# TCS-LAST-MILE

## Title - Event-Registration-Attendee-Management

Event Registration & Attendee Management is a Salesforce-based solution designed to simplify event organization for institutions and businesses. This project enables organizers to create and manage event listings, register attendees, automate confirmation/reminder emails, and track attendance and engagement through dashboards and reports.

## Event Management System

### Problem Understanding & Industry Analysis

The core problem is that many organizations struggle to manage event registrations efficiently, leading to lost data, poor communication, and low attendee engagement. Most educational institutions, clubs, and businesses require a centralized system to handle registrations, automate notifications, and improve attendee experience.

### Requirement Gathering

- Events must be created, scheduled, updated, and cancelled.
- Attendees need to register for events with personal details (name, email, contact, etc.).
- Organizers require automation for confirmation and reminders.
- Attendance tracking and reports are necessary for insights.
- Simple user interface for both admin and participants.

### Stakeholder Analysis

- Event Organizers: Manage events, track registrations, analyze reports.
- Attendees/Participants: Register for events, receive notifications, view schedules.
- Admins: Oversee processes, manage users, troubleshoot issues.

### Business Process Mapping

Event creation → Registration opens → Attendee registers → Confirmation sent → Reminders sent → Event occurs → Attendance recorded → Feedback/metrics tracked.

### Possible exceptions:

Event rescheduling/cancellation, attendee unregistration.

## Industry-Specific Use Case Analysis

- Educational Institutions: Manage workshops, seminars, student participation.
- IT Companies: Employee training, hackathons, client events.
- Professional Associations: Conferences, meetings, networking events.

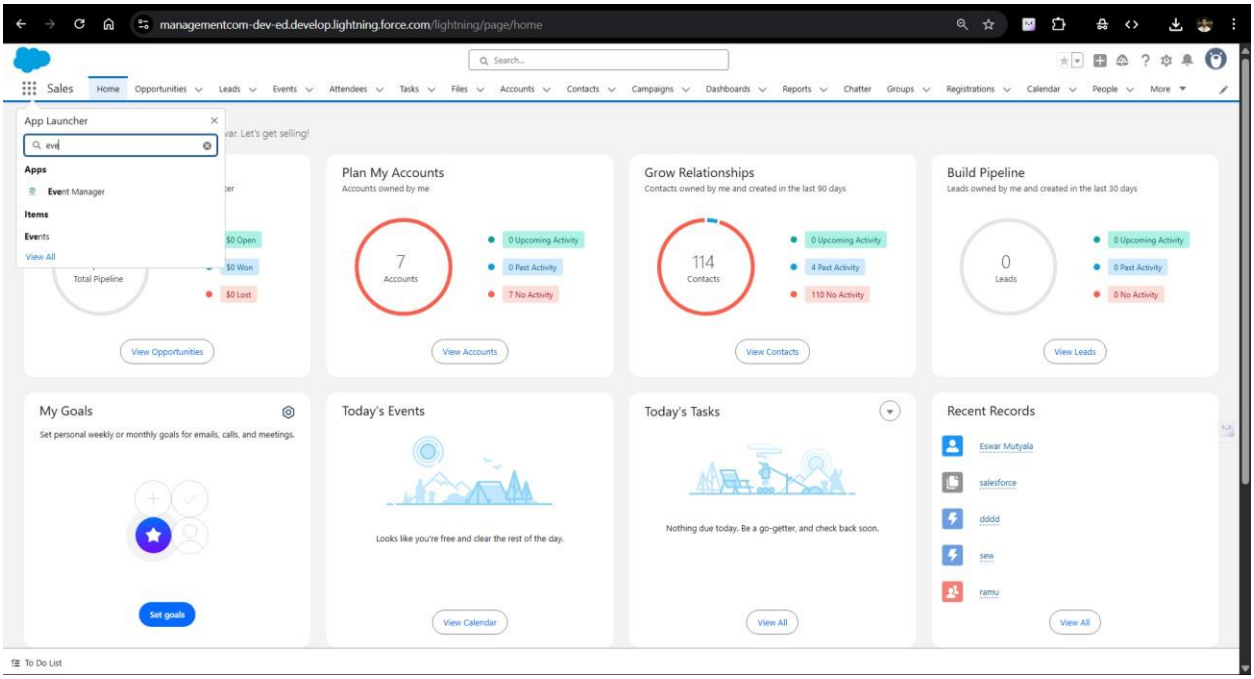
## AppExchange Exploration

- Review AppExchange apps for event management (Cvent, Blackthorn, etc.).
- Analyze what features they offer: registration forms, automation, reporting, scalability.
- Use insights for best practices and Salesforce-native enhancements.

# Phase 2 – EVENT MANAGEMENT AND REGISTRATION

1. **Salesforce Org Preparation**

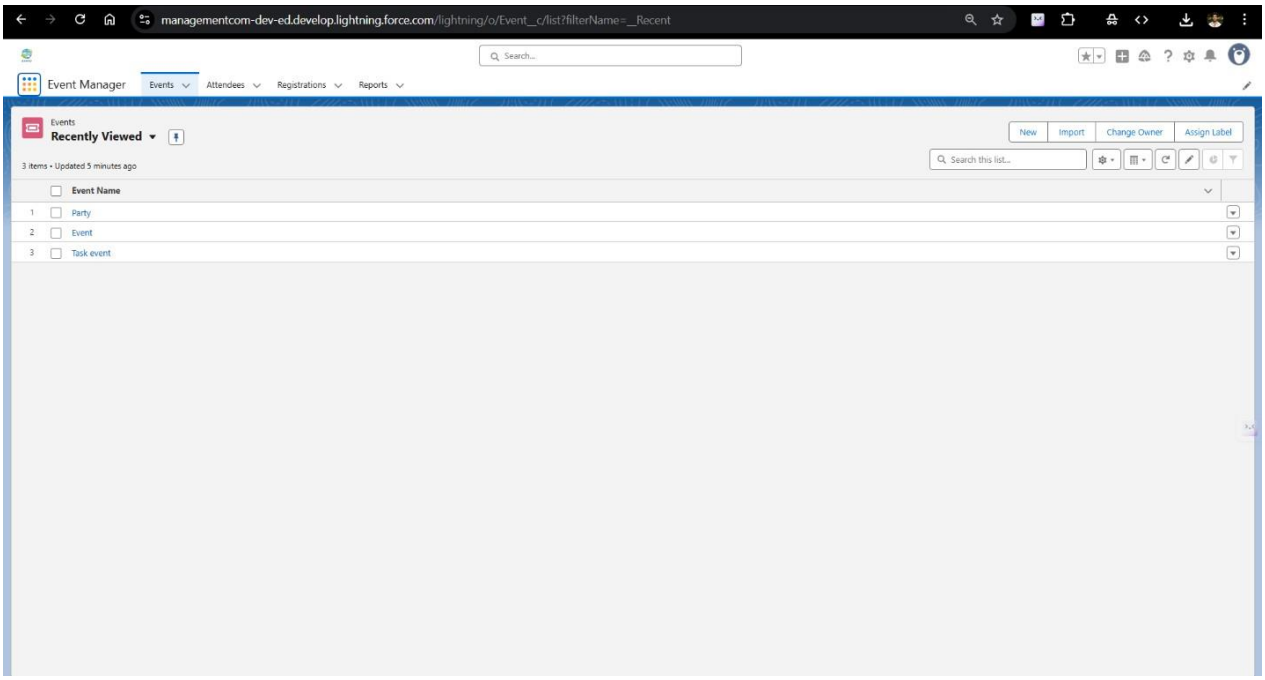
- Registered for a Salesforce Developer Edition org using [developer.salesforce.com].
- Switched user interface to Lightning Experience.
- Verified administrative access by confirming visibility of Setup, Profiles, Object Manager, App Manager, and required tabs.




2. **Lightning App Creation**

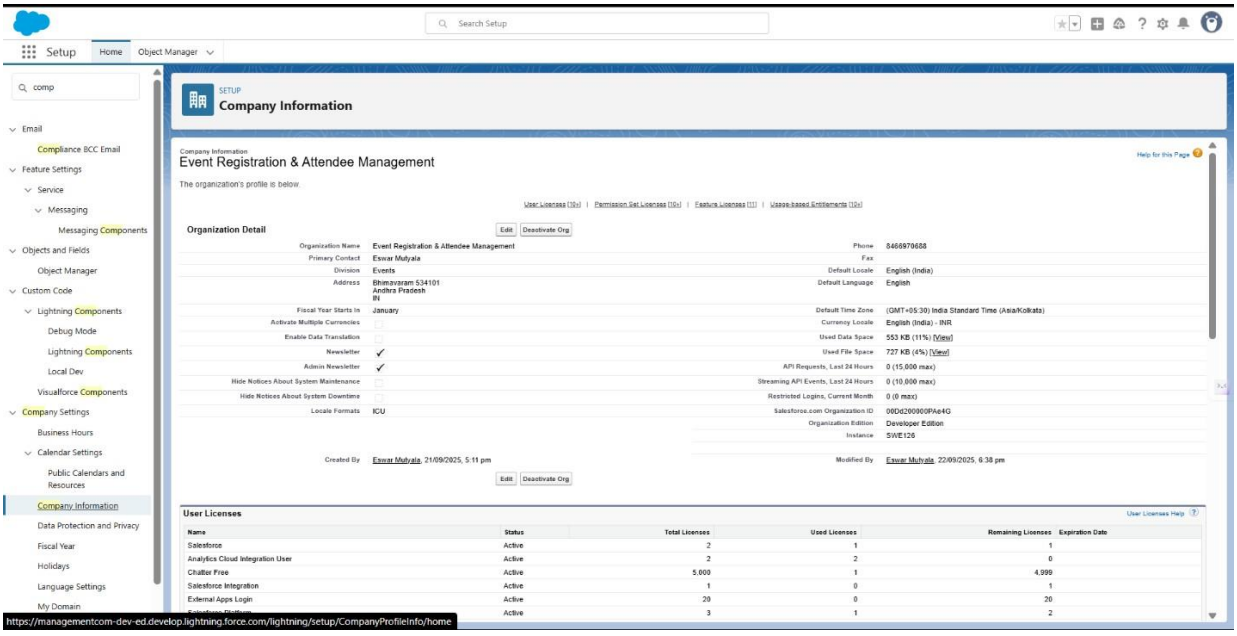
- Opened Setup > App Manager.
- Clicked New Lightning App.
- Provided the following information:
    - App Name: Event Management
    - Primary Color/Icon: [Custom or Default]
- Added the following navigation items (which will correspond to custom object tabs):
    - Events
    - Registrations
    - Attendees
- Made the app visible to target profiles (e.g., System Administrator, Event Admin, Organizer).

- Saved and launched the app.



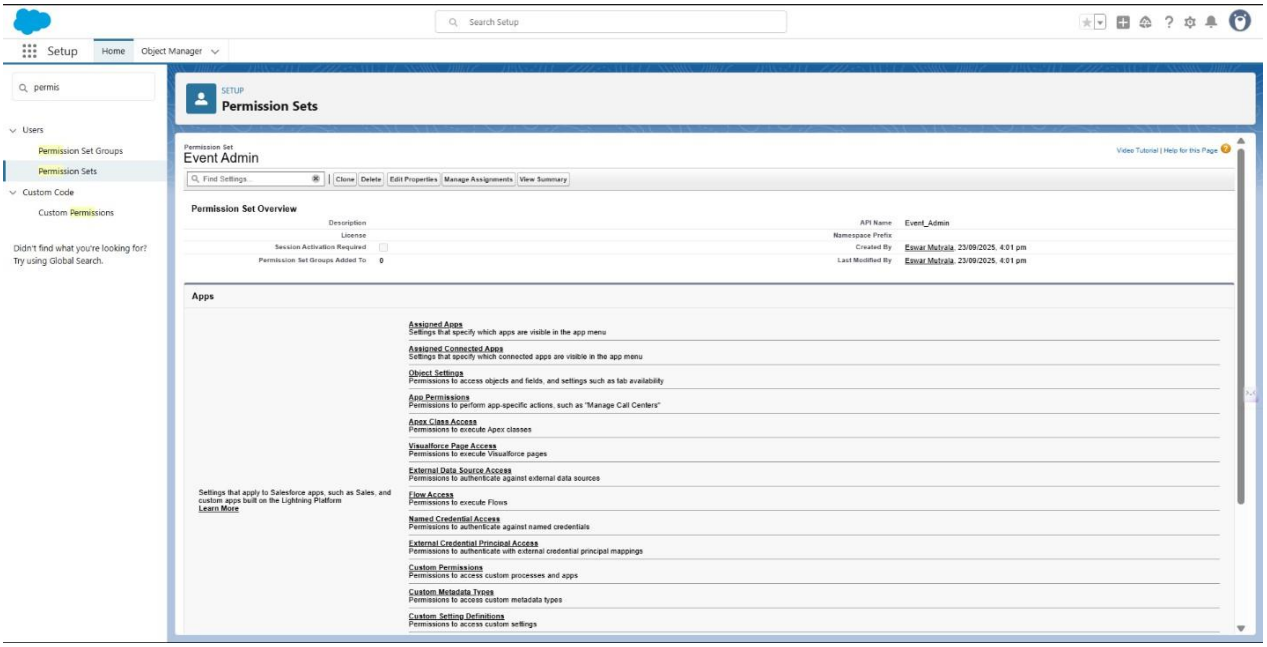

3. **Organization Information**

- Navigated to Setup > Company Information.
- Updated the following:
    - Organization Name: Event Registration C Attendee Management
    - Address: [Institution address]
    - Default Locale: English (United States)
    - Default Timezone: Set to the correct region
- Checked availability of Salesforce licenses for test and production users.



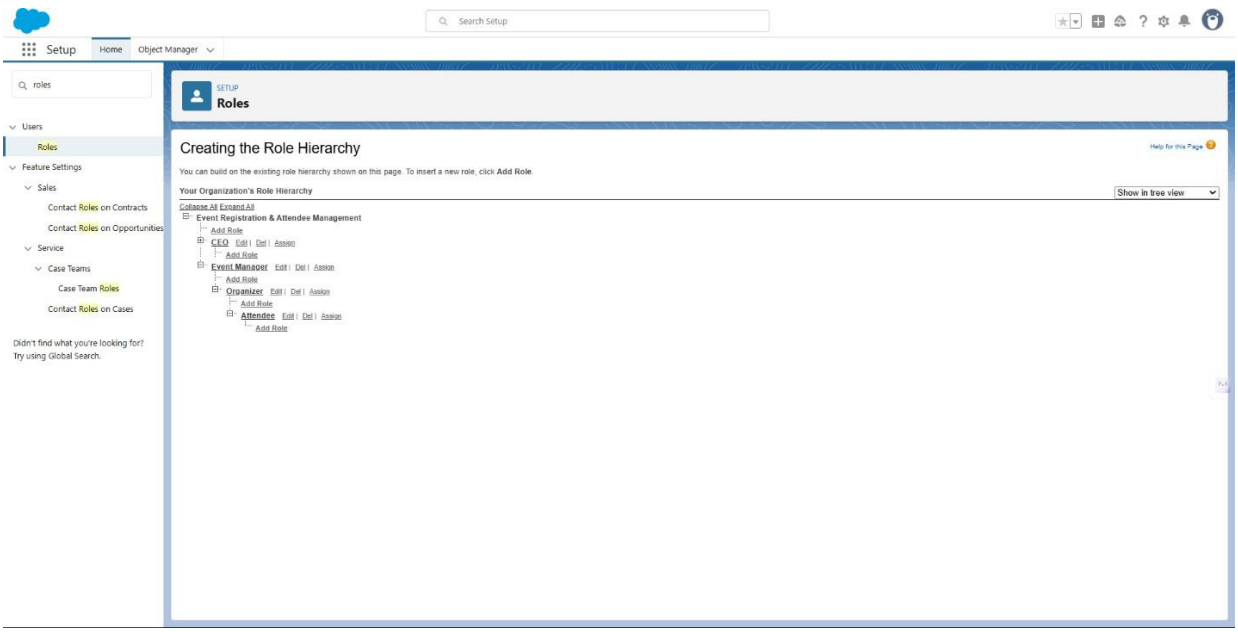
4 . **Profiles s Permission Sets**

- Cloned the “Standard User” profile to create:
    - Event Admin (full CRUD for project objects)
    - Organizer (manage own events/registrations)
    - Attendee (read-limited or registration-only, if applicable)
- Created a permission set (“Event Admin”) for fast assignment of object permissions (to be used after custom objects/tabs created).
- Set default tab visibility for all new project tabs to “Default On” for relevant profiles.




