**Node js**

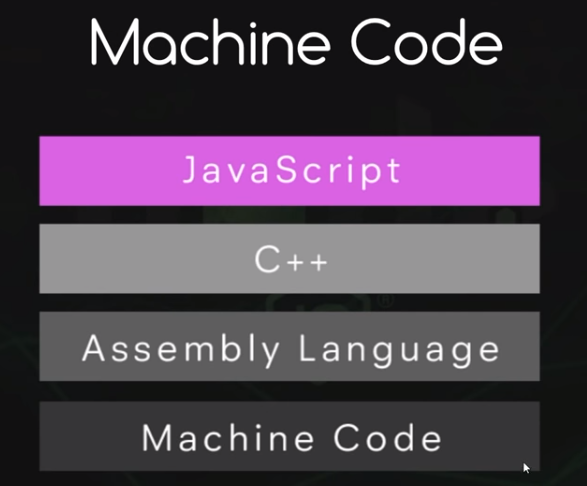
Node js is a platform to run javascript on the server/computer. If we want to create our own web application we no longer need to learn the additional programming languages like PHP etc, we can all do this is javascript via Nodejs.

That means node js gives us the capability of Read, delete, update files, easily communicate with the database and all the stuffs that you can expect from a server side programming language.

Why use Node Js??

* As it uses Javascript so no need to learn any programming language on the safer side.
* It is very fast as in runs on the **V8 engine\*** and uses **non-blocking\*** code.
* It has a huge ocean of open source packages(npm)
* Great for real time applications like chat utilizing the concept of web sockets.

**V8 engine\*:** When we write javascript, computer itself does not understand it. It takes a javasscript engine to convert javascript code into machine code, something a computer can understand so that it can be processed.



Node js itself is written in C++. It is written in C++ because it uses V8 engine created by google which is also written in C++.

**Global Objects:**

They are the global Objects which we can use in Node Js out of the box.

* Eg: **setTimeout(callback, delay[, arg][,…])** ,
* **console.log()**
* , **\_\_dirname**
* **\_\_filename** etc.

**Functions**

function hello() {

console.log("Hello");

}

hello();

var hello = function () { //annonymous function

console.log("Hello");

}

hello();

var hello = () => { //annonymous function

console.log("Hello");

}

hello();

var bye = function () {

console.log("Bye");

}

function callFn(fn) {

fn();

}

callFn(bye);

**Modules and require**

Suppose we have **counter.js**

var counter = someArray => {

return "Length of someArray is " + someArray.length;

}

**module.exports = counter;**

**app.js**

var counter = require('./counter');

console.log(counter(['one', 'two', 'three']))

**Q. What if we have multiple methods to export?**

var counter = (someArray) => {

console.log("Length of someArray is " + someArray.length);

}

var add = (a, b) => {

return `Sum of the two variables is ${a + b}`;

}

module.exports.counter = counter;

module.exports.add = add;

var some = require('./counter');

console.log(some.counter(['one', 'two', 'three']));

console.log(some.add(1, 2));

**We can also use:**

module.exports = {

counter: counter,

add: add

}

**Event Emitter**

var events = require('events'); // events is a inbuilt node module

var myEmitter = new events.EventEmitter();

myEmitter.on('someEvent', function (msg) {

console.log(msg);

});

myEmitter.emit('someEvent', 'the event was emitted');

var events = require('events'); // events is a inbuilt node module

var util = require('util');

var Person = function (name) {

this.name = name;

} // this is equivalent to class and Constructor

util.inherits(Person, events.EventEmitter);

var james = new Person('james');

var mary = new Person('mary');

var ryu = new Person('ryu');

var people = [james, mary, ryu];

people.forEach(function (person) {

person.on('speak', function (message) {

console.log(person.name + ' said: ' + message);

});

});

james.emit('speak', 'hey dudes');

mary.emit('speak', 'I am Marry');

**Reading and writing files**

var fs = require('fs');

// Synchronous way

var readMe = fs.readFileSync('readMe.txt', 'utf8');

fs.writeFileSync('writeMe.txt', readMe);

// Asynchronous way

fs.readFile('readMe.txt', 'utf8', function (err, data) {

fs.writeFile('writeMe.txt', data);

})

// CODE

TODO:

NOTE: Asynchronous code happens in the background while the code is also running.

