Phase 5: Apex Programming (Developer)

Flight Reservation & Scheduling System

Salesforce-Based Flight Operations and Scheduling System

Step 1: Log in to Salesforce Developer Org

1. Go to Salesforce Developer and sign up if you don’t have an org.
2. Log in to your **Developer Edition** or **Sandbox**.
3. Switch to **Lightning Experience** for easier navigation.

Step 2: Create Custom Objects

Purpose: Define your Airline entities.

Steps:

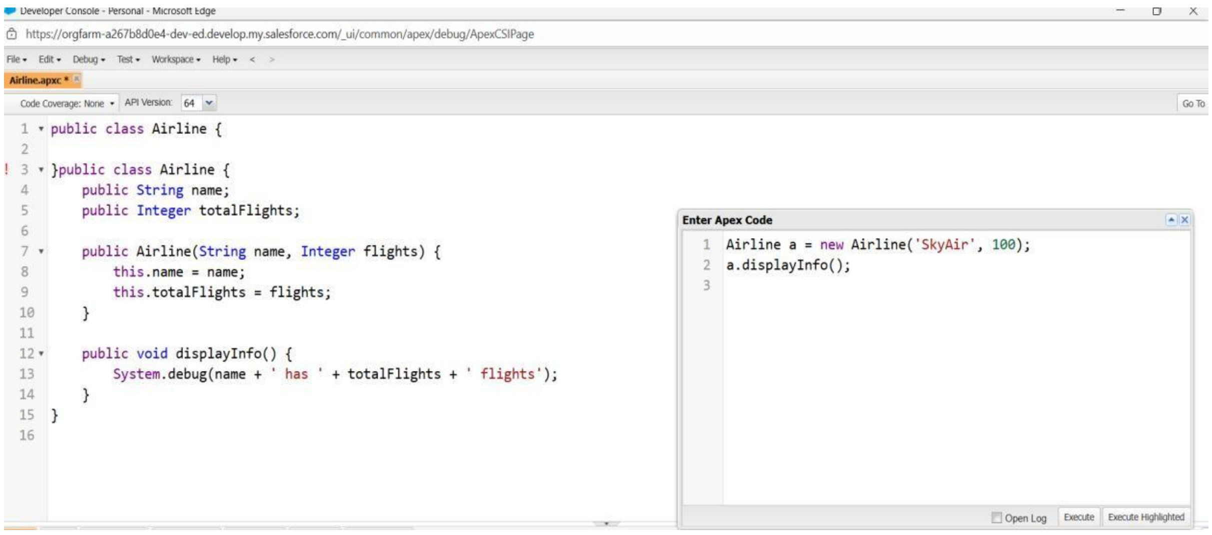
1. Go to **Setup → Object Manager → Create → Custom Object**.
2. Create these objects:
   - Airline__c
   - Flight__c
   - Passenger__c
   - Booking__c
3. Add **custom fields**:
   - Flight__c: Name, Status (Picklist: Scheduled, Completed, Cancelled), DepartureDate (DateTime)
   - Airline__c: Name, TotalFlights (Number)
   - Passenger__c: Name, Email
   - Booking__c: Flight (Lookup to Flight__c), Passenger (Lookup to Passenger__c)

Step 3: Create Apex Classes

Purpose: Write logic to manage your objects.

Steps:

1. Go to **Setup → Apex Classes → New**.
2. Create a class like Airline:


Developer Console - Personal - Microsoft Edge
https://orgfarm-a267b8d0e4-dev-ed.develop.my.salesforce.com/_ui/common/apex/debug/ApexCSIPage
File Edit Debug Test Workspace Help
Airline.apxc *
Code Coverage: None API Version: 64
Go To
Enter Apex Code
Open Log
Execute
Execute Highlighted

```
public class Airline {

}public class Airline {
    public String name;
    public Integer totalFlights;

    public Airline(String name, Integer flights) {
        this.name = name;
        this.totalFlights = flights;
    }

    public void displayInfo() {
        System.debug(name + ' has ' + totalFlights + ' flights');
    }
}
```

```
Airline a = new Airline('SkyAir', 100);
a.displayInfo();
```

