COMP 354

Introduction to Software Engineering

Maintenance Report (I)

Instructor: Dr. Malleswara Talla

Submission Date: July 11, 2025

Group Participant:

Nidhi Prasad ***40259273***
Narendra Mishra ***40224303***
Neelendra Mishra ***40224310***
Aryan Kotecha ***40249867***
Parsa Ghadimi ***40203370***
Kirubel Asrat 40153970

Table Of Content

1. Executive Summary

This document presents the design and development approach we followed to improve the existing python-cli-todo-program, a simple command-line to-do list tool. The original version allowed users to perform basic actions like adding, listing, completing, and deleting tasks through a shell script, with all data stored in a plain text file. While that setup worked for basic use, it lacked flexibility and was hard to scale or modify for more advanced functionality.

Our main improvement was replacing the plain text storage with a structured JSON format. This change not only made the data easier to manage and extend, but it also laid the groundwork for introducing new features like task prioritization, due dates, tagging, undo, progress tracking, and filtering/search capabilities. With JSON, tasks are stored in a structured, queryable format, which makes the system much easier to scale and maintain.

We also restructured the codebase into a modular design, separating core logic, data handling, CLI interaction, and utilities into their own components. This helps keep the code organized and makes it easier for future developers to understand and contribute to the project. The command-line interface was improved using Python's argparse module, and task output was made more readable and user-friendly with color-coded formatting using the colorama library.

Beyond those improvements, we also added error handling, input validation, and backup/undo mechanisms to make the tool more reliable and user-proof. We've also included a set of unit and integration tests to help ensure everything works as expected. Overall, this document serves as a guide not only for what was built but also for future teams who may take over and continue improving the project.

This project was developed with a strong focus on practical usability and long-term maintainability. Instead of just adding features for the sake of complexity, we focused on real improvements that enhance the user experience and align with common developer needs. For example, features like undo, progress tracking, and task search were added based on how people use to-do lists in real life quick, simple, and efficient.

Throughout development, we also kept future contributors in mind. We documented the system thoroughly, structured the code to follow standard practices like PEP8, and made sure the logic is easy to follow and extend. Whether someone wants to add cloud sync, a web interface, or more advanced analytics later, the foundation we've built will make it easier for them to pick up where we left off.

2. Introduction

The CLI To-Do Application project focuses on evolving a basic open-source command-line task manager into a feature-rich, modular, and maintainable Python-based system. Originally, the project supported simply add, list, complete, and delete operations using a shell script and plain text file for storage.

Our aim was to build on that foundation without losing its simplicity. We introduced structured JSON-based storage to make task data more manageable and extensible. On top of that, we redesigned the command-line interface using Python's ***argparse*** module, which makes it easier to pass arguments and interact with the program. We also added new features like task prioritization, due dates, undo actions, search functionality, and progress tracking. All these changes were made while following core software engineering principles to ensure the system remains clean, modular, and easy to maintain in the long run.

Source: **https://github.com/detronetdip/python-cli-todo-program**

3. System Design and Overview

3.1 Current System Analysis

The existing system consists of:

- Basic shell script interface (todo_cli.sh)
- Simple text file storage
- Limited functionality (add, delete, list, complete)
- Monolithic code structure

3.2 Monolithic code structure

The enhanced system will feature:

- Modular Python architecture
- JSON-based data storage
- Advanced CLI interface using argparse
- Rich feature set with priority management, due dates, and search capabilities

4. System Architecture

a. High-Level Architecture

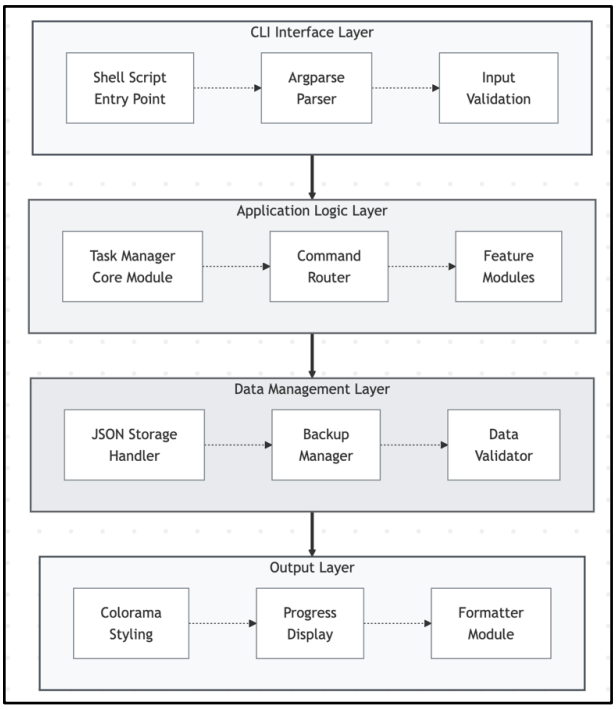


Figure 1 Layered System Architecture of the Enhanced CLI To-Do Application

b. Module Architecture

4.2.1 Core Modules

main.py

-> Entry point for the application
-> Command-line argument parsing
-> Main application flow control

