Node JS Cheat Sheet

|  |  |  |
| --- | --- | --- |
| Serial -Video Number | Topic/Code | Explanation |
|  | Different command line codes | *Touch*  *Mkdir*  *Rm*  *Rm -rf (*rf is a flag = recursive force*) =* Removing entire directory  *Node -v*  *Npm -v*  *Cd*  *Cd ..* |
|  | Running a node file | Node fileName.ext |
|  | NPM | Node package manager  Helps us install a lot of packages. |
|  | Npm install | Npm install package name    This is used to install a package using NPM.  Whenever we install a package there would be a folder called node\_modules. Its contains the files of that package.  When we want to use that package we just create a variable and store it inside it. EX:  Var something = require(“package name”); |
|  | Npm init | Creates package.json |
|  | Express | Npm install -express  Require(“express”)  Now express has a lot of methods so just calling express(); wouldn’t do we call this method and store it inside a variable.  Var app = express(); |
| 1. 268 | Creating routes | app.get("/", function(req,res){  })  Req is for request. What information was it requested with.  Res is for respond. With what information re you going to respond with.  These are objects  This code is for creating routes.  app.listen(port,"ip address”, function(){  })  we have to tell it to listen for particular request |
| 1. 269 | Package.json | Contains all the meta data of that corresponding package. Includes authors, tags and dependencies that are related to the project. |
| 1. 269 | --save | Adds dependencies.EX:  Npm install express -save OR npm I -S express |
| 1. 270 | Keeping your servers on all the time | Installing: npm I -g nodemon ()  Then nodemon app name |
| 1. 271 | For other undefined routes | We use \*.  app.get("\*",function(req,res){  res.send("doesn’t exist");  })  REMEMBER ORDER MATTERS HERE. SHOULD ALWAYS BE IN THE END. |
|  | /:name | Match with anything that comes after /  Get access to the value that was replaced with a user given value (r/:name => r/puppy) via req.params this would return an object with attributes and values so for the given example we would get  {  Name: “puppy”  }  Directly getting the value req.params.name |
| 1. 275 | Res.render(“filename”, {object Type so key value pair}) | Renders HTML file/EJS (embedded JS) file  Has to be inside views directory. For ejs files have to have that pacakage.  EX:  Res.render(“friends”, {  Friends(this is the variable that would be refered to inside the EJS file): friends(The value of that variable in this case friends is an array so were passing the value of friends)  })  In our EJS file we need to put value here to make the website dynamic. And we can have access to passed values through <%= TREATED AS JS %>  We have to wrap every line of JS with the tags.  <%= %> vs <% %>  1st one: The value would be calculated and return the result to the html  2nd one: for logics and loops etc |
| 1. 277 | Partials | Partials are templates we can use inside other templates. Has to be inside views/partials/fileName.  Adding partials  <% include(“partials/file”)%>  Remember to put / Infront of the CSS file so that it understands that the file is not inside the views directory rather look somewhere else. |
| 1. 278 | Use other directories | For using other directories such as for CSS files we use public directory (EX) we have to tell express to use that directory and serve the contents. |
|  | View engine | app.set(“view engine”, “ejs”); is basically telling express that the files that we would be rendering would always be ejs files so that we don’t have to write EJS as an extension every time we put a new file in render. |
| 1. 279 | App.post(“path”) | For posting data (ex; forms) |
| 1. 280 | Getting values from a form | To get values from a form we need to use req.body however this wouldn’t work as express doesn’t support this so we need to install another package. Called body-parser which returns an object  Npm install body-parser –save  var bodyParser = require("body-parser");  app.use(bodyParser.urlencoded({extended: true})); |
| 1. 280 | redirect | res.redirect("/pathname");  redirects you to another defined path  the default is a get request |
| 1. 281 | API | Application programming interface  It basically helps you to connect with other applications or hardwares.  Resources: IFFTT, programmable Web |
| 1. 283 | Making requests | var request = require("request");  request(‘link’,function(error,response,body){  })  Ex:  var request = require("request");  request('http://www.google.com',function(error,response,body){      //error handles the error      if(error){          console.log("Something went wring: " + error);      }      else if(response.statusCode == 200){          //recieved successfully          console.log(body);      }  })  //Also we have another way of doing it (ES6 version)  //request as of now is depreicated instead we have request promise  //to install it we need to do the following  //npm i -S request-promise  const rp = require("request-promise");  rp('https://jsonplaceholder.typicode.com/users/1')  .then((body)=> { //these are called promises this part is basically the response      const parsedData = JSON.parse(body);      console.log(parsedData.name + " Lives In " + parsedData.address.city);  })  .catch((err)=> {      console.log("Error ", err );  }); |