5. **Roles s User Assignment**

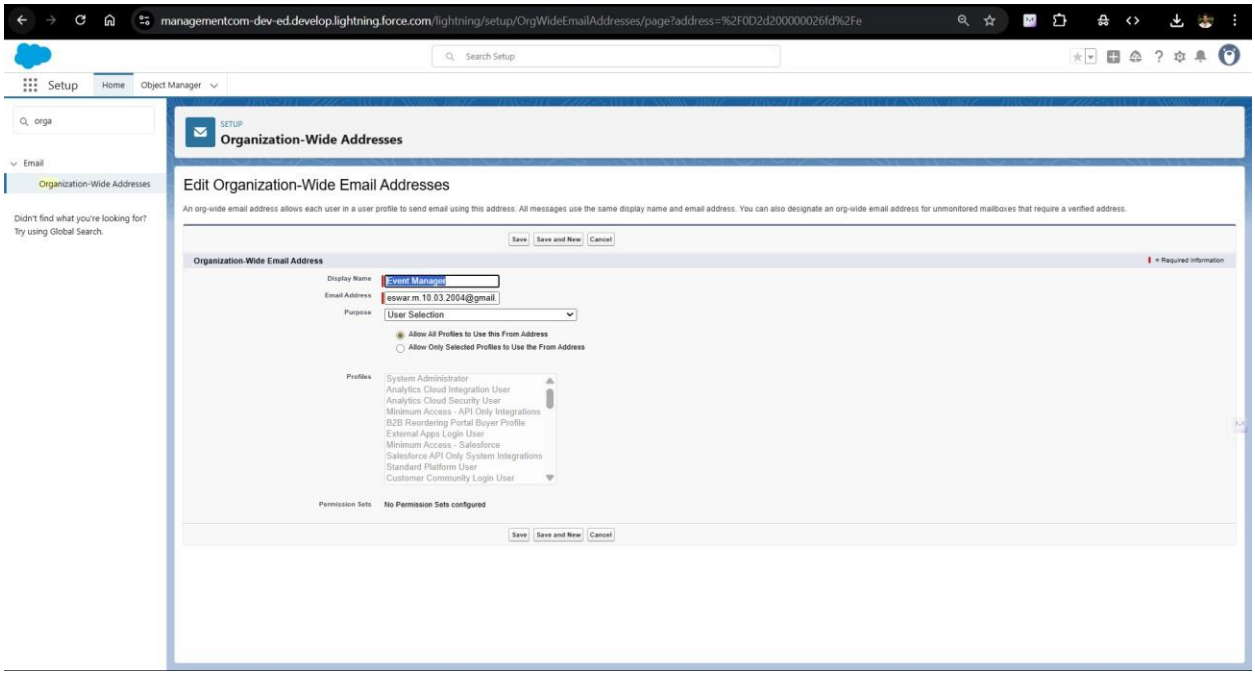
- Setup > Roles:
    - Built a simple hierarchy:
        - Event Manager (top; oversees all events)
        - Organizer (child; creates and manages assigned events)



- Setup > Users:
    - Added test users for each project role.
    - Assigned correct profile and role on each user's record.

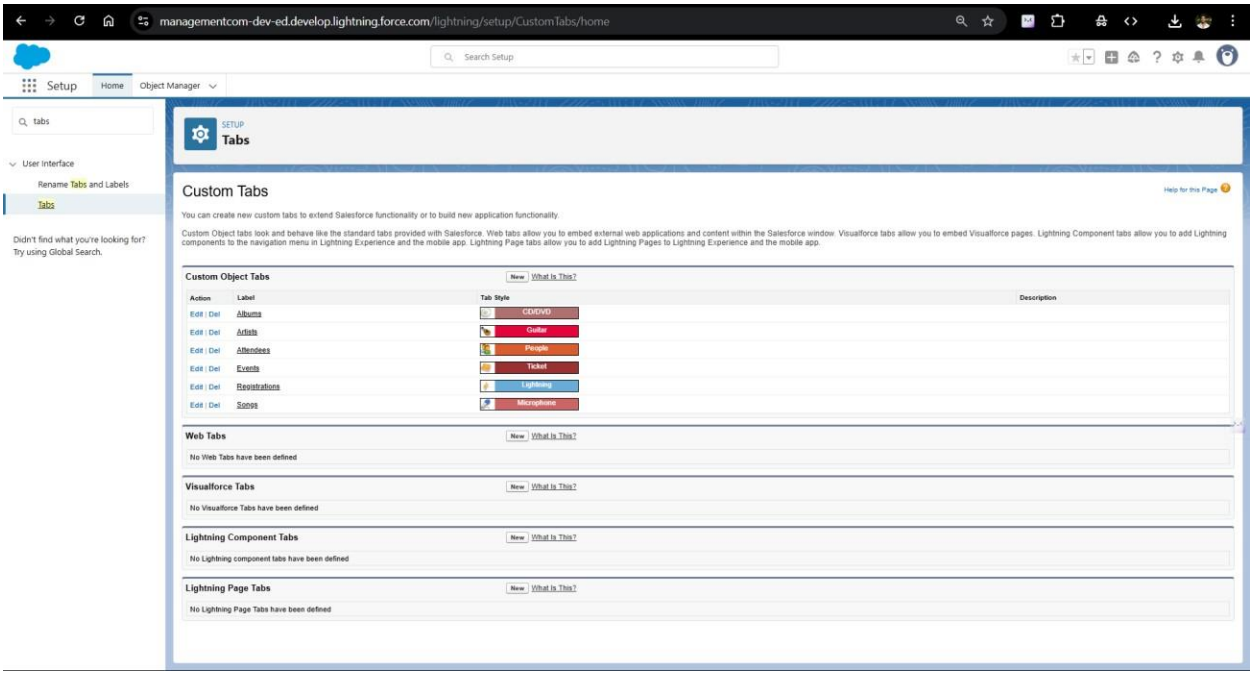
6. Org-Wide Defaults (OWD) C Sharing

- Setup > Sharing Settings:
    - Planned to set Org-Wide Defaults to Private for all project objects ("Event", "Attendee", "Registration") after creation.
    - Noted sharing rules will be added after object availability.
    - No custom login IP restrictions or password policies for initial development.



7. Custom Tabs Prep (Planned)

- Documented the plan to create tabs for:
    - Events
    - Attendees
    - Registrations



# Phase 3: Data Modeling & Object Creation

### Event Registration & Attendee Management

---

### Overview

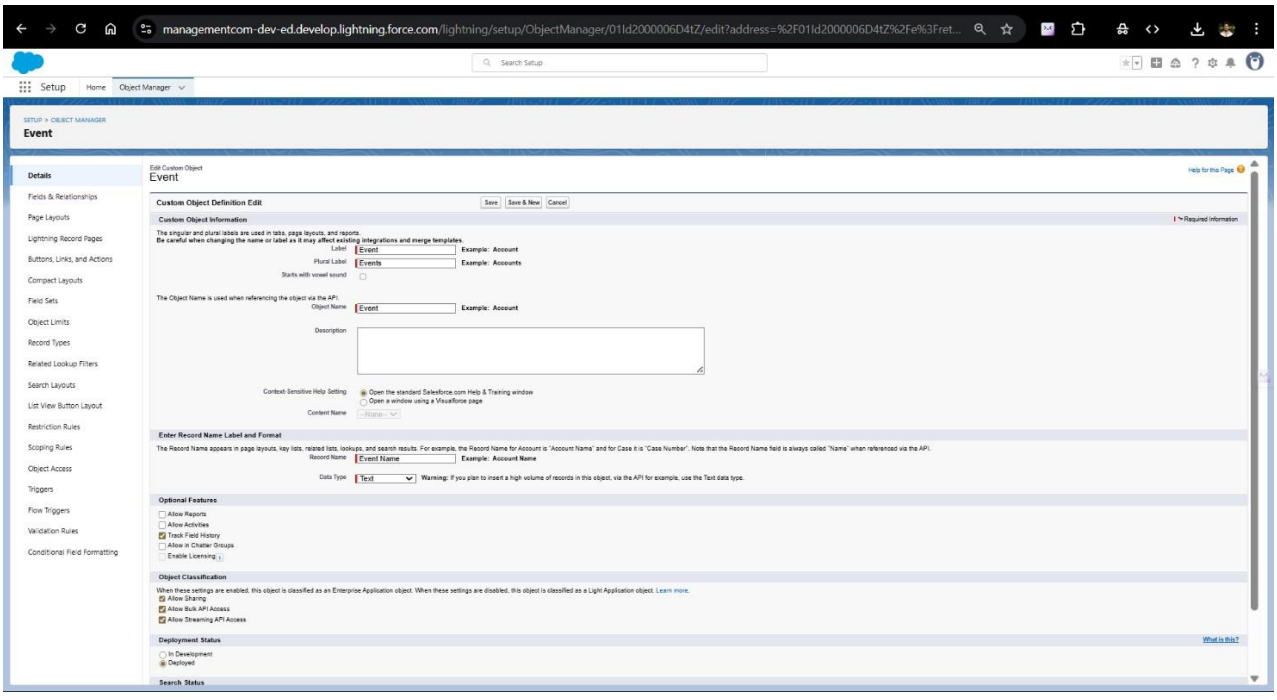
This phase covers the complete design and creation of the custom objects, fields, relationships, and page layouts for the Event Registration & Attendee Management system. Accurately modeling the data ensures the app meets business requirements, is easy to use, and supports scalability and automation.

---

### 1. Custom Object Creation

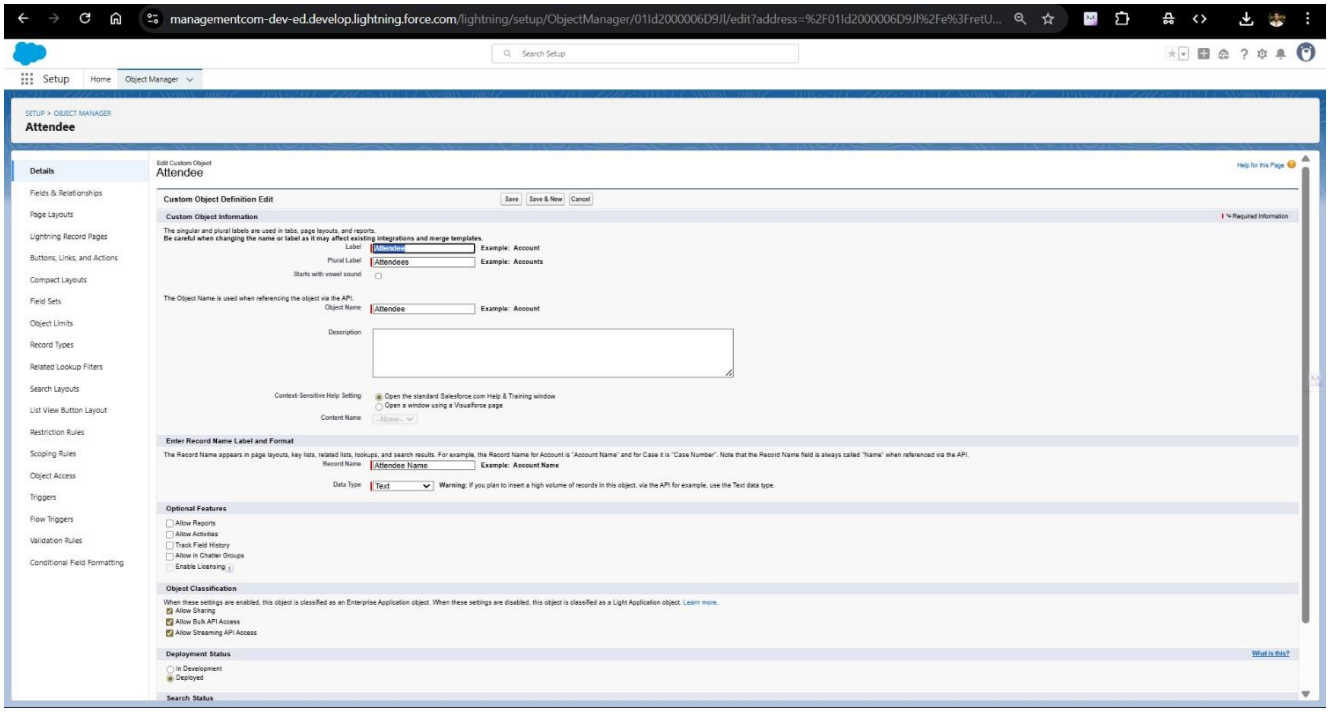
### 1.1 Create "Event" Object

- In Setup, go to **Object Manager** → *Create* → *Custom Object*.
- Enter:
    - Label: Event
    - Plural Label: Events
    - Object Name: Event__c
    - Record Name: Event Name (Text)
    - Enable Reports and Activities
- Click **Save**.



### 1.2 Create "Attendee" Object

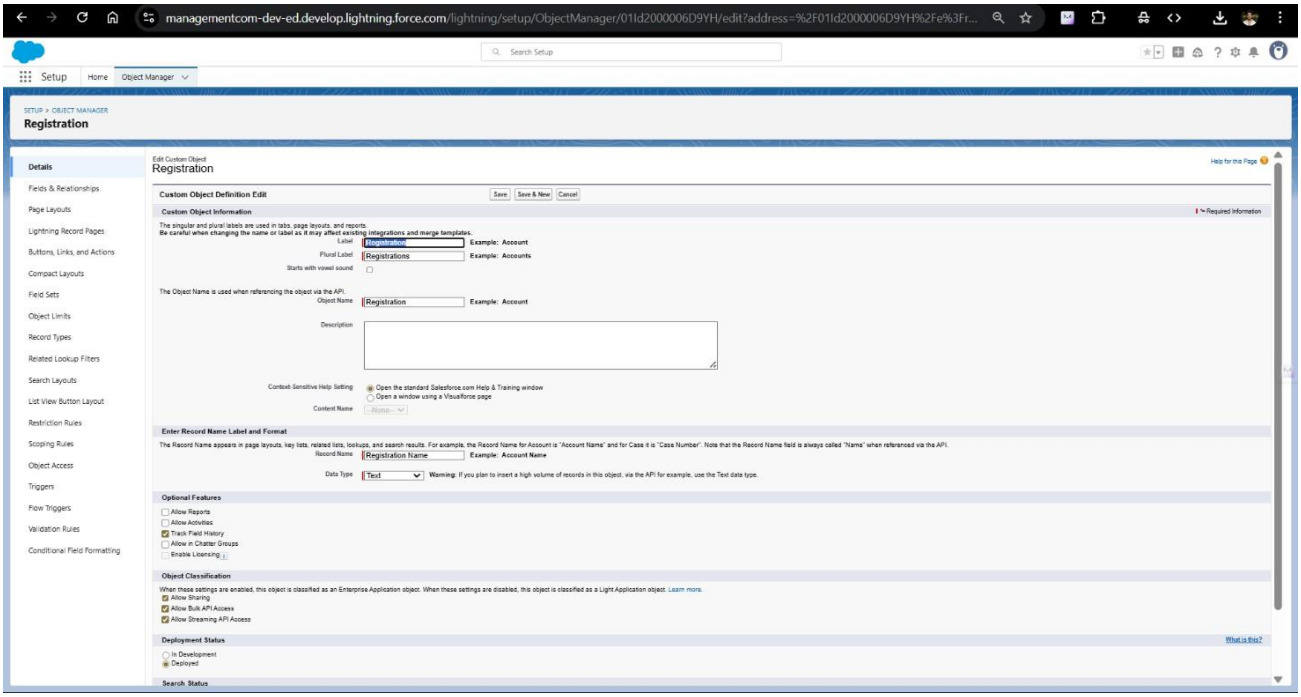
- Repeat the above steps.
- Enter:
    - Label: Attendee
    - Plural Label: Attendees
    - Object Name: Attendee__c
    - Record Name: Attendee Name (Text)
    - Enable Reports and Activities
- Click **Save.**




### 1.3 Create "Registration" Object

- Repeat the custom object wizard.
- Enter:
    - Label: Registration
    - Plural Label: Registrations
    - Object Name: Registration__c
    - Record Name: Registration Number (Auto Number or Text)

    - Enable Reports and Activities
- Click **Save**.

