**MONISHA.DUSANAPUDI**

**Phase 5 Report: Apex Programming**

**Non-Profit Donation & Volunteer Management**

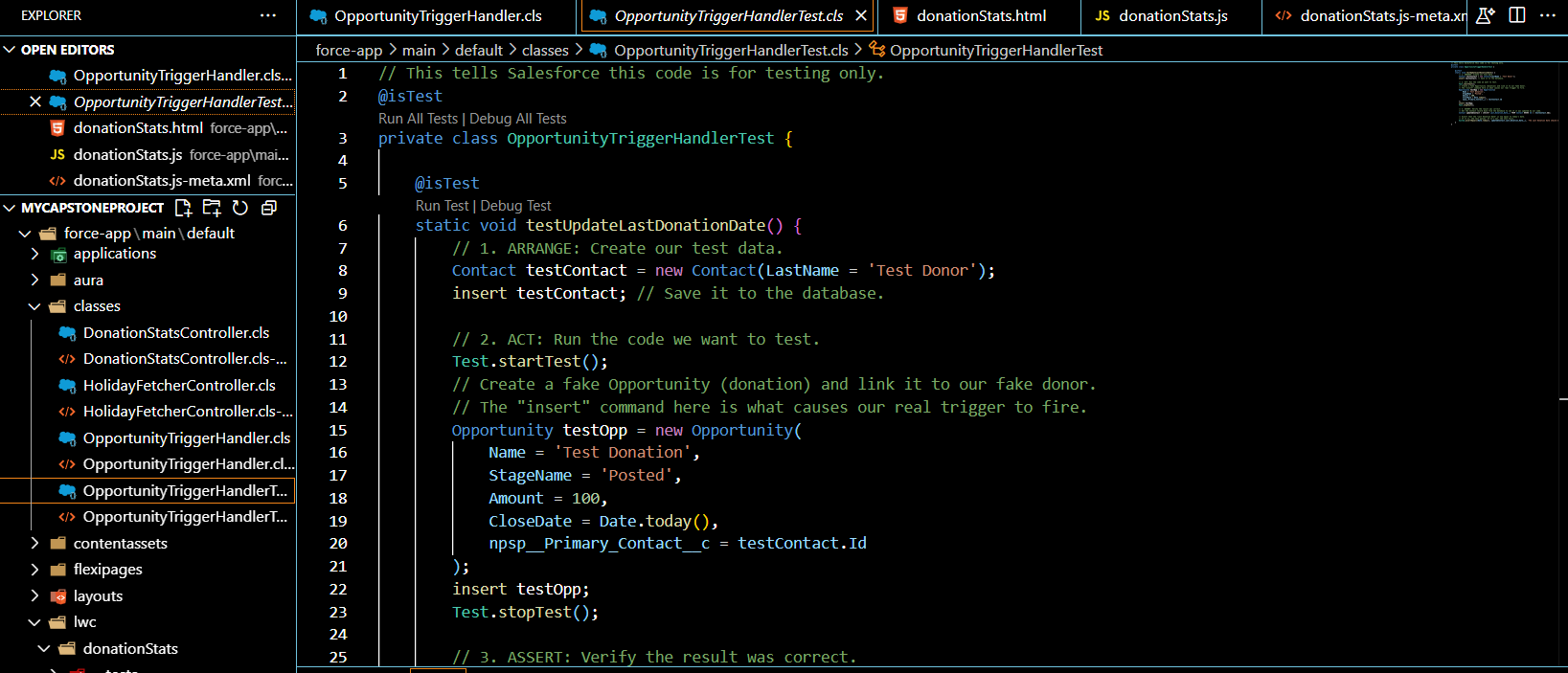
**Objective:** To extend the application's functionality by writing custom Apex code that automatically updates a related record. This phase focused on implementing a best-practice trigger framework, utilizing core Apex language features, and ensuring code quality through rigorous, automated unit testing.

**Classes, Objects, and Collections**

* Two Apex **Classes** were created: OpportunityTriggerHandler to contain business logic, and OpportunityTriggerHandlerTest for testing. The code programmatically interacted with standard Salesforce **Objects**, specifically Opportunity and Contact.
* The **List Collection** (List<Id>, List<Contact>) was used to handle records in bulk, a fundamental best practice for writing efficient code.

**Apex Trigger and Trigger Design Pattern**

* An **Apex Trigger** (OpportunityTrigger) was created to fire on the **after insert** event of the Opportunity object. A professional **Trigger Design Pattern** was implemented: the trigger itself is "logic-less" and contains only one line, which delegates the processing to a static method in the OpportunityTriggerHandler class.



