Open

WEB ENGINEERING

Open

PLAN

- Un peu d’histoire
- JavaScript : le langage d’action du Web (mais pas que …)
- Outillage
- Nodejs
- TypeScript
- Utilisation de framework : AngularJS
- Une autre vision GWT

Open

CONTEXTE HISTORIQUE

- Internet (~1980) et le Web (~1990)
  - Repose sur Internet, TCP/IP, modèle OSI
  - Un des services de l'Internet (port 80)
- 3 piliers : HTML, URI, HTTP
- Plusieurs évolutions
  - Séparation mise en forme (CSS), pages
  - Statiques vers interactions dynamiques
  - Cloud computing, Web sémantique, Web embarqué, systèmes pervasifs...
  - HTML5 (Conteneur d’applications complexes)

Open

ÉVOLUTIONS DU WEB

- Évolutions du Web => Web 2.0
  - Utilisateur proactif (wikipedia, blogs, etc.)
  - Interfaces riches, intégration d'applications
- Moyens techniques
  - Langage de scripts
  - côté serveur (PHP, ASP, C#...)
  - côté client (Javascript, flash...)
- Services Web (HTML => XML,JSON)
  - Échange de données entre applications

Open

JAVASCRIPT

Open

JAVASCRIPT

- Définition
- Syntaxe
- Les objets
- Les évènements
- Les boîtes de dialogue
- Accès à un élément quelconque d'une page

Open

JAVASCRIPT

- Mais à quoi ça sert ?
  - Manipulation et animation dynamiques du contenu des pages Web
  - Ouverture, fermeture de fenêtres
  - Communication avec « le » serveur
    - Warning : cross-domain scripting
  - Enregistrement des actions de l'utilisateur (!)
  - Réaction aux évènements utilisateur
  - Sauvegarde de données...

Open

DÉFINITION

JavaScript est un langage de programmation orienté objet développé par NETSCAPE dans les années 90, et s'appelait à l'origine LiveScript.

- Il fut adopté par la firme SUN (qui est à l'origine du langage JAVA) en 1995.
- Le JavaScript est une extension du langage HTML.

Open

POPULARITÉ DE JAVASCRIPT (1)

- JavaScript, le langage de programmation populaire avec la plus importante progression (mais pas le plus grand nombre de lignes de code déployées)
- JavaScript langage de l'année 2014 et 2015 selon l'Index TIOBE (nombre de développeurs + nombre de tutoriaux + nombre de vendeurs tiers, sur la base des recherches dans les moteurs Google, Bing, Yahoo!, Wikipedia, Amazon, YouTube et Baidu)

Open

POPULARITÉ DE JAVASCRIPT (2)

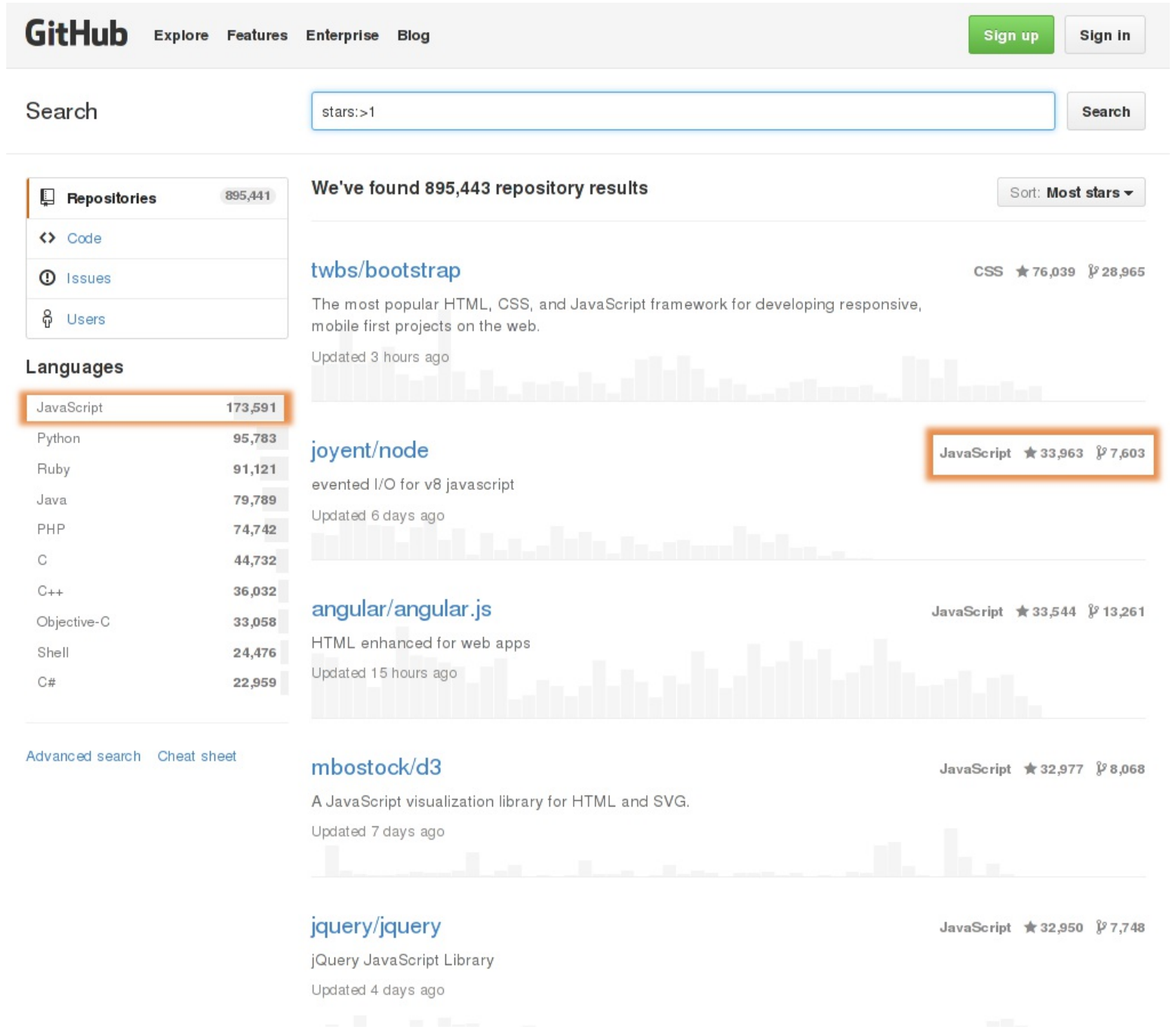
http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

Open

POPULARITÉ DE JAVASCRIPT (3)

- Autres indicateurs de popularité :
  - GitHub (plus nombre de dépôts de source avec JavaScript et plus projets JavaScript les plus populaires)
    - http://adambard.com/blog/top-github-languages-2014/
    - http://redmonk.com/sogrady/2014/01/22/language-rankings-1-14/
    - http://www.thoughtworks.com/radar/
    - http://pypl.github.io/PYPL.html

Open



https://github.com/search?q=stars%3A%3E1&s=stars&type=Repositories

Open

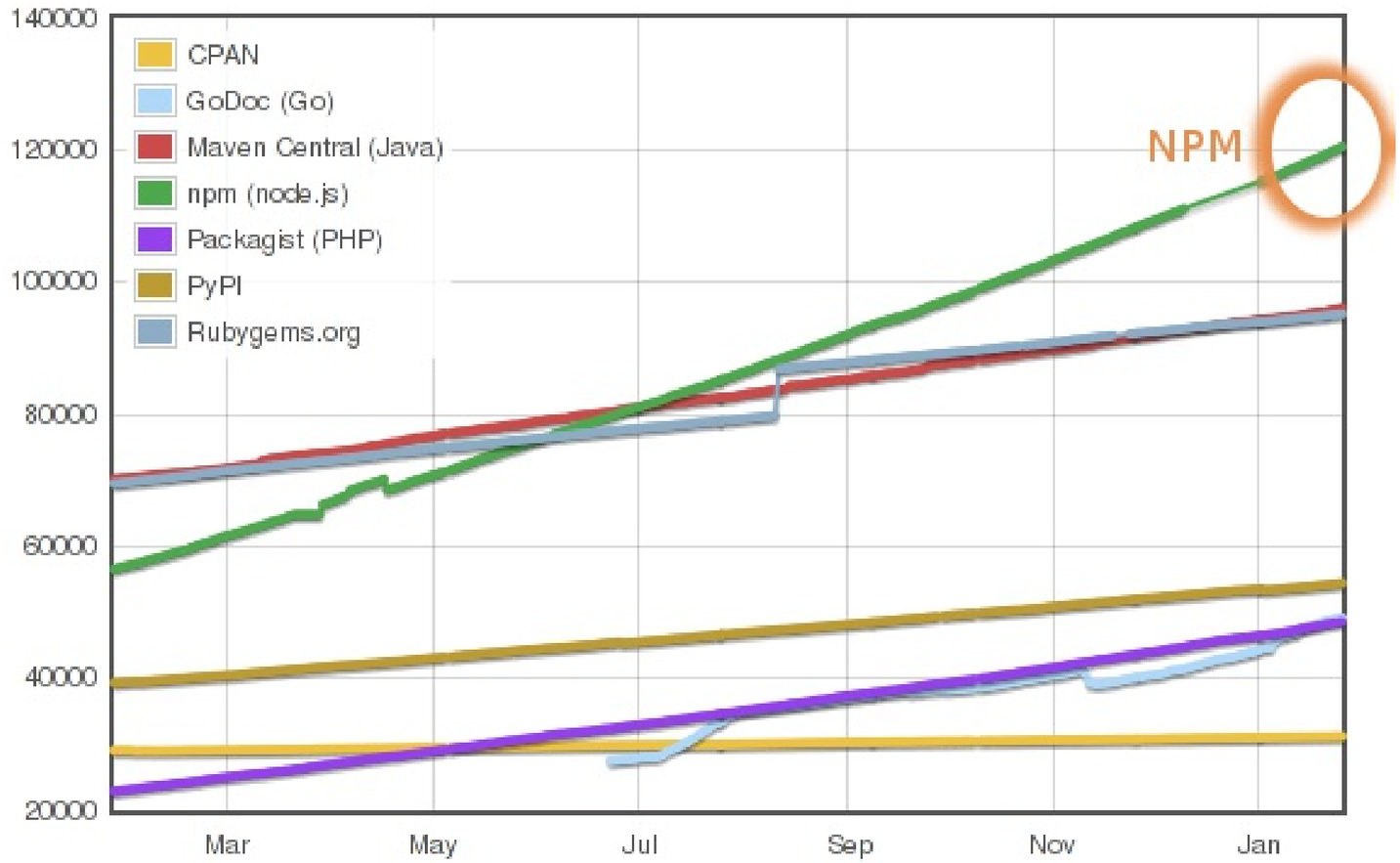
POPULARITÉ DE JAVASCRIPT (4)

- JavaScript présent dans tous les navigateurs web avec maintenant des implémentations à jour
- ECMAScript 6 disponible et implémentation progressive dans les différents moteurs
- Node.js et NPM à la base du nouveau succès de JavaScript (après avoir été le langage le plus détesté)
- Le plus de modules tiers disponibles grâce à NPM et GitHub. Il existe un module JavaScript existant pour quasiment chaque besoin, ou bien il est en train d'être écrit !

Open

POPULARITÉ DE JAVASCRIPT (5)

http://modulecounts.com/



Open

NODE.JS CHOISI PAR LES ENTREPRISES DU WEB

- Plus de téléchargements des modules NPM durant la semaine que le week-end et les vacances
- Les modules JavaScript sont plus utilisés et développés dans le cadre des entreprises que dans le cadre des loisirs
- Node.js utilisé en production par Dow Jones, eBay, Walmart, PayPal, LinkedIn, Yahoo!, The New York Times, Über, Airbnb, Groupon, SAP, Rakuten, TF1, Google
- Node.js supporté officiellement par Oracle (http://nodejs.org/industry/)

Open

D'AUTRES CHOIX

- D'autres acteurs n'utilisent pas Node.js côté serveur :
  - Chez Twitter c'est du Java (après avoir débuté avec du Ruby On Rails)
  - Chez Facebook c'est du Hack (un nouveau PHP)
  - Chez GitHub c'est du Ruby On Rails et du Erlang
  - Google utilise d'autres langages que JavaScript et d'autres plateformes que Node.js : notamment Go (en remplacement de C++)

Open

UBIQUITÉ

- On peut utiliser JavaScript :
  - Sur navigateur pour sites web et web apps
  - Sur serveur pour applications web
  - Sur navigateur + serveur pour applications isomorphiques
  - Sur serveur et poste développeur pour scripts/batches systèmes (export CSV, transformation/conversion de données, etc.)
  - Sur serveur et poste développeur pour scripts web (SlimerJS, PhantomJS, etc.)

Open

JAVASCRIPT

- Spécification de référence: ECMAScript-262 (édition 6.0 – 2015)
- Fonctionnalités additionnelles incompatibles
- Caractéristiques :
  - Langage interprété
  - moteur intégré dans les navigateurs
  - Basé objet mais orienté prototype
  - Dynamique (typage, fonctions, code...)
  - Événementiel

Open

ECMASCRIPT 6

- La spécification ECMAScript 6 est aussi appelée ES6, Harmony ou ES.next
- Finalisation de la spécification Juin 2015
- Les différents moteurs JavaScript implémentent déjà progressivement ECMAScript 6 depuis des années en évoluant en même temps que la spec
- Apporte ce qui manquait à JavaScript pour faciliter la vie des développeurs (et qu'on pouvait déjà trouver dans d'autres langages comme Python, Java, etc.) : modules, collections, template strings, iterateurs, générateurs, etc.

Open

JAVASCRIPT

- Orienté “prototype” ???
  - Basé objet mais pas orienté objet
  - On déclare un objet générique, puis héritage
- Dynamique ???
  - Typage, fonctions, code...
  - Pratique mais dangereux...
- Événementiel ???
  - Paradigme de programmation
  - Attente” puis réaction aux actions “utilisateur”

Open

DÉFINITION

Javascript	Java
Langage interprété	Langage compilé
Code intégré au HTML	Code (applet)
Langage peu typé	Langage fortement typé
Liaisons dynamiques: les références des objets sont vérifiées au chargement	Liaisons statiques: Les objets doivent exister au chargement (compilation)
Accessibilité du code	Confidentialité du code
Sûr: ne peut pas écrire sur le disque dur	Sûr: ne peut pas écrire sur le disque dur

Open

JAVASCRIPT

- Définition
- **Syntaxe**
- Les objets
- Les évènements
- Les boîtes de dialogue
- Accès à un élément quelconque d'une page

Open

SYNTAXE

- Intégration du script dans la page

```
*inline*
<script type=”text/javascript”>....</script>
```

Appel externe avec une réf. comme pour css

```
<script src=”/js/script.js” type=”text/javascript”/>
```

- Chargement du script
  - Importance du placement dans la page
  - Dans le header (délai)
  - Dans le body, ou en fin de body
  - Cache si appel externe (mieux), compression

Open

SYNTAXE

- Bonnes pratiques
  - Fichier .js externe
- Lecture par événement “onload”
- “Separation of concerns” (Dijkstra)
  - Pour le script principal
  - Utilisation de fonctions
- chargement dynamique (cf plus tard)

Open

SYNTAXE

- Le code JavaScript se trouve dans un fichier ayant pour extension .js
- L'inclusion de ce fichier se fait généralement au début des pages HTML grâce à la balise script:

```
<script type="text/javascript" src="url/nomdufichier.js"> </script>
```

Open

SYNTAXE

- Les structures conditionnelles ou répétitives (boucles)ont la même syntaxe qu'en langage C.
- JavaScript est un langage faiblement typé.