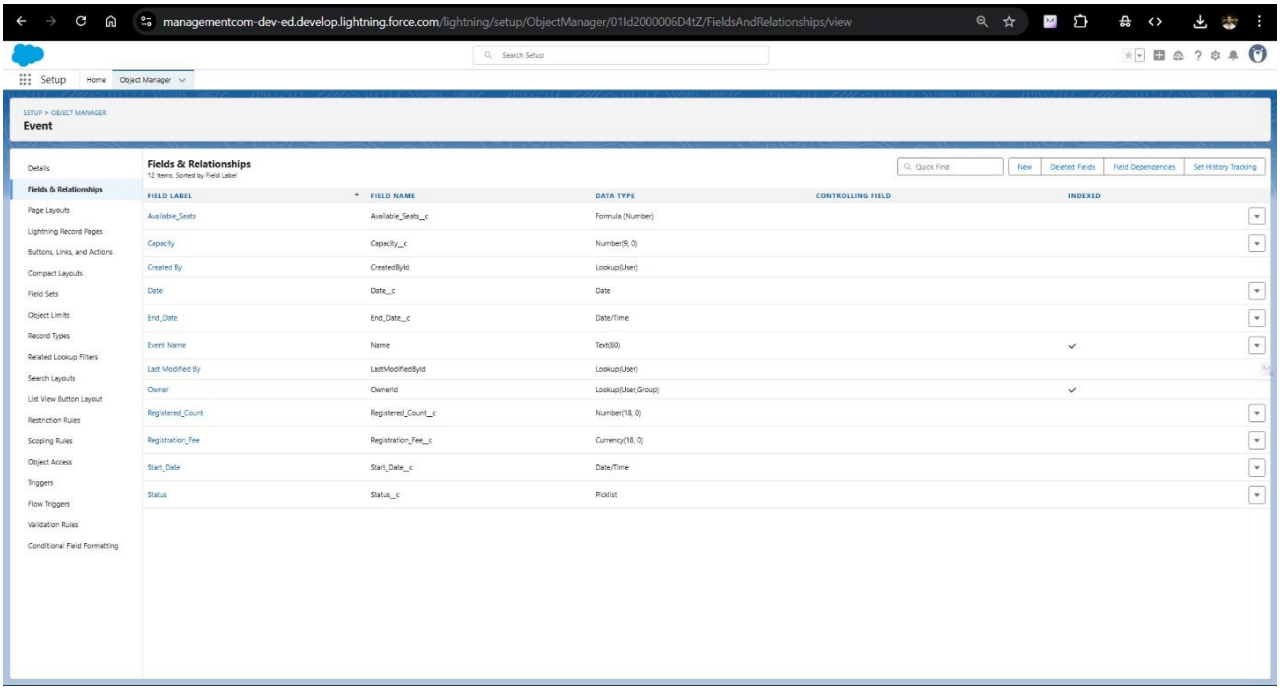



---

## 2. Add Fields to Objects

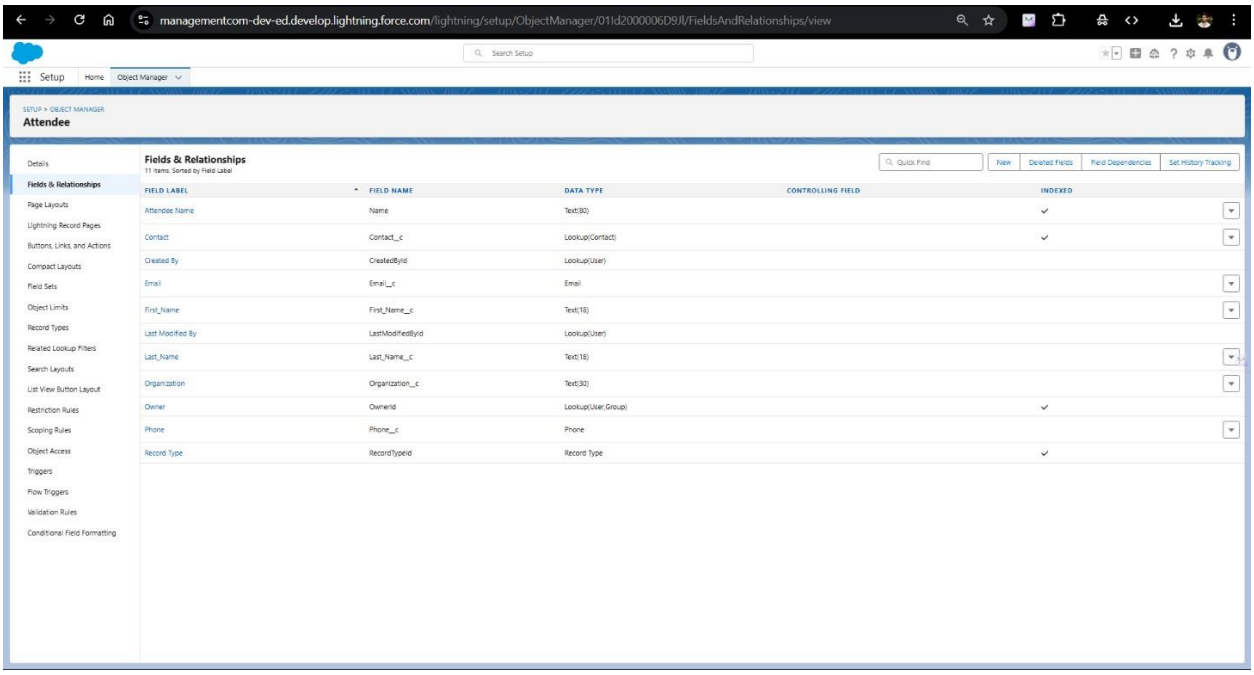
### 2.1 Event__c Fields

- **Start_Date__c** (Date/Time)
- **End_Date__c** (Date/Time)
- **Venue__c** (Text or Lookup to Venue)
- **Capacity__c** (Number, 0 decimals, default 0)
- **Registration_Fee__c** (Currency)
- **Status__c** (Picklist: Draft; Published; Closed; Cancelled)
- **Registered_Count__c** (Number, 0 decimals)
- **Available_Seats__c** (Formula): Capacity__c - Registered_Count__c
- **Registration_Open__c** (Checkbox)



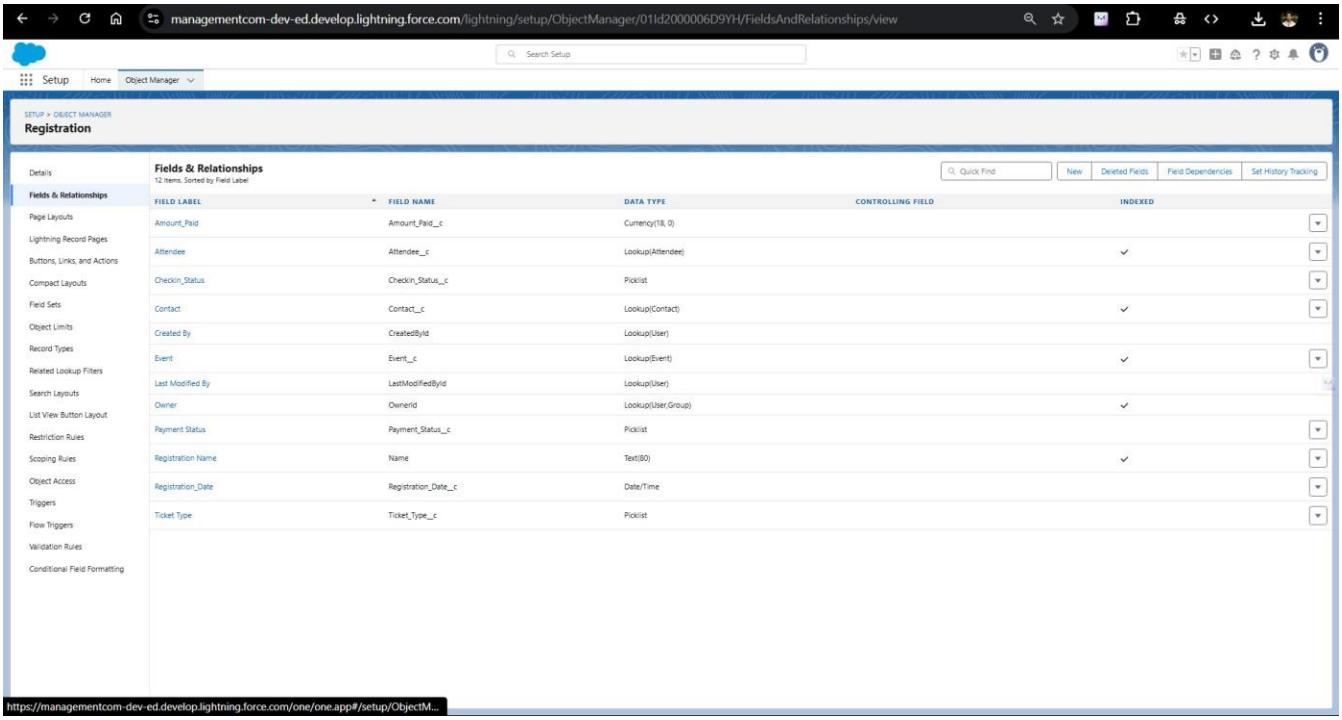

## 2.2 Attendee__c Fields

- **First_Name__c** (Text)
- **Last_Name__c** (Text)
- **Email__c** (Email)
- **Phone__c** (Phone)
- **Organization__c** (Text)
- **Contact__c** (Lookup to Contact, optional)



### 2.3 Registration__c Fields

- **Event__c** (Lookup to Event__c, required)
- **Attendee__c** (Lookup to Attendee__c, required)
- **Registration_Date__c** (Date/Time)
- **Ticket_Type__c** (Picklist)
- **Payment_Status__c** (Picklist: Pending, Paid, Failed, Refunded)
- **Amount_Paid__c** (Currency)
- **Checkin_Status__c** (Picklist: Not Checked-in, Checked-in)




---

### 3. Set up Relationships & Related Lists

- On **Registration__c**, both Event__c and Attendee__c are required lookup fields.
- On the **Event__c** page layout, ensure the “Registrations” related list is visible.
- On the **Attendee__c** page layout, ensure the “Registrations” related list is visible.

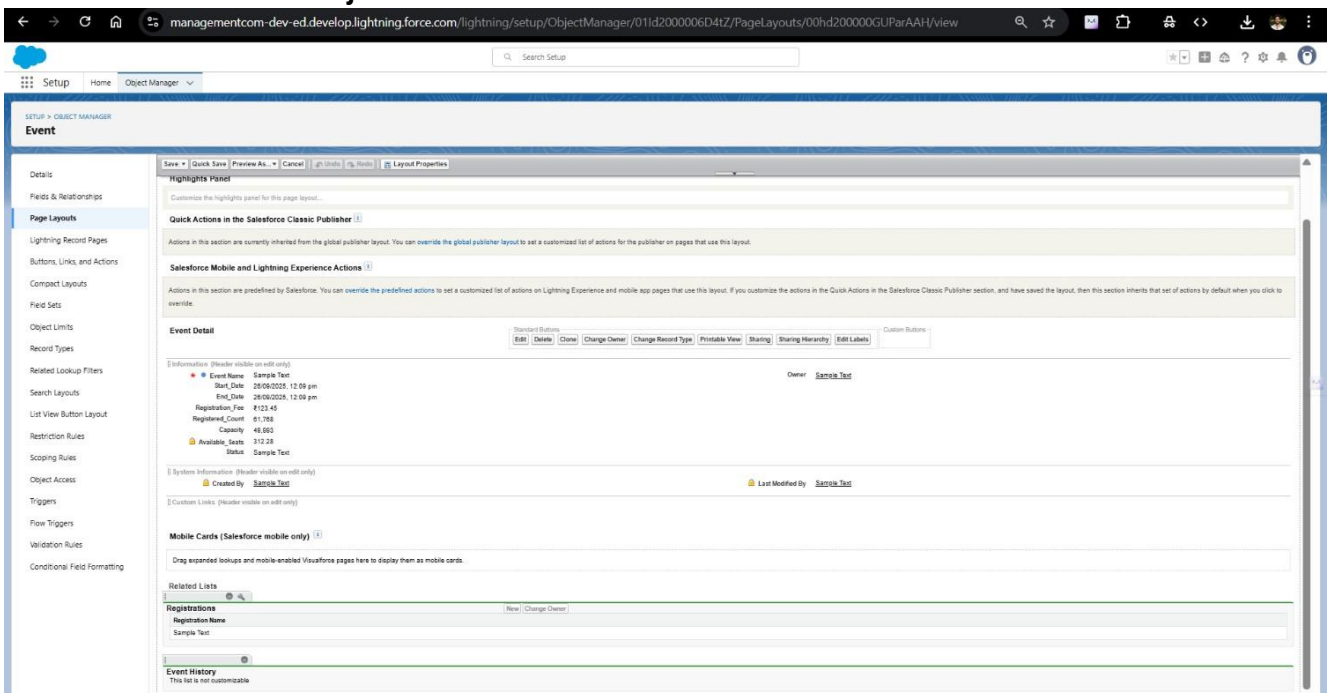
---

### 4. Page Layouts & Compact Layouts

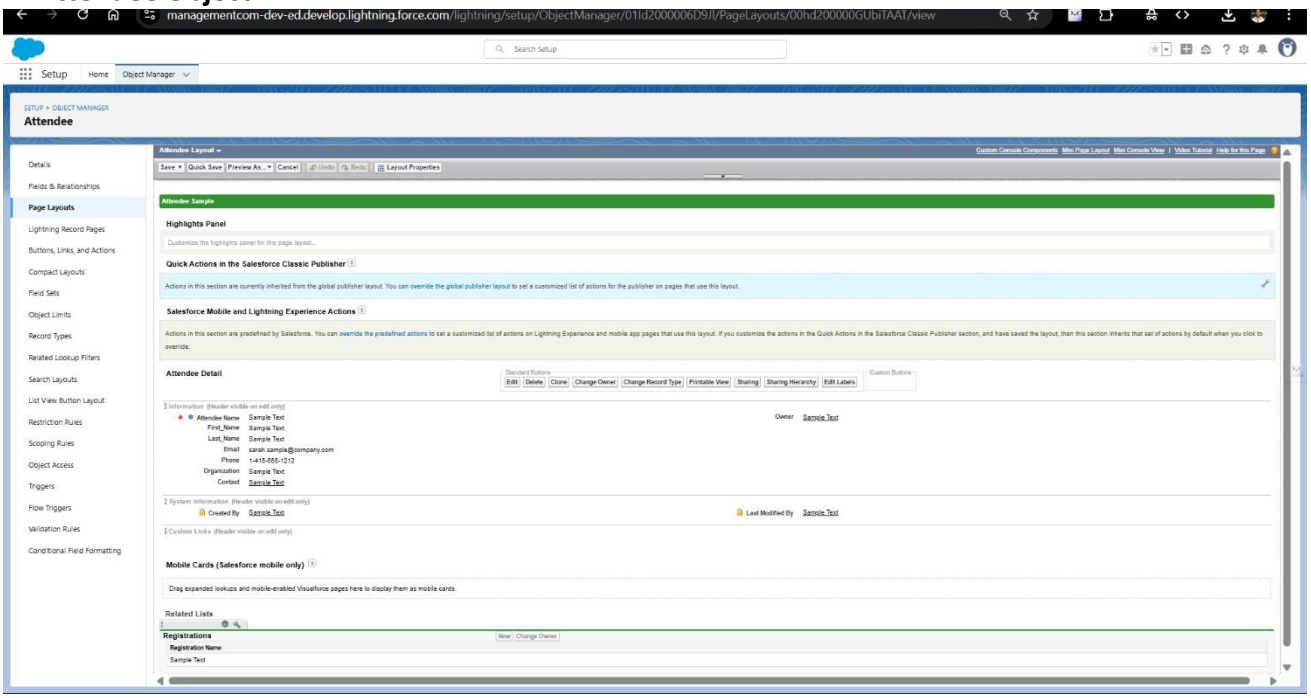
- For each object (Event__c, Registration__c, Attendee__c):
    - Edit the Page Layout.
    - Move most important fields to the top (e.g., Name, Start Date, Status for Event).

- Add related lists (e.g., Registrations).
- Save and assign the default Compact Layout to prioritize key fields (Name, Status, Date).

