PROJECT TITLE

EVENT MANAGEMENT (FIT SYNC)

Submitted by

Neeharika Saha

1. Project Overview

The project is focused on building an efficient event management solution for FIT SYNC, designed to streamline the management of fitness events, attendees, and locations. The solution has been developed using the Salesforce Platform, leveraging its powerful event management capabilities, automation features and robust CRM integration. Key Features of the project are: Objects & Relationships, Apex for backend, Email Automation for confirmation and communication. Through this project, we aim to enhance operational efficiency, improve attendee experience and facilitate effective management of fitness events.

2. Objectives

❖ **Business Goals:**

➔ **Client Management:** Store and track attendees information, preferences and communication history using standard and custom objects.

➔ **Event Management:** An event management software which will help to manage their events very effectively along with the attendees & location information.

➔ **Security Management:** Using Profiles and Roles, ensuring that sensitive data is protected and accessible only to authorised users. Configure Organization-Wide Defaults (OWD) and Sharing Rules to control data visibility and maintain data integrity.

❖ **Specific Outcomes:**

➔ **Automated Confirmation Emails:** Create an automated email alert system that sends confirmation emails to attendees upon successful registration using Apex Triggers.

➔ **Handling Large Data:** Implement Batch Apex processes for handling large volumes of data efficiently, ensuring that system performance remains optimal.

➔ **Error Handling:** Establish an Error Handling Framework to manage exceptions and ensure robust application performance, providing a better user experience and easier troubleshooting.

➔ **Integration:** Create integration capabilities using Apex REST APIs to facilitate data exchange between the Salesforce platform and external systems, enhancing data accessibility.

3. Salesforce Key Features and Concepts Utilized

❖ Salesforce Admin

➔ Objects & Relationships:

1. **Location:** Location__c (API Name) is created for Address Book.

Field	Datatype
Street	Text
City	Text
State	Text
Postal Code	Text
Country	Picklist
Landmark	Text
Verified	Checkbox

2. **Event Organizer:** Event_Organizer__c (API Name) is created for the event organizer.

Field	Datatype
Name	Text (Standard)
Email	Email
Alternative Email	Email
Phone	Phone
Alternative Phone	Phone
Address	Lookup - Location

3. **Event:** Event__c (API Name) is created for the details of the event.

Field	Datatype
Event #	Standard Auto Number
Name	Text
Status (Created, Published, In Progress, Completed, Post Poned, Cancelled)	Picklist
Organizer	Lookup - Event Organizer
Start Date/Time	Date/Time
End Date/Time	Date/Time
Max Seats	Number
# People Attending	Rollup Summary Field
Remaining Seats	Formula field - “Max Seats - # People Attending”
Location	Location - Lookup
Location Verified	Formula field- “Location__r.Verified__c”
Live?	Checkbox
Recurring?	Checkbox
Event Type (In-Person, Virtual)	Picklist
Frequency	Picklist

4. **Attendees:** Attendee__c (API Name) is created for details of Attendee.

Field	Datatype
Name	Text(Standard)
Email	Email
Phone	Phone
Company Name	Text
Address	Lookup - Location

5. **Speaker:** Speaker__c (API Name) is created for details of Speaker.

Field	Datatype
Name	Text(Standard)
Email	Email
Phone	Phone
Company	Text
Picture	URL
About Me	Rich Text Area

6. **Event-Attendee:** Event_Attendee__c (API Name) is created for registration of an attendee to a live event.

Field	Datatype
Event	M-D Event
Attendee	M-D Attendee

7. **Event-Speaker:** EventSpeakers__c (API Name) is created for assigning available speaker to an event.

Field	Datatype
Event	M-D Event
Speaker	M-D Speaker

8. **System Event:** Error_Log__c (API Name) is created to handle the exceptions occurring in the FIT SYNC app.

Field	Datatype
Log Date/Time (Store the Date & Time details when the Error Occurred)	Date/Time
Log Details (Stores the Complete Stack Track about the Problem)	Text Area Long
Process Name (The Name of Either Apex Class or Apex Trigger)	Text
Log #	Auto Generated Number

➔ **Sharing Rules, OWD & Roles:**

Object Permission Set-up: Provide the objects & fields level permission at the Profile level as per the table below.

	Profile		
Object Name	**Event Manager**	**Speaker**	**Attendee**
Event	CRED	R	R
Event - Organizer	CRE	R	R
Speaker	CRE	CRED	R
Attendee	R	X	CRE
Location	CRED	R	CRE
Event - Speaker	CRED	CRE	R
Event - Attendee	CRED	X	CR

[NOTE: **C - Created**, **R - Read**, **E - Edit**, **D - Delete**, **X - No Access**]

Organization Wide Default:

Object Name	Organization Wide Default
Event	Public Read Only
Event - Organizer	Public Read Only
Speaker	Private & Create a Sharing Rule to share the Speakers with Organizers (Role)
Attendee	Private & Create a Sharing Rule to share the Attendee with Organizers (Role)
Location	Public Read Only
Event - Speaker	Public Read Only
Event - Attendee	Public Read Only

Sharing Rule Setup: Share every Speaker & Attendee record with Organizer Role.

Speaker Object - Create a Sharing Rule which will share the Speaker records with the Role Organizer. And the permission should be Read/Edit.

Attendee Object - Create a Sharing Rule which will share the Attendee records with the Role Organizer. And the permission should be Read/Edit.

❖ Salesforce Development

➔ Apex Trigger

Trigger Development on Event - Speaker Object:

Develop a Trigger on Event - Speaker object which will throw an error if the Speaker Selected on Event - Speaker Record already has an Event against his name. i.e - For a Speaker there will be only one event at a time. Reject Duplicate Bookings.

