Node.JS

**To check node version on cli use**

Node –v

**Nope Package Manager (NPM)**

Is used to download and install modules which can then be used in your project.

**To check the version of node package manager (Npm) use:**

Npm –v

**To execute a node code on CLI use:**

Node file\_name.js

**TO CREATE A NODE PROJECT**

**Open CLI and create a folder in the directory where you want your project to be. Then navigate to that directory and…**

**Then call:**

**npm init**

**The above command will ask you of certain questions allowing you to make a choice of how you want your package.json to be. It will acutally ask of the project description, version, and project author**

===================================================================================================================

**NodeMon**

**By default, node.js cli doesn’t watch for changes like angular. So you need restart the server whenever you make any changes.**

**But you can use nodemon to watch for changes and keep the server on without restarting it whenever you make some changes. And is pretty simple.**

**To install nodemon on cli use**

npm install -g nodemon

**Once nodemon is installed, on CLI use:**

Nodemon projectname.js

**And stop using node projectname.js**

===================================================================================================================

**MODULES**

A module is an encapsulation of code in a single unit.

In a real world, it just a basket that contains many oranges. All the oranges (codes) are encapsulated into that single basket (module)

**USING A MODULE**

All modules need to be installed with NPM.

Because all modules are located or stored over the internet. So you need to install it as part of your project source file before you can include it.

Example: npm install express –save

After installing it, it then needs to be included in the app with the “require(‘module\_Name’)” method.

Example:

Var express = require(“express);

**CREATING A CUTOME MODULE AND USING IT IN OTHER FILES**

You can also create your own modules and include it in your project source files without having to rely on third party modules only.

There are two ways you can do this:

1. Create a module that export individual function or variables.

(Each function or variable must be exported and imported separately)

CREATE A FILE addition.js with below module syntess

Remember, all functions should be anonymous functions that should be assign to a variable

/\* Create a variable \*/

Var myName = "justice";

/\* Create an anonymos variable and assign it to a variable  \*/

var addition= function(a,b){

    return a+b;

}

// export the above two variables as modules that can be used in other class.

module.exports={addition, myName};

**Then include the above module “addition.js” in a different file index.js**

/\* import the above two exported “addition, myname” modules from addition.js module by using relative path ./  \*/

var {addition, myname} = require("./addition.js");

/\*Call the addition function in addition.js  \*/

console.log(addition(2, 5));

/\* On your CLI navigate to the project folder and call "node index.js" The OUTPUT WILL BE 7  \*/

1. Create a module that you will export the whole file.

(the whole file must be exported as a single module. And the whole file must be imported as a single module.

Then functions or variables must be accessed with file\_Name.function\_Name in the imported file)

Create a file operation.js

// export all the function or variables in the file as a single module

module.exports = {

    addition: function(a, b){

        return a + b;

    },

*subtraction: function(a, b){*

*return a - b;*

*}*

*}*

Then create another file index.js

// import the above file that was exported as a single unit

var operationFile = "./operation.js";

// CALL ADDITION FUNCTION IN OPERATION.JS

operationFile.addition(2,4);

// CALL SUBTRACTION FUNCTION IN OPERATION.JS

operationFile.subtraction(20,7);

The concept behind the two steps is that:

If you export files as seperate modules, You must import them as separate modules with their exact names.

But if you export file as a single module, you must import the whole file and access the function or variables in it with the filename.functionName

===================================================================================================================

**CREATING A BASIC SERVER WITH PLAIN NODE**

There is a createServer() method of “http” module class that you must include and use it to create your server like below:

// import the http module which contains the method to create the server

var http = require("http");

/\* Create a server with the createServer(function(request, response)) methode of the above htt module class.

request => is used to send additional parameters to the server that the user/client who makes the request may not include.

Response => is used to send information back to the client \*/

var server = http.createServer(function(request, response){

    // what content-type type and server code, should your server send response in to your users?

    response.writeHead(200,{"Content-Type" : "text/plain"});

    // what do you want your users to see when they make a request to this server

    response.end("hellooo word my friend");

});

// What port do you want this server to listen on? And what callback message do you want to show in the console when you start your server?

server.listen(7000,function(){

    console.log("server is listening on port 7000");

});

// execute the above file and hit http://localhost:7000/ on your browser

===================================================================================================================

**THE URL MODULE**

The “url” module split a web address into different parts.

