HEIG-VD

Haute Ecole d'Ingénieurie et de Gestion du Canton de Vaud

TWEB - TE2

AngularJS & GitHub API

Duchoud Thibaud
Jeudi le 22 Janvier 2016
thibaud.duchoud@heig-vd.ch

Table des matières

1. Introduction

Dans le cadre de ce deuxième TE2 du cours TWEB, il était demandé de créer une application web à l'aide d'AngularJS et utilisant une API reste de notre choix (parmis 3).

J'ai décidé d'utiliser l'API de **GitHub.**

2. Informations utiles

2.1. Informations

- API utilisée : GitHub
- Lien du repo GitHub : https://github.com/Manamiz/TWEB-TE2
- Lien du site sur Heroku : https://evening-retreat-35839.herokuapp.com/
- La marche à suivre pour déployer sur votre PC est affichée sur GitHub (readme)

2.2. API GitHub

L'API GitHub ne permet « que » 1000 requêtes par heure sur son API pour une IP. Une erreur (403) est affichée lorsque se problème survient. Il faut, malheureusement, réessayer plus tard.

Pour la recherche, elle permet 10 requêtes par minute. Le problème ne devrait donc pas survenir.

3. Technologies, librairies ...

Voici la liste des différentes technologies, librarires… utilisées :

- AngularJS
- Angular-animate et Animate.css
  - Animate.css est un fichier css (disponible en ligne) qui contient plusieurs types d'animations. Celles-ci sont utilisés (éventuellement avec Angular-Animate) pour afficher quelques effets.
- Angular-loading-bar
  - Cette libraire affiche une barre de chargement lors d'une requête HTTP faite avec Angular.

    Pour l'utiliser il suffit simplement d'inclure le .js et de l'ajouter dans l'app angular.

    ```
    angular.module('app', ['angular-loading-bar' ...])
    ```

L'affichage se fait automatiquement à chaque requête.

- Ui router
- Bootstrap et Ui Bootstrap
  - Ui Bootstrap permet de mieux gérer certains éléments Bootstrap avec Angular.
- Char js et Angular-chart
- Javascript, HTML, CSS

4. Fonctionnalités

Dans ce projet, diverses fonctionnalités ont été implémentées.

1. Rechercher un utilisateur de GitHub et afficher le résultat de cette recherche.
2. Sélectionner un utilisateur et afficher son profil :
   - ses informatiois personnelles et
   - ses « Repositories ».
3. Sélectionner un repository et afficher différentes informations, statistiques et graphiques concernant le repository sélectionné.

5. Implémentation

L'application WEB est peut être séparée en 3 pages.

- La page de recherche
- La page d'affichage d'un utilisateur
- La page d'affichage d'un repository

5.1. Général

5.1.1. Pages – UI Router

Les différentes pages correspondent simplement à 3 fichiers html différents. Ces 3 fichiers sont chargés à l'aide de UI-Router au moment souhaité.

3 états ont donc été définis pour chacune de ces 3 pages. Les états possèdent, si besoin, un champs controller pour permettre le passage de variable par l'url.

```
app.config(function($stateProvider) {
   // Etat concernant la page utilisateur
   $stateProvider.state('user', {
      url: '/user?username',
      templateUrl: 'views/user.html',
      controller: function($scope, $stateParams, $state) {
         $scope.username = $stateParams.username;
      }
   });

   // Etat concernant la page repository
   $stateProvider.state('repo', {
      url: '/repo?username&repoName',
      templateUrl: 'views/repo.html',
      controller: function($scope, $stateParams, $state) {
         $scope.repoName = $stateParams.repoName;
         $scope.username = $stateParams.username;
      }
   });

   // Etat concernant la page de départ (recherche)
   $stateProvider.state('start', {
      url: '/',
      templateUrl: 'views/start.html'
   });
});
```

5.1.2. Animations

Les animations sont réalisées à l'aide du fichier Animate.css celui-ci, trouvable facilement sur le web, contient différentes animations CSS.

On peut ensuite ajouter ces animations dans des propriétés CSS.

Il y a 2 animations différentes dans ce projet.

