JavaScript and Node FUNdamentals

A Collection of Essential Basics

//

AZAT MARDAN

JavaScript and Node FUNdamentals

A Collection of Essential Basics

Azat Mardan

This book is for sale at http://leanpub.com/jsfun

This version was published on 2015-05-05

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Tweet This Book!

Please help Azat Mardan by spreading the word about this book on Twitter!

The suggested tweet for this book is:

I just downloaded JavaScript and Node.js FUNdamentals!!! CC @azat_co

The suggested hashtag for this book is #JavaScriptFUNdamentals.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#JavaScriptFUNdamentals

Also By Azat Mardan

Rapid Prototyping with JS

Oh My JS

Introduction to OAuth with Node.js

ProgWriter [programmer + writer]

Быстрое Прототипирование с JS

ProgWriter 2.0: Beyond Books

5 Hacks to Getting the Job of Your Dreams to Live a Happier and Healthier Life

Contents

If it's not fun, it's not JavaScript.

1. JavaScript FUNdamentals: The Powerful and Misunderstood Language of The Web

1.1 Expressiveness

Programming languages like BASIC, Python, and C has boring machine-like nature which require developers to write extra code that's not directly related to the solution itself. Think about line numbers in BASIC or interfaces, classes and patterns in Java.

On the other hand JavaScript inherits the best traits of pure mathematics, LISP, C# which lead to a great deal of expressiveness[1] (and fun!).

More about Expressive Power in this post: What does "expressive" mean when referring to programming languages?[2]

The quintessential Hello World example in Java (remember, Java is to JavaScript like ham is to hamster):

```
1  public class HelloWorld {
2      public static void main(String[] args) {
3          System.out.println("Hello World");
4      }
5  }
```

The same example in JavaScript:

```
1  console.log('Hello World')
```

or from within an HTML page:

---

[1]http://en.wikipedia.org/wiki/Expressive_power
[2]http://stackoverflow.com/questions/638881/what-does-expressive-mean-when-referring-to-programming-languages

```
1  <script>
2  document.write('Hello World')
3  </script>
```

JavaScript allows programmers to focus on the solution/problem rather than to jump through hoops and API docs.

1.2 Loose Typing

Automatic type casting works well most of the time. It's a great feature that saves a lot of time and mental energy! There're only a few primitives types:

1. String
2. Number (both integer and real)
3. Boolean
4. Undefined
5. Null

Everything else is an object, i.e., mutable keyed collections. Read Stackoverflow on What does immutable mean?[3]

Also, in JavaScript there are String, Number and Boolean objects which contain helpers for the primitives:

```
1  'a' === new String('a') //false
```

but

```
1  'a' === new String('a').toString() //true
```

or

```
1  'a' == new String('a') //true
```

By the way, == performs automatic type casting while === not.

1.3 Object Literal Notation

Object notation is super readable and compact:

---

[3]http://stackoverflow.com/questions/3200211/what-does-immutable-mean

```
1  var obj = {
2      color: "green",
3      type: "suv",
4      owner: {
5          ...
6      }
7  }
```

Remember that functions are objects?

```
1  var obj = function () {
2      this.color: "green",
3      this.type: "suv",
4      this.owner: {
5          ...
6      }
7  }
```

1.4 Functions

Functions are **first-class citizens**, and we treat them as variables, because they are objects! Yes, functions can even have properties/attributes.

1.4.1 Create a Function

```
1  var f = function f () {
2      console.log('Hi');
3      return true;
4  }
```

or

```
1  function f () {
2    console.log('Hi');
3    return true;
4  }
```

Function with a property (remember functions are just objects that can be invoked, i.e. initialized):

```
1  var f = function () {console.log('Boo');}
2  f.boo = 1;
3  f(); //outputs Boo
4  console.log(f.boo); //outputs 1
```

Note: the return keyword is optional. In case it's omitted, the function will return `undefined` upon invocation.

1.4.2 Pass Functions as Params

```
1  var convertNum = function (num) {
2   return num + 10;
3  }
4
5  var processNum = function (num, fn) {
6     return fn(num);
7  }
8
9  processNum(10, convertNum);
```

1.4.3 Invocation vs. Expression

Function definition as follows:

```
1  function f () {};
```

Function invocation looks like this:

```
1  f();
```

Function expression always returns something, because it resolves to some value which could be a number, a string, an object or a boolean:

```
1  function f() {return false;}
2  f();
```

On the other hand, if a function doesn't return anything, it's a statement:

```
1   function f(a) {console.log(a);}
```

1.5 Arrays

Arrays are also objects which have some special methods inherited from Array.prototype[4] global object. Nevertheless, JavaScript Arrays are **not** real arrays. Instead, they are objects with unique integer (usually 0-based) keys.

```
1   var arr = [];
2   var arr2 = [1, "Hi", {a:2}, function () {console.log('boo');}];
3   var arr3 = new Array();
4   var arr4 = new Array(1,"Hi", {a:2}, function () {console.log('boo');});
```