**Create and remove directories**

var fs = require('fs');

// DELETE THE FILE

fs.unlink('myfile.txt');

// CREATING THE DIRECTORY

fs.mkdirSync('stuff');

// REMOVING THE DIRECTORY

fs.rmdirSync('stuff');

// ASYNCHRONOUS

// CREATING THE DIRECTORY

fs.mkdir('stuff', function () {

fs.readFile('readMe.txt', 'utf8', function (error, data) {

fs.writeFile('writeMe.txt', data)

})

});

// REMOVING THE DIRECTORY

fs.rmdirSync('stuff');

**Clients and Servers**

**Protocols:**

* A set of communication rules, that two sides agree to use while communicating.

Each computer or server is identified by its own IP address. So if we want to communicate between the two computers i.e, a client wants to make a request to the server then client needs to connect with the server’s IP address.

After connecting it opens a socket between the client and the server which is a channel through which the information can be send. The information which is sent is structured via different protocols for eg: HTTP, FTP and these protocols are like the different languages.

When the structure of the data/information has been decided(eg: http, FTP), they are sent through sockets between the two computers via protocol TCP.

**Creating a Server**

1. http module

* **var http = require('http');**

**var server = http.createServer();**

Above, http server is created

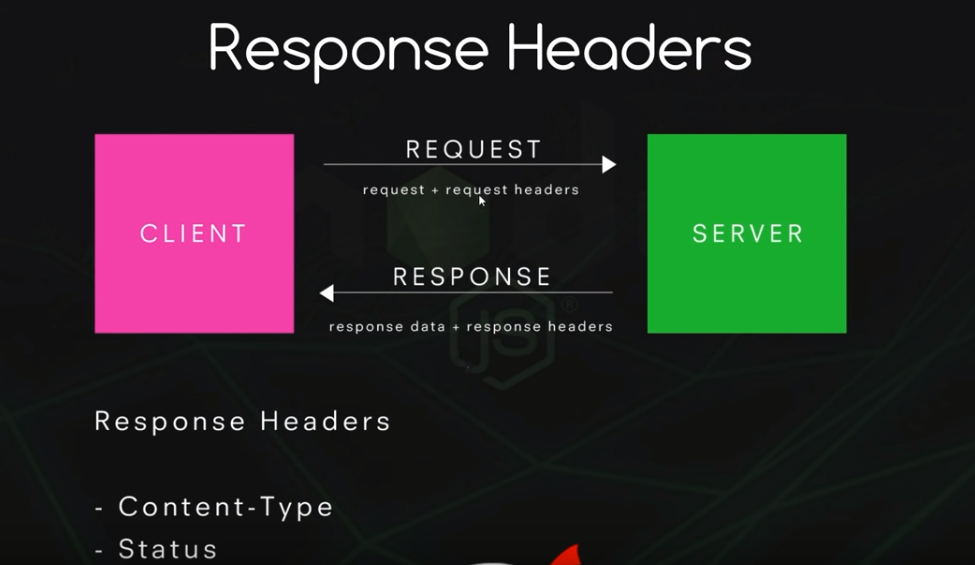
createServer method takes two parameters req and res. If we request something on this server then this function is fired.

**var server = http.createServer((request, response) => {**

**});**

request parameter has all the information about the request that has been made to it.

response parameter has all thr information which the server will send back to the client.



Here, request headers are the extra information that is sent along with the request.

var http = require('http');

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

response.writeHead(200, { 'Content-Type': 'text/plain' });

response.end('Hello World!');

});

server.listen('4000', 'localhost');

Here, the server is listening to the port: 4000 with the headers using: response.writeHead(200, { 'Content-Type': 'text/plain' });

So if the client request the port no : 4000, server responds with the status as 200 ok with the content-Type as 'text/plain'.

