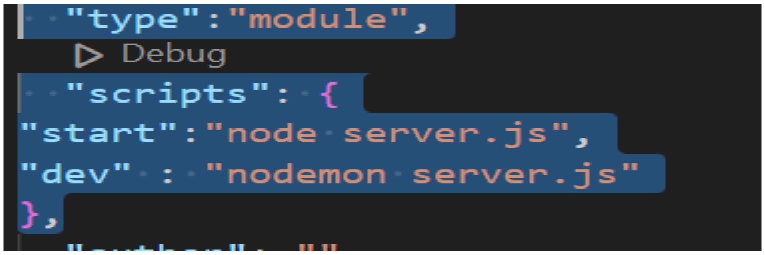
`

`Backend folder create → npm init → npm i bcryptjs cloudinary cookie-parser dotenv express mongoose jsonwebtoken express-fileupload validator
npm install --save-dev nodemon
npx nodemon server.js



```
  "type":"module",
    ▷ Debug
  "scripts": {
"start":"node server.js",
"dev" : "nodemon server.js"
},
```

Create a project and cluster in atlas
Psw MiKTsm88PDHJ2vrr
Ugs21016csetriveni
mongodb+srv://ugs21016csetriveni:MiKTsm88PDHJ2vrr@cluster0.5vqfu.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0
Goto cloudinary.com -> settings→ prod env → Api key generate →
Account settings → prod env → cloud name
config.env inside config folder

dbConnection.js

```
import mongoose from "mongoose"
export const dbConnection = () => {
    mongoose.connect(process.env.MONGO_URL,{
        dbName:"HOSPITAL_MANAGEMENT"
    }).then(()=>{
        console.log("DB Connected successfully")
    })
    .catch((err)=>{
    console.log("Error : DB Conection failed ")
    })
}
```

server.js

```
import app from './app.js';
import cloudinary from 'cloudinary'
cloudinary.v2.config({
    cloud_name: process.env.CLOUDINARY_CLOUD_NAME,
    api_key: process.env.CLOUDINARY_API_KEY,
    api_secret: process.env.CLOUDINARY_API_SECRET,
})
app.listen(process.env.PORT,()=>{
    console.log(`Server listening at port number: ${process.env.PORT}`)});
```

`

app.js

```
import express from "express"
import {config} from "dotenv"
import cors from "cors"
import cookieParser from 'cookie-parser';
import fileUpload from "express-fileupload";
import { dbConnection } from "./database/dbConnection.js";
const app = express()
config({path: "./config/config.env"})
app.use(cors(
    {
        origin:[process.env.FRONTEND_URL,process.env.DASHBOARD_URL],
        methods : ["GET","POST","PUT","DELETE"],
        credentials:true,
    }
    ))
app.use(cookieParser())
app.use(express.json())
app.use(fileUpload({
    useTempFiles:true,
    tempFileDir:"/tmp/"
}))
app.use(express.urlencoded({extended: true}))
dbConnection();
export default app;
```

`dotenv`: A module for loading environment variables from a `.env` file into `process.env`.

`cors`: Middleware for enabling Cross-Origin Resource Sharing (CORS), which allows your server to handle requests from different origins.

`cookie-parser`: Middleware for parsing cookies attached to the client request object.

`express-fileupload`: Middleware for handling file uploads in Express applications.

`config({ path: "./config/config.env" });`: Loads environment variables from the `config.env` file into `process.env`.

Configures CORS to allow requests from specific origins (`FRONTEND_URL` and `DASHBOARD_URL`), and allows various HTTP methods (`GET`, `POST`, `PUT`, `DELETE`). The `credentials: true` option allows cookies to be included in cross-origin requests.

`

app.use(cookieParser());: Adds middleware to parse cookies from incoming requests, making them accessible via req.cookies.

app.use(express.json());: Middleware for parsing JSON bodies in incoming requests.

app.use(fileUpload({...})): Configures express-fileupload to handle file uploads. useTempFiles: true uses temporary files during the upload process, and tempFileDir: "/tmp/" specifies the directory for these temporary files.
app.use(express.urlencoded({ extended: true }));: Middleware for parsing URL-encoded bodies (e.g., from forms). extended: true allows for rich objects and arrays to be encoded into the URL-encoded format.

Still now we made a successful connection with database and running at 4000 port

Lets create message box
catchAsyncErrors.js

```
export const catchAsyncErrors  = (theFunction) => {
    return (req,res,next)=>{
        Promise.resolve(theFunction(req,res,next)).catch(next);
    } }
```

catchAsyncErrors is a higher-order function that takes theFunction as an argument. This theFunction is expected to be an asynchronous function (or a function that returns a promise).
 returns a new function that takes the standard Express middleware parameters: req, res, and next.
Promise.resolve(theFunction(req, res, next)): This ensures that theFunction is treated as a promise. If theFunction is already returning a promise, Promise.resolve will not alter it. If it’s not, Promise.resolve will wrap it in a promise.
.catch(next): This catches any errors that occur during the execution of theFunction. If an error occurs, it is passed to the next middleware function, which is typically an Express error-handling middleware.

error.js

```
class ErrorHandler extends Error {
    constructor(message, statusCode) {
        super(message);
        this.statusCode = statusCode;
    }
}
```

`

```
export const errorMiddleware = (err, req, res, next) => {
    // Default error message and status code
    err.message = err.message || "Server Error";
    err.statusCode = err.statusCode || 500;
    // Specific error handling
    if (err.code === 11000) {
        // Handle duplicate key error
        err = new ErrorHandler(`Duplicate ${Object.keys(err.keyValue)}
entered`, 400);
    } else if (err.name === "JsonWebTokenError") {
        // Handle JWT errors
        err = new ErrorHandler("JSON Web Token is invalid", 400);
    } else if (err.name === "TokenExpiredError") {
        // Handle expired JWT
        err = new ErrorHandler("JSON Web Token has expired", 400);
    } else if (err.name === "CastError") {
        // Handle invalid MongoDB object ID
        err = new ErrorHandler(`Invalid ${err.path}`, 400);
    }
    // Handle validation errors with ternary operator
    const errorMessage = err.errors
        ? Object.values(err.errors).map(error => error.message).join(" ")
        : err.message;
    // Send the error response
    res.status(err.statusCode).json({
        success: false,
        message: errorMessage,
    });
};
export default ErrorHandler;
```

The `error.js` file provides custom error handling for an Express.js application. It defines an `ErrorHandler` class and a middleware function to catch and format errors before sending a response to the client.

`ErrorHandler` extends the built-in `Error` class to include an additional property, `statusCode`. This helps in setting a custom HTTP status code for different types of errors.

Constructor Parameters:`message`: The error message to be sent to the client. `statusCode`: The HTTP status code associated with the error.

`err.message = err.message || "Server Error";`

`

err.statusCode = err.statusCode || 500;Sets default values for err.message and err.statusCode if they are not already defined by the error.

Validation Error Messages: If err.errors is present (typically from Mongoose validation errors), it maps over all validation errors to create a concatenated error message. Otherwise, it uses the err.message as the error message.

messageSchema.js

```
import mongoose from "mongoose";
import validator from "validator";
const messageSchema = new mongoose.Schema({
    firstName: {
        type: String,
        required: true,
        minlength: [3, "First name must be at least 3 characters long"]
    },
    lastName: {
        type: String,
        required: true,
        minlength: [3, "Last name must be at least 3 characters long"]
    },
    email: {
        type: String,
        required: true,
        validate: [validator.isEmail, "Please provide a valid email"]
    },
    phone: {
        type: String,
        required: true,
        minlength: [10, "Phone number must be at least 10 digits long"],
        maxlength: [15, "Phone number cannot exceed 15 digits"]
    },
    message: {
        type: String,
        required: true,
        minlength: [10, "Message must be at least 10 characters long"]
    }
});
export const Message = mongoose.model("Message", messageSchema);
```

validator.isEmail(email, [options])

messageRouter.js

`

```
import express from 'express'
import { sendMessage } from '../controller/messageController.js'
const router = express.Router()
router.post("/send",sendMessage)
export default router
```

messageController.js

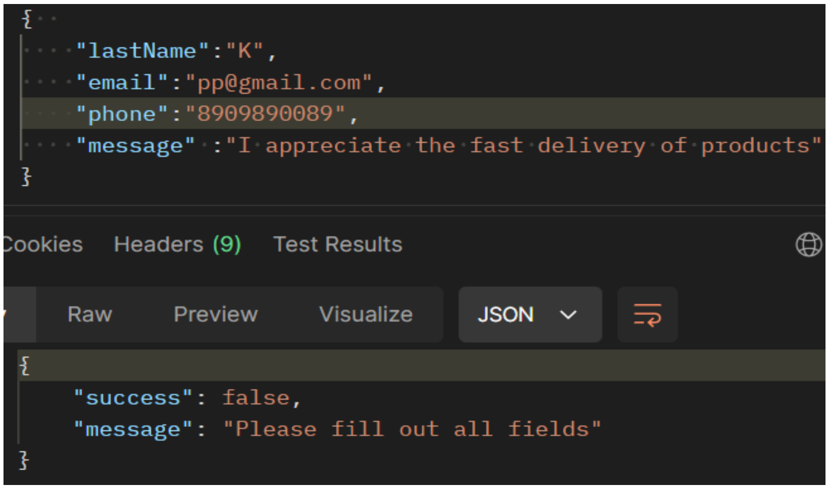
```
import { Message } from "../models/messageSchema.js";
import { catchAsyncErrors } from "../middleware/catchAsyncErrors.js";
import ErrorHandler from "../middleware/errorMiddleware.js";
export const sendMessage = catchAsyncErrors(async (req, res, next) => {
    const { firstName, lastName, email, phone, message } = req.body;

    if (!firstName || !lastName || !email || !phone || !message) {
        return next(new ErrorHandler("Please fill out all fields", 400));
    }
    await Message.create({ firstName, lastName, email, phone, message });
    res.status(200).json({
        success: true,
        message: "Message sent successfully",
    });
});
```

app.js add this

```
import messageRouter from "./router/messageRouter.js";
import { errorMiddleware } from "./middleware/errorMiddleware.js";
app.use(errorMiddleware) # at last
```

