Formation OAuth2

Développement du Client dans l'architecture oAuth2.

Sommaire

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.

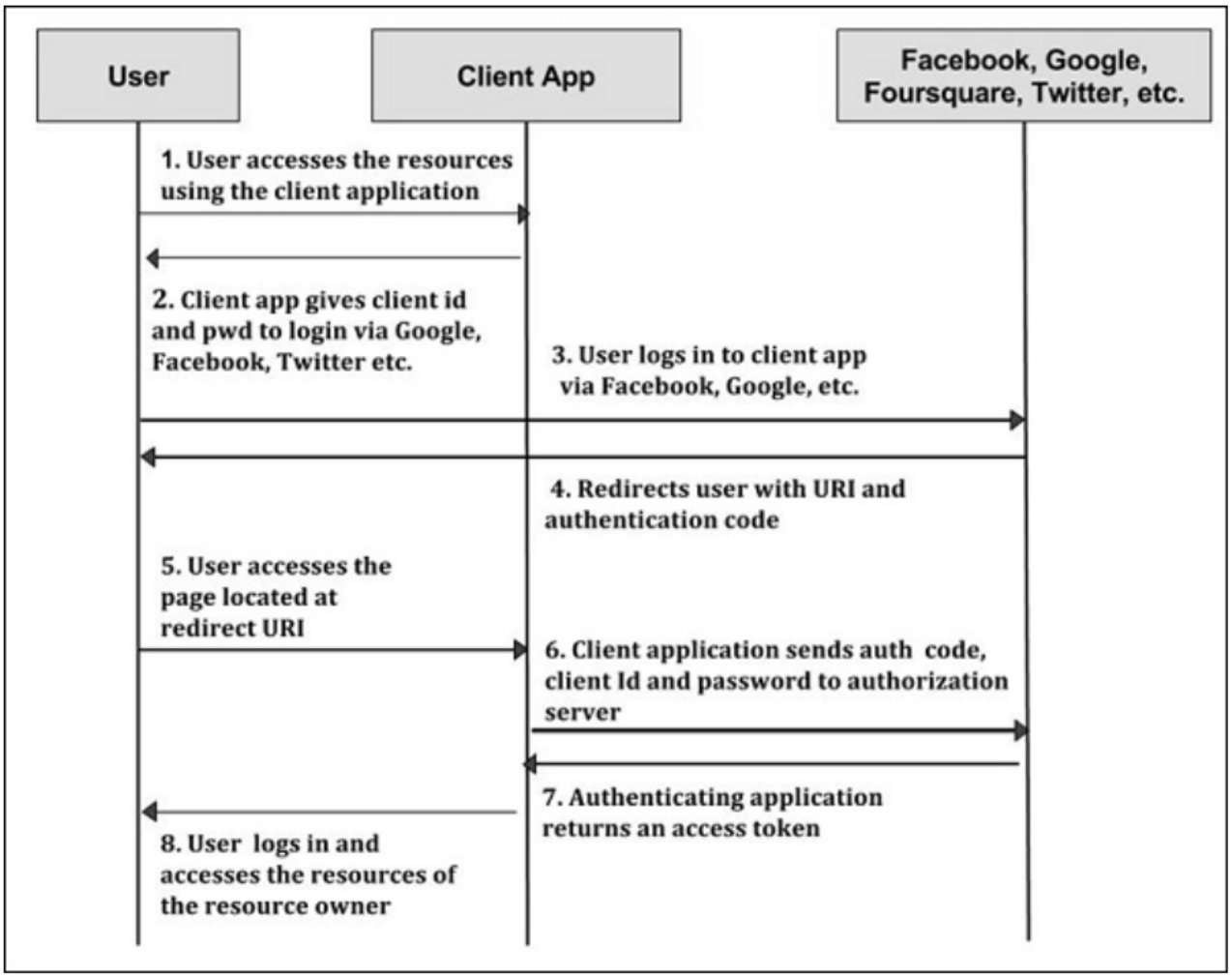
II. Pré-requis

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 :

