Node.js

Node.js is a JavaScript runtime environment, it allows you to run the JavaScript in the backend, you can perform various backend operations through node.js like accessing files, accessing os, accessing database, middleware, server and etc.

Earlier Javascript was used only for the front-end and it was understood by browser, but now with the help of node.js you can run the javascript without browser.

Node.js uses some libraries to run the JavaScript code, these libraries are called as node\_modules which provides the platform to execute JavaScript.

Note: You can run the JavaScript file directly using node.js, but if you want to run in browser, then you must include JavaScript in the HTML file and run the HTML.

Benefits

* Developers don’t have to switch from one language to another language when they want to communicate from the client to server applications i..e, front-end to back-end
* You don’t need ay interpreter because JavaScript can be executed directly from the node.js server
* Enables developers to write applications in one language for both front-end and backend
* JSON is the command format used to interchange the data between the front-end and backend, It is the native language for the JavaScript and you don’t need any kind of parsers to understand JSON
* JavaScript is the language used in various NoSQL databases like MongoDB, CoucheDB, interacting with them is much easier if you use JavaScript at the backend.

Event Loop & Asynchronous IO (non-blocking IO) mechanism

Node.js uses this mechanism at the backend which is very similar to browser to handle IO operations without blocking, because the IO operations are done by callback functions

Suppose your IO operation is synchronous

some statements; // 1st line

query(“select \* from emp”); // 2nd line

some statements; // 3rd line

The 3rd line is not executed until the query() execution completes, here the query() waits of the database to return the result, as it is interacting with the db, So here 3rd line is blocked, it is a blocked IO operation

In Non-blocking IO operation the tasks wouldn’t be blocked, because it is asynchronous

some statements; // 1st line

query(callback\_function); // 2nd line, callback would have the db interaction

some statements; // 3rd line

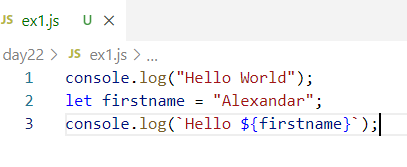
The 3rd line is not blocked, it will be executed after the query(), but callback is asynchronous and it might be executed later

Here node.js uses a single thread model and event loop will add the task to the event queue and pushes the task to the callstack for execution, if there’s a callback in any of the task, it will be called later.

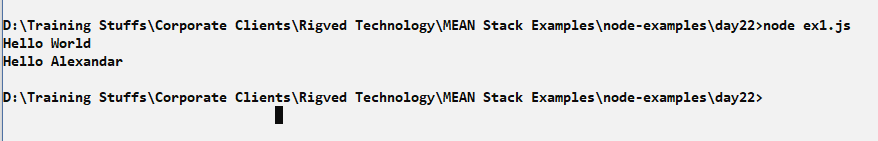
Note: Node.js supports all the new features of JavaScript like let, const, arrow, template string, padStart() padEnd(), rest, spread operators and so on.

Note: Since node.js runs the JavaScript at the backend, it doesn’t support some of the inbuilt objects we have in JavaScript that works in browser like document, sessionStorage, localStorage, alert()

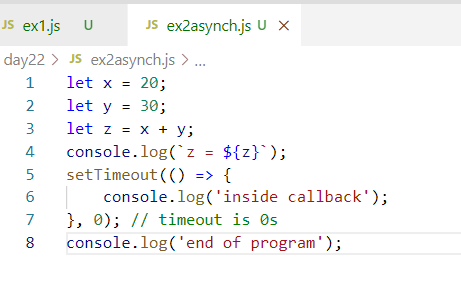
ex1.js



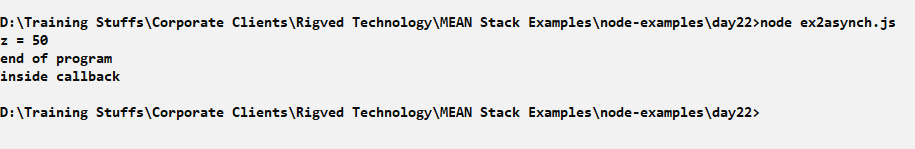
Output:



ex2async.js



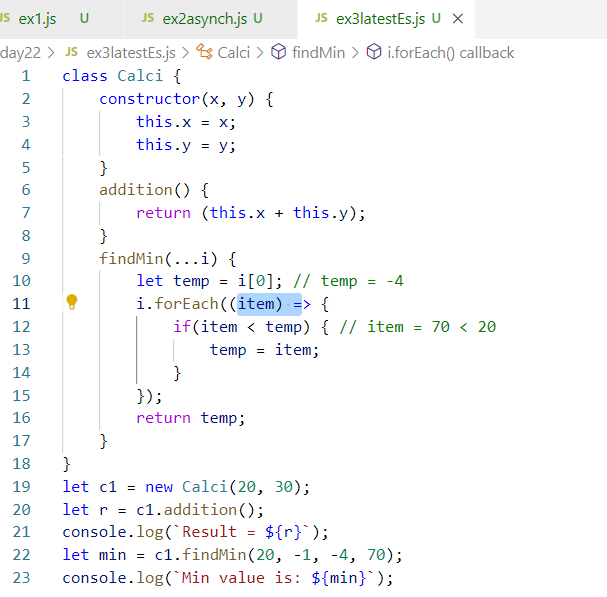
Output:



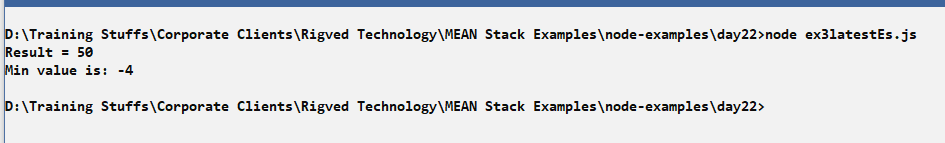
Here though setTimeout() is taking 0seconds the callback is added to the event queue after the console.log(‘end of the program’), hence its executed after end of the program

Using some new features of JavaScript

ex3latestEs.js



Output:



Node.js Modules

Modules are Javascript functions which are reusable and can be called from other Javascript functions.

There are mainly 3 types of modules

1. Core Modules: These are inbuilt modules that you can use in node.js by importing

Ex: http, fs, os, process, url

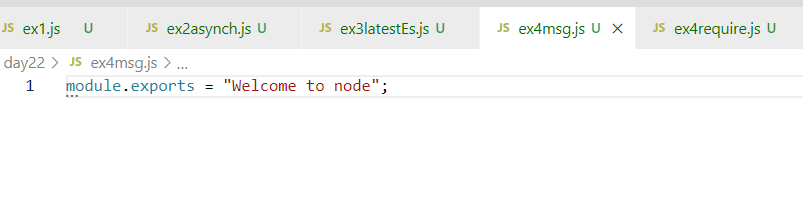
1. Local Modules: These are modules which you create and use in the program
2. Third party Modules: These are modules that are downloaded from the internet and can be used in the program

Ex: express, typescript, mongodb, bootstrap and etc

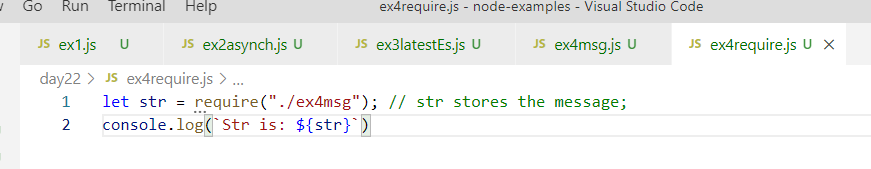
Local Modules:

These are the functionalities that can be created in a JavaScript file and import in another JavaScript file

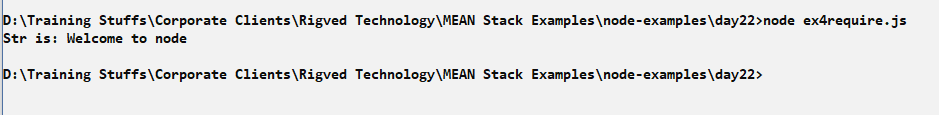
ex4msg.js



ex4require.js

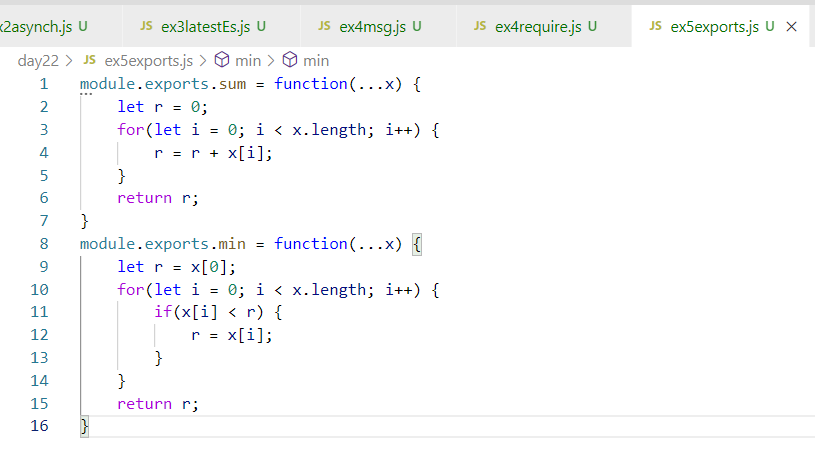


Output:

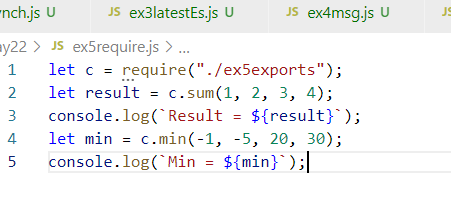


Creating multiple exports in the module with some name

ex5exports.js



ex5require.js



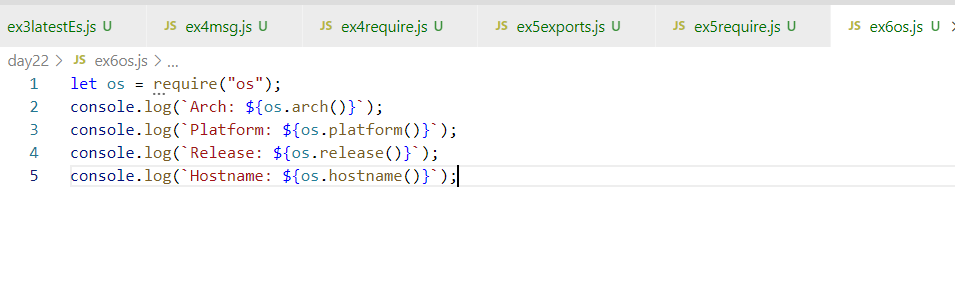
The above modules are local modules, which we created and imported, but we can also use lot of inbuilt modules & third party modules by downloading from the internet.

Using Inbuilt modules like os, fs, http, util, queryString and so on.

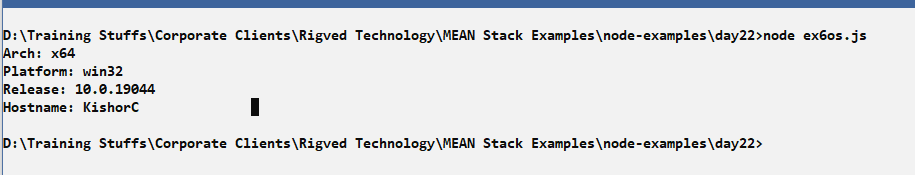
OS Module:

You get OS related informations like arch(), platform(), release(), type(), freemem(), totalmem() and so on.

ex6os.js



Output:



Third-party modules

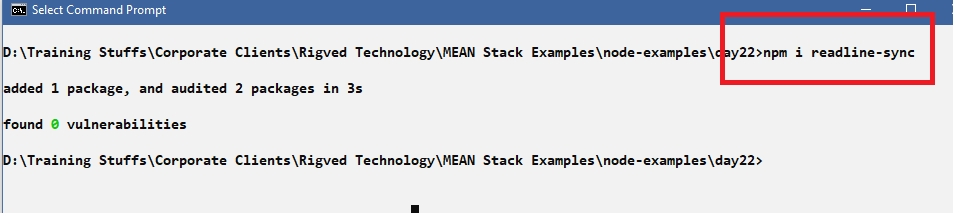
These are the modules available in the internet that need to be downloaded using npm

When you use npm a library will be downloaded and package.json file is the configuration file that will have the entry of the library downloaded.

Some of third party library are: readline-sync, express, typescript and so on.

readline-sync: It is used to take input from the keyboard, it is synchronous means all the functions are synchronous

Installing readline-sync

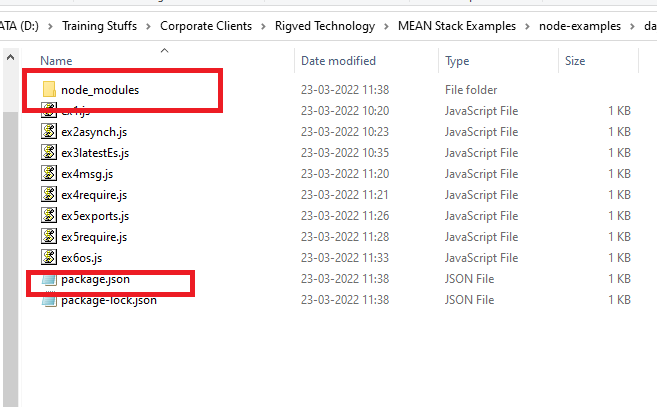


You will see node\_modules & package.json file in the folder that downloaded readline-sync.

node\_modules: Has the realine-sync library and their dependent libraries

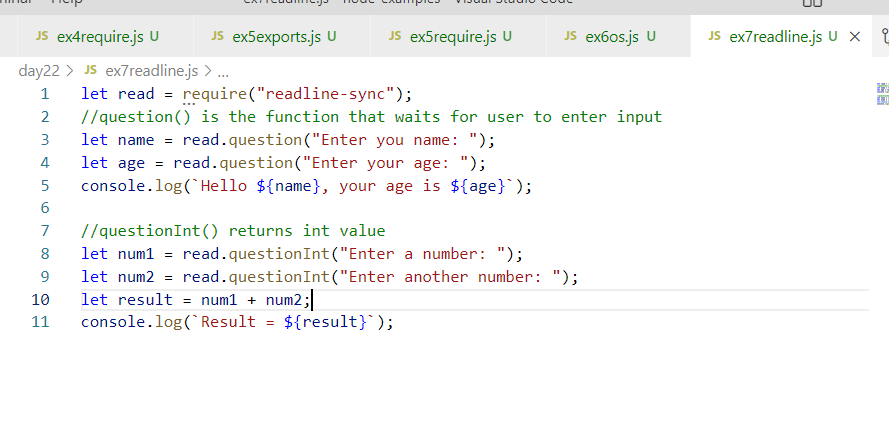
package.json: It is a configuration file in node.js this will have the entry of the library downloaded

package-lock.json: It is an auto-generated file you can ignore

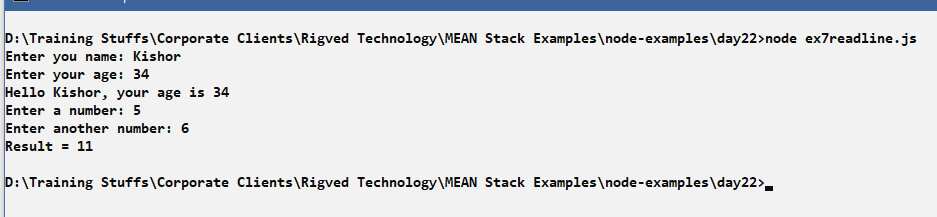


Now you can import the readline-sync library and take input from the keyboard

ex7readline.js



Output:



File handling

You can read/write data from/to file, it can be any kind of data like binary data or character data, in most of the case you also use json files which will have character data(text data). Node.js provides modules to perform read/write operations in both synchronous and asynchronous away.

fs module: It is an inbuilt module used to handle file data, it provides inbuilt functions to read / write data, it has functions as below

readFileSync: Reads the file in a synchronous mode

writeFileSync: Writes the file in a synchronous mode

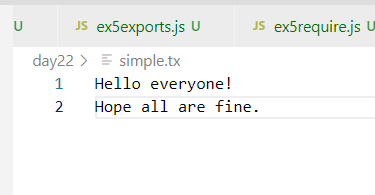
readFile: Reads the file in an asynchronous mode

writeFile: Writes the file in an asynchronous mode.

