Module d’apprentissage
AngularJS

Sommaire

CGI

1. Partie Théorique

1.1. Prérequis

Afin de débuter ce tutoriel il est bien d'avoir connaissance de quelques éléments de base :

- HTML : Structure d'un document, balises.
- CSS : Utilisation d'un sélecteur, utilisation du système clé valeur afin de mettre en forme.
- JavaScript : Déclaration d'une variable, déclaration d'une fonction.
- JQuery : Les connaissances en JQuery sont les bienvenues et aideront à avancer plus vite.
- Bootstrap de twitter : Framework CSS qui permettra de passer moins de temps à fabriquer des styles.

1.2. Introduction

1.3. Structure du projet

Un projet AngularJS ne repose sur aucune structure imposée, c'est pour cela que nous allons utiliser un pack d'outils nous permettant d'intégrer de bonnes pratiques de développement pour un projet pur web. Ce pack nous permettra d'utiliser facilement un serveur web desservant notre projet, des outils de test, de « repporting » sur la qualité du code ainsi qu'un mécanisme de « build » permettant de réduire la taille du code source ainsi que de l'offusquer avant de le mettre en production.

- Projet
  - app : répertoire du projet en développement.
    - 404.html : Page à afficher en cas de 404
    - favicon.ico
    - index.html : C'est l'unique page chargée avec toutes les dépendances.
    - scripts : Répertoire ou ira tout le code JavaScript du projet
      - app.js : Déclaration de l'application et des routes
      - main.js : Contrôleur principal
    - bower_components : Dépendances JavaScript de l'application
    - images : Répertoire d'images
    - styles : Fichier de style css, sass ou less au choix.
  - test : Répertoire où coder les tests
  - dist : Répertoire créé après un build, c'est l'application prête à être distribuée.
  - Grunfile.js : Fichier de configuration du projet, contenu des répertoires, étapes de build, port du serveur.
  - bower.json : Fichier répertoriant les dépendances JS du projet
  - node_modules : module node js installé localement permettant de faire fonctionner la structure du projet
  - package.json : dépendance des modules node js.

1.4. Concepts de base

1.4.1. MVC

AngularJS est un framework dont les éléments sont orientés MVC.

- Modèle

- Les Modules : Une application AngularJS est un module. Dans une usine logicielle, il peut être utile (pour un maximum de recyclage du code) de subdiviser une application en plusieurs modules.
- Les « Resources » : AngularJS utilise un service ressource qui permet de construire des objets orientés API.

- Vue
  - Fichier HTML : À l'instar des Framework web classiques avec backend, AngularJS fournit un système de templating basé sur des balises ou des attributs appelant des directives.
  - CSS : Rien de neuf n'est apporté par AngularJS.
  - Les « Directives » : Les directives sont du code JavaScript dédiées à enrichir la vue et l'expérience utilisateur.
- Controller
  - Les « Controller » : Les contrôleurs jouent le même rôle que les contrôleurs des différents Framework web, c'est-à-dire qu'il s'agit de l'interface de contrôle.

1.4.2. Autre concept essentiel : les scopes

Les « Directive » ainsi que les « Controller » transportent un objet essentiel appelé « scope ». Cet objet est le service central d'AngularJS, il assure le binding des données. Chaque appel à un service mettant à jour le « scope » adaptera dynamiquement les templates. Il est donc important que tous les appels JavaScript asynchrones reposent sur les services d'AgularJS, ils assurent le « data binding ».

2. Partie pratique

2.1. Initialisation

2.1.1. Utilisation de l'api

Pour ce tutoriel une api est fournie. L'objectif est de construire un site type microblogging. Il existe donc un modèle persisté en base desservi par des urls. Nous pourrons découvrir ces urls au fur et à mesure que nous déroulerons ce tutoriel mais la liberté étant le maitre mot et certain se laissant parfois aller vers leur créativité une documentation est disponible afin de pouvoir pleinement profiter de l'api.
L'adresse ip de l'API n'étant pas fixe, elle sera mentionnée `<api_ip :port>` tout le long de nos expériences.
L'adresse de la documentation est donc **`http://<api_ip :port>/api/swagger/`**

2.1.2. Création du projet

Comme mentionné précédemment nous allons « scaffolder » le projet avec « yeoman ».
Pour cela créons un nouveau dossier dans lequel nous initialiserons le projet.
Dans ce dossier :

```
yo angular
```

Quelques questions sont pour la configuration. Je recommande pour ce tuto d'utiliser compass et bootstrap, ces éléments permettrons d'avancer plus vite et se concentrer sur l'essentiel.

Commencez par modifier app.js, ce fichier contient les routes de notre application mais nous allons mettre l'instance de notre application dans une variable, ainsi il sera plus facile de l'appeler depuis n'importe quel fichier. En effet,

```
angular.module('monApplication', ['dépendance1', 'dépendance2', …]); // permet de déclarer l'instance
angular.module('monApplication') ; // appelle l'instance de l'application en cours.
```

Nous allons modifier le fichier pour obtenir une variable plus courte à appeler et insensible à un changement de nom de module :

```
var demoApp = angular.module('monApplication', ['dépendance1', 'dépendance2', …]) ;
```

Modifions également main.js afin qu'il appelle notre module depuis cette variable :

```
angular.module('monApplication').controller...
```

devient

```
demoApp.controller...
```

Enfin nous allons créer un fichier de paramètres/constantes « settings.js » dans le dossier script. Ce fichier pourra contenir la variable API_URL avec la base de l'url de l'api. Ainsi, si l'adresse devait changer durant la démo, nous pourrons facilement en mettre une nouvelle.

```
var API_URL = "http://<api_url:port>/api/";
```

Terminons notre initialisation en nous assurant que le fichier de settings est bien importé lors du lancement de notre page. Pour cela, rendez-vous dans « index.html » et ajouter la ligne :

```
<script src="scripts/settings.js"></script>
```

avant l'import d'AngularJS.