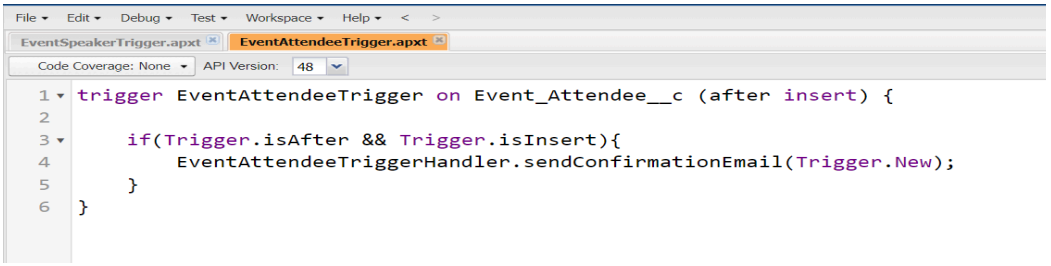
EventSpeakerTrigger.apxt

Code Coverage: None API Version: 48

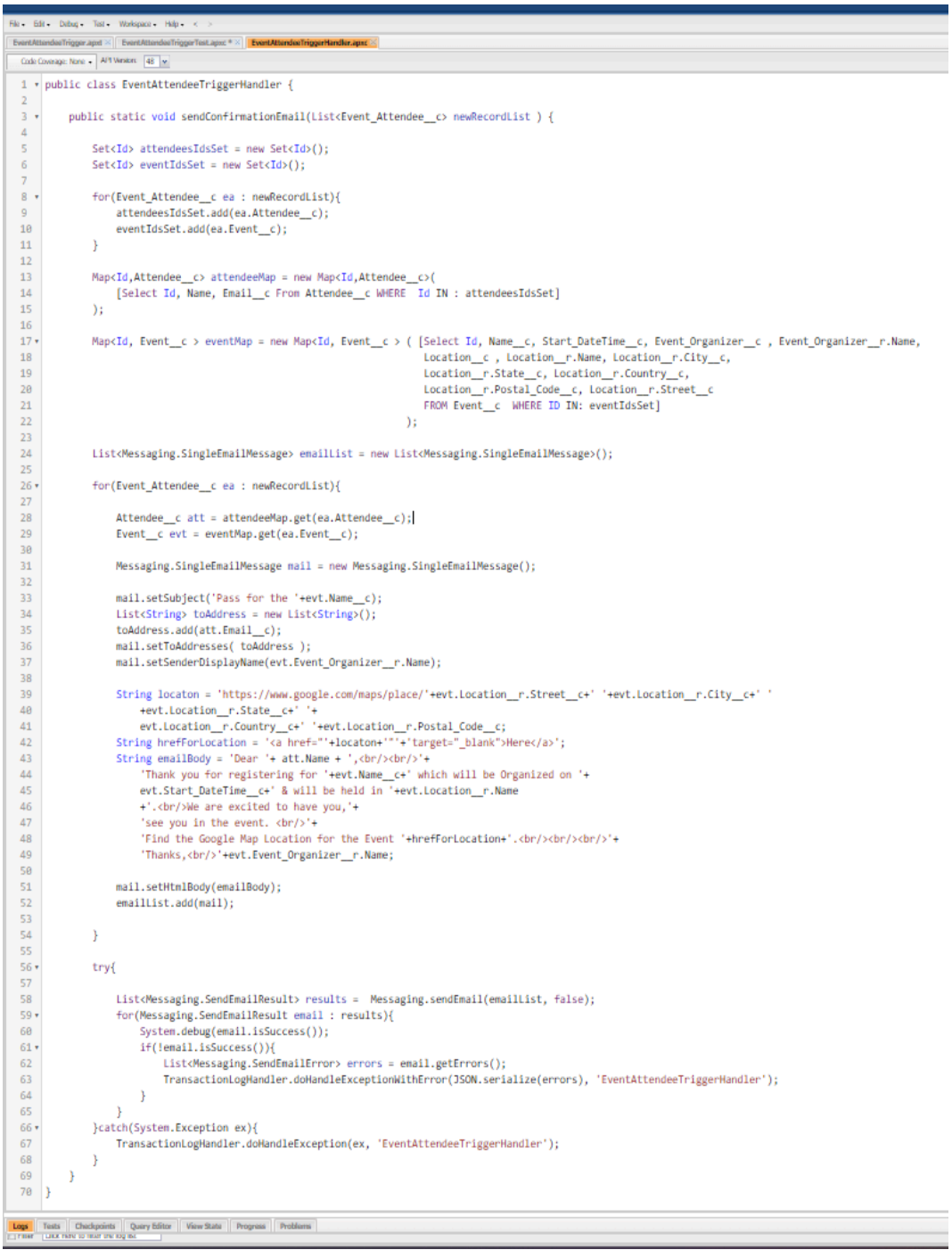
```
trigger EventSpeakerTrigger on EventSpeakers__c (before insert, before update ) {

    // Step 1 - Get the speaker id & event id
    // Step 2 - SOQL on Event to get the Start Date and Put them into a Map
    // Step 3 - SOQL on Event - Spekaer to get the Related Speaker along with the Event Start Date
    // Step 4 - Check the Conditions and throw the Error

    //Step 1 -  Start

    Set<Id> speakerIdsSet = new Set<Id>();
    Set<Id> eventIdsSet = new Set<Id>();

    for( EventSpeakers__c es : Trigger.New ){
         speakerIdsSet.add(es.Speaker__c);
         eventIdsSet.add(es.Event__c);
    }
    //Step 1 -  End

    // Step 2 Start
    Map<Id, DateTime> requestedEvents = new Map<Id, DateTime>();

    List<Event__c> relatedEventList = [Select Id, Start_DateTime__c From Event__c
                                       Where Id IN : eventIdsSet];

    for(Event__c evt : relatedEventList ){
        requestedEvents.put(evt.Id, evt.Start_DateTime__c);
    }
    // Step 2 End


    // Step 3 - Start
    List<EventSpeakers__c> relatedEventSpeakerList = [ SELECT Id, Event__c, Speaker__c,
                                                       Event__r.Start_DateTime__c
                                                       From EventSpeakers__c
                                                       WHERE Speaker__c IN : speakerIdsSet];

    // Step 3 - End

    // Step 4 - Start
    for( EventSpeakers__c es : Trigger.New ){ // - Salesforce Geek

        DateTime bookingTime = requestedEvents.get(es.Event__c);
        // DateTime for that event which is associated with this new Event-Speaker Record

        for(EventSpeakers__c es1 : relatedEventSpeakerList) {
            if(es1.Speaker__c == es.Speaker__c && es1.Event__r.Start_DateTime__c == bookingTime ){
                es.Speaker__c.addError('The speaker is already booked at that time');
                es.addError('The speaker is already booked at that time');
            }
        }

    }
    // Step 4 - End
}
```

Trigger Development on Event - Attendee Object:

Whenever a New Record gets created send email to Attendee saying that registration has been confirmed . In the email include the basic details of the attendee like name, email, phone.



```
File  Edit  Debug  Test  Workspace  Help  <  >
EventSpeakerTrigger.apxt   EventAttendeeTrigger.apxt
Code Coverage: None   API Version:  48
1  trigger EventAttendeeTrigger on Event_Attendee__c (after insert) {
2
3      if(Trigger.isAfter && Trigger.isInsert){
4          EventAttendeeTriggerHandler.sendConfirmationEmail(Trigger.New);
5      }
6  }
```



```
1  public class EventAttendeeTriggerHandler {
2
3      public static void sendConfirmationEmail(List<Event_Attendee__c> newRecordList ) {
4
5          Set<Id> attendeesIdsSet = new Set<Id>();
6          Set<Id> eventIdsSet = new Set<Id>();
7
8          for(Event_Attendee__c ea : newRecordList){
9              attendeesIdsSet.add(ea.Attendee__c);
10             eventIdsSet.add(ea.Event__c);
11         }
12
13         Map<Id,Attendee__c> attendeeMap = new Map<Id,Attendee__c>(
14             [Select Id, Name, Email__c From Attendee__c WHERE  Id IN : attendeesIdsSet]
15         );
16
17         Map<Id, Event__c > eventMap = new Map<Id, Event__c > ( [Select Id, Name__c, Start_DateTime__c, Event_Organizer__c , Event_Organizer__r.Name,
18                                                                 Location__c , Location__r.Name, Location__r.City__c,
19                                                                 Location__r.State__c, Location__r.Country__c,
20                                                                 Location__r.Postal_Code__c, Location__r.Street__c
21                                                                 FROM Event__c  WHERE ID IN: eventIdsSet]
22                                                               );
23
24         List<Messaging.SingleEmailMessage> emailList = new List<Messaging.SingleEmailMessage>();
25
26         for(Event_Attendee__c ea : newRecordList){
27
28             Attendee__c att = attendeeMap.get(ea.Attendee__c);
29             Event__c evt = eventMap.get(ea.Event__c);
30
31             Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();
32
33             mail.setSubject('Pass for the '+evt.Name__c);
34             List<String> toAddress = new List<String>();
35             toAddress.add(att.Email__c);
36             mail.setToAddresses( toAddress );
37             mail.setSenderDisplayName(evt.Event_Organizer__r.Name);
38
39             String locaton = 'https://www.google.com/maps/place/'+evt.Location__r.Street__c+' '+evt.Location__r.City__c+' '
40                 +evt.Location__r.State__c+' '+
41                 evt.Location__r.Country__c+' '+evt.Location__r.Postal_Code__c;
42             String hrefForLocation = '<a href="'+locaton+'"'+'target="_blank">Here</a>';
43             String emailBody = 'Dear '+ att.Name + ',<br/><br/>'+
44                 'Thank you for registering for '+evt.Name__c+' which will be Organized on '+
45                 evt.Start_DateTime__c+' & will be held in '+evt.Location__r.Name
46                 +'.<br/>We are excited to have you,'+
47                 'see you in the event. <br/>'+
48                 'Find the Google Map Location for the Event '+hrefForLocation+'.<br/><br/><br/>'+
49                 'Thanks,<br/>'+evt.Event_Organizer__r.Name;
50
51             mail.setHtmlBody(emailBody);
52             emailList.add(mail);
53
54         }
55
56         try{
57
58             List<Messaging.SendEmailResult> results =  Messaging.sendEmail(emailList, false);
59             for(Messaging.SendEmailResult email : results){
60                 System.debug(email.isSuccess());
61                 if(!email.isSuccess()){
62                     List<Messaging.SendEmailError> errors = email.getErrors();
63                     TransactionLogHandler.doHandleExceptionWithError(JSON.serialize(errors), 'EventAttendeeTriggerHandler');
64                 }
65             }
66         }catch(System.Exception ex){
67             TransactionLogHandler.doHandleException(ex, 'EventAttendeeTriggerHandler');
68         }
69     }
70 }
```

