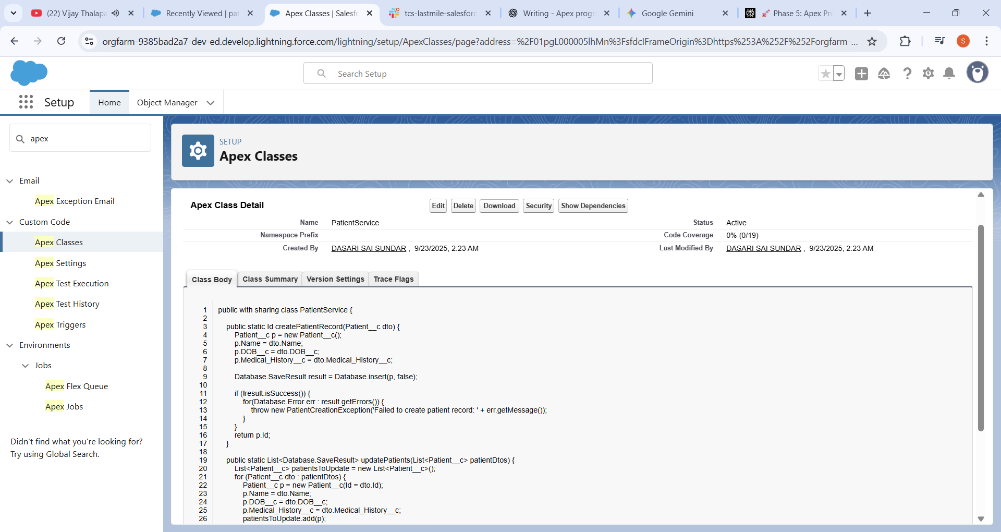
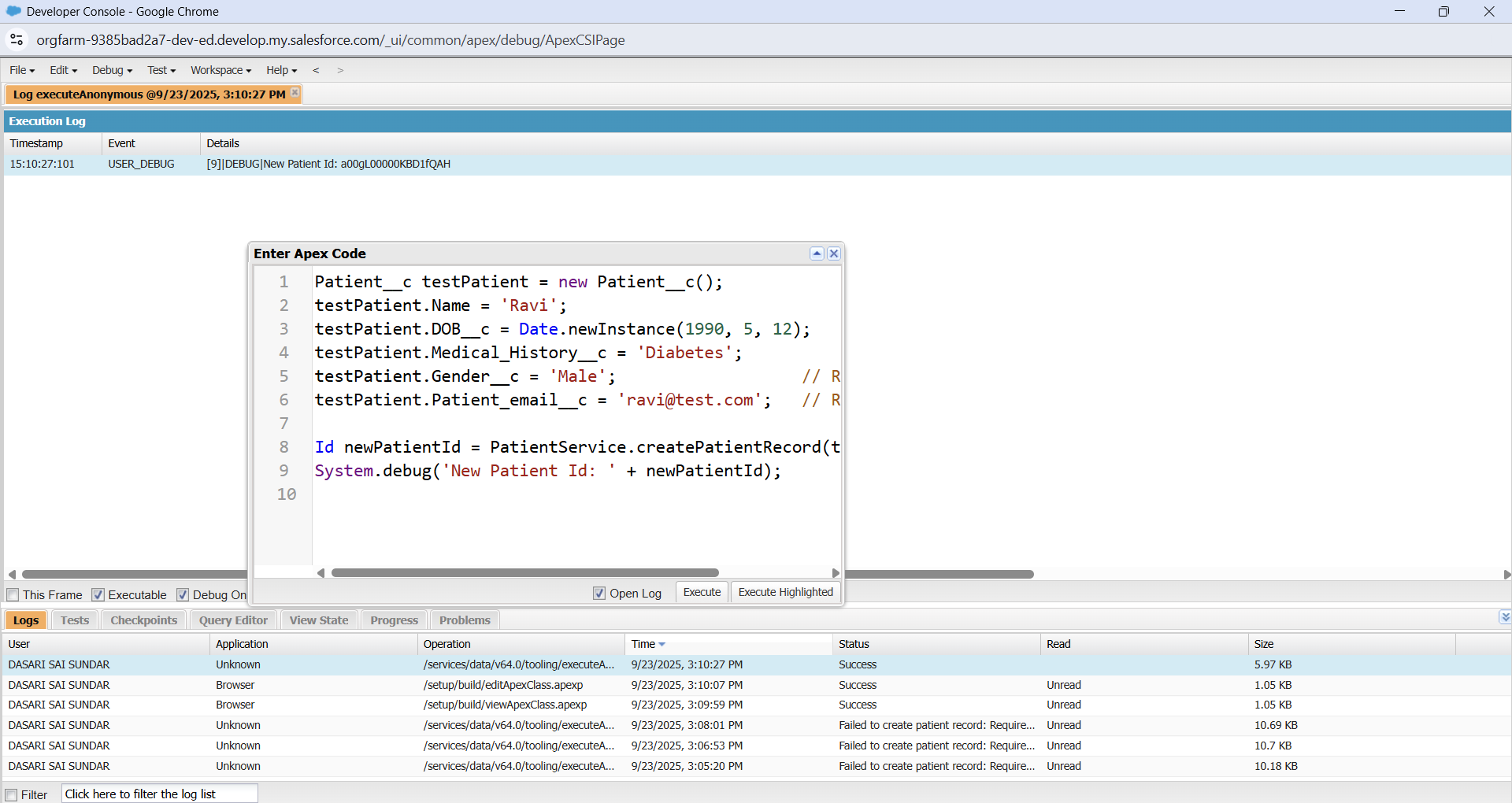
