Problem Statement: Salesforce Event Registration & Feedback App

Event management is a critical function for organizations, institutions, and communities. Whether it's workshops,
seminars, conferences, or social gatherings, each event involves multiple moving parts — from planning and promotion
to attendee registration, communication, and post-event feedback. Despite its importance, most small and mid-sized
organizations still rely on a patchwork of spreadsheets, emails, and third-party forms to handle these tasks.

This fragmented approach creates several challenges:

1. **Inefficient Registration Processes**
Attendee registration is often handled manually or through separate online forms with no direct integration into a
centralized system. This results in scattered data, duplicate entries, and inconsistent attendee information.
Organizers spend hours cleaning up and merging spreadsheets before each event.

2. **Inconsistent and Delayed Communication**
Sending confirmation emails, reminders, or updates to attendees is usually done manually. When reminders go out late —
or not at all — attendees miss important details, and engagement drops. This reduces the professionalism and reliability
of the event experience.

3. **Disconnected Feedback Collection**
Feedback forms are typically created in external tools (Google Forms, SurveyMonkey, etc.), which are disconnected from
attendee records. Analyzing this feedback requires manual data export and compilation. This not only wastes time but
also delays insights that could improve future events.

4. **Lack of Real-Time Insights and Reporting**
Because all these functions are spread across different tools, organizers cannot see real-time attendance numbers,
engagement metrics, or feedback summaries in one place. Management loses out on actionable insights, and event quality
stagnates over time.

5. **Impact on Attendee Experience**
Disorganized registration, poor communication, and slow feedback handling leave attendees with a fragmented and
unprofessional impression of the event. This can reduce repeat attendance, lower satisfaction scores, and harm the
organization's reputation.

The Need for a Unified System
Organizations need a single, integrated platform that manages all aspects of event organization — registration,
communication, and feedback — in one place. Such a system should automate repetitive tasks

(like confirmation and reminder emails), store all data in a centralized database, and provide dashboards for instant analytics.

This Project Provides the Solution

The Salesforce Event Registration & Feedback App addresses all these challenges by:

- Centralizing event, attendee, and feedback data within Salesforce.
- Automating registration confirmations, pre-event reminders, and post-event feedback requests.
- Allowing feedback to be directly linked to attendee and event records.
- Providing organizers with real-time reports and dashboards to measure attendance, engagement, and satisfaction.
- Optionally integrating AI to analyze feedback sentiment and summarize comments automatically.

This unified system reduces manual work, eliminates errors, speeds up communication, and improves the attendee
experience — all while giving organizers powerful analytics to refine future evnt.

Phase 2 (Org Setup) Work Completed

1. Salesforce Org Access & Preliminary Configuration

- Successfully logged in to the Salesforce Developer Org.
- Verified administrative access through the Setup (■■) interface.

2. Company Information, Locale, and Time Zone

- Navigated to Setup Company Settings Company Information.
- Reviewed and updated Company Name, Default Locale, Default Language, Default Time Zone, and Default Currency.
- Ensured alignment of system time, reporting, and email communications with organizational settings.

3. Email Deliverability & Org-Wide Email Address

- Configured Setup Email Deliverability to 'All Email' to enable outbound messaging.
- Created and verified an Org-Wide Email Address (noreply@yourorg.com) for standardized communications.

4. Business Hours and Holiday Configuration

- Defined Business Hours (Monday–Friday, 09:00–18:00).
- Configured sample holidays under Setup Company Settings Business Hours / Holidays.

5. My Domain Configuration

- Registered and deployed My Domain: orgfarm-d7ad62acc7-dev-ed (partitioned enhanced domains).
- Enabled secure login, Experience Cloud compatibility, and improved URL management.

6. Experience Cloud Enablement

- Activated Digital Experiences via Setup Digital Experiences Settings.
- Created an initial site framework 'Event Portal' using an available template.

7. Role Hierarchy Implementation

- Created a structured Role Hierarchy: Manager (top-level), Organizer (reports to Manager), Support/Viewer (optional).
- Established clear data visibility and sharing policies.

8. Permission Set Creation

- Developed Permission Sets to manage access control:
- Organizer Access – Full CRUD on Event, Attendee, and Feedback.
- Manager Access – Read/Report access to all objects with analytics capabilities.
- Assigned permission sets to test users accordingly.

9. Profiles and Test Users

- Utilized the Standard User profile for typical users.
- Created test users (organizer1@..., manager1@...) with Salesforce licenses and assigned appropriate permission sets.

10. Organization-Wide Defaults & Sharing Settings

- Configured Setup Security Sharing Settings: Event, Attendee, and Feedback objects set to 'Private'.
- Established Sharing Rules to grant Manager/Organizer appropriate record access.

11. Public Group Creation

- Created 'Organizers Group' to simplify Sharing Rule management.

12. Guest User Configuration (Experience Site)

- Configured the Guest User Profile for the 'Event Portal' site with minimum required Read/Create permissions.

13. Lightning App Creation

- Developed a Lightning App titled 'Event Manager'.
- Added navigation tabs for Events, Attendees, Feedback, Reports, and Dashboards.

14. Reporting & Dashboard Structure

- Created folders 'Event Reports' and 'Event Dashboards'.
- Applied folder-level sharing with the Manager role.

15. Login Access Policies

- Temporarily enabled 'Administrators can log in as any user' for testing.

16. Sample Data Import

- Used Setup Data Data Import Wizard to import sample Event and Attendee data for testing.

17. Validation Testing

- Organizer created an Event.
- Manager viewed and reported on Events.
- Guest User registered as Attendee and submitted Feedback.
- Verified Permission Sets and Sharing Rules functioning as intended.

18. Dev Hub (Optional)

- Dev Hub not enabled yet; planned for SFDX integration at a later phase.

Summary of Deliverables (Phase 2)

- Comprehensive configuration of the Salesforce Developer Org aligned with project requirements.
- Creation of a secure, role-based, and permission-controlled environment.
- Initial Experience Cloud site setup for future attendee interaction.
- Development of the 'Event Manager' Lightning App with reporting and dashboards.
- Successful validation of configurations using test users and sample data.

SETUP

Business Hours

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

Business Hours Edit

Save Cancel

Step 1. Business Hours Name

* = Required Information

Business Hours Name: nagement and Review Time

Use these business hours as the default: ☑

Active: ☑

Step 2. Time Zone

Time Zone: (GMT+05:30) India Standard Time (Asia/Kolkata)

Step 3. Business Hours

Sunday	12:00 AM	to	12:00 AM	☑ 24 hours
Monday	9:00 AM	to	10:00 PM	☐ 24 hours
Tuesday	9:00 AM	to	10:00 PM	☐ 24 hours
Wednesday	9:00 AM	to	10:00 PM	☐ 24 hours
Thursday	9:00 AM	to	10:00 PM	☐ 24 hours
Friday	9:00 AM	to	10:00 PM	☐ 24 hours
Saturday	9:00 AM	to	10:00 PM	☐ 24 hours

Save Cancel

company

- Company Settings
  - Business Hours
  - Calendar Settings
    - Public Calendars and Resources
  - Company Information
  - Data Protection and Privacy
  - Fiscal Year
  - Holidays
  - Language Settings
  - My Domain

Didn't find what you're looking for? Try using Global Search.

SETUP
Company Information

Company Information
Gyan ganga Institute of Technology and Sciences

Help for this Page

The organization's profile is below.

User Licenses [10+] | Permission Set Licenses [10+] | Feature Licenses [11] | Usage-based Entitlements [10+]

Organization Detail

Edit

Organization Name	Gyan ganga Institute of Technology and Sciences	Phone	
Primary Contact	OrgFarm EPIC	Fax	
Division		Default Locale	English (United States)
Address	United States	Default Language	English
Fiscal Year Starts In	January	Default Time Zone	(GMT-07:00) Pacific Daylight Time (America/Los_Angeles)
Activate Multiple Currencies	☐	Currency Locale	English (United States) - USD
Enable Data Translation	☐	Used Data Space	342 KB (7%) [View]
Newsletter	✓	Used File Space	17 KB (0%) [View]
Admin Newsletter	✓	API Requests, Last 24 Hours	0 (15,000 max)
Hide Notices About System Maintenance	☐	Streaming API Events, Last 24 Hours	0 (10,000 max)
Hide Notices About System Downtime	☐	Restricted Logins, Current Month	0 (0 max)
Locale Formats	ICU	Salesforce.com Organization ID	00DgK00000Bqbjj
		Organization Edition	Developer Edition
		Instance	CAN96
Created By	OrgFarm EPIC, 9/16/2025, 8:30 PM	Modified By	OrgFarm EPIC, 9/20/2025, 11:26 PM

Edit

Phase -3
Salesforce Custom Objects, Fields, Layouts, and Testing

---

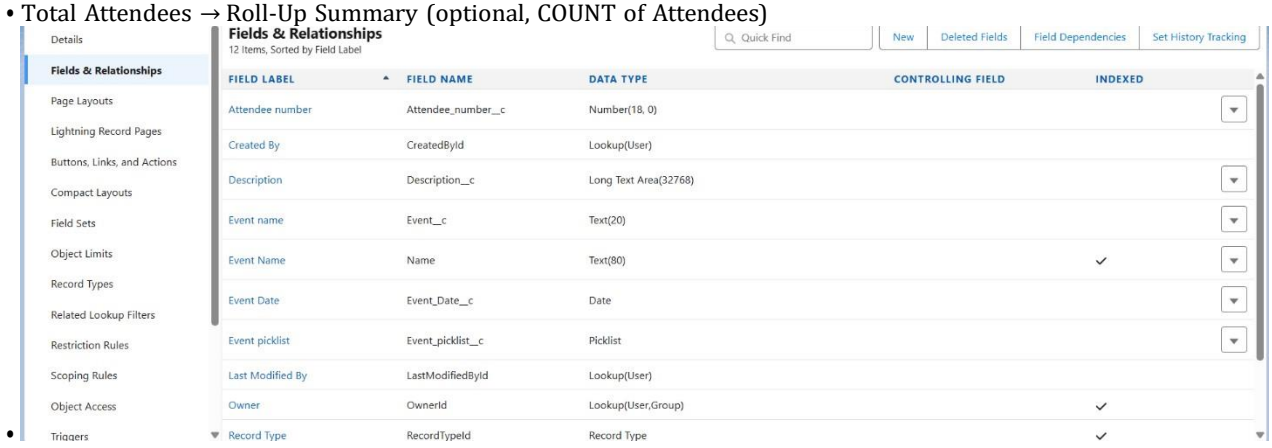
1. Custom Objects Creation

- Go to Setup → Object Manager → Create → Custom Object.
- Create Event object:
- Label: Event
- Plural Label: Events
- Record Name: Event Name (Text)
- Optional: enable Reports, Activities, Track Field History, etc.
- Repeat for Attendee and Feedback objects:
- Attendee: Label Attendee, Plural Attendees, Record Name Attendee Name (Text)
- Feedback: Label Feedback, Plural Feedbacks, Record Name Feedback Name (Auto Number or Text)

2. Add Fields & Relationships

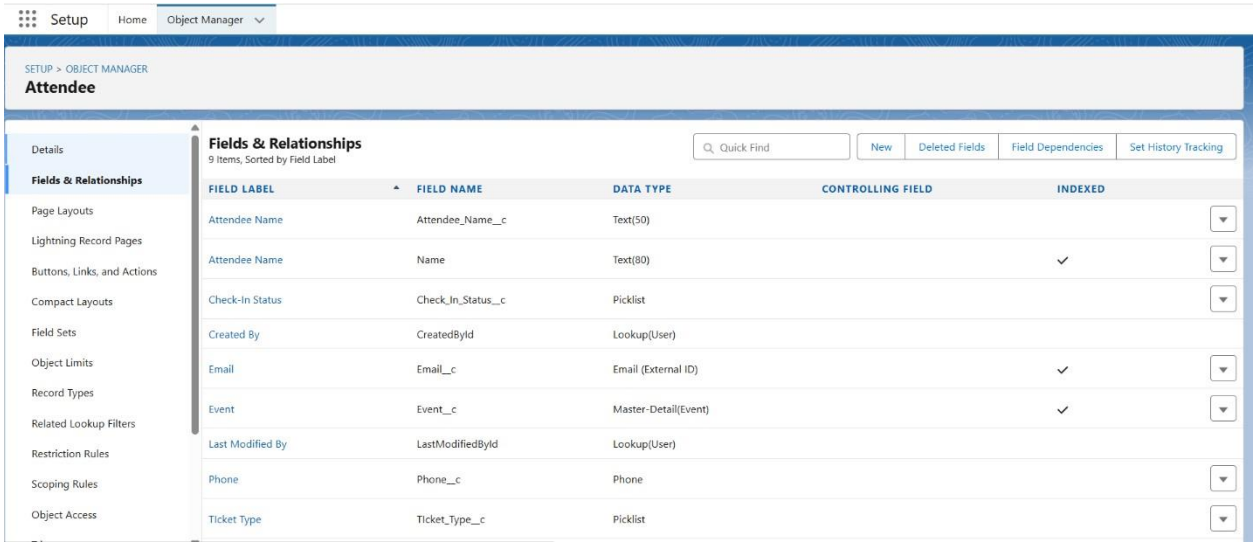
Event Object Fields

- Event Name → Text (already created as Record Name)
- Event Date → Date
- Venue → Text
- Description → Long Text Area
- Total Attendees → Roll-Up Summary (optional, COUNT of Attendees)



