# Introduction

## What is Node.js?

* A platformer which allows us to run JavaScript on a computer/server
* Read, delete and update files
* Easily communicate with a database

## Why use Node.js?

* It uses JavaScript
* Very fast
* Huge ecosystems of open source packages
* Great for real-time services

## What I need to know?

* HTML
* JavaScript

# The V8 Engine

## JavaScript Engines

* Computers don’t understand JavaScript
* A Javascript engine takes JavaScript, and converts it into something it does understand, like machine code

1. JavaScript
2. C++
3. Assembly Language
4. Machine Code

* Node.js is written in C++
* At the heart of Node.js is the V8 Engine
* The V8 engine converts our JS into machine code

# The Global Object

* Global object - an object that always exists in the global scope. We can use them anywhere in the file.
  + In JavaScript, there's always a global object defined.
  + In a web browser, when scripts create global variables defined with the var keyword, they're created as members of the global object.
  + Global object functions and variables documentation: <https://nodejs.org/dist/latest-v18.x/docs/api/globals.html>
* A global object in node is called a global
  + This allows us to use methods straight out of the box in Node.js

# Function Expressions

* Function Expression - a function that is created as a part of an expression
* Function Declaration - a function that is declared as a separate statement in the main code flow

### Function Expression Example

| var sayBye = function() {  console.log('Bye')  }  sayBye() |
| --- |

### Function Declaration Example

| function sayHi(){  console.log("Hi")  }  sayHi() |
| --- |

### Call Function Example

* A function can be used as an argument/parameter

| function callFunction(func){  func()  }  var sayBye = function() {  console.log('Bye')  }  callFunction(sayBye) |
| --- |

# Modules and require()

* Modules - a software design technique that emphasizes separating the functionality of a program into independent, interchangeable modules, such that each contains everything necessary to execute only one aspect of the desired functionality
  + As of now all it is is just another JavaScript file
* Require() - is used to load and cache JavaScript modules
  + Reads a JavaScript file, executes it, and then proceeds to return the export object. require() not only allows adding built-in core NodeJS modules but also community-based and local modules.

## Require example

* The require needs a path to the file

| var counter = require('./stream')  console.log(counter(['Dakota', 'Hat', 'Doom', 'Phone', 'JavaScript'])) |
| --- |

## Module example (stream.js)

* Modules.exports is needed for export the function/method

| var counter = function(arr){  return 'There are ' + arr.length + ' elements in this array'  }  module.exports = counter |
| --- |

# Module Patterns

## Require example

| var stuff = require('./stuff')  console.log(stuff.counter(['Dakota', 'Hat', 'Doom', 'Phone', 'JavaScript']))  console.log(stuff.adder(1, 5))  console.log(stuff.pi)  console.log(stuff.adder(stuff.pi, 5)) |
| --- |

## Export module example 1

| var counter = function(arr){  return 'There are ' + arr.length + ' elements in this array'  }  var adder = function(a, b){  return `The sum of the two numbers is ${a + b}`  }  var pi = 3.14159  module.exports.counter = counter  module.exports.adder = adder  module.exports.pi = pi |
| --- |

## Export module example 2

| module.exports.counter = function(arr){  return 'There are ' + arr.length + ' elements in this array'  }  module.exports.adder = function(a, b){  return `The sum of the two numbers is ${a + b}`  }  module.exports.pi = 3.14159 |
| --- |

## Export module example 3

| var counter = function(arr){  return 'There are ' + arr.length + ' elements in this array'  }  var adder = function(a, b){  return `The sum of the two numbers is ${a + b}`  }  var pi = 3.14159  module.exports = {  counter: counter,  adder: adder,  pi: pi  } |
| --- |

# The Node Event Emitter

## Emitter example 1

| var events = require('events') //events is a core module  //element.on('click', function(){}) //when this element finds this click event the function will activate  var myEmitter = new events.EventEmitter() // this is our own event emitter object, basically event constructor  myEmitter.on('someEvent', function(mssg){ //event and function with parameter  console.log(mssg)  })  myEmitter.emit('someEvent', 'the event was emitted') //event and argument |
| --- |

## Emitter example 2

| var events = require('events') //events is a core module  var util = require('util') //util is a core module  var Person = function(name){ //we want this to inherit the event emitter  this.name = name  }  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){ //we're taking each person and giving them an event listener  person.on('speak', function(mssg){  console.log(person.name + " said: " + mssg)  })  })  james.emit('speak', 'Hey dudes')  mary.emit('speak', 'What\'s up?')  ryu.emit('speak', 'Chicken sounds good right now') |
| --- |

# Reading & Writing Files (fs)

## Synchronous

### readFileSync

| var fs = require('fs') //fs is a core module  var readMe = fs.readFileSync('files/lorem.txt', 'utf8')  //file location (if it's in the same folder you don't need to specify), character encoding  //this will write the file before it runs any code after  console.log(readMe) |
| --- |

### writeFileSync

| var fs = require('fs') //fs is a core module  var readMe = fs.readFileSync('files/lorem.txt', 'utf8')  //file location (if it's in the same folder you don't need to specify), character encoding  //this will write the file before it runs any code after  fs.writeFileSync('files/new-lorem.txt', readMe)  //file location, what we want to write  //this will write the file before it runs any code after |
| --- |

## Asynchronous

| var fs = require('fs') //fs is a core module  fs.readFile('lorem.txt', 'utf8', function(err, data){  fs.writeFileSync('new-lorem.txt', data)  })  //file location (if it's in the same folder you don't need to specify), character encoding, call back function  //we're not blocking the code, as in the code below will fire while this is reading the file  //we're going to write the file once this is fully read |
| --- |

