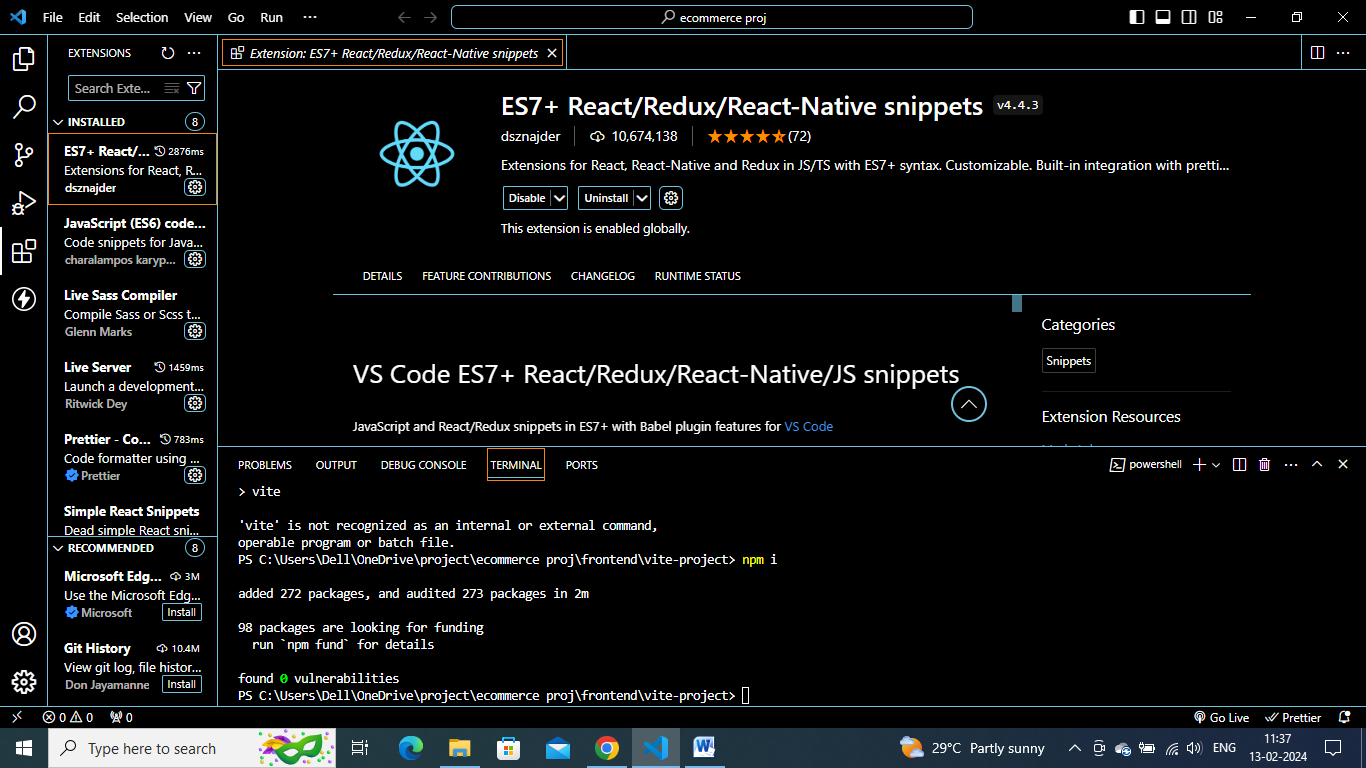
**Creating backend api first:**

Install es7 react/redux/react native snippets pluggin in vs code

Auto import –es6 ts,jsx.

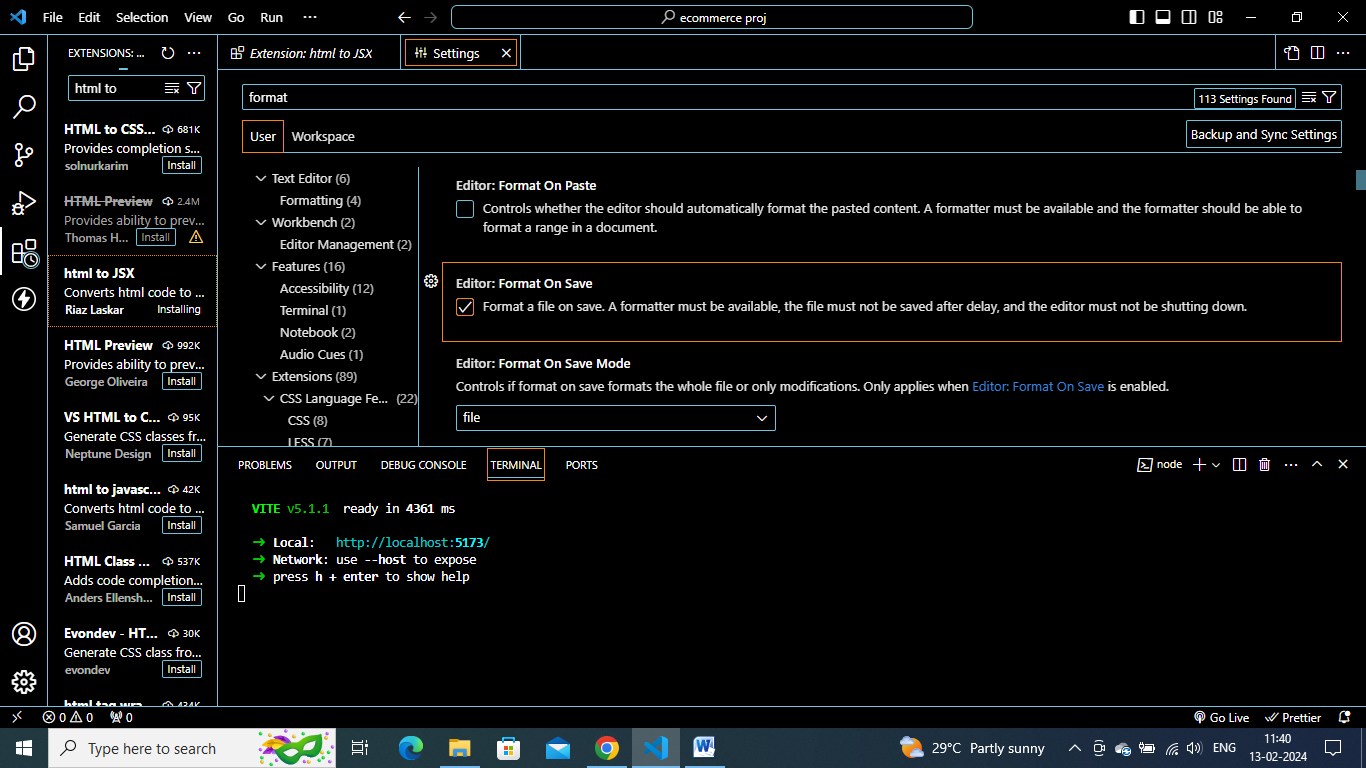
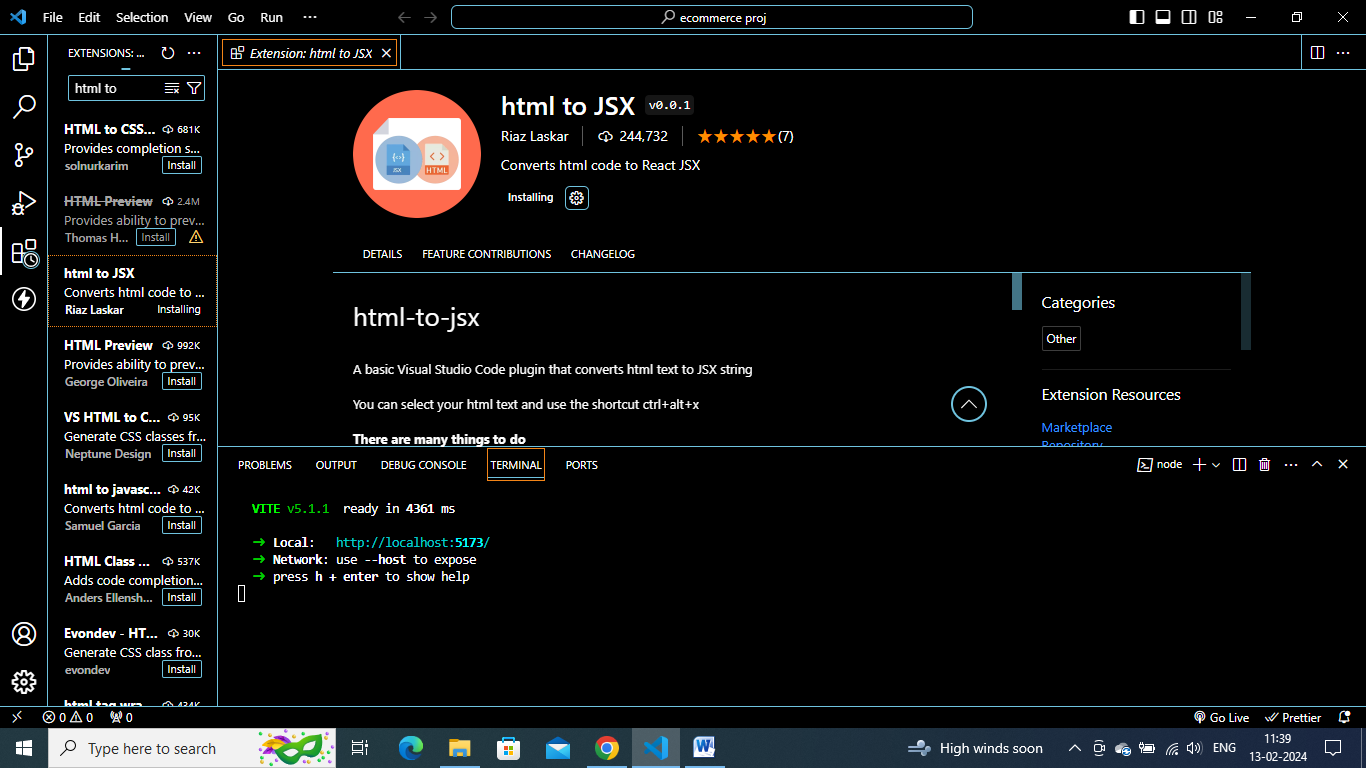
Dotenv



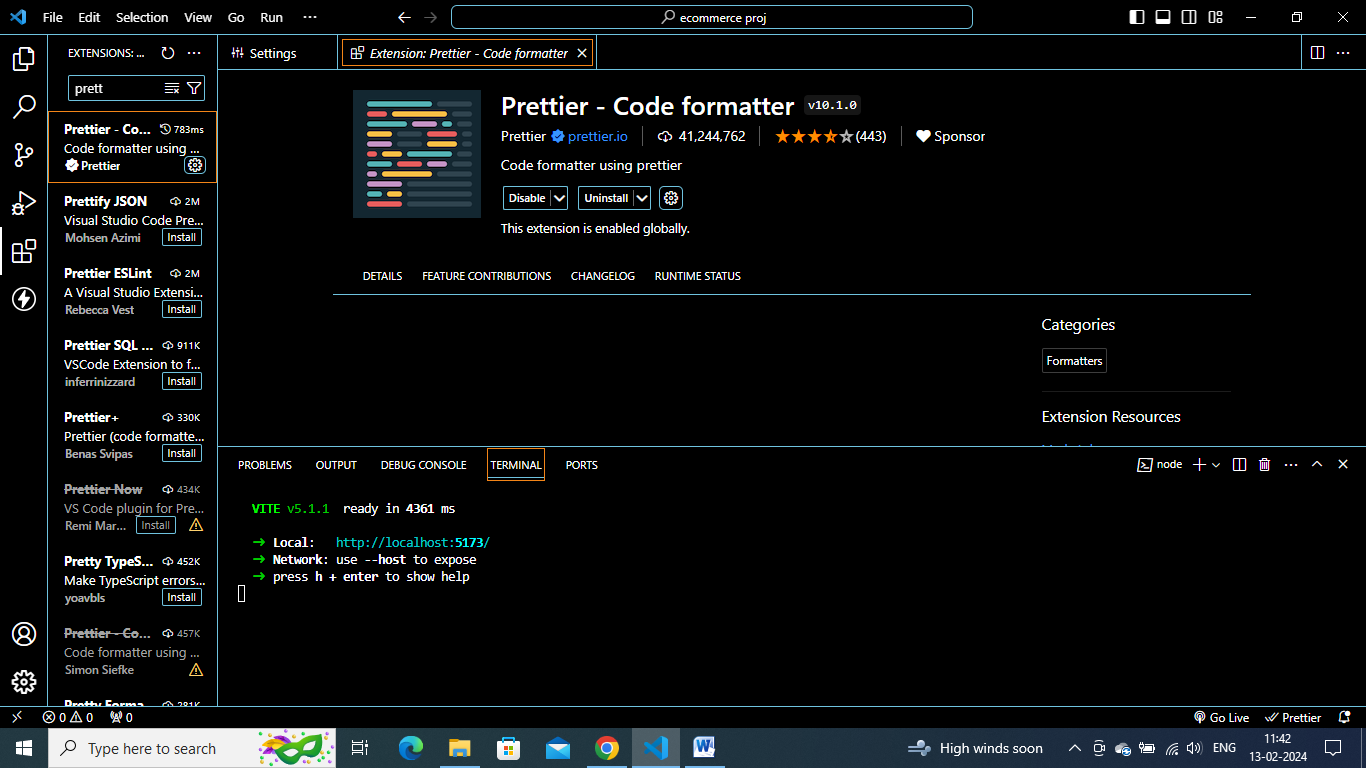




Html to jsx extension

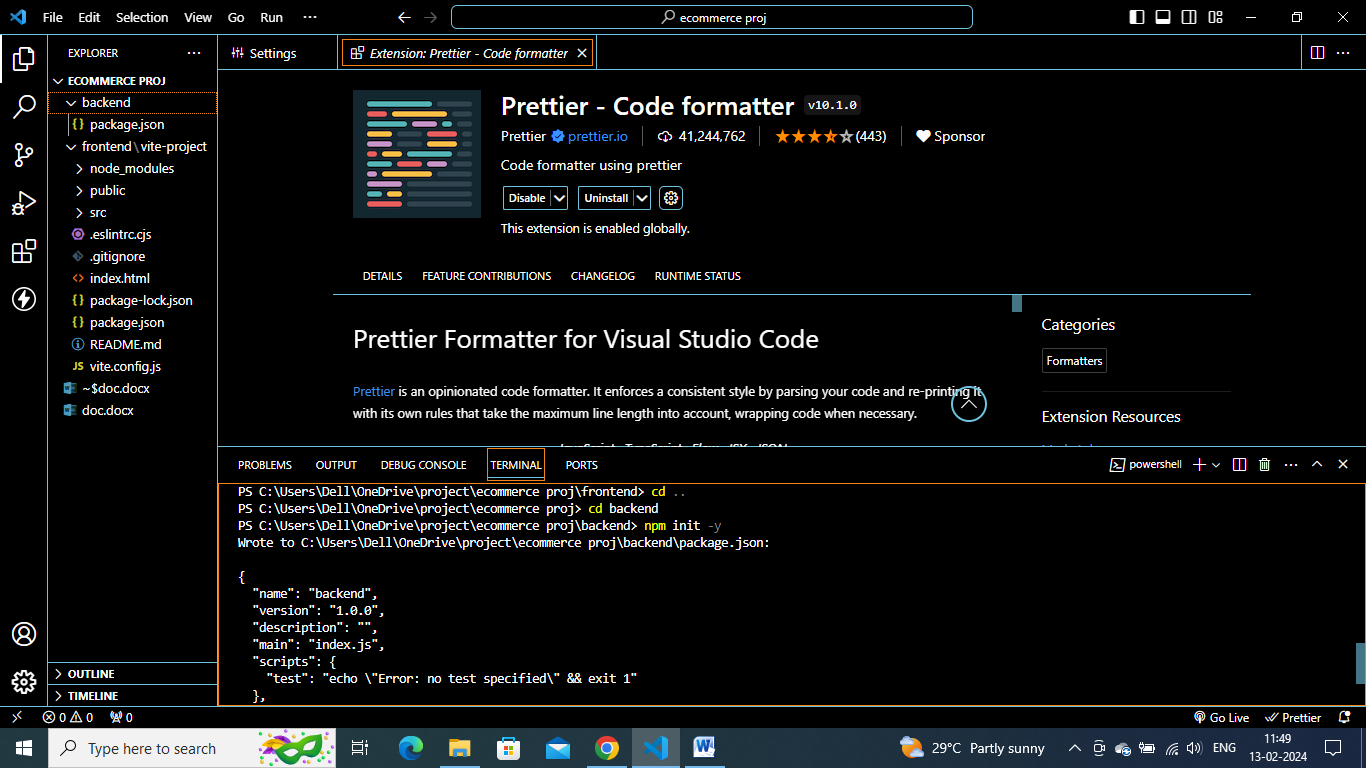


In settings – format search – format on save checked.