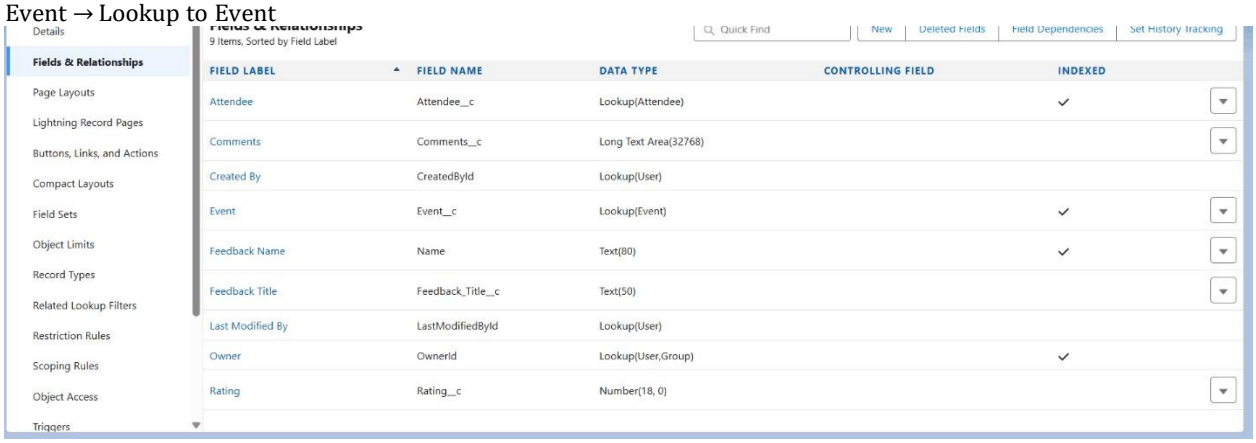
Attendee Object Fields

- Attendee Name → Text (Record Name)
- Email → Email
- Phone → Phone
- Event → Master-Detail Relationship to Event



Feedback Object Fields

- Feedback Name → Auto Number (Record Name)
- Rating → Number or Picklist
- Comments → Long Text Area
- Attendee → Lookup to Attendee
- Event → Lookup to Event



3. Page Layouts

Event Layout

- Fields: Event Name, Event Date, Venue, Description, Total Attendees
- Related Lists: Attendees, Feedback

Attendee Layout

- Fields: Attendee Name, Email, Phone, Event
- Related List: Feedback

Feedback Layout

- Fields: Feedback Name, Rating, Comments, Event, Attendee

4. Compact Layouts

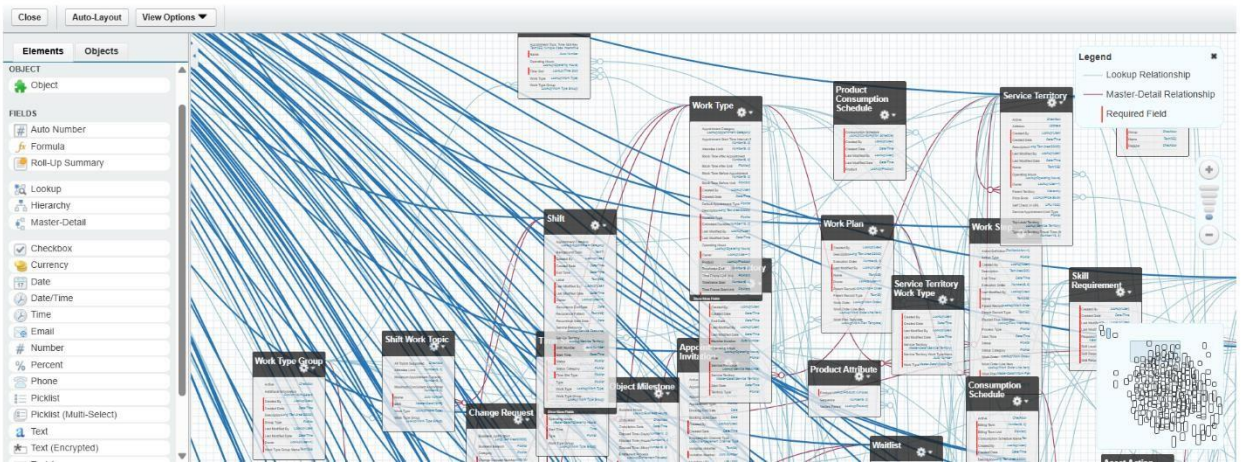
- Event: Event Name, Event Date, Venue
- Attendee: Attendee Name, Email, Event
- Feedback: Feedback Name, Rating, Attendee, Event
- Set each as Primary Compact Layout

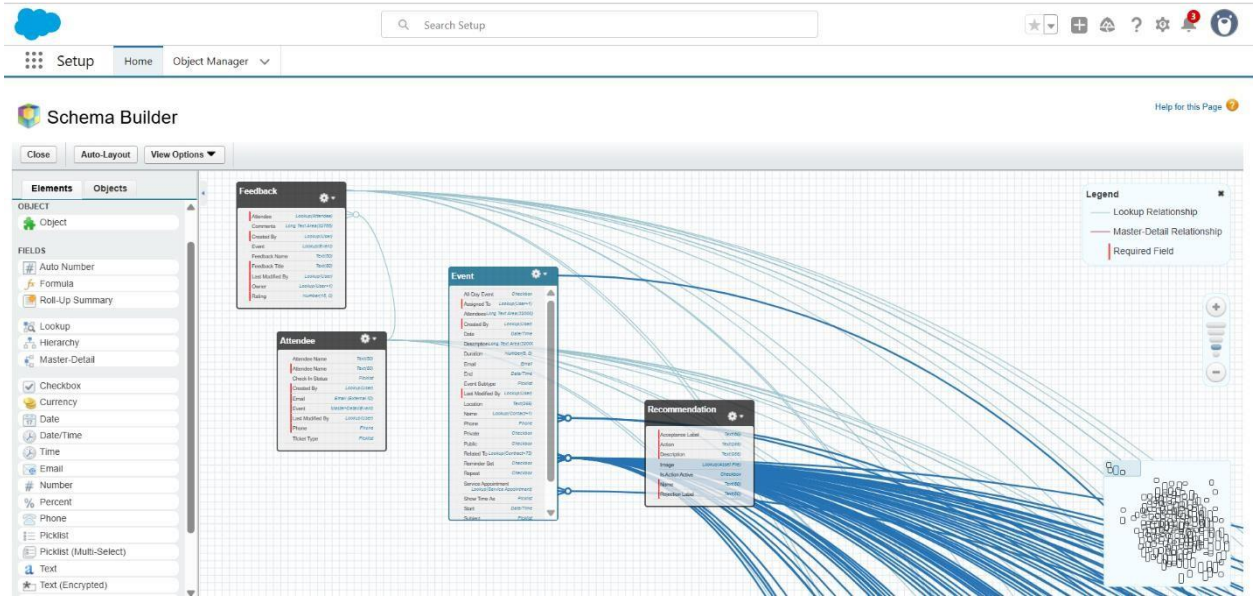
5. Record Types (Optional)

- Create Public Event Record Type for Event object
- Clone from Master layout
- Enable for required profiles
- Assign appropriate Page Layout

6. Schema Builder

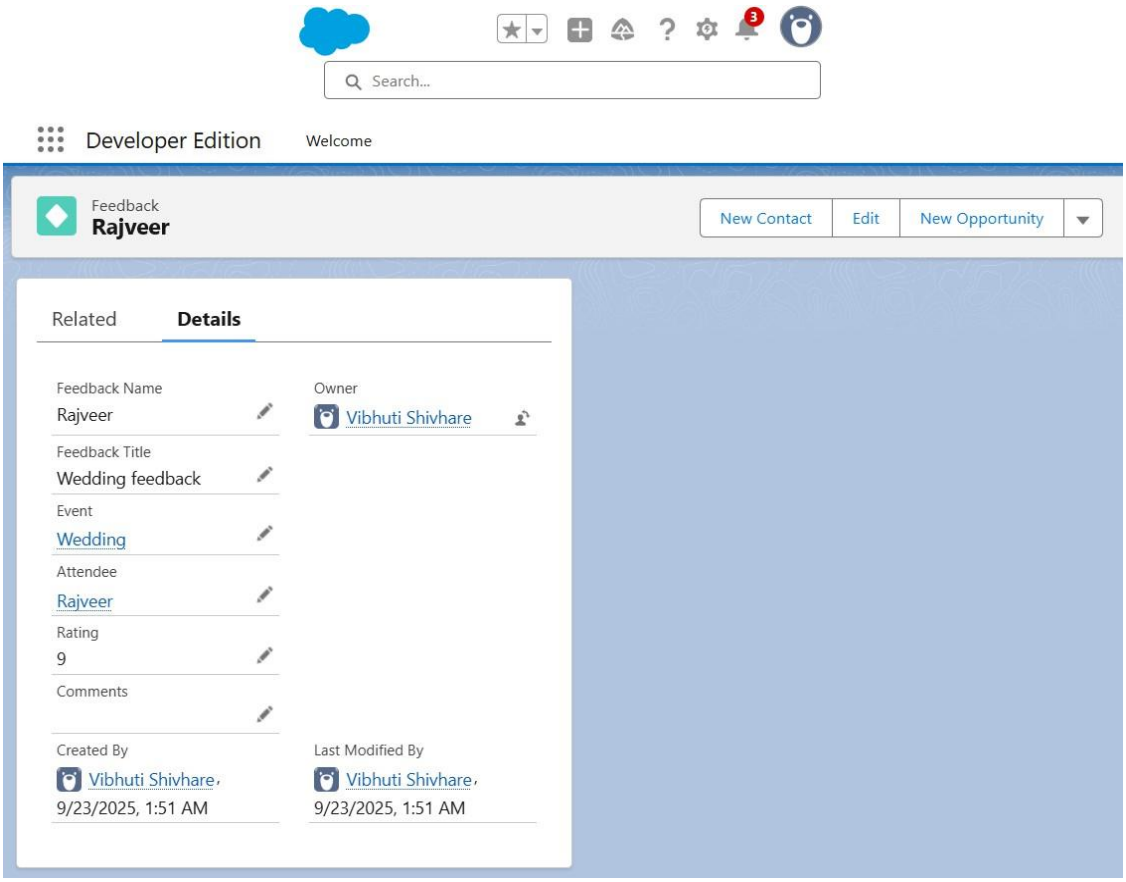
- Go to Setup → Schema Builder
- Select Event, Attendee, Feedback objects
- Check relationships:
- Attendee → Event (Master-Detail)
- Feedback → Event / Attendee (Lookup)
- Verify Roll-Up Summary field on Event → Total Attendees

