Lecture 2: Moving to the server side…

Olivier Liechti
TWEB

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Test alert

Manipulation

Warning

Today's agenda

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

14h00 - 15h00	60'	**Lecture** Introduction to Node.js and npm **Exercise** Deploy a basic Node.js web app on heroku
15h00 - 15h10	*10'*	*Break*
15h10 - 16h25	75'	**Lecture** Introduction to Express.js Introduction to Yeoman (Yo, Grunt) **Exercise** Deploy a basic Express.js app on heroku

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

What are the 3 ways to create JavaScript objects?

JavaScript 101
Reminder

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

#1 Using object literals

```
var student = {
  firstName : "john",
  lastName : "smith"
};
```

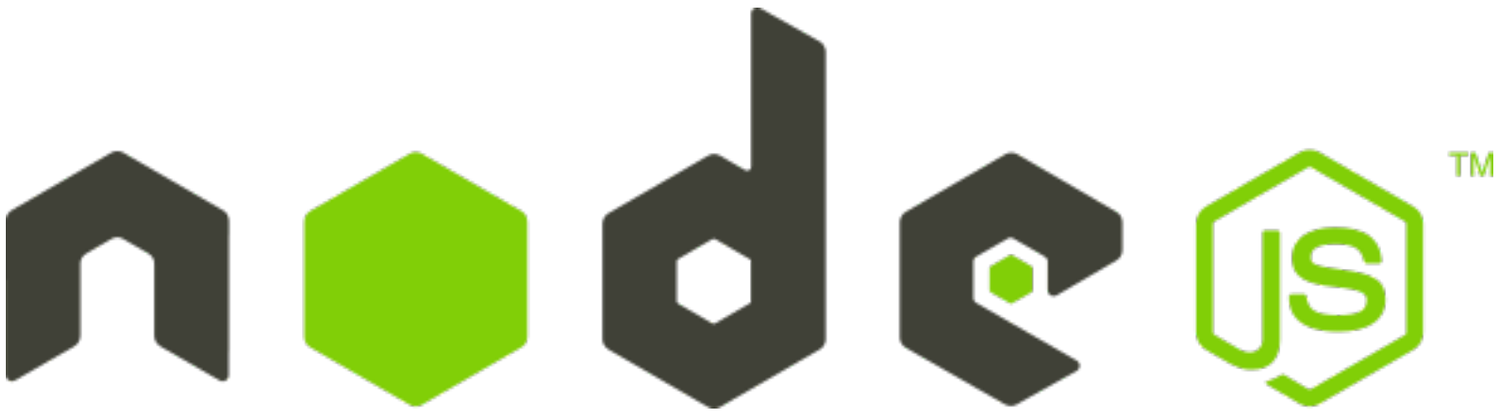
#2 Using other objects as prototypes

```
var father = {...};
var son = Object.create(father);
```

#3 Using constructor functions

```
var Student = function(firstName, lastName) {...};
var student = new Student("john", "smith");
```

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud



Node.js 101

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

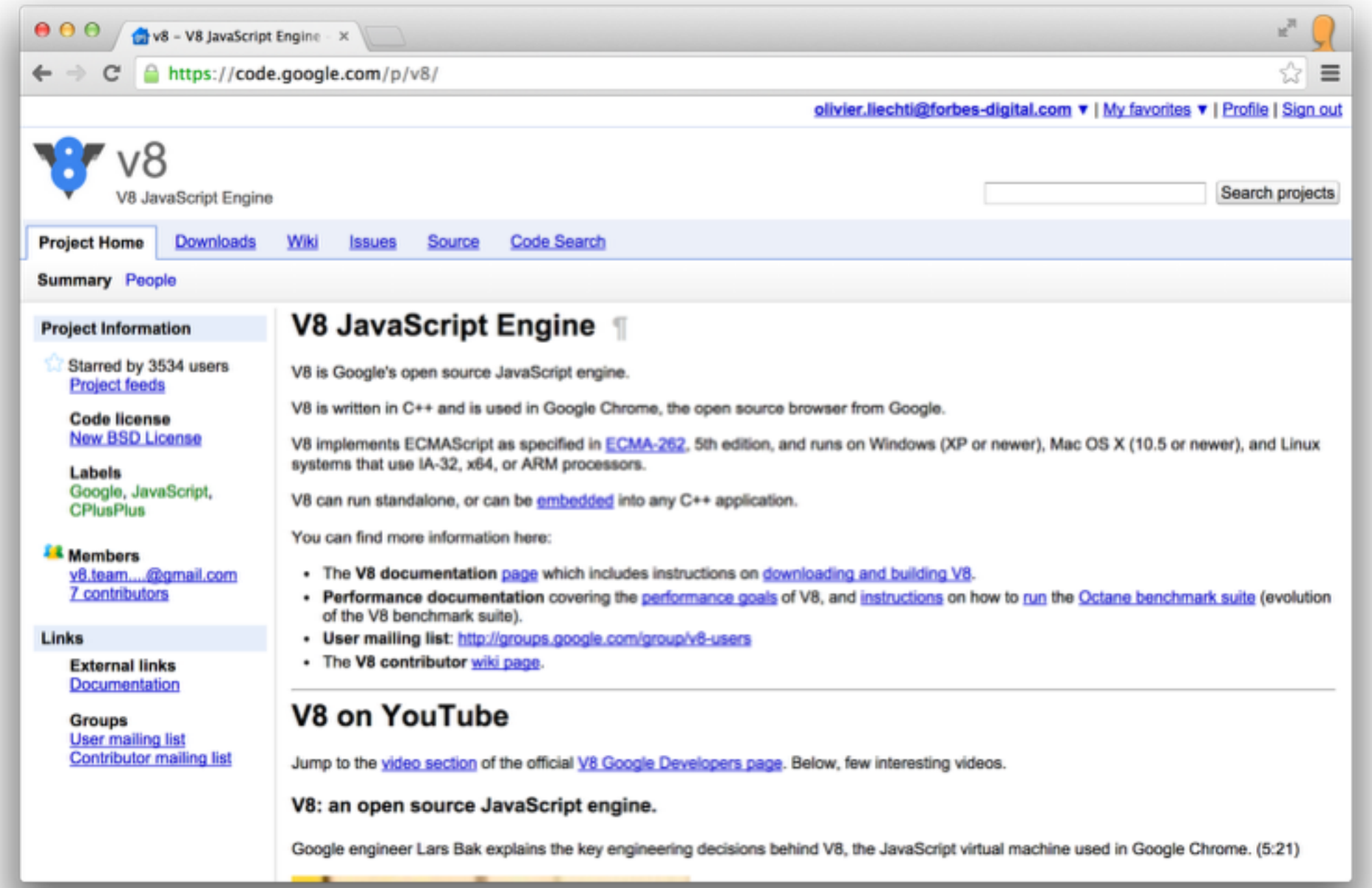
“Node.js is a **platform** built on Chrome's JavaScript runtime for easily building **fast**, **scalable network applications**.

not only!

Node.js uses an **event-driven**, **non-blocking I/O model** that makes it lightweight and efficient, perfect for **data-intensive real-time applications that run across distributed devices**.”

- About these Docs
- Synopsis
- Assertion Testing
- Buffer
- C/C++ Addons
- Child Processes
- Cluster
- Console
- Crypto
- Debugger
- DNS
- Domain
- Events
- File System
- Globals
- HTTP
- HTTPS
- Modules
- Net
- OS
- Path
- Process
- Punycode
- Query Strings
- Readline
- REPL
- Stream
- String Decoder
- Timers
- TLS/SSL
- TTY
- UDP/Datagram
- URL
- Utilities
- VM
- ZLIB

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud



Let's look at a **first example** (**NOT** good)

```
/*global require */

var fs = require("fs");

/**
 * Simple function to test the synchronous readFileSync function provided by
 * Node.js
 * @param {string} filename - the name of the file we want to read in the test
 */
function testSyncRead(filename) {
  console.log("I am about to read a file: " + filename);
  var data = fs.readFileSync(filename);
  console.log("I have read " + data.length + " bytes (synchronously).");
  console.log("I am done.");
}

// We get the file name from the argument passed on the command line
var filename = process.argv[2];

console.log("\nTesting the synchronous call");
testSyncRead(filename);
```

This is for the Brackets editor

We use a standard **Node module** for accessing the file system

fs.readFileSync is synchronous: it blocks the main thread until the data is available.

process is a global object provided by Node.js

```
$ node sample2.js medium.txt

Testing the synchronous call
I am about to read a file: medium.txt
I have read 1024 bytes (synchronously).
I am done.
```

*Synchronous functions are easier to use, but they have **severe** performance implications!!*

Let's look at a **second example** (**BETTER**)

```
/*global require */

var fs = require("fs");

/**
 * Simple function to test the asynchronous readFile function provided by Node.js
 * @param {string} filename - the name of the file we want to read in the test
 */
function testAsyncRead(filename) {
  console.log("I am about to read a file: " + filename);

  fs.readFile(filename, function (err, data) {
    console.log("Nodes just told me that I have read the file.");
  });

  console.log("I am done. Am I really?????");
}
// We get the file name from the argument passed on the command line
var filename = process.argv[2];

console.log("\nTesting the asynchronous call");
testAsyncRead(filename);
```

```
$ node sample2.js medium.txt

Testing the asynchronous call
I am about to read a file: medium.txt
I am done. Am I?
Nodes just told me that I have read the file.
```

fs.readFile is asynchronous: it does not block the main thread until the data is available.

We must provide a **callback function**, which Node.js will invoke when the data is available.

Problems can happen when an (asynchronous) function is called.

*Node.js developers **have to** learn the asynchronous programming style.*

Node.js v0.10.32 Manual & Documentation

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- Assertion Testing
- Buffer
- C/C++ Addons
- Child Processes
- Cluster
- Console
- Crypto
- Debugger
- DNS
- Domain
- Events
- File System
- Globals
- HTTP
- HTTPS
- Modules
- Net
- OS
- Path
- Process
- Punycode
- Query Strings
- Readline
- REPL
- Stream
- String Decoder
- Timers
- TLS/SSL
- TTY
- UDP/Datagram
- URL
- Utilities
- VM
- ZLIB

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

In JavaScript, functions are objects:

They can be passed as arguments.

They can be returned as values.

JavaScript 101
Reminder

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

#1 Functions are objects

```
var aFunction = function() {
  console.log("I am doing my job");
};
aFunction.aProperty = "aValue";
aFunction();
```

#2 Passing functions as arguments

```
var aFunction = function() {...};
var anotherFunction = function( f1 ) {... f1(); ...};
anotherFunction( aFunction );
```

#3 Return functions as values

```
var aFunction = function() {
  return function() {...}
  };
aFunction()();
```

Let's look at a **third example**

```
/*global require */

var http = require("http");

/**
 * This function starts a http daemon on port 9000. It also
 * registers a callback handler, to handle incoming HTTP
 * requests (a simple message is sent back to clients).
 */
function runHttpServer() {
  var daemon = http.createServer();

  daemon.on("request", function (req, res) {
    console.log("A request has arrived: URL=" + req.url);
    res.writeHead(200, {
      'Content-Type': 'text/plain'
    });
    res.end('Hello World\n');
  });

  console.log("Starting http daemon...");
  daemon.listen(9000);
}

runHttpServer();
```

We use a standard **Node module** that takes care of the HTTP protocol.

Node can provide us with a **ready-to-use** server.

We can attach **event handlers** to the server. Node will notify us asynchronously, and give us access to the request and response.

We can **send back** data to the client.

We have wired everything, let's **welcome** clients!

HTTP Node.js v0.10.32 Ma ×

nodejs.org/api/http.html#http_event_request

NPM REGISTRY
DOCS
BLOG
COMMUNITY
LOGOS
JOBS

@nodejs

Node.js v0.10.32 Manual & Documentation

Index | View on single page | View as JSON

Table of Contents

- HTTP
  - 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'
    - Event: 'clientError'
    - server.listen(port, [hostname], [backlog], [callback])
    - server.listen(path, [callback])
    - server.listen(handle, [callback])
    - server.close([callback])
    - server.maxHeadersCount
    - server.setTimeout(msecs, callback)
    - server.timeout
  - Class: http.ServerResponse
    - Event: 'close'
    - Event: 'finish'
    - response.writeContinue()

These are the events that are **emitted** by the class. You can write callbacks and **react** to these events.

How does Node.js use an **event loop** to offer an asynchronous programming model?

```
on('request', function(req, res) { // my code});
```

```
on('data', function(data) { // my code});
```

Callback functions that **you** have written and registered

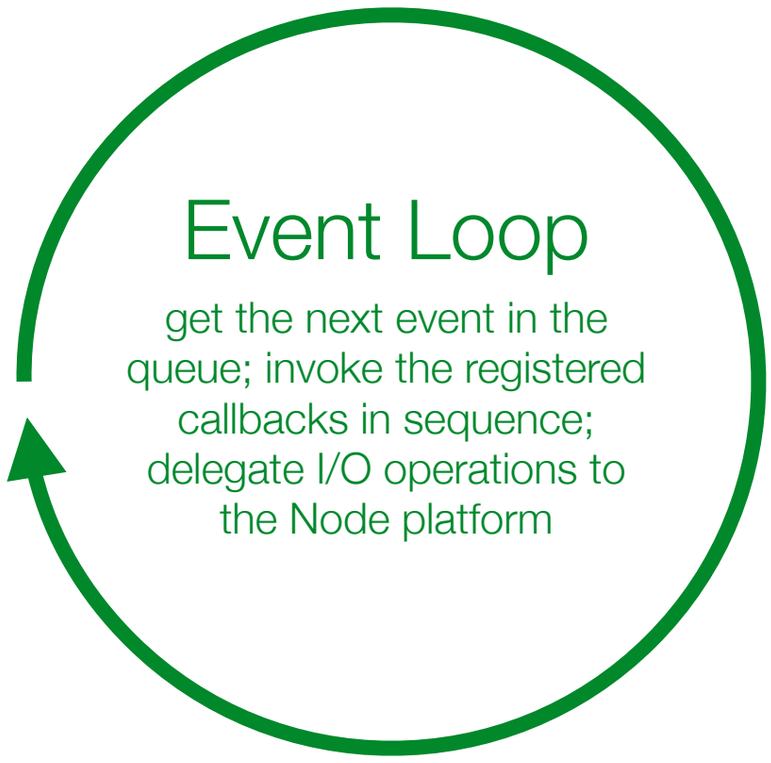
'request' **event**

'request' **event**

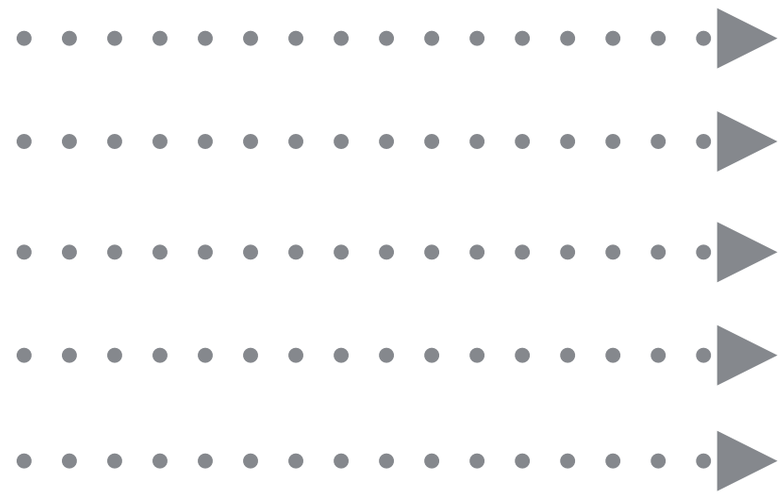
'data' **event**

'request' **event**

Queue of events that have been **emitted**



All the code that **you** write runs on a **single thread**



The **long-running tasks** (I/Os) are executed by Node in parallel; Node emits events to report progress (which triggers your callbacks).

