

¿Qué ofrece Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

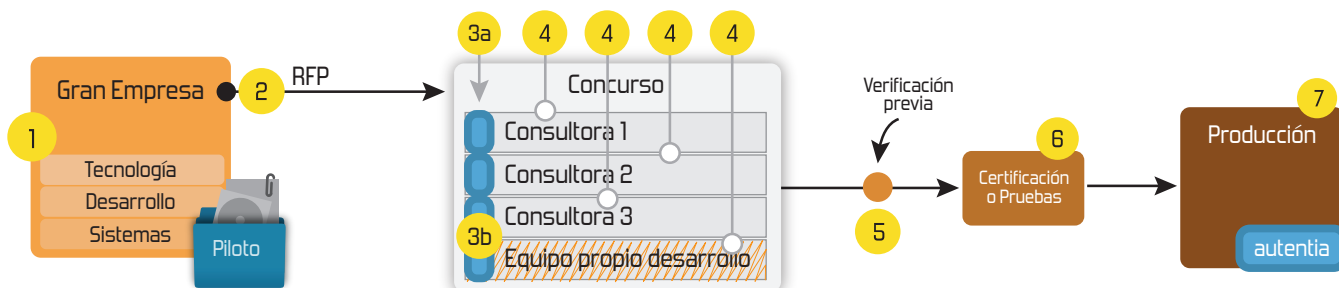
1. Desarrollo de componentes y proyectos a medida

2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías



1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces,
HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)
Gestor de contenidos (Alfresco)
Aplicaciones híbridas

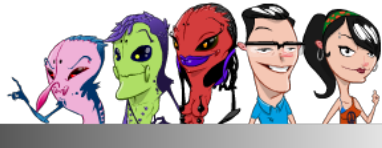
Tareas programadas (Quartz)
Gestor documental (Alfresco)
Inversión de control (Spring)

Control de autenticación y
acceso (Spring Security)
UDDI
Web Services
Rest Services
Social SSO
SSO (Cas)

JPA-Hibernate, MyBatis
Motor de búsqueda empresarial (Solr)
ETL (Talend)

Dirección de Proyectos Informáticos.
Metodologías ágiles
Patrones de diseño
TDD

BPM (jBPM o Bonita)
Generación de informes (JasperReport)
ESB (Open ESB)



David Gómez García

Titulado en Ingeniería Técnica en Informática por la Universidad Politécnica de Madrid en 1998, David ha participado en proyectos para telecomunicaciones, comercio electrónico, banca, defensa y sistemas de transporte terrestre. Ha liderado proyectos de aplicaciones web, infraestructura SOA, aplicaciones java cliente y pasarelas de comunicación entre sistemas.

Twitter: [Seguir a @dgomezg](#) 453 seguidores

[Ver todos los tutoriales del autor](#)

Catálogo de servicios Autentia



Síguenos a través de:



Últimas Noticias

» [Comentando el libro: Agile Management de Angel Medina](#)

» [Final de temporada de Terrakas Autentia freaklances](#)

» [Atención, APLAZADO Estreno último capítulo de Terrakas](#)

» [Vendedor: Soy inseguro, filtra o elige por mí: si quieres que te compre.](#)

» [Comentando el libro: El arte de pensar, de Rolf Dobelli](#)

[Histórico de noticias](#)

Últimos Tutoriales

» [Hola Mundo con Spring 3 MVC](#)

» [Ejecución de tests unitarios con junit en proyectos ant y su integración en jenkins y sonar para medir la cobertura.](#)

» [Comentando la AOS 2013](#)

» [Soporte en Sonar para un proyecto multi-lenguaje: configuración desde Jenkins](#)

» [Integración de Selenium Grid con Jenkins](#)

Fecha de publicación del tutorial: 2013-07-08

Integrar el login de Google en tu App con OAuth2 y Spring Security

En este tutorial, vamos a ver cómo montar en tres sencillos pasos la seguridad de nuestra aplicación con Spring Security, pero delegando la autenticación en un proveedor externo de OAuth2 como Google (o también podría ser Facebook, twitter, LinkedIn u otros).