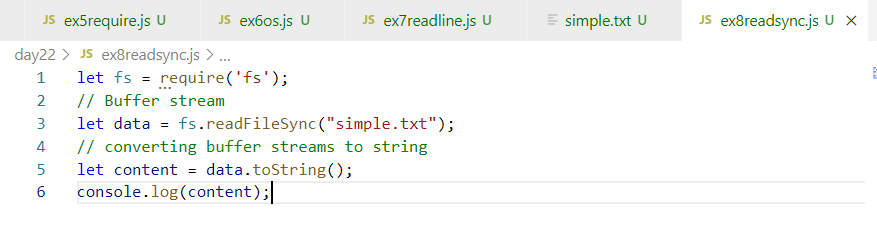
Read operation:

Create one text file that will have some data

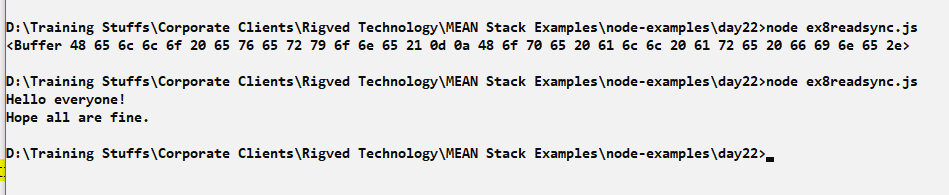
simple.txt



ex8readsync.js



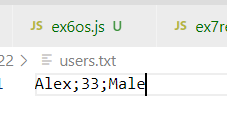
Output:



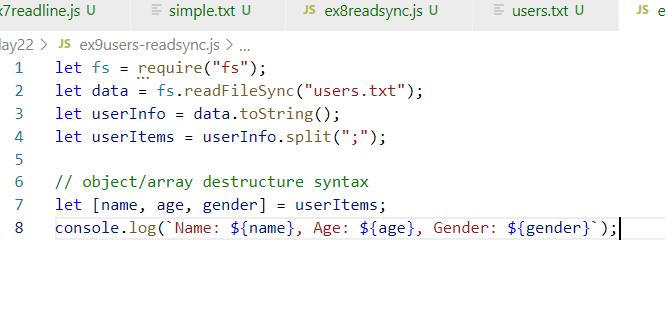
Reading the file that will have user information

Create a file that will have user data separate by a delimiter

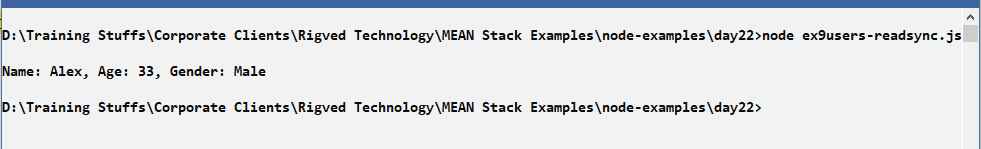
users.txt



ex9users-readsync.js



Output:



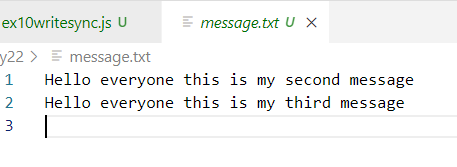
Writing file synchronously

You can use writeFileSync(), it takes 3 arguments, the 3rd argument is an object which will have a flag property that says append with ‘a+’.

ex10writesync.js



Output:



You can read and write json files

JSON files are a text files that can be used for configuration, database and so on, you can store some data and you can access the JSON data, JSON is native to JavaScript it can easily understand JSON and Javascript has some functions to convert JavaScript to JSON and vice versa

JSON.parse(jsonStringContent): It takes the JSON and converts to JavaScript object

JSON.stringify(javascriptObject): It takes the JavaScript object & converts to JSON

Note: We can’t store JavaScript objects’ in a file, we need to store them in a text format, JSON is the text format for JavaScript object

How the JSON data looks

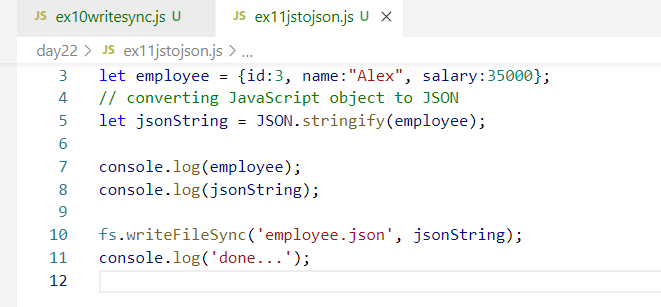
{“key”:value, “key:value”,…}, {“key”:value, “key”:value,…}

JSON file can also have JSON array which is a multiple JSON wrapped in []

[  
 {“key”:value, “key:value”,…},

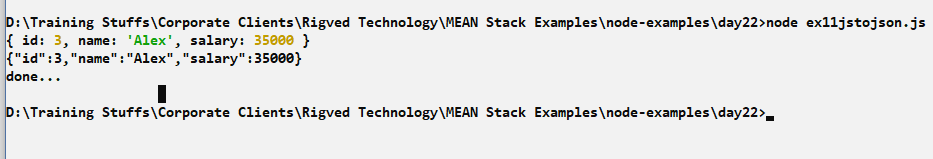
{“key”:value, “key”:value,…}  
]

Converting JavaScript object to JSON

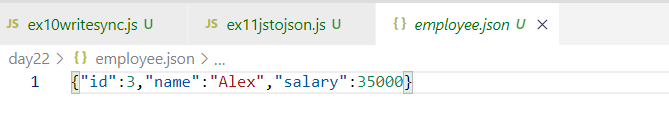


You can’t write JavaScript object into any file, you need to convert to JSON string and write to a JSON file

Output:

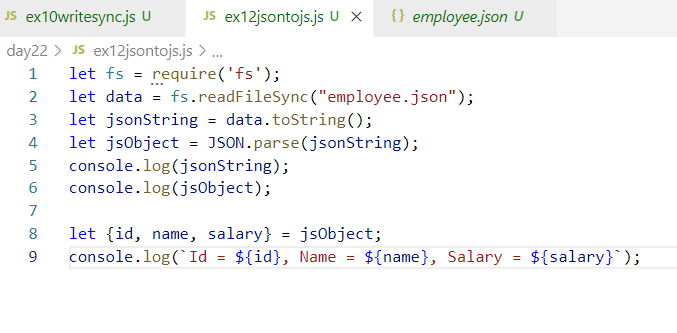


employee.json

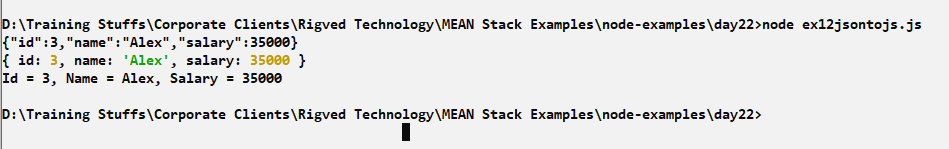


Note: Multiple JSON files need to be separated by , comma

Reading the JSON and converting to JavaScript object



Output:



Activity:

1. Try out all the above activities
2. Take input from the keyboard and read data like id, name & salary from keyboard and store this in a JSON file
3. Same above example do it for multiple records i.e., you must able to store multiple JSON data in the JSON file, but each data should be separated by comma
4. Continue the above example, and read the JSON file having multiple data and convert to Javascript object and print all the information in the console

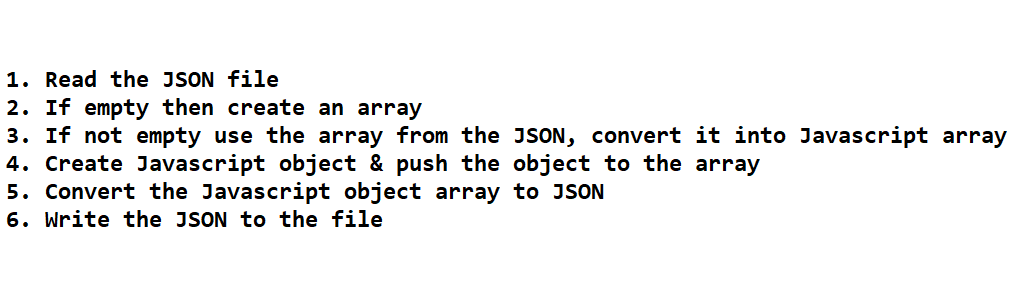
Hint: Multiple data/json/javascript object is represented by array

Solution:

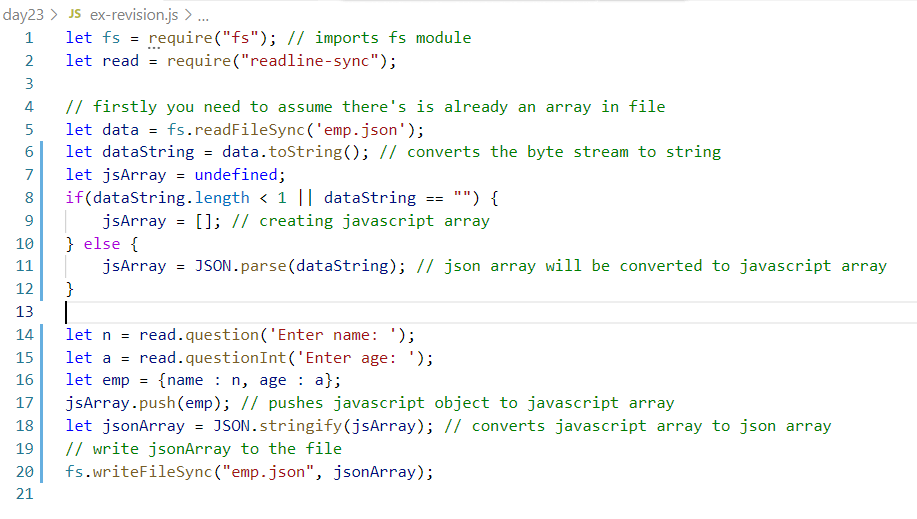
You need to store array of JSON in the file i.e.,

[{…}, {..}, {….}]

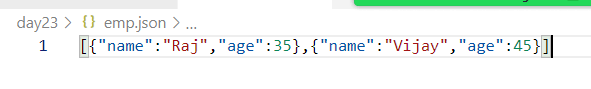
Means you need to write the json array not just json.



Solution:



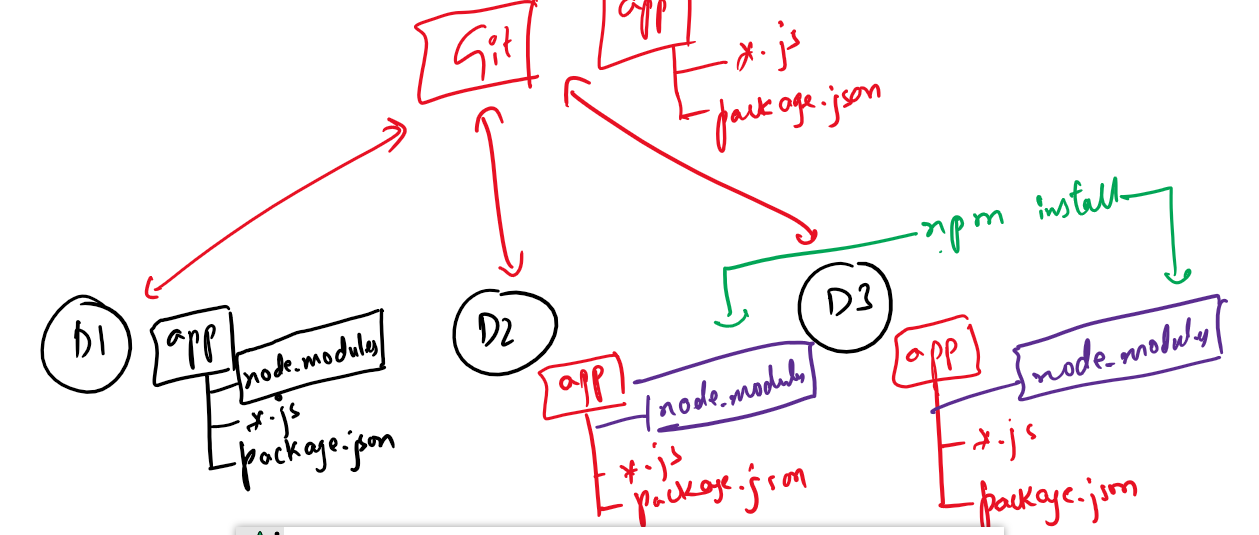
emp.json looks like below:



package.json:

It is a configuration file used in node.js,

* It can have dependencies entry that shows the libraries the node has downloaded in your working directory
* It can have commands that can be used with npm to start/run the application, test the application, build the application
* It can have even the entry point file, so that when you start the application the node can execute the entry point file
* It is used to re-download the dependencies in the working directory when developers share the application to multiple developers
* npm install would install all the node modules in the working directory.



Here D1, D2, D3 are developers who are working in the same project and they can download all the necessary libraries required for the application using npm install, which looks package.json and downloads the modules in node\_modules, so that you don’t have to push the node\_modules to the GIT

also, if any new library is downloaded in anyone’s machine its entry will be there in package.json, so when it is pushed to the GIT, other developers would pull the updated package.json, then enter npm install to get the new library in their working directory.

Node.js web application

It is used to write client & server programs where client sends the request & server generates the response, in order to develop web applications you need a server that can launch your application, node.js provides an inbuilt module called http module which can be used to develop web applications & create servers

Http module

It is one of the inbuilt module in the Node.js allows us to create server, handle request, generate response.

How to use HTTP module

let http = require(‘http’); // this references to http module, from this you can handle request, start server and so on.

How to handle request and start the server

*http  
 .createServer(callback)  
.listen(port, callback);* // all the 3lines can be written in a single line also as below

(OR)

*http.createServer(callback).listen(port, callback);*

Here

http is a module to create web application.

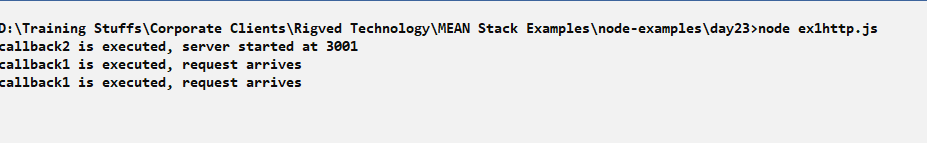
createServer: It is an inbuilt function inside http module to handle the request & generate the response, it takes a callback which is executed whenever the request comes to the server, it returns a server instance on which you can call listen

listen: It is also an inbuilt function that can specify the port number of the server in the 1st argument, the 2nd argument is a callback that is executed when the server starts

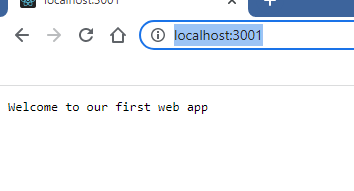
ex1http.js



Output in node console



Output in browser console

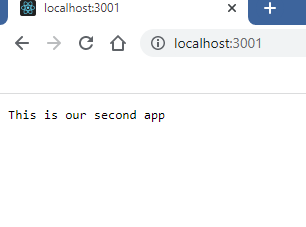


Note: You can pass the callbacks without creating by name in the function argument itself

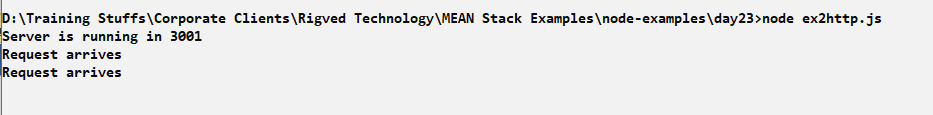
ex2http.js



Output in browser:



Output in console:



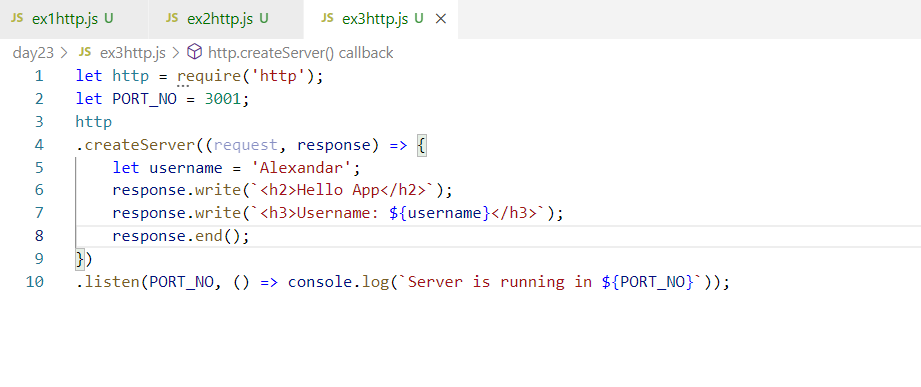
Writing HTML content in the response

By default the response content will be in text format, if you embed HTML code then it will be considered still as Text, to specify the text is an HTML content, you need to set response content using writeHead() function.