1. Lorsque l'on change de page (changement d'état ui-router)
    - Lorsque l'on change de page, on utilise l'attribut ui-view. Il est possible d'ajouter une animation sur lorsqu'on change de vue.

```
/* Affichage de la vue entrante */
[ui-view].ng-enter {
  animation:0.9s fadeInRight ease;
  opacity: 0;
}

/* Affichage de la vue sortante */
[ui-view].ng-leave {
  animation:0.9s fadeOutLeft ease;
  opacity: 1;
}
```

2. Lorsque l'on affiche le résultat de la recherche
    - Utilisant le même principe mais à l'aide de Angular-animate et l'attribut ng-show. Cette animation est principalement voyant lors de l'affichage de la liste des résultats de la recherche.

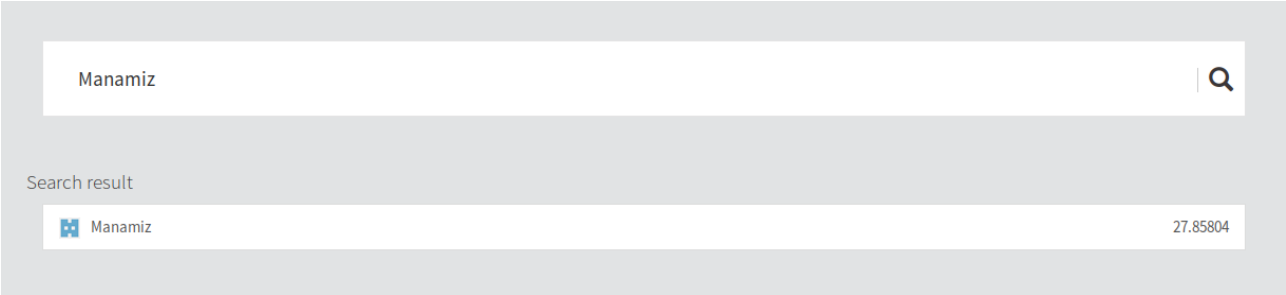
```css
/* Quand ça se cache */
.ng-hide-add { animation:0.9s zoomOut ease; }

/* Quand ça s'affiche */
.ng-hide-remove { animation:0.9s zoomIn ease; }
```

5.1.3. Retour

Sur la page d'un utilisateur et d'un repository se trouve un bouton permettant de revenir à la page de recherche et, seulement sur la page d'un repository, un bouton permettant de revenir à l'utilisateur.

Ces boutons appelent une fonction qui fait un simple $state.go('stateVoulu').

5.2. Page de recherche



Cette page est la première affichée à l'utilisateur. Elle permet de chercher un utilisateur.

Lorsqu'un recherche est faite, on accède à l'API GitHub de recherche et on affiche les résultats sous forme d'une liste.

Recherche dans la Factory

```javascript
// Cherche un utilisateur
githubFactory.searchUser = function(username) {
    return $http.get(urlBase + "/search/users", {
        params: {
            q: username
        }
    });
}
```

Recherche dans le contrôleur

```javascript
// Lance une recherche (requête http)
$scope.search = function() {

    $scope.users = undefined;
    $scope.noResult = false;

    githubFactory.searchUser($scope.username)
        .success(function(data) {
            $scope.users = data.items;

            // Si pas de résultats => affiche 'No result found'
            if($scope.users.length <= 0)
                $scope.noResult = true;
        })
        .error(function(data) {
            alert('(SEARCH USER) an error occured !');
        })

};
```

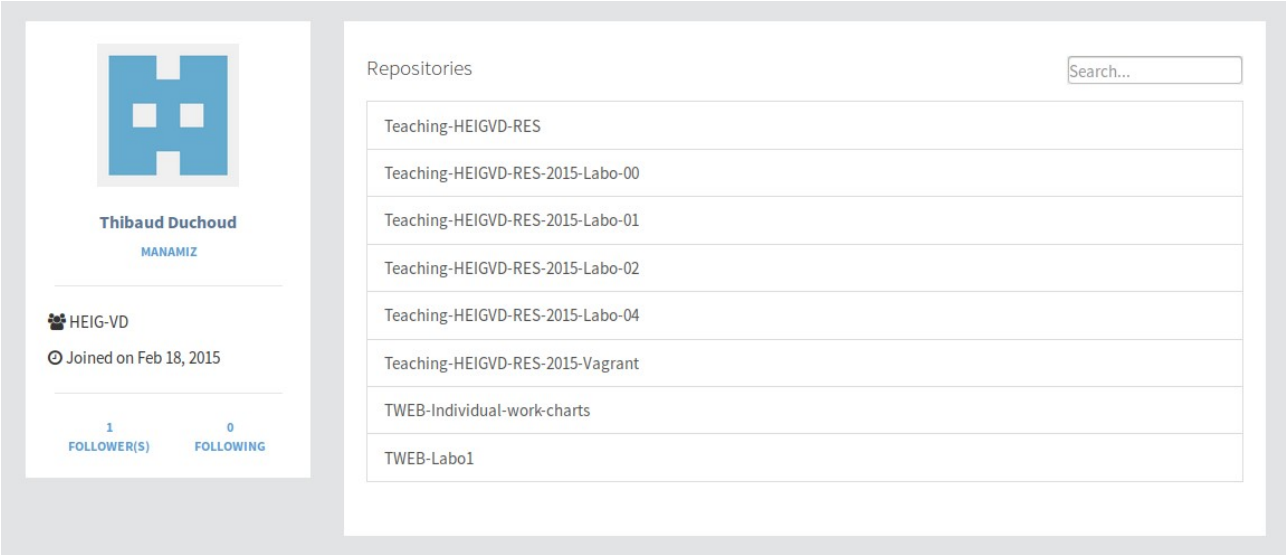
Lorsqu'il n'y a pas de résultats, un message est affiché.

