
BUILDING CROSS-PLATFORM APPS
WITH HTML5
Giorgio Natili
@GIORGIONATILI

AGENDA

- App Scaffolding
- Dependencies Management
- Reusable CSS
- ECMA6
- Web Components
- Routing

APP SCAFFOLDING

YEOMAN

Kickstart tool that enforces best practices

INSTALL YEOMAN

— install brew

— install *node* and *npm*

```
$ brew install node
```

— install *yeoman* through *npm*

```
$ npm install yo -g
```

USING YEOMAN (YO!)

- just type

```
$ yo
```

- and then
  - run a generator
  - update a generator
  - get some help
  - exit

A GENERATOR?!?

YEOMAN GENERATORS

Receipes for modern, modular, testable and scalable web apps

AMONG THE OTHERS: ANGULAR, WEBAPP, H5BP, IONIC, REACT, EXPRESS, GULP-WEBAPP, ETC.

CREATE A SIMPLE HTML5 APP

EXPLORE THE OPTIONS OFFERED FROM A GENERATOR


MIMOSA
A toolkit for modern browser development

MIMOSA INCLUDES

- JS Transpiler
- CSS Pre-processors
- HTML Micro-templaters
- Minification
- AMD/RequireJS Support

INSTALL MIMOSA

as a *npm* module

```
$ npm install mimosa -g
```

OUT OF THE BOX TRANSPILER SUPPORT

Transpiling is based on file extension: CoffeeScript compiler = .coffee, LiveScript compiler = .ls, etc.

(The same rule applies also to CSS files)

MICRO TEMPLATES

Mimosa offers several HTML micro templates engines:

- Dust
- Embedded CoffeeScript Templates (ECO)
- Embedded JavaScript Templates (EJS)
- Handlebars
- Hogan
- Jade
- Etc.

DEPENDENCY MANAGEMENT

Mimosa works with Bower to install your dependencies where they belong.

It keeps track of your Bower situation to rerun Bower installs when you need them

AMD SUPPORT

Mimosa fully supports AMD through requireJS, infact:

- AMD Path Verification
- Notify when AMD paths are broken or wrong
- Handle shims, maps
- Nested requires and defines
- Detect circular dependencies
- Manage common config for multiple modules

HINT & LINT

JavaScript files are hinted and CSS files are linted (configurable)

SOURCE MAPPING

Source mapping is fully supported, and it's possible to add breakpoints in .coffee files and use the standard debugging tools

CREATE A WEB APP WITH MIMOSA

```
$ mimosa skel:new webapp hello-app-mimosa
```

```
06:16:50 - Success - mimosa-web-package successfully installed into your project.
06:16:50 - Info - Did not find compiled directory public, so making it for you
06:16:51 - Info - Starting Bower install...
06:16:53 - Info - mimosa-bower created file assets/javascripts/vendor/requirejs/require.js
06:16:53 - Info - mimosa-bower created file assets/javascripts/vendor/jquery/jquery.js
06:16:53 - Info - Cleaned temp bower output directory .mimosa/bower/bower_components
06:16:53 - Info - Watching assets
06:16:53 - Success - Wrote file public/javascripts/main.js
06:16:53 - Success - Wrote file public/stylesheets/style.css
06:16:53 - Success - Wrote file public/javascripts/app/example-view.js
06:16:53 - Success - Wrote file public/javascripts/specs/example-view_spec.js
06:16:53 - Success - Wrote file public/javascripts/vendor/requirejs/require.js
06:16:53 - Success - Wrote file public/javascripts/vendor/jquery/jquery.js
06:16:54 - Success - Mimosa is starting your server: server.js
Express server listening on port 3000
06:16:55 - Success - 0 of 0 tests passed for .mimosa/testemRequire/testem.json.
```

BRUNCH

An ultra-fast HTML5 build tool

WHAT IT DOES

- Compiles scripts/stylesheets/templates
- Lints JS and CSS
- Wraps in Common.js or AMD modules (optional)
- Concatenates & minifies
- Generates source maps
- Copies and optimizes image assets
- Watches for file changes
- Notifies you of errors via console or notifications
- Auto reload (plugin)

IT SOUNDS LIKE MIMOSA

FEW IMPORTANT DIFFERENCES

- Brunch preceeds mimosa by 5-6 months
- as far as today Brunch seems more active on github
- Mimosa offers seprate compile phases for server and client side code
- Mimosa offers *CoffeeScript* breakpoints
- Mimosa allows to override configuration through profiles

Working on Lab 1

WHAT'S ABOUT GRUNT AND GULP?

