**Case Study on Ecommerce Application**

Submitted by: Mathavan T

Tech Stack: Python, MySQL

Project Type: Console-Based Application

# 1.INTRODUCTION

This project is a simple and effective terminal-based E-commerce Order Processing System. It is designed to simulate the basic operations of an online store including managing customers, products, carts, and orders. The system is built with Python and uses MySQL for database management.

The goal of this project is to demonstrate practical implementation of core software development concepts such as:

* Object-Oriented Programming (OOP)
* Data Access Object (DAO) design pattern
* Custom Exception Handling
* Modular Project Design
* Database Integration
* Unit Testing with Python

# 2.OBJECTIVES

* + To design a clean and modular backend for a shopping system
  + To apply Python programming using real-world object-oriented structures
  + To integrate MySQL for persistent storage of customers, products, orders
  + To implement exception handling and robust error reporting
  + To validate core features through unit tests

# 3.SYSTEM ARCHITECTURE & FLOW

The application starts with a simple menu that allows the user to perform actions such as:

1. Register a new customer

2. Add/delete products

3. Add/view/remove items from cart

4. Place and view orders

Each action internally uses:

* + Entity Classes to represent objects (e.g., Customer, Product)
  + DAO Interface & Implementation to access and manage data
  + Utility Classes to connect to MySQL
  + Custom Exceptions to handle invalid operations
  + Unit Tests to ensure code reliability

# 4. KEY MODULES

* entity/ - Contains Python classes for Customer, Product, Cart, Order, and OrderItem.
* dao/ - Defines the DAO interface and implements it using SQL statements to communicate with MySQL.
* util/ - Includes reusable functions for reading DB config and establishing database connections.
* myexceptions/ - Contains custom exception classes for missing data (e.g., ProductNotFoundException).
* app/EcomApp.py - Main menu-driven user interface. This is the starting point of the application.
* test/test\_cases.py - Includes test functions using Python’s unittest to verify core functionality.

# 5. Database Design

The system uses 5 main tables:

* + customers – stores customer information
  + products – stores available products
  + cart – holds products temporarily selected by the customer
  + orders – represents a completed order
  + order\_items – lists the items inside each order

All tables are connected using foreign keys for proper relational design.

# 6. TECHNICAL HIGHLIGHTS

* + Menu-driven interface for easy navigation
  + DAO separates logic from SQL
  + Centralized DB connection management
  + Reusable and extendable class structure
  + Unit-tested core operations
  + Modular and readable codebase

**8. UNIT TESTING**

Python’s built-in unit test module is used to validate all major operations of the application. The tests ensure that the system works correctly and handles errors gracefully.

The following test cases are implemented in test/test\_cases.py:

| **Test Case Description** | **Method Used** | **Expected Result** | **Status** |
| --- | --- | --- | --- |
| Test if product is created successfully | create\_product() | Product inserted into database | Pass |
| Test if product is added to cart successfully | add\_to\_cart() | Product appears in cart | Pass |
| Test if product is ordered successfully | place\_order() | Order stored, cart cleared | Pass |
| Test if exception is raised when customer/product not found | get\_orders\_by\_customer() or add\_to\_cart() | Raises custom exception | Pass |

# 7. CONCLUSION

This project not only simulates a mini shopping platform but also represents good software design. It’s organized, reliable, and easily extendable — suitable for backend processing and future frontend integrations.

This project is the result of applying the theoretical knowledge of OOP, databases, exception handling, and unit testing into a practical and functional solution.