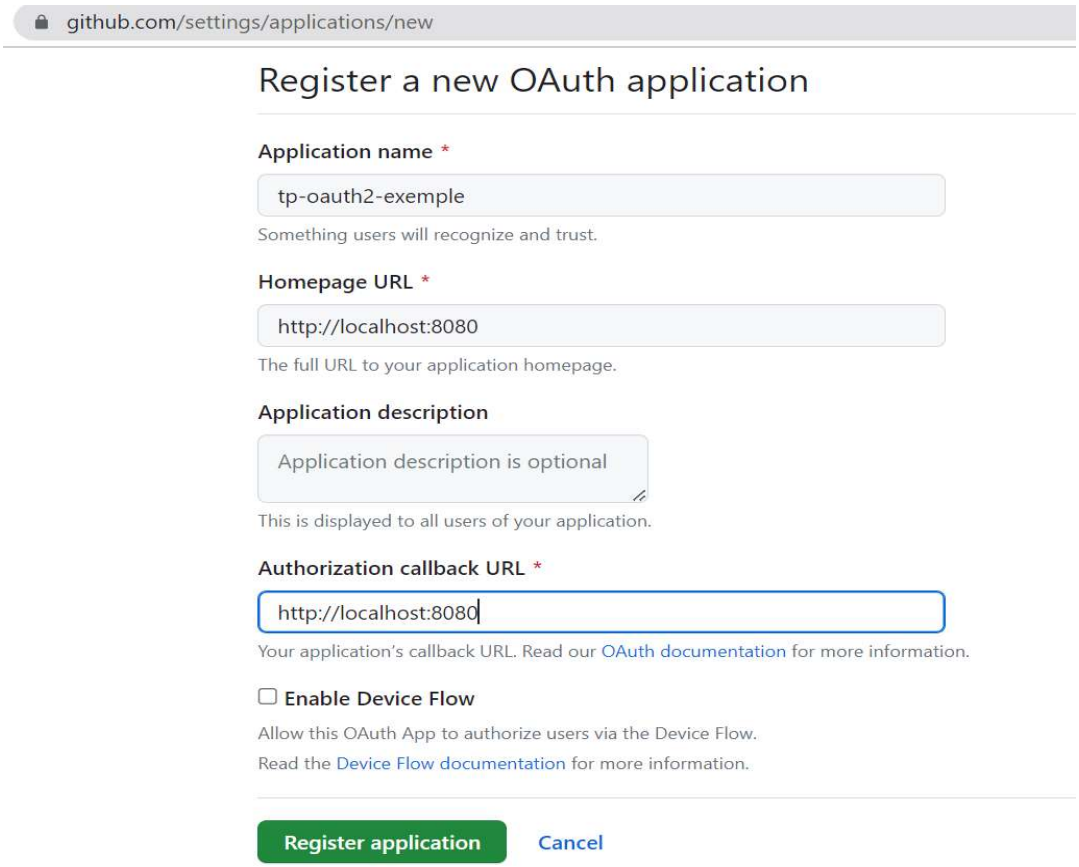


Explications :

