email sender

1. create a folder sever->npm init->npm i express dotenv cors

2. install nodemon : npm i -g nodemon

to run nodemon : nodemon app.js

3. in server folder create anoter folder routes in routes folder creates file -> router.js

4.creating our front end part

5. in frontend part :

on click of button we want to send our input data through api

the function :

*const* sendEmail = async (*e*) *=>* {

*e*.preventDefault();

*const* resp = await fetch("/register", {

      method: "POST",

      body: JSON.stringify({

        email,

      }),

      headers: {

        "Content-type": "application/json; charset=UTF-8",

      },

    });

    console.log(resp);

  };

6. creating a router : in folder : server/routs/router.js;

*const* express=require('express');

*const* router=new express.Router();

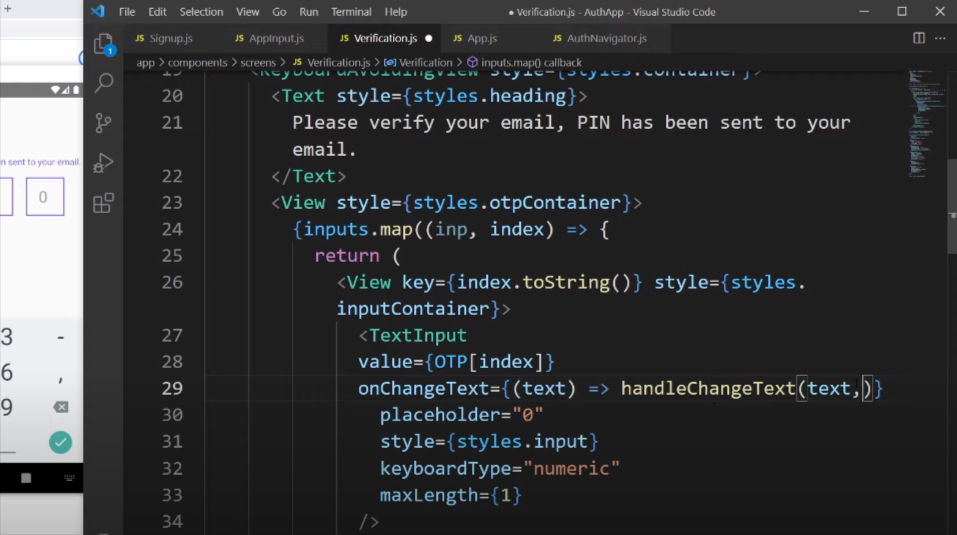
router.post('/register',(*reqs*,*resp*)*=>*{

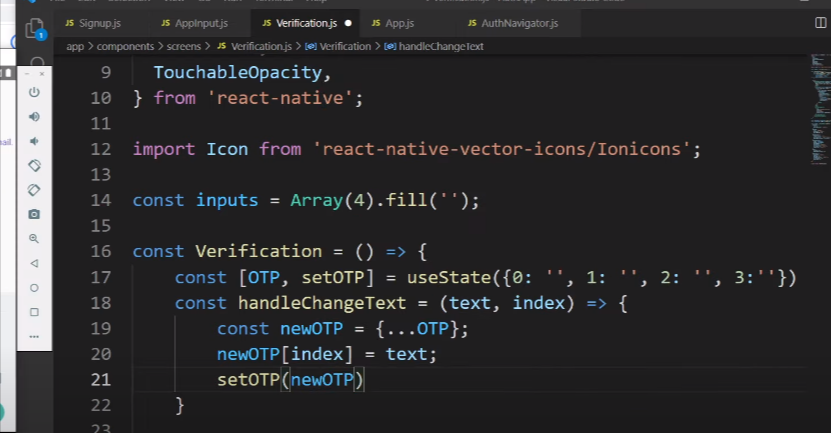
    console.log(*reqs*.body);

});

module.exports=router;