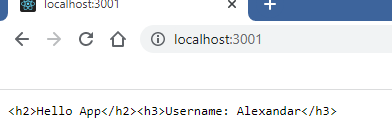
response.writeHead(status, {key : value});

Here status can be any HTTP status codes like 200 for success 404 for not found or errors, key: value are the response header properties like content-type, content-length and so o.

Without using writeHead() HTML code will not be recognized, as they are treated as text.



Since the content are treated as text, you will see html tags in the browser.



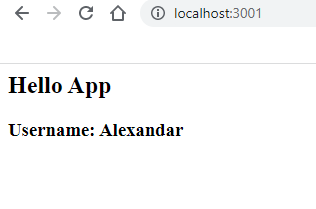
To avoid this we need to set the response header

ex3http.js



Because we have used response.writeHead(statusCode, headerProperties), we could able to see HTML content

Output:



Reading the data from the client

Client can send the data through query parameter or form body, we can extract those data at the server side

Query parameters: These are the data which are sent in the URL of the browser, query parameters will be in key value pairs after the url?, the data will be read at the server side

ie., <http://ip:port/path?key=value&key=value&> …

Form body: These are the data which user can enter and submit to the server so that server can read those data

How to read query parameter

You need to read the url using one url property of the request object i.e., request.url, this gives the resource url the user is trying to access, if the url has query parameters then you can read that too but for that you need to use url module and parse that query parmeter.

let url = require(‘url’);

If you read url through request as below

let reqURL = request.url; // if the reqURL is somePath?name=Alex&age=35

Then you can extract the query parameters as a javascript object through parse method of url module

i.e.,

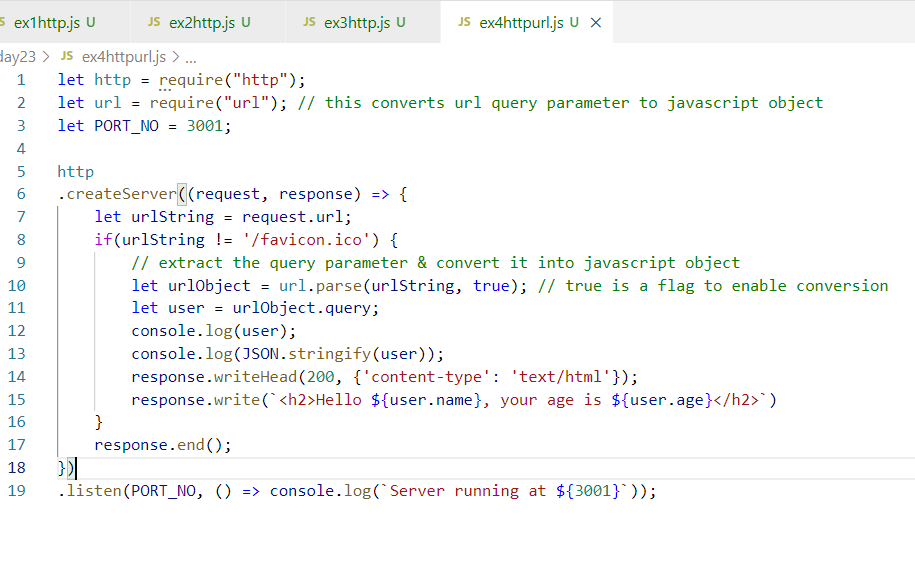
let urlString = url.parse(reqURL); // converts the url into javascript query object, then you can access that query property to get the query parameters

let obj = urlString.query;

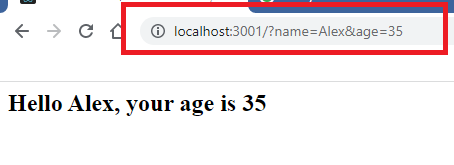
Note: Whenever you send a request to the server there will be two request sent one to the server and another to /favicon.ico, hence we must write an if condition to only get the request to the server.

Note: This problem you don’t face in other modules like express module

ex4httpurl.js



Output:



Activity:

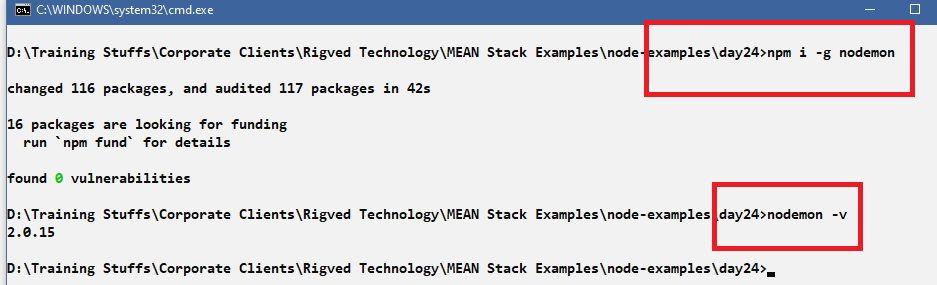
1. Try the above examples
2. Pass the query parameter in the URL and store that as a JSON data in the JSON file, it must retain the old data without erasing the old data, you can store name & age in json format.

Nodemon: It is a library used to auto-detect the changes in the code, instead of node you can use nodemon command to run the script, which can auto-detect the changes, so that you don’t have to stop & re launch your application

You need to install the nodemon library, you can install it globally so that it will be available in other directories also

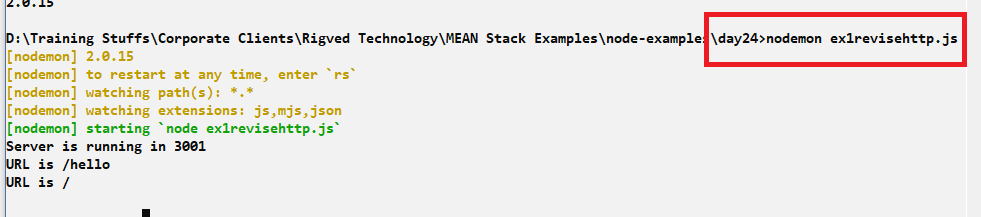
>> npm i -g nodemon

Installing nodemon

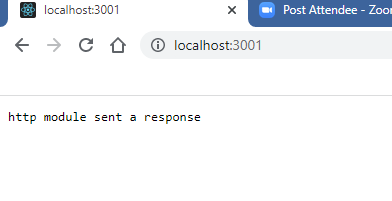


Now onwards we can use nodemon to run the script instead of node, but whenever you want run a different script you need to stop nodemon and run that script

Running the script



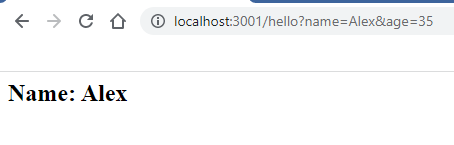
Output:



Do some changes in the code, you can see nodemon auto-detects it, example we will import the url module and read the query parameter



Output:



ReSTful Webservices:

Webservices are the services which are available over the internet with URL, it can be accessed by applications written in any language, it allows applications of different languages to exchange the data, most of the times the data will be in JSON format, which will be converted to the language the application understands.

ReST stands for Representational State Transfer, it means Representing data in various formats like JSON, XML, Text, CSV and transferring to the applications.

Two applications can exchange the data in a common format they can understand and convert that format to the type it can understand and do the task.

ex: Google pay can exchange the data to various banking applications written in different languages, google pay is one application that can communicate or exchange the data with many languages, it is possible because these banking apps are webservices which can consume the common format data and produce common format data the google pay can also understand & convert that to the type google pay understands, similarly phone pe also can understand these formats, similarly ATM machines, swiping machines all can access different banking web services

Postman application: It is used to test the webservices we create.

ReST Webservices:

you can create webservices in any languages could be in Javascript, Java, C#, Python, C++ and so on, these webservices will have 2 important information’s for the client.

1. URL of the webservice: Location of the webservice
2. HTTP methods to call the webservice: Type of operations the webservice does, like CRUD operations

Http methods: There are 4 methods in HTTP which is provided for users and application to implement different types of operations while perform CRUD operation

1. GET: It is used when the operation is fetch or retrieve.
2. POST: It is used when the operation is store or creating new resource
3. PUT: It is used when the operation is updating or modifying the existing resource
4. DELETE: It is used when the operation is removing any resource

These HTTP methods give idea to both consumers & producers, because along with URL the client also use HTTP methods to access the webservice.

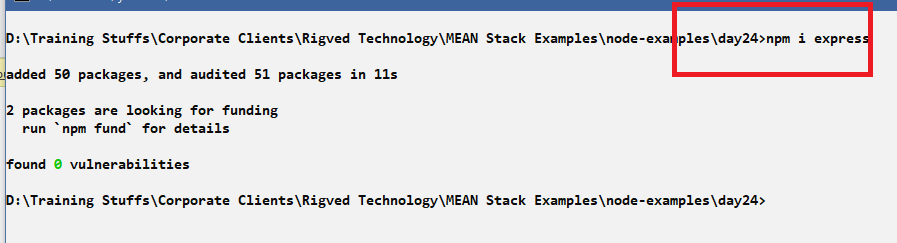
How to implement webservices in node.js

In Node.js there’s a third party module called express to create webservices.

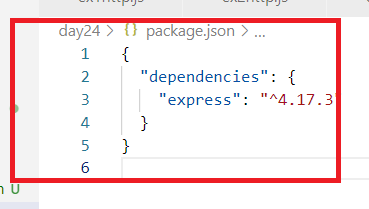
express module

It is a javascript module which you need to download to implement ReST webservices, it provides inbuilt functions that allows you to configure URL & HTTP methods to implement webservice

How to install express



You can verify the express installation in package.json



Now you can create ReST webservices through express module.

Use below code to import express

let expressModule = require(“express”);

The expressModule has a IIFE that needs to be called to create express object

You can create express module object as below

let app = expressModule(); // creates the object of express

From this app you can create webservices with some URL and use some http methods

app.get(url, callback): this creates a webservice with GET

app.post(url, callback): this creates a webservice with POST

app.put(url, callback): this creates a webservice with PUT

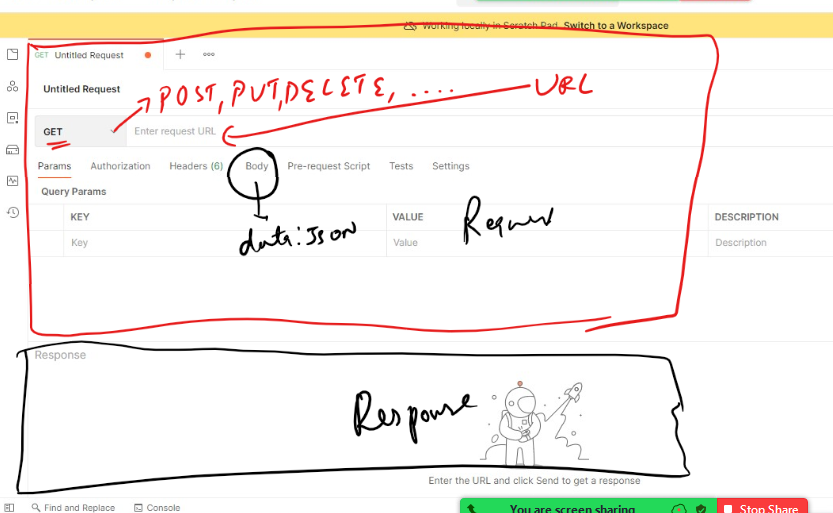
app.delete(url, callback): this creates a webservice with DELETE

All the above functions are part of express object, the callback executed when the client sends request to appropriate webservice

app.listen(port, callback): This runs the server in specific port, callback is executed once the server starts.

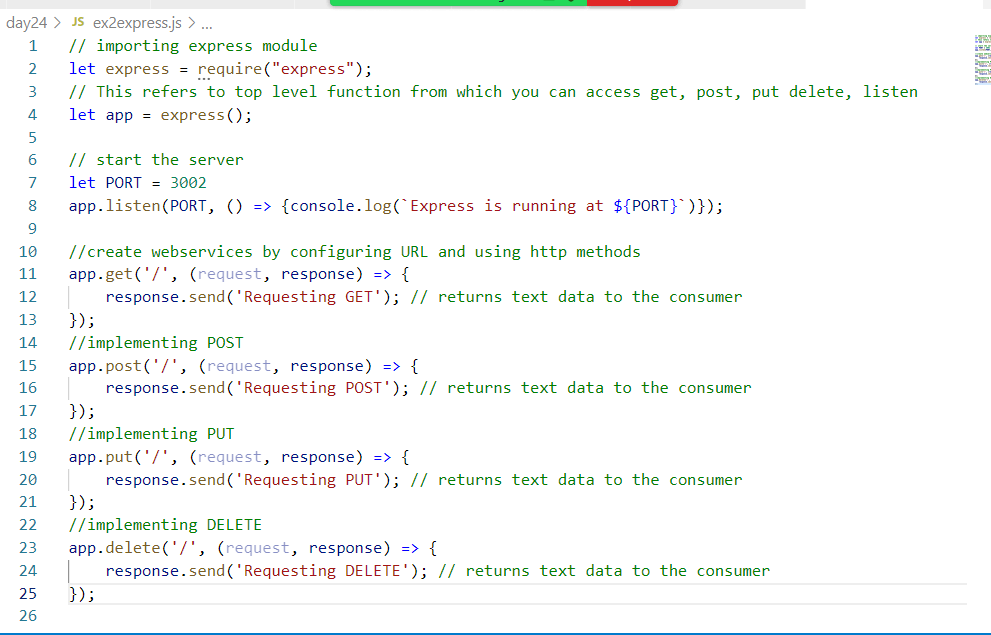
Postman: It must be installed to test your webservice, it can be installed from below website

<https://www.postman.com/downloads/>

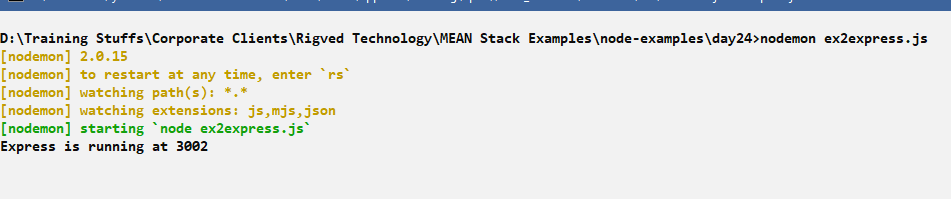


Create simple webservices that can return some text for all 4 HTTP methods.

ex2express.js



Run this program using node or nodemon

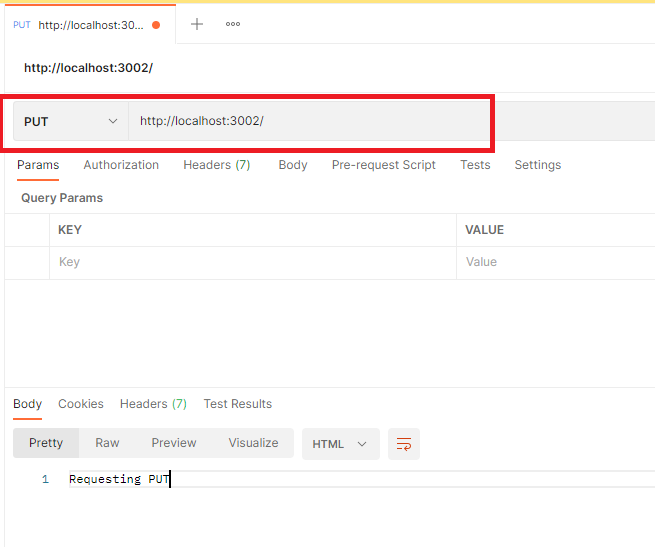


Use postman to access all the webservices, we have 4 webservices all of its url is same i.e.,

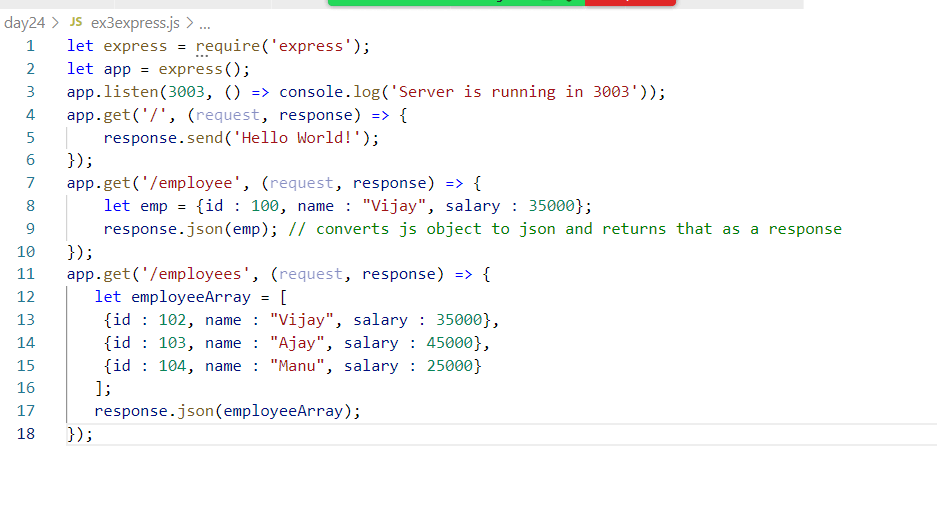
<http://localhost:3002/>

But they can be access with 4 HTTP methods

* GET, POST, PUT, DELETE.

