Project Name :NoteNest-The Note Taking App

Team ID : LTVIP2025TMID24714

Team Leader : P T Raghu

Team member : Bestha Surendra Babu

NoteNest - The Note Taking App

The NoteNest App back-end is designed to offer users a reliable and secure environment for managing their personal notes. It features user authentication with JWT-based token security, ensuring that each user's data remains private and protected. The API enables users to perform CRUD (Create, Read, Update, Delete) operations on their notes, allowing them to organize information effectively.

The backend is developed using Node.js and Express.js, leveraging MongoDB as the database to store user information and notes efficiently. Mongoose ODM is utilized to interact with MongoDB, simplifying data manipulation. Middleware such as express-validator is used for request validation, while bcryptjs ensures password encryption for secure authentication.

One of the core functionalities of the Notes App back-end is its authentication mechanism. Users must sign up and log in to access their notes. Each API request is validated using JWT authentication, ensuring that only authorized users can manage their data.

The Notes App back-end also includes error handling mechanisms to provide informative responses in case of invalid requests or server issues. This ensures a smooth user experience by preventing unauthorized access and handling common issues gracefully.

Designed to be modular and scalable, this back-end can be extended to include additional features such as note sharing, categorization, and reminders. It serves as an excellent foundation for developers who want to build a fully functional note-taking application.

Scenario Based Case Study

Background: Amit, a second-year Computer Science student, struggles with organizing his study materials. He often finds it difficult to keep track of important lecture notes, project ideas, and personal to-do lists. He has tried using traditional notebooks and various note-taking apps, but they either lack essential features or have cluttered interfaces. Amit is looking for a simple yet effective solution that allows him to store, manage, and retrieve his notes securely from any device.

Problem: During his semester exams, Amit realized that he had lost some important notes on a specific programming concept. His notes were scattered across different platforms—some written in physical notebooks, some stored in his mobile notes app, and others saved in multiple cloud drives. This inefficiency resulted in wasted time and stress, making his revision process difficult. Amit needed a centralized, easy-to-use platform where he could store, organize, and retrieve all his study materials efficiently.

Solution: Amit discovered the Notes App, a simple yet powerful web-based note-taking application with a secure back-end. After signing up, he was able to create, update, and organize his notes effortlessly. With user authentication and authorization, his data remained safe. The search and filter features helped him quickly find the required notes. Additionally, he could categorize his notes using tags like "Lecture Notes," "To-Do," and "Project Ideas," making organization seamless.

One day, Amit accidentally deleted an important note, but thanks to the automatic backup feature, he was able to recover it quickly. He also found that he could access his notes from multiple devices, allowing him to review his study material on the go.

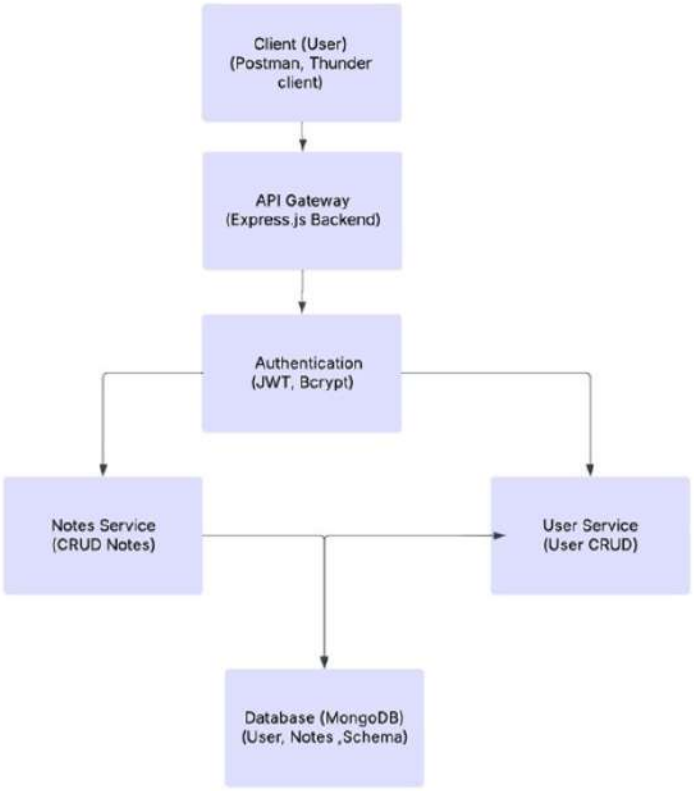
Usage:

- Student Usage: Amit uses the app to take notes on various subjects and organize them with relevant tags.
- Professional Usage: His senior, working as a software developer, uses the app to store project ideas, meeting notes, and technical documentation.
- Personal Usage: Amit's friend, Riya, uses it to maintain a daily journal, grocery lists, and travel plans.

