Bien choisir son Frameworks JavaScript: AngularJS, Backbone, Ember, Knockout

Choisir son Framework front end n'est jamais une chose facile, ce choix impactera le temps de chargement, la vitesse de développement et la maintenabilité du code source de votre site web. Il en existe des dizaines, dans cet article nous allons nous restreindre aux 4 plus fameux afin de vous aider à faire le bon choix.

Auteur : Feltz Ludovic (ludovic.feltz@softfluent.com)

Tous ces Frameworks permettent d'organiser votre code et possèdent le concept de vue, d'évènements, de modèle de donnée, routage ? Plusieurs critères sont à prendre en compte dans le choix d'un Framework, son poids par exemple définira le temps de chargement des pages ce qui est un critère important pour les applications mobiles, sa facilité d'utilisation, sa documentation et sa communauté active permettent de réduire le temps de développement. Nous allons commencer par un bref historique puis nous allons plonger dans le vif du sujet avec un exemple simple que nous allons reproduire en utilisant chacun de ces Frameworks

TODO : On gère pas connexion a la base de données, Single page application

Why backbone: http://backbonetutorials.com/why-would-you-use-backbone/ , event driven?

Compare frameworks : http://codebrief.com/2012/01/the-top-10-javascript-mvc-frameworks-reviewed/

Histoire :

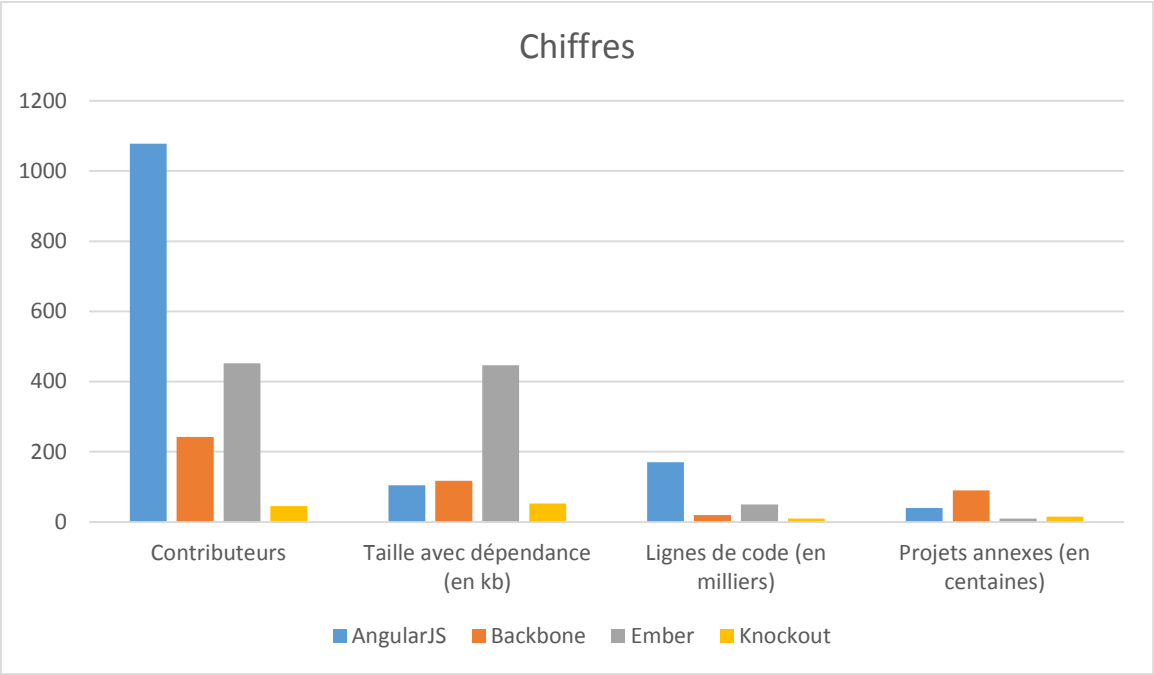
AngularJS: Apparu en 2009 sous le nom de GetAngular il est utilisé par Misko Hevery un des ingénieurs qui l'a développé pour recréer une application web qui représentait plus de 17 milles lignes de code. En 3 semaines il est parvenu à réduire ce nombre à seulement 1000 lignes ce qui a convaincu Google de sponsoriser ce projet ce qui a créé sa renommée. Son but est de simplifier le développement et les tests de site web MVC en ajoutant du vocabulaire au code HTML de votre application.

