JavaScript, HTML, CSS etc run on the client machine (front end)

Node JS run on the server (back end).

Chrome browser executes JavaScript code using the built-in V8 engine.

Node JS is a run-time environment which runs JavaScript code using V8 engine.

Node JS developers create APIs for web, android, IOS etc.

Download and install Node js. NPM gets installed by default.

Run the code (node -v) from cmd to check node version

Run the code (npm -v) from cmd to check npm version

Install VSCode editor.

To run node from command line, simply type node. (Rarely used, since it cannot be saved)

After executing every command we get the result and the ‘undefined’. We get this ‘undefined’ in browser console as well.

This is because the function does not return anything.

Run node using vscode, create a folder, open cmd from folder, write code .

It will open vscode in the current folder.

Create file named ‘index.js’. Write something to console.log/console.warn

Open Terminal, type node .\index.js, press enter.

Interview Question

Is console.log in Node JS and browser are the same?

No. In browser, console.log is a built in function. Whereas in nodeJs, console is a module which provides similar mechanism provided by web browsers.

**JS Fundamentals for Node js**

How to get data in one file in another file. In javascript we use ‘export and import’ where as in node js we use ‘require’

Source file = test.js

module.exports = {

    x:10,

    y:20,

    z:function(){

        return this.x+this.y

    }

};

Destination file

const test = require('./test');

console.log(test);

console.log(test.z());

Output

{ x: 10, y: 20, z: [Function: z] }

30

Interview Question: What is filter function?

Ans: filter() function is used to filter an array. It takes a call-back function as argument, which will be called on every element of the array. It returns an array.

Core modules – Built-in modules (console.log(), fs, Buffer, HTTP etc.)

* Global modules – Modules such as console which are not required to be imported.
* Non-global modules – modules such as fs to be imported

Eg.

const fs = require('fs');

fs.writeFileSync('Hello.txt', "Hello World");

a file named Hello.text with contents Hello World is created.

**Interview Point**

Instead of importing all the functions in the fs module, we can import only the required functions like,

const fs = require('fs').writeFileSync;

fs("Hi.txt", "This time only the writeFileSync function is imported");

const createFile = require('fs').writeFileSync;

createFile("abc.txt", "Not necessarily be fs");

To print directory name and file name

console.log(\_\_dirname);

console.log(\_\_filename);

Creating a basic server

Function as parameter

Arrow functions

Get output on browser

To create a server, we need to **import the http module which handles the request and response.**

const http = require ('http');

http.createServer((request, response)=>{

    response.write("Hello World! I have created a basic server");

    response.end();

}).listen(4500);

Now open the browser and type http://localhost:4500

In the above example an arrow function is passed as argument to createServer function. See the below code

const http = require ('http');

function myFunction(request, response){

    response.write("<H1>Arrow function replaced with normal function</H1>");

    response.end();

}

http.createServer(myFunction).listen(4500);

here myFunction/the arrow function is known as call back function because whenever createServer function is called, myFunction will also be called.

**All about package.json – never delete package.json**

How to create package.json? **npm init**

Install the required packages

– npm i colors (notice the dependencies added in the package.json file). It can be seen that a folder named node\_modules is created and under it a sub-folder named colors is also created.

**Interview point**

Is node JS single threaded or multithreaded?

Single threaded. (only one thread executing code line by line)

What happens if node\_modules file is deleted?

Run npm install from terminal, node\_modules folder will be back. Package.json file contains the dependencies, so the folders will be created accordingly.

**Correct way to push data on git?**

Since node\_modules folder is heavy, it should not be pushed into the git repository. Other developers who wants to work on the project can get the node\_modules folder by running npm install (package.json file should be there).

To avoid node\_modules folder while pushing data on git, create a file named .gitignore; inside the file mention which folder should not be included.

**Nodemon package (time saving)**

What is Nodemon package?

To avoid running the node command every time we change the code, we can use Nodemon package. Nodemon will continuously run the project.

How to install it?

npm install -g nodemon

How to use it?

nodemon .\index.js

**Interview point**

Is Node JS async or synch?

