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 file name.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 $() this is basically making it a string and returning the dynamic value. |
| 1. 287 | Locus | Locus is a trouble shooting tool package.  Npm i -D locus (here d stands for development tool)  Using locus: eval(require(‘locus’))  Where ever this code is written it would have access to all the variables uptill that part.  For example in the console we can use response.statusCode to get the value. |
| MONGO DB | | |
| 1. 303 | Mongod | Runs the mongo demon/process |
| 1. 303 | Shows dbs | Shows all databases |
| 1. 303 | Use dbName | Creates and switches to that db (if created just switches) |
| 1. 303 | Db(refers to the current db).collectionName.insert( {object} ) | For inserting into a collection |
| 1. 303 | Show collection | Shows all the collections in that database |
| 1. 303 | Db.collectionName.find() | Shows all the content inside.  Db.collectionName.find({name: “value”}) => finds corresponding item. |
| 1. 304 (notes) | Db.collection.updateOne({Attribute},{newAttribuute})  OR  Db.collection.updateMany({Attribute},{newAttribuute}) | What this does is it finds the object and replaces the whole thing with the new value. IF we want to preserve the value.  Db.collection.update({Attribute},{$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 | Pacakage 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”  }) |