Grunt is merely a set of tasks that would be executed one after another

Gulp doesn't make temporary files, that's why it's a "streaming build system"

Brunch is focused on speed

GETTING STARTED

```
$ npm install -g brunch
$ brunch new [skeleton url] [output directory]
$ brunch watch --server
```

A HUGE SET OF SKELETONS

Between the others: ember, jade, angular, chaplin, marionette, cordova, foundation, etc.

A complete list is available here.

CREATE AN HTML5 STATIC PAGE WITH BRUNCH

```
$ brunch new gh:dobromir-hristov/brunch-with-whey-protein
```


SLUSH
The streaming scaffolding system

WHY IS IT DIFFERENT?

It uses Gulp, it doesn't offer anything out of the box and for this reason is pretty light

(Highly recommended if you want to have control of every single depndence that you add to your project)

SLUSH GENERATORS

A huge list that includes: phonegap, go, Electron, angular, ember, backbone, express, web components, h5bp, ng-next, browserify, bootstrap, etc.

A comprehensive list of slush generators are available here

(*Slush generators can be installed locally!*)

GETTING STARTED

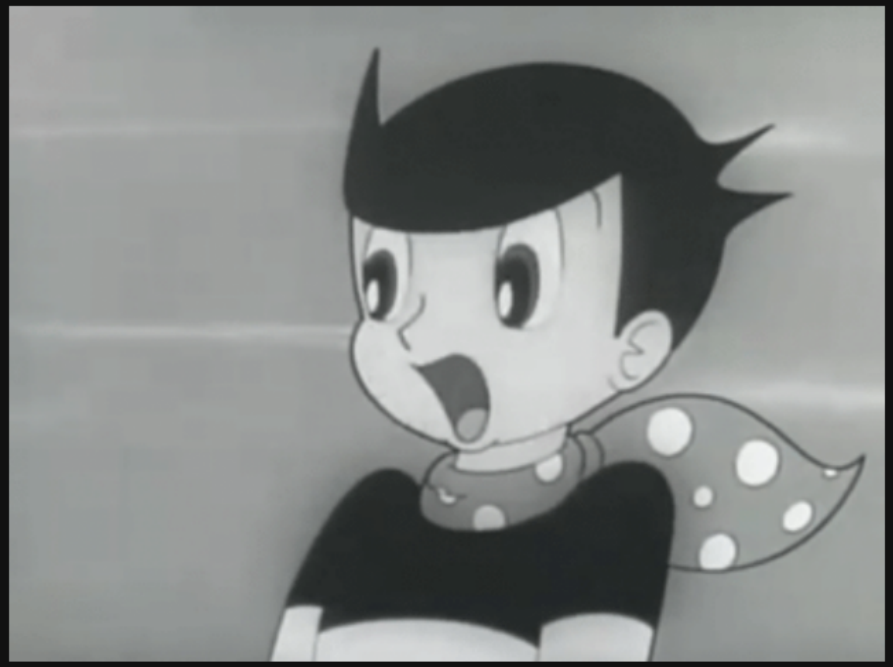
INSTALL AS AN NPM MODULE

```
$ npm install slush -g
```

USE ONLY THE TASKS YOU NEED FROM A GENERATOR

```
$ slush <generator>[:<tasks>] [<args>]
```

HOW TO DECIDED WHICH IS THE RIGHT TOOL?



WITH COMMON SENSE AND DEEP ANALYSIS OF YOUR REQUIREMNTS

- Do you have huge dependecies tree?
- Is the team big and distributed?
- Is the project enterprise?
- Are you already using tools like *ember-cli*?
- Do you already have a test suite?
- How much active is the project community?