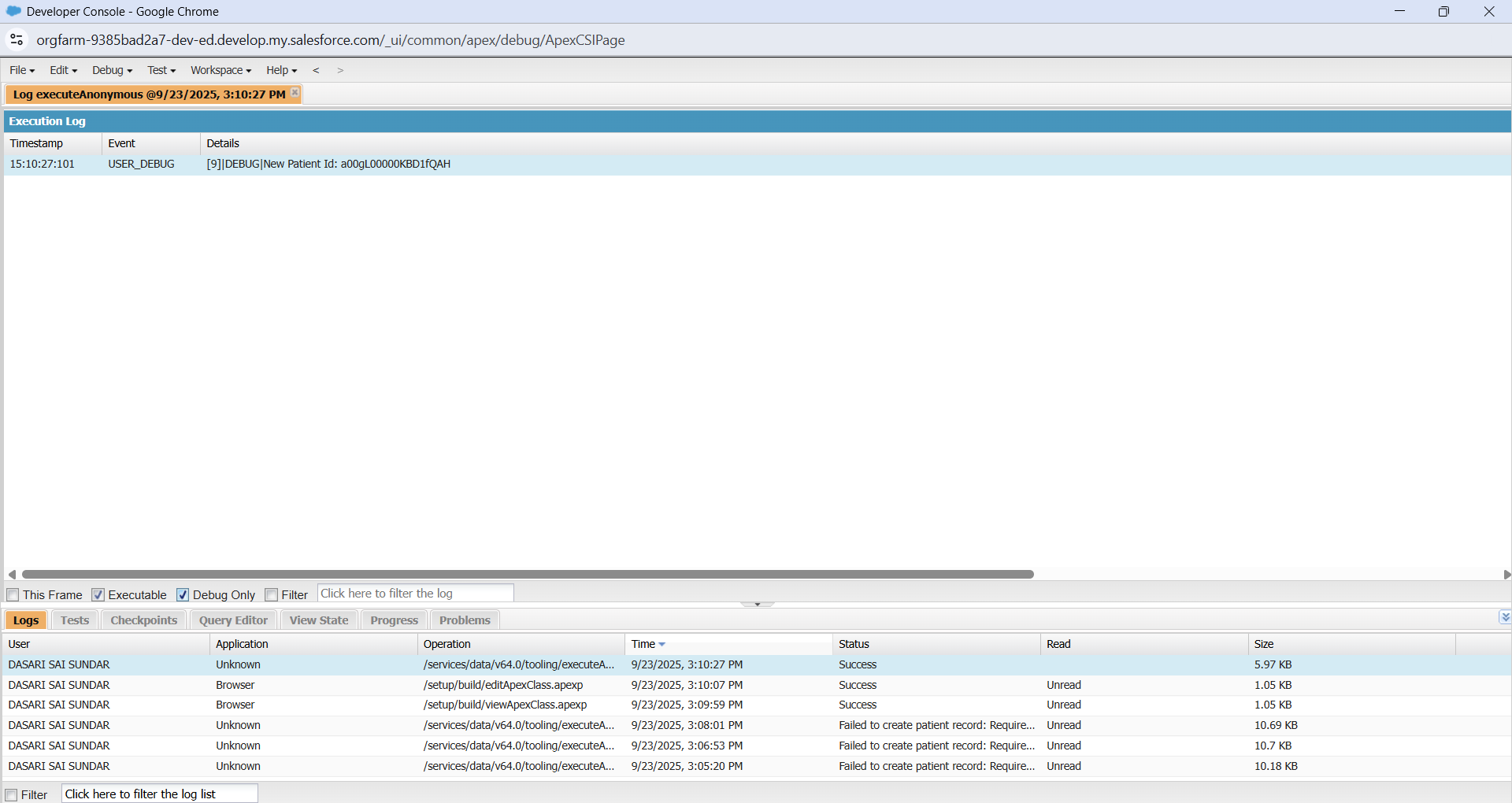
**Phase-5:Apex Classes And triggers:**