// Include the url module

var url = require('url');

===================================================================================================================

**Event and EventEmitter**

Event are things that occur when some action is performed on the site. Example. If user hovers on a button, if window just start or if a file is open, An event can be fired.

**All that you need to set up an event is to follow 3 approach.**

1) Create an event with the on(“event\_Name”, “callback\_func”) method

2) Create the function that will be called when the event is fired (callback function)

3) Fire the event with the emit(“name\_of\_event\_to\_fire”) method

/\* Include the events module in your project\*/

var events = require("events");

/\*  All event properties and methods are an instance of an EventEmitter object.

To be able to access these properties and methods, create an EventEmitter object: \*/

var eventEmitter = new events.EventEmitter();

/\*Crate a function that will do something or listen when the event is fired \*/

var greetings = function(){

    console.log("good morning my fans");

}

/\* Create an event with the name "scream" and call the callback function to be executed when this scream event is fired \*/

eventEmitter.on("scream",greetings);

/\* Fire the above "scream" event with the emit("event\_Name") method \*/

eventEmitter.emit("scream");

**TO PASS PARAMETER TO AN EVENT CALLBACK FUNCTION**

Normally the callback function of an event may accept a parameter. In that case, you only have to pass in the value as a second parameter to the emit(“eventName”, callback\_functn\_parameter\_value) method when firing the event.

Using the above included event as a reference:

/\*Crate a callback function with a parameter that will do something or listen when the event is fired \*/

var greetings = function(name){

    console.log("good morning " + name);

}

/\* Create an event with the name "scream" and call the callback function to be executed when this scream event is fired \*/

eventEmitter.on("scream",greetings);

/\* Fire the above "scream" event above and pass value to the callback function \*/

eventEmitter.emit("scream", ”justice”);

===================================================================================================================

**EXPRESS**

Express is a node.js server side framework for building server side applications. It makes it easy to handle multiple request like get, post, put and delete.

**To install express on CLI use:**

The “--save” below will tell npm to include express in your package.json. So always include –save whenever you installing any module so that you can identify which modules you used in your project even if you don’t commit all modules source files to your github repository

Npm install express --save

**To create a basic server with express, follow:**

/\* include the "express" module class in your project \*/

var express = require("express");

/\*Create an object of the above included express module \*/

var expre = express();

/\* Use the get("url", callback\_function) method of the express module to create a server\*/

expre.get("/",function(req, res){

    /\*what response do you want to send to your users \*/

 res.send("helloo word");

});

/\* What port number do you want your server to run on and what callback message do you want to show on console to the developer?  \*/

var server = expre.listen(3000, function(){

    console.log("server is running on port 3000");

});

OR ABOVE PORT NUMBER CAN BE…..

/\* When you deploy your application to the hosting server, manually inputting a port number will cause an issue because the hosting server is the one responsible to tell you the port number it runs on.

In other not to run into issue when you deploy your app, you can read the hosting server port with process.env.PORT || 7000 (the || 700 says hey, if there is no port number on the environment my server is on, use 7000. And you will typically see 7000 when working on a local environment)

To set a local env.PORT number on cli, pause the server and use: set PORT=5000 (now when you restart the server it will run on port 5000 instead of 7000 and does because you said here process.env.PORT || 7000 that if there is an env.PORT number use it, but if there is no env.PORT number use 7000 ) \*/

const port = process.env.PORT || 7000;

application.listen(port,function(){

    console.log("server is listening on port "+ port);

});

**To pass in required Parameters To Urls**

Required Parameters are values that must be passed in the URL.

To set a required Parameter use ":required\_Paremeter\_Name"

It must start with double colon : before the name eg. :name/:age

To see all required paremeters object use: req.params // “req” is the request object

To access a specific parameter use: req.params.requiredparameter\_Name

application.get("/api/:name/:age", function(req, res){

    //remember :name/:age above are the required parameters every

        //get the value of the name parameter

            res.send(req.params.name);

        // get all the object of required parameters

          res.send(req.params);

    });

/\*

if I pass http://localhost:5000/api/justice/21 to the url

res.send(req.params.name); above will be “justice”

res.send(req.params); will be. {"name":"justice","age":"21"}

\*/

**To pass in a query string or optional parameters**

**They are parameters users may choose to pass it or not.**

Query strings are searchable key values we append to url’s after the question mark symbol “?”.