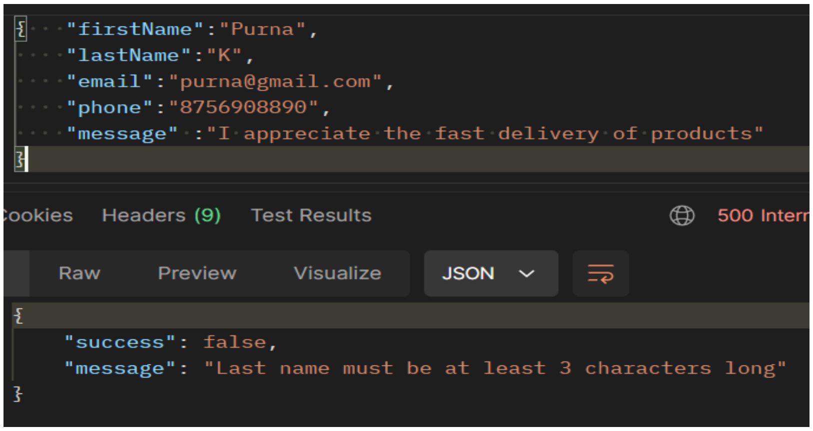
http://localhost:4000/api/v1/message/send

`



```
{
    "lastName":"K",
    "email":"pp@gmail.com",
    "phone":"8909890089",
    "message" :"I appreciate the fast delivery of products"
}
```

```
{
    "success": false,
    "message": "Please fill out all fields"
}
```



```
{   "firstName":"Purna",
    "lastName":"K",
    "email":"purna@gmail.com",
    "phone":"8756908890",
    "message" :"I appreciate the fast delivery of products"
}
```

```
{
    "success": false,
    "message": "Last name must be at least 3 characters long"
}
```

Utils → jwtToken.js

```
export const generateToken = (user, message, statusCode, res) => {
    const token = user.generateJsonWebToken();
    // Determine the cookie name based on the user's role
    const cookieName = user.role === 'Admin' ? 'adminToken' :
'patientToken';
    res
      .status(statusCode)
      .cookie(cookieName, token, {
        expires: new Date(
          Date.now() + process.env.COOKIE_EXPIRE * 24 * 60 * 60 * 1000
        ),
        httpOnly: true,
      })
      .json({
        success: true,
        message,
        user,
```

`

```
    token,
  });
};
```

user: This is the user object that contains user information (e.g., the user's role and method to generate the token).

message: A string message that will be included in the JSON response.

statusCode: The HTTP status code to be sent back to the client (e.g., 200 for success, 400 for errors).

res: The response object, used to send back the cookie and JSON data to the client.

const token = user.generateJsonWebToken();

This line generates a JSON Web Token (JWT) for the user by calling a method (generateJsonWebToken()) on the user object

The cookie name is either 'adminToken' or 'patientToken', as determined earlier.

The value of the cookie is the JWT (token).

The expires option sets the cookie to expire after a certain number of days, calculated from process.env.COOKIE_EXPIRE.

The httpOnly: true option ensures that the cookie cannot be accessed via JavaScript on the client side, improving security.

- The function generates a JWT for the user.
- It sets the token as an HTTP-only cookie with a name based on the user's role (admin or patient).
- It responds with a JSON object containing the success status, a message, user data, and the token.

This approach is often used in authentication systems, where JWTs are used to maintain user sessions. The token is stored in a cookie, which is then used to authenticate the user on subsequent requests.

tokens are typically used to maintain sessions and to verify that the user is allowed to access certain resources or perform certain actions.

A JWT has three parts:

- **Header**: Contains information about how the token is encoded (e.g., type of token and the hashing algorithm).

`

- **Payload:** Contains the user data (claims) such as user ID, email, or role. This part can be decoded by anyone, but since it's signed, it can't be tampered with.
- **Signature:** A cryptographic signature that is used to verify the token's authenticity.

When a user logs in, a token (like a JWT) is generated and sent to the client. The client then includes this token in future requests (usually in the headers or cookies), and the server verifies the token to identify and authenticate the user.

This token can be passed with the request, and the server will decode and verify it to authenticate the user.

Why Use Tokens?

- **Stateless Authentication:** The server doesn't need to store session data. All the information about the user is encoded in the token.
- **Scalability:** Easier to scale applications since no server-side session storage is needed.

A **cookie** is a small piece of data stored on the client's (user's) browser. Cookies are used to store data that can be sent back to the server with each HTTP request, allowing the server to "remember" information between page loads or visits.

Key Features of Cookies:

- **Stored in the browser:** Cookies are automatically sent with every HTTP request to the server for the domain that set the cookie.
- **Small size:** Typically limited to around 4 KB in size.
- **Expiration:** Cookies can be set to expire at a certain time, after which they are automatically deleted by the browser.
- **Security:**
  - **HttpOnly:** Cookies can be flagged as `HttpOnly`, which makes them inaccessible to JavaScript, reducing the risk of XSS (cross-site scripting) attacks.
  - **Secure:** Cookies can be marked as `Secure`, which means they will only be sent over HTTPS connections.

How Cookies Work:

- When the server wants to store a piece of information (like a session ID or token) on the client side, it sends a `Set-Cookie` header in the HTTP response.

`

- The client (browser) then stores this cookie and automatically includes it in future HTTP requests to the server.

Example:
css
Copy code

```
Set-Cookie: sessionId=abc123; HttpOnly; Secure; Expires=Wed, 21 Oct 2025
07:28:00 GMT;
```

- This sets a cookie called `sessionId` with a value of `abc123`.
- `HttpOnly`: The cookie cannot be accessed via JavaScript.
- `Secure`: The cookie is only sent over HTTPS connections.
- `Expires`: Specifies when the cookie will expire.

Why Use Cookies?

- **Session Management:** Cookies are often used to manage sessions, where a session ID is stored in the cookie, allowing the server to remember the user between requests.
- **Tracking:** Cookies can be used to store information about user activity (like what items are in a shopping cart).

Key Differences Between Tokens and Cookies:

- **Tokens** (like JWTs) are usually stored in **localStorage** or **cookies** and are included in requests as part of the Authorization header or as a cookie.
- **Cookies** are a mechanism for storing data in the user's browser and can store a variety of information, including tokens.

How They Work Together:

- In the code you provided, the token (JWT) is stored in a **cookie.** This means:
  1. A JWT is generated when a user logs in.
  2. The JWT is stored in a cookie in the user's browser.
  3. The browser automatically sends this cookie with every HTTP request to the server.
  4. The server uses the JWT to verify and authenticate the user on each request.

By using both tokens and cookies together, the application can manage user sessions securely and efficiently.

`

userSchema.js

```
import mongoose from "mongoose";
import validator from "validator";
import bcrypt from "bcryptjs";
import jwt from "jsonwebtoken";

const userSchema = new mongoose.Schema({
  firstName: {
    type: String,
    required: [true, "First Name Is Required!"],
    minLength: [3, "First Name Must Contain At Least 3 Characters!"],
  },
  lastName: {
    type: String,
    required: [true, "Last Name Is Required!"],
    minLength: [3, "Last Name Must Contain At Least 3 Characters!"],
  },
  email: {
    type: String,
    required: [true, "Email Is Required!"],
    validate: [validator.isEmail, "Provide A Valid Email!"],
  },
  phone: {
    type: String,
    required: [true, "Phone Is Required!"],
    minLength: [10, "Phone Number Must Contain Exact 10 Digits!"],
    maxLength: [10, "Phone Number Must Contain Exact 10 Digits!"],
  },
  nic: {
    type: String,
    required: [true, "NIC Is Required!"],
    minLength: [13, "NIC Must Contain Only 13 Digits!"],
    maxLength: [13, "NIC Must Contain Only 13 Digits!"],
  },
  dob: {
    type: Date,
    required: [true, "DOB Is Required!"],
  },
```

`

```
  gender: {
    type: String,
    required: [true, "Gender Is Required!"],
    enum: ["Male", "Female"],
  },
  password: {
    type: String,
    required: [true, "Password Is Required!"],
    minLength: [8, "Password Must Contain At Least 8 Characters!"],
    select: false, // Don't select password by default
  },
  role: {
    type: String,
    required: [true, "User Role Required!"],
    enum: ["Patient", "Doctor", "Admin"],
  },
  doctorDepartment: {
    type: String,
  },
  docAvatar: {
    public_id: String,
    url: String,
  },
});

// Hash the password before saving it to the database
userSchema.pre("save", async function (next) {
  if (!this.isModified("password")) {
    return next();
  }
  this.password = await bcrypt.hash(this.password, 10); // Hashing password
  next();
});


// Compare the password with the stored hashed password
userSchema.methods.comparePassword = async function (enteredPassword) {
  return await bcrypt.compare(enteredPassword, this.password); // Comparing hashed password
```

