Phase 5: Apex Programming (Developer)

---

1. Classes & Objects

Theory:
In Apex, classes are blueprints that define objects. They can contain variables (fields), methods, and constructors. Objects are instances of these classes. This allows Salesforce developers to encapsulate business logic and reuse it across the org.

In my project:

- JobApplicationController.cls → handles UI logic for job applications.
- JobController.cls → handles job object operations.
- TaxCalculation.cls → encapsulates tax calculation logic.
- EventReminder.cls → defines logic for event reminders.

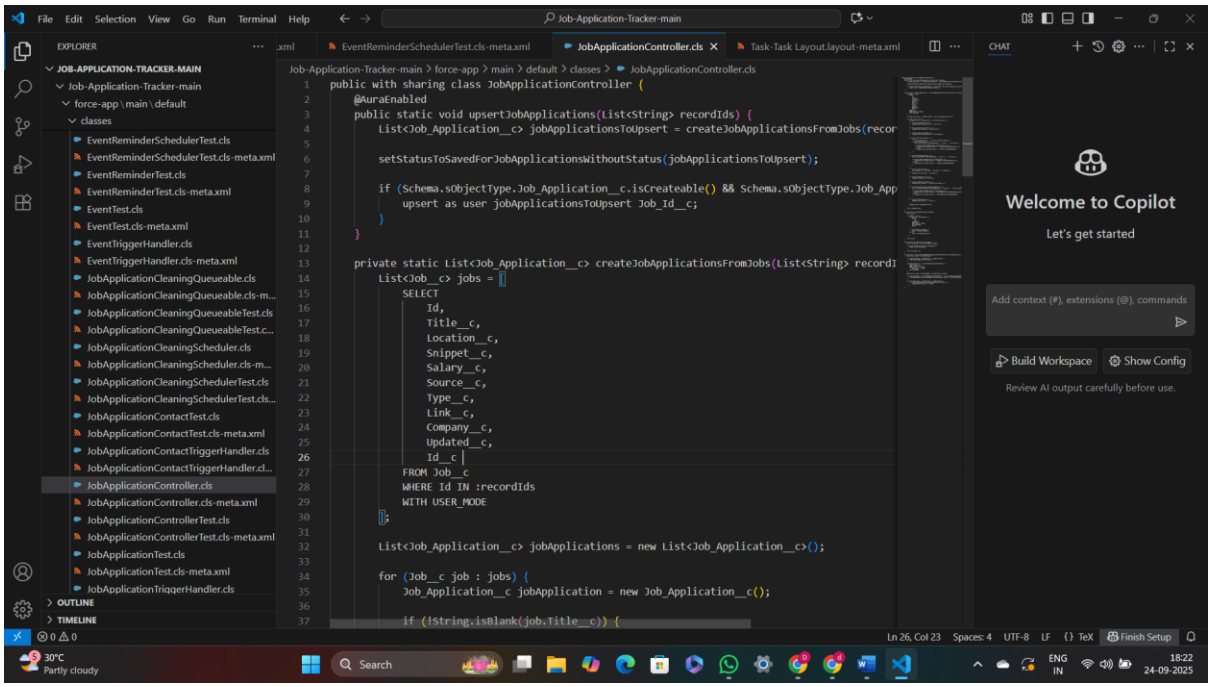
Control Statements

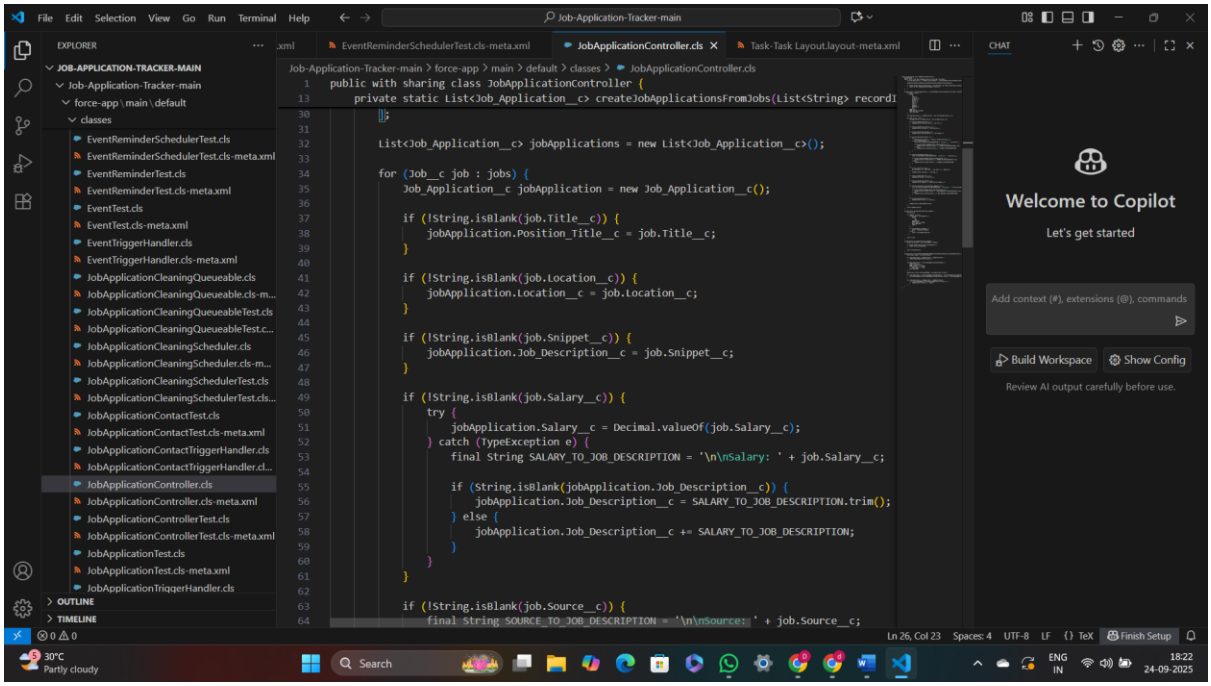
Theory:
Apex supports control statements like if-else, for, while, switch, which help in conditional and iterative logic.