¿Qué significa esto?, básicamente que delegaremos el control de usuario y password (la autenticación) en un sistema ajeno, de forma que no tenemos que preocuparnos de la gestión de usuarios (darlo de alta, gestionar sus contraseñas, sus recordatorios, etc..) ya que ese mantenimiento ya lo lleva a cabo el proveedor de OAuth contra el que conectamos.

En nuestra aplicación, por tanto, sólo nos preocuparemos de delimitar las restricciones de seguridad (la autorización); esto es, a quién o a qué tipo de usuarios dejamos acceder a qué partes.

Empezaremos por crear una aplicación web que tendrá una parte pública (accesible a cualquier usuario) y una parte privada (sólo para usuarios registrados). Configuraremos Spring Security para especificar las restricciones de seguridad de la parte privada. Por último configuraremos el sistema de autenticación de Spring Security, utilizando una librería con implementación de proveedores específicos de SpringSecurity para OAuth, para integrar el proceso de identificación del usuario (autenticación).

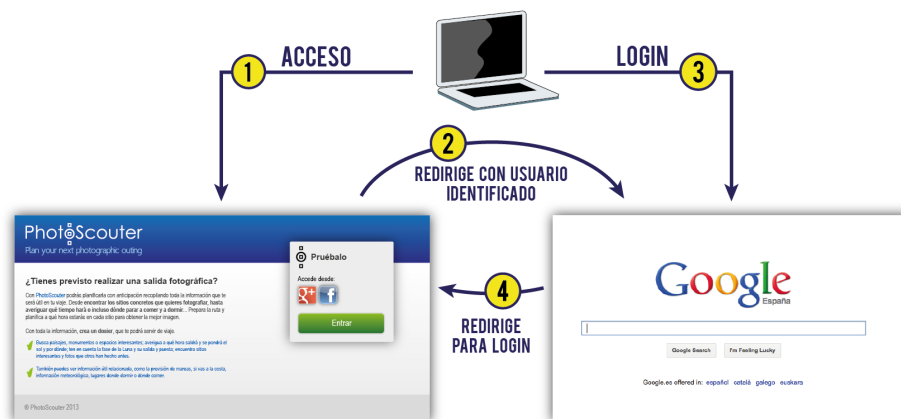
Iremos viendo estos pasos en los siguientes apartados:

- Estructura básica de la aplicación
- Obtener las API Keys para poder utilizar el Login de Google.
- Configurando Spring Security en nuestra aplicación.
- Configuración de Google como proveedor OAuth de autenticación
- La prueba de la aplicación

Una vez que tengamos configurada la autenticación utilizando un proveedor de OAuth, evolucionaremos nuestro sistema, incluyendo soporte a varios proveedores.

Estructura básica de la aplicación.

Nuestra aplicación de ejemplo constará de una "Landing Page", que consideraremos pública y a la que podrá acceder cualquier usuario. En la landing page, daremos la opción de que el usuario acceda a la parte privada, donde se mostrará su nombre y sus credenciales asignadas dentro de la aplicación. Esta página con la descripción del perfil será la parte que consideraremos privada y, por tanto, sólo de acceso para usuarios registrados.



El esquema general del proceso de login con OAuth

Vamos a utilizar Spring MVC para la aplicación web. Así que comenzamos por definir en el Web.xml el DispatcherServlet de Spring:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns="http://java.sun.com/xml/ns/javaee"
3           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4           xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
```