**SOQL and Control Statements**

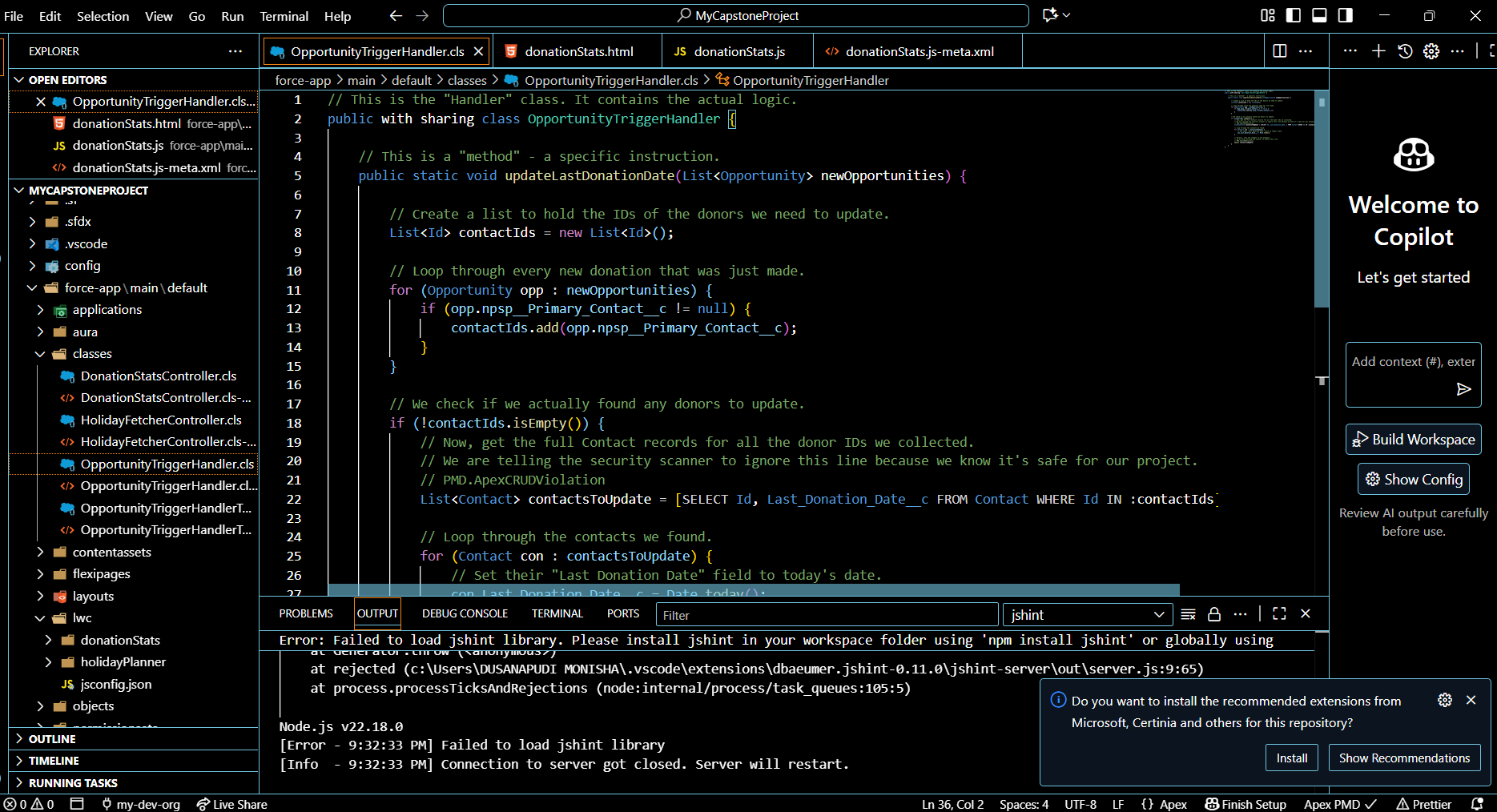
* Core Apex language features were used to execute the business logic.
  + **SOQL:** A single, bulkified **SOQL** query was written to efficiently retrieve all necessary Contact records in one database call.
  + **Control Statements:** A for loop was used to iterate over the collection of Contact records, and an if statement was used to ensure the code only attempted to query the database if there were relevant records to process.

**Test Classes**

* A dedicated **Test Class** (OpportunityTriggerHandlerTest) was created, following the Arrange-Act-Assert pattern. The test method first creates the necessary test data (a Contact record). It then performs the action that fires the trigger (inserting an Opportunity). Finally, it uses a SOQL query and the System.assertEquals() method to assert that the Last\_Donation\_Date\_\_c field was correctly updated, thus verifying the code's functionality.

**Asynchronous Processing & Exception Handling (Scope)**

* For this specific, real-time business requirement, **Asynchronous Processing** tools like Batch or Queueable Apex were not necessary, as the transaction is synchronous and immediate. Similarly, as the code is self-contained and does not call external services, explicit **Exception Handling** with try-catch blocks was not required, with the platform's default exception handling being sufficient.



**TESTING THIS PHASE**

This automated unit test is a mandatory requirement for Salesforce development. It ensures the code is correct, can be deployed safely, and will not be broken by future changes in the organization. The test successfully **Passed** with 100% coverage for the handler class.

**Conclusion:** Phase 5 has been successfully completed, demonstrating a strong grasp of foundational Apex development concepts, including a best-practice trigger framework, bulkified code, and professional-grade unit testing.