Another pattern is to provide a callback to node when invoking an asynchronous function.

What is **npm**?

---

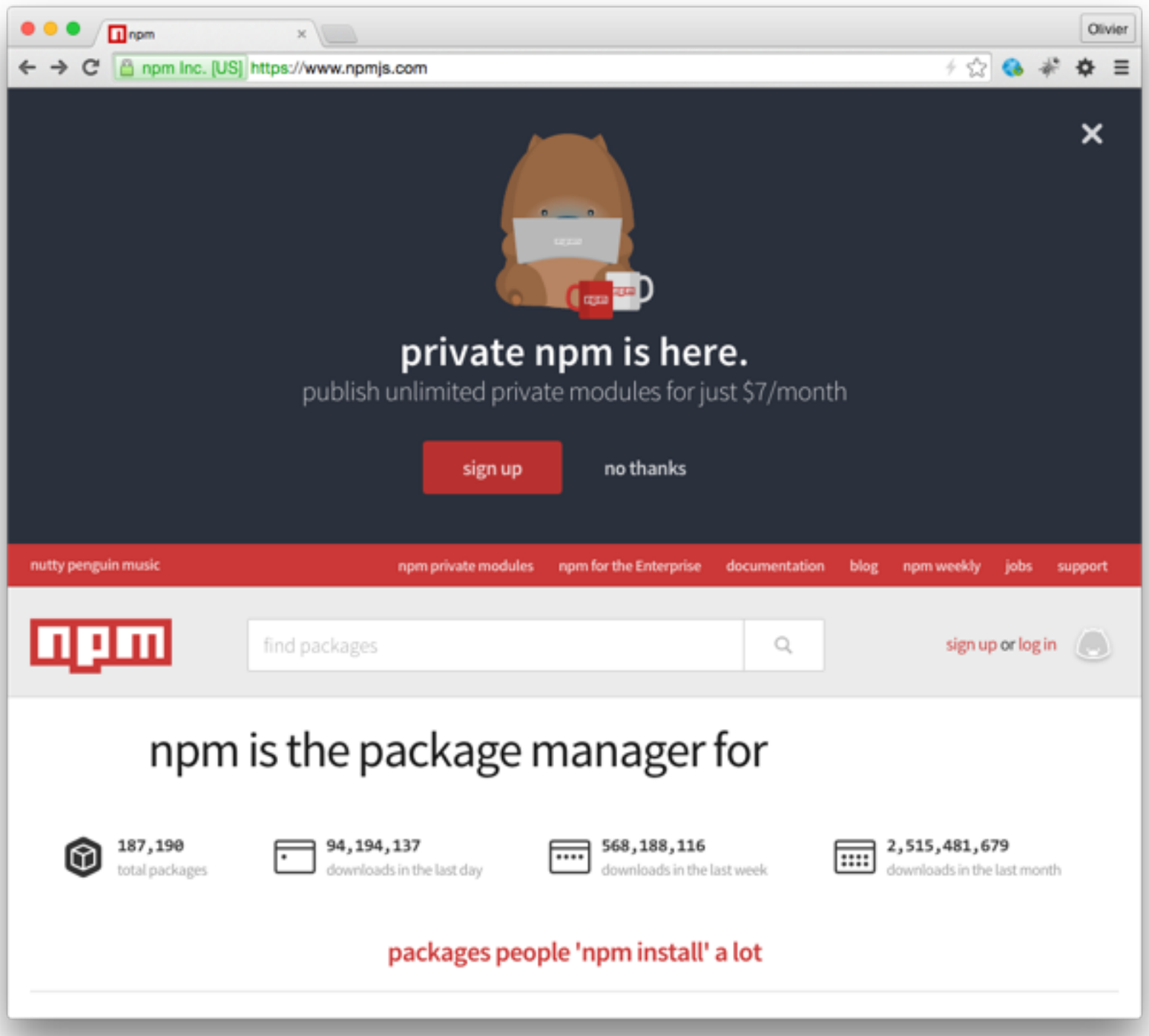
*”**npm** is the **package manager** for the Node JavaScript platform. It puts **modules** in place so that node can find them, and manages **dependency** conflicts intelligently.*

*It is extremely configurable to support a wide variety of use cases. Most commonly, it is used to **publish**, **discover**, **install**, and **develop** node programs.”*

https://www.npmjs.org/doc/cli/npm.html

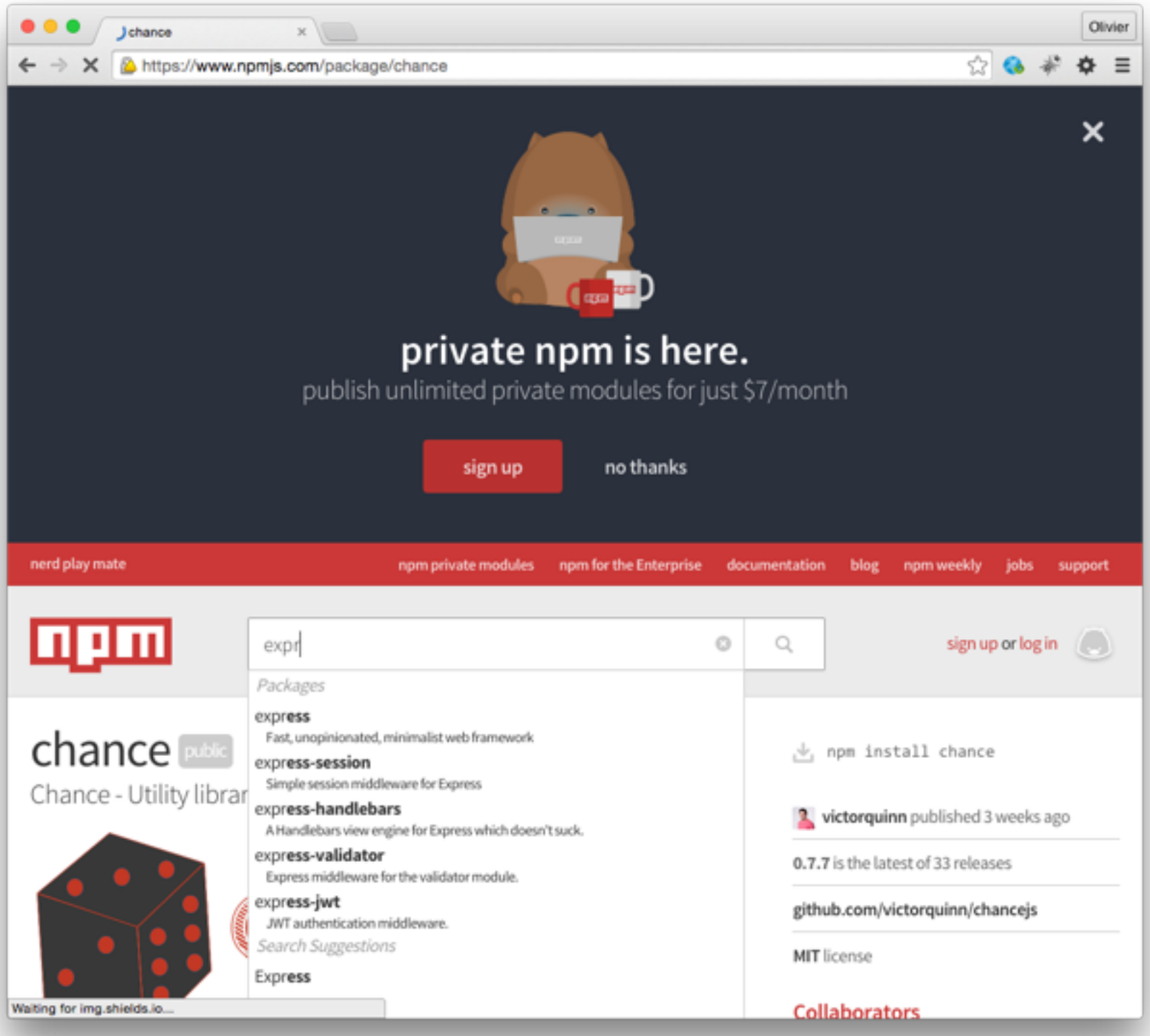
You **have to** read this:
https://www.npmjs.org/doc/misc/npm-faq.html

npm is a set of command line tools that work together with the **node registry**



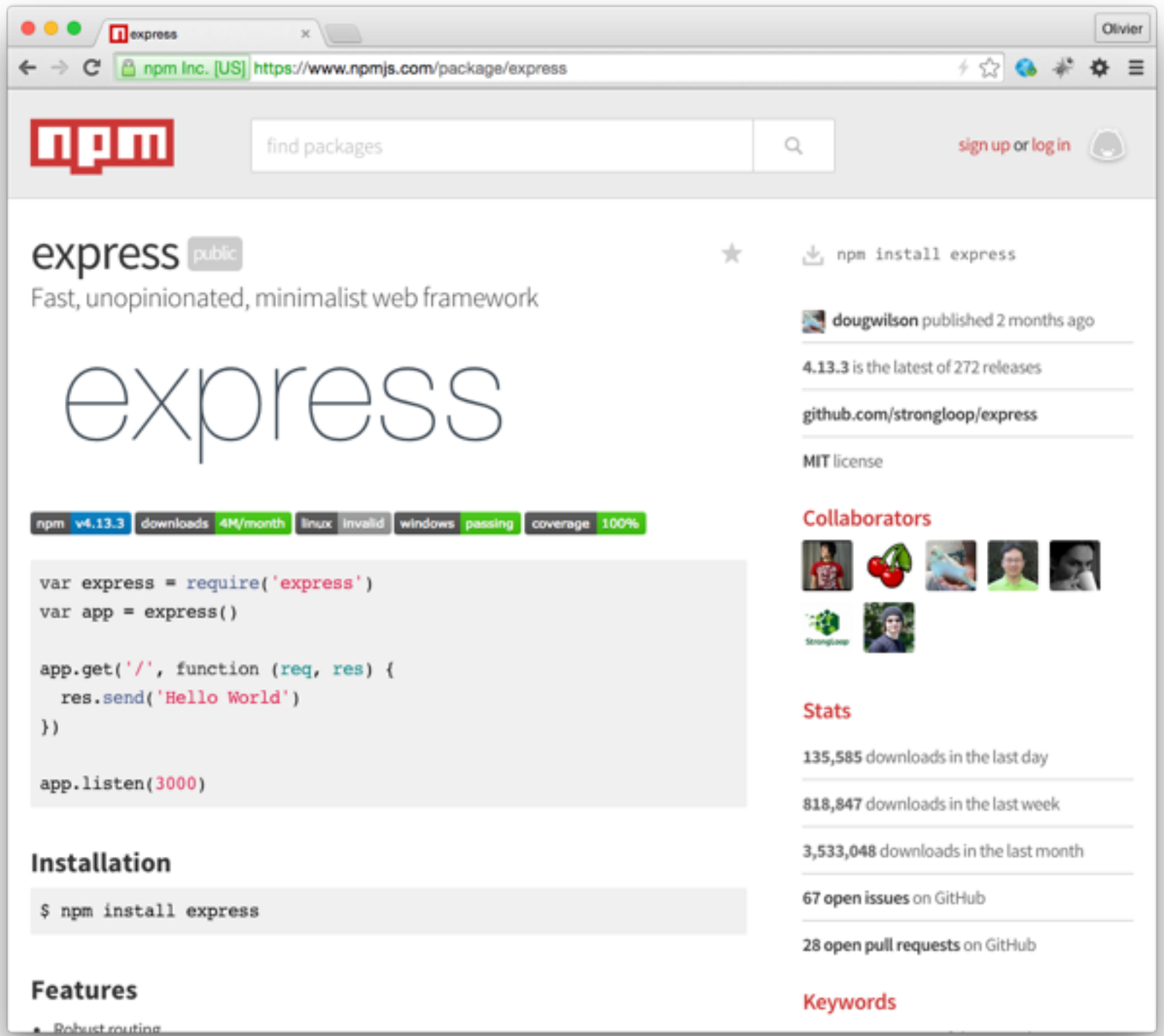
https://www.npmjs.org

npm is a set of command line tools that work together with the **node registry**



https://www.npmjs.org

npm is a set of command line tools that work together with the **node registry**



https://www.npmjs.org

npm is a set of command line tools that work together with the **node registry**

---

admin — bash — 120×30

```
Last login: Wed Sep 23 19:59:09 on console
$ npm help

Usage: npm <command>

where <command> is one of:
    access, add-user, adduser, apihelp, author, bin, bugs, c,
    cache, completion, config, ddp, dedupe, deprecate, dist-tag,
    dist-tags, docs, edit, explore, fa[illegible], find, find-dupes, get,
    help, help-search, home, i, info, init, install, issues, la,
    link, list, ll, ln, login, ls, outdated, owner, pack,
    prefix, prune, publish, r, rb, rebuild, remove, repo,
    restart, rm, root, run-script, s, se, search, set, show,
    shrinkwrap, star, stars, start, stop, t, tag, test, tst, un,
    uninstall, unlink, unpublish, unstar, up, update, v,
    verison, version, view, whoami

npm <cmd> -h     quick help on <cmd>
npm -l           display full usage info
npm faq          commonly asked questions
npm help <term>  search for help on <term>
npm help npm     involved overview

Specify configs in the ini-formatted file:
    /Users/admin/.npmrc
or on the command line via: npm <command> --key value
Config info can be viewed via: npm help config

npm@2.5.1 /usr/local/lib/node_modules/npm
$
```

init
install
publish
update

npm is a set of command line tools that work together with the **node registry**

- With npm, you can **install packages**:
  - **globally** (this is the case for tools and CLI utilities used across projects).
    - You need to use the **-g** flag if you want to do that.
  - **locally to a project** (this is the case for libs that your code depends on)
    - In this case, you first do a **npm init** and then use the **--save** flag. The dependencies are stored in a file named **package.json**
    - The modules are stored in a (often large) directory named **node_modules**, which you typically add to your **.gitignore** file.