For example <http://localhost:5000/api/justice/21?school=ogyeedom> here “school=ogyeedom” is the query string. The key is “school” and the value is “ogyeedom”

application.get("/api/:name/:age", function(req, res){

    res.send(req.query.school);

});

/\* if I pass http://localhost:5000/api/justice/21?school=ogyeedom to the url

    res.send(req.query.school); above will be “ogyeedom” \*/

To retrieve the full url of every request simply use req.url

Example http://localhost:5000/home

 req.url

//resurlt will be http://localhost:5000

**TO MAKE POST REQUEST AND RETRIVE FORM VALUES**

Let say I have these two form input field values with names “username” and “password” in my html template.

When a user click on the submit button, I will get both the headers request (content\_type) and the body request (form values) the user sends.

Example:

Header request => { Content-Type: “plain/text”, Content-Length: 29 }

Body request => { username:”justice”, password:”justilato”}

You can see all the above two request are in json. But express does not read json request.

So first, you need to tell express to read json request with express.use(json()) before you can access the above values with express like below.

From the above, you can access:

header request with express like: req.header(“Content-Type”)

body request with express like: req.body.username or req.body.passowrd

// tell express to read json form request

application.use(json());

//make a post request that the user will pass his name and password

express.post("/corses",function(req, res){

const userDetails = {

   //grab the user username and assign it

      username: req.body.username,

   //grab the user password and assign it

     password: req.body.password

}

//display a response status code and the user details he sent

  res.status(200).send(userDetails);

});

/\* you can now use post man to make a post request and pass in those two values to see it.

You also have to validate all user input. So check how to use Joi to validate request values to make sure users don’t break your db.

Use Joi instead of manual validation \*/

**THE DIFFERENCE BETWEEN THE next() method IN MIDDLEWARE and next() method in route callback FUNCTION**

All express.use() are middlewares that runs on every routes request. There are a lot of express.use() middleware methods you should check them out

var express = require('express');

var app = express();

/\* **Next() function in middLeware**

Will allow you to NAVIGATE to the url the user want to see \*/

/\* This is a middleware that will run on every request before you can decide to allow the user to navigate to the url he want to go or not\*/

app.use(function(req, res, next) {

  console.log("I am the middleware.");

/\*After printing the above message to the console, please go ahead and navigate to  whatever url the user has specied to go.

So if the user is navitage to http://localhost:8000 The above "I am the middleware"  will be executed on the console before

the below next() method will navigate to the url the user want to go (http://localhost:8000)

If you don't call below next() method. the above "I am the middleware." Will be log to the console and the program will end, it will not navigate the user to whatever url

he want to see

 \*/

next();

});

app.get('/', function(req, res, next) {

  res.send('Welcome to homepage');

});

/\* **Next function that is not in milldeware. That’s  (rout next() method)**

Will redirect to execute the callback function of the next route that has the same path name.\*/

app.get('/help', function(req, res, next) {

    res.write('am the first help page');

    /\*Hey, when you finnish priting the above "am the first help page" look at the next rout with

    the same path "/help" and execute it call back function

    Becuase there is another get "/help" path below when the user naviage to http://localhost:8000/help

    the result will be:

    am the first help page

    am the second help page\*/

    next();

  });

  app.get('/help', function(req, res, next) {

      res.write('am the second help page');

    });

    /\* FROM THE ABOVE YOU CAN SEE THAT YOU HAVE TWO "/help" path with different callback message

    Well, you can define multiple callback funct easily for the same "/help" path as below. is easy than above.

    \*/

   app.get('/help', function(req, res, next) {

      res.write('am the first help page');

  /\* execute the next callback function bellow because they all belongs to the same path “/help” \*/

      next();

   },

    function(req, res, next) {

      res.end('am the second help page');

    }

   );

  var server = app.listen(8000, function () {

    console.log("Example app listening on port 8000");

  })

**RES.SEND(), RES.WRITE() AND RES.END() DIFFERENCE**

**Res.send()**

Is use to send response to the user and close the connection automatically after the response is sent.

