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.

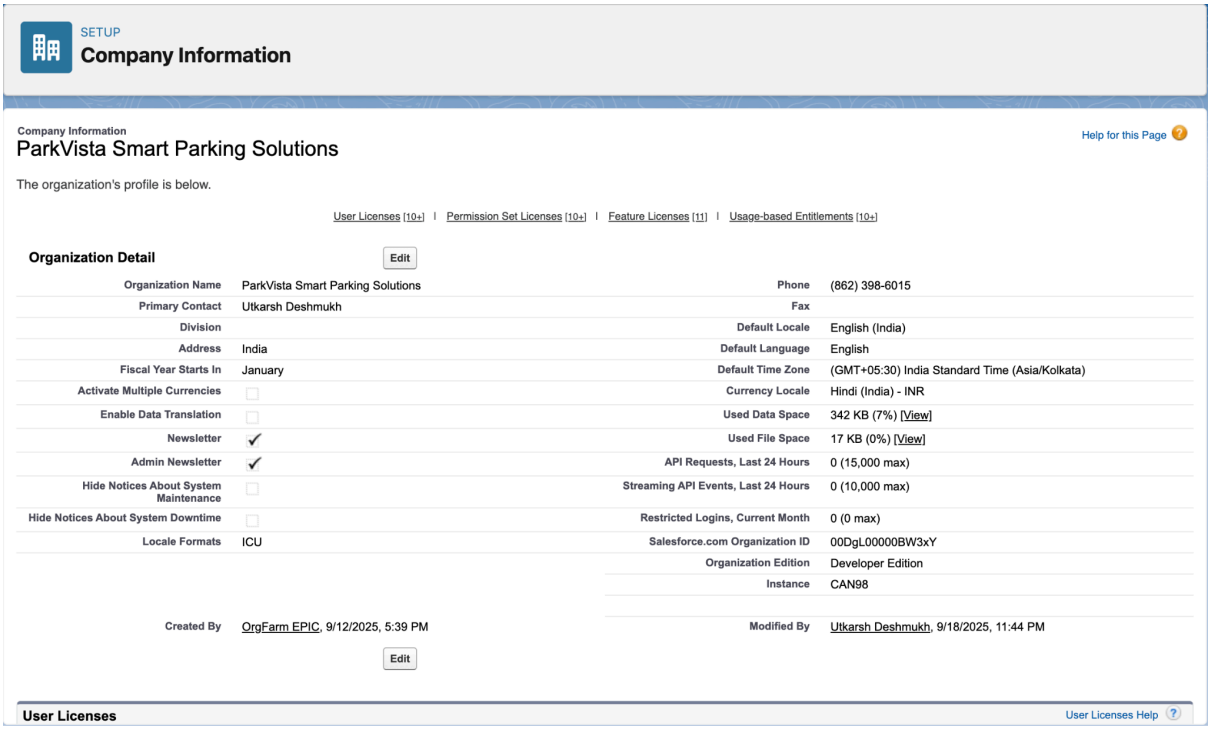
Phase 2: Org Setup and Configuration

Company Profile Setup -

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

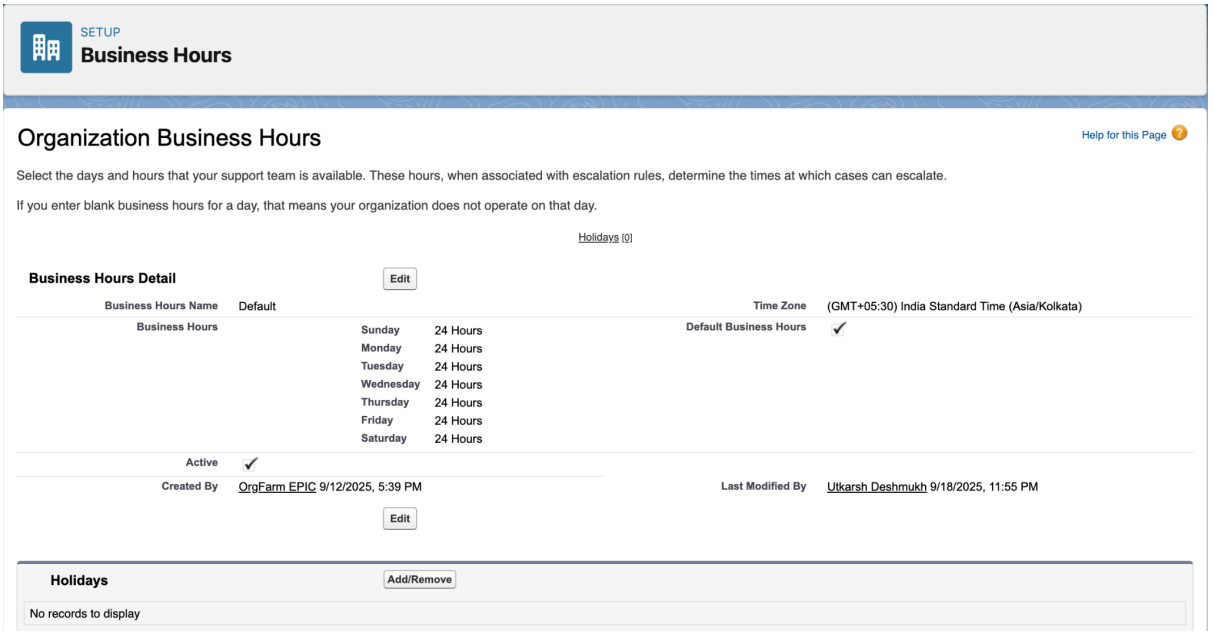
- **Name**: ParkVista Smart Parking Solutions – Dev
- **Time Zone**: (GMT+05:30) Asia/Kolkata
- **Locale**: English (India) → Controls date, time, and number formats
- **Language**: English → Default UI language for all new users

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



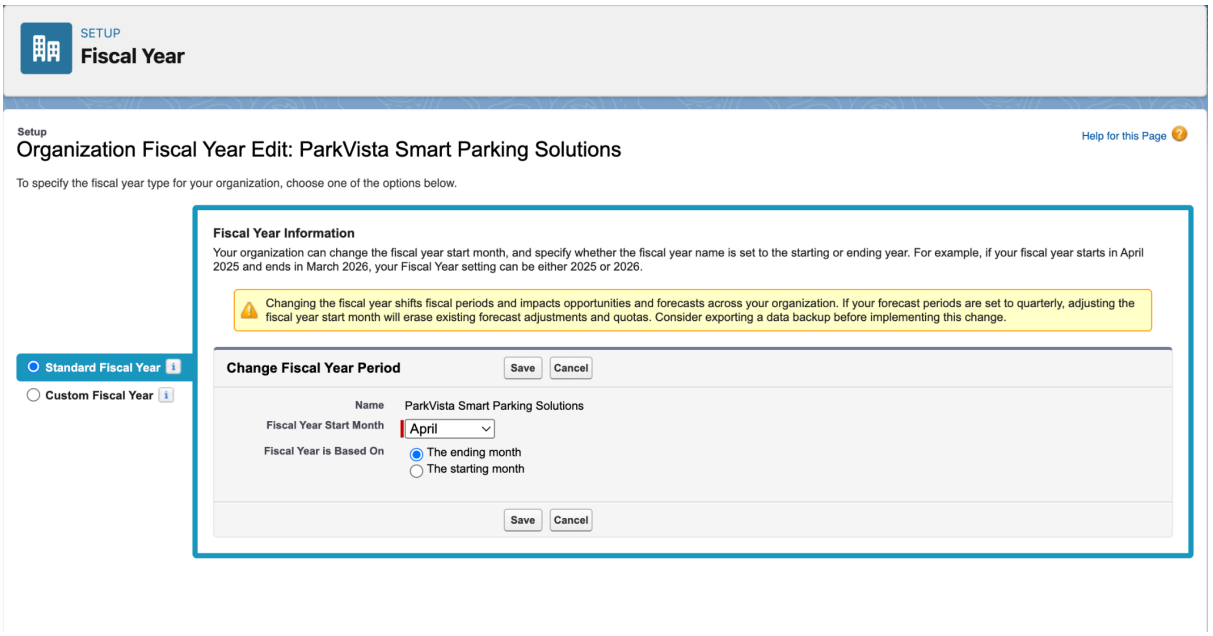
Business Hours & Holidays

- **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



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.

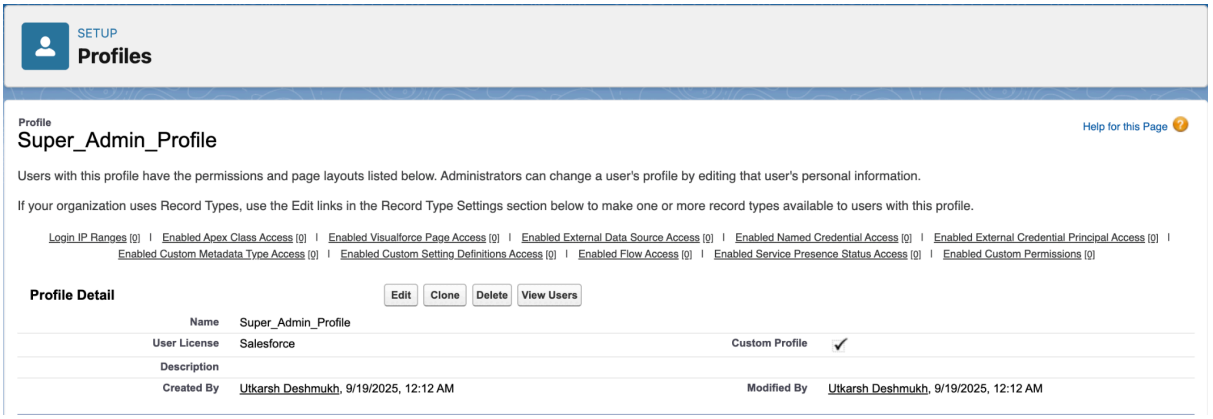


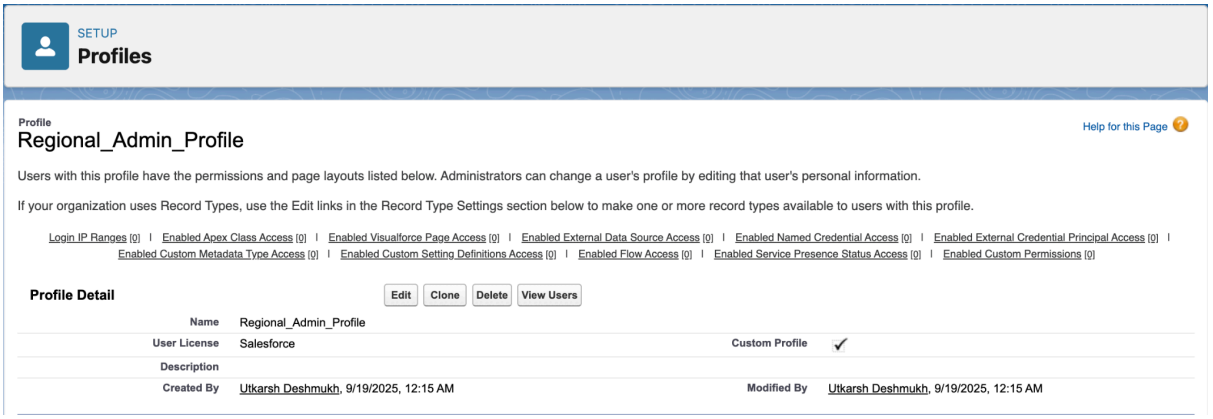
User Setup (Profiles, Roles, Permission Sets, Users)

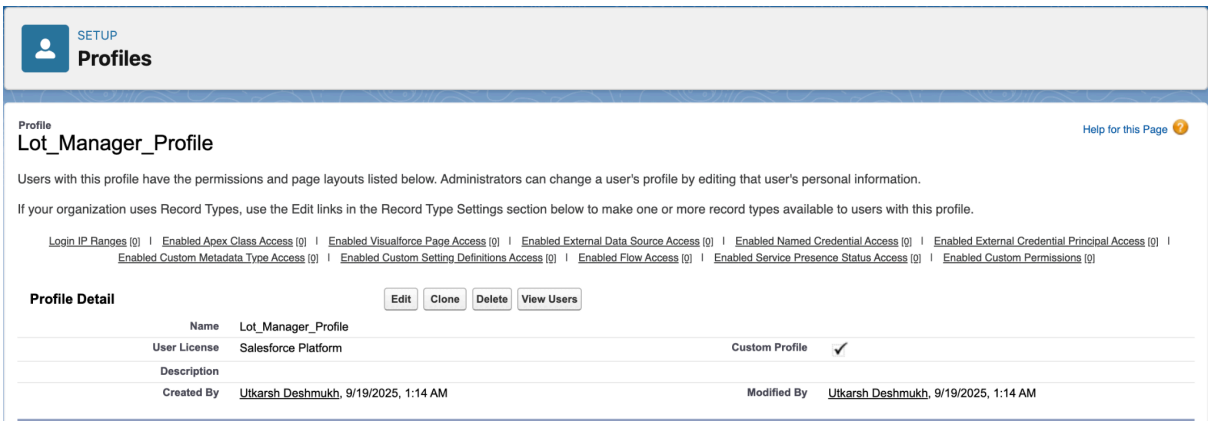
Profiles:

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.







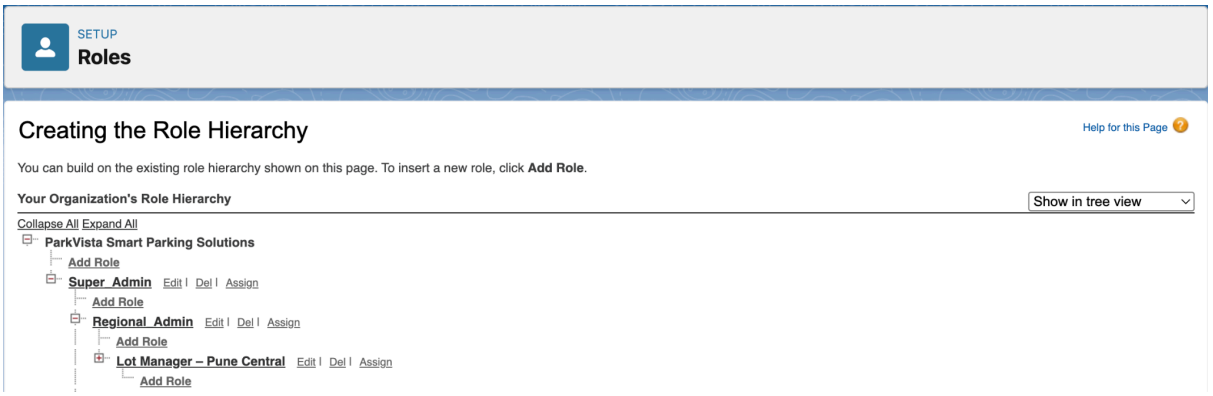
Roles:

Defined a role hierarchy to reflect organizational structure:

→Super Admin (top-level)

→ Regional Manager (reports to Super Admin)

→ Lot Manager (reports to Regional Manager)



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.**

SETUP
Users

User
Super Admin

User ProfileHelp for this Page

Permission Set Assignments [0] | Permission Set Assignments: Activation Required [0] | Permission Set Group Assignments [0] | Permission Set License Assignments [0] | Personal Groups [0] | Public Group Membership [0] | Queue Membership [0] | Team [0] | Managers in the Role Hierarchy [0] | OAuth Apps [0] | Third-Party Account Links [0] | Built-in Authenticators [0] | Installed Mobile Apps [0] | Authentication Settings for External Systems [0] | Login History [0+] | User Provisioning Accounts [0]

User Detail Edit | Sharing | Reset Password | Freeze | View Summary

Name	Super Admin	Role	Super_Admin
Alias	sadmin	User License	Salesforce
Email	ukarshrushh@gmail.com [Verify]	Profile	Super_Admin_Profile
Username	ukarshrushh@gmail.com	Active	✓
Nickname	SuperAdmin	Marketing User	
Title	Super Admin	Offline User	
Company	ParkVista	Knowledge User	
Department		Flow User	
Division		Service Cloud User	
Address	India	Site.com Contributor User	
Time Zone	(GMT+05:30) India Standard Time (Asia/Kolkata)	Site.com Publisher User	
Locale	English (India)	WDC User	
Language	English	Mobile Push Registrations	View
Delegated Approver		Data.com User Type	
Manager		Accessibility Mode (Classic Only)	
Receive Approval Request Emails	Only if I am an approver	Debug Mode	
Federation ID		High-Contrast Palette on Charts	
App Registration: One-Time Password Authenticator		Load Lightning Pages While Scrolling	✓
App Registration: Salesforce Authenticator		Send Apex Warning Emails	

