PROJECT REPORT

ON

CLI-Based Personal Task Manager Application

SUBMITTED BY: SARTAK JINDAL

REG NO:-25BCE10189

TECHNOLOGY USED: Python, SQLite3

TABLE OF CONTENTS

---

1. ABSTRACT

The Personal Task Manager is a command-line interface (CLI) application designed to help individuals organize their daily activities effectively. Built using the Python programming language, the application leverages SQLite, a lightweight disk-based database, to ensure data persistence. The system allows users to create secure accounts, log in, and manage their to-do lists. Key features include adding tasks with priorities and due dates, viewing active tasks sorted by urgency, marking tasks as complete, and deleting tasks. This project

demonstrates the fundamental principles of CRUD (Create, Read, Update, Delete) operations, user session management, and database integration in a Python environment.

2. INTRODUCTION

In today's fast-paced world, effective time management is crucial. While many complex graphical applications exist for task management, there is often a need for lightweight, fast, and distraction-free tools.

This project provides a terminal-based solution where users can quickly interact with their task lists without the overhead of a Graphical User Interface (GUI). It is designed for users who prefer keyboard-driven workflows and need a reliable way to store their tasks locally on their machine. The application ensures that tasks are tied to specific users, maintaining privacy even on a shared computer.

3. PROBLEM STATEMENT AND OBJECTIVES

Problem Statement

Managing tasks via mental notes or scattered paper scraps often leads to forgotten deadlines and poor prioritization. Users need a centralized, digital system to store tasks that persists even after the computer is turned off, allowing them to categorize tasks by urgency (priority) and deadlines (due dates).

Objectives

The primary objectives of this project are:

1. To develop a functional application using Python that runs in the console/terminal.
2. To implement a persistent storage system using SQLite3 to save user data and tasks.
3. To create a secure user authentication system (Registration and Login) to ensure data privacy.
4. To implement core task management functionalities: Adding, Viewing, Updating status, and Deleting tasks.
5. To implement logic that automatically sorts tasks based on priority (High > Medium > Low) and impending due dates.

4. SYSTEM ANALYSIS AND DESIGN

4.1 Technology Stack

- **Programming Language:** Python 3.x (Chosen for its readability and ease of handling string operations and database connections).
- **Database:** SQLite3 (Chosen because it is serverless, requires zero configuration, and stores data in a single local file, making it ideal for a desktop application).
- **Standard Libraries:**
    - sqlite3: For database interaction.
    - datetime: For handling timestamps when creating tasks.

4.2 Database Design (Schema)

The application uses a relational database model named sarthak.db. It consists of two related tables:

Table 1: Users (users)

This table stores user credentials.

Column Name	Data Type	Constraints	Description
user_id	INTEGER	PRIMARY KEY, AUTOINCREMENT	Unique identifier for the user.
username	TEXT	UNIQUE, NOT NULL	The login name chosen by the user.
password	TEXT	NOT NULL	The user's password (stored as plain text for simplicity in this iteration).

Table 2: Tasks (tasks)

This table stores the actual to-do items and links them to a user.

Column Name	Data Type	Constraints	Description
task_id	INTEGER	PRIMARY KEY, AUTOINCREMENT	Unique identifier for the task.
user_id	INTEGER	FOREIGN KEY (references users)	Links the task to the specific user who created it.

Column Name	Data Type	Constraints	Description
description	TEXT	NOT NULL	The details of the task itself.
priority	TEXT	NOT NULL	The urgency level (High, Medium, Low).
due_date	TEXT	NOT NULL	The deadline date (format YYYY-MM-DD).
created_date	TEXT	NOT NULL	The date the task was added into the system.
is_complete	INTEGER	DEFAULT 0	Status flag: 0 for Active, 1 for Complete.

---

5. IMPLEMENTATION DETAILS AND FEATURES

The application is structured around a main loop that presents different menus based on the user's authentication state (tracked by a global variable LOGGED_IN_USER_ID).

Key Features:

1. Database Initialization (setup() function)

On application launch, the code connects to SQLite. It executes SQL CREATE TABLE IF NOT EXISTS statements. This ensures that if the database file doesn't exist, it is created along with the required table structures automatically.

