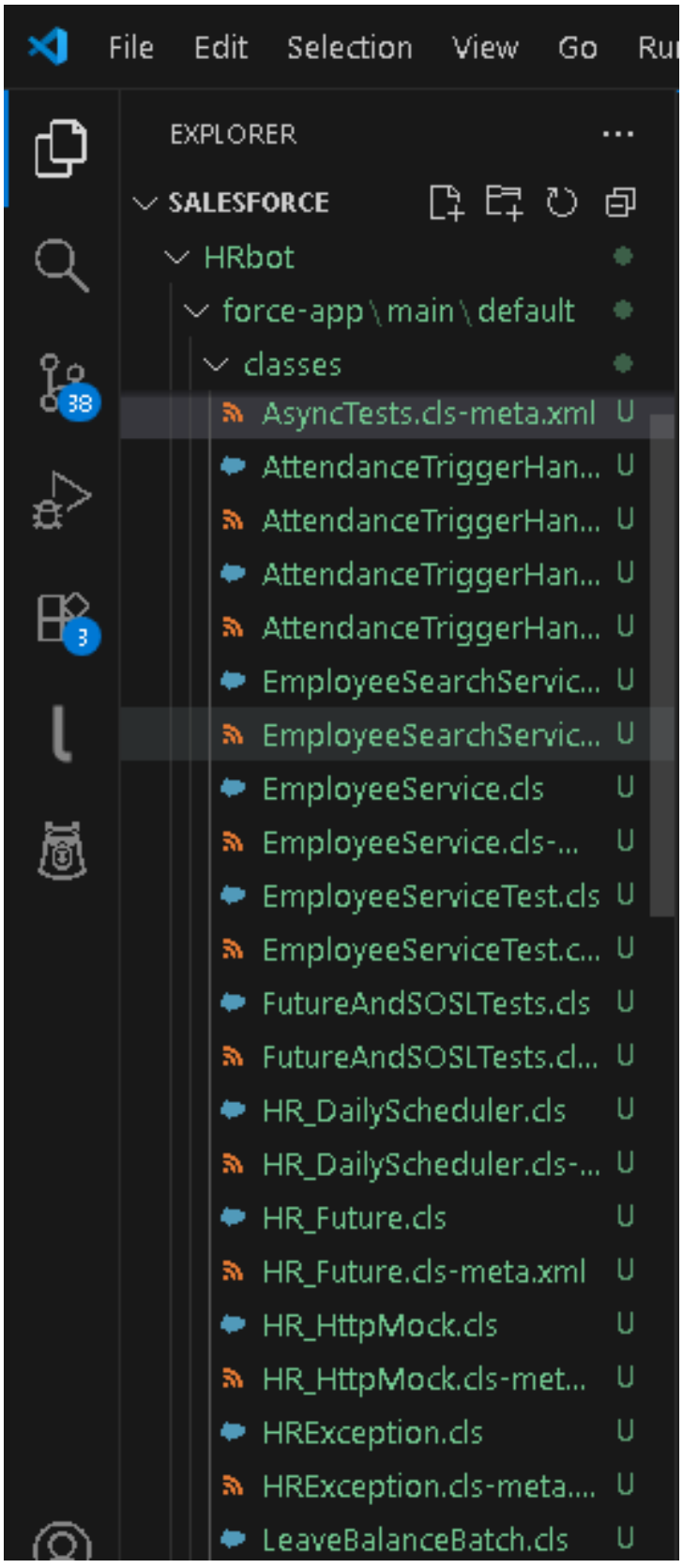
PHASE - 5

Project: AI-Powered HR & Employee Management Bot

1. Classes & Objects (create core classes)

- Create files in force-app/main/default/classes
  - LeaveTriggerHandler.cls
  - AttendanceTriggerHandler.cls
  - EmployeeService.cls
  - EmployeeSearchService.cls
  - RecalcLeaveBalanceQueueable.cls
  - LeaveBalanceBatch.cls
  - HR_DailyScheduler.cls
  - HR_Future.cls
  - HRException.cls
  - TestDataFactory.cls



1. Apex Triggers (before/after) using the Trigger Design Pattern

- One trigger per object → route to a handler class.

