Project Report: Daily Sales Record CRUD System

- **Student Name:** Praveen Gurjar
- **Registration Number:** 25BCG10032
- **Course Name:** Problem Solving and Programming
- **Course Code:** - CSE1021

Contents

1 Introduction

This report details the design and implementation of the Daily Sales Record CRUD System. The application, developed in Python, is a small-scale data management tool designed to modernize sales tracking for small businesses. It focuses on the core capability of performing Create, Read, Update, and Delete (CRUD)

operations on sales records, demonstrating the application of Object-Oriented Programming (OOP) principles and file persistence techniques learned in the course.

1.1 Project Objective

The goal is to develop a robust, modular system that replaces error-prone manual sales tracking methods with a digital solution offering reliable data management, basic filtering, and revenue summarization.

2 Problem Definition & Context

Manual sales tracking using spreadsheets or physical registers is inefficient for small businesses due to several factors: high susceptibility to data entry errors, difficulty in searching specific transaction details, and slow calculation times for essential financial summaries (e.g., total weekly revenue). The system addresses this by providing a structured and automated method for sales record management.

3 Functional Requirements (FRs)

The project is built around three major functional modules:

- **FR1: Sales Record Management (CRUD):** Complete functionality to add, view, modify, and remove sales records.
- **FR2: Search & Filtering:** Ability to filter the entire sales record list based on user-defined date ranges (Start Date to End Date).
- **FR3: Reporting & Summary:** Capability to calculate and display the total revenue ($) generated by the records currently filtered or viewed.

4 Non-functional Requirements (NFRs)

- **Usability:** The application employs a clear Command Line Interface (CLI) with guided menus and input prompts, ensuring ease of use without prior training.
- **Reliability:** All data is persistently stored in a local CSV file. The Data Access Layer ensures records are saved and loaded correctly to prevent data loss.
- **Error Handling:** Robust input validation is implemented to check for correct data types (e.g., numeric for price/quantity) and formats (e.g., YYYY-MM-DD for dates), providing informative error messages.
- **Maintainability:** The system is designed with a clear separation of concerns (Model, Logic, Persistence) using separate Python modules, facilitating easier code maintenance and future feature extensions.

5 System Architecture

The system utilizes a standard **3-Tier Architecture** model to ensure modularity and separation of business logic from the user interface and data storage mechanics.

1. **Presentation Layer (main_app.py):** Handles the user interface, command input, and output display.
2. **Business Logic Layer (sales_manager.py):** Processes all core functionalities, including CRUD, validation checks, and revenue calculations.
3. **Data Access Layer (sales_repository.py):** Manages data persistence by interacting directly with the CSV file.

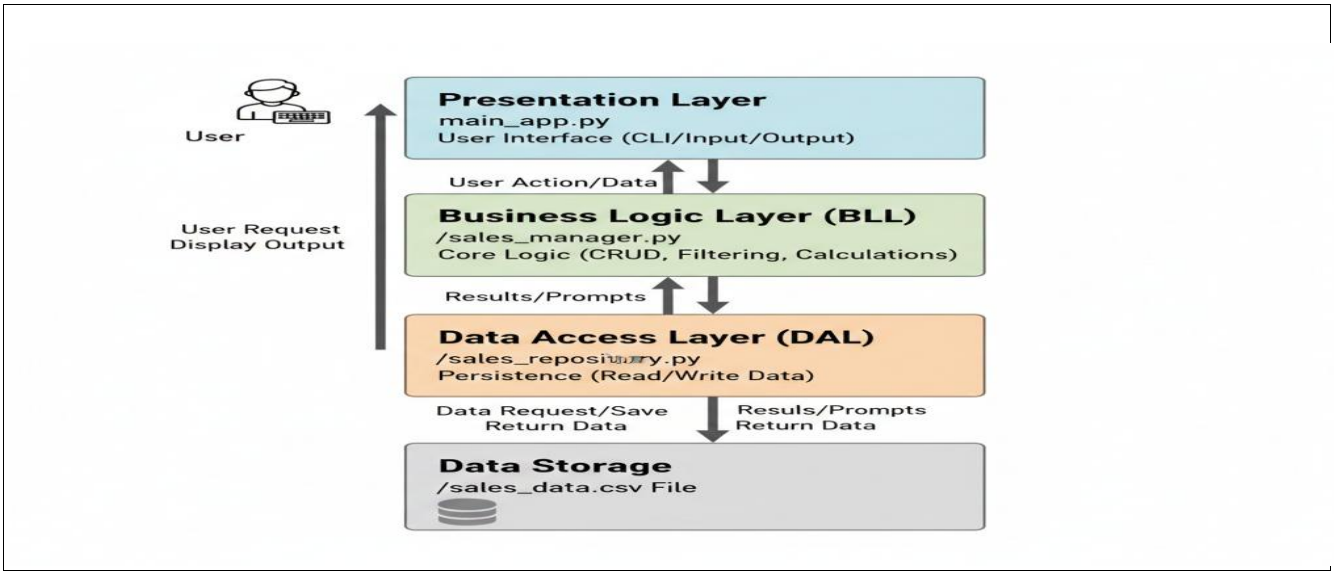

Figure 1: Visual representation of the 3-Tier System Architecture Diagram.