```

5      http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
6      version="2.5">
7
8      <servlet>
9          <servlet-name>springWeb</servlet-name>
10         <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
11     </servlet>
12
13     <servlet-mapping>
14         <servlet-name>springWeb</servlet-name>
15         <url-pattern>/*</url-pattern>
16     </servlet-mapping>
17 </web-app>

```

Una vez configurado el DispatcherServlet de Spring, configuramos los componentes de infraestructura de Spring MVC en el fichero {servlet-name}-servlet.xml. En este caso, el fichero se llamará springWeb-servlet.xml

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xmlns:context="http://www.springframework.org/schema/context"
5         xmlns:mvc="http://www.springframework.org/schema/mvc"
6         xsi:schemaLocation="http://www.springframework.org/schema/beans
7                             http://www.springframework.org/schema/beans/spring-beans.xsd
8                             http://www.springframework.org/schema/context
9                             http://www.springframework.org/schema/context/spring-context.xsd
10                            http://www.springframework.org/schema/mvc
11                            http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd">
12
13     <mvc:annotation-driven/>
14
15     <!-- Rutas por defecto para los archivos estaticos -->
16     <mvc:resources mapping="/js/*" location="/js/" />
17     <mvc:resources mapping="/img/*" location="/img/" />
18     <mvc:resources mapping="*.html" location="/" />
19     <mvc:resources mapping="/css/*" location="/css/" />
20
21     <bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceV:
22         <property name="prefix" value="/WEB-INF/jsp/" />
23         <property name="suffix" value=".jsp" />
24     </bean>
25
26 </beans>

```

Con la estructura de Spring y de la aplicación web creadas, utilizaremos como página pública una versión ligeramente modificada de uno de los ejemplos de Twitter Bootstrap, en la que incluimos un botón con un enlace a la página privada:

```

1  <a class="btn btn-large btn-primary right" href="/userHome">Entra</a>

```

Ahora definiremos un controlador de Spring MVC que simplemente pondrá información del perfil del usuario disponible en la vista:

```

1  package com.dgomezg.playground.spring.oauth.controller;
2
3  import org.springframework.security.access.annotation.Secured;
4  import org.springframework.security.core.context.SecurityContextHolder;
5  import org.springframework.stereotype.Controller;
6  import org.springframework.web.bind.annotation.RequestMapping;
7  import org.springframework.web.bind.annotation.RequestMethod;
8  import org.springframework.web.bind.annotation.ResponseBody;
9
10 @Controller
11 public class HomeController {
12
13     @RequestMapping(value="/userHome", method = RequestMethod.GET)
14     public void getUserProfile(Model model) {
15         model.addAttribute("user", SecurityContextHolder.getContext().getAuthentication())
16     }
17
18 }

```

Por último, para acabar la estructura básica de la aplicación, preparamos la página privada donde mostraremos información del usuario registrado (en este caso será solo el perfil del usuario, para mantener el ejemplo sencillo). Esta página la crearemos en /WEB-INF/jsp/userHome.jsp

```

1  <html>
2
3      <body>
4
5          <div>Usuario Registrado: <%=user%></div>
6
7      </body>
8  </html>

```

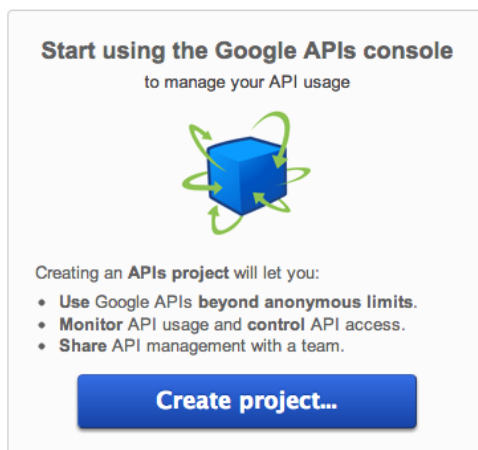
Con esta estructura, tendremos ya la aplicación base sobre la que añadir la seguridad con el proceso de Login delegado en proveedores externos. Es normal que, en este punto, la aplicación dé un error al pulsar en el botón de "Entrar" de la parte pública. Eso se debe a que no hemos configurado todavía los procesos de seguridad.

Vamos ahora a ello. Vamos a empezar con integrar la validación de usuario con Google.

Obtener las API Keys para poder utilizar el Login de Google.