➔ Batch Apex

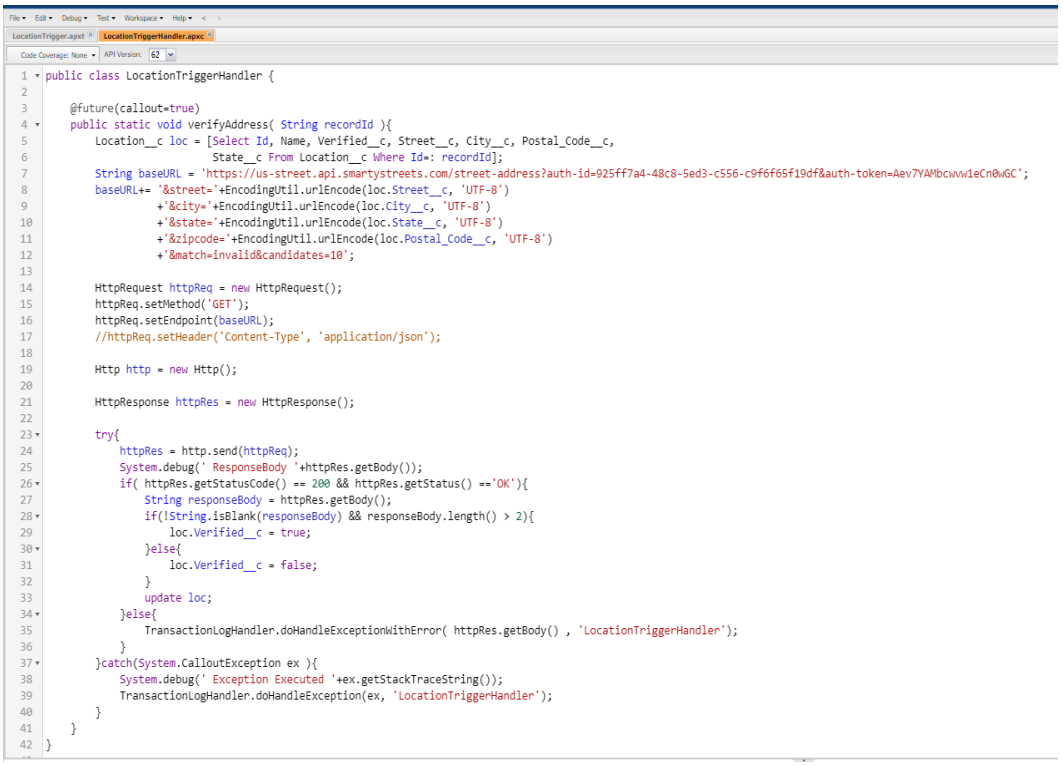
An Apex Batch which should delete all the event records which are more than 2 months old & have been Organized.

```apex
// Return delete all those records older than 2 months
public class DeleteEventBatch implements Database.Batchable<sObject> {

    public Database.QueryLocator start(Database.BatchableContext batchContext){
        String Query = 'Select Id, Name From Event__c Where Live__c = true AND End_Date_Time__c < LAST_N_MONTHS:2';
        return Database.getQueryLocator(Query);
    }

    public void execute(Database.BatchableContext batchContext, List<Event__c> eventList ){
        if(!eventList.isEmpty()){
            List<Database.DeleteResult> deleteResult = Database.delete(eventList);
            for(Database.DeleteResult dr : deleteResult){
                if(!dr.isSuccess()){
                    List<Database.Error> errors = dr.getErrors();
                    String errorInString = String.join(errors, ',');
                    TransactionLogHandler.doHandleExceptionWithError(errorInString, 'DeleteEventBatch');
                }
            }
        }
    }

    public void finish(Database.BatchableContext batchContext){
        Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();

        mail.setSubject('Batch Processing Completed');
        List<String> toAddress = new List<String>();
        toAddress.add('neeharika.saha.21@aot.edu.in');
        mail.setToAddresses( toAddress );
        mail.setSenderDisplayName('NSAHA');

        mail.setHtmlBody('The DeleteEventBatch Batch Prcoessing has been Completed');
        List<Messaging.SingleEmailMessage> emails = new List<Messaging.SingleEmailMessage> {mail};
        Messaging.sendEmail(emails);
    }
}
```

➔ Future Method

Trigger Development to verify the address of the Location Object. Use the **SmartyStreets API** to verify the address & update "Location Verified" field on Location Object.

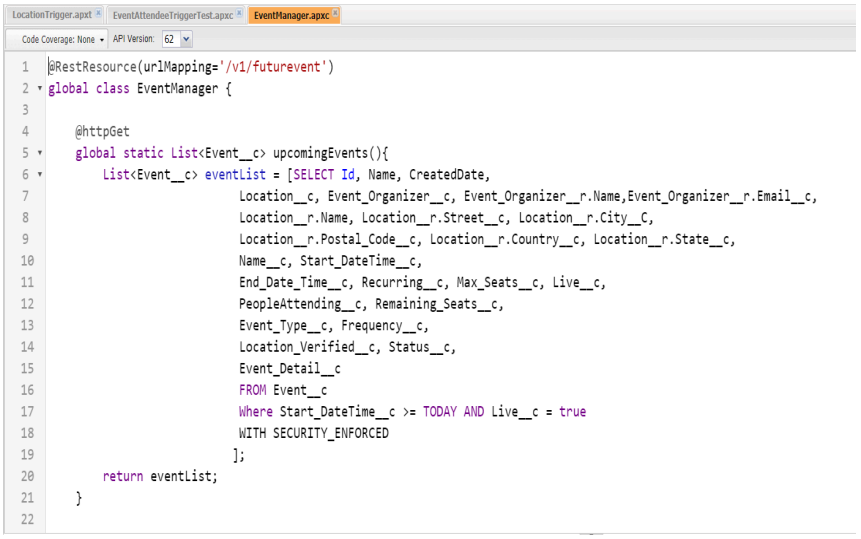
```apex
trigger LocationTrigger on Location__c (after insert) {

    if(Trigger.isAfter && Trigger.isInsert ){
        for( Location__c loc : Trigger.New ){
            LocationTriggerHandler.verifyAddress(loc.Id);
        }
    }

}
```



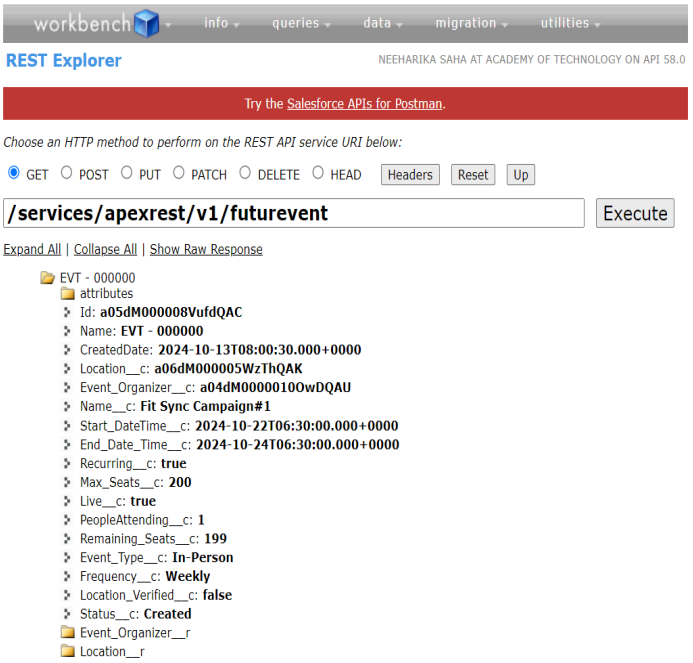
```apex
public class LocationTriggerHandler {

    @future(callout=true)
    public static void verifyAddress( String recordId ){
        Location__c loc = [Select Id, Name, Verified__c, Street__c, City__c, Postal_Code__c,
                           State__c From Location__c Where Id=: recordId];
        String baseURL = 'https://us-street.api.smartystreets.com/street-address?auth-id=925ff7a4-48c8-5ed3-c556-c9f6f65f19df&auth-token=Aev7YAMbcwvw1eCn0wGC';
        baseURL+= '&street='+EncodingUtil.urlEncode(loc.Street__c, 'UTF-8')
                  +'&city='+EncodingUtil.urlEncode(loc.City__c, 'UTF-8')
                  +'&state='+EncodingUtil.urlEncode(loc.State__c, 'UTF-8')
                  +'&zipcode='+EncodingUtil.urlEncode(loc.Postal_Code__c, 'UTF-8')
                  +'&match=invalid&candidates=10';

        HttpRequest httpReq = new HttpRequest();
        httpReq.setMethod('GET');
        httpReq.setEndpoint(baseURL);
        //httpReq.setHeader('Content-Type', 'application/json');

        Http http = new Http();

        HttpResponse httpRes = new HttpResponse();

        try{
            httpRes = http.send(httpReq);
            System.debug(' ResponseBody '+httpRes.getBody());
            if( httpRes.getStatusCode() == 200 && httpRes.getStatus() =='OK'){
                String responseBody = httpRes.getBody();
                if(!String.isBlank(responseBody) && responseBody.length() > 2){
                    loc.Verified__c = true;
                }else{
                    loc.Verified__c = false;
                }
                update loc;
            }else{
                TransactionLogHandler.doHandleExceptionWithError( httpRes.getBody() , 'LocationTriggerHandler');
            }
        }catch(System.CalloutException ex ){
            System.debug(' Exception Executed '+ex.getStackTraceString());
            TransactionLogHandler.doHandleException(ex, 'LocationTriggerHandler');
        }
    }
}
```