Backbone : Crée en 2010 par Jeremy Ashkenas qui à aussi participer au développement de Coffee Script, il est très léger ce qui lui a permis de se faire un nom parmi les autres Frameworks JavaScript. Il repose principalement sur le modèle MVP (model view presenter). (key value binding and custom events, collection, connect to existing API with RESTful JSon interface)

Ember : Développé en 2007 par SproutIt puis par Apple il est finalement forké en 2007 par Yehuda Katz, un des principaux contributeurs de JQuery et Ruby. Il est utilisé par des grands noms tels que Yahoo, Groupon et ZenDesk. Il est basé sur une architecture MVC. Son avantage est de permettre le développement d'applications d'une seule page grâce à son système de route et de template (view, controller, models, router)

Knockout : Né en 2010 et maintenu comme un projet open source par Steve Sanderson, employé chez Microsoft. Simplifie l'utilisation du JavaScript en appliquant le pattern MVVM.

Les chiffres



Le temps de chargement d'une page web est crucial pour sa réussite. Les utilisateurs ne montrent pas beaucoup de patience quant au temps de chargement d'une page web, c'est pourquoi il essentiel de prendre en compte le temps de chargement et d'initialisation d'une librairie. Malgré le nombre de ligne de code plus important que ses concurrent **AngularJS** est plus légère car elle n'a besoin d'aucunes dépendances. **Backbone** nécessite l'utilisation de **underscore** ce qui lui fait prendre du poids malgré son nombre de lignes de codes moins important (verifier qu'il a pas besoin de JQuery en plus, sinon il est intégré). **Ember** a besoin de **JQuery** et de **Handlebar** pour fonctionner, d'où son poids très important par rapport à ses concurrents. Enfin **Knockout** ne nécessite aucune autre librairie ce qui en fait la plus légère de ce comparatif. La taille de votre site impactera aussi, si un Framework est lourd mais son utilisation produit peu de ligne de code il pourra tout de même être avantageux.

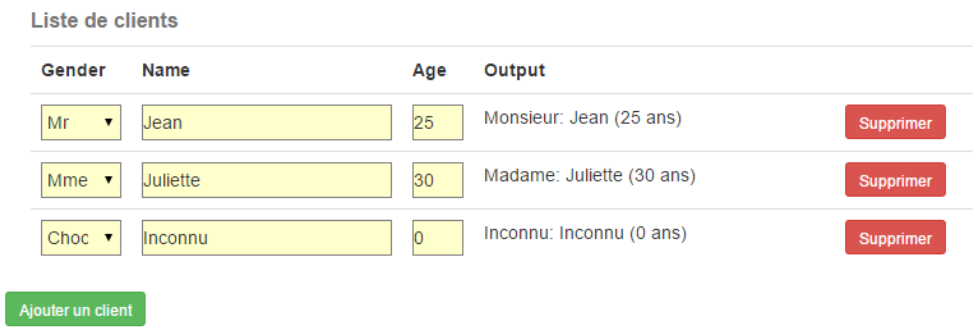
Le nombre de contributeur on s'en fou en vrai, non ?

Les concepts clefs

Pas de rafraîchissement de la page

Exemple similaire

Pour comparer ces différents Frameworks nous allons nous baser sur un exemple d'application simple que nous allons reproduire en utilisant les 4 Frameworks puis nous commenterons le code produit.



Cet exemple nous permet de gérer une liste de clients, d'en ajouter et d'en supprimer. Un affichage formaté nous permet de vérifier que lors d'une modification de notre model les modifications sont affichés directement sans rafraichir la page.

Pour peupler nos modèles nous allons à chaque fois réutiliser le même jeu de données :

```
var Genders = [
    {
        id: 0,
        key: "Mr",
        value: "Monsieur"
    },
    {
        id: 1,
        key: "Mme",
        value: "Madame"
    }
]


var ClientsList = [
    {
        id: 0,
        gender: Genders[0],
        name: "Jean",
        age: 25,
    },
    {
        id: 1,
        gender: Genders[1],
        name: "Juliette",
        age: 30,
    }
]
```

AngularJS :