`

```
};

// Generate JWT token
userSchema.methods.generateJsonWebToken = function () {
  return jwt.sign({ id: this._id }, process.env.JWT_SECRET_KEY, {
    expiresIn: process.env.JWT_EXPIRES,
  });
};

export const User = mongoose.model("User", userSchema);
```

userController.js

```
import { catchAsyncErrors } from "../middlewares/catchAsyncErrors.js";
import { User } from "../models/userSchema.js";
import ErrorHandler from "../middlewares/error.js";
import { generateToken } from "../utils/jwtToken.js";
import cloudinary from "cloudinary";
import bcrypt from "bcryptjs"

export const patientRegister = catchAsyncErrors(async (req, res, next) =>
{
  const { firstName, lastName, email, phone, nic, dob, gender, password }
=
    req.body;
  if (
    !firstName ||
    !lastName ||
    !email ||
    !phone ||
    !nic ||
    !dob ||
    !gender ||
    !password
  ) {
    return next(new ErrorHandler("Please Fill Full Form!", 400));
  }


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

`

```
  if (isRegistered) {
    return next(new ErrorHandler("User already Registered!", 400));
  }

  const user = await User.create({
    firstName,
    lastName,
    email,
    phone,
    nic,
    dob,
    gender,
    password,
    role: "Patient",
  });
  generateToken(user, "User Registered!", 200, res);
});



export const login = async (req, res, next) => {
  try {
    const { email, password, role } = req.body;

    // Validate input
    if (!email || !password || !role) {
      return next(new ErrorHandler("Please fill in all fields!", 400));
    }

    // Validate role
    if (!["Patient", "Doctor", "Admin"].includes(role)) {
      return next(new ErrorHandler("Invalid role!", 400));
    }

    // Find the user by email and include password
    const user = await User.findOne({ email }).select("+password");
    if (!user) {
      return next(new ErrorHandler("Invalid email or password!", 401));
    }
```

`

```
    // Compare the password
    const isPasswordMatch = await user.comparePassword(password);
    if (!isPasswordMatch) {
      return next(new ErrorHandler("Invalid email or password!", 401));
    }

    // Check if the role matches
    if (user.role !== role) {
      return next(new ErrorHandler("User not found with this role!",
401));
    }

    // Use generateToken to issue and send the token
    generateToken(user, "Login successful!", 200, res);
  } catch (error) {
    return next(new ErrorHandler(error.message, 500));
  }
};
```

http://localhost:4000/api/v1/user/patient/register

```
{
    "firstName":"Karthika",
    "lastName":"Vandan",
    "email":"karthikavandan@gmail.com",
    "phone":"7890890989",
    "0000000000password":"karthikavandan",
    "dob":"11/09/2003",
    "nic":"1234567891234",
    "gender":"Female",
    "role":"Patient"
}
```

```
{
    "success": true,
    "message": "User Registered!",
    "user": {
        "firstName": "Sandeep",
        "lastName": "kavi",
        "email": "sandeepkavi@gmail.com",
        "phone": "9878887777",
```

`

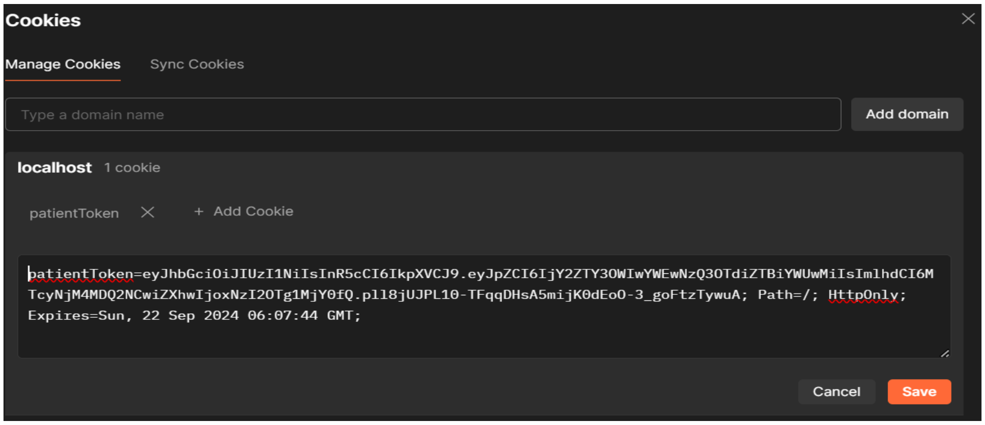
```
        "nic": "1234567891234",
        "dob": "2003-11-08T18:30:00.000Z",
        "gender": "Male",
        "password": "$2a$10$ekkFiZpQfaHegaHwNGJrwelNrdq3NxwswcKdjMoUkQy.Lesvj4p9S",
        "role": "Patient",
        "_id": "66e679b0aa074797be0bae02",
        "__v": 0
    },
    "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY2ZTY3OWIwYWEwNzQ3OTdiZTBiYWUwMiIsImlh
dCI6MTcyNjM4MDQ2NCwiZXhwIjoxNzI2OTg1MjY0fQ.pll8jUJPL10-TFqqDHsA5mijK0dEoO-3_goFtzTywuA
"
}
```

http://localhost:4000/api/v1/user/login   for postman testing

```
{
    "email": "sandeepkavi@gmail.com",
    "password": "sandeepavi",
    "role": "Patient"
}
```

```
{
    "success": true,
    "message": "Login successful!",
    "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY2ZTY3OWIwYWEwNzQ3OTdiZTBiYWUwMiIsImlh
dCI6MTcyNjM4MjE4NSwiZXhwIjoxNzI2OTg2OTg1fQ.RHLqBzROzPK8iFHlB-AiXYfcCJImlS56-FQYGpHQD8U
"
}
```

A cookie is generated with name patientToken



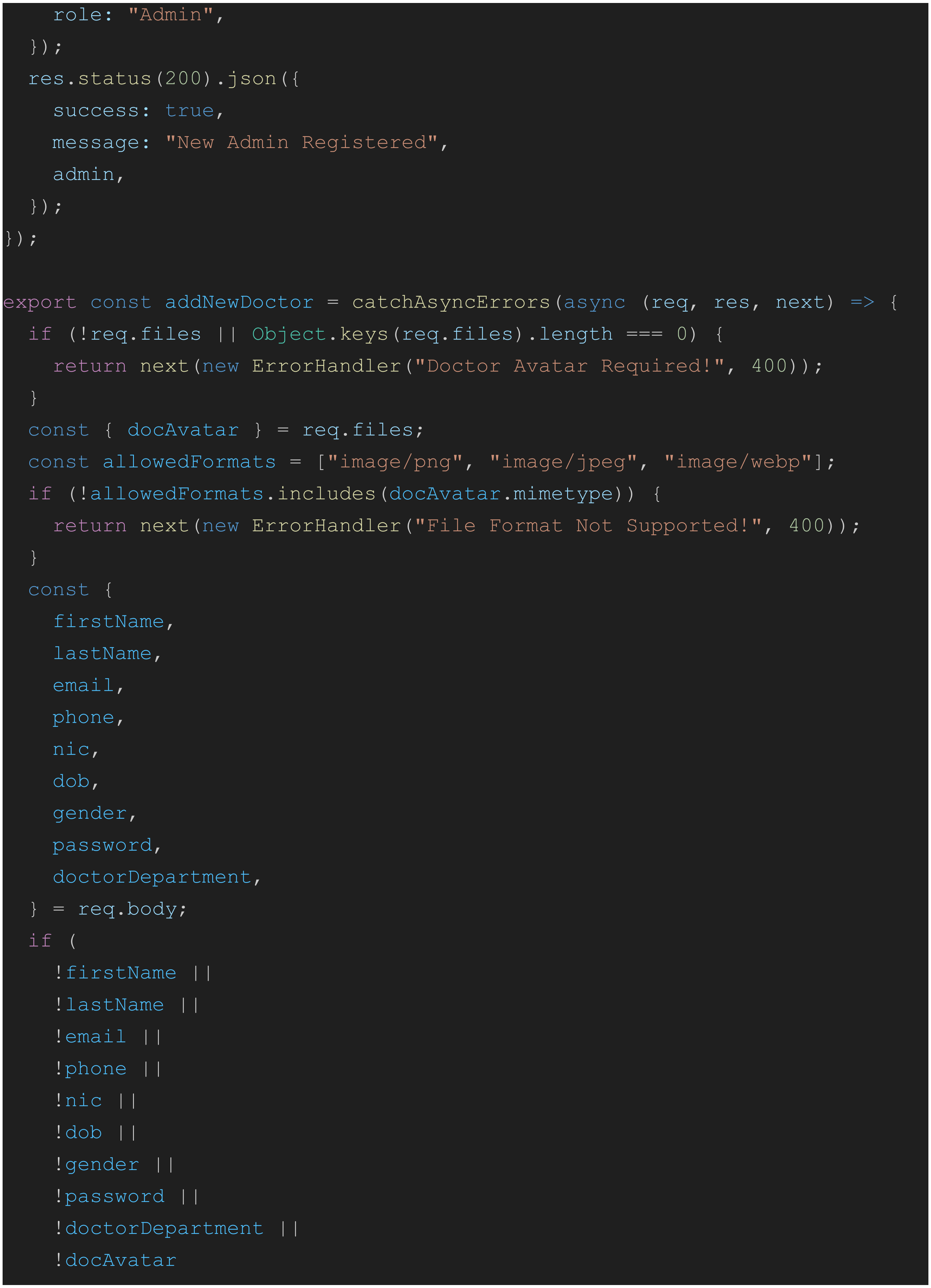
```
patientToken=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY2ZTY3OWIwYWEwNzQ3OTdiZTBiYWUwMiIsImlhdCI6M
TcyNjM4MDQ2NCwiZXhwIjoxNzI2OTg1MjY0fQ.pll8jUJPL10-TFqqDHsA5mijK0dEoO-3_goFtzTywuA; Path=/; HttpOnly;
Expires=Sun, 22 Sep 2024 06:07:44 GMT;
```

`

Admin user creation
userController.js

