DAY 2

NODE.JS

Belkacem Alidra

June 26, 2018

nomades.ch

TODAY

- Node.js modules
- Callback and error handling
- async
- Back on yesterday's exercise
  - Finish first static version
  - Tests with curl and Postman
- Testing
  - mocha
  - chai
  - supertest
  - Cucumber.js
  - Zombie.js
- Let's code !

NODE.JS MODULES

REQUIRE()

Modules installed from npm are simply files and directories saved into the **node_modules/** directory.

require will look for them:

```
const mod = require("mod");  // node_modules/mod.js
                             // node_modules/mod/index.js
```

You can use the same logic inside of your project:

```
const mod = require("./mod"); // ./mod.js
                              // ./mod/index.js
```

MODULE.EXPORTS

From a module, visibility is simply controlled by setting **module.exports**:

```
/*
 * mod.js */
"use strict";

// i is local to this module, invisible from the "outside".
let i=0;

// nexti() is local to this module, invisible from the "outside".
function nexti() {
    return i++;
}

// next() is exposed to require().
module.exports = {
    next: function () {
        return 42 + nexti();
    }
};
```

CALLBACK & ERROR HANDLING

CALLBACK & ERROR HANDLING

Node.js uses closures (callback) extensively.

Usually, the first argument given to a callback function is an Error, e.g.

```
1 fs.readFile("/etc/hosts", function callback(err, buf) { if (err)
2     return console.error(err.message); // do something with buf
3 });
```

CALLBACK & ERROR HANDLING

Rather than throwing, errors and values are propagated through callback recursively:

```
1 "use strict";
2 const fs = require("fs");
3
4 // callback the word count of a given file path.
5 function wc(path, callback) {
6     fs.readFile(path, function (err, buf) {
7         if (err)
8             return callback(err);
9         const count = buf.toString().trim().split(/\s+/).length;
10        return callback(/* no error */null, count);
11    });
12 }
13
14 // node callback-err.js <path>
15 const path = process.argv[2];
16 wc(path, function (err, count) {
17    /* here we're given either (Error, undefined) or (null, Number) */
18    if (err)
19        console.error(err.message);
20    else
21        console.log(count);
22 });
```

CALLBACK & ERROR HANDLING

```php
function register()
{
    if (!empty($_POST)) {
        $msg = '';
        if ($_POST['user_name']) {
            if ($_POST['user_password_new']) {
                if ($_POST['user_password_new'] === $_POST['user_password_repeat']) {
                    if (strlen($_POST['user_password_new']) > 5) {
                        if (strlen($_POST['user_name']) < 65 && strlen($_POST['user_name']) > 1) {
                            if (preg_match('/^[a-z\d]{2,64}$/i', $_POST['user_name'])) {
                                $user = read_user($_POST['user_name']);
                                if (!isset($user['user_name'])) {
                                    if ($_POST['user_email']) {
                                        if (strlen($_POST['user_email']) < 65) {
                                            if (filter_var($_POST['user_email'], FILTER_VALIDATE_EMAIL)) {
                                                create_user();
                                                $_SESSION['msg'] = 'You are now registered so please login';
                                                header('Location: ' . $_SERVER['PHP_SELF']);
                                                exit();
                                            } else $msg = 'You must provide a valid email address';
                                        } else $msg = 'Email must be less than 64 characters';
                                    } else $msg = 'Email cannot be empty';
                                } else $msg = 'Username already exists';
                            } else $msg = 'Username must be only a-z, A-Z, 0-9';
                        } else $msg = 'Username must be between 2 and 64 characters';
                    } else $msg = 'Password must be at least 6 characters';
                } else $msg = 'Passwords do not match';
            } else $msg = 'Empty Password';
        } else $msg = 'Empty Username';
        $_SESSION['msg'] = $msg;
    }
    return register_form();
}
```


Arguably, “callback hell” is caused by poor coding practices. callbackhell.com is worth reading on the subject.

