# File Structure :

#### **1. Root Directory**

* **config/**
  + **db.js**: Contains the MongoDB connection logic.
  + **index.js**: Manages configuration settings for different environments (dev, prod, etc.).

#### **2. controllers/**

* **auth.controller.js**: Handles user authentication, including registration, login, logout, and JWT token management.
* **availability.controller.js**: Manages mentor availability, including adding, updating, and fetching availability slots.
* **booking.controller.js**: Manages user bookings such as creating, updating, canceling, and viewing booking details.
* **mentor.controller.js**: Handles mentor-related actions like profile management, retrieving mentor details, etc.
* **service.controller.js**: Manages various services offered by mentors.
* **user.controller.js**: Manages user data and profile-related actions.
* **webhook.controller.js**: Handles webhook events, likely for third-party service integrations (e.g., Razorpay).

#### **3. helper/**

* **apiError.js**: Utility class for creating consistent API error responses.
* **asyncHandler.js**: Simplifies the handling of async functions in routes by catching errors and passing them to the error handler middleware.

#### **4. middleware/**

* **auth.js**: Middleware for checking if the user is authenticated and for verifying JWT tokens.
* **error.js**: Global error handling middleware for catching and responding to application errors.
* **validate.js**: Middleware for validating incoming request bodies based on defined schemas.

#### **5. models/**

* **availability.model.js**: MongoDB schema for managing mentor availability.
* **booking.model.js**: Schema defining bookings (e.g., user, mentor, time slots, etc.).
* **service.model.js**: Schema defining services provided by mentors.
* **user.model.js**: User schema for storing user data (e.g., name, email, password, role).

#### **6. routes/v1/**

* **auth.route.js**: Routes for authentication (e.g., register, login, logout).
* **availability.route.js**: Routes for handling availability (e.g., adding, editing, fetching).
* **booking.route.js**: Routes for booking-related operations.
* **home.route.js**: Likely a home or status check route.
* **mentor.route.js**: Routes for mentor management.
* **service.route.js**: Routes related to the services offered by mentors.
* **user.route.js**: Routes for user profile management.
* **webhook.route.js**: Routes for handling webhook events, likely linked to payment gateway interactions.

#### **7. services/**

* **auth.service.js**: Contains business logic related to authentication, such as user registration and login.
* **availability.service.js**: Handles mentor availability-related business logic.
* **booking.service.js**: Manages booking-related logic, such as creating, updating, and deleting bookings.
* **email.service.js**: Manages sending emails, possibly through Nodemailer (e.g., confirmation emails, alerts).
* **mentor.service.js**: Business logic for managing mentors, including profile updates and retrieving mentor details.
* **service.service.js**: Logic for managing services offered by mentors.
* **token.service.js**: Responsible for generating, validating, and refreshing JWT tokens.
* **user.service.js**: Contains logic related to user profile and data management.
* **zoom.service.js**: Likely handles Zoom integration, perhaps for scheduling and managing virtual meetings between mentors and mentees.

#### **8. template/**

* **confirmation.ejs**: Email template for confirmation emails sent to users, likely for verifying email addresses or confirming bookings.

#### **9. util/**

* **httpStatus.js**: Contains HTTP status codes and their descriptions for consistent responses across the app.

#### **10. validations/**

* **auth.validation.js**: Joi validation schemas for validating authentication requests (e.g., login, registration).
* **availability.validation.js**: Validation for managing mentor availability.
* **booking.validation.js**: Validation for booking-related operations.
* **service.validation.js**: Validation for mentor services.
* **user.validation.js**: Validation schemas for user-related operations (e.g., updating profile).

#### **11. Other files**

* **.env.example**: Example environment file listing necessary environment variables (e.g., database connection, JWT secret, API keys).
* **.gitignore**: Specifies files and directories to be ignored by Git, such as node\_modules/ and environment files.
* **app.js**: Entry point for setting up the Express application, configuring middleware, setting up routes, and error handling.
* **index.js**: Main file to initialize the server, connect to MongoDB, and listen on a specified port.
* **package.json**: Contains project metadata and a list of dependencies and scripts.
* **package-lock.json**: Automatically generated file that records the exact version of each installed package to ensure consistent installations.

#### **1. Project Overview**

MentorHub is an online platform designed to connect mentors with mentees. The backend is built using **Node.js**, with **Express** as the web framework and **MongoDB** as the database. The backend manages functionalities such as authentication, booking of mentors, scheduling, and payments through Razorpay.

The main components of the backend include:

* **User authentication is done** using JWT and bcrypt for password hashing.
* **Booking Management** for scheduling mentor sessions.
* **Payment Integration** using Razorpay.
* **Mentor Availability Management** for setting availability.
* **API Validations** using Joi for schema validation.
* **Email notifications** with Nodemailer for booking confirmations.

