# Core modules

Core modules means the features which is already present in the programming languages, example if core modules is like FS, buffer, http and varies other database connections.

Core modules is of 2 types

1. Non global : Which require importing
2. Global : Modules which does not require importing

Nodemon – for realtime refresh of the output

Chalk.colors – npm i colors : to color the console.

# Creating server:

const http = require('http');

http.createServer((req,resp)=>{

resp.writeHead(200,{'content-type':'application\json'});

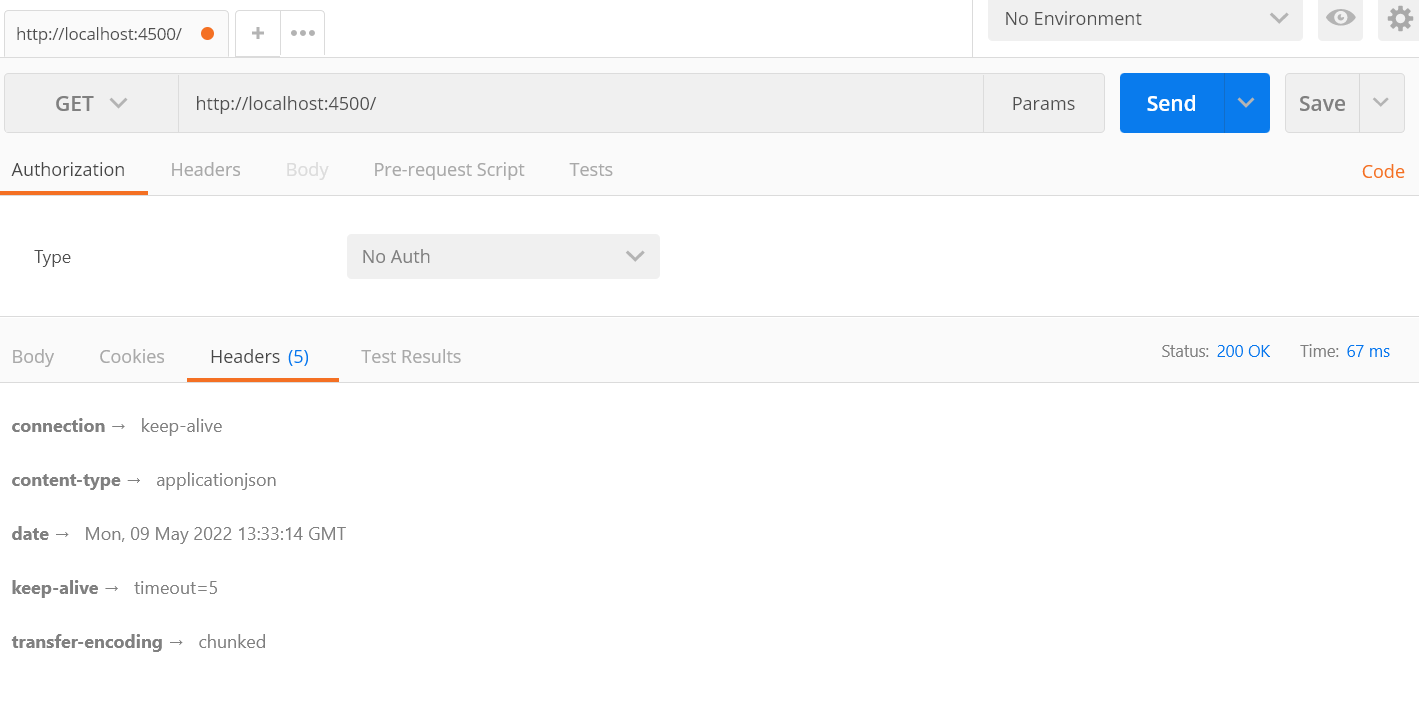
 resp.write(JSON.stringify({FirstName:'Ashutosh',LastName:'Sharma'}));

 resp.end();

}).listen(4500);

Here, write head defining which type of the document it is.

Example: In the below snippet of the postman, you would be able to see the output received from the head.



# Process node function

This function very useful to pass parameter from the console.