* NODE MAILER CODE SInnept
* *const* express = require("express");
* *const* router = new express.Router();
* *const* nodemailer = require("nodemailer");
* router.post("/register", (*reqs*, *resp*) *=>* {
* *let* { email } = *reqs*.body;
* try {
* *var* transporter = nodemailer.createTransport({
* service: "gmail",
* auth: {
* user: process.env.EMAIL, //email ID
* pass: process.env.PASSWORD, //Password
* },
* });
* *const* mailOption = {
* from: process.env.EMAIL,
* to: email,
* subject: "Sending email with react and nodejs",
* html: "<h1>Congrulation you have successfully send email<h1/>",
* };
* transporter.sendMail(mailOption, (*error*, *data*) *=>* {
* if (*error*) {
* console.log(`${*error*}`);
* } else {
* console.log(`Mail sent  ${*data*.response}`);
* *resp*.status(201).json({ status: 201, data });
* }
* });
* } catch (error) {
* *resp*.status(201).json({ status: 404, error });
* }
* });
* module.exports = router;
* seting multiple inpute data of form filed :
* *const* [inputData, setInputData] = useState({
* fname: "",
* email: "",
* password: "",
* });
* *const* handelInput = (*e*) *=>* {
* *const* { name, value } = *e*.target;
* setInputData({ ...inputData, [name]: value });
* };





otp //

// hashing password :

1 . install : npm i bcryptjs

2. inside import bcryptjs inside the module folder file

3 .

userSchemas.pre("save",async *function*(next){

    if(this.isModified("password")){

        this.password=await bcryptjs.hash(this.password,12);

    }

    next();

})

// genrating jwt token

// JWT TOKEN

userSchemas.methods.getJWTToken = *function* () {

  return jwt.sign({ id: this.\_id }, process.env.JWT\_SECRET, {

    expiresIn: process.env.JWT\_EXPIRE,

  });

};

// compareing a password

userSchemas.methods.comparePassword = async *function* (*enterPassword*) {

  return await bcyptjs.compare(*enterPassword*, this.password);

};

// creating cookies and jwt token in utiles folder

*const* sendToken = (*user*, *statusCode*, *res*) *=>* {

*const* token = *user*.getJWTToken();

  // options for cokkie

*const* options = {

    expires: new *Date*(

*Date*.now() + process.env.COOKIE\_EXPIRE \* 24 \* 60 \* 60 \* 1000 // given time in ms

    ),

    httpOnly: true,

  };

*res*.staus(*statusCode*).cookie("token", token, options).json({

    sucess: true,

    token,

    user,

  });

};

module.exports=sendToken

BACKENDED ERROR HANDELING :

Error hadling 1:not found

step 1 : make a folder with the name of utiles ,in utiles make a file name : errorhandler.js inside errorhandler.js

*class* ErrorHandler extends *Error*{

*constructor*(*message*,*statusCode*){

        supper(*message*);

        this.statusCode=*statusCode*;

*Error*.captureStackTrace(this,constructor);

    }

}

module.exports=ErrorHandler;

step 2 : make a folder middleware , in middleware make a file name Error.js , and in Error.js exports Errorhandler.js

--> inside error.js

*const* ErrorHandler = require("../utils/ErrorHandler");

*module*.*exports* = (*err*, *req*, *resp*, *next*) *=>* {

*err*.statusCode = *err*.statusCode || 500;

*err*.message = *err*.message || "internal Server Error";

*resp*.staus(*err*.statusCode).json({

    sucess: false,

    error: *err*,

  });

};

step 3 : exporting Error.js in app.js and use it like :

app.use(errorMiddleware);

step 4 : in porduct controllers.js

if (!product) {

    return next(new ErrorHandler("product not found", 404));

  }

2.Errorhadeling 2 : async and await error handeling

note : for smaller project we can use try catch block

for bigger project :

step 1 : make a file in middleware folder with name : catchAsyncError.js

*module*.*exports* = (*thefunc*) *=>* (*reqs*, *resp*, *next*) *=>* {

*Promise*.resolve(thefunc(*reqs*, *resp*, *next*)).catch(*next*);