**request.url**

This property of request can get the url path which has been typed in browser/client.

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

console.log(request.url);

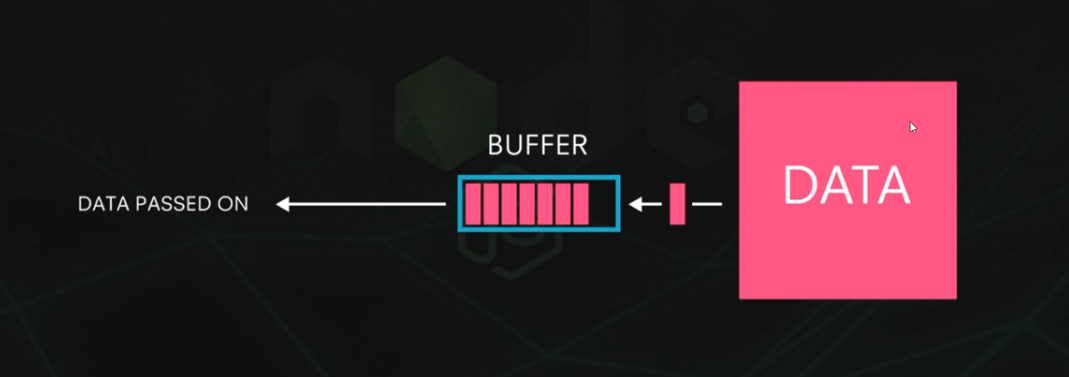
});

**Streams & Buffers**

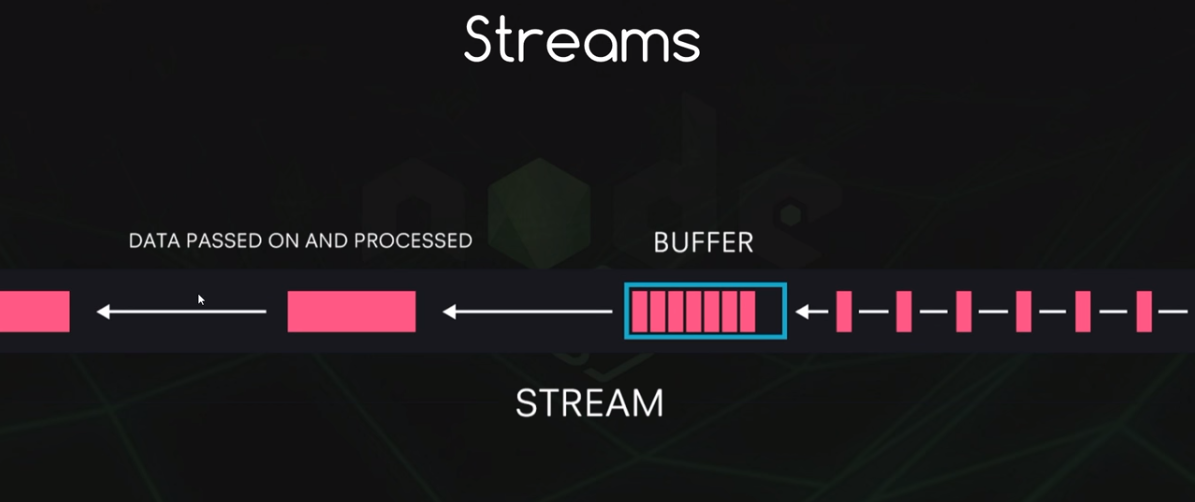
Buffer is a temporary storage spot for a chunk of data that is being transferred from one place to another.

Buffer is filled with data.

If we have a large amount of data and we want to move it from point A to point B then we can move it a little bit at a time by gathering a small amount of it in a buffer and then moving it on so we are transferring a small amount of data at a time.