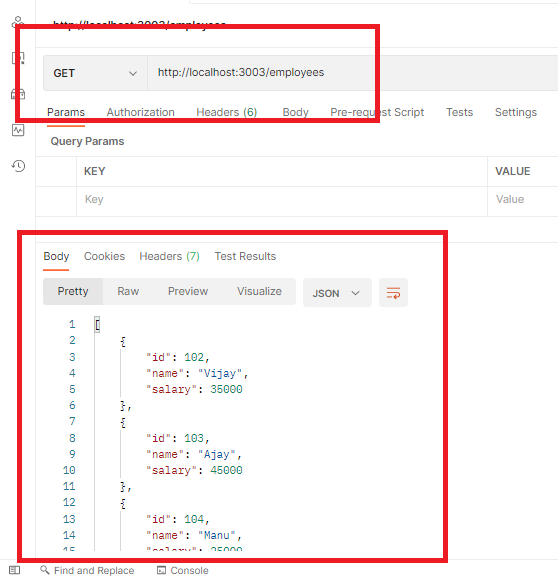


Currently the webservice is return the text response, but it can return JSON also, it is done with the help of json() function present in the response.



response.json() takes the javascript object & converts to json, it eliminates you writing JSON.stringify() and then return that JSON to client

Output:



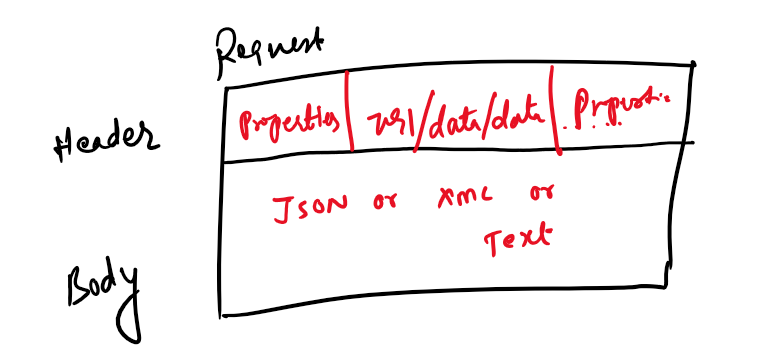
Activity:

1. Try the above exercise
2. Using the fs module read the json file and return the json data via express module GET & with some URL, Test this in postman

Sending data to the webservice from the client application

There are two ways you can send the data to the webservice from the client side via

1. Request URL
2. Request Body

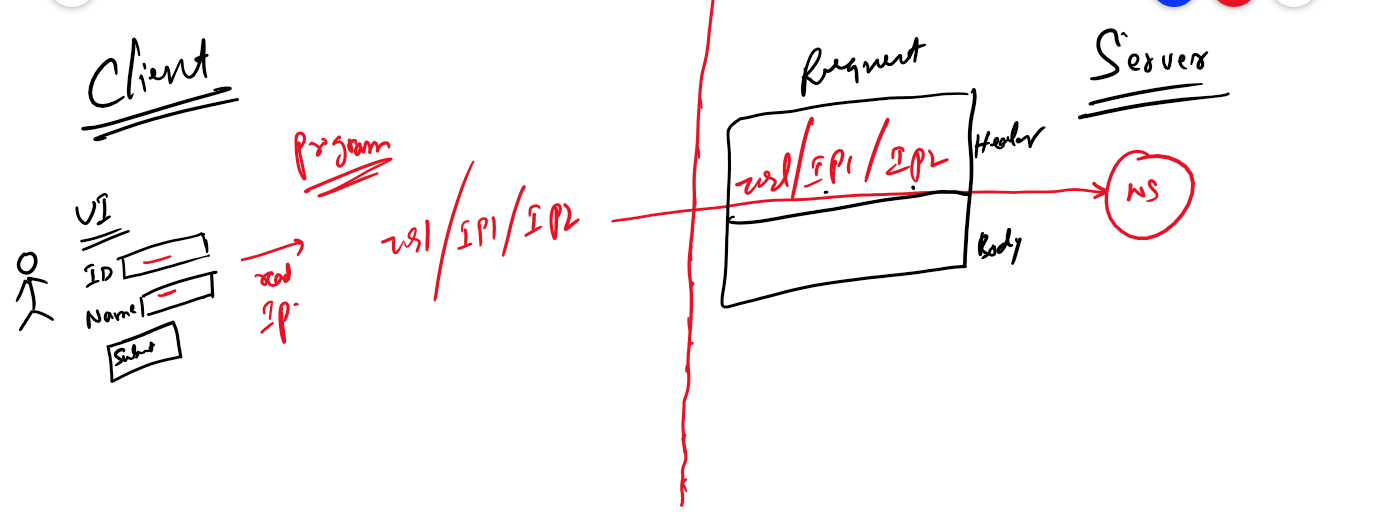


Data through URL: If client sends the request and the data through URL, data needs to be separated by /

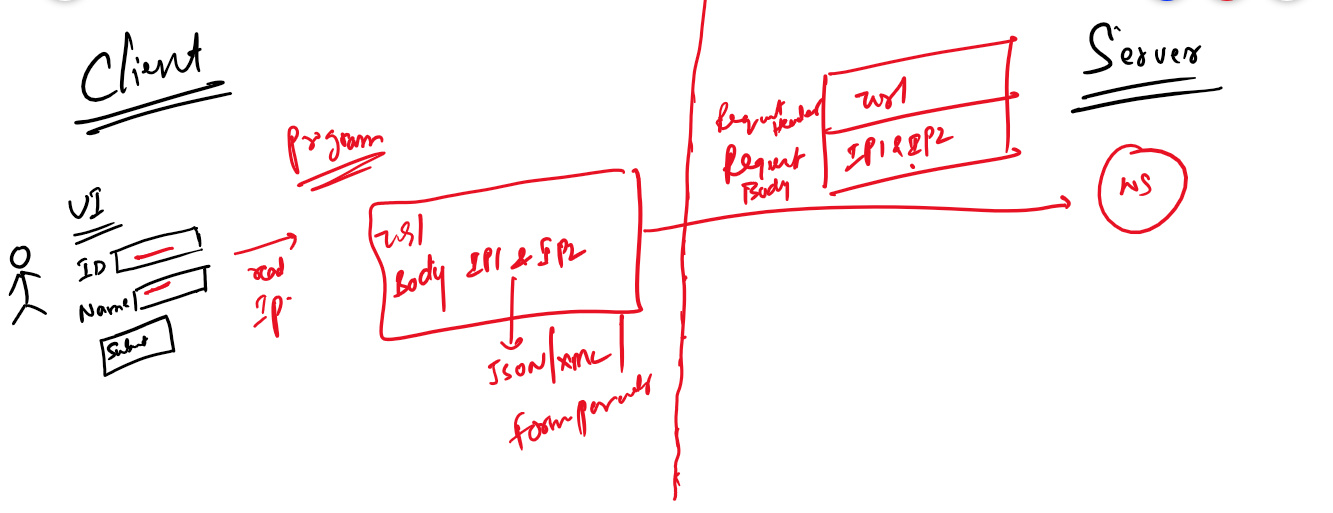
i.e., path/data

ex: <some-url>/100: Here 100 is the data

ex: <some-rul>/200/Ajay: Here 200 & Ajay are the data



Data through Body: If client sends the request and the data through Body, data could be in Form parameters, JSON, XML and so on



How the webservice reads the data coming from the client

Webservice can read the data from the request object, this request object has some properties that allows you to extract the data present in the URL or Body of the request,

Some of the properties of the request object are:

1. params: This reads the data from the URL of the request

i.e., request.params.parameterName: This extracts the data present in the url

1. body: This reads the data from the Body of the request

i.e., request.body: This extracts the data present in the request body

Why there’s a parameterName in the params

Because when client sends the request there will be a webservice that can accept the data via url with a key that acts like a placeholder to map the value

In webservice you will have a url with the key i.e., called as path parameter

<some-path>/:parameterName/:parameterName

Here the webservice will have the above url’s with some parameterName with : at the beginning, this can accept dynamic data

ex: We have a webservice for a url : /employee/:id/:name

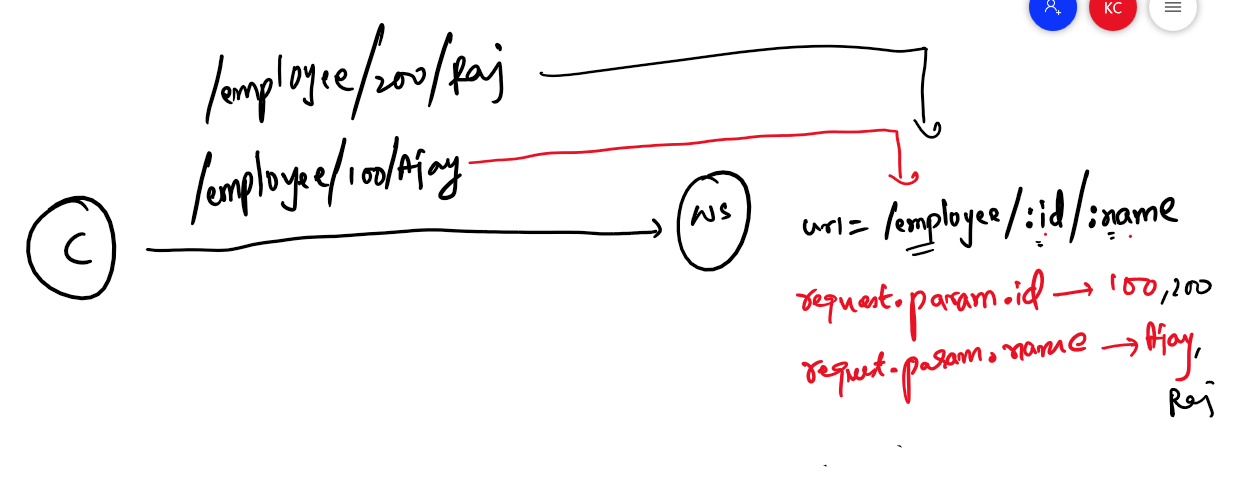
Here employee is the path & :id & :name are the parameters that can take any value, but employee should not change as it is static path, but :id/:name are dynamic path

What client & server must do, When the URL has the data as below

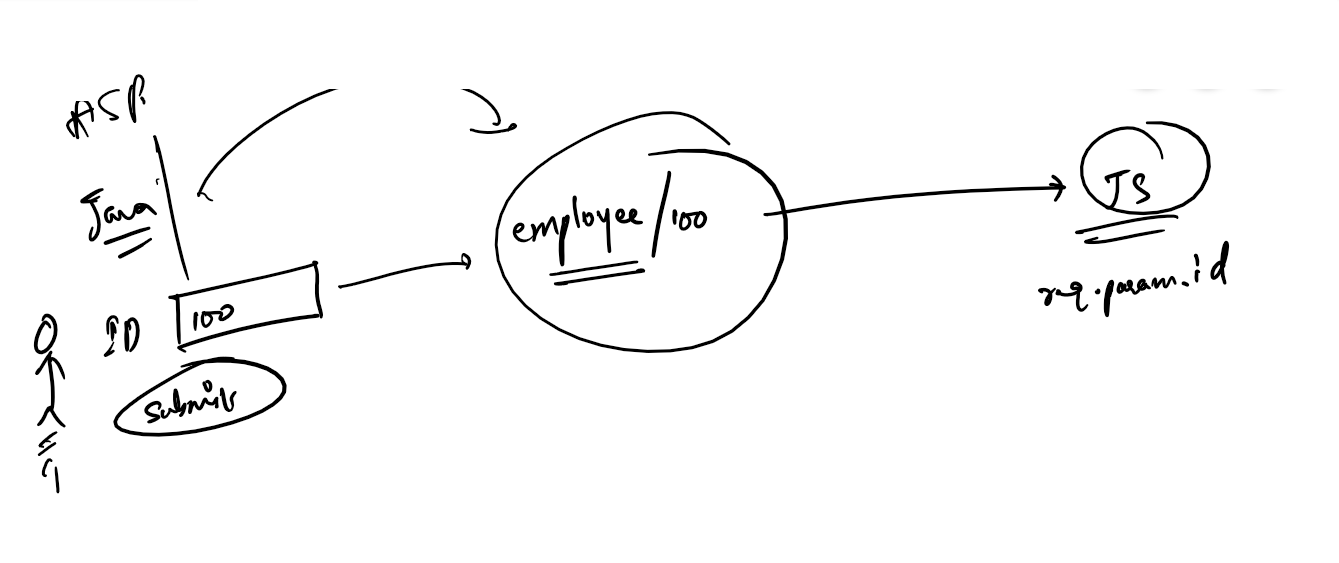
/employee/100/Ajay: It matches to /employee/:id/:name

/employee/200/Raj: It matches to /employee/:id/:name

Here :id & :name will be replaced with either 100/Ajay or 200/Raj, the webservice can extract the values using that id & name



Now the webservice can have a function that can handle the request coming to the /employee/:id/:name, where /employee/ is static and :id & :name are dynamic, they are read using request.param.parameterName



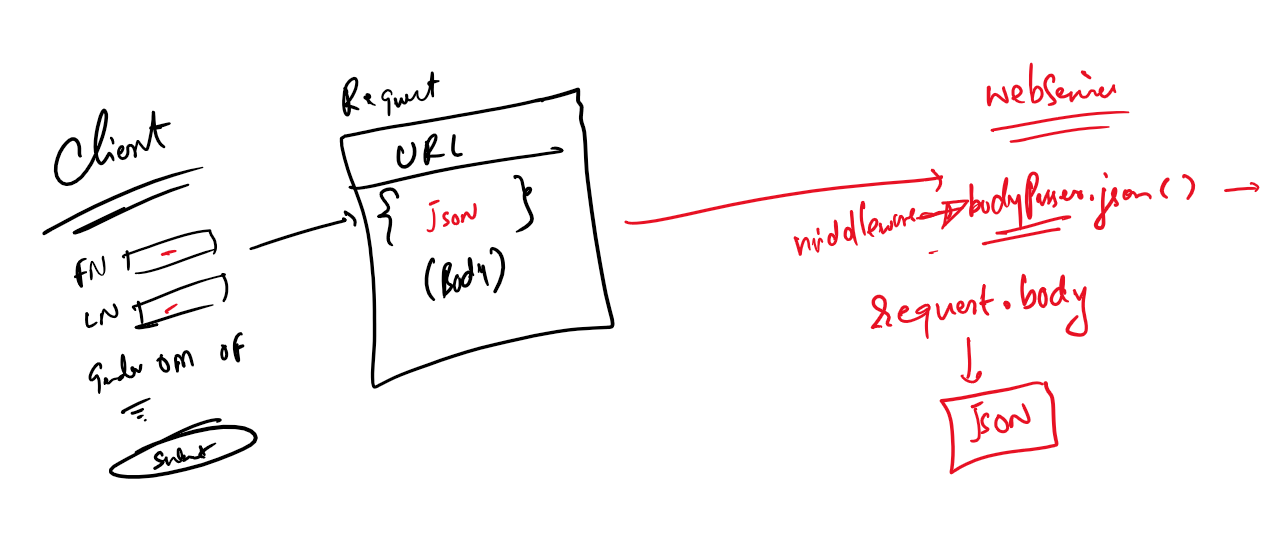
What Client and Server must do when the data is in the request body

When the client sends the data via request body, then the server/webservice must use bodyParser to extract the data,

Middleware Body Parser: It is a library that can read data from the request body, it could read json, xml, text data coming from the request body.