It’s equivalent to both res.write() + res.end()

So whenever you call res.send(), a respond will be sent to the user and get closed automatically. Therefore you cannot call res.send() twice.

app.get('/help', function(req, res, next) {

      res.send();

/\*error can't send request hearders to the user once it has been sent \*/

      res.send();

});

**Res.write()**

It use to send response to the user without closing it.

It equivalent to only res.write() So it can be called multiple times.

app.get('/help', function(req, res, next) {

      res.write();

      res.write();

res.write();

res.end();

});

**res.end()**

It sends response to the client and close it. Therefor it can be called once.

app.get('/help', function(req, res, next) {

res.end();

});

===================================================================================================================

**MONGO DB**

Is a collection database which does not use sql language, that is why it is a nosql database.

All database has collections not tables as in mysql.

All the dates in a collection are grouped in key value pars model.

Remember collections are nothing but objects.

Example:

Db = student

Collection = studentDetails (below are sample data in this collection)

{

studentName: “justice”,

studentAge: 24

},

{

studentName: “shadrack”,

studentAge: 20

}

**MONGODB INSTALLATION**

There are two things you need to do to get mongodb working on your system. 1. Download mongdb itself 2) download gui for mongodb which is compass

1. Go to mongodb website and download mongodb and install it on your system c drive. Once it is installed, navigate to the installation source folder on your c drive and open the “bin” folder. Then open “mongod.exe” inside the bin folder. mongod.exe is responsible for starting your mongodb. Then go to root directory of your c drive and create this directory data/db (It’s where db collections will be installed)
2. Search for mongodb compass on google and download and install it on your system. Once its done, run it and it will connect to your mongodb once “mongod.exe” is running. Then create a database and a collection and start using it. Compass allow you to work graphically with mongodb instead of creating all collections on the CLI

**NODE.JS, MONGODB CONNECTION**

To connect node to mongodb, you need a connector. The node connector to mongodb is called “mongoose”

You can download it on your cli with below command.

Npm install mongoose –save

Then create a new file for your db connection and do the following..

// include the mongoose connector in your project

var mongoose = require("mongoose");

/\*

We use mongoose.connect(db\_hot, callback\_func) method with requires two parameters

1. The database information in string. Bellow “mongodb://localhost:27017/edureka” is of three part 1) “mongodb://” the type of the database

2) “localhost:27017/” the host the database is running on. You can find this easily on your compass interface. 3) “edureka” the name of the database you want to connect to

1. A callback function with a parameter that contains object of error in case the db connection fails

\*/

var dbconnectionurl = "mongodb://localhost:27017/edureka";

// forget about the second object parameter below { useNewUrlParser: true } it does nothing but node requires that you put it there

mongoose.connect(dbconnectionurl, { useNewUrlParser: true }, function(err){

//check if connection is successful or not

    if(err){

        console.log("database connection fail");

    }

    else{

        console.log("connected");

    }

});

// import below courses model schema. (this will make this single file have both the connection and the schema of the db)

const course = require("./courses")

**CREATING A COLLECTION SCHEMA**

Collection schema represents the structure of keys or tapes of values you want to assign to keys in your collection.

Because without schema, if you have a “name” key that accepts only string of names. Someone can pass integer to it and it will get stored. But with schema, you can decide what data types you want such name key values to have.

Create a new file and do the following to create a collection schema

// include mongoose

const mongoose = require("mongoose");

/\*

Use new monggose.schema({ keyName:{type:”data type you want this key to accept”, required: “this key is always required”} }) to create a new schema.

\*/

var coursesSchema = new mongoose.Schema({

    courserName:{

        type: String,

        required: true

    },

    courseId:{

        type:String

    }

});

//export the above schema with a given name by doing: mongose.model(“name\_of\_collection”, name\_of\_schema\_you \_want\_to\_expert);

Const courses = module.exports = mongoose.model("course",coursesSchema);

===================================================================================================================

**MYSQL**

Node js can easily connect to mysql database and perform crude operation.

