NoteNest - The Note-Taking App

Project Documentation

1. Introduction

1.1 Purpose

NoteNest is a modern note-taking application designed to help users efficiently store, manage, and retrieve their notes. With secure authentication, user-friendly design, and seamless accessibility across devices, NoteNest aims to enhance productivity for students, professionals, and individuals.

1.2 Objectives

- Provide a **secure** and **scalable** platform for note management.
- Enable CRUD operations (Create, Read, Update, Delete) on notes.
- Implement **user authentication** and **data encryption** for security.
- Ensure a responsive and **intuitive UI/UX** for ease of use.
- Offer future scalability with features like note sharing and reminders.

1.3 Target Audience

- **Students**: Organizing lecture notes, assignments, and study materials.
- **Professionals**: Managing work-related documents and meeting notes.
- **Personal Users**: Storing personal thoughts, journals, and to-do lists.

2. System Architecture

2.1 Overview

The NoteNest architecture follows the **MERN (MongoDB, Express.js, React.js, Node.js) stack**, ensuring a seamless flow between frontend and backend services.

2.2 Components

1. **Frontend (React.js)**:
    - React-based UI with Bootstrap for styling.
    - Responsive design for desktops and mobile devices.
    - Uses Axios to interact with backend APIs.
2. **Backend (Node.js & Express.js)**:
    - RESTful API endpoints for CRUD operations.
    - JWT authentication for secure access.
    - Error handling and validation mechanisms.
3. **Database (MongoDB & Mongoose)**:
    - Stores user information and notes.
    - Uses schema-based design with indexing for fast retrieval.

3. User Authentication Flow

- **Registration**: Users provide an email and password.
- **Password Encryption**: bcrypt hashes passwords before storing them in MongoDB.
- **Login**: Users authenticate via email and password.
- **JWT Token Generation**: Upon successful login, a JWT token is issued for session management.
- **Secure API Access**: All requests must include a valid JWT token.

4. API Endpoints Documentation

4.1 User Authentication APIs

Method	Endpoint	Description
POST	/api/auth/signup	Registers a new user
POST	/api/auth/login	Authenticates user and returns JWT
GET	/api/auth/user	Retrieves logged-in user details

4.2 Notes Management APIs

Method	Endpoint	Description
POST	/api/notes	Creates a new note
GET	/api/notes	Retrieves all notes for a user
PUT	/api/notes/:id	Updates a note by ID
DELETE	/api/notes/:id	Deletes a note by ID

5. Database Schema Design

5.1 User Schema

```
{
  "name": "String",
  "email": "String",
  "password": "String (hashed)",
  "createdAt": "Date"
}
```

5.2 Note Schema

```
{
  "title": "String",
  "content": "String",
  "user": "ObjectId (references User)",
  "tags": ["String"],
  "createdAt": "Date"
}
```

6. UI/UX Design Overview

6.1 User Interface

- Clean and minimal design using **React.js & Bootstrap**.
- Easy navigation for **creating, editing, deleting, and searching** notes.
- Mobile-responsive layout for cross-device compatibility.

6.2 User Experience Strategy

- **Simple Navigation**: Users can easily create and find notes.
- **Dark Mode Support**: Enhances usability in low-light conditions.
- **Search & Filter**: Quick access to relevant notes.

7. Testing & Deployment

7.1 Testing

- **Frontend Testing**: Performed using Jest & React Testing Library.
- **Backend Testing**: API tests conducted using Postman.
- **Security Testing**: Validates authentication and data privacy.

7.2 Deployment

- **Frontend**: Deployed on **Vercel** or **Netlify**.
- **Backend**: Hosted on **Heroku** or **AWS EC2**.
- **Database**: Deployed using **MongoDB Atlas**.

8. Security Measures

- **JWT Token Expiry**: Prevents session hijacking.
- **Password Hashing**: Ensures secure storage.
- **CORS Handling**: Prevents unauthorized API access.
- **Rate Limiting**: Protects against brute force attacks.

9. Challenges & Solutions

9.1 Challenge: Managing User Authentication

- **Solution**: Implemented JWT-based authentication with bcrypt hashing.

9.2 Challenge: Ensuring Scalable Data Handling

- **Solution**: Used indexed queries and optimized MongoDB schema design.

10. Future Enhancements

- **Note Sharing**: Share notes with specific users.
- **Categorization**: Enable folder-based organization.
- **Reminders & Notifications**: Alerts for upcoming tasks.
- **Offline Mode**: Access notes without an internet connection.
- **Advanced Search**: Implementing AI-based search for better retrieval.
- **Voice-to-Text Notes**: Enabling voice input for notes.

11. Step-by-Step Setup Guide

1. Clone the repository: git clone <repo_url>
2. Navigate to project folder: cd notenest
3. Install dependencies: npm install (for backend) and cd client && npm install (for frontend)
4. Configure environment variables: .env file setup for database and authentication secrets.
5. Start backend: npm start
6. Start frontend: cd client && npm start

12. API Flow & Diagrams

- **Flowcharts** illustrating user authentication and note management request-response cycle.
- **ER Diagram** for database relationships.

13. Troubleshooting & FAQs

- **Issue**: API request failing.
    - **Solution**: Check if the backend server is running.
- **Issue**: Login not working.
    - **Solution**: Ensure correct credentials and check JWT token validity.

14. Technology Comparison

- **Why MERN Stack?**
    - Full JavaScript stack simplifies development.
    - React for dynamic UI, Express for lightweight API handling.
    - MongoDB for flexible schema handling.

15. Conclusion

NoteNest provides a **secure, scalable, and efficient** solution for note-taking, catering to a diverse audience. Its modular design allows for future improvements and expansion.

16. References

- MongoDB Documentation: https://www.mongodb.com/docs/
- React.js Documentation: https://reactjs.org/docs/
- Express.js Guide: https://expressjs.com/en/guide/
- Node.js Security Best Practices: https://nodejs.org/en/docs/guides/security/