**Task Management System — Requirements Document**

Date: 2025-08-11

# 1. Introduction

Objective: Build a Task Management System using React, Node.js/Express, and MySQL within 2 weeks (14–15 days).

Build a web application where users can create, categorize, update, delete, and mark tasks complete. Users manage personal task lists, filter by status, and view due dates.

# 2. Technology Stack

* Frontend: React (hooks, React Router), Axios/Fetch.
* Backend: Node.js + Express, JWT auth, bcrypt for hashing.
* Database: MySQL with a query builder or ORM (Sequelize/Knex).
* Tooling: npm/Yarn, ESLint/Prettier, Git & GitHub.

# 3. User Roles

* Unauthenticated Visitor
* Authenticated User
* Admin (optional)

# 4. Core Entities

* User
* Task
* Category

# 5. Functional Requirements

* User signup, login, logout (JWT-based auth).
* CRUD for tasks: title, description, due date, priority, status.
* Categorize tasks and filter by category, status, and due date.
* Responsive UI with list and detail views.
* Basic analytics: count of open vs completed tasks.

# 6. Non‑Functional Requirements

* Responsive design (mobile-first where applicable).
* Error handling on both client and server with meaningful messages.
* Validation on client and server (avoid SQL injection & XSS).
* Use environment variables for secrets (no secrets in repo).
* Basic logging for API requests and errors.

# 7. UI Pages / Screens

* Auth pages: Login, Signup.
* Main list page with filters and pagination where relevant.
* Create/Edit form page with validation and success/error feedback.
* Detail view (where applicable).

# 8. Example API Endpoints

* POST /api/auth/signup — create account
* POST /api/auth/login — obtain JWT
* GET /api/tasks — list tasks (query: status, category)
* POST /api/tasks — create task
* GET /api/tasks/:id — get task
* PUT /api/tasks/:id — update task
* DELETE /api/tasks/:id — delete task
* GET /api/categories — list categories
* POST /api/categories — create category

# 9. Suggested Database Schema (MySQL)

| Table | Columns |
| --- | --- |
| users | id PK, name, email UNIQUE, password\_hash, created\_at |
| categories | id PK, name, user\_id FK(users.id) |
| tasks | id PK, user\_id FK, category\_id FK, title, description, priority ENUM, status ENUM, due\_date, created\_at, updated\_at |

# 10. Validation & Security

* Use server-side validation for all inputs; sanitize and validate using middleware.
* Hash passwords with bcrypt; store only password hashes.
* Use JWT for stateless auth; store token securely (httpOnly cookie or memory).
* Authorize access to resources by ownership (e.g., only owner can edit/delete).

# 11. Milestones (2 Weeks Plan)

1. Day 1–2: Project setup (client + server), Git repo, MySQL schema draft.
2. Day 3–5: Implement core backend APIs and database models.
3. Day 6–8: Build core React pages and integrate APIs.
4. Day 9–10: Authentication, authorization, and protected routes.
5. Day 11–12: Validation, error states, and UI polish.
6. Day 13: Testing (Postman + basic component tests) and bug fixes.
7. Day 14: Final demo, README, deployment (optional).

# 12. Deliverables

* Source code in a public/private GitHub repo with clear README.
* MySQL schema DDL (SQL) and seed data (if any).
* Postman collection (or API docs) covering all endpoints.
* Demo video or walkthrough screenshots (optional but encouraged).

# 13. Acceptance Criteria

* All listed core features functional as per requirements.
* No critical console/server errors; appropriate HTTP status codes.
* Auth-protected routes cannot be accessed without valid token.
* Form validation prevents invalid entries; friendly error messages.
* README explains setup steps, env variables, and run commands.

# 14. Stretch Goals (Optional)

* Dockerize the app (client + server + MySQL).
* Use pagination on large lists and server-side filtering.
* Add role-based access control (admin vs user).
* Add unit tests for services/controllers; component tests for UI.
* CI workflow for lint + tests (GitHub Actions).

# 15. Getting Started (Suggested Steps)

1. Initialize Git repository with two folders: client (React) and server (Node/Express).
2. Create .env.example files for both client and server documenting required variables.
3. Design database schema and create tables via SQL migrations or ORM sync.
4. Implement REST APIs incrementally and test via Postman.
5. Build React pages and wire up API calls with Axios/Fetch.
6. Protect routes/pages that require authentication.
7. Write a README with setup, scripts, and API docs links.