

Formation OAuth2

Développement du Client dans l'architecture oAuth2.

Sommaire

I. Objectif	3
II. Pré-requis	3
III. Architecture de oAuth2	3
IV. Création de l'application au niveau du GITHUB	4
V. Création de l'application au niveau de Google	6
VI. Développement de l'application CLIENT	8
1. Création du projet Maven	8
2. Le fichier pom.xml	9
3. La classe de démarrage MainApplication.....	10
4. La classe de démarrage SecurityConfig	10
5. La classe contrôleur HomeController	11
6. Le fichier application.properties	11
VII. Tests	12

I. Objectif

L'objectif de ce TP est de vous expliquer l'architecture de la norme d'autorisation oAuth2 et vous montrer comment développer le Client.

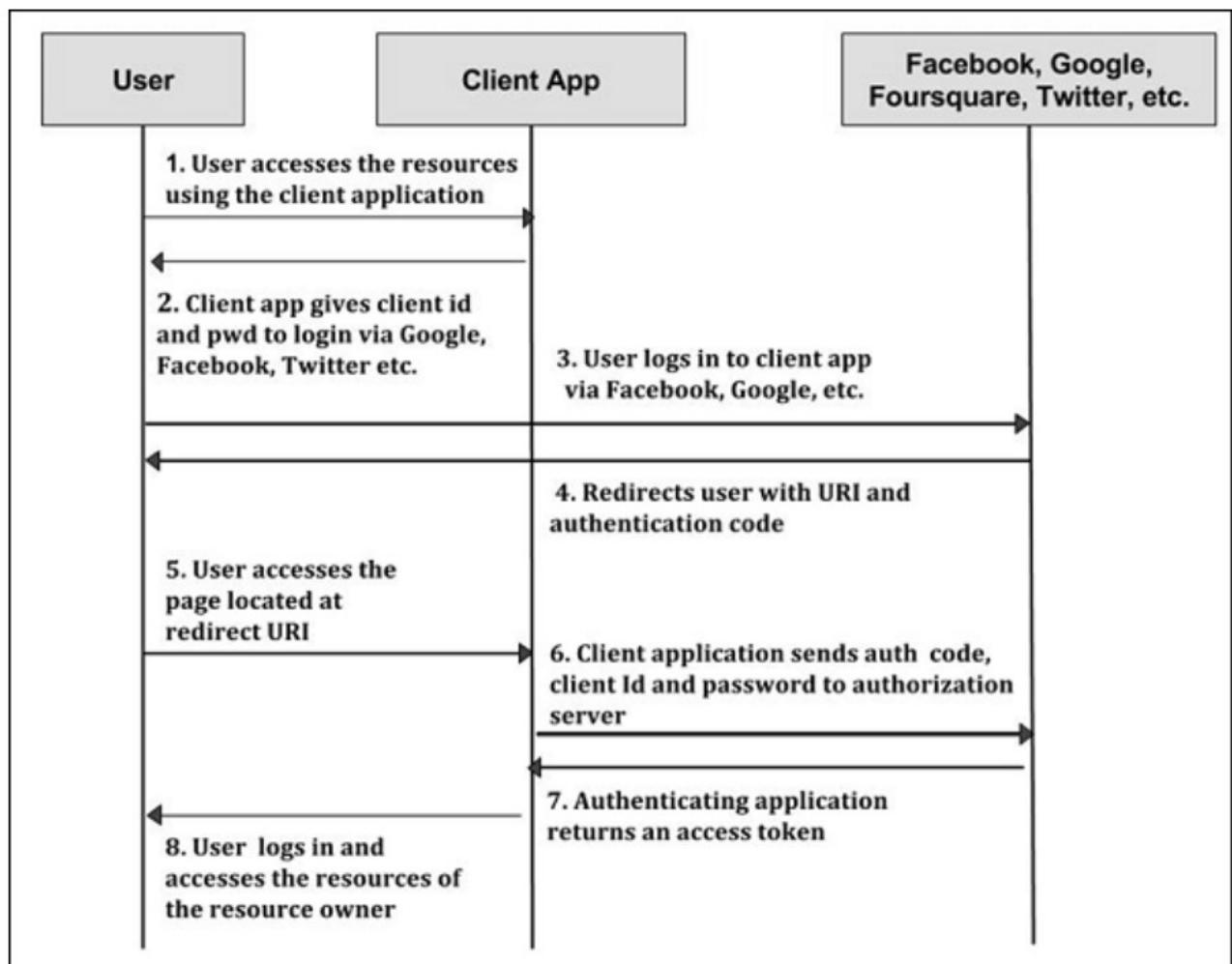
II. Pré-requis

Pour réaliser ce TP vous aurez besoin de :

- JDK 17.
- Spring Boot 3.
- Eclipse ou autre IDE ;
- Connexion à Internet pour permettre à Maven de télécharger les dépendances.

