

Project Title: *ParkVista – Transforming-Urban-Parking-with-Salesforce-CRM*

Industry: Urban Mobility / Smart Parking

Project Type: B2C Salesforce CRM Implementation

Target Users: Parking Lot Managers, Admins, and Vehicle Owners

Problem Statement

In growing cities, parking management is often manual, leading to overbooking, revenue leakage, and poor customer experience. Vehicle owners struggle to find and reserve slots, while lot managers lack automation and real-time insights.

To address this, the organization wants to implement a Salesforce CRM to:

- Digitize parking reservations and renewals
- Prevent overbooking with real-time slot availability validation
- Auto-assign nearest parking lots based on location
- Automate customer confirmations, reminders, and payment integrations
- Provide dashboards for slot utilization, lead conversion, and revenues

Use Cases

Lead Management

- Capture leads from online booking forms
- Auto-assign leads based on parking lot location
- Qualify leads with intent scoring (duration, slot type, location)

Parking Lot & Slot Management

- Maintain inventory of lots and slots with location, pricing, and status
- Auto-update expired reservations to “available” using batch jobs
- Prevent double booking through validation rules

Reservation Scheduling

- Allow customers to reserve slots online
- Send email confirmations and reminders
- Support bulk/corporate booking requests

Payments & Confirmation

- Integrate with payment gateways for reservation fees
- Track payments, refunds, and receipts

Reporting & Dashboards

- Real-time dashboards for slot utilization and revenues
- Lead funnel, conversion rates, and customer insights
- Parking lot performance tracking

Phase 1: Problem Understanding & Industry Analysis

Requirement Gathering

This is the step where you collect the exact needs from different stakeholders.

Example Requirements for ParkVista:

- Users should be able to book/reserve parking slots online.
- Admin should assign leads (parking requests) to available agents.
- System must send SMS/Email confirmations for reservations.
- Agents should track real-time slot availability.
- Managers should have a dashboard to monitor leads, reservations, and revenue.

Stakeholder Analysis

Identify who will use or benefit from the system.

Example Stakeholders:

- **Drivers/End Users:** People booking/reserving parking slots.
- **Parking Attendants/Agents:** Manage slot availability and help customers.
- **Admin/Manager:** Monitor lead funnel, track revenue, optimize slot usage.
- **IT/CRM Team:** Maintain Salesforce configurations.

Business Process Mapping

Current Manual Flow (Problem):

- Driver arrives → Searches manually for parking → Agent notes slot availability → Cash payment → No tracking.

Proposed Salesforce Flow:

- Driver submits lead via web/app → Lead auto-captured in Salesforce → Lead qualified → Converted to reservation → Payment tracked → Dashboard updated.

Industry-Specific Use Case Analysis

Parking industry pain points and opportunities.

- **Pain Points:**
 1. Manual slot management leads to double-booking.
 2. No visibility on peak usage hours.
 3. Poor customer experience (long wait times).
- **Opportunities:**
 1. CRM-based lead-to-reservation flow improves efficiency.
 2. Dashboards give real-time insights into occupancy and revenue.
 3. Automated notifications reduce no-shows and cancellations.

AppExchange Exploration

Check Salesforce AppExchange for existing apps to take inspiration.

- **Parking-Specific / Similar Apps:**
 1. Event/venue booking apps (adaptable for slot reservations).
 2. Resource scheduling apps.
 3. Payment integration apps (Stripe, PayPal connectors).
- **Decision:** Since it's a capstone project, ParkVista will implement **custom objects and flows** but can mention inspiration from these apps.

Phase 2: Org Setup and Configuration

Company Profile Setup -

Set basic org details under **Setup** → **Company Information** → **Edit**

- **Name:** ParkVista Smart Parking Solutions – Dev
- **Time Zone:** (GMT+05:30) Asia/Kolkata
- **Locale:** English (India) → Controls date, time, and number formats
- **Language:** English → Default UI language for all new users

- **Currency:** INR (₹) → Corporate currency for pricing and reservation billing

The screenshot shows the 'Company Information' page in the Salesforce Setup. The organization's name is 'ParkVista Smart Parking Solutions'. The 'Organization Detail' section contains various configuration settings, many of which have checkboxes. Some checked items include 'Primary Contact' (Utkarsh Deshmukh), 'Address' (India), 'Fiscal Year Starts In' (January), 'Newsletter' (checked), 'Admin Newsletter' (checked), and 'Locale Formats' (ICU). Other settings like 'Default Locale' (English (India)), 'Default Language' (English), and 'Default Time Zone' (GMT+05:30 India Standard Time (Asia/Kolkata)) are also listed. The page includes navigation links for User Licenses, Permission Set Licenses, Feature Licenses, and Usage-based Entitlements. At the bottom, it shows the creation and modification details.

Organization Detail	
Organization Name	ParkVista Smart Parking Solutions
Primary Contact	Utkarsh Deshmukh
Division	
Address	India
Fiscal Year Starts In	January
Activate Multiple Currencies	<input type="checkbox"/>
Enable Data Translation	<input type="checkbox"/>
Newsletter	<input checked="" type="checkbox"/>
Admin Newsletter	<input checked="" type="checkbox"/>
Hide Notices About System Maintenance	<input type="checkbox"/>
Hide Notices About System Downtime	<input type="checkbox"/>
Locale Formats	ICU
Phone	(862) 398-6015
Fax	
Default Locale	English (India)
Default Language	English
Default Time Zone	(GMT+05:30) India Standard Time (Asia/Kolkata)
Currency Locale	Hindi (India) - INR
Used Data Space	342 KB (7%) [View]
Used File Space	17 KB (0%) [View]
API Requests, Last 24 Hours	0 (15,000 max)
Streaming API Events, Last 24 Hours	0 (10,000 max)
Restricted Logins, Current Month	0 (0 max)
Salesforce.com Organization ID	00Dg1.00000BW3xY
Organization Edition	Developer Edition
Instance	CAN98

Created By: OrgFarm EPIC, 9/12/2025, 5:39 PM Modified By: Utkarsh Deshmukh, 9/18/2025, 11:44 PM

Business Hours & Holidays

- **Path:** Setup → Business Hours → New
- **Name:** Standard Business Hours (24×7 Parking Availability)
- **Time Zone:** GMT+05:30 Asia/Kolkata (or regional timezone)
- **Working Hours:** 12:00 AM – 11:59 PM, all 7 days of the week (round-the-clock availability)
- **Holidays:** None added, since parking is operational every day without exception
- **Save:** Set this as the default business hours for the org

SETUP

Business Hours

Organization Business Hours

Select the days and hours that your support team is available. These hours, when associated with escalation rules, determine the times at which cases can escalate.

If you enter blank business hours for a day, that means your organization does not operate on that day.

[Help for this Page](#)

Business Hours Detail		Edit	Time Zone	(GMT+05:30) India Standard Time (Asia/Kolkata)
Business Hours Name	Default		Default Business Hours	<input checked="" type="checkbox"/>
Business Hours	Sunday	24 Hours		
	Monday	24 Hours		
	Tuesday	24 Hours		
	Wednesday	24 Hours		
	Thursday	24 Hours		
	Friday	24 Hours		
	Saturday	24 Hours		

Active

Created By [OrgFarm EPIC](#) 9/12/2025, 5:39 PM

Last Modified By [Utkarsh Deshmukh](#) 9/18/2025, 11:55 PM

[Edit](#)

Holidays

[Add/Remove](#)

No records to display

Fiscal Year Setup

- Path:** Setup → Company Settings → Fiscal Year
- Type:** Standard Fiscal Year (no need for custom unless rules are different)
- Configuration:** Set **Starting Month = April** (so it runs April → March, as per India's financial year)
- Save:** Apply settings to define fiscal periods for reports, forecasts, and opportunities.

SETUP

Fiscal Year

Organization Fiscal Year Edit: ParkVista Smart Parking Solutions

To specify the fiscal year type for your organization, choose one of the options below.

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

Standard Fiscal Year [?](#)

Custom Fiscal Year [?](#)

Name: ParkVista Smart Parking Solutions

Fiscal Year Start Month: [▼](#)

Fiscal Year is Based On: The ending month The starting month

[Save](#) [Cancel](#)

User Setup (Profiles, Roles, Permission Sets, Users)

Profiles:

Created three custom profiles by cloning standard ones to define baseline access.

- **Super_Admin_Profile** – cloned from *System Administrator* to provide full access for super admin users.
- **Regional_Manager_Profile** – cloned from *Standard User* for regional-level operations.
- **Lot_Manager_Profile** – cloned from *Standard Platform User* to restrict access to core lot management functions.

The screenshot shows the 'Super_Admin_Profile' setup page. At the top, there's a header with a user icon, 'SETUP', and 'Profiles'. Below the header, the profile name 'Super_Admin_Profile' is displayed. A note says: '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.' Another note for Record Types is present. A long list of permissions follows, including 'Login IP Ranges', 'Enabled Apex Class Access', etc. Below this is a 'Profile Detail' section with fields for Name (Super_Admin_Profile), User License (Salesforce), Description, Created By (Utkarsh Deshmukh, 9/19/2025, 12:12 AM), and Modified By (Utkarsh Deshmukh, 9/19/2025, 12:12 AM). Action buttons for Edit, Clone, Delete, and View Users are at the top of the detail section.

The screenshot shows the 'Regional_Admin_Profile' setup page. The layout is identical to the Super Admin profile page, with the same header, profile name, notes, and permission lists. The 'Profile Detail' section shows the following details:

- Name: Regional_Admin_Profile
- User License: Salesforce
- Description: (empty)
- Created By: Utkarsh Deshmukh, 9/19/2025, 12:15 AM
- Modified By: Utkarsh Deshmukh, 9/19/2025, 12:15 AM

Action buttons for Edit, Clone, Delete, and View Users are also present.

Profile Lot_Manager_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 \[0\]](#) | [Enabled Apex Class Access \[0\]](#) | [Enabled Visualforce Page Access \[0\]](#) | [Enabled External Data Source Access \[0\]](#) | [Enabled Named Credential Access \[0\]](#) | [Enabled External Credential Principal Access \[0\]](#) | [Enabled Custom Metadata Type Access \[0\]](#) | [Enabled Custom Setting Definitions Access \[0\]](#) | [Enabled Flow Access \[0\]](#) | [Enabled Service Presence Status Access \[0\]](#) | [Enabled Custom Permissions \[0\]](#)

Profile Detail	
Name	Lot_Manager_Profile
User License	Salesforce Platform
Description	
Created By	Utkarsh Deshmukh, 9/19/2025, 1:14 AM
Modified By	Utkarsh Deshmukh, 9/19/2025, 1:14 AM

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

Roles:

Defined a role hierarchy to reflect organizational structure:

- **Super Admin (top-level)**
- **Regional Manager (reports to Super Admin)**
- **Lot Manager (reports to Regional Manager)**

Creating the Role Hierarchy

You can build on the existing role hierarchy shown on this page. To insert a new role, click **Add Role**.

Your Organization's Role Hierarchy

[Collapse All](#) [Expand All](#) [Show in tree view](#)

- ParkVista Smart Parking Solutions
 - [Add Role](#)
 - Super_Admin** [Edit](#) [Del](#) [Assign](#)
 - [Add Role](#)
 - Regional_Admin** [Edit](#) [Del](#) [Assign](#)
 - [Add Role](#)
 - Lot Manager – Pune Central** [Edit](#) [Del](#) [Assign](#)
 - [Add Role](#)

Permission Sets:

Not configured at this stage. Planned for later phases to grant additional object-level or feature-specific permissions beyond the baseline profiles.

Users:

Created sample users and assigned them respective profiles and roles to test the setup.

- **Super Admin** assigned to **Super_Admin_Profile** and **Super Admin role**.
- **Regional Admin** assigned to **Regional_Admin_Profile** and **Regional Admin role**.
- **Lot Manager** assigned to **Lot_Manager_Profile** and **Lot Manager-Pune Central role**.

User
Super Admin

[Permission Set Assignments](#) | [Permission Set Assignments: Activation Required](#) | [Permission Set Group Assignments](#) | [Permission Set License Assignments](#) | [Personal Groups](#) | [Public Group Membership](#) | [Queue Membership](#) | [Team](#) | [Managers in the Role Hierarchy](#) | [OAuth Apps](#) | [Third-Party Account Links](#) | [Built-in Authenticators](#) | [Installed Mobile Apps](#) | [Authentication Settings for External Systems](#) | [Login History](#) | [User Provisioning Accounts](#)

User Detail		Edit	Sharing	Reset Password	Freeze	View Summary
Name	Super Admin	Role <u>Super_Admin</u>				
Alias	sadmin	User License <u>Salesforce</u>				
Email	ukarshrushh@gmail.com [Verify]	Profile <u>Super_Admin_Profile</u>				
Username	ukarshrushh@gmail.com	Active <input checked="" type="checkbox"/>				
Nickname	SuperAdmin [i]	Marketing User <input type="checkbox"/>				
Title	Super Admin	Offline User <input type="checkbox"/>				
Company	ParkVista	Knowledge User <input type="checkbox"/>				
Department		Flow User <input type="checkbox"/>				
Division		Service Cloud User <input type="checkbox"/>				
Address	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 View				
Delegated Approver		Data.com User Type [i]				
Manager		Accessibility Mode (Classic Only) <input type="checkbox"/> [i]				
Receive Approval Request Emails	Only if I am an approver	Debug Mode <input type="checkbox"/> [i]				
Federation ID		High-Contrast Palette on Charts <input type="checkbox"/> [i]				
App Registration: One-Time Password Authenticator	[i]	Load Lightning Pages While Scrolling <input checked="" type="checkbox"/> [i]				
App Registration: Salesforce Authenticator	[i]	Send Apex Warning Emails <input type="checkbox"/>				

User
Regional Admin

[Permission Set Assignments](#) | [Permission Set Assignments: Activation Required](#) | [Permission Set Group Assignments](#) | [Permission Set License Assignments](#) | [Personal Groups](#) | [Public Group Membership](#) | [Queue Membership](#) | [Team](#) | [Managers in the Role Hierarchy](#) | [OAuth Apps](#) | [Third-Party Account Links](#) | [Built-in Authenticators](#) | [Installed Mobile Apps](#) | [Authentication Settings for External Systems](#) | [Login History](#) | [User Provisioning Accounts](#)

User Detail		Edit	Sharing	Reset Password	Freeze	View Summary
Name	Regional Admin	Role <u>Regional_Admin</u>				
Alias	radmin	User License <u>Salesforce</u>				
Email	ukarshrushh@gmail.com [Verify]	Profile <u>Regional_Admin_Profile</u>				
Username	utkarshdeshmukh.tech@gmail.com	Active <input checked="" type="checkbox"/>				
Nickname	User17582692431275067735 [i]	Marketing User <input type="checkbox"/>				
Title	Regional Admin – Maharashtra	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	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 View				
Delegated Approver		Data.com User Type [i]				
Manager		Accessibility Mode (Classic Only) <input type="checkbox"/> [i]				
Receive Approval Request Emails	Only if I am an approver	Debug Mode <input type="checkbox"/> [i]				
Federation ID		High-Contrast Palette on Charts <input type="checkbox"/> [i]				
App Registration: One-Time Password Authenticator	[i]	Load Lightning Pages While Scrolling <input checked="" type="checkbox"/> [i]				
App Registration: Salesforce Authenticator	[i]	Salesforce CRM Content User <input checked="" type="checkbox"/>				

Licenses:

1. Salesforce License (Full CRM Access):

- Assigned to **Super Admin and Regional Admins**.
- Provides full access to standard and custom objects, reports, dashboards, and setup permissions (for Super Admin).