SETUP
Users

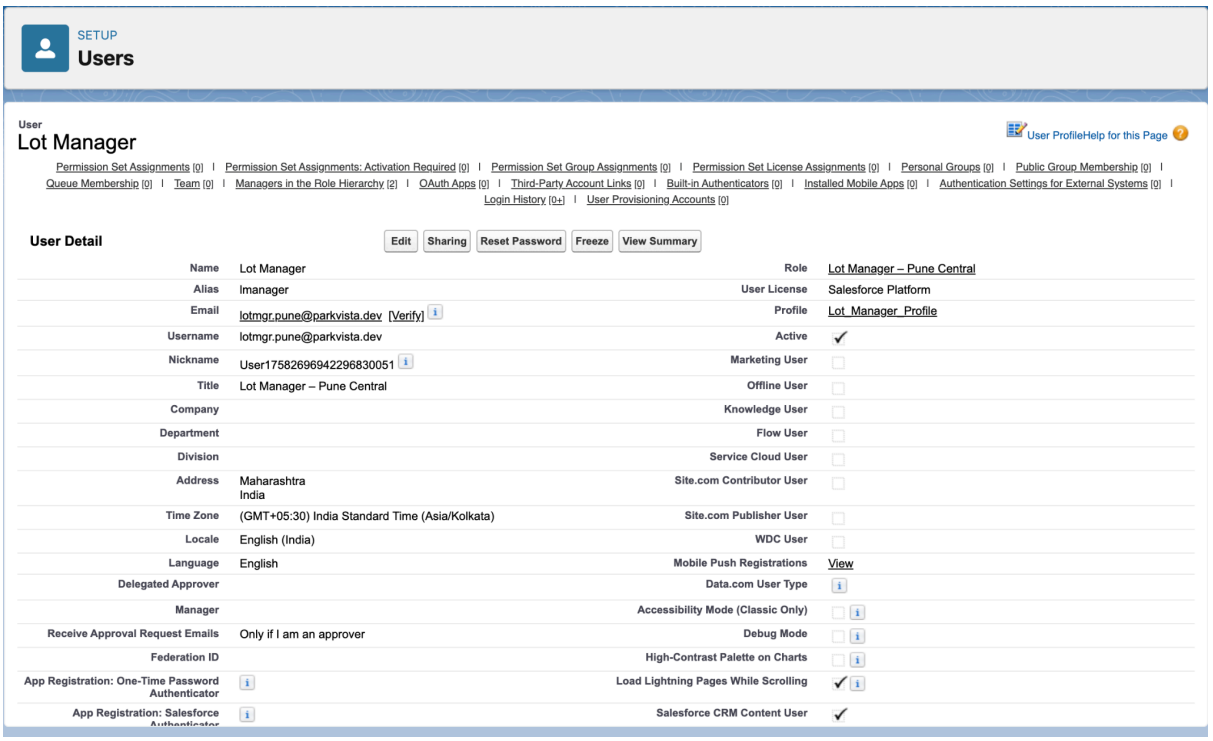
User
Regional Admin

User ProfileHelp for this Page

Permission Set Assignments [0] | Permission Set Assignments: Activation Required [0] | Permission Set Group Assignments [0] | Permission Set License Assignments [0] | Personal Groups [0] | Public Group Membership [0] | Queue Membership [0] | Team [0] | Managers in the Role Hierarchy [1] | OAuth Apps [0] | Third-Party Account Links [0] | Built-in Authenticators [0] | Installed Mobile Apps [0] | Authentication Settings for External Systems [0] | Login History [0+] | User Provisioning Accounts [0]

User Detail Edit | Sharing | Reset Password | Freeze | View Summary

Name	Regional Admin	Role	Regional_Admin
Alias	radmin	User License	Salesforce
Email	ukarshrushh@gmail.com [Verify]	Profile	Regional_Admin_Profile
Username	utkarshdeshmukh.tech@gmail.com	Active	✓
Nickname	User17582692431275067735	Marketing User	
Title	Regional Admin – Maharashtra	Offline User	
Company		Knowledge User	
Department		Flow User	
Division		Service Cloud User	
Address	Maharashtra India	Site.com Contributor User	
Time Zone	(GMT+05:30) India Standard Time (Asia/Kolkata)	Site.com Publisher User	
Locale	English (India)	WDC User	
Language	English	Mobile Push Registrations	View
Delegated Approver		Data.com User Type	
Manager		Accessibility Mode (Classic Only)	
Receive Approval Request Emails	Only if I am an approver	Debug Mode	
Federation ID		High-Contrast Palette on Charts	
App Registration: One-Time Password Authenticator		Load Lightning Pages While Scrolling	✓
App Registration: Salesforce Authenticator		Salesforce CRM Content User	✓



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, and Reservations, but no standard CRM setup permissions.

Org-Wide Defaults (OWD)

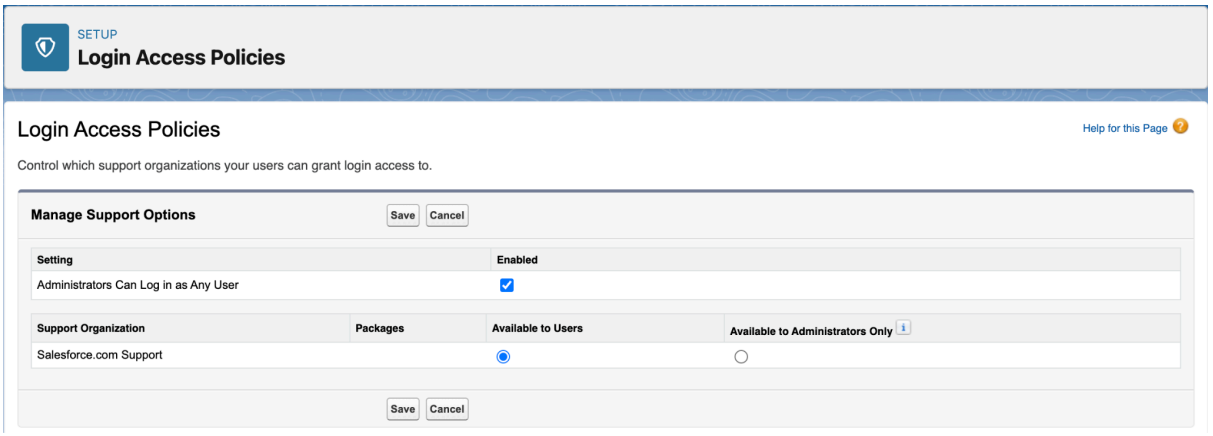
At this stage, OWD is not yet configured as the required custom objects are not available. Once objects like **Parking Lot, Parking Spot, Reservation, and Payment** are created, OWD will be defined to set the baseline record-level access.

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.



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):

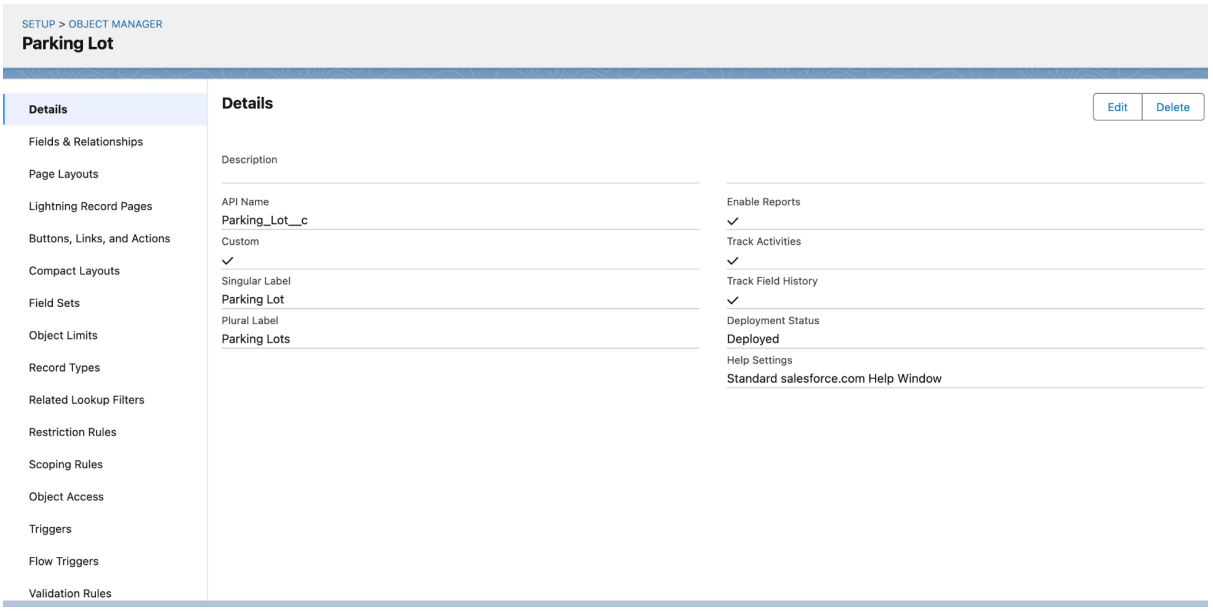
Phase 2 established the **ParkVista Salesforce org** with all essential configurations, including **user roles, profiles, licenses, and login access policies.** These setups provide a **secure and organized** structure, enabling smooth **administration** and **controlled access** for future processes.

Phase 3: Data Modeling & Relationships – ParkVista Smart Parking Solutions

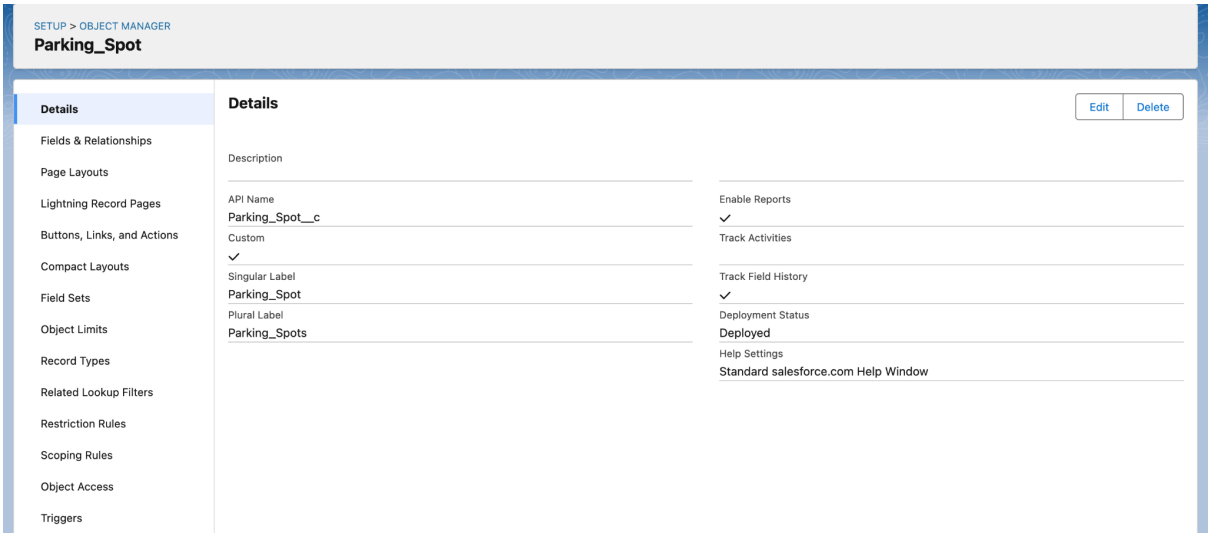
The purpose of this phase was to establish the Salesforce data structure for managing parking lots, parking spots, reservations, and users efficiently. Proper data modeling ensures accurate tracking of availability, bookings, and customer details, forming the foundation for automation in later phases.

1. Standard & Custom Objects

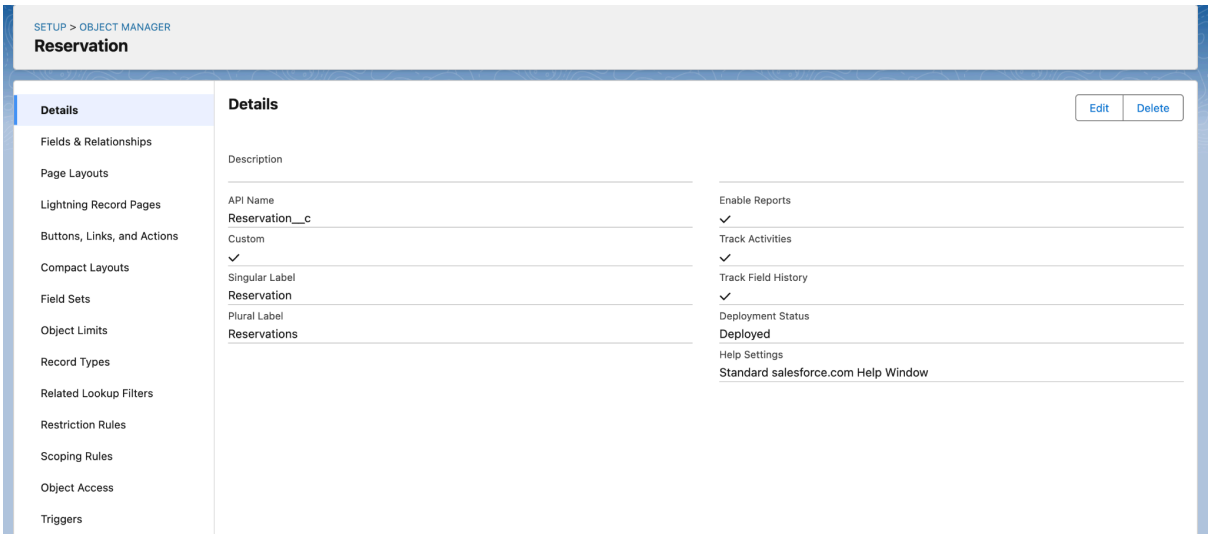
- **Parking Lot : Represents physical parking lots and manages slot inventory.**



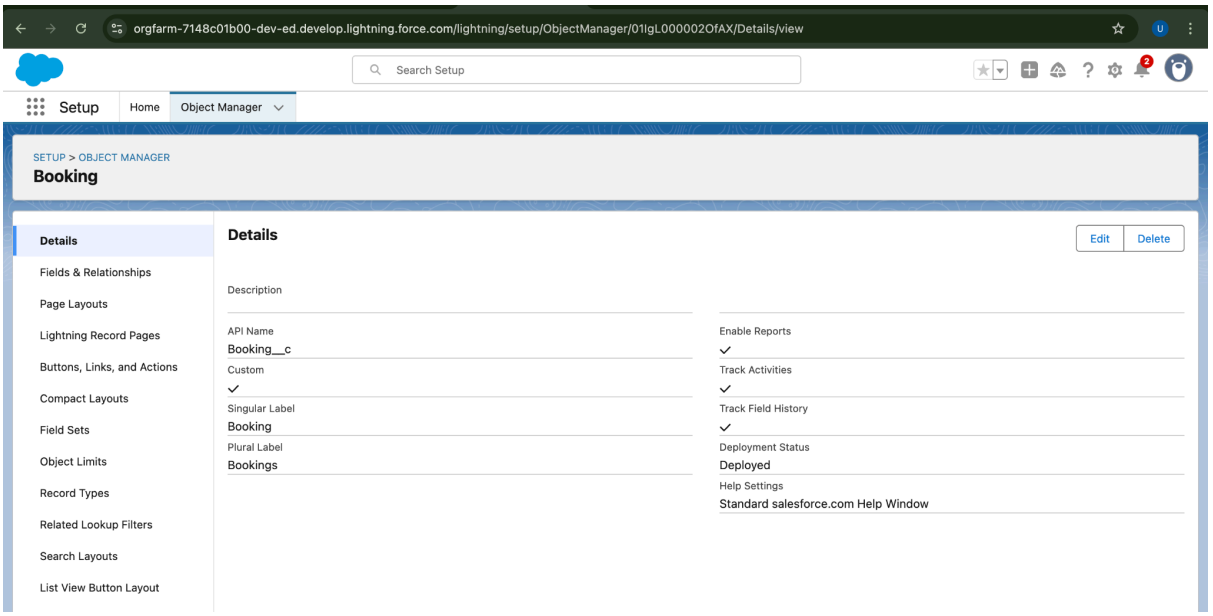
- **Parking Spot : Represents individual parking spots within a lot**



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



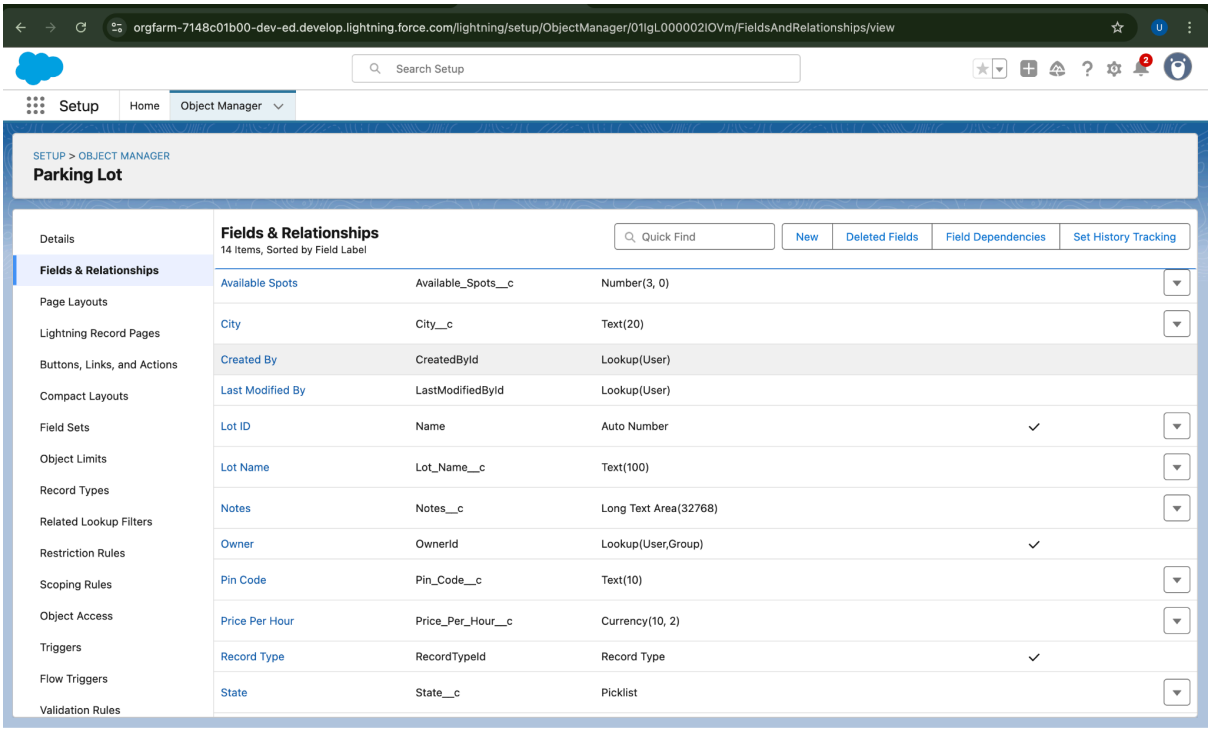
- **Booking Object: Maintains confirmed bookings for reporting and customer reference.**