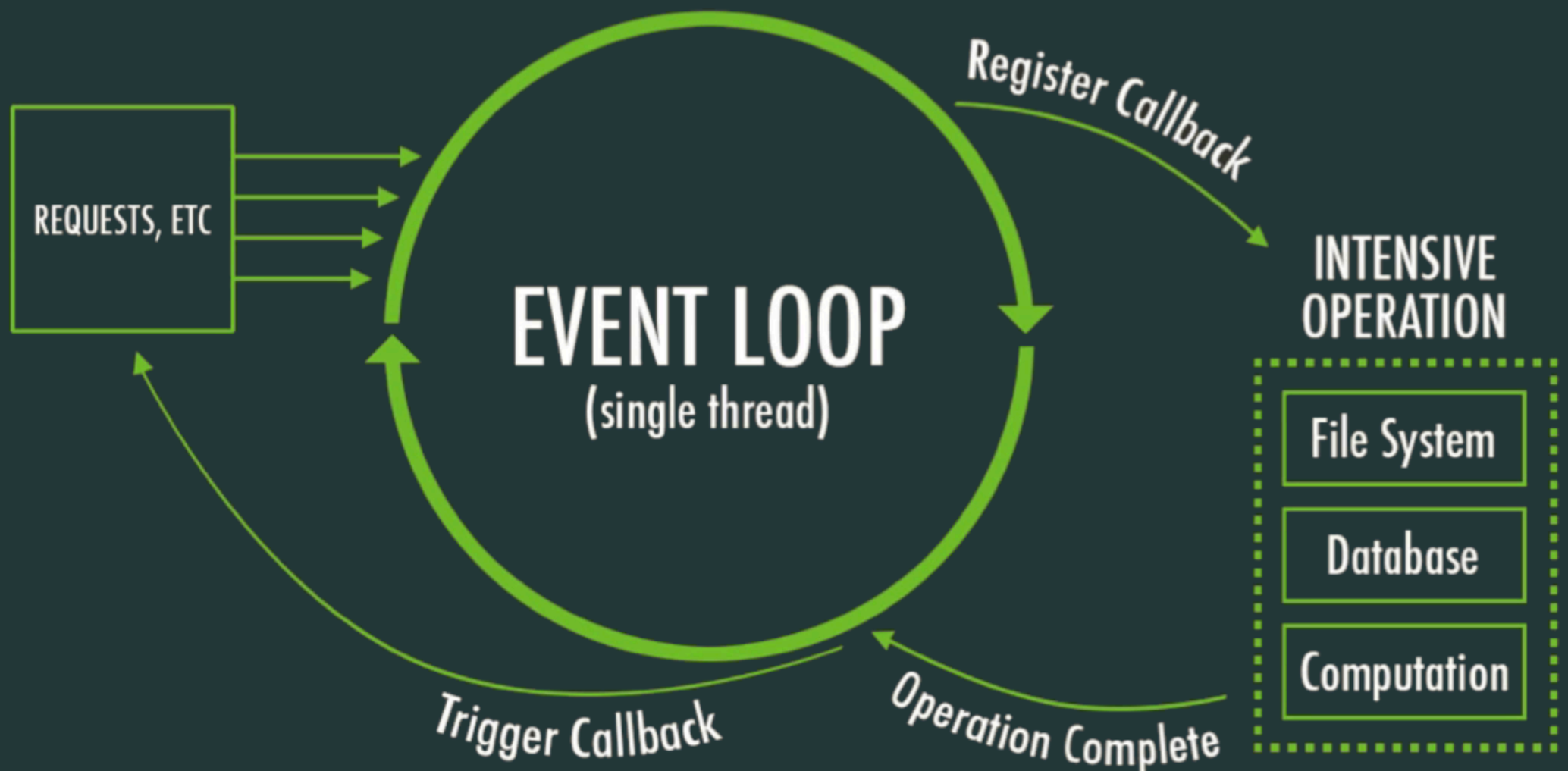
ASYNC

ASYNC LOOP

Remember that I/O are executed in worker threads asynchronously.