2. User Registration and Login

- **Registration:** Accepts a username and password. Tries to INSERT them into the users table. A try...except sqlite3.IntegrityError block handles cases where a username already exists.
- **Login:** Accepts credentials and performs a SELECT query with a WHERE clause matching both username and password. If a match is found, the global session variable is updated with the retrieved user_id.

3. Adding Tasks (add_task() function)

Only accessible when logged in. It prompts for task details. The current date is automatically calculated using datetime.now(). An SQL INSERT query saves the task into the tasks table, linking it using the current LOGGED_IN_USER_ID.

4. Viewing Tasks with Sorting Logic (view_tasks() function)

This is the core display feature. It fetches active tasks (is_complete = 0) for the logged-in user.

Crucial Logic: The SQL query uses a customized ORDER BY clause:

SQL

```
ORDER BY
  CASE priority
    WHEN 'High' THEN 1
    WHEN 'Medium' THEN 2
    WHEN 'Low' THEN 3
    ELSE 4
  END,
  due_date ASC
```

This ensures that "High" priority tasks always appear at the top, followed by Medium and Low. Within the same priority level, tasks with sooner due dates appear first.

5. Task Completion and Deletion

- **Mark Complete:** The user provides a Task ID. The system runs an SQL UPDATE query to change is_complete from 0 to 1. It includes a check (AND user_id = ?) to ensure users can only update their own tasks.
- **Delete:** An SQL DELETE query removes a specific record based on Task ID and User ID. A "Delete All" feature allows wiping all tasks tied to the current user ID.

---

6. RESULT AND OUTPUT SCREENSHOTS (SIMULATION)

Since this is a text-based report, standard CLI outputs are simulated below to represent program execution.

Scenario 1: Initial Launch and Registration

Plaintext

Database setup complete.

--- WELCOME ---

1. Register

2. Login

3. Exit

Enter choice: 1

Username: sarthak_dev

Password (simple text): securepass123

User 'sarthak_dev' registered successfully!

Scenario 2: Logging in and Adding Tasks

Plaintext

--- WELCOME ---

1. Register

2. Login

3. Exit

Enter choice: 2

Username: sarthak_dev

Password: securepass123

Login successful. Welcome, sarthak_dev!

--- TASK MANAGER ---

1. View Active Tasks

2. Add New Task

3. Mark Task Complete

4. Delete Specific Task

5. Delete ALL My Tasks

6. Logout

Enter choice: 2

Task Description: Finish Project Report

Priority (High/Medium/Low): High

Due Date (YYYY-MM-DD): 2023-11-25

Task added successfully!

(Assume another Medium priority task is added subsequently)

Scenario 3: Viewing sorted tasks

Plaintext

Enter choice: 1

--- YOUR TO-DO LIST ---

ID | Priority | Due Date | | Task Description

---

1 | High | 2023-11-25 | ACTIVE | Finish Project Report

2 | Medium | 2023-11-30 | ACTIVE | Submit code review

---

7. LIMITATIONS AND FUTURE SCOPE

Limitations

1. **Security Risk:** Passwords are currently stored in plain text in the database. In a production environment, this is a severe security vulnerability. They should be hashed using libraries like bcrypt.
2. **Input Validation:** The application assumes users enter dates in the correct 'YYYY-MM-DD' format and select valid priority levels. Robust error handling for incorrect input formats is missing.
3. **User Interface:** The command-line interface, while functional, may not be intuitive for non-technical users compared to a GUI.

Future Scope

1. **Graphical User Interface (GUI):** The application could be ported to use Tkinter or PyQt to provide buttons, calendars for date selection, and visual color coding for priorities.
2. **Password Hashing:** Implement secure password hashing during registration and login procedures.
3. **Task Categories/Tags:** Add a feature to categorize tasks (e.g., "Work", "Personal", "Shopping") for better organization.
4. **Search Functionality:** Allow users to search for tasks by keywords in the description.

8. CONCLUSION

The CLI-Based Personal Task Manager project successfully implements a functional CRUD application using Python and SQLite. It effectively meets the objective of providing a persistent, multi-user system for organizing daily tasks. The project demonstrates a solid understanding of database schema design, SQL query integration within Python code, and managing application state through user sessions. While the current version has limitations regarding security and input validation, it serves as a strong foundational framework that can be expanded into a more robust and feature-rich application in the future.