MERN Note App

Document Version Control

Date Issued	Version	Description	Author
2025-04-06	1.0	HLD	Siddharth Rukadikar

Abstract

The **MERN Note App** is a full-stack web application designed to provide users with a seamless experience in creating, managing, and organizing their notes. Leveraging the MERN (MongoDB, Express.js, React.js, Node.js) stack, the application offers features such as user authentication, CRUD (Create, Read, Update, Delete) operations for notes, and a responsive user interface. The app ensures data persistence and accessibility across devices, enhancing user productivity and data security.

1 Introduction

○ Purpose of this High-Level Design Document

This High-Level Design (HLD) document aims to provide a comprehensive overview of the **MERN Note App** architecture and design. It serves as a blueprint for the development team, outlining the system's structure, components, and interactions. The document facilitates a clear understanding of the application's design, ensuring alignment among stakeholders and guiding the development process.

○ Scope

The HLD encompasses the overall architecture of the **MERN Note App**, detailing the frontend and backend components, data flow, and integration points. It covers user authentication, note management functionalities, and the technologies employed. The document also addresses non-functional requirements such as performance, scalability, and security considerations.

○ Definitions

- **MERN Stack:** A collection of technologies including MongoDB, Express.js, React.js, and Node.js used for building full-stack web applications.
- **CRUD Operations:** Refers to Create, Read, Update, and Delete functionalities essential for data management.
- **JWT (JSON Web Token)**: A compact, URL-safe means of representing claims to be transferred between two parties, commonly used for authentication.

2 General Description

○ Product Perspective

The **MERN Note App** is designed as a standalone web application that enables users to manage their notes efficiently. It operates in a client-server architecture, with the frontend developed using React.js and the backend utilizing Node.js with Express.js. MongoDB serves as the database for storing user information and notes. The application aims to provide a responsive and intuitive user interface, ensuring accessibility across various devices.

○ Problem statement

In the digital age, individuals require reliable tools to document and organize their thoughts, tasks, and important information. Existing solutions may lack customization, offline capabilities, or seamless synchronization across devices. The **MERN Note App** addresses these challenges by offering a user-friendly platform with robust features for note-taking and management.

○ PROPOSED SOLUTION

The proposed solution is a web-based application that allows users to:

- **Register and Authenticate**: Secure user registration and login functionalities using JWT.
- **Manage Notes**: Perform CRUD operations on notes, including features like pinning important notes.
- **Search and Organize**: Efficient search functionality to locate notes quickly.
- **Responsive Design**: Ensure accessibility and usability across desktops and mobile devices.

○ FURTHER IMPROVEMENTS

Enabling multiple users to share and collaborate on notes in real-time. Allowing users to access and edit notes without an internet connection, with synchronization upon reconnection.

○ Technical Requirements

- **Frontend**: React.js with Vite for fast development and optimized builds.
- **Backend**: Node.js with Express.js for handling API requests and business logic.
- **Database**: MongoDB for data storage, offering flexibility and scalability.
- **Authentication**: Implementation of JWT for secure user authentication.

○ Data Requirements

- **User Data**: Secure storage of user credentials and profile information.
- **Notes Data**: Efficient storage and retrieval of notes, including metadata such as creation date, last modified date, and pinned status

○ Tools used

- **Integrated Development Environment (IDE)**: Visual Studio Code for code development.
- **Version Control**: Git and GitHub for source code management and collaboration.
- **Package Managers**: npm for managing project dependencies.

○ Constraints

- **Internet Dependency**: The application requires an active internet connection for full functionality, though future versions may incorporate offline capabilities.
- **Browser Compatibility**: The application is optimized for modern web browsers; older browsers may not support all features.

○ Assumptions

- Users have basic proficiency with web applications and note-taking tools.
- Users will access the application through devices with internet connectivity and modern web browsers.

3 Design Details

○ Process Flow

User Registration and Authentication:

- **Registration**: Users provide their email and password to create an account. The backend hashes the password and stores the user data in MongoDB.
- **Login**: Users enter their credentials. The backend verifies the credentials and, upon success, issues a JWT for session management.