6 Design Diagrams

6.1 Use Case Diagram

The diagram illustrates the primary user goals, such as managing records, generating reports, and searching.

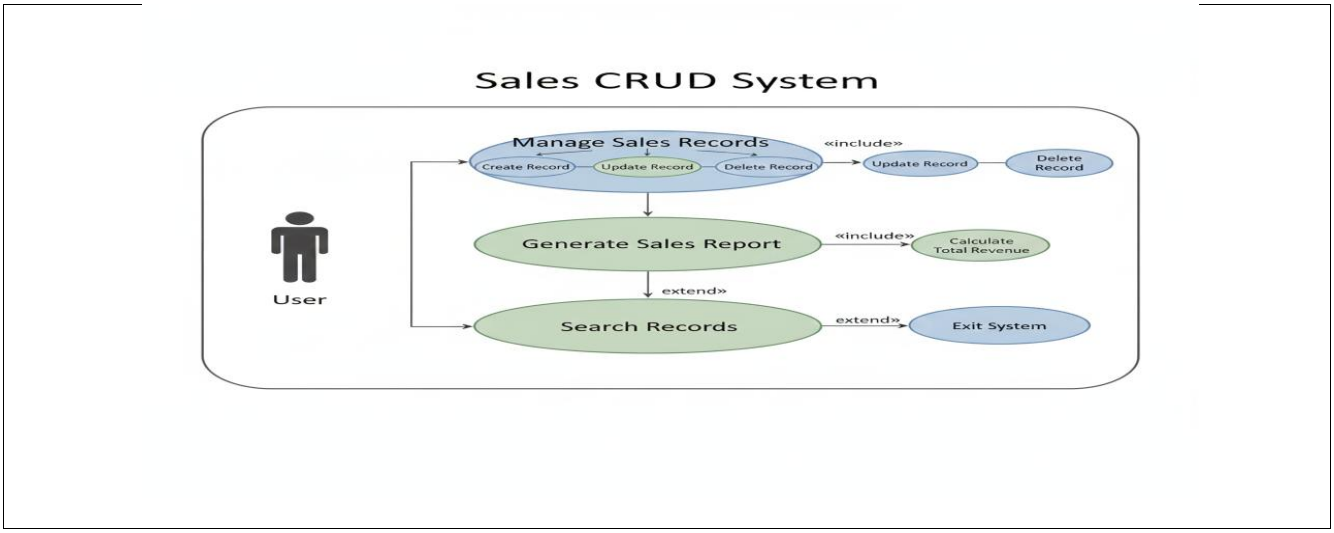

Figure 2: Visual representation of the Use Case Diagram for Sales CRUD System.

6.2 Workflow Diagram

This process flow maps the sequential steps required for the critical operation: **Creating a New Sales Record**, including the necessary validation loops.

6.3 Sequence Diagram

The sequence diagram demonstrates the flow of messages between the main components during the **Update Record** operation.

6.4 Class/Component Diagram

The diagram visualizes the structure of the implemented Python modules and classes, emphasizing dependencies.