1.6 Prototypal Nature

There are **no classes** in JavaScript because objects inherit directly from other objects which is called prototypal inheritance. There are a few types of inheritance patterns in JS:

- Classical
- Pseudo-classical
- Functional

Example of the functional inheritance pattern:

```
1   var user = function (ops) {
2     return { firstName: ops.name || 'John'
3            , lastName: ops.name || 'Doe'
4            , email: ops.email || 'test@test.com'
5            , name: function() { return this.firstName + this.lastName}
6            }
7   }
8
9   var agency = function(ops) {
10    ops = ops || {}
11    var agency = user(ops)
12    agency.customers = ops.customers || 0
13    agency.isAgency = true
14    return agency
15  }
```

---

[4]https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/prototype#Properties

1.7 Conventions

Most of these conventions (with semi-colons being an exception) are stylistic, and highly preferential and don't impact the execution.

1.7.1 Semi-Colons

Optional semi-colons, except for two cases:

1. In for loop construction: `for (var i=0; i++; i<n)`
2. When a new line starts with parentheses, e.g., Immediately-Invoked Function Expression (IIFE): `;(function(){...}())`

1.7.2 camelCase

camelCase, except for class names which are CapitalCamelCase, e.g.,

```
1  var MainView = Backbone.View.extend({...})
2  var mainView = new MainView()
```

1.7.3 Naming

_,$ are perfectly legitimate characters for the literals (jQuery and Underscore libraries use them a lot).

Private methods and attributes start with _ (does nothing by itself!).

1.7.4 Commas

Comma-first approach

```
1  var obj = { firstName: "John"
2            , lastName: "Smith"
3            , email: "johnsmith@gmail.com"
4            }
```

1.7.5 Indentation

Usually it's either tab, 4 or 2 space indentation with their supporters' camps being almost religiously split between the options.

1.7.6 White spaces

Usually, there is a space before and after =, +, { and } symbols. There is no space on invocation, e.g., `arr.push(1);`, but there's a space when we define an anonymous function: `function () {}`.

1.8 No Modules

At least until ES6[5], everything is in the global scope, a.k.a. `window` and included via `<script>` tags. However, there are external libraries that allow for workarounds:

- CommonJS[6]
- AMD and Require.js[7]

Node.js uses CommonJS-like syntax and **has** built-in support for modules.

To hide your code from global scope, make private attributes/methods use closures and immediately-invoked function expressions[8] (or IIFEs).

1.9 Immediately-Invoked Function Expressions (IIFEs)

```
1  (function () {
2  window.yourModule = {
3  ...
4  };
5  }());
```

This snippet shows an example of an object with private attribute and method:

[5]https://wiki.mozilla.org/ES6_plans

[6]http://www.commonjs.org/

[7]http://requirejs.org/

[8]http://en.wikipedia.org/wiki/Immediately-invoked_function_expression

```
1  (function () {
2    window.boo = function() {
3      var _a = 1;
4      var inc = function () {
5        _a++;
6        console.log(_a);
7        return _a;
8      };
9      return {
10       increment: inc
11     };
12   }
13 }());
14 var b = window.boo();
15 b.increment();
```

Now try this:

```
1  b.increment();
2  b.increment();
3  b.increment();
```

1.10 Keyword "this"

Mutates/changes a lot (especially in jQuery)! Rule of thumb is to re-assign to a locally scoped variable before attempting to use this inside of a closure:

```
1  var app = this
2  $('a').click(function(e){
3    console.log(this) //most likely the event or the      target anchor element
4    console.log(app) //that's what we want!
5    app.processData(e)
6  })
```

When in doubt: console.log!

1.11 Pitfalls

JS is the only language that programmers think they shouldn't learn. Things like === vs. ==, global scope leakage, DOM, etc. might lead to problems down the road. This is why it's important to understand the language or use something like CoffeeScript, that takes way most of the issues.

1.12 Further Learning

If you liked this article and would like to explore JavaScript more, take a look at this amazing **free** resource: Eloquent JavaScript: A Modern Introduction to Programming[9].

Of course for more advanced JavaScript enthusiasts and pros, there's my book Rapid Prototyping with JS[10] and intensive programming school HackReactor[11], where I teach part-time.

---

[9]http://eloquentjavascript.net/

[10]http://rpjs.co

[11]http://hackreactor.com

2. Node.js FUNdamentals: JavaScript on The Server

Node.js is a highly efficient and scalable non-blocking I/O platform that was built on top of Google Chrome V8 engine and its ECMAScript. This means that most front-end JavaScript (another implementation of ECMAScript) objects, functions and methods are available in Node.js. Please refer to JavaScript FUNdamentals[1] if you need a refresher on JS-specific basics.

Developers can install Node.js from its website[2] (Figure 4-1) and follow this overview of main Node.js concepts.

