# Software Requirements Specification (SRS)

## Task Management System (TMS) Frontend

**Document Version**: 1.0  
**Date**: June 12, 2025

### 1. Introduction

#### 1.1 Purpose

This Software Requirements Specification (SRS) document defines the functional, non-functional, and technical requirements for the Task Management System (TMS) frontend hosted at <https://fk-tms-frontend.vercel.app/>. The document serves as a blueprint for developers, testers, and stakeholders to ensure the system meets business objectives, provides a seamless user experience, and integrates with backend services. It consolidates requirements from the BRS and FRS, adding detailed technical specifications and system design considerations.

#### 1.2 Scope

The TMS frontend is a Single Page Application (SPA) developed using a modern JavaScript framework (e.g., React, Next.js) and deployed on Vercel’s Frontend Cloud. It enables users to manage tasks through Create, Read, Update, Delete (CRUD) operations, collaborate with team members via comments and notifications, and view task schedules in a team calendar. The system integrates with backend APIs for data persistence and supports responsive, accessible interfaces. This SRS covers user authentication, task management, collaboration features, UI/UX requirements, deployment specifications, and system constraints.

#### 1.3 Definitions, Acronyms, and Abbreviations

* **TMS**: Task Management System
* **SPA**: Single Page Application
* **API**: Application Programming Interface
* **CRUD**: Create, Read, Update, Delete
* **UI/UX**: User Interface/User Experience
* **REST**: Representational State Transfer
* **WCAG**: Web Content Accessibility Guidelines
* **CDN**: Content Delivery Network
* **JWT**: JSON Web Token
* **OAuth**: Open Authorization Protocol

#### 1.4 References

* Vercel Documentation: <https://vercel.com/docs>
* WCAG 2.1 Guidelines: <https://www.w3.org/WAI/standards-guidelines/wcag/>

### 2. Overall Description

#### 2.1 Product Perspective

The TMS frontend is a client-side application that interacts with users through a web browser and communicates with a backend system via REST or GraphQL APIs. It is hosted on Vercel’s Frontend Cloud, leveraging automatic scaling, global CDN distribution, and preview deployments. The system is part of a larger TMS ecosystem, where the backend handles data storage, business logic, and third-party integrations (e.g., email notifications).

#### 2.2 Product Functions

* **User Management**: Register, log in, log out, and manage user roles (Admin, Team Member).
* **Task Management**: Create, view, update, and delete tasks with attributes like title, description, due date, priority, status, and assignees.
* **Collaboration**: Add comments to tasks, receive notifications for updates, and view a team calendar.
* **UI/UX**: Provide a responsive, accessible SPA with light/dark themes.
* **Integration**: Connect with backend APIs for data operations and authentication.
* **Deployment**: Host on Vercel with support for production and preview environments.

#### 2.3 User Classes and Characteristics

* **End Users (Team Members)**: Create and manage tasks, collaborate with teams, and view schedules. Require intuitive UI and minimal technical expertise.
* **Admins**: Manage users and tasks with elevated permissions. Need access to user management and task deletion controls.
* **Developers**: Implement and maintain the frontend, requiring clear API documentation and Vercel deployment workflows.
* **Testers**: Verify functionality, performance, and accessibility using automated and manual testing tools.

#### 2.4 Operating Environment

* **Client-Side**: Runs in modern web browsers (Chrome, Firefox, Safari, Edge) with JavaScript enabled.
* **Hosting**: Deployed on Vercel’s Frontend Cloud with serverless architecture and CDN.
* **Backend**: Assumed to be a separate system providing REST/GraphQL APIs, hosted independently.
* **Devices**: Supports desktop (min 1024px), tablet (min 768px), and mobile (min 320px) screens.

#### 2.5 Design and Implementation Constraints

* The frontend must use a JavaScript framework compatible with Vercel (e.g., React, Next.js).
* No local file I/O or direct database access; all data operations must use APIs.
* The system must comply with WCAG 2.1 Level AA for accessibility.
* JavaScript must be enabled in the user’s browser.

