**Salesforce Project Report: Simple Event Management & Attendee Tracking System**

**An Automated Event & Attendee CRM Implementation**

**Project Overview:**

The Simple Event Management & Attendee Tracking System is a comprehensive Customer Relationship Management (CRM) solution built on the Salesforce platform. The system's core function is to streamline the event management process for a local organization, replacing manual spreadsheets for tracking events and attendees. Key features include custom objects for managing events and attendees, a real-time, automated process for sending email confirmations and checking for maximum capacity, and a custom user interface for staff to easily create and manage events. The CRM also features a basic integration endpoint for sending emails via an Organization-Wide Email Address and provides managers with reports and dashboards to monitor event attendance and key performance metrics.

**Objectives:**

The primary objective of this project is to centralize all event and attendee data within a single, scalable Salesforce environment to drive data accuracy and streamline operations. By automating the registration process, the CRM aims to eliminate manual data entry, ensure real-time capacity checks, and enhance attendee communication, thereby improving the overall event experience. A further goal is to empower event managers with an intuitive interface for better event management and attendee tracking. For management, the objective is to provide actionable insights into event performance through comprehensive reports and dashboards, linking event data directly to business value and enabling data-driven decision-making.

**Phase 1: Problem Understanding & Industry Analysis**

1. Requirement Gathering  
   Functional Requirements

* Event organizers should be able to create and manage events with details like name, date, time, and capacity.
* Attendees should be able to register for a specific event.
* A system should automatically send email confirmations to registered attendees.
* The system should prevent an event from being overbooked.
* Organizers should be able to view a list of all attendees for a specific event.

Non-Functional Requirements

* Data security: Only authorized organizers can create and manage events.
* Performance: The system should handle attendee registrations quickly and without delays.
* Usability: The interface for creating and managing events should be simple and intuitive for organizers.
* Scalability: The system should support multiple events and thousands of attendees.

1. Stakeholder Analysis

|  |  |  |
| --- | --- | --- |
| **Stakeholder** | **Role in the System** | **Needs/Expectations** |
| Event Organizers | Manages the creation and logistics of events | Easy-to-use interface, accurate attendee counts, reliable communication tools. |
| Community Members | End users who register for events | A simple and fast registration process, timely email confirmation. |
| Volunteers | Optional stakeholders who might assist with event day logistics | A quick way to check attendee lists on the day of the event. |
| Admin | Manages platform operations and security | Control over data, ability to monitor system activity and user access. |

1. Business Process Mapping  
   Step 1: Event organizer creates a new event record in Salesforce.  
   Step 2: A community member registers for the event (by creating a new Attendee record).  
   Step 3: The system automatically checks if the event is at full capacity.  
   Step 4: If the event is not full, the system creates the Attendee record.  
   Step 5: The Attendee receives an automated email confirmation.  
   Step 6: Event organizers can view reports on event attendance and check-ins.
2. Industry-Specific Use Case Analysis  
   Community Industry: The community needs a simple, free solution to connect with its members and manage gatherings efficiently.  
   Technology/CRM Industry: Salesforce provides a scalable, secure, and customizable platform for managing event organizer-attendee interactions without the need for manual data entry or spreadsheets.
3. AppExchange Exploration  
   Before custom development, we explored Salesforce AppExchange for existing solutions:

* **Event Management Apps:** Most apps were designed for complex, ticketed events (e.g., conferences, large-scale paid events) and were too costly or had unnecessary features for a simple, non-profit use case.
* **Non-profit Cloud Solutions:** While many of these exist, they were overly complex for the project's core requirement of simple event and attendee management.

# Phase 2: Org Setup & Configuration

Company Profile Setup

Set basic org details under Setup > Company Information > Edit:

* **Name:** Project/Org name .
* **Time Zone:** Default working timezone for your location.
* **Locale:** Controls date/number formats.
* **Language:** Default UI language for new users.
* **Currency:** Corporate currency.

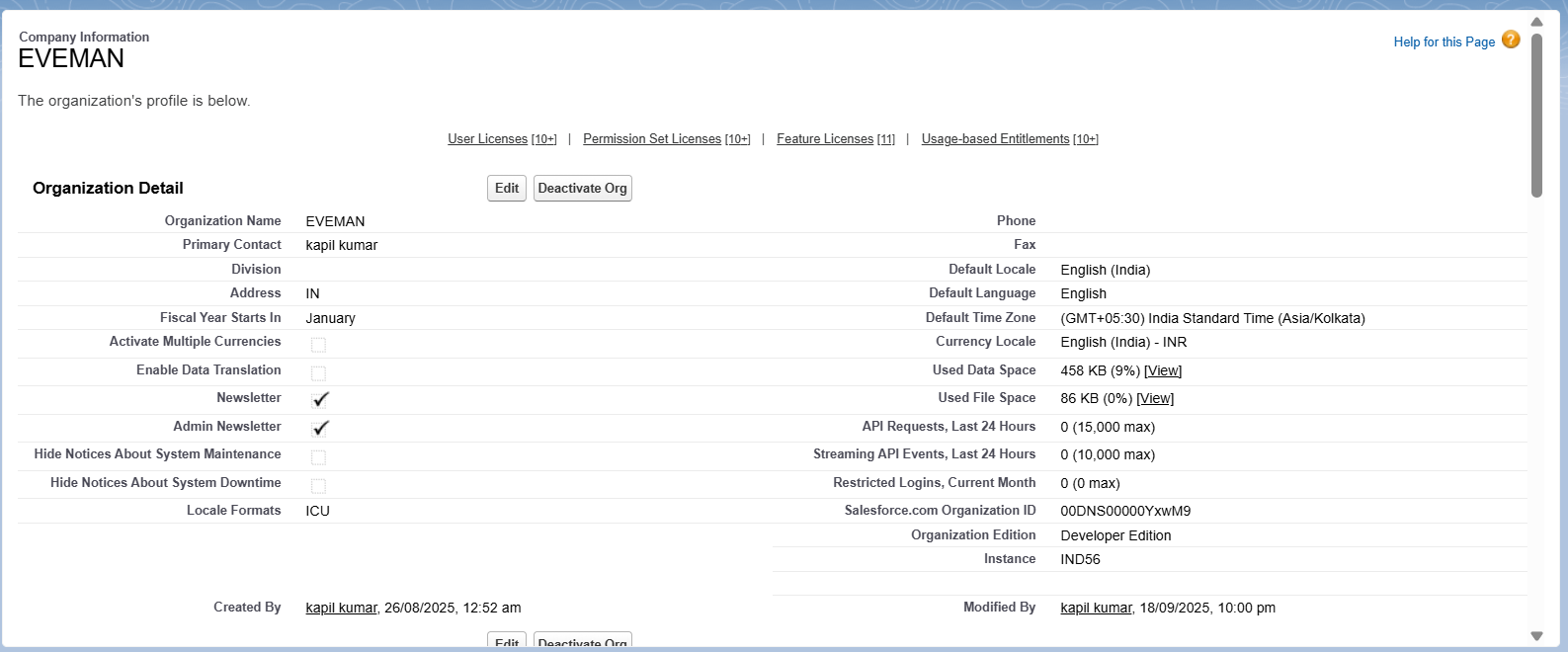
User Setup

For this project, a single System Administrator user is sufficient.

Dev Org Setup

* To implement this project, a **Salesforce Developer Edition** org was set up.
* A **GitHub Repository** was created for source control to manage versions of your project files.

**VS Code** with the **SFDX CLI** was set up for a streamlined development experience, allowing for easy creation and deployment of Apex classes and triggers.



# Project Title – “Simple Event Management & Attendee Tracking System”

## Phase 3: Data Modeling & Relationships

### Standard & Custom Objects

For this project, we primarily used custom objects. The standard Contact object could be used for attendees in a more advanced project, but we've created a custom Attendee object to keep the scope focused.

* **Event\_\_c** (Custom): Tracks event details such as name, date, and location.
* **Attendee\_\_c** (Custom): Stores attendee information, including name, email, and phone number.

### Master-Detail Relationship

To connect attendees to a specific event, a Master-Detail relationship was created. This ensures every attendee record is directly linked to an event and inherits its security and sharing settings.

* **Master Object:** Event\_\_c
* **Detail Object:** Attendee\_\_c
* **Why:** An attendee must be linked to an event, and the deletion of an event should cascade and delete all related attendees. This relationship also allows for roll-up summary fields (though not used in this project) and ensures consistent data.

### Fields for Event\_\_c

To capture essential event details, the following custom fields were added:

* **Max Capacity** (Max\_Capacity\_\_c): Number
* **Date** (Date\_\_c): Date
* **Time** (Time\_\_c): Text
* **Location** (Location\_\_c): Text Area
* **Description** (Description\_\_c): Long Text Area

### Fields for Attendee\_\_c

To capture essential attendee information, the following custom fields were added:

* **Email** (Email\_\_c): Email
* **Phone** (Phone\_\_c): Phone
* **Event** (Event\_\_c): Master-Detail Relationship, linked to Event\_\_c.

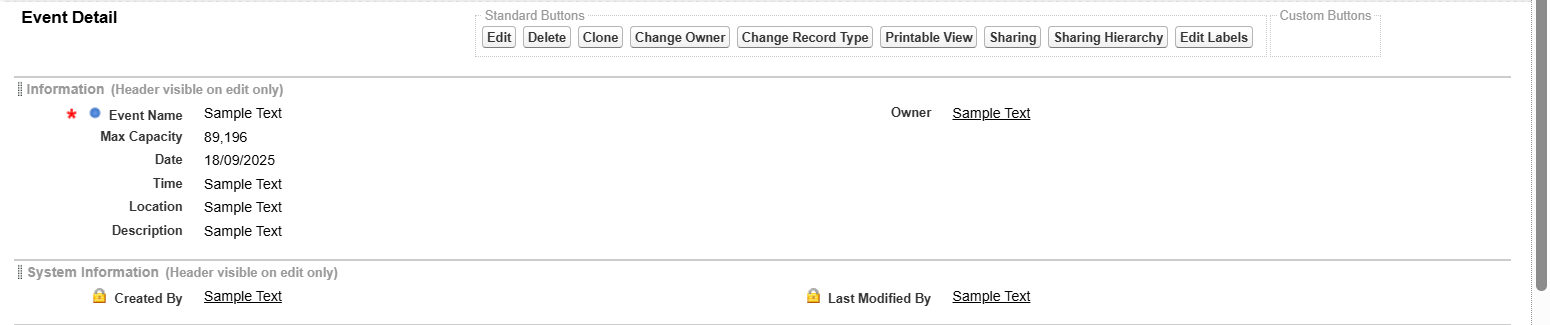
### Schema Builder

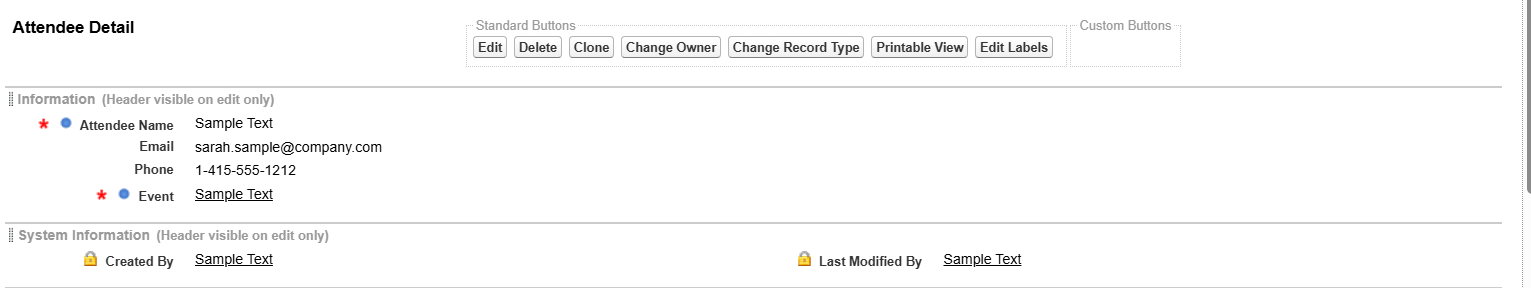
The Schema Builder provides a visual representation of your data model. In your org, you will see a clean, direct relationship between your custom Event and Attendee objects.

### Page Layouts & Compact Layouts

Page layouts were created to organize fields and related lists for easy data entry and viewing. Compact layouts were also configured to display key information in the highlights panel.

* **Event Page Layout:** Includes details fields and a related list for Attendees.
* **Attendee Page Layout:** Includes details fields and a lookup field for Event.
* **Compact Layouts:** The Event and Attendee compact layouts were configured to display key fields in the highlights panel, providing a quick overview of the record.





# Phase 4: Process Automation (Admin)

This document outlines the core declarative automation implemented for the "Simple Event Management & Attendee Tracking System." This automation enforces data quality and automates key processes to improve efficiency.

