EMN FIL

Sommaire

AngularJS

Qu’est-ce que AngularJS ?

- Un **framework** JavaScript pour créer des **S**ingle **P**age **A**pplication (SPA)
- Ce n’est pas une librairie
- Mais une solution complète pour créer une application coté client
- AngularJS vous permet d’étendre l’HTML
- OpenSource

Pourquoi utiliser un framework ?

Historique

- **GetAngular** est créé en 2009 par deux développeurs :

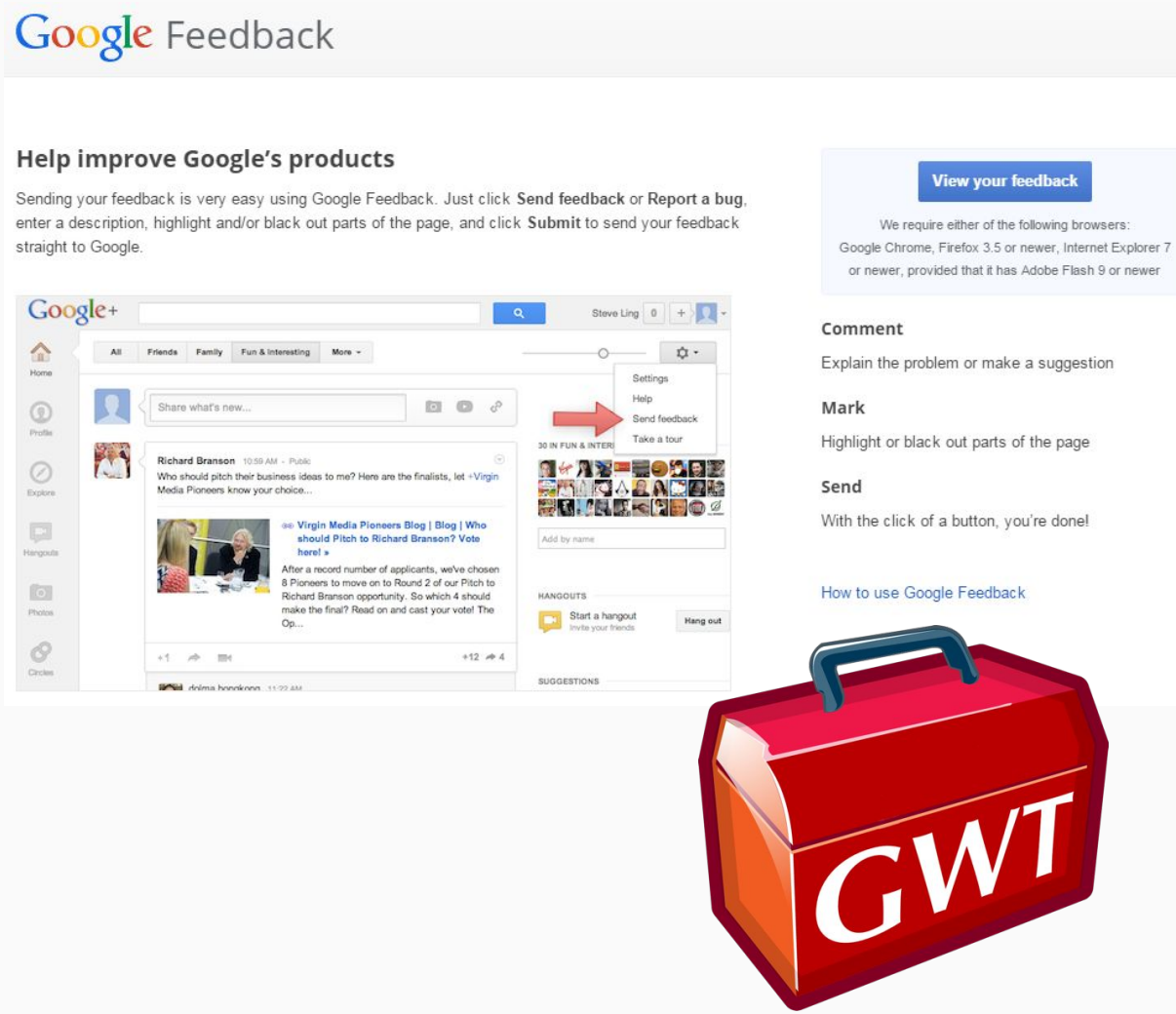
  Misko Hevery et **Adam Abrons**

- Un “outil” permettant d'interagir à la fois avec le front et le back

Pendant ce temps là chez Google ...