➔ Integration (Both Apex REST & REST API)

Develop an Apex Rest to send the Event Details to 3rd parties in JSON format.

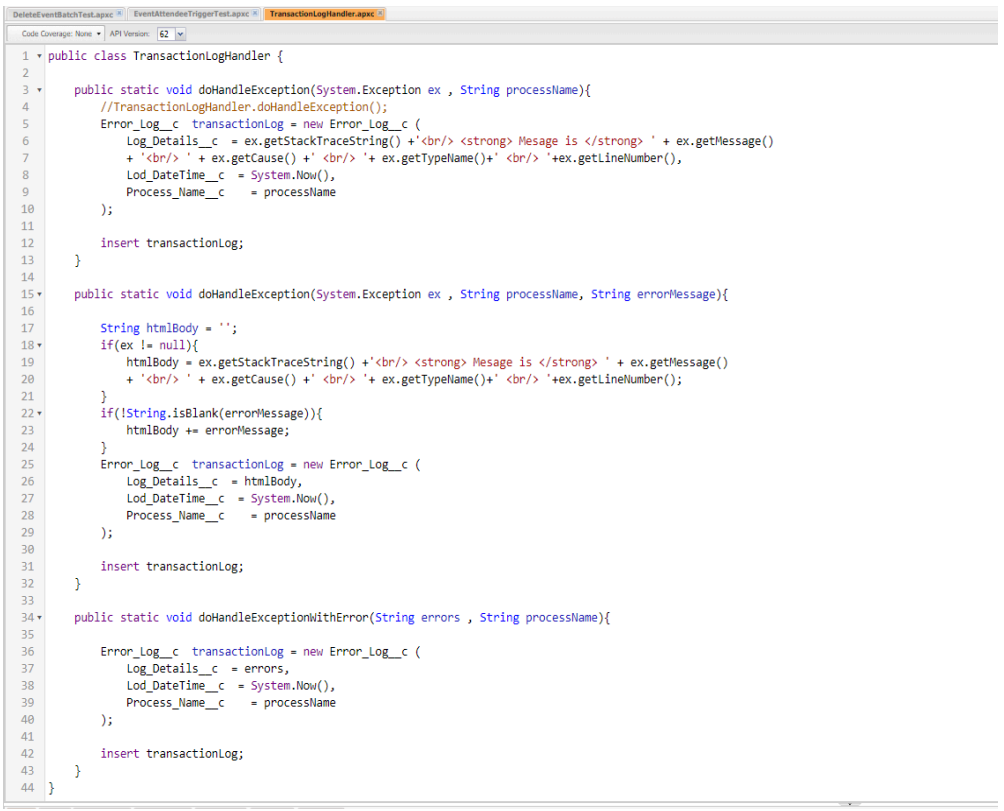


```apex
@RestResource(urlMapping='/v1/futurevent')
global class EventManager {

    @httpGet
    global static List<Event__c> upcomingEvents(){
        List<Event__c> eventList = [SELECT Id, Name, CreatedDate,
                            Location__c, Event_Organizer__c, Event_Organizer__r.Name,Event_Organizer__r.Email__c,
                            Location__r.Name, Location__r.Street__c, Location__r.City__C,
                            Location__r.Postal_Code__c, Location__r.Country__c, Location__r.State__c,
                            Name__c, Start_DateTime__c,
                            End_Date_Time__c, Recurring__c, Max_Seats__c, Live__c,
                            PeopleAttending__c, Remaining_Seats__c,
                            Event_Type__c, Frequency__c,
                            Location_Verified__c, Status__c,
                            Event_Detail__c
                            FROM Event__c
                            Where Start_DateTime__c >= TODAY AND Live__c = true
                            WITH SECURITY_ENFORCED
                           ];
        return eventList;
    }

```



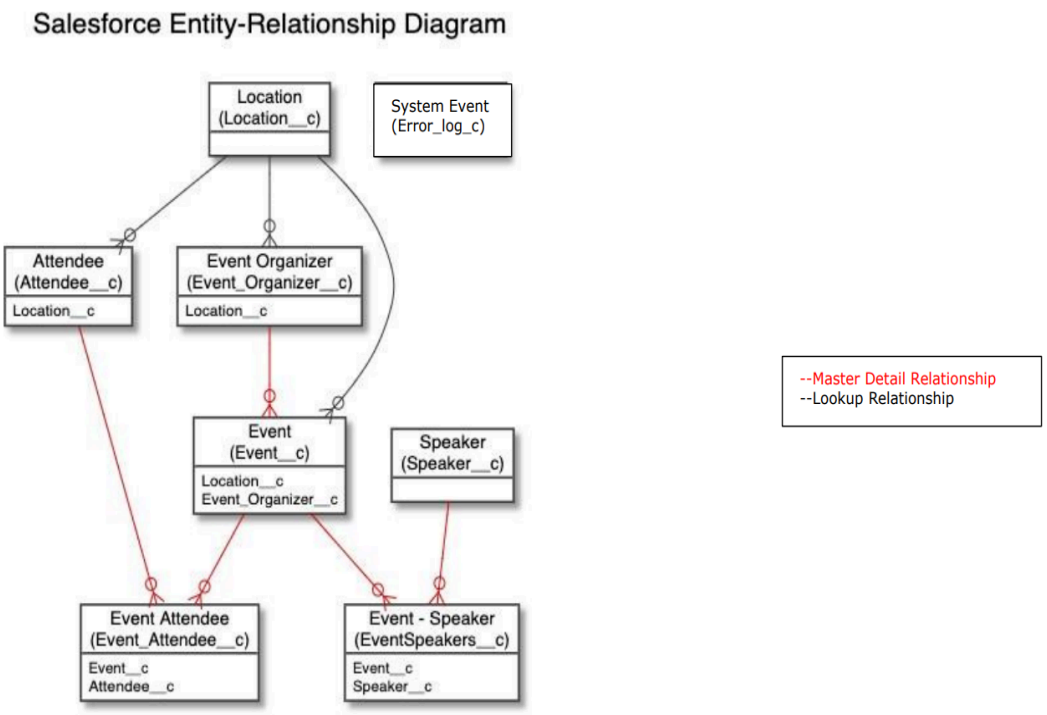
❖ Reusable Error Handling Framework

Developed a reusable Apex Class which contains a method to insert the System Event (Error_Log__c) Object records. This method must contain the parameters to get the dynamic details of the fields (Log Date/Tile, Log Details & Process Name).



```apex
public class TransactionLogHandler {

    public static void doHandleException(System.Exception ex , String processName){
        //TransactionLogHandler.doHandleException();
        Error_Log__c  transactionLog = new Error_Log__c (
            Log_Details__c  = ex.getStackTraceString() +'<br/> <strong> Mesage is </strong> ' + ex.getMessage()
            + '<br/> ' + ex.getCause() +' <br/> '+ ex.getTypeName()+' <br/> '+ex.getLineNumber(),
            Lod_DateTime__c  = System.Now(),
            Process_Name__c    = processName
        );

        insert transactionLog;
    }

    public static void doHandleException(System.Exception ex , String processName, String errorMessage){

        String htmlBody = '';
        if(ex != null){
            htmlBody = ex.getStackTraceString() +'<br/> <strong> Mesage is </strong> ' + ex.getMessage()
            + '<br/> ' + ex.getCause() +' <br/> '+ ex.getTypeName()+' <br/> '+ex.getLineNumber();
        }
        if(!String.isBlank(errorMessage)){
            htmlBody += errorMessage;
        }
        Error_Log__c  transactionLog = new Error_Log__c (
            Log_Details__c  = htmlBody,
            Lod_DateTime__c  = System.Now(),
            Process_Name__c    = processName
        );

        insert transactionLog;
    }

    public static void doHandleExceptionWithError(String errors , String processName){

        Error_Log__c  transactionLog = new Error_Log__c (
            Log_Details__c  = errors,
            Lod_DateTime__c  = System.Now(),
            Process_Name__c    = processName
        );

        insert transactionLog;
    }
}
```

4. Detailed Steps to Solution Design



1. Requirements Gathering:

- ❖ Identify the key business goals, including client management, event management, and security management.
- ❖ Define the specific outcomes, such as automated confirmation emails, handling large data sets, error handling, and integration.

