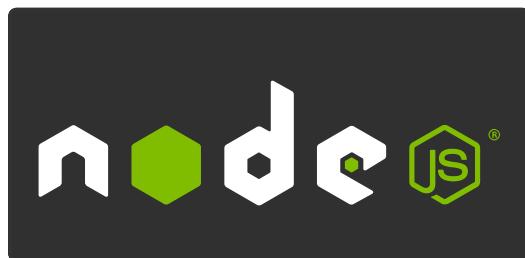


NODE.JS

SERVER SIDE JAVASCRIPT PLATFORM



APPLICATION DEVELOPMENT WITH NODE.JS

By Philippe Poulard

AGENDA

- Overview : async programming, event loop
- Core : REPL, HTTP, Events, Streams, File System
- Modules, npm, semver
- REST Web app with Express
- Socket.io
- Data access : MySQL, MongoDB, ORM
- Tools : debugging, testing, monitoring, frontend tools, deploying

Before starting (1/2)

REQUIREMENT

1. Node JS, npm
2. MySQL + MySQL Workbench
3. Mongodb + Mongohub (Mac) or
<http://robomongo.org/>
4. POSTMAN REST client for chrome
5. chrome, as mentioned above

Before starting (2/2)

EXERCICES OVERVIEW

1. Play with REPL
2. Starters : HTTP server, sync/async, shell
3. A simple Webapp
4. A simple module
5. A REST web app
6. A chat with WebSocket
7. MySQL + REST + Express
8. Mongo : bulk loading, aggregation
9. Tests unit in Node.js

READY ?

1. Ensure that everything is installed :

```
$ node --version  
v0.12.0  
$ npm --version  
2.5.1
```

2. and say "hello" :

```
hello.js  
console.log("Hello World");  
$ node hello.js
```

...to ensure everything works fine.

OVERVIEW

JS REMINDERS

5 SHADES OF 'THIS'

The value of `this` depends on how a function is invoked.

INVOKED AS A FUNCTION

```
function foo() {};  
foo(); //=> this === window  
var bar = function() {};  
bar(); //=> this === window
```

INVOKING AS A METHOD

```
function foo() {
    return this;
}
// invoke as a function
foo(); //=> this === window
var bar = {
    'foo': foo
};
// invoke as a method of 'bar'
bar.foo(); //=> this === bar
```

INVOKING AS A CONSTRUCTOR

```
function Foo() {
    this.bar = function() {
        return this;
    }
}
// exemple 1
var foo = new Foo();
console.log(foo); //=> Foo
console.log(foo.bar() instanceof Foo); //=> true

// exemple 2
var fool = Foo();
console.log(typeof fool); //=> undefined
console.log(typeof window.bar); //=> function
```

INVOKING BY PASSING THIS

```
function foo() {
  console.log(this); //=> this === element
  console.log(arguments); //=> 'a', 'b', 'c'
}
var element = document.querySelector('#foo');
element.addEventListener('click', function(e){
  console.log(this); //=> element
  console.log(arguments); //=> e === Event
  foo.call(element, 'a', 'b', 'c');
});
```

or

```
foo.apply(element, [ 'a', 'b', 'c' ]);
```

INVOKING WITH BINDING

```
var Foo = function() {
  this.counter = 0;
  this.inc = function() {
    this.counter += 1;
    console.log(this);
  };
};

var foo = new Foo();
var element = document.querySelector('#foo');

// #1
element.addEventListener('click', foo.inc); //=> this === element
// #2
element.addEventListener('click', function(){ foo.inc(); }); //=>
// #3
element.addEventListener('click', foo.inc.bind(foo)); //=> this =
```

BIND EXAMPLE 2

ES3

```
var obj = {
  doIt: function() {},
  handle: function() {
    var that = this;
    document.addEventListener('click', function(e) {
      that.doIt();
    });
  }
}
```

ES5

```
var obj = {
  doIt: function() {},
  handle: function() {
    document.addEventListener('click', function(e) {
      this.doIt();
    }.bind(this));
  }
}
```

ES6

```
var obj = {
  doIt: function() {},
  handle: function() {
    document.addEventListener('click', (e) => this.doIt());
  }
}
```

THE 3 LIVES OF JAVASCRIPT

- mid 1990's : DHTML
- 2000's : jQuery, prototype and RIA
- 2010's : Node.js and Angular

SERVER-SIDE JAVASCRIPT

- PHP in Apache
 - Java in Tomcat
 - Javascript in Node.js
-
- Node.js : a runtime environment and a library of modules
 - Google's V8 interpreter (chrome) → Node.js
 - Node.js ecosystem : npm, the node package manager



NON-BLOCKING MODEL

I/O is expensive

Thread-per-connection is memory-expensive

EVENT LOOP

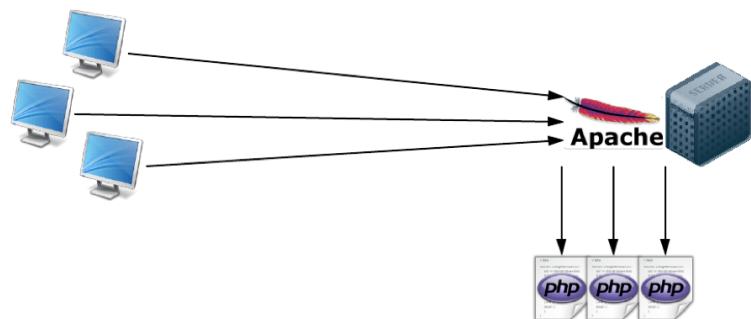
An event loop is an entity that handles and processes external events and converts them into callback invocations.

SYNC / ASYNC

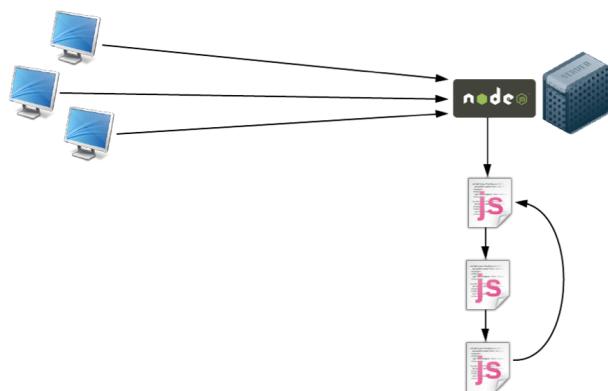
- Single-thread
- No preemptive multi-task
 - Non-blocking model (remember AJAX)
- Need uninterrupted CPU time ? (or the I/O cannot be made asynchronously)
 - fork out a separate process

Everything runs in parallel except your code !