➢ **Event Object**

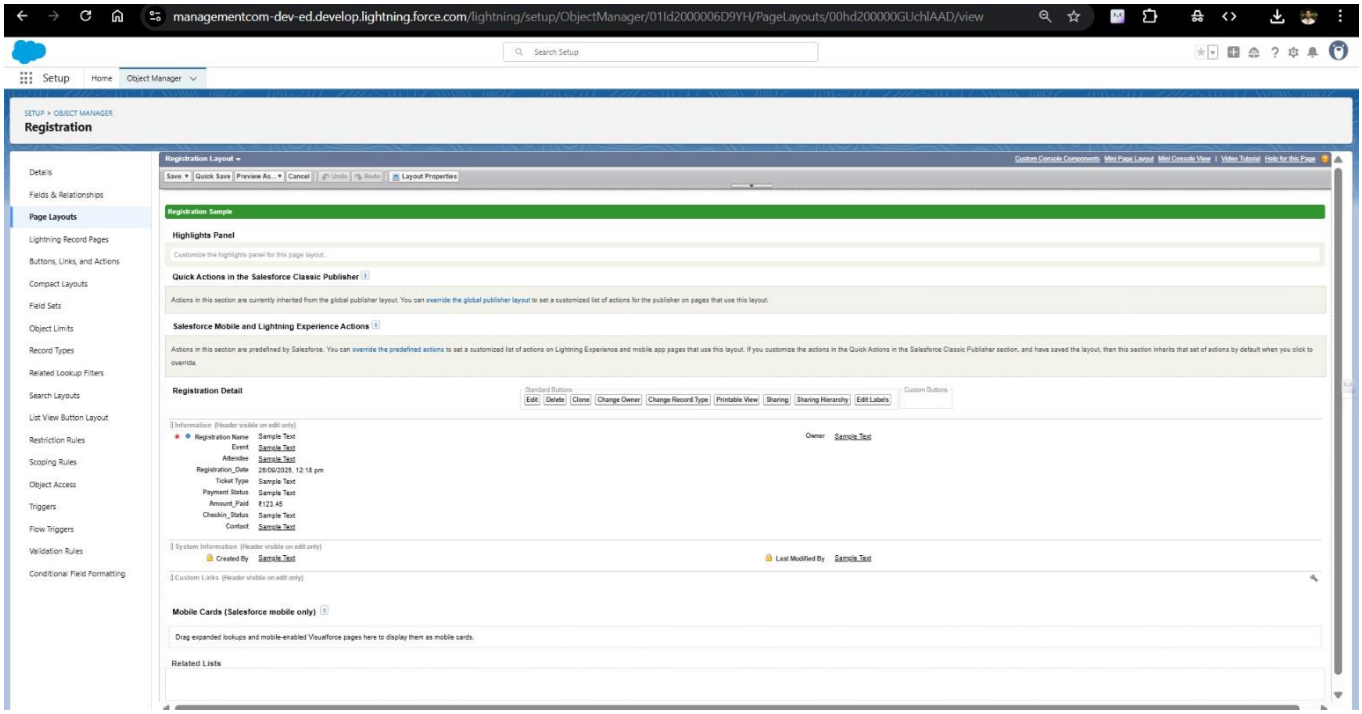



> **Attendee Object**




>

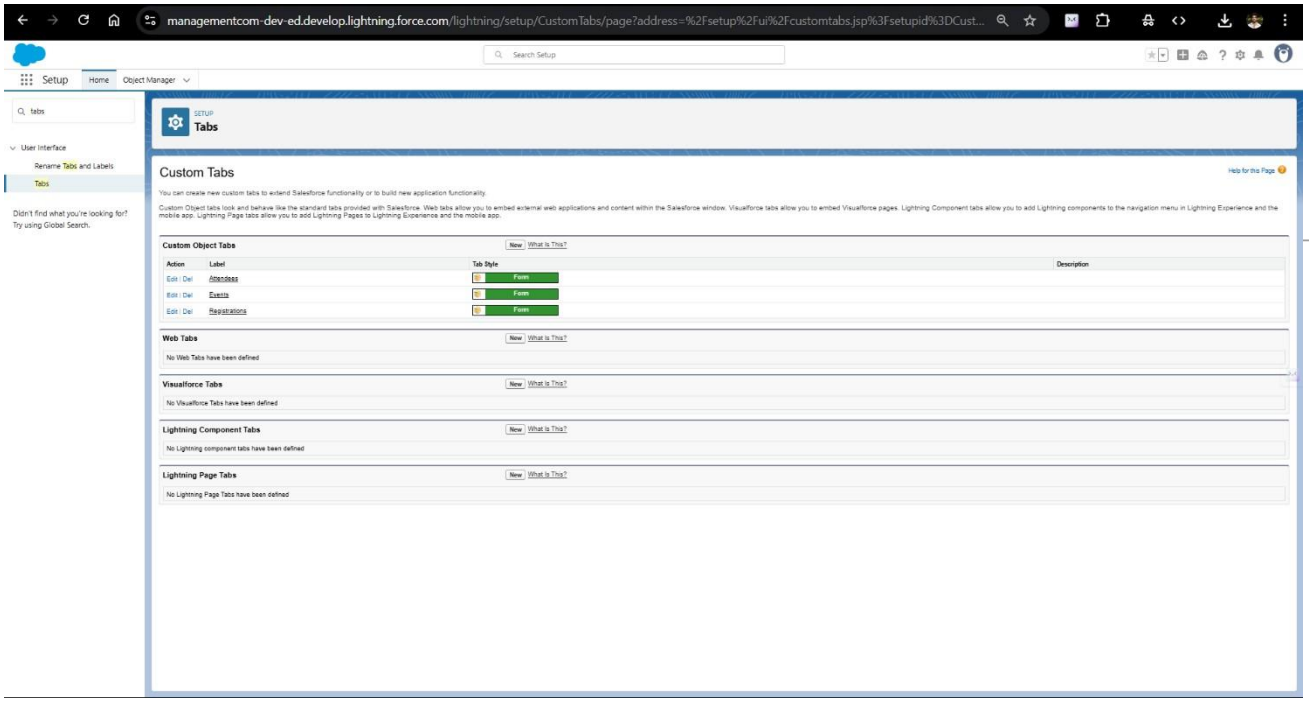
> **Registration Object**




---

## 5. Custom Tabs

- Setup → Tabs → New → Custom Object Tab.
- Create tabs for "Event", "Attendee", and "Registration" objects.
- Choose icons/styles; save each tab.
- Check tab visibility in profiles/permission sets.



**6. Object Security & Profile Permissions**

- Confirm that Event Admin/Organizer profiles have:
    - Read, Create, Edit, Delete on Event, Attendee, Registration objects.
- Set **Tab Visibility** to Default On.
- Other profiles (e.g., Attendee) get minimal or read-only access.

---

**7. Test Data Entry**

- Switch to your Event Management app (App Launcher).
- Create a sample Event, Attendee, Registration.
- Confirm field relationships, required field enforcement, and that related lists display as intended.

## Phase 4: Business Logic, Validation & Automation

---

**Overview**

This phase implements critical business rules, validation, and process automation to ensure data integrity, correct user behavior, and a seamless event registration workflow. Tools include validation rules, formulas, record-triggered flows, and lightning email templates—prioritizing declarative solutions unless Apex is necessary.

---

**1. Validation Rules & Formulas**

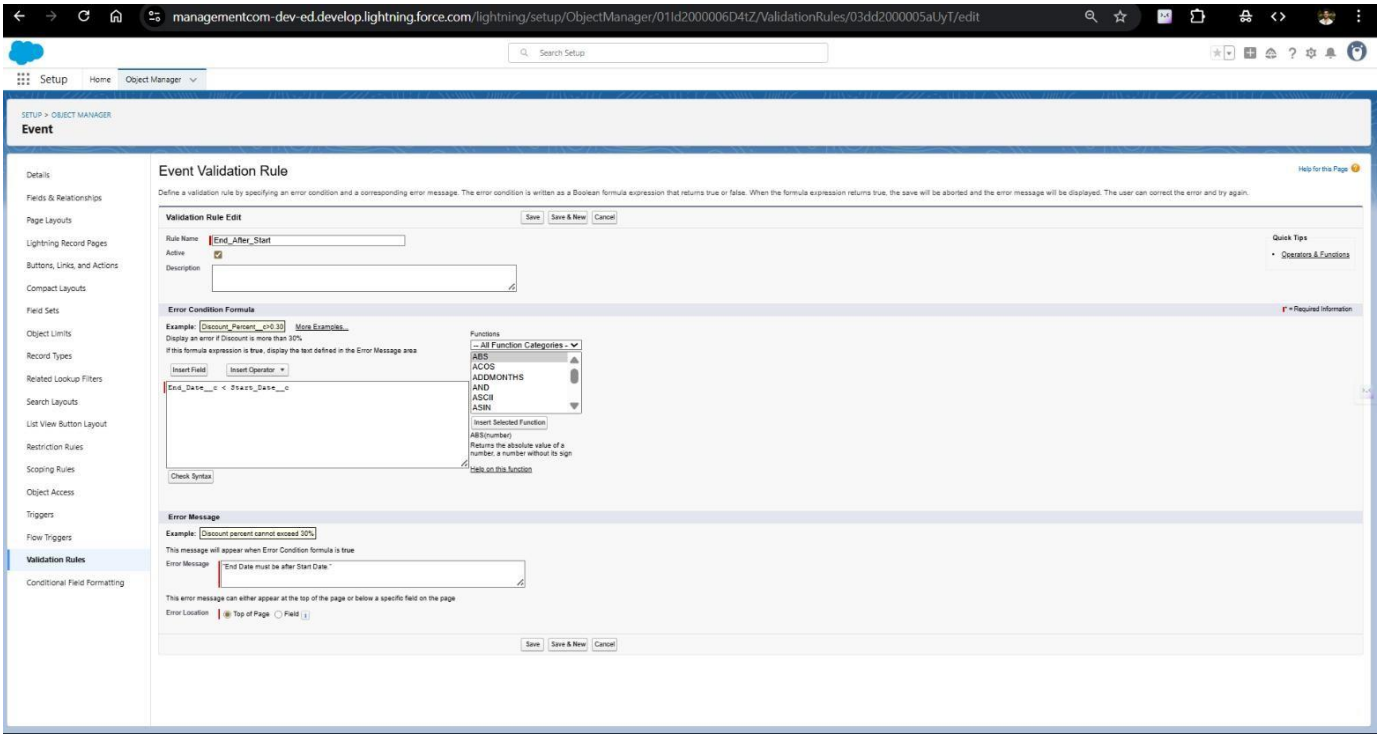
**1.1 Event End Date Must Be After Start Date**

- **Location:** Object Manager → Event__c → Validation Rules → New.
- **Rule Name:** End_After_Start
- **Error Condition Formula:**

text

End_Date__c < Start_Date__c

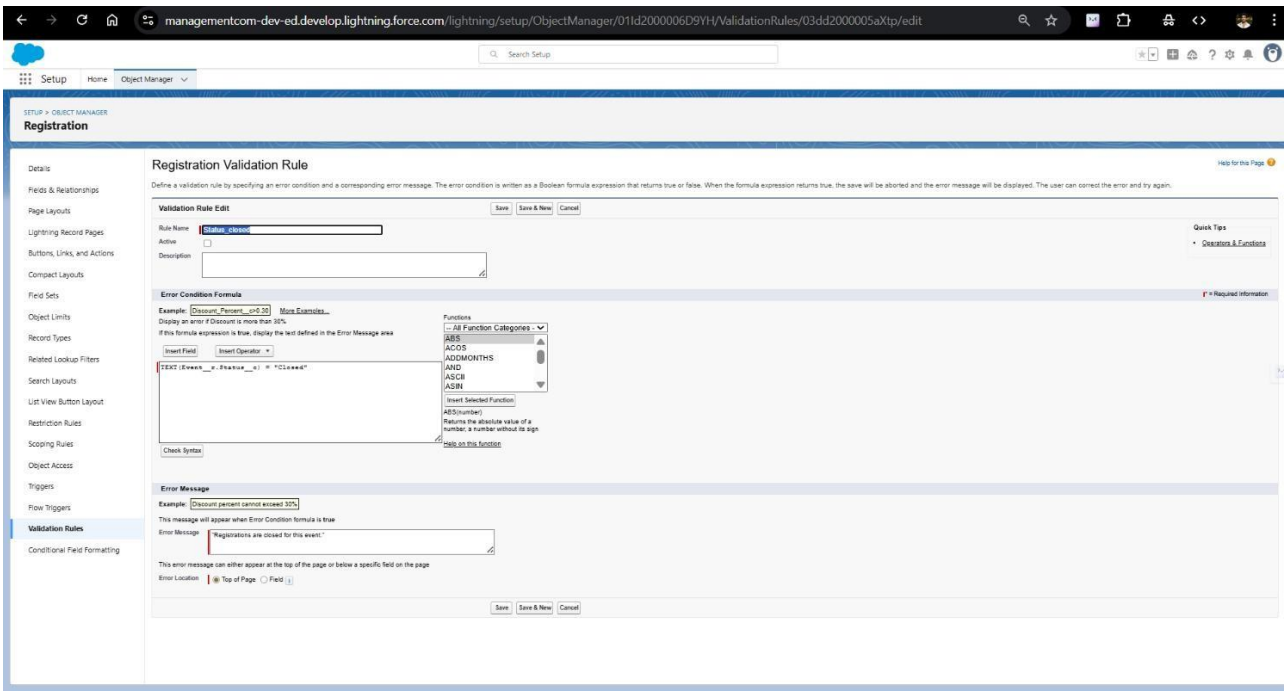
- **Error Message:** "End Date must be after Start Date."
- **Usage:** Prevents user error in scheduling events.




---

## 1.2 Registration Prohibited When Event Closed

- **Location:** Object Manager → Registration__c → Validation Rules → New.
- **Formula:** Event__r.Status__c = "Closed"
- **Error Message:** "Registrations are closed for this event."
- **Usage:** Ensures no registrations when event is not accepting new attendees.



### 1.3 Formula Fields

- **Available_Seats__c** on Event__c:

text

Capacity__c - Registered_Count__c

- **Usage:** Provides real-time feedback on event capacity to users/admins and automations.

---

### 2.1 Flow: Update Registered_Count__c

**Purpose**

- Automatically maintain a running total of confirmed registrations per event and enforce event closure when full.

**Type**

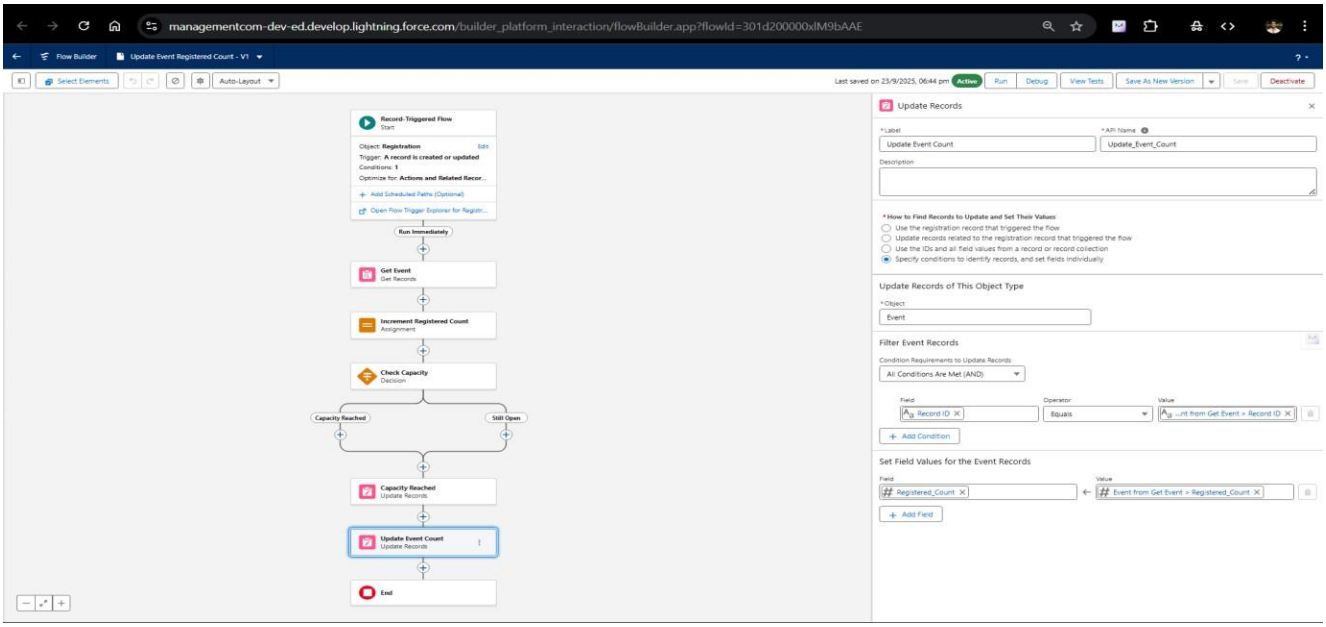
- Record-Triggered Flow

**Trigger:**

- When a Registration__c record is created, updated, or deleted.