- Click **Save**.
- Test the class using **Developer Console → Debug → Open Execute Anonymous Window**:

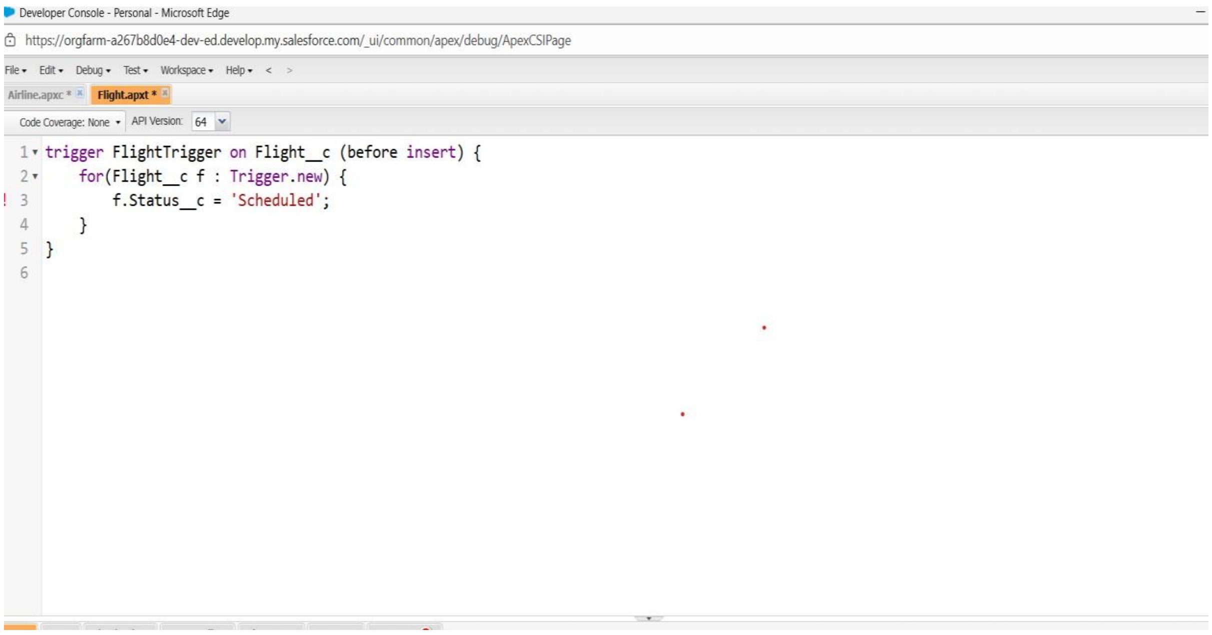
  `Airline a = new Airline('SkyAir', 100); a.displayInfo();`

Step 4: Create Apex Triggers

Purpose: Automate actions when records are created/updated/deleted.

Steps:

1. Go to **Setup → Object Manager → Flight__c → Triggers → New**.
2. Create a trigger:


Developer Console - Personal - Microsoft Edge
https://orgfarm-a267b8d0e4-dev-ed.develop.my.salesforce.com/_ui/common/apex/debug/ApexCSIPage
File Edit Debug Test Workspace Help
Airline.apxc *
Flight.apxt *
Code Coverage: None API Version: 64

```
trigger FlightTrigger on Flight__c (before insert) {
    for(Flight__c f : Trigger.new) {
        f.Status__c = 'Scheduled';
    }
}
```