Commençons par créer une application AngularJS très simple. Nous avons donc le contenu de notre page HTML (index.html):

```
<html ng-app="myApp">
<head>
    <title>AngularJS</title>

    <!-- Angular 1.3.0 (no dependency) -->
    <script src="/script/controllers.js"></script>

</head>
<body>
    <div class="container" ng-controller="AppController as controller">
        <!–
              Corps de la vue
        -->
    </div>
</body>
```

Et ce code javascript (script/controllers.js):

```
var app = angular.module('myApp', []);
app.controller("AppController", ['$scope', function ($scope) {
      /*
       *  Corps du controller
       */
}]) ;
```

On remarque que angularJS définit ses propre balise HTML commençant par ng-*. La première se trouve directement dans la balise HTML ouvrante, elle permet de définir la racine de notre application donnant ainsi la possibilité au développeur de dire à angular d'utiliser toute la page ou seulement une portion de celle-ci.

La deuxième balise que l'on voit apparaitre est ng-controller elle va nous fournir le contexte de notre application et va nous permettre de lier notre vue avec notre modèle.

Angular encourage à utiliser le pattern MVC pour structurer le code.

Ensuite nous allons peupler notre modèle avec le jeu de donnée décrit précédemment et nous allons ajouter une directive afin d'avoir un formatage du texte. Pour cela nous ajoutons simplement les variables clients, genders et une directive à notre Controller de la façon suivante:

```
app.controller("AppController", ['$scope', function ($scope) {
    $scope.clients = ClientsList
    $scope.genders = Genders;
}])
.directive('clientFormatted', function ()
    return {
        template: '{{client.gender.value}}: {{client.name}} ({{client.age}} ans)'
    };
});
```

Enfin nous modifions notre vue afin d'afficher tous les clients dans un tableau, le corps de notre page devient donc:

```
<body>
    <div class="container" ng-controller="AppController as controller">

        <!-- Corps de l'application -->

        <table class="table">
            <caption>Liste de clients avec AngularJS</caption>
            <tr>
```

```
                <th>Gender</th>
                <th>Name</th>
                <th>Age</th>
                <th>Output</th>
                <th></th>
            </tr>
            <tr ng-repeat="client in clients" title="{{client.ID}}">
                <td>
                    <select ng-model="client.gender" ng-options="gender.key for gender
in genders">
                        <option value="">Choose gender</option>
                    </select>
                </td>
                <td><input type="text" ng-model="client.name"></td>
                <td><input type="text" ng-model="client.age"></td>
                <td><client-formatted></client-formatted></td>
                <td><button type="button" ng-
click="removeClient(client);">Supprimer</button></td>
            </tr>
        </table>
        <button type="button" ng-click="addClient();">Ajouter un client</button>
    </div>
</body>
```

On voit apparaitre ici 4 nouvelles balise appartenant à AngularJS. Ng-repeat s'apparente à une « foreach ». Pour chaque clients de notre controller on crée une balise <tr> dont le titre sera l'ID de ce client. Les accollades : {{expression}} permette à Angular d'effectuer une liaison avec le modèle.

Ng-model permet d'effectuer une liaison dans les deux sens (two way binding) avec notre modèle. C'est-à-dire qu'il permet de récupérer la valeur du modèle et que dès lors qu'une valeur est modifiée le modèle est automatiquement mis à jour !

Ng-option permet de peupler notre liste déroulante et de définir ce qui sera affiché. Ici on affichera la clef de chaque « gender »

Enfin ng-click permet de spécifier le nom de la fonction qui sera appeler lors d'un click sur le bouton. Il ne nous reste plus qu'a ajouter nos fonctions « removeClient » et « addClient » à notre controller :

```
$scope.addClient = function () {
    $scope.clients.push({ gender: Genders[0], name: "Inconnu", age: 0 });
};
$scope.removeClient = function (client) {
    index = $scope.clients.indexOf(client);
    $scope.clients.splice(index, 1);
};
```

addClient ajoute seulement un nouveau client à notre collection grâce à la fonction JavaScript push() et removeClient supprime un client avec splice.

Simple n'est-ce pas ? Passons maintenant à Knockout, nous verrons que son comportement est très proche de celui d'AngularJS.