2.1.3. Resource utilisateur

Les utilisateurs sont au centre de ce projet, c'est pour cela que nous allons commencer par construire notre première « Resource User »,

Créons le dossier « resource » dans script avec le fichier « user.js » et plaçons-y ce code :

```
demoApp.factory('User', ["$resource", function($resource){
      /* définissons la resource user. Ça définition repose
       * sur son url paramétrée. Nous définissons
       * ainsi tous les services :
       * update, save, delete, list, retrieve.
       */
     var User = $resource( API_URL + "users/:userId/:action",
         { userId: "@identifier.id", action: "@action"},
         {
             'update': {
                 'method': 'PUT'
             },
             'save': {
                 'method': 'POST'
             },
             'delete': {
                 'method': 'DELETE'
             },
             'list': {
                 'method': 'POST'
             },
```

```
            'retrieve': {
                'method': 'GET'
            }
        });

    return User;

}]);
```

Notre ressource étant créée, assurons-nous de bien l'importer en ajoutant dans « index.html » la ligne :

```
<script src="scripts/resources/user.js"></script>
```

Après l'import de « app.js »

2.1.4. Authentification et récupération d'un User

L'api dispose d'une authentification simple et classique. Ainsi une url permet de récupérer les identifiants dans un format json et renvoie un token qui faudrait mettre dans une entête à chaque requête. Ce token permettra à l'api d'identifier la session en cours avec son utilisateur associé.

Format du json envoyé :

```
{
    'username': 'login',
    'password': 'pass'
}
```

Format de la réponse :

```
{
    'token': 'abdjfhkgusv45bjkbc564njbcn...'
}
```

Ajoutons dans notre « resource » utilisateur une méthode permettant de nous authentifier et de conserver ce token sur la session. L'url authentifiant l'utilisateur est différente de celle configurée dans notre « resource » nous allons donc avoir besoin d'un service permettant de faire les requêtes ajax avec angular, pour l'importer, il suffit d'ajouter le service dans la déclaration de notre ressource :

```
demoApp.factory('User', ["$resource", "$http", function($resource, $http){
```

Ajoutons également une variable nous permettant de vérifier si un utilisateur est authentifié :

```
User.isLogged = false;
/* successCb et errorCb sont des fonctions callbacks, ils seront
 * appelés dès que l'authentification sera terminée afin de permettre
 * au controller l'ayant appelé d'enchaîner une action.
 */
User.login = function(login, password, successCb, errorCb){
    var data = {username: login, password: password};
    $http.post(API_URL + 'auth/',data)
        .success(function(data, status, headers, config){
            if(status===200 && data && data.token){
                $http.defaults.headers.common['Authorization'] = 'Token ' +
data.token;
                User.isLogged = true ;
                successCb(data, status, headers, config);
            }else{
```

```
                errorCb(data, status, headers, config);
            }
        })
        .error(errorCb);
 }
```

Nous sommes, avec ce code, capable d'authentifier un utilisateur, mais pour les requêtes futures nous avons besoin de connaître son id et éventuellement d'autres informations, écrivons une méthode qui fera ce travail et que nous appellerons une fois l'authentification terminée :

```
/* On deporte un peu de logique et récupère les callbacks
 * afin de les jouer à la toute fin des appels ajax.
 */
function setUser(successCb, errorCb){
    $http.get(API_URL + "current_user")
        .success(function(data,status,headers,config){
            if(status===200 && data){
                User.isLogged = true;
                User.id = data.id;
                User.first_name = data.first_name;
                User.last_name = data.last_name;
                User.username = data.username;
                User.photo_id = data.photo_id;
                successCb(User);
            }else{
                errorCb(data, status, headers, config);
            }
        })
        .error(errorCb);
  }
```

Dans notre fonction login, changeons juste ces lignes :

```
                User.isLogged = true ;
                successCb(data, status, headers, config);
```

par :

```
                setUser(successCb, errorCb);
```

Plusieurs points à noter :

- Pour récupérer le « current_user », on appelle une url spécifique pour ça définie dans l'api.
- Quand l'authentification est terminée, l'éventuel call back dont on pourrait avoir besoin ne peut reposer que sur le user lui-même. Par exemple quand on voudra gérer des problèmes de permission. Les variables data, status,… ne sont plus intéressantes à exploiter ici.
- Le token est récupéré et est mis par défaut dans les entêtes des requêtes pour toute la session (cf. `$http.defaults.headers.common['Authorization'] = 'Token ' + data.token;`)

La session étant illimitée tant que la page n'est pas fermée, nous devons permettre à l'utilisateur de pouvoir changer de session. Il faut pour cela ajouter une méthode logout :

```
User.logout = function(successCb){
    //réinitialisation du token
    delete $http.defaults.headers.common['Authorization'];
    // L'information isLogged est maintenant fausse
```

```
    User.isLogged = false;
    // Enlevons toutes les informations utilisateur
    User.id = null;
    User.first_name = null;
    User.last_name = null;
    User.username = null;
    User.photo_id = null;
    successCb();
}
```