2. Data Modeling and Relationships:

- ❖ **Custom Objects Creation:**
  - ➢ Create objects like Event__c, Attendee__c, Event_Organizer__c, Speaker__c, and Location__c with their respective fields.
  - ➢ Define Master-Detail relationships between Event_Attendee__c (for event registration) and Event_Speaker__c (for speaker assignments).

- **Field Definitions:**
  - Define custom fields such as Status, Max Seats, # People Attending, Location Verified, etc., using appropriate data types like Picklist, Formula, Lookup, Rollup Summary, etc.
- **Formula and Rollup Fields:**
  - Implement formulas like "Max Seats - # People Attending" for remaining seat calculation and Location__r.Verified__c for location verification status.

3. Security Design:

- **Object-Level Permissions:**
  - Configure Profiles and Roles to manage CRUD permissions for different user types (Event Manager, Speaker, Attendee).
- **Organization-Wide Defaults (OWD):**
  - Set OWDs for objects like Event, Attendee, Location, Speaker to control public or private access.
  - Create Sharing Rules for objects like and Attendee to share records with the Organizer role.
- **Field-Level Security:**
  - Ensure that sensitive fields (like email addresses and phone numbers) are restricted based on user roles.

4. Apex Development:

- **Apex Triggers:**
  - Develop a trigger on Event_Attendee__c to automatically send confirmation emails with attendee details.
  - Create a trigger on Event_Speaker__c to prevent duplicate speaker bookings by checking if a speaker is already assigned to an event.
- **Batch Apex:**
  - Implement a batch job that deletes event records that are older than two months and marked as organized.
- **Future Methods:**
  - Create a future method to verify the address using SmartyStreets API and update the Location Verified field.

5. Email Automation:

- **Trigger-Driven Emails:**
  - Set up an email alert system using Apex triggers to send confirmation emails upon successful registration.
  - Leverage dynamic email templates to include event details, attendee information, and Google Maps links.

6. Integration:

- **Apex REST API:**
  - Develop an Apex REST API to send event details in JSON format to third-party systems.
- **RESTful Integration:**
  - Ensure secure data exchange using RESTful web services between Salesforce and external systems for better accessibility and real-time updates.

7. Error Handling:

- **Error Log Object:**
  - Create an Error_Log__c object to store error details, including log date/time, stack trace, and process name.
- **Reusable Error Handling Framework:**
  - Develop a reusable Apex class that inserts error records dynamically, helping capture and log exceptions for better troubleshooting.

8. Performance Optimization:

- **Batch Processing:**
  - Optimize data handling with Batch Apex to process large datasets efficiently, especially for deleting outdated event records.

9. Testing:

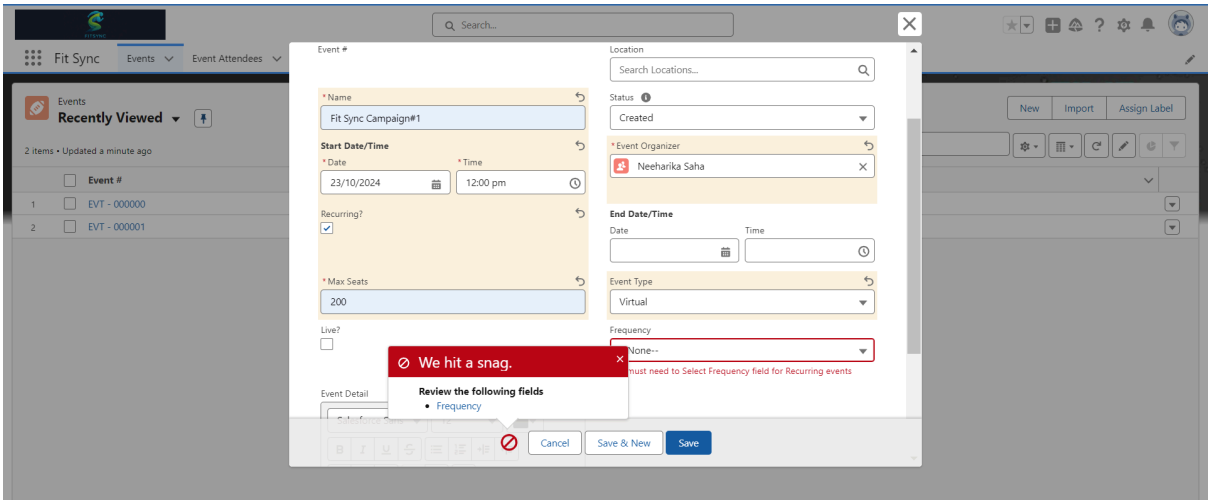
- **Unit Testing:**
  - Create test classes for all Apex triggers, future methods, and batch processes to ensure code quality.

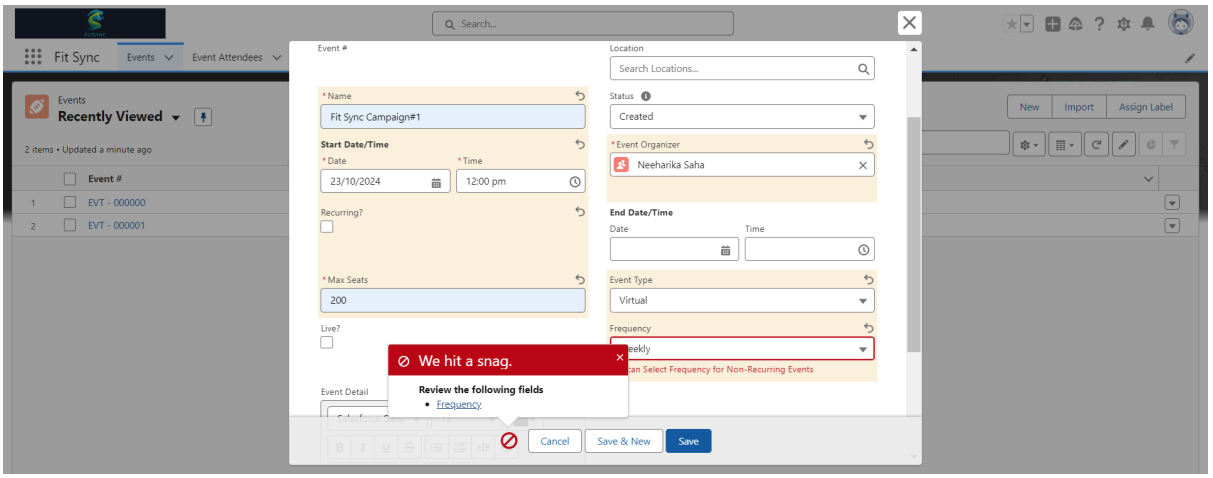
5. Testing and Validation

Validation Rule Setup:

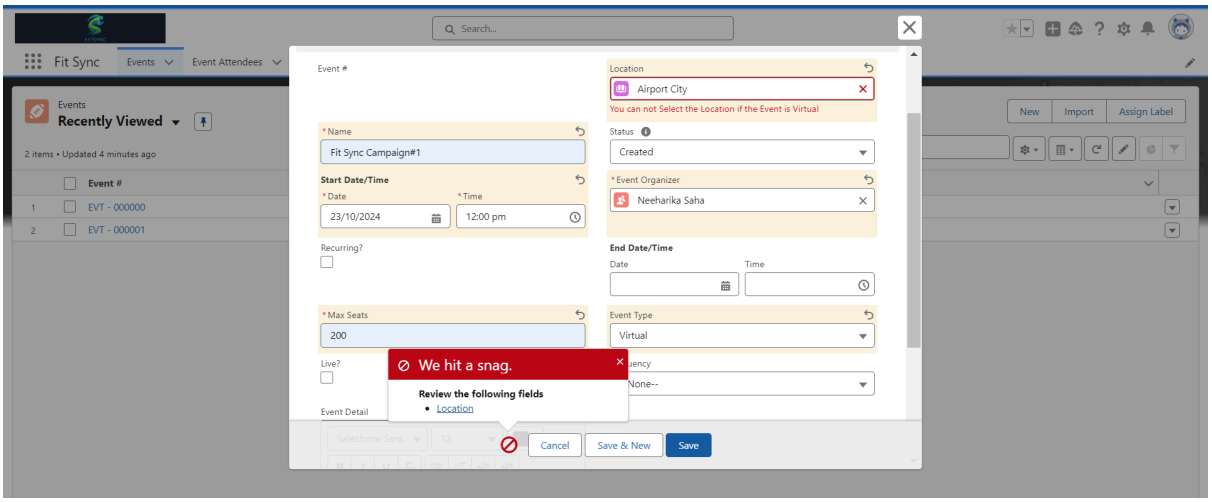
1. Validation Rule on Event Object

1.1 - If Recurring? checkbox is checked then user must need to fill Frequency field & If checkbox is unchecked then User can not select Frequency field.

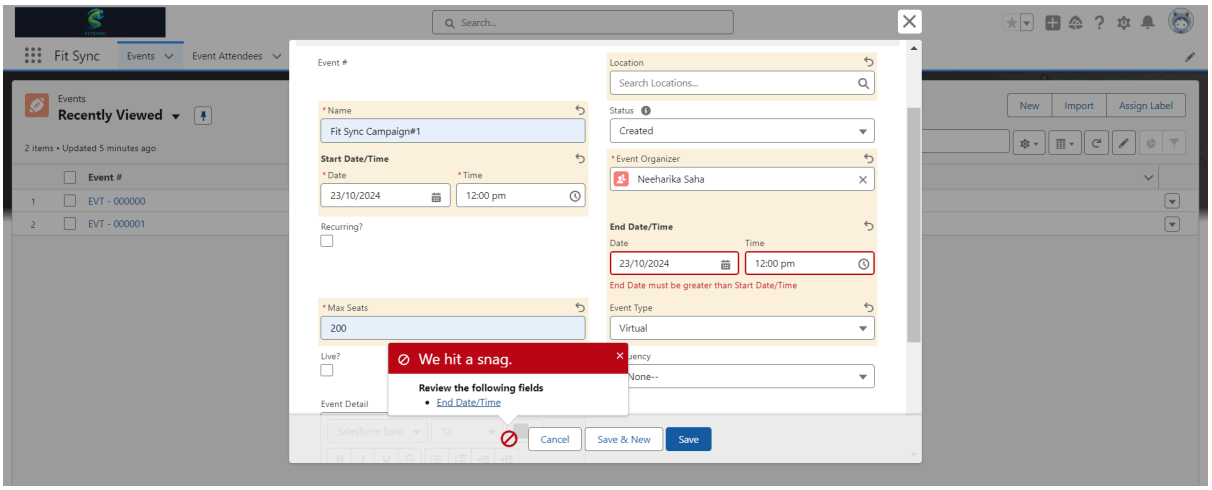




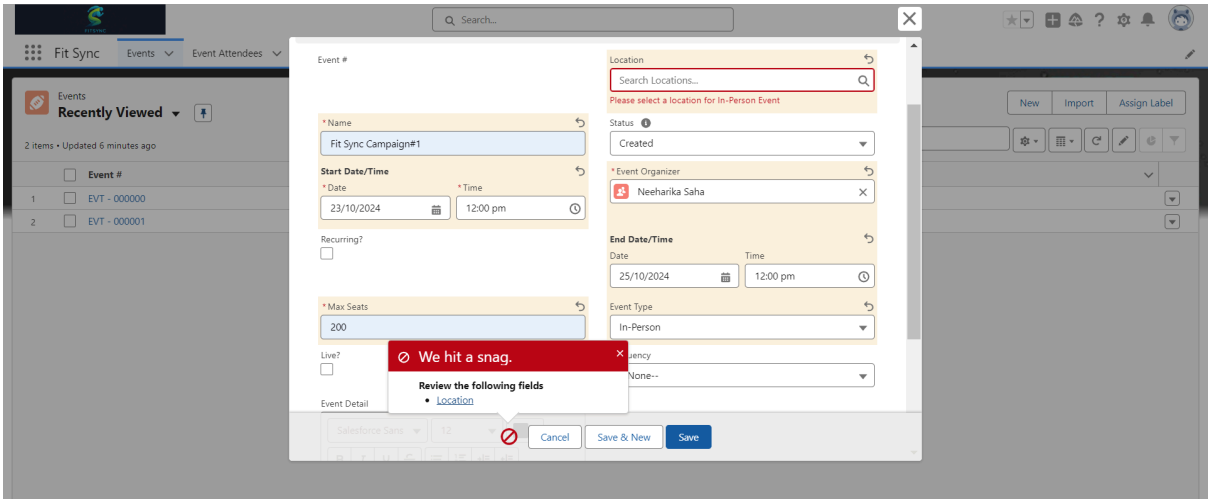
1.2 - If Virtual is Selected as Value for Event Type field then Prevent User to Select Location on Event Record.



1.3 - End Date/Time must be at-least 1 day ahead of Start Date/Time (If there is a value in End Date/Time field)

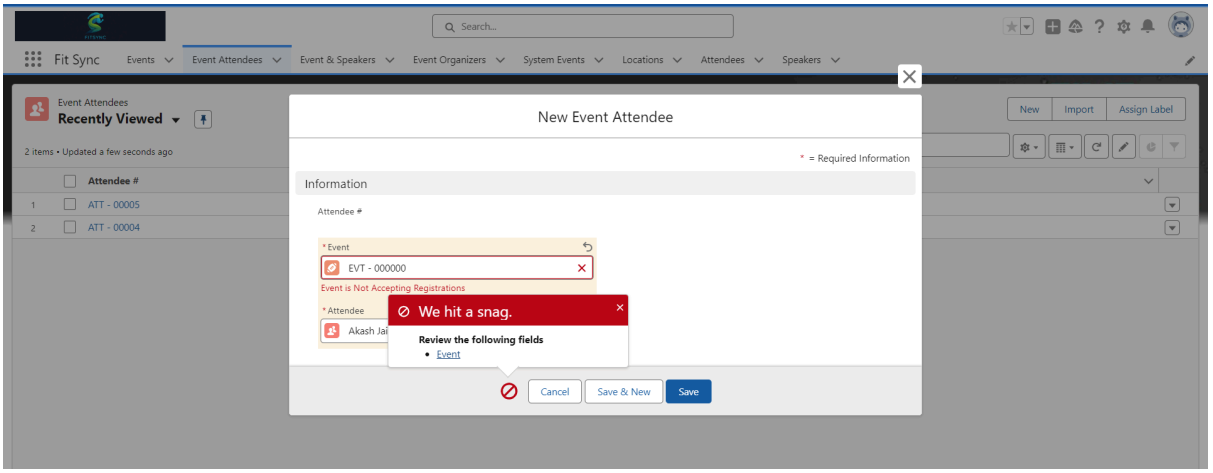


1.4 - If the Event Type field value is In-Person then the user must need to select Location on Event Record.