A WEB APP



A NODE.JS WEB APP



CORE API

HTTP

(hello world from the web)

```
helloWeb.js
var http = require('http');
var server = http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World\n');
});
server.listen(1337);
console.log('Server running at http://localhost:1337/');

$ node helloWeb.js
```

Go ! → <http://localhost:1337/>

EXERCICE

- Return an HTML response with HTML content.
(I know it's somewhat simple but we'll need that code base later)

```
var http = require('http');
```

Get the http module that allow to create a Web server

```
var server = http.createServer();
```

http is a JS object with functions

```
var server = http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World\n');
});
```

Define the function that will be invoked for each incoming request

```
server.listen(1337);
```

Launch (asynchronously) the server

REPL

read-eval-print-loop

Interact with Node in Javascript

```
$ node
> 1 + 5
6
> var add = function (a, b) { return a + b; }
undefined
> add(1, 5)
6
> add
[Function]
```

- `_` contains the last result
- Multi-line statements appear with `...`
- Quit with `process.exit()`

EXERCICE : PLAY WITH REPL

1. Create an array of fruits
2. Display its size
3. Add a value at the end / at the start
4. Remove the last / the first value
5. Get slices
6. Create an object
7. Add a member
8. Create a function

GLOBAL OBJECTS

Try this in the REPL :

```
> console.log("Hello World !");
> global.console.log("Hello World !");
> window.console.log("Hello World !");

ReferenceError: window is not defined
at repl:1:1
at REPLServer.defaultEval (repl.js:132:27)
at bound (domain.js:254:14)
at REPLServer.runBound [as eval] (domain.js:267:12)
at REPLServer. (repl.js:279:12)
at REPLServer.emit (events.js:107:17)
at REPLServer.Interface._onLine (readline.js:214:10)
at REPLServer.Interface._line (readline.js:553:8)
at REPLServer.Interface._ttyWrite (readline.js:830:14)
at ReadStream.onkeypress (readline.js:109:10)
```

`global.console` and `console` are identical the same way `window.document` and `document` are identical in the browser.

But `window` is undefined in Node.

MAIN GLOBAL OBJECTS

Available in all modules. Sometimes they aren't actually in the global scope but in the module scope

- `global` : useful inside a module to retrieve the top-level scope
- `process`
- `console` : used to print to stdout and stderr
- `require()` : is local to each module
- `require.resolve()` : to resolve the file name of a module
- `__filename` : of the code being executed
- `__dirname`
- `module` : a reference to the current module
- `exports` : shortcut for `module.exports`

MAIN GLOBAL OBJECTS

- `setTimeout()`
- `clearTimeout()`
- `setInterval()`
- `clearInterval()`

THE PROCESS GLOBAL OBJECT

- `"exit", "beforeExit", "uncaughtException"` : events
- `env` : the user environment
- `stdout, stderr, stdin`
- `nextTick()` : once the current event loop complete, call a callback function

EXERCICE : GLOBALS IN THE REPL

Display with the REPL some global objects and process objects.

What do you notice ?

EXERCICE : SYNC / ASYNC

What is wrong with this code ?

```
sync.js
for (var i = 0; i < 10; i++) {
  process.nextTick(function () {
    console.log(i);
  });
}
console.log('You might think this gets printed last.')

$ node sync.js
```

EXERCICE : SYNC / ASYNC

Repair the previous code with async

```
$ mkdir async  
$ cd async  
$ npm init  
$ npm install async --save  
  
async.js  
var async = require('async');  
// your code here
```

<https://www.npmjs.com/package/async#parallel>

EVENTS

A single thread but no blocking operations.

Long tasks (read a file) are launched in background and a callback function is called once completed.

LISTENING EVENTS

Remember jQuery :

```
$( "canvas" ).on( "mouseleave", function() { ... } );
```

Now in Node :

```
server.on("close", function() { ... })
```

...where is defined that "close" event ?

THE HTTP API

A convenient response : events are defined in the API

<https://nodejs.org/api/http.html>

Table of Contents

- **HTTP**
 - [http.METHODS](#)
 - [http.STATUS_CODES](#)
 - [http.createServer\(\[requestListener\]\)](#)
 - [http.createClient\(\[port\]\[, host\]\)](#)
 - **Class: http.Server**
 - [Event: 'request'](#)
 - [Event: 'connection'](#)
 - [Event: 'close'](#)
 - [Event: 'checkContinue'](#)
 - [Event: 'connect'](#)
 - [Event: 'upgrade'](#)

USING THE HTTP API

```
var http = require('http');

var server = http.createServer(function(req, res) {
  res.writeHead(200);
  res.end('Hello world');
});

server.on('close', function() {
  console.log('Bye bye !');
})

server.listen(8080);

server.close();
```

EMITTING EVENTS

Create an `EventEmitter`:

```
var EventEmitter = require('events').EventEmitter;
var game = new EventEmitter();
```

Use `emit()`:

```
game.emit('gameover', 'You lost !');
```

Listen the event:

```
game.on('gameover', function(message) { });
```

THE EVENTS API

<https://nodejs.org/api/events.html>

Table of Contents

- [Events](#)
 - [Class: events.EventEmitter](#)
 - [emitter.addListener\(event, listener\)](#)
 - [emitter.on\(event, listener\)](#)
 - [emitter.once\(event, listener\)](#)
 - [emitter.removeListener\(event, listener\)](#)

THE HTTP SERVER CLASS...

Class: http.Server

This is an **EventEmitter** with the following events:

Event: 'request'

```
function (request, response) { }
```

Emitted each time there is a request. Note that there may be multiple connections). **request** is an instance of **http.IncomingMessage**

...is an **EventEmitter**

STREAM

<https://nodejs.org/api/stream.html>

```
var readable = getReadableStreamSomehow();
readable.on('readable', function() {
  var chunk;
  while (null !== (chunk = readable.read())) {
    console.log('got %d bytes of data', chunk.length);
  }
});
```

STANDARD I/O

- `stdin` is a Readable Stream,
- `stdout` is a Writable Stream.

```
process.stdin.setEncoding('utf8');

process.stdin.on('readable', function() {
  var chunk = process.stdin.read();
  if (chunk !== null) {
    process.stdout.write('data: ' + chunk);
  }
});

process.stdin.on('end', function() {
  process.stdout.write('end');
});
```

READABLE STREAM

Can read data in "flowing mode" or "paused mode"

Streams start in paused mode.

- Paused mode :
 - call `stream.read()` to get chunks of data
- Flowing mode can be activated:
 - by adding a 'data' event handler
 - by piping (`pipe()`) the input to a destination
 - by calling `resume()`
- Flowing mode can be deactivated:
 - by removing any 'data' event handler and pipe destinations
 - by calling `pause()` if there are no pipe

In flow mode, if no data handler is attached and if no pipe destination is set, data may be lost.

READABLE STREAM EVENTS

- `readable` : when a chunk of data can be read
- `data` : in flowing mode (by attaching a 'data' event listener), data will be passed as soon as it is available
- `end` : when no more data to read
- `close` : for streams that are closeable
- `error`

WRITABLE STREAM

- `write()`
- `drain()` : indicate when it is appropriate to begin writing more data to the stream.
- `cork()` / `uncork()` : bufferize / flush data
- `end()` :
- Events : `finish`, `pipe`, `unpipe`, `error`

HTTP STREAM

```
var http = require('http');

var server = http.createServer(function (req, res) {
  // req is an http.IncomingMessage, which is a Readable Stream
  // res is an http.ServerResponse, which is a Writable Stream

  var body = '';
  // we want to get the data as utf8 strings
  // If you don't set an encoding, then you'll get Buffer objects
  req.setEncoding('utf8');

  // Readable streams emit 'data' events once a listener is added
  req.on('data', function (chunk) {
    body += chunk;
  });

  // the end event tells you that you have entire body
  req.on('end', function () {
    try {
      var data = JSON.parse(body);
    } catch (er) {
      // uh oh! bad json!
      res.statusCode = 400;
      return res.end('error: ' + er.message);
    }

    // write back something interesting to the user:
    res.write(typeof data);
    res.end();
  });
});

server.listen(1337);
```

SOCKET & PIPE

A TCP server which listens on port 1337 and echoes whatever you send it:

```
var net = require('net');

var server = net.createServer(function (socket) {
  socket.write('Echo server\r\n');
  socket.pipe(socket);
});

server.listen(1337, '127.0.0.1');

$ telnet 127.0.0.1 1337
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^].
Echo server
Yohoo !
Yohoo !
Is there anybody there ??
Is there anybody there ??
```

