I-Notebook

1. Create an app

PS D:\Workspace\React> npx create-react-app inotebook

1. Create a folder named backend for node.js
2. Open terminal
3. Fire command and answer the questions

$npm init

1. Now install packages like express, mongoose and nodemon

$npm i express

$npm i mongoose

$npm i nodemon

1. Now creating models for user and notes
2. Create folder models
3. Create user model

//user.js

const mongoose = require('mongoose');

const Schema = mongoose.Schema;

const userSchema = new Schema({

    name: {

        type: String,

        required: true,

    },

    email: {

        type: String,

        required: true,

        unique: true,

    },

    password: {

        type: String,

        require: true,

    },

    time: {

        type: Date,

        default: Date.now

    },

});

module.exports = mongoose.model('user', userSchema);

1. Create notes model

//notes.js

const mongoose = require('mongoose');

const Schema = mongoose.Schema;

const notesSchema = new Schema({

    title: {

        type: String,

        required: true,

    },

    desciption: {

        type: String,

        required: true,

    },

    tag: {

        type: String,

        default: "general",

    },

    time: {

        type: Date,

        default: Date.now

    },

});

module.exports = mongoose.model('notes', notesSchema);

1. Now create a mongo connection

//db.js

const mongoose = require('mongoose');

const url = "mongodb://localhost:27017/"; // copy url from mongo compas

const connectToMongoose = () => {

    mongoose.connect(url).then(() => {

        console.log("connected to mangoose successfuly!");

    }).catch((err) => console.log(err));

}

module.exports = connectToMongoose;

1. Create index.js

//index.js

const express = require('express');

const connectToMongoose = require('./db');

const app = express();

const port = 3000;

connectToMongoose();

app.get('/', () => {

    console.log("hello");

})

app.listen(port, () => {

    console.log(`app is listening at port http://localhost:${port}`)

});

1. Now add a script to run file index.js edit package.json

"scripts": {

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

    "start": "nodemon index.js" //this is means ‘npm start’ to run project

  },

1. Now lets create a routers for authentication and notes
2. Routers/notes.js

//notes.js

const express = require('express');

const router = express.Router();

router.get('/', (req, res) => {

    res.send([]);

});

module.exports = router;

1. Routers/auth.js

//auth.js

const express = require('express');

const router = express.Router();

router.get('/', (req, res) => {

    obj = {

        name: 'suraj',

        password: 'khot'

    }

    res.send(obj);

});

module.exports = router;

1. Now add routers to a=index.js files

//index.js

const express = require('express');

const connectToMongoose = require('./db');

const app = express();

const port = 3000;

connectToMongoose();

// routers

app.use('/api/auth',require('./routers/auth'));

app.use('/api/notes',require('./routers/notes'));

app.listen(port, () => {

    console.log(`app is listning at port http://localhost:${port}`)

});

1. now start server

PS D:\Workspace\React\inotebook\Backend> npm start

> inotebook-backend@1.0.0 start

> nodemon index.js

[nodemon] 3.1.4

[nodemon] to restart at any time, enter `rs`

[nodemon] watching path(s): \*.\*

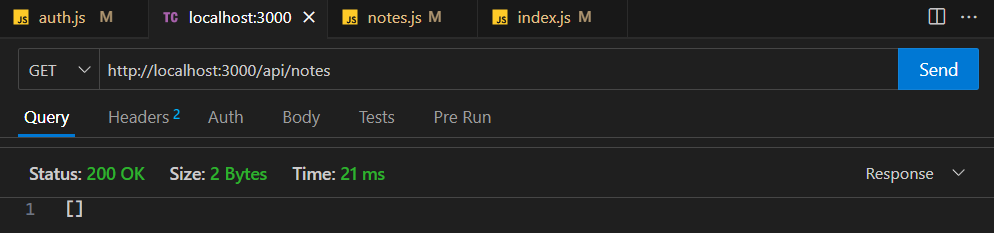
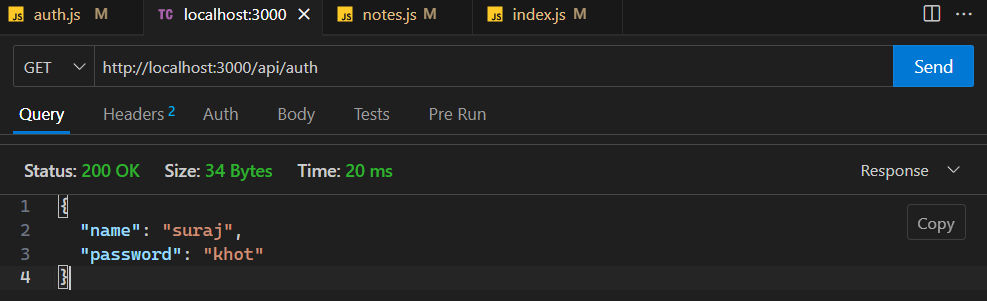
[nodemon] watching extensions: js,mjs,cjs,json

[nodemon] starting `node index.js`

app is listning at port http://localhost:3000

connected to mangoose successfuly!

Now check by using thunder client



1. now creating a router for accepting user using user schema(name,email,password)
2. auth.js

//auth.js

const express = require('express');

const router = express.Router();

const User = require('../models/User'); //import user schema

//sending post request and save user to db

router.post('/', (req, res) => {

//accepting request

    const request = req.body;

    const user = User(request);

    user.save();

    res.send(request);

});

module.exports = router;

1. now for that we require middleware instead we add one middleware in index.js

//index.js

//code

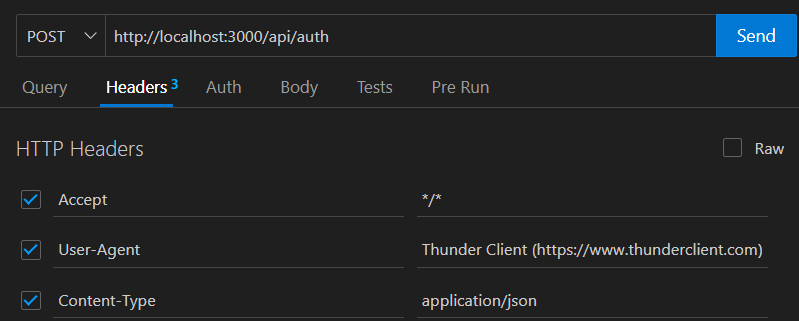
// middleware for accept json

app.use(express.json());

