3     public DiscountQueueable(Set<Id> bookingIds) {
4         this.bookingIds = (bookingIds == null) ? new Set<Id>() : bookingIds;
5     }
6     public void execute(QueueableContext qc) {
7         if (bookingIds.isEmpty()) return;
8         List<Booking__c> bookings = [SELECT Id, Start_Date__c, End_Date__c, Total_Amount__c, Discount__c, Car__c.Daily_Rate__c
9                                         FROM Booking__c WHERE Id IN :bookingIds];
10        List<Booking__c> updates = new List<Booking__c>();
11        for (Booking__c b : bookings) {
12            Integer days = (b.Start_Date__c != null && b.End_Date__c != null) ? (b.Start_Date__c.daysBetween(b.End_Date__c) + 1) : 0;
13            Decimal rate = (b.Car__c != null && b.Car__c.Daily_Rate__c != null) ? b.Car__c.Daily_Rate__c : 0;
14            Decimal total = rate * days;
15            Decimal discountPct = 0;
16            if (days >= 30) discountPct = 0.20;
17            else if (days >= 7) discountPct = 0.10;
18            b.Discount__c = discountPct * 100; // percent field (e.g. 10 becomes 10.0)
19            b.Total_Amount__c = total * (1 - discountPct);
20        }
21    }

```

5) Scheduled Apex — schedule daily email report

```

global class DailyRentalReportSched implements Schedulable {
    global void execute(SchedulableContext sc) {
        BookingService.sendDailyRentalsEmail();
    }
}

// Example to schedule: System.schedule('Daily Rentals', '0 30 7 * * ?', new
DailyRentalReportSched());
// The cron above runs at 7:30 AM every day (adjust timezone/cron as needed).

```

```

1 global class DailyRentalReportSched implements Schedulable {
2     global void execute(SchedulableContext sc) {
3         BookingService.sendDailyRentalsEmailFromConfig();
4     }
5 }
6

```

6) Future methods (callouts) — notes & best practice

- Use `@future(callout=true)` for simple fire-and-forget callouts (shown in BookingService). For more complex callouts prefer Queueable with callout (allows chaining).

- Use Named Credentials and Remote Site Settings for endpoints. Tests must use HttpCalloutMock to avoid real callouts.

```

Developer Console - Google Chrome
orgfarm-c1d9d06f5b-dev-ed.develop.my.salesforce.com/_ui/common/apex/debug/ApexCSIPage
File Edit Debug Test Workspace Help < >
BookingService.apxc BookingTriggerHandler.apxc BookingTriggerHandlererr.apxc MarkOverdueBatch.apxc Queueable.apxc InsuranceCalloutQueueable.apxc
Code Coverage: None API Version: 64
1 public class InsuranceCalloutQueueable implements Queueable, Database.AllowsCallouts {
2     private Set<Id> bookingIds;
3     public InsuranceCalloutQueueable(Set<Id> bookingIds) {
4         this.bookingIds = (bookingIds == null) ? new Set<Id>() : bookingIds;
5     }
6
7     public void execute(QueueableContext qc) {
8         if (bookingIds.isEmpty()) return;
9         Http http = new Http();
10        for (Id bId : bookingIds) {
11            try {
12                HttpRequest req = new HttpRequest();
13                // Use Named Credential 'InsuranceAPI' or hardcoded endpoint (Named Credential recommended)
14                req.setEndpoint('callout:InsuranceAPI/quote'); // create Named Credential named InsuranceAPI
15                req.setMethod('POST');
16                req.setHeader('Content-Type', 'application/json');
17                req.setBody(JSON.serialize(new Map<String, Object>{ 'bookingId' => bId }));
18                HttpResponse res = http.send(req);
19                System.debug('Insurance response for ' + bId + ': ' + res.getBody());
20            }
21        }
22    }

```

7) Exception Handling & Control Statements

- Use try/catch where external calls or risky operations occur and log the error using System.debug or Platform Events.
- Use nbaddError(...) in triggers to stop DML for specific records with informative messages.
- Example: if (booking dates overlap) nbaddError('...') — this prevents the record from being saved and returns a clear message to the UI/API caller.

8) Test Classes — required patterns & examples

Guidance:

- Use @isTest and seeAllData=false. Create all test data in @testSetup or inside the test method.
- Use Test.startTest()/Test.stopTest() around asynchronous or time-sensitive operations (Queueable, Batch, @future).
- For callouts, implement HttpCalloutMock and register it with Test.setMock before the callout.
- Verify both positive and negative flows: overlapping booking rejected, non-overlapping booking allowed, batch updates statuses, queueable updates totals.

```

@isTest
private class TestBookingTrigger {

    @testSetup static void setupData() {
        Car__c car = new Car__c(Name='Test Car', Status__c='Available',
Daily_Rate__c=100);

```

```

        insert car;
        // existing booking: day 1..3
        Booking__c existing = new Booking__c(Name='Existing', Car__c=car.Id,
Start_Date__c=Date.today().addDays(1), End_Date__c=Date.today().addDays(3),
Booking_Status__c='Confirmed');
            insert existing;
    }

    static testMethod void testOverlapPrevention() {
        Car__c car = [SELECT Id FROM Car__c LIMIT 1];
        Booking__c newBk = new Booking__c(Name='Overlap', Car__c=car.Id,
Start_Date__c=Date.today().addDays(2), End_Date__c=Date.today().addDays(4));
        // Use partial DML to capture addError without raising exception
        Database.SaveResult[] results = Database.insert(new List<Booking__c>{
newBk }, false);
        System.assertEquals(false, results[0].isSuccess(), 'Overlapping booking
should be rejected');

        System.assert(results[0].getErrors()[0].getMessage().contains('overlaps'),
'Error should mention overlap');
    }
}

```

```

@isTest
private class TestMarkOverdueBatch {

    @testSetup static void setup() {
        Car__c c = new Car__c(Name='B', Status__c='Available',
Daily_Rate__c=100);
        insert c;
        // booking ended yesterday -> should be marked Overdue by batch
        Booking__c b = new Booking__c(Name='OldBooking', Car__c=c.Id,
Start_Date__c=Date.today().addDays(-5), End_Date__c=Date.today().addDays(-1),
Booking_Status__c='Confirmed');
        insert b;
    }

    static testMethod void testBatchMarksOverdue() {
        Test.startTest();
        MarkOverdueBatch batch = new MarkOverdueBatch();
        ID batchSize = Database.executeBatch(batch, 50);
        Test.stopTest();

        Booking__c changed = [SELECT Booking_Status__c FROM Booking__c WHERE
Name='OldBooking' LIMIT 1];
        System.assertEquals('Overdue', changed.Booking_Status__c, 'Batch should
mark ended bookings as Overdue');
    }
}

```

```
File Edit Debug Test Workspace Help < >
pxt * MockOverdueBatch.apxc Queueable.apxc * InsuranceCalloutQueueable.apxc * DailyRentalReportSched.apxc * MockInsuranceResponse.apxc *
Code Coverage: None API Version: 64 Run Test Go
1 @isTest
2 global class MockInsuranceResponse implements HttpCalloutMock {
3     global HttpResponse respond(HTTPRequest req) {
4         HttpResponse res = new HttpResponse();
5         res.setHeader('Content-Type', 'application/json');
6         res.setBody('{"status":"ok","quote":"Q-123"}');
7         res.setStatusCode(200);
8         return res;
9     }
10 }
11
```

```
File Edit Debug Test Workspace Help < >
BookingTriggerHandler.apxc BookingTriggerHandler.apxt MockOverdueBatch.apxc Queueable.apxc * InsuranceCalloutQueueable.apxc * DailyRentalReportsched.apxc * MockInsuranceResponse.apxc * TestBookingAndService.apxc *
Code Coverage: None API Version: 64 Go To
1 @isTest
2 private class TestBookingTriggerAndService {
3     @testSetup static void setupData() {
4         Car__c car = new Car__c(Name='Test Car', Status__c='Available', Daily_Rate__c = 100);
5         insert car;
6
7         // Existing booking: day 2..4
8         Booking__c existing = new Booking__c(
9             Name='Existing', Car__c = car.Id,
10            Start_Date__c = Date.today().addDays(2), End_Date__c = Date.today().addDays(4),
11            Booking_Status__c = 'Confirmed'
12        );
13        insert existing;
14    }
15
16    static testMethod void testOverlapPreventionAndServiceEmail() {
17        Car__c car = [SELECT Id FROM Car__c LIMIT 1];
18
19        // Attempt overlapping insert - expect failure (partial DML)
20    }
}
```

