

**FINAL REPORT**

**PHARMACEUTICALS MANAGEMENT SYSTEM**

**CSM 216**

**K23CH**

**AARNAV SURESH**

**12310230**

**18/11/24**

**ACKNOWLEDGMENT**

I, Aarnav Suresh, would like to express my heartfelt gratitude to my teacher, Aman, for his constant guidance, valuable feedback, and unwavering support throughout the development of this Pharmaceutical Management System project. His expertise and encouragement have been a significant factor in the completion of this work.

I would also like to thank Lovely Professional University for providing the necessary resources and a conducive environment for learning, which helped me accomplish this project.

This project would not have been possible without the help and support of my teacher and the university, and I am sincerely grateful for all their assistance.

**TABLE OF CONTENTS**

1. INTRODUCTION
2. OBJECTIVES AND SCOPE OF THE PROJECT
3. APPLICATION TOOLS
4. PROJECT DESIGN
5. FLOWCHART
6. PROJECT IMPLEMENTATION
7. TESTING AND VALIDATION
8. CONCLUSION
9. REFERENCE

**INTRODUCTION**

The Pharmaceutical Management System is designed to streamline the management of drugs and suppliers in the pharmaceutical industry. The purpose of this project is to create a user-friendly software solution using Python and Tkinter, which helps organizations effectively manage their inventory, including drug information, quantities, prices, and supplier details.

The significance of this system lies in its ability to improve the efficiency of pharmaceutical operations by providing a centralized platform for storing, updating, and retrieving data related to drugs and suppliers. This reduces the time and effort spent on manual processes, minimizes errors, and enhances overall operational efficiency.

The objective of this project is to develop a comprehensive solution for drug management, allowing users to add, update, view, and delete drug and supplier information in a simple, intuitive interface. This system aims to address the challenges faced by pharmaceutical companies in maintaining an organized database, ensuring smooth workflow, and supporting data-driven decision-making.

**OBJECTIVES / SCOPE OF THE PROJECT**

**Objectives:**

1. **Drug Management:** To develop a system that allows users to add, view, update, and delete information about drugs, including details such as drug name, quantity, and price.
2. **Supplier Management:** To create a feature that enables users to manage suppliers, including adding, viewing, updating, and deleting supplier information such as name and contact details.
3. **User-Friendly Interface:** To design an intuitive and easy-to-use graphical user interface (GUI) using Tkinter, which simplifies interaction with the system.
4. **Data Integrity:** To ensure that all operations performed (such as adding, updating, or deleting records) are accurate, with appropriate error handling to prevent incorrect data entry or deletion.
5. **Efficient Data Handling:** To provide a method for storing and managing data in a way that ensures quick access and minimal manual work for the user.

**Scope:**

* **Functionality:** The system will focus on the management of drugs and suppliers, which includes the addition, viewing, updating, and deletion of records for both entities.
* **User Roles:** The system will be designed for individual users who need to manage drug and supplier data in a pharmaceutical environment.
* **Technology:** The system will be developed using Python, Tkinter for the graphical user interface (GUI), and an in-memory dictionary structure for storing drug and supplier data.
* **Limitations:** This system is intended for small-scale use and does not include advanced features like cloud storage or user authentication. The data is stored temporarily and will be lost once the application is closed. Future versions could address these limitations by integrating database management systems and additional functionalities.

**Intended Outcomes:**

1. A fully functional Pharmaceutical Management System that simplifies drug and supplier management for users.
2. A robust and error-resistant application that ensures proper data handling and provides an efficient workflow for managing pharmaceutical information.
3. A scalable foundation for future enhancements, such as integrating persistent data storage and expanding the system’s capabilities.