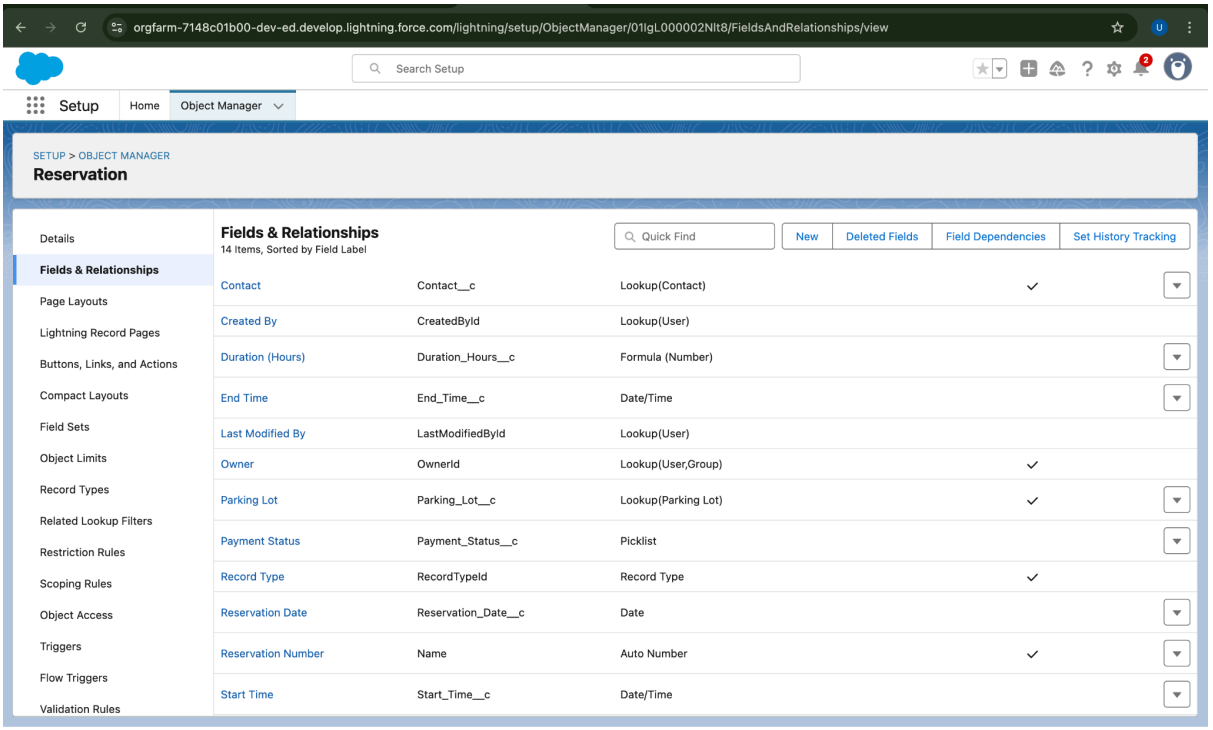
- **Contact (Standard): Represents customers making reservations.**

2.Fields

- **Fields for Parking_Lot__c**



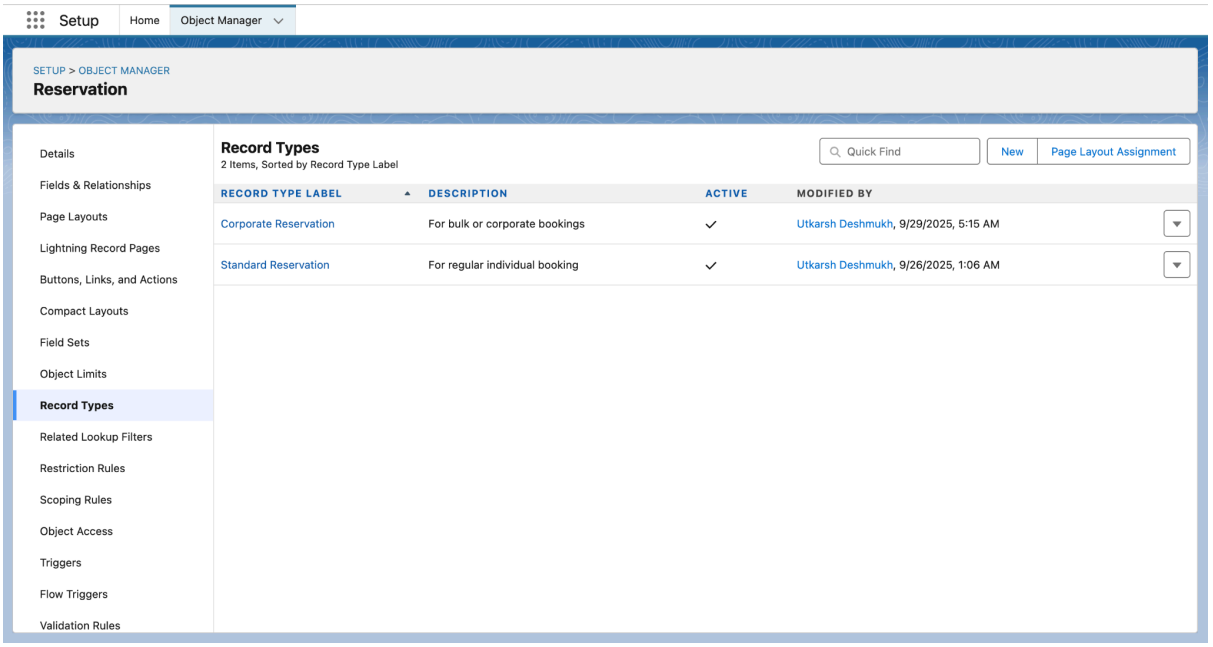
Fields for Reservation__c



3. Record Types

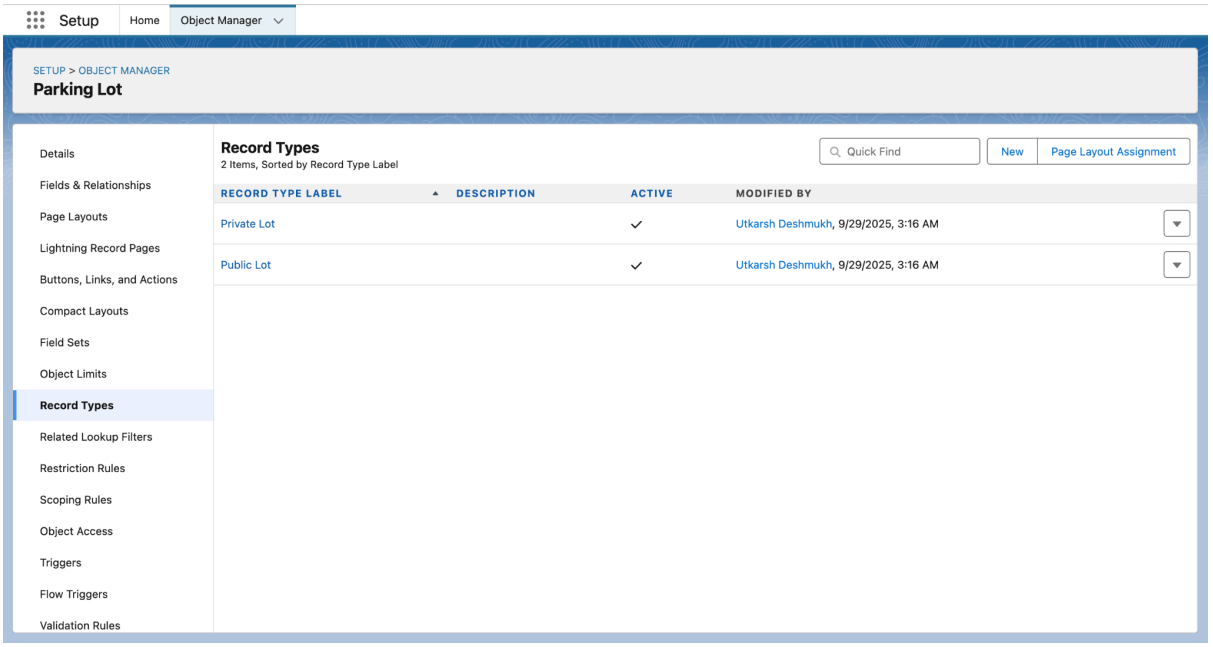
Record Types on Reservation Object:

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

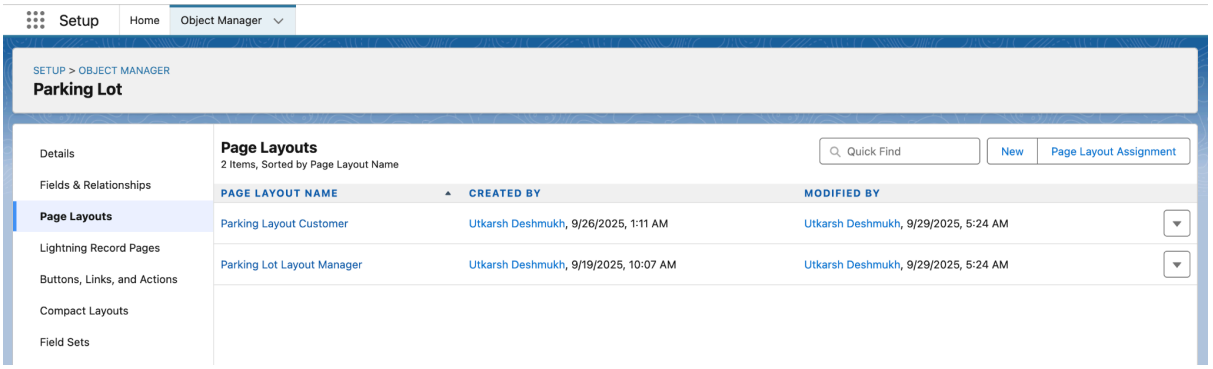


Record Types on Parking Lot Object:

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



4. Page Layout



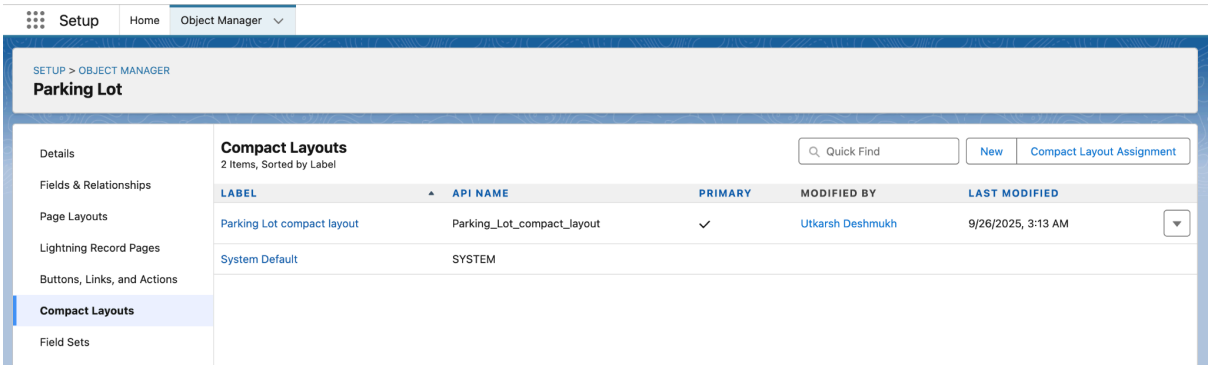
Customer Layout: Shows only booking-relevant fields: Lot Name, Available Spots, Price Per Hour, Status. Hides internal fields like Total Slots, Notes, Owner.

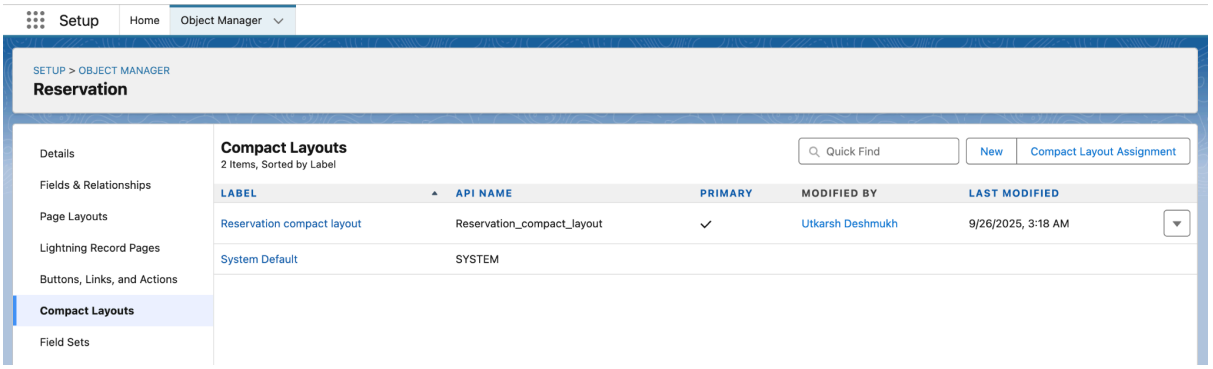
Manager/Admin Layout : Shows all fields for full management: Lot Name, Total Slots, Available Spots, Price Per Hour, Status, Notes, Owner, City, Pin Code, State.

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





5. Lookup vs Master-Detail vs Hierarchical Relationships

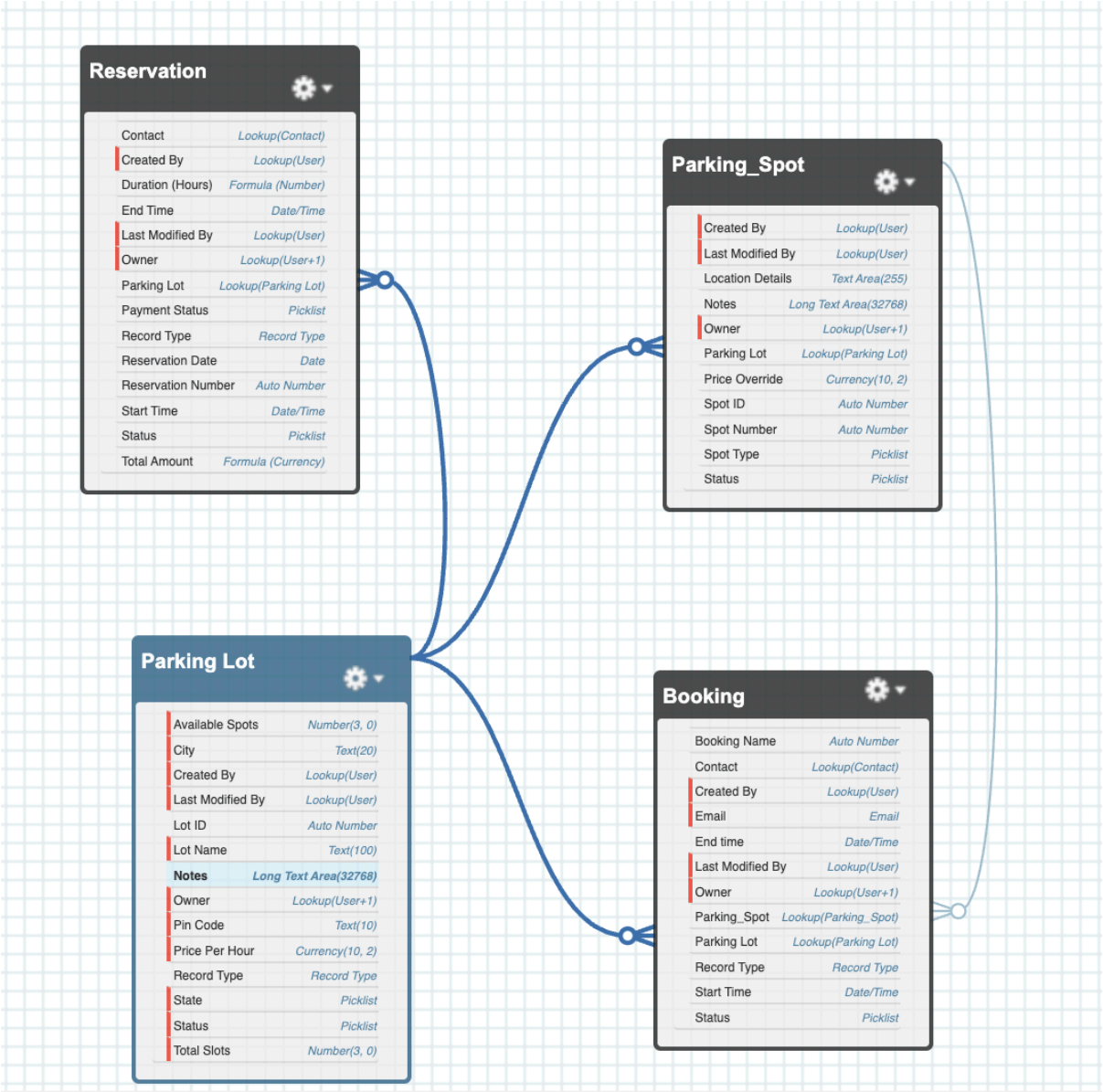
Reservation → Parking Lot: Lookup relationship. Each reservation points to a parking lot (Reservation__c.Parking_Lot__c). This allows a reservation to be linked to a lot without enforcing deletion cascade.

