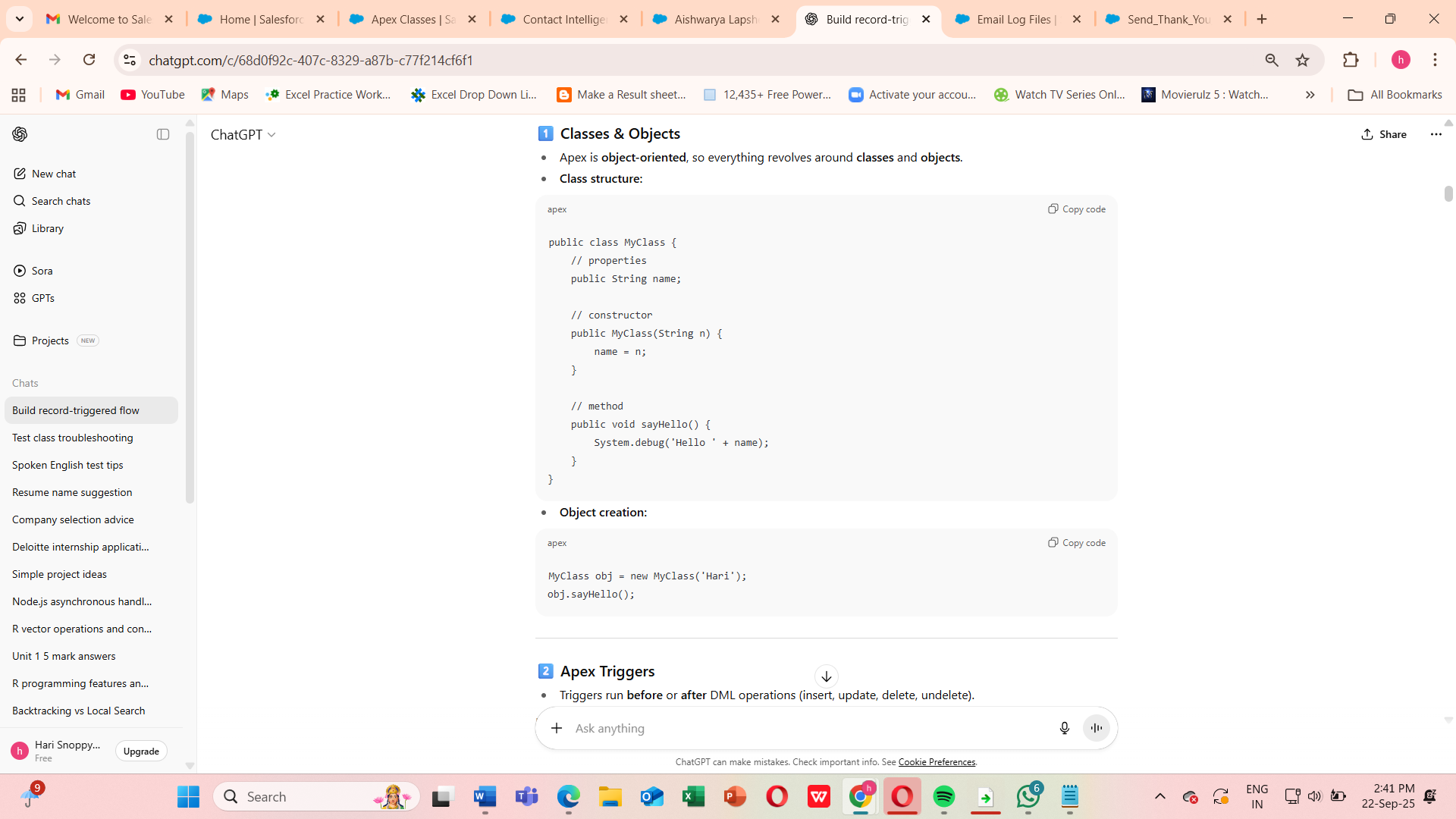
**Phase 5: Apex Programming (Developer)**

**1.Classes & Objects**

* Apex is **object-oriented**, so everything revolves around **classes** and **objects**.
* **Class structure:**



