Node JavaScript

Node.js is a Run time Environment for executing JavaScript code outside the browser

**Machine Code**

**Js Engine**

**Js Code**

Each browser are uses different Js Engine so same Js code may react differently in different browser

Eg..,

* Chrome uses V8 Engine for executing JavaScript
* Edges uses Chakra Engine for executing JavaScript
* Mozilla uses Spider Monkey Engine for executing JavaScript

Before 2009 JavaScript code are executed by only Browser, but after 2009 Rayan Dahl Came up with a idea that he have took v8 Engine of Chrome and Embedded the C++ and Make it as Node.js

This allows to execute the Js Code outside the browser But it does not include Windows & Document Object instead it include some other useful Object other than that.

Node Js have asynchronous nature for executing the request. Means only Single Thread is allocated to handle all the request asynchronously

For an instance, at Synchronous Architecture Each request has been allocated one thread, For Example the concurrent Application we will get a lot of Request at a same time , for each request we will allocate a thread if request is heavy we will be running out of thread, So we need more hardware to over come this problem but adding more hardware is more costly as well as at lite request time a lot of hardware remains unused /Idle.

By Default ASP.Net, Ruby on Rails ..etc, are Synchronous in nature

How Node Works?

Since Node uses single threaded Architecture means Single thread is used to handle all the request ->How ?

When request comes ,it will handles the request by giving to data base while data base executing the query the same thread without waiting for the database execution it goes to serve other request came. when database complete the execution it put the result in Event Queue. The thread regularly check the Event Queue, if Result is available in the Event Queue then thread servers the Request.

**Global Object**

Console Object is one of the example of Global Object which can be accessed anywhere in any files

In browsers we have window Object which is used to represent the global scope. All the functions and variables that are represented globally can be called using this window Object.

We write it as console.log() or window.console.log() .If the window is not represented JavaScript Engines automatically prefix that. Because that’s where it is defined.

In Node we don’t have this window Object which is replaced by Object called global.

If define a variable its scope will be stopped within the file which cannot be accessed outside the file like global variable this is because of Modular functions of Node Js

**Modular**

Every file in the Node Application is consider as a Module.

Variable & function that are defined in that modules are scoped within that Module

Suppose you want to make a function or a variable to use in a module to another module you need to explicitly export make it public and import it in the specific module

Every Node Application have at least one Module as the Main Module ( App.js)

Console.log(module) – Will Provide the complete properties of the Modules

**Creating Global Variable and Functions**

We having following variable and functions in logger.js Module

var Url =” <https://www.google.com>”;

functions logger(message)

{

//sending some http request

//doing something

}

This is scoped with in this module suppose we want to export this outside the module we need to add this to exports Object of module

module.exports.URL= Url;

module.exports.Logging=logger;

URL & Logging are going to public name which is going to be known by others Modules which is going to use it like Properties concepts In C#, Url and logger private names is not known by other Modules which is using it.

**Importing a Module in another Module**

require() is the function is used to import a module’s Variable and functions into another module.

Adding Logger.js into app.js

require(‘./logger’);

.- represent the current folder know as period symbol.

require(‘./logger.js’); is same as require(‘./logger’); any thing can be used. Because node assume it a JavaScript file and add .js next to it

suppose it inside some folder of current directory(Soft folder)

url will be ‘./subfolder/Logger’

Suppose file is in the Parent folder then url will be ‘../Parentfolder/Logger’.

Require function will return the export object of Module. Using that export object returned we can call all the publicly exported functions or variable.

var exp= require(‘./logger’);

exp.Logger(“Some Message”);

exp is the export object that have logger Module’s publicly exported Variable and functions.

It is better to store the Module inside a const variable to avoid unnecessary assignment in the further steps

jshint is a tool which is used for identify errors in the js code.

Suppose if we need to load alone a single function or variable we export like

module.exports=logger;

this is will point the logger method alone

var exp= require(‘./logger’);

exp(“Some Message”);

here the exp will act like logger method not export object

Previous way we can use It for when there is a lot of things to exports. Current one is used for only one function or variable only there to load

***NPM(Node Package Manager)***

A command line tool or registry of third party library that we can add to out node Application, pretty much any kind of functionality that you need to add to your application is more slightly free open source package or library or node module on NPM registry

There are a lot of building blocks which is free and reusable code.

***How to add Node Module Application?***

This can be done using NPM

{NPM always comes with Node installation need not required Separate Installation}

Version of Node will be different from version of npm because they are developed independently.

***Installing Specific Version***

npm i -g [npm@5.5.1](mailto:npm@5.5.1)

i -> Install

-g -> Global

***package.json***

It is Json file that include some information about your Application Or Project. It hold bunch of Meta data of your application. It holds all the details about the packages or module add to applications like Package name ,author name, Git repo location.

Generally All node Application have this package.json file.

To create a package.json file we have to run npm init

Will ask bunch of Question for Accepting press enter

Note : before Adding any Package to your Application as best Practise you have to add package.json file to the Application

Fastest way to create package.json file with accepting all the default values is npm init --yes

Yes in sense accepting all the default values

***Adding Third Party Library to the Node Application***

Whenever you are working with node Packages good starting point will be ***npmjs.com.***

Installing Sample Package

npm install underscore

install 🡪 to install a package in the node Application we can also use i instead of it

underscore is the Package name.