3. Click **Save**.

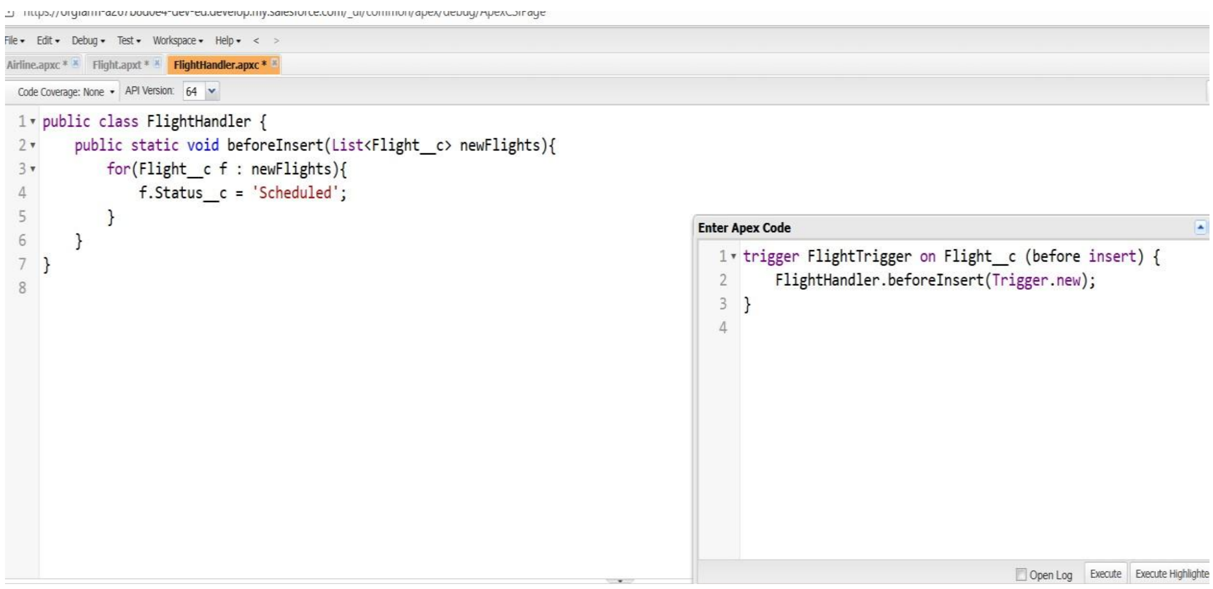
4. Test: Go to **Flight__c** → **New Record**, add a flight → **Save** → check Status is automatically "Scheduled".

Step 5: Implement Trigger Handler (Design Pattern)

Purpose: Keep triggers clean.

Steps:

1. Create **Apex Class** → **FlightHandler**:



```
public class FlightHandler {
    public static void beforeInsert(List<Flight__c> newFlights){
        for(Flight__c f : newFlights){
            f.Status__c = 'Scheduled';
        }
    }
}
```

```
trigger FlightTrigger on Flight__c (before insert) {
    FlightHandler.beforeInsert(Trigger.new);
}
```

2. Update trigger to delegate:

```
trigger FlightTrigger on Flight__c (before insert) {
FlightHandler.beforeInsert(Trigger.new); }
```

Step 6: Use SOQL & SOSL

Purpose: Query records in Salesforce.

Steps:

1. Open **Developer Console** → **Query Editor**.

2. Run a SOQL query:

```
SELECT Name, Status__c FROM Flight__c WHERE Status__c='Scheduled'
```

3. Run a SOSL query in **Apex**:

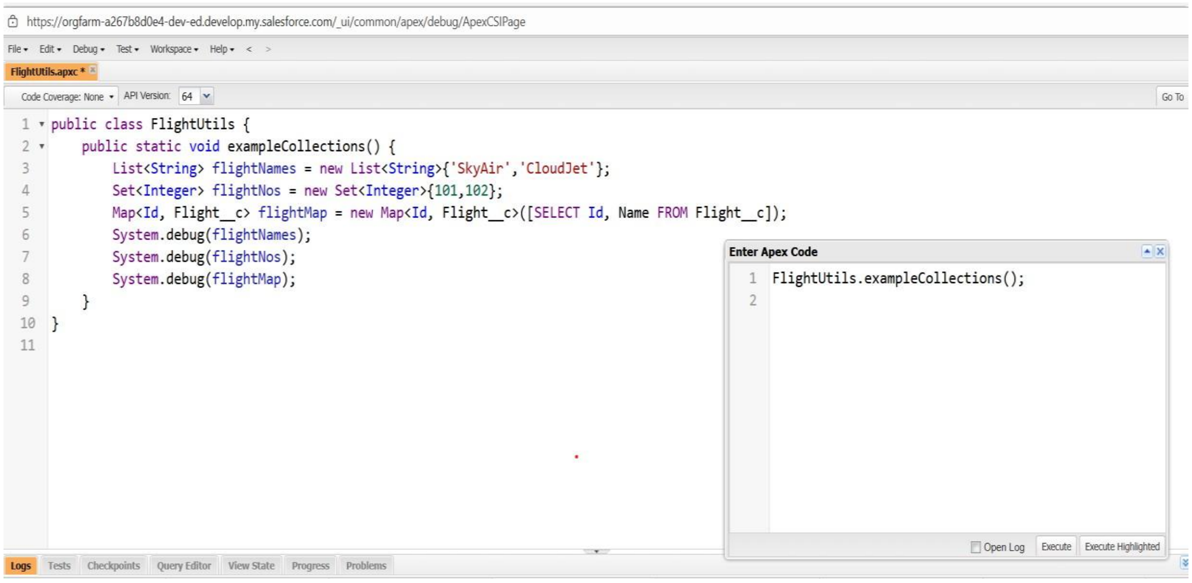
```
List<List<SObject>> results = [FIND 'SkyAir*' IN ALL FIELDS RETURNING
Flight__c(Name, Status__c)];
```

