Knockout, Durandal et Angular

Présentation des 3 Frameworks JavaScript pour le binding

Jérôme Romagny

20/07/2014

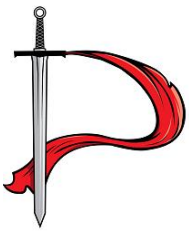


Table des matières

Rappels JavaScript

1..

```
var viewModel = {
    prop: "my property",
    funcA: function () { }
}
return viewModel;
```

2..

```
     var ViewModel = function () {
            var self = this;
            // private
            var privateVariable = "private !";
            var privateFunction = function () {
                return "private !";
            };

            // public
            this.publicVariable = "public !";
            this.publicFunction = function () {
                //var result1 = privateVariable;
                //var result2 = privateFunction();
                //
                var result = self.publicVariable;
                alert(result);
            };
        };

      var viewModel = new ViewModel();
```

3..

```
      var ViewModel = function () {
            // private
            var propA = "my property";
            function funcA() {
            };
            var funcA2 = function () {
            };
            var funcB = function (x,y) {
                return x * y;
            };

            // public
            return {
                propA: propA,
                funcA: funcA,
                funcA2: funcA2,
                funcB:funcB
            }
        };
        // instance
        var viewModel = new ViewModel();
```

- ✓ Seuls les membres « retournés » seront « public » et accessibles
- ✓ Note : si on passe moins de paramètres qu'attendus par une fonction > *undefined*
- ✓ Si on passe plus de paramètres qu'attendus > *ignorés*

4 ..Instanciation immédiate …

```
var ViewModel = function () {
            // …
}();
```

Sans data binding…

On utilise jQuery et ses méthodes val, text, html pour définir dynamiquement le contenu de la page html

```
<body>
    <span id="firstLabel"></span>
    <input id="firstText" type="text" />
    <span id="description"></span>

    <script src="Scripts/jquery-2.1.1.min.js"></script>
    <script>
        var person = {
            firstName: "Marie",
            description : "a <i>nice</i> girl"
        }

        $(function () {
            $("#firstLabel").text(person.firstName);
            $("#firstText").val(person.firstName);
            $("#description").html(person.description);
        });
    </script>
</body>
```

Astuces

Avoir l'IntelliSense avec Visual Studio

Fichier distant

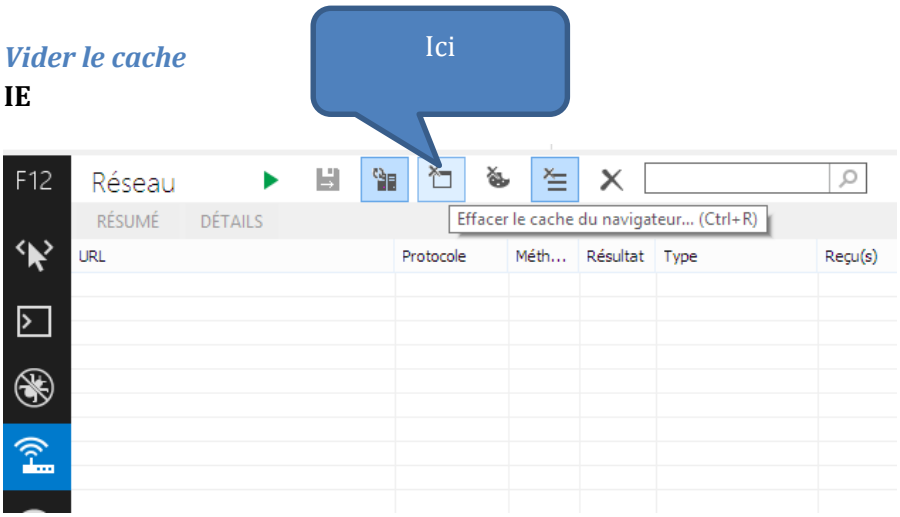
```
/// <reference path="//cdnjs.cloudflare.com/ajax/libs/knockout/3.1.0/knockout-min.js" />
```

Fichier local au projet (glisser déposer depuis l'explorateur de solutions)

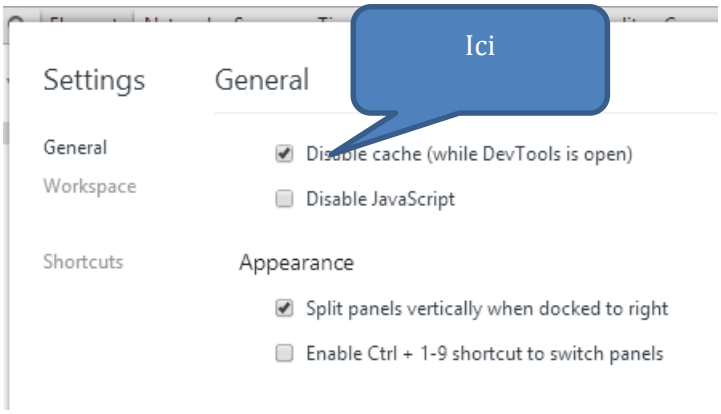
```
/// <reference path="C:\Users\romagny\Documents\Visual Studio 2013\Projects\
DurandalDemo\DurandalDemo\Scripts/knockout-3.1.0.debug.js" />
```

Vider le cache

IE

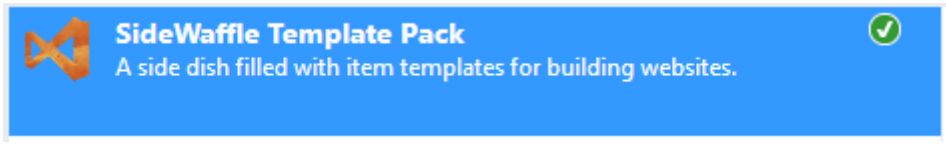


Chrome



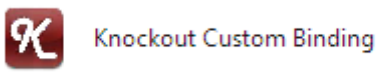
Sidewaffle (templates pour Visual Studio)

Propose de nombreux templates pour **Knockout, Durandal, Angular** (et plus…). Il suffit de télécharger le *.vsix ou depuis Visual Studio chercher « sidewaffle » dans Outils/Extensions et mises à jour/



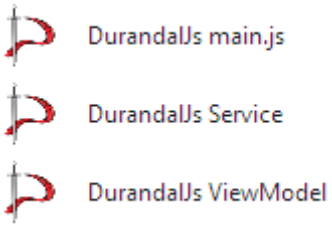
http://sidewaffle.com/

➔ **Knockout**



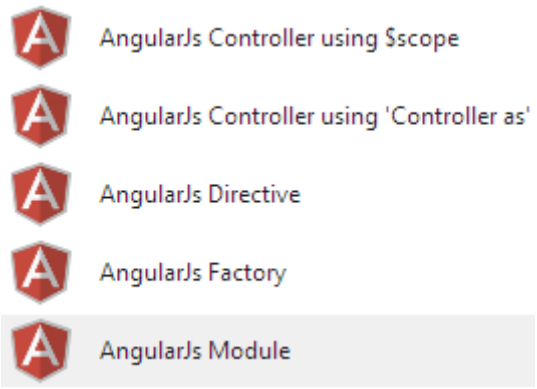
Permet d'ajouter un fichier pour custom binding (bindingHandler)

➔ **Durandal**



Main, service et ViewModel

➔ **Angular**



Contrôleurs(avec $scope et « as »), directives (pour créer une directive personnalisée), factory(pour créer un service) et modules(bootstrapping)

I. Knockout

Framework JavaScript MVVM.

Installation

Téléchargement ou **Package NuGet** depuis Visual Studio

CDN

http://www.asp.net/ajaxlibrary/cdn.ashx
http://cdnjs.com/libraries/knockout

1. Observables

- Les **observables** sont des **fonctions** JavaScript.
- **Two-way** binding

Les observables ne sont pas utiles dans les cas :

- de propriétés readonly
- binding one time
- de charger seulement une liste d'objets

a. Observable

Utile pour les propriétés

```
// viewModel
var vm = {
    name: ko.observable('Marie')
}

vm.name('Deborah'); // write
var name = vm.name(); //read

// apply bindinds
ko.applyBindings(vm);
```

…

```
<span data-bind="text: name" />
<input type="text" data-bind="value: name" />
```

b. *ObservableArray*

Utile pour les collections (exemple : people)

➔ Notification ajout/suppression d'éléments
➔ pas de suivi des changements de valeurs

```
var Person = function (name) {
     this.name = ko.observable(name);
 };

 // viewModel
 var vm = {
     people: ko.observableArray([])
 }

 vm.people.push(new Person('Marie')); // add
 vm.people.push(new Person('patrick'));

 var name = vm.people()[0].name(); //read
 vm.people()[0].name('Deborah'); // write

 vm.people.remove(vm.people()[0]); // remove one
 vm.people.removeAll(); // remove all

 // apply bindinds
 ko.applyBindings(vm);
```

…

```
<ul data-bind="foreach:people">
        <li data-bind="text:name"></li>
    </ul>
```

observableArray methods

indexOff('value')	Index
slice(2,4)	Retourne items entre l'index de début/fin
push('value')	Ajoute un élément à la fin
unshift('value')	Ajout de l'élément au début
pop()	Supprime le dernier élément
remove(item)	Supprime l'élément
removeAll()	Supprime tous les éléments
shift()	Supprime le premier élément
reverse()	Ordre inversé
sort()	Tri

c. *Computed*

- avant se nommait « dependentObservable »
- « Expression qui évalue une autre expression »
- *this* pointer

```
vm = (function () {
     // private
     var firstName = ko.observable('Marie'),
         lastName = ko.observable('Bellin')

     // public
     return {
         firstName: firstName,
         lastName: lastName
     }
 })(); // immediate instanciation

 vm.fullName = ko.computed(function () {
     return this.firstName() + ' ' + this.lastName();
 }, vm)

 ko.applyBindings(vm);
```

…

```
<span data-bind="text:fullName" />
```

d. *Binding html*

On est obligé d'indiquer les parenthèses pour les observables que dans le cas ou c'est un observableArray avec une propriété derrière.

```
<ul data-bind="foreach:people" >
        <!--  -->
</ul>
<span data-bind="text: people().length"></span>
<span data-bind="text: selectedPerson.FirstName"></span>
```

Note : bien sûr on peut ajouter les parenthèses si on le désire, ce n'est qu'une facilité pour le html

2. Built-in bindings

Texte et apparence

visible	Rend visible/invisible
text	Valeur de texte (en lecture seule) .Note ne peut pas être appliqué à des contrôles permettant la saisie (tels que input de type text)
html	Met en forme du contenu html
css	Css class
style	Style css
attr	Attribut (exemple src pour indiquer la source d'une image)

http://knockoutjs.com/documentation/visible-binding.html

Forms

click	Handler sur event click
event	Event (exemple mouseover)
submit	Form submitted
enable	Rend accessible un élément selon une condition
disable	Rend inaccessible
value	Texte en lecture/écriture
checked	Pour checkbox/bouton radio
options	Pour dropdown/select
selectedOptions	Eléments sélectionnés courants d'une dropdown/select
uniqueName	Name attribute

Control flow

if	condition
ifnot	
foreach	boucle
with	Pour l'élément spécifié

http://knockoutjs.com/documentation/foreach-binding.html

a. Templates

http://knockoutjs.com/documentation/template-binding.html

Note : il est conseillé d'utiliser un template par « ligne » plutôt qu'un template pour l'ensemble d'une liste.