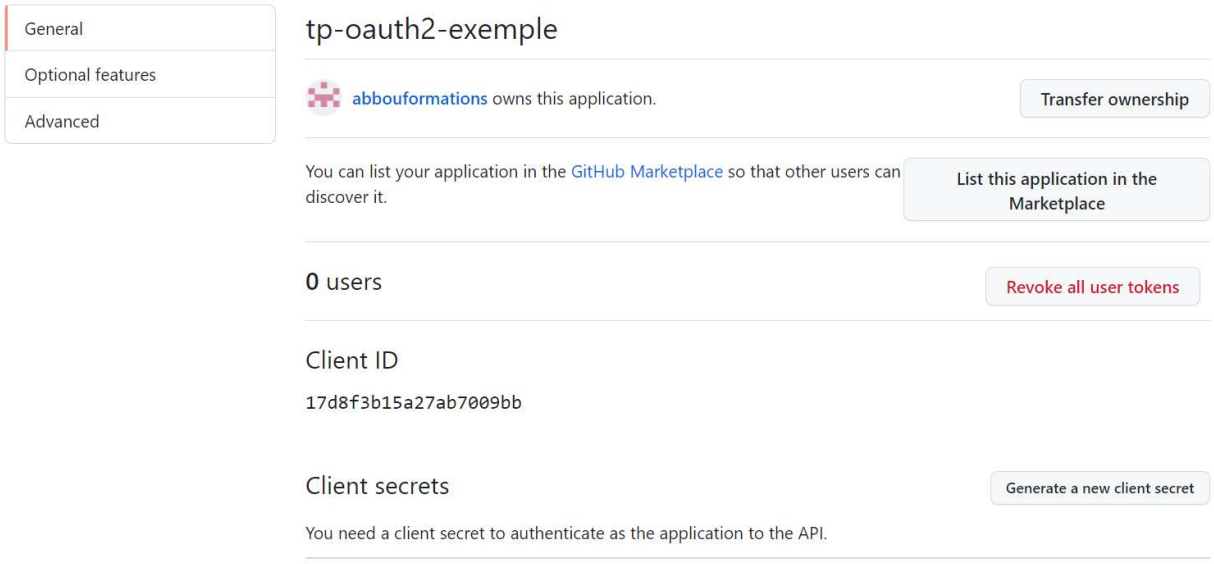
- **Étape 1** : Tout d'abord, l'utilisateur accède aux ressources à l'aide de l'application cliente telle que Google, Facebook, Twitter, etc.
- **Étape 2** : Ensuite, l'application cliente recevra l'identifiant client et le mot de passe client lors de l'enregistrement de l'URI de redirection (Uniform 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

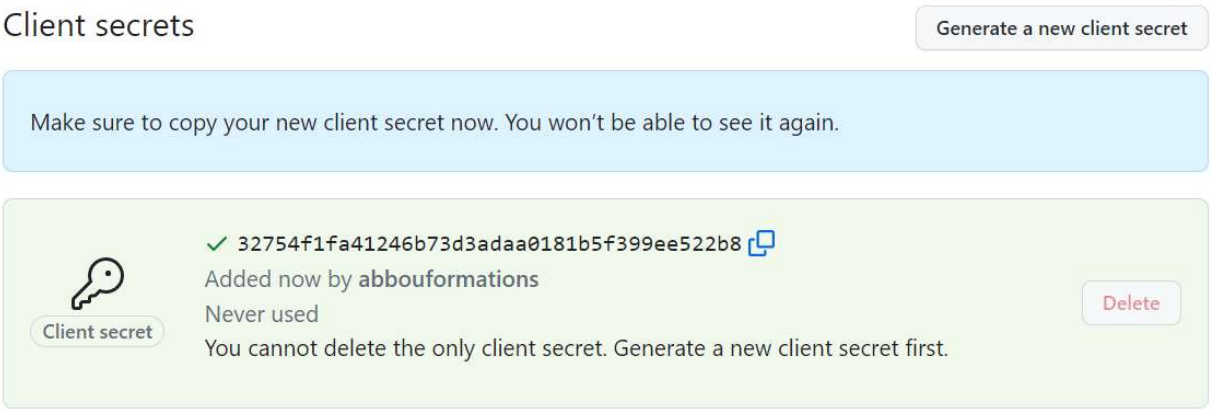


- Dans ***« Application name »***, entrer le nom de votre application.
- Dans ***« Homepage URL »,*** entrer l'URL suivante : http://localhost:8080
- Dans « ***Authorization callback URL » ,*** entrer l'URL suivante : http://localhost:8080/login/oauth2/code/github
- Cliquer sur ***« Register application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



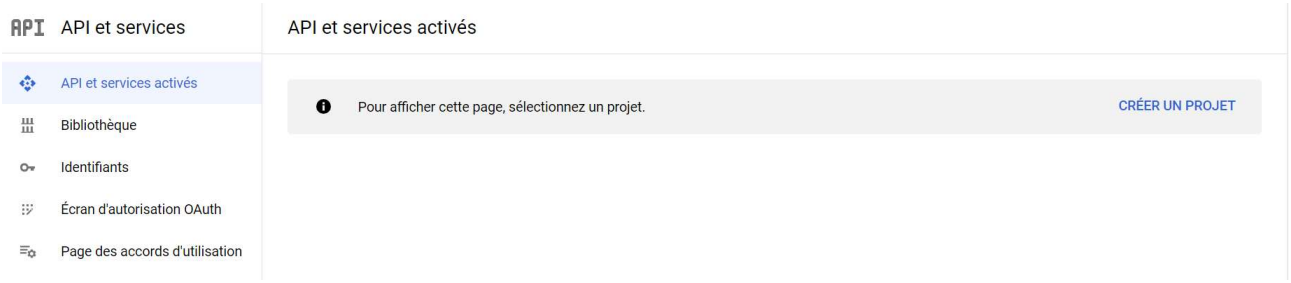
- Noter le Client ID.