Async. (If a function takes too long to get data from different sources, thread will not wait to get the data; instead it will continue executing the code after it in the call stack. Once the data is received the former code will be executed. Js is faster because of the async nature)

**Node JS and JavaScript are single threaded, but async. How?**

JS is single threaded because it has only one thread to execute code line by line. But if any line of code require input from other sources and take time to get data, then javaScript will not wait till the time that data is received. It will continue to execute the rest of the code and once the data is available, the skipped stack will be executed. This is the reason JavaScript is faster.

Make a server

Crease header and API body

Create an API with static data

Put data in another file

const http = require("http");

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

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

    resp.write(JSON.stringify({name:"Hari", email:"hari@gmail.com"}));

    resp.end();

}).listen(4500);

Open browser and type [localhost:4500](http://localhost:4500/)

Also try out this API call in postman

Now to fetch the data in a separate file, create file named data.js and keep the data in that file.

//data.js

const data = {name:"John", email:"john@gmail.com"}

module.exports = data;

const http = require("http");

const data = require('./data');

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

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

    resp.write(JSON.stringify(data));

    resp.end();

}).listen(4500);

Now for multiple users

//data.js

const data = [

    {name:"John", email:"john@gmail.com"},

    {name:"Hari", email:"hari@gmail.com"},

    {name:"Binu", email:"binu@gmail.com"},

    {name:"Herman", email:"german@gmail.com"},

]

module.exports = data;

**Interview Point**

What is 200 OK

201 Created

404 Not found

500 Internal Server Error

**Input from Command Line**

* Set input from command line
* Create file with input
* Delete the file with input

console.log(process.argv);//argv stands for argument vector

From command line

**output**

PS C:\Data\NodeJs\Node js Tutorial> node .\index.js

[

'C:\\Program Files\\nodejs\\node.exe',

'C:\\Data\\NodeJs\\Node js Tutorial\\index.js'

]

PS C:\Data\NodeJs\Node js Tutorial> node .\index.js Hari Kumar

[

'C:\\Program Files\\nodejs\\node.exe',

'C:\\Data\\NodeJs\\Node js Tutorial\\index.js',

'Hari',

'Kumar'

]

console.log(process.argv[2]);

PS C:\Data\NodeJs\Node js Tutorial> node .\index.js Hari Kumar

**output**

Hari

**How to create a file dynamically from command line**

const fs = require('fs');

const input = process.argv;

fs.writeFileSync(input[2], input[3]);

PS C:\Data\NodeJs\Node js Tutorial> node index.js apple.txt "Apple is a fruit"

It can be seen that a file named apple.txt is created and the contents of the file is Apple is a fruit.

**How to remove a file dynamically from command line**

const fs = require('fs');

const input = process.argv;

fs.unlinkSync(input[2]);

PS C:\Data\NodeJs\Node js Tutorial> node index.js apple.txt

It can be seen that the file named apple.txt has been deleted

**Create and remove file from command line**

const fs = require('fs');

const input = process.argv;

if (input[2]==add){

    fs.writeFileSync(input[3],input[4]);

}else if(input[2]==remove){

    fs.unlinkSync(input[3]);

}else

    console.log("invalid input");

**Show File List**

* Make files in a folder
* Use **path module**
* Get file names and print

const fs = require('fs');

const path = require('path');

fs.mkdir(path.join(\_\_dirname, 'files'),

    (err) => {

        if (err) {

            return console.error(err);

        }

        console.log('Directory created successfully!');

    });

const dirPath = path.join(\_\_dirname, "files");

for(i=0; i<5; i++){

    fs.writeFileSync(`${dirPath}/myFile${i}.txt`, "A simple text file");

}

* Imported fs and path modules
* Created a directory named files
* Created five files inside the files directory.

**To list the files in a directory**

const { log } = require('console');

const fs = require('fs');

const path = require ('path');

const dirPath = path.join(\_\_dirname, 'files');

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

    console.log(files);

});

Note that the files are listed as an array.

