Project Title:

Simple Task Manager Application

By

Group Four (4)

1. **2520401300025** Micheal Dalu 08163427469
2. **2520401300029** MUBARAQ ALIYU 09074148247
3. **2520401300003** LUKMAN UMAR MUHAMMAD 08168786691
4. **2520401300016** FRANCIS STEPHEN ADEWALE ADEUSI 08033942682
5. **2520401300032** Abdulkarim Hussaini 07034903683

Problem Statement

Let's be honest—most of us struggle to stay organized. Between schoolwork, personal commitments, and random to-do lists scribbled on paper (that somehow disappear when we need them most), it's easy to lose track of things.

Some people try using their phone's default notes app. Others rely on their memory—until something slips through the cracks. And then there are those bulky task management tools that feel more complicated than the tasks themselves.

So yeah… managing daily activities manually or with ineffective tools can quickly turn into chaos. That's where the Simple Task Manager Application comes in. It's designed to help users plan, organize, and track their tasks without the confusion or clutter—just a clean, simple way to stay on top of things.

Objective

The goal here is simple: make task management actually manageable.

The Simple Task Manager Application aims to:

Allow users to easily create, and delete tasks, mark tasks as completed — because life changes, and so do priorities.

Enable task categorization or grouping — like separating completed tasks from pending .

Provide reminders or notifications — so users never miss an important deadline.

Offer a clear, user-friendly interface that's simple enough for anyone to use without needing a tutorial.

Help users track progress — giving them that small sense of accomplishment when they tick off completed tasks.

In short, this system should make organizing daily life less stressful and more structured.

Requirements

Functional Requirements (What the system does)

Users can create new tasks with titles, descriptions, and due dates.

Users can edit or delete tasks as needed.

The system should allow categorization or tagging of tasks (e.g., work, school, personal).

Users should be able to mark tasks as completed.

Users can view all tasks in a simple list format.

Non-Functional Requirements (How the system feels and performs)

Usability: The interface should be intuitive and friendly—no confusing menus or clutter.

Performance: Tasks should load instantly, with minimal lag.

Reliability: The system should save data consistently and prevent data loss.

Scalability: It should handle an increasing number of tasks without slowing down.

Compatibility: The application should work smoothly across devices (desktop, tablet, mobile).

Scope

User Scope

The application is mainly designed for students, professionals, and individuals who want to manage their daily activities more effectively. Whether it’s keeping track of assignments, chores, or work deadlines, the app provides a reliable, easy-to-use platform for staying organized.

System Scope

The system will:

Handle task creation, modification, and deletion.

Maintain a task database for saving user data.

Display progress and completion status.

It won’t, however, attempt to replace full-fledged productivity suites like Trello or Asana—it’s intentionally simple, lightweight, and focused on personal task tracking.

System Design

1. System Overview

1.1 Architecture Pattern

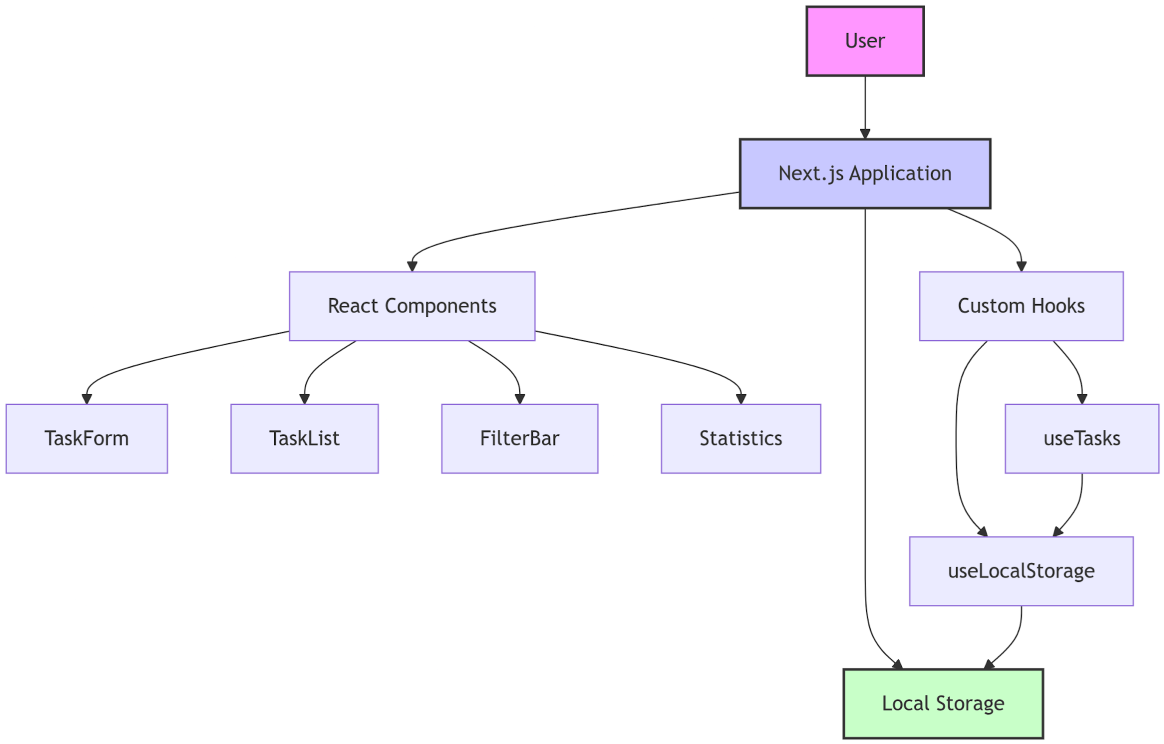
Client-Side Single Page Application (SPA) with local persistence

