Task Manager using OOP and File Management Software Documentation

1. System Architecture Overview

The system follows a modular, Object-Oriented structure with clear separation of concerns. It includes:

- **Models** – Represent core data structures like tasks.
- **Managers** – Handle business logic (e.g., adding, listing, updating, and deleting tasks).
- **Services** – Abstract file read/write operations to ensure persistence.
- **CLI Interface** – Manages user interactions via command-line arguments.

Component Breakdown:

- **Task**: A class representing a single task with properties such as `id`, `title`, `description`, `dueDate`, and `status`.
- **TaskManager**: Maintains a list of `Task` instances and contains logic for CRUD operations.
- **FileService**: Handles reading from and writing to a JSON file (`tasks.json`) to store task data persistently.
- **app.js**: Parses CLI commands and interacts with the `TaskManager`.

2. Object-Oriented Programming (OOP) Principles

The project demonstrates strong adherence to OOP principles:

Principle	Implementation Example
Encapsulation	Each class manages its own data and behavior (e.g., `Task` encapsulates task details).
Abstraction	File operations are abstracted inside `FileService`, so main logic doesn't deal with raw file handling.
Inheritance	While inheritance isn't central here due to the project scope, future expansion (e.g., `ReminderTask`, `ProjectTask`) could extend `Task`.
Polymorphism	(Not fully used here but the CLI can be extended to accept different commands flexibly).

3. Data Storage (Simulated Database)

Tasks are stored in a local file named `tasks.json`, simulating a flat-file database.

Example JSON:

```json
[
  {
    "id": 1,
    "title": "Complete Node.js Assignment",
    "description": "Finish the CLI task manager project",
    "due Date": "2025-07-30",
    "status": "incomplete"
  },
  {
    "id": 2,
    "title": "Submit Report",
    "description": "Upload the final report to Canvas",
    "due Date": "2025-07-31",
    "status": "incomplete"
  }
]
```

Schema:

Field	Type	Description
`id`	Integer	Unique identifier for each task
`title`	String	Title of the task
`description`	String	Detailed description of the task
`dueDate`	String	Due date in ISO format
`status`	String	Task status: "incomplete" or "completed"

---

4. Instructions for Setup and Running

Prerequisites:

- Node.js installed
- Git installed (for cloning)

Steps to Run the Project:

```
# Clone the repository
git clone https://github.com/your-group/task-manager.git
cd task-manager

# Install dependencies (if any)
npm install

# Run the CLI interface
node src/cli/app.js

# View all tasks
node src/cli/app.js list

# Update task
node src/cli/app.js update 2 --status completed

# Delete task
node src/cli/app.js delete 2
```

---

5. Error Handling

The system includes basic error handling to ensure smooth execution:

Scenario	Error Handling
Missing required command arguments	Displays usage help
File read/write errors	Shows error message and exits gracefully
Invalid task ID (e.g., not found)	Informs user task ID is invalid
Empty task list	Shows "No tasks found" message