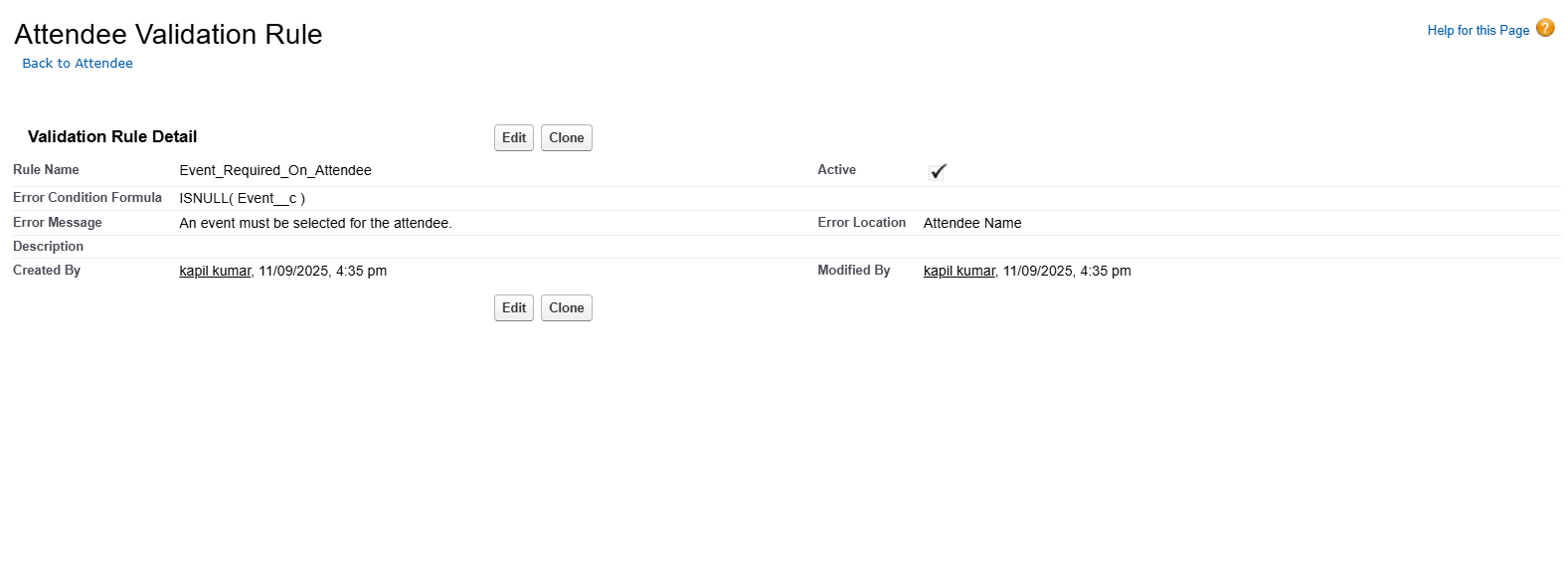
### Validation Rule: Max Capacity Check

1. **Purpose:** To prevent the creation of an Event with an invalid capacity, ensuring data accuracy and is a prerequisite for the overbooking check in Phase 5.
2. **Rule Name:** Max\_Capacity\_Must\_Be\_Positive
3. **Error Condition Formula:** Max\_Capacity\_\_c < 1

**Error Message:** "Maximum Capacity must be a number greater than 0."

1. **Error Location:** The Max Capacity field.

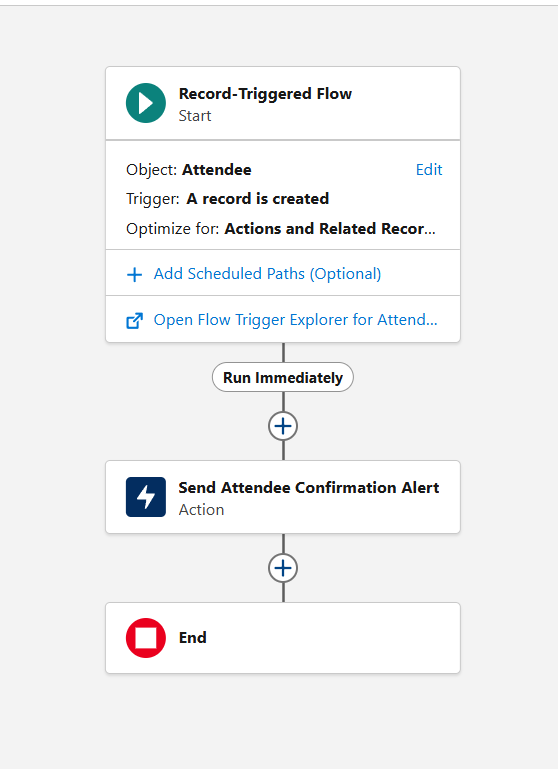
**Impact:** Ensures every event has a valid capacity, preventing errors in downstream automation.

****

### Flow Builder: Attendee Confirmation

1. **Purpose:** To automate the communication process by sending an instant confirmation email to a new attendee upon registration.
2. **Type:** Record-Triggered Flow.
3. **Object:** Attendee.
4. **Trigger:** A record is **Created**.
5. **Action:** **Send Email** action.
6. **Email Template:** Attendee Confirmation (Lightning Email Template).

**Impact:** This flow provides instant feedback to the registrant and eliminates the need for manual confirmation, saving time for the event manager.



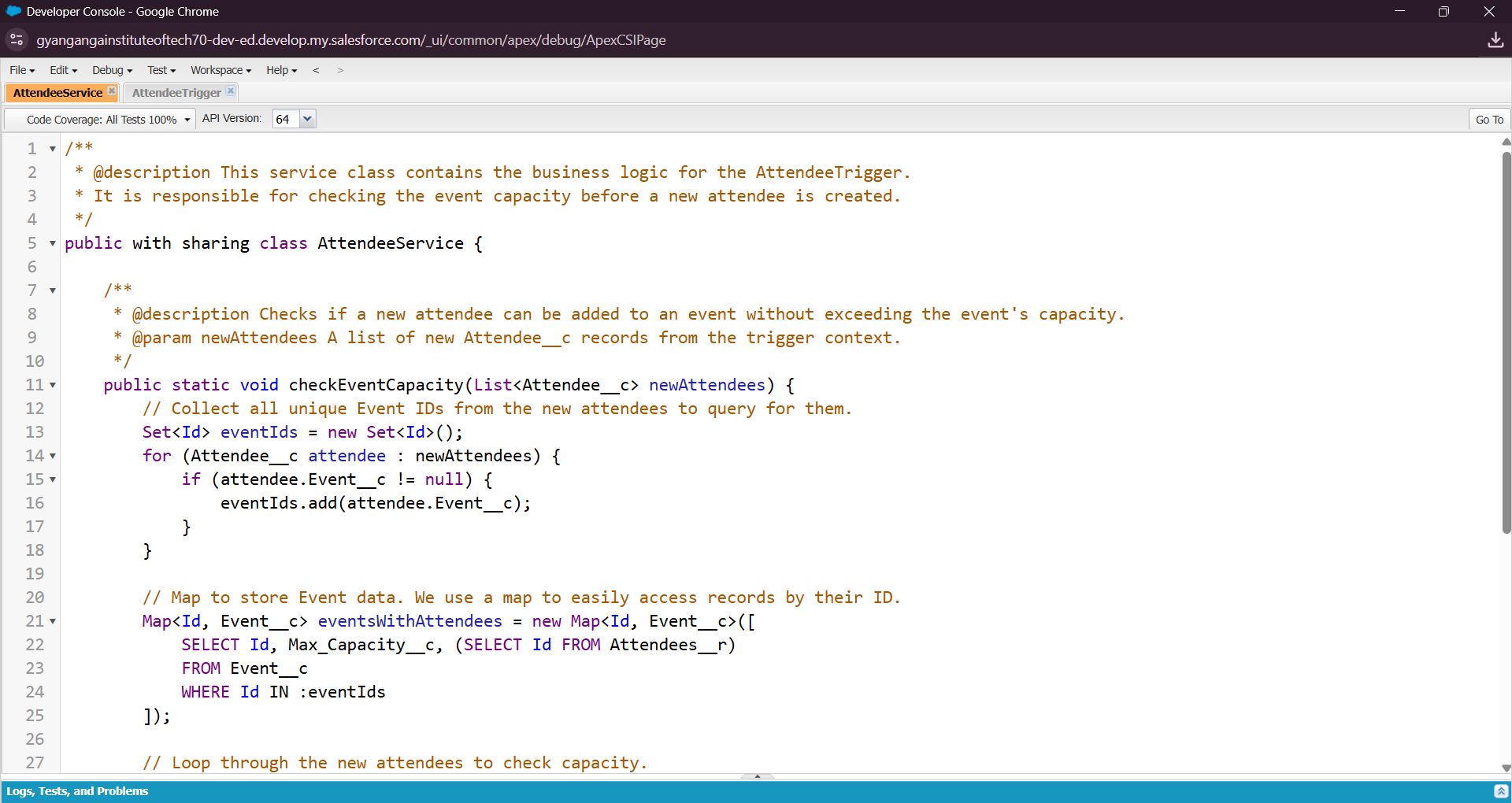
# Phase 5: Apex Programming (Developer)

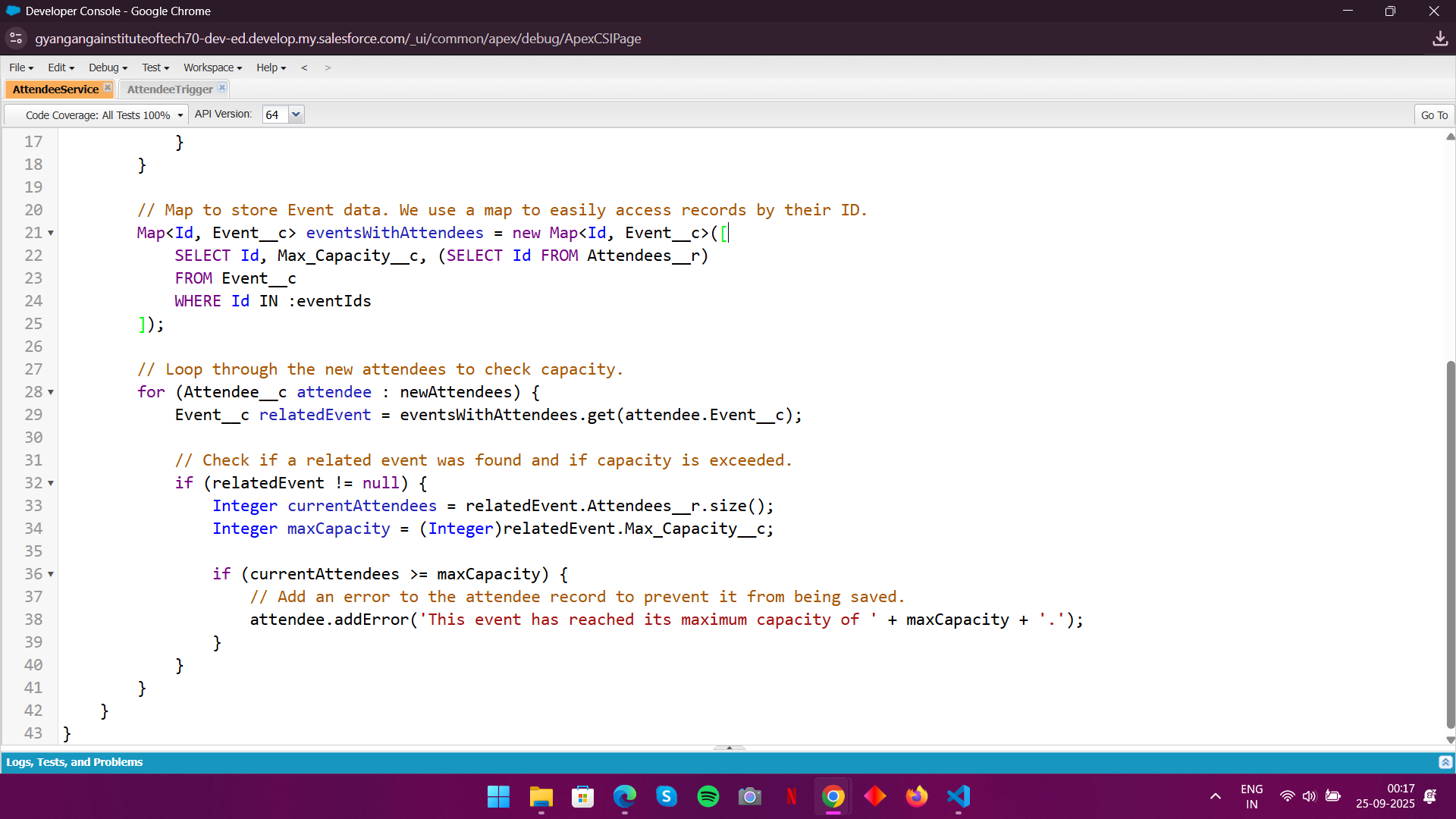
This document outlines the core Apex programming components for the "Simple Event Management & Attendee Tracking System." This phase focuses on server-side logic to enforce a key business rule: preventing overbooking.

### Classes & Objects

The primary purpose of the AttendeeService class is to contain the logic for our Apex Trigger. This class is designed to check an Event's capacity before a new Attendee record is saved.

* **Business Logic Implemented:** The class checks the number of existing Attendee records for a given Event.
* **Capacity Check:** If the count of existing attendees is equal to or greater than the Max\_Capacity\_\_c field on the Event, the class returns an error.