III. Architecture de oAuth2

Le schéma suivant vous montre l'architecture d'oAuth2 :



Explications :

- **Étape 1** : Tout d'abord, l'utilisateur accède aux ressources à l'aide de l'application cliente telle que Google, Facebook, Twitter, etc.
- **Étape 2** : Ensuite, l'application cliente recevra l'identifiant client et le mot de passe client lors de l'enregistrement de l'URI de redirection (Uniform Resource Identifier).
- **Étape 3** : L'utilisateur se connecte à l'aide de l'application d'authentification. L'ID client et le mot de passe client sont propres à l'application client sur le serveur d'autorisation.
- **Étape 4** : Le serveur d'authentification redirige l'utilisateur vers un URI (Uniform Resource Identifier) de redirection à l'aide d'un code d'autorisation.
- **Étape 5** : L'utilisateur accède à la page située à l'URI de redirection dans l'application cliente.
- **Étape 6** : L'application client recevra le code d'authentification, l'identifiant client et le mot de passe client, et les enverra au serveur d'autorisation.
- **Étape 7** : L'application d'authentification renvoie un jeton d'accès à l'application cliente.
- **Étape 8** : Une fois que l'application cliente obtient un jeton d'accès, l'utilisateur commence à accéder aux ressources du propriétaire de la ressource à l'aide de l'application cliente.

IV. Création de l'application au niveau du GITHUB

- Aller au lien <https://github.com/settings/applications/new>

github.com/settings/applications/new

Register a new OAuth application

Application name *
tp-oauth2-exemple
Something users will recognize and trust.

Homepage URL *
http://localhost:8080
The full URL to your application homepage.

Application description
Application description is optional
This is displayed to all users of your application.

Authorization callback URL *
http://localhost:8080
Your application's callback URL. Read our OAuth documentation for more information.

Enable Device Flow
Allow this OAuth App to authorize users via the Device Flow.
Read the [Device Flow documentation](#) for more information.

Register application **Cancel**

- Dans « **Application name** », entrer le nom de votre application.
- Dans « **Homepage URL** », entrer l'URL suivante : <http://localhost:8080>
- Dans « **Authorization callback URL** », entrer l'URL suivante : <http://localhost:8080/login/oauth2/code/github>
- Cliquer sur « **Register application** ». La page suivante est affichée :

General tp-oauth2-exemple

Optional features

Advanced

 **abbouformations** owns this application. **Transfer ownership**

You can list your application in the [GitHub Marketplace](#) so that other users can discover it. **List this application in the Marketplace**

0 users **Revoke all user tokens**

Client ID
17d8f3b15a27ab7009bb

Client secrets **Generate a new client secret**

You need a client secret to authenticate as the application to the API.

- Noter le Client ID.

- Cliquer sur le bouton « **Generate a new client secret** » pour générer le mot de passe, ce dernier est généré comme illustré au niveau de l'écran ci-dessous :

Client secrets

Generate a new client secret

Make sure to copy your new client secret now. You won't be able to see it again.

Client secret

✓ 32754f1fa41246b73d3adaa0181b5f399ee522b8 [Copy](#)

Added now by abboformations

Never used

You cannot delete the only client secret. Generate a new client secret first.

Delete

V. Création de l'application au niveau de Google

- Aller au lien <https://console.cloud.google.com/>
- Créer un projet (par exemple : tp-oauth2-exemple).

API API et services

API et services activés

Bibliothèque

Identifiants

Écran d'autorisation OAuth

Page des accords d'utilisation

Pour afficher cette page, sélectionnez un projet.