When run above command two things will be happened it create an entry in package. json under dependency and also store the package in node\_module

(First time your installing a package node\_module folder will be created and packages are stored in it)

For the older version you have use

npm i underscore --save

to register with package.json we have use --save. Now a days it not necessary to use --save for updating package.json and it will download latest version package and store it in the node\_module folder.

***Uninstalling npm***

npm uninstall -g npm

***Installing Mongoose***

Mongoose is used to store ours data in Mongo Db

**Using the installed Packages**

We need to import the Package inside our Program

var \_unders= req(“underscore”);

By default it Node will take underscore as

1.Core Module

2. If not Available in the Core Module it will take it as file or folder in this Project

3. If it is not file or folder it will Assume it module from node module.

Note : Earlier we said like to refer file or folder we used ‘./underscore’

‘./underscore’ means it will look for the file underscore.js or it will search for the folder underscore & inside underscore folder look for the file index.js

**NPM Packages and Source Code**

Need not to copy or load the node\_module folder to source code repository, because it might be bigger in size

*The Questions arise what happen if we not including the node\_module our Application will work as before?*

Answer is yes because even though we not including node\_module, All the dependencies are will updated in Package.json file. So that after getting the Application from Source Code repo just we have to run following command

***npm i***

this will download required packages (mentioned in Package.json).

package-lock.json created by NPM when we install npm package purely for it to do its jobs we need not to worry about it

***To exclude a file from commit to Git repo***

* Just include a file call .gitignore and this will hold file and folder need to ignored while committing
* Just include file name and folder name
* For folder name we have like Folder\_name/
* Forward slash indicate it is folder not a file
* For file name of the file alone enough

Semantic Versioning

Each Packages have the versions like 4.8.1 in the format of ***Major.Minor.Patch***

Major- Adding new features that will break existing one

Minor -> Adding new features without breaking existing features

Patch -> fixing the bugs in Previous Features

Symbols like Caret(^), tilde (~) are used

^4.8.1 and 4.x is similar indicate upgrading of Minor and Patch are allowed not Major version

~4.8.1 and 4.8.x are similar upgrading of Path is allowed not allowed to upgrade major and minor version

4.8.1 no upgrading is allowed similar version only should use.

***To see all the list of dependency installed and their versions***

npm list

tree like structure will be displayed which shows a list of dependency and their dependencies

Suppose you want only the dependencies of Application not package dependencies then

npm list –depth=0

***To View Package.json file of the Particular Package***

npm view package-name

Eg.., npm view mongoose

***To View only the dependency in Package.json file***

npm view mongoose dependencies

***To view version of current Package***

npm view mongoose version

***To view All versions of current Package***

npm view mongoose versions

***Installing Specific version of Package***

npm i mongoose@2.4.2

***To identify what package is Outdated and what its stable and newer version***

***npm outdated***

it shows result as five columns,

* Package 🡪 Name of Package
* Current 🡪 Current version of Package.
* Wanted 🡪 Recommend version (based on the Caret and tilde symbol, if caret is mentions it shows latest Minor version ,tilde is mentioned it shows latest Patch version )
* Latest 🡪 Latest available version
* Location🡪 Place where Package is located

***Updating to Latest Recommended Version***

npm update 🡪 this will update latest version of Minor and Patch not Major release in order not to break the existing functionality.

***Update the Dependencies to very latest Version(Updating Major versions)***

We have install another one tool for this purpose.

Name of Package is npm-check-updates

Command to install it is npm i -g npm-check-updates

This Package used to check the outdated Packages and shows the latest Package of it.

For this Purpose you have to run following command,

***npm-check-updates***

ncu both commands are same will shows the outdated Packages and latest of it

To update in Package.json file alone

***ncu -u***

this will update in Package.json file not download latest packages

Now run ***npm i*** to install latest version of Mongoose

Development Dependencies

There are some Packages that is used during development time which is not need for Productions

There are two types of Dependencies

1. Application dependencies
2. Development dependencies

The development dependencies are not need for Productions which is like Code Analysis Tool, Tool to write unit Test, etc..,

Eg;

JSHint is a community-driven tool that detects errors and potential problems in JavaScript code. Since JSHint is so flexible, you can easily adjust it in the environment you expect your code to execute. JSHint is open source and will always stay this way.

To install a package as development dependencies

***npm i jshint --save-dev***

--save to update in Package.json (mandatory to use to save package in dev dependencies)

-dev is save under Development dependencies

Note: This kind of Dependencies are stored under devDependencies

in Package.json not under dependencies.

***Uninstall Package***

npm uninstall Package-name

Instead of uninstall we can also use un

This is remove the entry in Package.json as well as remove the packages node\_module folder.

**Global Packages**

Packages like jshint are specific to the Application, there are some of other Package which is command line tool that can accessed from everywhere

Eg.., npm ,ng

Which is not for specific folder or specific Project

If want to install a Package Globally we use -g while installing that we saw already

Eg npm i -g [npm@5.5.1](mailto:npm@5.5.1)

Also we use -g to all the above commands like

npm -g outdated

this will search through out the system and will take the all the Packages are outdated.

