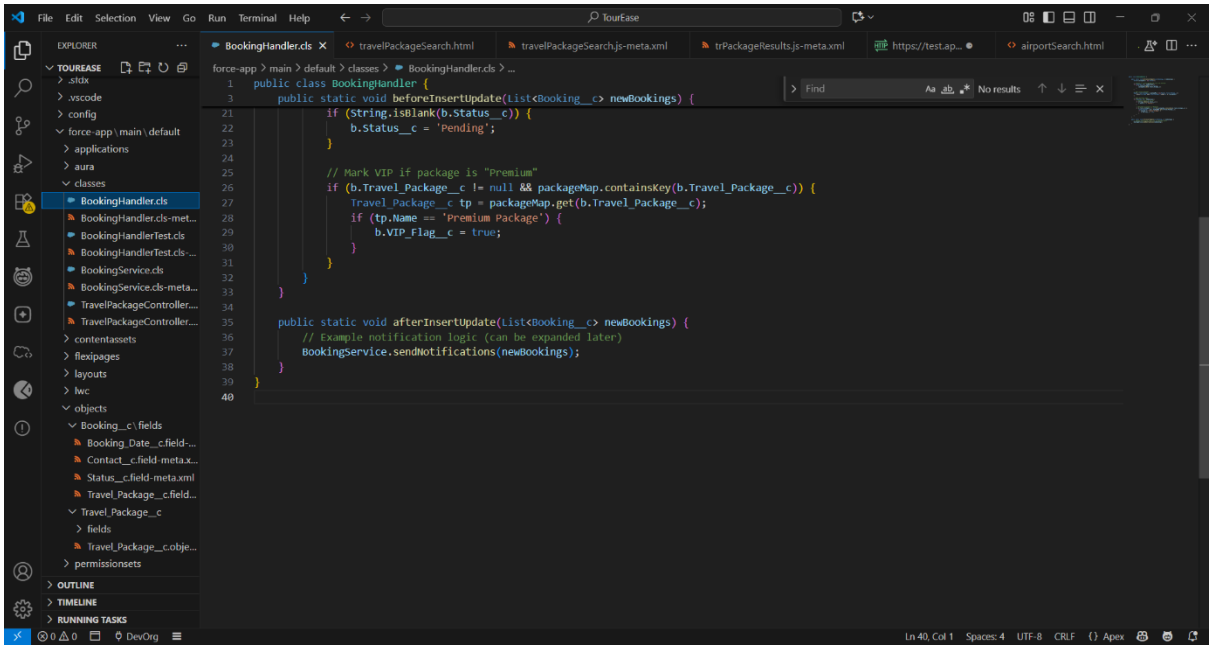
Phase 5: Apex Programming (Developer) – TourEase Project