✓ Avec jQuery Templates

```
<ul data-bind="template: { name: 'itemTmpl',foreach: people}"></ul>

<script id="itemTmpl" type="text/html">
     <li>${name}</li>
</script>

<script src="//ajax.googleapis.com/ajax/libs/jquery/2.1.1/jquery.min.js"></script>
<script src="Scripts/jquery.tmpl.js"></script>
<script src="//ajax.aspnetcdn.com/ajax/knockout/knockout-3.1.0.js"></script>
```

✓ Knockout Template

```
<ul data-bind="template: { name: 'itemTmpl',foreach: people}"></ul>

<script id="itemTmpl" type="text/html">
   <li data-bind="text: name"></li>
</script>
```

Sans script…

```
<div data-bind="template: { name: 'itemTmpl',foreach: people}"></div>

<div id="itemTmpl">
   <span data-bind="text: name"></span><br/>
</div>
```

✓ Avec commentaires

```
    <ul>
        <!-- ko foreach:people -->
        <li data-bind="text: name"></li>
        <!-- /ko -->
    </ul>
```

✓ Enfin …

```
     <ul data-bind="foreach: people">
           <li data-bind="text: name"></li>
     </ul>
```

Choix Template

Exemple

```
<body>
    <table>
        <thead>
            <tr>
                <td>Name</td>
                <td></td>
            </tr>
        </thead>
        <tbody data-bind="template: { name: templateChoice,foreach: people}" />
    </table>

    <script id="itemTmpl" type="text/html">
        <tr><td><span data-bind="text: name"></span><td><button
class="edit">Edit</button></td></tr>
    </script>
    <script id="editTmpl" type="text/html">
        <tr>
            <td><input type="text" data-bind="value: name" /></td>
            <td><button data-bind="click:$root.save">Save</button></td>
        </tr>
    </script>


    <script src="//ajax.googleapis.com/ajax/libs/jquery/2.1.1/jquery.min.js"></script>
    <script src="//ajax.aspnetcdn.com/ajax/knockout/knockout-3.1.0.js"></script>

    <script>
        var Person = function (name) {
            this.name = ko.observable(name);
        };

        // viewModel
        var vm = {
            people: ko.observableArray([]),
            selectedPerson: ko.observable(),
            showDetails:ko.observable(false),
            templateChoice: function () {
                return vm.showDetails() ? "editTmpl" : "itemTmpl";
            },
            save: function () {
                // save ..
                vm.showDetails(false);
            }
        }

        $('table').on('click', '.edit', function () {
            vm.showDetails(true);

        });

        vm.people.push(new Person('Marie')); // add
        vm.people.push(new Person('patrick'));

        // apply bindinds
        ko.applyBindings(vm);

    </script>
</body>
```

b. *Binding Context*

$data	Item courant
$parent	item parent
$parents	Tous les parents
$root	Element au top(ViewModel)

with	Définit un contexte
	`<div data-bind="with: selectedPerson">`
	`<span data-bind="text:name"/>`
	`</div>`

documentation binding contextl

c. *Ko.dataFor()*

Permet de récupérer l'élément courant.

Utile pour les listes. En effet, il est préférable de récupérer l'évènement « click » sur l'élément plutôt que de lier chaque ligne avec « data-bind = "click :methode" »

Exemple

```
$("table").on("click", ".edit", function () {
        var person = ko.dataFor(this);
        app.ViewModel.editPerson(person);
});
```

3. Les 3 patterns pour ViewModels

a. *Object Literals*

```
$(function () {

   // namespace
   var my = my || {};

    // The ViewModel
    my.vm = {
        myProp: ko.observable(),
        myArray: ko.observableArray([]),
        myFunc: function () {

        }
    };

    // …
    my.vm.myFunc();
    ko.applyBindings(my.vm);
});
```

b. Module Pattern

```
$(function () {

   // namespace
   var my = my || {};

    // The ViewModel
    my.vm = (function () {
        // private
        var myProp = ko.observable();
        var myArray = ko.observableArray([]);
        var myFunc = function () {

        };

        // public
        return {
            myProp: myProp,
            myArray: myArray,
            myFunc: myFunc
        }
    })(); // immediate instanciation

    // ….
    my.vm.myFunc();
    ko.applyBindings(my.vm);
});
```

c. Revealing Module Pattern

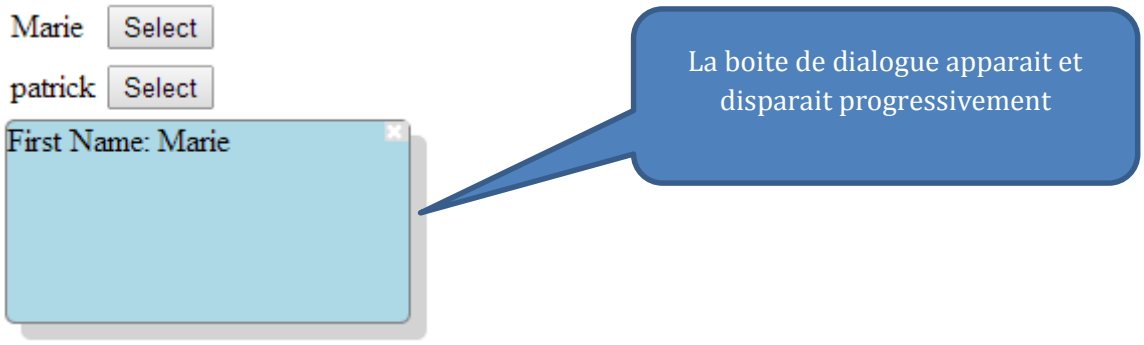
```
$(function () {

   // namespace
   var my = my || {};

    // The ViewModel
    my.vm = (function () {
        // private
        var myProp = ko.observable(),
        myArray = ko.observableArray([]),
        myFunc = function () {

        }

        // public
        return {
            myProp: myProp,
            myArray: myArray,
            myFunc: myFunc
        };
    })(); // immediate instanciation

    // …
    my.vm.myFunc();
    ko.applyBindings(my.vm);
});
```

4. Custom bindings

documentation custom bindings

- **init** est appelé uniquement une fois à l'initialisation
- **update** est appelé à chaque changement de valeur de l'accesseur

Exemple On affiche la personne sélectionnée dans une boite de dialogue



bindingHandler

```
ko.bindingHandlers.fadeVisible = {
        init: function (element, valueAccessor) {
            $(element).toggle(valueAccessor());
        },
        update: function (element, valueAccessor, allBindingsAccessor) {
            var shouldDisplay = valueAccessor();
            shouldDisplay ? $(element).fadeIn(2000) : $(element).fadeOut(2000);
        }
    };
```

```
<div class="dialog" data-bind="fadeVisible: $root.canShowDetails() ? true:false">
    <div data-bind="with:selectedPerson">
        <button data-bind="click:$parent.closeDetails"></button>
        <span>First Name: </span><span data-bind="text: name"></span>
    </div>
</div>
```

- ✓ Le ViewModel a un observable canShowDetails qui est vrai si on sélectionne une personne
- ✓ En cliquant sur le bouton de la boite de dialogue on passe canShowDetails à false.
- ✓ Important : il faut garder l'élément sélectionné pour ne pas gâcher l'effet

ViewModel

```
var Person = function (name) {
        this.name = ko.observable(name);
    };

    // viewModel
    var vm = {
        people: ko.observableArray([]),
        selectedPerson: ko.observable(),
        canShowDetails: ko.observable(false),
        closeDetails: function () {
            vm.canShowDetails(false);
        }
    }

    $('table').on('click', '.person', function () {
        var person = ko.dataFor(this);
        vm.selectedPerson(person);

        vm.canShowDetails(true);
    });

    vm.people.push(new Person('Marie')); // add
    vm.people.push(new Person('patrick'));


    // apply bindinds
    ko.applyBindings(vm);
```

Autre exemple : Pour valider une saisie lorsque l'utilisateur presse « enter »

5. Ko utilities

Array utilities

ko.utils.arrayFilter
ko.utils.arrayFirst
ko.utils.arrayForEach
ko.utils.arrayIndexOf
ko.utils.arrayMap
ko.utils.arrayPushAll
ko.utils.arrayRemoveItem
ko.utils.compareArrays
ko.utils.unwrapObservable

6. Data Features

- Ko.toJS() renvoie un objet javascript sans binding
- Ko.toJSON() renvoie une chaine au format Json

Dans le « **dataService** » utiliser *ko.toJSON()*

```
var addPerson = function (person) {
        return $.ajax({
            url: urlBase,
            type: 'POST',
            dataType: 'json',
            contentType: 'application/json; charset=utf-8',
            data: ko.toJSON(person),
        });
    };
```

7. Subscribe

```
search.subscribe(function (search) {
            if (search)
                searchPeople(search);
            else
                getPeople();
//
        });
```

8. Extenders

documentation extenders

9. Change tracker

J'utilise ici la librairie de John Papa pour traquer les changements .L'intérêt est de détecter lors de l'édition d'un formulaire par exemple si les informations ont été modifiées.

```
function changeTracker(objectToTrack, hashFunction) {
    hashFunction = hashFunction || ko.toJSON;
    var lastCleanState = ko.observable(hashFunction(objectToTrack));

    var result = {
        somethingHasChanged: ko.computed(function () {
            return hashFunction(objectToTrack) != lastCleanState()
        }),
        markCurrentStateAsClean: function () {
            lastCleanState(hashFunction(objectToTrack));
        }
    };

    return function () { return result }
};
```

Exemple je suis les modifications sur la personne sélectionnée(les propriétés de « selectedPerson » sont des observables)

```
vm.tracker = new changeTracker(selectedPerson) ;
```

- On peut reset les changements avec **markCurrentStateAsClean()**
  `tracker().markCurrentStateAsClean();`
- Et savoir si l'objet traqué a été modifié grâce à **somethingHasChanged.**
  Exemple un bouton accessible qu'après des modifications.
  `<button data-bind="click: $root.save,enable:$root.tracker().somethingHasChanged">Save</button>`

10. Ko Validation

a. *Avec ko.extenders*

```
ko.extenders.required = function (target, overrideMessage) {
    target.hasError = ko.observable();
    target.validationMessage = ko.observable();

    // check if value is empty
    function validate(newValue) {
        target.hasError(newValue ? false : true);
        target.validationMessage(newValue ? "" : overrideMessage || "This field is
required");
    }

    validate(target());
    target.subscribe(validate);

    return target;
};
```

Afficher les erreurs

```
<p>
  <input data-bind="value: Twitter,validationOptions: { errorElementClass: 'input-
validation-error' }" type="text" class="form-control" placeholder="Enter twitter" />
  <span data-bind='visible: Twitter.hasError, text: Twitter.validationMessage' />
</p>
```