Publishing the Package

1.Creating the Account in npm js

***npm adduser***

To create a new user

It will ask for

* User name : Need to all lowercase
* Password : at least 10 character
* Email : Email address

*Account will be created*

Creating Package

1. We need to add package.json file to our package {npm init}
2. Add a file called index.js {Starting Point of the Program}
3. Add your code in index.js and export it

2.Login into the Account

npm login

It will ask for

* + - User name
    - Password
    - Email

3.After login Publish the Package

npm publish

It is published to Npm repo now you add to any Application by just installing it as like Previous Packages.

**Updating the Published Package**

Depending upon the changes you have made you have to update the version number

* Just fixing the errors then change it in Patches
* Just add the features then change it in Minor
* Just add the features that affect existing features then change it in Major.

You can do to it in two ways

* Manually Updating Package.json file under version
* Using npm command line
  + npm version patch -> to update Patch
  + npm version Minor-> to update Minor
  + npm version Major -> to update Major

and then npm publish the packages once again in npm repo.

***Express***

Fast and light weight framework to build Web Application

***Restful Service***

* Restful Services also called as Restful API
* Rest stands for Representational State Transfer.
* Not all but Most of Application in the world are using client-server architecture. Client is front end part and Server is backend part. Client need to talk to server to get/save the data. The Communication happened using http protocol.
* At server we expose bunch of services that accessed by client through http Protocol, Client can directly call the Services using sending http request. Here only REST comes into picture.
* Rest is conversion to build this HTTP Services.
* Type of HTTP Method determine the Type of Operations need to be Perform on Server, Following are standard HTTP Methods
  + POST: to create a resource
  + PUT: to update it
  + GET: to read it
  + DELETE: to delete it



Note : Generally we can able see the api in the URL to indicate that it is Company is Provide Restful services

***Express***

Fast and light weight framework to build Web Application

Installing Express Packages

***npm i express***

Using the Express Module

1. First we have to load Express Module,

const express= require(‘express’);

This will return a function lets call it as express

1. We need to call the express functions which we got on previous steps

const app = express();

This will return a object of type Express

1. This Object have some useful method like app.get(), app.post, app.put(), app.delete() 🡪all are http methods
2. We are going to see app.get()

* This going to take two arguments
  + First one is Path
  + Second one is call back functions which is called when we got request to above end point(Path)
  + Call back function will take 2 arg 🡪 req,res
* app.get(‘/’ ,(req,res) => {

res.send(‘Hello World’);

} ) ;

* ‘/ ‘ 🡪 indicate Homepage

Note

Request object have bunch of useful properties that we can refer in give information about incoming request. For information we redirect to expressjs.com 🡪 api reference 🡪 4.x

* Finally we have listen to given port

app.listen(3000,()=>console.log(“App is listening”));

call back functions is optional that can used for our utilities

* After this we have to run our Applications

npm index.js

the application starts to listen on Port 3000

Open the Browser and load localhost:3000/ you will receive “Hello World” Message

We can also have path like ‘/api/course/’ we load our browser like localhost:3000/api/course

***Nodemon***

Nodemon is short for Node monitor, when ever we are making any changes we have to stop the services manually and restart it,.

To Avoid doing this manually we have package called nodemon which will keep on the watch all the files in current folder any file with any extension and if any changes & save it Nodemon will restart the Application or Process due to changes automatically.

Installing Nodemon

***npm i -g nodemon***

Running the Services

***nodemon index.js***

nodemon is the package which we installed already and index.js is file which we running.



Environment Variable

In Above Program we have given 3000 Port number is hard coded value, now in Developer machine it is working properly but this will not case in the Production environment.

So, Port number need to be dynamically assign by the hosting environment.

For this Purpose we are using environment Variable. Environment Variable is basically is part of environment in which process run. Generally its value will be set outside the Application.

To read PORT environment Variable Value, This can be done one global object is called process

const port = process.env.PORT|| 3000;

* process 🡪Global Object
* env 🡪 short for environment
* PORT🡪 Environment Variable Name

if environment Variable is set already it will take that value otherwise by default it will take 3000

Setting Environment variable by following command

***set PORT=5000***

* PORT 🡪 environment variable
* set 🡪Keyword used to set environment variable
* 5000 🡪 is value
* Now port number will be 5000

app.listen(port,()=>console.log(`Application is listening.......${port}`))

Note in above statement we used Template string which new features in Js where we add directly variable inside a string. Back tick(`) need to used at starting and end of the printing statement.

***Ctrl +^C to stop any Process***

***GET with Parameter***

Getting a Specific Records like Getting courses published on Particular date.

app.get('/api/courses/:year/:month/:day',(req,res)=>{

res.send(req.params.year+"/"+req.params.day);

});

Any thing above code followed by /: is parameter, any number of parameter we can have .Generally

<http://localhost:5200/api/courses/1995/06/22>

req.params 🡪 will returns complete json object with key value pair that like {"year":"1995","month":"06","day":"22"}

here their name based on the route Parameter.

for knowing the specific value of a parameter in URL we can use like req.params.year

***Query String***

In Express we also get Query String Parameter, these are parameter we add in the URL after Questions(?) mark.

We use Query String Parameter to provide additional data to backend Services.

Route Parameter for essential or required Values

Query String for anything that is Optional

Reading Query String

<http://localhost:5200/api/courses/1995/06/22>?sortBy=name

***req.query***

{

sortBy: name

}

Will be returned

***Requesting For a particular record in the Browser(GET)***

var courses =[{id :1,name :"Angular"},{id :2,name :"Node js"},{id :1,name:"Python"}];

app.get("/api/courses/:id",(req,res)=>

{

const course = courses.find(x=>x.id===parseInt(req.params.id));

if(!course) res.status(404).send("Found No Records");

res.send(course);

});

Generally req.params.id will returns String type,So we need to convert to int so we can use one Global Method parseInt().