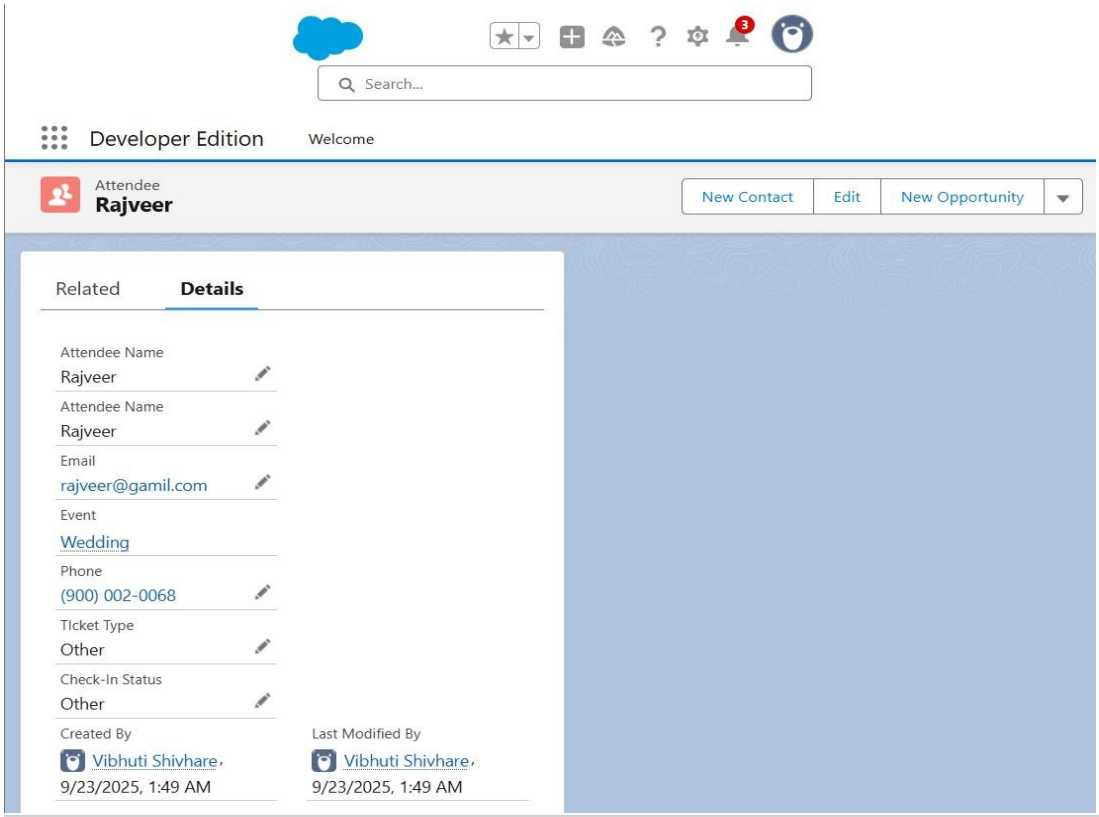


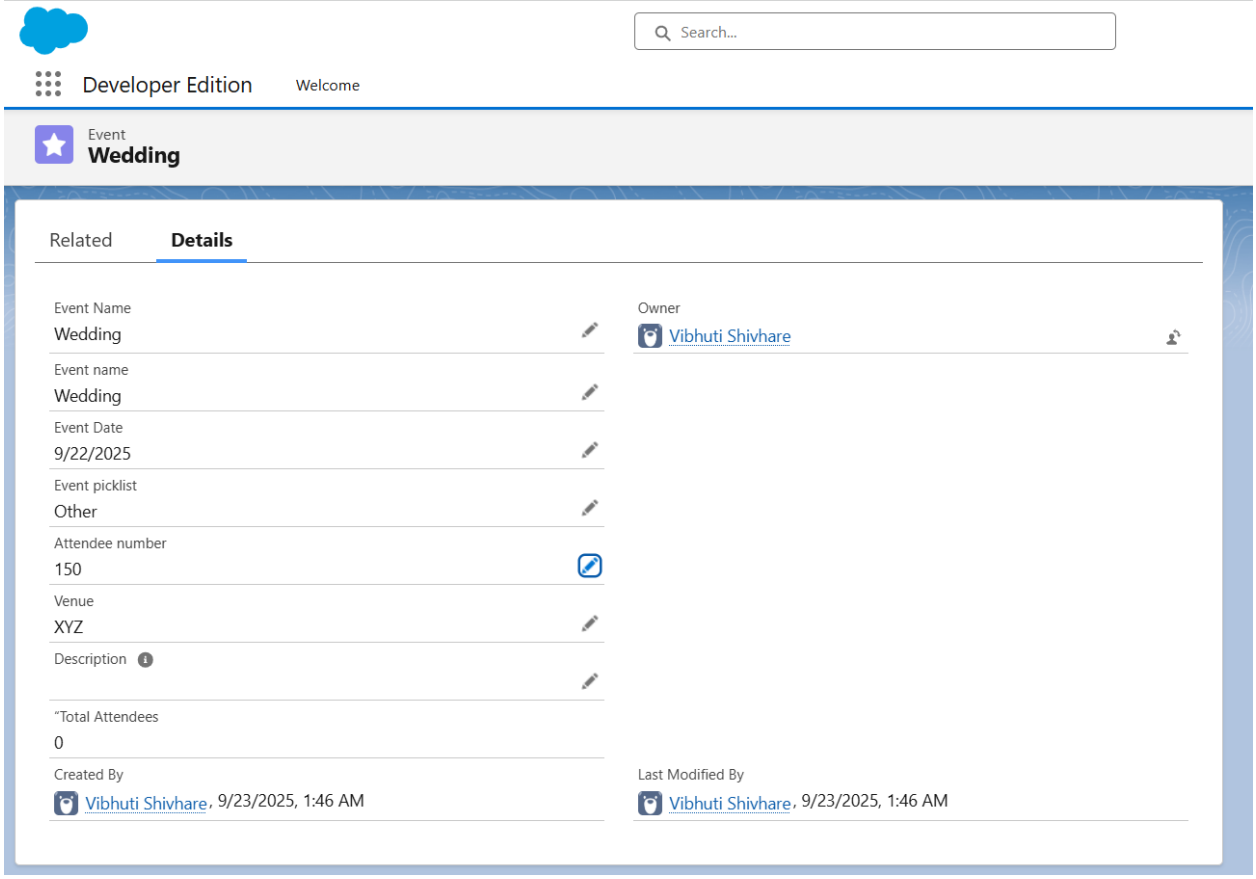


7. Testing & Verification

- App Launcher → Event tab → New Event record → Fill details → Save
- Attendee tab → New Attendee → Select Event → Save
- Feedback tab → New Feedback → Select Attendee + Event → Save
- Verify Page Layouts and Related Lists




Search...
Developer Edition
Welcome
Attendee
Rajveer
New Contact
Edit
New Opportunity
Related
Details
Attendee Name
Rajveer
Attendee Name
Rajveer
Email
rajveer@gamil.com
Event
Wedding
Phone
(900) 002-0068
Ticket Type
Other
Check-In Status
Other
Created By
Vibhuti Shivhare, 9/23/2025, 1:49 AM
Last Modified By
Vibhuti Shivhare, 9/23/2025, 1:49 AM


Search...
Developer Edition
Welcome
Event
Wedding
Related
Details
Event Name
Wedding
Owner
Vibhuti Shivhare
Event name
Wedding
Event Date
9/22/2025
Event picklist
Other
Attendee number
150
Venue
XYZ
Description
"Total Attendees
0
Created By
Vibhuti Shivhare, 9/23/2025, 1:46 AM
Last Modified By
Vibhuti Shivhare, 9/23/2025, 1:46 AM

Phase 4: Process Automation (Admin)

This phase covers automating business processes in Salesforce using admin tools like Validation Rules, Workflow Rules, Process Builder, Flow Builder, and more.

---

1. Validation Rules

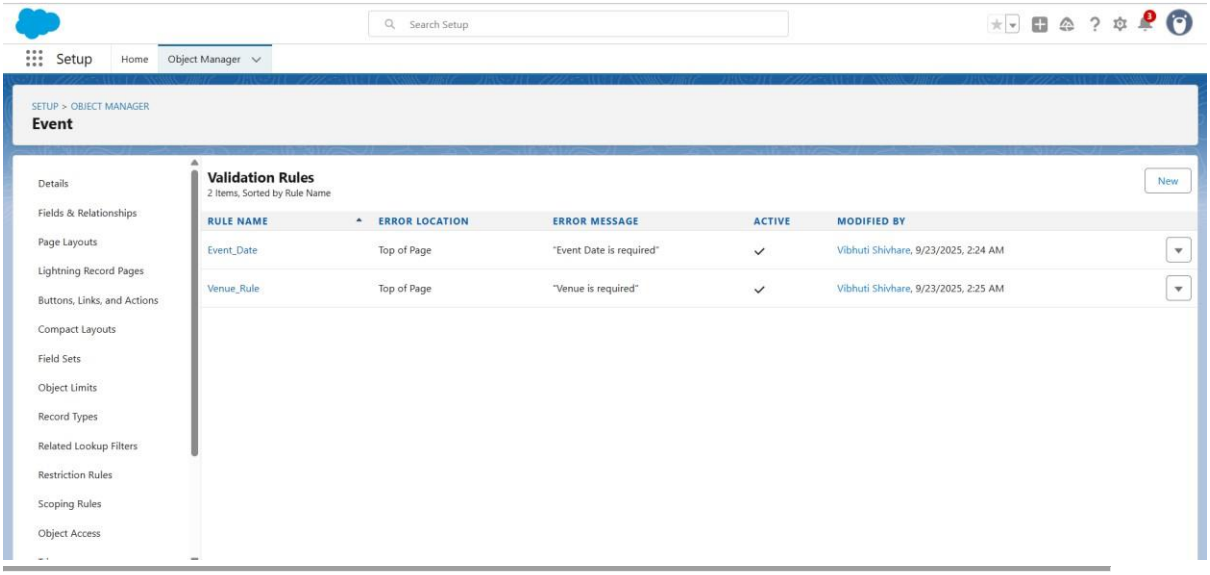
Purpose: Ensure data integrity by enforcing specific criteria before a record is saved.

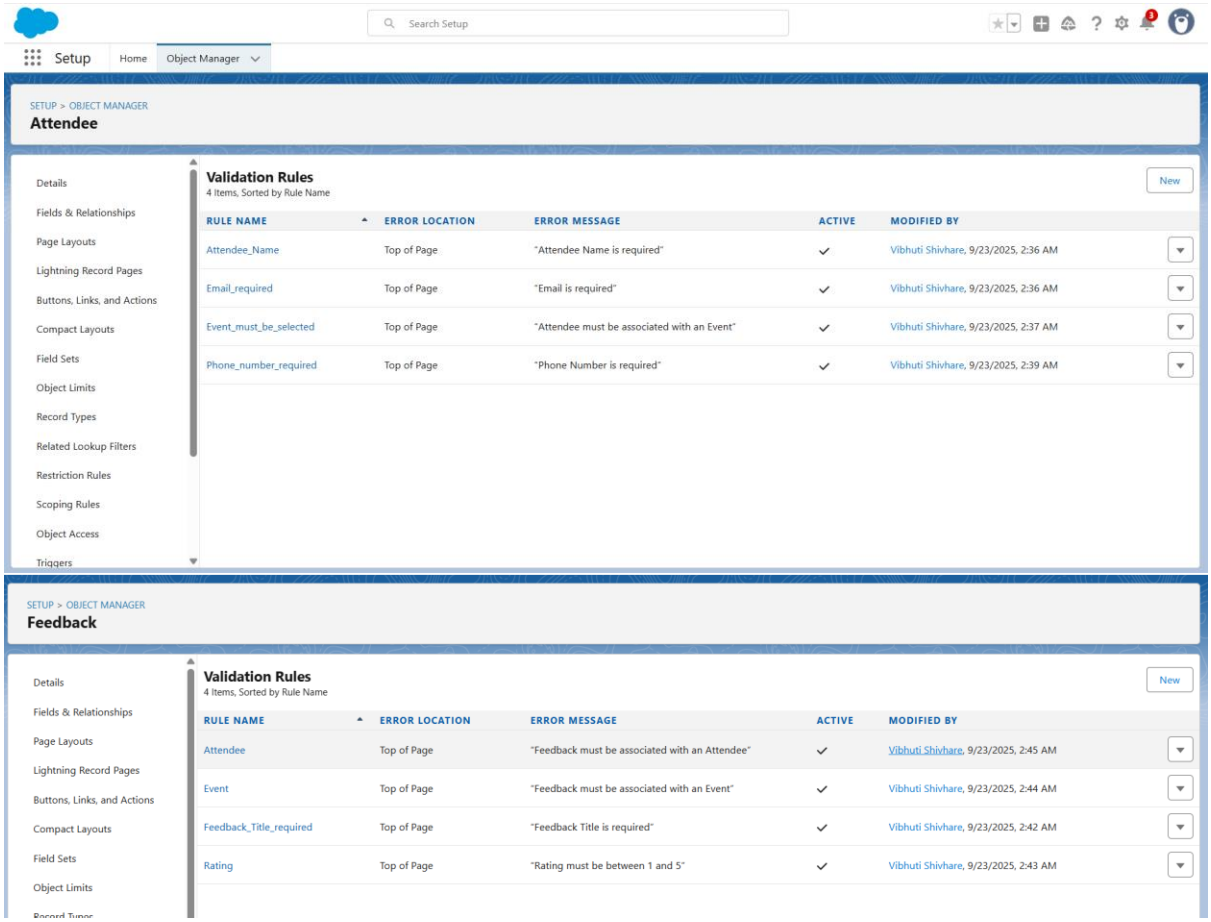
Key Points:

- Use formulas to define conditions.
- Display error messages when validation fails.
- Example: Ensure End_Date__c is after Start_Date__c.