FILESYSTEM

<https://nodejs.org/api/fs.html>

Contains all the expected material for naming files, listing directories (`fs.readdir()`), reading and writing files.

```
fs.readFile('/etc/passwd', function (err, data) {
  if (err) throw err;
  console.log(data);
});
```

Readable and Writable stream are available with :

- `createReadStream()`

```
fs.createReadStream(
  'sample.txt',
  {start: 90, end: 99}
);
```
- `createWriteStream()`

EXERCICE : SHELL

Implement 3 shell commands : pwd, ls, wget

```
// your code here shell.js
```

Steps :
input is a Buffer

```
foo  
<Buffer 61 73 64 0a>  
bar  
<Buffer 62 61 72 0a>
```

→ need to stringify

1. log stdin input on console
2. stringify the stdin buffer
3. match the first word with a regexp
4. implement the 'pwd' command : use 'process'
5. implement 'ls' with 'fs.readdir()'
6. implement 'wget' with 'http.get()'

HTTP

Low-level Web APIs

```
var http = require('http');  
var url = require('url');  
var qs = require('querystring');  
  
• https://nodejs.org/api/url.html  
• https://nodejs.org/api/querystring.html  
  
> var qs = require('querystring')  
undefined  
> qs.parse('foo=bar&baz=qux&baz=quux&corge')  
{ foo: 'bar', baz: [ 'qux', 'quux' ], corge: '' }
```

URL

```
> var url = require('url')
undefined
> url.parse('http://localhost:3000/path/to/file?q=1#here')
{ protocol: 'http:',
  slashes: true,
  auth: null,
  host: 'localhost:3000',
  port: '3000',
  hostname: 'localhost',
  hash: '#here',
  search: '?q=1',
  query: 'q=1',
  pathname: '/path/to/file',
  path: '/path/to/file?q=1',
  href: 'http://localhost:3000/path/to/file?q=1#here' }
```

EXERCICE : A SIMPLE WEB APPLICATION

PROCESSING THE URL

1. Examine the API of the URL module :

<https://nodejs.org/api/url.html>

2. Get the previous sample `helloworld.js`

1. write the pathname to the console

2. modify it to write back the parts of the URL

3. get

`http://localhost:1337/path/to/page.html&c=d`

4. what do you notice ?

DISPLAY PAGES

1. Display a specific page for the following paths :
 1. / : a welcome page
 2. /today : display the current date
 3. /about : an about page
2. /about.html : supply a static file

POPULATE A TEMPLATE

1. Given the following template :

```
infos.html
<html>
  <head>
    <title>My Node.js server</title>
  </head>
  <body>
    <h1>Hello {name}!</h1>
    <ul>
      <li>Node Version: {node}</li>
      <li>V8 Version: {v8}</li>
      <li>URL: {url}</li>
      <li>Time: {time}</li>
    </ul>
  </body>
</html>
```

2. return it populated at

```
/infos.html?name=Bob
```

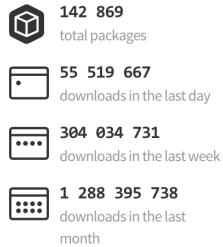
MODULES

MODULES

- Node.js features are available in modules
- Node.js is shipped with built-in modules
- Public modules are available with npm
 - HTML templates
 - Database connectors
 - Dev tools
 - ...

NPM

npm is the package manager for
javascript.



Public repo : <https://www.npmjs.com/>

UNDERSTANDING REQUIRE()

Buit-in modules :

```
var http = require('http'); // get http.js
var url = require('url'); // get url.js
```

Third-party modules :

```
var http = require('module1'); // get [paths]/node_modules/module1
var url = require('./module2'); // get module2.js from the current directory
var url = require('../module3'); // get module3.js from the parent directory
```

Lookup paths :

```
paths:
[ '/Users/bill/devnode/myproject/node_modules',
  '/Users/bill/devnode/node_modules',
  '/Users/bill/node_modules',
  '/Users/node_modules',
  '/node_modules' ]
```

HOW DO MODULES WORK ?

You have to export public members, others will be kept private.

```
var exports = {};
(function() {
    var a = 10;
    exports.foo = a * 10;
})();
console.log(exports.a); // undefined
console.log(exports.foo); // 100
```

fooModule.js

Everything around show how your module is wrapped.

```
var foo = require('./fooModule');
console.log(foo.a); // undefined
console.log(foo.foo); // 100
```

REQUIRE JSON DATA

```
{
    "poll-frequency": 10,
    "urls": ["test.com", "test.org"],
    "data": { "name": "test", "encoding": "utf8" }
}
```

config.json

REPL:

```
> var config = require('./config')
undefined
> config.urls
[ 'test.com', 'test.org' ]
```

EXERCICE : DESIGN A MODULE

1. Create a file `shapes/circles.js`
2. Define the functions `area` and `circumference`, and export them.
3. Create the project descriptor with `npm init`
4. Create a file `circlesTest.js` that use the module for computing the area of a circle with a specific radius.

MODULE INHERITANCE

You want your module inherit of `EventEmitter`?

```
myStream.js
var util = require("util");
var EventEmitter = require('events').EventEmitter;

// Define the constructor for your derived "class"
function MyStream() {
    // call the super constructor to initialize `this`
    EventEmitter.call(this);
    // your own initialization of `this` follows here
}

// Declare that your class should use EventEmitter as its prototype
// This is roughly equivalent to:
//     MyStream.prototype = Object.create(EventEmitter.prototype)
util.inherits(MyStream, EventEmitter);

// Your events
MyStream.prototype.write = function(data) {
    this.emit("data", data);
}

exports.MyStream = MyStream;
```

This is how the `HTTP` class is also an `EventEmitter`.

MODULE INHERITANCE : USAGE

```
myStreamTest.js
var MyStream = require("./myStream").MyStream;
var EventEmitter = require('events').EventEmitter;

var stream = new MyStream();

console.log(stream instanceof EventEmitter); // true
console.log(MyStream.super_ === EventEmitter); // true

stream.on("data", function(data) {
  console.log('Received data: "' + data + '"');
})
stream.write("It works!"); // Received data: "It works!"
```

Similarly, you can extend a Readable or Writable stream : <https://nodejs.org/api/stream.html> (see § API for Stream Implementors)

PUBLISH A MODULE

- `npm adduser` : create a user on the npm repo
- Ensure you have :
 - `package.json` : at least with name, version and dependencies
 - `README.md` : a more detailed presentation of your module, the documentation of your API, some tutorials, etc
- `npm publish` : publish your module

LOCAL VS GLOBAL

Install globally with `npm install -g whatever`

- `require('whatever')` ? → install local
- use in the shell CLI ? → install global, binaries will end up in PATH env var
 - ! Global modules can't be include in your projects with `require()`

INSTALL GLOBAL MODULES

- Global install will drop :
 - modules in `/usr/local/lib/node_modules/`,
 - executables in `/usr/local/bin/`
 - and man pages in `/usr/local/share/man/`

```
$ npm install -g grunt-cli
npm ERR! Please try running this command again as root/Administrator.

$ sudo npm install -g grunt-cli
Password:
/usr/local/bin/grunt -> /usr/local/lib/node_modules/grunt-cli/bin
/grunt
grunt-cli@0.1.13 /usr/local/lib/node_modules/grunt-cli
└── resolve@0.3.1
  ├── nopt@1.0.10 (abbrev@1.0.5)
  └── findup-sync@0.1.3 (lodash@2.4.1, glob@3.2.11)

$ npm list -g | grep grunt
└─ grunt-cli@0.1.13
```

EXERCICE : USE A GLOBAL MODULE

- install markdown globally
- find in the doc the executable that transform markdown to HTML and run it
- install markdown locally
- display this HTML content to the console (REPL):

```
<p>A <strong>markdown</strong> para !</p>
```

INSTALL LOCAL MODULES

- Local install will drop:
 - modules in `./node_modules/`,
 - executables in `./node_modules/.bin/`,
 - man page won't be installed at all

```
$ npm install bcrypt
$ npm install async --save
$ npm install mocha-unfunk-reporter --save-dev
```

`--save` and `--save-dev` will update `package.json`

```
$ npm install
```

...will install modules declared in `package.json` of an existing project.