**2.Apex Triggers**

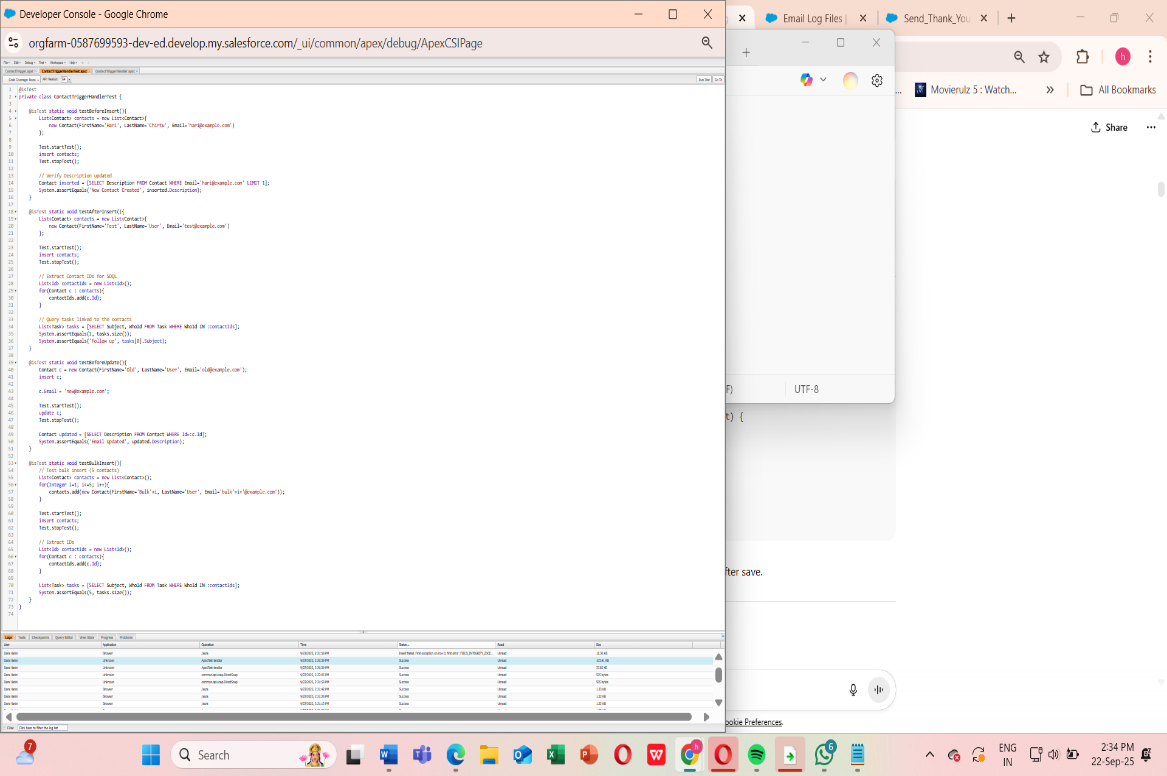
**Apex Trigger**

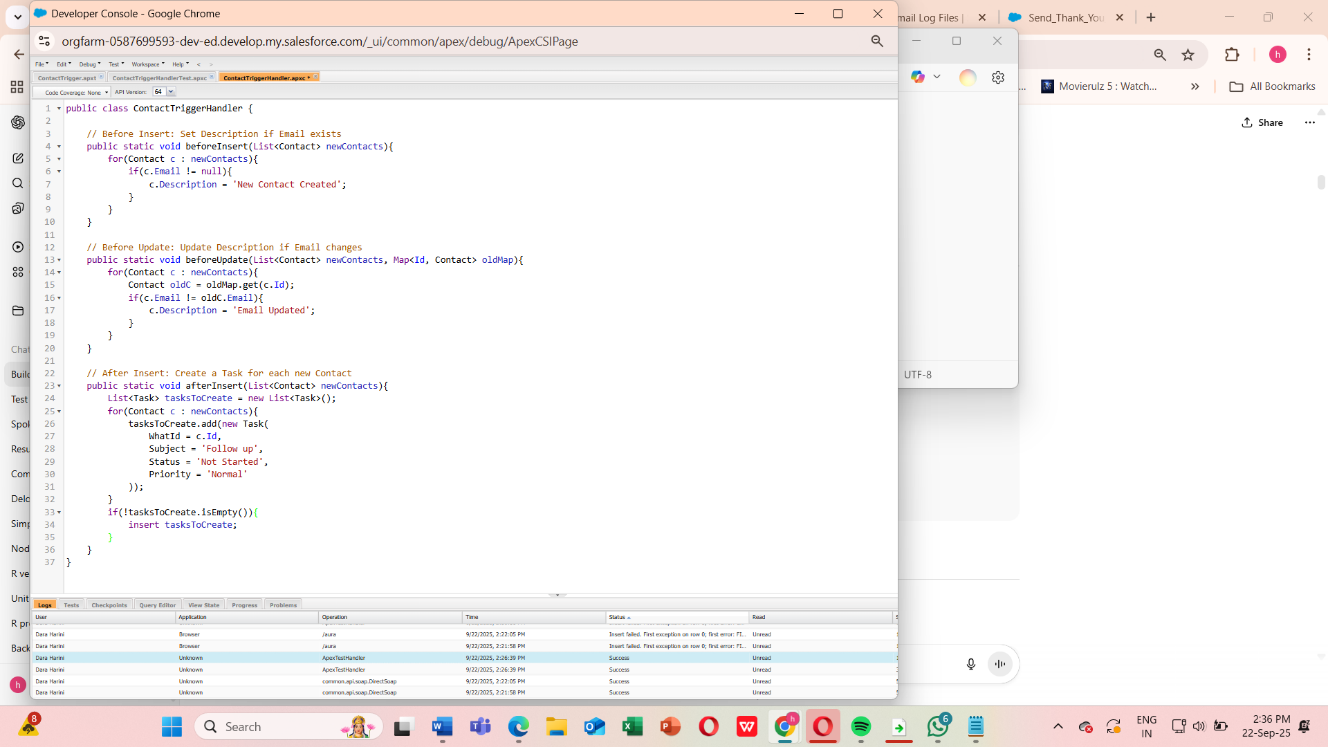
* Trigger Name: ContactTrigger
* SObject: Contact
* Purpose: Just calls the handler methods (router).