Steps to Create:

1. Go to **Setup → Object Manager → [Object] → Validation Rules**.
2. Click **New**.
3. Enter **Rule Name** and **Description**.
4. Define the **Error Condition Formula**.
5. Enter **Error Message** and location (field-level or top of page).
6. Save and activate.





2.Workflow Rules

Purpose: Automate standard internal processes like sending emails, updating fields, or creating tasks based on record criteria.

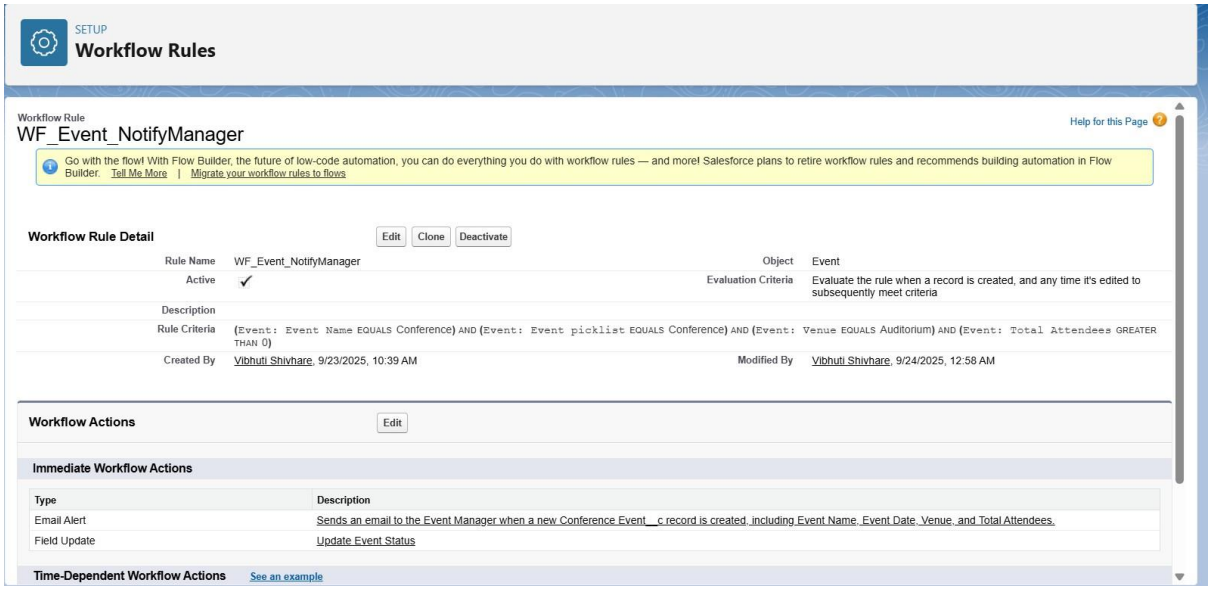
Components:

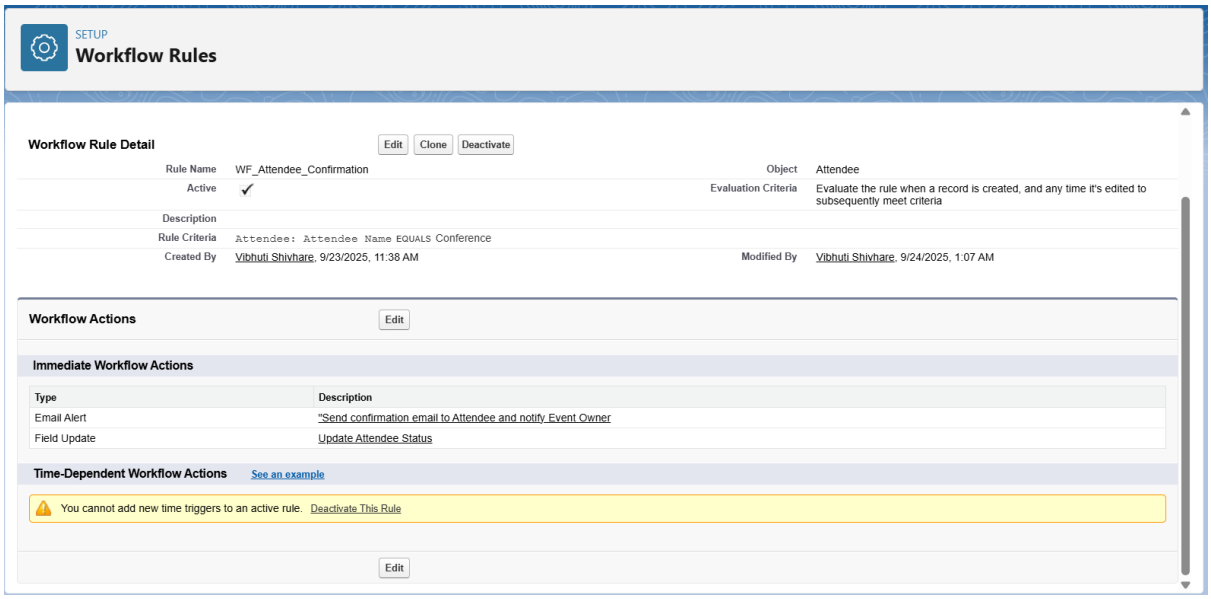
- **Rule Criteria:** When the workflow triggers.
- **Workflow Actions:** Email Alerts, Field Updates, Tasks, Outbound Messages.

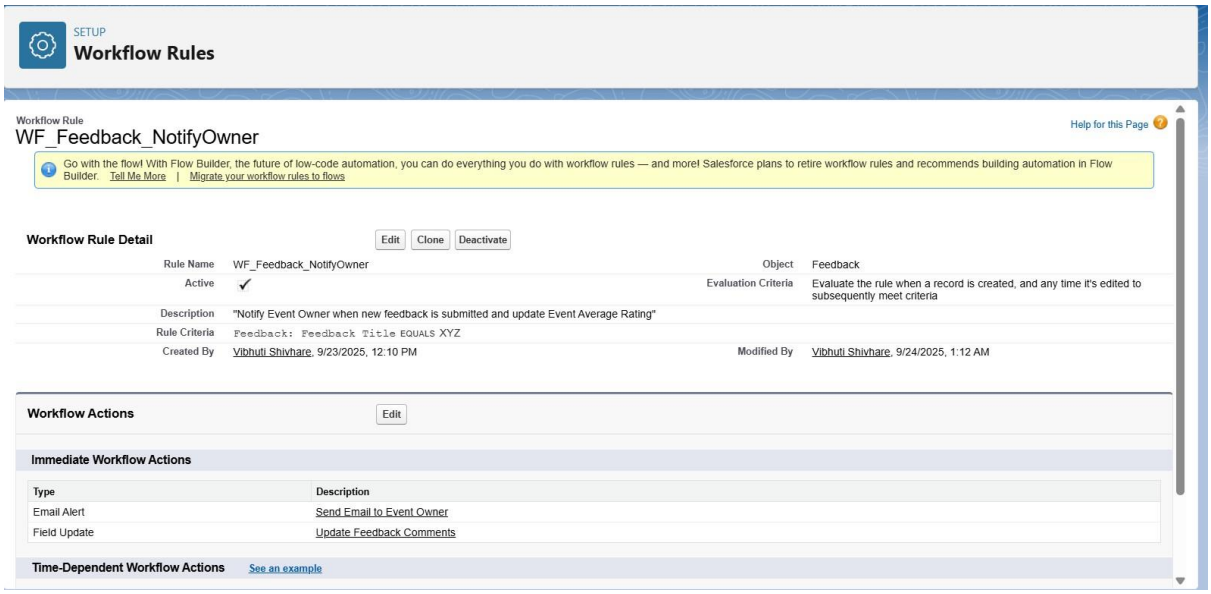
Steps to Create:

1. Go to **Setup → Workflow Rules**.
2. Click **New Rule** → Select Object → Next.
3. Define **Rule Criteria**.
4. Add **Workflow Actions**.

5. Save and **Activate**.







3️⃣ Process Builder

Purpose: Advanced automation that can update related records, launch flows, send emails, or call Apex.

Steps to Create:

1. Go to **Setup → Process Builder → New**.
2. Enter **Process Name** and select **The process starts when...**.
3. Add **Criteria** for triggering actions.
4. Add **Immediate or Scheduled Actions** (Email Alerts, Field Updates, etc.).
5. Save and **Activate**.

4️⃣ Approval Process

Purpose: Automates record approvals with defined steps and approvers.

Steps to Create:

1. Go to **Setup → Approval Processes → Create New Approval Process**.
2. Choose **Use Standard Setup Wizard**.
3. Define **Entry Criteria** and **Approvers**.
4. Specify **Approval Steps** and **Actions** (Email Alerts, Field Updates, Tasks).
5. Save and **Activate**.

5️⃣ Flow Builder

Purpose: Powerful automation tool to create **Screen Flows, Record-Triggered Flows, Scheduled Flows, and Auto-launched Flows**.

Types:

- **Screen Flow:** For user interaction.
- **Record-Triggered Flow:** Automates actions on create/update/delete.
- **Scheduled Flow:** Runs at specified intervals.
- **Auto-launched Flow:** Runs without user interaction, usually from Process Builder or Apex.

Steps to Create:

1. Go to **Setup → Flow → New Flow**.
2. Select **Flow Type** → Click **Create**.

3. Drag **Elements** to define logic (Screen, Get Records, Update Records, Decision).
4. Connect elements → Save → **Activate**.

6️⃣ Email Alerts

Purpose: Automatically send emails based on triggers.

Steps to Create:

1. Go to **Setup → Email Alerts → New Email Alert**.
2. Choose **Object**, **Recipients**, and **Email Template**.
3. Link to Workflow, Process Builder, or Flow.
4. Save.

7️⃣ Field Updates

Purpose: Automatically update field values based on conditions.

Steps to Create:

1. Can be used with Workflow, Process Builder, or Flow.
2. Define **Target Field** and **Update Logic** (Formula, Static Value, Related Field).
3. Save and activate.

8️⃣ Tasks

Purpose: Automatically create tasks for users to follow up on records.

Steps to Create:

1. Available via Workflow, Process Builder, or Flow.
2. Define **Task Subject, Due Date, Priority**, and **Assigned To**.
3. Save and activate.

9️⃣ Custom Notifications

Purpose: Send real-time notifications to users in Salesforce mobile, desktop, or in-app.

Steps to Create:

1. Go to **Setup → Custom Notifications → New**.
2. Define **Notification Name, Channels**, and **Target Object**.
3. Use Workflow, Process Builder, or Flow to trigger notifications
4. Save.

.

Phase 5: Apex Programming (Developer)

This phase introduces the fundamentals of **Apex programming** in Salesforce. It includes object-oriented programming concepts, triggers, SOQL/SOSL, collections, control statements, and asynchronous processing. Below is documentation with **working code examples** and explanations for key concepts.

1⃣ Classes & Objects

Purpose: Classes are blueprints for creating objects. They encapsulate variables (fields), methods (functions), and logic. Objects are instances of classes.

Example: Attendee Handler Class

```
public class AttendeeHandler {

    // Method to assign default status to new Attendees
    public static void assignDefaultStatus(List<Attendee_c> attendees) { for (Attendee_c att :
        attendees) {
            if (String.isBlank(att.Status_c)) { att.Status_c =
                'Registered';
            }
        }
    }

    // Method to link Feedback to Attendee
    public static void linkFeedback(List<Feedback_c> feedbackList) { for (Feedback_c fb :
        feedbackList) {
            if (fb.Attendee_c == null) {
                    fb.Attendee_c = [SELECT Id FROM Attendee_c LIMIT 1].Id;
            }
        }
    }
}
```

2️⃣ Apex Triggers (before/after insert/update/delete)

Purpose: Triggers allow developers to execute custom logic **before or after DML operations** (insert, update, delete, undelete) on Salesforce records.