Step 7: Use Collections

Purpose: Handle multiple records in code.

Steps:

1. Create a new Apex Class FlightUtils:



https://orgfarm-a267b8d0e4-dev-ed.develop.my.salesforce.com/_ui/common/apex/debug/ApexCSIPage

FlightUtils.apxc *

Code Coverage: None · API Version: 64

```
public class FlightUtils {
    public static void exampleCollections() {
        List<String> flightNames = new List<String>{'SkyAir','CloudJet'};
        Set<Integer> flightNos = new Set<Integer>{101,102};
        Map<Id, Flight__c> flightMap = new Map<Id, Flight__c>([SELECT Id, Name FROM Flight__c]);
        System.debug(flightNames);
        System.debug(flightNos);
        System.debug(flightMap);
    }
}
```

Enter Apex Code

```
FlightUtils.exampleCollections();
```

2. Execute in **Developer Console → Execute Anonymous**:

```
FlightUtils.exampleCollections();
```

Step 8: Use Control Statements

Purpose: Handle conditions and loops.

Steps:

1. In your Apex Class, use if-else:

```
for(Flight__c f : [SELECT Name, Status__c FROM Flight__c]){
if(f.Status__c == 'Cancelled') System.debug(f.Name + ' is cancelled.');
else System.debug(f.Name + ' is active.'); }
```

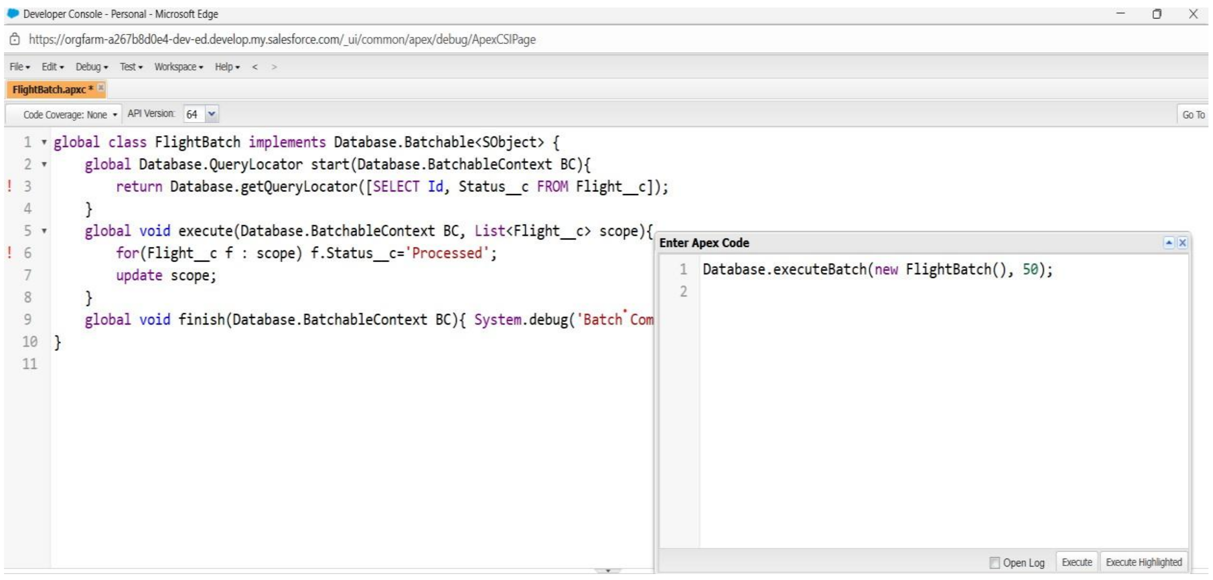
2. Execute in **Anonymous Window**.