2. Salesforce Platform License:

- Assigned to **Lot Managers**.
- Grants access to custom objects like **Parking Lot**, **Parking Spot**, **Reservation**, and **Payment**, but no standard CRM setup permissions.

Org-Wide Defaults (OWD)

At this stage, OWD is not yet configured as the required custom objects are not available. Once objects like **Parking Lot**, **Parking Spot**, **Reservation**, and **Payment** are created, OWD will be defined to set the baseline record-level access.

Sharing Rules

Currently not configured as no objects exist. Sharing Rules will be applied later to **open up access** beyond the role hierarchy where required.

Login Access Policies:

- **Navigate:** Setup → Quick Find → Login Access Policies.
- **Enabled:** “Administrators Can Log in as Any User”
 - Allows Super Admin to temporarily access other user accounts for troubleshooting.

The screenshot shows the 'Login Access Policies' page in the Salesforce Setup. At the top, there's a 'Manage Support Options' section with 'Save' and 'Cancel' buttons. Below it is a table with two rows. The first row has a 'Setting' column labeled 'Administrators Can Log in as Any User' with a checked checkbox. The second row has columns for 'Support Organization' (labeled 'Salesforce.com Support'), 'Packages' (radio button selected), 'Available to Users' (radio button selected), and 'Available to Administrators Only' (radio button unselected). At the bottom of the page are 'Save' and 'Cancel' buttons.

Session Settings:

- **Navigate:** Setup → Quick Find → Session Settings.
- **Configured:**
 - Session Timeout: 2 hours
 - Force Logout on Session Timeout: Enabled (optional, for extra security)

Dev Org Setup

- **Salesforce Org:** A Salesforce Developer Edition org was created to implement the **ParkVista Smart Parking Solutions** project. This environment

provides full access to **Salesforce features** required for both **Admin** and **Developer** tasks.

- **Source Control:** A **GitHub repository** was created to manage **version control, track changes, and maintain code history** for **Apex classes, LWC components, and configuration metadata**.
- **Development Environment:**
 - **VS Code** was set up as the integrated development environment.
 - **Salesforce CLI (SFDX)** installed to deploy, retrieve, and test **Salesforce metadata** and **Lightning Web Components (LWC)** during development.
 - Connected VS Code with the Developer Edition org using **SFDX authentication**.

Outcome:

This setup ensures a structured development workflow, enables collaboration via GitHub, and allows implementation of both Admin configurations and Developer components efficiently.

Conclusion (Phase 2):

Phase 2 established the **ParkVista Salesforce org** with all essential configurations, including **user roles, profiles, licenses, and login access policies**. These setups provide a **secure and organized** structure, enabling smooth **administration** and **controlled access** for future processes.

Phase 3: Data Modeling & Relationships

– ParkVista Smart Parking Solutions

The purpose of this phase was to establish the Salesforce data structure for managing parking lots, parking spots, reservations, and users efficiently. Proper data modeling ensures accurate tracking of availability, bookings, and customer details, forming the foundation for automation in later phases.

1. Standard & Custom Objects

- **Parking Lot : Represents physical parking lots and manages slot inventory.**

SETUP > OBJECT MANAGER
Parking Lot

Details	Details	
Fields & Relationships	Description	<input type="button" value="Edit"/> <input type="button" value="Delete"/>
Page Layouts		
Lightning Record Pages	API Name Parking_Lot__c	
Buttons, Links, and Actions	Custom ✓	
Compact Layouts	Singular Label Parking Lot	
Field Sets	Plural Label Parking Lots	
Object Limits		
Record Types		
Related Lookup Filters		
Restriction Rules		
Scoping Rules		
Object Access		
Triggers		
Flow Triggers		
Validation Rules		

- **Parking Spot : Represents individual parking spots within a lot**

SETUP > OBJECT MANAGER
Parking_Spot

Details	Details	
Fields & Relationships	Description	<input type="button" value="Edit"/> <input type="button" value="Delete"/>
Page Layouts		
Lightning Record Pages	API Name Parking_Spot__c	
Buttons, Links, and Actions	Custom ✓	
Compact Layouts	Singular Label Parking_Spot	
Field Sets	Plural Label Parking_Spots	
Object Limits		
Record Types		
Related Lookup Filters		
Restriction Rules		
Scoping Rules		
Object Access		
Triggers		

- **Reservation : Tracks booking of parking spots and calculates charges.**

The screenshot shows the Salesforce Object Manager interface for the 'Reservation' object. The left sidebar lists various configuration tabs: Fields & Relationships, Page Layouts, Lightning Record Pages, Buttons, Links, and Actions, Compact Layouts, Field Sets, Object Limits, Record Types, Related Lookup Filters, Restriction Rules, Scoping Rules, Object Access, and Triggers. The main 'Details' tab is selected. The API Name is set to 'Reservation__c'. The Singular Label is 'Reservation' and the Plural Label is 'Reservations'. Under the 'Enable Reports' section, 'Track Activities' and 'Track Field History' are checked. Deployment Status is set to 'Deployed'. Help Settings point to the standard Salesforce help window. At the top right, there are 'Edit' and 'Delete' buttons.

- **Booking Object: Maintains confirmed bookings for reporting and customer reference.**

The screenshot shows the Salesforce Object Manager interface for the 'Booking' object. The left sidebar lists various configuration tabs: Fields & Relationships, Page Layouts, Lightning Record Pages, Buttons, Links, and Actions, Compact Layouts, Field Sets, Object Limits, Record Types, Related Lookup Filters, Search Layouts, and List View Button Layout. The main 'Details' tab is selected. The API Name is set to 'Booking__c'. The Singular Label is 'Booking' and the Plural Label is 'Bookings'. Under the 'Enable Reports' section, 'Track Activities' and 'Track Field History' are checked. Deployment Status is set to 'Deployed'. Help Settings point to the standard Salesforce help window. At the top right, there are 'Edit' and 'Delete' buttons.

- **Contact (Standard): Represents customers making reservations.**

2.Fields

- **Fields for Parking_Lot_c**

The screenshot shows the Salesforce Object Manager interface for the 'Parking Lot' object. The left sidebar lists various setup categories like Details, Fields & Relationships, Page Layouts, etc. The main content area displays the 'Fields & Relationships' section with 14 items, sorted by Field Label. Each item shows the field name, its type, and its description. For example, 'Available Spots' is a Number(3, 0) field, and 'City' is a Text(20) field.

Fields & Relationships		
14 Items, Sorted by Field Label		
Available Spots	Available_Spots__c	Number(3, 0)
City	City__c	Text(20)
Created By	CreatedById	Lookup(User)
Last Modified By	LastModifiedById	Lookup(User)
Lot ID	Name	Auto Number
Lot Name	Lot_Name__c	Text(100)
Notes	Notes__c	Long Text Area(32768)
Owner	OwnerId	Lookup(User,Group)
Pin Code	Pin_Code__c	Text(10)
Price Per Hour	Price_Per_Hour__c	Currency(10, 2)
Record Type	RecordTypeId	Record Type
State	State__c	Picklist

Fields for Reservation_c

The screenshot shows the Salesforce Object Manager interface for the 'Reservation' object. Similar to the previous screenshot, it displays the 'Fields & Relationships' section with 14 items, sorted by Field Label. The fields include relationships to other objects like 'Contact', 'Parking Lot', and 'Payment Status', as well as various data types such as Date/Time, Picklist, and Formula.

Fields & Relationships		
14 Items, Sorted by Field Label		
Contact	Contact__c	Lookup(Contact)
Created By	CreatedById	Lookup(User)
Duration (Hours)	Duration_Hours__c	Formula (Number)
End Time	End_Time__c	Date/Time
Last Modified By	LastModifiedById	Lookup(User)
Owner	OwnerId	Lookup(User,Group)
Parking Lot	Parking_Lot__c	Lookup(Parking Lot)
Payment Status	Payment_Status__c	Picklist
Record Type	RecordTypeId	Record Type
Reservation Date	Reservation_Date__c	Date
Reservation Number	Name	Auto Number
Start Time	Start_Time__c	Date/Time

3. Record Types

Record Types on Reservation Object:

- **Standard Reservation:** Regular customer bookings.
- **Corporate Reservation:** Bookings for corporate accounts or special groups.

The screenshot shows the Salesforce Object Manager interface for the 'Reservation' object. The left sidebar lists various setup options like Details, Fields & Relationships, Page Layouts, etc., with 'Record Types' selected. The main content area displays a table titled 'Record Types' with two items: 'Corporate Reservation' and 'Standard Reservation'. The table includes columns for 'Record Type Label', 'Description', 'Active', and 'Modified By'. Both records were created by 'Utkarsh Deshmukh' on different dates.

RECORD TYPE LABEL	DESCRIPTION	ACTIVE	MODIFIED BY
Corporate Reservation	For bulk or corporate bookings	✓	Utkarsh Deshmukh, 9/29/2025, 5:15 AM
Standard Reservation	For regular individual booking	✓	Utkarsh Deshmukh, 9/26/2025, 1:06 AM

Record Types on Parking Lot Object:

- **Private Lot:** Internal/admin-managed lots (e.g., restricted access).
- **Public Lot:** Available for general customer reservations.

The screenshot shows the Salesforce Object Manager for the 'Parking Lot' object. On the left, a sidebar lists various setup options like Details, Fields & Relationships, Page Layouts, etc., with 'Record Types' selected. The main content area displays a table titled 'Record Types' with two items: 'Private Lot' and 'Public Lot'. Both records are marked as Active and were modified by Utkarsh Deshmukh on 9/29/2025 at 3:16 AM. A 'Quick Find' search bar and 'New' button are at the top right.

RECORD TYPE LABEL	DESCRIPTION	ACTIVE	MODIFIED BY
Private Lot		✓	Utkarsh Deshmukh, 9/29/2025, 3:16 AM
Public Lot		✓	Utkarsh Deshmukh, 9/29/2025, 3:16 AM

4. Page Layout

The screenshot shows the Salesforce Object Manager for the 'Parking Lot' object. On the left, 'Page Layouts' is selected in the sidebar. The main content area displays a table titled 'Page Layouts' with two items: 'Parking Layout Customer' and 'Parking Lot Layout Manager'. Both layouts were created by Utkarsh Deshmukh. The 'Parking Layout Customer' was created on 9/26/2025 at 1:11 AM and last modified on 9/29/2025 at 5:24 AM. The 'Parking Lot Layout Manager' was created on 9/19/2025 at 10:07 AM and last modified on 9/29/2025 at 5:24 AM. A 'Quick Find' search bar and 'New' button are at the top right.

PAGE LAYOUT NAME	CREATED BY	MODIFIED BY
Parking Layout Customer	Utkarsh Deshmukh, 9/26/2025, 1:11 AM	Utkarsh Deshmukh, 9/29/2025, 5:24 AM
Parking Lot Layout Manager	Utkarsh Deshmukh, 9/19/2025, 10:07 AM	Utkarsh Deshmukh, 9/29/2025, 5:24 AM

Customer Layout: Shows only booking-relevant fields: Lot Name, Available Spots, Price Per Hour, Status. Hides internal fields like Total Slots, Notes, Owner.

Manager/Admin Layout : Shows all fields for full management: Lot Name, Total Slots, Available Spots, Price Per Hour, Status, Notes, Owner, City, Pin Code, State.

5. Compact Layout

Parking Lot Compact Layout: Shows key fields for quick view: Lot Name, Available Spots, Price Per Hour, Status. Useful for lookup fields and mobile view.

Reservation Compact Layout: Shows main info: Reservation Number, Parking Lot, Start Time, End Time, Status. Allows users to see essential reservation details quickly

The screenshot shows the Salesforce Setup interface for the Object Manager. The left sidebar has links for Details, Fields & Relationships, Page Layouts, Lightning Record Pages, Buttons, Links, and Actions, and Compact Layouts. The Compact Layouts link is selected and highlighted in blue. The main content area is titled "Compact Layouts" and shows a table with one item: "Parking Lot compact layout" (API Name: Parking_Lot_compact_layout). The table includes columns for Label, API Name, Primary, Modified By, and Last Modified. A "Quick Find" search bar and "New" and "Compact Layout Assignment" buttons are at the top right.

The screenshot shows the Salesforce Setup interface for the Object Manager. The left sidebar has links for Details, Fields & Relationships, Page Layouts, Lightning Record Pages, Buttons, Links, and Actions, and Compact Layouts. The Compact Layouts link is selected and highlighted in blue. The main content area is titled "Compact Layouts" and shows a table with one item: "Reservation compact layout" (API Name: Reservation_compact_layout). The table includes columns for Label, API Name, Primary, Modified By, and Last Modified. A "Quick Find" search bar and "New" and "Compact Layout Assignment" buttons are at the top right.

