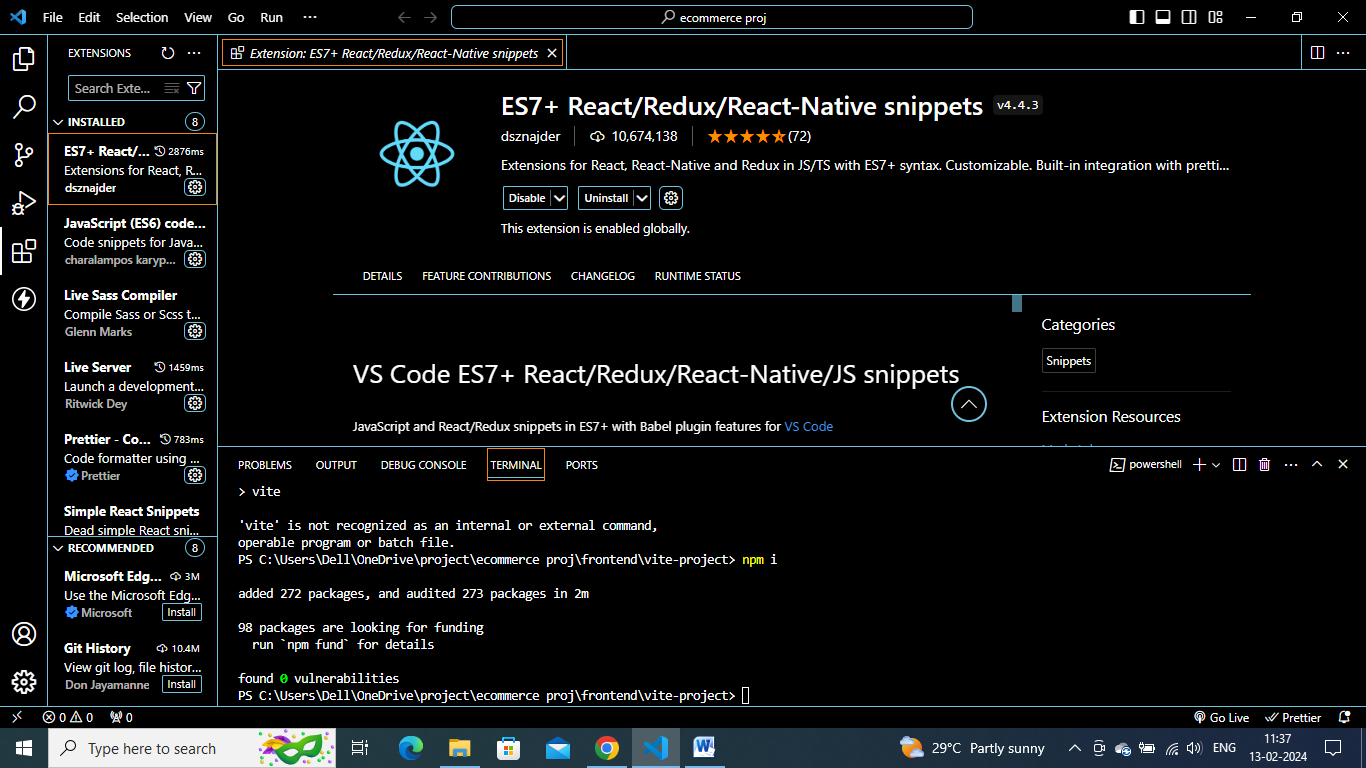
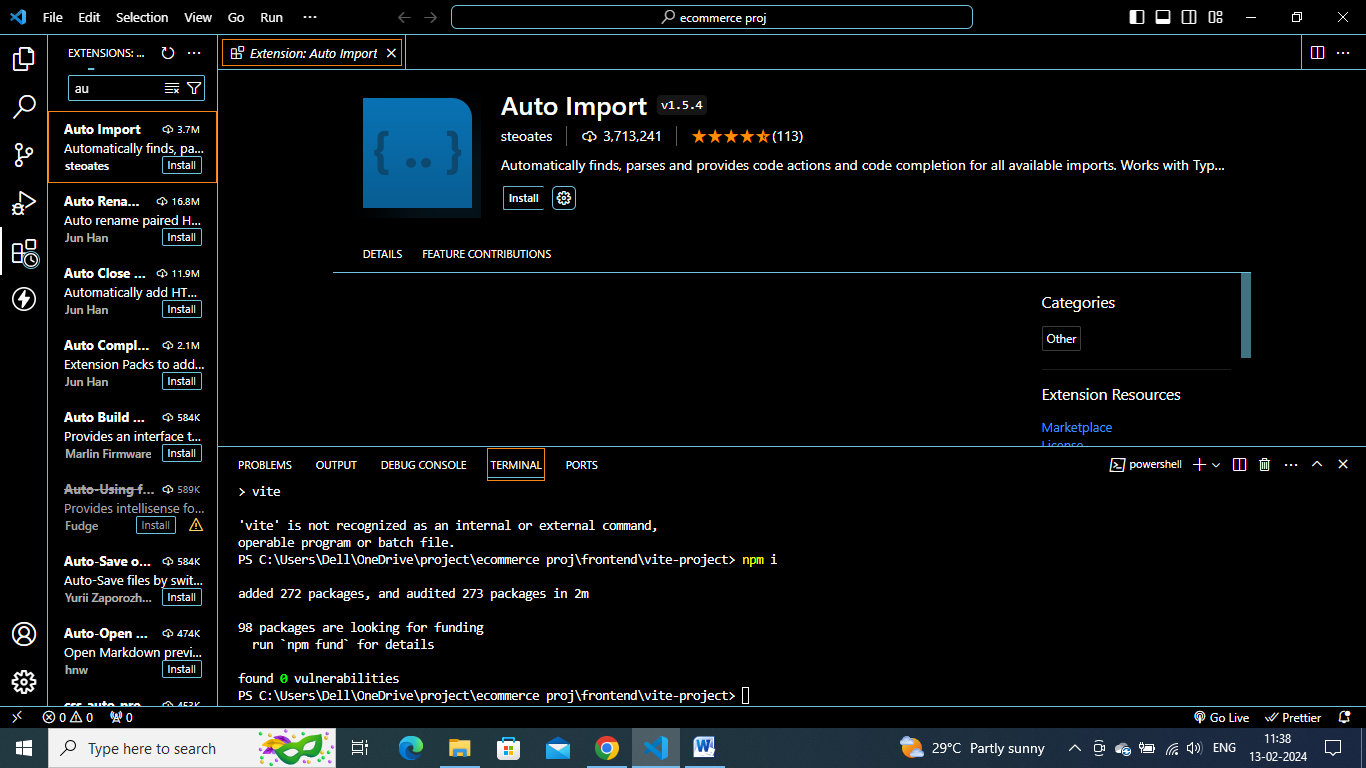
**Creating backend api first:**

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

Auto import –es6 ts,jsx.

