Task organizer

Planning

The objective of this project is to design and build a to-do list manager application that helps users plan and prioritize their daily activities more efficiently. In addition to supporting common features like adding, editing, and categorizing tasks, the application will introduce a time estimation feature that allows users to assign expected hours to each task. This enables the system to optimize task prioritization and help users allocate their time more effectively.

The scope of the project includes core functionality such as using CRUD operations, persistent storage through SQLite to reliably save tasks across sessions, and developing a basic frontend interface using HTML, CSS, and JavaScript for user interaction. The application will also provide category-based filtering and display the total estimated hours for tasks, helping users gain insight into their workload. By focusing on these features, the project ensures a clear, consistent and achievable set of deliverables.

The stakeholders for this application are students, professionals, and individuals who need a lightweight and efficient tool to manage their daily responsibilities. At a high level, the system must provide a simple and intuitive interface, ensure that tasks are stored securely and reliably, and allow users to prioritize tasks easily. From the feasibility perspective, the project is achievable by using Python for the backend, with optimal frontend components. Financial cost is minimal since the project uses open-source tools, and operational maintenance is straightforward.

Requirement Analysis

The system is designed to provide all the essential features of a task manager while emphasizing time management and prioritization. Users should be able to create a new task by entering details such as title, description, category, estimated hours, and due date. Once created, tasks must be stored in SQLite so they persist reliably across sessions, ensuring that the user's progress is never lost. Users should also be able to retrieve and view their saved tasks easily, update task details whenever plans change, and remove tasks that are completed or no longer relevant.

A key requirement of the application is the ability to filter tasks by category and prioritize them based on the estimated time needed and their due dates. This ensures that the system goes beyond simple task tracking and actively supports effective time allocation. The frontend interface, built with HTML, CSS, and JavaScript, must remain simple and intuitive, lowering the barrier for users to use the application.

Non-functional requirements emphasize usability, performance, and reliability. The application should present a clear and minimal interface that avoids overwhelming the user, while SQLite ensures quick lookups and smooth task management.

Reliability is guaranteed through persistent storage, making sure no data is lost between sessions. Finally, the system should be designed with scalability in mind so that it can be expanded in the future with additional features such as notifications, recurring tasks, or even integration with external calendars.

SDLC Model

For this project, the Agile Software Development Life Cycle (SDLC) model was chosen. Agile is well suited because it allows development in small, iterative cycles where features can be delivered, tested, and refined incrementally.

In this context, the first iterations will focus on code CRUD functionality and persistent storage, ensuring that tasks can be created and saved reliably. Subsequent iterations will add enhancements such as category filtering, time-based prioritization, etc. This iterative approach reduces risk, and provides flexibility to adjust requirements as the project evolves.

Agile also aligns well with DevOps practices: continuous integration and version control with GIT will support frequent commits, testing, and incremental deployments. This provides a strong foundation for scaling the project in the future, for example by introducing APIs for mobile integration or deploying the application to cloud environments.

Design

System architecture

- Frontend (HTML/CSS/JavaScript): Provides simple interactive forms and dynamic task views, allowing users to add, edit, delete, and filter tasks directly in the browser.
- Backend (Flask - Python): A lightweight framework that exposes endpoints for handling CRUD operations and task prioritization logic. Flask ensures smooth integration between the user interface and the database.
- Persistent storage (SQLite): A file-based relational database that stores all tasks, including their categories, due dates, and estimated hours. Its simplicity and reliability make it well-suited for the scope of this project.
- Version Control (GitHub): Manages all project source code, enabling version tracking, structured commits, and collaborative-ready workflows while preparing the project for potential DevOps extensions.

Integration Points / Interfaces

- Frontend - User: The user interacts with the task manager features through forms that support adding new tasks, editing existing ones, filtering tasks by category, and displaying workload summaries such as total estimated hours per day.

- Frontend - Backend: JavaScript communicates with Flask via HTTP requests (GET, POST, PU, DELETE). Data is exchanged in JSON format, ensuring a lightweight and consistent interaction model.
- Backend - Database: Flask uses Python's sqlite3 module to interact with SQLite. Queries handle insertion, retrieval, updating, and deletion of task records, as well as calculations of time estimates and prioritization.