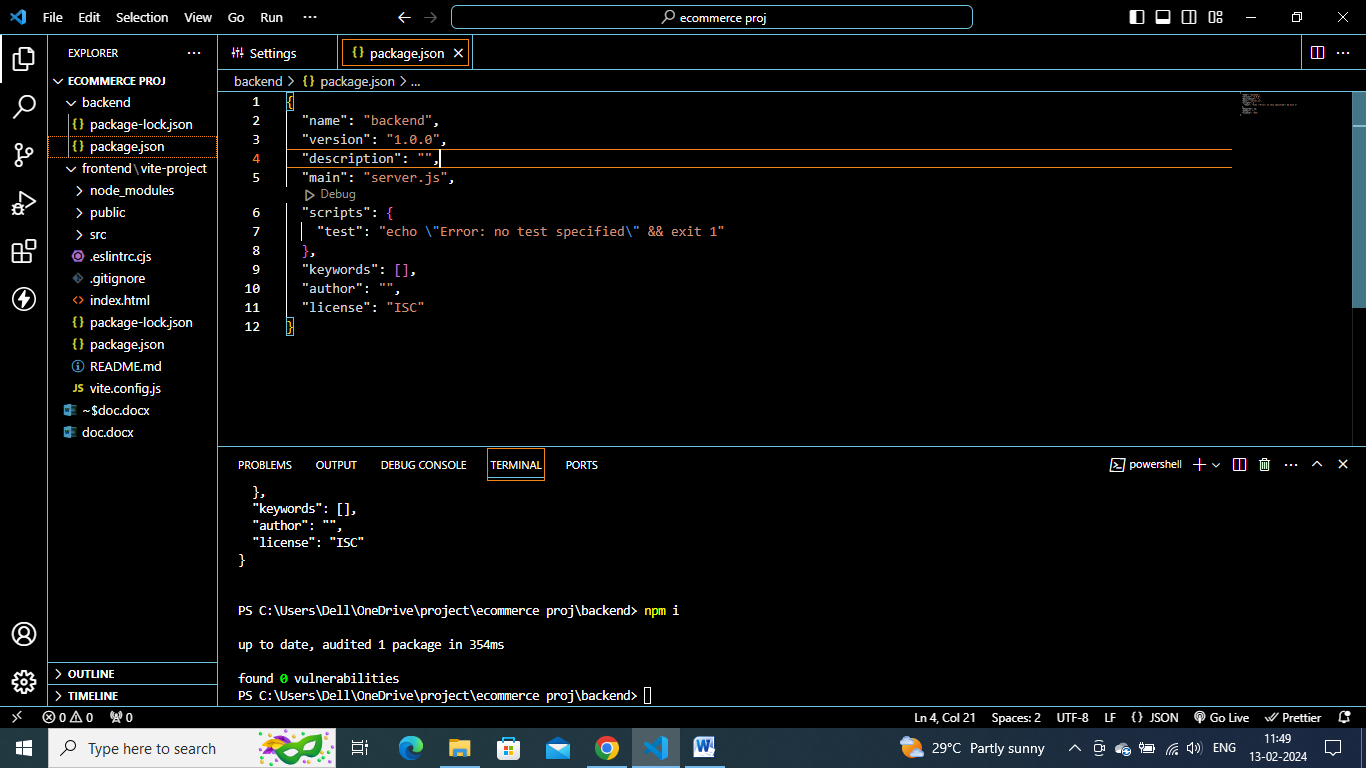


Create backend named folder

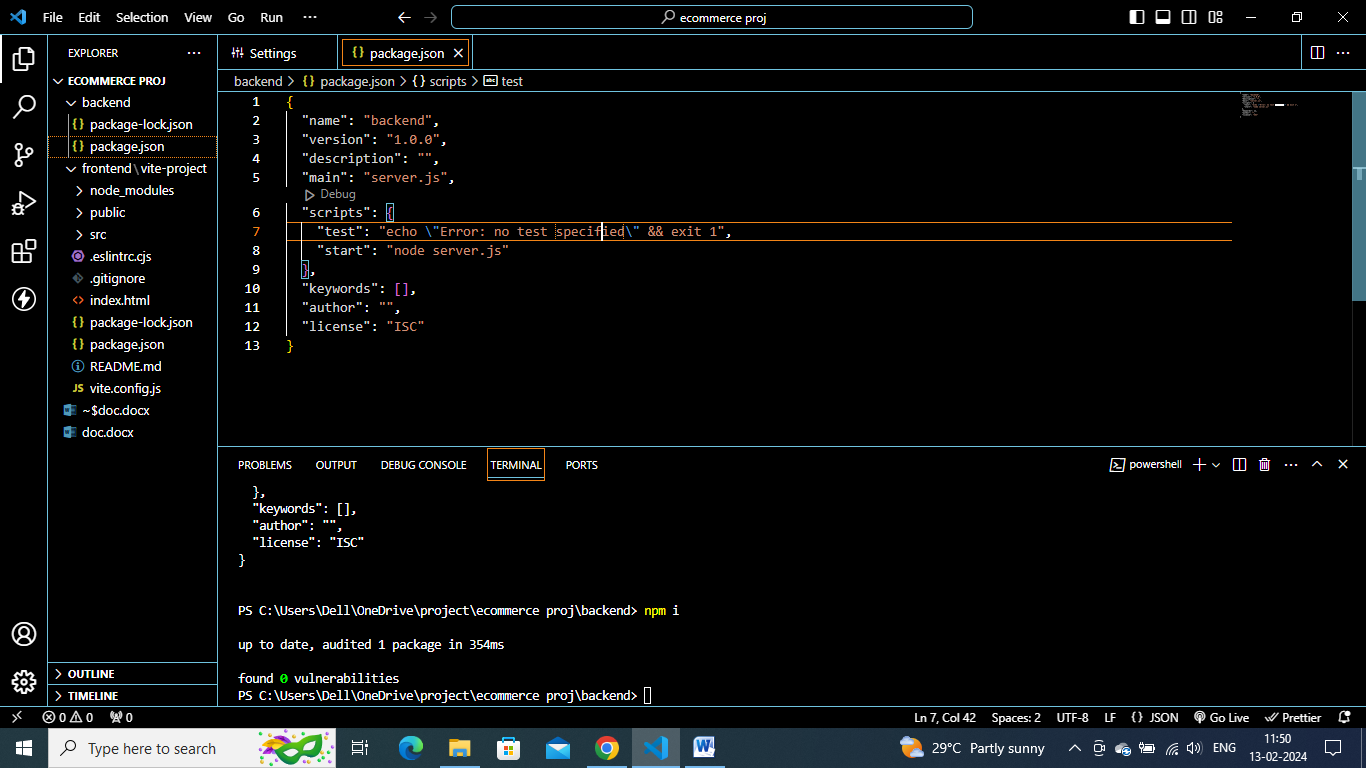
Do npm init –y inside it.



Observe a packge.json will be created .



Package.json has all these information . changing index.js to server.js



Can add start script like this.

By doing npm start this application will be running.

Can add author and keywords.

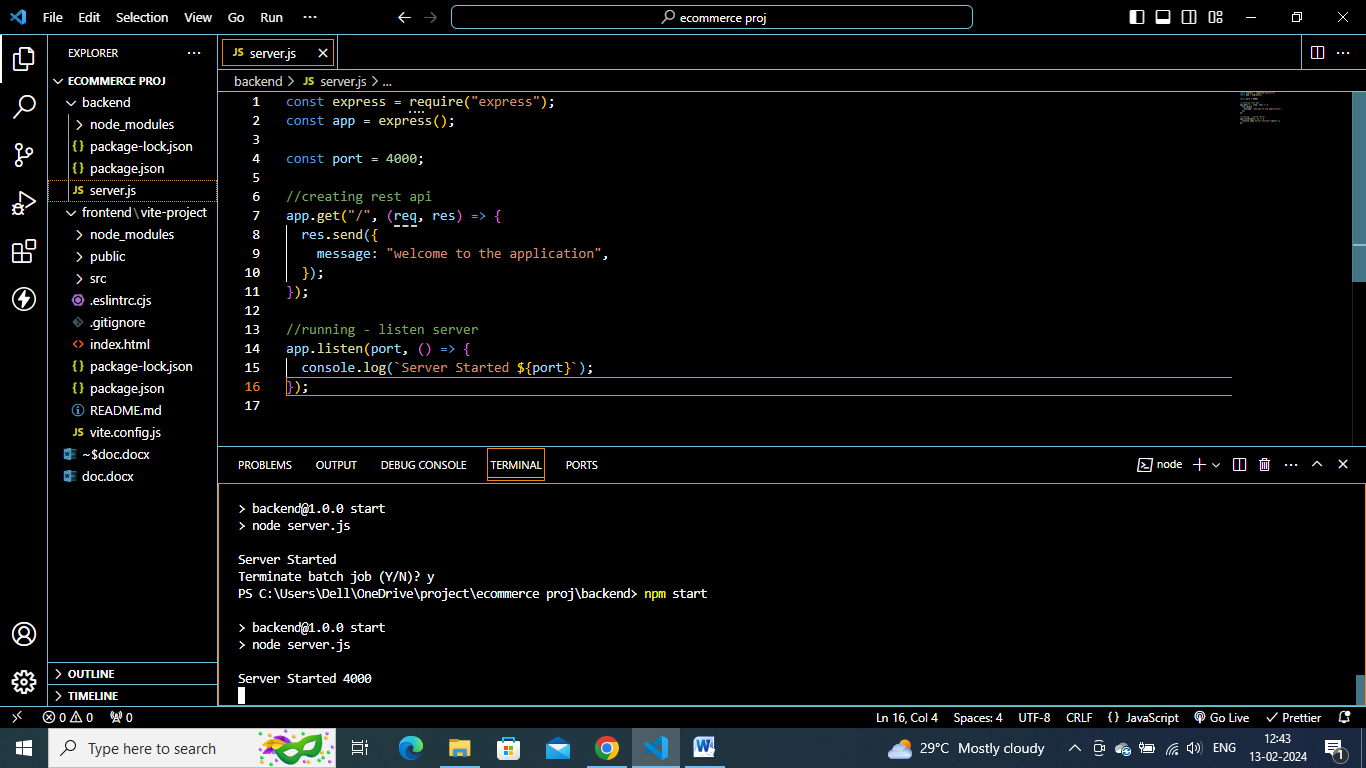
Npm I express colors’



Nodemodules and packagelock.json will be created.

**Creating server:**

Creating server.js file



Create server.js and run it using “node server.js” or “npm start”.

**Server.js:**

const express = require("express");

const app = express();

const port = 4000;

//creating rest api

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

  res.send({

    message: "welcome to the application",

  });

});

//running - listen server

app.listen(port, () => {

  console.log(`Server Started ${port}`);

});

import colors package:

const colors=require('colors');

  console.log(`Server Started ${port} `.bgCyan.white);

creating .env file :  
PORT= 4000

npm I dotenv morgan- morgan helps to check which url we hit.

**Server.js**

//configure env

const dotenv = require("dotenv").config();

if .env file is in other folder like that – then can mention path like this.

const dotenv = require("dotenv").config({path:''});

const port = process.env.PORT || 4000;

run application and check

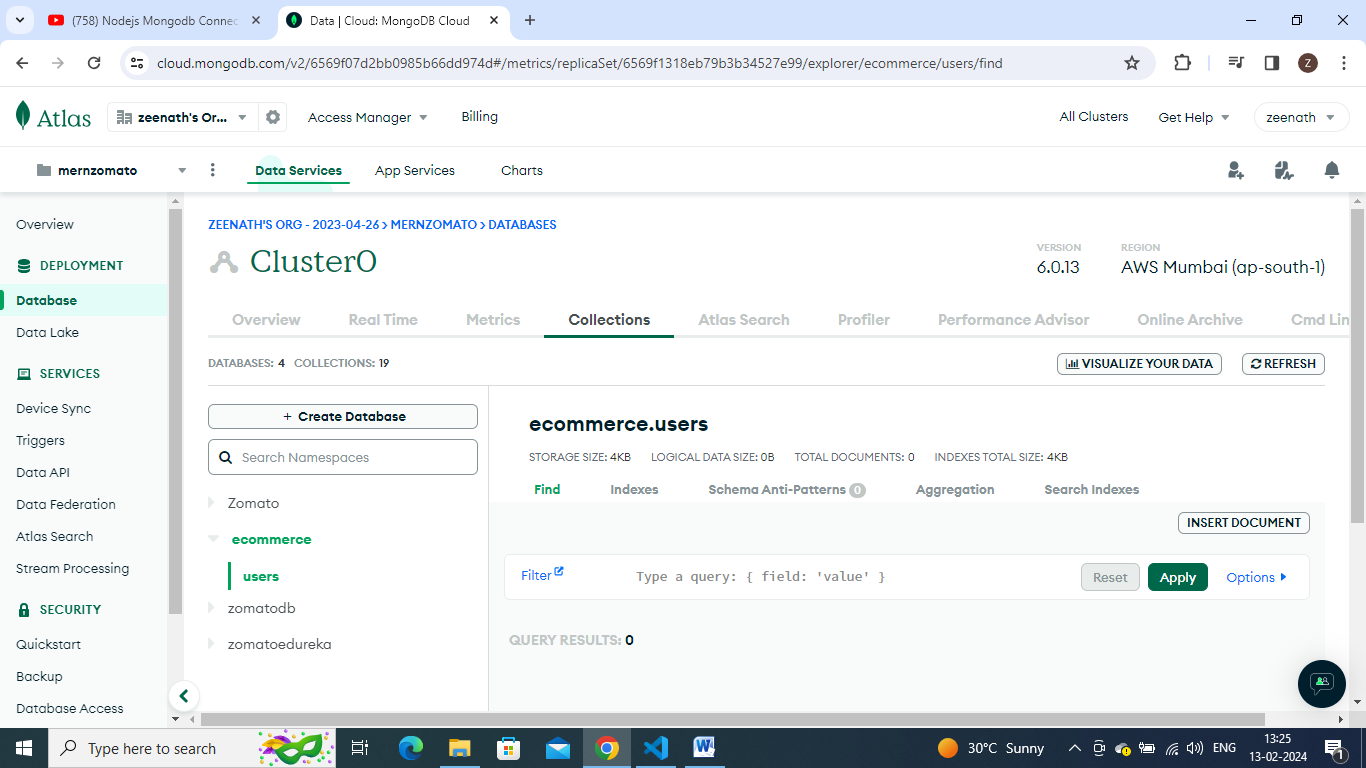
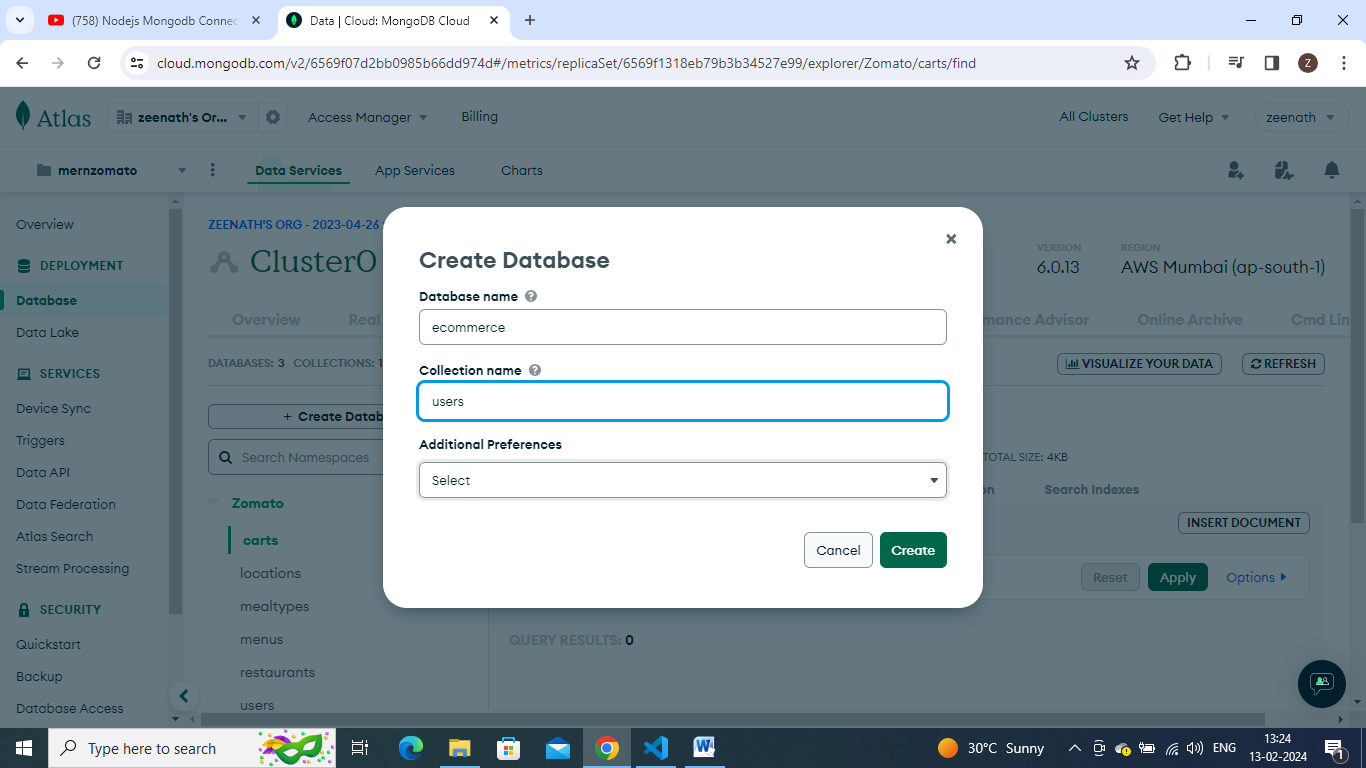
successfully created a server and somewt secured our project using .env file