---

[1] http://webapplog.com/js-fundamentals/
[2] http://nodejs.org
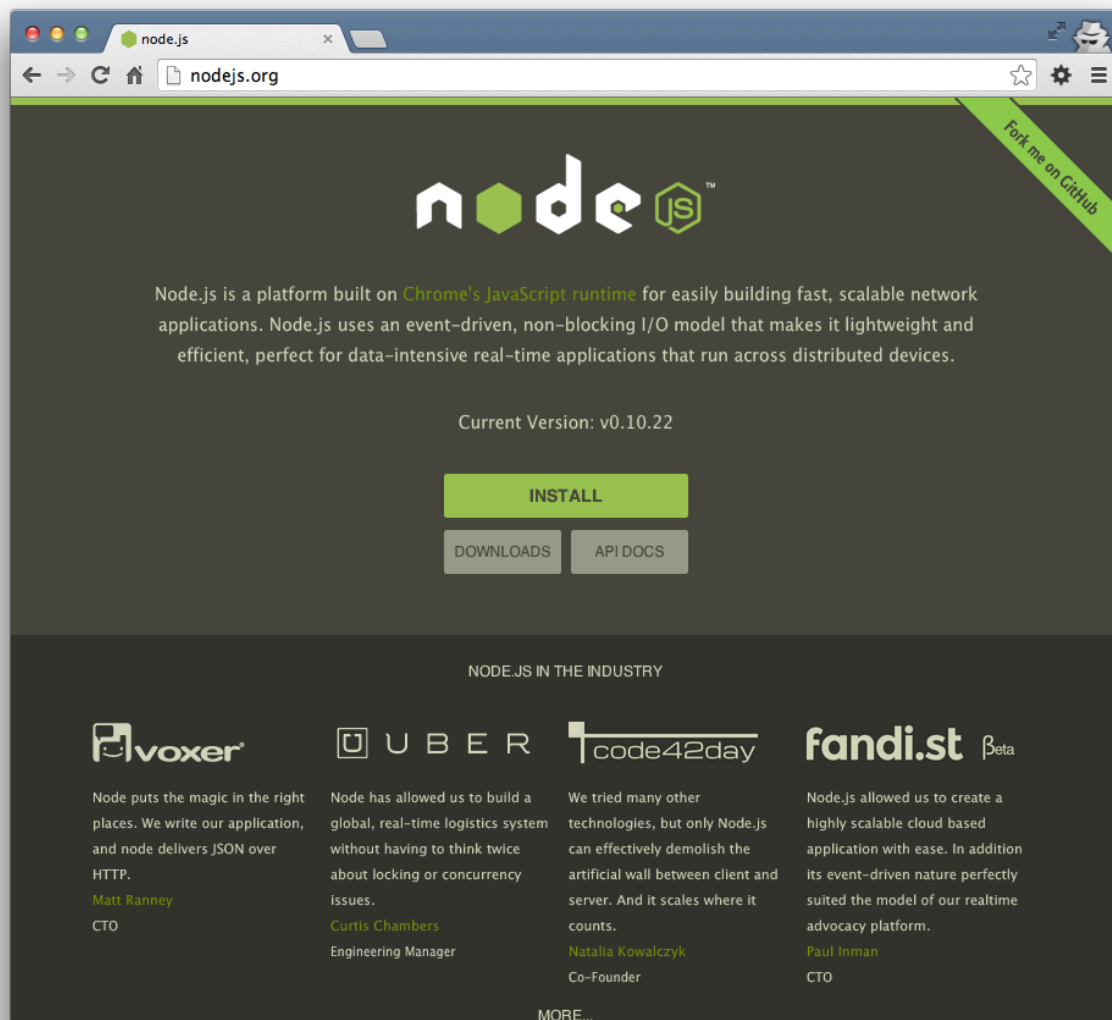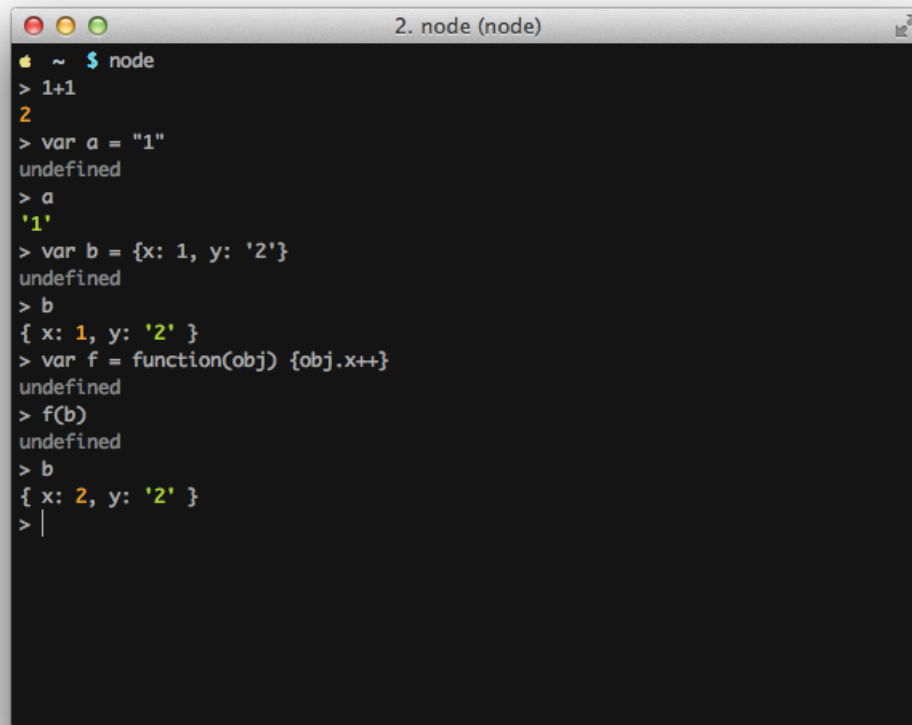
Figure 4-1: Node.js home page