```
export const addNewAdmin = catchAsyncErrors(async (req, res, next) => {
  const { firstName, lastName, email, phone, nic, dob, gender, password }
=
    req.body;
  if (
    !firstName ||
    !lastName ||
    !email ||
    !phone ||
    !nic ||
    !dob ||
    !gender ||
    !password
  ) {
    return next(new ErrorHandler("Please Fill Full Form!", 400));
  }

  const isRegistered = await User.findOne({ email });
  if (isRegistered) {
    return next(new ErrorHandler("Admin With This Email Already Exists!",
400));
  }

  const admin = await User.create({
    firstName,
    lastName,
    email,
    phone,
    nic,
    dob,
    gender,
    password,
```

`



```
    role: "Admin",
  });
  res.status(200).json({
    success: true,
    message: "New Admin Registered",
    admin,
  });
});

export const addNewDoctor = catchAsyncErrors(async (req, res, next) => {
  if (!req.files || Object.keys(req.files).length === 0) {
    return next(new ErrorHandler("Doctor Avatar Required!", 400));
  }
  const { docAvatar } = req.files;
  const allowedFormats = ["image/png", "image/jpeg", "image/webp"];
  if (!allowedFormats.includes(docAvatar.mimetype)) {
    return next(new ErrorHandler("File Format Not Supported!", 400));
  }
  const {
    firstName,
    lastName,
    email,
    phone,
    nic,
    dob,
    gender,
    password,
    doctorDepartment,
  } = req.body;
  if (
    !firstName ||
    !lastName ||
    !email ||
    !phone ||
    !nic ||
    !dob ||
    !gender ||
    !password ||
    !doctorDepartment ||
    !docAvatar
```

`

```
  ) {
    return next(new ErrorHandler("Please Fill Full Form!", 400));
  }
  const isRegistered = await User.findOne({ email });
  if (isRegistered) {
    return next(
      new ErrorHandler("Doctor With This Email Already Exists!", 400)
    );
  }
  const cloudinaryResponse = await cloudinary.uploader.upload(
    docAvatar.tempFilePath
  );
  if (!cloudinaryResponse || cloudinaryResponse.error) {
    console.error(
      "Cloudinary Error:",
      cloudinaryResponse.error || "Unknown Cloudinary error"
    );
    return next(
      new ErrorHandler("Failed To Upload Doctor Avatar To Cloudinary",
500)
    );
  }
  const doctor = await User.create({
    firstName,
    lastName,
    email,
    phone,
    nic,
    dob,
    gender,
    password,
    role: "Doctor",
    doctorDepartment,
    docAvatar: {
      public_id: cloudinaryResponse.public_id,
      url: cloudinaryResponse.secure_url,
    },
  });
  res.status(200).json({
    success: true,
```

`

```
    message: "New Doctor Registered",
    doctor,
  });
});

export const getAllDoctors = catchAsyncErrors(async (req, res, next) => {
  const doctors = await User.find({ role: "Doctor" });
  res.status(200).json({
    success: true,
    doctors,
  });
});

export const getUserDetails = catchAsyncErrors(async (req, res, next) => {
  const user = req.user;
  res.status(200).json({
    success: true,
    user,
  });
});

// Logout function for dashboard admin
export const logoutAdmin = catchAsyncErrors(async (req, res, next) => {
  res
    .status(201)
    .cookie("adminToken", "", {
      httpOnly: true,
      expires: new Date(Date.now()),
    })
    .json({
      success: true,
      message: "Admin Logged Out Successfully.",
    });
});

// Logout function for frontend patient
export const logoutPatient = catchAsyncErrors(async (req, res, next) => {
  res
    .status(201)
    .cookie("patientToken", "", {
```

`

```
      httpOnly: true,
      expires: new Date(Date.now()),
    })
    .json({
      success: true,
      message: "Patient Logged Out Successfully.",
    });
});
```

auth.js in middlewares folder

```
import { User } from "../models/userSchema.js";
import { catchAsyncErrors } from "./catchAsyncErrors.js";
import ErrorHandler from "./error.js";
import jwt from "jsonwebtoken";

// Middleware to authenticate dashboard users
export const isAdminAuthenticated = catchAsyncErrors(
  async (req, res, next) => {
    const token = req.cookies.adminToken;
    if (!token) {
      return next(
        new ErrorHandler("Dashboard User is not authenticated!", 400)
      );
    }
    const decoded = jwt.verify(token, process.env.JWT_SECRET_KEY);
    req.user = await User.findById(decoded.id);
    if (req.user.role !== "Admin") {
      return next(
        new ErrorHandler(`${req.user.role} not authorized for this
resource!`, 403)
      );
    }
    next();
  }
);

// Middleware to authenticate frontend users
export const isPatientAuthenticated = catchAsyncErrors(
```

`

```
  async (req, res, next) => {
    const token = req.cookies.patientToken;
    if (!token) {
      return next(new ErrorHandler("User is not authenticated!", 400));
    }
    const decoded = jwt.verify(token, process.env.JWT_SECRET_KEY);
    req.user = await User.findById(decoded.id);
    if (req.user.role !== "Patient") {
      return next(
        new ErrorHandler(`${req.user.role} not authorized for this
resource!`, 403)
      );
    }
    next();
  }
);

export const isAuthorized = (...roles) => {
  return (req, res, next) => {
    if (!roles.includes(req.user.role)) {
      return next(
        new ErrorHandler(
          `${req.user.role} not allowed to access this resource!`
        )
      );
    }
    next();
  };
};
```

userRouter

```
import express from "express";
import { login, patientRegister, addNewAdmin, addNewDoctor, getAllDoctors,
getUserDetails, logoutPatient, logoutAdmin} from
"../controller/userController.js";
import {
```

`

```
  isAdminAuthenticated,
  isPatientAuthenticated,
} from "../middlewares/auth.js";
const router = express.Router();
router.post("/patient/register", patientRegister);
router.post("/login", login);
router.post("/admin/addnew",isAdminAuthenticated,  addNewAdmin);
router.post("/doctor/addnew", isPatientAuthenticated, addNewDoctor);
router.get("/doctors", getAllDoctors);
router.get("/patient/me", isPatientAuthenticated, getUserDetails);
router.get("/admin/me", isAdminAuthenticated, getUserDetails);
router.get("/patient/logout", isPatientAuthenticated, logoutPatient);
router.get("/admin/logout", isAdminAuthenticated, logoutAdmin);
export default router;
```

http://localhost:4000/api/v1/user/admin/addnew

```
{
    "firstName":"Kruthika",
    "lastName":"Vandan",
    "email":"admin1@gmail.com",
    "phone":"7890890989",
    "password":"Admin@123",
    "dob":"11/09/2003",
    "nic":"1234567891234",
    "gender":"Female"
}
```

```
{
    "success": true,
    "message": "New Admin Registered",
    "admin": {
        "firstName": "Kruthika",
        "lastName": "Vandan",
        "email": "admin1@gmail.com",
        "phone": "7890890989",
        "nic": "1234567891234",
        "dob": "2003-11-08T18:30:00.000Z",
        "gender": "Female",
        "password": "$2a$10$8FPcyzt5rI/DbMPTEU.6deDJuKUQf77AfNFatncR8mrSP.wrvQEQ.",
```

`

```
        "role": "Admin",
        "_id": "66e6fee08ae53b42a1f56c9f",
        "__v": 0
    }
}
```

http://localhost:4000/api/v1/user/login

```
{
    "email": "admin1@gmail.com",
    "password": "Admin@123",
    "role": "Admin"
}  ------------------------------------------
{
    "success": true,
    "message": "Login successful!",
    "user": {
        "_id": "66e6fee08ae53b42a1f56c9f",
        "firstName": "Kruthika",
        "lastName": "Vandan",
        "email": "admin1@gmail.com",
        "phone": "7890890989",
        "nic": "1234567891234",
        "dob": "2003-11-08T18:30:00.000Z",
        "gender": "Female",
        "password": "$2a$10$8FPcyzt5rI/DbMPTEU.6deDJuKUQf77AfNFatncR8mrSP.wrvQEQ.",
        "role": "Admin",
        "__v": 0
    },
    "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY2ZTZmZWUwOGFlNTNiNDJhMWY1NmM5ZiIsImlh
dCI6MTcyNjQxNTIzOCwiZXhwIjoxNzI3MDIwMDM4fQ.x9WMS1TQZCWQvZfj28kbp7wZ18FPgnNPbC_5QQh2XVA
"
}
```

Login any one Admin , …login successful and a adminToken is generated as cookie

If you click on getAdmin now, you get that admin who is logged in

`

GET request
http://localhost:4000/api/v1/user/admin/me

