What is OAuth2?

Since OAuth was developed as an authentication protocol, the end effect of each OAuth flow is that the app obtains an access token in order to enter or change a user's account. The access token itself contains no information about the recipient.

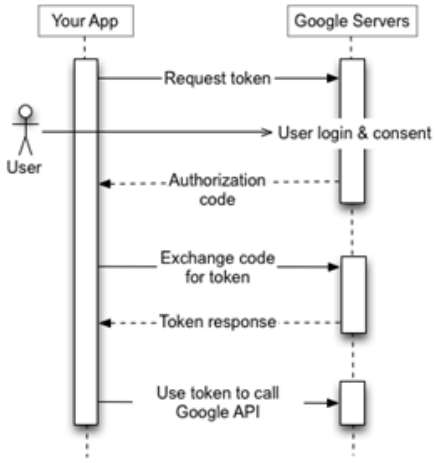
This article discusses how to integrate single sign-on features with an existing Spring Boot web application using the OAuth2 Client library – enabling end users to connect using their own Google accounts rather than application-managed credentials.

The vast majority of applications and websites have relied on simple username and password authentication. This causes complications when you try to grant an app access to data or features stored in another service that needs account authentication.

OAuth2 addresses these issues by encouraging the app's end user to communicate directly with the app's identity provider service and generate a cryptographically signed token granting access to designated scopes. This token is sent to the client, which uses it to authenticate requests made on the user's behalf.

Google OAuth API Authentication Flows

If access to and storage of confidential user information or features is needed, server-side authentication should be used. Server-side authentication, also known as the OAuth authorization code flow, is intended for applications that operate on web servers. For the majority of use cases, Google recommends this approach because it is the most convenient way to enforce OAuth.



Clients Setup

Maven Dependencies

The first step is to use the spring-boot-starter-oauth2-client starter in pom.xml file.

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

Configuring the Spring Boot application

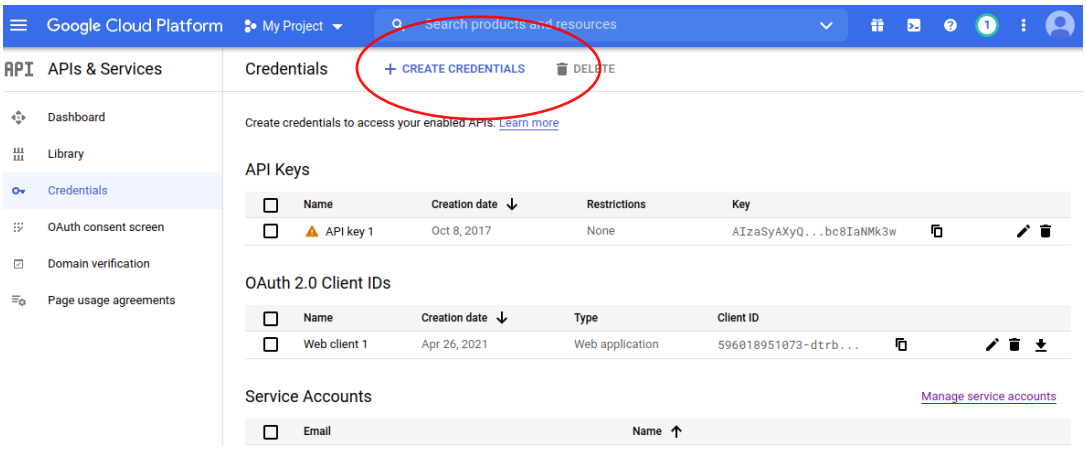
Add new JAVA class call “controller” and this is a REST controller.