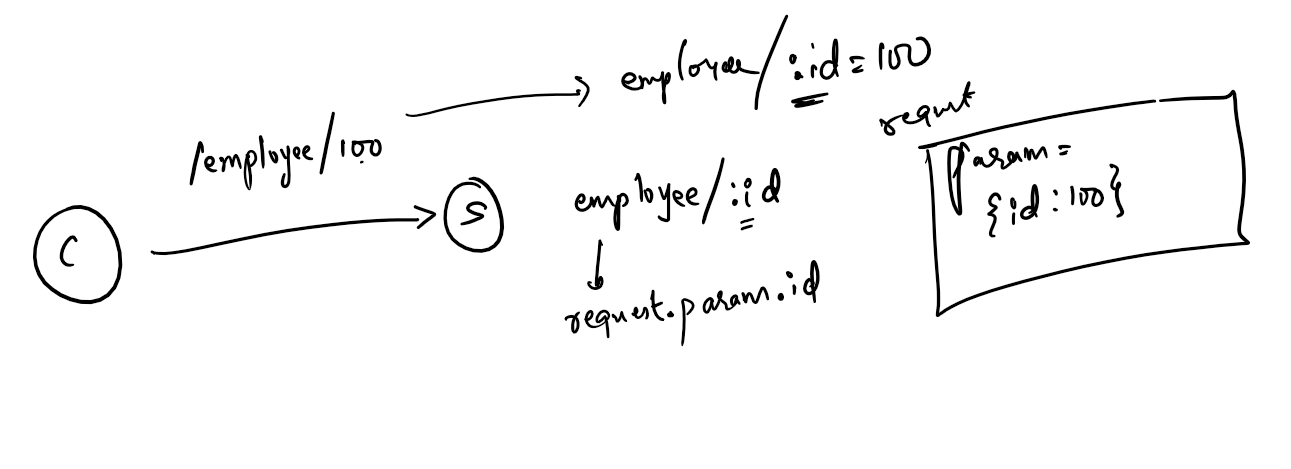
Middleware helps the server to intercept the request to extract the data, the body parser is a kind of middleware that intercepts the request to extract/parse the request body to convert the data to specific type when the data is read using request.body.

The request object has a property called request.body that can get the data once the body parser parses the data

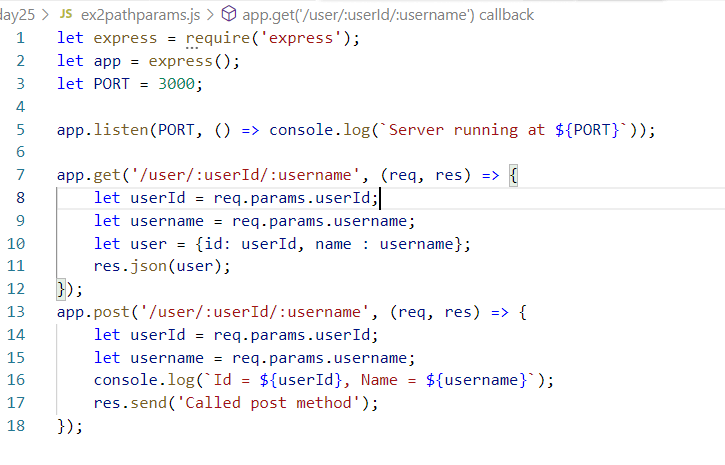


body parser is a library that you need to import using require(“body-parser”), from the reference of this module you can call the inbuilt functions to parse JSON data.

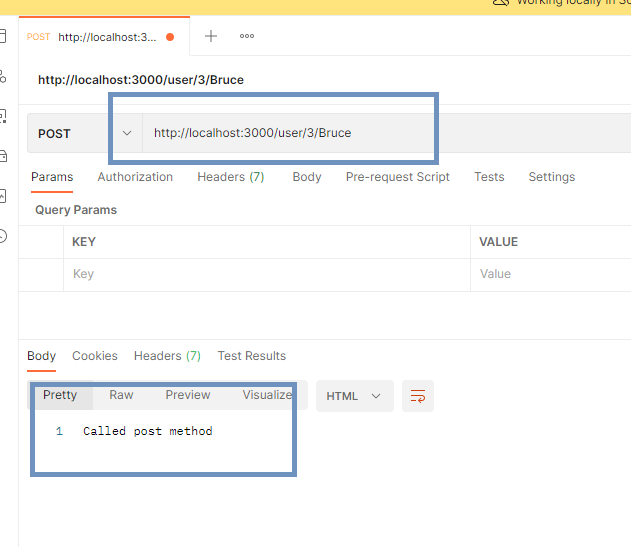
Sending the data from the URl and extracting the data via path parameters

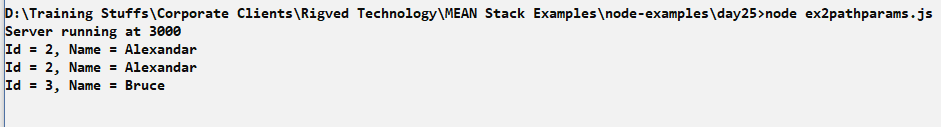


ex2pathparams.js

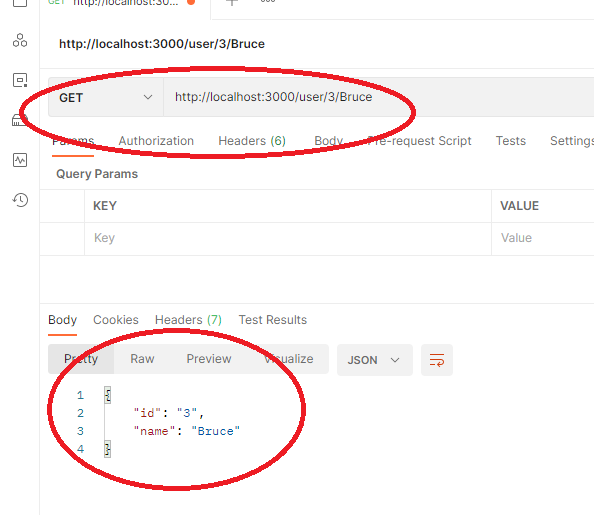


Output for post





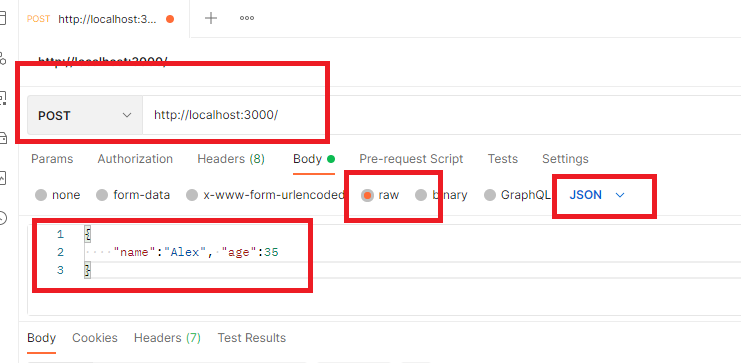
Output for GET



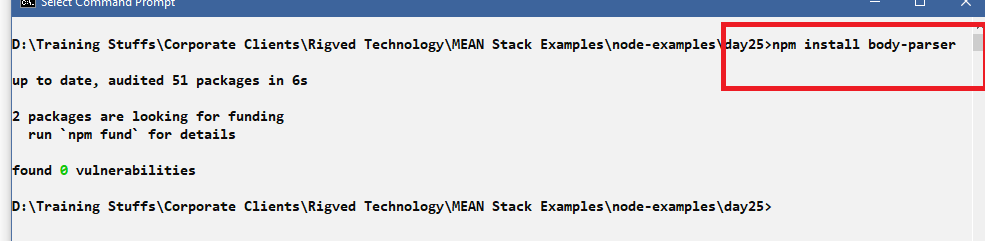
Reading the JSON data and extracting via request.body through body parser

Note: Client can’t send data in the request body in case HTTP.GET

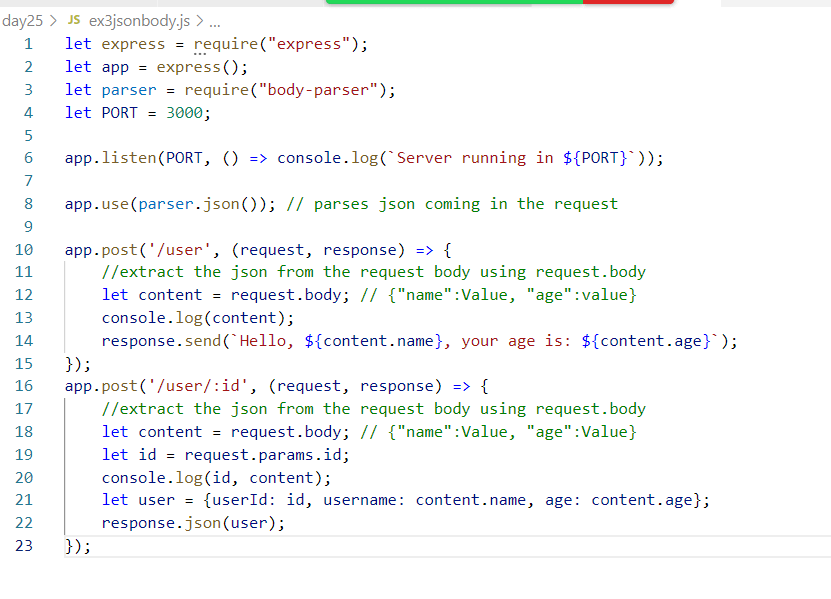
Note: In postman you can send the data in request body in the body tab of the request section



Install the body parser



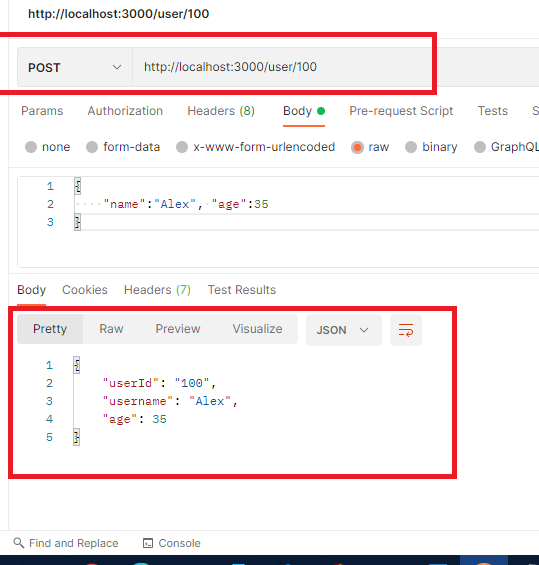
ex3jsonbody.js



Here app.use(parser.json()) will parse the data coming in the request object, if you omit it, then the request.body doesn’t extract json data

The app.post(‘/user/:id’, callback) is accepting the data from the URL and also from the request body, it returns the json data to the client.

Output:



Query Parameters

This is another way of sending the data in request via URL, where the data is going to have the key and value separated by &

/employee/?id=100&name=Alex

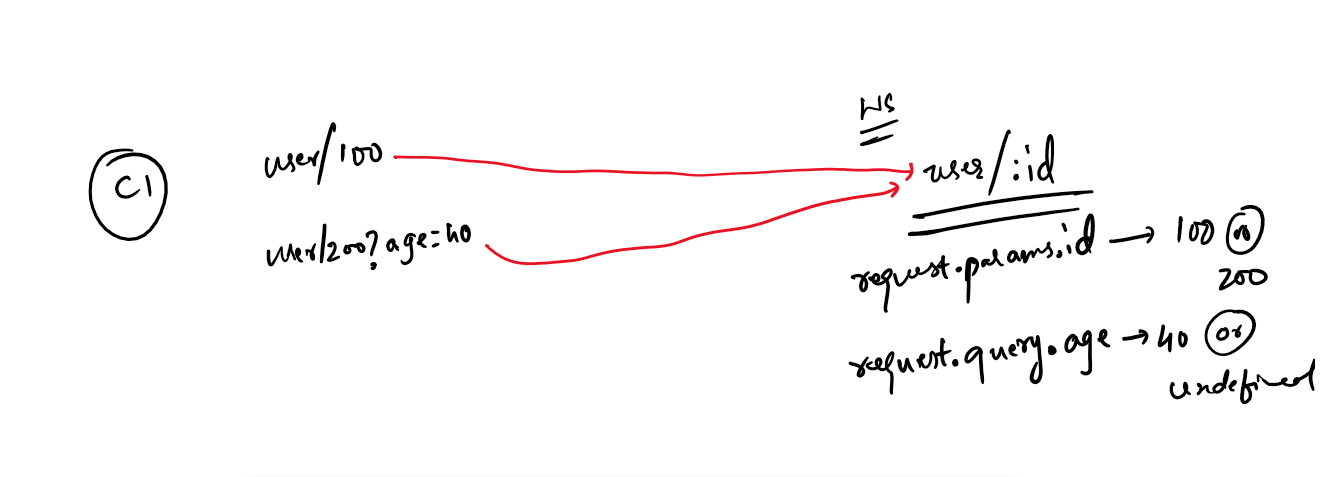
You can also use query parameters with path parameters, but query parameters must be at the end

i.e., /employee/100/?age=35

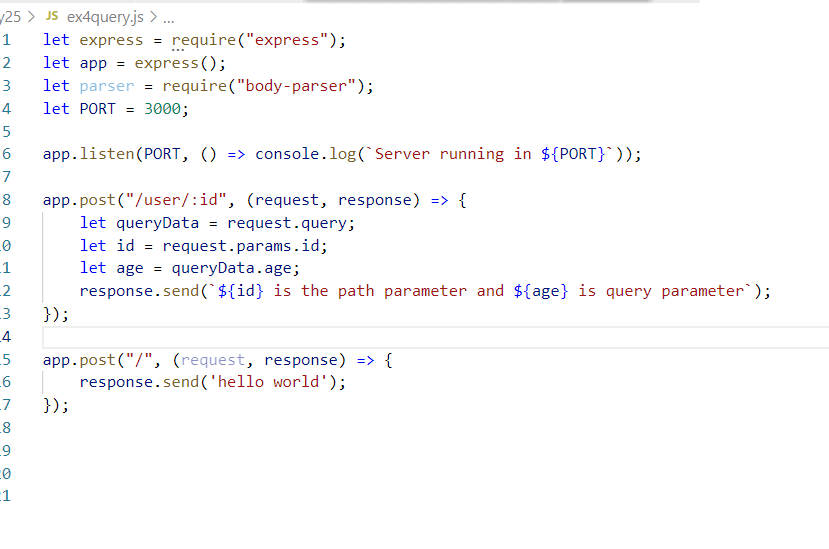
Here the path parameter is 100 & query parameter is age = 35

Path parameter vs Query parameter

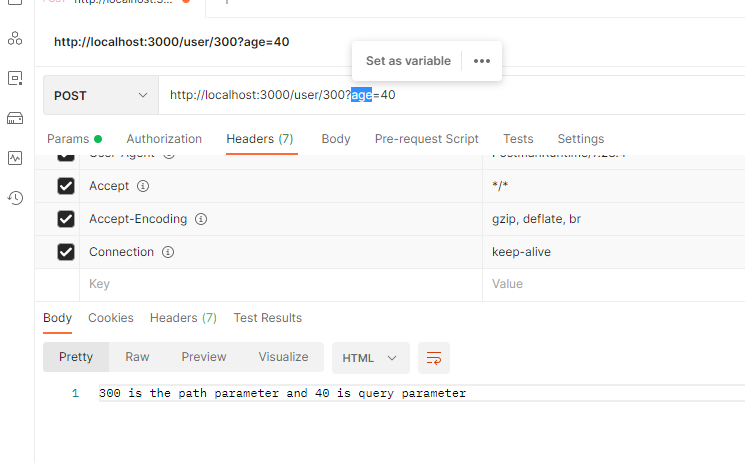
Path parameters are mandatory, it must be mapped to a webservice matching with the path, however query parameters are optional, there wouldn’t be any webservice with matching path, but they can read the query parameters



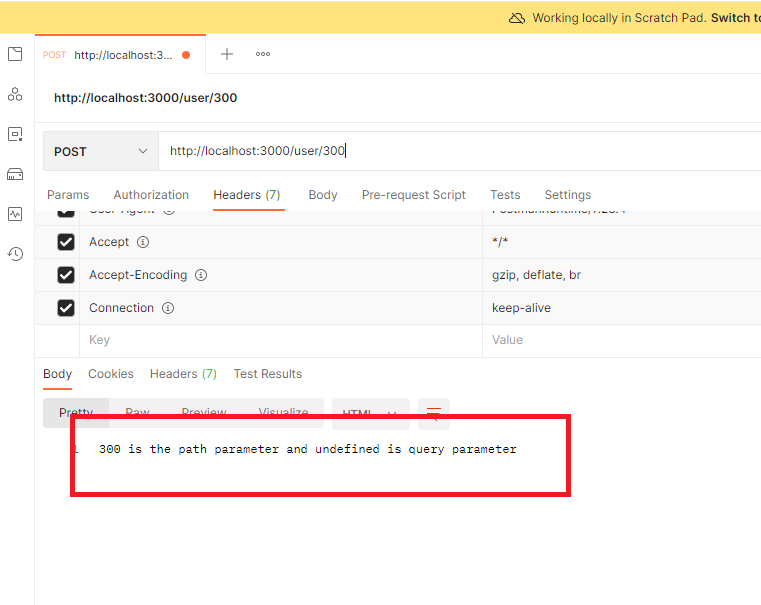
ex4query.js



Output:



If the age is not passed still the server can generate response, as query parameter is optional.



