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

In this phase, we move beyond Salesforce’s point-and-click tools and implement custom **Apex code** to handle complex business logic, scalability, and automation needs for the EV Charging CRM.

1. **Apex Classes & Objects**

* Apex classes are containers for reusable business logic.
* **Example :** Confirming a booking.

|  |
| --- |
| **public class BookingService {**  **public static void confirmBooking(Id bookingId) {**  **Booking\_\_c b = [SELECT Id, Status\_\_c FROM Booking\_\_c WHERE Id = :bookingId];**  **b.Status\_\_c = 'Confirmed';**  **update b;**  **}**  **}** |

1. **Apex Triggers**

* Triggers execute automatically when records are inserted, updated, or deleted.
* **Example:** Prevent double booking of the same charging slot.

|  |
| --- |
| trigger BookingTrigger on Booking\_\_c (before insert, before update) {  for (Booking\_\_c b : Trigger.new) {  Boolean exists = [  SELECT Id FROM Booking\_\_c  WHERE Station\_\_c = :b.Station\_\_c  AND Slot\_Time\_\_c = :b.Slot\_Time\_\_c  AND Id != :b.Id  LIMIT 1  ] != null;  if (exists) {  b.addError('This slot is already booked. Please choose another time.');  }  }  } |

1. **Trigger Design Pattern**

* Best practice is to move logic into handler classes, keeping triggers clean.

|  |
| --- |
| trigger BookingTrigger on Booking\_\_c (before insert, before update) {  BookingHandler.beforeSave(Trigger.new, Trigger.oldMap);  }  public class BookingHandler {  public static void beforeSave(List<Booking\_\_c> newList, Map<Id, Booking\_\_c> oldMap) {  // validation / logic here  }  } |

1. **SOQL & SOSL**

* **SOQL**: Query Salesforce records.
* **SOSL**: Search across multiple objects.

|  |
| --- |
| // SOQL  List<Customer\_\_c> customers = [SELECT Id, Name FROM Customer\_\_c WHERE Email\_\_c LIKE '%gmail.com'];  // SOSL  List<List<SObject>> results = [FIND 'John\*' IN ALL FIELDS RETURNING Customer\_\_c(Id, Name), Booking\_\_c(Id, Status\_\_c)]; |

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

* Used for handling bulk data.

|  |
| --- |
| List<String> stationNames = new List<String>();  Set<Id> customerIds = new Set<Id>();  Map<Id, Booking\_\_c> bookingMap = new Map<Id, Booking\_\_c>([SELECT Id, Status\_\_c FROM Booking\_\_c]); |

1. **Control Statements**

* Apex supports conditional logic similar to Java.

|  |
| --- |
| for (Booking\_\_c b : [SELECT Id, Status\_\_c FROM Booking\_\_c]) {  if (b.Status\_\_c == 'Pending') {  b.Status\_\_c = 'Confirmed';  }  } |

1. **Batch Apex**

* For large-scale data processing.

|  |
| --- |
| global class UpdateRevenueBatch implements Database.Batchable<SObject> {  global Database.QueryLocator start(Database.BatchableContext bc) {  return Database.getQueryLocator('SELECT Id, Amount\_\_c FROM Payment\_\_c');  }  global void execute(Database.BatchableContext bc, List<Payment\_\_c> scope) {  for (Payment\_\_c p : scope) {  p.Processed\_\_c = true;  }  update scope;  }  global void finish(Database.BatchableContext bc) {}  } |

1. **Queueable Apex**

* For smaller async jobs.

|  |
| --- |
| public class NotifyCustomerJob implements Queueable {  public void execute(QueueableContext context) {  // send email/notification  }  } |

1. **Scheduled Apex**

* Run jobs at specific times (e.g., daily reports).

|  |
| --- |
| global class DailyBookingReminder implements Schedulable {  global void execute(SchedulableContext sc) {  // logic to send reminders  }  } |

1. **Future Methods**

* For async callouts or heavy logic.

|  |
| --- |
| public class PaymentGateway {  @future(callout=true)  public static void processPayment(Id paymentId) {  // external API call  }  } |

1. **Exception Handling**

|  |
| --- |
| try {  // risky logic  } catch (Exception e) {  System.debug('Error: ' + e.getMessage());  } |

1. **Test Classes**

* Every Apex class/trigger must be covered by test classes for deployment.

|  |
| --- |
| @isTest  private class BookingTriggerTest {  @isTest static void testDoubleBooking() {  // insert bookings and assert addError works  }  } |

1. **Asynchronous Processing**

* **Future →** lightweight async jobs.
* **Batch →** large dataset processing.
* **Queueable →** chain jobs.
* **Scheduled →** run at specific times.