};

step 2 : import it in productcontroller.js

*const* catchAsyncError=require('../middleware/catchAsyncError');

step 3 :prevsious

*exports*.createProduct = async (*req*, *resp*, *next*) *=>* {

*const* productData = await products.create(req.body);

  resp.status(202).json({

    sucess: true,

    productData,

  });

};

useing catchAsyncError

*exports*.createProduct = catchAsyncError(async (*req*, *resp*, *next*) *=>* {

*const* productData = await products.create(*req*.body);

*resp*.status(202).json({

    sucess: true,

    productData,

  });

});

Error 3 : unhandeled promise rejection

occur due to :due to promise rejection like :

when we connecting data base then at that time we use catch to remove catch from there we can do this by unhandeled promise rejection

in bootom of server .js file

process.on("unhandledRejection",*err=>*{

  console.log(`Error: ${*err*}`);

  console.log("Shutting down the server due to unhadled promise Rejection");

  server.close(()*=>*{

    process.exit(1);

  })

});

Error 4 : uncaughtException

we need to write code on the top of server.js

// handeling uncaught exception

process.on("uncaughtException",*err=>*{

  console.log(`Error: ${*err*.message}`)

  console.log("Shutting down due to uncaugthException");

  process.exit(1);

})

Error 5 : mongodb error :

like :CastError(wrong mongodb id error)

inside : middleware folder -> in error.js

*const* ErrorHandler = require("../utils/ErrorHandler");

*module*.*exports* = (*err*, *req*, *resp*, *next*) *=>* {

*err*.statusCode = *err*.statusCode || 500;

*err*.message = *err*.message || "internal Server Error";

  // wrong mongodb id error

  if (*err*.name === "CastError") {

*const* message = `Ressource not found , Invalid : ${*err*.path}`;

*err* = new ErrorHandler(message, 400);

  }

*resp*.status(*err*.statusCode).json({

    sucess: false,

    error: *err*.stack, // we can use err.stack to know the complete error

  });

==> search , filter , pagination Api forming

*const* { json } = require("express");

*const* products = require("../models/productmodels");

*class* ApiFeatures {

*constructor*(*query*, *querystr*) {

    // http://localhost:80006?keyword=somasa -- here keyword is querystr  and products.find() is query

    this.query = *query*;

    this.querystr = *querystr*;

  }

  //search features

  search() {

*const* keyword = this.querystr.keyword

      ? {

          name: {

            $regex: this.querystr.keyword,

            $options: "i", // means case insensentive

          },

        }

      : {};

    // console.log(keyword);

    this.query = this.query.find({ ...keyword });

    return this;

  }

  filter() {

*const* queryCopy = { ...this.querystr }; // we have done destructuring here because it give me refernce not the actural value

    // Removing some field for catergory

    // console.log(queryCopy);

*const* removeFields = ["keyword", "page", "limit"];

    removeFields.forEach((*key*) *=>* delete queryCopy[key]);

    // filter for price

*let* queryCpystry = JSON.stringify(queryCopy);

    queryCpystry = queryCpystry.replace(

      /\b(gt|gte|lt|lte)\b/g,

      (*key*) *=>* `$${*key*}`

    ); // .replace(/\b()\b/g,kery=>`$${key}`)

    this.query = this.query.find(JSON.parse(queryCpystry));

    return this;

  }

  pagination(*resultPerPage*) {

*const* currentPage = *Number*(this.querystr.page) || 1;

*const* skip = *resultPerPage* \* (currentPage - 1);

    this.query = this.query.find().limit(*resultPerPage*).skip(skip);

    return this;