Use forEach function to get individual files, like

const { log } = require('console');

const fs = require('fs');

const path = require ('path');

const dirPath = path.join(\_\_dirname, 'files');

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

    files.forEach((file)=>{

        if(file=="myFile3.txt"){

            console.log(file);

        }

    })

});

**Interview Question**

Is it possible to access files and folders outside the project directory?

No. When we run node inside a folder the folder becomes a web server. It is not possible to access files outside the web server.

What is path module is used for?

Path module is used to access folder

**CRUD with file system**

Create, Read, Update, Delete and Interview Question

Create a file named test.txt inside a folder named crud

* We need to import fs and path modules
* To create a folder named crud, we have to use mkdir command of fs module

CREATE

const fs = require('fs');

const path = require('path');

fs.mkdir(path.join(\_\_dirname, 'crud'), (err)=>{

    if(err){

        console.log("directory named crud could not be created");

    }else{

        const dirPath = path.join(\_\_dirname, 'crud');

        fs.writeFileSync(`${dirPath}/test.txt`, "file named test.txt is created inside crud directory");

    }

});

READ

const fs = require('fs');

const path = require('path');

const dirPath = path.join(\_\_dirname, 'crud');

//const filePath = `${dirPath}/test.txt`;

const filePath = dirPath+"/test.txt";

fs.readFile(filePath, "utf-8", (err, character)=>{

    if (!err){

        console.log(character);

    }

});

UPDATE

const fs = require('fs');

const path = require('path');

const dirPath = path.join(\_\_dirname, 'crud');

const filePath = `${dirPath}/test.txt`;

fs.appendFile(filePath, "\nThis second line is appended to file as part of update operation",

(err)=>{

    if(!err){

        console.log("File is updated");

    }

});

DELETE operation of file can be done using the fs.unlinkSync() method as already explained above.

Rename the test.txt file to mytest.txt

const fs = require('fs');

const path = require('path');

const dirPath = path.join(\_\_dirname, 'crud');

const oldFile = path.join(dirPath+"/test.txt");

const newFile = path.join(dirPath+"/myTest.txt");

fs.rename(oldFile, newFile, (err)=>{

    if(!err){

        console.log("File renamed successfully");

    }

});

**Interview Question**

What is buffer?

Buffer is a temporary memory location.

**Asynchronous in Node JS**

In synchronous programming functions are executed one at a time in the chronological order, that is first function, second function and third function and so on.

In asynchronous programming second function does not wait for the first function to be executed, for example

setTimeout(()=>console.log("First function"), 2000);

console.log("Second function");

console.log("Third function");

Result of the above code is:

Second function

Third function

First function

drawback of asynchronous programming

let a = 20;

let b = 5;

setTimeout(()=> b=10, 2000);

console.log(a+b);

Out of the above code is 25 and not 30.

//handle the above drawback using promise

let a = 20;

let b = 0;

let myPromise = new Promise((resolve, reject)=>{

    setTimeout(()=>{

        resolve(100);

    }, 2000);

});

myPromise.then(data=>{

    b = data;

    console.log(a+b);

});

Tommy's Way of sorting out such issue of asynchronous nature

let a = 100;

let b = 0;

setTimeout(()=>{

    b = 200;

    console.log(a+b);

}, 2000);

**How Node JS Woks**

Call stack, Node JS API, CallBack Queue, Example, Interview Question

* Every function is registered in the call stack prior to execution
* By default internally the main () function is first registered in the call stack. It follows LIFO. Therefore the main () function will be the last to be removed from the call stack.
* If there is a third party (C or C++) function such as SetTimeout(), it will be moved to Node API block and then one by one to **Callstack Queue** according to the priority.
* Once the call stack becomes empty, those functions in the **callstack queue** will be moved back to call stack for execution one by one.

**Express js**

Express js is a framework of Node js.