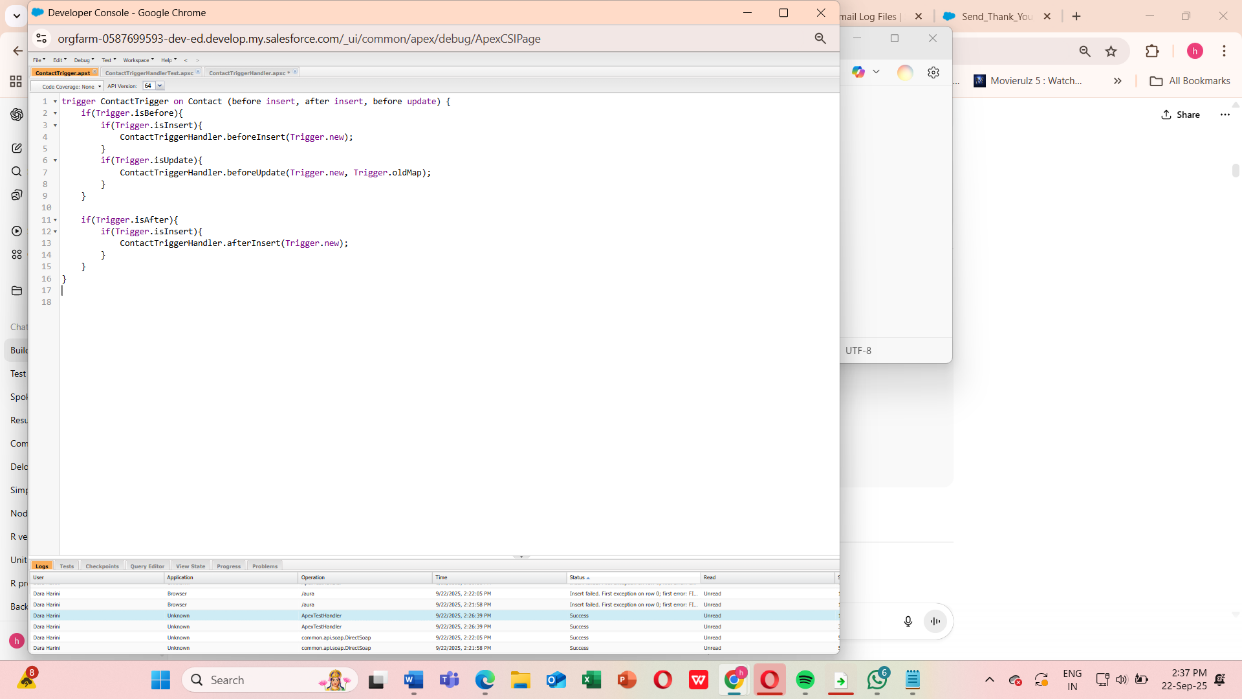
**Apex Class (Handler)**

* Class Name: ContactTriggerHandler
* Purpose: Contains all logic for trigger events (beforeInsert, afterInsert, beforeUpdate).

**Apex Test Class**

* **Class Name:** ContactTriggerHandlerTest
* **Purpose:** Test all trigger scenarios (insert/update).
* Triggers run **before** or **after** DML operations (insert, update, delete, undelete).





**3.Trigger Design Pattern**

Best practice: **Keep triggers clean**, move logic to a **handler class**.

**Step 1: Create a Test Contact**

1. Go to **App Launcher → Contacts → New Contact**.
2. Fill in:
   * **First Name / Last Name**
   * **Email** (required for your trigger logic)
3. Click **Save**.

This action should fire your **ContactTrigger → ContactTriggerHandler → Task creation**.

**Step 2: Open the Debug Log**

1. Go to **Setup → Debug Logs**.
2. Find the **entry for your user** with a recent timestamp.
3. Click **View**.

**Step 3: Verify Trigger Execution**

1. Look for **TRIGGER\_START** or your trigger name: ContactTrigger.
2. Confirm **beforeInsert** logic ran:
3. CODE\_UNIT\_STARTED [ContactTriggerHandler.beforeInsert]
4. SYSTEM\_DEBUG|New Contact Created
5. Confirm **afterInsert** logic ran (Task creation):
6. CODE\_UNIT\_STARTED [ContactTriggerHandler.afterInsert]
7. DML\_BEGIN|Insert Task
8. DML\_END|Insert Task

**Step 4: Check Field Updates**

* Go to **Contacts → Your new contact**
* Check the **Description field** → should say 'New Contact Created'.

**Step 5: Check Related Tasks**

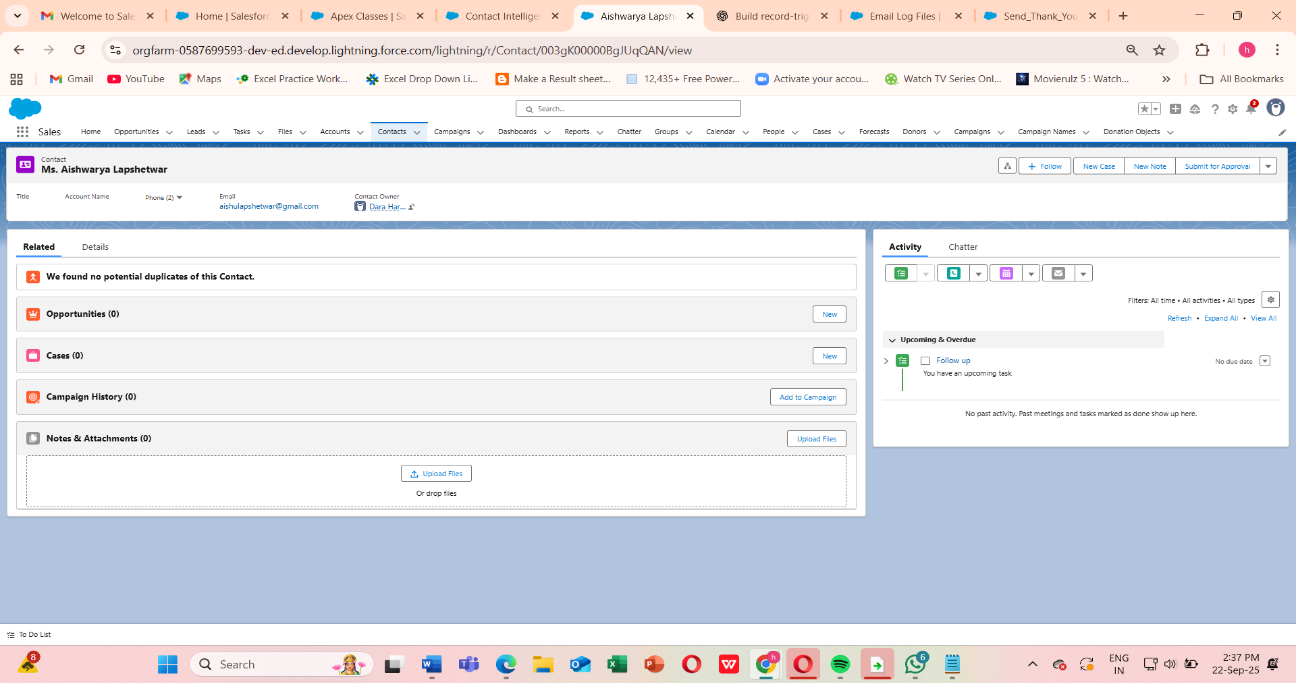
* Scroll to the **Tasks related list** on that Contact
* Verify a **Task “Follow up”** was automatically created.