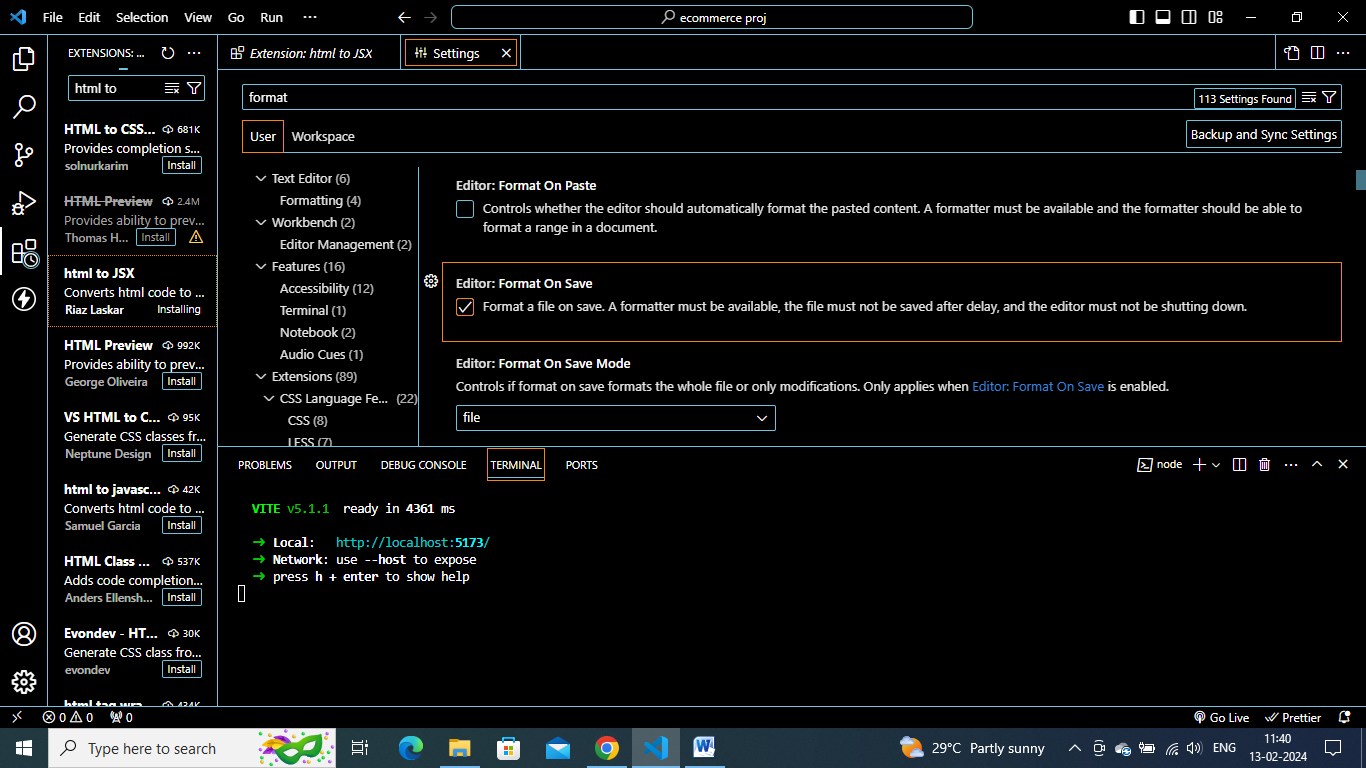
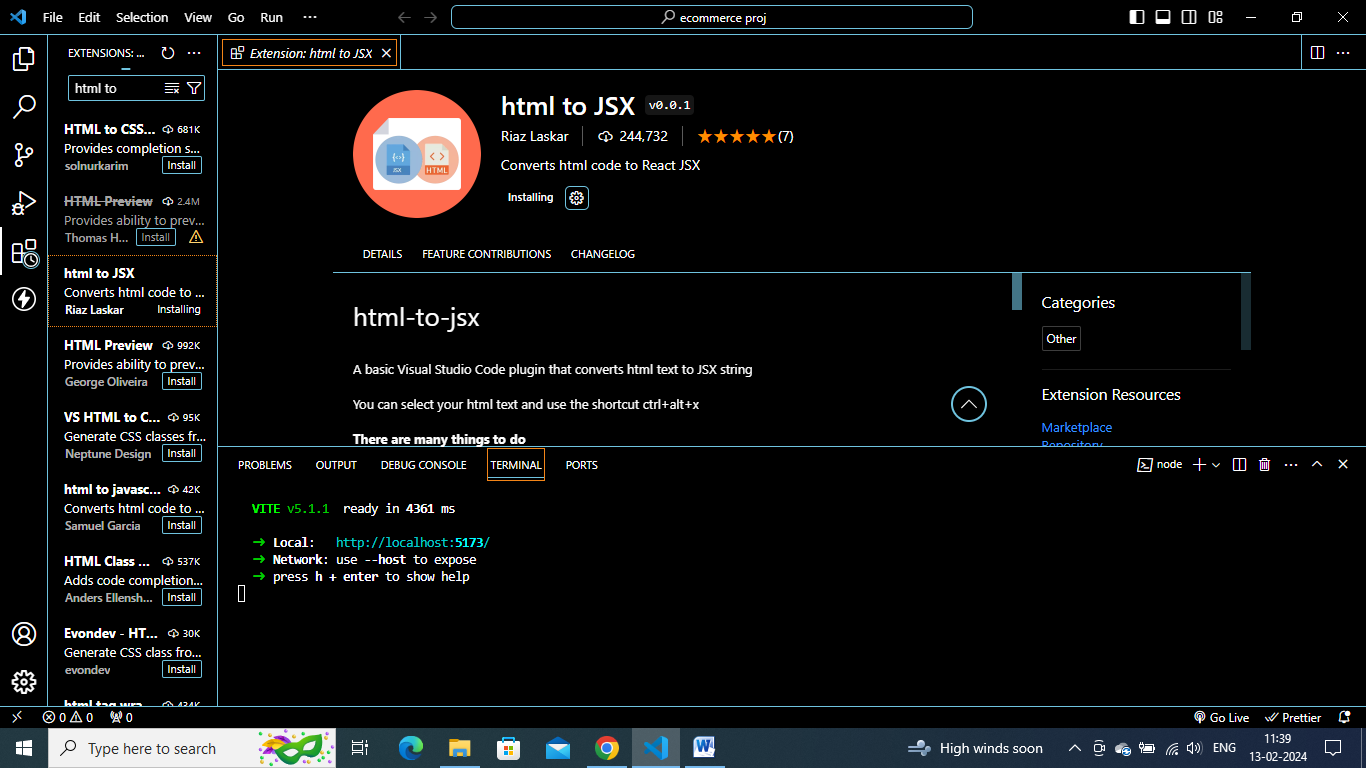
Dotenv