Working on Lab 2

DEPENDENCIES MANAGEMENT


NPM
Installs, publishes and manages node programs

NODE PACKAGE MANAGER

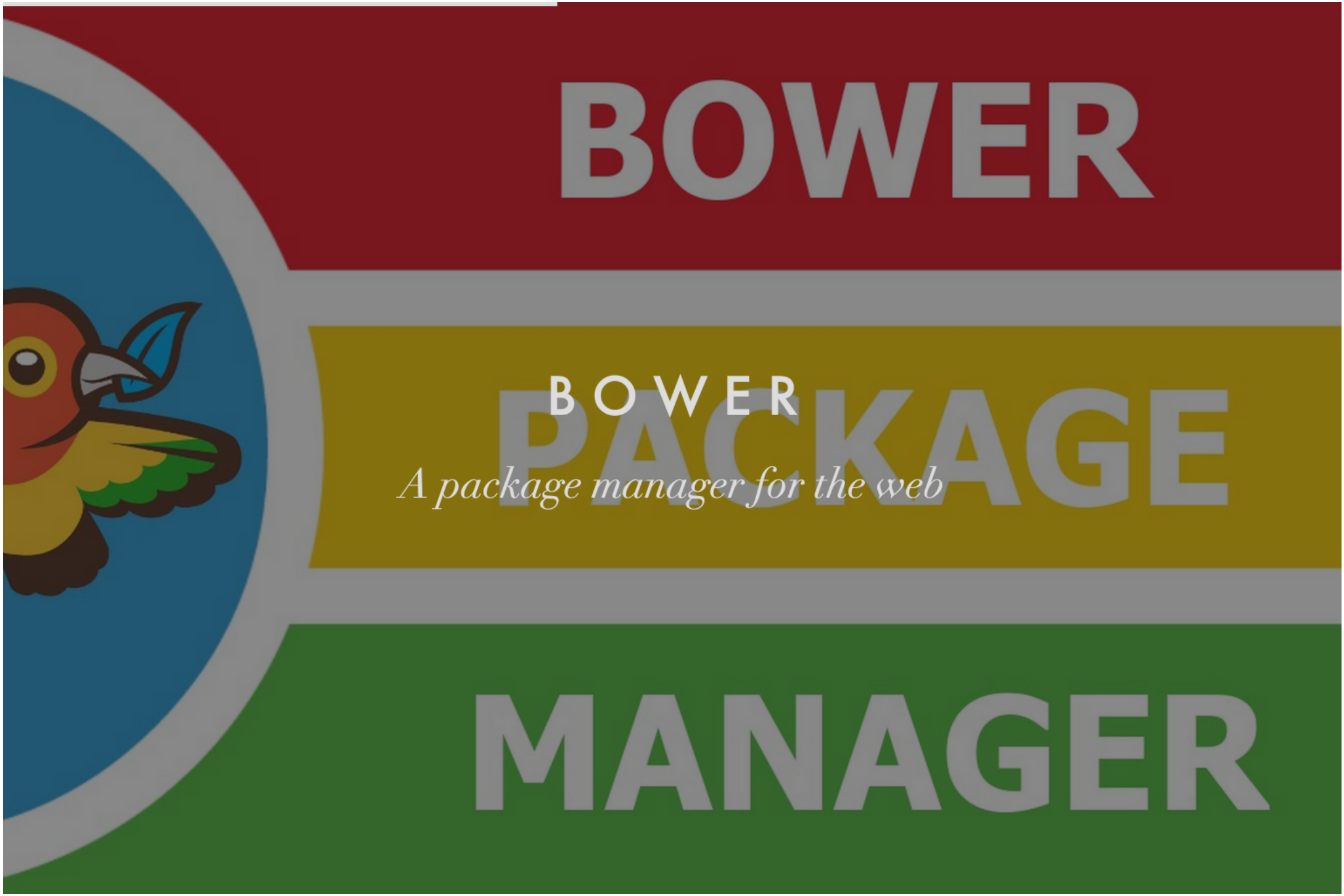
- Originally designed only for server side dependencies
- Nested dependencies (each module has its own dependencies tree)
- Can run build scripts and register utilities to your CLI