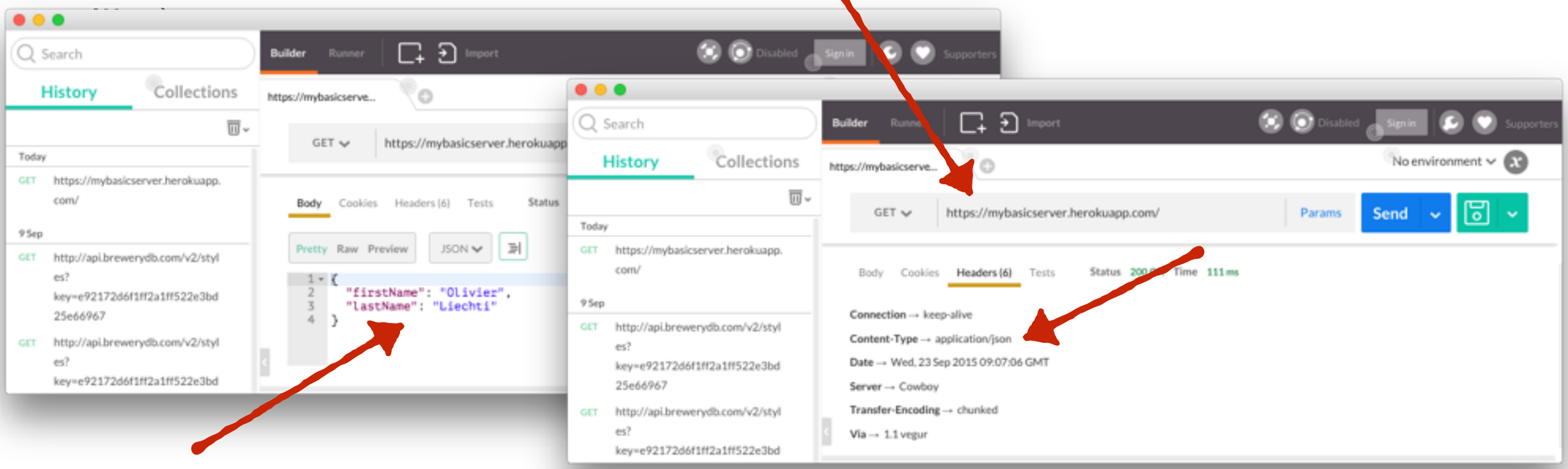
heig-vd

Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Exercise: your first node.js server on heroku

Requirements

- You must implement a **simple HTTP server** in node.js, test it locally and deploy it on heroku.
- When HTTP clients send requests to the server, you will always return a JSON payload (you will not look at the URL...).
- The **JSON payload** should be created from a **JavaScript object** that you create in your code (a student, a beer, ...or whatever you



Hints

- To **serialize** a JavaScript object as a **JSON string**, use code like this:

```
JSON.stringify(student)
```

- When heroku starts your application, it will **randomly pick an available TCP port**. It will assign the value of this port to an environment variable.
- With the following code, you can **retrieve the value** (and if it is not defined because you run locally, you can fallback on a default port):

```
var port = process.env.PORT || 3000;
daemon.listen(port);
```

- To recognize your app as a node.js server, heroku needs to find
  - a file named `package.json` (it can contain `{}` for now)
  - a file named `Procfile` (content: `web: node server.js`)

heig-vd

Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Environment Setup

How do I setup my **development environment** for Node.js applications?

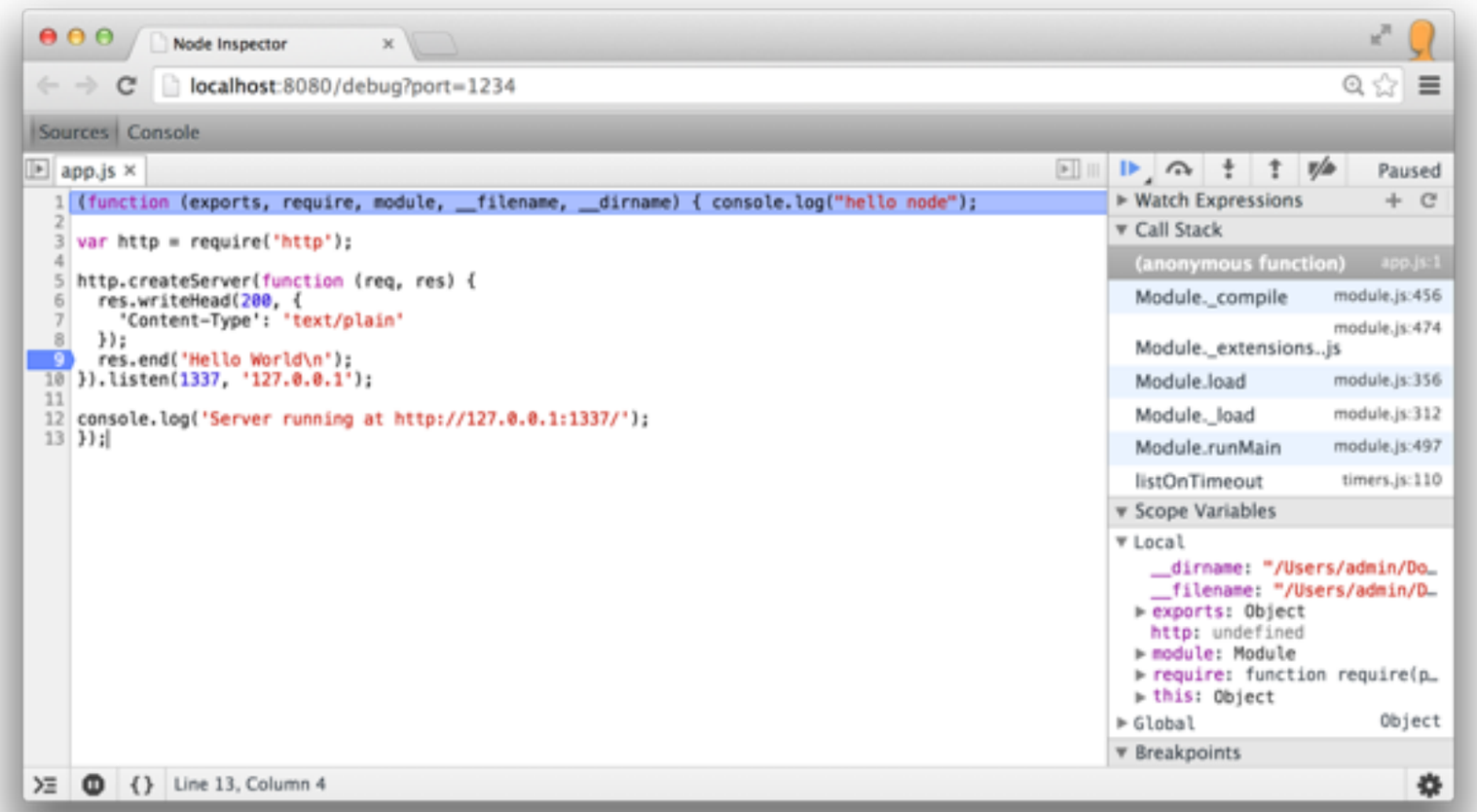
- **You should use the same IDE / editor as the one you use for your client-side code**
  - IntelliJ, Brackets and Atom work well for server-side applications
  - It's up to you to select your favorite tool, but take the time to learn how to use it efficiently (and to customize it)
- You **need** a debugger
  - Node ships with a **debugger** (http://nodejs.org/api/debugger.html). Use it with "node debug server.js"
  - A third-party module, **node-inspector**, is available and provides an integration with chrome dev tools (https://github.com/node-inspector/node-inspector).

How do I setup my **development environment** for Node.js applications?

```
$ npm install -g node-inspector

$ node-debug --debug-port 5858 app.js
```

or *(if you have an existing app running in debug mode)*

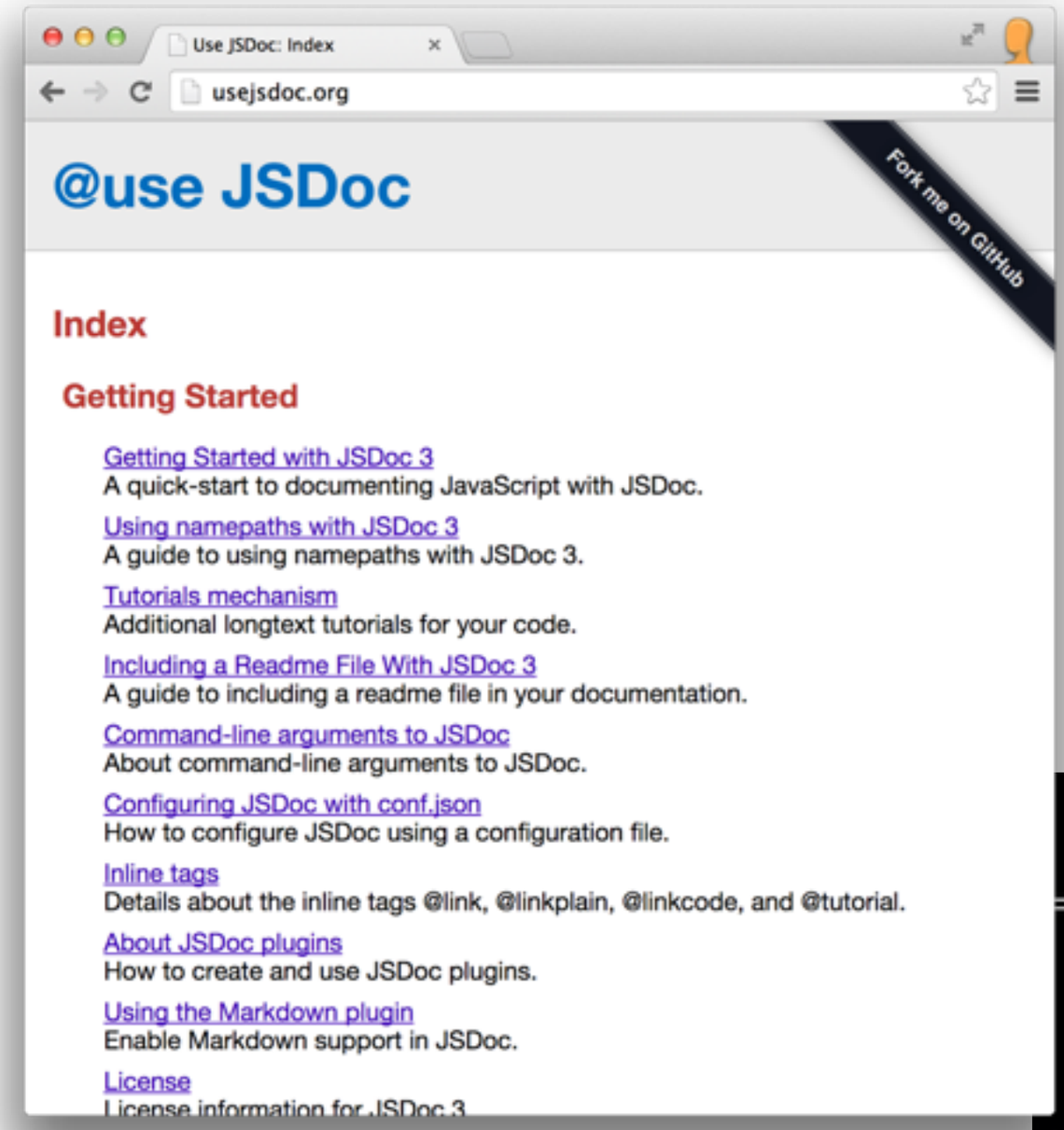
```
$ node-inspector
```



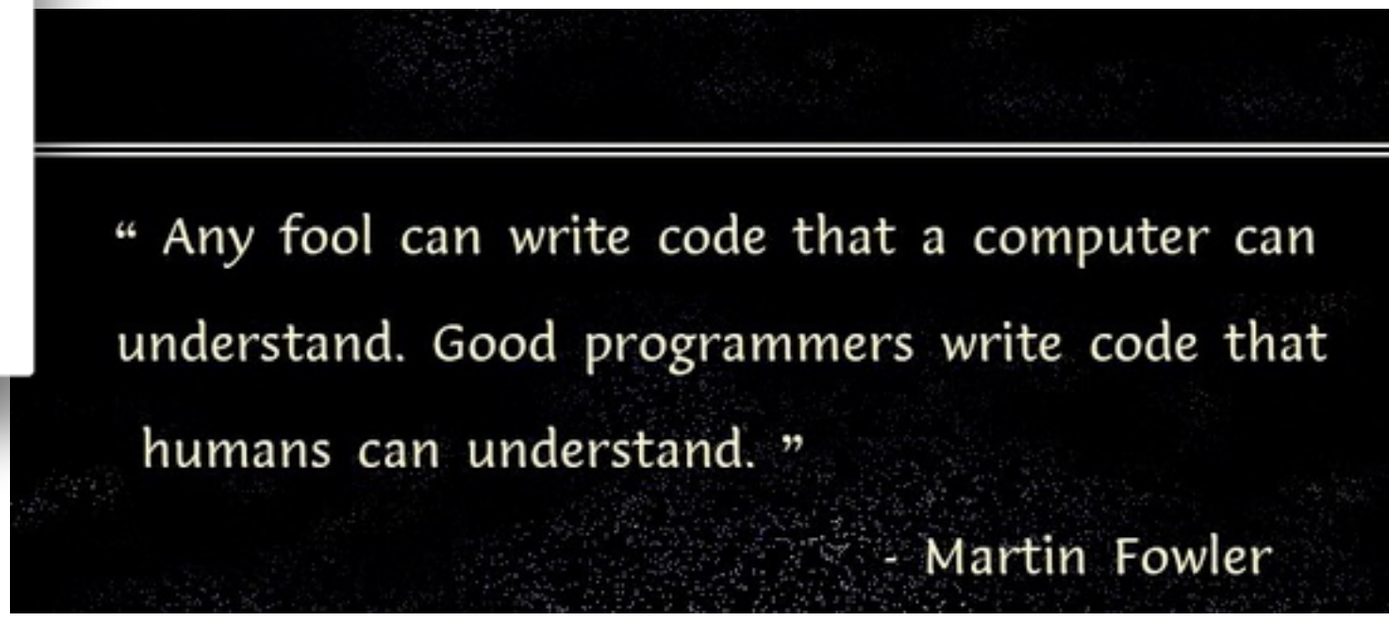
By default, Node.js uses port 5858 for the debugger. If you start several node processes on your host, you will have conflicts. In this case, make sure to use the –`debug-port` option

Document your code

http://usejsdoc.org/
https://github.com/jsdoc3/jsdoc




```
npm install -g jsdoc
jsdoc --destination doc --recurse src
open ./doc/index.html
```



heig-vd

Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Express

Fast, unopinionated, minimalist
web framework for Node.js

Express.js 101

Express.js: core functionality

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- **Implementation of the Model View Controller (MVC) pattern**
- **Routing**
  - **Mapping** between HTTP request attributes (URL, method, etc.) and **controllers** (handlers)
- **Middleware**
  - **Interception** and possibly **transformation** of HTTP requests and responses during their processing.
- **Template engines**
  - Rendering of views with **pluggable template engines**.