**Connecting to mongodb:**

Install mongodb compass

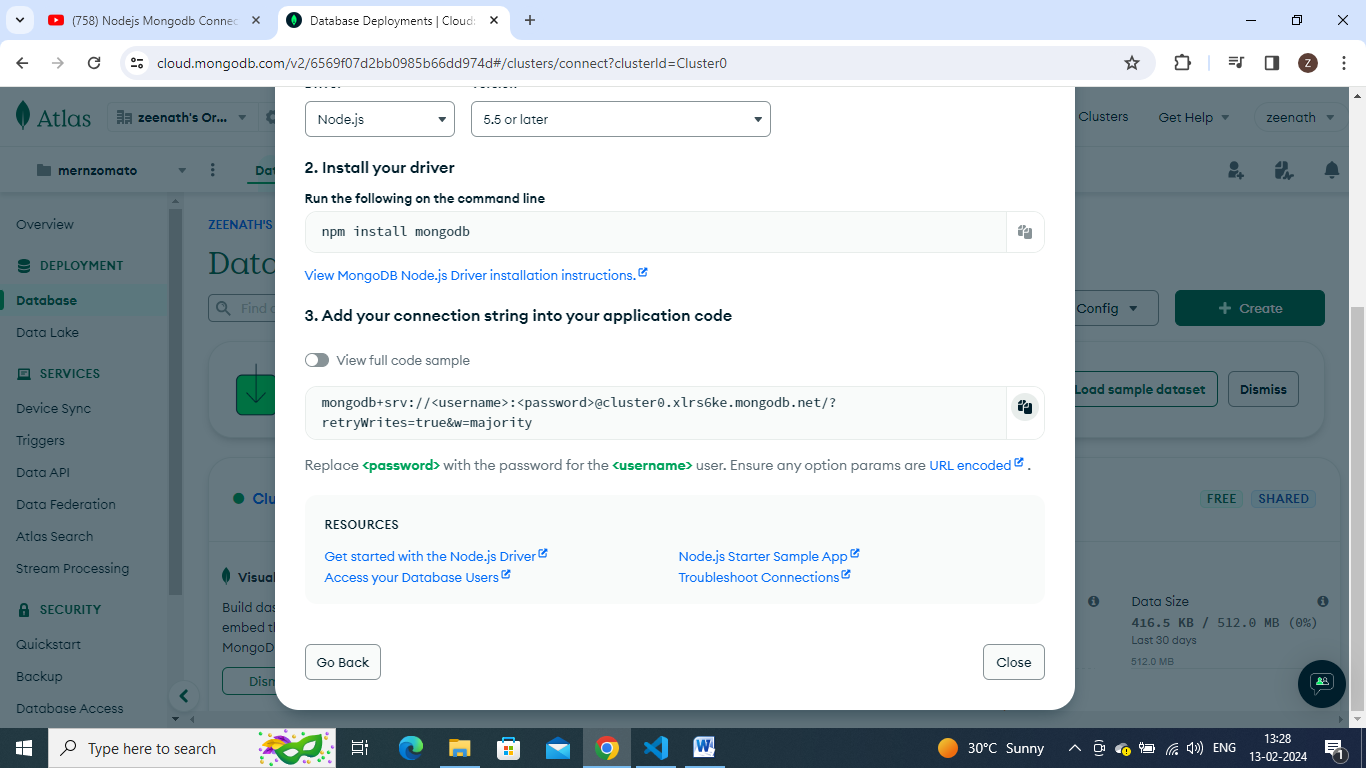
Create a free shared cluster in mongodb atlas.

Browse collections – create db and create collection



Before this in database access,create username and pswd and in network access, set ip addresses.

Go to database tab-> connect->



Copy connection link.

mongodb+srv://<username>:<password>@cluster0.xlrs6ke.mongodb.net/?retryWrites=true&w=majority

MONGO\_URL =mongodb+srv://<username>:<password>@cluster0.xlrs6ke.mongodb.net/?retryWrites=true&w=majority

Create a variable like this in .env file

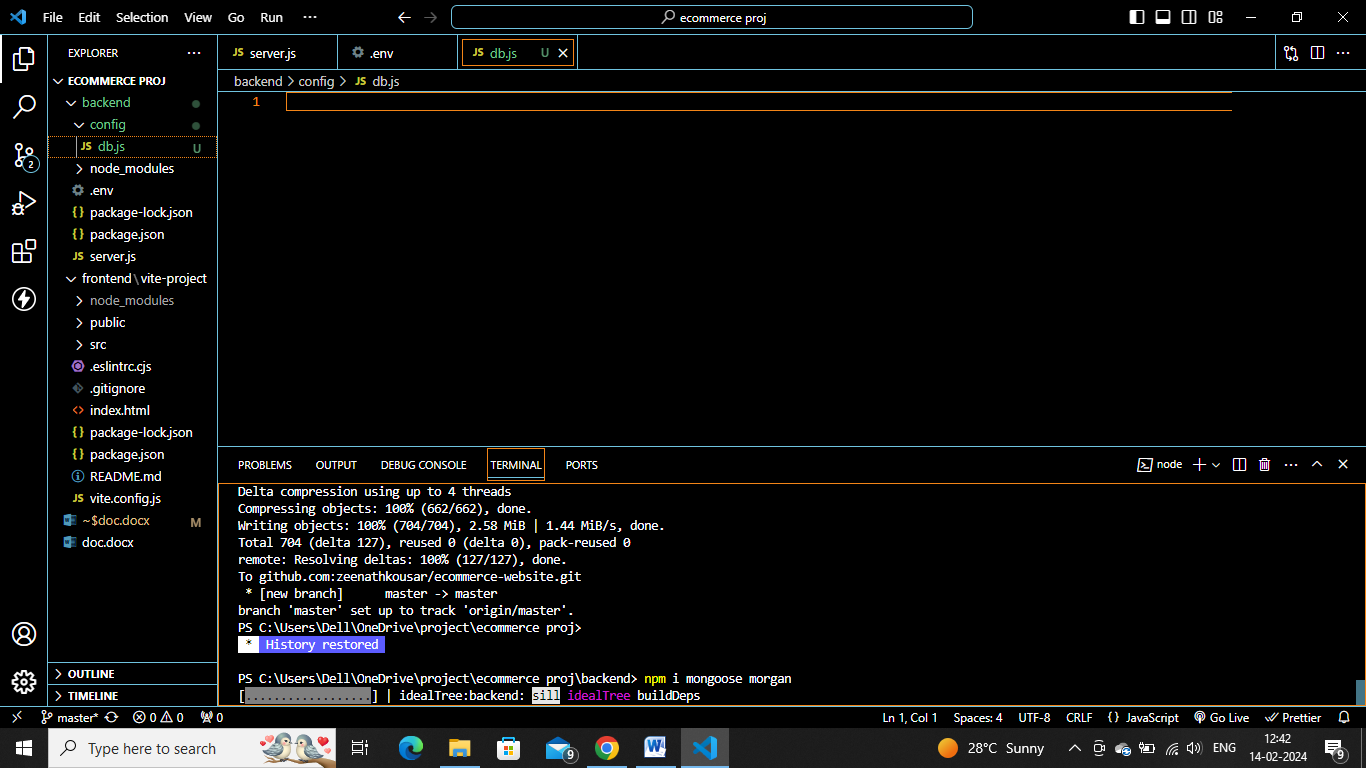
MONGO\_URL =mongodb+srv://zomatoedureka:zomatoedureka@cluster0.xlrs6ke.mongodb.net/ecommerce?retryWrites=true&w=majority

Add username,pswd and db name

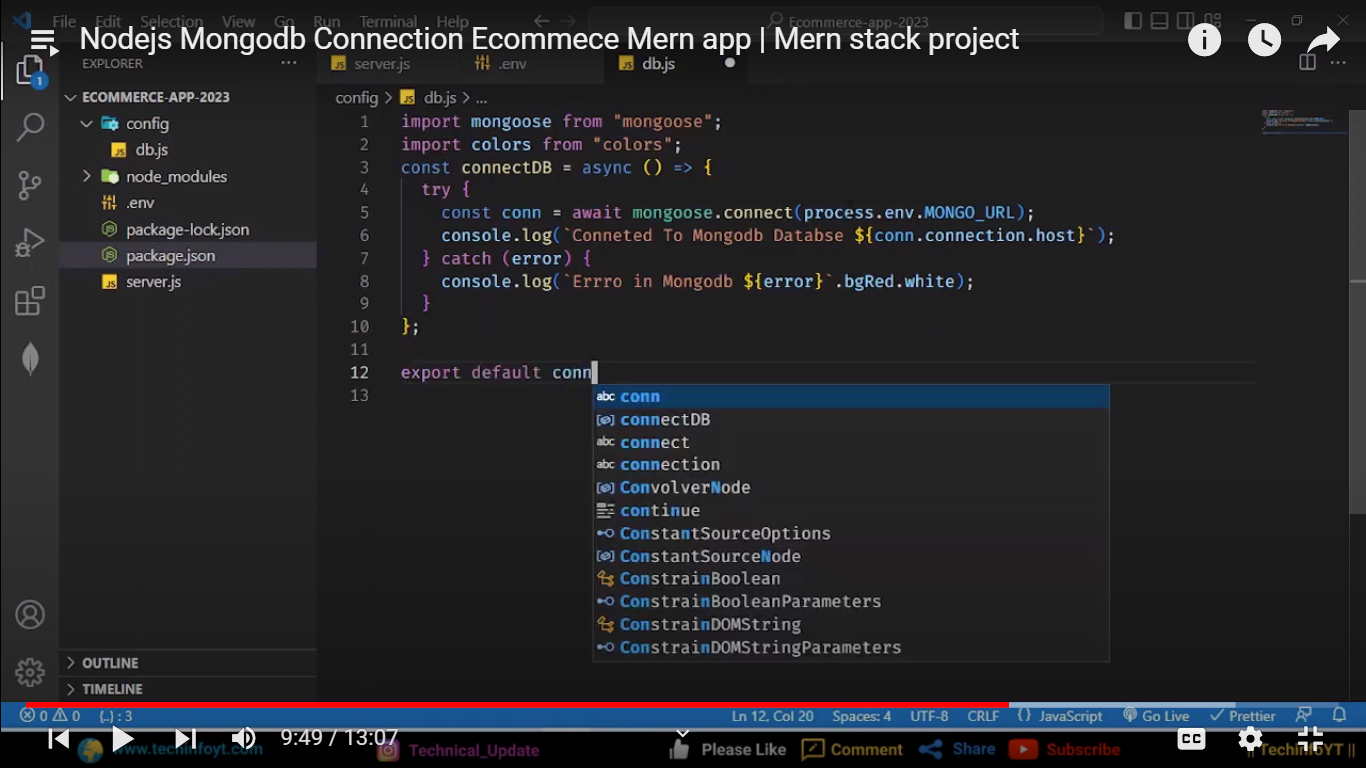
**Create config folder> db.js**

**Npm I mongoose morgan**

**Morgan- to show api requests on console**



**Db.js**



**In video – they are using import like stmts syntax-**  bcoz, they mentioned type:module as a script in package.json

**Main file- server.js**

U can import morgan

const morgan = require("morgan");