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

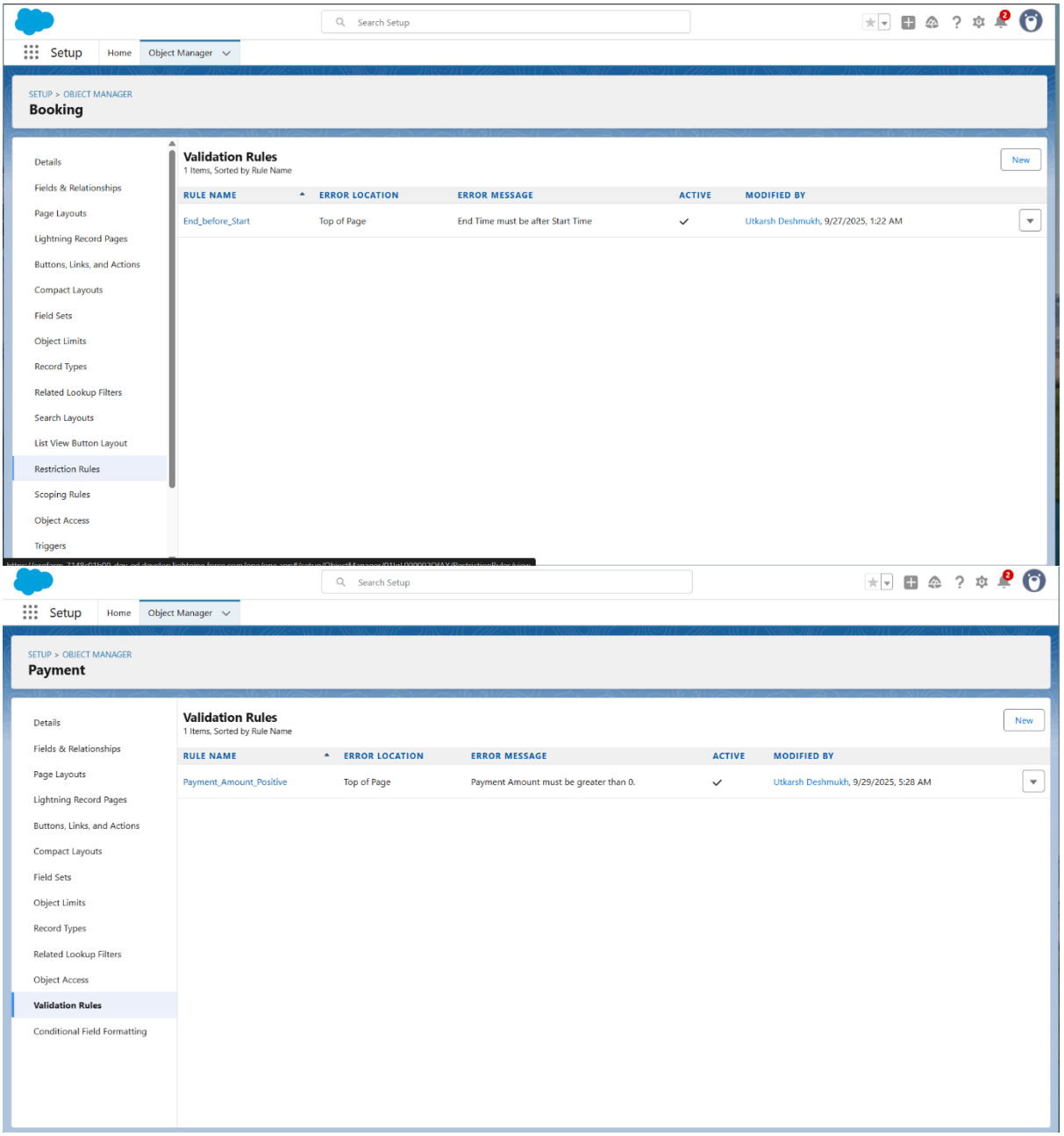
1. Validation Rules

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 End_Time__c ≤ Start_Time__c.
- **Payment Amount Validation:** Ensure Payment_Amount__c > 0.
- **Available Spot Check:** Prevent booking if Available_Spots__c = 0.



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:

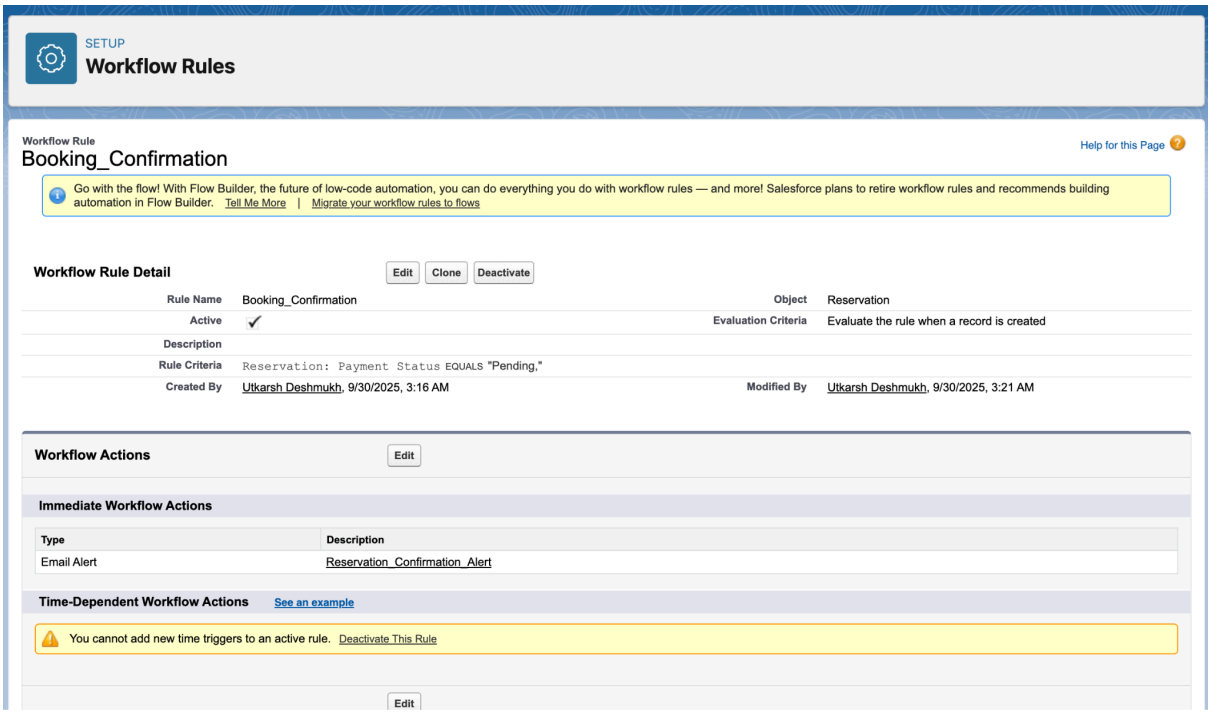
Reservation__c (Custom Object)

Rule Name:

Reservation_Confirmation_Workflow

Rule Criteria:

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



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

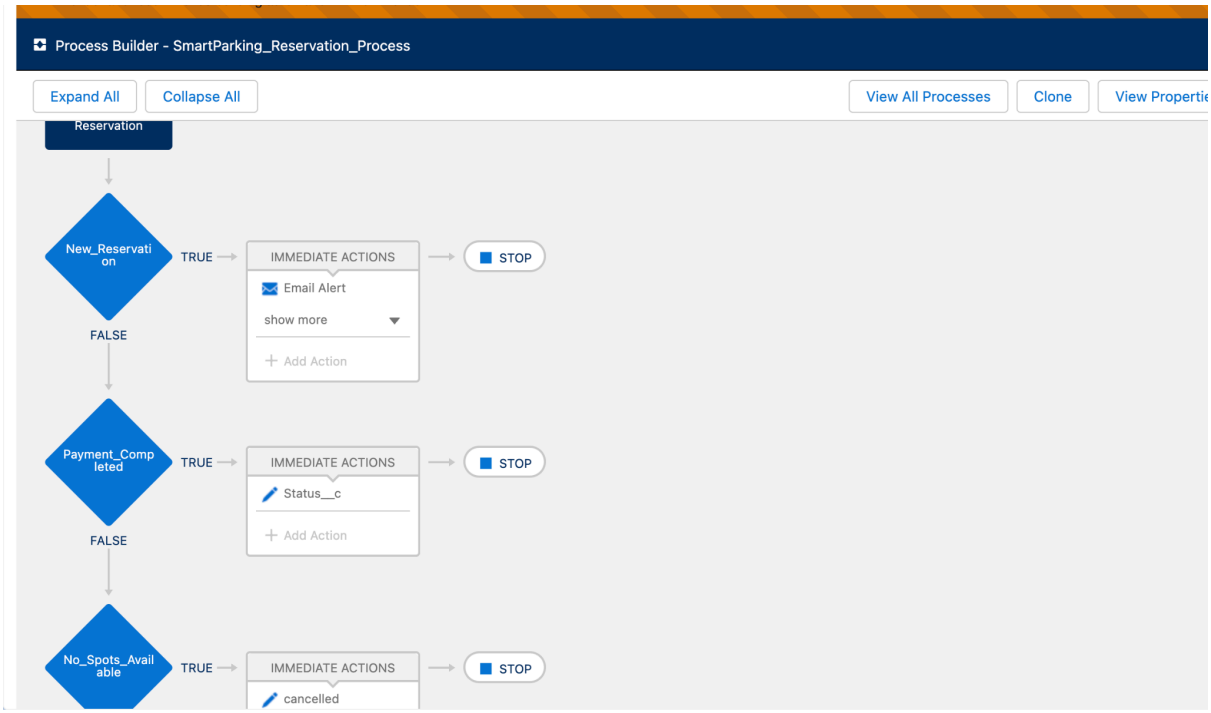
Criteria Name: No_Spots_Available

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

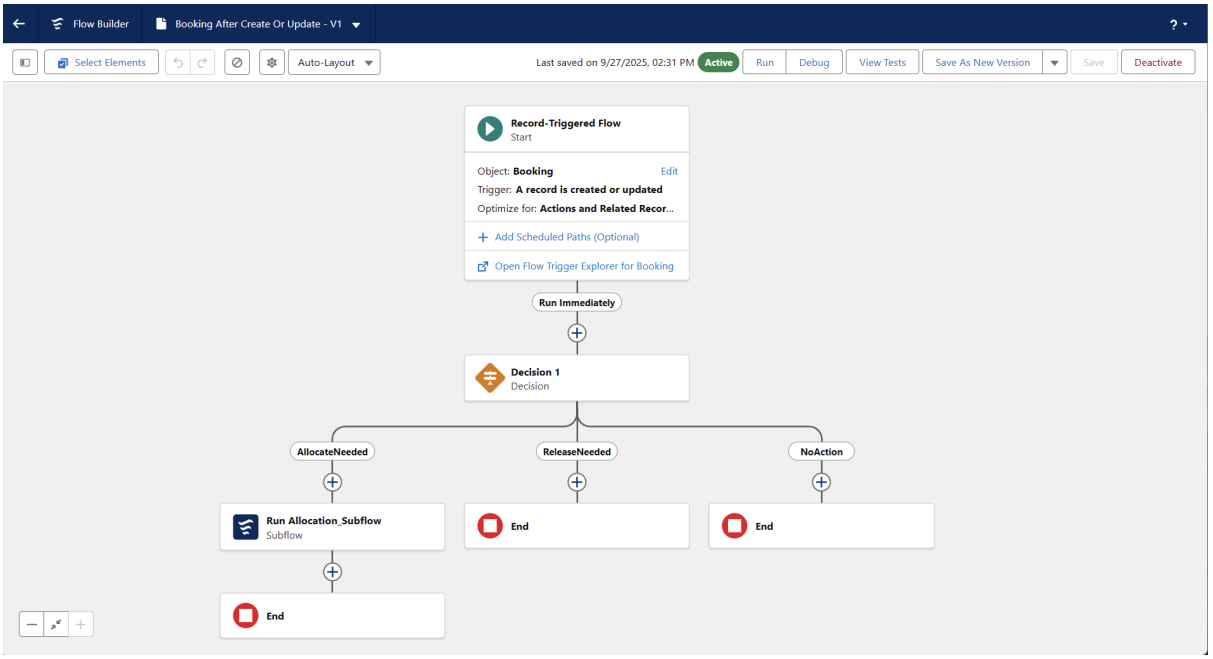
Optimization: Actions and Related Records

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

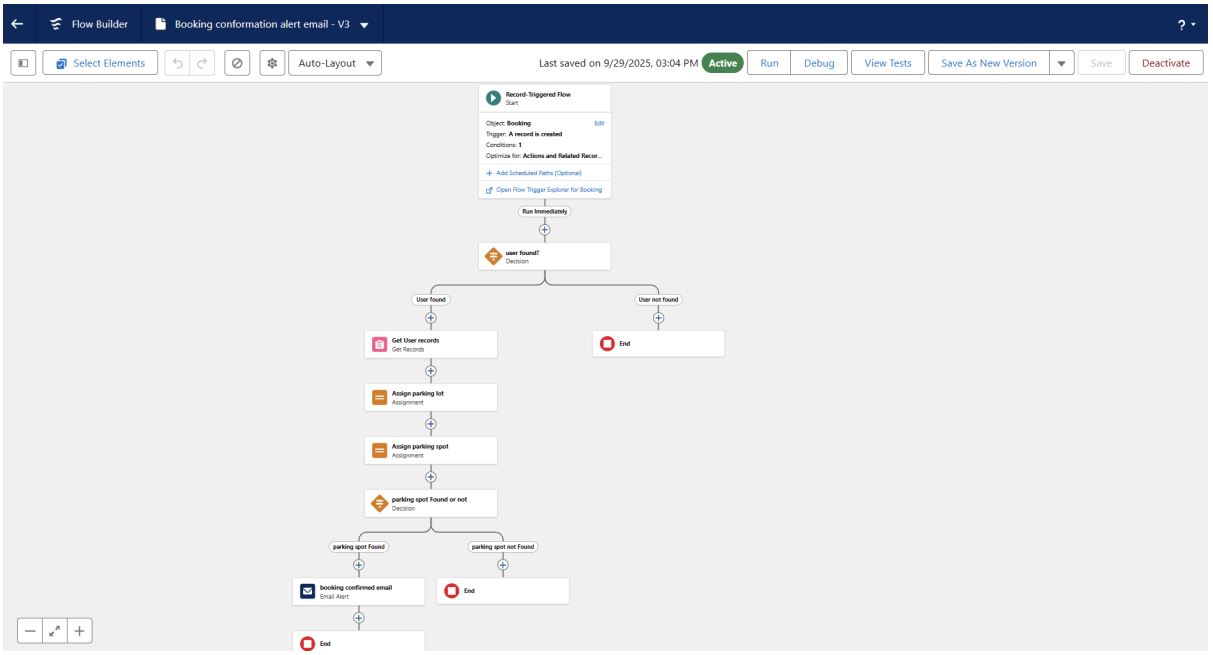
Optimization: Actions and Related Records

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.

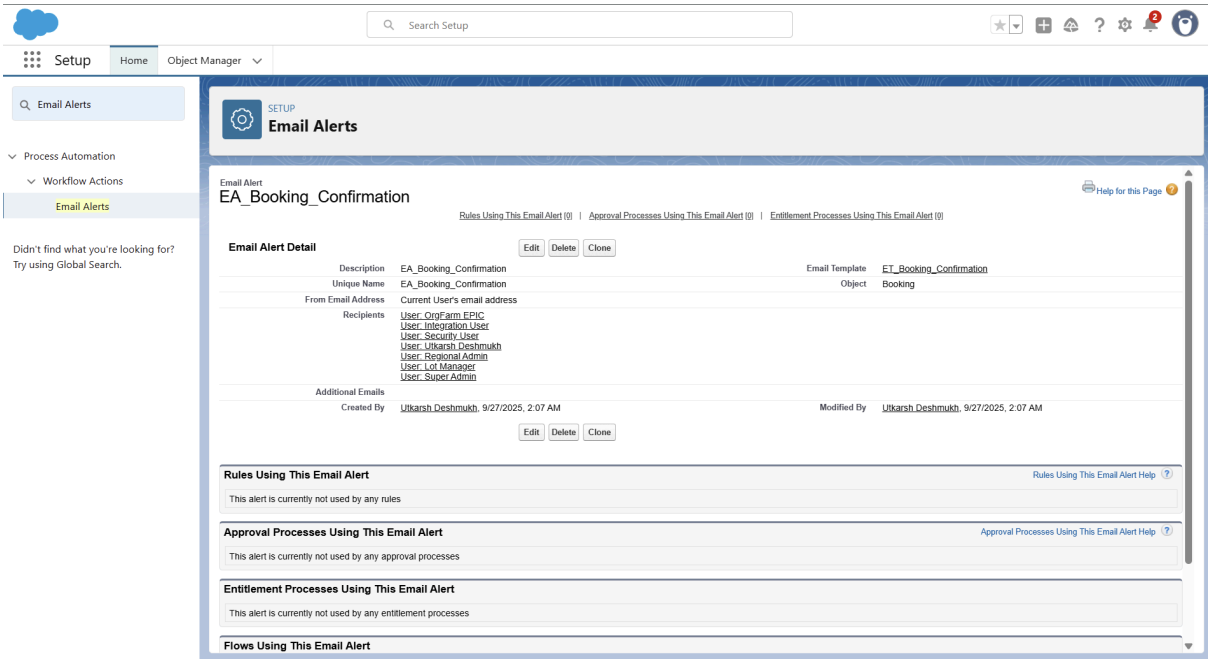


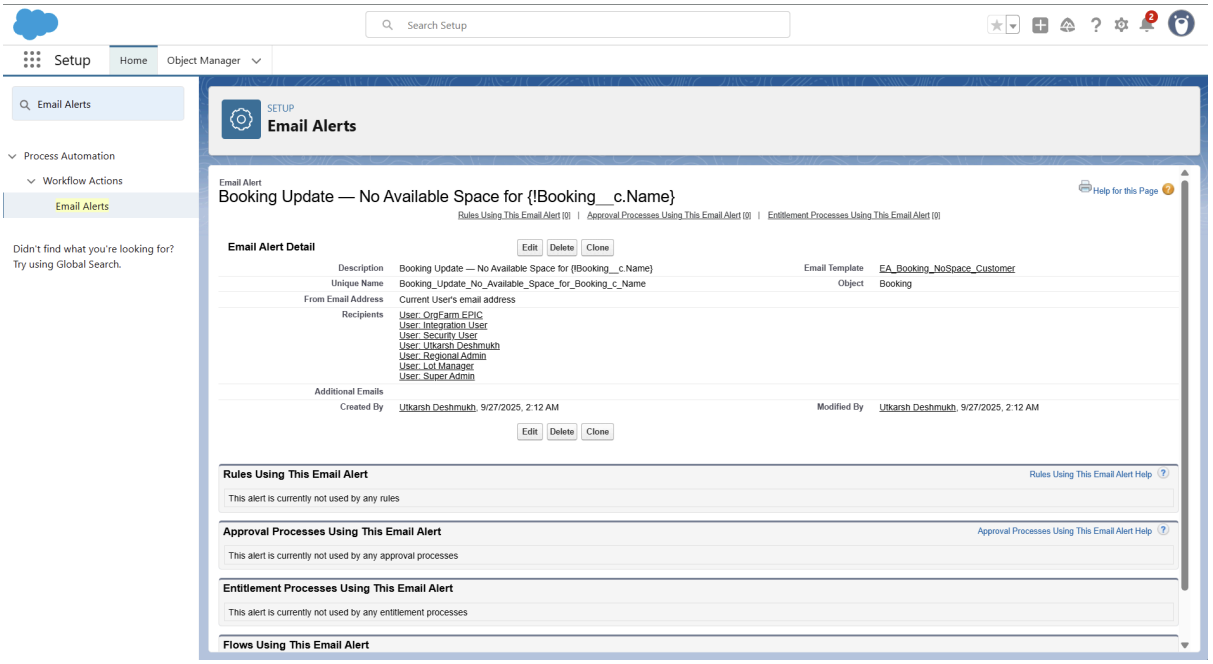
4. Email Alerts:

- **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.