As a result, it is sometime tricky to understand in which order callback are executed.

```
1 "use strict";
2
3 const fs = require("fs");
4
5 ["/usr/share/dict/words", "/etc/hosts"].forEach(path => {
6     fs.readFile(path, (err, buf) => {
7         // which file will be printed first?
8         console.log(`${path} contains ${buf.length} bytes.`);
9     });
10 });
```


REQUESTS, ETC
EVENT LOOP
(single thread)
Register Callback
INTENSIVE OPERATION
File System
Database
Computation
Trigger Callback
Operation Complete

ASYNC LOOP

Solution using the async module:

```
1 "use strict";
2
3 const fs    = require("fs");
4 const async = require("async");
5
6 /* build an array of tasks (function) reading the files */
7 const tasks = ["/usr/share/dict/words", "/etc/hosts"].map(path => {
8     return function (done) {
9         fs.readFile(path, (err, buf) => {
10            if (err)
11                return done(err);
12
13            const content = buf.toString();
14            return done(null, content);
15        });
16    };
17 });
18
19 /* print the results */
20 async.parallel(tasks, (err, results) => {
21    if (err)
22        console.error(err.message);
23    else
24        console.dir(results);
25 });
```

ASYNC LOOP

Solution using Promise (ES7 async / await in Node.js >= 7.6)

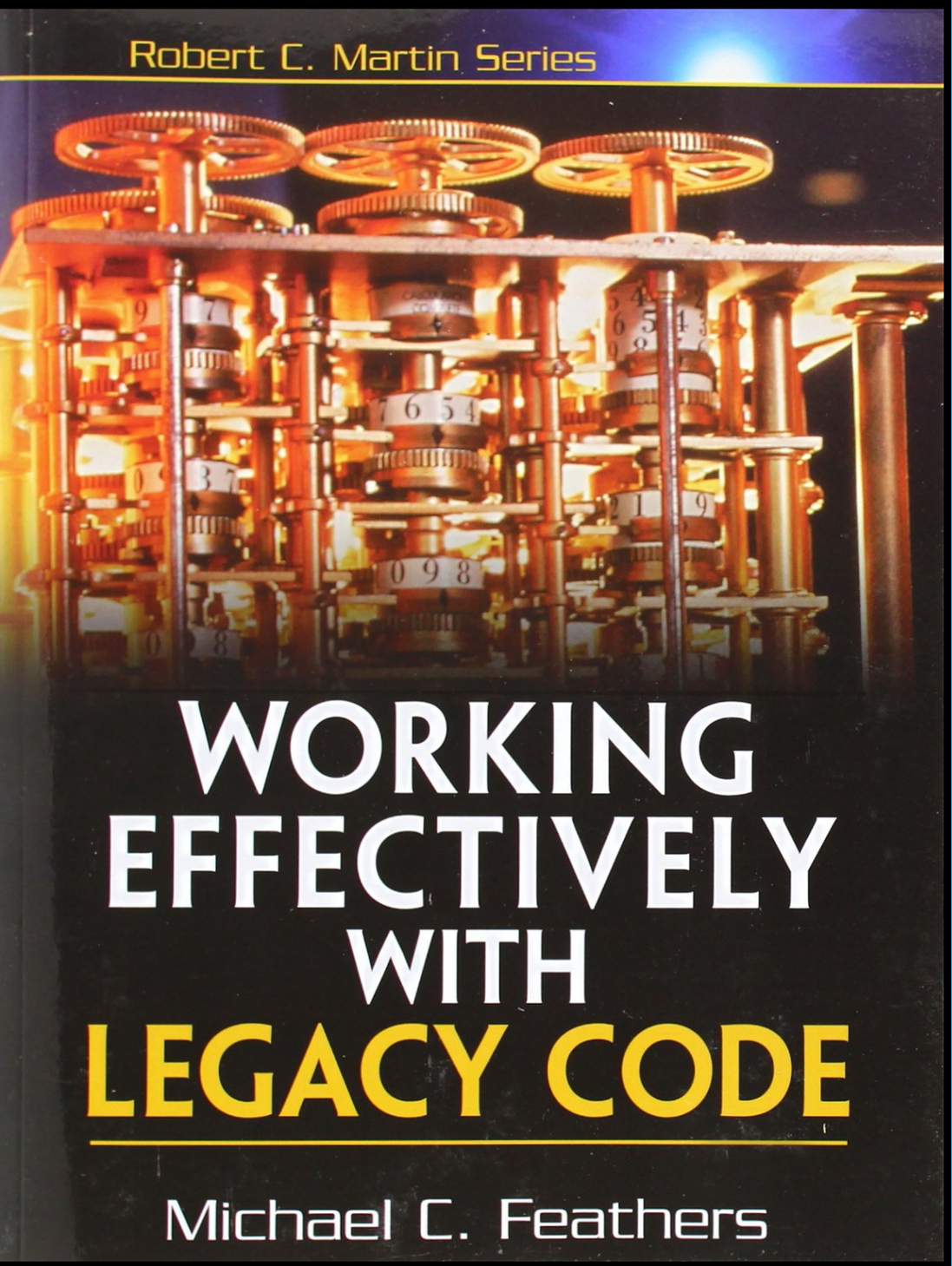
```
1 "use strict";
2
3 const fs = require("fs");
4 /* NOTE: no need to require Promise */
5
6 /* build an array of tasks (promises) reading the files */
7 const tasks = ["/usr/share/dict/words", "/etc/hosts"].map(path => {
8     return new Promise((resolve, reject) => {
9         fs.readFile(path, (err, buf) => {
10            if (err)
11                return reject(err);
12
13            const content = buf.toString();
14            return resolve(content);
15        });
16    });
17 });
18
19 /* print the results */
20 Promise.all(tasks)
21     .catch(err => {
22         console.error(err.message);
23     }).then(results => {
24         console.dir(results);
25     });
```