How to install express js? – npm install express

Example to some response to the client

const express = require('express');

const app = express();

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

    resp.send(`<H1>Hello, This is home page</H1>\n

    <a href="/about">Go to About Page</a>`);

});

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

    resp.send(`<H1>You are in About page</H1>\n<a href="/home">Go to Home Page</a>`);

});

app.listen(5500);

//localhost:5500?name="Hari Kumar"

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

    resp.send("<H1>Hello "+JSON.parse(req.query.name)+", Welcome to our Home page</H1>");

});

//To send an input box and a button

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

    resp.send(`

    <input type="text" placeholder="Enter Name"/>

    <button>submit</button>

    `);

});

//To send json data as single object

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

    resp.send({

        name: "Anil Kumar",

        email: "anil@gmail.com",

        mob:"9996858068"});

});

//To send JSON data as array of objects

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

    resp.send([

        {

            name: "Anil Kumar",

            email: "anil@gmail.com",

            mob: "9996858068"

        },

        {

            name: "Kiran",

            email: "kiran@gmail.com",

            mob: "9996823542"

        }

    ]);

});

//To send html page as response

const express = require('express');

const path = require('path');

const app = express();

const publicFolderPath = path.join(\_\_dirname, 'public');

//app.use(express.static(publicFolderPath));

express().use(express.static(publicFolderPath));

// http://localhost:5500/

//http://localhost:5500/help.html

//http://localhost:5500/about.html

app.listen(5500);

Note: use method is called on express() function of the express module which is assigned to app; and static() function is called directly on the express module.

Static() function is used to load static pages.

How to remove extension from file name in the url. For that we have to use the get function instead of static function

//To avoid file extension from the url

const express = require('express');

const path = require('path');

const app = express();

const publicFolderPath = path.join(\_\_dirname, 'public');

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

    resp.sendFile(`${publicFolderPath}/index.html`);

    //http://localhost:5500/

});

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

    resp.sendFile(`${publicFolderPath}/about.html`);

    //http://localhost:5500/about

});

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

    resp.sendFile(`${publicFolderPath}/help.html`);

    //http://localhost:5500/help

});

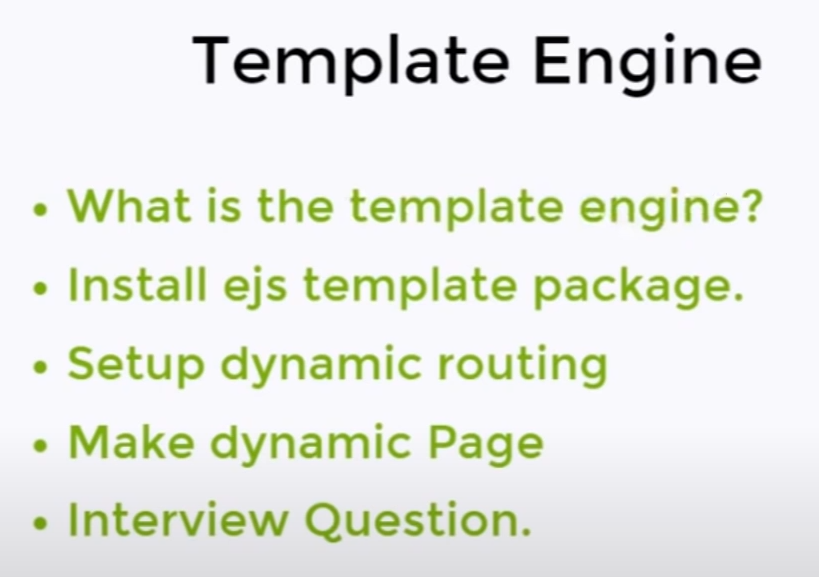
app.get('\*', (req, resp)=>{

    resp.sendFile(`${publicFolderPath}/p404.html`);

    //http://localhost:5500/asdf

});

app.listen(5500);



Whenever we use template engine, all the ejs files should be in a folder named views.