Open

SYNTAXE

PORTÉE DES VARIABLES:

- Une variable déclarée en début de script sera considérée comme étant globale.
- Les variables utilisées/déclarées dans une fonction restent locales à la fonction.

Open

SYNTAXE

- The JavaScript syntax is similar to C# and Java
  - Operators (**+, *, =, !=, &&, ++, …)**
  - Variables (typeless)
  - Conditional statements (**if**, **else**)
  - Loops (**for**, **while**)
  - Arrays (**my_array**[]) and associative arrays (**my_array['abc']**)
  - Functions (can return value)
  - Function variables (like the *C#* delegates)

Open

JAVASCRIPT DATA TYPES

- Numbers (integer, floating-point)
- Boolean (true / false)
- String type – string of characters

```
var myName = "You can use both single
            or double quotes for strings";
```

- Arrays

```
var my_array = [1, 5.3, "aaa"];
```

- Associative arrays (hash tables)

```
var my_hash = {a:2, b:3, c:"text"};
```

Open

EVERYTHING IS OBJECT

- Every variable can be considered as object
- For example strings and arrays have member functions:

```
// file: objects.html
var test = "some string";
alert(test[7]);                // shows letter 'r'
alert(test.charAt(5));         // shows letter 's'
alert("test".charAt(1));       // shows letter 'e'
alert("test".substring(1,3)); // shows 'es'
var arr = [1,3,4];
alert (arr.length); // shows 3
arr.push(7);         // appends 7 to end of array
alert (arr[3]);      // shows 7
```

Open

STRING OPERATIONS

The + operator joins strings (slow)

```
string1 = "lol ";
string2 = "cats";
alert(string1 + string2);  // lol cats
```

What is "11" + 11?

```
alert("11" + 11);  // 1111
```

Converting string to number:

```
alert(parseInt("11") + 11);       // 22
alert(parseInt("011",8) + 11);      // 20 ???
alert(parseInt("011", 10) + 11);  // 22
```

Open

ARRAYS OPERATIONS AND PROPERTIES

Declaring new empty array:

```
var arr1 = new Array(); var arr2 = [];
```

Declaring an array holding few elements:

```
var arr = [1, 2, 3, 4, 5];
```

Appending an element / getting the last element:

```
arr.push(3); var element = arr.pop();
```

Reading the number of elements (array length):

```
arr.length;
```

Finding element's index in the array:

```
arr.indexOf(1);
```

Open

CONDITIONAL STATEMENT (IF)

```
unitPrice = 1.30;
if (quantity > 100) {
  unitPrice = 1.20;
}

/*
Greater than: >
Less than: <
Greater than or equal to: >=
Less than or equal to: <=
Equal: ==
Not equal: !=
*/
```

Open

CONDITIONAL STATEMENT (IF)

The condition may be of Boolean or integer type:

```
var a = 0;
var b = true;
if (typeof(a)=="undefined" || typeof(b)=="undefined") {
  document.write("Variable a or b is undefined.");
}
else if (!a && b) {
  document.write("a==0; b==true;");
} else {
  document.write("a==" + a + "; b==" + b + ";");
}
```

Open

SWITCH STATEMENT

The switch statement works like in C#:

```
switch (variable) {
  case 1:
    // do something
    break;
  case 'a':
    // do something else
    break;
  case 3.14:
    // another code
    break;
  default:
    // something completely different
}
```

Open

LOOPS

Like in C# (for, while, do while)

```
/*
for loop
while loop
do ... while loop
*/
var counter;
for (counter=0; counter<4; counter++) {
  alert(counter);
}
for (var key in haystack);
while (counter < 5) {
  alert(++counter);
}
```

Open

SYNTAXE

- Les fonctions:
  - La syntaxe de définition d'une fonction est:

```
function nomFonction(arg1,arg2,...)
```

  - Contrairement au langage C, on ne donne pas le type des arguments ni celui de la valeur de retour éventuelle

Open

SYNTAXE

- Les chaînes de caractères:
  - Une chaîne de caractères est délimitée par des guillemets simples ou doubles.
  - Exemple:

```
texte = "un beau texte"
```

Ou

```
texte = 'un beau texte'
```

Open

SYNTAXE

- Les chaînes de caractères:
  - Pour manipuler des chaînes de caractères, il existe de nombreuses fonctions.
  - Contrairement au langage C, il est possible de comparer deux chaînes de caractères à l'aide de l'opérateur ==.

Open

FUNCTIONS

- Code structure – splitting code into parts
- Data comes in, processed, result returned

```
function average(a, b, c) {
  var total;
  total = a+b+c;
  return total/3;
}
```

- Anonymous functions

```
var average = function (a, b, c) { ... }
var res1 = average(1,2,3);
var myAvg = average;
var res2 = myAvg(1,2,3);
```

Open

FUNCTION ARGUMENTS AND RETURN VALUE

- Functions are not required to return a value
- When calling function it is not obligatory to specify all of its arguments
- The function has access to all the arguments passed via arguments array

```
function sum() {
  var sum = 0;
  for (var i = 0; i < arguments.length; i ++)
    sum += parseInt(arguments[i]);
  return sum;
}
alert(sum(1, 2, 4));
```

Open

JAVASCRIPT

- Définition
- Syntaxe
- **Les objets**
- Les évènements
- Les boîtes de dialogue
- Accès à un élément quelconque d'une page

Open

LES OBJETS

- L'utilisation principale de JavaScript est la manipulation des objets d'une page et plus particulièrement des objets des formulaires
- Il est possible d'interagir à deux niveaux:
  - Au niveau du navigateur internet
  - Au niveau de la page affichée dans le navigateur

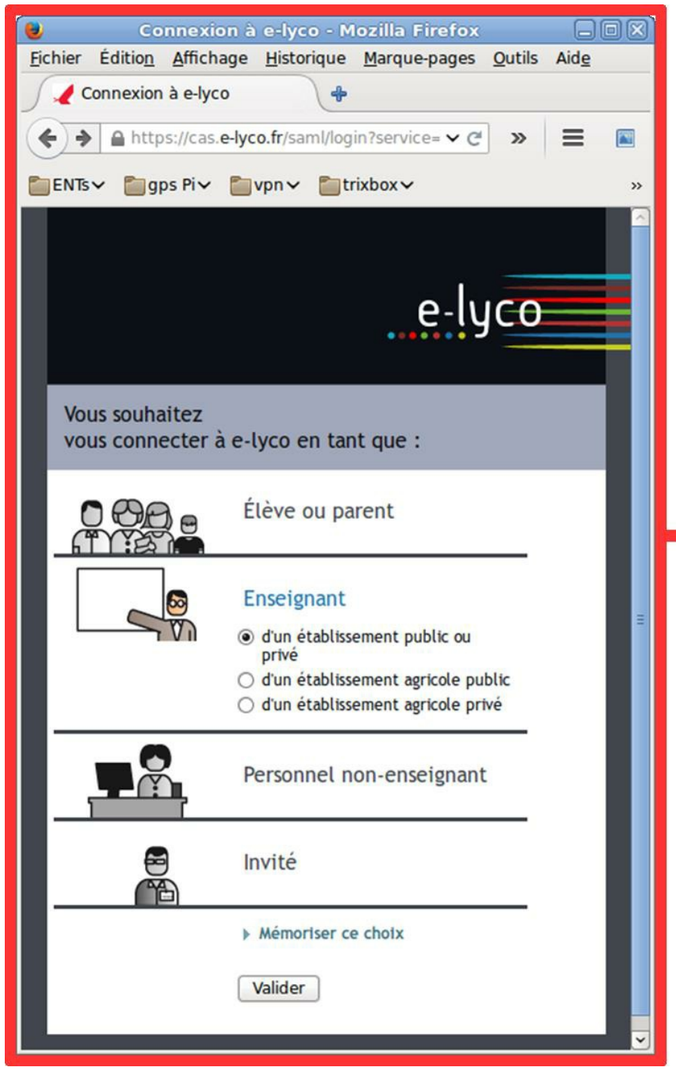
Open

LES OBJETS

- Une page est composée de façon hiérarchique.