task_manager.py

-> Core task operations (CRUD)
-> Task validation and processing
-> Business logic implementation

data_handler.py

-> JSON file operations
-> Data persistence and retrieval
-> Backup and recovery functions

cli_interface.py

-> User interface management
-> Input/output formatting
-> Color-coded display using colorama

utilities.py

-> Helper functions
-> Date/time utilities
-> Common validation functions

4.2.2 Feature Modules

priority_manager.py

-> Priority level management
-> Priority-based sorting
-> Priority validation

date_manager.py

-> Due date handling
-> Date validation
-> Reminder calculations

search_engine.py

-> Search functionality
-> Filtering operations
-> Query processing

undo_manager.py

-> Undo/redo operations
-> Action history tracking
-> State management

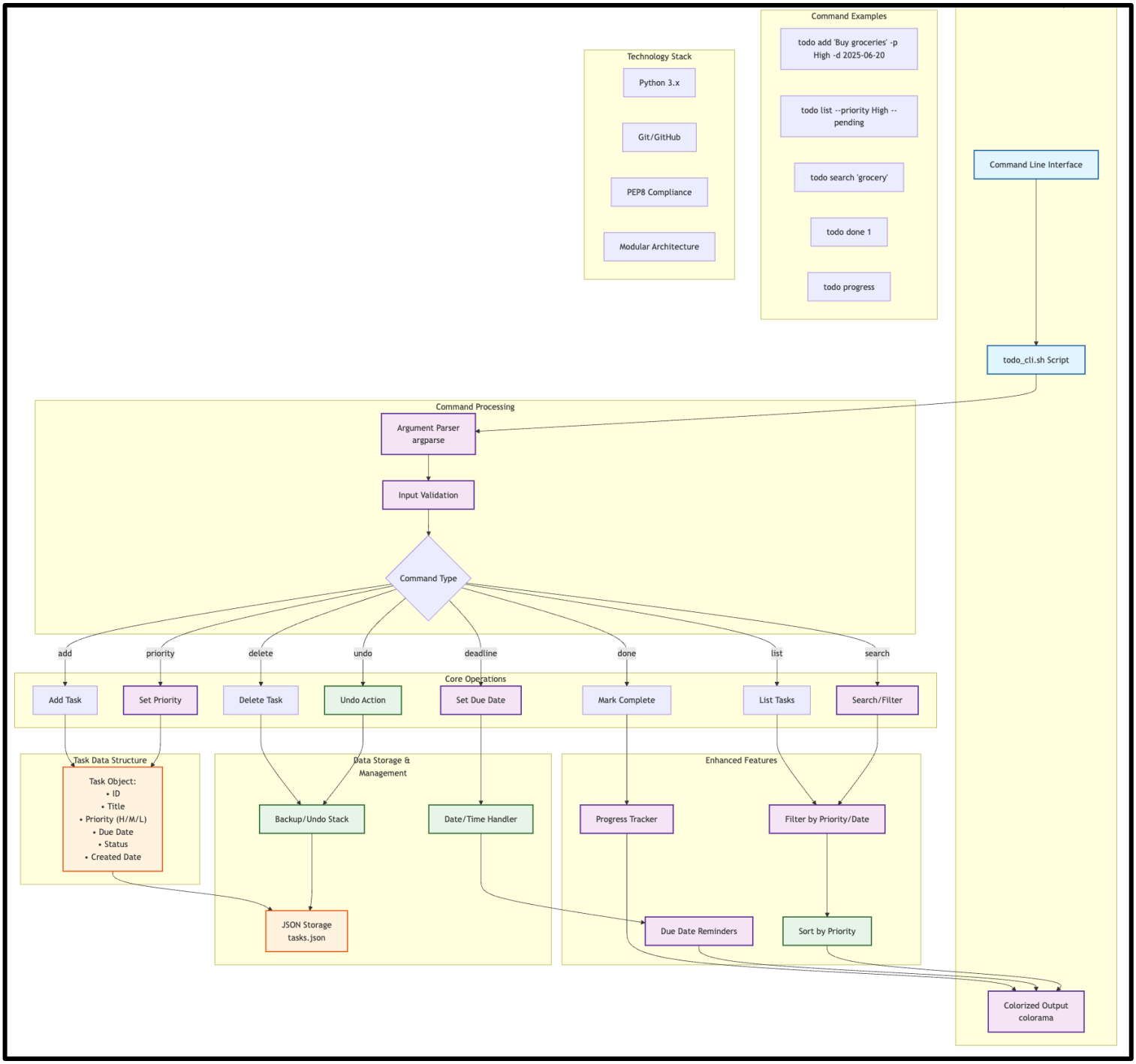


Figure 2 Functional Flow Diagram of the Enhanced CLI To-Do Application

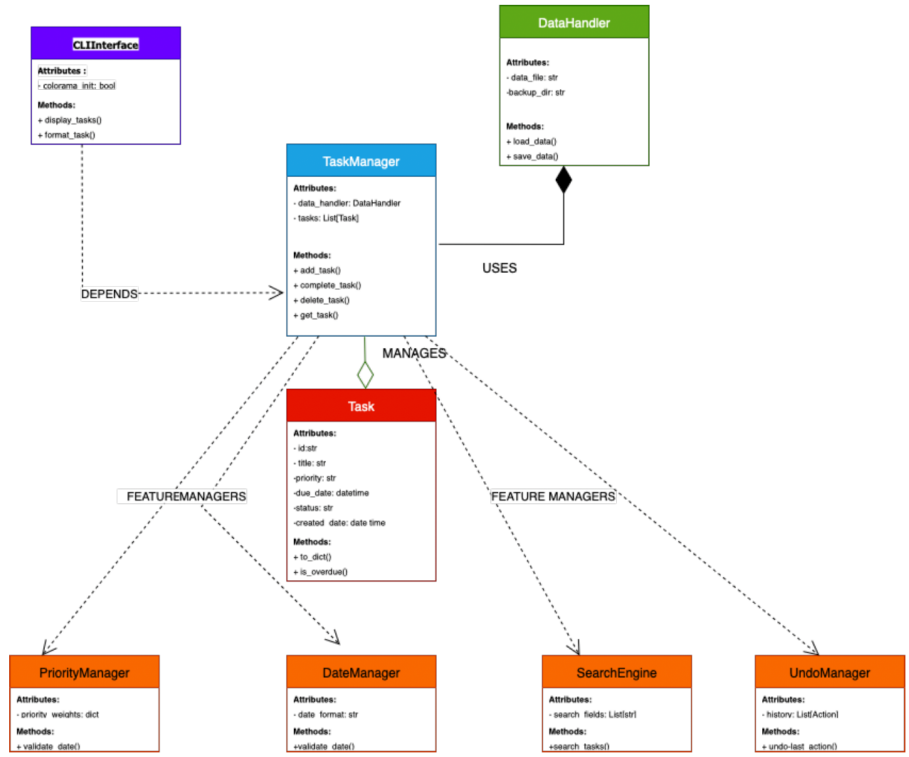


Figure 3 UML Diagram

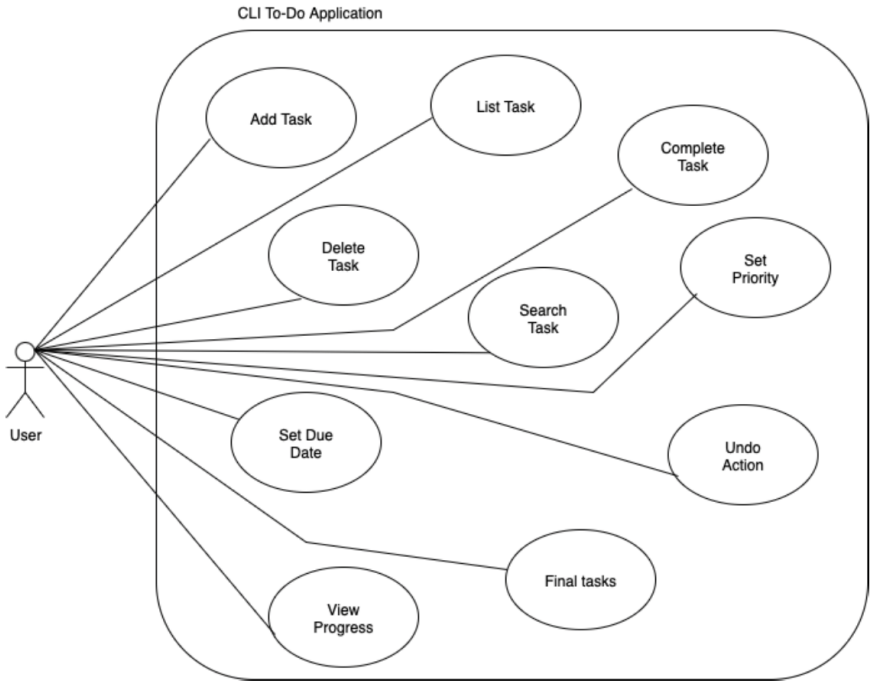


Figure 4 Use Case Diagram

5. Data Design

5.1 Data Structure

5.1.1 Task Object Schema

```Json
{ "id": "unique_task_id",
  "title": "Task description",
  "priority": "High|Medium|Low",
  "due_date": "YYYY-MM-DD",
  "status": "pending|completed",
  "created_date": "YYYY-MM-DD HH:MM:SS",