Step 9: Implement Batch Apex

Purpose: Process large datasets.

Steps:

1. Create **Apex Class → FlightBatch**:


Developer Console - Personal - Microsoft Edge
https://orgfarm-a267b8d0e4-dev-ed.develop.my.salesforce.com/_ui/common/apex/debug/ApexCSIPage
File Edit Debug Test Workspace Help
FlightBatch.apxc *
Code Coverage: None API Version: 64
Go To
Enter Apex Code
Open Log
Execute
Execute Highlighted

```
global class FlightBatch implements Database.Batchable<SObject> {
    global Database.QueryLocator start(Database.BatchableContext BC){
        return Database.getQueryLocator([SELECT Id, Status__c FROM Flight__c]);
    }
    global void execute(Database.BatchableContext BC, List<Flight__c> scope){
        for(Flight__c f : scope) f.Status__c='Processed';
        update scope;
    }
    global void finish(Database.BatchableContext BC){ System.debug('Batch Com
}
```

```
Database.executeBatch(new FlightBatch(), 50);
```

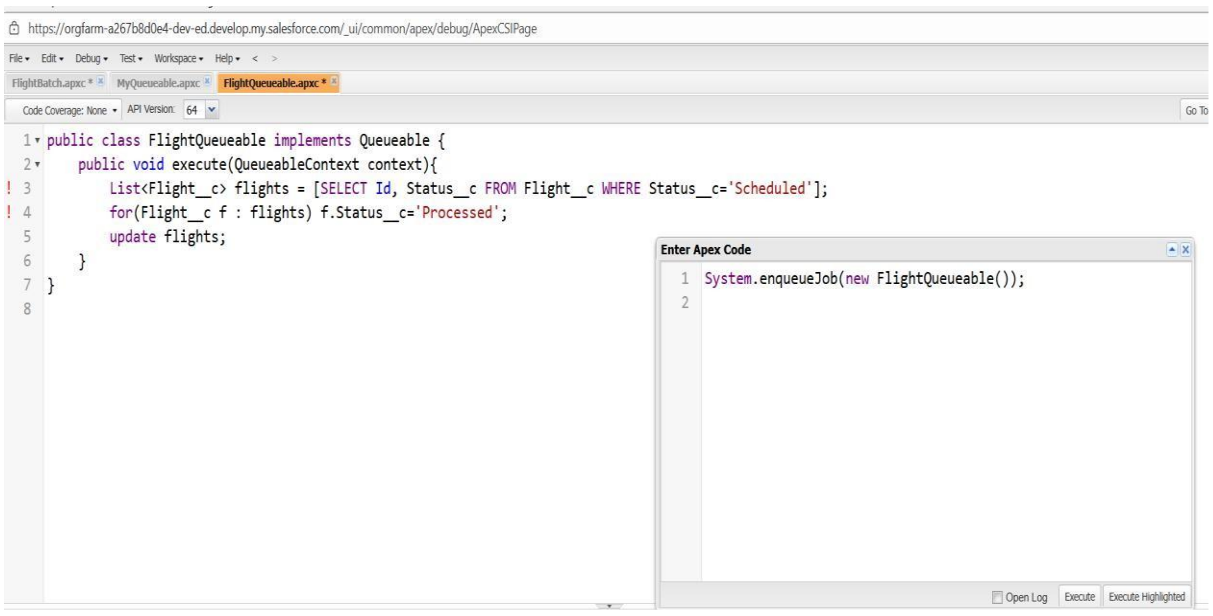
2. Run in **Execute Anonymous**:

```
Database.executeBatch(new FlightBatch(), 50);
```