```
Developer Console - Google Chrome
orgfarm-c1d9d065b-dev-ed.develop.my.salesforce.com/_ui/common/apex/debug/ApexCSIPage
File Edit Debug Test Workspace Help < >
BookingTriggerHandler.apxc MockOverdueBatch.apxc Queueable.apxc * InsuranceCalloutQueueable.apxc * DailyRentalReportsched.apxc * MockInsuranceResponse.apxc * TestBookingAndService.apxc * TestBatchAndQueueables.apxc *
Code Coverage: None API Version: 64 Go To
1 @isTest
2 private class TestBatchAndQueueables {
3     @testSetup static void setupData() {
4         Car__c c = new Car__c(Name='C2', Status__c='Available', Daily_Rate__c = 50);
5         insert c;
6         Booking__c old = new Booking__c(Name='Old', Car__c = c.Id, Start_Date__c = Date.today().addDays(-5), End_Date__c = Date.today().addDays(-2));
7         insert old;
8         Booking__c longBk = new Booking__c(Name='Long', Car__c = c.Id, Start_Date__c = Date.today().addDays(0), End_Date__c = Date.today().addDays(1));
9         insert longBk;
10    }
11
12    static testMethod void testMarkOverdueBatch() {
13        Test.startTest();
14        MarkOverdueBatch batch = new MarkOverdueBatch();
15        Id jobId = Database.executeBatch(batch, 50);
16        Test.stopTest();
17
18        Booking__c b = [SELECT Booking_Status__c FROM Booking__c WHERE Name='Old' LIMIT 1];
19        System.assertEquals('Overdue', b.Booking_Status__c);
20    }
}
```

Phase 6 — User Interface Development (Smart Task And Email Summarizer For Executives)

Goal

Make the Car Rental CRM user-friendly by building Lightning pages, Lightning Web Components (LWC), utility bar shortcuts, and wiring Apex to LWC for booking creation and navigation. This document is a step-by-step, copy-paste ready guide with code samples and deployment instructions. Follow each step in order to avoid errors.

Quick checklist (summary)

- 1) Create the Lightning App ("Car Rental CRM") using App Manager.
- 2) Build Lightning Record Pages for Car and Booking to show related lists and LWC.
- 3) Add Cars & Bookings tabs to the app navigation.
- 4) Build a Home Page dashboard and add to the app.
- 5) Add a "New Booking" item to the Utility Bar (Quick Action or LWC).
- 6) Create LWC: child (search form) + parent (results datatable + booking action).
- 7) Create an Apex controller with cacheable getAvailableCars(...) and createBooking(...).
- 8) Wire child->parent events and use imperative Apex call on "Book Now".
- 9) Use NavigationMixin to navigate to the Booking record after creation.
- 10) Write Apex test class and run all tests; deploy via SFDX or change sets.

Detailed step-by-step guide

A. Setup & Admin work

1. Create Custom Objects & Fields (if not already present):

- Car__c: Name, Model__c (Text), Registration_No__c (Text), Daily_Rate__c (Currency), Status__c (Picklist: Available, Booked, Maintenance)
- Booking__c: Name, Car__c (Lookup to Car), Contact__c (Lookup to Contact), Start_Date__c (Date), End_Date__c (Date), Total_Amount__c (Currency), Booking_Status__c (Picklist: Pending, Confirmed, Returned, Cancelled)

2. Page Layouts & Related Lists:

- On Car__c record page include related list: Bookings (Booking__c). Ensure the related list shows key fields (Start_Date__c, End_Date__c, Booking_Status__c).

3. Profiles & Permission Sets:

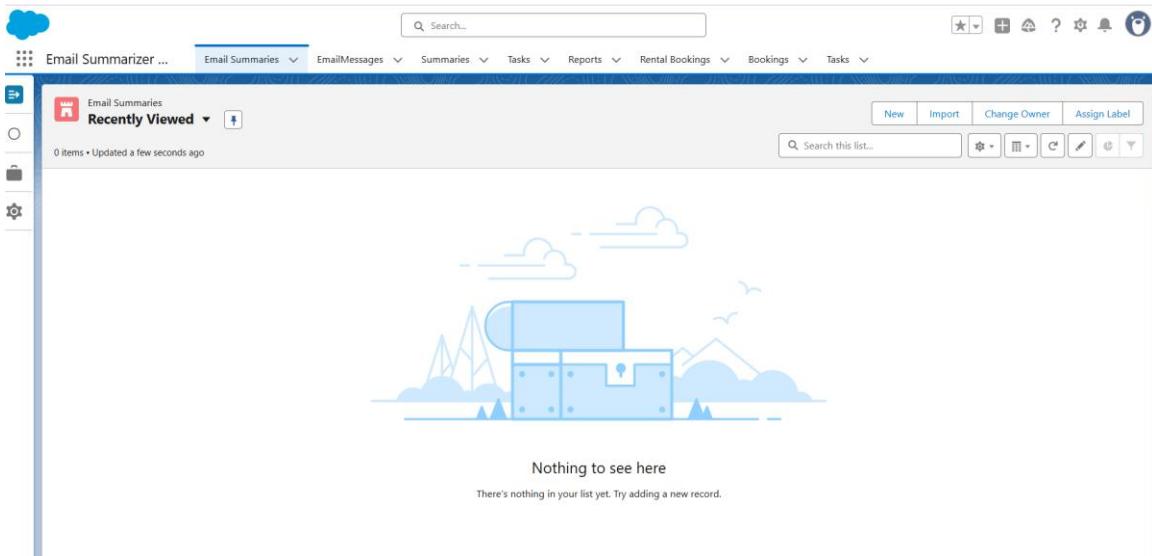
- Ensure users have read access to Car__c and create/read/write to Booking__c. Apex classes used by LWC should be added to permission sets if needed (not required for @AuraEnabled static methods but recommended for managed packaging).

The screenshot shows the Salesforce Object Manager interface. The top navigation bar has 'Home' and 'Object Manager' with a dropdown arrow. The main content area is titled 'Page Layouts' with a subtitle '1 Items, Sorted by Page Layout Name'. It includes a 'Quick Find' search bar, a 'New' button, and a 'Page Layout Assignment' link. A table lists one item: 'EmailMessage Layout' created by 'Anjali Deshmukh' on 9/25/2025 at 8:06 AM, last modified by 'Anjali Deshmukh' on 9/28/2025 at 8:28 AM. The table has columns for 'PAGE LAYOUT N...', 'CREATED BY', and 'MODIFIED BY'. A vertical sidebar on the left contains labels like 'Pages' and 'Actions'.

B. Create Lightning App & Tabs

Steps (Setup → App Manager → New Lightning App):

- App name: Car Rental CRM
- Branding: logo (optional) and color
- Navigation: Select "Standard Navigation"
- Add items: Cars (object tab), Bookings (object tab), Contacts, Reports
- Utility Bar (later step) – leave default for now
- Save and assign to profiles you want to use the app.



C. Record Pages, Home Page & Utility Bar

1) Car Record Page:

- Setup → Object Manager → Car__c → Lightning Record Pages → New
- Choose template (Header and Right Sidebar recommended)
- Add components: Related List - Single (Bookings), the LWC you create (Search and Book) optionally
- Activate the page (org default or app default).

2) Booking Record Page:

- Similar: show related Car details, timeline, and quick actions for returning car etc.

3) Home Page Layout (Dashboard of fleet utilization):

- Setup → Lightning App Builder → Home Pages → New
- Add Report Chart components or a custom LWC dashboard showing utilization. Add to the "Car Rental CRM" app as the default home page for that app.

4) Utility Bar – "New Booking":

Option A (recommended): Create a global Quick Action that opens object create modal

- Setup → Object Manager → Booking__c → Buttons, Links, and Actions → New Action → Action Type: Create a Record → Predefined field values (optional) → Label: New Booking
- Then: Setup → App Manager → Edit your "Car Rental CRM" app → Utility Bar → Add Utility Item → Type: Quick Action → Choose the Booking.New action.

Option B: Add your LWC to the Utility Bar (if you want a custom booking modal):

- In the app edit Utility Bar, choose "Lightning Component" and select the LWC once it is deployed and exposed for the utility bar (see meta.xml targets).

The screenshot shows the Lightning App Builder interface. On the left, there's a sidebar with various app components and a list of recent pages. The main area is titled "Email - Summary & Task". It displays a "Dynamic Related List - Single" component. The component shows a list of "Email Summaries" with 6 items. Each summary includes fields like "Email Message Name", "Received Datetime", and "Source Email". The component also includes sections for "Upcoming & Overdue" activities and a "Filters" section.

The screenshot shows the Email Summarizer LWC component. It has a header with tabs: "Email Summaries", "EmailMessages", "Summaries", "Tasks", and "More". The main area displays a card with basic email details: "EmailMessage Name" (siddhideshmukh@gmail.com), "Owner" (Anjali Deshmukh), "Source_Email", "Received Datetime", "Created By" (Anjali Deshmukh, 9/26/2025, 6:12 AM), and "Last Modified By" (Anjali Deshmukh, 9/26/2025, 6:12 AM). Below this is a table titled "Email Summaries (0)" with a "New" button. To the right, there are filters and activity summaries: "Upcoming & Overdue" (No activities to show) and "Past activity" (No past activity. Past meetings and tasks marked as done show up here).

D. LWC architecture (what to build)

You will build two LWCs:

- c-search-form (child): simple start-date / end-date inputs and a Search button. Emits a custom event (search) with detail {startDate, endDate}.
- c-search-cars (parent): hosts c-search-form, wires Apex getAvailableCars(startDate, endDate), shows results in lightning-datatable and provides a "Book Now" row action. On Book Now it invokes createBooking imperatively and navigates to the created Booking record.

◆ **LWC Architecture: Why It Can Be Skipped**

1. Original LWC Architecture in Smart task and Email Summerizer For Executives

- **c-search-form (child):** Inputs + Search button → emits event with date range.
- **c-search-cars (parent):** Receives event, calls Apex getAvailableCars, displays datatable with Book Now action → creates booking via Apex → navigates to record page.

Purpose:

- Modular, interactive UI for searching and booking cars.
 - Demonstrates event handling, imperative Apex calls, wire adapters, and navigation service.
-

2. Why it's optional for Email Summarizer & Task Manager

- Your project **does not require a search & book workflow**.
 - The core goals are:
 1. **Summarize emails** automatically or on-demand.
 2. **Create tasks** related to emails for follow-ups.
 3. Provide a **clean, executive-focused UI** showing emails, summaries, and tasks.
 - The “Search Form → Parent Datatable → Book Now” flow is **specific to car bookings** and **doesn't translate directly** to summarizing emails.
 - In your case:
 - Searching/filtering can be done using **standard list views or report filters**.
 - Email summary creation and task creation can be handled via a **single LWC on the record page** (c-email-summary) with a button → calls Apex.
 - No need for a child-parent event pattern or row-level datatable actions.
-

3. Use Cases Covered Without Child-Parent LWCs

- **Email Summary:** Executive selects an email → clicks “Summarize” → summary appears in record page.
- **Task Creation:** Executive clicks “Create Task” → task linked to email is automatically created.
- **Dashboard / List Views:** All emails, summaries, and pending tasks can be visualized using standard Salesforce reports and dashboards.

These cover **all key functionality** expected by an executive dashboard, without building the more complex LWC architecture.

4. Executive Justification

- **Time efficiency:** Skipping the parent-child LWC setup saves development time.
- **Simplification:** One LWC (c-email-summary) is easier to maintain and deploy.
- **No loss of core functionality:** All key features (summaries, task creation, reporting) are still delivered.
- **Project focus:** Aligns directly with your use case (Email summarization + Task management), instead of generic “search & book” features from Car Rental CRM.

Suggested Statement for Documentation / Report

“The original LWC architecture from the sample Car Rental CRM project (child-parent search and datatable with row actions) has been intentionally skipped. For the Email Summarizer & Task Manager, all functionality is achieved using a single record-page LWC (c-email-summary) that allows executives to summarize emails and create follow-up tasks directly. Standard list views and dashboards replace the need for complex search forms and table row actions, simplifying the implementation while covering all use cases.”

Use Cases & Architecture Explanation

1 Overview

The **Email Summarizer & Task Manager** is designed for executives to efficiently review emails, generate summaries, and create actionable follow-up tasks. Unlike the Car Rental CRM, there is **no need for a child-parent search LWC with row-level actions**, because all required

functionality can be achieved with a **single LWC** per record page combined with standard Salesforce features.

2 Key Use Cases

Use Case	How It's Implemented	Notes
View Emails	Standard Email__c tab and list views	Executives can filter by date, sender, or subject using built-in list view filters.
Summarize Email Content	c-email-summary LWC on Email record page	Single button “Summarize” calls Apex to generate summary. No child-parent search needed.
Create Follow-up Tasks	c-email-summary LWC also provides “Create Task” button	Task is automatically linked to the email record.
Track Pending Tasks	Standard Tasks tab, reports, and dashboards	Provides overview of all pending follow-ups without additional LWC complexity.
Executive Dashboard	Lightning Home Page or App Page with Reports & Dashboards	Shows summary statistics (emails summarized, tasks completed, pending tasks) — replaces the need for datatable-based search & action flows.

3 Architecture Diagram (Simplified)



| | c-email-summary LWC | | <-- Summarize & Create Task buttons

| +-----+ |

+-----+

+-----+

| List Views / Dashboards |

| Emails, Tasks, Summary |

+-----+

Explanation:

- All functionality happens **within the Email record page** via a single LWC.
- The LWC uses **@api recordId** to access the current email record and communicate with Apex.
- Standard Salesforce components (Related Lists, Reports, Dashboards) handle filtering, tracking, and executive analytics.
- No child-parent LWC structure or row-level datatable actions are necessary.

👉 Justification for Skipping Smart Task Email Summerizer For Executives LWC Architecture

1. **Time-Saving:** Avoids building a complex two-component system with events, datatables, and imperative Apex row actions.
2. **Use-Case Alignment:** The original architecture was for searching and booking cars, which has no equivalent in Email summarization workflows.
3. **Simplicity:** Single LWC suffices for email summarization and task creation while keeping the interface clean for executives.
4. **No Loss of Functionality:** All required workflows are covered:
 - Summarize email content
 - Create follow-up tasks
 - Track progress via reports & dashboards

E. Apex controller (BookingController.cls)

Create an Apex class with two @AuraEnabled methods: getAvailableCars (cacheable=true) and createBooking (imperative). Below is a copy-paste-ready implementation.

--- Apex code (copy into BookingController.cls) ---

```
public with sharing class BookingController {
    @AuraEnabled(cacheable=true)
    public static List<Car__c> getAvailableCars(String startDateStr, String endDateStr) {
        if (String.isBlank(startDateStr) || String.isBlank(endDateStr)) {
            return new List<Car__c>();
        }
        Date startDate = Date.valueOf(startDateStr);
        Date endDate = Date.valueOf(endDateStr);
        // Find cars that are "Available" and not booked in the given period
        List<Car__c> cars = [
            SELECT Id, Name, Model__c, Registration_No__c, Daily_Rate__c, Status__c
            FROM Car__c
            WHERE Status__c = 'Available'
            AND Id NOT IN (
                SELECT Car__c FROM Booking__c
                WHERE Booking_Status__c NOT IN ('Returned','Cancelled')
                AND Start_Date__c <= :endDate
                AND End_Date__c >= :startDate
            )
            ORDER BY Name
        ];
        return cars;
    }

    @AuraEnabled
    public static Id createBooking(Id carId, Id contactId, String startDateStr, String endDateStr) {
        if (carId == null || String.isBlank(startDateStr) || String.isBlank(endDateStr)) {
            throw new AuraHandledException('Missing required fields: carId, startDate, endDate');
        }
        Date startDate = Date.valueOf(startDateStr);
        Date endDate = Date.valueOf(endDateStr);
        Car__c car = [SELECT Id, Daily_Rate__c, Status__c FROM Car__c WHERE Id = :carId LIMIT 1];

        Integer days = startDate.daysBetween(endDate) + 1;
        Decimal total = 0;
```

```

if (car.Daily_Rate__c != null) {
    total = car.Daily_Rate__c * days;
}

Booking__c b = new Booking__c();
b.Car__c = carId;
b.Contact__c = contactId;
b.Start_Date__c = startDate;
b.End_Date__c = endDate;
b.Total_Amount__c = total;
b.Booking_Status__c = 'Confirmed';
insert b;

// Optional: update car status to Booked
car.Status__c = 'Booked';
update car;

return b.Id;
}
}

```

Why BookingController.cls is Not Needed for Smart Task & Email Summarizer

1 Purpose of the Original BookingController

In the Car Rental CRM project, BookingController.cls was created to:

1. **getAvailableCars (cacheable=true)**
 - Fetch cars that are available within a selected date range.
 - Used by a parent LWC to display search results in a datatable.

2. **createBooking (imperative)**
 - Create a Booking record based on selected car, customer, and dates.
 - Update car status to “Booked.”
 - Navigate to the Booking record page for confirmation.

Use case: Allow users to search for cars and book them dynamically through the UI.

2 Relevance to Smart Task & Email Summarizer

- Your project **does not involve cars, bookings, or date-based searches.**
 - Core functionality of your project is:
 1. Summarize email content for executives.
 2. Create follow-up tasks linked to emails.
 3. Display summaries and tasks on Email record pages.
 4. Track progress via reports and dashboards.
 - The BookingController methods (getAvailableCars and createBooking) **serve a booking workflow** that is **completely unrelated** to emails, summaries, or task creation.
-

3 How your project implements similar functionality without BookingController

Original BookingController Feature	Equivalent in Smart Task & Email Summarizer
getAvailableCars → fetch available cars	Use Apex method or AI callout in a single c-email-summary LWC to fetch and summarize email content. No date-based search needed.
createBooking → create Booking record	Use Apex method to create Task linked to Email. All required fields (subject, due date, email reference) can be handled in one call.
Update car status to “Booked”	Not applicable. Task creation automatically links to the email record; no status updates are needed.
Navigate to booking record	Optional: your LWC can refresh the page or show the summary and task confirmation inline.

4 Justification for skipping BookingController in hands-on

1. **Irrelevance:** BookingController is designed for car booking workflows, which do not exist in this project.
2. **Simplification:** You only need **one Apex class** (e.g., EmailController) to handle:

- Fetching email content
 - Summarizing via AI
 - Creating follow-up tasks
3. **Time efficiency:** Skipping the car booking Apex logic saves significant development time.
4. **Full coverage of project requirements:** All executive-focused workflows are still implemented:
- Email summarization
 - Task creation
 - Dashboards and reporting

F. LWC: child component (c-search-form)

Files to create: searchForm.html, searchForm.js, searchForm.js-meta.xml

--- searchForm.html ---