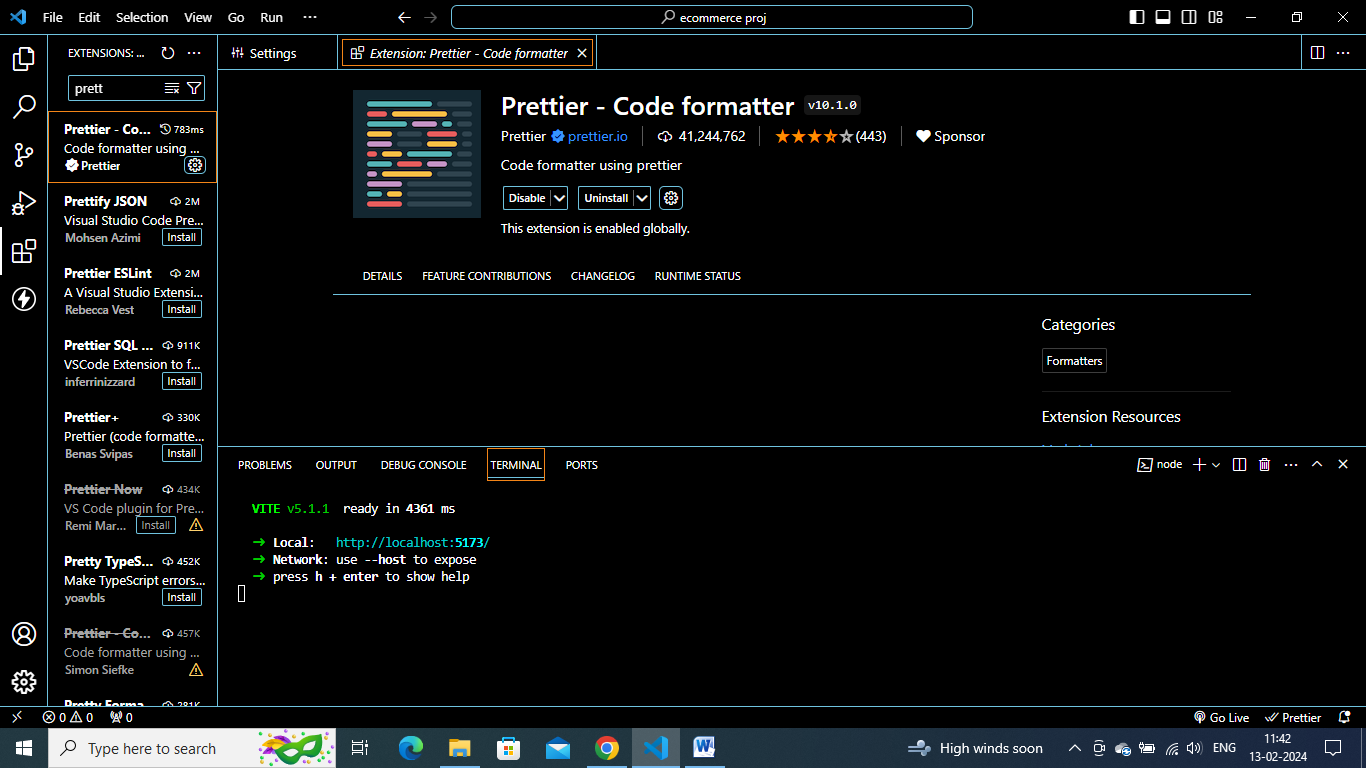




Html to jsx extension

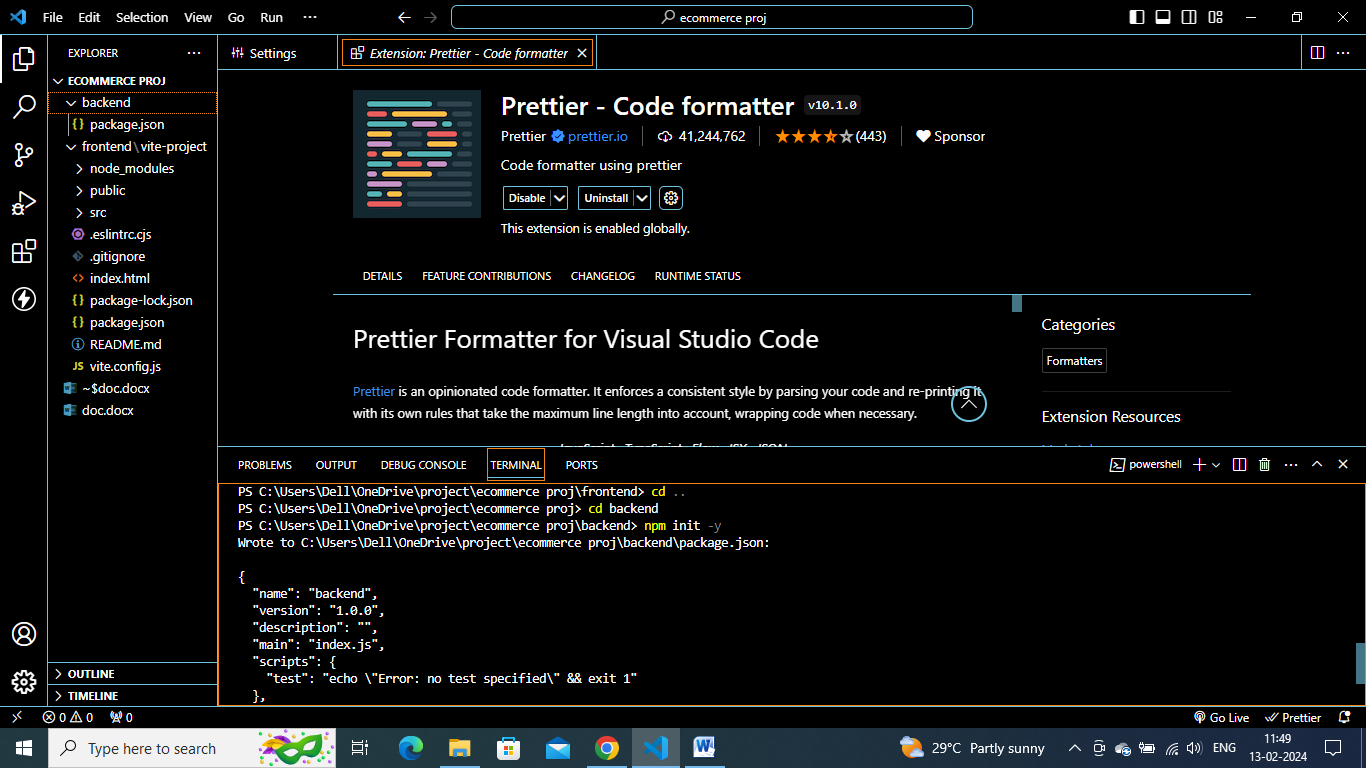


In settings – format search – format on save checked.

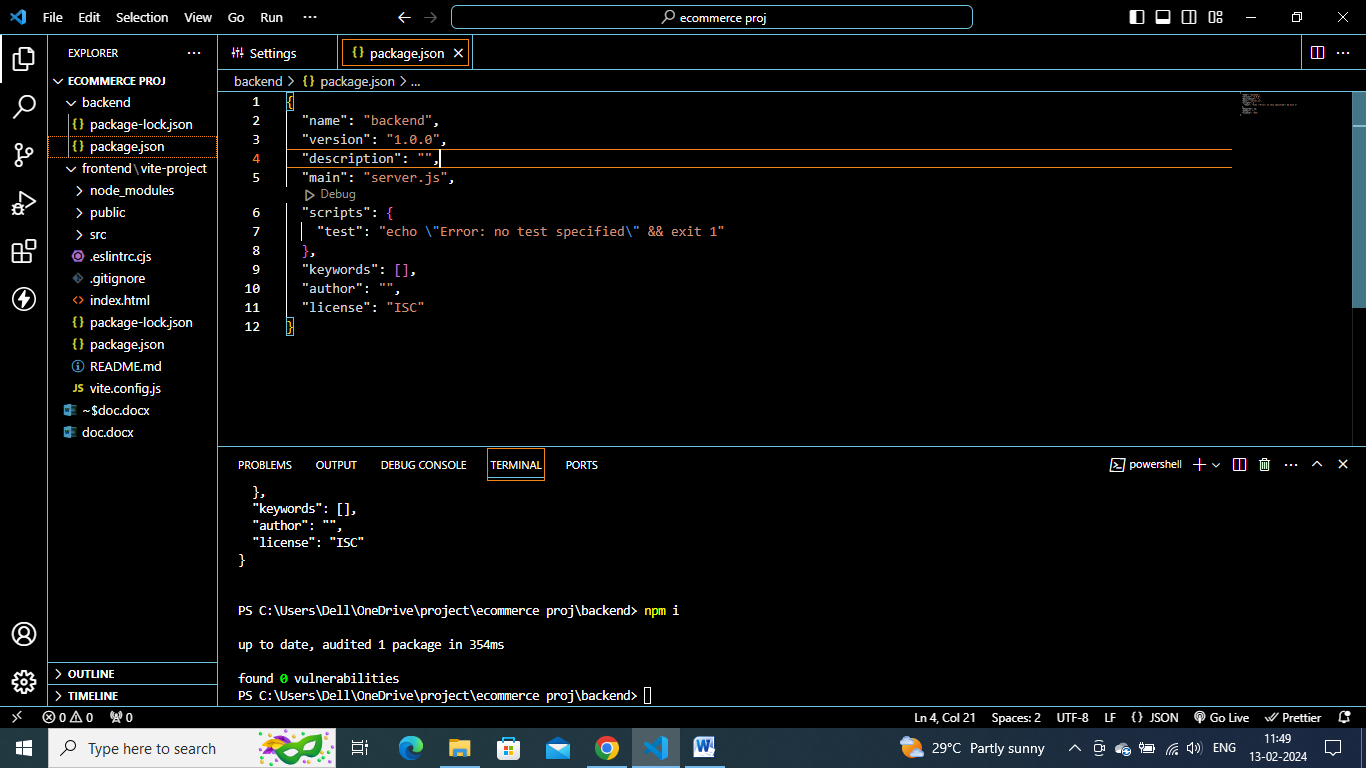


Create backend named folder

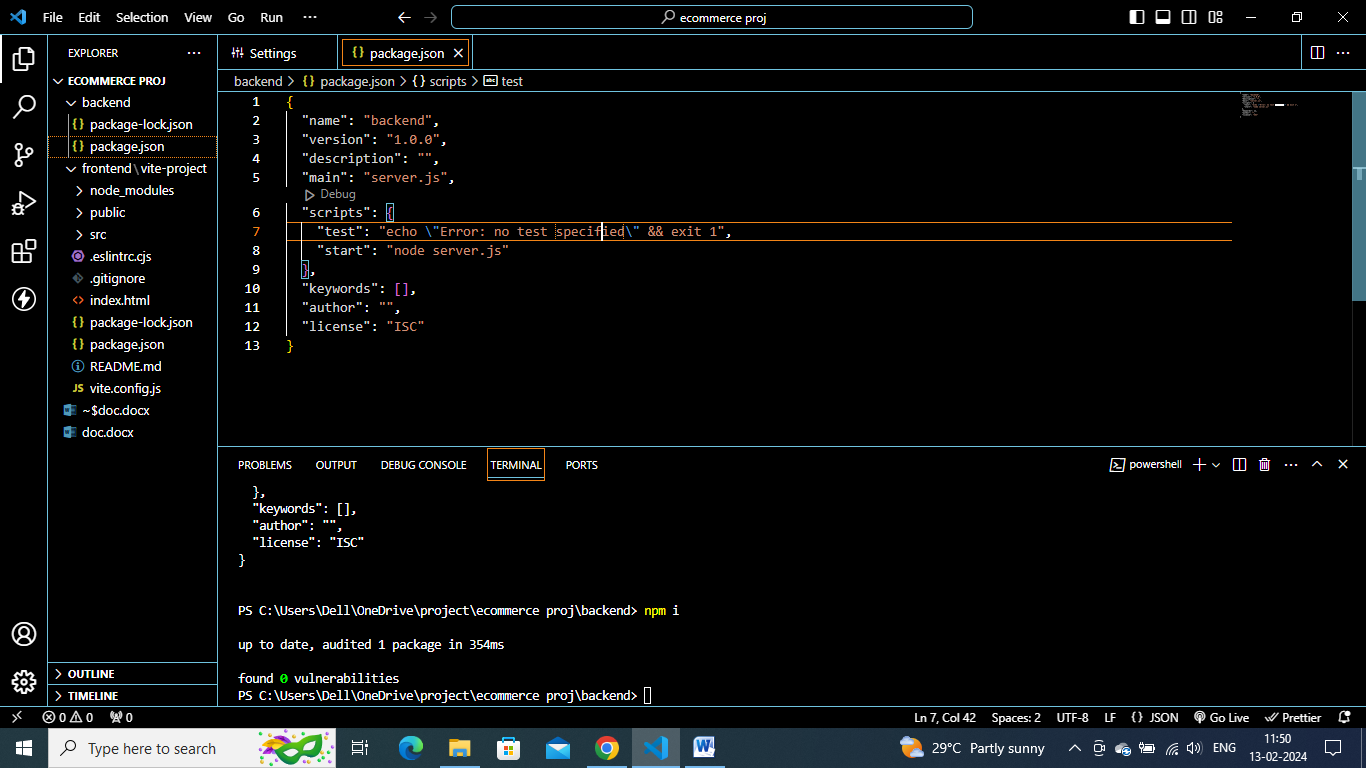
Do npm init –y inside it.



Observe a packge.json will be created .