MANAGING DEPENDENCIES

- Runtime and dev dependencies can be managed in the project descriptor file `package.json`
- `npm init` : create `package.json`
- `npm update` : update dependencies according to "semver" (see later)
 - To restore a missing npm, use the command:
 - `curl -L https://npmjs.com/install.sh | sh`

Dependencies can refer GIT repositories

- `npm search postgresql` : searching a module, may take a while...

PACKAGE.JSON

Interactive description :

<http://browsenpm.org/package.json>

```
{  
  "name": "module-name",  
  "version": "10.3.1",  
  "description": "An example module to illustrate the usage of a  
  "author": "Your Name <you.name@example.org>",  
  "contributors": [{  
    "name": "Foo Bar",  
    "email": "foo.bar@example.com"  
  }],  
  "bin": {  
    "module-name": "./bin/module-name"  
  },  
  "scripts": {  
    "test": "vows --spec --isolate",  
    "start": "node index.js",  
    "predeploy": "echo im about to deploy",  
    "postdeploy": "echo ive deployed",  
    "prepublish": "coffee --bare --compile --output lib/foo src/f  
  },  
  "main": "lib/foo.js",  
  "repository": {  
    "type": "git",  
    "url": "https://github.com/nodejitsu/browsenpm.org"  
  },  
  "bugs": {  
    "url": "https://github.com/nodejitsu/browsenpm.org/issues"  
  },  
  "keywords": [  
    "nodejitsu",  
    "example",  
    "browsenpm"  
  ],  
  "dependencies": {  
    "primus": "*",  
    "async": "~0.8.0",  
    "express": "4.x",  
    "winston": "git://github.com/flatiron/winston#master",  
    "bigpipe": "bigpipe/pagelet",  
    "plates": "https://github.com/flatiron/plates/tarball/master"  
  },  
  "devDependencies": {  
    "vows": "^0.7.0",  
    "assume": "<1.0.0 || >=2.3.1 <2.4.5 || >=2.5.2 <3.0.0",  
    "pre-commit": "*"  
  },  
  "preferGlobal": true,  
  "private": true,  
  "publishConfig": {  
    "registry": "https://your-private-hosted-npm.registry.nodejit  
  },  
  "subdomain": "foobar",  
  "analyze": true,  
  "license": "MIT"  
}
```

SEMANTIC VERSIONING "SEMVER"

```
{  
  "name": "myApp",  
  "version": "1.2.3",  
  "dependencies": {  
    "markdown": "~0.4"  
  }  
}  
"version" : "1.2.3",
```

- **1** : major version, incompatibility
- **2** : minor version, probably lost of compatibility
- **3** : patch, preserve compatibility

"SEMVER" RANGES

Used to manage versions of dependencies

- Hyphen range : `1.2.3 - 2.3.4` $\Rightarrow 1.2.3 \leq v \leq 2.3.4$
- x range : `1.2.x` $\Rightarrow 1.2.0 \leq v < 1.3.0$
- Tilde range : `~1.2.3` $\Rightarrow 1.2.3 \leq v < 1.(2+1).0 \Rightarrow 1.2.3 \leq v < 1.3.0$
- Caret range : `^1.2.3` $\Rightarrow 1.2.3 \leq v < 2.0.0$

<https://docs.npmjs.com/misc/semver>

REST WEB APP WITH EXPRESS

THE EXPRESS FRAMEWORK

- REST-style
- routes can be chained
- easy capture path fragments
- easy capture query parameters
- error handling
- templating
- serving static files
- plugins ("middleware")

Switch to Express :

```
$ npm install express
$ npm install express-params
```

BEFORE

```
if (page == '/') {
    res.end('Welcome !');
} else if (page == '/today') {
    res.end('Today : ' + new Date());
} else if (page == '/about') {
    res.end('This web application runs on Node.js');
}
```

AFTER

```
var app = require('express')();
var params = require('express-params');
params.extend(app);

app.get('/', function(req, res) {
    res.end('Welcome !');
});
app.get('/today', function(req, res) {
    res.end('Today : ' + new Date());
});
app.get('/about', function(req, res) {
    res.end('This web application runs on Node.js with Express');
});
app.listen(1337);
```

REST

HTTP/REST	CRUD	SQL
POST	Create	INSERT
GET	Read	SELECT
PUT	Update	UPDATE
DELETE	Delete	DELETE

ROUTING

ROUTE METHOD

```
// respond with "Hello World!" on the homepage
app.get('/', function (req, res) {
  res.send('Hello World!');
});
// accept POST request on the homepage
app.post('/', function (req, res) {
  res.send('Got a POST request');
});
// accept PUT request at /user
app.put('/user', function (req, res) {
  res.send('Got a PUT request at /user');
});
// accept DELETE request at /user
app.delete('/user', function (req, res) {
  res.send('Got a DELETE request at /user');
});
app.all('/secret', function (req, res) {
  console.log('Accessing the secret section ...');
});
```

ROUTE PATHS

Routes are tested one after the other until one matches.

```
app.get('/path', function (req, res) {
  res.send('Hello World!');
});
```

- `'/ab?cd'` // will match acd and abcd
- `'/ab+cd'` // will match abcd, abbcd, abbbcd, and so on
- `'/ab*cd'` // will match abcd, abxcd, abRABDOMcd, ab123cd, and so on
- `'/ab(cd)?e'` // will match /abe and /abcde
- `/a/` // will match anything with an a in the route name
- `/>.*/fly$/` // will match butterfly, dragonfly; but not butterflyman, dragonfly man, and so on

EXTRACT DATA

```
app.param('uid', /^[0-9]+$/);

app.get('/user/:uid', function(req, res, next){
  var uid = req.params.uid;
  res.send('user ' + uid);
});

app.get('/user/:name', function(req, res, next){
  var name = req.params.name;
  res.send('user ' + name);
});
```

ROUTE HANDLERS

```
app.get('/path', function (req, res) {
  res.send('Hello World!');
});
```

A route can be handled using more than one callback function, specify the `next` object.

```
app.get('/example/b', function (req, res, next) {
  console.log('response will be sent by the next function ...');
  next();
}, function (req, res) {
  res.send('Hello from B!');
});
```

CHAINING ROUTES

```
app.route('/book')
  .get(function(req, res) {
    res.send('Get a random book');
  })
  .post(function(req, res) {
    res.send('Add a book');
  })
  .put(function(req, res) {
    res.send('Update the book');
  });
});
```

EXPRESS MIDDLEWARES

An Express application is essentially a series of middleware calls. The next middleware in line in the request-response cycle is accessible by the `next` object.

```
// a middleware sub-stack which handles GET requests to /user/:id
app.get('/user/:id', function (req, res, next) {
  console.log('ID:', req.params.id);
  next();
}, function (req, res, next) {
  res.send('User Info');
});

// handler for /user/:id which prints the user id
app.get('/user/:id', function (req, res, next) {
  res.end(req.params.id);
});
```