**Streams**



**Types of streams:**

1. **Writable:** Write data to a Stream.
2. **Readable:** Read data from a Stream.
3. **Duplex:** can Read and Write to a Stream.

When we are sending a response to a client from NodeJs, we are sending a Client some data, so in this instant we are talking about writable Stream.

When Node is Reading the data then it is Readable Stream.

var fs = require('fs');

/\*\*

\* Here, the data under the readMe.txt will be passed into

\* the variable myReadStream via buffer chunk by chunk

\* createReadStream inherits from EventEmitter

\*/

var myReadStream = fs.createReadStream(\_\_dirname + '/readMe.txt');

/\*\*

\* myReadStream inherits from EvenEmitter

\* thus having on function and data Event

\*/

myReadStream.on('data', chunk => {

console.log('new chunk received');

console.log(chunk);

})

In the above the data under the file readMe.txt is read into the variable myReadStream chunk by chunk using the function createReadStream which inherits form EventEmitter.

**Output:**

new chunk received

<Buffer 57 68 61 74 20 41 72 65 20 41 72 74 69 63 6c 65 73 3f 0d 73...

>

new chunk received

<Buffer 69 63 6c 65 73 20 61 72 65 20 6f 6d 69 74 74 65 64 20 61 6c 74 6f...

>

new chunk received

<Buffer 2e 0d 0a 55 73 69 6e 67 20 41 72 74 69 63 6c 65 73 20 77 69 74 68...

>

We are seeing about output as a buffer because we have not set the encoding to utf8. If we want the text also we have to use:

var myReadStream = fs.createReadStream(\_\_dirname + '/readMe.txt', 'utf8');

Write

var fs = require('fs');

/\*\*

\* Here, the data under the readMe.txt will be passed in

\* the variable myReadStream via buffer in chunks

\* createReadStream inherits from EventEmitter

\*/

var myReadStream = fs.createReadStream(\_\_dirname + '/readMe.txt', 'utf8');

var myWriteStream = fs.createWriteStream(\_\_dirname + '/writMe.txt');

/\*\*

\* myReadStream inherits from EvenEmitter

\* thus having on function and data Event

\*/

myReadStream.on('data', chunk => {

myWriteStream.write(chunk);

})

**PIPES**

In the above when are reading from one file i.e, myReadStream.txt we are listening it and then passing it to myWriteStream like:

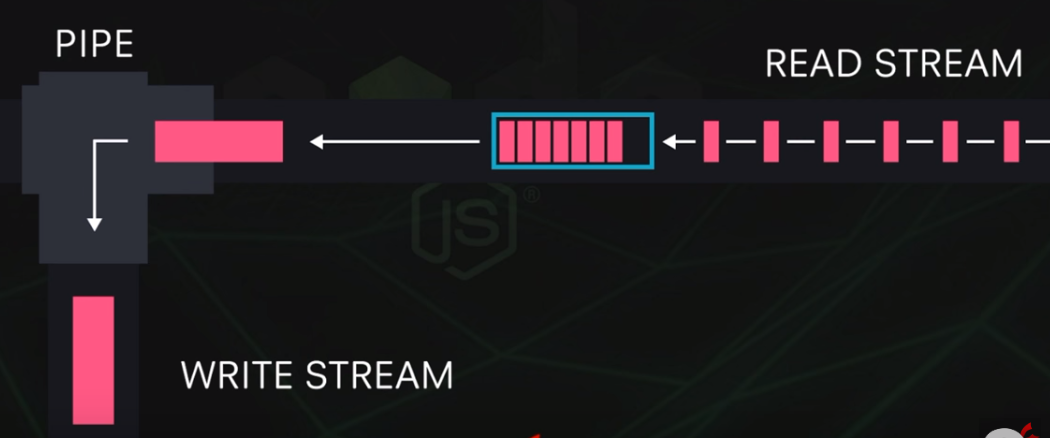
myReadStream.on('data', chunk => {

myWriteStream.write(chunk);

})

We can do this more elegently using pipe

myReadStream.pipe(myWriteStream);



var http = require('http');

var fs = require('fs');

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

response.writeHead(200, { 'Content-Type': 'text/plain' });

var myReadStream = fs.createReadStream(\_\_dirname + '/readMe.txt', 'utf8');

myReadStream.pipe(response);

});

server.listen('4000', 'localhost');

Here, NodeJs is sending the response to the browser. So we are sending plaintext to the browser.