connectdb gets improted directly in this, if u are using vscode

//configure morgan- middlewares

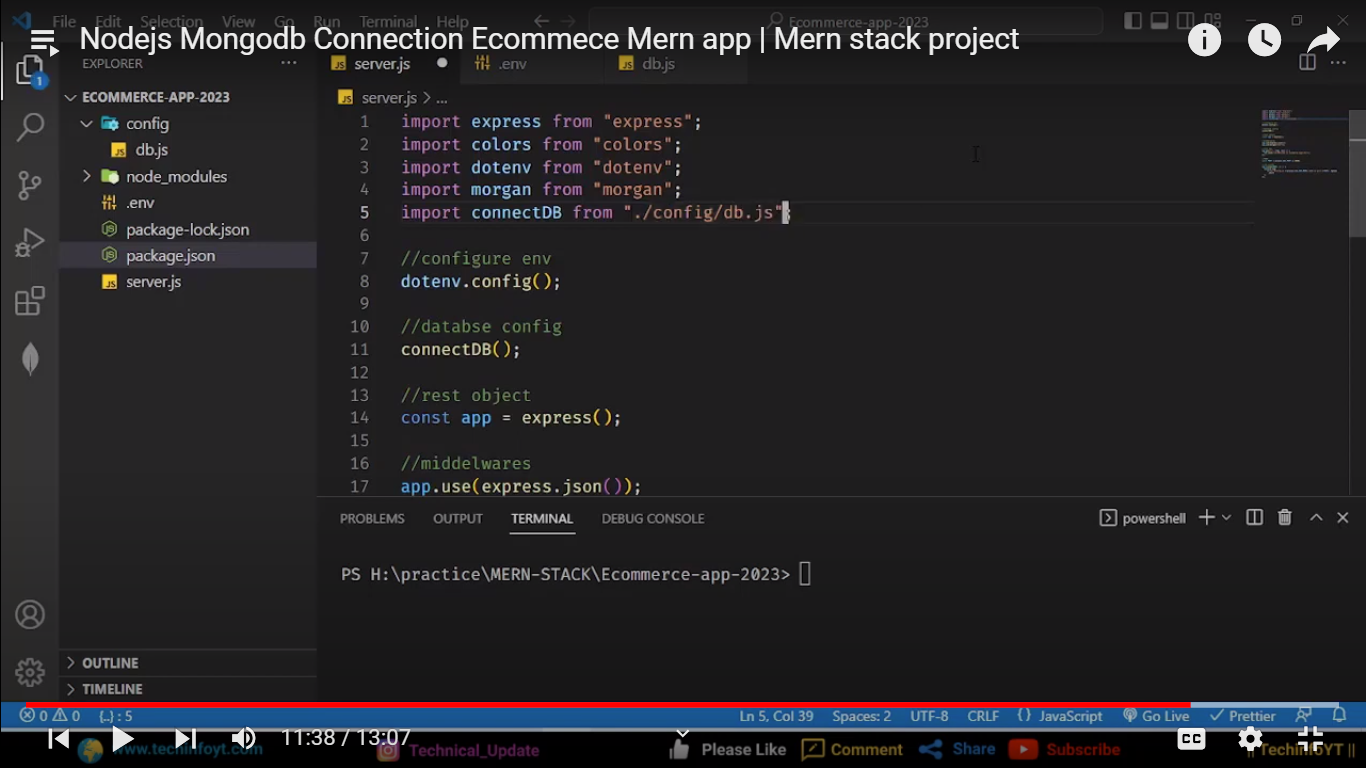
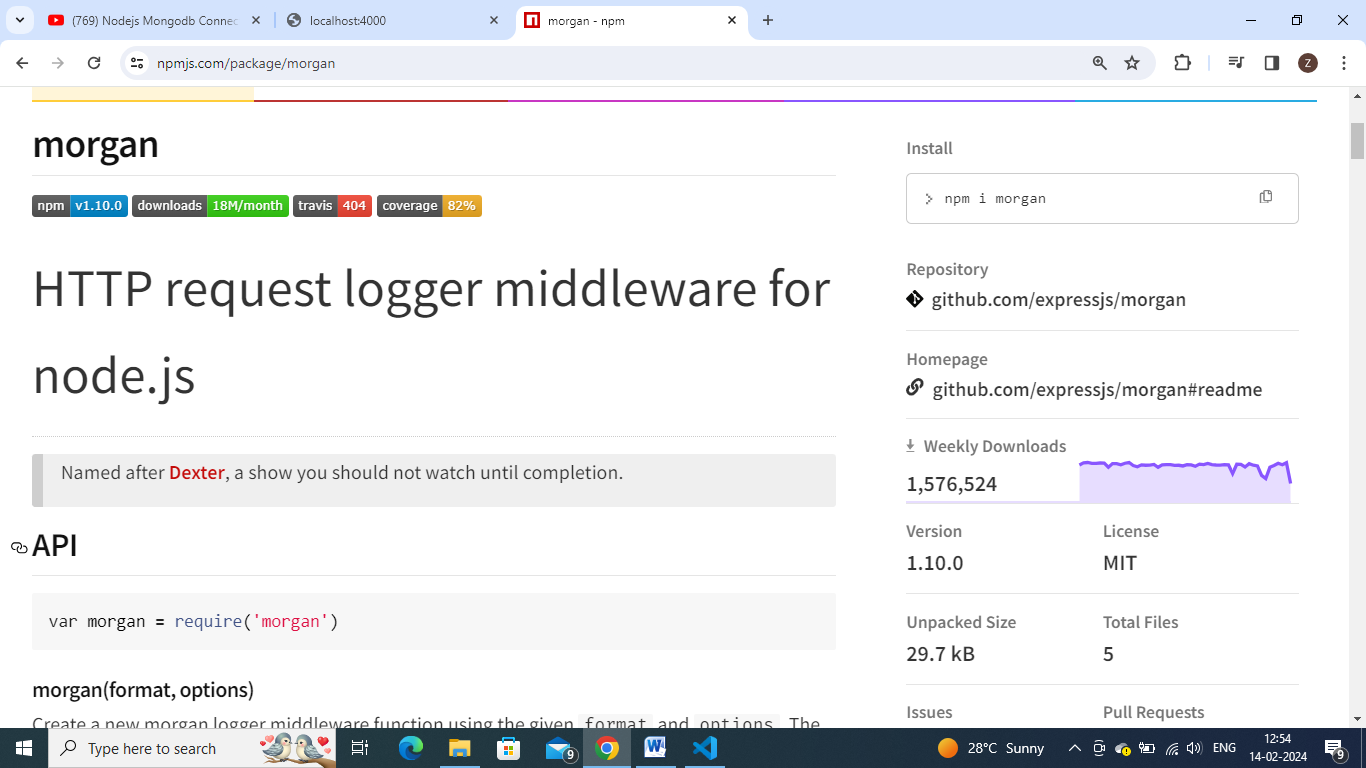
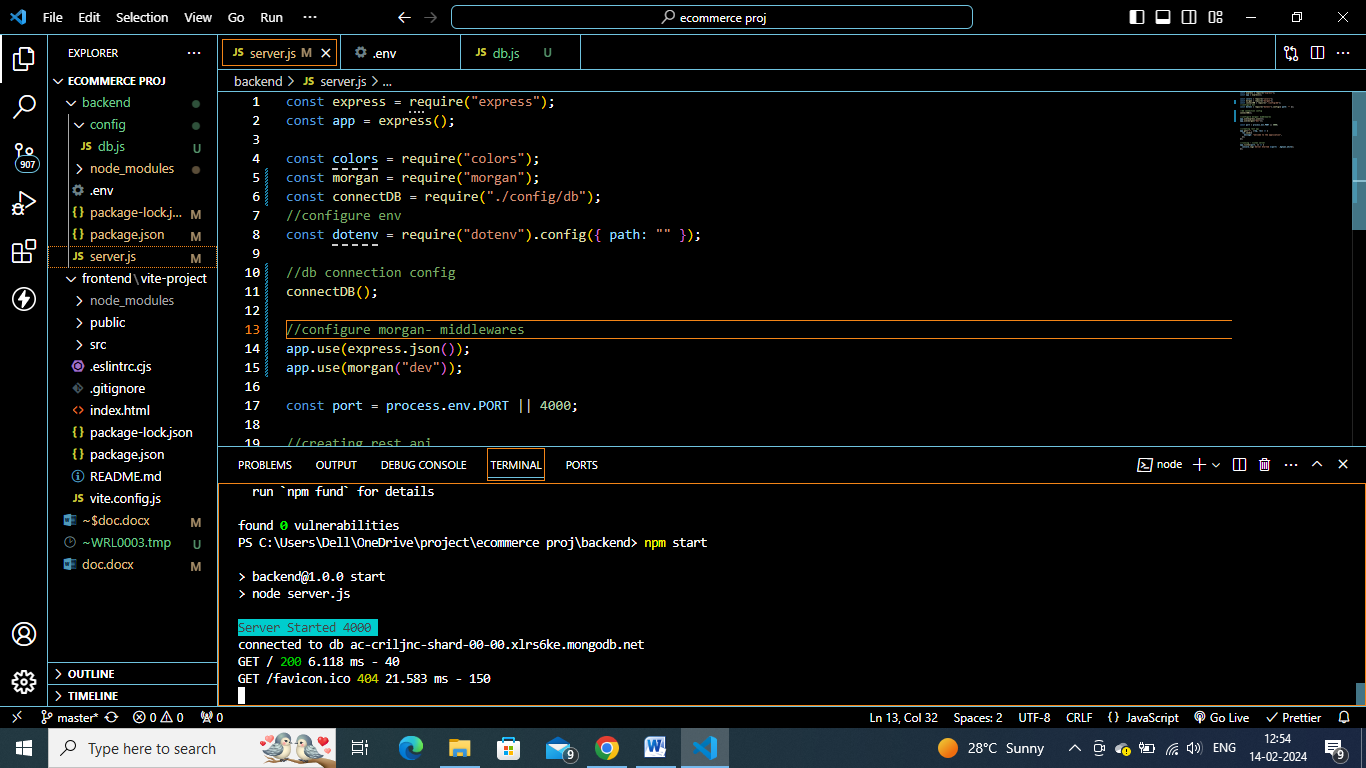
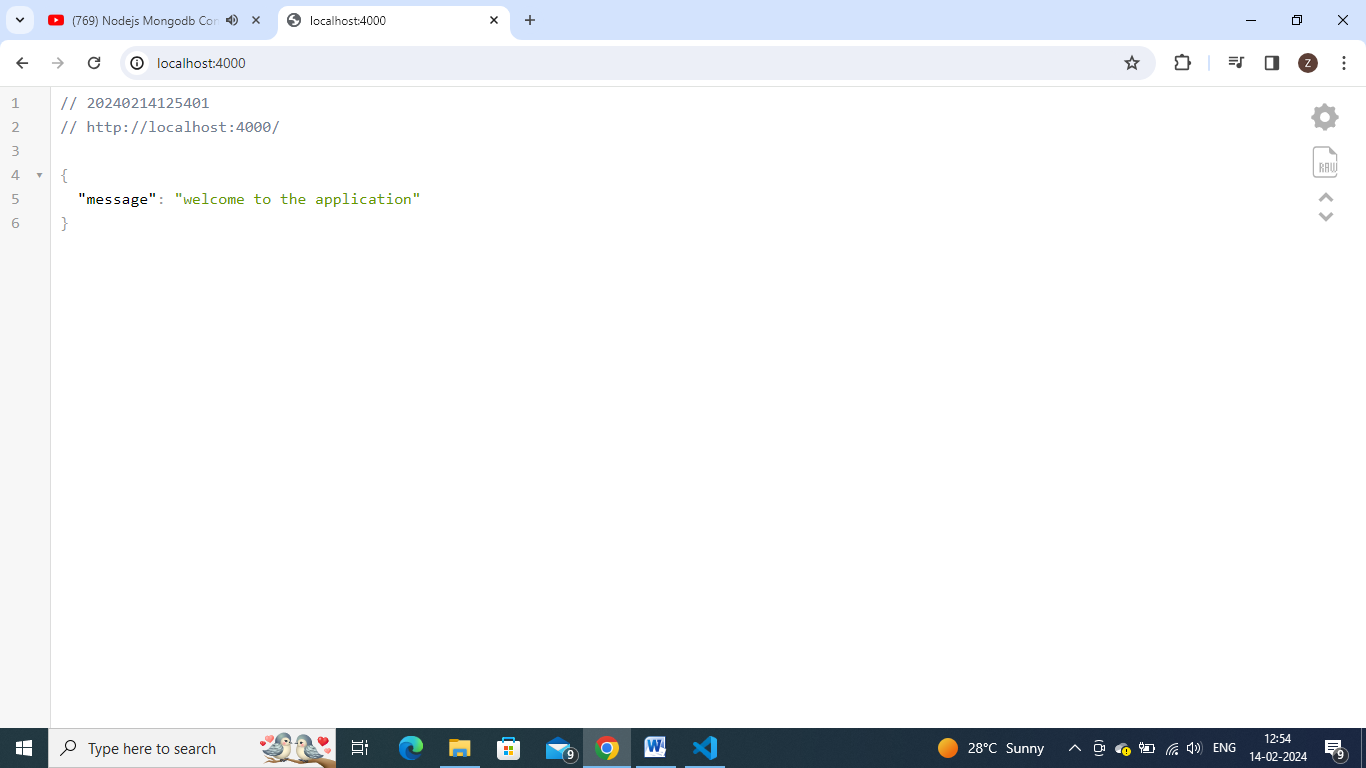
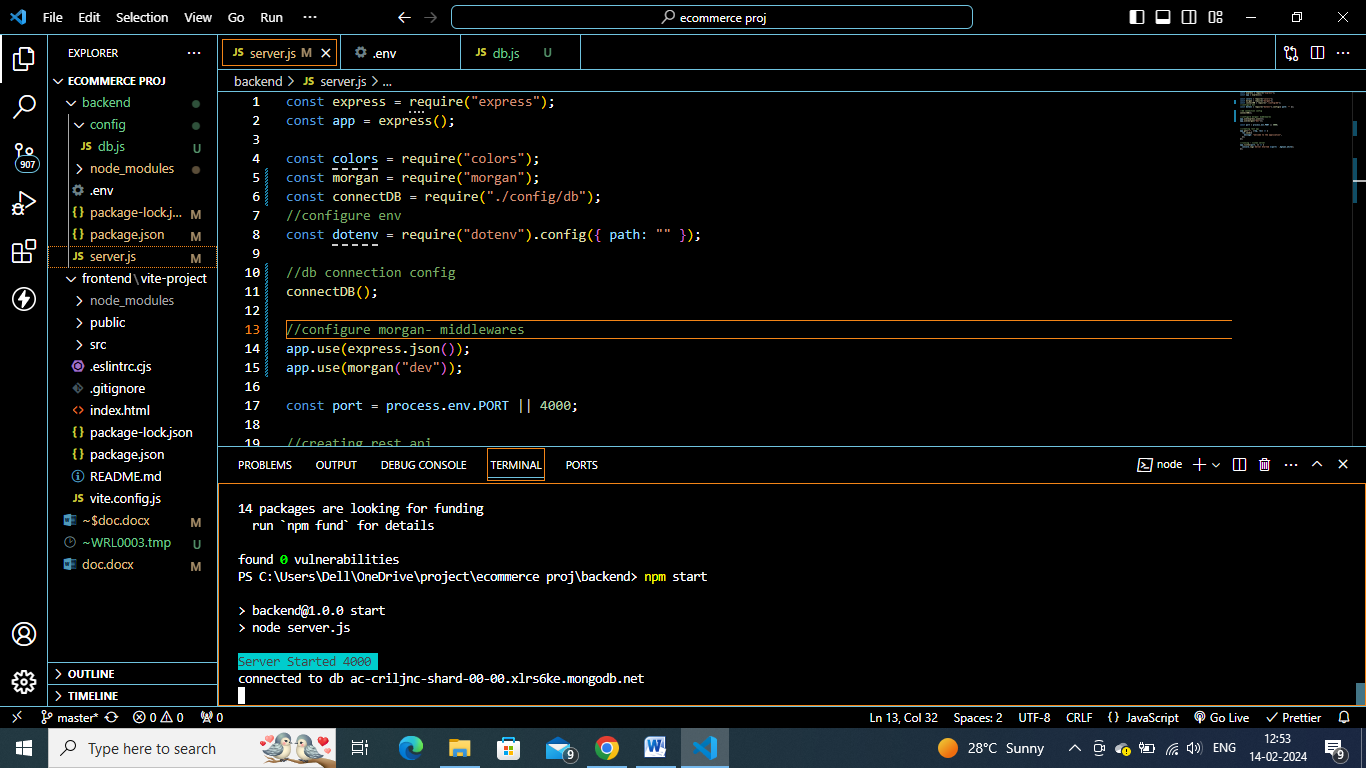
app.use(express.json())

app.use(morgan('dev'))

//db connection config

connectDB();

run application and check:



In import- es6 feature- need to give extension also – db.js

**Db.js**

const mongoose = require("mongoose");

const colors = require("colors");

const connectDB = async () => {

  try {

    const conn = await mongoose.connect(process.env.MONGO\_URL);

    console.log(`connected to db ${conn.connection.host}`);

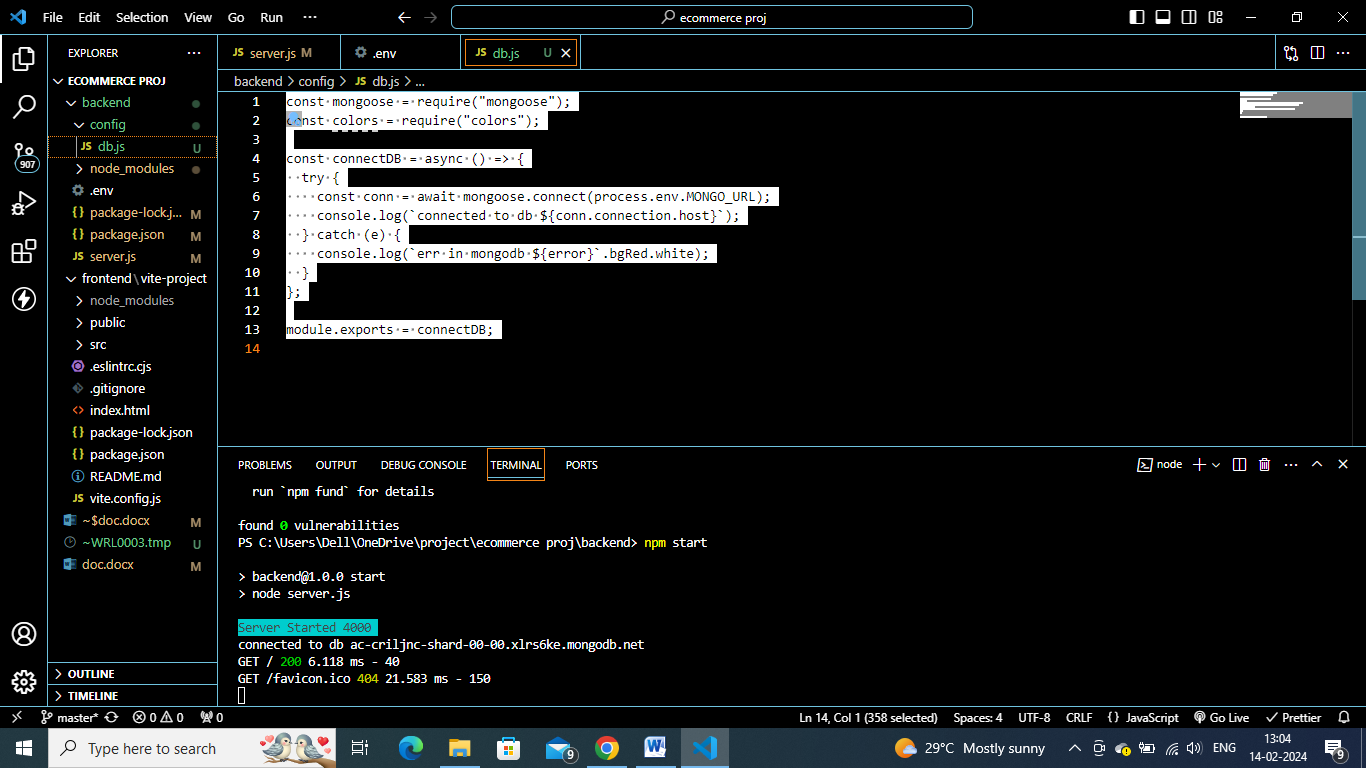
  } catch (e) {

    console.log(`err in mongodb ${error}`.bgRed.white);