# Creating and removing Directories

## Deleting files

* I suggest using unlinkSync
* This will only delete if the file exists, if it doesn’t it will throw you an error

| var fs = require('fs') //fs is a core module  fs.unlinkSync('files/new-lorem.txt') |
| --- |

## Make directories sync

| var fs = require('fs') //fs is a core module  fs.mkdirSync('stuff') //this will create a directory named stuff |
| --- |

## Remove directories sync

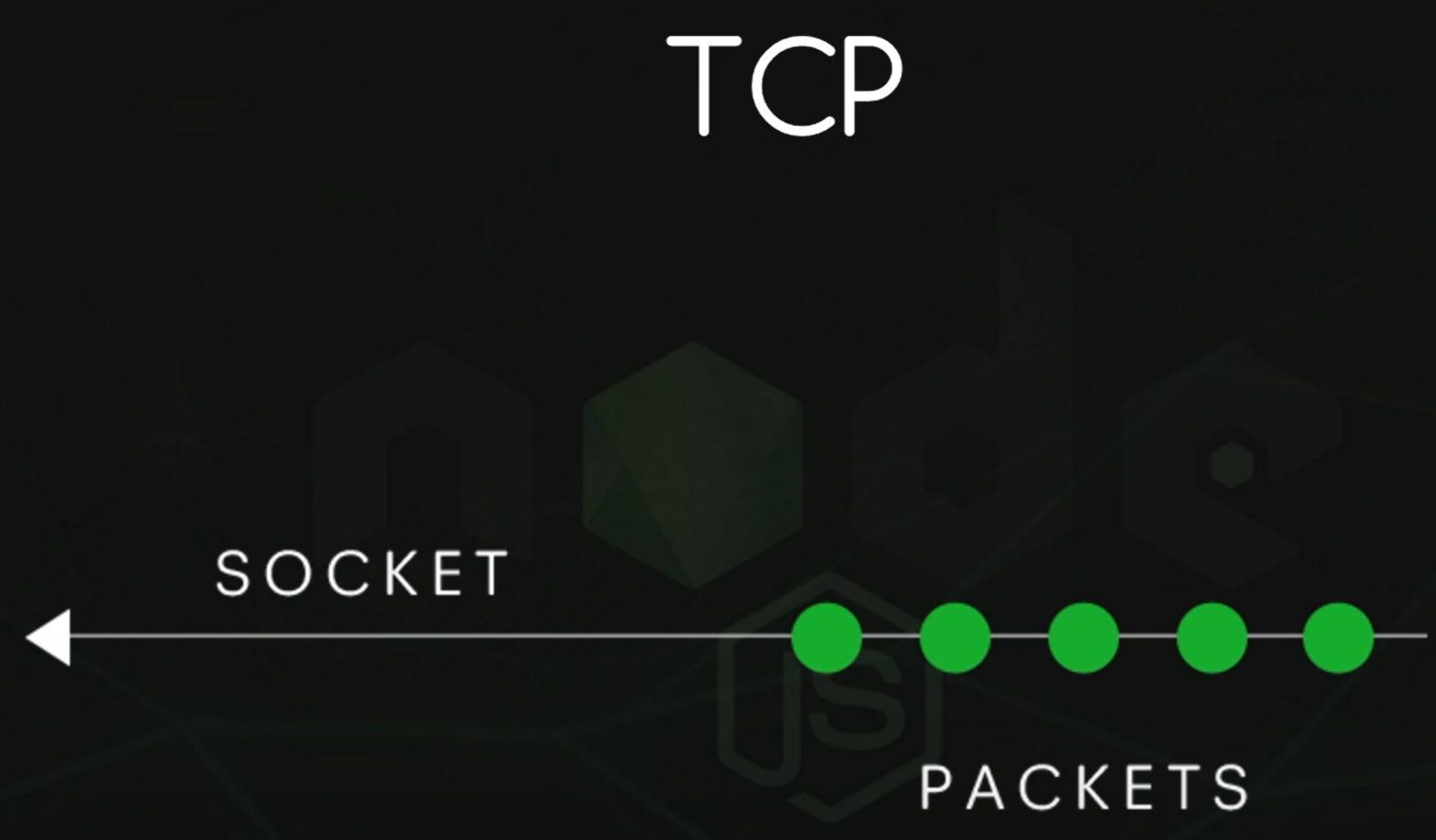
| var fs = require('fs') //fs is a core module  fs.rmdirSync('stuff') //this will remove a directory named stuff |
| --- |

# Clients and Servers



* Protocols - a set of communication rules, that two sides agree to use when communicating
  + The client and the server