- 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 :



V. Création de l'application au niveau de Google

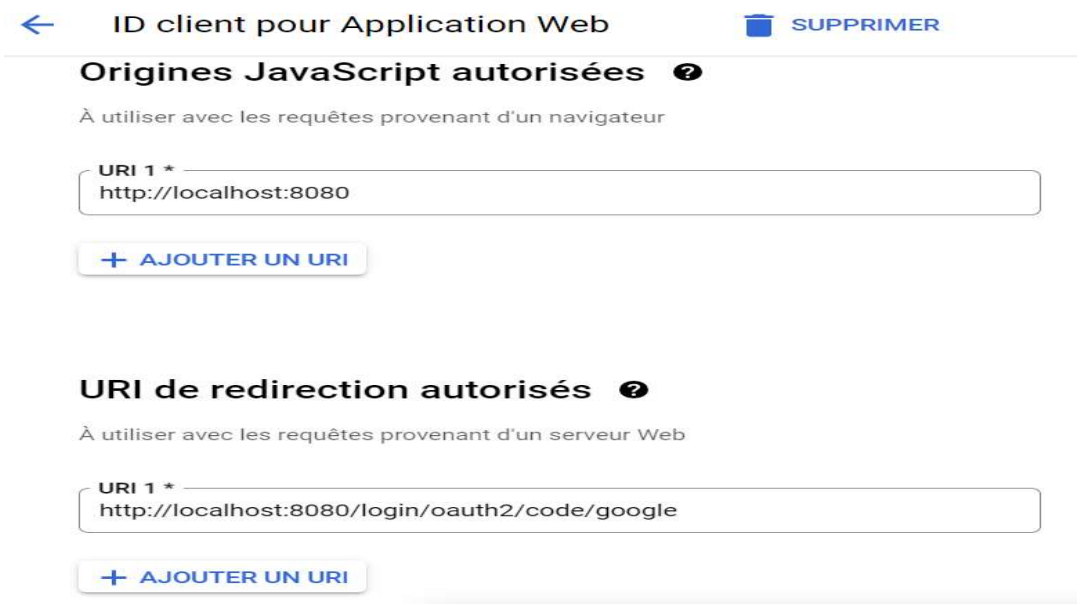
- Aller au lien **https://console.cloud.google.com/**
- Créer un projet (par exemple : tp-oauth2-exemple).