**APPICATION TOOLS**

**Tools, Software, and Libraries Used**

1. **Programming Language:**
   * **Python**: The core programming language used to develop the Pharmaceutical Management System. Python was chosen for its simplicity, readability, and vast library support, making it ideal for rapid development.
2. **IDE (Integrated Development Environment):**
   * **PyCharm**: A powerful and feature-rich IDE for Python development, which offers tools such as debugging, code completion, and project management.
   * **Visual Studio Code**: Another popular code editor used for Python development, known for its extensibility and support for various Python extensions.
3. **Libraries/Packages:**
   * **Tkinter**: A standard Python library used for building the graphical user interface (GUI) of the application. Tkinter was used to create windows, buttons, labels, and other interactive elements.
   * **ttk (Themed Tkinter)**: A sub-library within Tkinter used to enhance the appearance of the GUI components like tables (Treeview) and buttons, providing a more polished and modern look.
4. **Version Control:**
   * **Git**: A version control system used to track changes in the project’s codebase. Git helps in maintaining a history of changes, collaborating with others, and reverting to previous versions if needed.
   * **GitHub**: A cloud-based platform used for hosting the Git repository, enabling easy access to the project from different devices and allowing collaboration if necessary.

**PROJECT DESIGN**

The Pharmaceutical Management System is structured around a graphical user interface (GUI) built using Tkinter. The main components of the system include the core application (PharmaApp), various windows for managing drugs and suppliers, and supporting functionality for adding, updating, viewing, and deleting records.

The structure follows an object-oriented design, with separate classes representing different windows and functionalities within the system.

**Main Components and Classes**

1. **PharmaApp (Main Application Class):**
   * **Purpose:** This is the core of the application, initializing the main window and providing buttons for navigating to different functions like adding, viewing, updating, and deleting drugs and suppliers.
   * **Key Functions:**
     + create\_widgets(): This function sets up the main interface with buttons for each operation (Add, View, Update, Delete) related to both drugs and suppliers.
     + **Navigation Methods**: open\_add\_drug\_window(), open\_view\_drugs\_window(), open\_update\_drug\_window(), open\_delete\_drug\_window(), open\_add\_supplier\_window(), open\_view\_suppliers\_window(), open\_update\_supplier\_window(), open\_delete\_supplier\_window(). These methods open respective windows to perform operations on drugs and suppliers.
2. **AddDrugWindow (Add Drug Window):**
   * **Purpose:** This class creates a window that allows the user to input details for adding a new drug (name, quantity, price).
   * **Key Functions:**
     + add\_drug(): Takes input from the user, assigns a unique drug ID, and stores the drug in the drugs dictionary in the PharmaApp class.
3. **ViewDrugsWindow (View Drugs Window):**
   * **Purpose:** Displays all the drugs stored in the system in a table format using the Treeview widget.
   * **Key Functions:**
     + load\_data(): Iterates through the drugs dictionary and populates the Treeview with details of each drug.
4. **UpdateDrugWindow (Update Drug Window):**
   * **Purpose:** Allows the user to update details of an existing drug (such as name, quantity, and price) based on the drug ID.
   * **Key Functions:**
     + update\_drug(): Checks if the entered drug ID exists and updates the details of the drug in the drugs dictionary.
5. **DeleteDrugWindow (Delete Drug Window):**
   * **Purpose:** Lets the user delete a drug from the system based on its ID.
   * **Key Functions:**
     + delete\_drug(): Deletes the drug from the drugs dictionary if the provided ID matches an existing drug.
6. **AddSupplierWindow (Add Supplier Window):**
   * **Purpose:** Provides a window for adding new supplier information (name, contact info).
   * **Key Functions:**
     + add\_supplier(): Takes the input data, generates a unique supplier ID, and stores the supplier in the suppliers dictionary.
7. **ViewSuppliersWindow (View Suppliers Window):**
   * **Purpose:** Displays all the supplier information in a table format.
   * **Key Functions:**
     + load\_data(): Iterates through the suppliers dictionary and populates the Treeview with the supplier details.
8. **UpdateSupplierWindow (Update Supplier Window):**
   * **Purpose:** Lets the user update supplier details (name, contact info) based on the supplier ID.
   * **Key Functions:**
     + update\_supplier(): Checks if the entered supplier ID exists and updates the supplier's information in the suppliers dictionary.
9. **DeleteSupplierWindow (Delete Supplier Window):**
   * **Purpose:** Allows the user to delete a supplier from the system using their supplier ID.
   * **Key Functions:**
     + delete\_supplier(): Deletes the supplier from the suppliers dictionary if the provided ID matches an existing supplier.