Afin de tester notre authentification nous allons devoir créer une page de login et de logout. Cela signifie une vue et un controller :

- Créons un dossier « controller » dans « scripts » et un dossier « user » dans controller. Créons les fichiers « LoginCtrl.js » et « Logout.js » dans user.
- Créons les chemins « views/user/login.html » et « views/user/logout.html »
- Enfin créons « styles/user/login.scss » et « styles/user/logout.scss »

Tous les fichiers que nous allons éditer sont maintenant créés alors codons !
Tout d'abord il faut référencer notre nouvelle vue dans « app.js » en ajoutant à notre module une nouvelle route :

```
.when('/login', {
    templateUrl: 'views/user/login.html',
    controller: 'LoginCtrl',
})
.when('/logout', {
    templateUrl: 'views/user/logout.html',
    controller: 'LogoutCtrl',
})
```

Posons un formulaire de « login » dans login.html, nous utiliserons une mise en forme en adéquation avec bootstrap de twitter pour rendu plus sympa sans effort :

```
<section id="login-section">
    <form class="form-horizontal" id="login-form" role="form">
        <div class="form-group">
            <label for="signin-login" class="col-sm-2 control-
label">Login</label>
            <div class="col-sm-10">
                <input type="text" class="form-control" id="signin-login"
placeholder="Login">
            </div>
        </div>
        <div class="form-group">
            <label for="signin-password" class="col-sm-2 control-
label">Password</label>
            <div class="col-sm-10">
                <input type="password" class="form-control" id="signin-
password" placeholder="Password">
            </div>
        </div>
        <div class="form-group">
            <div class="col-sm-offset-2 col-sm-10">
                <!-- signin est la fonction du scope qui lancera
l'authentification -->
```

```
                <button ng-click="signin()" class="btn btn-default">Sign
in</button>
            </div>
        </div>
    </form>
</section>
```

Par la même occasion, nous pouvons également faire une page de logout avec un lien vers la page de login.

```
<section id="logout-section">
    <h1>Thank you for visiting us</h1>
    <a class="btn btn-primary" href="#/login">Go to login screen</a>
</section>
```

Nos vues étant prêtes il ne reste plus qu'à ajouter les « controller » associés, dans « LoginCtrl » :

```
demoApp.controller('LoginCtrl', ["$scope", "$location", "User",
function($scope, $location, User){
    // Définition de la fonction signin se déclenchant
    // sur click du bouton du même nom
    $scope.signin = function(){
        var login = $('#signin-login').val(); //récupération du login
        var password = $('#signin-password').val();//récupération du mdp
        User.login(login, password,
            // Si tout se passe bien, on redirige vers la main page
            function(){
                if($scope.$$phase || $scope.$root.$$phase){
                    $location.path("/");
                }else{
                    $scope.$apply(function(){
                        $location.path("/");
                    });
                }
            },
            // En cas de problème, on avertit l'utilisateur
            function(data, status, headers, config){
                alert('Authentication fails...');
            })
    }
    $scope.$watch('$root.$routeChangeStart', function(event, currRoute,
prevRoute){
        if(currRoute && currRoute.controller === 'LoginCtrl'){
            $scope.isLogged = User.isLogged;
        }
    });
}]);
```

On peut maintenant tester notre code pour vérifier que tout se passe bien, pour cela utiliser l'utilisateur dont le couple login/password est test01/test01.

On peut par exemple vérifier que nous savons bien récupérer les informations de cet utilisateur en les affichant sur la « mainpage », pour cela il faut éditer son « controller » (« main.js ») et ajouter le service « User » :

```
demoApp.controller('MainCtrl', ["$scope", "User", function($scope, User){
```

En ajoutant ces dependances nous avons ouvert un crochet, il faut penser à bien le fermer ! Sinon le navigateur préviendra d'une erreur javascript. Maintenant il faut que l'utilisateur soit dans le scope pour afficher ces informations sur la page :

```
$scope.user = User;
```

Il ne reste plus qu'à afficher notre utilisateur, ouvrez la vue associée à ce « controller » (« main.html ») et ajouter ces lignes :

```
<!-- Message de bienvenue pour {{user.first_name}} {{user.last_name}},
ng-if est une directive, si la condition n'est pas vraie, le bloc ne sera pas
affiché-->
<h3 class="welcome-message" ng-if="user.isLogged">Bienvenue
{{user.first_name}} {{user.last_name}}</h3>
```

Afin de tester notre page de logout, nous pouvons laisser un lien.

```
<a class="btn btn-lg btn-warning" ng-href="#/logout">Logout</a>
```

Enfin ajoutez le bout de code de déconnexion dans « LogoutCtrl » :

```
demoApp.controller('LogoutCtrl', ["$scope", "User", function($scope, User){
    User.logout(function(){
    });
}]);
```

Attention de ne pas oublier de mettre un peu de html dans « logout.html » sinon cette partie ne chargera pas sans pour autant faire une erreur.

Pour terminer vous pourrez styliser vos écran de login/logout en éditant les scss associés et sans oublier de les référencer dans le main.scss avec les lignes :

```
@import "user/login.scss";
@import "user/logout.scss";
```

2.2. Modules

2.2.1. Faire une directive menu

Afin d'améliorer la navigation dans notre application, nous allons ajouter un menu. Un menu intelligent Qui saura reconnaître seul la page en cours.

