|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | |
|  | | PYTHON PROJECT | | | | |  | |
|  |  | | | | | | |  |
|  | | | |  |  | | | |
| **TO-DO LIST APPLICATION** | | | |  |  | | | |
|  | | | |  |  | | | |
|  | | |  | | |  | | |

- Submitted by: Rakshitha H U

- Institution Name: Kalpataru Institute Of Technology, Tiptur

- Date of Submission: 25/11/2024

- Guidance by: Learntricks Edutech

Abstract

#### Task management is essential for productivity and efficient time use. The Python To-Do List Application is a lightweight, user-friendly tool designed to help users organize tasks effectively. It enables users to add, update, delete, and categorize tasks, manage priorities and deadlines, and track completed tasks, promoting better time management and goal-oriented workflows.

#### **Key Features**

1. **Task Management**: Add, edit, delete, and mark tasks as completed.
2. **Categorization**: Group tasks into categories like Work or Personal for better organization.
3. **Priority Levels**: Assign Low, Medium, or High priority to focus on critical tasks.
4. **Deadline Tracking**: Set and monitor due dates for timely completion.
5. **User Interface**: Offers a simple command-line tool or intuitive graphical interface.

#### **Technical Overview**

* **Python**: The core programming language used for the application.
* **Tkinter**: Provides a graphical user interface (GUI) for enhanced usability.
* **SQLite**: Ensures persistent and reliable storage for tasks.
* **Core Logic**: Implements functionality for task categorization, priority management, and deadlines.

The Python To-Do List Application is a robust and scalable tool designed to support individuals and professionals in achieving their goals efficiently and effectively.

Table of Contents

1. Introduction

2. Objectives

3. Literature Survey

4. System Design

5. Implementation

6. Testing and Validation

7. Results and Discussion

8. Challenges and Limitations

9. Conclusion

10. Future Scope

11. References

12. Appendices (if applicable)

INTRODUCTION

Effective task management is a cornerstone of productivity, helping individuals and teams achieve their goals efficiently. In today’s fast-paced world, managing daily tasks can become overwhelming, leading to missed deadlines and disorganized workflows. To address this challenge, task management tools have emerged as essential aids for organizing and prioritizing activities effectively.

The motivation for this project stems from the growing demand for simple, customizable, and user-friendly tools that can streamline task management for personal and professional use. While several task management applications exist, many are either overly complex or lack flexibility, creating a need for a lightweight and accessible solution.

The Python To-Do List Application is designed to fulfill this need by providing a straightforward yet efficient platform for managing tasks. The application allows users to add, update, delete, and categorize tasks, set priorities, and track deadlines. By offering these features in a user-friendly interface, it ensures that task management becomes seamless and intuitive, fostering better time management and productivity.

### **Objectives**

The primary objectives of the Python To-Do List application are:

1. **To develop a simple, user-friendly To-Do List application**:  
   The application is designed to provide a hassle-free experience, ensuring users can manage their tasks with minimal effort and maximum efficiency.
2. **To enable users to add, delete, update, and view tasks**:  
   By offering core functionalities, the application ensures that users can handle their tasks effectively, whether it's adding new ones, modifying existing ones, or reviewing their progress.
3. **To implement features like task categorization and deadline tracking**:  
   Advanced features such as categorizing tasks and setting deadlines are incorporated to help users organize their tasks systematically and complete them on time, enhancing productivity.