Requesting for “/ api/courses/1”

It will return {id :1,name :"Angular"}

Requesting for “/api/courses/12” which is actually not available id as we are sending as Parameter .

It will return message “Found no Record” and Set the Status as 404

***How to Check Status ?***

*Right click 🡪 inspect 🡪 Network Tab 🡪 in filter select all 🡪 refer the Page🡪 now able to identify the status*

***How to respond to HTTP Post Request?***

app.post( path, call back function) which is used to handle the post request

app.post('/api/courses',(req,res)=>

{

const course= { id :1,name :req.body.name };

courses.push(course);

/\* Adding the Object to Courses array \*/

res.send(course);

});

When We need to read an object which is in the Body of the Request, This can done using following Code

*req.body.PropertyName*

*eg.., req.body.name*

*Assuming that in the request body we have an object that having name Property*

In order Above Code to be work, we need to enable parsing of Json Object in the body of the Request. By default, this is disabled in Express package

At top after getting App Object,

app.use(express.json());

Basically what we are doing here is adding some piece of Middleware,When we call express.json(), this will return piece of Middleware, then we use app.use to that middle ware in the request processing pipeline.

**POSTMAN**

To call http services we use an tool called Postman which is a chrome extension

**Input Validation**

Joi is Node Package which is used to validate the input.

***Installing Joi***

**npm i** [**joi@13.1.0**](mailto:joi@13.1.0)

As above package(Joi) is not available, we need to use alternative package

**npm i @hapi/joi**

JOI is renamed as  @hapi/joi

**Importing to index.js**

const Joi = require('@hapi/joi');

On the Top of Index.js we need to load this Module

**const Joi = require(‘joi’);**

It returns a class as in JavaScript we will use pascal conversion for to determine Class in which first letter of every letter need to upper case.

At Route Handler with JOI we need to define Schema for the Validations. A schema defines the shape of the Object what properties we have in that Object.

**Defining a Schema**

const schema =

{

Name=Joi.string().min(3).required()

};

Name Property should be String with minimum of 3 character and It should be required means compulsorily need to include in request Object.

**Validations**

Joi.validate( req.body , schema);

Validate method is used to validate req.body against schema which we defined, Validate Method will returns an Object which have two properties

* Error 🡪 indicate Validation Failure, Object we send is invalid.
* Value 🡪 Validation is Successful , Object we send is Valid.

Note : Only one of this will have the Value at a time

app.post('/api/courses',(req,res)=>

{

const schema ={ name : Joi.string().min(3).required() };

const result =Joi.validate(req.body,schema);

if (result.error)

{

res.status(400).send(result.error.details[0].message);

}

else

{

const course= {id :1,name :req.body.name };

courses.push(course);

res.send(course);

}

});

If there is some error result.error have some values , then it will true result.error.details[0] .message will give exact message,result.error is long json object we sending what is needed to client

Bad Request

res.status(400);

400 🡪 Bad Request

404 🡪 Not Found

**Put Request**

Put request is used for Updating the data, put() is used for this purposes,

app.put(path, call back functions)

path must have route parameter in order to identify the data which need to update

eg.., “api/courses/:id”

call back function is route handler,

Object destructing Feature in Modern Javascript

const result =Joi.validate(req.body,schema);

if (result.error)

{

res.status(400).send(result.error.details[0].message);

}

Since we are getting result object but we are using only error of result of Object remaining is unused to avoid this we can use Object destructing Feature of Modern JavaScript, Instead of using

const result =Joi.validate(req.body,schema);

we are going to use

const {error} =Joi.validate(req.body,schema);

result.error is equivalent to {error} we can use either way, above code will be

const {error} =Joi.validate(req.body,schema);

if (error)

{

res.status(400).send(error.details[0].message);

}

**Put request Services**

app.put('/:id',(req,res)=>

{

//Checking Courses Id is Valid

var course = courses.find(x=>x.id == req.params.id)

if(!course){

res.status(404).send("Invalid Id");

return;

}

// Validating Provided Data is Correct

var schema ={name : Joi.string().min(3).required()}

var {error}= Joi.validate(req.body,schema);

if(error)

{

res.status(400).send(error.details[0].message);

return;

}

//Sucessful

course.name= req.body.name;

res.send(course);

});

Here three scenario is used

1. Validating id is valid at route parameter
2. Validate the data is in correct shape
3. If Both above two is satisfied then Updating the value

**HTTP Delete Request**

Delete request is used for delete the data, delete() is used for this purposes,

app.delete(path, call back functions)

path must have route parameter in order to identify the data which need to update

eg.., “api/courses/:id”

call back function is route handler,

To delete we need to identify the Index of Array, this can be done using,

const index= Courses.indexOf(course);

course 🡪 object for which we need to identify index in Courses Array

indexOf 🡪 Method will return index

To remove one Object from Courses array we can use Splice Method,

Courses.splice(index,1)

Index 🡪 index of item to removed.