BASIC COMMANDS

- `$ npm install [--save,--save-dev,--save-optional]`
- `$ npm update <module-name>`
- `$ npm prune`
- `$ npm uninstall  <module-name>`
- `$ npm shrinkwrap`

THE PACKAGE.JSON FILE

```
{
  "name": "test",
  "version": "1.0.0",
  "description": "Test project",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Giorgio Natili",
  "license": "ISC"
}
```



BOWER

A package manager for the web

BOWER

- A npm module
- Used to search and install client side dependencies
- Based upon a flat dependency tree
- Built in order to automatically resolve version conflicts

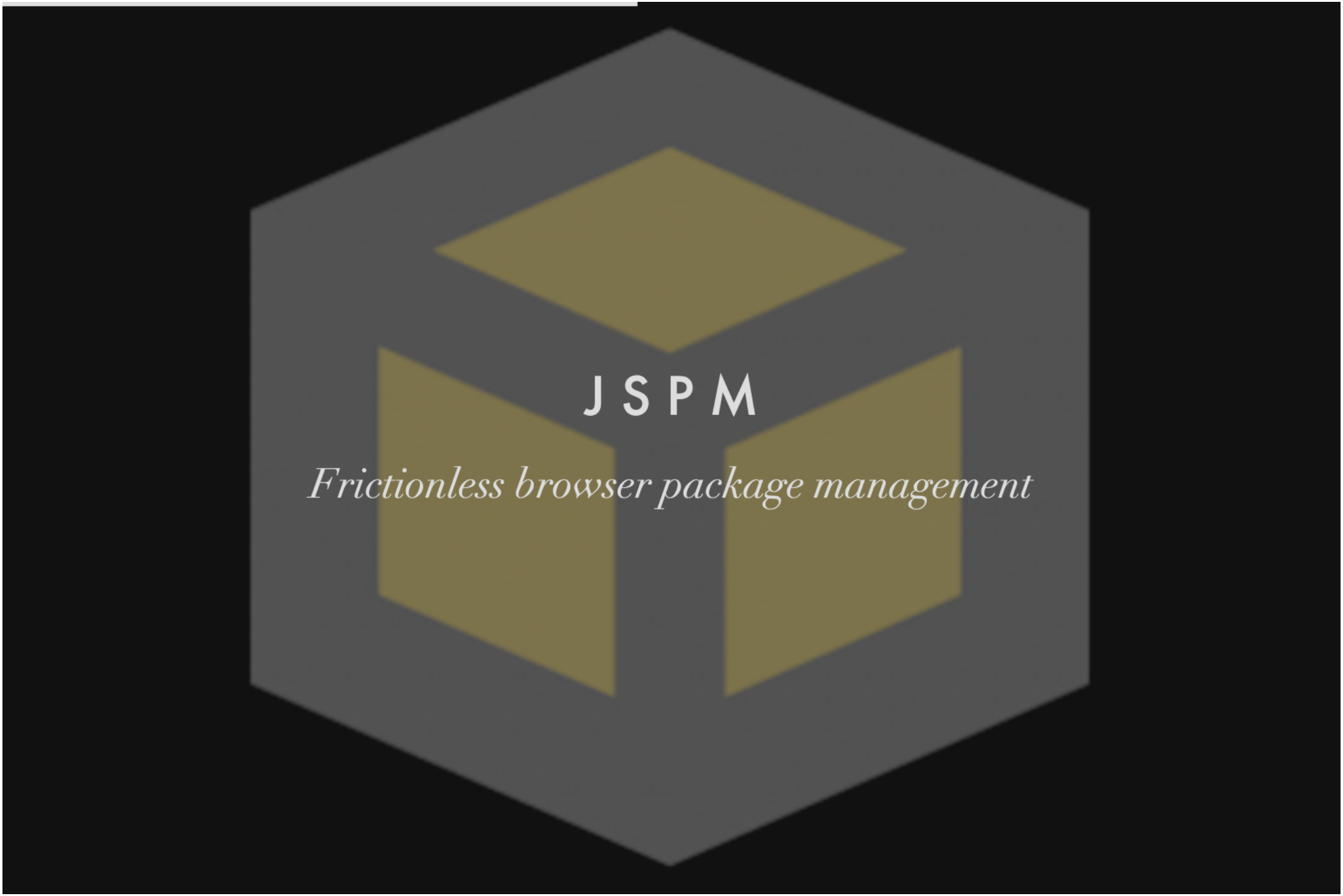
BASIC COMMANDS

- `$ bower init`
- `$ bower install [--save,--save-dev]`
- `$ bower cache [list,clean] <package>`
- `$ bower prune <package>`
- `$ bower search`
- `$ bower update  <package>`

IT'S NOT THE SAME AS NPM!