Activities:

1. Try all the above activities
2. Use GET, POST, PUT, DELETE methods of express module and perform CRUD operations in file system in a json file,
   1. store userid, name and age through post for the url /user
   2. show all the users present in the file in the postman for the url /user
   3. show a particular user based on the id for the url /user/:id
   4. delete a particular user based on the id for the url /user/:id

Note: Ensure the operations reflects in the file system, using arrays you should able to store, delete and retrieve

Interacting with database using node.js

Node.js provides modules to interact with any database like mysql, oracle, mongodb, and etc, if you want to interact with any databases then you need to have drivers

Drivers: These are the libraries written in specific language which allows the language to communicate with the specific database, drivers are just programs written for every programming languages by that these programming languages can easily interact with the database.

Since we are using Javascript to write programs we have database drivers that work with Javascript,

mongodb is a library that you can use to interact with the mongodb database

mysql is a library that you can use to interact with the mysql database

Note: Most of the javascript database drivers are named same as the database, these are node\_mdoules that you can download using npm

MongoDB is a NoSQL database which stores the documents in the database, these documents are JavaScript objects which will have \_id to uniquely recognize the document.

How to interact with MongoDB using JavaScript

We need to use mongodb, express, body-parser libraries optionally cors library.

mongodb: It is a node module that helps Java Script to interact with MongoDB

express: It provides inbuilt functions to perform CRUD operations through routes & helps creating webservices

body-parser: It helps parsing request body

cors: It helps real front end applications to interact with the backend webservices, by default every backend webservices denies front end applications to exchange the data, through cors you can specify what all the frontend applications can interact with the backend webservices

cors stands for Cross Origin Resource Sharing

Where to use cors at the backend

We need to use the cors in the middleware like we used body parser in the middleware.

How to use cors

let cors = require(“cors”);

app.use(cors());

Here the app is the express module function, the use is a middleware that enables cors, so that any frontend can send the request now, but we also can configure the cors() to accept the request only from the selected list of clients.

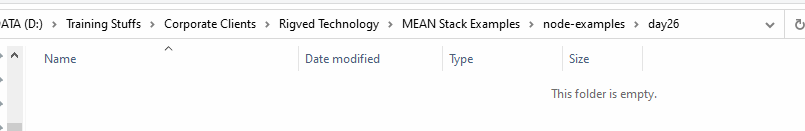
To interact with mongodb using node.js we need to install all the necessary libraries

* express
* mongodb
* body-parser
* cors

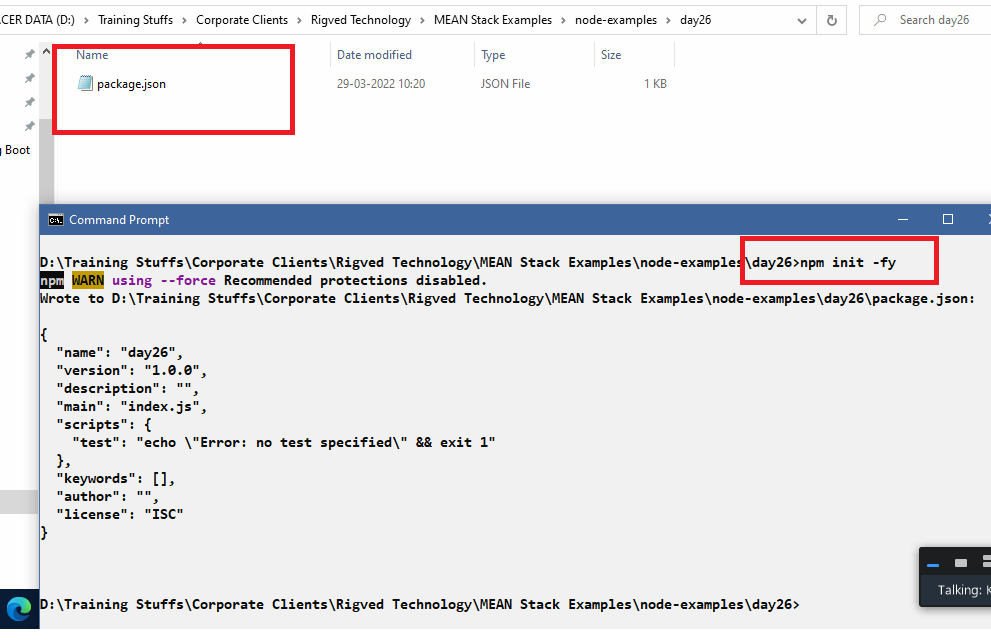
Firstly we can create package.json file if its not present using npm init -fy

Note: Even if package.json is not present it will be created when you use npm install

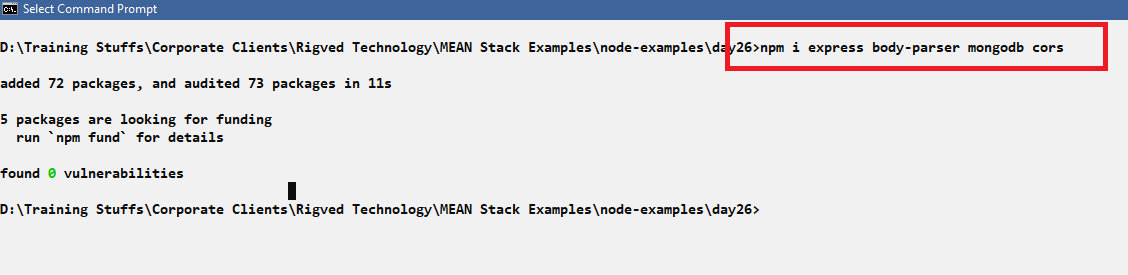
Currently day26 doesn’t have package.json



Using npm init -fy



Installing express, mongodb, cors, body-parser together



package.json looks like below

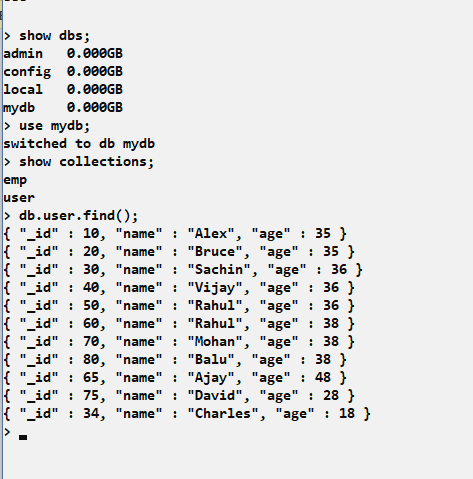


Note: mongodb mentioned in the package.json is just a library not a database, it helps javascript to interact with the mongodb

Let us use mongodb database

* ensure command prompt recognizes mongod & mongo commands
* start the mongodb using mongod
* you can use mongo command to interact directly with mongodb

Currently the mongodb has a ‘user’ collection in the database ‘mydb’



Now our Javascript program must interact with MongoDB, but it must know the location of mongodb database which is running in 27017 port & the IP address is localhost, hence the URL to connect to mongodb would be

mongodb://localhost:27017

How to connect to mongodb database.

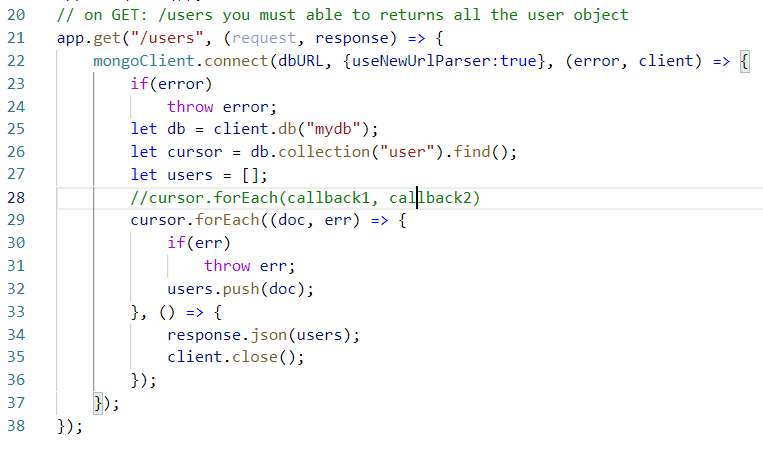
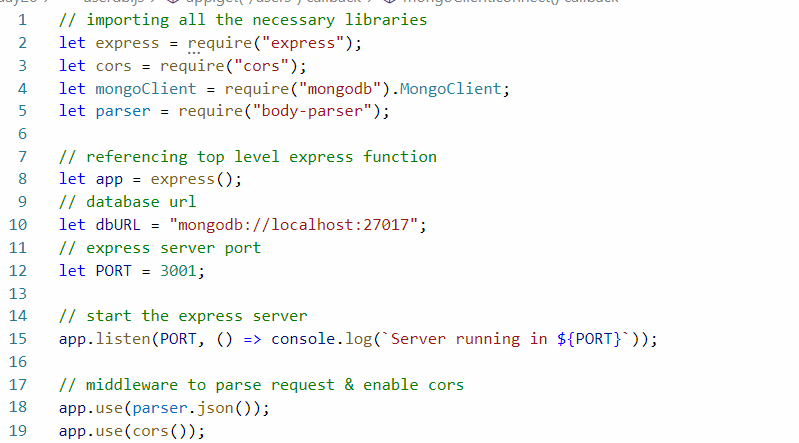
We need to use MongoClient class to connect to MongoDB

let mongoClient = require(“mongodb”).MongoClient;

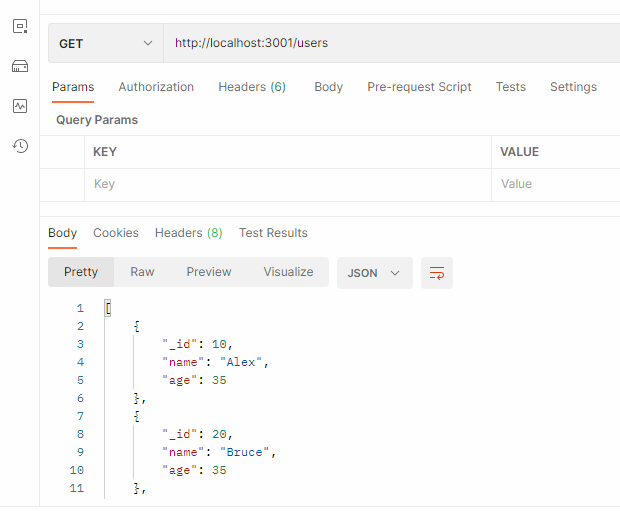
mongoClient.connect(dabaseURL, urlParser, callback);

Our first program to fetch all the records in the mongodb database

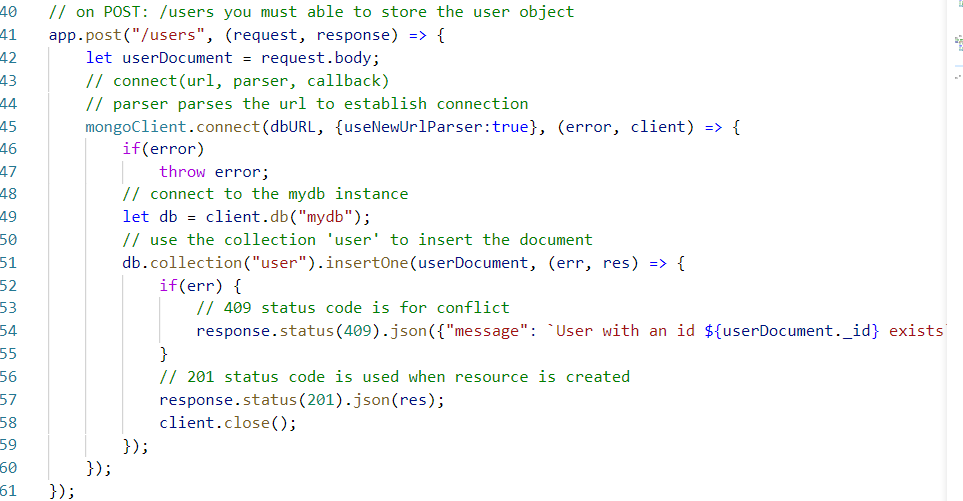
userdb.js



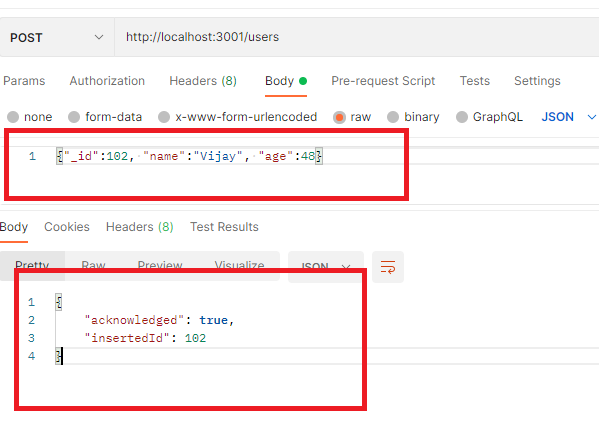
Output:



Storing the document in the mongodb

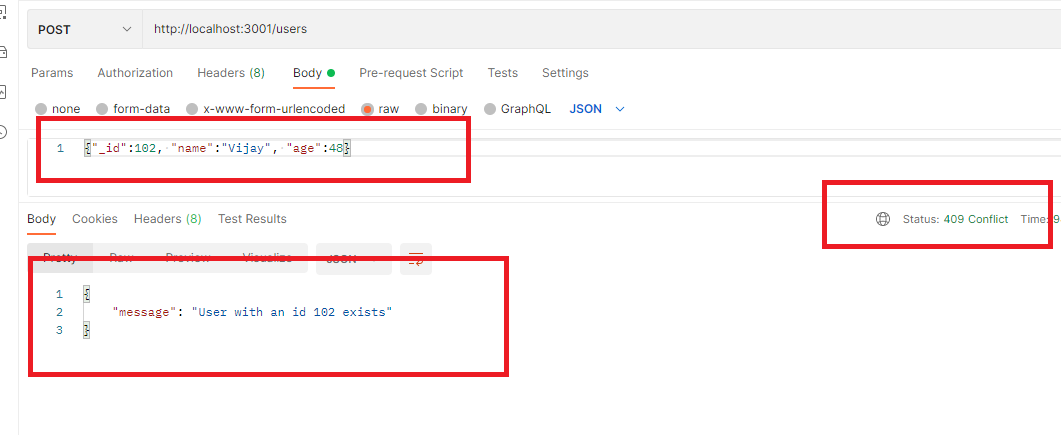


Output:



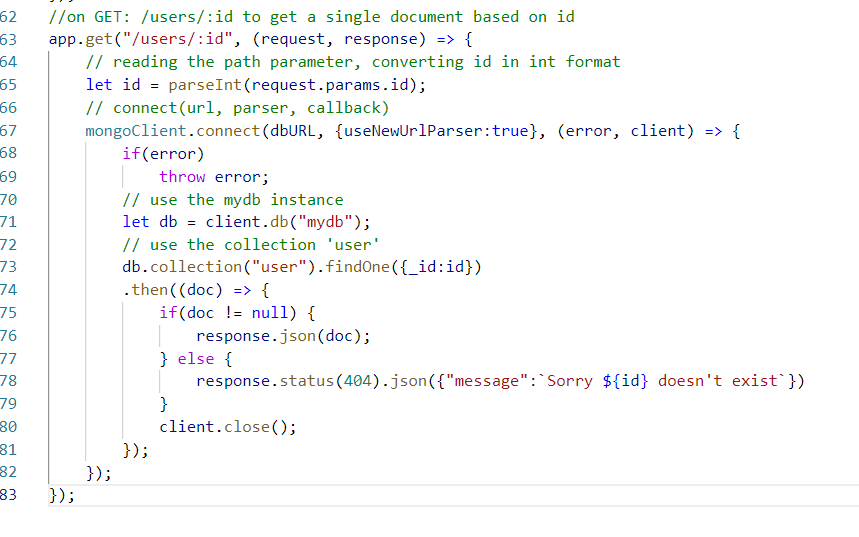
Note: the response we get is very useful at the front side, ex: it can read the insertedId 102 and show to the user 102 is stored, if in case it is auto-generated.

If the \_id is already existing then you get the error



Note: Here the response is customized in the insertOne() callback, but mongodb itself returns another response if the id already exists, which can also be used.