Outcome:

With the Notes App, Amit has streamlined his study process. He no longer worries about losing important information, as all his notes are securely stored and easily accessible. The app's intuitive interface and useful features have enhanced his productivity, helping him focus more on learning and less on managing scattered notes.

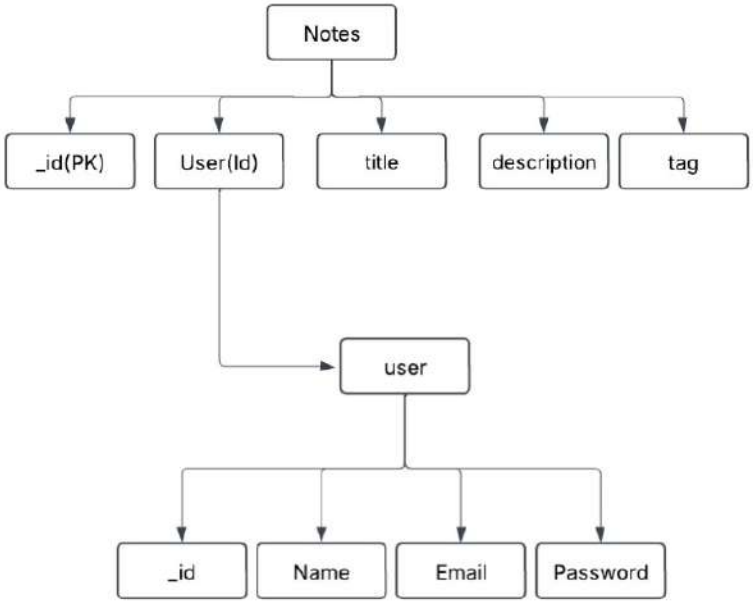
Technical Architecture:



Client (Frontend or API Testing Tools)

The user interacts with the application through a web or mobile front-end, or directly via

ER-Diagram



Key Features:

1. User Dashboard: A central dashboard where users can view, create, edit, and delete their notes in an organized manner.
2. Note Creation: Users can create and save notes with a title, description, and optional tags for easy categorization.
3. Edit & Update Notes: Existing notes can be updated with new content without creating a new entry.
4. Delete Notes: Users can remove notes that are no longer needed.
5. User Authentication: Secure login and signup system using JWT (JSON Web Token) authentication.
6. Secure Data Storage: All user notes are securely stored in a MongoDB database with unique user associations.
7. Tagging System: Notes can be categorized using tags for easier organization and filtering.
8. Search & Filter: Users can quickly find notes by searching based on title, content, or tags.
9. Date Tracking: Each note includes a timestamp to track when it was created or last updated.

PRE REQUISITES

To develop this Notes App using Node.js, Express.js, MongoDB, and JWT authentication, ensure you have the following prerequisites installed:

1. Node.js and npm

· Install Node.js, which includes npm (Node Package Manager), required for running JavaScript on the backend.

· Download: Node.js Official Website

· Installation Guide: Node.js Installation

"Npm init"

2. MongoDB

· Set up MongoDB to store notes securely.

· Install MongoDB locally or use a cloud-based service like MongoDB Atlas.

· Download: MongoDB Community Edition

· Installation Guide: MongoDB Installation Docs

3. Express.js

· Express.js is a web framework for Node.js, handling routing, middleware, and API development.

Install Express.js via npm:

"npm install express"

4. Mongoose (MongoDB ODM)

· Mongoose helps interact with MongoDB efficiently.

· Install Mongoose:

"npm install mongoose"

5. JSON Web Token (JWT)

· JWT is used for user authentication.

· Install JWT package:

"npm install jsonwebtoken"

6. bcrypt.js (Password Encryption)

· Used to securely hash user passwords.

· Install bcrypt:

"npm install bcryptjs"

7.Postman or Thunder Client(Vs extension)

Purpose: A comprehensive API platform for designing, testing, and documenting APIs.

Download: Access the latest version suitable for your operating system from the Postman Downloads page.

Installation Guide: Follow the instructions provided in the Postman Installation Guide to set up the application on your system

Roles and Responsibility

· Develop and maintain the server-side logic using Node.js and Express.js.

· Implement user authentication and authorization using JWT and bcrypt.js.

· Design and manage the MongoDB database schema using Mongoose.

· Create RESTful APIs for handling CRUD operations (Create, Read, Update, Delete) for notes.

· Ensure data security and user access control mechanisms are in place.