  }

}

module.exports = ApiFeatures;

export : this in product controllers and used it in

==> User Authentication and password

--> Here we will going to create

i)Register user

ii)login User

iii)Logout User

iv) reset password

npm i bcryptjs jsonwebtoken validator nodemailer cookie-parser body-parser

step 1->create regiter and login page

step 2 -create a cookie in utiles folder with the name of (jwttoken)

step 3 - > create a authentication for user to see your proudct

-> creating a middler ware in middleware folder with the name auth.js

==> forget password

step 1 : create a method in usermodule file with the name getResetPassword

userSchemas.methods.getResetPasswordToken = *function* () {

  // genrating token

*const* resetToken = crypto.randomBytes(20).toString("hex");

  // hashing and adding resetPasswordToken to userSchema

  this.resetPasswordToken = crypto

    .createHash("sha256")

    .update(resetToken)

    .digest("hex");

  this.resetPasswordExpire = *Date*.now() + 15 \* 60 \* 1000;

  return resetToken;

};

step 2 : in userControllers make a function reset user password

// reset password

*exports*.resetPassword = catchAsyncError(async (*reqs*, *resp*, *next*) *=>* {

*const* { email } = *reqs*.body;

*const* preuser = await User.findOne({ email: email });

  if (!preuser) {

    return next(

      new ErrorHandler("User is not registerd , please register", 401)

    );

  }

*const* resetToken = preuser.getResetPasswordToken();

  await preuser.save({ validateBeforeSave: false });

*const* resetPasswordUrl = `${*reqs*.protocol}://${*reqs*.get(

    "host"

  )}/api/v1/password/reset/${resetToken}`;

*const* message = `Your password reset token is :- \n\n ${resetPasswordUrl}\n\nIf you have not requested this email please ignore it`;

  try {

    await sendEmail({

      email: preuser.email,

      subject: "Ecommerce password Recovery",

      message,

    });

*resp*.status(200).json({

      sucess: true,

      message: `Email sent to ${email} sucessfully`,

    });

  } catch (error) {

    (preuser.restPasswordToken = undefined),

      (preuser.resetPasswordExpire = undefined);

    await preuser.save({ validateBeforeSave: false });

    return next(new ErrorHandler(error.message, 404));

  }

});

step 3 : in utiles file create a file sendEmail.

*const* nodemailer = require("nodemailer");

*exports*.sendEmail = async (*options*) *=>* {

*var* transporter = nodemailer.createTransport({

    service: "gmail",

    auth: {

      user: process.env.EMAIL, //email ID

      pass: process.env.PASSWORD, //Password

    },

  });

*const* mailOption = {

    from: process.env.EMAIL,

    to: *options*.email,

    subject: *options*.subject,

    html: `${*options*.message}`,

  };

  await transporter.sendMail(mailOption);

};

==> Restpassword

step 1 : create a exports.restpassword in usercontrollers

*exports*.resetPassword = catchAsyncError(async (*reqs*, *resp*, *next*) *=>* {

  // creating token hash

*const* resetPasswordToken = crypto

    .createHash("sha256")

    .update(*reqs*.params.token)

    .digest("hex");

*const* preuser = await User.findOne({

    resetPasswordToken,

    resetPasswordExpire: { $gt: *Date*.now() },

  });

  if (!preuser) {

    return next(

      new ErrorHandler(

        "Reset password token is invalid or has been expired",

        400

      )

    );

  }

  if (*reqs*.body.password != *reqs*.body.confirmPassword) {

    return next(new ErrorHandler("password does not match", 400));

  }

  preuser.password = *reqs*.body.password;

  preuser.resetPasswordToken = undefined;

  preuser.resetPasswordExpire = undefined;

  await preuser.save();

  sendToken(preuser, 202, *resp*);