Express.js: server-side routing

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- **Routing** consists in finding some piece of code (a function) to execute when an HTTP request has been issued.
- We will see later (in a few weeks) that routing can happen on the client side. Today, we are looking at **routing on the server side**.
- Routing is part of the typical **Model-View-Controller (MVC) pattern** implemented by web frameworks (not only in JavaScript, but also in other languages).
- **Routing consists in finding the right controller when a request comes in**.
- The controller will then get a model and delegate the rendering of a view to a template engine.
- Comparison with **Java EE**: routing is declared in **servlet mappings**, either in the **`web.xml`** file or in **`@WebServlet`** annotations.

From the API reference

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Routing

Routing refers to the definition of end points (URIs) to an application and how it responds to client requests.

A route is a combination of a URI, a HTTP request method (GET, POST, and so on), and one or more handlers for the endpoint. It takes the following structure `app.METHOD(path, [callback...], callback)`, where `app` is an instance of `express`, `METHOD` is an HTTP request method, `path` is a path on the server, and `callback` is the function executed when the route is matched.

The following is an example of a very basic route.

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

// respond with "hello world" when a GET request is made to the homepage
app.get('/', function(req, res) {
  res.send('hello world');
});
```

- Route **methods** (GET, POST, PUT)
- Route **paths** ('/', '/home', '/students')
- **Request** and **response objects**

From the API reference

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Router

A `router` object is an isolated instance of middleware and routes. You can think of it as a "mini-application," capable only of performing middleware and routing functions. Every Express application has a built-in app router.

A router behaves like middleware itself, so you can use it as an argument to app.use() or as the argument to another router's use() method.

The top-level `express` object has a `Router()` function that creates a new `router` object.

Router([options])

Create a new router as follows:

```
var router = express.Router([options]);
```

- For large applications, it is better to split the controllers in multiple, isolated components. You should use multiple routers for that purpose.
- `express.Router()` creates a new router. (no need for `r = new Router()`)

Express.js: middleware (filters)

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- **Incoming HTTP requests can be processed by multiple components, organized in a pipeline**
- The components can inspect and even modify the incoming HTTP requests and HTTP responses (think about security, compression, etc.).
- Express.js calls these components "**middleware**" functions
- Middleware can be chained. They can intercept requests at different levels (all requests, requests under a certain path, requests handled by a specific router, etc.)
- Comparison with **Java EE**: an Express.js middleware is very similar to a **Servlet Filter**.
- **Built-in middleware** components is available (in separate npm modules).
- We will use the **express.static** middleware to serve static content (e.g. HTML)

From the API reference

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Using middleware

Express is a routing and middleware web framework with minimal functionality of its own: An Express application is essentially a series of middleware calls.

Middleware is a function with access to the request object (`req`), the response object (`res`), and the next middleware in the application's request-response cycle, commonly denoted by a variable named `next`.

Middleware can:

- Execute any code.
- Make changes to the request and the response objects.
- End the request-response cycle.
- Call the next middleware in the stack.

If the current middleware does not end the request-response cycle, it must call `next()` to pass control to the next middleware, otherwise the request will be left hanging.

An Express application can use the following kinds of middleware:

- Application-level middleware
- Router-level middleware
- Error-handling middleware
- Built-in middleware
- Third-party middleware

You can load application-level and router-level middleware with an optional mount path. Also, you can load a series of middleware functions together, creating a sub-stack of the middleware system at a mount point.

From the API reference

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Application-level middleware

Bind application-level middleware to an instance of the app object with `app.use()` and `app.METHOD()`, where `METHOD` is is the HTTP method of the request that it handles, such as GET, PUT, POST, and so on, in lowercase. For example:

```
var app = express();

// a middleware with no mount path; gets executed for every request to the app
app.use(function (req, res, next) {
  console.log('Time:', Date.now());
  next();
});

// a middleware mounted on /user/:id; will be executed for any type of HTTP request to /user/:id
app.use('/user/:id', function (req, res, next) {
  console.log('Request Type:', req.method);
  next();
});

// a route and its handler function (middleware system) which handles GET requests to /user/:id
app.get('/user/:id', function (req, res, next) {
  res.send('USER');
});
```

Express.js: template engines

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- **In the MVC design pattern:**
  - the **controller** creates a **model** (or even better, it gets a model from a service component located in the business tier)
  - the **controller** places this model somewhere in memory (in a given **scope**, such as request or session scopes)
  - the **controller** delegates the rendering action to the appropriate **view.**
  - the **view** retrieves the model from the scope and generates the final data by injecting variables in some kind of **template**.
- **With Express.js:**
  - the **controller** is a JavaScript function
  - the **model** is a JavaScript object
  - the **view** is a template in one of the pluggable template engines (the default is jade, but there many others and you can even implement your own!)

From the API reference

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Once the view engine is set, you don't have to explicitly specify the engine or load the template engine module in your app, Express loads it internally as shown below, for the example above.

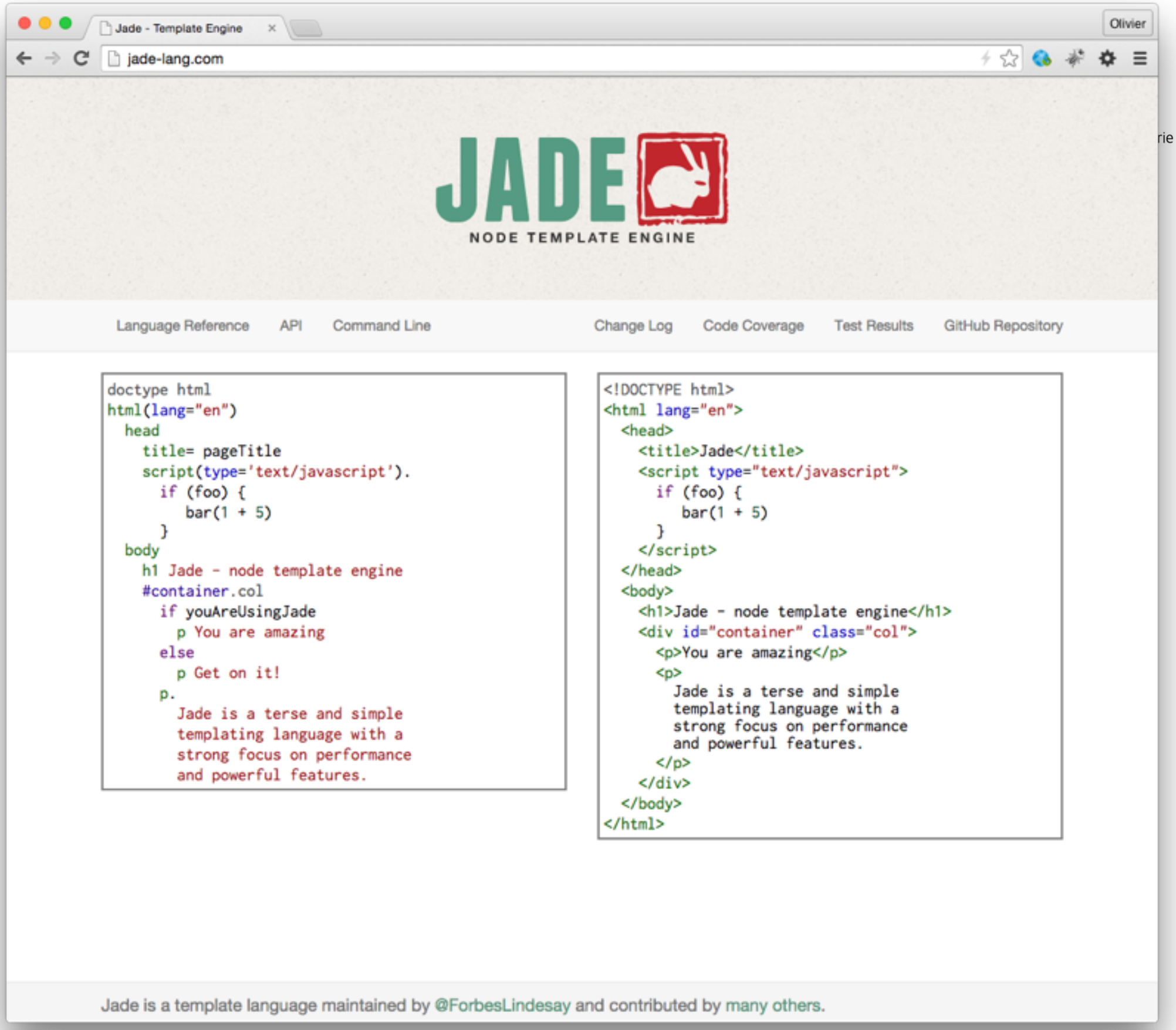
```
app.set('view engine', 'jade');
```

Create a Jade template file named "index.jade" in the views directory, with the following content.

```
html
  head
    title!= title
  body
    h1!= message
```

Then create a route to render the "index.jade" file. If the `view engine` property is not set, you will have to specify the extension of the view file, else you can omit it.

```
app.get('/', function (req, res) {
  res.render('index', { title: 'Hey', message: 'Hello there!'});
});
```


Jade - Template Engine
Olivier
jade-lang.com
JADE
NODE TEMPLATE ENGINE
Language Reference
API
Command Line
Change Log
Code Coverage
Test Results
GitHub Repository
doctype html
html(lang="en")
head
title= pageTitle
script(type='text/javascript').
if (foo) {
bar(1 + 5)
}
body
h1 Jade - node template engine
#container.col
if youAreUsingJade
p You are amazing
else
p Get on it!
p.
Jade is a terse and simple
templating language with a
strong focus on performance
and powerful features.
<!DOCTYPE html>
<html lang="en">
<head>
<title>Jade</title>
<script type="text/javascript">
if (foo) {
bar(1 + 5)
}
</script>
</head>
<body>
<h1>Jade - node template engine</h1>

You are amazing

Jade is a terse and simple
templating language with a
strong focus on performance
and powerful features.

</body>
</html>
Jade is a template language maintained by @ForbesLindesay and contributed by many others.

HTML2Jade - HTML to Jac

Olivier

html2jade.org

jade CONVERTER

Do not output enveloping html and body tags

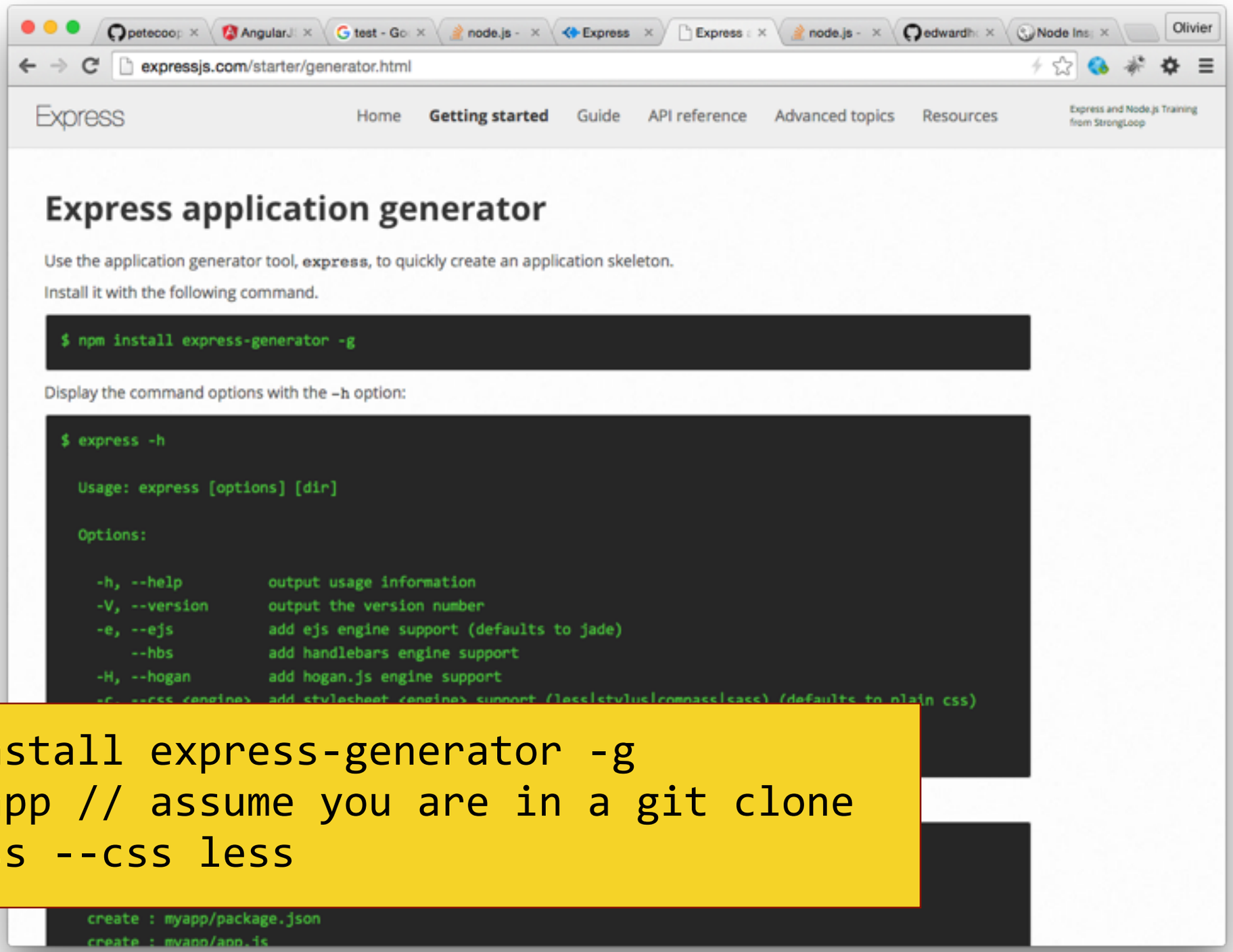
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Jade</title>
    <script type="text/javascript">
      foo = true;
      bar = function () {};
      if (foo) {
        bar(1 + 5)
      }
    </script>
  </head>
  <body>
    <h1>Jade - node template engine</h1>
    <div id="container" class="col">
      <p>You are amazing</p>
      <p>Jade is a terse and simple
         templating language with a
         strong focus on performance
         and powerful features.</p>
    </div>
  </body>
</html>
```

```
doctype html
html(lang='en')
  head
    title Jade
    script(type='text/javascript').
      foo = true;
      bar = function () {};
      if (foo) {
      bar(1 + 5)
      }
  body
    h1 Jade - node template engine
    #container.col
      p You are amazing
      p
        | Jade is a terse and simple
        | templating language with a
        | strong focus on performance
        | and powerful features.
```