**1.Apex classes:**



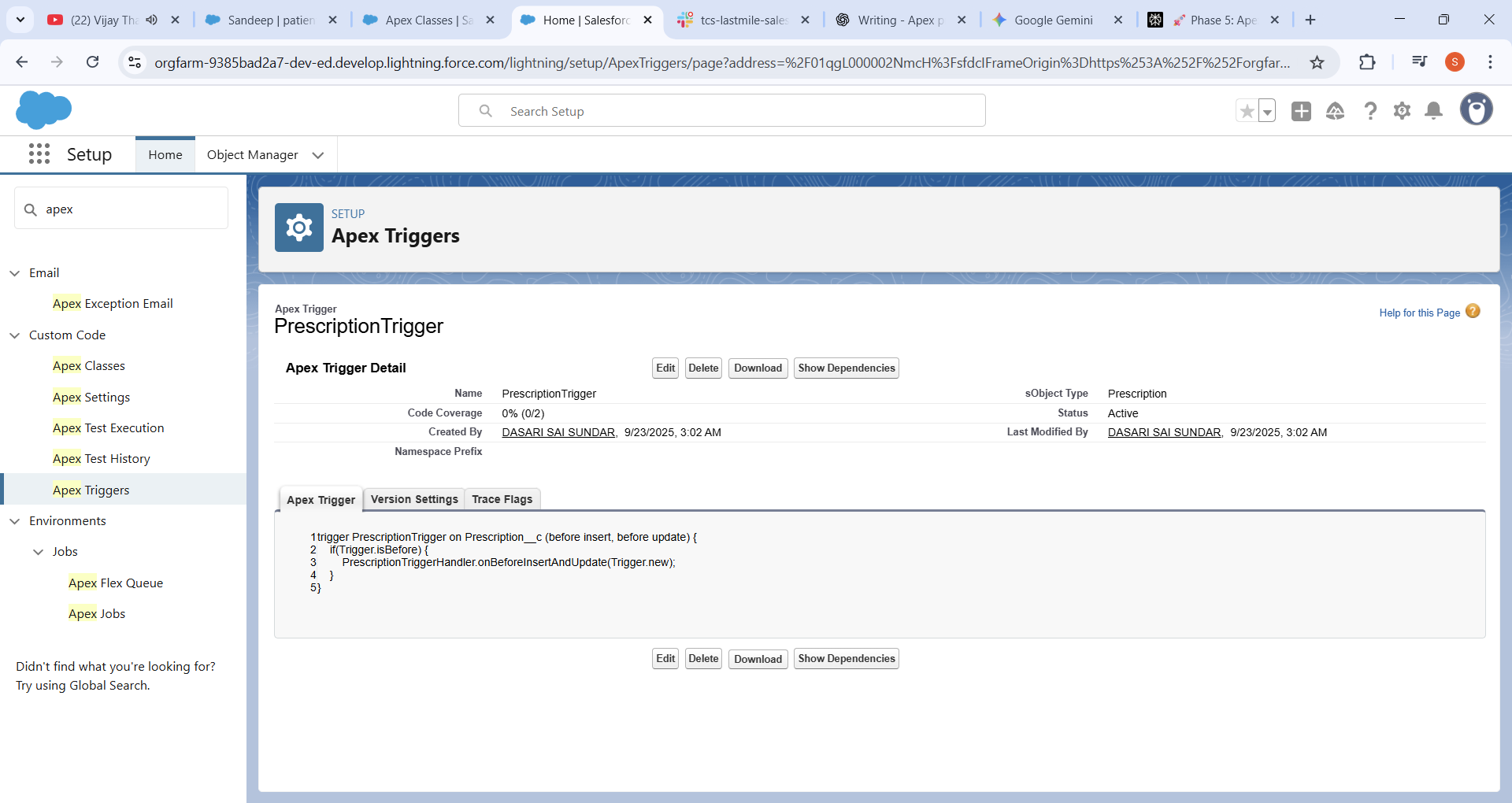
Apex Class 🡪PatientService.cls Created



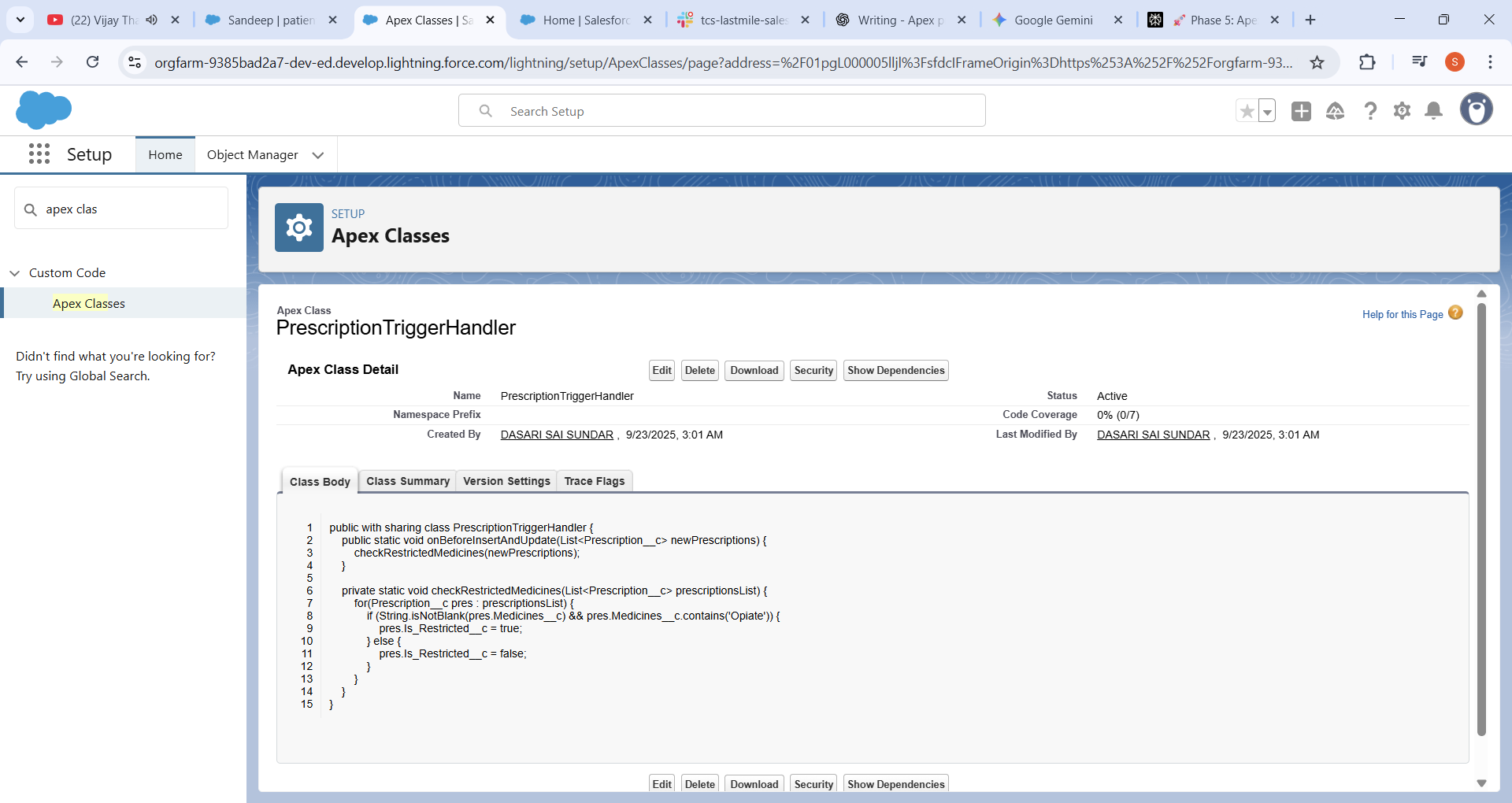


* **Create a new Patient\_\_c record:**
  + Patient\_\_c testPatient = new Patient\_\_c();
  + Initializes a new instance of the custom object Patient\_\_c.
* **Set basic patient details:**
  + Name, DOB\_\_c, and Medical\_History\_\_c fields are assigned appropriate values.
  + These fields store the patient’s name, date of birth, and medical history.
* **Provide all required fields:**
  + Gender\_\_c and Patient\_email\_\_c are mandatory fields in Salesforce.
  + The code sets default values for these fields to ensure the record can be successfully created.
* **Insert the record using the service class:**
  + Id newPatientId = PatientService.createPatientRecord(testPatient);
  + The createPatientRecord method handles the database insert operation.
  + It also performs error handling: if the record cannot be inserted, a custom exception is thrown.
* **Log the new patient ID:**
  + System.debug('New Patient Id: ' + newPatientId);
  + Prints the Salesforce-generated record Id to the debug log, confirming the record was created successfully.

