Phase 5: Apex Programming

Introduction

Declarative tools like Flows and Process Builder provide a great deal of automation within Salesforce. However, some scenarios require more complex logic, granular control over data, or large-scale processing that goes beyond what declarative tools can handle. In such cases, Salesforce developers use **Apex**, a strongly-typed, object-oriented programming language designed specifically for the Salesforce platform.

Phase 5 of the ApexHub project focuses on implementing custom automation through Apex Classes, Triggers, Batch Apex, and Test Classes. This ensures that business processes can be extended, optimized, and scaled in ways that declarative automation cannot achieve.

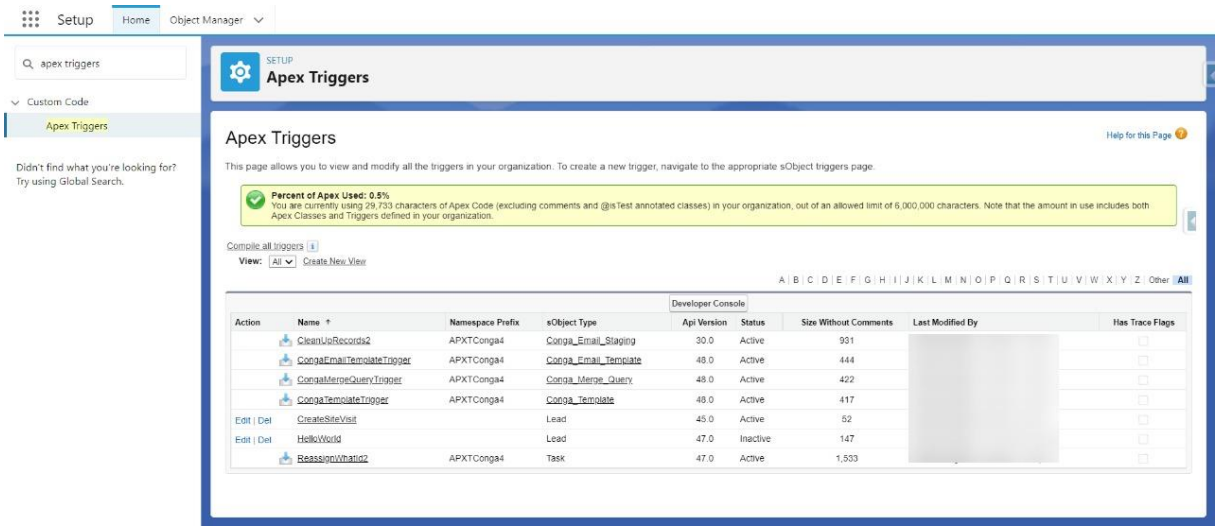
Objectives

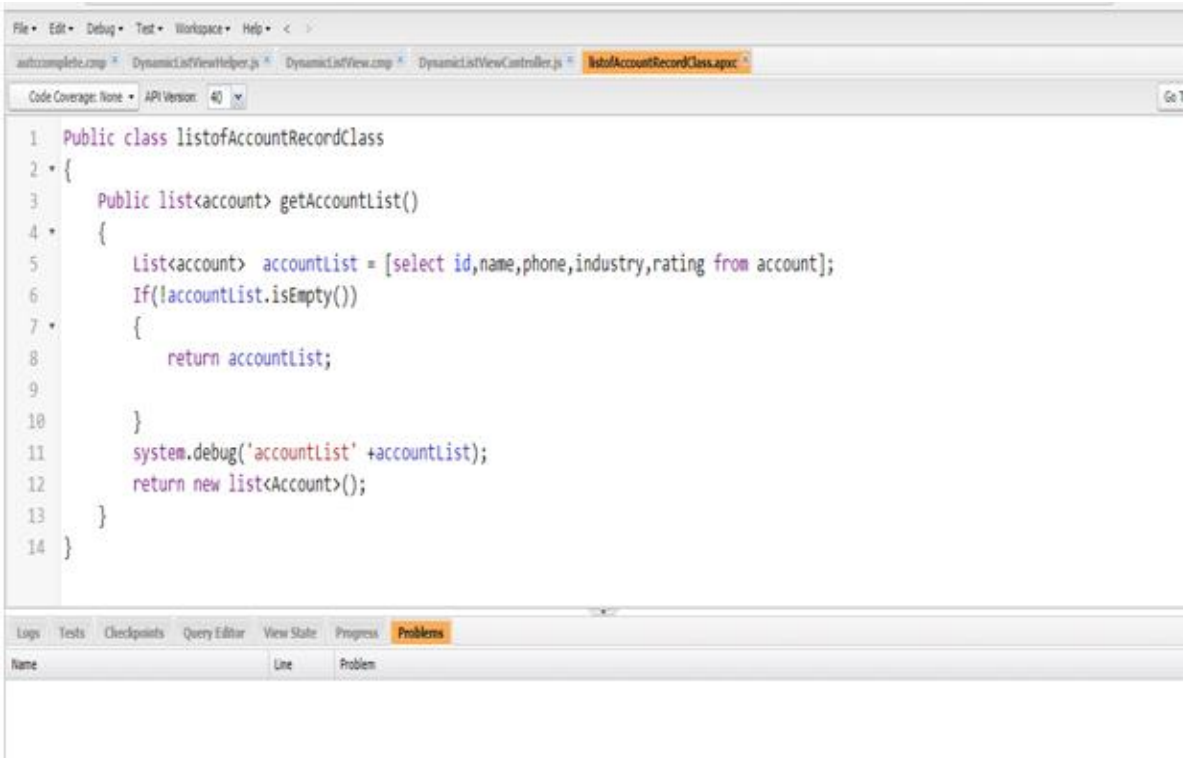
1. Handle complex logic and scenarios not supported by declarative tools.
2. Implement reusable Apex Classes for centralized business logic.
3. Create Triggers for event-driven automation on standard and custom objects.
4. Use SOQL/SOSL queries and DML operations efficiently without breaching governor limits.
5. Follow bulk-processing best practices to ensure scalability.
6. Implement robust error-handling mechanisms using try-catch and custom exceptions.
7. Develop Test Classes with at least 75% code coverage to meet deployment requirements.