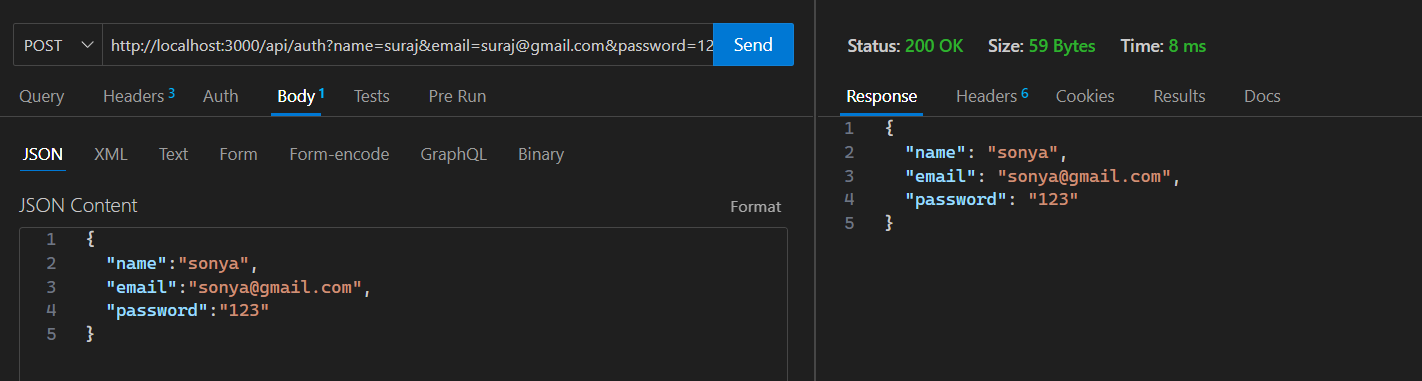
//code

1. now sending req through thunder client

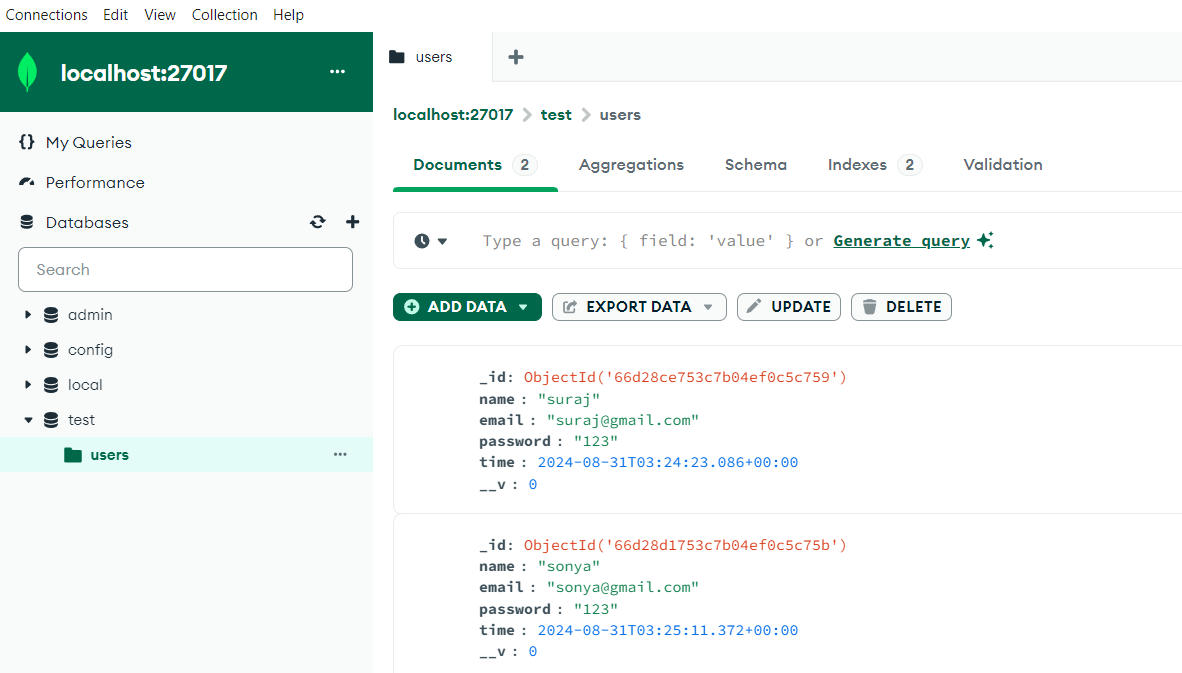
for that define content type as application/json in body and send



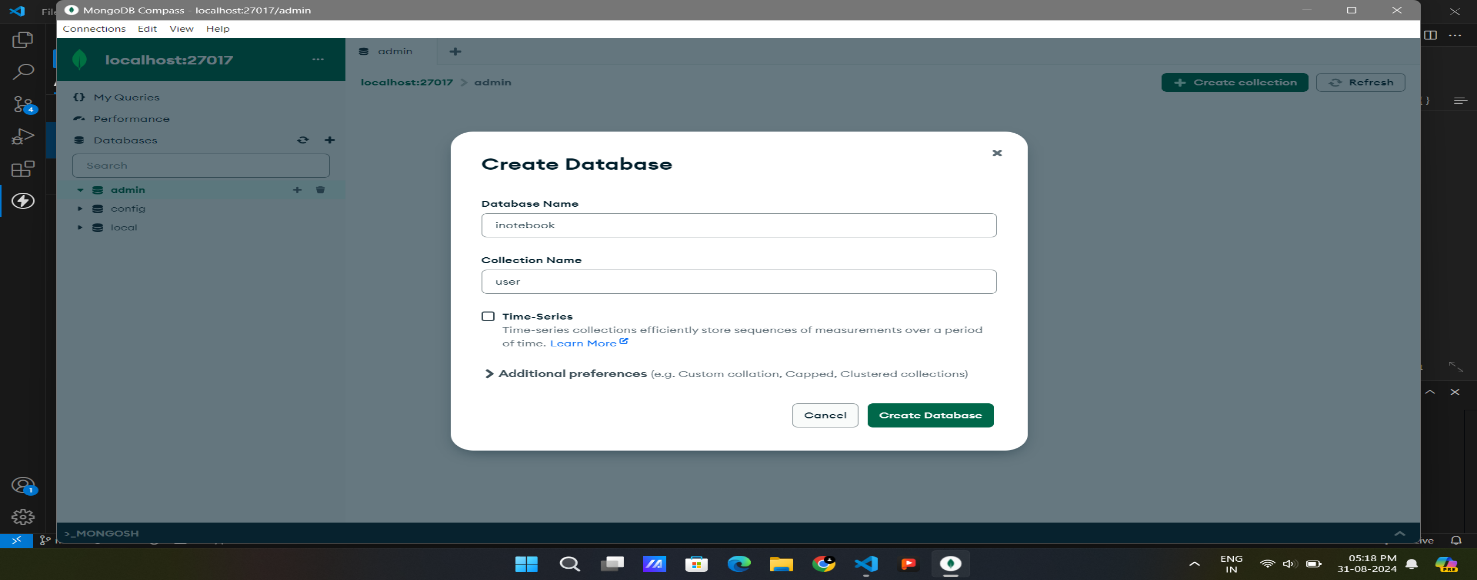
1. now send data



1. now see in mongoodb



1. now creating custom db in mongo and adding validations also saving user
2. create db in mongo



1. now change url in db.js

//db.js

const url = "mongodb://localhost:27017/inotebook";

//code

1. first thing is we do not want duplicate values(email)so that change our user schema logic

createIndexes : it will add indexes so that.. unique

const mongoose = require('mongoose');

const Schema = mongoose.Schema;

const userSchema = new Schema({

//code

});

const User = mongoose.model('user', userSchema);

User.createIndexes();

module.exports = User;

1. now use validator of express

url = <https://express-validator.github.io/docs/>

install it PS D:\Workspace\React\inotebook\Backend> npm install express-validator

const express = require('express');

const router = express.Router();

const User = require('../models/User');

const { validationResult, body } = require('express-validator');

router.post(

    '/',

    [

body('name', 'Enter a valid name').isLength({ min: 2 }),

        body('password', 'password must be at least 5 char').isLength({ min: 5 }),

        body('email', 'enter valid email').isEmail(),

    ],

    (req, res) => {

        const errors = validationResult(req);

        if (!errors.isEmpty()) {

            return res.status(400).json({ errors: errors.array() })

        }

        User.create({

            name: req.body.name,

            password: req.body.password,

            email: req.body.email

        }).then(user => res.json(user)).catch(err => res.json({ error: 'This email is already exists in database' }))

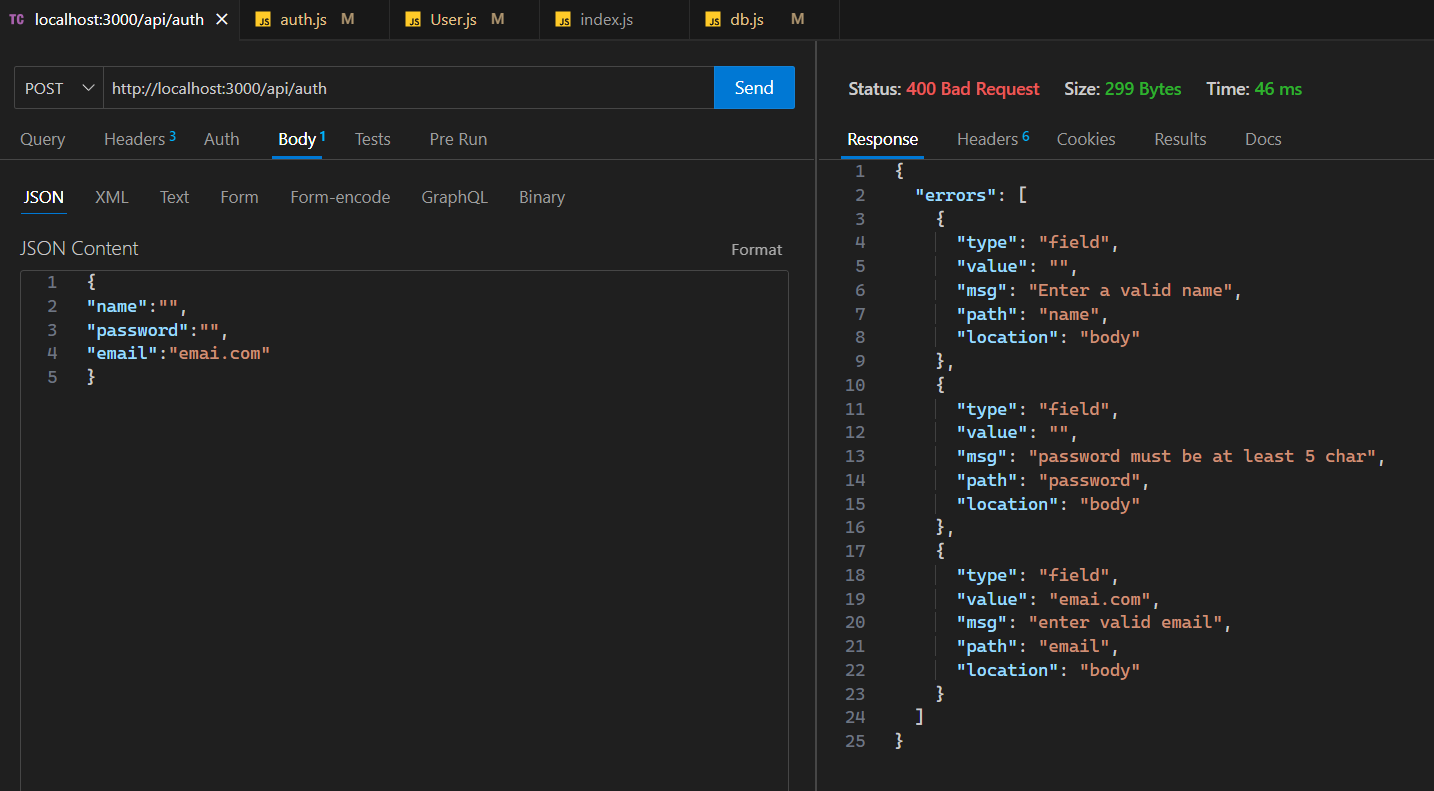
    });

module.exports = router;