});

step 2 : create a route for this accourding to the message we have send

==> Error handeling in for duplicate user

==>

==>To upload Photo

step 1-> npm i cloudinary and npm i express-fileupload

step 2 > import fileUpload from ‘express-fileUpload int the app.js and

app.use(fileUpload({

    limits:{fileSize:50\*1024\*1024},

    useTempFiles:true

}));

step 2> import cloudinary in server.js and

cloudinary.config({

  cloud\_name: process.env.CLOUD\_NAME,

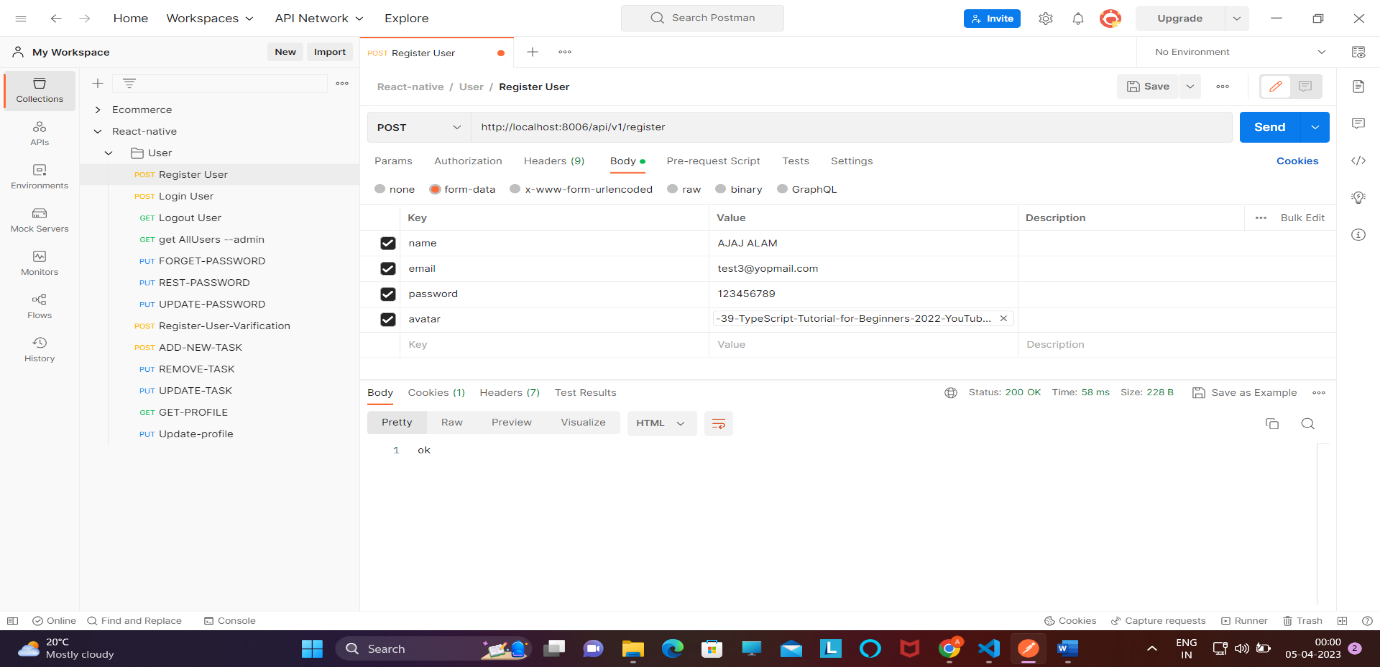
  api\_key: process.env.API\_KEY,

  api\_secret: process.env.CLOUD\_API\_SECRET,

});

step 3 : register on cloudinary and go to dasboard and copy all required fild in config.env file

step 4 –



step 5:

import cloud cloudinary where we want to upload

*const* avatar=*reqs*.files.avatar.tempFilePath; // to know the temp filepath

*const* mycloud=await cloudinary.v2.uploader.upload(avatar,{

    folder:"todo-app"

  })

fs.rmSync("tmp",{recursive:true}) // deleting the temp file by importing fs

avatar: {

      public\_id:mycloud.public\_id,

      url: mycloud.secure\_url,

    },