HTML2jade help you convert a HTML snippet to a Jade snippet. Useful for testing out how something would look in Jade vs HTML

Experiment by *chenka* | Powered by *Jade*, *html2jade*, *Ace*

Example




```
$ npm install express-generator -g
$ cd myapp // assume you are in a git clone
$ express --css less
```

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

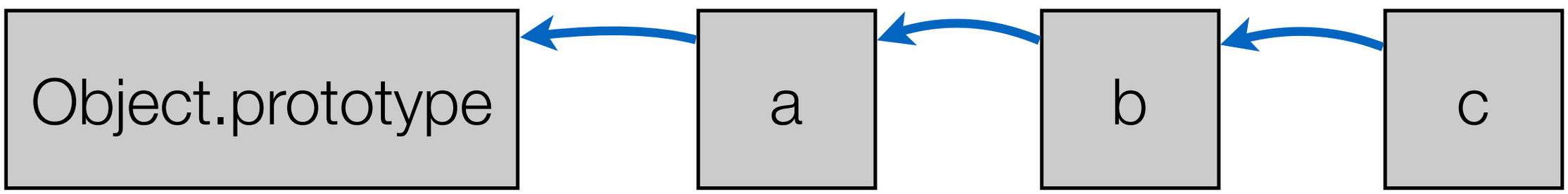
What is the difference between Object.getPrototypeOf(myF) and myF.prototype?

JavaScript 101
Reminder

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

#1 Every object **inherits** from a prototype object

```
var a = {};
var b = Object.create(a);
var c = Object.create(b);
```



Object is a function, **Object.prototype** is an object

#2 Object.prototype is the default ancestor for objects

```
typeof Object.prototype === object
```

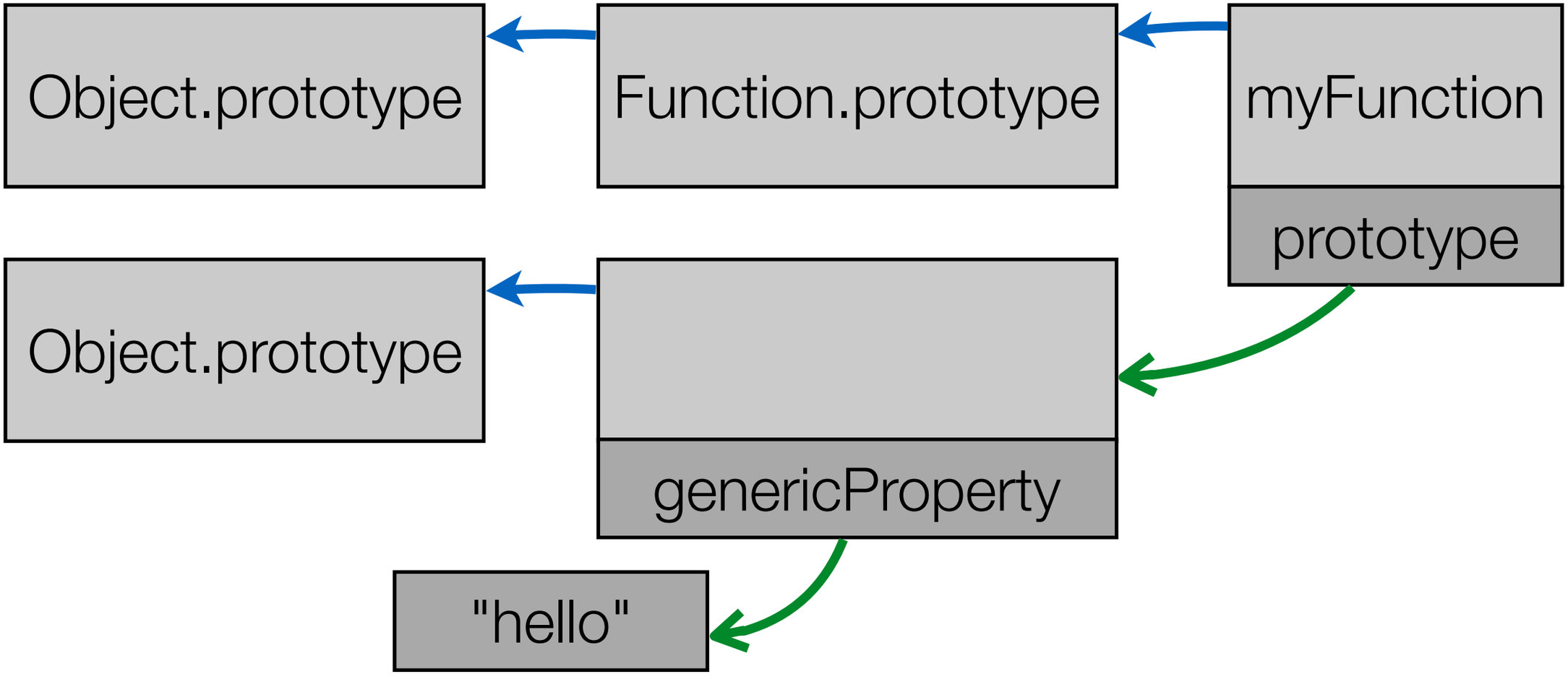
JavaScript 101
Reminder

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

#3 Every function has a **prototype property**

#4 Every function **inherits** from Function.prototype

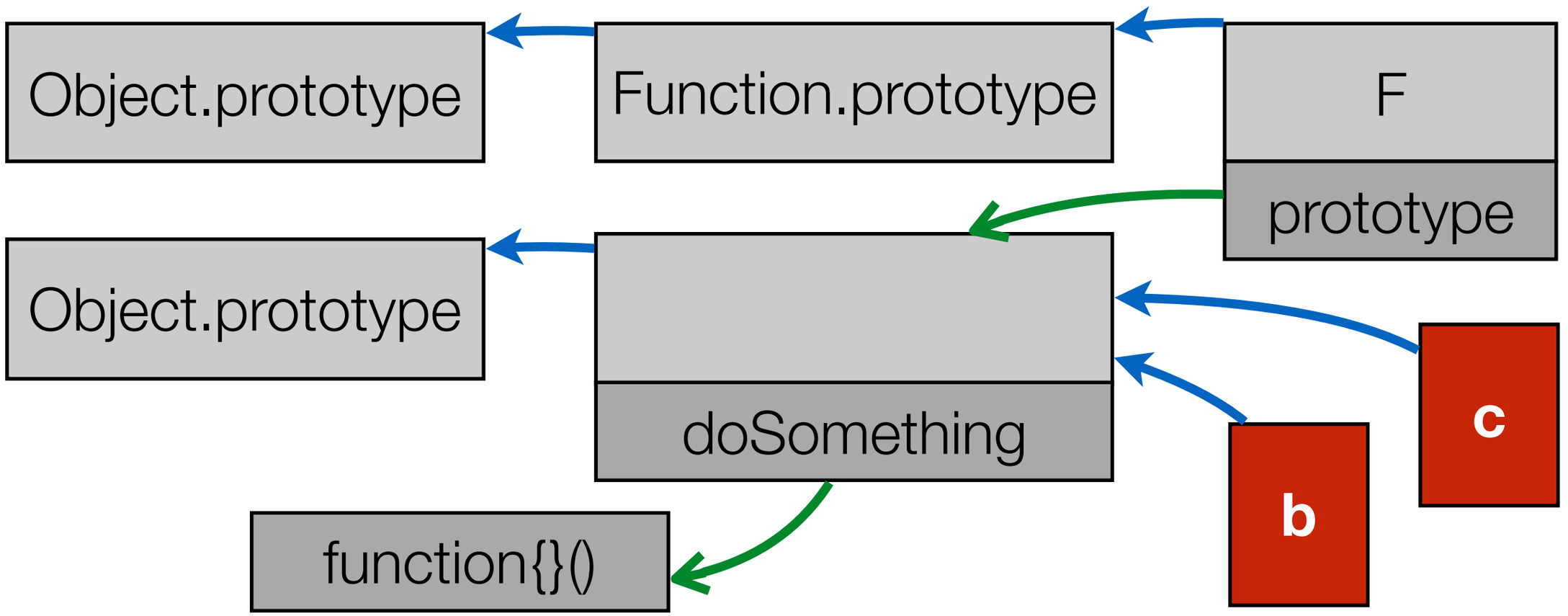
```
var myFunction = function() {};
myFunction.prototype.genericProperty = "hello";
```

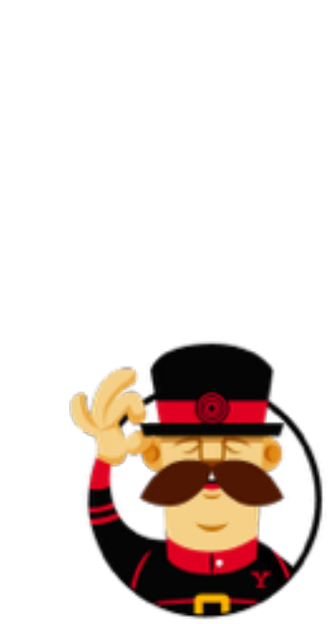


heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

#5 When you use the **new** keyword with a function F, you create an object. This object inherits from F.prototype.

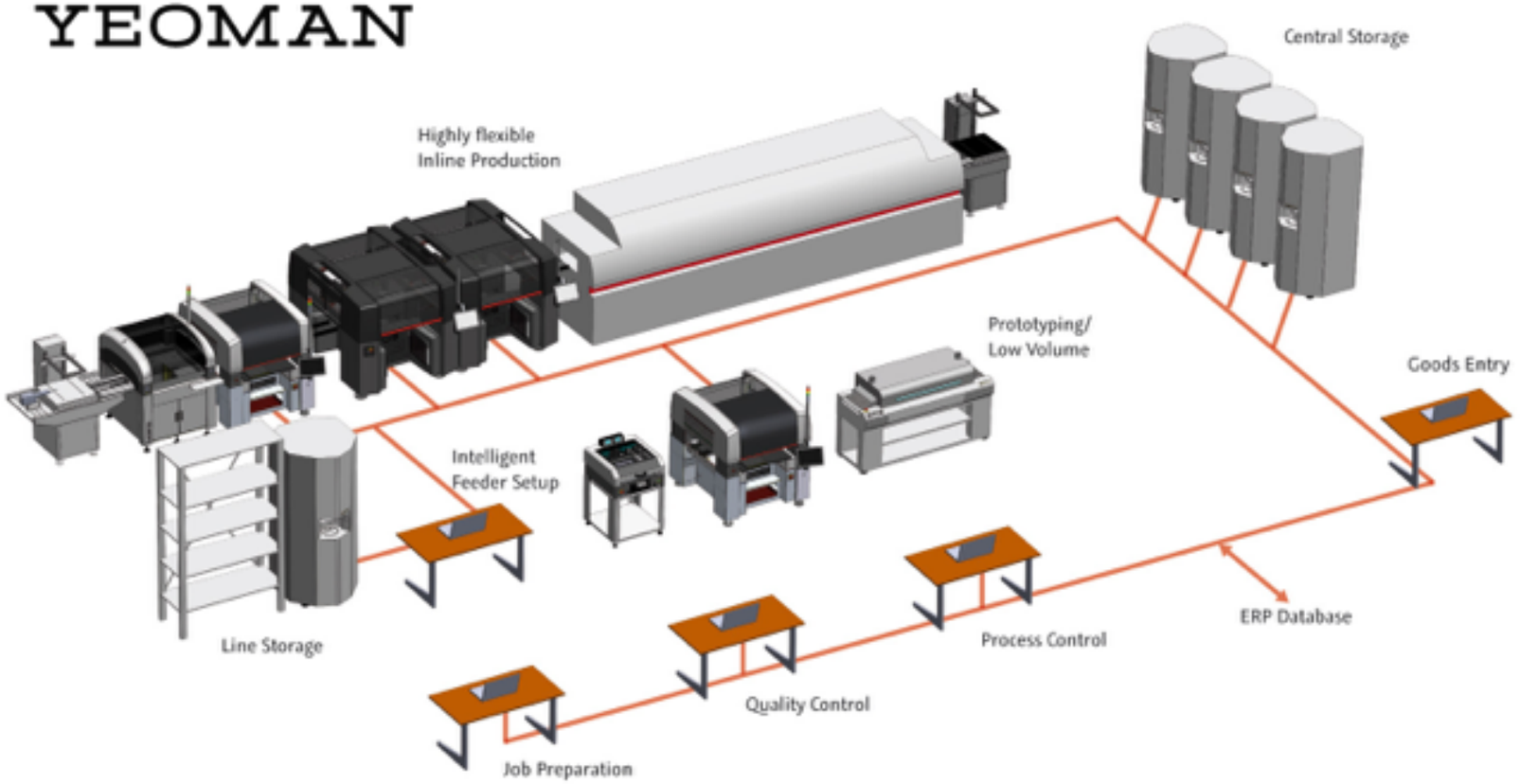
```
var F = function() {};
F.prototype.doSomething = function(){};
var b = new F();
var c = new F();
// b.doSomething() and c.doSomething() execute the same code
```





YEOMAN

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud



Scaffolding & Development Pipelines

How do I **bootstrap** and **structure** my project?