5. Lookup vs Master-Detail vs Hierarchical Relationships

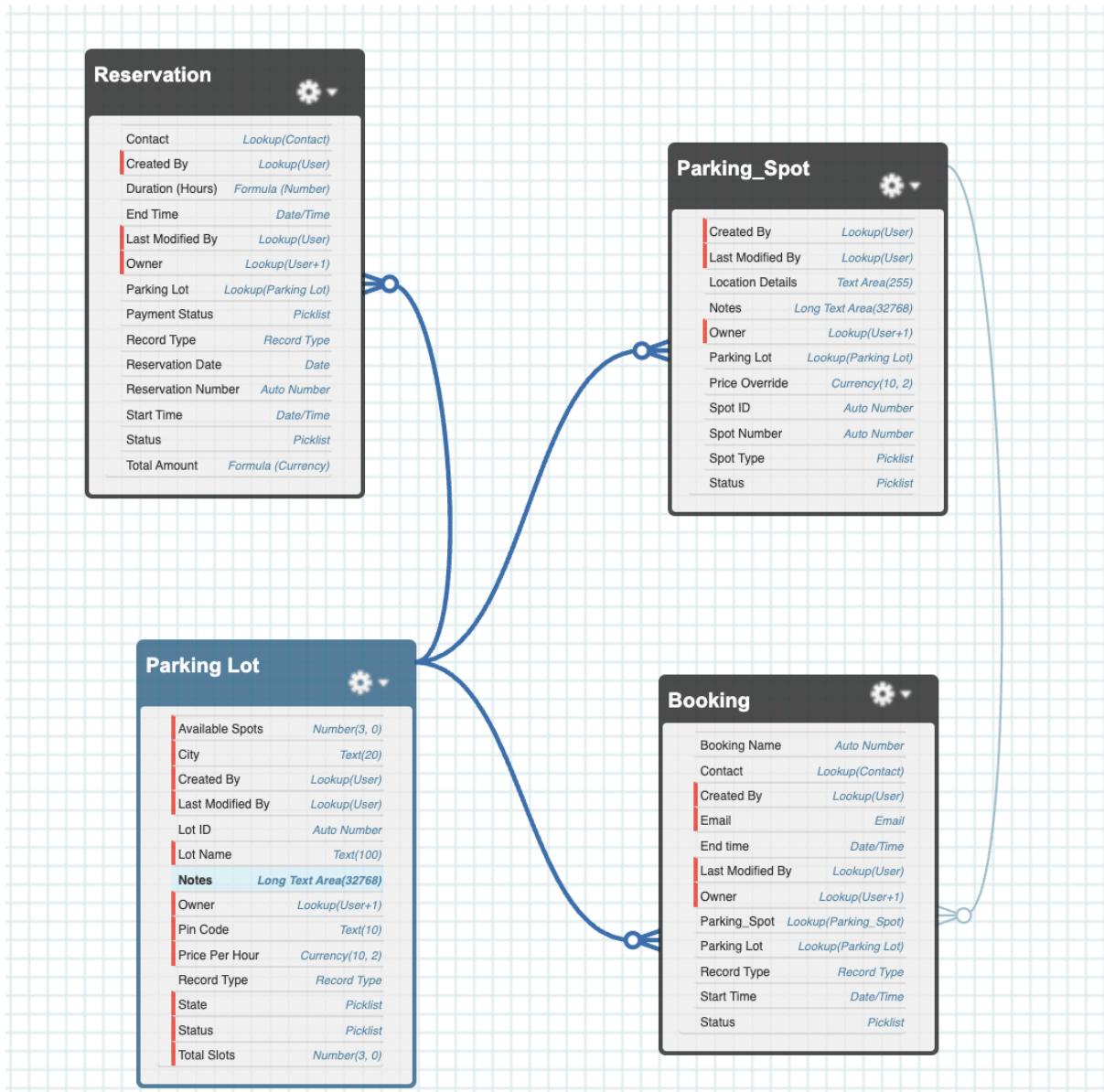
Reservation → Parking Lot: Lookup relationship. Each reservation points to a parking lot (Reservation__c.Parking_Lot__c). This allows a reservation to be linked to a lot without enforcing deletion cascade.

Parking Spot → Parking Lot: Lookup relationship. Each parking spot points to a parking lot (Parking_Spot__c.Parking_Lot__c). This links spots to lots while keeping them independent if the lot is deleted.

Purpose: Enables connecting reservations and parking spots to the respective parking lot, supports reporting, and ensures data consistency without enforcing master-detail constraints.

Workflow: When a customer books a reservation, the system links it to the correct parking lot via the lookup. Parking spots are also associated with the lot for availability tracking and assignment.

6. Schema Builder



Phase 4: Process Automation

1. Validation Rules

Description:

Validation Rules enforce **data integrity** by preventing users from saving records with invalid or illogical values.

Smart Parking Examples:

- **Booking Date/Time Check:** Prevent a reservation if $\text{End_Time_c} \leq \text{Start_Time_c}$.
- **Payment Amount Validation:** Ensure $\text{Payment_Amount_c} > 0$.
- **Available Spot Check:** Prevent booking if $\text{Available_Spots_c} = 0$.

The image displays two screenshots of the Salesforce Object Manager interface, specifically for the 'Validation Rules' section of the 'Booking' and 'Payment' objects.

Booking Object Validation Rules:

- Rule Name:** End_before_Start
- Error Location:** Top of Page
- Error Message:** End Time must be after Start Time
- Active:** ✓
- Modified By:** Utkarsh Deshmukh, 9/27/2025, 1:22 AM

Payment Object Validation Rules:

- Rule Name:** Payment_Amount_Positive
- Error Location:** Top of Page
- Error Message:** Payment Amount must be greater than 0.
- Active:** ✓
- Modified By:** Utkarsh Deshmukh, 9/29/2025, 5:28 AM

2. Workflow Rules

Reservation Confirmation Email

Purpose:

Automatically notifies the customer via email when a parking reservation is successfully created, ensuring timely communication and enhancing user experience.

Object:

Reservation__c (Custom Object)

Rule Name:

Reservation_Confirmation_Workflow

Rule Criteria:

- **Evaluation Criteria:** Evaluate the rule when a record is created

The screenshot shows the 'Workflow Rules' page in Salesforce. The top navigation bar includes 'SETUP' and 'Workflow Rules'. Below the header, it says 'Workflow Rule' and 'Booking_Confirmation'. A message bar at the top right says 'Help for this Page' with a question mark icon. The main section is titled 'Workflow Rule Detail' for 'Booking_Confirmation'. It shows the rule is active, created by Utkarsh Deshmukh on 9/30/2025 at 3:16 AM, and modified by Utkarsh Deshmukh on 9/30/2025 at 3:21 AM. The evaluation criteria is 'Evaluate the rule when a record is created'. The object is 'Reservation'. The rule criteria is 'Reservation: Payment Status EQUALS "Pending,"'. The 'Workflow Actions' section contains an immediate workflow action: 'Email Alert' named 'Reservation_Confirmation_Alert'. A note below states 'You cannot add new time triggers to an active rule.' with a link to 'Deactivate This Rule'.

3. Process Builder:

SmartParking_Reservation_Process

Object: Reservation__c

Trigger: When a record is created or edited

Purpose:

Automate key actions for reservations: sending email confirmations, updating reservation status, and notifying admins when no parking spots are available.

Stage 1: Reservation Created

Criteria Name: New_Reservation

Condition: Reservation record is created (`[Reservation__c].Number != null`)

Immediate Actions:

- Email Alert: Send Booking Confirmation Email to customer.
- Field Update: Set Status__c = 'Pending'.

Purpose: Confirms to the customer that their booking request has been registered.

Stage 2: Payment Completed

Criteria Name: Payment_Completed

Condition: Payment status is Paid (`[Reservation__c].Payment_Status__c = 'Paid'`)

Immediate Actions:

- Field Update: Set Status__c = 'Confirmed'.
- Email Alert: Send Reservation Confirmed Email to customer.

Purpose: Updates reservation as confirmed once payment is done and informs the customer.

Stage 3: No Available Spots

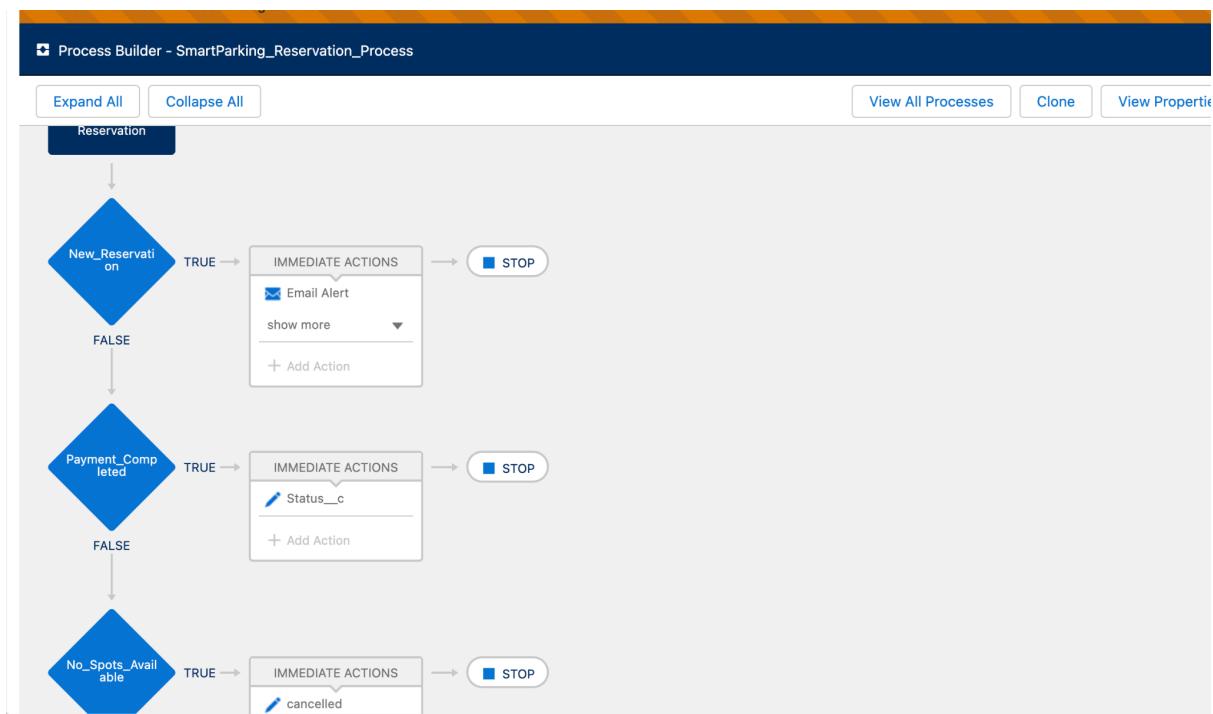
Criteria Name: No_Spots_Available

Condition: Parking lot has zero available spots
(`[Reservation__c].Parking_Lot__r.Available_Spots__c = 0`)

Immediate Actions:

- Custom Notification: Notify admin.
- Field Update: Set Status__c = 'Cancelled'.

Purpose: Alerts admin about overbooking attempts and updates reservation status to failed.



4. Flow Builder:

Flow 1: Booking After Create or Update

Type: Record-Triggered Flow

Trigger Object: Booking__c

Trigger Condition: When a record is created or updated

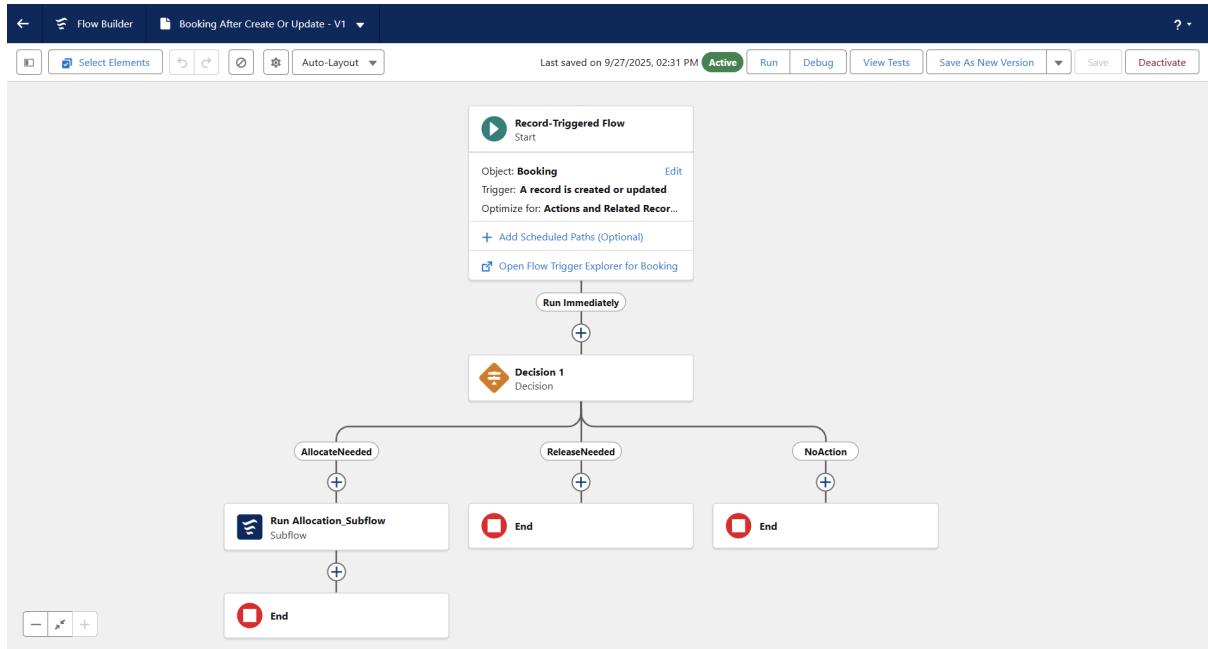
Optimization: Actions and Related Records

Purpose:

Automatically determines if a parking allocation or release action is needed whenever a booking record is created or updated. Ensures bookings are processed without manual intervention.

Key Notes:

- Keeps booking allocations synchronized.
- Modular design using subflow.
- Reduces errors and manual processing.