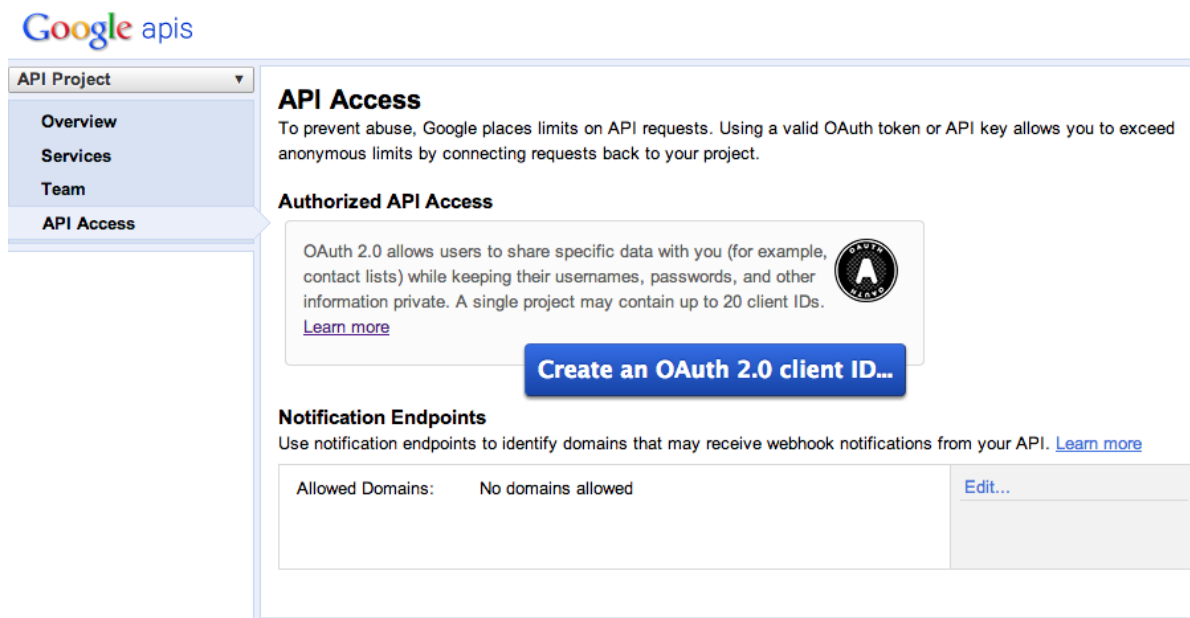
Para invocar el servicio de Login de Google, necesitamos obtener un código de uso (un API Key) para nuestra aplicación. Para ello, accederemos a la [consola de APIs de Google](#).

Si no lo hemos hecho antes, el primer paso que necesitamos es crear un proyecto



Consola en Google APIs para dar de alta nuestro proyecto

Pulsamos en el botón "Create project..." y en la siguiente pantalla podemos seleccionar los servicios que vamos a querer utilizar de Google. En este caso, como sólo vamos a utilizar el Login, no es necesario seleccionar ningún servicio adicional; pasamos a obtener el API Key. Para ello, seleccionamos en el menú de la izquierda, la opción API Access:



Creación de un cliente para el acceso al API de OAuth de Google

Pulsamos en el botón "Create an OAUTH 2.0 client ID..." e introducimos los datos de nuestra aplicación

Create Client ID

Branding Information

The following information will be shown to users whenever you request access to their private data using your new client ID.

Product name:

Product X

Google account:

dgomezg@autentia.com - you

Link your project to this account's profile and reputation.

Product logo:

http://example.com/example_logo.png

Update

Max size: 120x60 pixels

Home Page URL:

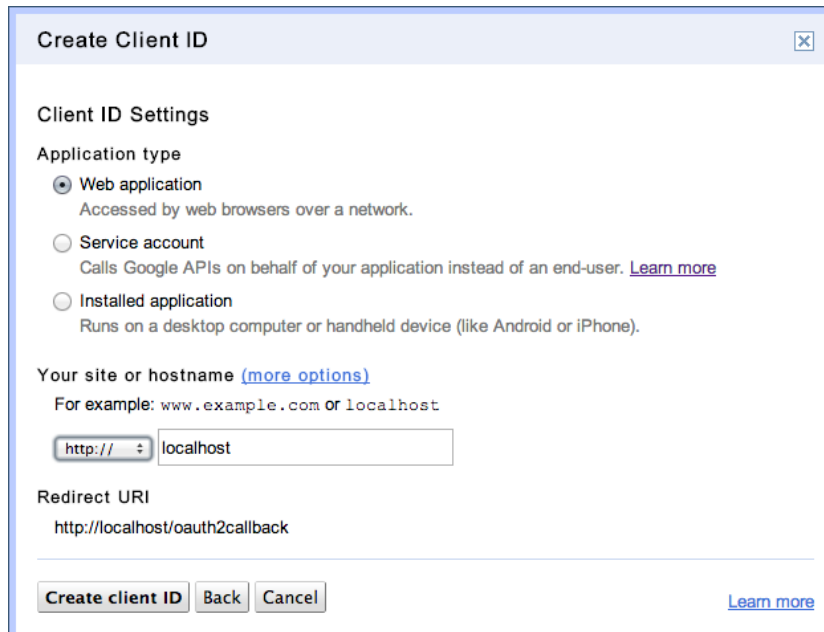
Next

Cancel

Learn more

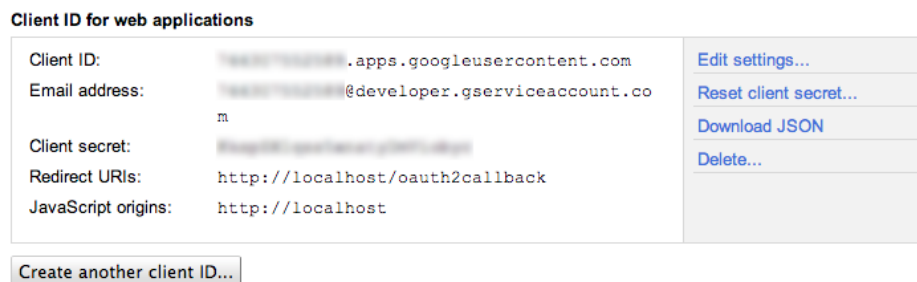
Detalles del cliente para el acceso al API de OAuth de Google

En el siguiente apartado, especificamos el tipo de aplicación, en nuestro caso, una aplicación web, e indicamos, para nuestro tutorial, que la URL es http://localhost



Tipo del cliente para el acceso al API de OAuth de Google

Pulsamos en el botón "Create client ID" y nos llevará al panel de control donde podremos ver el API Key generado que tendremos que configurar más adelante en nuestra aplicación. En concreto, necesitaremos los valores del "Client ID" y el "Client Secret".



Datos a usar en el cliente para el acceso al API de OAuth de Google

Desde esta consola de administración podremos crear nuevos clientes de autorización ("Create another client ID...") o modificar, por ejemplo la URL a la que Google deberá devolver la llamada después de haber realizado la autenticación ("Edit settings..."). Volveremos sobre esto un poco más tarde.

Configurando Spring Security en nuestra aplicación.

Spring Security es un módulo del framework de Spring que nos permite configurar de forma muy sencilla la gestión de seguridad en nuestras aplicaciones web.

Se basa en un Filtro, o más bien en una cadena de filtros (SpringSecurityFilterChain), que se encargará de coordinar las distintas tareas que participan en el control de la seguridad de una aplicación Web.

Para añadir seguridad con Spring Security en nuestra aplicación web, comenzaremos por configurar dicho filtro de seguridad en nuestro descriptor web.xml:

```
1 <filter>
2   <filter-name>springSecurityFilterChain</filter-name>
3   <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
4 </filter>
5 <filter-mapping>
6   <filter-name>springSecurityFilterChain</filter-name>
7   <url-pattern>/*</url-pattern>
8 </filter-mapping>
```

Con esta configuración, se crea un servletFilter (DelegatingFilterProxy) que buscará en el contexto de configuración de aplicación de Spring, un bean con id "springSecurityFilterChain" en quien delegará las comprobaciones de seguridad.