L'utilisateur peut ensuite cliquer sur un utilisateur affiché et accéder à sa page personnelle. Lors de ce click, une méthode qui fait un $state.go('user', {username : username}) est appelée pour rediriger l'utilisateur vers sa page en passant son nom d'utilisateur par l'url.

Aller à l'utilisateur

```
// Va à l'état user (page utilisateur)
$scope.goToUser = function(username) {
   $state.go('user', {
      username: username
   });
}
```

5.3. Page d'un utilisateur



Lorsqu'un utilisateur est redirigé sur cette page, après une requête, les données sont chargées via l'API GitHub. Les requêtes sont faites au chargement de la page. La fonction loadUserInfos est appelée dans l'attribute (directive) ng-init.

Elles sont ensuite ajoutées au scope et affichées à l'écran.

L'utilisateur peut ensuite filtrer les repositories à l'aide du champ texte « Search... » qui est un simple filtre Angular.

Il peut ensuite cliquer sur un repository se trouvant dans la liste et il sera rediriger vers la page du repository correspondant.

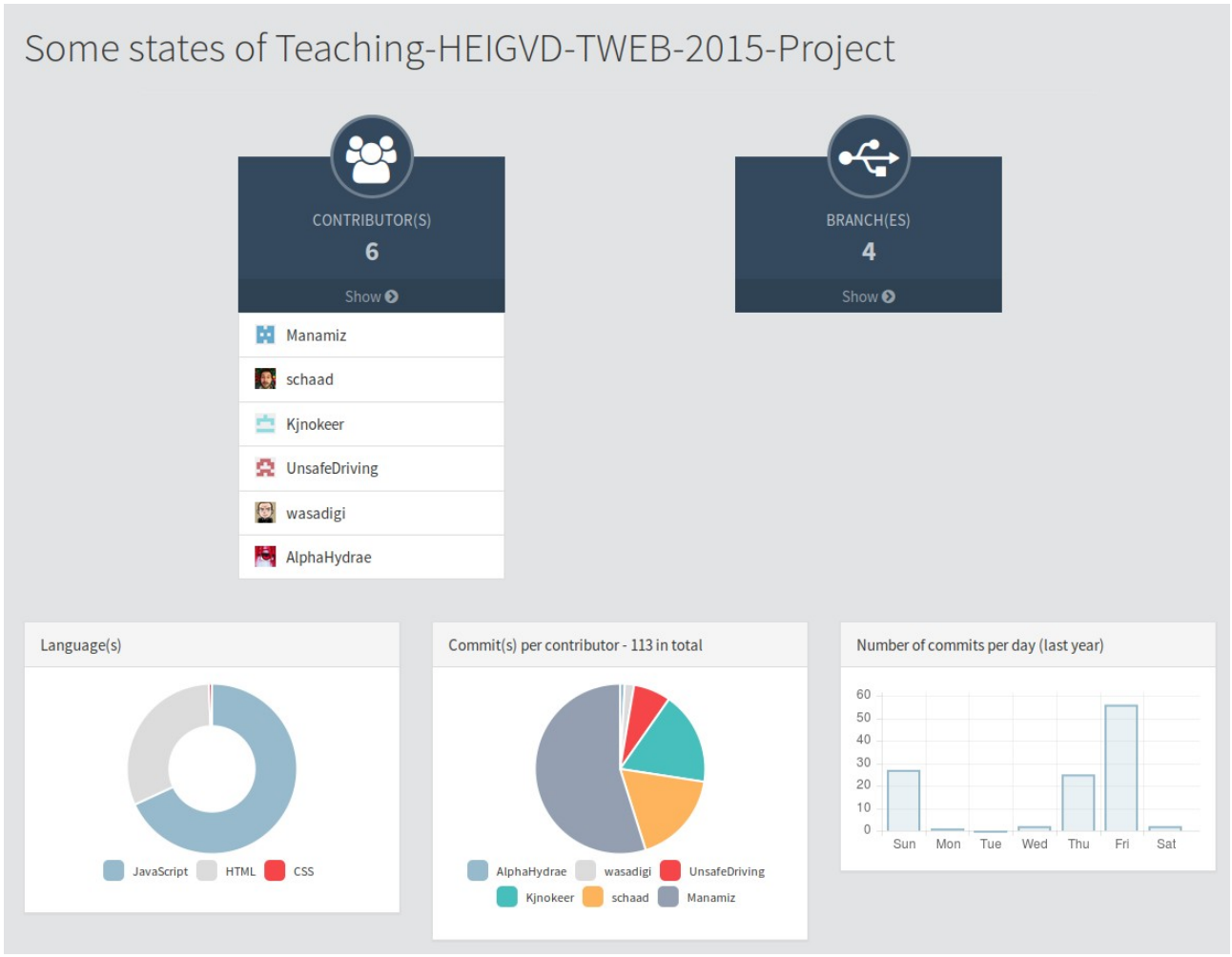
Requêtes dans la Factory

```
// Récupère un utilisateur
githubFactory.getUser = function(username) {
    return $http.get(urlBase + "/users/" + username);
}

// Récupère les repositories d'un utilisateur
githubFactory.getUserRepositories = function(username) {
    return $http.get(urlBase + "/users/" + username + "/repos");
}
```