But you first need a mysql connected called “mysql”

// run below code to install the connected

npm install mysql

**CREATING A CONNECTION**

//INCLUDE THE MYSQL MODULE

var mysql = require('mysql');

// CREATE THE DATABASE CONNECTION DETAILS

var con = mysql.createConnection({

  host: "localhost",

  user: "yourusername",

  password: "yourpassword"

});

//CONNECT THE DATABASE AND CHECK FOR ERRORS

con.connect(function(err) {

  if (err) {

RETURN “CONNECTION EROOR”;

}

  console.log("Connected!");

});

**TO QUERY THE DATABASE**

The connection object created in the example above, has all the method for querying the database.

Use it with below syntess to perform any crude operation.

Var sql = “select \* from userstable”;

con.connect(function(err) {

    if (err) { return “eroor please”};

  else{

    console.log("Connected!");

    con.query(sql, function (err, result) {

      if (err) throw err;

      console.log("Result: " + result);

    });

  }

  });

===================================================================================================================

**DEFINNING A SEPRATE ROUTES FOLDER**

It helps to organize your project with different rout files instead of adding all in idex.js

So create a folder “routes/studentroute.js” and do the following…

// import the express module

var express = require("express");

// create a router object of express

var router = express.Router();

// use the router object to define a route of any method (get, post, delete and put)

router.get("/names", function(reg, res){

 res.send("this route can be accesed with localhost:200/api/names");

});

// use the router object to define a route

router.post("/names/:firstname", function(reg, res){

 res.send("this route can be accesed with localhost:200/api/names/post\_studentName by using postman");

});

//export the above router so that it can be imported in index.js

module.exports = router;

Then in index.js

// Import the above file

Var studentroutes = require(“./routes/studentroute.js”);

/\* Below is saying hey, all the urls that will have localhost:200/api/something point to studentroutes. So all urls that are localhosta:200/api/….

Will be handled in studentroutes

\*/

Express.use(“/api”, studentroutes);

===================================================================================================================

**TO SETP UP A VIEWS ENGINE FILES**

Node.js has a lot of view engines like pug, ejs and others. So because of that, you need to tell node in your application what type of template engine you are using. And also the top folder in which all the template files are going to be in.

//SET VIEW ENGINE WITH EXPRESS.SET(“view engine”, “type\_of\_view\_engine”)

app.set("view engine", "ejs");

// SET THE FOLDER WHERE ALL SUB\_VIEW EJS FILES WILL BE KEPT with express.set(“views”, “top\_folder\_that\_contains\_all\_template\_files”);

app.set("views", "myviews");

// Then create a file student.ejs in the above “myviews” folder

//Then in index.js use the res.render(“views\_subfile\_name”, {object of values to pass to the file})

//HOMEPAGE ROUTE

app.get("/", (req, res)=>{

    res.render("student");

});

===================================================================================================================

**HOW TO HASH A PASSWORD WITH bcrypt**

First install bcrypt with:

Npm install bcrypt --save

TO HASH A PASSWORD

1. Import bcrypt module
2. Use bcrypt.hash(yourPlain\_pawword, anyNumber, callback\_funct\_that\_returns\_error\_or\_the\_hashed\_password);

// IMPORT BCRYPT FOR HASHING PASSWORD

const bcrypt = require("bcrypt");

// bcrypt.hash(plainPassword, number\_Of\_Times\_to\_hash password, (err, your\_hashed\_passWord));

  bcrypt.hash(body.password, 8, (err, hashedPassword)=>{

      //If error hashing password

         if(bcrypt\_err){

         }

         //If password is hashed successfully.

         else{

          // Save hashedPassword in db

         }

     } );

TO CHECK IF LOGIN INPUT PASSWORD IS WHAT IS HASHED IN DB

Use bcrypt.compare(loginInput\_password, hashed\_passwowrdInDb, callback\_funct\_that\_returns\_error\_or\_bolearn);

        //bcrypt.compare(plainPassword, hashedPasword\_in\_db, (err, valid));

      //  err => if error happen while checking the two password

      //  valid => returns boolean of either login\_input\_password is what is hashed in db or not

        bcrypt.compare(body.password, db.password,(err, valid)=>{

            // IF USER PASSWORD IS NOT WHAT IS IN DB

           if(!valid){

           }

           //IF USER PASSWORD IS WHAT IN DB

           else{

           }

       });

===================================================================================================================

**HOW TO AUTHENTICATE API WITH JWT (JsonWebToken)**

Is a way of protecting your API by generating a specific token for all your registered users after successful login. These users are the client that are going to use the API like angular, react, vue.js. these tokens must be included in the headers of every request the client will make except signup and login pages.