Esta configuración de spring la incorporaremos a un fichero de contexto específico, que nombraremos como spring-security.xml:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xmlns:security="http://www.springframework.org/schema/security"
5       xsi:schemaLocation="http://www.springframework.org/schema/beans
6                           http://www.springframework.org/schema/beans/spring-beans.xsd
7                           http://www.springframework.org/schema/security
8                           http://www.springframework.org/schema/security/spring-security-3.1.xsd">
9
10    <security:http>
11      <security:intercept-url pattern="/userHome" access="IS_AUTHENTICATED_FULLY" />
12      <security:intercept-url pattern="/*" access="IS_AUTHENTICATED_ANONYMOUSLY" />
13      <security:logout />
14    </security:http>
15  </beans>
```

De momento hemos configurado, a través de la etiqueta `security:http`, que utilizaremos seguridad a nivel de peticiones http. Al configurar esta etiqueta, Spring crea y configura el bean `springSecurityFilterChain` y todos los filtros que forman parte de dicha cadena de filtros de seguridad.

También hemos añadido dos patrones de URL con las restricciones de seguridad, de forma que la ruta `/userHome` solo será accesible para usuarios que estén autenticados y el resto para usuarios anónimos (es decir, aquellos que no se hayan identificado todavía).

Por último, debemos registrar un Listener que permita cargar y configurar este fichero de seguridad, de forma que el `DelegatingFilterProxy` funcione como esperamos. Para ello, completamos la definición de nuestro descriptor `web.xml`

```

1 <context-param>
2   <param-name>contextConfigLocation</param-name>
3   <param-value>classpath:/spring-security.xml</param-value>
4 </context-param>
5
6 <listener>
7   <description>Spring context loader listener</description>
8   <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
9 </listener>
```

Con esta configuración, hemos configurado un `ContextLoaderListener` que se encargará de crear un contexto de Spring con la configuración de `spring-security.xml`. Este contexto de Spring quedará asociado al contenedor de Servlets y, por tanto, accesible para el `DispatcherServlet` o el `DelegatingFilterProxy` definido en el mismo `web.xml`

Con este apartado, ya tenemos configurada la gestión básica de seguridad, pero aún no hemos especificado cuál es el mecanismo de Login que queremos utilizar.

Configuración de Google como proveedor OAuth de autenticación

Para añadir la llamada a Google para realizar la autenticación, utilizaremos la librería `spring-security-pac4j` en la que ya se definen algunos componentes que, implementados como Proveedores de autenticación sobre Spring-Security, gestionan el detalle de la llamada a cada varios Proveedores de autenticación a través de OAuth.

Para este tutorial, nosotros nos limitaremos a utilizar el `GoogleProvider`, pero existen varias implementaciones adicionales, que nos permitirán configurar la autenticación de usuarios con Facebook, LinkedIn, twitter, y varios otros.

Para utilizar los componentes de esta librería, añadiremos las coordenadas maven, a nuestra gestión de dependencias

```

1 <dependency>
2   <groupId>com.github.leleuj.springframework.security</groupId>
3   <artifactId>spring-security-oauth-client</artifactId>
4   <version>1.1.0</version>
5 </dependency>
```

Una vez añadidas las dependencias, podemos complementar la configuración de `spring-security.xml` para hacer uso del proveedor de login

```

<authentication Providers definition -->
<le2Provider" class="org.scribe.up.provider.impl.Google2Provider">
  <name="key" value="{pon aqui el Client ID creado en la consola de Google API (en el punto 1)}"/>
  <name="secret" value="{secret Key de la consola de Google API}"/>
  <name="scope" value="EMAIL_AND_PROFILE"/>

  <providersDefinition" class="org.scribe.up.provider.ProvidersDefinition">
    <name="baseUrl" value="http://localhost:8080/j_spring_oauth_security_check" />
    <name="providers">

    <ef bean="google2Provider" />
    >
    >

    <nProvider" class="com.github.leleuj.ss.oauth.client.authentication.OAuthAuthenticationProvider">
      <name="providersDefinition" ref="providersDefinition" />

    <gle2EntryPoint" class="com.github.leleuj.ss.oauth.client.web.OAuthAuthenticationEntryPoint">
      <name="provider" ref="google2Provider" />

