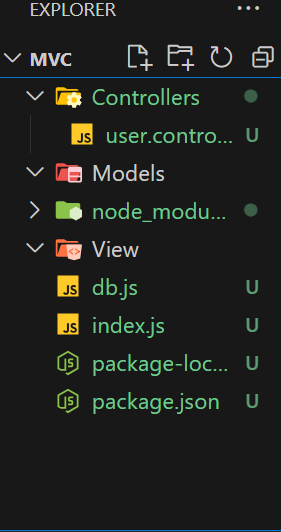
## Page:1

# MVC PROJECT



## Page:2

# Index.js

const express = require("express");

const { connection } = require("./db"); //destucture of connection

const userRouter = require("./Controllers/user.controller");

const app = express();

app.use(express.json()); //connect with frontend add cors

//user Router

app.use("/user", userRouter); //using MiddleWare

//server running

app.listen(8080, async () => {

try {

await connection;

console.log("server is running on port 8080");

console.log(">>>>>>>>>>>>>>>>connected to db>>>>>>>>>>>>>>>");

} catch (error) {

console.log(error);

}

});

# Package.json

{

  "name": "mvc",

  "version": "1.0.0",

## Page:3

  "description": "",

  "main": "index.js",

  "scripts": {

    "test": "echo \"Error: no test specified\" && exit 1",

    "server": "nodemon index.js"

  },

  "keywords": [],

  "author": "",

  "license": "ISC",

  "dependencies": {

    "express": "^4.21.0",

    "mongoose": "^8.6.3",

    "nodemon": "^3.1.5"

  }

}

# Db.js

const mongoose = require("mongoose");

const connection = mongoose.connect("mongodb://127.0.0.1:27017/nupurDB");

//schema ->Blue Print

const userScheme = new mongoose.Schema({

name: String,

age: Number,

city: String,

});

const UserModel = mongoose.model("user", userScheme);

## Page:4

module.exports = { connection, UserModel };

# controller

# usercontoller.js

const express = require("express");

const { UserModel } = require("../db"); //destucture of connection

const userRouter = express.Router();

userRouter.get("/get", async (req, res) => {

try {

const alluserdata = await UserModel.find();

console.log(alluserdata);

if (alluserdata) {

res.status(200).json({ userdata: alluserdata });

} else {

res.status(404).json({ message: "no user found" });

}

} catch (error) {

res.status(404).json({ message: error.message });

}

});

//post part

userRouter.post("/addproduct", async (req, res) => {

// console.log(req.body);

try {

const singleuser = new UserModel(req.body);

await singleuser.save();

## Page:5

res.status(200).json({ message: "User Created Successfully" });

} catch (error) {

res.status(404).json({ message: error.message });

}

});

//delete part

userRouter.delete("/delete/:id", async (req, res) => {

const { id } = req.params;

console.log(id);

try {

await UserModel.findByIdAndDelete(id);

res.status(200).json({ message: "Deleted...." });

} catch (error) {

res.status(404).json({ message: error.message });

}});

//patch part

userRouter.patch("/update/:id", async (req, res) => {

const { id } = req.params;

try {

await UserModel.findByIdAndUpdate(id, req.body);

res.status(200).json({ message: "Updated...." });

} catch (error) {

res.status(404).json({ message: error.message });

}});

module.exports = userRouter;

## Page:6

# Explanation

1. Express and Middleware

Express is a popular framework for building web applications using Node.js. It simplifies routing and handling HTTP requests.

Setting up Express:

js

Copy code

const express = require("express");

const app = express();

express() initializes the Express application.

Middleware:

js

Copy code

app.use(express.json());

This middleware parses incoming JSON requests so that you can access req.body in your routes.

2. Database Connection (Mongoose)

Mongoose is a library that allows us to interact with MongoDB using schemas and models.

Connecting to MongoDB:

js

Copy code

const mongoose = require("mongoose");

const connection = mongoose.connect("mongodb://127.0.0.1:27017/nupurDB");

mongoose.connect() connects your application to a MongoDB database. Here, it's connecting to the nupurDB database running on the local machine.

Schema and Model:

js

## Page:7

Copy code

const userSchema = new mongoose.Schema({

name: String,

age: Number,

city: String,

});

const UserModel = mongoose.model("user", userSchema);

mongoose.Schema defines the structure (blueprint) of the data (i.e., the fields and their types).

UserModel is a model created from the schema, which allows us to interact with the "user" collection in MongoDB.

3. Routing

Express uses routes to handle different HTTP methods like GET, POST, PATCH, and DELETE.

Importing and using routers:

js

Copy code

const userRouter = require("./Controllers/user.controller");

app.use("/user", userRouter);

userRouter is a router that handles routes related to users. The app.use("/user", userRouter) connects all routes in userRouter under the /user path.

4. User Router (Controller)

The userRouter defines routes for CRUD (Create, Read, Update, Delete) operations related to user data.

4.1 GET: Retrieve User Data

js

Copy code

userRouter.get("/get", async (req, res) => {

try {

const alluserdata = await UserModel.find();

## Page:8

res.status(200).json({ userdata: alluserdata });

} catch (error) {

res.status(404).json({ message: error.message });

}

});

GET /user/get: This route retrieves all the users from the MongoDB database using UserModel.find().

It returns the user data in the response if found, or a 404 error if something goes wrong.

4.2 POST: Create a New User

js

Copy code

userRouter.post("/addproduct", async (req, res) => {

try {

const singleuser = new UserModel(req.body);

await singleuser.save();

res.status(200).json({ message: "User Created Successfully" });

} catch (error) {

res.status(404).json({ message: error.message });

}

});

POST /user/addproduct: This route creates a new user. The user data is passed in the request body (req.body), saved to MongoDB, and a success message is returned.

4.3 DELETE: Remove a User by ID

js

Copy code

userRouter.delete("/delete/:id", async (req, res) => {

const { id } = req.params;

try {

await UserModel.findByIdAndDelete(id);

res.status(200).json({ message: "Deleted...." });

## Page:9

} catch (error) {

res.status(404).json({ message: error.message });

}

});

DELETE /user/delete/:id: This route deletes a user by its ID (req.params.id). If the deletion is successful, a confirmation message is returned.

4.4 PATCH: Update a User by ID

js

Copy code

userRouter.patch("/update/:id", async (req, res) => {

const { id } = req.params;

try {

await UserModel.findByIdAndUpdate(id, req.body);

res.status(200).json({ message: "Updated...." });

} catch (error) {

res.status(404).json({ message: error.message });

}

});

PATCH /user/update/:id: This route updates a user by its ID using the data from the request body (req.body).

5. Starting the Server

js

Copy code

app.listen(8080, async () => {

try {

await connection;

console.log("server is running on port 8080");

console.log(">>>>>>>>>>>>>>>>connected to db>>>>>>>>>>>>>>>");

} catch (error) {

console.log(error);

## Page:10

}});

This code starts the Express server on port 8080. Before the server starts, it waits for the database connection (await connection). If the connection is successful, it logs the messages.

Key Concepts

Express: Handles the server setup, routing, and middleware.

Mongoose: Manages the MongoDB connection and provides an abstraction for handling database operations via schemas and models.

CRUD Operations: Create, Read, Update, and Delete operations on user data using appropriate HTTP methods like POST, GET, PATCH, and DELETE.

Let me know if you'd like further clarification on any of these sections!