#### **2. Project Structure**

The file structure for MentorHub's backend follows a modular approach. Here's a breakdown:

##### **Root Directory**

* **config/**
  + db.js: Contains the MongoDB connection logic using Mongoose.
  + index.js: Central configuration file for the application, loading environment variables and connecting to services.
* **controllers/**
  + Handles business logic for different services like authentication, user management, bookings, etc.
  + Files include:
    - auth.controller.js: Manages authentication (login, registration).
    - availability.controller.js: Handles mentor availability.
    - booking.controller.js: Manages bookings between users and mentors.
    - mentor.controller.js: Handles mentor-related actions.
    - service.controller.js: Provides general service endpoints.
    - user.controller.js: Manages user data and operations.
    - webhook.controller.js: Handles webhooks (like for payment confirmations).
* **helper/**
  + Utility functions to handle API errors and asynchronous handler methods.
  + Files:
    - apiError.js: Custom error handling for API responses.
    - asyncHandler.js: Simplifies error handling in asynchronous functions.
* **middleware/**
  + Contains middleware used in the application such as authentication checks and validation.
  + Files:
    - auth.js: Authentication middleware to verify JWT tokens.
    - error.js: Global error handling middleware.
    - validate.js: Middleware for request validation using Joi.
* **models/**
  + Defines the Mongoose schemas for different entities in the application.
  + Files:
    - availability.model.js: Mentor availability schema.
    - booking.model.js: Booking schema.
    - service.model.js: General services schema.
    - user.model.js: User schema for both mentors and mentees.
* **routes/v1/**
  + Defines the API routes for different services.
  + Files:
    - auth.route.js: Routes for user authentication (login, register).
    - availability.route.js: Routes for mentor availability.
    - booking.route.js: Routes for booking management.
    - home.route.js: General home route.
    - mentor.route.js: Routes for mentor management.
    - service.route.js: General service routes.
    - user.route.js: Routes for user management.
    - webhook.route.js: Routes for webhook handling (e.g., payment notifications).
* **services/**
  + Contains service files that handle the business logic of the application.
  + Files:
    - auth.service.js: Manages authentication services like login and registration.
    - availability.service.js: Handles logic related to mentor availability.
    - booking.service.js: Manages the logic for handling bookings.
    - email.service.js: Manages sending emails (e.g., booking confirmation).
    - mentor.service.js: Handles mentor-specific services.
    - service.service.js: Provides general services across the platform.
    - token.service.js: Manages JWT token generation and validation.
    - user.service.js: Handles user-related business logic.
    - zoom.service.js: Provides integration with Zoom for session bookings.
* **template/**
  + Stores templates used by the backend, such as email templates.
  + confirmation.ejs: Template for booking confirmation emails.
* **util/**
  + Helper utilities and constants.
  + Files:
    - httpStatus.js: HTTP status codes used across the application.
* **validations/**
  + Schema validation using Joi for different API endpoints.
  + Files:
    - auth.validation.js: Validates user registration and login inputs.
    - availability.validation.js: Validates mentor availability inputs.
    - booking.validation.js: Validates booking-related inputs.
    - service.validation.js: Validates general service inputs.
    - user.validation.js: Validates user profile inputs.

# **Project Setup**

#### **Prerequisites**

Before you start, ensure you have the following installed on your machine:

* **Node.js** (v14 or above)
* **MongoDB** (Make sure your MongoDB instance is running and accessible)
* **Cloudinary** (For image storage)
* **Zoom API credentials** (For online meetings)
* **Razorpay** (For payment processing)

#### **Steps to Set Up the Backend**

1. **Clone the Repository**

git clone <your-repository-url>

cd mentorhub-backend

**2** **Install Dependencies** Run the following command to install all required dependencies:

npm install

This will install all the project dependencies and devDependencies listed in the package.json file:

* **bcrypt**, **cloudinary**, **cookie-parser**, **cors**, **dotenv**, **ejs**, **express**, **joi**, **jsonwebtoken**, **moment**, **mongoose**, **multer**, **nodemailer**, **razorpay**, etc.

**3 Configure Environment Variables** You need to create a .env file at the root of your project directory. Use the .env.example file as a reference.

cp .env.example .env

In the .env file, fill in the following details:

PORT=<port number>

DB\_URL=<mongodb url>

CLOUDINARY\_CLOUD\_NAME=<cloudinary cloud name>

CLOUDINARY\_API\_KEY=<cloudinary api key>

CLOUDINARY\_API\_SECRET=<cloudinary api secret>

SMTP\_PASSWORD=<smtp password>

SMTP\_USERNAME=<smtp username>

EMAIL\_FROM=<email from>

SMTP\_HOST=<smtp host>

SMTP\_PORT=<smtp port>

RAZORPAY\_KEY\_ID=<razorpay key id>

RAZORPAY\_KEY\_SECRET=<razorpay key secret>

ZOOM\_ACCOUNT\_ID=<zoom account id>

ZOOM\_CLIENT\_ID=<zoom client id>

ZOOM\_CLIENT\_SECRET=<zoom client secret>

**4 Run the Development Server** To start the server in development mode, use:

Npm run dev

This command uses **nodemon**, which automatically restarts the server when file changes are detected.

**5 Running the Server in Production** To start the server in production mode, use:

npm start

**6 Verify the Setup**

Use a tool like Postman or cURL to hit the API endpoints defined in the project.

* Check if the server is running without any errors, and verify connections to MongoDB, Cloudinary, and other external services.
* Open a browser and go to http://localhost:<PORT> to check if the server is running.
* Ensure MongoDB is connected and functional.
* Verify that third-party services like Cloudinary, SMTP, Zoom, and Razorpay are configured correctly.