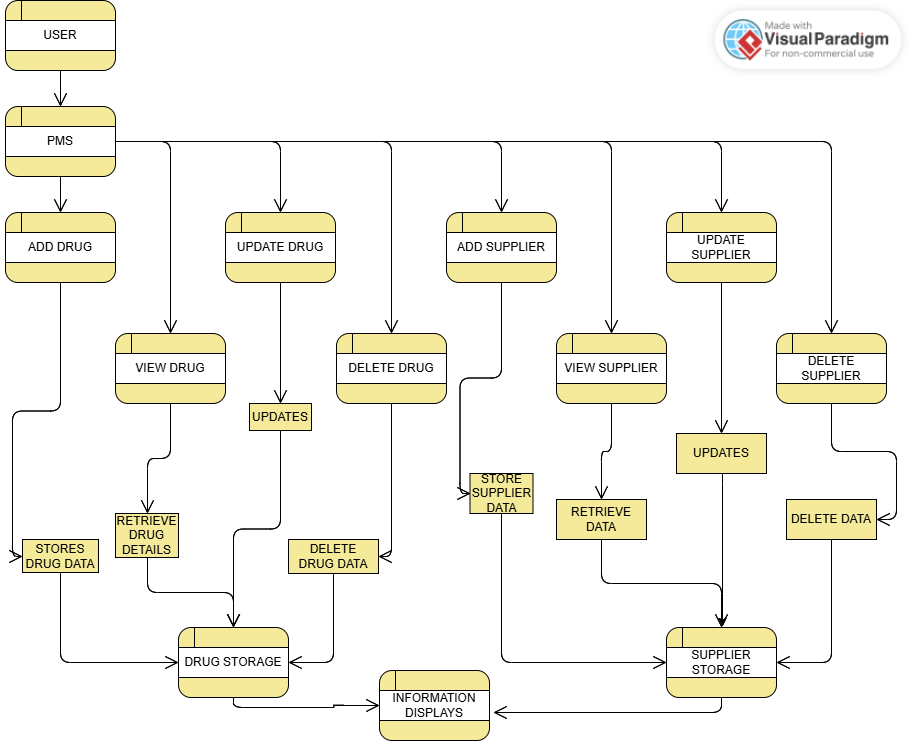
**How the Components Interact**

1. **Data Storage:** The drug and supplier data are stored in dictionaries (self.drugs and self.suppliers) in the PharmaApp class. Each entry is associated with a unique ID, and each drug or supplier is represented as a dictionary containing relevant attributes (name, quantity, price for drugs; name, contact info for suppliers).
2. **Interaction Between Classes:**
   * When a user clicks a button in the main window (PharmaApp), it opens the appropriate window (e.g., Add Drug, View Drugs, etc.) by calling one of the navigation methods.
   * Each window (e.g., AddDrugWindow, UpdateDrugWindow) handles specific operations (add, view, update, delete) for either drugs or suppliers.
   * For instance, when adding a new drug, the AddDrugWindow gets the details from the user, creates a new drug entry, and stores it in the drugs dictionary. Similarly, the ViewDrugsWindow retrieves and displays data from the drugs dictionary.
3. **Error Handling and Success Messages:** After each action (adding, updating, deleting), the system uses message boxes (via messagebox.showinfo() or messagebox.showerror()) to notify the user of the success or failure of the operation.