  "completed_date": "YYYY-MM-DD HH:MM:SS",
  "tags": ["tag1", "tag2"]
}
```

5.1.2 Application State Schema

```Json
{
  "tasks": [
     {/* Task objects */}
   ],
  "settings": {
  "next_id": 1,
  "default_priority": "Medium",
  "date_format": "%Y-%m-%d"
},
  "history": [
     {
       "action": "add|delete|complete|modify",
       "timestamp": "YYYY-MM-DD HH:MM:SS",
       "task_data": {/* Task object */}
     }
  ]
}
```

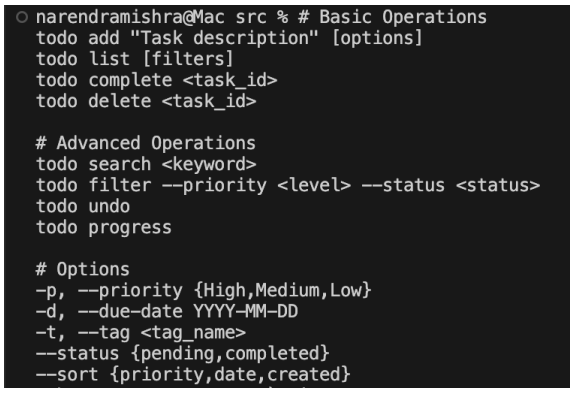
5.2 File Structure

```
todo_app/
├── data/
│   ├── tasks.json          # Main task storage
│   ├── backup/             # Backup files
│   └── history.json        # Undo history
├── src/
│   ├── main.py
│   ├── task_manager.py
│   ├── data_handler.py
│   ├── cli_interface.py
│   ├── utilities.py
│   ├── priority_manager.py
│   ├── date_manager.py
│   ├── search_engine.py
│   └── undo_manager.py
├── tests/
│   ├── test_task_manager.py
│   ├── test_data_handler.py
│   └── test_search_engine.py
├── docs/
│   ├── user_manual.md
│   └── api_documentation.md
├── scripts/
│   └── todo_cli.sh         # Shell script launcher
└── requirements.txt
```

6. Interface Design

6.1 Command-Line Interface

6.1.1 Command Structure



6.1.2 Example Usage

```
narendramishra@Mac src % # Add a high priority task with due date
todo add "Complete project documentation" -p High -d 2025-07-15

# List all pending high priority tasks
todo list --priority High --status pending

# Search for tasks containing "project"
todo search "project"

# Show progress summary
todo progress
```

6.2 Output Format

ID	Task	Priority	Due Date	Status
1	Complete project documentation	**High**	2025-07-10	Pending
2	Review code change	**Medium**	2025-07-15	Pending
3	Setup development environment	**Low**	-	**Completed**

6.2.1 Colour Coding

- **Green:** Completed tasks
- **Red**: Overdue tasks
- **Yellow**: High priority tasks
- **Blue**: Medium priority tasks
- **White**: Low priority task

7. Detailed Component Design

7.1.1 Task Manager Component

```python
class TaskManager:
    def __init__(self, data_handler):
        self.data_handler = data_handler
        self.tasks = []

    def add_task(self, title, priority='Medium', due_date=None):
        """Add a new task"""
        pass

    def complete_task(self, task_id):
        """Mark a task as completed"""
        pass

    def delete_task(self, task_id):
        """Delete a task"""
        pass

    def get_tasks(self, filters=None):
        """Retrieve tasks with optional filters"""
        pass

    def update_task(self, task_id, **kwargs):
        """Update task properties"""
        pass
```