IT'S ALSO NOT THE ONLY WAY!

Working on Lab 3


JSPM
Frictionless browser package management

WHAT IS JSPM

- It's a package manager for the *SystemJS* universal module loader
- Loads any module format such as ES6, AMD, CommonJS from any registry (github, npm, etc.)
- Uses a flat versioned dependency management tree
- Its main focus is front-end dependencies

BASIC COMMANDS

- `$ jspm init`
- `$ jspm install [npm:,githhub:]<package>`
- `$ jspm run <module>`
- `$ jspm bundle app/main main-bundle.js --inject`

(For a detailed guide about the jspm CLI refer to the https://github.com/jspm/jspm-cli)

WHY ITS DIFFERENT FROM BOWER

— Bower has only one purpose-to download source files you need from the web to your hard disk

— JSPM is not only a module downloader, it manages dependency injection in your app

— JSPM offers out of the box *traceur* or *babel* to run *ES6*

— It suppors modules import

```
var $ = require('jquery'); // AMD-style import!! Magic!!
```

ECMA 6

FEATURES REVIEW

THE LET KEYWORD

Declares a variable local to the block scope...

```
var x = 1;
if(x < 10) {
  let v = 1;
  v = v + 21;
  console.log(v);
}
console.log(v); //v is not defined>)
```

THE CONST KEYWORD

Declares a "variable" that once is set can't change...

```
const username = 'Giorgio';
username = 'Jonathan'; // "username" is read-only
```

TEMPLATE STRINGS

Concatenates strings and variable wrapping variables in a ${} sign...

```
let name = 'giorgio';
let email = 'me@webplatform.io';
console.log(`username ${name} and e-mail ${email}`);
```

ARROW FUNCTIONS

It's a shorter form of a function that lexically binds the this value...

```
(param1, param2, paramN) => { statements }
(param1, param2, paramN) => expression
```

VARIADIC ARGUMENTS AND DEFAULT VALUES

Functions can be now defined to explicitly accept a vairable number of arguments and default values...

```
function printAll(...names){
  for (item of names) {
    console.log(item);
  }
}
```

DESTRUCTURING ASSIGNEMENT

Expression that makes it possible to extract data from arrays or objects using a syntax that mirrors the construction of array and object literals...

```
var [name, surname, age] = ["Giorgio", "Natili", 40];
console.log(name, surname, age);
```

ITERATORS

Iterators allow iteration over arbitrary objects…

```
function RangeIterator(min, max) {
   this[Symbol.iterator] = function () {
      var current = min;
      return {
        next: function () {
          return {
            done: /*some conditions */ true,
            value: /*the current value */ Math.random()
          };
        };
}
```

GENERATORS

With ES6 generators we can pause and resume multiple times the execution of a function...

```
function *foo(x) {
  var y = 2 * (yield (x + 1));
  var z = yield (y / 3);
  return (x + y + z);
}
```

ES MODULES

WHAT IS A MODULE?

- In software, a module is a part of a program
- Programs are composed of one or more independently developed modules that are combined together trough events or injection
- A single module can contain one or several functions

ES6 MODULES

- Each module is defined in its own file
- The functions or variables defined in a module are not visible outside unless you explicitly export them
- The export keyword exports the desired functions
- The default keyword indicates that this is a default export and doesn't require a name

SYNTAX

```
// Saved in a file math-utils.js
function sum(num1, num2) {
  return num1 + num2;
}
export { sum as add };
```

```
// Using the module
import {add} from 'math-utils'
```

ABOUT IMPORT AND EXPORT

— A module can export multiple things by prefixing their declarations with the keyword export

```
export const sqrt = Math.sqrt;
```