2.1 Read-Eval-Print Loop (a.k.a. Console) in Node.js

Like in many other programming language and platforms, Node.js has a read-eval-print loop tool which is opened by $ node command. The prompt changes to › and we can execute JavaScript akin to Chrome Developer Tools console as shown in Figure 4-2. There are slight deviations in ECMAScript implementations in Node.js and browsers (e.g., {}+{}), but for the most part the results are similar.

Figure 4-2: Node.js console
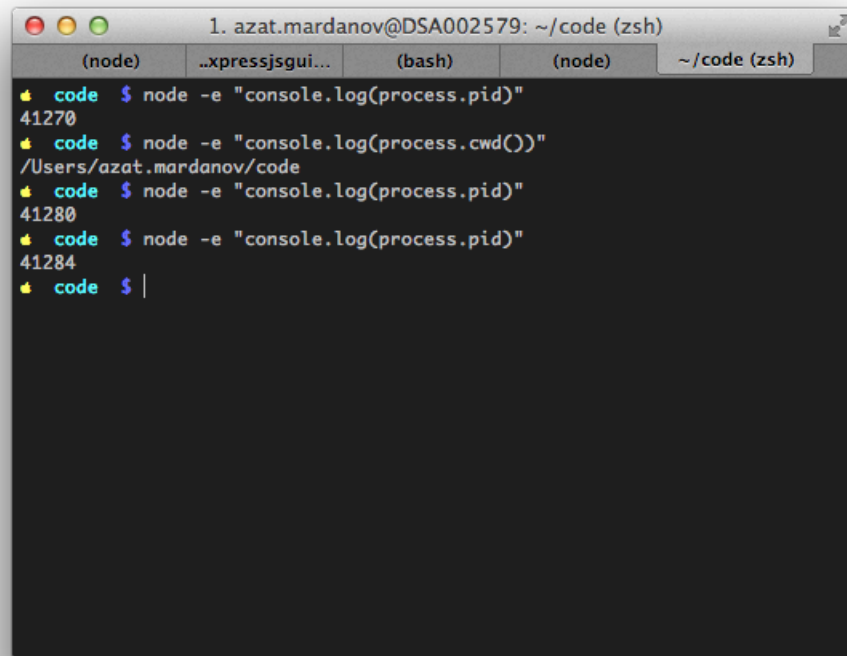
So as you see, we can write JavaScript in the console all day long, but sometime we can to save script so we can run them later.

2.2 Launching Node.js Scripts

To start a Node.js script from a file, simply run $ `node filename`, e.g., $ `node program.js`. If all we need is a quick set of statements, there's a `-e` option that allow to run inline JavaScript/Node.js, e.g., $ `node -e "console.log(new Date());"`.

2.3 Node.js Process Information

Each Node.js script that is running is a process in its essence. For example, `ps aux | grep 'node'` will output all Node.js programs running on a machine. Conveniently, developers can access useful process information in code with `process` object, e.g., `node -e "console.log(process.pid)"` as illustrated in Figure 4-3.

Figure 4-3: Node.js process examples using pid and cwd.

2.4 Accessing Global Scope in Node.js

As you know from JS FUNdamentals[3], browser JavaScript by default puts everything into its global scope. This was coined as one of the bad part of JavaScript in Douglas Crockford's famous [JavaScript: The Good Parts]. Node.js was designed to behave differently with everything being local by default. In case we need to access globals, there is a `global` object. Likewise, when we need to export something, we should do so explicitly.

In a sense, `window` object from front-end/browser JavaScript metamorphosed into a combination of `global` and `process` objects. Needless to say, the `document` object that represent DOM of the webpage is nonexistent in Node.js.

2.5 Exporting and Importing Modules

Another *bad part* in browser JavaScript is that there's no way to include modules. Scripts are supposed to be linked together using a different language (HTML) with a lacking dependency

---

[3]http://webapplog.com/

management. CommonJS[4] and RequireJS[5] solve this problem with AJAX-y approach. Node.js borrowed many things from the CommonJS concept.