Open

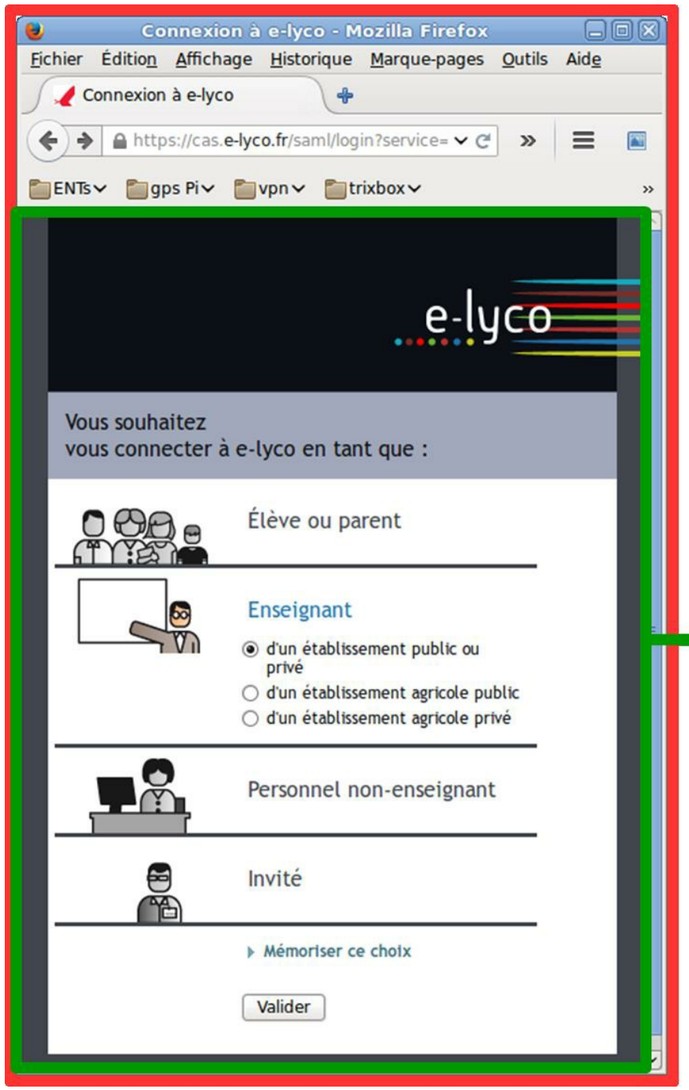
LES OBJETS



Objet fenêtre

Open

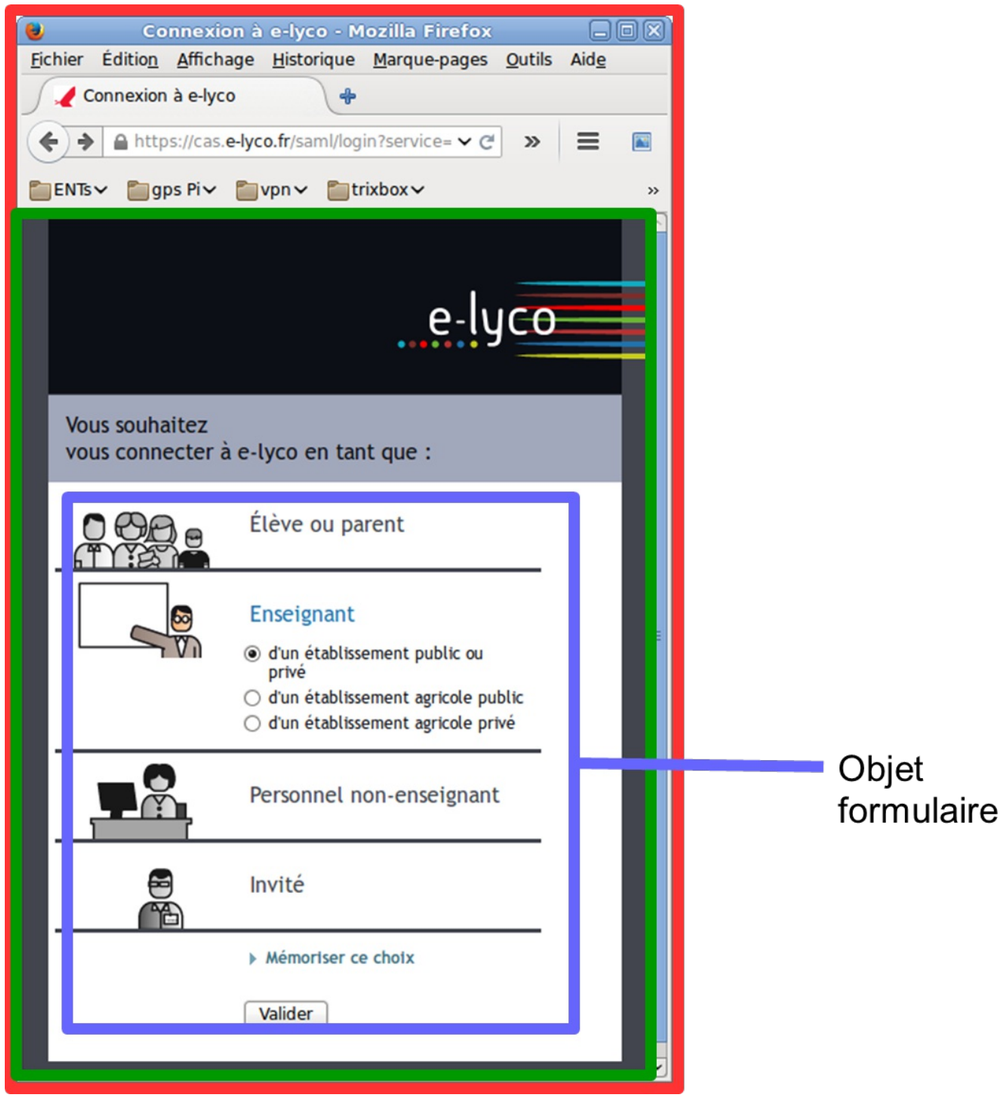
LES OBJETS



Objet document

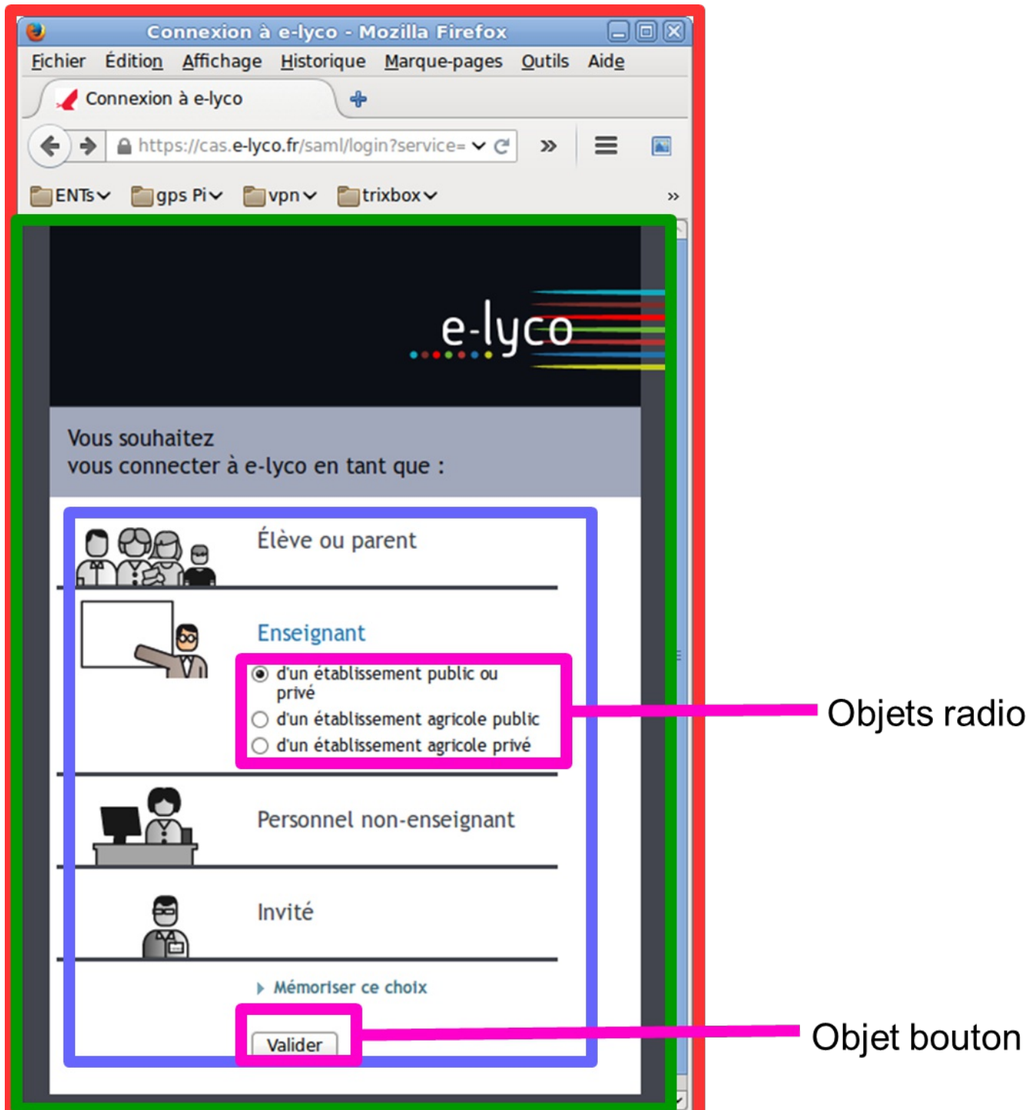
Open

LES OBJETS



Open

LES OBJETS



Open

LES OBJETS

- L'accès se fait de façon hiérarchique.

```
window.document.forms["nomDuformulaire"].nomDeLélément
```

- Pour chaque niveau il existe des méthodes et des attributs. Exemple:

```
<html><body>
    <form name="monForm">
        <label for="login">Votre login :</label>
        <input type="text" name="login" id="login" />
    </form>
</body></html>
```

- Si je souhaite avoir la valeur contenue dans le champ login du formulaire, j'utiliserai le code JavaScript suivant:

```
leLogin=window.document.forms["monForm"].login.value;
```

Open

LES OBJETS

- Classes are functions
- Objects are associative arrays
- **this** refers to the owning code
- **new** copies the prototype into new reference

```
var person = new Object();
person.first = "Tim";
person.last = "Scarfe";

person.sayHello = function() {
   console.log("Hello, " + this.first + " " + this.last);
};
person.sayHello();
```

Open

LES OBJETS

```
function User(first, last) {
    this.first = first;
    this.last = last;

    this.sayHello = function() {
        console.log("Hello, " + this.first
                + " " + this.last);
    }
}

var svi = new User("Svi", "Ivanov");
var alex = new User("Alex", "Ivanova");
svi.sayHello();
alex.sayHello();
```

Open

JAVASCRIPT

- Définition
- Syntaxe
- Les objets
- **Les évènements**
- Les boîtes de dialogue
- Accès à un élément quelconque d'une page

Open

LES ÉVÈNEMENTS

- L'action sur un élément de la page se fait lors d'un événement particulier (clique de souris, changement de la valeur d'un champ, etc)
- Voici quelques évènements possibles pour une page WEB
  - Click (onClick)
  - Load (onLoad)
  - Unload (onUnload)
  - MouseOver (onMouseOver)
  - MouseOut (onMouseOut)
  - Focus (onFocus)
  - Change (onChange)
  - Submit (onSubmit)

Open

LES ÉVÈNEMENTS

- Pour qu'un objet réagisse à un événement, il faut lui associer une fonction de traitement.
- Exemple:

```
<html><body>
<script src="mesFonctions.js" type="text/javascript"></script>
 <form name="monForm">
    <label for="login">Votre login :</label>
    <input type="text" name="login" id="login"
    onchange="afficheLogin();">
    <input type="text" name="loginBis" id="loginBis">
 </form>
</body></html>
```

- Le fichier mesFonctions.js contient le code suivant:

```
function afficheLogin(){
window.document.forms["monForm"].loginBis.value =
  window.document.forms["monForm"].login.value;
}
```

Open

JAVASCRIPT

- Définition
- Syntaxe
- Les objets
- Les évènements
- **Les boîtes de dialogue**
- Accès à un élément quelconque d'une page

Open

LES BOÎTES DE DIALOGUE

- Il existe 3 types de boîtes de dialogue:
  - alert : affiche un message et un bouton ok

  - confirm : affiche un message et un bouton ok et annuler

Open

LES BOÎTES DE DIALOGUE

- prompt : affiche une zone de saisie et un bouton ok.

- confirm retournera true ou false selon la réponse (ok ou annuler).
- prompt retournera le message contenu dans la zone de saisie.

Open

JAVASCRIPT

- Définition
- Syntaxe
- Les objets
- Les évènements
- Les boîtes de dialogue
- **Accès à un élément quelconque d'une page**

Open

ACCÈS À UN ÉLÉMENT QUELCONQUE D'UNE PAGE

- La manière la plus simple pour avoir accès à un élément quelconque d'une page est d'utiliser son identifiant

```
var obj = document.getElementById("idelement") ;
```

- Il est ainsi possible d'avoir des informations et d'agir sur l'élément
  - **innerHTML** : va recupérer/modifier le contenu HTML de l'élément
  - **textContent** : va recupérer/modifier le contenu de l'élément
  - **nodeName** : va recupérer le nom de l'élément

Open

AN EXAMPLE

```
<div id="div1"></div>
<input type="button" value="Vert" onclick="vert();"/>
<input type="button" value="Rouge" onclick="rouge();"/>
```

Open

AN EXAMPLE

```
#div1{
  background-color:#ff0000;
  width:200px;
  height:200px;
}
```

Open

AN EXAMPLE

```
function vert(){
   document.getElementById('div1').style.backgroundColor='#00ff00';
}
function rouge(){
   document.getElementById('div1').style.backgroundColor='#ff0000';
}
```

Open

TOOLS FOR JAVASCRIPT DEVELOPMENT

UNLEASH THE POWER OF JAVASCRIPT TOOLING

Open

TABLE OF CONTENTS

Open

JAVASCRIPT EDITORS

Open

JETBRAINS WEBSTORM

- JavaScript IDE suitable for both client-side JavaScript and server (Node.js) development
  - Livereload
  - Build
  - Code highlighting/Intellisense for:
    - LESS/SASS/Stylus CSS preprocessors
    - CoffeeScript/TypeScript JavaScript preprocessors
    - AngularJS directives
    - Emmet Coding
  - Has only 30-days-long free trial

Open

SUBLIME TEXT 2/3

- Editor for scripting languages
  - JavaScript, PHP, Python, Ruby, etc…
  - Basic code highlighting with vanilla installation
    - Endless number of configurable plugins
  - Free product
    - Paid only if used commercially
  - With plugins support for:
    - LESS/SASS/Stylus
    - CoffeeScript/TypeScript

Open

ATOM: A HACKABLE TEXT EDITOR

- Cross-platform editing
- Built-in package manager
- Smart autocompletion
- File system browser
- Under the hood
  - Atom is a desktop application built with HTML, JavaScript, CSS, and Node.js integration. It runs on Electron, a framework for building cross platform apps using web technologies.
- Open source
  - support by Github

Open

VISUAL CODE EDITOR

- Smart autocompletion with IntelliSense.
  - smart completions based on variable types, function definitions, and imported modules.
- Debugging support
- Git commands built-in
- Extensible and customizable
- great integration of TypeScript and Angular 2
- Open source
  - support by Microsoft

Open

PROJECT TOOLS

NO MATTER THE EDITOR

Open

PROJECT TOOLS

- NPM, Yarn & Bower
  - Install Node.js packages or client libraries
- Grunt & Gulp
  - Tasks runner
  - Create different tasks for build/development/test cases
- Yeoman
  - Scaffolding of applications
  - One-line-of-code to create a project template with views/routes/modules/etc…

Open

PACKAGE MANAGEMENT

NPM, YARN & BOWER

Open

PACKAGE MANAGEMENT: NPM

- Node.js Package Management (NPM)
- Package manager for Node.js modules

```
npm init #in CMD (Win) or Terminal (MAC/Linux)
```

- Initializes an empty Node.js project with package.json file

```
 npm init
//enter package details
name: "NPM demos"
version: 0.0.1
description: "Demos for the NPM package management"
entry point: main.js
test command: test
git repository: http://github.com/user/repository-name
keywords: npm, package management
author: doncho.minkov@telerik.com
license: BSD-2-Clause
```

Open

PACKAGE MANAGEMENT: NPM

- Installing modules

```
npm install package-name [--save][--save-dev][--save-optional]
# Installs a package to the Node.js project
```

-S, *--save*: Package will appear in your dependencies
in package.json
-D, *--save-dev*: Package will appear in
your devDependencies
-O, *--save-optional*: Package will appear in
your optionalDependencies.

```
npm install express --save-dev
```

Before running the project

```
npm install ## Installs all missing packages from package.json
```

Open

PACKAGE MANAGEMENT: BOWER

- Bower is a package management tool for installing client-side JavaScript libraries
  - Like jQuery, KendoUI, AngularJS, etc…
  - It is a Node.js package and should be installed first

```
npm install –g bower
bower init # in CMD (Win) or Terminal (Mac/Linux)
```

- Asks for pretty much the same details as *$ npm init*
- Creates bower.json file to manage libraries

Open

PACKAGE MANAGEMENT: BOWER

- Searching for libraries

```
bower search kendo
```

```
Search results:

    kendo-ui git://github.co
    angular-kendo-ui git://g
    kendo-backbone git://git
    knockout-kendo git://git
    kendo-global git://githu
```

- Installing libraries

```
bower install kendo-ui
```

```
bower cached         git://gith
bower validate       2014.1.318
i.git#*
bower cached         git://gith
bower validate       2.0.3 agai
.0.3
bower resolution     Removed un
bower install        kendo-ui#2
bower install        jquery#2.0

kendo-ui#2014.1.318 bower_comp
└── jquery#2.0.3
```

Open

GRUNT & GULP

TASKS RUNNER

Open

GRUNT

- Grunt is a Node.js task runner
  - It can runs different tasks, based on configuration
  - Tasks can be:
    - Concat and minify JavaScript/CSS files
    - Compile SASS/LESS/Stylus
    - Run jshint, csshint
    - Run Unit Tests
    - Deploy to Git, Cloud, etc…
    - And many many more

Open

GRUNT

- Why use a task runner?
  - Task runners gives us automation, even for different profiles:

DEVELOPMENT	TEST	BUILD
jshint	jshint	jshint
stylus	stylus	stylus
csshint	csshint	csshint
connect	mocha	concat
watch		uglify
		copy

Open

CONFIGURING GRUNT

- To configure grunt, create a Gruntfile.js file in the root directory of your application
  - It is plain-old Node.js
  - Grunt is configured programmatically
  - Create an module that exports a single function with one parameter – the grunt object

```
module.exports = function (grunt) {
//configure grunt
};
```

Open

CONFIGURING GRUNT (2)

- All the configuration is done inside the module
- First execute the *grunt.initConfig()* method and pass it the configuration

```
module.exports = function (grunt) {
grunt.initConfig({
  ...
});
};
```

Open

CONFIGURING GRUNT PLUGINS

- To use a plugin in grunt:
  - Install the plugin

```
npm install grunt-contrib-jshint --save-dev
```

- Load the plugin

```
//inside the grunt module
grunt.loadNpmTasks('grunt-contrib-jshint');
```

- Configure the plugin

```
//inside the grunt.initConfig()
grunt.initConfig({
  jshint: {
    app: ['Gruntfile.js',
      'path/to/scripts/**/*.js']
  }
});
```

Open

GRUNT PLUGINS

Open

GRUNT PLUGINS: BUILD

- *jshint (grunt-contrib-jshint)*
  - Runs jshint for specified files
- *csslint(grunt-contrib-csshint)*
  - Runs csslint for specified files
- *stylus (grunt-contrib-stylus)*
  - Compiles STYL files into CSS files
- *uglify (grunt-contrib-uglify)*
  - Minifies configured JavaScript files
- *concat (grunt-contrib-concat)*
  - Concats configured JavaScript files

Open

GRUNT PLUGINS: DEVELOPMENT

- *connect (grunt-contrib-connect)*
  - Stars a Web server on a given port and host
- *watch (grunt-contrib-watch)*
  - Watches for changes to configured files
  - Can run other tasks on file changed

Open

GULP: THE STREAMING BUILD SYSTEM

Streams come to us from the earliest days of unix and have proven themselves over the decades as a dependable way to compose large systems out of small components that do one thing well.

You can then plug the output of one stream to the input of another and use libraries that operate abstractly on streams to institute higher-level flow control.

