NoteNest

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:**

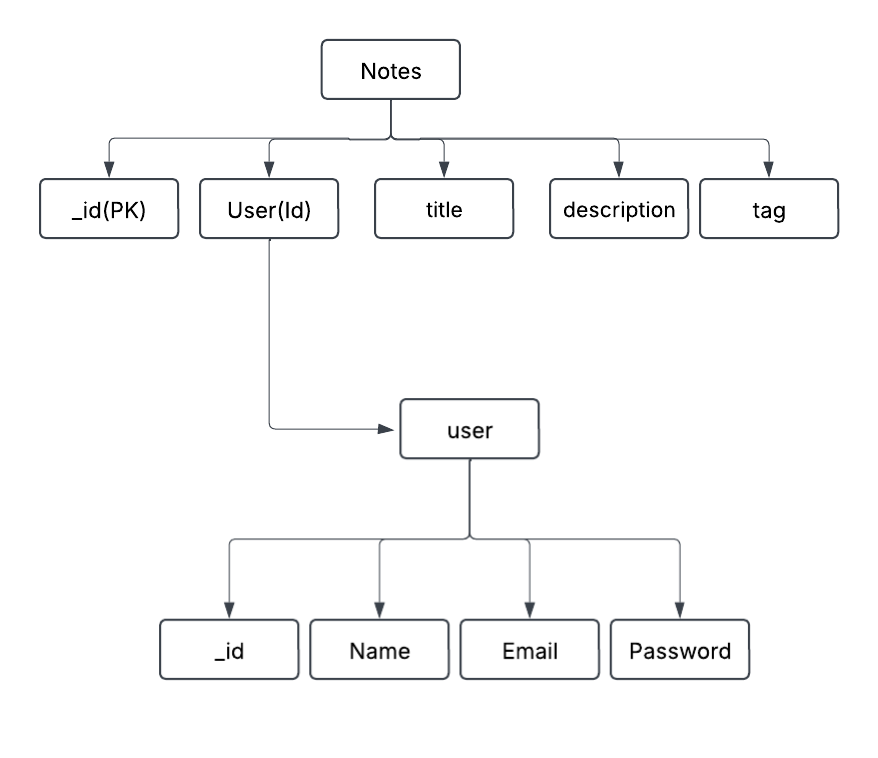
**With the Notes App, Amit has streamlined his study process. He no longer worries about losing important information, as all his notes are securely stored and easily accessible. The app's intuitive interface and useful features have enhanced his productivity, helping him focus more on learning and less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

### 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)

?    ??? noteRoutes.js   # Routes for CRUD operations on notes

??? /controllers     # Business logic for handling requests

?    ??? authController.js # Handles authentication-related logic

?    ??? noteController.js # Handles notes-related logic

??? /middleware      # 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

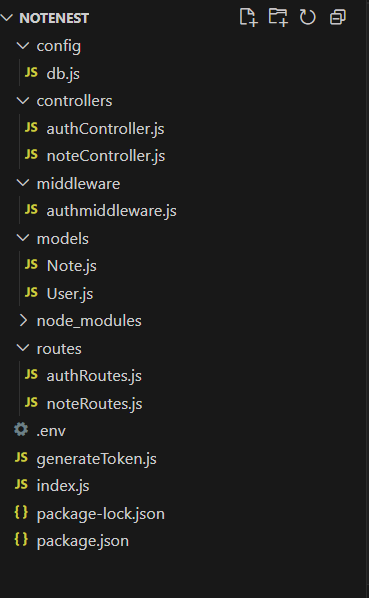
??? /public          # Static files (if needed)

??? .env             # Environment variables (MongoDB URI, JWT secret)

??? index.js         # Main entry point of the backend

generateToken.js #For Token generation

??? package.json     # Project dependencies



**3. Install Required npm Packages**

**Backend npm Packages**

Run the following command inside your project directory:

“**npm install express mongoose corsdotenvbcryptjsjsonwebtoken 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 validatio

Following is the project code….