Flow 2: Booking Confirmation Alert Email – V3

Type: Record-Triggered Flow

Trigger Object: Booking__c

Trigger Condition: When a record is created

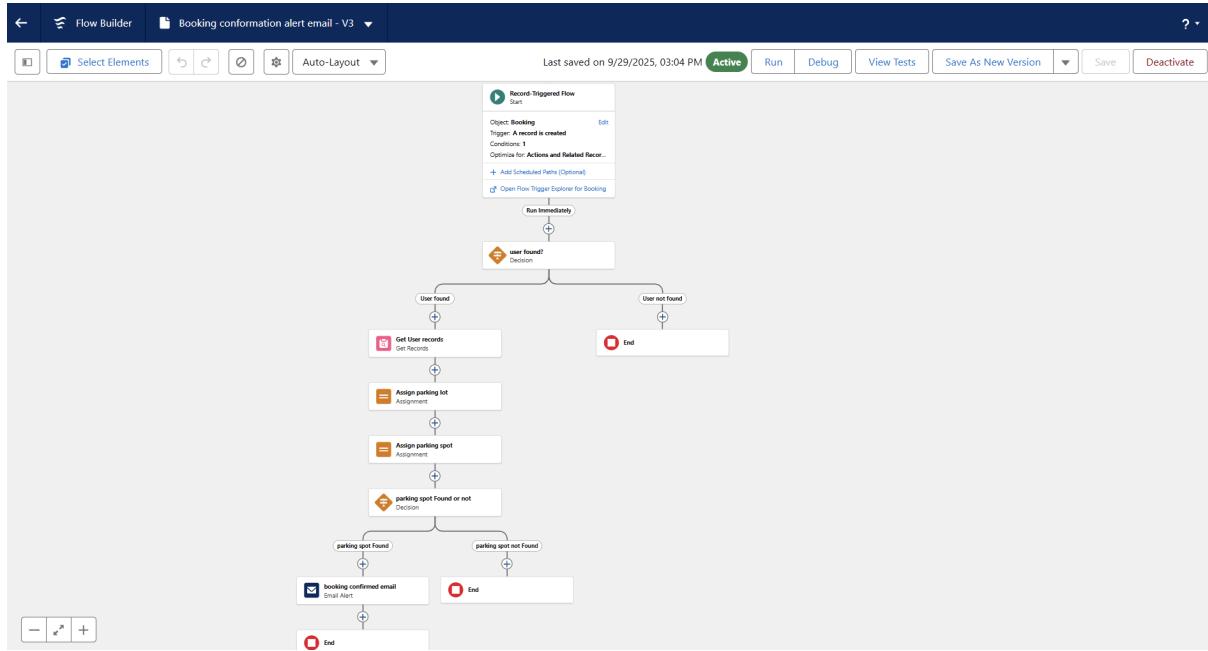
Optimization: Actions and Related Records

Purpose:

Automates sending booking confirmation emails and assigns parking when a booking is created. Ensures users are notified and parking is allocated correctly.

Key Notes:

- Automates email communication to reduce manual work.
- Ensures parking is correctly assigned before confirmation.
- Handles errors gracefully if user or parking data is missing.



4. Email Alerts:

- **Description:**

Email Alerts automatically send predefined email templates when triggered by Workflow, Process Builder, or Flow.

- **Examples for Smart Parking:**

- Booking Confirmation to customer when booking is successful.
- No-Space Notification when system can't allocate a space.

Email Alert Detail

Description: EA_Booking_Confirmation
Unique Name: EA_Booking_Confirmation
From Email Address: Current User's email address
Recipients: User_OpFarm_EPIC, User_Integration_User, User_Security_User, User_Utkarsh_Deshmukh, User_RegionAdmin, User_LotManager, User_SuperAdmin
Additional Emails: Created By: Utkarsh Deshmukh, 9/27/2025, 2:07 AM
Modified By: Utkarsh Deshmukh, 9/27/2025, 2:07 AM

Rules Using This Email Alert
This alert is currently not used by any rules.

Approval Processes Using This Email Alert
This alert is currently not used by any approval processes.

Entitlement Processes Using This Email Alert
This alert is currently not used by any entitlement processes.

Flows Using This Email Alert

Email Alert Detail

Description: Booking Update — No Available Space for {!Booking__c.Name}
Unique Name: Booking_Update_No_Available_Space_for_Booking_c_Name
From Email Address: Current User's email address
Recipients: User_OpFarm_EPIC, User_Integration_User, User_Security_User, User_Utkarsh_Deshmukh, User_RegionAdmin, User_LotManager, User_SuperAdmin
Additional Emails: Created By: Utkarsh Deshmukh, 9/27/2025, 2:12 AM
Modified By: Utkarsh Deshmukh, 9/27/2025, 2:12 AM

Rules Using This Email Alert
This alert is currently not used by any rules.

Approval Processes Using This Email Alert
This alert is currently not used by any approval processes.

Entitlement Processes Using This Email Alert
This alert is currently not used by any entitlement processes.

Flows Using This Email Alert

Conclusion – Phase 4: Process Automation

In this phase, various automation tools such as Validation Rules, Workflow Rules, Process Builder, Approval Processes, Email Alerts, Field Updates, Tasks, and Custom Notifications were explored. While it may appear that not all features were implemented individually, most of these functionalities are integrated within the Flow Builder. Using Record-Triggered and Auto-Launched Flows, we have effectively automated validations, notifications, allocations, and updates, demonstrating that Flow Builder serves as a central tool encompassing multiple automation capabilities.

Phase 5: Apex Programming (Developer)

1. ReservationTrigger + ReservationHandler

Purpose:

Automates parking lot slot availability whenever a reservation is made.

Importance:

Without this, admins would have to manually reduce available spots when someone books. This ensures real-time slot tracking.

Workflow:

- User creates a new `Reservation__c` record (via UI or API).
- `ReservationTrigger` fires **after insert**.
- Calls `ReservationHandler.handleReservation()`.
- Handler reduces the `Available_Spots__c` of the linked `Parking_Lot__c`.

Work in app:

If Parking Lot had 10 spots and someone books → system automatically decrements to 9.

```

1  public with sharing class ReservationHandler {
2
3      public static void handleReservation(List<Reservation__c> newReservations){
4          Set<Id> lotIds = new Set<Id>();
5          for(Reservation__c r : newReservations){
6              if(r.Parking_Lot__c != null) lotIds.add(r.Parking_Lot__c);
7          }
8
9          Map<Id, Parking_Lot__c> lotMap = new Map<Id, Parking_Lot__c>(
10             [SELECT Id, Available_Spots__c, Price_Per_Hour__c
11              FROM Parking_Lot__c
12              WHERE Id IN :lotIds]
13         );
14
15         for(Reservation__c r : newReservations){
16             if(r.Parking_Lot__c != null && lotMap.containsKey(r.Parking_Lot__c)){
17                 Parking_Lot__c lot = lotMap.get(r.Parking_Lot__c);
18                 if(lot.Available_Spots__c > 0){
19                     lot.Available_Spots__c -= 1;
20                 }
21             }
22         }
23
24         update lotMap.values();
25     }
26 }

```

The screenshot shows the ReservationHandler.cls code in the main editor. The code defines a class with a static method handleReservation that takes a list of new reservations. It iterates through the list, adds the parking lot IDs to a set, and then creates a map of parking lots from a query. Finally, it loops through the new reservations again, checks if the parking lot exists in the map, and if so, decrements the available spots by one. The bottom status bar shows the deployment log: "16:21:47.833 Ended SFDX: Deploy This Source to Org" and "16:22:05.924 Starting SFDX: Deploy This Source to Org".

```

1 trigger ReservationTrigger on Reservation__c (after insert) {
2     for(Reservation__c r : Trigger.new){
3         if(Trigger.isAfter && Trigger.isInsert){
4             ReservationHandler.handleReservation(Trigger.new);
5         }
6     }
}

```

2. ReservationHandlerTest

Purpose:

Ensures your business logic works correctly.

Importance:

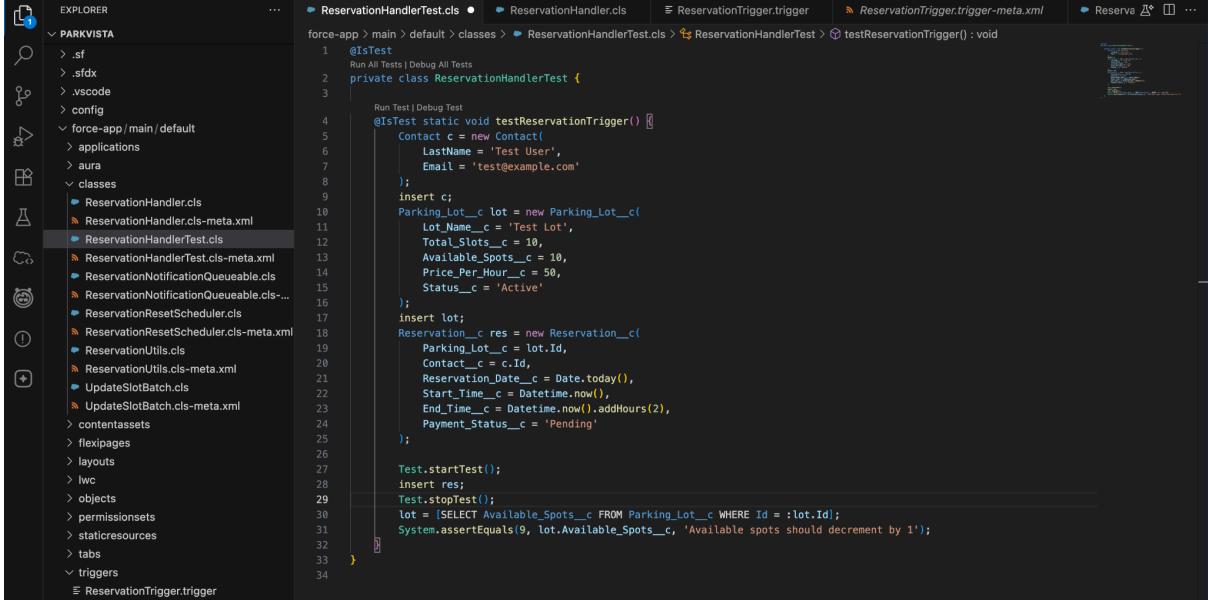
Required for **deployment to production** (Salesforce needs test coverage). Also protects against future code breaking.

Workflow:

- Creates a test parking lot.
- Inserts a reservation.
- Asserts that available spots decreased by 1.

Work in app:

Nothing runs in production, but ensures your logic works as intended.



The screenshot shows the Salesforce IDE interface. The left sidebar is the Explorer, displaying the project structure for 'PARKVISTA'. The main area is the Editor tab, showing the code for 'ReservationHandlerTest.cls'. The code is a test class for a Reservation Handler. It includes setup code to create a Contact, a Parking Lot, and a Reservation, followed by assertions to check the available spots.

```
1  @isTest
2  Run All Tests | Debug All Tests
3
4  private class ReservationHandlerTest {
5      @Test
6      Run Test | Debug Test
7      Contact c = new Contact(
8          LastName = 'Test User',
9          Email = 'test@example.com'
10     );
11    insert c;
12    Parking_Lot__c lot = new Parking_Lot__c(
13        Lot_Name__c = 'Test Lot',
14        Total_Slots__c = 10,
15        Available_Spots__c = 10,
16        Price_Per_Hour__c = 50,
17        Status__c = 'Active'
18    );
19    insert lot;
20    Reservation__c res = new Reservation__c(
21        Parking_Lot__c = lot.Id,
22        Contact__c = c.Id,
23        Reservation_Date__c = Date.today(),
24        Start_Time__c = Datetime.now(),
25        End_Time__c = Datetime.now().addHours(2),
26        Payment_Status__c = 'Pending'
27    );
28
29    Test.startTest();
30    insert res;
31    Test.stopTest();
32    lot = [SELECT Available_Spots__c FROM Parking_Lot__c WHERE Id = :lot.Id];
33    System.assertEquals(9, lot.Available_Spots__c, 'Available spots should decrement by 1');
34 }
```

3. ReservationNotificationQueueable

Purpose:

Sends **email confirmation** to customers after booking.

Importance:

Keeps customers informed and adds professionalism. Queueable ensures sending is done asynchronously (non-blocking).

Workflow:

- Takes reservations as input.
- Queries `Contact__r.Email` of each reservation.
- Sends a “Reservation Confirmed” email.

Work in app:

If a customer books, they immediately receive an email:
"Your reservation for Lot A is confirmed."

```

1  public class ReservationNotificationQueueable implements Queueable {
2
3      private List<Id> reservationIds;
4
5      public ReservationNotificationQueueable(List<Reservation__c> res){
6          this.reservationIds = new List<Id>();
7          for(Reservation__c r : res){
8              this.reservationIds.add(r.Id);
9          }
10     }
11
12     public void execute(QueueableContext ctx){
13         List<Messaging.SingleEmailMessage> emails = new List<Messaging.SingleEmailMessage>();
14         List<Reservation__c> reservations = [
15             SELECT Id, Name, Parking_Lot__r.Lot_Name__c, Contact__r.Email
16             FROM Reservation__c
17             WHERE Id IN :reservationIds
18         ];
19
20         for(Reservation__c r : reservations){
21             if(r.Contact__r != null && r.Contact__r.Email != null){
22                 Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();
23                 mail.setToAddresses(new String[]{r.Contact__r.Email});
24                 mail.setSubject('Reservation Confirmed: ' + r.Name);
25                 mail.setPlainTextBody(
26                     'Your reservation for lot ' + r.Parking_Lot__r.Lot_Name__c + ' is confirmed.'
27                 );
28                 emails.add(mail);
29             }
30         }
31
32         if(!emails.isEmpty()){
33             Messaging.sendEmail(emails);
34         }
35     }
36 }
37

```

4. ReservationUtils

Purpose:

Sends **reminder emails** to customers whose reservations are about to start soon.

Importance:

Improves customer experience, prevents no-shows.

Workflow:

- Finds all reservations starting in the **next 1 hour**.
- Sends a reminder email with lot name and start time.

