# ­­my clinic

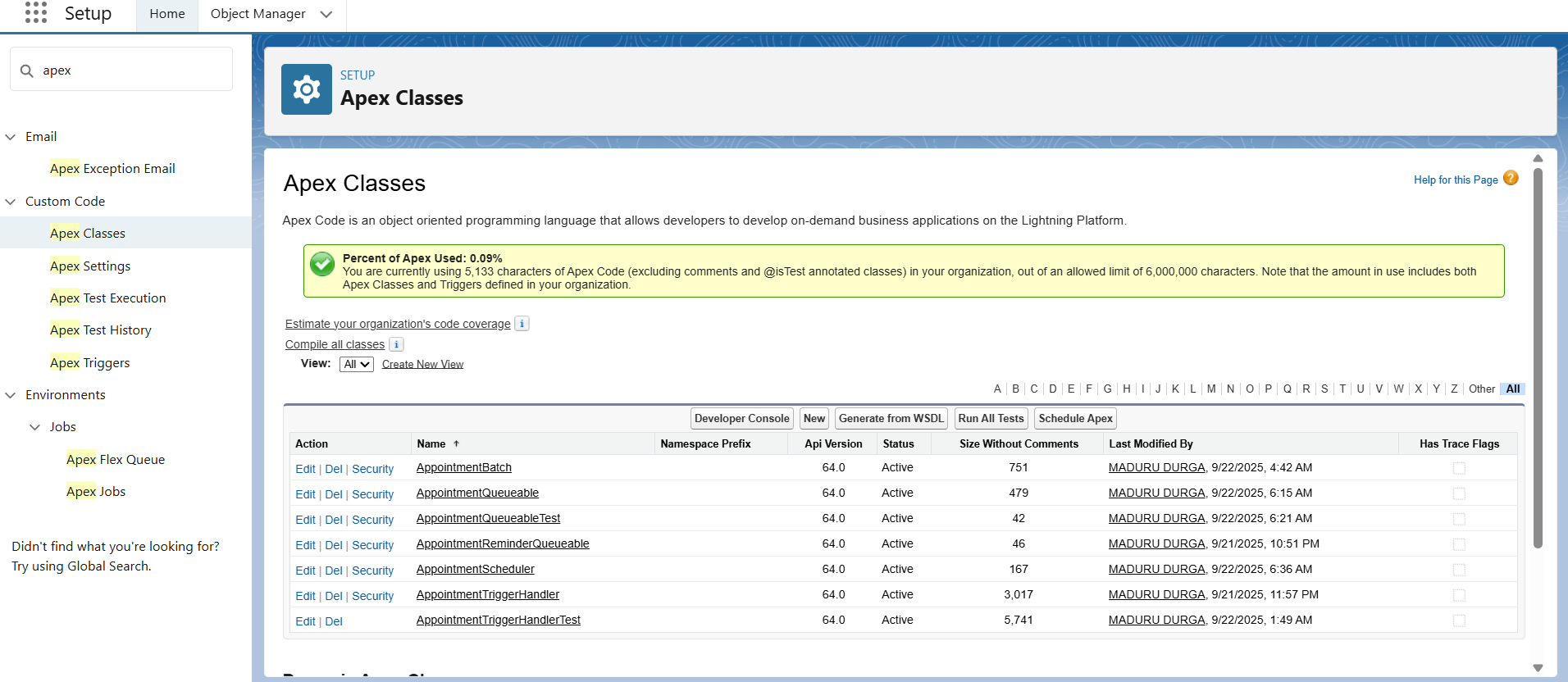
# Smart Appointment Booking – CRM Project

## Phase 5: Apex Programming (Developer)

* Goal: The goal of this phase was to implement Apex triggers in Salesforce to automate appointment booking processes, enforce business rules, and improve data integrity within the Smart Appointment Booking System.

### 1. Class & Objects

* Created a **Appointment Trigger** Apex Class to handle the business logics for the appointment bookings
* Created a **AppointmentTriggerHandler** class to handle **AppointmentTriggerTest class.**

****

### 2.Apex Triggers (Before/After Insert/Update/Delete)

* Created A appointment Trigger on Appointment\_\_c Object.
* Trigger methods Contains before insert, before update, after update, after insert.
* Prevent double booking of a doctor at the same date and time.
* Restrict bookings from past dates and times.
* Ensure appointments are only booked during business hours (9 AM to 5 PM).
* Automatically clear follow-up date on appointment cancellation.
* Covered above 90% of code Coverage

A screenshot of a computer

AI-generated content may be incorrect.