In your project:

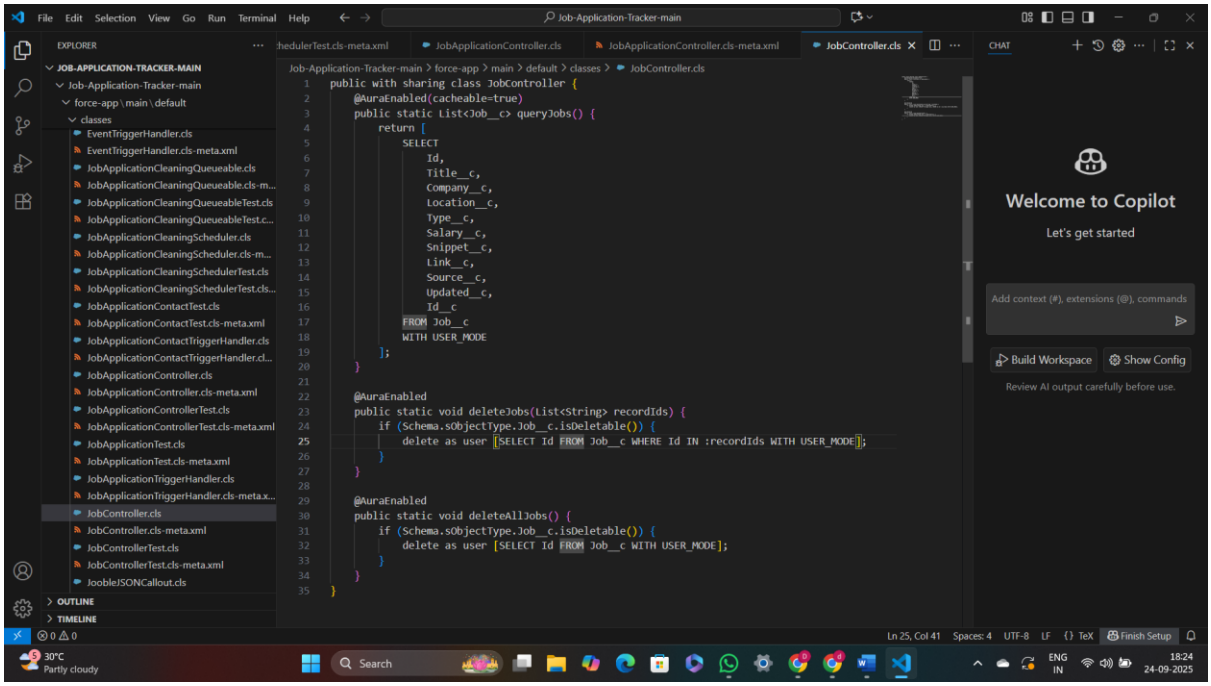
- TaxCalculation.cls → uses if-else to calculate taxes.

JobApplicationController.cls:



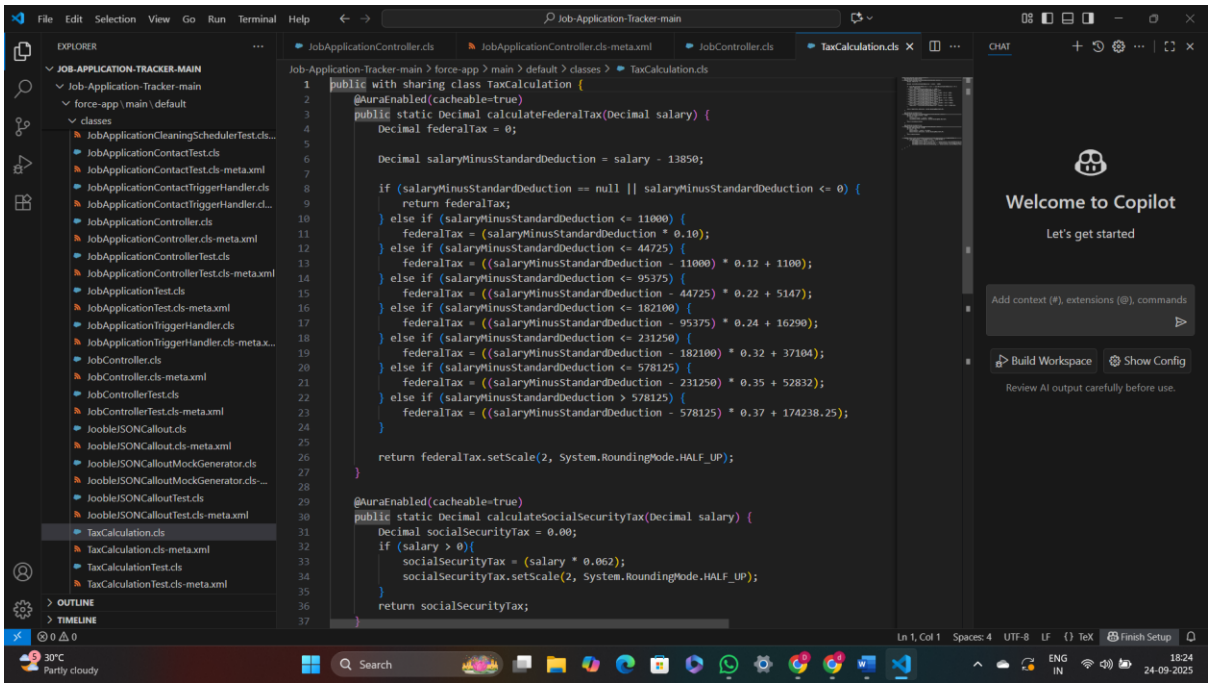


JobController.cls:



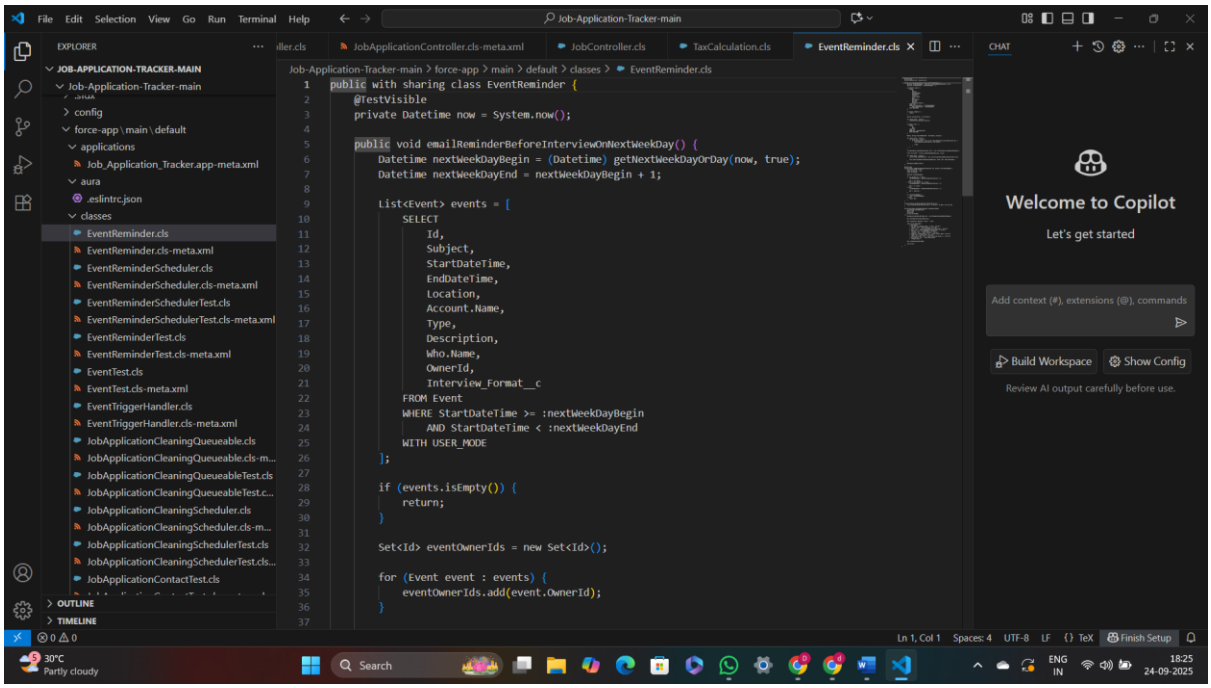
```
public with sharing class JobController {
    @AuraEnabled(cacheable=true)
    public static List<Job__c> queryJobs() {
        return [
            SELECT
                Id,
                Title__c,
                Company__c,
                Location__c,
                Type__c,
                Salary__c,
                Snippet__c,
                Link__c,
                Source__c,
                Updated__c,
                Id__c
            FROM Job__c
            WITH USER_MODE
        ];
    }

    @AuraEnabled
    public static void deleteJobs(List<String> recordIds) {
        if (Schema.sObjectType.Job__c.isDeletable()) {
            delete as user [SELECT Id FROM Job__c WHERE Id IN :recordIds WITH USER_MODE];
        }
    }

    @AuraEnabled
    public static void deleteAllJobs() {
        if (Schema.sObjectType.Job__c.isDeletable()) {
            delete as user [SELECT Id FROM Job__c WITH USER_MODE];
        }
    }
}
```

TaxCalculation.cls:



```
public with sharing class TaxCalculation {
    @AuraEnabled(cacheable=true)
    public static Decimal calculateFederalTax(Decimal salary) {
        Decimal federalTax = 0;

        Decimal salaryMinusStandardDeduction = salary - 13850;

        if (salaryMinusStandardDeduction == null || salaryMinusStandardDeduction <= 0) {
            return federalTax;
        } else if (salaryMinusStandardDeduction <= 11000) {
            federalTax = (salaryMinusStandardDeduction * 0.10);
        } else if (salaryMinusStandardDeduction <= 44725) {
            federalTax = ((salaryMinusStandardDeduction - 11000) * 0.12 + 1100);
        } else if (salaryMinusStandardDeduction <= 95375) {
            federalTax = ((salaryMinusStandardDeduction - 44725) * 0.22 + 5147);
        } else if (salaryMinusStandardDeduction <= 182100) {
            federalTax = ((salaryMinusStandardDeduction - 95375) * 0.24 + 16290);
        } else if (salaryMinusStandardDeduction <= 231250) {
            federalTax = ((salaryMinusStandardDeduction - 182100) * 0.32 + 37104);
        } else if (salaryMinusStandardDeduction <= 578125) {
            federalTax = ((salaryMinusStandardDeduction - 231250) * 0.35 + 52832);
        } else if (salaryMinusStandardDeduction > 578125) {
            federalTax = ((salaryMinusStandardDeduction - 578125) * 0.37 + 174238.25);
        }

        return federalTax.setScale(2, System.RoundingMode.HALF_UP);
    }

    @AuraEnabled(cacheable=true)
    public static Decimal calculateSocialSecurityTax(Decimal salary) {
        Decimal socialSecurityTax = 0.00;
        if (salary > 0){
            socialSecurityTax = (salary * 0.062);
            socialSecurityTax.setScale(2, System.RoundingMode.HALF_UP);
        }
        return socialSecurityTax;
    }
```