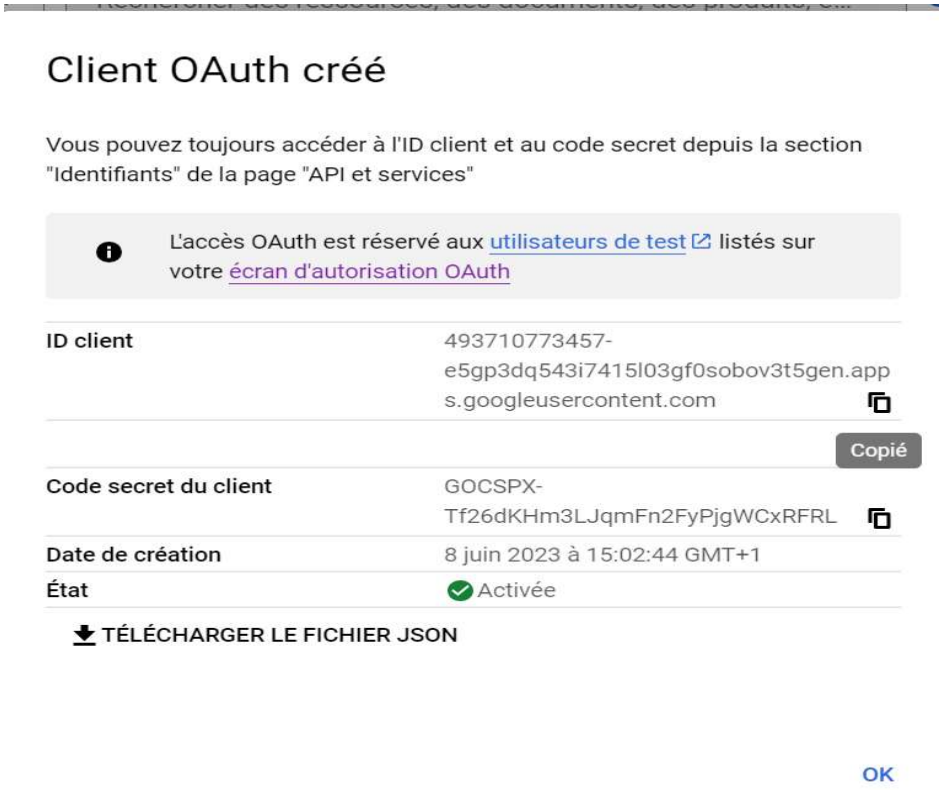
- Aller à l'écran de consentement, voici le lien : https://console.cloud.google.com/apis/credentials/consent?project=tp-oauth2-exemple
- Cocher « Externe » et cliquer sur Créer.
- Entrer : le nom de l'application, l'adresse email d'assistance utilisateur, l'adresse email du développeur et cliquer ensuite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

- 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



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



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 :

```java
package ma.formations.security.oauth.configuration;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import static org.springframework.security.config.Customizer.withDefaults;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.web.SecurityFilterChain;

@Configuration
@EnableWebSecurity
public class SecurityConfig {

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

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

5. La classe contrôleur HomeController

```java
package ma.formations.security.oauth.presentation;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HomeController {

    @GetMapping("/")
    public String home() {
        return "hello, home !";
    }

    @GetMapping("/secured")
    public String secured() {
        return "hello, secured !";
    }

}
```

6. Le fichier application.properties

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

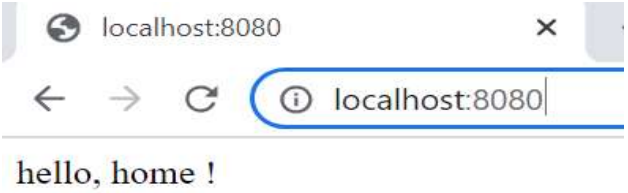
```
logging.level.org.springframework.security=trace

#google
spring.security.oauth2.client.registration.google.clientId=YOUR GOOGLE CODE ID APP
spring.security.oauth2.client.registration.google.clientSecret=YOUR GOOGLE SECRET APP