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

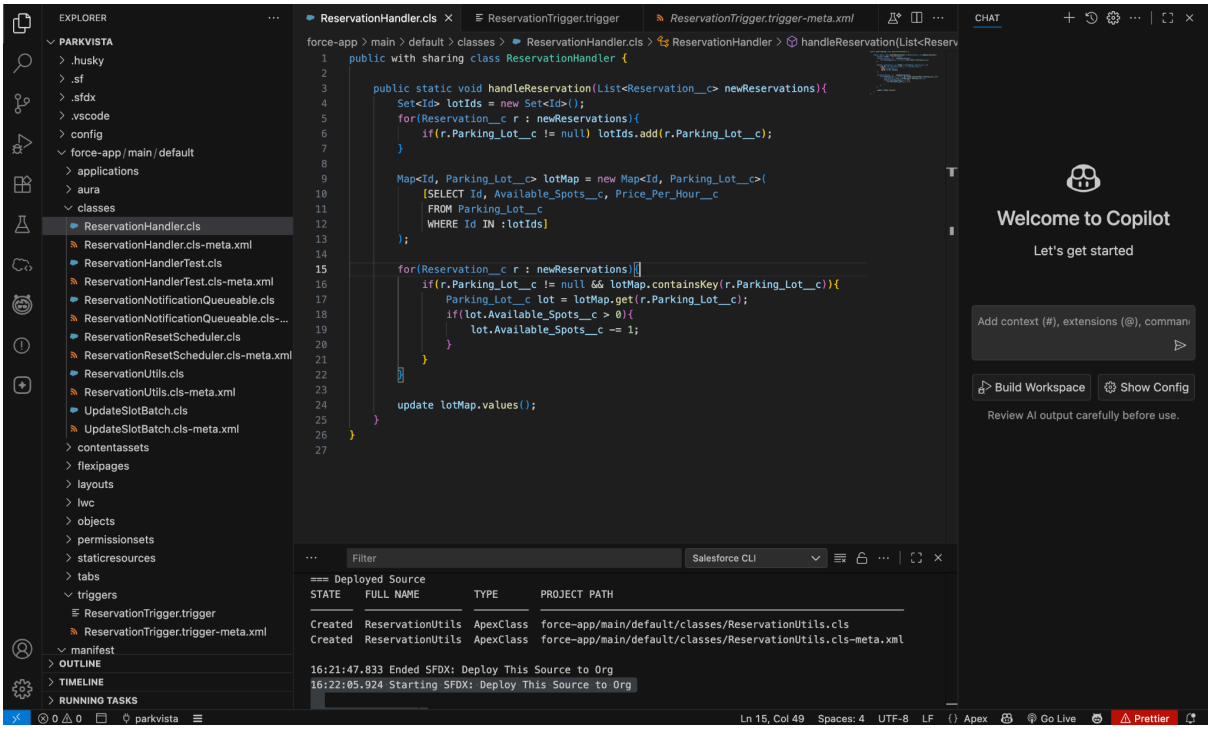
Purpose:
Automates parking lot slot availability whenever a reservation is made.

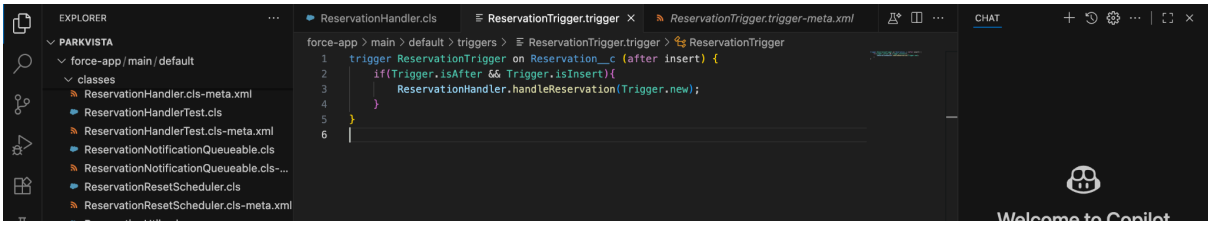
Importance:
Without this, admins would have to manually reduce available spots when someone books. This ensures real-time slot tracking.

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.





2. ReservationHandlerTest

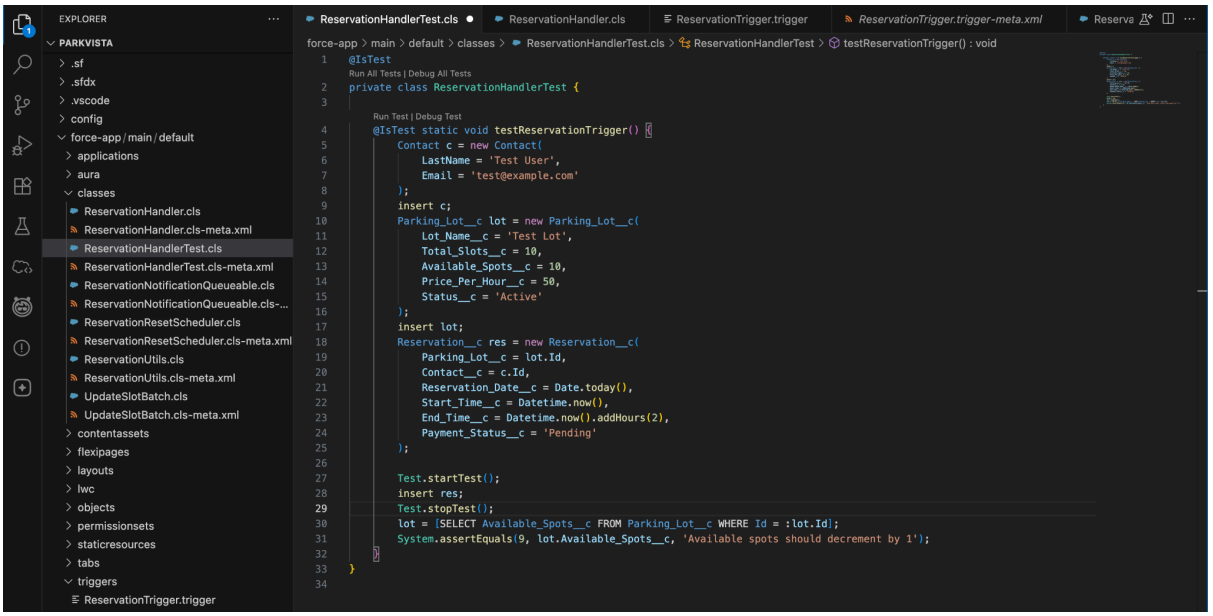
Purpose:
Ensures your business logic works correctly.

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.



3. ReservationNotificationQueueable

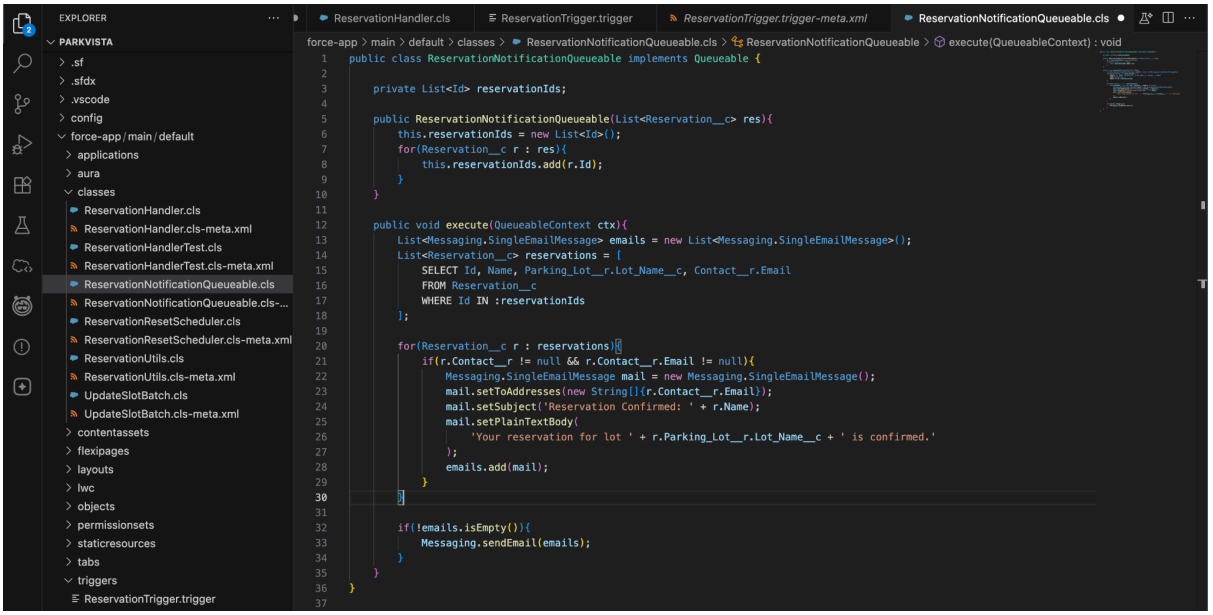
Purpose:
Sends **email confirmation** to customers after booking.

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."



4. ReservationUtils

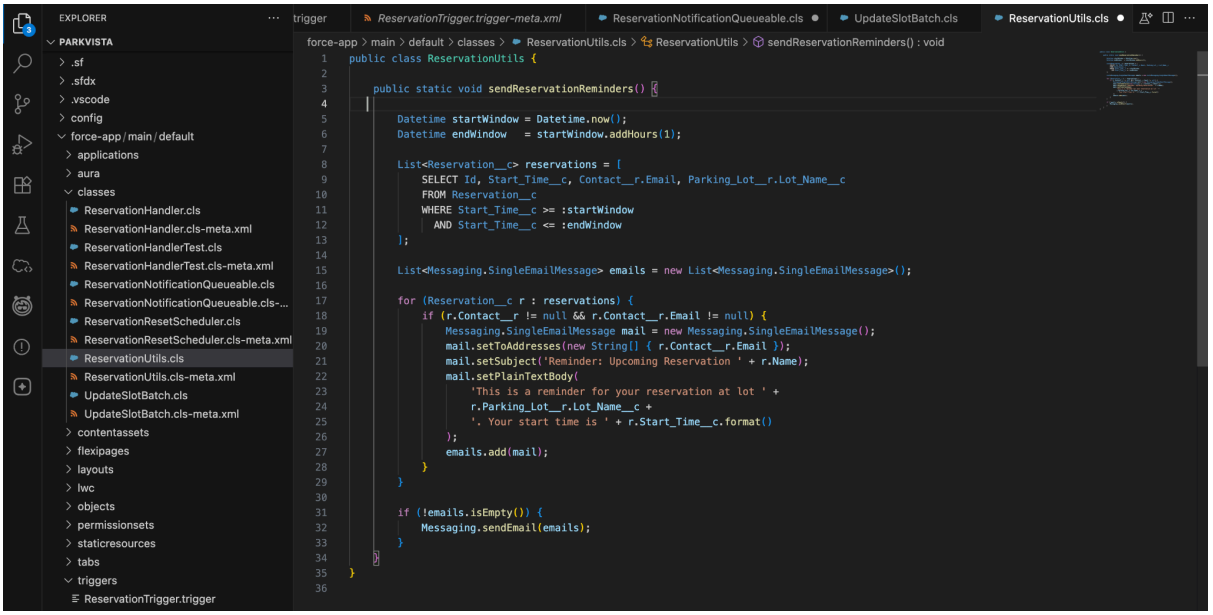
Purpose:
Sends **reminder emails** to customers whose reservations are about to start soon.

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.”*



5. UpdateSlotBatch

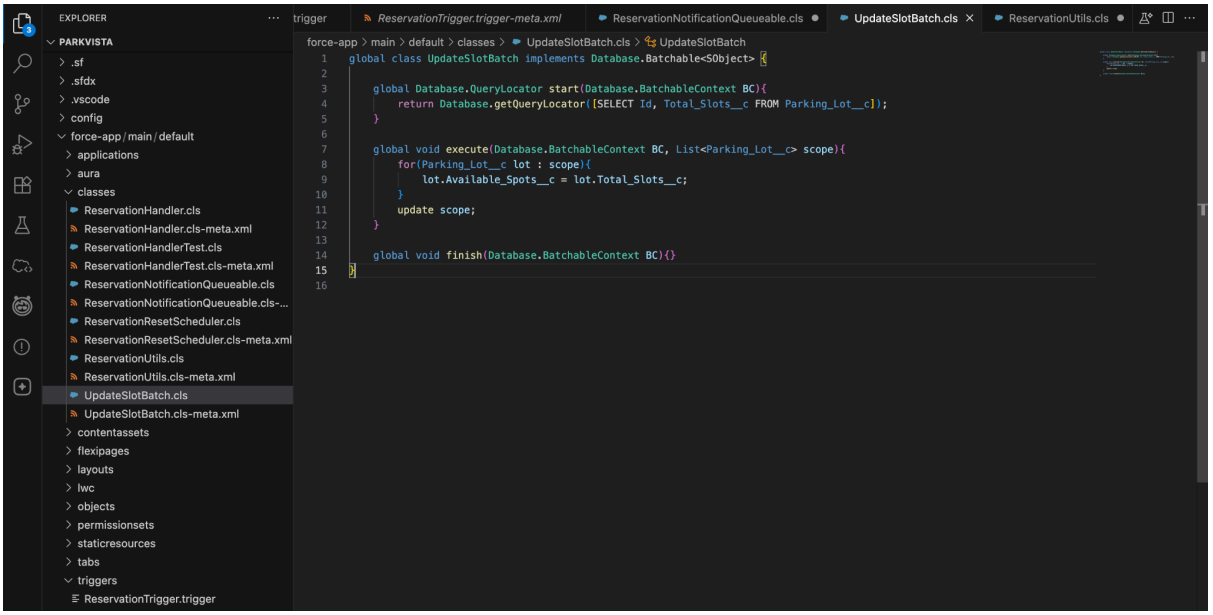
Purpose:
Resets parking lot availability back to full capacity.

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.



6. ReservationResetScheduler

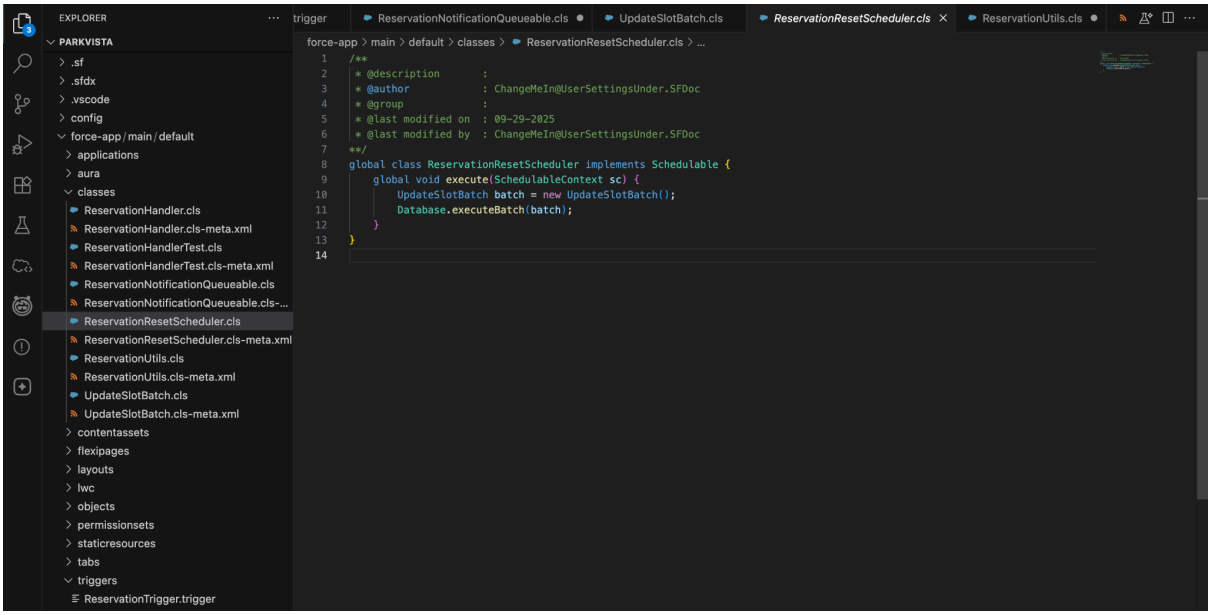
Purpose:
Schedules the batch (`UpdateSlotBatch`) to run automatically.

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.



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

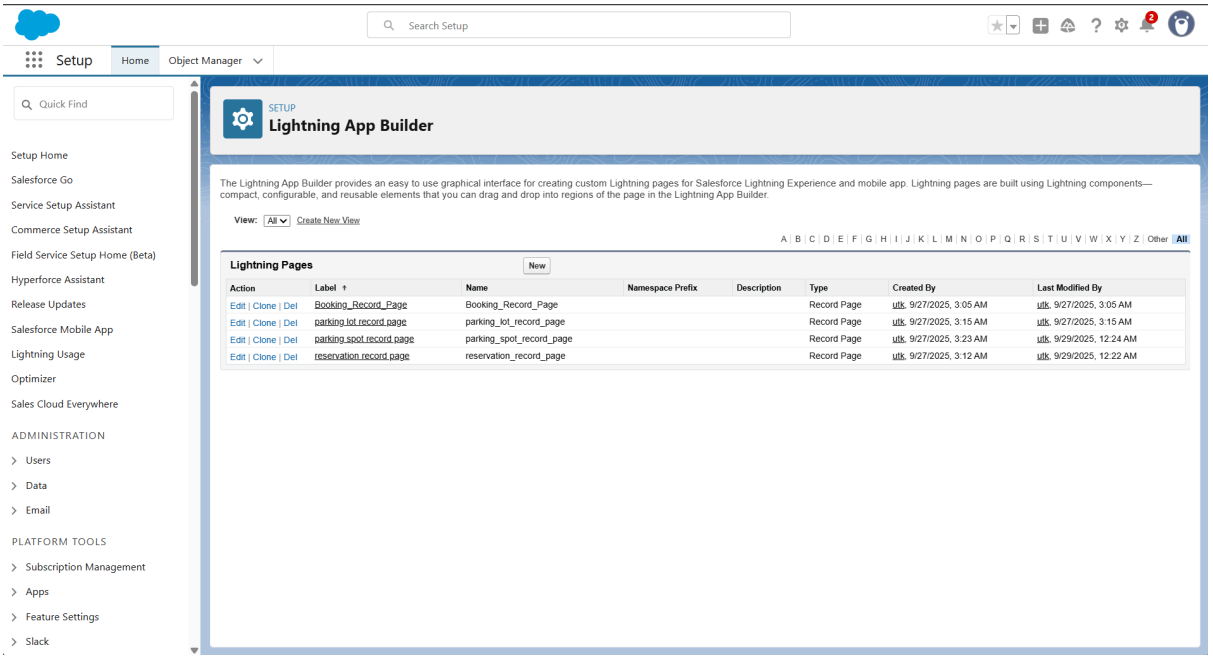
Purpose: To create custom pages for apps, record pages, and home pages without code.