· Optimize database queries to improve performance.

· Implement middleware for logging, request validation, and error handling.

· Ensure API documentation is maintained for frontend developers.

Project Setup and Configuration

1. Install Required Tools and Software

Ensure you have the following installed on your system:

Node.js (Server-side JavaScript runtime)

Download: https://nodejs.org/en/download/

 MongoDB (Database)

Download: https://www.mongodb.com/try/download/community

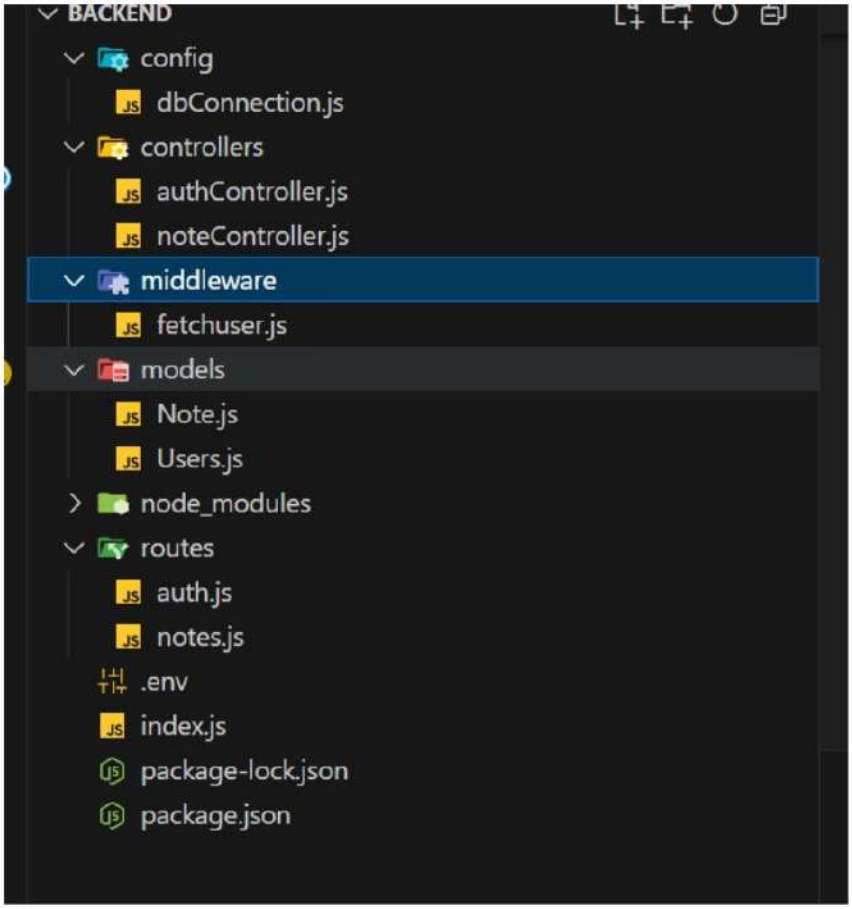
2. Create Project Folders and Files

/notes-app-backend

??? /models # Database models (schemas)

? ??? User.js # User schema

```
?   ??? Note.js        # Note schema
??? /routes          # API route files
? ? ??? authRoutes.js # Routes for authentication (register, login)
ers ??? noteRoutes.js # Routes for CRUD operations on notes
                      # Business logic for handling requests
? ? ??? authController.js # Handles authentication-related logic
are??? noteController.js # Handles notes-related logic
                        # Middleware functions for authentication & error handling
? ??? authMiddleware.js # Middleware to protect routes (JWT authentication)
? ??? errorMiddleware.js # Centralized error handling
??? /config          # Configuration files (DB connection, environment variables)
? ??? db.js           # MongoDB connection setup
??? /public          # Static files (if needed)
??? .env             # Environment variables (MongoDB URI, JWT secret)
??? index.js         # Main entry point of the backend
??? package.json     # Project dependencies
```



3. Install Required npm Packages

Backend npm Packages

Run the following command inside your project directory:

"npm install express mongoose cors dotenv bcryptjs jsonwebtoken express-validator nodemon"

- Express - Web framework for Node.js
- Mongoose - ODM for MongoDB
- Dotenv - For managing environment variables
- Bcrypt.js - For password encryption
- Jsonwebtoken (JWT) - For authentication
- Express-validator - Middleware for request validation

Backend Development

1. Setup Express Server
   - Create an index.js file in the root folder.
   - Configure the server with necessary middleware (cors, body-parser, express.json()).

- Start the server using nodemon.