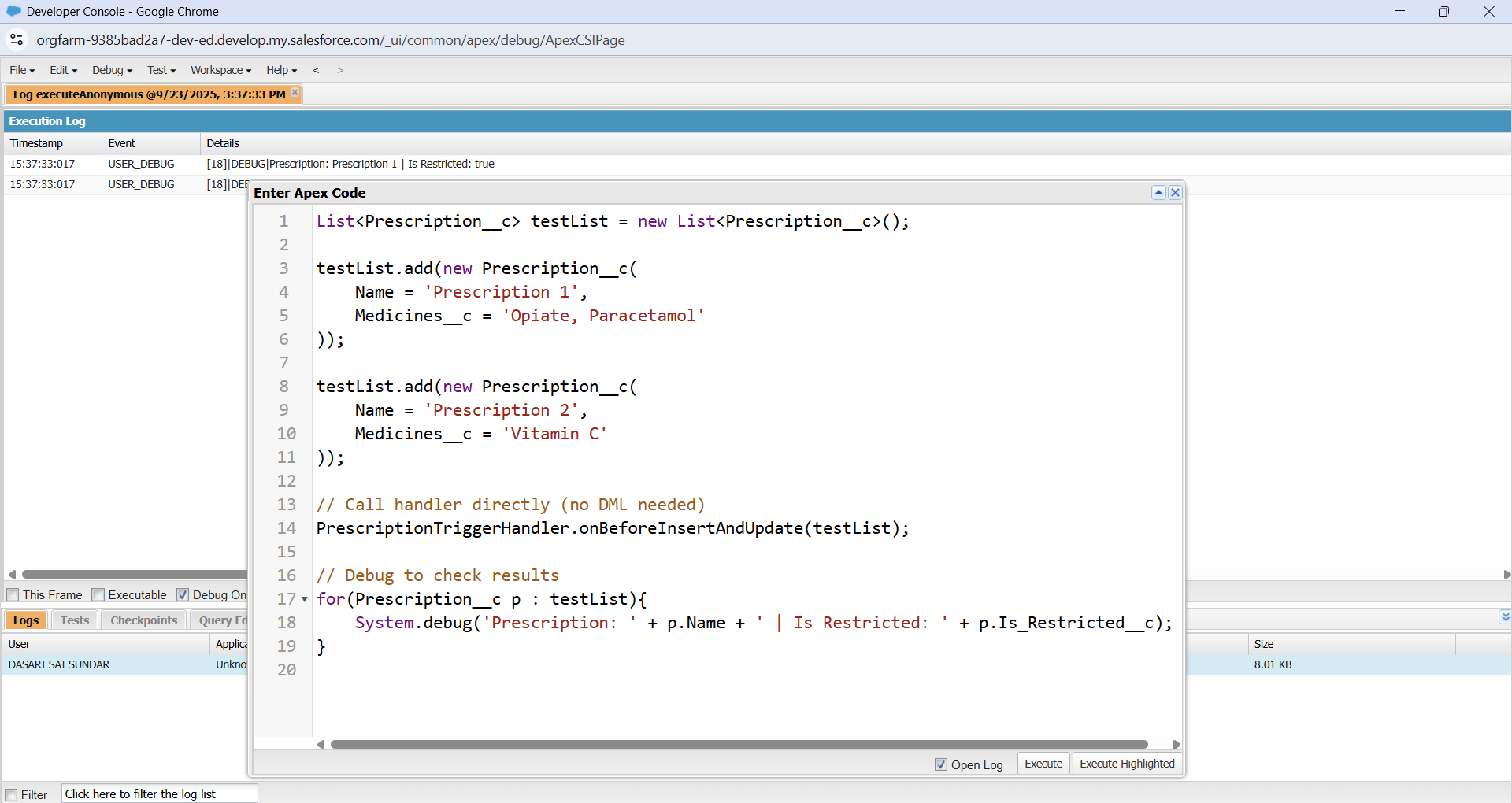
**2.Apex Trigger and Handler**:



Prescription Trigger🡪Apex Trigger for the prescription



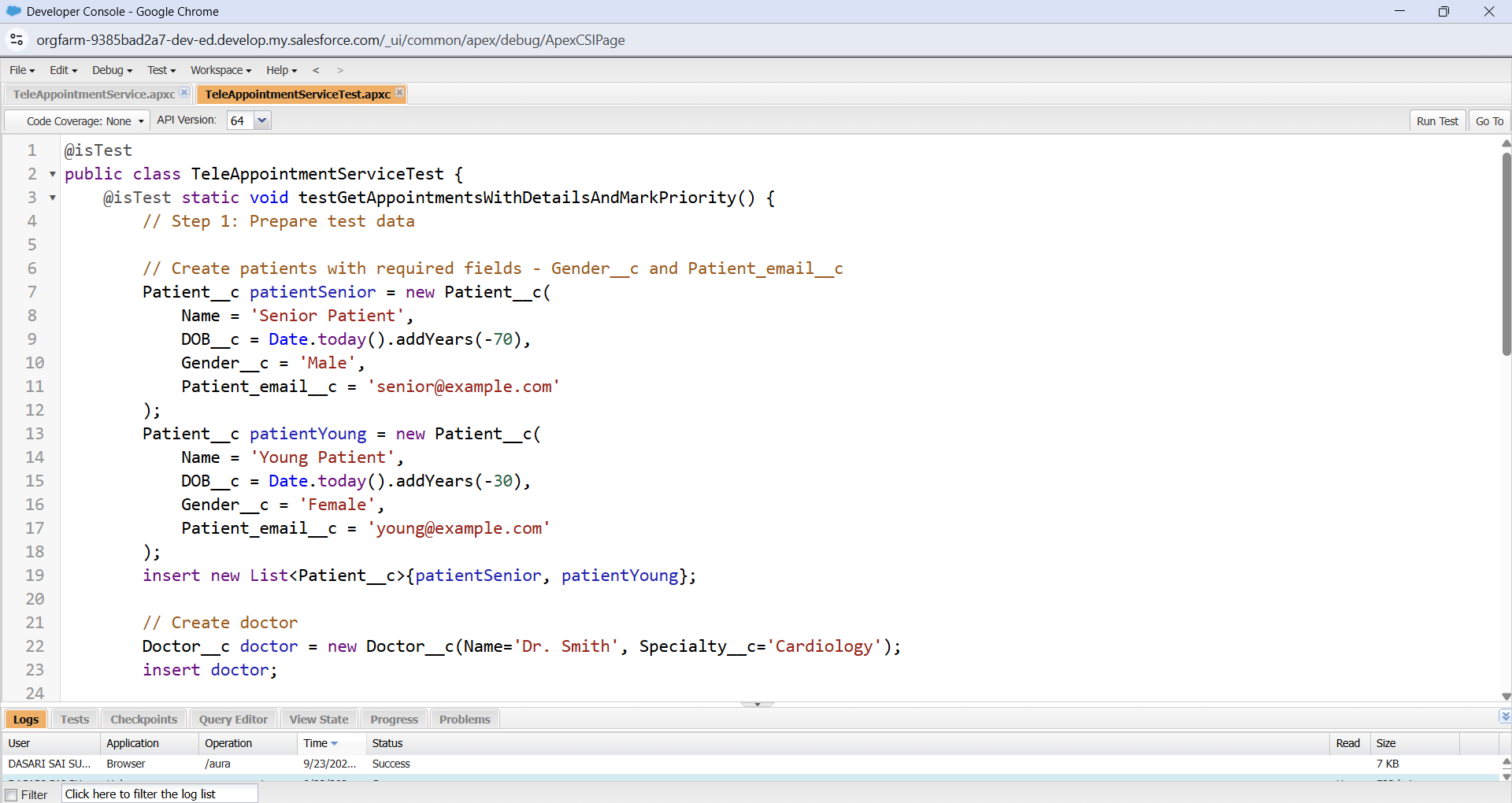
Prescription TriggerHandler 🡪 Apex class for the Trigger Handling