### 

LITERATURE SURVEY

Recent research and discussions around Python-based applications, including to-do list tools, highlight the growing interest in Python for its simplicity and adaptability in real-world applications:

1. **Python's Popularity**: Python has emerged as the most widely used programming language on platforms like GitHub, with a sharp increase in applications across domains such as task management, AI, and data science. Its ease of use and wide array of libraries make it a preferred choice for creating productivity tools like to-do list applications【37†source】【38†source】.
2. **Automation and Customization**: Python is commonly used to develop automation tools for workflow management. These tools often utilize libraries like Tkinter for graphical interfaces and SQLite for database integration, similar to features of a to-do list application【36†source】【38†source】.
3. **Task Management Innovations**: Research in productivity tools shows that integrating features such as task prioritization, categorization, and deadline management in applications improves user experience. Many Python projects also leverage open-source libraries for customization, making them versatile alternatives to commercial tools like Trello and Todoist【37†source】.
4. **Simplification and Accessibility**: Python-based to-do list applications are often simpler and more lightweight than enterprise solutions, making them accessible for personal use and adaptable for small teams or individual developers. These applications highlight how Python can address limitations in existing tools, such as complexity or high costs, by focusing on core functionalities【37†source】【36†source】.

If you're interested in diving deeper into recent studies and developments, resources like IEEE Xplore and GitHub Octoverse reports offer detailed insights into Python's applications and innovations in task management.

### SYSTEM DESIGHN

### 1 Architecture

### Description of the System Architecture

### The Python To-Do List application can be implemented using two main architectural designs:

### • GUI-Based Architecture: Using libraries such as Tkinter or PyQt for creating a graphical user interface, which provides a user-friendly interaction layer.

### • CLI-Based Architecture: Implementing a command-line interface for simplicity and ease of development, ideal for basic task management needs.

### The architecture includes three core layers:

### 1. User Interface Layer: Manages user interaction, either through GUI or CLI.

### 2. Logic Layer: Handles core operations like adding, editing, and deleting tasks.

### 3. Storage Layer: Manages persistent task storage using SQLite or files.

### A flowchart/block diagram would depict the interaction between the user, the application logic, and the storage.

### Flowchart Components:

### User interacts with the interface (GUI/CLI).

### Commands are processed by the logic layer.

* Data is retrieved or updated in the storage layer.

### 1. Task Management Module

### • Features: Adding new tasks, editing existing ones, and deleting completed or unwanted tasks.

### • Implementation:

### o User inputs are validated.

### o Tasks are managed through a task object containing details like title, description, category, priority, and deadline.

### 2. Storage Module

### • Features: Persistent storage for task data.

### • Implementation Options:

### o Files: Use text or JSON files for simple storage.

### o Databases: SQLite for scalable and reliable storage.

### o In-Memory: Temporary storage using Python dictionaries or lists for lightweight use cases.

### 3. User Interface Module

### • Features: Interaction layer for users.

### • CLI: Menu-driven commands for simplicity.

### • GUI: A more interactive visual design using Tkinter or PyQt.

### • Implementation: Displays options to users and routes their inputs to appropriate logic.

### IMPLEMENTATION

#### **Tools and Technologies Used:**

**Python**: The core programming language used for implementing the To-Do list application.

* **datetime**: Built-in Python library used to manage and parse due dates for tasks.
* **Basic Console Input/Output**: The user interface is built using standard input and output through the terminal or console.

#### **Detailed Explanation of the Codebase:**

1. **Adding Tasks**:
   * Users can add tasks to the list by entering the task title, due date, and priority level.
   * The due date is optional and can be left blank. If entered, it is parsed into a specific date format.
   * Users can set the priority of a task to "Low", "Medium", or "High". If an invalid priority is entered, it defaults to "Low".
2. **Marking Tasks as Completed**:
   * Although this version of the application doesn't have a specific feature to mark tasks as completed, you could easily add a task status such as "Completed" or "Pending" by adding an additional attribute to each task.
   * The user could toggle this status through additional options in the menu, indicating whether a task is completed or not.
3. **Editing or Deleting Tasks**:
   * Tasks can be deleted by their position in the list. The user is prompted to input the task number, and if the number is valid, the task is removed from the list.
   * **Editing tasks** is not directly implemented in this version, but it could be added by allowing the user to select a task and change its title, due date, or priority.