Work in app:

Customer gets: “*Reminder: Your reservation at Lot A starts at 3:00 PM.*”

The screenshot shows the Salesforce IDE interface. The left sidebar (Explorer) lists various files and components under the 'PARKVISTA' project. The right pane (Editor) displays the code for the `ReservationUtils` class, specifically the `sendReservationReminders()` method. The code uses SOQL to query reservations between two windows and then sends a reminder email to each contact.

```

trigger ReservationTrigger.trigger-meta.xml
  ...
  force-app > main > default > classes > ReservationUtils.cls > ReservationUtils > sendReservationReminders() : void
  ...
public static void sendReservationReminders() {
    Datetime startWindow = Datetime.now();
    Datetime endWindow = startWindow.addHours(1);

    List<Reservation__c> reservations = [
        SELECT Id, Start_Time__c, Contact__r.Email, Parking_Lot__r.Lot_Name__c
        FROM Reservation__c
        WHERE Start_Time__c >= :startWindow
        AND Start_Time__c <= :endWindow
    ];

    List<Messaging.SingleEmailMessage> emails = new List<Messaging.SingleEmailMessage>();

    for (Reservation__c r : reservations) {
        if (r.Contact__r != null && r.Contact__r.Email != null) {
            Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();
            mail.setToAddresses(new String[] { r.Contact__r.Email });
            mail.setSubject('Reminder: Upcoming Reservation ' + r.Name);
            mail.setPlainTextBody(
                'This is a reminder for your reservation at lot ' +
                r.Parking_Lot__r.Lot_Name__c +
                '. Your start time is ' + r.Start_Time__c.format()
            );
            emails.add(mail);
        }
    }

    if (!emails.isEmpty()) {
        Messaging.sendEmail(emails);
    }
}

```

5. UpdateSlotBatch

Purpose:

Resets parking lot availability back to full capacity.

Importance:

Keeps lots accurate at the start of each day or after resets. Prevents “permanent” reduction from old bookings.

Workflow:

- Batch queries all parking lots.
- Sets `Available_Spots__c = Total_Slots__c`.
- Updates them in bulk.

Work in app:

If a lot had 10 total spots but was down to 2, the next day batch resets back to 10.

The screenshot shows the Salesforce IDE interface. The left sidebar (Explorer) displays the project structure under 'PARKVISTA'. The 'trigger' folder is selected, showing files like 'ReservationHandler.cls', 'ReservationHandlerTest.cls', 'ReservationHandlerTest.cls-meta.xml', 'ReservationNotificationQueueable.cls', 'ReservationNotificationQueueable.cls-meta.xml', 'ReservationResetScheduler.cls', 'ReservationResetScheduler.cls-meta.xml', 'ReservationUtils.cls', 'ReservationUtils.cls-meta.xml', 'UpdateSlotBatch.cls', and 'UpdateSlotBatch.cls-meta.xml'. The right pane (Editor) shows the code for 'UpdateSlotBatch.cls'. The code defines a class 'UpdateSlotBatch' that implements the 'Database.Batchable<SObject>' interface. It includes methods for 'start', 'execute', and 'finish'. The 'start' method queries parking lots and initializes their available slots. The 'execute' method iterates over the lots and updates their availability based on their total slots. The 'finish' method is empty.

```

trigger
  ReservationTrigger.trigger-meta.xml
  ReservationNotificationQueueable.cls
  UpdateSlotBatch.cls
  ReservationUtils.cls

global class UpdateSlotBatch implements Database.Batchable<SObject> {
    global Database.QueryLocator start(Database.BatchableContext BC){
        return Database.getQueryLocator(['SELECT Id, Total_Slots__c FROM Parking_Lot__c']);
    }
    global void execute(Database.BatchableContext BC, List<Parking_Lot__c> scope){
        for(Parking_Lot__c lot : scope){
            lot.Available_Spots__c = lot.Total_Slots__c;
        }
        update scope;
    }
    global void finish(Database.BatchableContext BC){}
}

```

6. ReservationResetScheduler

Purpose:

Schedules the batch ([UpdateSlotBatch](#)) to run automatically.

Importance:

Admins don't need to manually reset slots. Ensures lots are always accurate every morning/night.

Workflow:

- You schedule it in Salesforce (e.g., 12 AM daily).
- Scheduler triggers [UpdateSlotBatch](#).
- Batch resets all lots' availability.

Work in app:

Every day at midnight, all lots get reset to full available slots.

The screenshot shows the Salesforce IDE interface. On the left, the Project Explorer displays the project structure under 'PARKVISTA'. It includes folders for .sf, sfdo, config, force-app/main/default (applications, aura, classes, triggers), and various XML files like ReservationHandler.cls-meta.xml and ReservationUtils.cls-meta.xml. In the center, the Editor tab shows the Apex class 'ReservationResetScheduler.cls' with the following code:

```
1 /**
2  * @description      :
3  * @author          : ChangeMeIn@UserSettingsUnder.SFDoc
4  * @group           :
5  * @last modified on : 09-29-2025
6  * @last modified by : ChangeMeIn@UserSettingsUnder.SFDoc
7 */
8 global class ReservationResetScheduler implements Schedulable {
9     global void execute(SchedulableContext sc) {
10         UpdateSlotBatch batch = new UpdateSlotBatch();
11         Database.executeBatch(batch);
12     }
13 }
14
```

Conclusion(Phase 5):

The implemented Apex classes and triggers effectively demonstrate core Apex programming concepts, including object-oriented programming, trigger handling with design patterns, SOQL queries, collections, control statements, batch and queueable asynchronous processing, scheduled jobs, and test classes. Thus, they comprehensively cover the majority of Phase 5 Apex Developer requirements.

Phase 6: User Interface Development

Lightning App Builder

Purpose: To create custom pages for apps, record pages, and home pages without code.

Key Points:

- Drag-and-drop components to design UI.
 - Create **App Pages**, **Home Pages**, and **Record Pages**.
 - Use **Dynamic Visibility** to show/hide components based on criteria.
 - Components can be **Standard**, **Custom**, or **Third-party**.

Steps:

1. Go to **Setup** → **Lightning App Builder** → New.
 2. Choose the type of page: **App Page / Home Page / Record Page**.
 3. Drag components from the left pane onto the layout.
 4. Save and **Activate** the page for users or profiles.

The screenshot shows the Salesforce Setup page with the 'Object Manager' tab selected. The main content area is titled 'Lightning App Builder' and displays a list of 'Lightning Pages'. The table has columns for Action, Label, Name, Namespace Prefix, Description, Type, Created By, and Last Modified By. The data is as follows:

Action	Label	Name	Namespace Prefix	Description	Type	Created By	Last Modified By
Edit Clone Del	Booking_Record_Page	Booking_Record_Page			Record Page	utk. 9/27/2025, 3:05 AM	utk. 9/27/2025, 3:05 AM
Edit Clone Del	parking_lot_record_page	parking_lot_record_page			Record Page	utk. 9/27/2025, 3:15 AM	utk. 9/27/2025, 3:15 AM
Edit Clone Del	parking_spot_record_page	parking_spot_record_page			Record Page	utk. 9/27/2025, 3:23 AM	utk. 9/29/2025, 12:24 AM
Edit Clone Del	reservation_record_page	reservation_record_page			Record Page	utk. 9/27/2025, 3:12 AM	utk. 9/29/2025, 12:22 AM

Record Pages

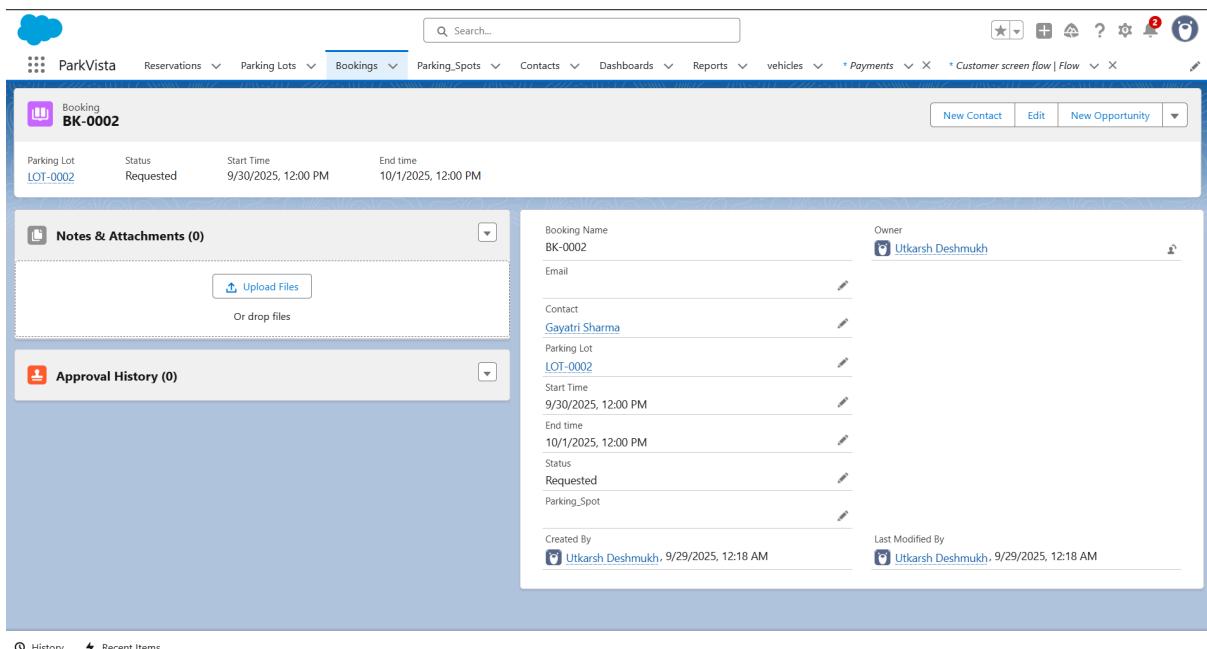
Purpose: Customize the layout and UI for individual objects' records.

Key Points:

- Can override default Salesforce record pages.
- Allows embedding **Related Lists**, **Components**, and **Custom LWC**.
- Assign pages by **App, Record Type, and Profile**.

Steps:

1. Open **Lightning App Builder** → **Record Page** → **New**.
2. Select the **Object** (e.g., Vehicle, Booking).
3. Choose a layout template.
4. Add required components.
5. Activate the page for specific apps, profiles, or as default.



Tabs

Description: Tabs are navigation items that allow users to access objects, Visualforce pages, or Lightning components quickly. They make it easier for Operators and Admins to move across key modules.

Smart Parking Implementation:

- **Custom Object Tabs Created:**
 - Parking Lot → Manage parking lots.
 - Parking Space → Manage individual slots.
 - Booking → View/create reservations.

- Payment → Track transactions.
- Vehicle → Register/manage vehicles.

The screenshot shows the Salesforce Setup interface with the 'Tabs' page selected. The left sidebar has sections for 'User Interface' and 'Tabs'. The main content area is titled 'Custom Tabs' and contains four sections: 'Custom Object Tabs', 'Web Tabs', 'Visualforce Tabs', and 'Lightning Component Tabs'. Each section lists tabs with their labels, edit/delete links, and tab styles (e.g., Books, Car, Wrench, Factory).

Action	Label	Tab Style	Description
Edit Del	Bookings	Books	
Edit Del	Parking_Spots	Car	
Edit Del	Parking_Lots	Wrench	
Edit Del	Reservations	Factory	
Edit Del	vehicles	Car	

Home Page Layouts

Description: Home Page Layouts (via Lightning App Builder) display dashboards, key metrics, and shortcuts. They are profile-based and can differ for Admins vs Operators.

Smart Parking Implementation:

- **Admin Home Page:**
 - Dashboard → Total revenue today, monthly revenue trends.
 - List View → Recent Payments, Failed Payments.
 - Report Chart → Occupancy by Lot.
- **Operator Home Page:**
 - List View → Today's upcoming bookings.
 - Metrics → Spaces available right now.
 - Shortcut → “Quick Allocate” button.

Utility Bar

Description: The Utility Bar provides always-available tools at the bottom of the Lightning UI.

Smart Parking Implementation:

- **Quick Allocate:** Opens a Screen Flow → lets operators allocate space instantly for a walk-in booking.
- **Live Occupancy:** Custom LWC showing available/reserved/occupied slots.
- **Notifications:** Displays system alerts like “Payment Failed for Booking #1234”.

LWC (Lightning Web Components)

1. Overview

The “HomeParking” LWC displays **available parking lots** to users, allows them to **book a parking spot**, and navigates them to the reservation record page. It leverages **Apex methods**, **Wire adapters**, **Imperative calls**, **Events**, and **Navigation Service**.

2. Features Covered

- **LWC (Lightning Web Components):** Component structure with HTML, JS, and CSS.
- **Apex with LWC:** Uses @AuraEnabled Apex methods getParkingLots(), bookReservation(), getContactId().
- **Events in LWC:** Uses ShowToastEvent to notify success/error.
- **Wire Adapters:** @wire(getParkingLots) fetches active parking lots reactively.
- **Imperative Apex Calls:** bookReservation() and getContactId() are called imperatively in JS.
- **Navigation Service:** Uses NavigationMixin to redirect users to the reservation record page after booking.

3. Workflow

1. **Load Home Page:**

- LWC HomeParking uses @wire(getParkingLots) to fetch all active parking lots.
- Displays Lot_Name__c, Available_Spots__c, and Price_Per_Hour__c.

2. Fetch Current User Contact:

- connectedCallback() calls getContactId(USER_ID) imperatively to fetch the contact linked to logged-in user.

3. Book Parking Spot:

- Clicking the “Book” button triggers handleBook().
- Calls bookReservation(lotId, contactId) imperatively.
- Shows **success or error toast** using ShowToastEvent.

4. Navigate to Reservation:

- After successful booking, uses NavigationMixin.Navigate to open the reservation record page.