| 1. 285 | Working with API | The response that comes in the body is always a string. So when we want to get information given by an api we need to turn it into an object and to do that we need to do the following.  Var parsedData = JSON.parse(body);  To get access to an object value.  parsedData[“ ”][“ ”] or  parsedData.ObjectName.ObjectName or  ‘$(parsedData.ObjectName.ObjectName) lives in parsedData.ObjectName.ObjectName’ //here $() this is basically making it a string and returning the dynamic value. |
| 1. 287 | Locus | Locus is a trouble shooting tool package.  Npm i -D locus (here d stands for development tool)  Using locus: eval(require(‘locus’))  Where ever this code is written it would have access to all the variables uptill that part.  For example in the console we can use response.statusCode to get the value. |
| MONGO DB (ODM) | | |
| 1. 303 | Mongod | Runs the mongo demon/process |
| 1. 303 | Shows dbs | Shows all databases |
| 1. 303 | Use dbName | Creates and switches to that db (if created just switches) |
| 1. 303 | Db(refers to the current db).collectionName.insert( {object} ) | For inserting into a collection |
| 1. 303 | Show collection | Shows all the collections in that database |
| 1. 303 | Db.collectionName.find() | Shows all the content inside.  Db.collectionName.find({name: “value”}) => finds corresponding item. |
| 1. 304 (notes) | Db.collection.updateOne({Attribute},{newAttribuute})  OR  Db.collection.updateMany({Attribute},{newAttribuute}) | What this does is it finds the object and replaces the whole thing with the new value. IF we want to preserve the value.  Db.collection.update({Attribute},{$set: {attributes}})  Ex:  Db.dogs.update({name: “puffy”},{$set: {breed: “puddle”, cute: false } } ) |
| 1. 304 (notes) | Db.collection.deleteOne({attribute})  OR  Db.collection.deleteMany({attribute}) | Specify with attribute. |
| 1. 305 | Mongoose | Package to use mogoDB.  Npm install mongoose  Var mongoose = require(“mongoose”)  Mongoose.connect(“mongodb://localhost/dbName”) //this creates DB if not created |
| 1. 305 | Creating schema | NOT DEFINING A STRUCTURE but creating a pattern. This is easy for us to code.  Var schemaName = new mongoose.Schema( {  attributes  } );  //compiling it into a model/pattern  Var varName = mongoose.model(“varName(singular version of the collection so cat to cats)”, catSchema);  varName. methods |
| 1. 305 | Adding values | EX:  Var puffy = new cat({  Name:”puffy”,  Age:”5”  });  Puffy.save(function(err, cat){  If(err){  Console.log();  }else{  Console.log(“saved”);  }  }); |
| 1. 306 | Create and save | Creates and saves at the same time.  EX: cat.create({  Name:”puffy”,  Age:”5”  }) |
| 1. 309 | Droping collection | Db.collection.drop() |
| 1. 310 | Find using an id | EX: campground.findById(id. Function(){  }) |
| 1. 312 | Adding default values to a attribute | EX: created :{type: Date, default: Date.now} |
| RESTFUL Routing | | |
| 1. 309 & 311\*\* | Restful routing | There are 7 of them.   |  |  |  |  | | --- | --- | --- | --- | | Name | URL | HTTP Verb | Description | | Index | /campground | GET | Shows the list of campgrounds | | New | /campground/new | GET | Displays the form to new camp | | Create | /campground | POST | The form submits the data to /campground as a POST request. | | Show | /campground/:id | GET | Shows info about one campground | | Edit | /campgrounds/:id/edit | GET | Editing one campground (FORM) | | Update | /campgrouds/:id | PUT | The form submits here.  Update a particular camp. | | Destroy | /campground/:id | DELETE | Delete a particular campground. | |
| 1. 312 | CSS public directory | app.use(express.static("public")); |
| 1. 315 | New way of submitting data | <input type="text" name="blog[title]">  What this will do is, instead of making the value for title available directly from req.body.title it will put it inside of an object, like so: req.body.blog.title  Now all of the values from the inputs in the form get added into one object (req.body.blog) and you can easily input that data into the database simply by passing in req.body.blog to Blog.create()  IAN’s comment:  In this instance you cannot use dot notation over bracket notation. If you used blog.title, then in your back-end code you'd have to access that value separately using req.body['blog.title'], which wouldn't allow you to access all of the values of the items inside of the object simultaneously with something like req.body.blog  In the form, each input has a value (once the user has typed something in), then on submit of the form, all that data gets sent to the back-end (to be put into the database in this case). We could just set each input to it's own variable, so if you used name then the value for that input would be accessible in the back-end via req.body.name, but this is cumbersome so we short circuit the whole process by assigning each input as a key (pointing to the input value) within one object (e.g., blog). Body-parser is the middleware that allows us to do this and unless we use bracket notation in the form, then it will not recognize the input's value as being a part of an object, able to be referenced by a key. |