To export an object in Node.js, use `exports.name = object;`, e.g.,

```
 1  var messages = {
 2    find: function(req, res, next) {
 3    ...
 4    },
 5    add: function(req, res, next) {
 6    ...
 7    },
 8    format: 'title | date | author'
 9  }
10  exports.messages = messages;
```

While in the file where we import aforementioned script (assuming the path and the file name is `route/messages.js`):

```
 1  var messages = require('./routes/messages.js');
```

However, sometime it's more fitting to invoke a constructor, e.g., when we attach properties to Express.js app (more on Express.js in Express.js FUNdamentals: An Essential Overview of Express.js[6]). In this case `module.exports` is needed:

```
 1  module.exports = function(app) {
 2    app.set('port', process.env.PORT || 3000);
 3    app.set('views', __dirname + '/views');
 4    app.set('view engine', 'jade');
 5    return app;
 6  }
```

In the file that includes the example module above:

---

[4]http://www.commonjs.org/

[5]http://requirejs.org/

[6]http://webapplog.com/express-js-fundamentals/

```
1  ...
2  var app = express();
3  var config = require('./config/index.js');
4  app = config(app);
5  ...
```

The more succinct code: `var = express(); require('./config/index.js')(app);`.

The most common mistake when including modules is a wrong path to the file. For core Node.js modules, just use the name without any path, e.g., require('name'). Same goes for modules in `node_-modules` folder. More on that later in the NPM section.

For all other files, use `.` with or without a file extension, e.g.,

```
1  var keys = require('./keys.js'),
2    messages = require('./routes/messages.js');
```

In addition, for the latter category it's possible to use a longer looking statements with `__dirname` and `path.join()`, e.g., require(path.join(__dirname, 'routes', 'messages'));'

If `require()` points to a folder, Node.js will attempt to read `index.js` file in that folder.

2.6 Buffer is a Node.js Super Data Type

Buffer is a Node.js addition to four primitives (boolean, string, number and RegExp) and all-encompassing objects (array and functions are also objects) in front-end JavaScript. We can think of buffers as extremely efficient data stores. In fact, Node.js will try to use buffers any time it can, e.g., reading from file system, receiving packets over the network.

2.7 __dirname vs. process.cwd

`__dirname` is an absolute path to the file in which this global variable was called, while `process.cwd` is an absolute path to the process that runs this script. The latter might not be the same as the former if we started the program from a different folder, e.g., `$ node ./code/program.js`.

2.8 Handy Utilities in Node.js

Although, the core of the Node.js platform was intentionally kept small it has some essential utilities such as

- URL[7]
- Crypto[8]
- Path[9]
- String Decoder[10]

The method that we use in this tutorials is `path.join` and it concatenates path using an appropriate folder separator (/ or \\).

2.9 Reading and Writing from/to The File System in Node.js

Reading from files is done via the core `fs` module[11]. There are two sets of methods: async and sync. In most cases developers should use async methods, e.g., fs.readFile[12]:

```
1  var fs = require('fs');
2  var path = require('path');
3  fs.readFile(path.join(__dirname, '/data/customers.csv'), {encoding: 'utf-8'}, fu\
4  nction (err, data) {
5    if (err) throw err;
6    console.log(data);
7  });
```

And the writing to the file:

```
1  var fs = require('fs');
2  fs.writeFile('message.txt', 'Hello World!', function (err) {
3    if (err) throw err;
4    console.log('Writing is done.');
5  });
```

2.10 Streaming Data in Node.js

Streaming data is a term that mean an application processes the data while it's still receiving it. This is useful for extra large datasets, like video or database migrations.

Here's a basic example on using streams that output the binary file content back:

---

[7]http://nodejs.org/api/url.html

[8]http://nodejs.org/api/crypto.html

[9]http://nodejs.org/api/path.html

[10]http://nodejs.org/api/string_decoder.html

[11]http://nodejs.org/api/fs.html

[12]http://nodejs.org/api/fs.html#fs_fs_readfile_filename_options_callback

```
1  var fs = require('fs');
2  fs.createReadStream('./data/customers.csv').pipe(process.stdout);
```

By default, Node.js uses buffers for streams.

For a more immersive training, take a loot at stream-adventure[13] and Stream Handbook[14].

2.11 Installing Node.js Modules with NPM

NPM comes with the Node.js platform and allows for seamless Node.js package management. The way `npm install` work is similar to Git in a way how it traverses the working tree to find a current project[15]. For starter, keep in mind that we need either the `package.json` file or the `node_modules` folder, in order to install modules locally with `$ npm install name`, for example `$ npm install superagent`; in the program.js: `var suparagent = require('superagent');`.

The best thing about NPM is that it keeps all the dependencies local, so if module A uses module B v1.3 and module C uses module B v2.0 (with breaking changes comparing to v1.3), both A and C will have their own localized copies of different versions of B. This proves to be a more superior strategy unlike Ruby and other platforms that use global installations by default.

The best practice is **not to include** a `node_modules` folder into Git repository when the project is a module that supposed to be use in other application. However, it's recommended **to include** `node_modules` for deployable applications. This prevents a breakage caused by unfortunate dependency update.

Note: The NPM creator like to call it `npm` (lowercase[16]).

2.12 Hello World Server with HTTP Node.js Module

Although, Node.js can be used for a wide variety of tasks, it's mostly knows for building web applications. Node.js is thrives in the network due to its asynchronous nature and build-in modules such as net and http.