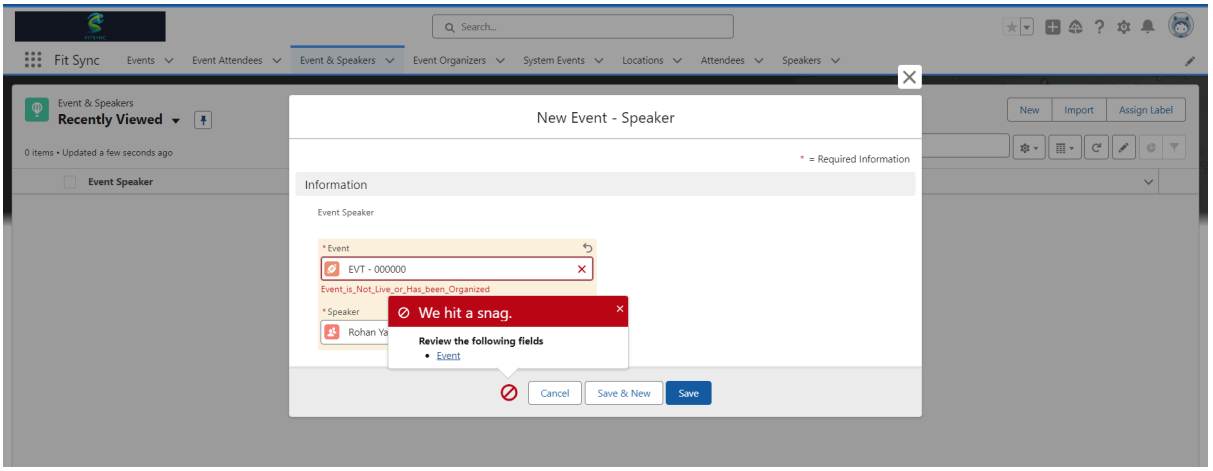
2. Event Attendee Object

2.1 - Attendee can only be associated with the Event whose End Date is in future & Event Live Checkbox is checked and Event is accepting the Attendees (means Remaining Seats field value is not 0).



3. Event Speaker Object

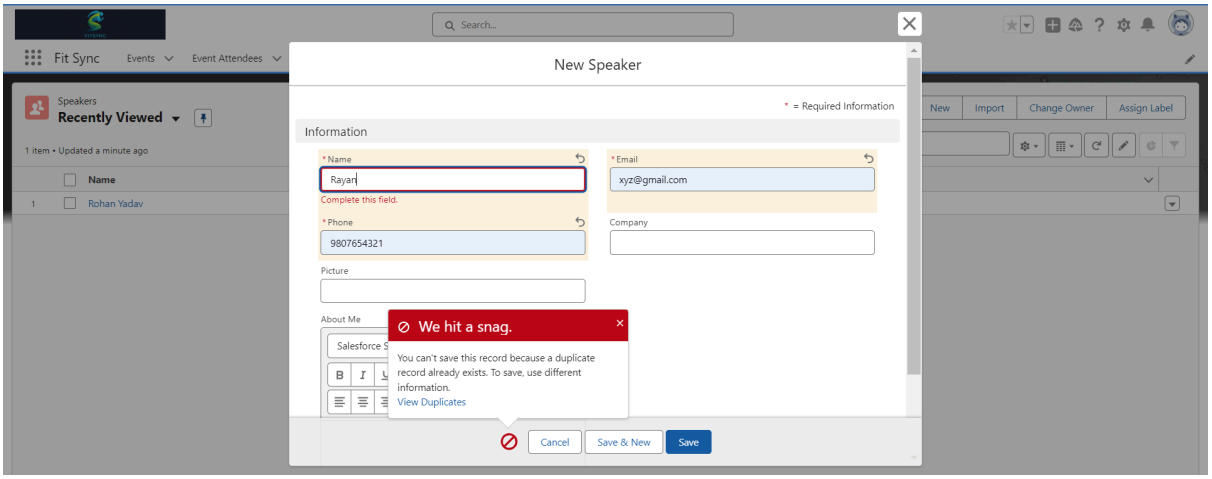
3.1 - Speaker can only be associated with the Event whose End Date is in future & Event Live Checkbox is checked



Duplicate Rule Setup:

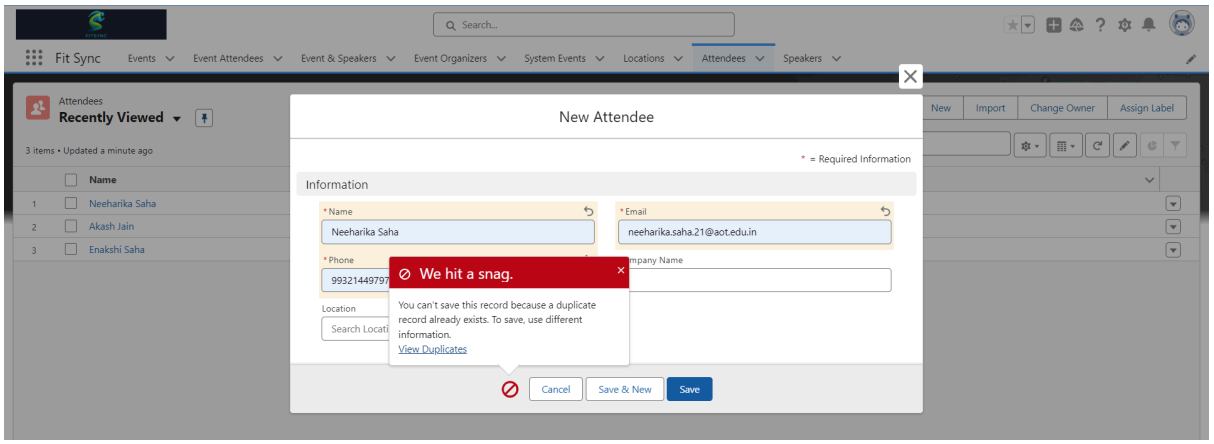
1. Speaker Object

1.1 - User can not create duplicate speaker record with the same Email & Phone



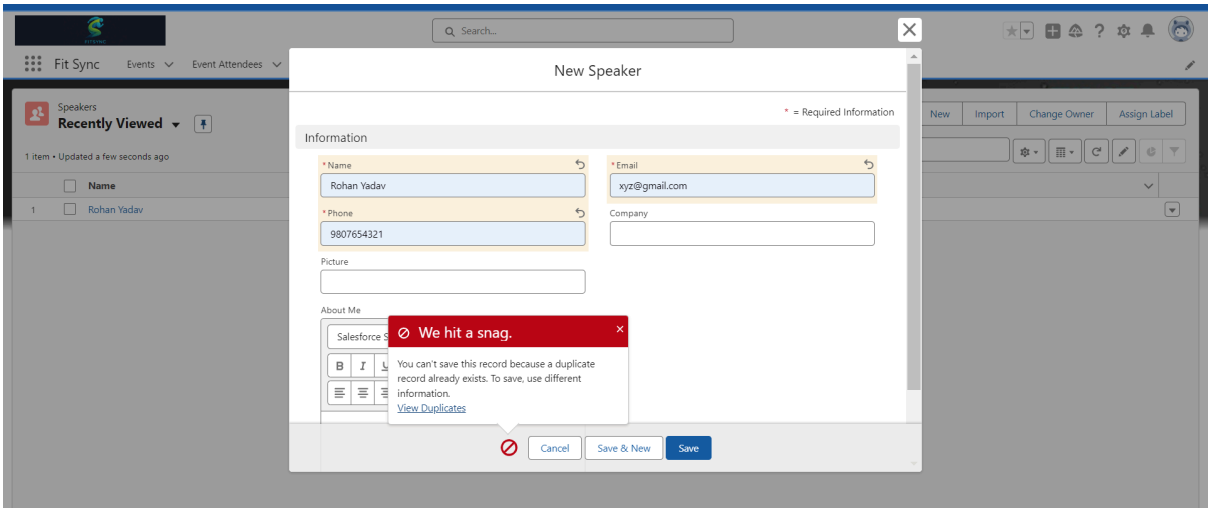
2. Attendee Object

2.1 - User can not Create duplicate attendee with the Same Name, Email & Phone



3. Event Organizer Object

3.1 - Apply the same rule as the Speaker object for not having a duplicate Organizer in the System.



Develop Unit Test Cases:

Develop the Unit Test for the above Apex Trigger(s) & The code coverage must be at-least 85% for Trigger & Handler/Helper class.