/////

body('name', 'Enter a valid name').isLength({ min: 2 }), is like

body(‘key\_name’, ‘error value to display’).fn



///

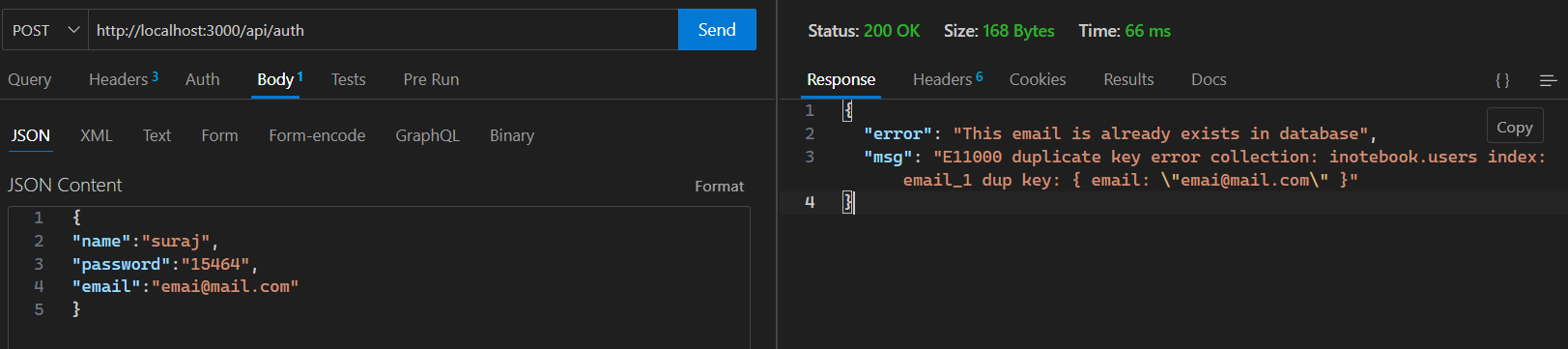
  const errors = validationResult(req);

        if (!errors.isEmpty()) {

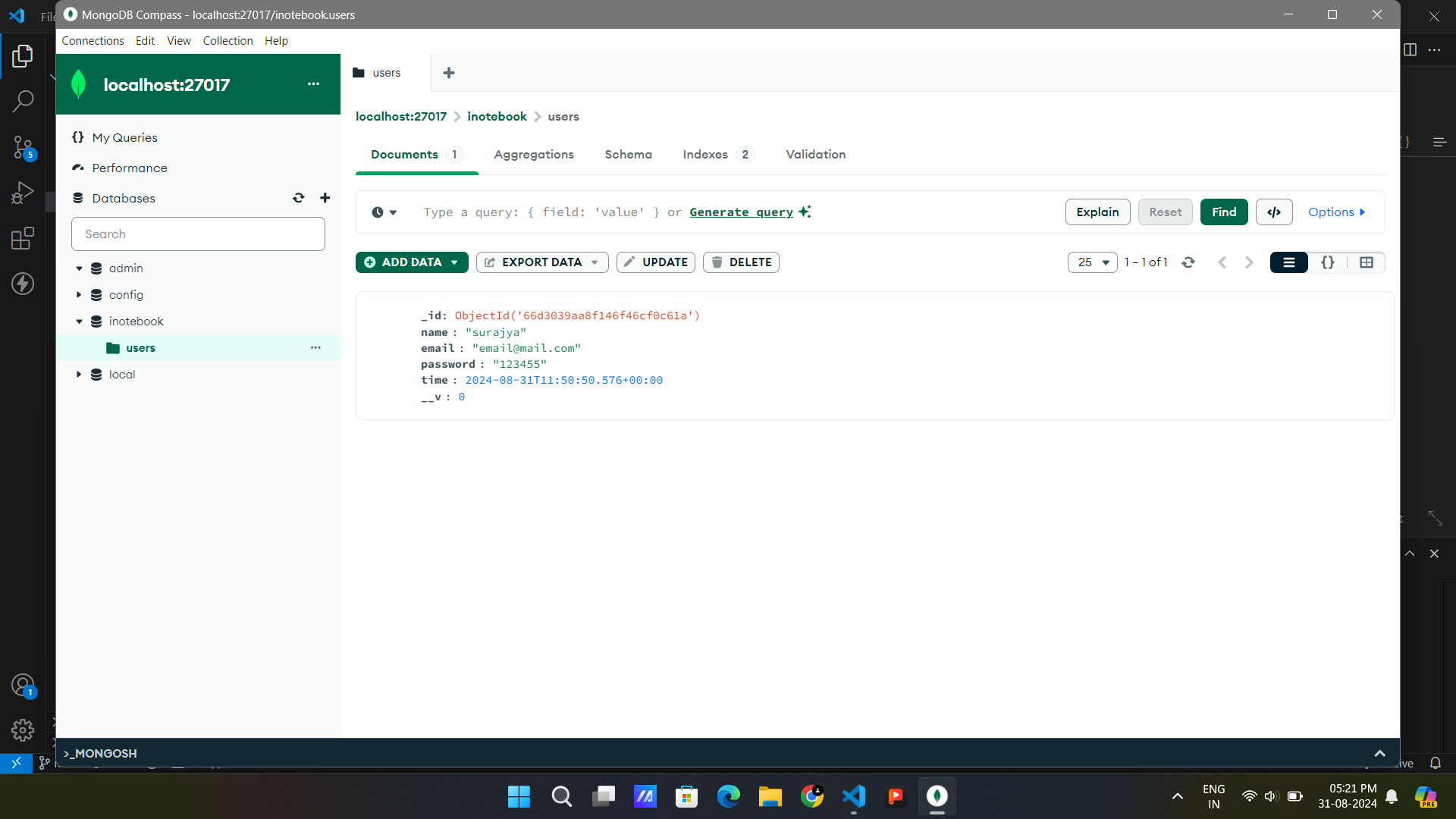
            return res.status(400).json({ errors: errors.array() })

        }

This lines are every time same in all pgm



1. output like



1. now refactoring code

chaning port cause our react app is listen to 3000

//index.js

const express = require('express');

const connectToMongoose = require('./db');

const app = express();

const port = 5000;

// middleware for accept json

app.use(express.json());

connectToMongoose();

// routers

app.use('/auth', require('./routers/auth'));

app.use('/notes', require('./routers/notes'));

app.listen(port, () => {

    console.log(`app is listning at port http://localhost:${port}`)

});

Adding async wait instead of promise

const express = require('express');

const router = express.Router();

const User = require('../models/User');

const { validationResult, body } = require('express-validator');

router.post(

    // url

    '/user',

    // validating body's parameters

    [

        body('name', 'Enter a valid name').isLength({ min: 2 }),

        body('password', 'password must be at least 5 char').isLength({ min: 5 }),

        body('email', 'enter valid email').isEmail(),

    ],

    // callback

    async (req, res) => {

        // Extracts the validation errors of an express request

        const errors = validationResult(req);

        if (!errors.isEmpty()) {

            return res.status(400).json({ errors: errors.array() })

        }

        try {

            // checking wheter email exists

            let user = await User.findOne({ email: req.body.email });

            // if exists 400

            if (user) {

                res.status(400).send("user with this email already exists in our database");

            }

            // else create new user

            else {

                user = await User.create({

                    name: req.body.name,

                    password: req.body.password,

                    email: req.body.email

                });

                // send user as res

                res.send(user);

            }

        } catch (error) {

            res.status(500).send("Some error is occured!");

        }

    });

module.exports = router;

1. now hashing password using pkg

PS D:\Workspace\React\inotebook\Backend> npm i bcryptjs

PS D:\Workspace\React\inotebook\Backend> npm i jsonwebtoken

//auth.js

const express = require('express');

const router = express.Router();

const User = require('../models/User');

const { validationResult, body } = require('express-validator');

const bcrypt = require('bcryptjs');

const jwt = require('jsonwebtoken');

const JWT\_SecureStr = "ILovePrograming";

router.post(

    // url

    '/user',

    // validating body's parameters

    [

        body('name', 'Enter a valid name').isLength({ min: 2 }),

        body('password', 'password must be at least 5 char').isLength({ min: 5 }),

        body('email', 'enter valid email').isEmail(),

    ],

    // callback

    async (req, res) => {

        // Extracts the validation errors of an express request

        const errors = validationResult(req);

        if (!errors.isEmpty()) {

            return res.status(400).json({ errors: errors.array() })

        }

        try {

            // checking wheter email exists

            let user = await User.findOne({ email: req.body.email });