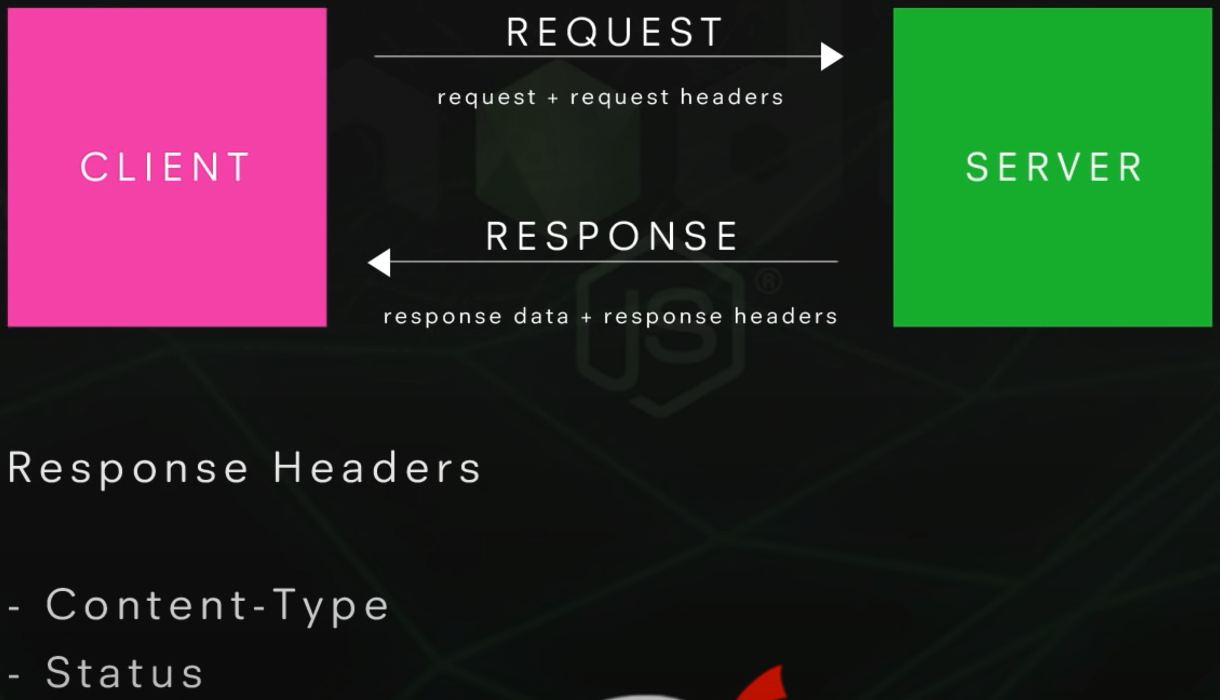


* TCP - Transmission Control Protocol (TCP) is a standard that defines how to establish and maintain a network conversation by which applications can exchange data.
  + TCP works with the Internet Protocol (IP), which defines how computers send packets of data to each other.
* Ports - a program running on a computer can listen for requests sent to a particular port number
  + E.g. 172.24.86.76:3000

# Creating a Server

| var http = require('http')  //these two line simply make a server but we also need a way for the client to talk to the server  var server = http.createServer(function(request, response){    }) |
| --- |

## Response Headers



* You can think of headers as a extra information about the request or the response
* In the browser bar type “127.0.0.1:3000” without the quotes
  + As long as the ip and port number are there you can put anything after it and this will still show up

| var http = require('http')  //these two line simply make a server but we also need a way for the client to talk to the server  var server = http.createServer(function(request, response){  console.log('request was made: ' + request.url) //this returns the url of the request  response.writeHead(200, {'Content-Type': 'text/plain'}) //status, object: value  response.end('Hey Dudes') //end the response  })  server.listen(3000, '127.0.0.1')  //port number, ip address (local host: 127.0.0.1)  //when we get a request to this port it will fire the createServer function  console.log('This is listening to port 3000') |
| --- |

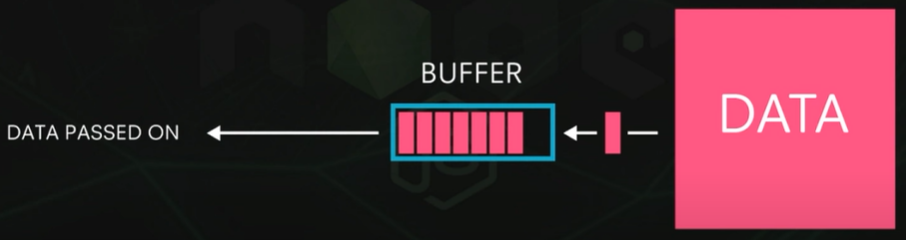
### Output

* I changed the url after the port number and the request shows what was updated

| PS C:\Users\kotag\OneDrive\Desktop\javascript testing> node server  This is listening to port 3000  request was made: /  request was made: /favicon.ico  request was made: /testing  request was made: /favicon.ico  request was made: /testing2  request was made: /favicon.ico |
| --- |

# Streams and Buffers

* Buffer - temporary storage spot for a chunk of data that is being transferred from one place to another
  + The buffer is filled with data, then passed along
  + Transfers small chunks of data at a time



* Stream - a stream of data that flows over time from one place to another
  + Data source flows to the buffer, the data in the buffer is then processed, afterwards the data would go to the client
    - Think about streaming movies online, chunks of the movie are sent over instead of waiting a while for the movie to be fully loaded.
    - We can consume that data right when it arrives



# Readable Streams

* Writable streams - allow node js it write data to a stream
* Readable streams - allow node js to read data from a stream
  + More efficient than just waiting for a whole file to be loaded
* Duplex - can read and write to a stream

| var fs = require('fs')  var myReadStream = fs.createReadStream(\_\_dirname + '/files/lorem.txt')  //creates a readable stream which we can read data from  //so we read the file,  //it fills up the buffer,  //the buffer passes that chunk on,  //every time the chunk is passed it recognizes it because we are listening to that data event  myReadStream.on('data', function(chunk){  console.log('new chunk received:')  console.log(chunk)  }) |
| --- |