//Using Template Engine - ejs

const express = require('express');

const path = require('path');

const app = express();

const publicFolderPath = path.join(\_\_dirname, 'public');

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

    resp.sendFile(`${publicFolderPath}/index.html`);

});

//tell node that we are using template engine ejs

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

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

    //to render the contents of ejs page

    resp.render('profile');

});

app.listen(5500);

//Sending dynamic page using Template Engine - ejs

const express = require('express');

const path = require('path');

const app = express();

const publicFolderPath = path.join(\_\_dirname, 'public');

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

    resp.sendFile(`${publicFolderPath}/index.html`);

});

//tell node that we are using template engine ejs

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

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

    //to send dynamic page

    const user = {name:"Hari", age:35, email:"hari@gmail.com"}

    resp.render('profile', {user});

});

app.listen(5500);

<body>

    <h1>Profile.ejs is displayed</h1>

    <!-- string interpolation in ejs -->

    <h2><%=user.name%><br><%=user.age%><br><%=user.email%></h2>

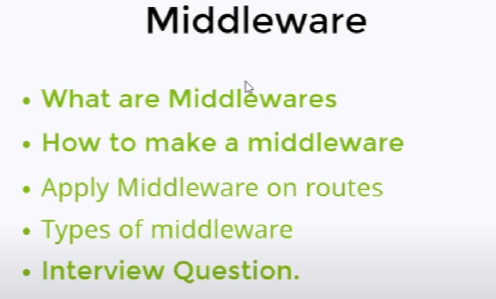
</body>

In order to put header on each page, a folder named common is created inside the views folder. Inside common folder a file named header is created. Now this file should be included in each ejs file to display the header, like

<%- include ('common/header')%>

Note the hiphen sing after the % sign. Change it to = and see the difference.

**At this point a new file index2.js is created**



The middleware in node. js is a function. It will have all the access for requesting an object, responding to an object, and moving to the next middleware function in the application request-response cycle. Below is an example of application level middleware (applies to all the routes of the application)

const express = require('express');

const app = express();

//create a middleware

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

    if(!req.query.age){

        resp.send('<h1>Please provide age</h1>');

    }else if(req.query.age<18){

        resp.send('<h1>You are a minor; You cannot access this page</h1>');

    }else{

        next();

    }

}

app.use(reqFilter);

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

    resp.send("<h1>Welcome to Home page</h1>")

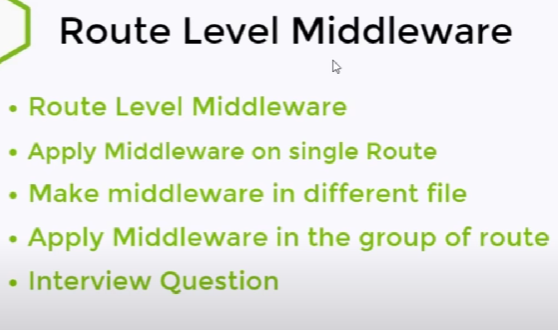
});

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

    resp.send("<h1>Users page</h1>")

});





To apply middleware on Home page route

//app.use(reqFilter);

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

    resp.send("<h1>Welcome to Home page</h1>")

});

//To keep the middleware in a different file

const express = require('express');

const reqFilter = require('./index2ReqFilter');

const app = express();

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

    resp.send("<h1>Welcome to Home page</h1>")

});

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

    resp.send("<h1>Users page</h1>")

});

app.listen(5500);

Here the middleware is kept in index2ReqFilter.js file.

//middleware file name is index2ReqFilter.js

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

    if(!req.query.age){

        resp.send('<h1>Please provide age</h1>');

    }else if(req.query.age<18){

        resp.send('<h1>You are a minor; You cannot access this page</h1>');

    }else{

        next();

    }

}

If the middleware is required to be applied on a group of routes, then instead of giving the middleware name on every http method, coding can be done as follows

//To use the middleware on a group of routes

const express = require('express');

const reqFilter = require('./index2ReqFilter');

const app = express();

const route = express.Router().use(reqFilter);

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

    resp.send("<h1>Welcome to Home page</h1>")

});

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

    resp.send("<h1>Users page</h1>")

});

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

    resp.send("<h1>About page</h1>")

});

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

    resp.send("<h1>Contact Page</h1>");