Step 10: Implement Queueable Apex

Steps:

1. Create **Apex Class → FlightQueueable**:


https://orgfarm-a267b8d0e4-dev-ed.develop.my.salesforce.com/_ui/common/apex/debug/ApexCSIPage
File Edit Debug Test Workspace Help
FlightBatch.apxc *
MyQueueable.apxc
FlightQueueable.apxc *
Code Coverage: None API Version: 64
Go To
Enter Apex Code
Open Log
Execute
Execute Highlighted

```
public class FlightQueueable implements Queueable {
    public void execute(QueueableContext context){
        List<Flight__c> flights = [SELECT Id, Status__c FROM Flight__c WHERE Status__c='Scheduled'];
        for(Flight__c f : flights) f.Status__c='Processed';
        update flights;
    }
}
```

```
System.enqueueJob(new FlightQueueable());
```

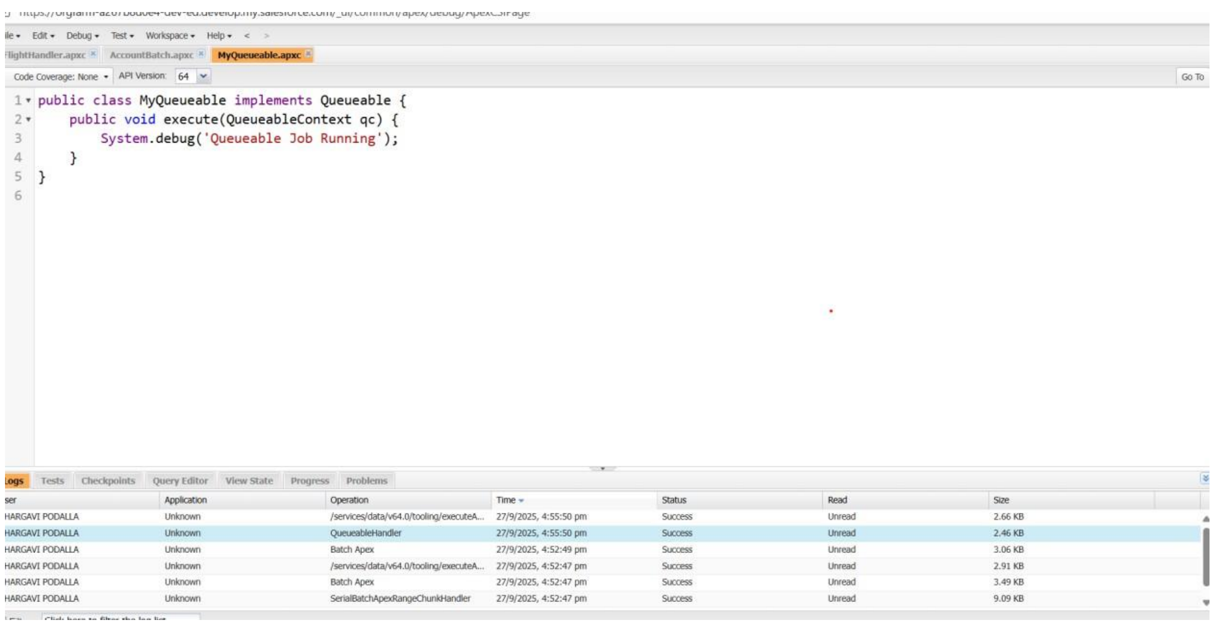
2. Enqueue job:

```
System.enqueueJob(new FlightQueueable());
```

Step 11: Implement Scheduled Apex

Steps:

1. Create **Apex Class → FlightScheduler**:



```
public class MyQueueable implements Queueable {
    public void execute(QueueableContext qc) {
        System.debug('Queueable Job Running');
    }
}
```

ser	Application	Operation	Time	Status	Read	Size
HARGAVI PODALLA	Unknown	/services/data/v64.0/tooling/executeA...	27/9/2025, 4:55:50 pm	Success	Unread	2.66 KB
HARGAVI PODALLA	Unknown	QueueableHandler	27/9/2025, 4:55:50 pm	Success	Unread	2.46 KB
HARGAVI PODALLA	Unknown	Batch Apex	27/9/2025, 4:52:49 pm	Success	Unread	3.06 KB
HARGAVI PODALLA	Unknown	/services/data/v64.0/tooling/executeA...	27/9/2025, 4:52:47 pm	Success	Unread	2.91 KB
HARGAVI PODALLA	Unknown	Batch Apex	27/9/2025, 4:52:47 pm	Success	Unread	3.49 KB
HARGAVI PODALLA	Unknown	SerialBatchApexRangeChunkHandler	27/9/2025, 4:52:47 pm	Success	Unread	9.09 KB

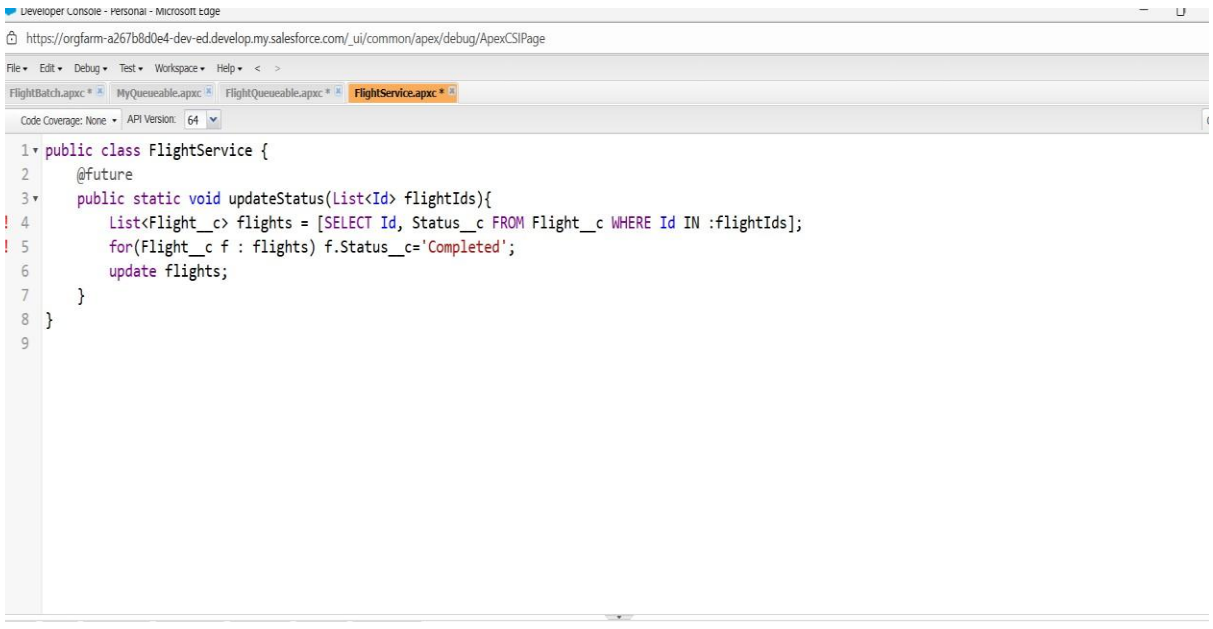
2. Schedule job in **Execute Anonymous**:

```
String cronExp = '0 0 12 * * ?'; // Daily 12 PM
System.schedule('DailyFlightJob', cronExp, new FlightScheduler());
```

Step 12: Use Future Methods

Steps:

1. Create **Apex Class → FlightService**:



```
public class FlightService {
    @future
    public static void updateStatus(List<Id> flightIds){
        List<Flight__c> flights = [SELECT Id, Status__c FROM Flight__c WHERE Id IN :flightIds];
        for(Flight__c f : flights) f.Status__c='Completed';
        update flights;
    }
}
```