```
{
    "success": true,
    "user": {
        "_id": "66e6fee08ae53b42a1f56c9f",
        "firstName": "Kruthika",
        "lastName": "Vandan",
        "email": "admin1@gmail.com",
        "phone": "7890890989",
        "nic": "1234567891234",
        "dob": "2003-11-08T18:30:00.000Z",
        "gender": "Female",
        "role": "Admin",
        "__v": 0
    }
}
```

Similarly login a patient , patienttoken cookie is generated,
http://localhost:4000/api/v1/user/patient/me   GET req a patient

```
{
    "success": true,
    "user": {
        "_id": "66e679b0aa074797be0bae02",
        "firstName": "Sandeep",
        "lastName": "kavi",
        "email": "sandeepkavi@gmail.com",
        "phone": "9878887777",
        "nic": "1234567891234",
        "dob": "2003-11-08T18:30:00.000Z",
        "gender": "Male",
        "role": "Patient",
        "__v": 0
    }
}
```

Latest login one is brought in get request and cookie remains only for latest logins
2 cookies are there now, one for adminToken, patientToken

Once any Admin is logged in , they can add new doctor or new Admin

`

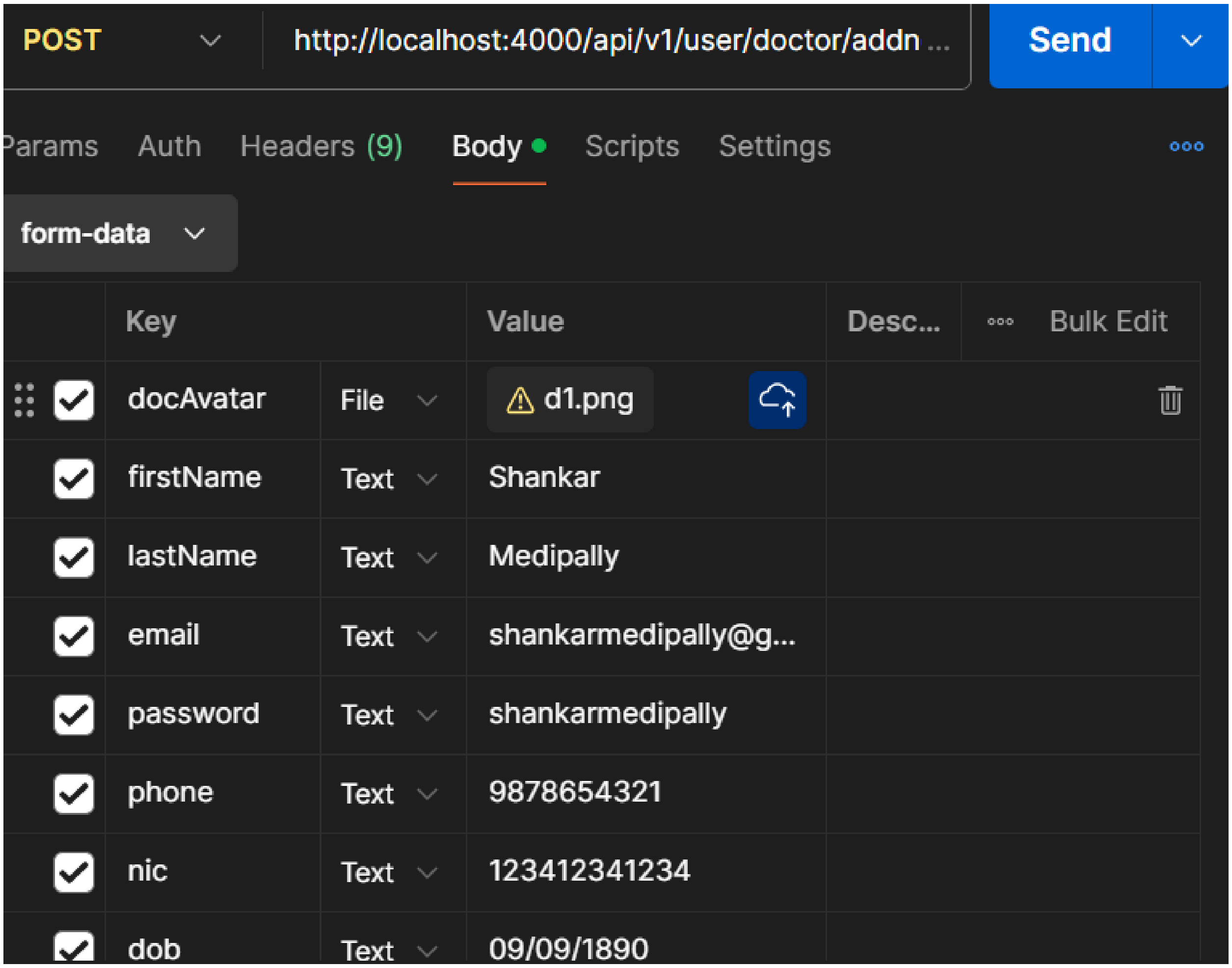
http://localhost:4000/api/v1/user/admin/addnew