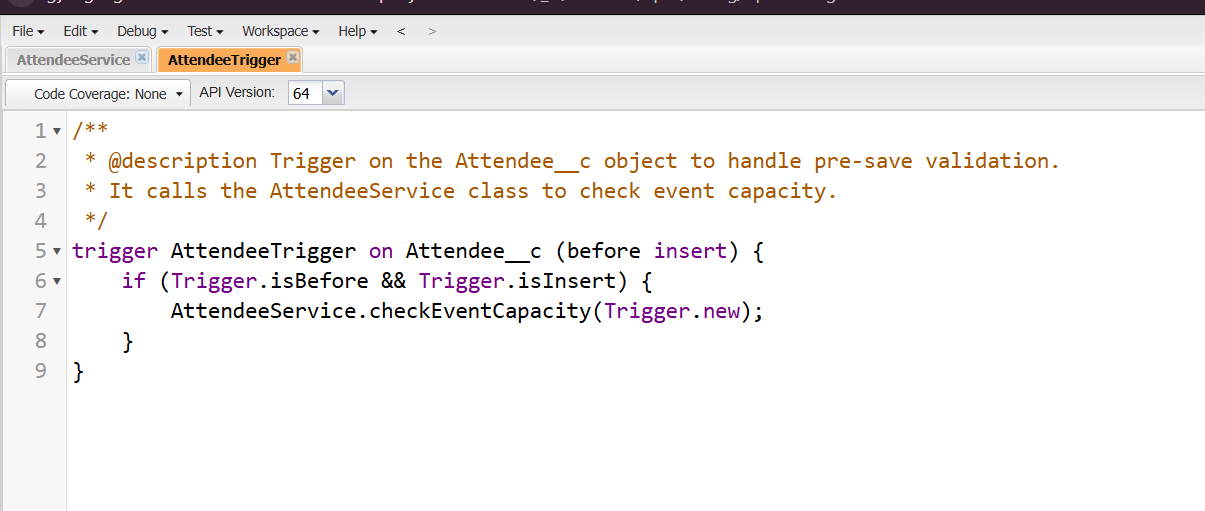




### Apex Triggers (before/after insert/update/delete)

We created a single **Apex Trigger** to call the logic in our service class.

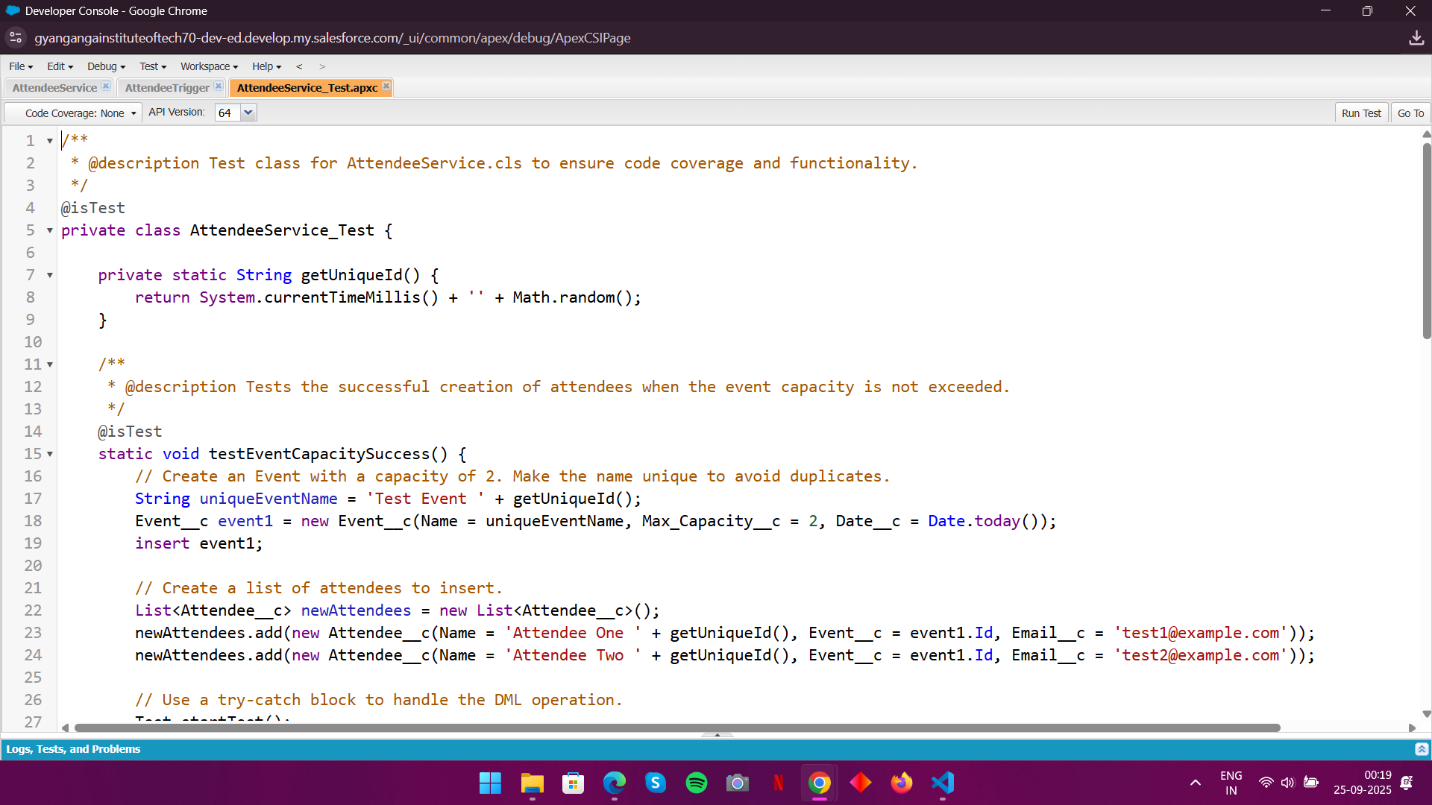
* **Trigger Name:** AttendeeTrigger
* **Object:** Attendee
* **Trigger Events:** before insert
* **Purpose:** To prevent a user from creating a new Attendee record if the parent Event has already reached its maximum capacity.
* **Impact:** This ensures data integrity by preventing overbooking and eliminates the need for manual checks by the event manager.

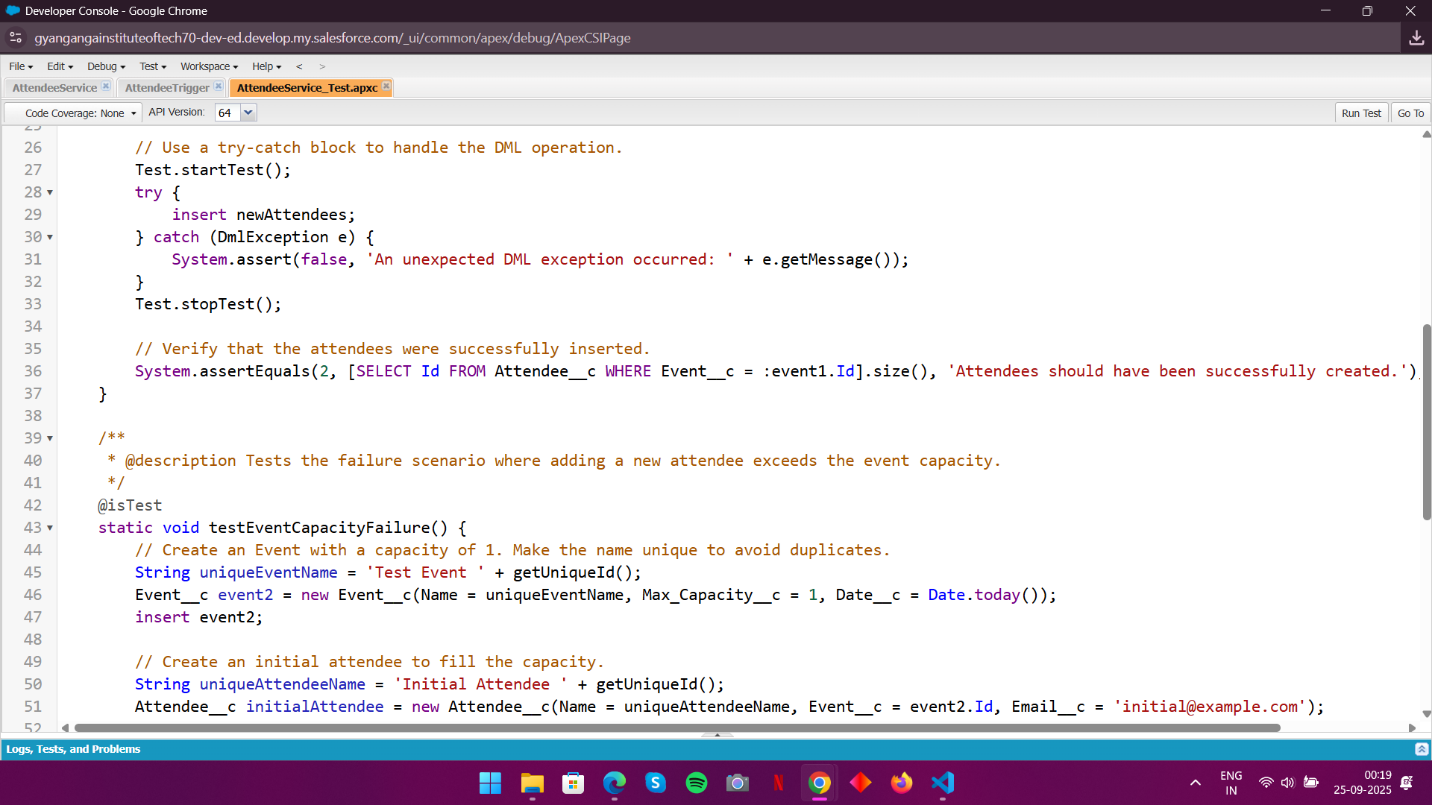


### Test Classes

A formal **Apex Test Class** (AttendeeService\_Test.cls) was created to ensure our code works as expected.

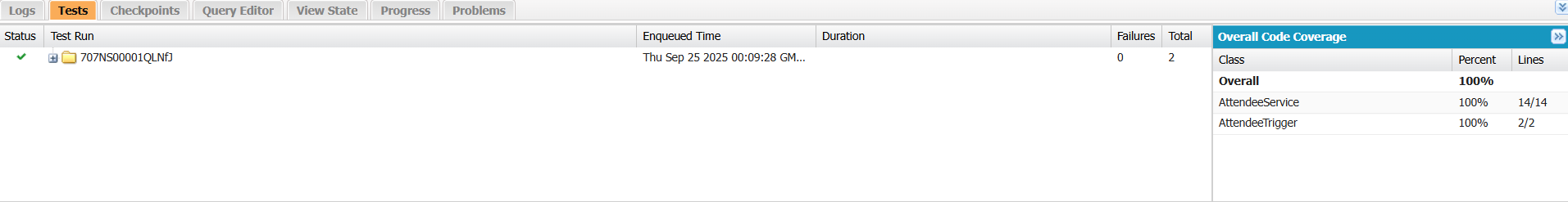
* **Code Coverage:** The test class provides sufficient code coverage to meet the Salesforce deployment requirement of ≥75%.
* **Verification:** It simulates the creation of Attendee records for an event and verifies that the trigger correctly prevents a record from being saved when the event is at capacity.





**Test case Coverage**

Showcasing the 100 percent coverage of the code implemented.



**Phase 6: User Interface Development**

This document outlines the user interface components of the "Simple Event Management & Attendee Tracking System." The focus of this phase is on using declarative tools like the Lightning App Builder to create a userfriendly and efficient interface without the need for custom code.

# Record Pages

Record pages were customized for the core objects to provide event managers with an at-a-glance view of all critical information.

**1. Event Record Page**

**Purpose:** To give managers a central hub for all information related to a single event.

**Step-by-Step Process:**

Go to **Setup** → **Object Manager** → **Event**.

Click on **Lightning Record Pages**.

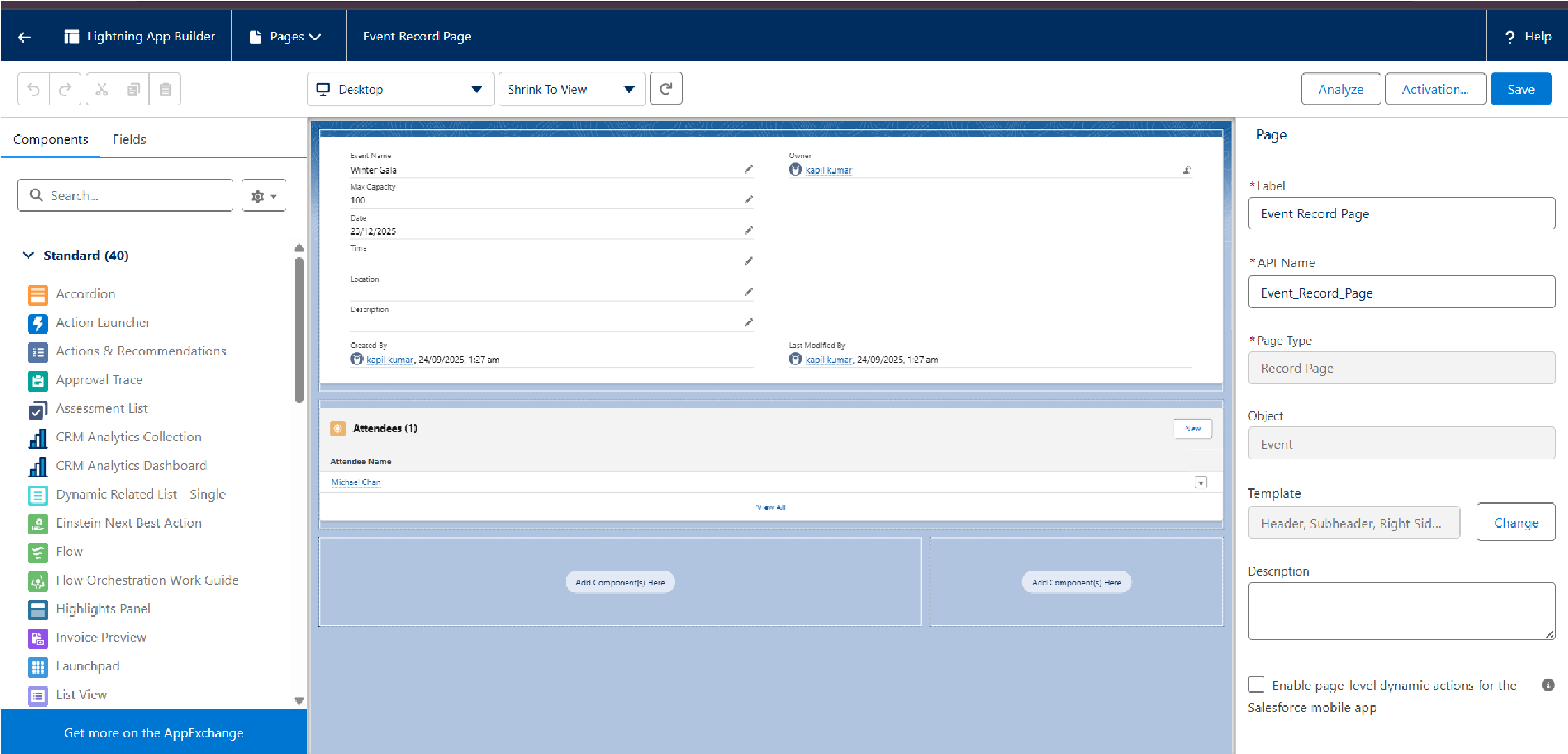
Click **New**, select the "Record Page" option, and give it a name (e.g., "Event Record Page"). Use the **Lightning App Builder** to drag and drop standard components to create a two-column layout.