On peut également modifier la feuille de style

```
.input-validation-error {
    border: 1px solid #e64343;
    box-shadow: 0 0 2px 0 rgba(230, 67, 67, 0.4);
}

    .input-validation-error:focus {
        background: #fcecec;
        border: 1px solid #e64343;
        box-shadow: 0 0 2px 0 rgba(230, 67, 67, 0.4);
    }

.validationMessage {
    color: Red;
}
```

b. *Avec le plugin ko.validation*

- Extend
- Validation group

```
var Person = function (id, firstName, lastName, twitter, isNew) {
    this.Id = id;
    this.FirstName = ko.observable(firstName).extend({ required: true });
    this.LastName = ko.observable(lastName).extend({ required: true });
    this.Twitter = ko.observable(twitter).extend({
        pattern: {
            message: 'Enter a valide Twitter',
            params: '^@([A-Za-z0-9_]+)'
        },
        required: { message: 'Enter the person\'s Twitter' }
    });
    this.IsNew = isNew;

    this.errors = ko.validation.group(this);
}
```

Configuration

```
  ko.validation.configure({
                registerExtenders: true,
                messagesOnModified: true,
                insertMessages: true,
                parseInputAttributes: true,
                messageTemplate: null,
                decorateElement: true
            });
```

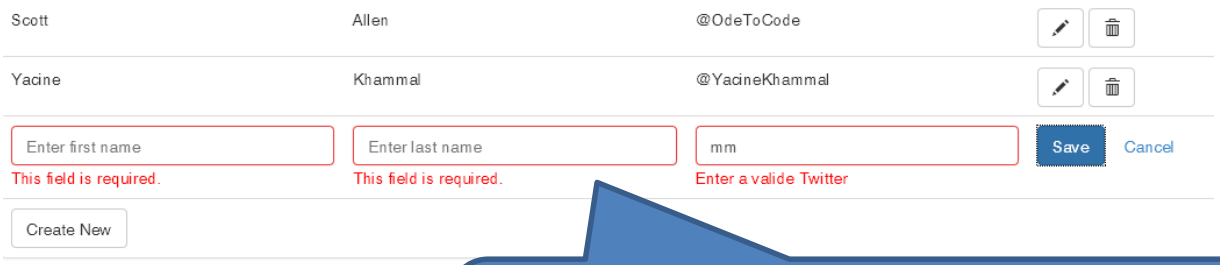
Test : exemple lors de la validation de données utilisateurs saisies

```
            if (selectedPerson().errors().length > 0) {
                selectedPerson().errors.showAllMessages();
                return;
             }
```

Afficher les erreurs

```
<input data-bind="value: Twitter,validationOptions: { errorElementClass: 'input-
validation-error' }" type="text" class="form-control" placeholder="Enter twitter" />
```

Inutile d'ajouter un champ pour « validationMessage », le plugin le génère



Lors de la création d'une personne la vérification se fait à la validation (bouton « save »).

Lors de l'édition d'une personne la validation se fait après modification des champs

11. Plus ...

Boilerplate une solution complète SPA

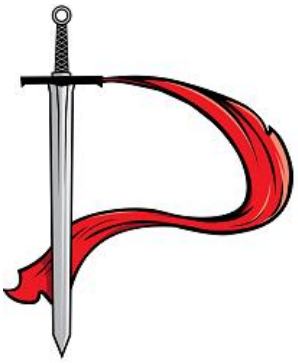
Sammy pour la navigation/historique SPA

Qunit pour les tests

http://www.responsinator.com/ pour tester son site dans toutes tailles/orientations d'écrans

Breeze ou amplify offrent des facilités pour le développement de SPA

II. Durandal

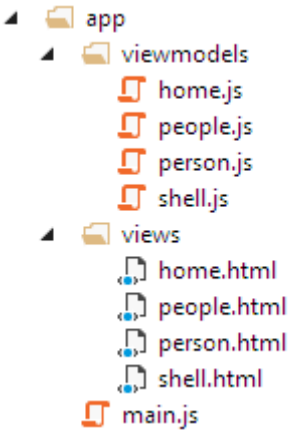


Durandal utilise :

- **jQuery** (DOM)
- **Require** (SoC, enregistrement et chargement de modules)
- **Knockout** (Binding)

Installation : avec **Package NuGet** depuis Visual Studio + **Durandal Transtions**

- ✓ Les librairies sont installées dans le répertoire « **Scripts** »
- ✓ On range ses fichiers dans un répertoire « **app** »



Note : il existe également des templates SPA

Convention :

- correspondance nom page html, fichier js (exemple : home.html et home.js)
- pour mieux organiser son code :views rangées dans un répertoire « views » et viewmodels dans répertoire « viewmodels »

Page de départ index

(*.html ou *.cshtml pour Mvc)

- ✓ Réfrencer jQuery, knockout, require, etc. Pour require indiquer le chemin vers « main »

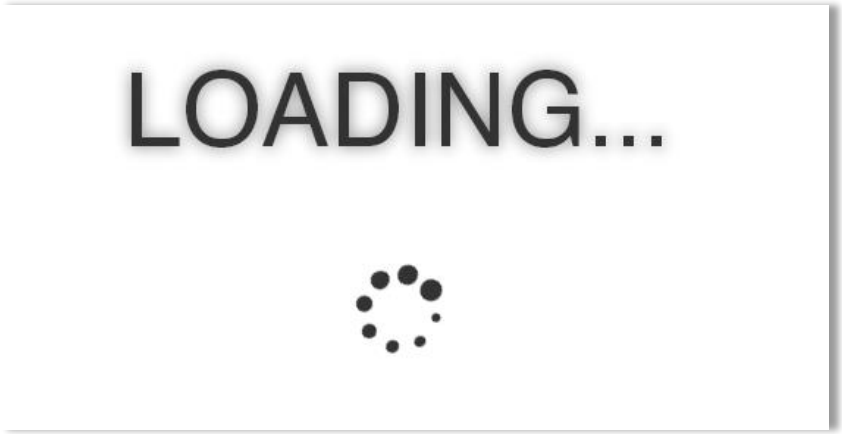
```
<script src="//ajax.googleapis.com/ajax/libs/jquery/2.1.1/jquery.min.js"></script>
<script src="Scripts/knockout-3.1.0.js"></script>
<!-- … main -->
<script src="Scripts/require.js" data-main="app/main"></script>
```

- ✓ **applicationHost** (id recherché par Durandal pour le shell)

```
<div id="applicationHost"></div>
```

Avec splashscreen (utilisation de bootsrap et font awesome)

```
<div id="applicationHost">
        <div class="splash">
            <div class="message">
                Loading...
            </div>
            <i class="fa fa-spinner fa-spin"></i>
        </div>
    </div>
```



Exemple de page index.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Durandal demo</title>
    <link rel="stylesheet"
href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap.min.css" />
    <link href="Content/font-awesome.min.css" rel="stylesheet" />
    <link href="Content/durandal.css" rel="stylesheet" />
    <style>
        .splash {
            text-align: center;
            margin: 10% 0 0 0;
        }

            .splash .message {
                font-size: 5em;
                line-height: 1.5em;
                -webkit-text-shadow: rgba(0, 0, 0, 0.5) 0 0 15px;
                text-shadow: rgba(0, 0, 0, 0.5) 0 0 15px;
                text-transform: uppercase;
            }

            .splash .fa-spinner {
                text-align: center;
                display: inline-block;
                font-size: 5em;
                margin-top: 50px;
            }

        .page-host {
            position: absolute;
            left: 0;
            right: 0;
            top: 50px;
            bottom: 0;
            overflow-x: hidden;
            overflow-y: auto;
        }
    </style>
</head>
<body>
    <div id="applicationHost">
        <div class="splash">
            <div class="message">
                Loading...
            </div>
            <i class="fa fa-spinner fa-spin"></i>
        </div>
    </div>

    <script
src="//ajax.googleapis.com/ajax/libs/jquery/2.1.1/jquery.min.js"></script>
    <script src="Scripts/knockout-3.1.0.js"></script>

    <script src="Scripts/require.js" data-main="app/main"></script>
</body>
</html>
```

1. Main

- **useConvention** > correspondance noms views et viewmodels
- **setRoot** définit le « shell » (+ la transition possible)

Require.js :

Les chemins (paths) représentent des « raccourcis » que l'on pourra utiliser pour localiser les fichiers + urlBase

- ✓ **define** pour définir un module (nom, modules requis)
- ✓ **require** pour indiquer les modules requis/utilisés par le module/fonction
- ✓ **requirejs** pour configurer les options…

```
// définit des chemins réutilisables par "raccourcis"
requirejs.config({
    paths: {
        'text': '../Scripts/text',
        'durandal': '../Scripts/durandal',
        'plugins': '../Scripts/durandal/plugins',
        'transitions': '../Scripts/durandal/transitions',
        'services': '../app/services',
    }
});

define('jquery', function () { return jQuery; });
define('knockout', ko);

define(['durandal/system', 'durandal/app', 'durandal/viewLocator'], function (system,
app, viewLocator) {
    //>>excludeStart("build", true)
    system.debug(true);
    //>>excludeEnd("build");

    app.title = 'Durandal demo';

    app.configurePlugins({
        router: true,
        dialog: true,
        widget: true,
        observable: true
    });

    app.start().then(function () {

        viewLocator.useConvention();

        app.setRoot('viewmodels/shell', 'entrance');
    });
});
```

2. Shell

On définit les routes, ainsi que les fonctions accessibles pour toutes les pages.

```
define(['plugins/router', 'durandal/app'], function (router, app) {
    return {
        router: router,
        search: function () {
            app.showMessage('Search not yet implemented...');
        },
        activate: function () {
            router.map([
                { route: ['home', ''], title: 'Home page', moduleId:
'viewmodels/home', nav: true },
                { route: 'people', title: "People page", moduleId:
'viewmodels/people', nav: true },
                 { route: 'person/:id', title: 'person', moduleId:
'viewmodels/person', nav: false }
            ]).buildNavigationModel();

            return router.activate();
        }
    };
});
```

```
<div>
<!-- barre de navigation -->
    <nav class="navbar navbar-default navbar-fixed-top" role="navigation">
        <div class="navbar-header">
            <button type="button" class="navbar-toggle" data-toggle="collapse">
                <span class="sr-only">Toggle Navigation</span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
            </button>
            <a class="navbar-brand" href="#">
                <i class="fa fa-home"></i>
                <span>Durandal</span>
            </a>
        </div>
        <div class="collapse navbar-collapse">
            <ul class="nav navbar-nav" data-bind="foreach: router.navigationModel">
                <li data-bind="css: { active: isActive }">
                    <a data-bind="attr: { href: hash }, text: title"></a>
                </li>
            </ul>
            <ul class="nav navbar-nav navbar-right">
                <li class="loader" data-bind="css: { active: router.isNavigating }">
                    <i class="fa fa-spinner fa-spin fa-2x"></i>
                </li>
            </ul>
            <form class="navbar-form navbar-right" role="search" data-
bind="submit:search">
                <div class="form-group">
                    <input type="text" class="form-control" placeholder="Search">
                </div>
            </form>
        </div>
    </nav>
    <!-- navigation -->
    <div class="page-host" data-bind="router: { transition:'entrance' }"></div>
</div>
```

3. Pages

« Couple view /viewmodel »