Pour cela nous allons éditer/créer 3 fichiers :

- « scripts/directives/demoMenu.js », notre directive
- « views/ui/menu.html » la vue associée à notre menu
- « Index.html » afin d'ajouter notre menu et d'importer notre nouvelle directive

La directive est un concept bien pensé par les créateurs d'angular, simple à appeler, permet de créer du code facilement recyclable et résout quelques problèmes que les développeurs JavaScript connaissent bien liés au chargement de code trop tôt ou trop tard.

Commençons par faire la vue de notre menu, basée (toujours) sur bootstrap :

```
<header class="navbar navbar-inverse" role="banner">
    <div class="">
        <div class="navbar-header">
            <button class="navbar-toggle"  type="button">
                <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="#/">Demo App</a>
        </div>
```

```
        <nav class="collapse navbar-collapse bs-navbar-collapse"
role="navigation">
            <ul class="nav navbar-nav">
                 <!-- nous mettrons ici les liens vers les pages de notre
application -->
            </ul>
            <ul class="nav navbar-nav navbar-right">
                <li>
                    <a href="#/logout" class='' ng-
show="user.isLogged">Logout</a>
                </li>
            </ul>
        </nav>
    </div>
</header>
```

Passons maintenant à notre directive, elle permettra d'ajouter de l'intelligence à ce menu :

```
demoApp.directive('demoMenu', ['User', function(User){
    return {
        templateUrl: 'views/ui/menu.html',
        restrict: 'E',
        scope:{

        },
        replace: true,
        link: function(scope, element, attrs){
            // intégration de l'utilisateur dans le scope
            scope.user = User;
            scope.$root.$on('$routeChangeStart', function(event, currRoute,
prevRoute){
                //désélection de l'entré menu en cours
                $(element.find('li.active')).removeClass('active');
                if(currRoute){
                    var item = $(element.find('a[href="#' +
currRoute.originalPath + '"]'));
                    if(item.length>0){
                        //sélection de la nouvelle entrée menu en cours
                        item.parent('li').addClass('active');
                    }
                }
            }, true);
        }
    }
}]);
```

Notre directive permet au menu de se mettre à jour à chaque fois que la route change. Si son url correspond à un lien pointé sur le menu, ce lien se mettra en surbrillance même si on n'a pas directement cliqué dessus.

Il ne reste plus qu'à intégrer notre menu à la page. Pour cela, éditions « index.html » en ajoutant au dessus du « div » avec l'attribut « ng-view » :

```
<demo-menu id="demo-menu"></demo-menu>
```

Enfin, il ne faut pas oublier d'importer la directive en la plaçant après « app.js » :

```
<script src="scripts/directives/demoMenu.js"></script>
```

2.2.2. Jouer avec les utilisateurs

Nous allons permettre aux utilisateurs de d'abord pouvoir créer un compte, puis également de changer de mot de passe.
A l'instar d'un grand site de microblogging connu, notre application dispose de la possibilité de suivre les posts de certaines personnes (followees). Pour cela nous allons intégrer une page permettant de chercher un utilisateur puis de le suivre.
Pour les plus aguerrit, l'application fournit également la possibilité pour les utilisateurs d'uploader un avatar mais ce tutoriel ne traitera pas ce point.
Pour créer un compte nous exigerons un minimum d'informations, commençons par créer un formulaire (toujours sur la base de bootstrap) dans le fichier
« views/user/request_account.html » :

```
<section id="tb-request-account">
    <form class="form-horizontal" id="tb-resquest-account-form" role="form">
        <h2>Request account</h2>
        <div class="form-group">
            <label for="first_name_input" class="col-sm-2 control-
label">First name</label>
            <div class="col-sm-10">
                <input type="text" class="form-control" id="first_name_input"
placeholder="First name">
            </div>
        </div>
        <div class="form-group">
            <label for="last_name_input" class="col-sm-2 control-label">Last
name</label>
            <div class="col-sm-10">
                <input type="text" class="form-control" id="last_name_input"
placeholder="Last name">
            </div>
        </div>
        <div class="form-group">
            <label for="username_input" class="col-sm-2 control-
label">Username</label>
            <div class="col-sm-10">
                <input type="text" class="form-control" id="username_input"
placeholder="Username">
            </div>
        </div>
        <div class="form-group">
            <label for="email_input" class="col-sm-2 control-
label">Email</label>
            <div class="col-sm-10">
                <input type="email" class="form-control" id="email_input"
placeholder="Enter email">
            </div>
        </div>
        <div class="form-group">
            <label for="password_input" class="col-sm-2 control-
```

```
label">Password</label>
            <div class="col-sm-10">
                <input type="password" class="form-control"
id="password_input" placeholder="Password">
            </div>
        </div>
        <div class="form-group">
            <label for="confirm_password_input" class="col-sm-2 control-
label">Confirm password</label>
            <div class="col-sm-10">
                <input type="password" class="form-control"
id="confirm_password_input" placeholder="Password">
            </div>
        </div>
        <input type="button" ng-click="submitForm()" class="btn btn-primary"
value="Create Account">
        <a href="#/login" class="btn btn-warning">Cancel</a>
    </form>
</section>
```