CRÉER UN PROJET

- Aller à l'écran de consentement, voici le lien : <https://console.cloud.google.com/apis/credentials/consent?project=tp-oauth2-exemple>
- Cocher « Externe » et cliquer sur Créer.
- Entrer : le nom de l'application, l'adresse email d'assistance utilisateur, l'adresse email du développeur et cliquer ensuite sur « Enregistrer et continuer ».
- Cliquer ensuite sur **Identifiants** et cliquer sur « **CREER DES IDENTIFIANTS** », la liste déroulante suivante s'affiche :

API API et services

Identifiants

+ CRÉER DES IDENTIFIANTS SUPPRIMER RESTAURER DES IDENTIFIANTS SU

Créer des identifiants pour

N'oubliez pas de sauvegarder vos modifications.

Clés API

Aidez-moi à choisir

ID client OAuth

Compte de service

Nom

Aucune clé API à afficher.

- Cliquer sur « ID client OAuth ».
- Cliquer sur « Configurer l'écran de consentement OAuth».
- Dans « Type d'application », choisir « Application Web »
- Entrer le nom de l'application.
- Dans « **Origines JavaScript autorisées** », entrer l'URL de votre application : <http://localhost:8080>

Dans «URI de redirection autorisé », entrer l'URL suivante :

<http://localhost:8080/login/oauth2/code/google>

ID client pour Application Web SUPPRIMER

Origines JavaScript autorisées ?

À utiliser avec les requêtes provenant d'un navigateur

URI 1 *

[+ AJOUTER UN URI](#)

URI de redirection autorisés ?

À utiliser avec les requêtes provenant d'un serveur Web

URI 1 *

[+ AJOUTER UN URI](#)

Cliquer ensuite sur Créer, le CLIENT-ID et le CLIENT-SECRET seront générés comme expliqué ci-dessous :

Client OAuth créé

Vous pouvez toujours accéder à l'ID client et au code secret depuis la section "Identifiants" de la page "API et services"

! L'accès OAuth est réservé aux [utilisateurs de test](#) listés sur votre [écran d'autorisation OAuth](#)

ID client	493710773457-e5gp3dq543i7415l03gf0sobov3t5gen.app.s.googleusercontent.com	Copier
Code secret du client	GOCSXP-Tf26dKHm3LJqmFn2FyPjgWCxRFRL	Copier
Date de création	8 juin 2023 à 15:02:44 GMT+1	
État	Activée	

[TÉLÉCHARGER LE FICHIER JSON](#)

OK

VI. Développement de l'application CLIENT

1. Création du projet Maven

Créer un projet Maven par exemple **oauthsociallogin**. L'arborescence finale du projet est :



2. Le fichier pom.xml

- Modifier le fichier pom.xml comme suit :

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.0.1</version>
    <relativePath/>
  </parent>
  <groupId>ma.formations.security.oauth</groupId>
  <artifactId>oauthsociallogin</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>oauthsociallogin</name>
  <description>Authentication using a social web site</description>
  <properties>
    <java.version>17</java.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-oauth2-client</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
      <groupId>org.projectlombok</groupId>
      <artifactId>lombok</artifactId>
      <optional>true</optional>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
        <configuration>
          <excludes>
            <exclude>
              <groupId>org.projectlombok</groupId>
```

```
<artifactId>lombok</artifactId>
</exclude>
</excludes>
</configuration>
</plugin>
</plugins>
</build>
</project>
```

Explication :

- Remarquer que nous avons utilisé Spring Boot version 3.0.1.
- La JDK doit être 17.
- Pour développer le CLIENT dans l'architecture d'oAuth2, nous avons besoin des deux dépendances suivantes :

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-oauth2-client</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

3. La classe de démarrage MainApplication

La classe de démarrage est :

```
1 package ma.formations.security.oauth;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class MainApplication {
8
9     public static void main(String[] args) {
10         SpringApplication.run(MainApplication.class, args);
11     }
12 }
13 }
14 }
```

4. La classe de démarrage SecurityConfig

- Créer la classe **SecurityConfig** suivante :