2. Call asynchronously:

```
FlightService.updateStatus(new List<Id>{'a0F1t00000123AB'});
```

Step 13: Exception Handling

Steps:

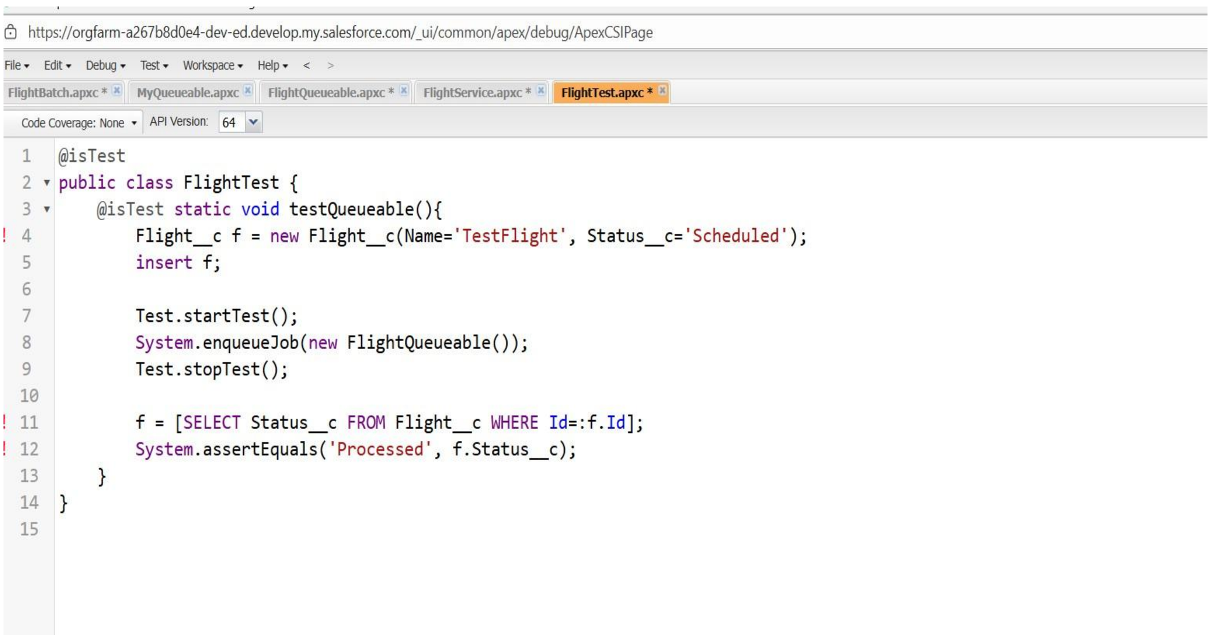
1. Wrap DML operations in **try-catch**:

```
try {
   Flight__c f = [SELECT Id FROM Flight__c WHERE Name='SkyAir' LIMIT
1];    f.Status__c='Delayed';     update f;
} catch(DmlException e) {
   System.debug('Error: ' + e.getMessage()); }
```

Step 14: Write Test Classes

Steps:

1. Create **Apex Class → FlightTest**:



https://orgfarm-a267b8d0e4-dev-ed.develop.my.salesforce.com/_ui/common/apex/debug/ApexCSIPage

File ▾ Edit ▾ Debug ▾ Test ▾ Workspace ▾ Help ▾ < >
FlightBatch.apxc * MyQueueable.apxc FlightQueueable.apxc * FlightService.apxc * FlightTest.apxc *
Code Coverage: None ▾ API Version: 64

```
@isTest
public class FlightTest {
    @isTest static void testQueueable(){
        Flight__c f = new Flight__c(Name='TestFlight', Status__c='Scheduled');
        insert f;

        Test.startTest();
        System.enqueueJob(new FlightQueueable());
        Test.stopTest();

        f = [SELECT Status__c FROM Flight__c WHERE Id=:f.Id];
        System.assertEquals('Processed', f.Status__c);
    }
}
```

2. Run tests in **Setup → Apex Test Execution → Run All Tests**.

Following this, you will have **fully implemented Apex logic, triggers, asynchronous processing, and test coverage in your Salesforce Org** for the Airline Management System.