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

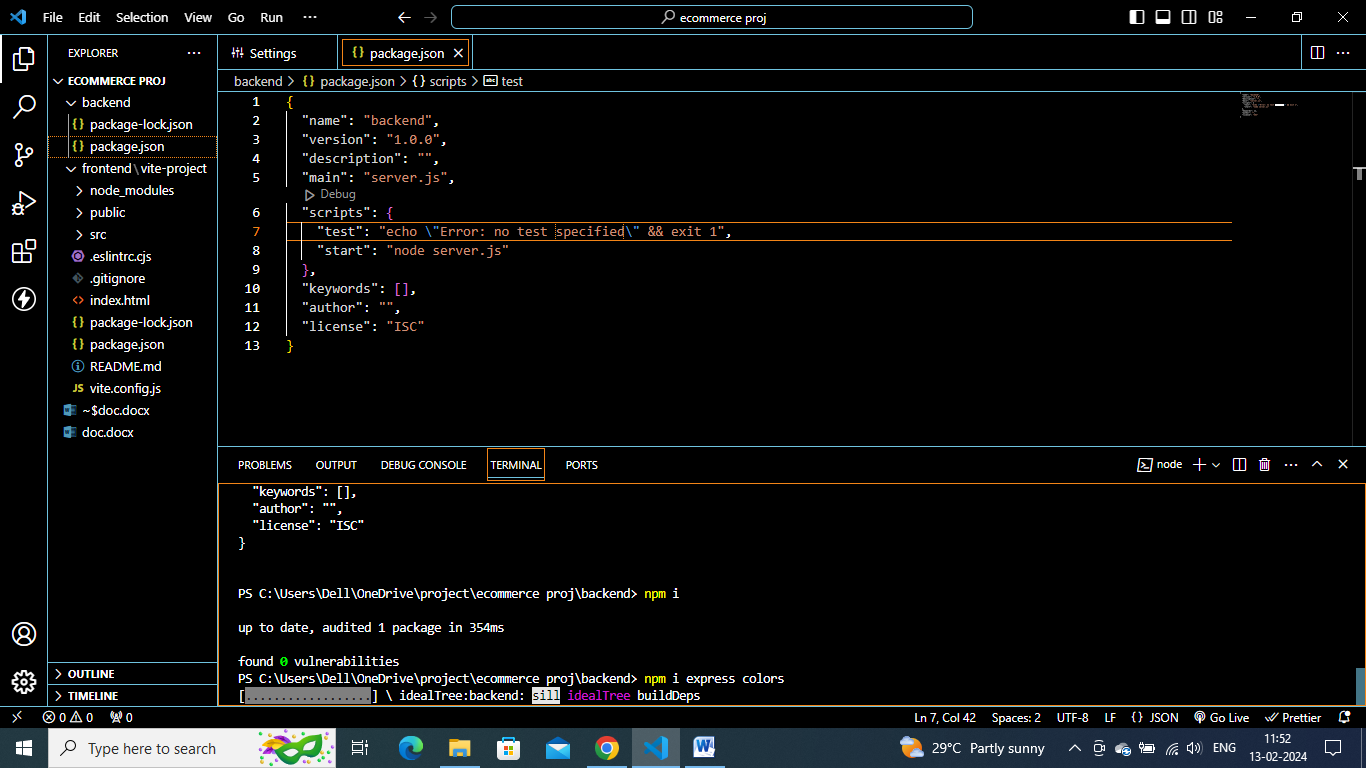


Can add start script like this.

By doing npm start this application will be running.

Can add author and keywords.

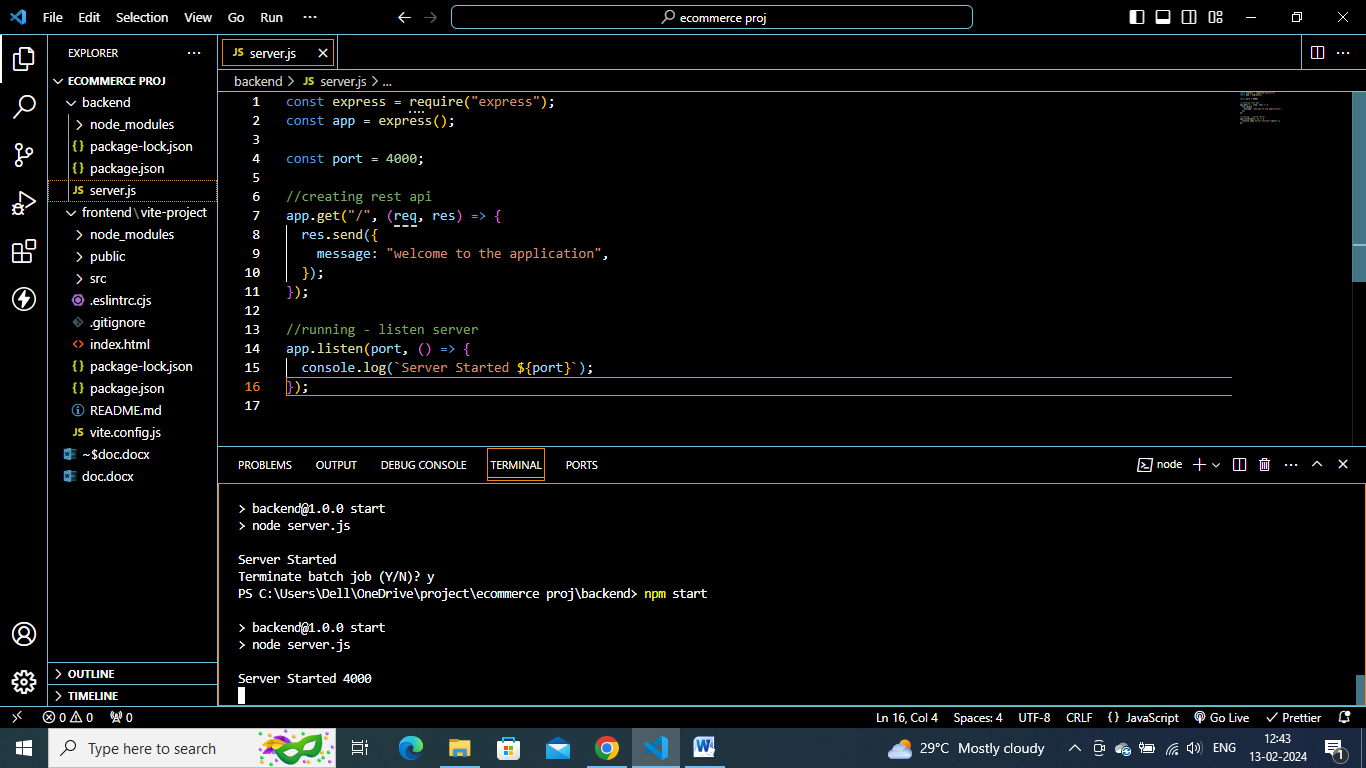
Npm I express colors’



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

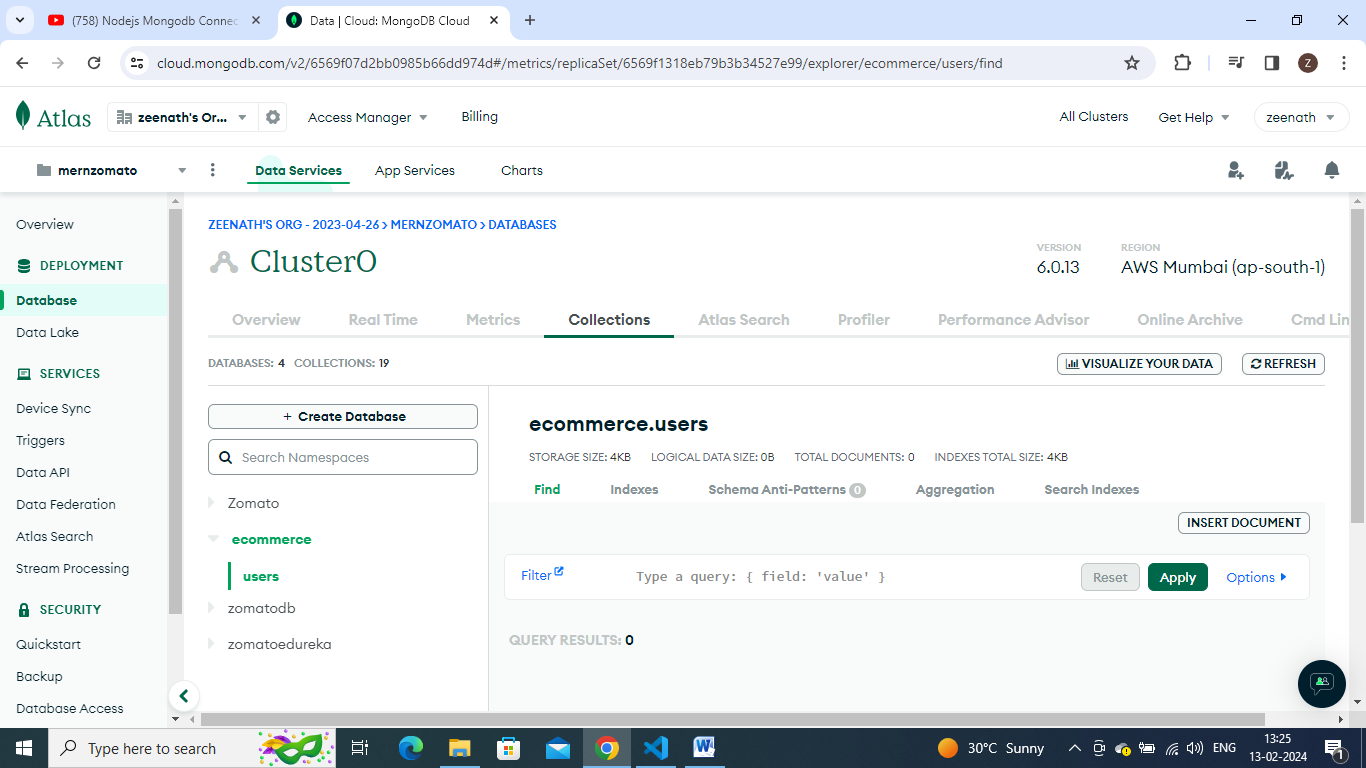
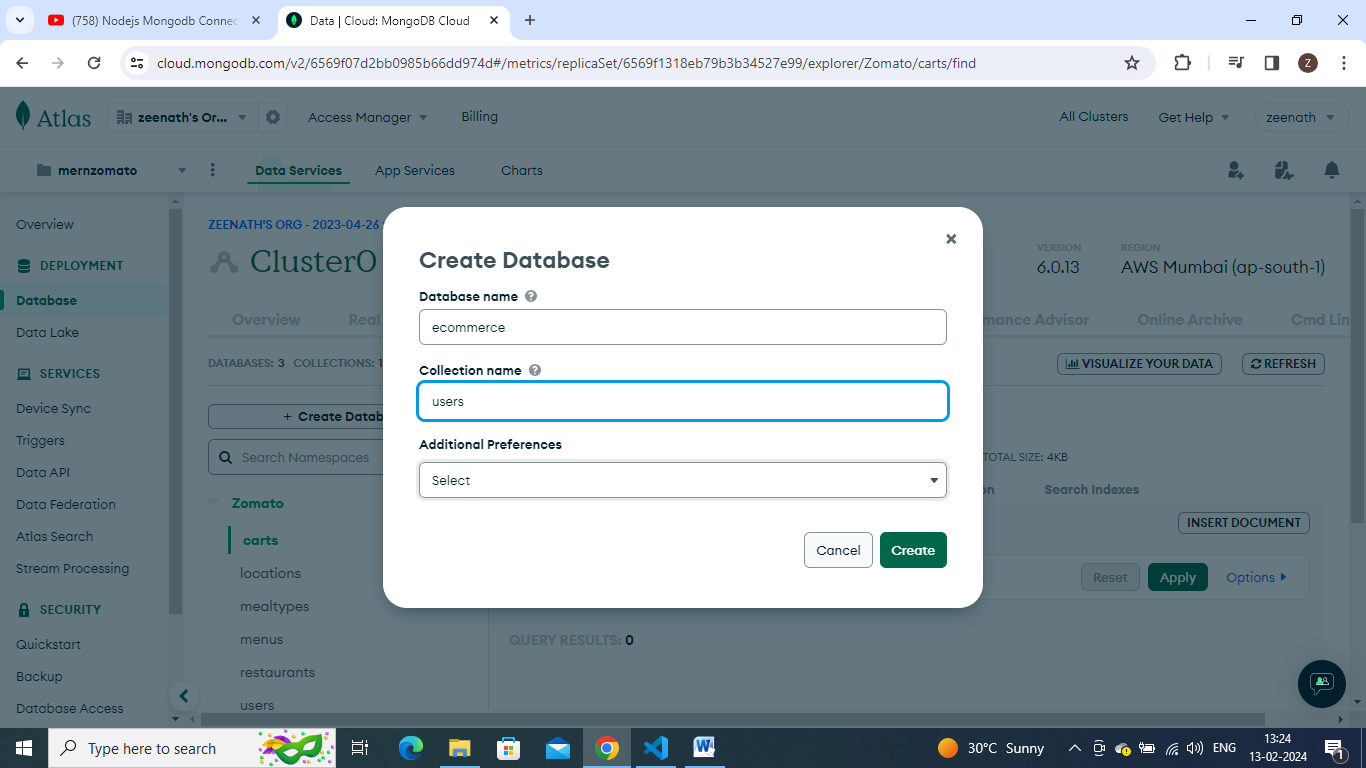
successfully created a server and somewt secured our project using .env file