Open

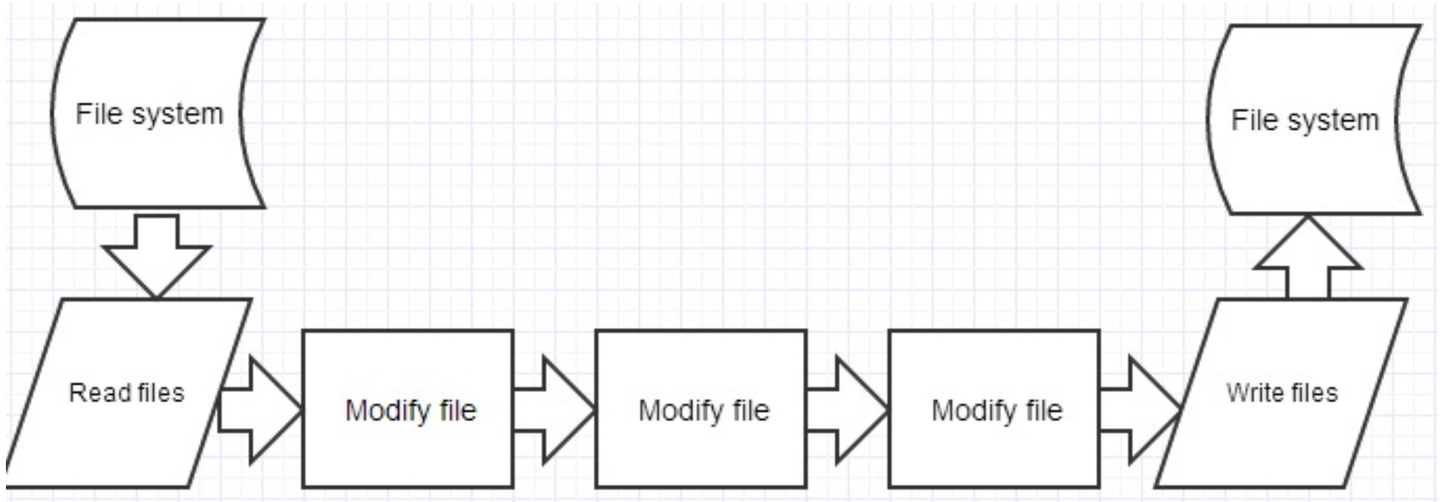
WHY STREAMS?

PICTURE A BUILD SYSTEM IN YOUR HEAD.

- It should take in files, modify them, and output the new ones

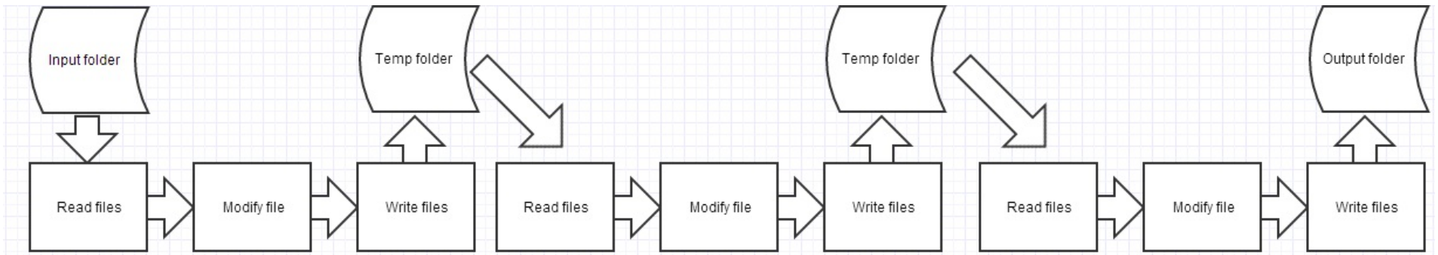
Open

YOU PICTURED THIS



Open

YOU DIDN'T PICTURED THIS



Open

WHAT IS WRONG WITH GRUNT

- Plugins do multiple things
  - Want a banner? Use the javascript minifier
  - Plugins do things that don't need to be plugins
  - Need to run your tests? Use a plugin
- Grunt config format is a mess that tries to do everything
- Not idiomatic with "the node way"
- Headache of temp files/folders due to bad flow control

Your build system should empower not impede

It should only manipulate files - let other libraries handle the rest.

Open

SAMPLE GRUNTFILE

```
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
```

1. Runs tests
2. Lints code
3. Concats javascript
4. Minifies it
5. Runs again if files are changed

Open

WHAT'S THE DIFFERENCE?

- With Gulp your build file is code, not config
- You use standard libraries to do things
- Plugins are simple and do one thing - most are a ~20 line function
- Tasks are executed with maximum concurrency
- I/O works the way you picture it

Open

Gulp does nothing but provide some streams and a basic task system Gulp has only 5 functions you need to learn

Open

```
gulp.task(name, fn)
```

- It registers the function with a name.
- You can optionally specify some dependencies if other tasks need to run first.

Open

```
gulp.run(tasks...)
```

- Runs all tasks with maximum concurrency

Open

```
gulp.watch(glob, fn)
```

- Runs a function when a file that matches the glob changes
- Included in core for simplicity

Open

```
gulp.src(glob)
```

- This returns a readable stream.
- Takes a file system glob (like grunt) and starts emitting files that match.
- This is piped to other streams

Open

```
gulp.dest(folder)
```

- This returns a writable stream
- File objects piped to this are saved to the file system

Open

YEOMAN

APPLICATION SCAFFOLDING

Open

YEOMAN

- Yeoman is a Node.js package for application scaffolding
  - Uses bower & NPM to install the js package
  - Has lots of generators for many types of applications:
    - MEAN, AngularJS, Kendo-UI, WebApp, WordPress, Backbone, Express, etc…
    - Each generators install both needed Node.js packages and client-side JavaScript libraries
    - Generated Gruntfile.js for build/test/serve

Open

YEOMAN

Install Yeoman:

```
npm install –g yo
```

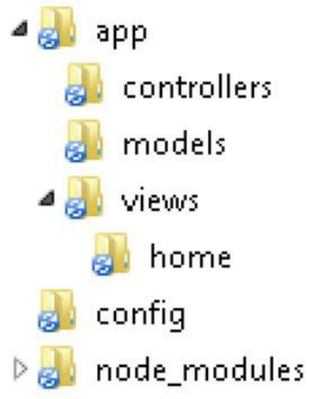
Install Yeoman generator:

```
npm install –g generator-jhipster
```

Scaffold Express application:

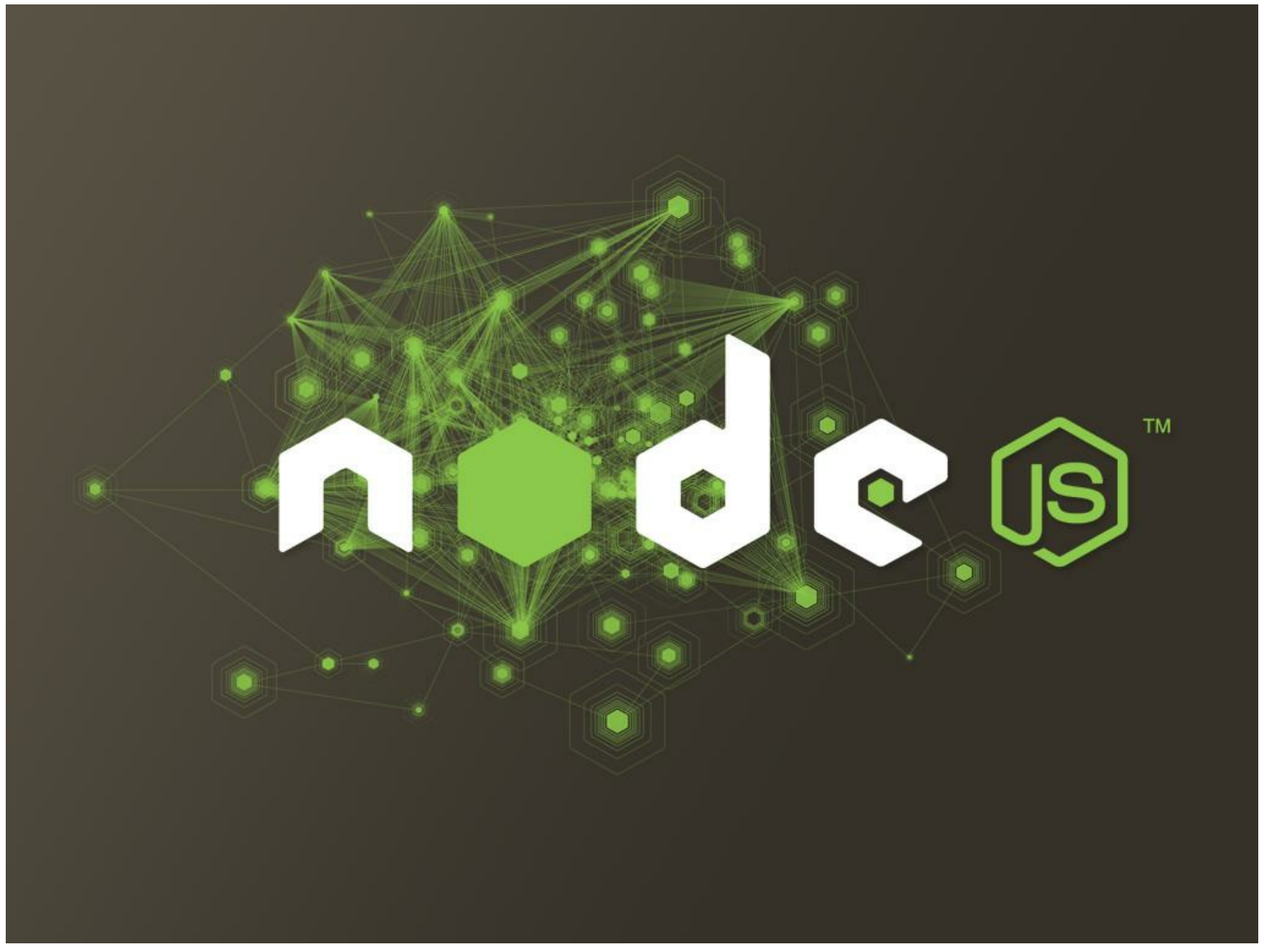
```
cd path/to/app/directory
yo jhipster
```

Generates:



Open

PRÉSENTATION DE NODEJS



Open

PRÉSENTATION NODE.JS (1)

- Node.js, plateforme événementielle aux entrées-sorties non-bloquantes (asynchrone) pour développer des applications en JavaScript "côté serveur"
- Moteur léger et écologique !
  - (peu gourmand en ressources, CPU, RAM)
- Noyau en JavaScript et C++ par-dessus le moteur JavaScript V8 de Google Chrome
  - (moteur V8 régulièrement mis à jour dans Node.js)
- Logiciel libre à licence permissive (licence MIT) avec la quasi totalité des modules également à licence permissive (licence MIT ou BSD)

Open

PRÉSENTATION NODE.JS (2)

- Les méthodes de l'API sont asynchrones par défaut
- Le streaming au coeur de l'API
- Manipule nativement le JSON
- Sait gérer nativement toutes les requêtes HTTP
  - (pour réaliser l'équivalent d'appels AJAX côté serveur)
- Développement aisé d'applications (grande agilité)
- Les applications réalisées sont extrêmement rapides
- Mettre frontal Nginx (basé événements) à la place de Apache (basé processus), sinon performances bridées

Open

UTILISATIONS DE NODE.JS

- Node.js excellent pour :
  - API web (REST, services JSON)
  - Sites web (notamment streaming de bout en bout : par exemple streaming de fichiers depuis base de données puis streaming de ces fichiers dans archive ZIP puis streaming HTTP vers navigateur web)
  - Très nombreuses connexions et montée en charge (gère des milliers de connexions simultanées à faible charge sur un seul processus)
  - Communications temps réel (sockets, polling, etc.)

Open

LIMITATIONS

- Node.js inadapté pour :
  - Usage intensif du CPU : 3D, transcodage vidéo, et en général tout calcul mathématique non fourni par l'API du coeur (les fonctions cryptographiques font, elles, partie du coeur et sont rapides, en C++)
  - Mise en mémoire de grande quantité de données (pas plus de 1Go augmentable à 1,7Go max). Mais de par programmation événementielle cette limite est moins gênante qu'elle le serait avec d'autres langages comme Java par exemple (qui consomme une très grande quantité de mémoire).

Open

PHILOSOPHIE DE NODE.JS

- http://substack.net/node_aesthetic
- La philosophie de Node.js :
  - Bibliothèque centrale minimale
  - Programmabilité et composabilité maximale
  - Réutilisabilité radicale

Agilité maximale

Innovation rapide car décentralisée

Open

CALLBACK NODE.JS NORMALISÉ

- Un callback normalisé doit être passé comme argument à toutes les méthodes asynchrones

```
function(err, res) {
  if (err) {
    console.error("Failed:", err);
    return;
  }
  console.info("Success with produced result:", res);
}
```

Mais on peut utiliser des promesses en « promisifiant » les méthodes de Node.js (cf. infra)

Open

RECHERCHE DE MODULES

- https://www.npmjs.com/
- https://github.com/
  - Comparer la popularité grâce :
    - aux étoiles (>10 → c'est bien)
    - au nombre des contributeurs (>3 → c'est bien)

Open

MODULES SERVEUR UTILES (1)

- **Tracer** : logs serveur
- **Commander** : Gestion de la ligne de commande
- **Node-convict** : lecture / enregistrement de configuration
- **Express** : cadriciel web/REST (web framework)
- **Koa** : cadriciel web/REST (web framework) utilisant les promesses et les générateurs

Open

MODULES SERVEUR UTILES (2)

- **Bookshelf.js** : ORM fonctionnant avec PostgreSQL, MySQL, SQLite3, Oracle
- **Ldapjs** : Client et serveur LDAP
- **Nodemailer** : Envoi de courriels
- **Archiver** : Génération d'archives ZIP, TAR en streaming
- **Fast-csv** : création / lecture de CSV

Open

MODULES CLIENT ET SERVEUR UTILES

- **Create-error** : création nouveaux types d'erreur
- **Bluebird** : Module pour promesses, ultra-rapide, avec plein d'utilitaires
- **Moment** : affichage, manipulations dates et durées
- **Validator.js** : Validation et nettoyage de chaînes de caractères
- **Nunjucks** : template engine proposant layouts et streaming (développé par Mozilla)
- **Base64url** : encode, decode, escape, unescape

Open

MODULES CLIENT UTILES

- **Loglevel** : logs en console débraillables à l'exécution et compatible avec tout navigateur web (idéal pour assurer du débogage et du support)
- **Dexie** : API IndexedDB avec des promesses, permettant notamment la recherche plein texte

Open

BONNES PRATIQUES AVEC MODULES NPM (1)

- Trouvez les modules pouvant être mis à jour

```
npm install –prefix ~/ -g npm-check-updates
npm-check-updates
```

- Mettez à jour tous les modules (+ validez par tests) :

```
npm-check-updates -u
```