    <nentication filter -->
    <nFilter" class="com.github.leleuj.ss.oauth.client.web.OAuthAuthenticationFilter">
      <name="providersDefinition" ref="providersDefinition" />
      <name="authenticationManager" ref="authenticationManager" />
```

Por último, configuramos, también en `spring-security.xml`, para que el proveedor de autenticación de Spring utilice este proveedor que conecta con Google, y configuramos cual es el entry-point por defecto para la seguridad http:

```

1 <security:authentication-manager alias="authenticationManager">
2   <security:authentication-provider ref="oAuthProvider"/>
3 </security:authentication-manager>
4
5 <security:http entry-point-ref="google2EntryPoint">
6   <security:custom-filter after="CAS_FILTER" ref="oAuthFilter" />
7   <security:intercept-url pattern="/userHome" access="IS_AUTHENTICATED_FULLY" />
8   <security:intercept-url pattern="/**" access="IS_AUTHENTICATED_ANONYMOUSLY" />
9   <security:logout />
10 </security:http>
```

Por último, será necesario que volvamos a la [consola de APIs de Google](#) y modifiquemos, para nuestra aplicación, la URL a la que debe volver Google una vez que la autenticación haya sido realizada con éxito. Para ello, sobre el CLient ID que creamos en el segundo paso de este tutorial, pulsamos sobre el enlace "Edit Settings..." y añadimos `http://localhost:8080/j_spring_oauth_security_check` como URL de callback autorizada para el login

Client ID for web applications

Client ID:

1028100123456789.apps.googleusercontent.com

Email address:

1028100123456789@developer.gserviceaccount.com

Client secret:

1028100123456789-1028100123456789-1028100123456789

Redirect URIs:

http://localhost/oauth2callback

JavaScript origins:

http://localhost

Edit settings...

Reset client secret...

Download JSON

Delete...

Create new application

Notifications

Use notifications

Allow access to your data

Client ID Settings

Authorized Redirect URIs

One per line. For example: https://example.com/path/to/callback

http://localhost/oauth2callback
http://localhost:8080/j_spring_oauth_security_check

Authorized JavaScript Origins

One per line. For example: https://example.com

http://localhost

Update

Cancel

[Learn more](#)

Feedback

Send feedback

La prueba de la aplicación

Una vez desplegada la aplicación, sin estar identificados aún en el sistema, accedemos con el navegador a la dirección de la página pública

Heading

[View details »](#)[View details »](#)[View details »](#)

Página de login para nuestra aplicación



Login con Google

Observamos que la URL ha cambiado y estamos identificándonos en el servidor de Google. Esta es una de las ventajas de OAuth, nuestra aplicación nunca tiene que gestionar el proceso de validación, siendo el proveedor de autenticación quién se encarga de ello y nos dará un token de usuario en caso de que dicha validación sea correcta.

Al introducir nuestro usuario y password correctos en Google, se nos redirigirá de nuevo a la parte protegida de nuestra aplicación que, en este caso, simplemente muestra el token de seguridad devuelto por Google (tal y como lo habíamos configurado en nuestro controlador)



Página privada en nuestra aplicación

A partir de aquí, sólo tendríamos que ir añadiendo funcionalidad a nuestra aplicación, añadiendo nuevos controladores a Spring y especificando sus restricciones de seguridad

Cuando hemos configurado la autenticación delegando el registro en proveedores externos, es habitual, permitir a los usuarios de nuestra aplicación que puedan utilizar distintas cuentas personales (facebook, twitter, LinkedIn, Google u otras). Para ello, una vez que hemos llegado a este punto, podríamos modificar ligeramente el proceso de login para permitir conectar con distintos proveedores de OAuth. Pero dejaremos esta modificación para un segundo tutorial que publicaré próximamente.

Por último, he dejado disponible el código de este ejemplo en mi repositorio GitHub con [ejemplos de Spring](#)



Esta obra está licenciada bajo [licencia Creative Commons de Reconocimiento-No comercial-Sin obras derivadas 2.5](#)