1.2 Technology Stack

Frontend: Next.js 15 + TypeScript + React 19
Styling: Tailwind CSS
State Management: React Hooks + Local Storage

Build Tool: Next.js Build System

2. High-Level Architecture



High Level Architecture

3. Core Components

- **TaskForm**: Handles task creation with validation
- **TaskList**: Displays filtered/sorted tasks
- **FilterBar**: Manages filters and search
- **Statistics**: Shows task metrics

4. Data Management

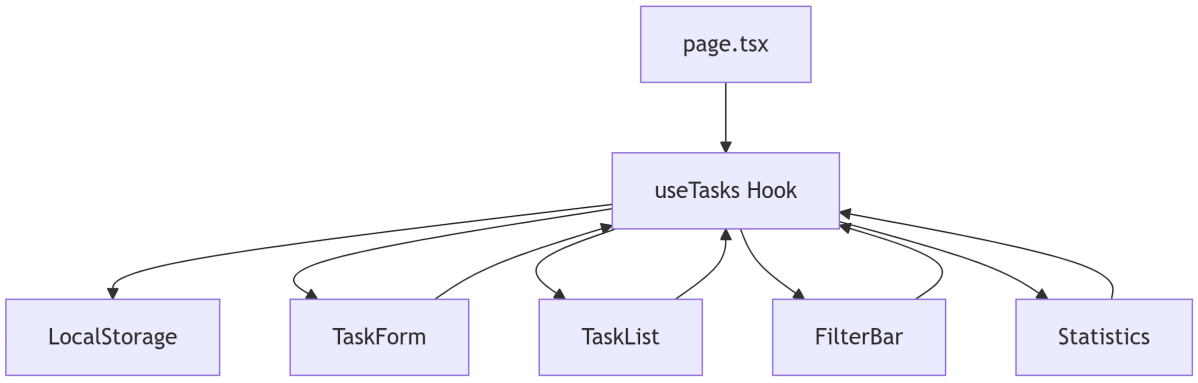
4.1 State Architecture

```
typescript
```

```
// Application State
{
 tasks: Task[],            // All tasks
 filters: {               // Current filters
  status: string,
  priority: string,
  category: string,
  searchTerm: string,
  sortBy: string
 }
}
```

4.2 Data Flow

1. **User Action** → Component → Hook
2. **Hook** → Update State → Local Storage
3. **State Change** → Re-render Components



Data flow diagram

5. Key Features Implementation

5.1 Task Management

- **Create**: Form validation + UUID generation
- **Read**: Filtering + Sorting + Search
- **Update**: Status toggling + Editing
- **Delete**: Remove from state + storage