**Step 6: Test Update Logic**

1. Edit the same Contact and change the **Email** field.
2. Save → this triggers beforeUpdate.
3. Verify in debug log and Contact record:
   * Description should now say 'Email Updated'.

**Once these steps are done**, you’ve fully validated:

* **beforeInsert logic**
* **afterInsert Task creation**
* **beforeUpdate logic**
* **Bulk-safe execution**
* **Debug logs confirming execution**



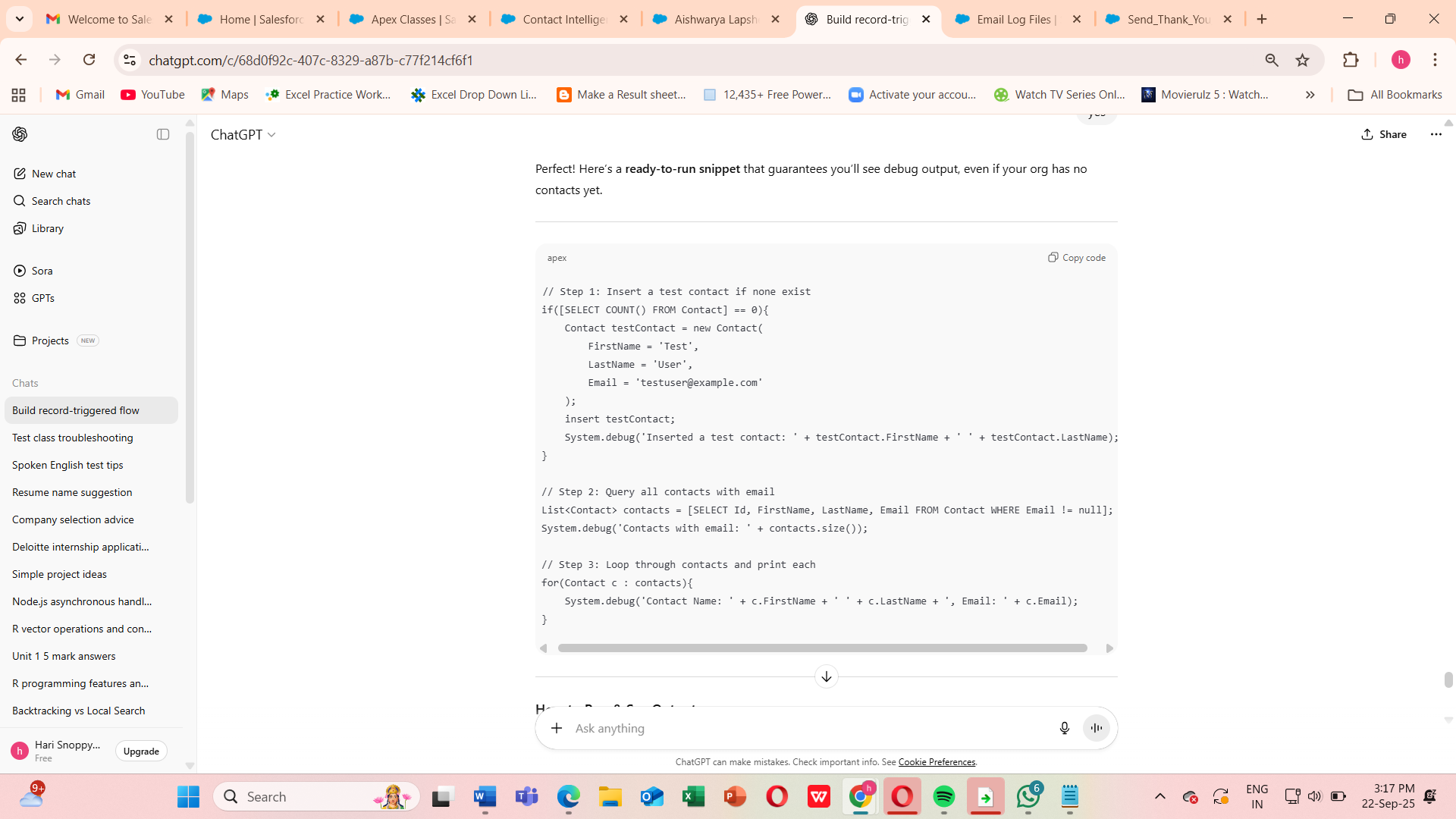
**4.Understand SOQL (Salesforce Object Query Language)**

**Purpose**

* Retrieve records from **Salesforce objects** (like Contact, Account, Lead).
* Similar to SQL, but for Salesforce.