### Output

| PS C:\Users\kotag\OneDrive\Desktop\javascript testing> node server  new chunk received:  <Buffer 4c 6f 72 65 6d 20 69 70 73 75 6d 20 64 6f 6c 6f 72 20 73 69 74 20 61 6d 65 74 2c 20 63 6f 6e 73 65 63 74 65 74 75 72 20 61 64 69 70 69 73 63 69 6e 67 ... 65486 more bytes>  new chunk received:  <Buffer 73 74 20 70 65 6c 6c 65 6e 74 65 73 71 75 65 20 65 6c 69 74 20 75 6c 6c 61 6d 63 6f 72 70 65 72 2e 20 41 6d 65 74 20 63 6f 6d 6d 6f 64 6f 20 6e 75 6c ... 9695 more bytes> |
| --- |

### Also

| var fs = require('fs')  var myReadStream = fs.createReadStream(\_\_dirname + '/files/lorem.txt', 'utf8')  //creates a readable stream which we can read data from  //so we read the file,  //it fills up the buffer,  //the buffer passes that chunk on,  //every time the chunk is passed it recognizes it because we are listening to that data event  myReadStream.on('data', function(chunk){  console.log('new chunk received:')  console.log(chunk)  }) |
| --- |

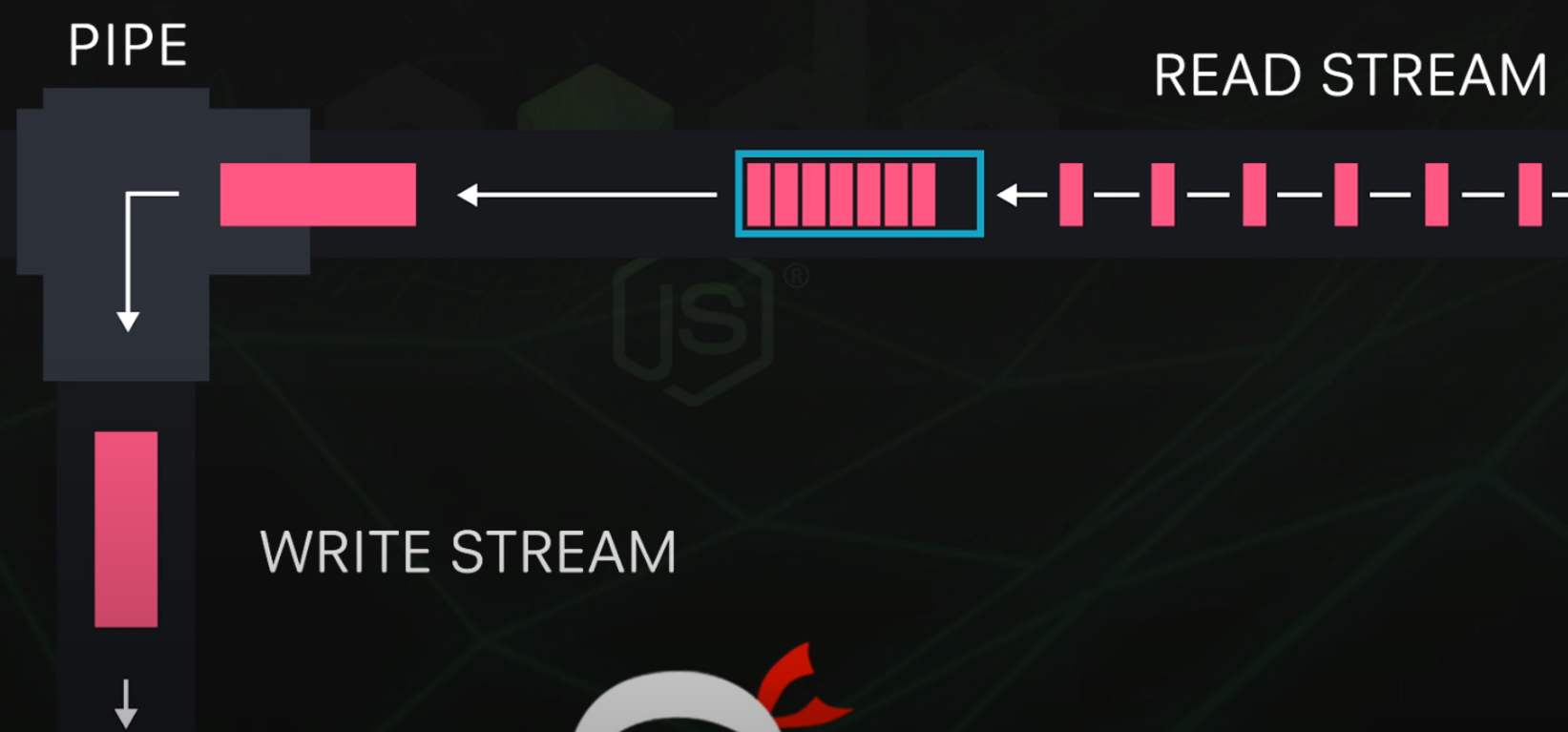
* This will print out the 100 paragraphs of Lorem Ipsum

# Writable Streams

| var fs = require('fs')  var myReadStream = fs.createReadStream(\_\_dirname + '/files/lorem.txt', 'utf8')  var myWriteStream = fs.createWriteStream(\_\_dirname + '/files/new-lorem.txt')  //this will create a file in files  myReadStream.on('data', function(chunk){  console.log('new chunk received:')  myWriteStream.write(chunk) //this writes the chunk in the file  }) |
| --- |