```
define(function () {

    return {
        activate: function () {

        }
    };
});
```

On indique uniquement les modules dont a besoin

```
define(['services/dataService', 'services/model', 'plugins/router'],
    function (dataService, model, router) {
        var people = ko.observableArray();
        var selectedPerson = ko.observable();
        var initialized = false;

        var vm = {
            activate: activate,
            people: people,
            router: router,
            getPeople: function () {
                dataService.getPeople()
                    .done(makePeople)
                    .fail(function (jqXHR, textStatus, err) {
                        alert('Unable to load people : ' + textStatus);
                    });
            },
            GoToPerson: function (person) {
                var url = '#/person/' + person.Id;
                router.navigate(url);
            }
        };

        function activate() {
            if (initialized) {
                return;
            }
            initialized = true;
            return vm.getPeople();
        };

        function makePeople(allPeople) {
            people([]);
            var temp = people();
            allPeople.forEach(function (person) {
                var newPerson = new model.Person(person.Id, person.FirstName,
person.LastName, person.Twitter, false);
                temp.push(newPerson);
            });
            people.valueHasMutated();
        };

        return vm;
    });
```

Views : Ne pas indiquer les balises html , head

4. Navigation

Routes (définies dans shell.js ou un fichier de config à part) :

- *route* c'est la route affichée exemple si on définit « **people** » on aura **http://localhost:15597/#people** . La route de la page index a une url `''` ou `['home', '']`
- *moduleId* chemin du viewmodel exemple « **viewmodels/people** »
- *title* le **texte** affiché dans la **barre de navigation**
- *nav* : indique si la route sera affichée dans la barre de navigation
- hash : exemple #people

Exemples

```
router.map([
                { route: ['home', ''], title: 'Home page', moduleId:
'viewmodels/home', nav: true },
                { route: 'people', title: "People page", moduleId:
'viewmodels/people', nav: true },
                 { route: 'person/:id', title: 'person', moduleId:
'viewmodels/person', nav: false }
            ]).buildNavigationModel();
```

- **mapUnknownRoutes**

```
router.mapUnknownRoutes('viewmodels/catalog', "#catalog");
```

- **goBack** (pour le retour en arrière dans la navigation

```
goBack: function () {
    router.navigateBack();
},
```

Passage de paramètres

La route définie

```
{ route: 'person/:id', title: 'person', moduleId: 'viewmodels/person', nav: false }
```

- Si le paramètre était entre parenthèses, celui-ci ne serait pas obligatoire. La route serait person(/ :id)
- on peut utiliser **encodeURIComponent()** pour le paramètre passé

La méthode du ViewModel permettant la navigation vers la page détails de la personne

```
var vm = {
            // ...
            goToPerson: function (person) {
                var url = '#/person/' + person.Id;
                router.navigate(url);
            }
        };
```

Un bouton permettant de déclencher la navigation

```
<button data-bind="click:  $root.goToPerson"></button>
```

Récupération du paramètre dans « **activate** » et « **canActivate** » de la page appelée

```
  var vm = {
        activate: function (id) {
            dataService.getPerson(id)
                .done(function (person) {
                    selectedPerson(new model.Person(person.Id, person.FirstName,
person.LastName, person.Twitter, false));
                })
        },
        goBack: function () {
            router.navigateBack();
        },
        selectedPerson: selectedPerson
    }
```

5. Logger

Utilisant [Toastr]

```
define(['durandal/system'],
    function (system) {
        var logger = {
            log: log,
            logError: logError
        };

        return logger;

        function log(message, data, source, showToast) {
            logIt(message, data, source, showToast, 'info');
        }

        function logError(message, data, source, showToast) {
            logIt(message, data, source, showToast, 'error');
        }

        function logIt(message, data, source, showToast, toastType) {
            source = source ? '[' + source + '] ' : '';
            if (data) {
                system.log(source, message, data);
            } else {
                system.log(source, message);
            }
            if (showToast) {
                if (toastType === 'error') {
                    toastr.error(message);
                } else {
                    toastr.info(message);
                }

            }

        }
    });
```

III. Angular

https://angularjs.org/ Api Documentation

Framework JavaScript Open Source développé par Google. Facilite le développement de SPA (Single page Application). AngularJS s'appuie sur le pattern MVC et MVVM (pour le binding), l'injection de dépendances (IoC) et aide à la navigation, offre des Services, etc.

Installation

➔ **Local** On peut télécharger les sources sur le site https://angularjs.org/ (et ajouter à un dossier « Scripts ») ou avec les packages **NuGet** depuis Visual Studio
Référencer dans l'en-tête de la page

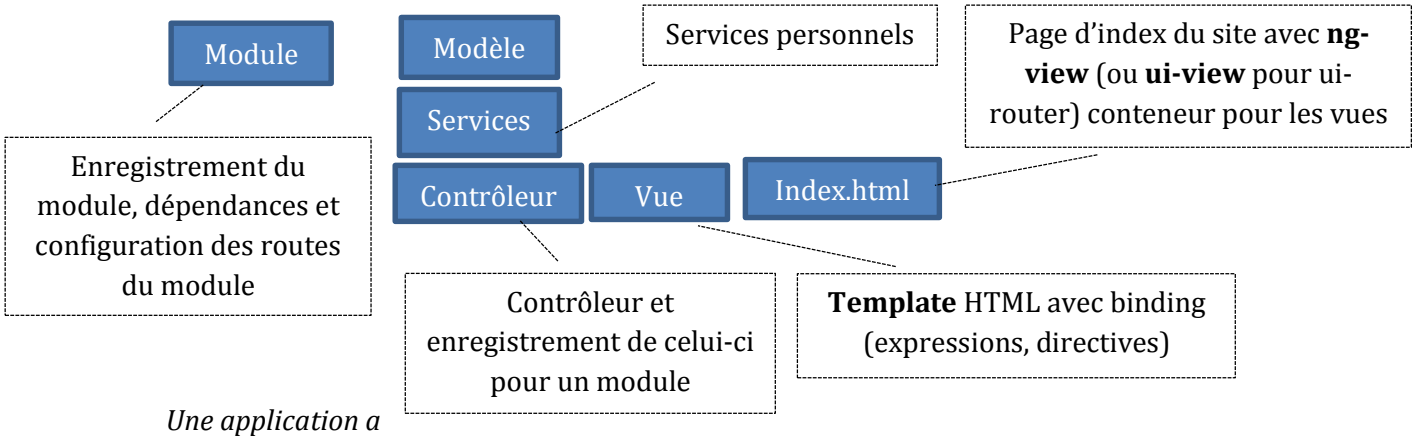
```
<script src="Scripts/angular.js"></script>
```

➔ **CDN** Google , cdnjs

```
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.js"></script>
```

On peut utiliser la version **minimisée « angular.min.js »**

MVC/ MVVM



Une application a

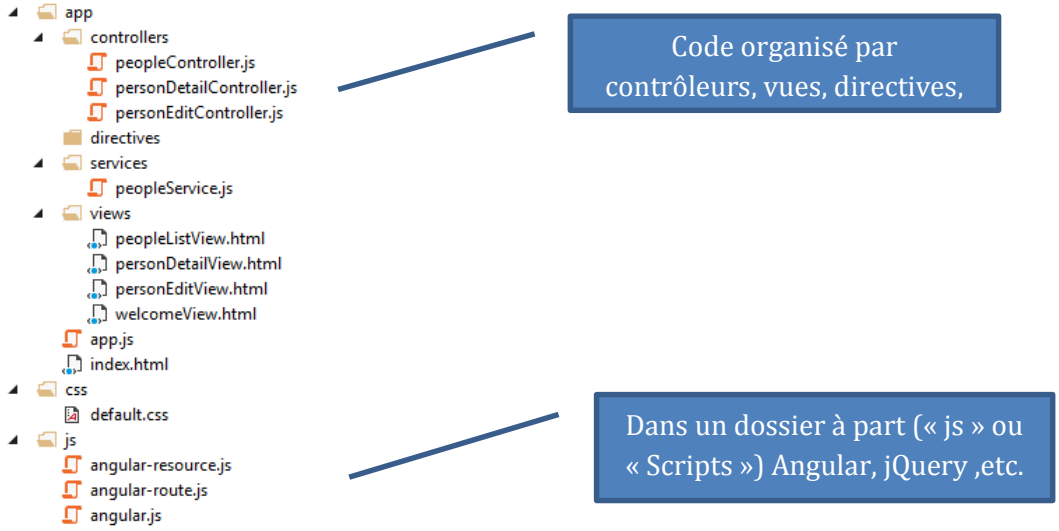
- *1 ou plusieurs modules*
  - *Chaque module a x contrôleurs*
    - *Chaque vue (template HTML) est reliée à un contrôleur et en affiche les données (data binding).*

La page index affiche les vues dans un conteneur.

Organisation du code

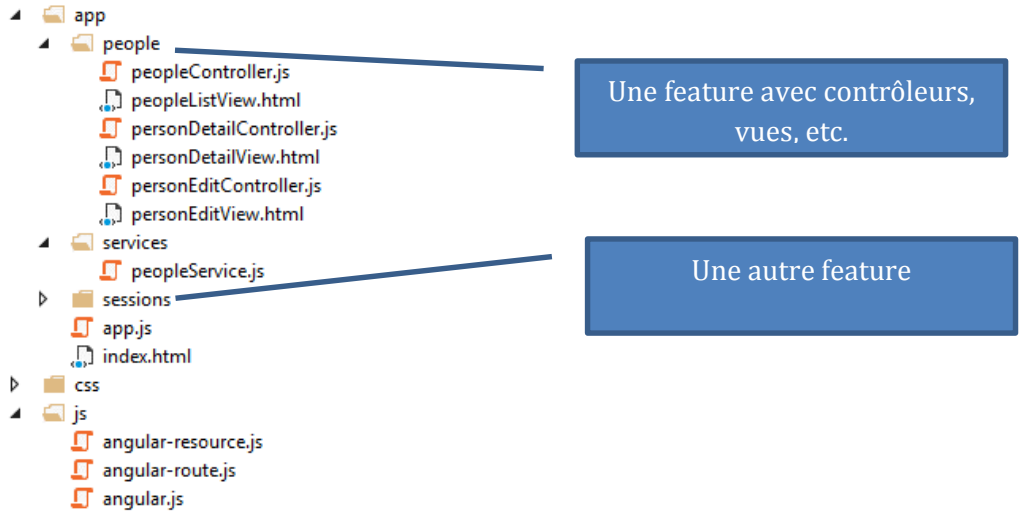
Par type

Convient aux applications avec un seul module



Par feature

Plus adaptée pour les applications avec plusieurs modules/ features



On peut même mélanger les 2 organisations (exemple une feature « people » qui est divisée en dossiers « controllers » , « views », « services »).

Conventions de nommage

PascalCase pour les noms de contrôleurs avec un suffixe « Ctrl » ou « Controller »
camelCase pour les directives, services, etc.

1. Module

Enregistrement de module avec la méthode « module » de l'objet « angular ». On donne un nom au module et on ajoute les dépendances.