Maintenant associons un « controller » à notre vue dans le fichier « script/controllers/user/RequestAccount.js »:

```
demoApp.controller('RequestAccountCtrl', ["$scope", "User", function($scope,
User){
    $scope.user = User;

    $scope.createUser = function(){
        $scope.requestSent = false;
        if(!$scope.$$phase && !$scope.$root.$$phase){
            $scope.$apply();
        }
    }

    $scope.submitForm = function(){
        var data = {
            first_name: $('#first_name_input').val(),
            last_name: $('#last_name_input').val(),
            username: $('#username_input').val(),
            email: $('#email_input').val(),
            password: $('#password_input').val(),
            password_confirmation: $('#confirm_password_input').val(),
        }

        // Vérification des informations (light)
        if(data.password !== data.password_confirmation){
            alert("password and password confirmation are different");
            return false;
        }
        if(!data.username || !data.email || !data.first_name
|| !data.last_name || !data.password){
            alert("All field are mandatory!")
```

```
            return false;
        }

        // Création de l'utilisaeur
        var user = new User(data);
        user.$create({userId: null, action: null}, function(){
            alert("Account was created, please wait for administrator
validation.");
        },
        function(){
            alert("Something went wrong, verify your information before
submitting it.");
        });
    }
}]);
```

Pour accéder à cette vue, ajoutons un bouton « create account » sur l'écran de login en ajoutant la ligne suivante :

```
<a href="#/request_account" class="btn btn-primary">Create Account</a>
```

Pour ceux qui préfèrent, on peut imaginer mettre ce lien comme entrée dans le menu et le cacher lorsque l'utilisateur est authentifié.

Enfin n'oubliez pas d'importer notre nouveau « controller » depuis « indesx.html » ainsi que de lui créer une route dans « app.js ».

N'hésitez pas à tester la fonctionnalité. Créez un utilisateur et utilisez-le pour vous identifier.

Maintenant que nous avons notre propre utilisateur, nous allons faire une page où nous pouvons en chercher d'autres et les suivre.

Nous allons créer de nouveaux fichiers : « views/user/search_user.html », « controller/user/SearchUserCtrl.js »

Notre « controller » nous permettra d'utiliser le « search » de la « resource user ». Pour cela ajoutons une fonction de « search » dans la « resource user » :

```
User.search = function(query_string, successCb, errorCb){
    var data = { query_string: query_string};
    user = new User(data);
    user.$list({userId: null, action:"search"}, successCb, errorCb);
}
```

Le controller de recherche pourra donc contenir le code suivant :

```
demoApp.controller('SearchUserCtrl', ["$scope", "User", function($scope,
User){
    $scope.user = User;
    $scope.search_result = []; // liste des résultats de la recherche
    // L'input que nous placerons sur la page
    // aura pour id 'search-user-input'
    $('#search-user-input').on('keyup', function(event){
        var input = $(this);
        var query_string = input.val();
        // L'api ne retourne un résultat que si
        // la recherche dépasse 3 lettres
        if(query_string.length>3){
            User.search(query_string,
                function(data){
```

```
                    $scope.search_result = data.search_result;
                    // L'affectation de notre nouvelle liste de résultats
                    // suffit, le binding des données nous assure d'avoir
                    // une vue à jour !
                },
                function(){
                    $scope.search_result = [];
                    alert("Search encoutered an error.");
                }
            );
        }
    });
}]);
```

La page html associée doit contenir un champ de recherche et un tableau de résultats comme suit :

```
<section>
    <form class="form-horizontal" role="form">
        <div class="form-group">
            <label for="signin-login" class="col-sm-2 control-
label">Login</label>
            <div class="col-sm-10">
                <input id="search-user-input" type="text" placeholder="Search
user" class="form-control">
            </div>
        </div>
    </form>
    <table class="table table-striped">
        <thead>
          <tr>
            <th>First Name</th>
            <th>Last Name</th>
            <th>Username</th>
            <th></th>
          </tr>
        </thead>
        <tbody>
          <tr ng-repeat="userSearch in search_result" ng-if="user.id !=
userSearch.id">
            <td>{{userSearch.first_name}}</td>
            <td>{{userSearch.last_name}}</td>
            <td>{{userSearch.username}}</td>
            <td><!-- follow button --></td>
          </tr>
        </tbody>
      </table>
</section>
```

Pour pouvez déjà faire quelques tests pour vérifier le fonctionnement. Evidemment, n'oubliez pas de :

- créer une route dans « app.js »,
- référencer le fichier source de votre « controller » dans l'index,

- créer une entrée dans le menu pour accéder facilement à la page de préférence visible que par les utilisateurs identifiés.

Le but de cette vue est tout de même de trouver des utilisateurs pour les suivre. Il faut donc modifier la « resource user » afin qu'elle nous permette d'envoyer la bonne requête, modifier la vue de recherche pour voir les utilisateurs que l'on suit déjà et ajouter un bouton nous permettant de suivre les autres.
Ajoutons trois fonctions dans à notre « resource » :

```
// Créer un nouveau lien, le user actuel suit le user avec l'id followeeId
User.follow = function(followeeId, successCb, errorCb){
    var data = {"followee_id":followeeId};
    $http.post(API_URL + "current_user/follow", data)
       .success(successCb)
       .error(errorCb);
}

// Détruire un lien, le user actuel ne suit plus le user avec l'id followId
User.unfollow = function(followId, successCb, errorCb){
    var data = {"follow_id":followId};
    $http.delete(API_URL + "current_user/follow/" + followId)
        .success(successCb)
        .error(errorCb);
}


// Récupérer les followers de l'utilisateur avec l'id userId
User.get_followers = function(userId, successCb, errorCb){
    if(!userId){
        userId = User.id;
    }
    var user = new User;
    user.$retrieve({userId: userId, action:"followers"}, successCb, errorCb);
}

// Récupérer les followers de l'utilisateur avec l'id userId
User.get_followees = function(userId, successCb, errorCb){
    if(!userId){
        userId = User.id;
    }
    var user = new User;
    user.$retrieve({userId: userId, action:"followees"}, successCb, errorCb);
}
```