Add the Highlights Panel component to the top of the page.

Add the Record Detail component to the main canvas to display all fields.

Drag the **Attendees related list** component onto the page.

Click **Save** and then **Activate** to assign the page as the default for all users and apps.



**2. Attendee Record Page**

**Purpose:** To provide a clean, focused view of a single attendee's details.

**Step-by-Step Process:**

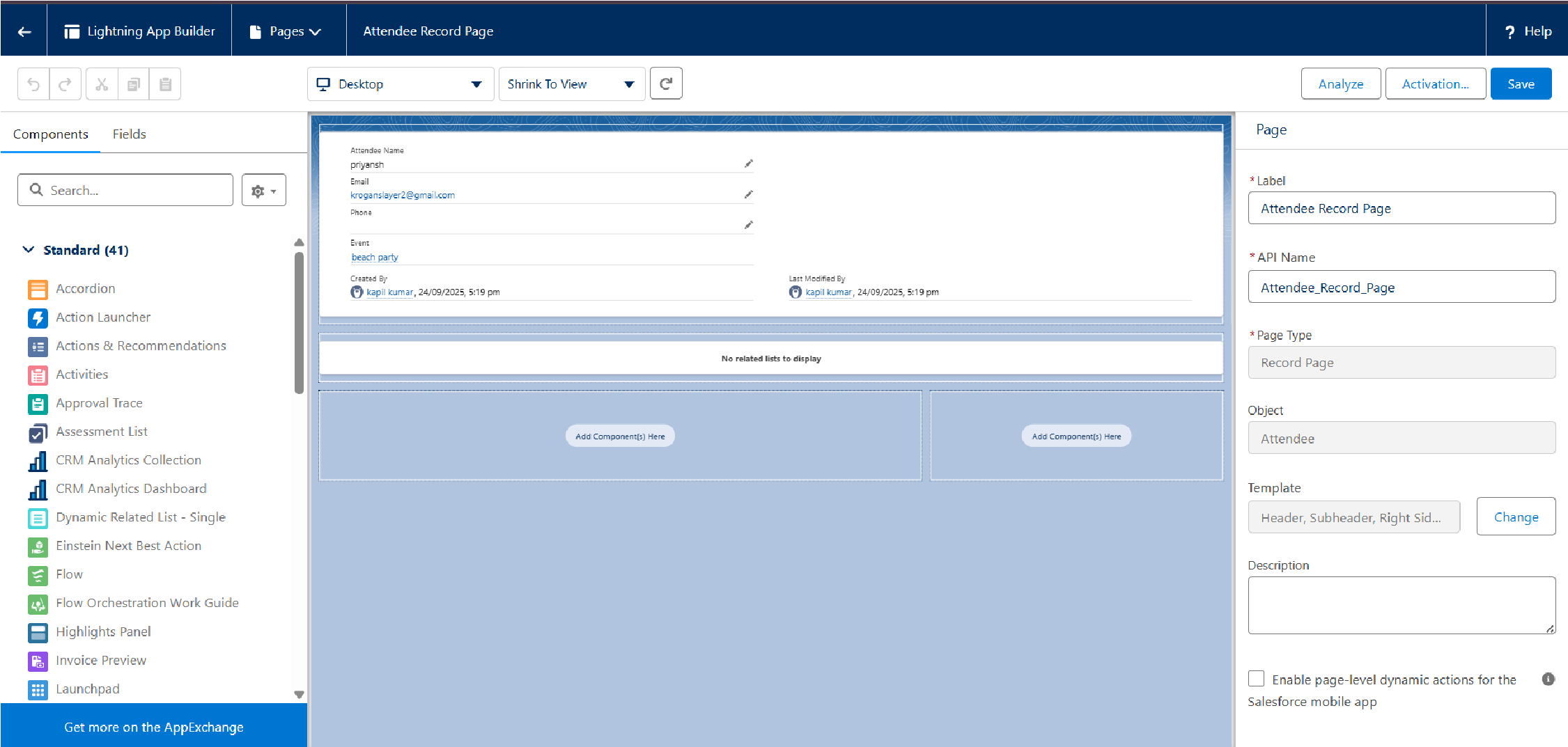
Go to **Setup** → **Object Manager** → **Attendee**.

Click on **Lightning Record Pages**.

Click **New**, select "Record Page," and give it a name (e.g., "Attendee Record Page").

Use the **Lightning App Builder** to drag and drop the Highlights Panel and Record Detail components. Ensure the Event lookup field is prominently displayed to show which event the attendee is registered for.

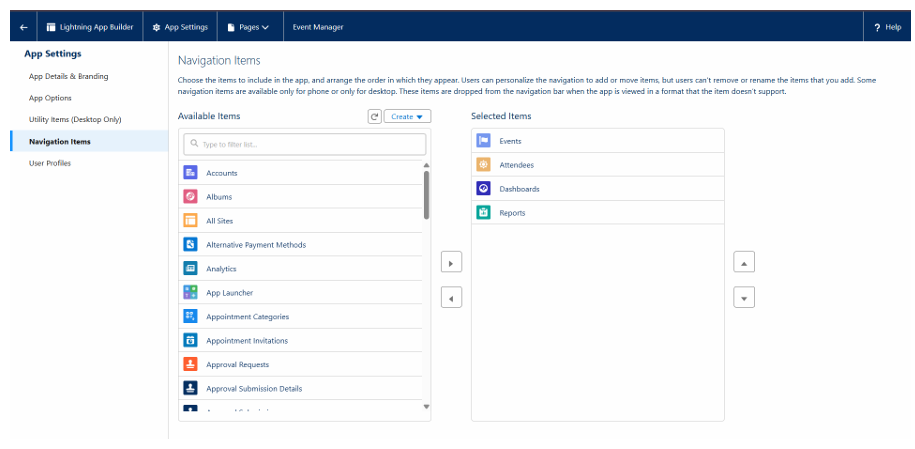
Click **Save** and then **Activate** to assign the page as the default.

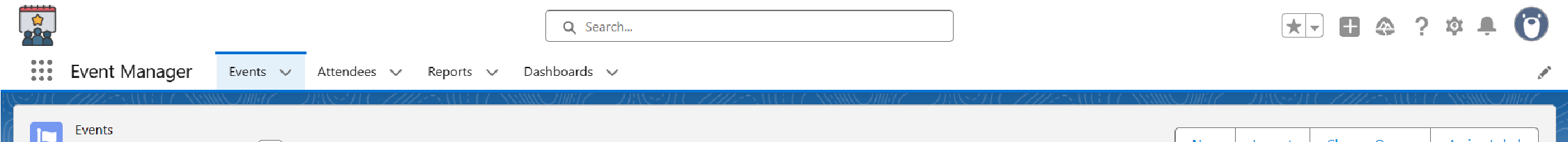


# Tabs

Custom tabs were created to allow users to navigate the application and access records.

* Step-by-Step Process: Go to Setup → App Manager.
* Find your "Event Manager" Lightning App and click Edit.
* Go to Navigation Items and use the shuttle to add the Events and Attendees tabs to the navigation bar.
* Click Save





# Home Page Layouts

The Home Page layout was customized to provide relevant information for a user as soon as they log in to the "Event Manager" app.

**Step-by-Step Process:**

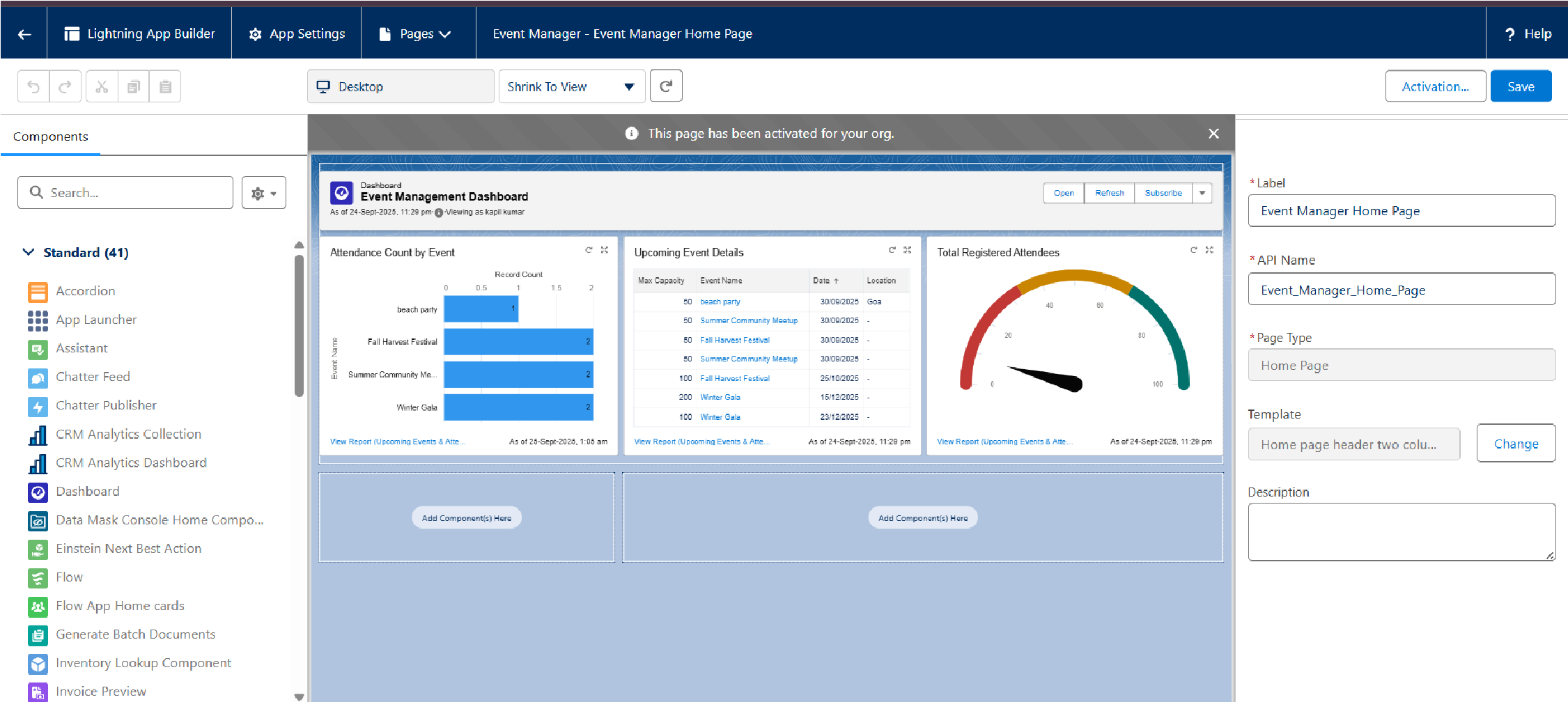
Go to **Setup** → **Lightning App Builder**.

Click **New** and select the "Home Page" option. Give it a name (e.g., "Event Manager Home Page").

Drag the **Dashboard** component onto the canvas.