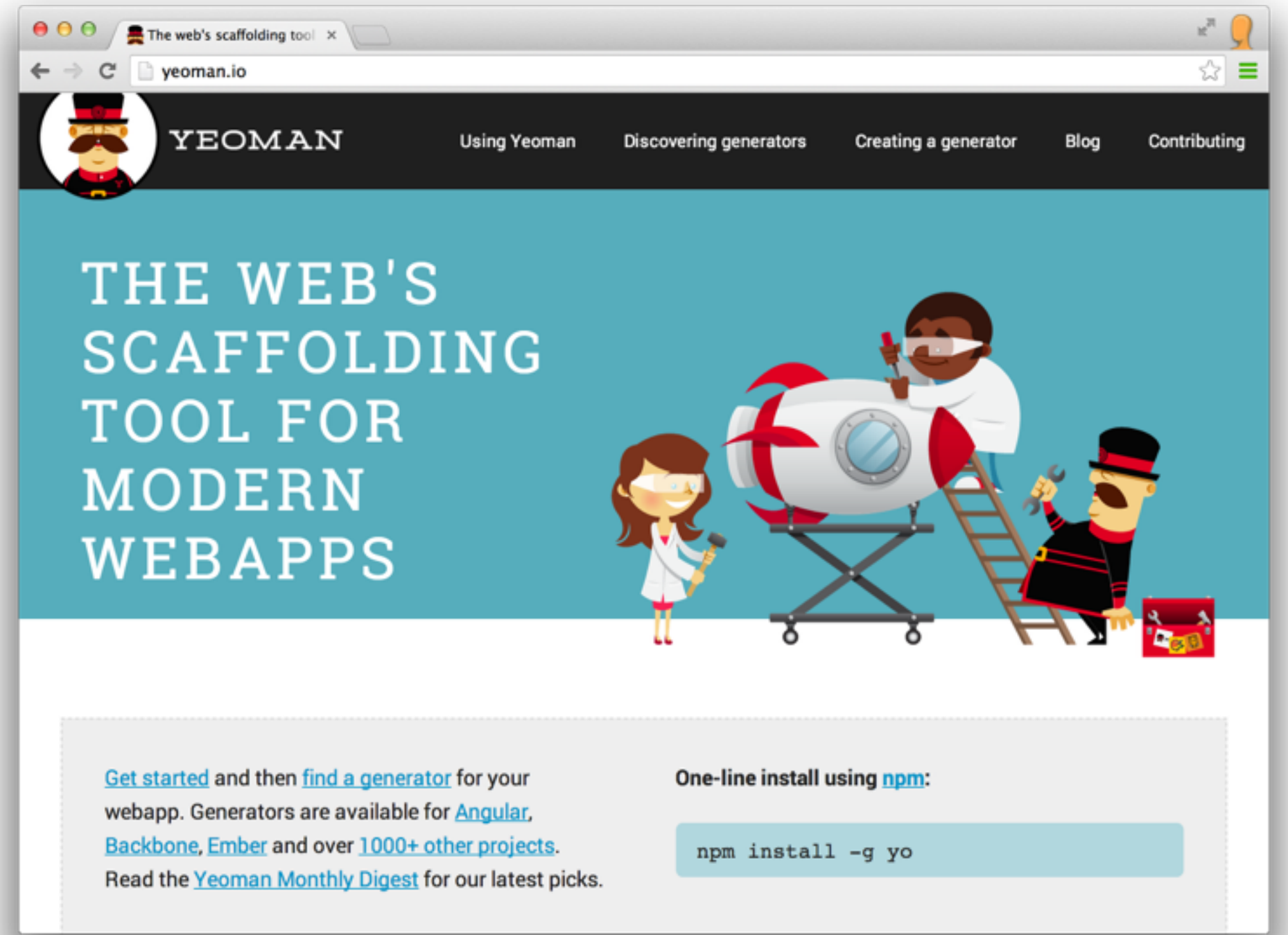
- Based on the specifications, we know that we will develop components **both on the client and on the server side**. We also want to **automate the build process** for our project.
- What should we do? **Start from scratch** or use some kind of **skeleton**? What are our **options**? What are the **professional** front-end developers doing?

https://www.youtube.com/watch?v=vDbbz-BdyYc

Paul Irish, "Delivering the goods" - Fluent 2014 Keynote
by O'Reilly • 7 months ago • 29,621 views
Fluent 2014, "Keynote With **Paul Irish**". About **Paul Irish** (Google): **Paul Irish** is a front-end developer who loves the web. He is on ...
HD

Fluent 2013: Paul Irish, "JavaScript Authoring Tooling"
by O'Reilly • 1 year ago • 35,810 views
http://fluentconf.com To view a complete archive of the Fluent 2013 tutorials and sessions, check out the All Access video ...
HD

Meet **Yeoman**.


The web's scaffolding tool
yeoman.io
YEOMAN
Using Yeoman
Discovering generators
Creating a generator
Blog
Contributing
THE WEB'S SCAFFOLDING TOOL FOR MODERN WEBAPPS
Get started and then find a generator for your webapp. Generators are available for Angular, Backbone, Ember and over 1000+ other projects. Read the Yeoman Monthly Digest for our latest picks.
One-line install using npm:
npm install -g yo

What is **Yeoman**?

- Yeoman is a **combination of tools**, which allows to you to setup a **complete**, **automated**, **efficient** and **reliable development workflow**.
- **Yo** is a tool for generating project skeletons (**scaffolding**). You can create and share your skeletons. **Yo generators are npm modules** and you can find one for most popular web frameworks.
- **Bower** is a tool for managing "**web dependencies**". Not only javascript modules, but also CSS files, images, etc.
- **Grunt** is a **task runner**. It is the tool that drives your automated process, by executing a series of tasks. There are lots of grunt plugins provided by the community for all aspects of your project.

Ok... generators look cool. But how should I **pick the “right” one**?

Generators | Yeoman

yeoman.io/generators/

YEOMAN

Using Yeoman | Discovering generators | Creating a generator | Blog | Contributing

GENERATORS

Your generator must have the "yeoman-generator" keyword and a repo description to be listed. Official generators are marked with

express

Name	Description	Author	Stars
angular-fullstack	AngularJS with an Express server	Tyler Henkel	1985
express	An express generator for Yeoman, based on the express command line tool	petecoop	180
mean-seed	MEAN Seed / MEAN Stack (AngularJS, node.js app) - MongoDB, Express, Angular, Node, Grunt, Bower, Yo. 'Core' and 'Module' subgenerators for	Luke Madera	102

How do you **pick a generator** for your project?

- You probably **have an idea of the framework(s) you want to use** on the server and or client side (express, angular, backbone, etc.). You will use this as a first filter.

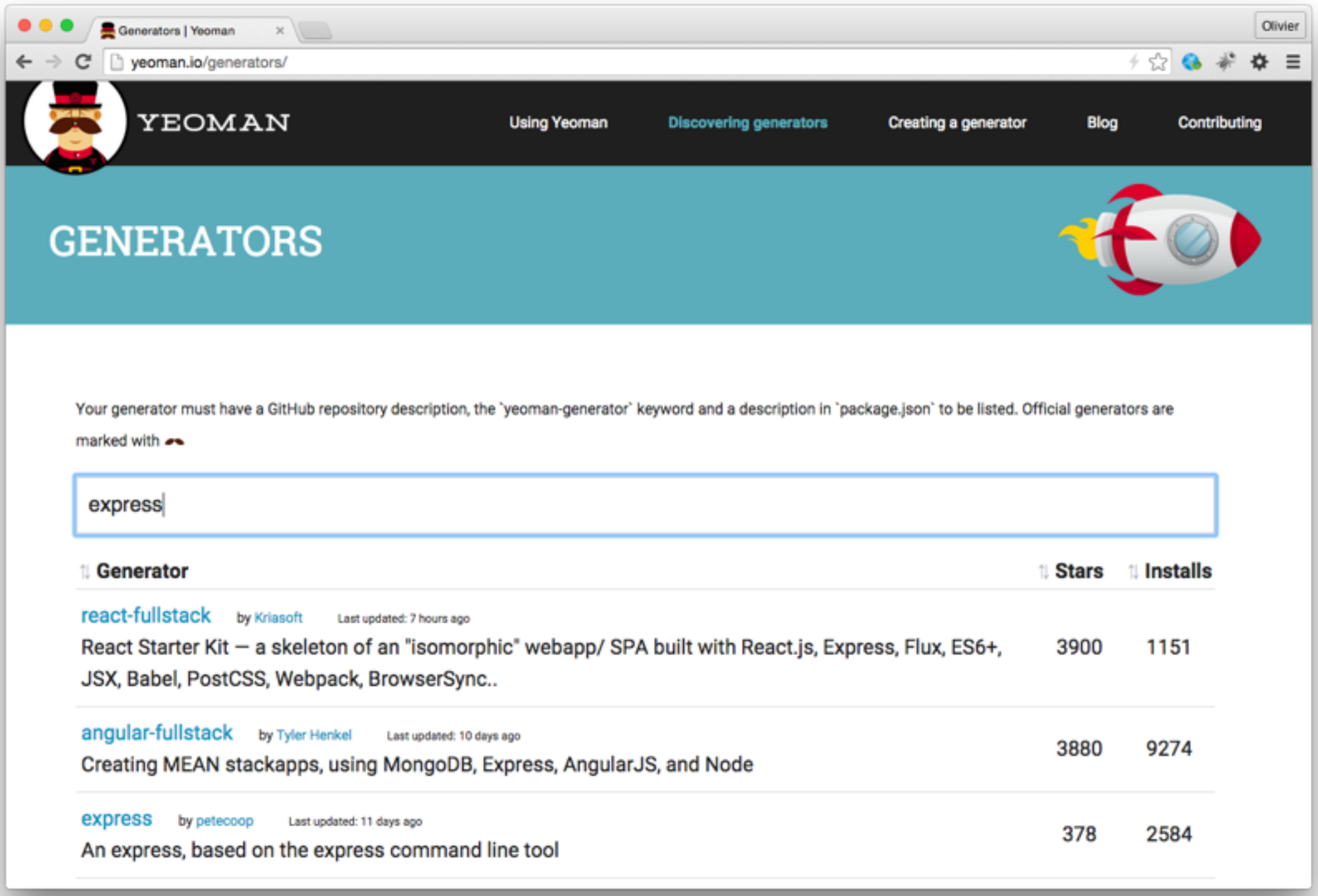
- Some of the generators are **supported by the Yeoman Team**. That is probably a good indication about the quality and support over time (evolution).

- Developers who use generators can **“star”** those they like. **Sorting by popularity** is also an interesting indication. If the community is big, you can expect issues to be reported and fixed, to see new features, etc.

- After you have identified **promising candidates**, you need to get a **first impression**. Generate and build a project with each candidate. Look at their Github repository. Do you like what you see? Do you like the documentation?

- Often, you will need to choose between “**lightweight**” and very “**rich**” generators. Lightweight generators are easier to learn and give you more control (but more work). Rich generators do a lot of things out-of-the-box but can be intimidating at first (learning curve to understand the skeleton).

Meet the **express** generator.



Meet the **express** generator.

---

petecoop/generator-expre ×

Olivier

GitHub, Inc. [US] https://github.com/petecoop/generator-express

Features

- Basic or MVC style file structure
- CoffeeScript Support
- Gulp or Grunt build tools with file watching and livereload
- .editorconfig for consistent coding styles within text editors
- Support View engines:
  - Jade
  - Handlebars
  - Swig
  - EJS
  - Marko
  - Nunjucks
- Supported CSS pre-processors
  - SASS (both node-sass and ruby sass)
  - LESS
  - Stylus
- Supported Databases (with MVC structure):
  - MongoDB
  - MySQL
  - PostgreSQL
  - RethinkDB
  - SQLite

Getting started

- Make sure you have yo installed: `npm install -g yo`
- Install the generator **globally**: `npm install -g generator-express`
- Run: `yo express` and select Basic. Add `--coffee` if you require CoffeeScript.
- Run: `grunt` or `gulp` to run the local server at `localhost:3000`, the grunt/gulp tasks include live reloading for views, css in public/css and restarting the server for changes to app.js or js in routes/

Why and **when** is this generator interesting?

- It is a good solution if you want to use **Express.js on the server side** and are not concerned by the client side (no need for a client-side framework such as Angular.js)
- It is **more powerful** than the standard generator provided by the Express.js framework.
- It allows you to easily integrate **persistence** , with a choice of several relational (e.g. MySQL) and NoSQL (e.g. MongoDB) data stores. In that sense, it generates a full multi-tiered MVC application.
- It comes with a **Grunt file**, which includes the **live reload plugin**.
- The generated skeleton and Grunt file are rather **lightweight and straightforward**. It is fairly easy to get into the generated code and to add your application-specific code.
- **Recommendation**: generate a first application skeleton, without persistence (not an MVC app). Play with the application (you don't need a database). When you have become familiar with the tools and code structure, regenerate an app with persistence (you will need to install MongoDB before that).

Example

express-generator-demo — node — 101×35

```
$ yo express
? Would you like to create a new directory for your project? No
? Select a version to install: Basic
? Select a view engine to use: Jade
? Select a css preprocessor to use (Sass Requires Ruby): less
? Select a build tool to use: Grunt
   create bower.json
   create package.json
   create .bowerrc
   create .editorconfig
   create .gitignore
   create app.js
   create bin/www
   create routes/index.js
   create routes/user.js
   create views/error.jade
   create views/index.jade
   create views/layout.jade
   create public/css/style.less
   create Gruntfile.js


I'm all done. Running npm install & bower install for you to install the required dependencies. If th
is fails, try running the command yourself.


grunt-develop@0.4.0 node_modules/grunt-develop

cookie-parser@1.4.0 node_modules/cookie-parser
├── cookie-signature@1.0.6
└── cookie@0.2.2

serve-favicon@2.3.0 node_modules/serve-favicon
├── fresh@0.3.0
├── ms@0.7.1
```

Example

express-generator-demo — vim — 101×35

vim | bash

```
      options: {
        livereload: reloadPort
      }
    }
  }
});

grunt.config.requires('watch.server.files');
files = grunt.config('watch.server.files');
files = grunt.file.expand(files);

grunt.registerTask('delayed-livereload', 'Live reload after the node server has restarted.', functi
on () {
  var done = this.async();
  setTimeout(function () {
    request.get('http://localhost:' + reloadPort + '/changed?files=' + files.join(','),  function (
err, res) {
      var reloaded = !err && res.statusCode === 200;
      if (reloaded) {
        grunt.log.ok('Delayed live reload succes
      } else {
        grunt.log.error('Unable to make a delaye
      }
      done(reloaded);
    });
  }, 500);
});

grunt.registerTask('default', [
  'less',
  'develop',
  'watch'
]);
};
```

To **debug** your app, you need to edit the Gruntfile and use the **`node-inspector`** command:

```
grunt.initConfig({
  pkg: grunt.file.readJSON('package.json'),
  develop: {
    server: {
      file: 'bin/www',
      nodeArgs: ['--debug']
    }
  },
```

Meet the **angular-fullstack** generator.