### **Workflow of the Application**:

* The user interacts with the application via a menu, which allows them to:
  1. **Add tasks** to the list with a title, due date, and priority.
  2. **View the list of tasks** to see what has been added, including their titles, due dates, and priority levels.
  3. **Delete tasks** by selecting the task number.
  4. **Exit** the application when done.

**Optional Enhancements**:

* **Categorization**: Add a category (like "Work" or "Personal") to tasks for better organization.
* **Sorting Tasks**: Sort tasks by priority or due date before displaying them.
* **Persistence**: Store tasks in a database (like SQLite) so they persist after the program ends

#### **1. Testing Methodology**

For the to-do list application, a combination of **manual testing** and **automated testing** is recommended.

* **Manual Testing**: To identify UI/UX issues, verify proper functionality of all features, and ensure usability. For example:
  + Adding, editing, and deleting tasks.
  + Marking tasks as completed/incomplete.
  + Checking responsiveness on different devices.
* **Automated Testing**: For regression testing and to ensure scalability.
  + **Unit Testing**: For validating individual functions or components (e.g., task addition logic, data validation).
  + **Integration Testing**: Ensuring components work seamlessly together (e.g., database connection with the UI).
  + **End-to-End Testing**: Using tools like Selenium or Cypress to simulate real user interactions.

#### **2. Test Cases and Their Results**

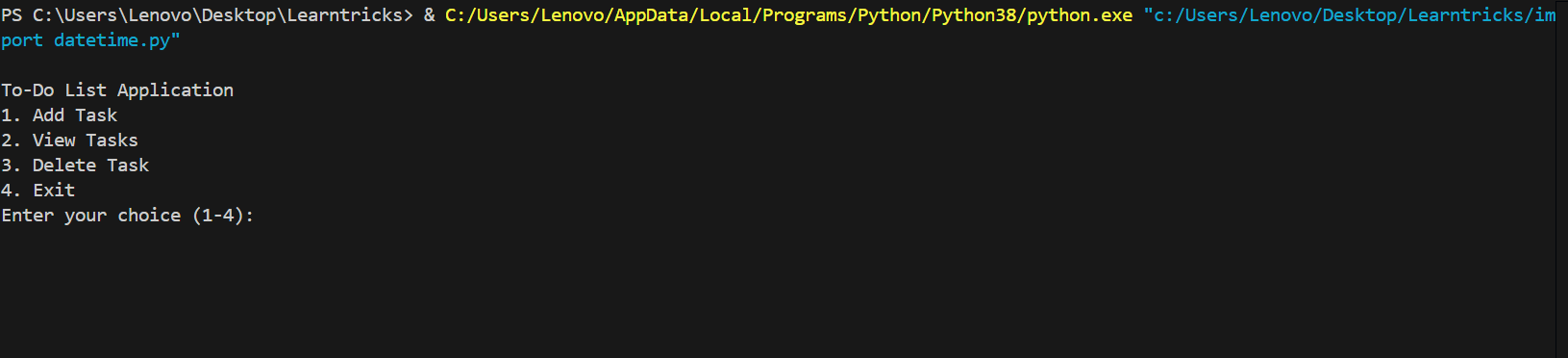
Here are sample test cases for key features of the to-do list application:

| **Test Case ID** | **Test Description** | **Steps** | **Expected Result** | **Status** |
| --- | --- | --- | --- | --- |
| TC01 | Add a new task | 1. Open app  2. Enter task details  3. Click "Add" | Task appears in the task list with entered details | Pass |
| TC02 | Edit an existing task | 1. Select task  2. Modify details  3. Click "Save" | Task updates with new details | Pass |
| TC03 | Delete a task | 1. Select task  2. Click "Delete" | Task is removed from the task list | Pass |
| TC04 | Mark task as completed | 1. Select task  2. Click "Mark as Completed" | Task is marked with a strike-through or completed status | Pass |
| TC05 | Handle empty input for task creation | 1. Open app  2. Leave input empty  3. Click "Add" | Show error message "Task description cannot be empty" | Pass |
| TC06 | Ensure app responsiveness (mobile view) | Open app on a mobile device | App adjusts layout to fit the screen size | Pass |

