Phase 5 — Apex Programming (Developer)

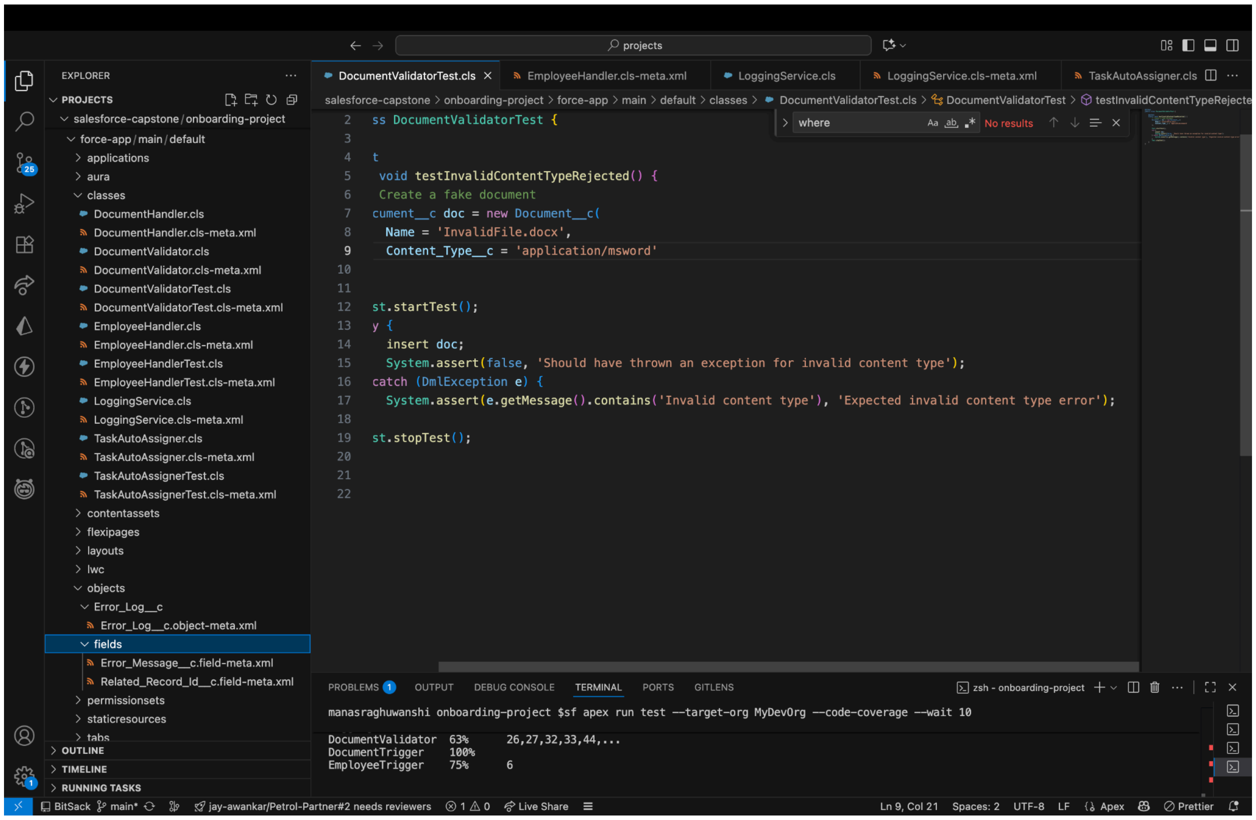
Goal: Implement complex business logic that can't be achieved with declarative tools using Apex classes, triggers, and test classes. Ensure code is well-tested (>= 75% coverage) and bulk-safe.

1. **Plan the Apex work**
   - Identify logic that needs Apex: complex validations (file format checks), batch processes, transaction control, callouts to external systems.
   - Draft method signatures and responsibilities (e.g., `DocumentValidator.validateFormat(documentId)`, `TaskAutoAssigner.assignTasksForEmployee(employeeId)`).

2. **Set up your dev environment**
   - Use VS Code with Salesforce Extensions Pack (SFDX) or the Developer Console for quick edits.
   - Authorize your Dev Hub / scratch org or Developer Org: `sfdx auth:web:login`.