Historique

- Projet de 6 mois
- 17 000 lignes de code
- Appli difficilement maintenable
- Beaucoup de code Front non testable



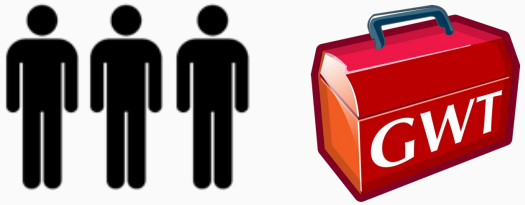
Historique

Misko Hevery rejoint l'équipe de **Google Feedback** en 2009

\+ GetAngular +

Pari avec son manager

Historique

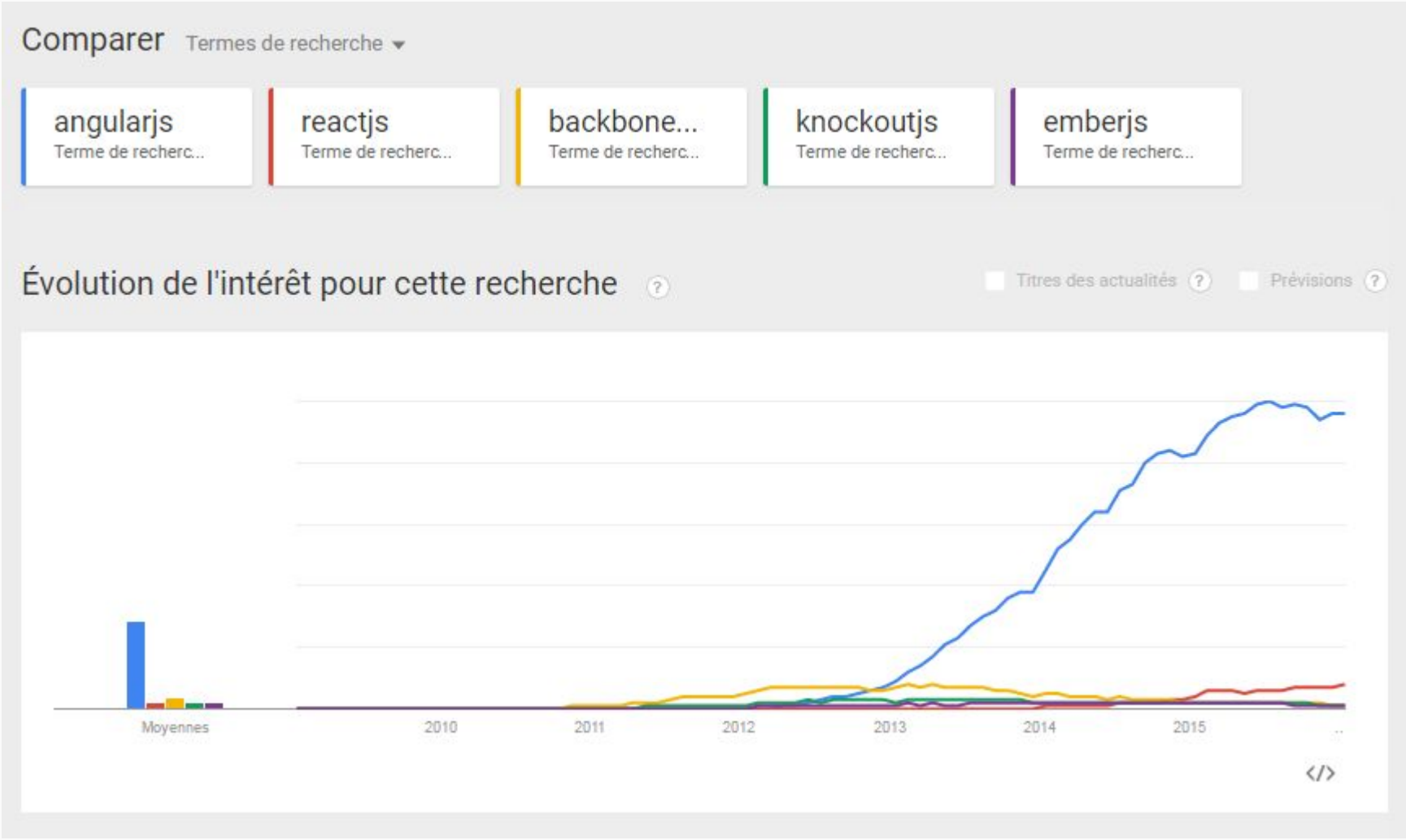


17 000
lignes de code

1 500
lignes de code **testées**

“Le framework JS le plus populaire ?”