```
(function () {
    'use strict';

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

Bonne pratique .. IIFE (Immediately-invoked function expression) + 'use strict'

Obtenir le module (par son nom)

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

Enregistrement du module (nommé « app ») + dépendances avec la méthode module de l'objet

Ex de module « app »

```
(function () {
    'use strict';

    // 1 enregistrement du module
    var app = angular.module('app', ['ngRoute']);

    // 2 Routes
    app.config(function ($routeProvider) {
        $routeProvider
            .otherwise({ redirectTo: "/" })
            .when("/", { templateUrl: 'app/welcomeView.html' })
            .when("/people/", { templateUrl: 'app/people/peopleListView.html',
controller: 'PeopleController' })
            .when("/person/:id", { templateUrl: 'app/people/personDetailView.html',
controller: 'PersonDetailController' })
            .when("/person/edit/:id", { templateUrl: 'app/people/personEditView.html',
controller: 'PersonEditController' })
    });

})();
```

Configuration des routes du module. On peut ou non indiquer le contrôleur, templateUrl pour le chemin vers le template, la vue

a. Dépendances de module :

- angular.js
- angular-animate.js
- angular-aria.js
- angular-cookies.js
- angular-loader.js
- angular-messages.js
- angular-mocks.js
- angular-resource.js
- angular-route.js
- angular-sanitize.js
- angular-scenario.js
- angular-touch.js

Par exemple si on veut utiliser « ngRoute » (pour la navigation) :

- On référence « angular-route.js » dans la page d'index
- On ajoute la dépendance « ngRoute » lors de l'enregistrement du module (« app ») et on injecte $routeProvider.
- On peut ensuite utiliser les services dans les contrôleurs du module en injectant $routeProvider

b. *Routing du module*

2 Possibilités : avec ngRoute ou ui-router

1. ngRoute

On a besoin de « **angular-route.js** » que l'on référence dans la page d'index.

```
<script src="js/angular-route.js"></script>
```

.. Puis on ajoute la dépendance « **ngRoute** » au module.

...Enfin on utilise **$routeProvider** pour définir les différentes routes .On peut également créer un fichier de configuration (exemple « config-route ») dédié à cette tâche.

```
(function () {
    'use strict';

    var app = angular.module('app', ['ngRoute']);

    app.config(function ($routeProvider) {
        $routeProvider
            .when("/", {
                templateUrl: 'app/people/peopleListView.html',
                controller: 'PeopleController as vm'
            })
            .when("/person/:id", {
                templateUrl: 'app/people/personDetailView.html',
                controller:'PersonDetailController as vm'
            })
        .otherwise({ redirectTo: "/" });
    })
})();
```

Injection de **ngRoute** et utilisation de **$routeProvider** pour configurer les routes du module

Route par défaut. Cela pourrait être une « welcome » page aussi.

Route avec paramètre.

On définit le chemin du template de la vue correspondante

On ajoute « as vm » pour un contrôleur AS, sinon on ne met que le nom du contrôleur

Redirection vers la page par défaut

Exemple de route :

- ✓ **templateUrl :** le chemin vers le template de la page
- ✓ **controller :** le contrôleur associé. Facultatif, on peut aussi le définir avec « ng-controller » dans la page.

```
    $routeProvider.when("/", {
                templateUrl: 'app/people/people.html',
                controller: 'PeopleController as vm'
            })
```

"/" est la route utilisée par défaut. Cela donnera quelque chose comme « http://localhost:7090/default.html#/ » dans la barre d'adresse.
Pour une autre page on pourrait indiquer "/prices/" par exemple. Ce qui donnerait « http://localhost:7090/default.html#/prices/ » dans la barre d'adresse.

Resolve pour exécuter une action avant la navigation (documentation)

Route avec passage de paramètres

```
        $routeProvider.when("/person/:id", {
                templateUrl: 'app/people/person.html',
                controller: 'PersonController as vm'
            })
```

✓ **Id** est le paramètre passé.

- Si on avait plusieurs paramètres à passer … /person/ :id/ :name
- Pour un nouvel élément à créer on peut passer « 0 » dans un lien et faire un test à la réception du paramètre `<a href="#/person/0">Ajouter</a>`
- Les contrôleurs récupérant des **paramètres passés** utilisent **$routeParams**

Activer une route

- Depuis une **vue** : on indique « # » suivi de l'url et des paramètres de la route définie dans le module

```
<a href="#/">Accueil</a>
```

```
<a href="#/people/">Liste des personnes</a>
```

Avec paramètres

```
<a href="#/person/1">Détails</a>
<a href="#/person/{{person.Id}}">Détails</a>
```

représentant l'url du site

url définie pour la vue

La valeur passée

- Depuis un **contrôleur** : avec **$location.** On injecte $location dans le contrôleur puis on utilise la méthode path pour être redirigé vers une autre vue

```
$location.path("/people/");
```

Depuis la vue il suffit d'appler la méthode du contrôleur redirigeant

```
<button ng-click="goPeopleList()">Liste des personnes</button>
```

Avec **paramètre**

```
$location.path("/person/" + $scope.person.Id);
```

On peut également passer un paramètre depuis la vue

```
<button ng-click="getPerson(person)">{{person.Name}}</button>
<button ng-click="getPerson(person.Id)">{{person.Name}}</button>
```

➔ Contrôleur récupérant un paramètre passé utilise **$routeParams** que l'on nject

```
(function () {
    'use strict';

    angular.module('app').controller('PersonDetailController', ['$routeParams',
'peopleService', personDetailController]);

    function personDetailController($routeParams, peopleService) {
        var vm = this;
        vm.title = "Détails";
        vm.person = {};
        var cache = null;

        var getPerson = function (id) {
            peopleService.getPerson(id).then(function (response) {
                vm.person = response.data;
            })
        }
        activate();
        function activate() {
            getPerson($routeParams.id);
        }
    }
})();
```

Injection de $routeParams pour récupérer les paramètres

Récupération de l'id défini lors de la configuration des routes dans le module (« /person/:id »)

2. ui-router

Télécharger ui-router https://github.com/angular-ui/ui-router

Problèmes connus avec ui-view et ng-class/ng-show/ng-hide.

Ajouter « angular-ui-router.js » (ou version minimisée) à son projet et le référencer

```
<script src="js/angular-ui-router.js"></script>
```

Ajout de la dépendance au module et configuration des « états »

```
(function () {
    "use strict";
    var app = angular.module("app", ["ui.router", "ngResource"]);

    app.config(["$stateProvider",
                "$urlRouterProvider",
        function ($stateProvider, $urlRouterProvider) {
            $urlRouterProvider.otherwise("/");

            $stateProvider
                .state("home", {
                    url: "/",
                    templateUrl: "app/welcomeView.html"
                })
                .state("peopleList", {
                    url: "/people",
                    templateUrl: "app/people/peopleListView.html",
                    controller: "PeopleController as vm"
                })
                .state("personDetail", {
                     url: "/person/:id",
                     templateUrl: "app/people/personDetailView.html",
                     controller: "PersonDetailController as vm",
                     resolve: {
                         person: ['peopleService', '$stateParams', function
(peopleService, $stateParams) {
                             var id = $stateParams.id;
                             return peopleService.getPerson(id).then(function
(response) {
                                 var result  = response.data;
                                 return result;
                             });
                         }]
                     }
                 })
        }]
    );
}());
```

Etats avec le nom de l'état (Ex « home »), l'url, le chemin vers le template de la vue (templateUrl), le contrôleur et resolve le code exécuté avant

Important avec ui-router on utilise ui-view à la placve de ng-view pour indiquer le conteneur recevant les vues dans la page index.html

```
<div ui-view></div>
```

On indique le nom de l'état avec ui-sref

```
<td><a ui-sref="home">Accueil</a></td>
```

➔ Avec Paramètre

Dans la **Vue**

```
<td><a ui-sref="personDetail({id:person.Id})">{{person.Name}}</a></td>
```

Etat dans le **module**

```
            .state("personDetail", {
                url: "/person/:id",
                templateUrl: "app/people/personDetailView.html",
                controller: "PersonDetailController as vm",

                resolve:{
                    person : function () {
                        var result = new person(1, "a", "b");
                        return result;
                    }
            }
            })
```

Contrôleur récepteur

```
(function () {
    'use strict';

    angular.module('app').controller('PersonDetailController', ["person",
personDetailController]);

    function personDetailController(person) {
        var vm = this;

        vm.title = "Détails de la personne";
        vm.person = person;
    }
})();
```

Autre exemple avec **$http**

```
.state("personDetail", {
                url: "/person/:id",
                templateUrl: "app/people/personDetailView.html",
                controller: "PersonDetailController as vm",
                resolve:{
                person: ['$http', '$stateParams',function ($http, $stateParams) {

                    var id = $stateParams.id;
                    return $http.get("/api/People/" + id).then(function (response) {
                            var result = response.data;
                            return result;
                        });
                    }]
            }
            })
```

La même chose avec $http utilisé dans un **service personnalisé**

```
  .state("personDetail", {
                      url: "/person/:id",
                      templateUrl: "app/people/personDetailView.html",
                      controller: "PersonDetailController as vm",
                      resolve: {
                          person: ['peopleService', '$stateParams', function
(peopleService, $stateParams) {

                              var id = $stateParams.id;
                              return peopleService.getPerson(id).then(function
(response) {
                                  var result = response.data;
                                  return result;
                              });
                          }]
                      }
                  })
```

Le service simplifié utilisé

```
(function () {
    'use strict';

    angular.module('app').service('peopleService', peopleService);

    function peopleService($http) {
        var baseUrl = "/api/People/";

        var getPerson = function (id) {
            return $http.get(baseUrl + "/" + id);
        }

        return {
            getPerson: getPerson
        }
    }

})();
```

➔ Pour activer une route

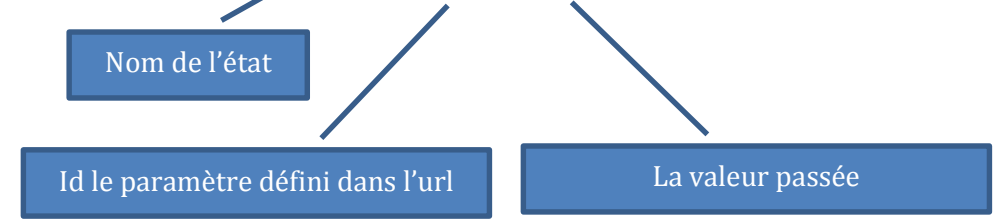
3 possibilités :

- Depuis une **vue** : Lien avec **ui-sref** : on indique le nom de l'état représentant la vue vers laquelle naviguer.

```
<a ui-sref="home">Accueil</a>
```

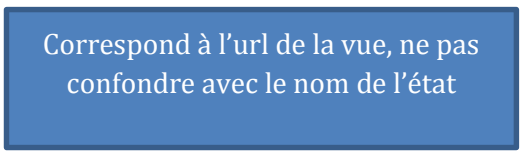
On passe des paramètres entre parenthèses + accolades

```
<a ui-sref="personDetail({id:person.Id})">{{person.Name}}</a>
```