            // if exists 400

            if (user) {

                res.status(400).send("user with this email already exists in our database")

            }

            // else create new user

            else {

                // securing pass

                // create a salt

                var salt = bcrypt.genSaltSync(10);

                // use bcrypt

                let securePassword = bcrypt.hashSync(req.body.password, salt);

                user = await User.create({

                    name: req.body.name,

                    password: securePassword,

                    email: req.body.email

                });

                // sending jwt token

                let uid = {

                    id: user.id,

                }

                var authToken = jwt.sign(uid, JWT\_SecureStr);

                res.send({ authToken });

            }

        } catch (error) {

            res.status(500).send("Some error is occured!");

        }

    });

module.exports = router;

1. now lets create a login to user
2. create a router

auth.js

//! register user

router.post(

//code

    });

//! login user

router.post(

    // url

    '/login',

    // validating body's parameters

    [

        body('password', 'password cannot be blank').exists(),

        body('email', 'enter valid email').isEmail(),

    ],

    // callback

    async (req, res) => {

        // Extracts the validation errors of an express request

        const errors = validationResult(req);

        if (!errors.isEmpty()) {

            return res.status(400).json({ errors: errors.array() });

        }

        // destructuring email pass from request

        const { email, password } = req.body;

        try {

            // checking wheter user exists

            let user = await User.findOne({ email });

            // if not exists 400

            if (!user) {

                return res.status(400).json({ error: "please enter correct credintials!" });

            }

            // checking req pass with hash stored in db

            let checkPass = await bcrypt.compare(password, user.password);

            // if password is incorrect

            if (!checkPass) {

                return res.status(400).json({ error: "please enter correct credintials!" });

            }

            // else sending jwt token

            let uid = {

                id: user.id,

            }

            var authToken = jwt.sign(uid, JWT\_SecureStr);

            res.json({ authToken });

        } catch (error) {

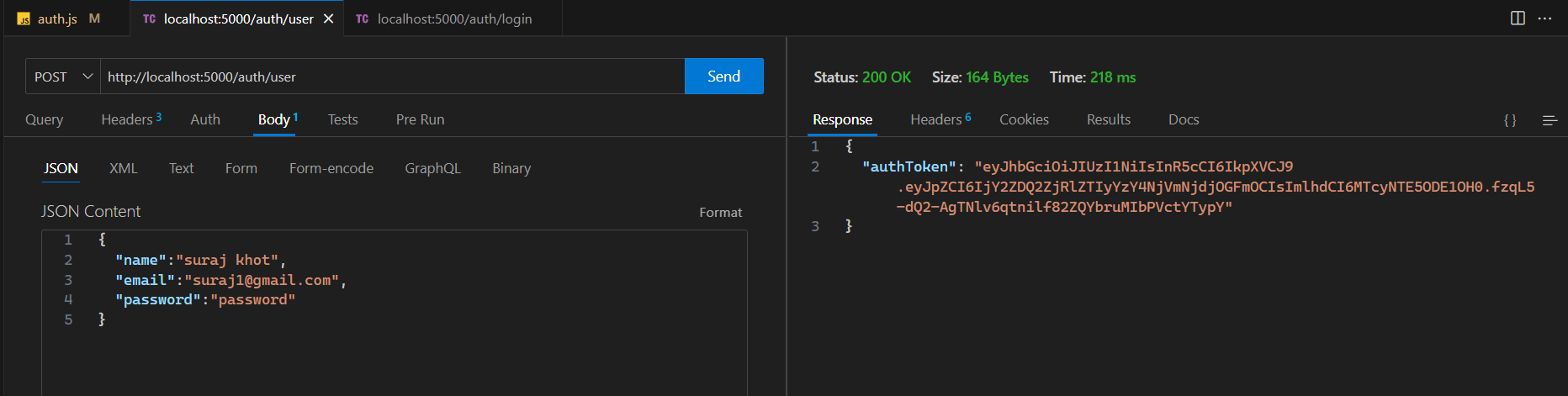
            res.status(500).send("Some error is occured!");

        }

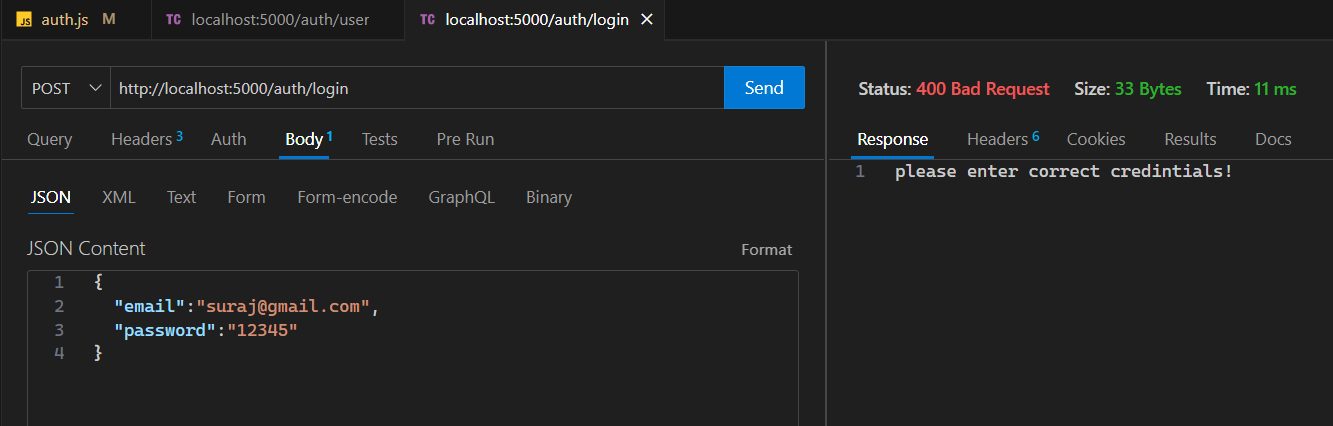
    });

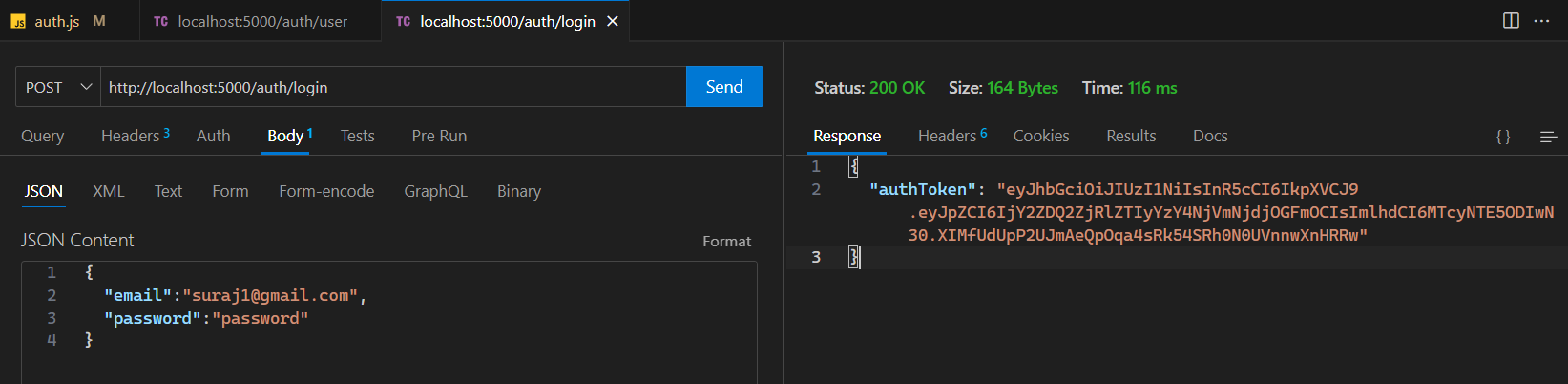
module.exports = router;

1. output
2. create user



1. now both wrong





1. now getting data after user will login:
2. create a middleware to verify token(we are sending token as user id + secure string while login or sign up)

//getuser.js

const jwt = require('jsonwebtoken');

const JWT\_SecureStr = "ILovePrograming";

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

    try {

        // getting token from header

        let token = req.header('auth-token');

        // if no token 400

        if (!token) {

            return res.status(400).send({ error: "please authenticate using token" });

        }

        // Synchronously verify given token using a secret or a public key to get a decoded token token

        //means getting data as userid+secure code

        const data = jwt.verify(token, JWT\_SecureStr);

        //get user id from data and send it as \*\*\*\*\*\*request to next fn

        req.userId = data.id;

        //call next then next function will call

        next();

    } catch (error) {

        res.status(500).send({ error: error.message });

    }

}

module.exports = getUserData;

1. now create a router for it

auth.js

//code

// !get user data

router.post('/getuser', getUserData, async (req, res) => {

    try {

        // getting user id from middleware request

        let uid = req.userId;

        // find the user by its id and send data except password

        let data = await User.findById(uid).select('-password');

        res.send(data);