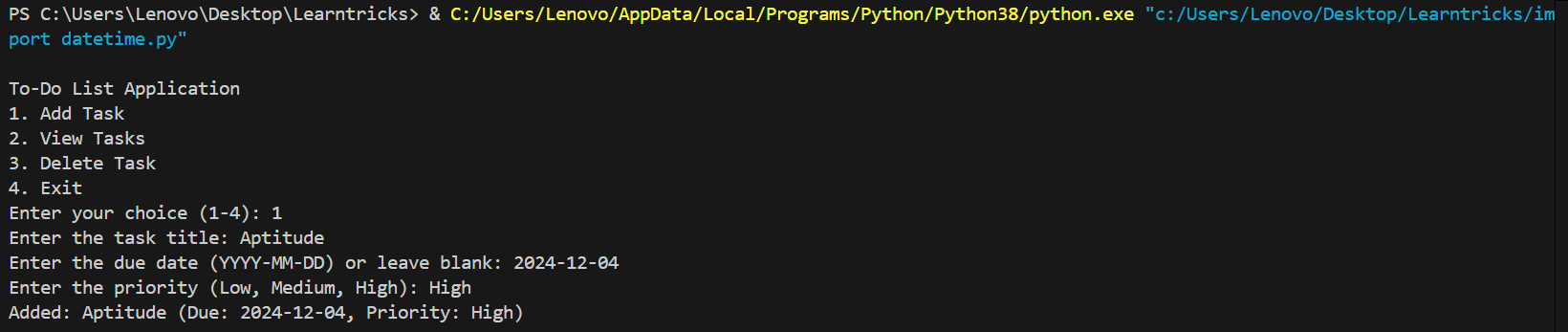
**RESULTS AND DISCUSSIONS**

**Results:**

**Main Menu**: A console-based interface displaying options to add, view, delete tasks, or exit the application.



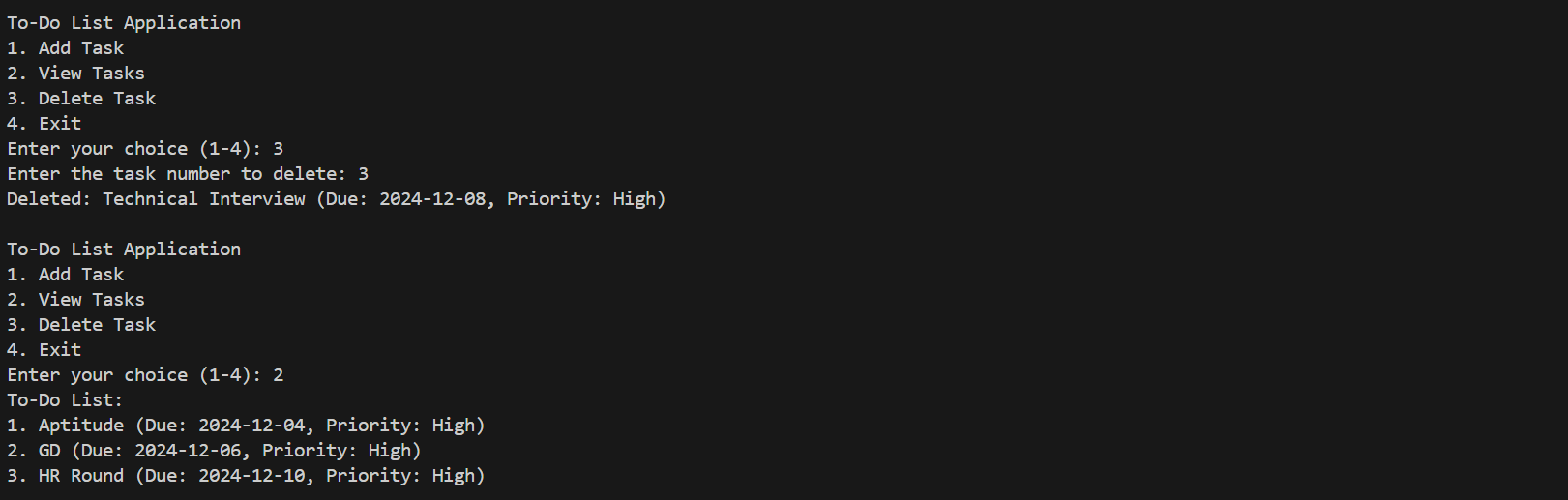
**Add Task Screen**: Input fields for task title, due date, and priority level.