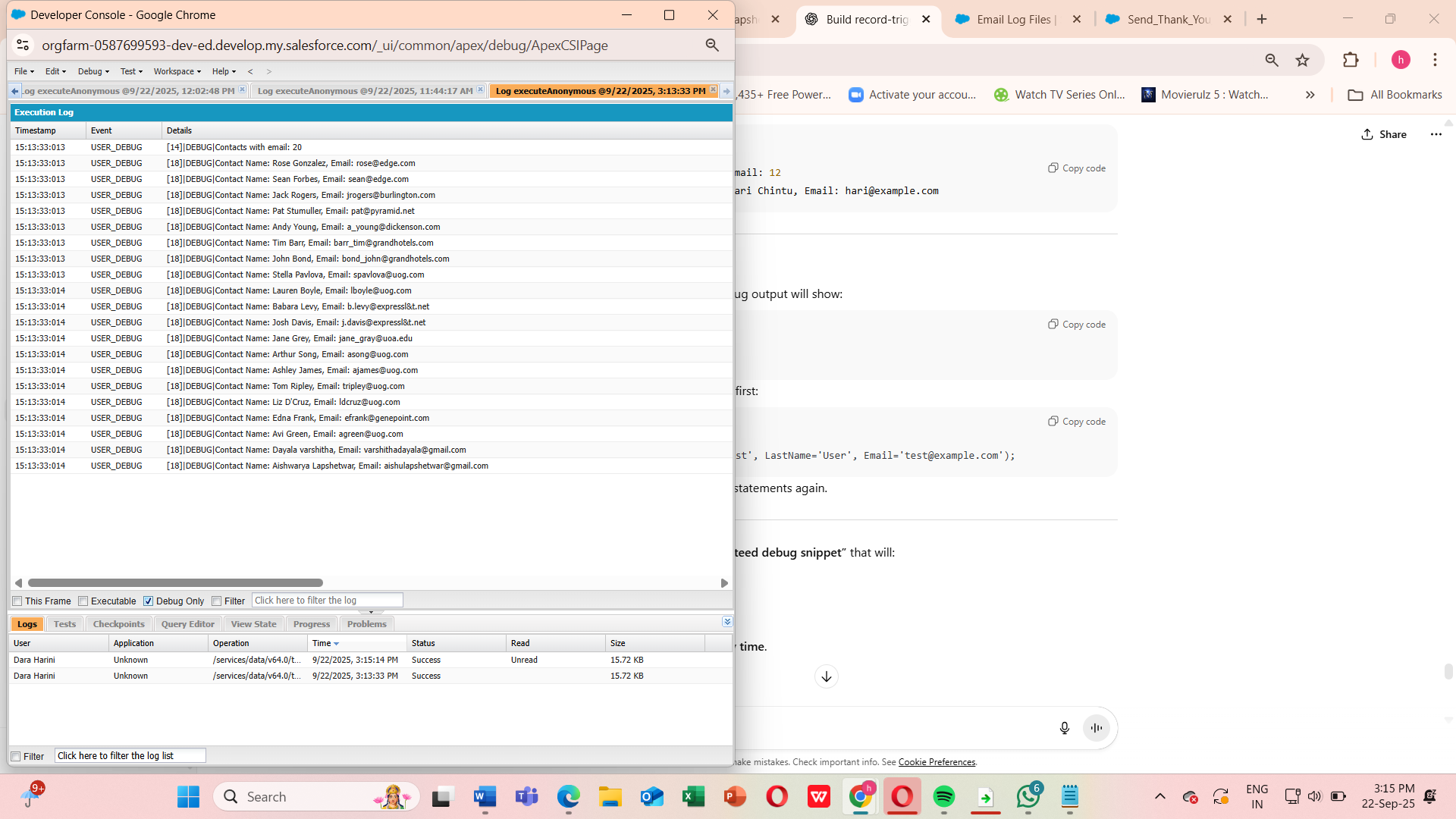
**Step 1: Execute the Code**

1. Go to **Developer Console → Debug → Open Execute Anonymous Window**.
2. Paste your code:



**How to Run & See Output**

1. Open **Developer Console → Debug → Open Execute Anonymous Window**.
2. Paste the snippet above.
3. Click **Execute**.
4. Go to **Logs → Refresh → Open latest log**.
5. Filter by **Debug Only** or search for DEBUG.



**5. Understand Asynchronous Apex**

1.Why Use Asynchronous Apex

* Some operations take too long to run in real-time (triggers have limits).
* Salesforce has governor limits for synchronous DML, SOQL, CPU time, etc.
* Async Apex lets you run code in the background, outside the normal transaction.

Types of Asynchronous Apex

1. Future Methods – run simple background jobs.
2. Queueable Apex – more flexible than future, can chain jobs.
3. Batch Apex – process large data volumes (>50,000 records).
4. Scheduled Apex – run jobs at a specific time or recurring schedule.