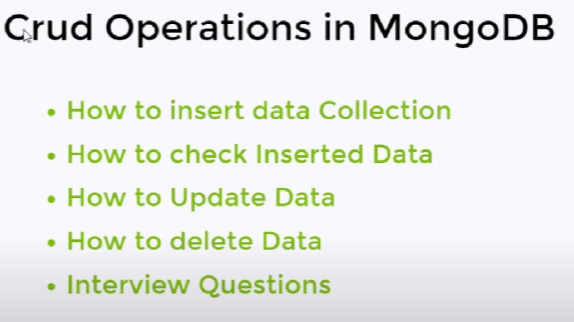
});

app.use('/', route);

app.listen(5500);



|  |  |
| --- | --- |
| **MongoDB** | **MySQL** |
| NoSQL database. | SQL database |
| Data stored in a collection | Data store in a table |
| Collection doesn’t have rows and columns | Table has rows and columns |
| Data is stored in the form of JSON object | Data is stores in rows or records |



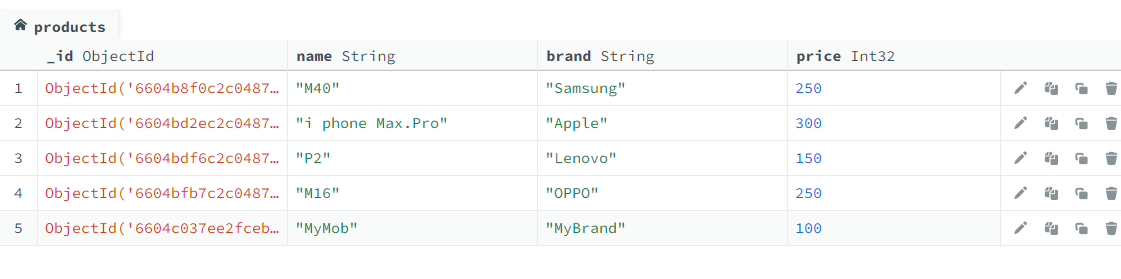
To insert data from command line



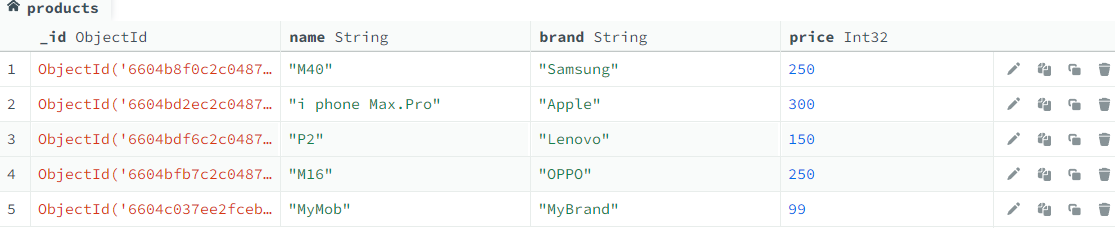
To read data from command line



To update data from command line

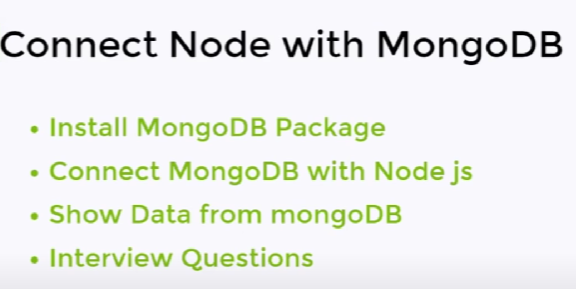






To delete object from command line



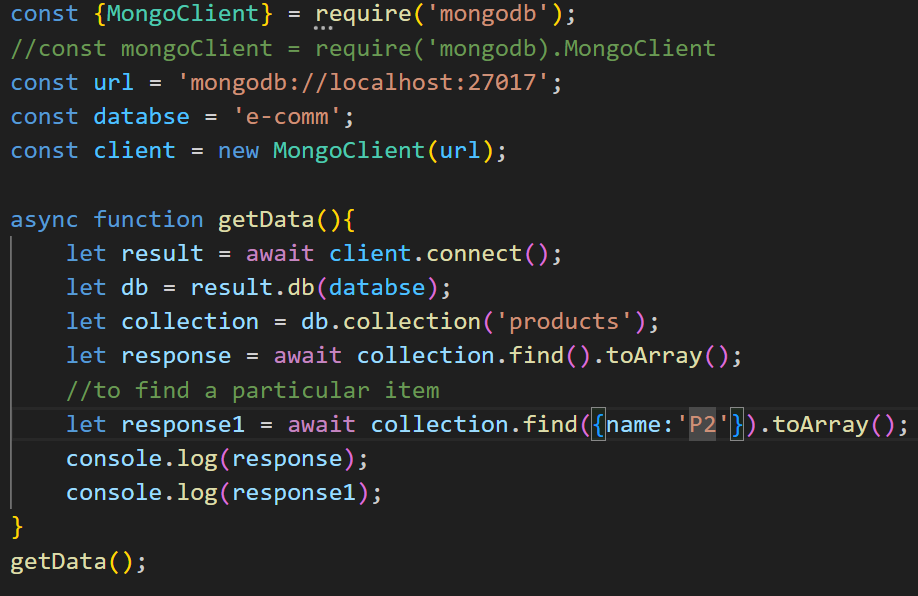


Install MongoDB driver using npm



Import MongoClent at the top of the file





connect() function on the MongoClient instance returns a promise, so we have to use await; await can be used only in an async function. Again toArray functions return promise which is handled using await.