ERROR HANDLING

Hanler with 4 arguments will handle errors.

```
app.use(function(err, req, res, next) {
  console.error(err.message);
  console.error(err.stack);
  res.status(500).send('Something broke!');
});

app.post('/some_path', function(req, res, next) {
  var data = req.body.some_data;
  if (typeof data == 'undefined') {
    next(new Error("Please fill the form with data"));
  } else {
    // ...
  }
});
```

SERVING STATIC FILES

`express.static` is the only built-in middleware in Express

```
var options = {
  dotfiles: 'ignore',
  etag: false,
  extensions: ['htm', 'html'],
  index: false,
  maxAge: '1d',
  redirect: false,
  setHeaders: function (res, path, stat) {
    res.set('x-timestamp', Date.now());
  }
}
app.use(express.static('public', options));
app.use(express.static('uploads'));
app.use(express.static('files'));
```

OTHER MIDDLEWARE

```
$ npm install cookie-parser

var express = require('express');
var app = express();
var cookieParser = require('cookie-parser');

// load the cookie parsing middleware
app.use(cookieParser());



- body-parser : parse json and urlencoded bodies
- express-session
- cookie-parser
- passport : authentication
- serve-favicon
- multer : upload files
- ...

http://expressjs.com/resources/middleware.html
```

TEMPLATING

Template engines can be plugged to Express : Jade,
Mustache, Handlebars, EJS...

```
$ npm install express-handlebars

var express = require('express');
var exphbs = require('express-handlebars');
var app = express();

var hbs = exphbs.create({
  // Specify helpers which are only registered on this instance
  helpers: {
    foo: function () { return 'FOO!'; },
    bar: function () { return 'BAR!'; }
  }
});
app.engine('handlebars', hbs.engine);
app.set('view engine', 'handlebars');

app.get('/', function (req, res) {
  res.render('home', {
    showTitle: true,
    // Override `foo` helper only for this rendering.
    helpers: {
      foo: function () { return 'foo.'; }
    }
  });
});
app.listen(3000);
```

TEMPLATING : HANDLEBARS

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Example App - Home</title>
  </head>
  <body>
    <!-- Uses built-in `if` helper. -->
    {{#if showTitle}}
      <h1>Home</h1>
    {{/if}}
    <!-- Calls `foo` helper, overridden at render-level. -->
    <p>{{foo}}</p>
    <!-- Calls `bar` helper, defined at instance-level. -->
    <p>{{bar}}</p>
  </body>
</html>
```

Loops :

```
<ul>
  {{#each user}}
    <li>{{name}}</li>
  {{/each}}
</ul>
```

EXERCICE : A REST WEB APP

Store items in the current session.

AWESOME TASK LIST

Don't forget

- ✘ Feed the cat
- ✘ Remember to get more milk
- ✘ Wake up at 06:00
- ✘ Learn Node.js

What do you need to do ?

SOCKET.IO

Socket.io : real-time events in the network

Based on WebSocket, as an HTTP workaround (the server can't push messages to the client in HTTP),
allow broadcasting.



Socket.io supply fallback mechanisms if necessary :

- Adobe Flash Socket
- AJAX long polling
- AJAX multipart streaming
- Forever Iframe
- JSONP Polling

USING SOCKET.IO

```
$ npm install socket.io
var http = require('http');
var fs = require('fs');

var server = http.createServer(function(req, res) {
  fs.readFile('./index.html', 'utf-8', function(error, content)
    res.writeHead(200, {"Content-Type": "text/html"});
    res.end(content);
  });
});

var io = require('socket.io').listen(server);

io.sockets.on('connection', function (socket) {
  console.log('Client connection received.');
});

server.listen(1337);
```

SOCKET.IO CLIENT

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Socket.io</title>
  </head>
  <body>
    <h1>Communication avec socket.io !</h1>

    <script src="/socket.io/socket.io.js"></script>
    <script>
      var socket = io();
    </script>
  </body>
</html>
```

index.html

/socket.io/socket.io.js is handled directly by
Socket.io

```
$ node socket.js
```

Set the DEBUG env var to see it in action :

```
$ DEBUG=* node socket.js
```

SHOW THE CODE RUNNING IN CHROME and open
the dev tool to see network exchanges

SOCKET.IO SERVER API

```
io.sockets.on('connection', function (socket) {
  // emit a message to a client :
  socket.emit('message', 'You are connected !');
  // receive a message from a client :
  socket.on('message', function (message) {
    // send a broadcast message (to all other clients)
    socket.broadcast.emit('message', 'A client say ' + message);
  });
  socket.on('disconnect', function () {
    // emit a message to all :
    io.emit('message', 'A client left');
  })
});
```

SOCKET.IO CLIENT API

```
<script>
var socket = io.connect('http://localhost:8080');
socket.emit('entering', name);
socket.on('message', function(message) {
  alert(message);
})
</script>
```

Can also manage "rooms" and endpoints (paths).

EXERCICE : A CHAT

- connect with a user name
- send broadcast messages posted by users

Start here : <http://socket.io/get-started/chat/>

USING JQUERY

```
<script src="http://code.jquery.com/jquery-1.11.1.js"></script>

<form action="">
  <input id="text" autocomplete="off" />
  <button>Send</button>
</form>
<ul id="discussion"></ul>

$('#discussion').append($('- ').text(message));

$('form').submit(function(){
  // your code here
  return false;
});

$('#text').val()

$('#text').val('')

```

DATA ACCESS

MYSQL

```
$ npm install mysql
```

CONNECTION

```
var mysql = require('mysql');
var connection = mysql.createConnection({
  host      : 'localhost',
  port      : 3306,
  user      : 'bob',
  password  : 'secret',
  database  : 'address_book'
});

connection.connect(function(err) {
  if (err) {
    console.error('error connecting: ' + err.stack);
    return;
  }
  console.log('connected as id ' + connection.threadId);
});
```

SQL QUERIES

```
var mysql = require('mysql');
connection.query({
  sql: 'SELECT * FROM `books` WHERE `author` = ?',
  timeout: 40000, // 40s
  values: ['David']
}, function (error, results, fields) {
  // fields will contain information about the returned results
});
```

ESCAPING

```
var sorter = 'date';
var sql = 'SELECT * FROM posts WHERE id > '
+ connection.escape(userId); // escape values
+ ' ORDER BY '
+ connection.escapeId(sorter); // escape SQL identifiers

?? : placeholder for SQL identifiers

var sql = "SELECT * FROM ?? WHERE ?? = ?";
var inserts = ['users', 'id', userId];
sql = mysql.format(sql, inserts);
```

GETTING QUERY INFOS

```
connection.query('INSERT INTO posts SET ?', {title: 'test'}, function (err) {
  if (err) throw err;
  // ID of an inserted row
  console.log(result.insertId);
});

connection.query('DELETE FROM posts WHERE title = "wrong"', function (err) {
  if (err) throw err;
  // number of affected rows
  console.log('deleted ' + result.affectedRows + ' rows');
})

connection.query('UPDATE posts SET ...', function (err, result) {
  if (err) throw err;
  // number of changed rows
  console.log('changed ' + result.changedRows + ' rows');
})
```

OTHER FEATURES

CONNECTION POOL

```
var mysql = require('mysql');
var connection = mysql.createPool({
  connectionLimit : 10,
  host : 'localhost',
  port : 3306,
  user : 'bob',
  password : 'secret',
  database : 'address_book'
});
```

TRANSACTION MANAGEMENT: COMMIT, ROLLBACK

```
connection.beginTransaction(function(err) {
  if (err) { throw err; }
  connection.query('INSERT INTO posts SET title=?', title, function (err) {
    if (err) {
      connection.rollback(function() {
        throw err;
      });
    }
  });
  // ...
})
```

EXERCICE : MYSQL REST

Manage a set of users in MySQL

Api	Type	Description
/users	POST	Takes name, email and password as input data and add new user.
/users	GET	Returns every users
/users/:userId	GET	Returns users with match of userId.
/users/:userId/	PUT	Update email of user of this ID.
/users/:email	DELETE	Delete user from database.