From the component properties panel, select your **"Event Management Dashboard."**

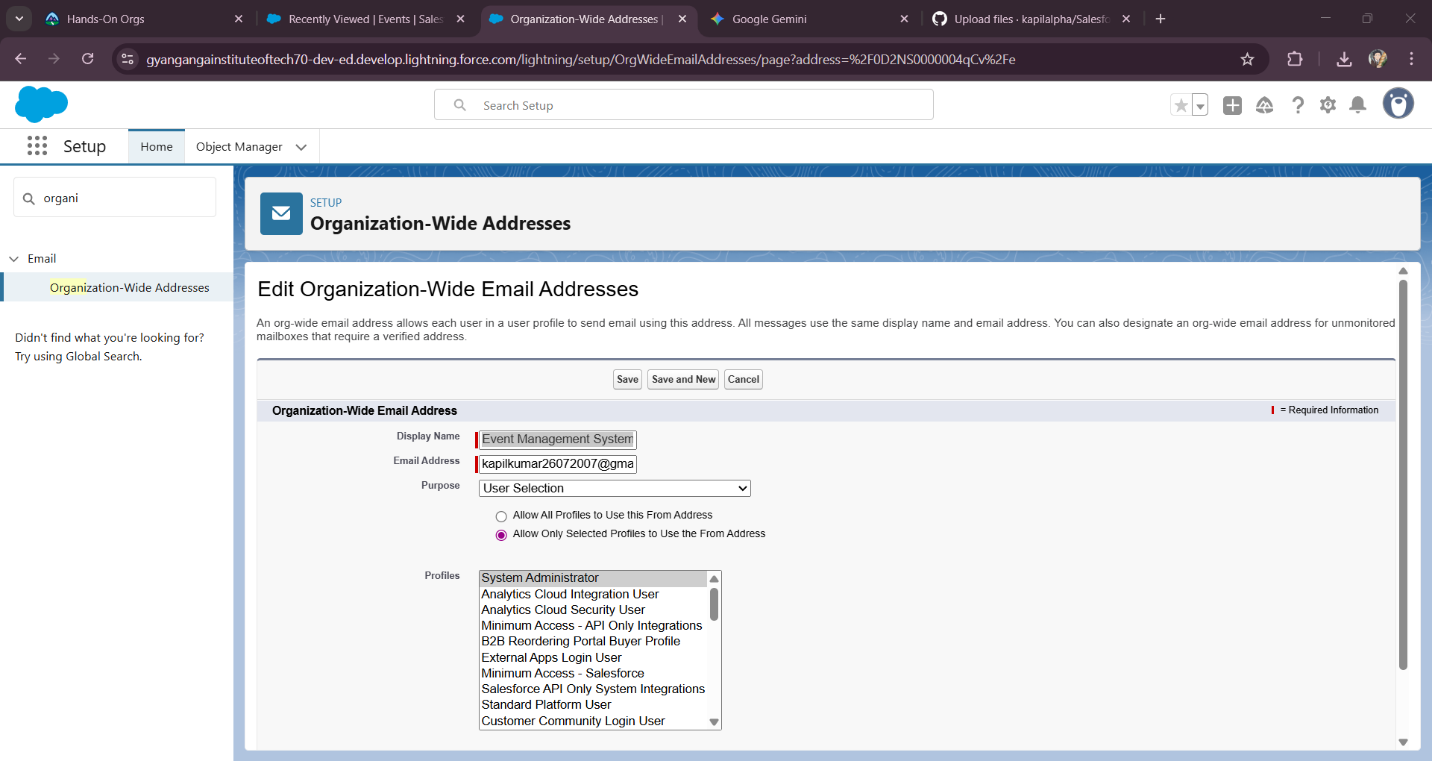
Click **Save** and then **Activate** to assign the page as the default for the "Event Manager" app.



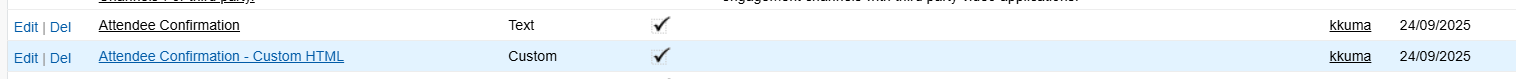
### Phase 7: Integration & External Access

**Goal:** To establish the necessary Admin configurations to enable the automated email confirmation and ensure the system can communicate securely.

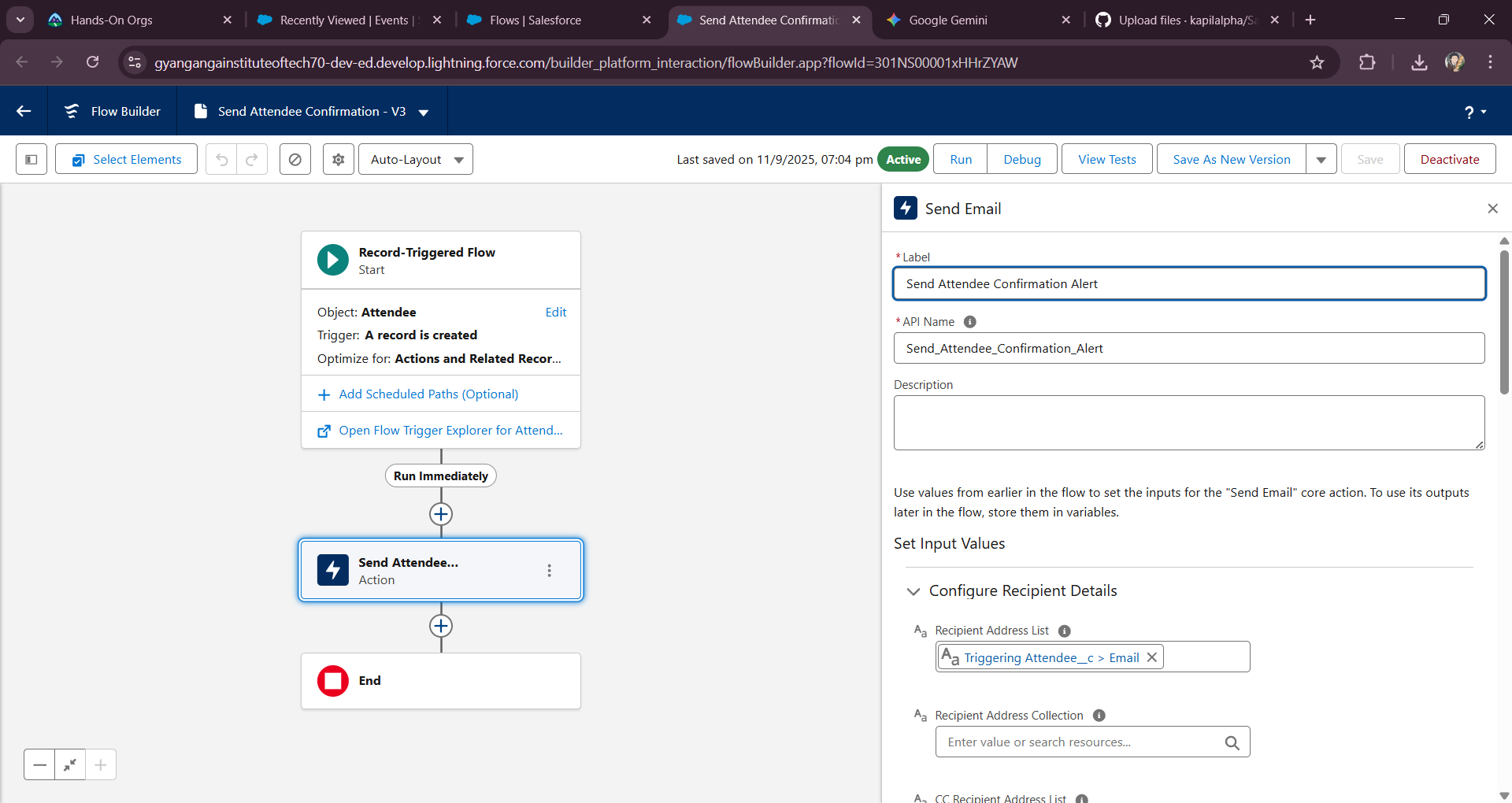
1. **Configure Organization-Wide Email Address:** Go to **Setup** and search for Organization-Wide Addresses. Create a new entry and provide a display name (e.g., Event Manager) and an email address. This is the **"From"** address for all automated emails sent from the system. You will need to verify this email address.



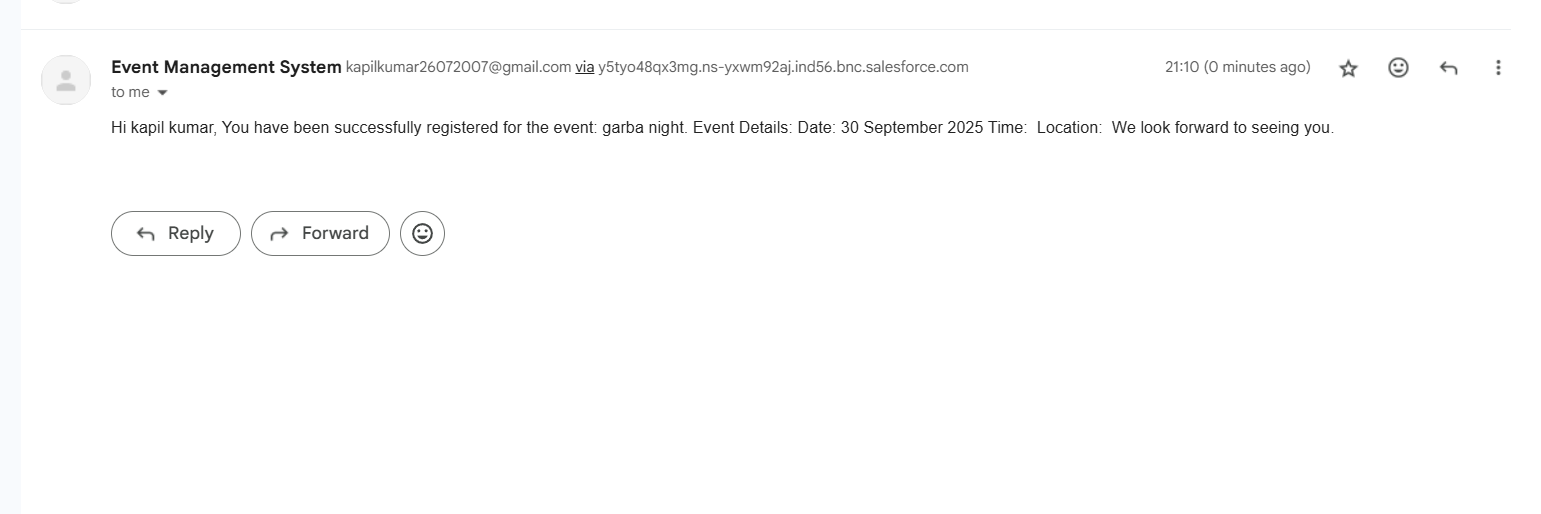
1. **Update Email Alert:** Go to **Setup** → **Email** → **Classic Email Alerts**. Find the **Email Alert** we created previously and select the new **Organization-Wide Email Address** you just configured.



1. **Flow Configuration Review:** Open your **Record-Triggered Flow** and double-check that the **Email Alert** action is correctly configured and is called after the Attendee record is created.



1. **Test End-to-End Functionality:** Create a new **Event** and a new **Attendee** record. Verify that the **confirmation email is sent successfully** and uses the Organization-Wide Email Address you set up.

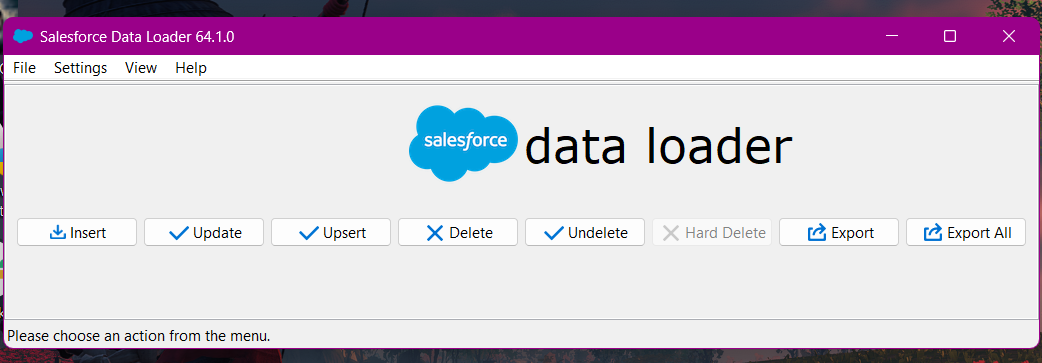


### Phase 8: Data Management & Deployment

**Goal:** To load sample data, migrate the finished application from the development environment to the live production org, and perform final checks.