  }

};

module.exports = connectDB;



Got get request hit endpoints bcoxz of morgan

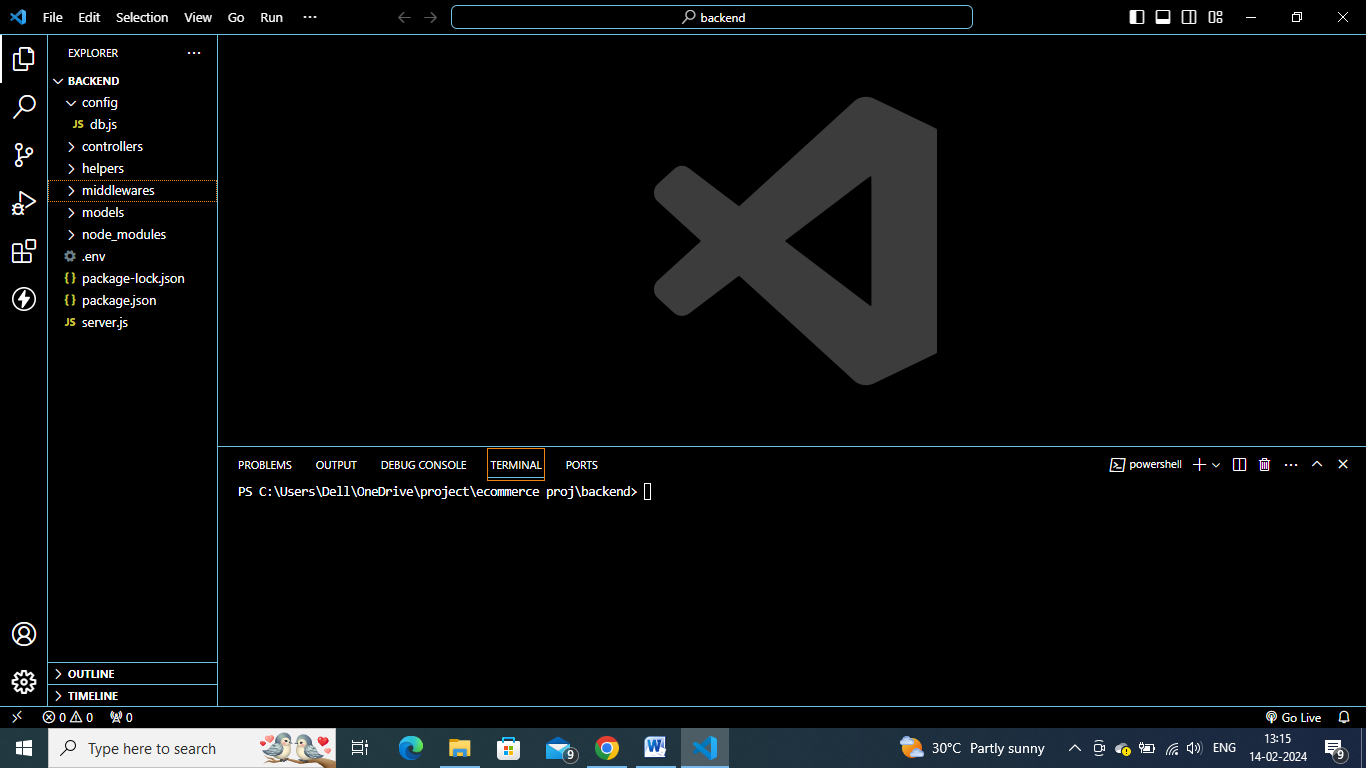
Showing requsta in console- req type,status code ,

In production, it has no need, we will remove this in production

**Video 4:**

**Mvc arch and hasing:**

**Create models,controllers,**helpers,middlewares fodler , so that all code will be separated , easy to understand and to debug easily



**creating usermodel**

**models>userModel.j**s

import mongoose, create schema and model

const mongoose = require("mongoose");

const userSchema=new mongoose.Schema({

})

const userModel= mongoose.model('users',userSchema)

module.exports=userModel

to add timestamp on each creation of user

const mongoose = require("mongoose");

const userSchema=new mongoose.Schema({

    name:{

        type:String,

        required:true,

        trim:true

    },

    email:{

        type:String,

        required:true,

        unique:true

    },

    password:{

        type:String,

        required:true,

    },

    phone:{

        type:String,

        required:true

    },

    address:{

        type:String,

        required:true

    },

    role:{

        type:Number,

        default:0

    }

},{timestamps:true})

const userModel= mongoose.model('users',userSchema)

module.exports=userModel

successfully created model, now creating routes for this.

**Routes:** lets first create register route,

Create route folder> authRoute.js

**Authroute.js:**

We import express and create routes using it.

const express = require("express");

//router object

const router = express.Router();

//routing

//REGISTER || Method post

router.post("/register", registerController);

module.exports = router;

now we will create registercontroller

inside controller folder->create authController.js

const registerController = () => {};

module.exports = { registerController };

as in future we can export more functions so exported as a obj.

**importing registercontroller in authRoute.js**

const { registerController } = require("../controllers/authController");

to check whether it is working or not

import this route in main file- **server**.js

**server.js:**

const authRoute = require("./routes/authRoute");

//routes

app.use('/api/v1/auth',authRoute);

//creating rest api

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

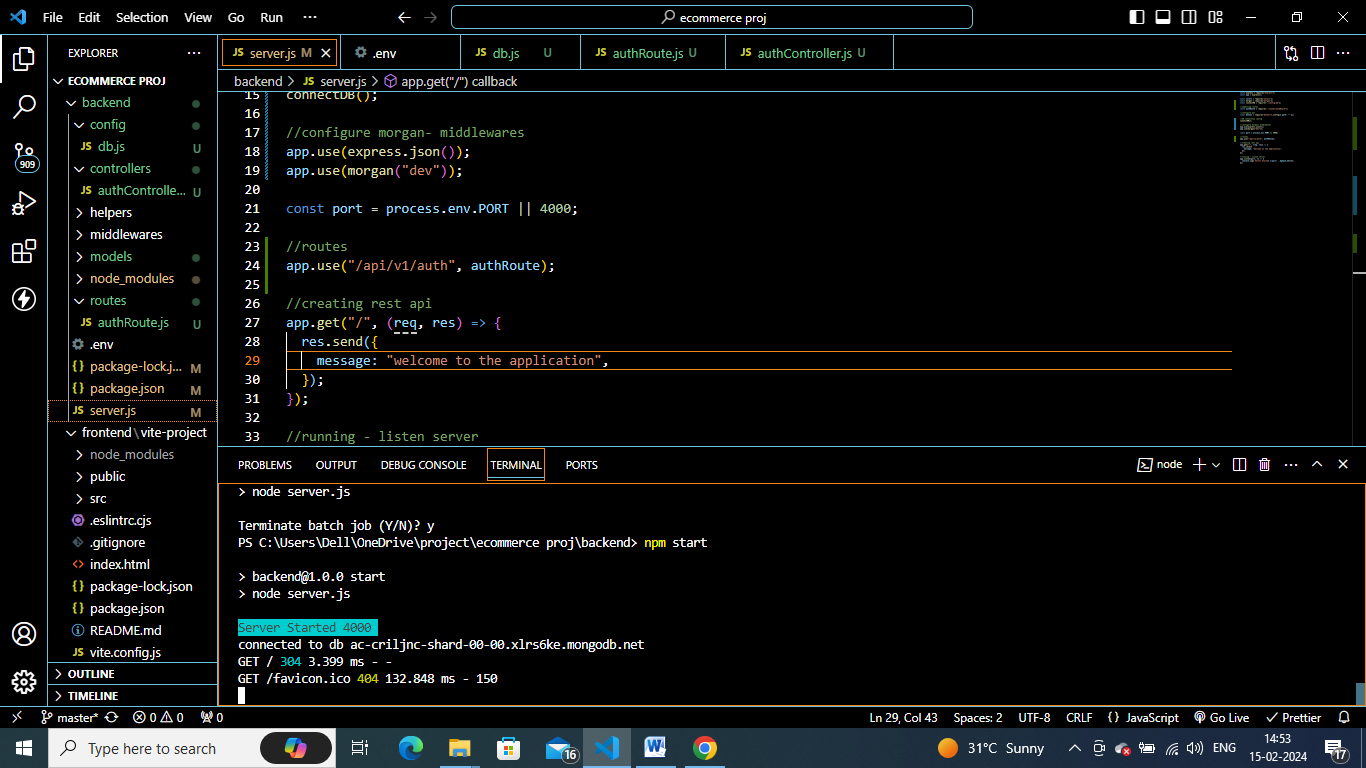
  res.send({

    message: "welcome to the application",

  });

});

Run and check



Server running successfully on port 4000.

For hashing password – install bcrypt and create helper method for this, authhelper.js

Npm I bcrypt

**In helper folder- create authHelper.js**

In this file, crete 2 folders- to encrypt and compare/dcrypt

By using bcrypt.hash() – we can hash the password

Bcrypt.hash() takes 2 args- pswd and saltrounds

const bcrypt=require('bcrypt');

export const hashfunc=async(password)=>{

    try{

        const saltrounds=10;

        const hashedPassword = await bcrypt.hash(password,saltrounds)

    }

    catch(err){

        console.log(err)

    }

}

Now creating another func for comparing password.

export const comparePassword= async(password,hashpassword) =>{

    return  bcrypt.compare(password,hashpassword)

}

We using here inbuilt func bcrypt.compare

**Day 5:**implementing register controller

const registerController = async (req, res) => {

  try {

  } catch (err) {

    console.log(err);

    res.status(500).send({

      success: false,

      message: "Error in Registration",

      error

    });

  }

};

module.exports = { registerController };

now writing actual try part.

Import user model and take data getting from req.body

Perform validations.

const userModel = require("../models/userModel");

const registerController = async (req, res) => {

  try {

    //taking input data

    const { name, email, password, phone, address } = req.body;

    //validations

    if (!name) {

      return res.send({ error: "Name is Required" });

    }

    if (!email) {

      return res.send({ error: "Email is Required" });

    }

    if (!password) {

      return res.send({ error: "Password is Required" });

    }

    if (!phone) {

      return res.send({ error: "Phoneno is Required" });

    }

    if (!address) {

      return res.send({ error: "Address is Required" });

    }

  } catch (err) {

    console.log(err);

    res.status(500).send({

      success: false,

      message: "Error in Registration",

      err,

    });

  }

};

Then check if already it is an existing user   
after checking for existing user, writing code for registration ,

const { hashfunc } = require("../helpers/authHelper");

const userModel = require("../models/userModel");

const registerController = async (req, res) => {

  try {

    //taking input data

    const { name, email, password, phone, address } = req.body;

    //validators

    if (!name) {

      return res.send({ error: "Name is Required" });

    }

    if (!email) {

      return res.send({ error: "Email is Required" });

    }

    if (!password) {

      return res.send({ error: "Password is Required" });

    }

    if (!phone) {

      return res.send({ error: "Phoneno is Required" });

    }

    if (!address) {

      return res.send({ error: "Address is Required" });

    }

    //check  user

    const existingUser = await userModel.findOne({ email });

    //check for esisting user

    if (existingUser) {

      return res.status(200).send({

        success: true,

        message: "you already Registred!! Please Login",

        error,

      });

    }

} catch (err) {

    console.log(err);

    res.status(500).send({

      success: false,

      message: "Error in Registration",

      error,

    });

  }

};

module.exports = { registerController };

//registering user

    //check for esisting user

    if (existingUser) {

      return res.status(200).send({

        success: true,

        message: "you already Registred!! Please Login",

      });

    }

    //register user

    const hashedPassword = await hashfunc(password);

    //saving to db

    const user=new userModel({name,email,phone,address,password:hashedPassword}).save()

    res.status(201).sned({

      success:true,

      message:'User Registered Successfully',

      user

    })

  } catch (err) {

    console.log(err);

    res.status(500).send({

      success: false,

u can even remove the validations here and can keep in clientside also.

In this project we are keeping validations both on client and backend side,

**Testing APIS:**

Testing the above apis in postman

Keep server running to check output in postman

app.use("/api/v1/auth", authRoute);

post request at Localhost:port/api/v1/auth/register in postman.

**authHelper.js**

const bcrypt = require("bcrypt");

const hashfunc = async (password) => {

  try {

    const saltrounds = 10;

    const hashedPassword = await bcrypt.hash(password, saltrounds);

    return hashedPassword;

  } catch (err) {

    console.log(err);

  }

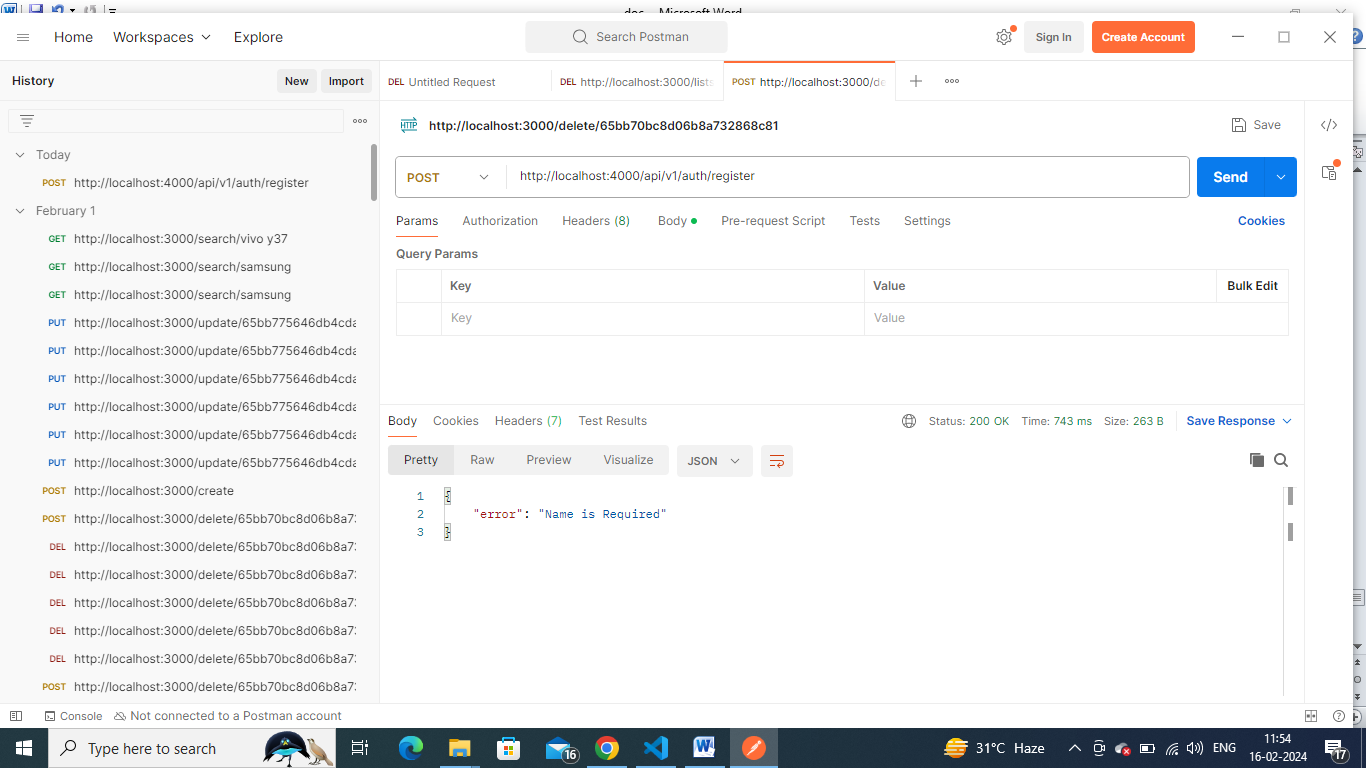
};