Don't design the UI, just the server-side REST API

Submit your request with the help of POSTMAN REST client on chrome

Passwords can be hashed with the `crypto` module :

https://nodejs.org/api/crypto.html#crypto_crypto_createhash_algorithm

MONGODB

```
$ mkdir -p ./data/db  
$ mongod --dbpath=./data/db --port 27017
```

Store schemaless JSON documents

Flexible, scalable, high performance
and availability



```
$ npm install mongodb
```

CONNECTION

```
var mongodb = require('mongodb');  
var MongoClient = mongodb.MongoClient;  
var url = "mongodb://localhost:27017/myproject";  
// Use connect method to connect to the Server  
MongoClient.connect(url, function doConnect(err, db) {  
    var coll = db.collection("myusers");  
    // do something...  
    db.close();  
});
```

JSON documents are organized in collections

INSERTION

An `_id` field will be added for each document (except if it exist)

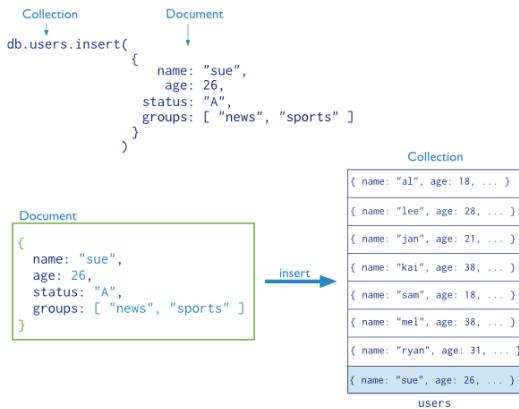
```
// get our users data
var users = require("./data/users.json");
// get the "myusers" collection
db.collection("myusers", function (err, collection) {
  if (err) return callback(err);
  // insert the users
  collection.insert(users, callback);
});
```

BULK LOADING

Large number of insertions have to be performed with bulk operations.

```
var DATA = require("data/users.json");
var coll = db.collection("myusers");
var bulk = coll.initializeUnorderedBulkOp();
for (var i = 0; i < DATA.length; i++) {
  coll.insert(DATA[i]);
}
```

INSERTION



EXERCICE : MONGODB BULK LOADING

Data set : stats about cities by département

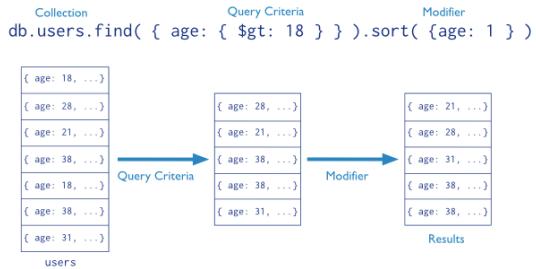
- Load with bulk insertion this data set:
<http://public.opendatasoft.com/explore/dataset/code-insee-postaux-geoflar/>
- Use Robomongo for browsing data.

COLLECTIONS

```
db.collection("myusers", function (err,coll) {  
    coll.stats(function doStats(err, stats) {  
        console.log(stats);  
        db.close();  
    });  
});  
  
db.dropCollection("myusers");
```

QUERYING

```
coll.find(  
  { age: { $gt: 18 } },  
  { name: 1, address: 1 }  
) .limit(5)
```



COMPARISON WITH SQL

```
db.users.find(  
  { age: { $gt: 18 } },  
  { name: 1, address: 1 }  
) .limit(5)
```

collection
query criteria
projection
cursor modifier

```
SELECT _id, name, address  
FROM users  
WHERE age > 18  
LIMIT 5
```

projection
table
select criteria
cursor modifier

QUERYING CONDITIONS

```
coll.find(); // find all  
  
coll.find( { type: "snacks" } ); // equality  
  
coll.find( { type: { $in: [ 'food', 'snacks' ] } } ); // enum  
  
coll.find( { type: 'food', price: { $lt: 9.95 } } ); // AND  
  
.find(  
  {  
    $or: [ { qty: { $gt: 100 } }, { price: { $lt: 9.95 } } ]  
  }  
)
```

UPDATE

Update the data and the schema as well

```
coll.update(  
  { item: "MNO2" },  
  {  
    $set: {  
      category: "apparel",  
      details: { model: "14Q3", manufacturer: "XYZ Company"  
    },  
    $currentDate: { lastModified: true }  
  }  
)
```

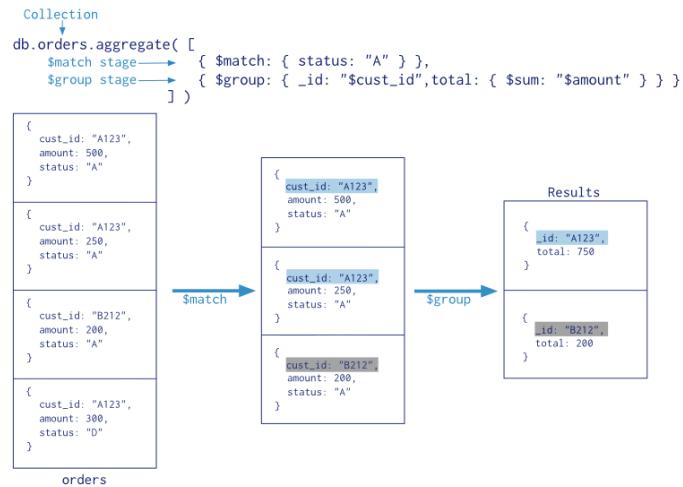
UPDATE FROM THE CONSOLE

```
> db.users.insert({name: 'Philippe', country: 'France'});
WriteResult({ "nInserted" : 1 })
> var me = db.users.findOne({name: 'Philippe'});
> me
{
  "_id" : ObjectId("556d6b030e9d920f8c6b336d"),
  "name" : "Philippe",
  "country" : "France"
}
> me.eyesColor = 'Blue'
Blue
> db.users.save(me);
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.users.findOne();
{
  "_id" : ObjectId("556d6b030e9d920f8c6b336d"),
  "name" : "Philippe",
  "country" : "France",
  "eyesColor" : "Blue"
}
```

DELETE

```
coll.remove( {} ); // remove all
coll.remove( { type : "food" } );
coll.remove( { type : "food" }, 1 ); // remove 1
```

MONGODB AGGREGATION FRAMEWORK



STAGE OPERATORS

- `$match`
- `$group`: note that grouping with `_id : null` makes a single group
- `$project`: reshapes the documents in the stream, such as by adding new fields or removing existing fields.
- `$limit` and `$skip`
- `$sort`
- `$unwind`: flatten an array field
- ...