1🡪 Number of Object need to be removed.

app.delete("/:id",(req,res)=>

{

var course = courses.find(x=>x.id == req.params.id)

if(!course){

res.status(404).send("Invalid Id");

return;

}

var index = courses.indexOf(course);

courses.splice(index,1);

res.send(course);

});

***Advanced Express***

**Middleware**

* Middleware or Middleware function is one of the Core concept of Express
* Middleware function is generally a function which take request Object, either return response to client or passes to another middleware.
* Route Handler call back function is example of first scenario of Middleware
  + Just it request object and Process it, send its response.
* express.json() is the example of second scenario
  + when we call express.json will return middleware functions, the job of the middleware function is to read the request, if there is Json Object in the body of the Request, it will parse the body of the request into a json object and then it will set req.body property
* When we receive a request in the server, that goes through Request Processing Pipeline.
* In this pipeline we will have one or middleware functions.

**Building a Custom Middleware Function**

* Building our Own middleware function

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

{

*console*.log("logging");

    next();

});

* To build custom middleware function we need 3 parameters
  + Req 🡪 request
  + Res 🡪 response
  + next 🡪Reference to the next middle ware functions
  + next() 🡪 for passing the control to next middleware function if next() not included end of the function this app will be hanged over this place none of remaining operation will not be performed because we not ending request response cycle.
* All the middleware functions are executed in the sequence.

**Build-in Middleware**

* express.json() 🡪 Parse json object in request body
* express.urlencoded() 🡪 parses incoming request with URL encoded payloads that request with body like key1=value1 , key2=value2…, which is traditional approach which not used often this days
  + this middleware will parses the body and set value to req.body
  + if we not setting {extended : true } while installing this middleware we will get warning

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

* express.static() 🡪 to serve static files
  + we will pass an argument will be name of the folder

app.use(express.static('public'));

* + **public is folder name**
  + we can put all our static assets like css, text file ,pic file….. so on.., inside this file
  + just <http://localhost:5400/filename> is alone enough need not to include folder name eg.., <http://localhost:5400/MyPhoto.jpg>
  + result displayed MyPhoto.jpg

**Third Party Middleware**

* There are a lot of third party middleware is there which we can use in ours App
* Url

<https://expressjs.com/en/resources/middleware.html>

* Using morgan Middleware 🡪 which is used log all the incoming HTTP request
  + First we have to install it 🡪 npm install morgan

*var* logger = require("morgan");

app.use(logger('tiny'));

* + We have to include in our file
  + Use it like previous middlewares
  + Tiny is the format we are having various format like this
  + Output will be in console window as follows

GET /api/actions/ 200 99 - 124.939 ms

* + By default it will write to console window but we can make it write to some text files also

**Environment**

* Since , Our code need to be crosses different environment,
* Some codes are required for all environment some codes are not required
* To have a control over environment we have variable called NODE\_ENV
* Its value can be set outside the Application
* Setting the value

*set NODE\_ENV = development;*

*Note : No need of Double or single Quotes.*

* Reading the value in the Application

*var environment = process.env.NODE\_ENV;*

If value is not set by default it will return undefined, Another Method to get the value is

*var environment = app.get(‘env’);*

app is express Object as we seen Previous Chapters

get() 🡪 functions which will return various setting over the Application

env 🡪 determine Current Environment

* Control over Code

if(app.get('env')=="development"){

app.use(logger('tiny'));

console.log('Morgan is enabled.............');

}

If environment is Development below code will be executed if not will be skipped.

Managing Configuration

* How to store configuration settings for your application & override them in each environment.
* There are more package out there, Most Popular is rc .
* But here I am going to use npm config
* Installing config package
  + npm install config
* We need to create a folder inside our Project called ‘config’.
* We adding new file called default.json inside config folder
  + This holds default setting for all the environment
  + Here we have json object for defining default configuration setting
* We are adding development.json holds setting for development environment.
  + We can over ride the default settings as well as we can add additional settings
* Like this for each environment we can have separate configuration files.
* Back to our index.js
* At top we load our config module
  + var config = require(‘config’);
* Using the method
  + config.get(‘name’);
  + config 🡪 Object we got at previous step
  + get() 🡪 used for get the value of Provided Property
  + ‘name’ 🡪 Property name
* For Property.Nexted Property like mail ={name: ….}

Property Name will be “mail.name”

Storing Secretes like Password in the configuration file need to avoided. Passwords can be stored in the Environmental Variables.

***Setting Environment Variable***

set EnvironmentVariableName = ”Value”;

Example ..,

set APP\_Password = 1234

***Reading the this environment Variable***

We need create a file “custom-environment-variables.json” 🡪 name is much more important

Inside that file we will use like

“password” : “APP\_Password”

This will returns Environmental variable value

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

Note : Reading the value

config.get(“password”);

**Debugging**

Up to now we are using console.log() to log the messages at console.

Some Environment we required log messages but in some environment we don’t need, console.log() will not provide any control over it. So, we have to manually comments the lines but this is very tedious,

To avoid this, we can use debug package which provide control over logging the mes sages to console.

**Installing**

npm install debug

**Adding to Module**

*const* debug = require("debug")('app:startUp');

* Above Statement will return a functions which we can use it logging the Messages
* Argument : is Namespace which is going to be the global representation of Debugger

**Using**

debug("Logged...."); 🡪 by this way we can log the messages, debug is reference name of debug package.

**Enabling the debugger at console.**

set DEBUG = app:startup

**Resetting Debugger at Console**

set DUBUG =

Note : Not assigning anything is Resetting.