4. Apex Controller: ParkingLotController.cls

```

ParkingLotController.cls • JS homeParking.js | homeParking.html | Extension: Codex – OpenAI's coding agent | homeParking.js-meta.xml | parkingAvailability.js-meta.xml
force-app > main > default > classes > ParkingLotController.cls > ParkingLotController
1  public with sharing class ParkingLotController {
2
3      @AuraEnabled(cachable=true)
4      public static List<Parking_Lot__c> getParkingLots() {
5          return [
6              SELECT Id, Lot_Name__c, Available_Spots__c, Price_Per_Hour__c, Status__c
7              FROM Parking_Lot__c
8              WHERE Status__c = 'Active'
9          ];
10
11
12      @AuraEnabled
13      public static Reservation__c bookReservation(Id lotId, Id contactId) {
14          if(lotId == null || contactId == null){
15              throw new AuraHandledException('Parking lot or contact Id is missing.');
16          }
17
18          Parking_Lot__c lot = [
19              SELECT Id, Available_Spots__c
20              FROM Parking_Lot__c
21              WHERE Id = :lotId
22              LIMIT 1
23          ];
24
25          if(lot.Available_Spots__c <= 0) {
26              throw new AuraHandledException('No available spots');
27          }
28
29          lot.Available_Spots__c -= 1;
30          update lot;
31
32          Reservation__c res = new Reservation__c(
33              Parking_Lot__c = lotId,
34              Contact__c = contactId,
35              Reservation_Date__c = Date.today(),
36              Start_Time__c = Datetime.now(),
37              End_Time__c = Datetime.now().addHours(2),
38              Payment_Status__c = 'Pending'
39          );
40          insert res;
41
42          // Query the inserted reservation to include Parking_Lot__r.Lot_Name__c
43          res = [SELECT Id, Parking_Lot__r.Lot_Name__c FROM Reservation__c WHERE Id = :res.Id LIMIT 1];
44          return res;
}

```

```

ParkingLotController.cls
1  public with sharing class ParkingLotController {
13    public static Reservation__c bookReservation(Id lotId, Id contactId) {
44      return res;
45    }
46
47    @AuraEnabled(cacheable=true)
48    public static Map<Id, List<String>> getParkingSlotsVisual() {
49      Map<Id, List<String>> lotSlotsMap = new Map<Id, List<String>>();
50      List<Parking_Lot__c> lots = [
51        SELECT Id, Total_Slots__c, Available_Spots__c
52        FROM Parking_Lot__c
53        WHERE Status__c = 'Active'
54      ];
55
56      for(Parking_Lot__c lot : lots){
57        List<String> slots = new List<String>();
58        for(Integer i = 0; i < lot.Total_Slots__c; i++){
59          slots.add(i < lot.Available_Spots__c ? 'A' : 'O'); // Available or Occupied
60        }
61        lotSlotsMap.put(lot.Id, slots);
62      }
63      return lotSlotsMap;
64    }
65
66    @AuraEnabled(cacheable=true)
67    public static Id getContactId(Id userId) {
68      if(userId == null) return null;
69
70      // Step 1: get AccountId from User
71      User u = [SELECT AccountId FROM User WHERE Id = :userId LIMIT 1];
72      if(u.AccountId == null) return null;
73
74      // Step 2: get Contact
75      List<Contact> contacts = [SELECT Id FROM Contact WHERE AccountId = :u.AccountId LIMIT 1];
76      return (contacts.size() > 0) ? contacts[0].Id : null;
77    }
78  }

```

5. LWC: homeParking.html

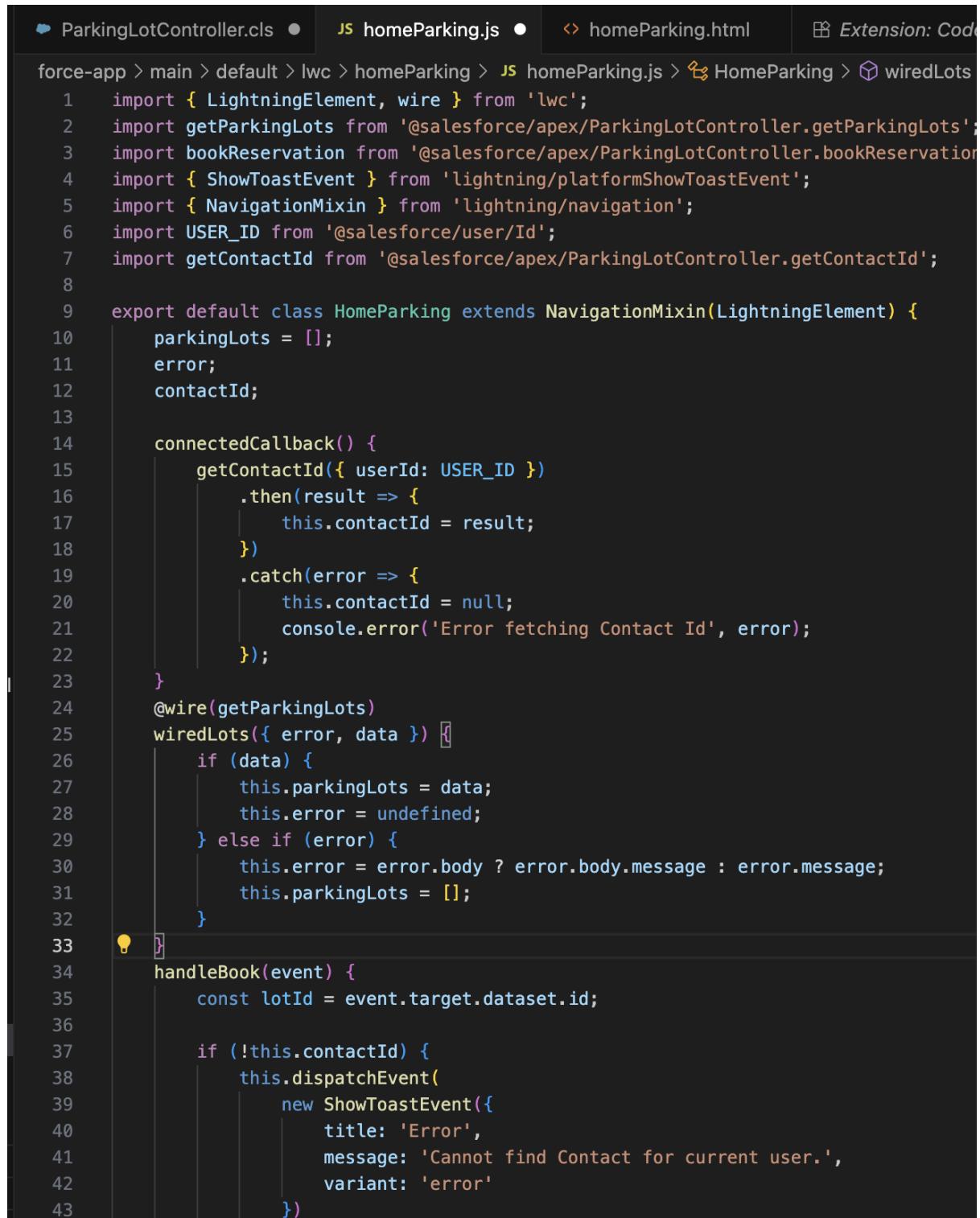
```

EXPLORER
  PARKVISTA
    .husky
    .sf
    .sfdx
    .vscode
    config
    force-app/main/default
      applications
      aura
      classes
        ParkingLotController.cls
        ParkingLotController.cls-meta.xml
        ReservationHandler.cls
        ReservationHandler.cls-meta.xml
        ReservationHandlerTest.cls
        ReservationHandlerTest.cls-meta.xml
        ReservationNotificationQueueable.cls...
        ReservationResetScheduler.cls
        ReservationResetScheduler.cls-meta.xml
        ReservationUtils.cls
        ReservationUtils.cls-meta.xml

force-app > main > default > lwc > homeParking > homeParking.html > template
1  <template>
2    <lightning-card title="Available Parking Lots">
3      <template if:true={parkingLots}>
4        <template for:each={parkingLots} for:item="lot">
5          <div key={lot.Id} class="lot-card">
6            <p><strong>{lot.Lot_Name__c}</strong></p>
7            <p>Available Spots: {lot.Available_Spots__c}</p>
8            <p>Price per Hour: {lot.Price_Per_Hour__c}</p>
9
10           
11           <lightning-button
12             label="Book"
13             data-id={lot.Id}
14             onclick={handleBook}
15             variant="brand">
16             </lightning-button>
17           </div>
18         </template>
19       </template if:true={error}>
20         <p class="slds-text-color_error">{error}</p>
21       </template>
22     </lightning-card>
23   </template>
24 </template>

```

6. LWC: [homeParking.js](#)



```
force-app > main > default > lwc > homeParking > JS homeParking.js ● ↗ homeParking.html Extension: Code > HomeParking > wiredLots
1 import { LightningElement, wire } from 'lwc';
2 import getParkingLots from '@salesforce/apex/ParkingLotController.getParkingLots';
3 import bookReservation from '@salesforce/apex/ParkingLotController.bookReservation';
4 import { ShowToastEvent } from 'lightning/platformShowToastEvent';
5 import { NavigationMixin } from 'lightning/navigation';
6 import USER_ID from '@salesforce/user/Id';
7 import getContactId from '@salesforce/apex/ParkingLotController.getContactId';
8
9 export default class HomeParking extends NavigationMixin(LightningElement) {
10     parkingLots = [];
11     error;
12     contactId;
13
14     connectedCallback() {
15         getContactId({ userId: USER_ID })
16             .then(result => {
17                 this.contactId = result;
18             })
19             .catch(error => {
20                 this.contactId = null;
21                 console.error('Error fetching Contact Id', error);
22             });
23     }
24     @wire(getParkingLots)
25     wiredLots({ error, data }) {
26         if (data) {
27             this.parkingLots = data;
28             this.error = undefined;
29         } else if (error) {
30             this.error = error.body ? error.body.message : error.message;
31             this.parkingLots = [];
32         }
33     }
34     handleBook(event) {
35         const lotId = event.target.dataset.id;
36
37         if (!this.contactId) {
38             this.dispatchEvent(
39                 new ShowToastEvent({
40                     title: 'Error',
41                     message: 'Cannot find Contact for current user.',
42                     variant: 'error'
43                 })
44             );
45     }
46 }
```

```
force-app > main > default > lwc > homeParking > JS homeParking.js > HomeParking > wiredLots
9  export default class HomeParking extends NavigationMixin(LightningElement) {
34    handleBook(event) {
43        ...
44    );
45    return;
46 }
47 bookReservation({ lotId: lotId, contactId: this.contactId })
48     .then(res => {
49         this.dispatchEvent(
50             new ShowToastEvent({
51                 title: 'Success',
52                 message: `Reservation confirmed for ${res.Parking_Lot__r.Lot_Name__c}`,
53                 variant: 'success'
54             })
55         );
56         this[NavigationMixin.Navigate]({
57             type: 'standard__recordPage',
58             attributes: {
59                 recordId: res.Id,
60                 objectApiName: 'Reservation__c',
61                 actionName: 'view'
62             }
63         });
64     })
65     .catch(error => {
66         this.dispatchEvent(
67             new ShowToastEvent({
68                 title: 'Error',
69                 message: error.body ? error.body.message : error.message,
70                 variant: 'error'
71             })
72         );
73     });
74 }
75 }
```

Conclusion (Phase 6):

In this phase, we successfully implemented **Lightning Web Components (LWC)** and integrated them into the **Lightning App Builder**. The custom LWC for the Smart Parking App displays available parking lots, enables booking functionality, and redirects users to reservation details. Through this, we demonstrated the use of **Apex with LWC**, **Events in LWC**, **Wire Adapters**, **Imperative Apex Calls**, and **Navigation Service**. Finally, by placing the component on the Home page via Lightning App Builder, the application became **interactive, user-friendly, and seamlessly accessible to end users**. This phase marks the transition of backend logic into a **real-time, dynamic user interface**, completing the front-end integration for the project.

Phase 7: Integration & External Access

Objective

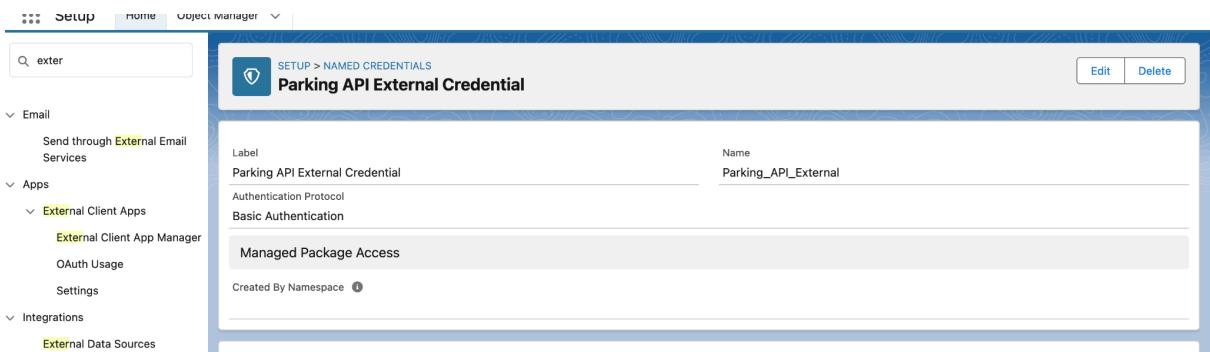
Securely connect Salesforce to an external service (e.g., Parking API) using **Named Credentials** with **Basic Authentication**. This avoids hardcoding credentials and ensures centralized security.

Step 1: Create External Credential

Purpose:

Stores the authentication details (username/password, API key, or OAuth token) securely in Salesforce. Avoids hardcoding credentials in Apex or other integrations and ensures a single source of truth for authentication.

1. Navigate to **Setup** → **External Credentials**.
2. Click **New External Credential**.
3. Fill in details:
 - **Label:** Parking API External Credential
 - **Name:** Parking_API_External
 - **Authentication Protocol:** Basic Authentication
4. Save the External Credential.



Step 2: Assign External Credential to Permission Set

Purpose:

Controls **which users can use the External Credential**. Assigning to a Permission Set ensures only authorized users can perform API callouts using this credential, maintaining security and compliance.