BACK ON YESTERDAY'S EXERCISE

BACK ON YESTERDAY'S EXERCISE

- Let's see your version
- Body parser ?
- Let's create a module
- Curl
  - % curl -H"Content-type: application/json" -XPOST -d@post.json "$HOST/api/posts"
- Postman

TESTING



THE MAIN THING THAT DISTINGUISHES LEGACY CODE FROM NON-LEGACY CODE IS A LACK OF COMPREHENSIVE TESTS.

Michael Feathers

LEAGACY CODE

Michael Feathers describe two coding strategies:

1. Edit and Pray
2. Cover and Modify

SETTING UP

Installation

% *npm install --save-dev mocha chai supertest*

Setup the test script in package.json:

```
"scripts": {
    "test": "mocha"
},
```

Create the test directory and add a blank file:

% *mkdir test && touch test/index.js*

You can now run the mocha tests by calling:

% *npm test*

TESTING HELLO WORLD

MOCHA

mocha helps you to give your test a structure and some control flow. Full documentation at mochajs.org.

```
"use strict";

const chai    = require("chai");
const expect  = chai.expect;
const request = require("supertest");

require("../hello-express.js");

describe("Hello World app", () => {
    it("should return 200", done => {
        request("localhost:3000").get("/").end((err, res) => {
            expect(res.status).to.eql(200);
            return done();
        });
    });

    it("should say Hello", done => {
        request("localhost:3000").get("/").end((err, res) => {
            expect(err).to.not.exist;
            expect(res.text).to.eql("Hello World!");
            return done();
        });
    });
});
```