Here's a quintessential Hello World examples where we create a server object, define request handler (function with req and res arguments), pass some data back to the recipient and start up the whole thing.

---

[13]http://npmjs.org/stream-adventure
[14]https://github.com/substack/stream-handbook
[15]https://npmjs.org/doc/files/npm-folders.html
[16]http://npmjs.org/doc/misc/npm-faq.html#Is-it-npm-or-NPM-or-Npm

```
1  var http = require('http');
2  http.createServer(function (req, res) {
3    res.writeHead(200, {'Content-Type': 'text/plain'});
4    res.end('Hello World\n');
5  }).listen(1337, '127.0.0.1');
6  console.log('Server running at http://127.0.0.1:1337/');]
```

The req and res parameters have all the information about a given HTTP request and response correspondingly. In addition, req and res can be used as streams (look in the previous section).

2.13 Debugging Node.js Programs

The best debugger is `console.log()`, but sometime we need to see the call stack and orient ourselves in async code a bit more. To do that, put `debugger` statements in your code and use `$ node debug program.js` to start the debugging process[17]. For more developer-friendly interface, download node inspector[18].

2.14 Taming Callbacks in Node.js

Callbacks[19] are able to Node.js code asynchronous, yet programmers unfamiliar with JavaScript, who come from Java or PHP, might be surprised when they see Node.js code described on Callback Hell[20]:

```
1  fs.readdir(source, function(err, files) {
2    if (err) {
3      console.log('Error finding files: ' + err)
4    } else {
5      files.forEach(function(filename, fileIndex) {
6        console.log(filename)
7        gm(source + filename).size(function(err, values) {
8          if (err) {
9            console.log('Error identifying file size: ' + err)
10         } else {
11           console.log(filename + ' : ' + values)
12           aspect = (values.width / values.height)
13           widths.forEach(function(width, widthIndex) {
```

---

[17]http://nodejs.org/api/debugger.html
[18]https://github.com/node-inspector/node-inspector
[19]https://github.com/maxogden/art-of-node#callbacks
[20]http://callbackhell.com/

```
14              height = Math.round(width / aspect)
15              console.log('resizing ' + filename + 'to ' + height + 'x' + height)
16              this.resize(width, height).write(destination + 'w' + width + '_' + f\
17  ilename, function(err) {
18                  if (err) console.log('Error writing file: ' + err)
19              })
20          }.bind(this))
21        }
22      })
23    })
24    }
25  })
```

There's nothing to be afraid of here as long as two-space indentation is used. ;-) However, callback code can be re-write with the use of event emitters, promises or by utilizing the async library.

2.15 Introduction to Node.js with Ryan Dahl

Last, but not least, take a few minutes and watch this Introduction to Node.js with Ryan Dahl video[21].

2.16 Moving Forward with Express.js

For more meticulous Node.js instructions, take a look at Rapid Prototyping with JS: Agile JavaScript Development[22] and Node School[23] (Figure 4-1).

---

[21]http://www.youtube.com/embed/jo_B4LTHi3I

[22]http://rpjs.co

[23]http://nodeschool.io

Figure 4-4: Node School home page

3. Express.js FUNdamentals: The Most Popular Node.js Framework

Express.js is an amazing framework for Node.js projects and used in the majority of such web apps. Unfortunately, there's a lack of tutorials and examples on how to write good production-ready code. To mitigate this need, we released Express.js Guide: The Comprehensive Book on Express.js[1]. However, all things start from basics, and for that reason we'll give you a taste of the framework in this post, so you can decide if you want to continue the learning further.

3.1 Express.js Installation

Assuming you downloaded and installed Node.js (and NPM with it), run this command:

```
1  $ sudo npm install -g express@3.4.3
```

3.2 Express.js Command-Line Interface

Now we can use command-line interface (CLI) to spawn new Express.js apps:

```
1  $ express -c styl expressfun
2  $ cd expressfun && npm install
3  $ node app
```

Open browser at http://localhost:3000.

Here is the full code of expressfun/app.js if you don't have time to create an app right now:

---

[1]http://expressjsguide.com

```
1   var express = require('express');
2   var routes = require('./routes');
3   var user = require('./routes/user');
4   var http = require('http');
5   var path = require('path');
6
7   var app = express();
8
9   // all environments
10  app.set('port', process.env.PORT || 3000);
11  app.set('views', __dirname + '/views');
12  app.set('view engine', 'jade');
13  app.use(express.favicon());
14  app.use(express.logger('dev'));
15  app.use(express.bodyParser());
16  app.use(express.methodOverride());
17  app.use(app.router);
18  app.use(express.static(path.join(__dirname, 'public')));
19
20  // development only
21  if ('development' == app.get('env')) {
22    app.use(express.errorHandler());
23  }
24
25  app.get('/', routes.index);
26  app.get('/users', user.list);
27
28  http.createServer(app).listen(app.get('port'), function(){
29    console.log('Express server listening on port ' + app.get('port'));
30  });
```

3.3 Routes in Express.js

If you open the expressfun/app.js, you'll see two routes in the middle:

```
1   ...
2   app.get('/', routes.index);
3   app.get('/users', user.list);
4   ...
```

The first one is basically takes care of all the requests to the home page, e.g., http://localhost:3000/ and the latter of requests to /users, such as http://localhost:3000/users. Both of the routes process URLs case insensitively and in a same manner as with trailing slashes.