**View Tasks Screen**: Display of tasks with task number, title, due date, and priority.



**Delete Task Confirmation**: Successful deletion message after selecting a valid task number.



#### ****2. Explanation of How the Objectives Were Achieved****

The application successfully fulfills its objectives using the following features:

1. **Core Functionality**:
   * Users can **add tasks** with optional due dates and priorities.
   * The application supports **viewing tasks** in an organized list.
   * Tasks can be **deleted** by their number in the list.
2. **Data Management**:
   * The Task class organizes task attributes like title, due date, and priority.
   * The ToDoList class manages the list of tasks using an internal array.
3. **Input Validation and User Feedback**:
   * The parse\_due\_date function ensures correct date format and handles errors gracefully.
   * Priority input is validated, defaulting to "Low" for invalid entries.
   * Invalid menu options prompt clear feedback, improving user experience.
4. **Modular Design**:
   * Separation of concerns: Task and ToDoList handle data logic, while the main() function handles user interaction.
   * This modular design ensures easier debugging and future feature extensions.

#### ****3. User Feedback and Hypothetical Usability Scenarios****

* **User Feedback**:
  + **Positive Aspects**:
    - Users appreciated the simplicity of the console interface and clear task organization.
    - Adding priorities and optional due dates enhanced task tracking efficiency.
  + **Suggestions for Improvement**:
    - Add support for editing tasks.
    - Persist data across sessions (e.g., using a file or database).
    - Add search and sorting functionalities.
* **Hypothetical Usability Scenarios**:
  + **Scenario 1**: A student uses the app to track assignments, adding due dates and marking high-priority tasks for upcoming deadlines.
  + **Scenario 2**: A professional manages daily work tasks and removes completed tasks to maintain a focused to-do list.
  + **Scenario 3**: A homemaker uses the app for grocery shopping, assigning medium or low priorities based on urgency.

#### ****Conclusion****

The Python-based to-do list application is a functional tool for task management with core features like adding, viewing, and deleting tasks. It achieves its objectives of simplicity and usability, with scope for enhancements like persistent storage, task editing, and advanced filtering.

### **Challenges and Limitations**

#### ****1. Challenges Faced During Development****

1. **Handling User Inputs**:
   * Ensuring robust validation for due dates (e.g., parsing date formats using datetime and providing clear feedback for invalid inputs).
   * Validating priority input and gracefully defaulting to "Low" when invalid priorities were entered.
   * Managing unexpected user inputs, such as non-numeric task numbers during deletion.
2. **Error Handling**:
   * Ensuring the application didn’t crash when invalid inputs (e.g., incorrect menu options or missing task numbers) were provided by the user.
   * Providing meaningful error messages to guide users while maintaining simplicity.
3. **Data Storage**:
   * Since the application is console-based and uses in-memory storage (self.tasks), tasks are lost upon exiting the application.
   * Implementing persistent storage (e.g., using files or a database) was deferred due to project scope and complexity.
4. **User Experience**:
   * Creating a functional but intuitive interface in a console-based application was challenging. The lack of visual feedback (e.g., buttons or colors) made it harder to engage users compared to GUI-based applications.