Knockout :

Knockout utilise le pattern MVVM. Une petite explication de ce pattern qui permet de garder une organisation simple d'une application graphique:

- Le « **model** »: indépendant de l'interface graphique c'est lui qui stocke les informations (généralement en base de donnée). Dans cet article nous n'aborderons pas de la persistance des données en base.
- Le « **view model** » : est une représentation des données en code seulement (ici en Javascript), par exemple une liste de donnée avec des fonctions permettant d'ajouter ou supprimer des éléments, etc … Ce sont des données non sauvegardé sans lien avec l'interface graphique puisqu'il n'y a aucune notion d'affichage ou de boutons.
- La « **view** » : affiche les informations du « **view model** », envoi des commandes tel qu'un clique sur un bouton, et est automatiquement mis a jour dès qu'il y a un changement sur le « **view model** »

Avec Knockout il faut lier manuellement le modèle à la vue. Commençons donc par modifier notre modèle, Genders n'étant pas modifiable sa structure ne change pas :

```
var Genders = [ //Immutable (don't need observable)
    {
        id: 0,
        key: "Mr",
        value: "Monsieur"
    },
    {
        id: 1,
        key: "Mme",
        value: "Madame"
    }
]
```

Nous modifions donc le modèle de **Client**, on crée donc un constructeur avec la structure suivante :

```
var Client = function (id, gender, name, age) {
    this.id = id;
    this.gender = ko.observable(gender);
    this.name = ko.observable(name);
    this.age = ko.observable(age);

    this.clientFormatted = ko.computed(function () {
        return this.gender().value + ": " + this.name() + " (" + this.age() + " ans)";
    }, this);
}
```

Knockout a besoin de savoir quels champs du **view model** observer afin de notifier la ou les **vue** des changements, pour cela on utilise **ko.observable()** qui prend en paramètre la valeur par défaut.

Comme avec AngularJS on veut avoir une sortie formatée qui affiche les informations de notre Client. Pour cela on utilise **ko.computed(function(){…}).** Permet à knockout de savoir que cette propriétée dépend d'un ou plusieurs **observable** et donc qu'il doit la mettre en jours lorsqu'une des dépendances est modifié.

Exemple des fonctionnalités propres

Développer son propre Framework ?

Utiliser un Framework connu à l'avantage d'offrir au développeur de la rapidité sur un Framework qu'il connait déjà et de faciliter le transfert de la maintenance du code source à un client ou à une équipe tierce. Mais alors pourquoi développer son propre Framework et réinventer la route ? Tout d'abord cela permet d'avoir plus de souplesse, on peut le modifier, l'adapter et le faire évoluer selon les besoins tout en restant plus léger qu'un Framework existant dont on n'utilisera probablement pas

toutes les fonctions. On décide de l'architecture à adopter et on n'est pas restreint à celle imposé par le Framework que l'on utilise.

Malgré ces avantages réinventer la roue n'est pas toujours la bonne solution, même si les composants développé seront réutilisables au fils des projets un nouveau venu aura du mal en en comprendre les subtilités et mettra donc du temps à s'adapter. De plus il ne pourra pas forcement s'aider d'internet ou de documentations pour avancer. De plus Il faudra gérer vous-même la maintenance, les évolutions et les adaptations aux nouveautés des langages sur lesquels il repose. La compatibilité entre les navigateur par exemple, est un problème qui est …

Cette solution … **TODO**

Si l'envi vous prenais de développer votre propre Framework n'hésitez pas à visiter ce site : http://blogs.infinitesquare.com/b/jonathan/archives/mon-framework-mvvm-a-moi

Utiliser un CDN ou héberger la librairie ? (Supprimer ça ?)

CDN : Content Delivery Network

Avantages : Réduit la latence, augmente le parralélisme, meilleur caching (dépend du CDN utilisé)

Latence : Quand un utilisateur essaye Personne éloignés de votre serveur mettent.

Economie bande passante : pas sur votre serveur

Oui pour un grosse lib (JQuery, AngularJS), probablement non sinon

Src : http://encosia.com/3-reasons-why-you-should-let-google-host-jquery-for-you/

Pas de vainqueurs…

Même si **AngularJS** semble légèrement devant ses concurrents il n'en est pas le vainqueur à l'unanimité. En effet chacun se démarque par son approche différente du sujet. **Knockout** est un bon concurrent grâce à sa légèreté et la concision du code produit. **Amber** avec son système de routage peut être très utile. **TODO**

Il n'y a donc pas de réponse catégorique quant au choix d'un Framework ou d'utiliser son propre Framework, il dépendra avant tout du besoin, du temps et de l'équipe qui participera au projet.