1) db.js:-

const mongoose = require('mongoose');

constconnectDB = async () => {

    try {

        await mongoose.connect(process.env.MONGO\_URI);

        console.log('MongoDB connected');

    } catch (error) {

        console.error(error.message);

        process.exit(1);

    }

};

module.exports = connectDB;

2) authController.js:-

const User = require('../models/User');

constjwt = require('jsonwebtoken');

constbcrypt = require('bcryptjs');

const{ validationResult } = require('express-validator');

exports.registerUser = async (req, res) => {

    const errors = validationResult(req);

    if (!errors.isEmpty()) return res.status(400).json({ errors: errors.array() });

    const{ name, email, password } = req.body;

    try {

        const user = new User({ name, email, password });

        await user.save();

        res.status(201).json({ message: 'User registered successfully!' });

    } catch (error) {

        res.status(500).json({ error: error.message });

    }

};

exports.loginUser = async (req, res) => {

    const errors = validationResult(req);

    if (!errors.isEmpty()) return res.status(400).json({ errors: errors.array() });

    const{ email, password } = req.body;

    try {

        const user = await User.findOne({ email });

        if (!user || !(await bcrypt.compare(password, user.password))) {

            return res.status(401).json({ message: 'Invalid credentials' });

        }

        const token = jwt.sign({ id: user.\_id }, process.env.JWT\_SECRET, { expiresIn: '1h' });

        res.status(200).json({ token });

    } catch (error) {

        res.status(500).json({ error: error.message });

    }

};

exports.getUserDetails = async (req, res) => {

    try {

        const user = await User.findById(req.user).select('-password'); // Exclude password

        if (!user) {

            return res.status(404).json({ message: 'User not found' });

        }

        res.json(user);

    } catch (err) {

        console.error(err);

        res.status(500).json({ message: 'Server error' });

    }

};

exports.updatePassword = async (req, res) => {

    const errors = validationResult(req);

    if (!errors.isEmpty()) {

        return res.status(400).json({ errors: errors.array() });

    }

    const{ currentPassword, newPassword } = req.body;

    try {

        const user = await User.findById(req.user);

        if (!user) {

            return res.status(404).json({ message: 'User not found' });

        }

        constisMatch = await bcrypt.compare(currentPassword, user.password);

        if (!isMatch) {

            return res.status(400).json({ message: 'Current password is incorrect' });

        }

        const salt = await bcrypt.genSalt(10);

        user.password = await bcrypt.hash(newPassword, salt);

        await user.save();

        res.json({ message: 'Password updated successfully' });

    } catch (err) {

        console.error(err);

        res.status(500).json({ message: 'Server error' });

    }

};

exports.deleteAccount = async (req, res) => {

    try {

        const user = await User.findByIdAndDelete(req.user);

        if (!user) {

            return res.status(404).json({ message: 'User not found' });

        }

        res.json({ message: 'Account deleted successfully' });

    } catch (err) {

        console.error(err);

        res.status(500).json({ message: 'Server error' });

    }

};

3) noteController.js:-

const Note = require('../models/Note');

const{ validationResult } = require('express-validator');

exports.getNotes = async (req, res) => {

    const notes = await Note.find({ user: req.user.id });

    res.json(notes);

};

exports.createNote = async (req, res) => {

    const errors = validationResult(req);

    if (!errors.isEmpty()) return res.status(400).json({ errors: errors.array() });

    const{ title, content } = req.body;

    const note = new Note({

        user: req.user.id,

        title,

        content,

    });

    await note.save();

    res.status(201).json(note);

};

exports.updateNote = async (req, res) => {

    const errors = validationResult(req);

    if (!errors.isEmpty()) return res.status(400).json({ errors: errors.array() });

    const note = await Note.findOneAndUpdate(

        { \_id: req.params.id, user: req.user.id },

        req.body,

        { new: true }

    );

    res.json(note);

};

exports.deleteNote = async (req, res) => {

    await Note.findOneAndDelete({ \_id: req.params.id, user: req.user.id });

    res.json({ message: 'Note deleted' });

};

4) authmiddleware.js:-

constjwt = require('jsonwebtoken');

const auth = (req, res, next) => {

    constauthHeader = req.header('Authorization');

    if (!authHeader || !authHeader.startsWith('Bearer ')) {

        console.log('Authorization header missing or invalid:', authHeader); // Debugging log

        return res.status(401).json({ message: 'No token, authorization denied' });

    }

    const token = authHeader.split(' ')[1];

    console.log('Token received:', token); // Debugging log

    try {

        const decoded = jwt.verify(token, process.env.JWT\_SECRET);

        console.log('Decoded Payload:', decoded); // Debugging log

        req.user = decoded.id;

        next();