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.



Record Pages

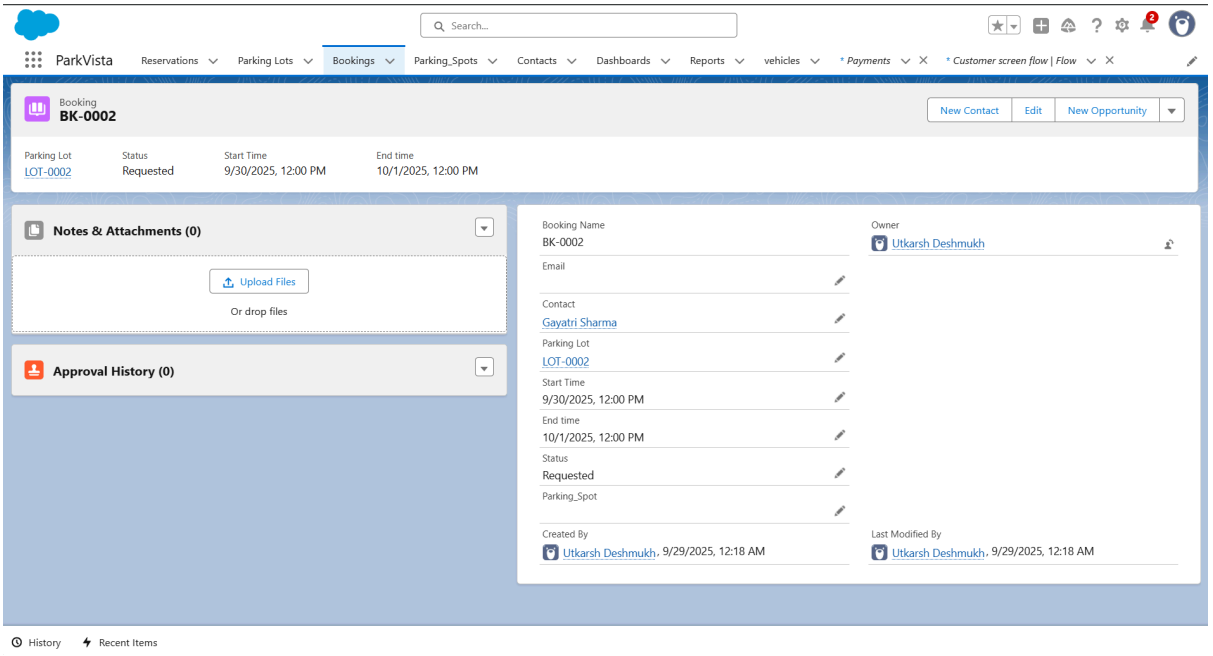
Purpose: Customize the layout and UI for individual objects' records.

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.



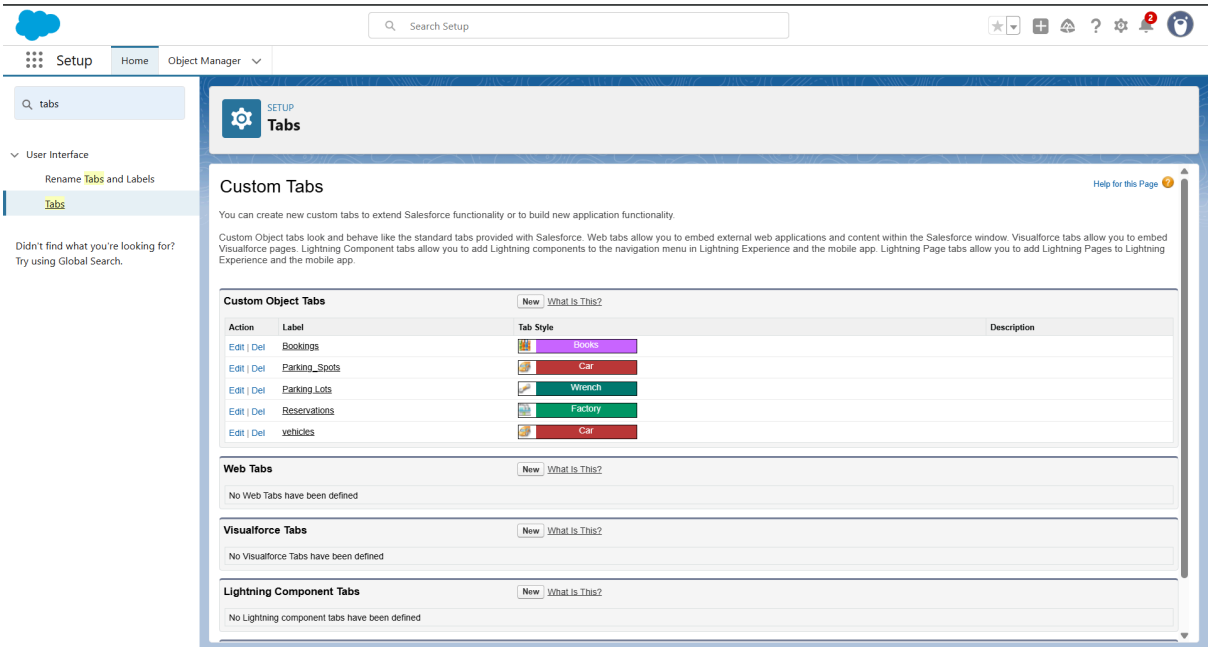
Tabs

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.



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.

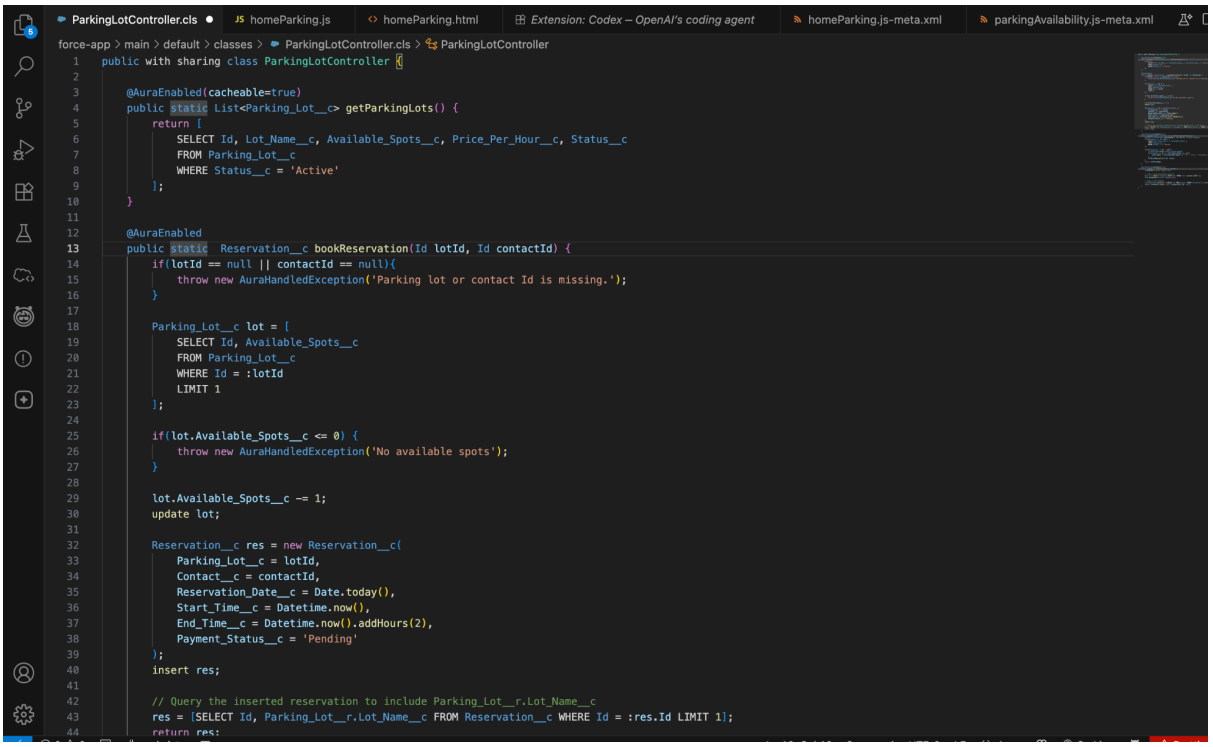
3. Workflow

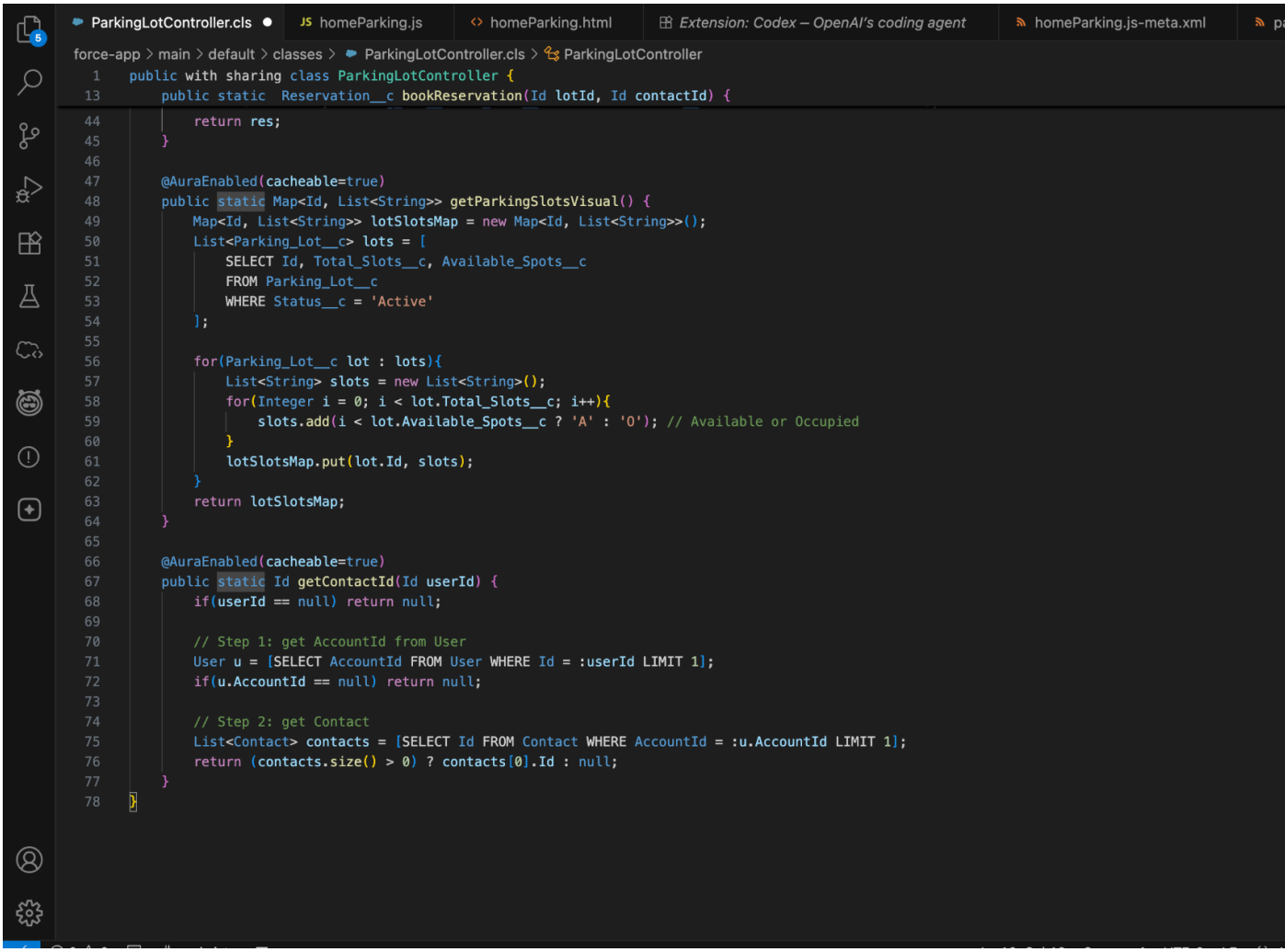
1. **Load Home Page:**

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

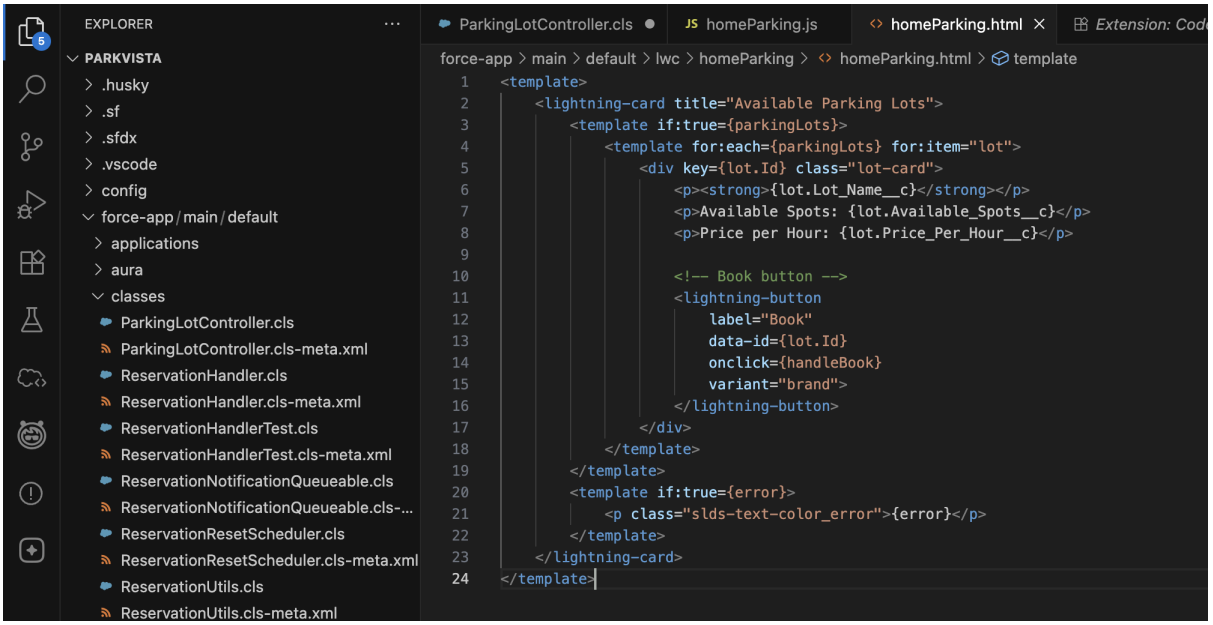
2. **Fetch Current User Contact:**
   - connectedCallback() calls getContactId(USER_ID) imperatively to fetch the contact linked to logged-in user.
3. **Book Parking Spot:**
   - Clicking the “Book” button triggers handleBook().
   - Calls bookReservation(lotId, contactId) imperatively.
   - Shows **success 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