1. Classes & Objects

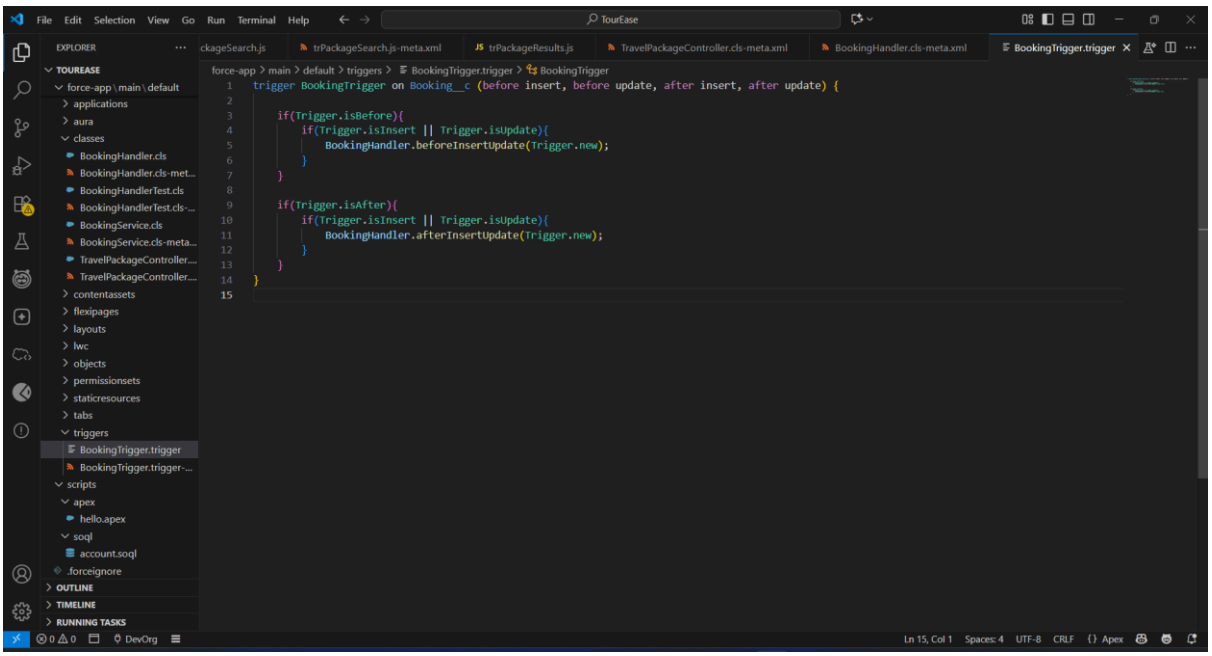
- **Purpose:** Encapsulate logic and data; create reusable methods for your project.



---

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

- **Purpose:** Execute logic automatically when records change.



---

3. Trigger Design Pattern

- **Purpose:** Organize triggers to separate business logic (trigger handler pattern).
- **Example:**

```
trigger BookingTrigger on Booking__c (before insert, before update) {
    BookingTriggerHandler.handleBeforeInsertUpdate(Trigger.new);
}

public class BookingTriggerHandler {
    public static void handleBeforeInsertUpdate(List<Booking__c> bookings){
        for(Booking__c b : bookings){
            if(b.Status__c == null) b.Status__c = 'Pending';
        }
    }
}
```

---

4. SOQL & SOSL

- **Purpose:** Query Salesforce data (SOQL) or search records (SOSL).

---

5. Collections: List, Set, Map

- **Purpose:** Store multiple records efficiently.
- **Example:**

```
List<Booking__c> bookings = [SELECT Id, Name FROM Booking__c];
Set<Id> bookingIds = new Set<Id>();
Map<Id, Booking__c> bookingMap = new Map<Id, Booking__c>(bookings);
```

---

6. Control Statements

- **Purpose:** Apply logic using if, for, while, switch.
- **Example:**

```
for(Booking__c b : bookings){
    if(b.Status__c == 'Pending'){
        b.Status__c = 'Confirmed';
    }
```

```
}
```