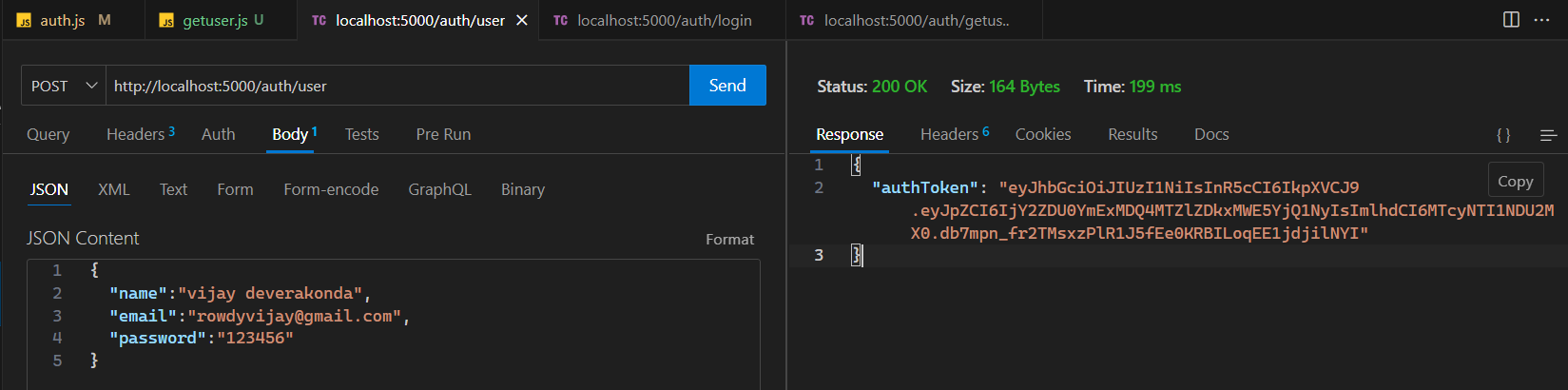
    } catch (error) {

        res.status(500).send("Some error is occured!");

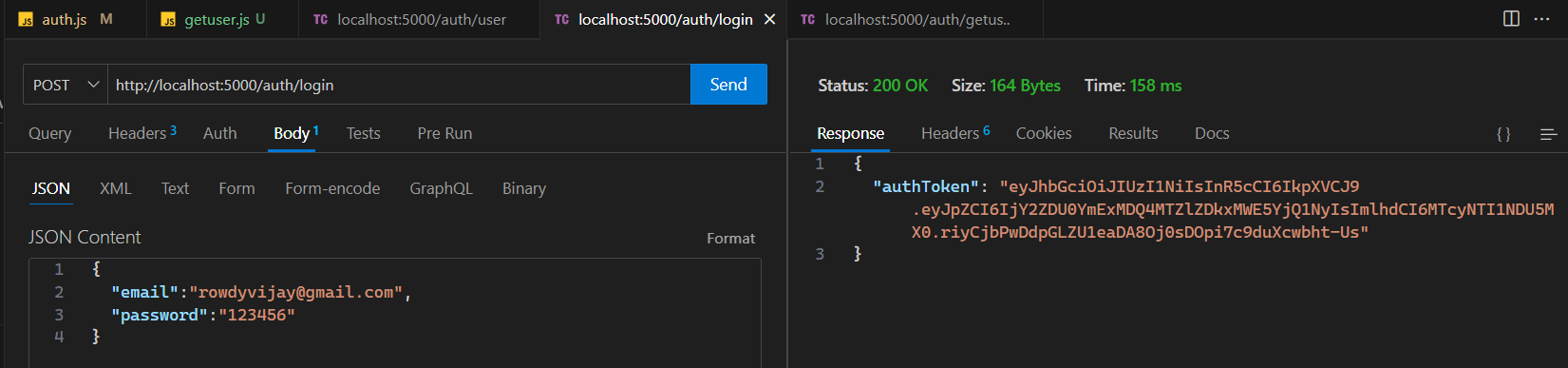
    }

});

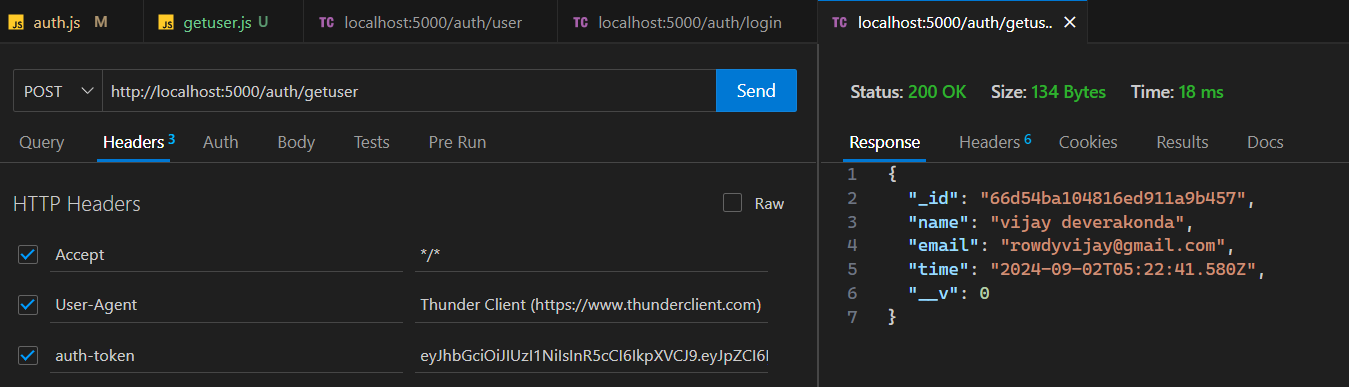
1. now verify output
2. create account



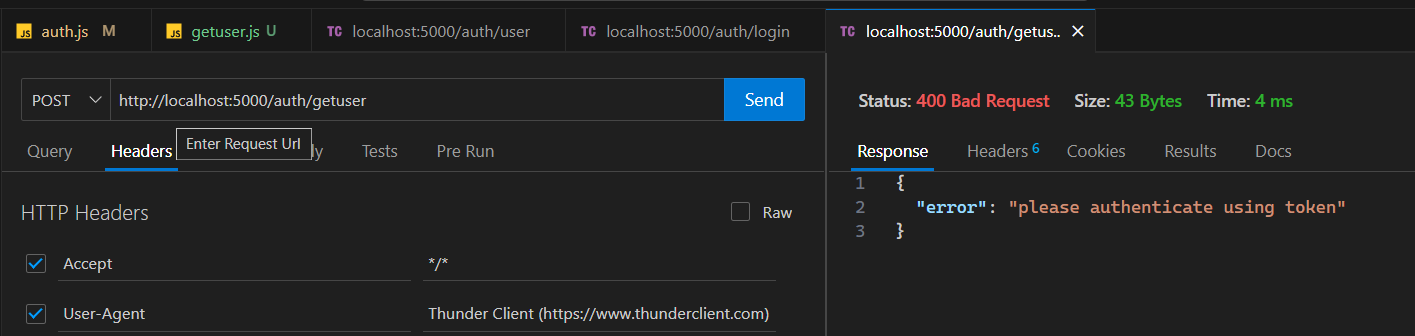
1. login



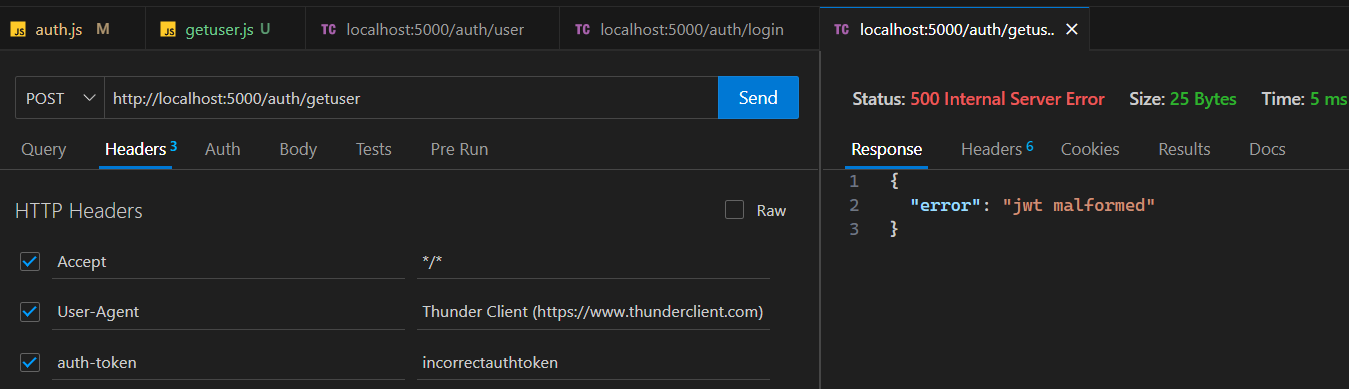
1. copy the token and send header as /getuser url



1. forgot to send auth token



1. sending incorrect token



1. now create and fetch note
2. update a model note to store user means its id

Notes.js

const notesSchema = new Schema({

    user: {

        type: mongoose.Schema.Types.ObjectId,//linking user model using foreign key

        ref: 'user',

    },

//code

1. now create route

//notes.js

const express = require('express');

const getUserData = require('../middleware/getuser');

const Notes = require('../models/Notes');

const router = express.Router();

const { validationResult, body } = require('express-validator');