- Depuis une **vue** avec un simple lien/**url**

```
<a href="../index.html#/people">Afficher la liste des personnes.</a>
```



- Depuis un **contrôleur** : avec **$state.go**

Il suffit depuis la vue d'appeler la méthode du contrôleur

```
<button ng-click="vm.goPeopleList()">Retourner à la page d'accueil</button>
```

Avec **paramètre**

```
$state.go("personDetail", { id: 1 });
```

Nom de l'état

Paramètre passé

2. Contrôleurs

Ordre création

(Créer un dossier dans le dossier app avec la catégorie exemple « people »)

- ➔ Créer le contrôleur (ajout d'un template contrôleur Sidewaffle possible) Dépendances ($scope, services), membres et activate (les données chargées/configurations à l'activation)
- ➔ Créer le Template html (+style, animation, etc.) et ajouter le binding
- ➔ Ajouter la route (dans « app » ou « config.route »)
- ➔ Référencer le contrôleur dans la page d'index

a. Injection de dépendances

```
(function () {
    'use strict';

    angular.module('app').controller('PeopleController', ['$state', 'peopleService',
PeopleController]);

    function PeopleController($state, peopleService) {
        var vm = this;
        // etc.
    }
})();
```

Avec **$inject**

```
(function () {
    'use strict';

    angular.module('app').controller('PeopleController',PeopleController);

    PeopleController.$inject = ['$state', 'peopleService'];

    function PeopleController($state, peopleService) {
        var vm = this;
      // etc.
    }
})();
```

b. *Contrôleur avec $scope*

Nom du contrôleur et ajout au module

Dépendances

Ex

```
(function () {
    'use strict';

    angular.module('app').controller('PeopleController', ['$scope', '$location',
'peopleService', PeopleController]);

    // 3 controleur
    function PeopleController($scope, $location, peopleService) {

        $scope.people = [];

        $scope.getPeople = function () {
            peopleService.getPeople().then(function (response) {
                    $scope.people = response.data;
                });
        }

        $scope.deletePerson = function (id) {
            peopleService.deletePerson(id).then(function (response) {
                $location.url("/people");
            });
        }

        activate();
        function activate() {
            $scope.getPeople();
        }
    }
})();
```

$scope membres bindés par la vue

Chaque contrôleur a un **$scope** (passé en paramètre au contrôleur), toute propriété lui étant ajoutée est accessible depuis la vue. **$rootScope** étant le scope global de l'application.

Contrôleur « As »

$scope n'est pas utilisé

```
(function () {
    'use strict';

    angular.module('app').controller('PeopleController', ['$state', 'peopleService',
PeopleController]);

    function PeopleController($state, peopleService) {
        var vm = this;
        vm.people = [];

        vm.getPeople = function () {
            peopleService.getPeople().then(function (response) {
                    vm.people = response.data;
                });
        }

        activate();
        function activate() {
            vm.getPeople();
        }
    }
})();
```

Utilisation

« Contrôleur as vm » le contrôleur ici est « people »

Ne pas oublier vm devant

```
<body ng-app="app">
    <div ng-controller="PeopleController as vm">
        <ul ng-repeat="person in vm.people">
            <li>{{person.name}}</li>
        </ul>
    </div>
</body>
```

Si on déclare le contrôleur en même temps que les routes

```
app.config(function ($routeProvider) {
        $routeProvider
            .when("/", {
                templateUrl: 'app/people/peopleListView.html',
                controller: 'PeopleController as vm'
            })
        .otherwise({ redirectTo: "/" });
    })
```

« Contrôleur as vm »

Dans la vue

Ne pas oublier « vm » pour l'accès aux membres du contrôleur

```
<h2>(<span>{{vm.people.length}}</span>) personne(s). </h2>
<table>
    <thead>
        <tr>
            <th>Nom</th>
            <th>Twitter</th>
            <th></th>
        </tr>
    </thead>
    <tbody ng-repeat="person in vm.people">
        <tr>
            <td>{{person.Name}}</td>
            <td>{{person.Twitter}}</td>
            <td><button ng-click="vm.getPerson(person)">Détails</button></td>
            <td><a href="#/person/{{person.Id}}">Détails</a></td>
        </tr>
    </tbody>
</table>
```

3. La page d'index

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>AngularJS Demo</title>

    <!-- Styles -->
    <link href="css/default.css" rel="stylesheet" />

    <!-- AngularJS  -->
    <script src="js/angular.js"></script>
    <script src="js/angular-route.js"></script>

    <!--module-->
    <script src="app/app.js"></script>

    <!-- services -->
    <script src="app/services/peopleService.js"></script>

    <!-- contrôleurs -->
    <script src="app/people/peopleController.js"></script>
    <script src="app/people/personDetailController.js"></script>
    <script src="app/people/personEditController.js"></s
</head>

<body ng-app="app">
    <header>
        <h1>Démonstration d'AngularJS</h1>
        <ul>
            <li><a href="#/">Accueil</a></li>
            <li><a href="#/people/">Liste des personnes</a></li>
            <li><a href="#">A propos</a></li>
            <li><a href="#">Contact</a></li>
        </ul>
    </header>
    <section>
        <div ng-view></div>
    </section>
</body>
</html>
```

Portée d'Angularjs : nom du module (vide si aucun module défini)

La directive ng-view (ou ui-view avec ui-router)permet d'indiquer l'emplacement des vues (templates) dans la page d'index ..

On n'indique pas de contrôleurs ceux-ci étant définis dans app.js avec les routing.

A savoir qu'il est possible d'utiliser plusieurs contrôleurs par page. Ex

```
<div ng-controller="PeopleController">
        <!--  -->
</div>
<div ng-controller="SessionController">
        <!--  -->
</div>
```

Contrôleur « as vm »

```
<div ng-controller="shell as vm">
    <h1>{{vm.title}}</h1>
</div>
```

4. Vues

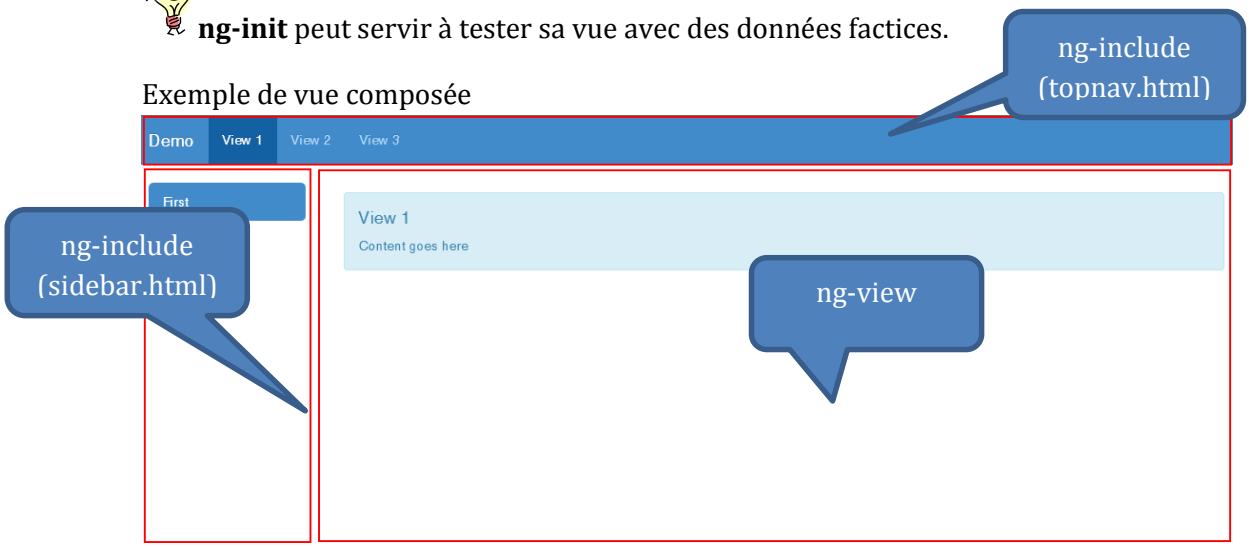
Un template :

- Sauvegardé au format HTML (*.html)
- Affiche les données du contrôleur lié avec binding.

```
<div>
    <h2>{{title}}</h2>
    <table class="table">
        <tr>
            <td>Nom</td>
            <td>{{person.Name}}</td>
        </tr>
        <tr>
            <td>Twitter</td>
            <td>{{person.Twitter}}</td>
        </tr>
    </table>
    <div><a href="#/person/edit/{{person.Id}}">Editer</a></div>
</div>
```

ng-init peut servir à tester sa vue avec des données factices.

Exemple de vue composée



5. Binding (Directives, expressions)

Documentation directives

a. *Texte en lecture*

Expressions {{ }}

```
<span>{{person.Name}}</span>
```

➔ Filtres

Pour filtrer, trier et formater une expression

Expression|filtername :parameter

Principaux filtres :

currency	{{expression|currency : "USD$"}}
date	{{expression|date : "short"}}
filter	Person in people|filter :searchTerm
json	{{expression|json }}
Lowercase,uppercase	{{expression|uppercase}}
number	{{expression| number :1}}
orderby	Person in people|orderBy'LastName'
limitTo	Person in people|limitTo :10

Il est possible de créer ses propres filtres

ng-bind

Equivalent avec **ng-bind**

```
<span ng-bind="person.Name"></span>
```

ng-repeat

Pour les listes. Exemple ci-dessus, le contrôleur a un tableau $scope.people, on fait une boucle

```
  <table>
            <tbody ng-repeat="person in people">
                <tr>
                    <td>{{person.Name}}</td>
                    <td>{{person.Twitter}}</td>
                </tr>
            </tbody>
   </table>
```

$index, $first,$last, $odd, $even

b. *ng-model pour la saisie*

```
<input ng-model="person.FirstName" />
```

Note : si la propriété liée à ng-model n'existe pas elle est créée.

c. *ng-click*

Pour notifier le contrôleur. Pour déclencher une fonction de son contrôleur par exemple

```
<button ng-click="save()"></button>
<button ng-click="selectPerson(this.person)"></button>
```

d. *ng-submit*

Pour les formulaires. Dans la balise form permettant de déclencher une méthode par exemple à la validation.

e. *ng-include*

Permet d'insérer un template d'un fichier html ou défini dans la page.

➔ Template d'un **fichier** html inséré

```
<div ng-include=" '/Templates/people.html' "></div>
```

➔ **Template** dans la **page**

```
<div ng-include="'itemTmpl.html'"></div>

<script type="text/ng-template" id="itemTmpl.html">
<!-- code retiré pour la clarté -->
</script>
```

➔ Template avec switch selon la **condition**

Note il faut ajouter l'extension « .html » malgré que ce soit dans la page.