- Utilisez des numéros de version strictes de module (pas de ^ ou ~)

```
npm install loglevel --save --save-exact
npm install jshint --save-dev --save-exact
```

Open

BONNES PRATIQUES AVEC MODULES NPM (2)

- Factorisez les dépendances communes :

```
npm dedupe
```

- Définissez strictement l'arbre de dépendances :

```
npm shrinkwrap
git add npm-shrinkwrap.json
```

Open

BONNES PRATIQUES AVEC MODULES NPM (3)

- Vérifiez les licences de tous les modules utilisés (y compris les licences de toutes les dépendances)

```
npm install –prefix ~/ -g licence-checker
license-checker
license-checker | grep licences | sort | uniq
```

On trouve essentiellement des licences :

```
- MIT
- BSD
- Apache
```

Open

FACTORISER LE CODE AVEC MODULES NPM (1)

- Avec NPM on peut utiliser (installer) un module présent dans le registre NPM mais pas que …
- Avec NPM on peut aussi utiliser (installer) un module présent sur un dépôt Git public, comme GitHub, ou privé

Open

FACTORISER LE CODE AVEC MODULES NPM (2)

- Forme des URL de module sur dépôt Git :

```
git://github.com/user/project.git#commit-ish
git+ssh://user@hostname:project.git#commit-ish
git+ssh://user@hostname/project.git#commit-ish
git+http://user@hostname/project/blah.git#commit-ish
git+https://user@hostname/project/blah.git#commit-ish
```

- Exemple pour utiliser (installer) un module privé :

```
npm install git+ssh://git@git.entreprise.fr:module1.git --save-dev –save-exact
```

- Suppression du besoin d'utilisation de git-submodules dans beaucoup de cas

Open

MODULES COMMONJS

- NPM et Node.js utilisent le standard CommonJS pour gérer les modules. La spec modules ES6 en est inspirée. Exemple de module fourni (repository.js) et de module importé (behavior.js) :

```
// repository.js
exports.getDocsCount = getDocsCount;
function getDocsCount() {
    return db.documents.count();
}
// behavior.js
var repository = require('./repository');
repository.getDocsCount();
```

Open

MODULES COMMONJS POUR CODE CLIENT (NAVIGATEUR)

- Grâce à Browserify on peut utiliser les modules CommonJS avec tous les avantages que cela représente au niveau du code JavaScript déployé sur le client (navigateur)
- On peut ainsi disposer d'environ 80% des 120.000 modules NPM disponibles à ce jour côté client.
- Les développeurs peuvent utiliser les mêmes modules qu'ils connaissent côté client que côté serveur (et inversement :-))
- Pour mettre en pratique lire absolument : https://github.com/substack/browserify-handbook

Open

AUTRES GESTIONNAIRES DE MODULES (BOWER, YEOMAN)

- Il y a actuellement plusieurs gestionnaires de modules JavaScript :
  - NPM
  - Bower
  - Yarn
- Bower et Yeoman sont moins puissants que NPM et sont généralement utilisés par les développeurs connaissant peu (la puissance de) NPM et Browserify
- NPM a vocation à devenir l'unique gestionnaire de modules JavaScript. Mais Yarn est très populaire

Open

OUTILS DE CONSTRUCTION D'APPLICATIONS

- **Grunt** : Outil très célèbre, basé sur la configuration, pour construire des applications web.
  - Facile à comprendre et à utiliser.
  - **Désavantage** : ne tire pas partie du streaming, fichiers lus et modifiés sur disque à chaque tâche.
- **Gulp** : Outil de construction basé sur le code, plutôt que la configuration, et utilisant le streaming
- **NPM** : NPM seul peut aussi servir à construire des applications (minification, etc.)

Open

PROMESSES

- On peut gérer le code asynchrone avec des callbacks. C'est la pratique par défaut dans Node.js
- Mais peut devenir très pénible quand plusieurs actions asynchrones s’enchaînent ou se déroulent en parallèle
  - La meilleure solution → les promesses (aussi appelées engagements ou futures)
  - En anglais on parle de promises ou de thenables

Open

PROMESSES

Exemple de code asynchrone classique :

```
fs.readFile('file.json', function(err, val) {
    if (err) {
        console.error("Unable to read file");
    } else {
        try {
            val = JSON.parse(val);
            console.log(val.success);
        } catch( e ) {
            console.error("Invalid JSON in file");
        }
    }
});
```

Open

PROMESSES

```
var P = require('bluebird');
var fs = P.promisifyAll(require('fs'));

fs.readFileAsync('file.json').then(JSON.parse)
.then(function(val) {
    console.log(val.success);
})
.catch(SyntaxError, function(err) {
    console.error("Invalid JSON in file", err);
})
.catch(function(err) {
    console.error("Unable to read file", err) ;
});
```

Open

VEILLE SÉCURITÉ ET CORRECTIFS

- Audit permanent de TOUS les modules JavaScript sur NPM et émissions d'alertes :
  - http://nodesecurity.io/
- Nouvelles versions de Node.js avec correctifs de sécurité :
  - http://nodejs.org/

Open

NSP

```
npm install --prefix ~/ -g nsp
nsp audit-package
   Name     Installed  Patched  Vulnerable Dependency
   connect    2.7.5    >=2.8.1  nodesecurity-jobs > kue > express
nsp audit-shrinkwrap
  Name     Installed  Patched  Vulnerable Dependency
  connect    2.7.5    >=2.8.1  nodesecurity-jobs > kue > express
```

Open

MON PREMIER PROGRAMME EN NODE.JS

```
// bonjour.js
console.log('Bonjour');
```

```
$ node bonjour.js
```

Open

TYPESCRIPT

Open

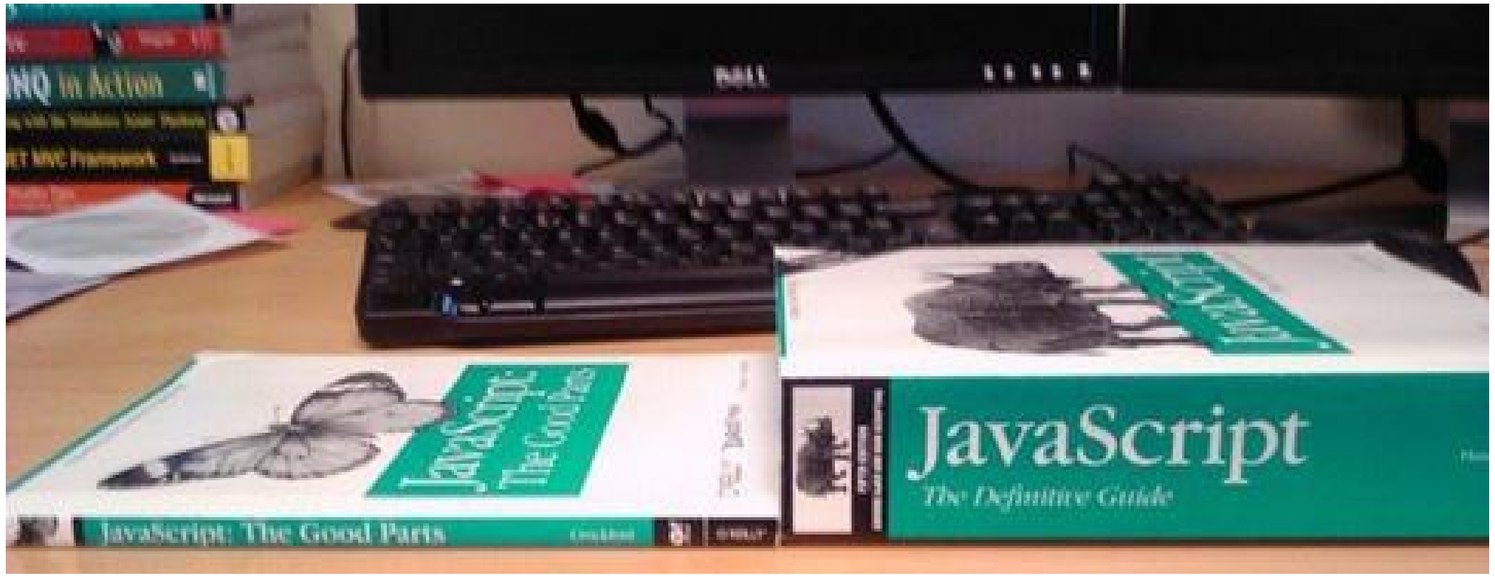
WHAT IS TYPESCRIPT?

- Free and open source, strongly supported by Microsoft
- Based on ecmascript 4 + ecmascript 6
- Created by the father of C# Anders Hejlsberg
- A superset of JavaScript
- To answer why we need JavaScript+, we need to understand what's wrong with vanilla JavaScript

Open

WHAT IS THE PROBLEM ?

- Why do people hate working in JavaScript?



Using state of the art software engineering practices ;)

Open

WHAT IS THE PROBLEM ?

- JS is designed for small things
- We now use to do big things
- But JavaScript is not suited for building large applications

> *Your JavaScript code gets complex; it becomes extremely unwieldy*

Open

LET'S LOOK AT TYPESCRIPT

- To get started with TypeScript, grab it from http://typescriptlang.org
- Let's look at TypeScript, first the core concept…

Open

TYPESCRIPT - FIRST GLANCE - OPTIONAL STRONG TYPE CHECKING

```
// jsfunction
f(x, y) {    return x * y;}
// tsfunction
f(x : number, y : number) : number {     return x * y;}
// Type information is enforced in design and
// compile time, but removed at runtime
```

Open

TYPESCRIPT FEATURES

- Static Type Checking
- Modules and Export
- Interface and Class for traditional Object Oriented Programming
- Works with all your existing JavaScript libraries
- Low learning overhead compared to similar JavaScript systems (CoffeeScript or Dart)
- Amazing Visual Studio, visual code studio, eclipse or IntelliJ tooling
- Outstanding team and refactoring scenarios

Open

SUMMARY - WHY TYPESCRIPT ? (EXPECTED BENEFITS)(1)

- Have to learn one more thing - there is a learning curve, very easy if you already know JavaScript, or if you know C# or Java very well.
- You still have to learn JavaScript - Understanding how TypeScript converts to JavaScript will teach you better JavaScript
- Some definition files don't exist or incomplete, but I think this will vanish very quickly. These aren't hard to write if you really need something.

Open

SUMMARY - WHY TYPESCRIPT ? (EXPECTED BENEFITS)(2)

- Modules and classes enable large projects and code reuse
- Compile-time checking prevents errors
- Definition files for common JavaScript libraries makes them very easy to work with, and provides strong type checking
- Source map debugging makes debug easy

Open

INITIAL CONCLUSION - IF I HAVE TO MAKE A DECISION FOR YOU…

- If you see yourself using more JavaScript. You have to look at TypeScript.
- If you and your team has to work on JavaScript together, you have to look at TypeScript.
- Once you've done the initial hard work and converted a project. You can't stand going back.

Open

INTRODUCTION TO

Open

EARLY WEB

- Web designed for documents
- Server creates pages / browser displays
- Data input sent to and processed by the server
- Updated pages created on the server and resent

Open

FIRST EXAMPLE - PHP

```
<!doctype html>
<html>
<head>
</head>
<body>
  <form method="post" action="hello.php">
    <label>Name:</label>
    <input type="text" id="yourName”>
    <input type="submit" value="Say Hello" />
    <hr>
<?php
    echo "<h1>Hello ".$HTTP_POST_VARS["yourName"]."!</h1>";
?>
  </form>
</body>
</html>
```

Open

WEB EVOLUTION - AJAX

- Interactive client-side web
  - Collect input from user
  - Update display
  - Communicate with server
- Client-side processing enabled by
  - JavaScript
  - DOM manipulation
  - HTTP server messaging

Open

FIRST EXAMPLE - JQUERY

```
<html>
<head>
  <script src="http://code.jquery.com/jquery-1.9.1.min.js"></script>
  <script type="text/javascript">
   $(function() {
     $("#yourName").keyup(function () {
       $("#helloName").text("Hello " + this.value + "!");
     });
   });
 </script>
</head>
<body>
  <div>
    <label>Name:</label>
    <input type="text" id="yourName">
    <hr>
```

Open

JQUERY

- Simplifies event binding and DOM manipulation
- Common API across multiple browsers
- Supports plug-in modules to extend functionality
- Requires writing JavaScript code to wire

Open

TODAY'S WEB – CAN WE DO BETTER?

- Follow good programming practices
  - Separate: data / display / processing
  - Simplify connecting data to display
- Let us focus on the technologies of the web
  - HTML
  - CSS
  - JavaScript

Open

FIRST EXAMPLE - ANGULARJS

```
<html ng-app>
<head>
  <script src="https://code.angularjs.org/1.5.3/angular.min.js"></script>
</head>
<body>
  <div>
    <label>Name:</label>
    <input type="text" ng-model="yourName">
    <hr>
    <h1>Hello {{yourName}}!</h1>
  </div>
</body>
</html>
```

Open

IMPERATIVE VS. DECLARATIVE

```
<input type="text" id="yourName">
<h1 id="helloName"></h1>
<script type="text/javascript">
  $(function() {
    $("#yourName").keyup(function () {
      $("#helloName").text("Hello " + this.value + "!");
    });
  });
</script>
```

to AngularJS declarative relationships

```
<input type="text" ng-model="yourName">
<h1>Hello {{yourName}}!</h1>
```

Open

ABSTRACTIONS

- jQuery abstracts browser functionality
  - e.g. DOM traversal, event binding
- AngularJS abstracts relationships (and more)

> *AngularJS, and all web apps, are built on browser functionalities*

Open

THE DOM ABSTRACTION

- HTML is a declarative document language
- Browser translates HTML into a Document Object Model (DOM)
- DOM is the browser’s in-memory document representation
- JavaScript can manipulate the DOM

Open

ANGULARJS "*COMPILES*" HTML

- Browsers send a document (i.e. DOM) ready event
- AngularJS can intercede and rewrite the DOM
- The rewrite is driven by markup in the DOM

Open

Model-View-Controller (MVC)

- Software architectural pattern
  - Separates display from data
  - Originated with Smalltalk programmers
  - From work at Xerox PARC in the late 70’s
- Models represent knowledge
- Views provide a (visual) representation of attached model data
- Controllers connect to and control views and models

Open

MVC AND VARIATIONS

- Different variations of the pattern
- Model-View-ViewModel (MVVM)
- Model-View-Presenter (MVP)
- Variations differ on…
  - connectivity
  - cardinality
  - directionality

Open

MODEL-VIEW-WHATEVER

" **MVC vs MVVM vs MVP**. What a controversial topic that many developers can spend hours and hours debating and arguing about. For several years AngularJS was closer to **MVC** (or rather one of its client-side variants), but over time and thanks to many refactorings and api improvements, it's now closer to **MVVM – the $scope object could be considered the ViewModel** that is being decorated by a function that we call a Controller.