Create: force-app/main/default/triggers/LeaveTrigger.trigger

trigger LeaveTrigger on Leave__c (before insert, before update, after insert, after update) {

LeaveTriggerHandler h = new LeaveTriggerHandler();

if (Trigger.isBefore) {

```
if (Trigger.isInsert) h.beforeInsert(Trigger.new);
if (Trigger.isUpdate) h.beforeUpdate(Trigger.new, Trigger.oldMap);
} else if (Trigger.isAfter) {
if (Trigger.isInsert) h.afterInsert(Trigger.new);
if (Trigger.isUpdate) h.afterUpdate(Trigger.new, Trigger.oldMap);
}
}

Create: force-app/main/default/classes/LeaveTriggerHandler.cls
public with sharing class LeaveTriggerHandler {
public void beforeInsert(List<Leave__c> newList) {
calculateDurations(newList);
validateOverlaps(newList, null);
}
public void beforeUpdate(List<Leave__c> newList, Map<Id, Leave__c> oldMap) {
calculateDurations(newList);
validateOverlaps(newList, oldMap);
}
public void afterInsert(List<Leave__c> newList) {
Set<Id> empIds = new Set<Id>();
Set<Id> approvedLeaveIds = new Set<Id>();
for (Leave__c l : newList) {
if (l.Employee__c != null) empIds.add(l.Employee__c);
if (l.Approval_Status__c == 'Approved') approvedLeaveIds.add(l.Id);
}
if (!empIds.isEmpty() && !approvedLeaveIds.isEmpty()) {
System.enqueueJob(new RecalcLeaveBalanceQueueable(empIds));
HR_Future.sendLeaveApprovedWebhook(approvedLeaveIds);
}
}
public void afterUpdate(List<Leave__c> newList, Map<Id, Leave__c> oldMap) {
Set<Id> empIds = new Set<Id>();
Set<Id> approvedLeaveIds = new Set<Id>();
for (Leave__c l : newList) {
if (l.Employee__c != null) empIds.add(l.Employee__c);
Leave__c oldRec = oldMap.get(l.Id);
if (oldRec.Approval_Status__c != 'Approved' && l.Approval_Status__c == 'Approved') {
approvedLeaveIds.add(l.Id);
}
}
if (!empIds.isEmpty() && !approvedLeaveIds.isEmpty()) {
System.enqueueJob(new RecalcLeaveBalanceQueueable(empIds));
HR_Future.sendLeaveApprovedWebhook(approvedLeaveIds);
}
}

text
```

```
private void calculateDurations(List<Leave__c> list) {
    for (Leave__c l : list) {
        if (l.Start_Date__c != null && l.End_Date__c != null) {
            if (l.Start_Date__c > l.End_Date__c) {
                l.addError('End Date must be same or after Start Date.');
            } else {
                l.Duration_Days__c = l.Start_Date__c.daysBetween(l.End_Date__c) + 1;
            }
        }
    }
}

private void validateOverlaps(List<Leave__c> newList, Map<Id, Leave__c> oldMap) {
    Set<Id> empIds = new Set<Id>();
    for (Leave__c l : newList) if (l.Employee__c != null) empIds.add(l.Employee__c);
    if (empIds.isEmpty()) return;

    Map<Id, List<Leave__c>> existingByEmp = new Map<Id, List<Leave__c>>();
    for (Leave__c e : [
        SELECT Id, Employee__c, Start_Date__c, End_Date__c, Approval_Status__c
        FROM Leave__c
        WHERE Employee__c IN :empIds AND Approval_Status__c IN ('Submitted','Approved')
    ]) {
        if (!existingByEmp.containsKey(e.Employee__c)) {
```

```
        existingByEmp.put(e.Employee__c, new List<Leave__c>());
      }
      existingByEmp.get(e.Employee__c).add(e);
    }

    for (Leave__c l : newList) {
      if (l.Employee__c == null || l.Start_Date__c == null || l.End_Date__c == null) continue;
      List<Leave__c> bucket = existingByEmp.get(l.Employee__c);
      if (bucket == null) continue;
      for (Leave__c e : bucket) {
        if (oldMap != null && e.Id == l.Id) continue; // ignore self on update
        if (l.Start_Date__c <= e.End_Date__c && e.Start_Date__c <= l.End_Date__c) {
          l.addError('Overlapping leave exists for these dates.');
          break;
        }
      }
    }
  }
}

}
```

