STACKOVERFLOW LIKE

Rapport

Rapport de projet pour StackOverflow Like (SegFault) web site

PARTIE II

Elysée Jonas

Rabérin Alexandre

Sommaire

Ce projet constitue la suite du projet "StackOverFlow" rendu précédemment. Le but principal est d'enrichir la version précédente de notre projet en rajoutant des fonctionnalités afin de rendre notre projet *scalable*, dynamique et résistant aux pannes. Pour cela, une division de notre projet en une API et une application cliente (Angular) est nécessaire.

A – Fonctionnalités

Afin de réaliser notre projet, le projet a été séparé en deux sous-parties :

- Une partie cliente
- Une partie serveur qui devient une API.

La partie serveur se charge de renvoyer des données s'adaptant à l'état de ses différents services, et la partie client se charge d'afficher les données renvoyées par le serveur. Les nouvelles fonctionnalités implémentées sont les suivantes :

JWT

Le Java Web Token (**JWT**) est un élément clé qui permet de rendre notre projet stateless. Il permet d'identifier un utilisateur sans stockage de données au niveau des instances. Dans le cas où nous avons plusieurs instances de notre API, le JWT permet de stocker des données chez le client (au niveau de son navigateur) afin que chaque service récupère ses informations afin d'identifier l'utilisateur.

Ce token contient le nom d'utilisateur, sa date de création, ainsi que sa date d'expiration.

Features Flipping

Principe et mise en place

Le **features flippling** est une fonctionnalité qui permet d'adapter le contenu de la vue en fonction de fonctionnalités activées ou non (soit des données renvoyées par le serveur au client) selon certains booléens désignant des fonctionnalités spécifiques. Ainsi, le programmeur peut facilement rajouter ou retirer du contenu de la vue s'il le souhaite. Cela peut servir dans le cas où une fonctionnalité du serveur est défaillante ou en cours de production (exemple : accepter l'ajout d'un commentaire sur une question).

Le features flipping se fait dans notre projet au moyen d'un serveur de configuration lancé auparavant. Au lancement de l'application API, elle interroge le serveur de configuration afin de récupérer l'ensemble des fonctionnalités actuellement disponible, puis renvoi des données en accord avec ces options. Ceci permet de gérer les configurations sur de multiple instances de manière synchrone.

Serveur de Configuration

Le serveur de configuration est la troisième entité de notre projet. Elle permet d'initialiser des valeurs de configuration pour les instances de notre API, de manière générale ou spécifique. On peut de cette manière gérée des configurations spécifiques pour des environnements de production, de tests ou de développement.

Dans notre projet, il n'y a qu'une configuration générale aux différentes instances de notre API. Le serveur retransmet les valeurs de la configuration qu'il a pu récupérer sur un dépôt **Git** hébergé sur

Github. Cela permet donc de gérer le features flipping simplement en effectuant un push sur le dépôt Git de la configuration.

Health Check

Le **Health Check** est une mécanique de notre service à s'auto analyser afin de déterminer son état et donc de savoir s'il est complètement opérationnel ou non. Ainsi, nous disposons à présent d'une URL technique qui permet l'analyse des différentes composantes de notre API et permettrait à un système de monitoring de gérer les cas d'instances ayant un état de « santé » incorrecte.

Il est donc possible de récupérer ce status via la route /health de notre API. La route fournit une réponse qui établit un détail des composants traités et leur statuts respectifs. Dans notre cas nous pouvons retrouver des éléments tel que l'état du Circuit Breaker, l'état d'occupation du disque, l'état du serveur de configuration ainsi qu'un module de test de réponses de notre API réalisé par nous-même.

Circuit Breaker & Graceful Degredation

Le design pattern **Circuit Breaker** permet d'adapter le résultat renvoyé au client dépendamment de l'état de notre service. En effet, si le client fait une requête à l'API et qu'elle n'est pas en mesure de répondre, une réponse par défaut est renvoyée à l'utilisateur en guise de remplacement.

Dans notre application, nous avons implémenter un service qui permet de récupérer une phrase aléatoire depuis une API tierce. Il s'agit d'une API sur laquelle en donnant un nombre, celle-ci nous renvoie une information à propos de ce nombre. Cette phrase est affichée sur la page d'accueil de notre site. Le circuit breaker mis en place permet de contrôler la réponse renvoyée au client et évite l'envoi brutale d'une erreur. Dans le cas où l'API tierce ne répond pas ou qu'un autre problème survient, nous avons pris soin de mettre en place une réponse par défaut issue du service tiers. Ceci nous permet donc de dégradé notre service sans pour autant le rendre inopérant en remplaçant une phrase aléatoire par une phrase fixe mais fonctionnelle.

II – Nouveautés du projet

API Grails

Le projet Grails de la première partie a été revue pour être transformé en une API utilisant le JSON comme représentation des ressources envoyées ou reçues. Il n'est donc plus une application web statefull comme auparavant car il permet l'utilisation d'un token d'authentification (JWT), permettant d'implémenter une logique stateless et scalable. Ainsi, toute les vues qui avaient étaient créées dans ce projet sont obsolètes et inutiles remplacées par une application AngularJS. La logique d'authentification utilise donc Spring Security REST comme remplaçant.

Client Angular

La partie affichage des informations (vues) a été séparée de ce qui formait initialement notre serveur avec l'application. Cette séparation a donné lieu à la création d'une application cliente en Angular qui vient requêter l'API Grails pour traiter les réponses JSON qui lui sont fournies. Nous avons donc ré-implémenter l'intégralité de la partie affichage de notre projet grâce à cette technologie. Cela nous permet d'avoir un client dynamique adapté aux changements apportés à notre serveur qui devient une API qui peut être requêté en mode de manière stateless.

Cette application a été réalisée en AngularJS, et non AngularJS 2. Elle se décompose en plusieurs modules faisant appels à des services dédiés. L'application supporte l'internationalisation grâce à angular translate.

Marshallers

Du côté de l'API, il nous a été nécessaire de mettre en place un procédé permettant de contrôler le contenu des réponses fournies par l'API. Afin que les données renvoyées par nos contrôleurs soient adaptées aux différentes vues de notre nouveau client, nous avons créé des marshallers au niveau de notre serveur API. Cela permet notamment d'envoyer seulement les données nécessaires à l'affichage et donc de ne pas charger la réponse JSON avec des données inutiles. Par exemple, sur la page d'accueil où nous affichons les questions les plus récentes posées sur le site. Dans ce cas il est inutile d'avoir l'intégralité des réponses et commentaires de la question. Pour cela nous contrôlons la quantité d'information envoyé en fonction de la route de l'API appelée. Pour ceci, les marshallers nous permettent d'implémenter des configurations spécifiques pour une domaine classe donnée.