Task Management System Software Requirements Specification (SRS)

1. Product Overview
The Task Management System (TMS) is a MERN-stack application that provides a lightweight workflow for administrators to assign tasks and for employees to manage them. The React frontend communicates with an Express/MongoDB backend through a JSON REST API secured with JWTs.

2. Stakeholders & User Roles
- **Administrator**
- Authenticates via email/password.
- Creates tasks for employees, monitors workload distribution, and manages personal profile settings.
- **Employee**
- Authenticates via email/password.
- Views assigned tasks, updates task status, and manages personal profile settings.
- **System Operator**
- Maintains infrastructure (MongoDB, Node servers) and provides environment secrets.

3. Functional Requirements
3.1 Authentication
- Users must log in with a valid email/password combination stored in MongoDB.
- Successful authentication returns ‘{ user, token }‘, where ‘token‘ is a JWT signed with ‘JWT_SECRET‘.
- Sessions persist client-side via ‘localStorage‘ (‘tms_auth‘). The frontend revalidates the token on load via ‘/api/auth/me‘.
- Users can log out, which clears the stored token and user data.

3.2 Profile Settings
- Both roles can update display name, email, and password via the Profile Settings panel.
- Validation:
- Email uniqueness enforced server-side.
- Password is optional; only provided values are re-hashed with bcrypt before storage.
- After successful update, the backend returns the sanitized user object; the frontend refreshes context/local storage.

3.3 Administrator Module
- Create a task with title, optional description, due date, and employee assignment.
- View a live workload table showing per-employee counts (‘new‘, ‘active‘, ‘completed‘, ‘failed‘).

- See the employee picker populated with names/emails sourced from ‘/api/tasks/team/snapshot‘.

3.4 Employee Module
- View summary cards showing counts of tasks by status.
- Inspect task cards (title, description, due date, status chip).
- Change task state:
- ‘new active‘ (accept task)
- ‘active completed‘ or ‘active failed‘
- Receive inline feedback when tasks are loading or when no tasks exist.

3.5 General UI
- Login page with a single glassmorphic card for credentials and inline error handling.
- Navigation header shows role, user name/email, and logout action in both dashboards.

4. Non-Functional Requirements
- **Security**: JWT for stateless auth, bcrypt hashing, CORS restricted to ‘CLIENT_ORIGIN‘, no sensitive data stored in repo.
- **Performance**: Lightweight API calls; frontend caches auth in context. MongoDB indexes default ‘_id‘ plus unique ‘email‘.
- **Reliability**: Nodemon/dev servers auto-restart on changes; errors handled via centralized middleware returning JSON.
- **Scalability**: Mongoose models ready for future extensions (e.g., more task metadata). Task subdocuments stored inside User for simplicity; can be refactored later into a separate collection if needed.

5. System Architecture
Layer	Technology	Responsibilities
Client	React 19 + Vite + Tailwind CSS 4	Routing (single page), context for auth/data, dashboards, HTTP via Axios
API	Express 5	Routing, validation, auth middleware, controllers
Data	MongoDB 7.x (via Mongoose 9)	Persist users and embedded tasks

Data Flow:
1. User submits credentials ‘/api/auth/login‘.
2. JWT stored in ‘localStorage‘; Axios interceptor attaches it to future requests.
3. Dashboards fetch ‘/api/tasks‘ (self) or ‘/api/tasks/team/snapshot‘ (admin) and render results.
4. Updates (tasks, profile) send authenticated POST/PATCH/PUT requests; backend responds

with updated documents.