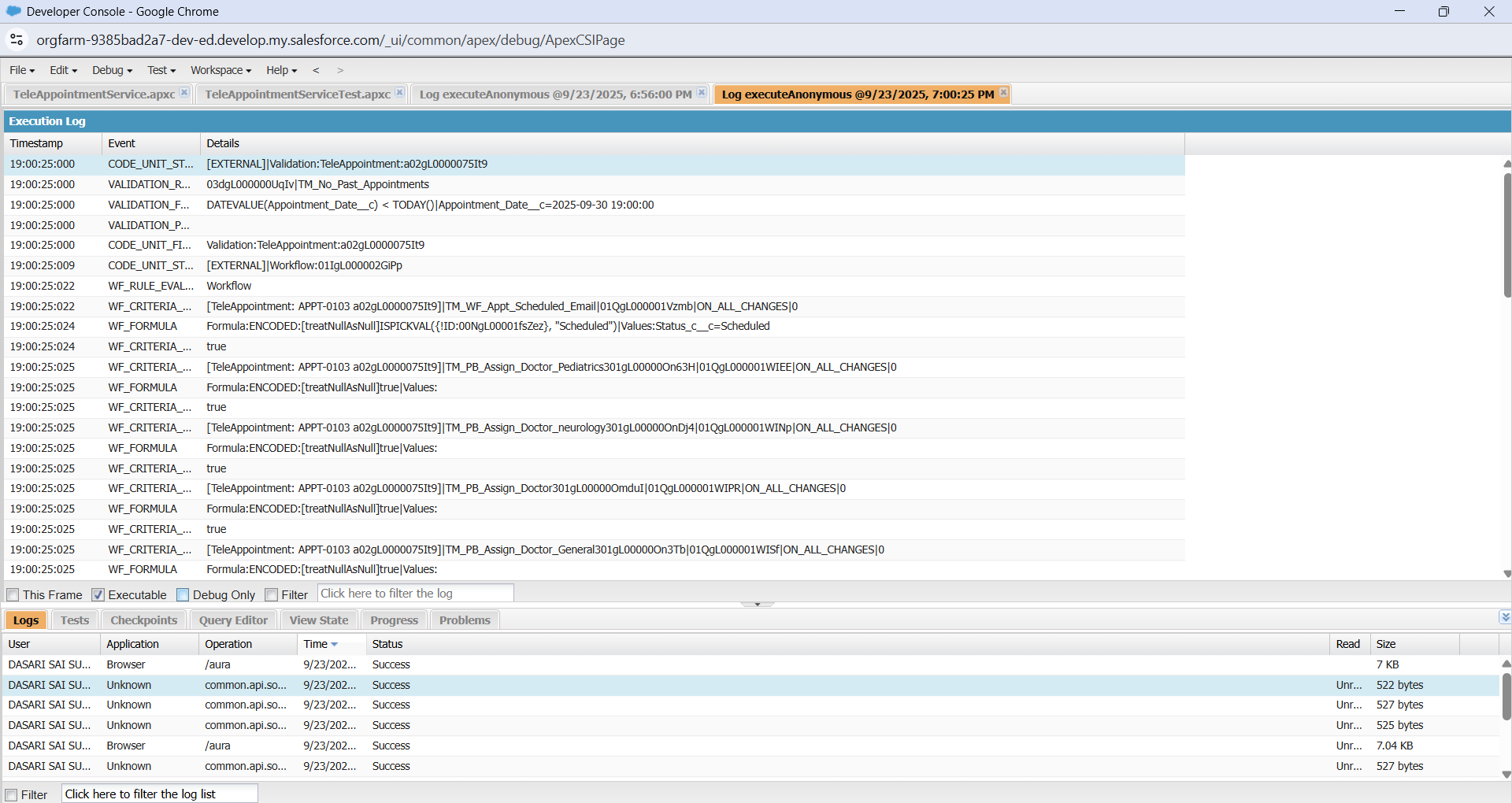
Apex triggers run automatically when records are inserted, updated, or deleted. They delegate logic to a handler class, which processes all records in Trigger.new safely in bulk. The handler applies business rules like marking restricted prescriptions before or after save. Unit tests verify the trigger works correctly for all scenarios.