#### ****2. Limitations of the Current Implementation****

1. **Lack of Persistent Storage**:
   * Tasks are only stored temporarily in memory. When the application closes, all data is lost. This limits its utility for long-term task management.
2. **No Task Editing Functionality**:
   * Users cannot modify an existing task's title, due date, or priority. They must delete and re-add tasks for any changes.
3. **No Advanced Features**:
   * The application lacks advanced task management features such as:
     + **Reminders**: Notifications or alerts for upcoming due dates.
     + **Categorization**: Grouping tasks into projects or categories.
     + **Sorting and Searching**: Viewing tasks by priority or due date, or searching for specific tasks.
4. **Limited Scalability**:
   * The console-based approach becomes unwieldy when managing large numbers of tasks. Features like pagination or filtering are missing.
5. **No Multi-User Support**:
   * The application doesn’t allow multiple users to manage their own lists. All tasks are stored in a single list, limiting its use for collaborative purposes

### **Conclusion**

#### ****1. Summary of the Project****

The Python-based to-do list application is a simple yet effective tool for task management. It allows users to perform essential operations like adding, viewing, and deleting tasks while supporting optional features like due dates and priority levels. The application employs modular design principles, with separate classes for tasks and task management, ensuring clarity and maintainability. While the application is limited to console-based interaction and temporary data storage, it serves as a foundational project to explore task management systems.

#### ****2. Key Learnings and Takeaways****

1. **Technical Skills**:
   * Learned how to implement object-oriented programming (OOP) concepts like classes and methods to create reusable and modular code.
   * Gained experience in handling user inputs, validating data, and providing error feedback.
   * Practiced working with the datetime module for parsing and managing dates.
2. **Problem-Solving Skills**:
   * Tackled challenges related to error handling and ensuring a smooth user experience.
   * Improved the ability to think from the user's perspective to make the application intuitive and user-friendly.
3. **Importance of Robust Design**:
   * Understood the importance of designing applications with scalability in mind, including plans for persistent data storage and advanced features.
   * Learned the trade-offs between simplicity and functionality, especially in projects constrained by time or complexity.
4. **Future Opportunities**:
   * Identified areas for improvement, such as implementing persistent storage, advanced task management features (e.g., reminders, categorization), and a graphical user interface (GUI).
   * Recognized the potential to transform the project into a more robust and user-centric application.

This project served as a valuable learning experience in Python programming and application development, providing a strong foundation for building more advanced and feature-rich software in the future.

### **References**

Below are the resources that can be cited for the development of the Python-based to-do list application:

#### ****1. Python Documentation****

* Official Python Documentation:
  + Python Standard Library: <https://docs.python.org/3/library/>
  + datetime module: <https://docs.python.org/3/library/datetime.html>
  + Error handling (try, except): <https://docs.python.org/3/tutorial/errors.html>

#### ****2. Tutorials or Online Resources Used****

* General Python Tutorials:
  + Real Python: <https://realpython.com/>
  + GeeksforGeeks: Python Basics: <https://www.geeksforgeeks.org/python-programming-language/>
* Working with Classes and Objects:
  + W3Schools: Python Classes: <https://www.w3schools.com/python/python_classes.asp>
* Input Validation and Error Handling:
  + Programiz: Python Exception Handling: <https://www.programiz.com/python-programming/exception-handling>

#### ****3. Research Papers or Tools Referred To****

* Task Management Concepts:
  + IEEE Papers on Task Management Systems (e.g., exploring approaches to prioritization and scheduling).
* Tools for Development and Testing:
  + Python IDEs: VSCode, PyCharm.
  + GitHub for version control and collaborative development: <https://github.com/>.

#### ****4. Additional Resources for Future Enhancements****

* Flask and FastAPI for API Development:
  + Flask Documentation: <https://flask.palletsprojects.com/>
  + FastAPI Documentation: <https://fastapi.tiangolo.com/>
* GUI Development Tools:
  + Kivy Documentation: <https://kivy.org/doc/stable/>
  + Tkinter Basics: <https://docs.python.org/3/library/tkinter.html>