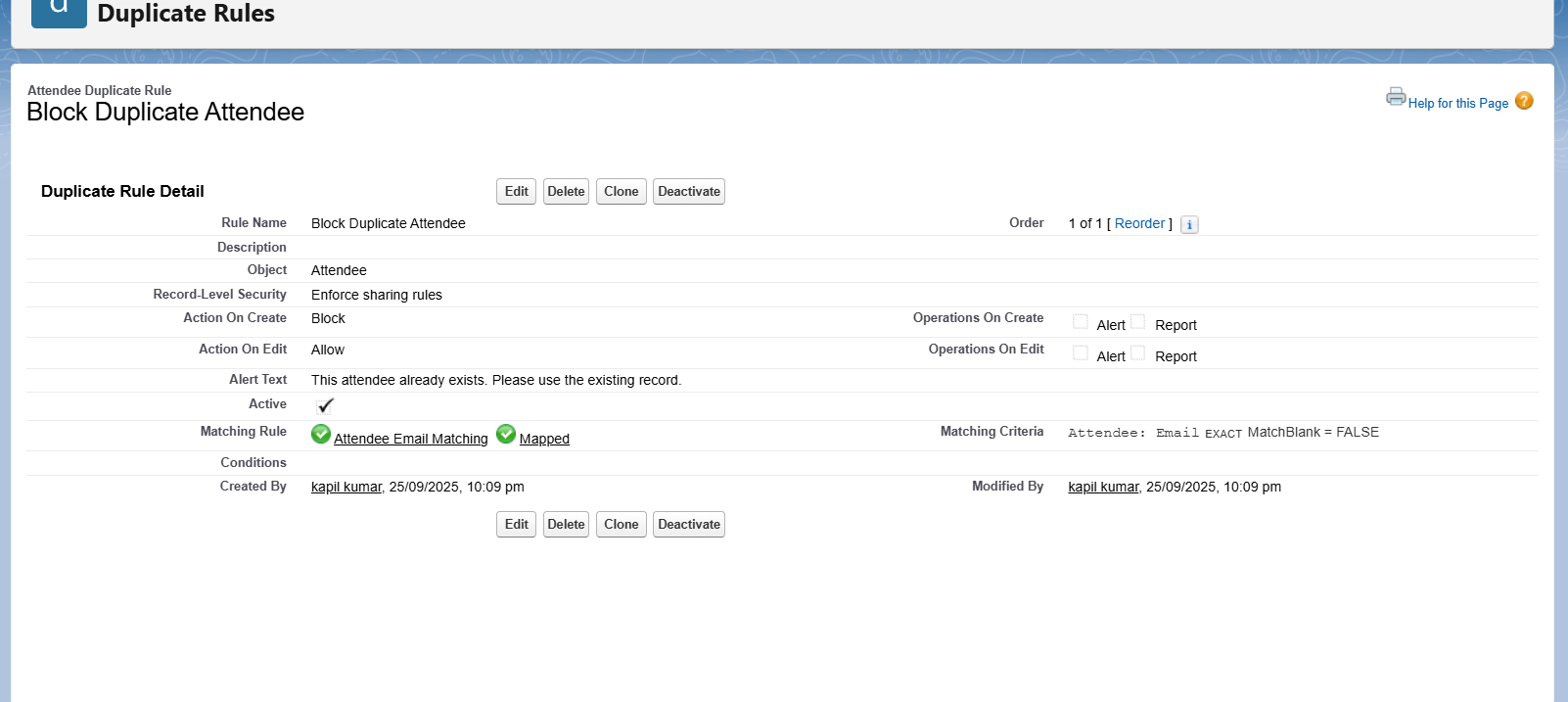
**1. Data Loader (Data Import)**

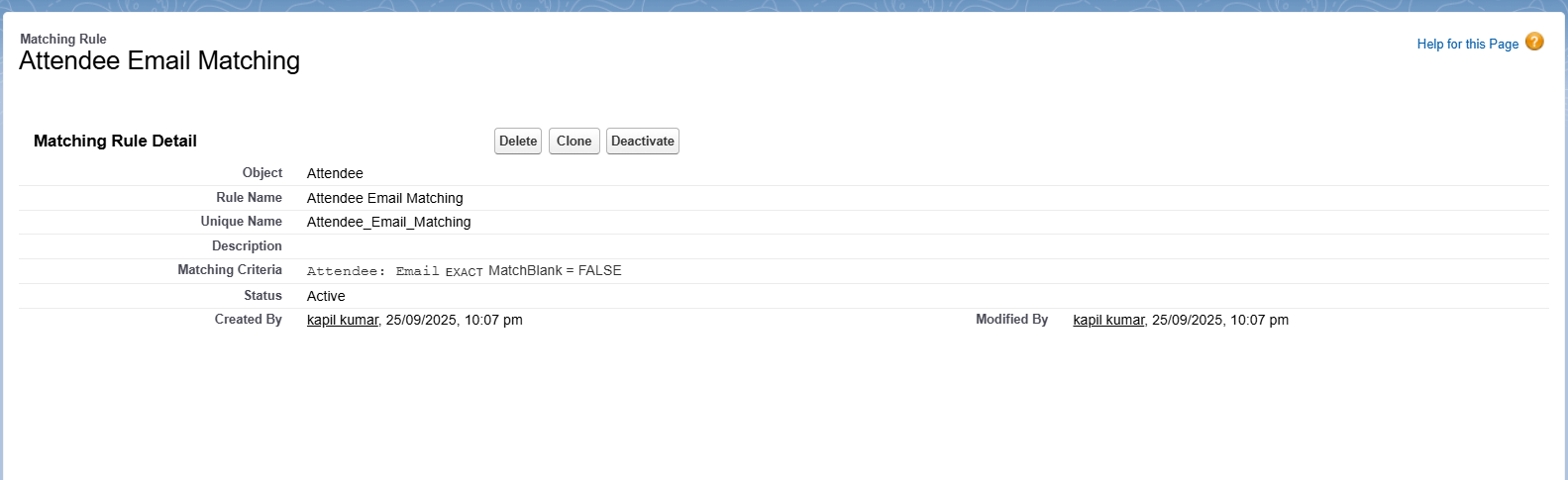
* Use the **Data Loader** desktop application to import sample data into your development environment before final deployment. This is crucial for testing with realistic data.
* First, use the **Insert** function to load parent **Event** records from a CSV file.
* Once the events are imported, export them to get their unique Salesforce IDs.
* Update your Attendee CSV file by adding a column for the Event ID and mapping each attendee to a parent Event record.
* Finally, use the **Insert** function again to load the child **Attendee** records, ensuring they are correctly linked to their parent Event.



**2. Duplicate Rules**

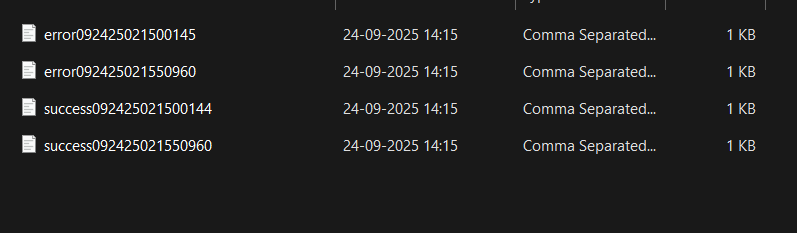
* Configure **Duplicate Rules** on the Attendee object to prevent users from creating duplicate records based on the Email\_\_c field. This is a best practice for data integrity.
* Go to **Setup** → **Duplicate Rules**.
* Create a new rule for the Attendee object.
* Configure matching rules to identify duplicates based on a unique identifier like the Email field.
* Set the rule to either **Block** or **Alert** the user when a duplicate is found.





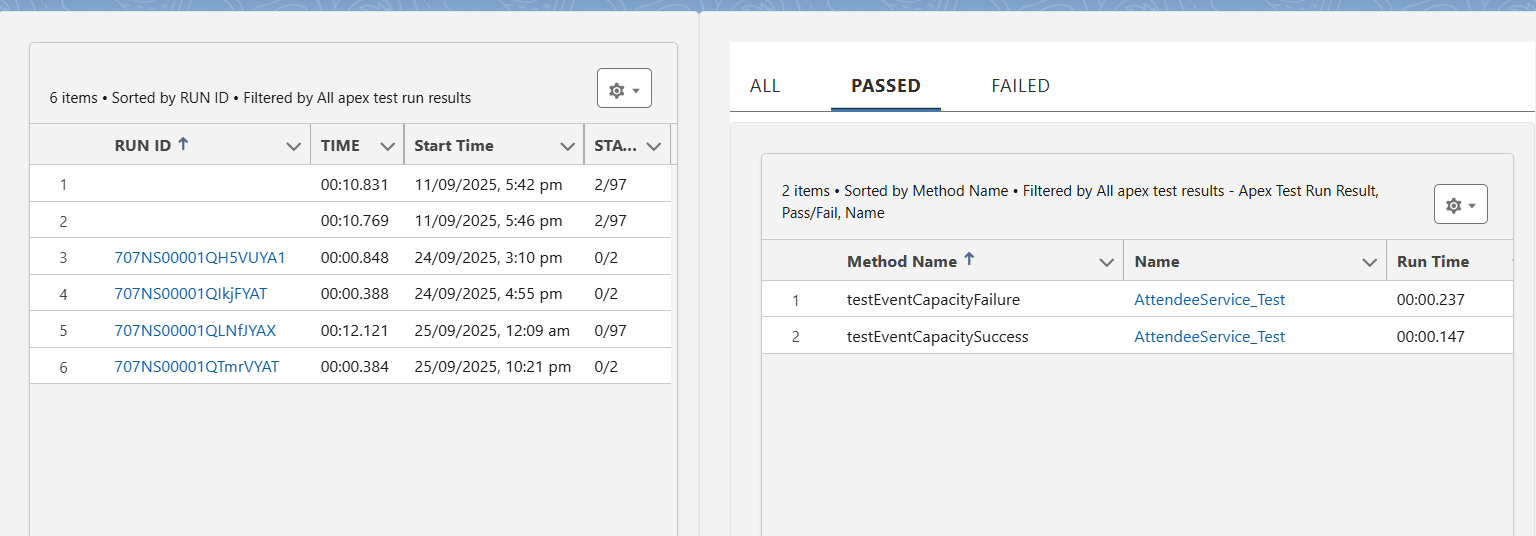
**3. Deployment Preparation (Change Sets or SFDX CLI)**

* Prepare all project components for migration to the live Production environment.
* If using Change Sets, gather all your components (custom objects, fields, flows, Apex classes, triggers, and the permission set) into an **Outbound Change Set**.
* If using the SFDX CLI, ensure all your source code and metadata are up-to-date in your local project folder by running sfdx force:source:retrieve.
* Validate the deployment to the target Production org to check for any errors before going live.



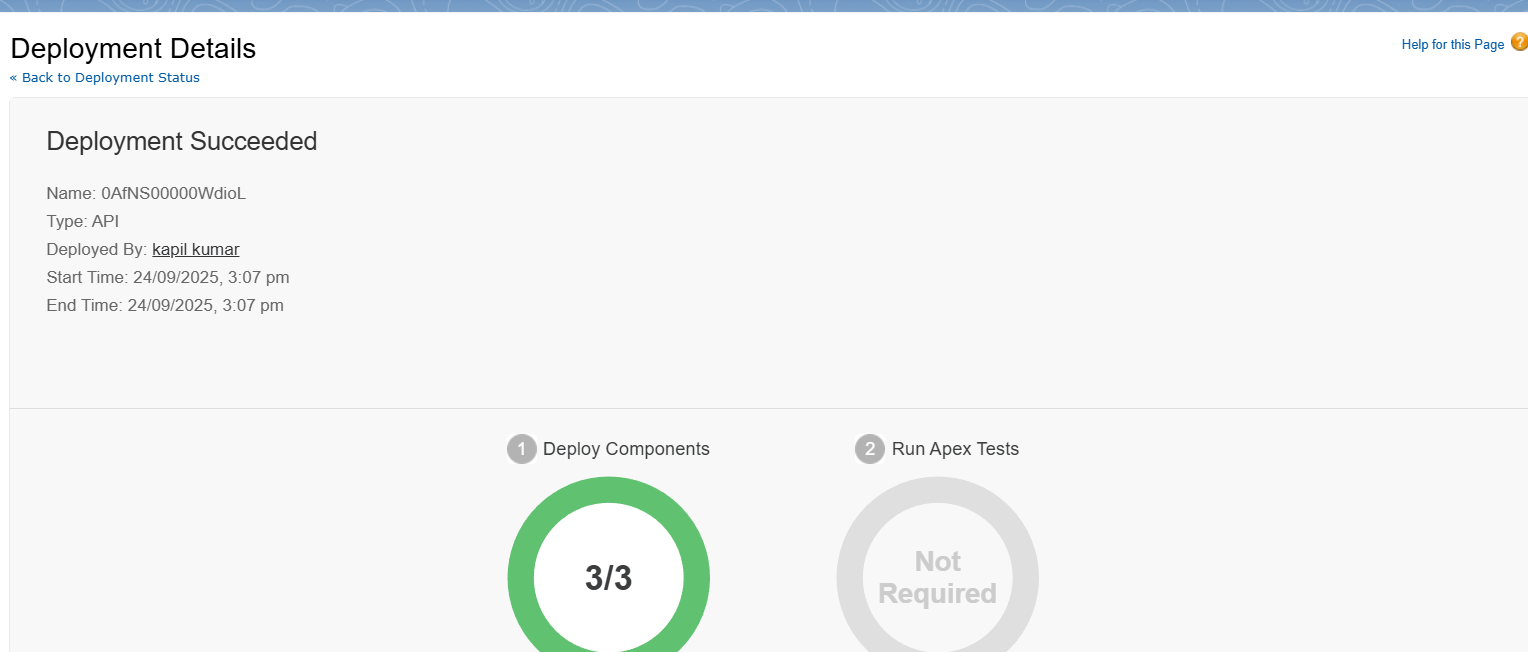
**4. Deployment (Go-Live)**

* Deploy the validated Change Set or use the SFDX CLI command (sfdx force:source:deploy) to migrate all your project components from your development environment to the Production environment.
* **Crucially**, perform this during a low-traffic period to minimize any user impact.



**5. Post-Deployment Checks**

* After deployment, perform a final set of checks in the live Production environment to ensure everything is working as expected.
* **Verify Deployment Status** in Setup to confirm the deployment was successful.
* **Run All Apex Tests** to ensure all your code has the required coverage and no functionality was broken during the migration.
* **Perform Manual Functionality Testing** by creating a new Event and Attendee record to test the core "happy path" (email confirmation) and "unhappy path" (capacity check) scenarios.