ACCUCUMULATOR

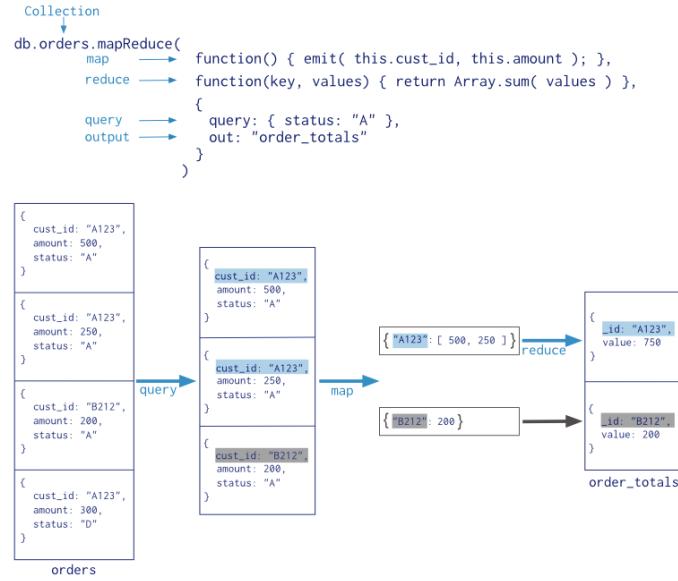
- `$sum`, `$avg`, `$min`, `$max`, `$first`, `$last`
- `$push`: reverse of `$unwind`
- ...

- `$match`

SINGLE PURPOSE AGGREGATION OPERATIONS

```
coll.count( { a: 1 } ); // count records under condition  
  
coll.distinct( "b" ); // select distinct records  
  
coll.group( {  
    key: { a: 1 },  
    cond: { a: { $lt: 3 } },  
    reduce: function(cur, result) { result.count += cur.count },  
    initial: { count: 0 }  
} );
```

MONGODB MAP-REDUCE



EXERCICE : MONGODB AGGREGATION

With the same data set :

1. return département with population > 1000000
2. return average city population by département
3. return largest and smallest city by département

Note : duplicate fields (such as population) exist for the same fields.code_commune_insee

DB MAPPING

SQL : ORM

```
var orm = require("orm");
orm.connect("mysql://username:password@host/database", function (
  var Person = db.define("person", {
    name      : String,
    surname   : String
    age       : Number, // FLOAT
    male      : Boolean
  });
  Person.find({ surname: "Doe" }, function (err, people) {
    // SQL: "SELECT * FROM person WHERE surname = 'Doe'"
    console.log("First person: %s, age %d", people[0].fullName);
    people[0].age = 16;
    people[0].save(function (err) { });
  });
});
```

MONGO : MONGOOSE

```
var mongoose = require('mongoose');
mongoose.connect('mongodb://localhost/test');
var Cat = mongoose.model('Cat', { name: String });
var kitty = new Cat({ name: 'Zildjian' });
kitty.save(function (err) { });
```

TOOLS

DEBUG

Node.js built-in debugger

```
$ node debug myscript.js  
myscript.js  
x = 5;  
setTimeout(function () {  
  debugger;  
  console.log("world");  
, 1000);  
console.log("hello");
```

Commands :

- cont, c - Continue execution
- next, n - Step next
- step, s - Step in
- out, o - Step out
- pause - Pause running code

ISSUES

Logs:

```
process.nextTick(function() { throw err; })
^
AssertionError: null == { [MongoError: connect ECONNREFUSED]
  name: 'MongoError', message: 'connect ECONNREFUSED' }
```

Sometimes you don't have the message

```
/path/to/node_modules/mongodb/lib/utils.js:97
  process.nextTick(function() { throw err; });
^
Error
  at Object.<anonymous> (/path/to/node_modules/mongodb/node_modules/mongodb-core/lib/error.js:42:24)
    at Module._compile (module.js:460:26)
    at Object.Module._extensions..js (module.js:478:10)
    at Module.load (module.js:355:32)
    at Function.Module._load (module.js:310:12)
    at Module.require (module.js:365:17)
    at require (module.js:384:17)
    at Object.<anonymous> (/path/to/node_modules/mongodb/node_modules/mongodb-core/index.js:2:17)
      at Module._compile (module.js:460:26)
      at Object.Module._extensions..js (module.js:478:10)
```

TROUBLESHOOTING

The purpose of `uncaughtException` is not to catch and go on, but to allow to free resources and log the error context. When an error is raised to the event loop, the program should be considered inconsistent.

```
process.on('uncaughtException', function(err) {
  console.log(JSON.stringify(process.memoryUsage()));
  console.error("UncaughtException : the program will end. "
    + err + ", stacktrace: " + err.stack);
  return process.exit();
});

console.log("Insert done " + JSON.stringify(DATA[0]));

try {
  var bwr = bulk.execute();
} catch(err) {
  console.log("Stupid Mongo err : " + err);
}
```

PREVENT ERRORS

- Use Node.js clusters : the Master Cluster will be able to restart a slave
- Use a framework like Express, that manages errors for you
- "Promises" can help to manage errors efficiently : errors raised by some code inside a promise will be caught and propagated to the `fail` or `catch` function or in the error callback of the `then` function

TESTING

Testing with Mocha

```
var assert = require('assert');

describe('my module', function() {
  before(function() {
    // Some setup
  });

  it('should do amazing things', function() {
    // Put some assertions in here
    assert.equal(n, 0);
  });

  after(function() {
    // Some tear-down
  });
});
```

Use before, after, beforeEach, afterEach

TESTING WITH MOCHA

```
{  
  "name": "testing",  
  "version": "1.0.0",  
  "description": "Test test",  
  "main": "index.js",  
  "scripts": {  
    "test": "mocha tests --recursive",  
    "testjson": "mocha -R json tests --recursive > target/report/  
  },  
  "keywords": [  
    "test",  
    "mocha"  
,  
  ],  
  "license": "ISC",  
  "devDependencies": {  
    "mocha": "^2.2.4",  
    "mocha-unfunk-reporter": "^0.4.0"  
  }  
}  
  
$ npm test  
Users management  
✓ Should count 0 people for new users  
1) Should count 1 people after addind a user  
  
1 passing (13ms)  
1 failing  
$ npm run testjson
```

EXERCICE : TESTING WITH MOCHA

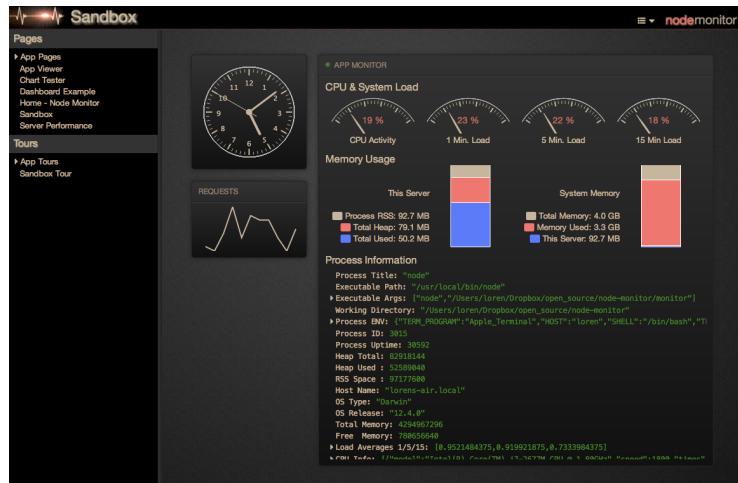
Given the following module :

```
var Users = function() {  
  this.people = [];  
}  
  
Users.prototype.add = function(user) {  
  // add that user  
}  
  
Users.prototype.count =function() {  
  return this.people.length;  
}  
  
module.exports = Users;
```

1. check that the number of new user is 0
2. check that adding a user fail (not implemented)