…

I hereby declare AngularJS to be MVW framework - Model-View-Whatever.

…"

Igor Minar – Google AngularJS Development Team

Open

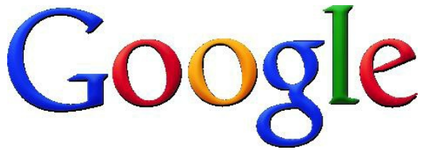
SUMMARY WHAT IS ANGULARJS?

- An MVC framework for **efficiently** creating dynamic views in a web browser (using HTML and JavaScript)
- Some highlights/focuses:
  - Complete application framework
  - From *jQuery replacement* to a massive *enterprise* **Single Page Application (SPA)**
  - Fully dynamic MVVM with POJOs
  - Low level-DOM manipulation/markup invention with directives and templates
  - AJAX / REST API interaction
  - Code organization, dependency injection, testability
  - Comprehensive SPA and routing support

Open

WHY SHOULD WE CARE?

- It's open source
- Actively developed by Google
  - Google is paying devs to actively develop Angular



- Actively developed by open source community (on GitHub)

Open

ANGULAR.JS #1?

- Angular.js appears to be winning the JavaScript framework battle (and for good reason)

- Lets see some evidence…

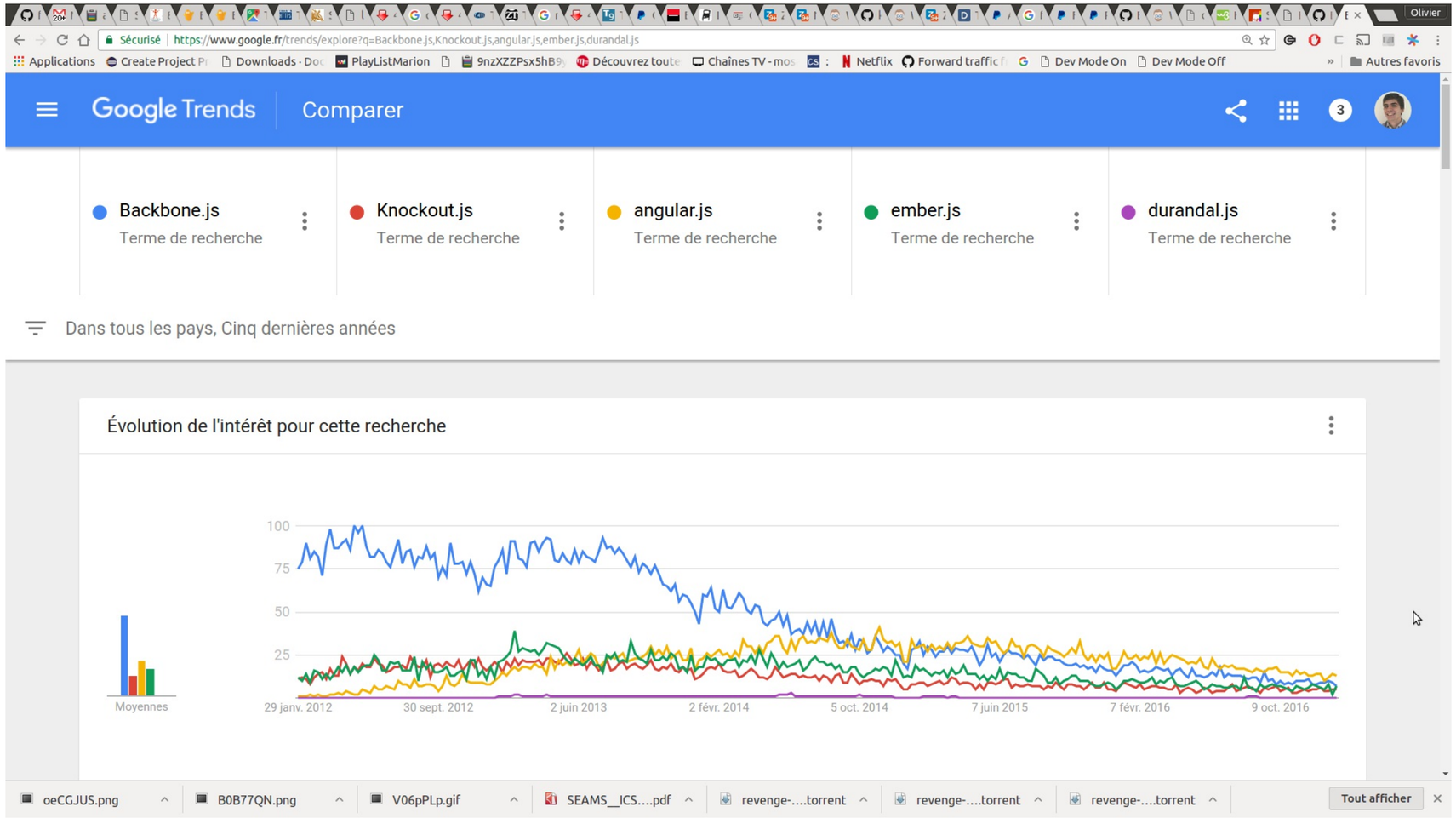
Open

WHY CARE? – GITHUB STATS

	Angular	Ember	Backbone	Knockout	Durandal
Stars	54,449	17,438	25,976	7,977	1,761
Watches	4,477	1,093	1,627	606	192
Forks	27,072	3,646	5,641	1,371	395
commit	8,329	13,969	3,337	1,484	1,189
release	185	232	30	38	6
contributors	1,562	638	292	62	70

Open

WHY CARE? – GOOGLE TRENDS



Open

AngularJS vs jQuery

- jQuery is a library meant for is DOM manipulation, animations and an AJAX wrapper. NOT an application framework
- **Pros**
  - None. Angular has built in ‘basic’ jQuery.
  - If full-blown jQuery is added Angular will automatically use it. Generally full blown NOT needed.
- **Cons**
  - Horrible choice for creating dynamic UIs.
  - Verbose code, hard to maintain, not organized
  - Not MVVM or MVC

Open

ANGULARJS VS BACKBONEJS

- Provides structure to web applications as well as model, view, templating, basic routing and RESTful interactions.
- **Pros**
  - Older and more mature, more people using it
- **Cons**
  - *Previous generation* web app framework
    - No MVVM w/o addons – use jQuery for DOM manip.
    - No DI, not as easily testable
    - Not a full-featured framework, meant to be ‘light’
    - Extremely verbose for what you get

Open

ANGULARJS VS DURANDAL.JS

- Provides an app framework around Knockout
- **Pros**
  - Has a few features not baked into Angular (though readily available 3rd party from Angular community)
- **Cons**
  - Uses Knockout for data binding, suffers from same knockout issues
  - Lacking many Angular.js features
  - No one is using it
  - Is mostly a 1 man show (Rob Eisenberg)

Open

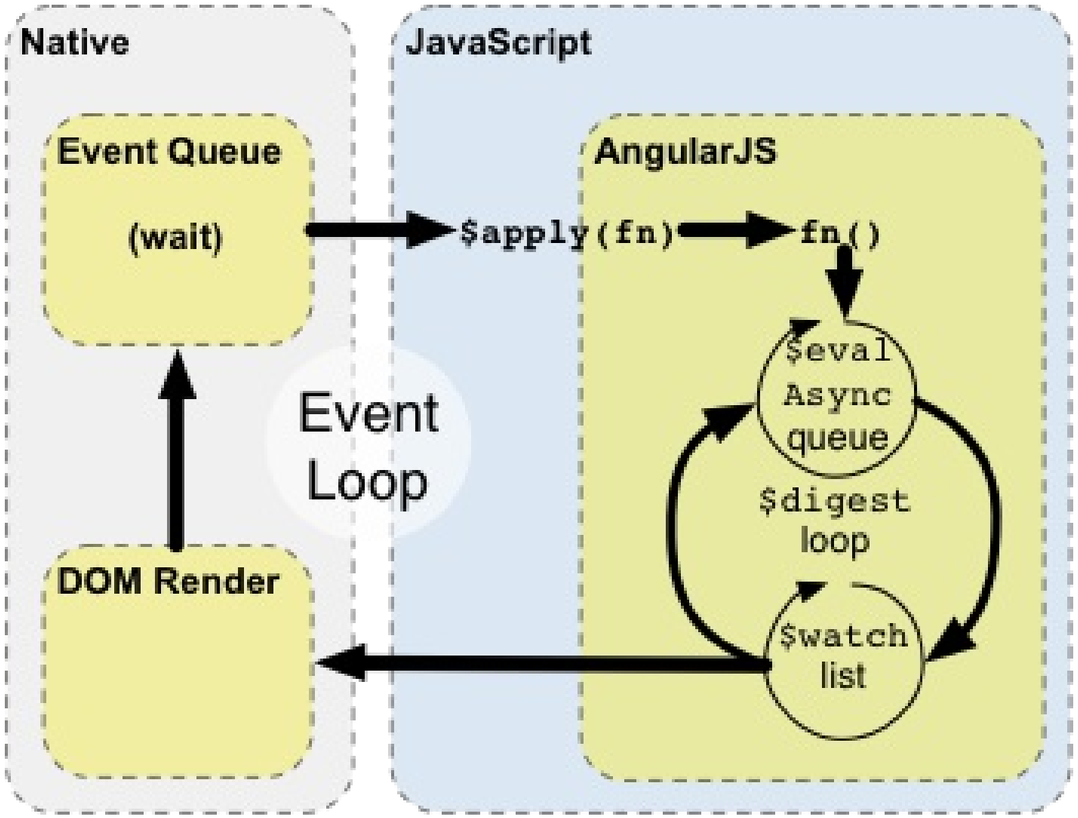
WHY ANGULAR CAN BE GOOD (2)

1. FLEXIBLE! As big or small as you want it to be
   - Two line jQuery replacement to a MASSIVE enterprise app
2. POJOs make life so easy. No ‘observables’, wrappers etc. Uses dirty checking for 2-way binding.
   - Fully embraces the dynamic nature of JavaScript
3. The community and popularity
4. DI, services, factories, providers offer flexibility and familiarity to traditionally server side paradigms
5. Directives offer DSL-like extension to HTML for your domain specific use cases
6. Scopes, although tricky, offer extreme flexibility

Open

WHY ANGULAR CAN BE GOOD (2)

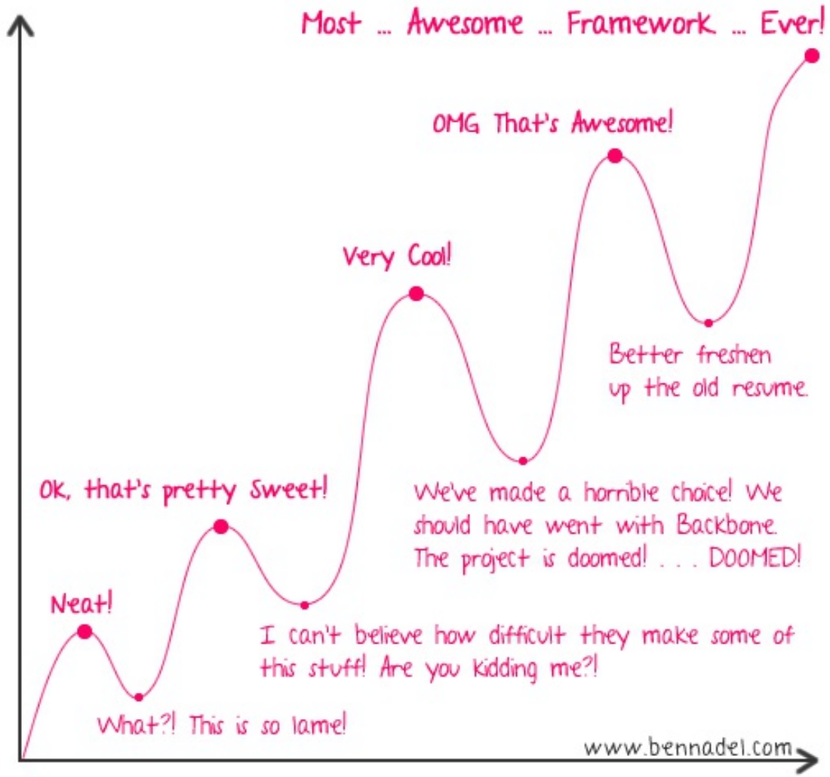
1. Efficient



Open

WHY I DON'T DIDN'T LIKE ANGULAR

- Scopes are hard initially, but awesome
- Learning curve === eventual productivity



My Feelings About AngularJS Over Time

Open

CODE TIME! A SIMPLE EXAMPLE

```
<html>
<head>
  <script src="https://code.angularjs.org/1.5.3/angular.min.js"></script>
</head>
<body  ng-app="my-app">
  <div ng-controller="exempleCtrl">
    <label>Name:</label>
    <input type="text" ng-model="yourName">
    <hr>
    <h1>Hello {{yourName}}!</h1>
  </div>
</body>
</html>
```

Open

CODE TIME! A SIMPLE EXAMPLE

```
var app = angular.module("my-app", []);

app.controller("exempleCtrl", function($scope) {
   $scope.yourName = "World"
});
```

Open

CODE TIME! A SIMPLE EXAMPLE

```
var app = angular.module("my-app", []);
app.controller("exempleCtrl", function($scope) {
    $scope.yourName = "World";
    var observeYourName = function(){
      console.log('new value' + $scope.yourName);
    };
    $scope.$watch('yourName', observeYourName);
});
```

Open

SIMPLE EXAMPLE

- ng-app attribute causes Angular to scan children for recognized tokens
  - Creates the “root scope” $rootScope
  - $rootScope ≈ a ViewModel (Angular sees three “directives”
  - *({{yourName}}*
- Evaluated against the current $rootScope and updates the DOM on any change. "1 – way bound"
  - *ng-model="firstName"*
- An input to be 2-way bound against $scope.yourName
  - *ng-model="lastName"*
- An input to be 2-way bound against *$scope.yourName*

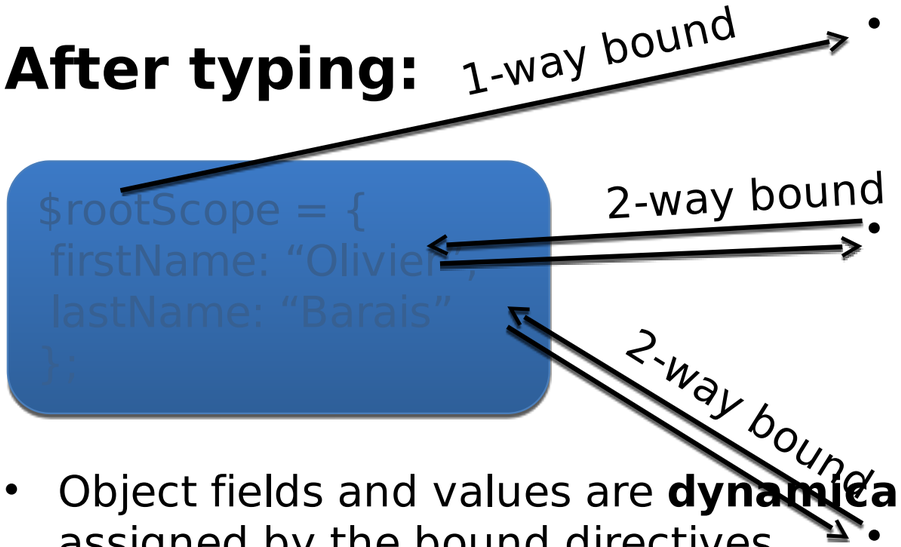
Open

Original $rootScope:

Directives

Perform the 1 or 2 way binding between the DOM and the model ($rootScope)

After typing:



- Object fields and values are **dynamically** assigned by the bound directives.