EventFeedbackTrigger

```
trigger EventFeedbackTrigger on Feedback__c (after insert, after update, after delete, after
undelete) {
    Set<Id> eventIds = new Set<Id>();

    if(Trigger.isInsert || Trigger.isUndelete){
        for(Feedback__c f : Trigger.new) if(f.Event__c != null) eventIds.add(f.Event__c);
    }

    if(Trigger.isDelete){
        for(Feedback__c f : Trigger.old) if(f.Event__c != null) eventIds.add(f.Event__c);
    }

    if(Trigger.isUpdate){
        for(Feedback__c fNew : Trigger.new){
            Feedback__c fOld = Trigger.oldMap.get(fNew.Id); if(fNew.Event__c != null)
            eventIds.add(fNew.Event__c); if(fOld.Event__c != null)
            eventIds.add(fOld.Event__c);
        }
    }

    if(eventIds.isEmpty()) return;

    // Aggregate query for feedback counts
    Map<Id, Integer> eventToCount = new Map<Id, Integer>(); for(AggregateResult ar : [
        SELECT Event__c e, COUNT(Id) cnt FROM
        Feedback__c
        WHERE Event__c IN :eventIds GROUP
        BY Event__c
    ]) {
        eventToCount.put((Id) ar.get('e'),
Integer.valueOf(String.valueOf(ar.get('cnt'))));
    }

    // Update Event records with new counts
```

```
    List<Event_c> eventsToUpdate = new List<Event_c>();
    for(Event_c ev : [SELECT Id, Feedback_Count_c FROM Event_c WHERE Id IN
:eventIds]) {
        Integer cnt = eventToCount.containsKey(ev.Id) ? eventToCount.get(ev.Id) : 0;
        ev.Feedback_Count_c = cnt; eventsToUpdate.add(ev);
    }

    if(!eventsToUpdate.isEmpty()) update eventsToUpdate;
}
```

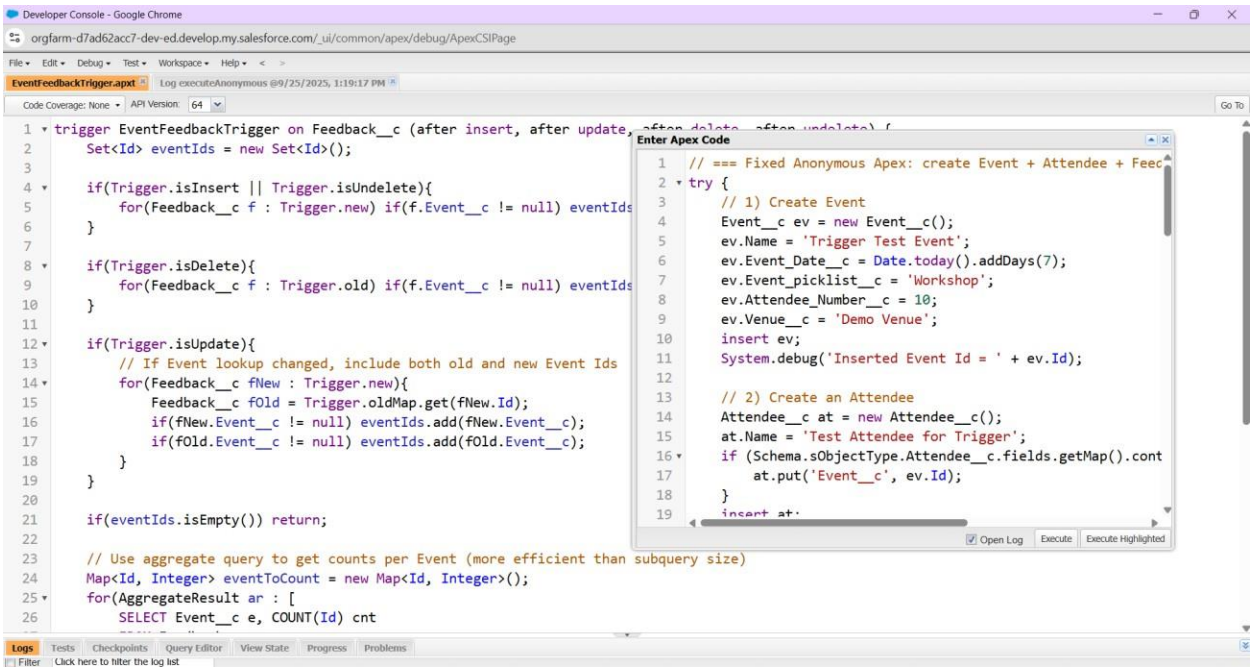
This trigger keeps the **Feedback_Count____c** field on the **Event___c** object up to date whenever feedback records are inserted, updated, deleted, or undeleted.

Anonymous Apex Test Script

```
try {
    // 1) Create Event
    Event__c ev = new Event__c(Name = 'Trigger Test Event', Event_Date__c =
Date.today().addDays(7), Event_picklist__c = 'Workshop', Attendee_Number__c = 10, Venue__c =
'Demo Venue');
    insert ev;

    // 2) Create an Attendee
    Attendee__c at = new Attendee__c(Name = 'Test Attendee for Trigger', Event__c = ev.Id);
    insert at;

    // 3) Create Feedback records
    List<Feedback__c> fbs = new List<Feedback__c>();
    fbs.add(new Feedback__c(Name = 'Feedback 1', Event__c = ev.Id, Attendee__c = at.Id,
Comments__c = 'Great event'));
    fbs.add(new Feedback__c(Name = 'Feedback 2', Event__c = ev.Id, Attendee__c = at.Id,
Comments__c = 'Loved it'));
    insert fbs;

    // 4) Query Event to check Feedback_Count__c
    ev = [SELECT Id, Feedback_Count__c FROM Event__c WHERE Id = :ev.Id]; System.debug('Feedback count
    after insert: ' + ev.Feedback_Count__c);

    // 5) Delete one feedback delete fbs[0];
    ev = [SELECT Id, Feedback_Count__c FROM Event__c WHERE Id = :ev.Id]; System.debug('Feedback count
    after delete: ' + ev.Feedback_Count__c);

    // 6) Undelete feedback undelete
    fbs[0];
    ev = [SELECT Id, Feedback_Count__c FROM Event__c WHERE Id = :ev.Id]; System.debug('Feedback count
    after undelete: ' + ev.Feedback_Count__c);
```

```
} catch (Exception ex) {
        System.debug('Exception: ' + ex.getMessage());
}
```

This is used in **Developer Console → Execute Anonymous** to test the **EventFeedbackTrigger** functionality.
It creates sample **Event, Attendee, Feedback** records and verifies that the Feedback_Count__c field on Event__c updates correctly during **insert, delete, and undelete** operations.



3️⃣ Trigger Design Pattern

- Keep triggers lean by delegating logic to **Handler Classes**.
- Ensure **one trigger per object**.
- Support bulk operations.
- Example: AttendeeTriggercalls AttendeeHandler.assignDefaultStatus().

4️⃣ SOQL & SOSL

- **SOQL**: Query records from a single object or related objects.

  List<Event__c> events = [SELECT Id, Name FROM Event__c WHERE Venue__c = 'Demo Venue'];

- **SOSL**: Search across multiple objects.

  List<List<SObject>> results = [FIND 'Workshop' IN ALL FIELDS RETURNING Event__c(Id, Name), Attendee__c(Id, Name)];

5️⃣ Collections: List, Set, Map

- **List**: Ordered collection.
- **Set**: Unique values.
- **Map**: Key-value pairs.

6️⃣ Control Statements

- Use if, for, while, switchfor logic control.
- Bulkify loops and avoid nested SOQL queries.

7️⃣ Batch Apex

- Used for processing large data sets asynchronously in batches.
- Implement Database.Batchableinterface.

8️⃣ Queueable Apex

- Asynchronous processing with the ability to chain jobs.
- More flexible than future methods.

9️⃣ Scheduled Apex

- Schedule Apex jobs to run at specific times.
- Implement Schedulableinterface.

🔟 Future Methods

- Lightweight async execution for simple background tasks.
- Must be static and return void.

1️⃣1️⃣ Exception Handling

- Use try-catch-finallyblocks.
- Catch DmlExceptionand log errors properly.

1️⃣2️⃣ Test Classes

- Ensure at least **75% code coverage** for deployment.
- Test both positive and negative scenarios.
- Use @isTestannotation.

1️⃣3️⃣ Asynchronous Processing

- Includes Batch Apex, Queueable Apex, Scheduled Apex, and Future Methods.
- Improves scalability and avoids governor limit issues.

IN MY PROJECT I MAINLY USED APPEX TRIGGER (before/after insert/update/delete) AND OTHER THING AUTOMATICALLY GOT IMPLIMENTED

Phase 6: User Interface Development

This phase covers Salesforce UI customization using Lightning App Builder, Lightning Web Components (LWC), and Apex integration. It enables building dynamic, user-friendly interfaces for Salesforce users.

🔲 Lightning App Builder

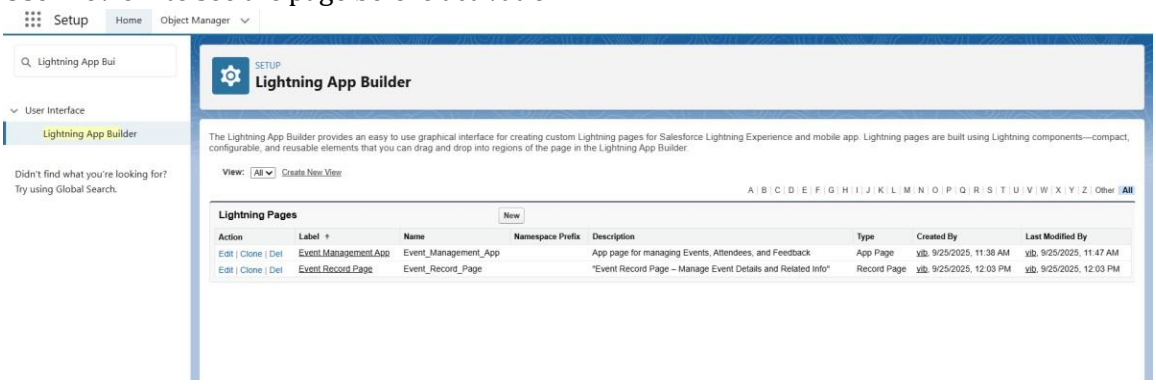
Purpose: Customize pages (Home, Record, App pages) using drag-and-drop components.

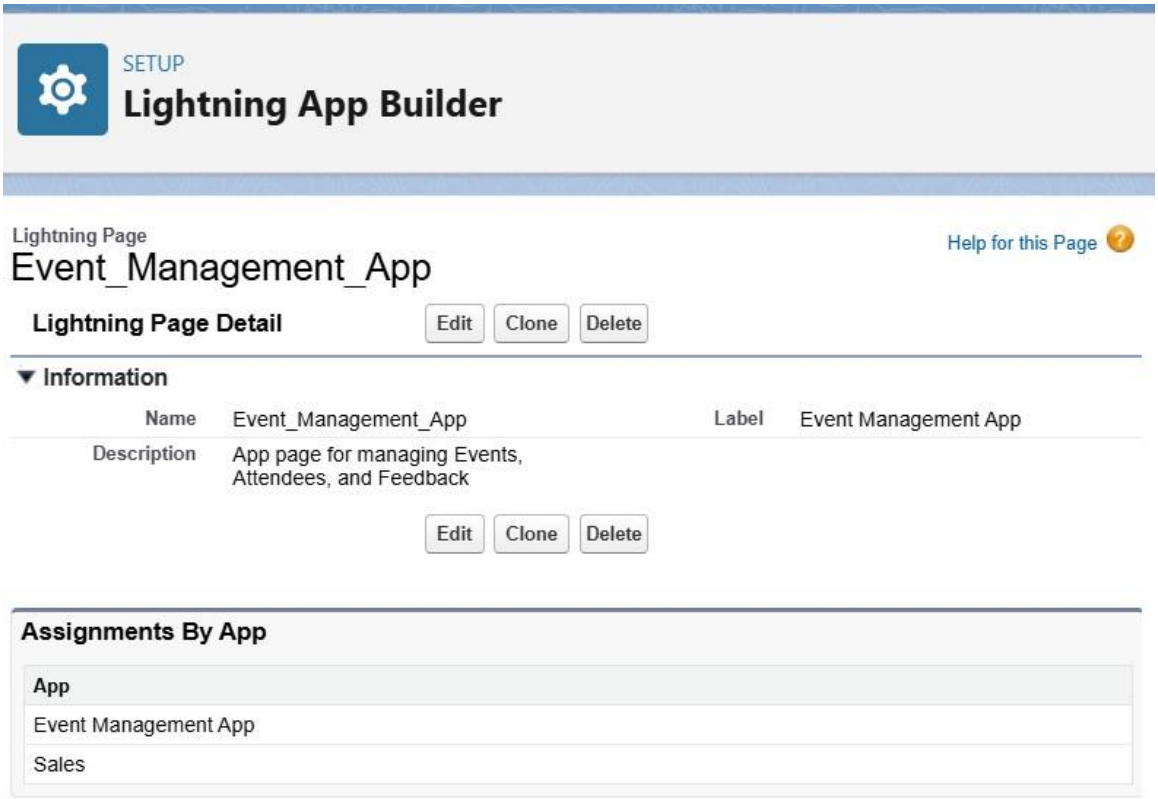
Steps:

- Go to Setup → Lightning App Builder.
- Click New → Choose Page Type (App Page, Home Page, Record Page).
- Enter Name & Description → Click Next.
- Choose Layout → Click Finish.
- Drag & drop standard or custom components.
- Click Save → Activate to make the page live.

