MEN

* Sheryians Coding School

Node.js

Made from v8 engine (chrome) – run js smoothly aside from browser. Hence, purpose of Node.js is to run js without browser help, run js in our Server(nodejs helps to create server).

Run time environment for JavaScript.

Installation: “node js download”

Download LTS (long term support) pack, LTS : name of function doesn’t change fast.

Cmd: node -v

Create new folder and open it in vs code.

Create new js file (say index.js)

To run this file within terminal (outside browser):

Open new terminal (Ctrl+J)

Cmd: node file\_name , node index.js

To clear terminal (Cmd: clear)

New line in terminal (Ctrl+l)

Now production level Project Folder Setup:

Open clean folder in vs code

Open terminal, Cmd: npm init -y

If error as “running scripts is disabled on this system”

First run Cmd: Set-ExecutionPolicy -Scope CurrentUser Unrestricted

Now above sterp will create package.json, indicates that we can start creating an application. => package.json is like label for our App (show what our app contains eg : dependencies: cat-me)

Node\_modules => that contains id in node\_modules(package for cat-me lies in node\_modules)

Cmd: npm i //to reinstall node\_modules, if we delete it accidently

Package: Reusable code that other coders wrote, now we also copy it. Eg: npm (installed with nodejs)

To use package from, we first download package using given commands in

terminal.

Now make a new js code, require the package and run it.

Ie: const catMe = require(‘*cat-me*’) //require

catMe(); //function call

console.log(catMe()) // print cat

Creating a server:

Require http, it is downloaded with nodejs

Create server

const http = require(‘http’) //require http

const server = http.createServer( (req, res)=>{ //create server

res.end(“Hello World”) //on any request on server its response by showing

HelloWorld

} ) //create server with callback function with request and response

server.listen(*3000*, *callback-optional*) // **RUNNING** server and our data pass to server

through port 3000

Cmd: to run surver dynamicly(realtime change), npx nodemon *filename*

Cmd: to stop server, Ctrl + c

Server Routing (Controller):

req.url gives the requested route of our server.

Eg; If user send request on “localhost:3000/about” here the route is ‘/about’ Similarly if user send request (or user go to) “localhost:3000” here the route is ‘/’.

A computer screen shot of a program code

AI-generated content may be incorrect.

Express:

In production level we don’t use above methods to create servers and routes, we use express.

Express is a package

Install Express package Cmd: npm i express

Require Express

Call express (Express is by default closed toolbox 🧰)

const express = require("express"); //require

const app = express(); //opened express Toolbox and tools are now stored in ‘app’

variable

app.get(‘/’, (req,res)=>{ //creating route for home page

res.send(“Home Page”) // to send response

})

app.listen(3000); // to run server(no need to create server)

NOTE: Express too use http for creating server

Rendering HTML using express in FrontEnd:

For that we need to setup VieweEngine, we use ejs

Download ejs Cmd: npm i ejs

Add code: app.set('view engine', 'ejs');

in app.js file, just after the Express call.

Create new folder inside Project Folder with name “views”

Inside views we create file that contain HTML code but with .ejs extension (say index.ejs)

Code: res.render(‘index’) //inside route

MiddleWare:

If we want any request to reach a certain function before going to certain route, we use middleware.

They are three types: All three types runs for all route by default

Built-in

Custom

Third Party

Custom Middleware:

Code: app.use((*req*, *res*, *next*) => {

 console.log("Middleware 1 Passed");

return next(); //to continue flow of program

}); //runs before every route

Code: app.get( "/",  (*req*, *res*, *next*) => {

    console.log("Home route Middleware");

    return next();

  }, (*req*, *res*) => {

    res.send()

    console.log("Home")

  }); // Middleware for route specific request

Third-party Middleware:

To use 3rd party middleware, we first must download it. For eg we use “morgan” (also known as logger), which gives the Method(app.get), route(‘/’), ResponseCode, ResponseTime

NOTE: Must READ Documentation

Cmd : To download, npm i morgan

Require morgan

Use morgan Code: app.use(morgan("dev"));

Built-In Middleware:

Form Control:

Is used to take data from user in frontend and take it to server and use it.

Create form in ejs file *//better use text type for email too*

A screen shot of a computer code

AI-generated content may be incorrect.

To take data in backend, we must have certain route, create new route and pass that route inside action in ejs file (in above’s first line)

From ‘get’ method the data comes to req.query //print it

A screenshot of a computer code

AI-generated content may be incorrect.

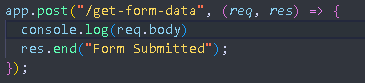
When we print req.quey it prints bad data, so we have to give the name for every input as, *//better use text type for email too*

A screenshot of a computer screen

AI-generated content may be incorrect.

Now we can have data in backend in un understandable form.

NOTE: When we submit it, the data is also shown on URL which is not good, so we use app.post() instead of app.get() during route creation. NOTE: The data now doesn’t come to req.quey it comes to req.body.