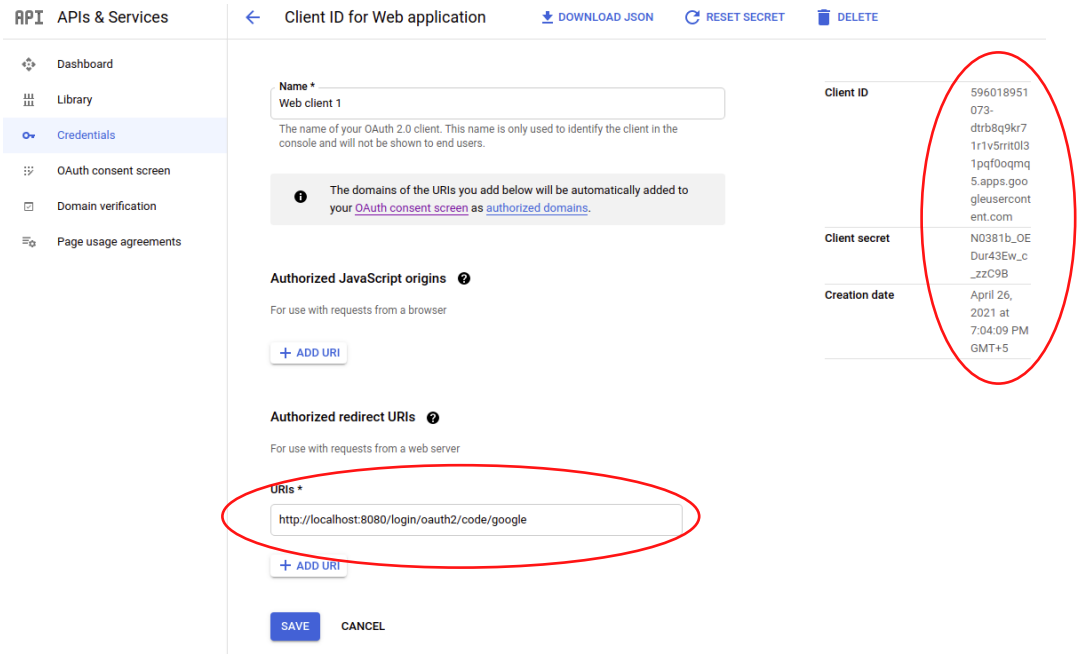
```
package SLIIT.Software.Security;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
@RestController
public class Controller {
   @GetMapping("/")
   public String helloSLIIT() {
      return "Hello SLIIT";
   }
   @GetMapping("/restricted")
   public String restricted() {
      return "Please log in";
   }
}
```

Authentication Setup

To access Google OAuth2 client credentials, navigate to the Google API Console's "Credentials" portion.

Then, in the Google Console, setup an allowed redirect URI, which is the direction to which users would be routed after successfully logging into Google.





It generates credentials of form "OAuth2 Client ID" for our web application in this section. As a part of this, Google creates a client id and secret for us.

Security Configuration

Add security configuration class file.

```
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {

   @Override
   public void configure(HttpSecurity httpSecurity) throws Exception {

      httpSecurity
            .antMatcher("/**").authorizeRequests()
            .antMatchers("/").permitAll()
            .anyRequest().authenticated()
            .and()
            .oauth2Login();
   }
```

Need to update the **application.properties** file with the client credentials. Spring Security properties are prefixed by "spring.security.oauth2.client.registration," followed by the client's name and the property's name:

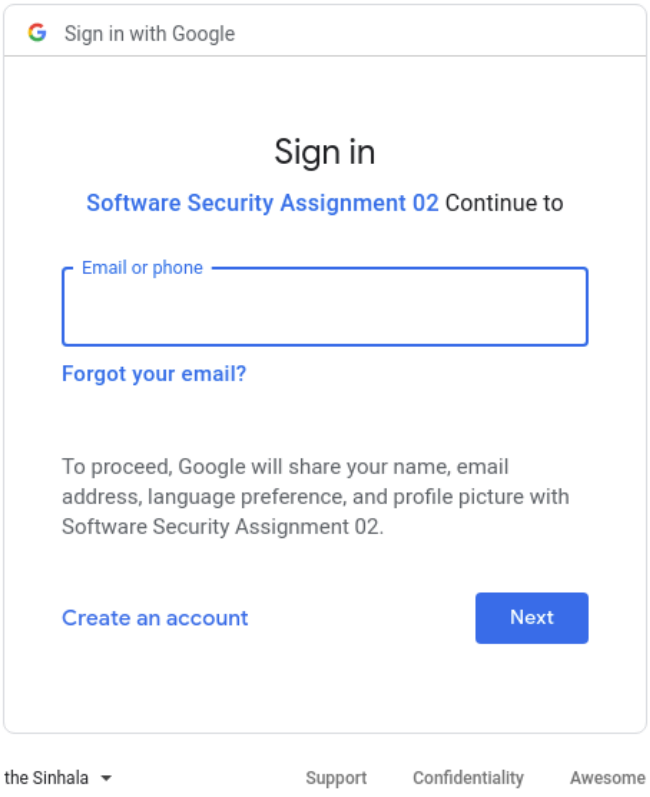
```
spring.security.oauth2.client.registration.google.client-id=<Google client ID>

spring.security.oauth2.client.registration.google.client-secret=<Client secret>
```