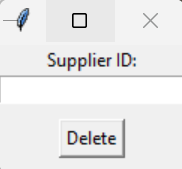
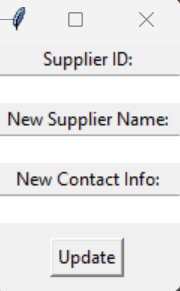
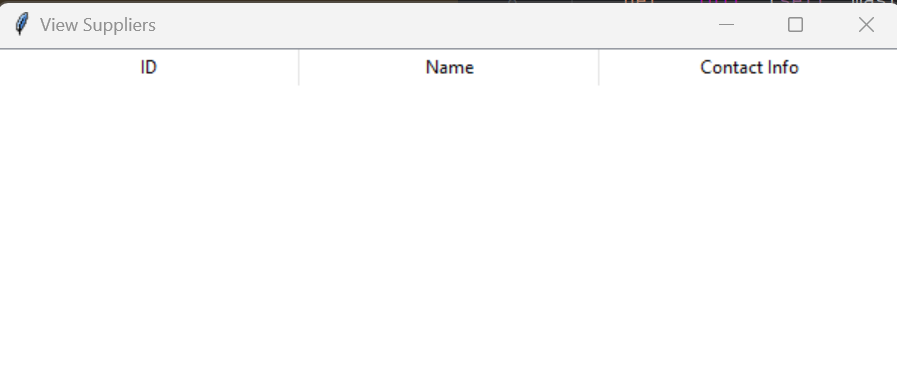
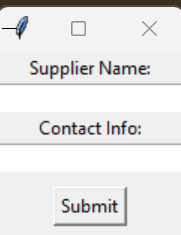
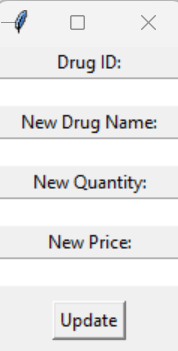
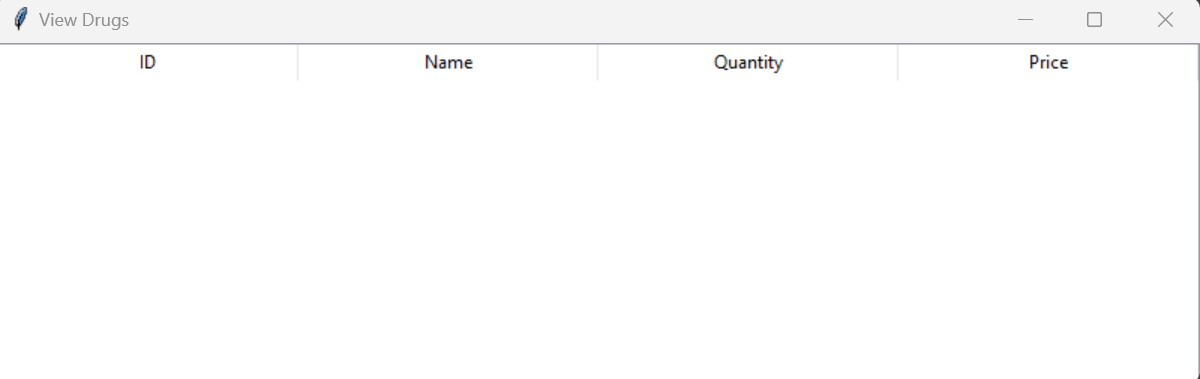
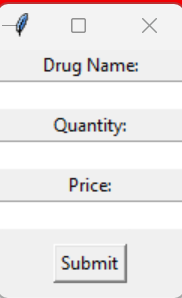
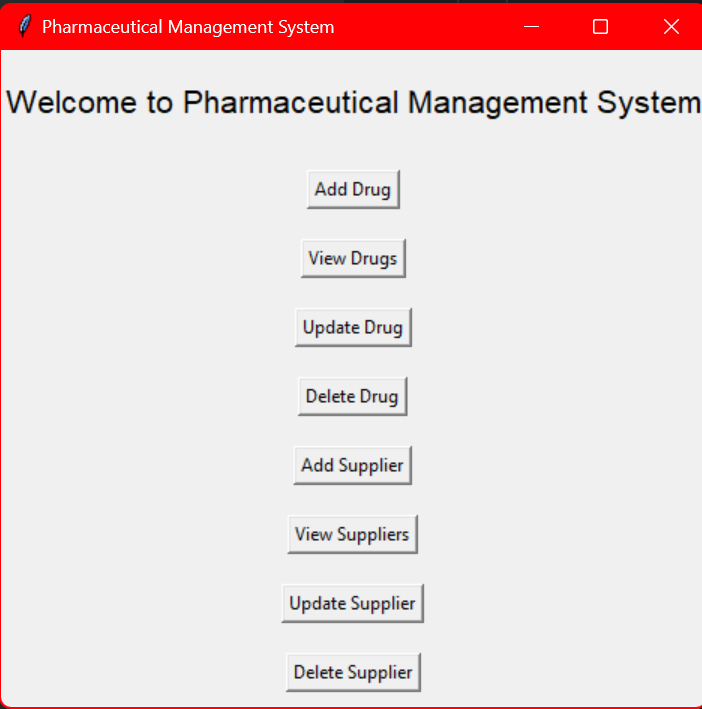
**Overall Flow:**