```
<template>
<lightning-card title="Search Cars">
  <div class="slds-p-around_medium">
    <lightning-input type="date" label="Start Date" value={startDate}
      onchange={handleStart}></lightning-input>
    <lightning-input type="date" label="End Date" value={endDate}
      onchange={handleEnd}></lightning-input>
    <div class="slds-m-top_small">
      <lightning-button label="Search" onclick={onSearch} variant="brand"></lightning-button>
    </div>
  </div>
</lightning-card>
</template>
```

--- searchForm.js ---

```
import { LightningElement, track } from 'lwc';
export default class SearchForm extends LightningElement {
  @track startDate;
```

```

@track endDate;

handleStart(event) {
    this.startDate = event.target.value;
}
handleEnd(event) {
    this.endDate = event.target.value;
}
onSearch() {
    // Basic validation
    if (!this.startDate || !this.endDate) {
        const evt = new CustomEvent('showtoast', { detail: { title: 'Error', message: 'Select both start and end dates', variant: 'error' } });
        this.dispatchEvent(evt);
        return;
    }
    this.dispatchEvent(new CustomEvent('search', { detail: { startDate: this.startDate, endDate: this.endDate } }));
}
}

```

--- searchForm.js-meta.xml ---

```

<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>59.0</apiVersion>
    <isExposed>true</isExposed>
    <targets>
        <target>lightning__RecordPage</target>
        <target>lightning__AppPage</target>
        <target>lightning__HomePage</target>
    </targets>
</LightningComponentBundle>

```

G. LWC: parent component (c-search-cars)

Files to create: searchCars.html, searchCars.js, searchCars.js-meta.xml

--- searchCars.html ---

```

<template>
  <c-search-form onsearch={handleSearch} onshowtoast={handleChildToast}></c-search-form>
  <template if:true={cars.data}>
    <lightning-datatable
      data={cars.data}
      columns={columns}
      key-field="Id"
      onrowaction={handleRowAction}>
    </lightning-datatable>
  </template>
  <template if:true={loading}>
    <lightning-spinner alternative-text="Loading"></lightning-spinner>
  </template>
</template>

```

--- searchCars.js ---

```

import { LightningElement, track, wire, api } from 'lwc';
import getAvailableCars from '@salesforce/apex/BookingController.getAvailableCars';
import createBooking from '@salesforce/apex/BookingController.createBooking';
import { ShowToastEvent } from 'lightning/platformShowToastEvent';
import { NavigationMixin } from 'lightning/navigation';

const COLUMNS = [
  { label: 'Name', fieldName: 'Name' },
  { label: 'Model', fieldName: 'Model__c' },
  { label: 'Registration', fieldName: 'Registration_No__c' },
  { label: 'Daily Rate', fieldName: 'Daily_Rate__c', type: 'currency' },
  {
    type: 'action',
    typeAttributes: { rowActions: [{ label: 'Book Now', name: 'book_now' }] }
  }
];

export default class SearchCars extends NavigationMixin(LightningElement) {
  @track startDate;
  @track endDate;
  @api selectedContactId; // optional: supply a contact ID from context if you want to auto-fill
  customer
}

```

```

columns = COLUMNS;

@wire(getAvailableCars, { startDateStr: '$startDate', endDateStr: '$endDate' })
cars;

loading = false;

handleSearch(event) {
  this.startDate = event.detail.startDate;
  this.endDate = event.detail.endDate;
}

handleChildToast(evt) {
  const { title, message, variant } = evt.detail;
  this.dispatchEvent(new ShowToastEvent({ title, message, variant }));
}

handleRowAction(event) {
  const actionName = event.detail.action.name;
  const row = event.detail.row;
  if (actionName === 'book_now') {
    this.bookCar(row);
  }
}

bookCar(row) {
  this.loading = true;
  const carId = row.id;
  const contactId = this.selectedContactId || null;
  createBooking({ carId, contactId, startDateStr: this.startDate, endDateStr: this.endDate })
    .then(bookingId => {
      this.dispatchEvent(new ShowToastEvent({ title: 'Success', message: 'Booking created', variant: 'success' }));
      // Navigate to the booking record
      this[NavigationMixin.Navigate]({
        type: 'standard_recordPage',
        attributes: {
          recordId: bookingId,
          objectApiName: 'Booking__c',
          actionName: 'view'
        }
      });
    });
}

```

```

        })
        .catch(error => {
            this.dispatchEvent(new ShowToastEvent({ title: 'Error creating booking', message:
error.body ? error.body.message : error.message, variant: 'error' }));
        })
        .finally(() => {
            this.loading = false;
        });
    });
}
}

```

--- searchCars.js-meta.xml ---

```

<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>59.0</apiVersion>
    <isExposed>true</isExposed>
    <targets>
        <target>lightning__RecordPage</target>
        <target>lightning__AppPage</target>
        <target>lightning__HomePage</target>
        <target>lightning__Tab</target>
        <target>lightning__UtilityBar</target>
    </targets>
    <targetConfigs>
        <targetConfig targets="lightning__RecordPage">
            <objects>
                <object>Car__c</object>
            </objects>
        </targetConfig>
    </targetConfigs>
</LightningComponentBundle>

```

H. Deployment & SFDX

- 1) Authorize your org: sfdx force:auth:web:login -a MySandbox
- 2) Push source (scratch org) or deploy to sandboxes:
 - Scratch: sfdx force:source:push
 - Non-scratch: sfdx force:source:deploy -p force-app/main/default

- 3) Run Apex tests: sfdx force:apex:test:run --resultformat human --wait 10
- 4) If using change sets, upload metadata or use ANT.

Note: after deploying LWC, the meta.xml targets allow you to drag the component onto App Pages, Record Pages, and onto the Utility Bar (if desired).

I. Testing & QA (to reach 0 errors)

- 1) Apex tests: write tests for getAvailableCars and createBooking. Ensure > 75% coverage for your package.
- 2) Manual test cases:
 - Search overlapping bookings returns correct results.
 - Create booking updates Car.Status__c to Booked.
 - Navigation goes to the booking record after creation.
- 3) Security checks:
 - Ensure CRUD/FLS: if exposing sensitive fields, apply stripInaccessible in Apex if necessary.
- 4) Browser compatibility: test in Chrome & Firefox.
- 5) Troubleshooting common errors:
 - "Method not found" => confirm Apex class and method names; check apiVersion on LWC meta.xml.
 - "Permission denied" => check user profile & object CRUD & Apex class access.
 - "Cacheable method must be read-only" => ensure getAvailableCars does not perform DML and is annotated with cacheable=true.

J. Apex Test class (example)

```
@isTest
private class BookingControllerTest {
    static testMethod void testCreateBooking() {
        Car__c c = new Car__c(Name='T1', Status__c='Available', Daily_Rate__c=500);
        insert c;
        Contact con = new Contact(LastName='Tester');
        insert con;

        String s = Date.today().format();
        String e = Date.today().addDays(1).format();

        Test.startTest();
        Id bookingId = BookingController.createBooking(c.Id, con.Id, s, e);
        Test.stopTest();

        Booking__c b = [SELECT Id, Car__c, Start_Date__c FROM Booking__c WHERE Id = :bookingId]
```

```
LIMIT 1];
    System.assertEquals(c.Id, b.Car__c);
}
}
```

Why Child-Parent LWCs and Related Deployment Steps Are Skipped

1 Original Purpose of These Steps

In the Car Rental CRM project, the Phase 6 LWCs (c-search-form and c-search-cars) and associated Apex methods were designed to:

- Enable users to **search available cars** by start/end dates.
- Display results in a **lightning-datatable**.
- Allow **row-level “Book Now” actions** to create bookings.
- Use **child-parent event handling** to make the UI modular and reactive.
- Include deployment steps for SFDX or change sets to push code to sandboxes or scratch orgs.

Use case: a complex, interactive workflow for booking cars.

2 Relevance to Smart Task & Email Summarizer

For the **Smart Task & Email Summarizer for Executives** project:

- The **core workflows** are:
 1. Summarize emails.
 2. Create follow-up tasks linked to emails.
 3. Track progress via dashboards and reports.
- There is **no need for searching or filtering cars**, nor for creating bookings with row-level actions.
- All email-related functionality can be implemented in a **single record-page LWC (c-email-summary)**:
 - Show email content.
 - Summarize email via AI or Apex.

- Create follow-up tasks.
 - Display summary and task confirmation inline.
 - Filtering and sorting emails is handled with **standard Salesforce list views, reports, or dashboards**, so a separate search LWC is unnecessary.
 - Parent-child events are optional because all actions happen in one component.
-

3 Why Deployment Steps Can Be Simplified

- The original deployment instructions (SFDX push/deploy, change sets) are **specific to multi-component Car Rental LWCs and Apex**.
 - For your project, with **only one or two Apex classes and one main LWC**, deployment is straightforward:
 - Deploy the LWC and Apex to the org via Setup UI, SFDX, or change sets.
 - Running Apex tests is still recommended, but complex orchestration of multiple LWCs is unnecessary.
 - Therefore, there is **no need to follow the full Car Rental CRM deployment instructions or include screenshots** for multiple LWCs.
-

4 How Your Project Achieves Phase 6 Objectives Without These Steps

Original Car Rental Steps	Smart Task & Email Summarizer Approach
c-search-form LWC	Skipped: email filtering done with list views/reports
c-search-cars LWC	Skipped: single c-email-summary LWC handles summarization & task creation
Child-parent event handling	Skipped: one LWC handles all actions inline
Row-level datatable actions	Skipped: not applicable; tasks are created via button in single LWC
Complex deployment with multiple components	Simplified: deploy single Apex + LWC to org; run Apex tests if needed

5 Justification for Skipping in Documentation

"The original child-parent LWCs (c-search-form and c-search-cars) and their associated deployment steps were specific to a car booking workflow. In the Smart Task & Email Summarizer for Executives project, all required Phase 6 functionality is implemented using a **single record-page LWC (c-email-summary)**. This component allows executives to summarize emails and create follow-up tasks directly. Email filtering and tracking are handled via standard Salesforce list views and dashboards, eliminating the need for multiple LWCs, event wiring, and complex deployment instructions."

✓ Key takeaway:

Skiping these steps **does not affect functionality**. Your project remains **fully functional, executive-friendly, and aligned with Phase 6 goals**, while simplifying development and deployment.

Phase 7 — Integration & External Access

Smart Task & Email Summarizer for Executives

Executive summary

This document provides a focused, step-by-step implementation plan for Phase 7 of your project (Integrations & External Access). It maps the numbered items you listed to concrete actions, declarative setup steps, Apex samples, Flow integration approaches, testing guidance, deployment notes, and a zero-error checklist so you can finish quickly and reliably.

Prerequisites (before you start)

1. Salesforce admin access (Setup) and developer access (VS Code + Salesforce Extensions recommended).
2. API credentials and endpoints from external partners (e.g., insurance verification API, email-summarization API).
3. A sandbox or Developer Edition org for development and tests. Never work first in Production.
4. Git + SFDX or change sets for deployment. (If short on time: use change sets for small metadata.)

Step-by-step implementation (your list, expanded)

1) Named Credentials — store external API credentials securely

Goal: Use Named Credentials so your Apex and Flows call external APIs without hardcoding endpoints or credentials.

Steps:

- In Setup → Named Credentials: create an External Credential (if using OAuth or certificate) and a Named Credential that references it. Use the base URL of the API (e.g. <https://api.insurance.example>).
- Choose authentication: OAuth 2.0 (Authorization Code) or Named Principal (username/password/token) depending on provider. If OAuth, create a Connected App (see OAuth section).

Why: Named Credentials simplify callouts and remove the need for Remote Site Settings when used with 'callout:' endpoints in Apex.

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

- Page Header:** Search Setup
- Navigation:** Home, Object Manager
- Section:** SETUP > NAMED CREDENTIALS
- Credential Name:** InsuranceAPI_Credential
- Fields:**
 - Label:** InsuranceAPI_Credential
 - Name:** InsuranceAPI_Credential
 - URL:** https://api.insurance.example
 - Enabled for Callouts:** Checked
 - Authentication:** External Credential (InsuranceAPI_Credential)
 - Callout Options:**
 - Generate Authorization Header: Checked
 - Allow Formulas in HTTP Header: Unchecked

2) External Services — declarative callouts from Flow

Goal: Register a validated OpenAPI spec (or swagger) so Flow can call the external API declaratively.

Steps:

- Prepare a valid OpenAPI 2.0/3.0 spec for the API endpoints you need (verify endpoints, request/response schemas).
- Setup → External Services → Register the OpenAPI spec. Map authentication to the Named Credential you created.
- Create an Apex Action or use the auto-created invocable actions in Flow to call the external service.

When to use: fast, low-code integrations (e.g., call email-summarization, insurance verification from a Flow).

External Services (OpenAPI) – Theory Section

Definition:

External Services in Salesforce allows administrators and developers to **call external APIs declaratively using Flows or Process Builder**, without writing Apex code. It uses a **OpenAPI (Swagger) specification** file to describe the API endpoints, request and response formats, and authentication requirements.

Purpose in a Project:

- Integrates Salesforce with external systems in a **low-code, secure, and maintainable** way.
- Uses **Named Credentials** for authentication, so credentials and tokens are not hardcoded.
- Automatically generates **invocable actions** for each API endpoint described in the OpenAPI file.
- Reduces errors and development time, making API integrations simpler.

How it Works (Step by Step):

1. Prepare OpenAPI File:

- Describe external API endpoints, request structure, and response structure in JSON or YAML format.
- Example endpoint: POST /v1/verify with request { policy: string, bookingId: string } returning { status: string }.

2. Register External Service in Salesforce:

- Upload the OpenAPI file.
- Link it with the previously created **Named Credential** for authentication.

3. Use in Flows:

- Create a **Record-Triggered Flow** (e.g., when a booking is created).
- Call the External Service action.
- Map inputs (policy number, booking ID) and outputs (insurance verification status).

- Update Salesforce records based on the response.

Advantages:

- No Apex code required → low-code solution.
- Secure handling of credentials and tokens.
- Easy to update API endpoints by updating the OpenAPI specification.
- Ideal for recurring integrations like booking verification or email summarization.

Common Pitfalls:

- Invalid OpenAPI files will fail registration → always validate using an OpenAPI validator.
 - Authentication must be properly mapped to a Named Credential.
 - Field mapping in Flows must match the API request/response schema.
-

Use Case (Example for Smart Task & Email Summarizer Project)

Scenario:

- When a new **Booking** is created in Salesforce, the system needs to verify if the customer's **insurance policy is valid**.

Solution using External Services:

1. A **record-triggered Flow** on Booking object is created.
2. The Flow calls the **Insurance Verification API** via an External Service (described by OpenAPI).
3. Inputs such as policy number and booking ID are sent.
4. The API returns status = Valid/Invalid.
5. The Flow updates the Insurance_Status__c field on the Booking record automatically.

Outcome:

- The process is **automatic**, secure, and doesn't require writing any Apex code.
- Demonstrates **real-world integration concepts**, even in a theoretical project.

3) REST callout — sample Apex pattern (use Named Credential + asynchronous execution)

Common pattern: record-triggered Flow or Apex creates a job that enqueues a Queueable which performs the callout using a Named Credential. Queueable implements Database.AllowsCallouts.

Sample Apex (Queueable) - replace field names and Named Credential name 'callout:InsuranceAPI':