**Connecting to mongodb:**

Install mongodb compass

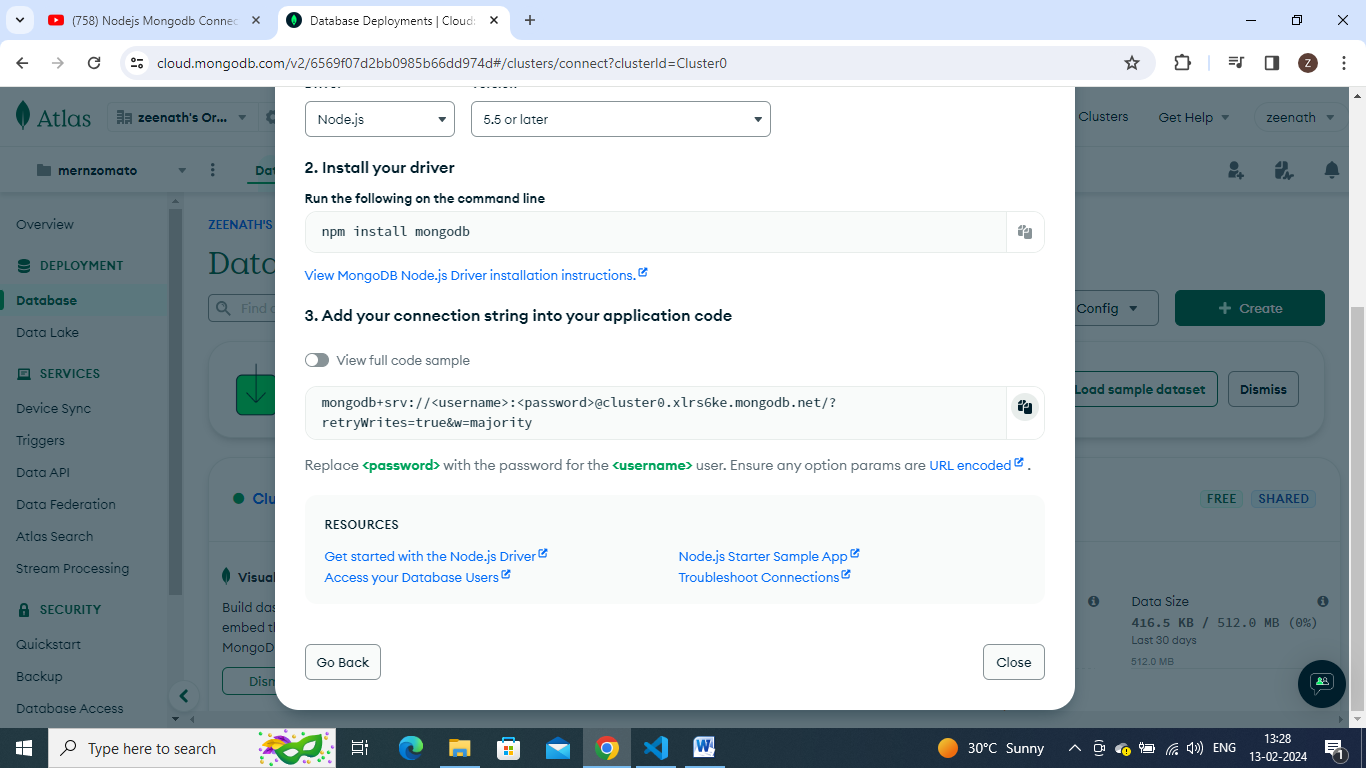
Create a free shared cluster in mongodb atlas.