“Le framework JS le plus populaire ?”




Google

Concepts

- **MVC** (ou MV-VM, en fait MVW Model View Whatever)
- **2 ways data binding** ❤️
- **Modules** & **Dependency Injection**
- **Tests unitaires**

Mais aussi :

- **Directive**
- Template
- Scope
- Filtre
- Service
- Factory
- Controller
- Decorator
- Provider
- ...

Exemples d’applications

www.madewithangular.com

On essaye ?

Bootstrap *(= initialisation d'AngularJS)*

```
ng-app="appName"
```

```
angular.bootstrap(
    element,
    ["appName"]
);
```

Bootstrap

```
1.  <!doctype html>
2.  <html ng-app>
3.   <head>
4.     <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.9/angular.min.js"></script>
5.   </head>
6.   <body>
7.     <div>
8.       <label>Name:</label>
9.       <input type="text" ng-model="yourName" placeholder="Enter a name here'>
10.      <hr>
11.      <h1>Hello {{yourName}}!</h1>
12.    </div>
13.  </body>
14. </html>
```

Exercice 1

Hello World

- Bootstrap (**ng-app** + **angular.min.js**)
- Créer un input pour le nom (**ng-model**)
- Afficher le résultat avec une expression {{ }}

```
<html>
 <head>
   <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/
1.4.9/angular.min.js"></script>
 </head>
 <body>
    <!-- votre code -->
 </body>
</html>
```


Donne des super pouvoirs à HTML

Directives

“Donnent des super pouvoirs à votre HTML”

`ng-app`

`ng-init`

`ng-model`

`ng-repeat`

Exercice 2

Créer et afficher une liste de contacts

- Bootstrap (**ng-app** + **angular.min.js**)
- Initialiser la liste de contacts (**ng-init**)

```
["Victor","Rémi","Jacques"]
```

- Afficher la liste (**ng-repeat)**

Filtres

A utiliser dans la vue, pour modifier, filtrer, ou organiser les variables

`uppercase`

`filter`

`orderBy`

Exercice 3

Utilisation de filtres

- Modifier la liste de contacts pour travailler avec des objets

```
{
    firstName: ‘Victor’,
    lastName: ‘Lambert’,
    phone : ‘0668548448’
}
```

- Appliquer le filtre **`uppercase`** sur le nom de votre contact
- Trier la liste par ordre alphabétique suivant le prénom (**`orderBy`**)
- Créer un champ input pour filtrer la liste de contacts (**`filter`**)
- Utiliser une liste déroulante pour trier la liste de contacts par le nom ou le prénom

Les modules

Modules

Pour organiser nos différents composants et mettre un peu d'ordre dans tout ça !

Création d'un module (avec ses dépendances)

```
angular.module('app',[]);
```

Récupération d'un module (pour y ajouter un controller, une directive ...)

```
angular.module('app');
```

Controller et Scope

Scope : permet de faire la liaison entre le Controller et la Vue (ViewModel)

```
function ContactsCtrl($scope){
   $scope.contacts = [
      {
         firstname: “Victor”,
         lastname: “Lambert”,
         phone : “0668548448”
      },
      ...
   ];
}
```

Dependency Injection

Implicit Annotation /!\ Minification /!\

```
someModule.controller('MyController', function($scope, greeter) {
  // ...
});
```

Inline Array Annotation (recommandée)

```
someModule.controller('MyController', ['$scope', 'greeter', function($scope, greeter) {
  // ...
}]);
```

`$inject` Property Annotation (recommandée)

```
someModule.controller('MyController', MyController);

MyController.$inject = ['$scope', 'greeter'];

function MyController($scope, greeter) {
  // ...
}
```

Exercice 4

Création d'un module et d'un controller

- Créer un module pour organiser l'application et préciser le nom à **`ng-app`**
- Créer un **`ContactsCtrl`** qui fournit à la vue la liste des contacts grâce à **`$scope`** (**`ng-controller`**)

  `http://www.json-generator.com/NJOXnL2tl`
- Utiliser la notation **`$inject`** pour injecter **`$scope`**