EventReminder.cls:



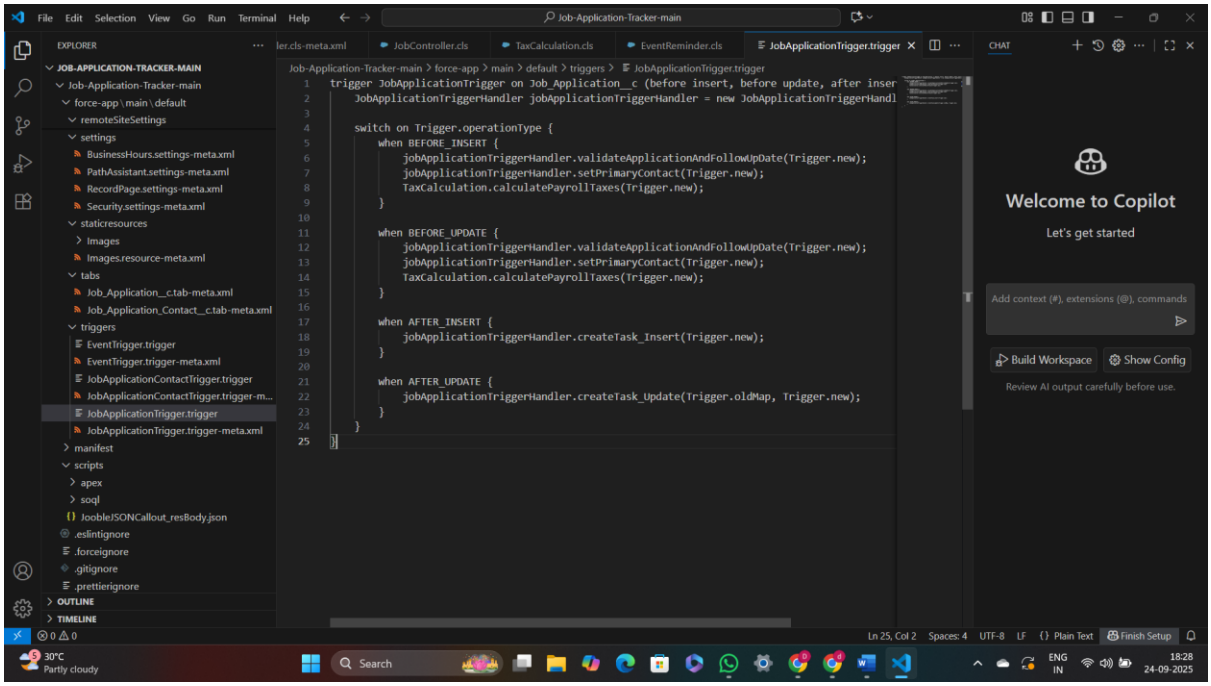
2. Apex Triggers (before/after insert/update/delete)

Theory:
Triggers are pieces of Apex code that run before or after records are inserted, updated, deleted, or undeleted. They are used to enforce business rules or automate processes.

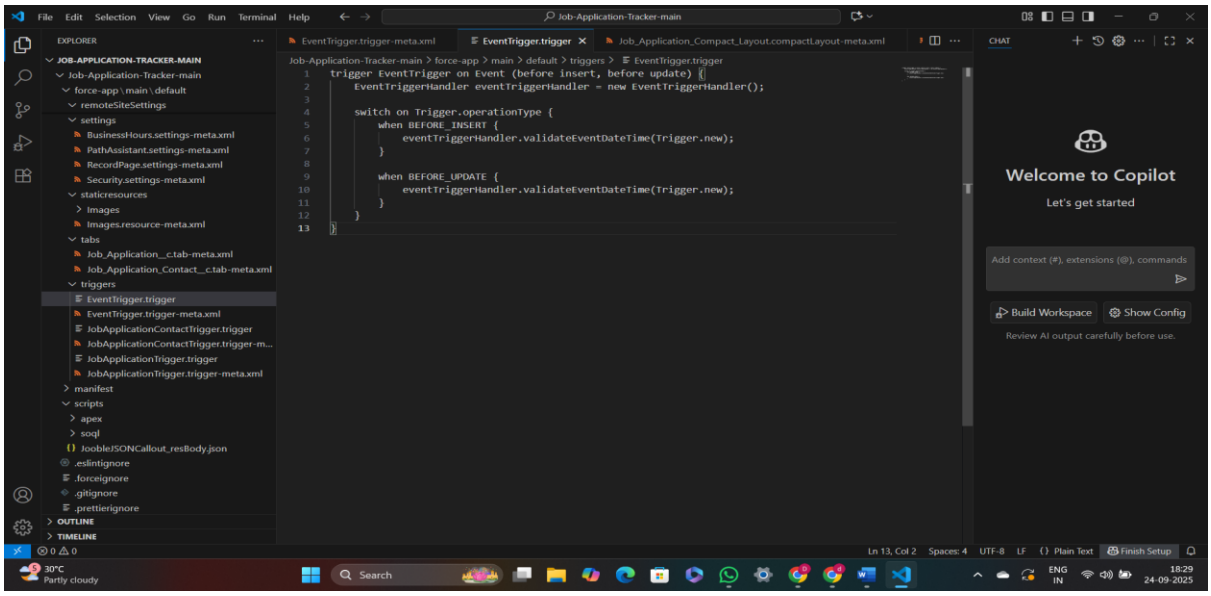
In your project:

- JobApplicationTrigger.trigger → fires on job applications.
- EventTrigger.trigger → fires on events.
- JobApplicationContactTrigger.trigger → fires on job application contacts.