**Logic:**

1. If Payment_Status__c transitions to "Paid" (from another status), increment Event__c.Registered_Count__c by one.
2. If Payment_Status__c changes from "Paid" to "Refunded" or if the registration is deleted, decrement Event__c.Registered_Count__c.
3. If Registered_Count__c is now greater than or equal to Capacity__c, update Event__c.Status__c to "Closed" and (optionally) alert an admin.



### 2.2 Flow: Email Confirmation

**Purpose**

- Instantly notifies the attendee of their successful registration and/or payment for transparency and engagement.

**Type**

- Record-Triggered Flow
  (Object: Registration__c, After Insert or Payment_Status__c update)

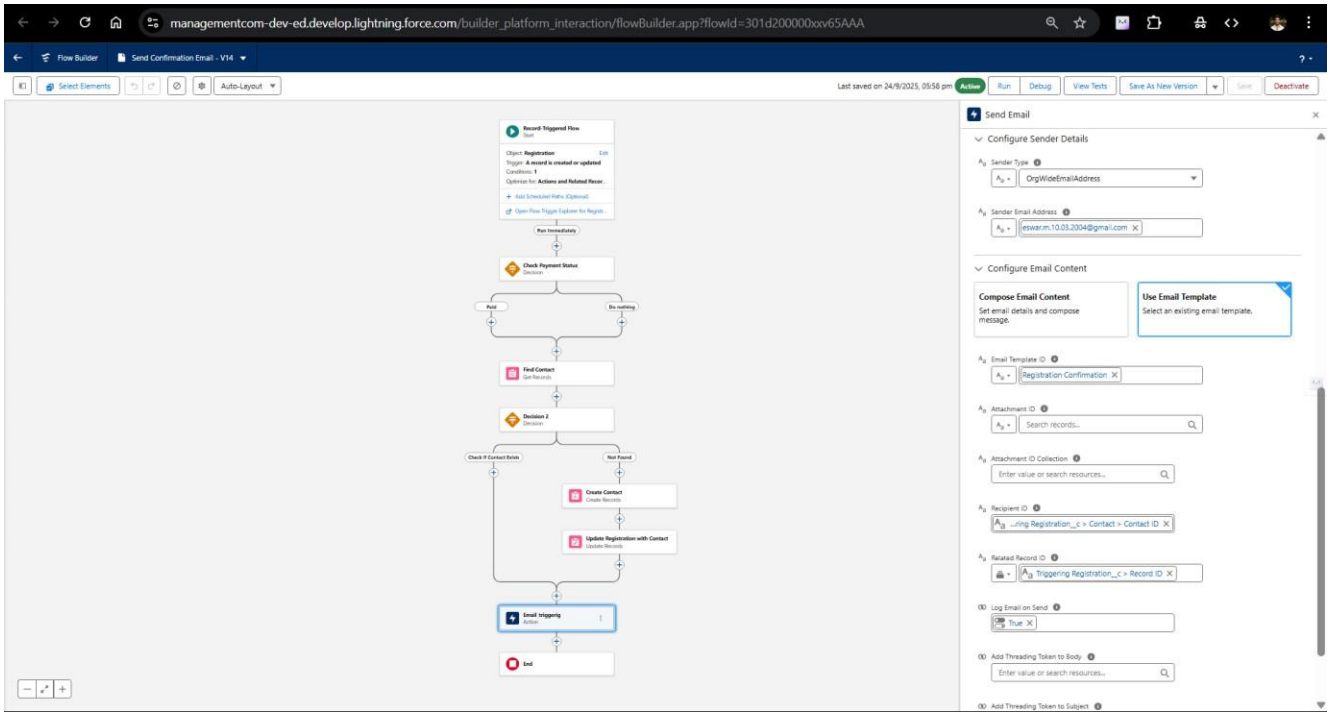
**Process Breakdown**

**Trigger:**

- New Registration__c is created, or Payment_Status__c is set to "Paid".

**Logic:**

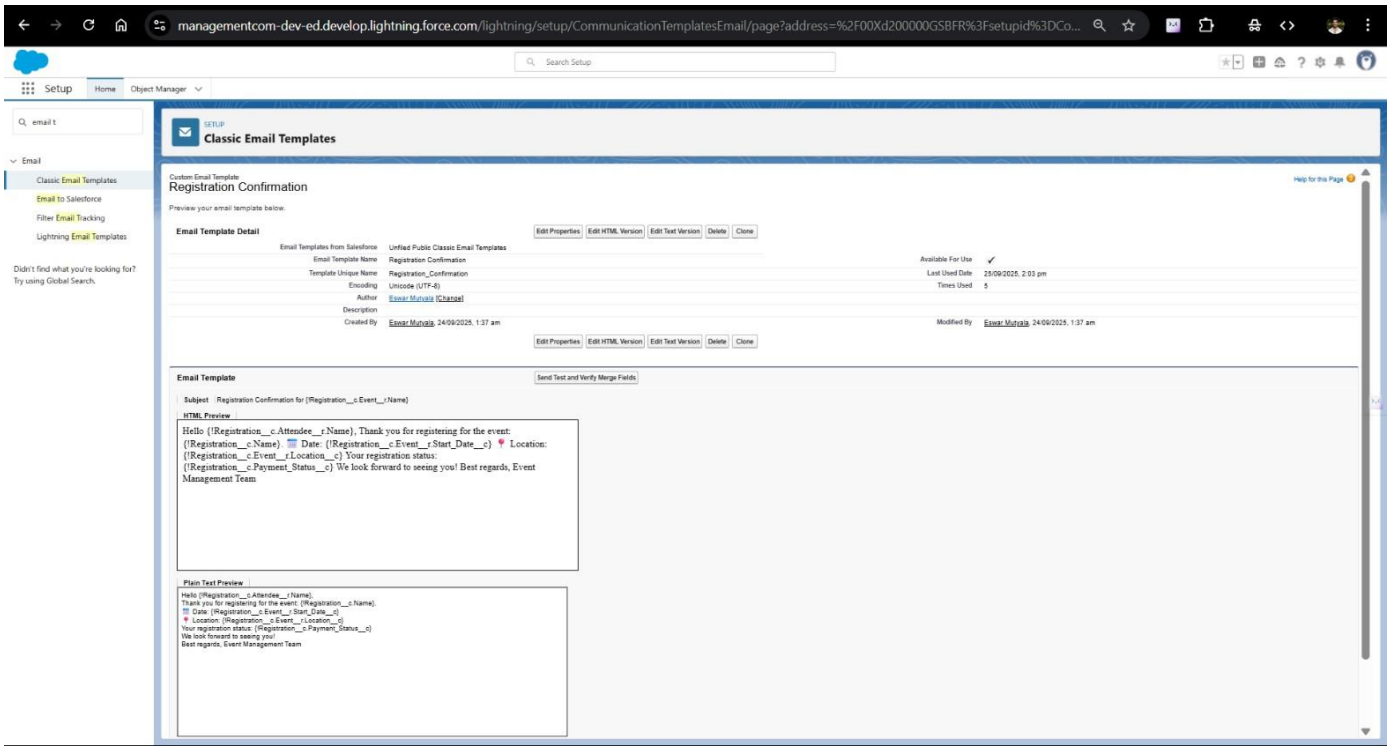
1. Gather information from Registration__c, Attendee__c, and Event__c.
2. Use a Lightning Email Template—populating necessary fields with merge data.
3. Send the template-based email to the attendee's email address.



## 3. Email Templates & Notifications

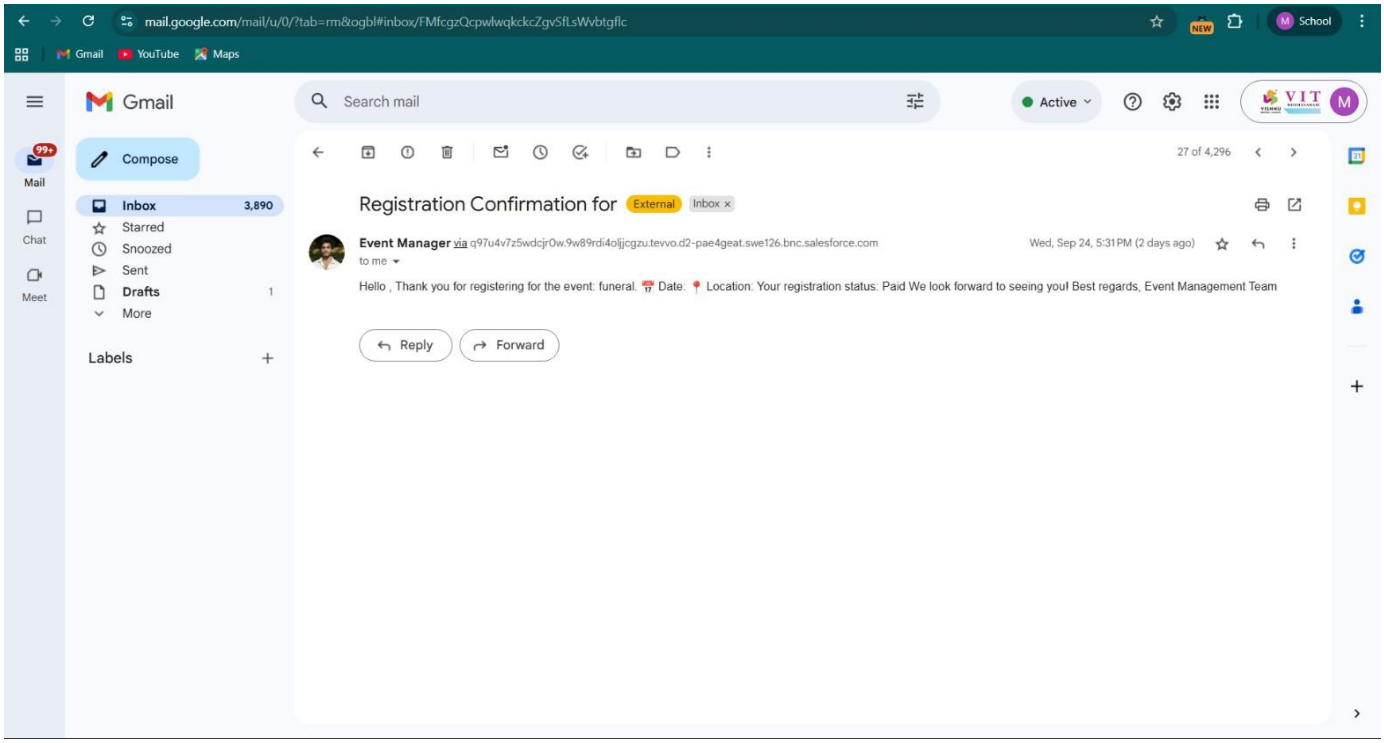
### 3.1 Setup

- **Path:** Setup → Email Templates → New Email Template (Lightning).
- **Template:** Registration Confirmation
- **Merge Fields:** Event Name, Attendee First/Last Name, Date, etc.
- **Use:** Referenced by confirmation flow for automatic attendee notification.



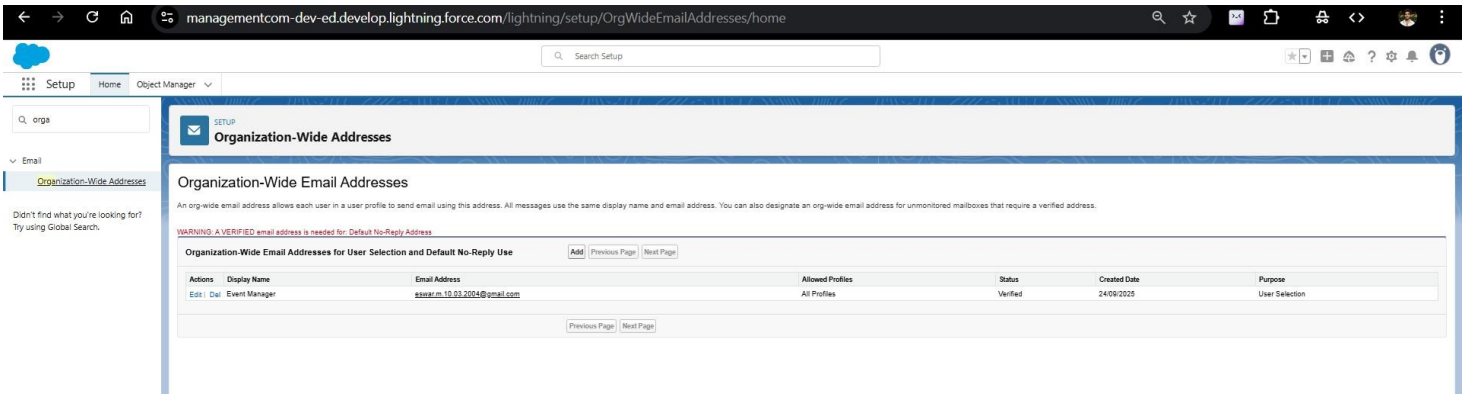

## 4. Testing & Iteration

- **Test Cases:**
  - Create event, register attendee (capacity available): should succeed, confirmation email sent.
  - Register when event is full: should block registration, with error.
  - Change payment status to Paid: Registered_Count__c should increment.
  - Refund/Cancel registration: Registered_Count__c should decrement.
- **Reporting:** Ensure Available_Seats__c and Registered_Count__c always reflect accurate numbers.
- **Iterate on flows/rules as needed after initial tests.




**5. Security & Permissions (in context)**

- Confirm flow and field security for each profile:
    - Only Event Admins can change status or capacity.
    - Attendees can only register (not directly change event data).
- Check OWD/sharing settings support automation.




# Phase 5: Apex Programming

## RegistrationController

### Overview

In this phase, we implement the core backend logic for event registration using Apex. The main class, RegistrationController, is exposed as an AuraEnabled method to support Lightning Web Components (LWCs) or other UI integrations by creating and managing Attendee and Registration records programmatically.

**Apex Class: RegistrationController**

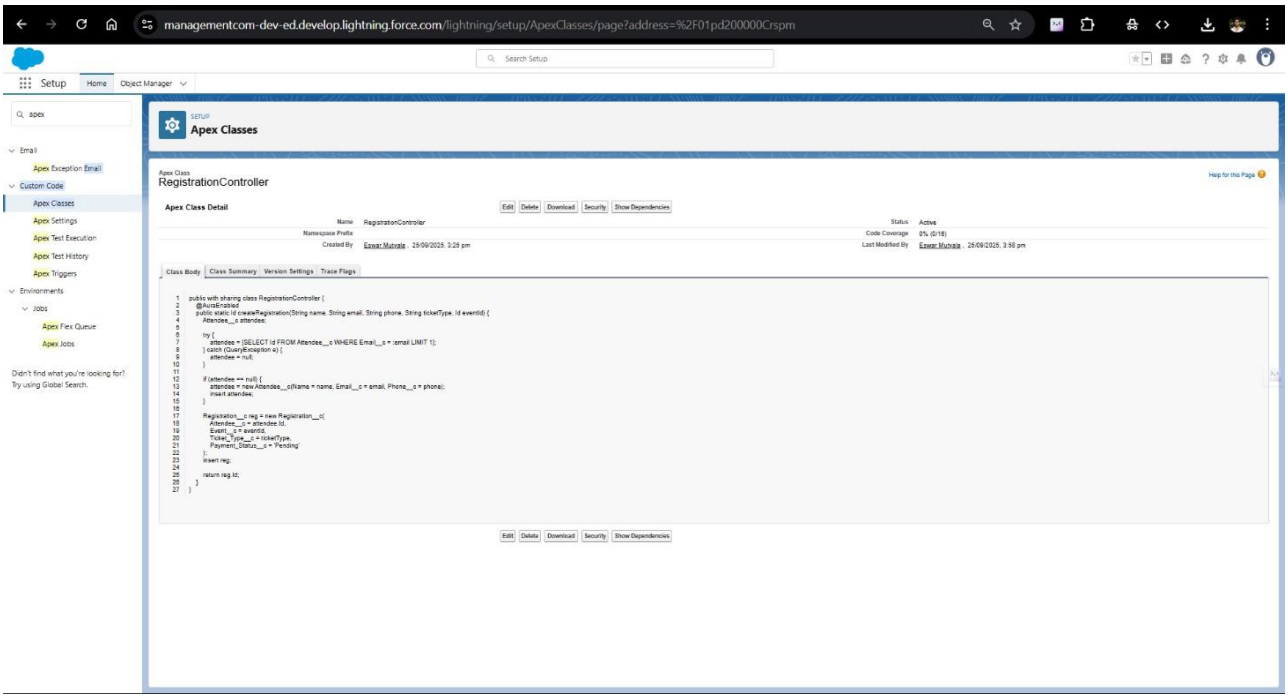
**Purpose**

- This Apex class handles attendee registrations in code.
- It finds or creates an Attendee record based on email.
- Creates a Registration record associated with the Event and Attendee.
- Sets the initial registration Payment Status as 'Pending'.
- Returns the Registration record Id for further use by the UI or calling components.