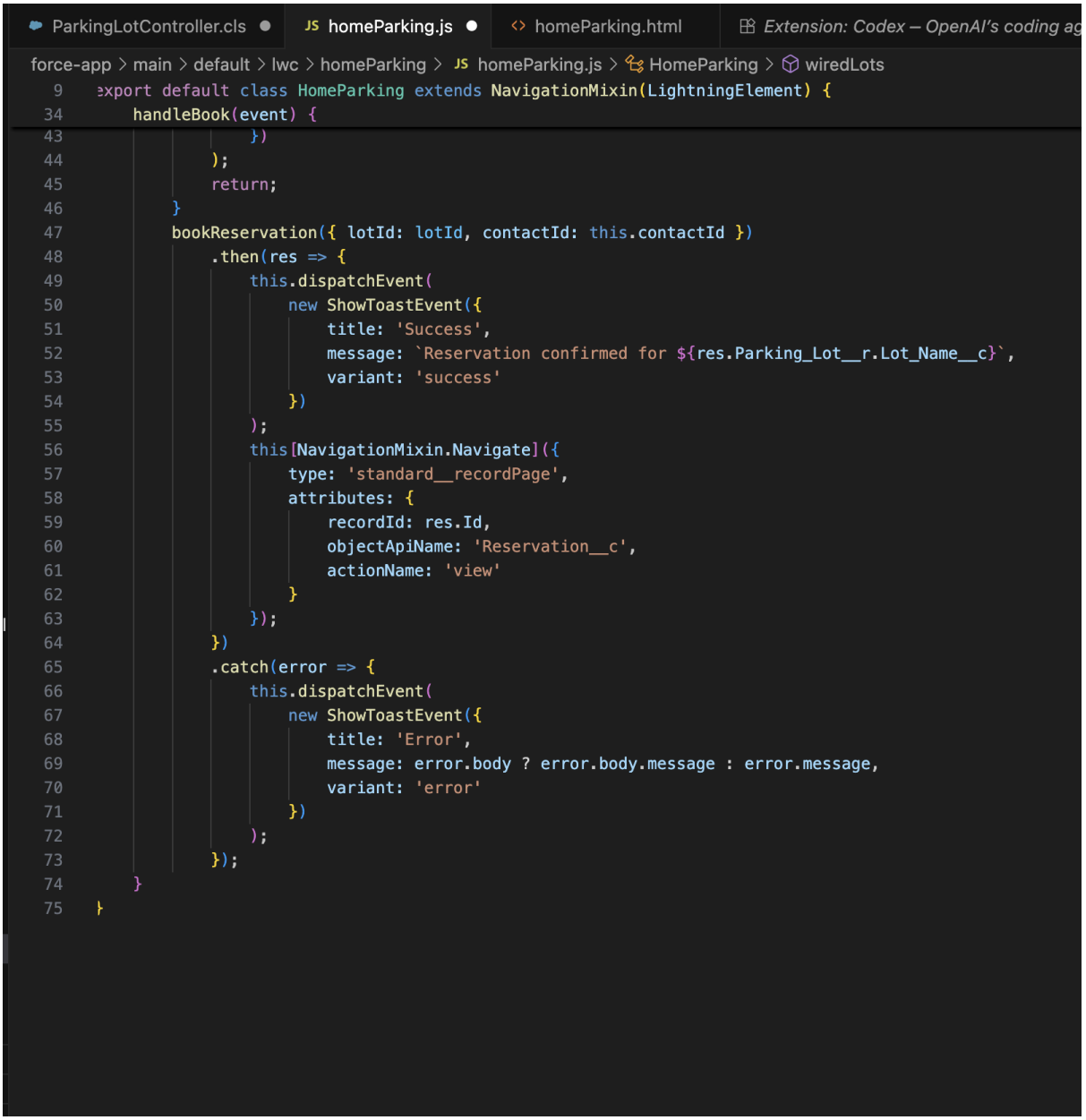




5. LWC: homeParking.html



6. LWC: homeParking.js



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

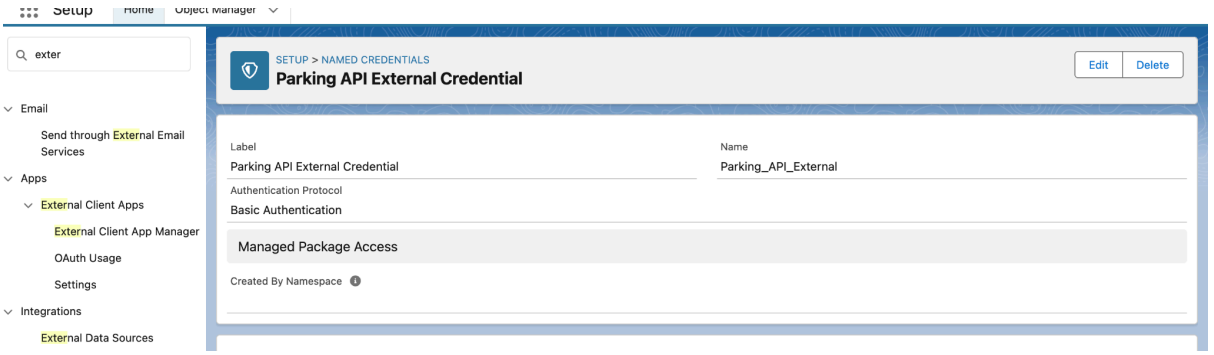
Securely connect Salesforce to an external service (e.g., Parking API) using **Named Credentials** with **Basic Authentication**. This avoids hardcoding credentials and ensures centralized security.

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**

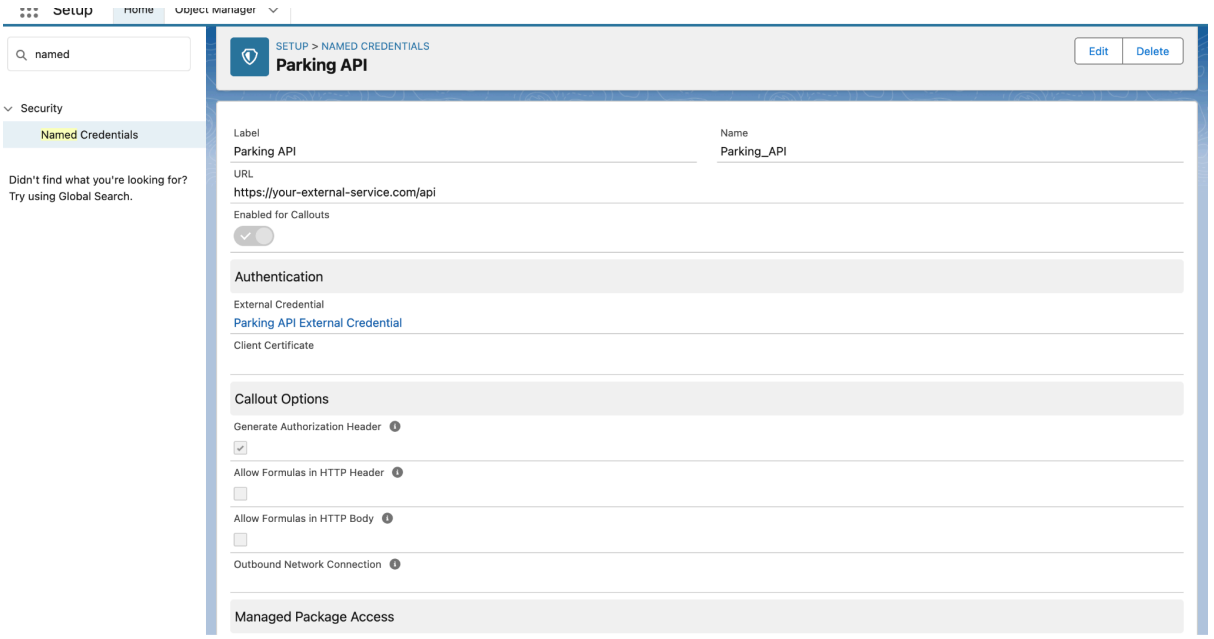
Step 3: Create Named Credential

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_External
3. Save the Named Credential.

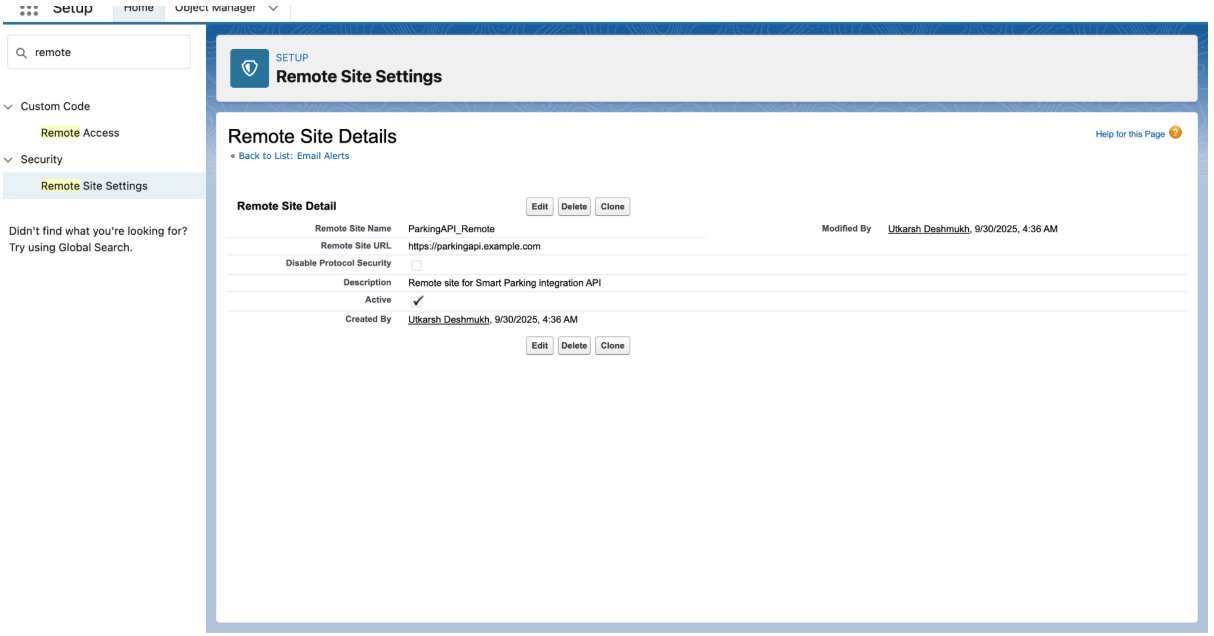


Step 4: Remote Site Settings

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.

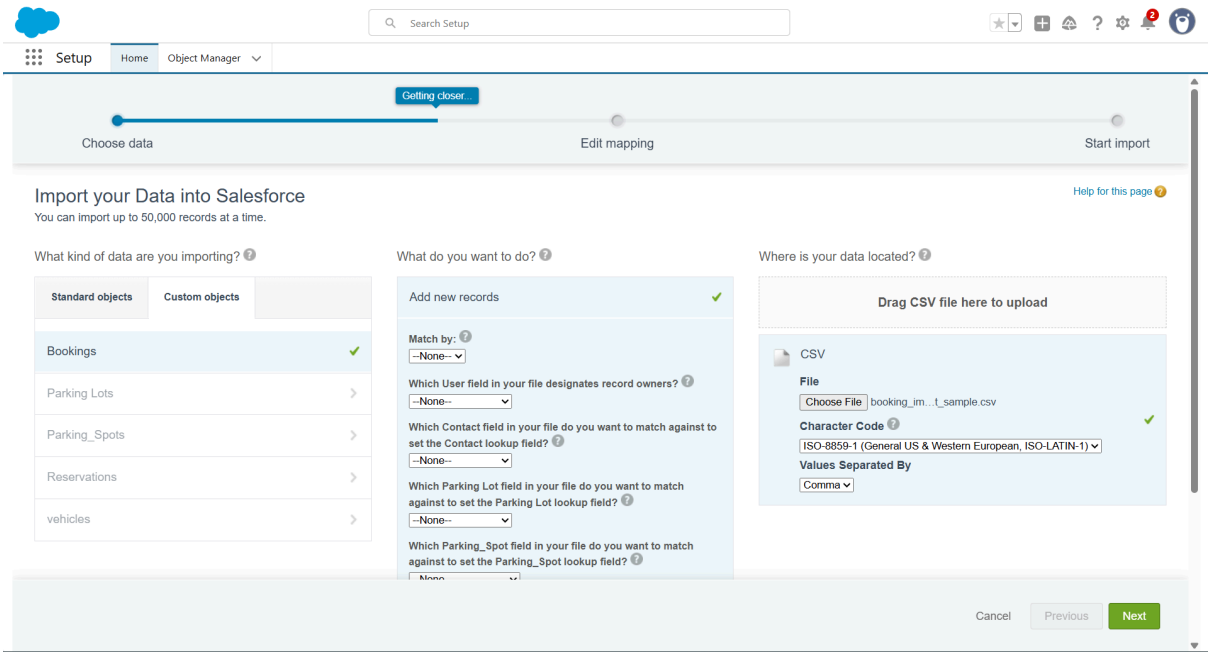
1. Data Import Wizard

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.



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.

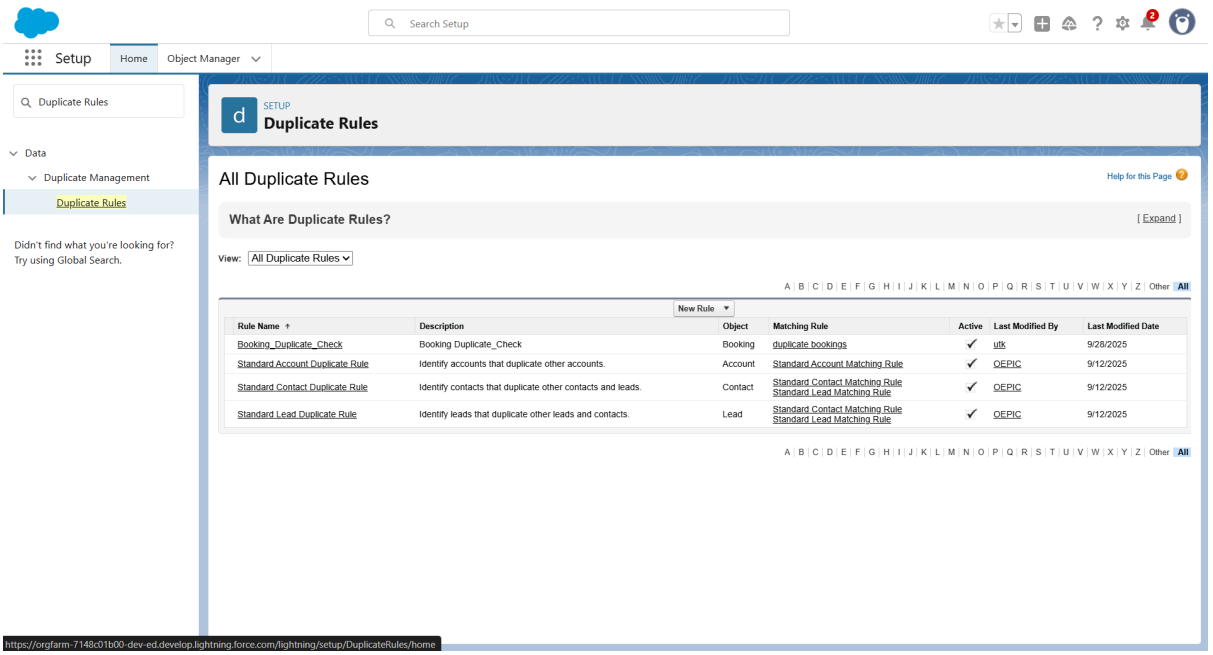
3. Duplicate Rules

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.



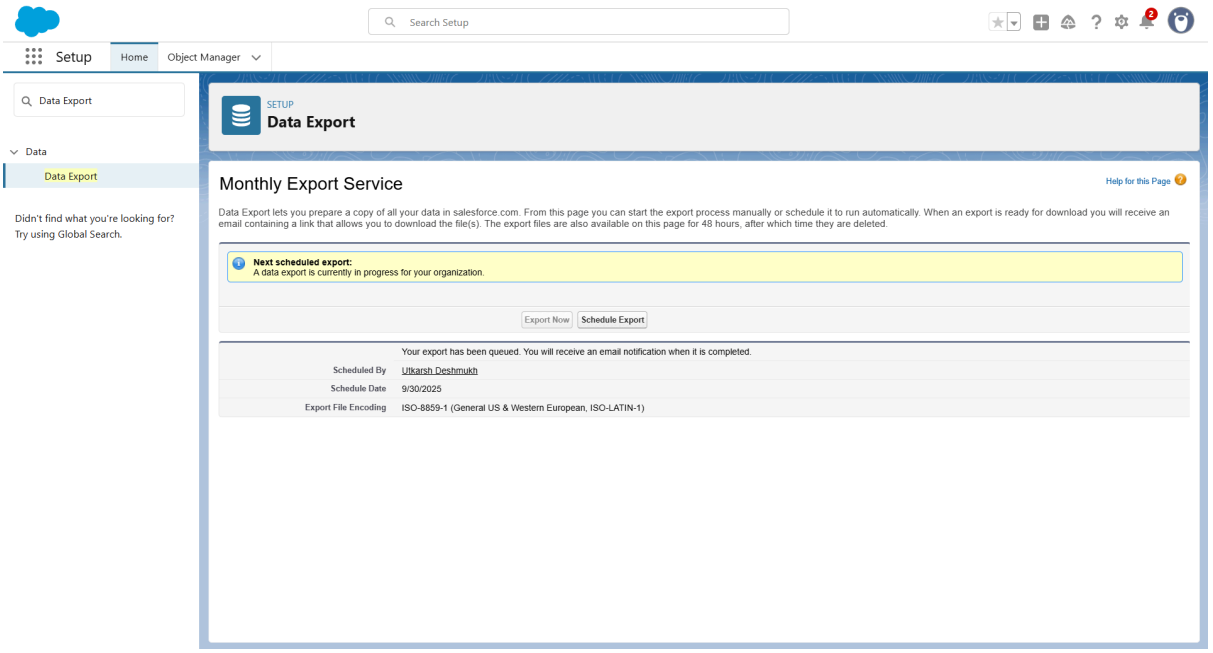
4. Data Export & Backup

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.

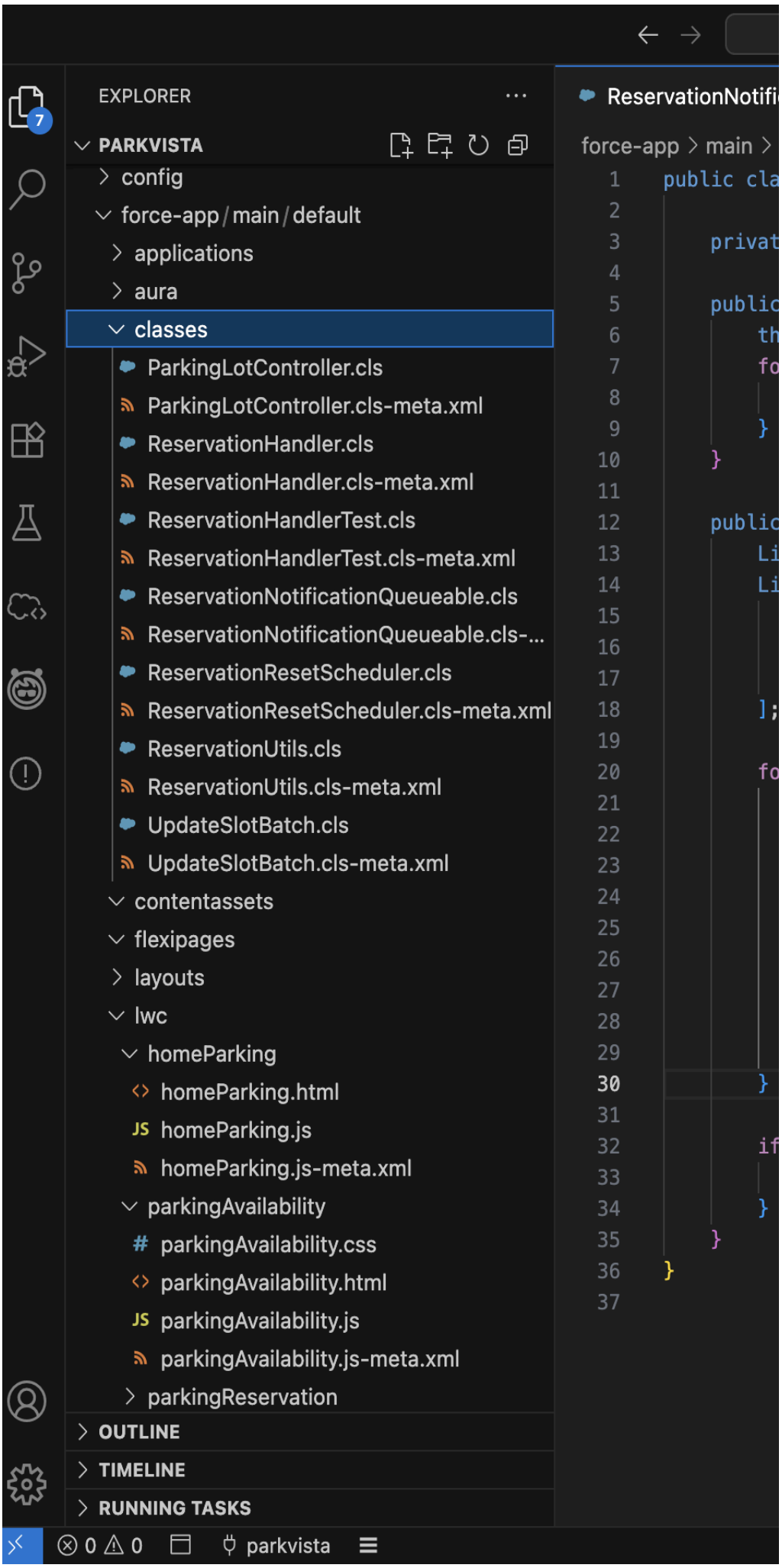
6. VS Code & SFDX

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.