* As this runs the file ‘new-lorem’ is being written

# Pipes



* Pipes helps us do the exact same thing as read and write streams but faster
  + Taking data from a read stream and pipe it to a write stream
  + Instead of listening for the on data event when we receive a chunk of data the pipe automatically does it for us, piping it down to the write stream
  + We don’t have to manually do read write streams

| var fs = require('fs')  var myReadStream = fs.createReadStream(\_\_dirname + '/files/lorem.txt', 'utf8')  var myWriteStream = fs.createWriteStream(\_\_dirname + '/files/new-lorem.txt')  //we can only use this method on a readable stream, the argument would be the file we write to  myReadStream.pipe(myWriteStream) |
| --- |

## Changing the webpage

* This changes the webpage from before but now shows the entire Lorem Ipsum instead

| var http = require('http')  var fs = require('fs')  var server = http.createServer(function(request, response){ //the response object is a writable stream  console.log('request was made: ' + request.url)  response.writeHead(200, {'Content-Type': 'text/plain'}) //we are reading plain text  var myReadStream = fs.createReadStream(\_\_dirname + '/files/lorem.txt', 'utf8') //reads the content of the file  myReadStream.pipe(response) //this ends the response, when ever we get data chunks the page will change  })  server.listen(3000, '127.0.0.1')  console.log('This is listening to port 3000') |
| --- |

# Serving HTML Pages

## server.js

| var http = require('http')  var fs = require('fs')  var server = http.createServer(function(request, response){  console.log('request was made: ' + request.url)  response.writeHead(200, {'Content-Type': 'text/html'}) //we are reading html text  var myReadStream = fs.createReadStream(\_\_dirname + '/index.html', 'utf8') //reads the content of the file  myReadStream.pipe(response)  })  server.listen(3000, '127.0.0.1')  console.log('This is listening to port 3000')  //when you run this 127.0.0.1:3000 will now show the website |
| --- |

## index.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">  <style>  body{background: skyblue; font-family: Verdana, Geneva, Tahoma, sans-serif; color: white; padding: 30px;}  h1{font-size: 48px; text-transform: uppercase; letter-spacing: 2px; text-align: center;}  p{font-size: 16px; text-align: center;}  </style>  <title>Document</title>  </head>  <body>  <h1>This is a test site for Node.js.</h1>  <p>Purely for testing purposes.</p>  </body>  </html> |
| --- |

# Serving JSON

| var http = require('http')  var fs = require('fs')  var server = http.createServer(function(request, response){  console.log('request was made: ' + request.url)  response.writeHead(200, {'Content-Type': 'application/json'})  //we are reading application JSON  var myObj = { //created an object for the response to end and put in the browser  name: 'Ryu',  job: 'Ninja',  age: 29  }  response.end(JSON.stringify(myObj))  //makes the object into a string so it can print on the browser  })  server.listen(3000, '127.0.0.1')  console.log('This is listening to port 3000') |
| --- |

# Basic Routing

## server.js

| var http = require('http')  var fs = require('fs')  var server = http.createServer(function(request, response){  console.log('request was made: ' + request.url)  //if the user is at /home or just / we want to send them to the index.html file  //if we put anything else the browser will just look for it even if the html file doesn't exist  if(request.url === '/home' || request.url === '/'){  response.writeHead(200, {'Content-Type': 'text/html'})  fs.createReadStream(\_\_dirname + '/index.html').pipe(response)  }  //if the user is at /contact we want to send them to the contact.html file  else if(request.url === '/contact'){  response.writeHead(200, {'Content-Type': 'text/html'})  fs.createReadStream(\_\_dirname + '/contact.html').pipe(response)  }  //if the user puts /api we want to show the JSON  else if(request.url === '/api/coders'){  //in the real world you would get this information from a database  var coders = [{name: 'Dak', age: 20}, {name: 'Walk', age: 17}]  response.writeHead(200, {'Content-Type': 'application/json'})  response.end(JSON.stringify(coders))  }  //if the site doesn't exist  else {  response.writeHead(404, {'Content-Type': 'text/html'})  fs.createReadStream(\_\_dirname + '/404.html').pipe(response)  }  })  server.listen(3000, '127.0.0.1')  console.log('This is listening to port 3000') |
| --- |

## contact.html

| <body>  <h1>This is a test site for Node.js. Contact.HTML</h1>  <p>Purely for testing purposes.</p>  </body> |
| --- |

## 404.html

| <body>  <h1>404 oops!</h1>  <p>The page you're looking for doesn't exist.</p>  </body> |
| --- |

# The Node Package Manager (npm)

* npmjs.org has a bunch of packages that you can add to your js library through the terminal. I downloaded the express package by pasting ‘npm install express’ in the bottom terminal.
* To uninstall files type ‘npm install express’ in the bottom terminal and they should all go away

# Package.JSON

* Type ‘npm init’ and it will ask you a series of questions most of them just press enter, if something isn’t right type what needs to be changed

## package.json file

| {  "scripts": {  "start": "node server.js"  },  "name": "javascript-testing",  "version": "1.0.0",  "description": "a little test app",  "main": "server.js",  "devDependencies": {},  "keywords": [  "fun",  "coders"  ],  "author": "me",  "license": "ISC"  } |
| --- |

* Now if we install express with ‘ -save’ the json file will update and show dependencies

| {  "scripts": {  "start": "node server.js"  },  "name": "javascript-testing",  "version": "1.0.0",  "description": "a little test app",  "main": "server.js",  "keywords": [  "fun",  "coders"  ],  "author": "me",  "license": "ISC",  "dependencies": {  "express": "^4.18.2"  }  } |
| --- |