//! getting note

router.get('/getnote', getUserData, async (req, res) => {

    try {

        // get user id from middl. request

        const uid = req.userId;

        let note = await Notes.find({ user: uid });

        res.send(note);

    } catch (error) {

        return res.status(500).send({ error: error.message });

    }

});

//! create note

router.post('/createnote', getUserData,

    // validating body's parameters

    [

        body('title', 'title at least 2 char').isLength({ min: 2 }),

        body('desciption', 'description at least 5 char').isLength({ min: 5 }),

    ],

    async (req, res) => {

        // Extracts the validation errors of an express request

        const errors = validationResult(req);

        if (!errors.isEmpty()) {

            return res.status(400).json({ errors: errors.array() })

        }

        // if condition matches

        try {

            // get user id from middl. request

            const uid = req.userId;

            // destructuring req body

            const { title, desciption, tag } = req.body;

            // create

            let note = new Notes({

                title,

                desciption,

                tag,

                user: uid,

            });

            const saved = await note.save();

            res.send({ saved })

        } catch (error) {

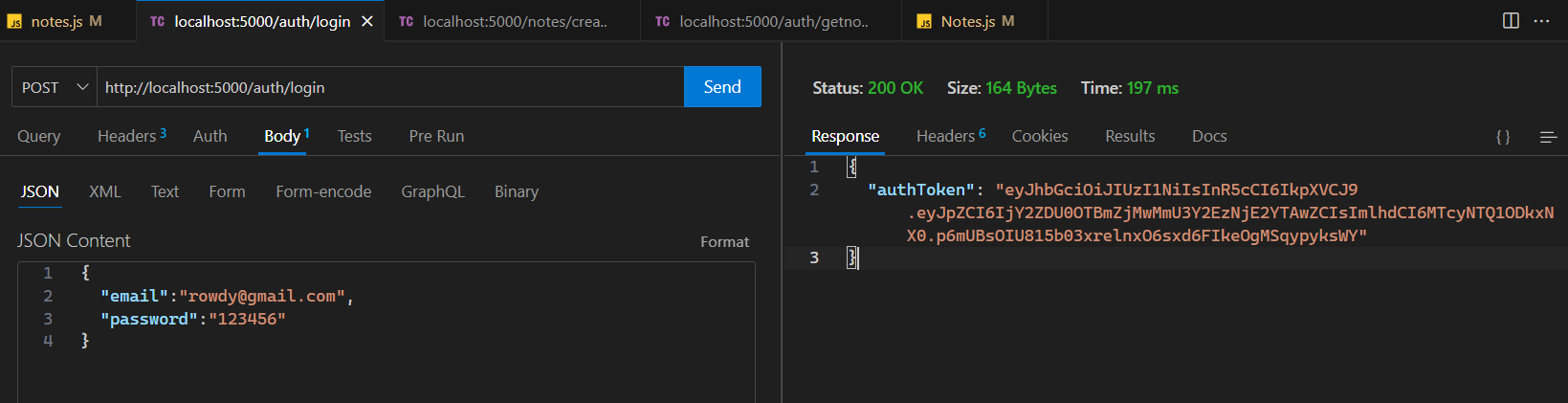
            return res.status(500).send({ error: error.message });

        }

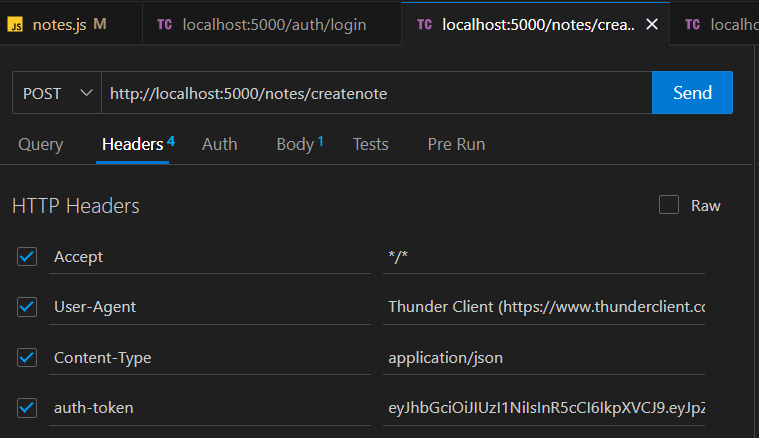
    });

module.exports = router;

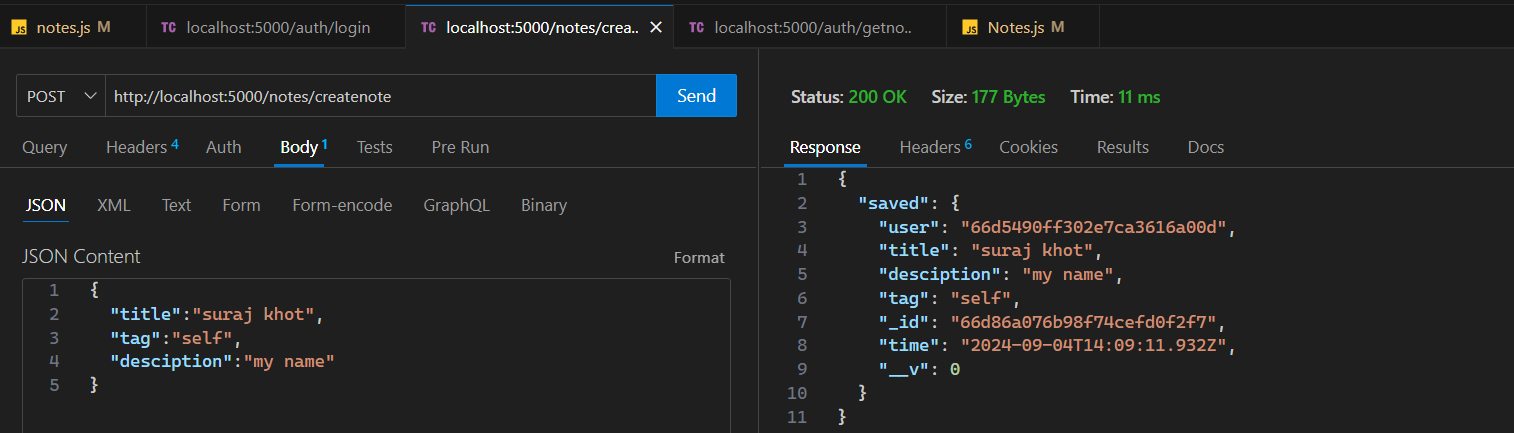
1. now test
2. login



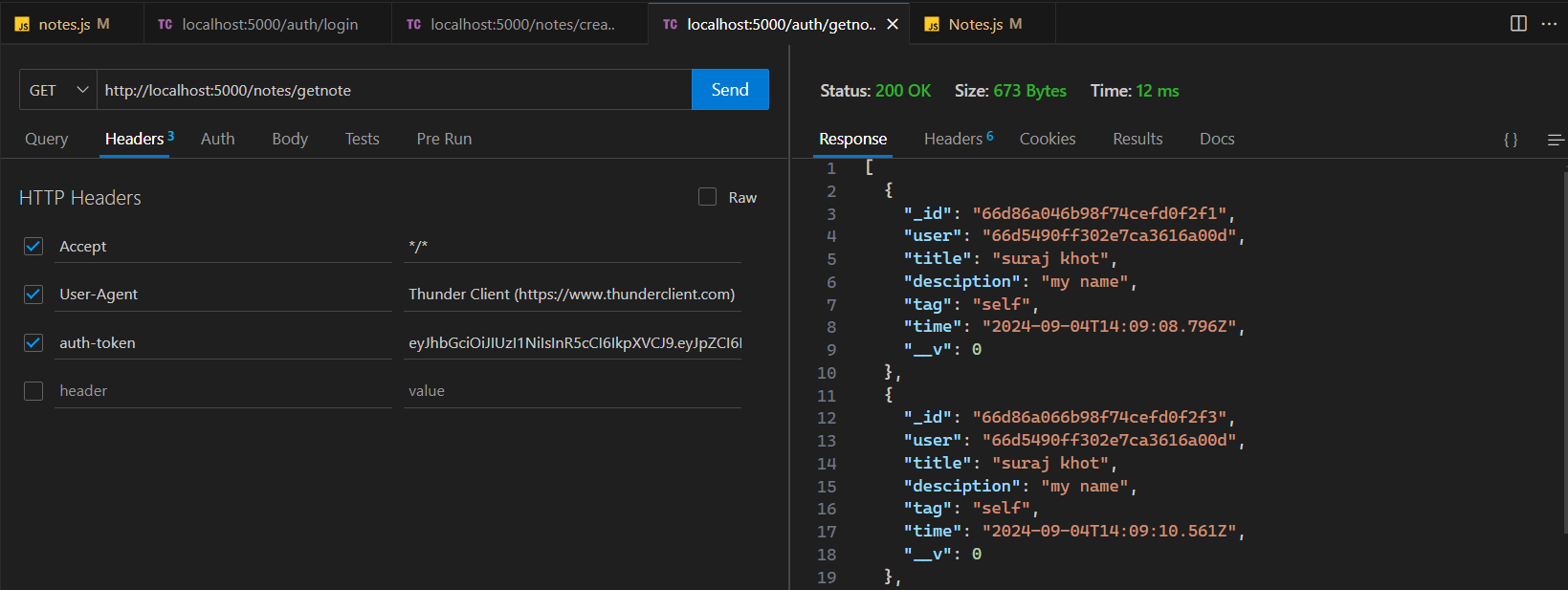
1. create note
2. set header for middleware