7.1.2 Key Methods

- **add_task()**: Validates input and creates new task
- **complete_task()**: Updates task status and completion date
- **delete_task()**: Removes task and creates backup
- **get_tasks()**: Retrieves tasks with filtering support
- **update_task()**: Modifies existing task properties

7.2 Data Handler Component

7.2.1 Class Structure

```python
class DataHandler:
    def __init__(self, data_file='data/tasks.json'):
        self.data_file = data_file
        self.backup_dir = 'data/backup'

    def load_data(self):
        """Load tasks from JSON file"""
        pass

    def save_data(self, data):
        """Save tasks to JSON file"""
        pass

    def create_backup(self):
        """Create backup of current data"""
        pass

    def restore_backup(self, backup_file):
        """Restore from backup"""
        pass
```

7.3 CLI Interface Component

7.3.1 Class Structure

```python
class CLIInterface:
    def __init__(self):
        self.colorama_init()

    def display_tasks(self, tasks):
        """Display tasks with formatting"""
        pass

    def display_progress(self, completed, total):
        """Show progress summary"""
        pass

    def format_task(self, task):
        """Format individual task for display"""
        pass

    def get_color_for_status(self, status, priority):
        """Return appropriate color for task"""
        pass
```

8. Algorithm Design

8.1 Task Prioritization Algorithm

```python
def sort_tasks_by_priority(tasks):
    """
    Sort tasks by priority and due date
    Priority order: High -> Medium -> Low
    Within same priority: Sort by due date (earliest first)
    """
    priority_weights = {'High': 3, 'Medium': 2, 'Low': 1}

    return sorted(tasks, key=lambda task: (
        -priority_weights.get(task['priority'], 1),
        task['due_date'] if task['due_date'] else '9999-12-31'
    ))
```

8.2 Search Algorithm

```python
def search_tasks(tasks, query):
    """
    Search tasks using keyword matching
    Supports partial matching and case-insensitive search
    """
    results = []
    query_lower = query.lower()

    for task in tasks:
        if (query_lower in task['title'].lower() or
            query_lower in task.get('tags', [])):
            results.append(task)

    return results
```

8.3 Reminder Algorithm

```python
def get_due_soon_tasks(tasks, days_ahead=3):
    """
    Get tasks due within specified days
    """
    today = datetime.now().date()
    upcoming_date = today + timedelta(days=days_ahead)

    due_soon = []
    for task in tasks:
        if task['due_date'] and task['status'] == 'pending':
            due_date = datetime.strptime(task['due_date'], '%Y-%m-%d').date()
            if today <= due_date <= upcoming_date:
                due_soon.append(task)

    return due_soon
```

9. Error Handling Strategy

9.1 Exception Hierarchy

```python
class TodoAppException(Exception):
    """Base exception for todo application"""
    pass

class InvalidTaskException(TodoAppException):
    """Raised when task data is invalid"""
    pass

class TaskNotFoundException(TodoAppException):
    """Raised when task is not found"""
    pass

class DataCorruptionException(TodoAppException):
    """Raised when data file is corrupted"""
    pass
```

9.2 Error Handling Patterns

- **Input Validation**: Validate all user inputs before processing
- **File Operations**: Handle file not found, permission errors
- **Data Integrity**: Validate JSON structure and data types
- **Graceful Degradation**: Fallback to basic functionality if advanced features fail

10. Demonstration of CLI Functionality

```
/Users/narendramishra/PycharmProjects/python348/.venv/bin/python /Users/narendramishra/Desktop/maintenanceProj/main.py
usage: main.py [-h] [--add ADD] [--priority {High,Medium,Low}] [--due DUE]
               [--done DONE] [--delete DELETE] [--list] [--undo]
               [--search SEARCH] [--progress]

CLI To-Do Application

optional arguments:
  -h, --help            show this help message and exit
  --add ADD             Add a new task
  --priority {High,Medium,Low}
                        Set task priority
  --due DUE             Set task due date (YYYY-MM-DD)
  --done DONE           Mark task as complete
  --delete DELETE       Delete task by ID
  --list                List all tasks
  --undo                Undo last action
  --search SEARCH       Search tasks
  --progress            Show task progress

Process finished with exit code 0
```