---

7. Batch Apex

- **Purpose:** Process large datasets asynchronously.
- **Example:**

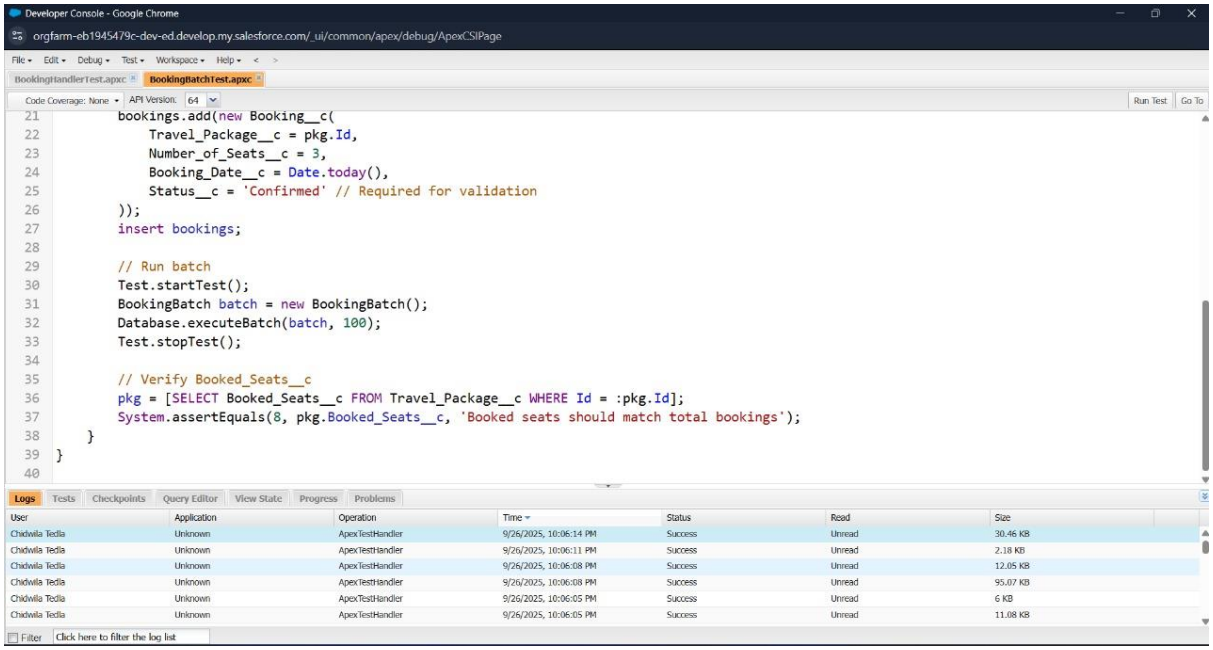
```
global class BookingBatch implements Database.Batchable<sObject> {
    global Database.QueryLocator start(Database.BatchableContext BC){
        return Database.getQueryLocator([SELECT Id, Status__c FROM Booking__c WHERE Status__c='Pending']);
    }
    global void execute(Database.BatchableContext BC, List<Booking__c> scope){
        for(Booking__c b : scope) b.Status__c = 'Confirmed';
        update scope;
    }
    global void finish(Database.BatchableContext BC){}
}
```

---

8. Queueable Apex

- **Purpose:** Run asynchronous jobs with complex logic.
- **Example:**

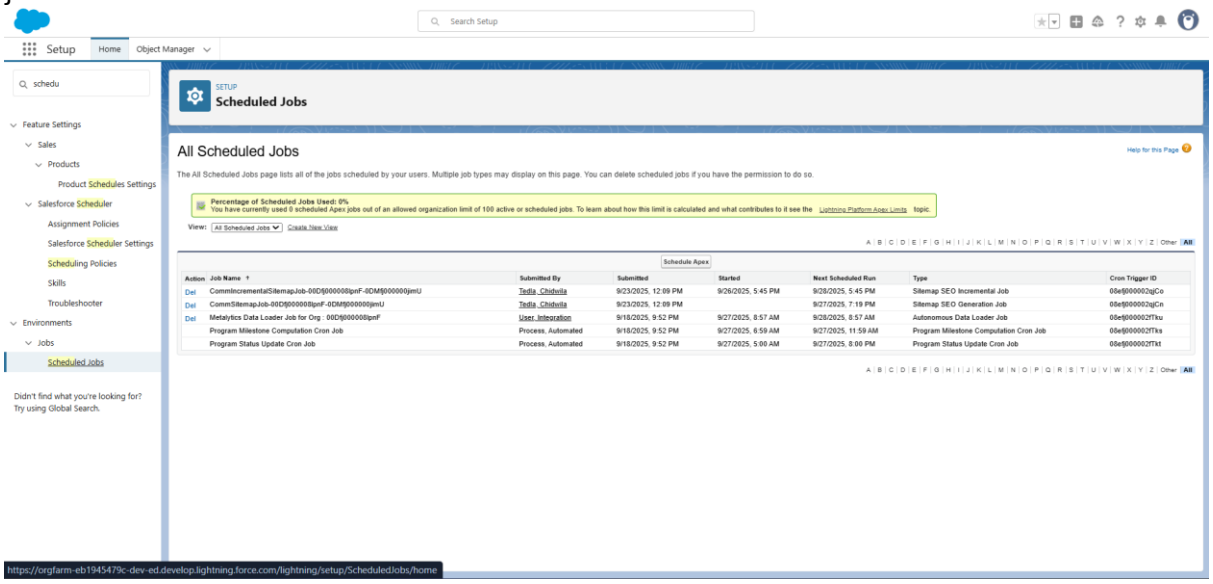
```
public class BookingQueueable implements Queueable {
    public void execute(QueueableContext context){
        List<Booking__c> bookings = [SELECT Id, Status__c FROM Booking__c WHERE Status__c='Pending'];
        for(Booking__c b : bookings) b.Status__c = 'Confirmed';
        update bookings;
    }
}
```