#### 2.6 Assumptions and Dependencies

* **Assumptions**:
  + Backend APIs are available, documented, and return JSON data.
  + Users have modern browsers with JavaScript enabled.
  + Vercel provides sufficient resources for hosting and scaling.
* **Dependencies**:
  + Backend system for API services.
  + Third-party OAuth providers (e.g., Google, GitHub) for authentication.
  + Vercel’s infrastructure for deployment and scaling.

### 3. Functional Requirements

#### 3.1 User Authentication and Authorization

* **SRS-001**: The system shall provide a login interface for email/password and OAuth (Google, GitHub) authentication.
  + Input: Email/password or OAuth credentials
  + Output: JWT token stored in browser (HTTP-only cookie or local storage)
  + Validation: Email format (regex: ^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$), password (min 8 characters, 1 uppercase, 1 number)
  + Error Handling: Display “Invalid credentials” for failed login attempts
* **SRS-002**: The system shall provide a registration interface.
  + Input: Email, password, full name (max 50 characters)
  + Output: User account created, confirmation email sent via backend API
  + Validation: Unique email, password confirmation match
  + Error Handling: Display “Email already exists” for duplicate emails
* **SRS-003**: The system shall enforce role-based access control.
  + Roles: Admin (full access), Team Member (task CRUD, comments)
  + Implementation: API responses include user role; frontend restricts UI elements accordingly
* **SRS-004**: The system shall provide a logout function.
  + Output: Clear session data, redirect to login page
  + Implementation: Remove JWT token from storage

#### 3.2 Task Management

* **SRS-005**: The system shall provide a task creation form.
  + Input: Title (required, max 100 characters), description (optional, max 1000 characters), due date (optional, ISO format), priority (Low, Medium, High), status (To-Do, In Progress, Done), assignees (select from user list)
  + Output: Task created via POST /api/tasks, displayed in dashboard
  + Validation: Non-empty title, valid date format
  + Error Handling: Display “Title required” or “Invalid date”
* **SRS-006**: The system shall display tasks in a paginated table.
  + Output: Table with columns (title, status, priority, due date, assignees), 10 tasks per page
  + Implementation: GET /api/tasks with query params (page, limit)
  + Features: Clickable rows to view/edit tasks
* **SRS-007**: The system shall provide a task edit form.
  + Input: Pre-populated fields from GET /api/tasks/:id
  + Output: Task updated via PUT /api/tasks/:id, reflected in dashboard
  + Validation: Same as SRS-005
* **SRS-008**: The system shall allow task deletion with a confirmation modal.
  + Output: Task removed via DELETE /api/tasks/:id
  + Restriction: Only Admins or task creators can delete
  + Error Handling: Display “Unauthorized” for restricted users
* **SRS-009**: The system shall provide task filtering.
  + Input: Dropdowns for status, priority; date picker for due date range
  + Output: Filtered task list via GET /api/tasks with query params (status, priority, dueDateStart, dueDateEnd)
* **SRS-010**: The system shall provide task sorting.
  + Input: Dropdown for sort field (title, due date, priority) and order (asc, desc)
  + Output: Sorted task list via GET /api/tasks with query params (sortBy, sortOrder)

#### 3.3 Collaboration Features

* **SRS-011**: The system shall allow users to add task comments.
  + Input: Comment text (max 500 characters)
  + Output: Comment posted via POST /api/tasks/:id/comments, displayed with user name and timestamp
  + Validation: Non-empty comment
  + Error Handling: Display “Comment too long”
* **SRS-012**: The system shall send notifications for task updates.
  + Events: Task assignment, status change, new comment
  + Output: In-app notification (real-time or on refresh), optional email via backend API
  + Configuration: Toggle email notifications in settings (PATCH /api/users/:id/settings)
* **SRS-013**: The system shall display a team calendar.
  + Input: Month/year selection
  + Output: Calendar with tasks marked on due dates (GET /api/tasks/calendar), clickable to view details
  + Implementation: Use a library like FullCalendar