Ajoutons quelques lignes dans « SearchUserCtrl » permettant de récupérer les followers et les followees de notre utilisateur :

```
$scope.followers = [];
$scope.followees = [];
$scope.$watch("user.isLogged", function(){
    if(User.id){
        refreshFollow();
    }
});
```

```
// Cette fonction pour rafraichir les lists de followers
// et de followees pourra être appelée quand on créera une nouvelle relation
function refreshFollow(){
    User.get_followers(null,
       function(data){
           $scope.followers = data.followers;
       },
       function(){
           alert("Can't retrieve followers");
       }
    );
    User.get_followees(null,
       function(data){
          $scope.followees = data.followees;
       },
       function(){
          alert("Can't retrieve followees");
       }
    );

}
```

Ajoutons une fonction dans notre scope permettant de suivre ou ne plus suivre un autre utilisateur :

```
// Vérifier si un utilisateur est suivi pour savoir quel bouton afficher.
$scope.isFollowed = function(user){
    for(var i in $scope.followees){
        var follow = $scope.followees[i];
        if(follow.followee.id === user.id){
            return true;
        }
    }
    return false;
}

// Suivre l'utilisateur user
$scope.follow = function(user){
    User.follow(user.id,
        function(data){
            refreshFollow();
        },
        function(){
            alert("Can't follow user!");
        }
    );
}

// Ne plus suivre l'utilisateur user
$scope.unfollow = function(user){
    for(var i in $scope.followees){
        var follow = $scope.followees[i];
```

```
        if(follow.followee.id === user.id){
            console.log(follow.id);
            User.unfollow(follow.id,
                function(data){
                    refreshFollow();
                },
                function(){
                    alert("Can't unfollow user!");
                }
            );
            break;
        }
    }
}
```

Pour faire appelle à ces fonctions, il nous faut des boutons, remplaçons dans notre écran de recherche la ligne :

```
            <td><!-- follow button --></td>
```

Par :

```
<td ng-if="!isFollowed(userSearch)"><button ng-click="follow(userSearch)"
class="btn btn-success btn-small">Follow</button></td>
<td ng-if="isFollowed(userSearch)"><button ng-click="unfollow(userSearch)"
class="btn btn-warning btn-small">Unfollow</button></td>
```

Nous pouvons maintenant nous inscrire ou nous désinscrire au suivi des posts d'un autre utilisateur !

Il serait pertinent de réserver une place sur la page d'accueil afin d'afficher les « followers » et le « followees » du compte actif.

2.2.3. Jouer avec les posts

Nous allons utiliser la page principale pour afficher le flux des posts suivit par l'utilisateur. Tout d'abord créons la fonction qui récupère ces posts dans la « resource user » :

```
User.get_followed_posts = function(successCb, errorCb){
    var user = new User();
    user.$retrieve({userId: User.id, action: "followed_posts"}, successCb,
errorCb);
}
```

Ajoutons aussi une nouvelle « resource » dédiée aux posts, elle nous permettra d'utiliser l'api afin d'en créer de nouveau, dans « resources/post.js » :

```
demoApp.factory('Post', ["$resource", function($resource){
    var Post = $resource( API_URL + "posts/:postId/:action",
        { userId: "@identifier.id", action: "@action"},
        {
            'save': {
                'method': 'POST'
            }
        });
    Post.create = function(data, successCb, errorCb){
        var post = new Post(data);
        post.$save({postId: null, action: null}, successCb, errorCb);
    }
```

```
    return Post;
}]);
```

Transformons le « controller » principal « main.js » pour récupérer les posts suivit par l'utilisateur ainsi qu'une fonction permettant d'en créer de nouveaux :

```
// Cette fonction met à jour la liste des posts suivit.
function refreshFollowedPosts(){
    User.get_followed_posts(function(data){
        $scope.followedPosts = data.followed_posts.reverse();
    },
    function(){
        alert("An error occured while retrieving followed posts.")
    });
}

// L'utilisateur peut ne pas être initialisé au chargement de la vue
// On attend qu'il soit logué pour charger les posts
$scope.$watch("user.isLogged", function(){
    if(User.id){
        refreshFollowedPosts();
    }
});

// fonction qui se déclenchera sur pression du bouton post
$scope.post = function(){
    var data = {
        title: $('#post-title').val(),
        body: $('#post-body').val(),
        author_id: User.id
    }
    // Vérifions que les champs ne sont pas vides
    if(!data.title || !data.body){
        alert("All fields are mandatories!");
        return false;
    }
    Post.create(data,
       function(){
          refreshFollowedPosts();
          $('#post-title').val('');
          $('#post-body').val('');
       },
       function(){
           alert('A probleme occured with post.');
       }
    );
}
```

Nous pourrons aussi créer une directive pour nos posts. Si notre application évolue, il sera plus facile d'ajouter des fonctionnalités isolées. Créons le fichier « directives/demoPost.js » et n'oubliez pas de l'importer dans « index.html »:

```
demoApp.directive('demoPost', ['User', function(User){
    return {
```

```
        templateUrl: 'views/ui/post.html',
        restrict: 'E',
        scope:{
           post: "="
        },
        replace: true,
        link: function(scope, element, attrs){

        }
    }
}]);
```