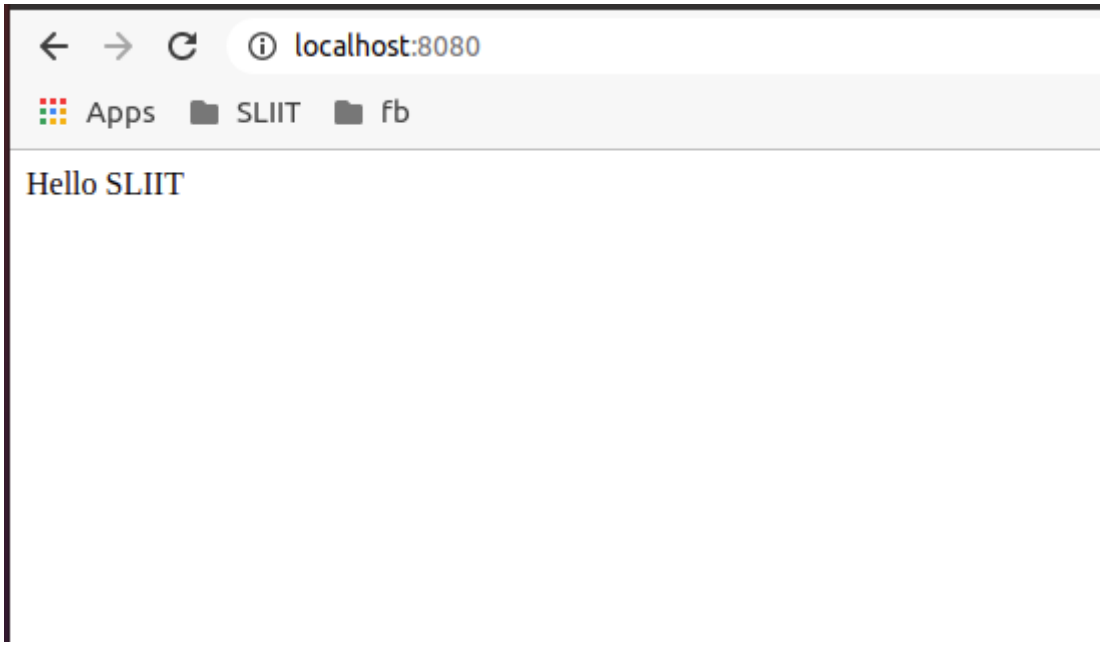
Now, when try to access a protected URL, the application will display an auto-generated login page with the client.

Login with OAuth 2.0

[invalid_request]

Google





Appendix

Pom.xml

```
<?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>2.4.5</version>
		<relativePath/> <!-- lookup parent from repository -->
	</parent>
	<groupId>SLIIT</groupId>
	<artifactId>Software-Security</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<name>Software-Security</name>
	<description>Assignments</description>
	<properties>
```

```
		<java.version>16</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-security</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-web</artifactId>
		</dependency>

		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-test</artifactId>
			<scope>test</scope>
		</dependency>
		<dependency>
			<groupId>org.springframework.security</groupId>
			<artifactId>spring-security-test</artifactId>
			<scope>test</scope>
		</dependency>
	</dependencies>

	<build>
		<plugins>
			<plugin>
				<groupId>org.springframework.boot</groupId>
				<artifactId>spring-boot-maven-plugin</artifactId>
			</plugin>
		</plugins>
	</build>

</project>
```

Controller.java

```
package SLIIT.Software.Security;

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

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

@RestController
public class Controller {
   @GetMapping("/")
   public String hello() {
      return "Hello SLIIT";
   }

   @GetMapping("/restricted")
   public String restricted() {
      return "Please log in";
   }

}
```

SecurityConfig.java

```
package SLIIT.Software.Security;


import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {

   @Override
   public void configure(HttpSecurity httpSecurity) throws Exception {

      httpSecurity
            .antMatcher("/**").authorizeRequests()
            .antMatchers("/").permitAll()
            .anyRequest().authenticated()
            .and()
            .oauth2Login();
   }



}
```

SoftwareSecurityApplication.java

```
package SLIIT.Software.Security;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SoftwareSecurityApplication {

	public static void main(String[] args) {
		SpringApplication.run(SoftwareSecurityApplication.class, args);
	}

}
```

Application.properties

```
spring.security.oauth2.client.registration.google.client-id=596018951073-
dtrb8q9kr71r1v5rrit0l31pqf0oqmq5.apps.googleusercontent.com
spring.security.oauth2.client.registration.google.client-secret=N0381b_OEDur43Ew_c_zzC9B
```