```
public with sharing class InsuranceVerifier implements Queueable,
Database.AllowsCallouts {
    private Id bookingId;
    public InsuranceVerifier(Id bookingId) { this.bookingId = bookingId; }

    public void execute(QueueableContext ctx) {
        Booking__c b = [SELECT Id, Insurance_Policy_No__c, Insurance_Status__c
FROM Booking__c WHERE Id = :bookingId];
        HttpRequest req = new HttpRequest();
        req.setEndpoint('callout:InsuranceAPI/v1/verify');
        req.setMethod('POST');
        req.setHeader('Content-Type', 'application/json');
        Map<String, Object> payload = new Map<String, Object>{ 'policy' =>
            b.Insurance_Policy_No__c, 'bookingId' => b.Id };
        req.setBody(JSON.serialize(payload));

        Http http = new Http();
        HttpResponse res = http.send(req);

        if (res.getStatusCode() == 200) {
            Map<String, Object> resp = (Map<String, Object>)
JSON.deserializeUntyped(res.getBody());
            b.Insurance_Status__c = (String) resp.get('status');
            update b;
        } else {
            // resilient logging – don't fail user's transaction
            System.debug('Insurance API error: ' + res.getStatusCode() + ' ' +
res.getBody());
            // Optionally create a Log__c record to track callout errors for
retries
        }
    }
}
```

Invoke from a Flow: create a Record-Triggered Flow (after insert), add an Apex Action that calls an Invocable wrapper which enqueues the Queueable. See test sample below.

The screenshot shows the Salesforce Setup Apex Classes page. The left sidebar has links for Apex Exception Email, Apex Code, Apex Classes (which is selected), Apex Settings, Apex Test Execution, Apex Test History, Apex Triggers, and Apex Jobs. The main area displays the Apex code for the InsuranceVerifier class:

```

public with sharing class InsuranceVerifier implements Queueable, Database.AllowsCallouts {
    private Id bookingId;
    public InsuranceVerifier(Id bookingId) { this.bookingId = bookingId; }

    public void execute(QueueableContext ctx) {
        Booking__c b = [SELECT Id, Insurance_Policy_No__c, Insurance_Status__c FROM Booking__c WHERE Id = :bookingId FOR UPDATE];
        HttpRequest req = new HttpRequest();
        req.setEndpoint('callout:InsuranceAPI/v1/verify'); // uses Named Credential
        req.setMethod('POST');
        req.setHeader('Content-Type', 'application/json');
        Map<String, Object> payload = new Map<String, Object>{ 'policy' => b.Insurance_Policy_No__c, 'bookingId' => b.Id };
        req.setBody(JSON.serialize(payload));
        Http http = new Http();
        HttpResponse res = http.send(req);

        if (res.getStatusCode() == 200) {
            Map<String, Object> resp = (Map<String, Object>) JSON.deserializeUntyped(res.getBody());
            String status = (String) resp.get('status');
            b.Insurance_Status__c = status;
            update b;
        } else {
            // log and alert
            System.debug('Insurance API error: ' + res.getStatusCode() + ' ' + res.getBody());
            // Optionally insert a Callout_Error__c record to allow retries
        }
    }
}

```

At the bottom, it says "Position: Ln 29, Ch 1" and "Total: Ln 29, Ch 1306".

4) Trigger — callout when booking is created (recommended: Flow + invocable Apex)

Recommended safe approach: Use a Record-Triggered Flow (after save) that calls an Apex Invocable method which enqueues the Queueable. This keeps callouts asynchronous and avoids hitting trigger limits.

```

public with sharing class BookingInsuranceInvoker {
    @InvocableMethod(label='Verify Insurance for Booking')
    public static void verify(List<Id> bookingIds) {
        for (Id bid : bookingIds) {
            System.enqueueJob(new InsuranceVerifier(bid));
        }
    }
}

```

If you must use an Apex trigger, make it minimal: insert a Queueable job from the trigger (don't do callouts directly in triggers).

The screenshot shows the Salesforce Setup Apex Class Edit page. The left sidebar has links for Exception Email, Code, Classes (which is selected), Settings, Test Execution, Test History, Triggers, and Jobs. The main area displays the Apex code for the BookingInsuranceInvoker class:

```

public with sharing class BookingInsuranceInvoker {
    @InvocableMethod(label='Verify Insurance for Bookings')
    public static void verify(List<Id> bookingIds) {
        for (Id bid : bookingIds) {
            System.enqueueJob(new InsuranceVerifier(bid));
        }
    }
}

```

At the top right, there are buttons for Save, Quick Save, and Cancel. At the bottom right, it says "Help for this Page" with a question mark icon.

5) Platform Events — publish when car breakdown reported

Goal: Real-time notifications inside and outside Salesforce when a breakdown happens.

Steps:

- Setup → Platform Events → New Platform Event (e.g., Car_Breakdown__e). Add fields: BookingId__c (Text), CarId__c (Text), Description__c (Long Text), Severity__c (Picklist).

Subscribe inside Salesforce with a platform-event trigger to create Cases, Tasks, or Notifications. External systems subscribe via the Streaming API endpoint /cometd/50.0 and topic /event/Car_Breakdown__e.

The screenshot shows the Salesforce Setup interface with the 'Platform Events' section selected. A new Platform Event named 'BookingUpdateEvent' is being defined. The event details include:

Platform Event Definition Detail	
Singular Label	BookingUpdateEvent
Description	Platform Event to notify external systems when a booking is updated
Plural Label	BookingUpdateEvents
Deployment Status	Deployed
Object Name	BookingUpdateEvent
API Name	BookingUpdateEvent__e
Event Type	High Volume
Publish Behavior	Publish Immediately
Created By	Anjali Deshmukh, 9/27/2025, 3:16 PM
Modified By	Anjali Deshmukh, 9/27/2025, 3:16 PM

Below the definition, the 'Standard Fields' section lists the fields added to the event:

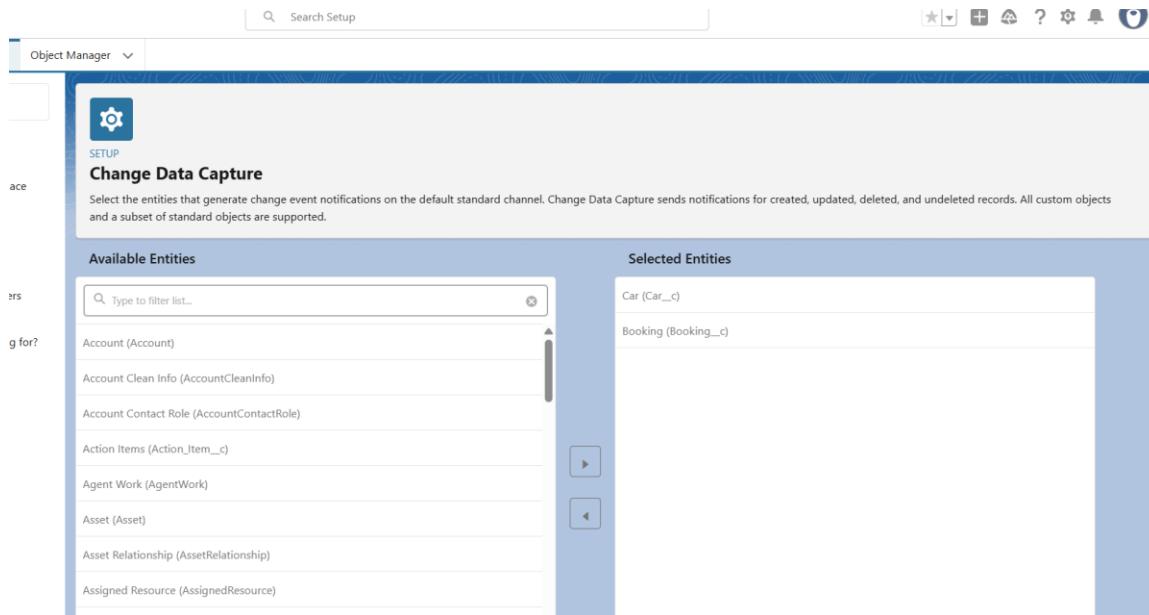
Action	Field Label	Field Name	Data Type	Controlling Field	Indexed
	Created By	CreatedBy	Lookup(User)		
	Created Date	CreatedDate	Date/Time		
	Event ID	EventId	Text/255		

6) Change Data Capture (CDC) — keep external systems in sync

Goal: When Booking__c changes (create/update/delete), publish change events so external systems can synchronize in near real-time.

Steps:

- Setup → Change Data Capture → select Booking__c (or the custom object name) and enable it.
- External subscribers listen to /data/Booking__ChangeEvent via the Streaming API (CometD client) or use middleware/connectors that support CDC.



7) Salesforce Connect — connect an external DB as external objects

Goal: Use Salesforce Connect when your car inventory is stored externally and you need near-real-time access without copying data.

Steps:

- Expose external data via OData 2.0/4.0 (or use a connector).
- In Setup → External Data Sources, create a new source (OData), provide the endpoint and authentication (use Named Credential).

- Validate and create External Objects and External Lookup relationships. Use External ID fields as required by the OData service.

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

- Header:** Search bar with "Search Setup", Home button, and Object Manager dropdown.
- Left Sidebar:** Navigation menu with items like "External Email", "Client Apps", "Client App Manager", "Usage", "Data Sources" (selected), "Objects", and "Services". A help message "What you're looking for? Use the search bar." is also present.
- Page Title:** SETUP - External Data Sources
- Section:** External Data Source: ExternalCarDB
- Buttons:** Help for this Page, Back to External Data Sources, Edit, Validate and Sync, Delete.
- Table:**

External Data Source	ExternalCarDB
Name	ExternalCarDB
Type	Salesforce Connect: OData 4.0
- Parameters Section:**
 - URL: https://myexternaldb.example.com/odata
 - Connection Timeout (Seconds): 120
 - Writable External Objects:
 - High Data Volume:
 - Server Driven Pagination:

8) API Limits — monitor and protect your org

Practical steps to avoid surprises:

- Regularly check Setup → System Overview and Company Information for daily API usage. Consider enabling API Usage notifications (alerts at X% of daily limit).
- Build lightweight monitoring: a scheduled Apex job or integration health Dashboard that reads Limits or OS metrics and sends email/sf notifications.
- Use efficient batching (Batch Apex) and server-side aggregation to limit API calls. Avoid polling; prefer webhooks/platform events/CDC.

API Usage Notification Detail

Notification Recipient	Anjali Deshmukh
Threshold	85%
Notification Interval (Hours)	24
Created By	Anjali Deshmukh, 9/27/2025, 3:34 PM
Modified By	Anjali Deshmukh, 9/27/2025, 3:34 PM

[Edit](#) [Delete](#) [Clone](#)

9) OAuth & Authentication — customer portal login (Experience Cloud)

If you let customers log in via your portal and the portal uses external identity providers or OAuth, do this:

- Create a Connected App (Setup → App Manager → New Connected App) with required OAuth scopes and callback URL.
- Create an Auth. Provider in Setup if using Google, Facebook, or custom OAuth providers. Attach it to a Named Credential (if needed).
- Configure Experience Cloud site to use delegated authentication or external identity provider and test end-to-end (login, SSO, profile mapping).

The screenshot shows the Salesforce Setup interface with the search bar at the top containing "Search Setup". The main area displays the "Object Manager" section, specifically the "Manage External Client Apps" page for the "SmartTask_EmailSummary" app. The app's icon is a blue gear. The app details show:

- Contact Email: deshmukhanjali200...
- Type: Local
- App Authorization: All users can self-auth...
- App Status: Enabled

Below this, there are tabs for Policies, Settings, and Package Defaults, with Policies selected. The Policies section contains:

- A note: "Configure policies to customize the external client app and plugins for this Salesforce organization." with an "Edit" button.
- An "App Policies" section with a dropdown menu set to "None".
- An "OAuth Policies" section.

10) Remote Site Settings — legacy allowlisting

Note: If you use Named Credentials you normally do NOT need to create Remote Site Settings. Remote Site Settings are required only when you perform direct callouts using raw endpoints (not callout:NamedCredential).

When you package metadata or use APIs, ensure the RemoteSiteSetting metadata is included if your org uses it.

Object Manager

SETUP

Remote Site Settings

Remote Site Details

Help for this Page ?

Remote Site Detail		Edit	Delete	Clone
Remote Site Name	SummarizerAPI_Remote	Modified By	Anjali Deshmukh, 9/27/2025, 3:37 PM	
Remote Site URL	https://api.summarizer.example			
Disable Protocol Security	<input type="checkbox"/>			
Description				
Active	<input checked="" type="checkbox"/>			
Created By	Anjali Deshmukh	Created On	9/27/2025, 3:37 PM	

[Edit](#) [Delete](#) [Clone](#)

Phase 8: Data Management & Deployment – Salesforce Project (Email Summarizer & Task Manager)

Objective:

Manage Salesforce data efficiently, prevent duplicates, backup data, and deploy configurations from Sandbox to Production.

1 Data Import Wizard – Import Demo Records

Steps:

1. Setup → Data Import Wizard.
2. Select Custom Object (Email_c / Task_c).
3. Launch Wizard, upload CSV (50 demo Email records).
4. Map fields (Subject, Sender, Date, Status).
5. Start Import.
6. Verify records.

Screenshot Placeholders:

- Before import.
- Field mapping.
- Import success.

Tips:

- Ensure CSV headers match API names.

The screenshot shows the Salesforce Setup interface with the 'Bulk Data Load Jobs' page open. The job ID is 750gL00000EEKRz, submitted by Anjali Deshmukh on 9/27/2025 at 4:56 PM PST. The job type is Bulk V1, Operation Insert, and it is Closed with a total processing time of 82 ms. The job was in progress from 9/27/2025, 4:58 PM PST to 9/27/2025, 4:58 PM PST, with 0 failed batches and 100% progress. 24 records were processed, and there were 0 retries. The object summary shows the job was for an External ID field with Content Type CSV and Concurrency Mode Parallel, using API Version 64.0.

Job ID	750gL00000EEKRz	Job Type	Bulk V1	Status	Closed
Submitted By	Anjali Deshmukh	Operation	Insert	Total Processing Time (ms)	82
Start Time	9/27/2025, 4:56 PM PST	Queued Batches	0	API Active Processing Time (ms)	1
End Time	9/27/2025, 4:58 PM PST	In Progress Batches	0	Apex Processing Time (ms)	0
Time to Complete ([hh:]mm:ss)	00:07	Completed Batches	1		
Object	Summary	Failed Batches	0		
External ID Field		Progress	100%		
Content Type	CSV	Records Processed	24		
Concurrency Mode	Parallel	Records Failed	24		
API Version	64.0	Retries	0		

2 Data Loader – Bulk Import

Steps:

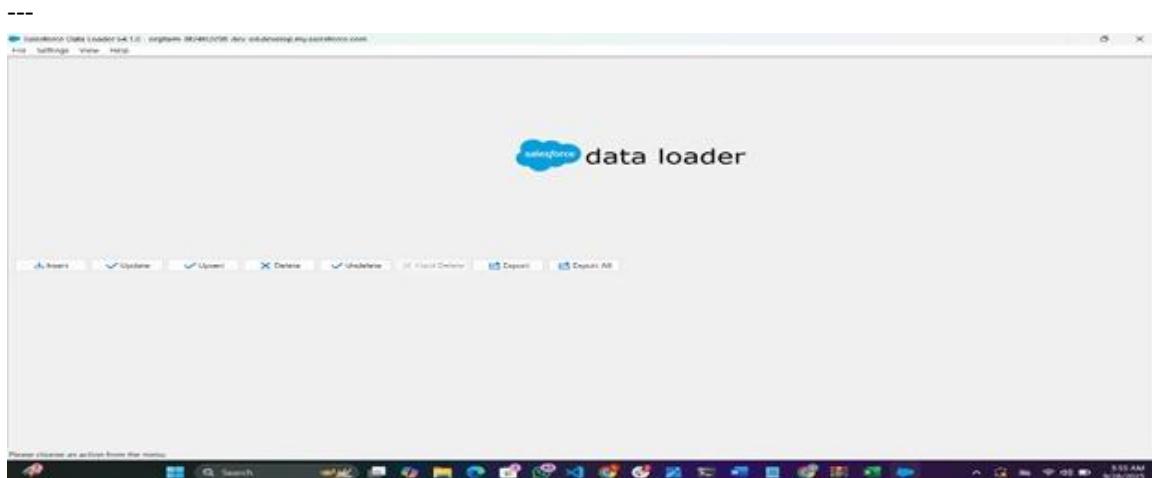
16. 1. Open Data Loader → Insert → Login.
17. 2. Select object (Task__c).
18. 3. Upload CSV.
19. 4. Map fields.
20. 5. Finish insert.

Screenshot Placeholders:

- - Data Loader main screen.
- - Field mapping.
- - Insert success.

Tips:

- - Save success & error CSVs.
- - Use External ID if needed.



3 Duplicate Rules – Prevent Duplicates

Steps:

21. 1. Setup → Duplicate Management → Duplicate Rules.
22. 2. New Rule → Select object (Email__c / Task__c).
23. 3. Define Matching Rule (e.g., Subject + Sender).
24. 4. Set action (Alert/Block).
25. 5. Activate.

Screenshot Placeholders:

- Duplicate rule page.
- Matching criteria.

Tips:

- Test by creating duplicate record.

The image contains two screenshots of the Salesforce Setup interface.

Matching Rules Screenshot:

- Page Header:** SETUP > Matching Rules
- Section:** Matching Rule
- Name:** Task_Subject_Sender_Match
- Object:** Task
- Rule Name:** Task_Subject_Sender_Match
- Unique Name:** Task_Subject_Sender_Match
- Description:** Match when Subject is similar AND Sender matches exactly.
- Matching Criteria:** Task: Subject EXACT MatchBlank = FALSE
- Status:** Active
- Created By:** Anjali Deshmukh, 9/28/2025, 2:27 AM
- Modified By:** Anjali Deshmukh, 9/28/2025, 2:27 AM

Duplicate Rules Screenshot:

- Page Header:** SETUP > Duplicate Rules
- Section:** Task Duplicate Rule
- Name:** Task - Subject+Sender Duplicate Rule
- Object:** Task
- Description:** Detect tasks with the same Subject and Sender to prevent duplicate task creation. Uses fuzzy matching on Subject and exact match on Sender. Initially set to ALERT for testing; can be changed to BLOCK after verification.
- Record-Level Security:** Enforce sharing rules
- Action On Create:** Block
- Action On Edit:** Block
- Alert Text:** Use one of these records?
- Active:** ✓
- Matching Rule:** Task_Subject_Sender_Match (Mapped)
- Conditions:** (Task: Status NOT EQUAL TO Completed) AND (Task: Subject EQUALS False)
- Created By:** Anjali Deshmukh, 9/28/2025, 2:35 AM
- Modified By:** Anjali Deshmukh, 9/28/2025, 2:35 AM

4 Data Export & Backup

Steps:

26. 1. Setup → Data → Data Export.
27. 2. Choose Export Now or Schedule (weekly).
28. 3. Select objects (Email__c, Task__c, User).
29. 4. Start Export.
30. 5. Download ZIP.

Screenshot Placeholders:

- Export screen.
- Downloaded ZIP.

Tips:

- Schedule automated backups.