| 1. 317 | Making a field mandatory in semantic | Add required to the input |
| 1. 317 | File traversal | / means go back to the root folder, then traverse forward/downward.  ./ means begin in the folder we are currently in (current working directory) and traverse forward/downward in the tree.  ../ means go up one directory, then begin the traverse. |
| 1. 318 | JS date manipulation | .toDateString()  Makes the date human readable. |
| 1. 318 | How to write HTML inside a text body(form) | Use <%- blog.body %> instead of <%=blog.body%>    What this will do is it would rather than substituting the value there it would evaluate the content inside. So run HTML if there are any inside. |
| 1. 318 | Show small amount of content | JS code  .substring(0,100) |
| 1. 319 | Why PUT doesn’t work | PUT request is basically a get request because HTML forms don’t support put request they only support GET and POST. So when we say method=”POST” then the it actually goes to the GET route with all the values inside the form in the URL. So how can we use PUT as it is still an HTTP request that updates values. So for that we need to do the following.ADD the following code at the end of the url in the action attribute of the form.  **?\_method=”PUT”**  What this is doing is that it sends as a get request however because of that additional line it treats as PUT. (METHOD OVERRIDE)  EX:   <form action="/blogs/<%=blog.\_id%>?\_method=PUT" method="POST" class="ui form">  This itself doesn’t do anything so we need to install the method-override package.  Npm i -S method-override  var methodOverride  = require("method-override");  app.use(methodOverride("\_method"));  So what this is saying that whenever in the url you see method you make it a put request.  ALL OF THIS IS BECAUSE OF RESTful ROUTING |
| 1. 319 | Updating items in one go | EX:   blog.findByIdAndUpdate(id to find, new data, function(err, updatedBlog){        }) |
| 1. 320 | Delete Request | As we are following the RESTful convention we have to use this,  app.delete("/blogs/:id", function(req,res){  <form action="/blogs/<%=blog.\_id%>?\_method=DELETE" method="POST" class="ui form">  otherwise we could have easily done  app.get("/deleteBlogs/:id", function(req,res){ |
| 1. 322 | Sanitizer | Sanitizes inputs of all sorts of script tags.  Npm i -S express-sanitizer  var expressSanitizer= require("express-sanitizer");  //this should always be after body parser  app.use(expressSanitizer());  req.sanitize(content);  ex:  req.body.blog.body =  req.sanitize(req.body.blog.body); |
| 1. 324 | Embedding Data | To have a one to many or many to many relationship we need to update our schema to handle it. For example, a person can have multiple post so his schema should have an array that can hold references to all his posts. To do that we need to have a post attribute in the schema that would hold post content. So,  posts:[postSchema] //Has to be a schema of the content |
| 1. 326 | Object Reference | Rather than putting the corresponding post in one array. So an array would have something like this,  Post =[  {  Title:”asdasdas”,  Author:”asasdasd”  },  {},{} …  ]  We can put references to these post in the array.So it would look like this,  Post =[  {  Id:5656535  },  {},{} …  ]  In side a post we have  { id: 5656535  Title:”asdasdas”,  Author:”asasdasd”  }  To do this we have to change the post attribute in the schema.  Posts:[  {  Type:mongoose.Schema.Types.ObjectId,  ref:”model variable name (Post)”  }  ]  So what this is saying is that the post array holds references to id of a post object. |
| 1. 327 | Modularizing the code | Put diffferent parts in different sections, import things that are required then at the end put  module.exports = send the file you want to be exported.  Var varName = require(“./models/post”)  This is basically the return statement. |
| 1. 326 | Save vs update | The main difference is that with .save() you already have an object in your client side code or had to retrieve the data from the server before you are writing it back, and you are writing back the whole thing.  On the other hand .update() does not require the data to be loaded to the client from the server. All of the interaction happens server side without retrieving to the client.So .update() can be very efficient in this way when you are adding content to existing documents.  In addition, there is the multi parameter to .update() that allows the actions to be performed on more than one document that matches the query condition.  There are some things in convenience methods that you lose when using .update() as a call, but the benefits for certain operations is the "trade-off" you have to bear. For more information on this, and the options available, see the documentation.  <https://stackoverflow.com/questions/22278761/mongoose-difference-between-save-and-using-update/22278847>  save doesn’t take parameters. EX:  found |