Requêtes dans le contrôleur

```
// Charge les infos de l'utilisateur (toutes les requêtes sont faites ici)
$scope.loadUserInfos = function() {

    githubFactory.getUser($scope.username)
        .success(function(data) {
            $scope.user = data;
        })
        .error(function(data) {
            alert('(GET USERS) An error occured !');
            $scope.backToSearch();
        });

    githubFactory.getUserRepositories($scope.username)
        .success(function(data) {
            $scope.repos = data;

            if($scope.repos.length <= 0)
                $scope.noRepo = true;
        })
        .error(function(data) {
            alert('(GET USERS REPOS) An error occured !');
            $scope.backToSearch();
        });

}
```

5.4. Page d'un repository



Cette page récupère les différentes informations concernant le repositoy choisi et les affiche.

Cette récupération est faite, comme pour utilisateur, lors d'un chargement de la page. La méthode loadRepoInfos est appelée dans l'attribut (directive) ng-init.

Les graphiques sont implémentés simplement en récupérant les données via l'API GitHub sans modification. Elles sont poussées dans les tableaux data et labels correspondant au graphique.

Simplement pour le nombre de commits par jour pour l'année passée. L'API GitHub nous renvoie la liste des semaines contenant un tableau de 7 cases (les 7 jours) et chaque case contient le nombre de commits pour le jour correspondant. En commençant par le dimanche. (il y a d'autres informations concernant la semaine typiquement mais qui ne sont pas utiles ici)

Voici un exemple simple :

> Objet (simplifié) reçu par l'API. Le premier tableau est la première semaine, le deuxième tableau la deuxième semaine ...
>
> => { [10, 0, 0, 0, 0, 3, 0], [1, 2, 3, 4, 5, 6, 7] ... }
>
> On a donc 10 commits le dimanche, 0 le lundi, mardi, mercredi, samedi et 3 le vendredi …

Voici l'implémentation :

Requête vers l'API dans la Factory

```
// Récupère les statistiques des commits d'un repository
githubFactory.getRepoStatsCommitActivity = function(username, repoName) {
    return $http.get(urlBase + '/repos/' + username + '/' + repoName + '/stats/commit_activity');
}
```

Requête dans le contrôleur

```
githubFactory.getRepoStatsCommitActivity($scope.username, $scope.repoName)
    .success(function(data) {
        $scope.commits = data;

        // Création du graphique des commits par jour l'année dernière
        angular.forEach(data, function(value, key) {
            var i = 0;

            // On additione les commits de chaque jour qu'on ajoute aux datas du graphique
            angular.forEach(value.days, function(value) {
                $scope.commitsActivityData[i] += value;
                i++;
            })
        });
    })
    .error(function(data) {
        alert('(GET REPOS COMMIT) An error occured !');
        $scope.backToSearch();
    });
```

5.5. Angular

5.5.1. Directive

Lors de la recherche, il peut être interessant d'appuyer simplement sur la touche Enter, après avoir entré l'utilisateur à recherché, pour lancer la recherche.

