# Enoncé

Suite à la présentation de **Matthis Duclos** sur le framework **Backbone.js**, j’ai voulu essayer de développer la même application, mais cette fois-ci basée sur le framework **angularjs**.

Pour cela, j’ai utilisé à l’identique le backend développé par Matthis en Node.js.

# Outils

J’ai utilisé le générateur angularjs de **Yeoman** pour générer la structure de mon projet.

Pour une présentation des outils **npm**, **bower** et **grunt**, je vous recommande la documentation écrite par Matthis.

# Structure du répertoire de travail

Yeoman nous génère donc la structure de notre projet :

banquesqli-front-angular01

|-- app

|-- node\_modules

|-- test

Le répertoire **app/** contient tout ce qui sera déployé en prod :

app

|-- bower\_components

|-- scripts

|-- styles

|-- views

**scripts/** > les fichiers JS

**styles/** > les feuilles de style CSS

**views/** > les fichiers HTML

# Structure commune à toutes les IHM

|  |  |
| --- | --- |
| Menu du haut | |
| Menu de gauche | Contenu |

Cette structure est définie directement dans **app/index.html**

On indique la zone de contenu grâce à l’attribut ng-view :

<!-- Barre du haut -->

<div ng-include src="'views/navbar-top.html'"></div>

<div>

<!-- Menu de gauche -->

<div ng-include src="'views/menu-left.html'"></div>

<!-- Zone de contenu -->

<div **ng-view** class="animate-fade"></div>

</div>

# Le fichier app.js

On y définit l’ensemble de nos routes.

Une URL = un fichier HTML + un contrôleur

Ex : when('/admin/account/:number', {

templateUrl: 'views/admin/editAccount.html',

controller: 'AdminEditAccountCtrl'

})

La portion HTML présente dans le fichier editAccount.html sera insérée **dans la div ng-view**.

# Système d’authentification

L’application backend nous impose l’utilisation d’un **token de sécurité** pour chacune de nos requêtes. Ce token est renvoyé suite à une authentification réussie.

## Redirection vers la page de login

Première chose à faire, il faut s’assurer que lorsque l’utilisateur n’est pas authentifié, il ne puisse pas accéder à une page sécurisée.

(app.js)

angular.module('banquesqliAngular01App')

.run(function ($rootScope, $location) {

// A chaque changement de route

$rootScope.$on('$routeChangeStart', function (event, next, current) {

// Si l’utilisateur n’est pas authentifié

// et qu’il essaye d’accéder à une page autre que le login

if (!$rootScope.isLogged() && next.templateUrl != 'views/login.html') {

// On le redirige vers la page de login

$location.path('/login');

}

});

});

## Stockage du token de sécurité

Aussitôt reçu, le token de sécurité est stocké dans un service partagé. Il est aussi sauvegardé dans la zone mémoire sessionStorage du navigateur.

## Envoi du token de sécurité

Le token de sécurité doit être présent dans chacune des requêtes envoyées au backend, sous peine de recevoir une erreur.

Ceci est fait grâce à un **intercepteur** qui ajoute le token en paramètre de la requête.

On enregistre cet intercepteur dans la configuration de l’application, dans le fichier app.js.

## Token invalide

Un token n’est valide que durant un temps limité. Il faut dans ce cas rediriger l’utilisateur vers la page de login pour qu’il s’authentifie à nouveau.

Là encore, l’utilisation d’un intercepteur facilite les choses.

# Inconvénients

## Couche métier

Nos objets métier sont mélangés au reste de l’application. La documentation officielle n’indique aucune règle concernant la couche métier, le développeur est complètement libre.

De simples objets JS peuvent être utilisés.

Très souvent, nos objets métier correspondent à nos ressources.

# Pistes d’amélioration

* Requirejs - Avantages ?
* Essayer la réorganisation du projet en fonctionnalités
* Généraliser la validation de formulaire et créer des directives pour générer les messages d’erreur dans le DOM
* Edition d’un RIB au format PDF