9. Scheduled Apex

- **Purpose:** Run Apex classes at a scheduled time.
- **Example:**

```
public class BookingScheduler implements Schedulable {
    public void execute(SchedulableContext sc){
        BookingQueueable job = new BookingQueueable();
        System.enqueueJob(job);
    }
}
```

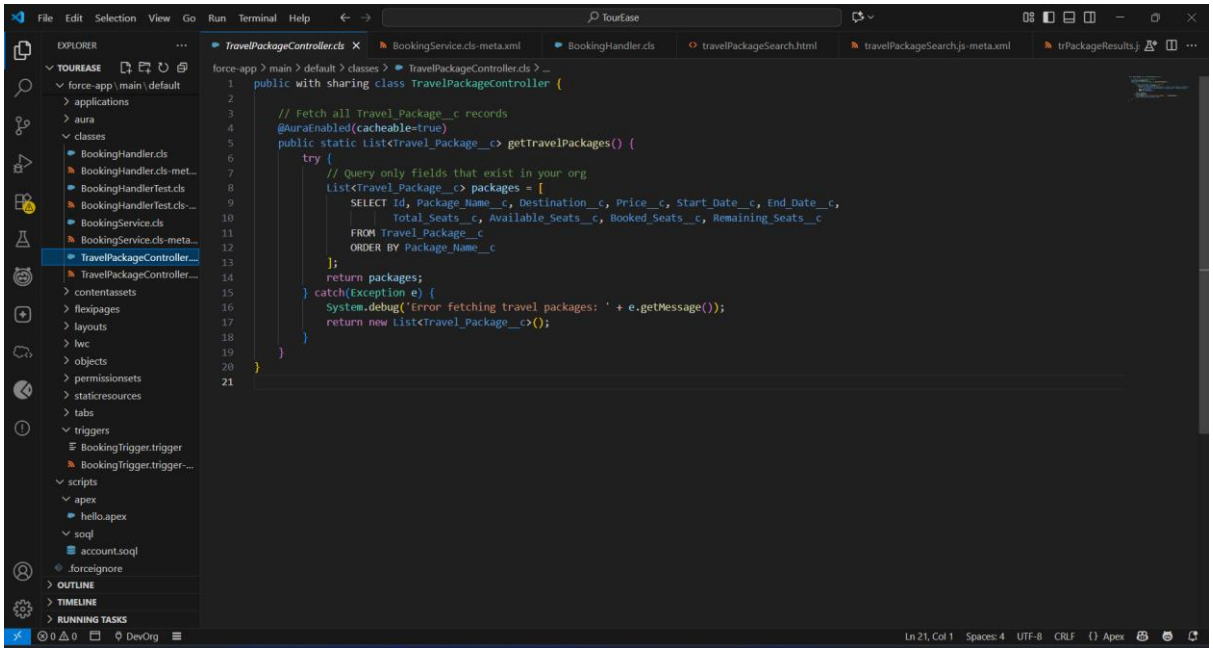


---

10. Exception Handling

- **Purpose:** Catch and manage runtime errors.
- **Example:**

```
try {
    update bookings;
} catch(DmlException e){
    System.debug('Error updating bookings: ' + e.getMessage());
}
```



