Knockout, Durandal et Angular

Présentation des 3 Frameworks JavaScript pour le binding

Jérôme Romagny

20/07/2014

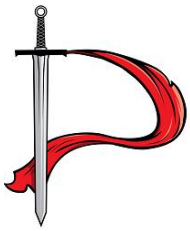


Table des matières

Binding sans framework …

On pourrait utiliser jQuery et ses méthodes val, text, html pour définir dynamiquement le contenu de la page html par exemple

```
<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>
```

Astuce : 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" />
```

Sidewaffle (templates pour Visual Studio)

Sidewaffle : extension pour Visual Studio ajoutant des templates pour **Knockout, Durandal, Angular**, etc.

I. Knockout

1. Installation

Avec un **gestionnaire de packages :**

Bower

```
bower i knockout -S
```

Autres possibilités :

- Téléchargement sur le site
- **Package NuGet** depuis Visual Studio
- **CDN** : cdnjs

2. 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

Documentation

Notification de changement de valeur

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

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

ko.applyBindings(vm);
```

...

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

b. ObservableArray

Documentation

Notification ajout/suppression d'éléments

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

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

ko.applyBindings(vm);
```

...

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

Fonction	*Description*
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

Binding dans le html : N'indiquer les parenthèses dans le html que s'il y a une propriété derrière

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

c. Computed

Documentation

```
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);
```

Html

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

3. 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)

Control flow

if	condition
ifnot	
foreach	boucle
with	Pour l'élément spécifié

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

4. Templates

Documentation

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 () {
        vm.showDetails(false);
      }
    }

    $('table').on('click', '.edit', function () {
      vm.showDetails(true);
    });
    vm.people.push(new Person('Marie')); // add
    vm.people.push(new Person('patrick'));
    ko.applyBindings(vm);
  </script>
```

5. Binding Context

Documentation

$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>`

6. 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 :method" »

Exemple

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

7. Custom bindings

Documentation

- **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'));

ko.applyBindings(vm);
```

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

8. Ko 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

9. 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),
    });
  };
```

10. Subscribe

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

11. Extenders

Documentation

12. 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>

13. 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
         });
```

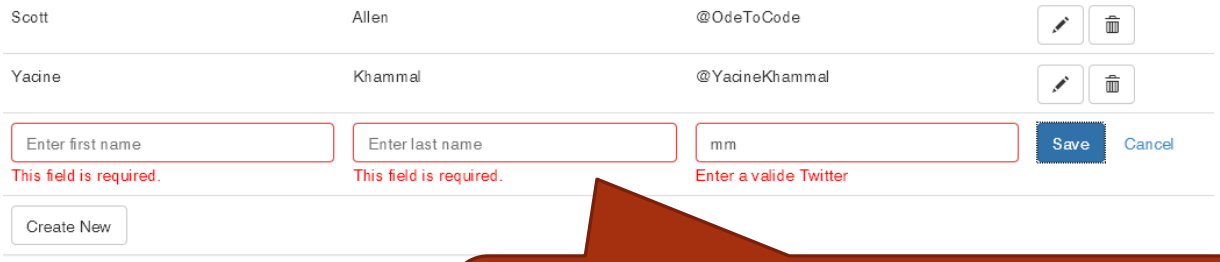
Test : exemple lors de la validation de données utilisateurs saisies

```
         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

II. Durandal

Documentation

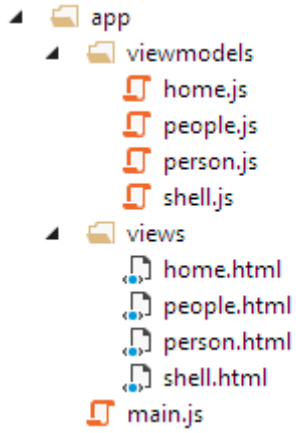
Durandal utilise :

- **jQuery** (DOM)
- **Knockout** (Binding)

1. 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

2. Convention

- correspondance nom page html → fichier js (exemple : home.html et home.js)
- views dans un répertoire « views » et viewmodels dans répertoire « viewmodels » 2.

3. Page de départ index

(*.html ou *.cshtml pour Mvc)

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

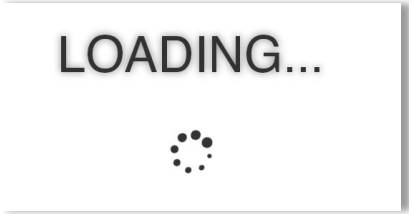
```
<script src="//ajax.googleapis.com/ajax/libs/jquery/2.1.1/jquery.min.js"></script>
<script src="Scripts/knockout-3.1.0.js"></script>
<!-- … main -->
        <script src="Scripts/require.js" data-main="app/main"></script>
```

- ✓ **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>
```

4. Main

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

```
// 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');
    });
});
```

5. 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>
    <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>
    <div class="page-host" data-bind="router: { transition:'entrance' }"></div>
</div>
```

6. 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

7. 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
  }
```

8. 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);
        }
      }
    }
  });
```