CHAI

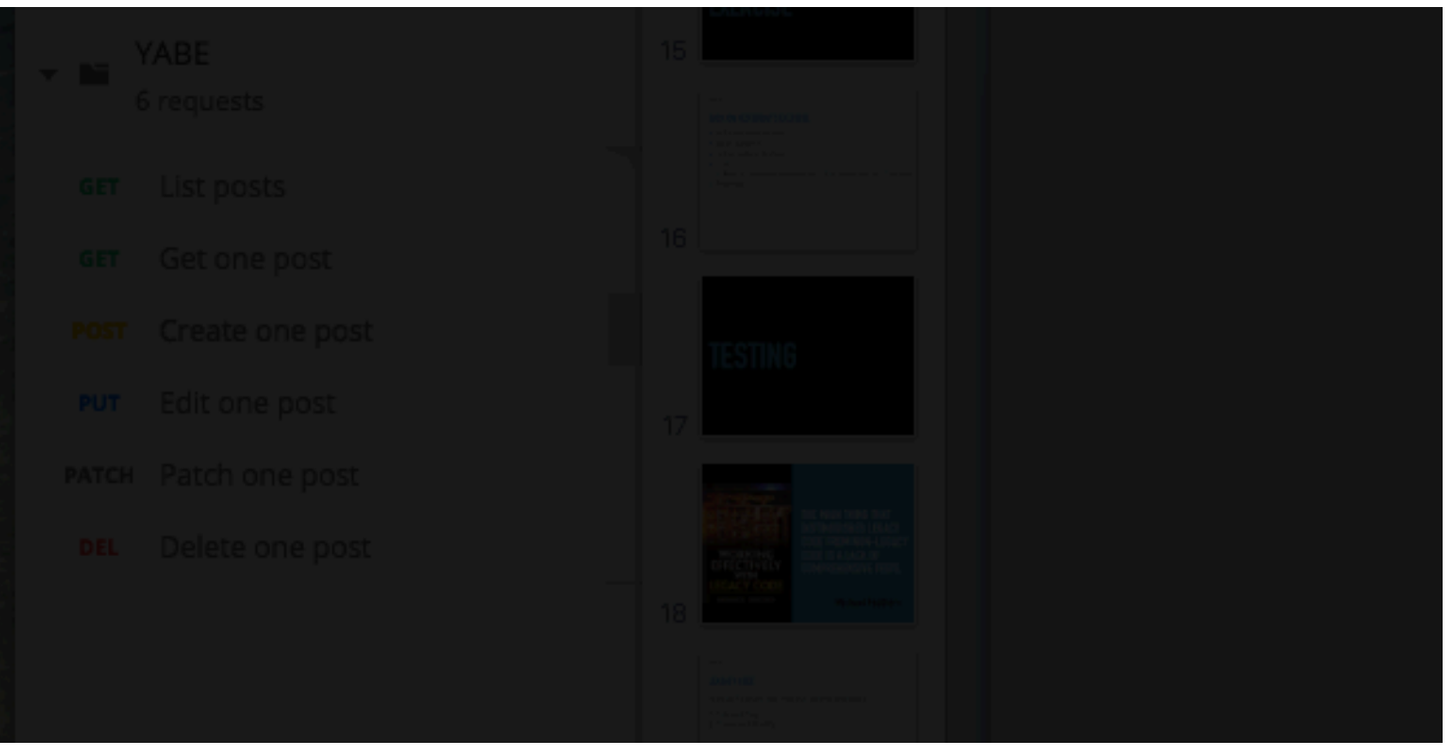
chai is an “assertion” library supporting several interfaces depending on your taste.
Full documentation at chaijs.com.

```
1 expect(answer).to.be.a('number');
2 expect(password).to.be.a('string').and.to.equal('Open Sesame');
3 expect(clients).to.have.lengthOf(1000);
4 expect(tea).to.have.property('flavors').with.lengthOf(3);
```

SUPERTEST

supertest is a specialized assertion library for HTTP.
It is very handy even to only make requests (e.g. POST with body).
The documentation is at the GitHub project page.

```
1 const request = require('supertest');
2 const express = require('express');
3
4 const app = express();
5
6 app.get('/user', function(req, res) {
7     res.status(200).json({ name: 'tobi' });
8 });
9
10 request(app)
11   .get('/user')
12   .expect('Content-Type', /json/)
13   .expect('Content-Length', '15')
14   .expect(200)
15   .end(function(err, res) {
16       if (err) throw err;
17   });
```



CUCUMBER.JS

Cucumber is a *Behaviour-Driven Development* testing tool.
Gherkin, Cucumber's non-technical and human readable language, is used to define test cases.
Documentation and examples at the GitHub project page.

```
# features/documentation.feature
Feature: Example feature
    As a user of Cucumber.js
    I want to have documentation on Cucumber
    So that I can concentrate on building awesome applications

Scenario: Reading documentation
    Given I am on the Cucumber.js GitHub repository
    When I click on "CLI"
    Then I should see "Running specific features"
```

ZOMBIE.JS

Zombie is a very fast, headless full-stack testing using Node.js. It *simulate* a browser environment and is able to evaluate Javascript. More at zombie.js.org.

```
1 const Browser = require('zombie');
2 const browser = new Browser();
3
4 browser.visit("https://github.com/cucumber/cucumber-js/tree/master", () => {
5     browser.clickLink("CLI").then(() => {
6         browser.assert.success();
7         browser.assert.text('body', /Running specific features/);
8         browser.tabs.closeAll();
9     });
10 });
```

LET'S CODE !

YET ANOTHER (TESTED!) BLOG ENGINE

1. Cover your blog server with tests, then
2. Refactor your code into functions and modules.

Questions ?

READ ON LATER

#NoTDD by Eric Gunnerson.