Comme je n'ai pas utilisé de *Form*, j'ai décidé de créer une directive permettant d'appeler une fonction lors de l'appui sur la touche Enter.

J'ai créé une directive qui va bind les événements sur *keydown* et *keypress* sur l'élément. Ensuite je check le code de l'événement et si c'est Enter (13), j'appel ce qui se trouve dans l'attribut myEnter qui est, en l'occurence, la fonction que je souhaite appeler.

On place ensuite simplement cette directive sur l'élément que l'on souhaite (ajout de l'attribut my-enter).

Directives myEnter et exemple d'utilisation

```
// Directive utilisée lorsque l'on souhaite appeler une fonction
//  lors de l'appui sur la touche Enter sur un élément
app.directive('myEnter', function() {
    return function(scope, element, attrs) {
        element.bind('keydown keypress', function (event) {
            if(event.which === 13) {
                scope.$eval(attrs.myEnter);
                event.preventDefault();
            }
        });
    };
});

// Exemple d'utilisation.
// La méthode search va être appelée lors de l'appui sur la touche Enter
<input my-enter="search()" ng-model="username" type="text" />
```

5.5.2. Factory

Pour l'utilisation d'une API, j'ai décidé d'utiliser une Factory.

Celle-ci permet de regrouper toutes les fonctions qui concernent l'API que j'utilise et donc d'y stocker au même endroit pour tous les contrôleurs les urls de l'API.

Ça permet de centraliser les requêtes faîtes à l'API et les contrôleurs souhaitant y accéder peuvent le faire simplement en y injectant la Factory.

Exemple de factory (pas complet)

```
// Factory regroupant les requêtes faites à l'API GitHub
app.factory('githubFactory', ['$http', function($http) {

    var urlBase = 'https://api.github.com';
    var githubFactory = {};

    githubFactory.searchUser = function(username) {
        return $http.get(urlBase + "/search/users", {
            params: {
                q: username
            }
        });
    }

    githubFactory.getUser = function(username) {
        return $http.get(urlBase + "/users/" + username);
    }

    githubFactory.getUserRepositories = function(username) {
        return $http.get(urlBase + "/users/" + username + "/repos");
    }

    return githubFactory;
}]);
```

Utilisation d'une factory

```
app.controller('RepoCtrl', ['$scope', '$http', '$state', 'githubFactory',
    function($scope, $http, $state, githubFactory) {

    githubFactory.getUser($scope.username)
        .success(function(data) {
            $scope.user = data;
        })
        .error(function(data) {
            alert('(GET USERS) An error occured !');
        });
```

5.5.3. Contrôleurs

Il y a un contrôleur par page.

1. MainCtrl : Contrôleur utilisé pour la page d'index (recherche).
    - Il contient la méthode de recherche* qui va chercher les utilisateurs correspondant à celui entré et les afficher. Il contient également la méthode permettant d'aller vers la page utilisateur.
2. UserCtrl : Contrôleur utilisé par la page d'affichage d'un utilisateur.
    - Il contient les méthodes permettant la navigation entre pages (retour à la recherche et aller vers la page repository). Il contient également la méthode d'initialisation* qui va récupérer les données correspondantes à l'utilisateur. Et les mettre dans le scope afin de les afficher.
3. RepoCtrl : Contrôleur utilisé par la page d'affichage d'un repository.
    - Il contient les méthodes permettant la navigation entre pages (retour à la recherche et retour à l'utilisateur). Il contient également une méthode d'initialisation* qui va récupérer les données correspondantes au repository, les travailler si besoin et les mettre dans le scope afin de les affichées. Sous forme de graphique ou non.

* Les méthodes qui passent par la Factory pour accéder à l'API GitHub. Les méthodes qui font des requêtes HTTP à l'API GitHub donc.