    } catch (err) {

        console.error('JWT Verification Error:', err.message); // Debugging log

        res.status(401).json({ message: 'Token is not valid', error: err.message });

    }

};

module.exports = auth;

5) Note.js:-

const mongoose = require('mongoose');

constNoteSchema = new mongoose.Schema({

    user: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },

    title: { type: String, required: true },

    description: { type: String},

    tag:{type: String, default: "General"},

    date: {type: Date, default: Date.now}

}, { timestamps: true });

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

6) User.js:-

const mongoose = require('mongoose');

constbcrypt = require('bcryptjs');

constUserSchema = 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}

}, { timestamps: true });

// Hash password before saving

UserSchema.pre('save', async function (next) {

    if (!this.isModified('password')) return next();

    const salt = await bcrypt.genSalt(10);

    this.password = await bcrypt.hash(this.password, salt);

    next();

});

module.exports = mongoose.model('User', UserSchema);

7) authRoutes.js:-

const express = require('express');

const{ registerUser, loginUser,getUserDetails, updatePassword, deleteAccount } = require('../controllers/authController');

const{ body } = require('express-validator');

const router = express.Router();

constauthMiddleware = require('../middleware/authMiddleware');

router.post('/createuser', [

    body('name').notEmpty().withMessage('Name is required'),

    body('email').isEmail().withMessage('Please include a valid email'),

    body('password').isLength({ min: 6 }).withMessage('Password must be at least 6 characters'),

], registerUser);

router.post('/login', [

    body('email').isEmail().withMessage('Please include a valid email'),

    body('password').notEmpty().withMessage('Password is required'),

], loginUser);

router.post('/getuser', authMiddleware, getUserDetails);

// Route to update password

router.put('/update-password', authMiddleware, [

    body('currentPassword').notEmpty().withMessage('Current password is required'),

    body('newPassword').isLength({ min: 6 }).withMessage('New password must be at least 6 characters'),

], updatePassword);

// Route to delete account

router.delete('/delete-account', authMiddleware, deleteAccount);

module.exports = router;

8) noteController.js:-

const express = require('express');

const{ createNote, getNotes, updateNote, deleteNote } = require('../controllers/noteController');

const{ body } = require('express-validator');

constauthMiddleware = require('../middleware/authMiddleware');

const router = express.Router();

router.get('/fetchnotes', authMiddleware, getNotes);

router.post('/addnote', authMiddleware, [

    body('title').notEmpty().withMessage('Title is required')

], createNote);

router.put('/updatenote/:id', authMiddleware, [

    body('title').notEmpty().withMessage('Title is required')

], updateNote);

router.delete('/deletenote/:id', authMiddleware, deleteNote);

module.exports = router;

9) .env:-

PORT=5000

MONGO\_URI=mongodb+srv://reenashaik02:Reena593@cluster0.c8bkb.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0

JWT\_SECRET=862d91fbb60e12777b84bc071deec30264dfd7f15a7517094f4c846d944fd1d1a7af5ad9a28488f2e5224ab0bd3e1825426d6c91bc664b575fdd83f31612a553

10) generateToken.js:-

require('dotenv').config(); // Load environment variables

constjwt = require('jsonwebtoken');

const payload = {

    id: "reenashaik02", // Replace with an actual user ID from your database

};

const token = jwt.sign(payload, process.env.JWT\_SECRET, { expiresIn: '1h' });

console.log('Generated Token:', token);

11) index.js:-

const express = require('express');

constcors = require('cors');

constconnectDB = require('./config/db');

constauthRoutes = require('./routes/authRoutes');

constnoteRoutes = require('./routes/noteRoutes');

require('dotenv').config();

const app = express();

// Middleware

app.use(cors());

app.use(express.json());

// Routes

app.use('/api/auth', authRoutes);

app.use('/api/notes', noteRoutes);

// Connect Database

connectDB();

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => {

    console.log(`Server running on http://localhost:${PORT}`);

});

﻿

POST

http://localhost:5000/api/auth/createuser

http://localhost:5000/api/auth/createuser

The POST /api/auth/createuser endpoint allows clients to register a new user by submitting their details to the server.

﻿

Body

raw (json)

json

{

"name":"mahendra Kari",

"email": "xyz@gmail.com",

"password": "00000000"

}

POST

http://localhost:5000/api/auth/login

http://localhost:5000/api/auth/login

The POST /api/auth/login endpoint allows users to authenticate by submitting their credentials, typically returning a token upon successful login.