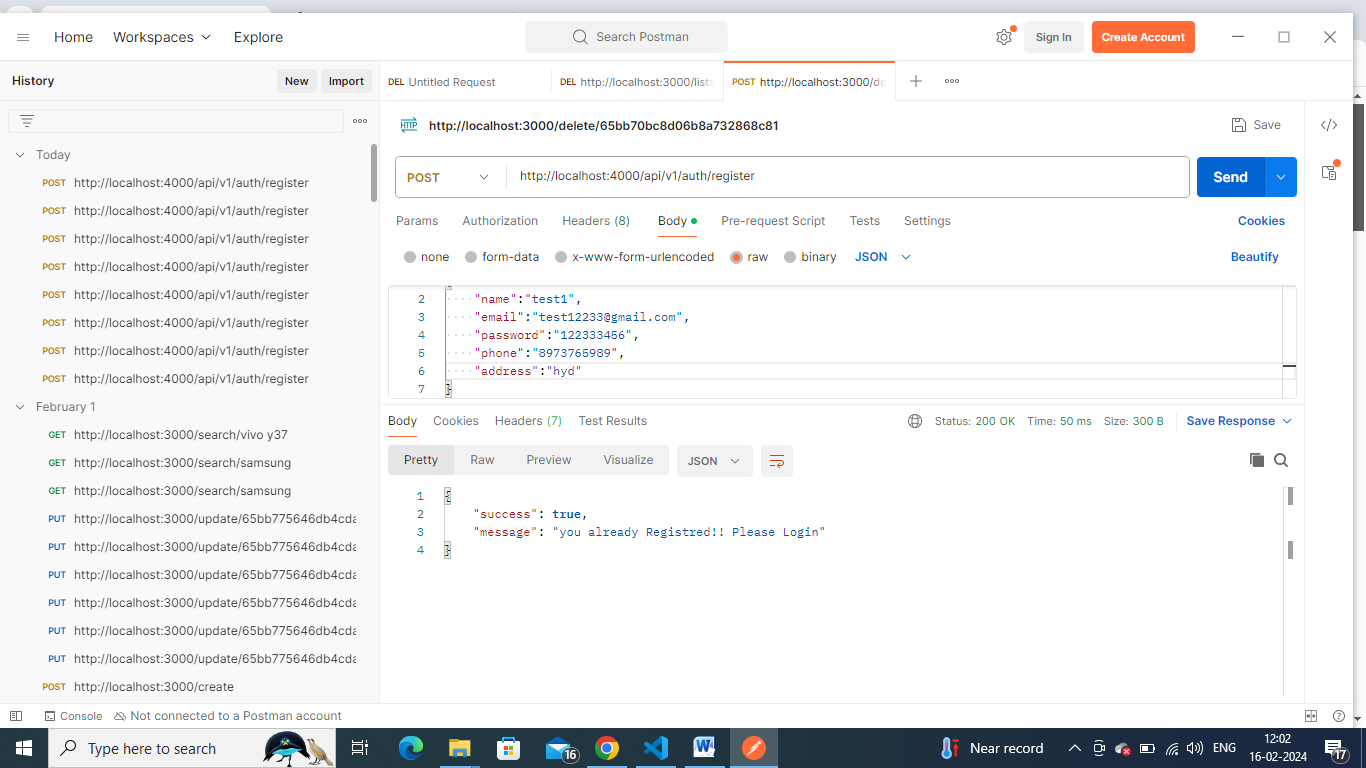
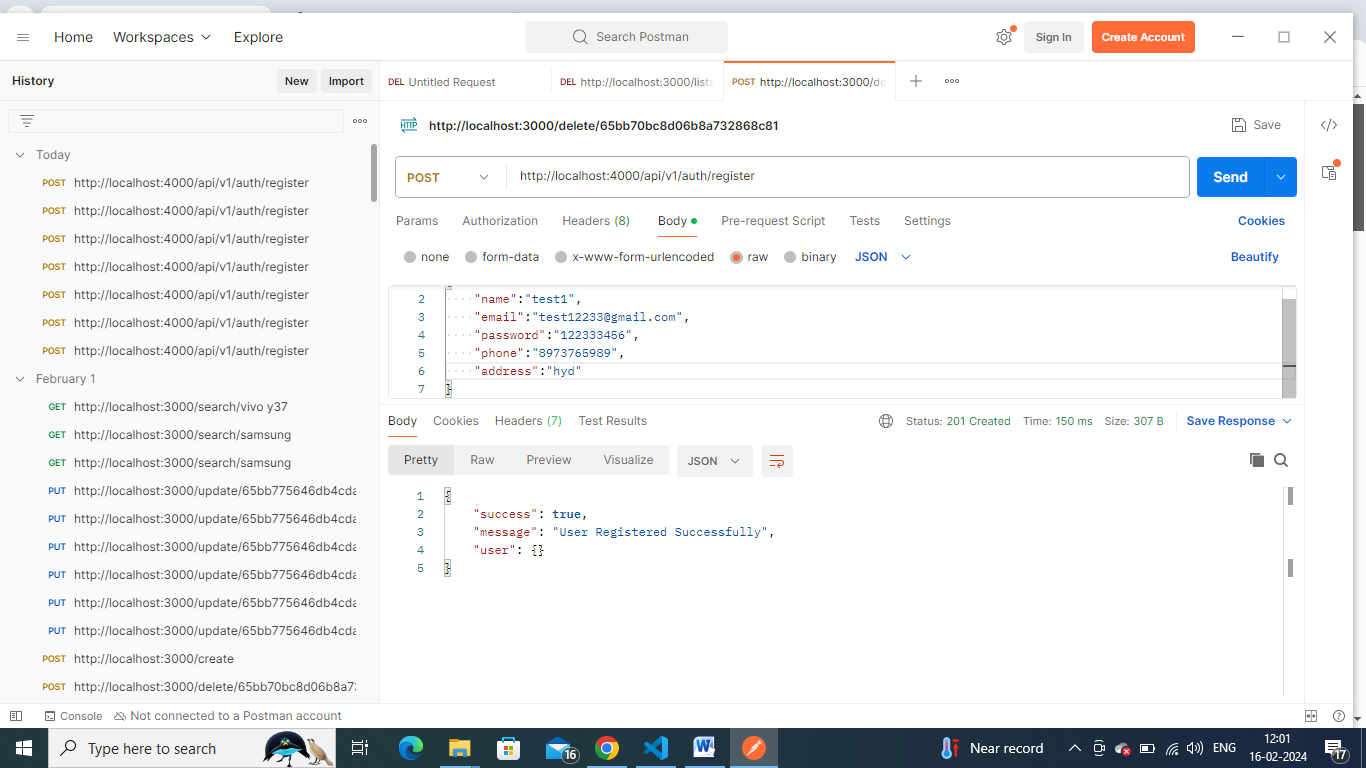
const comparePassword = async (password, hashpassword) => {

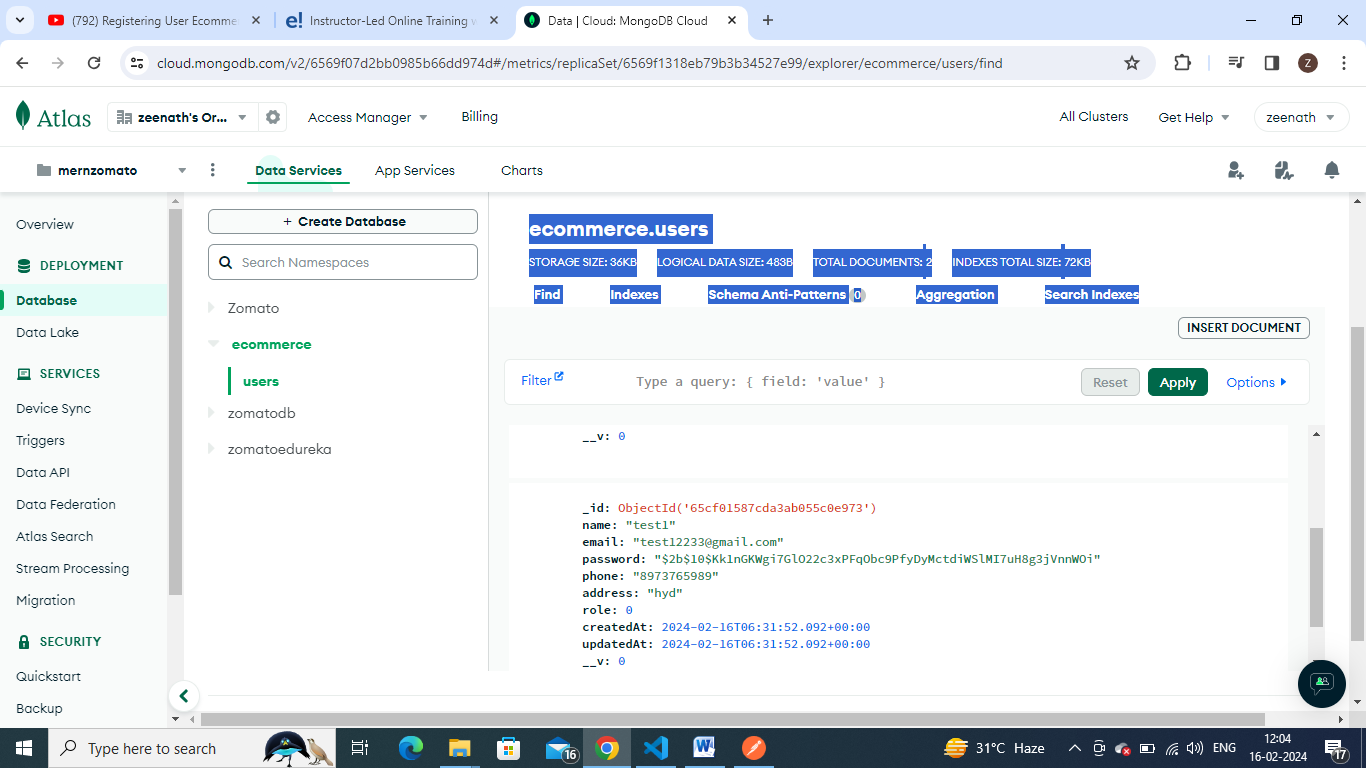
  return bcrypt.compare(password, hashpassword);

};

module.exports = { hashfunc, comparePassword };







    //saving to db

    const user =await  new userModel({

      name,

      email,

      phone,

      address,

      password: hashedPassword,

    }).save();

    res.status(201).send({

      success: true,

      message: "User Registered Successfully",

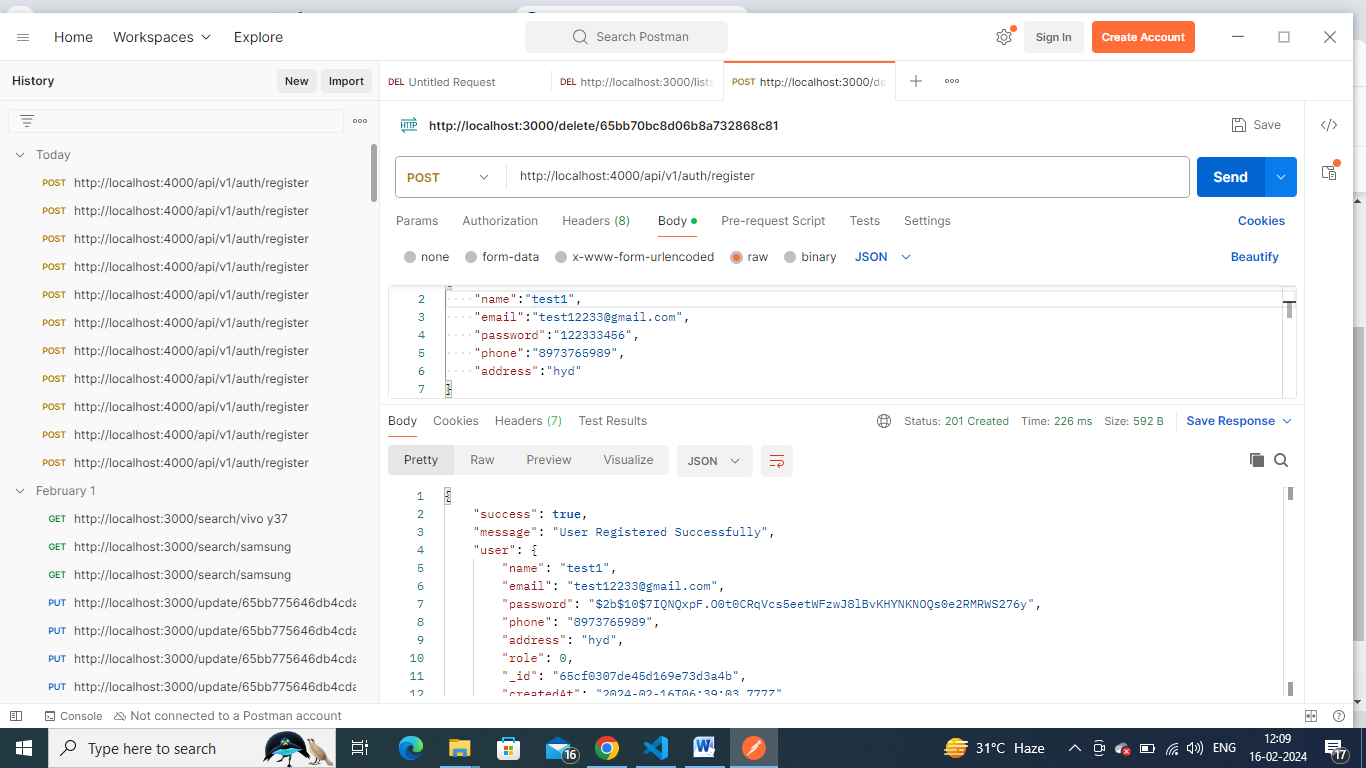
      user,

    });

  }

Add await here while saving data as we are not getting user obj in postman response

Now delete the record from mongodb and check



Got user obj in response.

To make it more secure , we can use json webtoken.

**Jsonwebtoken:**

Npm I jsonwebtoken

Import this package in server.js

const JWT=require('jsonwebtoken')

for this lets create a secret key

**.env file:**

JWT\_SECRET= HFGRTYNN!@#123

Creating new route method for login in authcontroller.js, and creating route in authRoute.js

**authRoute.js**

const express = require("express");

const { registerController,loginController } = require("../controllers/authController");

//router object

const router = express.Router();

//routing

//REGISTER || Method post

router.post("/register", registerController);

//LOGIN  || Method post

router.post("/login",loginController);

module.exports = router;

**authController.js**

//POST LOGIN

const loginController = async (req, res) => {};

module.exports = { registerController, loginController };

authController.js

//POST LOGIN

const loginController = async (req, res) => {

  try{

  }

  catch(err){

    console.log(err);

    res.status(500).send({

      success:false,

      message:"Error in login",

      err

    })

  }

};

Adding try block

Taking email and password from user (req.body)

Check if email exists – if yes – match the password .