1. **Step 1:** The user starts the application by running PharmaApp, which displays the main menu with buttons for each operation.
2. **Step 2:** The user clicks a button (e.g., Add Drug), which opens the relevant window (e.g., AddDrugWindow).
3. **Step 3:** The user performs the action (e.g., adds a drug), and the data is processed and stored in the respective dictionary (drugs or suppliers).
4. **Step 4:** The user can view, update, or delete records, and the system reflects the changes accordingly.

**FLOWCHART**



**PROJECT IMPLEMENTATION**



**TESTING/VALIDATION**

**1. Unit Testing**

* **Purpose**: Verify that individual components or modules of the system function as expected.
* **What was tested**:
  + **Drug Management Functions**:
    - Adding a drug.
    - Viewing drug details.
    - Updating drug details.
    - Deleting a drug.
  + **Supplier Management Functions**:
    - Adding a supplier.
    - Viewing supplier details.
    - Updating supplier details.
    - Deleting a supplier.
* **Approach**:
  + Mock data was used to simulate user inputs.
  + Each function was tested independently to ensure accuracy.
  + Boundary conditions, such as invalid drug IDs or negative quantities, were tested.

**2. Integration Testing**

* **Purpose**: Ensure that different modules interact correctly when integrated.
* **What was tested**:
  + Interactions between drug and supplier modules.
  + Data retrieval and updates to shared storage (e.g., viewing a drug immediately after adding it).
  + User interface interactions with backend logic.
* **Approach**:
  + Tested the flow of data between UI elements (buttons, text fields) and backend functions.
  + Verified that adding or updating drugs/suppliers reflects accurately in the storage and UI.
* **Tools**: Manual testing with real input scenarios and Python scripts for automated testing.

**3. System Testing**

* **Purpose**: Validate the entire application against the specified requirements.
* **What was tested**:
  + End-to-end workflows, such as adding a drug, viewing it, updating it, and deleting it.
  + Performance under normal and high-load conditions.
  + User experience across different functionalities.
* **Approach**:
  + Conducted black-box testing to verify the system’s behavior without delving into the code.
  + Ensured all features worked together seamlessly.
* **Tools**: Manual testing with real-world scenarios.