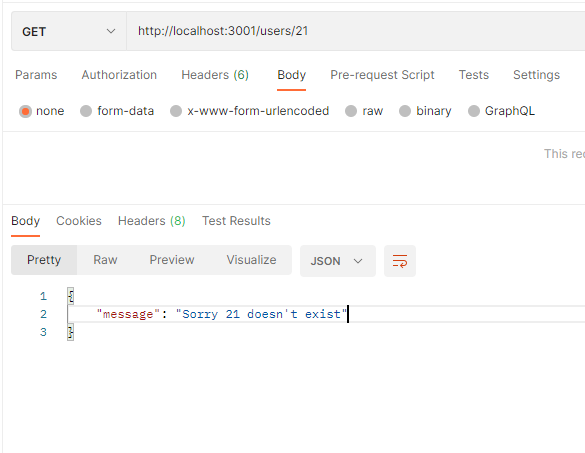
Getting a single document based on the \_id



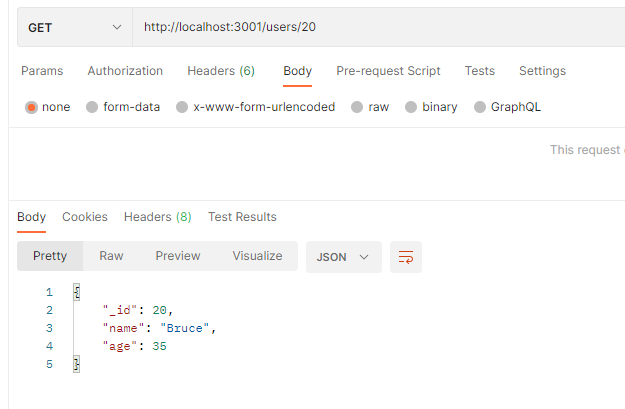
The findOne() returns the Promise object, which will have then() & catch() function the then() function is executed when the Promise is success, if there’s an error then catch() function is executed

Promise: It is mainly used to perform asynchronous task to avoid complexity in the callbacks.

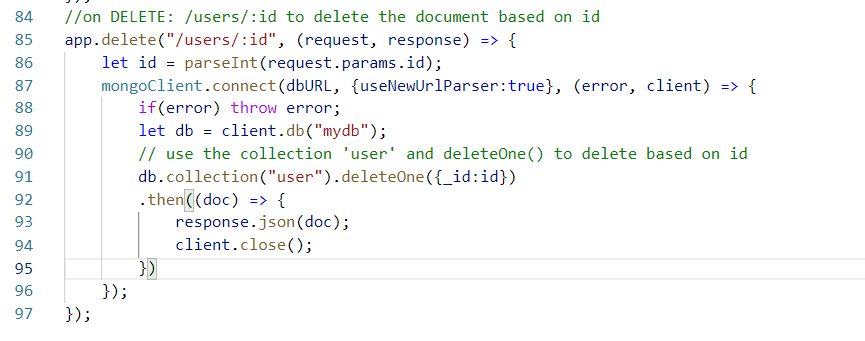
Output: if record doesn’t exist



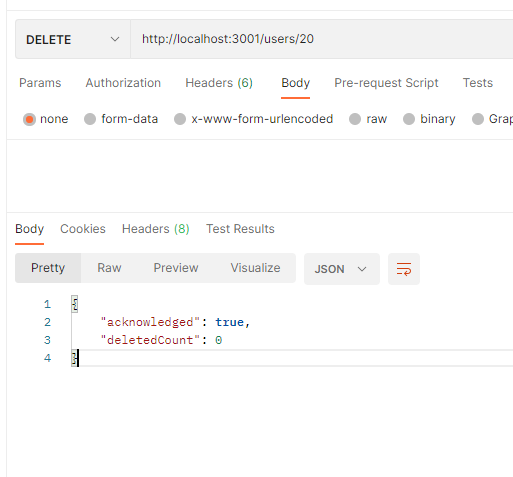
Output if record exists



Deleting the document based on id

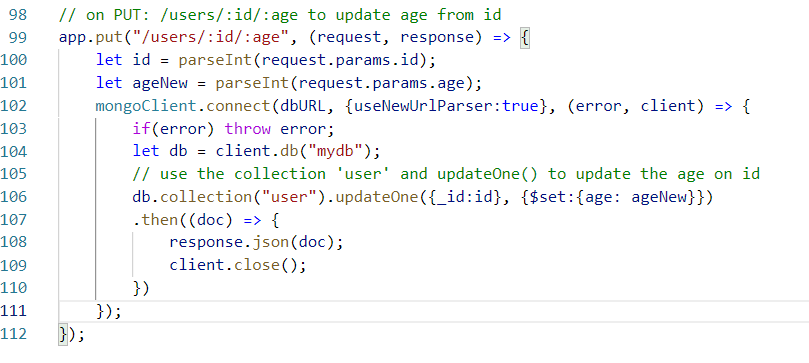


Output:

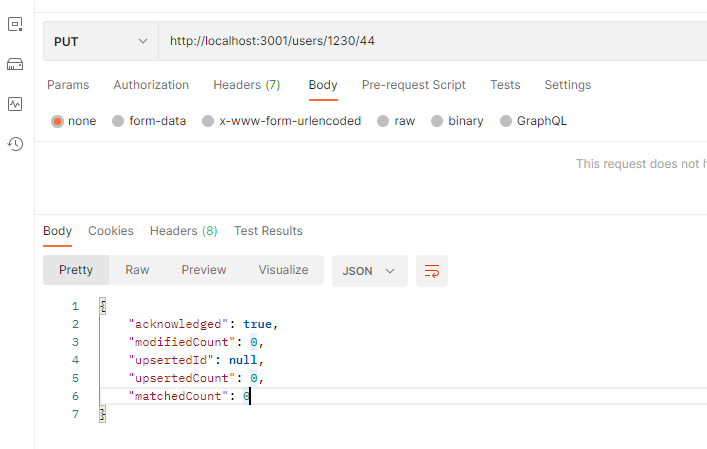


If you want to delete many documents, then you must use deleteMany({filter}); the filter must match multiple documents.

Updating the age in the document using id,



Output:



Activity:

1. Try all the above activities
2. Using node.js and mongodb perform crud operations on a document that has nested document, you need to perform storing the document, delete a document, getting all the documents, getting a single document, updating the document

The document must have following structure

\_id, name, salary, address.state, address.city, address.pin

ex: The above structure will look as below

{\_id: 1, name: “Raj”, salary:35000, address: {“state”:”KA”, “city”:”BLR”, “pin”:560001}}

1. Try to connect to mysql database and perform CRUD operations on a table employee having id, name & salary columns

Mongoose library: It is a library you can use to interact with MongoDB database, but it uses schema to interact with the documents, so that the documents should adhere to the schema rules, it is better to use mongoose when you want documents to have well defined structure, mongodb library in the other hand doesn’t follow any rules, it can work with the documents which is not having any structure & types at all

How to create a schema for the user document

You must have a reference to the mongoose module

let mongoose = require(‘mongoose’);

// defining the schema for the document

let userSchema = mongoose.Schema({\_id:Number, name: String, age: Number});

let UserDoc = mongoose.model(‘User, userSchema, user);

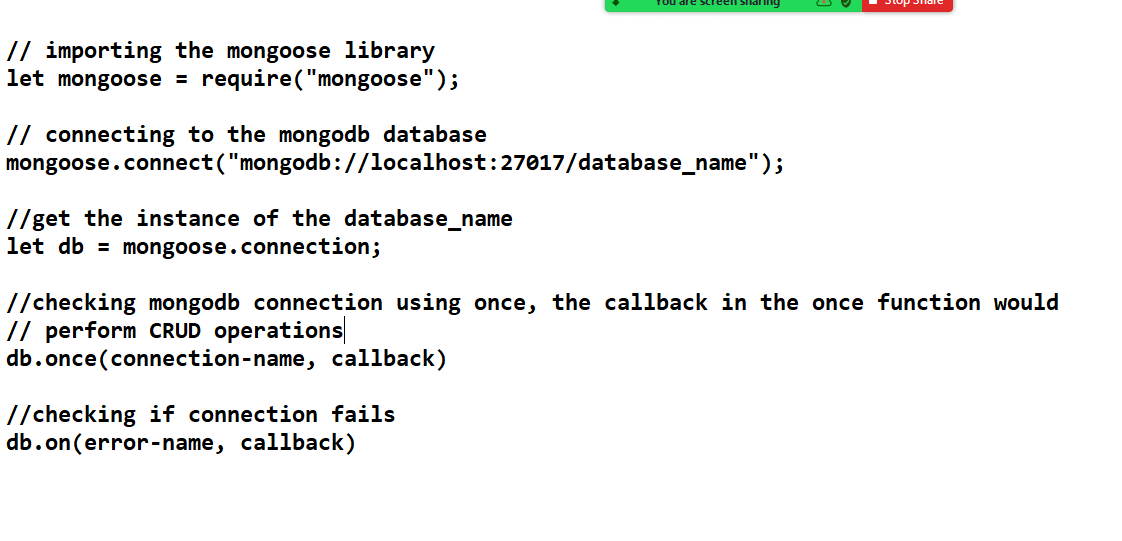
Here the UserDoc is used to create the document that follows userSchema rule, ‘User’ is just a name to the model, user in the 3rd argument is the collection where the document is stored

Now you can create the document for userDoc

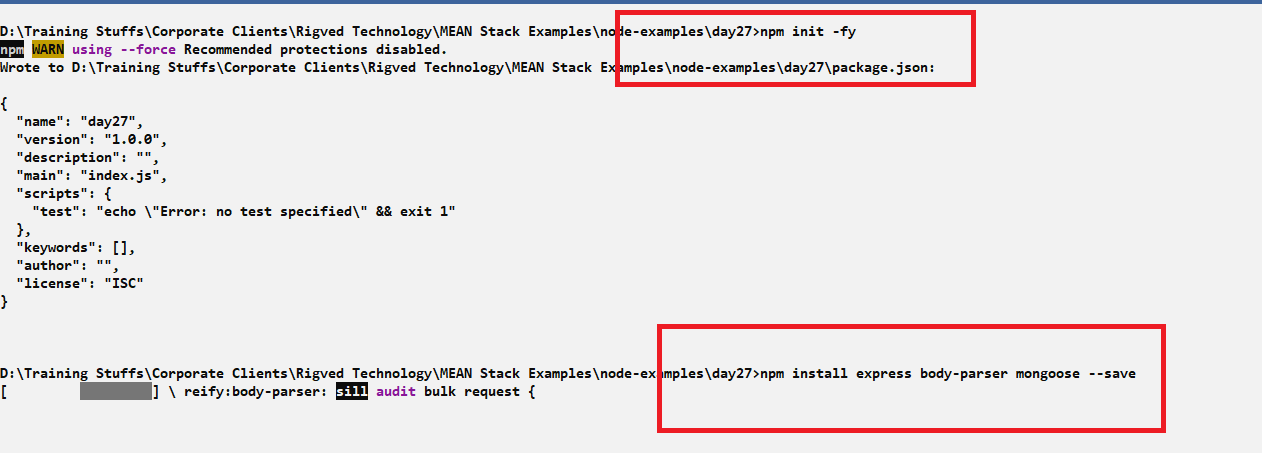
let user = new UserDoc({\_id:1, name:”Alex”, age:45});

user.save(callback); // stores the document

the callback is executed once the document is saved

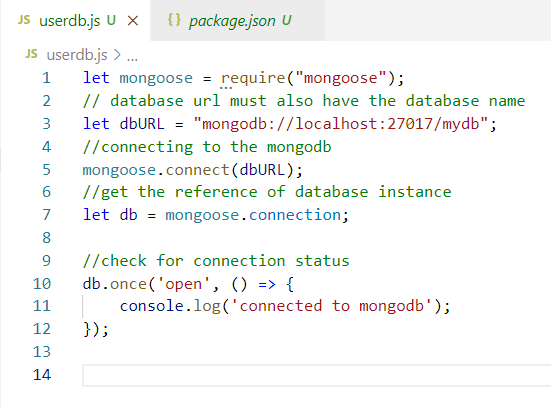


Installing mongoose express body-parser library

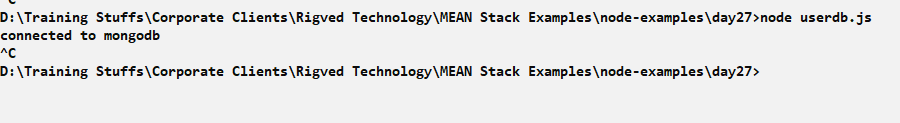


Note: --save it is to ensure that package.json doesn’t miss entries of the libraries you are installing, it is an optional argument, but recommended to use, because in some cases npm may not make entries of the libraries you install

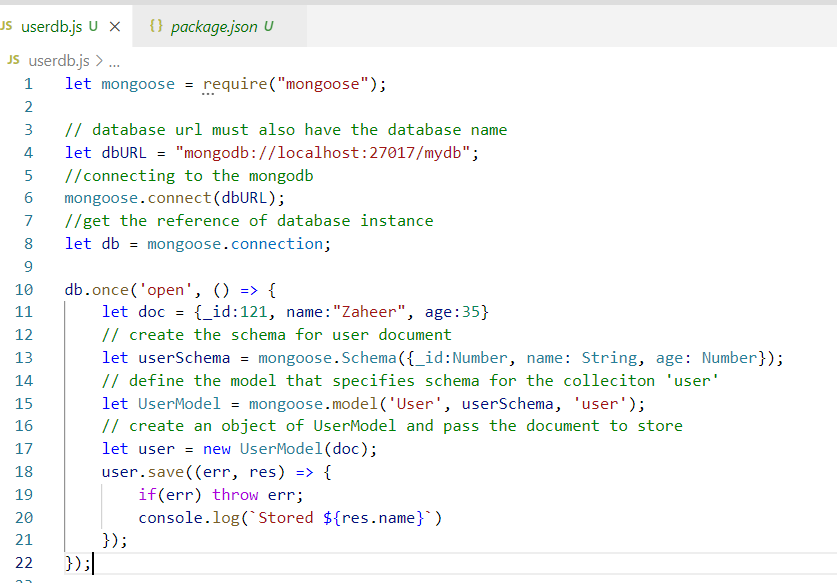
Checking the connection with mongodb



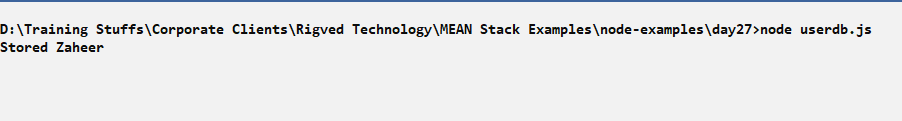
Output:



Defining the schema for the user document



Output:



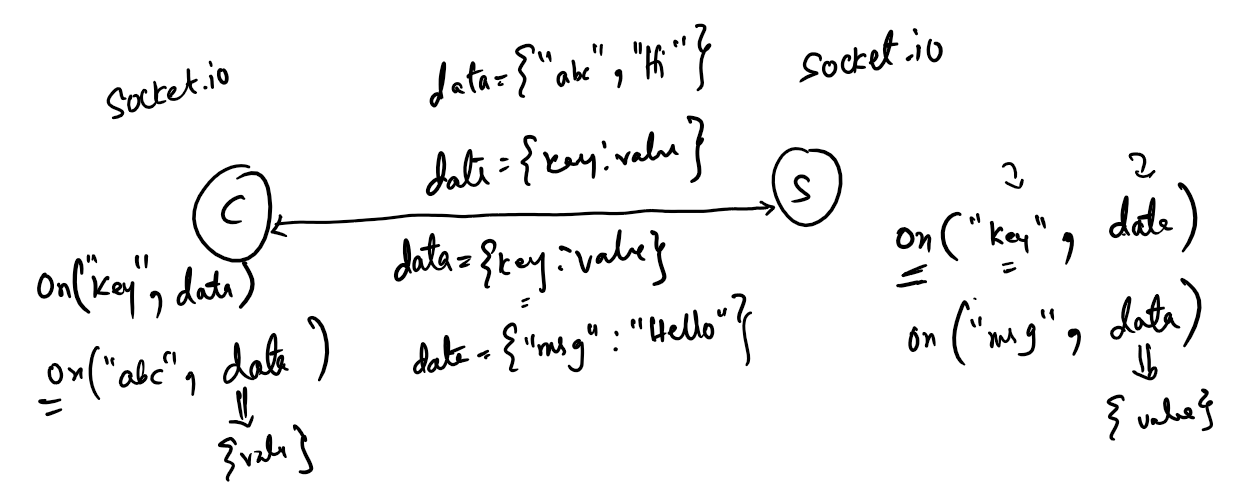
When you see the database you will see a new property in the document \_\_v which is like a version number to specify any changes in the document.

Socket IO

It allows client & server to exchange the data which are messages, they are bidirectional, it is useful in case of chatting programs where client & server communicates in the form messages

SockeIO is used by most of the messengers applications like whatsapp, online games, facebook messaging apps and so on

We need to install socket.io library when client & server wants to exchange messages.