```
{
    "firstName":"Hrithik",
    "lastName":"Roshan",
    "email":"admin6@gmail.com",
    "phone":"7890890989",
    "password":"Admin6@123",
    "dob":"11/05/1890",
    "nic":"1234567891234",
    "gender":"Male"
}
```

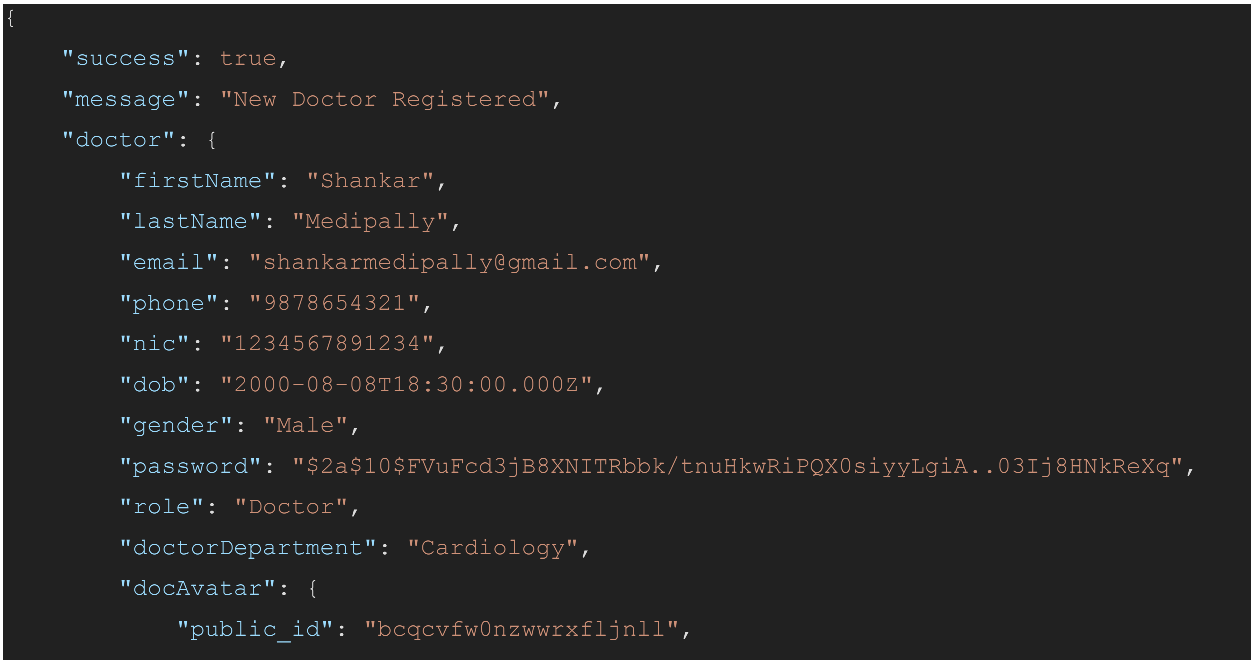
```
{
    "success": true,
    "message": "New Admin Registered",
    "admin": {
        "firstName": "Hrithik",
        "lastName": "Roshan",
        "email": "admin6@gmail.com",
        "phone": "7890890989",
        "nic": "1234567891234",
        "dob": "1890-11-04T18:38:50.000Z",
        "gender": "Male",
        "password": "$2a$10$o87K6NdkZq8KlCVq7VwNMO9HoKmH9e4FBLsBpL9H4p4WH4/uhUXEq",
        "role": "Admin",
        "_id": "66e70e526a7dcef567f2d461",
        "__v": 0
    }
}
```

To add a doctor, you need to upload docAvatar image file
So in postman, body, file select from dropDown, select file from computer

`



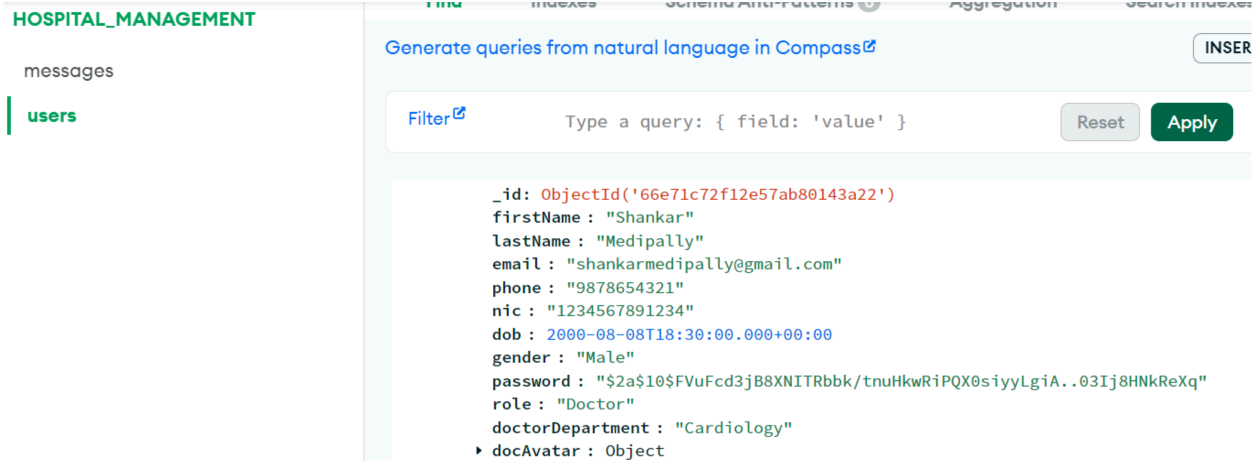
	Key		Value	Desc...	Bulk Edit
☑	docAvatar	File	⚠ d1.png		
☑	firstName	Text	Shankar		
☑	lastName	Text	Medipally		
☑	email	Text	shankarmedipally@g...		
☑	password	Text	shankarmedipally		
☑	phone	Text	9878654321		
☑	nic	Text	123412341234		
☑	dob	Text	09/09/1890		



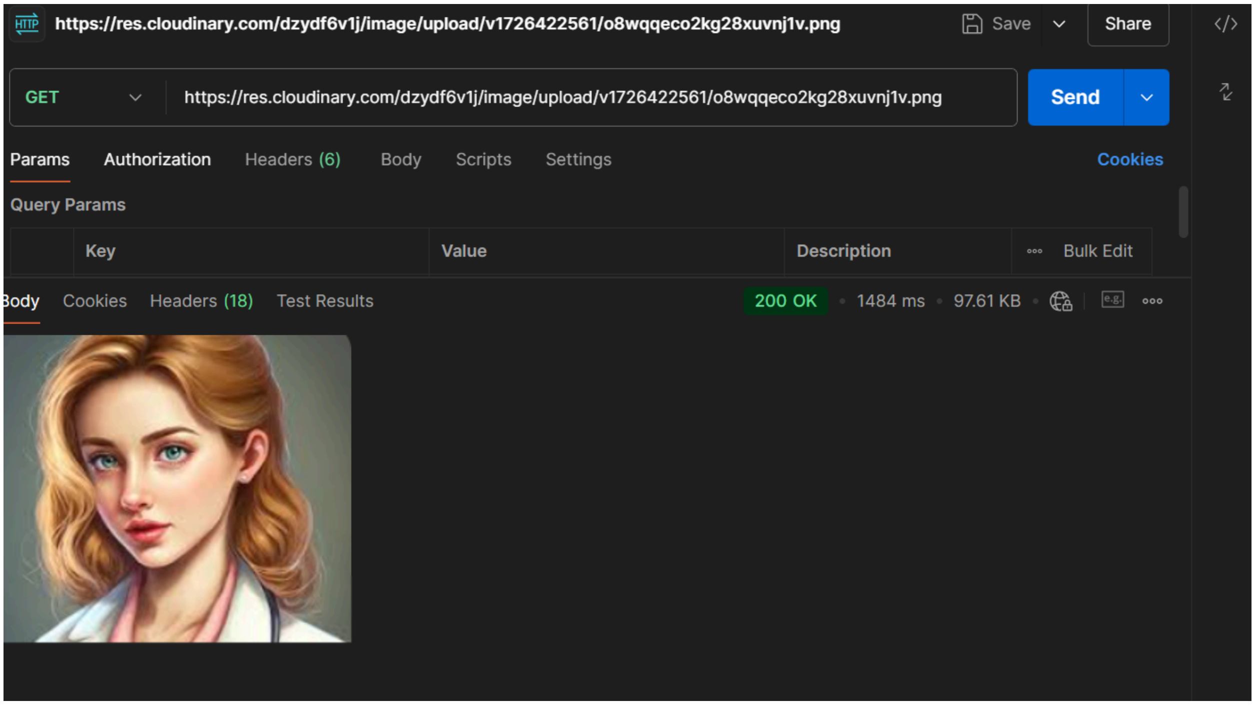
```
{
    "success": true,
    "message": "New Doctor Registered",
    "doctor": {
        "firstName": "Shankar",
        "lastName": "Medipally",
        "email": "shankarmedipally@gmail.com",
        "phone": "9878654321",
        "nic": "1234567891234",
        "dob": "2000-08-08T18:30:00.000Z",
        "gender": "Male",
        "password": "$2a$10$FVuFcd3jB8XNITRbbk/tnuHkwRiPQX0siyyLgiA..03Ij8HNkReXq",
        "role": "Doctor",
        "doctorDepartment": "Cardiology",
        "docAvatar": {
            "public_id": "bcqcvfw0nzwwrxfljnll",
```

`

```
            "url": 
"https://res.cloudinary.com/dzydf6v1j/image/upload/v1726422126/bcqcvfw0nzwwrxfljnll.pn
g"
        },
        "_id": "66e71c72f12e57ab80143a22",
        "__v": 0
    }
}
```


HOSPITAL_MANAGEMENT
messages
users
Generate queries from natural language in Compass
INSER
Filter
Type a query: { field: 'value' }
Reset
Apply

```
_id: ObjectId('66e71c72f12e57ab80143a22')
firstName : "Shankar"
lastName : "Medipally"
email : "shankarmedipally@gmail.com"
phone : "9878654321"
nic : "1234567891234"
dob : 2000-08-08T18:30:00.000+00:00
gender : "Male"
password : "$2a$10$FVuFcd3jB8XNITRbbk/tnuHkwRiPQX0siyyLgiA..03Ij8HNkReXq"
role : "Doctor"
doctorDepartment : "Cardiology"
docAvatar : Object
```



GET https://res.cloudinary.com/dzydf6v1j/image/upload/v1726422561/o8wqqeco2kg28xuvnj1v.png

http://localhost:4000/api/v1/user/doctors get request

```
{
    "success": true,
```

`

```
    "doctors": [
        {
            "docAvatar": {
                "public_id": "bcqcvfw0nzwwrxfljnll",
                "url":
"https://res.cloudinary.com/dzydf6v1j/image/upload/v1726422126/bcqcvfw0nzwwrxfljnll.png"
            },
            "_id": "66e71c72f12e57ab80143a22",
            "firstName": "Shankar",
            "lastName": "Medipally",
            "email": "shankarmedipally@gmail.com",
            "phone": "9878654321",
            "nic": "1234567891234",
            "dob": "2000-08-08T18:30:00.000Z",
            "gender": "Male",
            "role": "Doctor",
            "doctorDepartment": "Cardiology",
            "__v": 0
        },
        {
            "docAvatar": {
                "public_id": "o8wqqeco2kg28xuvnj1v",
                "url":
"https://res.cloudinary.com/dzydf6v1j/image/upload/v1726422561/o8wqqeco2kg28xuvnj1v.png"
            },
            "_id": "66e71e25f12e57ab80143a26",
            "firstName": "Ishitha",
            "lastName": "Lol",
            "email": "ishitha@gmail.com",
            "phone": "9878654321",
            "nic": "1234567891234",
            "dob": "2000-08-08T18:30:00.000Z",
            "gender": "Female",
            "role": "Doctor",
            "doctorDepartment": "Pediatrition",
            "__v": 0
        }
    ]
}
```

`

Donno why my api key changed in cloudinary, so i changed in config.env

```
PORT = 4000
MONGO_URL =
mongodb+srv://ugs21016csetriveni:MiKTsm88PDHJ2vrr@cluster0.5vqfu.mongodb.n
et/?retryWrites=true
DASHBOARD_URL= https://localhost:5174
FRONTEND_URL = http://localhost:5173
JWT_SECRET_KEY = THISISMERNPROJECT
JWT_EXPIRES = 7d
COOKIE_EXPIRE = 7
CLOUDINARY_CLOUD_NAME = dzydf6v1j
CLOUDINARY_API_SECRET =  62UOi_fYz3OHCtxCvluf8tXa7hc
CLOUDINARY_API_KEY = 685962987978646
```

```
// UeyXv8KogFR0XstoR0phDDhZQOw
```

Old Key

Adding this in messageController.js

```
export const getAllMessages = catchAsyncErrors(async (req, res, next) => {
  const messages = await Message.find();
  res.status(200).json({
    success: true,
    messages,
  });
});
```

messageRouter.js only Admin can see messages : once Admin logs in only

```
import express from 'express'
import { getAllMessages, sendMessage } from
'../controller/messageController.js'
import { isAdminAuthenticated } from '../middlewares/auth.js'
const router = express.Router()
router.post("/send",sendMessage)
router.get("/getall",isAdminAuthenticated,getAllMessages)
export default router
```

If delete adminToken and do this isAdminAuthenticated stuff→ wont work

`

http://localhost:4000/api/v1/message/getall   get request