— The default exported value is the more important and is unique in each module

```
export default function () { ... };
```

— When importing a module you can decide the variable name to use to store the default export

```
import _ from from 'underscore';
```

IMPORT ALIASES

— As for the export, imported properties can be aliased (aka as)

```
import {add as summ} from 'math-utils';
```

— Imported values can be then re-exported with an appropriate alias

```
import { sum } from 'math-utils';
export { sum as add };
```

USING ECMA6 TODAY

TRACEUR

Traceur is a JavaScript.next-to-JavaScript-of-today compiler that allows you to use features from the future today...

BABEL

It's the new name of 6to5, it's a tool that translates the new ES6 (aka ES2015) syntax to a browser compliant JavaScript...

BABEL VS TRACEUR

- Both offer the same ES6 support
- Transpiling performance is kind of the same
- Transpiled code performance is pretty similar
- Bable transpiled code is more readable
- Babel supports JSX (aka ReactJS)

ES6 WITH JSPM

```
$ npm install live-server -g
```

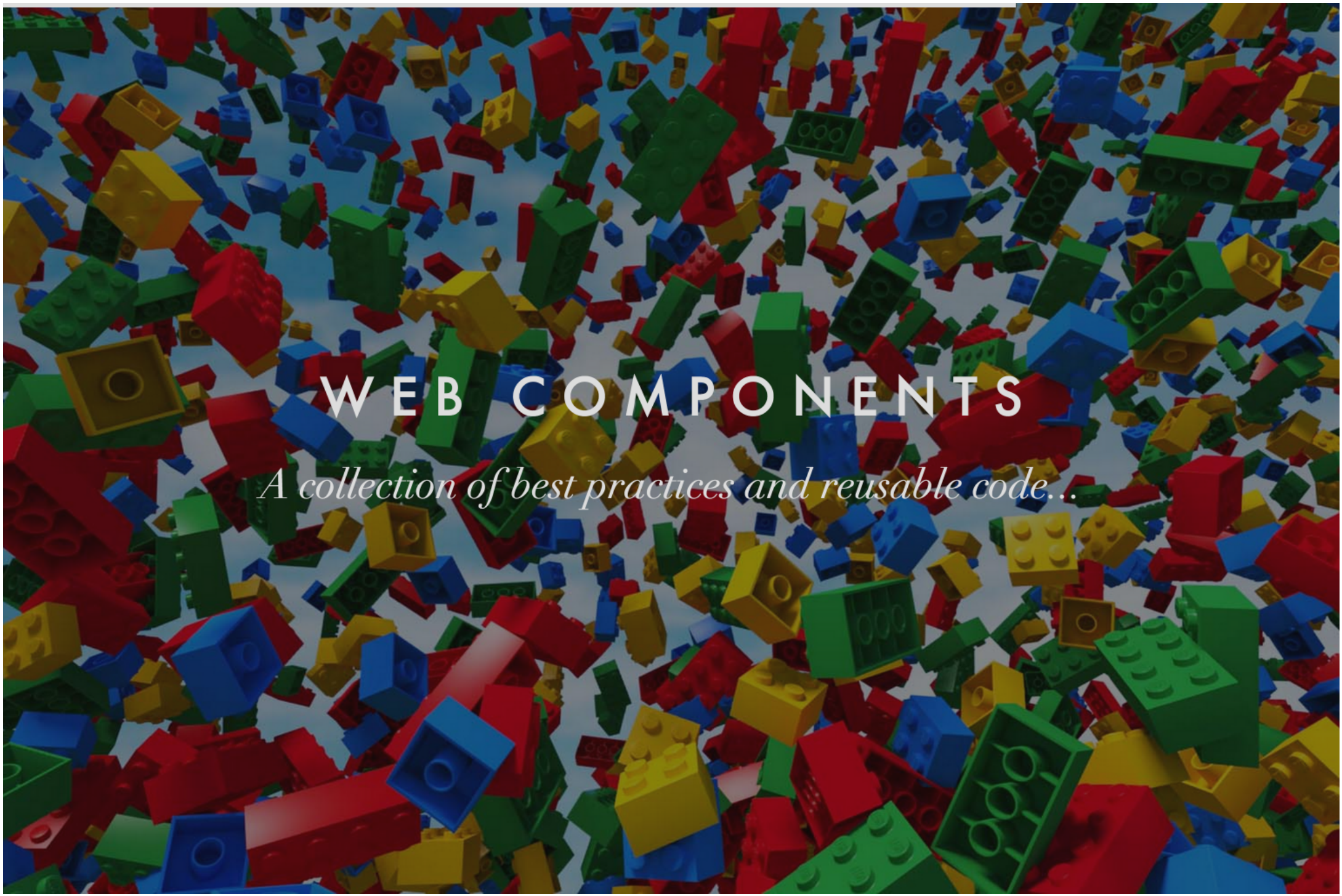
```
$ jspm init
```

```
<script src="jspm_packages/system.js"></script>
<script src="config.js"></script>
<script>
  System.import('scripts/app');
</script>
```