La vue associée dans « views/ui/post.html » :

```
<li>
    <h4>{{post.title}}</h4>
    <h5>{{post.author.first_name}} {{post.author.last_name}}</h5>
    <p>{{post.body}}</p>
</li>
```

Dans la vue principale ajoutons les éléments suivants :

```
<!-- Formulaire de création d'un nouveau post -->
<div id="new-post">
    <div class="form-group">
        <label for="post-title" class="col-sm-2 control-label">Title</label>
        <div class="col-sm-12">
            <input type="text" class="form-control" id="post-title"
placeholder="Title">
        </div>
    </div>
    <div class="form-group">
        <label for="post-title" class="col-sm-2 control-label">Body</label>
        <div class="col-sm-12">
            <textarea class="form-control" id="post-body"
placeholder="Title"></textarea>
        </div>
    </div>
    <button ng-click="post()" class="btn btn-primary">Post</button>
</div>

<!—Liste des posts -->
<ul>
    <demo-post ng-repeat="post in followedPosts" post="post"></demo-post>
</ul>
```

L'exercice serait maintenant de créer une page dédié aux utilisateurs, lorsque l'on click sur ceux-ci, on peut voir le fil de leurs posts.

2.2.4. Vérifier l'accès aux pages et se souvenir de l'utilisateur

Certaines pages sont réservées aux personnes identifiées, il faut donc pouvoir rediriger l'utilisateur vers l'écran de login au cas ou il se dirigerait vers une mauvaise page inutilisable en tant qu'utilisateur anonyme.

Pour réussir cette prouesse, il suffit de charger un bout de code avant la première vue qui se chargera d'analyser les routes. Il faut également pouvoir identifier si une route est libre d'accès ou non.
Rendez-vous dans « app.js ». A chacune de nos routes nous allons ajouter l'information « isFree », si c'est faux, nous sauront qu'un utilisateur anonyme n'a rien à faire là. Exemple :

```
.when('/', {
    templateUrl: 'views/main.html',
    controller: 'MainCtrl',
    access: {
        isFree: false
    }
})
.when('/login', {
    templateUrl: 'views/user/login.html',
    controller: 'LoginCtrl',
    access: {
        isFree: true
    }
})
.when('/logout', {
    templateUrl: 'views/user/logout.html',
    controller: 'LogoutCtrl',
    access: {
        isFree: false
    }
})
```

Nous nous contenterons ici de dire si une route est libre d'accès mais nous pourrions ajouter une clé contenant un tableau de rôle qui ont le droit d'accéder ou non à une url (eg `target-role: [ 'admin', 'manager']`)
Pour filtrer les accès nous allons écrire une directive qui écoute les changements de routes, plaçons là dans le fichier « scripts/directives/demoCheckUser » :

```
demoApp.directive('demoCheckUser', ['$rootScope', '$location', 'User',
function ($root, $location, User) {
    return {
        link: function (scope, elem, attrs, ctrl) {
            $root.$on('$routeChangeStart', function(event, currRoute,
prevRoute){
                if ((!currRoute.access && !User.isLogged) ||
(!currRoute.access.isFree && !User.isLogged)) {
                    // Si on arrive ici, notre utilisateur n'est
                    // pas authentifié et cherche à utiliser une route
                    // non libre
                    if(scope.$$phase || $root.$$phase){
                        $location.path("/login");
                    }else{
                        scope.$apply(function() {
                            $location.path("/login");
                        });
                    }
                }
```

```
            });
        }
    }
}]);
```

Pour que la directive soit effective, il faut, bien évidemment, importer notre nouveau fichier depuis l'« index.html ». Nous devons charger la directive avant la vue en mettant par exemple l'attribut « demo-check-user » sur l'élément « body » de notre page. Pour le tester, il suffit de se rendre sur « #/ » et de voir si vous êtes bien redirigé à la page de login.

On remarquera également que par défaut si le niveau d'accès d'une page n'est pas renseigné, il sera considéré comme non libre.

Ajoutons maintenant de la mémoire à notre site.

Pour cela il faut d'abord vérifier que la dépendance « angular-cookie » est bien satisfaite, pour cela, vérifiez qu'un dossier de ce nom est bien présent dans « bower_components », si ce n'était pas le cas, depuis votre projet exécutez la commande :

```
Bower install angular-cookie
```

Ensuite il faut vérifier que angular-cookie.js est bien importé depuis l'index. Afin que cela fonctionne, il faut également ajouter une dépendance à notre module depuis app.js :

```
var demoApp = angular.module('teamBookApp', [
    ...'ngCookies',...
])
```

Nous sommes maintenant prêts à persister le token de session pour la faire durer au-delà d'un rafraichissement ou de la fermeture de la page !

Nous allons centraliser les contrôles autour du cookie dans la « resource user ».