Name	Description	Author	Stars
angular	AngularJS	The Yeoman Team	2617
angular-fullstack	AngularJS with an Express server	Tyler Henkel	1985

AngularJS Full-Stack generator

build passing

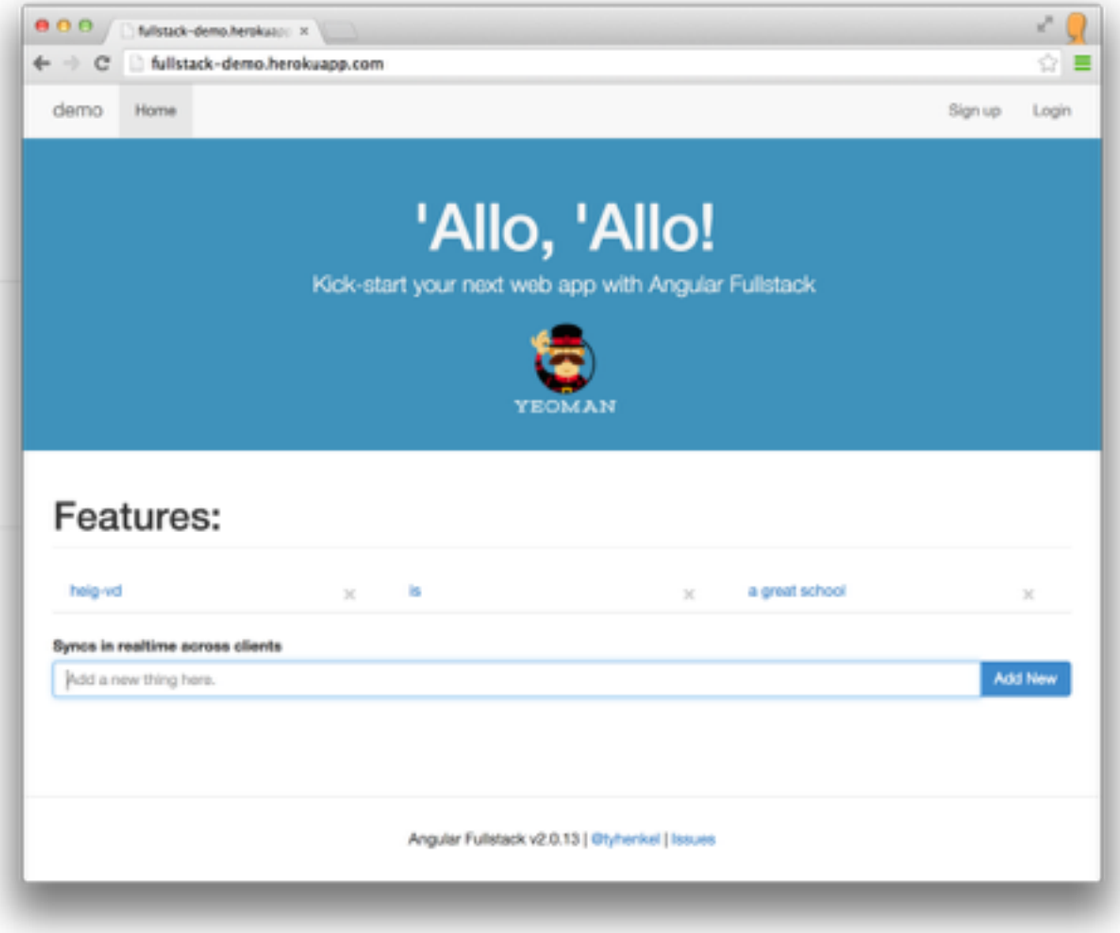
GITTER JOIN CHAT →

> Yeoman generator for creating MEAN stack applications, using MongoDB, Express, AngularJS, and Node - lets you quickly set up a project following best practices.

Example project

Generated with defaults: http://fullstack-demo.herokuapp.com/.

Source code: https://github.com/DaftMonk/fullstack-demo



Why and **when** is this generator interesting?

- It uses **Express.js** on the server side, **AngularJS** and **Socket.IO** on the client side and the glue between the frameworks.
- It has **sub-generators** to iteratively add new server-side and client-side components (new entities, new REST endpoints, new UI pages, etc.)
- The **structure of the AngularJS components** (folders and files were the UI elements are coded) follows best practices.
- It comes with support for **persistence** (with MongoDB), for **push notifications** (with Socket.IO) and for **authentication** (with Passport.js). The framework implements an interesting pattern for notifying CRUD operations to all connected users in realtime.
- Out-of-the-box, it provides a **complete and functional application** (which you can test on-line with their demo app).
- It supports **deployment on heroku** (and other cloud providers).
- The drawback is that the generated code and the Gruntfile is much more complex, compared to the express generator. There is a much steeper learning curve and if you are starting with JavaScript, npm and Grunt, you can get lost and intimated.
- **For this reason, I recommend to start with the express generator, and when you are up-to-speed to switch to the angular-fullstack generator.**

Build Pipeline

Sources → Pre-Processors → Raw assets → Code analysis → Clean assets → Optimizers → Optimized assets

Warnings ↻ Clean assets

```
.jade
.md
.scss
.stylus
.sass
.ejs
.coffee
...
.png
.svg
```

→ Pre-Processors →

```
.html
.css
.js
.png
.svg
```

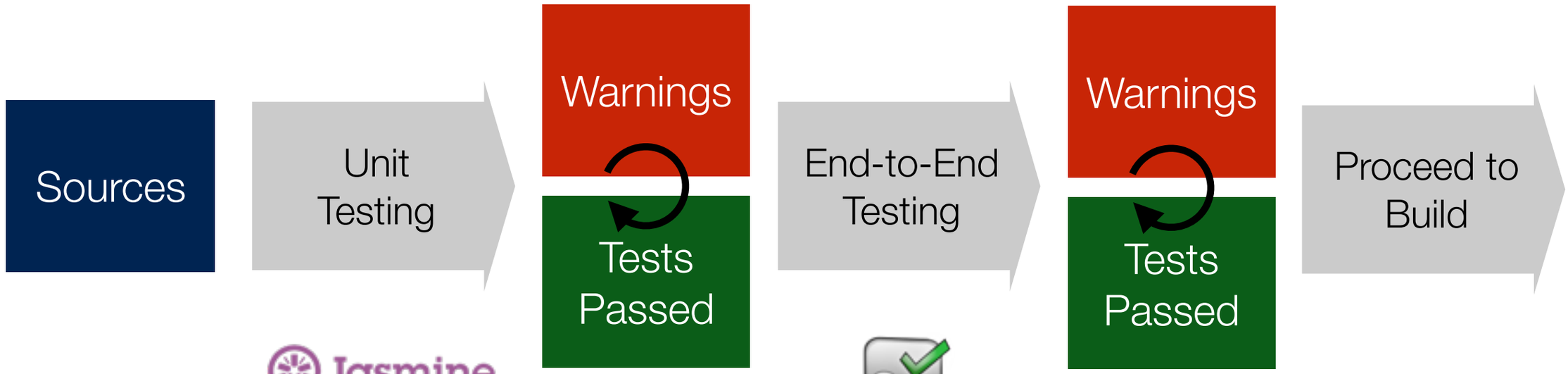
→ Code analysis: JS lint, CSS lint →

```
.html
.css
.js
.png
.svg
```

→ Optimizers: Minify, Concatenate, Concatenate, Opt. images →

```
.html
.css
.js
.png
.svg
```

Test Pipeline



Grunt

Getting Started | Plugins | Documentation | API

GRUNT

The JavaScript Task Runner

Why use a task runner?

In one word: automation. The less work you have to do when performing repetitive tasks like minification, compilation, unit testing, linting, etc, the easier your job becomes. After you've configured it through a Gruntfile, a task runner can do most of that mundane work for you—and your team—with basically zero effort.

Why use Grunt?

The Grunt ecosystem is huge and it's growing every day. With literally hundreds of plugins to choose from, you can use Grunt to automate just about anything with a minimum of effort. If someone hasn't already built what you need, authoring and publishing your own Grunt plugin to npm is a breeze. See how to get started.

Available Grunt plugins

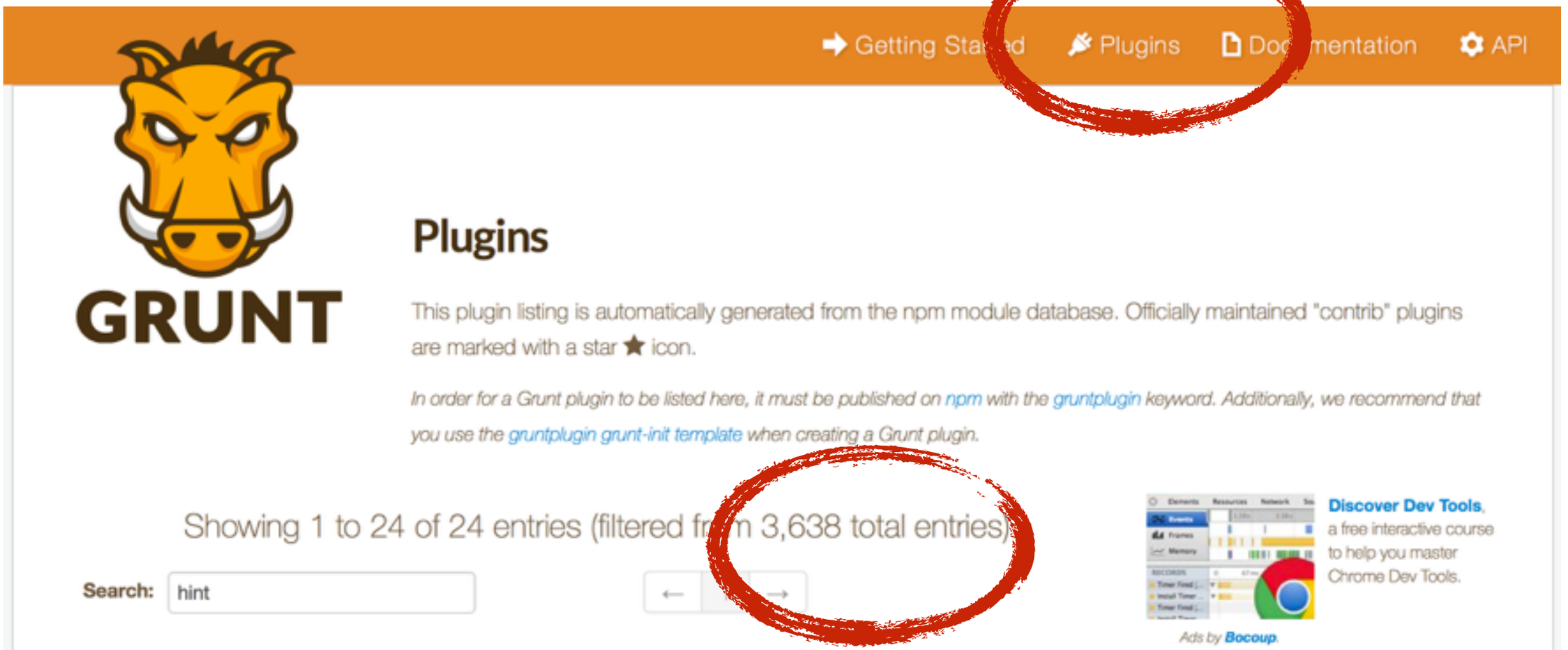
Many of the tasks you need are already available as Grunt Plugins, and new plugins are published every day. While the plugin listing is more complete, here's a few you may have heard of.

CoffeeScript | handlebars | jade (Node Template Engine) | JSHint | less | RequireJS | Sass | stylus

grunt

```
$ grunt
Running "jshint:gruntfile" (jshint) task
>> 1 file lint free.

Running "jshint:src" (jshint) task
>> 1 file lint free.

Running "jshint:test" (jshint) task
>> 1 file lint free.

Running "qunit:files" (qunit) task
Testing test/tiny-pubsub.html....OK
>> 4 assertions passed (23ms)

Running "clean:files" (clean) task
Cleaning "dist"...OK

Running "concat:dist" (concat) task
File "dist/ba-tiny-pubsub.js" created.

Running "uglify:dist" (uglify) task
File "dist/ba-tiny-pubsub.min.js" created.
Uncompressed size: 389 bytes.
Compressed size: 119 bytes gzipped (185 bytes minified).

Done, without errors.

$ _
```

Grunt


Getting Started
Plugins
Documentation
API
GRUNT
Plugins
This plugin listing is automatically generated from the npm module database. Officially maintained "contrib" plugins are marked with a star ★ icon.
In order for a Grunt plugin to be listed here, it must be published on npm with the gruntplugin keyword. Additionally, we recommend that you use the gruntplugin grunt-init template when creating a Grunt plugin.
Showing 1 to 24 of 24 entries (filtered from 3,638 total entries)
Search: hint
Discover Dev Tools, a free interactive course to help you master Chrome Dev Tools.
Ads by Bocoup.
Plugin
Updated
Grunt Version
Downloads
last 30 days
contrib-jshint by Grunt Team
Validate files with JSHint.
6 months ago
~0.4.0
538387
htmlhint by Yanis Wang
Validate html files with htmlhint.
3 months ago
~0.4.1
3045
contrib-jshint-jsx by Grunt Team

Telling grunt what to do: **Gruntfile.js**

- Let's define **two workflows**: a "test" workflow and a "default" workflow. I will be able to type "grunt test" and a "grunt" on the command line to run them.

```
// this would be run by typing "grunt test" on the command line
grunt.registerTask('test', ['jshint', 'qunit']);

// the default task can be run just by typing "grunt" on the command line
grunt.registerTask('default', ['jshint', 'qunit', 'concat', 'uglify']);
```

- The workflows use a few standard grunt plugins. Let's load them in the Gruntfile.js.

```
grunt.loadNpmTasks('grunt-contrib-uglify');
grunt.loadNpmTasks('grunt-contrib-jshint');
grunt.loadNpmTasks('grunt-contrib-qunit');
grunt.loadNpmTasks('grunt-contrib-watch');
grunt.loadNpmTasks('grunt-contrib-concat');
```

- Each grunt plugin can be configured. Here, we specify what files to lint and how.

```
jshint: {
  // define the files to lint
  files: ['gruntfile.js', 'src/**/*.js', 'test/**/*.js'],
  // configure JSHint (documented at http://www.jshint.com/docs/)
  options: {
      // more options here if you want to override JSHint defaults
    globals: {
      jQuery: true,
      console: true,
      module: true
    }
  }
}
```

A simple, complete Gruntfile.js

```
module.exports = function(grunt) {

  grunt.initConfig({
    pkg: grunt.file.readJSON('package.json'),
    concat: {
      options: {
        separator: ';'
      },
      dist: {
        src: ['src/**/*.js'],
        dest: 'dist/<%= pkg.name %>.js'
      }
    },
    uglify: {
      options: {
        banner: '/*! <%= pkg.name %> <%= grunt.template.today("dd-mm-yyyy") %> */\n'
      },
      dist: {
        files: {
          'dist/<%= pkg.name %>.min.js': ['<%= concat.dist.dest %>']
        }
      }
    },
    qunit: {
      files: ['test/**/*.html']
    },
    jshint: {
      files: ['Gruntfile.js', 'src/**/*.js', 'test/**/*.js'],
      options: {
        // options here to override JSHint defaults
        globals: {
          jQuery: true,
          console: true,
          module: true,
          document: true
        }
      }
    },
    watch: {
      files: ['<%= jshint.files %>'],
      tasks: ['jshint', 'qunit']
    }
  });

  grunt.loadNpmTasks('grunt-contrib-uglify');
  grunt.loadNpmTasks('grunt-contrib-jshint');
  grunt.loadNpmTasks('grunt-contrib-qunit');
  grunt.loadNpmTasks('grunt-contrib-watch');
  grunt.loadNpmTasks('grunt-contrib-concat');

  grunt.registerTask('test', ['jshint', 'qunit']);

  grunt.registerTask('default', ['jshint', 'qunit', 'concat', 'uglify']);

};
```

http://gruntjs.com/sample-gruntfile

Configure the grunt plugins.

