**Medicine Store Backend API Documentation**

**Overview**

This documentation provides a detailed guide to the backend of the Medicine Store System, built using Node.js, Express.js, and MongoDB. The backend is designed to manage pharmacy operations, including user management, medicine inventory, and order processing.

**Table of Contents**

* [Overview](#overview)
* [Architecture](#architecture)
* [Technologies Used](#technologies-used)
* [Setup and Installation](#setup-and-installation)
* [Environment Variables](#environment-variables)
* [Folder Structure](#folder-structure)
* [API Endpoints](#api-endpoints)
  + [User Endpoints](#user-endpoints)
  + [Medicine Endpoints](#medicine-endpoints)
  + [Order Endpoints](#order-endpoints)
* [Middleware](#middleware)
* [Authentication](#authentication)
* [Error Handling](#error-handling)
* [Testing](#testing)
* [Deployment](#deployment)
* [Future Enhancements](#future-enhancements)
* [References](#references)

**Architecture**

The project is designed with a modular structure to ensure scalability and maintainability. The key components include:

* **Controllers:** Contain the business logic for handling requests and interfacing with the database.
* **Models:** Define MongoDB schemas using Mongoose for various entities like Users, Medicines, and Orders.
* **Routes:** Map HTTP endpoints to corresponding controller functions.
* **Middleware:** Include functionalities for authentication (JWT), error handling, and request validation.
* **Helpers:** Utility functions that support common tasks throughout the project.

**Technologies Used**

* **Node.js** – JavaScript runtime environment
* **Express.js** – Web framework for building APIs
* **MongoDB** – NoSQL database for data storage
* **Mongoose** – ODM for MongoDB
* **JWT (jsonwebtoken)** – For authentication
* **bcryptjs** – For password hashing
* **dotenv** – For environment variable management
* **cors** – To enable Cross-Origin Resource Sharing
* **nodemon** – Development tool for auto-restarting the server

**Setup and Installation**

**1. Clone the Repository**

git clone https://github.com/tanmayMt/Medicine-Store-Project-Using-MERN.git

cd Medicine-Store-Project-Using-MERN

**2. Install Dependencies**

npm install

**3. Create the Environment File**

Create a .env file in the root directory and add the following variables:

.env

PORT=5000

MONGO\_URI=your\_mongodb\_connection\_string

JWT\_SECRET=your\_jwt\_secret

**4. Start the Server**

For development, use nodemon:

npm run dev

Your server should now be running at **http://localhost:5000**.

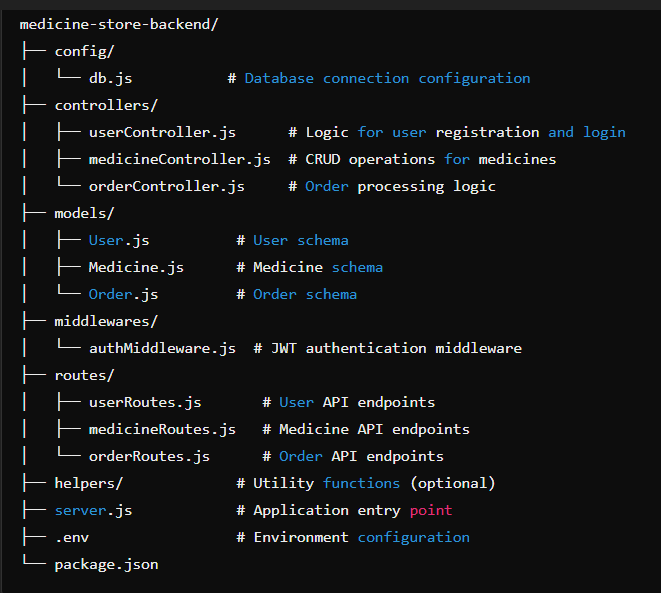
**Environment Variables**

The application uses the following environment variables:

* **PORT:** The port number the server listens on.
* **MONGO\_URI:** MongoDB connection string.
* **JWT\_SECRET:** Secret key for signing JWT tokens.

**Folder Structure**

A typical structure for the backend project:



**API Endpoints**

**User Endpoints**

**POST /api/users/register**

* **Description:** Register a new user.
* **Request Body:**

json

CopyEdit

{

"name": "John Doe",

"email": "john@example.com",

"password": "password123"

}

* **Response:** Success message or error details.

