**Phase 5: Apex Programming (Developer)**

**Classes & Objects**

We created custom **Apex Classes** to encapsulate business logic.

**Example:**

* **DonationHandler.cls** → Handles logic for donation insert/update.
* **BadgeService.cls** → Calculates and assigns badges based on loyalty points.

**Benefit:** Separating logic into classes ensures **reusability and clean architecture**.

**Apex Triggers (Before/After Insert/Update/Delete)**

**Key Triggers Implemented:**

1. **DonationTrigger**
   * *After Insert:* Update Donor’s total contributions and loyalty points.
   * *After Update:* If donation status changes to *Completed*, recalculate totals.
2. **CampaignTrigger**
   * *After Insert/Update:* Check if Amount Raised >= Goal → update status to *Completed*.

**Benefit:** Maintains **real-time accuracy** of donation totals and campaign progress.

**Trigger Design Pattern**

To avoid bulky triggers, we followed the **Trigger Handler Pattern**:

* One trigger per object → delegates logic to a handler class.
* Example: DonationTrigger → calls DonationHandler.onAfterInsert().

**Benefit:** Improves **readability, scalability, and testability**.

**SOQL & SOSL**

* **SOQL (Salesforce Object Query Language):**  
  Used in DonationHandler to fetch donor’s existing contributions.
* Donor d = [SELECT Id, Total\_Contributions\_\_c FROM Donor\_\_c WHERE Id = :donorId];
* **SOSL (Salesforce Object Search Language):**  
  Used for donor search by email/phone.
* List<List<sObject>> results = [FIND 'john@example.com' IN ALL FIELDS RETURNING Contact(Id, Name)];

**Benefit:** Efficiently fetches records, supporting **donor and campaign management**.

**Collections (List, Set, Map)**

* **List:** Store multiple donations for bulk updates.
* **Set:** Store unique donor IDs to avoid duplicate queries.
* **Map:** Map Donor ID → Total Contributions for quick lookups.

**Benefit:** Optimizes performance by handling bulk records gracefully.

**Control Statements**

Applied **if-else, loops, and switch cases** in handler classes.

**Example:**

if(donation.Status\_\_c == 'Completed') {

updateLoyaltyPoints(donation.Donor\_\_c, donation.Amount\_\_c);

}

**Benefit:** Provides **flexible decision-making logic** for real scenarios.

**Batch Apex**

Used for **large-scale data recalculations**.

**Example:**

* Batch job to **recalculate loyalty points** for all donors at the end of the fiscal year.

**Benefit:** Ensures **data consistency** even with thousands of records.

**Queueable Apex**

Used for **background operations that need chaining**.

**Example:**

* When a donation is completed, queueable job sends donor details to an external CSR portal.

**Benefit:** Handles **asynchronous tasks** without blocking user actions.

**Scheduled Apex**

**Use Case:** Monthly scheduled job to generate **Campaign Summary Reports** for staff.

**Benefit:** Automates recurring tasks without manual intervention.

**Future Methods**

**Use Case:**

* On donation completion, send an **email acknowledgment** using a future method (non-blocking).

**Benefit:** Improves **performance and user experience**.

**Exception Handling**

Wrapped logic in **try-catch blocks** to handle errors gracefully.

**Example:**

try {

update donor;

} catch (DmlException e) {

System.debug('Error updating donor: ' + e.getMessage());

}

**Benefit:** Prevents failures from breaking entire transactions.

**Test Classes**

Created test classes for **all Apex code**:

* **DonationHandlerTest** → validates loyalty point updates.
* **CampaignHandlerTest** → validates campaign status updates.

**Best Practices Followed:**

* Test coverage > 75% (mandatory).
* Positive and negative test cases.
* Bulk data testing.

**Benefit:** Guarantees **quality and deployability** of code.

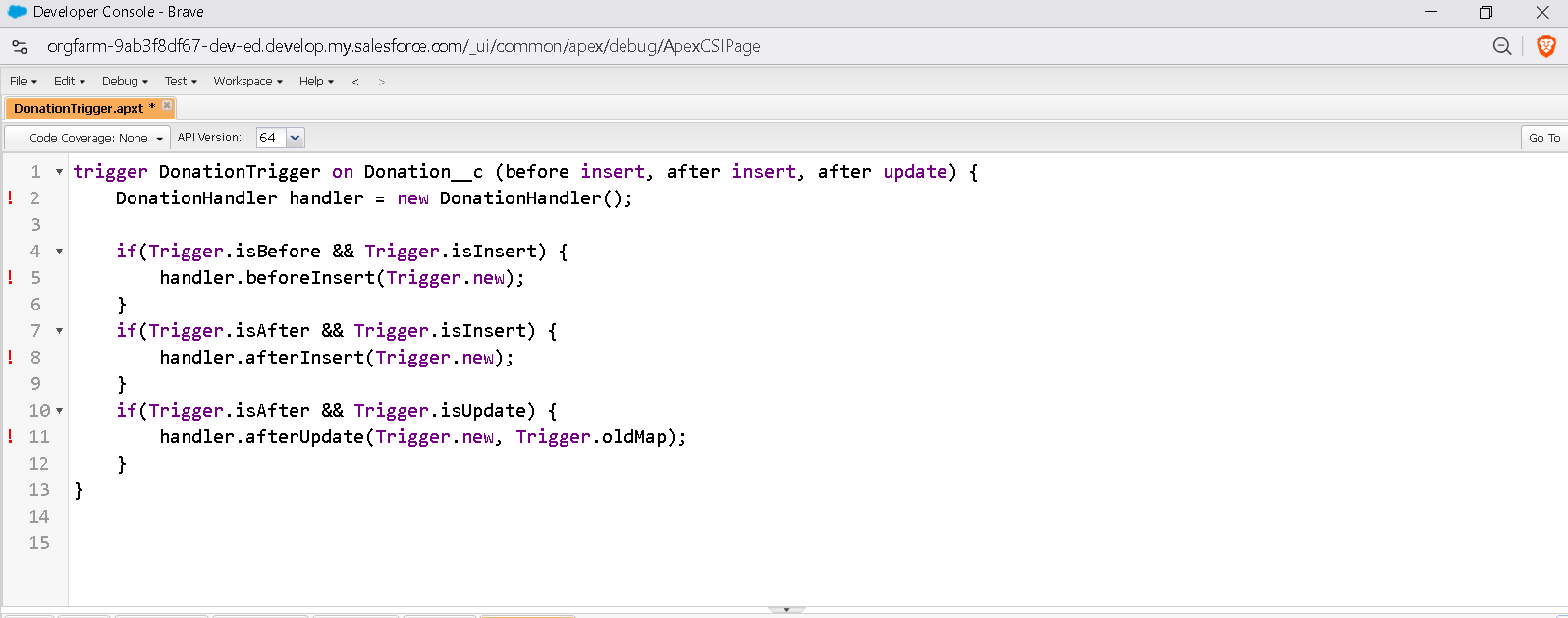
**Asynchronous Processing**

We leveraged **Batch Apex, Queueable, Scheduled, and Future methods** to handle:

* Bulk updates (Batch Apex).
* External system integration (Queueable).
* Periodic reports (Scheduled).
* Quick email sending (Future).

**Benefit:** Keeps the system **scalable, fast, and user-friendly**.

**Screenshots:**

****