1. now set note



1. now get notes



1. now just update an existing note
2. create a route

//notes.js

/code

// !    update note

router.put(

    // url

    '/update/:id',

    // middleware

    getUserData,

    // fn

    async (req, res) => {

        // getting uid from middleware

        const uid = req.userId;

        // destucture body

        const { title, desciption, tag } = req.body;

        try {

            // new note

            let newNote = {};

            if (title) { newNote.title = title; }

            if (desciption) { newNote.desciption = desciption; }

            if (tag) { newNote.tag = tag; }

            // find note by its id

            let note = await Notes.findById(req.params.id);

            // if note not found

            if (!note) {

                return res.status(404).send({ error: "Note not found" });

            }

            // if user in note is not equal to requested user

            if (note.user.toString() != uid) {

                return res.status(401).send({ error: "not valid user" });

            }

            // update note

            note = await Notes.findByIdAndUpdate(req.params.id, { $set: newNote }, { new: true });

            res.send(newNote);

        }

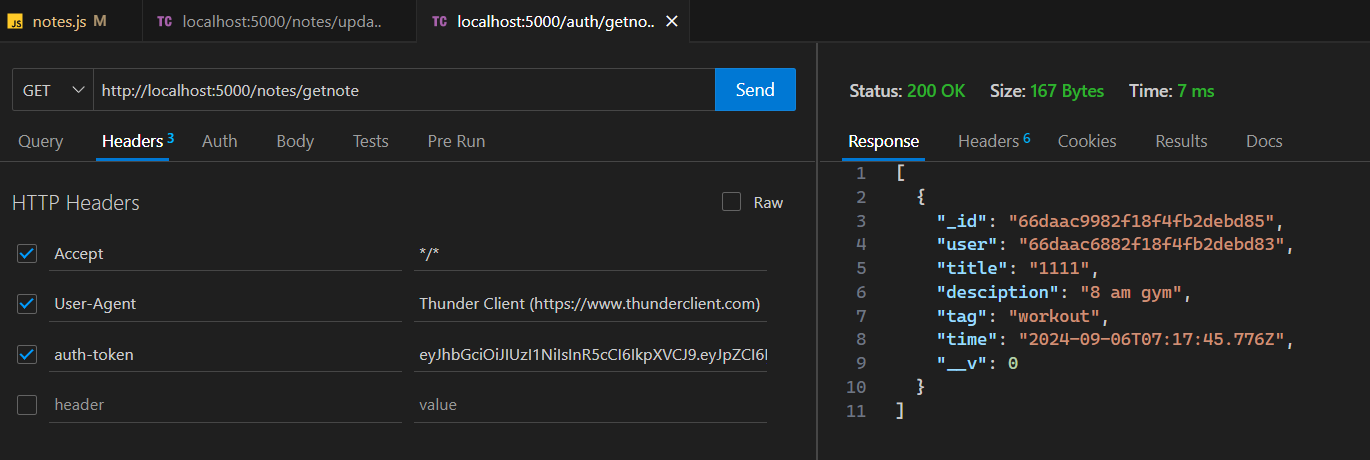
        catch (error) {

            return res.status(500).send({ error: "Internal server error" });

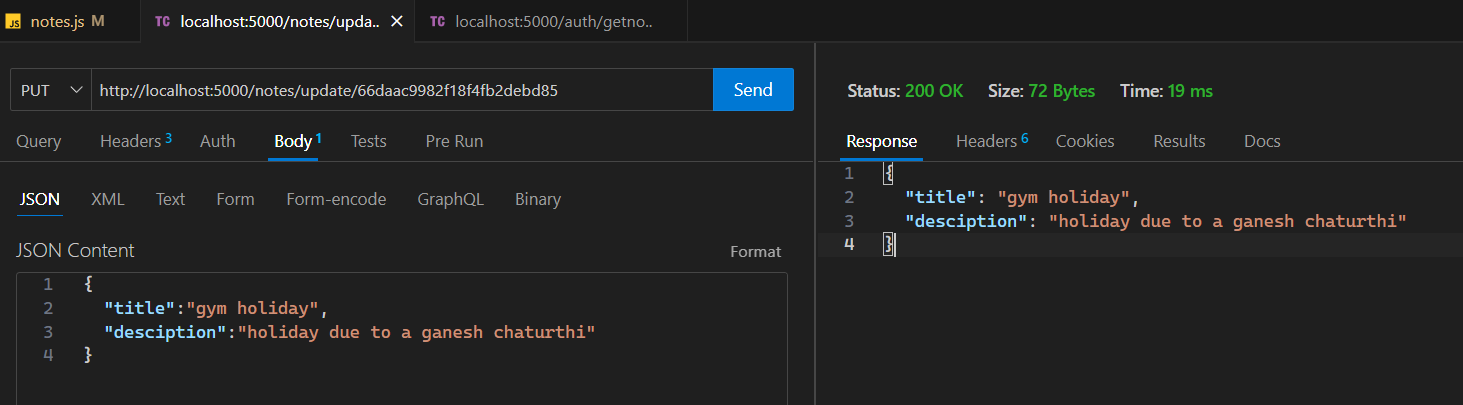
        }

    });

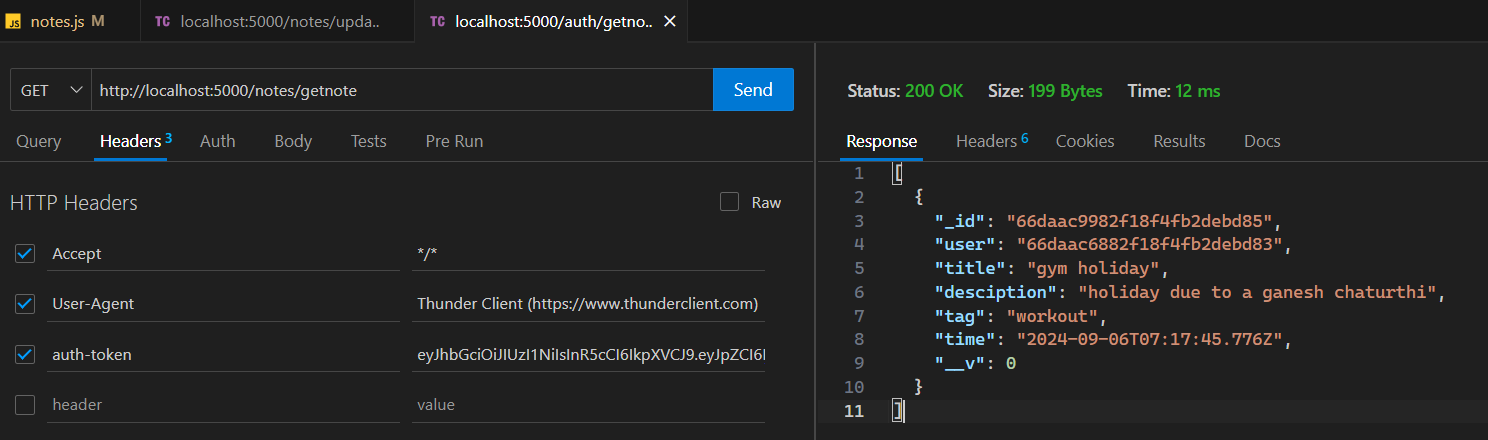
1. output
2. get existing note



1. update it by its id with header user id:



1. now check it



1. now delete a note
2. create a route

// !    delete note

router.delete(

    // url

    '/delete/:id',

    // middleware

    getUserData,

    // fn

    async (req, res) => {

        // getting uid from middleware

        const uid = req.userId;

        try {

            // find note by its id

            let note = await Notes.findById(req.params.id);

            // if note not found

            if (!note) {

                return res.status(404).send({ error: "Note not found" });

            }

            // if user in note is not equal to requested user

            if (note.user.toString() != uid) {

                return res.status(401).send({ error: "not valid user" });

            }

            // delete note

            note = await Notes.findByIdAndDelete(req.params.id);

            res.send({ "sucess": "note deleted" })

