Simple Notes Application Requirements Document

1. Introduction

1.1 Purpose

The purpose of this document is to outline the requirements for a simple notes application. The application will allow users to register, log in, and manage their notes (add, edit, delete). The application will be developed using Next.js for the frontend, Express.js for the backend, and an in-memory database for storage.

1.2 Scope

This document covers the functional and non-functional requirements of the application, including user authentication, notes management, and the technology stack to be used.

2. Functional Requirements

2.1 User Authentication

2.1.1 User Registration

Description: Users should be able to register by providing their email and password.

Input: Email, Password

Output: Success or error message

Validation: Email must be unique and in a valid format. Password must meet security criteria (e.g., minimum length).

2.1.2 User Login

Description: Users should be able to log in using their registered email and password.

Input: Email, Password

Output: Success or error message, authentication token

Validation: Email and password must match the registered credentials.

2.1.3 User Logout

Description: Users should be able to log out, which will invalidate their session.

Output: Success message

2.2 Notes Management

2.2.1 Add Note

Description: Users should be able to add a new note with a title and content.

Input: Title, Content

Output: Success or error message, new note details

Validation: Title and content must not be empty.

2.2.2 Edit Note

Description: Users should be able to edit an existing note's title and content.

Input: Note ID, Title, Content

Output: Success or error message, updated note details

Validation: Note ID must exist. Title and content must not be empty.

2.2.3 Delete Note

Description: Users should be able to delete an existing note.

Input: Note ID

Output: Success or error message

Validation: Note ID must exist.

2.2.4 View Notes

Description: Users should be able to view a list of their notes.

Output: List of notes (ID, Title, Content, Created At, Updated At)

3. Non-Functional Requirements

3.1 Performance

The application should respond to user actions within 2 seconds.

The application should be able to handle up to 100 concurrent users.

3.2 Security

Passwords must be hashed before storing.

Authentication tokens must be securely generated and validated.

The application must prevent common security vulnerabilities (e.g., SQL injection, XSS).

3.3 Usability

The user interface should be intuitive and easy to navigate.

Error messages should be clear and helpful.

3.4 Reliability

The application should have an uptime of 99.9%.

The application should handle unexpected errors gracefully.

4. Technology Stack

4.1 Frontend

Framework: Next.js

Language: JavaScript/TypeScript

Styling: CSS Modules, Tailwind CSS (optional)

4.2 Backend

Framework: Express.js

Language: JavaScript/TypeScript

Database: In-memory storage (e.g., an array or a simple in-memory database like Redis for more advanced use cases)

4.3 Authentication

Method: Email and password

Token: JSON Web Tokens (JWT)

5. Architecture

5.1 Overview

The application will follow a client-server architecture. The frontend will be developed using Next.js and will communicate with the backend via RESTful APIs. The backend will be developed using Express.js and will handle user authentication and notes management. An in-memory database will be used for storing user credentials and notes.

5.2 Components

5.2.1 Frontend

Pages: Register, Login, Dashboard (Notes Management)

Components: Layout, Navbar, NoteForm, NoteList, NoteItem

5.2.2 Backend

Routes:

/api/auth/register (POST): User registration

/api/auth/login (POST): User login

/api/auth/logout (POST): User logout

/api/notes (GET): Get all notes

/api/notes (POST): Add a new note

/api/notes/:id (PUT): Edit a note

/api/notes/:id (DELETE): Delete a note

5.2.3 Database

In-memory Storage: Arrays or a simple in-memory database like Redis

6. API Specifications

6.1 User Registration

Endpoint: /api/auth/register

Method: POST

Request Body: { "email": "user@example.com", "password": "password123" }

Response: { "message": "User registered successfully" }

6.2 User Login

Endpoint: /api/auth/login

Method: POST

Request Body: { "email": "user@example.com", "password": "password123" }

Response: { "token": "jwt-token" }

6.3 User Logout

Endpoint: /api/auth/logout

Method: POST

Response: { "message": "User logged out successfully" }

6.4 Add Note

Endpoint: /api/notes

Method: POST

Request Body: { "title": "Note Title", "content": "Note Content" }

Response: { "id": 1, "title": "Note Title", "content": "Note Content", "createdAt": "timestamp", "updatedAt": "timestamp" }

6.5 Edit Note

Endpoint: /api/notes/:id

Method: PUT

Request Body: { "title": "Updated Title", "content": "Updated Content" }

Response: { "id": 1, "title": "Updated Title", "content": "Updated Content", "createdAt": "timestamp", "updatedAt": "timestamp" }

6.6 Delete Note

Endpoint: /api/notes/:id

Method: DELETE

Response: { "message": "Note deleted successfully" }

6.7 View Notes

Endpoint: /api/notes

Method: GET

Response: [ { "id": 1, "title": "Note Title", "content": "Note Content", "createdAt": "timestamp", "updatedAt": "timestamp" }, ... ]

7. User Interface

7.1 Register Page

Fields: Email, Password

Actions: Register, Redirect to Login

7.2 Login Page

Fields: Email, Password

Actions: Login, Redirect to Register

7.3 Dashboard Page

Components: NoteForm, NoteList, NoteItem

Actions: Add Note, Edit Note, Delete Note, Search Notes

8. Development and Deployment

8.1 Development Environment

Frontend: Local development server using Next.js

Backend: Local development server using Express.js

Database: In-memory storage

8.2 Deployment

Frontend: Deploy to Vercel or Netlify

Backend: Deploy to Heroku or any other Node.js hosting service

Database: In-memory storage (for production, consider using a persistent database like MongoDB or PostgreSQL)