3. **Create Apex classes (bulk-safe)**
   - `DocumentValidator` (static utility)
     - Responsibility: validate file types, sizes, and required metadata.
     - Example methods: `validateContentType(Blob fileBody, String contentType)`, `validateFileSize(Integer bytes)`.
   - `TaskAutoAssigner` (service class)
     - Responsibility: create OnboardingTask__c records for a given employee in bulk.
     - Use `@future(callout=true)` only when making callouts; prefer Queueable for asynchronous processing.
   - `EmployeeHelper` (domain/service layer)
     - Centralize operations like setting default fields and orchestrating task creation.

4. **Create Triggers (thin triggers, logic in classes)**
   - `Trigger EmployeeTrigger on Employee__c (after insert, after update)`
     - Keep trigger logic minimal; call `EmployeeHelper.handleAfterInsert(Trigger.new)`.
   - `Trigger DocumentTrigger on Document__c (before insert, before update)`
     - Call `DocumentValidator` to validate files and set status fields.



```
ss DocumentValidatorTest {

t
 void testInvalidContentTypeRejected() {
 Create a fake document
cument__c doc = new Document__c(
  Name = 'InvalidFile.docx',
  Content_Type__c = 'application/msword'


st.startTest();
y {
  insert doc;
  System.assert(false, 'Should have thrown an exception for invalid content type');
catch (DmlException e) {
  System.assert(e.getMessage().contains('Invalid content type'), 'Expected invalid content type error');

st.stopTest();
```

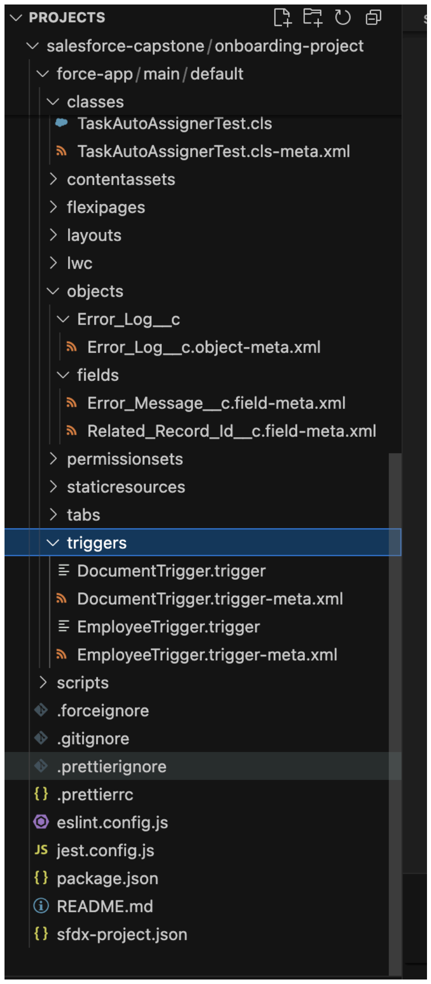
```
manasraghuwanshi onboarding-project $sf apex run test --target-org MyDevOrg --code-coverage --wait 10

DocumentValidator  63%      26,27,32,33,44,...
DocumentTrigger    100%
EmployeeTrigger    75%      6
```

5. **Write Test Classes**
   - Create comprehensive tests for each Apex class and trigger.
   - Use `@IsTest` classes and `Test.startTest()` / `Test.stopTest()` blocks.
   - Assert bulk behavior by inserting 200 records in a single test to ensure governor limits are respected.
   - Aim for **>75% overall coverage**, but test meaningful behavior (asserts) — high coverage alone is not enough.

6. **Run static analysis & code quality checks**
   - Use PMD for Apex or SonarQube if available.
   - Run `sfdx force:apex:test:run` and fix failing tests.

7. **Deploy and monitor**
   - Deploy to Sandbox first using change sets or SFDX. Run all tests in target org.
   - Monitor logs (Debug Logs) for unexpected exceptions.



```
PROJECTS
  salesforce-capstone / onboarding-project
    force-app / main / default
      classes
        TaskAutoAssignerTest.cls
        TaskAutoAssignerTest.cls-meta.xml
      contentassets
      flexipages
      layouts
      lwc
      objects
        Error_Log__c
          Error_Log__c.object-meta.xml
          fields
            Error_Message__c.field-meta.xml
            Related_Record_Id__c.field-meta.xml
      permissionsets
      staticresources
      tabs
      triggers
        DocumentTrigger.trigger
        DocumentTrigger.trigger-meta.xml
        EmployeeTrigger.trigger
        EmployeeTrigger.trigger-meta.xml
    scripts
    .forceignore
    .gitignore
    .prettierignore
    .prettierrc
    eslint.config.js
    jest.config.js
    package.json
    README.md
    sfdx-project.json
```