```
<div ng-include="isEditing ? 'editTmpl.html' : 'itemTmpl.html'"></div>

<script type="text/ng-template" id="itemTmpl.html">
<!-- code retiré pour la clarté -->
</script>
<script type="text/ng-template" id="editTmpl.html">
<!-- code retiré pour la clarté -->
</script>
```

f. *ng-src*

Pour image.On peut définir une expression pour modifier l'image par exemple

```
<img ng-src="{{person.img}}" />
```

g. *ng-show et ng-hide*

Pour afficher/cacher un élément

Exemple affiche seulement si l'élément n'est pas indéfni

```
<div ng-show="selectedPerson"></div>
```

(Les nombres égal à 0, les variables undefined ou null, chaines de caractères vides renvoient false)

h. *Styles avec ng-class*

Pour définir un style CSS

https://docs.angularjs.org/api/ng/directive/ngClass

i. *+ ng-if, ng-switch, etc.*

Il est possible également de créer ses propres directives …

6. Formulaire

Un **formulaire** doit avoir un **nom**, de même chaque **champ (input) doit avoir un nom**. Les champs utilisent **ngModel** pour binder les propriétés (de $scope ou vm selon le contrôleur).

Formulaire avec ngSubmit

```
<form name="personForm" ng-submit="vm.save()">

    <input type="submit">Save</input>
</form>
```

Formulaire avec ngClick

```
<form name="personForm">

   <input type="submit"  ng-click="vm.save()">Save</input>
</form>
```

Validation de formulaire

Angular offre une validation des saisies pour les formulaires.

novalidate : permet de valider malgré les erreurs
directives disponibles : *required, pattern, minlength, maxlength, min,max*
Breeze Angular validation : Si on utilise Breeze avec Angular il existe une validation.
ngMessages (référencer angular-messages.js et ajouter la dépendance dans le module) permet de simplifier l'affichage des erreurs de validation.

Directives de validation pour les champs
ng-minlength, ng-maxlength, ng-pattern, ng-change

Validation
$pristine et $dirty : un champ du formulaire a été modifié
$valid et $invalid : le formulaire contient ou non des erreurs de validations
$error toutes les validations

Un champ peut avoir une ou plusieurs règles de validation :

```
<input name="inputName" type="text" ng-model="vm.person.Name" ng-class="{'has-error':
personForm.inputName.$invalid }" placeholder="Entrez le nom de l'utilisateur" ng-
minlength="2" ng-maxlength="20" required />
```

➔ $submitted pour savoir si le formulaire a été soumis

```
<span ng-show="personForm.$submitted && personForm.inputName.$error.required">Nom
d'utilisateur requis.</span>
```

➔ Un **bouton** qui n'est **accessible** que si le **formulaire est valide**

```
<form name="editForm" novalidate>
   < !--  code retiré pour la clarté -->
   <button ng-click="save()" ng-disabled='!editForm.$valid'>Save</button>
</form>
```

➔ Savoir si un formulaire a été modifié avec $dirty

```
<span ng-show="personForm.$dirty">Formulaire modifié</span>
```

Marquer un champ invalide

➔ Contour de champ

✓ Styles ng-invalid (et ng-valid est également utilisable).

```
input.ng-invalid {
    border-color: #ec3f41;
  }
```

✓ On peut également utiliser ng-class pour définir un style

```
<input name="inputName" type="text" ng-model="vm.person.Name" ng-class="{'has-error':
personForm.inputName.$invalid }" required />
```

✓ On peut utiliser les styles Bootstrap (bootsrap validation states) avec les classes « has-error », « has-success », etc.

Validation states

Bootstrap includes validation styles for error, warning, and success states on form controls. To use, add `.has-warning`, `.has-error`, or `.has-success` to the parent element. Any `.control-label`, `.form-control`, and `.help-block` within that element will receive the validation styles.

Conveying validation state to assistive technologies

Using these validation styles to denote the state of a form control only provides a visual indication, which will not be conveyed to users of assistive technologies – such as screen readers.

Ensure that an alternative indication of state is also provided. For instance, you can include a hint about state in the form control's `<label>` text itself (as is the case in the following code example), or associate an additional element with textual information about the validation state with the form control using `aria-describedby` (see the example in the following section). In the case of an error, you could also use the `aria-invalid="true"` attribute on the form control.

EXAMPLE

Input with success

Input with warning

Input with error

➔ Afficher un message d'erreur à côté du champ.

Informations de l'utilisateur

Name : Nom d'utilisateur requis.

```
<span ng-show="personForm.inputName.$error.required">Nom d'utilisateur requis.</span>
```

Il est possible d'Afficher un message différent selon l'erreur

```
<input name="inputName" type="text" ng-model="vm.person.Name" ng-class="{'has-error':
personForm.inputName.$invalid }" placeholder="Entrez le nom de l'utilisateur" ng-minlength="2"
required />
<span ng-show="personForm.inputName.$error.required">Nom d'utilisateur requis.</span>
<span ng-show="personForm.inputName.$error.minlength">Le nom d'utilisateur doit avoir 2 caractères
minimum.</span>
```

➔ Utilisation possible de **ngMessages** (à partir d'AngularJS 1.3)

Ajouter une référence à « angular-messages.js » et une dépendance au module

```
var app = angular.module('app', ['ngRoute','ngMessages']);
```

```
<input name="inputTwitter" type="text" ng-model="vm.person.Twitter" ng-class="{'has-
error': personForm.inputTwitter.$invalid }" placeholder="Entrez le Twitter de cet
utilisateur" required/>
<div ng-messages="personForm.inputTwitter.$error">
   <div ng-message="required">Twitter requis.</div>
    <!-- etc. -->
</div>
```

« blur » : le binding se fait à la perte de focus du champ, la règle de validation est évaluée à ce moment

input Api reference

➔ databing options avec **ng-model-options** (à partir d'AngularJS 1.3)

```
<input name="inputName" type="text" ng-model="person.Name" placeholder="Nom de
l'utilisateur (requis)" ng-model-options="{updateOn: 'blur'}" required />
<span ng-show="personForm.inputName.$error.required">Nom d'utilisateur requis.</span>
```

on peut également utiliser debounce (le changement de valeur après un certain temps défini)et les cumuler (documentation)

Le formulaire n'est soumis que s'il n'a pas d'erreurs de validation

Exemple de formulaire

```
<form name="personForm" ng-submit="personForm.$dirty && personForm.$valid &&
vm.save()">
    <fieldset>
        <legend>Informations de l'utilisateur</legend>
        <div ng-class="{'has-error': personForm.$submitted &&
personForm.inputName.$invalid }">
            <label for="inputName">Name</label>
            <input name="inputName" type="text" ng-model="vm.person.Name"
placeholder="Nom de l'utilisateur (requis)" ng-minlength="2" required />
            <span ng-show="personForm.$submitted &&
personForm.inputName.$error.required">Nom d'utilisateur requis.</span>
            <span ng-show="personForm.$submitted &&
personForm.inputName.$error.minlength">Le nom d'utilisateur doit avoir 2 caractères
minimum.</span>
        </div>
        <div ng-class="{'has-error': personForm.$submitted &&
personForm.inputTwitter.$invalid }">
            <label for="inputTwitter">Twitter</label>
            <input name="inputTwitter" type="text" ng-model="vm.person.Twitter"
placeholder="Twitter de l''utilisateur (requis)" ng-pattern="/^@[a-zA-Z0-9]*/"
required />
            <span ng-show="personForm.$submitted &&
personForm.inputTwitter.$error.required">Twitter requis.</span>
            <span ng-show="personForm.$submitted &&
personForm.inputTwitter.$error.pattern">Twitter invalide.</span>

        </div>
        <input type="submit" value="Valider" />
        <input type="button" ng-click="vm.cancel()" value="Annuler" />
    </fieldset>
</form>
```

La classe CSS (bordure rouge) est ajoutée si le formulaire a été soumis et comporte des erreurs

Texte affiché si le champ comporte cette erreur et le formulaire a été soumis

CSS

```
.has-error input{ border-color: red; }
```

7. Services

Api référence

$http
$log
$timeout
$interval
…

UI related services
$window
$browser
$location
$animate
$anchorScroll

Il est possible également de **créer ses propres services**.

IIFE et 'use strict' , bonnes pratiques

Enregistrement du service avec la méthode « factory » ou « service » de l'objet angular

Ex

```
(function () {
    'use strict';

    angular.module('app').service('peopleService', peopleService);

    function peopleService($http) {
        var baseUrl = "/api/People/";

        var getPeople = function () {
            return $http.get(baseUrl);
        }
        return {
            getPeople: getPeople
        }
    }

})();
```

Utilisations des services :
$location : un service pour la navigation
dataService : le service personnalisé que l'on a créé

```
var peopleController = function ($scope, $location, peopleService) { }
```

$http

Voyons plus en détail ce service permettant de faire des requêtes http vers un serveur.
La documentation
On utilise :
$http.get, $http.head, $http.post, $http.put, $http.delete, $http.jsonp (retournent des « promises »)

- $http.get pour les requêtes http get
- $http.post pour les requêtes http post
- Il est possible de finir les headers avec $http

```
$http({ method: 'GET', url: '/url' }).
   success(function (data, status, headers, config) {
   }).
   error(function (data, status, headers, config) {
   });
```

```
(function () {
    'use strict';
    angular.module('app').service('peopleService', peopleService);

    function peopleService($http) {
        var baseUrl = "/api/People/";
        var getPeople = function () {
            return $http({
                method: 'GET',
                url: baseUrl,
                params: { 'update-time': new Date().getTime() }
            });
        }
        var getPerson = function (id) {
            return $http.get(baseUrl + "/" + id);
        }
        var addPerson = function (person) {
            return $http.post(baseUrl, person);
        }
        var updatePerson = function (person) {
            return $http.put(baseUrl, person);
        }
        var deletePerson = function (id) {
            return $http({
                method: 'DELETE',
                url: baseUrl + '/' + id,
            });
        }
        return {
            getPeople: getPeople,
            getPerson: getPerson,
            addPerson: addPerson,
            updatePerson: updatePerson,
            deletePerson: deletePerson
        }
    }
})();
```

Ajout d'un paramètre en cas de problème de cache avec AngularJS

..Utilisation depuis un contrôleur

```
        $scope.getPeople = function () {
            peopleService.getPeople().then(function (response) {
                    $scope.people = response.data;
                });
        }
        $scope.deletePerson = function (id) {
            peopleService.deletePerson(id).then(function (response) {
                $location.url("/people");
            });
        }
```

Récupération des données

```
  $scope.save = function () {
            if ($scope.person.Id == 0) { // ajout
                peopleService.addPerson($scope.person).then(function () {
                    $location.path("/people/");
                })
            }
            else { // modification
                peopleService.updatePerson($scope.person).then(function () {
                    $location.path("/person/" + $scope.person.Id);
                })
            }
        }
```

RESTfull : $resource

8. Animation

1er Convention et CSS

Exemple création d'une animation « shuffle-animation » .On doit définir pour l'animation :

.ng-enter,
.ng-enter-active(fin entrée)
.ng-leave
.ng-leave-active(fin quitter)

```
.animate-container {
    position: relative;
}

    .animate-container > div {
        padding: 10px;
        width: 100%;
    }

.shuffle-animation.ng-enter,
.shuffle-animation.ng-leave {
    position: absolute;
}

.shuffle-animation.ng-enter {
    -o-transition: ease-out all 0.3s 0.4s;
    -webkit-transition: ease-out all 0.3s 0.4s;
    transition: ease-out all 0.3s 0.4s;
}