**Enabling Multiple debugger at console.**

set DEBUG = app:startup,app:db

**Enabling all the debugger at console.**

set DEBUG = app:\*

**Enabling debugger while Running the Appliation**

set DEBUG = app:startup nodemon index.js

**For Multiple debugger defining**

*const* debug1 = require("debug")('app:startUp');

*const* debug2 = require("debug")('app:db');

**Templating Engine**

Suppose we need to return the response to client in form of html Mark-ups we can use Template Engine,

* Popular Templates Engines : Pug, Mustache, EJS

Each Template Engine will have different Syntax to generate Dynamic HTML and returning to Client

Step 1 : **Installing Pug**

npm install pug

Step 2 : **Setting View Engine**

app.set(“view engine”,”pug”);

Step 3 : **Setting path of the view (optional Settings).**

app.set(“views”,”./views”);

Note :

1. “./views” is the Default Path but we can provide Alternative Path too.
2. We have to place all our Templates and views in side views folder.

Step 4: **Create a file called “index.pug”, inside “./views” which can hold our template like html code.**

**Step 5 : writing html code in index.pug**

Note : we need not to use opening and closing markups, Code can be like

|  |
| --- |
| html  head  title= title1  body  h1=message |

title1 & message are going to be variable, whose value can be set in the index.js & Send to client.

Step 5 : Sending Response to client using Pug template which we created,

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

res.render("index",{title:"Mastering" , message :"Hello"});

});

Res.render 🡪 to render the template or views.

Index 🡪 index.pub which is template name (file name) inside view folder

Second object 🡪 to set the value of the variables at index.pug and sending to client

Restructuring Project

* We are going to move the relevant codes under relevant files
* For example, I am going to move all the request handler (PUT, Delete, Post, Get) which is hitting the “api/courses” to separate file, following need to performed.
* Need not to import all the packages to separate module which is need to import on Main Module index.js only.
* We have import in separate module only express module and from express module we have create route Object

Step 1. Importing Express Module

var express = required(‘express’);

Step 2 : Creating route Object.

var router = express.Router();

Step 3. Need to call all the route handler using this router Object.

router.get ( '/api/courses', (*req*,*res*)*=>*{

res.render("index",{title:"Mastering" , message :"Hello"});

});

Step 4. Finally, we have to exports this router.

module.exports= router;

Step 5 . loading in index.js

var courses= required(‘./courses’);

Step 6 Mapping the request

app.use(“api/courses”, courses);

api/courses 🡪 Requested Path

course 🡪 is a router which points to above object

Step 6 Now we can rename route handler path to ‘/’ as the path is given to router at previous step

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

res.render("index",{title:"Mastering" , message :"Hello"}); });

**Asynchronous JavaScript**

* There are three patterns to deal with async code
  + Call-backs
  + Promises
  + Async / Await
* Callback

*console*.log("Before");

*let* userId = Timer(*function*(*i*){

*console*.log(i);

});

*console*.log(`user Id : ${userId}`)

*console*.log("After");

*function* Timer(*callback*)

{

    setTimeout(()*=>*{

*console*.log("ReadMe ....After 2000 ms");

        callback(12);

    },2000);

return 1;

}

* SetTimeout() is an Async Function we need to get the result after the functions is done with the work.
* Call back used here which can used like functions,
  + Function definition is given in the Timer functions call
  + Reference is placed in the parameter of Timer function

Timer(userId,*function*(*user*){

*console*.log(user);

    GitRepository(user.userName,(*Repositories*)*=>*{

*console*.log(Repositories);

    });

});

* Above Statements Provides Nested structure for the complex Program it will become very difficult to understand, This Problem is called Call Back Hell
* To overcome this Problem, we can use Named Function instead of anonyms function.

*let* users=Timer(userId,UserDisplay);

*console*.log(`user Id : ${userId}`)

*console*.log("After");

*function* UserDisplay(*user*){

*console*.log(user);

        GitRepository(user.userName,RepositoriesDisplay);

    }

*function* RepositoriesDisplay(*Repositories*){

*console*.log(Repositories);

}

* Meant say like we can use function name instead Providing anonyms definition and we provide function definition separately as shown above.
* Complete Modified Code

*console*.log("Before");

*let* userId="NI22";

*let* users=Timer(userId,UserDisplay);

*console*.log(`user Id : ${userId}`)

*console*.log("After");

*function* UserDisplay(*user*){

*console*.log(user);

        GitRepository(user.userName,RepositoriesDisplay);

    }

*function* RepositoriesDisplay(*Repositories*){

*console*.log(Repositories);

}

*function* Timer(*userId*, *callback*)

{

    setTimeout(()*=>*{

               callback({userId:userId ,userName:"Naveen"});

    },2000);

}

*function* GitRepository(*userName*,*callback*)

{

    setTimeout(()*=>*{callback(['repo1','repo2','repo3'])},2000)

}

**Promises**

*let* userId=1;

*let* p = new Promise((*resolve*, *reject*)*=>*

{

    if(userId===1)

    {

        setTimeout(()*=>*{*console*.log("Message")},2000);

    }

    else

    {

        reject(new Error("Message..of Error"));

    }

});

*console*.log("Before");

p

.then(*result* *=>* *console*.log(result))