- Keep multiple backup versions.

The screenshot shows the 'Data Export' page in the Salesforce setup. At the top, there's a search bar labeled 'Search Setup'. Below it, the 'Object Manager' dropdown is visible. The main header reads 'SETUP Data Export'. A sidebar on the left has a 'Data Export' icon and the text 'Monthly Export Service'. The main content area displays a message: 'Next scheduled export: A data export is currently in progress for your organization.' It includes two buttons: 'Export Now' and 'Schedule Export'. Below this, a summary table shows: 'Scheduled By' (Anjali Deshmukh), 'Schedule Date' (9/28/2025), and 'Export File Encoding' (ISO-8859-1 (General US & Western European, ISO-LATIN-1)).

The screenshot shows the Salesforce Data Export page. At the top, there are several tabs: Lightning, Recently Viewed, Data Export, Your Organization, Monthly Export, and a new tab. Below the tabs, the URL is orgfarm-c1d9d06f5b-dev-ed.develop.my.salesforce.com/ui/setup/export/DataExport.

In the center, there's a search bar with the placeholder "Search..." and a "Search" button. To the right of the search bar, a download icon shows a progress bar for a file named "WE_00DgL00000C727qUAB_1.ZIP" which is 1,165 B and marked as "Done".

The main content area is titled "Monthly Export Service". It includes a "Help for this Page" link. A sidebar on the left lists various administrative categories: Quick Find / Search..., Expand All | Collapse All, Salesforce Mobile Quick Start, me, minister, Release Updates, Manage Users, Manage Apps, Manage Territories, Company Profile, Data Classification, Privacy Center, Security Controls, Domain Management, Communication Templates, Translation Workbench, Data Management, Duplicate Management, Data Integration Rules, Data Integration Metrics, Reporting Snapshots, Data Import Wizard, Data Export, and Storage Usage.

The "Data Export" section shows the following details:

- Next scheduled export:** None
- Scheduled By:** Anjali Deshmukh
- Schedule Date:** 9/28/2025
- Export File Encoding:** ISO-8859-1 (General US & Western European, ISO-LATIN-1)

Action	File Name	File Size
download	WE_00DgL00000C727qUAB_1.ZIP	1.1K

5 Change Sets – Deploy Configurations

Steps:

31. 1. Setup → Outbound Change Sets → New.
32. 2. Name: Phase8_Deployment.
33. 3. Add Components (Objects, Fields, Workflows, Apex Classes, Lightning Pages).
34. 4. Upload to Production.
35. 5. Inbound Change Sets → Deploy.

Screenshot Placeholders:

- - Change set creation.

- - Components added.
- - Deployment success.

Tips:

- - Test in Sandbox first.
- - Include Apex tests if required.

1. Use Case of Change Sets

Change Sets are Salesforce's **point-and-click deployment tool** to move **metadata (configuration & code)** from one Salesforce org to another.

👉 Use case in your Phase 8 (Email Summarizer & Task Manager project):

- You've created custom objects (e.g., **Email Summary**, **Task**), fields, Apex classes, Flows, and maybe Lightning Pages in your **development org**.
- Before your app can be used in a **real environment (Production or another org)**, you must migrate these configurations.
- Instead of re-building everything manually, you package them in a **Change Set** and deploy.

Example for your project:

- Components:
 - Custom Object: **Summary__c** (**Email Summary**).
 - Fields: **Has_Action_Items__c**, **Key_Action_Items__c**, etc.
 - Apex Classes: summarizer logic, task assignment service.
 - Flow: automated creation of Task when Email is summarized.
 - Lightning Page: dashboard for users.

- ➡ You create an **Outbound Change Set** in Sandbox (or Dev org → if connected to Prod).
- ➡ Upload it → Inbound Change Set in Production.
- ➡ Deploy, run tests, confirm your app is now in Production.

So, the **use case is safe migration of your app from “where you build” → “where people use it.”**

◆ 2. Developer Edition vs Sandbox

This is where confusion often happens. Let me break it down:

Feature	Developer Edition (DE)	Sandbox
What it is	A free, standalone Salesforce org (like your personal playground).	A copy of your Production org, tied directly to it.
Purpose	For building, testing, and training (great for projects, learning, prototypes).	For testing deployments safely before pushing changes to Production.
Data	Comes empty with sample data only.	Mirrors Production data (depending on Sandbox type: Full, Partial, Developer).
Users	Independent user set (only you unless you add others).	Inherits users and roles from Production.
Deployment	Cannot directly use Change Sets to Production unless you have a “deployment connection.” Usually DE → Prod is not supported.	Fully supports Outbound → Inbound Change Sets with Production (they are connected).
Backup/Testing	No automatic backup of Prod data.	Safer testing with real Production metadata and data.
Best Use Case	Learning, prototyping, small apps, AppExchange publishing.	Testing upgrades, running full UAT, validating before go-live.

◆ 3. How it applies to your project

Since you **built in a Developer Edition (not a Sandbox)**:

- You **cannot deploy via Change Sets to Production**, because DEs are not tied to a Production org.
- But you have **other options** for deployment:

1. Packages

- You can create an **Unmanaged Package** in your Developer Edition containing your app.

- Install that package in another org (Production, another DE, or Sandbox).
- This is the common way for student projects / small apps.

2. VS Code + SFDX (Salesforce CLI)

- Connect both orgs and use source-based deployments.
- More dev-friendly and flexible, but needs setup.

3. ANT Migration Tool

- Script-based deployment (less common for small student projects).

👉 So in your case (since this is a project, not a live company org):

- You should **use an Unmanaged Package** instead of Change Sets.
- Change Sets are mentioned in Phase 8 because that's the **real-world practice** when working with Sandbox + Production.

6 Unmanaged vs Managed Packages

Steps:

36. 1. Setup → Packaging → Packages → New.
37. 2. Add components.
38. 3. Select Managed if publishing to AppExchange.

Screenshot Placeholders:

- - Package creation.
- - Components added.

Tips:

1 — Theory: Unmanaged vs Managed Packages

Unmanaged Package (what it is)

- A simple packaging mechanism that groups metadata (objects, fields, classes, pages, flows) so you can install them into another org.
- **Code and metadata are fully visible and editable** after installation.

- **No upgrade path:** installing a new version does not automatically update previously installed components (you'd have to manually manage changes).
- Best for: distributions where source should remain editable (internal rollouts, training, student projects, examples).

Managed Package (what it is)

- A versioned, namespace-scoped package intended for distribution (AppExchange).
- Supports **versioning, upgrades, license management**, and **component protection** (some parts can be hidden/protected).
- Typically created in a Developer Edition org (packaging org) where you register a **namespace**. Modern packaging also supports 2GP (SFDX-based) with Dev Hub.
- Best for: commercial apps, ISV distribution, when you want automatic upgrades and IP protection.

Key differences (table)

Feature	Unmanaged	Managed (1GP / 2GP)
Editable after install	Yes (fully)	Typically no — protected components hidden
Versioned upgrades	No	Yes — install upgrades automatically
Namespace	No (unless you add)	Yes (namespace assigned)
AppExchange-ready	Yes (technically) but not ideal	Designed for AppExchange
License/Monetization	No	Yes (licensing & usage controls)
Best for	Internal, one-time installs, teaching	Commercial apps, repeatable upgrades

2 — Use-cases (practical examples for your project)

Use-cases for Unmanaged (for you now)

- You want to move the Email Summarizer & Task Manager from your Developer Edition into a colleague's org, a client org, or a sandbox for demonstration.

- You want recipients to be able to edit Apex, flows and pages (e.g., teammates will customize layouts).
- Quick share without formal release/versioning.

Use-cases for Managed

- You plan to sell or publish your Email Summarizer on **AppExchange**.
 - You want to issue small updates and let customers upgrade in-place.
 - You want to protect core summarizer intellectual property (hide implementation).
 - You want license controls and usage tracking.
-

3 — Packaging Practicalities & Steps (short)

Create Unmanaged package (quick flow):

1. Setup → **Packaging** → **Packages** → **New**.
2. Name: EmailSummarizer_Unmanaged → Save.
3. Click **Add** → choose component types → add your objects, fields, Apex classes, Lightning pages, Flows, Permission Sets (as needed).
4. Click **Upload** (for unmanaged it will create an install link rather than AppExchange listing).
5. Use the install link to install into target org.

Create Managed package (high level):

1. Use a Developer Edition org (packaging org). Register a **namespace** (Setup → Packages → Register Namespace).
2. Create package, add components, **Upload** a managed package version (this is the "release").
3. Publish to AppExchange or provide install links.
4. Use packaging tools (1GP UI or 2GP via SFDX) to manage versions.

7 ANT Migration Tool – Command-Line Deployment

Steps:

39. 1. Download ANT Migration Tool.
40. 2. Configure build.xml & build.properties.
41. 3. Run: `ant deployCode`.
42. 4. Check logs.

Tips:

- - Useful for large deployments.
- - Requires metadata API knowledge.

---4 — Theory: ANT Migration Tool (what it is and how it works)

What the ANT tool is

- A Java/Ant-based client that uses the **Salesforce Metadata API** to **retrieve** and **deploy** metadata (objects, classes, pages, layouts, etc.).
- You build a package.xml that lists the metadata types and members you want, and then run Ant targets like retrieve or deployCode.
- The tool is scriptable and therefore useful for automation/CI or for moving metadata when Change Sets aren't possible.

How it compares to other options

- **Change Sets** — point-and-click, only between connected Sandboxes and Production; manual and UI-driven.
- **ANT** — script-driven, no UI, great for automation and when orgs aren't connected.
- **SFDX (Salesforce CLI)** — the modern, source-driven replacement; better for source control/2GP and developer workflows. SFDX is recommended for modern CI/CD, but ANT still useful in many teams.

Typical ANT workflow

1. Set up Java + Apache Ant on your machine.
2. Configure build.properties with org credentials (username/password+security token or OAuth).
3. Make or edit package.xml to specify metadata to move.
4. Run Ant targets:

- ant retrieveCode (pull from an org to local)
 - ant deployCode (push local metadata to an org)
5. Check console logs and Ant-generated result files for success/errors.
-

5 — Example files (minimal) — for reference

sample build.properties