#github
spring.security.oauth2.client.registration.github.client-id=YOUR GITHUB CODE ID APP
spring.security.oauth2.client.registration.github.client-secret=YOUR GITHUB SECRET APP
```

VII. Tests

- Aller au lien http://loclahost:8080/, remarquer la ressource est bien consultée :

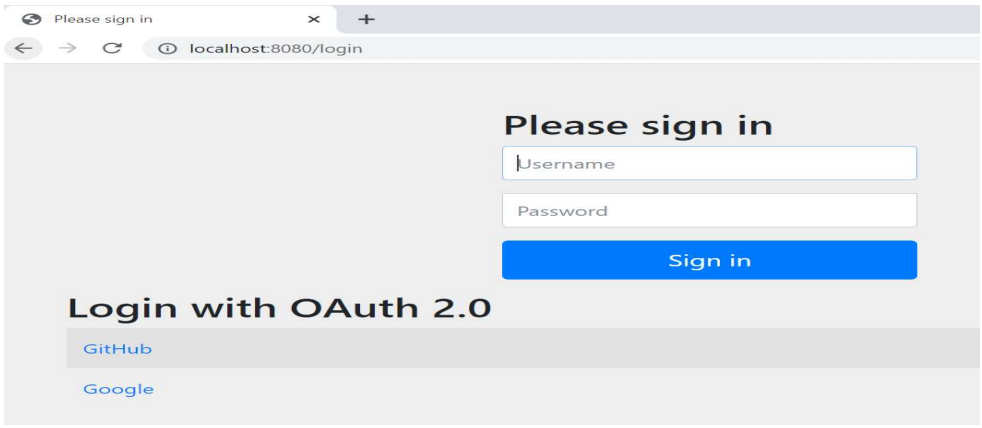


Explication :

- 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://loclahost:8080/secured , la page par défaut suivante s'affiche :

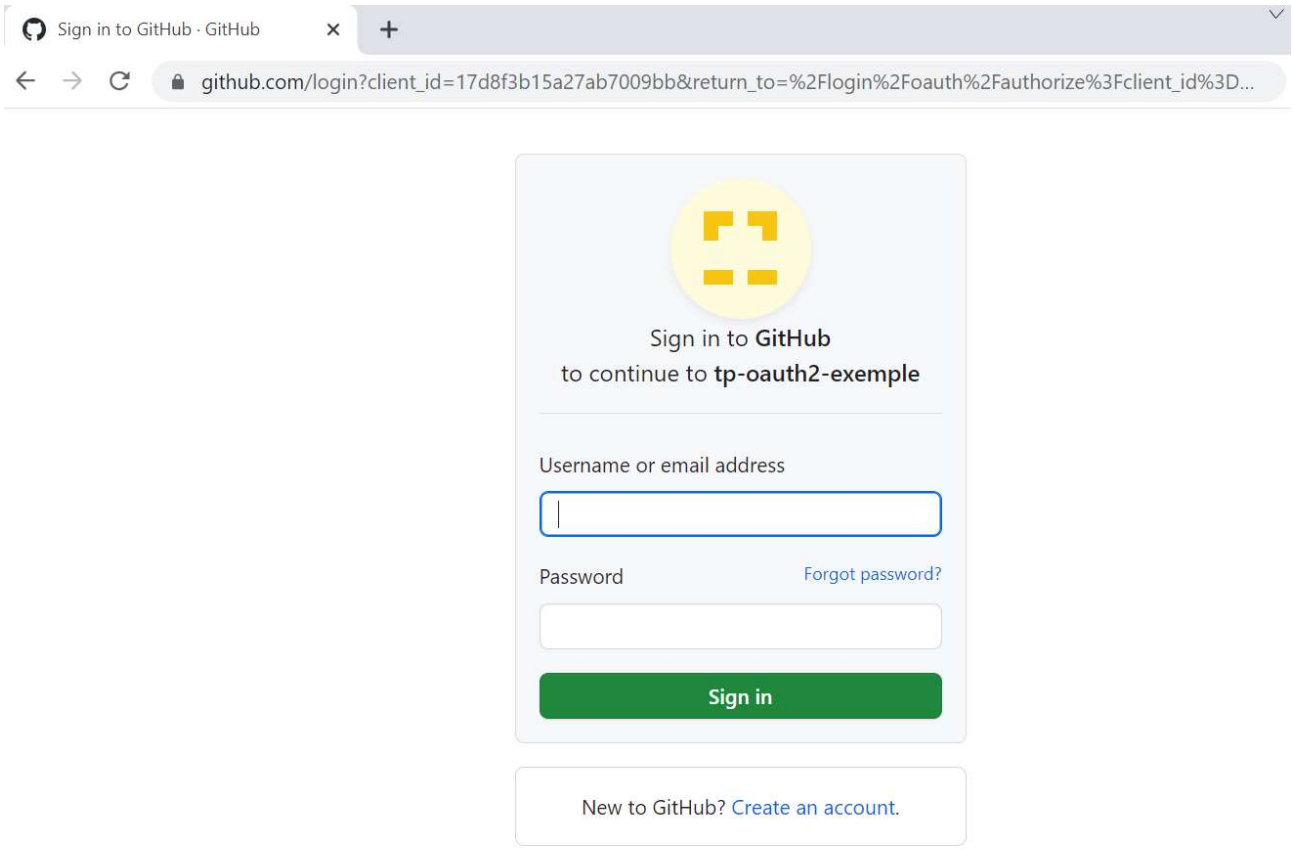


Explications :

- 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.