Example index.js Code:

```
const PORT = 8000;
const app = express();
app.use(express.json());
app.use(cors());
const server = http.createServer(app);

const io = new Server(server, {
  cors: {
    origin: "http://localhost:3000",
    methods: ["GET", "POST"],
  },
});
```

2. User Authentication

- Implement routes and middleware for user registration, login, and authentication.

- Protect sensitive routes using JWT-based authentication.

```
const jwt = require('jsonwebtoken');
const dotenv = require('dotenv');
// Load environment variables from .env file
dotenv.config();
const jwt_secret = process.env.JWT_SECRET_KEY;
fetchuser = (req, res,next) => {
    // get token from header 'auth-token'
    const token = req.header('auth-token');
    // if token is invalid
    if (!token){
        // if token is invalid
        return res.status(401).send({error: "Please authenticate using a valid token"});
    }

    try {
        // verify received token
        const data = jwt.verify(token, jwt_secret);
        // add user id to req object
        req.user = data.user;
        // call next middleware
        next();
    }
    catch (error){
        // if token is invalid
        res.status(401).send({error: "Please authenticate using a valid token"});
    }
}

module.exports = fetchuser;
```

3. Define API Routes

- Create separate files inside the /routes folder for different functionalities:
- authRoutes.js (User Authentication)
- noteRoutes.js (CRUD operations for notes)

routes > auth.js > ...

```
const express = require('express');
const { body } = require('express-validator');
const fetchuser = require('../middleware/fetchuser');
const { createUser, login, getUser, deleteAccount, updatePassword } = require('../controllers

const router = express.Router();
// User Registration
router.post('/createuser', [
    body('name', 'Enter a valid name').isLength({ min: 3 }),
    body('email', 'Enter a valid email').isEmail(),
    body('password', 'Password must be at least 8 characters').isLength({ min: 8 })
], createUser);
// User Login
router.post('/login', [
    body('email', 'Invalid email!').isEmail(),
    body('password', 'Enter a valid password').exists()
], login);
// Get User Data
router.post('/getuser', fetchuser, getUser);
// Delete Account
router.delete('/deleteaccount', fetchuser, deleteAccount);
// Update Password
router.put('/updatepassword', fetchuser, [
    body('oldpassword', 'Enter your old password').exists(),
    body('newpassword', 'New password must be at least 8 characters').isLength({ min: 8 })
], updatePassword);

module.exports = router;
```

4. Implement Error Handling

- Use a centralized error-handling mechanism.
- Return appropriate error responses with HTTP status codes.

Database

Mongdb Connection and Model creation:Demo link

1. Install and Configure MongoDB

- Install Mongoose (npm install mongoose).
- Create a database connection file inside the /config folder.

Database Connection File (config/db.js)

```
const mongoose = require('mongoose');
const dotenv = require('dotenv');

// Load environment variables
dotenv.config();

// Get MongoDB URI from environment variables
const mongooseURI = process.env.MONGO_URI ;

// MongoDB connection
const dbConnect = async () => {
    try {
        await mongoose.connect(mongooseURI, {
            useNewUrlParser: true,
            useUnifiedTopology: true,
        });
        console.log(" MongoDB connected successfully!");
    } catch (error) {
        console.error(" MongoDB connection error:", error.message);
        process.exit(1);
    }
};

module.exports = dbConnect;

```

Create a .env file in the root folder and add the MongoDB connection string:

```
.env
PORT = 5000
JWT_SECRET_KEY = "YOUR_SECRET_KEY"
MONGO_URI= "Paste your mongodb url here"
```

2. Define Database Models

Inside /models, create User.js and Note.js.

User_Model:

```
const mongoose = require('mongoose');

//  schema for users model
const userSchema = new mongoose.Schema({
    name:{
        type: String,
        required: true
    },
    email:{
        type: String,
        required: true,
        unique: true
    },
    password:{
        type: String,
        required: true
    },
    date:{
        type: Date,
        default: Date.now
    }
});

module.exports = mongoose.model('user',userSchema);
```

Note_Model:

```
const mongoose = require('mongoose');

//  schema for notes model
const noteSchema = new mongoose.Schema({
    user:{
        type: mongoose.Schema.Types.ObjectId,
        ref: 'user',
        required: true
    },
    title:{
        type: String,
        required: true
    },
    description:{
        type: String,
        required: true
    },
    tag:{
        type: String,
        default: "General"
    },
    date:{
        type: Date,
        default: Date.now
    }
});

module.exports = mongoose.model('Note',noteSchema);
```

3. Connect Database to Backend

Modify index.js to ensure database is connected before starting the server.

4. Start the Server

Run the following command:

"npm start"