```
{
    "success": true,
    "messages": [
        {
            "_id": "66d464c9f1c5661e6aded998",
            "firstName": "Triveni",
            "lastName": "Pilla",
            "email": "triveni@gmail.com",
            "phone": "9867543327",
            "message": "Hey, Good Morning !",
            "__v": 0
        },
        {
            "_id": "66d465e870a4846fa7f84367",
            "firstName": "Karthik",
            "lastName": "Kanda",
            "email": "karthik@gmail.com",
            "phone": "8767543325",
            "message": "Good, Good Good !",
            "__v": 0
        },
        {
            "_id": "66d46cb459520296cbfb3f80",
            "firstName": "Koushik",
            "lastName": "Killop",
            "email": "koushik@gmail.com",
            "phone": "8976543346",
            "message": "I would like to know more about your products",
            "__v": 0
        },
        {
            "_id": "66e67904e5c7b8427e30de0b",
            "firstName": "Shiva",
            "lastName": "Kumar",
            "email": "shivakumar@gmail.com",
            "phone": "9089098976",
            "message": "Good Morning, Can i purchase 2 products at a time ?",
            "__v": 0
        }
```

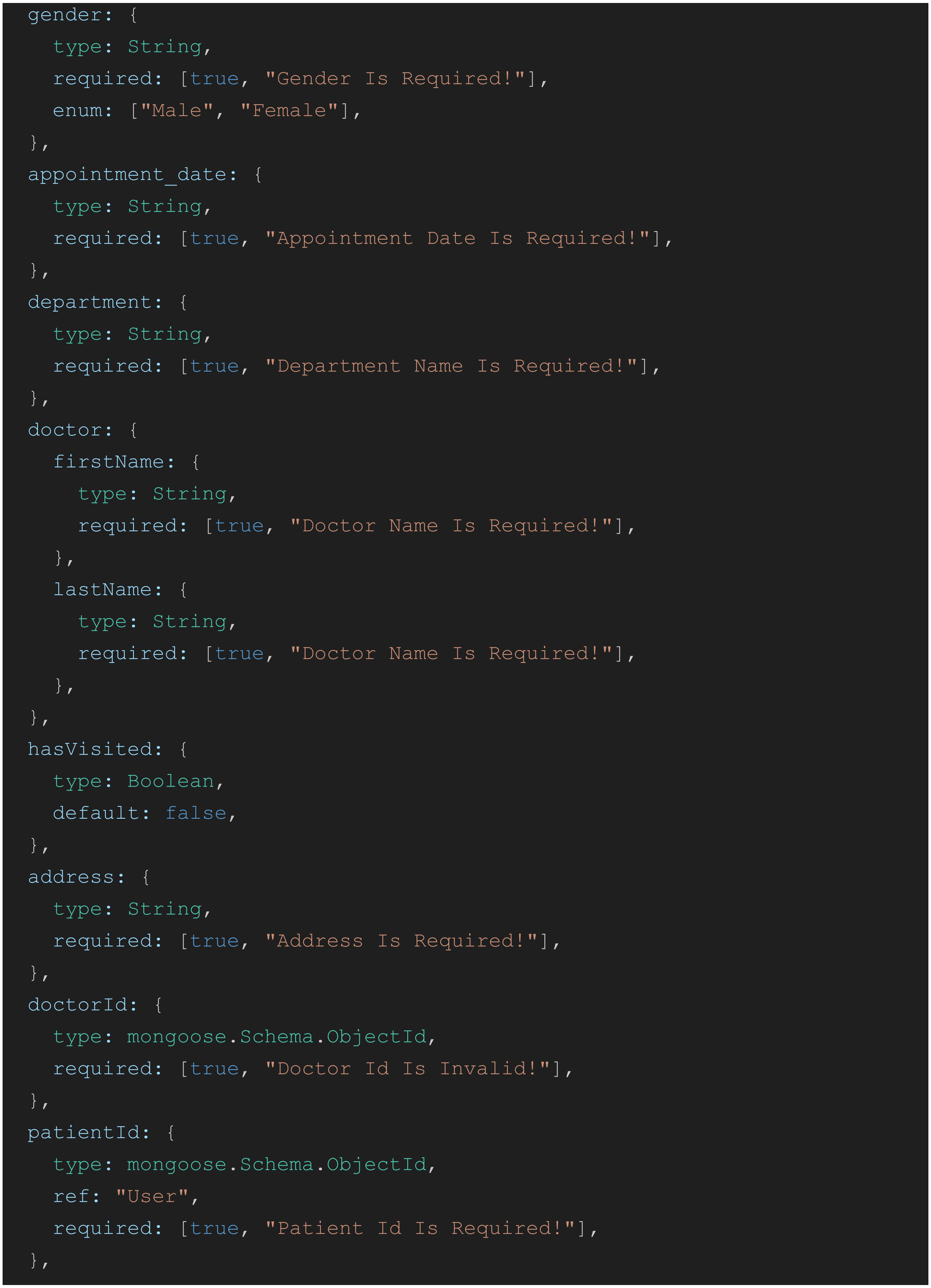
`

```
    ]
}
```

AppointmentSchema.js in models folder

```
import mongoose from "mongoose";
import { Mongoose } from "mongoose";
import validator from "validator";

const appointmentSchema = new mongoose.Schema({
  firstName: {
    type: String,
    required: [true, "First Name Is Required!"],
    minLength: [3, "First Name Must Contain At Least 3 Characters!"],
  },
  lastName: {
    type: String,
    required: [true, "Last Name Is Required!"],
    minLength: [3, "Last Name Must Contain At Least 3 Characters!"],
  },
  email: {
    type: String,
    required: [true, "Email Is Required!"],
    validate: [validator.isEmail, "Provide A Valid Email!"],
  },
  phone: {
    type: String,
    required: [true, "Phone Is Required!"],
    minLength: [10, "Phone Number Must Contain Exact 10 Digits!"],
    maxLength: [10, "Phone Number Must Contain Exact 10 Digits!"],
  },
  nic: {
    type: String,
    required: [true, "NIC Is Required!"],
    minLength: [13, "NIC Must Contain Only 13 Digits!"],
    maxLength: [13, "NIC Must Contain Only 13 Digits!"],
  },
  dob: {
    type: Date,
    required: [true, "DOB Is Required!"],
  },
```

`



```
  gender: {
    type: String,
    required: [true, "Gender Is Required!"],
    enum: ["Male", "Female"],
  },
  appointment_date: {
    type: String,
    required: [true, "Appointment Date Is Required!"],
  },
  department: {
    type: String,
    required: [true, "Department Name Is Required!"],
  },
  doctor: {
    firstName: {
      type: String,
      required: [true, "Doctor Name Is Required!"],
    },
    lastName: {
      type: String,
      required: [true, "Doctor Name Is Required!"],
    },
  },
  hasVisited: {
    type: Boolean,
    default: false,
  },
  address: {
    type: String,
    required: [true, "Address Is Required!"],
  },
  doctorId: {
    type: mongoose.Schema.ObjectId,
    required: [true, "Doctor Id Is Invalid!"],
  },
  patientId: {
    type: mongoose.Schema.ObjectId,
    ref: "User",
    required: [true, "Patient Id Is Required!"],
  },
```

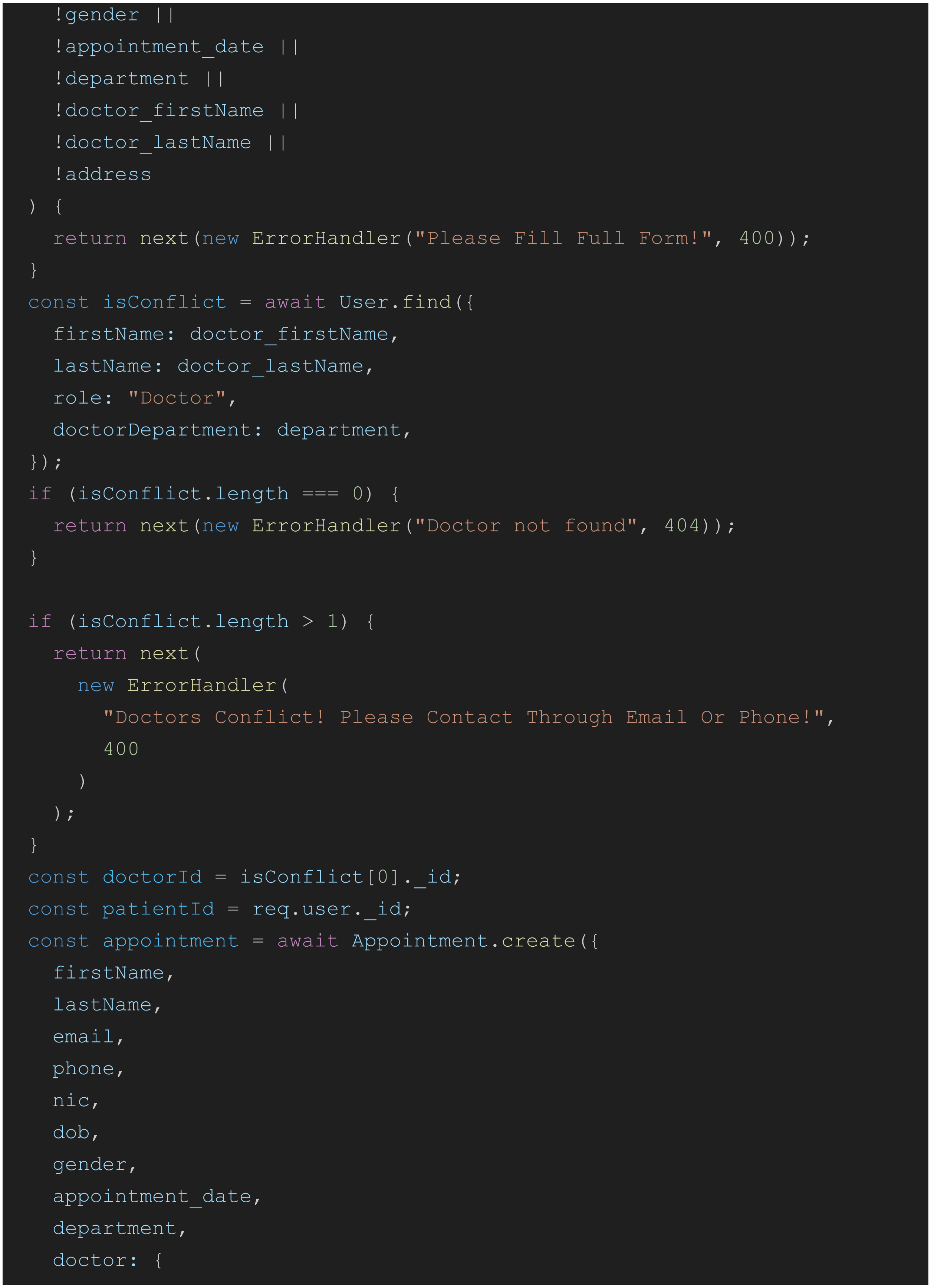
`