**2: Future Methods**

**Purpose**

* Execute logic asynchronously.
* Cannot return values.
* Used for **callouts or heavy DML operations**

[Trigger or Code]

|

v

+------------------+

| Future Method | --> Runs in background, logs/processes records

+------------------+

|

v

+------------------+

| Queueable Apex | --> Can process complex objects, chain jobs

+------------------+

|

v

+------------------+

| Batch Apex | --> Handles large datasets in chunks

+------------------+

|

v

+------------------+

| Scheduled Apex | --> Runs at scheduled times, can call Batch/Queueable

+------------------+

**Key Points:**

* **Future Methods**: Simple, async, void, primitive params.
* **Queueable**: Flexible, supports objects, can chain.
* **Batch Apex**: Large volumes, processes in chunks.
* **Scheduled Apex**: Runs on time schedule, can trigger batch/queueable

**6.Exception Handling & Test Classes**

This is crucial because Salesforce **requires all Apex code to be tested**, and handling errors properly ensures **bulk-safe, robust triggers and classes**.

**Apex Exception Handling**

**Why**

* Prevents your code from **failing unexpectedly**.
* Lets you **catch errors and log them** without stopping other processes.

**Key Points**

* **try** → code that may fail.
* **catch** → handles specific exceptions (DmlException, QueryException, NullPointerException, etc.).
* **finally** → optional, always executes.

**Test Classes in Apex**

**Why**

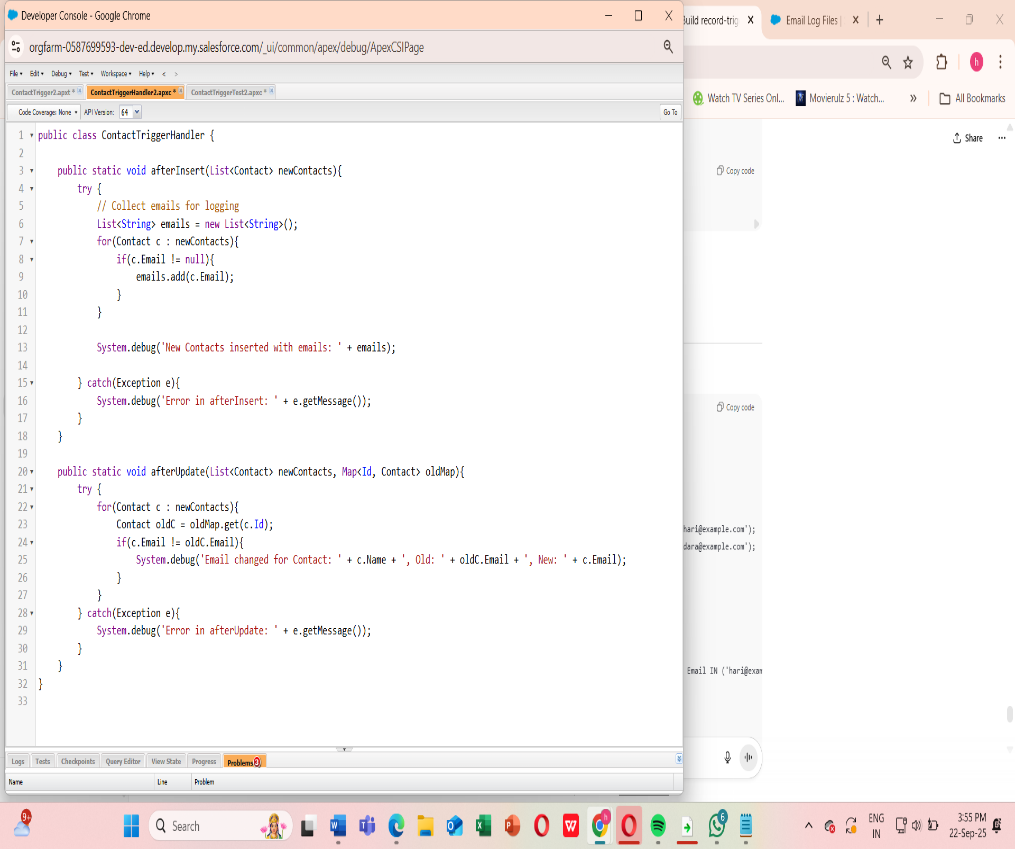
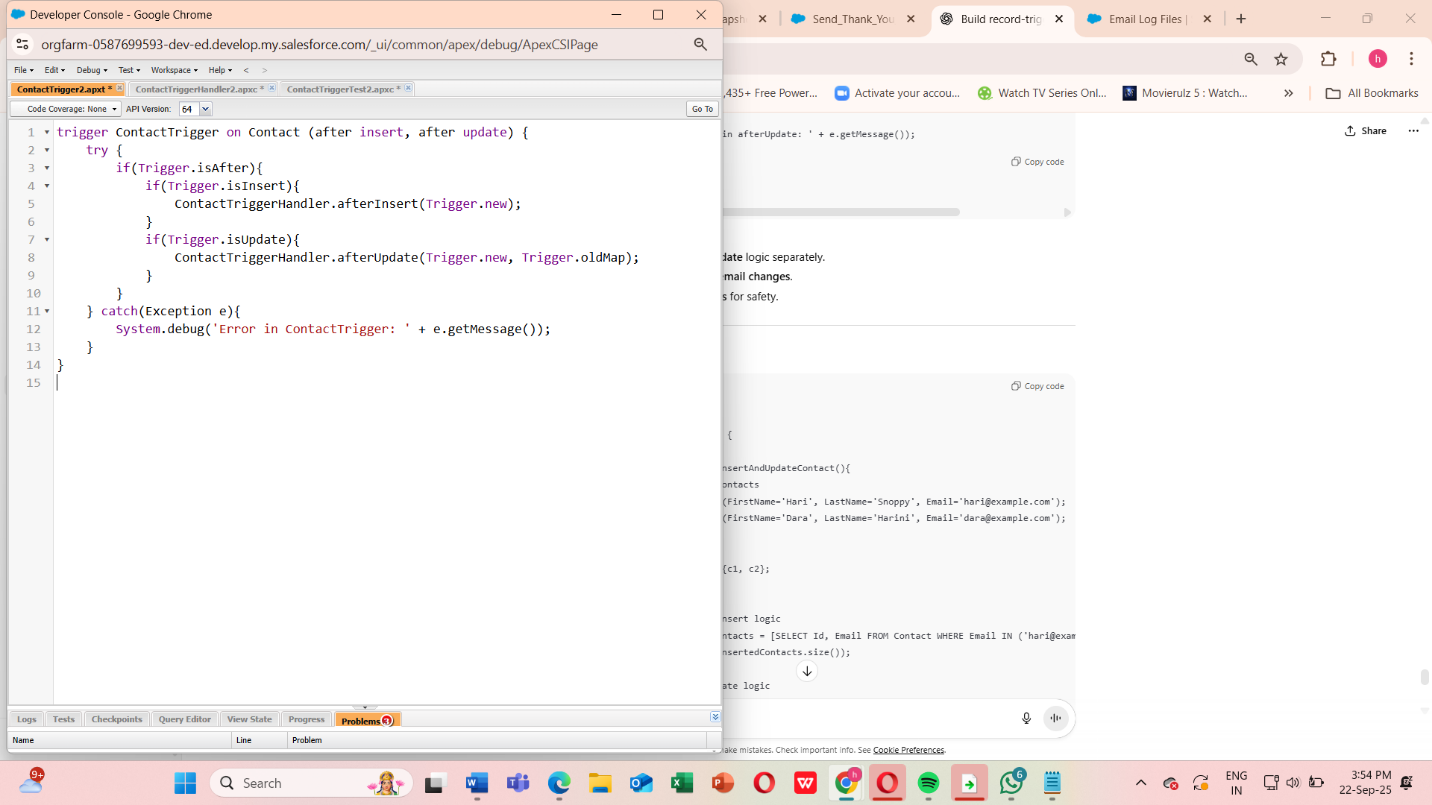
* Salesforce **requires ≥75% code coverage** to deploy Apex.
* Test classes **simulate real data and trigger logic** without affecting production data.