Exercice 4 (suite)

Utilisation de $scope

- Initialiser la valeur du tri avec **`ng-init`** puis afficher le **`$scope`** dans le controller
- Dans le **`ng-repeat`**, créer un champ input lié à la même variable que celle du filtre de contacts
- Filtrer depuis le filtre global, puis sur un des filtres dans le **`ng-repeat`**, que constatez-vous ?
- Installer l'extension chrome **AngularJS Batarang** pour visualiser le **`$scope`** et comprendre le fonctionnement constaté précédemment

Les routes

Routes

Pour gérer la transition entre nos différentes pages

```
angular.module('contactsApp.routes', ['ngRoute'])
    .config(['$routeProvider', function($routeProvider) {
      $routeProvider
        .when('/', {
          templateUrl: 'partials/home.html'
        })
        .when('/contacts', {
          templateUrl: 'partials/contacts.html',
          controller: 'ContactsCtrl'
        })
        .otherwise({
          redirectTo: '/'
        });
    }]);
```

Exercice 5

Routes et organisation de notre application

- Charger **angular-route.min.js**
- Créer **app.js**, **controllers.js** et **routes.js**
- Configurer 2 routes (Accueil et Liste de contacts) dans **routes.js**
- Déplacer le code HTML dans un nouveau fichier (**partials/contacts.html**) et remplacer par **ng-view**
- Créer une page d'accueil (**partials/home.html**)
- Configurer une redirection vers l'accueil en cas d'une route inconnue
- Configurer deux nouvelles routes pour créer et éditer un contact ainsi que vues/controllers associées
- et on s'arrête là … nous avons besoins d'un "model" pour gérer nos données

Services

Le “Model” de notre application

```
someModule.service('MyService', function () {
  this.sayHello = function () {
    console.log('hello');
  };
});
```

- Singletons
- Fonction constructeur
- Appelée avec un **`new`** à l’injection

Exercice 6

Création d'un service pour partager nos données entre les différents constructeurs

- Créer un fichier **services.js** et un service **ContactSrv** avec les méthodes **getContacts()**, **getContact(id)**, **saveContact(contact)** et **deleteContact(id)**
- Injecter le service dans **ContactsCtrl** et **EditContactCtrl** pour pouvoir utiliser ses méthodes
- Utiliser **$routeParams** pour récupérer l'id du contact à éditer
- Utiliser **$location** pour rediriger vers la page des contacts à la fin de l'édition/création
- On recharge la page après une création/modification, pas de persistance ...

Services *vs* Factory *vs* Provider

- Service et Factory sont des Provider
- Un service est appelé avec un `new` (Fonction constructeur) et retourne donc `this` par défaut
- Une factory retourne le résultat de la fonction
- Un service peut retourner autre chose que `this` et peut donc être écrite comme une factory
- Un provider retourne le résultat de `this.$get`
- Un provider permet d'être configuré avant d'être utilisé
- On peut écrire un service ou une factory à partir d'un provider

Les 3 sont plus ou moins la même chose, dans le sens où ils sont 3 manières d'instancier des objets singletons

Explications en détail, ou encore

Promesses

Pour gérer les traitements asynchrones

```
// Simple GET request example:
$http({
  method: 'GET',
  url: '/someUrl'
}).then(function successCallback(response) {
    // this callback will be called asynchronously
    // when the response is available
  }, function errorCallback(response) {
    // called asynchronously if an error occurs
    // or server returns response with an error status.
  });
```

Exercice 7

Externalisation des données dans un fichier JSON

- Déplacer les contacts dans un fichier **contacts.json**
- Créer une fonction dans le service qui charge votre fichier JSON et retourne la liste des contacts
- Améliorer cette fonction en ne chargeant que les données qu'au premier accès, et retourner un objet en mémoire les autres accès
- Utiliser `$q` pour renvoyer une promesse à partir d'un objet, et faire en sorte que la fonction renvoie toujours une promesse
- Adapter les méthodes à cette fonction (le service ne renverra plus que des promesses), et adapter ensuite les appels depuis les controllers

Exercice 7 (bis)

Simulation d’un accès serveur

- Utiliser **`$timeout`** pour simuler un accès serveur lors du chargement du fichier JSON
- Grâce au **`$rootScope`** et aux directives **`ng-show`** et **`ng-hide`**, afficher un message de chargement

Exercices

Réponses

https://goo.gl/xvZKes

https://goo.gl/eqBwXS