I am working on a nextjs project. I want to build a backed by using next api. here is my router file. file path is :

---src/app/api/users/route.ts---

here is code example ```import { ObjectId } from 'mongodb';

import { client } from '../lib/dbConnect';

// ! get all data from db

export async function GET() {

await client.connect();

const db = client.db('db\_users');

const collection = db.collection('users\_collection');

// Retrieve all documents

const cursor = collection.find();

// Print all documents

const allDocs = await cursor.toArray();

console.log(allDocs);

const data = allDocs;

const result = { data, message: 'get request invoked successfully' };

return new Response(JSON.stringify(result));

}

// ! add a new data

export async function POST(req: Request) {

// await dbConnect();

console.log('post route hit');

const resultData = await req.json();

console.log('resultData ', resultData);

await client.connect();

const db = client.db('db\_users');

const collection = db.collection('users\_collection');

const resultFromDB = await collection.insertOne(resultData);

console.log('post route hit result : ');

return new Response(

JSON.stringify({

data: resultData,

result: resultFromDB,

message: 'Post request successful invoke',

}),

);

}

// ! update a data

/\* it just update data, if not found then do nothing, but it can not change all data \*/

export async function PUT(req: Request) {

const updateUser = await req.json();

console.log('updateUser.id ', updateUser.id);

console.log('updateUser.user ', updateUser.user);

await client.connect();

// Access the database and collection

const database = client.db('db\_users');

const collection = database.collection('users\_collection');

// Specify the document to update using its \_id

const id = updateUser.id; // Replace with the actual \_id value

console.log('--id ', id);

const filter = { \_id: new ObjectId(id) };

// Define the update operation

const updateDocument = {

$set: {

user: updateUser.user, // Replace with the actual field and value to update

},

};

// Perform the update

const result = await collection.updateOne(filter, updateDocument);

console.log(

`\*\*\* --- ${result.matchedCount} document(s) matched the filter, updated ${result.modifiedCount} document(s)`,

);

return new Response(

JSON.stringify({

data: updateUser,

result: updateUser,

message: 'Put request successful invoke',

}),

);

}

// ! delete a data

export async function DELETE(req: Request) {

const updateUser = await req.json();

await client.connect();

// Access the database and collection

const database = client.db('db\_users');

const collection = database.collection('users\_collection');

// Delete a single document

const id = updateUser.id; // Replace with the actual \_id value

console.log('--id ', id);

const filter = { \_id: new ObjectId(id) };

const deleteOneResult = await collection.deleteOne(filter);

console.log(`Deleted ${deleteOneResult.deletedCount} document(s)`);

return new Response(

JSON.stringify({

data: updateUser,

result: deleteOneResult,

message: 'Patch request successful invoke',

}),

);

}

```

---src/app/api/lib/dbConnect.ts---

here is code example : ```import { MongoClient, ServerApiVersion } from 'mongodb';

const uri = `mongodb+srv://${process.env.dbusername}:${process.env.dbpassword}@rupushiv1.aq2wh.mongodb.net/?retryWrites=true&w=majority&appName=rupushiv1`;

// Create a MongoClient with a MongoClientOptions object to set the Stable API version

export const client = new MongoClient(uri, {

  serverApi: {

    version: ServerApiVersion.v1,

    strict: true,

    deprecationErrors: true,

  },

});

```

---src/app/api/dbtest/route.ts---

here is code example : ```export async function GET() {

  const result = { header: 'header-data', message: 'get request invoked successfully' };

  return new Response(JSON.stringify(result));

}

```

Now concise the code. I want to implement it with mongoose. Now write the code with mongoose here is user modal ```

Here is user Schema ```const userSchema = new Schema({

name: {

type: String,

required: true

},

email: {

type: String,

required: true,

unique: true,

match: [/^\w+([\.-]?\w+)\*@\w+([\.-]?\w+)\*(\.\w{2,3})+$/, 'Please enter a valid email address']

},

password: {

type: String,

required: true

},

alias: {

type: String,

required: true

},

role: {

type: String,

required: true,

enum: ['user', 'admin', 'moderator'], // Add/remove roles as needed

default: 'user'

}

}, {

timestamps: true // Adds createdAt and updatedAt automatically

});

```