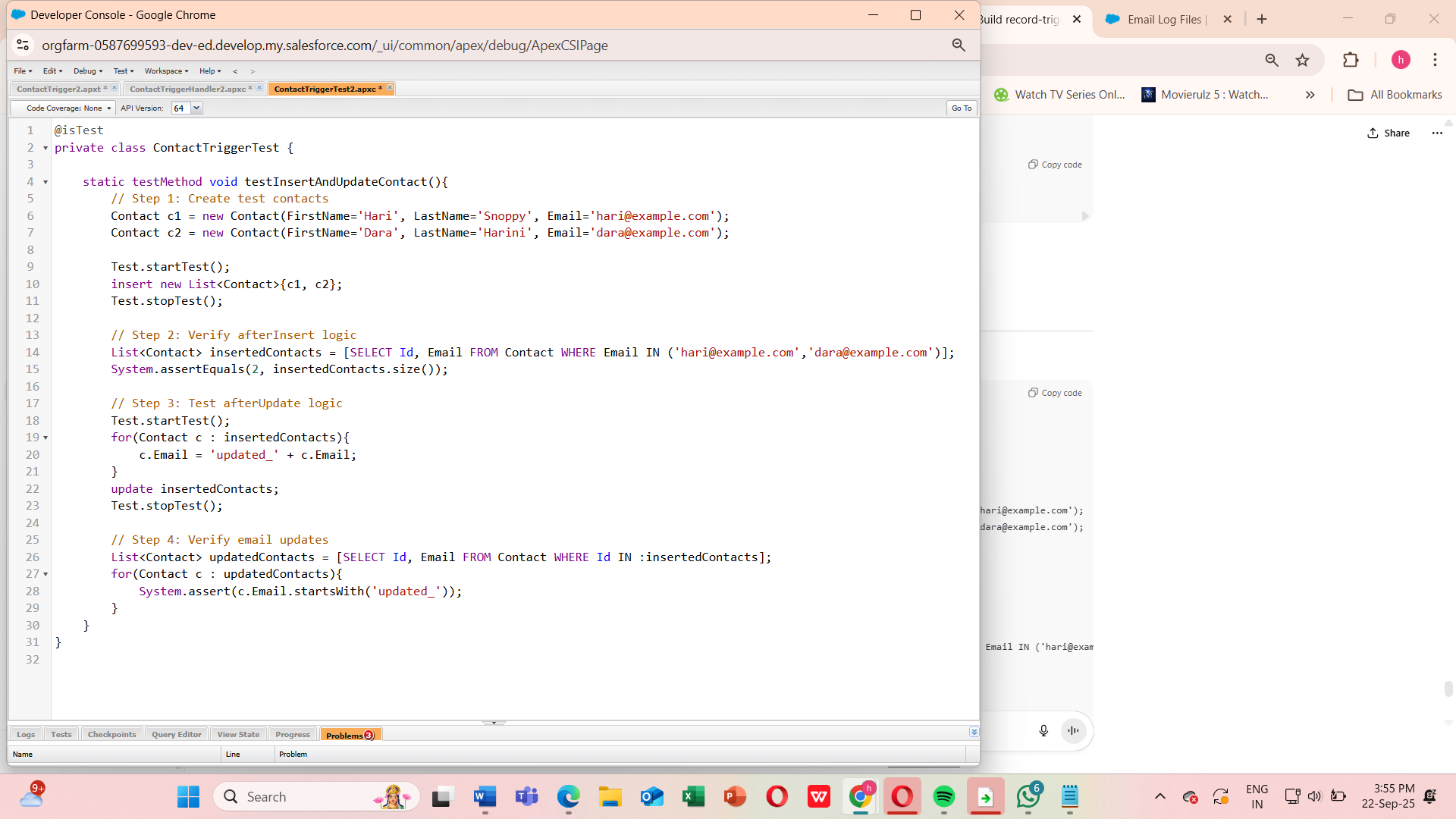
**Key Points**

* Use @isTest annotation.
* Wrap async calls inside Test.startTest() / Test.stopTest().
* Use System.assert to validate behavior.

**How to Run**

1. Save **Trigger**, **Handler**, and **Test Class** in your org.
2. Insert a few contacts via **App Launcher or Execute Anonymous**.
3. Update a contact’s email → check **debug logs**.
4. Run **All Tests** → ensure **code coverage ≥75%**.