```
  status: {
    type: String,
    enum: ["Pending", "Accepted", "Rejected"],
    default: "Pending",
  },
});

export const Appointment = mongoose.model("Appointment",
appointmentSchema);
```

Appointment controller.js

```
import { catchAsyncErrors } from "../middlewares/catchAsyncErrors.js";
import ErrorHandler from "../middlewares/error.js";
import { Appointment } from "../models/appointmentSchema.js";
import { User } from "../models/userSchema.js";

export const postAppointment = catchAsyncErrors(async (req, res, next) =>
{
  const {
    firstName,
    lastName,
    email,
    phone,
    nic,
    dob,
    gender,
    appointment_date,
    department,
    doctor_firstName,
    doctor_lastName,
    hasVisited,
    address,
  } = req.body;
  if (
    !firstName ||
    !lastName ||
    !email ||
    !phone ||
    !nic ||
    !dob ||
```

`



```
  !gender ||
  !appointment_date ||
  !department ||
  !doctor_firstName ||
  !doctor_lastName ||
  !address
) {
  return next(new ErrorHandler("Please Fill Full Form!", 400));
}
const isConflict = await User.find({
  firstName: doctor_firstName,
  lastName: doctor_lastName,
  role: "Doctor",
  doctorDepartment: department,
});
if (isConflict.length === 0) {
  return next(new ErrorHandler("Doctor not found", 404));
}

if (isConflict.length > 1) {
  return next(
    new ErrorHandler(
      "Doctors Conflict! Please Contact Through Email Or Phone!",
      400
    )
  );
}
const doctorId = isConflict[0]._id;
const patientId = req.user._id;
const appointment = await Appointment.create({
  firstName,
  lastName,
  email,
  phone,
  nic,
  dob,
  gender,
  appointment_date,
  department,
  doctor: {
```

`

```
      firstName: doctor_firstName,
      lastName: doctor_lastName,
    },
    hasVisited,
    address,
    doctorId,
    patientId,
  });
  res.status(200).json({
    success: true,
    appointment,
    message: "Appointment Send!",
  });
});

export const getAllAppointments = catchAsyncErrors(async (req, res, next)
=> {
  const appointments = await Appointment.find();
  res.status(200).json({
    success: true,
    appointments,
  });
});
export const updateAppointmentStatus = catchAsyncErrors(
  async (req, res, next) => {
    const { id } = req.params;
    let appointment = await Appointment.findById(id);
    if (!appointment) {
      return next(new ErrorHandler("Appointment not found!", 404));
    }
    appointment = await Appointment.findByIdAndUpdate(id, req.body, {
      new: true,
      runValidators: true,
      useFindAndModify: false,
    });
    res.status(200).json({
      success: true,
      message: "Appointment Status Updated!",
    });
  }
```

`

```
);
export const deleteAppointment = catchAsyncErrors(async (req, res, next)
=> {
  const { id } = req.params;
  const appointment = await Appointment.findById(id);
  if (!appointment) {
    return next(new ErrorHandler("Appointment Not Found!", 404));
  }
  await appointment.deleteOne();
  res.status(200).json({
    success: true,
    message: "Appointment Deleted!",
  });
});
```

Postman : http://localhost:4000/api/v1/appointment/post

```
{
       "firstName": "Sandeep",
        "lastName": "kavi",
        "email": "sandeepkavi@gmail.com",
        "phone": "9878887777",
        "nic": "1234567891234",
        "dob": "11/08/2003",
        "gender": "Male",
         "appointment_date": "1234",
            "department": "Cardiology",
            "doctor_firstName": "Shankar",
            "doctor_lastName": "Medipally",
            "address": "Address"


}
```

```
{
    "success": true,
    "appointment": {
        "firstName": "Sandeep",
        "lastName": "kavi",
        "email": "sandeepkavi@gmail.com",
        "phone": "9878887777",
        "nic": "1234567891234",
```

`

```
        "dob": "2003-11-07T18:30:00.000Z",
        "gender": "Male",
        "appointment_date": "1234",
        "department": "Cardiology",
        "doctor": {
            "firstName": "Shankar",
            "lastName": "Medipally"
        },
        "hasVisited": false,
        "address": "Address",
        "doctorId": "66e71c72f12e57ab80143a22",
        "patientId": "66e679b0aa074797be0bae02",
        "status": "Pending",
        "_id": "66e7b6de5aa297d98142b415",
        "__v": 0
    },
    "message": "Appointment Send!"
}
```

http://localhost:4000/api/v1/appointment/getall

Returns all records of appointments

Put request: http://localhost:4000/api/v1/appointment/update/66e7b8945aa297d98142b41b

Pass the record by updating , send its object id in url

```
{
    "success": true,
    "message": "Appointment Status Updated!"
}
```

Delete request: http://localhost:4000/api/v1/appointment/delete/66e7b8945aa297d98142b41b

```
{
    "success": true,
    "message": "Appointment Deleted!"
}
```

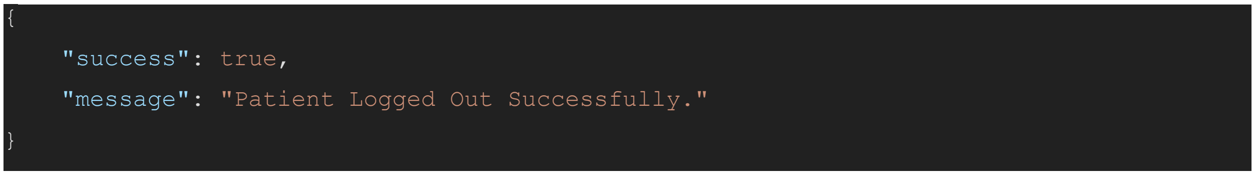
But problem is any patient login, then any patient can send appointments

http://localhost:4000/api/v1/user/patient/logout   get request

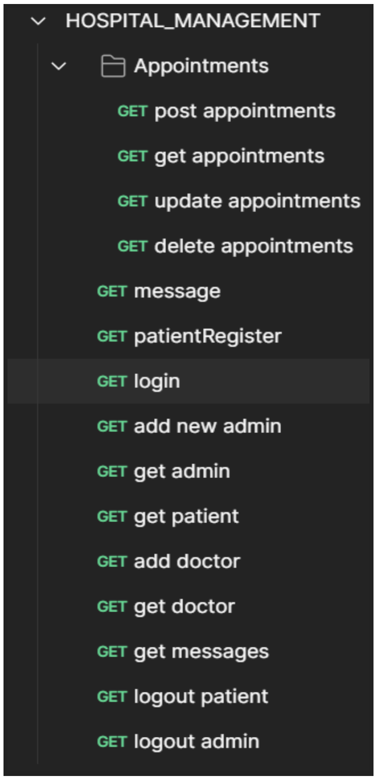
`

http://localhost:4000/api/v1/user/admin/logout

Cookies are deleted



```
{
    "success": true,
    "message": "Patient Logged Out Successfully."
}
```



HOSPITAL_MANAGEMENT

Appointments

GET post appointments

GET get appointments

GET update appointments

GET delete appointments

GET message

GET patientRegister

GET login

GET add new admin

GET get admin

GET get patient

GET add doctor

GET get doctor

GET get messages

GET logout patient

GET logout admin

Frontend
PS D:\Hospital-management> npm create vite@latest
Npm install
 npm i axios react-multi-carousel react-icons react-router-dom react-toastify

Npm run dev

Replacing app.css code with own code
Delete assets folder
Remove code of app.jsx and do rafce enter
Delete index.css file and its import in main.jsx

Create dashboard folder
Npm create vite@latest ./

`

Npm install

Del index.css, change app.css, del assets folder

```
 npm i axios react-icons react-router-dom react-toastify
```