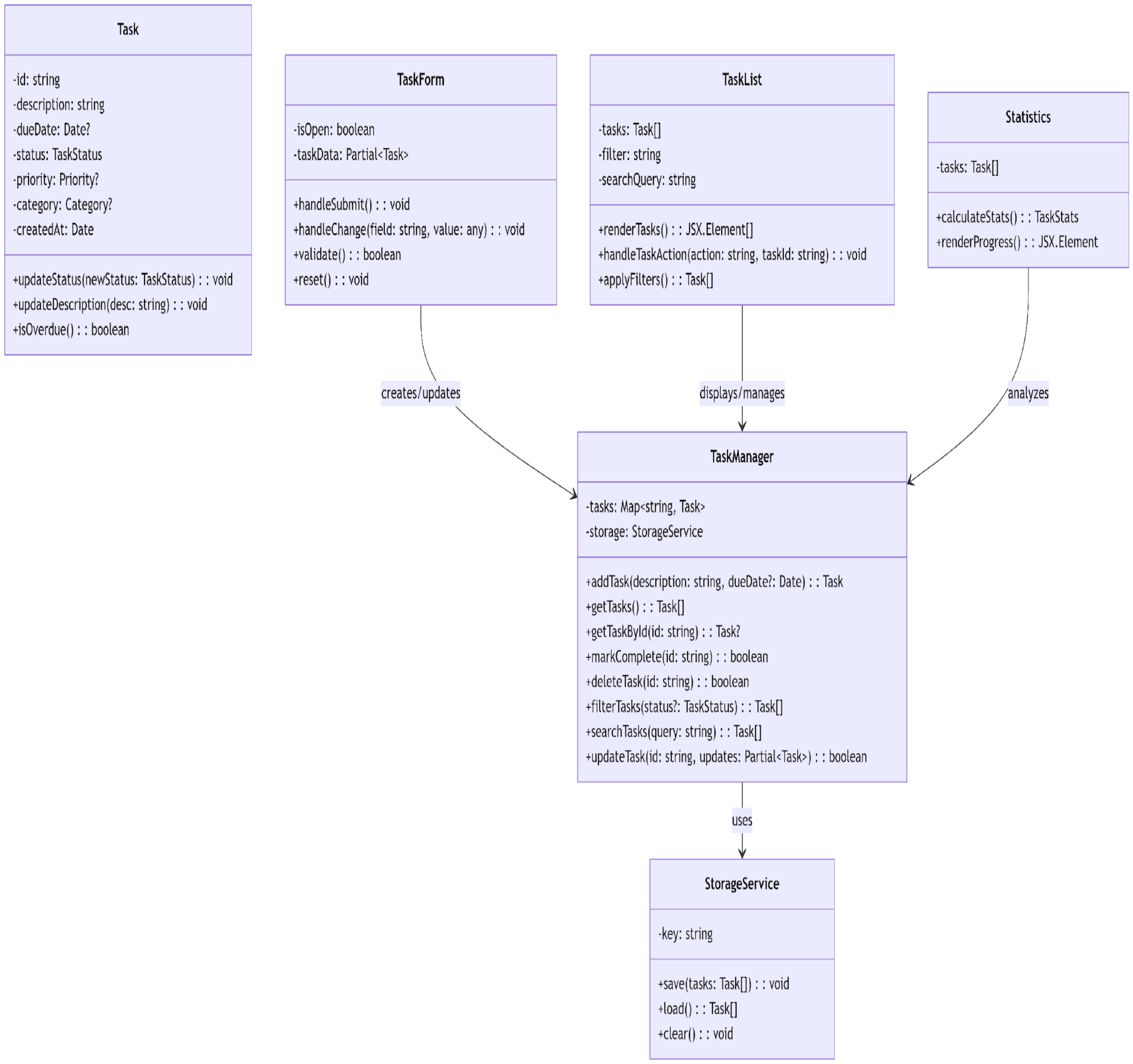
5.2 Filtering System

- **Multi-criteria**: Status + Priority + Category
- **Real-time**: Immediate filter application
- **Search**: Text-based task description search

6. Technology Decisions

6.1 Framework Choice

- **Next.js**: SSR capability + optimized builds
- **TypeScript**: Type safety + better DX
- **Tailwind CSS**: Rapid UI development


Task
-id: string
-description: string
-dueDate: Date?
-status: TaskStatus
-priority: Priority?
-category: Category?
-createdAt: Date
+updateStatus(newStatus: TaskStatus) : : void
+updateDescription(desc: string) : : void
+isOverdue() : : boolean
TaskForm
-isOpen: boolean
-taskData: Partial<Task>
+handleSubmit() : : void
+handleChange(field: string, value: any) : : void
+validate() : : boolean
+reset() : : void
TaskList
-tasks: Task[]
-filter: string
-searchQuery: string
+renderTasks() : : JSX.Element[]
+handleTaskAction(action: string, taskId: string) : : void
+applyFilters() : : Task[]
Statistics
-tasks: Task[]
+calculateStats() : : TaskStats
+renderProgress() : : JSX.Element
creates/updates
displays/manages
analyzes
TaskManager
-tasks: Map<string, Task>
-storage: StorageService
+addTask(description: string, dueDate?: Date) : : Task
+getTasks() : : Task[]
+getTaskById(id: string) : : Task?
+markComplete(id: string) : : boolean
+deleteTask(id: string) : : boolean
+filterTasks(status?: TaskStatus) : : Task[]
+searchTasks(query: string) : : Task[]
+updateTask(id: string, updates: Partial<Task>) : : boolean
uses
StorageService
-key: string
+save(tasks: Task[]) : : void
+load() : : Task[]
+clear() : : void

Class Diagram

6.2 State Management

- **React Hooks**: Sufficient for app complexity
- **Local Storage**: Meets persistence requirements
- **No External State**: Avoid unnecessary dependencies

7 User Input

- Form validation
- Empty task prevention
- Date format validation

8. Deployment Architecture

8.1 Production Build

```
Static Export → Vercel/CDN → Client Browser
```

9 Hosting

- **Platform**: Vercel (optimized for Next.js)
- **CDN**: Global content delivery
- **SSL**: HTTPS enabled

10. Performance

10.1 Load Time

- **Initial Load**: ~50-100ms (static assets)
- **Task Operations**: Instant (client-side)
- **Filtering**: < 16ms (60fps)

10.2 Storage Limits

- **Theoretical**: ~5MB localStorage
- **Practical**: 1000+ tasks easily supported