Data flow

The process begins when the user interacts with the frontend by submitting a task through a form. Input is validated within the browser to ensure essential fields such as task title and estimated hours are included. After validation, the frontend sends the request (POST, GET, PUT, DELETE) to the Flask backend. The backend receives the request, applies the appropriate logic, and executes SQL operations on the SQLite database. Once the operation is completed, FLask returns a JSON response to the frontend. The interface then updates dynamically, showing the user the new or modified task, recalculating total hours if needed, and reflecting updated task priorities.

Modules

- Task management module: Handles the creation, retrieval, updating, and deletion of tasks stored in SQLite.
- Category and filtering module: Enables users to organize tasks into categories and display only the tasks relevant to a chosen context.
- Time estimation and prioritization module: Uses task due dates and estimated hours to suggest an optimal order for task completion.
- User interface module: Built with HTML, CSS and JavaScript, it provides forms and dynamic task lists that respond to user input in real time.
- Backend module: Implements CRUD operations, task prioritization, and database queries using Flask and SQLite.

Algorithms

1. Manage Tasks
    - Create task: Validate task fields in the frontend, insert the task into the tasks table with its details (title, description, category, estimated hours, due date), and return confirmation.
    - Read task(s): Query one or multiple tasks from the tasks table (optionally filtered by category) and return the results in JSON format.
    - Update task: Receive updated values, modify the corresponding task record in the tasks table, and return confirmation.
    - Delete task: Remove the specified task from the tasks table and confirm deletion.
2. Filter and Prioritize Tasks
    - Filter by category: Query the database for tasks matching the selected category and return them to the frontend

        - Prioritize by time: Retrieve tasks and order them based on a combination of due date and estimated hours, providing users with a suggested order for completion
3. Track Total Estimated Hours
    - Calculate workload: Retrieve all tasks from the table, sum their estimated hours, and return the total to the frontend.
    - Update on changes: Recalculate total hours dynamically whenever a task is added, updated, or deleted.

Data Model and Database Schema

To define the data that needs to be stored, it is necessary to identify the fields required for managing tasks effectively. Since the application must support basic task management, category filtering, and prioritization based on estimated hours and due dates, the schema includes fields for descriptive information, time management, and organizational grouping. The following tables summarize the required fields for different entities in the system.

Tasks

Required Fields:

- Title: Short description of the task.
- Description: Optional longer explanation of the task.
- Category: Label used for grouping tasks (e.g., Work, Personal, Study).
- Estimated Hours: Number of hours the task is expected to take.
- Due Date: The deadline for completing the task.
- Priority: Calculated or assigned field indicating the order of importance.
- Status: Indicates whether the task is pending, in progress, or completed.

Categories

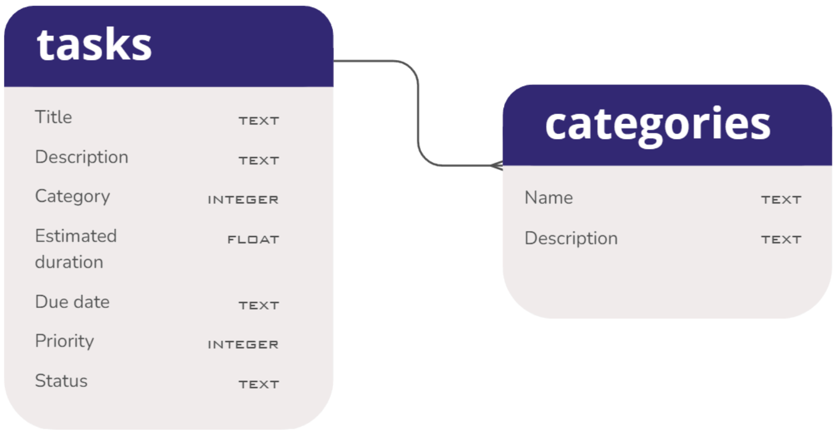
Required Fields:

- Name: Unique name of the category (e.g., Work, Personal).
- Description: Optional field for clarifying the purpose of the category.

Given these requirements, the database schema is organized into two main tables with the possibility of extending to a third if user accounts are added:

- Tasks: Represents individual tasks, including descriptive details, estimated hours, due dates, status, and links to categories.
- Categories: Represents task groupings, enabling users to organize tasks by context or project type.

A one-to-many relationship exists between Categories and Tasks, since each category can contain multiple tasks, but each task belongs to only one category. If user accounts are added, there would also be a one-to-many relationship between Users and Tasks, allowing each user to manage their own set of categories and tasks independently.



Key expected dates

- Database: 29/09
- Finished backend: 31/09 - 1/10
- Finished frontend: 2/10
- Improvements and error fixing: 4/10