```

1 package ma.formations.security.oauth.configuration;
2
3 import org.springframework.context.annotation.Bean;
4 import org.springframework.context.annotation.Configuration;
5 import static org.springframework.security.config.Customizer.withDefaults;
6 import org.springframework.security.config.annotation.web.builders.HttpSecurity;
7 import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
8 import org.springframework.security.web.SecurityFilterChain;
9
0 @Configuration
1 @EnableWebSecurity
2 public class SecurityConfig {
3
4     @Bean
5     SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
6         return http.authorizeHttpRequests(auth-> {
7             auth.requestMatchers("/").permitAll();
8             auth.anyRequest().authenticated();
9
0         }).oauth2Login(withDefaults())
1             .formLogin(withDefaults())
2             .build();
3     }
4 }
-
```

5. La classe contrôleur HomeController

```

1 package ma.formations.security.oauth.presentation;
2
3 import org.springframework.web.bind.annotation.GetMapping;
4 import org.springframework.web.bind.annotation.RestController;
5
6 @RestController
7 public class HomeController {
8
9     @GetMapping("/")
10     public String home() {
11         return "hello, home !";
12     }
13
14     @GetMapping("/secured")
15     public String secured() {
16         return "hello, secured !";
17     }
18
19 }
```

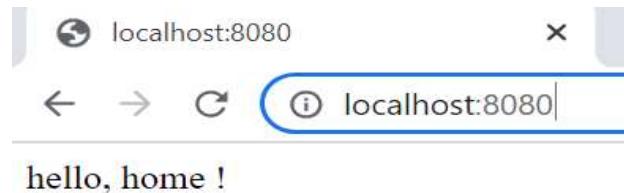
6. Le fichier application.properties

Modifier le fichier **application.properties** comme suit :

```
1logging.level.org.springframework.security=trace
2
3#google
4spring.security.oauth2.client.registration.google.clientId=YOUR GOOGLE CODE ID APP
5spring.security.oauth2.client.registration.google.clientSecret=YOUR GOOGLE SECRET APP
6
7#githhub
8spring.security.oauth2.client.registration.github.client-id=YOUR GITHUB CODE ID APP
9spring.security.oauth2.client.registration.github.client-secret=YOUR GITHUB SECRET APP
```

VII. Tests

- Aller au lien <http://localhost:8080/>, remarquer la ressource est bien consultée :

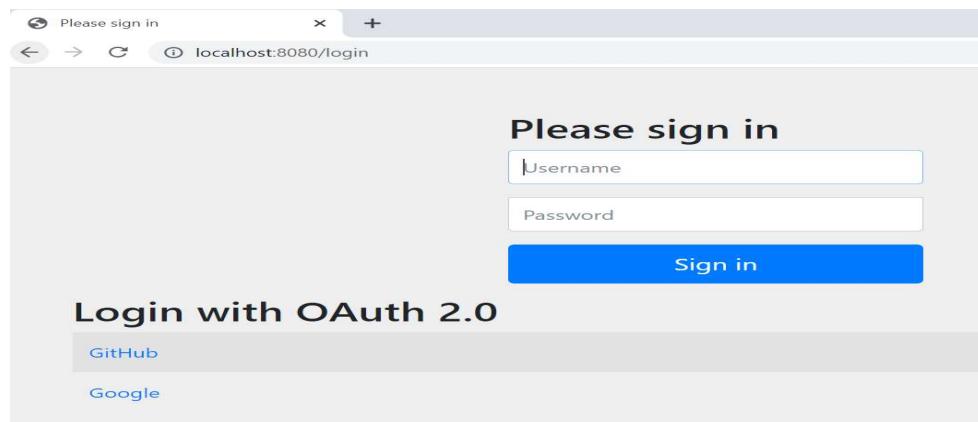


Explication :

- Au niveau de la classe SecurityConfig, la ressource « / » peut être accédée par tout le monde :