Create: force-app/main/default/triggers/AttendanceTrigger.trigger
trigger AttendanceTrigger on Attendance__c (before insert, before update) {
AttendanceTriggerHandler h = new AttendanceTriggerHandler();
if (Trigger.isBefore) {
if (Trigger.isInsert) h.beforeInsert(Trigger.new);
if (Trigger.isUpdate) h.beforeUpdate(Trigger.new, Trigger.oldMap);
}
}

Create: force-app/main/default/classes/AttendanceTriggerHandler.cls
public with sharing class AttendanceTriggerHandler {

```
public void beforeInsert(List<Attendance__c> newList) {
validate(newList, null, true);
}
public void beforeUpdate(List<Attendance__c> newList, Map<Id, Attendance__c> oldMap) {
validate(newList, oldMap, false);
}
```

text

```
private void validate(List<Attendance__c> list, Map<Id, Attendance__c> oldMap, Boolean isInsert) {

  Set<Id> empIds = new Set<Id>();

  Map<String, Attendance__c> newKeys = new Map<String, Attendance__c>();



  for (Attendance__c a : list) {

    if (a.Attendance_Date__c != null && a.Attendance_Date__c > Date.today()) {

      a.addError('Attendance date cannot be in the future.');

    }

    if (a.Employee__c != null && a.Attendance_Date__c != null) {

      empIds.add(a.Employee__c);

      String key = a.Employee__c + ':' + String.valueOf(a.Attendance_Date__c);

      if (newKeys.containsKey(key)) {

        a.addError('Duplicate attendance in this transaction for the same date.');

      } else {

        newKeys.put(key, a);

      }

    }

  }



  if (empIds.isEmpty()) return;
```

```
    for (Attendance__c existing : [
        SELECT Id, Employee__c, Attendance_Date__c
        FROM Attendance__c
        WHERE Employee__c IN :empIds
    ]) {
        String key = existing.Employee__c + ':' + String.valueOf(existing.Attendance_Date__c);
        if (newKeys.containsKey(key)) {
            if (oldMap == null || existing.Id != newKeys.get(key).Id) {
                newKeys.get(key).addError('Attendance already exists for this employee and date.');
            }
        }
    }
}
}
```

Trigger Design Pattern rules you're following

- One trigger per object, zero logic in trigger body.
- All logic routed to a handler with methods per context (beforeInsert, afterUpdate, etc.).
- Bulk-safe: use Sets/Maps; 1 SOQL per dataset; never query inside loops.
- Use addError for validation; push heavy work to async (Queueable/Future).