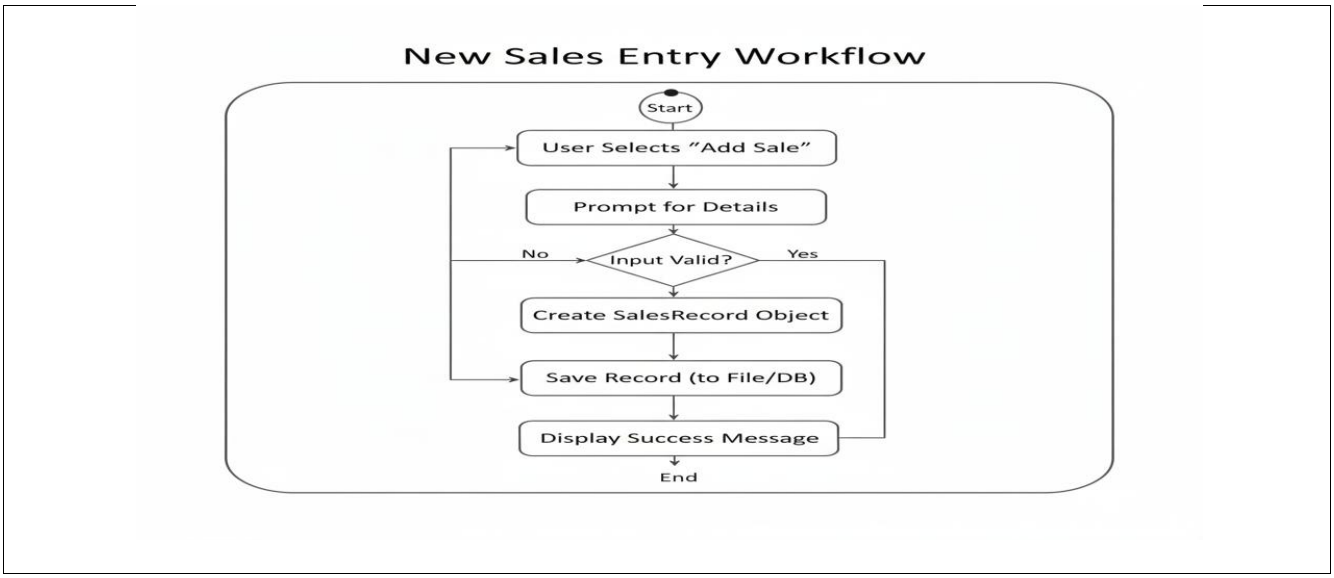

Figure 3: Visual representation of the Workflow Diagram for New Sales Entry.

6.5 Database/Storage Design (ER Diagram)

Since CSV file storage is used, the schema is defined by a single entity: Sales Record. **Sales Record Schema**

Sales Record ={SalesID, Date, ItemName, Quantity, UnitPrice, TotalAmount}

7 Design Decisions & Rationale

The primary decision was to use Python and CSV files. The rationale is that Python's clean syntax facilitates complex logic implementation, and the CSV file provides a simple, accessible form of data persistence that meets the reliability needs of a small, single-user system without requiring a heavy database installation. The three-layer modular structure ensures code quality and adherence to the maintainability NFR.

8 Implementation Details

The system is built using Python 3.11+, utilizing only standard libraries (e.g., csv, datetime). The codebase is organized into four main files within the src/ directory, ensuring clear separation of responsibilities:

- sales_record.py: Defines the data model.

- sales_repository.py: Handles file read/write (Persistence).

- sales_manager.py: Implements all business logic (CRUD, Filtering, Reporting).
- main_app.py: The application's entry point and CLI interface.

Execution is initiated via the command: python src/main_app.py.

9 Screenshots / Results

Screenshots demonstrating the working system have been placed in the dedicated /screenshots directory within the repository, including:

- Main application menu.
- Successful record addition and creation.
- Output of the date-range filtering and the calculated Total Revenue summary.

10 Testing Approach

A comprehensive testing approach was employed to validate the system:

- **UnitTesting:** Core functions, such as calculate_total_revenue() and ID generation, were tested manually to ensure mathematical accuracy and uniqueness.
- **Input Validation Testing:** Comprehensive tests were performed to verify that the CLI rejects incorrect data types for numerical fields (e.g., text for price) and enforces the correct YYYY-MM-DD format for dates.
- **Data Integrity Testing:** The persistence mechanism was tested by adding, deleting, and updating records, followed by reloading the application to confirm data fidelity in the CSV file.

11 Challenges Faced

The main challenges were centered on ensuring robust data handling when using file I/O:

- **Maintaining Unique IDs:** Generating a unique, persistent ID for new records in a CSV environment required implementing a method to load all records and calculate the maximum existing ID before assigning the next one.
- **Robust Date Parsing:** Handling date filtering required careful conversion of user input strings into standardized datetime objects to ensure accurate range comparisons.

12 Learnings & Key Takeaways

- Gained practical mastery over **OOP principles** by mapping business requirements(sales management) to distinct software components (classes).
- Learned the complexities of **data persistence** in a file-based environment, particularlyaround serialization/deserialization and error handling during file read/write.
- Reinforced the value of **modular architecture** as it significantly simplified the testingand maintenance of individual system components.

13 Future Enhancements

Potential extensions to the system include:

- Migrating the CLI to a modern Graphical User Interface (GUI) to boost usability.
- Implementing user authentication and basic role management (Admin/Clerk) for security.
- Integrating a library like Pandas for advanced data analysis and graphical reporting features.