```
@Bean
SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    return http.authorizeHttpRequests(auth-> {
        auth.requestMatchers("/").permitAll();
```

- Aller au lien <http://localhost:8080/secured>, la page par défaut suivante s'affiche :

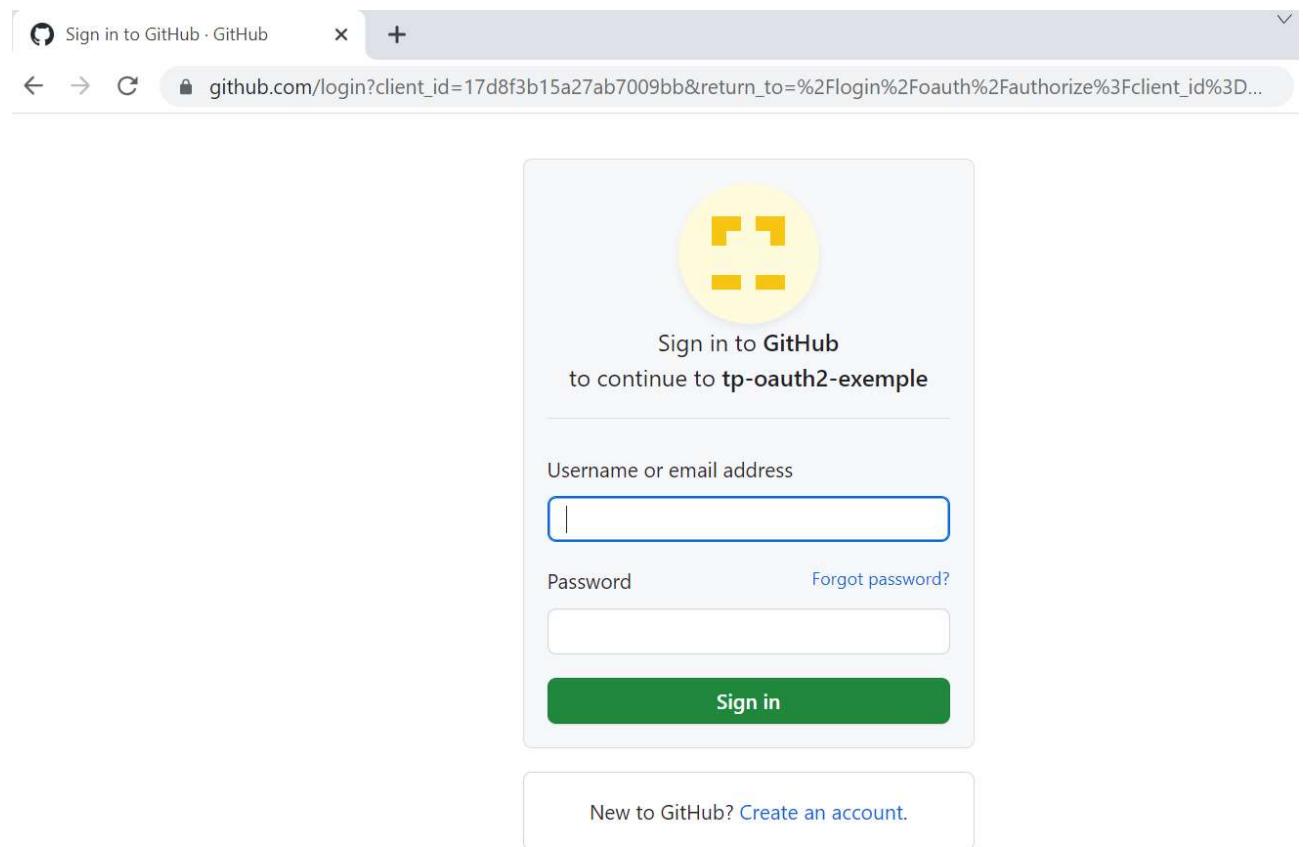


Explications :

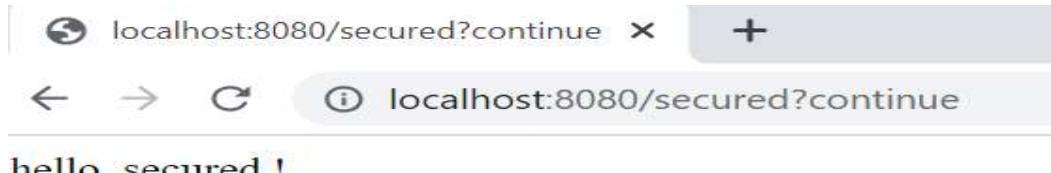
- Au niveau de la classe SecurityConfig, nous avons configuré Spring Security de telle sorte d'afficher la page d'authentification par défaut :

```
}).oauth2Login(withDefaults())
    .formLogin(withDefaults())
    .build();
```

- Cliquer sur GitHub, l'application redirige l'utilisateur vers la page d'authentification de GitHub :



- Entrer vos identifiants et cliquer sur « Sign in ». La page suivante s'affiche :



- Idem pour Google.