Create file using process

const fs = require('fs');

*// fs.readFile('test.txt.txt','utf8', (err, data) => {*

*//     console.log(err, data)*

*//   })*

const read = process.argv;

if(read[2]=="add"){

  fs.writeFileSync(read[3],read[4])

}else if(read[2]=="remove"){

  fs.unlinkSync(read[3])

}else{

  console.log("Invalid file")

}

Why array starts with [2], because first 2 array is reserved for system.

# ReadDIR of file system : FS system to read the files https://www.geeksforgeeks.org/node-js-fs-readdir-method/

In below example we are checking if file exist in the system by taking out the input from the user

To run this we write in console : Node Filename “ReadFile.js” , it run all files in a loop and then print the actual files.

const fs = require('fs');

*// fs.readFile('test.txt.txt','utf8', (err, data) => {*

*//     console.log(err, data)*

*//   })*

const UserInput = process.argv; *//it takes userinput from console.*

const path = require('path');

const dir = path.join(\_\_dirname);

console.log(dir) *//Printing dir*

console.log(UserInput[2])*//Reading user input*

fs.readdir(dir,(err,files)=>{

  if(err){

    console.log(err);

  }else{

    console.log(files);

    console.log(files.length)

  }

  for (i=0; i<files.length;i++){

const ipos = files[i].indexOf(UserInput[2]);

if(ipos==0){

console.log("File.js exist in the system");

  }}

});

# Curd operation with File System : Curd means create, update, read and delete, you can read this in detail from website.

# setTimeOut

This function delays the code execution.

setTimeout(() => {

    console.log("THis will be delayed");

}, 2000);

# Promise and then

Promise and then used to hold the execution until wait is over.

Example : let WaitData = new Promise((resolve,reject)=>{

setTimeout(() => {

    resolve("I'm working fine");

}, 2000);

})

WaitData.then(()=>{

    console.log(WaitData)

})

.then method is used id resolve parameters works.

# Express.js

Express.js is used to create API using get method, it is same like creating server and loading HTML

Example of express.js

const express = require('express');

const app = express();

app.get('',(req,resp)=>{

    resp.send("welcome to homepage")

});

app.get('/about',(req,resp)=>{

    resp.send("welcome to about page")

});

app.listen(4600);

# render HTML file in Express.js

const express = require('express'); *//initiating express.js*

const path = require('path'); *//Initiating path*

const app = express(); *//creating executable file of express so that its functions can be used*

const FilePath = path.join(\_\_dirname); *// File path will fold the dir name*

app.use(express.static(FilePath));*//express.statis will load only static HTML files, app.use is express funvtion to render*

console.log(\_\_dirname);

app.listen(4500);*//listen to port for an output*

In default port, index.html will open

# Creating Alias using node.js

In order to create alias we will not user app.send or app.use(express.static(FilePath)); instead we will use app.sendFile with accurate path.

Example:

app.get('/homepage',(req,resp)=>{

    resp.sendFile(`${\_\_dirname}/index.html`);

})

With above code, if user will open localhost/homepage it will open index.html

Complete codes

const express = require('express'); *//initiating express.js*

const path = require('path'); *//Initiating path*

const app = express(); *//creating executable file of express so that its functions can be used*

const FilePath = path.join(\_\_dirname); *// File path will fold the dir name*

app.use(express.static(FilePath));*//express.statis will load only static HTML files, app.use is express funvtion to render*

console.log(\_\_dirname);

app.listen(4500);*//listen to port for an output*

console.log(`${\_\_dirname}/index.html`)

app.get('/homepage',(req,resp)=>{

    resp.sendFile(`${\_\_dirname}/index.html`);*//This will open index.html with "homepage" alias*

})

# Creating 404 page

To create 404 page, use \* in the get method. Example

app.get('\*',(req,resp)=>{

    resp.sendFile(`${\_\_dirname}/error.html`);

})

With above code if we open wrong page the it will open error.html

# What is template engine in node.js

1. To add dynamic webpage using node.js, ejs needs to be installed, “npn I ejs -g”
2. Create a folder with neame “view, it has to be view always
3. Create server using express.js , add below code of ejs
4. app.set('view engine','ejs');

Then instead of using resp.send/resp.sendFile, use resp.render and then filename (Without path)

app.get('/profile',(req,resp)=>{

*// resp.send("welcome to about page")*

    resp.render('profile');

});

Full code :

const express = require('express');

const path = require('path');

const app = express();

app.get('',(req,resp)=>{

    resp.send("welcome to homepage")

});

app.set('view engine','ejs');

app.get('/profile',(req,resp)=>{

*// resp.send("welcome to about page")*

    resp.render('profile');

});

app.listen(4600);

# Difference between printing text, static page and Dynamic page

|  |  |  |
| --- | --- | --- |
| Text | Static Page | Dynamic page |
| Resp.send | Resp.sendFile() | Resp.render() |

# How to pass variable from node to HTML.

TO pass variable from node to HTML, in CJS, we can use this syntax - <% Variable %>

Make sure to include variable under render arguments.

