**Functional Requirements Specification (FRS)**

**Task Management System (TMS) Frontend**

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

**1. Introduction**

**1.1 Purpose**

This Functional Requirements Specification (FRS) document specifies the detailed functional and technical requirements for the Task Management System (TMS) frontend hosted at <https://fk-tms-frontend.vercel.app/>. The document defines the system’s capabilities, user interactions, and integration points to ensure the application meets business objectives outlined in the associated BRS.

**1.2 Scope**

The TMS frontend is a Single Page Application (SPA) built with 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, and interact with backend APIs for data persistence. The FRS covers user authentication, task management, collaboration features, UI/UX specifications, and deployment requirements.

**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

**1.4 References**

* BRS for Task Management System Frontend (artifact\_id: 57325d2a-4425-43bd-954c-cd0947b6a135)
* Vercel Frontend Cloud Documentation: <https://vercel.com/docs>
* Web Accessibility Standards: <https://www.w3.org/WAI/standards-guidelines/wcag/>

**2. System Overview**

The TMS frontend is a web-based application that provides an intuitive interface for task management and team collaboration. It integrates with a backend API for data operations and is hosted on Vercel for scalability and performance. Key features include user authentication, task CRUD operations, task filtering/sorting, team collaboration tools (e.g., comments, notifications), and a responsive, accessible UI.

**3. Functional Requirements**

**3.1 User Authentication and Authorization**

* **FR-001**: The system shall provide a login page where users can authenticate using email/password or third-party providers (e.g., Google, GitHub OAuth).
  + Input: Email/password or OAuth credentials
  + Output: JWT token stored in browser for session management
  + Validation: Email format, password strength (8+ characters, 1 uppercase, 1 number)
* **FR-002**: The system shall provide a registration page for new users to create accounts.
  + Input: Email, password, full name
  + Output: User account created, confirmation email sent
  + Validation: Unique email, matching password confirmation
* **FR-003**: The system shall support role-based access control with at least two roles: Admin and Team Member.
  + Admin: Can manage users and delete tasks
  + Team Member: Can create, update, and view tasks
* **FR-004**: The system shall provide a logout functionality that clears session data from the browser.
  + Output: User redirected to login page

**3.2 Task Management**

* **FR-005**: The system shall allow users to create tasks via a form.
  + Input: Title (required, max 100 characters), description (optional, max 1000 characters), due date (optional), priority (Low, Medium, High), status (To-Do, In Progress, Done), assignees (one or more users)
  + Output: Task created and displayed in user’s dashboard
  + Validation: Title must not be empty
* **FR-006**: The system shall display a list of tasks in a user’s dashboard with columns for title, status, priority, due date, and assignees.
  + Output: Paginated list (10 tasks per page) with clickable rows to view/edit tasks
* **FR-007**: The system shall allow users to update tasks via an edit form pre-populated with existing task data.
  + Input: Same as FR-005
  + Output: Updated task reflected in dashboard
* **FR-008**: The system shall allow users to delete tasks with a confirmation prompt.
  + Output: Task removed from dashboard
  + Restriction: Only Admins or task creators can delete tasks
* **FR-009**: The system shall provide filters to view tasks by status, priority, or due date range.
  + Input: Filter selections via dropdowns or date pickers
  + Output: Filtered task list
* **FR-010**: The system shall provide sorting options for tasks by title, due date, or priority.
  + Input: Sort selection via dropdown
  + Output: Sorted task list

**3.3 Collaboration Features**

* **FR-011**: The system shall allow users to add comments to tasks.
  + Input: Comment text (max 500 characters)
  + Output: Comment displayed in task details with user name and timestamp
* **FR-012**: The system shall send notifications for task updates (e.g., assignment, status change, new comment).
  + Output: In-app notification and optional email
  + Configuration: Users can toggle email notifications in settings
* **FR-013**: The system shall display a team calendar view showing task due dates.
  + Input: Month/year selection
  + Output: Calendar with tasks marked on due dates, clickable to view details

**3.4 User Interface and Experience**

* **FR-014**: The system shall be built as an SPA using React or Next.js for seamless navigation.
  + Output: No full page reloads during navigation
* **FR-015**: The system shall provide a responsive design compatible with desktop (min 1024px), tablet (min 768px), and mobile (min 320px) screens.
  + Output: Layout adjusts dynamically based on screen size
* **FR-016**: The system shall support light and dark themes, toggleable via a settings menu.
  + Output: Theme applied across all pages, persisted in local storage
* **FR-017**: The system shall comply with WCAG 2.1 Level AA standards.
  + Features: Keyboard navigation, screen reader support, sufficient color contrast (min 4.5:1)
* **FR-018**: The system shall display a loading indicator during API calls.
  + Output: Spinner or progress bar for operations >500ms
* **FR-019**: The system shall display error messages for failed operations (e.g., invalid input, API errors).
  + Output: User-friendly message with retry option where applicable