MONITORING

- <https://github.com/lorenwest/node-monitor>
- <https://github.com/lorenwest/monitor-dashboard>

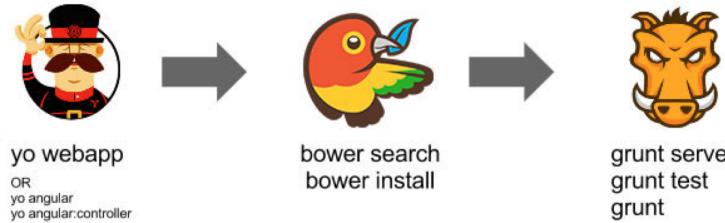


ALM

- requirements management,
- software architecture,
- computer programming,
- software testing,
- software maintenance,
- change management,
- continuous integration,
- project management,
- and release management

FRONTEND TOOLS

- Yeoman : web's scaffolding tool
- Bower : package manager for the Web
- Grunt : automate build tasks
- Gulp : Grunt, the streaming way



This presentation is powered by Grunt

```
$ sudo npm install -g grunt-cli  
$ grunt serve --port 4000
```

Gruntfile.js

```
/* global module:false */  
module.exports = function(grunt) {  
  var port = grunt.option('port') || 8000;  
  // Project configuration  
  grunt.initConfig({  
    pkg: grunt.file.readJSON('package.json'),  
    meta: {  
      banner: '/!*!\n' +  
        ' * reveal.js <%= pkg.version %> (<%= grunt.template.toda  
        ' * http://lab.hakim.se/reveal-js\n' +  
        ' * MIT licensed\n' +  
        ' * \n' +  
        ' * Copyright (C) 2015 Hakim El Hattab, http://hakim.se\n'  
        ' */'  
    },  
    qunit: {  
      files: [ 'test/*.html' ]  
    },  
    uglify: {  
      options: {  
        banner: '<%= meta.banner %>\n'  
      },  
      build: {  
        src: 'js/reveal.js',  
        dest: 'js/reveal.min.js'  
      }  
    },  
    sass: {  
      core: {  
        files: {  
          'css/reveal.css': 'css/reveal.scss',  
        }  
      },  
      themes: {  
        files: {  
          'css/theme/black.css': 'css/theme/source/black.scss',  
          'css/theme/white.css': 'css/theme/source/white.scss',  
          'css/theme/league.css': 'css/theme/source/league.scss',  
          'css/theme/beiae.css': 'css/theme/source/beiae.scss'.  
        }  
      }  
    }  
  });  
};
```

```

    'css/theme/night.css': 'css/theme/source/night.scss',
    'css/theme/serif.css': 'css/theme/source/serif.scss',
    'css/theme/simple.css': 'css/theme/source/simple.scss',
    'css/theme/sky.css': 'css/theme/source/sky.scss',
    'css/theme/moon.css': 'css/theme/source/moon.scss',
    'css/theme/solarized.css': 'css/theme/source/solarize
    'css/theme/blood.css': 'css/theme/source/blood.scss'
  }
},
autoprefixer: {
  dist: {
    src: 'css/reveal.css'
  }
},
cssmin: {
  compress: {
    files: {
      'css/reveal.min.css': [ 'css/reveal.css' ]
    }
  }
},
jshint: {
  options: {
    curly: false,
    eqeqeq: true,
    immed: true,
    latedef: true,
    newcap: true,
    noarg: true,
    sub: true,
    undef: true,
    eqnull: true,
    browser: true,
    expr: true,
    globals: {
      head: false,
      module: false,
      console: false,
      unescape: false,
      define: false,
      exports: false
    }
  },
  files: [ 'Gruntfile.js', 'js/reveal.js' ]
},
connect: {
  server: {
    options: {
      port: port,
      base: '.',
      livereload: true,
      open: true
    }
  }
},
zip: {
  'reveal-js-presentation.zip': [
    'index.html',
    'css/**',
    'js/**',
    'lib/**',
    'images/**',
    'plugin/**'
  ]
},
watch: {
  options: {
    livereload: true
  },
  js: {
    files: [ 'Gruntfile.js', 'js/reveal.js' ],
    tasks: 'js'
  },
  theme: {
    files: [ 'css/theme/source/*.scss', 'css/theme/template
    tasks: 'css-themes'
  },
  css: {
    files: [ 'css/reveal.scss' ],
    tasks: 'css-core'
  },
  html: {
    files: [ 'index.html' ]
  }
}
});
// Dependencies

```

```
    ...
    grunt.loadNpmTasks( 'grunt-contrib-qunit' );
    grunt.loadNpmTasks( 'grunt-contrib-jshint' );
    grunt.loadNpmTasks( 'grunt-contrib-cssmin' );
    grunt.loadNpmTasks( 'grunt-contrib-uglify' );
    grunt.loadNpmTasks( 'grunt-contrib-watch' );
    grunt.loadNpmTasks( 'grunt-sass' );
    grunt.loadNpmTasks( 'grunt-contrib-connect' );
    grunt.loadNpmTasks( 'grunt-autoprefixer' );
    grunt.loadNpmTasks( 'grunt-zip' );

    // Default task
    grunt.registerTask( 'default', [ 'css', 'js' ] );

    // JS task
    grunt.registerTask( 'js', [ 'jshint', 'uglify', 'qunit' ] );

    // Theme CSS
    grunt.registerTask( 'css-themes', [ 'sass:themes' ] );

    // Core framework CSS
    grunt.registerTask( 'css-core', [ 'sass:core', 'autoprefixer' ];

    // All CSS
    grunt.registerTask( 'css', [ 'sass', 'autoprefixer', 'cssmin' ] );

    // Package presentation to archive
    grunt.registerTask( 'package', [ 'default', 'zip' ] );

    // Serve presentation locally
    grunt.registerTask( 'serve', [ 'connect', 'watch' ] );

    // Run tests
    grunt.registerTask( 'test', [ 'jshint', 'qunit' ] );
};

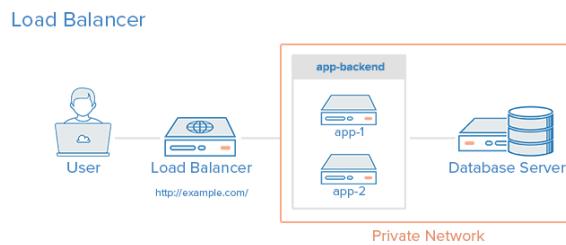
};
```

DEPLOYMENT

SYSTEM INTEGRATION

stability, performance, security, maintainability

health check and balance traffic



- `systemd` (Fedora), `foreverjs`, `pm2` : ensure Node.js will stay running in case of a crash
- `stagecoach` : deploy node.js applications to your staging and production servers
- `n` : node version manager

CLUSTERING

Separate processes → same server port.

A server is down ? → others will be used.

A peak load of traffic ? → allocate another worker.

CLUSTERING : THE APP

app.js

```
var express=require("express");
var app=express();

app.get('/',function(req,res) {
    res.end("Hello world !");
});
app.listen(3000,function() {
    console.log("Running at PORT 3000");
});
```

CLUSTERING : THE LAUNCHER

cluster.js

```
var cluster = require('cluster');
var numCPUs = require('os').cpus().length;

if (cluster.isMaster) {
    for (var i = 0; i < numCPUs; i++) {
        cluster.fork(); // clone the process for each core
    }
    cluster.on('exit', function(worker, code, signal) {
        console.log('worker ' + worker.process.pid + ' died');
    });
} else {
    require("./app.js");
}

$ node cluster.js
```