Activities

1. Apex Classes & Triggers

- **Purpose:** Automate record creation and updates when certain events occur.
- **Example:**
    - Trigger: When an **Opportunity is Closed Won**, automatically create an **Invoice__c** or **Project__c** record.
    - Class: Centralize logic for calculating **loyalty points** for customers.
- **Steps Taken:**
    - Created a single trigger per object (Trigger Framework).
    - Separated logic into reusable classes for maintainability.





2. SOQL & SOSL

- **Purpose:** Query Salesforce data programmatically.
- **Examples:**
  - SOQL: Retrieve all invoices related to an account.
  - SOSL: Search for recipes or projects using a keyword across multiple objects.
- **Best Practices:**
  - Used selective queries with filters.
  - Limited records retrieved to stay within governor limits.

3. DML Operations

- **Purpose:** Perform insert, update, delete, and upsert operations on records.
- **Examples:**
    - Insert new Project or Invoice records automatically.
    - Update Opportunity status fields in bulk.
- **Best Practices:**
    - Avoided DML inside loops.
    - Grouped operations using collections (List, Set, Map).

4. Bulk Processing & Best Practices

- **Purpose:** Ensure scalability and performance.
- **Examples:**
    - Bulkified triggers to handle multiple records.
    - Implemented Batch Apex for mass updates (e.g., **updating product prices** for thousands of records).

```
global class MyBatchClass implements Database.Batchable<sObject> {
    global (Database.QueryLocator | Iterable<sObject>) start(Database.BatchableContext bc) {
        // collect the batches of records or objects to be passed to execute
    }
    global void execute(Database.BatchableContext bc, List<P> records){
        // process each batch of records
    }
    global void finish(Database.BatchableContext bc){
        // execute any post-processing operations
    }
}
```