* If we try to uninstall express without ‘ -save’ package.json file won’t update even though we uninstalled the package
  + The dependences will still be listed, this tells us a couple of things:
    - If another developer were to use this the .json file would tell them what needs to be installed
    - If you want to update the json file ‘ -save’ is required
    - To install all of the packages in dependencies, in the terminal just type ‘npm install’

# Installing Nodemon

* Nodemon monitors your application files so that when it is running in a browser we're listening to the server. If we make a change in one of the application call files, like server.js, then we can monitor that and when we save that file it restarts the server automatically.
  + We don’t have to go back and start canceling out of the process then start running the server.js through node again. We can basically refresh the browser and get the new change
  + npm install -g nodemon
    - -g means we’re going to install it globally (no matter what application we work with on our computer we can use Nodemon)
  + To run it we type: nodemon server.js
    - There might be a powershell execution policy that needs to be updated
    - <https://www.youtube.com/watch?v=Q2uLUuq0Ft4>
  + So if we make any changes to the server.js file we the website will change automatically without us needing to cancel the file

# Introduction to Express

* Express - easy and flexible routing system
  + Integrates with many templating engines
  + Contains a middleware framework
  + In the terminal type npm install express -save
    - This will show up in the node modules folder

## HTTP Methods

* HTTP Methods - are the kind of requests that we make
  + GET - what we made when we type a URL into an address bar
    - app.get(‘route’, fn)
  + POST - when we post some data to a server from a web form
    - app.post(‘route’, fn)
  + DELETE -
    - app.delete(‘route’, fn)
  + PUT -

| var express = require('express')  var app = express() //this allows the app var to have access to all of express functions (HTTP Methods)  //when the user goes to 127.0.0.1:3000/ this function will run  app.get('/', function(request, response){  response.send('this is the homepage')  })  //when the user goes to 127.0.0.1:3000/contact this function will run  app.get('/contact', function(request, response){  response.send('this is the contact page')  })  app.listen(3000) //listening to port 3000 |
| --- |

# Route Parameters

| var express = require('express')  var app = express()  app.get('/', function(request, response){  response.send('this is the homepage')  })  app.get('/contact', function(request, response){  response.send('this is the contact page')  })  //it could be /profile/:123 or /profile/:456 could pertain to a person in a database query  app.get('/profile/:id', function(request, response){  //this will print the id portion of /profile/:id  response.send('You requested to see profile id ' + request.params.id)  })  app.listen(3000) |
| --- |

# Templating Engines

* With template engines we can embed data and Javascript code into our html files so we can then inject this dynamic content into the file which we return to the client or browser
  + We will use ejs which is a lightweight template engine
  + npm install ejs -save
* We create a folder called views and created a file in that folder called name profile.ejs which is now our ejs template

### profile.ejs

| <body>  <h1>Welcome to the profile of <%= person %></h1>  <p><strong>Age: </strong><%= data.age %></p>  <p><strong>Job: </strong><%= data.job %></p>  </body> |
| --- |

### server.js

| var express = require('express')  var app = express()  //what we want to set and which engine we want to use  app.set('view engine', 'ejs')  app.get('/', function(request, response){  //sends back the index.html page  response.sendFile(\_\_dirname + 'index.html')  })  app.get('/contact', function(request, response){  //sends back the contact.html page  response.sendFile(\_\_dirname + '/contact.html')  })  app.get('/profile/:id', function(request, response){  //dummy data  var data = {age: 29, job: 'programer'}  //when we put our profile id in the top bar this will read the profile.ejs file  //ejs file, property (we are passing data through)  response.render('profile', {person: request.params.id, data: data})  })  app.listen(3000) |
| --- |

* When outputting data in html we need <%= data %>
* When we are outputting javascript in html we need <% function(){} %>

### Added data in server.js

| var express = require('express')  var app = express()  app.set('view engine', 'ejs')  app.get('/', function(request, response){  response.sendFile(\_\_dirname + 'index.html')  })  app.get('/contact', function(request, response){  response.sendFile(\_\_dirname + '/contact.html')  })  app.get('/profile/:id', function(request, response){  var data = {age: 29, job: 'programer', hobbies: ['Sleeping', 'Eating', 'DnD']}  response.render('profile', {person: request.params.id, data: data})  })  app.listen(3000) |
| --- |

### profile.ejs

| <body>  <h1>Welcome to the profile of <%= person %></h1>  <p><strong>Age: </strong><%= data.age %></p>  <p><strong>Job: </strong><%= data.job %></p>  <h2>Hobbies</h2>  <ul>  <% data.hobbies.forEach(function(item){ %>  <li><%= item %></li>  <% }); %>  </ul>  </body> |
| --- |

# Partial Templates

* Template partials are small bits of reusable template or tag parts. You could create a Template partial for any number of purposes, anywhere that you need to reuse a small portion of a template, including partial or complete tags, other variables, etc.
* When doing this in the ejs use this to include the partials
  + <%- include ("partials/nav.ejs") %>
* I also made index and contact into ejs files as well