#### 3.4 User Interface and Experience

* **SRS-014**: The system shall be an SPA using React or Next.js.
  + Implementation: Client-side routing with React Router or Next.js Pages Router
  + Output: Seamless navigation without full page reloads
* **SRS-015**: The system shall be responsive across devices.
  + Breakpoints: Desktop (≥1024px), tablet (≥768px), mobile (≥320px)
  + Implementation: CSS media queries or Tailwind CSS
* **SRS-016**: The system shall support light and dark themes.
  + Implementation: CSS variables, toggle stored in local storage
  + Output: Theme applied across all pages
* **SRS-017**: The system shall comply with WCAG 2.1 Level AA.
  + Features: ARIA labels, keyboard navigation, color contrast (min 4.5:1)
  + Tools: Test with axe or Lighthouse
* **SRS-018**: The system shall display loading indicators for API calls.
  + Output: Spinner for operations >500ms
  + Implementation: Use React state or context
* **SRS-019**: The system shall display user-friendly error messages.
  + Examples: “Failed to load tasks, please try again”
  + Implementation: Handle API errors with try-catch, display via toast notifications

#### 3.5 API Integration

* **SRS-020**: The system shall integrate with backend APIs.
  + Protocols: REST or GraphQL
  + Endpoints (REST example):
    - POST /api/auth/login: Authenticate user
    - POST /api/users: Register user
    - GET/POST/PUT/DELETE /api/tasks: Task CRUD
    - POST /api/tasks/:id/comments: Add comment
    - GET /api/tasks/calendar: Calendar data
  + Authentication: Bearer token in headers
* **SRS-021**: The system shall cache API responses.
  + Implementation: Use browser local storage or library like SWR
  + Cache duration: 5 minutes, invalidated on task updates

#### 3.6 Deployment and Hosting

* **SRS-022**: The system shall be deployed on Vercel’s Frontend Cloud.
  + Features: Automatic scaling, global CDN, domain management
  + Output: Application accessible at <https://fk-tms-frontend.vercel.app/>
* **SRS-023**: The system shall support preview deployments.
  + Implementation: Vercel Git integration for branch-based previews
  + Output: Unique URLs for testing non-production changes
* **SRS-024**: The system shall use Vercel environment variables.
  + Examples: API\_BASE\_URL, OAUTH\_CLIENT\_ID
  + Implementation: Configure in Vercel dashboard

### 4. Non-Functional Requirements

#### 4.1 Performance

* **SRS-025**: Initial page load time shall be ≤2 seconds on a 4G connection.
  + Implementation: Optimize bundle size, use Vercel’s edge caching
* **SRS-026**: API calls shall timeout after 10 seconds.
  + Implementation: Configure fetch/axios with timeout option
* **SRS-027**: The system shall handle 1,000 concurrent users with <5% error rate.
  + Implementation: Leverage Vercel’s serverless scaling

#### 4.2 Security

* **SRS-028**: All API requests shall use HTTPS with TLS 1.2 or higher.
* **SRS-029**: The system shall sanitize inputs to prevent XSS and injection attacks.
  + Implementation: Use libraries like DOMPurify
* **SRS-030**: Session tokens shall be stored in HTTP-only cookies or secure local storage.
  + Implementation: Set secure and SameSite attributes for cookies

#### 4.3 Reliability

* **SRS-031**: The system shall achieve 99.9% uptime.
  + Implementation: Rely on Vercel’s infrastructure SLAs
* **SRS-032**: The system shall handle API failures gracefully.
  + Output: Fallback UI with retry option

#### 4.4 Compatibility

* **SRS-033**: The system shall support Chrome, Firefox, Safari, and Edge (latest versions).
* **SRS-034**: The system shall require JavaScript, with a fallback message if disabled.
  + Output: “Please enable JavaScript to use this application”

