
SMART STUDENT TASK & TIME MANAGER

PROJECT REPORT

NAME : SHARON PHILIP

REG NO : 24 BAS10119

1. Introduction

Managing academic tasks, assignments, deadlines, and revision schedules is a common challenge faced by students. With multiple courses, varying deadlines, and limited time available each day, students often struggle with planning and prioritizing their work effectively. This leads to missed deadlines, poor time management, and increased stress.

The **Smart Student Task & Time Manager** is designed to address this problem by providing a simple and efficient Java-based task management application. The tool helps students create, organize, and prioritize tasks while receiving smart recommendations and reminders. It promotes better productivity, planning, and academic performance.

Objectives:

- To design a lightweight Java application for effective task management.
- To help students prioritize tasks using an intelligent scoring system.
- To provide reminders for urgent and overdue tasks.
- To apply Java OOP concepts, modular design, and file-based data persistence.

2. Problem Statement

Students frequently manage multiple assignments, deadlines, and study goals across different subjects. Without a proper system for tracking tasks and prioritizing work, they risk missing deadlines or spending time inefficiently. Existing task managers are often overloaded with features or require installation of heavy frameworks.

This project provides a **simple, portable, console-based Java application** that helps students track tasks, receive reminders, and get suggestions based on urgency and priority.

3. Functional Requirements

1. User Management

- Create a new user profile.
- Select an existing user.

2. Task Management (CRUD)

- Add a new task with title, description, due date, priority, estimated time, and tags.
- Edit task details.
- Delete tasks.
- Mark tasks as complete/incomplete.

3. Suggestion Engine

- Recommend the next best task based on urgency, priority, and estimated time.

4. Task Reports

- List all tasks.
- Show overdue tasks.
- Show tasks due within 24 hours.
- Show completed tasks.

5. Data Persistence

- Save all data using Java serialization.
- Automatically load existing data on application start.

6. Reminder Feature

- Display tasks due soon and overdue tasks when the program opens.

4. Non-Functional Requirements

- **Performance:** File operations should complete quickly (<500ms).
- **Usability:** Simple and clear CLI menus.
- **Reliability:** Data is backed up automatically using .bak files.
- **Maintainability:** Modular OOP design with separate packages for model, service, storage, and app layers.
- **Error Handling:** Validation for inputs; graceful handling of faulty data.
- **Scalability:** Support for unlimited tasks within reasonable limits.

5. System Architecture

The system follows a **three-layer architecture**:

Presentation Layer

- App.java
Responsible for user interaction using console menus.

Service Layer

- UserService.java
- TaskManager.java
- ReminderService.java
Handles business logic such as creating users, adding tasks, scoring tasks, and retrieving reminders.

Persistence Layer

- Storage.java
Manages file operations using Java serialization.

6. Design Diagrams

6.1 Use Case Diagram (Text-based)

Actor: Student

Use Cases:

- Create Profile
- Select Profile
- Add Task
- Edit Task
- Delete Task
- Mark Complete
- View Tasks
- Get Recommendation
- View Reminders
- Save Data

6.2 Workflow Diagram

1. Start application
2. Load saved data
3. User selects or creates profile
4. Display reminders (due soon + overdue)
5. Show main menu
6. User performs operations
7. Save data on exit

6.3 Sequence Diagram – Add Task

User → App → TaskManager → Storage → File System

- User enters task details
- TaskManager creates a Task object

- TaskManager requests Storage to persist
- Storage serializes and saves to data/users.ser

6.4 Class Diagram (Text Format)

User

```

    |- username
    |- createdAt
    L tasks : List<Task>
```

Task

```

    |- id
    |- title
    |- description
    |- dueDate
    |- priority
    |- estimatedMinutes
    |- completed
    L tags
```

TaskManager

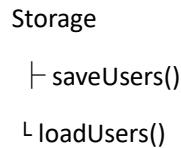
```

    |- addTask()
    |- deleteTask()
    |- recommendNext()
    |- tasksDueWithin()
    L overdueTasks()
```

UserService

```

    |- createUser()
    |- getUser()
    L persist()
```



7. Design Decisions & Rationale

- **File-Based Serialization:**
Chosen for simplicity and to avoid external libraries, making the project portable and easy to assess.
 - **Console-Based Interface:**
Suitable for beginner-friendly CLI programs and keeps focus on logic and architecture.
 - **Scoring Algorithm:**
Combines due date urgency, priority weight, and estimated time to generate a “next task” recommendation.
 - **Package Structure:**
Ensures separation of concerns and adheres to OOP modularity.
-

8. Implementation Details

Major Classes

- **User** – stores user details and task list.
- **Task** – represents a single task with metadata.
- **UserService** – manages users and persistence.
- **TaskManager** – core logic: add, delete, recommend tasks.
- **ReminderService** – calculates tasks due soon or overdue.
- **Storage** – serialization handler.
- **App** – main CLI interface.

Programming Concepts Used

- Encapsulation
- Collections (ArrayList, Map)
- Serialization
- Exception Handling
- File I/O
- Java Time API