1. SOQL & SOSL (exact queries you need)
   Create: force-app/main/default/classes/EmployeeService.cls
   public with sharing class EmployeeService {
   public static void recalcLeaveBalances(Set<Id> employeeIds) {
   if (employeeIds == null || employeeIds.isEmpty()) return;
2. text

```
3.  Map<Id, Decimal> usedByEmp = new Map<Id, Decimal>();
4.  for (AggregateResult ar : [
5.      SELECT Employee__c emp, SUM(Duration_Days__c) used
6.      FROM Leave__c
7.      WHERE Approval_Status__c = 'Approved' AND Employee__c IN :employeeIds
8.      GROUP BY Employee__c
9.  ]) {
10.     usedByEmp.put((Id)ar.get('emp'), (Decimal)ar.get('used'));
```

```
    }

    List<Employee__c> updates = new List<Employee__c>();
    for (Employee__c e : [
        SELECT Id, Annual_Leave_Entitlement__c FROM Employee__c WHERE Id IN :employeeIds
    ]) {
        Decimal entitlement = e.Annual_Leave_Entitlement__c == null ? 0 : e.Annual_Leave_Entitlement__c;
        Decimal used = usedByEmp.containsKey(e.Id) ? usedByEmp.get(e.Id) : 0;
        e.Available_Leave_Balance__c = entitlement - used;
        updates.add(e);
    }
    if (!updates.isEmpty()) update updates;
}
}
```

Create: force-app/main/default/classes/EmployeeSearchService.cls

```
public with sharing class EmployeeSearchService {
@AuraEnabled(cacheable=true)
public static List<Employee__c> searchEmployees(String term) {
String q = String.isBlank(term) ? '' : term + '';
List<List<SObject>> results = [FIND :q IN ALL FIELDS RETURNING Employee__c(Id, Name)];
return (List<Employee__c>)results[0];
}
}
```

1. Collections: List, Set, Map (how you use them)
   - Set<Id> to collect unique Employee__c IDs for bulk queries.
   - Map<Id, List<Leave__c>> to group existing leave per employee for overlap validation.
   - Map<String, Attendance__c> to de-dupe within the same transaction (key = EmployeeId:Date).
   - Lists to batch updates (List<Employee__c> updates).
2. Control Statements (keep it simple)
   - Early returns if collections are empty.
   - If/else to detect status changes (Submitted → Approved).
   - For loops only over in-memory lists or query results (no SOQL inside loops).
3. Queueable Apex (async recalculations)
   Create: force-app/main/default/classes/RecalcLeaveBalanceQueueable.cls
   ```
   public with sharing class RecalcLeaveBalanceQueueable implements Queueable {
   private Set<Id> empIds;
   public RecalcLeaveBalanceQueueable(Set<Id> employeeIds) { this.empIds = employeeIds;
   }
   public void execute(QueueableContext ctx) {
   EmployeeService.recalcLeaveBalances(empIds);
   }
   }
   ```
4. Future Methods (webhook to your bot)
   - Requires Named Credential HR_Bot_NC (already set up earlier).

Create: force-app/main/default/classes/HR_Future.cls

```
public with sharing class HR_Future {
@future(callout=true)
public static void sendLeaveApprovedWebhook(Set<Id> leaveIds) {
if (leaveIds == null || leaveIds.isEmpty()) return;
List<Leave__c> leaves = [
SELECT Id, Employee__r.Name, Start_Date__c, End_Date__c, Leave_Type__c
FROM Leave__c WHERE Id IN :leaveIds
];
Http h = new Http();
for (Leave__c l : leaves) {
try {
HttpRequest req = new HttpRequest();
req.setMethod('POST');
req.setEndpoint('callout:HR_Bot_NC/leaveApproved');
req.setHeader('Content-Type','application/json');
req.setBody(JSON.serialize(l));
h.send(req);
} catch (Exception e) {
System.debug('Webhook failed for ' + l.Id + ': ' + e.getMessage());
}
}
}
}
```

1. Batch Apex (nightly self-heal)
   Create: force-app/main/default/classes/LeaveBalanceBatch.cls

   ```
   public with sharing class LeaveBalanceBatch implements Database.Batchable<SObject> {
   public Database.QueryLocator start(Database.BatchableContext bc) {
   return Database.getQueryLocator('SELECT Id FROM Employee__c');
   }
   public void execute(Database.BatchableContext bc, List<SObject> scope) {
   Set<Id> empIds = new Set<Id>();
   for (SObject s : scope) empIds.add((Id)s.get('Id'));
   EmployeeService.recalcLeaveBalances(empIds);
   }
   public void finish(Database.BatchableContext bc) { }
   }
   ```
2. Scheduled Apex (run batch daily)
   Create: force-app/main/default/classes/HR_DailyScheduler.cls