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

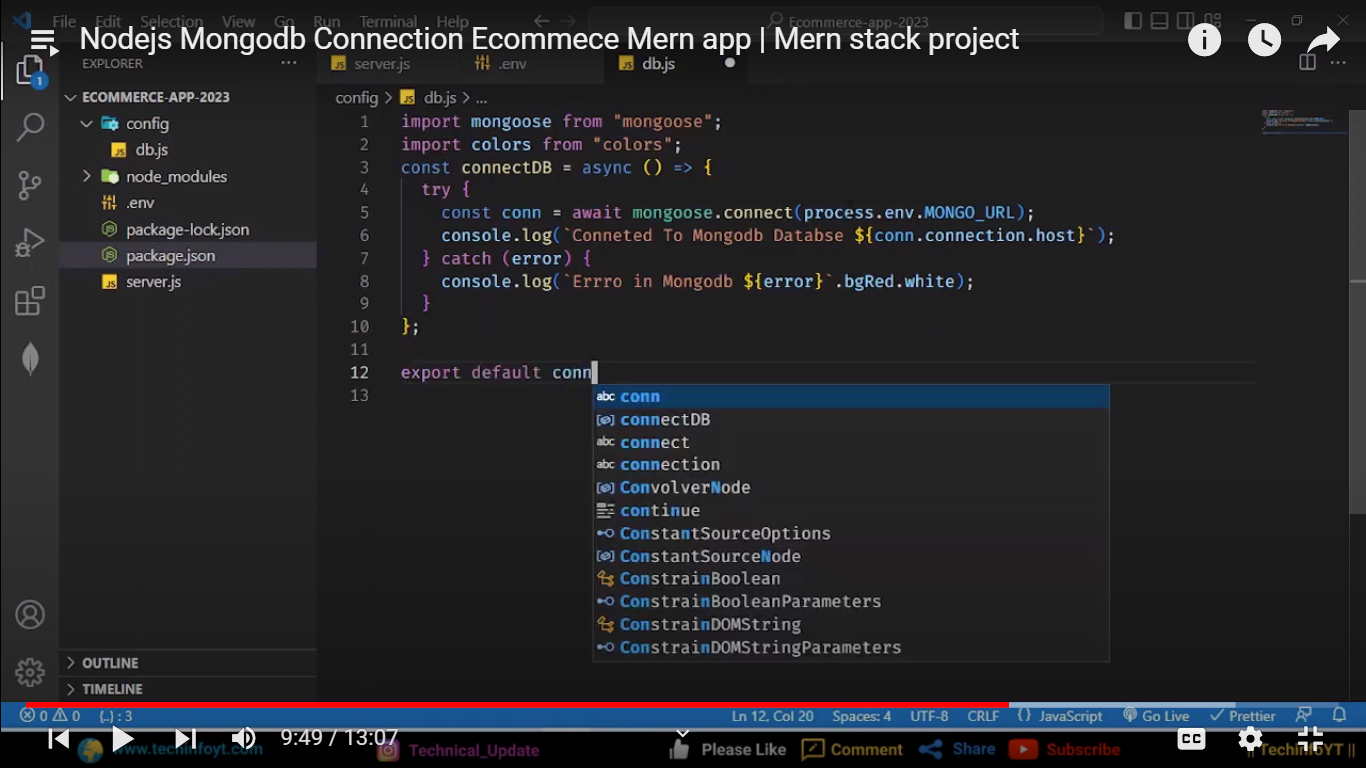
**Create config folder> db.js**

**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

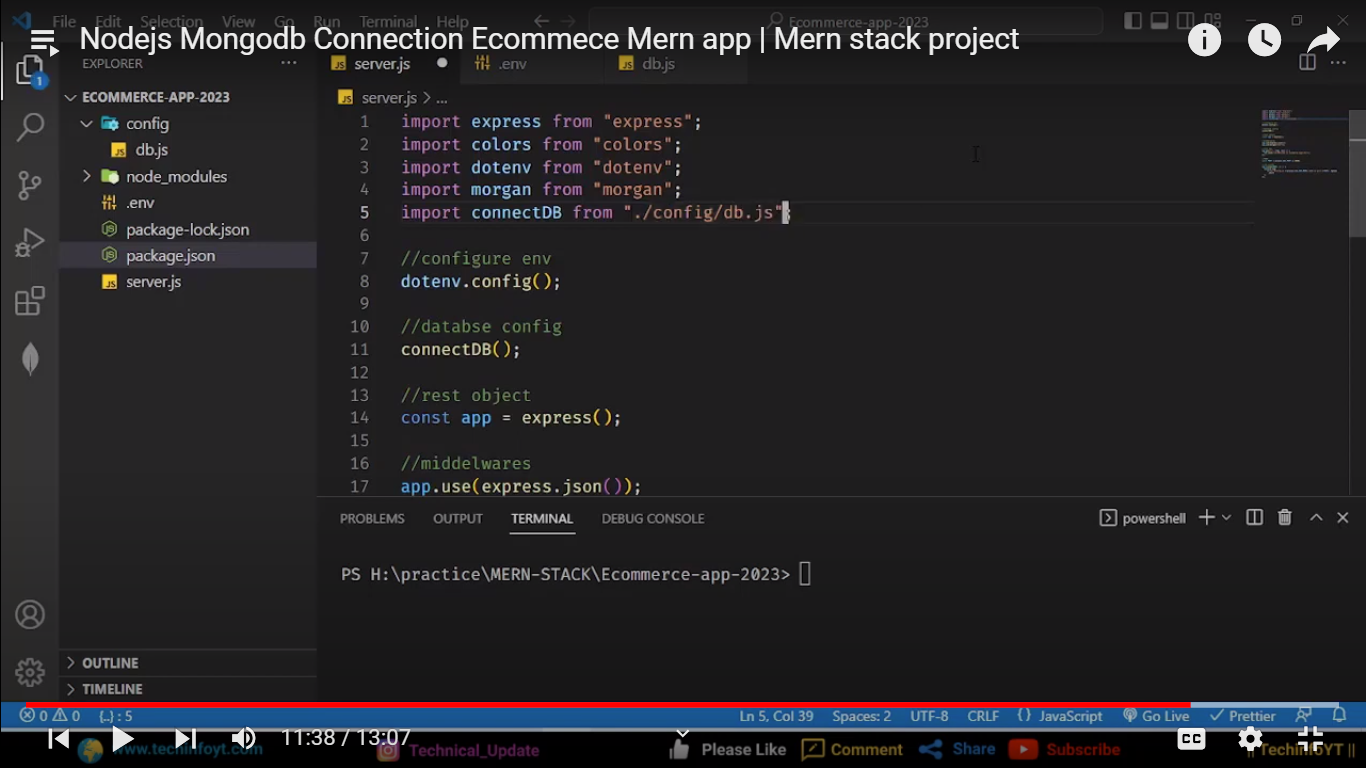