Watch is a special plugin. It allows to execute tasks when files are changed in the file system.

Load the grunt plugins.

Define the two workflows.

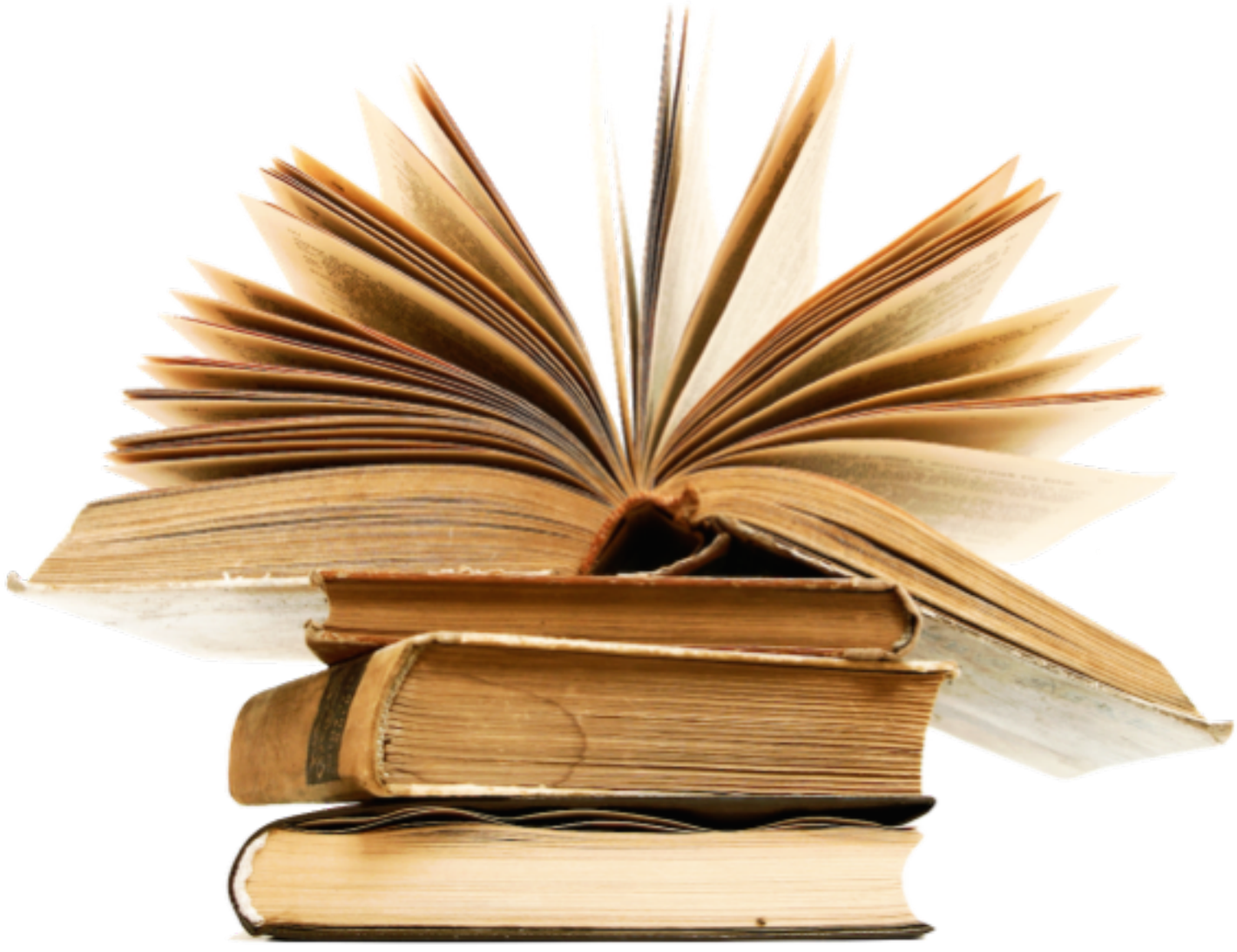
Executing a more complex workflow...

```
MacBook-Pro-de-admin:demo admin$ grunt build
Running "clean:dist" (clean) task
Cleaning .tmp...OK
Cleaning dist/package.json...OK
Cleaning dist/public...OK
Cleaning dist/server...OK

Running "injector:stylus" (injector) task
Missing option `template`, using `dest` as template instead
Injecting styl files (5 files)
>> Nothing changed

Running "concurrent:dist" (concurrent) task

    Running "stylus:server" (stylus) task
    File .tmp/app/app.css created.

    Done, without errors.


    Execution Time (2014-10-13 04:48:31 UTC)
    loading tasks  121ms  ▇▇▇▇▇▇ 26%
    stylus:server  343ms  ▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇ 74%
    Total 464ms

    Running "jade:compile" (jade) task
    File .tmp/app/about/about.html created.
    File .tmp/app/account/login/login.html created.
    File .tmp/app/account/settings/settings.html created.
    File .tmp/app/account/signup/signup.html created.
    File .tmp/app/admin/admin.html created.
    File .tmp/app/main/main.html created.
    File .tmp/components/modal/modal.html created.
    File .tmp/components/navbar/navbar.html created.

    Done, without errors.


    Execution Time (2014-10-13 04:48:31 UTC)
    loading tasks  121ms  ▇▇▇▇▇ 20%
    jade:compile   497ms  ▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇ 80%
    Total 619ms

    Running "svgmin:dist" (svgmin) task
    Total saved: 0 B

    Done, without errors.


    Execution Time (2014-10-13 04:48:31 UTC)
    loading tasks  476ms  ▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇ 67%
    svgmin:dist    234ms  ▇▇▇▇▇▇▇▇ 33%
    Total 710ms

    Running "imagemin:dist" (imagemin) task
    ✔ client/assets/images/yeoman.png (saved 3.73 kB - 30%)
    Minified 1 image (saved 3.73 kB)

    Done, without errors.


    Execution Time (2014-10-13 04:48:31 UTC)
    loading tasks     1s  ▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇ 83%
    imagemin:dist  207ms  ▇▇▇▇ 17%
    Total 1.2s

Running "injector:scripts" (injector) task
Missing option `template`, using `dest` as template instead
Injecting js files (16 files)
>> Nothing changed

Running "injector:stylus" (injector) task
Missing option `template`, using `dest` as template instead
Injecting styl files (5 files)
>> Nothing changed

Running "injector:css" (injector) task
Missing option `template`, using `dest` as template instead
>> Nothing changed

Running "wiredep:target" (wiredep) task

Running "useminPrepare:html" (useminPrepare) task
Going through client/index.html to update the config
Looking for build script HTML comment blocks

Configuration is now:

  concat:
  { generated:
   { files:
      [ { dest: '.tmp/concat/app/vendor.css', src: [] },
        { dest: '.tmp/concat/app/app.css',
          src: [ '{.tmp,client}/app/app.css' ] },
        { dest: '.tmp/concat/app/vendor.js',
          src:
           [ '{client,node_modules}/bower_components/jquery/dist/jquery.js',
             '{client,node_modules}/bower_components/angular/angular.js',
             '{client,node_modules}/bower_components/angular-resource/angular-resource.js',
             '{client,node_modules}/bower_components/angular-cookies/angular-cookies.js',
             '{client,node_modules}/bower_components/angular-sanitize/angular-sanitize.js',
             '{client,node_modules}/bower_components/angular-bootstrap/ui-bootstrap-tpls.js',
             '{client,node_modules}/bower_components/lodash/dist/lodash.compat.js',
             '{client,node_modules}/bower_components/angular-socket-io/socket.js',
             '{client,node_modules}/bower_components/angular-ui-router/release/angular-ui-router.js',
             '{client,node_modules}/socket.io-client/socket.io.js' ] },
        { dest: '.tmp/concat/app/app.js',
          src:
           [ '{.tmp,client}/app/app.js',
             '{.tmp,client}/app/about/about.controller.js',
             '{.tmp,client}/app/about/about.js',
             '{.tmp,client}/app/account/account.js',
             '{.tmp,client}/app/account/login/login.controller.js',
             '{.tmp,client}/app/account/settings/settings.controller.js',
             '{.tmp,client}/app/account/signup/signup.controller.js',
             '{.tmp,client}/app/admin/admin.controller.js',
             '{.tmp,client}/app/admin/admin.js',
             '{.tmp,client}/app/main/main.controller.js',
             '{.tmp,client}/app/main/main.js',
             '{.tmp,client}/components/auth/auth.service.js',
             '{.tmp,client}/components/auth/user.service.js',
             '{.tmp,client}/components/modal/modal.service.js',
             '{.tmp,client}/components/mongoose-error/mongoose-error.directive.js',
             '{.tmp,client}/components/navbar/navbar.controller.js',
             '{.tmp,client}/components/socket/socket.service.js' ] } ] } }

  uglify:
  { generated:
   { files:
      [ { dest: 'dist/public/app/vendor.js',
          src: [ '.tmp/concat/app/vendor.js' ] },
        { dest: 'dist/public/app/app.js',
          src: [ '.tmp/concat/app/app.js' ] } ] } }

  cssmin:
  { generated:
   { files:
      [ { dest: 'dist/public/app/vendor.css',
          src: [ '.tmp/concat/app/vendor.css' ] },
        { dest: 'dist/public/app/app.css',
          src: [ '.tmp/concat/app/app.css' ] } ] } }

Running "autoprefixer:dist" (autoprefixer) task
File .tmp/app/app.css created.

Running "ngtemplates:main" (ngtemplates) task
>> No templates found
File .tmp/templates.js created.
Added .tmp/templates.js to <!-- build:js app/app.js -

Running "ngtemplates:tmp" (ngtemplates) task
File .tmp/tmp-templates.js created.
Added .tmp/tmp-templates.js to <!-- build:js app/app.

Running "concat:generated" (concat) task
File .tmp/concat/app/vendor.css created.
File .tmp/concat/app/app.css created.
File .tmp/concat/app/vendor.js created.
File .tmp/concat/app/app.js created.

Running "ngAnnotate:dist" (ngAnnotate) task
>> 2 files successfully generated.

Running "copy:dist" (copy) task
Created 68 directories, copied 356 files

Running "cdnify:dist" (cdnify) task
Going through dist/public/index.html to update script refs
✔ bower_components/jquery/dist/jquery.js changed to //ajax.googleapis.com/ajax/libs/jquery/1
✔ bower_components/angular/angular.js changed to //ajax.googleapis.com/ajax/libs/angularjs/1
✔ bower_components/angular-cookies/angular-cookies.js changed to //ajax.googleapis.com/ajax/
✔ bower_components/angular-resource/angular-resource.js changed to //ajax.googleapis.com/aja
✔ bower_components/angular-sanitize/angular-sanitize.js changed to //ajax.googleapis.com/aja

Running "cssmin:generated" (cssmin) task
>> Destination not written because minified CSS was empty.
File dist/public/app/app.css created: 146.58 kB → 120.4 kB

Running "uglify:generated" (uglify) task
File dist/public/app/vendor.js created: 1.8 MB → 398.01 kB
File dist/public/app/app.js created: 26.05 kB → 16.02 kB

Running "rev:dist" (rev) task
dist/public/app/app.js >> 4e83113a.app.js
dist/public/app/vendor.js >> b05ecff2.vendor.js
dist/public/app/app.css >> 32a41561.app.css
dist/public/assets/images/yeoman.png >> d535427a.yeoman.png

Running "usemin:html" (usemin) task

Processing as HTML - dist/public/index.html
Update the HTML to reference our concat/min/revved script files
<script src="app/vendor.js"" changed to <script src="app/b05ecff2.vendor.js"
<script src="app/app.js"" changed to <script src="app/4e83113a.app.js"
Update the HTML with the new css filenames
<link rel="stylesheet" href="app/app.css"" changed to <link rel="stylesheet" href="app/32a415
Update the HTML with the new img filenames
Update the HTML with data-main tags
Update the HTML with data-* tags
Update the HTML with background imgs, case there is some inline style
Update the HTML with anchors images
Update the HTML with reference in input

Running "usemin:css" (usemin) task

Processing as CSS - dist/public/app/32a41561.app.css
Update the CSS to reference our revved images

Running "usemin:js" (usemin) task
```

```
Execution Time (2014-10-13 04:48:30 UTC)
clean:dist           471ms  ▇▇ 2%
concurrent:dist       1.6s  ▇▇▇▇▇▇▇ 8%
wiredep:target       850ms  ▇▇▇▇ 4%
autoprefixer:dist    260ms  ▇ 1%
concat:generated     399ms  ▇▇ 2%
ngAnnotate:dist      925ms  ▇▇▇▇ 5%
copy:dist             4.7s  ▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇ 24%
cdnify:dist           5.4s  ▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇ 27%
cssmin:generated     596ms  ▇▇▇ 3%
uglify:generated        4s  ▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇ 20%
Total 19.8s
```

heig-vd

Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud



References

MUST READ for the Tests

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- **Understanding the Node.js Event Loop**
  - http://strongloop.com/strongblog/node-js-event-loop/
- **Mixu's Node book: What is Node.js? (chapter 2)**
  - http://book.mixu.net/node/ch2.html
- **Node.js Explained, video**
  - http://kunkle.org/talks/