* Trigger fires automatically when a Prescription\_\_c record is inserted or updated.
* Trigger delegates control toPrescriptionTriggerHandler.onBeforeInsertAndUpdate(Trigger.new).
* Handler loops through each record in Trigger.new and checks the Medicines\_\_c field.
* If the medicines contain 'Opiate', it sets Is\_Restricted\_\_c = true; otherwise, it sets it to false.
* Records are saved to the database with the updated Is\_Restricted\_\_c value

**3.SOQL for the salesforce:**

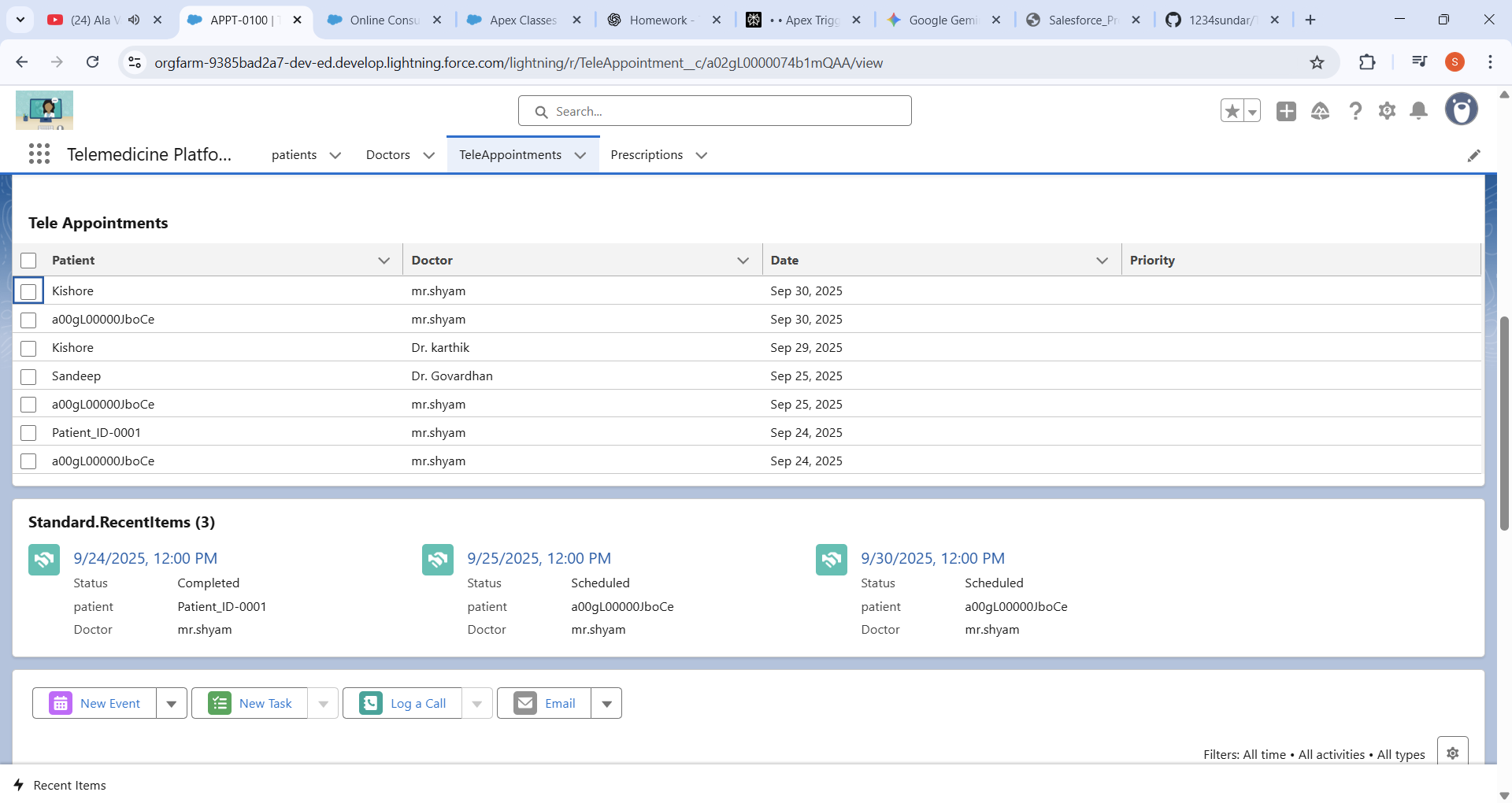
****

Teleappointment Service and Service test is created

****

1. Created test patients, doctors, and future appointments with all required info.
2. Queried appointments with related doctor and patient details for full context.
3. Calculated patient ages to flag senior patients’ appointments as priority.
4. Updated appointments to mark priority status based on age condition.
5. Ran assertions verifying data integrity and priority logic worked perfectly in tests.