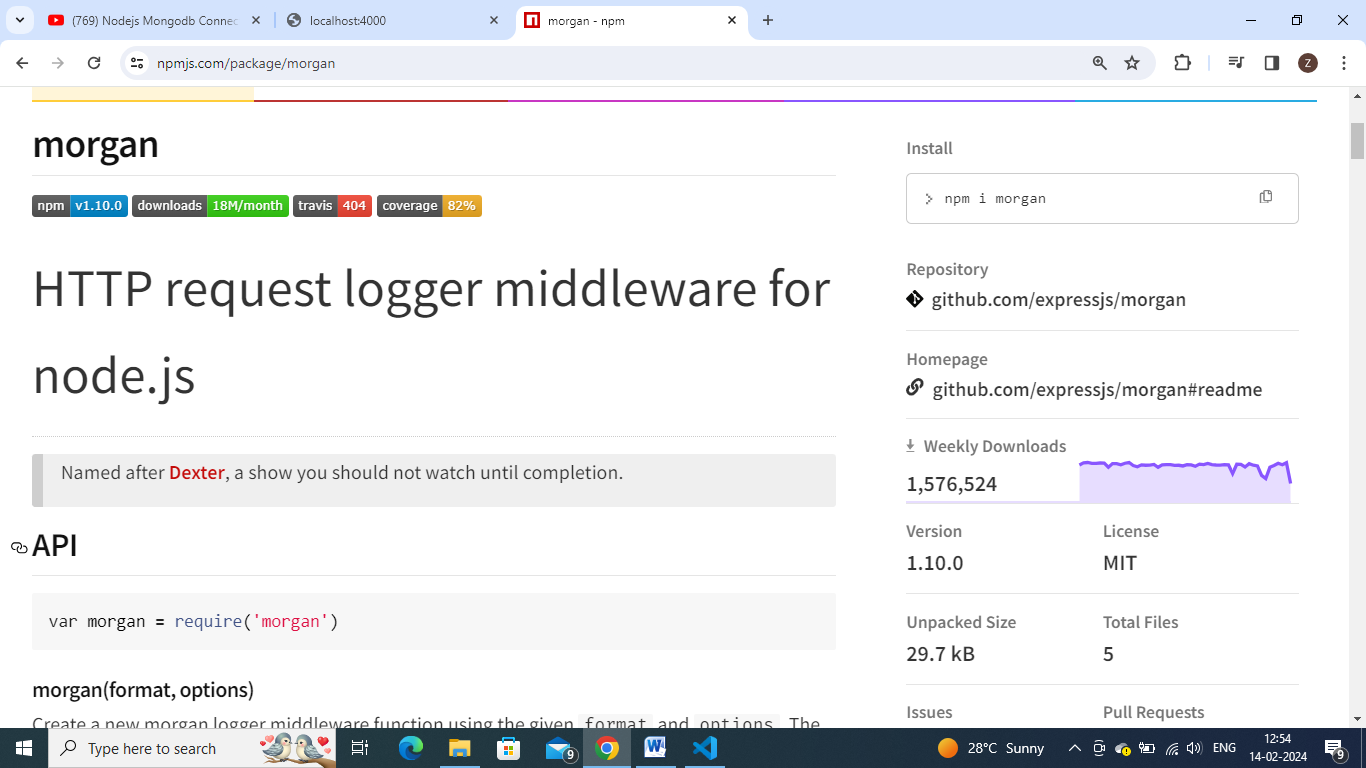
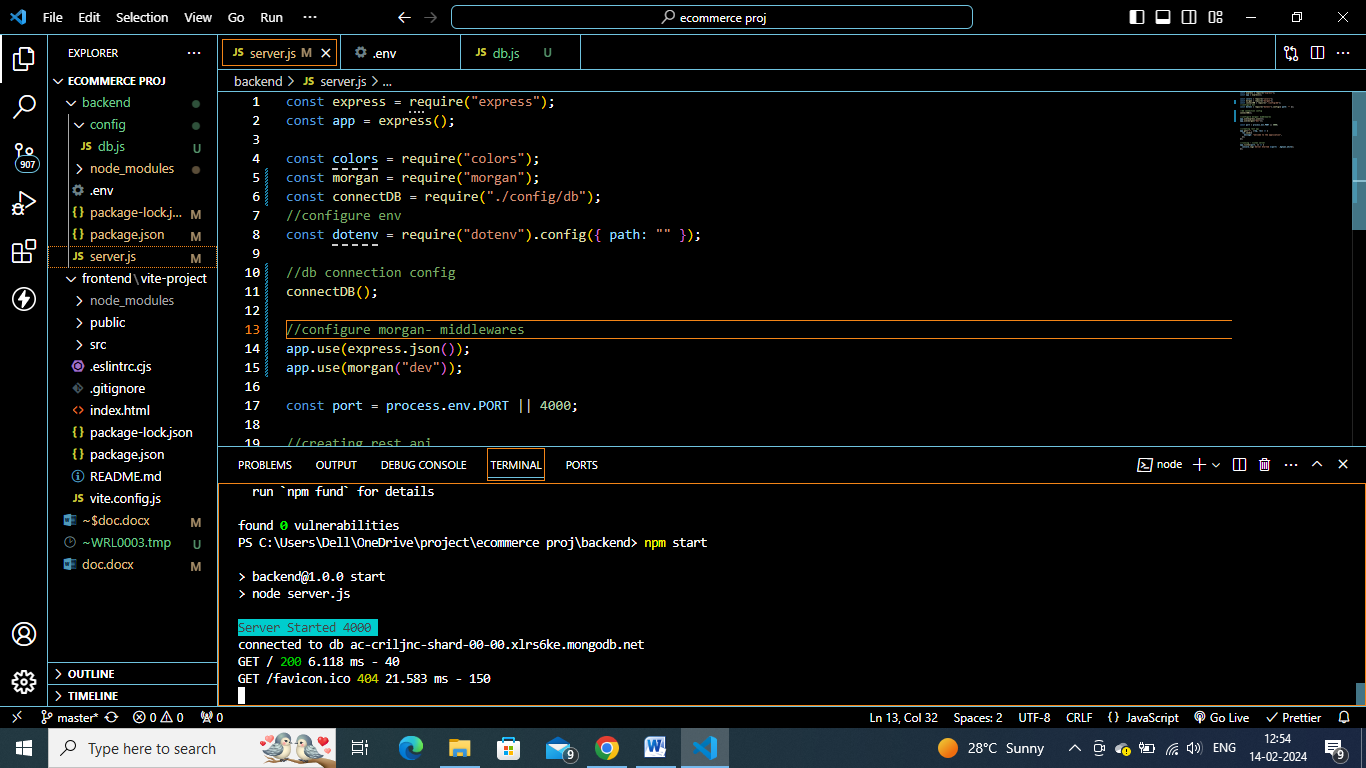
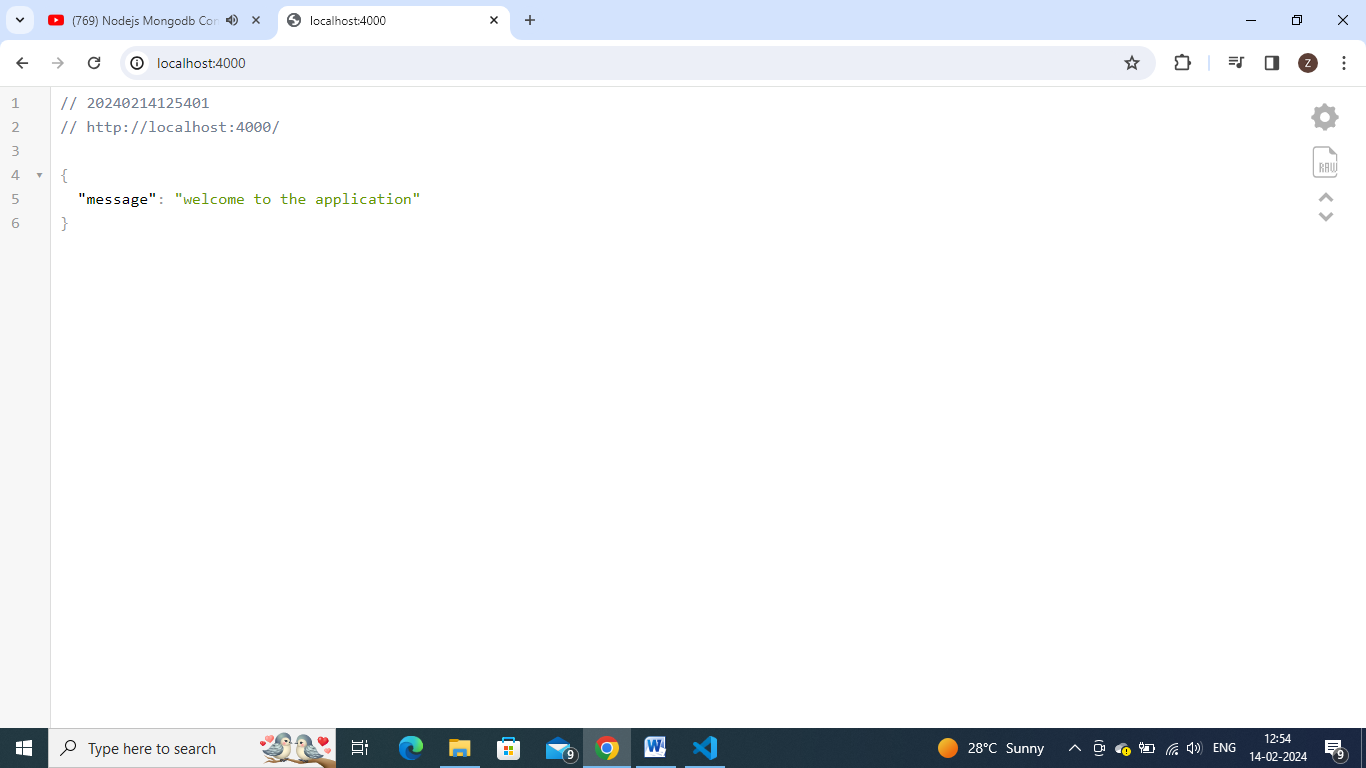
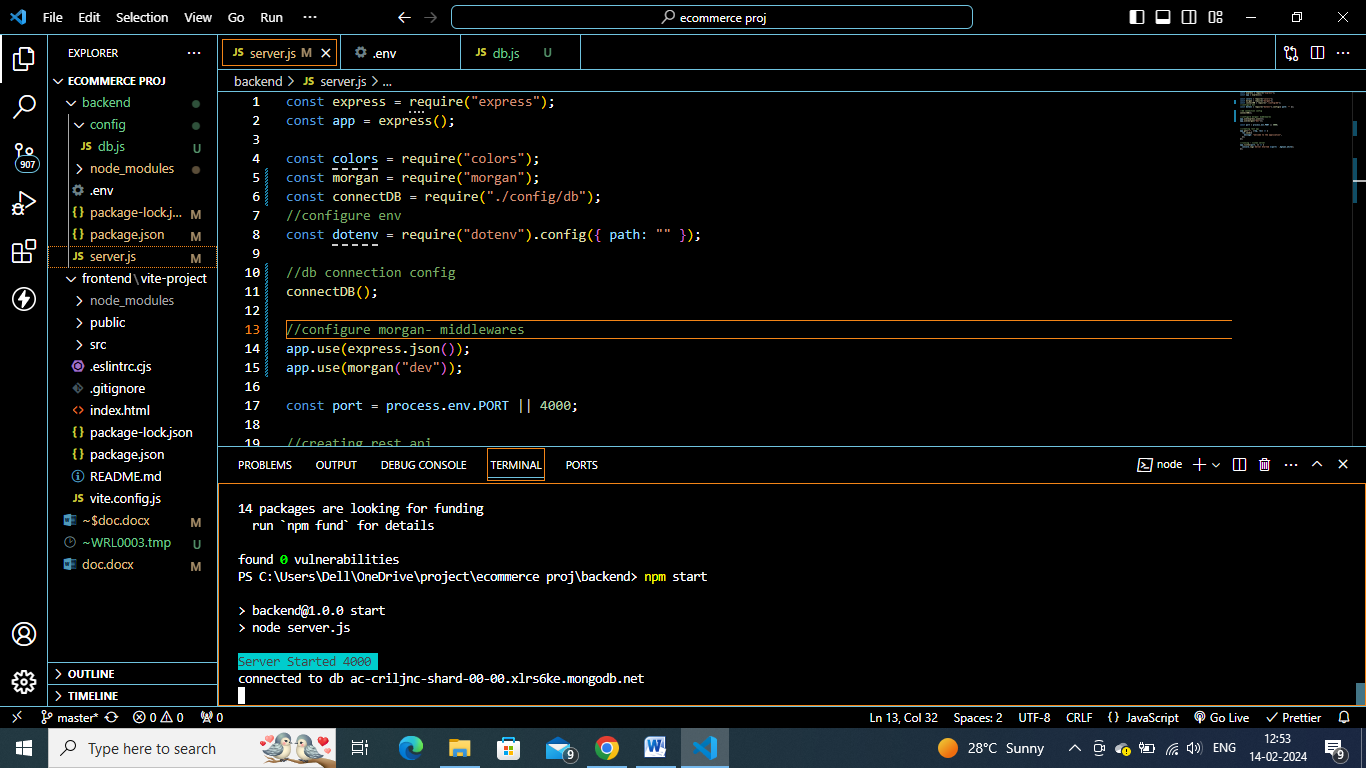
app.use(express.json())