**POST /api/users/login**

* **Description:** Authenticate user and return a JWT token.
* **Request Body:**

json

CopyEdit

{

"email": "john@example.com",

"password": "password123"

}

* **Response:** A JWT token along with user information.

**Medicine Endpoints**

**POST /api/medicines/**

* **Description:** Add a new medicine record.
* **Request Body:**

json

CopyEdit

{

"name": "Paracetamol",

"price": 20,

"stock": 100,

"expiryDate": "2025-12-31"

}

* **Response:** Newly created medicine object.
* **Authentication:** Protected (typically admin-only).

**GET /api/medicines/**

* **Description:** Retrieve all medicine records.
* **Response:** Array of medicine objects.

**PUT /api/medicines/:id**

* **Description:** Update an existing medicine record.
* **Parameters:** id (Medicine ID) in the URL.
* **Request Body:** Fields to update.
* **Response:** Updated medicine object.

**DELETE /api/medicines/:id**

* **Description:** Delete a medicine record.
* **Parameters:** id (Medicine ID) in the URL.
* **Response:** Success message.

**Order Endpoints**

**POST /api/orders/**

* **Description:** Create a new order.
* **Request Body:**

json

CopyEdit

{

"userId": "user\_id",

"items": [

{ "medicineId": "medicine\_id", "quantity": 2 }

],

"totalPrice": 40,

"orderDate": "2025-02-19"

}

* **Response:** Confirmation and order details.
* **Authentication:** Protected (user-specific or admin-only).

**GET /api/orders/**

* **Description:** Retrieve orders.
* **Response:** List of orders.
* **Authentication:** Protected.

**Middleware**

**Authentication Middleware**

* **Purpose:** Protect routes by verifying JWT tokens.
* **Usage:** Include in routes that require authenticated access.
* **Example Implementation:**

js

CopyEdit

const jwt = require("jsonwebtoken");

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

const token = req.header("Authorization")?.split(" ")[1];

if (!token) return res.status(401).json({ message: "No token provided" });

try {

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

req.user = decoded;

next();

} catch (error) {

res.status(401).json({ message: "Token is not valid" });

}

};

module.exports = authMiddleware;

**Error Handling Middleware**

* **Purpose:** Centralize error responses across the API.
* **Usage:** Use a custom middleware at the end of your middleware stack to catch errors.

**Authentication**

* **JWT Authentication:**  
  When users log in, a JWT token is generated. This token should be sent in the Authorization header (e.g., Bearer token) for protected endpoints.
* **Password Hashing:**  
  Passwords are hashed using bcryptjs before being stored in the database.

**Error Handling**

* **Standard Error Format:**  
  All errors are returned in a JSON object with a message and a status code.
* **Centralized Middleware:**  
  Errors caught in controllers are forwarded to a global error handler for a consistent response.

**Testing**

* **Using Postman:**  
  Test each API endpoint by sending HTTP requests to ensure correct functionality.
* **Automated Testing:**  
  Optionally, implement tests using Jest or Mocha to cover API endpoints and business logic.

**Deployment**

* **Environment Setup:**  
  Make sure all environment variables are correctly set up on the hosting platform.
* **Hosting Options:**  
  Deploy using platforms like Heroku, Vercel, or Render.
* **CI/CD Integration:**  
  Consider integrating continuous deployment for smoother updates.

**Future Enhancements**

* **Role-Based Access Control:**  
  Enhance the authentication system to support multiple user roles (e.g., admin, pharmacist, customer).
* **File Uploads:**  
  Implement features for uploading images or documents related to medicines.
* **Advanced Reporting:**  
  Integrate analytics to provide sales and inventory reports.
* **Scalability:**  
  Refactor components for better performance and scalability as user load increases.

**References**

* [Express.js Documentation](https://expressjs.com/)
* [MongoDB Documentation](https://docs.mongodb.com/)
* [Mongoose Documentation](https://mongoosejs.com/)
* [JWT Documentation](https://jwt.io/)
* [bcryptjs Documentation](https://www.npmjs.com/package/bcryptjs)

1:28:10