NOTE: Form is by default in ‘get’ method; to make it post we must define its method as post.

A screenshot of a computer screen

AI-generated content may be incorrect.

After all this, Now, express can’t read data (undefined). So we need to use two Built-in middlewares:

Codes: app.use(express.json());

app.use(express.urlencoded({ extended: true }));

Link CSS in Frontend

Create new folder usually “public”, inside this we make all files required for Frontend.

Create a CSS file in public folder.

NOTE: CSS is a static file: No need to authenticate user. User can access it from server without authentication. So, we need another Built-in middleware as

Code: app.use(express.static (“public”));

Now link this CSS in ejs file of “views” folder

Code: <link rel="stylesheet" href="style.css"> //just above head tag

Link Other JS in Frontend

Same as up but, during link in ejs,

Code: <script src="script.js"></script>

Code: <script src="/assets/script.js"></script>

MongoDB:

Install

* [MongoDB\_Community](https://www.mongodb.com/try/download/community) (Server): Data is stored here
* Compass: To visualize saved data

CMD: npm i mongoose

Create new folder models, followed by js file (say user.js): In user.js we make schema

Schema: Tells about the basic property of washingmachine (user/,model).

Eg: Color, Volume etc

In user.js:

Require mongoose: const mongoose = require("mongoose")

Create Schema Object:

const userSchema = new mongoose.Schema({

  username: String,

  email: String,

  password: String,

  age: Number,

  gender: {

    type: String,

    enum: ["M", "F"],

  },

});

Implementing above schema:

const userModel = mongoose.model("*user-model\_name*", userSchema);

Schema is in user.js file, to use this in our app.js we need to export

module.exports = userModel;

Require exported user model

const userModel = require (“./models/user”);

Now we have to connect to our database, to do so again make new folder “config” followed by “db.js”

In db.js:

Require mongoose: const mongoose = require("mongoose");

Connect local DB:

const connection = mongoose.connect("mongodb://0.0.0.0/men").then(() => {

  console.log("DataBase Connected");

});

Export: module.exports = connection;

Require it in app.js: const dbConnection = require (“./config/db”)

Creating a user: (app.js)

* Taking data form frontend (form)
* Take data to server and create user

Create new route “/register” and render any ejs file(register.ejs) on request. [Create new ejs file in views, say register.ejs]

A computer screen shot of a computer code

AI-generated content may be incorrect.

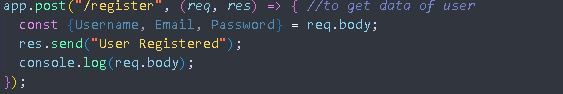
A computer screen shot of a code

AI-generated content may be incorrect.

NOTE: Change in method changes the route.

Now, data comes to the server, It’s time to create user in DB. To do so, we have to extract data from req.body.

const {Username, Email, Password} = req.body; //destructuring



NOTE: Any operation to be performed in DB userModel is compulsory.

Create user in DB:

userModel.create({

    Username : Username,

    Email : Email,

    Password : Password

  });

NOTE: This create is an asynchronous function so correct syntax will be

A screen shot of a computer code

AI-generated content may be incorrect.

We can also create new const and send user data as respond as

A screen shot of a computer program

AI-generated content may be incorrect.

🤧☠️: Don’t make object of req.body as same as Schema field

A screen shot of a computer code

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

Mongoose gives us unique id for each data, even if data are same



User created.

Full Stack Dev Must know CRUD Operation:

C- Create //done

R- Read

U – Update

D -Delete

READ:

Read data of Created user. Two methods;

1.

Create new route say “/get-user”

userModel.find();//take all user of database and return it inside then( (*users*)=>{} )

NOTE: In any condition it reads array (even data is not found- empty array)

Code:

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

  userModel.find().then((*users*) => {

    res.send(users);

  });

});

For conditional finding

Code:

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

  userModel

    .find({

      username: "d",

    })

    .then((*users*) => {

      res.send(users);

    });

});

One User Finding //find first registered user from db of multiple same *username*

NOTE: If user not found return null

Code:

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

  userModel.findOne({

    username: 'a',

  }).then((*users*) => {

    res.send(users);

  });

});

Update:

Read then update;

* .findOneAndUpdate(*obj, obj*) //is as asynchronous function

Create new route

Code:

app.get("/update-user", async (*req*, *res*) => {

  await userModel.findOneAndUpdate(

    {

      username: "a", //read

    },

    {

      email: "a@gmail.com", //update

    }

  );

  res.send("User Updated");

});

Delete:

* findOneAndDelete(*obj*) //asynchronous

Create new route

Code:

app.get("/delete-user", async (*req*, *res*) => {

  const deleted = await userModel.findOneAndDelete({

    username: "a",

  });

  res.send(deleted);

});

GOOGLE DRIVE PROJECT: