Task Management System Requirement Document

Document Version: 1.0

Date: 15 March 2025

Author: Anshika Gupta

Table of Contents

1. Introduction

This document outlines the requirements for a **full-stack task management system**, designed to enable users to manage their tasks efficiently. The application supports essential features such as **user registration**, **login**, **task creation**, **updating**, **filtering**, and **role-based access control**. To ensure business integrity, it enforces rules like preventing task creation on **public holidays** and **weekends**.

It is designed as a **scalable**, **secure**, and **user-friendly** system, with a frontend built in **Angular 19** and a backend powered by **Node.js and PostgreSQL**.

2. Functional Requirements

- Authentication & User Management (JWT Authentication, Role-Based Access Control)
- CRUD Operations on Tasks
- Task Filtering & Pagination
- Tasks Cannot Be Created on Public Holidays & Weekends
- Only Admins Can Delete Tasks

3. Non-Functional Requirements

3.1 Security

- **JWT Authentication** – Users sign in via **JWT tokens** to access APIs.
- **Role-Based Access Control (RBAC)** – Admins/validators can delete, users cannot.
- **Rate Limiting** – **Express Rate Limit** to prevent API abuse.
- **Soft Deletes for Audit** – Deleted tasks remain stored but hidden.
- **Input Validation** – **Zod/Yup validation** to prevent bad inputs.
- **Database Indexing** – Speeds up querying and prevents performance bottlenecks.
- **Data Encryption** – Passwords are stored using **bcrypt**.
- **Helmet.js** for setting secure HTTP headers.
- Environment variables are managed via .env for secrets like JWT_SECRET.

3.2 Performance Optimization

- Indexed DB queries for fast database access.
- Pagination on task APIs to limit large responses.
- **Public holiday data is cached in a DB table** to avoid external API calls.
- Frontend responsiveness: Used CDS and tailwind.
- Angular lazy loading and module chunking to reduce load time.

- Optimized filtering using Sequelize query conditions.

3.3 Scalability

- Modular, component-based Angular 19 frontend.
- API-first design on the backend (Node.js + Express).
- Dockerized app for containerized deployment.
- Supports horizontal scaling via Docker Compose / Kubernetes.

3.4 Error Handling

- Centralized global error handling using middleware.
- Consistent API response structure with success, message, and data.
- Logging via custom LoggingService.

3.5 Testing & CI/CD

- Unit tests were implemented using Jest / Jasmine.
- Integrated with **CircleCI** for continuous integration.
- Automated build + Docker image generation.
- Test coverage reports to ensure code reliability.

3.6 Swagger API Documentation

- Swagger UI exposed at /api-docs.
- Uses swagger-jsdoc with inline annotations in routes & controllers.
- Contract-first design reviewed across teams before freezing.

3.7 Best Practices & Production Considerations

- Secure headers, HTTPS enforcement, and CORS control
- Persist JWT token via HTTPS only cookie for secure authentication across network.
- SQL Injection protection with Sequelize ORM.
- Validation for input types & enum values (e.g. role, priority).
- Metadata (roles, priorities, public holidays) fetched via global API /api/meta for consistency across UI forms.
- Built using **Angular 19** for the latest performance, SSR, and standalone APIs.

4. Agile Considerations

- Incremental feature rollout
- Sprint-based approach for task completion
- User feedback-driven iterations

4.1 Goals:

- Provide a **secure and role-based task management system**
- Ensure **intuitive and responsive UI**
- Support **task filtering & pagination**
- Prevent **task creation on public holidays & weekends**Allow **only Admins to delete tasks**

4.2 User Stories:

ID	User Story	Benefit
1	As a user, I want to create tasks with a title, description, priority, and due date.	Allows users to manage their workload.
2	As a user, I want to filter tasks based on priority or due date.	Make tasks search efficient.
3	As a user, I want to paginate through tasks.	Improves performance and usability.
4	As an admin, I want to delete tasks.	Maintains data integrity and control.
5	As a user, I want to mark tasks as completed.	Helps in tracking task progress.
6	As a system, I want to **prevent task creation on public holidays & weekends**.	Ensures compliance with business rules.
7	As a user, I want to undo my changes on task	Help is reverting any unexpected changes

4.3 Sprint Breakdown:

Sprint	Features
Sprint 0	Discovery Planning & Setup
Sprint 1	User authentication, Task CRUD API
Sprint 2	Frontend integration, UI design
Sprint 3	Filtering, Pagination

Sprint 4	Security enhancements, Testing

5. Acceptance Criteria

ID	Feature	Acceptance Criteria
1	Authentication	Users can register/login via JWT.
2	Task CRUD	Users can create, update, and view tasks.
3	Filtering & Pagination	Users can filter tasks and paginate results.
4	Public Holiday & Weekend Restriction	Task creation is blocked on public holidays & weekends.
5	Undo Feature	Users can undo the last change within 2 minutes of the change being made
6	Role-Based Access	Admins can delete tasks.

6. Assumptions made:

1. Assuming to show the public holiday dates disabled on the due date picker.
2. Storing public holidays in the database
3. Setting role and priorities values as an enum
4. Sending the public holiday and enum constant values to the front end using app metadata api.
5. Assuming a one-level undo feature from BE and DB and allowing the user to undo
6. Schedule a script to run at the end of the day, to remove the historical records which are older than 20mins.
7. Assuming reset form feature on FE form

7. API Endpoints & JSON Contracts

Please check the JSON contract file on the link provided below:
json-contract.txt

8. Testing Plan Summary

Layer	Tool / Framework	Purpose / Coverage
Backend	Jest + Supertest	Unit & API tests for endpoints, role logic, validation
Frontend	Jasmine + Karma	Component tests, MobX store logic
Integration	Postman + Cypress	End-to-end task flows (create → update → delete)
API Contract Validation	Swagger / OpenAPI	Live testing, schema validation, and API contract enforcement

9. Deployment & DevOps Strategy

- **CI/CD Pipelines (GitHub Actions)**
- **Containerized (Docker)**
- **Production Hosting (AWS, GCP, or Vercel for frontend)**

10. Open Questions

1. **Should we integrate a task calendar view?**
2. **Should users be able to assign tasks to other users?**
3. **Should admins be able to override public holiday restrictions?**