AngularJS

Test 2 - TWEB

David Villa

Introduction

Le but de ce test était d'évaluer nos connaissances concernant un jeu de technologies web utilisables côté client tel qu'AngularJS.

Pour ce faire, plusieurs sujets de travail étaient proposés. Pour ma part j'ai choisi le sujet 2. A savoir :

Développer une application AngularJS pour afficher du contenu obtenu via l'API GitHub

Il s'agissait d'utiliser l'API de GitHub, de proposer une visualisation des données au sein de l'UI puis de déployer cette application sur Heroku ainsi que sur un repo GitHub.

Conception

API GitHub

La prise en main des informations misent à disposition par l'API de GH était plutôt aisée. GH proposant une bonne documentation.

Après quelques tests, il s'est vite avéré que GH apposait une restriction concernant le nombre de requêtes pouvant être traitées par heure lors d'une utilisation anonyme.

Pour pallier à cette restriction, j'ai utilisé un détour, pas spécialement propre mais efficace, en liant une application à GH et en utilisant le token générée pour cette dernière pour effectuer mes requêtes. Ainsi au lieu d'être limité à 60 req/h, je pouvais effectuer 5000 req/h par utilisateur.

Vue

Mon application est décomposée en 3 vues, à savoir :

- Main
- User
- Repo

La vue Main fournit la vue de la page d'accueil, elle est très simpliste et explique en quelques mots ce que l'application propose.

La vue User utilise quelques informations de l'API de GH concernant l'utilisateur et propose la liste des repos sur lesquels l'utilisateur contribue. Un affichage alternatif est proposé en cas d'échec de la requête.

La vue Repo utilise quelques informations de l'API de GH concernant le repo choisit ainsi qu'un graphique représentant le nombre de commit respectif des différents contributeurs du repo.

Constant, routes et controllers

Mon application comporte quatre contrôleurs, à savoir :

- main
- user
- repo
- search

Actuellement le contrôleur main ne sert à rien et sa présence est plus un oubli de suppression qu'autre chose. On peut expliquer sa présence pour préparer une amélioration future !

Le contrôleur user est associé à la vue User, il s'occupe de récupérer le nom de l'utilisateur passé à l'aide de l'URL puis d'effectuer une requête à l'API de GH (users/:login) afin de récupérer les informations concernant ce dernier.

Si l'utilisateur est actif sur, au moins, un repo, une seconde requête est effectuée à l'API afin de récupérer la liste des repo auquels il participe (users/:login/subscription).

Les informations retournées par l'API sont stockées dans le scope du contrôleur pour une utilisation au sein de la vue qui lui est associée.

Le contrôleur repo est associé à la vue Repo, il s'occupe de récupérer les informations concernant le repo sélectionné dans la vue User à l'aide d'une requête à l'API (repos/:login_creator/:repo_name).

En cas de succès de la requête précédente, une seconde requête à l'API (repos/:login_creator/:repo_name/contributors) est effectuée afin de récupérer les informations sur les utilisateurs ayant contribué au développement du repo.

Une fois encore, les informations retournées par l'API sont stockées dans le scope du contrôleur pour une utilisation au sein de la vue qui lui est associée.

Le contrôleur search est associé, quant à lui, à la navbar. Il fournit une fonction search(username) permettant d'effectuer une redirection vers la vue User lorsque le bouton est cliqué.

Mon module possède aussi un service contenant une constante, à savoir, l'URL vers l'API de GH.

Finalement, j'utilise $routeProvider pour configurer les redirections vers les différentes vue ainsi que l'association des contrôleurs aux vues.

Conclusion

Un problème que j'avais déjà rencontré durant le projet de TWEB s'est de nouveau manifesté durant l'élaboration de ce petit projet. Ce problème étant de savoir comment partager des variables entre différents contrôleurs sans avoir besoin d'utiliser l'URL ou autre instance tel que $localStorage ou $globalScope.

Après discussion avec des camarades de classe j'ai pris connaissance des services d'Angular. Malheureusement je n'ai pas eu le temps de les mettre en application dans ce laboratoire et j'ai donc conservé le transfert de variable à l'aide de l'URL entre deux contrôleurs. Ce qui n'est pas spécialement propre.

Pareil pour ce qui des requêtes. Dans mon projet toutes les requêtes s'effectuent à l'aide de $http et ses différentes méthodes. L'utilisation de l'objet $resource d'Angular serait plus propre et permettrait d'économiser un certain nombre de requête tout en pouvant être placé au sein d'un service afin d'être partagé entre les différents contrôleurs. Mais une fois encore, j'ai pris connaissance de ceci tardivement et n'ai pas eu le temps de l'utiliser au sein de ce projet.

Ce qui m'impressionne chaque fois que j'utilise Bootstrap est la possibilité d'avoir un rendu graphique agréable sans pour autant avoir des connaissances poussées en design.

Les fonctionnalités ne se courent pas après, mais de mon avis le but de ce projet était de tester notre connaissance des technologies et non la capacité de développer une application évoluée dans un cours laps de temps.

Pour conclure, je dirais que mon projet est peut-être un peu minimaliste mais il est fonctionnel et utilise les différentes technologies vue en cours, ou leur équivalent tout en ayant une interface simple et agréable.