#### 4.5 Scalability

* **SRS-035**: The system shall scale automatically with traffic.
  + Implementation: Vercel’s serverless architecture

#### 4.6 Maintainability

* **SRS-036**: The codebase shall follow modular design principles.
  + Implementation: Use component-based architecture, TypeScript for type safety
* **SRS-037**: The system shall include unit tests for critical components.
  + Tools: Jest, React Testing Library

### 5. System Architecture

#### 5.1 Overview

* **Client-Side**: React/Next.js SPA with client-side routing and state management (e.g., Redux, React Context).
* **Backend Integration**: REST or GraphQL APIs, accessed via fetch/axios.
* **Hosting**: Vercel’s Frontend Cloud with serverless functions for SSR/SSG if using Next.js.
* **Data Flow**: User inputs → Frontend → API calls → Backend → Database → Response → Frontend.

#### 5.2 Component Diagram

* **UI Components**: Login, Registration, Dashboard, Task Form, Task List, Calendar, Settings
* **Services**: API Service (handles HTTP requests), Auth Service (manages tokens), Cache Service
* **Libraries**: React/Next.js, Tailwind CSS, FullCalendar, axios, DOMPurify

#### 5.3 Data Model (Frontend Perspective)

* **User**: { id, email, fullName, role, settings { theme, notifications } }
* **Task**: { id, title, description, dueDate, priority, status, assignees, comments }
* **Comment**: { id, taskId, userId, text, timestamp }

### 6. External Interfaces

#### 6.1 User Interfaces

* **Login Page**: Email/password fields, OAuth buttons, error messages
* **Registration Page**: Email, password, full name fields, confirmation message
* **Dashboard**: Task table, filters, sort options, pagination
* **Task Form**: Fields for title, description, due date, priority, status, assignees
* **Task Details**: Task data, comment section, update/delete buttons
* **Calendar**: Monthly view with task markers
* **Settings**: Theme toggle, notification preferences

#### 6.2 Hardware Interfaces

* None (browser-based application).

#### 6.3 Software Interfaces

* **Backend API**: REST/GraphQL endpoints for authentication, tasks, and comments.
* **OAuth Providers**: Google, GitHub for authentication.
* **Vercel**: Hosting, environment variables, preview deployments.

#### 6.4 Communication Interfaces

* **HTTPS**: For all API requests.
* **WebSocket (Optional)**: For real-time notifications (future consideration).

### 7. Acceptance Criteria

* Users can register, log in, and log out without errors.
* CRUD operations on tasks are functional with valid inputs.
* Filters, sorting, and pagination work as specified.
* Comments and notifications are operational for collaboration.
* The UI is responsive, accessible (WCAG 2.1 Level AA), and supports themes.
* API integration is seamless with proper error handling.
* Deployments on Vercel complete successfully, with preview and production environments accessible.
* Performance metrics (load time ≤2s, 1,000 concurrent users) are met.

### 8. Risks and Mitigation

* **Risk 1**: Backend API delays or changes.
  + **Mitigation**: Use mock APIs and contract testing during development.
* **Risk 2**: Accessibility compliance gaps.
  + **Mitigation**: Conduct regular audits with tools like axe.
* **Risk 3**: Performance degradation under load.
  + **Mitigation**: Optimize assets, use Vercel’s edge caching, and implement lazy loading.
* **Risk 4**: Browser compatibility issues.
  + **Mitigation**: Test across supported browsers using BrowserStack or similar tools.

### 9. Future Considerations

* **Real-Time Features**: Add WebSocket support for live updates.
* **Third-Party Integrations**: Connect with Slack, Google Calendar, or Microsoft Teams.
* **Offline Support**: Implement PWA features for offline task viewing/editing.
* **Analytics Dashboard**: Add task completion trends and productivity metrics.

### 10. Approval

**Prepared by**: Project Manager  
**Reviewed by**: M.Abdullah Awais  
**Date**: June 12, 2025