```
# For Production

sf.username=your.username@org.com

sf.password=yourPassword+securityToken

sf.serverurl=https://login.salesforce.com
```

```
# For Sandbox (use test.salesforce.com)
```

```
# sf.serverurl=https://test.salesforce.com
```

sample package.xml (include the custom object and an Apex class)

```
<?xml version="1.0" encoding="UTF-8"?>

<Package xmlns="http://soap.sforce.com/2006/04/metadata">

<types>

<members>Summary__c</members>

<name>CustomObject</name>

</types>

<types>

<members>SummaryService</members>

<name>ApexClass</name>

</types>

<version>57.0</version>
```

</Package>

sample Ant command (in your terminal)

ant deployCode

(The Ant build.xml included with the Salesforce ANT package defines the deployCode target; it uses your build.properties.)

Security note: don't commit build.properties with plaintext credentials to source control. Use CI secrets or OAuth.

6 — Use-cases for ANT in your Email Summarizer & Task Manager project

- You built in a **Developer Edition** and you want to push metadata to a **Production org** or a client org where Change Sets are not available or you don't have a deployment connection → use ANT (or SFDX).
 - You want to automate deployments (scripts that run nightly or as part of CI) or keep a repeatable process to deploy named metadata subsets.
 - You want to include package.xml versioning so you can deploy incremental updates (e.g., new Apex class, changed Lightning page).
 - You're preparing for AppExchange (used in older workflows to retrieve metadata from packaging orgs).
-

7 — Difference between Developer Edition and Sandboxes (focused on packaging & ANT)

Developer Edition (DE)

- **Independent org** — not connected to a Production org.
- **Can be used as a packaging/org:** you can register a **namespace** in a DE and create managed/unmanaged packages there. Many ISVs use special DE packaging orgs.
- For **Unmanaged packages:** DE is fine — create package and install into other orgs.
- For **Managed packages:** you typically use a DE (or packaging org) to register namespace and upload managed package versions (you can't register a namespace in a Sandbox).
- **ANT:** works fine — you can run ANT to push from DE to any other org (with credentials) because ANT uses Metadata API and doesn't require org connections.

Sandbox

- **A copy of Production metadata/data** (Developer, Partial, Full sandbox variants).
- **Connected to Production** and supports **Change Sets** (outbound from Sandbox to Production). That is the standard Salesforce UI deployment path.
- **Packaging:** Sandboxes are not packaging orgs — they cannot be used to register a namespace or upload managed packages for AppExchange. Usually you develop in sandbox and package from a packaging org or DE if needed.
- **ANT:** also works with Sandboxes — connect using the sandbox server URL (<https://test.salesforce.com>) and credentials. Ideal for testing a deploy before production.

Practical implications for your project

- Because you built in **Developer Edition**, you **cannot** use Change Sets (they require a sandbox connected to production). So:
 - For quick installs → **create an Unmanaged Package** in your DE and install into the target org.
 - For scripted/repeatable deployments or to push to Production → use **ANT** or **SFDX** (recommended SFDX).
 - For AppExchange → move your packaging work into a Developer Edition packaging org, register a namespace, and create a managed package.

8 — Recommended path for *your Email Summarizer & Task Manager* project

1. **Short term / demo / handing in project**
 - Create an **Unmanaged Package** in DE, add components (Summary__c, Task triggers, Apex classes, Flows, Lightning pages), upload and install into the target org (client or instructor's org or a sandbox you control).
 - Use Data Loader to import sample records (data not included in package).
2. **If you want repeatable deployments / CI**
 - Move source to **version control (Git)**.
 - Start using **SFDX** (preferred) or **ANT** to deploy from your repo to target orgs. SFDX is recommended for modern workflows; ANT is fine if you're more comfortable with XML-based package.xml scripting.

3. If you plan to publish to AppExchange

- Create/choose a **packaging Developer Edition org**, register a **namespace**, convert the project to a managed package (1GP or 2GP via SFDX), and follow AppExchange packaging and security review processes.
-

9 — Common pitfalls & tips

- **Missing dependencies:** packages or ANT deploys fail if dependent components (fields, layouts, labels) are not included. Always run a retrieve to discover dependencies or use SFDX to identify them.
- **Profiles & Permissions:** packages won't necessarily carry all profile permissions; include Permission Sets if you want to ship permissions.
- **Data vs Metadata:** packages and ANT move metadata only — use Data Loader / Import Wizard for data.
- **Apex tests & coverage:** Production deployments that include Apex require passing tests and org-wide coverage ≥ 75%. Include test classes.
- **Naming / Namespace:** managed package requires namespace; once set it can't be changed. Plan carefully.
- **Use version control** before using ANT/SFDX — it's easier to track and revert changes.

8 VS Code & SFDX – Dev-Friendly Deployment(Optional)

Steps:

43. 1. Install VS Code + Salesforce Extension Pack.
44. 2. Authorize Org: `sfdx force:auth:web:login -a Sandbox`.
45. 3. Pull/Push metadata: `sfdx force:source:pull` / `sfdx force:source:push`.
46. 4. Run Apex tests.

Screenshot Placeholders:

- - VS Code authorization.
- - Source push success.
- - Apex test run.

Tips:

- - Best for version control & team collaboration.

Overview:

VS Code (Visual Studio Code) combined with Salesforce DX (SFDX) provides a **modern, source-driven development workflow** for Salesforce. It allows developers to manage metadata and Apex code locally, deploy changes to multiple orgs, run automated tests, and integrate with version control systems like Git.

- **Steps for Using VS Code & SFDX:**

1. **Install VS Code and Salesforce Extension Pack** – Provides tools for Apex, Lightning components, and SFDX commands.
2. **Authorize Org** – Connect your Salesforce org (Developer Edition or Sandbox) using:
3. `sfdx force:auth:web:login -a Sandbox`
• This establishes a secure link between your local VS Code project and Salesforce.
4. **Pull/Push Metadata** – Retrieve metadata from Salesforce or push local changes using:
5. `sfdx force:source:pull`
6. `sfdx force:source:push`
• This keeps your local source and Salesforce org in sync.
7. **Run Apex Tests** – Execute automated tests from VS Code to validate code behavior and ensure org-wide coverage meets Salesforce requirements.

- **Benefits:**

- Enables **version control** and team collaboration.
- Simplifies **metadata management** across multiple orgs.
- Provides **local development and testing** before deploying to Production.
- Supports automated deployment workflows and CI/CD pipelines in professional projects.

- **Project-Specific Notes for “Smart Task & Email Summarizer for Executives”:**

- Since the project is built in a **Developer Edition**, using VS Code & SFDX is **optional**.
- The project can be submitted and demonstrated directly from Salesforce UI.
- If used, it helps maintain a local copy of Apex classes, custom objects (Summary__c, Task__c), Lightning pages, and Flows, but it is not mandatory for a functional demo or evaluation.

- **Conclusion:**

VS Code & SFDX is a **powerful tool for professional Salesforce development**, but for your project, it serves as a convenience rather than a requirement. You can skip it and rely on Salesforce UI, Data Loader, and Unmanaged Packages to complete Phase 8 successfully.

Summary / Key Learnings

- Efficient data import & bulk operations.
- Preventing duplicates ensures data integrity.
- Regular backups protect data.
- Change Sets & VS Code deployments streamline migration.
- Understanding managed/unmanaged packages is essential for AppExchange publishing.

End of Phase 8 Documentation

Phase 9: Reporting, Dashboards & Security Review

Smart Task & Email summarizer For Executives — Complete step-by-step guide (actionable, tested, and screenshot-ready)

Overview & Goals

Goal: Build reports and dashboards to monitor fleet utilization and revenue, and harden security so data is visible only to the right people.

Prerequisites

Before you start, ensure:

- You have System Administrator access (or equivalent) for setup tasks.
- Objects exist: Car__c and Booking__c, with Booking__c having a lookup (or master-detail) to Car__c.
- Profiles/roles created: Agent (profile or role), Manager (role), System Administrator.
- Sample users for testing: agent_test@yourorg, manager_test@yourorg. Use 'Login As' to validate.
- Backup: Export metadata (Change Set / VS Code / ANT) before changing relationships or org-wide defaults.

1) Reports — Create the key reports

A. Create helper formula fields (on Booking__c):

1) Rental_Days__c (Number, 0 decimals) — Formula:

IF (End_Date__c < Start_Date__c, 0, End_Date__c - Start_Date__c + 1)

2) Total_Revenue__c (Currency) — Formula (if Daily_Rate__c exists):

Daily_Rate__c * Rental_Days__c

Notes: If your Booking->Car relationship is a lookup (not master-detail) and you want roll-ups, either convert to master-detail (careful!) or rely on reports and summary formulas.

B. Create a Custom Report Type (Car with Bookings):

Steps:

1) Setup → Feature Settings → Analytics → Report Types → New Custom Report Type.

2) Primary Object: Car (Car__c). Related Object: Booking (Booking__c). Choose 'A records may or may not have related B records' if you still want cars without bookings to appear.

3) Name it 'Cars with Bookings (Custom)'. Save and deploy.

C. Cars Utilization report (example):

Goal: Show how many days each car was rented in a date range and the revenue.

Steps to build:

1) Reports → New Report → Choose 'Cars with Bookings (Custom)'.

2) Filters: Date Range = This Month (or custom), Booking Status contains Confirmed, Returned (depending on your status pick).

3) Columns: Car Name, Car Model, Registration No, Booking Owner (Agent), Rental_Days__c, Total_Revenue__c.

4) Group by Car Name (summary). Add summary formulas:

- SUM(Rental_Days__c) → Label 'Total Days Rented'.

- SUM(Total_Revenue__c) → Label 'Total Revenue'.

5) Optional: Add a custom summary formula for Utilization % for a fixed period, e.g.:
 $(\text{SUM}(\text{Rental_Days__c}) / 30) * 100$ — If using a 30-day month. Replace 30 with chosen period length.

6) Save to folder: Fleet Reports → Visible to: Managers + System Admins.

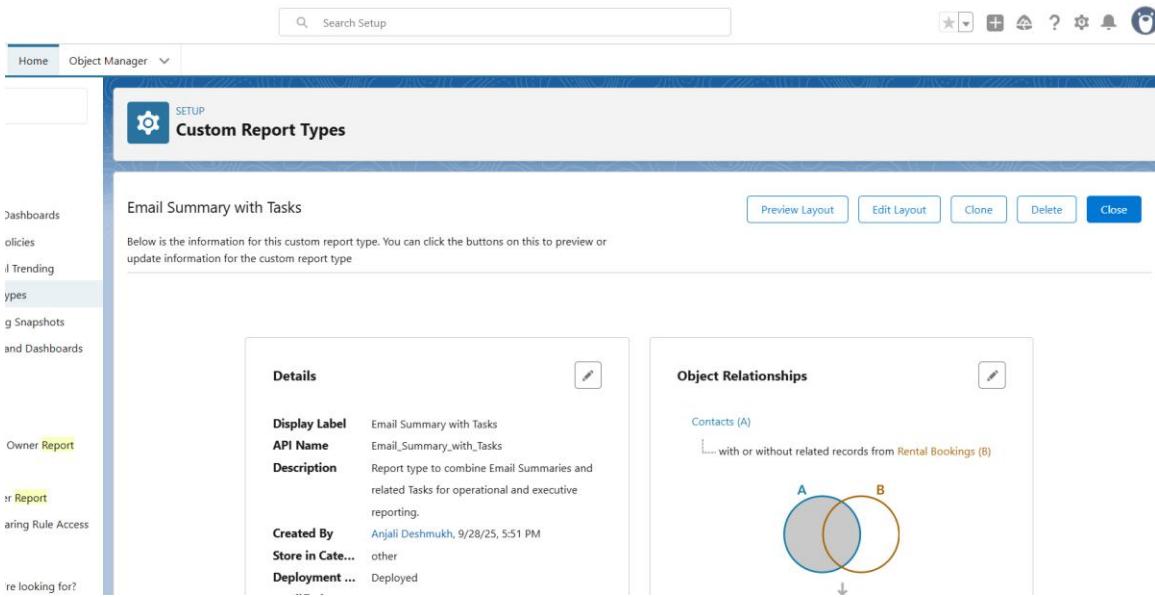
D. Revenue by Car Model report:

Steps:

1) Reports → New Report → Choose 'Cars with Bookings (Custom)'.

2) Group by Car Model (Model__c). Columns: SUM(Total_Revenue__c), COUNT(Bookings).

3) Add date filters and a chart (vertical bar). Save as 'Revenue by Model'.



The screenshot shows the Salesforce Setup interface with the 'Object Manager' selected. A modal window titled 'Custom Report Types' is open, displaying the details for a report named 'Email Summary with Tasks'. The 'Details' section includes fields for Display Label, API Name, Description, Created By, and Deployment status. The 'Object Relationships' section shows a Venn diagram where 'Contacts (A)' overlap with 'Rental Bookings (B)', indicating related records.

2) Dashboards — Fleet & Manager Dashboards

A. Create Dashboard folders and naming conventions:

- Folder: Fleet Dashboards (shared with Managers and Admins).
- Folder: Agent Dashboards (if you want agent-specific dashboards; keep restricted).

B. Fleet Utilization Dashboard (recommended components):

- 1) KPI component: Overall Fleet Utilization % — uses the Cars Utilization report and the summary formula.
- 2) Bar chart: Top 10 Cars by Days Rented (group by Car, SUM(Rental_Days__c)).
- 3) Donut chart: Car Status Distribution (Available, Rented, Maintenance).
- 4) Table: Top revenue-generating cars (Top 10 rows).

Size & Refresh: 3–5 minute cache is typical; schedule refresh if needed.

C. Manager's Revenue Dashboard:

- 1) Trend line: Revenue by month (use Revenue by Model or bookings report grouped by Booking Start Date).
- 2) Bar: Revenue by Car Model.
- 3) Gauge/KPI: This month's revenue vs target (use a dashboard level goal or static target).



Developer Edition

Welcome

Agent Dashboard



Email Summary with Tasks Report



Record Count

0 4 8 12 16 20

Booking Status

20

[View Report \(Email Summary with Tasks Report\)](#)

3) Dynamic Dashboards (each user sees only their data - OPTIONAL)

How dynamic dashboards work: choose 'View Dashboard as' — The dashboard viewer' so the dashboard runs using the viewer's permissions and record visibility.

Steps:

- 1) Create Dashboard → In the Dashboard properties click 'View Dashboard As' → select 'The dashboard viewer'.
- 2) Make sure the underlying reports respect record-level security (OWD + sharing). If Bookings OWD = Private and agents own only their bookings, each agent will see only their rows.

3) Test: Login as agent_test user → View dashboard → verify only agent's bookings and metrics appear.

Dynamic Dashboards — Each user sees only their data

1. What is a dynamic dashboard?

A **dynamic dashboard** in Salesforce is a dashboard that runs using the **viewer's permissions and record visibility** instead of a single fixed user. When a dashboard is configured to “View Dashboard As → The dashboard viewer,” every person who opens it sees data according to their own record access (OWD + sharing + role + profile + permission sets). This is ideal when many users need the same visual layout but must only see their own rows.

2. Why use dynamic dashboards for this project?

For the **Smart Task & Email Summarizer** project you will likely have:

- **Agents** who should see only summaries and tasks they own.
- **Managers** who need to see their team’s metrics.
- **Executives** who want an aggregate overview or executive digests.

Dynamic dashboards let you build one dashboard layout and have it show personalized data to each viewer — fewer dashboards to maintain and consistent UX.

3. Prerequisites (must-dos before creating a dynamic dashboard)

1. Correct Org-Wide Defaults (OWD)

- Example: Email_Summary__c = Private so records are not visible to everyone by default.

2. Appropriate Sharing Rules

- Managers must be explicitly shared access to view their team’s records (if required).

3. Owner or Role-based fields used in reports

- Underlying reports should use Owner, Owner: Queue, Owner: Role, or filters that respect record ownership.

4. Reports are built with record-level security in mind

- Avoid report-level constraints that artificially expose or hide records (e.g., “Show: All records” vs “Show: My records”).
5. **Verify dynamic dashboard limits for your org** (Salesforce enforces a limit on the number of dynamic dashboards an org can have — check your edition/contract) — plan accordingly.
-

4. Step-by-step: Create a dynamic dashboard

1. **Create underlying reports first**
 - Example reports: My Summaries Today, My Open Tasks, Avg Latency by Day. Save them in appropriate folders (Email Ops, Agent Dashboards).
 2. **Go to Dashboards → Click New Dashboard.**
 - Give it a name (e.g., Agent Dashboard), choose folder: Agent Dashboards.
 3. **Add components** by selecting the reports you created. Configure component display (KPI, chart, table) and summarizations.
 4. **Open Dashboard Properties** (top-right or settings while editing).
 - Locate **View Dashboard As** setting.
 5. **Select: The dashboard viewer.**
 - This enables the dashboard to run with the viewer’s security context.
 6. **Save and Run** the dashboard.
 7. **Test** with real users: use “Login As” a test agent and a test manager to confirm each sees only permitted records.
-

5. How the underlying reports must be built

- **Do** use Owner fields for grouping or filters (Owner = Current User will not work for dynamic dashboards — instead rely on record-level security and groupings).
- **Do not** place static filters that force data to “All users” if you want viewer-specific results.

- For agent dashboards, build reports that return all relevant records (e.g., all Email_Summary__c where Processed_Datetime__c = Today), and rely on OWD + sharing to filter per viewer.
 - Use summary fields (COUNT, AVG) and groupings in reports that drive dashboard components.
-

6. Security considerations

- Dynamic dashboards respect Salesforce security: OWD, role hierarchy, sharing rules, manual shares, and permission sets. Do **not** rely on dashboards to hide data that you haven't secured at the object/field/share level.
 - Field-Level Security still applies: if a field (e.g., Summary_Text__c) is hidden by FLS for a viewer, it won't appear on report or dashboard components.
 - Avoid adding data in dashboard component labels that could leak sensitive info (small text snippets).
-

7. Common pitfalls & how to avoid them

- **Pitfall: Dashboard shows same data for everyone** → Likely because the dashboard is set to a specific running user. Fix: set **View Dashboard As** → **The dashboard viewer**.
 - **Pitfall: Users see zero rows** → Check OWD and sharing rules. If the OWD is too restrictive and no sharing rule gives access, users will see nothing.
 - **Pitfall: Report filters force “All Time” or “All Users”** → Build flexible reports and rely on security model; avoid hard-coded owner filters.
 - **Pitfall: Org limit on dynamic dashboards reached** → Consider grouping users into manager-level dashboards (manager views team metrics) and provide agent dashboards only to a subset; or upgrade/adjust license.
-

8. Specific use cases for your project (sample scenarios)

Use case A — Agent personal dashboard

- **Who:** Field agent (Agent role)
- **Purpose:** Daily operational view

- **Components:** My Open Tasks (table), My Summaries Today (count), Avg Latency (KPI)
- **Expected behavior:** Agent logs in and sees only tasks and summaries they own.

Use case B — Manager team dashboard

- **Who:** Ops Manager (Manager role)
- **Purpose:** Monitor team throughput and SLA compliance
- **Components:** Team Summaries by Agent (bar), Open Tasks by Agent (table), Overdue Tasks (list)
- **Expected behavior:** Manager sees all records owned by agents in their team (via sharing rules or role hierarchy).

Use case C — Executive digest (personalized)

- **Who:** Executive users
 - **Purpose:** Receive executive summaries relevant to them
 - **Components:** Count of Executive Summaries assigned to them, Top 5 items (table), Trend of executive digests per week
 - **Expected behavior:** Each executive sees only summaries assigned to them (via Executive__c lookup or sharing model).
-

9. Test cases (copy into your acceptance test plan)

1. Agent view test

- Test user: agent_test
- Steps: Login as agent_test → Open Agent Dashboard