Il faut indiquer à la resource qu'elle va dépendre du service « $cookie » en l'ajoutant en entrée de fonction :

```
demoApp.factory('User', ["$resource", "$cookies", "$http", function($resource,
$cookies, $http){
```

Ajoutons une option rememberMe à notre fonction login afin de savoir si on souhaite persister la session.

```
var rememberMe = $('#demo-login-form input[type=checkbox]').is(':checked');
User.login = function(login, password, rememberMe, successCb, errorCb){
    var data = {username: login, password: password};
    $http.post(API_URL + 'auth/',data)
        .success(function(data, status, headers, config){
            if(status===200 && data && data.token){
                $http.defaults.headers.common['Authorization'] = 'Token ' +
data.token;
                // Si l'utilisateur le souhaite, nous devons
                // persister le token dans un cookie
                if(rememberMe){
                    $cookies.authToken =
$http.defaults.headers.common['Authorization'];
                }
                setUser(successCb, errorCb);
            }else{
                errorCb(data, status, headers, config);
            }
        })
        .error(errorCb);
```

```
}
```
Il nous faut également une méthode nous permettant de charger l'utilisateur depuis ce cookie :
```
/* nous utilisons toujours des callbacks pour gérer
 *la réussite ou l'échec depuis l'appel à cette méthode.
 */
User.tryToRestoreFromCookies = function(successCb, errorCb){
    if($cookies.authToken){
        $http.defaults.headers.common['Authorization'] = $cookies.authToken;
        setUser(successCb, errorCb);
    }else{
        errorCb(User);
    }
}
```
Enfin lors de la déconnexion de notre utilisateur, il faut penser à supprimer le contenu du cookie.
Ajoutons, dans la fonction « logout », la ligne :
```
delete $cookies.authToken;
```
L'utilisateur doit pouvoir cocher l'option « Remember me » dans le formulaire d'authentification, ajoutons-y le code suivant :
```
<div class="form-group">
    <div class="col-sm-offset-2 col-sm-10">
        <div class="checkbox">
            <label>
                <input type="checkbox"> Remember me
            </label>
        </div>
    </div>
</div>
```
Dans le « controlleur LoginCtrl », nous devons également tenir compte de ce nouveau paramètre :
```
User.login(login, password, rememberMe, //ajout du paramètre rememberMe
    function(){
        if($scope.$$phase || $scope.$root.$$phase){
            $location.path("/");
        }else{
            $scope.$apply(function(){
                $location.path("/");
            });
        }
    },
    function(data, status, headers, config){
        console.log(status);
    })
```
Il ne reste plus qu'un point à traiter, si on ferme la page et qu'on la rouvre à un point nécessitant l'utilisateur, il faut pouvoir le recharger dynamiquement sans passer l'écran de login.
Nous allons faire ceci depuis « demoCheckUser.js » comme suit :
```
demoApp.directive('demoCheckUser', ['$rootScope', '$location', 'User',
function ($root, $location, User) {
    return {
        link: function (scope, elem, attrs, ctrl) {
            $root.$on('$routeChangeStart', function(event, currRoute,
```

```
prevRoute){
                if ((!currRoute.access && !User.isLogged) ||
(!currRoute.access.isFree && !User.isLogged)) {
                    // essayons de retrouver l'utilisateur depuis un cookie
                    User.tryToRestoreFromCookies(
                        // Si ça a marcher nous pouvons ajouter du code
                        // ici, pour par exemple rediriger en fonction
                        // du rôle de l'utilisateur
                        function(user){
                            // Do stuff if some privileges has to be
                            // managed by route
                        },
                        // En cas d'échec nous renvoyons l'utilisateur
                        // à la page de login
                        function(data, status, headers, config){
                            if(scope.$$phase || $root.$$phase){
                                $location.path("/login");
                            }else{
                                scope.$apply(function() {
                                    $location.path("/login");
                                });
                            }
                        });
                }
            });
        }
    }
}]);
```

Attention l'utilisateur ne sera récupéré que si on passe sur une route non libre.
Tester en fermant la page et la rouvrant sur « #/ » !

2.2.5. Ajouter des effets de transition

Une librairie AngularJS nommée « angular-animate » permet d'intégrer des effets de transition en css ou bien javascript. Nous préfèrerons les intégrer en css pour des raisons de performance. N'hésiter pas à consulter la documentation et les exemples afin de voir tout ce qu'on peut et aussi trouver de l'inspiration et customiser vos effets de transition.
Nous animer le changement de page mais aussi customiser des effet d'apparition sur les listes.
Importons la librairie dans « index.html » avec la balise :

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

Ajoutons également une classe à notre div de principal avec l'attribut « ng-view » avec un nom explicite : « view-transition »
Nous devons également ajouter la dépendance à notre application dans app.js :

```
var demoApp = angular.module('teamBookApp', [
    ...'ngAnimate',...
])
```

Maintenant nous n'avons plus qu'a customiser nos classes et voici un exemple de transition par fondu :

```
.view-transition.ng-enter,
.view-transition.ng-leave {
```

```
    -webkit-transition: 400ms cubic-bezier(0.250, 0.250, 0.750, 0.750) all;
    -moz-transition: 400ms cubic-bezier(0.250, 0.250, 0.750, 0.750) all;
    -ms-transition: 400ms cubic-bezier(0.250, 0.250, 0.750, 0.750) all;
    -o-transition: 400ms cubic-bezier(0.250, 0.250, 0.750, 0.750) all;
    transition: 400ms cubic-bezier(0.250, 0.250, 0.750, 0.750) all;
}

.view-transition.ng-leave.ng-leave-active {
    opacity: 0;
}

.view-transition.ng-enter {
    opacity: 0;
}

.view-transition.ng-enter.ng-enter-active,
.view-transition.ng-leave {
    opacity: 1;
}
```

2.2.6. Tester le code

2.2.6.1. Tests unitaires

2.2.6.2. Tests « end to ends »

2.2.7. Déployer son application