.shuffle-animation.ng-leave {
    -o-transition: 0.3s ease-out all;
    -webkit-transition: 0.3s ease-out all;
    transition: 0.3s ease-out all;
}

.shuffle-animation.ng-enter,
.shuffle-animation.ng-leave.ng-leave-active {
    left: 2em;
    opacity: 0;
}

.shuffle-animation.ng-leave,
.shuffle-animation.ng-enter.ng-enter-active {
    left: 0;
    opacity: 1;
}
```

2nd utilisation

```
  <section class="col-md-10">
        <div class="animate-container">
            <div ng-view class="shuffle-animation"></div>
        </div>
    </section>
```

IV- Breeze

http://www.breezejs.com/
Documentation et Api

1. *Partie « Server »*

Création d'un Web Service

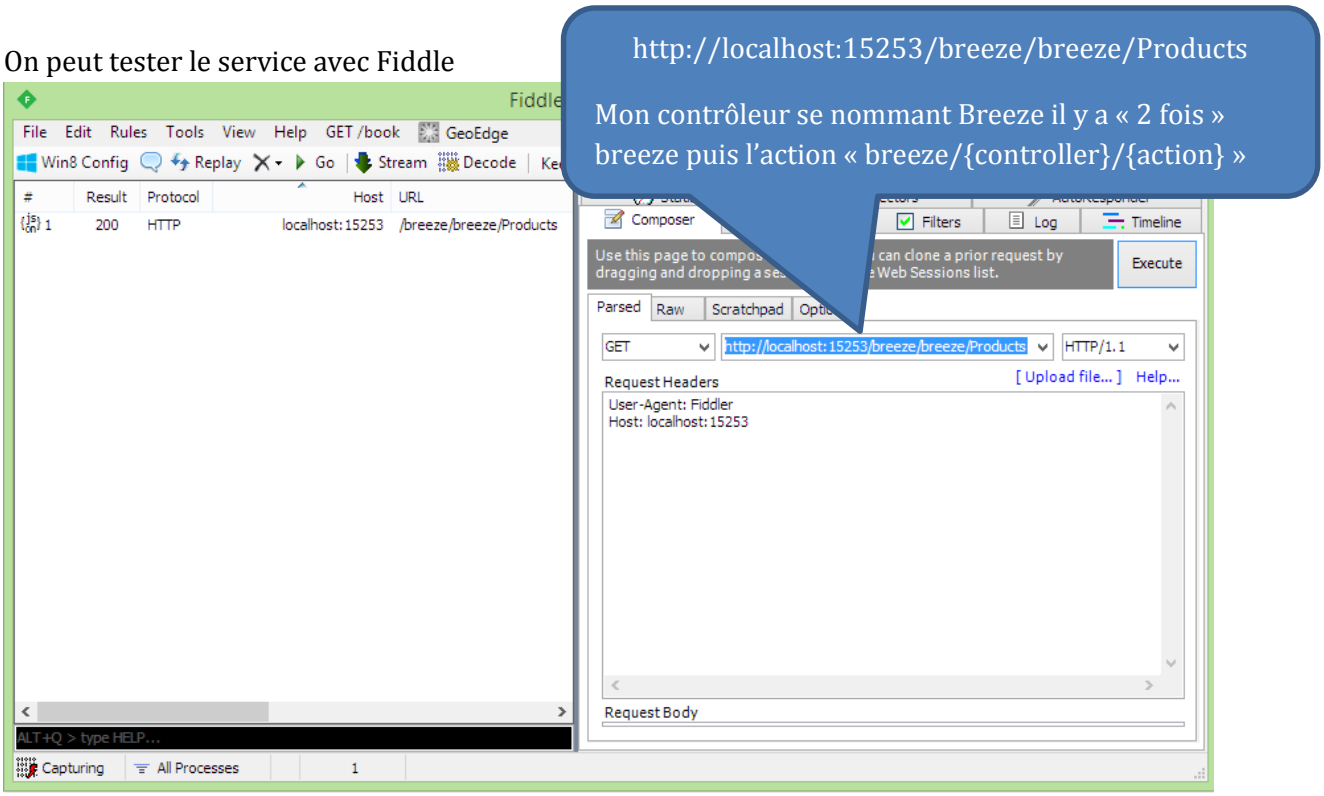
Requiert Breeze Server et Breeze context provider pour EF6 (Package NuGet) pour la partie server
Pour l'exemple j'utilise la base de données Northwind et un entity data model

```
using System;
using System.Linq;
using Breeze.WebApi2;
using System.Web.Http;
using Newtonsoft.Json.Linq;
using Breeze.ContextProvider.EF6;
using Breeze.ContextProvider;

namespace BreezeBase.Controllers
{
    [BreezeController]
    public class BreezeController : ApiController
    {
        private readonly IRepository _repository;

        public BreezeController()
            :this(new NorthwindRepository())
        { }
        public BreezeController(IRepository repository)
        {
            _repository = repository;
        }

        [HttpGet]
        public string Metadata()
        {
            return _repository.MetaData;
        }

        [HttpPost]
        public SaveResult SaveChanges(JObject saveBundle)
        {
            return _repository.SaveChanges(saveBundle);
        }

        [HttpGet]
        public IQueryable<Product> Products()
        {
            return _repository.Products();
        }

        [HttpGet]
        public IQueryable<Category> Categories()
        {
            return _repository.Categories();
        }

    }
}
```

Le repository

```
public interface IRepository
{
    IQueryable<Product> Products();
    IQueryable<Category> Categories();
    SaveResult SaveChanges(JObject saveBundle);
    string MetaData { get; }
}
public class NorthwindRepository : IRepository
{
    private readonly EFContextProvider<NorthwindEntities> _contextProvider
        = new EFContextProvider<NorthwindEntities>();

    public string MetaData
    {
        get { return _contextProvider.Metadata(); }
    }

    public IQueryable<Product> Products()
    {
        return _contextProvider.Context.Products;
    }

    public IQueryable<Category> Categories()
    {
        return _contextProvider.Context.Categories;
    }

    public SaveResult SaveChanges(JObject saveBundle)
    {
        return _contextProvider.SaveChanges(saveBundle);
    }
}
```

On peut tester le service avec Fiddle



BreezeWebApiConfig

```
public static class BreezeWebApiConfig {

   public static void RegisterBreezePreStart() {
     GlobalConfiguration.Configuration.Routes.MapHttpRoute(
         name: "BreezeApi",
         routeTemplate: "breeze/{controller}/{action}"
     );
   }
}
```

2. *Partie Client*

Breeze.js et q.js requis côté client(JavaScript)

Exemple avec Knockout

```
<body>

    <table>
        <thead>
           <tr>
               <td>Id</td>
               <td>Name</td>
           </tr>
        </thead>
        <tbody data-bind="foreach:categories">
            <tr>
                <td data-bind="text:CategoryID"></td>
                <td data-bind="text:CategoryName"></td>
            </tr>
        </tbody>
    </table>

    <script src="Scripts/jquery-2.1.1.min.js"></script>
    <script src="Scripts/knockout-3.1.0.js"></script>
    <script src="Scripts/q.js"></script>
    <script src="Scripts/breeze.min.js"></script>

    <script>

        var Category = function (categoryID, categoryName) {
            this.CategoryID = ko.observable(categoryID);
            this.CategoryName = ko.observable(categoryName);
        }

        var vm = {
            categories: ko.observableArray([])
        }


        var manager = new breeze.EntityManager("/breeze/breeze/");
        manager.metadataStore.fetchMetadata(manager.dataService)

        var query = new breeze.EntityQuery()
            .from("Categories")
            .orderBy("CategoryName");


        manager.executeQuery(query)
            .then(function (response) {
```

Binding avec Knockout

Manager

On définit l'url de base du service

Requête « like Linq ».ici on appelle la méthode « Categories » du contrôleur Breeze

```
                response.results.forEach(function (category) {
                    vm.categories.push(new Category(category.CategoryID(),
category.CategoryName()));
                })
            }).fail(function (e) {
                alert(e);
            });

        ko.applyBindings(vm);

    </script>
```

Exécution de la requête

Autre exemple + récupère que des **données partielles** (c'est-à-dire ne récupérer que certaines colonnes pour les afficher dans une vue « liste »). Et on récupérera toutes les colonnes pour la vue « détails » d'un élément.

```
      function getCategoriesPartials() {
            var query = new breeze.EntityQuery()
           .from("Categories")
           .select("CategoryID, CategoryName")
           .orderBy("CategoryName")
           .toType("Category")
           .using(manager)
           .execute(querySucceeded, queryFailed);

            function querySucceeded(data) {
                vm.categories(data.results);
            }
            function queryFailed() {

            }
        }
```

Exemple de code pour **mise à jour**

```
$("#btnSave").on("click", function () {

            var newCategory = {
                CategoryName: 'Fruits',
                Description: 'Good !!'
            };
            manager.createEntity("Category", newCategory);

            manager.saveChanges()
                .catch(saveFailed)
                .then(saveSuccess);

            function saveFailed() {
                alert("fail to save changes");
            }

            function saveSuccess(saveResult) {
                alert("Changes saved successfully");
            };
})
```

Création d'une entité

Sauvegarde des changements

Filter - Créer un « predicate »

```
var predicate = breeze.Predicate.create("CategoryID", "==", "1");
var predicate2 = breeze.Predicate.create("CategoryID", "==",
"1").or("CategoryID","==","2");
```

- Soit **trois arguments** (propertyName,comparison,value), soit objet **predicate**
- **Comparaisons supportées** : Equals, Contains, StartsWith, EndsWith, GreaterThan, GreaterThaneOrEqual,LessThan,LessThanOrEqual,NotEquals

Trier - Avec **orderBy.**

Supporte tri sur plusieurs colonnes et « desc »

Récupérer des données localement

Exemple je crée une fonction réutilisable

```
var result = getFromLocal("Products","ProductName",predicate);
```

```
        function getFromLocal(resource,orderBy,predicate) {
             return new breeze.EntityQuery.from(resource)
                 .orderBy(orderBy)
                 .where(predicate)
                 .using(manager)
                 .executeLocally();
         }
```

Créer une entité

```
            var newCategory = {
                 CategoryName: 'Fruits',
                 Description: 'Good !!'
             };
             manager.createEntity("Category", newCategory);
```

Marquer une entité à supprimer

```
        $("table").on("click", "#delete", function () {
             var entity = ko.dataFor(this);
             entity.entityAspect.setDeleted();
        })
```

Note : on peut s'abonner aux changements (subscribe)

Sauver ou annuler les changements

A ce niveau, les données n'ont été modifiées que localement (côté client).On a le choix entre mettre à jour la base de données ou annuler les changements.

- ✓ Save

```
manager.saveChanges() ;
```

- ✓ Cancel

```
if (manager.hasChanges())
    manager.rejectChanges();
```

EntityAspect

Contient les informations pour chaque entité.

On peut l'utiliser pour consulter/modifier l'état d'une entité (entityState), obtenir les valeurs originales d'une entité modifiée, pour la validation et suivre les changements (subscribe).

Méthodes utiles

```
var changes = manager.getChanges();
var entities = manager.getEntities();
```