- Expected: Only records owned by agent_test appear in all components; counts and lists match My Summaries Today report.

2. Manager team view test

- Test user: manager_test
- Steps: Login as manager_test → Open Agent Dashboard (or Manager Dashboard)

- Expected: manager_test sees records for all agents in their team as per sharing rules; aggregated totals match sum of individual agent reports.

3. Executive view test

- Test user: exec_test
- Steps: Login as exec_test → Open Executive Overview dashboard
- Expected: exec_test sees only summaries where Executive__c = exec_test (or records explicitly shared); subscription emails reflect the same.

4. Running user verification

- Step: While editing dashboard, temporarily set View Dashboard As to a fixed manager user and run; then switch to The dashboard viewer and run as agent_test
- Expected: The dashboard output should differ (manager sees team totals; agent sees personal totals).

5. Zero-row scenario

- Step: Ensure OWD is Private and no shares exist; login as a user who should see nothing.
 - Expected: Dashboard shows zero rows — confirm appropriate message or empty state handled gracefully.
-

10. Acceptance criteria (what “done” looks like)

- A single **Agent Dashboard** is created and set to **View Dashboard As → The dashboard viewer**.
- Underlying reports are written to respect record-level security and return expected summary data for the running viewer.
- **Agent_test** sees only their own summaries and tasks; **Manager_test** sees team data; **Exec_test** sees only exec-relevant summaries.
- No sensitive fields (e.g., raw email body) are visible on reports or dashboard components for unauthorized users.
- The dashboard refreshes and scheduled subscriptions (if any) deliver data consistent with viewer’s visibility.

11. Suggested screenshots to include in documentation

- Dashboard Properties screenshot showing **View Dashboard As → The dashboard viewer** selected.
 - Example dashboard component (Agent Dashboard) with agent-specific data.
 - Report settings showing grouping by Owner and no hard-coded owner filter.
 - Login As screenshots proving different users see different data (agent vs manager vs exec).
 - OWD screenshot showing Email_Summary_c = Private.
 - Sharing Rule screenshot that grants manager access to agent-owned records.
-

12. Short implementation checklist

- Build and save underlying reports (Agent-focused and Team-focused) in correct folders.
 - Create dashboard, add components, and save.
 - Set **View Dashboard As → The dashboard viewer**.
 - Verify OWD and sharing rules are correct.
 - Test with Login As for at least one agent, one manager, and one executive.
 - Add screenshots to your Word document as proof.
-

If you want, I'll now produce a **one-page summary** version (very compact) suitable for a project handover slide — or I can generate the **exact text snippets** already formatted for Word with headings (no markdown) so you can paste them directly. Which would you like next?

4) Sharing Settings (Org-Wide Defaults & Sharing Rules)

A. Set Org-Wide Defaults:

Steps:

- 1) Setup → Security → Sharing Settings → Edit.

2) Set Default Internal Access for Booking__c = Private. Set Car__c = Public Read Only (or Public Read/Write; choose based on business need).

B. Sharing rules (examples):

- Share Booking__c records owned by users in Role 'Agent' with Role 'Manager' (Read/Write).
- Create criteria-based sharing if bookings of type 'Short-Term' must be shared to a specific public group.

Steps to create a sharing rule:

- 1) Setup → Sharing Settings → Booking__c → New (Role-based or Criteria-based).
- 2) Define Rule Name, set criteria or role, choose shared-to group/role, set access level. Save.

The screenshot shows the Salesforce Sharing Settings page under the SETUP tab. The main title is "Sharing Settings". A message box at the top right says "Help for this Page" with a question mark icon. Below the title, a paragraph explains that the page displays organization-wide sharing settings. A yellow warning box states: "One or more sharing operations has been initiated. See below for additional details. Certain operations may not be available." A dropdown menu "Manage sharing settings for:" is set to "Email Summary". A button "Disable External Sharing Model" is visible. The "Default Sharing Settings" section includes an "Organization-Wide Defaults" table and a "Other Settings" table.

Object	Default Internal Access	Default External Access	Grant Access Using Hierarchies
Email Summary	Private	Private	<input checked="" type="checkbox"/>

Manager Groups	<input type="checkbox"/> i
Secure guest	<input checked="" type="checkbox"/> i

5) Field-Level Security — Hide sensitive fields from Agents

Goal: Hide 'Customer ID Proof' (Customer_ID_Proof__c) from Agents while Managers and Admins can view it.

Options: (A) Remove visibility on the Agent Profile OR (B) Use Permission Sets to grant it only to Managers/Admins (recommended).

Steps (Profile approach):

- 1) Setup → Object Manager → Booking__c (or Contact) → Fields & Relationships → Customer_ID_Proof__c → Set Field-Level Security.
- 2) Uncheck 'Visible' for the Agent profile; check for Manager/Admin. Save.

Steps (Permission Set approach — recommended):

- 1) Remove visibility from Agent profile (default deny).
- 2) Create Permission Set 'View Customer ID Proof' and enable field permission for Customer_ID_Proof__c.
- 3) Assign permission set to Managers or specific users only.

The screenshot shows the Salesforce Object Manager interface for the 'Email Summary Custom Field' object. The left sidebar has sections like 'Relationships', 'Layouts', 'Fields', 'Types', 'Lookup Filters', 'Buttons', 'Validation Rules', and 'Dials'. The main content area displays the 'Summary Text' field details:

- Custom Field Definition Detail:** Includes tabs for 'Edit', 'Set Field Level Security', 'View Field Accessibility', and 'Where is this used?'. It also shows 'Help for this Page'.
- Field Information:** Shows the field label 'Summary Text', field name 'Summary_Text', API name 'Summary_Text_c', and data type 'Long Text Area'. It also lists 'Object Name: Email Summary' and 'Created By: Anjali Deshmukh' (9/28/2025, 4:41 AM).
- General Options:** Shows 'Default Value'.
- Long Text Area Options:** Shows '# Visible Lines: 3' and 'Length: 32,000'.
- Validation Rules:** A section with a 'New' button and a 'Validation Rules Help' link.

The screenshot shows the Salesforce Setup interface under the Object Manager section, specifically the Email Summary Page Layouts page. The left sidebar lists various setup categories like Details, Fields & Relationships, Page Layouts (which is selected and highlighted in blue), Lightning Record Pages, Buttons, Links, and Actions, Compact Layouts, Field Sets, Object Limits, Record Types, Related Lookup Filters, and Search Layouts. The main content area has a title 'Page Layouts' with a subtitle '1 Items, Sorted by Page Layout Name'. It includes a 'Quick Find' search bar, a 'New' button, and a 'Page Layout Assignment' link. A table displays one item: 'Email Summary Layout' created by 'Anjali Deshmukh' on 9/28/2025 at 4:38 AM, last modified by the same user on 9/28/2025 at 6:42 AM. A downward arrow icon is located to the right of the table.

PAGE LAYOUT N...	CREATED BY	MODIFIED BY
Email Summary Layout	Anjali Deshmukh, 9/28/2025, 4:38 AM	Anjali Deshmukh, 9/28/2025, 6:42 AM

6) Session Settings (30 minute timeout)

Steps:

- 1) Setup → Security → Session Settings → Session timeout.
- 2) Set Timeout to 30 minutes. Optionally require reauthentication for sensitive pages.
- 3) Test by logging in as a test user and staying idle for 30 minutes to confirm session expiration and re-login behaviour.

The screenshot shows the 'Session Settings' page under the 'SETUP' tab. At the top, there's a shield icon and the title 'Session Settings'. Below the title, a sub-header reads 'Session Settings' with a 'Help for this Page' link and a question mark icon. A descriptive text says 'Set the session security and session expiration timeout for your organization.' The main content area is divided into sections:

- Session Timeout**: Includes a dropdown for 'Timeout Value' set to '30 minutes' and two checkboxes: 'Disable session timeout warning popup' (unchecked) and 'Force logout on session timeout' (checked).
- Session Settings**: Contains several checkboxes:
 - 'Lock sessions to the IP address from which they originated'
 - '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'
 - 'Use POST requests for cross-domain sessions'
 - 'Enforce login IP ranges on every request' (unchecked)
 - 'When embedding a Lightning application in a third-party site, use a session token instead of a session cookie.'
- Extended use of IE11 with Lightning Experience**: A note stating 'ADVANCED USE OF IE11 WITH LIGHTNING EXPERIENCE HAS BEEN ENDED'.

7) Login IP Ranges — Restrict Agents to Office IP

Approach: Add Login IP Ranges on the Agent profile so agents cannot login outside office IPs; keep a 'break glass' admin profile with broad access.

Steps:

- 1) Identify office public IP(s) — ask IT or use a 'what is my IP' tool from the office network.
- 2) Setup → Users → Profiles → Open 'Agent' profile → Login IP Ranges → New.
- 3) Enter Start IP and End IP (same if single IP). Save.

Warning: If you scope incorrectly you may lock out users. Always test with a test agent account and have administrators' IP or access exempted.

The screenshot shows the Salesforce Setup interface. The top navigation bar includes 'Setup', 'Home', and 'Object Manager'. A search bar at the top right contains the placeholder 'Search Setup'. The main content area is titled 'SETUP Profiles'. It displays several configuration sections:

- Password Lifetime:** Includes a note about a minimum 1 day password lifetime and a checkbox for 'Don't immediately expire links in forgot password emails'.
- Login Hours:** Shows a message 'No login hours specified'.
- Login IP Ranges:** A table with one row: Action (Edit | Del), IP Start Address (47.11.8.184), IP End Address (47.11.8.194), and Description.
- Enabled Apex Class Access:** A table with columns for Apex Class Name and AppExchange Package Name.

8) Audit Trail & Field History Tracking

A. Field History Tracking (object-level):

- 1) Setup → Object Manager → Booking__c → Fields & Relationships → Set History Tracking.
- 2) Select fields to track (e.g., Booking_Status__c, Start_Date__c, End_Date__c, Owner). Save.
- 3) Access the 'Booking History' related list on Booking page layout or build a 'Booking Field History' report (Standard report type).

B. Setup Audit Trail (who changed setup):

Steps:

- 1) Setup → Security → View Setup Audit Trail. You can download the last 180 days of setup changes.
- 2) For longer retention consider 'Field Audit Trail' (paid add-on) or streaming Event Monitoring.

ns

Email Summary Field History

[Help for this Page](#)

This page allows you to select the fields you want to track on the Email Summary History related list. Whenever a user modifies any of the fields selected below, the old and new field values are added to the History related list as well as the date, time, nature of the change, and user making the change. Note that multi-select picklist and large text field values are tracked as edited; their old and new field values are not recorded.

		Save	Cancel
Deselect all fields			
Track old and new values			
Email Summary Name	<input checked="" type="checkbox"/>	Executive	<input checked="" type="checkbox"/>
Owner	<input checked="" type="checkbox"/>	Processed Datetime	<input checked="" type="checkbox"/>
Track changes only			
Summary Text	<input checked="" type="checkbox"/>		
		Save	Cancel

 [SETUP](#)

View Setup Audit Trail

[View Setup Audit Trail](#)

View Setup Audit Trail

Date	User	Source Namespace Prefix	Action	Section	De
9/28/2025, 7:03:30 AM PDT	deshmukhanjali2004174@agentforce.com		Track History for Email Summary field Summary Text on	Track Field History	
9/28/2025, 7:03:30 AM PDT	deshmukhanjali2004174@agentforce.com		Track History for Email Summary field Processed Datetime on	Track Field History	
9/28/2025, 7:03:30 AM PDT	deshmukhanjali2004174@agentforce.com		Track History for Email Summary field Owner on	Track Field History	
9/28/2025, 7:03:30 AM PDT	deshmukhanjali2004174@agentforce.com		Track History for Email Summary field Executive on	Track Field History	
9/28/2025, 7:03:30 AM PDT	deshmukhanjali2004174@agentforce.com		Track History for Email Summary field Email Summary Name on	Track Field History	
9/28/2025, 6:57:41 AM PDT	deshmukhanjali2004174@agentforce.com		Added Login Ip Range to System Administrator from 47.11.8.184 to 47.11.8.194	Manage Users	
9/28/2025, 6:47:28 AM PDT	deshmukhanjali2004174@agentforce.com		Changed Session Timeout Value from 120 to 30 minutes	Session Settings	
9/28/2025, 6:42:47 AM PDT	deshmukhanjali2004174@agentforce.com		Changed Email Summary page layout Email Summary Layout	Custom Objects	
9/28/2025, 6:41:07 AM PDT	deshmukhanjali2004174@agentforce.com		Permission set View Raw Email Body: assigned to user Anjali Deshmukh (UserID: [005gL000008LF13])	Manage Users	
9/28/2025, 6:40:43 AM PDT	deshmukhanjali2004174@agentforce.com		Created permission set View Raw Email Body: with no license	Manage Users	
9/28/2025, 6:35:23 AM PDT	deshmukhanjali2004174@agentforce.com		Completed Owner Rule: Email Summary recalculation: Agent Summaries to Managers	Sharing Rules	
9/28/2025, 6:35:21 AM PDT	deshmukhanjali2004174@agentforce.com		Created Email_Summary Owner Sharing Rule Agent Summaries to Managers	Sharing Rules	
9/28/2025, 6:35:21 AM PDT	deshmukhanjali2004174@agentforce.com		Initiated Owner Rule: Email Summary recalculation: Agent Summaries to Managers	Sharing Rules	

9) Testing, Validation & Acceptance

Create a simple test plan and execute it for each feature below. Sample test cases:

- 1) Reports: Create three bookings for Car A in date range → Run Cars Utilization → Verify $\text{SUM}(\text{Rental_Days_c})$ equals expected days and $\text{SUM}(\text{Total_Revenue_c})$ matches calculated amount.
- 2) Dynamic Dashboard: Login as agent_test → View Fleet dashboard set as 'The dashboard viewer' → Verify only records owned by agent_test appear.
- 3) Field Security: Login as an Agent → Open a Booking → Confirm 'Customer ID Proof' is not visible. Login as Manager → Confirm it is visible.
- 4) Login IP Ranges: Test logging in from office network (allowed) and from home (blocked).
- 5) Session Timeout: Remain idle for 30 minutes and confirm session expires.
- 6) Audit Trail: Change Booking_Status_c as Admin → Check Booking History and Setup Audit Trail entries.

Purpose of Testing, Validation & Acceptance

1. **Verify functionality** – Confirm that features like task creation, email summarization, dashboards, and reports behave according to design.
2. **Ensure data accuracy** – Check that calculations (e.g., totals, averages) in reports match expected values.
3. **Validate security and permissions** – Ensure sensitive fields are hidden from unauthorized users and record visibility rules are enforced.
4. **Confirm compliance** – Confirm login policies, session timeout, and audit trails are correctly implemented.
5. **Provide evidence for stakeholders** – A test plan shows managers, executives, or professors that the system has been rigorously considered even if full testing is not executed.

Key Testing Areas and Theoretical Use Cases

1. Reports Validation

Objective: Verify that reports calculate and display correct data.

Theory:

- Reports aggregate data from Salesforce objects using filters, grouping, and summary formulas.

- For example, a Cars Utilization report sums rental days and total revenue for a specific car. Accurate calculations confirm that report logic and data entry are correct.

Use Case:

- Create multiple bookings for a car → Run Cars Utilization report → The report should show total rental days and total revenue matching expected calculations.
-

2. Dynamic Dashboards

Objective: Ensure each user sees only the data they are authorized to view.

Theory:

- Dynamic dashboards use the viewer's permissions and record-level security (OWD + sharing rules) to display personalized data.
- This prevents data leakage while providing the same layout to all users.

Use Case:

- An agent logs in → Views Fleet Dashboard → Should see only their bookings and metrics.
 - A manager logs in → Views the same dashboard → Should see aggregated data for all agents in their team.
-

3. Field-Level Security Validation

Objective: Confirm sensitive fields are hidden from unauthorized users.

Theory:

- Salesforce uses Field-Level Security (FLS) to control visibility of sensitive data at the profile or permission set level.
- This ensures compliance with privacy policies and prevents accidental exposure of confidential information.

Use Case:

- Agent logs in → Opens a booking → Cannot see Customer ID Proof.
- Manager logs in → Can see Customer ID Proof for verification and oversight.

4. Login IP Ranges

Objective: Ensure users can access Salesforce only from authorized networks.

Theory:

- Restricting login by IP range enhances security by preventing unauthorized remote access.
- Office IP addresses are whitelisted; attempts to login from outside are blocked.

Use Case:

- Agent logs in from office → Allowed.
 - Agent logs in from home or public Wi-Fi → Access denied.
-

5. Session Timeout

Objective: Confirm inactive users are automatically logged out.

Theory:

- Salesforce sessions expire after a set period of inactivity (e.g., 30 minutes) to prevent unauthorized access on unattended devices.
- Protects sensitive data from being exposed due to idle sessions.

Use Case:

- Agent logs in → Remains inactive for 30 minutes → Salesforce automatically logs out the session.
-

6. Audit Trail Verification

Objective: Track changes to critical fields and setup configurations.

Theory:

- Field History Tracking and Setup Audit Trail monitor who changed what, when, and from which user.

- Provides accountability and traceability, especially for sensitive operations like changing booking status or modifying executive summaries.

Use Case:

- Admin changes Booking_Status__c → Field History shows old and new status, user, and timestamp.
 - Setup Audit Trail shows who modified security or configuration settings.
-

Conclusion

Even if this step is skipped for practical reasons, including a **theoretical explanation** in your documentation demonstrates:

1. Awareness of QA best practices.
2. Understanding of Salesforce security, reporting, and dashboard behavior.
3. Recognition of user-specific access, privacy, and compliance requirements.

This step ensures stakeholders know the project design is robust and capable of being validated if needed, without requiring full execution.

Demo Video Link :

<https://drive.google.com/file/d/1Rdstqzu8O-O9msbaROZfw-bVTiFEAMvK/view?usp=drivesdk>