Else say to register

//POST LOGIN

const loginController = async (req, res) => {

  try{

    const {email,password}=req.body;

    //validation

    if(!email || !password){

      return res.status(404).send({

        success:false,

        message:'Invalid Email or Password'

      })

    }

    const match=await comparePassword(password,user.password)

  }

Here to compare password – we need user.password – so first check if user exists

  try{

    const {email,password}=req.body;

    //validation

    if(!email || !password){

      return res.status(404).send({

        success:false,

        message:'Invalid Email or Password'

      })

    }

    //check user

    const user=await userModel.findOne({email});

    if(!user){

      return res.status(404).send({

        success:false,

        message:'Email is not registered'

      })

    }

    const match=await comparePassword(password,user.password);

    if(!match){

      return res.status(200).send({

        success:false,

        message:'Invalid Password'

      })

    }

  }

Next condition is if passwords are also matched- then generating token

  try {

    const { email, password } = req.body;

    //validation

    if (!email || !password) {

      return res.status(404).send({

        success: false,

        message: "Invalid Email or Password",

      });

    }

    //check user

    const user = await userModel.findOne({ email });

    if (!user) {

      return res.status(404).send({

        success: false,

        message: "Email is not registered",

      });

    }

    const match = await comparePassword(password, user.password);

    if (!match) {

      return res.status(200).send({

        success: false,

        message: "Invalid Password",

      });

    }

    //token

    const token = await JWT.sign({ \_id: user.id }, process.env.JWT\_SECRET, {

      expiresIn: "7d",

    });

    res.status(200).send({

      success: true,

      message: "Login Successfull",

      user:{

        name:user.name,

        email:user.email,

        phone:user.phone,

        address:user.address

      },

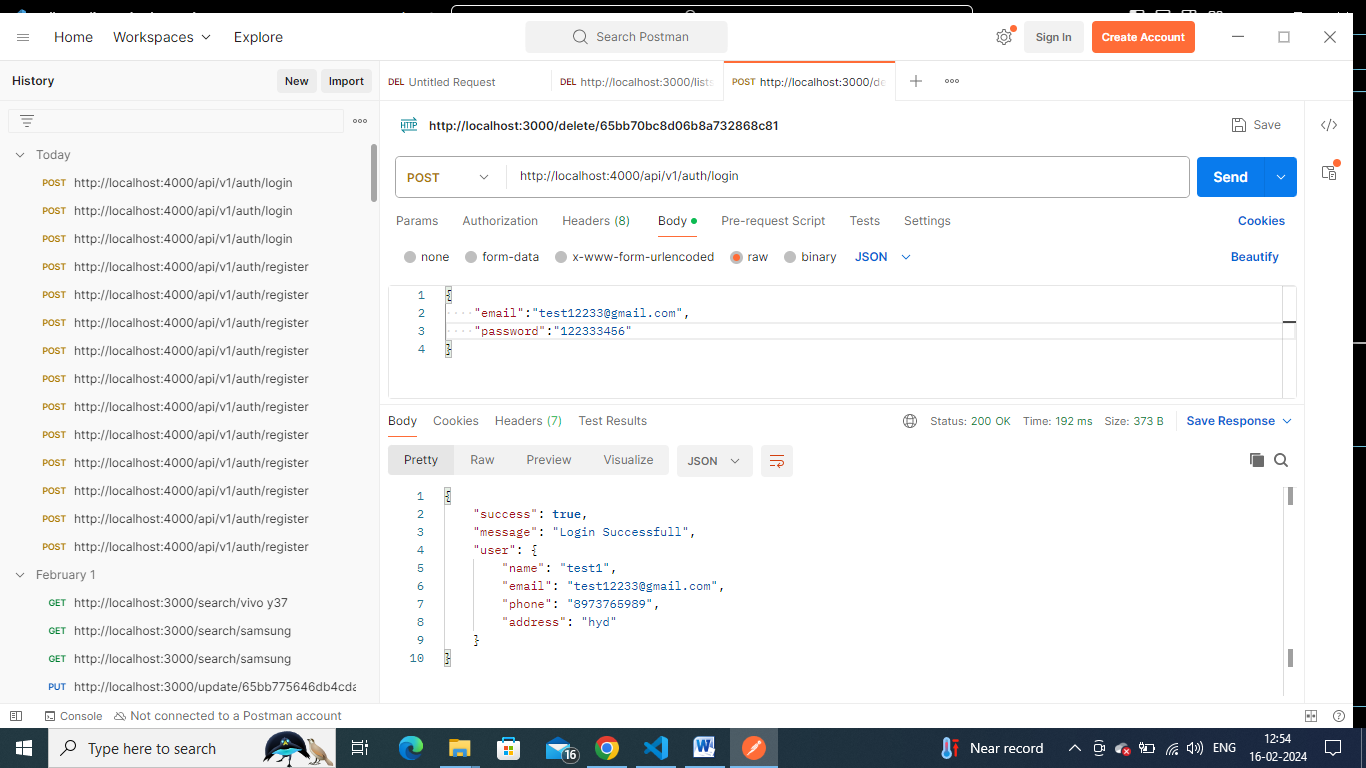
token

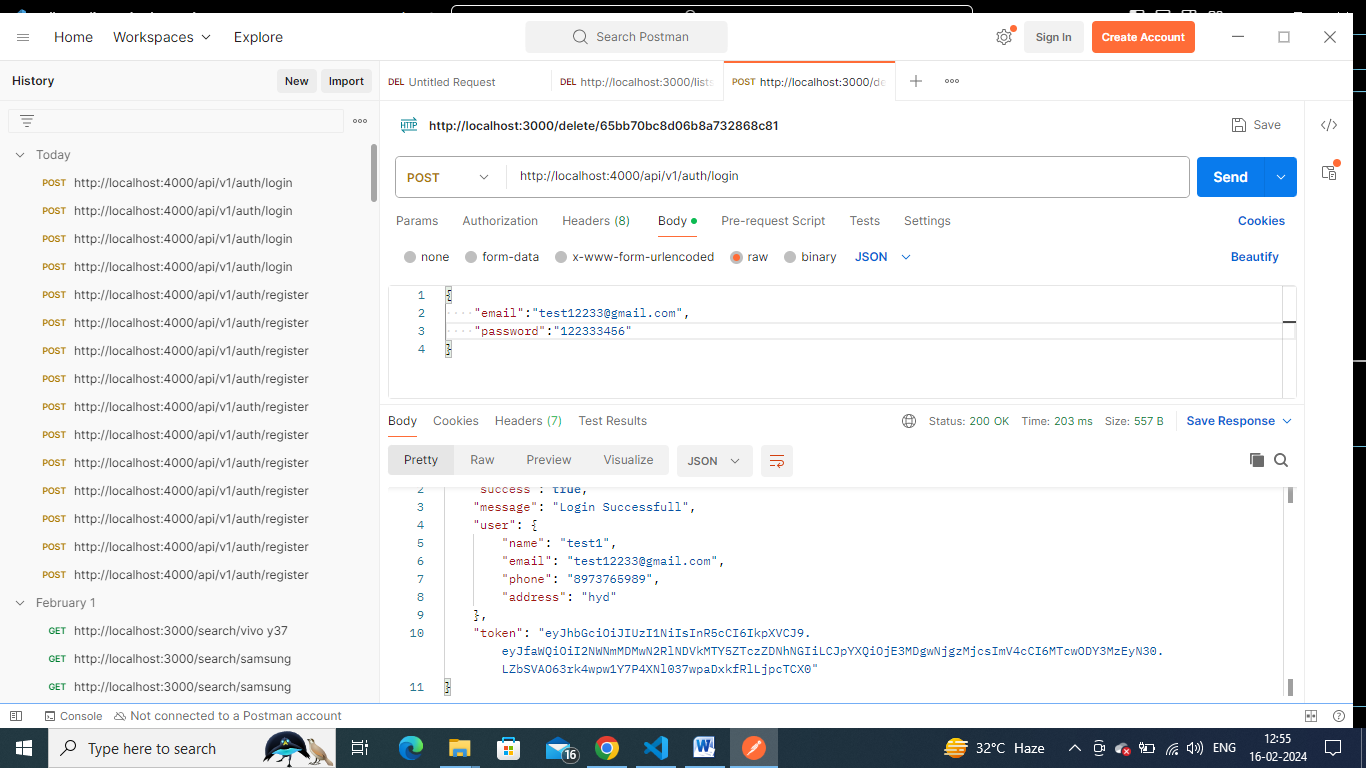
    });

  }

Generated token using jwt.sign() method. And sending user obj in respone- sent all details- name,email,phone,address, but not passing the password.

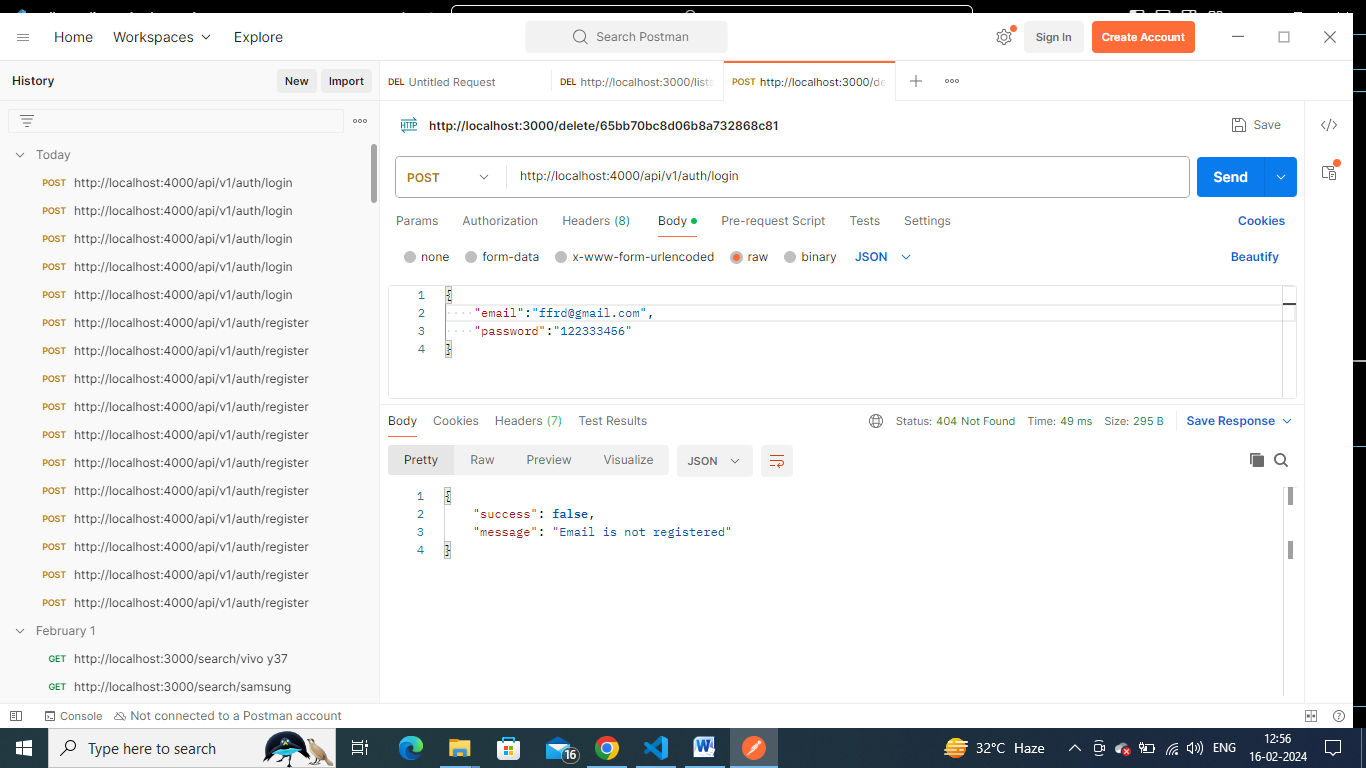
**Testing login endpoint:**

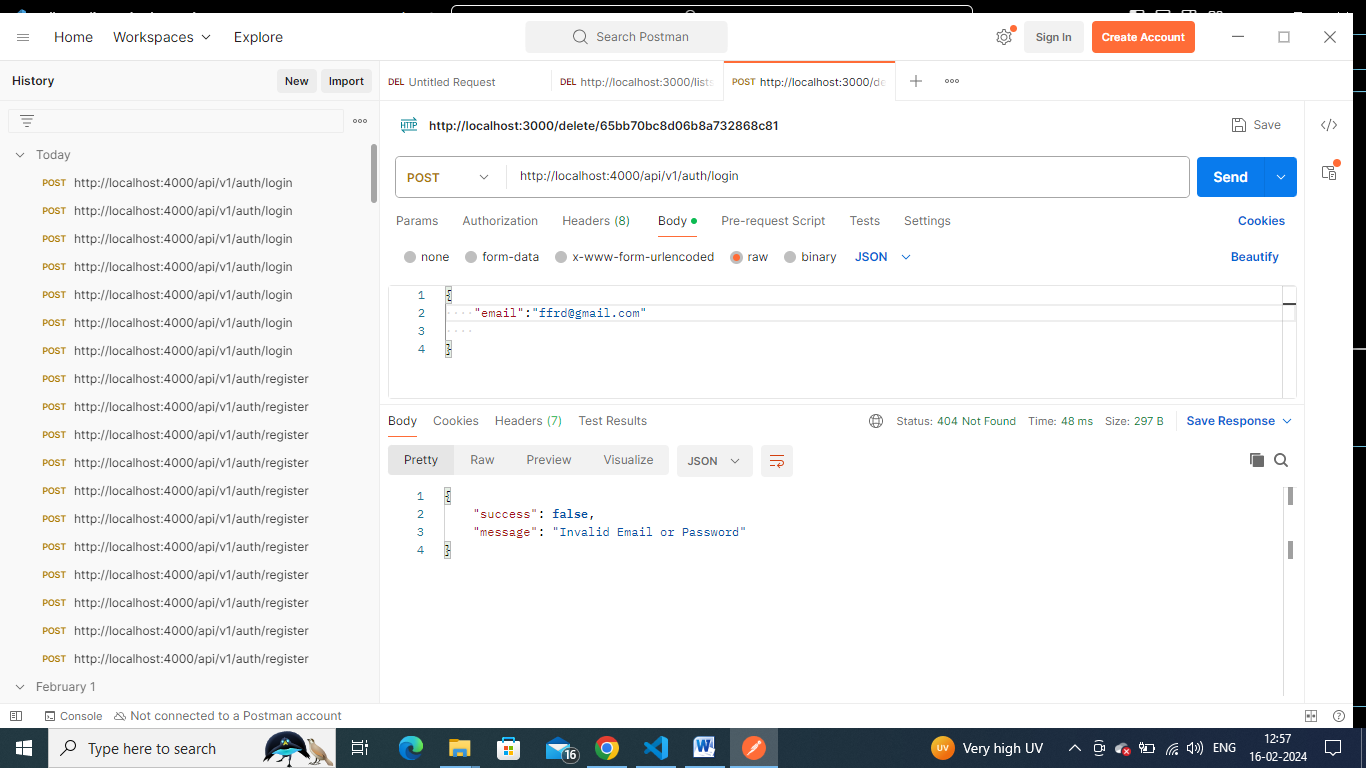




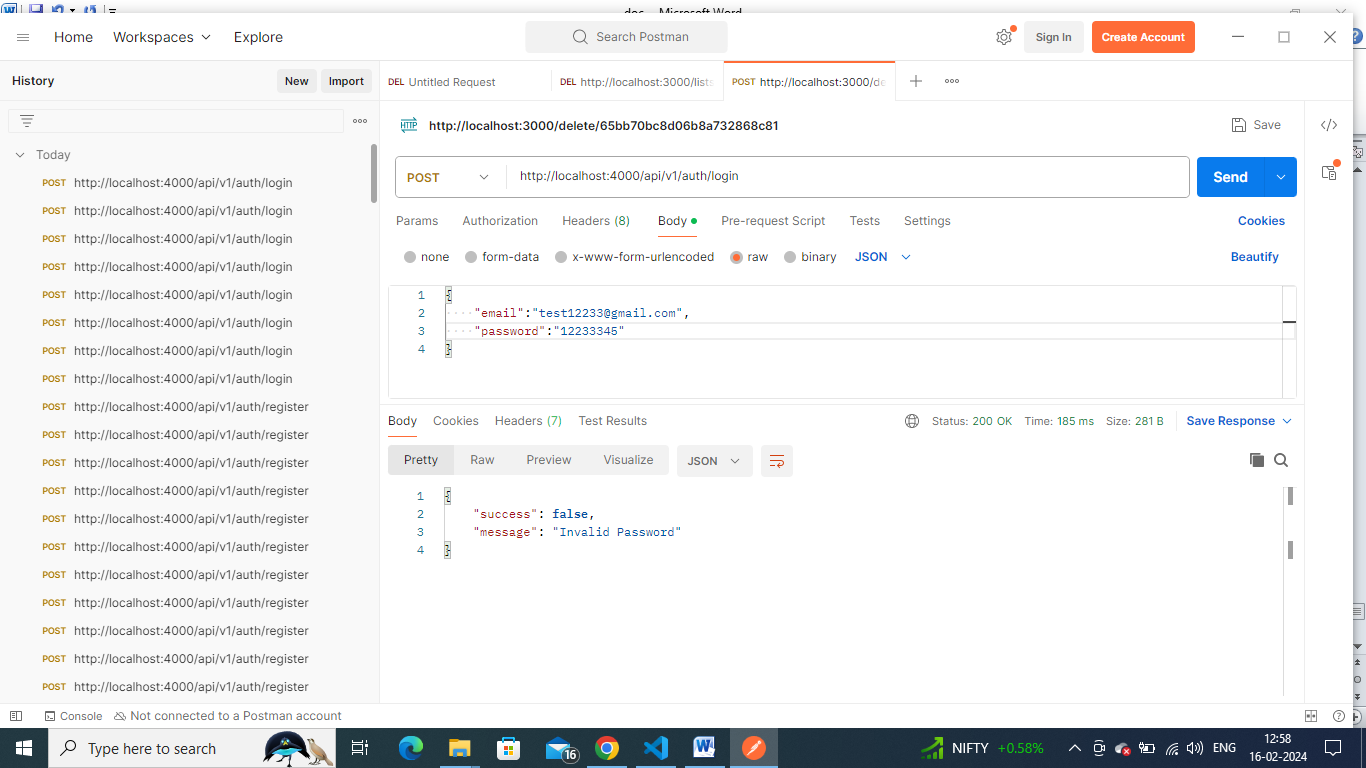
Got token also in response.