EXPLORER
PARKVISTA
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
ReservationNotificationQueueable.cls-...
ReservationResetScheduler.cls
ReservationResetScheduler.cls-meta.xml
ReservationUtils.cls
ReservationUtils.cls-meta.xml
UpdateSlotBatch.cls
UpdateSlotBatch.cls-meta.xml
contentassets
flexipages
layouts
lwc
homeParking
homeParking.html
homeParking.js
homeParking.js-meta.xml
parkingAvailability
parkingAvailability.css
parkingAvailability.html
parkingAvailability.js
parkingAvailability.js-meta.xml
parkingReservation
OUTLINE
TIMELINE
RUNNING TASKS
ReservationNotifi
force-app > main >
public cla
privat
public
th
fo
}
}
public
Li
Li
];
fo
}
if
}
}
}
0 0
parkvista

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:

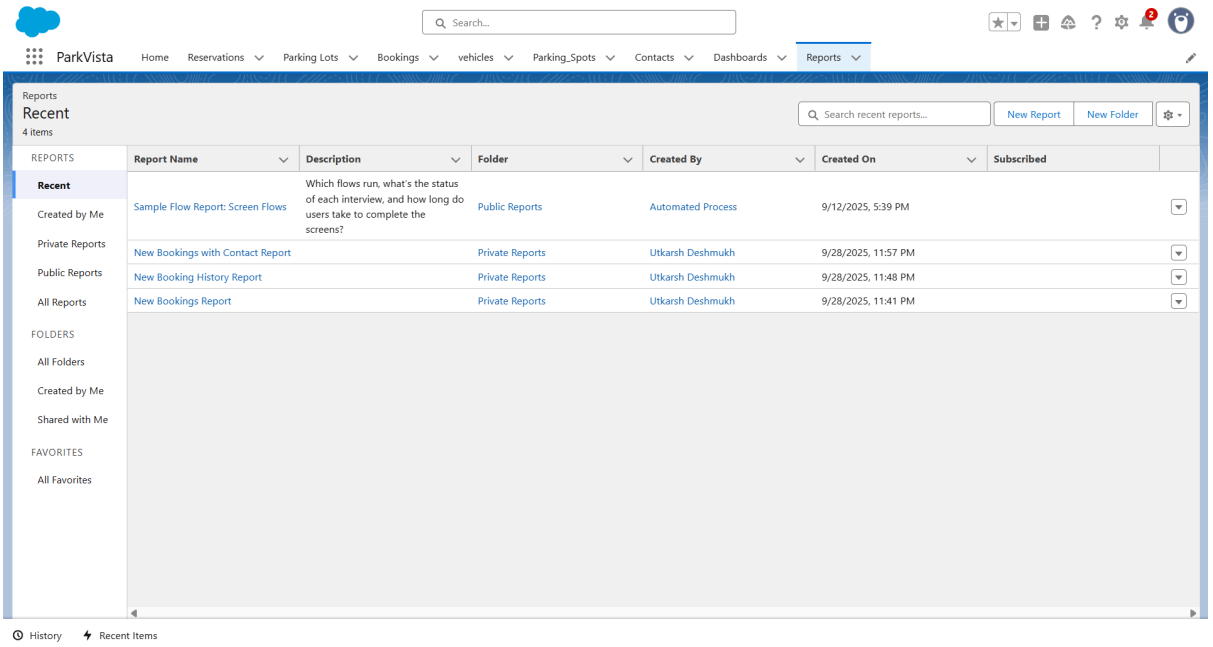
This phase ensures that operational and management decisions are data-driven. Reports provide detailed insights into bookings, payments, and parking occupancy. Dashboards aggregate reports into intuitive visualizations. Security measures, including sharing rules, field-level security, session restrictions, 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.



---

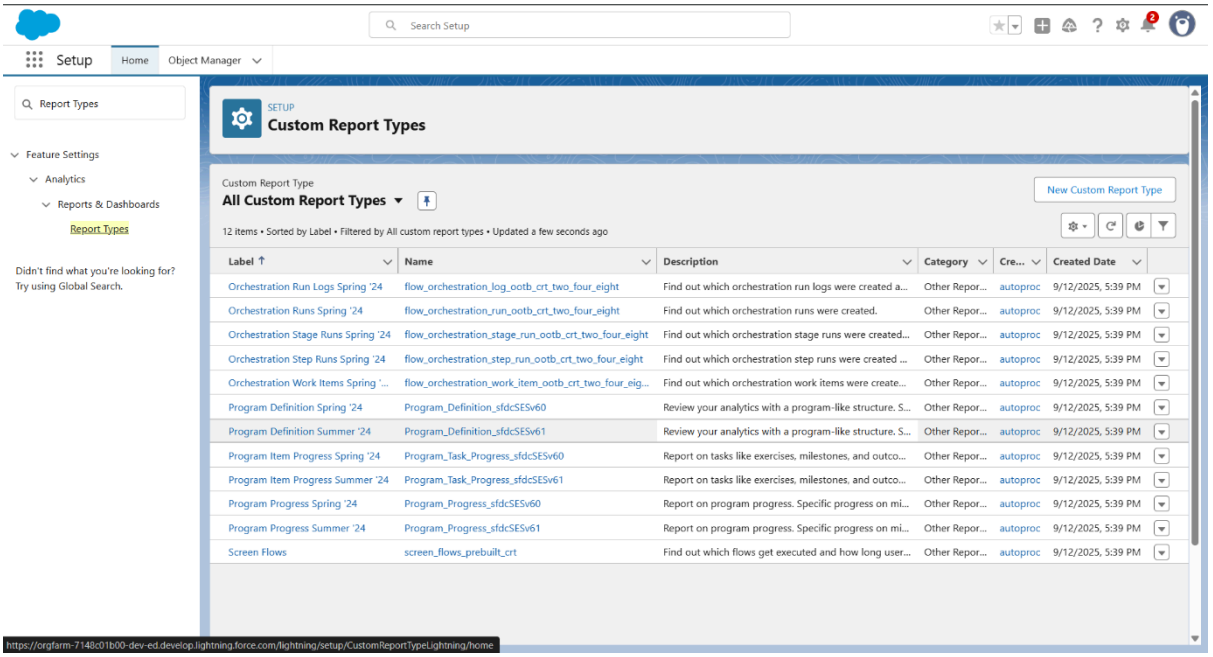
2. Report Types

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:

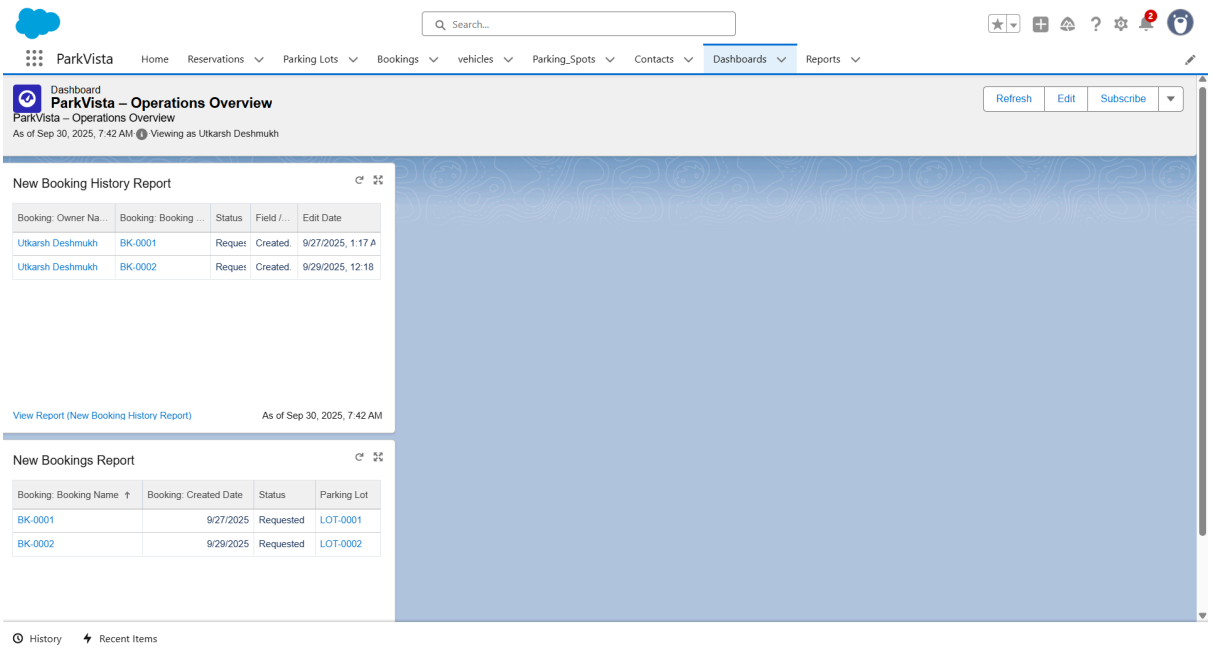
- **Bookings with related Parking Spaces:** Track space utilization per booking.
- **Payments with related Bookings:** Analyze revenue trends per booking.
- **Vehicles with related Customers:** Identify repeat customers or high-value vehicles.



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.

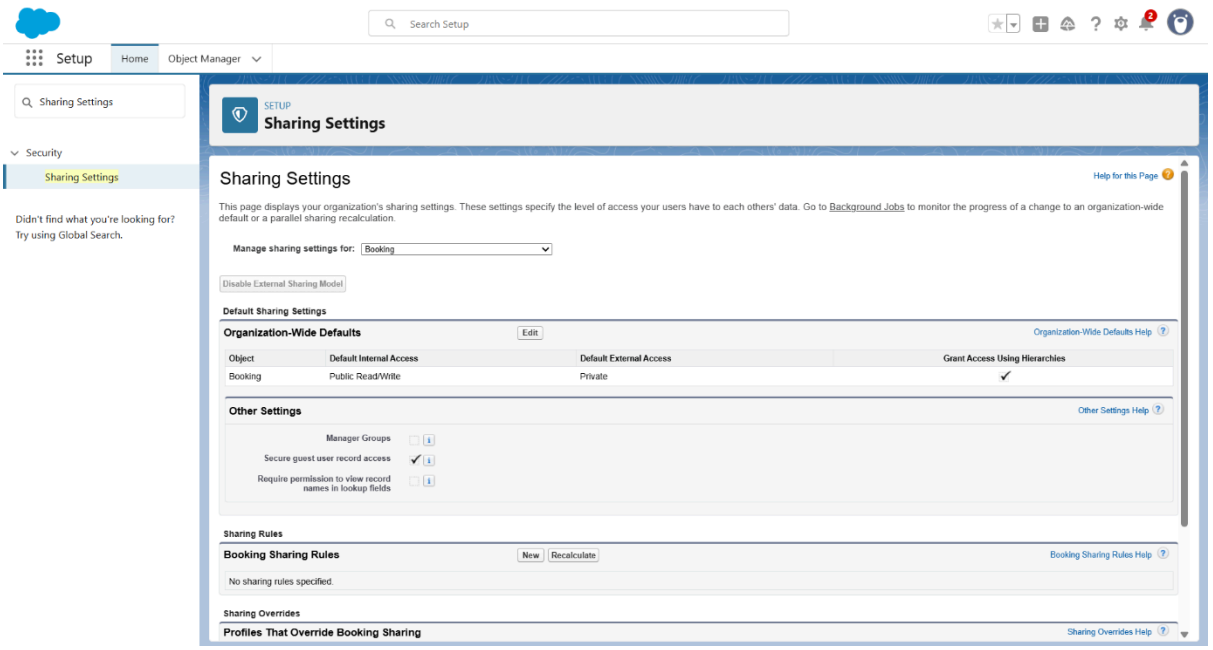


4. Sharing Settings

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)

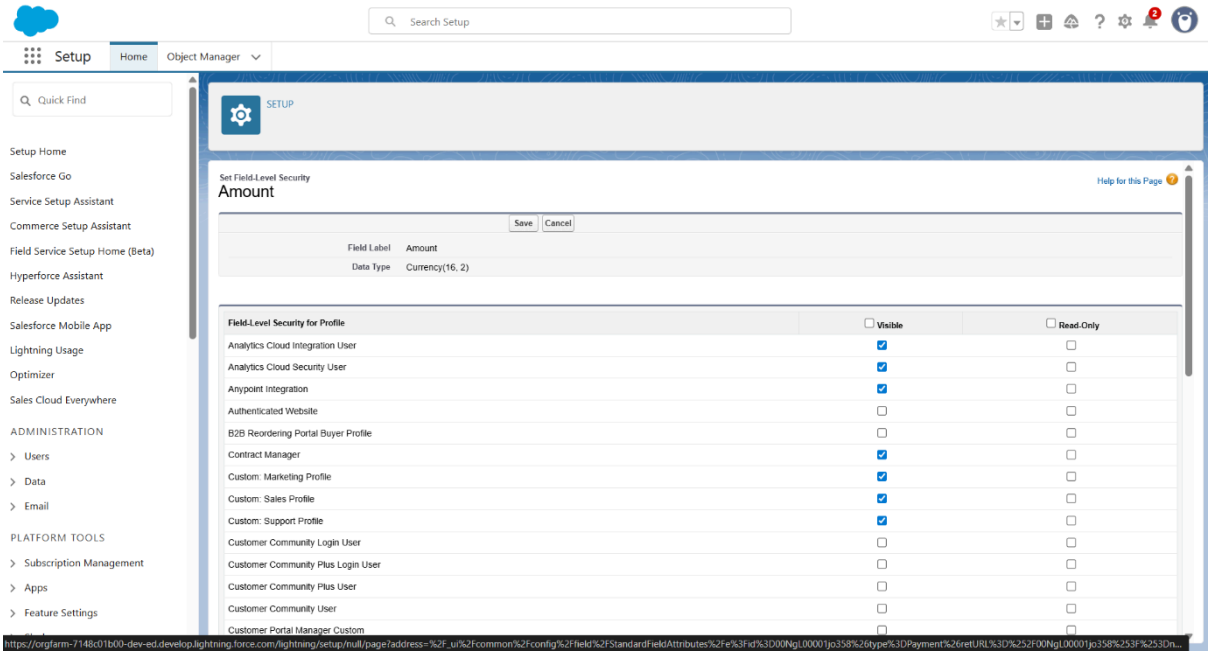


5. Field-Level Security (FLS)

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.

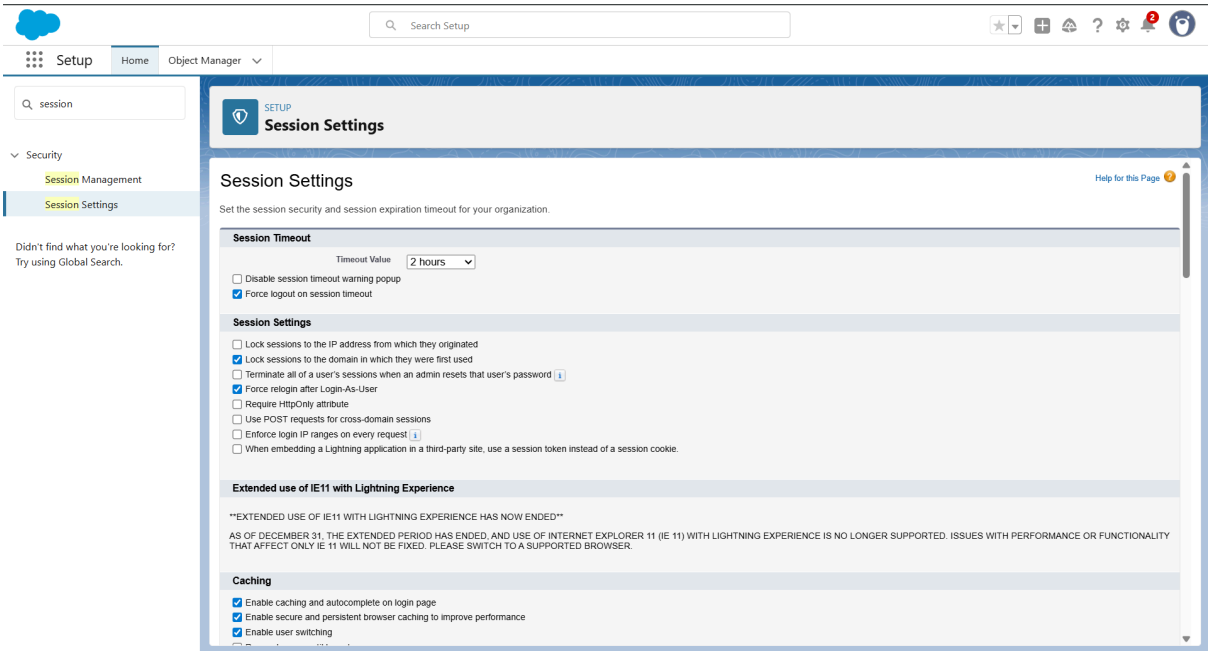


6. Session Settings

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



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