**3.5 API Integration**

* **FR-020**: The system shall integrate with backend APIs using REST or GraphQL for all data operations.
  + Endpoints (example):
    - POST /api/tasks: Create task
    - GET /api/tasks: List tasks
    - PUT /api/tasks/:id: Update task
    - DELETE /api/tasks/:id: Delete task
  + Authentication: Bearer token in request headers
* **FR-021**: The system shall cache API responses for frequently accessed data (e.g., task list) to reduce server load.
  + Cache duration: 5 minutes, invalidated on task updates

**3.6 Deployment and Hosting**

* **FR-022**: The system shall be deployed on Vercel’s Frontend Cloud with automatic scaling.
  + Output: Application accessible globally via CDN
* **FR-023**: The system shall support preview deployments for each git push to non-production branches.
  + Output: Unique preview URL generated by Vercel
* **FR-024**: The system shall use Vercel’s environment variables for API keys and configuration.
  + Example: API\_BASE\_URL for backend endpoint

**4. Non-Functional Requirements**

**4.1 Performance**

* **FR-025**: Initial page load time shall not exceed 2 seconds on a 4G connection.
* **FR-026**: API response times shall be handled gracefully, with timeouts after 10 seconds.
* **FR-027**: The system shall support up to 1,000 concurrent users with <5% error rate.

**4.2 Security**

* **FR-028**: All API requests shall use HTTPS with TLS 1.2 or higher.
* **FR-029**: The system shall sanitize user inputs to prevent XSS and SQL injection attacks.
* **FR-030**: Session tokens shall be stored securely in HTTP-only cookies or local storage with proper CSRF protection.

**4.3 Reliability**

* **FR-031**: The system shall achieve 99.9% uptime, leveraging Vercel’s infrastructure.
* **FR-032**: The system shall handle API failures gracefully with fallback UI messages.

**4.4 Compatibility**

* **FR-033**: The system shall support the latest versions of Chrome, Firefox, Safari, and Edge.
* **FR-034**: The system shall require JavaScript to be enabled, with a fallback message if disabled.

**5. Assumptions and Constraints**

**5.1 Assumptions**

* Backend APIs are available, documented, and return JSON data.
* Users have modern browsers with JavaScript enabled.
* Vercel’s Frontend Cloud provides sufficient resources for hosting and scaling.

**5.2 Constraints**

* The frontend shall not perform direct database operations; all data access is via APIs.
* The application must use a JavaScript framework compatible with Vercel (e.g., React, Next.js).
* No local file I/O is permitted in the frontend code.

**6. System Interfaces**

**6.1 User Interface**

* Login/Registration pages
* Dashboard with task list and filters
* Task creation/edit forms
* Task details page with comments
* Team calendar view
* Settings page for theme and notification preferences

**6.2 External Interfaces**

* Backend API: REST or GraphQL endpoints for data operations
* Third-party OAuth providers: Google, GitHub for authentication
* Email service: For notification delivery (e.g., via backend API)

**7. Acceptance Criteria**

* Users can register, log in, and log out successfully.
* Users can perform CRUD operations on tasks with valid inputs.
* Task filters and sorting options function as specified.
* Comments and notifications are operational for task collaboration.
* The UI is responsive, accessible, and supports light/dark modes.
* The application integrates with backend APIs without errors.
* Deployments on Vercel complete successfully, with preview and production environments accessible.

**8. Risks and Mitigation**

* **Risk 1**: Backend API changes may break frontend functionality.
  + **Mitigation**: Use API versioning and mock data for development.
* **Risk 2**: Accessibility compliance may increase development effort.
  + **Mitigation**: Use accessibility testing tools (e.g., axe) during QA.
* **Risk 3**: High API latency may impact user experience.
  + **Mitigation**: Implement caching and optimistic updates.

**9. Future Considerations**

* Support for real-time collaboration using WebSockets.
* Integration with third-party tools (e.g., Slack, Google Calendar).
* Offline support via Progressive Web App (PWA) features.

**10. Approval**

**Prepared by**: Sarim Nadeem (SQA & PM)  
**Reviewed by**: [M.Abdullah Awais]  
**Approved by**: [M.Abdullah Awais]  
**Date**: June 12, 2025