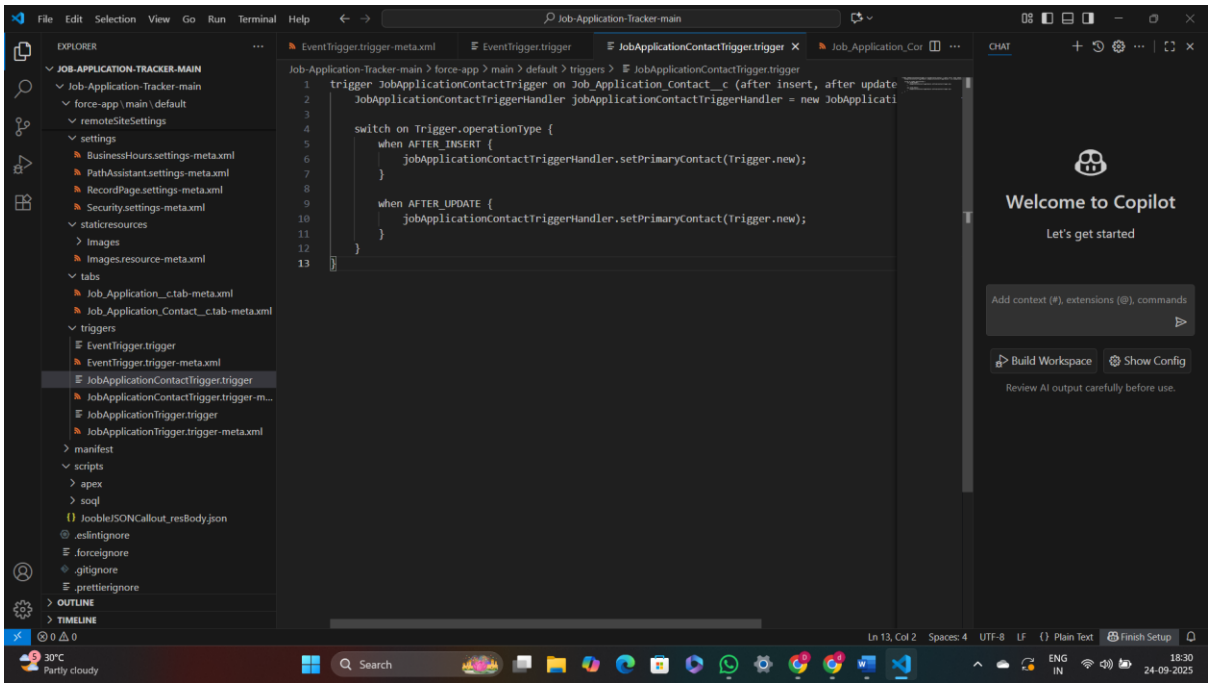
JobApplicationTrigger.trigger:



EventTrigger.trigger:



JobApplicationContactTrigger.trigger:

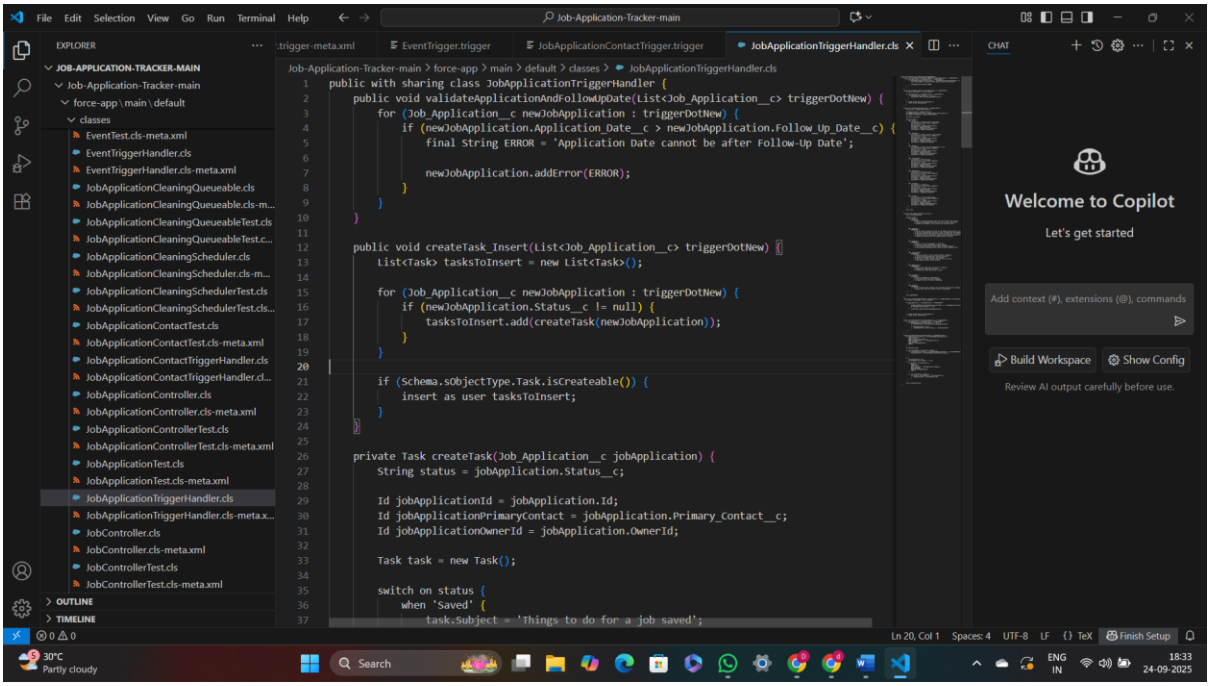


3. Trigger Design Pattern

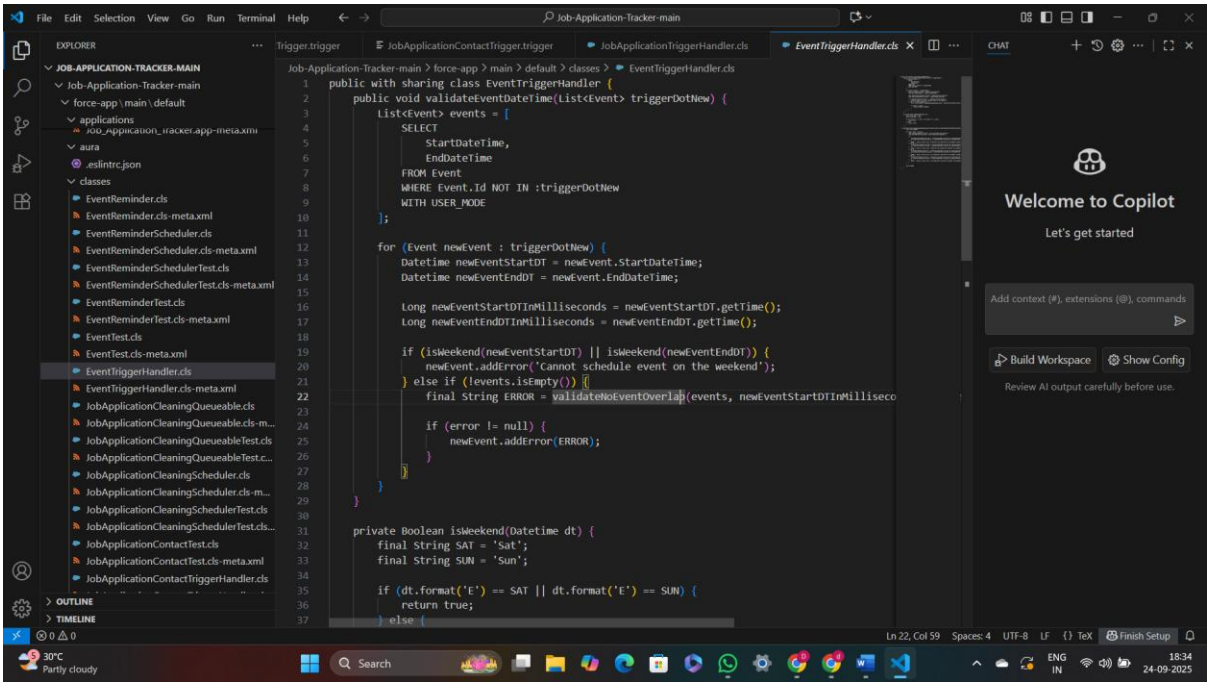
Theory:
To avoid bulky triggers, the Trigger Handler Pattern is used. Each trigger delegates logic to a handler class, keeping triggers clean and single-purposed.

In your project:

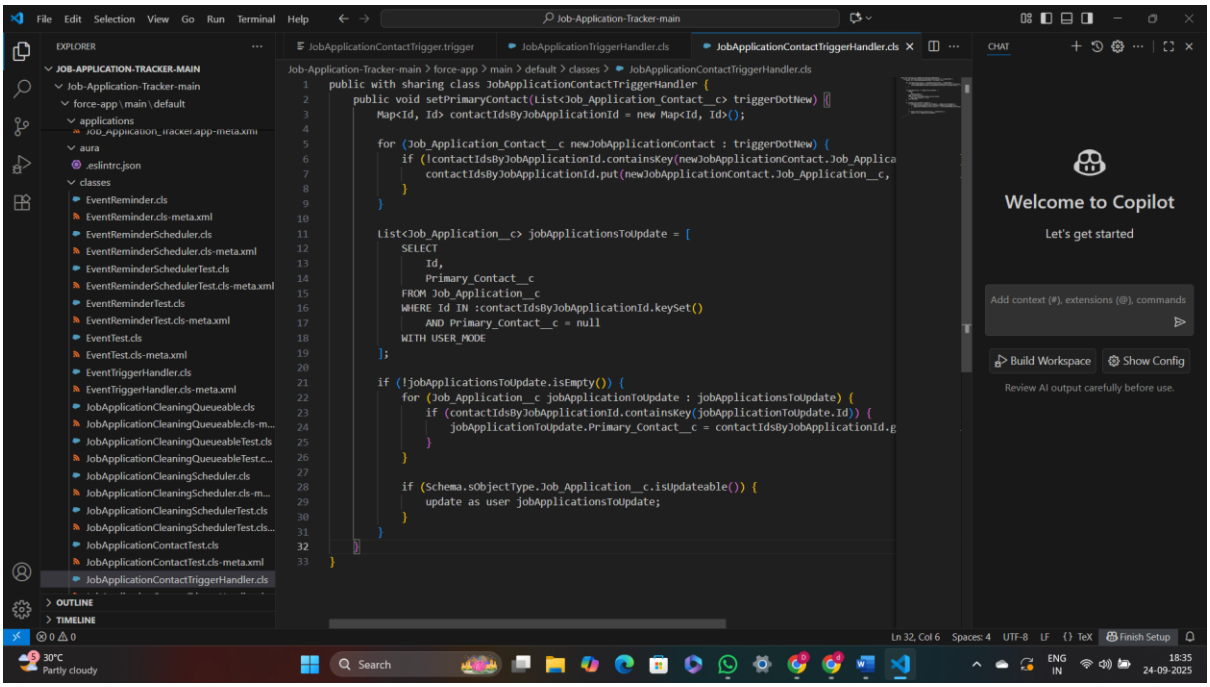
- **JobApplicationTriggerHandler.cls**
- **EventTriggerHandler.cls**
- **JobApplicationContactTriggerHandler.cls**