.catch(*error* *=>* *console*.log("error",error.message));

*console*.log("After");

* Promises is also one of Pattern to handle Async Code.
* Promises Object can be of three state
  + Pending (Initial state before Operation is completed)
  + Resolve
  + Reject
* Here Async code need to be displayed inside the Promises() which takes 2 parameters one is resolve, reject (Also you can give either one but at least one should Present)
  + Resolve is functions used to send the successful result to handler
  + Reject function used to indicate we have some error so we can send Error messages to handler.
* Handler can be of two
  + Then()🡪 which take a function as a parameter which act as Successful result handler
  + Catch() which takes a function as a parameter which act as Error Handers
  + In Error Object message used to determine the error message.
  + Either catch or then will execute both won’t.
* Code

*console*.log("Before");

*let* userId="NI22";

//Promises Handlers

Timer(userId).then(*user=>*GitRepository(user.userName))

.then(*repo=>* *console*.log(repo));

*function* Timer(*userId*)

{

    return new Promise((*resolve*,*reject*)*=>*{

        setTimeout(()*=>*{

            resolve({userId:userId ,userName:"Naveen"});

 },2000);

    });

}

*function* GitRepository(*userName*)

{

    return new Promise((*resolve*,*reject*)*=>*{

        setTimeout(()*=>*{resolve(['repo1','repo2','repo3'])},2000);

    });

}

Parallel Promises

* Here we are always deals with one Thread, So all the Async operation will be kicked Off at same time.
* When first async Operations done with its job then thread will released, so second will be started will not be wait until the values are got for the first async Operations.
* When we get the result will be available as an array. Each element in the array will holds values of Each Promises
* Any one Promises got error (Rejected), only error will be displayed, even though others are successfully got its value.

*var* p1= new Promise((*resolve*)*=>*

{

    setTimeout(()*=>*{

        resolve(11);

    },

    2000);

});

*var* p2= new Promise((*resolve*)*=>*

{

    setTimeout(()*=>*{

        resolve(21);

    },

    2000);});

*Promise*.all([p1,p2])

    .then(*result=>* *console*.log(result));

Promise.all will consolidate other Promises, Promises need to be given within array

Output will be [11,21]

This will wait unit all the Promises are completed

Suppose need not wait until all is completed need to Perform the Operation when first Promises is completed, then

*Promise*.race([p1,p2])

    .then(*result=>* *console*.log(result));

This will take only which Promises completed first. And do the Operation

**Await and Async (\*new Feature)**

* Used to write async code like sync code
* Handler need be Prefixed with await no change in Async code means then/catch will be replaced by await, remains as same as Promise
* Always await need to inside a function which is decorated with async modifier
* Async and await is build top of Promises

async *function* Operation()

{

*let* users= await Timer(userId);

*let* repo= await GitRepository(users.userName);

*console*.log(repo);

}

Operation();

*function* Timer(*userId*)

{

    return new Promise((*resolve*,*reject*)*=>*{

        setTimeout(()*=>*{

            resolve({userId:userId ,userName:"Naveen"});

 },2000);

    });

}

*function* GitRepository(*userName*)

{

    return new Promise((*resolve*,*reject*)*=>*{

        setTimeout(()*=>*{resolve(['repo1','repo2','repo3'])},2000);

    });

}

* While we are using Promises we had catch method to handle the rejected section but here we don’t have, instead we can use try ..catch block for this purposes.

async *function* Operation()

{

    try

    {

*let* users= await Timer(userId);

*let* repo= await GitRepository(users.userName);

*console*.log(repo);

    }

    catch(err)

    {

*console*.log(err.message);

    }

}

**Mongo Db**

* Still we are not using database for storing our data, while we are restarting the server our data will be get lost to avoid this we can store our data in database
* Most of the Node Application uses Mongo db for this Purpose.
* We have install package called mongoose (Provides simple API to work with mongo db)
* Installing : npm i mongoose

*var* mongoose = require('mongoose');

*var* mongoConnect = mongoose.connect("mongodb://localhost/checking")

    .then(()*=>console*.log("Connection is done"))

    .catch((*error*)*=>* *console*.error("failure ...to connect",error));

*var* mongoose = require('mongoose');

* including mongoose packages.

mongoose.connect("mongodb://localhost/checking")

    .then(()*=>console*.log("Connection is done"))

    .catch((*error*)*=>* *console*.error("failure ...to connect",error));

* Mongoose.connect() used to connect to mongo db
* Mongodb:// 🡪 indicates mongodb
* Localhost 🡪 local
* Checking 🡪 db name (if not present, mongo will creates its db while performing reading/writing operation).
* This function will return a Promises