Notes:

- Assign pages to Profiles or Apps.
- Use Preview to see the page before activation.





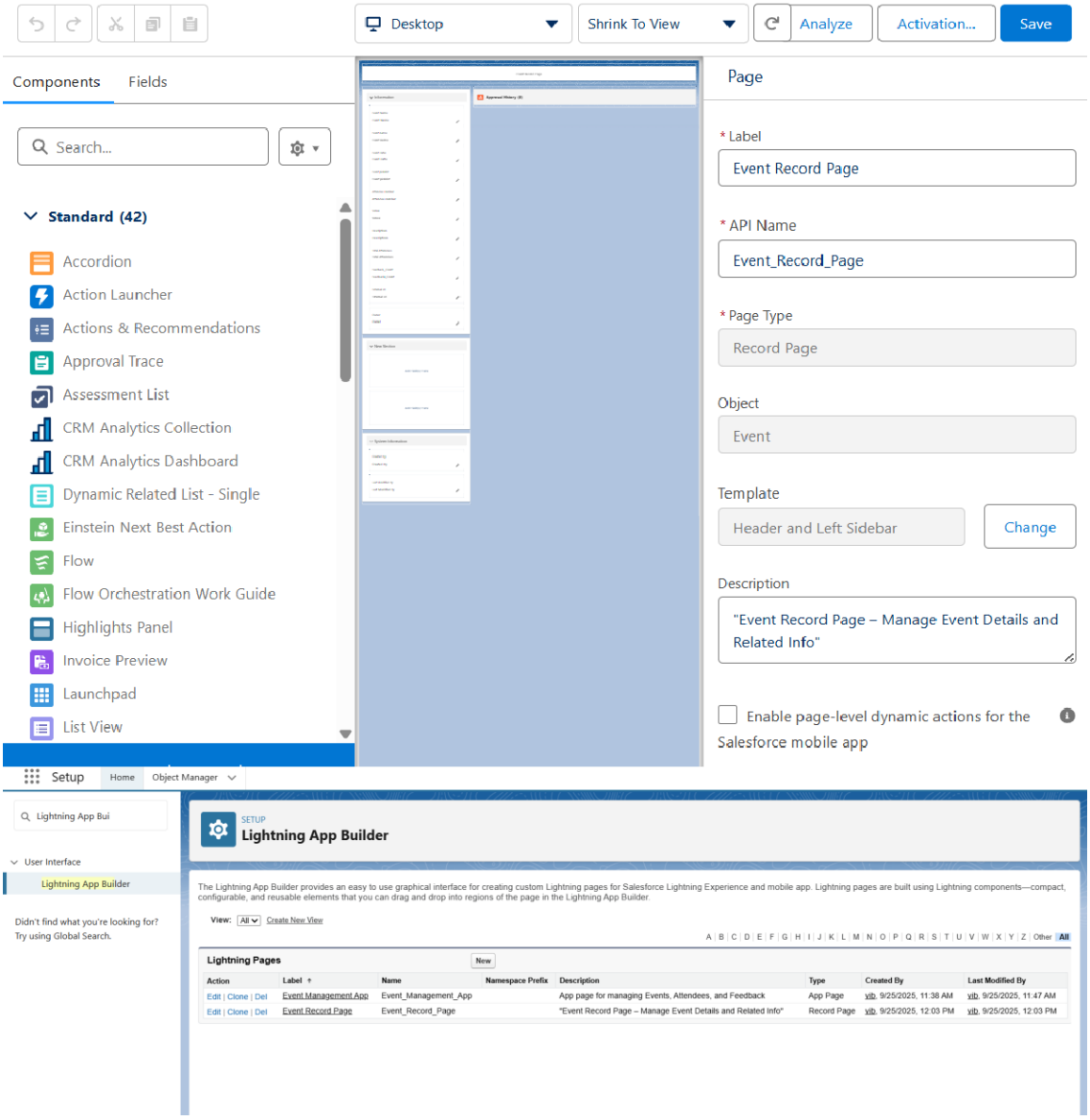
22 Record Pages

Purpose: Customize object record layouts.

Steps:

- Open Lightning App Builder → Record Page.
- Select Object → Existing Page / New Page.
- Configure Sections and add components: Related lists, Custom LWCs, Tabs.

- Click Save → Activate → Assign to App, Record Type, or Profile.



Tabs

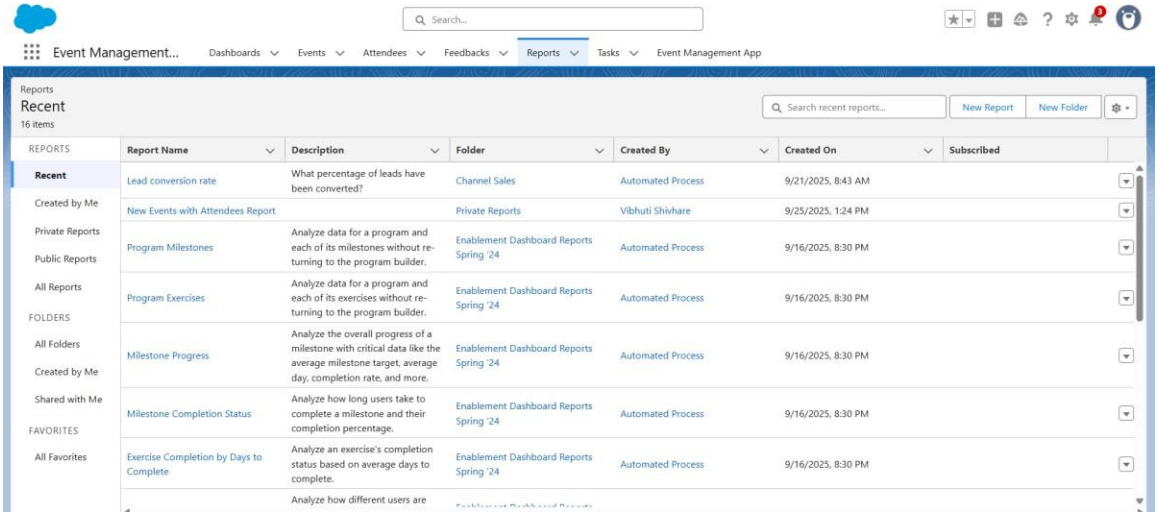
Purpose: Organize objects, components, or pages in a single app.

Steps:

- Go to App Manager → Edit App.
- Under Navigation Items, click Add Tab.
- Select Standard Object / Custom Object / Lightning Page.
- Save changes → Open app to verify tabs.

Notes:

- Control tab access using App Personalization Settings.
- Tabs can include Lightning Components, Visualforce Pages, or Reports.



4️⃣2️⃣ Home Page Layouts

Purpose: Create dashboards or personalized home pages.

Steps:

- Go to Lightning App Builder → Home Page.
- Click New → Standard Home Page.
- Drag & drop components: Reports, Dashboards, Custom LWCs.
- Save → Activate → Assign to Profiles.

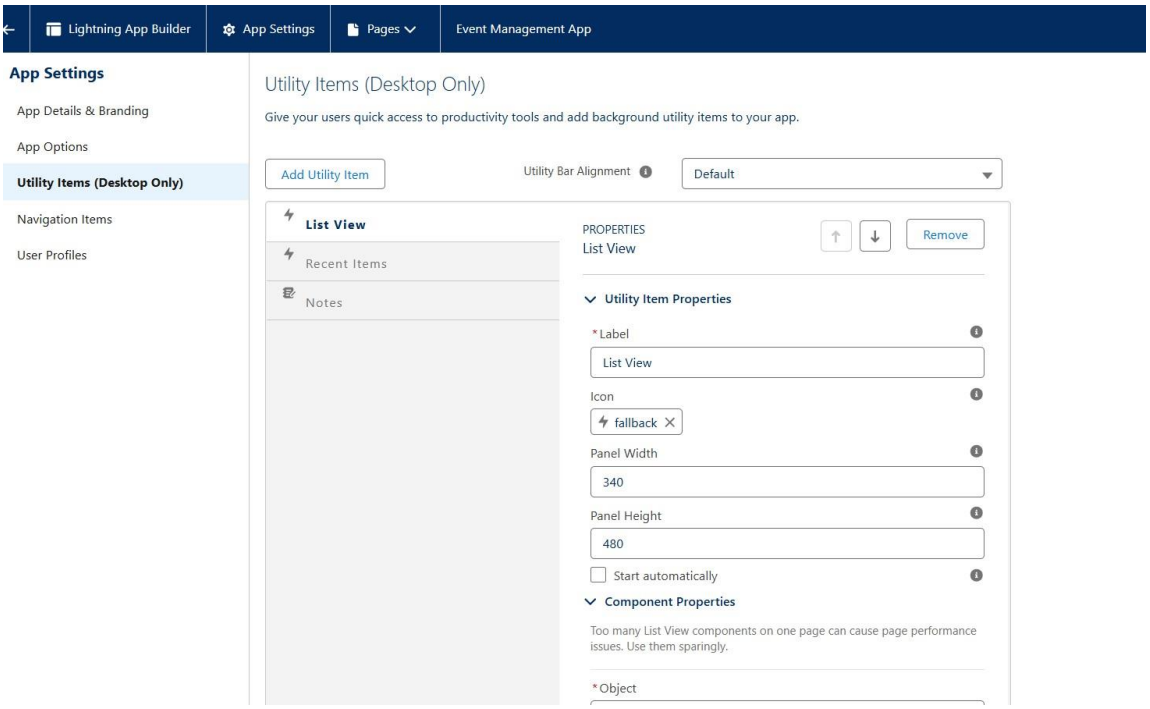
5️⃣2️⃣ Utility Bar

Purpose: Quick access to tools at the bottom of the app.

Steps:

- Open App Manager → Edit App → Utility Bar.
- Click Add Utility Item.
- Choose component (e.g., Notes, Custom LWC, Flows).
- Configure Label, Icon, Size.

- Save & verify in app.



6️⃣ Lightning Web Components (LWC)

Purpose: Build reusable, responsive UI components.

Structure:

- .js → Component logic
- .html → Template
- .js-meta.xml → Metadata

Steps:

- Open VS Code → SFDX project → Create LWC.
- Implement logic & styling.
- Deploy using SFDX: Deploy.

Notes:

- LWCs can be used in App Builder, Record Pages, Tabs, or Utility Bar.

7️⃣ Apex with LWC

Purpose: Fetch or manipulate Salesforce data using Apex.

Steps:

- Write @AuraEnabled Apex methods.
- Call methods in LWC using Wire Adapter (reactive) or Imperative Call (on user action).
- Handle responses & errors in LWC JS.

🔄 Events in LWC

Purpose: Handle component communication.

Types:

- Custom Events → Child → Parent
- Lightning Message Service → Cross-component communication
- Standard DOM events → Click, Change, Input

Steps:

- Create & dispatch CustomEvent in child component.
- Listen in parent with on<eventname> attribute.

🔌 Wire Adapters

Purpose: Retrieve Salesforce data reactively.

Examples:

- @wire(getRecord, { recordId, fields })
- @wire(getObjectInfo, { objectApiName })

Notes:

- Wire automatically updates UI when data changes.
- Mostly for read-only operations.

⚡ Imperative Apex Calls

Purpose: Call Apex methods on-demand.

Steps:

- Import Apex method in LWC JS.
- Call method inside a function (e.g., button click).
- Handle Promise: myApexMethod({ param1: value }).then(result => {/* handle result */}).catch(error => {/* handle error */});

Notes:

- Good for user-triggered actions.

- Gives explicit control over when the call happens.

✅ Best Practices

Keep LWCs modular and reusable.

Always handle errors in Apex calls.

Use Profiles / Permission Sets to control access.

Test Home Page and Record Page layouts for multiple profiles.

Deploy using Change Sets or SFDX for version control.

I mainly Concentrated on

• Lightning App Builder
• Record Pages
• Tabs
• Home Page Layouts
• Utility Bar

Phase 7: Integration & External Access

1. Named Credentials

- Simplifies authentication when making callouts to external systems.
- Stores URL, authentication type, and credentials securely in Salesforce.