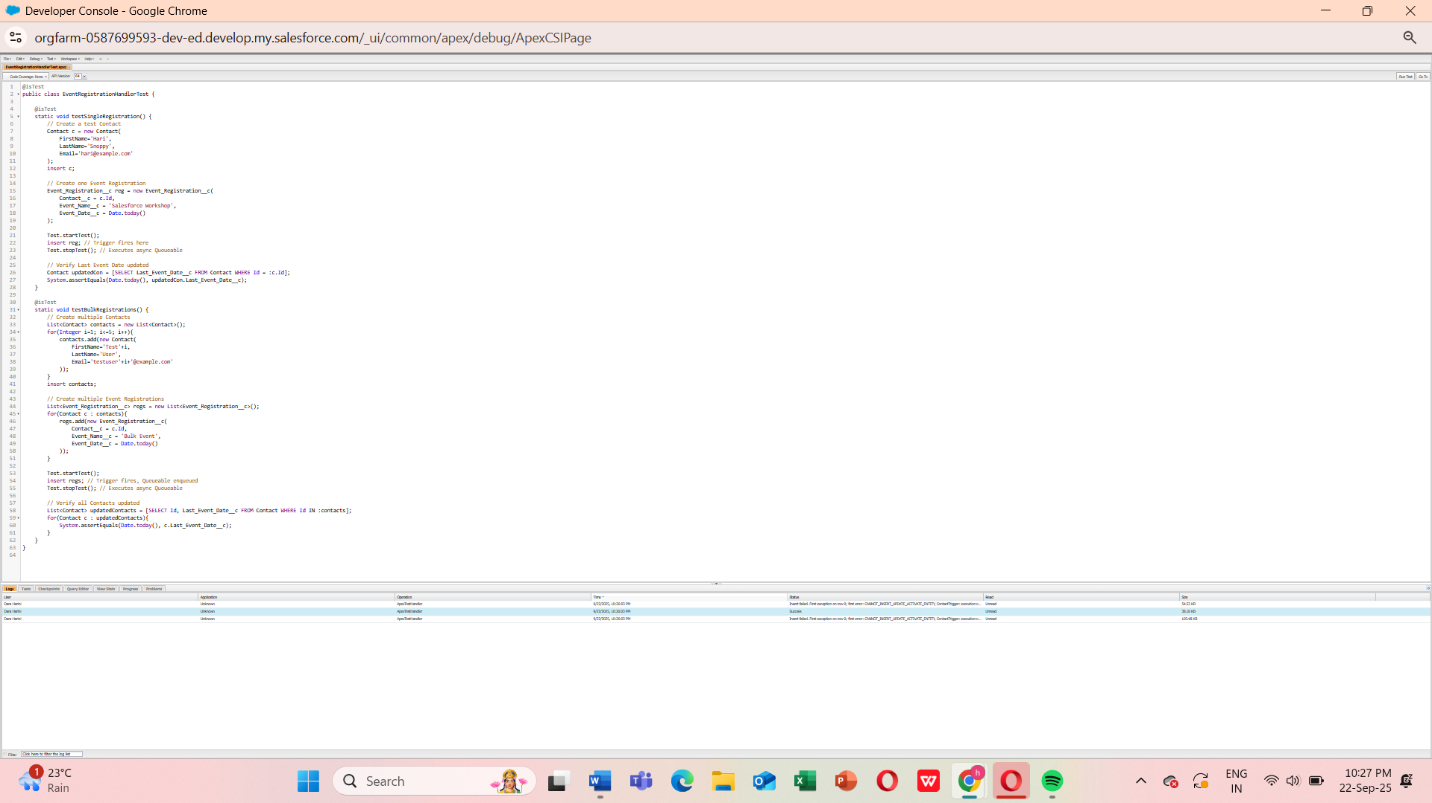


Now,

Trigger + Handler + Queueable + Test Class

**Step 1: Open Developer Console**

1. Click your **avatar → Developer Console**.
2. Go to **File → New → Apex Class**.
3. Name it: EventRegistrationHandlerTest → Click **OK**



**Run the Test**

1. In Developer Console → **Test → New Run**
2. Select EventRegistrationHandlerTest → Click **Run**
3. Wait for results → All tests should pass ✅