EventTriggerHandler.cls:



JobApplicationContactTriggerHandler.cls:



4. SOQL & SOSL

Theory:

- **SOQL (Salesforce Object Query Language): Used to fetch records from Salesforce objects (similar to SQL SELECT).**
- **SOSL (Salesforce Object Search Language): Used for searching text across multiple objects/fields.**

In your project:

- **Queries written in Apex classes like JobController.cls.**
- **Standalone query files in 📁 scripts/soql:**
  - **Job_Application__c.soql**
  - **Job__c.soql**
  - **account.soql**

Job_Application__c.soql: (Query that I had given)

SELECT Name, Status__c, Application_Date__c, Follow_Up_Date__c, Notes__c, Rating__c, Position_Title__c, Work_Model__c, Location__c, Job_Type__c, Job_Description__c, Salary__c, Take_Home_Pay__c, URL__c, Job_Id__c, Company__c, Primary_Contact__c, CreatedDate FROM Job_Application__c ORDER BY CreatedDate ASC

Job__c.soql:

SELECT Id, Title__c, Location__c, Snippet__c, Salary__c, Source__c, Type__c, Link__c, Company__c, Updated__c, Id__c, CreatedDate FROM Job__c ORDER BY CreatedDate ASC

account.soql:

SELECT Id, Name FROM Account

5. Collections: List, Set, Map

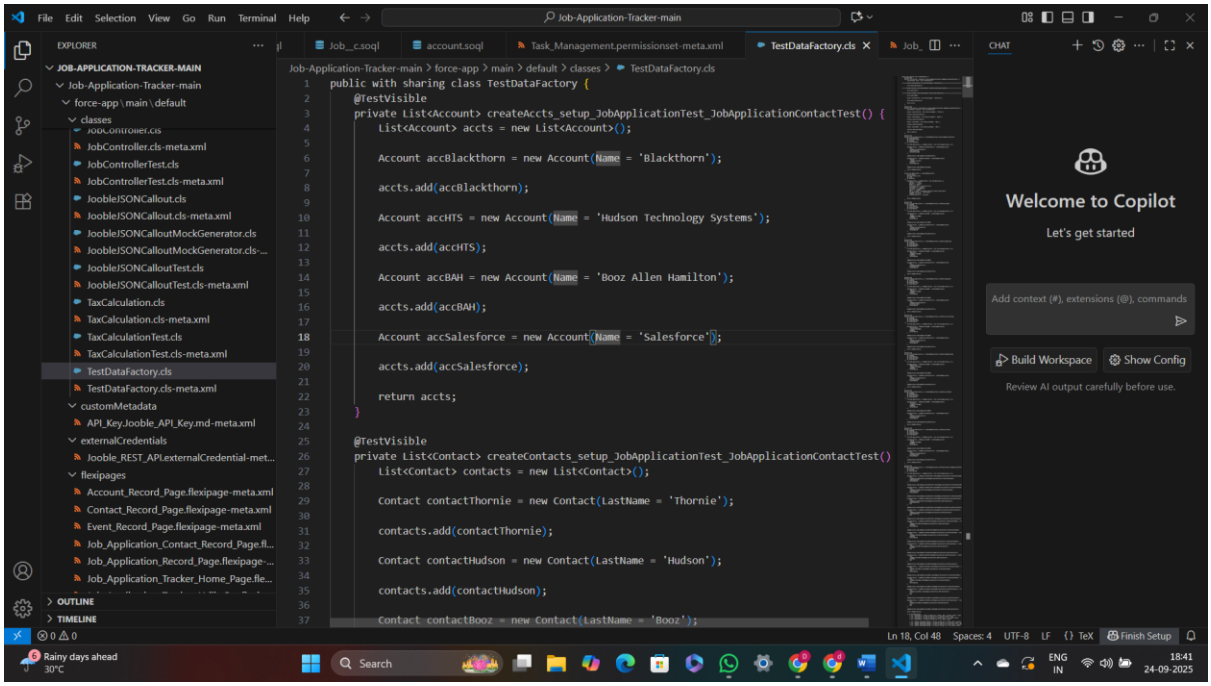
Theory:

- **List** → Ordered collection of elements (can contain duplicates).
- **Set** → Unordered collection of unique elements.
- **Map** → Key-value pairs.
  Collections make bulk processing efficient in Salesforce.

In your project:

- TestDataFactory.cls → creates bulk test data using lists and maps.
- JobApplicationController.cls & JobController.cls → use collections to store query results (List<Job__c>).

TestDataFactory.cls:



7. Batch Apex

Theory:
Batch Apex processes large data volumes asynchronously in smaller chunks (batches), respecting governor limits.