If we give another email





Giving correct email but wrong password



By using this token, we can protect our routes by creating a middleware

If token exits, then only we show that route, or else we protect that route

**Protected Middlewares:**

In middlewares folder- create authMiddleware.js

const JWT=require('jsonwebtoken');

//Protected Routes token base

const requireSign= async(req,res,next)=>{

    const decode=JWT.verify(req.)

}

module.exports = requireSign;

Token will be in headers of req, and inside headers , in authorizarization we have token

const JWT=require('jsonwebtoken');

//Protected Routes token base

const requireSign= async(req,res,next)=>{

    try{

        const decode=JWT.verify(req.headers.authorization, process.env.JWT\_SECRET);

        next()

    }

    catch(err){

        console.log(err);

    }

}

Creating a route –

**authRoute.js**

//test routes

router.get('/test',testController);

**authcontroleerjs**

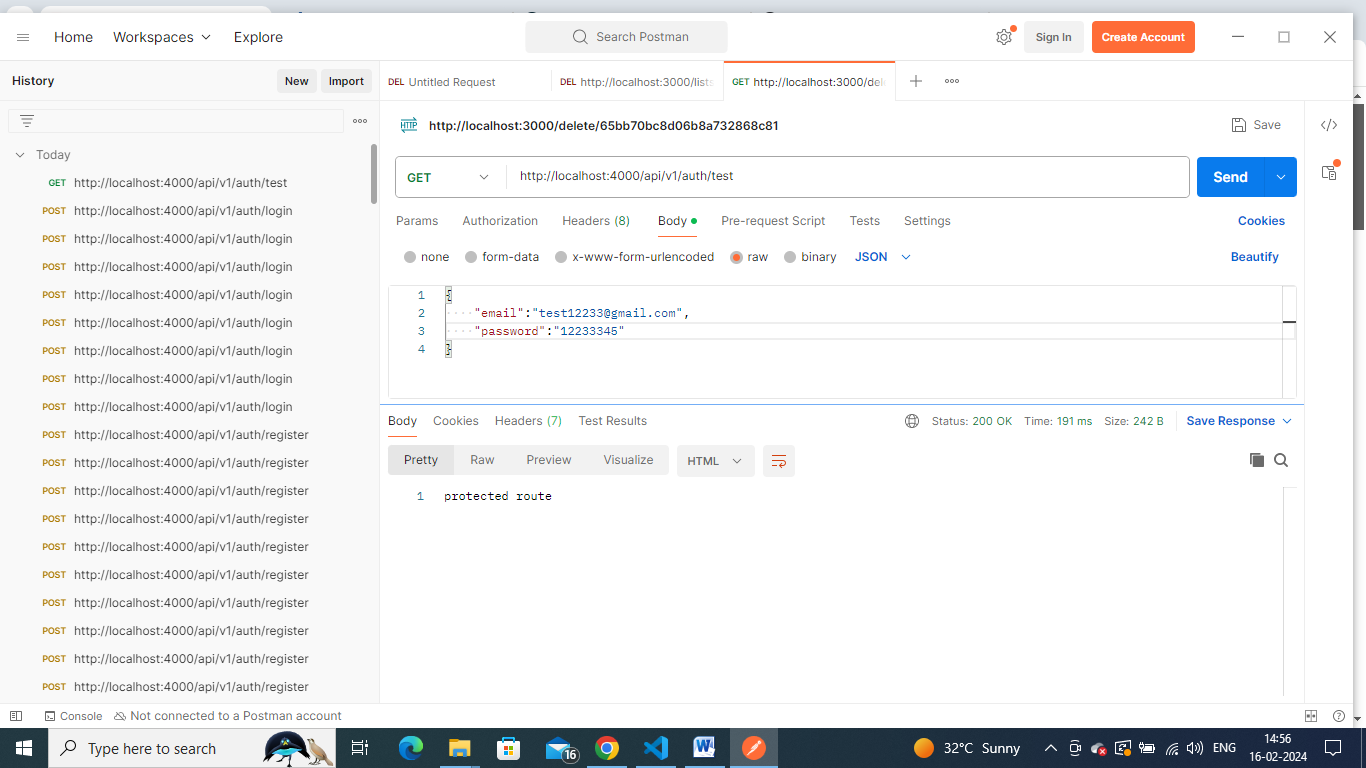
//testController

const testController = (req, res) => {

  res.send('protected route')

};

module.exports = { registerController, loginController,testController };

Run and check

Now add requiresign as a middleware to this route

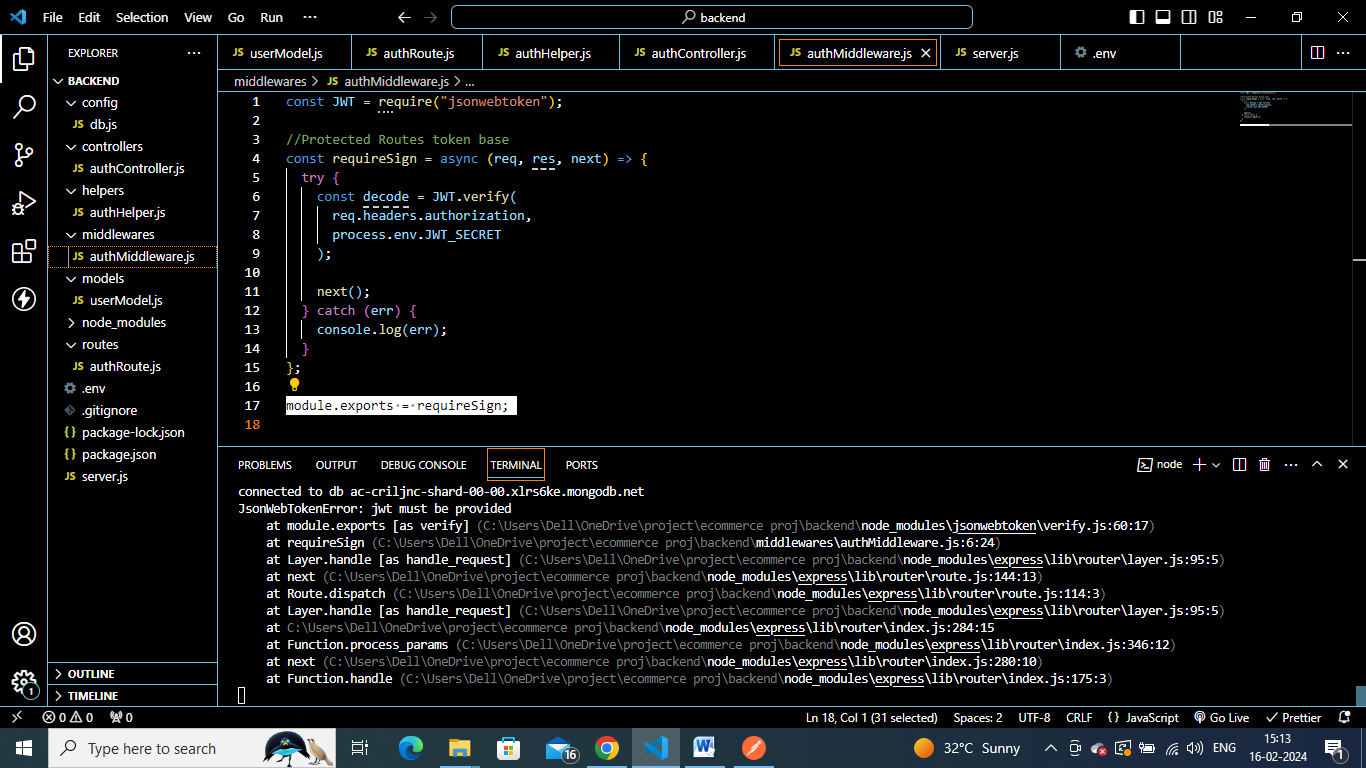
**authRoute.js**

const requireSign = require("../middlewares/authMiddleware");

router.get("/test",requireSign, testController);

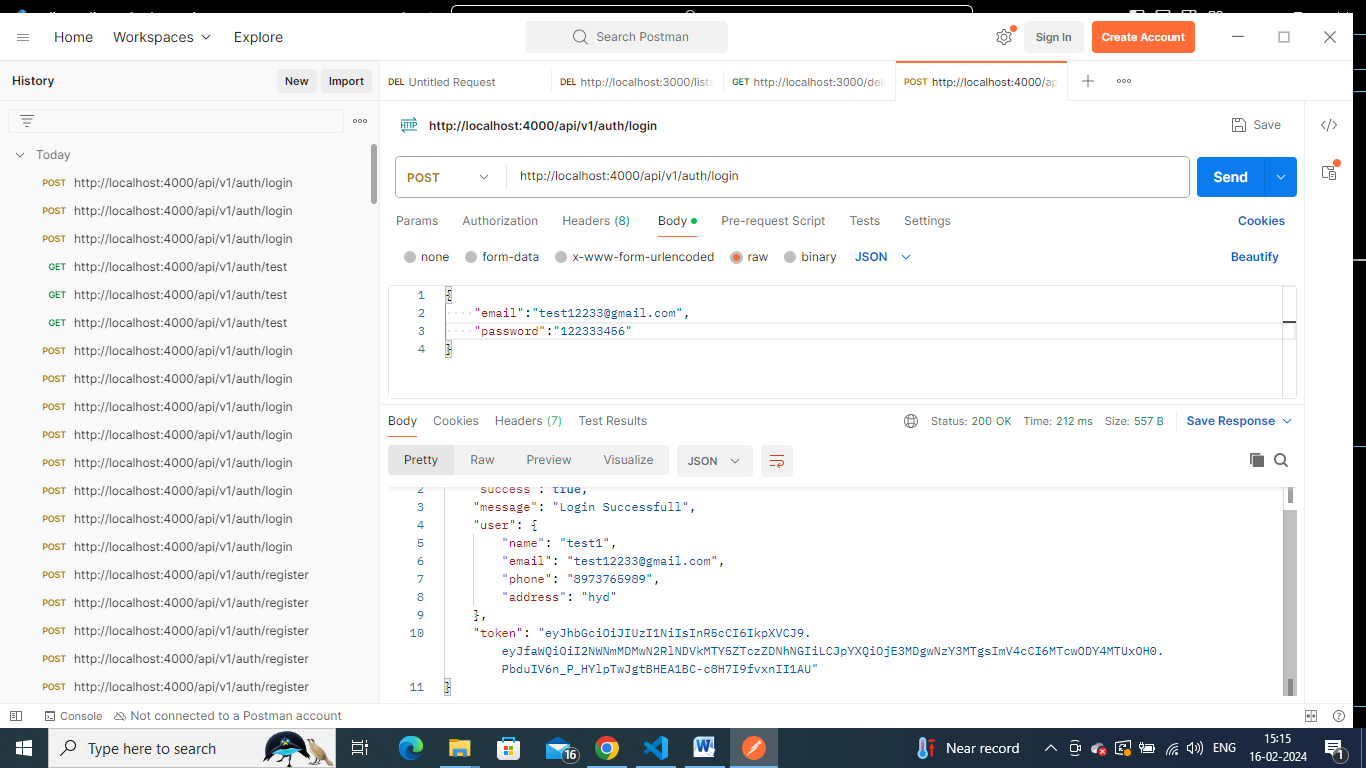


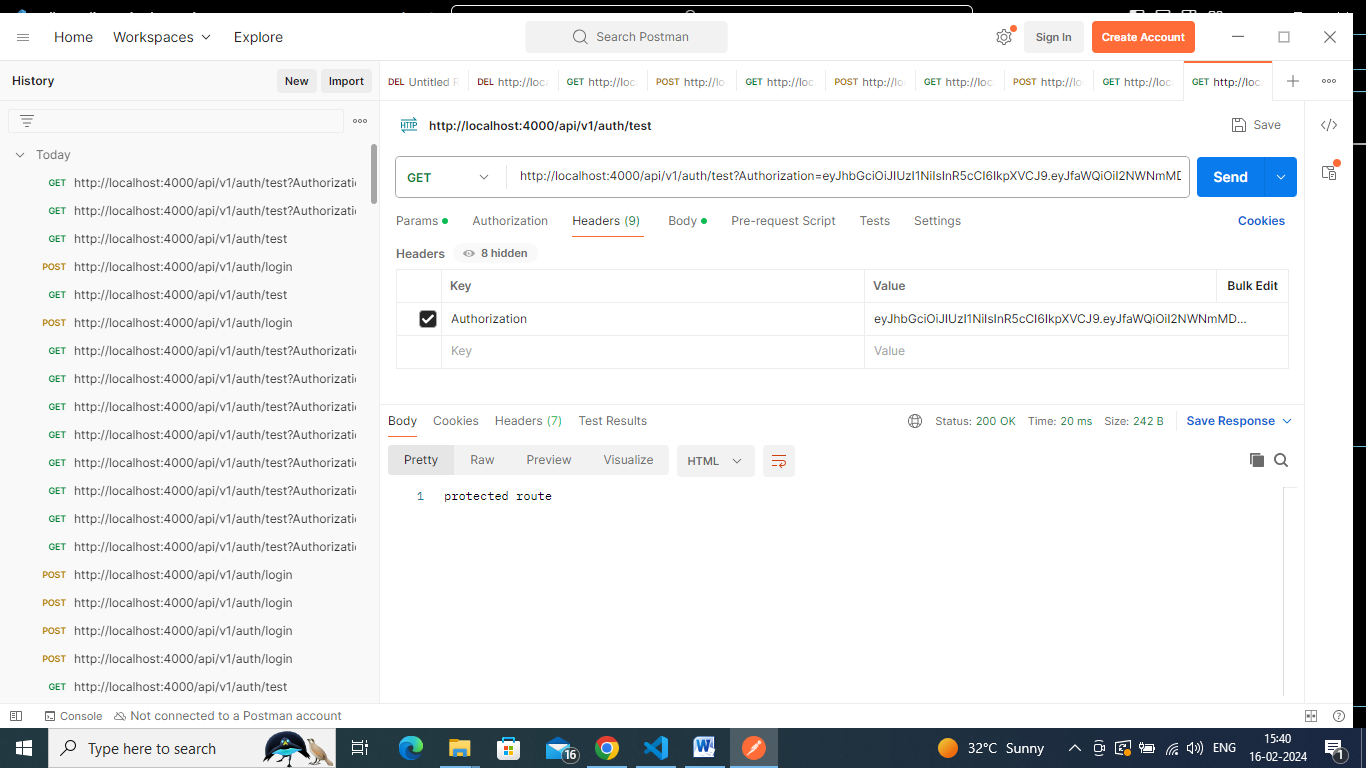
Now see , we did not got output, it got hanged.



In console, we got as jwt must be provided.

So we have to provide token, then only we can see the response, then only next() will get executed.

take this token which got generated while login and send it as a authorization to test route



Pass headers in headers tab- Authorization : token generated at login

Try to provide another token /change token – u wont get output as protected route., we get in console as invalid token.

**Note**:

From the API's point of view, it might be simpler to handle an authorization header instead of a cookie header. The latter might contain numerous cookies, you then have to parse all of them and find the required header.

On the client-side using tokens instead of cookies will work better for cross-domain requests. In some cases, third-party cookies might get dropped.

If the token is sent in the Authorization header, Cross-Origin Resource Sharing (CORS) won't be an issue as it doesn't use cookies.

Using the authorization header is slightly safer compared to cookies, because you are less vulnerable to CSRF attacks. However, in both cases you are vulnerable to XSS attacks.

Ideally, you should not store tokens at all in the browser and instead look at using the BFF pattern as described in this video:

[alert‘OAuth 2 0’; // The impact of XSS on OAuth 2 0 in SPAs](https://www.youtube.com/watch?v=lEnbi4KClVw&t=2564s)