### profile.ejs

| <style>  body{background: skyblue; font-family: Verdana, Geneva, Tahoma, sans-serif; color: white; padding: 30px;}  h1{font-size: 48px; text-transform: uppercase; letter-spacing: 2px; text-align: center;}  h2{font-size: 30px; text-transform: uppercase; letter-spacing: 2px; text-align: center;}  p, li{font-size: 16px; text-align: center;}  ul{list-style-type: none; padding: 0;}  </style>  <title>Document</title>  </head>  <body>  <%- include ("partials/nav.ejs") %>  <h1>Welcome to the profile of <%= person %></h1>  <p><strong>Age: </strong><%= data.age %></p>  <p><strong>Job: </strong><%= data.job %></p>  <h2>Hobbies</h2>  <ul>  <% data.hobbies.forEach(function(item){ %>  <li><%= item %></li>  <% }); %>  </ul>  </body> |
| --- |

### server.js

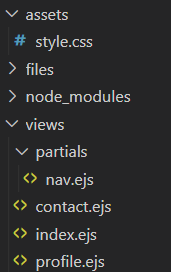
| var express = require('express')  var app = express()  app.set('view engine', 'ejs')  app.get('/', function(request, response){  response.render('index') //view name  })  app.get('/contact', function(request, response){  response.render('contact') //view name  })  app.get('/profile/:id', function(request, response){  var data = {age: 29, job: 'programer', hobbies: ['Sleeping', 'Eating', 'DnD']}  response.render('profile', {person: request.params.id, data: data})  })  app.listen(3000) |
| --- |

### partials/nav.ejs

| <nav>  <ul>  <li><a href="/">Home</a></li>  <li><a href="/contact">Contact</a></li>  </ul>  </nav> |
| --- |

# Middleware & Static files

* Static files - files that do not change
* Middleware - a type of computer software that provides services to software applications beyond those available from the operating system. It can be described as "software glue"
  + In our case its the code that runs between the request and the response
* I put the style part of index in a css style sheet called style.css inside a folder called assets



### server.js

| var express = require('express')  var app = express()  app.set('view engine', 'ejs')  //middleware  //route, fountion(request, response, next)  //- if we leave the route blank or / it will route to anywhere  //- the next parameter tells us that we finish our middleware now go to the next set of middleware  // app.use('/assets', function(request, response, next){  // console.log(request.url)  // next()  // })  app.use('/assets', express.static('assets'))  app.get('/', function(request, response){  response.render('index')  })  app.get('/contact', function(request, response){  response.render('contact')  })  app.get('/profile/:id', function(request, response){  var data = {age: 29, job: 'programer', hobbies: ['Sleeping', 'Eating', 'DnD']}  response.render('profile', {person: request.params.id, data: data})  })  app.listen(3000) |
| --- |

### index.ejs

| <body>  <%- include ("partials/nav.ejs") %>  <h1>This is a test site for Node.js.</h1>  <p>Purely for testing purposes.</p>  </body> |
| --- |

# Query Strings

* Query Strings - additional data added on to a HTTP request in the form of name value pairs
  + We see this in some blogs where we can say we want to see all the blog posts from the news category: mysite.com/blog/news?page=2 or mysite.com/blog/contact?person=kota&dept=tech
  + ? - starts the query string
  + & - we can add more name value pairs
    - Page = 2 is attached to the get request. This is our name value pair
  + We are parsing the request and pulling the data out

### server.js

| var express = require('express')  var app = express()  app.set('view engine', 'ejs')  app.use('/assets', express.static('assets'))  app.get('/', function(request, response){  response.render('index')  })  app.get('/contact', function(request, response){  //console.log(request.query) //this will tell us what data is on the query string  response.render('contact', {qs: request.query}) //we're passing any data through the contact view  })  app.get('/profile/:id', function(request, response){  var data = {age: 29, job: 'programer', hobbies: ['Sleeping', 'Eating', 'DnD']}  response.render('profile', {person: request.params.id, data: data})  })  app.listen(3000) |
| --- |

### contact.ejs

| <body>  <%- include ("partials/nav.ejs") %>  <h1>This is a test site for Node.js. Contact.HTML</h1>  <p>Purely for testing purposes.</p>  <!-- testing the output of the query string  <p><%= qs.dept %></p>  <p><%= qs.person %></p>  -->  <form id="contact-form">  <label for="who">Who do you want to contact?</label>  <input type="text" name="who" value="<%= qs.person %>"/>  <label for="department">Which department?</label>  <input type="text" name="department" value="<%= qs.dept %>"/>  <label for="email">Your email</label>  <input type="email" name="email"/>  <input type="submit" value="submit">  </form>  </body> |
| --- |

# Post Request

* Post is a request method that asks the server to accept/store data which is enclosed in the body of the request
  + Often used when submitting forms
* We are using body-parse
  + Parse incoming request bodies in a middleware before your handlers, available under the req.body property.
  + $ npm install body-parser

### server.js

| var express = require('express')  var bodyParser = require('body-parser')  var app = express()  //this is a shortcut to this piece of middleware where we can  //parse our post data for us and store that data in this variable  var urlencodedParser = bodyParser.urlencoded({ extended: false })  app.set('view engine', 'ejs')  app.use('/assets', express.static('assets'))  app.get('/', function(request, response){  response.render('index')  })  app.get('/contact', function(request, response){  response.render('contact', {qs: request.query})  })  // POST /contact gets urlencoded bodies  app.post('/contact', urlencodedParser, function(request, response) {  console.log(request.body) //this sends us the data posted in th body  // response.render('contact', {qs: request.query})  //we will have access to this data in the contact-success view  response.render('contact-success', {data: request.body})  })  app.get('/profile/:id', function(request, response){  var data = {age: 29, job: 'programer', hobbies: ['Sleeping', 'Eating', 'DnD']}  response.render('profile', {person: request.params.id, data: data})  })  app.listen(3000) |
| --- |