You also need to validate the token client will include to see if it’s the right token you generated for them when they login.

So there are two things involved:

1. Generate a token for them when they successfully login
2. Validate the token in every request except (login & signup)

First install jwt with:

Npm install jsonwebtoken –save

**To generate a Token After Login**

In login.js Generate a token for the user after successful login. By following the following

// IMPORT JWT

const jwt = require("jsonwebtoken");

// payload = the whole select user result in db,

// "12ghk3" = is any secret key you want to generate the token against.

// Always save this secret key in .env file becuase you will need it later when validating the token

// {expiresIn:"1h"} => the time the token will expire

//generate a token for the user

const token=  jwt.sign({payload: result}, "12ghk3", {expiresIn:"1h"} );

app.post("/login", (req, res)=>{

    // show the token to the user after you have validated his email and passowrd and is correct

    return res.status(200).json({

        success: 1,

        token: token

    });

});

**To VALIDATE A TOKEN SENT BY USERS IN THE HEARDERS**

Create a different file jwt-token-validator.js file. That contains an object of middleware function for verifying the token

// IMPORT JWT

var jwt = require("jsonwebtoken");

// export this object "checkToken"

module.exports= {

    // create an object of middleware function eg. app.use(req, res, next);

    checkToken:(req, res, next)=>{

// reg.get("authorization") below contians the header  authorization: "berear token" the user will include in every request

    var token = req.get("authorization");

    // if the headers contains a token

    if(token){

        /\* berear 6 is letter words plus a space and the token key. But you want to

        verify the token only. cut off the berear and the space with slice(7)

        \*/

        token = token.slice(7);

        // verify the token against the same secret key you used to generate it.

        // jwt.veryfy(heaerders\_Token, secret\_key\_you\_used\_to\_generat, callback\_func)

        jwt.verify(token, "12ghk3", (err, decoded)=>{

            // if token is not correct

            if(err){

           return   res.json({

                    success: 0,

                    message: "Invalid token"

                });

            }

            // if token is correct call the next() func to authorize the user to the requested url

            else{

             return  next();

            }

        });

    }

    // if the headers does not contain a token

    else{

      return  res.json({

            success: 0,

            message: "You did not add any token in the hearders"

        });

    }

    }

}

Then in index.js

Import the above middleware file and validate every route for the right token except login and signup

// IMPORT JWT TOKEN VALIDATION MIDDLEWARE

var {checkToken} = require("../../middlewares/jwt\_token\_validator");

// don't add the middleware here to ckeck login and signup token

app.post("/", createUser);

app.post("/login", loginUser);

// add the middleware to these route to authenticate the user before you authorize him

app.put("/",checkToken, updateUserById);

app.get("/",checkToken, getAllUsers);

/\* above createUser, loginUser, updateUserById, getAllUsers  are fuctions created to perform crude opertion\*/

===================================================================================================================

**HOW TO CREATE MVC CRUD IN NODE AND MYSQL**

For every single entity create a folder that will contains 3 files router, controller, service

Eg. Inside students folder student.service.js, student.controller.js, student.router.js

Basic example.

Inside index.js

var express= require("express");

var app = express();

require("dotenv").config();

// CONVERT ALL JAVASCRIPT OBJECT INTO JSON

app.use(express.json());

// IMPORT THE USERS ROUTEER

var userRouter  = require("./api/users/user.router");

//LET userRouter HANDLE ALL USER API THAT HAS /API/USERS

app.use("/api/users", userRouter);

//CRETATE THE SERVER

var portNumber = process.env.portNumber;

app.listen(portNumber, ()=>{

    console.log("server is listening on port "+ portNumber);

});

Then the following are the 3 files inside api/users folder

Inside user.router.js

// IMPORT EXPRESS

 var userRouter = require("express").Router();