The request handler itself (`index.js` in this case) is straightforward: everything from the HTTP request is in `req` and write results to the response in `res`:

```
1  exports.list = function(req, res){
2    res.send("respond with a resource");
3  };
```

3.4 Middleware as The Backbone of Express.js

Each line above the routes is a middleware:

```
1  app.use(express.favicon());
2  app.use(express.logger('dev'));
3  app.use(express.bodyParser());
4  app.use(express.methodOverride());
5  app.use(app.router);
6  app.use(express.static(path.join(__dirname, 'public')));
```

The middleware is a pass thru functions that add something useful to the request as it travels along each of them, for example `req.body` or `req.cookie`. For more middleware writings check out Intro to Express.js: Parameters, Error Handling and Other Middleware².

3.5 Configuration of an Express.js App

Here is how we define configuration in a typical Express.js app:

```
1  app.set('port', process.env.PORT || 3000);
2  app.set('views', __dirname + '/views');
3  app.set('view engine', 'jade');
```

An ordinary settings involves a name (e.g., `views`) and a value (e.g., path to the folder where out templates/views live). There is more than one way to define a certain settings, e.g, `app.enable` for boolean flags.

3.6 Jade is Haml for Express.js/Node.js

Jade template engine is akin to Ruby on Rails' Haml in the way it uses whitespace and indentation, e.g., `layout.jade`:

---

²http://webapplog.com/intro-to-express-js-parameters-error-handling-and-other-middleware/

```
1  doctype 5
2  html
3    head
4      title= title
5      link(rel='stylesheet', href='/stylesheets/style.css')
6    body
7      block content
```

Other than that, it's possible to utilize full-blown JavaScript code inside of Jade templates.

3.7 Conclusion About The Express.js Framework

As you've seen, it's effortless to create MVC web apps with Express.js. The framework is splendid for REST APIs as well. If you interested in them, visit the Tutorial: Node.js and MongoDB JSON REST API server with Mongoskin and Express.js[3] and Intro to Express.js: Simple REST API app with Monk and MongoDB[4].

If you want to know what are the other middlewares and configurations, check out Express.js API docs[5], Connect docs[6] and of course our book â€" Express.js Guide[7]. For those who already familiar with some basics of Express.js, I recommend going through ExpressWorks[8] â€" an automated Express.js workshop.

3.8 ExpressWorks

After you've mastered Node.js and Express.js basics in this book, you might want to consider working on ExpressWorks[9](Figure 5-1). This is an interactive class about the Express.js framework which is as of today is the most popular module on NPM.

---

[3]http://webapplog.com/tutorial-node-js-and-mongodb-json-rest-api-server-with-mongoskin-and-express-js/

[4]http://webapplog.com/intro-to-express-js-simple-rest-api-app-with-monk-and-mongodb/

[5]http://expressjs.com/api.html

[6]http://www.senchalabs.org/connect/

[7]http://expressjsguide.com

[8]http://webapplog.com/expressworks
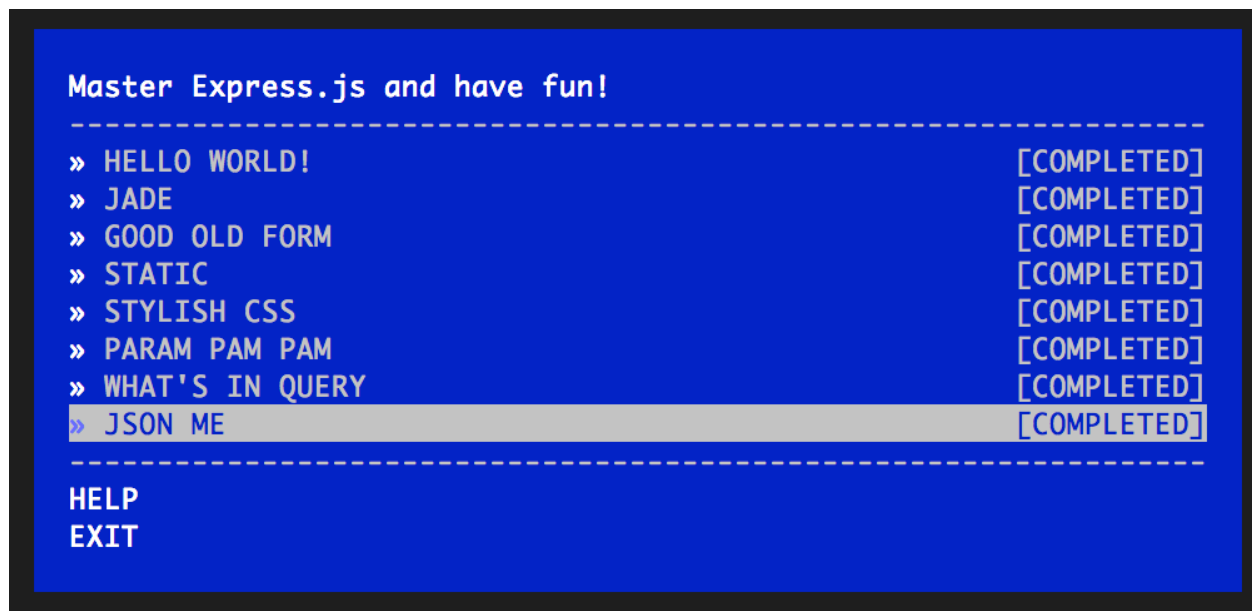
[9]http://webapplog.com/expressworks