### 3. Trigger Design Pattern

### Trigger Delegation: The AppointmentTrigger delegates all processing to the AppointmentTriggerHandler class.

### Clean Separation: This ensures the trigger remains clean, handling only the execution context and delegating the core logic to the handler class.

### Reusable and Modular Code: Multiple methods in the handler class manage different trigger events such as before insert, before update, after insert, and after update.

### Easier Maintenance: New logic can be added or existing logic modified in the handler class without affecting the trigger's core structure.

### 4. SOQL & SOSL

* Used the SOQL Queries to fetch the Appointment Records whose status is Confirmed for today.
* SOSL queried are used to find all Records by patient names

A screenshot of a computer

AI-generated content may be incorrect.

### 5. Collections (List, Set, Map)

### Used List<Appointment\_\_c> to process multiple appointment records in bulk during trigger execution.

### Applied Set<ID> to gather unique doctor Ids from appointments and prevent duplicate SOQL queries.

### Implemented Map<Id, Doctor\_\_c> to efficiently map doctor Ids to doctor records for quick lookups in business logic.

### 6. Control Statements

* Used If-Else conditions to check appointment status, such as verifying if the appointment date and time are valid or if a doctor is available.
* Applied For loops to iterate over lists of appointments and doctors to process bulk updates and validations efficiently.
* Incorporated Try-Catch blocks for handling exceptions during appointment booking, such as catching errors when a double booking is attempted or invalid data is provided.

A screenshot of a computer

AI-generated content may be incorrect.

### 7. Batch Apex

* Created a Batch Apex class **AppointmentBatch** that processes Appointment records in bulk. It queries all confirmed appointments and iterates over them in manageable chunks.
* The batch processes appointment records asynchronously, allowing operations such as sending reminders or updating statuses withoutimpacting user experience or hitting Salesforce governor limits. It improves scalability by handling large data volumes efficiently.

A screenshot of a computer

AI-generated content may be incorrect.

### 8. Queueable Apex

* Developed a Queueable Apex class **(AppointmentQueueable)** that processes a list of Appointment records asynchronously. The class accepts a list of appointment Ids through its constructor, allowing flexible and targeted background processing.
* The Queueable Apex job processes appointment records in the background, enabling operations such as updating appointment status, sending notifications, or performing record audits for selected appointments.
* Handles large and specific sets of appointments without hitting limits.
* Supports flexible, scalable asynchronous logic beyond simple trigger flows.
* Makes the system robust for growing user and appointment data demands

A screenshot of a computer

AI-generated content may be incorrect.

### 9. Scheduled Apex

* Created a Scheduled Apex class **(AppointmentScheduler)** that automatically runs background jobs (such as batch or queueable processes) at defined intervals, like once a day.
* Scheduled batch jobs to run at predefined times.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

### 10. Future Methods

* Created an Apex future method to perform background tasks asynchronously, such as sending email or SMS notifications after booking or updating appointments.

A screenshot of a computer

AI-generated content may be incorrect.

### 11. Exception Handling

* Added try-catch blocks in Apex classes and triggers to manage errors during operations such as booking, updating, or deleting appointments.

A screenshot of a computer

AI-generated content may be incorrect.

### 12. Test Classes

* Developed Apex test classes like **AppointmentTriggerHandlerTest** (using @isTest) to thoroughly test triggers, classes, batch jobs, and asynchronous processes in the appointment booking solution.

A screenshot of a computer

AI-generated content may be incorrect.

### 13. Asynchronous Processing

* Leveraged Salesforce asynchronous processing features including **Batch Apex, Queueable Apex, Scheduled Apex, and Future Methods** to handle heavy-lifting tasks in the background.
* **Batch Apex** for high-volume record updates and reporting.
* **Queueable Apex** for flexible background tasks on selected records.
* **Scheduled Apex** for daily or weekly routine automation.
* **Future Methods** for non-blocking notification and integration logic.

A screenshot of a computer

AI-generated content may be incorrect.