6. Data Model
6.1 User
Field	Type	Notes
‘_id‘	ObjectId	Primary key
‘name‘	String	Optional display name (defaults to email prefix)
‘email‘	String	Unique, lowercase
‘password‘	String	Bcrypt hash
‘role‘	Enum(‘admin‘, ‘employee‘)
‘tasks‘	Array\<Task>	Embedded task documents

6.2 Task (embedded)
Field	Type	Notes
‘_id‘	ObjectId	Auto-generated per subdocument
‘title‘	String	Required
‘description‘	String	Optional
‘status‘	Enum(‘new‘, ‘active‘, ‘completed‘, ‘failed‘)
‘dueDate‘	Date	Optional
‘createdAt/updatedAt‘	Date	Added via timestamps

7. API Specification

7.1 Authentication Routes (‘/api/auth‘)
Method & Path	Description	Auth	Payload	Response
‘POST /login‘	Authenticate user	No	‘{ email, password }‘	‘{ user:
{id,name,email,role}, token }‘
‘POST /register‘	(Reserved) Create users	No*	‘{ email, password, role?, name?,
tasks? }‘	Same as login
‘GET /me‘	Fetch current profile	Bearer token		‘{ user }‘
‘PUT /me‘	Update profile	Bearer token	‘{ name?, email?, password? }‘	‘{ user }‘

7.2 Task Routes (‘/api/tasks‘)
Method & Path	Description	Auth	Payload	Response
‘GET /‘	Fetch tasks	Bearer token	Optional ‘?userId=‘ for admins	‘{ owner,

tasks[] }‘ |
‘GET /team/snapshot‘	Admin overview	Admin token		‘{ team: [{ userId, name, email, newTask, active, completed, failed }] }‘
‘POST /‘	Create task	Admin token	‘{ employeeId, title, description?, dueDate?, status? }‘	‘{ message, task }‘
‘PATCH /:taskId‘	Update task status	Bearer token	‘{ status, employeeId? }‘	‘{ message, task }‘

7.3 Health Route
- ‘GET /api/health‘ ‘{ status: "ok", timestamp }‘

8. Frontend Module Breakdown
Component	Responsibility
‘AppContext‘	Loads auth/token, exposes login/logout/task/team/profile actions
‘Login‘	Credential form with inline errors, spinner
‘Nav‘	Greeting, status badge, logout
‘ProfileSettings‘	Shared settings form for admin/employee dashboards
‘AdminDashboard‘	Renders ‘ProfileSettings‘, task creation form, team table
‘EmployeeDashboard‘	Renders ‘ProfileSettings‘, summary cards, interactive task cards
‘ActiveTask‘	Handles status transitions with contextual buttons
‘TaskCount‘, ‘TaskInfo‘, ‘Form‘	Presentational widgets

9. Security Considerations
- **Password Hashing**: bcrypt with salt rounds (10).
- **JWT**: Signed with ‘JWT_SECRET‘, 7-day expiry. Stored client-side; interceptor injects into ‘Authorization: Bearer <token>‘.
- **Authorization**: Middleware ‘protect‘ decodes tokens and loads user; ‘authorizeRoles‘ restricts endpoints like task creation to admins.
- **CORS**: Locked to ‘CLIENT_ORIGIN‘ (default ‘http://localhost:5173‘) with credentials enabled.
- **Error Handling**: Centralized middleware returns descriptive messages and hides stack traces in production.
- **Data Validation**: Server ensures required fields and rejects duplicate emails. Frontend performs basic required-field checks.

10. Environment & Deployment
Backend ‘.env‘
‘‘‘

PORT=5000
MONGO_URI=mongodb://localhost:27017/tms
JWT_SECRET=<strong random string>
CLIENT_ORIGIN=http://localhost:5173
SEED_ADMIN_PASSWORD=123
‘‘‘

Commands:
- Install: ‘cd backend && npm install‘
- Run dev server: ‘npm run dev‘
- Seed demo users/tasks: ‘node src/seed/seedUsers.js‘

Frontend ‘.env‘ (optional)
‘‘‘
VITE_API_URL=http://localhost:5000/api
‘‘‘

Commands:
- Install: ‘cd frontend && npm install‘
- Run dev server: ‘npm run dev‘

11. Future Enhancements (Backlog)
- Task filtering/search for admins.
- Dedicated task collection for better querying at scale.
- Email verification and stronger password rules.
- Role-based dashboards for additional user types.

This SRS summarizes the current state of the Task Management System and should be updated as new features or architectural changes are introduced.