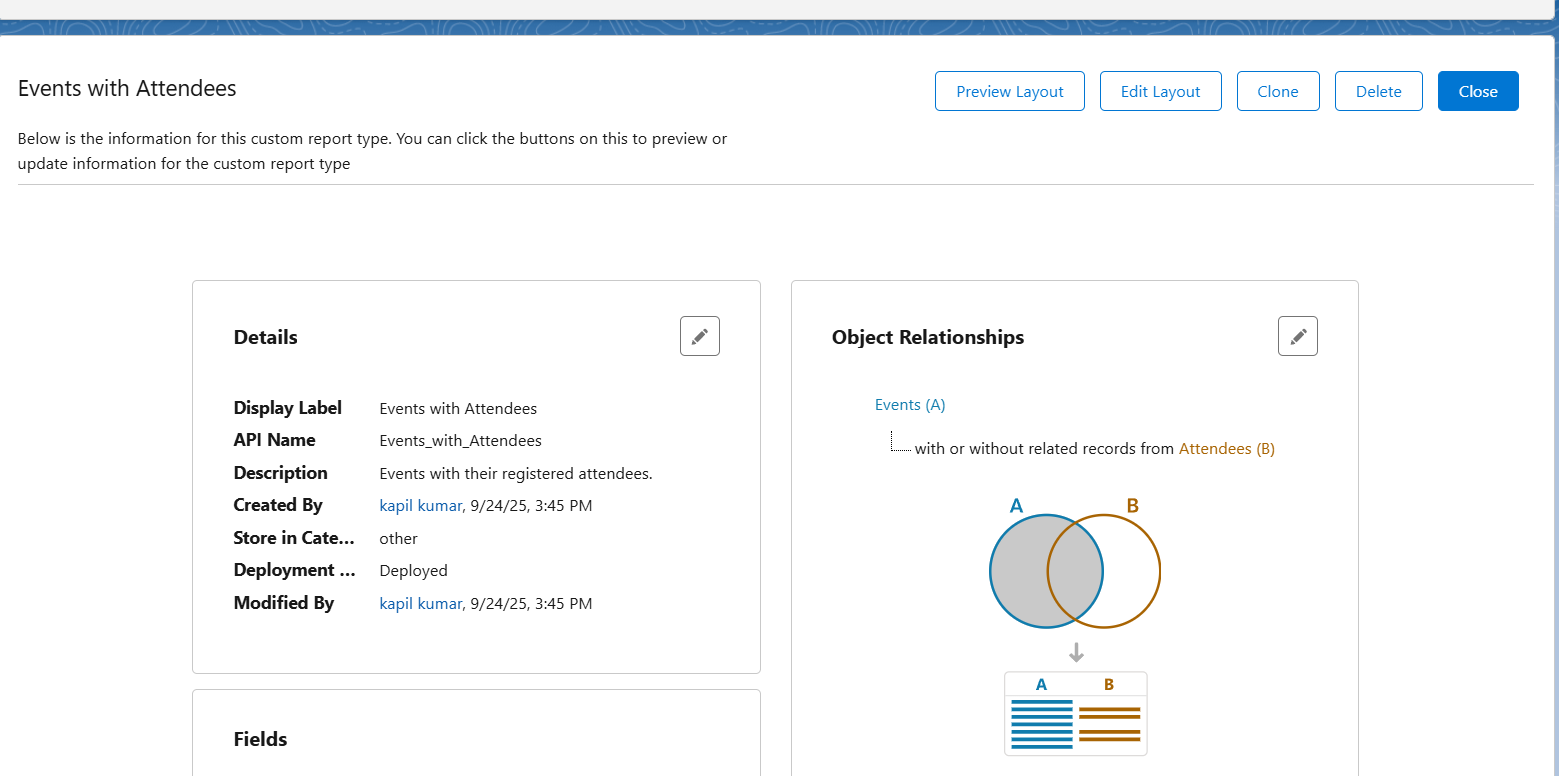


### Phase 9: Reporting, Dashboards & Security Review

**Goal:** To visualize the project data and finalize all security controls, ensuring the application is secure, scalable, and provides valuable business insights.

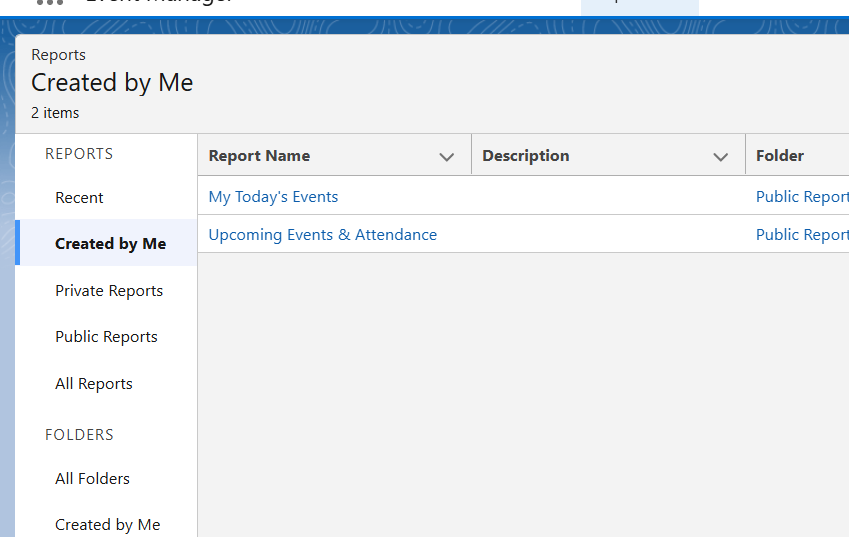
**1. Create Custom Report Type**

* Create a custom report type to report across the related Event and Attendee objects.
* Go to **Setup** → **Report Types**.
* Click **New Custom Report Type**.
* Select **Event** as the Primary Object.
* Relate the **Attendee** object by clicking the link to select a child object.
* Set the deployment status to **Deployed** and save the report type.



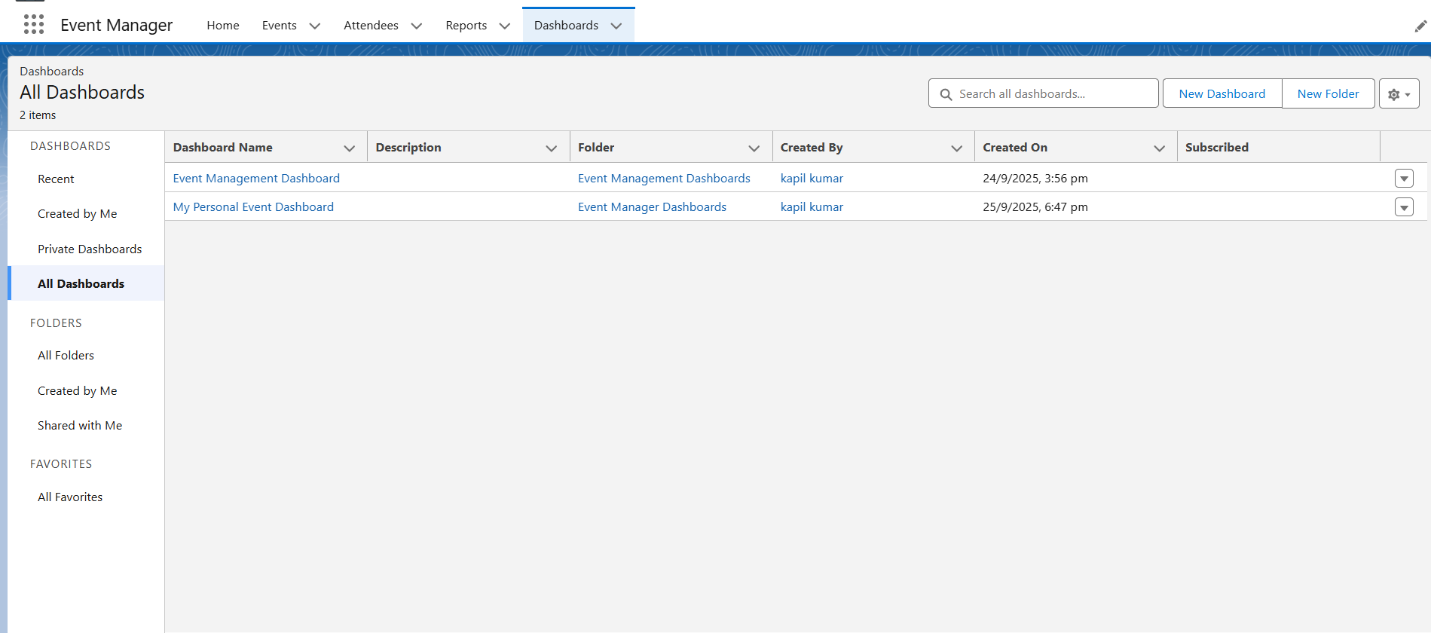
**2. Create Reports**

* Build reports that provide key insights into event data.
* Go to the **Reports** tab and click **New Report**.
* Select the custom report type you just created.
* Add columns for Event Name, Max Capacity, and Attendee Name.
* Apply filters to show relevant data, such as Date greater than TODAY to show upcoming events.
* Save the report in a public, shared folder.



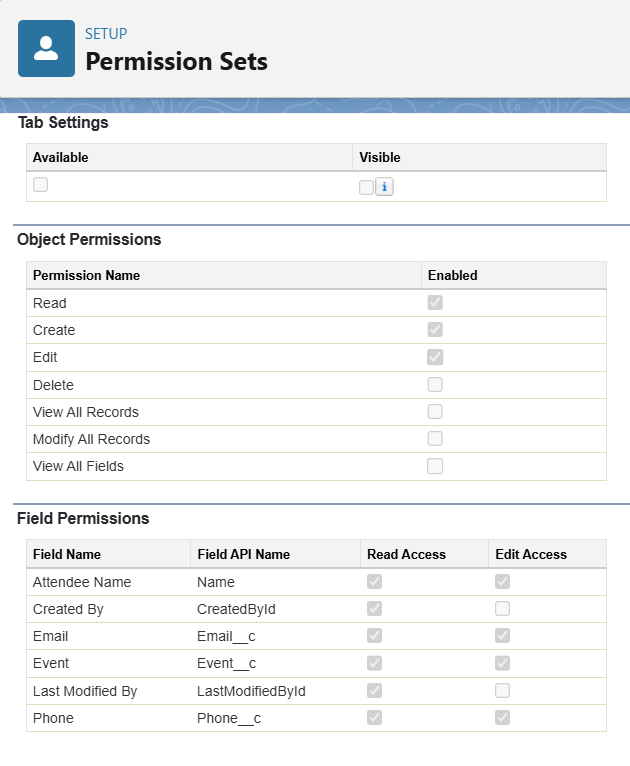
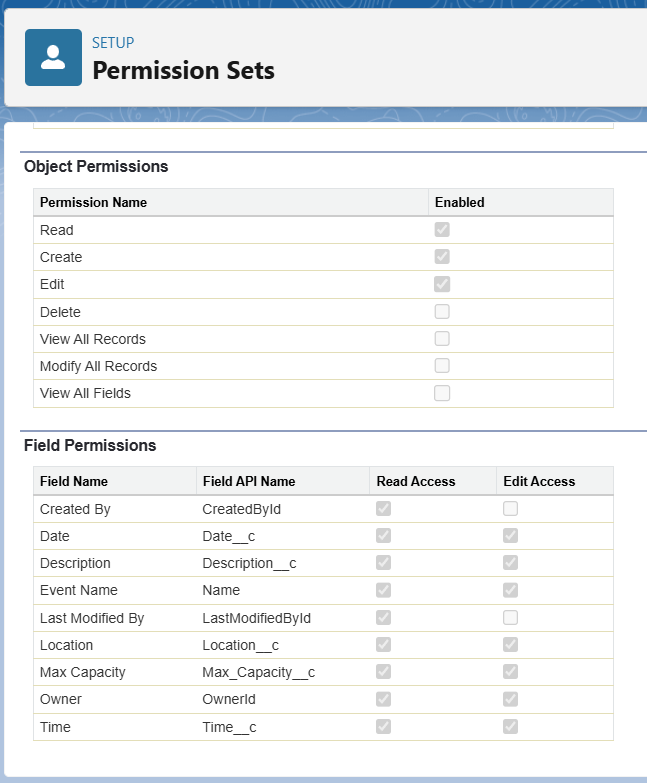
**3. Create Dashboards**

* Visualize your reports in a clear, at-a-glance dashboard.
* Go to the **Dashboards** tab and click **New Dashboard**.
* Save the dashboard to a public folder (e.g., Event Manager Dashboards).
* Add a widget for each report you created. Use charts (e.g., a bar chart for attendance) and tables to display the data.
* Click the **gear icon** to edit the dashboard properties and set the "Run As" user to **The dashboard viewer**. This ensures each user sees only their own data.



**4. Final Security Review**

* Conduct a final review of all security settings.
* **Object Permissions:** Go to **Setup** → **Permission Sets** and ensure that the Event\_Manager\_Access permission set has the correct CRUD permissions for both the Event and Attendee objects.
* **Field-Level Security (FLS):** Check the **Email** field on the Attendee object to confirm it is **Hidden** from any profiles that do not need to see it.
* **Sharing Settings:** Go to **Setup** → **Sharing Settings** and confirm that the **Organization-Wide Defaults (OWD)** are set to **Private** for the Event object to ensure users only see the records they own.