**Schemas**

* Used to define a shape of a document in mongo db collection at Node Js itself

*var* mongoSchema= new mongoose.Schema({

    Name : *String*,

    Author: *String*,

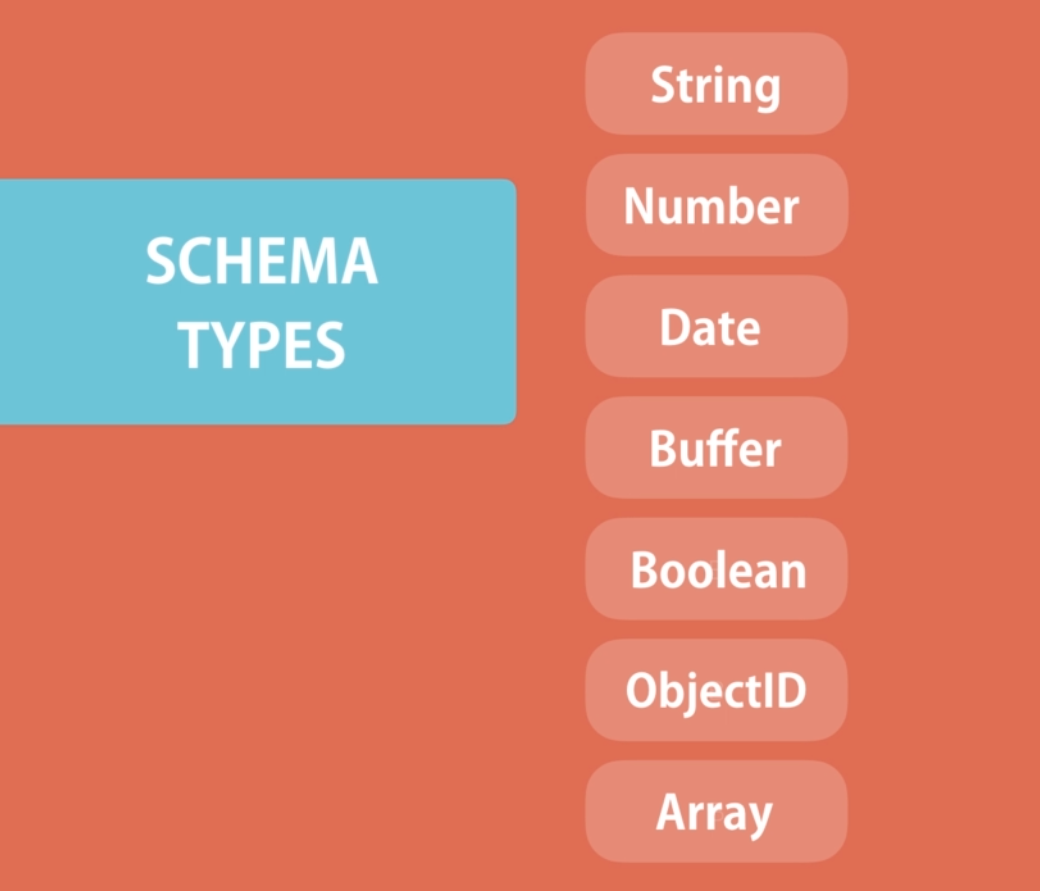
    tag:[*String*],

    isPublished: *Boolean*,

    date: {type:*Date*,default:*Date*.now}

});

* Schema is class which is used to create schema for mongo db , we uses Object for defining schema.
* The Properties such as **Name, Author** are normal string
* **Tag** is array
* **IsPublushised**  is Boolean
* **date** is date type which have default value as Date.now



* Buffer is used store Binary values
* ObjectId is used store unique identifier

*var* Course=mongoose.model('Course',mongoSchema);

Creating Class with Singular name of the collection in Mongodb

Courses 🡪 Course

And with schema

We can set the value as below,

*var* course= new Course({

    Name:"Node Js",

    Author:"Naveen",

    tag: ['Node Js Basics','Mongo db'],

    isPublished:true

});

**Storing to Mongo dB.**

await course.save();

will return a Promise which is async function which takes a long because it connecting to database.

*var* Course=mongoose.model('Course',mongoSchema);

async *function* DataStorage(){

    try {

*var* course= new Course({

    Name:"Node Js",

    Author:"Naveen",

    tag: ['Node Js Basics','Mongo db'],

    isPublished:true

});

*var* result =await course.save();

*console*.log(result)

    }

    catch(error)

    {

*console*.log(error);

    }

}

DataStorage();

Result will hold complete document along with object Id which is assigned in mongodb.

Querying Database

*var* Course=mongoose.model('Course',mongoSchema);

async *function* RetriveFromDataStorage()

{

*var* courses = await Course.find();

*console*.log(courses);

}

find() 🡪 will return all the documents in the Mongo dB

**Applying Condition**

*var* courses = await Course

.find({Author:"Naveen",isPublishe:true});

Filters are should be added in find method Parameter as an Object,

This record will display only record belong to Author is Naveen and isPublished is true

Applying limit Object to be returned

*var* courses = await Course

   .find({Author:"Naveen",isPublished:true})

   .limit(5);

This will return only the first 5 document that matches its condition,

More info …, <https://www.w3resource.com/mongodb/mongodb-skip-limit.php>

Sorting

*var* courses = await Course

   .find({Author:"Naveen",isPublished:true})

   .sort({Author:-1})

Sort Method will take Object as Parameter where we can provide one or more fields name, here Key as field name and value as 1 or -1 ,where 1 indicate Ascending Order, -1 indicate Descending Order.

More info <https://docs.mongodb.com/manual/reference/operator/aggregation/sort/>

Selecting Particular Values,

*var* courses = await Course

   .find({Author:"Naveen",isPublished:true})

   .select({Author:1,isPublished:1,\_id:0})

Select Method is used for this Purpose which will take Object as Parameter where we can provide one or more fields name, here Key as field name and value as 1 or -1 ,

where 1 indicate fields are need to be included, 0 indicate Field not to display \_id will be default included if not mentioned in the select to have control over we can like this.