

Task Management System

Technical Specification

1. Project Introduction

A lightweight, company-focused web application to create, assign, and track tasks. The app allows managers and staff to create tasks with attachments, set start and end dates, receive reminders, and view a simple dashboard showing tasks grouped by status (Upcoming, Started, Overdue). The system is designed to be easy to build and maintain while covering essential business needs.

2. Project Scope and Objectives

- Allow managers to create tasks and assign them to company users.
 - Provide per-user task lists and notifications (reminders) for tasks.
 - Show a simple dashboard summarizing tasks by status: **Upcoming, Started, Overdue**.
 - Support attachments for tasks.
 - Provide basic admin capabilities to manage users and tasks.
 - Keep the system minimal and easy to deploy and extend.
-

3. Required Features

Core user flows

- **Authentication:** Secure login for employees (email + password).
- **Task creation:** Managers (and permitted users) can create tasks with:
 - Title (required)
 - Description (optional/rich text)
 - Attachment(s) (optional — single or multiple files)

- Start date (date/time)
 - End date (date/time)
 - Priority (low / normal / high)
 - Assignee (one user; can be extended to multiple in future)
- **Task assignment:** Tasks are explicitly assigned to company users; assigned users appear in task views and receive notifications.
- **Task lifecycle:** Users can mark task statuses (e.g., Not started → In progress → Completed). System derives dashboard grouping (Upcoming, Started, Overdue) from start/end dates and status.
- **Dashboard (per user + admin):**
 - Summary counts and quick lists for:
 - **Upcoming** — tasks with `start_date` in the future and not started.
 - **Started** — tasks with status indicating work has begun (or current date \geq `start_date` and not completed).
 - **Overdue** — tasks where current date $>$ `end_date` and not completed.
 - Quick actions: open task, mark complete, view attachments.
- **Notifications / Reminders:**
 - Email and/or in-app notifications to assigned user when:
 - A task is assigned to them.
 - Reminder triggers before due date (configurable, e.g., 24 hrs before).
 - Task becomes overdue.
 - Reminders run via scheduled job/cron; admin can configure reminder timing (e.g., X hours/days before `end_date`).
- **Task list & filters:**
 - List(tasks) with search and filters: by assignee, priority, status, date range.
- **Attachments:**

- Upload and download attachments; files stored locally or on S3-compatible storage.
 - Limit size and allowed file types; scan or validate uploads as appropriate.
 - **Admin Panel:**
 - Manage users (create/edit/deactivate), view all tasks, reassign tasks, and configure reminder schedule.
 - **Export (optional minimal):**
 - Export filtered task list to CSV.
-

4. Database Requirements

(This structure is optional — you can adapt it to your stack.)

Suggested minimal tables and fields:

users

- id (PK)
- name
- email (unique)
- password_hash
- role (enum: admin, manager, user)
- active (boolean)
- created_at, updated_at

tasks

- id (PK)
- title (string, required)
- description (text)
- start_date (datetime, required)
- end_date (datetime, required)

- priority (enum: low, normal, high)
- status (enum: not_started, in_progress, completed, cancelled)
- assignee_id (FK → users.id)
- reporter_id (FK → users.id) — who created the task
- created_at, updated_at

attachments

- id (PK)
- task_id (FK → tasks.id)
- filename
- storage_path (string)
- uploaded_by (FK → users.id)
- created_at

notifications

- id (PK)
- user_id (FK → users.id)
- type (string — assignment, reminder, overdue)
- payload (JSON)
- read (boolean)
- created_at

settings (optional)

- id (PK)
- key (string)
- value (string / JSON) — e.g., reminder offsets

activity_logs (optional)

- id (PK)
- actor_id (FK → users.id)
- action (string)

- object_type (string)
- object_id (int)
- details (JSON)
- created_at

Notes:

- Index tasks on (assignee_id), (status), (start_date), (end_date).
 - Use datetime for start/end to support times.
 - For small company usage, SQLite is acceptable for prototyping; PostgreSQL or MySQL recommended for production.
-

5. Additional Optional Features

- **Role-based access control:** granular permissions (who can create, reassign, delete).
 - **Activity audit:** record who created/updated/deleted tasks and when.
 - **Configurable reminder rules:** per-task or global settings (e.g., remind 48 hours before end_date).
 - **Multi-file attachments** and file previews.
 - **Comment threads on tasks** for discussion and progress notes.
 - **Bulk actions:** bulk reassign, bulk export, bulk status updates.
 - **SLA & escalation:** escalate overdue tasks to managers after X hours/days.
 - **Localization:** Arabic/English with RTL support if needed.
 - **API:** simple REST API for integrations with other company systems.
 - **Mobile-friendly PWA** or a small mobile app later.
-

6. Suggested Technologies

(keep it simple — Laravel is recommended)

- **Recommended / Simple option: Laravel (PHP)** — Laravel provides authentication scaffolding, file storage adapters, scheduling (for reminders), queues, and a clean migration system — making it the simplest and fastest way for students or small teams to implement this app with minimal boilerplate.
 - **Other viable options:**
 - Node.js + Express + PostgreSQL/MySQL
 - Python (Django or Flask) + PostgreSQL/MySQL
 - **Frontend:** Blade templates (Laravel) or a minimal SPA (React/Vue) if you want a richer UI. For a simple app, server-rendered Blade + Alpine.js (small interactions) is sufficient.
 - **Database:** PostgreSQL or MySQL (production). SQLite for local development/prototyping.
 - **Storage:** Local disk for attachments (dev); S3-compatible storage (production).
 - **Notifications:** Laravel mail (SMTP) or external provider (SendGrid/Mailgun); in-app notifications via DB + polling or websockets for realtime.
 - **Jobs & Scheduling:** Use Laravel scheduler + queue worker (or cron) to run reminder jobs.
 - **Hosting/Deployment:** Docker or typical LAMP stack; simple VPS with Nginx + PHP-FPM is fine.
-

7. Project Delivery Conditions

- The project must be uploaded to **GitHub**.
- An invitation must be sent to: **a.s.alashrah@gmail.com**
- Deliver the project with full documentation for deployment and setup, including:
 - Environment variables and secrets (example .env.example)
 - Database migration and seeder instructions
 - How to configure and run scheduled jobs for reminders

- How to configure storage for attachments
- Admin/demo credentials (temporary) and steps to create initial admin user
- Deployment steps for production (Docker or server steps)
- Any additional config for email provider and queues