1. Navigate to **Setup → Permission Sets**.
2. Click **New Permission Set** and fill:
 - **Label:** Parking_API_Permission
 - **Name:** Parking_API_Permission
3. Under **External Credential Principal Access**, click **Add Assignment**:
 - Select **Parking API External Credential** created in Step 1.
4. Assign this Permission Set to your user:
 - **Manage Assignments → Add Assignments → Select User → Save**

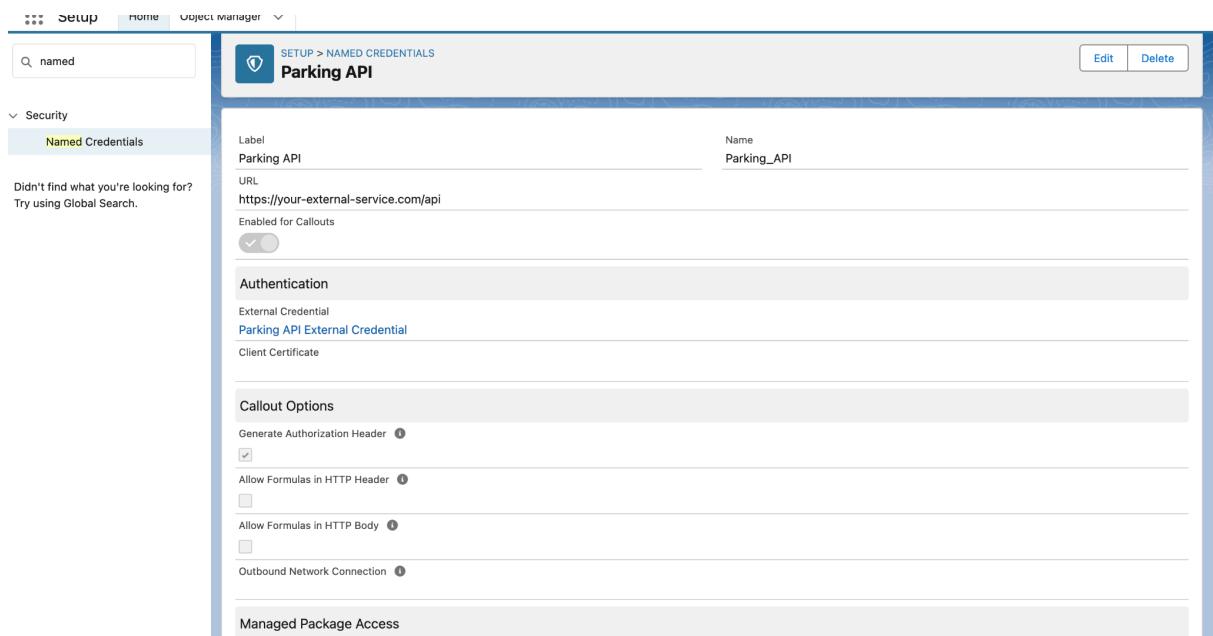
Step 3: Create Named Credential

Purpose:

Provides a **centralized reference** to the external service endpoint along with the authentication method (External Credential). Simplifies Apex, Flow, or External Service integrations because the system automatically handles authentication using the linked External Credential.

1. Navigate to **Setup** → **Named Credentials** → **New**.
2. Fill in details:
 - **Label:** Parking API
 - **Name:** Parking_API
 - **URL:** <https://your-external-service.com/api>
 - **Identity Type:** Named Principal
 - **Authentication Protocol:** External Credential
 - **Select External Credential:** Parking_API_External

3. Save the Named Credential.



Step 4: Remote Site Settings

Purpose:

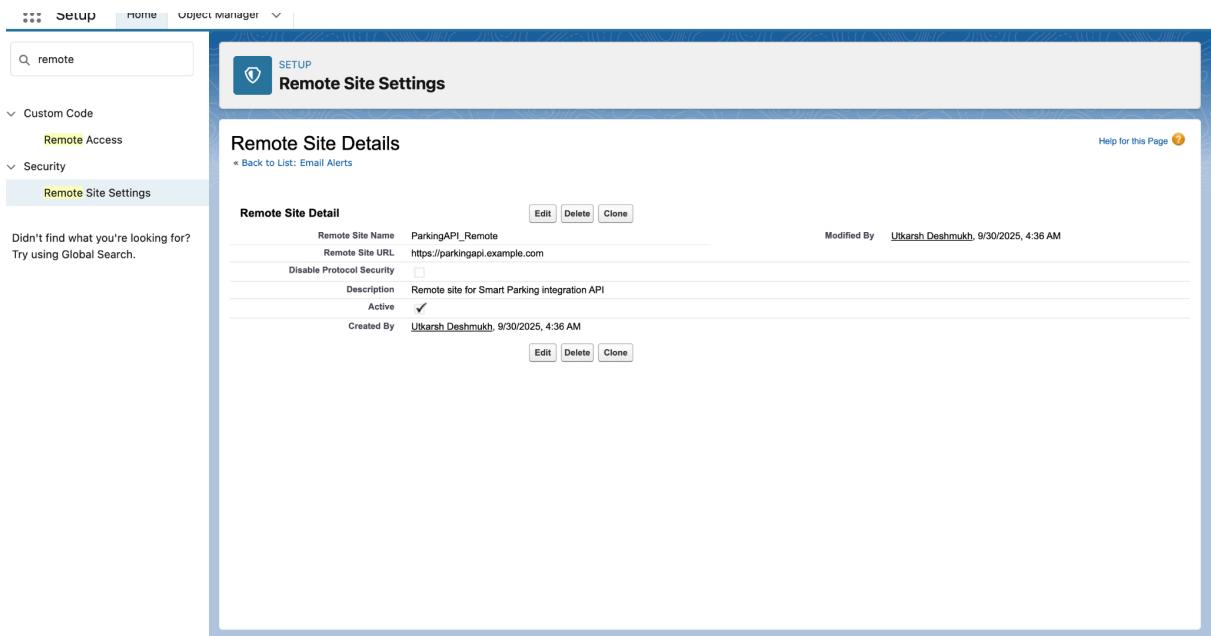
Whitelists the external endpoint in Salesforce. Required in some orgs for security policies so that Salesforce can make callouts to external URLs. Named Credentials sometimes bypass this, but having a Remote Site Setting ensures compatibility and avoids callout errors.

1. Navigate to **Setup** → **Remote Site Settings** → **New Remote Site**.

2. Fill details:

- **Label:** Parking API Remote Site
- **URL:** <https://parkingapi.example.com>

3. Save.



Conclusion – Phase 7: Integration & External Access

In this phase, several key integration and external access features were explored and partially implemented, including Named Credentials, Platform Events, Remote Site Settings, and OAuth authentication. These functionalities provide the foundation for secure communication with external systems, real-time data updates, and API interactions.

Due to the project's current scope, full implementations of some features like Salesforce Connect, External Services, and advanced callouts were deferred. However, the groundwork laid ensures that these components can be expanded and fully integrated in future development phases, aligning with the project's long-term vision for scalable and connected Smart Parking solutions.

Phase 8: Data Management & Deployment

Objective

The objective of this phase is to ensure that all data and metadata of the **Smart Parking Allotment System** are **accurately migrated, maintained, and safeguarded** across environments. This includes:

- Importing and managing parking-related data (Lots, Spaces, Bookings, Vehicles, Payments).
- Preventing duplicate records to maintain data quality.
- Establishing reliable backup mechanisms for business continuity.
- Deploying system changes efficiently from development to production.
- Using modern tools (Change Sets, SFDX) for smooth and scalable deployment.

Purpose

The purpose of this phase is to provide a **structured data management and deployment strategy** that ensures:

1. **Data Accuracy:** Only clean, valid, and non-duplicate records exist (e.g., unique License Plates, valid Payment amounts).
2. **Data Availability:** Backups exist to restore data in case of loss, corruption, or errors.
3. **Deployment Efficiency:** Flows, objects, and automation are moved from Sandbox → Production without errors.
4. **Scalability:** Supports future growth of the Smart Parking system (more lots, more bookings) using bulk tools.
5. **Developer Productivity:** Source-driven development (VS Code + SFDX) enables collaboration and version control.

1. Data Import Wizard

Description:

- A browser-based tool inside Salesforce to import data (up to 50,000 records).
- Supports Accounts, Contacts, Leads, Solutions, Campaign Members, and **all custom objects**.
- Best for **small volumes** and simple imports.

Smart Parking Implementation:

- Imported test Customers (Contacts) with emails.
- Imported Vehicles linked to Customers.
- Imported 20 sample Bookings for flow testing.

The screenshot shows the 'Data Import Wizard' page in the Salesforce setup. At the top, there's a progress bar with three steps: 'Choose data' (blue), 'Edit mapping' (grey), and 'Start import' (grey). Below the progress bar, the title 'Import your Data into Salesforce' is displayed, along with the note 'You can import up to 50,000 records at a time.' and a 'Help for this page' link.

The main area is divided into three sections:

- What kind of data are you importing?**: A table with two tabs: 'Standard objects' and 'Custom objects'. Under 'Custom objects', 'Bookings' is selected and highlighted with a green checkmark.
- What do you want to do?**: A dropdown menu set to 'Add new records' (highlighted with a green checkmark). It includes fields for matching user and contact fields, and parking lot fields.
- Where is your data located?**: A section for uploading a CSV file. It shows a 'CSV' icon, a 'Choose File' button with the path 'booking_im...t_sample.csv', character code 'ISO-8859-1 (General US & Western European, ISO-LATIN-1)', and 'Values Separated By' set to 'Comma'.

At the bottom right, there are 'Cancel', 'Previous', and 'Next' buttons.

2. Data Loader

Description:

- A client application (downloaded) for **large data volumes** (up to millions).

- Supports Insert, Update, Upsert, Delete, Hard Delete, Export, and Export All.
- Requires CSV file mapping.

Smart Parking Implementation:

- Inserted 200 Parking Spots per Lot.
- Updated Parking_Spot__c.Status__c from “Reserved” → “Available” before testing.
- Exported Bookings + Payments to verify revenue totals.

3. Duplicate Rules

Description:

- Prevents or warns about duplicate data.
- Uses **Matching Rules** (exact or fuzzy match).
- Can Block save or Allow with warning.

Smart Parking Implementation:

- Vehicle__c → Prevent duplicate License_Plate__c.
- Contact → Prevent duplicate Email addresses.

The screenshot shows the Salesforce Setup interface for Duplicate Rules. The left sidebar has 'Data' expanded, with 'Duplicate Management' and 'Duplicate Rules' selected. The main content area is titled 'Duplicate Rules' and shows a table of 'All Duplicate Rules'. The table includes columns for Rule Name, Description, Object, Matching Rule, Active status, Last Modified By, and Last Modified Date. The table lists several standard rules, such as 'Booking_Duplicate_Check' for Bookings and 'Standard Account Duplicate Rule' for Accounts. At the bottom of the table, there are navigation links for letters A through Z and an 'All' link.

Rule Name	Description	Object	Matching Rule	Active	Last Modified By	Last Modified Date
Booking_Duplicate_Check	Booking Duplicate_Check	Booking	duplicate bookings	✓	UI5	9/28/2025
Standard Account Duplicate Rule	Identify accounts that duplicate other accounts.	Account	Standard Account Matching Rule	✓	OEPIC	9/12/2025
Standard Contact Duplicate Rule	Identify contacts that duplicate other contacts and leads.	Contact	Standard Contact Matching Rule	✓	OEPIC	9/12/2025
Standard Lead Duplicate Rule	Identify leads that duplicate other leads and contacts.	Lead	Standard Lead Matching Rule	✓	OEPIC	9/12/2025

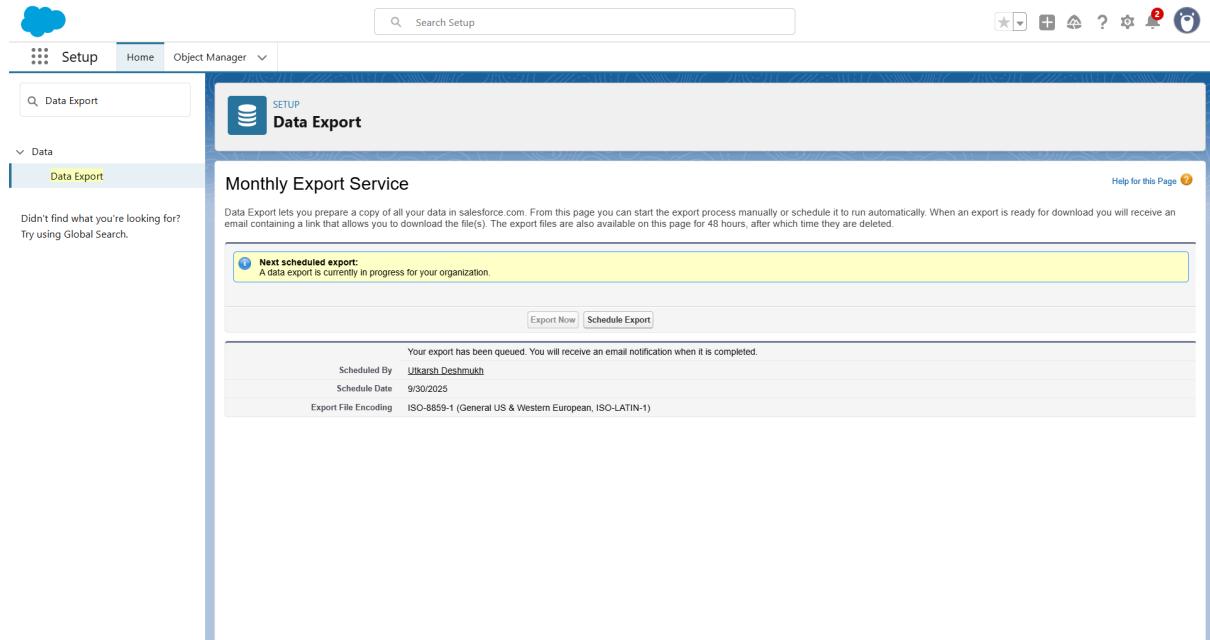
4. Data Export & Backup

Description:

- Ensures data safety by exporting records into CSV files.
- Options: **Data Export Service** (weekly/monthly) or **Data Loader Export** (on-demand or scheduled).
- Can include Salesforce Files, Attachments, and CRM Content.

Smart Parking Implementation:

- Monthly scheduled export for Parking_Lot__c, Parking_Spot__c, Booking__c, Vehicle__c, Payment__c.
- Daily Data Loader export for Bookings & Payments (finance backup).



5. Change Sets

Description:

- Point-and-click tool to deploy metadata between Salesforce orgs (Sandbox → Production).
- Supports objects, fields, flows, email templates, etc.
- Requires manual inbound validation before deployment.

Smart Parking Implementation:

- Outbound Change Set from Sandbox included:
 - Objects: Booking__c, Payment__c.
 - Flows: Booking Allocation, No-Show Scheduler.
 - Email Alerts: Booking Confirmation, Payment Success/Failure.
- Imported and validated in Production.

6. VS Code & SFDX

Description:

- **VS Code:** Modern code editor for Salesforce development.
- **Salesforce DX (SFDX):** CLI for managing source-driven development, scratch orgs, deployments.
- Supports version control (GitHub) + CI/CD pipelines.

Smart Parking Implementation:

- Connected VS Code with Salesforce Org using **SFDX Auth**.
- Pulled metadata (objects, flows, classes) into Git repo.
- Pushed updates from local → Sandbox → Production.
- Wrote Apex classes inside VS Code.

