## Phase 5: Apex Programming (Developer) – Event Management & Ticketing System

### 🔹 Objective

In this phase, we extend the functionality of the **Event Management & Ticketing System** by implementing **Apex classes, triggers, and asynchronous processing**. While earlier phases focused on data modeling and automation using Flows, this phase brings in custom logic to handle: - Ticket capacity validation - QR code generation for tickets - Automated attendee status updates - Large-scale operations using asynchronous Apex

This ensures the system is scalable, efficient, and handles real-world event scenarios.

### 🔹 Apex Components

#### 1. Apex Classes & Objects

* **TicketManager.cls** → Generates QR codes for tickets and handles validations.
* **CheckInHandler.cls** → Updates attendee check-in status when QR is scanned.
* **FeedbackHandler.cls** → Automates creation of feedback records after event completion.

**Step 1: Implement TicketManager.cls**

public class TicketManager {  
 public static void assignTicket(Id attendeeId, String ticketType) {  
 // Capacity validation  
 Event\_\_c ev = [SELECT Id, Capacity\_\_c, (SELECT Id FROM Tickets\_\_r) FROM Event\_\_c WHERE Id IN  
 (SELECT Event\_\_c FROM Attendee\_\_c WHERE Id = :attendeeId) LIMIT 1];  
  
 if(ev.Tickets\_\_r.size() >= ev.Capacity\_\_c) {  
 throw new CustomException('Event capacity reached! Cannot assign more tickets.');  
 }  
  
 // Create Ticket with QR Code placeholder  
 Ticket\_\_c t = new Ticket\_\_c(  
 Attendee\_\_c = attendeeId,  
 Ticket\_Type\_\_c = ticketType,  
 Status\_\_c = 'Issued',  
 QR\_Code\_\_c = 'Generated\_Url\_or\_Base64\_String'  
 );  
 insert t;  
 }  
}

#### 2. Apex Triggers

* **EventCapacityTrigger** → Prevents overbooking beyond event capacity.
* **AttendeeCheckInTrigger** → Marks Attendee as “Checked-In” after scanning QR.
* **FeedbackTrigger** → Auto-creates a Feedback\_\_c record after an event is marked “Completed”.

**Example Trigger (EventCapacityTrigger):**

trigger EventCapacityTrigger on Ticket\_\_c (before insert) {  
 for(Ticket\_\_c t : Trigger.new) {  
 Event\_\_c ev = [SELECT Capacity\_\_c, (SELECT Id FROM Tickets\_\_r) FROM Event\_\_c WHERE Id = :t.Event\_\_c LIMIT 1];  
 if(ev.Tickets\_\_r.size() >= ev.Capacity\_\_c) {  
 t.addError('Event capacity exceeded. Cannot assign more tickets.');  
 }  
 }  
}

#### 3. Trigger Design Pattern

All triggers call handler classes instead of writing logic inside the trigger. This makes the system: - Bulk-safe - Maintainable - Scalable

#### 4. SOQL & SOSL Examples

* **SOQL Example:** Fetch all attendees for an event.

List<Attendee\_\_c> attendees = [SELECT Name, Email\_\_c FROM Attendee\_\_c WHERE Event\_\_c = :eventId];

* **SOSL Example:** Search tickets and attendees by name or email.

List<List<SObject>> searchList = [FIND 'John\*' IN ALL FIELDS RETURNING Attendee\_\_c(Name, Email\_\_c), Ticket\_\_c(Name, Status\_\_c)];

#### 5. Asynchronous Apex

* **Batch Apex** → Send bulk reminders to attendees before event.
* **Queueable Apex** → Generate QR code asynchronously after ticket creation.
* **Scheduled Apex** → Run nightly job to send event summary reports.
* **Future Methods** → Send post-event feedback emails asynchronously.

**Example Queueable Class:**

public class QRCodeGeneratorQueueable implements Queueable {  
 private Id ticketId;  
 public QRCodeGeneratorQueueable(Id tId){ this.ticketId = tId; }  
  
 public void execute(QueueableContext qc){  
 Ticket\_\_c t = [SELECT Id FROM Ticket\_\_c WHERE Id = :ticketId LIMIT 1];  
 t.QR\_Code\_\_c = 'Generated\_QR\_String';  
 update t;  
 }  
}

#### 6. Exception Handling

Errors such as exceeding event capacity, invalid QR scan, or missing attendee details will be caught using try-catch blocks.

**Example:**

try {  
 TicketManager.assignTicket(attendeeId, 'VIP');  
} catch(Exception e) {  
 System.debug('Error: ' + e.getMessage());  
}

#### 7. Test Classes

* Ensure **75%+ code coverage**.
* Validate ticket assignment, capacity validation, QR code generation, and feedback automation.

**Example Test Class:**

@isTest  
public class TicketManagerTest {  
 @isTest static void testAssignTicket(){  
 Event\_\_c ev = new Event\_\_c(Name='Tech Conference', Capacity\_\_c=100);  
 insert ev;  
  
 Attendee\_\_c att = new Attendee\_\_c(Name='Test User', Email\_\_c='test@example.com', Event\_\_c=ev.Id);  
 insert att;  
  
 Test.startTest();  
 TicketManager.assignTicket(att.Id, 'General');  
 Test.stopTest();  
  
 List<Ticket\_\_c> tickets = [SELECT Id FROM Ticket\_\_c WHERE Attendee\_\_c = :att.Id];  
 System.assertEquals(1, tickets.size());  
 }  
}

### 🔹 Deliverables (Phase 5)

* Apex Classes for ticket assignment, check-in, and feedback.
* Triggers with handler classes.
* Batch, Queueable, Scheduled Apex jobs.
* Exception handling for safe execution.
* Unit test classes with high coverage.

### 🔹 Outcome

✅ Prevents ticket overbooking and ensures event capacity compliance.  
✅ Automates ticket QR code generation & attendee check-in.  
✅ Provides scalable async solutions for bulk operations (reminders, reports).  
✅ Strengthens reliability with exception handling & test coverage.

This completes **Phase 5: Apex Programming (Developer)** for the *Event Management & Ticketing System*.