**Code Summary**

```
public with sharing class RegistrationController {
    @AuraEnabled
    public static Id createRegistration(String name, String email, String phone, String ticketType, Id eventId) {
        Attendee__c attendee;
        try {
            attendee = [SELECT Id FROM Attendee__c WHERE Email__c = :email LIMIT 1];
        } catch (QueryException e) {
            attendee = null;
        }
        if (attendee == null) {
            attendee = new Attendee__c(Name = name, Email__c = email, Phone__c = phone);
            insert attendee;
        }
        Registration__c reg = new Registration__c(
            Attendee__c = attendee.Id,
            Event__c = eventId,
            Ticket_Type__c = ticketType,
            Payment_Status__c = 'Pending'
        );
        insert reg;
        return reg.Id;
    }
}
```

Apex Class(RegistrationController)




**Usage in Project**

- Called by Lightning Web Component (registrationForm) to create registrations from user input.
- Enables registration processing through Lightning UI and automates record creation.
- Forms the basis for further Apex enhancements such as capacity checking or payment integration

## Phase 6: User Interface Development

---

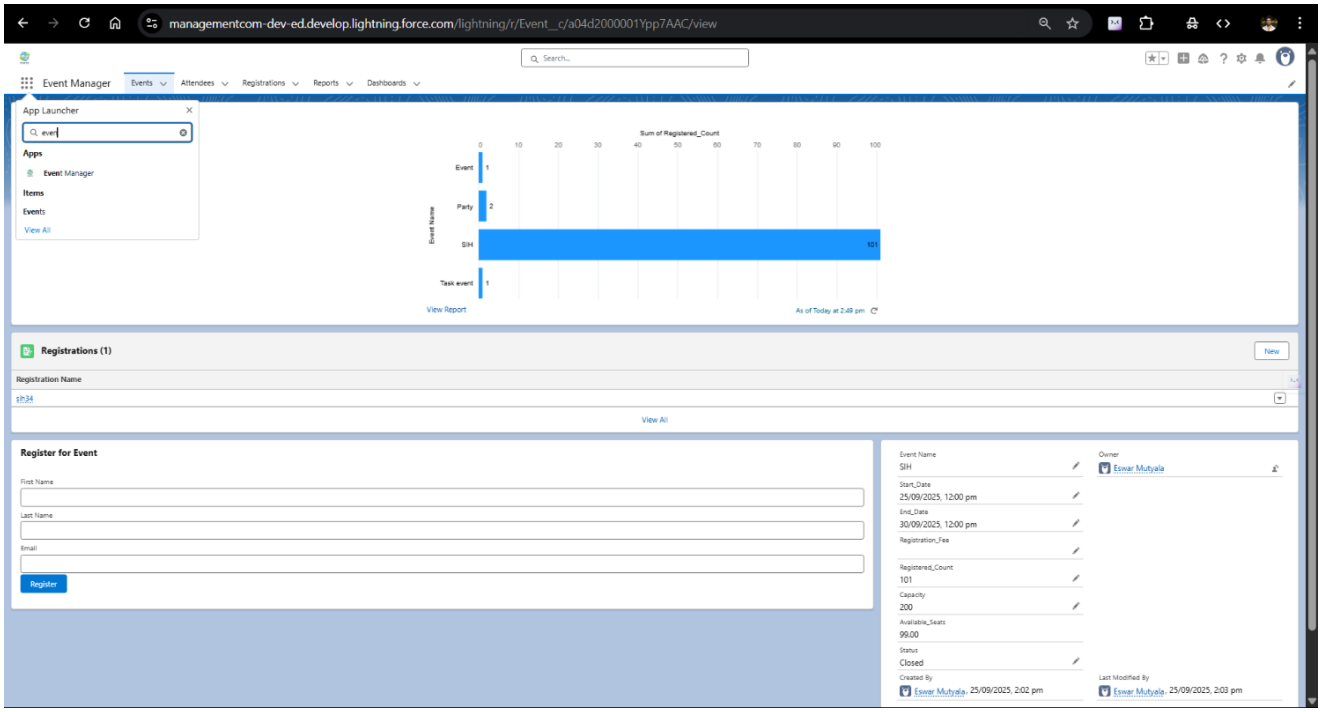
**Overview**

In this phase, I developed a user-friendly Lightning Experience interface enabling event management and attendee registration via intuitive navigation and interactive components. The process involved creating a dedicated Lightning App, customizing record pages, building Lightning Web Components integrated with Apex, and configuring tabs and utility features to enhance usability.

---

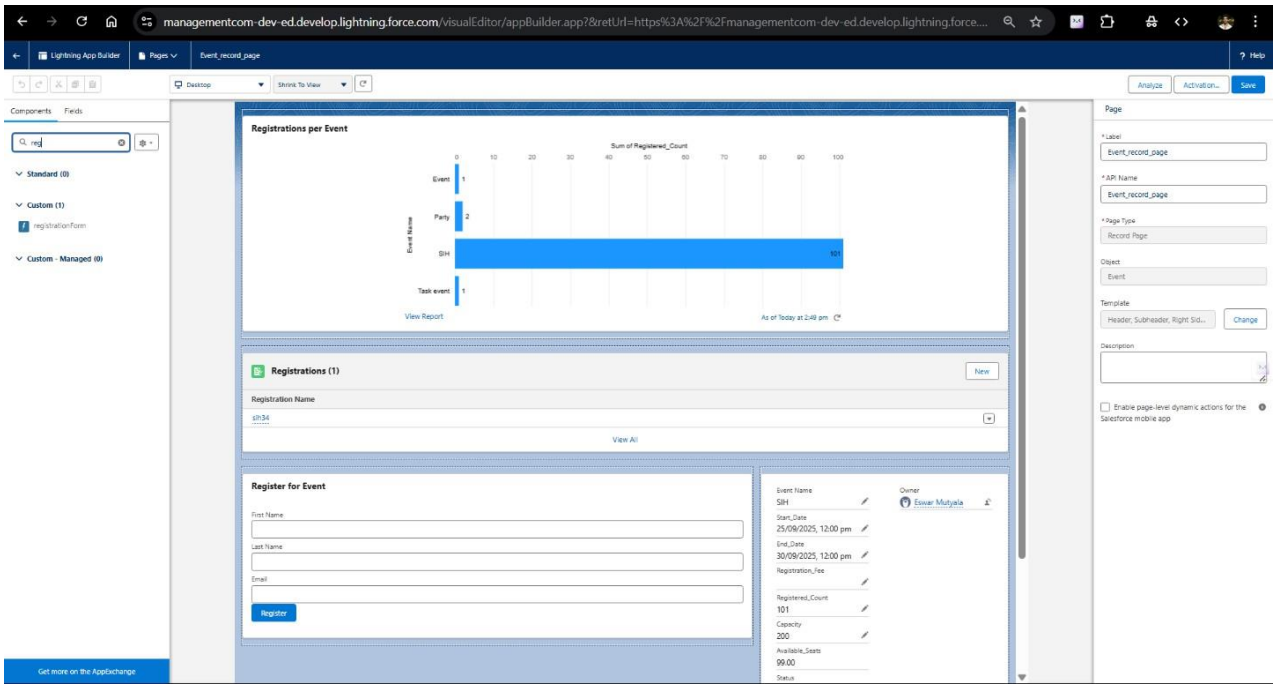
## 1. Created Lightning App - Event Manager

- Navigated to Setup → App Manager → New Lightning App.
- Defined the app as "Event Manager," with appropriate branding and standard navigation.
- Added custom object tabs for Event__c, Registration__c, and Attendee__c, along with Reports to the app's navigation.
- Assigned app visibility to my System Administrator profile.
- Saved and activated the app for use.




## 2. Customized Lightning Record Page for Event__c

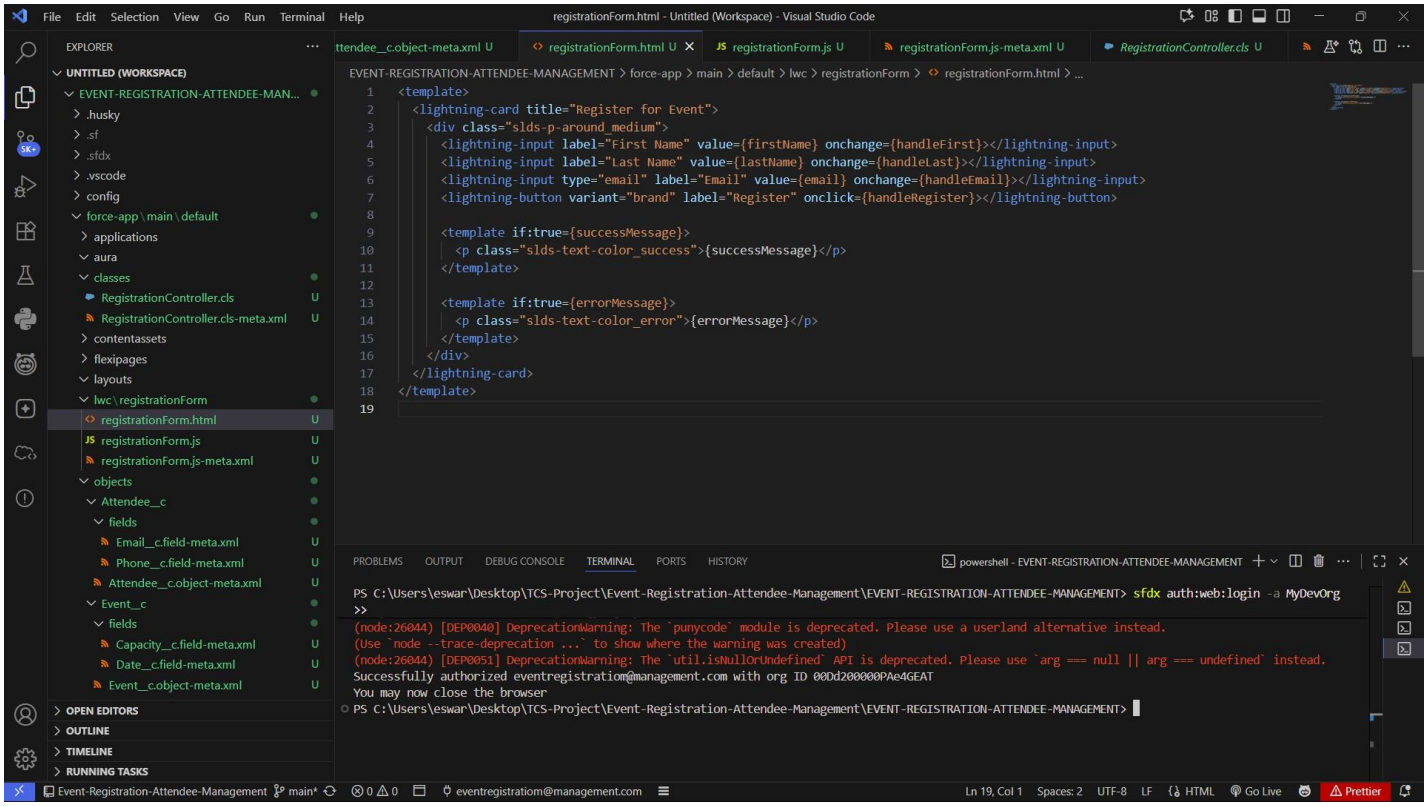
- Opened an Event record and selected **Edit Page** in the Lightning App Builder.
- Added the **Record Detail** component to display key event fields including Name, Capacity, Status, and Dates.
- Added a **Related List - Single** to show related Registrations, allowing quick access to attendees linked to the event.
- Incorporated a chart component to visually represent Registered_Count__c versus Capacity__c.
- Saved and activated this page layout; assigned as the default Lightning Record Page for the Event object.



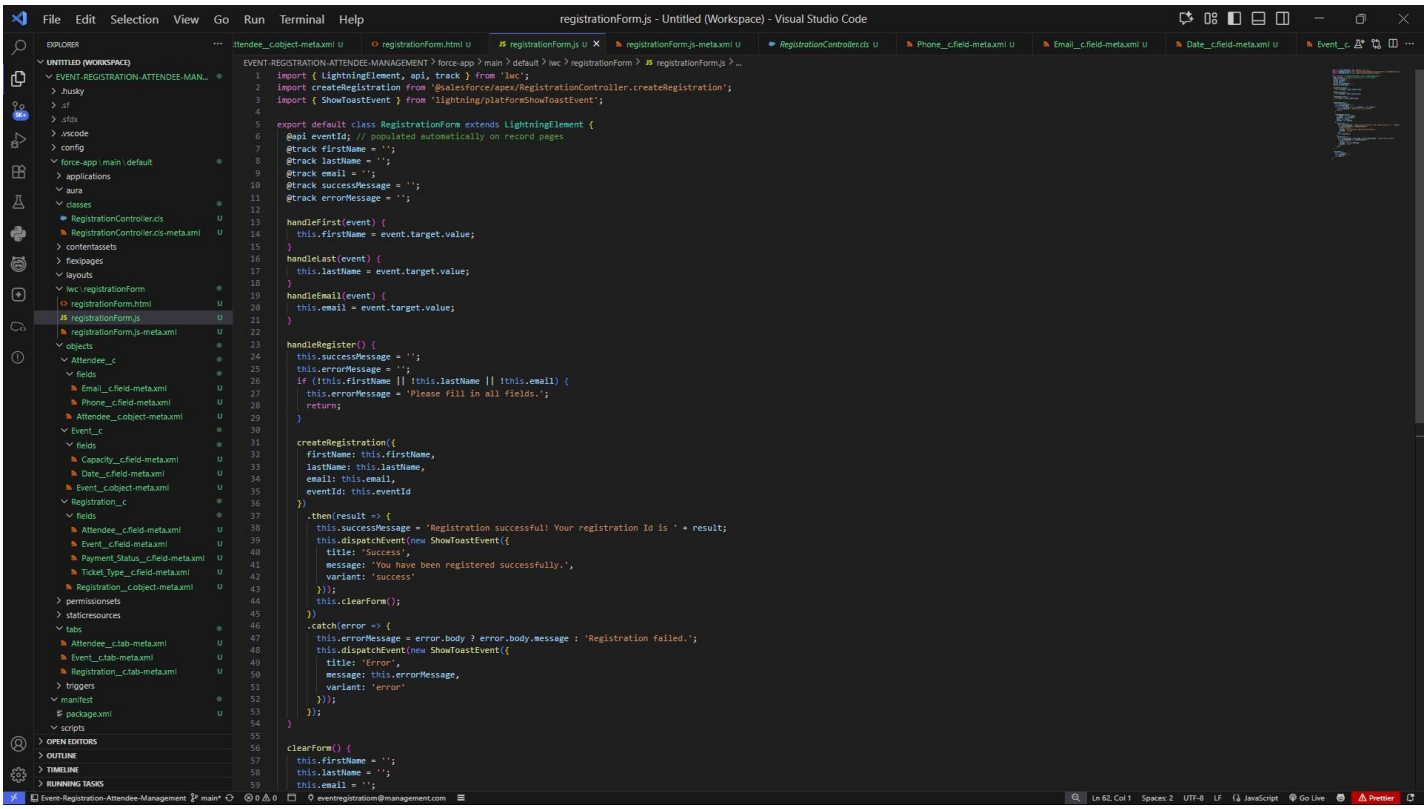

### 3. Developed Lightning Web Component: registrationForm

- Created the registrationForm LWC using Salesforce CLI and VS Code.
- The component UI includes:
    - Inputs for Full Name, Email, Phone.
    - Ticket Type dropdown with options Standard, VIP, Student.
    - Submit button for registration.
- Managed state and input handling in registrationForm.js.
- Implemented imperative Apex calls to backend RegistrationController class to:
    - Find or create Attendee records.
    - Create a linked Registration record with Payment_Status__c defaulted to 'Pending'.
- Handled success and error feedback via Lightning toast notifications.
- Placed the registrationForm component on the Event Lightning Record Page using App Builder.
- Activated changes, enabling inline attendee registration from the Event page.