Figure 5-1: ExpressWorks menu

3.9 Update

For migrating from Express.js 3.x to 4.x take a look at this guide: Migrating Express.js 3.x to 4.x: Middleware, Route and Other Changes[10].

Share on Twitter with ClickToTweet link: http://clicktotweet.com/HDUx0, or just click:

> "I've finished JavaScript and Node.js FUNdamentals: A Collection of Essential Basics by @azat_co #nodejs https://leanpub.com/jsfun"[11]

---

[10]http://webapplog.com/migrating-express-js-3-x-to-4-x-middleware-route-and-other-changes/

[11]http://ctt.ec/VQcEb

4. About the Author

Azat Mardan: A software engineer, author and yogi.

Azat Mardan has over 12 years of experience in web, mobile and software development. With a Bachelor's Degree in Informatics and a Master of Science in Information Systems Technology degree, Azat possesses deep academic knowledge as well as extensive practical experience.

Currently, Azat works as a Software Engineer Team Lead at DocuSign[1], where his team rebuilds 50 million user product (DocuSign web app) using the tech stack of Node.js, Express.js, Backbone.js, CoffeeScript, Jade, Stylus and Redis.

Recently, he worked as an engineer at the curated social media news aggregator website Storify.com[2] (acquired by LiveFyre[3]). Before that, Azat worked as a CTO/co-founder at Gizmo[4] â€" an enterprise cloud platform for mobile marketing campaigns, and he has undertaken the prestigious 500 Startups[5] business accelerator program. Previously, he was developing mission-critical applications for government agencies in Washington, DC: National Institutes of Health[6], National Center for Biotechnology Information[7], Federal Deposit Insurance Corporation[8], and Lockheed Martin[9]. Azat is a frequent attendee at Bay Area tech meet-ups and hackathons (AngelHack[10], and was a hackathon 2012 finalist with team FashionMetric.com[11]).

---

[1] http://docusign.com

[2] http://storify.com

[3] http://livefyre.com

[4] http://www.crunchbase.com/company/gizmo

[5] http://500.co/

[6] http://nih.gov

[7] http://ncbi.nlm.nih.gov

[8] http://fdic.gov

[9] http://lockheedmartin.com

[10] http://angelhack.com

[11] http://fashionmetric.com

In addition, Azat teaches technical classes at General Assembly[12] and Hack Reactor[13], pariSOMA[14] and Marakana[15] (acquired by Twitter) to much acclaim.

In his spare time, he writes about technology on his blog: webAppLog.com[16] which is number one[17] in "express.js tutorial" Google search results. Azat is also the author of Express.js Guide[18], Rapid Prototyping with JS[19] and Oh My JS[20].

---

[12] http://generalassemb.ly
[13] http://hackreactor.com
[14] http://parisoma.com
[15] http://marakana.com
[16] http://webapplog.com
[17] http://expressjsguide.com/assets/img/expressjs-tutorial.png
[18] http://expressjsguide.com
[19] http://rpjs.co
[20] http://leanpub.com/ohmyjs

5. Review

If you liked this book (or not), please leave your review on the Amazom page[1].

---

[1]http://amzn.to/19qbVzn

6. Errata

Please help us make this book better by submitting issues via other means of communication listed below in the **Contact Us** section.

7. Contact Us

Let's be friends on the Internet!

- Tweet Node.js question on Twitter: @azat_co[1]
- Follow Azat on Facebook: facebook.com/profile.php?id=1640484994[2]
- GitHub: github.com/azat-co[3]

Other Ways to Reach Us

- Email Azat directly: hi@azat.co[4]
- Google Group: rpjs@googlegroups.com[5] and https://groups.google.com/forum/#!forum/rpjs
- Blog: webapplog.com[6]
- HackHall[7]: community for hackers, hipsters and pirates

Share on Twitter with ClickToTweet link: http://clicktotweet.com/HDUx0, or just click:

"I've finished JavaScript and Node.js FUNdamentals: A Collection of Essential Basics by @azat_co #nodejs https://leanpub.com/jsfun"[8]

---

[1] https://twitter.com/azat_co

[2] https://www.facebook.com/profile.php?id=1640484994

[3] https://github.com/azat-co

[4] mailto:hi@azat.co

[5] mailto:rpjs@googlegroups.com

[6] http://webapplog.com

[7] http://hackhall.com

[8] http://ctt.ec/VQcEb