**To handle promise, either use async await or promise**

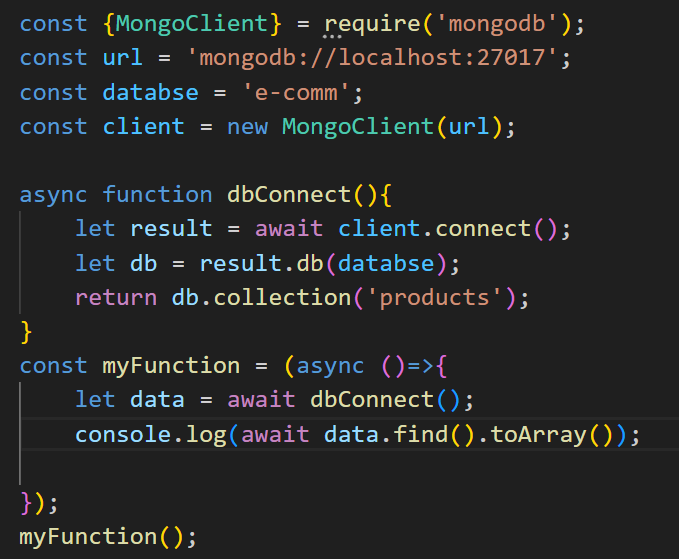
Before creating separate file for db connection, we will break the above code into functions.

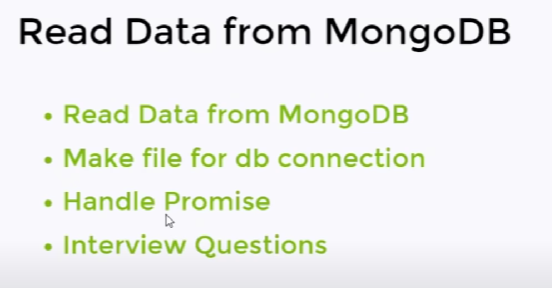
**First using promise to fetch data**



In this way we can call insert, update, and delete etc. functions

**Using async await** – for that first we create an async function.





Create separate file for db connection

In this example we have two collections (tables) named products and user in the database e-comm. We have created a file named mongo\_e-comm.js to connect to the database. See the code below

//file name mongo\_e-comm.js

const {MongoClient} = require('mongodb');

const url = 'mongodb://localhost:27017';

const databse = 'e-comm';

const client = new MongoClient(url);

async function connectUser(){

    let result = await client.connect();

    let db = result.db(databse);

    return db.collection('user');