Note Management:

- **Create Note:** Users can add a new note by entering a title, description, and selecting a category. The note is sent to the backend and saved in MongoDB.
- **Read/View Notes:** On login, the frontend fetches all notes using the authenticated API. Notes are displayed using UI cards.
- **Update/Edit Note:** Users can click on any note to update its contents. The updated data is sent via a PUT API and reflected in the frontend UI
- **Delete Note:** User can delete a note.
- **Pin/Unpin Note:** Notes can be pinned to appear on top. A toggle button changes the pin status via a PUT API.
- **Search Functionality:** A real-time search bar filters notes based on title or content.

UI Responsiveness:

- **Mobile Navigation:** Navbar and sidebar adjust based on screen size.
- **Hamburger Menu:** On smaller screens, user profile and logout options are shown in a collapsible right-drawer menu.
- **Search Bar:** Always stays inline with the navbar for easy access.

State Management:

- **Redux Toolkit:** Used to manage user authentication, notes, search input, and UI state globally.

Notifications:

- **Toast Alerts:** Success and error notifications are shown using libraries like react-toastify.
- **Use Cases:** Login success, note creation, updates, deletions, and errors are all communicated via toast.

Error Handling:

- **Efficient Rendering:** Notes are fetched once and stored in Redux for minimal re-fetching.
- **Optimistic Updates:** UI is updated immediately on create/update/delete, followed by server confirmation

Event log

- **User Login:** Logs timestamp and user email upon successful login.
- **Note Operations:** Each CRUD operation (Create, Read, Update, Delete) on notes is logged with:
  - Action type (create/edit/delete)
  - Note ID
  - User ID
  - Timestamp

Error Handling

- Frontend:
  - Form validations for empty fields, minimum length, etc.
  - API errors (401, 403, 404, 500) are caught and shown via toast messages.
  - Loading and error states managed in Redux store.
- Backend:
  - Middleware handles unknown routes and server-side exceptions.
  - Custom error handler middleware returns consistent error responses.
  - JWT errors (token expired, invalid) are handled gracefully with 401 responses

Performance

- **Efficient Queries:** MongoDB indexes used for user ID and timestamps to speed up data retrieval.
- If notes exceed a certain count infinite scroll can be implemented.

Reusability

- Frontend:
  - Lightweight React components.
  - Toast notifications and UI transitions are kept minimal for performance.
- Backend:
  - Optimized Node.js APIs with async/await and proper connection pooling with MongoDB.

- Minimal third-party libraries to reduce overhead.

- Database:
  - MongoDB Atlas with minimal document schema for quick reads/writes.
  - Notes are indexed by userId to isolate user data efficiently.

Application Compatibility

- **Browsers Supported:** Chrome, Firefox, Edge, Safari (latest versions).
- **Responsive Design:** Fully responsive on desktops and mobile devices

Resource Utilization

- Frontend (React):
  - Lightweight functional components
  - Code splitting for heavy components.
- Backend (Node.js + Express):
  - Asynchronous request handling.
  - Proper MongoDB indexing for queries.
- Storage:
  - Notes data stored efficiently in MongoDB Atlas.
  - User tokens and minimal metadata stored in LocalStorage.

Deployment

- Frontend:
  - Hosted on Render with continuous deployment from GitHub.
  - Environment-based configurations via .env.
- Backend:
  - Deployed on Render.
  - Node environment variables securely managed.
  - Uses MongoDB Atlas for cloud-based document storage.
- Version Control:
  - Git with GitHub for codebase management.

4 Dashboards

- User Dashboard:
    - Visual overview of user notes, pinned notes, and Profile

○ KPIs (Key Performance Indicators)

○ User Metrics:

- Total users registered.
- Daily/weekly active users.
- Login success vs. failure rate.

○ Note Metrics:

- Total number of notes.
- Average notes per user.
- Notes created/deleted per day

○ Performance Metrics:

- Average response time of APIs.
- Error rate (frontend/backend).

5 Conclusion

This document outlines the architecture, functionality, and deployment of the Note Management App. Built with React, Node.js, and MongoDB, the application offers secure authentication, intuitive UI, and robust CRUD operations for personal notes. The design ensures scalability, responsiveness, and maintainability, making it adaptable for future enhancements such as real-time collaboration, cloud sync, or AI-based note suggestions