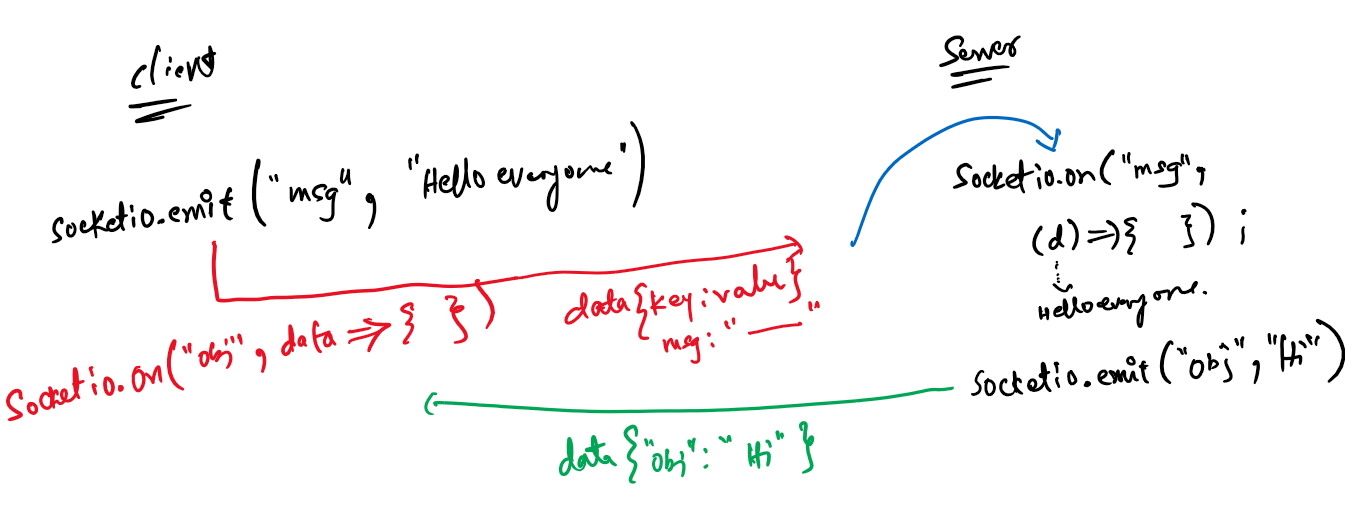


In SockeIO you don’t have to send the request & expect response for each data, you can send stream data to the server and server can receive those data without using any request object similarly server also can send stream of data to the client & client can also receive the data without using any response object.

SockeIO library needs to be used in both client & server side, it provides a function called on() which can be used to connect the client & server also to get the data, another function sockeio provides is emit() function that can send the data

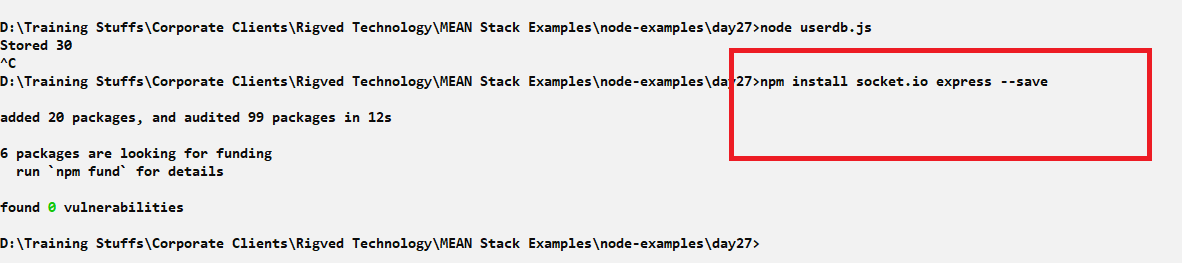
socketio.emit(key, data): This sends the data to the destination(server/client)

scoketio.on(key, callback): This receives the data that is emitted with the key that matches to the 1st argument

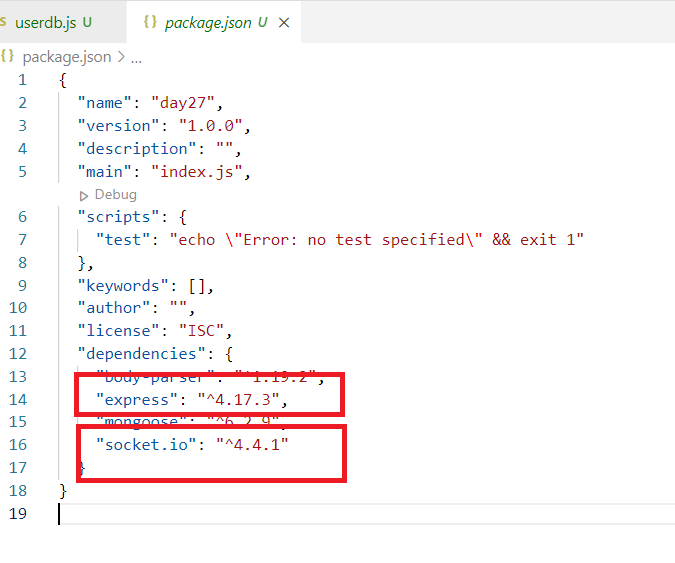


Note: SocketIO internally uses express & http module to create server and exchange the data

Installing socket.io & express



The package.json must have express & socke.io mandatorily



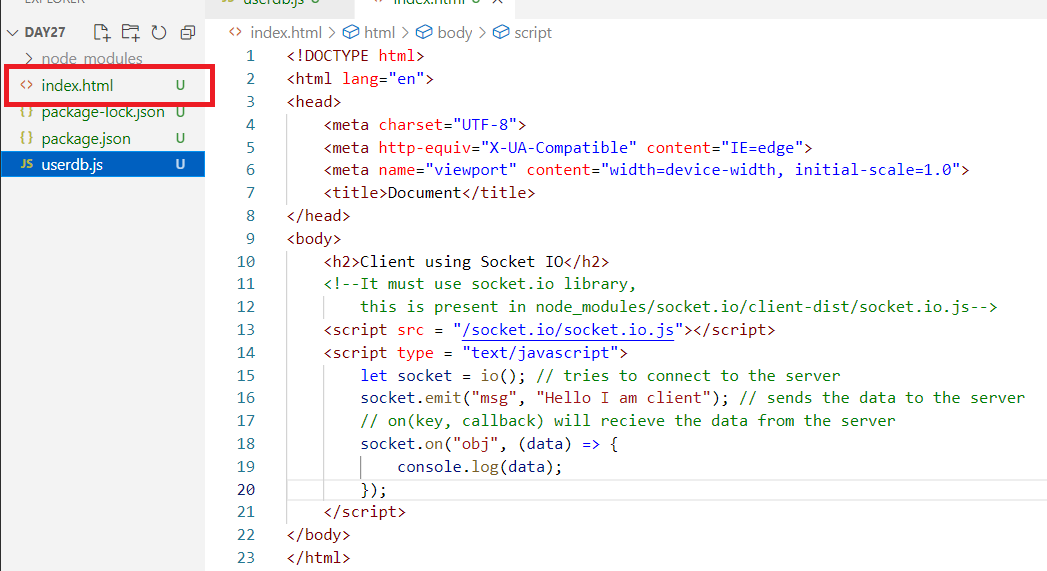
Here the client can be a browser and server can be a javascript program, we need to load the html file in the browser, this can be done using express response object sendFile function

response.sendFile(path, ‘filename.html’);

This code loads the html file by locating it in the path specified in the sendFile & shows the content on the browser

Create a html file that acts like client in the same directory where you will have server file

day27/index.html



This file loads the socket.io.js library to call io(), emit(), and on() function

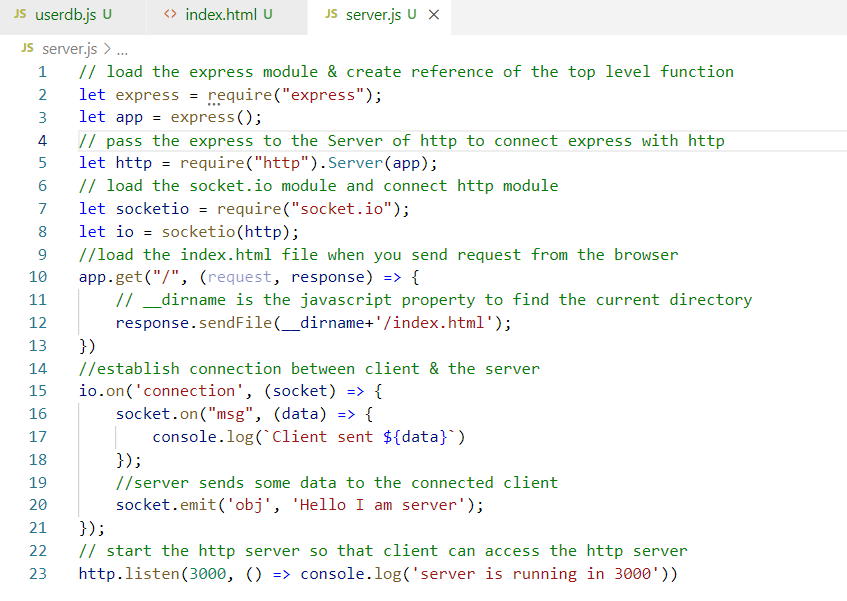
io(): It is used to connect to the server

emit(): it is used to send data to the destination currently its server

on(): it is used to receive data from the destination currently its server

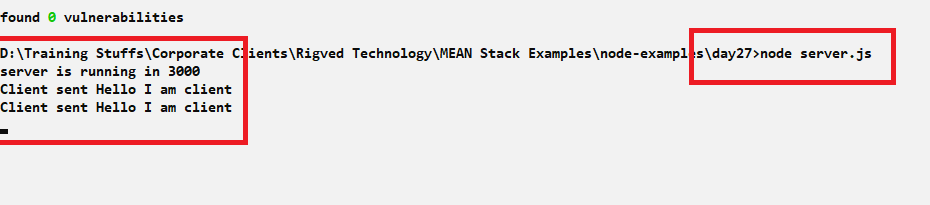
Now we need to create a server file that helps client to connect & can send & receive data to/from the client.

server.js

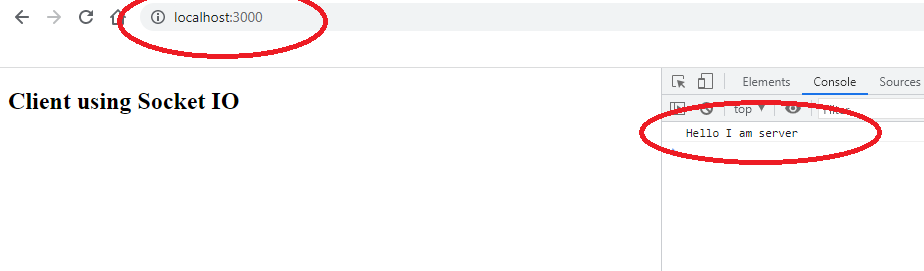


Once you run this file the server runs in 3000 port, the client i.e., browser can send request to this server which loads index.html at the client side, the index.html has a script which connects to this server & sends the data, once it connects the server would receive the data from the client & also sends data to the client, similarly client also sending & receiving the data from the server

Output:



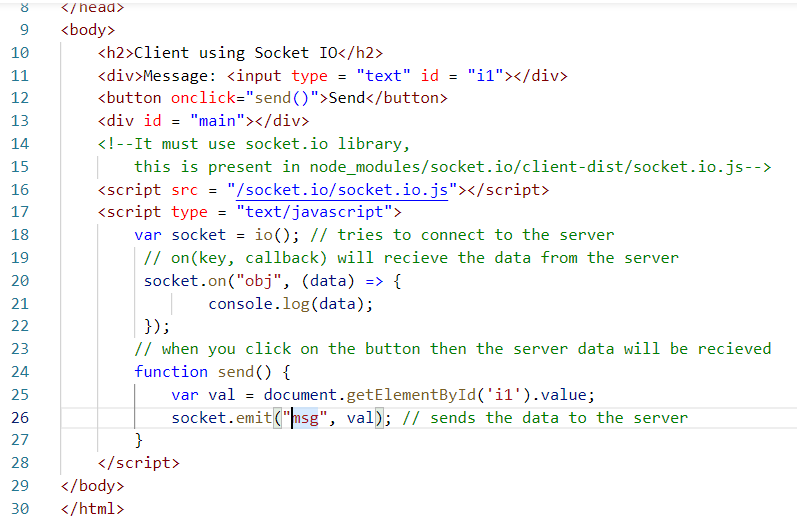
When you run the server.js you will first see server is running in 3000, but you will see client sent Hello I am client when you send request from the browser



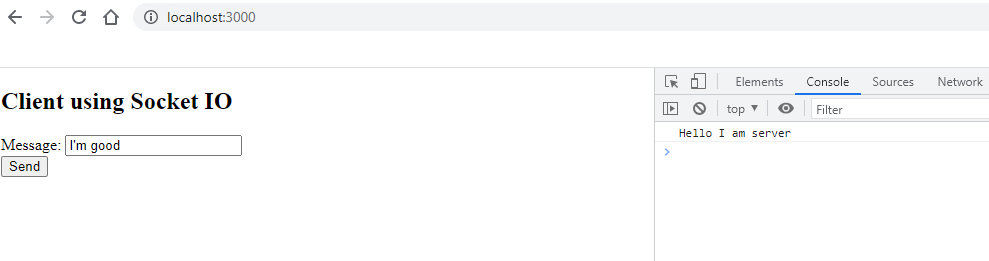
The browser shows the server data in the browser console as it receives the data the moment you send the url to <http://localhost:3000>

You can also handle javascript event and send the data to the server after clicking on any button, so that immediately the client doesn’t send the data

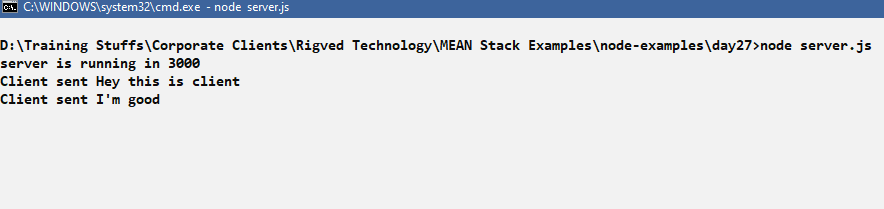
index.html



Output:

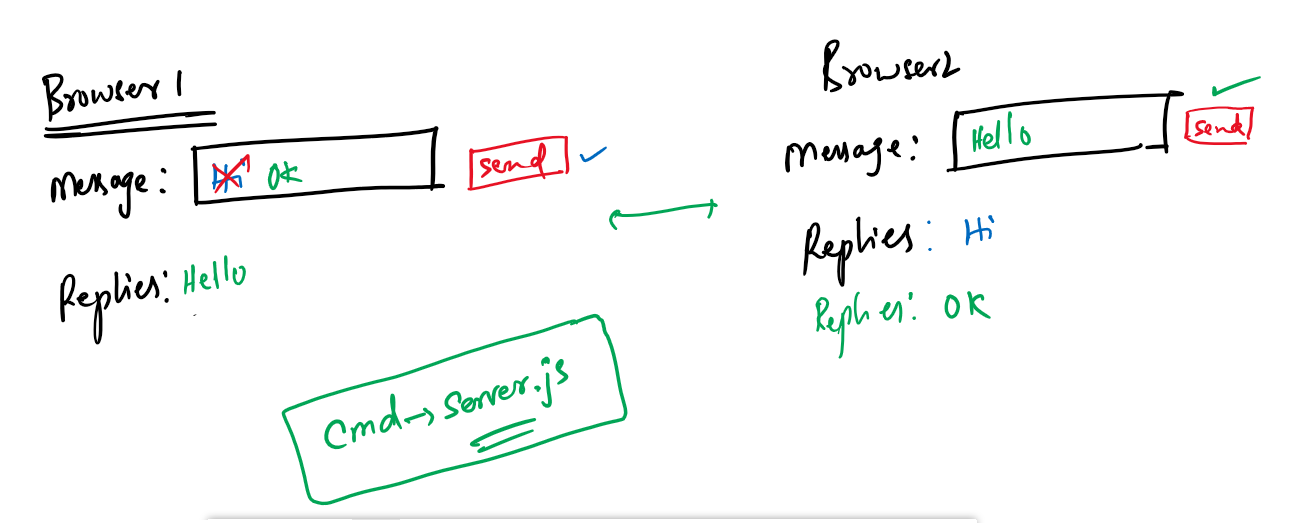


The server will display the client message



Activity

1. Try above activities
2. Create a chatting program where 2 browsers can exchange the data like 2 users chatting



Security:

Webservices are stateless it doesn’t recognize the request is sent by which client, hence the webservice uses token based authentication and authorization to find the users and their roles, It uses JWT JSON Web Token to generate the token the moment user enter credentials and store that token in the user machine, this JWT is sent along with the request each time so that webservice can identify the user & his role