2. External Services

- Allows Salesforce to integrate external APIs declaratively.
- Generates Apex actions automatically from an API schema (OpenAPI/Swagger).

3. Web Services (REST/SOAP)

- REST API enables interaction with Salesforce using standard HTTP methods (GET, POST, PUT, DELETE).
- SOAP API allows structured XML-based communication with Salesforce for integrations.

4. Callouts

- Outbound requests from Salesforce to external systems using HTTP.
- Can be synchronous (immediate response) or asynchronous (future method or queueable).

5. Platform Events

- Enables event-driven architecture within Salesforce for real-time updates.
- Subscribers receive notifications asynchronously when an event occurs.

6. Change Data Capture

- Tracks changes (create, update, delete, undelete) in Salesforce records.
- Pushes changes to external systems in near real-time via events.

7. Salesforce Connect

- Allows access to external data in real-time without storing it in Salesforce.
- Uses external objects and OData protocol to integrate external databases.

8. API Limits

- Salesforce enforces limits on API calls per 24-hour period.
- Helps prevent excessive load and ensures fair usage of resources.

9. OAuth & Authentication

- OAuth 2.0 enables secure token-based authentication for external apps.
- Supports user delegation and integration without exposing credentials.

10. Remote Site Settings

- Whitelist external URLs to allow Salesforce callouts.
- Ensures security by preventing unauthorized external requests.

This Phase is not needed in my project

Phase 8: Data Management & Deployment

This phase covers Salesforce data management and deployment strategies. The primary focus is on using the Data Import Wizard, while other tools are briefly explained for awareness.

1️⃣ Data Import Wizard

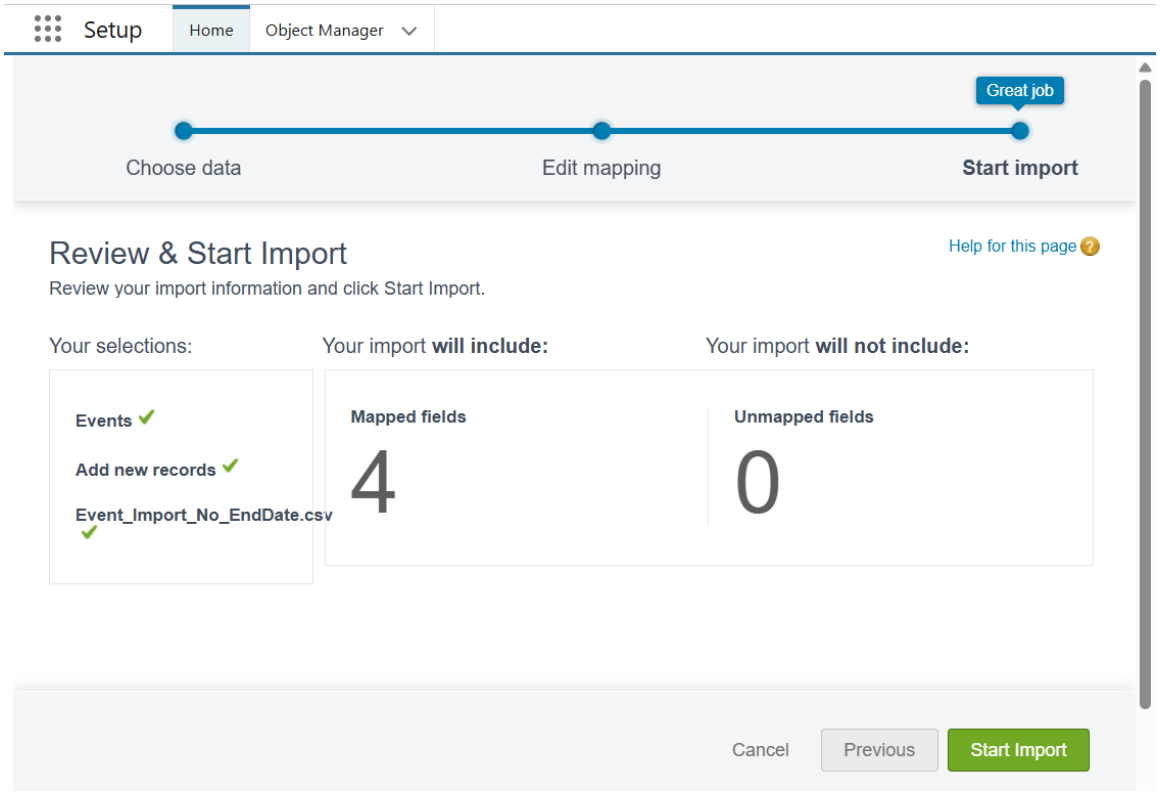
Purpose: Import data into Salesforce using a guided interface.

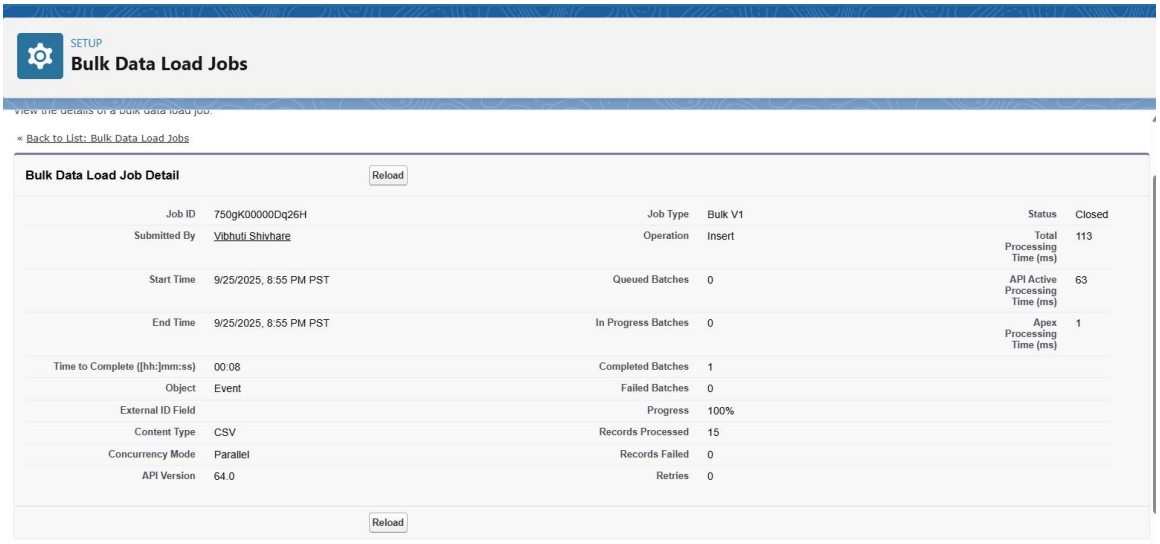
Steps:

1. Go to **Setup → Data Import Wizard**.
2. Select the object (e.g., Accounts, Contacts).
3. Upload CSV file with proper headers.
4. Map CSV fields to Salesforce fields.
5. Click **Start Import** → Monitor progress.

Notes:

- Supports standard and custom objects.
- Suitable for small to medium volume imports.

- 

- 

---

2. Data Loader

Purpose: Bulk data import/export tool for large volumes.

Quick Points:

- Download, install, and login with Salesforce credentials.

- Operations: Insert, Update, Upsert, Delete, Export.
- Supports large data volumes (millions of records).
- Requires API access and CSV mapping.

3️⃣ Duplicate Rules

Purpose: Prevent duplicate records in Salesforce.

Quick Points:

- Create rules via **Setup → Duplicate Management → Duplicate Rules**.
- Define object, matching criteria, and actions (Alert, Block).
- Use **Matching Rules** for custom logic.
- Helps maintain clean, high-quality data.

4️⃣ Data Export & Backup

Purpose: Backup Salesforce data regularly.

Quick Points:

- Go to **Setup → Data Export** and select objects.
- Schedule exports weekly or monthly.
- Download CSV ZIP files and store securely.
- Automation possible via third-party tools.

5️⃣ Change Sets

Purpose: Deploy metadata changes between orgs.

Quick Points:

- Use **Outbound Change Sets** to add components.
- Upload to target org → Deploy via **Inbound Change Sets**.
- Validate before activation.

- Works only between connected orgs (sandbox → production).

6️⃣ Unmanaged vs Managed Packages

Purpose: Package metadata for reuse or distribution.

Quick Points:

- **Unmanaged Package:** One-time deployment, code editable.
- **Managed Package:** Controlled distribution, code protected.
- Useful for deploying or sharing apps.
- Managed packages needed for AppExchange distribution.

7️⃣ ANT Migration Tool

Purpose: Command-line tool for metadata deployment/retrieval.

Quick Points:

- Requires **build.xml** and **package.xml** for metadata selection.
- Commands: retrieve, deploy.
- Monitor logs for errors/success.
- Useful in CI/CD pipelines.

8️⃣ VS Code & SFDX

Purpose: Salesforce development and deployment using VS Code and CLI.

Quick Points:

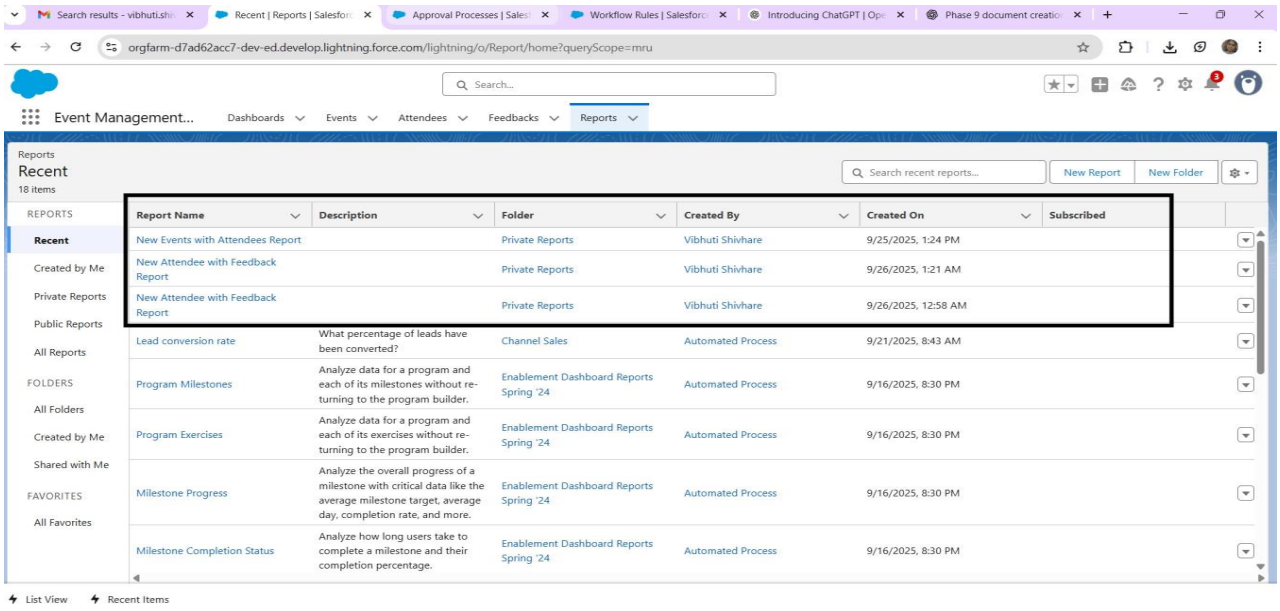
- Install **VS Code** + **Salesforce Extension Pack**.
- Authenticate orgs via SFDX CLI.
- Retrieve/deploy metadata using SFDX commands.
- Supports Lightning Web Components, Apex, and version control.

I MAINLY CONCENTRATED ON DATA IMPORT WIZARD

Phase 9 – Reporting, Dashboards

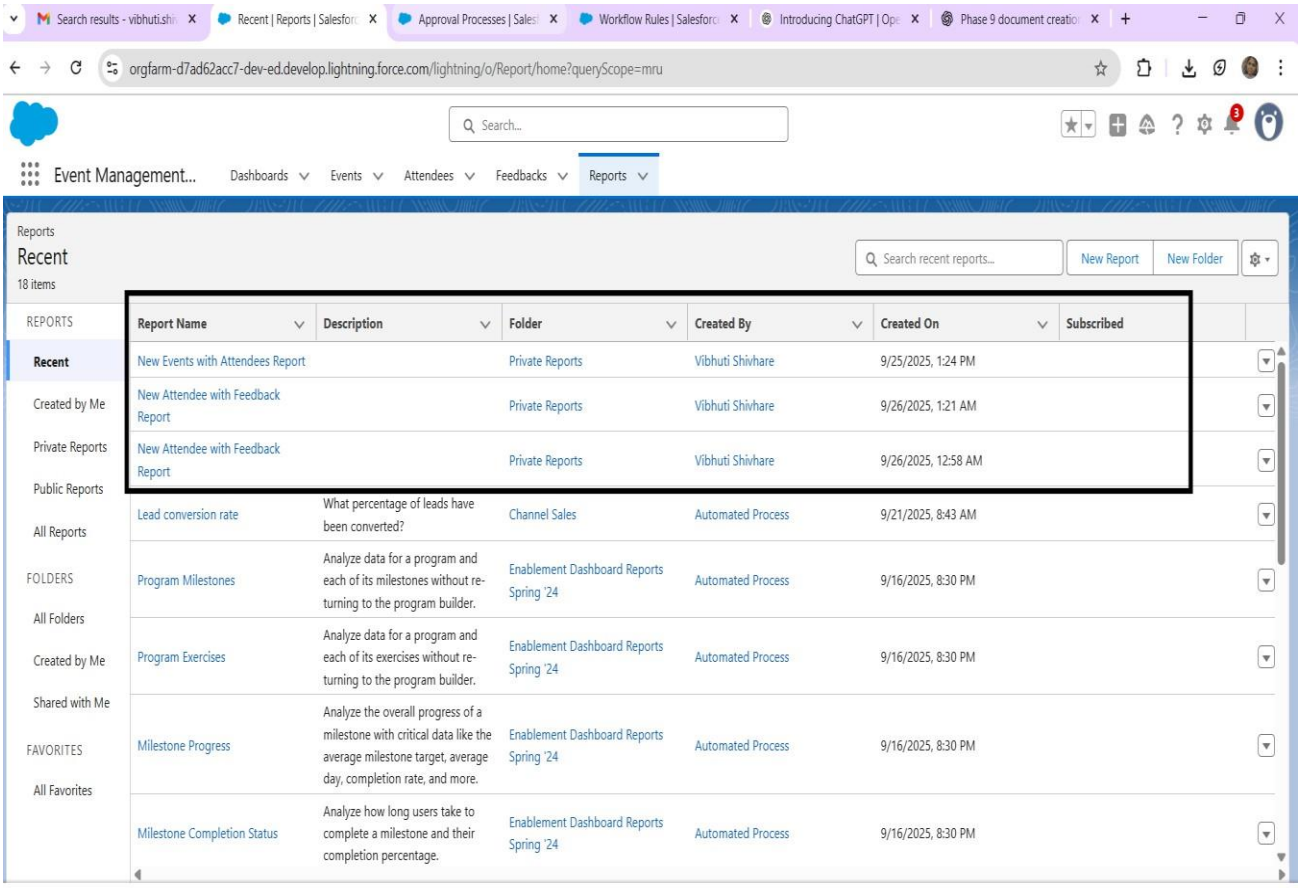
Step 1: Create a Report

- Go to App Launcher (grid icon) → Search “Reports”.
- Click “New Report”.
- Select the report type (example: Opportunities, Leads, or your custom object).
- Choose report format: Tabular, Summary, Matrix, or Joined.
- Add filters, fields, and grouping as needed.
- Save & Run the report.



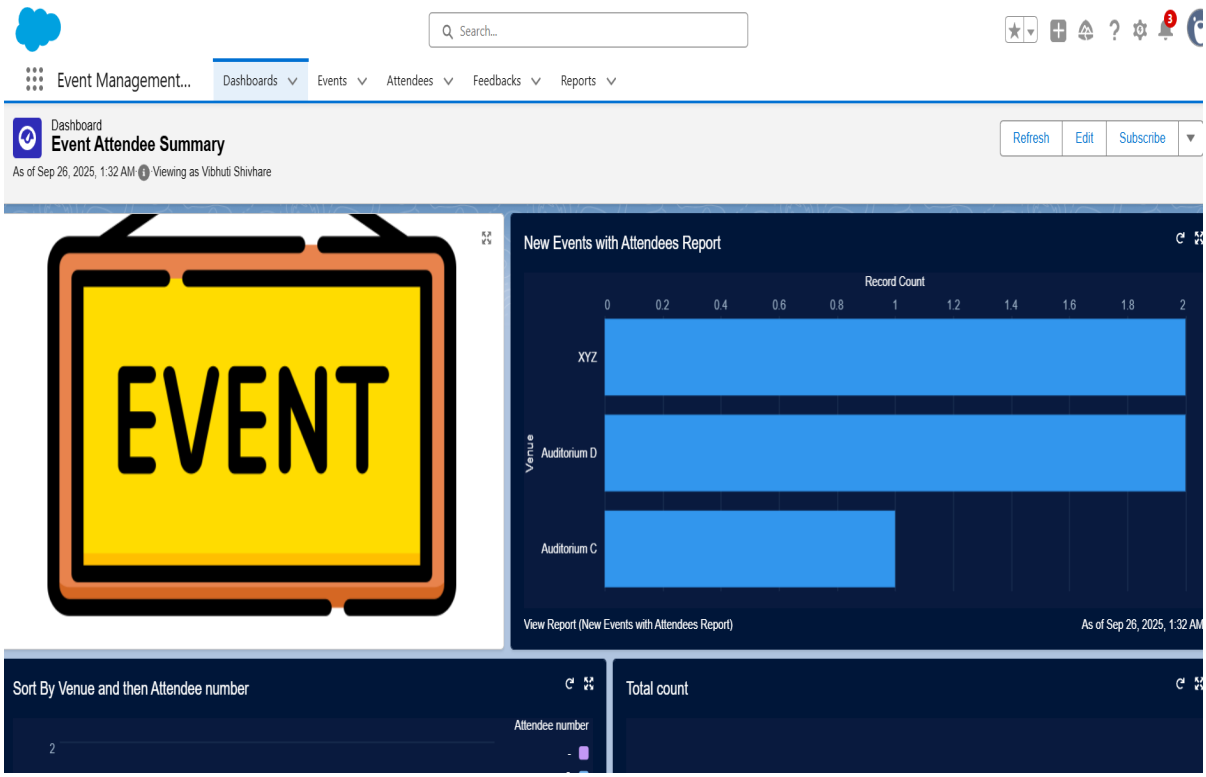
Step 2: Create a Custom Report Type (if needed)

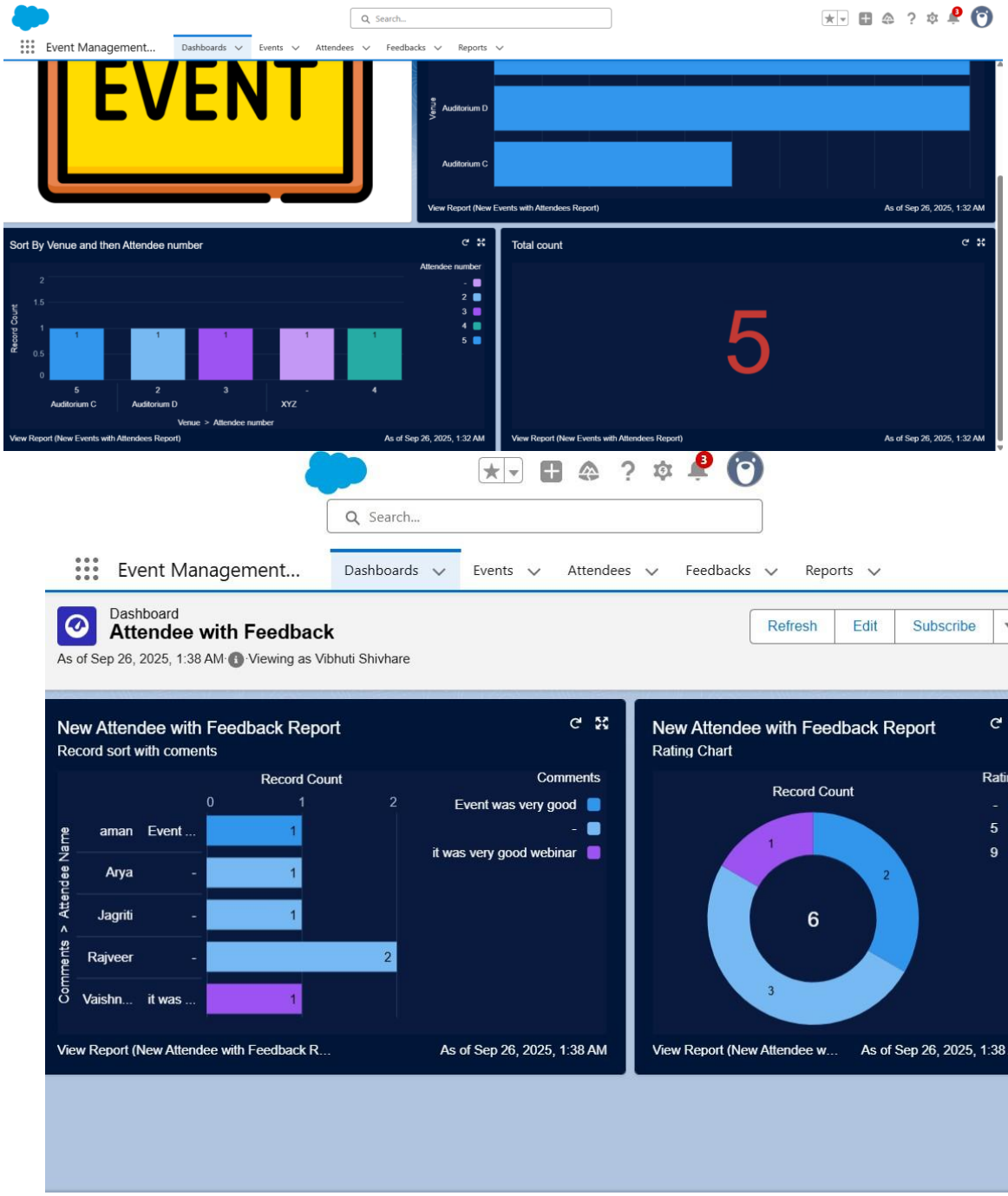
- Go to Setup → “Report Types” in Quick Find.
- Click “New Custom Report Type”.
- Select the primary object.
- Define related objects and field relationships.
- Save and deploy.



Step 3: Create a Dashboard

- Go to App Launcher → Search “Dashboards”.
- Click “New Dashboard”.
- Enter Name, Folder, Description.
- Add a new component (Chart, Table, Metric, Gauge).
- Select the source report for each component.
- Adjust layout and size.
- Save and view dashboard.


Search...
Event Management...
Dashboards
Events
Attendees
Feedbacks
Reports
Dashboard
Event Attendee Summary
As of Sep 26, 2025, 1:32 AM · Viewing as Vibhuti Shivhare
Refresh
Edit
Subscribe
EVENT
New Events with Attendees Report
Record Count
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
Venue
XYZ
Auditorium D
Auditorium C
View Report (New Events with Attendees Report)
As of Sep 26, 2025, 1:32 AM
Sort By Venue and then Attendee number
Attendee number
2
Total count



Step 4: Enable Dynamic Dashboards

- Open the dashboard you created.
- Click “Edit”.
- Under “View Dashboard As” select “Run as Logged-In User” (Dynamic) or choose a specific user (Static).
- Save.

Step 5: Configure Sharing Settings

- Go to Setup → "Sharing Settings".
- Set Organization-Wide Defaults (OWD) for each object.
- Configure Role Hierarchy (Setup → "Roles") to allow upward data access.
- Create Sharing Rules for wider access beyond OWD.
- Use Manual Sharing for individual records if needed.

Step 6: Apply Field-Level Security (FLS)

- Go to Setup → Object Manager → Select Object → Fields & Relationships.
- Click the field you want to secure.
- Click "Set Field-Level Security".
- Select which profiles/permission sets can see or edit the field.
- Save.

Step 7: Adjust Session Settings

- Go to Setup → "Session Settings".
- Set Session Timeout (e.g. 30 minutes).
- Enable/disable "Force Logout on Session Timeout".
- Save.

Step 8: Configure Login IP Ranges

- Go to Setup → "Profiles".
- Select the profile you want to secure.
- Scroll to "Login IP Ranges" → Click "New".
- Enter allowed IP ranges.
- Save.

Step 9: Review Audit Trail

- Go to Setup → "View Setup Audit Trail" in Quick Find.
- Review who made changes, what was changed, and when.
- Export if needed for compliance/security reviews.

Purpose of Phase 9

- Gain accurate insights with Reports and Dashboards.
- Provide role-based and user-based visibility with Dynamic Dashboards and Sharing Settings.

  Protect sensitive data with Field-Level Security and IP restrictions.

  Monitor configuration changes with Audit Trail.