registrationForm.html




registrationForm.js

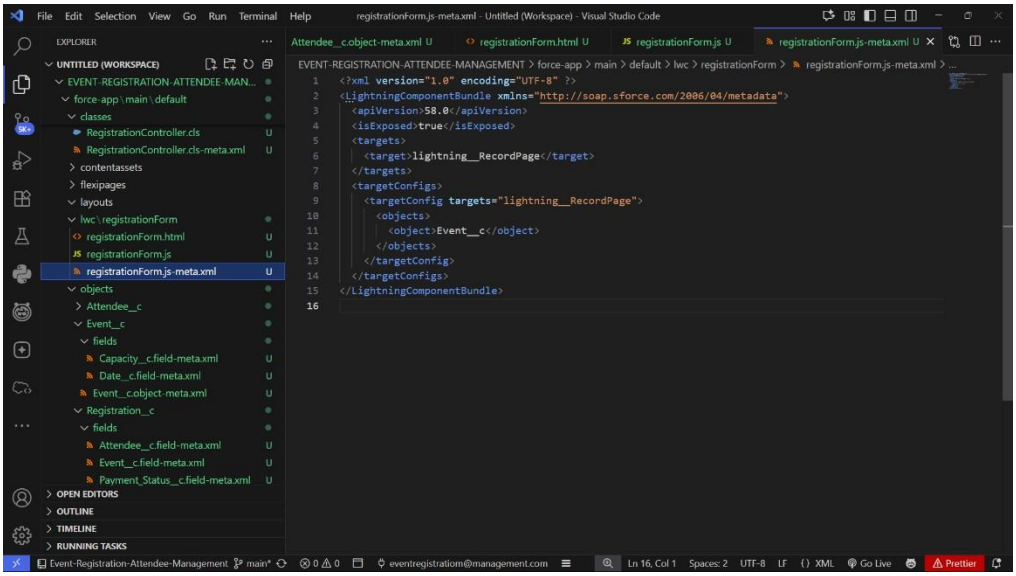


**4. Custom Objects & Fields: SFDX Source-Driven Development**

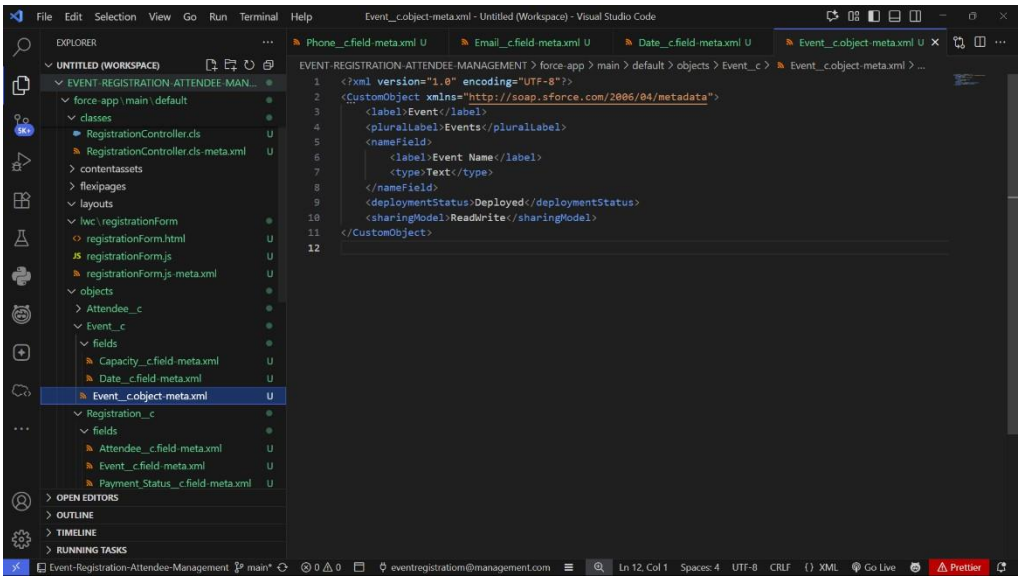
Instead of relying solely on Object Manager, I defined and managed all core objects and fields as metadata using Salesforce DX and Visual Studio Code:

- Created folder structure for force-app/main/default/objects/
- Defined metadata XML for:
    - **Event__c** (including fields like Name, Date__c, Capacity__c)
    - **Attendee__c** (fields: Name, Email__c, Phone__c)
    - **Registration__c** (fields: Attendee__c, Event__c, Ticket_Type__c, Payment_Status__c)

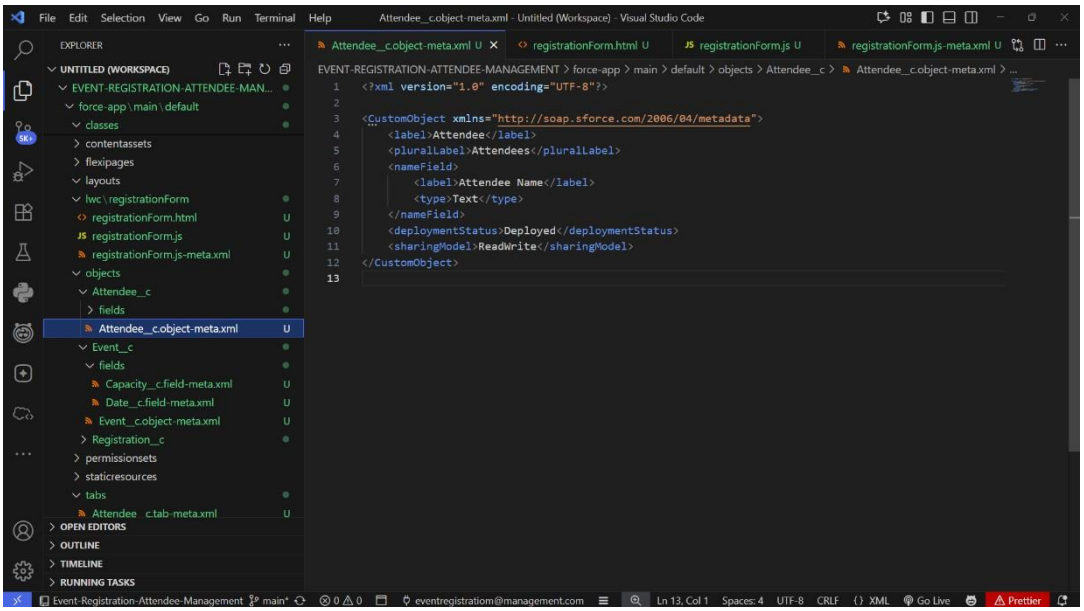
Registration:




Event:-



Attendee:-




Each individual field and object was defined in a .object-meta.xml or .field-meta.xml file, all tracked by source control for easy deployment.

# Phase 7: Integration & External Access

**Event Registration & Attendee Management — Student Project**

---

**Objective**

The objective of this phase was to enable integration of the Salesforce Event Registration system with external services to add value beyond Salesforce's native functionality. While the core application provides powerful features, linking to external systems and tools helps automate workflows and improve user efficiency.

---

### 7.1 Implemented Feature: Custom Link for External Event Lookup

To demonstrate a straightforward, user-initiated external integration, we implemented a **custom button** on the Event record page to facilitate quick event-related searches on an external platform.

- **Component:** Custom Button — Event Web Search
- **Purpose:**
  Provides event managers and users with one-click access to external web resources (e.g., Google or another event information site) about the event, helping validate or enrich event details easily.
- **Implementation Details:**

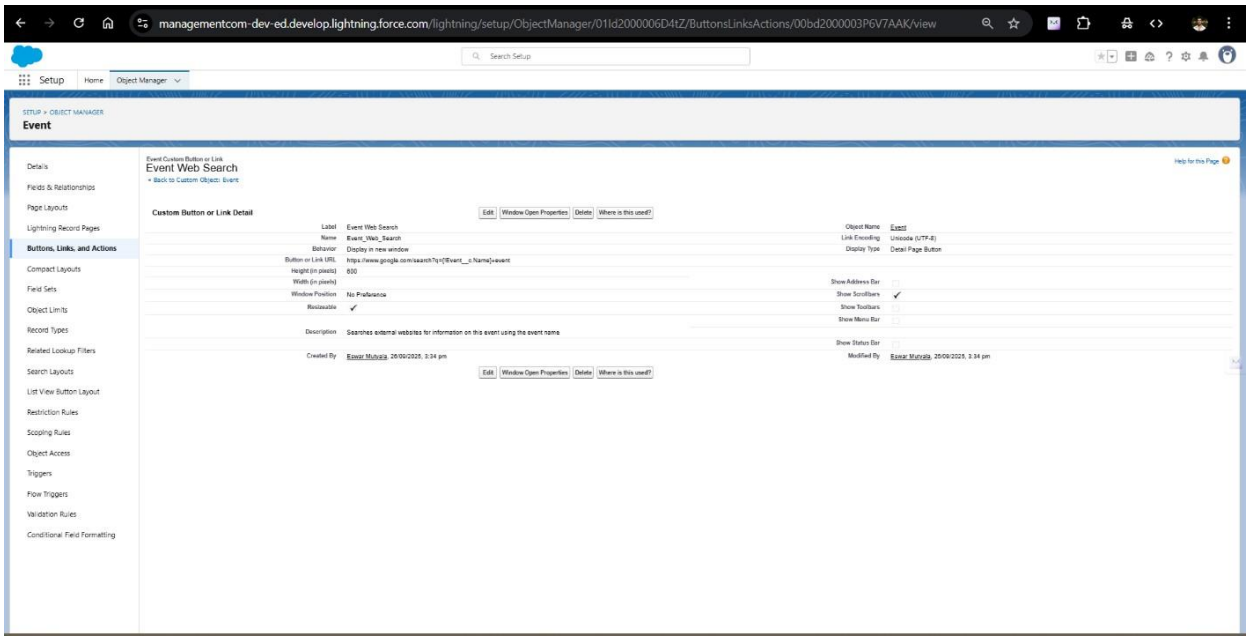
- **Object:** Event__c
- **Display Type:** Detail Page Button
- **Behavior:** Opens the external link in a new browser tab/window
- **Content Source:** Defined as a URL with dynamic merge field
- **URL Formula:**

text

https://www.google.com/search?q={!Event__c.Name}+event

- This URL leverages the merge field {!Event__c.Name} to dynamically include the current event's name into the search query, making the button context-aware.

Users can click this button from any Event record page to instantly search related event information externally, streamlining the workflow and reducing manual lookup time.



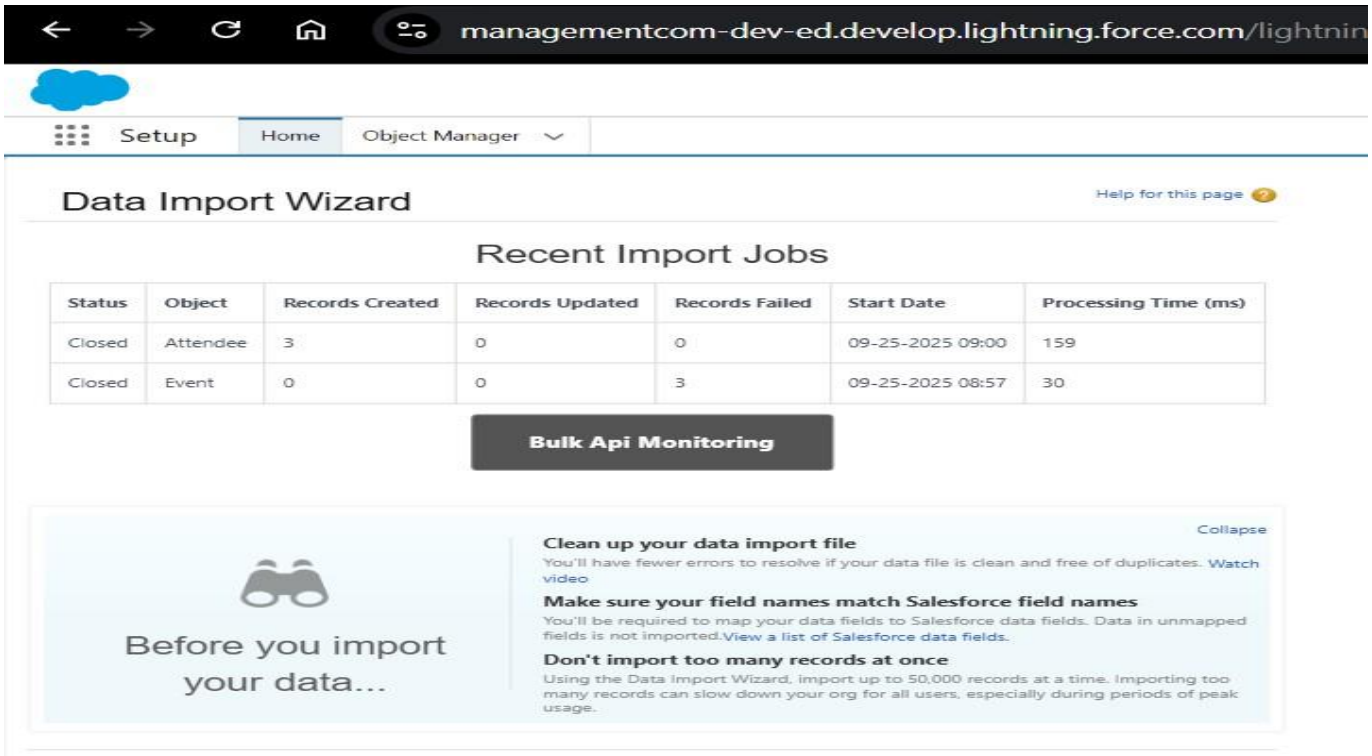

### 7.2 Future Enhancements

To evolve into a full enterprise-grade integration platform, future phases can include:

- Seamless payment gateway API integration (e.g., Stripe, PayPal) for live event ticketing.
- Real-time availability and pricing updates via external ERP or ticketing systems.
- Automated background checks or attendee validation services integrated via Apex callouts.
- Bidirectional data synchronization with marketing automation platforms.
- Enhanced event analytics via external BI tools using Salesforce Connect or REST APIs.