}

async function connectProducts(){

    let result = await client.connect();

    let db = result.db(databse);

    return db.collection('products');

}

module.exports = {connectUser, connectProducts};

**To find the list of products in the products collection using async-await**

const {connectUser, connectProducts} = require ('./mongo\_e-comm');

const getProducts = async ()=>{

    let data = await connectProducts();

    console.log(await data.find().toArray());

};

getProducts ();

**To find the list of products in the user collection using promise**

const {connectUser, connectProducts} = require ('./mongo\_e-comm');

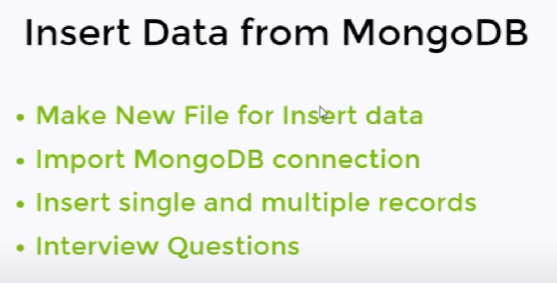
connectUser().then((res=>{

    res.find().toArray().then(users=>{

        console.log((users));

    });

}));



To insert product a new file named insertProduct.js is created

const {connectProducts} = require('./mongo\_e-comm');

const insertProduct = async ()=>{

    let db = await connectProducts();

    const result = await db.insertOne({

        name:"myMob", brand:"myBrand", price:500, category:"Mobile"

    });

    if(result.acknowledged){

        console.log("Object inserted successfully");

    }

}

insertProduct();

Code on insert multiple products – this time promise is used

connectProducts().then((res)=>{

    res.insertMany([

        {name:"userMob1", brand:"userBrand", price:699, category:"Mobile"},

        {name:"userMob2", brand:"userBrand", price:799, category:"Mobile"},

        {name:"userMob3", brand:"userBrand", price:899, category:"Mobile"},

    ]).then((result=>{

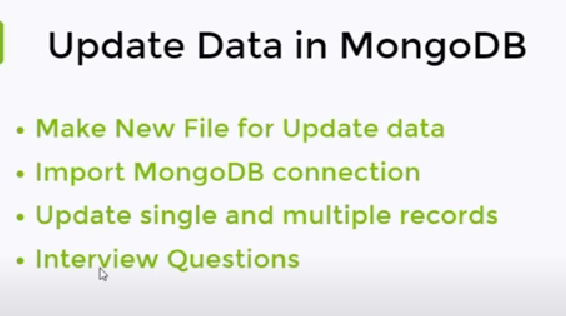
        if(result.acknowledged){

            console.log("Array of objects inserted successfully");

        }

    }));

});



Update single object – a file named updateProduct.js is created

const{connectProducts} = require ('./mongo\_e-comm');

const updateSingleProduct = async ()=>{

    let db = await connectProducts();

    let result =await db.updateOne(

        {name:"myMob"}, {$set:{price:499}}

    );

    if(result.acknowledged){

        console.log("Object updated successfully!");

    }

}

updateSingleProduct();

Update multiple objects

const{connectProducts} = require ('./mongo\_e-comm');

connectProducts().then(res=>{

    res.updateMany(

        {name: {$regex:"new"}}, {$set:{price:101, name:"userMob"}},

    ).then(result=>{

        if(result.acknowledged){

            console.log("Multiple objects updated successfully!");

        }

    });

});

In this example all objects whose name includes the word new are affected.



deleteProduct.js

const {connectProducts} = require('./mongo\_e-comm');

const deleteData = async ()=>{

    let db = await connectProducts();

    let result = await db.deleteOne({name:"myMob"});

    if(result.acknowledged){

        console.log("One object deleted successfully");

    }

}

deleteData();

Using deleteMany() to delete multiple records

connectProducts().then(resp=>{

    resp.deleteMany({name:"userMob"}).then(res=>{

        if(res.deletedCount===0){

            console.log("No Data found for deletion");

        }

        else if(res.acknowledged){

            console.log("Multiple records deleted successfully");

        }

    });

});