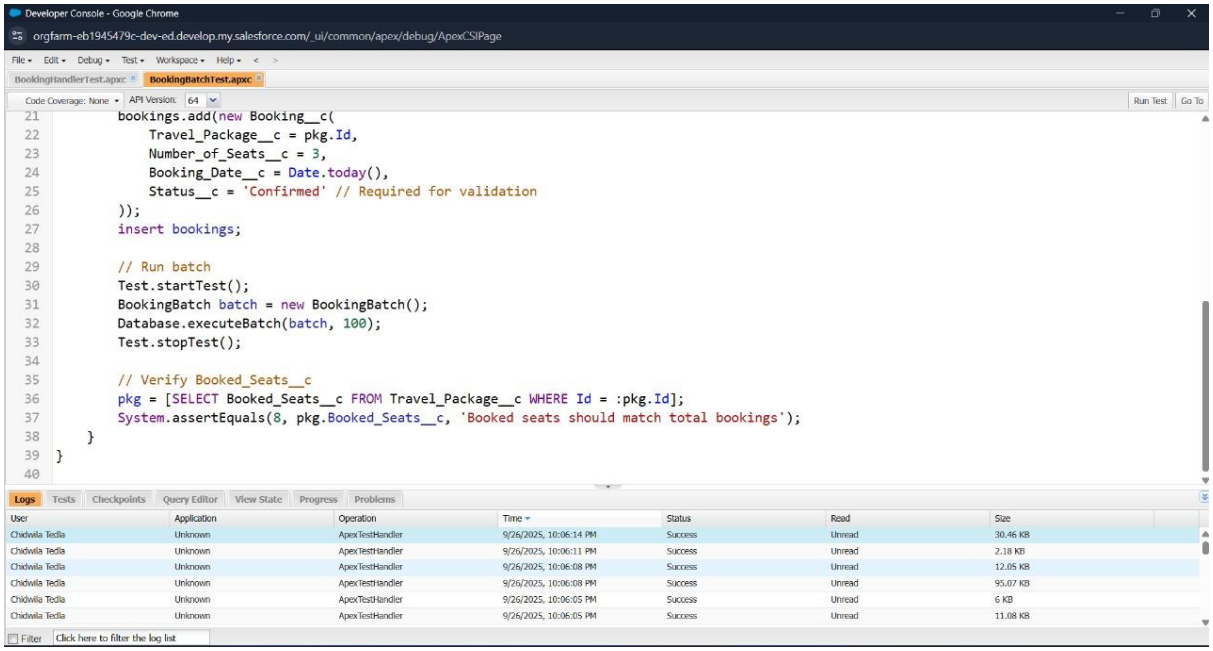
---

11. Test Classes

- **Purpose:** Ensure code works and deploy to production.
- **Example:**

```
@isTest
public class BookingControllerTest {
    @isTest static void testConfirmBooking(){
        Booking__c b = new Booking__c(Name='Test Booking', Status__c='Pending');
        insert b;
        BookingController.confirmBooking(b.Id);
        b = [SELECT Status__c FROM Booking__c WHERE Id=:b.Id];
        System.assertEquals('Confirmed', b.Status__c);
```

```
    }
}
```



Conclusion:

In Phase 5, the TourEase project utilized Apex programming to automate and manage core booking processes, including notifications and data handling. By implementing classes, triggers, SOQL/SOSL queries, collections, and control logic, along with asynchronous methods like Batch, Queueable, and Scheduled Apex, the system efficiently processes data while ensuring reliability through exception handling and test classes. This phase highlights how structured Apex code enables scalable, maintainable, and automated operations in a real-world Salesforce application.