        }

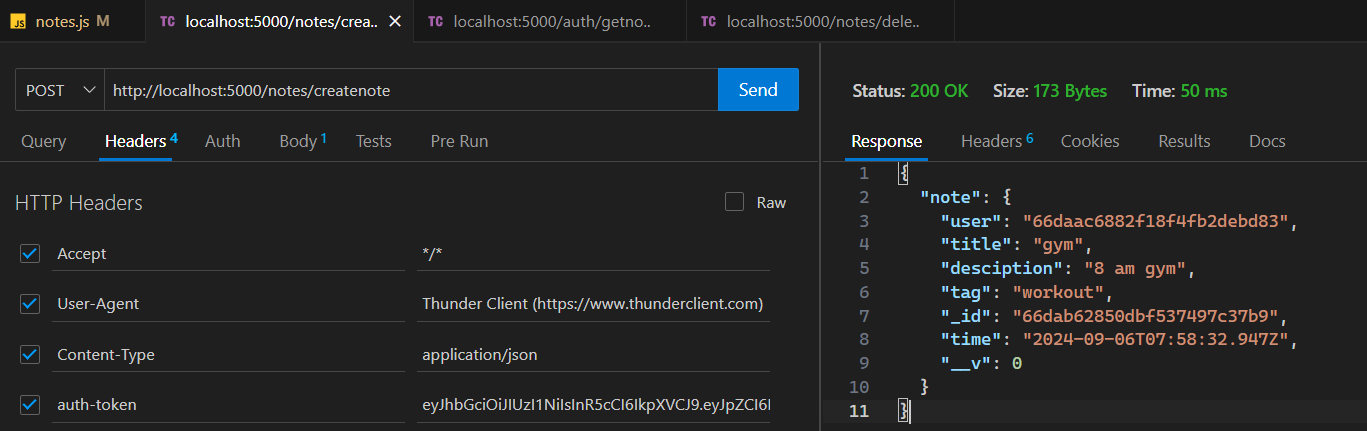
        catch (error) {

            return res.status(500).send({ error: "Internal server error" });

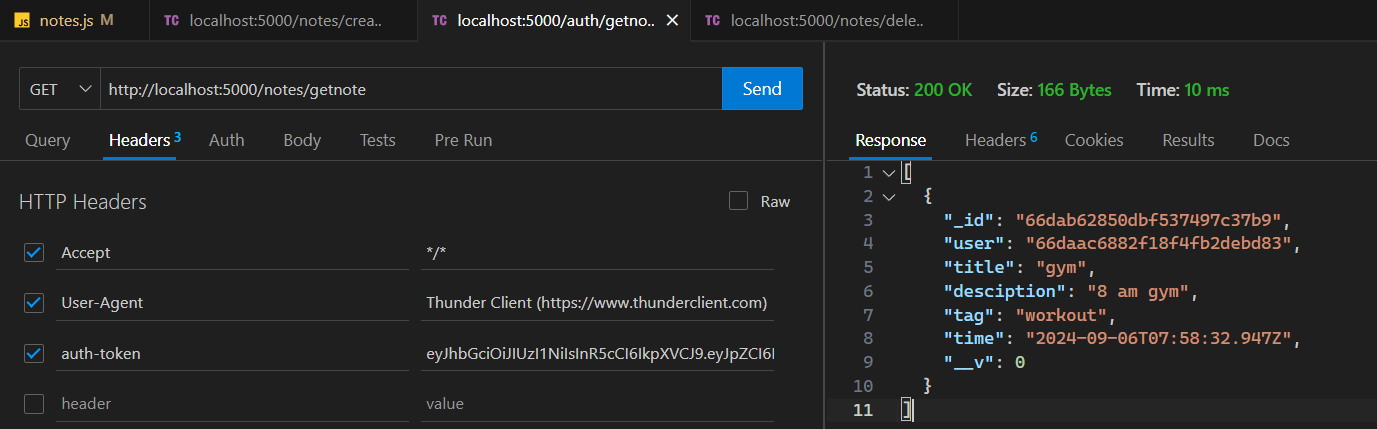
        }

    });

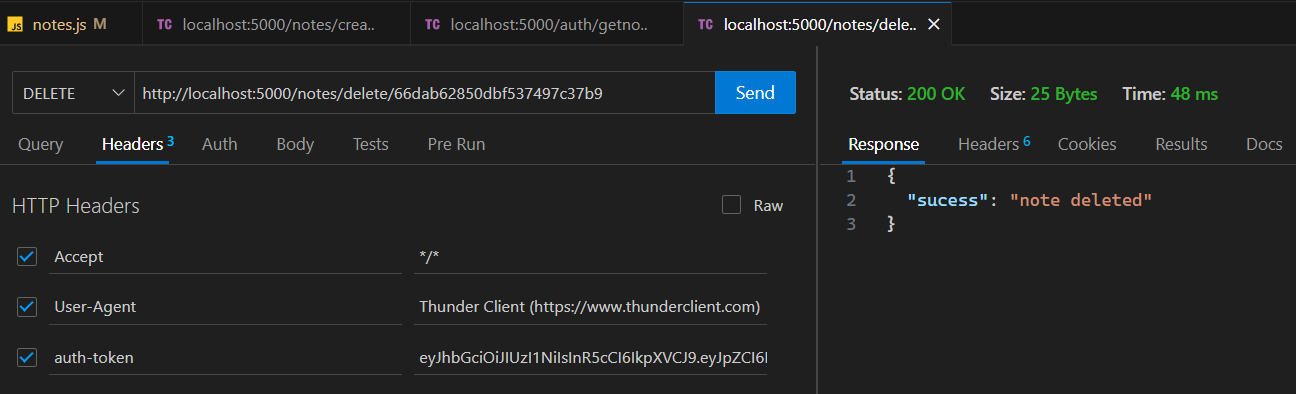
1. output
2. create a note



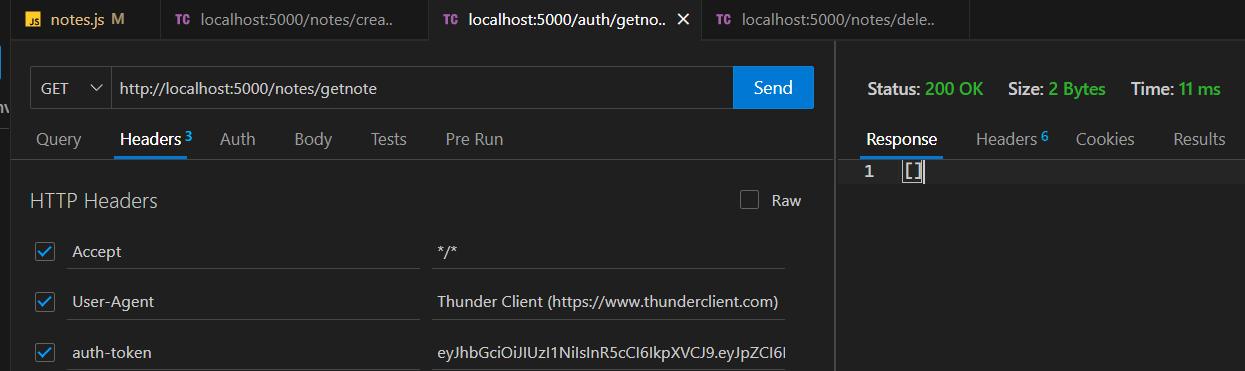
1. get note



1. delete note



1. check note



1. now lets div into frontend
2. now install router

PS D:\Workspace\React\inotebook> npm i react-router-dom

added 3 packages, and audited 1545 packages in 9s

1. add css and script of bootstrap
2. create components
3. now let’s checkout context (useContext) hook in react
4. create a file for note context

// inotebook\src\context\note\NoteContext.js

import { createContext } from 'react';

/\*

The value you want the context to have when there is no matching Provider in the tree above the component reading the context. This is meant as a "last resort" fallback.

Lets you create a Context that components can provide or read.

\*/

const NoteContext = createContext();

export default NoteContext;

1. create a file for state of note context

// inotebook\src\context\note\NoteState.js

import NoteContext from './NoteContext'

import { useState } from 'react';

const NoteState = (props) => {

    // initial state

    const s1 = {

        "name": "suraj",

        "sirname": "khot"

    }

    // use state snippet

    const [state, setstate] = useState(s1);

    // fun which update state

    const update = () => {

        setTimeout(() => {

            setstate({ "name": "rowdy" })

        }, 3000);

    }

    return (

        <NoteContext.Provider value={{ state, update }}>

            {props.children}

        </NoteContext.Provider>

    );

}

export default NoteState;

1. app.js wrap it with state of note context

import NoteState from './context/note/NoteState';

function App() {

  return (

    <>

      <NoteState>

        <Router>

          <Nav />

          <Routes>

            <Route path="/" element={<Home />} />

            <Route path="/about" element={<About />} />

          </Routes>

        </Router>

      </NoteState>

    </>

  );

}

export default App;

1. use it

import React, { useContext, useEffect } from 'react';

import NoteContext from '../context/note/NoteContext';

export default function Home() {

    // Accepts a context object (the value returned from React.createContext) and returns the current context value, as given by the nearest context provider for the given context.

    const context = useContext(NoteContext);

    // check for update if updates reload it

    useEffect(() => {

        // calling update fun from notestate

        context.update();

        // eslint-disable-next-line

    }, []);

    return (

        <div>

            <p>hi {context.state.name} {context.state.sirname}</p>

        </div>

    )

}

1. output