In your project:

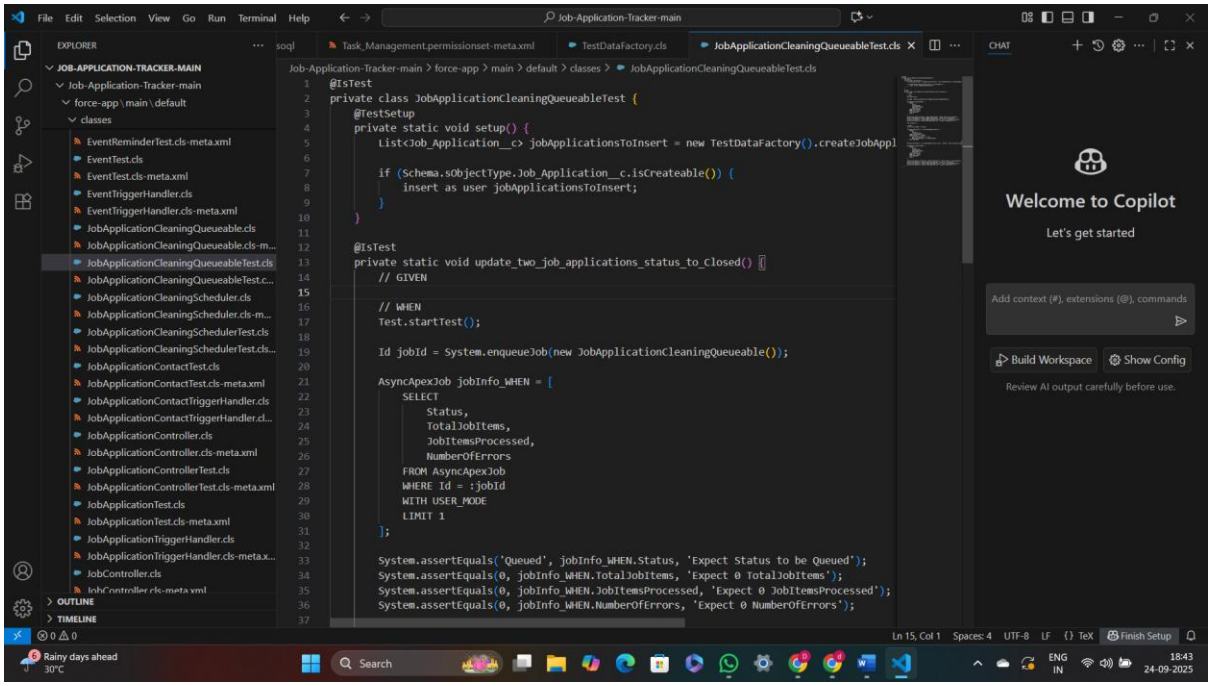
- **JobApplicationCleaningQueueable.cls (used like batch/queueable for cleaning old job applications).**

Queueable Apex

Theory:
Queueable Apex allows running jobs asynchronously, and supports job chaining for more flexibility than @future.

In your project:

- **JobApplicationCleaningQueueable.cls → runs cleaning logic asynchronously.**


File Edit Selection View Go Run Terminal Help
Job-Application-Tracker-main
EXPLORER
JOB-APPLICATION-TRACKER-MAIN
Job-Application-Tracker-main
force-app \main \default
classes
EventReminderTest.cls-meta.xml
EventTest.cls
EventTest.cls-meta.xml
EventTriggerHandler.cls
EventTriggerHandler.cls-meta.xml
JobApplicationCleaningQueueable.cls
JobApplicationCleaningQueueable.cls-m...
JobApplicationCleaningQueueableTest.cls
JobApplicationCleaningQueueableTest.c...
JobApplicationCleaningScheduler.cls
JobApplicationCleaningScheduler.cls-m...
JobApplicationCleaningSchedulerTest.cls
JobApplicationCleaningSchedulerTest.cls...
JobApplicationContactTest.cls
JobApplicationContactTest.cls-meta.xml
JobApplicationContactTriggerHandler.cls
JobApplicationContactTriggerHandler.cl...
JobApplicationController.cls
JobApplicationController.cls-meta.xml
JobApplicationControllerTest.cls
JobApplicationControllerTest.cls-meta.xml
JobApplicationTest.cls
JobApplicationTest.cls-meta.xml
JobApplicationTriggerHandler.cls
JobApplicationTriggerHandler.cls-meta.x...
JobController.cls
OUTLINE
TIMELINE
soql
Task_Management.permissionset-meta.xml
TestDataFactory.cls
JobApplicationCleaningQueueableTest.cls
Job-Application-Tracker-main > force-app > main > default > classes > JobApplicationCleaningQueueableTest.cls
@IsTest
private class JobApplicationCleaningQueueableTest {
@TestSetup
private static void setup() {
List<Job_Application__c> jobApplicationsToInsert = new TestDataFactory().createJobAppl
if (Schema.sObjectType.Job_Application__c.isCreateable()) {
insert as user jobApplicationsToInsert;
}
}
@IsTest
private static void update_two_job_applications_status_to_Closed() {
// GIVEN
// WHEN
Test.startTest();
Id jobId = System.enqueueJob(new JobApplicationCleaningQueueable());
AsyncApexJob jobInfo_WHEN = [
SELECT
Status,
TotalJobItems,
JobItemsProcessed,
NumberOfErrors
FROM AsyncApexJob
WHERE Id = :jobId
WITH USER_MODE
LIMIT 1
];
System.assertEquals('Queued', jobInfo_WHEN.Status, 'Expect Status to be Queued');
System.assertEquals(0, jobInfo_WHEN.TotalJobItems, 'Expect 0 TotalJobItems');
System.assertEquals(0, jobInfo_WHEN.JobItemsProcessed, 'Expect 0 JobItemsProcessed');
System.assertEquals(0, jobInfo_WHEN.NumberOfErrors, 'Expect 0 NumberOfErrors');
CHAT
Welcome to Copilot
Let's get started
Add context (#), extensions (@), commands
Build Workspace
Show Config
Review AI output carefully before use.
Ln 15, Col 1
Spaces: 4
UTF-8
LF
TeX
Finish Setup
Rainy days ahead
30°C
Search
ENG
IN
18:43
24-09-2025