**4. User Acceptance Testing (UAT)**

* **Purpose**: Ensure the system meets user expectations and requirements.
* **What was tested**:
  + Ease of use for adding, viewing, updating, and deleting drugs and suppliers.
  + Error handling and feedback messages (e.g., for invalid inputs).
* **Approach**:
  + Gathered feedback from peers or instructors acting as end-users.
  + Verified that the system is intuitive and free of critical errors.
* **Outcome**: Adjusted the user interface and error messages based on feedback to enhance usability.

**5. Regression Testing**

* **Purpose**: Ensure that new updates or changes do not break existing functionalities.
* **What was tested**:
  + Retested all core functionalities after making changes or enhancements.
* **Approach**:
  + Re-ran previously successful test cases after updates.
  + Verified that dependencies between components remained intact.
* **Tools**: Automated scripts using Python where feasible.

**CONCLUSION**

**Key Accomplishments**

1. **Comprehensive Drug and Supplier Management**
   * Successfully implemented functionalities for adding, viewing, updating, and deleting both drugs and suppliers.
   * Maintained clear and efficient data storage to handle information systematically.
2. **User-Friendly Interface**
   * Designed an intuitive GUI using Tkinter that allows users to interact with the system seamlessly.
   * Provided clear feedback messages to users for better usability and understanding.
3. **Robust Functionality**
   * Included error handling for invalid inputs, such as incorrect drug or supplier IDs.
   * Supported real-time updates, ensuring that changes made are instantly visible across different modules.
4. **Scalable Design**
   * Modular structure with classes for each functionality, making it easy to maintain and expand in the future.
   * A strong foundation for integrating advanced features like search, filters, or database integration.
5. **Testing and Validation**
   * Conducted unit, integration, system, and user acceptance testing to ensure the system’s reliability and correctness.
   * Ensured consistent performance and accuracy across various scenarios.

**Limitations**

1. **Local Data Storage**
   * Currently, data is stored in memory, meaning all data is lost when the application is closed.
   * No persistent storage like a database or file system is implemented.
2. **Limited Features**
   * The system lacks advanced features such as inventory alerts, supplier communication, or data export options.
   * No user authentication mechanism to control access or secure sensitive data.
3. **Performance for Large Data Sets**
   * While suitable for small-scale use, performance may degrade as the number of drugs and suppliers grows.
   * No optimization for handling large inventories efficiently.
4. **Lack of Cross-Platform Testing**
   * The application has been tested primarily on local systems and may not be optimized for diverse environments or operating systems.

**Significance and Impact**

The **Pharmaceutical Management System** simplifies drug and supplier management for small-scale pharmaceutical operations. It reduces manual record-keeping efforts, streamlines data access, and improves operational efficiency. By automating repetitive tasks, it allows users to focus on higher-value activities, such as customer interaction or inventory planning.

The system lays the groundwork for more sophisticated tools that could support larger-scale operations. Its modular design and intuitive interface make it an excellent learning tool for software development, particularly in GUI-based application design.

**Future Scope**

* **Integration with Databases**: Implementing a persistent storage solution such as MySQL or SQLite for long-term data retention.
* **Advanced Features**: Adding inventory alerts, data analytics, and report generation.
* **Security Enhancements**: Introducing user authentication and encryption for data protection.
* **Scalability**: Optimizing the application for handling larger datasets and expanding functionalities.

By addressing these limitations and expanding its features, the system can evolve into a robust and scalable solution for pharmaceutical management.

**REFERENCES**

**References (Main Points)**

1. **Python Documentation** – Official Python guide for programming concepts.
2. **Tkinter Documentation** – Resource for GUI design.
3. **Object-Oriented Programming** – Used concepts from *Python Object-Oriented Programming* by Dusty Phillips.
4. **Testing Framework** – Used Python’s unittest module for testing.
5. **Pharmaceutical Management Research** – Insights from online resources and ResearchGate articles.
6. **Guidance and Peer Support** – Inputs from **Mr. Aman** (Lovely Professional University) and peers for feedback and improvements.