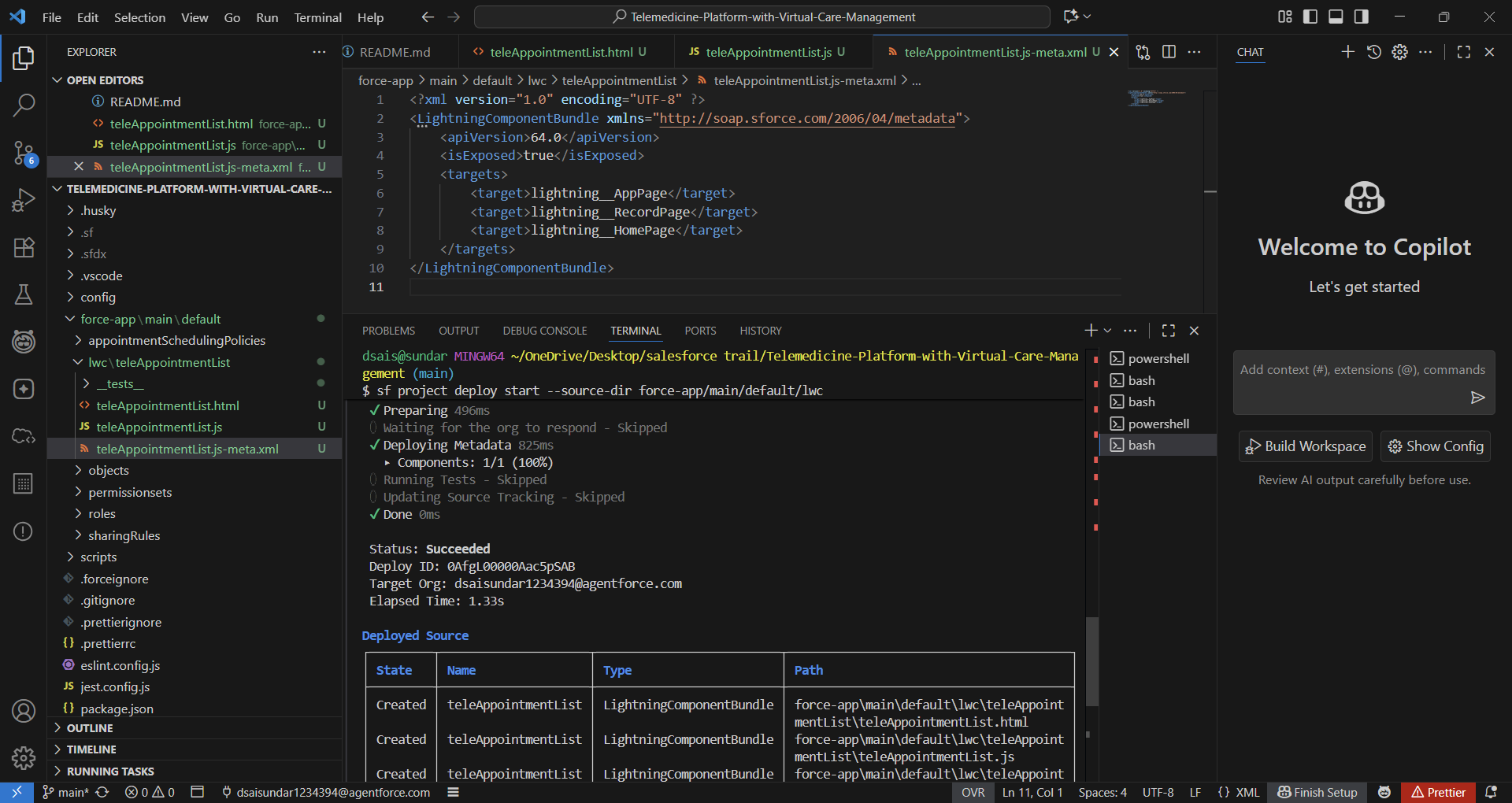
**4. Lightning Web Component (LWC):**

****

**Lightning Web Component (LWC)** is a dynamic dashboard for your appointments. It works by having three parts that connect and talk to each other.

1. The HTML: This file is the visual blueprint of your component. It uses a <lightning-datatable> to show the data on the screen.
2. The JavaScript: This is the logic that gets the data. It uses the @wire decorator to automatically call your Apex controller when the component loads.
3. The Apex Controller: This is the data fetcher. It runs a SOQL query on your database to get the latest appointment records and sends them back to the JavaScript.

your component loads on a page, the JavaScript tells the Apex controller to get the data, and the HTML displays it in a table for the user. It's an automated process that gives you a real-time view of your appointments.

****

The lightning web Component is created by the help of the vscode and deploy using the vs code