LIVE DEMO

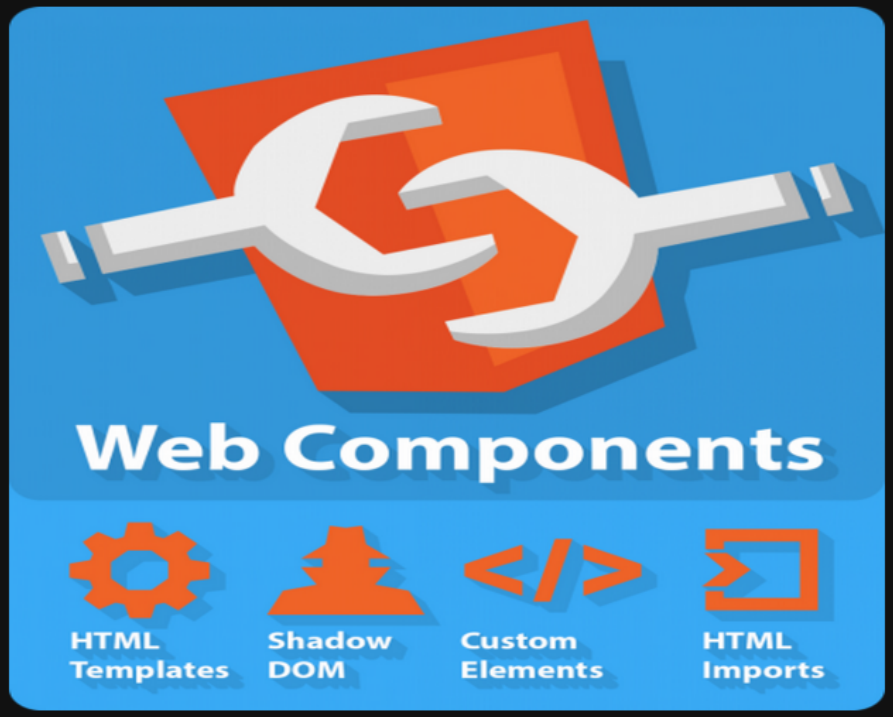
Working on Lab 4

WEB COMPONENTS

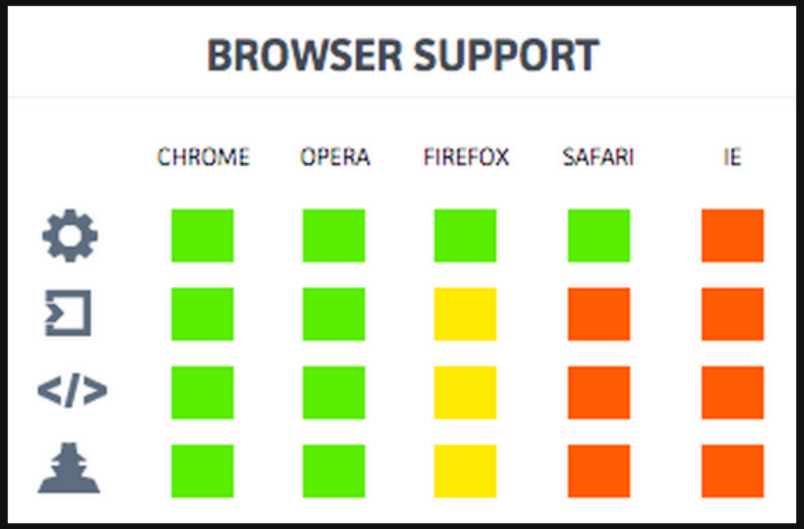

WEB COMPONENTS
A collection of best practices and reusable code...