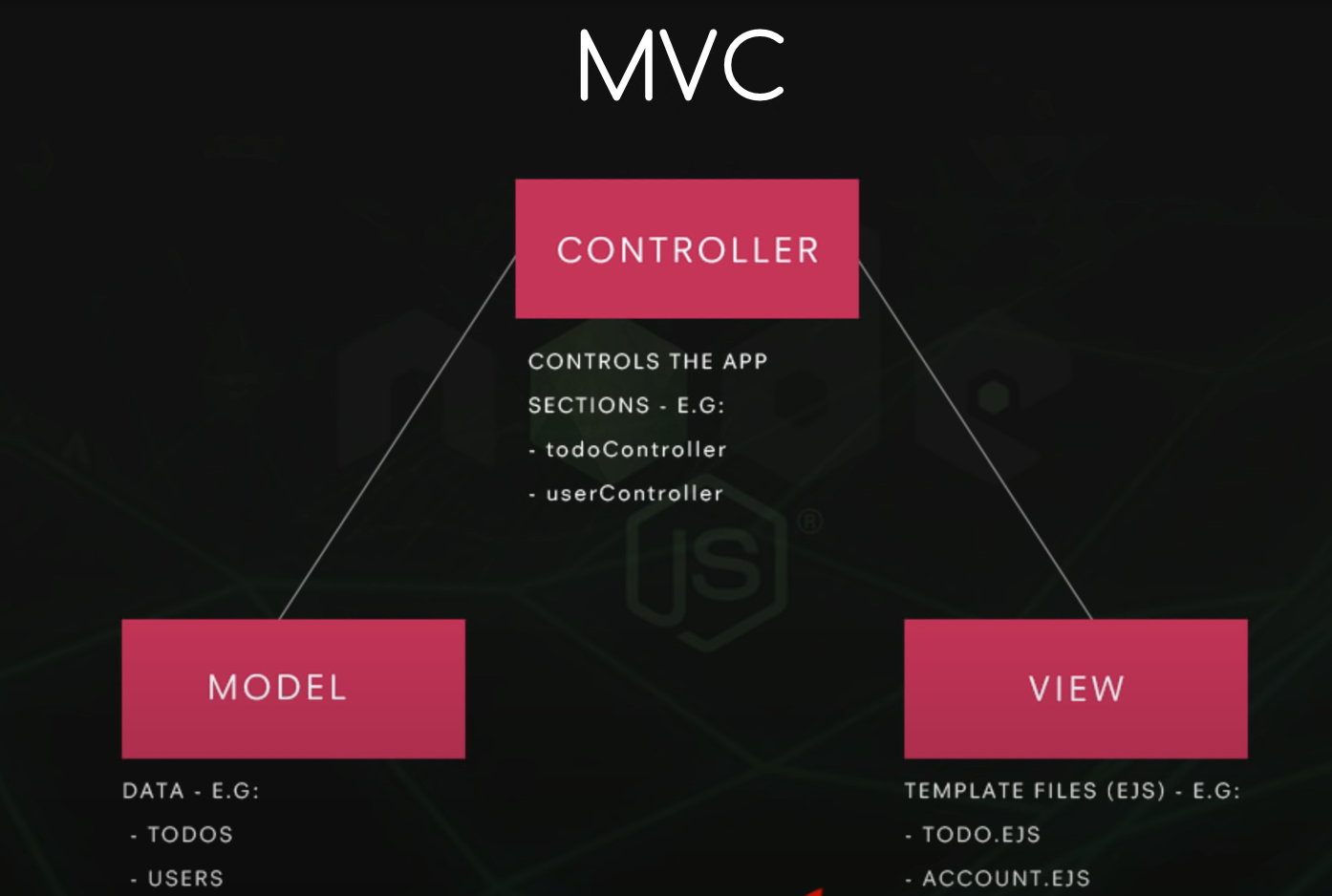
### contact.ejs

| <body>  <%- include ("partials/nav.ejs") %>  <h1>This is a test site for Node.js. Contact.HTML</h1>  <p>Purely for testing purposes.</p>  <!-- this is a POSt method and the action we are posting to-->  <form id="contact-form" method="POST" action="/contact">  <label for="who">Who do you want to contact?</label>  <input type="text" name="who" value="<%= qs.person %>"/>  <label for="department">Which department?</label>  <input type="text" name="department" value="<%= qs.dept %>"/>  <label for="email">Your email</label>  <input type="email" name="email"/>  <input type="submit" value="submit">  </form>  </body> |
| --- |

### contact-success.ejs

| <body>  <%- include ("partials/nav.ejs") %>  <h1>This is a test site for Node.js. Contact.HTML</h1>  <p>Thanks for getting in touch!</p>  <p>You contacted <%= data.who %> in the <%= data.department %> department.</p>  <p>We will reply to you at <%= data.email %></p>  </body> |
| --- |

# MVC Architecture



# NoSQL Databases and MongoDB

* NoSQL - alternative to an SQL database where we can store documents (JSON) in a db, instead of tables with rows and columns
  + Works well with javascript

### Storing JSON

| [  {  Item: ‘walk the dog’  },  {  Item: ‘eat some pie’  }  ] |
| --- |

* Mongodb - <https://www.mongodb.com/>
* Hosted version - <https://www.mongodb.com/atlas/database>

# Source

<https://www.youtube.com/playlist?list=PL4cUxeGkcC9gcy9lrvMJ75z9maRw4byYp>