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

- EXPLORER** View: Shows the project structure under **PARKVISTA**. The **classes** folder is currently selected.
- Editor**: A code editor window titled **ReservationNotificationQueueable.cls** is open. The code content is as follows:

```
force-app > main >
1  public class ReservationNotificationQueueable {
2
3      private String reservationId;
4
5      public void handleNotification() {
6          // Implementation
7      }
8
9  }
10 }
11
12 public class ReservationResetScheduler {
13     List<String> reservationIds = new List<String>();
14
15     public void scheduleReset() {
16         // Implementation
17     }
18 }
19
20 public class ReservationUtils {
21
22     public static void updateSlotBatch(List<Slot> slots) {
23         // Implementation
24     }
25
26     public static void updateSlotBatchMeta(List<Slot> slots) {
27         // Implementation
28     }
29
30     public static void updateSlotBatchMeta(List<Slot> slots) {
31         // Implementation
32     }
33
34     public static void updateSlotBatchMeta(List<Slot> slots) {
35         // Implementation
36     }
37 }
```

- OUTLINE**, **TIMELINE**, and **RUNNING TASKS** are visible in the bottom navigation bar.

Phase 9: Reporting, Dashboards & Security Review

Objective:

To enable stakeholders to access, analyze, and visualize Smart Parking system data efficiently, while maintaining strict data security and compliance with organizational policies.

Purpose:

This phase ensures that operational and management decisions are data-driven. Reports provide detailed insights into bookings, payments, and parking occupancy. Dashboards aggregate reports into intuitive visualizations. Security measures, including sharing rules, field-level security, session restrictions, IP range limitations, and audit trails, safeguard sensitive information and prevent unauthorized access.

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

Description:

Salesforce Reports allow data to be displayed in multiple formats to facilitate analysis.

Smart Parking Implementation:

- **Tabular Report:** List of all bookings with Customer, Parking Lot, Start/End Time.
- **Summary Report:** Total payments grouped by Parking Lot.
- **Matrix Report:** Occupancy by Lot (Rows = Lot, Columns = Status).
- **Joined Report:** Combine Bookings & Payments to analyze booking revenue.

Report Name	Description	Folder	Created By	Created On	Subscribed
Sample Flow Report: Screen Flows	Which flows run, what's the status of each interview, and how long do users take to complete the screens?	Public Reports	Automated Process	9/12/2025, 5:39 PM	<input type="checkbox"/>
New Bookings with Contact Report		Private Reports	Utkarsh Deshmukh	9/28/2025, 11:57 PM	<input type="checkbox"/>
New Booking History Report		Private Reports	Utkarsh Deshmukh	9/28/2025, 11:48 PM	<input type="checkbox"/>
New Bookings Report		Private Reports	Utkarsh Deshmukh	9/28/2025, 11:41 PM	<input type="checkbox"/>

2. Report Types

Description:

Report Types define which objects and fields are available in a report. Salesforce provides **Standard** and **Custom Report Types**:

- **Standard Report Types:** Predefined, e.g., Accounts, Contacts, Opportunities.
- **Custom Report Types:** Created to include custom objects or relationships not available in standard reports.

Smart Parking Implementation Examples:

- **Bookings with related Parking Spaces:** Track space utilization per booking.
- **Payments with related Bookings:** Analyze revenue trends per booking.
- **Vehicles with related Customers:** Identify repeat customers or high-value vehicles.

Custom Report Type

All Custom Report Types

Label	Name	Description	Category	Created By	Created Date
Orchestration Run Logs Spring '24	flow_orchestration_log_oob_crt_two_four_eight	Find out which orchestration run logs were created a...	Other Repor...	autproc	9/12/2025, 5:39 PM
Orchestration Runs Spring '24	flow_orchestration_run_oob_crt_two_four_eight	Find out which orchestration runs were created.	Other Repor...	autproc	9/12/2025, 5:39 PM
Orchestration Stage Runs Spring '24	flow_orchestration_stage_run_oob_crt_two_four_eight	Find out which orchestration stage runs were created...	Other Repor...	autproc	9/12/2025, 5:39 PM
Orchestration Step Runs Spring '24	flow_orchestration_step_run_oob_crt_two_four_eight	Find out which orchestration step runs were created ...	Other Repor...	autproc	9/12/2025, 5:39 PM
Orchestration Work Items Spring '...	flow_orchestration_work_item_oob_crt_two_four_eig...	Find out which orchestration work items were create...	Other Repor...	autproc	9/12/2025, 5:39 PM
Program Definition Spring '24	Program_Definition_sfdcSESV60	Review your analytics with a program-like structure. S...	Other Repor...	autproc	9/12/2025, 5:39 PM
Program Definition Summer '24	Program_Definition_sfdcSESV61	Review your analytics with a program-like structure. S...	Other Repor...	autproc	9/12/2025, 5:39 PM
Program Item Progress Spring '24	Program_Task_Progress_sfdcSESV60	Report on tasks like exercises, milestones, and outco...	Other Repor...	autproc	9/12/2025, 5:39 PM
Program Item Progress Summer '24	Program_Task_Progress_sfdcSESV61	Report on tasks like exercises, milestones, and outco...	Other Repor...	autproc	9/12/2025, 5:39 PM
Program Progress Spring '24	Program_Progress_sfdcSESV60	Report on program progress. Specific progress on mi...	Other Repor...	autproc	9/12/2025, 5:39 PM
Program Progress Summer '24	Program_Progress_sfdcSESV61	Report on program progress. Specific progress on mi...	Other Repor...	autproc	9/12/2025, 5:39 PM
Screen Flows	screen_flows_prebuilt_crt	Find out which flows get executed and how long user...	Other Repor...	autproc	9/12/2025, 5:39 PM

https://orgfarm-7148c01b00-dev-ed.lightning.force.com/lightning/setup/CustomReportTypeLightning/home

3. Dashboards

Description:

Dashboards consolidate multiple reports into visualizations like charts, graphs, gauges, and tables. They provide at-a-glance insights for different user roles.

Dashboard

ParkVista – Operations Overview

ParkVista – Operations Overview

As of Sep 30, 2025, 7:42 AM (Viewing as Ultkash Deshmukh)

Refresh Edit Subscribe

New Booking History Report

Booking Owner Na...	Booking Booking ...	Status	Field /...	Edit Date
Ultkash Deshmukh	BK-0001	Request	Created	9/27/2025, 1:17 A
Ultkash Deshmukh	BK-0002	Request	Created	9/29/2025, 12:18

View Report (New Booking History Report)

As of Sep 30, 2025, 7:42 AM

New Bookings Report

Booking Booking Name	Booking Created Date	Status	Parking Lot
BK-0001	9/27/2025	Requested	LOT-0001
BK-0002	9/29/2025	Requested	LOT-0002

History Recent Items

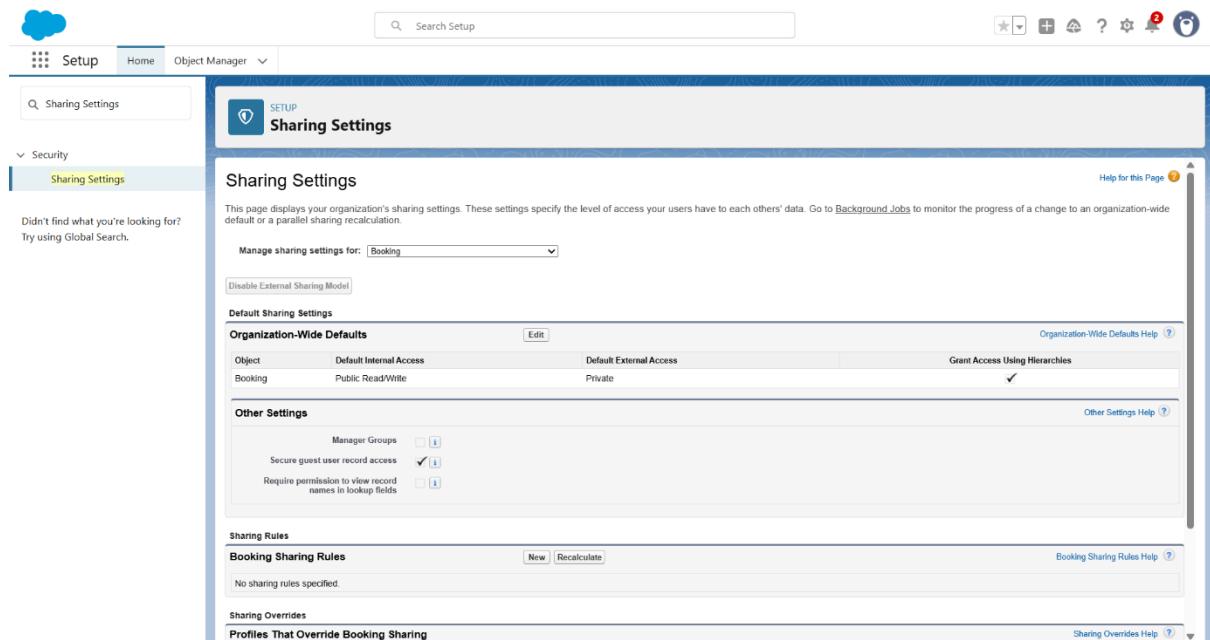
4. Sharing Settings

Description:

Organizational-Wide Defaults (OWD) define baseline visibility of records. Sharing rules and role hierarchies extend access as needed.

Smart Parking Implementation Examples:

- **Bookings:** Private (visible to owner and role hierarchy)
- **Parking Spaces:** Public Read/Write (accessible by operators)
- **Payments:** Private (Admin only)



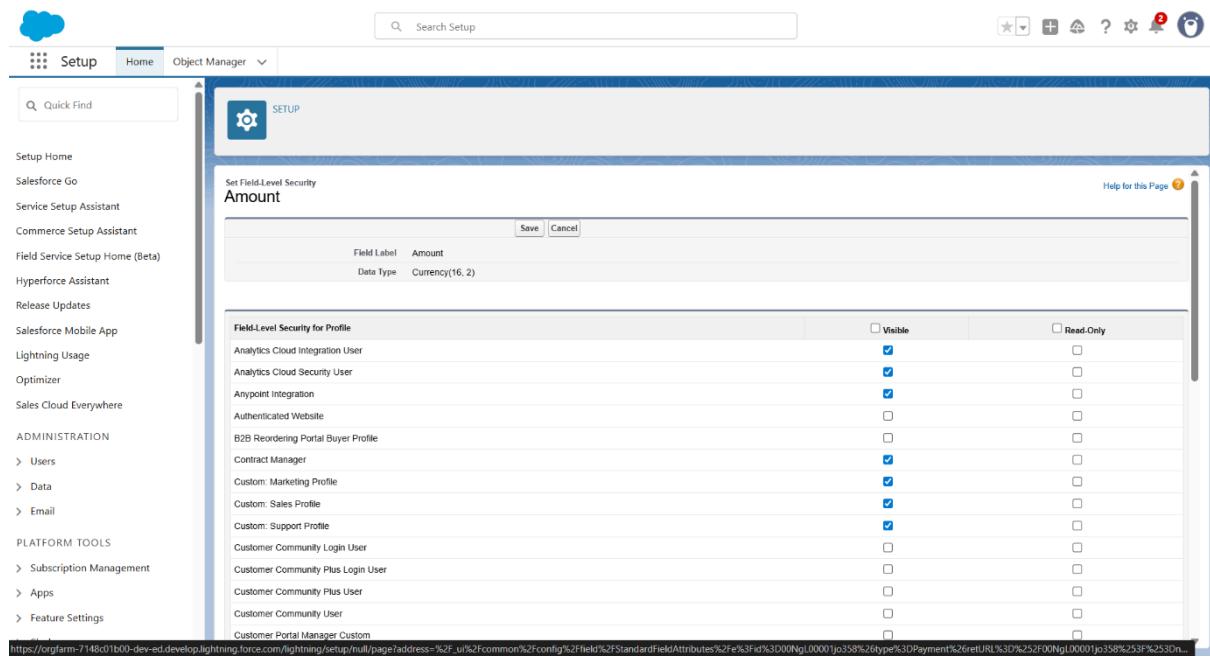
5. Field-Level Security (FLS)

Description:

FLS controls visibility and edit permissions of individual fields for each profile and permission set.

Smart Parking Implementation Examples:

- Hide Payment__c.Card_Number__c from Operators.
- Show Booking__c.Customer__c only to Operators/Admins.
- Admin has full access to all fields.



6. Session Settings

Description:

Session Settings define security policies such as session timeouts, login restrictions, and concurrent login limits.

Smart Parking Implementation Examples:

- Session timeout: 2 hours
- Require re-login after browser inactivity
- Prevent concurrent logins from multiple devices for Operators

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

- Header:** Includes a cloud icon, "Setup" button, "Home" link, "Object Manager" dropdown, a search bar with placeholder "Search Setup", and various global navigation icons.
- Left Sidebar:** Shows a search bar with "Q session" and a navigation tree with "Security" expanded, showing "Session Management" and "Session Settings". Below it, a message says "Didn't find what you're looking for? Try using Global Search."
- Main Content Area:**
 - Section Header:** "Session Settings" with a "SETUP" icon.
 - Section:** "Session Settings" with the sub-instruction "Set the session security and session expiration timeout for your organization."
 - Session Timeout:** Set to "2 hours".
 - 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
AS OF DECEMBER 31, THE EXTENDED PERIOD HAS ENDED, AND USE OF INTERNET EXPLORER 11 (IE 11) WITH LIGHTNING EXPERIENCE IS NO LONGER SUPPORTED. ISSUES WITH PERFORMANCE OR FUNCTIONALITY THAT AFFECT ONLY IE 11 WILL NOT BE FIXED. PLEASE SWITCH TO A SUPPORTED BROWSER.
 - Caching:**
 - Enable caching and autocomplete on login page
 - Enable secure and persistent browser caching to improve performance
 - Enable user switching

7. Login IP Ranges

Description:

Restrict logins to trusted IP addresses to enhance security.

Smart Parking Implementation Examples:

- Operators restricted to office network IP range.
- Admins allowed from all IPs for flexibility in remote management.

Phase 10: Final Presentation & Demo

Demo Video Link-

 PARKVISTASALESFORCECRM.mov