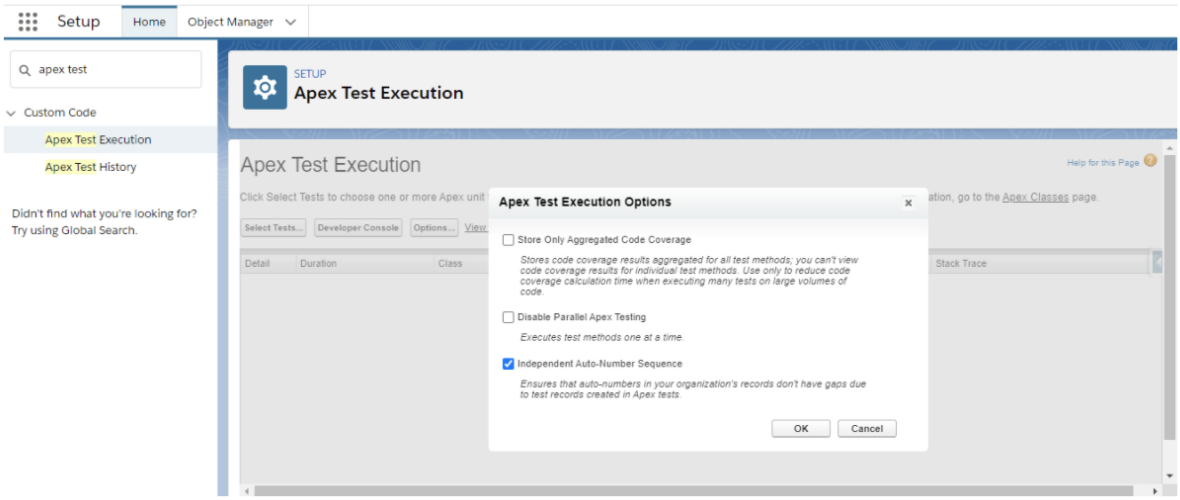
5. Error Handling

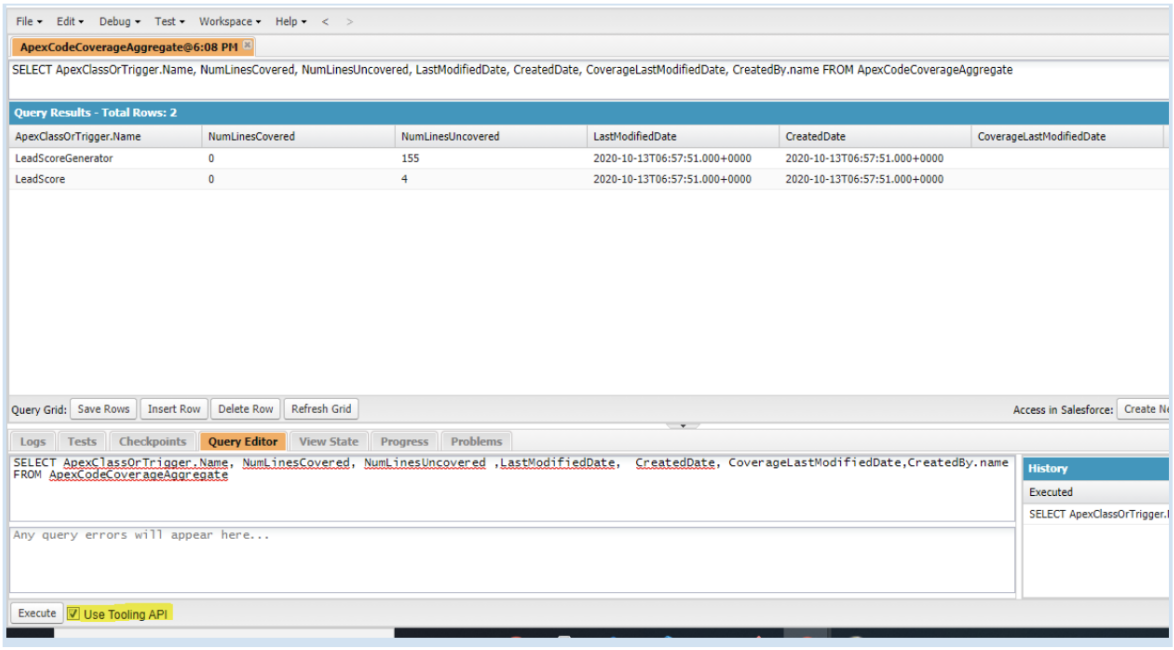
- **Purpose:** Prevent runtime failures and provide clear messages.
- **Examples:**
    - Used **try-catch blocks** to handle DML exceptions.
    - Created **custom exceptions** for recipe execution failures.

6. Test Classes

- **Purpose:** Validate logic and ensure Apex code meets deployment requirements.
- **Key Points:**

- Created test classes with positive, negative, and bulk test scenarios.
- Used System.assert() to validate expected results.
- Achieved **75%+ code coverage** for all Apex components.





Deliverables

✓ **Apex Trigger** – Auto-create **Invoice__c** record when Opportunity closes.
✓ **Apex Class** – Business logic for calculating **loyalty points**.
✓ **Batch Apex Job** – Large data update (e.g., product mass price update).
✓ **Test Class** – Test coverage with assert statements to validate outcomes.

Expected Outcomes

- Business processes extended beyond declarative automation.

- Reusable and scalable Apex logic to support the ApexHub project.
- Reliable system behavior under bulk data scenarios.
- Compliance with Salesforce deployment requirement of **75%+ test coverage**.
- Reduced risk of governor limit errors due to bulk-safe and optimized code.