   ```
   public with sharing class HR_DailyScheduler implements Schedulable {
   public void execute(SchedulableContext sc) {
   Database.executeBatch(new LeaveBalanceBatch(), 100);
   }
   }
   ```

Schedule it

- Setup → Apex Classes → Schedule Apex → Job Name: HR Daily
- Class: HR_DailyScheduler → Frequency: Daily → Time: 2:00 AM

1. Exception Handling (minimal but useful)
   Create: force-app/main/default/classes/HRException.cls
   public class HRException extends Exception {}

Guidelines

- Use addError in triggers for validation messages.
- Wrap callouts in try/catch and log. Throw HRException from service layer if needed.

1. Test Data Factory (keep tests clean)
   Create: force-app/main/default/classes/TestDataFactory.cls
   @isTest
   public class TestDataFactory {
   public static Employee__c makeEmployee() {
   Employee__c e = new Employee__c(
   Name = 'EMP-' + String.valueOf(Math.mod(Crypto.getRandomInteger(), 1000000)),
   Annual_Leave_Entitlement__c = 24
   );
   insert e;
   return e;
   }
   public static Leave__c makeLeave(Id empId, Date startD, Date endD, String status) {
   Leave__c l = new Leave__c(
   Employee__c = empId,
   Start_Date__c = startD,
   End_Date__c = endD,
   Approval_Status__c = status,
   Leave_Type__c = 'Planned'
   );
   insert l;
   return l;
   }
   }
2. Test Classes (coverage + asserts)
   Create: force-app/main/default/classes/LeaveTriggerHandlerTest.cls
   @isTest
   private class LeaveTriggerHandlerTest {
   @isTest static void durationAndOverlap() {
   Employee__c e = TestDataFactory.makeEmployee();
   // Existing approved leave
   Leave__c l1 = TestDataFactory.makeLeave(e.Id, Date.today(), Date.today().addDays(1),
   'Approved');
3. text
4. Test.startTest();
5. // Overlap attempt

```
    Leave__c l2 = new Leave__c(Employee__c = e.Id, Start_Date__c = Date.today(), End_Date__c = Date.today().addDays(2),
    Approval_Status__c='Submitted', Leave_Type__c='Sick');
    try {
        insert l2;
        System.assert(false, 'Expected overlap error');
    } catch (DmlException ex) {
        System.assert(ex.getMessage().contains('Overlapping leave'));
    }
    // Non-overlapping, check duration
    Leave__c l3 = new Leave__c(Employee__c = e.Id, Start_Date__c = Date.today().addDays(3), End_Date__c = Date.today().addDays(4),
    Approval_Status__c='Submitted', Leave_Type__c='Planned');
    insert l3;
    l3 = [SELECT Duration_Days__c FROM Leave__c WHERE Id = :l3.Id];
    System.assertEquals(2, Integer.valueOf(l3.Duration_Days__c));
    Test.stopTest();
}
@isTest static void approvalTriggersRecalcAndWebhook() {
    Employee__c e = TestDataFactory.makeEmployee();
    Leave__c l = TestDataFactory.makeLeave(e.Id, Date.today(), Date.today(), 'Submitted');
text
    Test.startTest();
    // Approve the leave
    l.Approval_Status__c = 'Approved';
    update l;
    Test.stopTest();

    // Balance should be entitlement - used(1)
    e = [SELECT Available_Leave_Balance__c, Annual_Leave_Entitlement__c FROM Employee__c WHERE Id = :e.Id];
    System.assertEquals(e.Annual_Leave_Entitlement__c - 1, e.Available_Leave_Balance__c);
}

}
```