**5. Documentation & Handoff**

* Finalize all project documentation and prepare for a formal handoff.
* Prepare a **user guide** explaining how to use the application.
* Prepare an **admin guide** that explains the back-end customizations.
* Hold a knowledge transfer session with the long-term system administrator (if applicable) to walk through the project.

### Phase 10: Quality Assurance Testing (QA)

**Goal:** To perform comprehensive User Acceptance Testing (UAT) by executing detailed test cases for all implemented automation, validation, and security features.

#### Test Case 1: Apex Capacity Check (Negative Scenario)

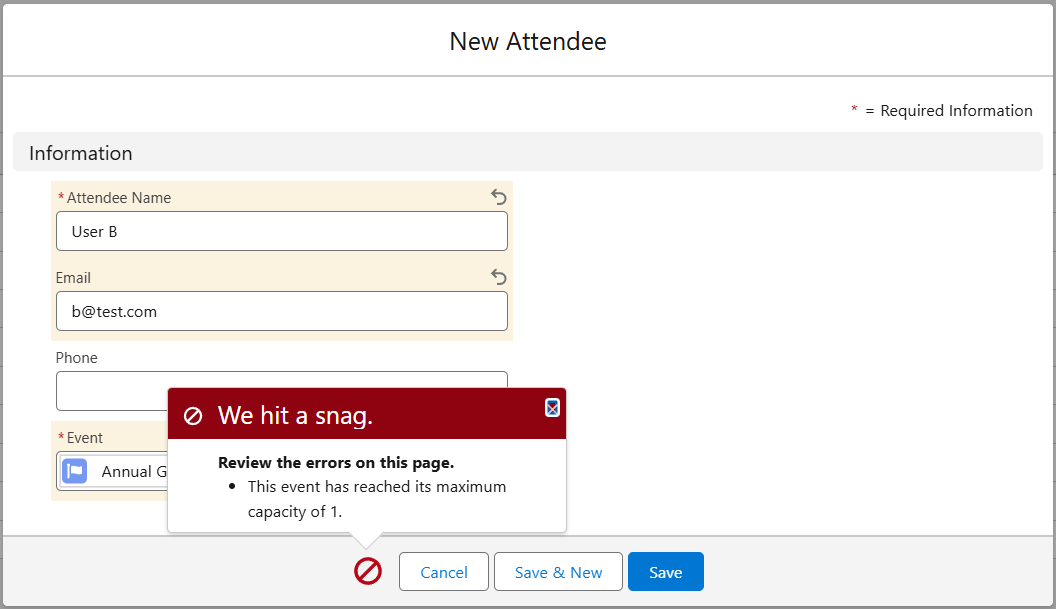
**Purpose:** Verify the Apex Trigger prevents over-registration when the event reaches its maximum capacity.

**Test Steps (Input Details):**

1. Go to the **Events** tab and create a new event: **Name:** Annual Gala, **Max Capacity:** 1.
2. Create the first attendee: **Name:** User A, **Email:** a@test.com, **Event:** Annual Gala. (This fills the capacity).
3. Attempt to create a second attendee: **Name:** User B, **Email:** b@test.com, **Event:** Annual Gala.

**Expected Result:** The system **must block the Save action**. An **Apex Validation Error** message must be displayed on the screen, for example: "Event is at maximum capacity."

**Actual Result :** The actual result matched the expected result. System showed the error for max capacity as expected as user tried to add more attendees than the max capacity.

**

#### Test Case 2: Record-Triggered Flow and Task Creation (Positive Scenario)

**Purpose:** Verify the Flow runs correctly, creates the necessary standard activity records, and sends the confirmation email.

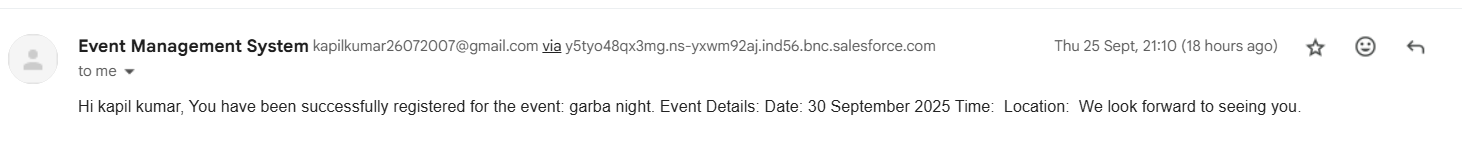
**Test Steps (Input Details):**

1. Create a new event: **Name:** Tech Seminar, **Max Capacity:** 50.
2. Create a new attendee: **Name:** Flow Test, **Email:** USE YOUR EMAIL ADDRESS HERE, **Event:** Tech Seminar.

**Expected Result:**

1. The Attendee record is created and linked.
2. A new standard **Task** is automatically created and linked to the event, appearing on the owner's homepage.
3. An email using the custom HTML template is received at flowtest@org.com.

**Actual Result :** The actual result matched the expected result. Mail was sent successfully by the org wide email address to the attendee mail address.

**

#### Test Case 3: Duplicate Rule Enforcement (Negative Scenario)

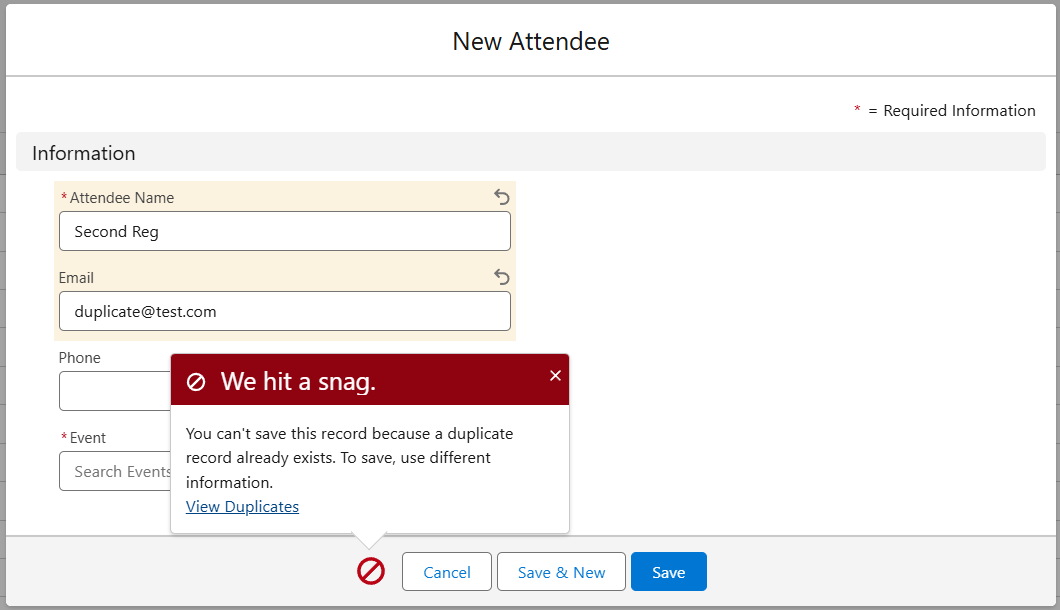
**Purpose:** Verify the Duplicate Rule correctly blocks a user from creating a duplicate attendee based on the email address.

**Test Steps (Input Details):**

1. Create the first attendee: **Name:** First Reg, **Email:** duplicate@test.com.
2. Attempt to create a second attendee: **Name:** Second Reg, **Email:** duplicate@test.com, **Event:** Any Event.

**Expected Result:** The system **must block the Save action**. The user must see a warning or error message indicating the record is a duplicate and is blocked from saving.

**Actual Result:** The actual result matched the expected result. The system found out that there is already a attendee with similar email so it prevented the user to make a new attendee.

**

#### Test Case 4: Validation Rule (Negative Scenario)

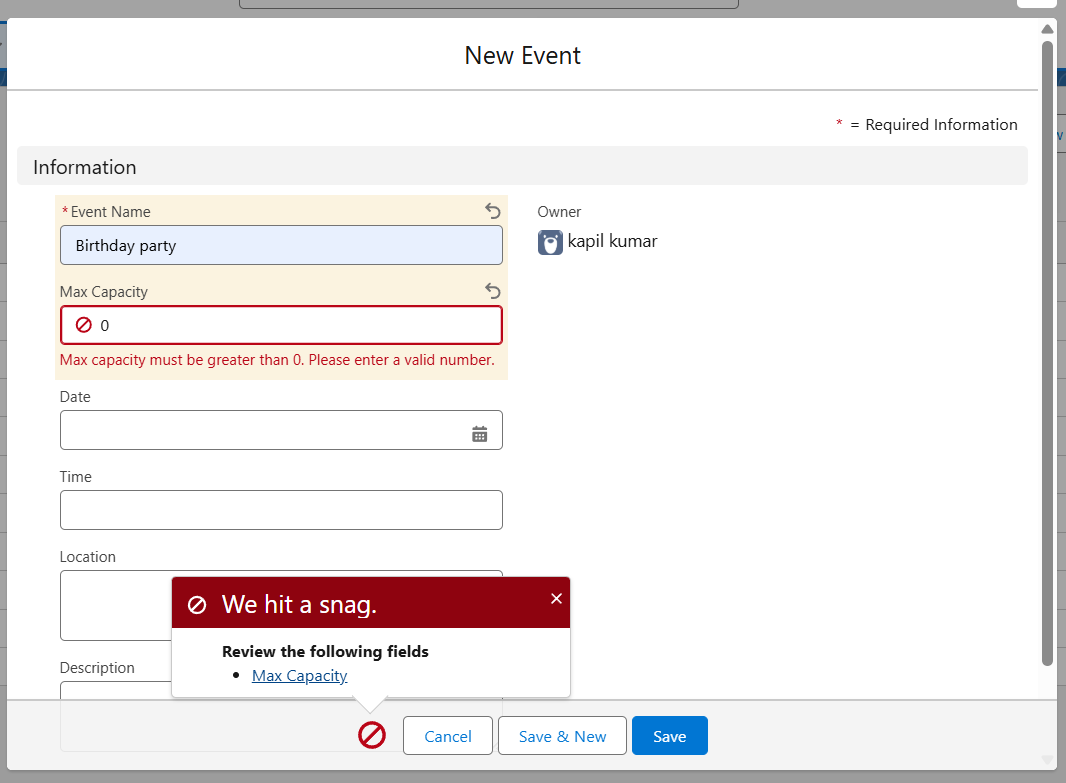
**Purpose:** Verify the Validation Rule prevents the creation of an Event with invalid capacity data.

**Test Steps (Input Details):**

1. Go to the **Events** tab, click New.
2. Attempt to create an event with **Max Capacity** set to **0**.

**Expected Result:** The system must **block the Save action**. A Validation Rule error message must be displayed next to the Max Capacity field, for example: "Max capacity must be greater than 0."

**Actual Result:** The actual result matched the expected result. The system showed the error for the 0 max capacity and stopped the saving action.

**

#### Test Case 5: Field-Level Security (FLS) Review

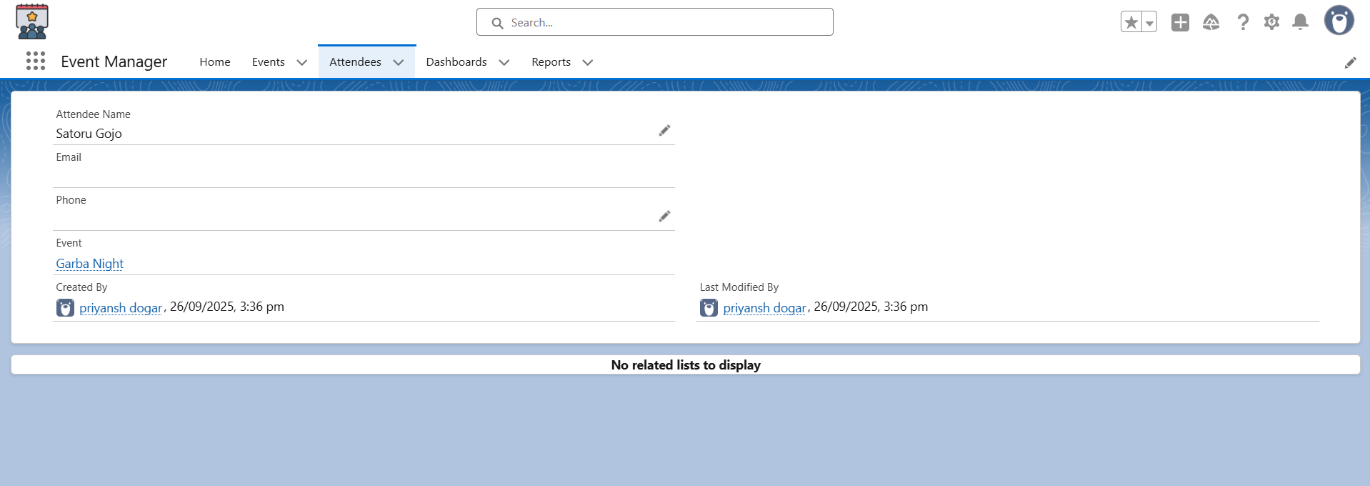
**Purpose:** Verify the sensitive Email field on the Attendee object is hidden from non-admin users.

**Test Steps (Input Details):**

1. Log in as the friend's user account (e.g., the Standard Platform User).
2. Go to an existing **Attendee** record that the user has permission to view.
3. Observe the visibility of the Email field.

**Expected Result:** The **Email** field on the Attendee record page **must be hidden** from the non-admin user.

**Actual Result:** The actual result matched the expected result. The email address is hidden from the standard user on the application as stated in the expected result.

**