Atelier 8
Sécuriser l'application

Sécurisez ZUUL

Une multitude de possibilités s'offre à nous afin de **sécuriser notre API.** Cela va de l'authentification basique par mot de passe à l'authentification par tokens, en passant par les filtres de ZUUL.

Nous allons implémenter dans ce chapitre une authentification basique par mot de passe.

Comme ZUUL est notre point d'entrée unique vers nos Microservices, il suffit de sécuriser son accès.

Nous allons donc utiliser **Spring Security** qui offre des mécanismes d'authentification prêts à l'emploi très facilement.

Ajoutez donc Spring Security à ZUUL :

pom.xml

```
  <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-security</artifactId>
    </dependency>
```

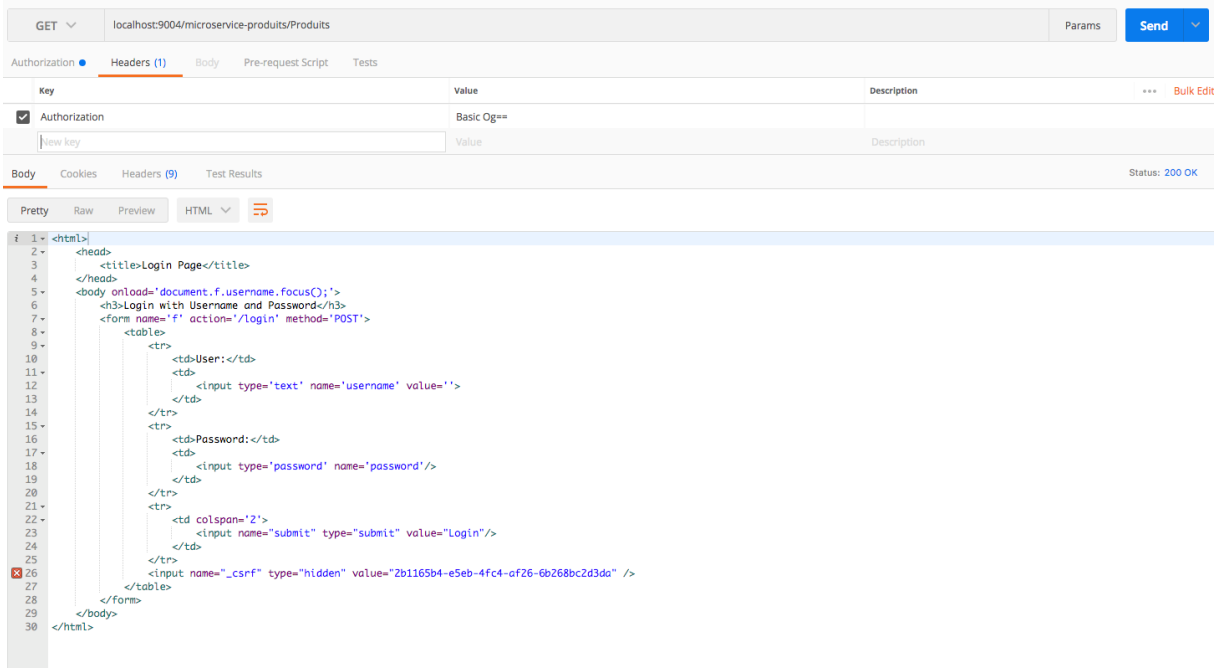
Nous allons maintenant définir un nom d'utilisateur et un mot de passe. Pour cela, rendez-vous dans ***zuul-server.properties*** **dans le dépôt GIT** et modifiez-le comme suit :

```
server.port 9004

#Eureka
eureka.client.serviceUrl.defaultZone: http://localhost:9102/eureka/

#Spring Security
spring.security.user.name=utilisateur
spring.security.user.password=mdp
```