Example : resp.render('profile',{name}); //Here name is the argument which need to pass.

Example :

const express = require('express');

const path = require('path');

const app = express();

app.get('',(req,resp)=>{

    resp.send("welcome to homepage")

});

app.set('view engine','ejs');

app.get('/profile',(req,resp)=>{

*// resp.send("welcome to about page")*

    const user = {

        name : 'Ashutosh',

        Age : '30',

        work : 'developer'

    }

    resp.render('profile',{user});

});

app.listen(4600);

HTML :

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Profile Page</title>

</head>

<body>

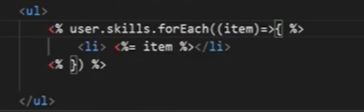
    <h1>This is the headline if the dynamic Page <%= user.name %>&nbsp;<%= user.work %></h1>

</body>

</html>

Use case:

What if there are multiple values which needs to be printed using some loop. Then can use this using same syntax however rendering syntax will change.



# How to use common templates like header and footer in the node.js

Its is very simple, you just need to create ejs template and call it using below syntax.

   <%- include('common/header')%>

# What is middleware

Middleware is act as a filter in server, example if there who doesn’t have proper token in URL then page will not be accessible to him (Age confirmation, country check, etc)

**Syntax:**

We can simply create a function where we will create routes then we will check condition, here next() will work if condition doesn’t match

const express = require('express');

const app = express();

const reqFilter =(req,resp,next)=>{

    if(!req.query.age){ *//checking if parameter is blank*

        resp.send('Query should not be blank'); *//This will be printed*

    }

    else if(req.query.age<20){

        resp.send('You cant access this page'); *//This will be printed if age is less than 20*

    }

    else{

        next(); *//if nothing works, it will pass to next*

    }

}

app.use (reqFilter); *//Calling our custom function here*

app.get('/',(req,resp)=>{

resp.send('Welcome to homespage');

})

app.listen(4500);

What if condition should only be applicable for only one route not all routes

In this case, we need to update the parameters and remove app.us(reqFilter) from the code.

Updated code of app.get

app.get('/',reqFilter,(req,resp)=>{

resp.send('Welcome to homespage');

})

# Mongo DB, creating and establishing connection

const {MongoClient} = require('mongodb') *//creating mongo instance syntac should be exact*

const url = 'mongodb://localhost:27017'; *// your data base link*

const database = 'Sample' *// your database name*

const client = new MongoClient(url); *//creating client object*

async function getData()

{

    let result = await client.connect(); *//creating connection to DB*

    let db = result.db(database); *// creating DB object*

    let collection = db.collection('learning'); *//creating collection form DB*

    let response = await (collection.find({}).toArray()); *// getting collection data, await is for promise*

*//collection.find is like SQL query which filter result and provide data*

    console.log(response); *// printing data*

}

getData(); *//executing function*

# Mongo DB different connection file.

We can save our connection in different file to save time

Connection file

const {MongoClient} = require('mongodb') *//creating mongo instance syntac should be exact*

const url = 'mongodb://localhost:27017'; *// your data base link*

const database = 'Sample' *// your database name*

const client = new MongoClient(url); *//creating client object*

async function getData()

{

    let result = await client.connect(); *//creating connection to DB*

    let db = result.db(database); *// creating DB object*

    let collection = db.collection('learning');

    return collection; *//creating collection form DB*

}

module.exports =  getData;

Now we can export getData function which have objects.

const getData = require('./connection'); *//importing connections*

async function main(){ *//async to handle promise*

    let data = await getData(); *// await to handle promis, saving objects in data*

    data = await data.find().toArray();

    console.log(data);

}

main();

s