 // IMPORT JWT TOKEN VALIDATION MIDDLEWARE

 var {checkToken} = require("../../middlewares/jwt\_token\_validator");

 // IMPORT USER CONTROLLER

  const { createUser,getAllUsers,updateUserById,deleteUserById,getUserById,loginUser} = require("./user.controller");

// USER ROUTES

  userRouter.post("/",createUser);

  userRouter.put("/",checkToken,updateUserById);

  userRouter.get("/",checkToken,getAllUsers);

  userRouter.get("/:id",checkToken,getUserById);

  userRouter.delete("/:id",checkToken,deleteUserById);

  userRouter.post("/login",loginUser);

module.exports = userRouter;

inside user.controller.js

//IMPORT THE USER SERVICE

const {create,getUsersById, getUsers,updateUserId,deleteUser,loginUser} = require("./user.service");

// IMPORT BCRYPT FOR HASHING PASSWORD

const bcrypt = require("bcrypt");

const jwt = require("jsonwebtoken");

var env = require("dotenv").config();

var salt = 10;

//EXPORT THIS MODULE

module.exports ={

     // CREATE NEW USER

    createUser: (req, res)=>{

     var body =   req.body;

     //HASH THE PASSWORD WITH bcrypt.hash(plain, password, 8, (callback))

     bcrypt.hash(body.password, salt, (bcrypt\_err, bcrypt\_success)=>{

         if(bcrypt\_err){

            return res.status(400).json({

                success: 0,

                message: "Erorr Please try angain"

            });

         }

         else{

             body.password = bcrypt\_success;

                 //CALL THE CREATE MODULE in users.service.js

     create(body, (err, result)=>{

        if(err){

            console.log(err);

           return res.status(400).json({

               success: 0,

               message: "Duplicate entry in db"

           });

        }

        else{

            return res.status(200).json({

               success: 1,

               data: result

            });

        }

    });

         }

     } );

    },

    // GET USER BY ID

    getUserById:(req, res)=>{

        id = req.params.id;

// call the getUsersByid module in users.service.js and pass in the values

        getUsersById(id,(err, result)=>{

            if(err){

                return res.status(500).json({

                    success:0,

                    message: "database connection error"

                });

            }

        if(!result){

            return res.status(500).json({

                success:0,

                message: "User not found"

            });

        }

        else{

            return res.status(200).json({

                success:1,

                data: result

            });

        }

        });

    }

/\*I have cut off some of the crude operations of this file \*/

}

Then inside user.service.js

 // IMPORT THE DATABASE CONNECTION

var connection = require("../../config/db.connection");

// EXPORT THIS MODULE

module.exports = {

    // OBJECT OF FUNCTION THAT INSERT VALUES TO DB

    create: (data, callback)=>{

        connection.query(

            "insert into registration (firstName, lastName, gender, email, password, number) values (?, ?, ?, ?, ?, ?)",

            [

                data.firstName,

                data.lastName,

                data.gender,

                data.email,

                data.password,

                data.number

            ],

            (error, result, fields)=>{

                if(error){

 /\*

 If there is error, go to

                \*/

                   return callback(error);

                }

                else{

                    return callback(null, result);

                }

            }

        );

    },

    //GET ALL USERS

    getUsers: function(callBack){

        connection.query("select firstName, lastName, gender, email, password, number from registration",(err, result, fields)=>{

            if(err){

                return callBack(err);

            }

            else{

                return callBack(null, result);

            }

        });

    },

    //GET USER BY ID

    getUsersById: (id, callBack)=>{

        connection.query('select firstName, lastName, gender, email, password, number from registration where id = ?',[id],(err, result, fields)=>{

            if(err){

                return callBack(err);

            }

            else{

                return callBack(null, result[0]);

            }

        });

    }

/\*I have cut off some of the crude operations of this file \*/

}

Then I also have a folder in my project root directory middlewares/jwt-token-validator.js for validation jwt token

var jwt = require("jsonwebtoken");

module.exports= {

    checkToken:(req, res, next)=>{

// reg.get("header\_Value\_key") below contians all the http hearder values

    var token = req.get("authorization");

    // if the headers contains a token

    if(token){

        /\* berear 6 is  letter words plus a space and the token key. But you want to

        verify the token only. remove the berear and the space with slice(7)