Create: force-app/main/default/classes/AttendanceTriggerHandlerTest.cls

```
@isTest
private class AttendanceTriggerHandlerTest {
@isTest static void preventDuplicatesAndFutureDate() {
Employee__c e = TestDataFactory.makeEmployee();
Attendance__c a1 = new Attendance__c(Employee__c = e.Id, Attendance_Date__c = Date.today());
insert a1;
```

text

```
  Test.startTest();

  Attendance__c dup = new Attendance__c(Employee__c = e.Id, Attendance_Date__c = Date.today());

  try { insert dup; System.assert(false); } catch (DmlException ex) { System.assert(ex.getMessage().contains('already exists')); }



  Attendance__c future = new Attendance__c(Employee__c = e.Id, Attendance_Date__c = Date.today().addDays(1));
```

```
    try { insert future; System.assert(false); } catch (DmlException ex) { System.assert(ex.getMessage().toLowerCase().contains('future')); }

    Test.stopTest();

}

}
```

Create: force-app/main/default/classes/AsyncTests.cls
@isTest
private class AsyncTests {
@isTest static void batchRecalc() {
Employee__c e = TestDataFactory.makeEmployee();
TestDataFactory.makeLeave(e.Id, Date.today(), Date.today(), 'Approved');
TestDataFactory.makeLeave(e.Id, Date.today().addDays(1), Date.today().addDays(2), 'Approved'); // 3 days total

text

```
    Test.startTest();

    Database.executeBatch(new LeaveBalanceBatch(), 50);

    Test.stopTest();



    e = [SELECT Available_Leave_Balance__c FROM Employee__c WHERE Id = :e.Id];

    System.assertEquals(21, e.Available_Leave_Balance__c);

}



@isTest

static void scheduleRuns() {

    String cron = '0 0 2 * * ?'; // 2 AM daily

    Test.startTest();

    System.schedule('Test HR Daily', cron, new HR_DailyScheduler());

    Test.stopTest();

}
```

```
}
```

Create: force-app/main/default/classes/HR_HttpMock.cls

```
@isTest
global class HR_HttpMock implements HttpCalloutMock {
global HTTPResponse respond(HTTPRequest req) {
HttpResponse res = new HttpResponse();
res.setStatusCode(200);
res.setBody('ok');
return res;
}
}
```

Create: force-app/main/default/classes/FutureAndSOSLTests.cls

```
@isTest
private class FutureAndSOSLTests {
@isTest static void futureWebhook() {
Test.setMock(HttpCalloutMock.class, new HR_HttpMock());
Employee__c e = TestDataFactory.makeEmployee();
Leave__c l = TestDataFactory.makeLeave(e.Id, Date.today(), Date.today(), 'Approved');
```

text

```
  Test.startTest();

  HR_Future.sendLeaveApprovedWebhook(new Set<Id>{l.Id});

  Test.stopTest();

}
```

```
@isTest static void soslSearch() {

  Employee__c e = new Employee__c(Name = 'EMPX-12345');

  insert e;

  Test.startTest();

  Test.setFixedSearchResults(new List<Id>{ e.Id }); // ensure deterministic SOSL result

  List<Employee__c> res = EmployeeSearchService.searchEmployees('EMPX-12345');

  Test.stopTest();

  System.assertEquals(1, res.size());
```

```
}

}
```

1. Asynchronous Processing (how to verify)

- Deploy:
  - sf project deploy start --target-org MySandbox --source-dir force-app
- Run tests:
  - sf apex test run --target-org MySandbox --wait 20 --result-format human
- Create a Leave__c in UI:
  - Overlapping dates → should block with error.
  - Approve a leave → Available_Leave_Balance__c decreases; Apex Jobs shows Queueable ran.
- Attendance__c in UI:
  - Same date twice for same employee → blocks; future date → blocks.
- Monitor:
  - Setup → Apex Jobs (Queueable/Batch results)
  - Setup → Scheduled Jobs (scheduler)
  - Debug Logs for webhook callouts