- {{firstName + " " + lastName}}
  - Watch for $scope changes and reevaluate the expression
- ng-model="firstName"
  - Watch for $scope.firstName changes, update the textbox
  - Watch for textbox changes, upd $scope.firstName
- ng-model="lastName"
  - Watch for $scope.lastName changes, update the textbox
  - Watch for textbox changes, upd $scope.lastName

Open

WHAT IS SCOPE?

- Scope is an object that refers to the application model. It is an execution context for expressions. Scopes are arranged in hierarchical structure which mimic the DOM structure of the application. Scopes can watch expressions and propagate events. (from Angular website)
- **Key points**
  - Scope is like a ViewModel that allows communication between JavaScript code and Views
  - *{{yourName}}* is an expr executed against scope
  - Scope can be hierarchal with DOM nesting of directives

Open

WHAT IS A DIRECTIVE?

- A reusable component for performing DOM interaction, templating and possibly two-way binding against $scope
  - The ONLY place JS to DOM interaction should occur
- Angular offers a huge amount of built in directives for common UI tasks, ex:

```
<div ng-show="someBool">someBool is true!</div>
```

  - 2 way binding inputs, setting classes, foreach loops of elements, clicking etc.

Open

WHAT IS A DIRECTIVE?

- You can write your own directives for domain specific purposes (a ‘DSL’ for HTML). Ex:

```
<slideshow title="Shocked Cats">
<slide src="cat1.jpg"></slide>
<slide src="cat2.jpg"></slide>
...
</slideshow>
```

Open

WHAT IS A DIRECTIVE?

- Or simply invoke an existing jQuery plugin

```
<datepicker ng-model="aDate"></datepicker>
```

- Or if you need <=IE8 support:

```
<input type="text" datepicker="" ng-model="aDate"/>
```

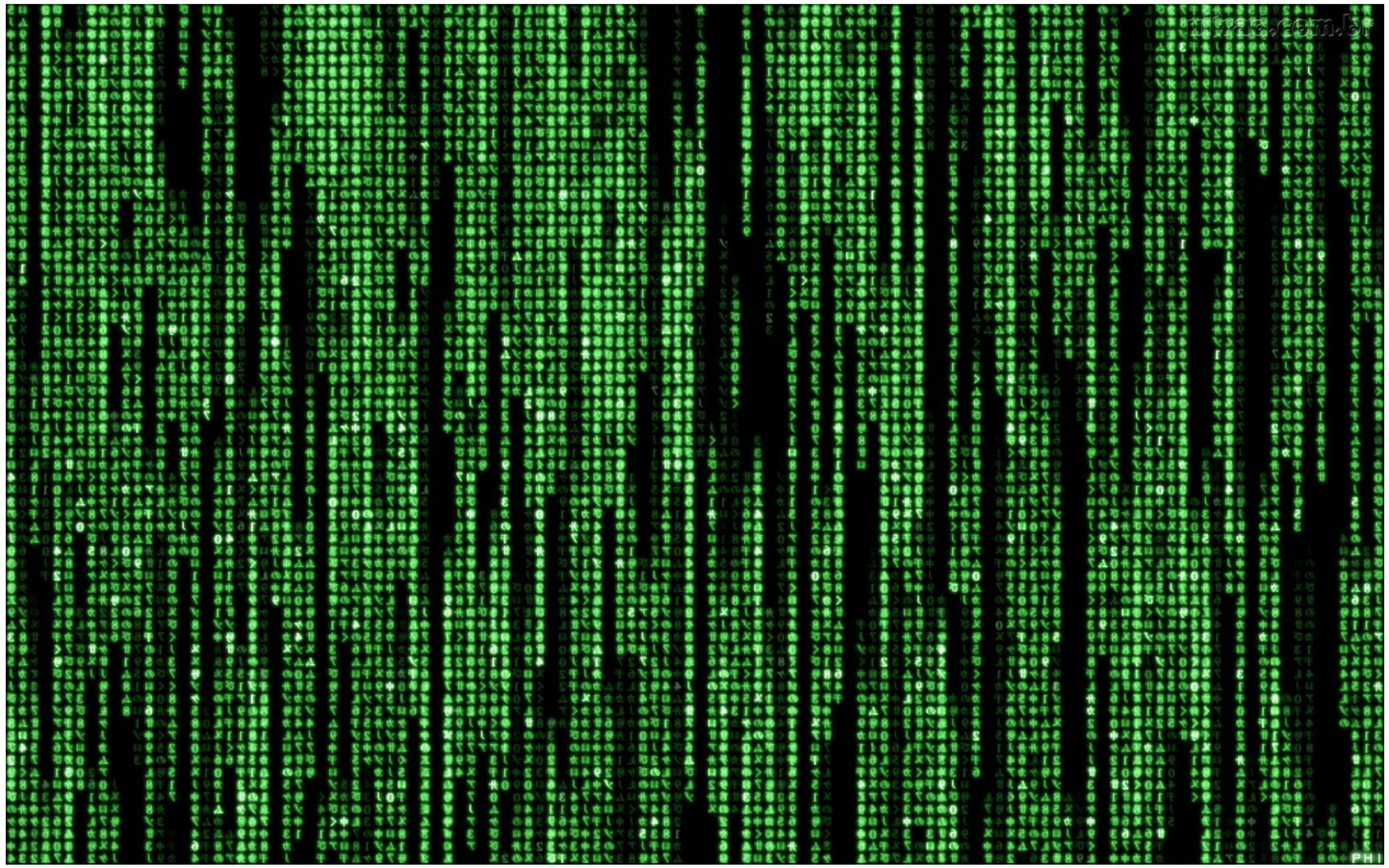
Open

WHAT IS A DIRECTIVE?

- HUGE amount of directives out there due to Angular's popularity. Rarely have to write your own other than domain specific directives
  - EX: AngularUI
  - Twitter bootstrap wrappers
  - Select2
  - Sorting
  - Input masking
  - Enhanced router
  - Etc…
  - Various wrappers for jQuery UI components (ex: datepicker)

Open

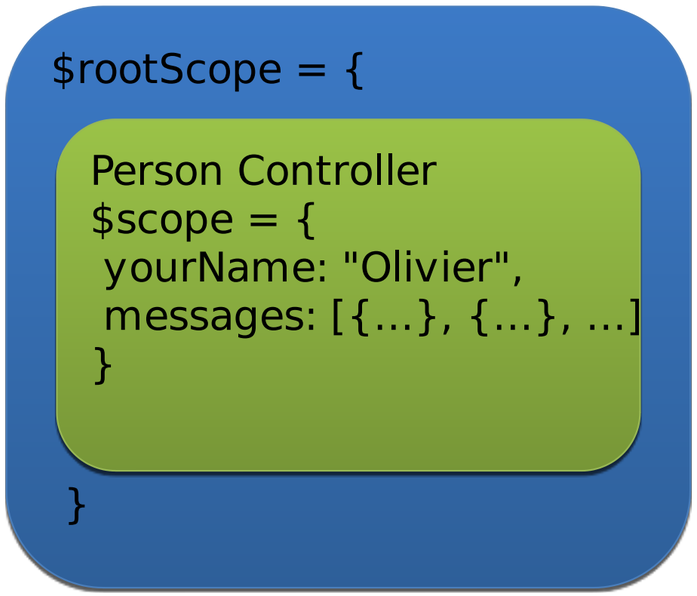
ADDING “MESSAGE UPDATES" WITH A CONTROLLER



Open

WHAT IS A CONTROLLER?

- A controller is really just a fancy name for a "scope container" that prototypically inherits from its parent scope container.
- A controller can interact with **$scope** (the 'view model') which the view can also interact with.

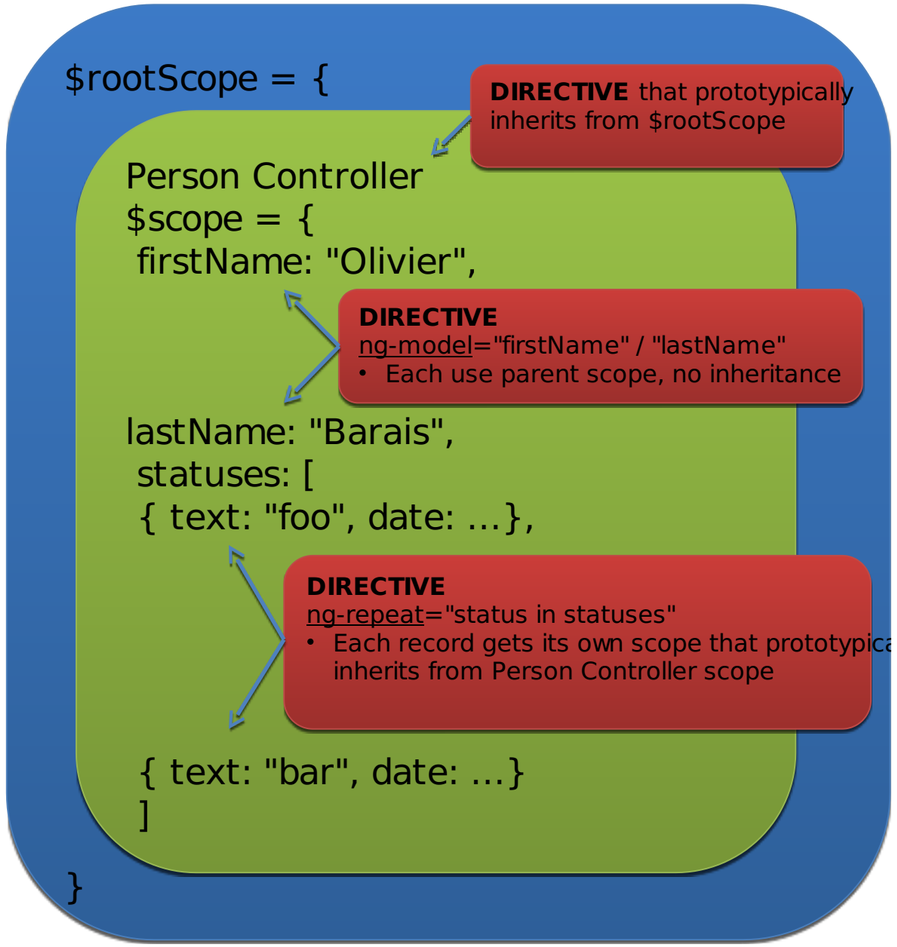


Open

DIRECTIVES AND SCOPE

- A controller is really a directive that is configured to prototypically inherit from its parent
- Directives can be configured for what type of scope they create and parent access they have

Use "AngularJS Batarang" plugin for Chrome to explore scopes



Open

SERVICES

- Software architectural components
- Services provide data and compute
- Exist across views
- Depend on other services
- AngularJS has 20+
- *$http* – service to communicate with servers
- *$log* – service to do pretty log messages

Open

SERVER COMMUNICATION

- *$http* service
  - Input config object
  - Returns promise
  - Communication is asynchronous

```
$http({method: ‘GET’, url: fetchUrl})
  .success(function(data, status) {
    // process the data here
  })
  .error(function(data, status) {
    // error handling
});
```

Open

PROMISES

- Promises represent result of an action
- Particularly used with asynchronous actions
- They are either resolved or rejected

Open

DEPENDENCY INJECTION (DI)

- DI is a software architectural pattern
- Separation of concerns
- Service creation independent from usage
- Good for
- Modular code
- Allows AngularJS to wire in correct order
- Supports substitution (for patching and testing)

Open

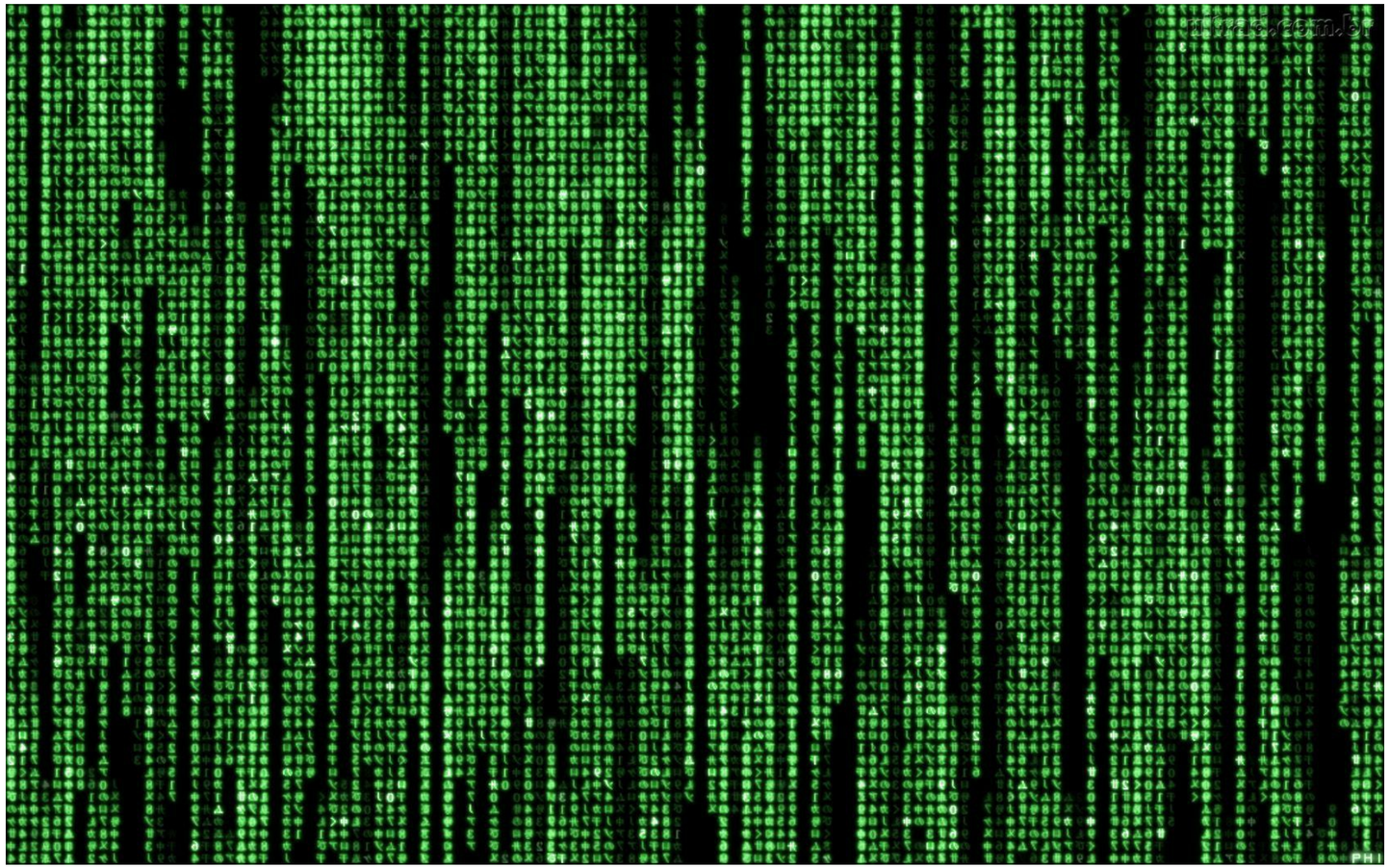
DI AND JAVASCRIPT MINIFICATION

- Services identified by parameter name
- Minification obfuscates the name
- Pass names as strings in array

```
angular.module('GoaleryServices')
  .factory('StatusManager',
    [          'CloudLogin', '$q',
      function (cloudLogin,   $q) {
...
}]);
```