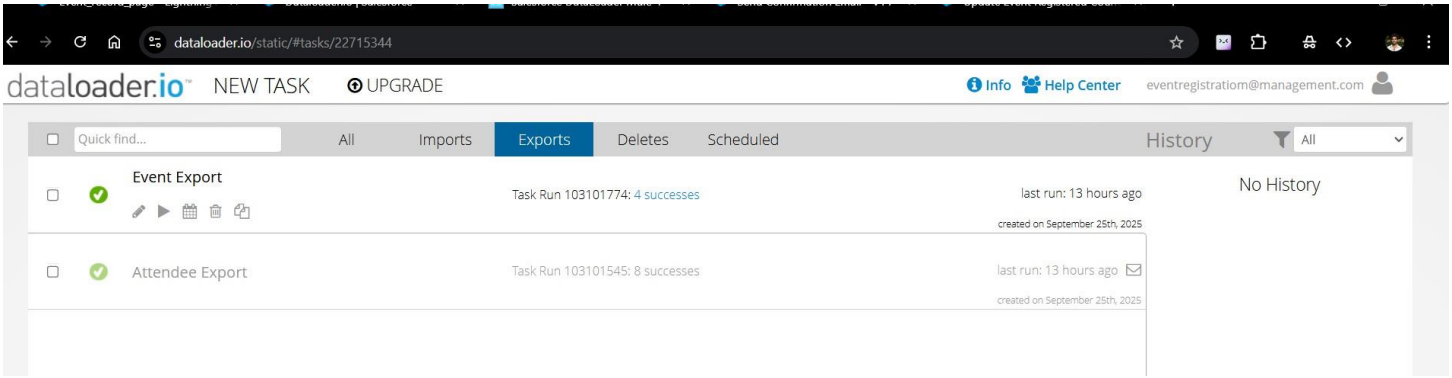
# Phase 8 – Data Management & Deployment

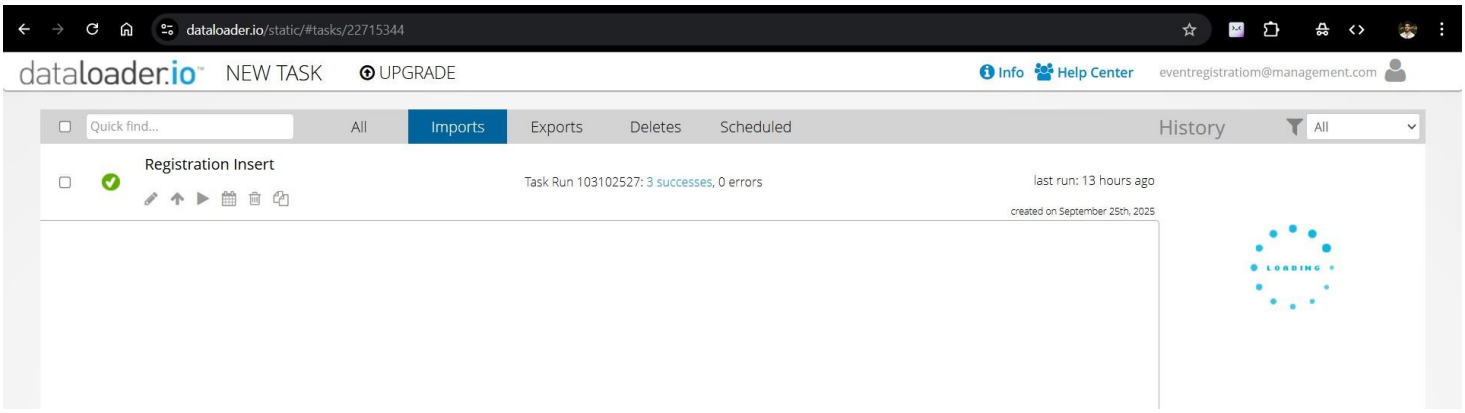
1. **Data Import Wizard (Attendees & Events)**
    - Prepared **CSV files** with sample data for testing:
        - **Events.csv** → Event Name, Date, Capacity.
        - **Attendees.csv** → Attendee Name, Email.
    - Used **Data Import Wizard** (Setup → Data → Data Import Wizard) to load **Event__c** and **Attendee__c** records into Salesforce.
    - Verified imported data by checking the respective object tabs.




2. **Data Loader (Registrations)**
    - Since **Registration__c** object had lookup relationships to **Event__c** and **Attendee__c**, used **Data Loader** for import.
    - Steps followed:
        - Exported Event__c and Attendee__c records to get their **record IDs**.
        - Mapped those IDs into **Registrations.csv** → Columns: Event__c, Attendee__c, Amount_Paid__c.
        - Imported registrations using **Insert** operation in Data Loader.

- Validated that registrations were properly linked to Events and Attendees.




3. **Duplicate Rules (Planned for Production)**
    - Identified potential need for duplicate prevention (e.g., same attendee registering multiple times for the same event).
    - Documented plan to configure **Duplicate Rules** on Attendee__c (based on Email) and Registration__c (based on Attendee + Event combination).
4. **Data Export & Backup**
    - Used **Data Export Service** (Setup → Data Export) to take a backup of all records for Events, Attendees, and Registrations after test data import.
    - Exported files were stored locally as a backup for recovery.

# Phase 9: Reporting, Dashboards & Security Review

### Objective

The objective of this phase was twofold:

1. To transform the raw data from the **Event Management application** into actionable insights through **reports and dashboards**.
2. To review and confirm the **application's security settings**, ensuring that event data (registrations, revenue, and attendee status) is protected and accessible only to authorized users.

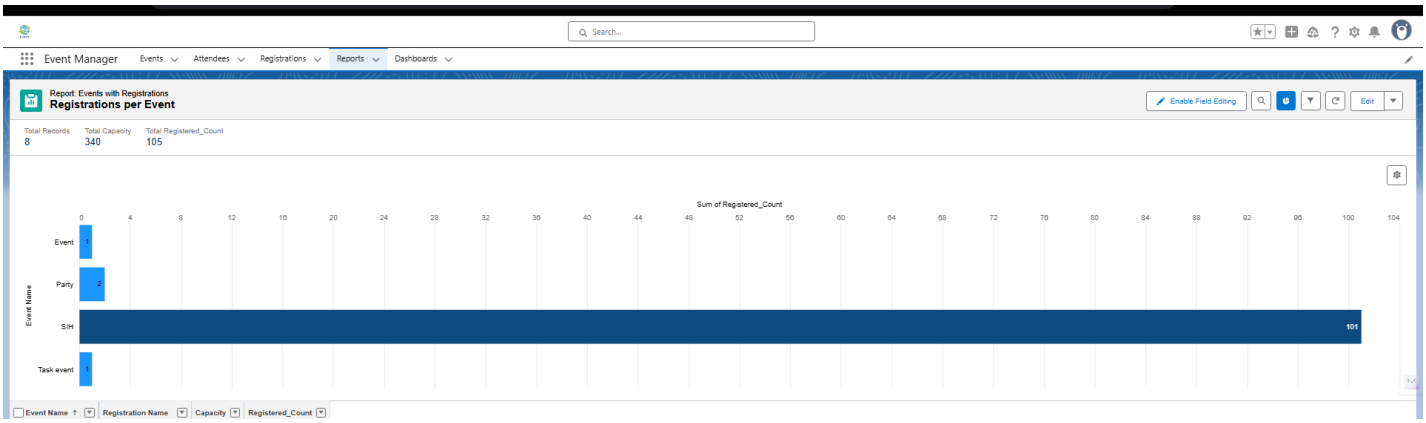
---

### 9.1 Reports

Reports in Salesforce were used to query event-related data, apply filters, group results, and provide summaries. For this project, three key reports were created to help **Event Managers and Organizers** monitor the success of events.

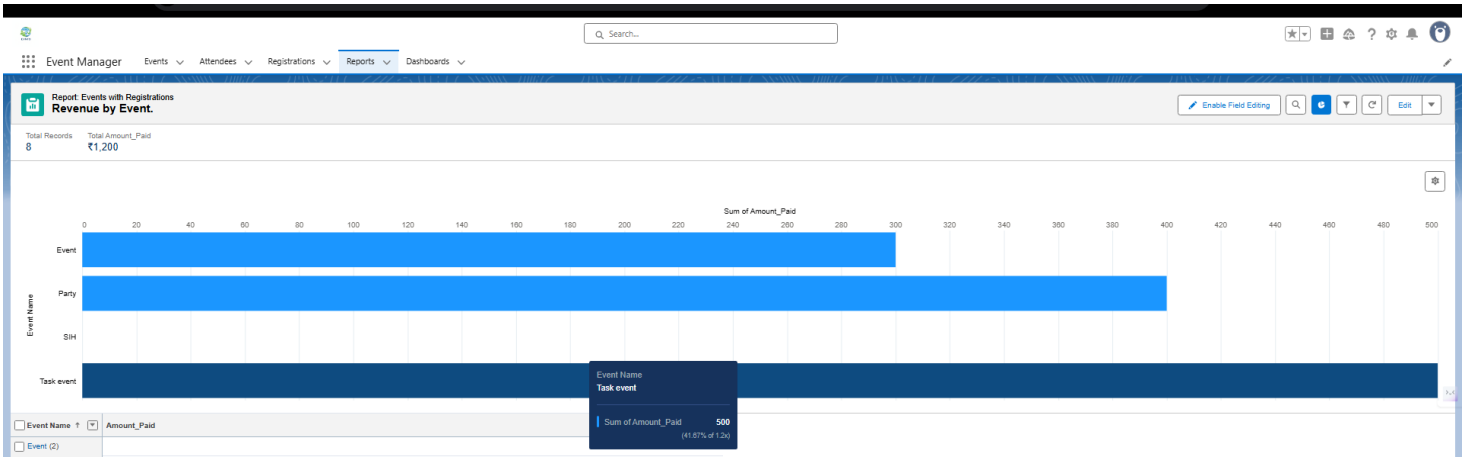
**Report 1: Registrations by Event**

- **Purpose:** To provide a quick overview of how many attendees have registered for each event.
- **Report Type:** Summary Report based on the custom report type **Events with Registrations**.
- **Configuration:** Grouped by *Event Name* and summarized by the **count of Registration records**.
- **Visualization:** A **Bar Chart** was added for an at-a-glance view of registrations across different events.



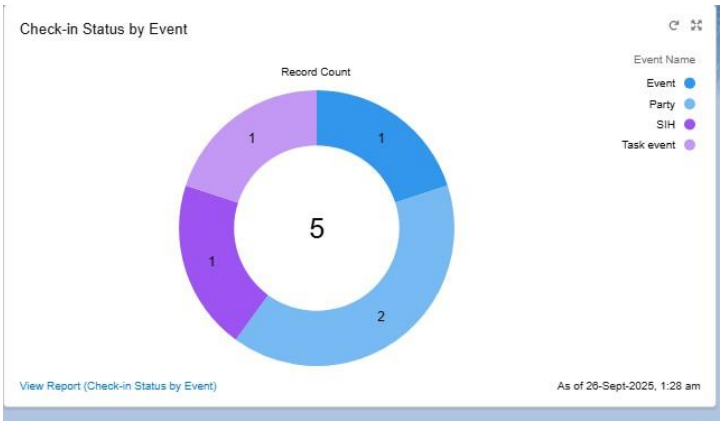
**Report 2: Revenue by Event**

- **Purpose:** To measure the total revenue generated from registrations for each event.
- **Report Type:** Summary Report using **Events with Registrations**.
- **Configuration:** Grouped by *Event Name*, with a summary field calculating the **SUM of Amount_Paid__c**.
- **Visualization:** A **Number Widget** was added to highlight total revenue per event.




**Report 3: Check-in Status by Event**

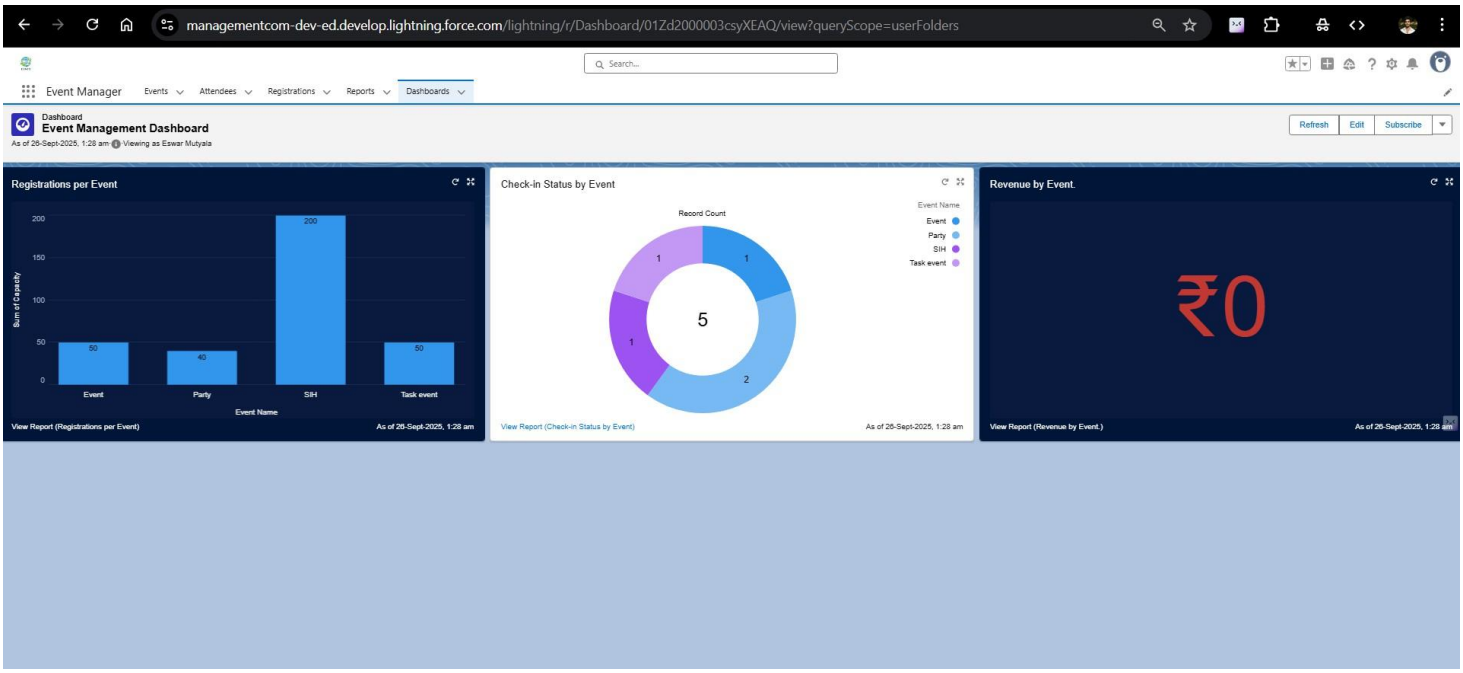
- **Purpose:** To analyze attendee participation by tracking who checked in vs. who did not.
- **Report Type:** Matrix Report using **Events with Registrations**.
- **Configuration:** Grouped by *Event Name* and then by *Check_in_Status__c*.
- **Visualization:** A **Donut Chart** was used to clearly show the proportion of attendees who checked in vs. those who did not.



### 9.2 Dashboard

**Event Management Dashboard**

- **Purpose:** To consolidate event performance metrics into a single, visual, and easy-to-read dashboard.
- **Components Added:**
    1. **Bar Chart** → Registrations by Event.
    2. **Number Widget** → Revenue by Event.
    3. **Donut Chart** → Check-in Status by Event.
- **Outcome:** Event Managers and Organizers can now instantly see **event popularity, financial performance, and attendee engagement** without manually analyzing records.




### 9.3 Security Review

Security settings were reviewed to ensure proper access control and data protection.

- **Sharing Settings:**
    - Org-Wide Defaults (OWD) were set to **Private** for custom objects (*Event, Registration, Attendee*).
    - Sharing Rules were documented to allow Organizers access only to their assigned events.
- **Field-Level Security (FLS):**
    - Sensitive fields such as *Amount_Paid__c* were restricted to Event Managers and Admins.
    - Attendees had read-only access to event details but not to financial data.

- **Session Settings & Login Policies:**
    - Default Salesforce session timeout and password policies were applied for added security.
    - No external login IP restrictions were applied in development, but they were noted as a best practice for production.
- **Audit Trail:**
    - The Salesforce Audit Trail feature was reviewed to ensure tracking of configuration changes within the org.

## Phase 10: Final Presentation & Project Assets

**Event Registration & Attendee Management — Student Project**

---

**Overview**
This phase concludes the development of the Salesforce Event Registration & Attendee Management application. The project has been fully developed, rigorously tested, and demonstrated successfully. The final deliverables include project documentation, source code repositories, and a demonstration video to showcase the solution's capabilities.

---

**Project Links & Showcase**

- **Live Demo Video:** https://drive.google.com/file/d/1ayn0NttZqcyPJzbqcOe-AyHnjtsQP7yT/view?usp=drivesdk
  A comprehensive walkthrough highlighting key features such as event creation, attendee registration via Lightning Web Components, automated capacity management, and email notifications.
- **Source Code Repository:** https://github.com/eswar2004vit/Event-Registration-Attendee-Management.git
  Full access to all Apex classes, Lightning Web Components, flows, custom objects, and metadata files used in the development of this project.

---

**Conclusion**
The Salesforce Event Registration & Attendee Management system successfully addresses the challenges of manual event tracking and attendee registrations. Through a well-architected custom data model, layered automation via declarative flows and Apex, and an interactive Lightning user interface, the solution enables efficient event management and real-time data insights.
This project provided hands-on experience in the complete Salesforce application lifecycle—from declarative setup through programmatic customization to deployment and integration—demonstrating the power and flexibility of the Salesforce platform to deliver business value.

I would like to express my heartfelt gratitude to the SmartBridge Team and the TCS Last Mile Salesforce Program for their unwavering guidance and invaluable support throughout this journey.

Sincerely,
**Eswar Gopala Swamy Mutyala**