        \*/

        token = token.slice(7);

        /\* verify the token against the secret key you used to generate it

        in your .env file \*/

        jwt.verify(token, process.env.jwt\_secret\_key,(err, decoded)=>{

            // if token is not correct

            if(err){

                res.json({

                    success: 0,

                    message: "Invalid token"

                });

            }

            // if token is correnct call the next() func authorize the user

            else{

                next();

            }

        });

    }

    // if the headers does not contain a token

    else{

        res.json({

            success: 0,

            message: "Access denied unauthorized user"

        });

    }

    }

}

This is also my .env file

portNumber = 8000

db\_user= root

db\_host= localhost

db\_name= node\_jwt\_uthentication

db\_port = 3306

db\_password=

jwt\_secret\_key = justilato12345

Above is a basic single users entity crude project. The professional way.

===================================================================================================================

Image upload with Multer

Multer is a node.js middleware for handling multipart/form-data, which is primarily used for uploading files. It just a middleware that will run on the url you want to accept image on from the user.

Because of this, don’t use multer as a global middleware because you will not accept images on all your urls. But use it only as a middleware on the url that you want to accept image upload on.

NOTE: Multer will not process any form which is not multipart (multipart/form-data).

So don't forget the enctype="multipart/form-data" attribute in your html form.

Multer adds a body object and a file or files object to the request object. The body object contains the values of the text fields of the form, the file or files object contains the files uploaded via the form.

So you can access a single or multiple upload image files with the “file” and “files” object like:

app.post('/url', function(req, res, next) {

    // req.file 🡪 is use to access the single uploaded image file

    //reg.files 🡪 is use to access an array of files in case multiple files were uploaded

    // req.body 🡪 will hold the text fields, if there were any

// console.log(req.file) this will give you the file object including filename, fileSize, destination

// when an image is upload. It will print the following

/\* fieldname: 'image',

originalname: 'blog.jpg',

encoding: '7bit',

mimetype: 'image/jpeg',

destination: './upload/images',

filename: 'image\_1619032407465.jpg',

path: 'upload\\images\\image\_1619032407465.jpg',

size: 330196

\*/

})

To install multer use:

npm install --save multer

MULTER gives you these methods to upload files of different kinds:

#### .single(html\_inpute\_field\_name)

Used to upload single image or file. The single file will be stored in req.file.

#### .array(html\_inpute\_field\_name [, maxCount])

Accept an array of files, all with the name fieldname. Optionally error out if more than maxCount files are uploaded. The array of files will be stored in req.files.

Example: upload images, and check if error happends

//import express and inizalize it

var express = require("express");

var app = express();

// import the path module (you don't need to install it, it comes with node.js)

const path = require("path");

// import multer

var multer = require("multer");

// creat multer disk storage object

const storage = multer.diskStorage({

    //destination object to upload the image (this will be created by node when the first image is uploaded)

    destination: './upload/images',

    // the filename object (grab the htmlInpueFilename+currentMilliSeconds+theFileExtension)

    filename: (req, file, cb) => {

        return cb(null, `${file.fieldname}\_${Date.now()}${path.extname(file.originalname)}`)

    }

})

// assign the above distorage objec to multer Storage

var upload = multer({ storage: storage });

// This is the url the user will go to upload the iamge to Note==> upload.single("image") is just a middleware

// use "uload.single(html\_inpute\_field\_name) when uploading a single image"

app.post("/image", upload.single("image"), function(req, res, next) {

    res.json({

        success: 1,

        profile\_url: `http://localhost:8080/profile/${req.file.filename}`

    })

});

// create a multar error handling midlware function that you will use it to check error while uploading images

function errHandler(err, req, res, next) {

    if (err instanceof multer.MulterError) {

        res.json({

            success: 0,

            message: err.message

        })

    }

}

// use the above function as a middleware to run on all url

app.use(errHandler);

// port number  the server is running on

app.listen(8080, function() {

    console.log("server is running on port 8080");

});

// you can check it with postman by choosing formdata and selecting a file on this post url localhost:8080/image

===================================================================================================================

**REST API**

REST = means, representational state transfer

API = means, application program interface

NOSQL

NoSQL = not only SQL.