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}`);

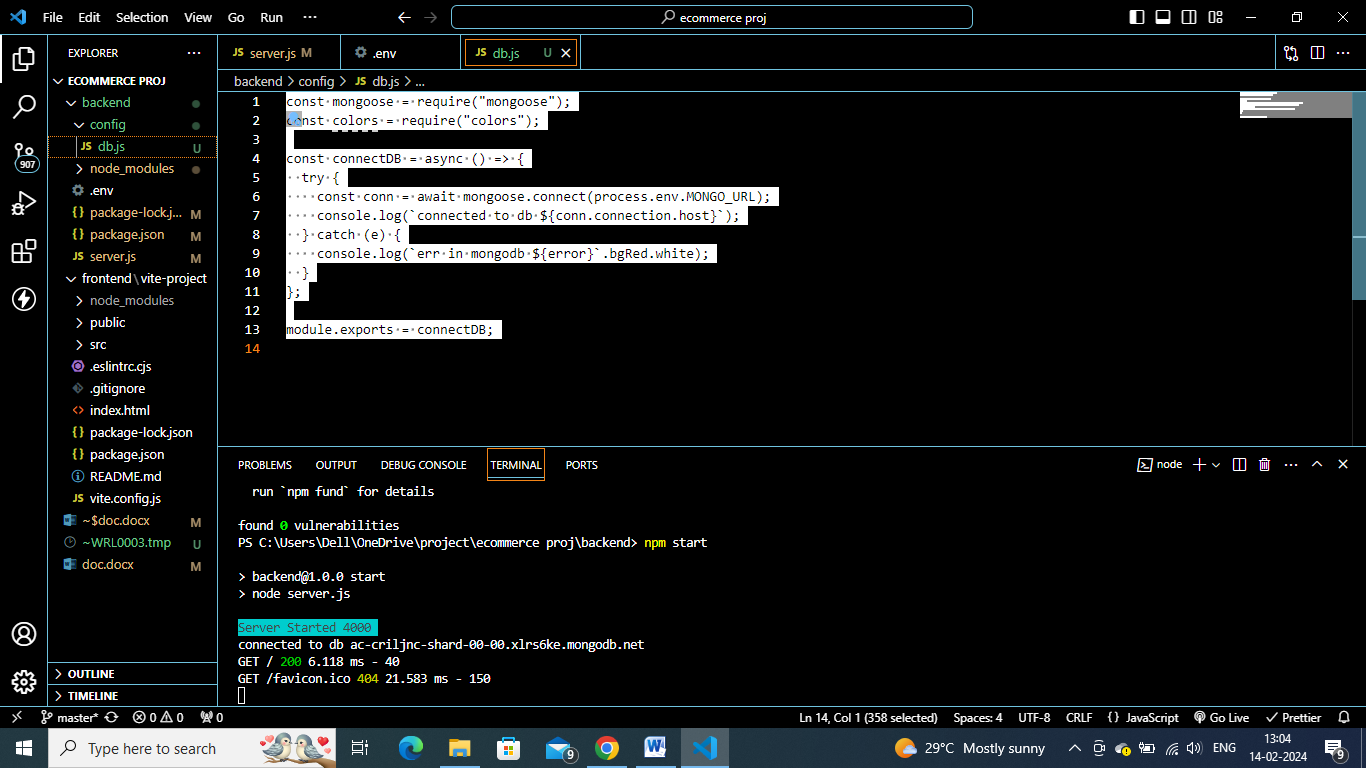
  } catch (e) {

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

  }

};

module.exports = connectDB;



Got get request hit endpoints bcoxz of morgan

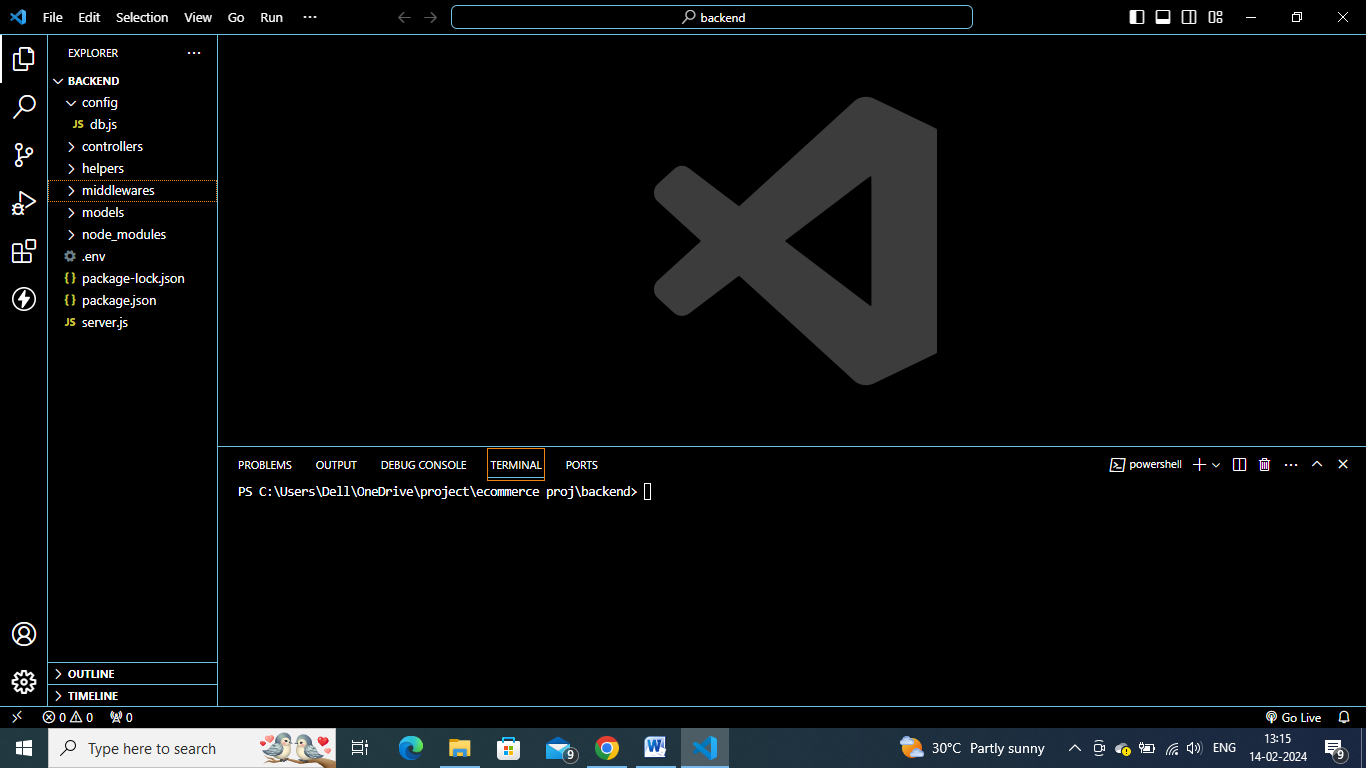
Showing requsta in console- req type,status code ,

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) => {

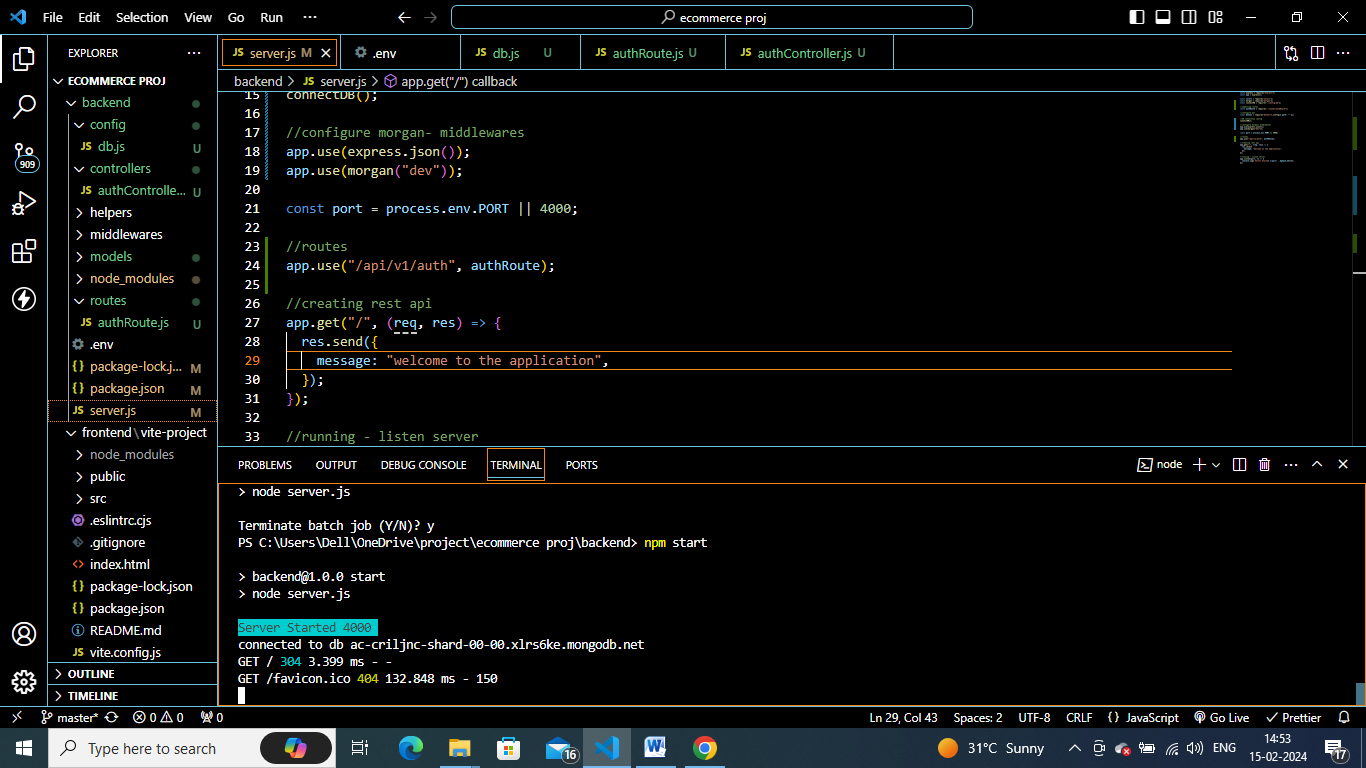
  res.send({

    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",

      error,

    });

  }

};

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