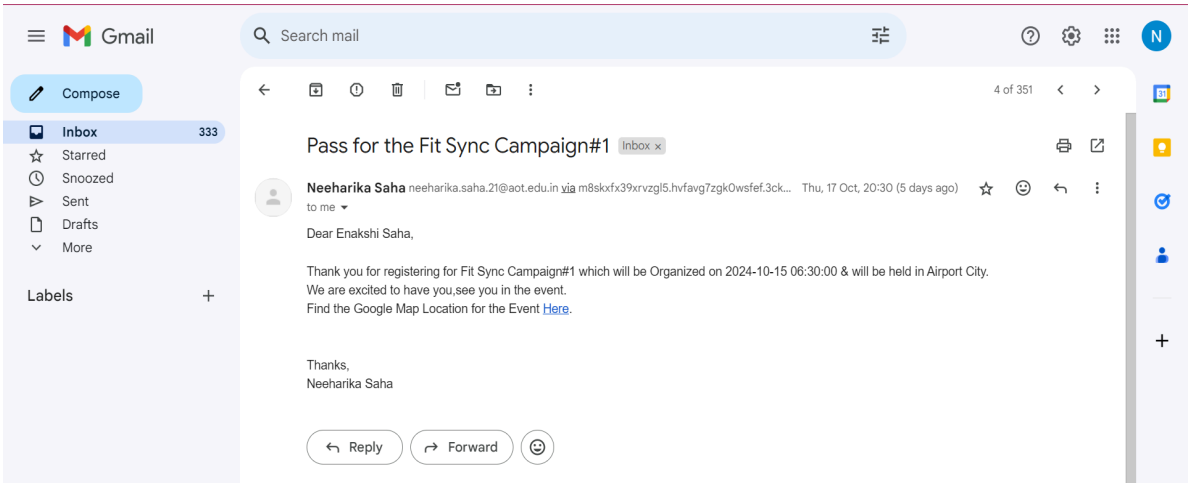
```apex
@isTest
public class EventAttendeeTriggerTest {

    @testSetup
    public static void setupData(){

        Event_Organizer__c  org = new Event_Organizer__c (
            Name = 'Neeharika Saha',
            Phone__c = '9932144979',
            Email__c = 'neeharika.saha.21@aot.edu.in'
        );
        insert org;

        Event__c event = new Event__c(
            Name__c = 'Fit Sync Campaign#1',
            Event_Organizer__c  = org.Id,
            Event_Type__c = 'In-Person',
            Frequency__c  = 'Weekly',
            Max_Seats__c  = 200,
            Recurring__c = true,
            Live__c = true,
            Start_DateTime__c = System.now()
            //End_Date_Time__c  = System.now().addDays(3)
        );
        insert event;
        Attendee__c att = new Attendee__c(
            Name = 'Enakshi Saha',
            Email__c = 'neeharika.javascript.11@gmail.com',
            Phone__c = '9436076423'
        );
        insert att;

        Event_Attendee__c evtAtt = new Event_Attendee__c(Event__c = event.Id, Attendee__c = att.Id);
        insert evtAtt;
    }

    @isTest
    static void sendEmailTest(){
        Test.startTest();
        try{
            integer i = 10/0;
        }catch(System.Exception ex ){

        }
        Test.stopTest();
    }
}
```

```apex
@isTest
public class DeleteEventBatchTest {

    @testSetup
    public static void testSetupData(){
        Event_Organizer__c  org = new Event_Organizer__c (
            Name = 'Neeharika Saha',
            Phone__c = '9932144979',
            Email__c = 'neeharika.saha.21@aot.edu.in'
        );
        insert org;
        List<Event__c> eventList = new List<Event__c>();
        for(Integer i=0; i<=200; i++){
            Event__c event = new Event__c(
                Name__c = 'Fit Sync Campaign#1'+i+1,
                Event_Organizer__c  = org.Id,
                Event_Type__c = 'Virtual',
                Frequency__c  = 'Weekly',
                Max_Seats__c  = 199,
                Recurring__c = true,
                Live__c = true,
                Start_DateTime__c = System.now().addMonths(-4),
                End_Date_Time__c  = System.now().addDays(3).addMonths(-4)
            );
            eventList.add(event);
        }
        insert eventList;
    }

    @isTest
    static void sendDeletEventTest(){
        Test.startTest();

        String jobId = Database.executeBatch(new DeleteEventBatch(), 250);

        Test.stopTest();

    }
}
```

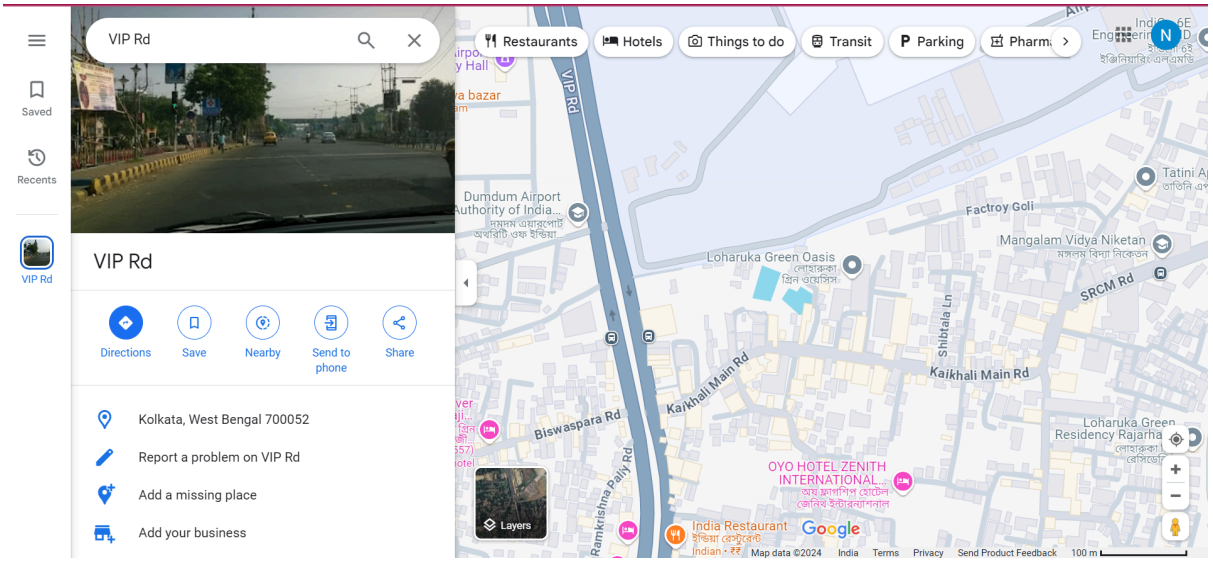
User Interface Testing:

Email Alerts

- An email alert system using Apex triggers to send confirmation emails upon successful registration.



❖ Leverage dynamic email templates to include event details, attendee information, and Google Maps links.



6. Key Scenarios Addressed by Salesforce in the Implementation Project

- **Efficient Event Management:**
  The project successfully implemented a comprehensive event management system that allows organizers to create, manage, and track various fitness events efficiently. It includes features for managing event details, attendees, and locations.

- **Automated Communication:**
  The automated email alert system ensures that attendees receive timely confirmation emails upon registration. This enhances attendee experience and reduces manual communication efforts.

- **Robust Data Handling:**
  With the implementation of Batch Apex processes, the system can efficiently handle large volumes of data, such as deleting outdated event records, ensuring optimal performance without compromising data integrity.

- **Error Management:**
  The reusable error handling framework captures and logs exceptions, enabling easier troubleshooting and improving overall system reliability.

- **Data Security and Compliance:**
  The solution employs Profiles, Roles, Organization-Wide Defaults, and Sharing Rules to control access to sensitive data. This ensures that only authorized users can view or edit specific information, thereby maintaining data integrity.

- **Integration with External Systems:**
  The development of Apex REST APIs facilitates seamless data exchange with third-party systems, enhancing data accessibility and providing real-time updates.

- **Validation and Duplication Checks:**
  Validation rules and duplicate rules prevent the creation of invalid or duplicate records, ensuring data quality and consistency throughout the system.

- **Dynamic User Experience:**
  The user interface is designed to provide a streamlined experience for event managers, speakers, and attendees, making the platform user-friendly and accessible.

7. Conclusion

The implementation of the event management solution for FIT SYNC using the Salesforce platform effectively addresses critical business goals related to event and client management. By leveraging Salesforce's robust features, the project enhances operational efficiency, streamlines communication, and ensures data security. The combination of automated processes, rigorous error handling, and integration capabilities fosters an improved user experience for attendees and organizers alike. Overall, this project showcases the potential of Salesforce to transform event management practices, making them more efficient, secure, and user-centric.