Open

FIXING THE UGLY DATES AND ORDERING WITH FILTERS



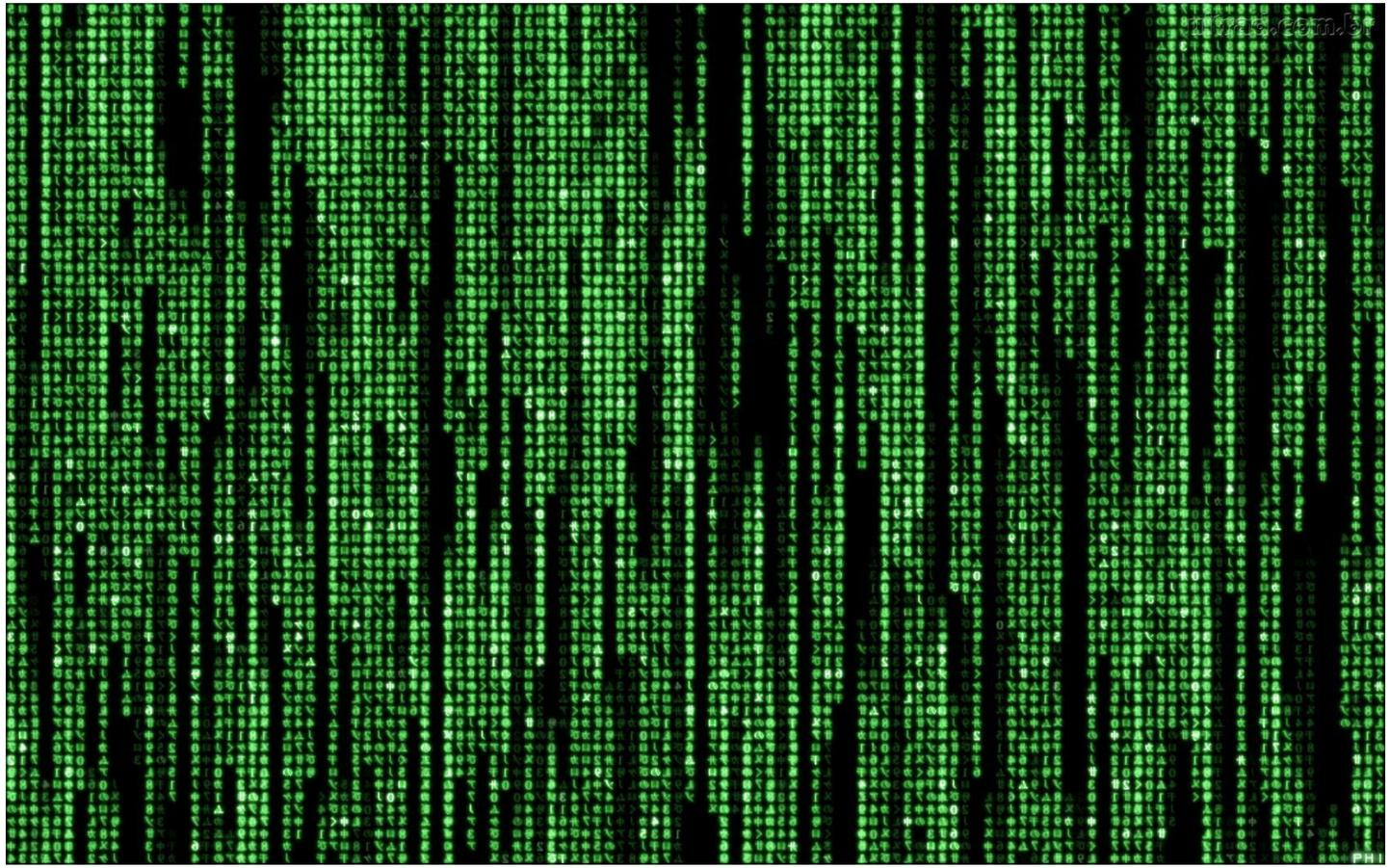
Open

WHAT IS A FILTER?

- A function that transforms an input to an output
  - Reminds me a lot of LINQ extension method lambdas in .NET
  - Can be "piped" UNIX style
  - Can create own
  - Angular has many built in filters:
    - currency, date, filter, json, limitTo, lowercase, number, orderBy, uppercase

Open

VALIDATION WITH NG-FORM



Open

WHAT IS NG-FORM?

- NOT a traditional HTML "form"
  - Requires a "name" and "ng-model" on each input you wish to validate
  - Angular will not push invalid values back into bound $scope
- Allows for validation of collections of controls
  - Applies CSS classes to elements based on their validity
  - Lots of built in validator directives that work with ng-form:

```
required=""
ng-minlength="{number}"
ng-maxlength="{number}"
ng-pattern="{string}"
ng-change="{string}"
```

Open

WHAT IS NG-FORM?

- Angular UI has some extensions
- AngularAgility - FormExtensions makes it easier
  - demo

Open

ANGULARJS IS PURE JAVASCRIPT

- Prototype-based scripting language
- Dynamic, weakly typed, first-class functions
- Great JavaScript book:
  - Crockford (2008) JavaScript: The Good Parts – O'Reilly

Open

TESTING ANGULARJS APPS

- JavaScript doesn’t have a compiler
- Must execute code to test
- Testability was a fundamental objective of AngularJS
  - Miško Hevery (AngularJS creator)
  - Previously created JsTestDriver

Open

ANGULARJS SUPPORTS TESTING

- Unit testing support
  - JsTestDriver
  - Jasmine
- DI allows substituting mocks for hard to test code
  - Server communication
  - Logging
- Angular Scenario Runner – E2E testing
  - Simulates user interations

Open

BUILDING WEB APPS

- Single web page
  - Loads the base HTML and included sources once
  - App changes views dynamically
- Server is called upon when needed
- Prefer using asynchronous server calls
  - Data changes
  - Fetch more data

Open

BUILDING WEB APPS

- Declarative view specification
- HTML augmented with:
- Directives, Markup, Filter, Form Controls
- Loaded either
- with a simple single web page
- dynamically into a view as partials

Open

URL ROUTING

- Define the mapping from URL to view
- Can also bind controller
- Define URL parameters

```
$routeProvider.when('/Book/:bookId', {
    templateUrl: 'book.html',
    controller: BookCntl
  });
  $routeProvider.when('/Book/:bookId/ch/:chapterId', {
    templateUrl: 'chapter.html',
    controller: ChapterCntl
  });
```

Open

DEEP LINKING

- AngularJS navigation updates the browser address bar
- Uses the HTML5 history API – fallback to hash-bang (#!) URL
- Users can link to pages in you app
- Server must recognize the link pattern and serve the application
- https://github.com/angular-ui/ui-router/wiki/Quick-Reference

Open

CREATING DIRECTIVES

- Directives package reusable HTML
- Naming nuance: "myDir" becomes "my-dir"
- Conceptual compile and link phases
- Can specify: scope, binding, restrictions, etc
- Supports transclusion
  - In computer science, transclusion is the inclusion of part or all of an electronic document into one or more other documents by hypertext reference.
- Consider creating a custom DSL

Open

MODULES

- Packaging of JavaScript code
- Modules declare dependencies
- AngularJS instantiates in correct order
- Provides separation of namespaces

Open

LET US PRACTICE

- Let us use the yeoman generator
- Install yo, grunt-cli, bower, generator-angular and generator-karma:

```
npm install -g grunt-cli bower yo generator-karma generator-angular
```

If you are planning on using Sass, you will need to first install Ruby and Compass:

- Install Ruby by downloading from here or use Homebrew
- Install the compass gem:

```
gem install compass
```

Open

LET US PRACTICE

- Make a new directory, and cd into it:

```
mkdir my-new-project && cd $_
```

- Run yo angular, optionally passing an app name:

```
yo angular [app-name]
```

Run grunt for building and grunt serve for preview

Open

GENERATORS AND SUB

- angular (aka angular:app)
- angular:controller
- angular:directive
- angular:filter
- angular:route
- angular:service
- angular:provider
- angular:factory
- angular:value
- angular:constant
- angular:decorator
- angular:view

Open

APP

- Sets up a new AngularJS app, generating all the boilerplate you need to get started. The app generator also optionally installs Bootstrap and additional AngularJS modules, such as angular-resource (installed by default).

Example:

```
yo angular
```

Open

ROUTE

- Generates a controller and view, and configures a route in app/scripts/app.js connecting them.
- Example:

```
yo angular:route myroute
```

  - Produces app/scripts/controllers/myroute.js:

```
angular.module('myMod').controller('MyrouteCtrl', function ($scope) {
  // ...
});
```

    - Produces app/views/myroute.html:

```
<p>This is the myroute view</p>
```

Open

- **Explicitly provide route URI**
- Example:

```
yo angular:route myRoute --uri=my/route
```

- Produces controller and view as above and adds a route to app/scripts/app.js with URI my/route

Open

CONTROLLER

- Generates a controller in app/scripts/controllers.
- Example:

```
yo angular:controller user
```

- Produces app/scripts/controllers/user.js:

```
angular.module('myMod').controller('UserCtrl', function ($scope) {
// ...
});
```

Open

DIRECTIVE

- Generates a directive in app/scripts/directives.
- Example:

```
yo angular:directive myDirective
```

- Produces app/scripts/directives/myDirective.js:

```
angular.module('myMod').directive('myDirective', function () {
return {
  template: '<div></div>',
  restrict: 'E',
  link: function postLink(scope, element, attrs) {
    element.text('this is the myDirective directive');
  }
};
});
```

Open

FILTER

- Generates a filter in app/scripts/filters.
- Example:

```
yo angular:filter myFilter
```

Produces app/scripts/filters/myFilter.js:

```
angular.module('myMod').filter('myFilter', function () {
  return function (input) {
    return 'myFilter filter:' + input;
  };
});
```

Open

VIEW

- Generates an HTML view file in app/views.
- Example:

```
yo angular:view user
```

- Produces app/views/user.html:

```
<p>This is the user view</p>
```

Open

SERVICE

- Generates an AngularJS service.
- Example:

```
yo angular:service myService
```

- Produces app/scripts/services/myService.js:

```
angular.module('myMod').service('myService', function () {
// ...
});
```

- You can also do yo angular:factory, yo angular:provider, yo angular:value, and yo angular:constant for other types of services.

Open

DECORATOR

- Generates an AngularJS service decorator.
- Example:

```
yo angular:decorator serviceName
```

- Produces app/scripts/decorators/serviceNameDecorator.js:

```
angular.module('myMod').config(function ($provide) {
  $provide.decorator('serviceName', function ($delegate) {
    // ...
    return $delegate;
  });
});
```

Open

OPTIONS

In general, these options can be applied to any generator, though they only affect generators that produce scripts.

Open

COFFEESCRIPT AND TYPESCRIPT

For generators that output scripts, the *--coffee* option will output CoffeeScript instead of JavaScript, and *--typescript* will output TypeScript instead of JavaScript.

- For example:

```
yo angular:controller user --coffee
```

- Produces app/scripts/controller/user.coffee:

```
angular.module('myMod')
.controller 'UserCtrl', ($scope) ->
```

Open

- For example:

```
yo angular:controller user --typescript
```

- Produces *app/scripts/controller/user.ts*:

```
/// <reference path="../app.ts" />

'use strict';

module demoApp {
    export interface IUserScope extends ng.IScope {
        awesomeThings: any[];
    }

    export class UserCtrl {

        constructor (private $scope:IUserScope) {
            $scope.awesomeThings = [
              'HTML5 Boilerplate',
              'AngularJS',
              'Karma'
```

Open

MINIFICATION SAFE

tl;dr: You don't need to write annotated code as the build step will handle it for you.

By default, generators produce unannotated code. Without annotations, AngularJS's DI system will break when minified. Typically, these annotations that make minification safe are added automatically at build-time, after application files are concatenated, but before they are minified. The annotations are important because minified code will rename variables, making it impossible for AngularJS to infer module names based solely on function parameters.

The recommended build process uses ng-annotate, a tool that automatically adds these annotations. However, if you'd rather not use it, you have to add these annotations manually yourself. Why would you do that though? If you find a bug in the annotated code, please file an issue at ng-annotate.

Open

ADD TO INDEX

- By default, new scripts are added to the index.html file. However, this may not always be suitable. Some use cases:
  - Manually added to the file
  - Auto-added by a 3rd party plugin
  - Using this generator as a subgenerator
- To skip adding them to the index, pass in the skip-add argument:

```
yo angular:service serviceName --skip-add
```

Open

BOWER COMPONENTS

- The following packages are always installed by the app generator:
  - angular
  - angular-mocks

Open

BOWER COMPONENTS

- The following additional modules are available as components on bower, and installable via `bower install`:
- angular-animate
- angular-aria
- angular-cookies
- angular-messages
- angular-resource
- angular-sanitize

All of these can be updated with `bower update` as new versions of AngularJS are released.

Open

CONFIGURATION

Yeoman generated projects can be further tweaked according to your needs by modifying project files appropriately.

Open

OUTPUT

- You can change the app directory by adding an appPath property to bower.json. For instance, if you wanted to easily integrate with Express.js, you could add the following:

```
{
 "name": "yo-test",
 "version": "0.0.0",
 ...
 "appPath": "public"
}
```

- This will cause Yeoman-generated client-side files to be placed in public.

Open

OUTPUT

- Note that you can also achieve the same results by adding an --appPath option when starting generator:

```
yo angular [app-name] --appPath=public
```

Open

TESTING

Running

```
grunt test
```

will run the unit tests with karma.