﻿

Body

raw (json)

json

{

"email": "xyz@gmail.com",

"password": "00000000"

}

POST

http://localhost:5000/api/auth/getuser

http://localhost:5000/api/auth/getuser

The POST /api/auth/getuser endpoint retrieves the authenticated user's details, typically requiring a valid authentication token in the request headers.

﻿

Request Headers

Authorization

Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY3YzcwYzE1MmNiNjNiMThhMDE1ZTFjZCIsImlhdCI6MTc0MTA5ODEyMywiZXhwIjoxNzQxMTAxNzIzfQ.XMGChoyTXPbDieF0eevQI8bHKfrNgjFraye8LWbHIc4

PUT

http://localhost:5000/api/auth/update-password

http://localhost:5000/api/auth/update-password

The PUT /api/auth/update-password endpoint allows authenticated users to securely change their current password by providing their existing password and a new password.

﻿

Request Headers

Authorization

Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY3YzcwYzE1MmNiNjNiMThhMDE1ZTFjZCIsImlhdCI6MTc0MTA5ODEyMywiZXhwIjoxNzQxMTAxNzIzfQ.XMGChoyTXPbDieF0eevQI8bHKfrNgjFraye8LWbHIc4

Body

raw (json)

json

{

"currentPassword":"00000000",

"newPassword":"123456789"

}

POST

http://localhost:5000/api/notes/addnote

http://localhost:5000/api/notes/addnote

The POST /api/notes/addnote endpoint allows authenticated users to create a new note by submitting its content to the server.

﻿

Request Headers

Authorization

Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY3YzcwYzE1MmNiNjNiMThhMDE1ZTFjZCIsImlhdCI6MTc0MTA5ODEyMywiZXhwIjoxNzQxMTAxNzIzfQ.XMGChoyTXPbDieF0eevQI8bHKfrNgjFraye8LWbHIc4

Body

raw (json)

json

{

"title":"my first by the user",

"content": "content is mandatory",

"tag": "work"

}

GET

http://localhost:5000/api/notes/fetchnotes

http://localhost:5000/api/notes/fetchnotes

The GET /api/notes/fetchnotes endpoint allows authenticated users to retrieve all their saved notes from the server.

﻿

Request Headers

Authorization

Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY3YzcwYzE1MmNiNjNiMThhMDE1ZTFjZCIsImlhdCI6MTc0MTA5ODEyMywiZXhwIjoxNzQxMTAxNzIzfQ.XMGChoyTXPbDieF0eevQI8bHKfrNgjFraye8LWbHIc4

PUT

http://localhost:5000/api/notes/updatenote/67c70fba7329d265dd752144

http://localhost:5000/api/notes/updatenote/67c70fba7329d265dd752144

The PUT /api/notes/updatenote/:id endpoint allows authenticated users to modify an existing note identified by its unique ID.

﻿

Request Headers

Authorization

Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY3YzcwYzE1MmNiNjNiMThhMDE1ZTFjZCIsImlhdCI6MTc0MTA5ODEyMywiZXhwIjoxNzQxMTAxNzIzfQ.XMGChoyTXPbDieF0eevQI8bHKfrNgjFraye8LWbHIc4

Body

raw (json)

json

{

"title":"updated note",

"description": "updated description",

"tag": "updated tag"

}

DELETE

http://localhost:5000/api/notes/deletenote/67c70fba7329d265dd752144

http://localhost:5000/api/notes/deletenote/67c70fba7329d265dd752144

The DELETE /api/notes/deletenote/:id endpoint allows authenticated users to remove a specific note identified by its unique ID.

﻿

Request Headers

Authorization

Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY3YzcwYzE1MmNiNjNiMThhMDE1ZTFjZCIsImlhdCI6MTc0MTA5ODEyMywiZXhwIjoxNzQxMTAxNzIzfQ.XMGChoyTXPbDieF0eevQI8bHKfrNgjFraye8LWbHIc4

DELETE

http://localhost:5000/api/auth/delete-account

http://localhost:5000/api/auth/delete-account

The DELETE /api/auth/deleteaccount endpoint allows authenticated users to permanently delete their account from the Database.

﻿

Request Headers

Authorization

Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY3YzcwYzE1M