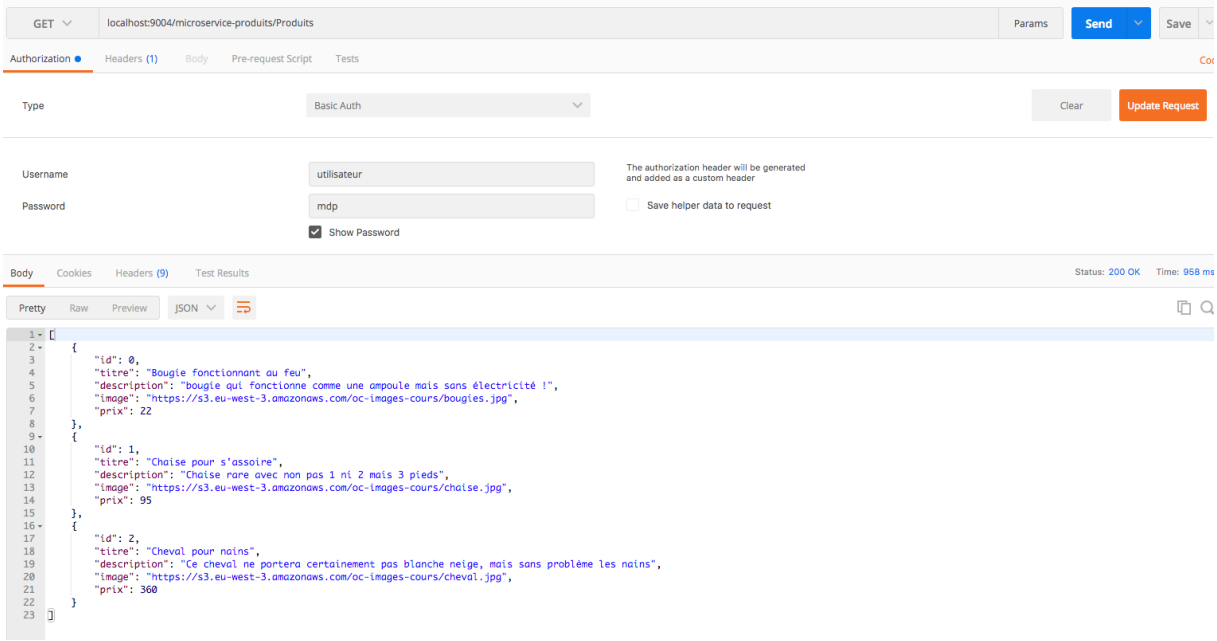
ZUUL est maintenant sécurisé, il ne répondra qu'aux requêtes présentant ces informations d'authentification.

Pour tester, rendez-vous dans Postman et **appelez Microservice-produits via ZUUL :** localhost:9004/microservice-produits/Produits.

Vous recevez alors en retour un formulaire HTML d'authentification :



Afin d'authentifier vos requêtes, rendez-vous dans **l'onglet "Authorization"** et choisissez "Basic Auth". Entrez ensuite les identifiants :



Réexécutez la requête ; vous recevez alors une réponse JSON, comme prévu.

2. Consommez des Microservices sécurisés

Si vous essayez, à nouveau, de lancer le client, vous constaterez que l'application ne marche pas.

Afin d'ajouter les informations d'authentification à utiliser, nous allons créer un **bean de configuration** de Feign.

Créez une classe ***FeignConfig*** dans un package "configuration" :

```java
package com.clientui.configuration;

import feign.auth.BasicAuthRequestInterceptor;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class FeignConfig {

    @Bean
    public BasicAuthRequestInterceptor mBasicAuthRequestInterceptor(){
        return  new BasicAuthRequestInterceptor("utilisateur", "mdp");
    }

}
```

On retourne un objet de type `BasicAuthRequestInterceptor` avec les informations d'authentification que Feign utilisera lors de la génération des requêtes.
Relancez les Microservices ; vous devriez alors constater que tout fonctionne correctement.

Résumé de la partie 2

Les **Edge Microservices** sont des Microservices qui s'occupent principalement de l'orchestration des Microservices métier contenant la logique de l'application.

Feign est un outil permettant de faire communiquer les Microservices très simplement entre eux, en générant automatiquement les requêtes HTTP adéquates à partir de classes appelées proxies.

Les fichiers de configuration des Microservices peuvent être externalisés vers un **dépôt GIT,** par exemple grâce à **Spring Boot Config.** Cet Edge Microservice va alors récupérer la dernière version de chaque fichier de configuration et la servir au Microservice correspondant.

Pour forcer l'actualisation des données d'un fichier de configuration dans un Microservice, il suffit d'envoyer une **requête POST** vers l'**endpoint/refresh** exposé par **Actuator.**

Eureka est un autre Edge Microservice qui permet de garder un registre de toutes les instances disponibles des Microservices métier. Chaque Microservice qui intègre Eureka ira s'enregistrer à chaque démarrage ou lancement d'une nouvelle instance. Eureka vérifie ensuite régulièrement que les Microservices sont toujours disponibles afin de mettre à jour son registre.

Ribbon est un équilibreur de charge côté client. Une fois intégré dans un Microservice, il est capable de communiquer automatiquement avec Eureka afin de choisir l'instance à appeler d'un Microservice donné. Un algorithme s'occupe ainsi de répartir les requêtes sur toutes les instances disponibles.

Lorsque vous avez un nombre important de Microservices, **ZUUL** se positionne comme un Microservice servant de point d'entrée unique aux autres Microservices. C'est ce que l'on appelle une **API Gateway.**

ZULL va permettre ainsi de réaliser des opérations communes à tous les Microservices, comme la sécurisation de l'accès, la mise en place d'un nombre d'appels limite à l'API, la transformation des requêtes afin de les enrichir avant qu'elles n'atteignent les Microservices cibles, etc.