We can send HTML to the browser same as aboove tweeking few things like text/plain to text/html and readMe.txt to index.html

var http = require('http');

var fs = require('fs');

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

response.writeHead(200, { 'Content-Type': 'text/html' });

var myReadStream = fs.createReadStream(\_\_dirname + '/index.html', 'utf8');

myReadStream.pipe(response);

});

server.listen('4000', 'localhost');

**JSON serving**

We cannot directly send JSON to the browser as end function of response expects String or Buffer.

So We use JSON.stringify

var http = require('http');

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

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

var jsonObj = {

name: 'Tanuj',

age: 26

}

response.end(JSON.stringify(jsonObj));

});

server.listen('4000', 'localhost');

**ROUTING using request.url**

var http = require('http');

var fs = require('fs');

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

console.log(request.url)

if (request.url === '/' || request.url === '/home') {

response.writeHead(200, { 'Content-Type': 'text/html' });

var myReadStream = fs.createReadStream(\_\_dirname + '/index.html', 'utf8');

myReadStream.pipe(response);

}

});

server.listen('4000', 'localhost');

var http = require('http');

var fs = require('fs');

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

if (request.url === '/' || request.url === '/home') {

response.writeHead(200, { 'Content-Type': 'text/html' });

var myReadStream = fs.createReadStream(\_\_dirname + '/index.html');

myReadStream.pipe(response);

}else{

response.writeHead(404, { 'Content-Type': 'text/plain' });

response.end('oops, this url not found')

}

});

server.listen('4000', 'localhost');

Here, if there are so many addresses then it becomes very tedious to apply if else case of every link. For this we have express server.

**NPM:**

NPM is a nope package manager which comes along when you install NODEJS. It is used to install third party packages or modules that we can apply to our application to make it more easier and readable. So, it is a huge ecosystem of third party packages.

express module help us in routing, templating etc.

We can also publish our own packages.

**Express:**

It has very easy and flexible routing system.

Integrates with many templating engines.

Contains a middleware framework.

const express = require('express');

const app = express();

app.listen(5000);

Here, By using require(‘express’), we are exposing a function exported by express package. So, we have to fire this package by doing **express();**

For GET, POST, DELETE etc we can use:

app.get(‘route’, fn)

app.post(‘route’, fn)

app.delete(‘route’, fn)

var express = require('express');

var app = express();

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

res.send('this is a homepage');

})

app.listen('4000');

We have not set Content-Type in express server as it it clever enough to know that.

**Express Route Params:**

/home/:id

Here **:id** is a param that is received dynamically from the browser.

var express = require('express');

var app = express();

app.get('/home/:id', (req, res)=>{

res.send('this is a homepage and id is'+ req.params.id);

})

app.listen('4000');

If you want to display html page in browser using express:

var express = require('express');

var app = express();

app.get('/home/:id', (req, res) => {

res.sendFile(\_\_dirname + 'index.html');

})

app.listen('4000');

**Template Engines**

If you want to inject dynamic content in the Page, Template Engine comes into the picture. If you are sending something in url and you have to bind this dynamic content into the html page, we use Template Engine.

ejs: It is a light weight js template engine.

**npm install ejs –save**

var express = require('express');

var app = express();

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

/\*\*

\* profile is profile.ejs

\*/

app.get('/home/:id', (req, res) => {

res.render('profile', { person: 'tanuj' });

})

app.listen('4000');

**profile.ejs**

<!DOCTYPE html>

<html>

<head>

<title>Chat Application</title>

</head>

<body>

<%= person %>

</body>

</html>