- **Add Task**: Adding a New Task

```
(.venv) narendramishra@Mac maintenanceProj % python main.py --add "LAB 2 DUE" --priority High  --due 2025-07-10
Task added.
```

- **List Tasks** : Listing All Tasks

```
(.venv) narendramishra@Mac maintenanceProj % python main.py --list
ID: 1 | Write final report | Priority: High | Due: 2025-07-15 | Status: pending
```

- **Delete Task**: Deleting a Task by ID

```
(.venv) narendramishra@Mac maintenanceProj % python main.py --delete 3
Task deleted.
```

- **Search Tasks**: Searching Tasks by Keyword

```
(.venv) narendramishra@Mac maintenanceProj % python main.py --search GYM
ID: 3 | Go for GYM | Priority: Low | Due: 2025-07-11 | Status: pending
```

- **Complete Task** : Marking a Task as Completed

```
(.venv) narendramishra@Mac maintenanceProj % python main.py --done 1
Task marked as complete.
```

- **Undo** : Undoing the Last Action

```
(.venv) narendramishra@Mac maintenanceProj % python main.py --undo
Undo successful.
```

- **Progress**: Viewing Task Progress

```
(.venv) narendramishra@Mac maintenanceProj % python main.py --progress
Completed: 5 / 5 (100.00% done)
```

- **Colour Schemas**: Different Colour for different set of priorities (High – yellow, Medium - blue, Green – completed, and Low - white)

```
ID: 1 | LAB 2 DUE | Priority: High | Due: 2025-07-10 | Status: completed
ID: 2 | Mentainance Project PART-1 DUE | Priority: High | Due: 2025-07-11 | Status: pending
ID: 3 | Go for GYM | Priority: Low | Due: 2025-07-11 | Status: pending
ID: 4 | Study for COMP 354 TEST2 | Priority: Medium | Due: 2025-07-12 | Status: pending
```

11. Performance Considerations

11.1 Scalability

- **File Size**: Optimize for handling up to 10,000 tasks
- **Search Performance**: Implement efficient search algorithms
- **Memory Usage**: Load only necessary data into memory

11.2 Optimization Strategies

- **Lazy Loading**: Load tasks only when needed
- **Indexing**: Create indexes for frequently searched fields
- **Caching**: Cache frequently accessed data

12. Deployment maintenance

12.1 Installation Requirements

- **Python 3.7+**
- **Required packages**: colorama, argparse (built-in)
- **Operating System**: Linux, macOS, Windows (with minor modifications)

12.2 Maintenance Guidelines

- **Code Style**: Follow PEP8 guidelines
- **Documentation**: Maintain comprehensive docstrings
- **Version Control**: Use semantic versioning
- **Regular Backups**: Implement automatic backup strategies

13. Future Enhancements

- **Sync Capabilities**: Cloud synchronization
- **Team Collaboration**: Shared task lists
- **Plugins**: Extensible plugin system
- **Web Interface**: Optional web-based interface

14. Actual Work Done

Our team enhanced the original Python CLI To-Do application by replacing the plain text storage with a structured JSON format. This enabled the addition of advanced features such as task priority, due dates, tagging, undo functionality, and progress tracking.

We refactored the entire system into modular Python components for better maintainability. Key modules include task management, data handling, search, undo, and CLI interface. The command-line experience was upgraded using argparse, and user feedback was improved with colorama for color-coded task displays.

Robust error handling, input validation, and backup mechanisms were implemented. Unit tests and integration tests were written to ensure code reliability. All work was thoroughly documented through a detailed system design document and a structured project layout, making the system easy to maintain and extend.

15. Challenges Faced

- **Backward Compatibility**: Adapting the existing codebase and maintaining some CLI behavior while implementing modular architecture proved complex.
- **Input Validation**: Designing a validation system that gracefully handles invalid flags, dates, and formats was time-consuming.
- **Undo Functionality**: Maintaining state and restoring deleted tasks correctly required designing a backup/rollback mechanism.
- **Team Syncing**: Coordinating work among team members, especially during testing phases, involved extra planning due to overlapping code areas.

16. Conclusion

The project successfully transforms a basic shell-based to-do list manager into a structured, scalable, and user-friendly Python CLI application. The key improvement transitioning to structured JSON storage enabled us to add critical features such as task prioritization, due dates, undo functionality, and intelligent filtering.
Following good software engineering practices, we achieved a modular codebase, clear documentation, and enhanced UX, making the tool both usable and maintainable. The final system is not only richer in functionality but also provides a strong foundation for future development and team collaboration.