IN A NUTSHELL

In a nutshell, they allow us to bundle markup and styles into custom HTML elements…



BROWSER SUPPORT



SURPRISED?

SHADOW DOM

It's the DOM inside a widget, encapsulated from the rest of the page in order to prevent conflicts...

- It offers CSS encapsulation
- It doesn't allow the co-existence of different versions of the same library in the same page

VANILLA SHADOW DOM

```
<button> Hello, world! </button>
<script>
var host = document.querySelector('button');
var root = host.createShadowRoot();
root.textContent = 'Whatever!';
</script>
```

POLYMER

Polymer is a library inspired by the current web components standards for building custom elements...

```
<dom-module id="hello-world">
    <template>
      <p>Hello <strong>{{who}}</strong> :)</p>
    </template>
```

```
Polymer({
    is: 'hello-world',
      properties: {
```

Hello **Unicorn** :)

X-TAG

It's a JavaScript library created and Maintained by Mozilla that brings Web Components Custom Element to all modern browsers...

```
xtag.register('x-accordion', {
    // extend existing elements
    extends: 'div',
      lifecycle:{
      created: function(), inserted: function(), removed: function()
```

```
<hello-world who="Unicorn"></hello-world>
```

VANILLAJS

```
(function() {
    // Creates an object based in the HTML Element prototype
    var element = Object.create(HTMLElement.prototype);

    // Fires when an instance of the element is created
    element.createdCallback = function() {};

    // Fires when an instance was inserted into the document
    element.attachedCallback = function() {};

    // Fires when an instance was removed from the document
    element.detachedCallback = function() {};

    // Fires when an attribute was added, removed, or updated
    element.attributeChangedCallback = function(attr, oldVal, newVal) {};

    // Registers custom element
    document.registerElement('hello-world', { prototype: element });
```

ARE YOU KIDDING ME?

WEB COMPONENTS AND GENERATORS

Install this https://github.com/webcomponents/generator-element interesting *yeoman* generator...

Working on Lab 5

(And explore the world of custom tasg https://customelements.io/!)

ROUTING

APPLICATION ROUTING

- Most times in a Single Page App (aka SPA), the appropriate resources are loaded when the URL changes
- The URL structure is used to specify which views should be rendered/loaded
- Its through the URL that a SPA understand which data have to be fetched from the server

APPLICATION STATE

The application state is mostly persisted in the client through information fragments used by the server to rebuild the actual state...

DON'T REINVENT THE WHEEL!

- PageJS
- Director
- Bespoke-hash
- Mithril

(ok, that's a micro framework it doesn't count!)

PAGEJS

```
Navigation page(path);
```

```
Matching paths
page('/about', callback);
```

```
Bindings
page([click|popstate|dispatch|hashbang])
```

(Routes are passed Context objects, these may be used to share state)

DIRECTOR

Routing Events

```
on, before, after, once
```

Scoped Routing

```
// Create routes inside the `users` scope.
router.path(/\/users\/(\w+)/, function () {
```

Wildcard Routes

```
router.on("/foo/?((\w|.)*)"), function (path) { });
```

Async Routes

```
var router = new director.http.Router().configure({ async: true });
```

ISN'T THIS INTERESTING?

CONCLUSIONS

Using frameworks is not bad, understanding what's behind the scene is the key point to get the most value from them...

Preferring polyfills to frameworks it's possible to remove not needed code once a feature is supported by browsers....

A technology should be used if it's the right tool to address product needs - not because it's our favorite tool...

An HTML5 based app can be wrapped as a native app in platforms like Android, iOS, Firefox OS, Windows and can be used with Electron to build destop apps...

THAT'S IT

Thanks so much!

@giorgionatili